Professional Edition &
Dotfuscator for Marketplace Apps

User's Guide

バージョン 4

© 2013 PreEmptive Solutions, LLC
All rights reserved.

ドキュメント バージョン 4.11
www.preemptive.com

商標

Dotfuscator、DashO、Overload-Induction、Runtime Intelligence、PreEmptive Analytics、PreEmptive Analytics for TFS、PreEmptive Solutions ロゴ、Dotfuscator ロゴ、および DashO ロゴは PreEmptive Solutions, LLC の商標です。

.NET™、MSIL™、Team Foundation Server™、および Visual Studio™ は Microsoft, Inc. の商標です。

Java は Oracle, Inc. の商標です。

その他すべての商標は、各社に帰属しています。

本書は "現状有姿" で提供されています。本ソフトウェアについて、商品性、特定目的への適合性、および非侵害保証を含む一切の明示または黙示の保証をいたしません。

本書には、誤植が含まれていたり、技術的誤りがある可能性があります。本ドキュメントおよびサポートしているソフトウェアは、予告なしに更新および修正することがあります。

PreEmptive Solutions, LLC は、本製品に組み込まれているテクノロジに関する知的財産権を有します。これらの知的財産所有権には、米国の特許ならびに米国およびその他の国において出願中の特許（これに限定されるものではありません）が含まれている可能性があります。

本製品は、その使用、コピー、配布、および逆コンパイルを制限するライセンスのもとに配布されます。PreEmptive Solutions, LLC からの書面による事前の許諾を受けることなく、その目的を問わず、どのような形態であっても複製することは禁じられています。

1 目次

1. 目次 3-12
2. Dotfuscator 13
2.1. はじめに 13-14
2.1.1. Dotfuscator エディション 14-15
2.2. 概要 15
2.2.1. Dotfuscator の登録とアクティブ化 15-17
2.2.2. スタンドアロン GUI クイック スタート 17-19
2.2.3. コマンド ライン クイック スタート 19-21
2.2.4. Visual Studio 統合 UI クイック スタート 21-25
2.2.5. 難読化後の出力の表示と検証 25-28
2.3. Dotfuscator による難読化の理解 28-29
2.3.1. 保護概念 29
2.3.1.1. 名前の変更 29-30
2.3.1.2. 制御フロー 30-31
2.3.1.3. 文字列の暗号化 31
2.3.1.4. 不要コードの除去 32
2.3.1.5. 改ざんの検出および防御 32
2.3.1.6. Shelf Life 32-33
2.3.1.7. ウォーターマーク 33
2.3.2. 難読化されたアプリケーションのビルドとデバッグ 33
2.3.2.1. アセンブリのリンク 33-34
2.3.2.2. 増分難読化 34
2.3.2.3. 難読化されたコードのデバッグ 34-35
2.3.3. 高度なトピック 35
2.3.3.1. スマート難読化 35-36
2.3.3.2. P/Invoke メソッド 36-37
2.3.3.3. Managed C++ および IJW（It Just Works）のサンク 37
2.3.3.4. マネージ リソースを持つアセンブリを Dotfuscator で処理する 37
2.3.3.5. サテライト DLL を使用するアセンブリを Dotfuscator で処理する 37
2.3.3.6. マルチ モジュール アセンブリを Dotfuscator で処理する 37-38

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.3.3.7. 厳密名のアセンブリを Dotfuscator で処理する 38-39
2.3.3.8. Authenticode 署名のアセンブリ 39-40
2.3.3.9. 64 ビット アセンブリを Dotfuscator で処理する 40
2.3.3.10. リフレクションおよび動的クラス読み込み 40-41
2.3.3.11. カスタム属性を使用した宣言による難読化 41-43
2.3.3.12. ビルド イベント 43-44
2.3.3.13. フレンド アセンブリ 44
2.3.3.14. 外部ツールの検索 44-45
2.4. Dotfuscator によるインストルメンテーションの理解 45-46
2.4.1. インストルメンテーションの差し込み 46-48
2.4.2. 例外の追跡 48
2.4.3. 改ざん通知 48-49
2.5. GUI による Dotfuscator 構成 49
2.5.1. スタンドアロン GUI 49-51
2.5.1.1. プロジェクトの作成 51-53
2.5.1.2. 入力の設定 53-55
2.5.1.2.1. ディレクトリ入力 55-56
2.5.1.2.2. Silverlight 入力 56-57
2.5.1.2.3. ClickOnce 入力 57
2.5.1.2.4. Windows ストア入力 57-58
2.5.1.3. ［設定］タブ 58-68
2.5.1.4. 各機能の設定の構成 68-69
2.5.1.5. プロジェクトのビルド 69
2.5.1.5.1. ［出力］タブ 69-70
2.5.1.6. プロジェクト ファイルとレポートの表示 70-71
2.5.1.7. ユーザー設定の設定 71-72
2.5.1.8. ［ヘルプ］メニュー 72
2.5.2. Visual Studio インターフェイス 72-73
2.5.2.1. Dotfuscator プロジェクトの作成 73
2.5.2.2. ソリューション エクスプローラーおよび Dotfuscator プロジェクト ツリー 73-74
2.5.2.3. プロジェクトの構成 74-75
2.5.2.4. Dotfuscator プロジェクトの配置 75

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.2.5. 入力の設定 75-76
2.5.2.6. プロジェクト プロパティ 77-86
2.5.2.7. 入力アセンブリ プロパティ 87-88
2.5.2.8. 入力パッケージ プロパティ 88-89
2.5.2.9. 各機能の設定の構成 89
2.5.2.10. プロジェクトのビルド 89
2.5.2.11. [表示]メニュー 89-90
2.5.2.12. [ヘルプ]メニュー 90-91
2.5.2.13. ユーザー設定の設定 91
2.5.3. [名前の変更]エディター 91
2.5.3.1. [名前の変更]エディターの[オプション]タブ 91-92
2.5.3.1.1. 拡張オーバーロード誘導によるメソッドの名前の変更 92-93
2.5.3.1.2. クラスの名前の変更オプション 93-95
2.5.3.1.3. XML シリアル化と名前の変更 95-96
2.5.3.1.4. 名前の変更時、明示的なメソッドのオーバーライドを導入する 96
2.5.3.2. [名前の変更]エディターの[除外]タブ 96-97
2.5.3.2.1. 名前の変更の対象除外 97-98
2.5.3.3. [名前の変更]エディターの[組み込み規則]タブ 98
2.5.4. [制御フロー]エディター 98
2.5.4.1. [制御フロー]エディターの[オプション]タブ 98-99
2.5.4.2. [制御フロー]エディターの[除外]タブ 99-100
2.5.5. [文字列の暗号化]エディター 100
2.5.5.1. [文字列の暗号化]エディターの[対象]タブ 100-101
2.5.6. [除去]エディター 101
2.5.6.1. 対象トリガーと条件付き対象の理解 101-102
2.5.6.2. [対象トリガー]タブ 102-103
2.5.6.3. [条件付き対象]タブ 103-104
2.5.6.4. [組み込み規則]タブ 104-105
2.5.6.5. [オプション]タブ 105
2.5.6.5.1. 定数のみ除去 105-106
2.5.6.6. 除去レポート 106
2.5.7. [リンク]エディター 106

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.7.1. ［入力アセンブリ］および［リンク アセンブリ］ 106-107
2.5.7.1.1. プライマリ アセンブリ 107
2.5.7.1.2. 名前の改変 107-108
2.5.7.2. エントリ ポイントの設定 108-109
2.5.7.2.1. エントリ ポイント 109-110
2.5.8. ［PreMark］エディター 110-111
2.5.8.1. ウォーターマーク文字列の長さ 111
2.5.8.2. 文字コード 111-112
2.5.8.3. ウォーターマークの抽出 112
2.5.9. 規則の編集インターフェイス 112-113
2.5.9.1. 個々の要素の選択 113-117
2.5.9.2. カスタム規則の作成 117
2.5.9.2.1. 名前空間による選択 117-118
2.5.9.2.2. 型による選択 118-120
2.5.9.2.3. メソッドによる選択 120-121
2.5.9.2.4. フィールドによる選択 121-122
2.5.9.2.5. プロパティによる選択 122-124
2.5.9.2.6. イベントによる選択 124-125
2.5.9.2.7. カスタム属性による選択 125-126
2.5.9.2.8. スーパータイプによる選択 126-127
2.5.9.3. 規則の編集と削除 127
2.5.9.4. 規則での宣言による難読化の使用 127-128
2.5.9.5. 規則のプレビュー 128
2.5.10. インストルメンテーション（改ざん、Shelf Life、例外、分析） 128-131
2.5.10.1. Shelf Life トークンの概要 131-132
2.5.10.1.1. Shelf Life Activation Key の概要 132
2.5.10.1.2. 期限切れと警告の操作 132-134
2.5.10.1.3. 期限切れと警告のレポート 134-135
2.5.10.1.4. 新しい Shelf Life トークンの生成 135-136
2.5.10.2. 改ざん通知 136-137
2.5.10.2.1. 改ざんレポート 137
2.5.10.2.2. 改ざん操作 137-139

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.10.2.3. 改ざんのシミュレート 139
2.5.10.3. 例外の追跡 139
2.5.10.3.1. 例外レポート 139-140
2.5.10.3.2. 例外操作 140-141
2.5.10.3.3. ユーザー指定の例外レポート情報の収集 141-143
2.5.10.4. アプリケーション分析（Runtime Intelligence） 143-144
2.5.10.4.1. アプリケーション分析を備えた Dotfuscator の構成および実行 144-147
2.5.10.4.2. Runtime Intelligence のカスタム属性 147
2.5.10.4.2.1. アセンブリ レベルの属性 147-149
2.5.10.4.2.2. エントリ ポイント属性 149-152
2.5.10.4.2.3. Exit Point 属性 152
2.5.10.4.2.4. 改ざん通知属性 152-153
2.5.10.4.2.5. Shelf Life および Sign of Life 属性 153
2.5.10.4.2.6. 例外の追跡属性 153
2.5.10.4.2.7. 機能の利用状況属性 153-155
2.5.10.4.2.8. パフォーマンス属性 155
2.5.10.4.2.9. 環境属性 155-156
2.5.10.4.2.10. 拡張キーを含むユーザー定義データの送信 156-157
2.5.10.4.2.11. メソッド パラメーターを拡張キーとして自動的に送信する 157-158
2.5.10.4.3. アプリケーション分析を備えた Dotfuscator のテストとデバッグ 158
2.5.10.4.3.1. メッセージ追跡の構成 158-159
2.5.10.4.3.2. メッセージの監視 159
2.5.10.4.3.3. メッセージ データのダウンロード 159-160
2.5.10.4.4. Runtime Intelligence 対応アプリケーションの例 160-166
2.5.11. 難読化されたスタック トレースのデコード 166-168
2.5.12. Lucidator の使用 168-170
2.5.13. カスタマー フィードバックのオプション 170-171
2.6. リファレンス 171
2.6.1. コマンド ライン インターフェイス リファレンス 171-172
2.6.1.1. コマンド ライン オプションの概要 172-176
2.6.1.2. コマンド ラインでのオプション指定による構成ファイルの補完またはオーバーライド 176-179
2.6.1.3. コマンド ラインからの構成ファイルの保存 179-180

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.1.4. コマンド ラインからのグラフィカル ユーザー インターフェイスの起動 180-181
2.6.2. MSBuild タスク リファレンス 181
2.6.2.1. Dotfuscate タスク 181-183
2.6.2.2. PreMark タスク 183
2.6.3. 構成ファイル リファレンス 183
2.6.3.1. バージョン 183-184
2.6.3.2. プロパティ リストとプロパティ 184-185
2.6.3.3. グローバル セクション 185
2.6.3.3.1. Library グローバル オプション 185
2.6.3.3.2. Verbose、Quiet、Investigate グローバル オプション 185-186
2.6.3.3.3. SuppressIldasmAttribute グローバル オプション 186
2.6.3.3.4. Debug グローバル オプション 186-187
2.6.3.3.5. NoDotfuscatorAttribute グローバル オプション 187
2.6.3.4. 入力アセンブリ リスト 187-188
2.6.3.4.1. アセンブリ単位のライブラリ モード 188-189
2.6.3.4.2. アセンブリ単位の宣言による難読化 189-190
2.6.3.4.3. アセンブリ単位のインストルメンテーション処理 190
2.6.3.4.4. アセンブリ単位の XAML の変換モード 190
2.6.3.5. ユーザー定義のアセンブリ ロード パス 190-191
2.6.3.6. 出力ディレクトリ 191
2.6.3.7. 一時ディレクトリ 191-192
2.6.3.8. 難読化属性の機能割り当て 192
2.6.3.9. 名前の変更セクション 192
2.6.3.9.1. 名前の変更規則 192-193
2.6.3.9.2. 名前の変更オプション 193-195
2.6.3.9.3. 名前の変更の対象除外リスト 195
2.6.3.9.4. 名前の変更の参照規則 195-196
2.6.3.9.5. 出力割り当てファイル 196
2.6.3.9.6. HTML 形式の名前の変更レポート 196-197
2.6.3.9.7. 入力割り当てファイル 197
2.6.3.10. 制御フローの難読化セクション 197
2.6.3.10.1. 制御フローの難読化レベル 197-198

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.3.10.2. 制御フローの難読化オプション 198
2.6.3.10.3. 制御フローの対象除外リスト 198
2.6.3.11. 文字列の暗号化セクション 198
2.6.3.11.1. 文字列の暗号化オプション 199
2.6.3.11.2. 文字列の暗号化対象選択リスト 199
2.6.3.12. 除去セクション（不要コードの除去） 199
2.6.3.12.1. 除去無効オプション 199-200
2.6.3.12.2. ConstOnly オプション 200
2.6.3.12.3. 除去トリガー リスト 200-201
2.6.3.12.4. 条件付き対象選択リスト 201
2.6.3.12.5. 除去の参照規則 201
2.6.3.12.6. 除去レポート 201-202
2.6.3.13. リンク セクション 202
2.6.3.13.1. リンク無効オプション 202
2.6.3.13.2. リンク アセンブリ 202-204
2.6.3.14. PreMark セクション 204
2.6.3.14.1. PreMark オプション 204-205
2.6.3.14.2. PreMark の要素 205-206
2.6.3.15. 署名セクション 206-207
2.6.3.16. デジタル署名セクション 207-208
2.6.3.17. EventList セクション 208-210
2.6.3.18. Runtime Intelligence セクション 210-211
2.6.3.19. 拡張属性セクション 211-212
2.6.3.20. SmartObfuscation セクション 212-213
2.6.3.21. XML 構成ファイルに関する注意 213
2.6.3.22. カスタム規則リファレンス 213-214
2.6.3.22.1. 対象除外規則 214
2.6.3.22.1.1. 名前空間の対象除外 214-215
2.6.3.22.1.2. 型の対象除外 215-216
2.6.3.22.1.3. メソッドの対象除外 216-217
2.6.3.22.1.4. フィールドの対象除外 217-218
2.6.3.22.1.5. プロパティの対象除外 218

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.3.22.1.6. イベントの対象除外 219
2.6.3.22.1.7. カスタム属性による対象除外 219-220
2.6.3.22.1.8. スーパータイプによる対象除外 220-221
2.6.3.22.1.9. アセンブリの対象除外 221
2.6.3.22.1.10. モジュールの対象除外 221
2.6.3.22.2. 対象選択規則 221-222
2.6.3.22.2.1. 名前空間の対象選択 222
2.6.3.22.2.2. 型の対象選択 222-223
2.6.3.22.2.3. メソッドの対象選択 223-224
2.6.3.22.2.4. フィールドの対象選択 224-225
2.6.3.22.2.5. プロパティの対象選択 225-226
2.6.3.22.2.6. イベントの対象選択 226
2.6.3.22.2.7. カスタム属性による対象選択 226-227
2.6.3.22.2.8. スーパータイプによる対象選択 227-228
2.6.3.22.2.9. アセンブリの対象選択 228
2.6.3.22.2.10. モジュールの対象選択 228
2.6.3.23. dotfuscator_v2.3.dtd 228-229
2.6.4. カスタム属性リファレンス 229
2.6.4.1. PreEmptive.Attributes 229-230
2.6.4.1.1. ApplicationAttribute 230
2.6.4.1.2. BinaryAttribute 230-231
2.6.4.1.3. BusinessAttribute 231-232
2.6.4.1.4. ExceptionTrackAttribute 232-238
2.6.4.1.4.1. ExceptionTypes 238-239
2.6.4.1.5. FeatureAttribute 239-240
2.6.4.1.5.1. FeatureEventTypes 240-241
2.6.4.1.6. InsertShelfLifeAttribute 241-246
2.6.4.1.7. InsertSignofLifeAttribute 246-247
2.6.4.1.8. InsertTamperCheckAttribute 247-252
2.6.4.1.9. PerformanceProbeAttribute 252-253
2.6.4.1.9.1. InjectionPoints 253-254
2.6.4.1.10. SetupAttribute 254-260

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.4.1.11. SinkElements 260
2.6.4.1.12. SourceElements 261
2.6.4.1.13. SystemProfileAttribute 261-262
2.6.4.1.14. TeardownAttribute 262-263
2.6.5. 割り当てファイル 263-264
2.6.5.1. dotfuscatorMap_v1.1.dtd 264
2.6.6. 高度なトピック 264
2.6.6.1. サイド バイ サイド インストール 264-265
2.6.6.2. 並行ビルド 265
2.7. サンプル 265
2.7.1. リフレクションのサンプル 265
2.7.1.1. リフレクションのサンプル ファイル 265-266
2.7.1.2. リフレクション サンプルのビルド 266
2.7.1.3. リフレクション サンプルの実行 266-267
2.7.1.4. Dotfuscator によるリフレクション出力の処理 267-268
2.7.1.5. グラフィカル ユーザー インターフェイスを使用したリフレクション サンプルの構成 268-270
2.7.1.6. リフレクション サンプルのまとめ 270
2.7.2. シリアル化のサンプル 270
2.7.2.1. シリアル化のサンプル ファイル 270-271
2.7.2.2. シリアル化サンプルのビルド 271
2.7.2.3. シリアル化サンプルの実行 271-272
2.7.2.4. Dotfuscator によるシリアル化出力の処理 272-273
2.7.2.5. グラフィカル ユーザー インターフェイスを使用したシリアル化のサンプルの構成 273-276
2.7.2.6. シリアル化サンプルのまとめ 276
2.7.3. リモートのサンプル 276
2.7.3.1. リモートのサンプル ファイル 276-277
2.7.3.2. リモート サンプルのビルド 277
2.7.3.3. リモート サンプルの実行 277-278
2.7.3.4. Dotfuscator によるリモート出力の処理 278-280
2.7.3.5. グラフィカル ユーザー インターフェイスを使用したリモート サンプルの構成 280-282
2.7.3.6. リモート サンプルのまとめ 282
2.7.4. ASP.NET のサンプル 282-283

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.7.4.1. ASP.NET のサンプル ファイル 283
2.7.4.2. ASP.NET サンプルの準備 283
2.7.4.3. ASP.NET サンプルのビルド 283
2.7.4.4. ASP.NET サンプルの実行 283-284
2.7.4.5. ASP.NET サンプル コードの確認 284-285
2.7.4.6. Dotfuscator による ASP.NET 出力の処理 285-286
2.7.4.7. グラフィカル ユーザー インターフェイスを使用した ASP.NET のサンプルの構成 286-287
2.7.4.8. ASP.NET サンプルのまとめ 287
3. 索引 288-331

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.1 はじめに

難読化を行う理由

.NET 用に作成されたプログラムは、リバース エンジニアリングが簡単です。.NET アプリケーションは、MSIL（Microsoft Intermediate Language）と呼ばれる高度な、表現的ファイル構文にコンパイルします。これには、メソッド名と変数名が含まれており、簡単にソースの形式に逆コンパイルすることができます。

アタッカーは自由に利用できる逆コンパイラを使用して、あらゆる .NET アプリケーションのソースや、公開しているソフトウェアのライセンス コード、コピー保護メカニズム、所有権を有するビジネス ロジックなどを、合法的であるかどうかは別として、簡単に見ることができます。ソフトウェア アプリケーションの詳細を隅々まで調べることができるため、セキュリティ上の欠陥を探して利用したり、ユニークなアイデアを盗んだり、機能をクラックすることなども可能です。

オプションはこれだけではありません。難読化は、アセンブリ内のシンボルの名前をシームレスに変更する技法であり、その他にも逆コンパイラを妨害する対策を備えています。難読化を正しく適用すると、アプリケーションを変えることなく、逆コンパイルに対する保護を格段に高めることができます。

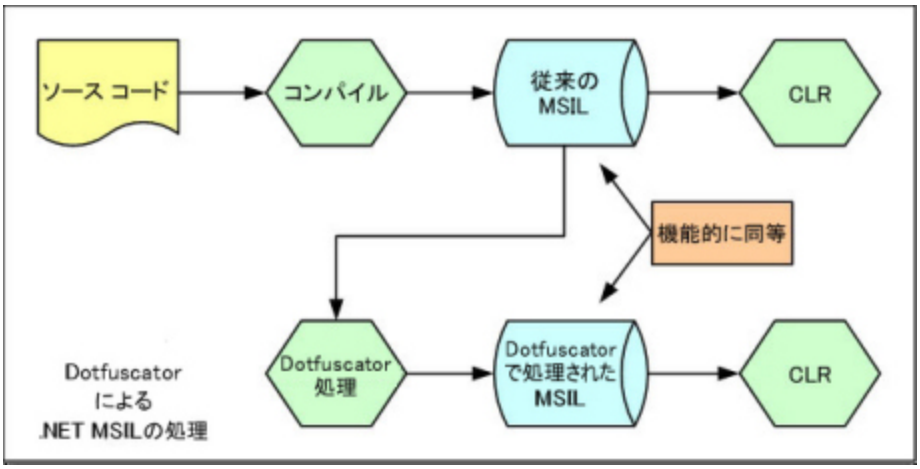
難読化ツールを判読可能なプログラム命令に対して適用すると、その出力を人間が解釈できなくなるほか、その人が使った逆コンパイラも機能しなくなるという副次的な効果が得られます。実行可能ロジックは保持されたまま、リバース エンジニアリングの意味が不明確になります。その結果、リバース エンジニアリングを試みると、変換があいまいなために失敗します。厳重な難読化を行うと、逆コンパイルの可能性が無数になり、そのうちのいくつかを再コンパイルすると、間違ったロジックが生成されることになります。計算機械である逆コンパイラには、どれが正しい意味に再コンパイルされるかを認識する手段はありません。人間は、ロジックを追うのが非常にたいへんな逆コンパイル アルゴリズムを自動化するため、逆コンパイラを記述し、使用します。逆コンパイラを混乱させる難読化プログラムは、逆コンパイラと同じことを行おうとする人間を思いとどまらせる効果があると言って間違いありません。

ビルド後の難読化

重要なのは、Dotfuscator は「ビルド後」のツールであるということを理解することです。つまり、これは既にコンパイルされたアプリケーション アセンブリを変更することで機能します。開発環境や開発ツールは、難読化に対応させるために変更されることはありません。また、ソース コードは、いかなる方法でも変更されず、読み取られることすらありません。難読化されたバイナリは、機能上、従来のバイナリと同等であり、共通言語ランタイム（CLR）で実行すると、結果は同じになります（ただし、逆は当てはまりません。難読化された MSIL を厳密に逆コンパイルできる場合でも、元のソース コードと比較すると、意味に重大な格差があります）。

次の図は、Dotfuscator の処理の流れを示しています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



要するに、通常のすべてのプロセスに従ってアプリケーションを開発し、ビルドした後、コンパイルされたバイナリで作業するように Dotfuscator を構成し、バイナリに対して Dotfuscator のビルド処理を実行するということです。Dotfuscator はバイナリを入力として受け取り、構成設定を通じて難読化を実行し、**同じバイナリのセット**を出力として生成します。ただし、それらのバイナリは難読化されます。

ほとんどの場合、アプリケーションの開発と難読化のプロセスは次のようになります。

1. プロジェクト ライフサイクルのほとんどについては、通常の開発および品質管理の手順に従います。
2. 中間地点を過ぎて、リリースにはまだ間があるどこかで、Dotfuscator UI を（Visual Studio または Dotfuscator 本体で）使用して、作成したアプリケーションで作業するように Dotfuscator を構成します。その際、基本的なスモーク テストを使用して、難読化によってアプリケーションで明らかな問題が引き起こされないことを確認します。
3. Visual Studio 統合、MSBuild タスク、または確かなコマンド ライン呼び出しを使用して、Dotfuscator をビルド処理に統合します。
4. プロジェクトの残りの品質管理で、難読化されたビルドを使用します。

2.1.1 Dotfuscator エディション

Professional

Dotfuscator Professional Edition は、最も有能で、最も強力なバージョンの Dotfuscator です。これは、商用アプリケーションおよび企業アプリケーションを製造する企業向けに設計されています。Dotfuscator Professional Edition を使用すると、逆コンパイラを妨害する優れた保護機能、メモリの節約と読み込み時間の短縮を可能にする高度なサイズ縮小、シームレスな構成のための緊密な Visual Studio 統合、パッチ リリースのための増分難読化、アセンブリに一意なタグを付けるウォーターマーク、そして電話およびテクニカル サポートが利用できます。

Community Edition

Dotfuscator Community Edition は、基本的な難読化機能を提供する無料版の難読化プログラムです。この主な目的は、識別子の名前を変更し、リバース エンジニアリングを阻止することです。Dotfuscator Community Edition は高度なテクノロジを組み込んでこのような保護を容易にし、短い識別子に名前を変更することによってサイズを縮小します。

Dotfuscator Community Edition は、以下のことは**行いません**。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- Microsoft SQL Server 内部で実行させるための、マネージ コードの Dotfuscator 処理。
- Visual Studio への緊密な統合、あるいは Visual Studio から独立して実行すること。
- Managed C++ アプリケーションのサポート。
- ClickOnce、APPX、または XAP パッケージのサポート。

相違の詳細については、http://www.preemptive.com/products/dotfuscator/compare-editions を参照してください。

以上の制限を持たない Dotfuscator が必要な場合は、Dotfuscator Professional Edition に関する詳しい情報が公開されている PreEmptive Solutions を参照してください。

このガイドには、Dotfuscator Community Edition に関する記載はありません。追加情報については、そのバージョンに付属のガイドを参照してください。

Dotfuscator for Marketplace Apps

Dotfuscator for Marketplace Apps には、Dotfuscator Professional Edition で利用可能な機能のほとんどが含まれています。ただし、これは Windows Phone 7/8 および、Windows ストアを経由する Windows RT を対象とするアプリケーション専用で、Dotfuscator for Windows Phone Edition の後継となることを意図しています。他のアプリケーションの種類のサポートが必要な場合、Dotfuscator Professional Edition についてのお問い合わせは、弊社営業部門にご連絡ください。

Dotfuscator for Marketplace Apps と Dotfuscator Professional Edition とで動作の異なる機能については、DMA アイコンで示します。

2.2 概要

このセクションでは、難読化のために Dotfuscator を使い始める方法について説明します。まず、ソフトウェアを登録することから始まり、Dotfuscator を使用する 3 つの標準的な方法（GUI、コマンド ライン、Visual Studio）を検討し、その後、難読化されたアセンブリの検査に取り組みます。

2.2.1 Dotfuscator の登録とアクティブ化

初めて Dotfuscator を使用するとき、登録を求められます。登録には 2 段階の手順が必要です。最初の手順では、まず登録フォームに入力します。その後、Dotfuscator がこの情報を電子メールまたは Web（いずれかを選択します）を経由して登録サーバーに送信します。登録サーバーがそれを正常に受信すると、サーバーはシリアル番号と確認コードを生成します。この情報はお客様が指定したメール アドレスに送信されます。

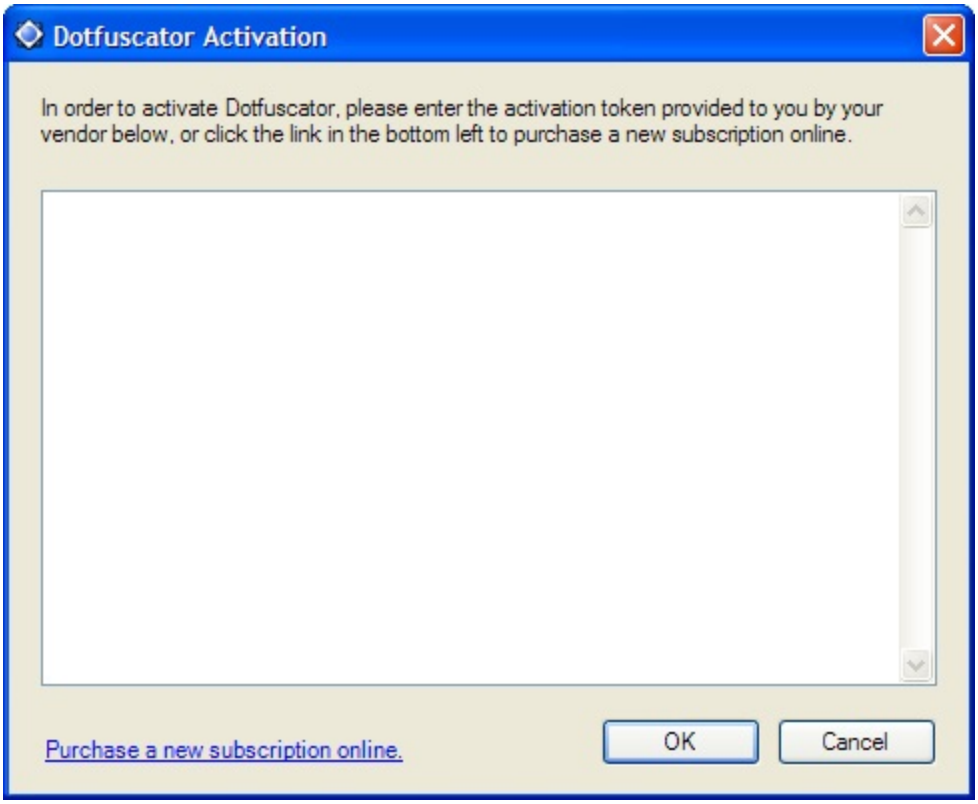
2 番目の手順は、Dotfuscator を次回実行するときに行います。このときに、電子メールで送付されたシリアル番号と確認コードを入力するように指示されます。入力した情報が確認されると、登録は完了です。

Dotfuscator のサブスクリプション ライセンス（永久ライセンスではなく）を購入した場合は、ユーザー登録後、PreEmptive のサーバーから登録情報およびサブスクリプション情報のダウンロードを試みます。サブスクリプションの有効期限が切れている場合は、サブスクリプションを更新するよう求められます。サーバーに接続できない場合は、ベンダーから提供されたアクティブ化トークンを入力するよう求められます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

グラフィカルなオフライン アクティブ化

Dotfuscator のグラフィカル ユーザー インターフェイスを使用しているときに、PreEmptive のサブスクリプション サーバーに接続できず、サブスクリプションをアクティブ化していない、またはサブスクリプションの有効期限が切れている場合は、オフラインのアクティブ化用ダイアログでアクティブ化トークンを入力するよう求められます。



アクティブ化ダイアログで[**Cancel**]を選択すると、Dotfuscator は機能縮小モードで読み込まれます。このモードで可能なのは、アクティブ化ダイアログへのアクセスのみです。これを行うには、[**Help | Activate Dotfuscator**]を選択します。

Dotfuscator は、サブスクリプションの有効期限が切れたときにそれを更新しなかった場合にも機能縮小モードで読み込まれます。Dotfuscator のステータス バーにはサブスクリプションの状態についての警告が示されます。アクティブ化ダイアログや Dotfuscator のステータス バーには、サブスクリプションの購入または更新についてのリンクも含まれています。

サブスクリプションを購入または更新後には、Dotfuscator を再起動するだけで、更新されたサブスクリプション情報が PreEmptive のサーバーからダウンロードされます。これで Dotfuscator は再び全機能モードで実行されます。PreEmptive のサーバーに接続できない場合は、アクティブ化ダイアログしか使用できません。

コマンド ラインによるオフライン アクティブ化

Dotfuscator のコマンド ライン インターフェイスを使用しているときに、PreEmptive のサブスクリプション サーバーに接続できず、サブスクリプションをアクティブ化していない、またはサブスクリプションの有効期限が切れている場合は、Dotfuscator は終了します。その際、`/offlineactivation` コマンド ライン オプションを使用してアクティブ化トークンを提供する方法を通知するメッセージが表示されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

サブスクリプションを購入または更新後には、Dotfuscator を再起動するだけで、更新されたサブスクリプション情報が PreEmptive のサーバーからダウンロードされます。PreEmptive のサーバーに接続できない場合は、**/offlineactivation** コマンド ライン オプションしか使用できません。

2.2.2 スタンドアロン GUI クイック スタート

このセクションでは Dotfuscator のスタンドアロン GUI の使い方について説明します。Dotfuscator のすべてのユーザー インターフェイスについては、「グラフィカル ユーザー インターフェイス リファレンス」を参照してください。

ステップ 1 - 既定のプロジェクトの作成

- Windows スタート メニューから Dotfuscator を起動します。
- [**プロジェクトを新規作成**]を選択して[**OK**]をクリックします。Dotfuscator のメイン プロジェクト ウィンドウが、[**入力**]タブが選択された状態で表示されます。
- 難読化するアセンブリを選択します。
  - [**入力の追加**]ダイアログ ボックスで[**参照**]をクリックします。
  - 次を参照します。

パス
`C:\Program Files (x86)\PreEmptive Solutions\Dotfuscator Professional Edition 4.x\samples\cs\GettingStarted\bin\Debug`

  - **GettingStarted.exe** を選択します。
  - [**開く**]をクリックします。
- 実行可能ファイルへのパスが[**入力ファイル**]の下のリスト ボックスに表示されます。
- [**ファイル**]メニューの[**プロジェクトの保存**]をクリックしてプロジェクトを保存します。
- [**プロジェクトの保存**]ダイアログで次の場所に移動します。

パス
`C:\Program Files (x86)\PreEmptive Solutions\Dotfuscator Professional Edition 4.x\samples\cs\GettingStarted\`

- [**ファイル名**]フィールドで **Dotfuscator.xml** と入力して[**保存**]をクリックします。

ステップ 2 - プロジェクトのビルド

- [**設定**]タブをクリックし、[**ビルド設定**]を選択します。[**出力先ディレクトリ**]フィールドは既定で次のように入力されています。**${configdir}\Dotfuscated**。

> **メモ**: **${configdir}** は Dotfuscator の構成ファイルへのパスが格納された変数です。

- プロジェクトを難読化する準備ができました。ツール バーの[**ビルド**]ボタンをクリックします。Dotfuscator が難読化された **HelloWorld** アプリケーションをビルドするまで、しばらく待ちます。難読化されたアセンブリは次の場所に格納されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

パス
C:\Program Files (x86)\PreEmptive Solutions\Dotfuscator Professional Edition 4.x\cs\GettingStarted\Dotfuscated

以上の手順を実行すると、[**出力**]タブに移動して難読化されたシンボルを参照したり、Dotfuscator が出力ディレクトリに作成した名前変更の割り当てファイル(**map.xml**)を確認したりできます。必要であれば、難読化したアプリケーションを実行することもできます。

次に、Dotfuscator の設定をさらに変更して、より強力な機能を使ってみることにします。

ステップ 3 - プロジェクトの構成

- [**設定**]タブをクリックして**グローバル オプション**をクリックします。
- [**一般**]セクションで、[**ビルドの進行状況**]プロパティに[**詳細な情報を表示する**]を設定します。このように設定すると、Dotfuscator の実行中、進行状況に関する詳しい情報がダイアログ下部の[**ビルド出力**]タブに表示されます。

- [**高度**]セクションで[**デバッグ シンボルの出力**]に[**JIT 最適化。PDB のシーケンス ポイント**]を設定します。このプロパティを設定すると、各出力アセンブリに対して、シンボル ファイルが PDB 形式で作成されます。デバッガーにこれらのファイルを使用すると、デバッグ セッションに関する有益な情報を確認することができます。通常は、行番号、ソース ファイル名、ローカル変数名などの情報です。PDB ファイルは出力アセンブリと一緒に**出力**ディレクトリに置かれます。
- [**機能**]セクションで、[**名前の変更を無効にする**]、[**制御フローを無効にする**]、[**文字列の暗号化を無効にする**]、および[**除去を無効にする**]に[**いいえ**]を設定します。こうすることで、Dotfuscator がアセンブリに適用する変換方法を細かく設定することができます。これらの機能を使用するための構成方法については、以下で説明します。

- [**設定**]タブをクリックし、[**レポート**]の下の[**名前の変更**]をクリックします。[*HTML として出力*]チェック ボックスをオンにすると、アプリケーションの名前変更情報と統計情報を含む有益なレポートを入手できます。このレポートは、割り当てファイルと同じディレクトリに出力されます。既定の場所は *${configdir}\Dotfuscated\Map.html* です。
- [**名前の変更**]タブをクリックして、[**オプション**] サブ タブをクリックします。[*拡張オーバーロード誘導を使用する*]をオンにします。この機能により、名前変更されたメソッド名およびフィールド名の重複性を最大 15% まで高めることができます。(C#、VB などの)ソース言語では戻り値やフィールドの型でオーバーロードすることは通常は許可されないので、逆コンパイルをいっそう困難にします。
- [**文字列の暗号化**]タブをクリックします。文字列の暗号化は選択されたものを対象とするため、入力アセンブリのすべてのメソッドを含めるためには、左ペインに表示されるアセンブリのチェック ボックス(ツリーのルートに表示)をオンにする必要があります。文字列の暗号化は、アプリケーション内の文字列をスクランブルします。たとえば、登録と検証の処理を迂回しようともくろむ者は、プログラムがユーザーにシリアル番号を要求する文字列を検索することがあります。その文字列を見つけると、その文字列の周辺の命令を探して、ロジックを変更するのです。文字列の暗号化は、検索で文字列が見つからなくすることにより、この作業を困難にします。

ステップ 4 - プロジェクトのリビルド

- [**ビルド**]をクリックすると、プロジェクトは難読化の準備が整います。前回と同様に、難読化されたアセンブリが次の場所に格納されます。

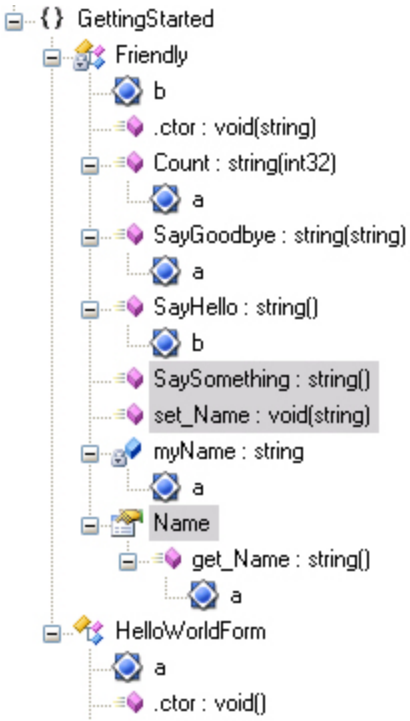
パス

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
C:\Program Files (x86)\PreEmptive Solutions\Dotfuscator Professional Edition
4.x\samples\cs\GettingStarted\dotfuscated
```

ステップ 5 - 出力の参照

- [**出力**]タブをクリックします。この画面では、Dotfuscator がコードを難読化した方法を示すツリーの中を移動することができます。
- ルート ツリーおよびすべてのサブフォルダー ツリーを展開します。**青い**菱形のアイコンは、名前が変更されたメソッドとフィールドを示しています。これらの各アイコンの上位には、元の名前が表示されています。すべてのメソッドとフィールドの名前が変更され、各メソッドの目的を判読するのがほぼ不可能になっていることを確認してください。これにより、コードのリバース エンジニアリング処理への対処として大きな効果を上げることができます。



- 現在強調表示されている **`SaySomething`** および **`set_Name`** メソッドだけでなく **`Name`** プロパティにも注目してください。Dotfuscator は、これらの項目がこのアプリケーションで使用されていないと判断しました。Dotfuscator の除去機能を使用して不要なコードを除去することで、アプリケーションのサイズをさらにコンパクトにすることができます。

次のステップ

これで、GUI を使って難読化を行うことができました。コマンド ライン インターフェイスで同じことを行うことができます。また、難読化された出力アセンブリを詳しく調べ、難読化の効果を理解することができます。

2.2.3 コマンド ライン クイック スタート

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

このセクションでは、スタンドアロン GUI クイック スタートと同じ設定でコマンド ライン インターフェイスを使用して難読化を行う方法を説明します。Dotfuscator コマンド ラインの完全なガイドはコマンド ライン インターフェイス リファレンスを参照してください。

コマンド ラインから Dotfuscator を起動するには、次の構文を使用します。

コマンド ライン クイック スタート
`dotfuscator [options] [configfile]`

コマンド ライン オプションは、「コマンド ライン オプションの概要」に詳しく説明されています。構成ファイルは Dotfuscator の各種オプションを指定するための XML ドキュメントです。スタンドアロン GUI を実行して各種ダイアログに値を入力すると、構成ファイルに設定が追加されます。構成ファイルのすべての要素の詳細については、「構成ファイル リファレンス」を参照してください。

既存の構成を使用する

先のセクションで作成した構成ファイルを、コマンド ライン ツールで使用することができます。作成済みの構成ファイルを使用する場合は、コマンドラインから次のようなコマンドを使って難読化を実行します。

既存の構成を使用する
`dotfuscator Dotfuscator.xml`

コマンド ライン スイッチのみを使用する

構成ファイルを使用せずに、コマンド ラインから Dotfuscator を実行することもできます。なぜなら、ほとんどの構成オプションが、コマンド ライン スイッチでも設定できるからです。入力アセンブリを次のように指定するだけで、先のセクションで使用したアセンブリを強力に難読化することができます。

コマンド ライン スイッチのみを使用する
`dotfuscator /in:GettingStarted.exe`

- **`in`** スイッチを使用すると、入力アセンブリのリストをカンマで区切って指定できます。
- 既定値では、出力アセンブリは **Dotfuscated** という名前の作業ディレクトリのサブディレクトリに置かれます。この設定は、コマンド ラインの **`out`** スイッチを使って変更できます。
- 既定値では、名前変更は有効に設定され、名前変更の割り当てファイルは **map.xml** という名前で作成されます。これも **Dotfuscated** のサブディレクトリに置かれます。この設定は、**`mapout`** スイッチを使って変更できます。
- 既定値では、文字列の暗号化、制御フロー、除去が有効に設定されます。

高度なコマンド ライン スイッチを使用する

前のセクションでスタンドアロン GUI に設定したオプションと同じ設定を使って、コマンド ラインから難読化プログラムを実行するには、次のようなコマンドを実行する必要があります。

高度なコマンド ライン スイッチを使用する
`dotfuscator /in:GettingStarted.exe /debug:on /v /enha:on`

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- 前のセクションと同じように **in** オプションを使用します。
- **v** オプションを指定すると、Dotfuscator が "詳細な情報を表示する" モードで動作します。
- **debug** オプションを指定すると、難読化された出力アセンブリに対してデバッグ シンボルが生成されます。
- **enha** オプションは拡張オーバーロード誘導を有効にします。

2.2.4 Visual Studio 統合 UI クイック スタート

このセクションでは、Visual Studio の中から Dotfuscator を使用する方法について説明します。結果として、Visual Studio のソリューション ビルドの一部として難読化を含むことができます。Dotfuscator プロジェクトは、他の 1 つ以上の Visual Studio プロジェクト（C#、VB.NET プロジェクトなど）からの入力ファイルを受け入れることもできますが、ファイルの参照ダイアログ ボックスから直接アセンブリを指定することもできます。

Visual Studio への Dotfuscator 統合については、「グラフィカル ユーザー インターフェイス リファレンス」の「Visual Studio 統合 UI の使用」を参照してください。

また、www.preemptive.com/demos.html でオンラインのデモを見ることができます。

DMA Dotfuscator for Marketplace Apps では、Visual Studio の中で Dotfuscator インターフェイスを利用できません。

ステップ 1 - Visual Studio で GettingStarted ソリューションを開く

Visual Studio の［ファイル］メニューから［**開く | プロジェクト/ソリューション**］をクリックして、次のファイルを開きます。

Getting Started
`C:\Program Files (x86)\PreEmptive Solutions\Dotfuscator Professional Edition 4.x\samples\cs\GettingStarted\GettingStarted.sln`

- ソリューションおよびプロジェクト ファイルは Visual Studio 7.0 形式です。それより新しいバージョンの Visual Studio をお使いの場合は、ファイルをアップグレードしてもよいかどうかを尋ねられます。続行するにはアップグレードに同意する必要があります。
- **GettingStarted** プロジェクトがソリューション エクスプローラーに表示されます。これは C# プロジェクトなので、これを使って **GettingStarted** 実行可能ファイルをビルドすることができます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ステップ 2 - Dotfuscator プロジェクトの作成

- Visual Studio の[**ファイル**]メニューで[**追加**]をポイントし、[**新しいプロジェクト**]をクリックします。
- [**プロジェクトの追加**]ダイアログで **Dotfuscator** プロジェクトをクリックします。[**Dotfuscator プロジェクト**]アイコンをクリックし、プロジェクトに *GetDotfuscated* と名前を付けます。[**OK**]をクリックしてプロジェクトを作成します。
- *GetDotfuscated* という名前のプロジェクトがソリューション エクスプローラーに表示されます。Dotfuscator プロジェクトを構成するための開始点としてソリューション エクスプローラーを使用します。
- 入力ファイルを指定する場合は、*GettingStarted* プロジェクトから出力を使用することを Dotfuscator に知らせます。**GetDotfuscated** ノードの最上位レベルで右クリックし、ショートカット メニューから[プロジェクト出力の追加]を選択します。この操作により、[**プロジェクト出力の追加**]ダイアログが表示されます。
- [**プロジェクト出力の追加**]ダイアログ ボックスで、[プロジェクト]ドロップダウンから **GettingStarted** プロジェクトを選び、[出力グループ]リストから**プライマリ出力**を選択します。[**OK**]をクリックします。
- ソリューション エクスプローラー内の Dotfuscator プロジェクトの**入力**ノードの下に、*GettingStarted の GettingStarted.exe (アクティブ)* が表示されます。

ステップ 3 - ソリューションのビルド

- ソリューション エクスプローラーで **GetDotfuscated** プロジェクト ノードを右クリックし、ショートカット メニューから[**プロパティ**]を選択します。作成した Dotfuscator プロジェクトのプロパティ ページが表示されます。左側のナビゲーション ツリーで**構成プロパティ**の**ビルド設定**をクリックします。[出力先ディレクトリ]には、既定で次のように入力されています。`${configdir}\Dotfuscated`

> **メモ**: `${configdir}` は Dotfuscator 構成ファイルのパスを保持する変数です。

- 既定の場所にサンプルがインストールされている場合、難読化されたアセンブリは次の場所に書き込まれます。

Building the Solution
`C:\Program Files (x86)\PreEmptive Solutions\Dotfuscator Professional Edition 4.x\samples\cs\GettingStarted\GetDotfuscated\Debug\Dotfuscated`

- ソリューションをビルドする準備ができました。[**ビルド**]メニューから[**ソリューションのビルド**]を選択します。最初に C# プロジェクトをビルドし、次に Dotfuscator プロジェクトをビルドします。Visual Studio の出力ウィンドウに Dotfuscator の出力が表示されます。

ソリューション エクスプローラー内で **GetDotfuscated** プロジェクトの**出力**ノードをダブルクリックすると、Dotfuscator の[**出力ブラウザー**]を表示できます。ここで、元の名前と Dotfuscator が適用された新しい名前を調べることができます。

次に、Dotfuscator の設定をさらに変更して、より強力な機能を使ってみることにします。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ステップ 4 - プロジェクト プロパティの構成

- ソリューション エクスプローラーで **GetDotfuscated** プロジェクト ノードを右クリックし、ショートカット メニューから[**プロパティ**]を選択します。
- 左側のツリーで**構成プロパティ**の**グローバル オプション**をクリックします。そうすると、Dotfuscator プロジェクトのグローバル オプションに関するプロパティ シートが表示されます。
  - [**機能**]セクションで、[**名前の変更を無効にする**]、[**制御フローを無効にする**]、[**文字列の暗号化を無効にする**]、および[**除去を無効にする**]の値に[*いいえ*]を設定します。こうすることで、Dotfuscator がアセンブリに適用する変換方法を細かく設定することができます。これらの機能を使用するための構成方法については、以下で説明します。
  - [**一般**]セクションで、[**ビルドの進行状況**]プロパティに[*詳細な情報を表示する*]を設定します。このように設定すると、ビルド中に、進行状況に関する詳しい情報が Visual Studio の出力ウィンドウに表示されるようになります。
  - [**高度**]セクションで[**デバッグ シンボルの出力**]プロパティに[*JIT 最適化。PDB のシーケンス ポイント*]を設定します(これは[デバッグ]プロジェクト構成の既定値です)。このオプションを設定すると、各出力アセンブリに対して、シンボル ファイルが PDB 形式で作成されます。デバッガーにこれらのファイルを使用すると、デバッグ セッションに関する有益な情報を確認することができます。通常は、行番号、ソース ファイル名、ローカル変数名などの情報です。PDB ファイルは出力アセンブリと一緒に出力ディレクトリに置かれます。
- **構成プロパティ**下の**レポート**で**名前の変更**をクリックします。[*HTML として出力*]チェック ボックスをオンにすると、アプリケーションの名前変更情報と統計情報を含む有益なレポートを入手できます。このレポートは、割り当てファイルと同じディレクトリに出力されます。既定の出力先は **`${configdir}\Dotfuscated\Map.html`** です。
- **構成プロパティ**下の**レポート**で**除去**をクリックします。除去レポート ファイル エディターで、除去レポート ファイルに `${configdir}\Dotfuscated\removal.xml` を指定します。また、[*HTML として出力*]をオンにすると、アプリケーションの除去情報と統計情報を含む書式設定されたレポートを入手できます。このレポートは **`removal.xml`** ファイルと同じディレクトリに出力されます。

ステップ 5 - 難読化設定の構成

- ソリューション エクスプローラーで Dotfuscator プロジェクトの**構成オプション** フォルダーを展開します。ここには、構成可能な難読化の設定ごとに 1 つのノードがあります。機能が無効になっている場合は、対応するノードが灰色で表示されます。機能の有効/無効を切り替えるには、ノードを右クリックして、[無効化]メニュー項目のチェックをオンまたはオフにします。あるいは、ステップ 4 で説明したように、[グローバル オプション]プロパティ シートを使用して設定できます。このセクションでは、名前の変更、制御フローの難読化、文字列の暗号化、および除去を使用します。
- **名前の変更**ノードをダブルクリックして、**オプション** サブタブをクリックします。[*拡張オーバーロード誘導を使用する*]をオンにします。この機能により、名前変更されたメソッド名およびフィールド名の重複性を最大 15% まで高めることができます。(C#、VB などの)ソース言語では戻り値やフィールドの型でオーバーロードすることは通常は許可されないので、逆コンパイルをいっそう困難にします。
- [HTML として出力]をオンにすると、アプリケーションの名前変更情報と統計情報を含む有益なレポートを入手できます。このレポートは割り当てファイルと同じディレクトリに出力されます。既定の場所は **`${configdir}\Dotfuscated\Map.html`** です。
- **文字列の暗号化**ノードをダブルクリックします。文字列の暗号化は選択されたものを対象とするため、入力アセンブリのすべてのメソッドを含めるためには、左ペインに表示されるアセンブリのチェック ボックス(ツリーのルートに表示)をオンにする必要があります。文字列の暗号化は、アプリケーション内の文字列をスクランブルします。たとえば、登録と検証の処理を迂回しようと試みる者は、プログラムがユーザーにシリアル番号を要求する文字列を検索することがあります。その文字列を見つけると、その文字列の周辺の命令を探して、ロジックを変更するのです。文字列の暗号化は、検索で文字列が見つからなくすることにより、この作業を困難にします。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

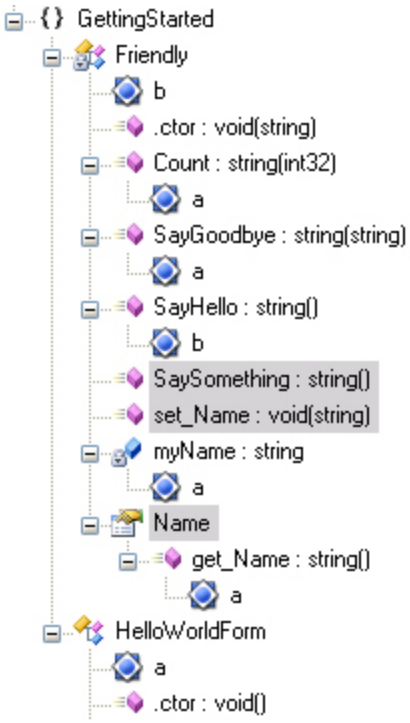
ステップ 6 - ソリューションのリビルド

- ソリューションを再度ビルドします。前回と同様に、難読化されたアセンブリが次の場所に格納されます。

Rebuild the Solution
`C:\Program Files (x86)\PreEmptive Solutions\Dotfuscator Professional Edition 4.x\samples\cs\GettingStarted\GetDotfuscated\Debug\Dotfuscated`

ステップ 7 - 出力の参照

- Dotfuscator プロジェクトの**出力**ノードをダブルクリックします。この画面では、Dotfuscator がコードを難読化した方法を示すツリーの中を移動することができます。
- ルート ツリーおよびすべてのサブフォルダー ツリーを展開します。**青い**菱形のアイコンは、名前が変更されたメソッドとフィールドを示しています。これらの各アイコンの上位には、元の名前が表示されています。すべてのメソッドとフィールドの名前が変更され、各メソッドの目的を判読するのがほぼ不可能になっていることを確認してください。これにより、コードのリバース エンジニアリング処理への対処として大きな効果を上げることができます。



- 現在強調表示されている **`SaySomething`** および **`set_Name`** メソッドだけでなく **`Name`** プロパティにも注目してください。Dotfuscator は、これらの項目がこのアプリケーションで使用されていないと判断しました。これを受けて、Dotfuscator の除去機能が不要なコードを除去し、アプリケーションのサイズをさらにコンパクトにします。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ステップ 8 - レポートの参照

- ソリューション エクスプローラーで **GetDotfuscated** プロジェクト ノードをクリックし、次にメニュー バーで[**表示**]の[**Dotfuscator**]をクリックします。このメニュー下にある 3 つのメニュー項目は、表示可能な HTML 形式の割り当てレポートと除去レポートがあれば有効になります。それらをクリックすると、対応するレポートが既定のブラウザーに表示されます。

次のステップ

以上の手順で、Visual Studio のアプリケーション内部を難読化することができました。コマンド ライン インターフェイスを使用して同じことができます。また、難読化された出力アセンブリを詳しく調べ、難読化の効果を理解することができます。

Dotfuscator の Visual Studio 統合についてさらに学ぶには、「Visual Studio 統合 UI の使用」を参照してください。

2.2.5 難読化後の出力の表示と検証

ステップ 1 - 逆アセンブラーの使用

.NET Framework SDK には、"ildasm" と呼ばれる逆アセンブラー ユーティリティが含まれています。これを使用すると、.NET アセンブリを IL アセンブリ言語のステートメントに逆コンパイルすることができます。ildasm を起動するには、.NET Framework SDK がインストールされていること、パスが設定されていることを確認してください。コマンド ラインから **ildasm** と入力します。

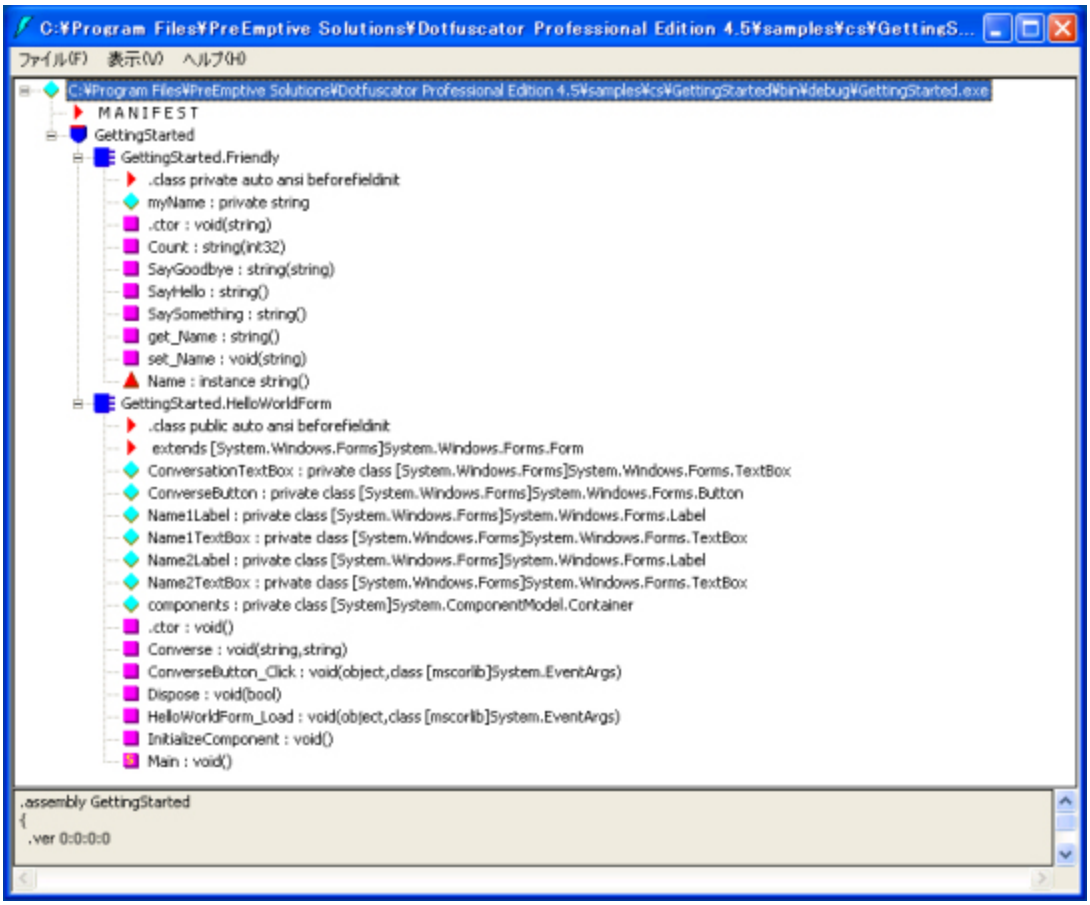
> **メモ**: Visual Studio がインストールされていて、ildasm が起動しない場合は、ildasm へのパスが設定されていない可能性があります。Visual Studio コマンド プロンプトを開くには、[**スタート**]メニューの[**Visual Studio[バージョン番号]**]をポイントし、[**Visual Studio Tools**]をポイントして[**Visual Studio[バージョン番号]コマンド プロンプト**]をクリックします。その後、**ildasm** と入力します。

- [**ファイル**]メニューの[**開く**]をクリックして、次を参照します。

パス
`C:\Program Files (x86)\PreEmptive Solutions\Dotfuscator Professional Edition 4.x\samples\GettingStarted\bin\Debug`

- **GettingStarted.exe** を選択します。
- [**開く**]をクリックします。逆アセンブルされたアセンブリのビューが表示されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

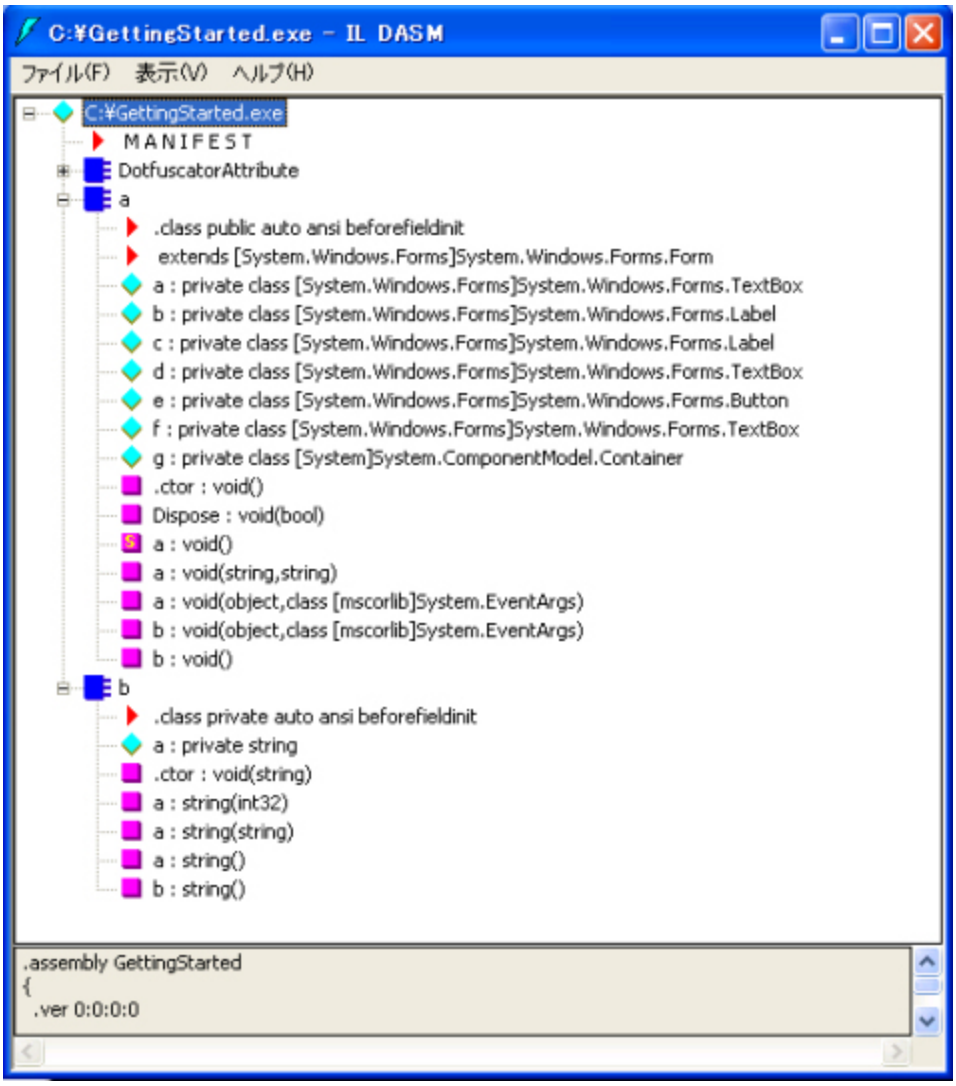


- 現在表示されている、難読化されていない **HelloWorld** アプリケーションを、難読化されたアプリケーションと比較するために、ildasm をもう 1 つ起動します。今度は以下を参照します。

パス
`C:\Program Files (x86)\PreEmptive Solutions\Dotfuscator Professional Edition 4.x\samples\GettingStarted\Dotfuscated`。

- **GettingStarted.exe** を選択します。
- ［**開く**］をクリックします。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



- 2 つの ildasm ウィンドウを並べて配置します。2 つの表示を比較してください。
- 難読化されていない逆アセンブリのウィンドウには、かなり理解しやすいメソッド名が表示されています。たとえば、**`ConverseButton_Click: void (object, class [mscorlib]System.EventArgs)`** というメソッドであれば、**Converse** ボタンがクリックされたときに呼び出されるメソッドと考えることができます。しかし、難読化されたアセンブリを見てください。Converse ボタンをクリックしたとき、どのメソッドが呼び出されるか推測できますか。見分けることは困難です。**`SaySomething`** メソッドがなくなっている点にも注目してください。これは、コード内のどこからも使用されていないメソッドが削除されたためです。
- **`SayHello:string()`** メソッドを元のアセンブリからダブルクリックし、**`SayHello:string()`** メソッドを難読化されたアセンブリからダブルクリックします。この 2 つのメソッドは同じものですが、逆アセンブルした IL コードをよく調べると、難読化されたアセンブリで文字列が暗号化されており、コードの判読が難しくなっているのがわかります。たとえば、難読化されていないアセンブリのウインドウから次の行を探してください。

暗号化されていない文字列:

```
IL_0000:  ldstr      "Hello, my name is "
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

次に、難読化された方のウィンドウから、この文字列を探してみてください。なかなか見つからないのは、文字列が暗号化されているためです。暗号化された文字列は次のように表示されています。

暗号化された文字列:

```
IL_0000: ldstr bytearray (09 42 26 44 29 46 2B 48 26 4A 67 4C 6D 4E 22 50
                          28 52 73 54 3B 56 36 58 34 5A 3E 5C 7D 5E 36 60
                          12 62 43 64 )
```

コードのリバース エンジニアリングを試みるアタッカーが、どれほど混乱させられるか想像してみてください。特に、複雑なアプリケーションの場合は、なおさら困難です。

ステップ 2 - 逆コンパイル

ソース コードを逆コンパイルできるのは、IL アセンブリ言語に詳しい知識の豊富なごく一部の人たちだけだと思うかもしれませんが、そうではありません。実際には、Reflector などの逆コンパイラを使用して、アプリケーションからソース コードを再作成することができます。これらのユーティリティを使うと、.NET のアセンブリを C#、VB、.NET、または Managed C++ などの高水準言語に直接戻すことが可能です。

このセクションでは Reflector for .NET http://www.red-gate.com/products/reflector/ を使用します。

Dotfuscator で処理された **GettingStarted.exe** ファイルを指定して .NET Reflector を実行し、**a()** などのメソッドを調べようとすると、次のメッセージが表示されます。

GettingStarted.exe を指定して .NET Reflector を実行する

```
This item appears to be obfuscated and can not be translated.
```

以上で、Dotfuscator を使用することによって、主要な逆コンパイラによるリバース エンジニアリングを完全に妨害できることが検証されました。

2.3 Dotfuscator による難読化の理解

Dotfuscator は、アプリケーションを保護するためのマルチパートの手法を提供しています。それぞれのパートが保護戦略全体の重要なピースを提供します。1 つのパートが危害を受けた場合、他のパートは有効な保護を提供し続けて、アタッカーが目標を達成することをますます難しくします。単に複数のパートがあるだけでなく、それぞれのパートが、他の難読化製品で使用されている単純な技術を超えた高度な機能を備えています。

最初のパートは難読化で、これは次のように細分されます。

- 名前の変更
- 制御フロー
- 文字列の暗号化

残りのパートは、難読化の域を超えて、アプリケーション全体の保護を強化します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- 不要コードの除去
- 改ざんの検出および防御
- Shelf Life
- ウォーターマーク

Dotfuscator はまた、難読化されたアセンブリのビルド、配置、およびデバッグに役立つ機能を提供します。次のものがあります。

- リンク
- 増分難読化
- 難読化されたコードのデバッグ

最後に、個別の言語機能やターゲット固有のアセンブリを使用するプロジェクトに適用される、追加の概念がいくつかあります。これらの詳細については、高度なトピックで読むことができます。

2.3.1 保護概念

2.3.1.1 名前の変更

Dotfuscator にはすべてのクラス、メソッド、およびフィールドを短い（領域を節約する）名前に変更する機能があります。逆コンパイルの出力を非常に理解しづらくするだけでなく、生成される実行可能ファイルのサイズが小さくなります。

大部分の商用難読化プログラムは、1 文字程に短い意味のない識別子を適用する名前変更技法を採用しています。難読化プログラムがコードを処理する際、置き換え用に次に利用可能な意味のない識別子を選択します。この表面上単純な名前変更方式は、キー属性を持っています。それは元に戻すことはできません。プログラム ロジックを保持したまま、名前が意味を持たなり、コードを解釈するすべての試みを防げます。 Dotfuscator は PreEmptive Solutions が Dotfuscatorのために開発した特許取得済みのオーバーロード誘導™をより厳重な難読化に使用します。オーバーロード誘導は、1 つの古い名前を 1 つの新しい名前に置き換えていくのではなく、できる限り多くのメソッドの名前を同じ名前に変更します。このように厳重に難読化を行うと、ロジックは壊れませんが、理解不能になります。実際の例を見てみましょう。次の簡単な例で、オーバーロード誘導技術の威力が理解できます。

難読化前のソース コード

```
private void CalcPayroll(SpecialList employeeGroup) {
   while (employeeGroup.HasMore()) {
        employee = employeeGroup.GetNext(true);
        employee.UpdateSalary();
        DistributeCheck(employee);
    }
}
```

Dotfuscator によるオーバーロード誘導を行った後のリバース エンジニアリングされたソース コード

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
private void a(a b) {
    while (b.a()) {
        a = b.a(true);
        a.a();
        a(a);
    }
}
```

この例でわかるのは、難読化されたコードは、名前変更の好ましい副作用としてサイズが小さくなっていることです。たとえば、20 文字の長さの名前の場合、名前を **a()** に変更するとサイズは 95% 縮小されます。また、名前の変更は文字列ヒープ エントリを節約することでも領域を節約します。すべての名前を "**a**" に変更するということは、**a** が 1 度だけ格納されることを意味し、名前が **a** に変更されたメソッドあるいはフィールドは、それぞれがその名前をポイントできるのです。オーバーロード誘導は、最短の識別子を連続的に再使用するので、この効果を高めます。

多くの顧客から、すべてのメソッドの最大 33% が "**a()**" という名前に変更されたという報告をいただいています。

この方法には明確な利点がいくつかあります。

- 名前の変更は、逆コンパイルした出力を理解しづらくします。しかしながら、逆コンパイラは印字不能文字あるいは、対象ソース言語では無効な名前を再度変更する機能を備えているため、識別子をそのような名前に変更することは無意味です。一方、オーバーロード誘導でメソッドの 3 分の 1 の名前が "**a()**" になることを考慮すると、逆コンパイルした出力を理解することは困難になります。
- オーバーロード誘導は、あらゆる名前変更システムに共通する制限以外の制限は受けません（そのような制限については後で説明します）。
- オーバーロード誘導は同じ文字を頻繁に使用する傾向があるので、（**aa**、**aaa** など）文字数の多い名前になるまでには時間がかかります。この点でも領域を節約します。

米国で特許を取得したオーバーロード誘導のアルゴリズムは、発生する可能性のあるすべての名前変更の衝突を判断し、安全な場合にのみメソッドのオーバーロードを誘導します。この処理が不可逆であることは実証可能です。つまり、（再度オーバーロード誘導を実行しても）元のメソッド名の関係を再構築するのは不可能です。

Dotfuscator は、メソッドの戻り値の型またはフィールドの型を、メソッドまたはフィールドの一意性を判断するときの条件として使用できるようにすることによって、**高度な Overload-Induction（オーバーロード誘導）**を提供します。この機能により、名前変更されたメソッド名およびフィールド名の重複性を最大 15% まで高めることができます。さらに、戻り値の型やフィールドの型によるオーバーロードは（C#、VB などの）ソース言語では通常は許可されていないので、逆コンパイルをいっそう困難なものにします。

2.3.1.2 制御フロー

従来の制御フローの難読化は、逆コンパイラを混乱させ、中断させるために、偽（にせ）の条件文やその他の紛らわしい構成体を追加します。この処理は、有効なフォワード（実行可能）ロジックを作り出している分岐処理、条件処理、反復処理などの構成要素を合成し、逆コンパイルが試みられたときに、その意味を判断できないコードを生成します。制御フローの難読化によって、アタッカーによる解析が非常に困難な、スパゲッティのように複雑にからみ合うロジックが生成されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Dotfuscator は、先進的な制御フローの難読化を使用しています。一方 Dotfuscator は、コード構造を追加するだけでなく、逆コンパイラがソース コードを再作成するのに使用するコード パターンを破壊することによって難読化を行います。結果として生成されるコードは、意味的にはオリジナルのコードと等価ですが、オリジナルのコードがどのように記述されていたかを示す手掛かりは含まれません。高度な逆コンパイラが開発されたとしても、その出力は当て推量でしょう。

次の例を考察してください。

難読化前のソース コード (c) 2001, Microsoft Corporation（WordCount.cs C# コード例の抜粋）

```
public int CompareTo(Object o) {
    int n = occurrences - ((WordOccurrence)o).occurrences;
    if (n == 0) {
        n = String.Compare(word, ((WordOccurrence)o).word);
    }
    return(n);
}
```

Dotfuscator による制御フロー難読化後にリバース エンジニアリングが行われたソース コード

```
public virtual int _a(Object A_0) {
    int local0;
    int local1;
    local0 = this.a - (c) A_0.a;
    if (local0 != 0) goto i0;
    goto i1;
    while (true) {
       return local1;
       i0: local1 = local0;
    }
    i1: local0 = System.String.Compare(this.b, (c) A_0.b);
    goto i0;
}
```

2.3.1.3 文字列の暗号化

Dotfuscator を使用すると、アセンブリに存在するユーザー文字列を隠すことができます。一般的なアタッカーの技法は、バイナリ内部の文字列参照を検索して、重要なコード セクションを見つけることです。たとえば、ユーザーのアプリケーションが時間でロックされる場合、アプリケーションは、タイムアウトが発生すると、メッセージを表示する可能性があります。アタッカーは逆アセンブルまたは逆コンパイルした出力に対し検索を行うことで、そのようなメッセージを検索します。そしてメッセージを見つけてしまったとき、アタッカーは機密性の高い時間ロック アルゴリズムに非常に近いところにいることでしょう。

Dotfuscator は、アプリケーションの機密性の高い部分の文字列を暗号化できるようにすることによってこうした問題に対処し、この種の攻撃に対する効果的な防御手段を提供します。

文字列の暗号化はわずかな実行時ペナルティが課されるため、アプリケーションの指定された箇所以外は、文字列の暗号化は実行されません。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.3.1.4 不要コードの除去

小さいアプリケーションほど、ダウンロード、インストール、読み込み、および実行速度がより高速になります。Dotfuscator の除去機能はコードを静的に分析して、未使用の型、メソッド、およびフィールドを検出し、それらを除去します。Dotfuscator は処理を行うときに、デバッグ情報と不必要なメタデータを MSIL ファイルから削除することで、アプリケーションをより小さくし、アタッカーが入手できるデータを減らします。

静的分析ではコード全体を調べ、"トリガー" またはエントリー ポイントと呼ばれる一連のメソッドから分析を開始します。通常、外部アプリケーションによって呼び出されることが予期されるメソッドは、トリガーとして定義されている必要があります。たとえば、単純なスタンドアロン アプリケーションでは、**`Main`** メソッドがトリガーとして定義されます。1 つのアセンブリに複数のトリガーを定義することができます。

> アセンブリのライブラリ モードをオンにすると、Dotfuscator は自動的に、公開されているすべての型とメンバーをエントリ ポイントとして扱うようになることに留意してください。

Dotfuscator は、各トリガー メソッドのコード全体を調べ、使用されているフィールド、メソッド、型を記録します。その後、呼び出されているメソッドを同様の方法で分析します。この処理は、呼び出されているメソッドがすべて分析されるまで続けられます。この処理が終了した時点で、アプリケーションの実行に最低限必要な型とそのメンバーのセットを判別できます。出力アセンブリには、これらの型だけが含まれます。

Dotfuscator は、すべての入力アセンブリの一覧と、それらがどのように除去されたかを示すレポートを XML 形式で生成します。列挙される各アセンブリには、型とそれらのメンバー（メソッド、フィールド、プロパティなど）の一覧と共に、その項目が除去されたかどうかを示す属性が表示されます。また、レポートには、各アセンブリにアタッチされているマネージ リソースの除去状況も記載されます。最後にレポートは、不要なコードの除去による全体的な効果に関する統計セクションを提供します。

2.3.1.5 改ざんの検出および防御

Dotfuscator では、アプリケーションの整合性を実行時に検証するコードを差し込むことができます。改ざんを検出した場合、アプリケーションのシャットダウンやランダム クラッシュの呼び出し（改ざんチェックの結果だったことを隠すため）、あるいは他の任意のカスタム動作を実行することができます。また、顧客のために PreEmptive Analytics Runtime Intelligence Service を使用して、改ざんが検出されたことを示すためにサービスにメッセージを送信することもできます。

2.3.1.6 Shelf Life

Shelf Life（使用期限）はアプリケーションの在庫管理機能で、期限切れや非アクティブ化、および通知のロジックをアプリケーションへ埋め込むことを可能にするものです。Dotfuscator は、アプリケーションの期限切れに対し、アプリケーションを終了したり Runtime Intelligence Service メッセージを送信したりして対処するためのコードを差し込みます。この機能は、ベータ版のアプリケーションでは特に有用です。ユーザーは、アプリケーションの期限切れや非アクティブ化を特定の日にスケジュールできます。また、オプションとして、特定の日数後にアプリケーションの期限が切れるか非アクティブ化される警告をユーザーに発することができます。

DMA Shelf Life は Dotfuscator for Marketplace Apps では利用できません。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Shelf Life（使用期限）の延長オプションを提供したい場合は、Shelf Life エンジンに更新された使用期限情報を提供するコードを記述して Shelf Life Token Source（たとえば外部設定ファイル）を指定すれば、これを実現できます。これは、インストルメンテーションとバイナリの再配布を必要とせずに、警告と有効期限を延長するためのメカニズムを提供します。

ユーザーは Shelf Life 機能を使用するために、まず **Shelf Life Activation Key（SLAK）**を入手する必要があります。このキーは PreEmptive によって発行され、Shelf Life の構成中にユーザーによって Dotfuscator に渡されます。

サポートされる .NET アプリケーションの種類

Dotfuscator は、以下に挙げるアセンブリ以外のすべての .NET アセンブリに対し、Shelf Life（使用期限）通知処理を行うことができます。

- ネイティブ コードおよびマネージ コードを含んでいる Managed C++ 入力アセンブリ
- マルチ モジュール入力アセンブリ
- Silverlight アセンブリ
- Windows Phone アセンブリ
- WinRT アセンブリ

2.3.1.7 ウォーターマーク

ウォーターマークは、ソフトウェアの無許可コピーの出所を追跡することに役立ちます。実行時の動作に影響を与えることなく、著作権情報や一意の ID 番号などのデータを .NET アプリケーション内に埋め込むために使われます。Dotfuscator のウォーターマーク アルゴリズムでは、アプリケーションのサイズが増加することはありません。また、アプリケーションに支障を来す余分なメタデータを生成することもありません。

2.3.2 難読化されたアプリケーションのビルドとデバッグ

2.3.2.1 アセンブリのリンク

Dotfuscator は複数の入力アセンブリを 1 つまたは複数の出力アセンブリに結合することができます。アセンブリのリンクを使用すると、特に、名前の変更および不要なコードの除去と併用した場合にはアプリケーションのサイズをさらに縮小することができ、配置シナリオを簡素化することができます。

たとえば、**A**、**B**、**C**、**D**、および **E** という入力アセンブリがあった場合、アセンブリ **A**、**B**、**C** を **F** という名前のアセンブリとしてリンクし、同時にアセンブリ **D** と **E** を **G** という名前のアセンブリとしてリンクすることができます。このリンクの注意点は、同じ入力アセンブリを複数の出力アセンブリにリンクできないということだけです。

アセンブリのリンク機能は Dotfuscator のその他の機能と完全に統合されているため、コードの難読化、未使用の型、メソッド、フィールドの除去、1 つのアセンブリへのリンクを 1 ステップで実行できます。

リンク対象のアセンブリを参照する他の入力アセンブリがある場合でも、アセンブリおよび型の参照の変更は透過的に行われ、出力アセンブリどうしは正しく連携します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Managed C++ アセンブリでは、リンク機能がサポートされていません。

2.3.2.2 増分難読化

アプリケーションに対し何度も Dotfuscator を実行するような場合には、増分難読化を使用することによって一貫性のある名前変更規則を維持することができます。名前割り当てファイルを参照することで、型やメンバーの名前を長期にわたり一貫して変更することができます。たとえば、従属グループの構成要素となるファイルのサブセットを再配布するとか、リソースに制約された環境でアセンブリの難読化を順次実行するなどの複合的なシナリオでは、名前の変更が一貫していることが望まれます。この機能を適切に活用すれば、プロジェクト内の一部のアセンブリのみ再配布が必要な場合には、難読化処理がより迅速になるという利点も生じます。

あるアプリケーションに対し Dotfuscator を使用し、そのアプリケーションを顧客に配布するというシナリオについて考えてみましょう。さて、アセンブリの 1 つに変更を加え、それをアップデートとして提供したいとします。当該アプリケーションに対し Dotfuscator を単純に再実行した場合、以前からあるクラスやメソッドの名前が前の変更結果とは異なるものにされてしまい、アプリケーション全体を顧客に再配布しなければならなくなる可能性があります。Dotfuscator の増分難読化を使用すれば、変更したアセンブリを公開できるため、同じ名前を保持できます。

順次ビルドのシナリオでも増分難読化が有用です。増分難読化によって、大規模なプロジェクトの難読化を、もっと処理しやすいアセンブリ グループの小規模な難読化に分解することが可能になります。3 つのファイル、**A.exe**、**B.dll**、および **C.dll** で構成される仮想プロジェクトがあり、**A** は **B** を、**B** は **C** を参照している場合は、このプロジェクトを次のようにビルドできます。最初に **C.dll** を難読化し、次に **C.dll** の実行で生成された割り当てファイルを使って **B.dll** を増分難読化してから、最後に **B.dll** からの割り当てファイルを使って **A.exe** を難読化することができます。

増分難読化には、再利用する必要のある名前を含んでいる入力割り当てファイルが必要です。ファイルの形式は、Dotfuscator を実行するごとに生成される出力割り当てファイルと同じです。最良の方法は、アプリケーションのビルドをリリースするごとに、出力割り当てファイルのコピーを安全な場所、たとえばバージョン管理などに保存しておくことです。そうすれば、増分更新が必要になったときはいつでも、そのファイルを入力割り当てファイルとして使用できます。

増分難読化を使用して実行する場合、プロジェクトに含まれるアセンブリがすべて必要なわけではなくても、Dotfuscator はアプリケーションのすべてのアセンブリへのアクセス権を持っていなければなりません。それらのアセンブリは、Dotfuscator が参照アセンブリを探すために使用する調査規則と同じ規則によって見つかりさえすればよいです。

2.3.2.3 難読化されたコードのデバッグ

難読化の重大な欠点に、アプリケーションの管理やトラブルシューティングの作業が困難になることがあります。アプリケーションが適切に難読化されると、すべての型、メソッド、フィールドの名前は、注意深く考え出されたわかりやすい名前から、意味のない名前に変更されます。これは、デバッガーを使うことをより困難にし、フィールドから提供されるスタック トレースの有用性に影響を与えます。

デバッグ問題を解決するために、Dotfuscator には、難読化されたアプリケーションのデバッグ シンボル ファイルを、コンパイラが出力する元のシンボル ファイルに可能な限り一致させて出力する機能があります。これらのファイルを使用すると、難読化されたアセンブリのデバッガーによるステップ スルーや、元のソース コードの確認が可能になります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

スタック トレース問題を解決するために、PreEmptive Solutions は **Lucidator** を提供します。これは、Dotfuscator を使って難読化されたプログラムによって作成されるスタック トレースの変換およびデコードを行う、スタンドアロン ツールです。Lucidator は、難読化されたスタック トレースを名前変更の割り当てファイルを使って自動的にデコードします。難読化されたスタック トレースを Lucidator に入力すると、難読化された名前が元の名前に置き換えられて、結果が表示されます。これと同様の変換機能は、Dotfuscator GUI にも組み込まれています。

Lucidator は適切なマップファイルへのアクセスである限り、Dotfuscator がインストールされているマシンと同じマシン上で実行する必要はありません。

2.3.3 高度なトピック

2.3.3.1 スマート難読化

「**スマート難読化**」は進行中の取り組みの 1 つで、自動的に既知の API 使用パターンおよびアプリケーションの種類を識別し、難読化規則を適用しようとするものです。

現在の実装では、次のようないくつかの共通のテクノロジやパターンを使用するアプリケーションおよびライブラリを認識できます。

- Windows Presentation Foundation（4.7.1000 から、アセンブリに対して XAML の変換オプションが有効な場合は、対象除外が無効とされます）
- Windows Communication Foundation
- Windows Workflow
- Windows Cardspace
- データ バインドされた Windows Forms コントロール
- 文字列として使用される列挙型の値
- VB アプリケーションの遅延呼び出し
- カスタム シリアル化
- Web サービス
- Silverlight（4.8.1000 から、アセンブリに対して XAML の変換オプションが有効な場合は、対象除外が無効とされます）

これらのアプリケーションの種類の場合、Dotfuscator の名前の変更機能および除去機能は、要素の名前を変更したり要素を削除したりすることによって出力アプリケーションが壊れる状況を特定できるよう、最善の努力をします。そのため、ユーザーが構成を追加しなくても、名前の変更および除去を自動的に回避できます。

スマート難読化規則は静的分析を使用して、名前の変更から除外したり、除去のエントリ ポイントとして使用する必要のある要素を判断します。そのような要素が見つかった場合、規則は通知を発行して［**スマート難読化**］タブに表示します。規則は操作が必要であると認識する場合がありますが、静的分析は十分な情報をもたらさないため、特定のどの操作を行うのかは決定できません。このような場合、規則は警告を発し、［**スマート難読化**］タブに表示します。

Dotfuscator では、スマート難読化を無効にできます。スマート難読化は既定で有効に設定されており、ほとんどの場合は有効にしておく必要があります。ただし、ユーザーが、もっと積極的な難読化を行ってもアプリケーションを損なわないと確信している場合には無効にすることができます。無効にする方法はいくつかあります。

- **`<smartobfuscation>`** セクションの **`<option>`** 設定を **`disable`** にする。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- Visual Studio UI の[プロジェクト プロパティ]ダイアログから。
- スタンドアロン UI の[設定]タブから。

Dotfuscator では、スマート難読化レポートの詳細を制御することができます。すべての操作と警告、または警告のみをレポートするか、まったくレポートしないかを選択できます。レポートのレベルを設定する方法はいくつかあります。

- 構成ファイルの smartobfuscationreport 要素に **`verbosity`** 属性を設定する。
- Visual Studio UI の[プロジェクト プロパティ]ダイアログから。
- スタンドアロン UI の[設定]タブから。

Dotfuscator では、スマート難読化レポートを表示するだけでなく、ファイルに保存することができます。以下のいずれかの場所でファイルのパスを指定することにより、レポートを保存するよう Dotfuscator に指示することができます。

- 構成ファイル内の smartobfuscationreport 要素内の **`file`** 要素。
- Visual Studio UI の[プロジェクト プロパティ]ダイアログから。
- スタンドアロン UI の[設定]タブから。

Dotfuscator は同じ名前の既存のスマート難読化レポートが検出された場合には、自動的にそのレポートの名前を変更してから、新しいレポートで既存レポートを上書きします。この動作を行わせたくない場合、Dotfuscator が既存のレポートを上書きする前に名前を変更しないように指示するには、いくつかの方法があります。

- 構成ファイルで、smartobfuscationreport 要素の **`overwrite`** 要素を **`true`** に設定する。
- Visual Studio UI の[プロジェクト プロパティ]ダイアログから。
- スタンドアロン UI の[設定]タブから。

スマート難読化レポート

Dotfuscator は、スマート難読化処理でフラグ付けされたすべての項目を一覧表示するスマート難読化レポートを XML 形式で生成します。レポートの内容はスマート難読化レポートの詳細設定を反映します。レポートの詳細設定が**警告のみ**または**なし**に設定されていると、スマート難読化でフラグ付けされた項目はレポートから除外されます。各エントリはスマート難読化によってフラグ付けされた 1 つの項目を表し、項目が何で、なぜ除外されたかの説明と、フラグ動作が警告であったか通知であったかを示す属性も示されます。

スマート難読化レポートの要素は、割り当てファイルおよび除去レポートの要素と類似しています。以下に、注意する必要のある点を示します。

- レポートには項目にフラグを付けた規則の名前、項目の ID、行われた動作、およびその動作の理由が含まれます。
- フラグ付けされた項目の ID は、項目の種類(型、メソッド、フィールド、プロパティなど)および完全な署名によって指定されます。
- 型名では、入れ子になったクラスの名前は "**/**" 文字で親と区切られます。
- コンストラクターには "**`.ctor`**" という名前が付けられますが、(静的初期化子、クラス コンストラクターなどとも呼ばれる)静的コンストラクターには "**`.cctor`**" という名前が付けられます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.3.3.2 P/Invoke メソッド

P/Invoke メソッド (**すなわち**、ネイティブ プラットフォーム メソッド) の名前は、対応するネイティブ関数の検索に使用されていることが Dotfuscator によって検出された場合、名前の変更対象から自動的に除外されます。エイリアスまたは序数によってネイティブ関数に対応付けられている P/Invoke メソッドの場合は、メソッドの名前を変更できます。

2.3.3.3 Managed C++ および IJW（It Just Works）のサンク

Dotfuscator は、たとえば Managed C++ コンパイラで生成されるような、マネージ コードとアンマネージ コード（ネイティブ コード）が共に含まれているアセンブリを処理することができます。Dotfuscator はそのようなコードの混在するモジュールに対して、名前の変更とメタデータの除去は行いますが、文字列の暗号化、制御フローの難読化、および不要なコードの除去については自動的に無効にします。ただし、これらの機能は、実行に含まれるその他の "純粋マネージ" 入力モジュールに対しては有効なままです。

2.3.3.4 マネージ リソースを持つアセンブリを Dotfuscator で処理する

マネージ リソースはモジュール（内部）に埋め込まれる場合と、モジュール外部のファイルに存在する場合があります。多くの場合、一部のマネージ リソースの名前は型の名前です（マネージ リソースの検索のための "ハブ アンド スポーク モデル" に関する詳細については、.NET Framework ドキュメントを参照してください）。

型の名前を変更すると、Dotfuscator は対応するマネージ リソースを見つけて名前を変更しようとします。リソースがアセンブリの内部にある場合、名前は自動的に変更されます。リソースが外部ファイル内に埋め込まれている場合、そのファイルは参照元のモジュールと同じディレクトリに存在する必要があります。リソースが他のアセンブリ内に埋め込まれている場合、そのアセンブリは入力アセンブリの 1 つである必要があります。

2.3.3.5 サテライト DLL を使用するアセンブリを Dotfuscator で処理する

ローカライズされたアプリケーションは、それらが使用するサテライト リソース DLL と共に、シームレスに難読化処理を行うことができます。Dotfuscator はランタイムが使用するルールと同じルールを使ってそのような DLL を自動検出し、難読化処理の入力として自動的に追加します。そのため、それらの DLL を入力として明示的に指定する必要はありません。

サテライト DLL に含まれるローカライズされたリソースは、メイン アセンブリ内にあるニュートラル カルチャのリソースと同期して名前が変更されます。

2.3.3.6 マルチ モジュール アセンブリを Dotfuscator で処理する

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

.NET アセンブリは複数のモジュール（**つまり**、ディスク上の複数のファイル）で構成されている場合があります。通常、1 つのアセンブリは 1 つのモジュールで構成されています。Visual Studio など、ほとんどのツールでサポートされているのは、このような単一モジュールのアセンブリです。しかし、ときには、複数のモジュールで構成されたアセンブリを作成することが望ましい場合もあります。Dotfuscator は、このようなマルチ モジュール アセンブリが使用されているシナリオもサポートします。マルチ モジュール アセンブリに対する Dotfuscator の処理は、複数の入力アセンブリを扱う場合の処理とは異なることに注意してください。

マルチ モジュール アセンブリを Dotfuscator で処理する場合、入力アセンブリとして指定する必要があるのは主モジュールだけです。主モジュール以外のモジュールは、主モジュールと同じディレクトリ内から検索されます。

主モジュールのアセンブリ マニフェストに記述されている他のモジュールのハッシュ値は、Dotfuscator によって自動的に更新されます。

2.3.3.7 厳密名のアセンブリを Dotfuscator で処理する

厳密名が付けられたアセンブリはデジタル署名されています。これにより、署名後にアセンブリが変更されていないかどうかを実行時に確認できます。署名には SHA1 ハッシュが使用され、ハッシュは RSA 公開/秘密キー ペアの秘密キーで署名されています。署名と公開キーは、共にアセンブリのメタデータに埋め込まれています。

Dotfuscator はアセンブリを変更してしまうため、Dotfuscator を使ってアセンブリを処理した**後に**署名を行うことは不可欠です。

Dotfuscator では、この署名の作成を難読化処理の一環として実行します。

難読化後の自動再署名

Dotfuscator では、難読化処理後に厳密名を持つアセンブリを自動的に再署名するため、手動で実行する必要がなくなりました。Dotfuscator は、既に署名済みのアセンブリを再署名し、遅延署名されているアセンブリの署名プロセスを完了します。

厳密名のアセンブリの再署名

Dotfuscator は、ビルド処理の一環として、既に厳密名が付けられているアセンブリを再署名することができます。公開/秘密キー ペアを探す場所を Dotfuscator に明示的に知らせるか、または入力アセンブリのカスタム属性（たとえば **System.Reflection.AssemblyKeyFileAttribute**）で指定される場所を使用することができます。

以下に、XML 構成ファイル内で明示的なキー ファイルを使用して再署名を設定する箇所を示します。カスタム属性によって指定されるキー ファイルはどれも使用されません。

明示的なキー ファイルによる再署名

```
<signing>
  <resign>
    <option>dontuseattributes</option>
    <key>
      <file dir="c:\temp" name="key.snk" />
    </key>
  </resign>
  ...
</signing>
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

遅延署名されたアセンブリの署名プロセスの完了

入力アセンブリが遅延署名されている場合、Dotfuscator はその署名プロセスを完了することができます。署名を完了するために必要な秘密キーの場所を Dotfuscator に知らせる必要があります。

以下に、XML 構成ファイル内で明示的なキー ファイルを使用して遅延署名を設定する箇所を示します。

遅延署名

```
<signing>
  ...
    <delaysign>
      <key>
        <file dir="c:\temp" name="key.snk" />
      </key>
    </delaysign>
</signing>
```

2.3.3.8 Authenticode 署名のアセンブリ

Authenticode で署名されたアセンブリは、信頼されたルート証明機関で発行したコード署名証明書によってデジタル署名されます。これを使用すれば、オペレーティング システムやランタイムがアプリケーションの発行元を特定したり、署名後にアセンブリが変更されたかどうかを判断することができます。この署名は、コード署名証明書の秘密キーで暗号化されたハッシュです。署名と公開キーは、共にアセンブリのメタデータに埋め込まれています。

Dotfuscator はアセンブリを変更してしまうため、Dotfuscator を使ってアセンブリを処理した後に Authenticode 署名を行うことは不可欠です。

Dotfuscator では、この署名の作成を難読化処理の一環として実行します。

難読化後の自動 Authenticode 署名

Dotfuscator はビルド処理の一環として、出力アセンブリの Authenticode 署名を実行します。PFX コンテナーとして格納するコード署名証明書を探す場所と、任意でそのコンテナーのパスワードを Dotfuscator に明示的に知らせる必要があります。

Dotfuscator では Authenticode 署名の実行時に Authenticode のタイムスタンプ サービスの URL を指定することができます。この URL は Dotfuscator の署名プロセス時にアクセスされ、そこから追加データが提供されます。その追加データを使用すれば、アセンブリの Authenticode 署名は、コード署名証明書の有効期限が切れた後も有効なままになります。

以下に、XML 構成ファイル内で Authenticode 署名を行う箇所を示します。

Authenticode デジタル署名

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<digitalsigning>
  <pfx password="secret123">
    <file dir="C:\test" name="authenticode.pfx" />
  </pfx>
</digitalsigning>
```

2.3.3.9 64 ビット アセンブリを Dotfuscator で処理する

Dotfuscator では、64 ビット アーキテクチャなど、特定の CPU アーキテクチャ用に明示的に作成されたマネージ アセンブリの難読化処理を透過的に実行できます。

Dotfuscator 自体はマネージ アプリケーションであり、32 ビットおよび 64 ビット バージョンの Windows 上で実行できます。

2.3.3.10 リフレクションおよび動的クラス読み込み

リフレクションおよび動的クラス読み込みは、.NET アーキテクチャにおいては非常に優れたツールです。しかし、プログラムの実行時にこのレベルのカスタマイズが行われると、Dotfuscator は指定されたプログラムに読み込まれたすべての型の名前を変更しても安全かどうかを確実に判断することができなくなります。

次の(C#)コードを考えてみましょう。

C# コード:

```
public object GetNewType() {
   Type type = Type.GetType( GetUserInputString(), true );
   object newInstance = Activator.CreateInstance( type );
   return newInstance;
}
```

このコードは名前で指定された型を読み込み、その型のインスタンスを動的に作成します。さらに、その名前はユーザーが入力する文字列によって指定されます。

ユーザーがどの型の名前を入力するかは、Dotfuscator では予測できません。その解決策として、読み込める可能性のあるすべての型の名前を対象から除外するために Dotfuscator を設定します。そうしても、メソッドおよびフィールドの名前を変更することは可能です。このような訳で、ユーザーの手作業による構成が重要な役割を果たしているのです。

多くの場合、状況は上記の場合ほど困難ではありません。次のコード例は、前述のコードを一部だけ変更したものです。

C# コードの変更例:

```
public MyInterface GetNewType() {
   Type type = Type.GetType( GetUserInputString(), true );
   object newInstance = Activator.CreateInstance( type );
   return newInstance as MyInterface;
}
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

コードから明らかなように、この例の場合、対象から除外する必要があるのは一部の特定の型（MyInterface を実装している型）だけです。

2.3.3.11 カスタム属性を使用した宣言による難読化

.NET Framework で、アセンブリの自動難読化を容易にするための 2 つのカスタム属性が用意されました。これらの属性により、構成ファイルを設定する必要がなくなりました。このセクションでは、Dotfuscator でこれらの属性を使用する方法を概説します。ここでの説明は、カスタム属性の基本的な使用方法や開発言語でのカスタム属性の適用方法を理解していることを前提としています。

以前のバージョンの .NET Framework をお使いの場合、Dotfuscator には、互換性のある属性が格納された DLL が付属しています。これらを使用するには、プロジェクトをビルドするときに **`PreEmptive.ObfuscationAttributes.dll`** を参照として追加します。ソース コード内の互換属性を参照する際に、**`System.Reflection`** 名前空間を **`PreEmptive.Dotfuscator.ObfuscationAttributes`** で置き換えます。

System.Reflection.ObfuscateAssemblyAttribute

この属性はアセンブリ レベルで使用され、Dotfuscator に対してアセンブリ全体の難読化方法を指示します。AssemblyIsPrivate プロパティを false に設定すると、Dotfuscator はそのアセンブリをライブラリ モードで実行します。このプロパティを true に設定すると、Dotfuscator はそのアセンブリをライブラリ モードでは実行せず、パブリックの型やメンバーなどを対象として可能な範囲で最大限の名前変更を行います。

System.Reflection.ObfuscationAttribute

この属性は型とそのメンバーを対象として使用され、Dotfuscator に対してその型やメンバーの難読化方法を指示します。

Feature プロパティ

この文字列プロパティの既定値は "**`all`**" です。このプロパティは、複数ある難読化の変換処理を個別に構成できるようにするために提供されています。それぞれ異なる機能文字列を持つ **`ObfuscationAttributes`** を 1 つの対象にタグ付けすることができます。

Dotfuscator は機能文字列の "**`default`**" と "**`all`**" を "**`renaming`**" に対応付けています。

次の一覧は、Dotfuscator が認識するその他の機能文字列をまとめたものです。

機能文字列	アクション
renaming	名前変更を構成します。
controlflow	制御フローの難読化を構成します。
stringencryption	制御フローの難読化を構成します。
trigger	指定された対象をエントリ ポイントとしてマークすることによって不要コードを除去するよう構成します。
conditionalinclude	指定された対象を条件付きで選択することによって不要コードを除去するよう構成します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

他の機能文字列に **renaming** を割り当てる必要がある場合は、[**設定**]タブの[**機能割り当て文字列**]プロパティ シートを使用して割り当てを設定できます。

Dotfuscator では、Dotfuscator が認識できない機能文字列が指定されている属性は無視されます。

Exclude プロパティ

このプロパティはブール型で、既定値は True です。True の場合、このプロパティは、タグ付けされた対象を Feature プロパティで指定された変換の対象から除外するように指示します。False の場合は、タグ付けされた対象を変換対象として選択するように指示します。

Dotfuscator の現在のバージョンでは、各変換処理に対して指定可能な Exclude プロパティの値は、それぞれの変換処理でサポートされている特定の 1 つの値のみです。Dotfuscator は、サポートされていない Exclude 値が指定されている規則を無視します。各処理でサポートされている値は次のとおりです。

機能文字列	サポートされている Exclude 値
renaming	True
controlflow	True
stringencryption	False
trigger	False
conditionalinclude	False

ApplyToMembers プロパティ

このプロパティはブール型で、既定値は True です。属性がアセンブリまたは型に適用される場合、このプロパティの True 値は、選択されている型の(入れ子になっている型を含む)すべてのメンバーに処理を適用するように指示します。このプロパティが false の場合、処理の対象となるのは型のみで、そのメンバーまたは入れ子になっている型には処理は適用されません。

宣言による難読化の有効化と無効化

Dotfuscator では、すべての入力アセンブリに対する宣言による難読化を有効または無効に切り替えることができます。無効になっている場合、難読化に関連するカスタム属性は無視されます。また、特定のアセンブリに対する難読化を無効に切り替えることも可能です。

宣言による難読化の属性の除去

Dotfuscator では、処理の完了後に難読化に関する属性を除去して、難読化の処理方法を調べる手掛かりを出力アセンブリ内に残さないようにできます。宣言による難読化の 2 つの属性にはブール型の "**StripAfterObfuscation**" プロパティが定義されています。このプロパティの既定値は true です。

また、Dotfuscator の構成設定には、難読化時の **StripAfterObfuscation** プロパティ値との組み合わせによって属性が除去されるかどうかを左右する設定項目があります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

次の表は、宣言による難読化の属性の除去に影響を与える設定項目を示し、それらの設定と StripAfterObfuscaton プロパティ値の組み合わせによって処理の結果がどのように変化するかをまとめたものです。

Dotfuscator で[難読化属性の使用]を選択	Dotfuscator で[難読化属性の除去]を選択	属性の StripAfterObfuscation プロパティ	結果
はい	はい	True または False	属性が除去される
はい	いいえ	True	属性が除去される
はい	いいえ	False	属性が保持される
いいえ	はい	True または False	属性が除去される
いいえ	いいえ	True または False	属性が保持される

機能割り当て文字列の使用

Dotfuscator では、難読化属性の Feature プロパティで指定された値に Dotfuscator が認識できる機能文字列を割り当てることができます。

たとえば、アプリケーションにおいて "**testmode**" という名前の機能を参照する難読化属性を指定したとします。Dotfuscator は既定値ではこの機能文字列を認識しないので、その属性は無視されます。後で、名前の変更や制御フローの難読化を設定するためにこれらの属性の使用が必要になった場合は、機能文字列の "**testmode**" に Dotfuscator に組み込みの "**renaming**" および "**controlflow**" 文字列を割り当てることによって、その設定を実現できます。

2.3.3.12 ビルド イベント

Dotfuscator では、ビルド工程の前後に実行するプログラムを指定できます。

ビルド前のイベント

Dotfuscator はそのビルド処理において、他のどの入力アセンブリを処理するよりも前に、ビルド前のイベントによって指定されたプログラムを実行します。

ビルド後のイベント

Dotfuscator はビルド処理の一番最後に、ビルド後のイベントによって指定されたプログラムを実行します。ビルドが成功した場合のみ、ビルドが失敗した場合のみ、または常時、プログラムを実行するように Dotfuscator に指示できます。また、出力モジュールごとに 1 回プログラムを実行するよう指示することもできます。

ビルド イベント プロパティ

Dotfuscator のビルド エンジンは、ビルド イベントを構成するときに使用できるプロパティをいくつか公開しています。

プロパティ名	説明
dotf.destination	出力先ディレクトリへのパス。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

dotf.inputmap.xml	入力割り当てファイルが指定されている場合、そのファイルの完全なパスと名前。
dotf.outputmap.xml	出力割り当てファイルが指定されている場合、そのファイルの完全なパスと名前。
dotf.removal.xml	XML 形式の除去ファイルが指定されている場合、そのファイルの完全なパスと名前。
dotf.config.file	現在の構成ファイルの完全なパス。
dotf.current.out.module	現在の出力モジュールの完全なパス。ビルド後のイベントで、出力モジュールごとにプログラムを呼び出す場合に使用されます。
dotf.current.in.module	現在の入力モジュールの完全なパス。ビルド後のイベントで、出力モジュールごとにプログラムを呼び出す場合に使用されます。

ビルド イベントでは、外部プロパティ(環境変数や、コマンド ラインで `-p` オプションを使って渡されるプロパティ)およびユーザー定義のプロジェクト プロパティも参照できます。

2.3.3.13 フレンド アセンブリ

.NET Framework では **フレンド アセンブリ** という概念が導入され、指定した他のアセンブリに対してアセンブリ内の型定義を公開する宣言を行えるようになりました。この宣言を行うには、`System.Runtime.CompilerServices.InternalsVisibleToAttribute` を使用します。Dotfuscator はこの属性の使用を検出し、設定されている名前の変更規則および不要コードの除去規則を以下の説明のように変更します。

A、**B** という 2 つのアセンブリがあり、**B** は **A** を参照し、**A** は `InternalsVisibleTo ( B )` とマークされているとします。

この場合、いくつかのケースが考えられます。

1. **A**、**B** 両方ともが入力アセンブリで、Dotfuscator がライブラリ モードである場合。**A** に他の外部の入力アセンブリ以外のフレンドがない場合は、Dotfuscator は内部名を安全に改変し、アセンブリ **B** 内の参照を修正します。**A** に他の外部フレンドがある場合、内部名は保持されます。
2. **A**、**B** 両方ともが入力アセンブリで、Dotfuscator がライブラリ モードでない場合。他の外部フレンド アセンブリが存在するかどうかに関係なく、**A** 内の内部名は既定で改変され、**B** 内の参照は修正されます。通常どおり、手動で構成することによって、名前および名前のグループを保持することができます。
3. **A** は入力アセンブリ、**B** は入力アセンブリ以外で、Dotfuscator がライブラリ モードである場合。**B** 内の潜在的な参照を壊さないために、**A** 内の内部名は改変されません。
4. **A** は入力アセンブリ、**B** は入力アセンブリ以外で、Dotfuscator がライブラリ モードでない場合。内部名は改変され、**B** 内の潜在的な参照は、**B** が入力アセンブリでないために修正されません。通常どおり、手動で構成することによって、名前および名前のグループを保持することができます。このケースでは、実際に **B** が **A** の内部を参照する場合は、手作業による構成が必要になります。

2.3.3.14 外部ツールの検索

Dotfuscator は Ildasm と Ilasm を使用して入力アセンブリを処理します。Ildasm は .NET Framework SDK に付属している MSIL 逆アセンブラーです。Ilasm は .NET Framework Redistributable に付属している MSIL アセンブラーです。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

.NET 1.1 以下がインストールされているシステムでは、Dotfuscator は各入力アセンブリに対して、そのアセンブリのコンパイルに使用されたバージョンの .NET Framework に付属するツールセットを使用しようとします。入力アセンブリに適合するバージョンのツールセットが見つからない場合は、より新しいバージョンが存在すれば、Dotfuscator はその新しいバージョンのツールを使用します。どのような場合でも、より古いバージョンのツールが使用されることはありません。

.NET 2.0 以上がインストールされているシステムでは、Dotfuscator は .NET 1.x アセンブリに対しても最新のツールを使用します。ビルド時、Dotfuscator は Ilasm に適切なコマンド ライン引数を渡して、出力アセンブリが正しいバージョンのフレームワークを対象とするようにします。

既定の場合、Dotfuscator は以下のアルゴリズムを使用してこれらの外部ツールを検索します。

- 入力アセンブリのコンパイルに使用された .NET Framework のバージョンを特定します。
- ツールの既定の場所に優先するユーザーが指定したプロパティを調べます。ツールおよびバージョン固有の方法で、これを行うことができます。次の表は、例を示しています。

プロパティ	値	
ILASM_v1.0.3705	C:\tools\ilasm.exe	.NET v1.0.3705 アセンブリは、このバージョンの Ilasm を使用します。
ILDASM_v2.0.50215	C:\tools\ildasm.exe	.NET v2.0.50215 アセンブリは、このバージョンの Ildasm を使用します。
ILDASM_v1.1	C:\tools\ildasm.exe	.NET v1.1.xxxx アセンブリは、このバージョンの Ilasm を使用します。

メモ:これらのプロパティは大文字小文字を区別します。

- ステップ 1 で特定された .NET Framework のバージョンに対応する .NET Framework ディレクトリおよび .NET Framework SDK ディレクトリ内を検索します。
- ステップ 1 で特定された .NET Framework のバージョンよりも新しいバージョンに対応する .NET Framework ディレクトリおよび .NET Framework SDK ディレクトリ内を検索します。

Dotfuscator が Ildasm および Ilasm の一方または両方を見つけられない場合はエラーとなります。

Dotfuscator は厳密名ツール(`sn.exe`)を使用して、厳密名のアセンブリの再署名を自動的に行います。このツールも .NET Framework SDK に付属しており、Dotfuscator は Ildasm と同じディレクトリでこのツールを探します。

2.4 Dotfuscator によるインストルメンテーションの理解

Dotfuscator は難読化のほかに、アプリケーションにコードを差し込む機能も提供しています。そのコードの一部は、難読化によりもたらされる保護を強化するため(つまり、改ざん通知と Shelf Life)に使用されます。また Dotfuscator では、アプリケーションが実働環境でどのように使用されるかについて、メッセージを送り返すためのコードを差し込むこともできます。この強力な機能により、アプリケーションの使用状況を深く理解することができ、このことが、ソフトウェアの優先順位についてより良い決定を下すために役立ちます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

この差し込まれたコードは「**インストルメンテーション**」と呼ばれ、ほとんどの場合、**PreEmptive Analytics** 製品のいずれか、Runtime Intelligence Service または PreEmptive Analytics for Team Foundation Server で使用されます。

Runtime Intelligence Service

Runtime Intelligence Service は、アプリケーションの使用実態に関する洞察をアプリケーション作成者に提供するホスト型ポータルです。Dotfuscator では、Runtime Intelligence Service へのデータ送信を行うインストルメンテーション コードを差し込むことができます。次の機能が含まれます。

- 改ざん通知
- Shelf Life
- 例外の追跡およびアプリケーション分析（以下で説明）

Dotfuscator は、アプリケーションが開始および終了したとき、機能が使用されたとき、あるいは例外が発生したときにメッセージが送信されるように、アプリケーションをインストルメント化することができます。また、インストルメンテーションでは、ランタイム プラットフォーム、一意のユーザーとインストール、およびアプリケーションのパフォーマンスに関するデータを送り返すことができます。Runtime Intelligence Service はこのライフサイクル データを集め、Runtime Intelligence Service のサブスクライバーが利用できる Runtime Intelligence ポータルを通じてこれを公開します。この機能を使用するには、Runtime Intelligence Service のサブスクライバーになる必要があります。

PreEmptive Analytics for Team Foundation Server（TFS）

PreEmptive Analytics for Team Foundation Server は、例外を集めて分析し、あなたが定義した規則と運用上のしきい値に完全に基づいて、Visual Studio / TFS 作業項目を自動的に作成します。PreEmptive Analytics for TFS は特に、フィードバック ドリブンの開発の合理化、ソフトウェアの品質とユーザー操作の向上、および開発効率の改善に役立つように設計されています。

PreEmptive Analytics for TFS は例外のインストルメンテーションを使用します。これは、機能、プラットフォーム、パフォーマンスなどのインストルメンテーションを必要としません。

サポートされる .NET アプリケーションの種類

Dotfuscator は、以下に挙げるアセンブリ以外のすべての .NET アセンブリにインストルメンテーション コードを差し込むことができます。

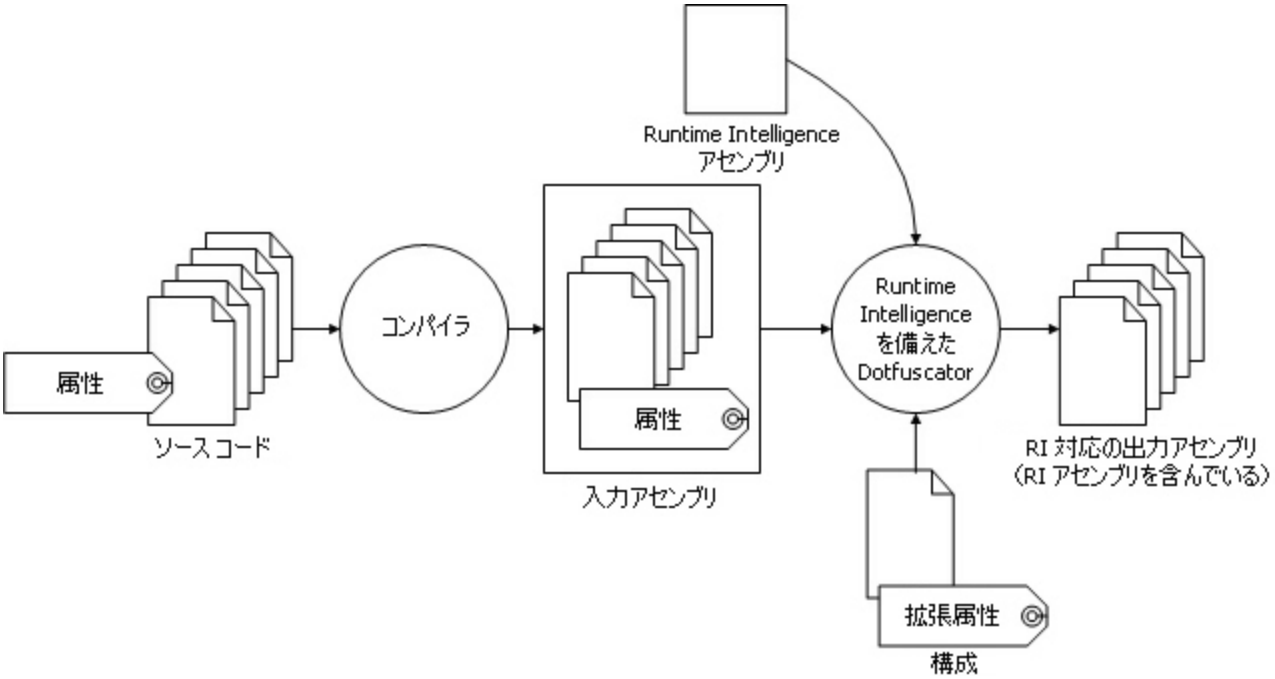
- ネイティブ コードおよびマネージ コードを含んでいる Managed C++ 入力アセンブリ
- マルチ モジュール入力アセンブリ
- .NET 1.0 を対象とする入力アセンブリ
- WinRT（近日サポート）

2.4.1 インストルメンテーションの差し込み

アプリケーションにアプリケーション分析を追加するための開発ワークフローは、アセンブリを難読化することに類似しています。どちらも特定のアセンブリで動作するように Dotfuscator を構成する必要があり、また、どちらも入力アセンブリと一致する一連の出力アセンブリを生成します。唯一の違いは、使用される特定の構成要素です。出力アセンブリには、データを収集および提供するために必要なすべてのランタイム コードが含まれます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

以下に、開発者の観点からのワークフローを図解します。



Runtime Intelligence メッセージの種類

Runtime Intelligence にはメッセージの種類がいくつか定義されています。

- アプリケーションおよびセッションの開始
- アプリケーションおよびセッションの終了
- 機能
- パフォーマンス プローブ
- システム プロファイル
- 改ざん検出
- 例外検出

アプリケーションおよびセッションの**開始**および**停止**メッセージ(**アプリケーションのライフサイクル メッセージ**)は、**アプリケーション**が実行を開始したときとシャットダウンしたときに送信されるものです。これらのメッセージに含まれる情報を使用して、アプリケーションの動作および基本的な使用パターンを追跡します。**機能**、**パフォーマンス プローブ**、または**システム プロファイル**のメッセージを使用して派生的な使用法や環境情報を取得します。

メッセージからのデータが Runtime Intelligence ポータルのダッシュボードを動かします。アプリケーションにこれらのメッセージを送信させるには、以下のことを実行する必要があります。

- Runtime Intelligence Service のサブスクライバーになります(これにより、ポータルのダッシュボードおよびデータにアクセスできるようになります)。
- Setup や Teardown などの Runtime Intelligence 属性をアプリケーションに設定します。これは、ソース コードの中で属性を用いて行えます。また、Dotfuscator GUI を介して属性を設定することによっても行えます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

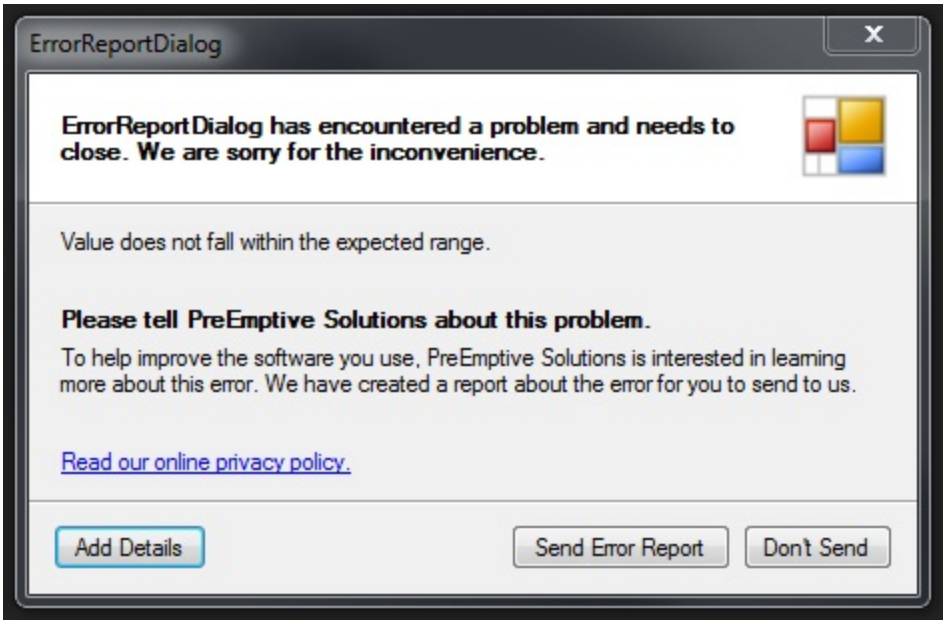
- ［**分析メッセージの送信**］オプションをオンにした Dotfuscator を介してアプリケーションを実行します。「アプリケーション分析を備えた Dotfuscator の構成および実行」を参照してください。

Runtime Intelligence データを含むメッセージの内容を見るには、「Runtime Intelligence 対応アプリケーションの例」を参照してください。

2.4.2 例外の追跡

例外の追跡は、ターゲット アプリケーションで例外が発生した際、その例外を自動的に検出し、対応するための 1 つの手段です。Dotfuscator は、キャッチされた例外、スローされた例外、あるいはハンドルされない例外を検出できるコードを差し込みます。検出されると、例外の追跡コードがユーザーから詳細を収集し、検出された例外を Runtime Intelligence エンドポイントに報告します。ユーザーは、前もって Runtime Intelligence メッセージの送信を止めてしまっている場合でも、例外レポートの送信を明示的に許可することができ、コメントや連絡先情報を提供してレポートと一緒に送信することができます。また、開発者は、例外が検出されたときに実行するカスタム動作を指定できます。

ハンドルされない例外の、よくある使用例の報告を容易にするために、Dotfuscator では、あらかじめ作成された例外レポート ダイアログを導入することができます。このダイアログにより、一貫性のあるユーザー操作が例外の報告に提供されるようになります。Dotfuscator に指示する必要のある最小構成は、ハンドルされない例外を追跡すること、例外レポート ダイアログを表示し、明示的なユーザー同意を得て、ユーザーから任意のコメントおよび連絡先情報を収集すること、そして、そのレポートを構成されている Runtime Intelligence エンドポイントへ送信することです。



2.4.3 改ざん通知

アプリケーションが改ざんされているかどうかを検出し、改ざんされている場合には任意で Runtime Intelligence Service のサブスクリプション顧客のために、Runtime Intelligence Service へメッセージを送信するようにアプリケーションをインストルメントすることができます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

適切に属性が設定された .NET アプリケーションで Dotfuscator を実行すれば、Dotfuscator は改ざん通知属性を処理し、適宜にアプリケーションをインストルメントします。結果として生じる出力アプリケーションは、いつでも改ざん通知を Runtime Intelligence データ サービスへ送信できる状態になっています。このレベルにおける改ざん通知とアプリケーション分析の唯一の違いは、属性にあります。

サポートされる .NET アプリケーションの種類

Dotfuscator は、以下に挙げるアセンブリ以外のすべての .NET アセンブリに改ざん処理を行うことができます。

- ネイティブ コードおよびマネージ コードを含んでいる Managed C++ 入力アセンブリ
- マルチ モジュール入力アセンブリ
- .NET 1.0 を対象とする入力アセンブリ
- .NET Compact Framework アセンブリ
- Silverlight アセンブリ
- Windows Phone アセンブリ
- WinRT アセンブリ

2.5 GUI による Dotfuscator 構成

Dotfuscator には利用可能な 2 種類のユーザー インターフェイスがあります。

- スタンドアロン GUI
- Visual Studio 用コンポーネントの緊密な統合セット

どちらのユーザー インターフェイスも、Dotfuscator プロジェクトのすべての側面を構成することができます。スタンドアロン GUI は一般に、大規模なプロジェクトや複雑な入力と出力を備えたプロジェクトで利用するのが望ましいです。Visual Studio GUI は一般に、通常は Visual Studio 内で完全にビルドされる小規模なプロジェクトで利用するのが望ましいです。どちらのユーザー インターフェイスも Dotfuscator プロジェクト ファイルを生成します。このファイルは、他のユーザー インターフェイスと互換性があり、自動ビルド スクリプトに統合することができます。

このセクションの最初の 2 つのトピックでは、2 つのユーザー インターフェイスで互いに異なるコンポーネントについて説明します。残りのセクションでは、両方に共通する要素と機能について説明します。

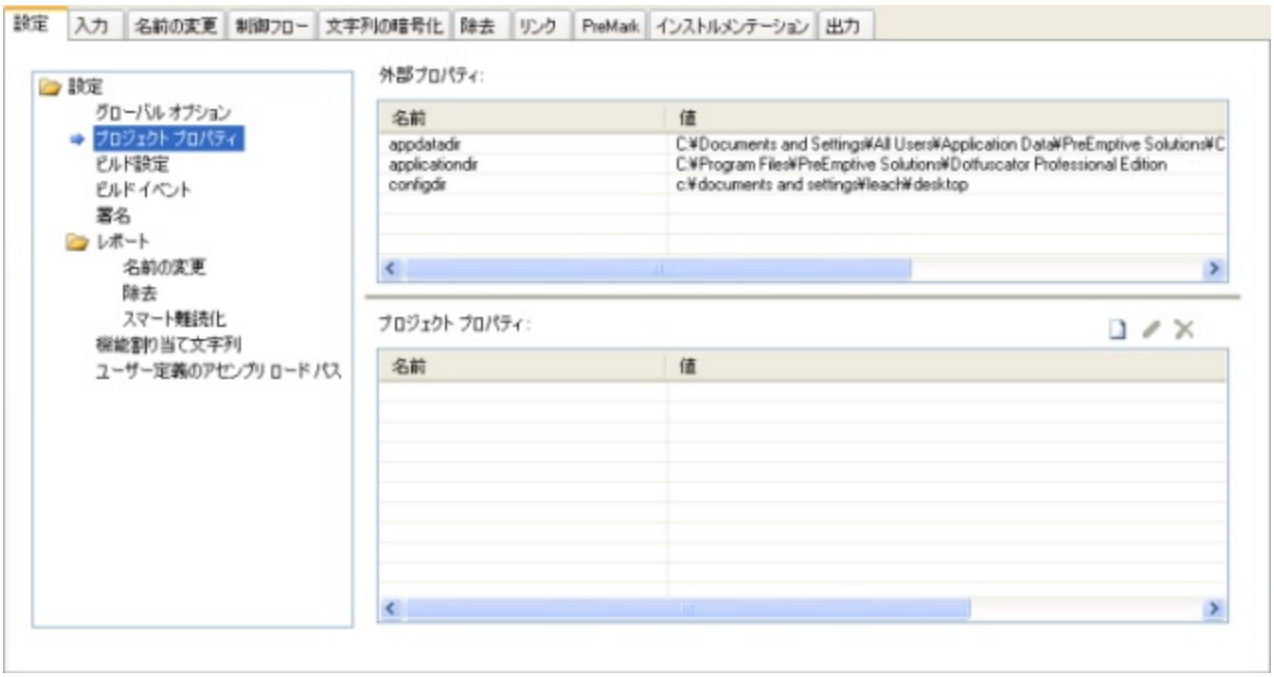
2.5.1 スタンドアロン GUI

メイン ウィンドウ

Dotfuscator のメイン ウィンドウは縦に配置された 4 つの操作領域で構成されています。

- ウィンドウの上部には、見慣れた Windows メニュー バー
- メニュー バーの下部には、頻繁に使用する操作のツール バーが表示
- ツール バーの下には、プロジェクトの設定項目やコマンドを実行するためのインターフェイスがタブごとに分類されて表示
- ウィンドウの最下部には、出力を見るためのタブ付きのスクロール可能なコントロール ペインが表示

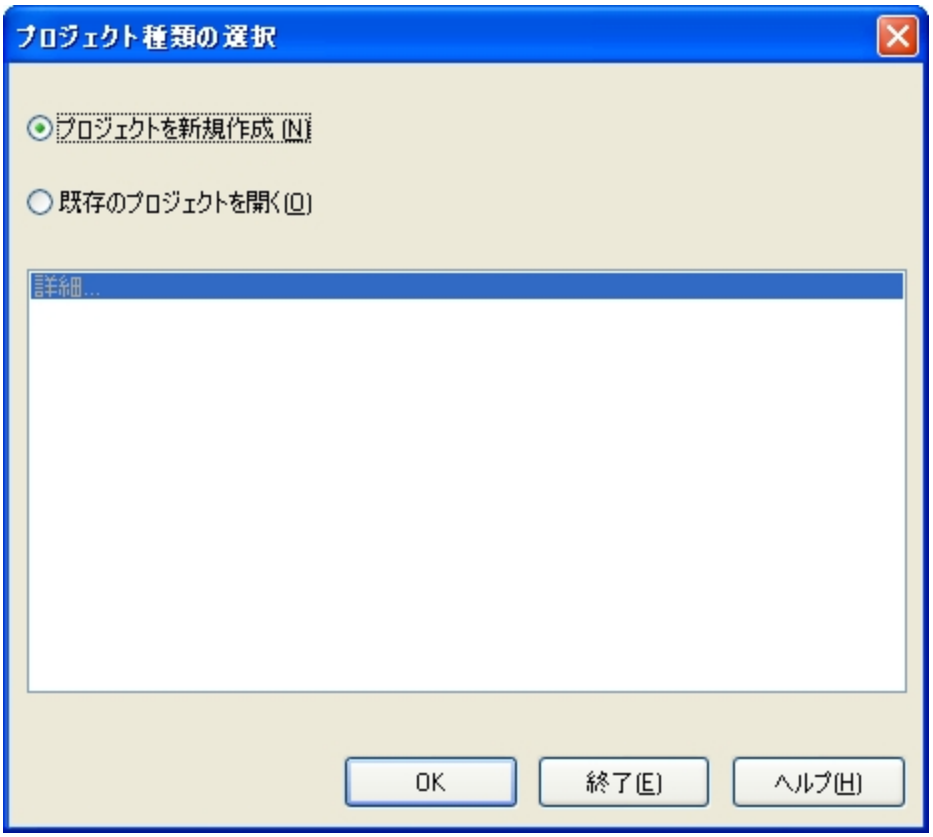
© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



[プロジェクト種類の選択]ウィンドウ

プロジェクトを読み込まずにユーザー インターフェイスを起動します。この操作を行うには、GUI のインストール ディレクトリから **`dotfuscator.exe`** を起動するか、またはインストーラーによって作成されたアイコンをクリックします。[**プロジェクト種類の選択**]ウィンドウが表示されるので、新しいプロジェクトを作成するか、または最近使用したプロジェクトの一覧から既存のプロジェクトを選択することができます。また、[**詳細...**] をクリックすることにより、ファイル システムを参照してプロジェクトを選択することも可能です。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



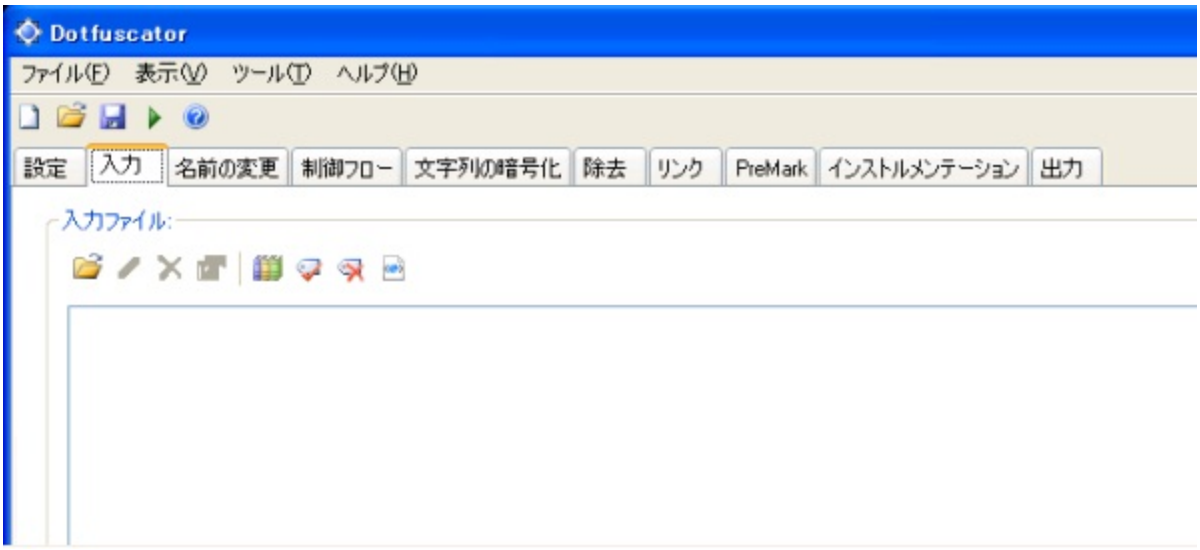
2.5.1.1 プロジェクトの作成

新しいプロジェクトを作成するには、次の 3 つの操作を実行する必要があります。

1:入力の選択

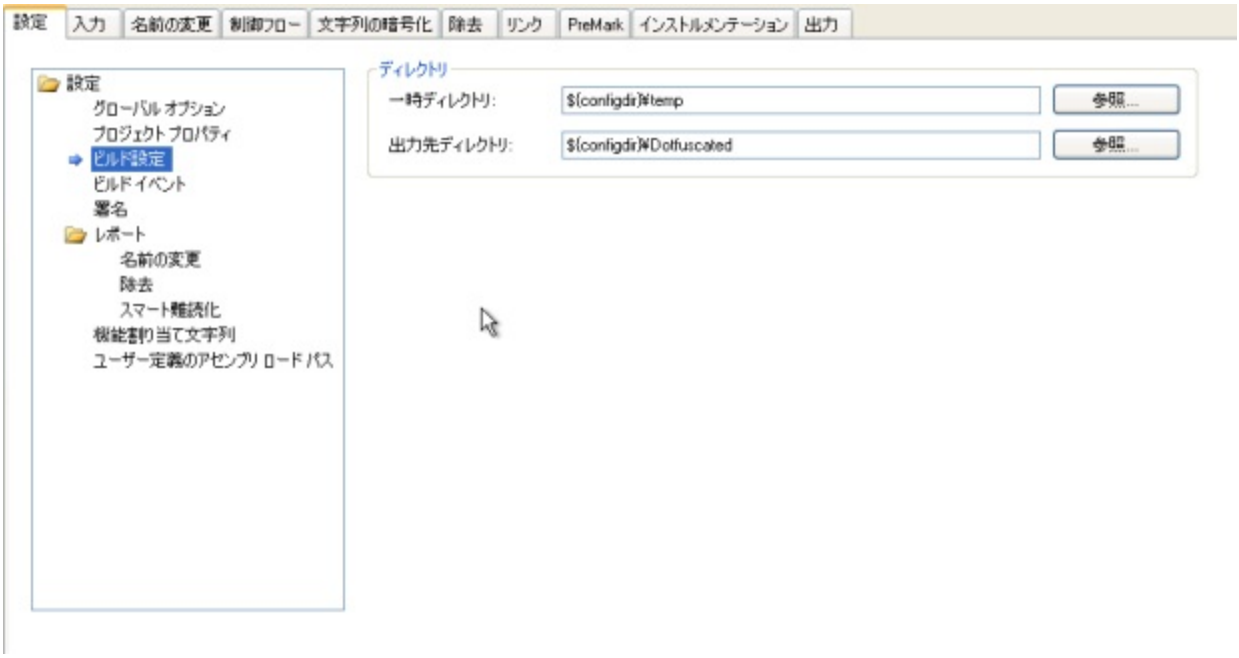
[**入力**]タブでは、ツール バーを使用して入力パッケージや入力アセンブリを追加できます。[**入力の追加**]ダイアログ ボックスで、パッケージやアセンブリのディレクトリとファイル名を入力したり、ディレクトリとファイル マスクを入力したり、あるいは、ファイル システムを参照して入力またはフォルダーを指定することができます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2:出力先ディレクトリの指定

新しいプロジェクトを作成した場合、出力先ディレクトリの既定値として **${configdir}\Dotfuscated** が設定されます。**${configdir}** は組み込みプロパティの 1 つで、プロジェクト ファイルの保存先のフォルダーに展開されます。



また、もう 1 つの方法として、ファイル システムを参照して適切な出力ディレクトリを選択することもできます。[**出力先ディレクトリ**] フィールドの右にある[**参照**]ボタンをクリックすると、[**出力先ディレクトリの選択**]ダイアログが表示されます。このダイアログにはディレクトリを参照するためのナビゲーション ツリーが表示されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

3:プロジェクト構成ファイルの保存

プロジェクトを保存するには、[**ファイル**]メニューの[**プロジェクトの保存**]または[**名前を付けてプロジェクトを保存**]をクリックするか、あるいはツール バーの[**プロジェクトを保存します**]をクリックします。プロジェクト ディレクトリに移動し、プロジェクト構成ファイル名を入力して[**保存**]をクリックすると、プロジェクトが保存されます。

2.5.1.2 入力の設定

アセンブリの追加

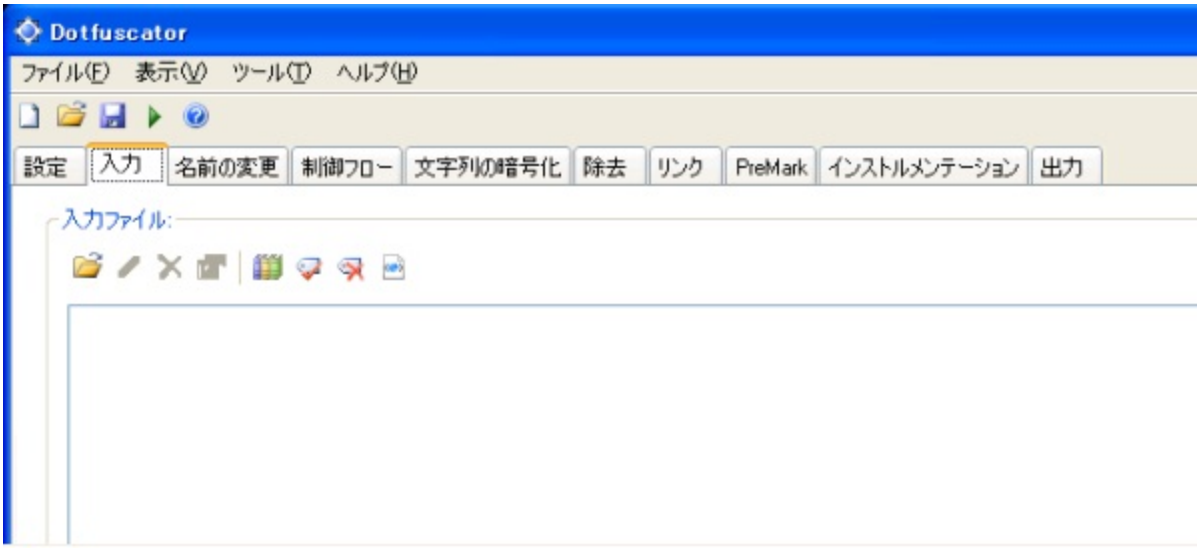
プロジェクトのパッケージおよびアセンブリの組み合わせを指定するには、[**入力**]タブを使用します。このタブでは、ツール バーの[**新しい入力を追加します**]ボタンを使用して新しいパッケージまたはアセンブリをプロジェクトに追加できます。このボタンをクリックすると[**入力の追加**]ダイアログ ボックスが表示されます。このダイアログ ボックスでは、パッケージまたはアセンブリのディレクトリとファイル名を入力するか、またはファイル システムを検索して追加対象を指定できます。

[**参照**]ダイアログ ボックスを使用すると、追加したいパッケージやアセンブリを複数選択することにより、複数の入力を 1 回の操作で追加できます。

また、[入力アセンブリ]リストにパッケージやアセンブリをドラッグ アンド ドロップすることもできます。

入力の編集と削除

プロジェクトの入力を変更するには[**選択した入力を編集します**]ボタンを使用し、プロジェクトからアセンブリを削除するには[**選択した入力をリストから削除します**]ボタンを使用します。これらの操作を実行するには、一覧で操作対象の入力を選択し、ツール バーの適切なボタンをクリックします。入力の削除は、削除対象の入力を選択して **Delete** キーを押す方法でも実行できます。



入力パスを編集する場合、入力するテキストにプロジェクト プロパティを埋め込むことができます。たとえば、次のようになります。

プロジェクト プロパティ

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
c:\${inputdir}\myapplication.exe
```

プロパティの置き換えは、「プロパティ リストとプロパティ」で説明している優先規則に基づいて実行されます。プロジェクト プロパティが埋め込まれている項目にカーソルを移動させると、置換されて実際に使用される値を完全な形で参照できます。

入力プロパティ

パッケージは固有のオプションを持つことができます。選択したパッケージに使用できるオプションがある場合は、ツール バーの[入力プロパティ]ボタンがアクティブになります。

ライブラリ モード

ツール バーの[すべてのアセンブリをライブラリ モードで実行します]ボタンを使用すると、すべてのアセンブリを対象として、ライブラリ モードの有効と無効を切り替えることができます

もう 1 つの方法として、リスト内の入力アセンブリ下に表示される[**ライブラリ モードで実行する**]チェック ボックスのオン/オフを切り替えることで、個々のアセンブリのライブラリ モードを選択することもできます。

XAML の変換モード

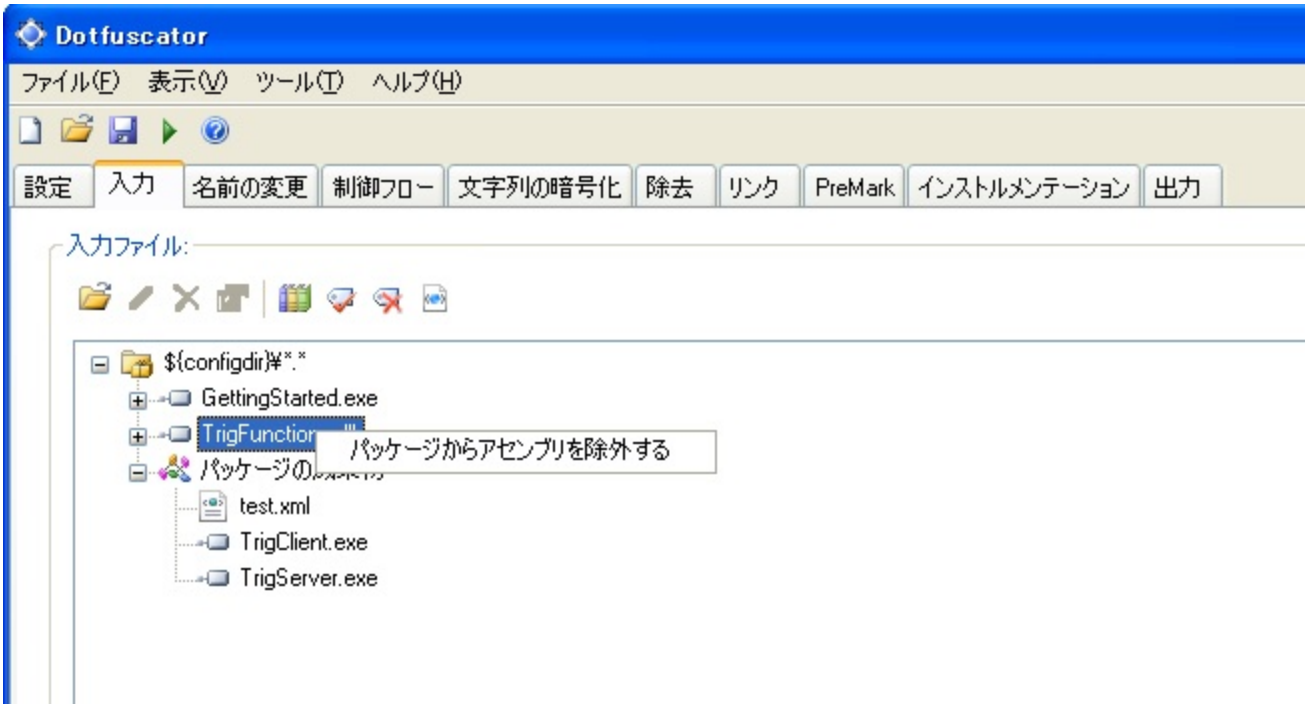
ツール バーの[XAML の変換]ボタンを使用すると、すべてのアセンブリを対象として、XAML の変換モードの有効と無効を切り替えることができます

もう 1 つの方法として、リスト内の入力アセンブリ下に表示される[**XAML の変換**]チェック ボックスのオン/オフを切り替えることで、個々のアセンブリに対する XAML の変換モードを設定することもできます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

パッケージ アセンブリを処理から除外する/処理に含める

パッケージ内のアセンブリ ノードを右クリックして[**パッケージからアセンブリを除外する**]を選択すると、パッケージに含まれる特定のアセンブリを Dotfuscator のラウンド トリップから除外することができます。これにより、Dotfuscator で処理されないパッケージ成果物のリストにアセンブリが追加されます。成果物リストに追加することで、アセンブリはあらゆる難読化およびインストルメンテーションを免除され、既存の厳密名および署名が保持されます。この設定は、もう一度右クリックして[**パッケージにアセンブリを含める**]を選択することにより切り替えることができます。



宣言による難読化

ツール バーの[**すべてのアセンブリに難読化属性を使用します**]ボタンと[**難読化属性をすべてのアセンブリから取り除きます**]ボタンを使用すると、すべての入力アセンブリを対象として、これらの設定の有効と無効を切り替えることができます。

もう 1 つの方法として、リスト内の入力アセンブリ下に表示される、これらの設定に対応するチェック ボックスのオン/オフを切り替えることで、個々のアセンブリの設定を構成することもできます。

インストルメンテーション属性

[**インストルメンテーション属性の使用**]および[**インストルメンテーション属性の除去**]設定は、リスト内の入力アセンブリ下に表示される、これらの設定に対応するチェック ボックスのオン/オフを切り替えることで、個々のアセンブリの設定を構成することができます。これらの設定の詳細については、「アプリケーション分析を備えた Dotfuscator の構成および実行」を参照してください。

2.5.1.2.1 ディレクトリ入力

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Dotfuscator には、Directory パッケージの入力を介して、1 つのディレクトリ内のすべてのファイルを難読化またはインストルメント化する機能が用意されています。Directory パッケージはディレクトリへの相対パスまたは絶対パスで構成され、任意で、一致させるファイルのワイルドカード指定子（ファイル マスク）を含みます。ファイル マスクに一致するすべてのマネージ アセンブリが、Dotfuscator への入力として使用されます。アンマネージ アセンブリや、ファイル マスクに一致しないその他のファイルは、パッケージの成果物として列挙されますが、Dotfuscator では処理されず、ビルド処理中に出力ディレクトリへコピーされます。

Directory パッケージを追加するには、[**入力の追加**]を選択して、[**入力の追加**]ダイアログ ボックスでパスとファイル マスク ワイルドカードを入力します。[**参照**]ボタンを選択し、目的のディレクトリへ移動して、ファイル名の入力欄を "**フォルダーの選択**" テキストのままにしておくことにより、ディレクトリ全体のファイル（*.*）を追加することもできます。明示的なパスを指定したり、プロジェクト プロパティを使用して、パスの一部または全部を置き換えるプロパティを指定したりできます。

すべてのプロジェクト設定は、Directory パッケージのすべての入力アセンブリに適用されます。また、パッケージ アセンブリの一覧に含まれるアセンブリに対し、対象除外規則を作成して Dotfuscator プロジェクトに保存できます。

2.5.1.2.2 Silverlight 入力

Dotfuscator には、Silverlight の展開ファイル *.XAP* を入力として指定する機能が用意されています。Silverlight パッケージは 1 つの XAP ファイルから成ります。このファイルが分析され、Silverlight パッケージとしてユーザー インターフェイスに表示されます。XAP に含まれるすべてのマネージ アセンブリが Dotfuscator の入力として使用されます。XAP に含まれるマネージ アセンブリ以外のファイルは、パッケージの成果物として列挙されますが、Dotfuscator で処理されずに、出力ディレクトリの出力 XAP ファイルに含められます。Dotfuscator は入力 XAP を基に、難読化およびインストルメント化されたアセンブリ、更新されたマニフェスト、およびアセンブリ ファイル以外のあらゆるファイルを含んでいる、1 つの XAP を出力します。

Silverlight パッケージを追加するには、[入力の追加]を選択して、[**入力の追加**]ダイアログ ボックスでパスとファイル名を入力します。[**参照**]ボタンを選択して検索することにより、特定の XAP ファイルを参照することもできます。明示的なパスを指定したり、プロジェクト プロパティを使用して、パスの一部または全部を置き換えるプロパティを指定したりできます。

すべてのプロジェクト設定は、Silverlight パッケージのすべての入力アセンブリに適用されます。また、パッケージ アセンブリの一覧に含まれるアセンブリに対し、対象除外規則を作成して Dotfuscator プロジェクトに保存できます。

署名された Silverlight 4 パッケージは Dotfuscator によって再署名することができます。署名オプションは[**パッケージ プロパティ**]ボタンまたはショートカット メニューのエントリからアクセスできます。オプションには[**証明書ファイル**]、[**証明書パスワード**]、および[**タイムスタンプ サーバーの URL**]設定があります。Silverlight 証明書オプションとして、出力 XAP ファイルの署名に使用する証明書ファイルのコンテナー、任意で証明書パスワードおよびタイムスタンプ サーバーを設定します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

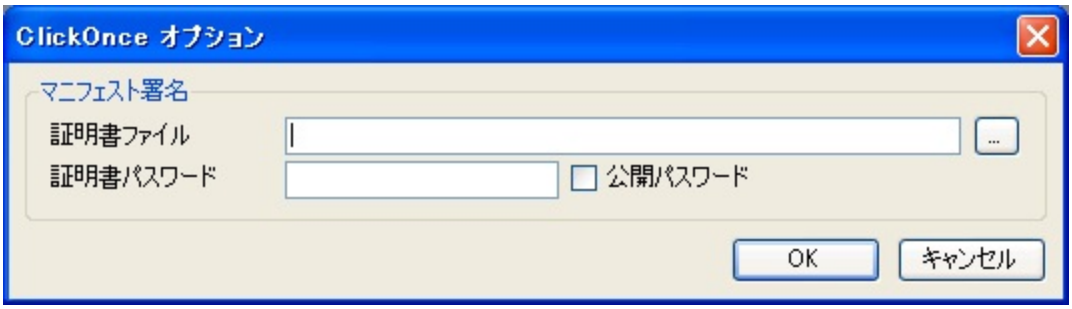
2.5.1.2.3 ClickOnce 入力

Dotfuscator には、ClickOnce の配置マニフェスト ファイル **.APPLICATION** を入力として指定する機能が用意されています。ClickOnce パッケージは 1 つの APPLICATION ファイルから成ります。このファイルが分析され、ClickOnce パッケージとしてユーザー インターフェイスに表示されます。application に含まれるすべてのマネージ アセンブリが Dotfuscator の入力として使用されます。配置およびアプリケーション マニフェストに含まれるマネージ アセンブリ以外のファイルは、パッケージの成果物として列挙されますが、Dotfuscator で処理されずに、出力ディレクトリの出力 application ファイルに含められます。Dotfuscator は入力マニフェストを基に、難読化およびインストルメント化されたアセンブリ、必要なすべてのマニフェスト、およびアセンブリ ファイル以外のあらゆるファイルを出力ディレクトリに出力します。

ClickOnce パッケージを追加するには、[**入力の追加**]を選択して、[**入力の追加**]ダイアログ ボックスでパスとファイル名を入力します。[**参照**]ボタンを選択して検索することにより、特定の配置マニフェスト ファイルを参照することもできます。明示的なパスを指定したり、プロジェクト プロパティを使用して、パスの一部または全部を置き換えるプロパティを指定したりできます。

すべてのプロジェクト設定は、ClickOnce パッケージのすべての入力アセンブリに適用されます。また、パッケージ アセンブリの一覧に含まれるアセンブリに対し、対象除外規則を作成して Dotfuscator プロジェクトに保存できます。

ClickOnce パッケージには、[**パッケージ プロパティ**]ボタンまたはショートカット メニューのエントリからアクセスできる 2 つの必須プロパティ、[**証明書ファイル**]と[**証明書パスワード**]があります。ClickOnce 証明書オプションは、すべての出力マニフェスト ファイルの署名に使用される、証明書および証明書パスワード(任意)を照会します。



2.5.1.2.4 Windows ストア入力

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Dotfuscator には、Windows ストアの展開ファイル *.APPX* を入力として指定する機能が用意されています。Windows ストア パッケージは 1 つの APPX ファイルから成ります。このファイルが分析され、Windows ストア パッケージとしてユーザー インターフェイスに表示されます。APPX に含まれるすべてのマネージ アセンブリが Dotfuscator の入力として使用されます。APPX に含まれるマネージ アセンブリ以外のファイルは、パッケージの成果物として列挙され、出力ディレクトリの出力 APPX に含められます。Dotfuscator は入力 APPX を基に、難読化およびインストルメント化されたアセンブリ、更新されたマニフェスト、およびアセンブリ ファイル以外のあらゆるファイルを含んでいる、1 つの APPX を出力します。

Windows ストア パッケージを追加するには、[入力の追加]を選択して、[**入力の追加**]ダイアログ ボックスでパスとファイル名を入力します。[**参照**]ボタンを選択して検索することにより、特定の APPX ファイルを参照することもできます。明示的なパスを指定したり、プロジェクト プロパティを使用して、パスの一部または全部を置き換えるプロパティを指定したりできます。

すべてのプロジェクト設定は、Windows ストア パッケージのすべての入力アセンブリに適用されます。また、パッケージ アセンブリの一覧に含まれるアセンブリに対し、対象除外規則を作成して Dotfuscator プロジェクトに保存できます。

すべての署名された Windows ストア パッケージは Dotfuscator によって再署名する必要があります。署名オプションは[**パッケージ プロパティ**]ボタンまたはショートカット メニューのエントリからアクセスできます。オプションには[**証明書ファイル**]、[**証明書パスワード**]、および[**タイムスタンプ サーバーの URL**]設定があります。APPX 証明書オプションとして、出力 APPX ファイルの署名に使用する証明書ファイルのコンテナー、任意で証明書パスワードおよびタイムスタンプ サーバーを設定します。

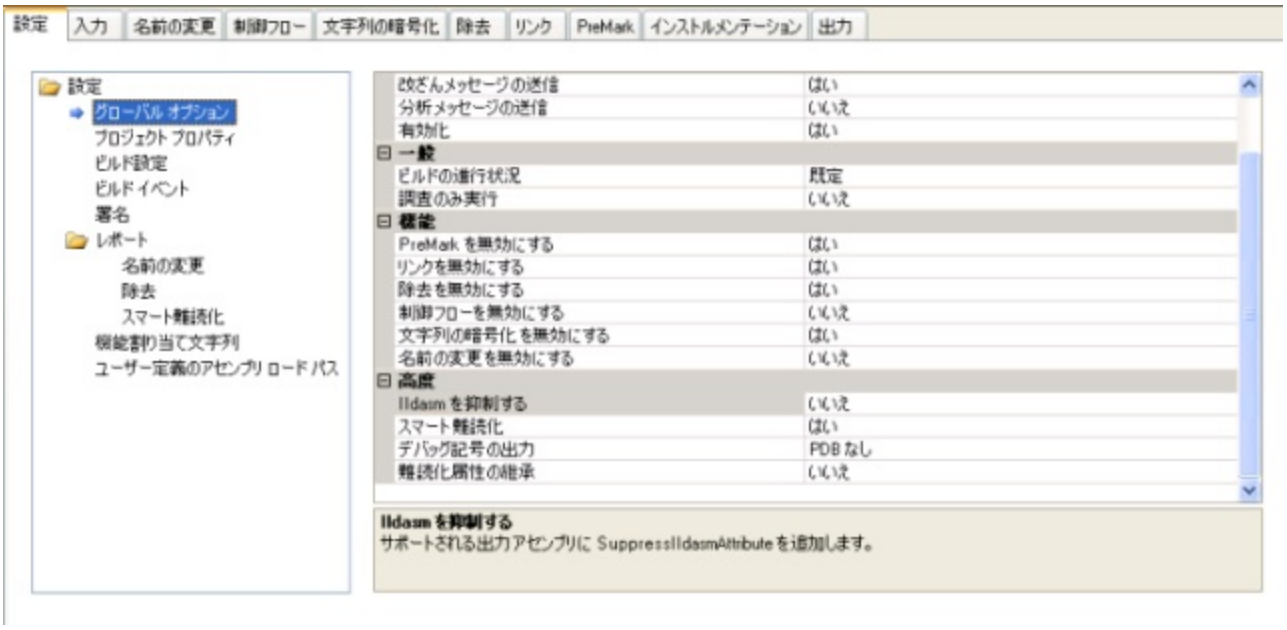
2.5.1.3 [設定]タブ

[**設定**]タブでは、[**グローバル オプション**]、[**プロジェクト プロパティ**]、[**ビルド設定**]、[**ビルド イベント**]、[**署名**]、[**レポート**]、[**機能割り当て文字列**](宣言による難読化用)、および[**ユーザー定義のアセンブリ ロード パス**]を設定できます。設定を行う対象の機能を選択するには、タブの左側に表示されるナビゲーション ペインで目的のノードを選択します。

グローバル オプション

[グローバル オプション]エディターを使用すると、プロジェクトのグローバル オプションを設定できます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



このタブでは、Dotfuscator の名前変更などの各機能を選択的に有効化または無効化することもできます。また、次のオプションを変更することもできます。

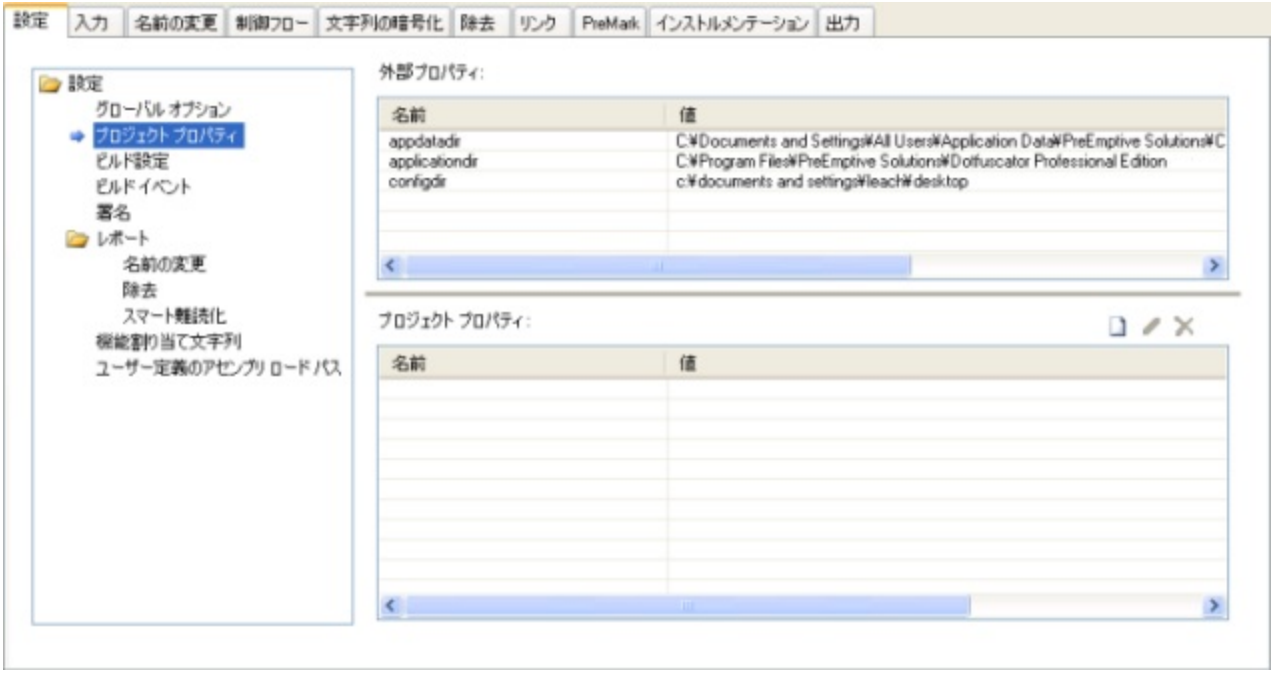
- [**デバッグ シンボルの出力**]。難読化されたアセンブリに対してデバッグ シンボルを生成し、JIT の動作を制御します。「Debug グローバル オプション」を参照してください。
- [**難読化属性の継承**]。このオプションは、型とメソッドに適用する宣言による難読化属性を、派生する型とオーバーライドするメソッドにも適用するかどうかを指定します。
- [**スマート難読化**]。このオプションでは、選択したアプリケーションの種類に対する名前の変更と除去の自動実行を有効または無効にすることができます。詳細については「スマート難読化」を参照してください。既定値では有効になっています。
- [**Ildasm を抑制する**]。これはすべての出力アセンブリへ SuppressIldasmAttribute を追加するよう Dotfuscator に指示します。これにより、Microsoft の Ildasm ユーティリティがアセンブリの IL を表示しないようになります。これは .NET 2.0 以上を対象とするアセンブリにのみ有効です。
- [**<機能>を無効にする**]。Dotfuscator では各変換処理を個別に有効または無効に設定できます。新規プロジェクトの既定値で無効になっている変換処理には、リンク、ウォーターマーク、除去、および文字列の暗号化があります。新規プロジェクトの既定値で有効になっている変換処理には、名前の変更、制御フロー、およびインストルメンテーションがあります。
- [**ビルドの進行状況**]。このオプションは、ビルドの過程で Dotfuscator が出力する情報の詳細度を制御します。
- [**調査のみ実行**]。このオプションは、Dotfuscator に対して、レポートのみを生成してアセンブリの出力を行わないように指示します。
- [**インストルメンテーション－有効化**]。このオプションでは、アプリケーションをインストルメントする機能を有効または無効にできます。既定値では、インストルメンテーションは有効になっています。
- [**ランタイムのマージ**]。Runtime Intelligence ライブラリを、別個の DLL として参照するのではなく出力アセンブリの 1 つに埋め込むことができます。既定値では、この動作は有効になっています。
- [**分析メッセージの送信**]。アプリケーション ライフサイクル、セッション ライフサイクル、および機能の使用に関するメッセージを Runtime Intelligence Service へ送信するよう、アプリケーションをインストルメントすることができます。「アプリケーション分析を備えた Dotfuscator の構成および実行」を参照してください。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

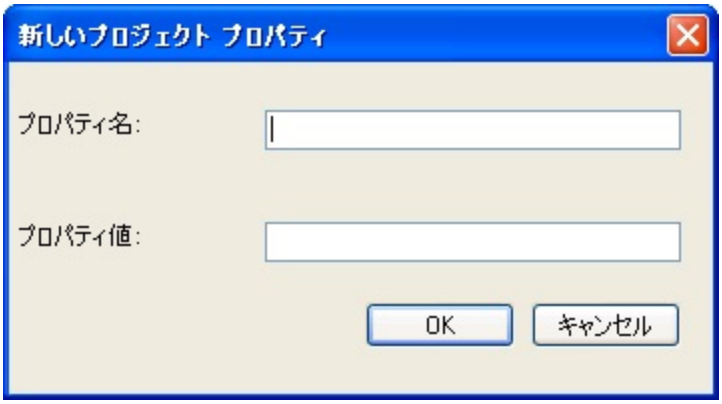
- [**Shelf Life メッセージの送信**]。Shelf Life（使用期限）警告、使用期限切れ、および Sign of Life に関するメッセージを Runtime Intelligence Service へ送信するよう、アプリケーションをインストルメントすることができます。「Shelf Life（使用期限）を備えた Dotfuscator の構成および実行」を参照してください。
- [**改ざんメッセージの送信**]。改ざん通知メッセージを Runtime Intelligence Service へ送信するよう、アプリケーションをインストルメントすることができます。「アプリケーション分析を備えた Dotfuscator の構成および実行」を参照してください。

プロジェクト プロパティ

[プロジェクト プロパティ]エディターでは、ユーザー定義の名前と値のペアを**プロジェクト プロパティ**として参照および追加することと、コマンド ラインで指定された**外部プロパティ**を参照することが可能です。詳細については、「プロパティ リストとプロパティ」を参照してください。プロジェクト プロパティを追加するには、プロジェクト プロパティ ツール バーの[**新規のプロパティを作成します**]ボタンをクリックします。



[**新しいプロジェクト プロパティ**]ダイアログが表示されます。**名前**と**値**を該当するボックスに入力してください。[**OK**]をクリックすると、指定したプロパティが設定されます。



© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

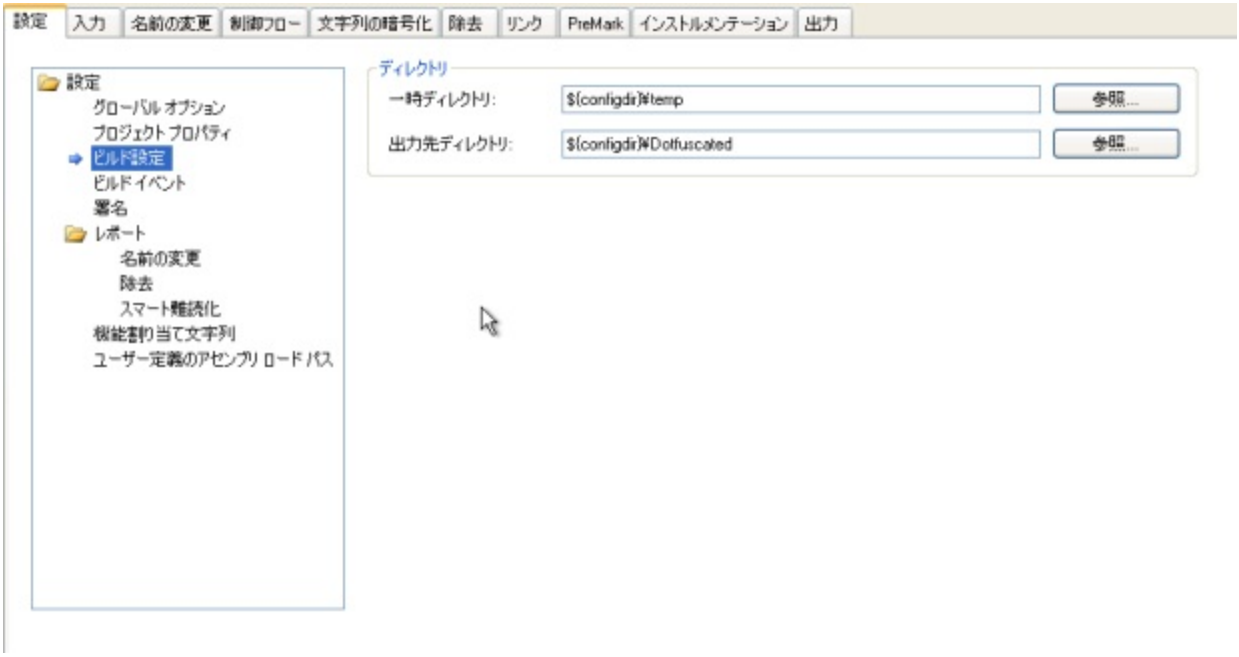
ツール バーの[**選択したプロパティを編集します**]ボタンおよび[**選択したプロパティを削除します**]ボタンを使用すると、同様の操作方法でプロジェクト プロパティを変更または削除できます。プロパティの削除は、削除対象のプロパティを選択して **Delete** キーを押す方法でも実行できます。

ビルドの設定

[ビルド設定]エディターでは、プロジェクトの出力先ディレクトリと任意で一時ディレクトリを指定できます。

新しいプロジェクトを作成した場合、出力先ディレクトリの既定値として **${configdir}\Dotfuscated** が設定されます。**${configdir}** は組み込みプロジェクト プロパティの 1 つで、プロジェクト ファイルの含まれるフォルダーに展開されます。

別の出力先ディレクトリを選択したい場合や一時ディレクトリを指定する場合は、適切なテキスト フィールドにそのディレクトリのパスを入力します。



[**一時ディレクトリ**]の指定は任意です。処理中に一時ファイルを格納するために使用されます。既定では、Dotfuscator は Windows の一時ディレクトリを使用します。このディレクトリを指定したい場合は、このフィールドにディレクトリの**パス**を入力するか、[**参照**]をクリックしてグラフィカルに場所を選択します。

[**出力先ディレクトリ**]は、ビルドからの出力が保存される場所であるため必須です。このフィールドにディレクトリの**パス**を入力するか、[**参照**]ボタンをクリックしてグラフィカルに場所を選択します。

ビルド イベント

[**ビルド イベント**]プロパティ ページでは、Dotfuscator プロジェクトのビルド イベントを指定できます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

各イベントでは、そのイベントの発生時に実行する外部プログラムを指定できます。指定プログラムが失敗した場合、Dotfuscator のビルドを中止(失敗)するかどうかを指定することもできます。

ビルド後のイベントでは、どの状態の場合にプログラムを実行するか(例:常時、ビルド成功時のみ、ビルド失敗時のみ)を指定できます。さらに、ビルド後のイベントをプロジェクトに対して 1 回だけ実行するか、出力モジュールごとに 1 回ずつ実行するかを指定できます。

署名

厳密な名前

[署名]エディターでは、厳密名のアセンブリの署名および再署名を自動的に行うよう Dotfuscator を構成することができます。詳細については、「厳密名のアセンブリを Dotfuscator で処理する」を参照してください。

厳密名が付けられたアセンブリはデジタル署名されています。これにより、署名後にアセンブリが変更されていないかどうかを実行時に確認できます。署名には SHA1 ハッシュが使用され、ハッシュは RSA 公開/秘密キー ペアの秘密キーで署名されています。署名と公開キーは、共にアセンブリのメタデータに埋め込まれています。

Dotfuscator はアセンブリを変更してしまうため、Dotfuscator を使ってアセンブリを処理した**後に**署名を行うことは不可欠です。

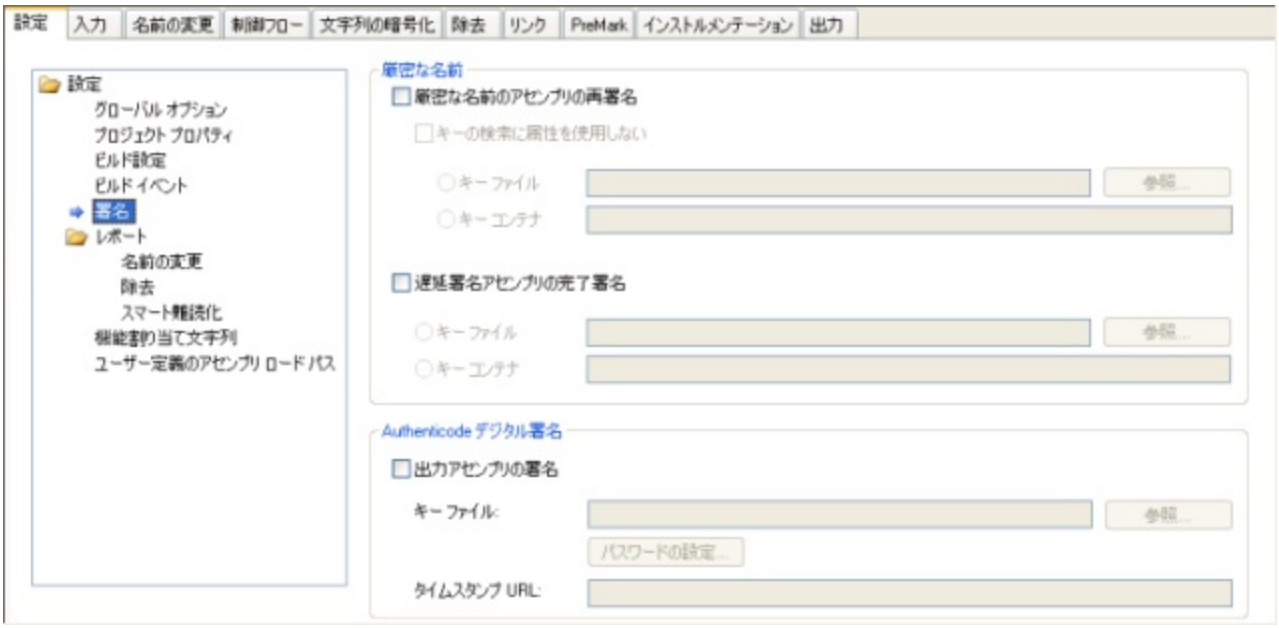
Dotfuscator はこのステップを難読化処理の一部として扱います。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Authenticode デジタル署名

Authenticode デジタル署名オプションを使用すると、アプリケーションに Authenticode デジタル署名を添付できます。セキュリティ証明書と同様、この署名は、あなたが難読化およびインストルメントを行っているアプリケーションがあなたの知的財産であり、結果として生成されるバイナリはあなたのみによって提供され変更されないことをユーザーに確認させることができます。この機能は、別のレベルのセキュリティを追加してアプリケーションを保護します。出力アセンブリに Authenticode 署名を添付するには、[**出力アセンブリの署名**]チェック ボックスをオンにし、[**参照**]ボタンをクリックして**キー ファイル** を見つけるか、テキスト ボックスにパスを入力します。[**キー ファイル**]フィールドに適切な値を入力したら、[**パスワードの設定**]ボタンをクリックしてキー ファイルのパスワードを設定します。

[タイムスタンプ URL]フィールドを使用すれば、Authenticode のタイムスタンプ サービスの URL を指定することができます。この URL は Dotfuscator の署名プロセス時にアクセスされ、そこから追加データが提供されます。その追加データを使用すれば、アセンブリの Authenticode 署名は、コード署名証明書の有効期限が切れた後も有効なままになります。この要素は省略可能です。ただし、省略した場合にはこの追加データは含まれないので、コード署名証明書の有効期限が切れるとアセンブリの Authenticode 署名は無効になります。



名前の変更レポート

名前の変更レポートは、ある特定の実行中に Dotfuscator によって名前が変更されたすべての要素の概要を提供します。[**名前の変更レポート ファイル(出力割り当てファイル)**]セクションでは、名前の変更割り当てファイルを保存する場所を指定できます。既定値のままにしておくか、パスを入力します。使用する割り当てファイル名およびパスがわかっている場合は、テキスト ボックスにファイル名とパスを直接入力できます。もう 1 つの方法として、ファイル システムを参照して、ファイルを出力する適切な場所を選択することもできます。テキスト ボックスの右にある[**参照**]ボタンをクリックすると、使い慣れたナビゲーション ツリーを持つ[名前の変更レポート(出力割り当てファイル)の選択]ウィンドウが表示されます。

また、既存の出力ファイルをバックアップすることなく、アプリケーションをビルドするたびにその出力ファイルを上書きするオプションもあります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Dotfuscator には、既定の .XML 形式のほかに、可読性の高い HTML 形式のレポートも生成できる、既定の変換処理があります。[**HTML として出力**]ボックスをオンにすると、[**カスタム変換(既定の場合は空白)**]フィールドが有効になります。フィールド名で示すように、カスタム変換に既定値を使用する場合は空白のままにし、そうしない場合は[**参照**]をクリックして場所を選択します。

除去レポート

除去レポートは、ある特定の実行中に Dotfuscator によって除去されたすべての要素の概要を提供します。[**除去レポート ファイル**]セクションでは、レポートを保存する場所を指定できます。ここでは 2 とおりの方法を使用できます。使用するファイル名およびパスがわかっている場合は、テキスト ボックスにファイル名とパスを直接入力できます。もう 1 つの方法として、ファイル システムを参照して、ファイルを出力する適切な場所を選択することもできます。テキスト ボックスの右にある[**参照**]ボタンをクリックすると、[**除去レポートファイルの選択**]ウィンドウが表示されます。このウィンドウにはファイルの場所を参照するためのナビゲーション ツリーが表示されます。

また、既存の出力割り当てファイルをバックアップすることなく、アプリケーションをビルドするたびにその出力ファイルを上書きするオプションもあります。除去レポートは、XML 形式のファイルのほかに可読性の高い HTML 形式のドキュメントとしても出力できます。このレポートは迅速な相互参照を提供しており、削除されたすべての型、フィールドおよびメソッドを即座に確認できるよう、項目間をすばやく移動できます。[**HTML として出力**]チェック ボックスをオンにすることにより、このレポートを XML 形式のレポートと同じ名前およびパス情報を使用して書き出すように指示できます。既定の HTML 形式レポートが、要求するレポート条件を満たしていない場合は、独自の XSL ドキュメントを提供することができます。Dotfuscator はそのドキュメントを使用して、XML 形式レポートの変換を行います。[**HTML として出力**]ボックスをオンにすると、[**カスタム変換(既定の場合は空白)**]フィールドが有効になります。フィールド名で示すように、カスタム変換に既定値を使用する場合は空白のままにし、そうしない場合は[**参照**]をクリックして場所を選択します。

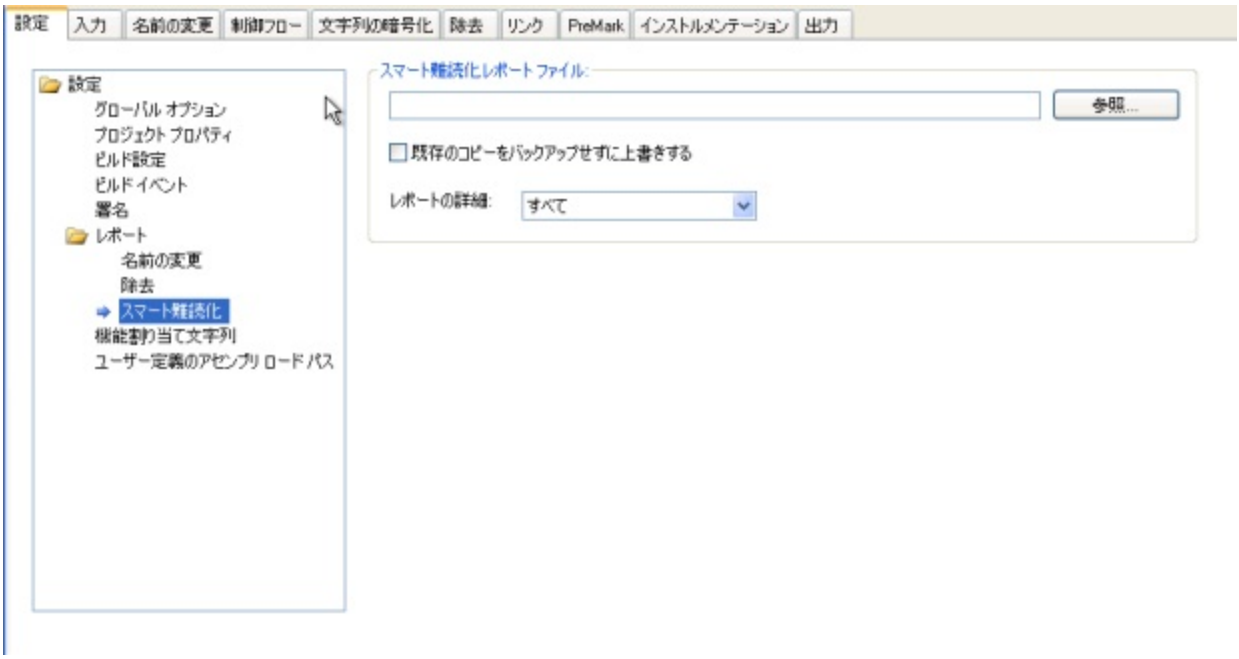
© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

スマート難読化レポート

スマート難読化レポートは、スマート難読化機能が指定する規則に基づいて名前の変更または除去ができなかったすべての要素の概要を提供します。スマート難読化によって項目が除外されたがレポートの出力先が指定されていない場合、レポートは[**スマート難読化レポート**]タブで Dotfuscator ウィンドウの下部にある[ビルド出力]タブの隣に表示されます。[**スマート難読化レポート ファイル**]フィールドでは、レポートを保存する場所を指定できます。ここでは 2 とおりの方法を使用できます。使用する割り当てファイル名およびパスがわかっている場合は、テキスト ボックスにファイル名とパスを直接入力できます。もう 1 つの方法として、ファイル システムを参照して、ファイルを出力する適切な場所を選択することもできます。テキスト ボックスの右にある[**参照**]ボタンをクリックすると[**スマート難読化レポート ファイルの選択**]ウィンドウが表示されます。このウィンドウには、ファイルの場所を参照するためのナビゲーション ツリーが表示されます。

既存の出力ファイルをバックアップすることなく、アプリケーションをビルドするたびにその出力ファイルを上書きするオプションもあります。また、任意でレポートの詳細を構成できます。レポートの詳細属性に使用できる値は、[**すべて**]、[**警告のみ**]および[**なし**]です。既定値は[**すべて**]です。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

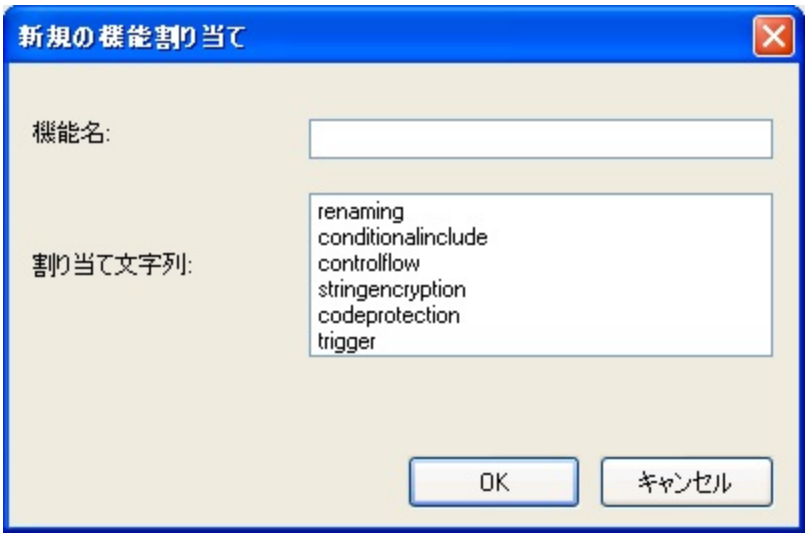


機能割り当て文字列

[機能割り当て文字列]エディターは、宣言による難読化を設定するために使用します。宣言による難読化は、ソース コードを記述する際に、ソース コード内で属性を修飾することによって実装されます。宣言による難読化の制御に使用する属性は、**System.Reflection.ObfuscateAssemblyAttribute** および **System.Reflection.ObufscationAttribute** です。System.Reflection.ObfuscateAssemblyAttribute は、アセンブリ全体としての難読化を制御します。System.Reflection.ObufscationAttribute は、個々の型とそのメンバーを対象とする難読化を制御します。[機能割り当て文字列]では、どの属性を使用して難読化する必要があるか、また難読化してはいけないのかをソース コード内で宣言することができます。Dotfuscator の「宣言による難読化」のサポートの詳細については、「カスタム属性を使用した宣言による難読化」を参照してください。このセクションでは、**機能割り当て**について説明し、Dotfuscator が認識するネイティブな機能文字列を紹介します。

ツール バーから、機能割り当て文字列の追加、編集、および削除を行えます。[**追加**]ボタンをクリックすると、Dotfuscator でサポートされている機能を機能文字列に割り当てるためのダイアログが表示されます。

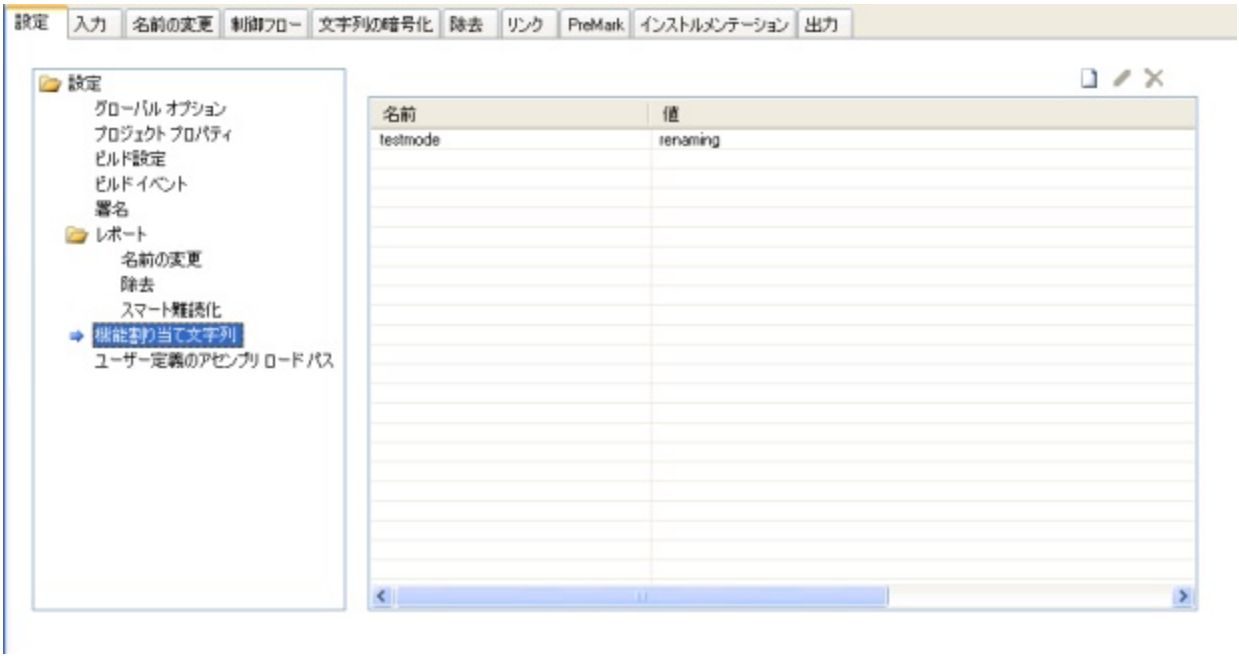
© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



[**機能名**]フィールドに入力した名前が文字列の名前です。選択された割り当て文字列は、その機能で修飾された構成が作成されることを意味します。

- **renaming** - この属性は名前の変更による難読化を設定します。
- **controlflow** - この属性は制御フローによる難読化を設定します。
- **stringencryption** - この属性は文字列の暗号化による難読化を設定します。
- **trigger** - この属性は、属性が指定された対象をエントリ ポイントとしてマークすることによる不要コードの除去を設定します。
- **conditionalinclude** - この属性は、属性が指定された対象を条件付きで選択することによる不要コードの除去を設定します。

機能を選択すると、**編集**および**削除**用ボタンが有効になります。[**難読化属性機能の割り当て文字列を編集します**]ボタンをクリックすると、Dotfuscator でサポートされている機能の文字列への割り当てを編集するためのダイアログが表示されます。機能割り当て文字列の削除は、削除対象の文字列を選択して **Delete** キーを押す方法でも実行できます。



© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ユーザー定義のアセンブリ ロード パス

このエディターを使用すると、プロジェクトでユーザーが定義したアセンブリ ロード パスを編集することができます。ツール バーから、ディレクトリの追加と削除、既存ディレクトリの編集、ディレクトリを検索する順序の変更が行えます。[**アセンブリ ロード パスを追加します**]ボタンをクリックすると、このウィンドウが表示されます。

アセンブリ ロード パスを指定するには 2 つの方法があります。使用する名前およびパスがわかっている場合は、テキスト ボックスに直接入力できます。もう 1 つの方法として、ファイル システムを参照して、ファイルを出力する適切な場所を選択することもできます。テキスト ボックスの右にある[**参照**]ボタンをクリックすると、[**フォルダーの参照**]ウィンドウが表示されます。このウィンドウには、ファイルの場所を参照するためのナビゲーション ツリーが表示されます。パスを指定したら[**OK**]をクリックします。このパスが[ユーザー定義アセンブリ ロード パス]ウィンドウに表示されます。

[**最初に検索**]チェック ボックスをオンにすると、Dotfuscator はその標準検索を適用する前にロード パスを検索します。チェックをオフにすると、Dotfuscator はその標準検索を適用した後にのみロード パスを検索します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.1.4 各機能の設定の構成

スタンドアロン ユーザー インターフェイスでは、各機能の設定を個別に構成することができます。各機能を構成するには、スタンドアロン ユーザー インターフェイスの個別のタブに表示される専用のエディターを使用します。個々のエディターの詳細については、以下の各トピックを参照してください。

- [名前の変更]エディター
- [制御フロー]エディター
- [文字列の暗号化]エディター
- [除去]エディター
- [リンク]エディター
- [PreMark]エディター
- [インストルメンテーション]エディター

2.5.1.5 プロジェクトのビルド

スタンドアロン GUI では、次の 2 とおりの方法で Dotfuscator のプロジェクトをビルドできます。

- ツール バーの[プロジェクトをビルドします]ボタンをクリックする。
- [ファイル]メニューの[**ビルド**]をクリックする。

ビルド中およびビルド後には、Dotfuscator の出力がコンソール画面に表示されます。

2.5.1.5.1 [出力]タブ

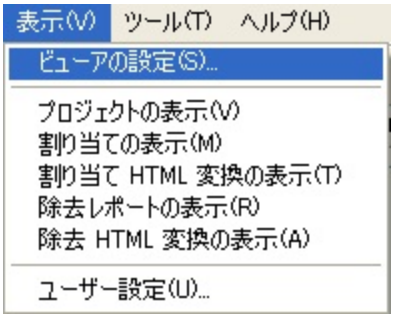
Dotfuscator によるプロジェクトの処理が完了したら、その結果を**出力**タブで確認できます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

このタブでは、ツリー ビューを参照して、Dotfuscator による型、メソッド、およびフィールドの名前変更の結果を閲覧できます。変更後の新しい名前は、ツリー内に元のノードの子ノードとして表示されます。

2.5.1.6 プロジェクト ファイルとレポートの表示

Dotfuscator では、選択した XML ビューアー ユーティリティに XML 構成ファイル、割り当てファイル、およびレポート ファイルを読み込むことによって、これらのファイルを直接参照することができます。[表示]メニューの[ビューアーの設定]をクリックして、使用するユーティリティを指定します。テキスト ファイルを表示する任意のアプリケーションをビューアー ユーティリティとして使用できます。



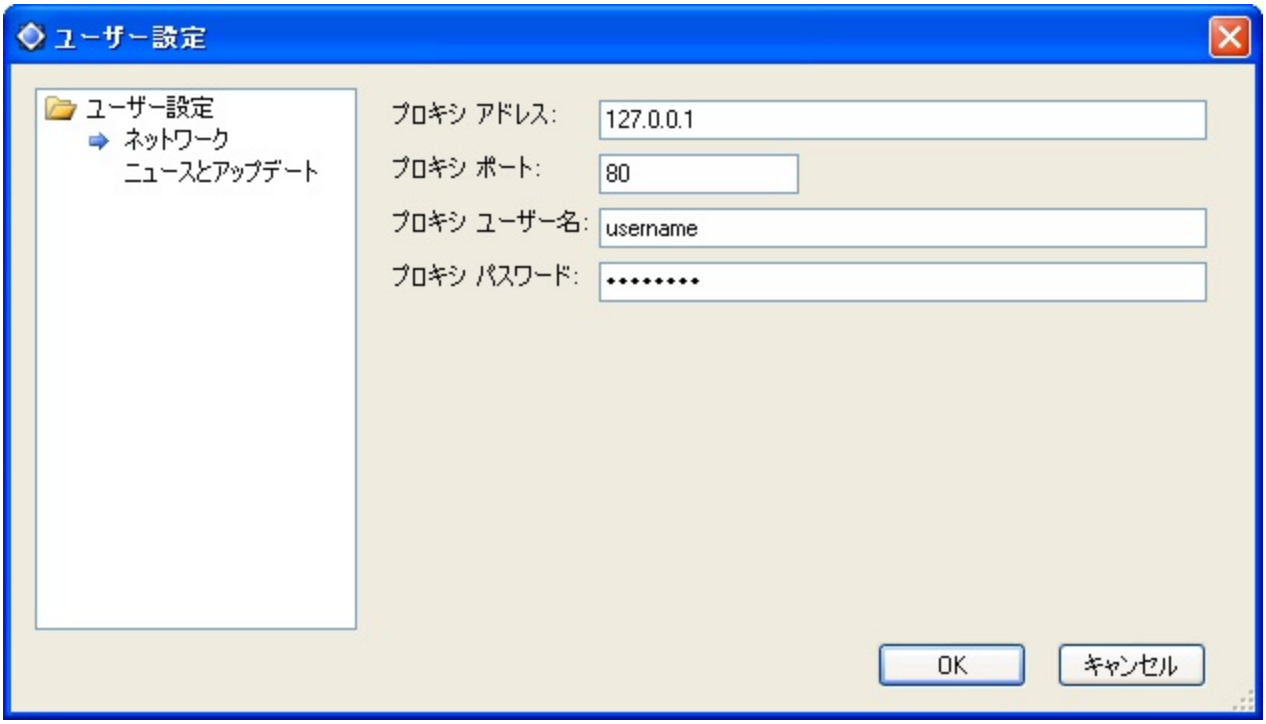
- プロジェクト構成ファイルを外部ビューアーに読み込むには、スタンドアロン ユーザー インターフェイスで[**表示**]メニューの[**プロジェクトの表示**]をクリックします。
- 割り当てファイルをビューアーに読み込むには、スタンドアロン ユーザー インターフェイスで[**表示**]メニューの[**割り当ての表示**]をクリックします。このメニュー オプションを使用すると、XML バージョンのレポートが表示されます。
- HTML 形式の割り当てファイルを生成するように選択した場合は、[**表示**]メニューの[**割り当て HTML 変換の表示**]をクリックすると、それを既定のブラウザーで表示できます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- 除去レポート ファイルをビューアーに読み込むには、ユーザー インターフェイスで[**表示**]メニューの[**除去レポートの表示**]をクリックします。このメニュー オプションを使用すると、XML バージョンのレポートが表示されます。
- HTML 形式の除去レポートを生成するように選択した場合は、[**表示**]メニューの[**除去 HTML 変換の表示**]をクリックすると、それを既定のブラウザーで表示できます。

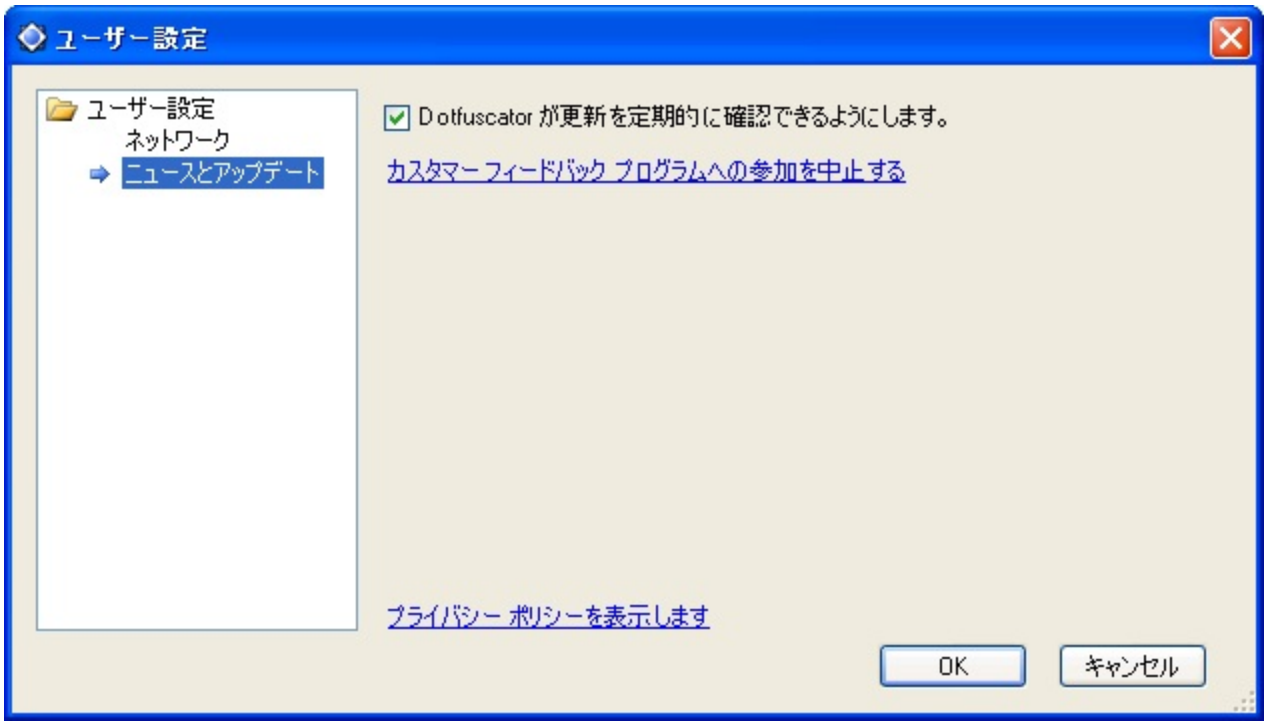
2.5.1.7 ユーザー設定の設定

[**表示**]メニューには[**ユーザー設定**]オプションがあります。このダイアログの[ネットワーク]セクションで、ネットワーク アクセスのプロキシ サーバーの構成設定を入力します。



このダイアログ ボックスの[**ニュースとアップデート**]セクションでは、Dotfuscator が更新を定期的に確認するように設定できます。このダイアログにはプライバシー ポリシーを表示するリンクが含まれています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



カスタマー フィードバック プログラムのダイアログを開くリンクもあります。このリンク内のテキストは現在の選択/非選択状態を表示します。プライバシーについて知りたい場合は、プライバシー ポリシーを表示しますリンクをクリックします。

2.5.1.8 [ヘルプ]メニュー

スタンドアロン GUI の[ヘルプ]メニューでは、以下のメニュー項目を選択できます。

- **ヘルプ**。このメニュー項目をクリックすると、Dotfuscator のオンライン ヘルプ(このドキュメント)が表示されます。
- **製品の登録**。このメニュー項目が有効になるのは、お使いの製品が未登録の場合のみです。この項目をクリックすると、Dotfuscator を登録するためのダイアログ ボックスが表示されます。詳細については「Dotfuscator の登録とアクティブ化」を参照してください。
- **Dotfuscator のアクティブ化**。この項目は、Dotfuscator のサブスクリプションがアクティブ化されていない、期限が切れている、または期限切れが近づいている場合にのみ有効になります。詳細については「Dotfuscator の登録とアクティブ化」を参照してください。
- **新機能**。このメニュー項目をクリックすると、ブラウザーが起動して Dotfuscator のホーム ページが表示されます。
- **カスタマー フィードバックのオプション**。Dotfuscator では、匿名による使用状況レポート システムを提供しています。このプログラムへ参加するかどうかはユーザーが選択できます。
- **更新情報を今すぐ確認**。このメニュー項目をクリックすると、Web を使用した Dotfuscator の更新チェックが直ちに実行されます。
- **バージョン情報**。このメニュー項目をクリックすると、Dotfuscator のバージョン情報ボックスが表示され、ユーザー情報とバージョン情報を確認できます。

2.5.2 Visual Studio インターフェイス

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

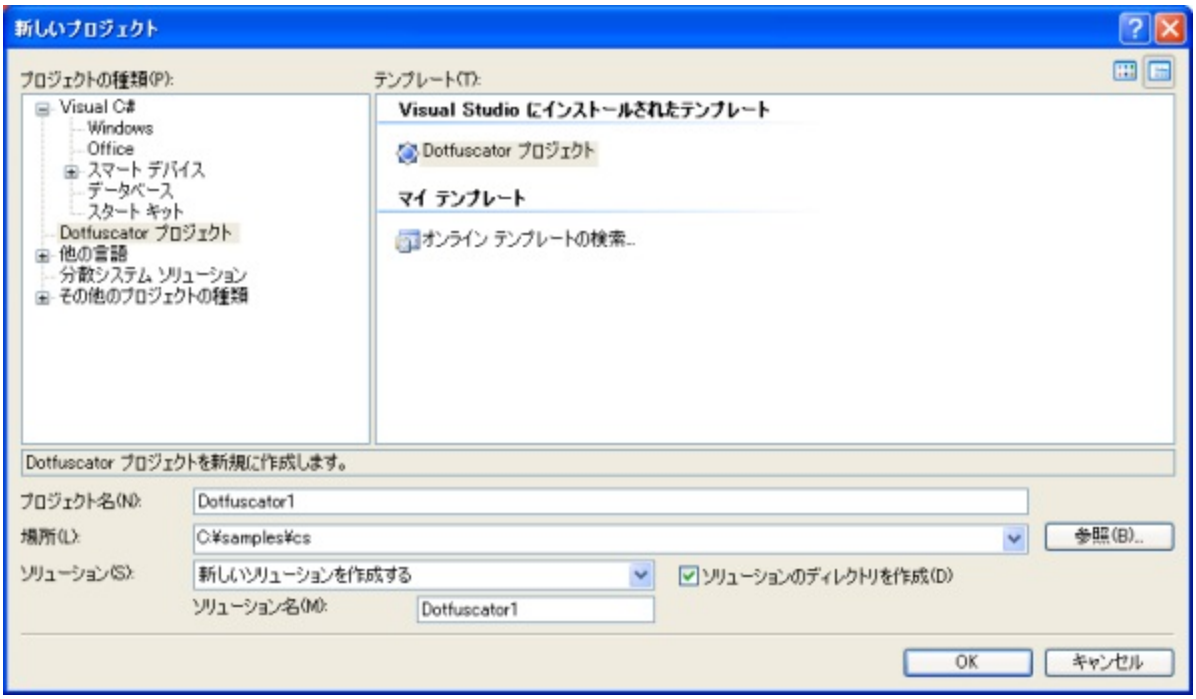
このセクションでは、Visual Studio に統合されている Dotfuscator の使用方法について説明します。Visual Studio の中で、Dotfuscator プロジェクトを作成および編集し、そのプロジェクトを既存のソリューションに追加することができます。ビルド過程において、Dotfuscator プロジェクトは、ソリューション内の開発プロジェクトで生成されたバイナリ出力（つまり、コンパイルされた .NET アセンブリ）を入力として受け取り、それに対する処理を実行した後、難読化処理された出力を配置プロジェクトに公開します。

DMA Visual Studio 内の Dotfuscator インターフェイスは Dotfuscator for Marketplace Apps では利用できません。

2.5.2.1 Dotfuscator プロジェクトの作成

Dotfuscator プロジェクトを作成し、それを既存または新規のソリューションに追加するには、Visual Studio プロジェクトのその他の種類の場合と同様の手順を実行します。

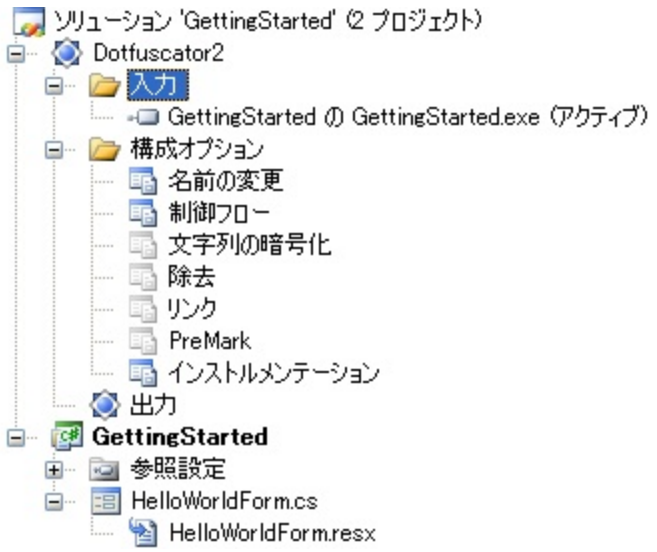
- [**新しいプロジェクト**]ダイアログ ボックスで、[**プロジェクトの種類**]の一覧から "**Dotfuscator プロジェクト**" フォルダーを開きます。
- [テンプレート]リストから "**Dotfuscator プロジェクト**" を選択します。
- プロジェクトの**名前**と**場所**を指定するか、または既定値を受け入れます。
- [**ソリューション**]フィールドで、ソリューションを新規作成するか、または既存のソリューションを追加するかを選択できます。
- [**ソリューション名**]にはソリューションの名前を指定します。
- 必要に応じて、[**ソリューションのディレクトリを作成**]および[**ソース管理に追加**]チェック ボックスをオンにして適用します。
- [**OK**]をクリックします。



2.5.2.2 ソリューション エクスプローラーおよび Dotfuscator プロジェクト ツリー

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Visual Studio で Dotfuscator プロジェクトの作業をする場合、**ソリューション エクスプローラー**が開始点となります。その他多くのプロジェクトの種類と同様、Dotfuscator プロジェクトは**ソリューション エクスプローラー**内にツリーとして現れます。



Dotfuscator プロジェクト ツリーの最上位のプロジェクト項目の名前は常に、Dotfuscator プロジェクトに指定した名前です。この項目のショートカット メニューから、プロジェクトのプロパティ ページへアクセスできます。Dotfuscator プロジェクトが表示するプロパティ ページの説明は、「プロジェクト プロパティ」を参照してください。

最上位の Dotfuscator プロジェクト項目の直下には、常に以下の 3 つの項目があります。

入力アセンブリフォルダー

ここで、Dotfuscator で処理する入力を管理します。この項目を右クリックすると、入力の追加が行えるショートカット メニューが表示されます。追加されたパッケージおよびアセンブリは、それぞれフォルダー内に項目として現れます。プロジェクトから入力を削除する場合は、その項目を選択し、**Delete** キーを押します。別の種類の入力については、「入力の設定」を参照してください。

構成オプション フォルダー

このフォルダーには、使用可能な構成エディターごとに 1 つの項目が含まれています。適切な項目をダブルクリックすると、それに該当するエディターが表示されます。

出力ブラウザー項目

Dotfuscator プロジェクトのビルド後、この項目をダブルクリックすると、出力アセンブリを参照できます。出力ブラウザーでは、クラス ブラウザーで提供されるビューに類似したアセンブリのビューが提供されます。元のシンボル名と難読化処理後のシンボル名に加え、どのシンボルが入力アセンブリから除去されたかが示されます。

2.5.2.3 プロジェクトの構成

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

入力とそのプロパティ以外の Visual Studio Dotfuscator プロジェクトの設定は、構成に依存します。つまり、現在アクティブな構成を基に、異なる難読化設定を持つことができるということです。たとえば、デバッグ ビルドではデバッグ シンボル ファイルを生成したいが、リリース ビルドでは生成したくないということがあります。

Dotfuscator プロジェクトの構成は、Visual Studio の構成マネージャーにより、他の種類の Visual Studio プロジェクトと同じ方法で管理されます。構成マネージャーを介して、Dotfuscator プロジェクトの構成を編集、作成、および削除することができます。また、プロジェクトをソリューション構成と関連付けることもできます。

既定値では、Dotfuscator プロジェクトを作成すると、2 つのプロジェクト構成が作成され、それぞれ対応するソリューション構成、"**デバッグ**" と "**リリース**" に関連付けられます。これら 2 つの構成は、デバッグ構成に[**デバッグ シンボルの出力**]を設定するためのプロジェクト プロパティがある点を除き、まったく同じです。

2.5.2.4 Dotfuscator プロジェクトの配置

Dotfuscator プロジェクトは、よく知られる C# や VB.NET などの他の Visual Studio プロジェクトの種類と同様、Visual Studio 内で出力を公開します。そのため、Visual Studio プロジェクトの使い方を理解している配置プロジェクトやセットアップ プロジェクトのソースとして、Dotfuscator プロジェクトを使用できます。このセクションは、ユーザーがセットアップ プロジェクトおよび配置プロジェクトに精通していることを前提にしています。

Dotfuscator は、次に示す出力グループを公開します。

出力グループ	説明
プライマリ出力	出力アセンブリおよびあらゆるパッケージ成果物が含まれる
ローカライズされたリソース DLL	サテライト DLL が含まれる
レポート	除去レポートおよび HTML レポート ファイルが含まれる
割り当てファイル	XML 形式の名前変更の割り当てファイルが含まれる
デバッグ シンボル	出力アセンブリがある場合は、出力アセンブリの PDB ファイルが含まれる

難読化処理されたアセンブリをすべてパッケージするようセットアップ プロジェクトに指示するには、単に、セットアップ プロジェクトで Dotfuscator プロジェクトの "プライマリ出力" グループをポイントするだけです。

アセンブリが他の Visual Studio プロジェクトから供給される場合、Dotfuscator はそのソース プロジェクトの依存関係も配置プロジェクトに公開します。

2.5.2.5 入力の設定

入力パッケージとアセンブリは、ソリューション エクスプローラー内の Dotfuscator プロジェクト ツリーにある**入力アセンブリ** フォルダーに表示されます。入力の追加は、最上位プロジェクト項目下の、"**入力**" フォルダー項目に関連付けられているショートカット メニューから、または Visual Studio の[プロジェクト]メニューから行えます。

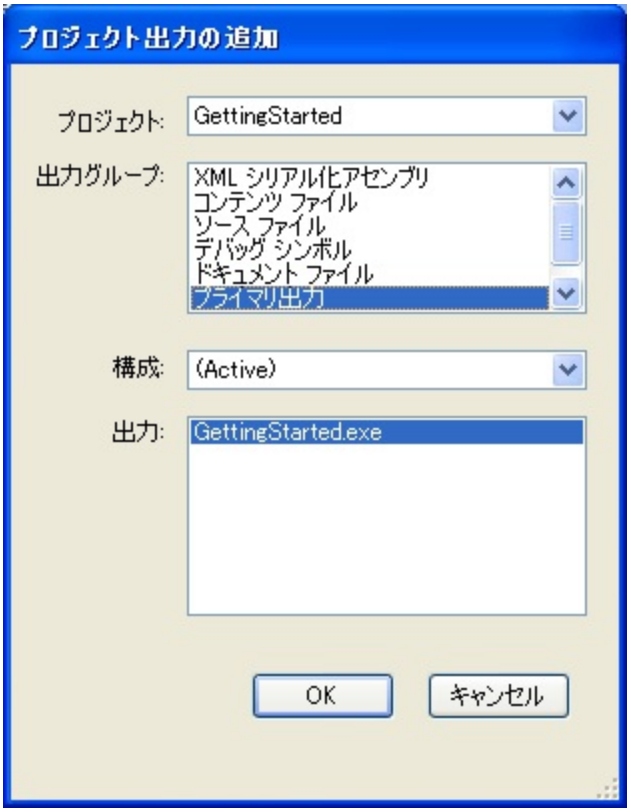
入力アセンブリの追加方法は 2 種類ありますが、これは、Dotfuscator で処理する対象をソリューション内の別の Visual Studio プロジェクトから出力されるものとするか、ファイル システム上のパッケージまたはアセンブリとするかによって決まります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

他プロジェクトからの入力アセンブリ

Dotfuscator に別プロジェクトの出力を使用させるには、Dotfuscator プロジェクトのショートカット メニューまたは Visual Studio の[プロジェクト]メニューにある[**プロジェクト出力の追加**]項目を使って、入力アセンブリを追加します。この項目を選択すると、Dotfuscator の[**プロジェクト出力の追加**]ダイアログ ボックスが表示されます。このダイアログ ボックスから、ソリューション内の他プロジェクトを参照できます。

Visual Studio プロジェクトの種類ごとに、他プロジェクトがアクセスできる出力グループのセットが定義されています。出力グループには、ソース プロジェクトによって作成されるファイルが含まれます。たとえば、下のスクリーンショットの場合、**GettingStarted C#** プロジェクトは**プライマリ出力**という出力グループを定義しており、そこに出力バイナリ(`GettingStarted.exe`)を置いています。



出力グループの内容は、プロジェクトの構成によって変わります。たとえば、**GettingStarted** プロジェクトはデバッグとリリースの両方の構成で `GettingStarted.exe` を作成しますが、ビルド実行時、これらのファイルは別のディレクトリへ書き込まれます。常に、構成を基に特定のグループを使用するか、あるいは、現在アクティブなプロジェクトの構成を基に出力グループを使用するかを Dotfuscator に指示できます。アクティブな構成が選択された場合、Dotfuscator はソース プロジェクトの構成が変わるたびに自動的に入力を更新します。

ファイル システムからの入力

ファイル システム上のパッケージまたは入力アセンブリを Dotfuscator に知らせる場合は、Dotfuscator プロジェクトのショートカット メニューまたは Visual Studio の[プロジェクト]メニューにある[**入力の追加**]項目を使って、入力を追加します。この項目を選択すると、1 つ以上の入力を選択できるファイルの参照ダイアログ ボックスが表示されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.2.6 プロジェクト プロパティ

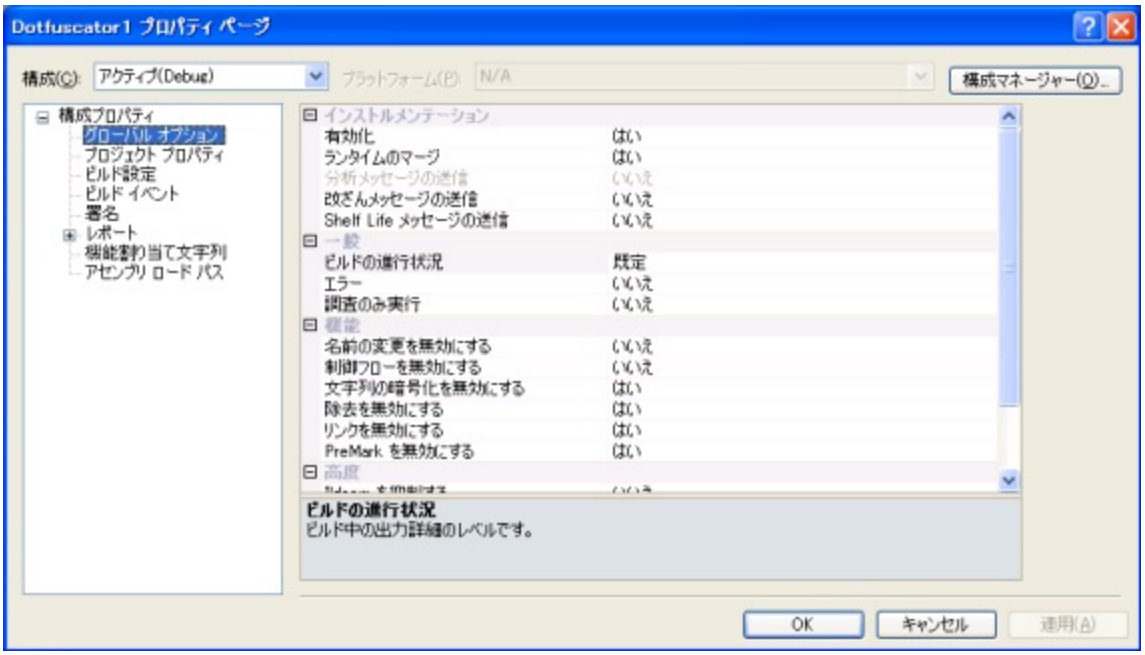
Visual Studio では、ソリューション エクスプローラーにある Dotfuscator プロジェクトの最上位項目からプロジェクトのプロパティページへアクセスできます。

構成プロパティ | グローバル オプション

[**グローバル オプション**]プロパティ ページでは、プロジェクトのグローバル オプションを設定できます。これらについては、「グローバル セクション」に詳しく説明されています。Dotfuscator の機能を選択的に有効にすることもできます。設定可能なプロパティの概要は次のとおりです。

- [**<機能>を無効にする**]。Dotfuscator では各変換処理を個別に有効または無効に設定できます。変換処理を無効にすると、ソリューション エクスプローラー内の該当する構成オプションのノードが淡色表示されます。変換処理を有効または無効にするには、このプロパティ シートを使用するか、もしくは目的のノードを右クリックし、[**無効化**]メニュー項目のチェックのオン/オフを切り替えて行います。新規プロジェクトの既定値で無効になっている変換処理には、リンク、ウォーターマーク、除去および文字列の暗号化があります。新規プロジェクトの既定値で有効になっている変換処理には、名前の変更、制御フローおよびインストルメンテーションがあります。
- [**ビルドの進行状況**]。このオプションは、ビルドの過程で Dotfuscator が出力する情報の詳細度を制御します。
- [**エラー**]。この設定は、診断目的のためにあります。これをオンにすると、ビルドの過程で一時ディレクトリに書き込まれる生成 IL が Dotfuscator によって削除されなくなります。
- [**スマート難読化**]。このオプションでは、名前の変更および除外について、選択したアプリケーションの種類における対象除外を自動的に適用するかどうか（有効または無効）を指定できます。詳細については「スマート難読化」を参照してください。既定値では有効になっています。
- [**調査のみ実行**]。このオプションは、Dotfuscator に対して、レポートのみを生成してアセンブリの出力を行わないように指示します。
- [**デバッグ シンボルの出力**]。難読化されたアセンブリに対してデバッグ シンボルを生成し、JIT の動作を制御します。「Debug グローバル オプション」を参照してください。
- [**Ildasm を抑制する**]。これはすべての出力アセンブリへ SuppressIldasmAttribute を追加するよう Dotfuscator に指示します。これにより、Microsoft の Ildasm ユーティリティがアセンブリの IL を表示しないようになります。これは .NET 2.0 以上を対象とするアセンブリにのみ有効です。
- [**インストルメンテーション**]。「アプリケーション分析を備えた Dotfuscator の構成および実行」を参照してください。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



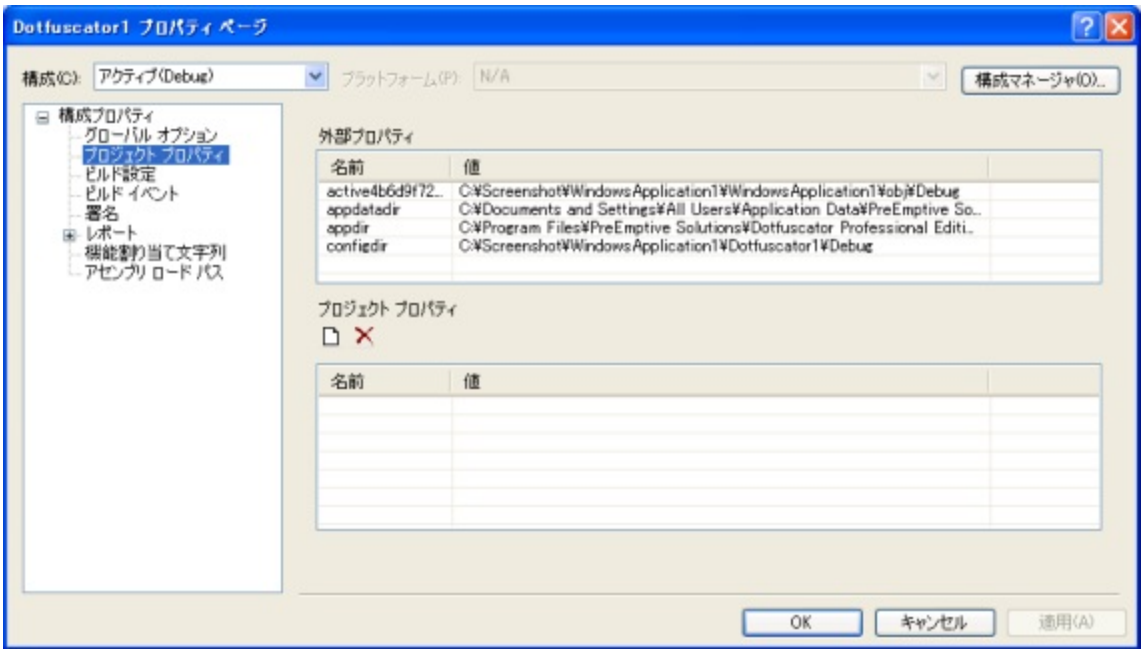
構成プロパティ | プロジェクト プロパティ

[プロジェクト プロパティ]ページには、ユーザー定義の名前と値のペアを**プロジェクト プロパティ**として追加、表示するための機能と、Dotfuscator 自体によって定義される**外部プロパティ**を表示するための機能が含まれています。

[**プロジェクト プロパティ**]は、ファイル名やパスが必要なあらゆるところで使用できる、単純な文字列の置換マクロと考えられます。詳細については、「プロパティ リストとプロパティ」を参照してください。

プロジェクト プロパティを追加するには、[**新規**]ボタンをクリックし、[**プロジェクト プロパティ**]グリッド内でプロパティの名前と値を編集します。1 つまたは複数の**プロジェクト プロパティ**を削除するには、[**プロジェクト プロパティ**]グリッド内で削除する行を選択し、[**削除**]ボタンをクリックします。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



構成プロパティ | ビルド設定

[**ビルド設定**]プロパティ ページでは、出力先ディレクトリと、任意で一時ディレクトリを指定できます。

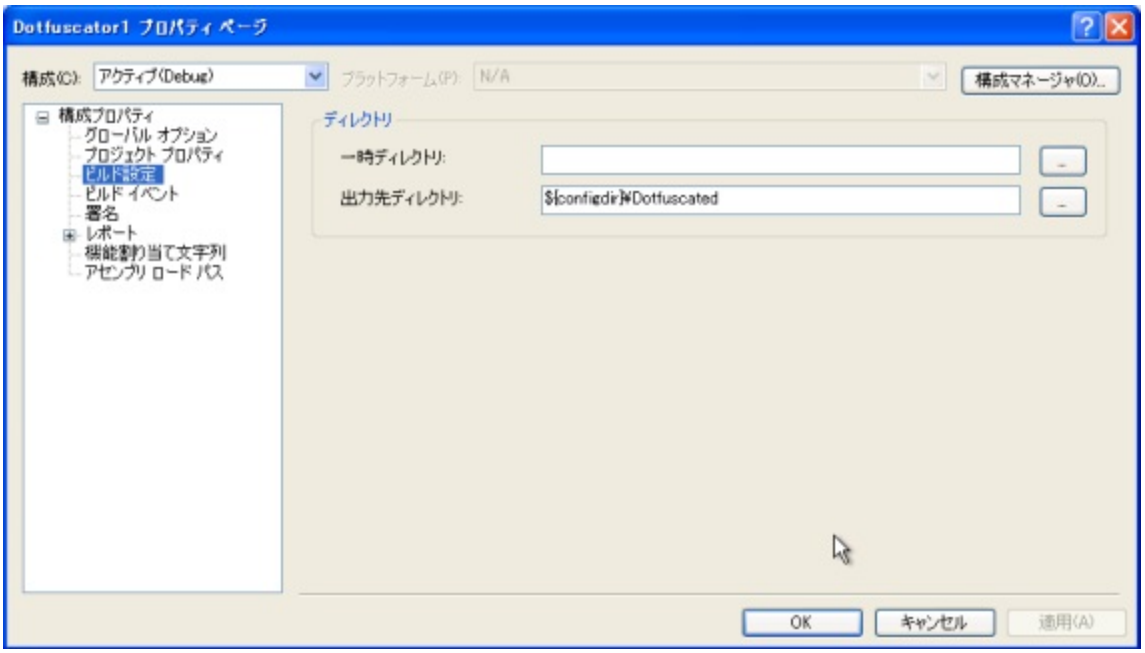
ビルド ディレクトリ

新しいプロジェクトを作成した場合、出力先ディレクトリの既定値として "`${configdir}\Dotfuscated`" が設定されます。`${configdir}` は組み込みプロパティの 1 つで、プロジェクト構成ファイルの保存先のフォルダーに展開されます。

別の出力先ディレクトリを選択する場合は、そのディレクトリを検索するか、[**出力先ディレクトリ**]テキスト ボックスに入力することができます。

任意で、処理中に一時ファイルを格納するために使用する一時ディレクトリを指定することもできます。指定しない場合は、Windows 環境で指定されている一時ディレクトリが使用されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



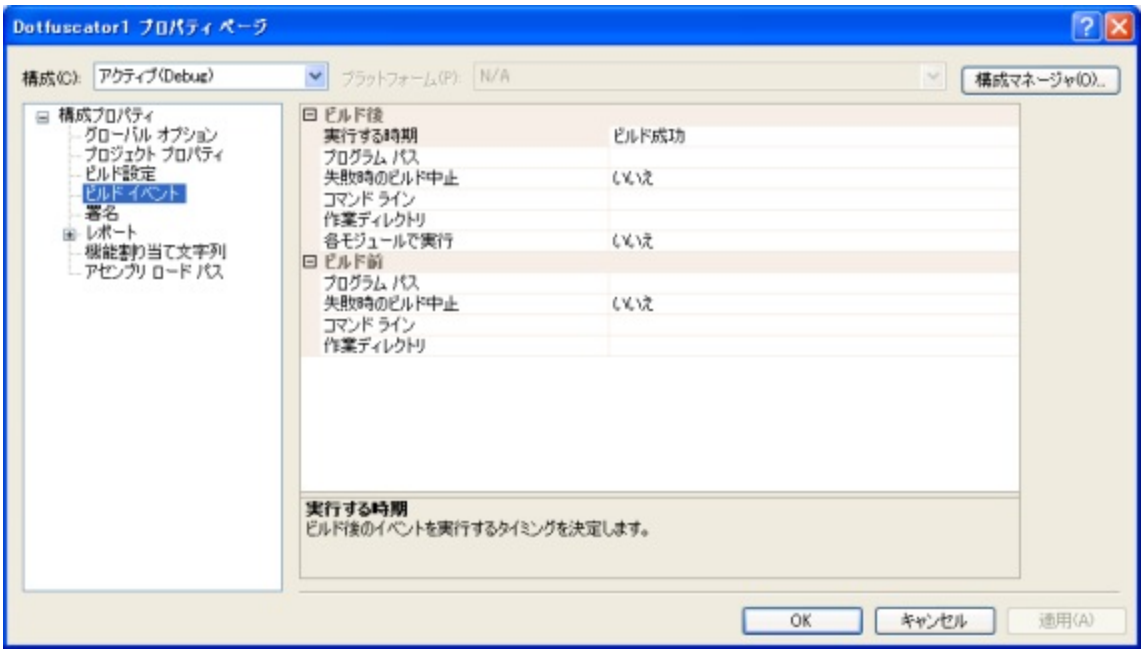
構成プロパティ | ビルド イベント

[**ビルド イベント**]プロパティ ページでは、Dotfuscator プロジェクトのビルド イベントを指定できます。各イベントでは、そのイベントの発生時に実行する外部プログラム(**プログラム パス**)を指定できます。そのプログラムの作業ディレクトリおよびコマンド ライン オプションを指定することも可能です。また、指定プログラムが失敗した場合、Dotfuscator のビルドを中止(失敗)するかどうかを指定することもできます。

ビルド後のイベントでは、どの状態の場合にプログラムを実行するか(例:常時、ビルド成功時のみ、ビルド失敗時のみ)を指定します。

さらに、ビルド後のイベントをプロジェクトに対して 1 回だけ実行するか、出力モジュールごとに 1 回ずつ実行するかを指定できます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



構成プロパティ | 署名

厳密な名前

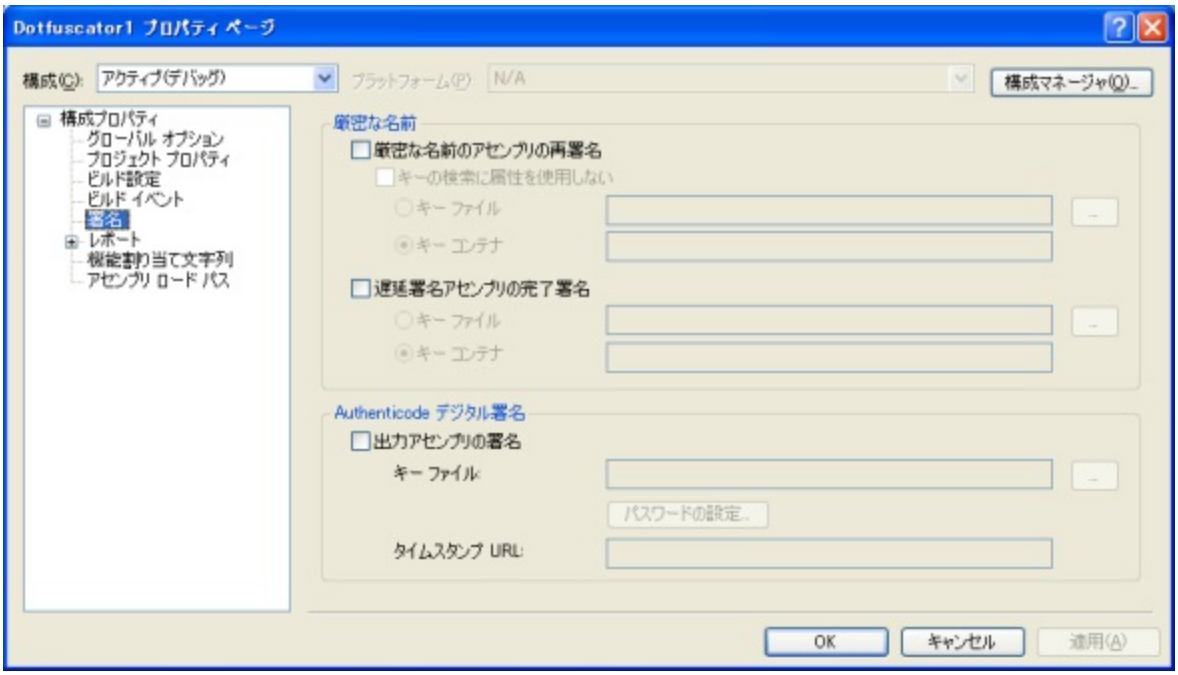
[**署名**]プロパティ ページでは、厳密名のアセンブリの署名および再署名を自動的に行うよう Dotfuscator を構成することもできます。詳細については、「厳密名のアセンブリを Dotfuscator で処理する」を参照してください。

Authenticode デジタル署名

Authenticode デジタル署名オプションを使用すると、アプリケーションに Authenticode デジタル署名を添付できます。セキュリティ証明書と同様、この署名は、あなたが難読化およびインストルメントを行っているアプリケーションがあなたの知的財産であり、結果として生成されるバイナリはあなたのみによって提供され変更されないことをユーザーに確認させることができます。この機能は、別のレベルのセキュリティを追加してアプリケーションを保護します。出力アセンブリに Authenticode 署名を添付するには、[**出力アセンブリの署名**]チェック ボックスをオンにし、[**...**]ボタンをクリックして**キー ファイル**を見つけるか、テキスト ボックスにパスを入力します。[**キー ファイル**]フィールドに適切な値を入力したら、[**パスワードの設定**]ボタンをクリックしてキー ファイルのパスワードを設定します。

[**タイムスタンプ URL**]フィールドを使用すれば、Authenticode のタイムスタンプ サービスの URL を指定することができます。この URL は Dotfuscator の署名プロセス時にアクセスされ、そこから追加データが提供されます。その追加データを使用すれば、アセンブリの Authenticode 署名は、コード署名証明書の有効期限が切れた後も有効なままになります。この要素は省略可能です。ただし、省略した場合にはこの追加データは含まれないので、コード署名証明書の有効期限が切れるとアセンブリの Authenticode 署名は無効になります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

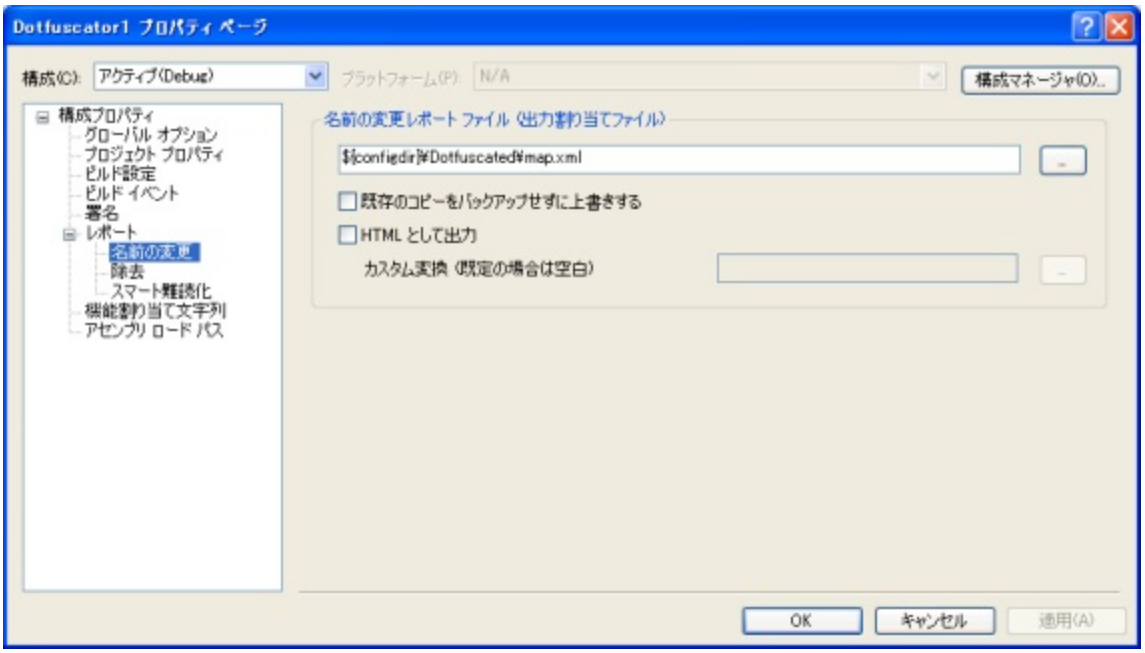


構成プロパティ | レポート | 名前の変更

名前の変更レポートは、ある特定の実行中に Dotfuscator によって名前が変更されたすべての要素の概要を提供します。[**名前の変更レポート ファイル(出力割り当てファイル)**]セクションでは、名前の変更割り当てファイルを保存する場所を指定できます。既定値のままにしておくか、パスを入力します。使用する割り当てファイル名およびパスがわかっている場合は、テキスト ボックスにファイル名とパスを直接入力できます。もう 1 つの方法として、ファイル システムを参照して、ファイルを出力する適切な場所を選択することもできます。テキスト ボックスの右にある[**...**]ボタンをクリックすると、使い慣れたナビゲーション ツリーを持つ[**出力割り当てファイルの選択**]ウィンドウが表示されます。

既存の出力ファイルをバックアップすることなく、アプリケーションをビルドするたびにその出力ファイルを上書きするオプションもあります。Dotfuscator には、既定の .XML 形式のほかに、可読性の高い HTML 形式のレポートも生成できる、既定の変換処理があります。[**HTML として出力**]ボックスをオンにすると、[**カスタム変換(既定の場合は空白)**]フィールドが有効になります。フィールド名で示すように、カスタム変換に既定値を使用する場合は空白のままにし、そうしない場合は "**...**" をクリックして場所を選択します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

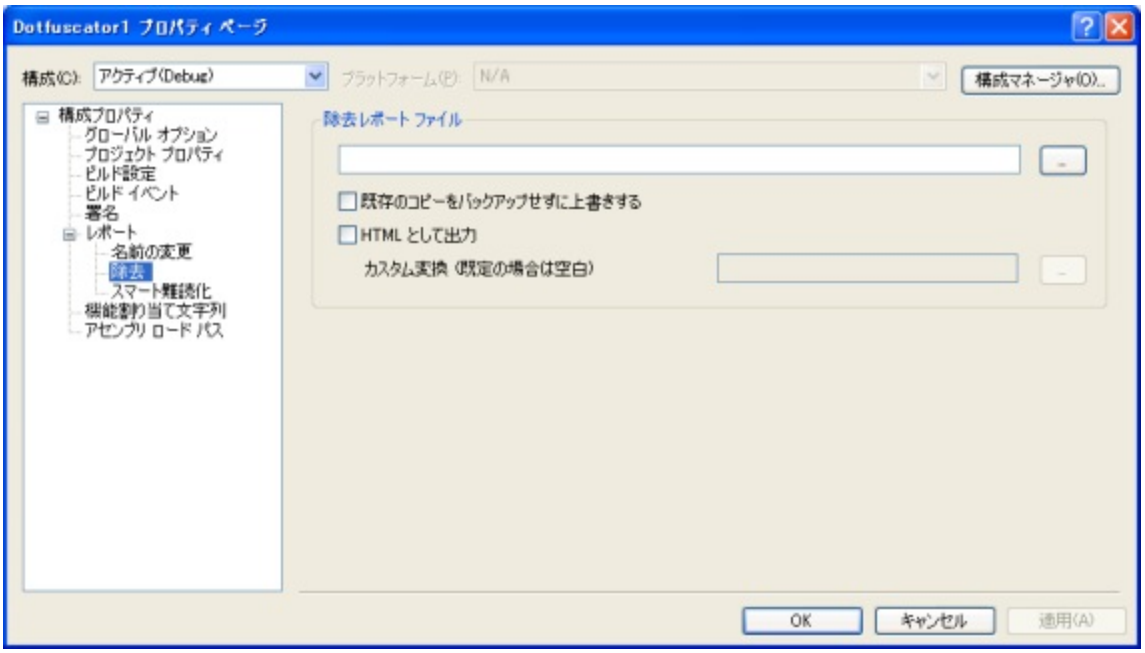


構成プロパティ | レポート | 除去

除去レポートは、ある特定の実行中に Dotfuscator によって除去されたすべての要素の概要を提供します。[**除去レポート ファイル**]セクションでは、レポートを保存する場所を指定できます。ここでは 2 とおりの方法を使用できます。使用するファイル名およびパスがわかっている場合は、テキスト ボックスにファイル名とパスを直接入力できます。もう 1 つの方法として、ファイル システムを参照して、ファイルを出力する適切な場所を選択することもできます。テキスト ボックスの右にある[**...**]ボタンをクリックすると、[**除去レポートファイルの選択**]ウィンドウが表示されます。このウィンドウにはファイルの場所を参照するためのナビゲーション ツリーが表示されます。

既存の出力ファイルをバックアップすることなく、アプリケーションをビルドするたびにその出力ファイルを上書きするオプションもあります。除去レポートは、XML 形式のファイルのほかに可読性の高い HTML 形式のドキュメントとしても出力できます。このレポートは迅速な相互参照を提供しており、削除されたすべての型、フィールドおよびメソッドを即座に確認できるよう、項目間をすばやく移動できます。[**HTML として出力**]チェック ボックスをオンにすることにより、このレポートを XML 形式のレポートと同じ名前およびパス情報を使用して書き出すように指示できます。既定の HTML 形式レポートが要求するレポート条件を満たしていない場合は、独自の XSL ドキュメントを提供することができます。Dotfuscator はそのドキュメントを使用して、XML 形式レポートの変換を行います。[**HTML として出力**]ボックスをオンにすると、[**カスタム変換(既定の場合は空白)**]フィールドが有効になります。フィールド名で示すように、カスタム変換に既定値を使用する場合は空白のままにし、そうしない場合は "**...**" をクリックして場所を選択します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

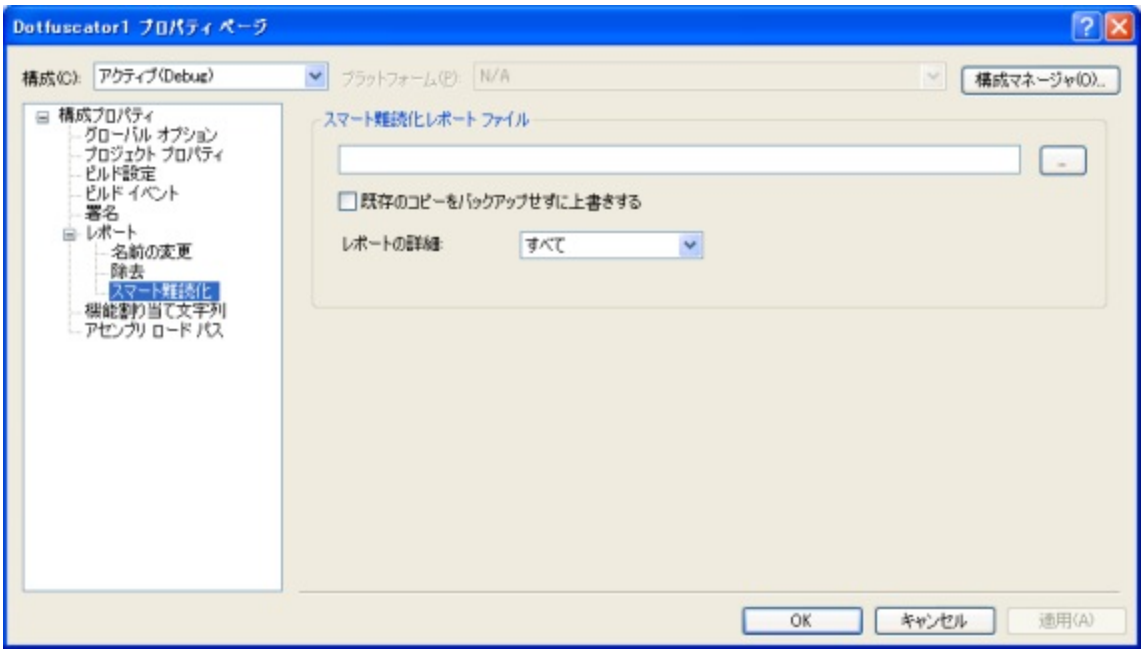


構成プロパティ | レポート | スマート難読化

スマート難読化レポートは、スマート難読化機能が指定する規則に基づいて名前の変更または除去ができなかったすべての要素の概要を提供します。スマート難読化によって項目が除外されたがレポートの出力先が指定されていない場合、レポートは Visual Studio ウィンドウの下部にある[出力]タブの[**スマート難読化レポート**]ビューに表示されます。[**スマート難読化レポート ファイル**]フィールドでは、レポートを保存する場所を指定できます。ここでは 2 とおりの方法を使用できます。使用する割り当てファイル名およびパスがわかっている場合は、テキスト ボックスにファイル名とパスを直接入力できます。もう 1 つの方法として、ファイル システムを参照して、ファイルを出力する適切な場所を選択することもできます。テキスト ボックスの右にある[**...**]ボタンをクリックすると、[**スマート難読化レポート ファイルの選択**]ウィンドウが表示されます。このウィンドウにはファイルの場所を参照するためのナビゲーション ツリーが表示されます。

既存の出力ファイルをバックアップすることなく、アプリケーションをビルドするたびにその出力ファイルを上書きするオプションもあります。また、任意でレポートの詳細を構成できます。レポートの詳細属性に使用できる値は、[**すべて**]、[**警告のみ**]および[**なし**]です。既定値は[**すべて**]です。

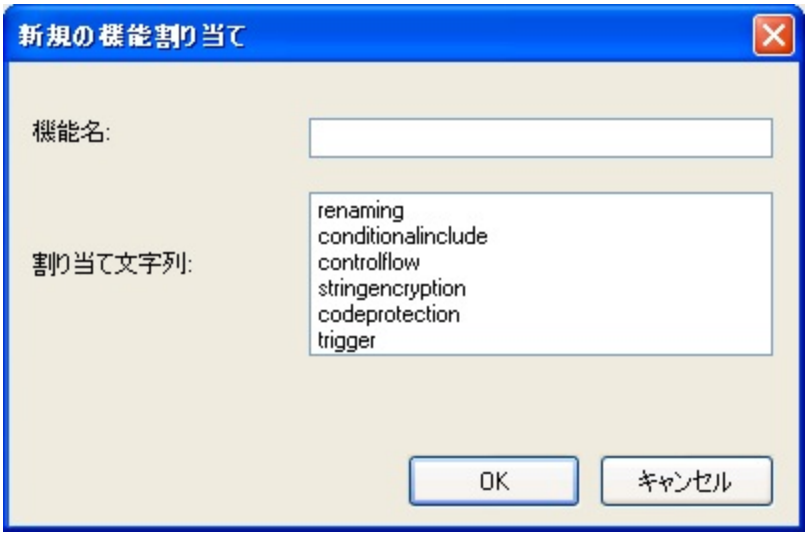
© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



構成プロパティ | 機能割り当て文字列

機能割り当て文字列のプロパティ ページは、宣言による難読化のためにあります。Dotfuscator の「宣言による難読化」のサポートの詳細については、「カスタム属性を使用した宣言による難読化」を参照してください。このセクションでは、**機能割り当て**について説明し、Dotfuscator が認識するネイティブな機能文字列を紹介します。

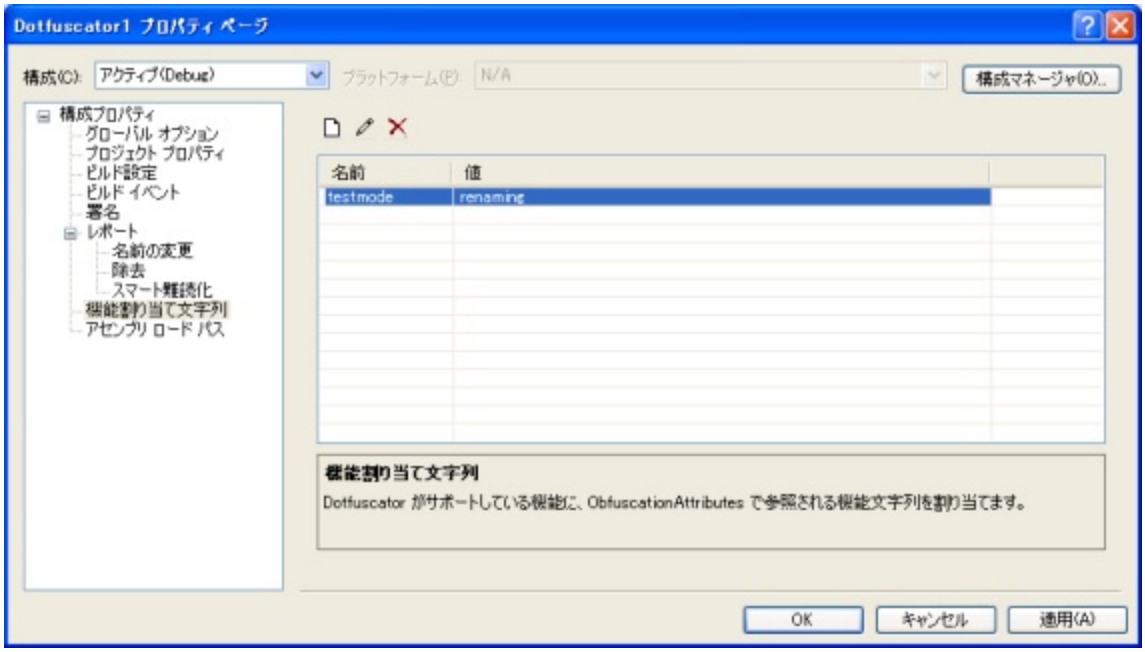
ツール バーから、機能割り当て文字列の追加、編集、および削除を行えます。[**追加**]および[**編集**]ボタンをクリックすると、Dotfuscator でサポートされている機能を機能文字列に割り当てるためのダイアログが表示されます。



[追加]アイコンをクリックすると、[新規の機能割り当て]ダイアログ ボックスが表示されます。機能名を入力し、その機能の適切な割り当て文字列を選択します。複数の割り当て文字列を選択するには、Ctrl キーを押したままの状態で選択対象の割り当て文字列をクリックしていきます。選択が完了したら[OK]をクリックします。

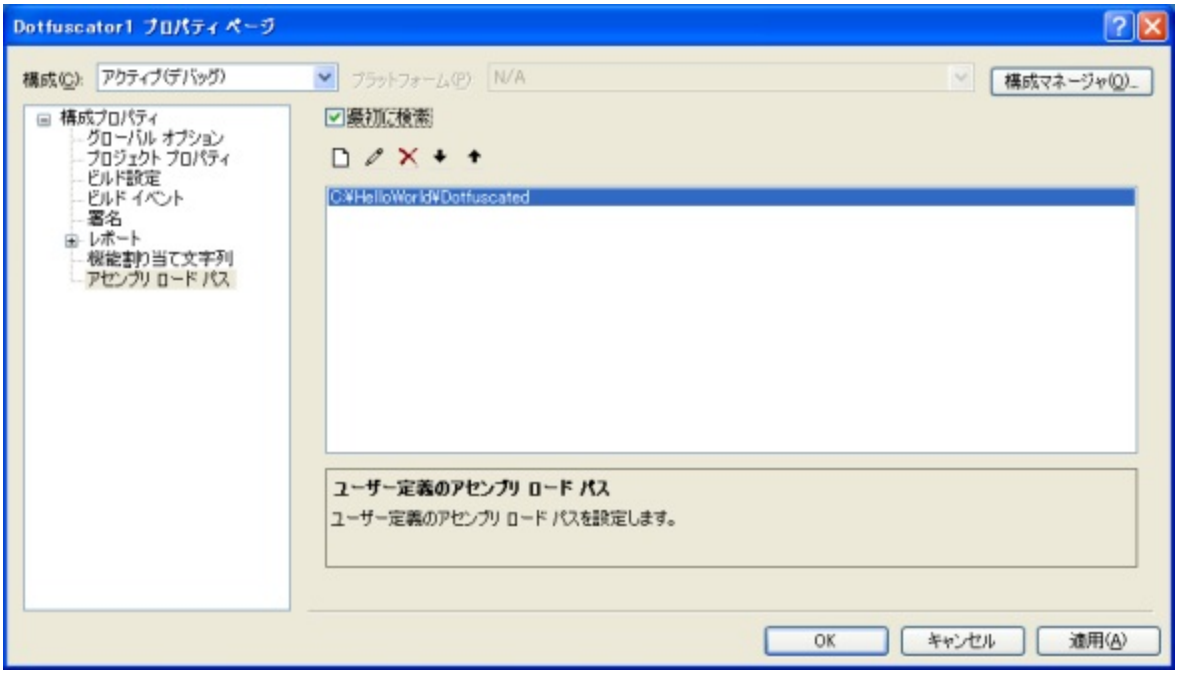
© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

新しい機能とその文字列が[機能割り当て文字列]プロパティ ページに表示されます。



構成プロパティ | アセンブリ ロード パス

このプロパティ ページを使用すると、プロジェクトでユーザーが定義したアセンブリ ロード パスを編集することができます。ツールバーから、ディレクトリの追加と削除、既存ディレクトリの編集、ディレクトリを検索する順序の変更が行えます。[**最初に検索**] チェック ボックスをオンにすると、Dotfuscator はその標準検索を適用する前にロード パスを検索します。チェックをオフにすると、Dotfuscator はその標準検索を適用した後にのみロード パスを検索します。



© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

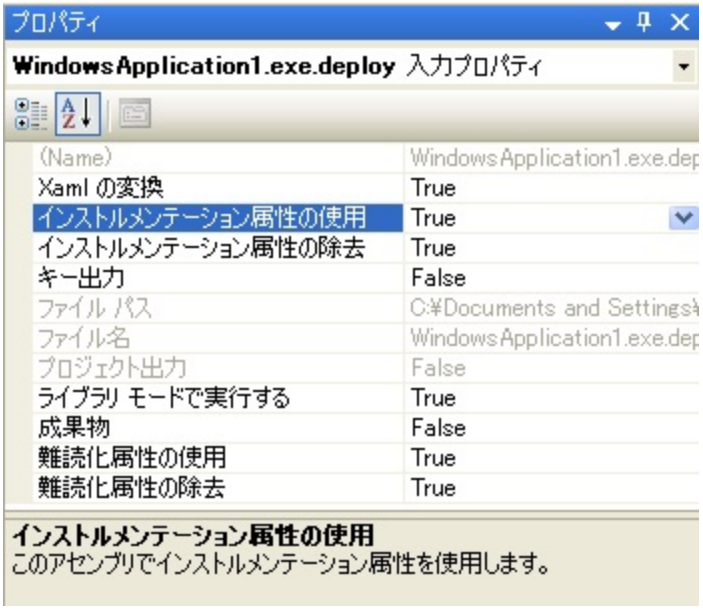
2.5.2.7 入力アセンブリ プロパティ

Visual Studio では、各入力アセンブリがプロパティ ウィンドウにプロパティを公開します。入力アセンブリのプロパティ ウィンドウを表示するには、ソリューション エクスプローラー内で Dotfuscator プロジェクト下にある入力ノードを選択してから、Visual Studio のプロパティ ウィンドウを（[表示]メニューから、または F4 キーを押して）起動します。複数の入力アセンブリを選択して、それらのプロパティを同時に管理することができます。プロパティ ウィンドウに公開されているプロパティは、選択しているアセンブリのグループに適用されます。

ここから、各アセンブリに対し、ライブラリ モードの設定、XAML の変換モードの設定、難読化するかパッケージ成果物のままにしておくかの指示が行えます。また、難読化属性の使用プロパティと難読化属性の除去プロパティを使って、宣言による難読化を構成することもできます。

また、選択アセンブリのインストルメンテーション オプションを設定することもできます。具体的には、選択しているアセンブリに対して Dotfuscator がインストルメンテーション属性を使用するかどうか、またこれらの属性を出力アセンブリから除去するかどうかを選択できます。詳細については、「アプリケーション分析を備えた Dotfuscator の構成および実行」を参照してください。

そのほかに、入力アセンブリのファイルに関する情報を表示する読み取り専用プロパティもあります。



プロパティ パネルの **Dotfuscator** セクションでは、次の属性の **True** または **False** の設定が行えます。

成果物: [**成果物**]を **True** に設定すると、指定されたアセンブリを入力として処理しないよう Dotfuscator に指示できます。そのアセンブリは難読化やインストルメント化が行われず、既存の署名はすべて保持されます。[**成果物**]を **False** に設定すると、Dotfuscator はそのアセンブリを入力として処理し、難読化およびインストルメント化を行います。

インストルメンテーション属性の使用: [**インストルメンテーション属性の使用**]を **True** に設定すると、これらの属性を処理し、示されているインストルメンテーションの変換をターゲット アセンブリで実行するよう Dotfuscator に指示できます。このオプションを **False** に設定すると、インストルメンテーション属性をすべて無視するよう Dotfuscator に指示できます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

> インストルメンテーション属性は、ソース コードに適用できるカスタム属性で、アプリケーションの安定性、機能、および使用状況を追跡したり、機能の使用期限を追加できます。

難読化属性の使用:[**難読化属性の使用**]の既定の設定は **True** です。この設定により、これらの属性を処理し、示されている難読化の変換をターゲット アセンブリで実行するよう Dotfuscator に指示できます。このオプションを **False** に設定すると、難読化属性をすべて無視するよう Dotfuscator に指示できます。

> 難読化属性はソース コードに適用できるカスタム属性で、さまざまな種類の難読化から対象選択または対象除外する型、メソッド、列挙、インターフェイス、およびメンバーを明示的に宣言できます。型、メソッド、列挙、インターフェイス、およびメンバーの対象選択または対象除外に使用する属性は、**System.Reflection.ObfuscationAttribute** です。特定のアセンブリに含まれている項目が難読化から対象選択または対象除外されることを示す場合は、**System.Reflection.ObfuscateAssemblyAttribute** を使用します。

キー出力:この設定によって、Dotfuscator プロジェクトの入力アセンブリの 1 つをキー出力アセンブリにするよう明示的に設定できます。キー出力は配置プロジェクトで使用されます。アセンブリのリンクが有効になっている場合、リンク元のアセンブリの 1 つがキー出力とマークされていたら、そのリンク先の出力アセンブリがキー出力になります。キー出力としてマークされた入力アセンブリがない場合は、Dotfuscator が 1 つを選択します。

このオプションの選択を解除すると、個々の属性が StripAfterObfuscation プロパティによって削除するよう指定されている場合を除き、出力アセンブリにこれらの属性を残すよう Dotfuscator に指示できます。

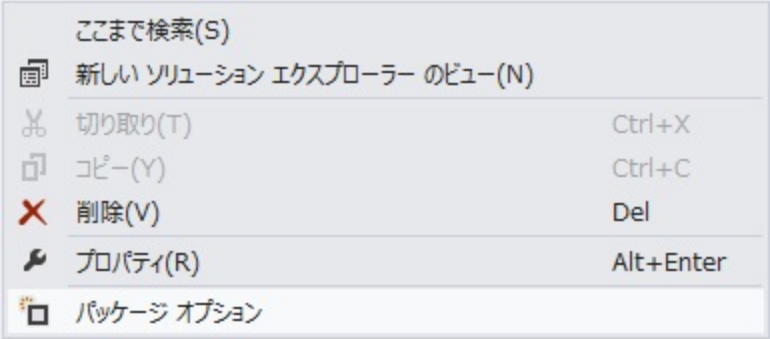
難読化属性の除去:Dotfuscator では、処理の完了後に難読化に関する属性をすべて除去して、難読化の処理方法を調べる手掛かりを出力アセンブリに残さないようにできます。このオプションを選択すると、ターゲットの出力アセンブリからこれらの属性を削除するよう Dotfuscator に指示できます。このオプションの選択を解除すると、個々の属性が StripAfterObfuscation プロパティによって削除するよう指定されている場合を除き、出力アセンブリにこれらの属性を残すよう Dotfuscator に指示できます。

XAML の変換:Dotfuscator は、Silverlight アプリケーションで使用される XAML リソース内の項目や Windows Presentation Foundation アプリケーションのコンパイル済み XAML リソース(BAML)内の項目の名前を変更することができます。既定値は True です。これは、マークアップ リソース内の項目の名前を変更し、その名前の変更と分離コード内の項目参照を一致させるよう Dotfuscator に指示します。このオプションを有効にしておくと、Windows Presentation Foundation および Silverlight アプリケーションの難読化が大いに強化されることに加え、名前の変更対象から手動で除外する必要がある項目数が減少します。

2.5.2.8 入力パッケージ プロパティ

Visual Studio では、一部のパッケージ入力がプロパティ ウィンドウにプロパティを公開します。入力のプロパティ ウィンドウを表示するには、ソリューション エクスプローラー内で Dotfuscator プロジェクト下にあるパッケージ ノードを選択して右クリックし、[**パッケージ オプション**]メニュー項目を選択します。そうすると、パッケージ固有のプロパティ ウィンドウが起動します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.5.2.9 各機能の設定の構成

Visual Studio Dotfuscator プロジェクトでは、各機能の設定を個別に構成することができます。機能ごとにエディターが用意されており、ソリューション エクスプローラーで展開されている Dotfuscator プロジェクト ツリーから適切な項目をダブルクリックすることによって、そのエディターを起動できます。各エディターについては、次のトピックに説明されています。

- [名前の変更]エディター
- [制御フロー]エディター
- [文字列の暗号化]エディター
- [除去]エディター
- [リンク]エディター
- [PreMark]エディター
- [インストルメンテーション]エディター

2.5.2.10 プロジェクトのビルド

Visual Studio Dotfuscator プロジェクトは他の Visual Studio プロジェクトの種類と同様に、プロジェクト レベルでビルドすることができます。既定値では、プロジェクトは、そのプロジェクトを含んでいるソリューションがビルドされるときにビルドされます。特定のソリューション構成について Dotfuscator プロジェクトをビルドしない場合は、Visual Studio の構成マネージャーを使用してビルドをオフにすることができます。

Dotfuscator プロジェクトのビルド時、入力にはアクティブな構成の設定が適用されます。出力パッケージおよび出力アセンブリは、プロジェクトの[**ビルド**]プロパティ ページで指定したディレクトリに書き込まれます。

ビルド中、Dotfuscator の出力は Visual Studio の出力ウィンドウに書き出され、ビルド エラーはすべてタスク一覧に追加されます。

ビルド成功後、出力ブラウザーが更新されます。出力ブラウザーは、ソリューション エクスプローラーで Dotfuscator プロジェクトの[**出力**]項目をダブルクリックするとアクティブになります。出力ブラウザーでは、クラス ブラウザーで提供されるビューに類似した出力アセンブリのビューが提供されます。元のシンボル名と難読化処理後のシンボル名に加え、どのシンボルが入力アセンブリから除去されたかが示されます。

HTML 形式の名前の変更レポートまたは除去レポートの生成を選択した場合、これらはビルド後に表示可能になります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.2.11 [表示]メニュー

Visual Studio 統合バージョンの Dotfuscator は、Visual Studio の[**表示**]メニューに[Dotfuscator]カスケード メニューを追加します。このメニューには、実行内容に応じて次の 4 つの項目が表示されます。2 つは成功ビルドによって生成されたレポートを表示するための項目で、もう 1 つはスタック トレースのデコード ツールを起動するための項目、最後の 1 つは新しい Shelf Life(使用期限)トークンを生成するための項目です。

HTML 形式レポートの表示

Visual Studio の中で、HTML 形式の名前の変更レポートと除去レポートを表示できます。HTML 形式レポート ファイルが存在し、Dotfuscator プロジェクトがソリューション エクスプローラー内でアクティブ プロジェクトになっている場合は、[**表示 | Dotfuscator**]メニューの 2 つのメニュー項目(1 つのレポートに付き 1 つ)が有効になります。いずれかをクリックすると、対応するレポートが既定のブラウザーに表示されます。

HTML 形式レポートの設定については「[名前の変更]エディターの[オプション]タブ」を、削除については「[除去]エディターの[オプション]タブ」を参照してください。

スタック トレースのデコード ツール

Visual Studio には、Dotfuscator のスタック トレース デコード ツールが Visual Studio ツール ウィンドウとして実装されています。このツールは、[**表示 | Dotfuscator**]メニューから[**難読化されたスタック トレースのデコード**]をクリックするとアクティブになります。ツールの使い方の詳細については、「難読化されたスタック トレースのデコード」を参照してください。

Shelf Life トークンの生成ツール

Visual Studio には、Dotfuscator の Shelf Life トークンの生成ツールが Visual Studio ツール ウィンドウとして実装されています。このツールは、[**表示**]メニューから[**Dotfuscator**]の[**新しい Shelf Life トークンの生成**]をクリックするとアクティブになります。このツールの使用方法については、「Shelf Life トークンの生成」を参照してください。

2.5.2.12 [ヘルプ]メニュー

Visual Studio 統合バージョンの Dotfuscator は、Visual Studio の[**ヘルプ**]メニューに[Dotfuscator]カスケード メニューを追加します。このメニューには以下の項目が含まれます。

- **製品の登録**。このメニュー項目が有効になるのは、お使いの製品が未登録の場合のみです。この項目をクリックすると、Dotfuscator を登録するためのダイアログ ボックスが表示されます。詳細については「Dotfuscator の登録とアクティブ化」を参照してください。
- **Dotfuscator のアクティブ化**。この項目は、Dotfuscator のサブスクリプションがアクティブ化されていない、期限が切れている、または期限切れが近づいている場合にのみ有効になります。詳細については「Dotfuscator の登録とアクティブ化」を参照してください。
- **新機能**。このメニュー項目をクリックすると、ブラウザーが起動して Dotfuscator のホーム ページが表示されます。
- **カスタマー フィードバックのオプション**。この項目を使用すると、PreEmptive Solution の匿名のカスタマー フィードバック プログラムを選択して、Dotfuscator ファミリのソフトウェア製品およびサービスの品質向上を促進することができます。
- **更新情報を今すぐ確認**。このメニュー項目をクリックすると、Web を使用した Dotfuscator の更新チェックが直ちに実行されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

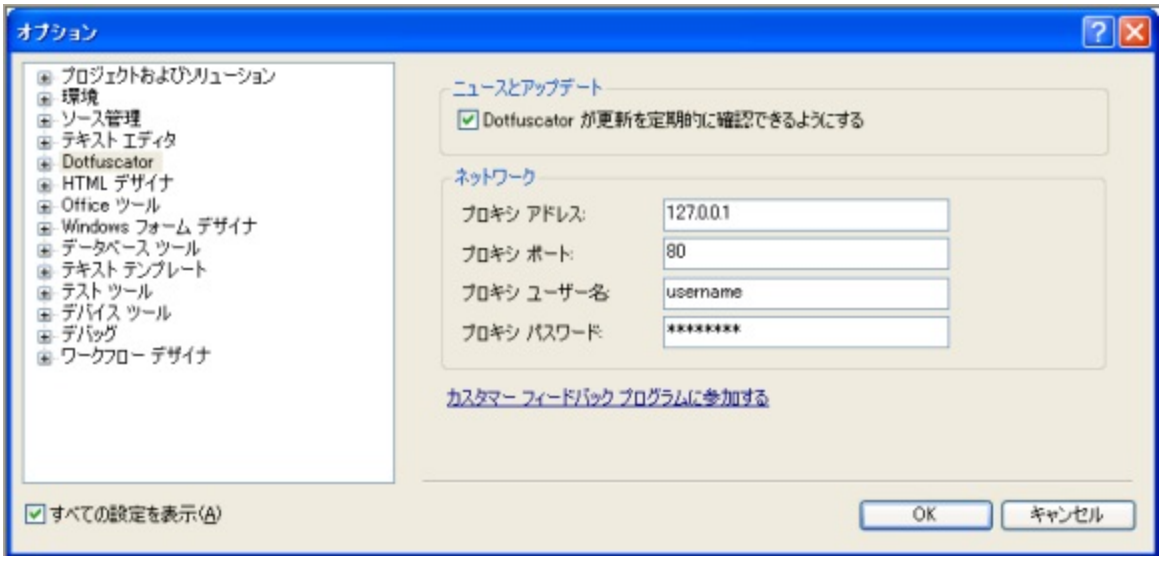
- **バージョン情報**。このメニュー項目をクリックすると、Dotfuscator のバージョン情報ボックスが表示され、ユーザー情報とバージョン情報を確認できます。

2.5.2.13 ユーザー設定の設定

Visual Studio では、Dotfuscator のユーザー設定は[**ツール**｜**オプション**｜**Dotfuscator**]メニューをクリックして利用できます。

このダイアログ ボックスの[**ニュースとアップデート**]セクションでは、Dotfuscator が更新を定期的に確認するように設定できます。

このダイアログの[**ネットワーク**]セクションでは、ネットワーク アクセスのプロキシ サーバーの構成設定を入力します。プロキシ サーバーがない場合、またはプロキシ サーバーの設定が Internet Explorer によって制御される場合には、プロキシ情報は必要ありません。



ユーザー設定の[オプション]ウィンドウにはカスタマー フィードバック プログラムのダイアログを起動するリンクがあります。このリンク内のテキストは、現在の選択/非選択状態を示します。

2.5.3 [名前の変更]エディター

[名前の変更]エディターには次の 3 つの構成用タブがあります。[**除外**]タブは、カスタム対象除外規則をグラフィカルに設定する場合に使用します。[**オプション**]タブは、名前の変更に関係するその他のオプションを構成する場合に使用します。[**組み込み規則**]タブは、カスタム規則の記述が不要で、一般的な名前変更の対象除外規則を実行する場合に使用します。

> 新規プロジェクトの場合、名前の変更の変換に対する既定の設定は「**有効化**」になっています。

2.5.3.1 [名前の変更]エディターの[オプション]タブ

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

[名前の変更]の[**オプション**]タブは名前の変更のオプションを設定したり、増分難読化で使用する割り当てファイルを指定する場合に使用します。

[**名前の変更オプション**]セクションには、[拡張オーバーロード誘導を使用する]、[名前空間を保持する]および[階層を保持する]を選択または選択解除できる 3 つのチェック ボックスがあります。

使用する**名前変更規則**をドロップダウン ボックスの一覧から選択できます。定義済みの名前変更規則については、「名前の変更規則」で説明しています。

[**XML シリアライザーと互換性のある名前付きメンバーを作成する**]チェック ボックスをオンにすると、名前の変更機能に追加規則が設定されます。すべてのクラスおよびメンバーの名前は、XML シリアル化を許可する方法で変更されます。「XML シリアル化と名前の変更」を参照してください。

[**名前の変更時、明示的なメソッドのオーバーライドを導入する**]チェック ボックスをオンにすると、オーバーライドしたメソッドがオーバーライドされたメソッドとは異なる名前を持つことができます。

[**名前の変更プレフィックス**]チェック ボックスをオンにすることによって、名前の変更に使用するプレフィックスを選択することができます。テキスト ボックスに入力した文字列が、難読化処理されたすべての型名の先頭に付加されます。文字列を何も入力しないと、Dotfuscator が入力モジュールごとに、モジュール名を基に適宜プレフィックスを生成します。名前の変更プレフィックスの詳細については、「名前の変更プレフィックス」を参照してください。

[**入力割り当てファイル**]セクションでは、Dotfuscator が連続して実行された場合にも名前付け規則が保持されるように、以前の実行で使用した割り当てファイルを指定できます。これは「増分難読化」として知られている処理です。この機能の詳細については、「増分難読化」セクションを参照してください。ここでは 2 とおりの方法を使用できます。使用する割り当てファイル名およびパスがわかっている場合は、テキスト ボックスにファイル名とパスを直接入力できます。もう 1 つの方法として、ファイル システムを参照して、ファイルを出力する適切な場所を選択することもできます。テキスト ボックスの右側にある[**参照**]ボタンをクリックすると、[**入力割り当てファイルの選択**]ウィンドウが表示されます。このウィンドウには、ファイルの場所を参照するためのナビゲーション ツリーが表示されます。

ナビゲーションを簡単にするために、この画面の下部にはアスタリスク(*)の付いた注釈があり、出力割り当てファイルが、[設定]タブにある[レポート]-[名前の変更]で構成されることを思い出させてくれます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.3.1.1 拡張オーバーロード誘導によるメソッドの名前の変更

Dotfuscator Professional Edition は、メソッドの戻り値の型またはフィールドの型を、メソッドまたはフィールドの一意性を判断するときの条件として使用できるようにすることによって、Overload-Induction（オーバーロード誘導）を拡張します。この機能により、名前変更されたメソッド名およびフィールド名の重複性を最大 15% まで高めることができます。さらに、戻り値の型やフィールドの型によるオーバーロードは（C#、VB などの）ソース言語では許可されていないので、逆コンパイルをいっそう困難なものにします。

この機能もアプリケーションのコンパイル時の解析に依存しています。このため、リモート メソッド呼び出しはこの機能を利用できません。その理由は、同じ型のメソッドと戻り値の型だけが異なるリモート オブジェクトのメソッドを呼び出すときに、リモート処理があいまい一致の例外（ambiguous match exception）をスローするためです。したがって、リモート処理を使用する場合には、2 つの選択肢があります。1 つ目は、拡張オーバーロード誘導を使用しないことです。通常のオーバーロード誘導は行われ、まったく安全です。2 つ目はリモート呼び出しされるクラスを名前の変更処理から除外することです。このオプションをリモート呼び出しで使用することがないように、既定値ではこの機能は無効になっています。

同じような理由で、シリアル化可能とマークされているすべての型に対し、拡張オーバーロード誘導は自動的に抑制されています。この動作が望ましくない場合は、構成ファイルの名前の変更セクションに "**enhancedOIOnSerializables**" オプションを追加するか（「名前の変更オプション」を参照してください）、ユーザー インターフェイスの［**名前の変更**］エディターの［**オプション**］タブにある［**シリアル化可能な型について抑制しない**］チェック ボックスをオンにすることによって、既定の動作を変更できます。

2.5.3.1.2 クラスの名前の変更オプション

完全なクラス名の変更

この既定の方法では、クラスおよび名前空間の名前は、新しい短い名前に変更されます。この考え方は非常に単純です。ここで挙げた例の場合、以下のようになります。

元の名前	変更後の名前
`Preemptive.Application.Main`	a
`Preemptive.Application.LoadData`	b
`Preemptive.Tools.BinaryTree`	c
`Preemptive.Tools.LinkedList`	d

メモ: すべてのクラスが `<global>` 名前空間に存在するようになりました。

Keepnamespace オプション

この方法は、名前空間の階層を保持したままクラスの名前を隠す方法としては優れています。サイズの縮小効果や難読化の度合いはいくらか減少しますが、名前空間の名前は保持されます。この方法は、難読化されたコードにリンクされている可能性のあるライブラリ、または難読化されたコードを既に使用しているアプリケーションにとっては有益です。以下に、このような名前の変更の例を示します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

元の名前	変更後の名前
Preemptive.Application.Main	Preemptive.Application.a
Preemptive.Application.LoadData	Preemptive.Application.b
Preemptive.Tools.BinaryTree	Preemptive.Tools.a
Preemptive.Tools.LinkedList	Preemptive.Tools.b

メモ:Keepnamespace と Keephierarchy オプションは、同時に指定できません。

Keephierarchy オプション

このオプションでは、名前空間やクラスの名前を変更しつつ、名前空間の階層を保持するように Dotfuscator に指示します。たとえば、次のようになります。

元の名前	変更後の名前
Preemptive.Application.Main	a.a.a
Preemptive.Application.LoadData	a.a.b
Preemptive.Tools.BinaryTree	a.b.a
Preemptive.Tools.LinkedList	a.b.b

メモ:KeepNamespace と KeepHierarchy オプションは、同時に指定できません。

名前の変更プレフィックス

ある種のケースにおいては、アセンブリ全体を通じてトップ レベルの型名として、たとえそれらの型が定義されているアセンブリ内でしか使用されない場合であっても、一意な名称を付与することが望ましいことがあります。これは、すべてのアセンブリを一度に Dotfuscator で処理することにより実現します。しかしながら、このやり方は必ずしもうまく行くとは限らないため、Dotfuscator を数回に分けて実行するような場合でも一意性を保持するために名前の変更プレフィックスを導入しました。

名前の変更プレフィックスは変更後の型名の先頭に付加されます。このプレフィックスは、特定の Dotfuscator の実行時にすべてのアセンブリで使用されるようユーザーが指定することができます。あるいは、Dotfuscator 自身に対象となる型のモジュール名から適切なプレフィックスを生成させることもできます。

この機能の興味深い利用方法としては、名前空間の誘導があります。ピリオド(".")で終わる名前の変更プレフィックス(たとえば "**MY_PREFIX**.")を定義することにより、型の難読化処理結果をユーザーが定義した名前空間下に格納することができます。

例

元の名前	プレフィックス	名前変更の種類	変更後の名前
Application.Main	[default]	default	MyApplicationa
Application.LoadData	myprefix	default	myprefixa
Tools.BinaryTree	myprefix	keephierarchy	a.myprefixa

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Tools.LinkedList	myprefix	keepnamespace	Tools.myprefixa
Tools.Proxy	mynamespace.	keepnamespace	Tools.mynamespace.a

プロジェクトに応じて名前の変更プレフィックスを設定するには、以下の 3 つの方法があります。

- GUI を使用し、[**名前の変更**]の[**オプション**]サブ タブにある[**名前の変更プレフィックス**]チェック ボックスをオンにします。名前の変更プレフィックスを自分で設定したい場合は、テキスト ボックスに入力します。空欄のままにしておけば、Dotfuscator にプレフィックスを生成させることができます。
- コマンド ラインで、**/prefix:[on > off]** オプションを使用します。コマンド ラインからカスタム プレフィックスを指定する場合は、**/p** オプションを使用して "**prefix**" プロパティを定義することで行います。

プレフィックスをオンにする

```
Dotfuscator /p=prefix=MY_PREFIX /pref:on [その他のオプション...]
```

- テキスト エディターあるいは XML エディターを使用し、名前の変更セクションに "**prefix**" オプションを追加します。カスタム プレフィックスを指定する場合は、プロパティ リスト セクションに "**propertylist**" プロパティを追加し、使用するプレフィックスを指定します。

カスタム プレフィックスの定義

```
<propertylist>
   <!-- 名前の変更プレフィックスが有効な場合、ここでプレフィックスを定義することで、
      その値を名前の変更のプレフィックスとして使用するよう指示されます -->
   <property name="prefix" value="MY_PREFIX"/>
</propertylist>
<renaming>
   <!-- 次の設定により、名前変更のプレフィックス機能が有効になります -->
   <option>prefix</option>
...
</renaming>
```

2.5.3.1.3 XML シリアル化と名前の変更

Dotfuscator では、XML シリアライザーに準拠する方法でクラスやメンバーの名前を変更するように、名前の変更アルゴリズムをグローバルに切り替えることができます。完全に難読化されたクラスをアプリケーションでシリアル化したい場合は、このモードで名前の変更機能を実行する必要があります。このモードは、シリアル化可能なクラスを名前の変更対象から除外する場合には必要ありません。

名前の変更機能がこのモードで実行されているときに従う規則の一覧は次のとおりです。

- 拡張オーバーロード誘導はオフになっています。
- すべての型のパブリック インスタンス プロパティおよびフィールドには、継承階層において一意な名前が与えられます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- プロパティ メタデータは、**System.Xml.Serialization** 名前空間のあらゆる属性で修飾されたプロパティに保持されます。
- **System.Collections.ICollection** インターフェイスを実装するパブリックの型では、**Add** メソッドは名前の変更対象から除外され、プロパティ メタデータは **Item** プロパティに保持されます。
- **System.Collections.IEnumerable** インターフェイスを実装するパブリックの型では、**Add** メソッドおよび **GetEnumerator** メソッドは名前の変更対象から除外されます。

このモードを設定する構成オプションについては、「名前の変更」オプションを参照してください。ユーザー インターフェイスでは、このモードは [名前の変更]エディターの[オプション]タブで設定できます。

2.5.3.1.4 名前の変更時、明示的なメソッドのオーバーライドを導入する

この機能は、Dotfuscator が明示的な(構文上でない)メソッドのオーバーライドを導入することによって、より多くのメソッドの名前を変更できるようにします。つまり、オーバーライドされるメソッドが、オーバーライドするメソッドとは異なる名前を持つことができます。たとえば、メソッドで Object.ToString() をオーバーライドする場合、通常、Dotfuscator はオーバーライドの関係を壊さないで名前を変更することはできません。これは、一般的に入力アセンブリには Object クラスがないため、ToString() の名前は変更されないからです。この設定を有効にすると、Dotfuscator はオーバーライドするメソッドの名前を変更でき、そのメソッドが Object.ToString() をオーバーライドするように意図して作られていることを CLR に指示するメタデータを導入できます。

元の名前	変更後の名前
System.Object.ToString()	ToString() [入力アセンブリにないクラス]
Preemptive.MyClass.ToString()	a()

2.5.3.2 [名前の変更]エディターの[除外]タブ

[名前の変更]エディターの[**除外**]タブでは、プログラムに対する名前の変更処理の対象から除外する必要がある項目の設定を、あらゆる部分にわたってきめ細かく制御できます。

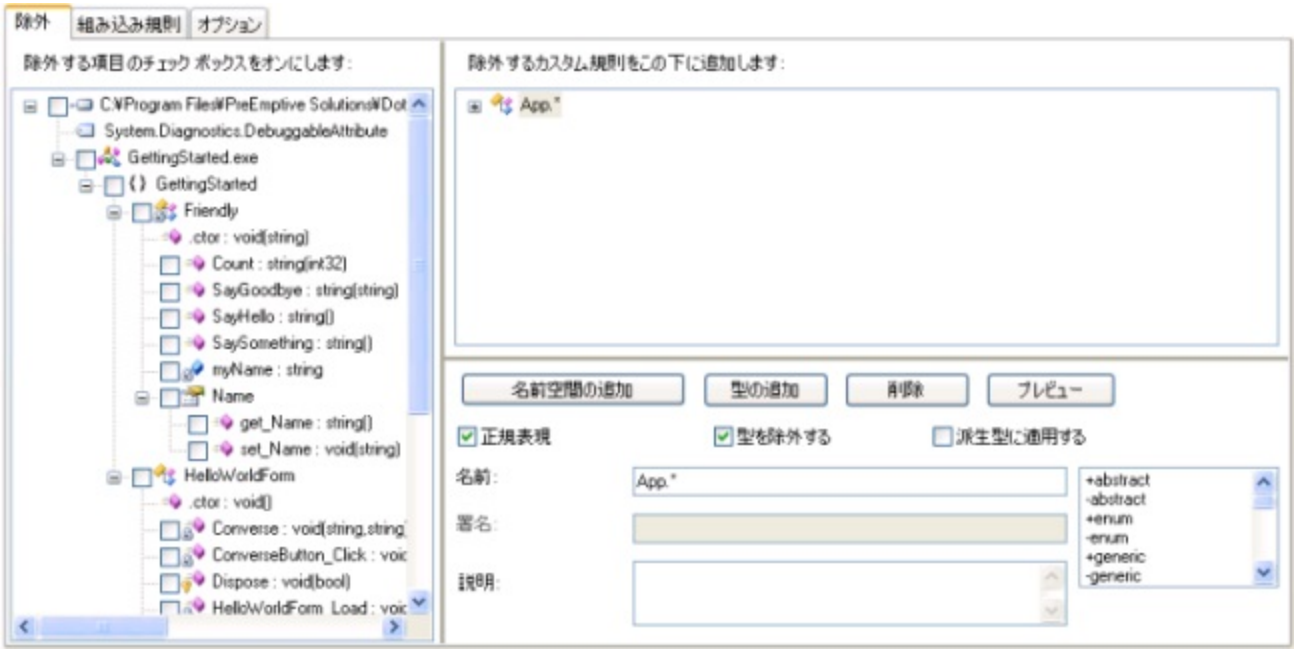
アプリケーションのツリー ビューを参照して、対象から除外する項目を確認し、特定の項目を名前の変更処理から除外できます。また、除外の対象とする複数の項目を選択するための、独自のカスタム規則をビジュアルに作成することもできます。

規則の結果をプレビューし、作成した対象除外規則を微調整できます。アプリケーションのツリー ビューでは、除外の対象として選択されたすべての項目が網掛け表示されます。すべての規則を適用した場合の効果をプレビューすることも、選択した特定の規則の効果だけをプレビューすることもできます。

対象選択規則および対象除外規則の使用の詳細については、「グラフィカルな規則の編集インターフェイスの使用」を参照してください。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

名前変更の除外規則の詳細については、名前の変更の対象除外規則を参照してください。



2.5.3.2.1 名前の変更の対象除外

対象除外リスト セクションは、入力アセンブリ名の変更を動的に微調整する方法を提供します。ユーザーは、実行時に適用される規則のリストを指定します。規則で特定のクラス、メソッド、またはフィールドを選択した場合、その項目の名前は変更されません。

これらの規則は、library オプションなどのグローバル オプションによって暗黙に適用される規則と併せて適用されます。

規則は論理的に **OR** で結合されます。

名前空間、型、メソッド、またはフィールドの選択には、正規表現(RE)を使用できます。これを実行するには、省略可能な **regex** 属性を使用します。**regex** の既定値は false です。**regex** が true の場合、**name** 属性の値は正規表現と解釈されます。regex が false の場合、name は文字どおりに解釈されます。正規表現は特定の文字(ピリオドなど)に特別な意味を割り当てているので、regex の値は重要です。以下に、簡単な正規表現の例を示します。

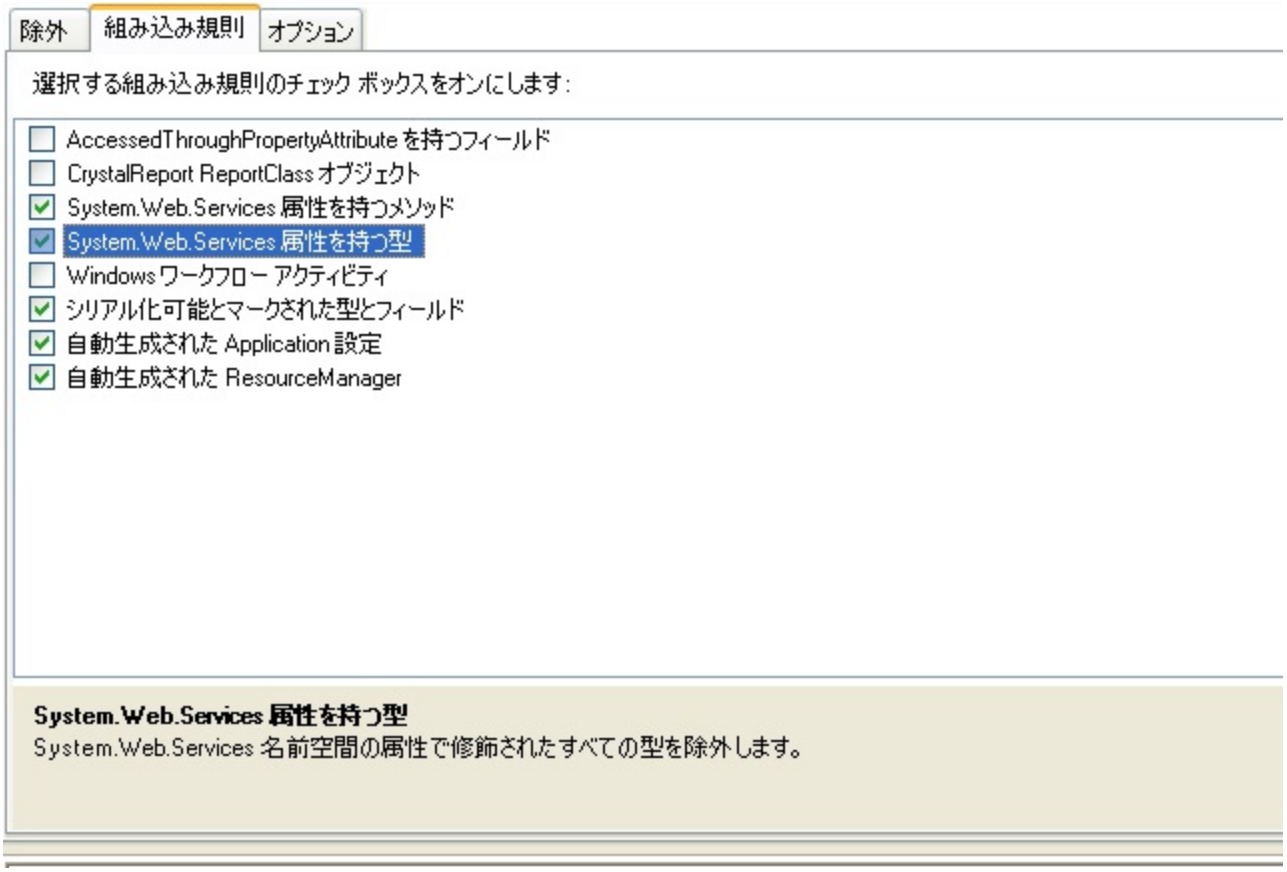
以下に、簡単な正規表現の例を示します。	
`.*`	すべてと一致
`MyLibrar.`	`Matches MyLibrary`、`MyLibrari` などと一致
`My[\.]Test[\.]I.*`	`My.Test.Int1`、`My.Test.Internal` などと一致
`Get.*`	`GetInt`、`GetValue` などと一致
`Get*`	`Ge`、`Get`、`Gett`、`Gettt` などと一致

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

正規表現の構文の詳細については、.NET Framework のドキュメントを参照してください。

2.5.3.3 [名前の変更]エディターの[組み込み規則]タブ

Dotfuscator の[**名前の変更**]エディターの[**組み込み規則**]タブには、**%ProgramData%\PreEmptive Solutions\Common\dotfuscatorReferenceRule_v1.4.xml** に定義されている名前変更の対象除外規則が表示されます。これらの規則は、特定のアプリケーションの種類またはテクノロジに適用される標準の対象除外です。規則はそれぞれ解説文を持っており、自身が選択されたときにそれをフォームに表示します。組み込み規則は対応するチェック ボックスをオンにすることによって、プロジェクトに適用できます。



2.5.4 [制御フロー]エディター

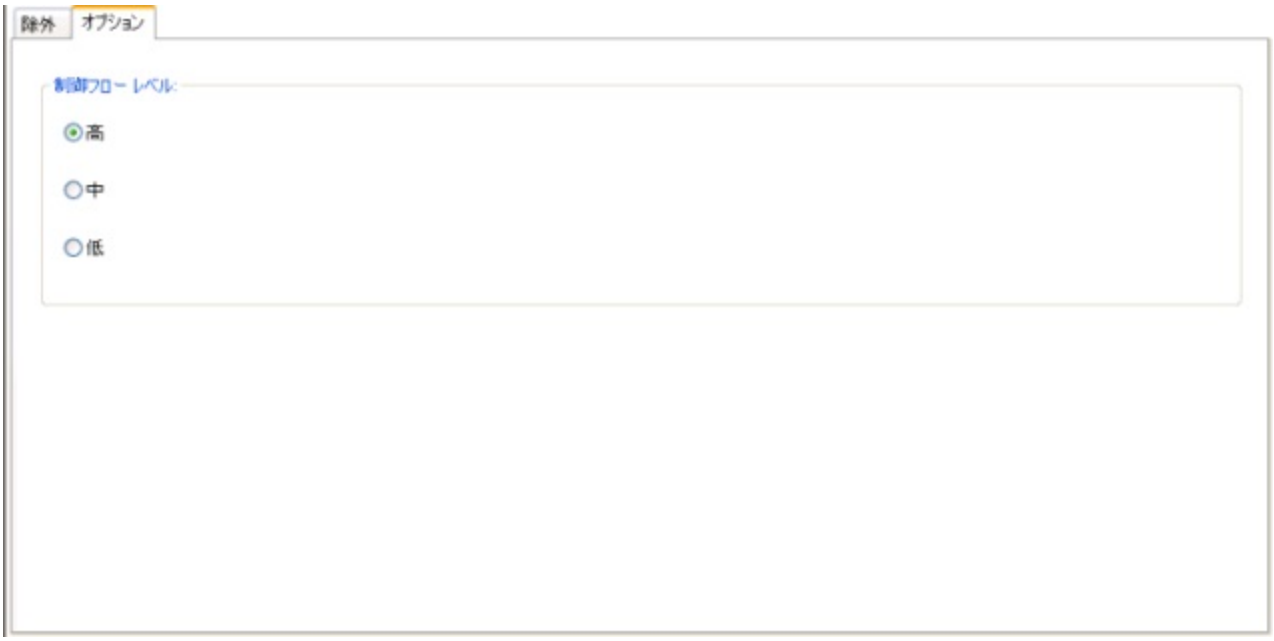
[**制御フロー**] エディターには次の 2 つの構成用タブがあります。[**除外**]タブは、カスタム対象除外規則をグラフィカルに設定する場合に使用します。[**オプション**]タブは、制御フローの難読化に関係するその他のオプションを構成する場合に使用します。

> 新規プロジェクトの場合、制御フローの変換に対する既定の設定は「**有効化**」になっています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.4.1 [制御フロー]エディターの[オプション]タブ

[**制御フロー**]エディターの[**オプション**]タブは、制御フローの難読化のグローバル レベルを設定する場合に使用します。



2.5.4.2 [制御フロー]エディターの[除外]タブ

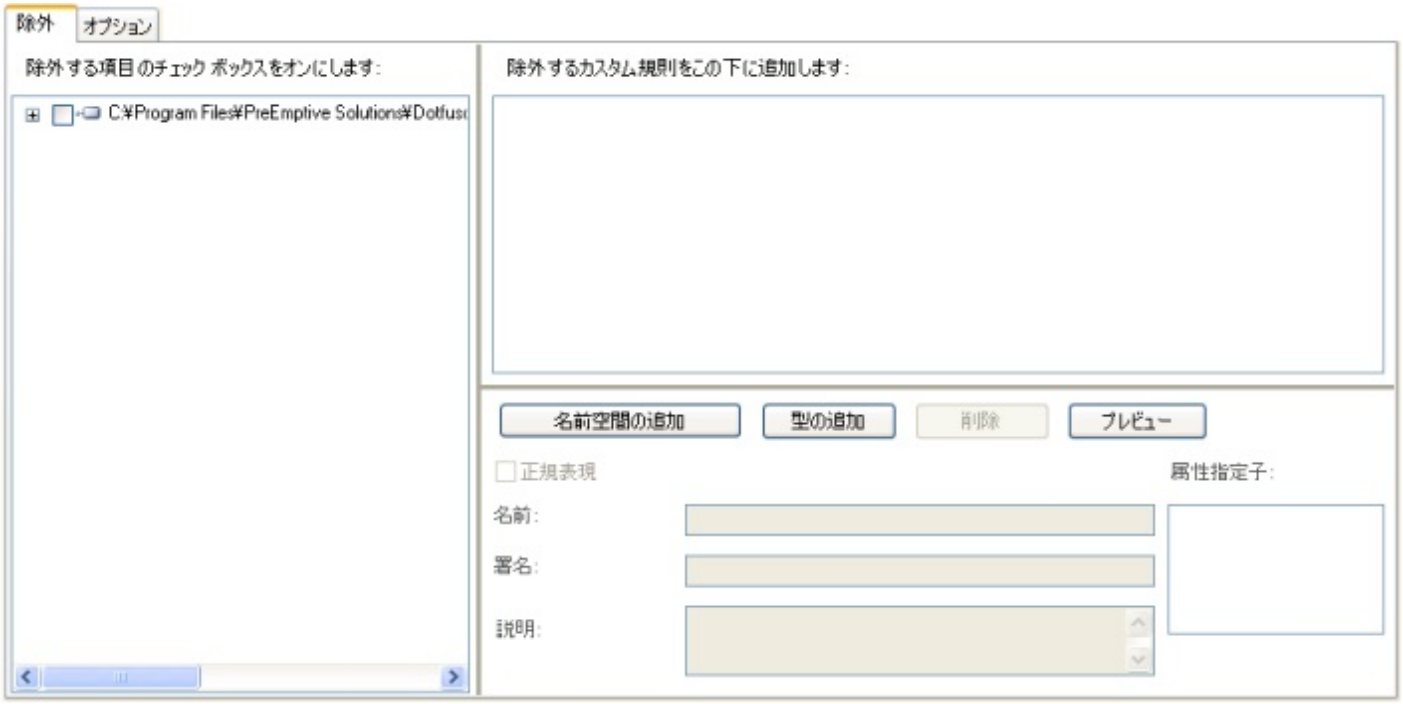
[**制御フロー**]エディターの[**除外**]タブでは、プログラムに対する制御フローの難読化処理から除外する必要のある項目の設定を、あらゆる部分にわたってきめ細かく制御できます。このタブでは、制御フローの難読化をすべて無効にすることもできます。

アプリケーションのツリー ビューを参照し、除外する項目の隣にあるチェック ボックスをオンにして、特定の項目を制御フローの難読化から除外できます。また、除外の対象とする複数の項目を選択するための、独自のカスタム規則をビジュアルに作成することもできます。

規則の結果をプレビューし、作成した対象除外規則を微調整できます。アプリケーションのツリー ビューでは、除外の対象として選択されたすべての項目が網掛け表示されます。すべての規則を適用した場合の効果をプレビューすることも、選択した特定の規則の効果だけをプレビューすることもできます。

対象選択規則および対象除外規則の使用に関する詳細については、規則の編集インターフェイスを参照してください。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.5.5 [文字列の暗号化]エディター

[**文字列の暗号化**]エディターの設定用のタブは、[**対象**]タブだけです。[**対象**]タブは、文字列の暗号化のカスタム対象選択規則をグラフィカルに設定する場合に使用します。

> 新規プロジェクトの場合、文字列の暗号化の変換に対する既定の設定は「**無効化**」になっています。

2.5.5.1 [文字列の暗号化]エディターの[対象]タブ

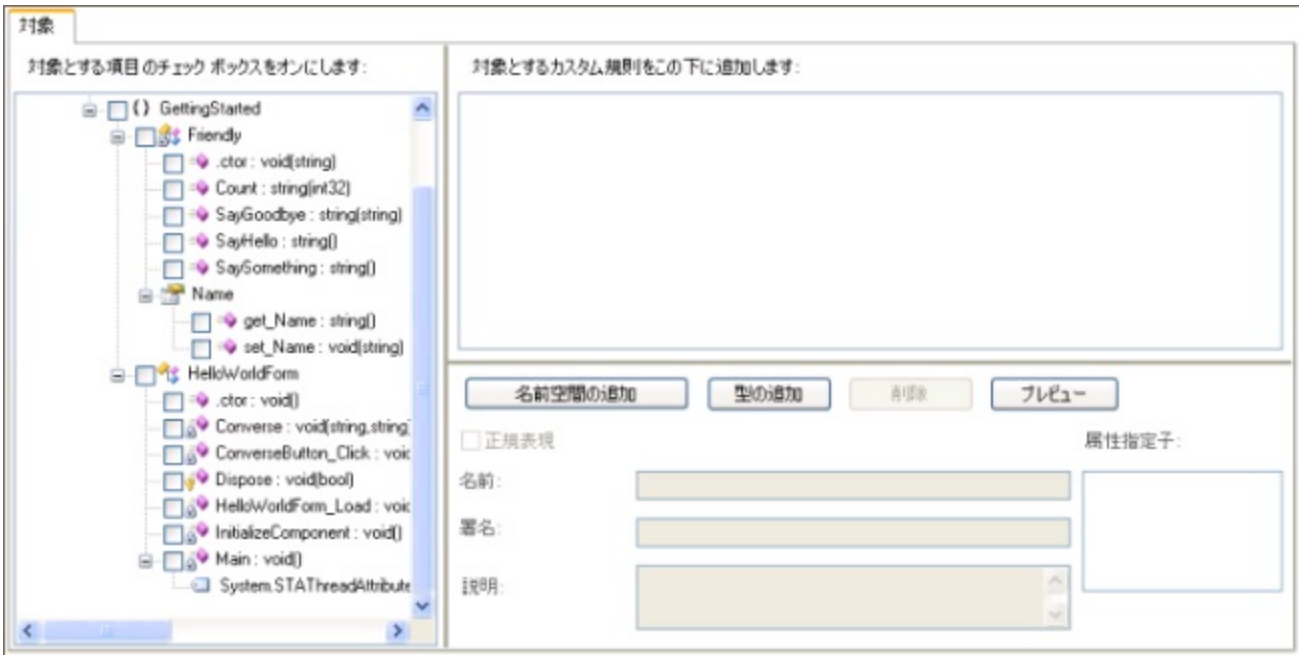
[**文字列の暗号化**]エディターの[**対象**]タブでは、文字列の暗号化処理の対象として選択するプログラムのあらゆる部分を、完全かつ詳細に制御できます。このタブでは、文字列の暗号化をすべて無効にすることもできます。

アプリケーションのツリー ビューを参照して、対象として選択する項目の隣にあるチェック ボックスをオンにして、特定の項目を文字列の暗号化の対象として選択できます。また、選択の対象とする複数の項目を選択するための、独自のカスタム規則をビジュアルに作成することもできます。

規則の結果をプレビューし、作成した対象選択規則を微調整できます。アプリケーションのツリー ビューは、対象として選択されたすべての項目を網掛け表示にします。すべての規則を適用した場合の効果をプレビューすることも、選択した特定の規則の効果だけをプレビューすることもできます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

対象選択規則および対象除外規則の使用に関する詳細については、規則の編集インターフェイスを参照してください。



2.5.6 [除去]エディター

[**除去**]エディターには、3 つの構成用タブがあります。

- [**対象トリガー**]タブは、コードを安全に除去できるかどうかを判断できるよう、アプリケーションのエントリ ポイントを Dotfuscator に指示するために使用します。
- [**条件付き対象**]タブは、リフレクションなどの特殊なケースのために、完全に除去してはいけない型について Dotfuscator に指示するために使用します。
- [**除去**]エディターの[**オプション**]タブは、除去レポートを構成する場合に使用します。

> 新規プロジェクトの場合、除去の変換に対する既定の設定は「**無効化**」になっています。

2.5.6.1 対象トリガーと条件付き対象の理解

Dotfuscator は、アプリケーションを静的に分析して、実際には使用されていない要素を判断し、それらの要素を出力バイナリから削除することができます。これにより、アプリケーションのサイズを縮小できます。

静的分析ではコード全体を調べ、"トリガー" またはエントリー ポイントと呼ばれる一連のメソッドから分析を開始します。通常、外部アプリケーションによって呼び出されることが予期されるメソッドは、トリガーとして定義されている必要があります。たとえば、単純なスタンドアロン アプリケーションでは、`Main` メソッドがトリガーとして定義されます。1 つのアセンブリに複数のトリガーを定義することができます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

アセンブリのライブラリ モードをオンにすると、Dotfuscator は自動的に、公開されているすべての型とメンバーをエントリ ポイントとして扱うようになることに留意してください。

Dotfuscator は、各トリガー メソッドのコード全体を調べ、使用されているフィールド、メソッド、型を記録します。その後、呼び出されているメソッドを同様の方法で分析します。この処理は、呼び出されているメソッドがすべて分析されるまで続けられます。この処理が終了した時点で、アプリケーションの実行に最低限必要な型とそのメンバーのセットを判別できます。出力アセンブリには、これらの型だけが含まれます。

Dotfuscator はある特定のメソッド（リフレクション / XAML など）が呼ばれていることを見分けることができないため、実行時に必要とされるものを削除しようとすることがあります。これを回避するには、**対象トリガー**を設定し、どのクラスのメンバー（メソッド、プロパティ、フィールド、イベント）を静的分析のための "エントリー ポイント" として扱う必要があるかを Dotfuscator に伝えます。Dotfuscator は、それらのメンバーとメンバーのすべての子孫をコール グラフに保持します。

しかしながら、ときに、これは最適な動作ではありません。次のようなアプリケーションを考えてみましょう。リフレクションを使用して一連の型を読み込み、それらをインターフェイスにキャストした後、そのインターフェイスでメソッドを呼び出す、つまり、本質的にはプラグイン モデルのアプリケーションです。Dotfuscator の静的分析は、読み込まれる可能性のある型を識別しようとはしませんが、そのインターフェイスでどのメソッドを呼び出そうとしているかはわかります。

このような場合は、条件付き選択対象として型を設定する必要があります。Dotfuscator はそれらを対象として選択し、それらがインターフェイスを実装することを把握します。インターフェイスのメソッドの一部が使用されていないと判断した場合は、それらのメソッドをインターフェイスから、および条件付きで対象として選択される型のすべての実装から除去します。除去されなかったメソッドはさらに、通常どおり除去するかどうか分析されます。

2.5.6.2 [対象トリガー]タブ

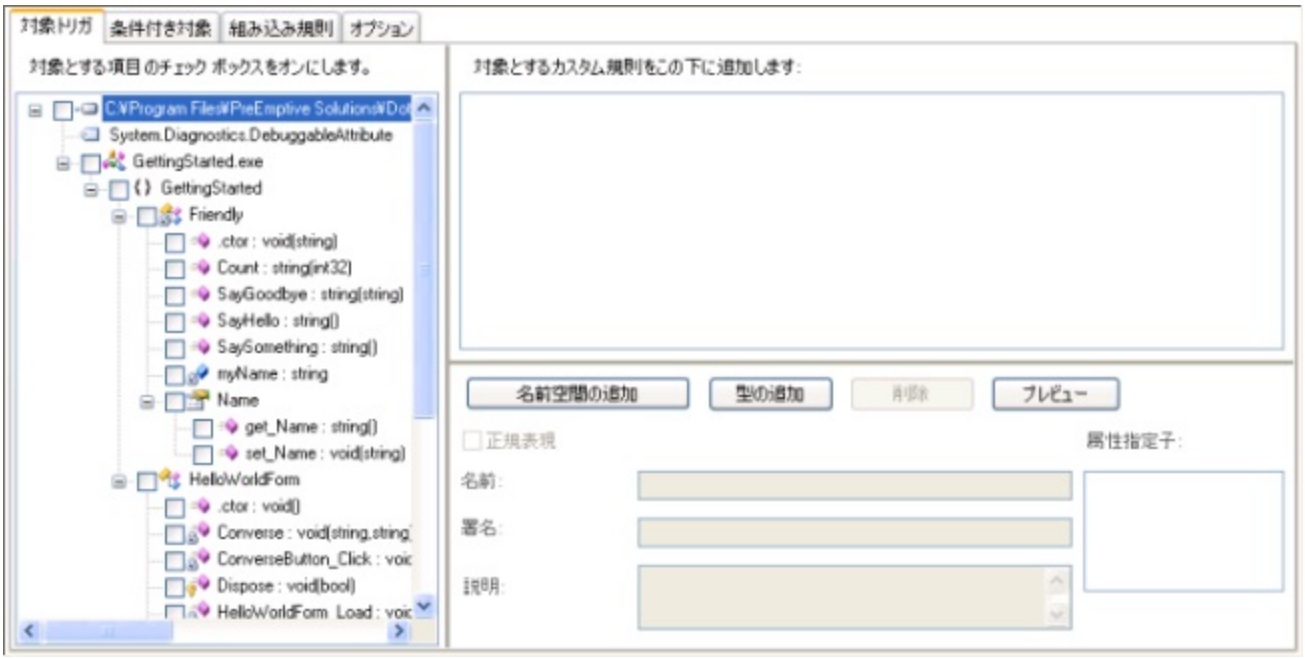
[**対象トリガー**]タブでは、不要コードの除去処理でアプリケーション エントリ ポイント（"トリガー"）として使用するすべてのメソッドをグラフィカルに指定できます。

アプリケーションのツリー ビューを参照し、対象として選択する項目の隣にあるチェック ボックスをオンにして、特定のメソッドをエントリ ポイントとして選択できます。また、対象となる複数のメソッドを選択するための、独自のカスタム規則をビジュアルに作成することもできます。

規則の結果をプレビューし、作成した対象選択規則を微調整できます。アプリケーションのツリー ビューでは、対象として選択されたすべての項目が網掛け表示されます。すべての規則を適用した場合の効果をプレビューすることも、選択した特定の規則の効果だけをプレビューすることもできます。

対象選択規則および対象除外規則の使用に関する詳細については、規則の編集インターフェイスを参照してください。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.5.6.3 [条件付き対象]タブ

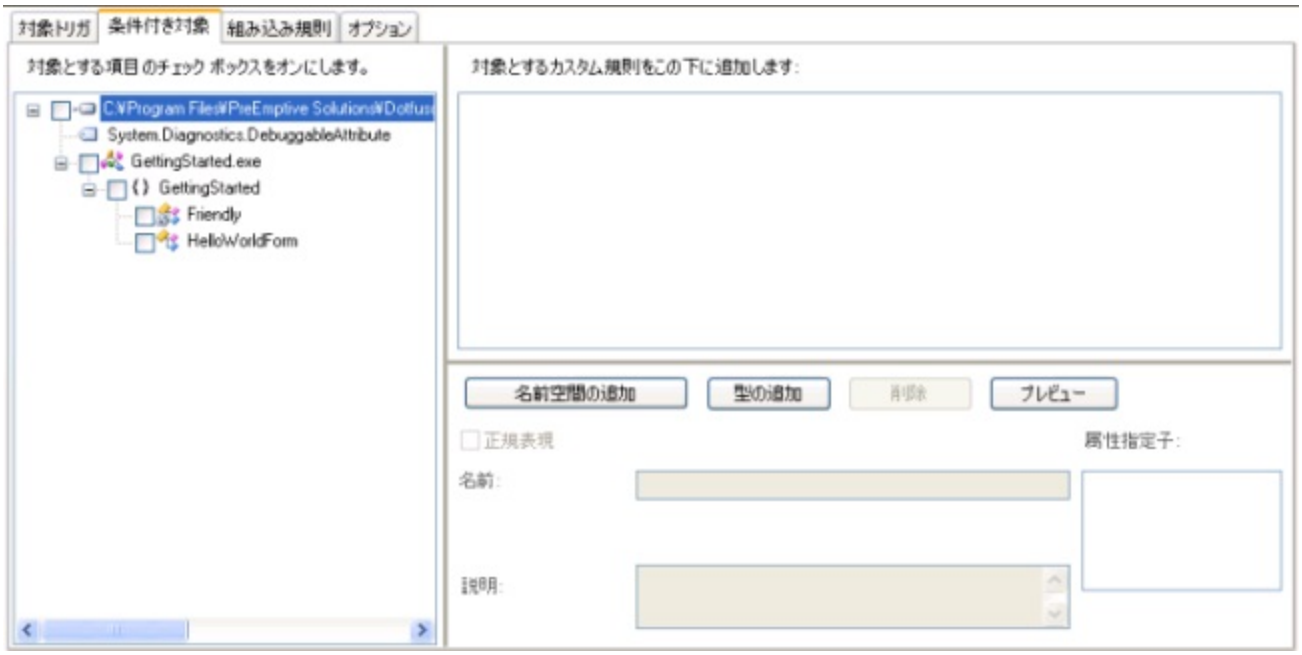
[**条件付き対象**]タブでは、条件付きで除去処理の対象として選択するすべての型をグラフィカルに指定できます。この機能の詳細については、対象トリガーと条件付き対象の理解を参照してください。

アプリケーションのツリー ビューを参照して、対象として選択する項目の隣にあるチェック ボックスをオンにして、条件付きで特定の型を対象として選択できます。また、対象となる複数の型を選択するための、独自のカスタム規則をビジュアルに作成することもできます。

規則の結果をプレビューし、作成した対象選択規則を微調整できます。アプリケーションのツリー ビューでは、対象として選択されたすべての項目が網掛け表示されます。すべての規則を適用した場合の効果をプレビューすることも、選択した特定の規則の効果だけをプレビューすることもできます。

対象選択規則および対象除外規則の使用に関する詳細については、規則の編集インターフェイスを参照してください。

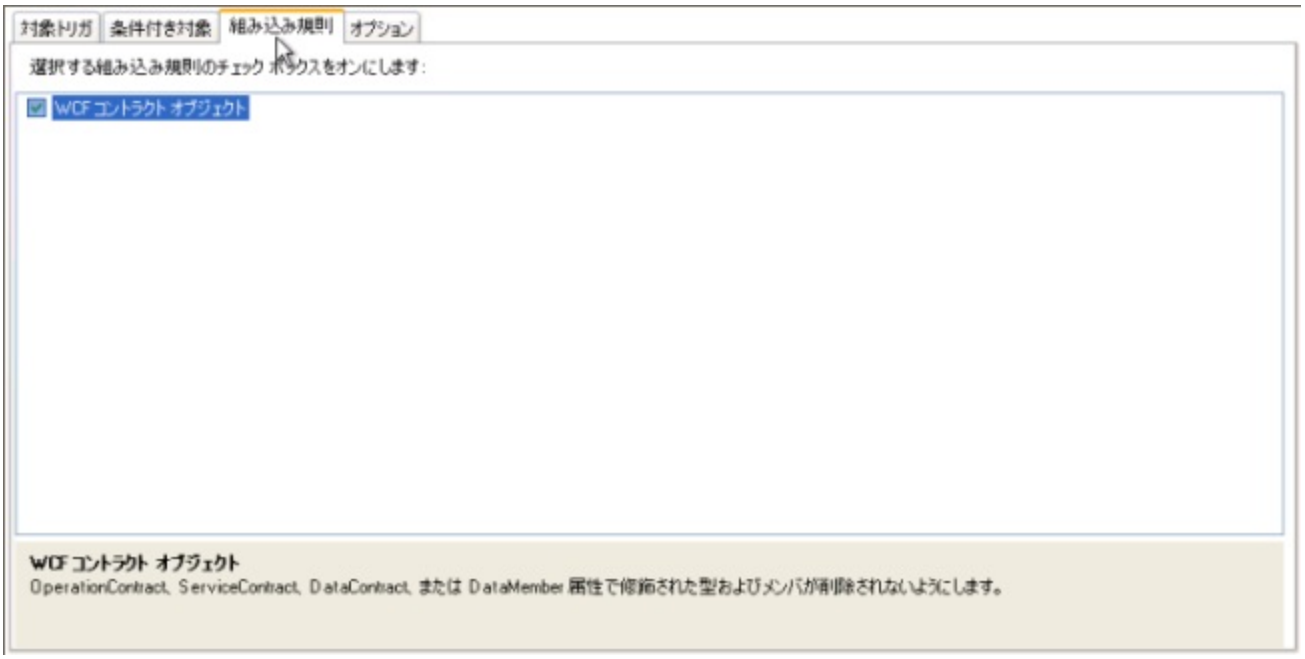
© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.5.6.4 [組み込み規則]タブ

Dotfuscator の[**除去**]エディターの[**組み込み規則**]タブには **%ProgramData%\PreEmptive Solutions\Common\dotfuscatorReferenceRule_v1.4.xml** に定義されている対象トリガー規則と条件付き対象規則が表示されます。これらの標準規則は、特定のアプリケーションの種類またはテクノロジに適用されます。規則はそれぞれ解説文を持っており、自身が選択されたときにそれをフォームに表示します。組み込み規則は対応するチェック ボックスをオンにすることによって、プロジェクトに適用できます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.5.6.5 [オプション]タブ

[**除去**]エディターの[**オプション**]タブは実行する削除の種類を選択するのに使用します。

[**リテラルのみ削除**]を選択すると定数宣言のみの削除を実行します。[**使用されていないメタデータとコードを削除**]を選択すると、定数定義を削除するだけでなく、通常のアルゴリズムに従って使用されていないメソッドおよびフィールドを判定して削除します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.6.5.1 定数のみ除去

アセンブリで不要コードを除去するよう設定したくはないが、一方で危険回避やアセンブリ サイズの削減を目的に除去を行いたいという状況に直面することもあるでしょう。このような場合、「定数のみ除去」は理想的な妥協点です。Dotfuscator は定数のみ除去の実行時、入力アセンブリから定数宣言（`const` フィールド）のみを除去します。未使用の型、メソッドおよびフィールドは検出せず、それらは出力アセンブリに反映されます。

定数のみ除去は、不要コードを完全に除去することが望ましくないような状況で行うのがよいでしょう。.NET コンパイラはコンパイル時に、コード内の `const` フィールドへの参照を、そのフィールドの実際の値で置き換えます。定数宣言はアセンブリにそのまま残り、外部アセンブリによる参照やリフレクションを介したアクセスがサポートされます。これらの状況をサポートする必要がない場合は、一般的に定数のみ除去を有効にするのがよいでしょう。

2.5.6.6 除去レポート

Dotfuscator は、すべての入力アセンブリの一覧と、それらがどのように除去されたかを示すレポートを XML 形式で生成します。列挙される各アセンブリには、型とそれらのメンバー（メソッド、フィールド、プロパティなど）の一覧と共に、その項目が除去されたかどうかを示す属性が表示されます。また、レポートには、各アセンブリにアタッチされているマネージ リソースの除去状況も記載されます。最後にレポートは、不要なコードの除去による全体的な効果に関する統計セクションを提供します。

除去レポートは、既定の変換（または独自の変換）を使用して HTML 形式に変換すると、非常に有益なレポートになります。既定の変換では、ブラウズ可能な相互参照レポートが生成され、除去された項目が赤色で示されます。

除去レポートの要素は割り当てファイルの要素と似ています。以下に、注意する必要のある点を示します。

- レポートには型、メソッド、フィールド、プロパティ、およびマネージ リソースの除去状態が含まれています。
- 型が除去された場合は、当然、そのメンバー（メソッド、フィールド、およびプロパティ）もすべて除去されています。
- 型名では、入れ子になったクラスの名前は "**/**" 文字で親と区切られます。
- コンストラクターには "`.ctor`" という名前が付けられますが、（静的初期化子、クラス コンストラクターなどとも呼ばれる）静的コンストラクターには "`.cctor`" という名前が付けられます。

2.5.7 [リンク]エディター

[リンク]エディターは操作の容易なドラッグ アンド ドロップ機能を備えているため、入力アセンブリを 1 つまたは複数の出力アセンブリに割り当てる設定をすばやく指定できます。複数の出力アセンブリへのリンクを設定した場合は、各出力アセンブリのリンクを設定後に個別に構成できます。

> 💡 新規プロジェクトの場合、リンクの変換に対する既定の設定は「**無効化**」になっています。

2.5.7.1 [入力アセンブリ]および[リンク アセンブリ]

リンク用の出力アセンブリを作成および構成するには、次の手順に従います。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- [**リンク アセンブリ**]ツール バーの[**新規のリンク アセンブリの作成**]ボタンをクリックします。新しいアセンブリのエントリが、リンク アセンブリのリストに現れます。
- 新しい出力アセンブリの名前は、[**リンクの詳細**]セクションにある[**出力アセンブリ**]フィールドに入力して指定します。パス情報の入力は必要ありません。出力アセンブリはすべて同様に、出力先ディレクトリへ書き込まれます。
- [**入力アセンブリ**]リストからリンクするアセンブリを選択し、それを[**リンク アセンブリ**]リスト内の新規アセンブリにドラッグします。そうすると、[**入力アセンブリ**]リストから選択したアセンブリが消え、新規アセンブリの子ノードとして現れます。
- プライマリ アセンブリとしてマークする入力アセンブリを右クリックし、ショートカット メニューから[**プライマリ アセンブリの設定**]を選択します。この操作は、プライマリ アセンブリを選択し、ツール バー ボタンを使用しても行えます。プライマリ アセンブリはタグ アイコンで示されます。
- 必要に応じて、新規アセンブリに対して名前変換ポリシーおよびエントリ ポイントを設定できます。
- 作成する各出力アセンブリについて、以上の手順を繰り返します。

[**リンク アセンブリ**]リストからアセンブリを削除するには、目的のアセンブリを選択し、ツール バーの[**選択したアセンブリの削除**]ボタンを使用するか、または **Delete** キーを押します。

下のスクリーンショットは、リンクされた出力が複数ある場合の例を示しています。最初に入力アセンブリの **Driver.exe** と **LibraryC.dll** が **out.exe** にリンクされ、次に **LibraryA.dll** と **LibraryB.dll** が **outlib.dll** にリンクされます。最後に、**LibraryE.dll** がリンクされないで "パス スルー" します。

2.5.7.1.1 プライマリ アセンブリ

リンクを設定する場合、入力アセンブリのうちの 1 つを **プライマリ アセンブリ**として指定する必要があります。Dotfuscator は、新しく作成された出力アセンブリに、プライマリ アセンブリに格納されているマニフェスト情報(バージョン番号、公開キーなど)を適用します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.7.1.2 名前の改変

リンカーは、アセンブリの結合中、名前付けの衝突を防ぐために名前の変更が必要になる場合があります。たとえば、2 つの入力アセンブリがまったく同じ名前のプライベート クラスを持っているかもしれません。その場合、アセンブリを結合するには、どちらかの名前を変更する必要があります

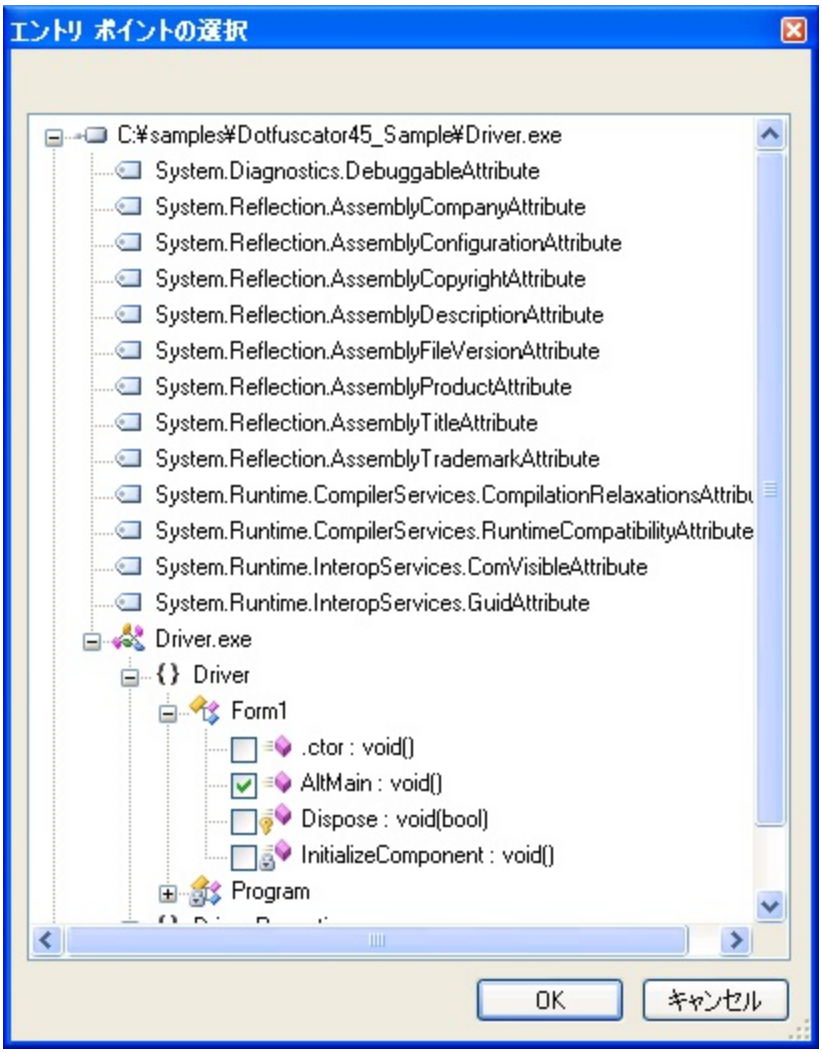
ほとんどの場合、（元の名前のうち少なくとも 1 つが、そのアセンブリ外に公開されていない場合）Dotfuscator は名前を安全に改変できます。しかし、同じ名前の公開されている型が 2 つあることがわかった場合は、何らかの指示がなければ、どちらかの名前を安全に改変することはできません。このような場合にのみ、名前変換ポリシーが使用されます。既定値の動作では、エラー メッセージを表示してビルドが停止します。その他のオプションでは、該当する名前の 1 つを改変して警告を発するか、あるいは該当する名前を警告なく改変します。

2.5.7.2 エントリ ポイントの設定

リンクされた出力アセンブリのエントリ ポイントを設定する必要がある場合は、[リンク]エディターの[エントリ ポイント]セクションにある[**参照**]ボタンによって表示される[**エントリ ポイントの選択**]ダイアログ ボックスを使用できます。

このダイアログ ボックスには、リンクされた出力アセンブリで使用できるすべてのメソッドがグラフィカルに表示されます。エントリ ポイントとして選択するメソッドにチェック マークを入れます。このダイアログにより、[エントリ ポイント]セクションの残りの情報が埋められます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.5.7.2.1 エントリ ポイント

.NET では、実行可能なアセンブリは、メタデータ内にエントリ ポイントとマークされたメソッドを持っている必要があります（通常、このメソッドは **Main** と呼ばれますが、名前は何でもかまいません）。これは、アセンブリが実行されるときに CLR が呼び出すメソッドです。ある場合には、リンクされている出力アセンブリごとにエントリ ポイントの指定が必要になります。次の表は、エントリ ポイントの規則についてまとめてあります。

入力	出力	エントリ ポイント規則
すべて EXE	**EXE**	ユーザー指定のエントリ ポイントが必要です。リンカーは、入力アセンブリ上のすべてのエントリ ポイントを削除し、出力アセンブリにユーザー指定のエントリ ポイントを適用します。
すべて DLL	**EXE**	ユーザー指定のエントリ ポイントが必要です。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

EXE と DLL が混在	**EXE**	入力の exe が 1 つだけである場合を除き、ユーザー指定のエントリ ポイントが必要です。exe が 1 つの場合、リンクはそのエントリ ポイントを使用します。既定値は、ユーザー指定のエントリ ポイントによってオーバーライドされます。
任意	**DLL**	ユーザー指定のエントリ ポイントは使用されません。リンカーは、入力アセンブリ上のすべてのエントリ ポイントを削除します。

2.5.8 [PreMark]エディター

[**PreMark**]エディターでは、ウォーターマークを作成する対象のアセンブリを選択し、選択したアセンブリに適用するウォーターマークを設定することができます。

> 新規プロジェクトの場合、PreMark 変換に対する既定の設定は「**無効化**」になっています。

アセンブリの選択

左側に入力アセンブリの一覧が表示されます。アセンブリの隣にあるチェック ボックスをオンにすることによって、そのアセンブリにウォーターマークを設定するよう選択できます。

ウォーターマークのオプション

ウォーターマーク文字列を暗号化してから入力アセンブリに適用するよう選択することができます。暗号化は、指定したパスフレーズに基づいて行われます。[**パスフレーズを使って暗号化**]チェック ボックスをオンにすると、パスフレーズの入力テキスト ボックスが有効になるので、そこにパスフレーズを入力できます。

メモ:暗号化することで、入力アセンブリに適用されるウォーターマークのサイズが大きくなる可能性があることに留意してください。

Dotfuscator は、エンコードされ、場合によっては暗号化もされるウォーターマーク文字列が、指定の出力アセンブリに収まるかどうかを前もって予測することはできません。ウォーターマークの設定時、その文字列が収まらない場合にどう対処するかを、[**ウォーターマーク文字列が大きすぎる場合**]ドロップダウン リストを使って Dotfuscator に指示できます。自動的にウォーターマーク文字列を切り捨てて警告を発するか、またはエラーによってビルドを中止するかどうかを選択できます。

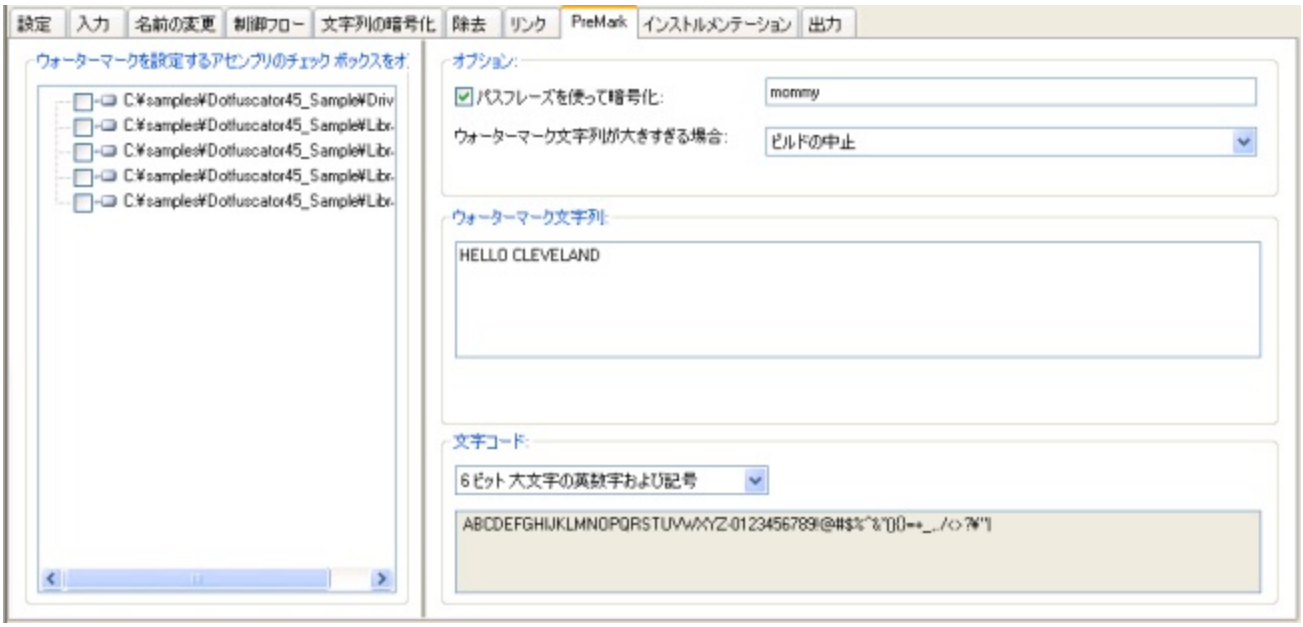
文字コード

ウォーターマーク文字列は、特定バイトにエンコードしてからアセンブリに適用する必要があります。Dotfuscator はいくつかの組み込み文字コードを用意しています。これらはサイズを最適化するための、一般的に使用される文字セットです。それらのいずれかを使用すると、より大きなウォーターマーク文字列が指定のアセンブリに収まるようになります。特殊化した文字コードのどれもが要求に合わない場合は、標準の UTF-8 エンコードを選ぶことができます。

ウォーターマーク文字列

ウォーターマーク文字列の入力時、現在選択されている文字コードに含まれない文字を入力すると、ユーザー インターフェイスがその旨を知らせます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.5.8.1 ウォーターマーク文字列の長さ

ウォーターマーク文字列の最大サイズは、構成オプションと対象となるアセンブリの複雑さによって決定されます。通常、アセンブリが大きければ、より大きい文字列を入れることができます。Dotfuscator は文字の符号化に必要なビット数を最小限に抑えるため、文字エンコード（文字コードと呼ぶ）を使用します。小さい文字エンコードを使用すれば、より長いウォーターマーク文字列を指定のアセンブリに対して作成できます。

また、暗号化アルゴリズムには一定のブロック長情報が必要です。このため、ウォーターマーク文字列の暗号化を選択した場合、指定可能なウォーターマーク文字列の最大長が暗号化をしない場合に比べて短くなる場合があります。

出力アセンブリが生成されるまで、Dotfuscator はウォーターマーク文字列の最大長を予測することはできません。しかし、Dotfuscator に対し、ビルド中にウォーターマーク文字列が出力アセンブリに収まらなかった場合どう対処するかを指示しておくことはできます。既定の設定では、アセンブリに収まるように文字列を切り詰め、出力ウィンドウに警告メッセージを出力します。エラーなしでビルドを停止させるよう、Dotfuscator に指示することもできます。どちらの場合も、メッセージにウォーターマークの最大サイズが示されます。

2.5.8.2 文字コード

Dotfuscator にはいくつかの文字コードが定義されており、それを使用してウォーターマーク文字列をエンコードすることができます。領域が問題となる場合は、より小さいエンコードを選択できますが、そうすると文字列内に表示できるキャラクタの数が低下します。

名前	説明	1 文字あたりのビット数
4bit-a	4 ビット 16 進数	4
6bit-a	6 ビット 大文字の英数字および記号	6

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

6bit-b	6 ビット 英数字	6
7bit-a	7 ビット 英数字および記号	7
UTF-8	任意の文字	可変

Dotfuscator のユーザー インターフェイスは、各文字コードに定義されている特定の文字を表示します。

2.5.8.3 ウォーターマークの抽出

Dotfuscator には、**premark** と呼ばれるコマンド ライン ツールが付属しています。このツールはアセンブリを入力として受け付け、ウォーターマークが設定されている場合は、それを出力します。このツールは Dotfuscator と同じディレクトリにインストールされます。

ウォーターマークの抽出
`premark [オプション] assembly1[,assembly2, ...]` `assembly1,...:  .NET` アセンブリまたはモジュールのリスト オプション： `/a` パスフレーズを要求します `/p=passphrase` ウォーターマーク文字列を復号化する際に使用するパスフレーズ（暗号化しない場合はなし）

また、ウォーターマークを抽出するための MSBuild タスクが定義されています。「PreMark タスク」を参照してください。

2.5.9 規則の編集インターフェイス

Dotfuscator の グラフィカル ユーザー インターフェイスは、アプリケーションで要素を対象として選択または対象から除外する規則をグラフィカルに指定するための、共通のインターフェイスを使用します。規則の編集インターフェイスは、以下の操作について規則を設定する場合に使用します。

- 名前の変更規則
- 制御フロー難読化対象除外規則
- 文字列暗号化対象選択規則
- 除去トリガー メソッドの選択規則
- 除去の条件付き対象の選択規則

このセクションでは、規則の編集インターフェイスを最大限に活用する方法を説明します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

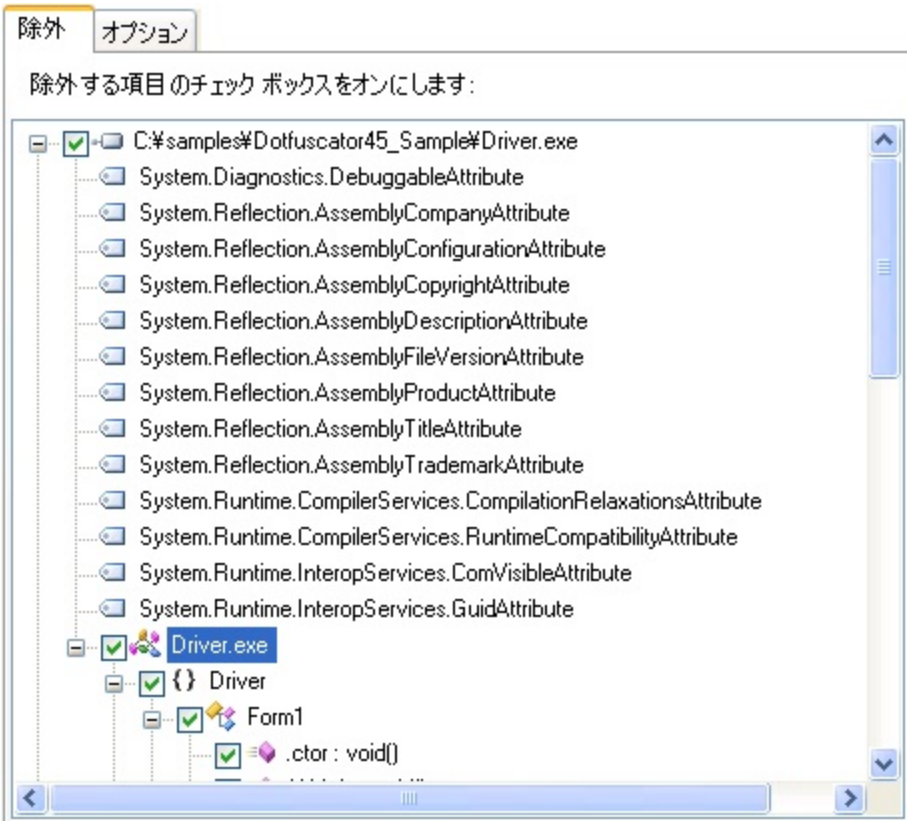
規則を作成するには、2 とおりの方法があります。1 つ目は、アプリケーションのツリー ビューで個々の要素のチェック ボックスをオンにする方法です。この操作によって、特定の要素を選択する単純な規則を作成できます。2 つ目は、規則の編集ビューに "ノード" を追加する方法です。この方法を使用すると、最初の方法の場合よりも高機能でカスタマイズ可能な規則を作成できます。規則の種類に応じて正規表現などの複数の選択基準を使い分けることも可能になります。また、この方法を使用すると、各規則で選択される項目をプレビューすることができます。

2.5.9.1 個々の要素の選択

アプリケーションのツリー ビューで、項目の隣にあるチェック ボックスをオンにして、個々の要素に対する選択規則を作成できます。隣にあるチェック ボックスをオンにすることによって、アセンブリ、モジュール、型、メソッド、およびフィールドを選択できます。

アセンブリ

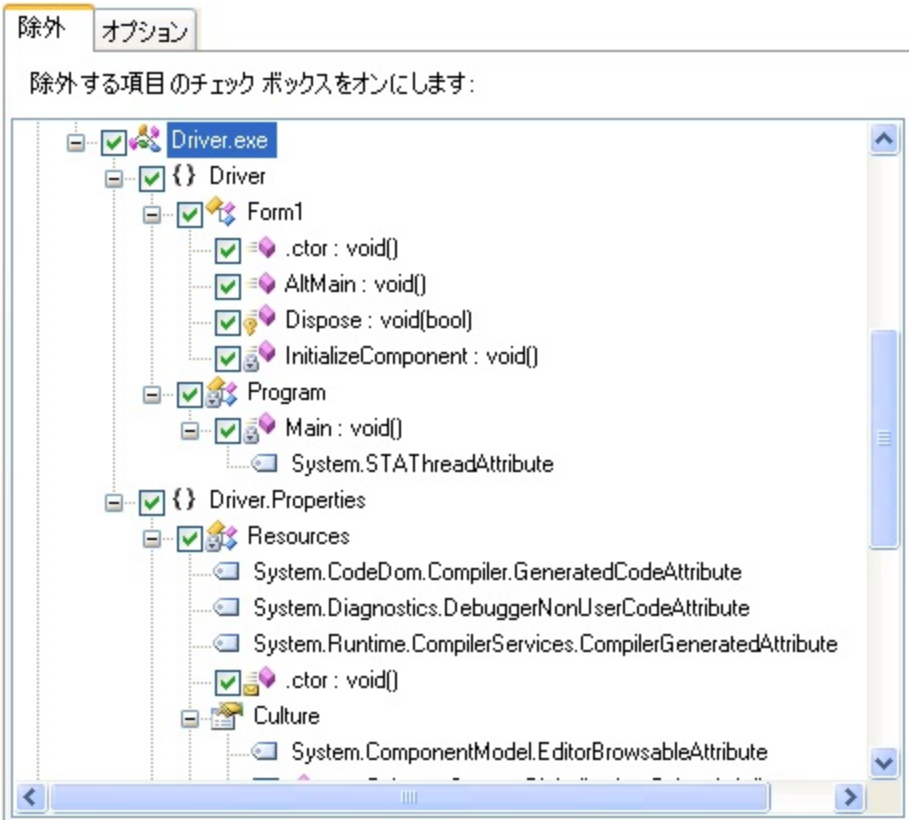
アプリケーションのツリー ビューの最上位ノードはパッケージまたはアセンブリを表します。アセンブリ ノードのチェック ボックスをオンにすると、すべての子ノードのチェック ボックスもオンになります。つまり、アセンブリを選択すると、アセンブリに含まれるすべての項目（モジュール、型、およびそのメンバー）を選択することになります。



© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

モジュール

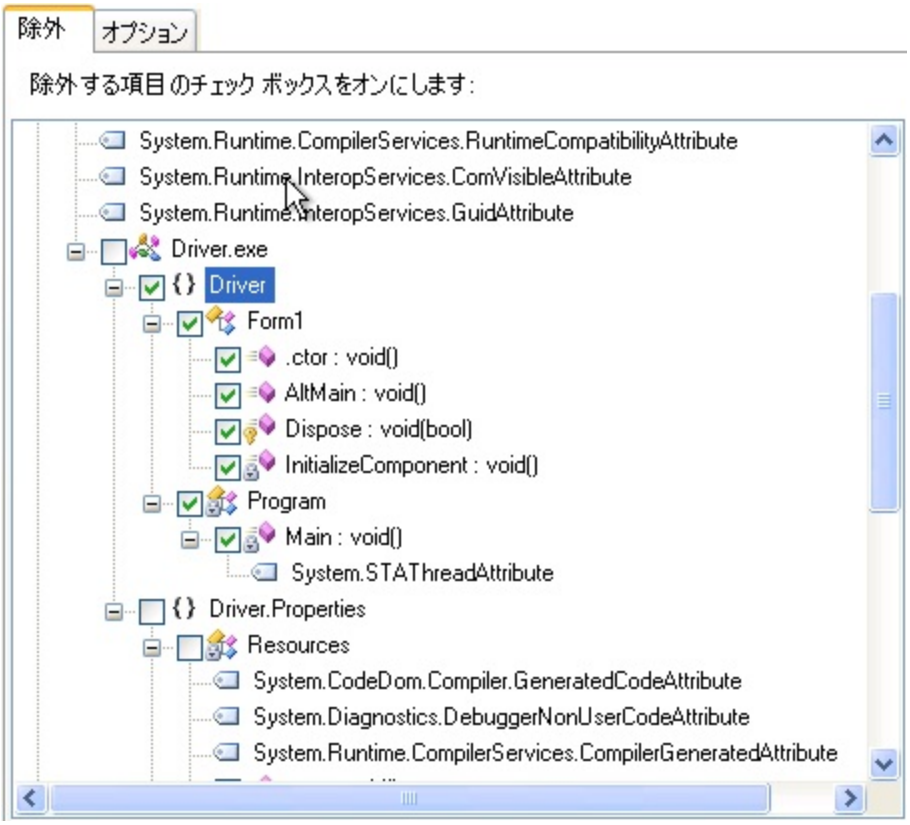
アプリケーションのツリー ビューでアセンブリ ノードの直下に表示されるノードは、アセンブリを構成するモジュールを表します（ほとんどの場合、各アセンブリに対して表示されるモジュールは 1 つです）。モジュール ノードのチェック ボックスをオンにすると、すべての子ノードのチェック ボックスもオンになります。つまり、モジュールを選択すると、モジュールに含まれるすべての項目（"グローバル" メソッド、フィールド、型、およびそのメンバー）を選択することになります。



名前空間

名前空間ノードは、アプリケーションのツリー ビューでモジュール ノードの子ノードとして表示されます。名前空間ノードのチェック ボックスをオンにすると、すべての子ノードのチェック ボックスもオンになります。つまり、名前空間を選択すると、その名前空間に含まれるすべての項目（型およびメンバー）を選択することになります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



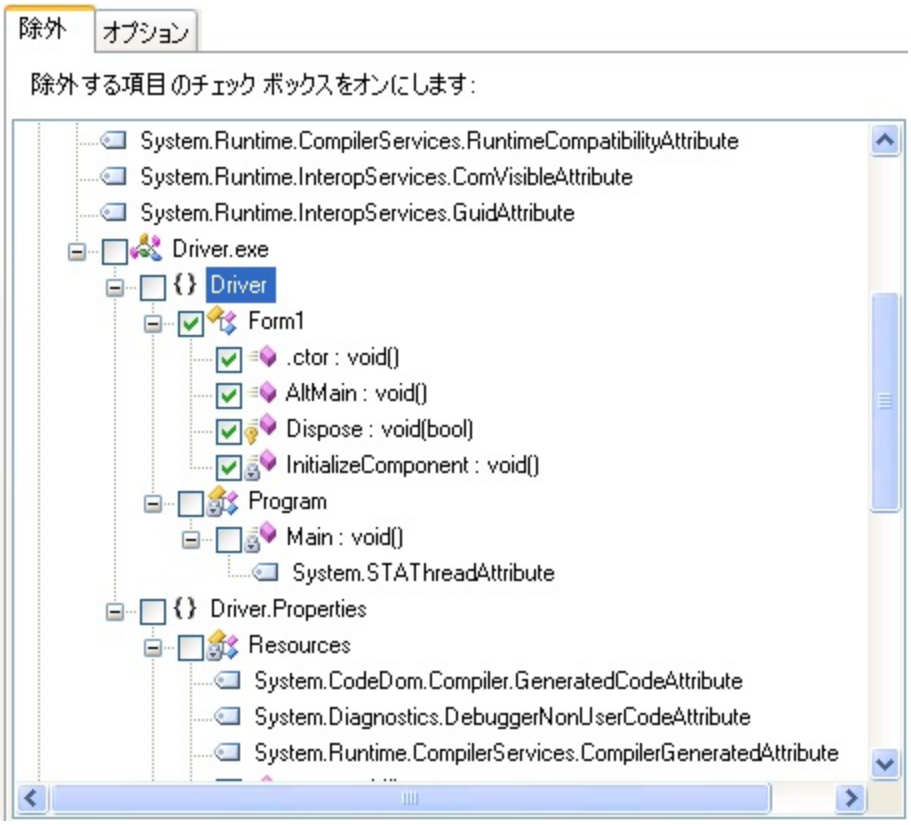
型

型ノードは、モジュール ノードまたは名前空間ノードの下に表示されます。入れ子になっている型もこのレベルに表示され、型名の前に '/' 文字で区切られた親の型名が付けられます。型ノードのチェック ボックスをオンにすると、作成している規則の種類に応じて、以下の 2 つのいずれかの状態になります。

名前の変更の対象除外規則を作成している場合、子ノードのチェック ボックスはオンになりません。つまり、名前の変更の対象除外規則の場合、型はメンバーから独立して選択されます。型ノードのチェック ボックスをオンにすると、**チェック ボックスをオンにした型名のみ**を名前の変更処理から除外する規則が生成されます。

その他の種類の規則を指定している場合は、すべての子ノードのチェック ボックスもオンになります。つまり、その他の規則の場合、型を選択すると、その型によって定義されているすべてのメンバーを選択することになります。

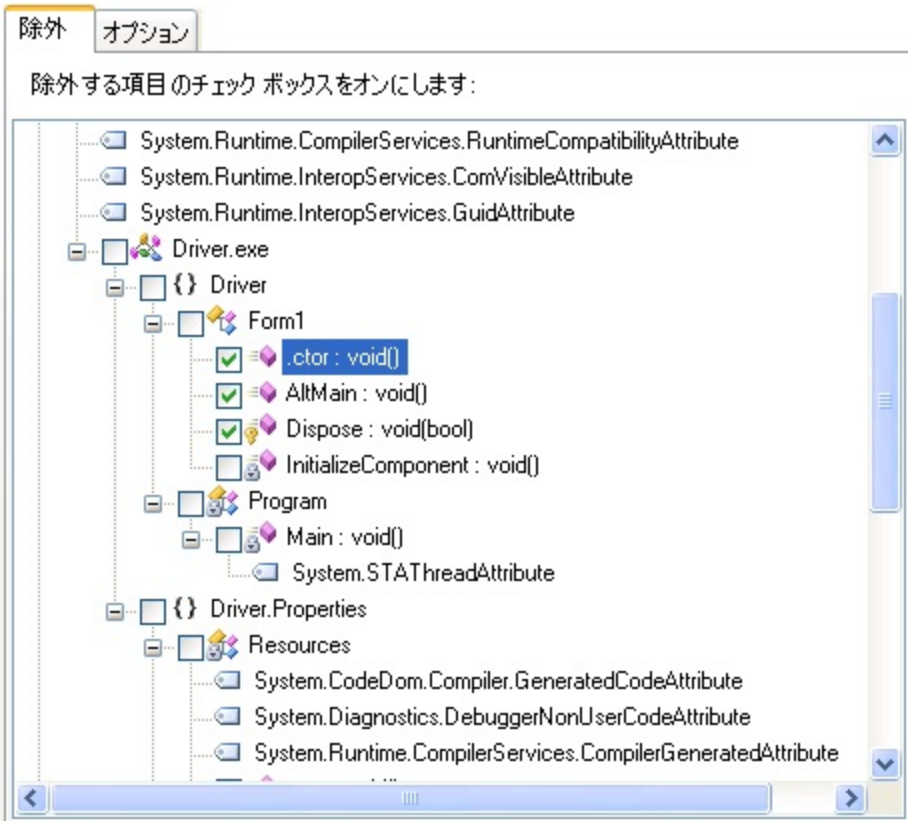
© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



メンバー

メンバーは、メソッド、フィールド、プロパティまたはイベントです。メンバー ノードは通常は型ノードの下に表示されますが、"グローバル" メンバーの場合はモジュール ノードの下に表示されることもあります。メンバー ノードのチェック ボックスをオンにすると、チェック ボックスをオンにしたメンバーを選択する規則が生成されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.5.9.2 カスタム規則の作成

規則の編集ビューにノードを追加して、カスタム規則を作成できます。規則の種類に応じて、正規表現や別の選択条件を規則に付加することができます。規則を構成したら、ノードを右クリックし、[**プレビュー**]をクリックして規則の結果をプレビューできます。規則で選択される項目は、アプリケーションのツリー ビューで網掛け表示になります。以下のセクションでは、それぞれの種類の規則について個別に説明します。

2.5.9.2.1 名前空間による選択

名前空間の規則を作成すると、条件を満たす名前空間に属するすべての型とそのメンバーが選択されます。

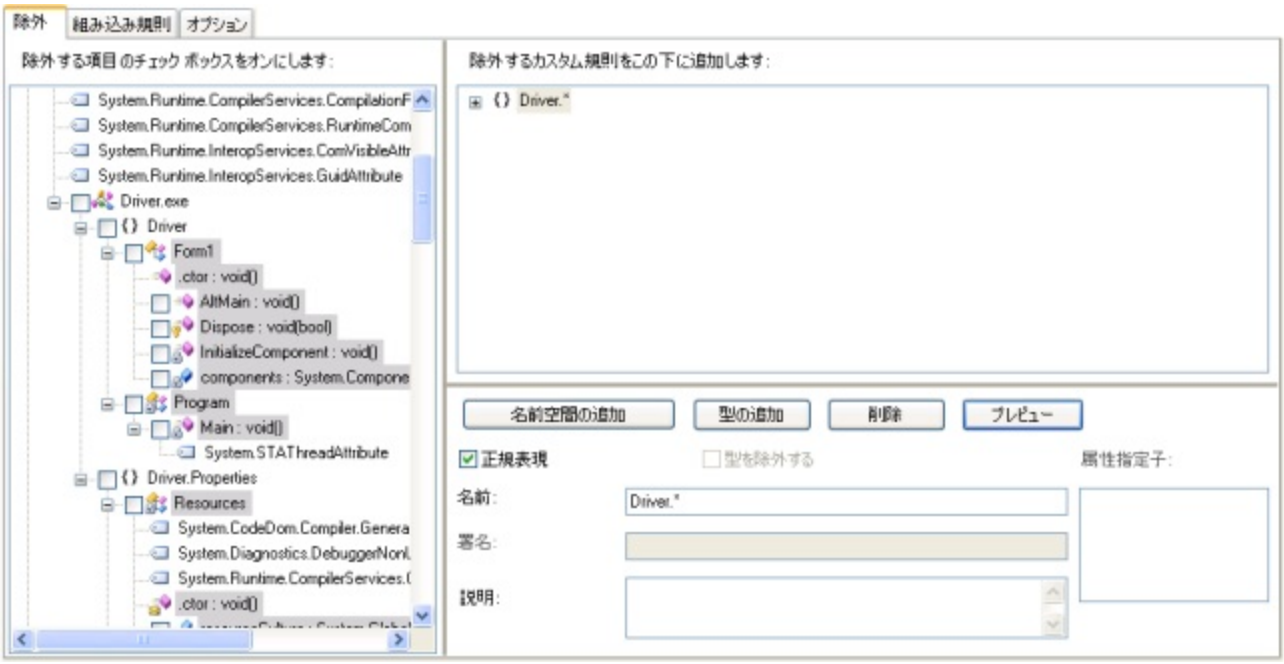
名前空間名

名前空間の規則を作成するには、[**名前空間の追加**]ボタンをクリックし、[**名前**]ボックスに名前を入力します。[正規表現]チェックボックスがオンになっている場合、入力された名前は正規表現として解釈されます。このチェック ボックスがオフになっている場合は、入力された名前は文字どおりに解釈されます(したがって、この場合は条件を満たす名前空間が複数存在することはありません)。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

名前空間の規則のノード

規則の編集ビューに表示される規則のノードは、その規則が正規表現であるかどうかを示す子要素を持っています。その規則で選択される項目をプレビューするには、ノードを右クリックし、表示されるメニューから[プレビュー]を選択します。



2.5.9.2.2 型による選択

型の規則によって選択される対象は、以下のように、作成している規則の種類によって異なります。

名前の変更の除外規則を作成している場合、[**型を除外する**]チェック ボックスがオンになっていると、指定した型名のみが除外の対象として選択されます。メンバーは除外の対象になりません。

その他の種類の規則を作成している場合、それらの規則では 0 個以上の型およびそのメンバーが選択されます。つまり、その他の規則の場合、型を選択すると、その型によって定義されているすべてのメンバーを選択することになります。

型名

型の規則を作成するには、[**型の追加**]ボタンをクリックし、[**名前**]ボックスに名前を入力します。[**正規表現**]チェック ボックスがオンになっている場合、入力された名前は正規表現として解釈されます。このチェック ボックスがオフになっている場合は、入力された名前は文字どおりに解釈されます。入れ子になっている型の場合、名前空間および親クラスの情報を含む完全修飾の型名を入力する必要があります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

型の属性指定子

型名に加えて、[**属性指定子**]ボックスの一覧に表示されている値を使用することにより、型の属性指定子に基づいて型を選択できます。属性指定子の前に付いている '-' は、その属性の否定を意味します(つまり、指定した属性を持たないすべての型を選択します)。一覧から複数の属性を選択することができます。複数の選択を行った場合に暗黙に適用される条件は、論理的に **AND** で結合されます。たとえば、一覧から **+public** と **+abstract** を選択することにより、パブリックかつ抽象的な型を選択できます。

属性の指定は、型名と論理的に **AND** で結合されます。したがって、特定の属性のセットを持つすべての型を選択する場合、すべての型を選択するような正規表現(つまり、".*")を型名として指定する必要があります。

[型を除外する]チェック ボックス

[**型を除外する**]チェック ボックスが有効になるのは、名前の変更の対象除外規則を作成している場合だけです。このチェック ボックスがオンになっている場合、条件を満たす型名は名前の変更対象から除外され、除外される型のメンバーを選択する追加の規則を指定できるようになります。このチェック ボックスがオフになっている場合、規則は、条件を満たす型のメンバーを選択しますが、型名自体は選択しません。このように、メソッドとフィールドは難読化の対象から除外し、型名だけを難読化する名前の変更の除外規則を記述できます。

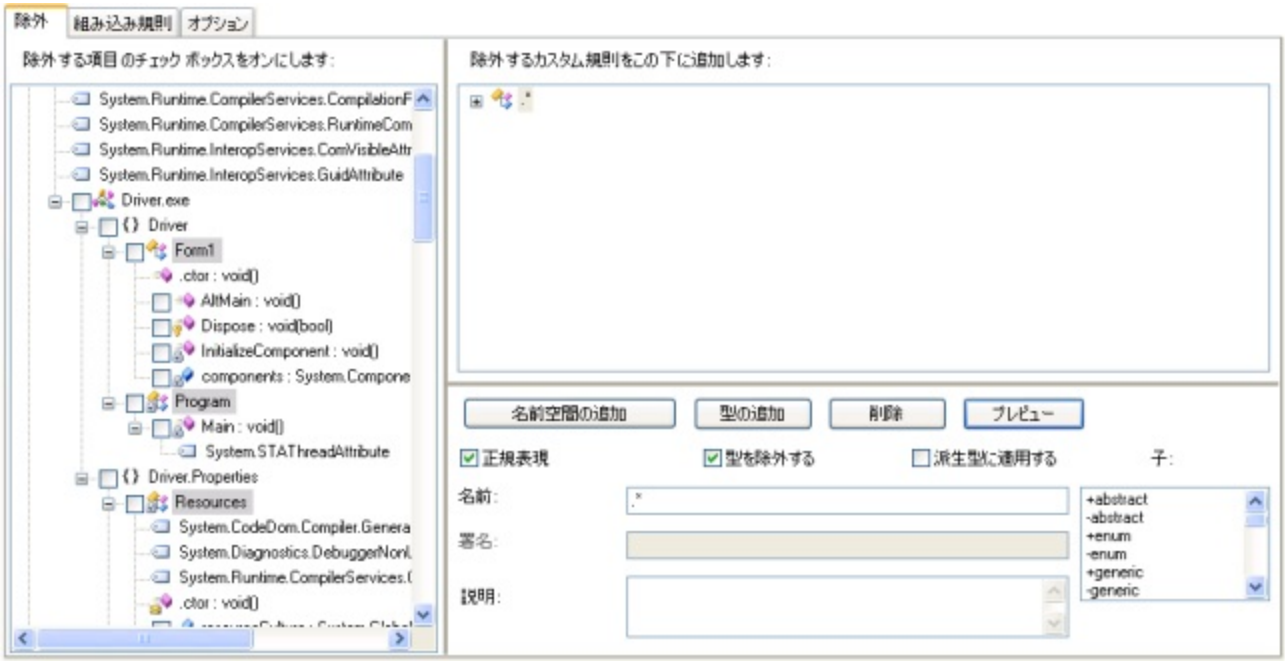
[派生型に適用する]チェック ボックス

[**派生型に適用する**]チェック ボックスが有効になるのは、名前の変更または不要コードの除去規則を作成している場合だけです。このチェック ボックスがオンになっている場合、条件を満たす型の子クラスも規則によって名前変更または不要コードの除去から除外されます。このように、継承関係にあるすべての階層を除外する名前の変更の除外規則を記述できます。

型の規則のノード

規則の編集ビューに表示される型の規則のノードは、その規則が正規表現かどうか、および規則と関連のある属性指定子が指定されているかどうかを示す子要素を持っています。その規則で選択される型をプレビューするには、ノードを右クリックし、表示されるメニューから[**プレビュー**]を選択します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



このスクリーン ショットでは、すべての具象型（抽象型ではない型）の名前を名前変更の対象から除外する型の規則が定義されています。

2.5.9.2.3 メソッドによる選択

メソッドの規則は、型の規則によって限定されるので、規則の編集ビューでは型ノードの子として表示されます。メソッドの規則は、（親の型の規則の条件を満たすすべての型のメソッドのうちで）条件を満たすすべてのメソッドを選択します。サポートされる選択条件は、メソッド名、メソッド属性、および署名です。

メソッド名

メソッドの規則を作成するには、親である型の規則のノードを右クリックして、[**メソッドの追加**]をクリックし、[**名前**]ボックスに名前を入力します。[**正規表現**]チェック ボックスがオンになっている場合、入力された名前は正規表現として解釈されます。このチェック ボックスがオフになっている場合は、入力された名前は文字どおりに解釈されます。

メソッドの属性指定子

メソッド名に加えて、[**属性指定子**]ボックスの一覧に表示されている値を使用することにより、メソッドの属性指定子に基づいてメソッドを選択できます。属性指定子の前に付いている '-' は、その属性の否定を意味します（つまり、指定した属性を持たないすべてのメソッドを選択します）。一覧から複数の属性を選択することもできます。複数の属性を選択した場合に暗黙に適用される条件は、論理的に AND で結合されます（つまり、選択されるメソッドのセットは、各属性指定子に一致するすべてのメソッドの共通部分です）。たとえば、一覧から **+public** と **+virtual** を選択することにより、パブリックかつ仮想のプロパティを選択できます。

属性の指定は、メソッド名および署名リストと論理的に **AND** で結合されます。したがって、特定の属性のセットを持つすべてのメソッドを選択する場合、すべてのメソッドを選択するような正規表現（つまり、"**.***"）をメソッド名として指定する必要があります。

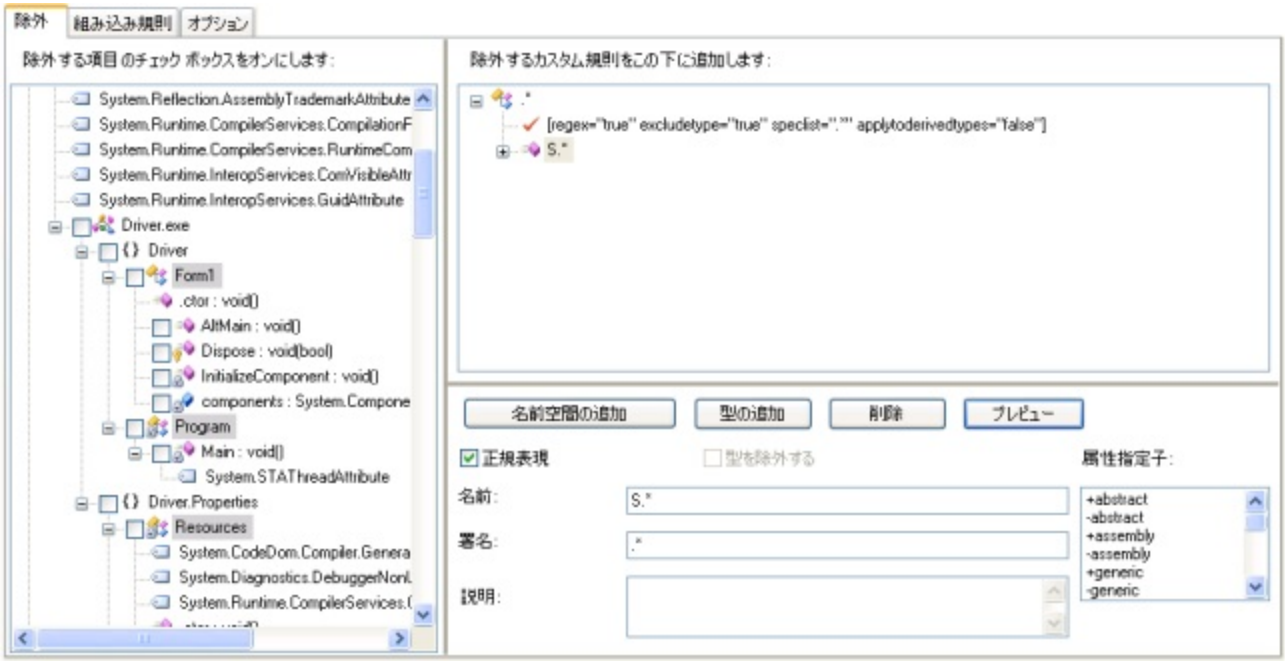
© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

メソッドの署名

メソッドを署名によって選択することもできます。署名は、メソッドの戻り値の型とパラメーターの型を特定します。メソッドの署名はメソッドの規則のスコープを縮小するので、署名に無関係にメソッドを選択する規則を作成する場合は、すべての署名（たとえば ".*"）を選択する正規表現を指定する必要があります。これが既定値です。

メソッドの規則のノード

規則の編集ビューに表示されるメソッド ノードは、その規則が正規表現かどうか、および規則と関連のある属性指定子および署名が指定されているかどうかを示す子要素を持っています。その規則で選択される項目をプレビューするには、ノードを右クリックし、表示されるメニューから［**プレビュー**］を選択します。



このスクリーンショットでは、名前が "**S**" で始まるすべてのパブリック メソッド（すべての型）の名前を選択するメソッドの規則が定義されています。

2.5.9.2.4 フィールドによる選択

フィールドの規則は、型の規則によって限定されるので、規則の編集ビューでは型ノードの子として表示されます。フィールドの規則は、（親の型の規則の条件を満たすすべての型のフィールドのうちで）条件を満たすすべてのフィールドを選択します。サポートされる選択条件は、フィールド名およびフィールドの属性です。

フィールド名

フィールドの規則を作成するには、親である型の規則のノードを右クリックして、［**フィールドの追加**］をクリックし、［**名前**］ボックスに名前を入力します。［**正規表現**］チェック ボックスがオンになっている場合、入力された名前は正規表現として解釈されます。このチェック ボックスがオフになっている場合は、入力された名前は文字どおりに解釈されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

フィールドの属性指定子

フィールド名に加えて、[**属性指定子**]ボックスの一覧に表示されている値を使用することにより、フィールドの属性指定子に基づいてフィールドを選択できます。属性指定子の前に付いている '-' は、その属性の否定を意味します(つまり、指定した属性を持たないすべてのフィールドを選択します)。一覧から複数の属性を選択することもできます。複数の属性を選択した場合に暗黙に適用される条件は、論理的に AND で結合されます(つまり、選択されるフィールドのセットは、各属性指定子に一致するすべてのフィールドの共通部分です)。たとえば、一覧から **+public** と **+static** を選択することにより、パブリックかつ静的なフィールドを選択できます。

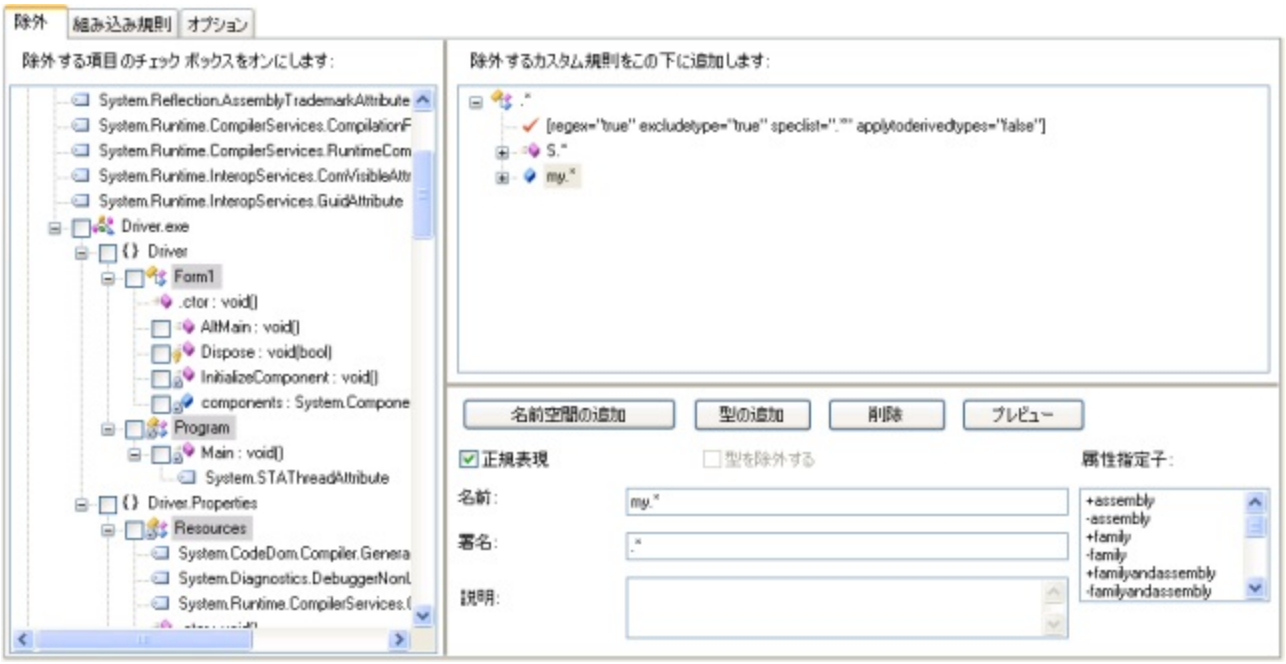
属性の指定は、フィールド名と論理的に **AND** で結合されます。したがって、特定の属性のセットを持つすべてのフィールドを選択する場合、すべてのフィールドを選択するような正規表現(つまり、"**.***")をフィールド名として指定する必要があります。

フィールドの署名

フィールドを署名によって選択することもできます。署名はフィールドのタイプを指定します。フィールドの署名はフィールドの規則のスコープを縮小するので、タイプに無関係にフィールドを選択する規則を作成する場合は、すべての署名(たとえば ".*")を選択する正規表現を指定する必要があります。これが既定値です。

フィールドの規則のノード

規則の編集ビューに表示されるフィールド ノードは、その規則が正規表現かどうか、および規則と関連のある属性指定子および署名が指定されているかどうかを示す子要素を持っています。その規則で選択されるフィールドをプレビューするには、ノードを右クリックし、表示されるメニューから[**プレビュー**]を選択します。



このスクリーンショットでは、名前が "**my**" で始まるすべてのフィールド(すべての型)の名前を選択するフィールドの規則が定義されています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.9.2.5 プロパティによる選択

プロパティの規則は、型の規則によって限定されるので、規則の編集ビューでは型ノードの子として表示されます。プロパティの規則は、(親の型の規則の条件を満たすすべての型のプロパティのうちで)条件を満たすすべてのプロパティを選択します。サポートされる選択条件は、プロパティ名およびプロパティの属性です。

プロパティ名

プロパティの規則を作成するには、親である型の規則のノードを右クリックして、[**プロパティの追加**]をクリックし、[**名前**]ボックスに名前を入力します。[**正規表現**]チェック ボックスがオンになっている場合、入力された名前は正規表現として解釈されます。このチェック ボックスがオフになっている場合は、入力された名前は文字どおりに解釈されます。

プロパティの属性指定子

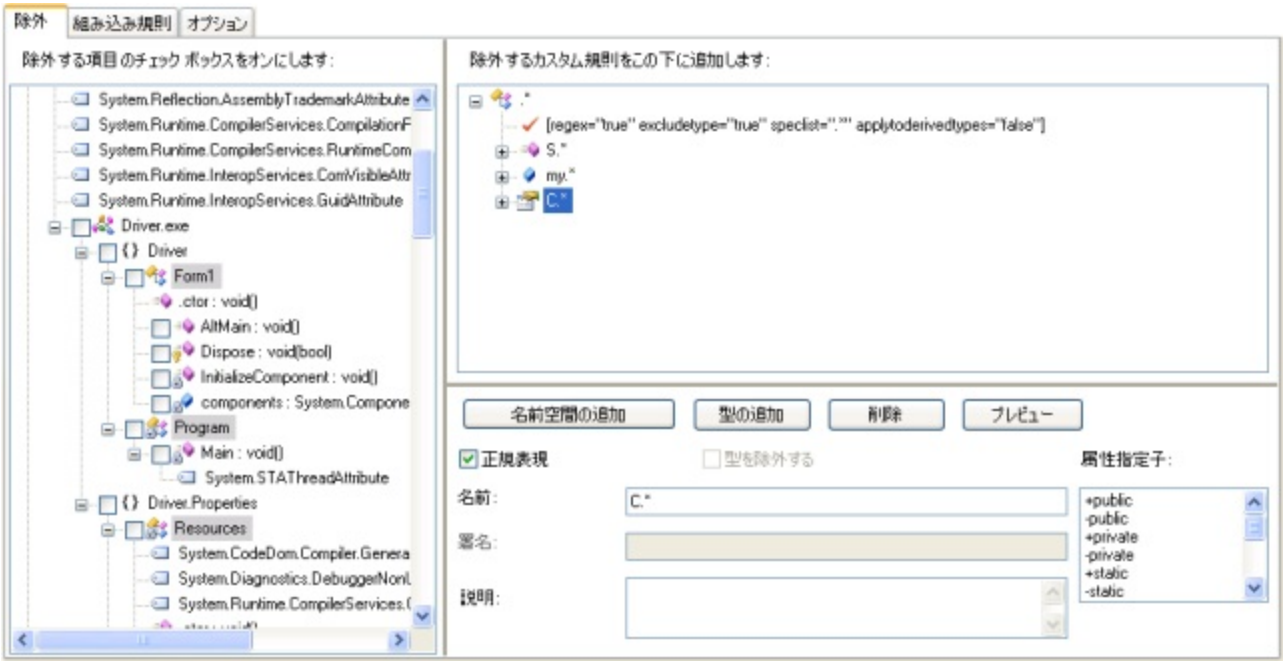
プロパティ名に加えて、[**属性指定子**]ボックスの一覧に表示されている値を使用することにより、プロパティの属性指定子に基づいてプロパティを選択できます。属性指定子の前に付いている '-' は、その属性の否定を意味します(つまり、指定した属性を持たないすべてのプロパティを選択します)。一覧から複数の属性を選択することもできます。複数の属性を選択した場合に暗黙に適用される条件は、論理的に AND で結合されます(つまり、選択されるプロパティのセットは、各属性指定子に一致するすべてのプロパティの共通部分です)。たとえば、一覧から **+public** と **+virtual** を選択することにより、パブリックかつ仮想のプロパティを選択できます。

属性の指定は、プロパティ名および署名リストと論理的に **AND** で結合されます。したがって、特定の属性のセットを持つすべてのプロパティを選択する場合、すべてのプロパティを選択するような正規表現(つまり、"**.***")をプロパティ名として指定する必要があります。

プロパティの規則のノード

規則の編集ビューに表示されるプロパティの規則のノードは、その規則が正規表現かどうか、および規則に属性指定子が指定されているかどうかを示す子要素を持っています。その規則で選択される項目をプレビューするには、ノードを右クリックし、表示されるメニューから[**プレビュー**]を選択します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



このスクリーンショットでは、名前が "**C**" で始まるすべてのパブリック プロパティ(すべての型)の名前を選択するプロパティの規則が定義されています。

2.5.9.2.6 イベントによる選択

イベントの規則は、型の規則によって限定されるので、規則の編集ビューでは型ノードの子として表示されます。イベントの規則は、(親の型の規則の条件を満たすすべての型のフィールドのうちで)条件を満たすすべてのイベントを選択します。サポートされる選択条件は、イベント名およびイベントの属性です。

イベント名

イベントの規則を作成するには、親である型の規則のノードを右クリックして、[**イベントの追加**]をクリックし、[**名前**]ボックスに名前を入力します。[**正規表現**]チェック ボックスがオンになっている場合、入力された名前は正規表現として解釈されます。このチェック ボックスがオフになっている場合は、入力された名前は文字どおりに解釈されます。

イベントの属性指定子

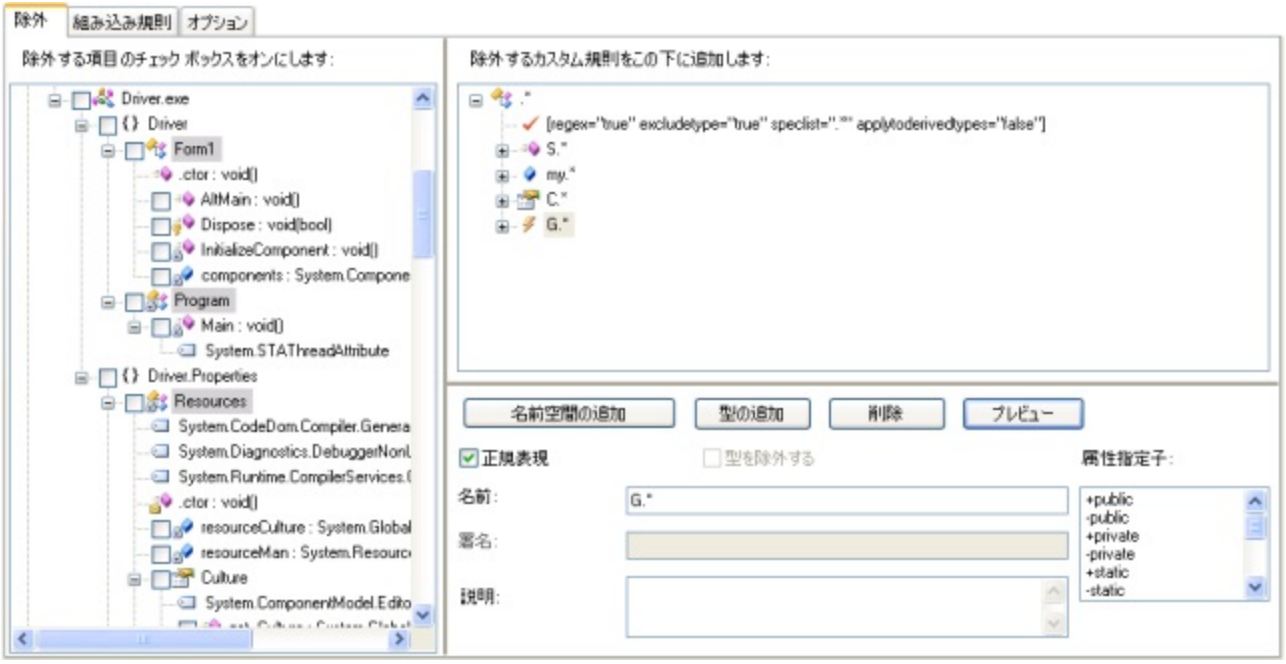
イベント名に加えて、[**属性指定子**]ボックスの一覧に表示されている値を使用することにより、イベントの属性指定子に基づいてイベントを選択できます。属性指定子の前に付いている '-' は、その属性の否定を意味します(つまり、指定した属性を持たないすべてのイベントを選択します)。一覧から複数の属性を選択することもできます。複数の属性を選択した場合に暗黙に適用される条件は、論理的に AND で結合されます(つまり、選択されるイベントのセットは、各属性指定子に一致するすべてのイベントの共通部分です)。たとえば、一覧から **+public** と **+static** を選択することにより、パブリックかつ静的なイベントを選択できます。

属性の指定は、イベント名と論理的に **AND** で結合されます。したがって、特定の属性のセットを持つすべてのイベントを選択する場合、すべてのイベントを選択するような正規表現(つまり、".*")をイベント名として指定する必要があります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

イベントの規則のノード

規則の編集ビューに表示されるイベントの規則のノードは、その規則が正規表現かどうか、および規則に属性指定子が指定されているかどうかを示す子要素を持っています。その規則で選択されるイベントをプレビューするには、ノードを右クリックし、表示されるメニューから[**プレビュー**]を選択します。



このスクリーンショットでは、名前が "**G**" で始まるすべてのイベント(すべての型)の名前を選択するイベントの規則が定義されています。

2.5.9.2.7 カスタム属性による選択

カスタム属性の規則は、型、メソッド、フィールド、プロパティまたはイベントの規則によって限定されるので、規則の編集ビューでは型、メソッド、フィールド、プロパティまたはイベントのノードの子として表示されます。カスタム属性の規則は、親ノードによって選択されていて、かつ規則の条件を満たすカスタム属性が指定されているすべての項目を選択します。

カスタム属性名

カスタム属性の規則を作成するには、親である型、メソッド、フィールド、プロパティまたはイベントの規則のノードを右クリックして、[**カスタム属性の追加**]をクリックし、[**名前**]ボックスに名前を入力します。[**正規表現**]チェック ボックスがオンになっている場合、入力された名前は正規表現として解釈されます。このチェック ボックスがオフになっている場合は、入力された名前は文字どおりに解釈されます。

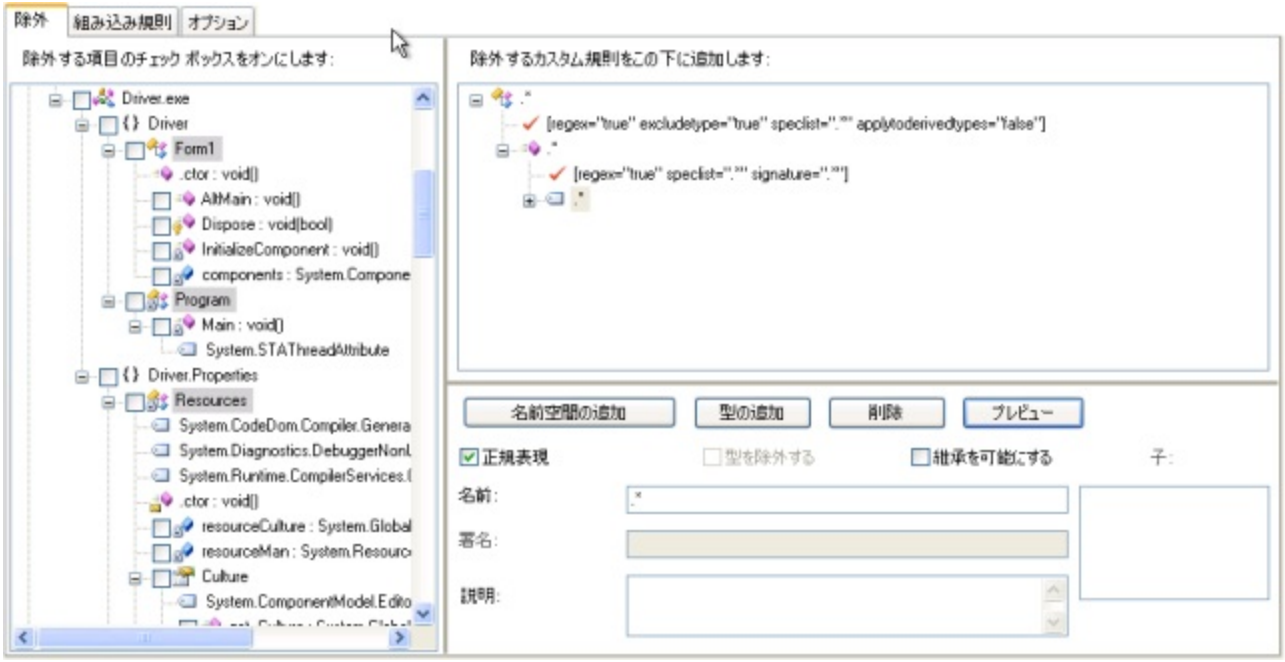
[継承を可能にする]チェック ボックス

[**継承を可能にする**]チェック ボックスは、カスタム属性の規則を継承階層に適用する方法を制御します。このチェック ボックスがオンになっている場合、オーバーライドするメソッド、プロパティ、イベント、および子の型も除外されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

カスタム属性の規則のノード

規則の編集ビューに表示されるカスタム属性のノードは、その規則が正規表現であるかどうかを示す子要素を持っています。カスタム属性の規則のノードでは、ノードを右クリックして、メニューから[**プレビュー**]オプションを選択することにより、その規則で選択される型、メソッド、フィールド、プロパティ、またはイベントをプレビューできます。



このスクリーン ショットでは、**GettingStarted.TestNamedAttribute** という名前のカスタム属性が指定されているすべてのメソッドを選択するカスタム属性の規則が定義されています。

2.5.9.2.8 スーパータイプによる選択

スーパータイプの規則は、型の規則によって限定されるので、規則の編集ビューでは型ノードの子として表示されます。スーパータイプ規則は、親の型の規則の条件に一致し、かつ指定したスーパータイプから派生する型のみが条件に一致するよう、型の規則のスコープを狭くします。

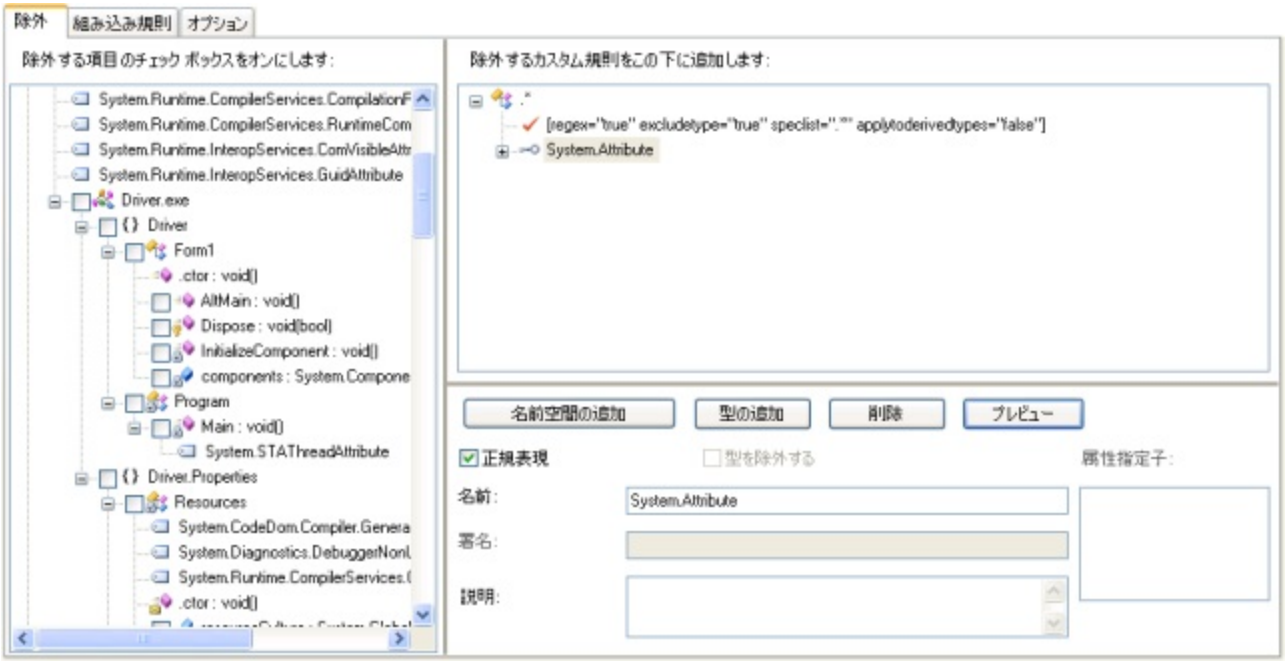
スーパータイプ名

スーパータイプの規則を作成するには、親である型の規則のノードを右クリックして、[**スーパータイプの追加**]をクリックし、[**名前**]ボックスに名前を入力します。[**正規表現**]チェック ボックスがオンになっている場合、入力された名前は正規表現として解釈されます。このチェック ボックスがオフになっている場合は、入力された名前は文字どおりに解釈されます。

スーパータイプの規則のノード

規則の編集ビューに表示されるスーパータイプの規則のノードは、その規則が正規表現であるかどうかを示す子要素を持っています。その規則で選択される型をプレビューするには、ノードを右クリックし、表示されるメニューから[**プレビュー**]を選択します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



このスクリーンショットでは、**System.Attribute** のスーパータイプであるすべての型を選択するスーパータイプ規則が定義されています。

2.5.9.3 規則の編集と削除

既存の規則を編集するには、規則の編集ビューで目的の規則をクリックします。次に、ビューの下部にあるコントロールを使用して、ノードに関連する値（たとえば、名前、属性指定子リストなど）を編集します。

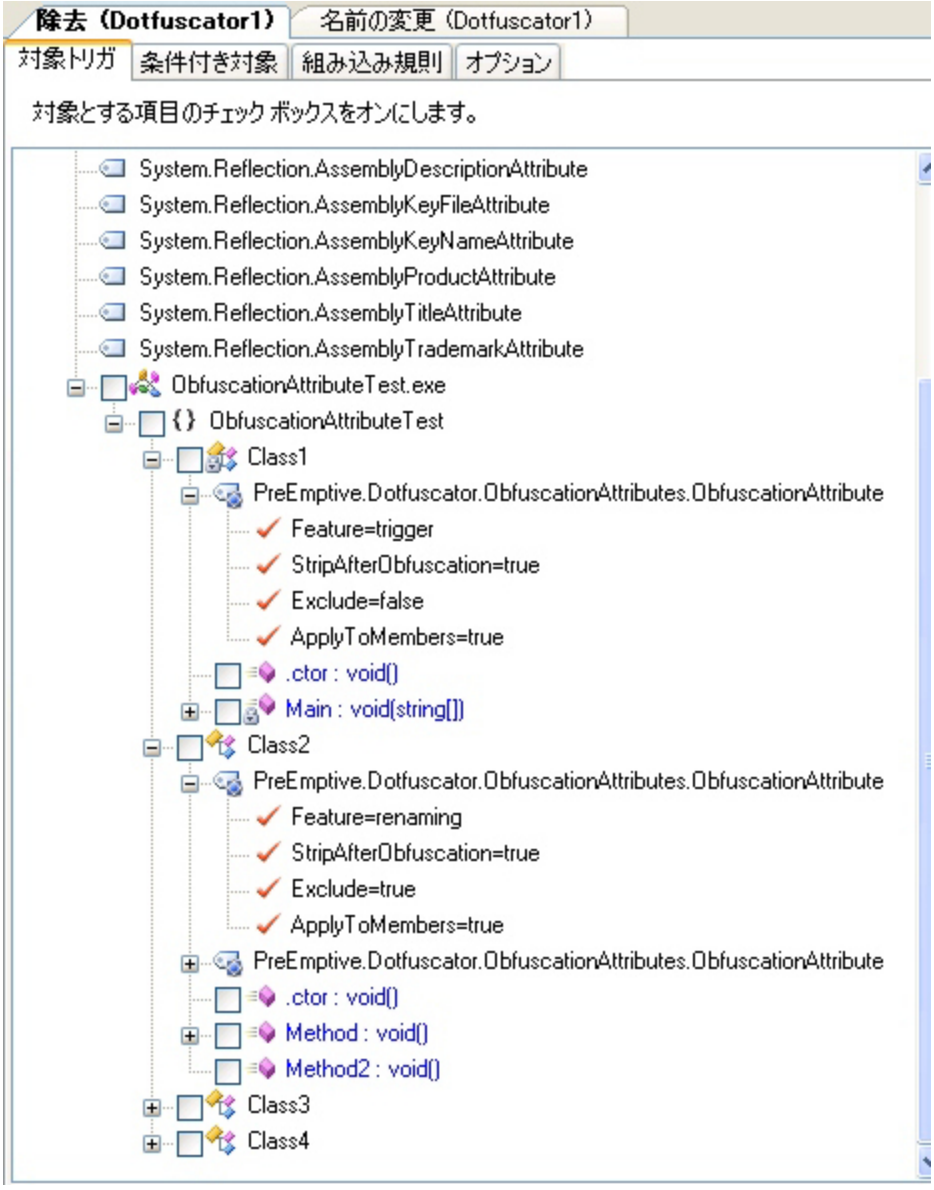
規則を削除するには、規則の編集ビューで目的の規則をクリックし、[**削除**]をクリックします。

2.5.9.4 規則での宣言による難読化の使用

規則のエディターでは、宣言による難読化がサポートされており、アプリケーションのツリー ビューにはすべての難読化属性（つまり、`System.Reflection.ObfuscateAssemblyAttribute` および `System.Reflection.ObfuscationAttribute`）の引数が表示されます。アプリケーションのツリー ビューでは、難読化属性によって選択されている項目（型、メソッド、フィールド）は青色で表示されます。

次のスクリーン ショットでは、`ObfuscationAttributes` を使用して、Class1 および Class2 のメソッドを除去トリガーとして指定しています。また、このビューでは、各属性のプロパティと値が展開表示されています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.5.9.5 規則のプレビュー

規則の編集ビューで 1 つの規則の結果をプレビューするには、規則のノードをクリックして、[**プレビュー**]をクリックします。規則で選択される項目は、アプリケーションのツリー ビューで網掛け表示になります。

規則の編集ビューで定義されている規則をすべて適用した場合の結果をプレビューするには、[**プレビュー**]ボタンをクリックします。すべての規則を適用した場合に選択されるすべての項目が、アプリケーションのツリー ビューで網掛け表示になります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.10 インストルメンテーション（改ざん、Shelf Life、例外、分析）

[**インストルメンテーション**]タブでは、Dotfuscator のコードの差し込み機能（**インストルメンテーション**）を設定する、カスタム属性の追加、編集、および調査が行えます。次の機能が含まれます。

- Shelf Life
- 改ざん通知
- 例外の追跡
- アプリケーション分析（Runtime Intelligence）

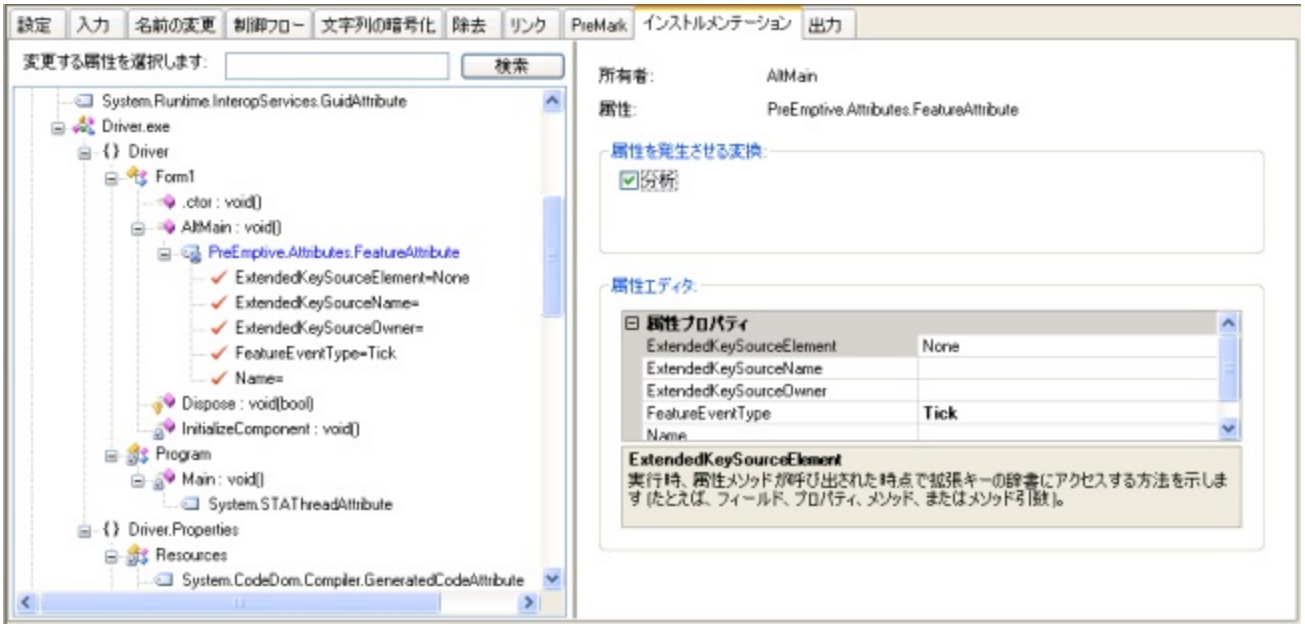
> 新規プロジェクトの場合、インストルメンテーションの変換に対する既定の設定は「**有効化**」になっています。

ソース コード内で既にサポートされているカスタム属性は、ユーザー インターフェイスを介して編集できます。変更内容は Dotfuscator 構成ファイルに保持され、ソース コード内の属性に優先します。

拡張カスタム属性もまた、ユーザー インターフェイスを介して追加および編集することができます。Dotfuscator は、拡張属性をソース コードに埋め込まれているカスタム属性と同様に扱います。

また、サポートされる属性であれば、それがコードに埋め込まれている属性であっても、サポートされている特定のコード変換に割り当てることができます。これは属性のオーバーロードをサポートしており、同じ属性のセットで複数の変換処理（たとえば、アプリケーション分析）を作動させることができます。

属性を追加するには、設定したいメソッドまたはアセンブリへ移動して、目的の属性を選択します。属性に引数がある場合は、[インストルメンテーション]エディターでそれを設定できます。

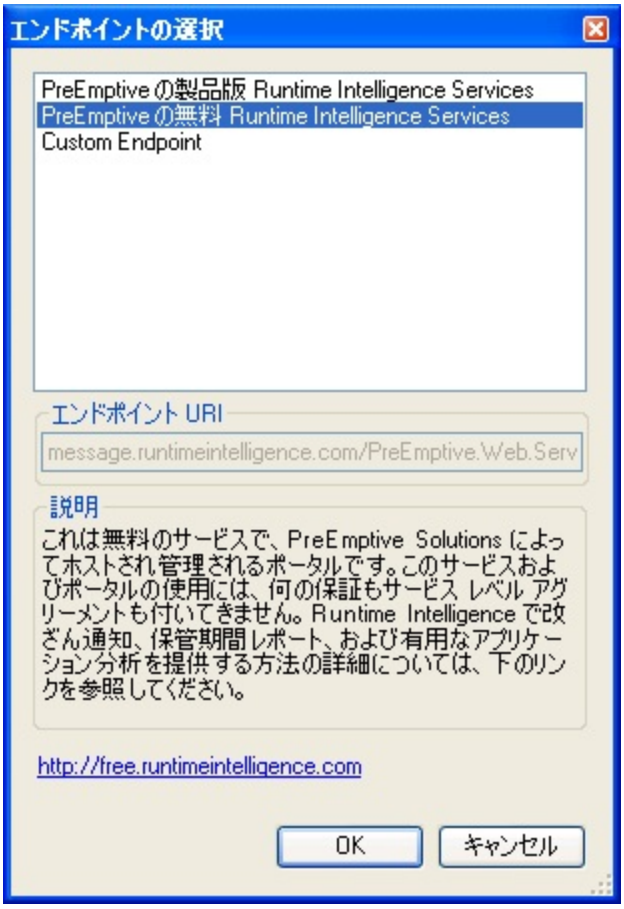


[**属性を発生させる変換**]セクションの[**分析**]チェック ボックスをオンにすると、属性付きメソッドを Runtime Intelligence で追跡される機能としてインストルメントします。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

カスタム エンドポイント

SetupAttribute の**属性エディター**内の **CustomEndpoint** フィールドで Runtime Intelligence Services メッセージのエンドポイントを明示的に指定します。カスタム エンドポイントを選択しないと、PreEmptive の製品版 Runtime Intelligence Services エンドポイントが使用されます。**CustomEndpoint** フィールドの[**...**]（省略記号）をクリックすると[エンドポイントの選択]ウィンドウが開きます。



次の 3 つのうち 1 つを選択することができます。

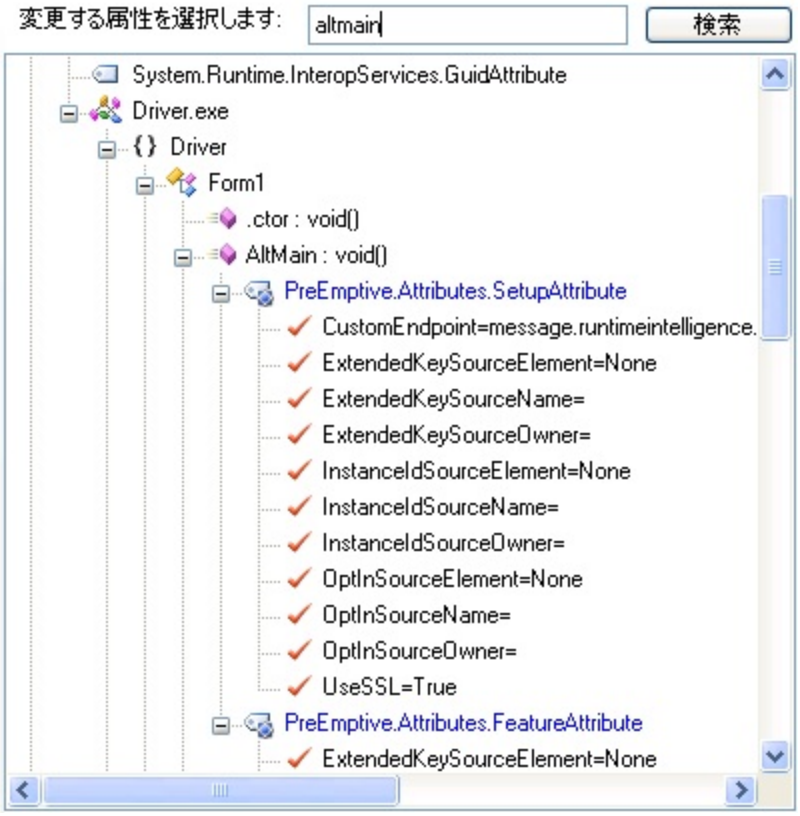
- PreEmptive の製品版 Runtime Intelligence Services
- PreEmptive の無料の Runtime Intelligence Services
- カスタム エンドポイント - ユーザー定義のカスタム エンドポイント

各オプションをクリックすると、[説明]フィールドにエンドポイントの説明が表示されます。選択を行って[OK]をクリックすると、CustomEndpoint フィールドにその場所が表示されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

属性検索フィールド

[**インストルメンテーション**]タブの**検索**機能を使用すると、インストルメンテーション ツリー ビューの項目を簡単に見つけることができます。[**変更する属性を選択します**]フィールドに探したい項目の名前の全部または最初の何文字かを入力し、[**検索**]をクリックします。また、名前の最初の数文字の後ろに **.***(ワイルドカード)を入力するか、正規表現を入力して項目を検索することもできます。Dotfuscator はその検索語が現れる最初の項目を選択します。[**検索**]ボタンをクリックすると、検索語に次に一致する項目を選択します。検索は項目名の最初に一致するため、インストルメンテーション属性を検索する場合は属性の名前全部を使用する必要があります。たとえば、Setup 属性を検索するには **PreEmptive.Attributes.SetupAttribute** と入力します。



© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.10.1 Shelf Life トークンの概要

Shelf Life（使用期限）はアプリケーションの在庫管理機能で、期限切れおよび通知のロジックをアプリケーションへ埋め込むことを可能にするものです。Dotfuscator は、アプリケーションの期限切れに対し、アプリケーションを終了したり Runtime Intelligence Service メッセージを送信したりして対処するためのコードを差し込みます。この機能は、ベータ版のアプリケーションでは特に有用です。ユーザーは、アプリケーションの期限切れや非アクティブ化を特定の日にスケジュールできます。また、オプションとして、特定の日数後にアプリケーションの期限が切れるか非アクティブ化される警告をユーザーに発することができます。

Shelf Life トークンは、アプリケーションおよび期限切れ情報を含んでいる暗号化されたデータのセットで、難読化時にアプリケーションのバイナリに差し込まれます。あるいは、それが有効期限を延長するために更新されることができるように、アプリケーションバイナリの外部にストリングして保存することができます。

秘密キー ファイルとパスワードの使用は任意ですが、使用すれば、ユーザー指定の秘密キーでトークン データに署名することにより、Shelf Life トークンの防御がさらに強力なものとなります。署名の検証を提供するのに必要な公開キー情報のみが Shelf Life トークンに格納されます。

アプリケーションに差し込まれる Shelf Life ランタイム コードに Shelf Life トークンを提供する方法は 2 つあります。既定の方法は、Dotfuscator が難読化処理中に暗号化データをバイナリに直接埋め込む方法です。もう 1 つの方法は、アプリケーション開発者が Shelf Life トークン データの独自の持続化を図り、その Shelf Life トークン データの文字列表記をカスタム メソッドやプロパティなど、ShelfLifeTokenSource プロパティに設定されている要素を介して、差し込まれたランタイム コードに提供する方法です。Shelf Life トークン データがバイナリの外部に存在する利点は、新しくインストルメント化したバイナリを配布することなく簡単に使用期限を延長できることですが、その代わりに Shelf Life トークン データがユーザーにより見えやすくなります。

2.5.10.1.1 Shelf Life Activation Key の概要

Shelf Life Activation Key（**SLAK**）は、Shelf Life 機能を Dotfuscator に導入し、インストルメントするアプリケーション内の適切な場所に差し込むために必要なデータ ファイルです。

Shelf Life Activation Key は PreEmptive によって発行され、使用期限の構成中、ユーザーによって Dotfuscator に渡されます。Shelf Life Activation Key を入手するには、PreEmptive Solutions にご連絡ください。PreEmptive は、お客様のビルド マシンに格納する Shelf Life Activation Key を含むデータ ファイルを発行します。

Shelf Life Activation Key を入手したら、1 つまたは複数のメソッドに Shelf Life 属性を追加することができます。[**インストルメンテーション**]タブにある[**属性エディター**]セクションに表示される **ActivationKeyFile** フィールドで、Shelf Life Activation Key ファイルのパスを選択する必要があります。そうすると、アプリケーション内で Shelf Life がアクティブ化されます。

2.5.10.1.2 期限切れと警告の操作

ユーザーは、期限が切れたり警告したりする際に起こす操作を構成によって、またはカスタム属性を使って指定することができます。ユーザーが既定の操作を選択した場合、アプリケーションは警告期間中に何の行動も取らず、期限が切れたら終了します。`InsertShelfLife` 属性を使用すると、Shelf Life の警告または期限切れの発生時にユーザーが指定したメソッドへのコールバックを実行するコードを差し込むことができます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

期限切れの通知

これを行うには、**InsertShelfLife** 属性を使用して **ExpirationNotificationSink** を指定します。Dotfuscator はこれを使用して Shelf Life（使用期限）の満了チェックの結果をアプリケーションに返すコードを生成します。**ExpirationNotificationSink** には、書き込み可能なブール値で評価されるプロパティまたはフィールド、あるいは **void(bool)** または **void(string,string)** 署名付きのメソッドまたはデリゲートを指定できます。Shelf Life の期限切れチェック後、アプリケーションの期限が切れていれば生成されたコードはブール値を true に設定し、そうでなければ false に設定します。**void(string,string)** 署名を持つメソッドまたはデリゲート シンクを使用する場合、生成されたコードはこのシンクを呼び出し、警告日付と満了日付を文字列引数としてこの順序で渡します。アプリケーションは、Shelf Life の期限切れ通知にどのような方法で対応してもかまいません。**InsertShelfLife** 属性には、**ExpirationNotificationSink** を指定するための 3 つのプロパティが定義されています。

- **ExpirationNotificationSinkElement**
- **ExpirationNotificationSinkOwner**
- **ExpirationNotificationSinkName**

これらのプロパティについては、「カスタム属性リファレンス」の InsertShelfLifeAttribute セクションで詳しく説明します。

期限切れ通知シンクの設定は省略可能です。省略すると、期限切れチェックが実行されてもアプリケーションに通知されません。

ExpirationNotificationSinkElement に **DefaultAction** を設定すると、Dotfuscator は Shelf Life 期限切れが検出された場合にアプリケーションを終了するコードを差し込みます。

警告の通知

これを行うには、**InsertShelfLife** 属性を使用して **WarningNotificationSink** を指定します。Dotfuscator はこれを使用して Shelf Life（使用期限）警告チェックの結果をアプリケーションに返すコードを生成します。**WarningNotificationSink** には、書き込み可能なブール値で評価されるプロパティまたはフィールド、あるいは **void(bool)** または **void(string,string)** 署名付きのメソッドまたはデリゲートを指定できます。Shelf Life の警告チェック後、アプリケーションが警告期間にある場合には、生成されたコードはブール値を true に設定し、そうでなければ false に設定します。**void(string,string)** 署名を持つメソッドまたはデリゲート シンクを使用する場合、生成されたコードはこのシンクを呼び出し、警告日付と満了日付を文字列引数としてこの順序で渡します。アプリケーションは、Shelf Life（使用期限）の警告通知にどのような方法で対応してもかまいません。**InsertShelfLife** 属性には、**WarningNotificationSink** を指定するための 3 つのプロパティが定義されています。

- **WarningNotificationSinkElement**
- **WarningNotificationSinkOwner**
- **WarningNotificationSinkName**

これらのプロパティについては、「カスタム属性リファレンス」の InsertShelfLifeAttribute セクションで詳しく説明します。

警告通知シンクの設定は省略可能です。省略すると、警告チェックが実行されてもアプリケーションに通知されません。

WarningNotificationSinkElement に **DefaultAction** を設定すると、Dotfuscator は警告通知を処理するコードを差し込みません。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

InsertShelfLife 属性の **ExpirationNotificationSink** および **WarningNotificationSink** を、属性付きメソッドと同じクラスに定義されているインスタンス フィールドとする場合の使用例を示します。

InsertShelfLife 属性の WarningNotificationSink と ExpirationNotificationSink の使用法

```
class ShelfLifeSample {
   bool instanceShelfLifeExpirationFlag;

   [PreEmptive.Attributes.InsertShelfLife(
      ActivationKeyFile = "C:\\shelflife.slkey",
      ExpirationDate = "2009-11-05",
      ExpirationNotificationSinkElement = SinkElements.Field,
      ExpirationNotificationSinkName = "instanceShelfLifeExpirationFlag",
      WarningDate = "2009-07-04",
      WarningNotificationSinkElement = SinkElements.Method,
      WarningNotificationSinkName = "CheckShelfLifeWarning"
   )]
   private void Verify() {
     // Dotfuscator は、Shelf Life の期限切れと警告の検出および通知のコードをここに追加します
   }
   private CheckShelfLifeExpirationState() {
      if ( instanceShelfLifeExpirationFlag ) {
         // アプリケーションの期限が切れています
      }
      else {
         // アプリケーションの期限は切れていません
      }
   }
   private CheckShelfLifeWarning(string warnDate, string expDate) {
      // 日付文字列を直接使用します
      Console.WriteLine("MyApp expires on " + expDate);

      // または、日付を分析して計算を行います
      int daysLeft = DateTime.Parse(expDate).Subtract(DateTime.Now);
      Console.WriteLine("MyApp will expire in " + daysLeft + " days");
   }
}
```

2.5.10.1.3 期限切れと警告のレポート

警告期間に入るとき、または期限切れになったときに、挿入された Shelf Life コードは Runtime Intelligence Service にメッセージを送信することができます。この機能を有効にしたい場合は、[**設定**]タブの[**グローバル オプション**]セクションにある[**Shelf Life メッセージの送信**]を有効(はい)にします。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

使用期限切れを検出するには、アプリケーション内で必ず実行される 1 つまたは複数のメソッドに **InsertShelfLife** 属性を設定します。Dotfuscator は処理中に **InsertShelfLife** 属性を見つけると、実行時に期限切れ検出を実行するためのコードを追加します。現在の日付が、埋め込まれている期限期日であるかそれを過ぎている場合、Shelf Life の期限切れメッセージが Runtime Intelligence Service に送信されます。

Dotfuscator では、任意でアプリケーションの使用期限切れの前に警告期間を指定することができます。現在の日付が、埋め込まれている警告期日であるかそれを過ぎている場合、Shelf Life の警告メッセージが Runtime Intelligence Service に送信されます。

InsertShelfLife 属性は実行時には必要ありません。そのため、Dotfuscator はこの属性を出力アプリケーションから取り除きます。

アプリケーションに **InsertShelfLife** 属性をいくつ含めてもかまいません。アプリケーションの使用期限期日であるか間もなく期日である場合は、同じアプリケーション セッションから同じ Shelf Life ID を持つ複数の Shelf Life メッセージが送信されます。

この属性を Setup 属性が含まれているメソッドに置かないでください。この属性を含んでいるメソッドは、Setup 属性を含んでいるメソッドの後に実行する必要があります。

InsertShelfLife 属性

```
[PreEmptive.Attributes.InsertShelfLife(
   ActivationKeyFile = "C:\\shelflife.slkey",
   ExpirationDate = "2009-11-05"
)]
public void DoStuff() { ...}
```

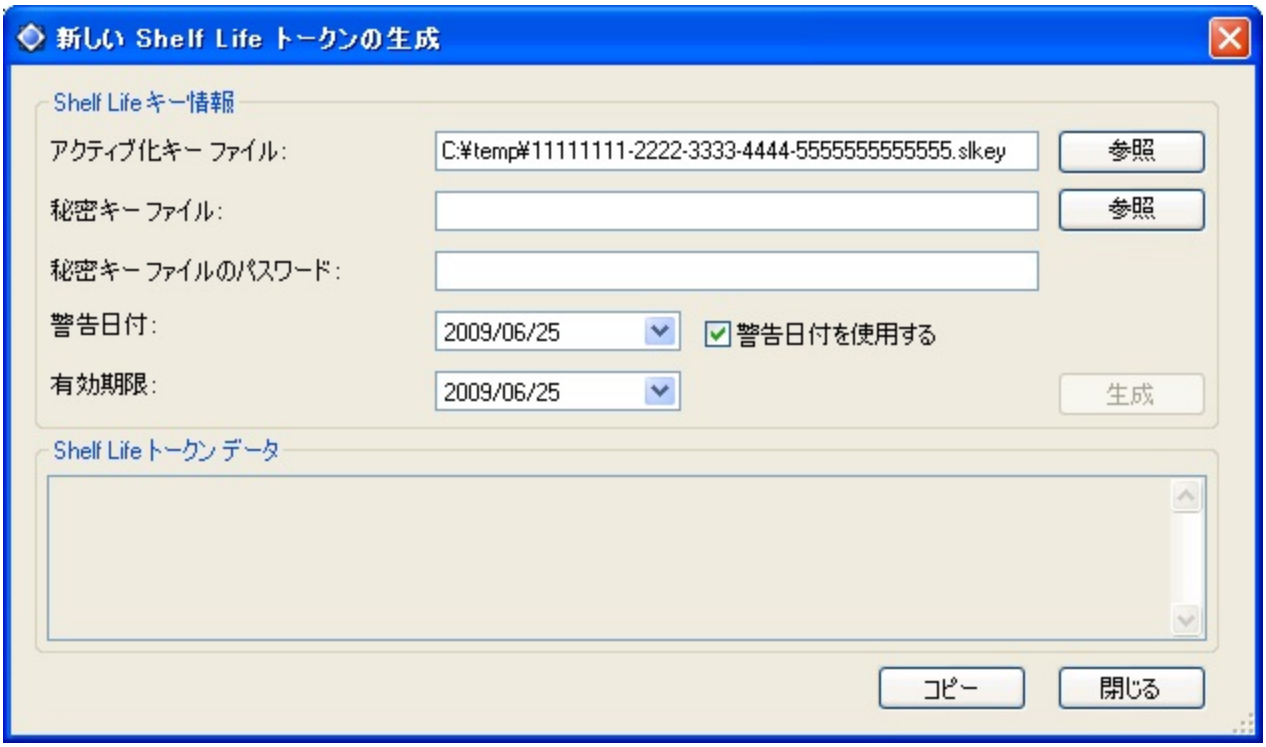
メモ:この機能を使用するには、Shelf Life Activation Key(SLAK)を別途購入する必要があります。

2.5.10.1.4 新しい Shelf Life トークンの生成

Shelf Life トークン ソースを持つ Shelf Life を使用する場合、Dotfuscator では新しい Shelf Life トークンを簡単に生成できます。Visual Basic の場合はメニュー バーの[**表示**|**Dotfuscator**|**新しい Shelf Life トークンの生成**]をクリックし、スタンドアロン Dotfuscator の場合は[**ツール**|**Shelf Life トークンの生成**]をクリックするだけです。

表示されるダイアログ ボックスで、適切な Shelf Life Activation Key ファイルと任意の PKCS #12 Private Key ファイルを検索、選択することによって、Shelf Life トークンの追加の検証を提供することができます。秘密キー ファイルを使用する場合は、[**秘密キー ファイルのパスワード**]フィールドに**パスワード**を入力してください。[**有効期限**]を設定し、任意で[**警告日付**]を設定します。[**警告日付**]フィールドの隣には[**警告日付を使用する**]チェック ボックスがあります。インストルメンテーション中に InsertShelfLifeAttribute を使って警告日付の動作を有効にしなかった場合、あるいは、インストルメンテーション中に有効にしたけれど、それを無効にする更新した Shelf Life トークンを提供したい場合は、このチェック ボックスをオフにしてください。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



Shelf Life キー情報が正しいと判断されると、[**生成**]ボタンがアクティブになり、クリックできるようになります。このボタンをクリックすると、Shelf Life がインストルメントされたアプリケーションが InsertShelfLifeAttribute の ShelfLifeTokenSource プロパティを介して使用できる、新しい Shelf Life トークンが生成されます。[**コピー**]ボタンをクリックすると、Shelf Life トークン データをクリップボードにコピーできます。

2.5.10.2 改ざん通知

アプリケーションが改ざんされているかどうかを検出し、改ざんされている場合には任意で Runtime Intelligence Service のサブスクリプション顧客のために、Runtime Intelligence Service へメッセージを送信するようにアプリケーションをインストルメントすることができます。

適切に属性が設定された .NET アプリケーションで Dotfuscator を実行すれば、Dotfuscator は改ざん通知属性を処理し、適宜にアプリケーションをインストルメントします。結果として生じる出力アプリケーションは、いつでも改ざん通知を Runtime Intelligence データ サービスへ送信できる状態になっています。このレベルにおける改ざん通知とアプリケーション分析の唯一の違いは、属性にあります。

改ざん通知メッセージの種類

改ざん通知は 1 つのメッセージの種類を定義します。

- 改ざん検出

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Runtime Intelligence Service は**改ざん検出**メッセージを定義します。アプリケーションは、Dotfuscator から実行された以降に自身が変更されていると判断したとき、このメッセージを送信します。アプリケーションにこれらのメッセージを送信させるには、以下のことを実行する必要があります。

- Setup や Teardown などの Runtime Intelligence 属性をアプリケーションに設定します。
- アプリケーションの整合性チェックを実行したいメソッドに InsertTamperCheck 属性を追加します。
- [**改ざんメッセージの送信**]オプションをオンにした Dotfuscator を介してアプリケーションを実行します。「アプリケーション分析を備えた Dotfuscator の構成および実行」を参照してください。

Runtime Intelligence データを含むメッセージの内容を見るには、「Runtime Intelligence 対応アプリケーションの例」を参照してください。

2.5.10.2.1 改ざんレポート

改ざんを検出するには、アプリケーション内で必ず実行される 1 つ以上のメソッドに **InsertTamperCheck** 属性を設定する必要があります。Dotfuscator は処理中に **InsertTamperCheck** 属性を見つけると、実行時にアセンブリ レベルの整合性チェックを実行するためのコードを追加します。整合性チェックに失敗した場合は、改ざん検出メッセージを Runtime Intelligence Service へ送信できます。また、アプリケーションを終了するコードや、それ以外のあらゆるアプリケーション定義のコードを呼び出すことができます(「改ざん操作」を参照してください)。**InsertTamperCheck** 属性は実行時には必要ありません。そのため、Dotfuscator はこの属性を出力アプリケーションから取り除きます。

アプリケーションに **InsertTamperCheck** 属性をいくつ含めてもかまいません。アプリケーションが改ざんされた場合には、1 つのアプリケーション セッションから複数の改ざん検出メッセージが同じグループ ID で送信されます。

> ⚠ この属性を、Setup 属性が含まれているメソッドに置いてはいけません。この属性を含んでいるメソッドは、Setup 属性を含んでいるメソッドの後に実行する必要があります。

InsertTamperCheck 属性
`[PreEmptive.Attributes.InsertTamperCheck()]` `public void DoStuff() { ...}`

2.5.10.2.2 改ざん操作

InsertTamperCheck 属性を使用すると、改ざん通知をマネージ サービスへ送信することに加え、改ざんの検証中に実行されるコードを差し込むことができます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

アプリケーション通知

これを実現するために、**InsertTamperCheck** 属性で **ApplicationNotificationSink** を指定できます。Dotfuscator はこの ApplicationNotificationSink を利用して、改ざんチェックの結果をアプリケーションへ伝えるコードを生成します。**ApplicationNotificationSink** には、書き込み可能なブール値で評価されるプロパティまたはフィールド、あるいは **void(bool)** 署名付きのメソッドまたはデリゲートを指定できます。改ざんチェック後、アプリケーションが改ざんされていれば、生成されたコードはブール値を true に設定し、そうでなければ false に設定します。アプリケーションは改ざん通知に応えて、どのように対応してもかまいません。

InsertTamperCheck 属性には、ApplicationNotificationSink を指定するための 3 つのプロパティが定義されています。

- **ApplicationNotificationSinkElement.**
- **ApplicationNotificationSinkOwner.**
- **ApplicationNotificationSinkName**

これらのプロパティについては、「カスタム属性リファレンス」の InsertTamperCheckAttribute セクションで詳しく説明します。

アプリケーション通知シンクの設定は省略可能です。省略すると、改ざんチェックが実行されてもアプリケーションに通知されません。

ApplicationNotificationSinkElement を "**DefaultAction**" に設定すると、Dotfuscator は改ざんが検出されたときにアプリケーションを終了するコードを差し込みます。

InsertTamperCheck 属性の **ApplicationNotificationSink** を、属性付きメソッドと同じクラス内に定義されているインスタンス フィールドとする場合の使用例を示します。

InsertTamperCheck 属性の ApplicationNotificationSink

```
class TamperSample {
   bool instanceTamperFlag;

   [PreEmptive.Attributes.InsertTamperCheck(
      ApplicationNotificationSinkElement = SinkElements.Field,
      ApplicationNotificationSinkName = "instanceTamperFlag"
   )]
   private void Verify() {
     // Dotfuscator はここに、改ざん検出および通知のコードを追加します
   }
   private CheckTamperState() {
      if ( instanceTamperFlag ) {
         // アプリケーションは改ざんされている
      }
      else {
         // アプリケーションは改ざんされていない
      }
   }
}
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.10.2.3 改ざんのシミュレート

Dotfuscator には、アセンブリを「改ざんする」ための単純なコマンド ライン ユーティリティが付属しています。これは **TamperTester.exe** という名前で Dotfuscator 自体と同じフォルダーにインストールされます。

TamperTester.exe
使用方法：`tampertester <file_name> [出力先フォルダー]`

このユーティリティから Dotfuscator アセンブリを実行することによって、改ざんの通知メッセージが期待どおりに生成され送信されるかどうかをテストできます。また、改ざん検出に応えて実行するよう記述したあらゆるアプリケーション コードをテストできます。

2.5.10.3 例外の追跡

このセクションでは、例外の追跡を使用する場合の開発プロセスについて説明します。また、例外の追跡に関連する Dotfuscator のユーザー インターフェイスおよび構成オプションについて説明します。

2.5.10.3.1 例外レポート

メソッド内で発生した例外を検出するには、そのメソッドに **`ExceptionTrack`** 属性を配置します。アセンブリ内のどこかで発生した例外を検出するには、そのアセンブリに **`ExceptionTrack`** 属性を配置します。

Dotfuscator は処理中に **`ExceptionTrack`** 属性を見つけると、設定されている種類の例外を検出するためのコードを追加します。

- **Caught**：キャッチされた例外の追跡が設定されている場合、Dotfuscator は 'catch' ブロックに入った直後の例外を追跡するコードを差し込みます。
- **Thrown**：スローされた例外の追跡が設定されている場合、Dotfuscator は 'throw' ステートメントによってスローされる直前の例外を追跡するコードを差し込みます。
- **Unhandled（既定）**：ハンドルされない例外の追跡が設定されている場合、Dotfuscator はメソッド レベルまたはアセンブリ レベルのいずれかで例外を追跡するコードを差し込みます。メソッド レベルでは、try/catch ブロックでメソッドをラップし、例外を再度スローします。アセンブリ レベルでは、UnhandledException イベント ハンドラーを現在の AppDomain（.NET Framework アプリケーションの場合）または現在の Application（Silverlight アプリケーションの場合）に登録します。

例外が検出されたら、**`ExceptionTrack`** 属性の **`SendReport`** プロパティを true（既定）に設定することにより、構成されている Runtime Intelligence エンドポイントにその例外を報告することができます。

また、Dotfuscator は、例外を引き起こす操作の説明や連絡先などの情報をユーザーから取得するように構成することができます。開発者はこれらの情報を利用して、追加情報を要求したり、修正されている問題の通知を提供することができます。この情報は、例外レポート メッセージに添付されます。この種のユーザー指定情報を取得するには、**`ReportInfoSource`** を指定します。**`ReportInfoSource`** の構成の詳細については、「ユーザー指定の例外レポート情報の収集」を参照してください。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

例外レポートの送信では、**OptInSource** が設定されている場合はユーザーの opt-in 設定を使用します。

例外レポート メッセージを送信することについて、ユーザーから明示的な同意を得るよう、Dotfuscator を構成することができます。この場合、ユーザーの明示的な同意は、構成されている Runtime Intelligence の opt-in 設定をオーバーライドします。例外レポート メッセージを送信するための明示的な同意を得るには、**ReportInfoSource** を指定します。**ReportInfoSource** の構成の詳細については、「ユーザー指定の例外レポート情報の収集」を参照してください。

レポート情報ソースの設定は省略可能です。省略すると、ユーザー指定の情報は収集されず、例外レポート メッセージの送信は Runtime Intelligence の opt-in 設定によって制御されます。

2.5.10.3.2 例外操作

例外レポート メッセージをマネージ サービスへ送信するほかに、**ExceptionTrack** 属性を使用すると、例外の検出時にユーザーが指定したメソッドへのコールバックを実行するコードを差し込むことができます。

例外通知

これを実現するには、**ExceptionTrack** 属性を使用して **ExceptionNotificationSink** を指定します。Dotfuscator はこれを使用して、検出された Exception オブジェクトをアプリケーションに返すコードを生成します。**ExceptionNotificationSink** には、System.Exception 型の書き込み可能なフィールドまたは設定可能なプロパティ、あるいは **void(System.Exception)** 署名付きのメソッドまたはデリゲートを指定できます。例外検出後、生成されたコードは、フィールド値またはプロパティ値をキャプチャされた Exception オブジェクトに設定します。メソッドまたはデリゲート シンクを使用している場合は、生成コードは、キャプチャされた Exception オブジェクトをシンクの唯一のパラメーターとして指定し、このシンクを呼び出します。アプリケーションは検出された例外に応えて、どのように対応してもかまいません。**ExceptionTrack** 属性には、**ExceptionNotificationSink** を指定するための 3 つのプロパティが定義されています。

- **ExceptionNotificationSinkElement**
- **ExceptionNotificationSinkOwner**
- **ExceptionNotificationSinkName**

これらのプロパティについては、「カスタム属性リファレンス」の ExceptionTrack セクションで詳しく説明します。

例外通知シンクの設定は省略可能です。省略すると、設定した種類の例外が検出されても、カスタム動作は発生しません。

ExceptionTrack属性の **ExceptionNotificationSink** を、属性付きメソッドと同じクラスに定義されているメソッドとする場合の使用例を示します。

ExceptionTrack の ExceptionNotificationSink をメソッドとする使用法

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
[ExceptionTrack(
   ExceptionNotificationSinkElement = SinkElements.Method,
   ExceptionNotificationSinkName = "Response"
)]
private void Foo() {
  ...
}

// 検出された例外に対応します
public void Response(System.Exception e) {
  ...
}
```

2.5.10.3.3 ユーザー指定の例外レポート情報の収集

例外の追跡を使用している場合は、明示的なユーザー同意のうえ、ユーザーからコメントおよび連絡先情報を収集できるよう、Dotfuscator を構成することができます。これらはキーと値のペアの形式で提供され、実行時の、例外レポート メッセージの構築中に読み取られます。

このユーザー指定のレポート情報を提供するには、送信したいメッセージに対応する属性に **ReportInfoSource** を指定します。

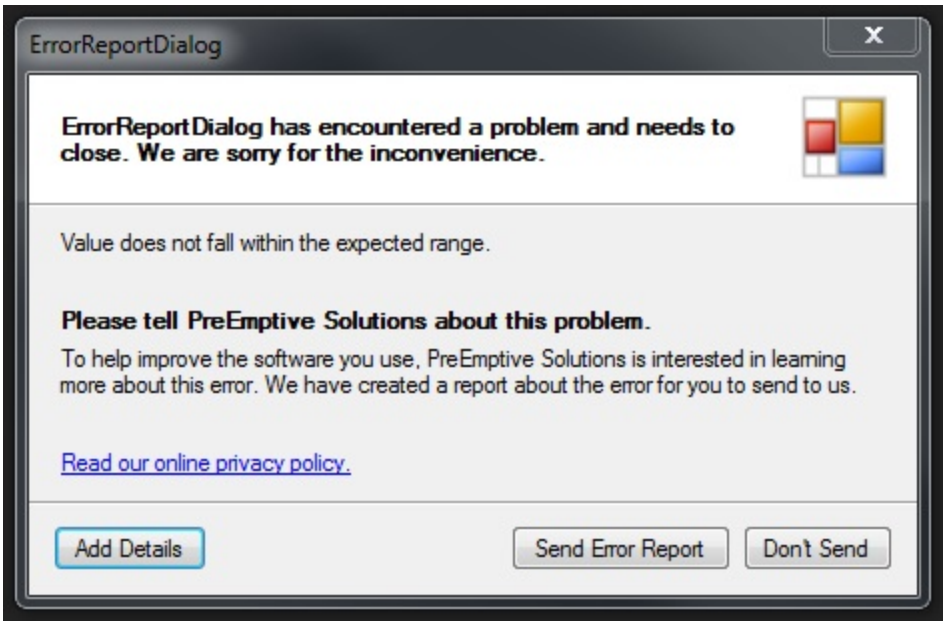
Dotfuscator は **ReportInfoSource** を利用して、実行時にキーと値のペアを収集するためのコードを生成します。**ReportInfoSource** は、IDictionary または **IDictionary<string,string>** で評価されるプロパティ、メソッド、フィールド、あるいはメソッド引数（メソッド レベルの例外追跡を使用している場合）です。開発者の責任において、例外が検出された時点で、必ず正しい値が **ReportInfoSource** で入手できるようにしてください。

組み込みの例外レポート ダイアログを ReportInfoSource として使用する

Dotfuscator では、あらかじめ作成された例外レポート ダイアログを導入することにより、ほとんどのシナリオの構成が容易に行え、一貫性のあるユーザー操作が例外レポートに提供されるようにすることができます。組み込みダイアログを使用するには、アセンブリは、.NET Framework バージョン 1.1 以降または Silverlight バージョン 2 以降を対象としている必要があります。**ReportInfoSource** として組み込みダイアログを使用するよう Dotfuscator に指示するには、**ReportInfoSourceElement** 値を "DefaultAction" に設定します。

.NET Framework で組み込みダイアログを使用する場合、ダイアログは Windows フォーム API を使用して構築、表示されます。これは、コンソールやサービス アプリケーションに予期せぬ結果をもたらす可能性があります。このような場合は、カスタムの **ReportInfoSource** を使用するのが望ましいかもしれません。アセンブリがまだ適切な Windows フォーム ライブラリを参照していない場合は、参照が追加されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



カスタム ReportInfoSource の使用

ExceptionTrackAttribute には、**`ReportInfoSource`** を指定するための 3 つのプロパティが定義されています。

- **ReportInfoSourceElement**。**`ReportInfoSourceElement`** には、**`SourceElements`** 列挙（フィールド、プロパティ、メソッド、または、メソッド レベルの例外追跡を使用している場合はメソッド引数）に定義されているいずれの値も指定できます。**`ReportInfoSourceElement`** がメソッド引数である場合には、属性がアタッチされるメソッドのメソッド パラメーターに対応している必要があります。
- **ReportInfoSourceOwner**。**`ReportInfoSourceElement`** がフィールド、メソッドまたはプロパティである場合には、**`ReportInfoSourceOwner`** はそのフィールド、メソッドまたはプロパティを定義しているクラスを指し示す必要があります。
- **ReportInfoSourceName**。**`ReportInfoSourceName`** には、実行時にユーザー指定のレポート情報を格納する **`IDictionary`** 型または **`IDictionary<string,string>`** 型のフィールド、メソッド、プロパティ、またはメソッド引数の名前を設定する必要があります。

これらのプロパティについては、「カスタム属性リファレンス」の ExceptionTrack セクションで詳しく説明します。

`ReportInfoSource` で提供される辞書には、次に示す 3 つのキーと値のペアを含めることができます。

- **consent**。これは、ユーザーが現在の例外レポート メッセージの送信に対し、明示的に opt-in または opt-out を選択しているかどうかを示すブール値の文字列表記です。この同意は、グローバルな Runtime Intelligence の opt-in 設定から独立しており、この設定をオーバーライドします。
- **comment**。これは、ユーザーから任意で提供されるカスタム コメントです。これを利用して、何を行っているときに例外が発生したのかなど、フィードバックを送信するようユーザーに要求することができます。
- **contact**。これは、ユーザーから任意で提供される連絡先です。内容は構造化されないため、アドレスや電話番号、あるいはソーシャル ネットワーキング Web サイトのユーザー名などを含めることができます。組み込みダイアログは、ユーザーがこれをアドレスとして提供することを要請します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

上で説明した以外のキーと値のペアはすべて無視されます。

ユーザー指定のレポート情報を収集することは任意です。受け取った辞書がヌルであるか、辞書に consent キーが入っていない、または、consent キーの値がヌルであるか値がブール値に解析されない場合は、グローバルな Runtime Intelligence の opt-in 設定が採用されます。comment キーまたは contact キーを省略した場合、この情報は結果として生じる Runtime Intelligence メッセージに含まれません。グローバルな Runtime Intelligence opt-in 設定の詳細については、「エントリ ポイント属性」を参照してください。

ExceptionTrack 属性の ReportInfoSource を、"**GetDictionary**" というメソッドとして定義する場合の使用例を示します。

ExceptionTrack 属性の ReportInfoSource の使用法

```
[ExceptionTrack(
   ReportInfoSourceElement = SourceElements.Method,
   ReportInfoSourceName = "GetDictionary"
)]
private void Foo() {
  ...
}

// 辞書を作成して、ユーザー指定のレポート情報を入力する
public IDictionary<string, string> GetDictionary() {
  Dictionary<string, string> dict = new Dictionary<string, string>();
  dict.Add("consent", "true");
  dict.Add("comment", "Foo() メソッドが例外をスローしました。");
  dict.Add("contact", "foo@bar.com");
  return dict;
}
```

2.5.10.4 アプリケーション分析（Runtime Intelligence）

Runtime Intelligence メッセージの種類

Runtime Intelligence にはメッセージの種類がいくつか定義されています。

- アプリケーションおよびセッションの開始
- アプリケーションおよびセッションの終了
- 機能
- パフォーマンス プローブ
- システム プロファイル
- 改ざん検出
- 例外検出

アプリケーションおよびセッションの**開始**および**停止**メッセージ（**アプリケーションのライフサイクル メッセージ**）は、**アプリケーション**が実行を開始したときとシャットダウンしたときに送信されるものです。これらのメッセージに含まれる情報を使用して、アプリ

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ケーションの動作および基本的な使用パターンを追跡します。**機能**、**パフォーマンス プローブ**、または**システム プロファイル**のメッセージを使用して派生的な使用法や環境情報を取得します。

メッセージからのデータが Runtime Intelligence ポータルのダッシュボードを動かします。アプリケーションにこれらのメッセージを送信させるには、以下のことを実行する必要があります。

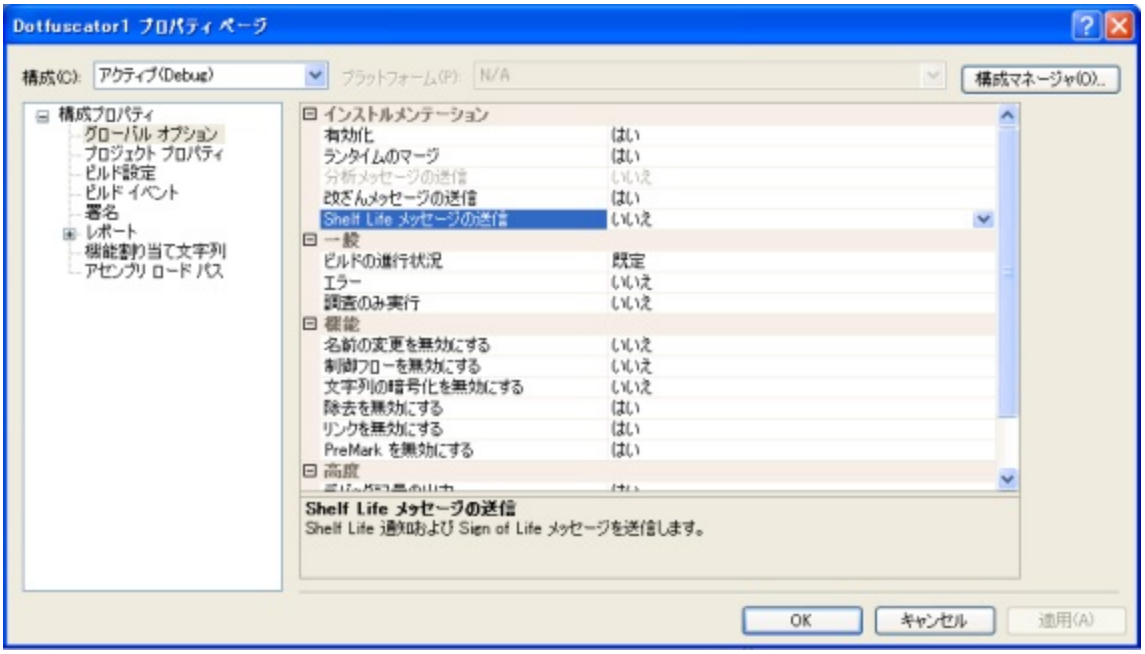
- Runtime Intelligence Service のサブスクライバーになります（これにより、ポータルのダッシュボードおよびデータにアクセスできるようになります）。
- Setup や Teardown などの Runtime Intelligence 属性をアプリケーションにインストルメントします。
- ［**分析メッセージの送信**］オプションをオンにした Dotfuscator を介してアプリケーションを実行します。「アプリケーション分析を備えた Dotfuscator の構成および実行」を参照してください。

;Runtime Intelligence データを含むメッセージの内容を見るには、「Runtime Intelligence 対応アプリケーションの例」を参照してください。

2.5.10.4.1 アプリケーション分析を備えた Dotfuscator の構成および実行

Runtime Intelligence のオプション

Runtime Intelligence の構成オプションは Dotfuscator の［**設定**］タブの［**グローバル オプション**］プロパティ ページで利用できます。Visual Studio の中では、［**グローバル オプション**］プロパティ ページは Dotfuscator プロジェクトのプロパティ ダイアログから利用できます。



スタンドアロン ユーザー インターフェイスでは、［**オプション**］プロパティ ページは［**設定**］タブで利用できます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

インストルメンテーション（改ざん検出、Shelf Life（使用期限）、例外の報告、および Runtime Intelligence）には、[**有効化**]、[**ランタイムのマージ**]、[**分析メッセージの送信**]、[**Shelf Life メッセージの送信**]および[**改ざんメッセージの送信**]という 5 つのオプションがあります。

インストルメンテーション（改ざん検出、Shelf Life（使用期限）および Runtime Intelligence）を有効にする

[**有効化**]オプションは、改ざん検出、Shelf Life（使用期限）および Runtime Intelligence のオン（既定値）/オフの切り替えを行います。これがオフになっている場合、Dotfuscator は入力アセンブリに含まれているすべての改ざん検出、Shelf Life（使用期限）およびアプリケーション分析の属性を無視します。

ランタイムをマージまたは参照する

"**はい**"（既定値）に設定されている場合、[**ランタイムのマージ**]オプションは Dotfuscator に対し、改ざん検出、Shelf Life（使用期限）および Runtime Intelligence に必要なランタイム コードを入力アセンブリの 1 つに追加するように指示します。Runtime Intelligence DLL の配布は必要ありません。

"**いいえ**" に設定されている場合は、Dotfuscator は別のアセンブリにランタイム コードを出力し、入力アセンブリに適切なアセンブリ参照を追加します。新しい DLL をアプリケーションと一緒に配布する必要があります。Dotfuscator は今後も、改ざん検出、Shelf Life（使用期限）、または Runtime Intelligence 用に構成されたアセンブリごとにインストルメンテーションのヘルパー メソッドを差し込む必要があります。

マージがどのように設定されていたとしても、Dotfuscator は依然として、Runtime Intelligence ライブラリを使用するためには、入力アセンブリの 1 つにコードを差し込む必要があります。Dotfuscator は最良の入力アセンブリを選択するために、入力アセンブリの依存関係分析を実行します。入力アセンブリ間の新しい依存関係を最小限にするように、このような方法で選択を行っていますが、新しい依存関係の追加が避けられない場合もあります。プロジェクト プロパティで、`accesspoint` という名前およびコードが挿入されるアセンブリの名前を示す値を指定することによって、Runtime Intelligence ランタイム コードが差し込まれるアセンブリをオーバーライドすることができます。

厳密な名前

入力アセンブリの一部が厳密名を持っている場合は、Dotfuscator はランタイム DLL に厳密な名前を付け、それを透過的に署名します。入力アセンブリのうちどれも厳密名を持っていなければ、Dotfuscator はランタイム DLL に厳密な名前を付けません。

改ざんメッセージまたはアプリケーション分析メッセージを送信する

既定では、**`InsertTamperCheck`** 属性で修飾されたメソッドを含んでいるアセンブリは、改ざんが検出されたときに改ざん通知メッセージを送信するコードを使ってインストルメント化されます。この機能は構成可能で、[**改ざんメッセージの送信**]オプションによって制御されます。

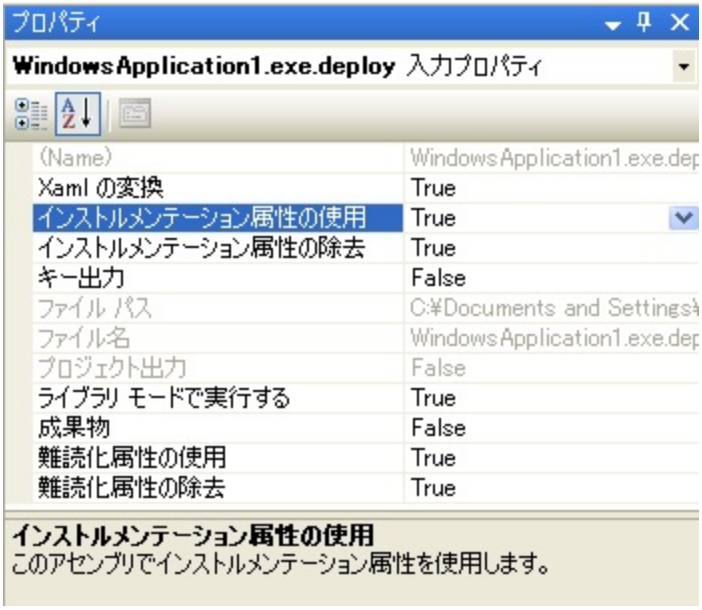
[**分析メッセージの送信**]オプションが "はい" に設定されている場合は、アプリケーションの開始および停止メッセージ、セッションの開始および停止メッセージ、機能の利用状況メッセージを送信するための（指定した属性に基づく）コードが Dotfuscator によって追加されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

アセンブリ レベルの構成オプション

Dotfuscator では、インストルメンテーション属性の処理（改ざん検出、Shelf Life（使用期限）およびアプリケーション分析など）をさらに細かく制御できます。開発者は、これらの属性を引き受けるのか無視するのか、また保持するのか削除するのかを Dotfuscator に指示することができます。これらの設定はアセンブリ レベルで適用することができます。

Visual Studio の中では、アセンブリ レベルのオプションは、入力アセンブリにフォーカスがあるときのプロパティ ツール ウィンドウで利用できます。



スタンドアロン ユーザー インターフェイスでは、アセンブリ レベルのオプションは［**入力**］タブで利用できます。

インストルメンテーション属性の使用

［**インストルメンテーション属性の使用**］を "**`True`**"（既定値）に設定すると、これらの属性を処理し、示されているインストルメンテーションをターゲット アセンブリで実行するよう Dotfuscator に指示できます。インストルメンテーション属性をすべて無視するよう Dotfuscator に指示するには、このプロパティを "**`False`**" に設定します。

"**`False`**" 設定は、テスト目的の場合や、アセンブリのセットを Dotfuscator で複数回実行しなければならないような高度なシナリオで役に立ちます。

インストルメンテーション属性の除去

［**インストルメンテーション属性の除去**］を "True"（既定値）に設定すると、ターゲットの出力アセンブリからこれらの属性を削除するよう Dotfuscator に指示できます。出力アセンブリにこれらの属性を残すよう Dotfuscator に指示するには、このプロパティを "**`False`**" に設定します。

"**`False`**" 設定は［**インストルメンテーション属性の使用**］オプションと同様、テスト目的の場合や、アセンブリのセットを Dotfuscator で複数回実行しなければならないような高度なシナリオで役に立ちます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

次の表は、これら 2 つのオプションをさまざまに組み合わせた場合の結果を示しています。

インストルメンテーション属性の使用	インストルメンテーション属性の除去	メモ
True	**True**	既定の設定。アセンブリはインストルメント化され、属性は削除されます。
False	**True**	アセンブリはインストルメント化されず、属性は取り除かれます。これは、改ざん検出またはアプリケーション分析機能を含んでいないテスト ビルドを作成する場合に有用です。
True	**False**	アセンブリはインストルメント化され、属性はそのまま残されます。現在のところ、この組み合わせはお勧めできません。
False	**False**	アセンブリはインストルメント化されず、属性はそのまま残されます。これは、難読化する必要があるが、後続の手順で改ざん検出またはアプリケーション分析の追加が必要になるアセンブリに対して有用です。

Runtime Intelligence とその他の Dotfuscator 機能

アプリケーション分析を使用するとき、その他の Dotfuscator 機能は使用可能です。挿入されたランタイム コードには難読化属性で注釈が付けられているため、アプリケーション分析を使用するアプリケーション上で名前の変更、除去、あるいはそれ以外の変換を実行されても、ユーザーによる構成は（入力アセンブリに対して必要となる設定以外には）何も要求されません。

2.5.10.4.2 Runtime Intelligence のカスタム属性

Runtime Intelligence のすべてのカスタム属性は **`PreEmptive.Attributes.dll`** 内に定義されています。このファイルは既定値では、*Dotfuscator 4* のインストール フォルダーにあります。Runtime Intelligence のカスタム属性をアプリケーションに追加するには、開発者はこの **`DLL`** への参照を追加して、コンパイル時に **`DLL`** を使用できるようにしておく必要があります。Runtime Intelligence コードを差し込むときに、Dotfuscator はこの **`DLL`** への参照を解除します。そのため、アプリケーションの実行時に DLL は必要なく、**`DLL`** をアプリケーションと共に配布する必要はありません。

カスタム属性 **`DLL`** を使用するほかに、Dotfuscator ユーザー インターフェイスの［インストルメンテーション］エディターを使用して、すべての Runtime Intelligence 属性を拡張属性として指定できます。これは、カスタム属性を追加するためにアプリケーション コードを変更したくない場合に有用です。

このセクションでは、カスタム属性について、その属性がどのようなもので、いつどこで使用するのかといった高レベルな説明を行います。プログラマー向けのリファレンスは、「カスタム属性リファレンス」を参照してください。

2.5.10.4.2.1 アセンブリ レベルの属性

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

すべての Runtime Intelligence メッセージには、メッセージを送信するエンティティおよびアプリケーションに関する共通情報が含まれている必要があります。開発者はこの情報を、アプリケーションを形成する 1 つまたは複数のアセンブリに格納されているカスタム属性のセットで提供します。

ほとんどの他の属性とは異なり、Dotfuscator はアセンブリ レベルの属性を処理後にアセンブリから削除しません。Dotfuscator はこれらの属性を削除するのではなく、アプリケーションの読み込み時に Runtime Intelligence のランタイムが読み取れる形式に変換します。

詳細については、「カスタム属性リファレンス」を参照してください。

一意識別子

アセンブリ レベルの属性のほとんどは、一意識別子を必要とします。記述する場所以外、識別子の生成および保持は開発者が行わなければなりません。生成される識別子文字列には標準 GUID 形式を使用することをお勧めします。たとえば、次のようになります。

一意識別子の例
123D35B3-BFDD-4797-9E9D-A39A57C1FD7B B85A9684-1964-489e-867B-D81E89DB7CCB

Application 属性

エントリ ポイント メソッドを含んでいるアセンブリはすべて、Application 属性も持っている必要があります。Application 属性は、Runtime Intelligence データを送信するアプリケーションついての情報を提供します。最も重要な情報はアプリケーション ID で、これは、アプリケーションの全有効期間にわたって変更してはいけない一意な識別子です。アプリケーションの名前やバージョン番号が変わっても、Runtime Intelligence Service がそのアプリケーションに関するデータを集めることができるようにするにはアプリケーション ID が必要です。

Application 属性には、ほかにも名前やバージョンなどのプロパティがあります。これらのプロパティは必須ではありません。名前およびバージョン番号がアプリケーション属性に指定されていない場合は、Runtime Intelligence コードはアセンブリを検討することによって、これらの取得を試みます。一方、名前およびバージョンがアプリケーション属性に指定されている場合には、Runtime Intelligence コードはアセンブリを検討する代わりに、これらの値を使用します。

Binary 属性

改ざんの確認メソッドを含んでいるアセンブリはすべて、Binary 属性も持っている必要があります。Binary 属性は、アプリケーション内の特定のコンポーネント(アセンブリ)に関する情報を提供します。各アセンブリは対応する ID と共に、自身に特有の Binary 属性を持っている必要があります。この属性に指定される ID が改ざん通知メッセージで送信されることにより、アプリケーションを形成する特定のバイナリ(アセンブリ)を識別できます。この属性を提示する必要はありません。

> **メモ**:ここで留意する点は、Binary 情報はアプリケーションの開始および停止メッセージの一部としては送信されないということです。この情報は改ざん検出メッセージの一部として送信されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Business 属性

エントリ ポイント メソッドを含んでいるアセンブリはすべて、Business 属性も持っている必要があります。Business 属性は、アプリケーションを構築した組織に関する情報を提供します。最も重要なプロパティは CompanyKey です。CompanyKey は、Runtime Intelligence Service の登録処理の一部として提供される一意な識別子です。

Runtime Intelligence が正しく動作するためには、Business 属性および CompanyKey が必要です。

2.5.10.4.2.2 エントリ ポイント属性

あらゆる種類の Runtime Intelligence データを送信するには、処理するアプリケーション内で適切なエントリ ポイント メソッドを識別できなければなりません。そのため、そのエントリ ポイント メソッドに Setup 属性を追加します。Dotfuscator は処理中に Setup 属性を見つけると、エントリ ポイント メソッドの最初の部分に Runtime Intelligence の初期化コードを追加します。また、**分析メッセージを送信**するように構成されている場合には、Dotfuscator は開始メッセージを Runtime Intelligence Service へ送信するコードを追加します。Setup 属性は実行時には必要ありません。そのため、Dotfuscator はこの属性を出力アプリケーションから取り除きます。

エントリ ポイント メソッドはアプリケーション内の 1 メソッドで、アプリケーションのスタートアップ シーケンスの一部として実行されるものです。これは、アプリケーションが開始するときにランタイムまたはフレームワークによって最初に呼び出されるメソッドである必要はありませんが、そのアプリケーションを起動するたびに必ず実行されるプロパティを持っている必要があります。

アプリケーションの種類別にエントリ ポイント メソッドの例を挙げます。

アプリケーションの種類	考えられるエントリ ポイント メソッド
コンソール アプリケーション	Main、または常に Main から呼び出されるメソッド
Windows フォーム アプリケーション	Main フォームのコンストラクター

Setup 属性にはいくつかの引数やプロパティがありますが、これらはすべて、プライバシーおよびデータ セキュリティのシナリオをサポートするためにあります。

プライバシーおよびセキュリティ設定

Opt-in

Runtime Intelligence データ収集の opt-in または opt-out の選択をユーザーに許可しているアプリケーションは、アプリケーションの開始時に、ユーザーの選択を Runtime Intelligence コードに伝える必要があります。

これを実現するために、Setup 属性で **`OptInSource`** を指定します。Dotfuscator は OptInSource を利用して、ユーザーの opt-in 選択を実行時に取得するコードを生成します。**`OptInSource`** は、ブール値で評価されるプロパティ、フィールド、あるいはメソッド引数です。開発者の責任において、エントリ ポイント メソッドが実行された時点で、必ず正しい値が **`OptInSource`** に格納されるようにしてください。

Setup 属性には、**`OptInSource`** を指定するための 3 つのプロパティが定義されています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- **OptInSourceElement**。`OptInSourceElement` には、`SourceElements` の列挙(フィールド、プロパティ、メソッド引数)に定義されているいずれの値も指定できます。`OptInSourceElement` がメソッド引数である場合には、`Setup` 属性がアタッチされるメソッドのメソッド パラメーターに対応している必要があります。
- **OptInSourceOwner**。`OptInSourceElement` がフィールドまたはプロパティである場合には、`OptInSourceOwner` はそのフィールドまたはプロパティを定義しているクラスを指し示す必要があります。
- **OptInSourceName**。`OptInSourceName` には、実行時にユーザーの opt-in 選択を格納するブール値のフィールド、プロパティ、またはメソッド引数の名前を設定する必要があります。

opt-in 設定は省略可能です。省略すると、Runtime Intelligence コードはすべてのメッセージを送信します(つまり、opt-in と見なします)。

Setup 属性の `OptInSource` をメソッド引数とする場合の使用例を示します。

Setup 属性の OptInSource の例

```
[Setup(
   OptInSourceElement = SourceElements.MethodArgument,
   OptInSourceName = "optIn",
   UseSSL = false
)]
private void MyEntryMethod(bool optIn) { ...}
```

アプリケーション インスタンス ID

アプリケーション インスタンス ID (例 : シリアル番号) を Runtime Intelligence データの一部として提供したいアプリケーションは、アプリケーションの開始時に、その ID を Runtime Intelligence コードに伝える必要があります。

これを実現するために、Setup 属性で `InstanceIdSource` を指定することができます。Dotfuscator は InstanceIdSource を利用して、アプリケーションのインスタンス ID を実行時に取得するコードを生成します。`InstanceIdSource` は、文字列値で評価されるプロパティ、フィールド、あるいはメソッド引数です。開発者の責任において、エントリ ポイント メソッドが実行された時点で、必ず正しい値が `InstanceIdSource` に格納されるようにしてください。

`OptInSource` と同様に、`Setup` 属性には `InstanceIdSource` を指定するための 3 つのプロパティが定義されています。

- **InstanceIdSourceElement**。`InstanceIdSourceElement` には、`SourceElements` の列挙(フィールド、プロパティ、メソッド引数)に定義されているいずれの値も指定できます。`InstanceIdSourceElement` がメソッド引数である場合には、`Setup` 属性がアタッチされるメソッドのメソッド パラメーターに対応している必要があります。
- **InstanceIdSourceOwner**。`InstanceIdSourceElement` がフィールドまたはプロパティである場合には、`InstanceIdSourceOwner` はそのフィールドまたはプロパティを定義しているクラスを指し示す必要があります。
- **InstanceIdSourceName**。`InstanceIdSourceName` には、実行時にアプリケーションのインスタンス ID を格納する文字列値のフィールド、プロパティ、またはメソッド引数の名前を設定する必要があります。

インスタンス ID 設定は省略可能です。省略すると、Runtime Intelligence データにインスタンス ID は含まれません。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Setup 属性の **InstanceIdSource** を、"**Program**" クラスの "**InstanceId**" という静的プロパティとして定義する場合の使用例を示します。

Setup 属性の InstanceIdSource の例

```
[Setup(
   InstanceIdSourceElement = SourceElements.Property,
   InstanceIdSourceName = "InstanceId",
   InstanceIdSourceOwner = typeof(Program),
   UseSSL = false
)]
private void MyEntryMethod() { ...}
```

SSL

UseSSL 引数は、メッセージを送信するために Runtime Intelligence コードがどの Web プロトコルを使用するかを制御します。UseSSL が true（既定値）に設定されている場合は、HTTPS プロトコルを使用してメッセージを送信します。UseSSL が false に設定されている場合は、HTTP プロトコルを使用してメッセージを送信します。

- プライバシーおよびセキュリティ設定プロパティについては、「カスタム属性リファレンス」の SetupAttribute セクションで詳しく説明します。

利用状況データのオフライン記憶域

Runtime Intelligence を備えたアプリケーションは、ネットワーク アクセスを利用できない状況にある場合、後で接続が復元されたときにデータを転送するよう、利用状況データを格納しておくことができます。利用状況データは分離ストレージに格納されます。この動作は既定値では有効になっており、既定の接続検出コードがインストルメント化されたアプリケーションに差し込まれます。開発者は、`OfflineStateSourceElement` を変更することによって、既定の動作をオーバーライドできます。`OfflineStateSourceElement` 値を **None** に変更すると、アプリケーションがネットワークに接続できない場合、利用状況データは格納されず、削除されるようになります。また開発者は、独自のネットワーク検出コードを書いて、アプリケーションのオフライン状態をインストルメント化されたメソッドのパラメーターのブール値、あるいはメソッドの戻り値として指定したり、フィールドまたはプロパティに指定することにより、ネットワーク接続の状態を Runtime Intelligence コードで利用できるようにすることもできます。これは、`OfflineStateSourceElement` プロパティを適切な値に設定し、`OfflineStateSourceName` および `OfflineStateSourceOwner` を設定することによって行います。

`OfflineStorageResultSinkElement` を介して、オフライン記憶域メカニズムへの利用状況データの格納が成功したか失敗したかを通知することもできます。値が **None** の場合、データ記憶域の成否に関する通知は行われません。値が **DefaultAction** の場合、記憶域メカニズムが利用状況データを格納できなければ、アプリケーションは直ちに終了します。`OfflineStorageResultSinkElement` を適切な値に設定し、`OfflineStorageResultSinkName` および `OfflineStorageResultSinkOwner` を設定することにより、開発者はオフライン記憶域の成否に対処するコードを書くことができます。選択したアプリケーション コードはパラメーター値で呼び出され、ブール型のプロパティまたはフィールドは、一番最近オフライン記憶域メカニズムへ利用状況データの保存を試みた結果によって設定されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

カスタム エンドポイント

SetupAttribute を追加するとき、メッセージの宛先のカスタム エンドポイントを設定できます。エンドポイント オプションには、無料の Runtime Intelligence Services ポータルのほかに、製品版の Runtime Intelligence Services ポータルや自由に選択できるカスタム エンドポイントが含まれています。

2.5.10.4.2.3 Exit Point 属性

分析用にアプリケーションを準備するには、開発者は適切な exit メソッドを識別できなければなりません。そのため、開発者はその exit メソッドに Teardown 属性を追加する必要があります。Dotfuscator は処理中に Teardown 属性を見つけると、exit メソッドの最後の部分に分析のクリーンアップ コードを追加します。Teardown 属性は実行時には必要ありません。そのため、Dotfuscator はクリーンアップ コードを追加した後、この属性を出力アプリケーションから取り除きます。

exit メソッドはアプリケーション内の 1 メソッドで、アプリケーションのシャットダウン シーケンスの一部として実行されるものです。これは、最後に呼び出されるメソッドである必要はありませんが、そのアプリケーションがシャットダウンするときに 1 回だけ実行されるプロパティを持っている必要があります。

同一のメソッドをエントリ ポイントと exit メソッドの両方にすることができます（たとえば、コンソール アプリケーションの Main メソッドがその例です）。

アプリケーションの種類別に exit メソッドの例を挙げます。

アプリケーションの種類	考えられる Exit メソッド
コンソール アプリケーション	Main、または常に Main から呼び出されるメソッド
Windows フォーム アプリケーション	Main フォームのコンストラクター
Windows フォーム アプリケーション	Main フォームの Dispose メソッド
Windows フォーム アプリケーション	Main フォームの OnClosed イベント ハンドラー

`Teardown` 属性には必要な引数もプロパティもないため、使い方が簡単です。

Teardown 属性

```
[Teardown()]
private void MyExitMethod() { ...}
```

2.5.10.4.2.4 改ざん通知属性

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

改ざん通知のカスタム属性は **PreEmptive.Attributes.dll** 内に定義されています。このファイルは既定値では、*Dotfuscator 4* のインストール フォルダーにあります。改ざん通知のカスタム属性をアプリケーションに追加するには、開発者はこの DLL への参照を追加して、コンパイル時に DLL を使用できるようにしておく必要があります。改ざん通知コードを差し込むときに、Dotfuscator はこの DLL への参照を解除します。そのため、アプリケーションの実行時に DLL は必要なく、DLL をアプリケーションと共に配布する必要はありません。

カスタム属性 DLL を使用するほかに、Dotfuscator ユーザー インターフェイスの[インストルメンテーション]エディターを使用して、すべての改ざん通知属性を拡張属性として指定できます。これは、カスタム属性を追加するためにアプリケーション コードを変更したくない場合に有用です。

このセクションでは、カスタム属性について、その属性がどのようなもので、いつどこで使用するのかといった高レベルな説明を行います。プログラマー向けのリファレンスは、「カスタム属性リファレンス」を参照してください。

2.5.10.4.2.5 Shelf Life および Sign of Life 属性

Shelf Life（使用期限）および Sign of Life のカスタム属性は **PreEmptive.Attributes.dll** に定義されており、このファイルは既定値では *Dotfuscator 4* インストール フォルダーにあります。アプリケーションに Shelf Life または Sign of Life のカスタム属性を追加するには、開発者はこの DLL への参照を追加し、コンパイル時にこの DLL を使用可能にする必要があります。Shelf Life コードを差し込むときに、Dotfuscator はこの DLL への参照を解除します。そのため、アプリケーションの実行時に DLL は必要なく、DLL をアプリケーションと共に配布する必要はありません。

カスタム属性 DLL を使用するほかに、Dotfuscator のユーザー インターフェイスの[インストルメンテーション]エディターを使用して Shelf Life または Sign of Life 属性を拡張属性として指定することもできます。これは、カスタム属性を追加するためにアプリケーション コードを変更したくない場合に有用です。

このセクションでは、カスタム属性について、その属性がどのようなもので、いつどこで使用するのかといった高レベルな説明を行います。プログラマー向けのリファレンスは、「カスタム属性リファレンス」を参照してください。

2.5.10.4.2.6 例外の追跡属性

例外の追跡のカスタム属性は **PreEmptive.Attributes.dll** 内に定義されています。このファイルは既定値では、*Dotfuscator 4* のインストール フォルダーにあります。例外の追跡のカスタム属性をアプリケーションに追加するには、開発者はこの DLL への参照を追加して、コンパイル時に DLL を使用できるようにしておく必要があります。例外の追跡コードを差し込むときに、Dotfuscator はこの DLL への参照を解除します。そのため、アプリケーションの実行時に DLL は必要なく、DLL をアプリケーションと共に配布する必要はありません。

カスタム属性 DLL を使用するほかに、Dotfuscator ユーザー インターフェイスの[インストルメンテーション]エディターを使用して、例外の追跡属性を拡張属性として指定することもできます。これは、カスタム属性を追加するためにアプリケーション コードを変更したくない場合に有用です。

このセクションでは、カスタム属性について、その属性がどのようなもので、いつどこで使用するのかといった高レベルな説明を行います。プログラマー向けのリファレンスは、「カスタム属性リファレンス」を参照してください。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.10.4.2.7 機能の利用状況属性

Dotfuscator では、Feature 属性によって機能の利用状況を追跡するサポートが提供されています。開発者は、機能の開始、停止、あるいは全体を割り当てるすべてのメソッドに Feature 属性を追加できます。Dotfuscator は処理中に Feature 属性を見つけた場合、[分析メッセージを送信] するよう構成されている場合には、Runtime Intelligence の機能の利用状況メッセージを送信するコードをそのメソッドに追加します。Feature 属性は実行時には必要ありません。そのため、Dotfuscator はこの属性を出力アプリケーションから取り除きます。

Feature 属性にはいくつかの引数やプロパティがあります。Runtime Intelligence Service を使って機能の利用状況の追跡を実装したい開発者は、次に挙げるプロパティの使い方を理解しておく必要があります。

名前

機能レベルの分析を理解するには、機能は名前で識別される必要があります。Name 引数は問題となっている機能の名前を定義する文字列値で、これは必須です。この名前は特定の規約に従う必要はありませんが、わかりやすい一意の名前にしてください。ただし、これは問題の Feature 属性が開始/停止ペアの一方である場合を除きます。この場合には、機能名は同一である必要があります。

イベントの種類

属性付きメソッドが実装する機能の部分を識別するために、**`FeatureEventType`** 引数を設定できます。**`FeatureEventType`** 引数は、次の値を伴う列挙です。

列挙値	実装
FeatureEventTypes.Tick	このメソッドは、指定された機能全体を実装します（既定値）。
FeatureEventTypes.Start	このメソッドの実行は、指定された機能の開始を示します。メッセージを送信するためのコードが、メソッドの先頭に追加されます。
FeatureEventTypes.Stop	このメソッドの実行は、指定された機能の停止を示します。メッセージを送信するためのコードが、メソッドの末尾に追加されます。

`FeatureEventType` 引数は省略可能です。省略すると、既定値（**`Tick`**）と見なされます。

メソッドが **`Find`** 機能の一部として呼び出される場合の、Feature 属性の使用例を示します。

Feature 属性の例

```
[Feature(
   "Find",
   FeatureEventType = FeatureEventTypes.Start
)]
private void BeginFind() { ...}
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

メソッドのロジックが機能を完全に含んでいる場合は、そのメソッドに 2 つの Feature 属性、start と stop を置くことができます。Dotfuscator は、メソッドの実行開始時に開始メッセージを送信し、メソッドの完了時に停止メッセージを送信します。

Feature 属性の Start と Stop

```
[Feature(
   "Find",
   FeatureEventType = FeatureEventTypes.Start
)]
[Feature(
   "Find",
   FeatureEventType = FeatureEventTypes.Stop
)]
private void BeginFind() { ...}
```

2.5.10.4.2.8 パフォーマンス属性

Runtime Intelligence コードを使用して、アプリケーションの実行中にパフォーマンスに関連する情報を収集し、送信することができます。アプリケーションにこのサポートを追加するには、アプリケーション内の 1 つ以上のメソッドに **PerformanceProbe** 属性を設定します。Dotfuscator は処理中にこの属性を見つけると、パフォーマンス情報を取得して、Runtime Intelligence Service へメッセージを送信するためのコードを追加します。

収集されるパフォーマンス データには次のものが含まれます。

- CPU 使用率
- 利用可能なメモリ
- 現在のプロセスで使用されているメモリ

パフォーマンス データ

```
[PreEmptive.Attributes.PerformanceProbe()]
public void DoSomething() { ...}
```

収集されたパフォーマンス データは、Runtime Intelligence ポータルの **Data Extract** レポートで入手できます。**File Feeds** セクションからダウンロードすることも可能です。

2.5.10.4.2.9 環境属性

Runtime Intelligence コードを使用して、アプリケーションが実行されているシステムに関する情報を収集し、送信することができます。アプリケーションにこのサポートを追加するには、アプリケーション内の 1 つのメソッドに **SystemProfile** 属性を設定します。Dotfuscator は処理中にこの属性を見つけると、システム プロファイルを収集して、Runtime Intelligence Service へメッセージを送信するためのコードを追加します。通常、このデータは、アプリケーションの実行中に 1 回だけ収集する必要があります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

以下は、収集されるシステム データの種類の概要です。

カテゴリ	収集されるデータの例
プロセッサ	プロセッサ数、クロック速度、製造元、プロセッサ ID
論理ディスク	論理ディスク数、ボリューム名、サイズ、空き領域、ファイル システム
メモリ	速度、容量
ネットワーク アダプター	IP アドレス、MAC アドレス
ドメイン	ドメイン名および役割
ディスプレイ	名前、リフレッシュ レート、垂直方向および水平方向の解像度
ビデオ	名前、メモリ サイズ、色数
ターミナル サービス	許可されている接続数
サウンド	名前、製造元
モデム	モデル、デバイスの種類

収集されたデータ

```
[PreEmptive.Attributes.SystemProfile()]
public void Initialize() { ...}
```

収集されたデータは、Runtime Intelligence ポータルの **Data Extract** レポートで入手できます。**File Feeds** セクションからダウンロードすることも可能です。

2.5.10.4.2.10 拡張キーを含むユーザー定義データの送信

Runtime Intelligence メッセージのほとんどの種類では、ユーザー定義データ(キーと値のペアの形式)を収集し、メッセージと一緒に送信することができます。

拡張キー情報を送信するには、送信したいメッセージに対応する属性に **`ExtendedKeySource`** を指定します。

Dotfuscator は **`ExtendedKeySource`** を利用して、実行時にキーと値のペアを収集するためのコードを生成します。**`ExtendedKeySource`** は、**`IDictionary`** または **`IDictionary<string,string>`** で評価されるプロパティ、メソッド、フィールド、あるいはメソッド引数です。開発者の責任において、属性付きメソッドが実行された時点で、必ず正しい値が **`ExtendedKeySource`** で入手できるようにしてください。

拡張キーをサポートする属性には、**`ExtendedKeySource`** を指定するための 3 つのプロパティが定義されています。

- **ExtendedKeySourceElement**。**`ExtendedKeySourceElement`** には、**`SourceElements`** の列挙(フィールド、プロパティ、メソッド、メソッド引数)に定義されているいずれの値も指定できます。**`ExtendedKeySourceElement`** がメソッド引数である場合には、属性がアタッチされるメソッドのメソッド パラメーターに対応している必要があります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- **ExtendedKeySourceOwner**。**`ExtendedKeySourceElement`** がフィールド、メソッドまたはプロパティである場合には、**`ExtendedKeySourceOwner`** はそのフィールド、メソッドまたはプロパティを定義しているクラスを指し示す必要があります。
- **ExtendedKeySourceName**。**`ExtendedKeySourceName`** には、実行時に拡張キーを格納する IDictionary のフィールド、メソッド、プロパティ、またはメソッド引数の名前を設定する必要があります。

拡張キーの設定はどんな場合でも省略可能です。省略すると、結果として生じる Runtime Intelligence メッセージに拡張キー情報は含まれません。

Feature 属性の ExtendedKeySource を、"**`GetDictionary`**" というメソッドとして定義する場合の使用例を示します。

Feature 属性の ExtendedKeySource の使用法

```
[Feature(
..."Click"
   ExtendedKeySourceElement = SourceElements.Method,
   ExtendedKeySourceName = "GetDictionary",
)]
private void button1_Click(object sender, EventArgs e) {
...
}

// 辞書を作成してカスタム データを入力する
public IDictionary<string, string> GetDictionary() {
  Dictionary<string, string> dict = new Dictionary<string, string>();
  dict.Add("key1", "val1");
  dict.Add("key2", "val2");
  return dict;
}
```

アプリケーションによって送信された拡張キー データは、Runtime Intelligence ポータルの Data Extract レポートで入手できます。"**File Feeds**" セクションからダウンロードすることも可能です。

2.5.10.4.2.11 メソッド パラメーターを拡張キーとして自動的に送信する

ユーザーが指定した名前と値のペアを実行時に収集して拡張キーで送ることに加え、インストルメント化されたメソッドのパラメーター名と値を自動的に収集し、拡張キー データのセットに追加することもできます。

拡張キー情報にパラメーター情報を追加するには、パラメーターと値を送信したいメソッドを修飾する属性に、**`ExtendedKeyMethodArguements`** を指定します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Dotfuscator は **ExtendedKeyMethodArguements** を利用して、指定したパラメーターの値を実行時に収集し、それらをキーと値の辞書に保管するためのコードを生成します。**ExtendedKeyMethodArguements** は、メッセージに含めるパラメーターを定義する文字列です。また任意で、レポートされるキーのうち、メッセージ データで名前付けするキーも定義します。

拡張キーをサポートする属性では **ExtendedKeyMethodArguement** が提供されます。

ExtendedKeyMethodArguements はどんな場合でも省略可能です。空白を指定するか、または省略すると、結果として生じる Runtime Intelligence メッセージにパラメーター情報は含まれません。

ExtendedKeyMethodArguements の値は以下のパターンの組み合わせで構成されます。

*	すべてのパラメーター
<param1>,<param2>	param1 および param2 の名前と値のみ
<param1>=<key1>,<param2>=<key2>	キー名がそれぞれ key1 と key2 の、param1 および param2 の値のみ

2.5.10.4.3 アプリケーション分析を備えた Dotfuscator のテストとデバッグ

一度 Dotfuscator を介してアプリケーションを実行したら、いつでもアプリケーション分析機能をテストできます。Runtime Intelligence を備えたアプリケーションが開始および停止されるとき、あるいは属性付き機能が使用されるとき、ホストされたサービスに適切なメッセージが送信されるはずです。正しい時点に正しいメッセージが送信されたことを確認する方法はいくつかあります。このセクションでは、クライアント側とサーバー側の検証技術について説明します。

2.5.10.4.3.1 メッセージ追跡の構成

開発者は下記のようにアプリケーション内でメッセージ追跡を構成し、出力を検査することによって、送信メッセージのクライアント側のトレースを獲得できます。

サンプルの App.config ファイル:

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<configuration>
  <system.diagnostics>
    <trace autoflush="true" indentsize="0">
      <listeners>
        <remove name="Default"/>
        <add name="myListener" type="System.Diagnostics.TextWriterTraceListener"
initializeData="c:\myListener.log" />
      </listeners>
      </trace>
    <switches>
      <add name="traceSwitch" value="4" />
    </switches>
  </system.diagnostics>
</configuration>
```

開発者は構成ファイル内の **<listeners>** 要素を使って、一部または全部のリスナーを追加、削除することができます。前の例を参照してください。既定値の追跡リスナーである **DefaultTraceListener**（Visual Studio の出力ウィンドウ）を削除して、追跡メッセージを c:\myListener.log へ書き込む **TextWriterTraceListener** を追加しています。

TraceSwitch

このクラスは、BooleanSwitch クラスによって提供される単純なオン/オフ制御の代わりに、複数レベルのサポートを提供します。TraceSwitch クラスは次の追跡レベルを使います。

追跡レベル	構成値	説明
Off	**0**	何のメッセージも追跡リスナーに出力しません
Error	**1**	エラー メッセージのみ追跡リスナーに出力します
Warning	**2**	エラー メッセージと警告メッセージを追跡リスナーに出力します
Info	**3**	情報、警告、およびエラー メッセージを追跡リスナーに出力します
Verbose	**4**	すべてのメッセージを追跡リスナーに出力します

メッセージ送信ランタイムで使用されるトレース スイッチの名前は "**traceSwitch**" ですが、追跡を実行するためには、構成ファイルのトレース スイッチ名とまったく同じ名前でなければなりません。

2.5.10.4.3.2 メッセージの監視

Fiddler（http://www.fiddlertool.com からダウンロードして入手できます）などの HTTP トラフィック記録ツールを使用すると、開発者は Runtime Intelligence メッセージをワイヤ上で監視できるようになりますが、それによってメッセージ ID を取得する別の方法を提供することにもなります。

> SetupAttribute で SSL をオフにしている場合は、メッセージのトラフィックの監視がより簡単になります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.5.10.4.3.3 メッセージ データのダウンロード

Runtime Intelligence ポータルは、.NET アプリケーションから発信される生のメッセージ データを安全にダウンロードする機能を提供します。データは、MS Excel と互換性のある CSV ファイルで入手できます。

生のメッセージ データを取得するには、開発者は http://www.runtimeintelligence.com の Runtime Intelligence ポータルにアクセスする必要があります。PreEmptive Solutions から提供された**ユーザー名**と**パスワード**を入力し、[**Data Extracts**]の下にある[**File Feeds**]にアクセスします。

2.5.10.4.4 Runtime Intelligence 対応アプリケーションの例

カスタム属性を使ってアプリケーション分析のインストルメンテーションを作動させる、単純な C# アプリケーション例を次に示します。

Sample C# Scenario

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
[assembly:Business("66ce94c5-08c2-4b3b-99e4-0a92a5bb3c17", "PreEmptive Solutions")]
[assembly:Application("8F6A00E8-0C11-433e-A683-30EB828C4B3C")]
[assembly:Binary("444745EB-92CE-45e8-A749-33A0A92364FC")]

Entry point:

   static class Program
   {
      /// <summary>
      /// Runtime Intelligence の使用例を示します。
      /// </summary>
      [STAThread]
      static void Main() {
         SoSetup("357-1113-1719");
         Application.Run(new Form1());
      }

      [Setup(
          OptInSourceElement = SourceElements.None,
          InstanceIdSourceElement = SourceElements.MethodArgument,
          InstanceIdSourceName = "instanceId",
          UseSSL = false
       )]
      public static void SoSetup(string instanceId) {
      // 空のメソッド。Dotfuscator がコードを提供します(メソッドが空
      // でない場合、Dotfuscator はメソッドの先頭にコードを追加します)
      }
   }

Exit point:

   public partial class Form1 :Form {
      [InsertTamperCheck()]
      public Form1() {
      // Dotfuscator はここにアプリケーションの整合性チェックと共に、整合性
      // チェックが失敗した場合にメッセージを送信するためのコードを追加します。
         InitializeComponent();
      }

      [Teardown]
      private void Form1_FormClosing(object s, FormClosingEventArgs e) {
      // 空のメソッド。Dotfuscator がコードを提供します(メソッドが空
      // でない場合、Dotfuscator はメソッドの末尾にコードを追加します)
      }
   }
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

SOSWinApp.exe のアプリケーションのスタートアップ メッセージ。このシリアル番号は 357-1113-1719 です。


```
<MessageCache>
  <InstanceId>357-1113-1719</InstanceId>
  <ApplicationGroupId>1cb4c0ba-f9b1-4935-a633-7cdc2ff6d38b</ApplicationGroupId>
  <Business>
    <CompanyName>PreEmptive Solutions</CompanyName>
    <CompanyId>66ce94c5-08c2-4b3b-99e4-0a92a5bb3c17</CompanyId>
  </Business>
  <TimeSentUtc>2007-10-01T21:46:00.2250193Z</TimeSentUtc>
  <ApiLanguage>.NET CLR</ApiLanguage>
  <ApiVersion>2.0.2795.27414</ApiVersion>
  <Id>1a97253b-045b-486d-a097-c8ca5eea0efd</Id>
  <Messages>
    <Message xsi:type="ApplicationLifeCycle">
      <SessionId>1cb4c0ba-f9b1-4935-a633-7cdc2ff6d38b</SessionId>
      <Event>
        <PrivacySetting>SupportOptout</PrivacySetting>
        <Code>Application.Start</Code>
      </Event>
      <Binary>
        <ModifiedDate>2007-10-01T17:40:53.947891-04:00</ModifiedDate>
        <Id>444745eb-92ce-45e8-a749-33a0a92364fc</Id>
        <Name>SOSWinApp, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null</Name>
        <Version>1.0.0.0</Version>
      </Binary>
      <TimeStampUtc>2007-10-01T21:45:50.2500433Z</TimeStampUtc>
      <Id>63d588c4-761a-4f6a-90fe-f3d89432a797</Id>
      <User>
        <IsAdministrator>false</IsAdministrator>
        <Name>724908d2f82cafe6b03e51438dcc5838</Name>
      </User>
      <Host>
        <RuntimeVersion>2.0.50727.42</RuntimeVersion>
        <IPAddress>172.16.7.42</IPAddress>
        <Name>04f07368bbabe92b7494f5fdd72c6476</Name>
        <OS>
          <OsInstallDate>2006-10-24T13:45:38</OsInstallDate>
          <OsName>Microsoft Windows XP Professional</OsName>
          <OsServicePackMajorVersion>2</OsServicePackMajorVersion>
          <OsServicePackMinorVersion>0</OsServicePackMinorVersion>
          <Locale>0409</Locale>
          <OSLanguage>1033</OSLanguage>
          <IsVirtualized>false</IsVirtualized>
        </OS>
      </Host>
```


© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
    </Message>
    <Message xsi:type="SessionLifeCycle">
      <SessionId>1cb4c0ba-f9b1-4935-a633-7cdc2ff6d38b</SessionId>
      <Event>
        <PrivacySetting>SupportOptout</PrivacySetting>
        <Code>Session.Start</Code>
      </Event>
      <Binary>
        <ModifiedDate>2007-10-01T17:40:53.947891-04:00</ModifiedDate>
        <Id>444745eb-92ce-45e8-a749-33a0a92364fc</Id>
        <Name>SOSWinApp, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null</Name>
        <Version>1.0.0.0</Version>
      </Binary>
      <TimeStampUtc>2007-10-01T21:45:50.4214882Z</TimeStampUtc>
      <Id>0e7c2426-dd62-44cd-ad66-4d9c05097af5</Id>
    </Message>
    <Message xsi:type="SessionLifeCycle">
      <SessionId>1cb4c0ba-f9b1-4935-a633-7cdc2ff6d38b</SessionId>
      <Event>
        <PrivacySetting>SupportOptout</PrivacySetting>
        <Code>Session.Start</Code>
      </Event>
      <Binary>
        <ModifiedDate>2007-10-01T17:40:53.947891-04:00</ModifiedDate>
        <Id>444745eb-92ce-45e8-a749-33a0a92364fc</Id>
        <Name>SOSWinApp, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null</Name>
        <Version>1.0.0.0</Version>
      </Binary>
      <TimeStampUtc>2007-10-01T21:45:50.4214882Z</TimeStampUtc>
      <Id>a44c5dd3-6328-47aa-82a2-63a28bf8270d</Id>
    </Message>
  </Messages>
  <SchemaVersion>02.00.00</SchemaVersion>
  <Application>
    <ApplicationType>.exe</ApplicationType>
    <Id>8f6a00e8-0c11-433e-a683-30eb828c4b3c</Id>
    <Name>SOSWinApp</Name>
    <Version>1.0.0.0</Version>
  </Application>
</MessageCache>
```


SOSWinApp.exe のアプリケーションのシャットダウン。このシリアル番号は 357-1113-1719 です。


```
<MessageCache>
  <InstanceId>357-1113-1719</InstanceId>
  <ApplicationGroupId>1cb4c0ba-f9b1-4935-a633-7cdc2ff6d38b</ApplicationGroupId>
```


© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<Business>
  <CompanyName>PreEmptive Solutions</CompanyName>
  <CompanyId>66ce94c5-08c2-4b3b-99e4-0a92a5bb3c17</CompanyId>
</Business>
<TimeSentUtc>2007-10-01T21:47:17.6401846Z</TimeSentUtc>
<ApiLanguage>.NET CLR</ApiLanguage>
<ApiVersion>2.0.2795.27414</ApiVersion>
<Id>2ab3e80c-2a37-4cbb-ae0c-905ba7f20f4d</Id>
<Messages>
  <Message xsi:type="SessionLifeCycle">
    <SessionId>1cb4c0ba-f9b1-4935-a633-7cdc2ff6d38b</SessionId>
    <Event>
      <PrivacySetting>SupportOptout</PrivacySetting>
      <Code>Session.Stop</Code>
    </Event>
    <Binary>
      <ModifiedDate>2007-10-01T17:40:53.947891-04:00</ModifiedDate>
      <Id>444745eb-92ce-45e8-a749-33a0a92364fc</Id>
      <Name>SOSWinApp, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null</Name>
      <Version>1.0.0.0</Version>
    </Binary>
    <TimeStampUtc>2007-10-01T21:47:17.5934269Z</TimeStampUtc>
    <Id>ae2d572d-6d22-4176-aa98-4ea69ef649fb</Id>
  </Message>
  <Message xsi:type="ApplicationLifeCycle">
    <SessionId>1cb4c0ba-f9b1-4935-a633-7cdc2ff6d38b</SessionId>
    <Event>
      <PrivacySetting>SupportOptout</PrivacySetting>
      <Code>Application.Stop</Code>
    </Event>
    <Binary>
      <ModifiedDate>2007-10-01T17:40:53.947891-04:00</ModifiedDate>
      <Id>444745eb-92ce-45e8-a749-33a0a92364fc</Id>
      <Name>SOSWinApp, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null</Name>
      <Version>1.0.0.0</Version>
    </Binary>
    <TimeStampUtc>2007-10-01T21:47:17.5934269Z</TimeStampUtc>
    <Id>086289bc-d8cb-453e-b5ea-b4ac238a4647</Id>
    <User>
      <IsAdministrator>false</IsAdministrator>
      <Name>724908d2f82cafe6b03e51438dcc5838</Name>
    </User>
    <Host>
      <RuntimeVersion>2.0.50727.42</RuntimeVersion>
      <IPAddress>172.16.7.42</IPAddress>
      <Name>04f07368bbabe92b7494f5fdd72c6476</Name>
```


© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
        <OS>
          <OsInstallDate>2006-10-24T13:45:38</OsInstallDate>
          <OsName>Microsoft Windows XP Professional</OsName>
          <OsServicePackMajorVersion>2</OsServicePackMajorVersion>
          <OsServicePackMinorVersion>0</OsServicePackMinorVersion>
          <Locale>0409</Locale>
          <OSLanguage>1033</OSLanguage>
          <IsVirtualized>false</IsVirtualized>
        </OS>
      </Host>
    </Message>
  </Messages>
  <SchemaVersion>02.00.00</SchemaVersion>
  <Application>
    <ApplicationType>.exe</ApplicationType>
    <Id>8f6a00e8-0c11-433e-a683-30eb828c4b3c</Id>
    <Name>SOSWinApp</Name>
    <Version>1.0.0.0</Version>
  </Application>
</MessageCache>
```


SOSWinApp.exe の改ざん検出メッセージ。このシリアル番号は 357-1113-1719 です。


```
<MessageCache>
  <InstanceId>357-1113-1719</InstanceId>
  <ApplicationGroupId>b38041f6-f01c-4a2a-8ce5-b6caaa7d39e8</ApplicationGroupId>
  <Business>
    <CompanyName>PreEmptive Solutions</CompanyName>
    <CompanyId>66ce94c5-08c2-4b3b-99e4-0a92a5bb3c17</CompanyId>
  </Business>
  <TimeSentUtc>2007-10-01T21:51:12.5359398Z</TimeSentUtc>
  <ApiLanguage>.NET CLR</ApiLanguage>
  <ApiVersion>2.0.2795.27414</ApiVersion>
  <Id>47da5317-4df6-4f15-85cd-f0a01ed5c00c</Id>
  <Messages>
    <Message xsi:type="SecurityMessage">
      <SessionId>b38041f6-f01c-4a2a-8ce5-b6caaa7d39e8</SessionId>
      <Event>
        <PrivacySetting>AlwaysSend</PrivacySetting>
        <Code>Security.Integrity.Tampering</Code>
      </Event>
      <Binary>
        <ModifiedDate>2007-10-01T17:50:18.4525198-04:00</ModifiedDate>
        <Id>444745eb-92ce-45e8-a749-33a0a92364fc</Id>
        <Name>SOSWinApp, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null</Name>
        <Version>1.0.0.0</Version>
```


© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
      </Binary>
      <TimeStampUtc>2007-10-01T21:51:02.7323458Z</TimeStampUtc>
      <Id>8f3c939a-79f3-4fbc-97b0-d8af5ef9ad4b</Id>
    </Message>
  </Messages>
  <SchemaVersion>02.00.00</SchemaVersion>
  <Application>
    <ApplicationType>.exe</ApplicationType>
    <Id>8f6a00e8-0c11-433e-a683-30eb828c4b3c</Id>
    <Name>SOSWinApp</Name>
    <Version>1.0.0.0</Version>
  </Application>
</MessageCache>
```

2.5.11 難読化されたスタック トレースのデコード

難読化の持つ潜在的な欠点の 1 つは、名前が改変されることにより、難読化処理されたアプリケーションのデバッグやトラブルシューティングが困難になる可能性があるということです。Dotfuscator はこの欠点を解決するために統合ツールを提供しています。このツールを使用すると、出力割り当てファイルを基に、難読化されたスタック トレースから元のシンボルを復元できます。

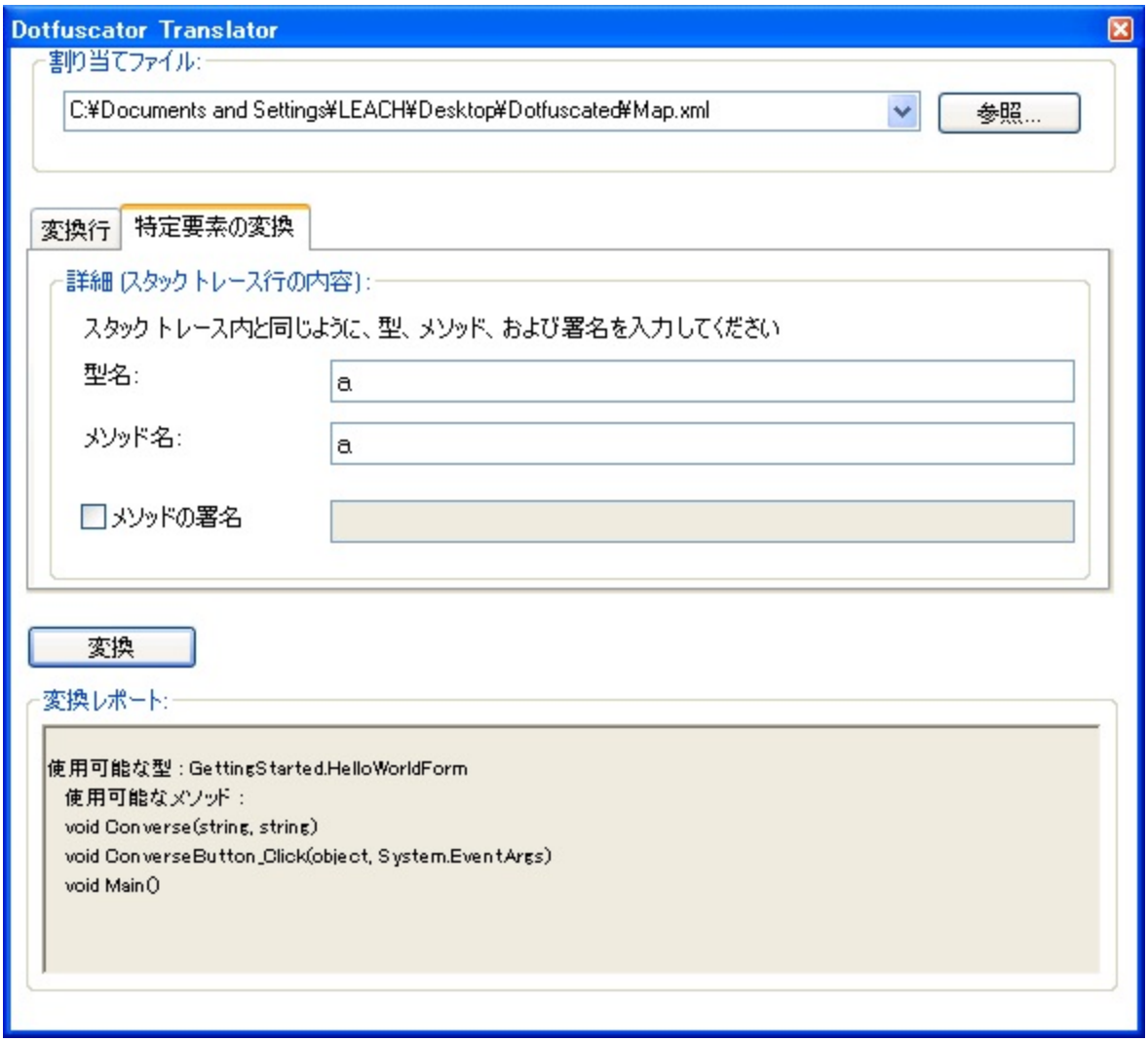
たとえば、難読化処理済みのアプリケーションを顧客に発送し、顧客の 1 人からスタック トレースを受け取った場合、そのスタック トレースは次のようなものになるでしょう。

スタック トレース

```
Unhandled Exception:System.ApplicationException:A bad thing happened!
   at a.a()
   at b.a(String A_0)
   at b.a(String[] A_0)
```

XML 形式の割り当てファイルか、できれば、その割り当てファイルを基に生成された HTML レポートを使用すれば、手動で元の名前を復元できますが、これは退屈で時間のかかる処理です。スタック トレースの変換ツールは、割り当てファイルを指定し、スタック トレースをウィンドウに貼り付けて[変換]ボタンを押すことにより、この処理を自動化します。変換されたスタック トレースが、下の画面のように表示されます。

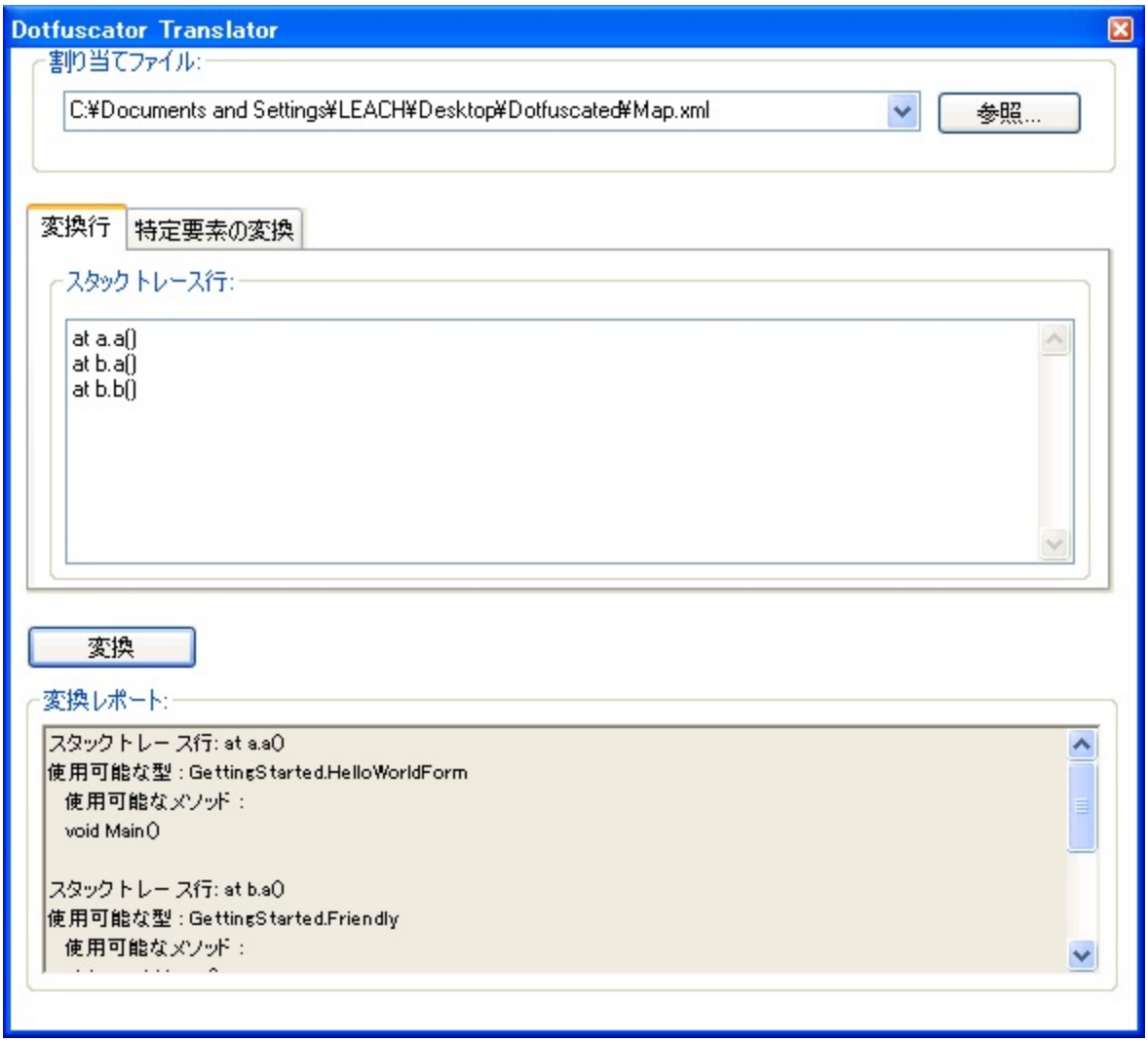
© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



難読化されたスタック トレースのうち、いくつかのメソッドはあいまいになる可能性があります。つまり、オーバーロード誘導や拡張オーバーロード誘導の使用によって、難読化されていないメソッドで一致するものが複数存在することがあるからです。このような場合、ツールはすべての可能性を表示します。

特定の型やメソッドを名前で検索したい場合は、[**特定要素の変換**]タブをクリックします。変換したい特定の項目の難読化処理後の名前を入力できる画面が表示されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



- 型を調べる場合は、[**型名**]テキスト ボックスに値を入力し、[**変換**]をクリックします。
- メソッドを調べる場合は、[**型名**]および[**メソッド名**]テキスト ボックスに値を入力します。署名の指定は任意で、[**メソッドの署名**]チェック ボックスをオンにし、その隣にあるテキスト ボックスに署名を入力することで指定できます。署名は、スタック トレースに示されるとおりの値である必要があります。

2.5.12 Lucidator の使用

スタック トレース全体を変換するには、Dotfuscator によって生成された割り当てファイルを開きます。次に、[**スタート | PreEmptive Solutions | Lucidator**]をクリックするか、またはコマンド ラインで次のコマンドを使用して、Lucidator を開きます。

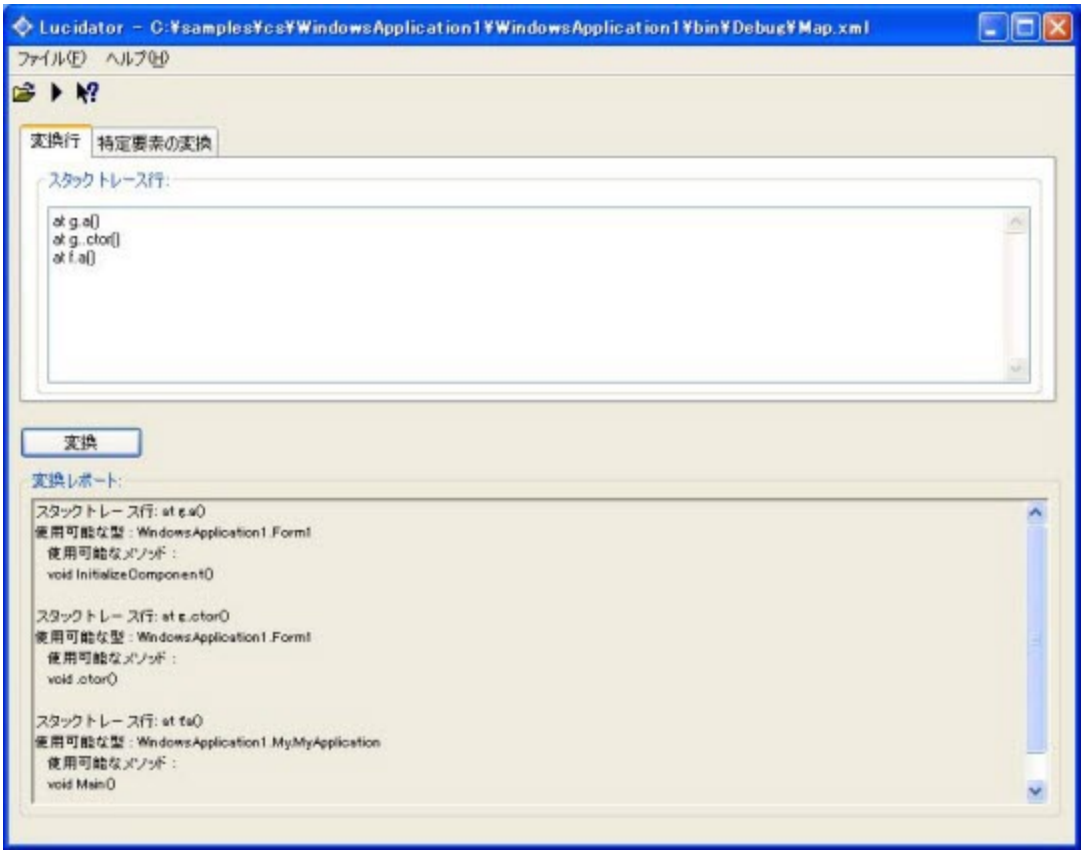
オプション	説明
`/mapfile=<map file>`	割り当てファイルを指定します(例:/mapfile=map.xml)
`/stacktracefile=<stack trace`	スタック トレースを含んでいるファイルを指定します

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

file>	(例:/stacktracefile=stacktrace.txt)
/c=<culture>	ユーザー インターフェイス言語を設定します(適切な言語リソースが必要です)。引数は小文字の言語コードです(例:/c=de、/c=ja、/c=zh-CHS)。

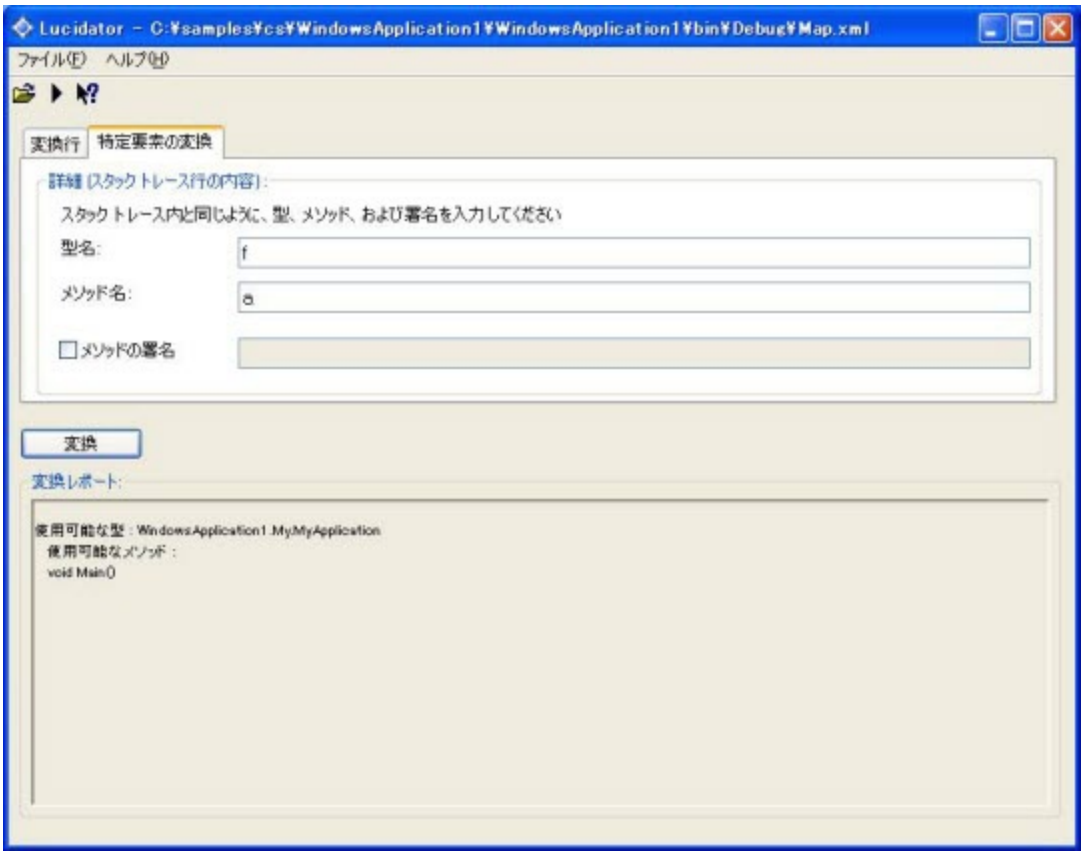
Lucidator ウィンドウがデスクトップに表示されたら、[**変換行**]タブを選択し、スタック トレースをウィンドウに貼り付けます。[**変換**]ボタンをクリックします。

変換されたスタック トレースがウィンドウの[**変換レポート**]セクションに表示されます。



難読化されたスタック トレース内のメソッドは、オーバーロード誘導や拡張オーバーロード誘導の使用が原因であいまいになる可能性があります。なぜなら、難読化されていないメソッドで一致するものが複数存在するかもしれず、Lucidator はすべての可能性を表示するからです。特定の型やメソッドを名前で検索する場合は、[**特定要素の変換**]タブをクリックします。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



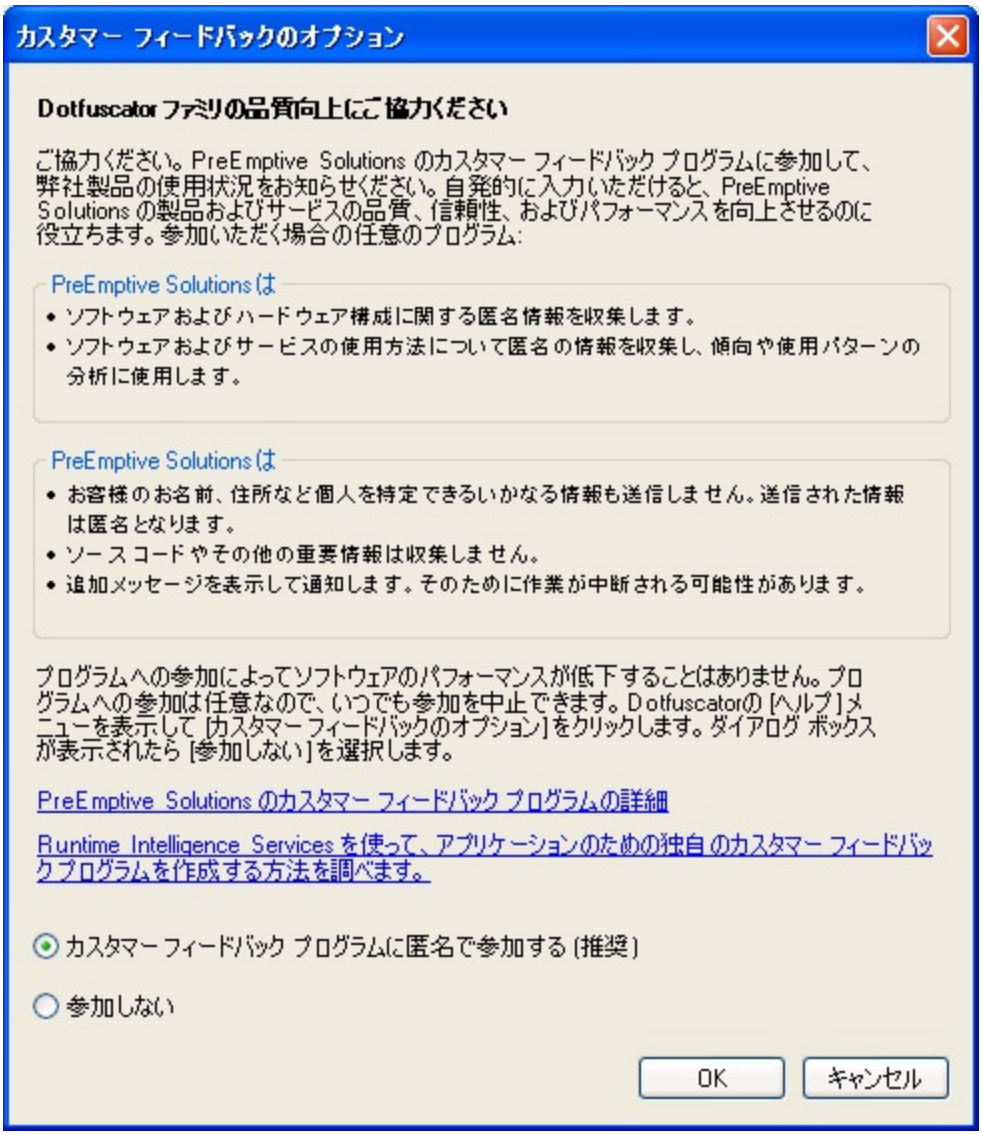
この画面で、変換したい特定の項目の難読化処理後の名前を入力します。たとえば、難読化された型名を変換する場合は、[**型名**]フィールドに難読化された名前を入力し、[**変換**]をクリックします。同様に、難読化されたメソッド名を変換する場合は、[**メソッド名**]フィールドに難読化された名前を入力し、[**変換**]をクリックします。署名の指定は任意で、[**メソッドの署名**]チェック ボックスをオンにし、その隣にあるテキスト ボックスに署名を入力することで指定できます。

> **メモ**: 署名は、スタック トレースに示されるとおりの値である必要があります。

2.5.13 カスタマー フィードバックのオプション

Dotfuscator は、匿名による使用状況レポート システムを提供しています。このプログラムへ参加するかどうかはユーザーが選択できます。このプログラムへ参加すると、PreEmptive Solutions によって高レベルな使用状況データが匿名で収集されるようになります。これは Dotfuscator ソフトウェア製品を改良することを目的としています。このオプションは、[ヘルプ | カスタマー フィードバックのオプション]メニューからいつでも変更することができます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.6 リファレンス

このセクションには、Dotfuscator のコマンド ライン構文、エディター、ビルドタスクの設定、構成ファイル オプション、およびカスタム属性への参照がすべて収められています。また、Dotfuscator の構成ファイルおよび割り当てファイルの DTD の参照先も収められています。

記事やホワイトペーパー、チュートリアルなどのその他のリソースは、www.preemptive.com/resources.html（英語）からダウンロードできます。

2.6.1 コマンド ライン インターフェイス リファレンス

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

コマンド ライン インターフェイスは、以下の操作を実行できるように設計されています。

- 構成ファイルの作成を必要とせずにコマンド ラインから難読化処理を実行する。
- コマンド ライン オプションを使用して、既存の構成ファイルで指定されているオプションをオーバーライドまたは補完する。
- コマンド ラインから構成ファイルを作成する。
- コマンド ラインでオプション、構成ファイル、またはその両方を指定して、スタンドアロン グラフィカル ユーザー インターフェイスを起動する。

2.6.1.1 コマンド ライン オプションの概要

コマンド ライン オプションは先頭にスラッシュ(**/**)またはハイフン(**-**)を付けます。

コマンド ライン オプション
使用法:`dotfuscator [options] [config_file]`

既存のオプション

以下に、既存のコマンド ライン オプションの概要を示します。

既存のオプション	説明
`/g`	スタンドアロン GUI を起動する。
`/i`	調査のみ実行する。
`/p=<property list>`	構成ファイルでユーザーが定義したプロパティの値を指定する。プロパティ リストには、名前と値のペアのリストをカンマで区切って記述する。(例:/p=projectdir=c:\\temp,projectname=MyApp.exe)
`/q`	メッセージを何も出力しない。
`/v`	詳細な情報を出力する。
`/nologo`	Dotfuscator の著作権およびライセンス情報の出力を抑制します。
`/?`	ヘルプを表示する。
`[config_file]`	実行時オプションが記述されている構成ファイルを指定する。

/v オプションを指定すると、実行中に処理の進行状況に関する情報が表示されます。このオプションで表示される情報の詳細レベルは、Dotfuscator のリリースによって異なる可能性があります。

/i オプションは、Dotfuscator に対して、出力アセンブリ ファイルを一切作成しないように指示します。構成ファイルによって割り当てファイルが指定されている場合、実行の結果はその割り当てファイルに格納されます(割り当てを生成しないでこのオプションを指定することはほとんど無意味です)。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

/**q** オプションは、Dotfuscator に対して、出力をまったく表示しないで実行するように指示します。このオプションは、アプリケーションのビルド工程に組み込んで使用する場合に適しています。このオプションを指定すると、詳細出力モードはオーバーライドされます。

/**p** オプションは、コマンド ラインで外部プロパティの値を設定するために使用します。コマンド ラインでプロパティの値を設定すると、構成ファイルの <**properties**> セクションで指定したプロパティの設定はオーバーライドされます。

<**proplist**> には名前と値のペアのリストをカンマで区切って記述します。-**p** オプションによるプロパティの宣言と値の割り当ては、たとえば次のような形で指定します。

-p オプションによるプロパティの宣言と値の割り当て:
`/p=projectdir=c:\temp,projectname=MyApp`

プロパティ値にスペースが含まれている場合は、次の例のように引用符で囲んで指定します。

プロパティ値を引用符で囲む
`/p=MyProperty="value has spaces"`

メモ: プロパティの名前では大文字と小文字が区別されます。

/**g** オプションは、Dotfuscator に対して、スタンドアロン グラフィカル ユーザー インターフェイスを起動するように指示します。このオプションを使用する際は、次のように -p オプションを併用することにより、外部プロパティや特定の構成ファイルを指定してグラフィカル ユーザー インターフェイスを起動することができます。

スタンド アロン GUI の起動
`Dotfuscator /g /p=projectdir=c:\temp project_template.xml`

グラフィカル ユーザー インターフェイスを起動するには、コマンド ライン引数を何も指定せずに Dotfuscator を実行するという方法もあります。

configfile は、Dotfuscator の毎回の実行に必要な構成ファイルです。構成情報や対象のアセンブリはコマンド ラインからは指定できないことに注意してください。これらの情報は構成ファイルに記述しておく必要があります。

追加オプション

追加オプションは、構成ファイルを事前に作成する必要なしにコマンド ラインから基本的な難読化処理を実行できるようにするためのものです。コマンド ラインの追加オプションで構成ファイルを使用した場合、構成ファイルで指定されているコマンドは、コマンド ラインで指定したオプションによって補完またはオーバーライドされます。詳細については、「コマンド ラインでのオプション指定による構成ファイルの補完またはオーバーライド」を参照してください。

追加オプションは先頭の 4 文字によって認識されます。

以下に、コマンド ラインの追加オプションの概要を示します。アスタリスク(*)は、そのオプションが指定されず、構成ファイルも指定されていない場合に使用される既定の設定を示します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

追加オプション	説明
/in [+|-]<file>[,[+|-]<file>]	入力を指定する。アセンブリ、パッケージ ファイルまたはディレクトリ ファイルのあらゆる組み合わせが指定できる。プレフィックスにより、入力アセンブリをライブラリ モード(**+**)とプライベート(**-**)のどちらとしてとして難読化するかを指定する。既定値のモードはアセンブリ ファイルの拡張子によって決定される(EXE はプライベート、.DLL はライブラリ モードで実行)。
/out:<directory>	出力ディレクトリを指定する。既定値のディレクトリは **.\Dotfuscated**。
/honor:[on|off*]	すべての入力アセンブリで見つかった難読化属性の使用命令を有効/無効にする。
/strip:[on|off*]	すべての入力アセンブリからの難読化属性の除去を有効/無効にする。
/makeconfig:<file>	すべての実行時オプション(コマンド ラインで指定されたオプション、および構成ファイルが存在する場合はそのファイルで指定されているオプション)を <file> に保存する。
/debug:[on:off*|impl|opt|pdb]	難読化されたアセンブリに対してデバッグ シンボルを生成し、JIT の動作を制御する。
/suppress:[on|off*]	サポートされる出力アセンブリへ SuppressIldasmAttribute を追加する。
/disable	他のオプションの設定にかかわらず、すべての変換を無効にする。
/rename:[on|off*]	名前の変更を有効/無効にする。
/mapout:<file>	出力割り当てファイルを指定する。 既定値のファイルは **.\Dotfuscated\map.xml**。
/mapin:<file>	入力割り当てファイルを指定する。
/clobbermap:[on|off*]	割り当てファイルの上書きモードを指定する。
/keep:[namespace|hierarchy|none*]	型の名前変更規則を指定する。
/prefix:[on:off*]	名前変更後のすべての型にプレフィックスを付ける。
/enhancedOI:[on|off*]	拡張オーバーロード誘導を使用する。
/refsrename:[on*|off]	入力割り当てファイルにのみ定義されている、参照されるメタデータの名前を変更する。
/naming: [loweralpha*|upperalpha|numeric|unprintable]	識別子の名前変更規則を指定する。
/controlflow:[high*|medium|low|off]	制御フローの難読化レベルを設定する。
/encrypt:[on*|off]	文字列の暗号化を有効化/無効化する。
/prune:[on*|off|const]	不要なコードの除去を有効化/無効化する。あるいは定数のみ除去を有効化する。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

/link:[[+]<name>[,[+]<name>],]out=<name>	名前付き出力アセンブリにリンクするアセンブリを指定する。'**+**' のプレフィックスは、プライマリ アセンブリであることを示す。リストを省略すると、入力のうち最初の入力をプライマリ アセンブリとして、すべての入力がリンクされる。コマンド ラインでは、複数のリンク オプションを指定できる。
/link:off	リンクを無効にする。リンク機能は、構成ファイルと共にリンク オプションを渡さない限り、既定値ではオフである。このオプションは、そのようなシナリオで役立つ。
/premark:[on|off*|only]	ウォーターマークを有効化/無効化する。"Only" オプションは、他のすべての変換処理を無効にする。
/watermark:<string>	ウォーターマーク文字列を指定する。引用符の使用は任意。既定で、すべての入力アセンブリがこの文字列によってウォーターマーク処理される。
/passphrase:<passphrase>	パスフレーズの指定は任意で、これはウォーターマーク文字列を暗号化する場合に使用される。
/charmap:<name>	ウォーターマーク文字列のエンコードに使用する文字コードを指定する。name は、サポートされている文字コードのうちの 1 つでなければならない。「文字コード」を参照。
/smart:[on|off]	スマート難読化を有効化/無効化する。「SmartObfuscation」を参照。
/soreport:[all|warn|none]	スマート難読化機能のレポート出力の詳細レベルを設定します。「SmartObfuscation」を参照。
/offlineactivation:<activation_file>	<activation_file> に含まれているアクティブ化データを使って Dotfuscator を手動でアクティブ化する。

例:

例 1

```
dotfuscator -in:my.dll
```

名前の変更、制御フローの難読化、不要コードの除去、および文字列の暗号化を有効にして、**my.dll** をライブラリとして難読化します(公開されているシンボルは除去されずに保持されます)。出力ディレクトリを指定していないので、出力アセンブリは既定の **.\Dotfuscated** ディレクトリに出力され、割り当てファイルとして **.\Dotfuscated\map.xml** が使用されます。

例 2

```
dotfuscator -in:-myapp.exe,-private.dll
```

myapp.exe および **private.dll** を 1 つのスタンドアロン アプリケーションとしてまとめて難読化します。DLL 内の公開されているシンボルも難読化されます。不要コードの除去は、myapp.exe 内のエントリ ポイント メソッドに基づいて有効化されます。

例 3

```
dotfuscator -in:myapp.exe -mapo:MyName.xml
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

myapp.exe をスタンドアロン アプリケーションとして難読化します。このコマンドでは名前変更の出力割り当てファイルが指定されています。

2.6.1.2 コマンド ラインでのオプション指定による構成ファイルの補完またはオーバーライド

Dotfuscator は、すべてまたは一部の設定を記述した構成ファイルを適用した上で、コマンド ラインでのオプションの指定によって構成ファイルの設定を補完またはオーバーライドできるという優れた特徴を備えています。そのため、標準的な構成ファイルをテンプレートとして使用しながら、その設定を必要に応じてすばやく調整することができます。

コマンド ライン オプション	構成ファイル オプション	メモ
`/in [+|-]<file>[,[+|-]<file>]`	*input* セクション	追加
`/out:<directory>`	*output* セクション	オーバーライド
`/honor:[on|off*]`	*inputassembly* セクション	オーバーライド
`/strip:[on|off*]`	*inputassembly* セクション	オーバーライド
`/debug:[on|off*|impl|opt|pdb]`	"`debug`" グローバル オプション	オーバーライド
`/suppress:[on|off*]`	"`suppressildasmattribute`" グローバル オプション	オーバーライド
`/disable`	*renaming*、*controlflow*、*stringencrypt* および *removal* の各セクションで "`disable`" オプションを設定	オーバーライド

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

/rename:[on:off]	"*renaming*" セクションで "disable" オプションを設定(または設定解除)	オーバーライド
/mapout:<file>	"*mapoutput*" セクション	オーバーライド
/mapin:<file>	"*mapinput*" セクション	オーバーライド
/clobbermap:[on|off]	"*mapoutput*" セクションの overwrite 属性	オーバーライド
/keep:[namespace|hierarchy|none]	"keepnamespace"、"keephierarchy" 名前変更オプションを設定(または設定解除)	オーバーライド
/enhancedOI:[on|off]	"enhancedOI" 名前変更オプションを設定(または設定解除)	オーバーライド
/refsrename:[on|off]	"mapinput" 要素の "obfuscatereferences" 属性	オーバーライド
/naming: [loweralpha|upperalpha|numeric|unprintable]	*renaming* セクションで "scheme" 属性を設定	オーバーライド

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

/controlflow:[high|medium|low|off]	"*controlflow*" セクションの "level" 属性を設定。off フラグは "disable" オプションを設定。	オーバーライド
/encrypt:[on|off]	*stringencrypt* セクションで "disable" オプションを設定(または設定解除)	オーバーライド
/prune:[on|off]	*removal* セクションで "disable" オプションを設定(または設定解除)	オーバーライド
/link:[[+]<name>[,[+]<name>],]out=<name>	<linkedassembly> 要素の子要素を設定	オーバーライド
/link:off	*linking* セクションで "disable" オプションを設定	オーバーライド
/premark:[on|off*|only]	*premark* セクションで "disable" オプションを設定(または設定解除)"only" 設定は構成ファイルに保存されない。	オーバーライド
/watermark	<watermark> 要素を設定	オーバーライド
/passphrase	<passphrase> 要素の設定および usepassphrase オプションの設定	オーバーライド

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

/charmap	*premark* セクションで "encoding" 属性を設定	オーバーライド

例

次の例では以下の構成ファイルを使用します。この構成は、出力割り当てファイルを指定して名前の変更を有効にしています。この構成ファイルは、例の中で "**myconfig.xml**" として参照されています。

例 1

```
<?xml version="1.0"?>
<!DOCTYPE dotfuscator SYSTEM
"http://www.preemptive.com/dotfuscator/dtd/dotfuscator_v2.2.dtd">
<dotfuscator version="2.2">
   <renaming>
      <mapping>
         <mapoutput overwrite="true">
            <file dir="${configdir}\reports" name="MyMap.xml"/>
         </mapoutput>
      </mapping>
   </renaming>
</dotfuscator>dotfuscator -in:my.dll myconfig.xml
```

このコマンドは、**my.dll** をライブラリ モードの入力アセンブリとして指定し（拡張子が DLL であるため、既定値としてライブラリ モードになります）、それに対して、構成ファイルで設定されている名前変更オプションを適用します。この例の場合、制御フローの難読化、文字列の暗号化、および不要コードの除去は構成ファイルで暗黙的に無効化されているので、これらの処理は実行されません。

出力ディレクトリは構成ファイルとコマンド ラインのいずれでも指定されていないため、出力 DLL は **".\Dotfuscated"** ディレクトリに格納されます。

例 2

```
dotfuscator -in:my.dll -keep:namespace -enha:on -cont:high myconfig.xml
```

また、このコマンドは **my.dll** を入力アセンブリとして指定します。さらに、このコマンドでは、名前の変更処理において名前空間を保持し、拡張オーバーロード誘導を使用するように指示しています。また制御フローの難読化も、最大限の難読化を行う "high" レベルに設定して有効化されています。

2.6.1.3 コマンド ラインからの構成ファイルの保存

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

アプリケーションに適したコマンド ライン オプションの設定が確認できたら、/makeconfig オプションを使用して、それらの設定を構成ファイルに保存することができます。このオプションを使用すると、コマンド ラインで指定したすべてのオプションが（既存の構成テンプレートが存在する場合はそのファイルの設定と結合されて）カスタム構成ファイルとして保存されます。保存された構成ファイルは、それ以後の実行時に単独で使用することができます。

例

```
dotfuscator -in:my.dll -keep:namespace -enha:on -cont:high -make:new.xml myconfig.xml
```

この例で生成される構成ファイル（**new.xml**）の内容を見ると、以下のように、コマンド ラインで指定されたオプションが元の構成ファイル（**myconfig.xml**）の設定と結合されていることがわかります。

構成ファイル

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE dotfuscator SYSTEM
"http://www.preemptive.com/dotfuscator/dtd/dotfuscator_v2.2.dtd">
<dotfuscator version="2.2">
  <input>
    <asmlist>
      <inputassembly>
        <option>library</option>
        <file dir="." name="my.dll" />
      </inputassembly>
    </asmlist>
  </input>
  <output>
    <file dir="C:\MSProjects\dotfuscatortest\Dotfuscated" />
  </output>
  <renaming>
    <option>enhancedOI</option>
    <option>keepnamespace</option>
    <mapping>
      <mapoutput overwrite="true">
        <file dir="${configdir}\reports" name="MyMap.xml" />
      </mapoutput>
    </mapping>
  </renaming>
  <controlflow level="high" />
</dotfuscator>
```

2.6.1.4 コマンド ラインからのグラフィカル ユーザー インターフェイスの起動

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

コマンド ライン オプションを何も指定せずに Dotfuscator を実行すると、スタンドアロン グラフィカル ユーザー インターフェイスが起動して、空のプロジェクトが開かれた状態になります。

もう 1 つの方法として、**/g** オプションを有効なコマンド ライン オプションの任意な組み合わせと一緒に指定することにより、これらのオプションが適用された状態でグラフィカル ユーザー インターフェイスを起動することもできます。

例:

```
dotfuscator -g -in:my.dll myconfig.xml
```

このコマンドはスタンドアロン グラフィカル ユーザー インターフェイスを起動します。**myconfig.xml** の設定が読み込まれ、**my.dll** が入力アセンブリとして設定されます。

2.6.2 MSBuild タスク リファレンス

バージョン 2.0 以降の .NET Framework には、MSBuild と呼ばれるタスク ドリブンのビルド エンジンが付属しています。Dotfuscator は、PreEmptive.Dotfuscator.Tasks.dll に定義されているタスクを使用して、MSBuild インターフェイスを提供します。この DLL は、以下に示す MSBuild 拡張ディレクトリのサブディレクトリにインストールされます。

MSBuild 拡張ディレクトリ:

```
$(MSBuildExtensionsPath)\PreEmptive\Dotfuscator\<major>
$(MSBuildExtensionsPath)\PreEmptive\Dotfuscator\<major>\<major>.<minor>
$(MSBuildExtensionsPath)\PreEmptive\Dotfuscator\<major>\<major>.<minor>\<major>.<minor>.<patch>
```

たとえば、バージョン `4.11.5` の `Dotfuscator` がインストールされた場合は、次のようなディレクトリになります。

```
$(MSBuildExtensionsPath)\PreEmptive\Dotfuscator\4
$(MSBuildExtensionsPath)\PreEmptive\Dotfuscator\4\4.11
$(MSBuildExtensionsPath)\PreEmptive\Dotfuscator\4\4.11\4.11.5
```

MSBuild スクリプトにインポートすることができるターゲット テンプレート、PreEmptive.Dotfuscator.Targets もあります。このファイルは同じディレクトリにインストールされます。

レイアウトは、独自の MSBuild ファイルを書くときに、目的とする特定のバージョンのほとんどを指し示すことができるように設計されています。詳細については、「サイド バイ サイド インストール」のセクションを参照してください。

2.6.2.1 Dotfuscate タスク

Dotfuscate タスクを使用して、MSBuild から Dotfuscator を実行することができます。以下の表は、Dotfuscate タスクによって提供されるプロパティを示しています。

プロパティ	型	説明

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ConfigPath	**String**	書き込み専用。Dotfuscator 構成ファイルへのパスを設定します。
DebugSymbols	**String[]**	読み取り専用。出力アセンブリに関連付けられている PDB ファイルを公開します。**PreEmptive.Dotfuscator.Targets** ファイルは、このプロパティを **DotfuscatedDebugSymbols** という名前の出力項目として公開しています。
InputAssemblies	**ITaskItem[]**	書き込み専用。現在は、入力アセンブリも構成ファイルに列挙する必要があります。これは、Visual Studio で生成されたプロジェクトファイルの相互運用のためにのみ存在します。将来、変更される可能性があります。
MappingFile	**String**	読み取り専用。名前変更の割り当てファイルを公開します。**PreEmptive.Dotfuscator.Targets** ファイルは、このプロパティを **DotfuscatorMappingFile** という名前の出力項目として公開しています。
OutputAssemblies	**String[]**	読み取り専用。出力アセンブリを公開します。**PreEmptive.Dotfuscator.Targets** ファイルは、このプロパティを **DotfuscatedAssemblies** という名前の出力項目として公開しています。
ReportFiles	**String[]**	読み取り専用。HTML 形式の名前の変更レポートや除去レポートなどのレポート ファイルを公開します。**PreEmptive.Dotfuscator.Targets** ファイルは、このプロパティを **DotfuscatorReportFiles** という名前の出力項目として公開しています。
SatelliteAssemblies	**String[]**	読み取り専用。出力アセンブリに関連付けられているサテライト アセンブリを公開します。**PreEmptive.Dotfuscator.Targets** ファイルは、このプロパティを **DotfuscatedSatelliteAssemblies** という名前の出力項目として公開しています。

Properties	**String**	書き込み専用。ユーザー定義の外部プロパティを設定します。文字列は、キー/値のペアを表す子要素を持つ、有効な XML 要素を含んでいる必要があります。
		
		たとえば、次のようになります。
		
		`<Properties>`
		
		`<Property1>Value1</Property1>`
		
		`<Property2>2</Property2>`
		
		`</Properties>`

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Visual Studio 2005 以降の Dotfuscator プロジェクト ファイル

Visual Studio 2005 以上で Visual Studio 2005 統合バージョンをお使いの場合、作成した Dotfuscator プロジェクト ファイルは自動的に MSBuild 形式であり続けます。そのため、Visual Studio 外で MSBuild を使用するビルド マシンでもプロジェクト ツリーを取り入れることができ、Visual Studio 内でと同じようにビルドを実行できます。

2.6.2.2 PreMark タスク

PreMark タスクを使用すると、以前にウォーターマーク設定したアセンブリからウォーターマークを抽出できます。これは、Dotfuscator インストール ディレクトリにある、コマンド ライン プログラムの Premark.exe と類似した機能を提供します。

プロパティ	型	説明
InputAssemblyPath	**String**	必須、読み書き。ウォーターマークの抽出を行うアセンブリへのパスを取得または設定します。
UsePassphrase	**Boolean**	読み書き。
Passphrase	**String**	読み書き。現在は、入力アセンブリも構成ファイルに列挙する必要があります。これは、Visual Studio で生成されたプロジェクト ファイルの相互運用のためにのみ存在します。将来、変更される可能性があります。
Watermark	**String**	読み取り専用。抽出されたウォーターマーク文字列を公開します。

2.6.3 構成ファイル リファレンス

Dotfuscator 構成ファイルには、任意の名前または拡張子を付けることができますが、通常の拡張子は `.xml` です。構成ファイルには、指定したアプリケーションが Dotfuscator でどのように難読化処理されるかについての情報が含まれています。構成ファイルは、「リファレンス」のセクションで説明している、`dotfuscator_v2.3.dtd`(またはそれ以前のもの)に準拠する XML ドキュメントです。

このセクションでは、Dotfuscator の XML 構成ファイルについて説明します。Visual Studio、スタンドアロン GUI、コマンドラインのいずれのインターフェイスを使用している場合にもリファレンスとして役立つように、各構成オプションを詳しく解説します。

2.6.3.1 バージョン

.xml ファイルの Version 属性は必須情報であり、ご利用になる Dotfuscator のバージョンに適合している必要があります。Version 属性は、準拠する DTD のバージョン番号に一致する必要があります。Dotfuscator のある時点のリリースは、以前のバージョンから変更されていない構成ファイルを使用できるように設計されています。たとえば、Version 1.0 の構成ファイルを使用して Dotfuscator 1.1 を実行できます。このとき、構成ファイルを編集する必要はありません。

> **メモ**: 構成ファイルのバージョンは、必ずしも Dotfuscator のバージョンと同じであるとは限りません。Dotfuscator のどのバージョンも、同じバージョンではなくても特定のバージョンの構成ファイルを参照することを想定しています。

バージョン

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<dotfuscator version="2.2">
```

2.6.3.2 プロパティ リストとプロパティ

<propertylist>(プロパティ リスト)セクションは省略可能であり、構成ファイルで後から利用される可能性のある(**<プロパティ>** と呼ばれる)変数の定義および値の代入を行うことができます。このセクションで定義したプロパティ定義は、*内部*プロパティと呼ばれます。

内部プロパティ

```
<!-- 拡張可能なプロパティを定義します -->
<!-- 省略可能 -->
<propertylist>
  <property name="projectname" value="myproject"/>
  <property name="projectdir" value="c:\myprojects"/>
</propertylist>
```

このセクションで変数またはプロパティ参照を定義せずに、構成ファイルでこれらを使用することができます。たとえば、コマンド ラインで変数を定義したり、環境変数を利用することができます。プロパティは、次のアルゴリズムを使用してそのプロパティに関連付けられた値を検索し、文字列の置き換えを行うことによって機能します。

1. 外部のプロパティ リストで値を確認します。
2. 見つからない場合は、プロパティと同じ名前の環境変数を確認します。
3. それでも見つからない場合、構成ファイルの propertylist セクションで内部定義を確認します。
4. それでも見つからない場合は、値に空の文字列を使用します。

外部のプロパティは、**-p** オプションを使用して、コマンド ラインから渡されます。組み込みの外部プロパティには次の 3 つがあります。

- Dotfuscator のインストール ディレクトリを表す "**applicationdir**"
- Dotfuscator のローカル データ ディレクトリを表す "**appdatadir**"
- 構成ファイルが存在するディレクトリを示す "**configdir**"

プロパティは、複数のプロジェクトや同じプロジェクトのさまざまなバージョン、あるいはさまざまなビルド環境間での移植を容易にするためのテンプレートとしての役割を果たす構成ファイルの作成に役立ちます。プロパティは、次の構文で参照されます。

プロパティの構文

```
${property_name}
```

プロパティ参照は大文字と小文字を区別するので、**${MyProjectDir}** は **${myprojectdir}** とは異なるプロパティを参照します。現在、プロパティの参照は、**<file>** 要素の **dir** 属性または **name** 属性の値としてのみ使用でき、構成ファイルのどの場所でも使用できるわけではありません。**<file>** 要素を使用するセクションの一覧は以下のとおりです。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

inputassembly	mapinput	mapoutput
output	tempdir	assembly
removalreport	transform	key
loadpaths	program	filelist
smartobfuscationreport	pfx	

プロパティ参照は、構成ファイル内のその他の場所では文字どおりに解釈されます。プロパティ参照は、入れ子にすることはできません。入れ子にすると、エラーになります。 プロパティ参照の使用例は、次のとおりです。

プロパティ参照の使用例は、次のとおりです。

```
<output>
  <file dir="${testdir}\output"/>
</output>
```

2.6.3.3 グローバル セクション

グローバル セクションは省略可能であり、実行全体にわたって適用する構成オプションを定義するためにあります。このセクションでは、各グローバル オプションについて詳しく説明します。

メモ: グローバル オプションは、大文字と小文字を区別しません。

2.6.3.3.1 Library グローバル オプション

このオプションは Dotfuscator 3.0 で非推奨となり、個々の入力アセンブリに対して適用できる、より詳細な library オプションに置き換えられました。Dotfuscator が旧バージョンの構成ファイルを読み込んだ場合にも、このオプションは有効なものとして処理されますが、ユーザー インターフェイスでその構成ファイルを保存すると、新しいオプションが使用されます。「アセンブリ単位のライブラリ モード」を参照してください。

Library グローバル オプション

```
<global>
 <!—set library option -->
 <option>library</option>
</global>
```

2.6.3.3.2 Verbose、Quiet、Investigate グローバル オプション

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

これらのオプションは、対応するコマンド ライン オプションと同じです。コマンド ライン オプションを設定すると、実行時にグローバル オプションが設定されます。反対に、グローバル オプションを設定し、コマンド ライン オプションを設定していない場合は、グローバル オプションが優先します。つまり、コマンド ラインから設定済みのグローバル オプションの設定を解除する方法はありません。

Verbose、Quiet、Investigate グローバル オプション

```
<global>
 <!-- 詳細表示モードで実行します -->
 <option>verbose</option>
 <!-- メッセージ非表示モードで実行します -->
 <option>quiet</option>
 <!-- 調査のみを実行して割り当てファイルを生成します -->
 <option>investigate</option>
</global>
```

2.6.3.3.3 SuppressIldasmAttribute グローバル オプション

このオプションを設定することにより、Microsoft の Ildasm ユーティリティがアセンブリ IL を表示しないよう Dotfuscator に指示します。これは .NET 2.0 以上を対象とするアセンブリにのみ有効です。

SuppressIldasm オプション

```
<global>
   <option>suppressildasm</option>
</global>
```

2.6.3.3.4 Debug グローバル オプション

これらのオプションを設定すると、各出力アセンブリに対して、シンボル ファイルが PDB 形式で作成されます。デバッガーにこれらのファイルを使用すると、デバッグ セッションに関する有益な情報を確認することができます。通常は、行番号、ソース ファイル名、ローカル変数名などの情報です。PDB ファイルは出力アセンブリと一緒に出力ディレクトリに置かれます。

このオプションは、既に PDB ファイルと関連付けられているアセンブリ、つまり、デバッグ シンボルを生成することをコンパイルに知らせているアセンブリに対して使います。このような場合、Dotfuscator は元の PDB ファイルの情報を利用して、難読化されたアセンブリ用に新しい PDB ファイルを作成します。このとき、行番号とソース ファイル名は元のアセンブリの情報に対応したものとなります。

入力 PDB ファイルがない場合は、Dotfuscator で作成される PDB ファイルには、難読化されたアセンブリ内の低レベルな MSIL（Microsoft Intermediate Language）命令に対応する行番号が含まれます。

Debug、DebugImpl、DebugOpt、および Pdb グローバル オプション

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<global>
   <!-- JIT 最適化を無効にし、PDB ファイルを作成して PDB のシーケンス ポイントを使用 -->
   <option>debug</option>
   <!-- JIT 最適化を無効にし、PDB ファイルを作成して PDB のシーケンス ポイントを使用 -->
   <option>debugimpl</option>

   <!-- JIT 最適化を有効にし、PDB ファイルを作成して暗黙のシーケンス ポイントを使用 -->
   <option>debugopt</option>

   <!-- デバッグ情報の追跡を有効にしないで PDB ファイルを作成する -->
   <option>pdb</option>
</global>
```

DebugImpl、DebugOpt および Pdb グローバル オプションは .Net 2.0 以上を対象とするアセンブリに対してサポートされます。これらのオプションが .Net 1.0 または 1.1 を対象とするアセンブリで使用された場合は、代わりに Debug グローバル オプションが使用されます。

2.6.3.3.5 NoDotfuscatorAttribute グローバル オプション

Dotfuscator は既定値では、処理するアプリケーションに "**DotfuscatorAttribute**" という名前のカスタム属性を挿入します。この属性には、プログラムの難読化に使用する Dotfuscator のバージョンに関する情報が格納されます。この情報には製品 ID（CE あるいは PE）とバージョン番号があります。これの意図するところは次の 3 つです。

- プログラムに対して使用された Dotfuscator のバージョンを確認するため。
- Dotfuscator の将来のバージョンでは、この情報を使用して、入力として指定されたサード パーティ製のアセンブリが難読化処理済みであるかどうかを識別し、そうであった場合に特別な処置をするようにするため。
- PreEmptive Solutions 社は他社のツール ベンダーと協力して、難読化処理されたコードをデバッグする際の苦痛を和らげることに積極的に取り組んでおり、それらのツールがこの属性によって処理対象を識別できるようにするため。

上の事項が、「匿名の」難読化済みプログラムのセキュリティを向上させることほど重要でない場合、あるいはサイズを縮小させることの方が優先度が高い場合には、構成ファイル内に **nodotfuscatorattribute** というグローバル オプションを手作業で設定することにより、このような属性の挿入を行わないようにできます。

NoDotfuscatorAttribute グローバル オプション

```
<global>
   <option>nodotfuscatorattribute</option>
</global>
```

2.6.3.4 入力アセンブリ リスト

入力アセンブリ リストには、Dotfuscate で処理するアセンブリおよび/またはパッケージのファイル名とディレクトリを含めます。また、パッケージまたはアセンブリ レベルで設定する構成オプションもここに含めます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

複数のモジュールから成るアセンブリの場合は、マニフェストを持つモジュールだけを含めます。

入力アセンブリ リスト

```
<input>
   <asmlist>
      <inputassembly>
       ...
        <file dir="c:\temp" name="myproj.dll"/>
       </inputassembly>
     ...
   </asmlist>
</input>
```

2.6.3.4.1 アセンブリ単位のライブラリ モード

この設定は、特定の入力アセンブリをライブラリの構成要素とすることを Dotfuscator に指示します。Dotfuscator にとってライブラリは、Dotfuscator の入力に含まれてない他のコンポーネントから参照されるアセンブリです。したがって、カスタムの対象除外を設定しているかどうかにかかわらず、名前の変更や不要コードの除去について注意すべき点があります。

library オプションを使用する際の規則を次に示します。

- パブリック クラスの名前と、入れ子になったパブリック クラスの名前は変更されません。これらのクラスのメンバー（フィールドおよびメソッド）のアクセス属性が public、family、famorassem である場合も、名前は変更されません。
- さらに、アクセス指定子にかかわらず、仮想メソッドの名前も変更されません。これによって、ライブラリのクライアント側ではプライベートな仮想メソッドを必要に応じてオーバーライドできるようになります。
- 上記の規則によって暗黙に適用される対象除外とともに、名前の変更に関するユーザー指定のカスタム対象除外が適用されます。
- プロパティとイベントのメタデータは、常に維持されます。
- 不要なコードを除去する規則は以下のとおりです。
  - パブリック クラスは、静的分析が不要と判断しても取り除かれません。
  - パブリック クラスのフィールドは、これらのアクセス属性が public、family、または famorassem である場合も取り除かれません。
  - パブリック クラスのメソッドは、これらのアクセス属性が public、family、または famorassem である場合も取り除かれません。さらに、このようなメソッドはエントリ ポイントとして処理されるため、その後に呼び出しツリーおよび呼び出し先メソッドが続きますが、これらのメソッドもすべて除去から免れます。

アセンブリに library オプションを指定しなかった場合、入力アセンブリはスタンドアロン アプリケーションとして、または他の入力アセンブリからのみ参照されるものとして処理されます。この場合、次のように難読化の度合いがより高くなります。

- アプリケーションの外部のクラス（つまり、Dotfuscator の処理に含まれないアセンブリのクラス）をオーバーライドするメソッドを除いて、すべての名前が変更されます。
- プロパティとイベントのメタデータは、アプリケーションの実行に不要なため取り除かれます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- 名前の変更に関するユーザー指定のカスタム対象除外も適用されます。
- 不要なコードを除去する規則は以下のとおりです。
  - 特に、対象選択されているクラス、メソッド、フィールドは除去されません。
  - すべてのトリガー メソッドおよびフィールドは除去されません。
  - 名前の変更から除外されるすべてのクラス、メンバー、およびフィールドも除去の対象から外されます。
  - それ以外の、対象となる一部のクラス、フィールド、またはメソッドから到達できないすべてのクラス、フィールド、メソッドは除去されます。

入力アセンブリにライブラリ モードを指定するには、その入力アセンブリの **<inputassembly>** 要素に **<option>** 要素を追加します。

アセンブリ単位のライブラリ モード

```
<inputassembly>
  <option>library</option>
  <file dir="c:\temp" name="myproj.dll"/>
</inputassembly>
```

2.6.3.4.2 アセンブリ単位の宣言による難読化

Dotfuscator の「宣言による難読化」のサポートの詳細については、「カスタム属性を使用した宣言による難読化」を参照してください。

宣言による難読化の有効化と無効化

Dotfuscator では、入力アセンブリに対する宣言による難読化を有効または無効に切り替えることができます。無効になっている場合、難読化に関連するカスタム属性は無視されます。

構成ファイルを設定して宣言による難読化を有効にするには、各 **<inputassembly>** 要素に **honorOAs** オプションを追加します。

入力アセンブリ要素への honorOAs オプションの追加

```
<inputassembly>
  <option>honoroas</option>
  ...
</inputassembly>
```

宣言による難読化の属性の除去

構成ファイルを設定して難読化属性の除去を指定するには、各 **<inputassembly>** 要素に **stripOAs** オプションを追加します。

入力アセンブリ要素への stripOAs オプションの追加

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<inputassembly>
  <option>stripoa</option>
  ...
</inputassembly>
```

2.6.3.4.3 アセンブリ単位のインストルメンテーション処理

Dotfuscator では、特定の入力アセンブリのインストルメンテーションを制御することができます。このオプションの設定が及ぼす影響については、「>アプリケーション分析を備えた Dotfuscator の構成および実行」のセクションで説明します。

インストルメント変換処理のいずれかが有効になっている場合、Dotfuscator はインストルメンテーション属性を受け取り、不要なインストルメンテーション属性を出力アセンブリから除去します。既定の動作は、次のオプションの一方または両方を指定することにより、入力アセンブリ レベルで変更できます。

- **nohonorsos** は、Dotfuscator がインストルメンテーション属性の指示に従って動作しないようにします。
- **nostripsos** は、Dotfuscator が不要なインストルメンテーション属性を除去しないようにします。

Instrumentation Processing by Assembly

```
<inputassembly>
  <!-- インストルメンテーション属性を除去しません -->
  <option>nostripsos</option>
  <!-- インストルメンテーション属性を使用しません -->
  <option>nohonorsos</option>
  ...
</inputassembly>
```

2.6.3.4.4 アセンブリ単位の XAML の変換モード

この設定は、特定の入力アセンブリに次のマークアップを含めることができることを Dotfuscator に示します。そのマークアップは、Silverlight アプリケーションで用いられる XAML、あるいは Windows Presentation Foundation アプリケーションで用いられるコンパイル済みの XAML リソース（BAML）です。いずれのマークアップも、名前変更のために分析され対象に加えられます。Dotfuscator で処理するため、変換されるマークアップは要素の分離コード参照とともに名前変更される識別子を持ちます。マークアップ リソースから参照されるプロパティは、そのプロパティ メタデータは保持されますが、名前が変更されます。

入力アセンブリに XAML の変換モードを指定するには、その入力アセンブリの **<inputassembly>** 要素に **<option>** 要素を追加します。

アセンブリ単位の XAML の変換モード

```
<inputassembly>
  <option>transformxaml</option>
  <file dir="c:\temp" name="myproj.dll"/>
</inputassembly>
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.3.5 ユーザー定義のアセンブリ ロード パス

入力アセンブリ内で使用している型の情報を知るためには、入力アセンブリによって参照されるアセンブリを読み込む必要があります。Dotfuscator は、Visual Studio や CLR 自体で使用される規則に類似した検索規則を使用します。

既定値の検索規則では参照されるアセンブリが見つからない場合のために、検索するディレクトリを追加指定する手段が用意されています。Dotfuscator はアルゴリズムの最終ステップとして、これらのディレクトリを指定された順序で検索します。ただし、追加オプション（プロジェクト プロパティおよび[設定]タブの[アセンブリ ロード パス]にある[**最初に検索**]チェック ボックス）が使用されていると、Dotfuscator はその標準検索を適用する前にロード パスから検索します。

ユーザー定義のアセンブリ ロード パスを XML 構成ファイルに追加するには、次のようにします。

ユーザー定義アセンブリ ロード パスの追加

```
<input>
  <loadpaths>
    <option>prepend</option>
    <file dir="C:\temp" />
     ...
  </loadpaths>
  ....
</input>
```

2.6.3.6 出力ディレクトリ

これは、出力アセンブリが書き込まれるディレクトリです。アプリケーションは無条件に、このディレクトリにあるファイルを常に上書きします。

出力ディレクトリ

```
<!-- 出力先ディレクトリが必要です-->
<output>
<file dir="c:\work"/>
</output>
```

2.6.3.7 一時ディレクトリ

このセクションは省略可能であり、Dotfuscator の作業ディレクトリを指定します。指定しない場合、既定値であるシステムの一時ディレクトリが作業ディレクトリになります。アプリケーションは、作業ディレクトリを使用して入力アセンブリで ildasm および ilasm を実行します。逆アセンブルされた出力は、入力アセンブリに埋め込まれたすべてのリソースと共に、このディレクトリに格納されます。これらのファイルは、処理の終了後、自動的に削除されます。

一時ディレクトリ

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<!-- 一時ディレクトリの指定は任意です -->
<!-- 指定しないと、既定でシステムの temp ディレクトリになります -->
<tempdir>
<file dir="c:\temp"/>
</tempdir>
```

2.6.3.8 難読化属性の機能割り当て

機能割り当ては、宣言による難読化のための機能です。Dotfuscator の「宣言による難読化」のサポートの詳細については、「カスタム属性を使用した宣言による難読化」を参照してください。このセクションでは、機能割り当てについて説明し、Dotfuscator が認識するネイティブな機能文字列を紹介します。

構成ファイル内の **<obfuscationattributemap>** 要素は、難読化属性の feature プロパティから取得した文字列を、Dotfuscator が認識する 1 つ以上の機能文字列に割り当てる場所です。

これらの割り当ては、XML 構成ファイル内で次のように指定します。

難読化属性の機能割り当て

```
<obfuscationattributemap>
  <feature name="testmode">renaming, controlflow</feature>
</obfuscationattributemap>
```

2.6.3.9 名前の変更セクション

名前の変更セクションでは、ユーザーは入力割り当てファイルおよび出力割り当てファイルの場所や、名前の変更から項目を除外するための詳細な規則など、名前の変更に固有のオプションを指定できます。

名前の変更セクションは省略可能です。このセクションが存在しない場合は、以下の既定の処理が適用されます。

- 既定の名前の変更が行われます（名前空間は削除されます）。
- **loweralpha** の名前の変更規則を使って新しい名前が選択されます。
- 割り当てファイルの読み取りまたは書き込みは行われません。
- アプリケーションの種類に従って行われる対象除外以上の除外は行われません。

「識別子の名前の変更」セクションでは、名前の変更オプション、割り当てファイル、およびカスタム対象除外規則について詳しく説明します。以下のセクションでは、それぞれの概要を説明します。

2.6.3.9.1 名前の変更規則

Dotfuscator では、難読化処理時に識別子の名前を生成するためのアルゴリズムがいくつか定義されており、その中から選んで使用できます。

- **Lower Alpha**。これは、Dotfuscator のすべてのバージョンにおける名前の変更規則の既定値です。**{a,b,c,...}**

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- **Upper Alpha**。この規則では、大文字の英文字を使用します。`{A,B,C,...}`
- **Numeric**。この規則では、数字だけを使用します。`{0,1,2,...}`
- **Unprintable**。この規則では、Unicode の上位部分にある印字不能文字群を使用します。

名前の変更規則は **`<renaming>`** 要素の属性として指定します。使用できる値は、**`loweralpha`**、**`upperalpha`**、**`numeric`**、またはr **`unprintable`** です。

Renaming Scheme

```
<renaming scheme="unprintable">
...
</renaming>
```

2.6.3.9.2 名前の変更オプション

Dotfuscator には、名前の変更アルゴリズムが**名前空間**を処理する方法を決定するためのオプションがいくつか用意されています。すなわち、"**`keepnamespace`**" および "**`keephierarchy`**" です。これらの詳細については「識別子の名前の変更」で説明します。

`disable` オプションは、主に利便性とトラブルシューティングのために用意されています。このオプションを設定すると、名前の変更セクションの残りの部分の内容に関係なく、Dotfuscator は名前の変更をまったく行いません。

名前の変更オプション - Keepnamespace

```
<renaming>
<!-- 名前空間をそのまま保持しますが、型名は変更します -->
<option>keepnamespace</option>

<!-- 名前空間の階層は保持しますが、名前空間の名前は -->
<!-- 変更します -->
<option>keephierarchy</option>

<!-- セクションの残り部分を無視し、名前の変更処理をスキップします -->
<option>disable</option>
...
</renaming>
```

Dotfuscator では、難読化された型名に対し、既定値の文字列またはユーザー指定の文字列をプレフィックスとして付加するように指定できます。この機能を使用する場合は、**`prefix`** オプションを指定します。詳細については、「名前の変更プレフィックス」を参照してください。

名前の変更プレフィックス機能の有効化

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<renaming>
   <!-- 次の設定により、名前変更のプレフィックス機能が有効になります -->
   <option>prefix</option>
...
</renaming>
```

Dotfuscator では、戻り値の型を使用する拡張オーバーロード誘導も使用できます。この機能を有効にするためのオプションは **enhancedOI** です。このオプションについては、「オーバーロード誘導によるメソッドの名前の変更」で詳しく説明します。

拡張オーバーロード誘導の適用

```
<renaming> <!-- 拡張オーバーロード誘導を適用します -->
 <option>enhancedOI</option>
 ...
</renaming>
```

拡張オーバーロード誘導は、既定では、シリアル化可能とマークされているクラスには適用されません。シリアル化可能な型も含め、すべての型に拡張オーバーロード誘導を適用したい場合は、**enhancedOIOnSerializables** オプションを使用します。

拡張オーバーロード誘導シリアル化オプションの適用

```
<renaming>

<!-- シリアル化可能な型にも拡張オーバーロード誘導を適用します -->
<option>enhancedOIOnSerializables</option>
...
</renaming>
```

XML シリアライザーと互換性のある方法で型やメンバーの名前を変更するように、名前の変更アルゴリズムを変更することができます。

名前の変更アルゴリズムを変更して XML シリアライザーとの互換性を保つ

```
<renaming>

<!-- XML シリアル化の互換性 -->
<option>xmlserialization</option>
...
</renaming>
```

詳細については、「XML シリアル化と名前の変更」を参照してください。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

名前の変更オプションの **<explicitoverrides>** を使用すると、Dotfuscator は明示的な（構文上でない）メソッドのオーバーライドを導入することによって、より多くのメソッドの名前を変更できるようになります。つまり、オーバーライドされるメソッドが、オーバーライドするメソッドとは異なる名前を持つことができます。たとえば、メソッドで Object.ToString() をオーバーライドする場合、通常、Dotfuscator はオーバーライドの関係を壊さないで名前を変更することはできません。これは、一般的に入力アセンブリには Object クラスがないため、ToString() の名前は変更されないからです。この設定を有効にすると、Dotfuscator はオーバーライドするメソッドの名前を変更でき、そのメソッドが Object.ToString() をオーバーライドするように意図して作られていることを CLR に指示するメタデータを導入できます。詳細については、「名前の変更時、明示的なメソッドのオーバーライドを導入する」を参照してください。

明示的なメソッドのオーバーライドを導入する

```
<renaming>
<!-- オーバーライドされるメソッドが、オーバーライドするメソッドとは
        異なる名前を持つことができます -->
   <option>explicitoverrides</option>
</renaming>
```

オーバーロードされたメソッドが同じ名前になるようにしたい場合は、名前の変更オプションの **<explicitoverrides>** を使用しないでください。

メモ: 名前の変更オプションは、大文字と小文字を区別しません。

2.6.3.9.3 名前の変更の対象除外リスト

このセクションは、入力アセンブリの名前変更を動的に微調整する方法を提供します。このセクションには、実行時に適用される "対象除外規則" のリストを含めることができます。規則で特定のクラス、メソッド、フィールド、プロパティ、またはイベントを選択した場合、その項目の名前は変更されません。

- これらの規則は、library オプションなどのグローバル オプションによって暗黙に適用される規則と併せて適用されます。
- 各規則は論理的に **OR** で結合されるため、1 つでも規則にマッチした項目の名前は変更されません。
- 対象除外リストは、型、メソッド、フィールド、プロパティ、イベント、アセンブリ、モジュール、または名前空間による名前の除外をサポートしています。
- 各種の規則については、「識別子の名前の変更」で詳しく説明します。

2.6.3.9.4 名前の変更の参照規則

参照規則を使用すると、外部ファイルから規則をインポートして、構成間で共有できるようになります。Dotfuscator の組み込みの名前の変更規則は、これを使用して **%ProgramData%\PreEmptive Solutions\Common\dotfuscatorReferenceRule_v1.1.xml** から規則をインポートしています。規則は **rulekey** 属性で参照されます。値には、参照される規則に定義した GUID を指定します。

参照規則リスト

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<referencerulelist>
   <referencerule rulekey="{0D471A86-E98F-4493-849B-85BD4CC884A1}"/>
   <referencerule rulekey="{C9D9BF84-4F0D-4e9f-B3EC-3038235AE741}"/>
</referencerulelist>
```

2.6.3.9.5 出力割り当てファイル

この機能は、特定の実行中に Dotfuscator が使用したすべての名前の変更割り当てのログを作成します。これは、統計セクションも提供します。

このオプションを指定することで、Dotfuscator の名前の変更機能に対し、名前の変更がどのように行われたかを追跡記録するよう指示できます。その結果、ユーザーは名前の変更状況をすぐに調査できるほか、今後 Dotfuscator を実行する際に入力として使用することができます。このオプションで作成されるファイルは、その後増分入力ファイル オプションで使用されます。

このファイルを不注意で失ってしまうと、将来のアプリケーションの増分更新の機会を失う可能性があります。そのため、このファイルを適切にバックアップしておくことが非常に重要になります。このような理由から、Dotfuscator は（同じ名前の）既存のファイルが検出された場合には、自動的にそのファイルの名前を変更してから、新しい割り当てファイルで既存の割り当てファイルを上書きするようになっています。

Dotfuscator に既存の割り当てファイルの名前を自動的に変更させないように指示するには、構成ファイルに **`overwrite="true"`** という属性を設定します。

割り当てファイルの形式については、「識別子の名前の変更」で説明します。

属性を上書きに設定する

```
<renaming>
...
<mapping>
  <mapoutput overwrite="true">
   <file dir="c:\work" name="testout.xml"/>
  </mapoutput>
</mapping>
</renaming>
```

2.6.3.9.6 HTML 形式の名前の変更レポート

出力割り当てファイルは、本来は解析向きの XML ドキュメントとしてフォーマットされます。名前の変更レポートを判読可能にするために、出力割り当てファイルを HTML 形式のドキュメントに変換するよう指示することができます。Dotfuscator は、あらかじめ定義されている XSL ドキュメントを出力割り当てファイルに適用することにより、変換処理を実施します。既定の HTML レポートが好みでない場合は、任意で、変換に使用する XML ドキュメントに独自のものを指定できます。出力レポートは、XML 形式の割り当てファイルと同じディレクトリに置かれます。ファイル名は XML ファイルと同じで、拡張子が **`.xml`** ではなく **`.html`** になります。

HTML 形式の名前の変更レポート

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<renaming>
...
<mapping>
  <mapoutput overwrite="true">
   <file dir="c:\work" name="testout.xml"/>
   <transform>
      <!-- 独自の XSL ファイルの指定は任意です -->
      <file dir="c:\mytransforms" name="map.xsl"/>
   </transform>
  </mapoutput>
</mapping>
</renaming>
```

2.6.3.9.7 入力割り当てファイル

Dotfuscator では、入力割り当てファイルによって、前回の Dotfuscator の実行で作成された名前をインポートすることができます（「増分難読化」として知られている処理です）。Dotfuscator はクラス、メソッド、およびフィールドの名前を、入力割り当てファイルに示されている名前に変更するように最善の努力をします。

<mapinput> 要素に入力割り当てファイルを指定できます。この要素には任意の **obfuscatereferences** 属性もありますが、記述しなければ既定値である "**true**" になります。この属性は、入力割り当てファイルに含まれているが、入力アセンブリのセット内には定義されていない名前を Dotfuscator でどのように処理するかを制御します。true の場合、現在の入力アセンブリのセット内にこのような名前への参照があれば、その名前は変更されます。

入力割り当てファイル

```
<renaming>
...
<mapping>
  <mapinput obfuscatereferences="true">
   <file dir="c:\work" name="testin.xml"/>
  </mapinput>
</mapping>
</renaming>
```

2.6.3.10 制御フローの難読化セクション

制御フロー セクションでは、制御フローの難読化から項目を除外するための詳細な規則など、制御フローの難読化に固有のオプションを指定できます。

制御フロー セクションは省略可能です。このセクションが存在しない場合、制御フローの難読化は無効になります。

2.6.3.10.1 制御フローの難読化レベル

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

制御フローの難読化のレベルは、"**low**"(低)、"**medium**"(中)、"**high**"(高)という 3 つの値のいずれかに設定されます。これらのレベルは、Dotfuscator の制御フローの難読化アルゴリズムの強力さに対応します。一般に、レベルが高いほど難読化はより強力になりますが、それと引き換えにコード サイズは増加し、パフォーマンスが低下します。これは、制御フローの難読化が強力であればあるほど、より多くの分岐命令をコードに追加することになるからです。

制御フローの難読化のレベルは、難読化されるすべてのメソッドに対しグローバルに適用されます。

2.6.3.10.2 制御フローの難読化オプション

"**disable**" オプションは、主に利便性とトラブルシューティングのために用意されています。このオプションを設定すると、制御フロー セクションの残りの部分の内容に関係なく、Dotfuscator は制御フローの難読化をまったく行いません。

制御フローの難読化オプション

```
<controlflow level="high">
<!-- 本セクションの残り部分を無視し、制御フローの難読化処理をスキップします -->
<option>disable</option>
...
</controlflow>
```

メモ: 制御フロー関連のオプションは、大文字と小文字を区別しません。

2.6.3.10.3 制御フローの対象除外リスト

このセクションは、入力アセンブリの制御フローの難読化を動的に微調整する方法を提供します。このセクションには、実行時に適用される "対象除外規則" のリストを含めることができます。規則で特定のクラスまたはメソッドを選択すると、その項目は制御フローの難読化の対象になりません。

メモ: library オプションは名前の変更処理の場合とは異なり、制御フローの難読化には影響しない点に注意してください。

各規則は論理的に **OR** で結合されるため、1 つでも規則にマッチした項目は制御フローの難読化の対象になりません。

対象除外リストは、型、メソッド、アセンブリ、モジュール、または名前空間によるモジュールの除外をサポートしています。

各種規則については、「制御フローの難読化」で詳しく説明します。

2.6.3.11 文字列の暗号化セクション

文字列の暗号化セクションでは、文字列の暗号化の対象になる型およびメソッドを指定するための詳細な規則など、文字列の暗号化に固有のオプションを指定できます。

文字列の暗号化セクションは省略可能です。このセクションが存在しない場合、文字列の暗号化機能は無効になります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.3.11.1 文字列の暗号化オプション

このオプションは、主として利便性とトラブルシューティングのために使用されます。このオプションを設定すると、文字列の暗号化セクションの残りの部分の内容に関係なく、Dotfuscator は文字列の暗号化をまったく行いません。

ユーザー文字列の暗号化オプション

```
<stringencrypt>
<!--本セクションの残り部分を無視し、文字列暗号化処理をスキップします-->
<option>disable</option>
...
</stringencrypt>
```

メモ: 文字列の暗号化オプションは、大文字と小文字を区別しません。

2.6.3.11.2 文字列の暗号化対象選択リスト

このセクションは、入力アセンブリに対する文字列の暗号化機能を動的に微調整する方法を提供します。このセクションには、実行時に適用される対象選択規則のリストを含めます。規則で特定のクラスまたはメソッドを選択すると、その項目は文字列の暗号化の対象になります。

メモ: 名前の変更処理とは異なり、library オプションは文字列の暗号化には影響しない点に注意してください。

各規則は論理的に **OR** で結合されるため、1 つでも規則にマッチした項目は文字列暗号化の対象になります。

対象選択リストは、型、メソッド、アセンブリ、モジュール、または名前空間によるメソッド選択をサポートしています。

各種規則については、「ユーザー文字列の暗号化」で詳しく説明します。

2.6.3.12 除去セクション(不要コードの除去)

<removal> セクションでは、不要コードの除去の対象になる型およびメンバーを指定するための詳細な規則など、除去機能に固有のオプションを指定できます。

<removal> セクションは省略可能です。このセクションが存在しない場合、除去機能は無効になり不要コードの除去は行われません。

2.6.3.12.1 除去無効オプション

このオプションは、主として利便性とトラブルシューティングのために使用されます。このオプションを設定すると、除去セクションの残りの部分の内容に関係なく、Dotfuscator は不要コードの除去処理をまったく行いません。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

除去無効オプション

```
<removal>
<!-- 本セクションの残り部分を無視し、除去 (不要コードの除去) 処理をスキップします -->
<option>disable</option>
...
</removal>
```

メモ: 除去オプションは、大文字と小文字を区別しません。

2.6.3.12.2 ConstOnly オプション

このオプションは定数のみ除去を有効にするために使用します。このモードでは、定数宣言のみが除去されます。未使用の型、メソッドおよびフィールドは出力アセンブリに反映されます。

定数のみ除去オプション

```
<removal>
<!-- 不要コードの完全な除去ではなく定数のみ除去を使用します -->
<option>constonly</option>
...
</removal>
```

メモ: 除去オプションは、大文字と小文字を区別しません。

2.6.3.12.3 除去トリガー リスト

不要コードの除去における **トリガー**とは、Dotfuscator がコードで使用されている型やメソッドおよびフィールドを判断するために行う静的依存関係の分析の開始点です。別の言い方をすると、トリガーとはアプリケーションあるいはライブラリの **エントリ ポイント**であると言えます。

与えられたトリガーは Dotfuscator によって分析され、そのアプリケーションまたはライブラリが機能するにはどのクラス、メソッドおよびフィールドが必要であるかを判断します。たとえば、トリガーが呼び出すすべてのメソッド、およびこれらのメソッドが呼び出すメソッドは、Dotfuscator で必要であると判断されます。つまり、Dotfuscator に対して特定の Main メソッドが必要であると通知した場合、その Main メソッドが呼び出すすべてのメソッドも必要であるということです。

なお、library グローバル オプションを使用する場合、トリガー リストは必要ありませんが、あっても受け付けられます。

トリガー リストは、library オプションなどのグローバル オプションが暗黙に適用されるトリガーと共に使用されます。

トリガー リストにトリガーを指定する方法は、構成ファイルの他の部分で除外および選択の対象にする要素を選択する方法と同じです。このリストには、実行時に適用される規則のリストを含めます。規則で特定のメソッドあるいはフィールドを選択すると、そのメソッドあるいはフィールドはトリガーになります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

各規則は論理的に OR で結合されるため、1 つでも規則にマッチした項目はトリガーになります。

トリガー リストは、型、メソッド、アセンブリ、モジュール、または名前空間によるフィールド、メソッド、プロパティ、およびイベントの指定をサポートしています。

各種規則については、「不要コードの除去」で詳しく説明します。

2.6.3.12.4 条件付き対象選択リスト

ある型が静的依存関係の分析によって検出できない場合（すなわち型が動的に読み込まれる場合）、その型は条件付きで対象として選択される必要があります。これは、その型自体は依存関係の分析対象になりますが、そのメンバーは依然として除去の対象になることを意味します。この機能の詳細については、対象トリガーと条件付き対象の理解を参照してください。

このセクションは、条件付きで対象として選択される型を動的に指定する方法を提供します。このセクションには、実行時に適用される対象選択規則のリストを含めます。規則で特定の型を選択すると、その項目は条件付きで対象として選択されます。

各規則は論理的に OR で結合されるため、1 つでも規則にマッチした項目は条件付きで対象として選択されます。

対象選択リストは、名前、アセンブリ、モジュール、または名前空間による型の選択をサポートしています。

各種規則については、「不要コードの除去」で詳しく説明します。

2.6.3.12.5 除去の参照規則

参照規則を使用すると、外部ファイルから規則をインポートして、構成間で共有できるようになります。Dotfuscator の組み込みの除去規則は、これを使用して **%ProgramData%\PreEmptive Solutions\Common\dotfuscatorReferenceRule_v1.4.xml** から規則をインポートしています。規則は **rulekey** 属性で参照されます。値には、参照される規則に定義した GUID を指定します。

参照規則

```
<referencerulelist>
   <referencerule rulekey="{0D458786-E99F-4593-849B-8512493884A1}"/>
   <referencerule rulekey="{C1159284-4F0D-4e9f-B3EC-303828419741}"/>
</referencerulelist>
```

2.6.3.12.6 除去レポート

除去レポートは、ある特定の実行中に Dotfuscator によって除去されたすべての要素の概要を提供します。これは、統計セクションも提供します。Dotfuscator は除去レポートを解析可能で変換が容易な XML 形式で書き込みます。名前変更割り当てファイルのように、Dotfuscator は既定の変換ファイルを持っており、このファイルを使用して判読可能な HTML 形式のレポートを生成できます。

Dotfuscator は同じ名前の既存の除去レポートが検出された場合には、自動的にそのレポートの名前を変更してから、新しいレポートで既存レポートを上書きするようになっています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Dotfuscator に既存の除去レポートの名前を自動的に変更させないように指示するには、構成ファイルに **overwrite="true"** という属性を設定します。

XML 形式の除去レポート ファイルについては、「不要コードの除去」で説明します。

除去レポート

```
<removal>
<removalreport overwrite="true">
  <file dir="c:\work" name="report.xml"/>
  <transform>
     <!-- 独自の XSL ファイルの指定は任意です -->
     <file dir="c:\mytransforms" name="removal.xsl"/>
  </transform>
</removalreport>
</removal>
```

2.6.3.13 リンク セクション

リンク セクションでは、アセンブリのリンクに固有のオプションを指定できます。リンク機能の詳細については、「アセンブリのリンク」を参照してください

リンク セクションは省略可能です。このセクションが存在しない場合、アセンブリのリンク機能は無効になります。

2.6.3.13.1 リンク無効オプション

このオプションは、主として利便性とトラブルシューティングのために使用されます。このオプションを設定すると、リンク セクションの残りの部分の内容に関係なく、Dotfuscator はリンク処理をまったく行いません。

リンク無効オプション

```
<linking>
<!-- 本セクションの残り部分を無視し、リンク処理をスキップします -->
<option>disable</option>
...
</linking>
```

メモ:リンク オプションは、大文字と小文字を区別しません。

2.6.3.13.2 リンク アセンブリ

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

<linkedassembly> 要素は、1 つまたは複数の入力アセンブリを特定の出力アセンブリにリンクすることを指定します。リンク セクションには複数の **<linkedassembly>** 要素を含むことができるため、このリンク機能を使って複数の出力アセンブリを作成できます。唯一、1 つの入力アセンブリを複数の出力アセンブリにリンクさせることはできないという制限があります。

<linkedassembly> 要素には子要素を含むことができ、名前変換ポリシーなどのリンク処理オプションや、プライマリ アセンブリ、入力アセンブリのリスト、および出力アセンブリの名前を指定できます。

オプション

現在指定できるオプションは "名前変換ポリシー" のみで、次のいずれかを指定できます。

- donotmangle（改変しない）
- manglesilently（暗黙の改変）
- mangleandwarn（改変して警告）

プライマリ アセンブリ

<primaryinput> 要素はプライマリ アセンブリを定義します。このアセンブリのマニフェスト情報を基に、出力アセンブリのマニフェストが作成されます。このアセンブリは、次で説明する **<assemblylist>** 要素にも記述する必要があります。

リンクするアセンブリ

リンクするアセンブリは、**<assemblylist>** 要素を使って列挙します。また、各アセンブリは入力アセンブリとしても列挙する必要があります。

出力アセンブリ

<outputassembly> 要素には出力アセンブリの名前と、任意でエントリ ポイント メソッドを指定できます。アセンブリは出力先ディレクトリに指定された名前で書き込まれます。

例

次の例の **<linkedassembly>** 要素は、入力アセンブリの **Driver.exe** と **LibraryC.dll** を **out.exe** という名前のアセンブリにリンクすることを指定しています。エントリ ポイント メソッドは **Driver** アセンブリ内の **Main** メソッドであると明示的に設定されています。

> **メモ**: エントリ ポイントを明示的に指定する必要があるのは、そのエントリ ポイントがあいまいな場合のみです。これはたとえば、出力アセンブリが .exe であったり、入力アセンブリに複数のエントリ ポイントがあったりする場合です。

リンク アセンブリ

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<linkedassembly>
  <option>donotmangle</option>
  <primaryinput>
    <assembly>
      <file dir="${configdir}" name="Driver.exe" />
    </assembly>
  </primaryinput>
  <assemblylist>
    <assembly>
      <file dir="${configdir}" name="Driver.exe" />
    </assembly>
    <assembly>
      <file dir="${configdir}" name="LibraryC.dll" />
    </assembly>
  </assemblylist>
  <outputassembly name="out.exe">
    <entrypoint>
      <type name="Driver.Form1">
        <method name="Main" signature="" />
      </type>
    </entrypoint>
  </outputassembly>
</linkedassembly>
```

2.6.3.14 PreMark セクション

PreMark セクションでは、ウォーターマークに固有のオプションを指定できます。アセンブリのウォーターマークの詳細については、「ウォーターマーク」を参照してください。

> **メモ**: PreMark セクションは省略可能です。このセクションが存在しない場合、ウォーターマーク機能は無効になります。

2.6.3.14.1 PreMark オプション

ウォーターマーカはいくつかの構成オプションをサポートしています。

usepassphrase オプションは、ウォーターマーク文字列を暗号化してから選択されているアセンブリに適用するよう、ウォーターマーカに指示します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

truncatestring オプションは、ウォーターマーク文字列が指定されたアセンブリに収まらない場合には文字列を切り捨てるように指示します。既定の動作では、ウォーターマーク文字列が大きすぎる場合にはビルドが中止されます。詳細については、「ウォーターマーク文字列の長さ」を参照してください。

disable オプションは、主に利便性とトラブルシューティングのために用意されています。このオプションを設定すると、PreMark セクションの残りの部分の内容に関係なく、Dotfuscator はウォーターマーク処理をまったく行いません。

PreMark オプション

```
<premark>
<!--本セクションの残り部分を無視し、ウォーターマーク処理をスキップします-->
<option>disable</option>
<!--ウォーターマーク文字列を暗号化します  -->
<option>usepassphrase</option>
<!--文字列が大きすぎる場合は、文字列を切り詰めて処理を続行します  -->
<option>truncatestring</option>
...
</premark>
```

Note: PreMark オプションは、大文字と小文字を区別しません。

2.6.3.14.2 PreMark の要素

premark セクションには子要素を含むことができ、ウォーターマークを設定する入力アセンブリや、ウォーターマーク文字列を暗号化する際に使用するパスフレーズ、使用する文字エンコード、およびウォーターマーク文字列自体を指定できます。

ウォーターマークを設定するアセンブリ

ウォーターマークを設定するアセンブリは、**<assemblylist>** 要素を使って列挙します。また、各アセンブリは入力アセンブリとしても列挙する必要があります。

<passphrase>

<passphrase> 要素では、ウォーターマーク文字列を暗号化する際に使用するパスフレーズを指定します。**usepassphrase** オプションを設定しパスフレーズを指定すれば、ウォーターマーク文字列は暗号化されます。

<encoding>

<encoding> 要素では、ウォーターマーク文字列をエンコードする際に使用する文字エンコード（つまり "文字コード"）を指定します。**<encoding>** 要素は **name** 属性を持ち、サポートされる文字コードを設定できます。文字コードとその名前については、「文字コード」で説明します。

<watermark>

<watermark> 要素には、選択したアセンブリに適用する文字列を指定します。文字列は、まず文字コードによりエンコードされ、その後任意で暗号化されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

例

次の例は、**MyApp.exe** にウォーターマークを適用するように PreMark を構成しています。ウォーターマーク文字列の **MY WATERMARK** は、まず **6bit-a** 文字コードを使ってエンコードされてから、パスフレーズ "**mommy**" を使って暗号化されます。

PreMark 要素

```
<premark>
  <option>usepassphrase</option>
  <option>truncatestring</option>
  <assemblylist>
    <assembly>
      <file dir="${configdir}" name="MyApp.exe" />
    </assembly>
  </assemblylist>
  <passphrase>mommy</passphrase>
  <encoding name="6bit-a" />
  <watermark>MY WATERMARK</watermark>
</premark>
```

2.6.3.15 署名セクション

署名セクションでは、厳密名の出力アセンブリに Dotfuscator で署名するかどうか、また署名する場合の方法を指定できます。アセンブリ署名の詳細については、「厳密名のアセンブリを Dotfuscator で処理する」を参照してください。

署名セクションは省略可能です。このセクションが存在しない場合、厳密名の入力アセンブリは Dotfuscator の処理後に再署名されず、Authenticode デジタル署名は適用されません。

<key> 要素の指定

署名するときに Dotfuscator が使用するキーまたはキー ペアを **<key>** 要素で指定できます。**<key>** 要素には、子要素の **<file>** または **<container>** のいずれかを含むことができます。**<file>** 要素は、キーまたはキー ペアを含んでいるファイルを参照します。A **<container>** 要素は "**name**" 属性を持ち、キー コンテナーの名前を指定できます。

<file> 要素

```
<key>
  <file dir="c:\temp" name="key.snk" />
</key>
```

<container> 要素

```
<key>
  <container name="foo"/>
</key>
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

<resign> 要素

Dotfuscator の処理前に既に署名されているアセンブリに再署名する場合は、**<resign>** 要素を使用します。使用するキーを指定するカスタム属性がアセンブリに記述されている場合、**<key>** 要素の指定は必要ありません。属性を無視したい場合は、**dontuseattributes** オプションを設定し、**<key>** 要素を指定します。アセンブリにキーを指定するカスタム属性がない場合は、**<key>** 要素を指定する必要があります。

次の例では、Dotfuscator にはキーを指定するカスタム属性をすべて無視するように通知し、代わりに手作業でキー ファイルを指定しています。

<resign> 要素

```
<signing>
  <resign>
    <option>dontuseattributes</option>
    <key>
      <file dir="c:\temp" name="key.snk" />
    </key>
  </resign>
  ...
</signing>
```

<delaysign> 要素

入力アセンブリが遅延署名されているとき、その署名プロセスを Dotfuscator で自動的に完了させる場合は、**<delaysign>** 要素と共に **<key>** 子要素を指定できます。

<delaysign> 要素

```
<signing>
  ...
  <delaysign>
    <key>
      <file dir="c:\temp" name="key.snk" />
    </key>
  </delaysign>
</signing>
```

2.6.3.16 デジタル署名セクション

デジタル署名セクションでは、出力アセンブリに対し Dotfuscator で Authenticode デジタル署名を行うかどうか、また署名を行う場合の方法を指定できます。

デジタル署名セクションは省略可能です。このセクションが存在しない場合、Authenticode 署名は適用されません。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

<pfx> 要素の指定

Authenticode 署名で使用するコード署名権限によって提供されている PKCS #12 ファイルを **<pfx>** 要素を用いて指定できます。A **<pfx>** 要素には、**<file>** 子要素があります。この要素にはコード署名証明書を含む PKCS #12(.pfx)ファイルの場所を指定します。また、**<pfx>** 要素は、証明書のロック解除に使用するパスワードを指定する **password** 属性もあります。

<pfx> 要素

```
<pfx password="secret123">
  <file dir="c:\temp" name="authenticode.pfx" />
</pfx>
```

<digitalsigning> 要素

出力アセンブリに Authenticode 署名を行うには、**<digitalsigning>** 要素を提供できます。子要素として **<pfx>** も使用します。

disable オプションは、主に利便性とトラブルシューティングのために用意されています。このオプションを設定すると、Authenticode 署名セクションの残りの部分の内容に関係なく、Dotfuscator は Authenticode 署名をまったく行いません。

<timestampurl> 子要素を使用すれば、Authenticode のタイムスタンプ サービスの URL を指定することができます。この URL は Dotfuscator の署名プロセス時にアクセスされ、そこから追加データが提供されます。その追加データを使用すれば、アセンブリの Authenticode 署名は、コード署名証明書の有効期限が切れた後も有効なままになります。この要素は省略可能です。ただし、省略した場合にはこの追加データは含まれないので、コード署名証明書の有効期限が切れるとアセンブリの Authenticode 署名は無効になります。

<digitalsigning> 要素

```
<digitalsigning>
  <!--本セクションの残り部分を無視し、Authenticode 署名をスキップします-->
  <option>disable</option>
  <!--Authenticode 署名に使用する証明書を指定します-->
  <pfx password="secret123">
    <file dir="c:\temp" name="authenticode.pfx" />
  </pfx>
  <!--任意でタイムスタンプ サービスへの URL を指定します-->
  <timestampurl>http://timestamp.comodoca.com/authenticode</timestampurl>
</digitalsigning>
```

2.6.3.17 EventList セクション

eventlist (イベント リスト) セクションでは、ビルド前およびビルド後のイベントを指定できます。Dotfuscator のビルド イベントの詳細については、「ビルド イベント」を参照してください。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

eventlist セクションは省略可能です。

<event> 要素

イベントは本質的には 1 つのプログラムであり、Dotfuscator のビルド工程におけるある特定の時点に Dotfuscator が実行するものです。プログラム名、作業ディレクトリ、およびコマンド ライン引数を指定できます。さらに、イベントは、**<option>** 要素を使って追加の構成設定をサポートすることもできます。

<event> 要素は **type** 属性を持ちます。現在、Dotfuscator が認識できるタイプは次の 2 つです。**prebuild** と **postbuild**。実行するプログラムは **<program>** 要素を使って指定します。

イベント <program> 要素

イベントの発生時に実行するプログラムは、**<program>** 要素で指定します。この要素は、プログラムとプログラムの場所を指定する **<file>** 要素だけでなく、コマンド ライン引数や作業ディレクトリを指定する **<environment>** 要素も含んでいます。

commandline 属性には、ファイルやディレクトリと同様にプロパティマクロを含むことができます。

Property Macros

```
<program>
  <file dir="c\temp" name="copyfiles.bat" />
  <environment commandline="${myproperty}" workingdir="c:\temp" />
</program>
```

ビルド前のイベント オプション

ビルド前のイベントは 1 つのオプションをサポートしています。オプションは、ビルド前イベントの **<event>** 要素の中に入れ子で **<option>** 要素を記述して設定できます。

オプション	
haltonfail	ビルド イベント プログラムがゼロ以外のエラー コードを返した場合、Dotfuscator のビルドを中止します。

ビルド後のイベント オプション

ビルド後のイベントはいくつかのオプションをサポートしています。オプションは、ビルド後イベントの **<event>** 要素の中に入れ子で **<option>** 要素を記述して設定できます。

オプション	
haltonfail	ビルド イベント プログラムがゼロ以外のエラー コードを返した場合、Dotfuscator のビルドを中止します。
runoneachmodule	出力モジュールごとに 1 回、ビルド後のイベントを実行します。
always	Dotfuscator のビルドが成功したか失敗したかにかかわらず、常にビルド後のイベントを実行しま

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

	す。
buildfails	Dotfuscator のビルドが失敗した場合にのみ、ビルド後のイベントを実行します。
buildsuccessful	Dotfuscator のビルドが成功した場合にのみ、ビルド後のイベントを実行します。

例

以下に、XML 構成ファイル内でビルド前およびビルド後のイベントを設定する箇所を示します。ビルド前のイベントでは、**c:\temp\copyfiles.bat** プログラムを引数なしで実行します。ビルドが成功したら、ビルド後のイベントでは、出力アセンブリごとに **PEVerify** プログラムを実行します。出力アセンブリ名はプロパティとして **PEVerify** に渡されていることに注目してください。

Example XML Configuration File Fragment

```
  <eventlist>
    <event type="prebuild">
      <program>
        <file dir="c\temp" name="copyfiles.bat" />
        <environment commandline="" workingdir="c:\temp" />
      </program>
    </event>
    <event type="postbuild">
      <option>runoneachmodule</option>
      <option>haltonfail</option>
      <program>
        <file dir="C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin"
name="PEVerify.exe" />
        <environment commandline="${dotf.current.out.module}"
                     workingdir="${dotf.destination}" />
      </program>
    </event>
  </eventlist>
```

2.6.3.18 Runtime Intelligence セクション

Runtime Intelligence セクションでは、入力アセンブリのインストルメンテーション属性の処理方法を制御するオプションを指定できます。インストルメンテーションと PreEmptive Analytics の詳細につては、「インストルメンテーション（改ざん、Shelf Life、例外、分析）」を参照してください。

Runtime Intelligence セクションは省略可能です。

sos 要素は **mergeruntime** 属性を定義します。この属性には **true**（既定値）または **false** を指定できます。**true** の場合、Dotfuscator は Runtime Intelligence ライブラリを入力アセンブリの 1 つにマージします。**false** の場合、Dotfuscator はランタイム ライブラリ DLL をほかの入力アセンブリと一緒に出力先ディレクトリへ書き込みます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Runtime Intelligence には、[**有効化**]、[**分析メッセージの送信**]、[**改ざんメッセージの送信**]、および[**Shelf Life メッセージの送信**]という 4 つの追加オプションがあります。

sendanalytics オプションは、アプリケーション分析のスタートアップ、シャットダウン、および機能のメッセージを Runtime Intelligence Service へ送信するコードを、マークされたアセンブリに追加するよう Dotfuscator に指示します。分析しないで改ざんメッセージを送信したい場合には、このオプションの設定を解除します。

dontsendtamper オプションは、改ざんが検出されても改ざん通知を送信しないよう、Dotfuscator に指示します。これは、改ざんを検出してアプリケーションにローカルで対応させたいが、アプリケーションから Runtime Intelligence Service へメッセージを送信させたくない場合に有用です。

sendshelflife オプションは、RIS Shelf Life の状態メッセージを送信するロジックを追加するコードを、マークされたアセンブリに追加するよう Dotfuscator に指示します。これには、警告、期限切れ、および Sign of Life メッセージが含まれます。

disable オプションは、主に利便性とトラブルシューティングのために用意されています。このオプションを設定すると、入力アセンブリ内にどのような属性があろうと、sos セクション内にほかのオプションがあろうと関係なく、Dotfuscator はこの属性の処理をまったく行いません。

Runtime Intelligence の処理

```
  <sos mergeruntime="true">
    <!-- Runtime Intelligence の処理を無効にします -->
    <option>disable</option>
    <!-- 起動、シャットダウン、および機能のメッセージを送信します -->
<option>sendanalytics</option>
    <!-- 改ざんメッセージを送信しません -->
    <option>dontsendtamper</option>
    <!-- Shelf Life メッセージを送信します -->
<option>sendshelflife</option>
  </sos>
```

2.6.3.19 拡張属性セクション

Dotfuscator では、アプリケーションのソース コードを変更しなくても、メソッドやアセンブリに拡張属性を付けることができます。拡張属性は、コード内のサポートされている既存のカスタム属性を変更したり、サポートされている属性の新しいインスタンスとして動作することができます。アプリケーションの処理および変換を行う際、Dotfuscator は拡張属性をカスタム属性と同等のものとして扱います。

<codetransformlist> 要素を使用すると、属性がコードに埋め込まれているものであっても、拡張属性を使って、それをサポートされている特定のコード変換に割り当てることができます。これは属性のオーバーロードをサポートしており、同じ属性のセットで複数の変換処理(アプリケーション分析など)を作動させることができます。

ほとんどの拡張属性はメソッド レベルで設定できます。メソッドを識別するには、定義している型、名前、および署名を指定する必要があります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

サポートされている属性の引数は、**<propertylist>** 要素を使用して指定できます。

カスタム属性リファレンスに一覧表示されているすべての属性が、サポートされる属性です。

サポートされている属性

```
<extattributes>
<extattribute name="PreEmptive.Attributes.FeatureAttribute">
  <codetransformlist>
  <codetransform name="analytics"/>
  </codetransformlist>
  <type name = "MyApplicaton.MainForm">
   <method name="Main" signature="string[]" />
  </type>
  <propertylist>
   <property name="Name" value="Execute"/>
   <property name="ActivationStatusSinkElement" value="field"/>
   <property name="ActivationStatusSinkName" value="activated"/>
  </propertylist>
</extattribute>
</extattributes>
```

拡張ランタイム属性（**ApplicationAttribute**、**BinaryAttribute**、**BusinessAttribute** など）は、アセンブリ レベルで設定できます。アセンブリを識別するには、その名前を assembly 要素に指定する必要があります。

拡張ランタイム属性

```
<extattributes>
  <extattribute name="PreEmptive.Attributes.ApplicationAttribute">
    <codetransformlist>
      <codetransform name="sosruntime" />
    </codetransformlist>
    <assembly>
      <file dir="${configdir}" name="MyApp.exe" />
    </assembly>
    <propertylist>
      <property name="Version" value="" />
      <property name="Name" value="" />
      <property name="ApplicationType" value="" />
      <property name="Guid" value="00000000-0000-0000-0000-000000000000" />
    </propertylist>
  </extattribute>
</extattributes>
```

2.6.3.20 SmartObfuscation セクション

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

スマート難読化により、Dotfuscator はアプリケーションの種類に特有の規則に基づいて、名前の変更や除去を実行できない要素を自動検出できるようになります。スマート難読化は既定で有効に設定されており、ほとんどの場合は有効にしておく必要があります。ユーザーが積極的な難読化を行ってもアプリケーションを損なわないと確信している場合には、このセクションのオプションを使用して無効に設定することができます。

スマート難読化にはレポート機能が用意されており、このセクションでレポートの詳細を構成できます。詳細属性に使用できる値は、**すべて**、**警告のみ**、および**なし**です。既定値は `all` です。

スマート難読化レポートは、任意でディスクに書き出すことができます。Dotfuscator は同じ名前の既存のスマート難読化レポートが検出された場合には、自動的にそのレポートの名前を変更してから、新しいレポートで既存レポートを上書きします。Dotfuscator に既存のスマート難読化レポートの名前を自動的に変更させないように指示するには、構成ファイルに `overwrite="true"` という属性が必要になります。

XML 形式のスマート難読化レポートについては、「スマート難読化」セクションで説明しています。

SmartObfuscation

```
<smartobfuscation>
  <!-- 本セクションの残り部分を無視し、スマート難読化処理をスキップします -->
  <option>disable</option>
  <smartobfuscationreport verbosity="all" overwrite="true">
    <!-- 出力レポート ファイルの指定は任意です -->
    <file dir="c:\myproject" name="smartobfuscation.xml"/>
  </smartobfuscationreport>
</smartobfuscation>
```

2.6.3.21 XML 構成ファイルに関する注意

Dotfuscator は、構成ファイルおよび割り当てファイルに XML 形式のドキュメントを使用します。これらのドキュメントが読み込まれると、"`doctype`" で指定される**文書型定義(DTD)**によって検証されます。Dotfuscator は検証を実行するために、関連する **DTD** にアクセスできる必要があります。

Dotfuscator は、**DTD** を検索するために以下の手順を実行します。

1. **DTD URI** がローカル ファイルを指定する場合は、Dotfuscator は指示された場所でそのファイルを検索します。見つからない場合は、エラーが発生します。
2. **DTD の URI** がWeb リソースを指定する場合は、Dotfuscator は最初に **URI** で指定された名前を持つファイルが Dotfuscator のキャッシュ内にあるかどうかを調べます。Dotfuscator のキャッシュは、**%ProgramData%\PreEmptive Solutions\Common** ディレクトリにあります。
3. 見つからない場合、Dotfuscator は **DTD** を取得するために **URI** にアクセスします。見つかった場合、Dotfuscator は次回の要求がネットワークにアクセスする必要がないように、インストール ディレクトリに **DTD** をキャッシュします。**DTD** が見つからない場合、または Dotfuscator がネットワークから DTD を取得できない場合は、エラーが発生します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.3.22 カスタム規則リファレンス

Dotfuscator では、アプリケーションに対する難読化規則をカスタマイズすることができます。対象選択規則および対象除外規則は、名前の変更、制御フローの難読化、文字列の暗号化および入力アセンブリの除去について動的に微調整する方法を提供します。これらの規則は、library オプションなどのグローバル オプションによって暗黙に適用される規則と併せて適用され、論理的に **OR** で結合されます。

このセクションの内容

対象除外規則

対象選択規則

2.6.3.22.1 対象除外規則

対象除外リスト セクションは、入力アセンブリの名前の変更および制御フローの難読化を動的に微調整する方法を提供します。ユーザーは、実行時に適用される規則のリストを指定します。規則で特定のクラス、メソッド、またはフィールドを選択した場合、その項目の名前は変更されません。また、その項目は制御フローの難読化の対象から除外されます。

これらの規則は、library オプションなどのグローバル オプションによって暗黙に適用される規則と併せて適用されます。

規則は論理的に **OR** で結合されます。

名前空間、型、メソッド、またはフィールドの選択には、正規表現（RE）を使用できます。これを実行するには、省略可能な **regex** 属性を使用します。**regex** の既定値は false です。**regex** が true の場合、**name** 属性の値は正規表現と解釈されます。regex が false の場合、name は文字どおりに解釈されます。正規表現は特定の文字（ピリオドなど）に特別な意味を割り当てているので、regex の値は重要です。以下に、簡単な正規表現の例を示します。

以下に、簡単な正規表現の例を示します。	
`.*`	すべてと一致
`MyLibrar.`	`Matches MyLibrary`、`MyLibrari` などと一致
`My[\.]Test[\.]I.*`	`My.Test.Int1`、`My.Test.Internal` などと一致
`Get.*`	`GetInt`、`GetValue` などと一致
`Get*`	`Ge`、`Get`、`Gett`、`Gettt` などと一致

正規表現の構文の詳細については、.NET Framework のドキュメントを参照してください。

2.6.3.22.1.1 名前空間の対象除外

このオプションは、指定した名前空間内のすべての型とそのメンバーを対象から除外します。名前空間の指定には正規表現を使用できます。

正規表現

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<namespace name="My.Excluded.Namespace"/>
```

2.6.3.22.1.2 型の対象除外

このオプションは、名前または属性指定子によって型を対象から除外します。型名の指定には正規表現を使用できます。型名は完全修飾名である必要があります。内側の（入れ子になっている）クラスは、外側と内側のクラスの間に '**/**' を区切り文字として使用して指定します。たとえば、次のようになります。

Inner (Nested) Classes

```
<type name="Library.Class1/NestedClass"/>
```

属性指定子の選択や選択解除には、**speclist** 属性を使用します。**speclist** 属性は、型の有効な属性指定子をカンマで区切って指定します。利用可能な値は以下のとおりです。

Legal Values

```
abstract
interface
nestedassembly
nestedfamily
nestedfamorassem
nestedprivate
nestedpublic
notpublic
public
sealed
serializable
enum
```

属性指定子の前に '**-**' を付けると、規則の意味が反転します（つまり、指定した属性を持たないすべてのクラスを対象から除外します）。'**+**' を指定することもできますが、必要ありません。このリストに暗黙に適用される規則は、論理的に **AND** で結合されます（つまり、対象から除外する型のセットは、各規則に一致するすべての型の共通部分となります）。たとえば、次の規則は、public 型かつ sealed 型のものを対象から除外します。

Public 型かつ Sealed 型を対象から除外する規則

```
<type name=".*" speclist="+public,+sealed" regex="true"/>
```

<type> 要素を使用して型を選択し、選択した型内からフィールド、メソッド、プロパティ、およびイベントを除外する規則を指定することもできます。これにより、メンバーを除外しても、メンバーが所有している型は除外しないでおくことができます。これを実行するには、省略可能な **excludetype** 属性を使用します。指定がない場合、既定値は **true** であり、その型名が名前の変更または制御フローの難読化の対象から除外されます。

フィールド、メソッド、プロパティ、およびイベントの対象除外規則の指定

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<type name="MyCo.Test.MyOtherTest" excludetype="false">
<!-- ここで指定したメソッドとフィールドは除外されます -->
...
</type>
```

型の規則がプロパティやイベントの規則を含んでいない場合、対象から除外される型のすべてのプロパティ名およびイベント名は保持されます。型の規則がプロパティの規則を 1 つ以上含んでいる場合は、それらのプロパティ名のみが保持され、それ以外のプロパティ名はすべて削除されます。型の規則がイベントの規則を 1 つ以上含んでいる場合は、それらのイベント名のみが保持され、それ以外のイベント名はすべて削除されます。

> **注意:型が対象から**除外されておらず、ライブラリ オプションが設定されていない場合、Dotfuscator はプロパティ名とイベント名を削除します。

名前の変更にのみ適用:

applytoderivedtypes 属性を指定することによって、継承階層全体に型の規則を適用することができます。この属性の値を **true** に設定すると、型の規則および、それに含まれるすべてのメソッド、フィールド、プロパティ、イベント、カスタム属性、またはスーパータイプの規則が、選択されている型とそれから派生するすべての型に適用されます。指定がない場合、既定値は **false** であり、指定した型にのみ型の規則が適用されます。

2.6.3.22.1.3 メソッドの対象除外

メソッドの除外は、まず **<type>** 要素を使用して型を選択し、次に対象から除外するメソッドを選択する規則を指定することによって行われます。メソッドは、名前や属性指定子によって対象から除外できるほか、署名によっても除外できます。 利用可能な属性指定子は以下のとおりです。

利用可能な属性指定子は以下のとおりです。

```
abstract
assembly
family
familyorassembly
final
private
public
static
virtual
```

属性指定子が明示的に設定されていない場合、**speclist** 属性は選択条件として使用されません。次の例では、"Set" で始まるすべてのパブリック インスタンス メソッドが選択されます。

次の例では、"Set" で始まるすべてのパブリック インスタンス メソッドが選択されます。

```
<method regex="true" name="Set.*" speclist="+public,-static"/>
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

メソッドの署名は、**signature** 属性を使用して指定します。署名は、メソッドの戻り値の型とパラメーターの型を特定します。

戻り値の型とパラメーターの型

```
signature=""  <!-- empty signature -->

signature="string(int,MyClass,MyClass[])"
```

署名が明示的に設定されていない場合、メソッドの署名は選択条件として使用されません。

次の例では、署名によってメソッドが選択されます。

署名によるメソッドの選択

```
<method name="DoIt" signature="string(int, System.Console,
System.Collection.ICollection, float[])"/>
```

Module:mod_name という名前の特別な型セレクターを使用すると、グローバル メソッドを指定できます。**mod_name** には、グローバル メソッドを含んでいるモジュールの名前を指定します。

2.6.3.22.1.4 フィールドの対象除外

フィールドの除外は名前の変更でのみ有効です。フィールドの除外は、まず **<type>** 要素を使用して型を選択し、次に対象から除外するフィールドを選択する規則を指定することによって行われます。フィールドもまた、名前および属性指定子によって対象から除外できます。利用可能な属性指定子は以下のとおりです。

利用可能な属性指定子は以下のとおりです。

```
public
private
static
assembly
family
familyandassembly
familyorassembly
notserialized
```

属性指定子が明示的に設定されていない場合、フィールドの属性は選択条件として使用されません。次の例では、"**ENUM_**" で始まるすべての静的フィールドが選択されます。

"ENUM_" で始まるすべての静的フィールド:

```
<field regex="true" name="ENUM_.*" speclist="+static"/>
```

フィールドの署名は、**signature** 属性を使用して指定します。署名はフィールドの型を指定します。

署名はフィールドの型を指定します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
signature=""  <!-- 空の署名 -->

signature="int"
```

署名が明示的に設定されていない場合、フィールドの型は選択条件として使用されません。

Module:mod_name という名前の特別な型セレクターを使用すると、グローバル フィールドを指定できます。**mod_name** には、グローバル フィールドを含んでいるモジュールの名前を指定します。

2.6.3.22.1.5 プロパティの対象除外

プロパティの除外は名前の変更でのみ有効です。プロパティの規則は、型の規則によって限定されるので、規則の編集ビューでは型ノードの子として表示されます。プロパティの規則は、(親の型の規則の条件を満たすすべての型のプロパティのうちで)条件を満たすすべてのプロパティを選択します。サポートされる選択条件は、プロパティ名およびプロパティの属性です。利用可能な属性指定子は以下のとおりです。

利用可能な属性指定子は以下のとおりです。

```
public
private
static
assembly
family
familyandassembly
familyorassembly
```

属性指定子が明示的に設定されていない場合、プロパティの属性は選択条件として使用されません。次の例では、"Sample" で始まるすべてのプロパティが選択されます。

次の例では、"Sample" で始まるすべてのプロパティが選択されます。

```
<propertymember regex="true" name="Sample.*"/>
```

プロパティの署名は、**signature** 属性を使用して指定します。署名はプロパティの型を指定します。

署名はプロパティの型を指定します。

```
signature=""  <!-- 空の署名 -->

signature="int"
```

署名が明示的に設定されていない場合、プロパティの型は選択条件として使用されません。

Module:mod_name という名前の特別な型セレクターを使用すると、グローバル プロパティを指定できます。**mod_name** には、グローバル プロパティを含んでいるモジュールの名前を指定します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.3.22.1.6 イベントの対象除外

イベントの除外は名前の変更でのみ有効です。イベントの規則は、型の規則によって限定されるので、規則の編集ビューでは型ノードの子として表示されます。イベントの規則は、(親の型の規則の条件を満たすすべての型のフィールドのうちで)条件を満たすすべてのイベントを選択します。サポートされる選択条件は、イベント名およびイベントの属性です。利用可能な属性指定子は以下のとおりです。

利用可能な属性指定子は以下のとおりです。

```
public
private
static
assembly
family
familyandassembly
familyorassembly
```

属性指定子が明示的に設定されていない場合、イベントの属性は選択条件として使用されません。次の例では、"On" で始まるすべてのイベントが選択されます。

次の例では、"On" で始まるすべてのイベントが選択されます。

```
<eventmember regex="true" name="On.*"/>
```

Module:mod_name という名前の特別な型セレクターを使用すると、グローバル イベントを指定できます。**mod_name** には、グローバル イベントを含んでいるモジュールの名前を指定します。

2.6.3.22.1.7 カスタム属性による対象除外

型、メソッド、フィールド、およびプロパティは、カスタム属性を基に選択的に除外できます。カスタム属性の規則では、項目(型、メソッド、フィールド、プロパティ)の追加情報であるカスタム属性の名前を基に項目を選択します。型、メソッド、フィールド、プロパティを選択する規則の中に、カスタム属性の規則を 1 つ以上入れ子にすることができます。

型、メソッド、フィールド、プロパティの規則には、カスタム属性の規則を複数関連付けることができます。この場合、カスタム属性の規則のいずれか 1 つにでも項目がマッチすれば、その項目は選択されます。

次の例では、カスタム属性の **MyCustomAttribute** または **MyOtherCustomAttribute** を持つすべての型が選択されます。

MyCustomAttribute または MyOther CustomAttribute を持つ型:

```
<type name=".*" excludetype="false" regex="true>
   <customattribute name="MyCustomAttribute"/>
...<customattribute name="MyOtherCustomAttribute"/>
</type>
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

カスタム属性の規則では、カスタム属性の名前の条件を正規表現によって記述することもできます。次の例も、カスタム属性の **MyCustomAttribute** または **MyOtherCustomAttribute** を持つすべての型が選択されます。

MyCustomAttribute または MyOther CustomAttribute を持つ型:

```
<type name=".*" excludetype="false" regex="true>
   <customattribute name="My.*CustomAttribute" regex="true"/>
</type>
```

次の例は、**MyCustomAttribute という名前のカスタム属性を持つすべてのメソッドを除外する方法を示しています。**

カスタム属性を持つメソッドの対象除外

```
<type name=".*" excludetype="false" regex="true">
   <method name=".*" regex="true">
      <customattribute name="MyCustomAttribute"/>
   </method>
</type>
```

カスタム属性の規則は、**allowinheritance** 属性を指定することによって、サブタイプやオーバーライドするメソッドおよびプロパティに適用することができます。この属性の値が **true** に設定されていると、指定したカスタム属性を持つサブタイプやオーバーライドするメソッドおよびプロパティも対象から除外されるようになります。

2.6.3.22.1.8 スーパータイプによる対象除外

型は、スーパータイプによって選択的に対象から除外できます。スーパータイプの規則では、指定した型が継承する型の名前を基に型を選択します。型を選択する規則の中に、スーパータイプの規則を 1 つ以上入れ子にすることができます。

型の規則には、スーパータイプの規則を複数関連付けることができます。この場合、スーパータイプの規則のいずれか 1 つにでも項目がマッチすれば、その項目は選択されます。

次の例では、**MySupertype** から継承するすべての型が選択されます。

MySupertype から継承するすべての型:

```
<type name=".*" excludetype="false" regex="true">
   <supertype name="MySupertype"/>
</type>
```

スーパータイプの規則では、スーパータイプの名前の条件を正規表現によって記述することもできます。次の例は、**MySupertype** または **MyOtherSupertype** のいずれかから継承するすべての型を選択する方法を示しています。

MySupertype または MyOtherSupertype から継承するすべての型:

```
<type name=".*" excludetype="false" regex="true">
   <supertype name="My.*Supertype" regex="true"/>
</type>
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.3.22.1.9 アセンブリの対象除外

アセンブリは、名前によって対象から除外できます。アセンブリが対象から除外されると、そのアセンブリのモジュール内のすべての型およびメンバーが除外されます。以下のような状況では、アセンブリを対象から除外するのが妥当です。

- アセンブリ **A** は Dotfuscator で処理する必要がある
- アセンブリ **B** は Dotfuscator で処理してはならない
- アセンブリ **B** はアセンブリ **A** に依存している

つまり、**A** は B だけにサービスを提供しています。**B** の内部に埋め込まれた **A** への参照を Dotfuscator で処理する場合、**B** は **A** と同時に処理されるよう指定しますが、**A** は名前の変更対象または制御フローの難読化からは除外します。

アセンブリの対象除外

```
<assembly>
  <file dir="c:\projects\project1\" name="ExcludedLib.dll"/>
</assembly>
```

2.6.3.22.1.10 モジュールの対象除外

モジュールは、名前によって対象から除外できます。アセンブリ属性を使用すると、モジュールを特定のアセンブリに限定できます。アセンブリ属性を指定する場合、アセンブリ名は物理ファイル名ではなく論理アセンブリ名で指定する必要があります。モジュールが対象から除外されると、そのモジュールで定義されているすべての型およびメンバーが対象から除外されます。

当然、指定したモジュールが複数のアセンブリ間で共有されている場合、そのモジュールはすべてのアセンブリから除外されます。

モジュールをすべてのアセンブリから除外する:

```
<module name="MyLibResource.dll" assemblyname="MyLib"/>
```

2.6.3.22.2 対象選択規則

対象選択リスト セクションは、入力アセンブリに対する文字列の暗号化および不要コードの除去機能を動的に微調整する方法を提供します。ユーザーは、実行時に適用される規則のリストを指定します。規則で特定のクラス、メソッド、フィールド、プロパティ、またはイベントを選択すると、その項目は文字列の暗号化および不要コードの除去の対象として選択されます

これらの規則は、library オプションなどのグローバル オプションによって暗黙に適用される規則と併せて適用されます。

規則は論理的に OR で結合されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

名前空間、型、メソッド、フィールド、プロパティ、またはイベントの選択には、正規表現(RE)を使用できます。これを実行するには、省略可能な **regex** 属性を使用します。**regex** の既定値は false です。**regex** が true の場合、**name** 属性の値は正規表現と解釈されます。regex が false の場合、name は文字どおりに解釈されます。正規表現は特定の文字(ピリオドなど)に特別な意味を割り当てているので、regex の値は重要です。

以下に、簡単な正規表現の例を示します。

正規表現:	
`.*`	Matches anything
`MyLibrar.`	Matches MyLibrary, MyLibrari, etc.
`My[\.]Test[\.]I.*`	Matches My.Test.Int1,My.Test.Internal, etc.
`Get.*`	Matches GetInt, GetValue, etc.
`Get*`	Matches Ge,Get,Gett,Gettt, etc.

正規表現の構文の詳細については、.NET Framework のドキュメントを参照してください。

2.6.3.22.2.1 名前空間の対象選択

このオプションは、指定した名前空間内のすべての型とそのメソッドを対象として選択します。名前空間の指定には正規表現を使用できます。

正規表現

```
<namespace name="My.Included.Namespace"/>
```

2.6.3.22.2.2 型の対象選択

このオプションは、名前または属性指定子によって型を対象として選択します。型名の指定には正規表現を使用できます。

型名は完全修飾名である必要があります。

内側の(入れ子になっている)クラスは、外側と内側のクラスの間に '/' を区切り文字として使用して指定します。たとえば、次のようになります。

Inner (Nested) Classes

```
<type name="Library.Class1/NestedClass"/>
```

属性指定子の選択や選択解除には、**speclist** 属性を使用します。**speclist** 属性は、型の有効な属性指定子をカンマで区切って指定します。利用可能な値は以下のとおりです。

Attribute Specifiers:

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
abstract
interface
nestedassembly
nestedfamily
nestedfamorassem
nestedprivate
nestedpublic
notpublic
public
sealed
serializable
enum
```

属性指定子の前に '-' を付けると、規則の意味が反転します（つまり、指定した属性を持たないすべてのクラスを対象として選択します）。'+' を指定することもできますが、必要ありません。このリストに暗黙に適用される規則は、論理的に **AND** で結合されます（つまり、対象として選択する型のセットは、各規則に一致するすべての型の共通部分となります）。たとえば、次の規則は public かつ sealed である型のメソッドをすべて対象として選択します。

Include Method with Public and Sealed Types:

```
<type name=".*" speclist="+public,+sealed" regex="true"/>
```

<type> 要素を使用して型を選択し、選択した型内から個別のメソッドを対象として選択する規則を指定することもできます。これにより、ある型の一部のメソッドでのみ文字列を暗号化でき、それ以外のメソッドでは暗号化できないようになります。なお、**<type>** 要素の **excludetype** 属性は文字列の暗号化の対象選択には使用されないことに注意してください。

Allow String Encryption in Some Methods of Type:

```
<type name="MyCo.Test.MyOtherTest">
<!-- individual methods included here -->
...
</type>
```

<type> 要素が入れ子になった **<method>** 要素を含んでいない場合は、すべてのメソッドが対象として選択されます。これは、対象除外規則とは対照的です。

除外にのみ適用：

applytoderivedtypes 属性を指定することによって、継承階層全体に型の規則を適用することができます。この属性の値を **true** に設定すると、型の規則および、それに含まれるすべてのメソッド、フィールド、プロパティ、イベント、カスタム属性、またはスーパータイプの規則が、選択されている型とそれから派生するすべての型に適用されます。指定がない場合、既定値は **false** であり、指定した型にのみ型の規則が適用されます。

2.6.3.22.2.3 メソッドの対象選択

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

メソッドの選択は、まず **<type>** 要素を使用して型を選択し、次に対象とするメソッドを選択する規則を指定することによって行われます。メソッドは、(前述の型のセクションで説明したように)名前や属性指定子によって対象として選択できるほか、署名によっても選択できます。利用可能な属性指定子は以下のとおりです。

利用可能な属性指定子は以下のとおりです。

```
abstract
assembly
family
familyorassembly
final
private
public
static
virtual
```

属性指定子が明示的に設定されていない場合、**speclist** 属性は選択条件として使用されません。

次の例では、**Set** で始まるすべてのパブリック インスタンス メソッドが選択されます。

Set で始まるすべてのパブリック インスタンス メソッド:

```
<method regex="true" name="Set.*" speclist="+public,-static"/>
```

メソッドの署名は、**signature** 属性を使用して指定します。署名は、メソッドの戻り値の型とパラメーターの型を特定します。

メソッドの戻り値の型とパラメーターの型:

```
signature=""  <!-- 空のパラメーター リスト -->

signature="string(int,MyClass,MyClass[])"
```

署名が明示的に設定されていない場合、メソッドの署名は選択条件として使用されません。

次の例では、署名によってメソッドが選択されます。

署名によるメソッドの選択

```
<method name="DoIt" signature="string(int, System.Console,
System.Collection.ICollection, float[])"/>
```

Module:mod_name という名前の特別な型セレクターを使用すると、グローバル メソッドを指定できます。**mod_name** には、グローバル メソッドを含んでいるモジュールの名前を指定します。

2.6.3.22.2.4 フィールドの対象選択

フィールドの対象選択は、除去トリガーおよび除去の条件付き選択対象でのみ有効です。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

フィールドの選択は、まず **<type>** 要素を使用して型を選択し、次にフィールドを選択する規則を指定することによって行われます。フィールドは、(前述の型のセクションで説明したように)名前および属性指定子によって選択できます。利用可能な属性指定子は以下のとおりです。

利用可能な属性指定子は以下のとおりです。

```
public
private
static
assembly
family
familyandassembly
familyorassembly
notserialized
```

属性指定子が明示的に設定されていない場合、フィールドの属性は選択条件として一切使用されません。

次の例では、"**ENUM_**" で始まるすべての静的フィールドが選択されます。

"ENUM_" で始まる静的フィールド:

```
<field regex="true" name="ENUM_.*" speclist="+static"/>
```

Module:mod_name という名前の特別な型セレクターを使用すると、グローバル フィールドを指定できます。**mod_name** には、グローバル フィールドを含んでいるモジュールの名前を指定します。

2.6.3.22.2.5 プロパティの対象選択

プロパティの対象選択は、不要コードの除去でのみ有効です。プロパティの規則は、型の規則によって限定されるので、規則の編集ビューでは型ノードの子として表示されます。プロパティの規則は、(親の型の規則の条件を満たすすべての型のプロパティのうちで)条件を満たすすべてのプロパティを選択します。サポートされる選択条件は、プロパティ名およびプロパティの属性です。利用可能な属性指定子は以下のとおりです。

利用可能な属性指定子は以下のとおりです。

```
public
private
static
assembly
family
familyandassembly
familyorassembly
```

属性指定子が明示的に設定されていない場合、プロパティの属性は選択条件として使用されません。

次の例では、"**Sample** " で始まるすべてのプロパティが選択されます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

"Sample" で始まるプロパティ:

```
<property regex="true" name="Sample.*"/>
```

プロパティの署名は、**signature** 属性を使用して指定します。署名はプロパティの型を指定します。

署名はプロパティの型を指定します。

```
signature=""  <!-- empty signature -->

signature="int"
```

署名が明示的に設定されていない場合、プロパティの型は選択条件として使用されません。

Module:mod_name という名前の特別な型セレクターを使用すると、グローバル プロパティを指定できます。**mod_name** には、グローバル プロパティを含んでいるモジュールの名前を指定します。

2.6.3.22.2.6 イベントの対象選択

イベントの対象選択は、不要コードの除去でのみ有効です。イベントの規則は、型の規則によって限定されるので、規則の編集ビューでは型ノードの子として表示されます。イベントの規則は、(親の型の規則の条件を満たすすべての型のフィールドのうちで)条件を満たすすべてのイベントを選択します。サポートされる選択条件は、イベント名およびイベントの属性です。利用可能な属性指定子は以下のとおりです。

利用可能な属性指定子は以下のとおりです。

```
public
private
static
assembly
family
familyandassembly
familyorassembly
```

属性指定子が明示的に設定されていない場合、イベントの属性は選択条件として使用されません。

次の例では、"**On**" で始まるすべてのイベントが選択されます。

"On" で始まるイベント:

```
<event regex="true" name="On.*"/>
```

Module:mod_name という名前の特別な型セレクターを使用すると、グローバル イベントを指定できます。**mod_name** には、グローバル イベントを含んでいるモジュールの名前を指定します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.3.22.2.7 カスタム属性による対象選択

型およびメソッドは、カスタム属性を基に選択的に対象として選択できます。カスタム属性の規則では、項目（型またはメソッド）の追加情報であるカスタム属性の名前を基に項目を選択します。型またはメソッドを選択する規則の中に、カスタム属性の規則を 1 つ以上入れ子にすることができます。

型またはメソッドの規則には、カスタム属性の規則を複数関連付けることができます。この場合、カスタム属性の規則のいずれか 1 つにでも項目がマッチすれば、その項目は選択されます。

次の例では、カスタム属性の **MyCustomAttribute** または **MyOtherCustomAttribute** を持つすべての型が選択されます。

MyCustomAttribute または MyOtherCustomAttribute を持つ型:

```
<type name=".*" excludetype="false" regex="true>
   <customattribute name="MyCustomAttribute"/>
...<customattribute name="MyOtherCustomAttribute"/>
</type>
```

カスタム属性の規則では、カスタム属性の名前の条件を正規表現によって記述することもできます。次の例も、カスタム属性の **MyCustomAttribute** または **MyOtherCustomAttribute** を持つすべての型が選択されます。

MyCustomAttribute または MyOtherCustomAttribute を持つ型の選択:

```
<type name=".*" excludetype="false" regex="true>
   <customattribute name="My.*CustomAttribute" regex="true"/>
</type>
```

次の例は、**MyCustomAttribute** という名前のカスタム属性を持つすべてのメソッドを対象として選択する方法を示しています。

MyCustomAttribute を持つメソッドの対象選択:

```
<type name=".*" excludetype="false" regex="true">
   <method name=".*" regex="true">
      <customattribute name="MyCustomAttribute"/>
   </method>
</type>
```

カスタム属性の規則は、**allowinheritance** 属性を指定することによって、サブタイプやオーバーライドするメソッドおよびプロパティに適用することができます。この属性の値が **true** に設定されていると、指定したカスタム属性を持つサブタイプやオーバーライドするメソッドおよびプロパティも対象から除外されるようになります。

2.6.3.22.2.8 スーパータイプによる対象選択

型は、スーパータイプを基に選択的に対象として選択できます。スーパータイプの規則では、指定した型が継承する型の名前を基に型を選択します。型を選択する規則の中に、スーパータイプの規則を 1 つ以上入れ子にすることができます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

型の規則には、スーパータイプの規則を複数関連付けることができます。この場合、スーパータイプの規則のいずれか 1 つにでも項目がマッチすれば、その項目は選択されます。

次の例では、**MySupertype** から継承するすべての型が選択されます。

MySuperType から継承する型の選択:

```
<type name=".*" excludetype="false" regex="true">
   <supertype name="MySupertype"/>
</type>
```

スーパータイプの規則では、スーパータイプの名前の条件を正規表現によって記述することもできます。次の例は、**MySupertype** または **MyOtherSupertype** のいずれかから継承するすべての型を選択する方法を示しています。

MySupertype または MyOtherSupertype から継承する型の選択:

```
<type name=".*" excludetype="false" regex="true">
   <supertype name="My.*Supertype" regex="true"/>
</type>
```

2.6.3.22.2.9 アセンブリの対象選択

アセンブリは、名前によって対象として選択できます。アセンブリが対象として選択されると、そのアセンブリのモジュール内のすべての型およびメソッドが対象として選択されます。

アセンブリの対象選択

```
<assembly>
 <file dir="c:\projects\project1\" name="IncludedLib.dll"/>
</assembly>
```

2.6.3.22.2.10 モジュールの対象選択

モジュールは、名前によって対象として選択できます。アセンブリ属性を使用すると、モジュールを特定のアセンブリに限定できます。アセンブリ属性を指定する場合、アセンブリ名は物理ファイル名ではなく論理アセンブリ名で指定する必要があります。モジュールが対象として選択されると、そのモジュールで定義されているすべての型およびメンバーが対象として選択されます。

当然、指定したモジュールが複数のアセンブリ間で共有されている場合、そのモジュールはすべてのアセンブリに対して処理対象として選択されます。

モジュールの対象選択

```
<module name="MyLibResource.dll" assemblyname="MyLib"/>
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.3.23 dotfuscator_v2.3.dtd

dotfuscator_v2.3.dtd は、Dotfuscator によって生成される構成ファイルのフォーマットが記述されている DTD (文書型定義) ファイルです。このファイルは、www.preemptive.com/dotfuscatorresources/dtd/dotfuscator_v2.3.dtd で参照できます。

2.6.4 カスタム属性リファレンス

このセクションには、インストルメンテーションに使用されるカスタム属性およびパブリック クラスのリファレンス ドキュメントが含まれています。これらのクラスは PreEmptive.Attributes.dll 内に定義されており、既定で Dotfuscator のインストール フォルダーにインストールされます。

2.6.4.1 PreEmptive.Attributes

クラス

ApplicationAttribute

BinaryAttribute

BusinessAttribute

ExceptionTrackAttribute

FeatureAttribute

InsertShelfLifeAttribute

InsertSignOfLifeAttribute

InsertTamperCheckAttribute

PerformanceProbeAttribute

SetupAttribute

SystemProfileAttribute

TeardownAttribute

列挙

ExceptionTypes

FeatureEventTypes

InjectionPoints

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

SinkElements

SourceElements

2.6.4.1.1 ApplicationAttribute

概要

アセンブリの、インストルメンテーション時のカスタム属性です。Dotfuscator はこの属性を、Runtime Intelligence コードで使用できる属性に変換します。この属性に指定した値は、アプリケーションを識別するために Runtime Intelligence メッセージで送信されます。これには改ざん通知およびアプリケーション分析のメッセージが含まれます。

ApplicationAttribute
`public class ApplicationAttribute :Attribute`

コンストラクター メンバー

名前	概要
ApplicationAttribute(String guid)	一意な ID を持つ ApplicationAttribute を作成します。Runtime Intelligence コードはアセンブリ自身を参照して、名前とバージョンを見つけます。
ApplicationAttribute(String guid, String name, String version)	一意な ID、名前、およびバージョンを持つ ApplicationAttribute を作成します。
ApplicationAttribute(String guid, String name, String version, String applicationType)	一意な ID、名前、バージョン、およびアプリケーションの種類を持つ ApplicationAttribute を作成します。

プロパティ メンバー

名前	概要
ApplicationType :String	アプリケーションの種類を取得します。ヌル値を指定できます。
Guid :String	アプリケーションの一意な ID を GUID 形式の文字列で取得します。
Name :String	アプリケーションの名前を取得します。ヌル値を指定できます。
Version :String	アプリケーションのバージョンを取得します。ヌル値を指定できます。

2.6.4.1.2 BinaryAttribute

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

概要

アセンブリの、インストルメンテーション時のカスタム属性です。Dotfuscator はこの属性を、Runtime Intelligence コードで使用できる属性に変換します。この属性に指定した値は、アプリケーションを形成する特定のバイナリ(アセンブリ)を識別するために、Runtime Intelligence メッセージで送信されます。

BinaryAttribute
`public class BinaryAttribute :Attribute`

コンストラクター メンバー

名前	概要
BinaryAttribute(String guid)	一意な ID を持つ BinaryAttribute を作成します。

プロパティ メンバー

名前	概要
Guid :String	バイナリの一意な ID を GUID 形式の文字列で取得します。

2.6.4.1.3 BusinessAttribute

概要

アセンブリの、インストルメンテーション時のカスタム属性です。Dotfuscator はこの属性を、Runtime Intelligence コードで使用できる属性に変換します。この属性に指定した値は、ビジネス情報を識別するために Runtime Intelligence メッセージで送信されます。

BusinessAttribute
`public class BusinessAttribute :Attribute`

コンストラクター メンバー

名前	概要
BusinessAttribute(String companyKey)	指定された CompanyKey を持つ BusinessAttribute を作成します。
BusinessAttribute (String companyKey, String companyName)	指定された CompanyKey および CompanyName を持つ BusinessAttribute を作成します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロパティ メンバー

名前	概要
CompanyKey :String	会社キーを取得します。会社キーは、サービスへの登録時に各会社へ提供されるトークンです。キーをヌルまたは空にすることはできません。
CompanyName :String	会社名を取得します。ヌル値を指定できます。

2.6.4.1.4 ExceptionTrackAttribute

概要

ExceptionTrackAttribute は、アセンブリおよびメソッドの、インストルメンテーション時の属性です。Dotfuscator は、この属性を持つあらゆるアセンブリまたはメソッドに例外の追跡コードを挿入します。実行時、この例外の追跡コードにより、キャッチされた、スローされた、あるいはハンドルされない例外を検出することができます。これらの検出を組み合わせるには、複数の ExceptionTrackAttributes をアセンブリやメソッドで定義します。検出されると、例外の追跡コードがユーザーから詳細を収集し、検出された例外を Runtime Intelligence エンドポイントに報告します。ユーザーは、前もって Runtime Intelligence メッセージの送信を止めてしまっている場合でも、例外レポートの送信を明示的に許可することができ、コメントや連絡先情報を提供してレポートと一緒に送信することができます。また、開発者は、例外が検出されたときに実行するカスタム動作を指定できます。例外の追跡レポート メッセージを送信するには、アプリケーションの中に Setup 属性および Teardown 属性が指定されたメソッドを含んでいる必要があります。ExceptionTrackAttribute 属性を Setup 属性が含まれているメソッドに置くことはできません。この属性を含んでいるメソッドは、Setup 属性を含んでいるメソッドの後に実行する必要があります。

例外通知

`SinkElement` は、例外が検出されたときに呼び出すメソッド、あるいはアプリケーションに設定するプロパティまたはフィールドを指定するために使用できます。ほかの、チェックが実行されると必ず呼び出される類の **`SinkElement`** とは異なり、**`ExceptionNotificationSinkElement`** は、設定した種類の例外が属性付きのアセンブリまたはメソッド内で検出された場合にのみ呼び出されます。

`SinkElement` がプロパティである場合、そのプロパティの型は **`Exception`** である必要があります。値には、検出発生時に検出された例外が設定されます。メソッド レベルの例外の追跡を使用している場合、プロパティは書き込み可能で、属性付きメソッドからアクセスできる必要があります。プロパティが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。アセンブリ レベルの例外の追跡を使用している場合、プロパティはパブリックかつ静的で、書き込み可能である必要があります。プロパティの名前を **`ExceptionNotificationSinkName`** で指定する必要があります。プロパティの型定義を **`ExceptionNotificationSinkOwner`** で指定する必要があります。メソッド レベルの例外の追跡を使用している場合は、これを設定しないでおいてもかまいません。その場合には、Dotfuscator が属性付きメソッドの型定義でプロパティを探します。アセンブリ レベルの例外の追跡を使用している場合には、**`ExceptionNotificationSinkOwner`** の設定は必須です。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

SinkElement がフィールドである場合、そのフィールドの型は **Exception** である必要があります。値には、検出発生時に検出された例外が設定されます。メソッド レベルの例外の追跡を使用している場合、フィールドは属性付きメソッドからアクセスできる必要があります。フィールドが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。アセンブリ レベルの例外の追跡を使用している場合、フィールドはパブリックかつ静的である必要があります。フィールドの名前を **ExceptionNotificationSinkName** で指定する必要があります。フィールドの型定義を **ExceptionNotificationSinkOwner** で指定する必要があります。メソッド レベルの例外の追跡を使用している場合は、これを設定しないでおいてもかまいません。その場合には、Dotfuscator が属性付きメソッドの型定義でフィールドを探します。アセンブリ レベルの例外の追跡を使用している場合には、**ExceptionNotificationSinkOwner** の設定は必須です。

SinkElement がメソッドである場合、そのメソッドは検出が発生したときに、単一の **Exception** 型パラメーターで呼び出されます。メソッド レベルの例外の追跡を使用している場合、ターゲット メソッドは属性付きメソッドからアクセスできる必要があります。ターゲット メソッドが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。アセンブリ レベルの例外の追跡を使用している場合、メソッドはパブリックかつ静的である必要があります。ターゲット メソッドの名前を **ExceptionNotificationSinkName** で指定する必要があります。ターゲット メソッドの型定義を **ExceptionNotificationSinkOwner** で指定する必要があります。メソッド レベルの例外の追跡を使用している場合は、これを設定しないでおいてもかまいません。その場合には、Dotfuscator が属性付きメソッドの型定義でターゲット メソッドを探します。アセンブリ レベルの例外の追跡を使用している場合には、**ExceptionNotificationSinkOwner** の設定は必須です。

SinkElement がメソッドの引数である場合、名前付き引数はデリゲート型である必要があり、このデリゲートは `void( System.Exception )` で署名されていなければなりません。デリゲートは、検出発生時、検出された例外の値に設定された Exception 型の引数で呼び出されます。メソッド引数の名前を **ExceptionNotificationSinkName** で指定する必要があります。**ExceptionNotificationSinkOwner** は使用されません。

SinkElement がデリゲートである場合は、デリゲート型のフィールドである必要があります。デリゲートは、検出発生時、検出された例外に設定された **Exception** 型の引数で呼び出されます。メソッド レベルの例外の追跡を使用している場合、フィールドは属性付きメソッドからアクセスできる必要があります。フィールドが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。アセンブリ レベルの例外の追跡を使用している場合、フィールドはパブリックかつ静的である必要があります。フィールドの名前を **ExceptionNotificationSinkName** で指定する必要があります。**ExceptionNotificationSinkOwner** は使用されません。

ExceptionNotificationSinkElement が "DefaultAction" または "None" に設定されている場合、Dotfuscator は検出した例外に対応するためのコードを一切差し込みません。しかしながら、**SendReport** 属性プロパティを介して Runtime Intelligence Service に適切なメッセージを送信するよう構成されている場合には、Dotfuscator はメッセージを生成するコードを生成します。

Dotfuscator はこのカスタム属性をインストルメンテーションした後、メタデータからこの属性を削除します。

ExceptionTrackAttribute
`public class ExceptionTrackAttribute :Attribute`

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

コンストラクター メンバー

名前	概要
ExceptionTrackAttribute()	クラスの新しいインスタンスを初期化します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロパティ メンバー

名前	概要
ExceptionNotificationSinkElement :SinkElements	設定した ExceptionType の例外の検出に関して対応を取るかどうか、またその対応方法を示します。シンク要素には、書き込み可能なフィールド、設定可能な Exception 型のプロパティ、呼び出す void(System.Exception) メソッドまたはデリゲートを指定できます。このプロパティを使用するには、ExceptionNotificationSinkName も設定しておく必要があります。メソッド レベルの例外の追跡を使用しており、ExceptionNotificationSinkElement がフィールド、メソッド、またはプロパティである場合、フィールド、メソッド、またはプロパティが属性付きメソッドと同じクラスで定義されない限り、ExceptionNotificationSinkOwner も定義する必要があります。アセンブリ レベルの例外の追跡を使用している場合は、必ず ExceptionNotificationSinkOwner も設定する必要があります。SinkElement を "None" または "DefaultAction" に設定した場合、Dotfuscator は検出した例外に対応するためのコードを一切差し込みません。SendReport プロパティが true に設定されている場合は、本プロパティが "None" または "DefaultAction" に設定されているかどうかにかかわらず、例外レポートを送信するコードが差し込まれるようになります。
ExceptionNotificationSinkName :String	設定した種類の例外が検出されたときに設定するプロパティ、フィールドの名前、または呼び出すメソッドの名前。アセンブリ レベルの例外の追跡を使用している場合、このプロパティ、フィールド、またはメソッドは、パブリックかつ静的で、書き込み可能である必要があります。このプロパティを設定した場合は、ExpirationNotificationSinkElement の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は検出した例外に対応するためのコードを一切差し込みません。SendReport プロパティが true に設定されている場合は、本プロパティが設定されているかどうかにかかわらず、例外レポートを送信するコードが差し込まれるようになります。
ExceptionNotificationSinkOwner :Type	ExceptionNotificationSinkElement がフィールドである場合には、ExceptionNotificationSinkOwner はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。アセンブリ レベルの例外の追跡では、ExceptionNotificationSinkOwner の設定は必須です。
ExceptionType :ExceptionTypes	この ExceptionTrackAttribute で追跡する例外の種類(caught、thrown、または unhandled)。既定値は Unhandled です。
ExtendedKeyMethodArguements : string	実行時、メッセージの拡張キー データに自動的に追加する必要のあるパラメーター名および値を示すパターン。サポートされているパターンの詳細については、「メソッド パラメーターを拡張キーとして自動的に送信す

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

	る」を参照してください。
ExtendedKeySourceElement :SourceElements	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書にアクセスする方法を示します（たとえば、フィールド、プロパティ、メソッド、またはメソッド引数）。このプロパティを使用するには、`ExtendedKeySourceName` も設定しておく必要があります。メソッド レベルの例外の追跡を使用しており、`ExtendedKeySourceElement` がフィールド、メソッド、またはプロパティである場合、フィールド、メソッド、またはプロパティが属性付きメソッドと同じクラスで定義されない限り、`ExtendedKeySourceOwner` も定義する必要があります。アセンブリ レベルの例外の追跡を使用している場合は、必ず `ExtendedKeySourceOwner` も設定する必要があります。
ExtendedKeySourceName : string	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書を格納するプロパティ、フィールド、メソッド、またはメソッド引数の名前。アセンブリ レベルの例外の追跡を使用している場合、このプロパティ、フィールド、またはメソッドは、パブリックかつ静的で、書き込み可能である必要があります。このプロパティを設定した場合、`ExtendedKeySourceElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は拡張キー情報を送信するコードを生成しません。
ExtendedKeySourceOwner :Type	`ExtendedKeySourceElement` がフィールドである場合には、`ExtendedKeySourceOwner` はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。アセンブリ レベルの例外の追跡では、`ExtendedKeySourceOwner` の設定は必須です。
PrivacyPolicyUri : string	例外レポートの送信を保護するプライバシー ポリシーの URL。このプロパティは、`ReportInfoSourceElement` が DefaultAction に設定されている場合にのみ使用されます。この場合は、PrivacyPolicyUri へのリンクが組み込みの例外レポート ダイアログに含まれます。このプロパティが設定されていない場合、例外レポート ダイアログにはプライバシー ポリシーのリンクが含まれません。
ReportInfoSourceElement :SourceElements	実行時、例外が検出された時点でユーザー指定のレポート情報の辞書にアクセスする方法を示します。既定値は None です。このプロパティが None に設定されている場合、Dotfuscator はユーザー指定の例外レポート情報を取得するためのコードを生成しません。このプロパティが DefaultAction に設定されている場合、Dotfuscator は組み込みダイアログを表示するためのコードを生成します。このダイアログは、例外メッセージを表示し、ユーザーに対して、自身のアドレスおよびコメントを入力し、レポートを送信することに同意するよう求めます。このプロパティがこれ以外の値に設定されている場合は、`ReportInfoSourceName` も設定しておく必要があります。メソッド レベルの例外の追跡を使用しており、`ReportInfoSourceElement` がフィールド、メソッド、またはプロ

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

	パティである場合、フィールド、メソッド、またはプロパティが属性付きメソッドと同じクラスで定義されない限り、ReportInfoSourceOwner も定義する必要があります。アセンブリ レベルの例外の追跡を使用している場合は、必ず ReportInfoSourceOwner も設定する必要があります。ユーザー指定のレポート情報の辞書で認識されるキーと値のペアの説明については、「ユーザー指定の例外レポート情報の収集」を参照してください。
ReportInfoSourceName : string	実行時、属性付きメソッドが呼び出された時点でユーザー指定のレポート情報の辞書を格納するプロパティ、フィールド、メソッド、またはメソッド引数の名前。アセンブリ レベルの例外の追跡を使用している場合、このプロパティ、フィールド、またはメソッドは、パブリックかつ静的で、書き込み可能である必要があります。このプロパティを設定した場合、ReportInfoSourceElement の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator はユーザー指定の例外レポート情報を取得するためのコードを生成しません。
ReportInfoSourceOwner :Type	ReportInfoSourceElement がフィールドである場合には、ReportInfoSourceOwner はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。アセンブリ レベルの例外の追跡では、ReportInfoSourceOwner の設定は必須です。
SendReport : bool	例外を追跡するときに例外レポートを送信するかどうかを示します。既定値は true です。このプロパティが false に設定されている場合、ReportInfoSource プロパティは無視されます。カスタム動作が構成されている場合は、このプロパティの値にかかわらず、ExceptionNotificationSink プロパティを介してその動作が常に実行されます。

2.6.4.1.4.1 ExceptionTypes

概要

追跡する例外の種類です。

Exception Types
`public enum ExceptionTypes`

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

列挙メンバー

フィールド	概要
Unhandled	ハンドルされない例外を追跡します。
Caught	'catch' ブロックに入った直後の例外を追跡します。
Thrown	'throw' ステートメントによってスローされる直前の例外を追跡します。

2.6.4.1.5 FeatureAttribute

概要

FeatureAttribute はアプリケーション分析処理の、インストルメンテーション時のカスタム属性です。Dotfuscator は、Runtime Intelligence の機能メッセージを送信するコードを属性付きメソッドに挿入します。

Dotfuscator はこのカスタム属性をインストルメンテーションした後、メタデータからこの属性を削除します。

機能のイベントの種類

機能は、一度きりのイベント(Ticks)として、あるいは個々の Stop イベントと Start イベントとして定義することができます。選択するイベントの種類は、Dotfuscator が生成するコードの種類に影響を与えます。

Tick イベントは、属性付きメソッドを実行するときに 1 つの Runtime Intelligence 機能メッセージを送信します。Start/Stop イベントでは、2 つのメッセージが別々に、1 つは開始、1 つは停止で送信されます。

Start/Stop イベントを使用するには 2 つの FeatureAttributes が必要です(必要ならば、同じメソッドを指定できます)。Start イベント用に生成されるコードは属性付きメソッドの最初に追加され、Stop イベント用に生成されるコードはメソッドの最後に追加されます。

Feature 属性
`public class FeatureAttribute :Attribute`

コンストラクター メンバー

名前	概要
FeatureAttribute(String featureName)	名前付き機能を表す新しい `FeatureAttribute` を作成します。機能の名前は、製品に指定されている機能の名前、または機能セットの名前と一致している必要があります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロパティ メンバー

名前	概要
ExtendedKeyMethodArguements : string	実行時、メッセージの拡張キー データに自動的に追加する必要のあるパラメーター名および値を示すパターン。サポートされているパターンの詳細については、「メソッド パラメーターを拡張キーとして自動的に送信する」を参照してください。
ExtendedKeySourceElement :SourceElements	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書にアクセスする方法を示します（たとえば、フィールド、プロパティ、メソッド、またはメソッド引数）。このプロパティを使用するには、`ExtendedKeySourceName` も設定しておく必要があります。`ExtendedKeySourceElement` がフィールド、メソッド、またはプロパティである場合は、そのフィールド、メソッド、またはプロパティが属性付きメソッドと同じクラスで定義されていない限り、`ExtendedKeySourceOwner` も設定する必要があります。
ExtendedKeySourceName : string	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書を格納するプロパティ、フィールド、メソッド、またはメソッド引数の名前。このプロパティを設定した場合、`ExtendedKeySourceElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は拡張キー情報を送信するコードを生成しません。
ExtendedKeySourceOwner :Type	`ExtendedKeySourceElement` がフィールドである場合には、`ExtendedKeySourceOwner` はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。
EventType :FeatureEventTypes	機能のイベントの種類を表します。分析の場合、Dotfuscator はイベントの種類を基に異なるメッセージを送信します。Start と Stop のペアで機能の使用をひとくくりにすると、機能に使った時間を追跡できます。

2.6.4.1.5.1 FeatureEventTypes

概要

FeatureAttribute で考えられる機能のイベントの種類です。

FeatureEventTypes
`public enum FeatureEventTypes :Attribute`

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

列挙メンバー

フィールド	概要
Tick	この `FeatureAttribute` は、時間内に 1 インスタンスとして使用する機能を表すメソッドをマークします。
Start	この `FeatureAttribute` は、機能の開始を表すメソッドをマークします。
Stop	この `FeatureAttribute` は、機能の終了を表すメソッドをマークします。

2.6.4.1.6 InsertShelfLifeAttribute

概要

メソッドの、インストルメンテーション時の属性です。Dotfuscator は、この属性を持つあらゆるメソッドに Shelf Life（使用期限）コードを挿入します。実行時に、アプリケーションの有効期限が間もなく切れる、もしくは切れた場合、Shelf Life コードにより警告または期限切れメッセージを送信することができます。警告期間を入力すると、ユーザー定義の操作を実行できます。有効期限が切れたときの既定の動作はアプリケーションの終了ですが、代わりにユーザー定義の操作を実行することができます。Shelf Life を使用するには、Setup 属性および Teardown 属性が指定されたメソッドをアプリケーションが含んでいる必要があります。InsertShelfLife 属性を Setup 属性が含まれているメソッドに置くことはできません。この属性を含んでいるメソッドは、Setup 属性を含んでいるメソッドの後に実行する必要があります。

アプリケーション通知

`SinkElement` は、警告または期限切れイベントが発生したときに呼び出すメソッド、あるいはアプリケーションに設定するプロパティまたはフィールドを指定するために使用できます。

`SinkElement` がプロパティである場合、そのプロパティは警告期間に入るかアプリケーションの有効期限が切れた場合には true、それ以外の場合は false に設定されます。プロパティは書き込み可能で、属性付きメソッドからアクセスでき、かつブール型である必要があります。プロパティが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。プロパティの名前を **`ExpirationNotificationSinkName`** または **`WarningNotificationSinkName`** で指定する必要があります。生成されたコードは、プロパティ セッターからスローされる例外をキャッチしようとも処理しようともしません。プロパティの型定義を **`ExpirationNotificationSinkOwner`** または **`WarningNotificationSinkOwner`** で指定する必要があります。どちらも設定しないでおくと、Dotfuscator は属性付きメソッドの型定義でプロパティを探します。

`SinkElement` がフィールドである場合、そのフィールドは警告期間に入るか期限切れになった場合には true、それ以外の場合は false に設定されます。フィールドは属性付きメソッドからアクセスでき、かつブール型である必要があります。フィールドが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。フィールドの名前を **`ExpirationNotificationSinkName`** または **`WarningNotificationSinkName`** で指定する必要があります。フィールドの型定義を **`ExpirationNotificationSinkOwner`** または **`WarningNotificationSinkOwner`** で指定する必要があります。どちらも設定しないでおくと、Dotfuscator は属性付きメソッドの型定義でフィールドを探します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

SinkElement がメソッドである場合、そのメソッドは、警告期間に入るかアプリケーションの有効期限が切れた場合には true、それ以外の場合は false に設定された単一のブール型パラメーターで呼び出されます。ターゲット メソッドは属性付きメソッドからアクセスでき、かつブール型のパラメーターを 1 つ受け取る必要があります。ターゲット メソッドが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。ターゲット メソッドの名前を **ExpirationNotificationSinkName** または **WarningNotificationSinkName** で指定する必要があります。生成されたコードは、ターゲット メソッドからスローされる例外をキャッチしようとも処理しようともしません。ターゲット メソッドの型定義を **ExpirationNotificationSinkOwner** または **WarningNotificationSinkOwner** で指定する必要があります。どちらも設定しないでおくと、Dotfuscator は属性付きメソッドの型定義でターゲット メソッドを探します。

SinkElement がメソッドの引数である場合、名前付き引数はデリゲート型である必要があり、このデリゲートは `void( bool )` で署名されていなければなりません。デリゲートは、警告期間に入るかアプリケーションの有効期限が切れた場合には true、それ以外の場合は false に設定されたブール値引数で呼び出されます。生成されたコードは、呼び出し先のデリゲートからスローされる例外をキャッチしようとも処理しようともしません。メソッド引数の名前を **ApplicationNotificationSinkName** で指定する必要があります。**ApplicationNotificationSinkOwner** は使用されません。

SinkElement がデリゲートである場合は、デリゲート型のフィールドである必要があります。デリゲートは、警告期間に入るかアプリケーションの有効期限が切れた場合には true、それ以外の場合は false に設定されたブール値引数で呼び出されます。フィールドが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。フィールドの名前を **ExpirationNotificationSinkName** または **WarningNotificationSinkName** で指定する必要があります。この場合、**ExpirationNotificationSinkOwner** または **WarningNotificationSinkOwner** は使用されません。デリゲートは **void(bool)** で署名されている必要があります。生成されたコードは、呼び出し先のデリゲートからスローされる例外をキャッチしようとも処理しようともしません。

ExpirationNotificationSinkElement を DefaultAction に設定すると、Dotfuscator はアプリケーションの有効期限が切れた場合にアプリケーションを終了するコードを差し込みます。

WarningNotificationSinkElement を DefaultAction に設定すると、Dotfuscator は警告固有のコードを一切差し込みません。

SinkElement を None に設定すると、Dotfuscator は警告期間に入るかアプリケーションの有効期限が切れた場合に対応するためのコードを一切差し込みません。しかしながら、[**グローバル オプション**]タブで Runtime Intelligence Service に適切なメッセージを送信するよう構成されている場合には、Dotfuscator はメッセージを生成するコードを生成します。

Dotfuscator はこのカスタム属性をインストルメンテーションした後、メタデータからこの属性を削除します。

InsertShelfLifeAttribute
`public class InsertShelfLifeAttribute :Attribute`

コンストラクター メンバー

名前	概要
InsertShelfLifeAttribute()	クラスの新しいインスタンスを初期化します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロパティ メンバー

名前	概要
ActivationKeyFile	Shelf Life Activation Key ファイルのパス。
ExpirationDate	アプリケーションの有効期限が切れる、またはアプリケーションが非アクティブになる日付を示します。 これは、アプリケーションの有効期限が切れる絶対日付（例：2009-12-31）、または Dotfuscator の処理後、期限切れになるまでの日数を指定できます。
ExpirationNotificationSinkElement :SinkElements	Shelf Life の期限切れ後、属性付きメソッドが呼び出された時点でアプリケーションに通知するかどうか、またその通知方法を示します。シンク要素には、書き込み可能なブール値フィールド、設定可能なブール値プロパティ、呼び出す void(boolean) メソッドまたはデリゲートを指定できます。このプロパティを使用するには、`ExpirationNotificationSinkName` も設定しておく必要があります。`ExpirationNotificationSinkElement` がフィールド、プロパティ、またはメソッドである場合は、属性付きメソッドと同じクラスでなければ、`ExpirationNotificationSinkOwner` も設定する必要があります。`SinkElement` を "DefaultAction" に設定すると、Dotfuscator は Shelf Life の期限切れが検出された場合にアプリケーションを終了するコードを差し込みます。
ExpirationNotificationSinkName :String	Shelf Life の期限切れチェック後に設定するプロパティ、フィールドの名前、または呼び出すメソッドの名前。このプロパティを設定した場合は、`ExpirationNotificationSinkElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator はアプリケーションに期限切れを通知するコードを生成しません。
ExpirationNotificationSinkOwner :Type	`ExpirationNotificationSinkOwner` は、`ExpirationNotificationSink` のメソッド、フィールド、プロパティ、引数、またはデリゲートを定義する型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きシンク要素が検索されます。
ExtendedKeyMethodArguements : string	実行時、メッセージの拡張キー データに自動的に追加する必要のあるパラメーター名および値を示すパターン。サポートされているパターンの詳細については、「メソッド パラメーターを拡張キーとして自動的に送信する」を参照してください。
ExtendedKeySourceElement :SourceElements	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書にアクセスする方法を示します（たとえば、フィールド、プロパティ、メソッド、またはメソッド引数）。このプロパティを使用するには、`ExtendedKeySourceName` も設定しておく必要があります。`ExtendedKeySourceElement` がフィールド、メソッド、またはプロ

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

	パティである場合、フィールド、メソッドまたはプロパティが属性付きメソッドと同じクラスで定義されない限り、ExtendedKeySourceOwner も定義する必要があります。
ExtendedKeySourceName : string	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書を格納するプロパティ、フィールド、メソッド、またはメソッド引数の名前。このプロパティを設定した場合、ExtendedKeySourceElement の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は拡張キー情報を送信するコードを生成しません。
ExtendedKeySourceOwner :Type	ExtendedKeySourceElement がフィールドである場合には、ExtendedKeySourceOwner はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。
PrivateKeyFile	PKCS #12 Private Key ファイルのパス。このファイルは Shelf Life トークンの追加の検証を提供するためのもので、使用は任意です。
PrivateKeyFilePassword	秘密キー ファイルを保護するパスワード。
ShelfLifeTokenSourceElement	ShelfLifeTokenSourceElement は、実行時、属性付きメソッドが呼び出された時点で任意の Shelf Life トークン ソースにアクセスする方法を示します（たとえば、フィールド、プロパティ、メソッド、またはメソッド引数）。このプロパティを使用するには、ShelfLifeTokenSourceName も設定しておく必要があります。ShelfLifeTokenSourceElement がフィールド、メソッド、またはプロパティである場合、フィールド、メソッド、またはプロパティが属性付きメソッドと同じクラスで定義されない限り、ShelfLifeTokenSourceOwner も定義する必要があります。
ShelfLifeTokenSourceName	実行時、属性付きメソッドが呼び出された時点で Shelf Life トークンを格納するプロパティ、フィールド、メソッド、またはメソッド引数の名前。このプロパティを設定した場合は、ShelfLifeTokenSourceElement の設定も必要になります。このプロパティを設定しないと、Dotfuscator はインストルメンテーション中に Shelf Life Activation Key から Shelf Life トークンを作成します。
ShelfLifeTokenSourceOwner	ShelfLifeTokenSourceElement がフィールドである場合には、ShelfLifeTokenSourceOwner はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。
WarningDate	アプリケーションの期限切れが間近に迫っていることを警告する日付を示します。 これは、アプリケーションが警告を発する必要のある絶対日付（例：2009-12-31）、または Dotfuscator の処理後、警告を発するまでの日数を指定できます。
WarningNotificationSinkElement	Shelf Life の期限切れが間近に迫っていることを、属性付きメソッドが呼

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

	び出された時点でアプリケーションに通知するかどうか、またその通知方法を示します。シンク要素には、書き込み可能なブール値フィールド、設定可能なブール値プロパティ、呼び出す void(boolean) メソッドまたはデリゲートを指定できます。このプロパティを使用するには、`WarningNotificationSinkName` も設定しておく必要があります。ApplicationNotificationSinkElement がフィールド、プロパティ、またはメソッドである場合、属性メソッドと同じクラスでなければ、`WarningNotificationSinkOwner` も設定する必要があります。`SinkElement` を "DefaultAction" に設定すると、Dotfuscator は追加の警告行為を実行するコードを一切差し込みません。
WarningNotificationSinkName	Shelf Life の警告チェック後に設定するプロパティ、フィールドの名前、または呼び出すメソッドの名前。このプロパティを設定した場合、`WarningNotificationSinkElement` の設定も必要になります。このプロパティを設定しないと、Dotfuscator は追加の警告行為を実行するコードを一切差し込みません。
WarningNotificationSinkOwner	`WarningNotificationSinkOwner` は、`WarningNotificationSink` のメソッド、フィールド、プロパティ、引数、またはデリゲートを定義する型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きシンク要素が検索されます。

2.6.4.1.7 InsertSignofLifeAttribute

概要

InsertSignOfLifeAttribute は、この属性によってインストルメント化されたメソッドが呼び出されるたびにメッセージを送信します。実行時に、Sign of Life コードはアプリケーションが実行されていることを示すメッセージを送信します。Sign of Life を使用するには、Setup 属性および Teardown 属性が指定されたメソッドをアプリケーションが含んでいる必要があります。InsertSignOfLife 属性を Setup 属性が含まれているメソッドに置くことはできません。この属性を含んでいるメソッドは、Setup 属性を含んでいるメソッドの後に実行する必要があります。

Dotfuscator はこのカスタム属性をインストルメンテーションした後、メタデータからこの属性を削除します。

InsertSignofLifeAttribute
`public class InsertShelfLifeAttribute :Attribute`

コンストラクター メンバー

名前	概要
InsertSignOfLifeAttribute()	クラスの新しいインスタンスを初期化します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロパティ メンバー

名前	概要
ActivationKeyFile	Shelf Life Activation Key ファイルのパス。
ExtendedKeyMethodArguements : string	実行時、メッセージの拡張キー データに自動的に追加する必要のあるパラメーター名および値を示すパターン。サポートされているパターンの詳細については、「メソッド パラメーターを拡張キーとして自動的に送信する」を参照してください。
ExtendedKeySourceElement :SourceElements	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書にアクセスする方法を示します（たとえば、フィールド、プロパティ、メソッド、またはメソッド引数）。このプロパティを使用するには、`ExtendedKeySourceName` も設定しておく必要があります。`ExtendedKeySourceElement` がフィールド、メソッド、またはプロパティである場合、フィールド、メソッドまたはプロパティが属性付きメソッドと同じクラスで定義されない限り、`ExtendedKeySourceOwner` も定義する必要があります。
ExtendedKeySourceName : string	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書を格納するプロパティ、フィールド、メソッド、またはメソッド引数の名前。このプロパティを設定した場合、`ExtendedKeySourceElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は拡張キー情報を送信するコードを生成しません。
ExtendedKeySourceOwner :Type	`ExtendedKeySourceElement` がフィールドである場合には、`ExtendedKeySourceOwner` はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。

2.6.4.1.8 InsertTamperCheckAttribute

概要

メソッドの、インストルメンテーション時の属性です。Dotfuscator は、この属性を持つあらゆるメソッドに改ざんチェック コードを挿入します。実行時に、アプリケーションの整合性チェックが失敗した場合、改ざんチェック コードにより Runtime Intelligence の改ざんメッセージを送信できます。また、アプリケーション内のカスタム コードを呼び出したり、単にアプリケーションを終了するためのコードを呼び出したりすることもできます。Runtime Intelligence の改ざんメッセージを送信したい場合は、Setup 属性が含まれているメソッドにこの属性を設定することはできません。この属性を含んでいるメソッドは、Setup 属性を含んでいるメソッドの後に実行する必要があります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

アプリケーション通知

SinkElement は、改ざんチェックが実行された後で呼び出すメソッド、あるいはアプリケーションに設定するプロパティまたはフィールドを指定するために使用できます。これにより、アプリケーションは改ざんが検出された場合または検出されなかった場合に、アプリケーション内部で発生しなければいけないことを指定できます。指定したフィールドまたはプロパティは、改ざんが検出されなかった場合でも常に設定されます（メソッドまたはデリゲートの場合は呼び出されます）。

SinkElement がプロパティである場合、そのプロパティは改ざんが検出された場合には true、検出されなかった場合には false に設定されます。プロパティは書き込み可能で、属性付きメソッドからアクセスでき、かつブール型である必要があります。プロパティが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。プロパティの名前を **ApplicationNotificationSinkName** で指定する必要があります。生成されたコードは、プロパティ セッターからスローされる例外をキャッチしようとも処理しようともしません。プロパティの型定義を **ApplicationNotificationSinkOwner** で指定する必要があります。**ApplicationNotificationSinkOwner** を設定しないでおくと、Dotfuscator は属性付きメソッドの型定義でプロパティを探します。

SinkElement がフィールドである場合、そのフィールドは改ざんが検出された場合には true、検出されなかった場合には false に設定されます。フィールドは属性付きメソッドからアクセスでき、かつブール型である必要があります。フィールドが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。フィールドの名前を **ApplicationNotificationSinkName** で指定する必要があります。フィールドの型定義を ApplicationNotificationSinkOwner で指定する必要があります。**ApplicationNotificationSinkOwner** を設定しないでおくと、Dotfuscator は属性付きメソッドの型定義でフィールドを探します。

SinkElement がメソッドの引数である場合、名前付き引数はデリゲート型である必要があり、このデリゲートは `void( bool )` で署名されていなければなりません。デリゲートは、改ざんが検出された場合には true、検出されなかった場合には false に設定されたブール値引数で呼び出されます。生成されたコードは、呼び出し先のデリゲートからスローされる例外をキャッチしようとも処理しようともしません。メソッド引数の名前を **ApplicationNotificationSinkName** で指定する必要があります。**ApplicationNotificationSinkOwner** は使用されません。

SinkElement がデリゲートである場合は、デリゲート型のフィールドである必要があります。デリゲートは、改ざんが検出された場合には true、検出されなかった場合には false に設定されたブール値引数で呼び出されます。フィールドが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。フィールドの名前を **ApplicationNotificationSinkName** で指定する必要があります。**ApplicationNotificationSinkOwner** は使用されません。デリゲートは `void( bool )` で署名されている必要があります。生成されたコードは、呼び出し先のデリゲートからスローされる例外をキャッチしようとも処理しようともしません。

SinkElement を DefaultAction に設定すると、Dotfuscator は改ざんが検出された場合にアプリケーションを終了するコードを差し込みます。

SinkElement を None に設定すると、Dotfuscator は改ざんが検出された場合に対応するためのコードを一切差し込みません（しかしながら、Runtime Intelligence Service にメッセージを送信するよう構成されている場合には、Dotfuscator はメッセージを送信するコードを生成します）。

Dotfuscator はこのカスタム属性をインストルメンテーションした後、メタデータからこの属性を削除します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

InsertTamperCheckAttribute
`public class InsertTamperCheckAttribute :Attribute`

コンストラクター メンバー

名前	概要
InsertTamperCheckAttribute()	クラスの新しいインスタンスを初期化します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロパティ メンバー

名前	概要
ApplicationNotificationSinkElement :SinkElements	改ざんチェック後、属性付きメソッドが呼び出された時点でアプリケーションに通知するかどうか、またその通知方法を示します。シンク要素には、書き込み可能なブール値フィールド、設定可能なブール値プロパティ、呼び出す void(boolean) メソッドまたはデリゲートを指定できます。このプロパティを使用するには、`ApplicationNotificationSinkName` も設定しておく必要があります。`ApplicationNotificationSinkElement` がフィールド、プロパティ、またはメソッドである場合は、属性付きメソッドと同じクラスでなければ、`ApplicationNotificationSinkOwner` も設定する必要があります。`SinkElement` を "DefaultAction" に設定すると、Dotfuscator は改ざんが検出された場合にアプリケーションを終了するコードを差し込みます。
ApplicationNotificationSinkName :String	改ざんチェック後に設定するプロパティ、フィールドの名前、または呼び出すメソッドの名前。このプロパティを設定した場合、`ApplicationNotificationSinkElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator はアプリケーションに改ざんを通知するコードを生成しません。
ApplicationNotificationSinkOwner :Type	`ApplicationNotificationSinkOwner` は、`ApplicationNotificationSink` のメソッド、フィールド、プロパティ、引数、またはデリゲートを定義する型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きシンク要素が検索されます。
ExtendedKeyMethodArguements : string	実行時、メッセージの拡張キー データに自動的に追加する必要のあるパラメーター名および値を示すパターン。サポートされているパターンの詳細については、「メソッド パラメーターを拡張キーとして自動的に送信する」を参照してください。
ExtendedKeySourceElement :SourceElements	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書にアクセスする方法を示します（たとえば、フィールド、プロパティ、メソッド、またはメソッド引数）。このプロパティを使用するには、`ExtendedKeySourceName` も設定しておく必要があります。`ExtendedKeySourceElement` がフィールド、メソッド、またはプロパティである場合、フィールド、メソッドまたはプロパティが属性付きメソッドと同じクラスで定義されない限り、`ExtendedKeySourceOwner` も定義する必要があります。
ExtendedKeySourceName : string	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書を格納するプロパティ、フィールド、メソッド、またはメソッド引数の名前。このプロパティを設定した場合、`ExtendedKeySourceElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は拡張キー情報を送信するコードを生成しません。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ExtendedKeySourceOwner :Type	`ExtendedKeySourceElement` がフィールド、メソッド、またはプロパティである場合、`ExtendedKeySourceOwner` はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。

2.6.4.1.9 PerformanceProbeAttribute

概要

メソッドの、インストルメンテーション時の属性です。Dotfuscator は、この属性を持つあらゆるメソッドに、Runtime Intelligence のパフォーマンス プローブ メッセージを生成および送信するためのコードを挿入します。Dotfuscator はこのカスタム属性をインストルメンテーションした後、メタデータからこの属性を削除します。

PerformanceProbe Attribute
`public class PerformanceProbeAttribute :Attribute`

コンストラクター メンバー

名前	概要
PerformanceProbeAttribute()	クラスの新しいインスタンスを初期化します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロパティ メンバー

名前	概要
ExtendedKeyMethodArguements : string	実行時、メッセージの拡張キー データに自動的に追加する必要のあるパラメーター名および値を示すパターン。サポートされているパターンの詳細については、「メソッド パラメーターを拡張キーとして自動的に送信する」を参照してください。
ExtendedKeySourceElement :SourceElements	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書にアクセスする方法を示します(たとえば、フィールド、プロパティ、メソッド、またはメソッド引数)。このプロパティを使用するには、`ExtendedKeySourceName` も設定しておく必要があります。`ExtendedKeySourceElement` がフィールド、メソッド、またはプロパティである場合、フィールド、メソッドまたはプロパティが属性付きメソッドと同じクラスで定義されない限り、`ExtendedKeySourceOwner` も定義する必要があります。
ExtendedKeySourceName : string	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書を格納するプロパティ、フィールド、メソッド、またはメソッド引数の名前。このプロパティを設定した場合、`ExtendedKeySourceElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は拡張キー情報を送信するコードを生成しません。
ExtendedKeySourceOwner :Type	`ExtendedKeySourceElement` がフィールドである場合には、`ExtendedKeySourceOwner` はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。
InjectionPoint :InjectionPoints	生成されたコードを差し込む、メソッド内の場所。既定では先頭です。
Name : string	`PerformanceProbeAttribute` のこのインスタンスの名前。これは、実行中に異なる時点で取得したパフォーマンス測定を区別するために使用することができます。

2.6.4.1.9.1 InjectionPoints

概要

生成されたコードを差し込む、メソッド内の位置です。

差し込み位置
`public enum InjectionPoints`

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

列挙メンバー

フィールド	概要
Beginning	メソッドの最初の部分にコードを差し込みます。
End	メソッドの最後の部分にコードを差し込みます。

2.6.4.1.10 SetupAttribute

概要

SetupAttribute はメソッドのインストルメンテーション時の属性です。この属性は、アプリケーション分析のインストルメンテーションの手順をガイドする場合に使用できます。

アプリケーション分析に適用した場合、Dotfuscator はこの属性を持つメソッドに Runtime Intelligence の初期化コードを挿入します。Runtime Intelligence コードを使用するアセンブリまたはアプリケーション内には、この属性を持つメソッドが 1 つ以上ある必要があります。実行時、このメソッドが呼び出されると、初期化コードが Runtime Intelligence のアプリケーションおよびセッションの開始メッセージを送信します。Runtime Intelligence メッセージの送信についてさらに構成するには、開発者は任意で、アプリケーションのインスタンス ID および "opt in" フラグに関する情報を指定できます。Dotfuscator は初期化コードを生成する際に、この情報を使用します。

Runtime Intelligence メッセージのエンドポイントまたは宛先を設定するために、Dotfuscator では、カスタム エンドポイントに無料の Runtime Intelligence Services ポータル、製品版の Runtime Intelligence Services ポータル、またはカスタム エンドポイントのいずれかを選択できます。

インスタンス ID

アプリケーションのインスタンス ID（たとえばシリアル番号）は、通常、アプリケーションの個々のインスタンスに特有のものであり、実行時、アプリケーションによって決定されます。開発者は**`InstanceIdSourceName`**、**`InstanceIdSourceElement`**、および（任意で）**`InstanceIdSourceOwner`** プロパティに値を指定することにより、実行時に Runtime Intelligence 初期化コードでインスタンス ID を使えるようにできます。これらのフィールドは、Runtime Intelligence メッセージの送信でインスタンス ID を利用できるようにするコードを生成するのに使われます。

`InstanceIdSourceElement` には、文字列型のプロパティ、文字列フィールド、文字列を返す引数なしのメソッド、または文字列型のメソッド引数を指定できます。

プロパティまたはフィールドの場合は、書き込み可能で、属性付きメソッドからアクセスでき、かつ文字列型である必要があります。メソッドの場合も、属性付きメソッドからアクセスできる必要があります。プロパティ、メソッド、またはフィールドが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。プロパティ、メソッド、およびフィールドの名前は **`InstanceIdSourceName`** に指定する必要があります。プロパティ、メソッド、またはフィールドの型定義を **`InstanceIdSourceOwner`** で指定する必要があります。**`InstanceIdSourceOwner`** を設定しないでおくと、Dotfuscator は属性付きメソッドの型定義でプロパティ、メソッド、またはフィールドを探します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

InstanceIdSourceElement がメソッド引数である場合は、文字列型の名前付き引数でなければなりません。メソッド引数の名前は **InstanceIdSourceName** に指定する必要があります。**InstanceIdSourceOwner** は使用されません。

利用状況データのオフライン記憶域

Runtime Intelligence を備えたアプリケーションは、ネットワーク アクセスを利用できない状況にある場合、後で接続が復元されたときにデータを転送するよう、利用状況データを格納しておくことができます。利用状況データは分離ストレージに格納されます。この動作は既定値では有効になっており、既定の接続検出コードがインストルメント化されたアプリケーションに差し込まれます。開発者は、`OfflineStateSourceElement` を変更することによって、既定の動作をオーバーライドできます。`OfflineStateSourceElement` 値を **None** に変更すると、アプリケーションがネットワークに接続できない場合、利用状況データは格納されず、削除されるようになります。また開発者は、独自のネットワーク検出コードを書いて、アプリケーションのオフライン状態をインストルメント化されたメソッドのパラメーターのブール値、あるいはメソッドの戻り値として指定したり、フィールドまたはプロパティに指定することにより、ネットワーク接続の状態を Runtime Intelligence コードで利用できるようにすることもできます。これは、`OfflineStateSourceElement` プロパティを適切な値に設定し、`OfflineStateSourceName` および `OfflineStateSourceOwner` を設定することによって行います。

`OfflineStorageResultSinkElement` を介して、オフライン記憶域メカニズムへの利用状況データの格納が成功したか失敗したかを通知することもできます。値が **None** の場合、データ記憶域の成否に関する通知は行われません。値が **DefaultAction** の場合、記憶域メカニズムが利用状況データを格納できなければ、アプリケーションは直ちに終了します。`OfflineStorageResultSinkElement` を適切な値に設定し、`OfflineStorageResultSinkName` および `OfflineStorageResultSinkOwner` を設定することにより、開発者はオフライン記憶域の成否に対処するコードを書くことができます。選択したアプリケーション コードはパラメーター値で呼び出され、ブール型のプロパティまたはフィールドは、一番最近オフライン記憶域メカニズムへ利用状況データの保存を試みた結果によって設定されます。

Opt In

Opt-in フラグは実行時に決定されるもう 1 つの値で、Runtime Intelligence データを送信するためのアプリケーションのアクセス許可が、アプリケーション ユーザーによって与えられているかどうかを示します。開発者は、**OptInSourceName**、**OptInSourceElement**、および（任意で）**OptInSourceOwner** プロパティに値を指定することにより、実行時に Runtime Intelligence 初期化コードで opt-in フラグを使えるようにできます。これらのフィールドは、Runtime Intelligence メッセージの送信者が opt-in 選択を利用できるようにするコードを生成するのに使われます。

OptInSourceElement には、ブール型のプロパティ、ブール型のフィールド、ブール型を返す引数なしのメソッド、またはブール型のメソッド引数を指定できます。

プロパティまたはフィールドの場合は、書き込み可能で、属性付きメソッドからアクセスでき、かつブール型である必要があります。メソッドの場合も、属性付きメソッドからアクセスできる必要があります。プロパティ、メソッド、またはフィールドが静的でない場合は、属性付きメソッドと同じクラスに定義する必要があり、またその属性付きメソッドは静的であってはなりません。プロパティ、メソッド、およびフィールドの名前は **OptInSourceName** に指定する必要があります。プロパティ、メソッド、またはフィールドの型定義を OptInSourceOwner で指定する必要があります。**OptInSourceOwner** を設定しないでおくと、Dotfuscator は属性付きメソッドの型定義でプロパティ、メソッド、またはフィールドを探します。

OptInSourceElement がメソッド引数である場合は、ブール型の名前付き引数でなければなりません。メソッド引数の名前は **OptInSourceName** に指定する必要があります。**OptInSourceOwner** は使用されません。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Dotfuscator はこのカスタム属性をインストルメンテーションした後、メタデータからこの属性を削除します。

SetupAttribute
`public class SetupAttribute :Attribute`

コンストラクター メンバー

名前	概要
SetupAttribute()	クラスの新しいインスタンスを初期化します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロパティ メンバー

名前	概要
CustomEndpoint :String	Runtime Intelligence Services メッセージの URI を定義します。この URI は Runtime Intelligence Services メッセージを生成する Web サービスのエンドポイントを表している必要があります。このプロパティを設定しないと、Runtime Intelligence Services メッセージは PreEmptive の製品版 Runtime Intelligence Services エンドポイントを表す URI に送られます。
ExtendedKeyMethodArguements : string	実行時、メッセージの拡張キー データに自動的に追加する必要のあるパラメーター名および値を示すパターン。サポートされているパターンの詳細については、「メソッド パラメーターを拡張キーとして自動的に送信する」を参照してください。
ExtendedKeySourceElement :SourceElements	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書にアクセスする方法を示します（たとえば、フィールド、プロパティ、メソッド、またはメソッド引数）。このプロパティを使用するには、`ExtendedKeySourceName` も設定しておく必要があります。`ExtendedKeySourceElement` がフィールド、メソッド、またはプロパティである場合、フィールド、メソッドまたはプロパティが属性付きメソッドと同じクラスで定義されない限り、`ExtendedKeySourceOwner` も定義する必要があります。
ExtendedKeySourceName : string	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書を格納するプロパティ、フィールド、メソッド、またはメソッド引数の名前。このプロパティを設定した場合、`ExtendedKeySourceElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は拡張キー情報を送信するコードを生成しません。
ExtendedKeySourceOwner :Type	`ExtendedKeySourceElement` がフィールドである場合には、`ExtendedKeySourceOwner` はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。
InstanceIdSourceElement :SourceElements	実行時、セットアップ メソッドが呼び出された時点でインスタンス ID にアクセスする方法を示します（たとえば、フィールド、プロパティ、またはメソッド引数）。このプロパティを使用するには、`InstanceIdSourceName` も設定しておく必要があります。`InstanceIdSourceElement` がフィールドまたはプロパティである場合、フィールドまたはプロパティが属性付きメソッドと同じクラスで定義されない限り、`InstanceIdSourceOwner` も定義する必要があります。
InstanceIdSourceName :String	実行時、セットアップ メソッドが呼び出された時点でインスタンス ID を格納するプロパティ、フィールド、またはメソッド引数の名前。このプロパティを設定した場合、`InstanceIdSourceElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は Runtime Intelligence メッセージと一緒にインスタンス ID が送信されるコードを生成しません。
InstanceIdSourceOwner :Type	`InstanceIdSourceElement` がフィールドまたはプロパティである場合には、`InstanceIdSourceOwner` はそのフィールドまたはプロパティを定義して

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

	いる型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。
OfflineStateSourceElement :SourceElements	実行時、セットアップ メソッドが呼び出された時点でアプリケーションのオフライン/オンライン状態にアクセスする方法を示します（たとえば、フィールド、プロパティ、またはメソッド引数）。**DefaultAction** と **None** のどちらも選択されていない場合は、`OfflineStateSourceName` も設定しておく必要があります。`OfflineStateSourceElement` がフィールドまたはプロパティの場合、フィールドまたはプロパティが属性付きメソッドと同じクラスで定義されない限り、`OfflineStateSourceOwner` も定義する必要があります。このプロパティの値が **DefaultAction** である場合、分析データのエンドポイントが利用不能な場合には、オフライン記憶域メカニズムによって（分離ストレージを使って）アプリケーションの利用状況データは自動的に格納されます。このプロパティの値が **None** である場合、分析データのエンドポイントが利用不能な場合には、アプリケーションの利用状況データは格納されません。
OfflineStateSourceName :String	実行時、セットアップ メソッドが呼び出された時点でアプリケーションのオフライン状態を格納するプロパティ、フィールド、またはメソッド引数の名前。このプロパティを設定した場合、`OfflineStateSourceElement` の設定も必要になります。
OfflineStateSourceOwner :Type	`OfflineStateSourceElement` がフィールドまたはプロパティである場合には、`OfflineStateSourceOwner` はそのフィールドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。
OfflineStorageResultSinkElement :SourceElements	利用状況データがオフライン記憶域メカニズムに格納されたときに、アプリケーションへ通知する方法を示します。このプロパティが DefaultAction でも None でもない場合は、`OfflineStorageResultSinkName` も設定しておく必要があります。`OfflineStorageResultSinkElement` がフィールドまたはプロパティの場合、フィールドまたはプロパティが属性付きメソッドと同じクラスで定義されない限り、`OfflineStorageResultSinkOwner` も定義する必要があります。このプロパティの値が **None** である場合は、アプリケーションの利用状況データがオフライン記憶域メカニズムに格納されても、通知は行われません。このプロパティの値が DefaultAction である場合は、記憶域のエラーが発生すると、アプリケーションは直ちに終了します。
OfflineStorageResultSinkName :String	アプリケーションの利用状況データをオフライン記憶域メカニズムに格納した、一番最近の結果を格納するプロパティ、フィールド、またはメソッド引数の名前。このプロパティを設定した場合、`OfflineStorageResultSinkElement` の設定も必要になります。
OfflineStorageResultSinkOwner :Type	`OfflineStorageResultSinkElement` がフィールドまたはプロパティである場合には、`OfflineStorageResultSinkOwner` はそのフィールドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。
OptInSourceElement	実行時、セットアップ メソッドが呼び出された時点で opt-in フラグにアクセスす

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

:SourceElements	る方法を示します（たとえば、フィールド、プロパティ、またはメソッド引数）。このプロパティを使用するには、`OptInSourceName` も設定しておく必要があります。`OptInSourceElement` がフィールドまたはプロパティの場合、フィールドまたはプロパティが属性付きメソッドと同じクラスで定義されない限り、`OptInSourceOwner` も定義する必要があります。
OptInSourceName :String	実行時、セットアップ メソッドが呼び出された時点で opt-in フラグを格納するプロパティ、フィールド、またはメソッド引数の名前。このプロパティを設定した場合、`OptInSourceElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は常に Runtime Intelligence メッセージが送信されるコードを生成します。
OptInSourceOwner :Type	`OptInSourceElement`OptInSourceOwner はそのフィールドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。
UseSSL :Boolean	Runtime Intelligence メッセージを Runtime Intelligence Service へ送信するとき、HTTPS プロトコルを使用します。既定値は true です。

2.6.4.1.11 SinkElements

概要

データのシンク候補は、インストルメント化されたコードが提供することになっています。用例は、「InsertTamperCheckAttribute」のアプリケーション改ざん通知に記述されています。

SinkElements
`public enum SinkElements :Attribute`

列挙メンバー

フィールド	概要
フィールド	シンク要素はフィールドです。
MethodArgument	シンク要素はデリゲート型のメソッド引数です。
Method	シンク要素は呼び出されるメソッドです。
None	シンク要素がありません。シンク要素が指定されていない場合、個々の属性は既定の操作の動作を実装する可能性があります。詳細については、属性のドキュメントを参照してください。
プロパティ	シンク要素は設定可能なプロパティです。
Delegate	シンク要素はデリゲート型のフィールドです。
DefaultAction	シンクが用供されたとき、既定の操作を実行します。個々の属性にはそれぞれ既定の操作があります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.6.4.1.12 SourceElements

概要

データのソース候補は、Runtime Intelligence で生成されたコードが使用することになっています。用例は、SetupAttribute のアプリケーション インスタンス ID および opt-in フラグに記述されています。

SourceElements
`public enum SourceElements :Attribute`

列挙メンバー

フィールド	概要
フィールド	ソース要素はフィールドです。
Method	ソース要素は呼び出されるメソッドです。
MethodArgument	ソース要素はメソッド引数です。
None	ソース要素がありません。これは既定値です。
プロパティ	ソース要素はプロパティです。

2.6.4.1.13 SystemProfileAttribute

概要

SystemProfileAttribute はメソッドのインストルメンテーション時の属性です。Dotfuscator は、この属性を持つあらゆるメソッドに、Runtime Intelligence のプロファイル メッセージを生成および送信するためのコードを挿入します。Dotfuscator はこのカスタム属性をインストルメンテーションした後、メタデータからこの属性を削除します。この属性には、すべての Runtime Intelligence 属性が持っている、オプションの **`ExtendedKeysSource`** プロパティ外にプロパティはありません。

SystemProfile Attribute
`public class SystemProfileAttribute :Attribute`

コンストラクター メンバー

名前	概要
SystemProfileAttribute()	クラスの新しいインスタンスを初期化します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロパティ メンバー

名前	概要
ExtendedKeyMethodArguements : string	実行時、メッセージの拡張キー データに自動的に追加する必要のあるパラメーター名および値を示すパターン。サポートされているパターンの詳細については、「メソッド パラメーターを拡張キーとして自動的に送信する」を参照してください。
ExtendedKeySourceElement :SourceElements	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書にアクセスする方法を示します(たとえば、フィールド、プロパティ、メソッド、またはメソッド引数)。このプロパティを使用するには、`ExtendedKeySourceName` も設定しておく必要があります。`ExtendedKeySourceElement` がフィールド、メソッド、またはプロパティである場合、フィールド、メソッドまたはプロパティが属性付きメソッドと同じクラスで定義されない限り、`ExtendedKeySourceOwner` も定義する必要があります。
ExtendedKeySourceName : string	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書を格納するプロパティ、フィールド、メソッド、またはメソッド引数の名前。このプロパティを設定した場合、`ExtendedKeySourceElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は拡張キー情報を送信するコードを生成しません。
ExtendedKeySourceOwner :Type	`ExtendedKeySourceElement` がフィールドである場合には、`ExtendedKeySourceOwner` はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。

2.6.4.1.14 TeardownAttribute

概要

TeardownAttribute はメソッドの、インストルメンテーション時の属性です。この属性は、アプリケーション分析のインストルメンテーションの手順をガイドする場合に使用できます。

アプリケーション分析で使用されている場合、Dotfuscator はこの属性を持つあらゆるメソッドに Runtime Intelligence のクリーンアップ コードを挿入します。Runtime Intelligence のインストルメンテーションを使用するアセンブリまたはアプリケーション内には、この属性を持つメソッドが 1 つ以上ある必要があります。実行時、このメソッドが呼び出されると、クリーンアップ コードが Runtime Intelligence のアプリケーションおよびセッションの停止メッセージを送信します。

Dotfuscator はこのカスタム属性をインストルメンテーションした後、メタデータからこの属性を削除します。

TeardownAttribute
`public class TeardownAttribute :Attribute`

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

コンストラクター メンバー

名前	概要
TeardownAttribute()	クラスの新しいインスタンスを初期化します。

プロパティ メンバー

名前	概要
ExtendedKeyMethodArguements : string	実行時、メッセージの拡張キー データに自動的に追加する必要のあるパラメーター名および値を示すパターン。サポートされているパターンの詳細については、「メソッド パラメーターを拡張キーとして自動的に送信する」を参照してください。
ExtendedKeySourceElement :SourceElements	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書にアクセスする方法を示します(たとえば、フィールド、プロパティ、メソッド、またはメソッド引数)。このプロパティを使用するには、`ExtendedKeySourceName` も設定しておく必要があります。`ExtendedKeySourceElement` がフィールド、メソッド、またはプロパティである場合、フィールド、メソッドまたはプロパティが属性付きメソッドと同じクラスで定義されない限り、`ExtendedKeySourceOwner` も定義する必要があります。
ExtendedKeySourceName : string	実行時、属性付きメソッドが呼び出された時点で拡張キーの辞書を格納するプロパティ、フィールド、メソッド、またはメソッド引数の名前。このプロパティを設定した場合、`ExtendedKeySourceElement` の設定も必要になります。このプロパティが設定されていない場合、Dotfuscator は拡張キー情報を送信するコードを生成しません。
ExtendedKeySourceOwner :Type	`ExtendedKeySourceElement` がフィールドである場合には、`ExtendedKeySourceOwner` はそのフィールド、メソッドまたはプロパティを定義している型の名前を示します。設定されていない場合は、属性付きメソッドの型で名前付きソース要素が検索されます。

2.6.5 割り当てファイル

Dotfuscator は古い名前と新しい名前を関連付けるために割り当てファイルを生成します。割り当てファイルには、クラス、メソッド、およびフィールドの新しい名前が示されています。名前を変更した後は、バグの追跡が困難になります。特に、メソッドのオーバーロードが非常に高い確率で発生するので、割り当てファイルが不可欠になります。

割り当てファイルは、増分難読化で使用するほか、難読化されたスタック トレースのデコードにも使用できます。また、割り当てファイルは、名前の変更の全体的な効果に関する統計を提供します。

割り当てファイルの要素はいずれも非常に単純です。以下に、注意する必要のある点を示します。

- **newname** 要素がない場合、その項目の名前は変更されません。
- 型名では、入れ子になったクラスの名前は "**/**" 文字で親と区切られます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

- コンストラクターには "`.ctor`" という名前が付けられますが、(静的初期化子、クラス コンストラクターなどとも呼ばれる) 静的コンストラクターには "`.cctor`" という名前が付けられます。これらの名前が変更されることはありません。

追加情報については、「割り当てファイルの DTD 管理」を参照してください。

2.6.5.1 dotfuscatorMap_v1.1.dtd

dotfuscatorMap_v1.1.dtd は、Dotfuscator によって生成、使用される名前変更割り当てファイルのフォーマットが記述されている DTD (文書型定義) ファイルです。このファイルは、www.preemptive.com/dotfuscator/dtd/dotfuscatorMap_v1.1.dtd で参照できます。

2.6.6 高度なトピック

以下のセクションでは、Dotfuscator の高度な使用法について詳しく説明します。

2.6.6.1 サイド バイ サイド インストール

Dotfuscator はバージョン 4.11 以上から、複数のバージョンをサイド バイ サイドでインストールすることができます。これを有効にするには、インストーラー ウィザードで[詳細]ボタンをクリックして、[現在のインストールとサイド バイ サイドでインストールする]をオンにします。また、Dotfuscator プロジェクトの既定として、どのバージョンの Dotfuscator が Visual Studio で使用されるかを構成することもできます。インストールするバージョンを "最新の Major"、"最新の Major.Minor"、または特定のバージョンから選択できます。既定値は、後で HKCU\Software\PreEmptive\Dotfuscator\DefaultMSBuildPath にあるレジストリ キーを編集することによって変更できます。

MSBuild 拡張

最新のバージョンをアンインストールした場合、major および major.minor ディレクトリ内の MSBuild タスクおよびターゲット ファイルは残ります。major または major.minor のファイルを前のバージョンに戻すには、特定のバージョンのサブディレクトリからそれらのファイルをコピーするか、またはファイルを手動で削除してから、[プログラムの追加と削除]を使用して元に戻したいバージョンで修復を実行します。

Visual Studio 統合

サイド バイ サイド インストールを実行する際、現在 Visual Studio に統合されている Dotfuscator を上書きしないように、Visual Studio の統合機能を無効にすることができます。Visual Studio に統合されている Dotfuscator のバージョンをアンインストールし、統合するバージョンを別のバージョンに戻したい場合は、[プログラムの追加と削除]を開いて、目的のバージョンの Dotfuscator で[変更]を選択し、まず統合を無効にして完了した後、機能を有効にして変更を再実行します。

ここで記述した手順を使用すると、複数バージョンの Dotfuscator をコマンド ラインから、MSBuild から、またスタンドアロンの Dotfuscator UI として実行できるようになります。ただし、Visual Studio と同時に統合できるのは 1 つのバージョンの Dotfuscator だけであるということに留意してください。Dotfuscator の Visual Studio プロジェクトは MSBuild と互換性があるため、1 つの .dotfuproj ファイルを両方のコンテキストでの実行に使用できます。

ただし、Visual Studio 統合は、DotfuscatorBinPath プロパティで指定されるバージョンの Dotfuscator を使用することができませ

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ん。Visual Studio 統合は常に、最近インストールされたバージョンの Dotfuscator を使用してビルドを行います。DotfuscatorBinPath が既定値でない場合は、"この Dotfuscator プロジェクトの DotfuscatorBinPath または DotfuscatorDataPath は、インストールされている Visual Studio Dotfuscator 拡張とは異なっています。" という警告を受けます。

2.6.6.2 並行ビルド

Dotfuscator の複数のコピーを同時に実行する場合は、次のファイルを Dotfuscator の実行可能ファイルと同じディレクトリにコピーすることをお勧めします。

`dotfuscator.dat`(通常は `%PROGRAMDATA%\PreEmptive Solutions\Dotfuscator [Edition]\4.0` にあります)
`dfusrprf.xml` (通常は `%LOCALAPPDATA%\PreEmptive Solutions\Dotfuscator [Edition]\4.0` にあります)
`dotfuscator.cfg`(通常は `%LOCALAPPDATA%\PreEmptive Solutions\Dotfuscator [Edition]\4.0` にあります)

2.7 サンプル

このセクションでは、各種の .NET アプリケーション用に Dotfuscator を設定する方法を示す目的で設計された、追加のサンプル アプリケーションについて説明します。

簡単な例を使用して Dotfuscator の基本的な使用法を短時間で習得するには、「Dotfuscator の使用」を参照してください。

このセクションのサンプルは C# を使用しています。VB.NET を使用したサンプルは、当社の Web サイトおよび Dotfuscator インストール ディレクトリから入手できます。これらのサンプルがインストール プログラムによって、Dotfuscator インストール ディレクトリに保存されていない場合、サンプル ファイルは www.preemptive.com/dotfuscator-samples.html から無料でダウンロードできます。

2.7.1 リフレクションのサンプル

リフレクションのサンプルでは、動的なクラス読み込みやメソッド呼び出しを利用するアプリケーションと共に Dotfuscator を使用するときに発生する問題を説明します。これらの動的なクラス読み込みやメソッド呼び出しなどの強力なテクノロジを利用して作成されたアプリケーションでは、実行するコードの特定を実行時まで遅延させることができます。このような場合、実行時に呼び出されるクラスやメソッドを静的に予測することは不可能です。したがって、Dotfuscator が名前変更の対象から除外すべき識別子を確実に判断することは不可能です。ただし、このような問題に対処するために、Dotfuscator には名前の変更規則を微調整する高度な機能が用意されています。

リフレクションのサンプルでは、型やメソッドを指定して要素を名前の変更対象から除外するには、Dotfuscator をどのように構成するかについて説明します。サンプルには、Dotfuscator のコマンド ラインを使用した場合の例と、グラフィカル インターフェイスを使用した場合の例が紹介されています。

2.7.1.1 リフレクションのサンプル ファイル

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

リフレクションのサンプルには以下のファイルが含まれています。

ファイル	説明
Reflection.doc	このドキュメント
Reflection.cs	リフレクション技術を例示する C# ファイル
reflection_config.xml	Dotfuscator 構成ファイル
make.bat	リフレクション アプリケーションのコンパイル用バッチ ファイル
run.bat	アプリケーションで Dotfuscator を実行するためのバッチ ファイル

リフレクションのサンプル ファイルは Dotfuscator のインストール フォルダーに存在します。また Dotfuscator Samples Download ZIP を www.preemptive.com/dotfuscator-samples.html（英語）からダウンロードできます。

2.7.1.2 リフレクション サンプルのビルド

リフレクション サンプルは、ユーザーの PATH 環境変数から C# コンパイラ（**csc.exe**）にアクセスできることを前提にしています。Visual Studio を使用している場合は、Windows の［スタート］ボタンをクリックして、［プログラム］、［Microsoft Visual Studio］、［Visual Studio ツール］の順にポイントし、［**Visual Studio コマンド プロンプト**］ショートカットを実行することでこれを確認できます。

コマンド プロンプトで、カレント ディレクトリをリフレクション サンプルを含むディレクトリに変更します。

make.bat を実行してアプリケーションをビルドします。このバッチ ファイルは Visual C# コンパイラを呼び出して、出力ファイル **Reflection.exe** を生成します。

2.7.1.3 リフレクション サンプルの実行

make.bat コマンドで生成した Reflection.exe アセンブリを実行することで、リフレクション サンプルを実行できます。リフレクション プログラムは、次のコードを使用して、現在のアセンブリから動的にクラスを読み込みます。

リフレクション サンプル コード

```
//現在のアセンブリから要求された型を取得します
assembly = this.GetType().Assembly;
type = assembly.GetType(typename, true);
instance = Activator.CreateInstance(type);
```

読み込むクラスを文字列型の変数 typename で指定していることに注意してください。サンプル コードを詳しく見ると、**`typename`** を値 **`Samples.Greeting`** で初期化していることがわかります。

プログラムをさらに見ていくと、**`Samples.Greeting`** の **`SayGreetings`** メソッドが同様のコードで直接起動されています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Sample.Greeting コード

```
MethodInfo method = type.GetMethod(methodname);
…
method.Invoke(instance,methodargs);
```

SayGreetings メソッドは、その名の通り、コンソールに挨拶のメッセージを表示します。

SayGreetings 友好的なメッセージ

```
Hello Bob!
Goodbye Bob!
```

2.7.1.4 Dotfuscator によるリフレクション出力の処理

リフレクション サンプルには、サンプルの Dotfuscator 構成ファイルが含まれています。このファイルは、対象除外規則を使用して、リフレクションによって呼び出された項目を対象から除外します。このファイルは **reflection_config.xml** という名前で、リフレクションのサンプルの他のファイルと同じディレクトリに格納されています。このファイルでは、参照される項目を対象から除外するセクションは次のようになっています。

対象除外規則を使って、リフレクションによって呼び出された項目を対象から除外する

```
<renaming>
   <excludelist>
      <type name="Samples.Greetings">
         <method name="SayGreetings" signature="string" />
      </type>
   </excludelist>
…
</renaming>
```

<renaming> タグは、そのタグ内の対象除外規則が、具体的な識別子の名前変更に関連するものであることを示しています。これは、Dotfuscator の他の機能では有効化/無効化を選択できる点とは対照的です。

<excludelist> タグは、名前の変更処理対象から除外する必要のある項目のリストを定義します。**<type name="Samples.Greetings">** タグは、名前の変更処理対象からクラス名 "**Samples.Greetings** を除外することを指示します。なお、このタグで参照されるのはクラス名そのものである点に注意してください。"**Greetings**" クラスに属するすべてのメソッドとフィールドは、明示的に対象から除外しない限り、名前の変更対象となります。メソッドを明示的に対象から除外している例は、**<method name="SayGreetings" signature="string" />** で示されています。

make.bat ファイルを実行すると、この構成ファイルを使用して Dotfuscator が実行されます。この処理の結果、"**output**" サブディレクトリに Reflection.exe アセンブリが出力されます。出力先は、構成ファイルの次のセクションを変更することで制御できます。

出力先を変更する

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<output>
 <file dir="${projectdir}\output" />
</output>
```

新しいアセンブリを実行すると、Dotfuscator が必要な項目を名前の変更処理対象から正しく除外していることを確認できます。

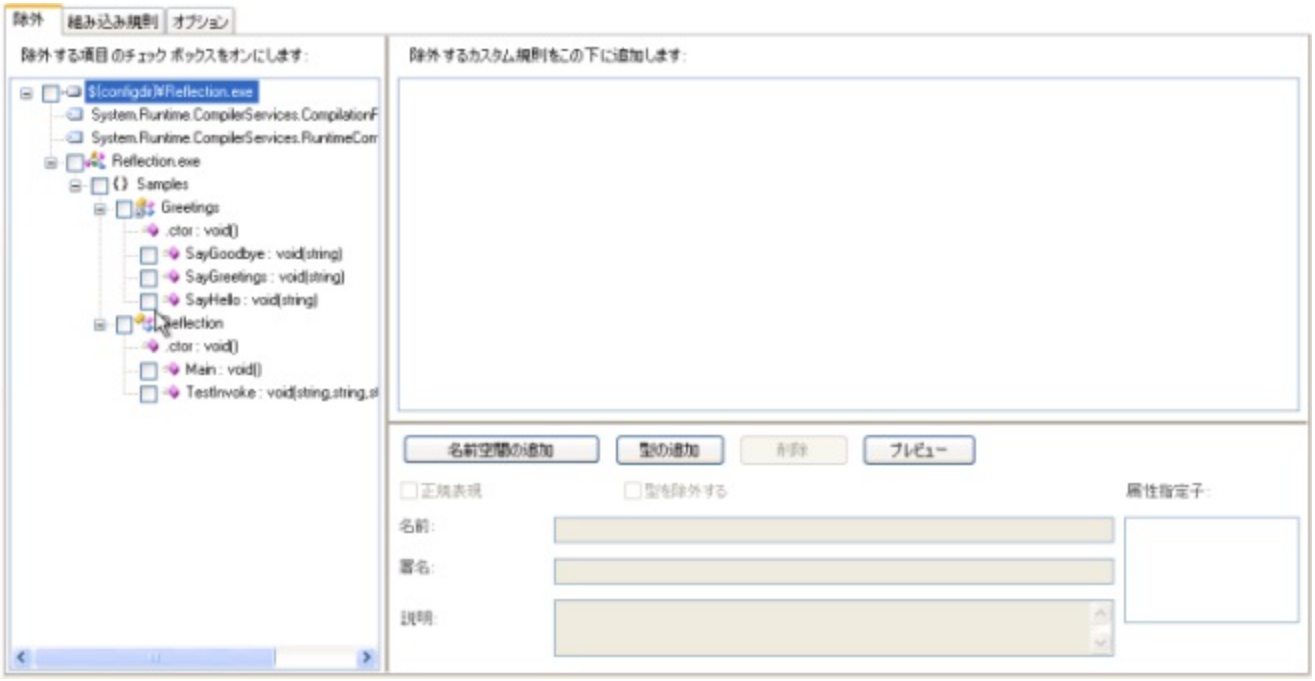
名前の変更処理から除外された、友好的なメッセージ
Hello Bob! Goodbye Bob!

2.7.1.5 グラフィカル ユーザー インターフェイスを使用したリフレクション サンプルの構成

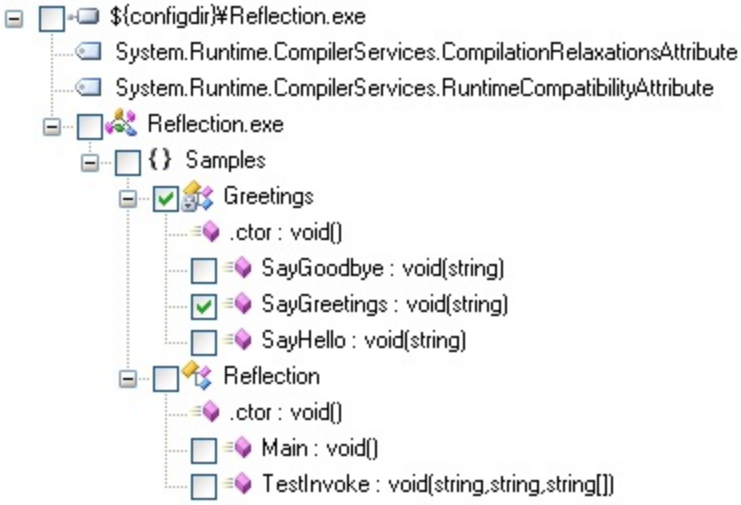
Dotfuscator のグラフィカル インターフェイスを使用して、構成ファイルをビジュアルに生成できます。名前の変更対象から除外する項目は、インターフェイスの[**名前の変更**]タブを使用して指定できます。

ツリーのアセンブリ ノードを展開すると、アプリケーション構造がグラフィカルに表示されます。この構造には、すべての名前空間、型、およびメソッドが含まれています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

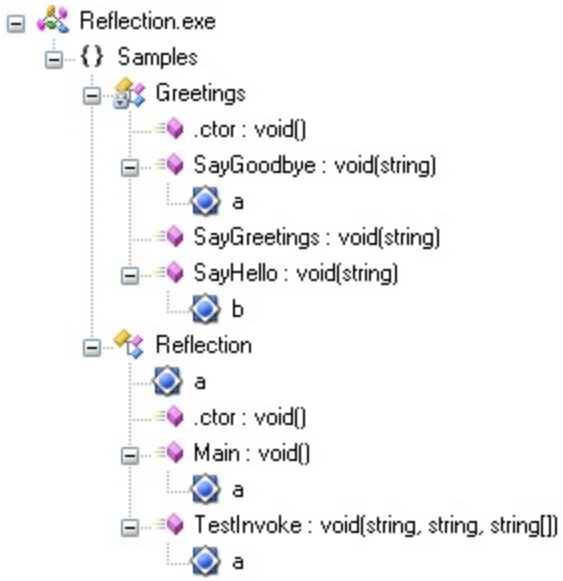


除外対象の項目の隣にあるチェック ボックスをオンにするだけの簡単な操作で、グラフィカルに名前の変更の対象除外リストを生成できます。このリフレクション サンプルでは、型 "**`Greetings`**" とメソッド "**`SayGreetings`**" の隣にあるチェック ボックスをオンにして、必要な除外を行っています。



プロジェクトをビルドすると、アプリケーションの [**出力**] タブで正しい出力が作成されたことを確認できます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.7.1.6 リフレクション サンプルのまとめ

名前によってクラスを読み込んだり、名前によってメソッドを呼び出したり、名前によってフィールドを参照したりするようなアプリケーションを Dotfuscator で正常に処理するには、適切な識別子を名前の変更対象から手作業で除外する必要があります。この作業を行うために、Dotfuscator には、規則に基づいた微調整可能な機能が用意されています。

2.7.2 シリアル化のサンプル

シリアル化のサンプルでは、利用しているシリアル化されたオブジェクトを難読化されていないコードと交換する必要のあるアプリケーションを Dotfuscator で使用する方法を例示します。難読化されたアプリケーションが、シリアル化されたオブジェクトを作成するだけ、もしくは利用するだけの場合、このサンプルは適用されません。

Dotfuscator を使用するときにはシリアル化を考慮する必要があります。これは、提供されているすべてのシリアル化フォーマッタは、保持しているデータ ストリームに型とフィールドの情報を埋め込むからです。このことは、シリアル化するクラスおよびフィールドは名前の変更対象から除外する必要があり、対象から除外しない場合は難読化されていないコードを保持しているオブジェクトをシリアル化解除できないことを暗黙のうちに示しています。メソッドの定義は保持されないので、それらを名前の変更対象から除外する必要はありません。

シリアル化のサンプルでは、名前の変更対象から型やフィールドを選択的に除外するための Dotfuscator の構成方法について例示します。サンプルには、コマンド ラインを使用した場合の例と、グラフィカル インターフェイスを使用した場合の例が紹介されています。

2.7.2.1 シリアル化のサンプル ファイル

シリアル化のサンプルには以下のファイルが含まれています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ファイル	説明
Serialization.doc	このドキュメント
Serialization.cs	シリアル化技術を示す C# ファイル
serialization_config.xml	Dotfuscator 構成ファイル
make.bat	シリアル化アプリケーションをコンパイルするバッチ ファイル
run.bat	アプリケーションで Dotfuscator を実行するためのバッチ ファイル

シリアル化のサンプル ファイルは Dotfuscator のインストール フォルダーに存在します。また Dotfuscator Samples Download ZIP を www.preemptive.com/dotfuscator-samples.html（英語）からダウンロードできます。

2.7.2.2 シリアル化サンプルのビルド

シリアル化サンプルは、ユーザーの PATH 環境変数から C# コンパイラ（**csc.exe**）にアクセスできることを前提にしています。Visual Studio を使用している場合は、Windows の［スタート］ボタンをクリックして、［プログラム］、［Microsoft Visual Studio］、［Visual Studio ツール］の順にポイントし、［**Visual Studio コマンド プロンプト**］ショートカットを実行することでこれを確認できます。

コマンド プロンプトで、カレント ディレクトリをシリアル化サンプルを含むディレクトリに変更します。

make.bat を実行してアプリケーションをビルドします。このバッチ ファイルは Visual C# コンパイラを呼び出して、出力ファイル **Serialization.exe** を生成します。

2.7.2.3 シリアル化サンプルの実行

make.bat コマンドで生成した Serialization.exe アセンブリを実行することで、シリアル化サンプルを実行できます。アプリケーションには、コマンド ライン パラメーターで制御される 2 つのモードがあります。"**-s**" オプションを使用すると、アプリケーションが "**Tester**" 型のオブジェクトのインスタンスを作成して、ファイル "**Sum.out**" に保存します。それぞれの Tester オブジェクトは、保存しているデータを構成する 3 つの整数のフィールドを持ちます。オブジェクトの保持は、以下の Tester クラスのコードで行います。

Tester クラス コード

```
// create a file stream to write the file
FileStream stream = new fileStream("Sum.out",FileMode.Create);

BinaryFormatter formatter = new BinaryFormatter();
// serialize to disk
formatter.Serialize(stream,this);
stream.Close();
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

"**-d**" オプションを指定して実行すると、以前ファイル "**Sum.out**" に保存したオブジェクトのシリアル化が解除されます。次のコードでシリアル化の解除を行っています。

シリアル化解除のコード

```
stream = new FileStream("Sum.out",FileMode.Open);
BinaryFormatter formatter = new BinaryFormatter();
return (Tester) formatter.Deserialize(stream);
```

2.7.2.4 Dotfuscator によるシリアル化出力の処理

シリアル化のサンプルに含まれるサンプルの Dotfuscator 構成ファイルでは、対象除外規則を使用して、オブジェクトの持続的表現を構成する情報を対象から除外する例を示しています。このファイルは **serialization_config.xml** という名前で、シリアル化のサンプルの他のファイルと同じディレクトリに格納されています。このファイルでは、参照される項目を対象から除外するセクションは次のようになっています。

シリアル化サンプルのリファレンス ファイル

```
<renaming>
<excludelist>
  <type name="Samples.Tester">
   <field name=".*" regex="true" />
  </type>
</excludelist>
…
</renaming>
```

<renaming> タグは、そのタグ内の対象除外規則が、具体的な識別子の名前変更に関連するものであることを示しています。これは、Dotfuscator の他の機能では有効化/無効化を選択できる点とは対照的です。

<excludelist> タグは、名前の変更処理対象から除外する必要のある項目のリストを定義します。**<type name="Samples.Tester">** タグは、名前の変更処理対象からクラス名 "**Samples.Tester**" を除外することを Dotfuscator に指示します。なお、このタグで参照されるのはクラス名そのものである点に注意してください。"**Tester**" に属するすべてのメソッドは名前の変更対象となります。**<field name=".*" regex="true" />** タグは Dotfuscator に、Tester クラスに含まれるすべてのフィールドを対象から除外することを指示します。大規模なクラスになると煩雑になるので各フィールドを個別に呼び出すことはせず、この例では正規表現を使用して対象から除外するフィールドを指定します。この例では、すべてのフィールドと一致する単純な表現 ".*****" を使用します。

make.bat ファイルを実行すると、この構成ファイルを使用して Dotfuscator が実行されます。この処理の結果、"output" サブディレクトリに **Serialization.exe** アセンブリが出力されます。出力先は、構成ファイルの次のセクションを変更することで制御できます。

アセンブリの出力

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<output>
<file dir="${projectdir}\output" />
</output>
```

新しいアセンブリを実行すると、Dotfuscator が、必要な項目を名前の変更処理対象から正しく除外していることを確認できます。また、難読化されたアプリケーションが、難読化されていないアプリケーションにより保存されるオブジェクトを正しくシリアル化解除でき、難読化されていないアプリケーションが、難読化されたアプリケーションにより保存されるオブジェクトをシリアル化解除できることにも注意してください。シリアル化されたオブジェクトは相互に完全に互換性があります。

2.7.2.5 グラフィカル ユーザー インターフェイスを使用したシリアル化のサンプルの構成

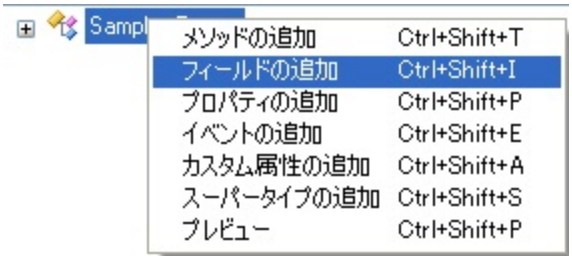
Dotfuscator のグラフィカル インターフェイスを使用して、構成ファイルをビジュアルに生成できます。名前の変更対象から除外する項目は、インターフェイスの[**名前の変更**]タブを使用して指定できます。ツリーのアセンブリ ノードを展開すると、アプリケーション構造がグラフィカルに表示されます。この構造には、すべての名前空間、型、およびメソッドが含まれています。

名前の変更の対象除外リストをグラフィカルに生成するには、インターフェイスの正規表現の機能を使用し、2 段階で処理を行います。まず、型 "**`Tester`**" に一致する型の除外規則を作成します。これを行うには、[**型の追加**]ボタンをクリックします。**Type** という名前の新しいノードが右側のウィンドウに追加されます。

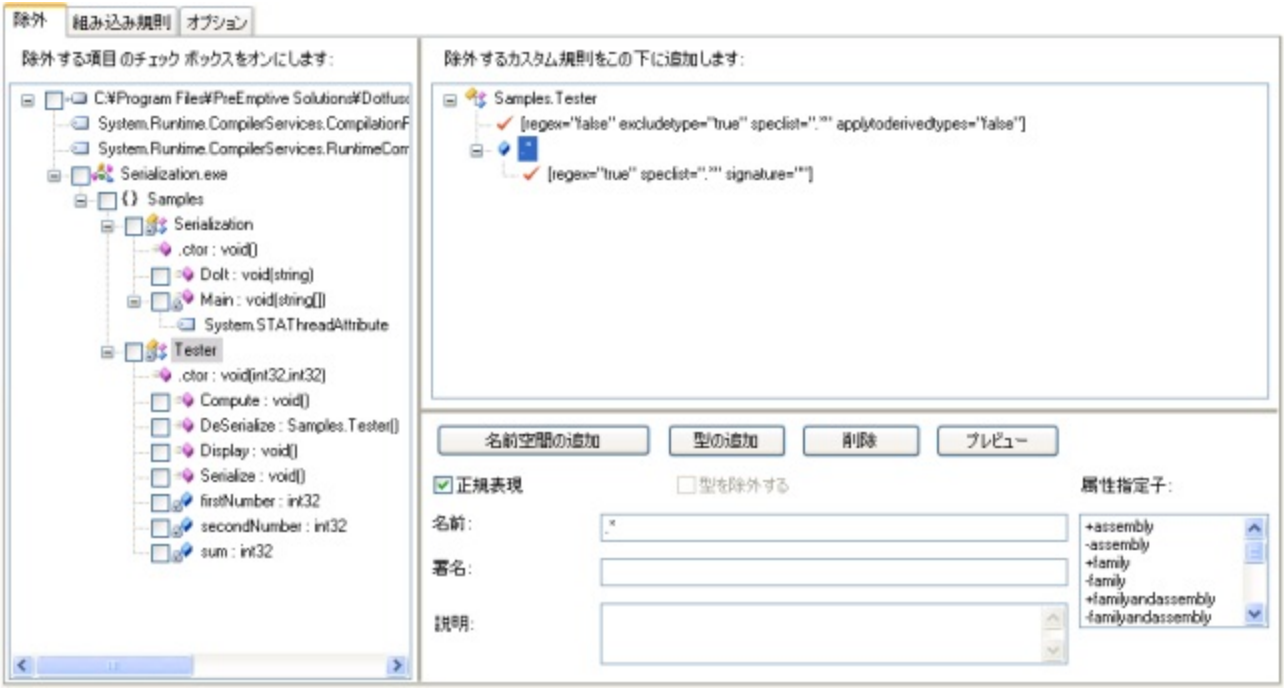
次に、この新しいノードのフィールド名を **Samples.Tester** に変更して、対象から除外するクラスを反映させます。対象から除外するのは 1 つのクラスだけなので、[正規表現] オプションの選択を解除します。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

フィールドの対象除外規則を追加するには、新しく追加された **Samples.Tester** ノードを右クリックします。これにより、ショートカット メニューが表示されます。



[**フィールドの追加**]をクリックすると、新しいノードがツリーに追加されます。名前を "**.***" に変更すると、すべてのフィールドと一致することになります。

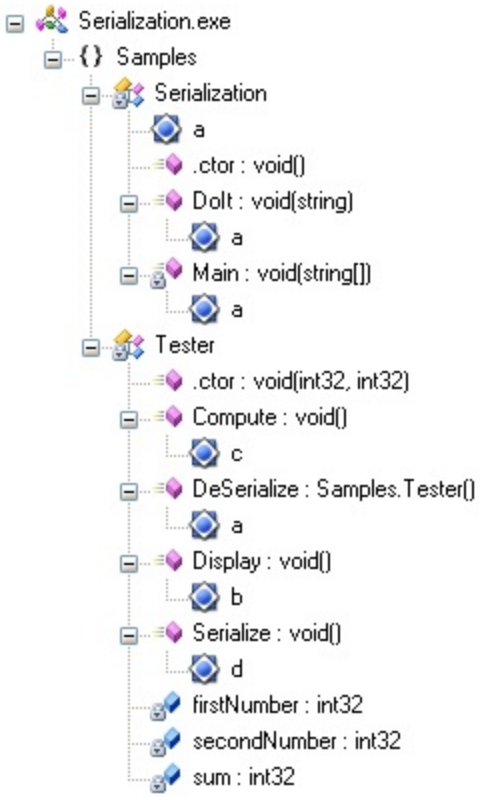


© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

[**プレビュー**]をクリックすると規則が適用されて、対象から除外される項目が左側のツリーに網掛け表示されます。**Samples.Tester** のすべてのフィールドが対象から除外され、期待どおりの結果になります。

プロジェクトをビルドすると、アプリケーションの [出力] タブで正しい出力が作成されたことを確認できます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.7.2.6 シリアル化サンプルのまとめ

シリアル化したオブジェクトを外部のアプリケーションと交換するアプリケーションを Dotfuscator で処理するには、名前の変更処理対象から適切なデータ要素を除外する必要があります。Dotfuscator は、このような対象除外規則を定義する非常に強力な機構を提供します。これらのガイドラインに従うことで、Dotfuscator で処理したアプリケーションが Dotfuscator で処理されていないアプリケーションとも安全かつ容易にデータを交換できることが確実になります。

2.7.3 リモートのサンプル

リモートのサンプルでは、.NET Remoting を利用するアプリケーションで Dotfuscator を使用する例を示します。従来のリモート アプリケーションの場合、リモート呼び出しを提供するオブジェクトは、呼び出される前にランタイムに登録されます。ランタイムは呼び出し要求を受けると、リフレクションを使用してサーバー オブジェクトのインスタンスを動的に作成します。この処理は実行時に行われるので、Dotfuscator が対象から除外する必要がある型を分析時に認識することはできません。Dotfuscator がリモートで使用する型の名前を変更しないように、ユーザーが手動で操作する必要があります。

リモートのサンプルでは、任意の型を名前の変更対象から除外するには、Dotfuscator をどのように構成するかについて説明します。このサンプルでは、同じ Dotfuscator プロジェクトに複数のアセンブリを設定する方法も例示しています。サンプルには、Dotfuscator のコマンド ラインを使用した場合の例と、グラフィカル インターフェイスを使用した場合の例が紹介されています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.7.3.1 リモートのサンプル ファイル

リモートのサンプルには以下のファイルが含まれています。

ファイル	説明
Remoting.doc	このドキュメント
TrigServer.cs	リモート サーバー用のコードを含んでいるファイル
TrigClient.cs	リモート クライアント用のコードを含んでいるファイル
ITrigFunctions.cs	共通インターフェイスを含んでいるファイル
remoting_config.xml	Dotfuscator 構成ファイル
make.bat	リモート アプリケーションのコンパイル用バッチ ファイル
run.bat	アプリケーションで Dotfuscator を実行するためのバッチ ファイル

リモートのサンプル ファイルは Dotfuscator のインストール フォルダーに存在します。また Dotfuscator Samples Download ZIP を www.preemptive.com/dotfuscator-samples.html（英語）からダウンロードできます。

2.7.3.2 リモート サンプルのビルド

リモートのサンプルは、ユーザーの PATH 環境変数から C# コンパイラ（**csc.exe**）にアクセスできることを前提にしています。Visual Studio を使用している場合は、Windows の［スタート］ボタンをクリックして、［プログラム］、［Microsoft Visual Studio］、［Visual Studio ツール］の順にポイントし、［**Visual Studio コマンド プロンプト**］ショートカットを実行することでこれを確認できます。

コマンド プロンプトで、カレント ディレクトリをリモートのサンプルを含むディレクトリに変更します。

make.bat を実行してアプリケーションをビルドします。このバッチ ファイルは Visual C# コンパイラを呼び出して、出力ファイル **TrigServer.exe**、**TrigClient.exe**、および **ITrigFunctions.dll** を生成します。

2.7.3.3 リモート サンプルの実行

リモートのサンプルを実行するには、まず **TrigServer.exe** アセンブリを起動します。このアプリケーションはポート 12345 に新しい HTTP チャネル リスナーを登録し、そのチャネルでのメソッド呼び出しを処理するクラス **`Samples.Trig`** を登録します。以下は、これを行う TrigServer のコードです。

TrigServer のコード

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
// チャネルを作成しリッスンを開始します
HttpChannel serverchannel = new HttpChannel(12345);
ChannelServices.RegisterChannel(serverchannel);

Type trigType = Type.GetType("Samples.Trig");

// 型を登録します
RemotingConfiguration.RegisterWellKnownServiceType(
   trigType,
   "TrigServer",
   WellKnownObjectMode.Singleton );
```

サーバーが確立されたら、クライアント アプリケーション **TrigClient.exe** を実行できます。このアプリケーションは次のコードに示すように、発信用 HTTP チャネルを作成して TrigServer に接続します。

TrigServer への接続のコード

```
// クライアント呼び出し用のチャネルを作成します
HttpChannel clientchannel = new HttpChannel(0);
ChannelServices.RegisterChannel(clientchannel);

// リモートの Trig server への参照を取得します
MarshalByRefObject rawobject =
   (MarshalByRefObject)RemotingServices.Connect(
      typeof(Samples.ITrigFunctions),
      "http://localhost:12345/TrigServer");
```

接続を確立すると、クライアントはリモート サーバー オブジェクトのメソッドを複数呼び出します。

リモート サーバー オブジェクトで呼び出されるメソッド

```
Console.WriteLine("Cos(0) = {0}", trig.Cos(0.0));
Console.WriteLine("Sin(0) = {0}", trig.Sin(0.0));
Console.WriteLine("Cos(PI) = {0}", trig.Cos(trig.PI()));
```

クライアント アプリケーションの出力で、アプリケーションの操作が正しいことが示されます。

出力

```
リモート サーバーへの接続を始める
Cos(0) = 1
Sin(0) = 0
Cos(PI) = -1
```

2.7.3.4 Dotfuscator によるリモート出力の処理

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

リモートのサンプルに含まれるサンプルの Dotfuscator 構成ファイルは、複数の入力アセンブリを使用し、対象除外規則を使用してリモートから呼び出したオブジェクトの型情報を対象から除外する例を示します。構成ファイルでは、以下のように **<input>** アセンブリにある複数の **<file>** エントリで、複数の入力アセンブリを表現しています。

複数の入力アセンブリおよび対象除外規則の使用

```
<input>
  <filelist>
   <file dir="${projectdir}" name="ITrigFunctions.dll"/>
   <file dir="${projectdir}" name="TrigServer.exe"/>
   <file dir="${projectdir}" name="TrigClient.exe"/>
  </filelist>
</input>
```

この他に、構成ファイルには、名前の変更の対象除外リストという重要な設定があります。

名前の変更の対象除外規則

```
<renaming>
  <excludelist>
   <type name="Samples.Trig" excludetype="true"/>
  </excludelist>
…
</renaming>
```

<renaming> タグは、そのタグ内の対象除外規則が、具体的な識別子の名前変更に関連するものであることを示しています。これは、Dotfuscator の他の機能では有効化/無効化を選択できる点とは対照的です。

<excludelist> タグは、名前の変更処理対象から除外する必要のある項目のリストを定義します。**<type name="Samples.Trig">** タグは、名前の変更処理対象からクラス名 "**Samples.Trig**" を除外すること Dtfuscator に指示します。なお、このタグで参照されるのはクラス名そのものである点に注意してください。"Trig" クラスのすべてのメソッドは名前の変更対象のままです。この例でメソッドの名前変更が可能なのは、クライアントとサーバーの両方を Dotfuscator で処理しているからです。Dotfuscator で処理していないクライアントによって呼び出されるインターフェイスを作成しているのであれば、インターフェイスのメソッド名も変更されないよう設定しなくてはならなかったでしょう。すべてのアセンブリが同じプロジェクトに含まれているので、Dotfuscator はメソッド呼び出しの参照を、適切な名前に変更されたサーバー メソッド呼び出しに変更できます。

make.bat ファイルを実行すると、この構成ファイルを使用して Dotfuscator が実行されます。この結果、"**output**" サブディレクトリに **TrigServer.exe**、**TrigClient.exe** および **ITrigFunctions.dll** アセンブリが出力されます。出力先は、構成ファイルの次のセクションを変更することで制御できます。

TrigServer

```
<output>
  <file dir="${projectdir}\output" />
</output>
```

メモ：入力アセンブリと出力アセンブリは一対一の関係にあります。この関係は変更できません。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

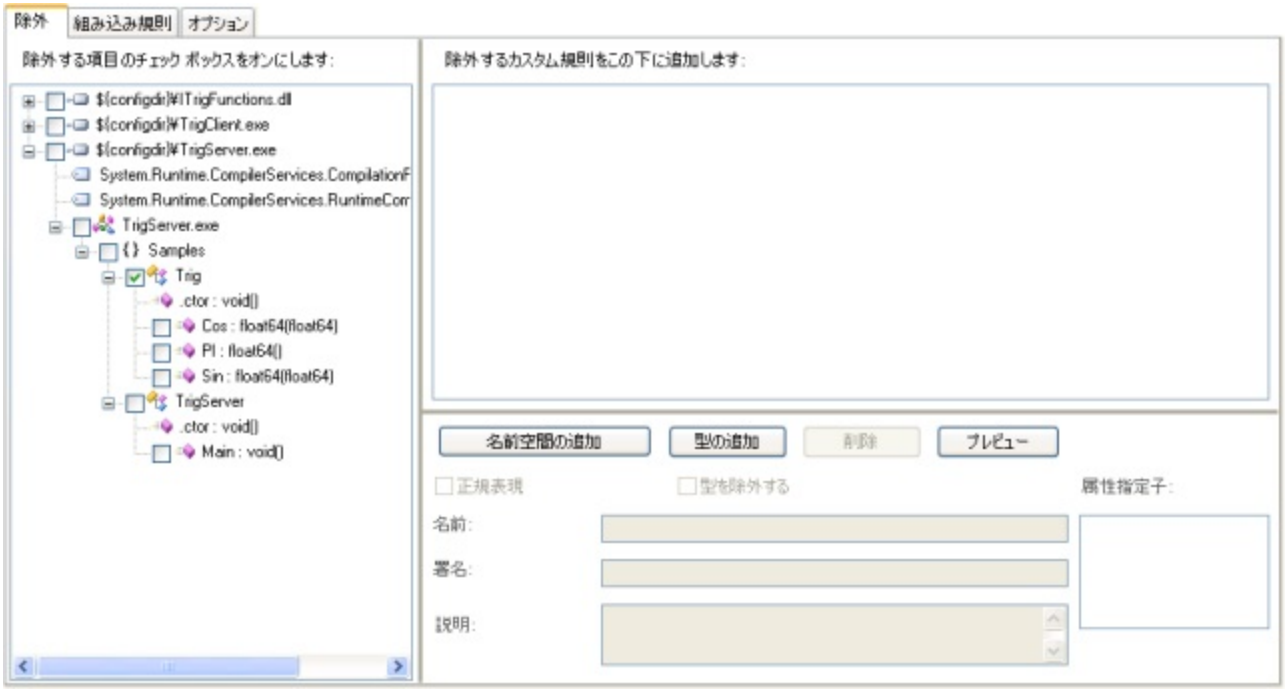
新しいアセンブリを実行すると、Dotfuscator が、必要な項目を名前の変更処理対象から正しく除外していることを確認できます。また、インターフェイスのメソッド名を変更したため、難読化されたクライアントが難読化されていないサーバーを呼び出すことも、その逆も不可能であることに注意してください。

2.7.3.5 グラフィカル ユーザー インターフェイスを使用したリモート サンプルの構成

リモート サンプルは、複数の入力アセンブリを保持する Dotfuscator の機能を利用しています。複数の入力アセンブリは、グラフィック インターフェイスを使用して簡単に追加できます。参照機能を使用すれば、簡単に各アセンブリを個別に見つけることができます。すべての入力アセンブリが[**入力アセンブリ**]リスト ボックスに表示されます。

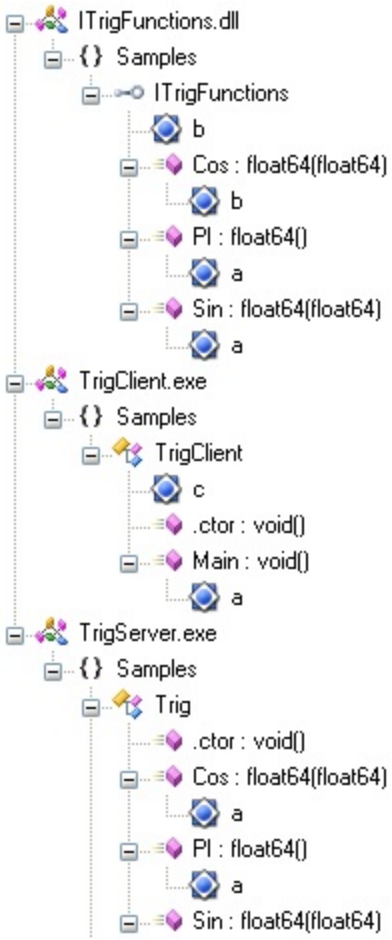
`TrigServer` アセンブリで定義されている "**`Samples.Trig`**" 型はリフレクションを使用して登録されるので、名前の変更対象から除外する必要があります。下図は、インターフェイスの [名前の変更] タブで、この項目が除外の対象として選択されている様子を示しています。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



ビルド後、[出力] タブに結果が表示されます。**`Samples.Trig`** クラス以外はすべての項目の名前が変更されたことに注目してください。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.



2.7.3.6 リモート サンプルのまとめ

リモート処理経由でクライアントにオブジェクトを渡すアプリケーションを Dotfuscator で処理するためには、クライアントに渡すオブジェクトの型名を名前変更処理の対象から除外することが重要になります。クライアントとサーバーの両方を Dotfuscator で処理する場合、Dotfuscator の威力がリモート処理において発揮されます。このような場合、Dotfuscator は内部メソッドやフィールドだけでなく、コンポーネントの外部インターフェイスの名前も積極的に変更できます。

2.7.4 ASP.NET のサンプル

ASP.NET では Web 開発者が利用できる一連のツールに、分離コードの概念が導入されました。分離コードを使用すると、どの .NET 言語でも Web アプリケーションのロジックを開発できます。このロジックは、Web サーバーが要求ベースで読み込む .NET アセンブリとしてプリコンパイルできます。Dotfuscator はこのようなアセンブリを処理し、使用しているサーバー インフラストラクチャによって Web アプリケーションに提供される保護層に新たな保護層を重ねることができます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ASP.NET のサンプルでは、名前の変更対象から型やフィールドを選択的に除外するための Dotfuscator の構成方法について例示します。サンプルには、Dotfuscator のコマンド ラインを使用した場合の例と、グラフィカル インターフェイスを使用した場合の例が紹介されています。

2.7.4.1 ASP.NET のサンプル ファイル

ASP.NET のサンプルには以下のファイルが含まれています。

ファイル	説明
asp.doc	このドキュメント
DotfuscatorASP.aspx	メインとなるサンプル ページ
DotfuscatorASP.aspx.cs	Dotfuscator で処理するコードを含んでいる C# 分離コード ファイル
asp_config.xml	Dotfuscator 構成ファイル
make.bat	シリアル化アプリケーションをコンパイルするバッチ ファイル
run.bat	アプリケーションで Dotfuscator を実行するためのバッチ ファイル
deploy.bat	難読化されたアセンブリを Web アクセス可能な場所へコピーするためのバッチ ファイル

ASP.NET のサンプル ファイルは Dotfuscator のインストール フォルダーに存在します。また Dotfuscator Samples Download ZIP を www.preemptive.com/dotfuscator-samples.html（英語）からダウンロードできます。

2.7.4.2 ASP.NET サンプルの準備

このサンプルでは、対象となるコンピューターに IIS バージョン 5.01 以上がインストールおよび設定されている必要があります。ASP.NET サンプルのインストール先を参照する仮想ディレクトリを、IIS の構成内に作成する必要があります。仮想ディレクトリの設定方法については、IIS のドキュメントを参照してください。

2.7.4.3 ASP.NET サンプルのビルド

ASP.NET のサンプルは、ユーザーの PATH 環境変数から C# コンパイラ（**csc.exe**）にアクセスできることを前提にしています。Visual Studio を使用している場合は、Windows の［スタート］ボタンをクリックして、［プログラム］、［Microsoft Visual Studio］、［Visual Studio ツール］の順にポイントし、［**Visual Studio コマンド プロンプト**］ショートカットを実行することでこれを確認できます。

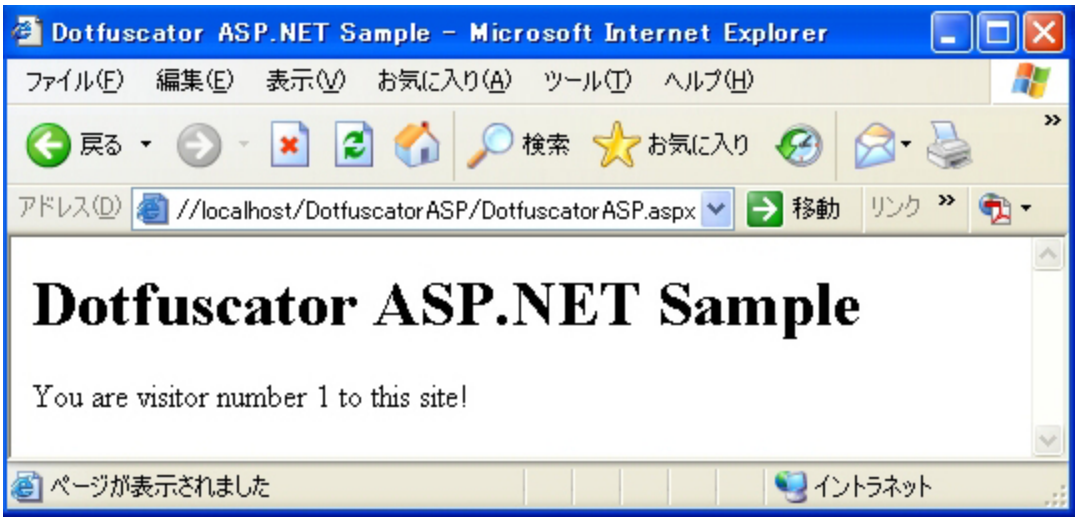
コマンド プロンプトで、カレント ディレクトリを ASP.NET サンプルを含むディレクトリに変更します。

make.bat を実行してアプリケーションをビルドします。このバッチ ファイルは Visual C# コンパイラを呼び出して、出力ファイル **DotfuscatorASP.dll** を生成します。出力ファイルは、現在のディレクトリの "bin" サブディレクトリに格納されます。bin は、IIS がコンパイル済みの分離コード アセンブリを検索する既定の場所です。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

2.7.4.4 ASP.NET サンプルの実行

このサンプル用に作成した仮想ディレクトリの **DotfuscatorASP.aspx** ページを Web ブラウザーで参照すると、**asp.net** サンプルを実行できます。以下のように表示されれば、サンプルは正しく構成されていることになります。



2.7.4.5 ASP.NET サンプル コードの確認

aspx ページ自体は非常に単純です。

ASPX Page

```
<%@ Page language="c#" Codebehind="DotfuscatorASP.aspx.cs" AutoEventWireup="false"
Inherits="DotfuscatorASP.Sample" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title>Dotfuscator ASP.NET Sample</title>
    </HEAD>
    <body>
        <h1>Dotfuscator ASP.NET Sample</h1>
        <form id="SampleForm" method="post" runat="server">
            <p>You are visitor number <asp:Label id="CountLabel" runat="server"
/> to this site!</p>
        </form>
    </body>
</HTML>
```

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

すべてのアプリケーション ロジックは、分離コード ファイル **DotfuscatorASP.aspx.cs** に格納されています。なお、aspx ページは、クラス "**DotfuscatorASP.Sample**" およびフィールド "**CountLabel**" を参照します。これらの項目は分離コードで定義されています。aspx ページは、実行時に Web サーバーで必要になるまで処理されないので、これらのシンボルはコンパイルされた分離コード アセンブリ内で変更しないでおく必要があります。これらのシンボルの名前は変更できないので、Dotfuscator が名前の変更処理から除外するように、特別な手順で設定する必要があります。

クラス "**DotfuscatorASP.Sample**" は、ページが読み込まれるごとに 1 ずつ加算される "**PageLoadCount**" という静的フィールドを持っています。この値は **CountLabel** フィールドに表示され、エンド ユーザーはブラウザー内でその値を見ることができます。なお、Web サーバーがアセンブリを再読み込みした場合、この値は保持されず、既定値 **0** にリセットされます。

2.7.4.6 Dotfuscator による ASP.NET 出力の処理

ASP.NET のサンプルに含まれるサンプルの Dotfuscator 構成ファイルは、aspx ファイルで参照する項目を、対象除外規則を使用して処理する例を示します。これらの項目は分離コード アセンブリで定義されています。このファイルは **asp_config.xml** という名前で、ASP.NET サンプルの他のファイルと同じディレクトリに格納されています。このファイルでは、参照される項目を対象から除外するセクションは次のようになっています。

ASP.NET ディレクトリ

```
<renaming>
  <excludelist>
   <type name="DotfuscatorASP.Sample">
    <field name="CountLabel"/>
   </type>
  </excludelist>
…
</renaming>
```

<renaming> タグは、そのタグ内の対象除外規則が、具体的な識別子の名前変更に関連するものであることを示しています。これは、Dotfuscator の他の機能では有効化/無効化を選択できる点とは対照的です。

<excludelist> タグは、名前の変更処理対象から除外する必要のある項目のリストを定義します。**<type name="DotfuscatorASP.Sample">** タグは、名前の変更処理対象からクラス名 "**DotfuscatorASP.Sample**" を除外することを Dtfuscator に指示します。なお、このタグで参照されるのはクラス名そのものである点に注意してください。"**Tester**" に属するすべてのメソッドは名前の変更対象となります。aspx ファイルはこのクラスから継承しているので、この処理が必要になります。**<field name="CountLabel"/>** タグは Dotfuscator に、ページの読み込みカウンターを表示するサーバー側のラベル コントロールを対象から除外することを指示します。

run.bat ファイルを実行すると、この構成ファイルを使用して Dotfuscator が実行されます。この結果、"**output**" サブディレクトリに **DotfuscatorASP.dll** アセンブリが出力されます。出力先は、構成ファイルの次のセクションを変更することで制御できます。

出力ディレクトリ

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

```
<output>
  <file dir="output" />
</output>
```

deploy.bat を実行すると、Web サーバーが検索する bin ディレクトリに、難読化されたアセンブリがコピーされます。その結果、難読化されていないアセンブリが上書きされます。

ブラウザーで Web サイトを表示すると、Dotfuscator が名前の変更処理対象から必要な項目を正しく除外したことを確認できます。なお、読み込みカウンターが既定値にリセットされている場合は、IIS が新しいアセンブリを再読み込みしたことを表しています。

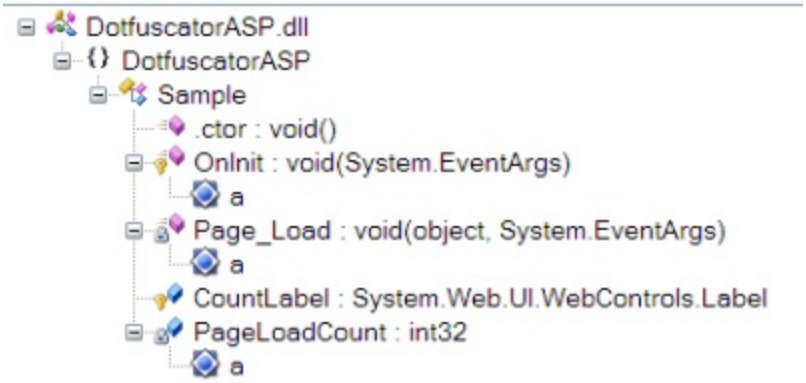
2.7.4.7 グラフィカル ユーザー インターフェイスを使用した ASP.NET のサンプルの構成

Dotfuscator のグラフィカル インターフェイスを使用して、構成ファイルをビジュアルに生成できます。名前の変更対象から除外する項目は、インターフェイスの[**名前の変更**]タブを使用して指定できます。ツリーのアセンブリ ノードを展開すると、アプリケーション構造がグラフィカルに表示されます。この構造には、すべての名前空間、型、およびメソッドが含まれています。

除外対象の項目の隣にあるチェック ボックスをオンにするだけで、名前の変更の対象除外リストをグラフィカルに生成できます。ここでは、型 "**DotfuscatorASP.Sample**" およびフィールド "**CountLabel**" を対象から除外する必要があります。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロジェクトをビルドすると、先ほど指定した型が名前の変更処理から正しく除外されていることがわかります。



2.7.4.8 ASP.NET サンプルのまとめ

プリコンパイルした分離コード アセンブリを利用する Web アプリケーションを Dotfuscator で処理するには、aspx ファイルが参照するすべてのシンボルが、名前の変更処理から除外されていることが必要です。対象から除外するのは、aspx の継承元の型や、aspx が名前によって直接参照するフィールドです。分離コードのその他の部分は、難読化することができます。

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

3 索引

[PreMark]エディター，110-111

[インストルメンテーション]エディター，128-131

[オプション]タブ，105

[名前の変更]の[オプション]タブ，91-92

[ヘルプ]メニュー，90-91 ，91 ，72

[リンク]エディター，106

[出力]タブ，69-70

[除去]エディター，101

[除去]エディターの[オプション]タブ，105

[除去]エディターの[組み込み規則]タブ，104-105

[除去]タブ，101

[条件付き対象]タブ，103-104

[制御フロー]エディター，98

[制御フロー]エディターの[オプション]タブ，98-99

[制御フロー]エディターの[除外]タブ，99-100

[設定]タブ，58-68

[組み込み規則]タブ，98 ，104-105

除去，104-105

名前の変更，98

[属性指定子]リスト ボックス，117 ，117-118

[対象トリガー]タブ，102-103

[入力]タブ，75-76

[入力アセンブリ]および[リンク アセンブリ]，106-107

[表示]メニュー，89-90 ，70-71

[文字列の暗号化]エディター，100

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

[文字列の暗号化]エディターの[対象]タブ，100-101

[名前の変更]エディター，91

[名前の変更]エディターの[オプション]タブ，91-92

[名前の変更]エディターの[除外]タブ，96-97

[名前の変更]エディターの[組み込み規則]タブ，98

[名前の変更]の[オプション]タブ，91-92

64 ビット アセンブリ，40

64 ビット アセンブリを Dotfuscator で処理する，40

accesspoint，144-147

ApplicationAttribute，152-153 ，153 ，153 ，230

ASP.NET サンプル コードの確認，284-285

ASP.NET サンプルのビルド，283

ASP.NET サンプルのまとめ，287

ASP.NET サンプルの実行，283-284

ASP.NET サンプルの準備，283

ASP.NET のサンプル ファイル，283

ASP.NET のサンプル，282-283

ASP.NET，282-283

Authenticode ，39-40

Authenticode 署名のアセンブリ，39-40

Authenticode

署名，39-40

BinaryAttribute，152-153 ，153 ，153 ，147-149 ，230-231

BusinessAttribute，152-153 ，153 ，153 ，147-149 ，231-232

ClickOnce 入力，57

ConstOnly オプション，200

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Debug グローバル オプション，186-187

Dotfuscate タスク，181-183

Dotfuscate

MSBuild タスク，181-183

Dotfuscator GUI，49，72-73，73，73-74，74-75，89，89，89-90，90-91，91，49-51，53-55，71-72，91-92，96-97，98，98，98-99，99-100，100，100-101，101，102-103，103-104，104-105，105，106，106-107，110-111，128-131，112-113，117，117-118，118-120

[インストルメンテーション]エディター，128-131

[ヘルプ]メニュー，90-91

[リンク]エディター，106

[除去]エディターの[オプション]タブ，105

[除去]エディターの[組み込み規則]タブ，104-105

[除去]タブ，101

[条件付き対象]タブ，103-104

[制御フロー]エディター，98

[属性指定子]リスト ボックス，117，117-118，118-120，120-121，121-122，122-124，124-125，125-126，126-127

[対象トリガー]タブ，102-103

[表示]メニュー，89-90，70-71

[文字列の暗号化]エディターの[対象]タブ，100-101

[名前の変更]エディターの[除外]タブ，96-97

[名前の変更]エディターの[組み込み規則]タブ，98

[名前の変更]の[オプション]タブ，91-92

PreMark，110-111

Shelf Life（使用期限）の期限切れトークンの生成，135-136

Visual Studio 統合，72-73，73-74，75，89，89-90

イベント，124-125

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

エディター，89，91，98，100，101

カスタマー フィードバックのオプション，170-171

カスタム規則，127

カスタム属性，125-126

スーパータイプ，126-127

スタンドアロン エクスプローラー，73-74

スタンドアロン，49-51，58-68，70-71，72

出力，69-70

ビルド，89，69

フィールド，121-122

プロジェクト ツリー，73-74

プロジェクトの作成，51-53

プロパティ，122-124

メソッド，120-121

ユーザー設定，91

リンク アセンブリ，106-107

制御フローの［オプション］タブ，98-99

制御フローの［除外］タブ，99-100

正規表現，117，117-118，118-120，120-121，121-122，122-124，124-125，125-126，128

宣言による難読化，127-128

対象選択規則および対象除外規則の編集，112-113

入力アセンブリ，87-88，53-55，106-107

名前空間，117-118

Dotfuscator エディション，14-15

Dotfuscator スタンドアロン GUI，75-76，51-53，58-68，68-69，69-70，70-71，72

Dotfuscator で処理する，37，181-183，267-268，272-273，285-286

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Dotfuscator による ASP.NET 出力の処理，285-286

Dotfuscator によるインストルメンテーションの理解，45-46

Dotfuscator によるシリアル化出力の処理，272-273

Dotfuscator によるリフレクション出力の処理，267-268

Dotfuscator によるリモート出力の処理，278-280

Dotfuscator による処理，37，37-38，38-39，40，41-43，181-183，267-268，272-273，278-280，285-286

ObfuscationAttribute，41-43

アセンブリ，37，37，37-38，38-39，40

Dotfuscator による難読化の理解，28-29

Dotfuscator の構成，92-93，96，95-96，97-98，263-264，34，30-31，106，228-229，264，75-76，58-68，68-69，96-97，98-99，99-100，103-104，106，110-111，214，221-222，183，183-184，184-185，185，185，185-186，186-187，187，187-188，188-189，189-190，190，190-191，191，191-192，192，192，192-193，193-195，195，195-196，196，196-197，197，197，197-198，198，198，198，199，199，199，201，201-202，202，204，204-205，205-206，206-207，208-210，210-211，211-212，212-213，213，190

Debug，186-187

DTD，228-229

honorOAs，189-190

HTML 形式の除去レポート，201-202

HTML 形式の名前の変更レポート，196-197

Library グローバル オプション，185

Library，183，185，186-187，188-189

NoDotfuscatorAttribute，187

nohonorSOs，190

nostripSOs，190

PreMark，110-111，204，204-205，205-206

Runtime Intelligence，210-211

stripOA，189-190

xaml の変換，190

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

XML シリアル化と名前の変更, 95-96

アセンブリ ロード パス, 190-191

ウォーターマーク, 204 , 204-205 , 205-206

エディター, 91 , 98 , 100 , 101 , 110-111 , 128-131

グローバル オプション, 183 , 185 , 185-186 , 186-187

スマート難読化, 212-213

バージョン属性, 183-184

ビルド イベント, 208-210

プロパティ リスト, 58-68 , 183 , 184-185

プロパティ, 183 , 184-185

メッセージ非表示, 185-186

リンク, 106 , 106-107 , 202 , 202 , 202-204

レポート, 106 , 105 , 196-197 , 201-202

一時ディレクトリ, 191-192

割り当てファイル, 263-264 , 264 , 91-92 , 196

機能割り当て, 192

出力ディレクトリ, 191

署名, 206-207

除去, 106 , 101 , 102-103 , 103-104 , 104-105 , 105 , 199 , 199-200 , 200-201 , 201 , 201-202

条件付き対象選択リスト, 103-104 , 201

制御フローの難読化, 30-31 , 98-99 , 99-100 , 197 , 197-198 , 198 , 198

増分難読化, 263-264 , 34

属性, 211-212

拡張, 211-212

対象除外リスト, 97-98 , 96-97 , 99-100 , 214

調査のみ実行, 185-186

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

入力アセンブリ，75-76，106-107，187-188

入力割り当てファイル，197

不要コードの除去，199

文字列の暗号化，100，100-101，198，199，199

名前の変更，93-95，92-93，96，95-96，97-98，91-92，185-186，192，192-193，193-195，195，195-196

Dotfuscator の構成 | 対象選択リスト，100-101，102-103，103-104，221-222

Dotfuscator の構成ファイル，176-179，179-180，183

オーバーライド，176-179

保存，179-180

補完，176-179

Dotfuscator の使用

Dotfuscator の登録とアクティブ化，15-17

Visual Studio 統合 UI クイック スタート，21-25

コマンド ライン クイック スタート，19-21

スタンドアロン GUI クイック スタート，17-19

概要，15

難読化後の出力の表示と検証，25-28

Dotfuscator の登録とアクティブ化，15-17

Dotfuscator プロジェクト，72-73，73，73-74，74-75，75，77-86

プロジェクト ツリー，73-74

プロパティ，77-86

構成，74-75

作成，73

配置，75

Dotfuscator プロジェクトの作成，73

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Dotfuscator プロジェクトの配置，75

Dotfuscator 名前の変更規則，192-193

loweralpha，192-193

numeric，192-193

unprintable，192-193

upperalpha，192-193

Dotfuscator，13，17-19，19-21，21-25，25-28，107，109-110，107-108，32，32-33，139，199-200，202

GUI，72-73，73-74，71-72

Shelf Life（使用期限），32-33

Visual Studio 統合クイック スタート，21-25

アセンブリ リンカー，109-110，107-108

アセンブリのリンク，107

コマンドライン クイック スタート，19-21

ステートメントランドアロン GUI クイック スタート，17-19

プロジェクト，73，74-75，77-86

プロジェクトの作成，73

リンク オプション，202

改ざん通知，32

型，118-120

構成，183，183-184

除去オプション，199-200

難読化された出力の検証，15，25-28

例外の追跡，139

dotfuscator_v2.3.dtd，228-229

dotfuscatorMap_v1.1.dtd，264

DTD，228-229，264，213

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Dotfuscator 構成ファイル，228-229，264，213

割り当てファイル，228-229，264，213

enhancedOI，193-195

Dotfuscator 名前の変更オプション，193-195

EventList セクション，208-210

ExceptionTrack 属性，139-140，232-238

ExceptionTrackAttribute，232-238

ExceptionTypes，238-239

Exit Point 属性，152

exit メソッド，152

FeatureAttribute，153-155，239-240，241-246，246-247

FeatureEventTypes，240-241

GUI による Dotfuscator 構成，49

honorOAs，189-190

HTML 形式の削除レポート，105

HTML 形式の名前の変更レポート，196-197

IJW サンク，37

ilasm.exe，44-45

Dotfuscator での使用，44-45

ildasm.exe，44-45

Dotfuscator での使用，44-45

InjectionPoints，253-254

InsertShelfLife 属性，134-135，132-134，241-246

InsertShelfLifeAttribute，241-246

InsertSignofLifeAttribute，246-247

InsertTamperCheck 属性，137，137-139，247-252

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

InsertTamperCheckAttribute，247-252

keephierarchy，193-195

Dotfuscator 名前の変更オプション，193-195

keepnamespace，193-195

Dotfuscator 名前の変更オプション，193-195

Library グローバル オプション，183 ，185

Library，185 ，188-189

グローバル オプション，183 ，185

loweralpha，192-193

Dotfuscator 名前の変更規則，192-193

Lucidator の使用，168-170

Lucidator，34-35

Managed C++ アセンブリ，37

Managed C++ および IJW（It Just Works）のサンク，37

MSBuild タスク リファレンス，181

MSBuild，181 ，181-183 ，183

Dotfuscator タスク，181 ，181-183

NoDotfuscatorAttribute グローバル オプション，187

NoDotfuscatorAttribute，187

nohonorSOs，190

nostripSOs，190

numeric，192-193

Dotfuscator 名前の変更規則，192-193

ObfuscateAssemblyAttribute，41-43 ，189-190

ObfuscationAttribute 機能割り当て，192

Dotfuscator の機能，192

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ObfuscationAttribute，41-43，189-190

P/Invoke メソッド，36-37

PerformanceProbeAttribute，155，252-253

PreEmptive.Attributes 名前空間，229，229-230

カスタム属性リファレンス，229

PreEmptive.Attributes，253-254，229-230，230，230-231，231-232，239-240，240-241，241-246，246-247，247-252，252-253，254-260，260，261，261-262，262-263，232-238，238-239

ApplicationAttribute，230

BinaryAttribute，230-231

BusinessAttribute，231-232

ExceptionTrack 属性，232-238

ExceptionTypes，238-239

FeatureAttribute，239-240，241-246，246-247

FeatureEventTypes，240-241

InjectionPoints，253-254

InsertShelfLifeAttribute，241-246

InsertSignofLifeAttribute，246-247

InsertTamperCheckAttribute，247-252

PerformanceProbeAttribute，252-253

SetupAttribute，254-260

SinkElements，260

SourceElements，261

SystemProfileAttribute，261-262

TeardownAttribute，262-263

PreEmptive.SOs.Attributes，229-230，230，262-263

PreMark オプション，204-205

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

PreMark セクション，204

PreMark タスク，183

PreMark の要素，205-206

PreMark，33 ，111 ，111-112 ，112 ，110-111 ，183 ，204 ，204-205 ，205-206

Dotfuscator，33 ，111 ，111-112 ，112

MSBuild タスク，183

ウォーターマーク文字コード，111-112

エディター，110-111

オプション，204-205

抽出，112

文字列，111

要素，205-206

Runtime Intelligence

Dotfuscator によるインストルメンテーションの理解，45-46

Exit Point 属性，152

Runtime Intelligence のカスタム属性，147

Runtime Intelligence 対応アプリケーションの例，160-166

Shelf Life および Sign of Life 属性，153

アセンブリ レベルの属性，147-149

アプリケーション分析（Runtime Intelligence），143-144

アプリケーション分析を備えた Dotfuscator のテストとデバッグ，158

アプリケーション分析を備えた Dotfuscator の構成および実行，144-147

インストルメンテーションの差し込み，46-48

エントリ ポイント属性，149-152

パフォーマンス属性，155

メソッド パラメーターを拡張キーとして自動的に送信する，157-158

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

メッセージ データのダウンロード，159-160

メッセージの監視，159

メッセージ追跡の構成，158-159

改ざん通知属性，152-153

拡張キーを含むユーザー定義データの送信，156-157

環境属性，155-156

機能の利用状況属性，153-155

例外の追跡属性，153

Runtime Intelligence セクション，210-211

Runtime Intelligence のカスタム属性，147

Runtime Intelligence 対応アプリケーションの例，160-166

Runtime Intelligence，32，48-49，137，137-139，139，132，131-132，134-135，132-134，139-140，140-141，141-143，143-144，147，152-153，153，153，147-149，149-152，152，153-155，155，155-156，156-157，144-147，158，158-159，159，159-160，160-166，229，229-230，189-190，210-211

Dotfuscator の構成，210-211

ExceptionTrack 属性，139-140

Exit メソッド，152

Feature 属性，153-155

InsertShelfLife 属性，134-135，132-134

InsertTamperCheck 属性，137-139

PerformanceProbe 属性，155

PreEmptive.Attributes，229-230

setup 属性，149-152

Shelf Life Activation Key，132

Shelf Life および Sign of Life 属性，153

Shelf Life トークンの概要，131-132

SLAK，132

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

SystemProfile 属性，155-156

Teardown 属性，152

アセンブリの属性，152-153，153，153，147-149

アプリケーション パフォーマンスの追跡，155

アプリケーション環境の追跡，155-156

エントリ ポイント，149-152

カスタム属性，147，229

サンプル，160-166

テスト，158，158-159，159，159-160

デバッグ，158

メッセージ追跡，158-159，159，159-160

ユーザー指定レポート，141-143

ユーザー定義データ，156-157

改ざんの検出，137，137-139

改ざん通知，137，137-139，152-153

概要，143-144

拡張キー，156-157

期限切れの検出，132-134

期限切れの通知，134-135，132-134

構成，144-147

例外の追跡属性，153

例外通知，140-141

SetupAttribute，149-152，254-260

Shelf Life Activation Key の概要，132

Shelf Life および Sign of Life 属性，153，241-246，246-247

Shelf Life トークンの概要，131-132

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Shelf Life（使用期限）

Shelf Life Activation Key の概要，132

Shelf Life トークンの概要，131-132

Shelf Life，32-33

期限切れと警告のレポート，134-135

期限切れと警告の操作，132-134

Shelf Life（使用期限）

Dotfuscator，32-33

アクティブ化キー，132

トークン，131-132

期限切れの検出，132-134

期限切れの通知，132-134

Shelf Life，32-33 ，132 ，131-132 ，134-135 ，132-134 ，135-136

InsertShelfLife 属性，134-135 ，132-134

SLAK，132

使用期限の期限切れトークンの生成，135-136

通知，134-135 ，132-134

Silverlight 入力，56-57

SinkElements，260

SLAK，132

SmartObfuscation セクション，212-213

sn.exe，44-45

Dotfuscator での使用，44-45

SourceElements，261

stripOAs，189-190

SuppressIldasmAttribute グローバル オプション，186

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

SystemProfileAttribute，155-156，261-262

TeardownAttribute，152，262-263

unprintable（印刷不能文字），192-193

unprintable

Dotfuscator 名前の変更規則，192-193

upperalpha，192-193

Dotfuscator 名前の変更規則，192-193

Verbose、Quiet、Investigate グローバル オプション，185-186

Visual Studio インターフェイス，72-73

Visual Studio 統合 UI クイック スタート，21-25

Visual Studio 統合，21-25，72-73，73，73-74，74-75，75，77-86，87-88，89，89，89-90，90-91，91，53-55

［ヘルプ］メニュー，90-91

［表示］メニュー，89-90

Dotfuscator プロジェクトの作成，73

Dotfuscator，72-73

エディター，89

クイック スタート，21-25

ビルド，89

プロジェクト ツリー，73-74

プロジェクト プロパティ，77-86

プロジェクトの構成，74-75

プロジェクトの配置，75

ユーザー設定，91

レポートの表示，89-90

入力アセンブリ プロパティ，87-88

入力アセンブリ，53-55

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

Visual Studio 統合クイック スタート，21-25

Windows ストア入力，57-58

xaml の変換，190

XML シリアル化と名前の変更，95-96

XML 構成ファイルに関する注意，213

アセンブリ リンカー，109-110 ，107-108

アセンブリ レベルの属性，147-149

アセンブリ ロード パス，190-191

アセンブリのリンク

アセンブリのリンク，33-34

エントリ ポイント，109-110

プライマリ アセンブリ，107

名前の改変，107-108

アセンブリのリンク，33-34 ，106 ，106-107 ，202 ，202 ，202-204

Dotfuscator，33-34

エディター，106

エントリ ポイント，109-110

プライマリ アセンブリ，107

名前の変換，107-108

アセンブリの結合

Dotfuscator，33-34

アセンブリの情報の難読化，36-37 ，37 ，37 ，40-41

アセンブリの属性，152-153 ，153 ，153 ，147-149

アセンブリの対象除外，221

アセンブリの対象選択，228

アセンブリ単位の XAML の変換モード，190

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

アセンブリ単位のインストルメンテーション処理，190

アセンブリ単位のライブラリ モード，188-189

アセンブリ単位の宣言による難読化，189-190

アプリケーション パフォーマンスの追跡，155

アプリケーション環境の追跡，155-156

アプリケーション分析（Runtime Intelligence），143-144

アプリケーション分析，153-155 ，144-147

Feature 属性，153-155

構成，144-147

アプリケーション分析を備えた Dotfuscator のテストとデバッグ，158

アプリケーション分析を備えた Dotfuscator の構成および実行，144-147

イベント，124-125

イベントによる選択，124-125

イベントの対象除外，219

イベントの対象選択，226

インストルメンテーション（改ざん、Shelf Life、例外、分析），128-131

インストルメンテーションの差し込み，46-48

ウォーターマーク

ウォーターマーク，33

ウォーターマークの抽出，112

ウォーターマーク文字列の長さ，111

文字コード，111-112

ウォーターマーク，33 ，111 ，111-112 ，112 ，204 ，204-205 ，205-206

Dotfuscator，33 ，111 ，111-112 ，112

ウォーターマーク文字列，111

抽出，112

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

文字コード，111-112

ウォーターマークの抽出，112 ，183

MSBuild タスク，183

ウォーターマーク文字列の長さ，111

エントリ ポイント，109-110 ，149-152 ，108-109

メソッド，149-152

設定，108-109

エントリ ポイントの設定，108-109

エントリ ポイント属性，149-152

オーバーロード誘導，29-30 ，193-195

拡張，92-93 ，193-195

カスタマー フィードバックのオプション，170-171

カスタム エンドポイント，128-131

カスタム規則，127 ，213-214 ，214 ，221-222

カスタム規則の作成，117

カスタム規則リファレンス

アセンブリの対象除外，221

アセンブリの対象選択，228

イベントの対象除外，219

イベントの対象選択，226

カスタム規則リファレンス，213-214

カスタム属性による対象除外，219-220

カスタム属性による対象選択，226-227

スーパータイプによる対象除外，220-221

スーパータイプによる対象選択，227-228

フィールドの対象除外，217-218

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

フィールドの対象選択，224-225

プロパティの対象除外，218

プロパティの対象選択，225-226

メソッドの対象除外，216-217

メソッドの対象選択，223-224

モジュールの対象除外，221

モジュールの対象選択，228

型の対象除外，215-216

型の対象選択，222-223

対象除外規則，214

対象選択規則，221-222

名前空間の対象除外，214-215

名前空間の対象選択，222

カスタム規則リファレンス，213-214

カスタム属性，147 ，41-43 ，229 ，125-126

リファレンス，229

カスタム属性による選択，125-126

カスタム属性による対象除外，219-220

カスタム属性による対象選択，226-227

カスタム属性リファレンス，229

カスタム属性を使用した宣言による難読化，41-43

クイック スタート，15 ，17-19 ，19-21 ，21-25

Visual Studio 統合クイック スタート，21-25

コマンドライン クイック スタート，19-21

スタンドアロン GUI クイック スタート，17-19

クラスの名前の変更オプション，93-95

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

グラフィカル ユーザー インターフェイス リファレンス

[PreMark]エディター，110-111

[オプション]タブ，105

[ヘルプ]メニュー，90-91 , 72

[リンク]エディター，106

[出力]タブ，69-70

[除去]エディター，101

[条件付き対象]タブ，103-104

[制御フロー]エディター，98

[制御フロー]エディターの[オプション]タブ，98-99

[制御フロー]エディターの[除外]タブ，99-100

[設定]タブ，58-68

[組み込み規則]タブ，104-105

[対象トリガー]タブ，102-103

[入力アセンブリ]および[リンク アセンブリ]，106-107

[表示]メニュー，89-90

[文字列の暗号化]エディター，100

[文字列の暗号化]エディターの[対象]タブ，100-101

[名前の変更]エディター，91

[名前の変更]エディターの[オプション]タブ，91-92

[名前の変更]エディターの[除外]タブ，96-97

[名前の変更]エディターの[組み込み規則]タブ，98

Dotfuscator プロジェクトの作成，73

Dotfuscator プロジェクトの配置，75

GUI による Dotfuscator 構成，49

Visual Studio インターフェイス，72-73

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

イベントによる選択，124-125

インストルメンテーション（改ざん、Shelf Life、例外、分析），128-131

エントリ ポイントの設定，108-109

カスタマー フィードバックのオプション，170-171

カスタム規則の作成，117

カスタム属性による選択，125-126

スーパータイプによる選択，126-127

スタンドアロン GUI，49-51

ソリューション エクスプローラーおよび Dotfuscator プロジェクト ツリー，73-74

フィールドによる選択，121-122

プロジェクト ファイルとレポートの表示，70-71

プロジェクト プロパティ，77-86

プロジェクトのビルド，89 , 69

プロジェクトの構成，74-75

プロジェクトの作成，51-53

プロパティによる選択，122-124

メソッドによる選択，120-121

ユーザー設定の設定，91 , 71-72

各機能の設定の構成，89 , 68-69

規則での宣言による難読化の使用，127-128

規則のプレビュー，128

規則の編集インターフェイス，112-113

規則の編集と削除，127

型による選択，118-120

個々の要素の選択，113-117

新しい Shelf Life トークンの生成，135-136

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

難読化されたスタック トレースのデコード，166-168

入力アセンブリ プロパティ，87-88

入力の設定，75-76，53-55

入力パッケージ プロパティ，88-89

名前空間による選択，117-118

グラフィカル ユーザー インターフェイス リファレンス，49

グラフィカル ユーザー インターフェイス，72-73，49-51，112-113

グラフィカル ユーザー インターフェイスを使用した ASP.NET のサンプルの構成，286-287

グラフィカル ユーザー インターフェイスを使用したシリアル化のサンプルの構成，273-276

グラフィカル ユーザー インターフェイスを使用したリフレクション サンプルの構成，268-270

グラフィカル ユーザー インターフェイスを使用したリモート サンプルの構成，280-282

グラフィカルな規則の編集インターフェイス，112-113

グローバル オプション，77-86，183，185，185，185-186，186-187，187

Debug，183，185，186-187

Library，183，185

NoDotfuscatorAttribute，187

メッセージ非表示，185-186

詳細表示，185-186

調査のみ実行，185-186

グローバル セクション，185

コード保護

オプション，198

コマンド ライン インターフェイス リファレンス

コマンド ライン インターフェイス リファレンス，171-172

コマンド ライン オプションの概要，172-176

コマンド ラインからのグラフィカル ユーザー インターフェイスの起動，180-181

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

コマンド ラインからの構成ファイルの保存，179-180

コマンド ラインでのオプション指定による構成ファイルの補完またはオーバーライド，176-179

コマンド ライン インターフェイス リファレンス，171-172

コマンド ライン インターフェイス，171-172 ，172-176 ，176-179 ，179-180 ，180-181

Dotfuscator，172-176 ，176-179

GUI の起動，180-181

オプションの概要，172-176

コマンド ライン オプションの概要，172-176

コマンド ライン クイック スタート，19-21

コマンド ラインからのグラフィカル ユーザー インターフェイスの起動，180-181

コマンド ラインからの構成ファイルの保存，179-180

コマンド ラインでのオプション指定による構成ファイルの補完またはオーバーライド，176-179

コマンドライン インターフェイス

構成ファイル，176-179 ，179-180

使用，171-172

コマンドライン クイック スタート，15 ，19-21

サイド バイ サイド インストール，264-265

サテライト DLL を使用するアセンブリを Dotfuscator で処理する，37

サテライト DLL，37

サンプル

ASP.NET サンプル コードの確認，284-285

ASP.NET サンプルのビルド，283

ASP.NET サンプルのまとめ，287

ASP.NET サンプルの実行，283-284

ASP.NET サンプルの準備，283

ASP.NET のサンプル ファイル，283

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ASP.NET のサンプル，282-283

Dotfuscator による ASP.NET 出力の処理，285-286

Dotfuscator によるシリアル化出力の処理，272-273

Dotfuscator によるリフレクション出力の処理，267-268

Dotfuscator によるリモート出力の処理，278-280

グラフィカル ユーザー インターフェイスを使用した ASP.NET のサンプルの構成，286-287

グラフィカル ユーザー インターフェイスを使用したシリアル化のサンプルの構成，273-276

グラフィカル ユーザー インターフェイスを使用したリフレクション サンプルの構成，268-270

グラフィカル ユーザー インターフェイスを使用したリモート サンプルの構成，280-282

サンプル，265

シリアル化サンプルのビルド，271

シリアル化サンプルのまとめ，276

シリアル化サンプルの実行，271-272

シリアル化のサンプル ファイル，270-271

シリアル化のサンプル，270

リフレクション サンプルのビルド，266

リフレクション サンプルのまとめ，270

リフレクション サンプルの実行，266-267

リフレクションのサンプル ファイル，265-266

リフレクションのサンプル，265

リモート サンプルのビルド，277

リモート サンプルのまとめ，282

リモート サンプルの実行，277-278

リモートのサンプル ファイル，276-277

リモートのサンプル，276

サンプル アプリケーション，15 ，265 ，265 ，270 ，276 ，282-283

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

ASP .NET のサンプル，265 , 282-283 , 283 , 283 , 283 , 283-284 , 284-285 , 285-286 , 286-287 , 287

シリアル化のサンプル，265 , 270 , 270-271 , 271 , 271-272 , 272-273 , 273-276 , 276

リフレクション サンプル，265 , 265 , 265-266 , 266 , 266-267 , 267-268 , 268-270 , 270

リモートのサンプル，265 , 276 , 276-277 , 277 , 277-278 , 278-280 , 280-282 , 282

難読化，265

ASP.NET の使用，282-283

シリアル化の使用，270

リフレクションの使用，265

リモートの使用，276

サンプル，265

シリアル化，270

シリアル化サンプルのビルド，271

シリアル化サンプルのまとめ，276

シリアル化サンプルの実行，271-272

シリアル化のサンプル ファイル，270-271

シリアル化のサンプル，270

スーパータイプ，126-127

スーパータイプによる選択，126-127

スーパータイプによる対象除外，220-221

スーパータイプによる対象選択，227-228

スタック トレース デコーダー，34-35

スタック トレースのデコード ツール，89-90

スタック トレースのデコード，34-35 , 166-168

スタンドアロン GUI クイック スタート，17-19

スタンドアロン GUI，49-51

クイック スタート，15 , 17-19

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

スタンドアロン グラフィカル ユーザー インターフェイス，49-51

プロジェクト，51-53

スマート難読化，35-36 ，212-213

ソリューション エクスプローラー，73-74

ソリューション エクスプローラーおよび Dotfuscator プロジェクト ツリー，73-74

タイムスタンプ，58-68

タスク リファレンス

Dotfuscate タスク，181-183

MSBuild タスク リファレンス，181

PreMark タスク，183

ディレクトリ入力，55-56

デジタル タイムスタンプ，58-68

デジタル署名セクション，207-208

デバッグ，158 ，166-168 ，186-187

DebugImpl，186-187

DebugOpt，186-187

Pdb，186-187

オプション，186-187

トリガー ファイル，75-76

トリガー リスト，200-201

バージョン，183-184

属性，183 ，183-184

はじめに

Dotfuscator エディション，14-15

Dotfuscator による難読化の理解，28-29

Dotfuscator，13

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

はじめに，13-14

難読化されたアプリケーションのビルドとデバッグ，33

保護概念，29

はじめに，13-14

パッケージ

ClickOnce 入力，57

Silverlight 入力，56-57

Windows ストア入力，57-58

ディレクトリ入力，55-56

パフォーマンス属性，155

ビルド イベント，43-44 , 77-86 , 208-210

Dotfuscator 内 , 43-44

ビルド コマンド，69

ビルド ディレクトリ，77-86

ビルド後のイベント，43-44

Dotfuscator 内，43-44

ビルド設定，77-86

ビルド前のイベント，43-44

Dotfuscator 内，43-44

ファイル リファレンス

ConstOnly オプション，200

Debug グローバル オプション，186-187

EventList セクション，208-210

HTML 形式の名前の変更レポート，196-197

Library グローバル オプション，185

NoDotfuscatorAttribute グローバル オプション，187

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

PreMark オプション，204-205

PreMark セクション，204

PreMark の要素，205-206

Runtime Intelligence セクション，210-211

SmartObfuscation セクション，212-213

SuppressIldasmAttribute グローバル オプション，186

Verbose、Quiet、Investigate グローバル オプション，185-186

XML 構成ファイルに関する注意，213

アセンブリ単位の XAML の変換モード，190

アセンブリ単位のインストルメンテーション処理，190

アセンブリ単位のライブラリ モード，188-189

アセンブリ単位の宣言による難読化，189-190

グローバル セクション，185

デジタル署名セクション，207-208

バージョン，183-184

プロパティ リストとプロパティ，184-185

ユーザー定義のアセンブリ ロード パス，190-191

リンク アセンブリ，202-204

リンク セクション，202

リンク無効オプション，202

一時ディレクトリ，191-192

拡張属性セクション，211-212

構成ファイル リファレンス，183

出力ディレクトリ，191

出力割り当てファイル，196

署名セクション，206-207

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

除去セクション（不要コードの除去），199

除去トリガー リスト，200-201

除去の参照規則，201

除去レポート，201-202

除去無効オプション，199-200

条件付き対象選択リスト，201

制御フローの対象除外リスト，198

制御フローの難読化オプション，198

制御フローの難読化セクション，197

制御フローの難読化レベル，197-198

難読化属性の機能割り当て，192

入力アセンブリ リスト，187-188

入力割り当てファイル，197

文字列の暗号化オプション，199

文字列の暗号化セクション，198

文字列の暗号化対象選択リスト，199

名前の変更オプション，193-195

名前の変更セクション，192

名前の変更の参照規則，195-196

名前の変更の対象除外リスト，195

名前の変更規則，192-193

フィールド，121-122

フィールドによる選択，121-122

フィールドの対象除外，217-218

フィールドの対象選択，224-225

プライマリ アセンブリ，107

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プラットフォーム呼び出し，36-37

プレフィックス，193-195

Dotfuscator 名前の変更オプション，193-195

フレンド アセンブリ，44

Dotfuscator，44

プロジェクト ツリー，73-74

プロジェクト ファイルとレポートの表示，70-71

プロジェクト プロパティ，77-86

プロジェクトのビルド，89 ，69

プロジェクトの構成，74-75

プロジェクトの作成，51-53

プロパティ リストとプロパティ，184-185

プロパティ，58-68 ，122-124 ，183 ，184-185

プロパティによる選択，122-124

プロパティの対象除外，218

プロパティの対象選択，225-226

マネージ リソース，37

マネージ リソースを持つアセンブリを Dotfuscator で処理する，37

マルチ モジュール アセンブリ，37-38

マルチ モジュール アセンブリを Dotfuscator で処理する，37-38

メソッド パラメーターを拡張キーとして自動的に送信する，157-158

メソッド，120-121

メソッドによる選択，120-121

メソッドの対象除外，216-217

メソッドの対象選択，223-224

メッセージ データのダウンロード，159-160

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

メッセージ，158-159 ，159 ，159-160

データのダウンロード，159-160

監視，159

構成，158-159

メッセージの監視，159

メッセージ追跡の構成，158-159

モジュールの対象除外，221

モジュールの対象選択，228

ユーザー指定の例外レポート情報の収集，141-143

ユーザー指定レポート，141-143

ユーザー設定，91 ，71-72

ユーザー設定の設定，91 ，71-72

ユーザー定義アセンブリ ロード パス，190-191

Dotfuscator，190-191

ユーザー定義のアセンブリ ロード パス，190-191

ユーザー文字列の暗号化

文字列の暗号化，31

リファレンス

dotfuscator_v2.3.dtd，228-229

dotfuscatorMap_v1.1.dtd，264

InjectionPoints，253-254

サイド バイ サイド インストール，264-265

リファレンス，171

高度なトピック，264

並行ビルド，265

リファレンス，171

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

リフレクション サンプルのビルド，266

リフレクション サンプルのまとめ，270

リフレクション サンプルの実行，266-267

リフレクション，40-41 ，265

アセンブリの難読化，265

難読化のサンプル アプリケーション，265

リフレクションおよび動的クラス読み込み，40-41

リフレクションのサンプル ファイル，265-266

リフレクションのサンプル，265

リモート サンプルのビルド，277

リモート サンプルのまとめ，282

リモート サンプルの実行，277-278

リモートのサンプル ファイル，276-277

リモートのサンプル，276

リンク アセンブリ，202-204

リンク セクション，202

リンク，106 ，202 ，202 ，202-204

Dotfuscator，33-34 ，106

アセンブリ，107 ，202 ，202 ，202-204

エディター，106

オプション，202

リンク無効オプション，202

レポート情報，141-143

一時ディレクトリ，191-192

改ざんのシミュレート，139

改ざんの検出および防御，32

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

改ざんレポート，137

改ざん検出，32 ，48-49 ，137 ，137-139 ，139 ，152-153 ，158-159 ，159 ，159-160 ，229 ，210-211

InsertTamperCheck 属性，137

カスタム属性リファレンス，229

テスト，139 ，158-159

メッセージ追跡，158-159 ，159 ，159-160

改ざん操作，137-139

改ざん通知

改ざんのシミュレート，139

改ざんの検出および防御，32

改ざんレポート，137

改ざん操作，137-139

改ざん通知，48-49 ，136-137

改ざん通知，32 ，48-49 ，137 ，137-139 ，139 ，136-137 ，152-153 ，210-211

Dotfuscator，32

概要，48-49

通知，48-49 ，137-139

改ざん通知属性，152-153

外部ツールの検索，44-45

概要，15

各機能の設定の構成，89 ，68-69

拡張オーバーロード誘導によるメソッドの名前の変更，92-93

拡張キーを含むユーザー定義データの送信，156-157

拡張属性，211-212

拡張属性セクション，211-212

割り当てファイル，263-264 ，196 ，197 ，213

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

DTD，213

出力，196

入力，197

環境属性，155-156

期限切れと警告のレポート，134-135

期限切れと警告の操作，132-134

期限切れの検出，132-134

期限切れの通知，134-135 ，132-134

機能の利用状況属性，153-155

機能割り当て，192

規則での宣言による難読化の使用，127-128

規則のプレビュー，128

規則の編集インターフェイス，112-113

規則の編集と削除，127

逆コンパイラ，13-14

妨害，13-14

逆コンパイル，13-14

型，118-120

選択，118-120

型による選択，118-120

型の対象除外，215-216

型の対象選択，222-223

検索機能，128-131

検出，137 ，158

厳密名のアセンブリ，38-39

厳密名のアセンブリを Dotfuscator で処理する，38-39

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

個々の要素の選択，113-117

更新，90-91，72

更新情報の確認，90-91，72

構成エディター，68-69，91，101，106，110-111，128-131

[PreMark]エディター，110-111

[インストルメンテーション]エディター，128-131

[リンク]エディター，106

[除去]エディター，101

[名前の変更]エディター，91

構成ファイル リファレンス，183

構成プロパティ，77-86

構成マネージャー，74-75

高度なトピック

64 ビット アセンブリを Dotfuscator で処理する，40

Authenticode 署名のアセンブリ，39-40

Managed C++ および IJW（It Just Works）のサンク，37

P/Invoke メソッド，36-37

カスタム属性を使用した宣言による難読化，41-43

サテライト DLL を使用するアセンブリを Dotfuscator で処理する，37

スマート難読化，35-36

ビルド イベント，43-44

フレンド アセンブリ，44

マネージ リソースを持つアセンブリを Dotfuscator で処理する，37

マルチ モジュール アセンブリを Dotfuscator で処理する，37-38

リフレクションおよび動的クラス読み込み，40-41

外部ツールの検索，44-45

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

厳密名のアセンブリを Dotfuscator で処理する，38-39

高度なトピック，35

高度なトピック，35 ，264

64 ビット アセンブリ，40

IJW サンク，37

P/Invoke メソッド，36-37

カスタム属性を使用した宣言による難読化，41-43

サテライト DLL を使用するアセンブリを Dotfuscator で処理する ，37

スマート難読化，35-36

ビルド イベント，43-44

フレンド アセンブリ，44

マネージ リソースを持つアセンブリを Dotfuscator で処理する ，37

マルチ モジュール アセンブリの難読化 ，37-38

リフレクションおよび動的クラス読み込み，40-41

外部ツール，44-45

厳密名のアセンブリ，38-39

辞書，141-143

識別子の名前の変更

Lucidator の使用，168-170

XML シリアル化と名前の変更，95-96

クラスの名前の変更オプション，93-95

拡張オーバーロード誘導によるメソッドの名前の変更，92-93

割り当てファイル，263-264

増分難読化，34

難読化されたコードのデバッグ，34-35

名前の変更，29-30

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

名前の変更の対象除外，97-98

名前の変更時、明示的なメソッドのオーバーライドを導入する，96

識別子の名前の変更，29-30

難読化技法，29-30

出力ディレクトリ，191

出力割り当てファイル，196

署名，39-40，77-86

署名セクション，206-207

除去，32，106，101，102-103，104-105，105，199，199-200，200-201，201，201，201-202

［オプション］タブ，105

［組み込み規則］タブ，104-105

Dotfuscator，32，106

HTML レポート，201-202

エディター，101

オプション，199-200

トリガー リスト，102-103，200-201

レポート，106，196-197，201-202

参照規則リスト，201

条件付き対象選択リスト，103-104，201

除去セクション（不要コードの除去），199

除去トリガー リスト，200-201

除去の参照規則，201

除去レポート，106，201-202

除去無効オプション，199-200

条件付き対象選択，103-104

条件付き対象選択リスト，201

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

新しい Shelf Life トークンの生成，135-136

制御フロー，30-31

制御フローの[オプション]タブ，98-99

制御フローの[除外]タブ，99-100

制御フローの対象除外リスト，198

制御フローの難読化

制御フロー，30-31

制御フローの難読化，30-31 ，98-99 ，197 ，197-198 ，198 ，198

[オプション]タブ，98-99

オプション，198

レベル，197-198

対象除外リスト，99-100 ，198

制御フローの難読化オプション，198

制御フローの難読化セクション，197

制御フローの難読化レベル，197-198

正規表現，117

イベントの対象除外，124-125

イベントの対象選択，124-125

カスタム属性の対象除外，125-126

カスタム属性の対象選択，125-126

スーパータイプの対象除外，126-127

スーパータイプの対象選択，126-127

フィールドの対象除外，121-122

フィールドの対象選択，121-122

プレビュー，128

プロパティの対象除外，122-124

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロパティの対象選択，122-124

メソッドの対象除外，120-121

メソッドの対象選択，120-121

型の対象除外，118-120

型の対象選択，118-120

削除，127

編集，127

名前空間の対象除外，117-118

名前空間の対象選択，117-118

宣言による難読化，41-43，127-128，189-190，192

機能割り当て，192

増分難読化，263-264，34

属性の処理，190

対象トリガーと条件付き対象の理解，101-102

対象除外規則，213-214，214

対象選択規則，213-214，221-222

知的財産，13-14

定数のみ除去，105-106

難読化，95-96，263-264，36-37，37，37，37，37-38，38-39，40，40-41，41-43，43-44，44-45，189-190，197，197-198，198，212-213，265

64 ビット アセンブリ，40

ASP.NET の使用，282-283

Managed C++ アセンブリ，37

XML シリアル化，95-96

カスタム属性の使用，41-43

サテライト DLL を使用するアセンブリ，37

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

シリアル化，270

シリアル化を含むか使用するアセンブリ，270

シリアル化を使用するアセンブリ，270

スマート，35-36，212-213

プラットフォーム呼び出しを使用するアセンブリ，36-37

マネージ リソース名，37

マルチ モジュール アセンブリ，37-38

リフレクションの使用，265

リフレクションを含むか使用するアセンブリ，265

リフレクションを使用するアセンブリ，40-41，265

リモートの使用，276

リモートを含むか使用するアセンブリ，276

リモートを使用するアセンブリ，276

厳密名のアセンブリ，38-39

混合コードのアセンブリ，37

制御フロー，30-31，197，197-198，198，198

増分，263-264，34

難読化アプリケーションのデバッグ，166-168

難読化されたアプリケーションのビルドとデバッグ，33

難読化されたコードのデバッグ，34-35

難読化されたスタック トレースのデコード，166-168

難読化後の出力の表示と検証，25-28

難読化属性の機能割り当て，192

入力アセンブリ プロパティ，87-88

入力アセンブリ リスト，187-188

入力アセンブリ，75-76，87-88，53-55，187-188

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

プロパティ，87-88

入力の設定，75-76 ，53-55

入力パッケージ プロパティ，88-89

入力割り当てファイル，197

表紙

表紙，1-2

表紙，1-2

不要コードの除去

除去レポート，106

対象トリガーと条件付き対象の理解，101-102

定数のみ除去，105-106

不要コードの除去，32

不要コードの除去，32 ，106 ，199

Dotfuscator，32

サイズの削減，32

Dotfuscator，32

文字コード，111-112

文字列の暗号化，31 ，100 ，100-101 ，198 ，199 ，199

[対象]タブ，100-101

エディター，100

オプション，199

対象選択リスト，100-101 ，199

難読化技法，31

文字列の暗号化オプション，199

文字列の暗号化セクション，198

文字列の暗号化対象選択リスト，199

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

並行ビルド，265

保護，13-14

保護概念，29

名前の改変，107-108

名前の変更，29-30，93-95，92-93，96，95-96，97-98，263-264，34，91，91-92，192，192-193，193-195，195，195-196，196-197

HTML 形式レポート，196-197

XML シリアル化，95-96

エディター，91

オプション，93-95，92-93，96，91-92，193-195

クラス，93-95

以前の動作，96

完全なクラス名，96

規則，192-193

参照規則リスト，195-196

対象除外リスト，97-98，96-97，195

難読化技法，29-30

明示的なメソッドのオーバーライドを導入する，96

名前の変更オプション，193-195

名前の変更セクション，192

名前の変更の参照規則，195-196

名前の変更の対象除外，97-98

名前の変更の対象除外リスト，195

名前の変更規則，192-193

名前の変更時、明示的なメソッドのオーバーライドを導入する，96

名前空間，117-118

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.

選択，117-118

名前空間による選択，117-118

名前空間の対象除外，214-215

名前空間の対象選択，222

要素，113-117

個別選択，113-117

例外，139 ，48

例外の追跡

ユーザー指定の例外レポート情報の収集，141-143

例外の追跡，139 ，48

例外レポート，139-140

例外操作，140-141

例外の追跡，139 ，48 ，139-140 ，140-141 ，141-143

Dotfuscator，139

ExceptionTrack 属性，139-140

ユーザー指定レポート，141-143

概要，48

通知，140-141

例外通知，140-141

例外の追跡属性，153

例外レポート，139-140

例外操作，140-141

例外通知，140-141

© 2013 PreEmptive Solutions, LLC. All Rights Reserved.