DevPartner 11.0.0

ユーザー ガイド

Micro Focus
575 Anton Blvd., Suite 510
Costa Mesa CA 92626

目次

はじめに

このマニュアルでは、Micro Focus® DevPartner® Studio ソフトウェアの使用を開始する方法について説明します。

対象読者

このマニュアルは、DevPartner Studioの新規ユーザーを対象としています。第1章でDevPartner Studioの概念の概要を示し、残りの各章でDevPartnerの個々のコンポーネントについて説明します。各コンポーネントの章では、最初に、新規ユーザーがDevPartner Studioを使用できるようにするための準備、設定、実行の手順について簡潔に説明します。

DevPartnerの前のバージョンのユーザーは、この「はじめに」から『DevPartnerインストール ガイド』までを参照して、前のDevPartnerバージョンとの違いを確認してください。

このマニュアルには、Professional EditionとEnterprise Edition、DevPartner for Visual C++ BoundsChecker Suiteなど、DevPartner Studioの全製品に関する情報が含まれています。

メモ： DevPartner for Visual C++ BoundsChecker Suite ではアンマネージ コードのみが分析されます。DevPartnerのメモリ分析機能、静的なコード分析機能、およびパフォーマンス エキスパート機能ではマネージ コードのみが分析されるため、これらの機能はDevPartner for Visual C++ BoundsChecker Suiteではサポートされていません。

このマニュアルでは、ユーザーがWindowsインターフェイス、Microsoft Visual Studio、およびソフトウェア開発の概念に精通していることを前提としています。

このマニュアルの内容

このマニュアルには、以下の章および付録が含まれています。

- 第**1**章「**DevPartner**の概要」では、DevPartnerの概念とコンポーネントについて説明します。
- 第 **2** 章「エラー検出」では、DevPartner を使用して C およびマネージ／アンマネージ C++ のコードのエラーを検出する方法について説明します。
- 第 **3** 章「静的なコード分析」では、Visual Basic と Visual C# のコードに含まれるさまざまなエラーを検出する場合に、DevPartner がどのように役立つかについて説明します。
- 第 **4** 章「自動コード カバレッジ分析」では、DevPartner を使用して、コードのどのくらいの部分がテストでカバーされているかを追跡する方法について説明します。
- 第 **5** 章「メモリに関する問題を検出する」では、DevPartner を使用して、メモリとオブジェクトの誤用が原因で発生するアプリケーションの異常を診断する方法について説明します。

- 第 **6** 章「自動パフォーマンス分析」では、最適化が必要なボトルネックとコードを検出する場合に、DevPartner がどのように役立つかについて説明します。
- 第 **7** 章「詳細パフォーマンス分析」では、システム パフォーマンスのさまざまな問題を分析する場合に、DevPartner がどのように役立つかについて説明しています。
- 第 **8** 章「**System Comparison**」では、コンピュータ システム間の違いを特定してアプリケーション開発の問題のトラブルシューティングを容易にするために、DevPartner がどのように役立つかについて説明します。
- 「付録 **A**」「**DevPartner Studio** サポートされるプロジェクト タイプ」では、DevPartner Studio の各機能でサポートされるプロジェクト タイプの一覧表を示します。
- 「付録 **B**」「コマンド ラインから分析を起動する」では、**`DPAnalysis.exe`** コマンド ライン インターフェイスについて説明します。
- 「付録 **C**」「分析セッションの制御」では、カバレッジ、メモリ、パフォーマンス、およびパフォーマンス エキスパートの各セッションに対する、セッション コントロール ファイルの作成について説明します。
- 「付録 **D**」「分析データを **XML** にエクスポートする」では、カバレッジ、パフォーマンス、およびパフォーマンス エキスパートのデータを XML ファイルにエクスポートする方法について説明します。

表記方法

このマニュアルの表記方法は以下のとおりです。

- スクリーン コマンドやメニュー名などは、[ ] で囲んで示します。例：

  [ツール **]** メニューから **[** オプション **]** を選択します。

- ファイル名は等幅フォントで示します。例：

  『DevPartner ユーザーガイド』マニュアル（**`Understanding DevPartner.pdf`**）は...

- コンピュータのコマンドとファイル名内の変数（ユーザーがインストール時に適切な値を指定するもの）は、イタリックの等幅フォントで示します。例：

  [移動先 **]** フィールドに「**`http://`*`servername`*`/cgi-win/itemview.dll`** 」と入力します。

補足情報

DevPartner Studioの特定のタスクに関するステップごとの手順説明は、機能レベルのオンライン ヘルプを参照してください。

機能レベルのオンライン ヘルプを利用して、DevPartner Studioの機能や使用方法を参照できます。

DevPartnerのコンポーネントについて、さらに詳しい情報が以下のように提供されています。Adobe Acrobat（.pdf）形式のマニュアルおよびDevPartner Studioリリース ノートがDevPartner StudioのMicro Focus SupportLine Webサイト製品ページ（http://supportline.microfocus.com/）にある**[**スタート**]** > **[**プログラム**]** > **[Micro Focus]** > **[DevPartner Studio]**メニューのInfoCenterオプションから入手できます。これらの資料はDevPartner StudioのDVDにも収録されています。

- DevPartner Studioのライセンスについては、『Distributed License Management ライセンス ガイド』を参照してください。
- DevPartnerの使用方法については、『DevPartner Studio ユーザーガイド』マニュアルを参照してください。
- 『DevPartner Studio クイック リファレンス』は、DevPartnerの機能の概要を提供します。製品をすぐに使用するためのアドバイスも記載されています。
- 『DevPartnerエラー検出ガイド』では、Micro Focus DevPartnerエラー検出ソフトウェアの使用方法を理解できるように、コンセプトと手順について説明します。
- 『DevPartner Studio リリース ノート』には、DevPartner Studioの既知の問題とテクニカル ノートが入っています。リリース ノート ドキュメントは DevPartner Studio インストール セットアップおよびDevPartner Studio InfoCenterから入手できます。

第 1 章

DevPartner の概要

この章では、DevPartner StudioProfessional Edition および DevPartner for Visual C++ BoundsChecker Suite の概要について説明します。このマニュアルでは、これら 2 つの DevPartner 製品の基礎となる概念について説明します。

メモ： DevPartner for Visual C++ BoundsChecker Suite では、アンマネージ コードのみが分析されます。DevPartner のメモリ分析機能、静的なコード分析機能、およびパフォーマンス エキスパート機能ではマネージ コードのみが分析されるため、これらの機能は DevPartner for Visual C++ BoundsChecker Suite ではサポートされていません。

DevPartner Studio とは？

DevPartner Studio は、エラーの自動検出、ソース コード分析、カバレッジ分析、メモリ分析、パフォーマンス プロファイリング、システム パフォーマンス分析、システム比較などの、プログラマの生産性を向上するためのさまざまなプログラマー生産性機能を備えています。

DevPartner では、さまざまな言語で作成された幅広いアプリケーション（マネージ アプリケーションとアンマネージ アプリケーションの両方）を分析できます。サポートされているすべてのプロジェクト タイプと言語のリストは、付録 A「DevPartner Studio サポートされるプロジェクト タイプ」を参照してください。

以下のセクションでは、DevPartner Studio の機能の概要について説明します。

エラー検出

DevPartner Studio では、マネージ プログラムとアンマネージ プログラムに対するエラーの自動検出が可能です。DevPartner エラー検出は BoundsChecker™ テクノロジに基づいて作成されており、Windows ベースのアプリケーションで以下のような検出が難しいエラーを検出できるように設計されています。

- メモリ リーク、リソース リーク、COM インターフェイス リーク
- Windows API コールの不正な使用
- メモリまたはポインタの不正な使用
- メモリ オーバーラン エラー
- 初期化されていないメモリの使用
- ダングリング ポインタの使用
- .NET ファイナライザのエラー

DevPartner エラー検出は、プロセス作成の瞬間からプロセスがメモリからアンロードされる最後の瞬間まで、アプリケーションを監視します。DLL のすべてのロードとアンロード、静的なコンストラクタとデストラクタ、およびアプリケーションの通常のフローを監視できます。また、アプリケーションの特定のファイルまたは特定の部分をフィルタして、特定の問題の解決に必要な情報のみを収集するように DevPartner エラー検出を調整することもできます。

DevPartner エラー検出の詳細については、「エラー検出」（23 ページ）を参照してください。

静的なコード分析

開発者は、DevPartnerを使用することによって、Visual Studio内で規格に準拠したVisual BasicとC#のコードを作成できます。DevPartnerでは、.NET Frameworkにおけるプログラミングやネーミングの違反の識別、メソッド コール構造の分析、コードの全体的な複雑度の追跡が行われます。

DevPartnerソフトウェアでは、以下のようなさまざまなコーディング エラーが検出されます。

- 変数名の不統一
- コーディング基準の違反
- Win32 APIバリデーション エラー
- 一般的なロジック エラー
- .NETの移植性の問題
- 構造化例外処理エラー

また、DevPartnerでは、拡張可能で広範なルール セットを使用して.NET環境で動作しない構造を特定することによって、Visual Basicのレガシー コードの移植も支援します。

DevPartnerの静的なコード レビューの詳細については、「静的なコード分析」（59ページ）を参照してください。

カバレッジ分析

開発者やテスト エンジニアは、DevPartnerカバレッジ分析を使用することによって、アプリケーションのすべてのコードをテストできます。カバレッジ分析を有効にしてテストを実行すると、テストによってカバーされているすべてのアプリケーション、コンポーネント、イメージ、メソッド、関数、モジュール、およびコードの各行がDevPartnerによって追跡されます。テストが終了すると、どのコードが実行されてどのコードが実行されていないかについての情報が表示されます。

DevPartnerによって、Webアプリケーションや ASP.NETアプリケーションなどのマネージ コード アプリケーション、およびアンマネージC++アプリケーションのカバレッジ データが収集されます（サポートされているすべてのテクノロジのリストについては、付録 A「DevPartner Studioサポートされるプロジェクト タイプ」を参照してください）。

カバレッジ分析機能とエラー検出機能は同時に実行できます。テストでカバーしたコードの割合を知ることで、エラー検出結果の信頼性を高めます。

コード カバレッジ分析の詳細については、「自動コード カバレッジ分析」（111ページ）を参照してください。

メモリ分析

DevPartnerは、マネージ Visual Studioアプリケーションがメモリを割り当てる方法を分析します。メモリ分析を有効にしてアプリケーションを実行すると、オブジェクトまたはクラスによって消費されたメモリの量が示され、オブジェクトをメモリに保持している参照が追跡され、メソッド内でメモリ割り当てを行うソース コード行が識別されます。

さらに、DevPartnerでは、コンテキストに応じてメモリのデータが表示されます。これにより、コード内のオブジェクト参照のチェーンやメソッドのコール シーケンスをたどることができます。コンテキストに応じてメモリのデータが表示されるため、プログラムによるメモリの使用状況をより詳細に理解でき、メモリ使用を最適化するために必要となる重要な情報を入手できます。

メモリ分析の詳細については、「メモリに関する問題を検出する」（135ページ）を参照してください。

パフォーマンス分析

DevPartner パフォーマンス分析機能では、コードが分析されて、パフォーマンスのボトルネックがないかどうかが調査されます。パフォーマンス分析では、パフォーマンスのボトルネックがソース コードの行単位で特定され、アプリケーションにおけるサードパーティ製コンポーネント、オペレーティング システム、および .NET Framework の使用方法についてメソッドレベルでの分析結果が示されます。

DevPartner では、Microsoft Visual Studio 2010、Visual Studio 2008、および Visual Studio 2005 でのパフォーマンス プロファイリングがサポートされています (サポートされているすべてのテクノロジのリストについては、付録 A「DevPartner Studio サポートされるプロジェクト タイプ」を参照してください)。

コードの重要な部分のパフォーマンスを向上させるには、DevPartner パフォーマンス分析を使用してパフォーマンスのボトルネックを検出し、行った改善内容がパフォーマンスの向上に役立っているかどうかを検証します。

アプリケーションのパフォーマンス分析の詳細については、「自動パフォーマンス分析」 (183 ページ) を参照してください。

詳細パフォーマンス分析

DevPartner パフォーマンス エキスパート機能では、DevPartner のパフォーマンス分析機能よりも詳細なパフォーマンス プロファイリングが行われます。パフォーマンス エキスパートでは、マネージ コードの Visual Studio アプリケーションに対して、解決が難しい以下のような問題についてのより詳細な分析が行われます。

- CPU /スレッドの使用
- ファイルとディスクの I/O
- ネットワーク I/O
- 同期待機時間

パフォーマンス エキスパートでは、実行時にアプリケーションが分析され、コード内の問題があるメソッドが検出されます。メソッド内の個別の行の詳細を表示したり、呼び出しの親子関係を調査したりすることができるため、問題を修正するための最善の方法を決定する場合に役立ちます。アプローチ方法を決定する場合には、パフォーマンス エキスパートからソース コード内の問題がある行に直接ジャンプできるため、問題を迅速に修正できます。

詳細については、「詳細パフォーマンス分析」 (211 ページ) を参照してください。

System Comparison

DevPartner System Comparison ユーティリティでは、2 つのコンピュータ システムを比較したり、コンピュータの現在の状態と過去の状態を比較したりすることで、アプリケーションが以下のように動作する原因を突き止めることができます。

- 特定のコンピュータでは動作するのに、別のコンピュータでは動作しない。
- コンピュータによって動作が異なる。
- 以前動作したコンピュータで動作しなくなった。

システムを比較するために、System Comparison では、コンピュータ システムに関する情報 (インストールされている製品、システム ファイル、ドライバ、その他の数多くのシステム特性など) を含むスナップショット ファイルと呼ばれる XML ファイルが作成されます。スナップショット ファイルが比較されて、それらの差異がレポートされます。

DevPartnerの他の機能とは異なり、System ComparisonはVisual Studio環境には統合されていません。System Comparisonはスタンドアロン ユーティリティとして動作するため、ターゲット システムへの影響が最小限に抑えられます。

System Comparisonは、システムのスナップ ショットを毎日夜間に取得するサービスと、手動でスナップショットを取ったり、スナップ ショット間の違いを特定したりできるユーザーインターフェイスで構成されています。System Comparisonには、コマンド ライン インターフェイスとソフトウェア開発キット（SDK）も含まれています。ソフトウェア開発者は、SDKを使用して比較についての追加情報を収集して、配置されるアプリケーションにスナップショット機能を埋め込むことができます。

System Comparisonユーティリティの詳細については、「System Comparison」（245ページ）を参照してください。

DevPartnerおよびVisual Studio

DevPartnerは、Visual Studio環境にシームレスに統合されています。シームレスに統合されているため、アプリケーションを作成およびデバッグしながらDevPartnerの機能を簡単に使用できます。アプリケーションを開発しながら、開発環境を離れることなく、頻繁にコード分析を実行できます。

DevPartnerでは、Visual Studio環境内でのアプリケーション開発が同時にサポートされます。これは、開発者が古いMicrosoft環境のコードを最新の.NET Frameworkに移行する場合に役立ちます。

以下の表に、さまざまなVisual Studio環境で使用可能なDevPartner機能を示します。

表1-1. DevPartner Studio Professional Editionでインストールされる機能

Microsoft Visual Studio 2010	Microsoft Visual Studio 2008	Microsoft Visual Studio 2005
パフォーマンス分析	パフォーマンス分析	パフォーマンス分析
カバレッジ分析	カバレッジ分析	カバレッジ分析
エラー検出	エラー検出	エラー検出
静的なコード分析	静的なコード分析	静的なコード分析
メモリ分析	メモリ分析	メモリ分析
パフォーマンス エキスパート	パフォーマンス エキスパート	パフォーマンス エキスパート

表1-2. DevPartner for Visual C++ BoundsChecker Suiteでインストールされる機能

Microsoft Visual Studio 2010	Microsoft Visual Studio 2008	Microsoft Visual Studio 2005
パフォーマンス分析	パフォーマンス分析	パフォーマンス分析
カバレッジ分析	カバレッジ分析	カバレッジ分析
カバレッジ分析	カバレッジ分析	エラー検出

Visual Studio のメニューとツールバー

Visual Studio には、DevPartner によってメニューといくつかのツールバーが追加されます。また、コンテキスト（右クリック）メニューを含むいくつかの Visual Studio メニューにメニュー コマンドが追加されます。メニュー コマンドとツールバーから、セッション コントロール、静的なコード レビューのルール、オプション、およびインストゥルメンテーション コントロールにアクセスできます。

DevPartner によって Visual Studio に追加されるツールバーを使用すると、DevPartner の機能に迅速にアクセスできます。以下の図は、Visual Studio 2010 の DevPartner ツールバーを表しています。

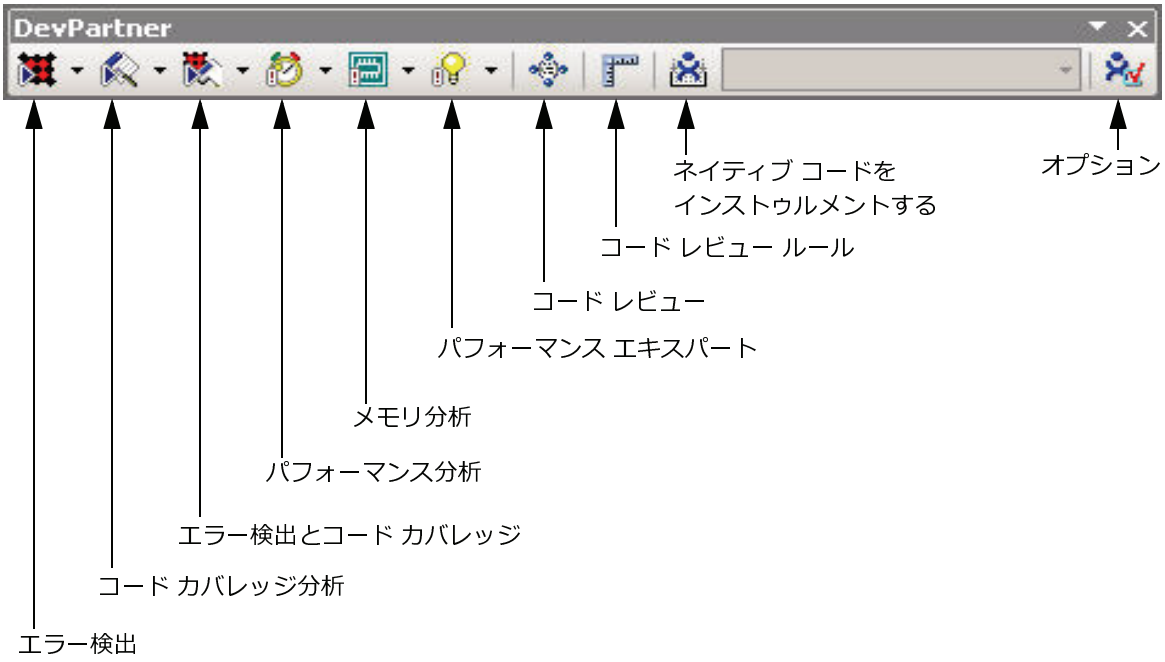


図 1-1 DevPartner ツールバー

また、IDE には、DevPartner のセッション コントロール ツールバーも配置されます。セッション コントロール ツールバーは、カバレッジ分析機能、メモリ分析機能、パフォーマンス分析機能、およびパフォーマンス エキスパート機能がアクティブである場合にアクティブになります。

図 1-2 セッション コントロール ツールバー

セッション コントロール ツールバーは、3 つのアイコンとプロセス リストから構成されています。

- データ収集を停止して、最終的なデータのスナップショットを取ります。
- データのスナップショットを取ります。
- このアイコンをクリックするまでに収集されたデータをクリアします。

アプリケーションで複数のプロセスが使用されている場合にプロセス リストでプロセスを選択すると、そのプロセスのみを対象にデータ収集が実行されます。

DevPartnerでは、メニューやツールバーが表示される他に、Visual Studioのドッキング可能なウィンドウとペインを使用して分析セッションの結果が表示されます。また、ソリューション エクスプローラを使用してセッション ファイル名が表示されます。さらに、Visual Studioの[オプション]、[ソリューションのプロパティ]、および[プロジェクトのプロパティ]に、DevPartnerコード分析処理を設定するためのページが追加されます。

Visual StudioでDevPartnerを使用する

Visual StudioでDevPartnerを使用するための一般的なワークフローでは、以下のような汎用のタスクを1つまたは複数実行します。

- Visual Studioで、ソリューションを開くか、作成します。
- コード分析処理のオプションを設定します。
- DevPartnerのメニューまたはツールバーで、実行する分析を有効化します。
- アプリケーションを実行します。
- DevPartnerによって返されたセッションの結果を表示します。

DevPartnerには、アプリケーションの監視対象部分の選択、表示するデータの選択、不要な情報を除去するためのフィルタの作成において、高い柔軟性が備えられています。

また、DevPartnerには、コマンド ラインから数多くの機能を実行するオプションも備えられています。コマンド ラインから実行するオプションによって、DevPartnerの機能を夜間ビルドのスモーク テストなどの自動バッチ プロセス処理で使用できるようになります。

統合されたオンライン ヘルプ

DevPartner Studioの各機能には、広範なオンライン ヘルプが用意されています。操作方法や参照情報については、まずオンライン ヘルプを参照してください。

DevPartnerのオンライン ヘルプはVisual Studioの他のヘルプと同じ形式であるため、Visual Studioヘルプに表示されます。DevPartner Studioヘルプには、各DevPartnerコンポーネントのボリュームが含まれています。

Visual Studio Team Systemのサポート

Visual Studio Team Systemは、Visual Studioソフトウェア開発プロジェクトのバージョン管理、障害管理、プロセス管理を行うMicrosoftのソフトウェアです。DevPartner Studioでは、Team Systemクライアント ソフトウェアがインストールされており、Team Foundation Serverに接続可能な場合に、Microsoft Visual Studio Team Systemがサポートされます。

DevPartner Studioでは、Visual Studio Team Systemへのバグタイプの作業項目の送信がサポートされています。バグを送信すると、DevPartnerによって、選択されたセッション データが作業項目フォームに自動的に入力されます。DevPartnerからバグを送信するには、アクティブなTeam Systemプロジェクトでタイプがバグの作業項目がサポートされている必要があります。DevPartnerでは、バグ タイプの作業項目のみにデータが自動的に追加されます。

DevPartnerデータを含む作業項目は、DevPartnerセッション ファイル内の以下のいずれかのビューから送信できます。

- カバレッジ、メモリ、パフォーマンス分析のセッション ファイル内、またはパフォーマンス エキスパート セッション ファイル内のメソッド リストまたはメソッド テーブル

- コード レビューの [問題] タブまたは [ネーミング] タブ
- 任意の [エラー検出] タブのエラーやリークのリスト、またはエラー検出の [モジュール] タブや [.NET パフォーマンス] タブのインスタンスのリスト

DevPartner から Team System の作業項目を送信するには、セッション ファイルでメソッドまたはその他の項目を右クリックして、**[** 作業項目の提出 **]** を選択します。DevPartner によって、[Title] フィールドおよび [Description] フィールドまたは [Symptom] フィールドに値が入力されます。他に必要なデータがあれば入力し、作業項目を保存します。

メモ : Visual Studio でチーム エクスプローラのコンテキスト メニューを使用する場合、作業項目にはセッション データは自動的に入力されません。

DevPartner Studio から Team System へのデータの送信の詳細については、このマニュアルの各章にある Visual Studio Team System についてのセクションを参照してください。開発およびプロジェクト管理活動をサポートするための Team System の使用方法については、Microsoft Visual Studio Team System のマニュアルを参照してください。

ターミナル サービスとリモート デスクトップを使用する

DevPartner Studio では Windows のターミナル サービスがサポートされています。ターミナル サービスを使用すると、直接コンピュータを使用して行うことができるすべての操作を実行できます。以下に、例を示します。

- DevPartner のオプションをリモート システムで設定する。
- リモート システムで分析を有効または無効にする。
- リモート システムで実行されているアプリケーションをプロファイルする。

ライセンス

ターミナル サービス接続には、ユーザー表示ごとに 1 つの DevPartner 同時ライセンスが必要です。ターミナル サービス接続経由で接続されるサーバーでは、DevPartner Studio リモート サーバー ライセンスは必要ありません。

ターミナル サービスで複数のセッションを実行する

ターミナル サーバーでは複数の DevPartner セッションを同時に実行できます。複数のセッションの起動は、単一のユーザーによる場合も、複数のユーザーによる場合もあります。DevPartner では、ワークスペースの設定はユーザー単位で保存されるため、単一ユーザーがコンソールの 2 つのインスタンスを起動した場合、両方のインスタンスが同じワークスペース設定を共有します。複数のユーザーがカバレッジ分析の複数のインスタンスを起動する場合、ワークスペースの設定は、インスタンスごとに個別に設定できます。

収集されるデータには、ターミナル サーバー上の全ユーザーのサーバー プロセスにおける動作が含まれています。セッション中、データ収集に注意を集中させるために、監視対象のプロセスを使用したり、監視対象のターゲットを呼び出すような余分なアプリケーション アプリケーションは排除または制限してください。

第2章
エラー検出

この章には、2つのセクションが含まれています。1つめのセクションでは、はじめてのユーザーを対象として、エラー検出を実行する簡単な手順について説明します。2つめのセクションでは、DevPartnerエラー検出機能を詳細に把握するための参考情報を示します。

エラー検出に関するタスクベースの追加情報については、DevPartnerのオンライン ヘルプを参照してください。より詳細な情報については、DevPartnerソフトウェア インストールに付属のPDF形式の『エラー検出ガイド』を参照してください。

エラー検出とは

DevPartnerは、CとC++開発の総合的なデバッグ ソリューションです。DevPartnerエラー検出による頻繁なチェックをアプリケーション開発サイクルに組み込むことによって、エラーのない安定したコードを作成できます。DevPartnerを使用すると、開発プロセスにかかる時間を増加させることなく、エラー検出と分析を自動化できます。以下の機能を使用すると、従来からのデバッグ手法とテスト手法では検出できない見つかりにくいバグを特定できます。

- 総合的なエラー検出
- 柔軟なデバッグ環境
- Visual Studioデバッガとの統合
- Microsoft Visual Studioとの統合
- 高度なエラー分析
- エラー検出のオープン アーキテクチャ

簡単な手順でエラー検出を使用する

以下の準備、設定、実行の手順に従ってDevPartnerエラー検出を使用できます。

すぐに使い始めるには、影付きのボックス内に示された手順に従います。影付きのボックス内に説明されている内容の詳細については、ボックスの下の追加説明を参照してください。

DevPartner Studioでアプリケーションを分析する場合、昇格されたシステム権限は必要ありません。DevPartnerでのアプリケーションの分析には、アプリケーションの作成とデバッグに使用するシステム権限で十分です。

準備：エラー検出の分析範囲を決定する

コード上でのDevPartnerエラー検出の実行方法や、検索が必要なエラーとメモリ リークの種類について検討します。

メモ： DevPartnerエラー検出では、ターゲット アプリケーションごとにデータ ファイルが作成されます。エラー検出を開始する前に、ターゲット実行可能ファイルを含むフォルダへの書き込みアクセス権があることを確認する必要があります。

以降の手順では、以下の事項を前提としています。

- ソリューション内に、アンマネージ ソース コードが含まれています。
- サポートされているVisual Studioのリリースでエラー検出を実行しています。

DevPartnerエラー検出でサポートされているすべてのプロジェクト タイプのリストは、「エラー検出でサポートされているプロジェクト タイプ」 170ページ）を参照してください。

セッションの実行方法を決定する

ユーザーの状況のニーズに応じて、いくつかの方法でDevPartnerエラー検出を実行できます。

- Microsoft Visual Studio 内から、またはスタンドアロン アプリケーションを使用して、ルーチン コード検証プロセス（日次または週次）の一部として対話的にエラー検出を実行します。
  - Visual Studio環境では、すべてのエラー検出機能にアクセスできます。ユーザーはDevPartnerの設定を構成してプログラムをチェックし、検出されたエラーを確認できます。
  - DevPartnerは、Visual Studioから分離したスタンドアロン アプリケーションとして実行できますが、その場合、ユーザーはVisual Studioエディタにアクセスしてコードを編集することができません。
- **bc.exe** を使用して、バッチ ファイルまたはコマンド ラインから実行するエラー検出を自動化します。
  - コマンド ラインからDevPartnerを起動すると、自動テスト スクリプトを設定できます。詳細については、「コマンド ラインからエラー検出を実行する」 56ページ）を参照してください。
- 主要な開発マイルストーンで詳細な検証を行うために、FinalCheck を使用してコードをインストゥルメントします（Visual Studioのみ）。

検索するエラー タイプを決定する

エラー検出を使用すると、ユーザーは、コード内で発生する可能性のある各種のエラーとリークを幅広く検索し、エラーを追跡することもできます。

- COM オブジェクト エラーがコードにないことを確認するには、COM オブジェクトの追跡を有効にします。
- 同期オブジェクトが正しく使用されていることを確認する、またはときどきデッドロックされ、原因が不明のアプリケーションに対してデッドロック分析を有効にします。
- カスタム アロケータの実装によってリークまたはエラーが発生していないことを確認するには、カスタム アロケータが対象となるように、メモリ追跡システムを拡張します。カスタム アロケータの説明を記述するには、使用しているアロケータについての説明情報を `UserAllocators.dat` ファイルに追加します。『エラー検出ガイド』の「ユーザーが作成したアロケータの使用」を参照してください。

設定：オプションと設定を構成する

DevPartner エラー検出は、特定のタイプのエラーをレポートするようにカスタマイズできます。また、不要な項目を無視またはフィルタで除外することもできます。

> この手順では、DevPartner のデフォルトのプロパティとオプションを使用します。設定を変更する必要はありません。

実行している DevPartner エラー検出の状態に応じて、複数のメニュー オプションを使用して [設定] ダイアログ ボックスにアクセスできます （「[設定] ダイアログ ボックスを使用する」41 ページ）を参照）。

デフォルトで、エラー検出は単純なメモリ リーク、数種類のメモリ エラー、およびリソース リークを検出します。デフォルト設定を編集すると、以下のタイプのエラー、リーク、およびイベントを含む各インスタンスも検出できます。

- API コールとバリデーション エラー
- 潜在的なデッドロックの状況
- COM インターフェイス リーク
- メモリ割り当てと割り当ての解放
- Windows メッセージとその他の重要なイベント （「Windows メッセージとイベント ログを追跡する」（54 ページ）を参照）

さらに、FinalCheck を使用するように DevPartner を構成できます。FinalCheck を使用してエラー検出を行うと、C または C++ アプリケーションをインストゥルメントし、エラーが発生した正確なステートメントを特定できます。FinalCheck では多くのリソースを使用し、実行に時間がかかりますが、検出が難しいメモリ、ポインタ、リークのエラーを検索して、正確な位置を特定します。

エラー検出の設定では、特定のエラーとリークに対してエラー検出を構成するだけでなく、以下の操作が可能です。

- エラー検出パラメータを定義する。
- 表示に使用するフォントと色を変更する。
- 構成ファイルにパラメータを保存して再利用する。
- 現在のセッションに別の構成ファイルをロードする。

実行：エラー検出でソリューションを実行する

これで、DevPartner エラー検出でソリューションを実行する準備ができました。

1. Visual Studio でソリューションを開きます。
2. **[DevPartner]** > **[** エラー検出を選択して開始 **]** を選択します。
3. プログラムでエラーをチェックする部分を実行します。
   エラー検出では、重大なエラーを検出するたびに **[** 検出されたプログラム エラー **]** ダイアログ ボックスがポップアップ表示されます （図 2-1 を参照）。重大ではないその他のエラーや一般的なエラーは、あとからそれらに対処できるように記録されて、エラー検出のメイン ウィンドウの左上に配置されている）検証結果ペインに表示されます。

[検出されたプログラム エラー]ダイアログ ボックス（「[検出されたプログラム エラー]ダイアログ ボックスを使用する」 33ページ）を参照）では、エラーの説明とその下にコール スタック情報が表示され、最後にエラーが検出されたコード セグメント（使用可能な場合）が表示されます。検出されたエラーのさらに詳しい説明を表示するには、[説明]ボタンをクリックします。

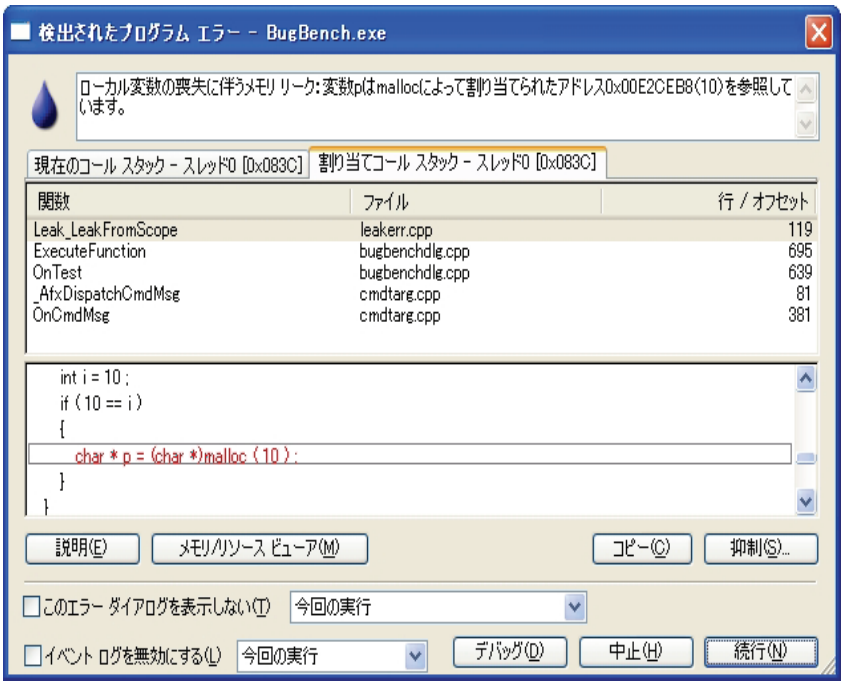


図 2-1 [検出されたプログラム エラー]ダイアログ ボックス

4 [検出されたプログラム エラー]ダイアログ ボックスが表示された場合、以下のいずれかの方法で応答して、引き続きプログラムを実行します。[検出されたプログラム エラー]ダイアログ ボックスが表示されない場合は、この手順をスキップできます。

- ◇ **[説明]** – エラーの詳細説明およびエラーを解決するための対処方法を説明します。
- ◇ **[抑制]** – このエラーの情報が入力された状態で、[抑制]ダイアログ ボックスが開きます。エラーを抑制すると、それ以降は、このエラーが出現してもエラー検出の対象になりません。「[抑制]と[フィルタ]ダイアログ ボックスを理解する」 36ページ）を参照してください。
- ◇ **[デバッグ]** – Visual Studio デバッガでエラーが発生した行のコードを開きます。
- ◇ **[中止]** – プログラムを停止し、検証結果ペインを表示します。
- ◇ **[続行]** – エラーを確認して、次の操作に進みます。セッション完了後にさらに確認するために、検証結果ペインにエラーが表示されます。

5 チェックが終了したら、プログラムを停止します。

プログラムが正常に終了しないことがあります。その場合は、エラーに対して必要な修正を行ってからプログラムを閉じます。プログラムを閉じるには、以下の方法のいずれかを使用します。

- ◇ [検出されたプログラム エラー]ダイアログ ボックスの**[中止]**をクリックします。
- ◇ **[デバッグ]**メニューから**[デバッグ停止]**を選択します。
- ◇ アプリケーションを停止します。

これで、基本のエラー検出セッションの実行を完了しました。検証結果ペインを確認して、検出したエラーとリークに基づいてデータを分析します。

検証結果ペインのデータを分析する

エラー検出のメイン ウィンドウの左上にある検証結果ペイン（図 2-2 を参照）では、一連のタブを使用して各種の情報を確認できます。

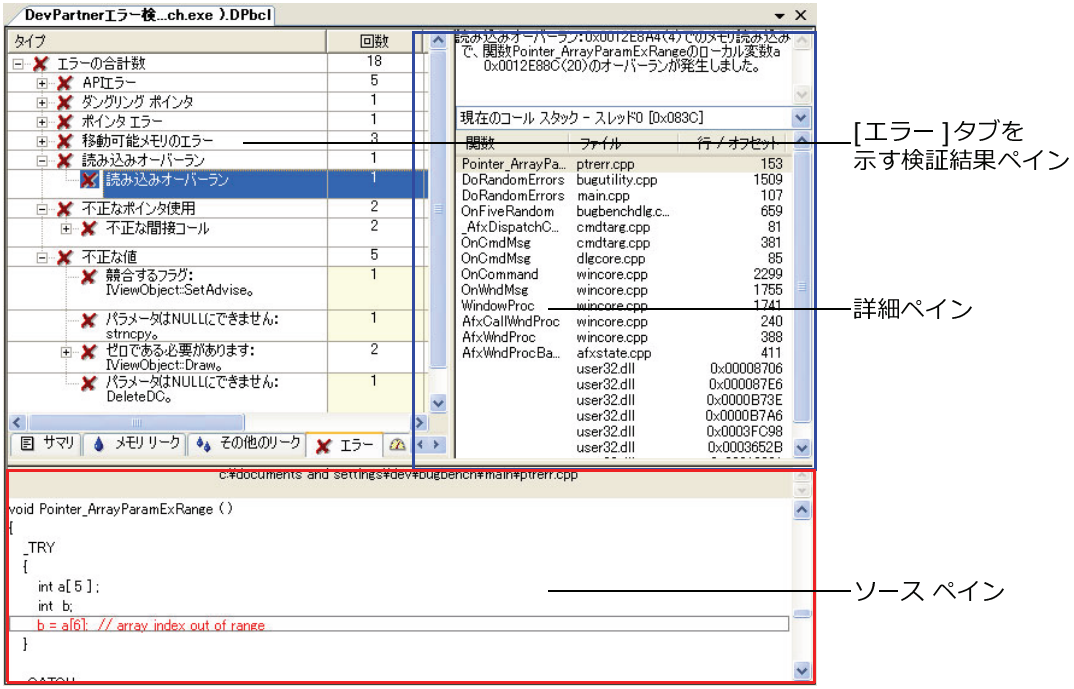


図 2-2 エラー検出のメイン ウィンドウ

セッション完了後、[検証結果]ペインの [サマリ]タブを使用してデータ確認を開始します（図 2-3 を参照）。

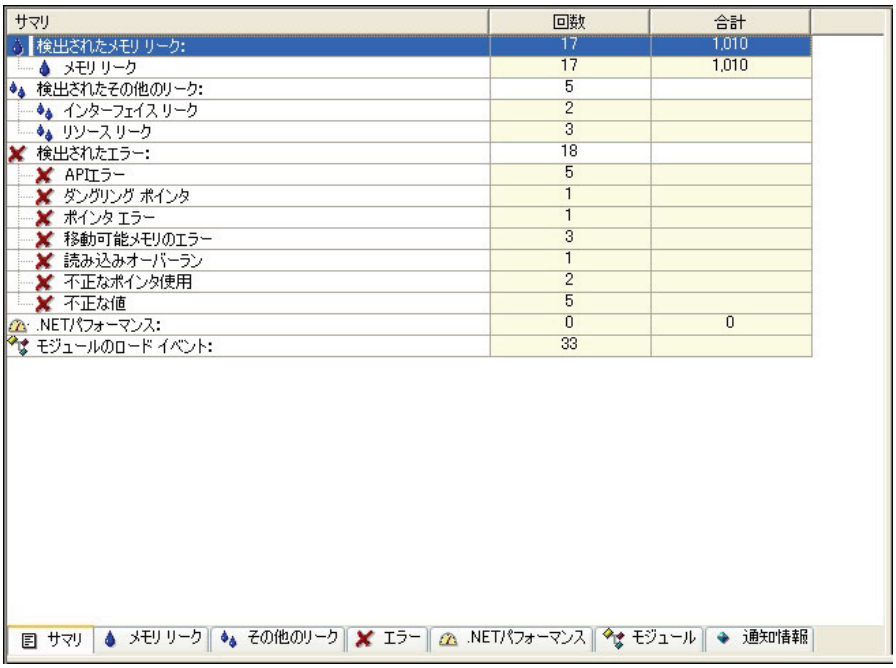


図 2-3 [サマリ]タブを表示した検証結果ペイン

[サマリ]タブには、現在のセッション内で検出したすべてのエラーとリークの概要が表示されます。イベントをダブルクリックして、選択したイベントの詳細を示すタブに移動します。

1 検出したエラーとリークの概要について、検証結果ペインの**[サマリ]**タブで確認します。

2 **[サマリ]**ペインのリストからエラーをダブルクリックします。

エラーまたはリークのタイプに関するタブが表示されます。エラーを診断してそれらを修正できるように、このタブではエラーがカテゴリごとに分類され、繰り返し発生するエラーに着目しやすくなっています。

3 1つのカテゴリを完全に展開して、特定のリーク、エラー、またはイベントを選択します。

リストの一番上のレベルに、リーク、エラー、およびイベントのカテゴリが表示されています。カテゴリを完全に展開すると、検出した個々のエラー、リーク、およびイベントが表示されます （図 2-4 29ページ）を参照。

図 2-4 選択したエラーを表示する [エラー] タブ

検証結果ペインには、[メモリ リーク]、[その他のリーク]、[エラー]、[.NET パフォーマンス]、[モジュール]、および [通知情報] のタブがあります（図 2-4、19 ページ）を参照）。これらのタブから、提示されたデータの分類や評価を行うためのその他のアクションを実行することもできます。

検証結果ペインのいずれかのタブで特定のエラーを右クリックし、[ソースの編集] を選択して、特定のエラーのソース コードにアクセスすることもできます。これにより、ソース ペインのエラーが発生したコード行にソース ファイルが開きます。

- これらのタブ上でデータを並べ替えるには、カラムのヘッダー（[その他のリーク] タブの [タイプ]、[回数]、または [デアロケータ] など）をクリックします。
- タブ上でイベントに関する詳細情報を表示するには、イベントを右クリックして [説明] を選択します。
- アプリケーション内の他のイベントのコンテキストと共にイベントを表示するには、イベントを右クリックして [通知情報で検索] を選択します。[通知情報] タブでは、アプリケーション内で発生したすべてのイベントを時系列順にリストします。

4 詳細ペインを確認します（図 2-2、17 ページ）を参照）。

詳細ペインの上部に、選択したエラーの詳細な説明が示されます。説明の下には、現在のコール スタックが表示されます。

詳細ペインは、エラー検出ウィンドウの右上セクションにあります。詳細ペインに表示される情報は、現在選択しているイベントに応じて異なります。エラーまたはイベントは常に詳細に説明されていますが、詳細ペインでは、コール スタック、P/Invoke 使用回数グラフ、COM 使用回数なども表示できます。

5 コール スタックを確認します。

コール スタックでは、エラーのタイプに応じて、エラーまたはリークが検出された場所やそれらが割り当てられた場所を示すことができます。複数のコール スタックが使用可能な場合は、ドロップダウン リストを使用してコール スタックを切り替えできます。

エラー検出のメイン ウィンドウの下部は、ソース ペインと呼ばれます。ソース ペインには、現在選択しているコール スタックに関連付けられているソース ファイルが表示されます。詳細ペインで別のコール スタックを選択すると、ソース コードが変わります。

6 ソース ペインを確認します。

ソース ペインには、現在選択しているコール スタックに関連付けられたコードが表示され、エラーやリークが検出された場所または割り当てられた場所が強調表示されています。

7 ソース コードを確認して、エラーまたはリークが検出された理由を判断します。

8 ソース ペインを右クリックして、**[**ソースの編集**]** を選択します。

ソース ファイルが Visual Studio エディタで開き、ソース ペインの表示と同じ位置が示されます。

9 ソース コードを編集してエラーを修正し、ソリューションを保存します。

エラー検出を使用して、エラーまたはリークをエディタ内で識別して位置を特定し、コード内で解決できました。

セッション ファイルを保存する

ヒント : **[**全般**]** 設定を使用して、**[**ファイル**]>[**閉じる**]** を選択することで、セッション ファイルの保存を求めるメッセージを表示するようエラー検出を設定できます。

セッション ファイルに結果を保存すると、以下が可能になります。

- 以前に発生した種類のリークおよびエラーを見直す。
- セッション データを XML へエクスポートする（「データを XML にエクスポートする」55 ページ）を参照）。エクスポートによりデータと他のユーザーと共有したり、セッション間でデータを比較できます。さらに、傾向を示すデータベースを構築できます。
- このセッションで検出されたエラーをいつでも修正する。

セッション ファイルは、拡張子 `.dpbcl` が付いて保存されます。デフォルトでは、実行可能ファイルと同じフォルダに保管されます。

10 セッション ファイルを保存するには、**[**ファイル**]>[**選択したファイルに名前を付けて保存**]** を選択します。

11 [ファイルに名前を付けて保存] ダイアログ ボックスを使用して、セッション ファイルの場所と名前を選択します。

エラー検出を Visual Studio で実行しているか、スタンドアロン アプリケーションで実行しているかにより、セッション ファイルを保存する手順が異なります。

Visual Studioからセッション ファイルを保存する

1 **[** ファイル **]** > **[** 選択したファイルに名前を付けて保存 **]** を選択します。

2 [ファイルに名前を付けて保存] ダイアログ ボックスを使用して、セッション ファイルの場所と名前を選択します。

スタンドアロン アプリケーションからセッション ファイルを保存する

1 **[** ファイル **]** > **[** セッション ログに名前を付けて保存 **]** を選択します。

2 [ファイルに名前を付けて保存] ダイアログ ボックスを使用して、セッション ファイルの場所と名前を選択します。

これで、この章の準備、設定、実行のセクションは終了です。エラー検出セッション実行のメカニズムの基礎が理解できました。詳細情報については、引き続きこの章の残りの部分を参照してください。高度なトピックの詳細な議論については『エラー検出ガイド』で参照してください。または、タスク中心の情報についてはDevPartnerオンライン ヘルプを参照してください。

ActiveCheckまたはFinalCheckを使用する場合を判断する

DevPartnerでは、ActiveCheck™とFinalCheck™の両方のテクノロジを使用して、Windowsアプリケーションを分析できます。

ActiveCheckを理解する

ActiveCheckテクノロジとは、ソース コードをインストゥルメンテーションせずに、エラー、リーク、およびイベントをチェックする標準処理のことです。コードのインストゥルメンテーションが必要ないため、ActiveCheckでエラーを検出する際は、ユーザーがコンパイルやリンクをやり直す必要はありません。ActiveCheckは、どのエラー検出セッションでも有効化されています。

ActiveCheckでは、以下の操作を実行できます。

- 実行時にAPIバリデーション エラーをレポートする
- プログラムの終了時にメモリ リークとリソース リークをレポートする
- メモリまたはリソースが割り当てられている行、またはエラーが発生している行にエラーを絞り込む
- 潜在的なデッドロックを特定する

DevPartnerはActiveCheckでエラー検出を使用する場合に、プログラムを分析します。これはAPIコール、メモリの割り当てと解放、Windowsメッセージ、その他の重大なイベントを監視します。このデータを使用してエラーを検出したり、プログラムの実行の詳細な追跡を行ったりできます。使用可能なソース コードがないプログラムに対しても、チェックが可能です。

ActiveCheckはコンパイルや再リンクによるオーバーヘッドを伴わないため、日次処理として使用できます。ソフトウェア開発サイクルを通してActiveCheckを使用すると、APIバリデーション エラー、デッドロック、リソース リーク、およびCOMインターフェイス リークが検出されます。

ï 2-1 および ï 2-2 に、ActiveCheck で検出されるエラーをリストします。

表 2-1. ActiveCheck で検出される API、COM、およびメモリ エラー

API エラーと COM エラー	メモリ エラー
• COM インターフェイス メソッドの失敗 • 不正な引数 • 不正な COM インターフェイス メソッド引数 • パラメータ範囲エラー • スレッドの不正な使用 • Windows 関数が失敗した場合 • Windows 関数が実装されていない場合	• ダイナミック メモリ オーバーラン • 解放されたハンドルがすでにアンロックされている場合 • ハンドルがすでにアンロックされている場合 • メモリ割り当ての競合 • アンロックされたメモリ ブロックをポインタが参照 • スタック メモリ オーバーラン • スタティック メモリ オーバーラン

表 2-2. ActiveCheck で検出されるデッドロック関連、.NET、ポインタ、およびリーク エラー

デッドロック関連エラー	.NET エラー	ポインタ エラーとリーク エラー
• デッドロック • 潜在的なデッドロック • スレッドのデッドロック • クリティカル セクションのエラー • セマフォ エラー • ミューテックス エラー • イベント エラー • ハンドル エラー • リソースの使用とネーミング エラー • 問題のある可能性が高いリソース使用状況 • Windows イベント エラー	• ファイナライザ エラー • `GC.Suppress` finalize が呼び出されていない場合 • Dispose 属性エラー • 処理されていないネイティブの例外がマネージ コードに渡された場合	• インターフェイス リーク • メモリ リーク • リソース リーク

FinalCheck を理解する

FinalCheck は、コードのコンパイル時にコードに対して診断ロジックを挿入する特徴的なテクノロジです。FinalCheck を使用すると、DevPartner はエラーが発生したステートメントを正確に特定できます。

プロジェクトの重要なマイルストーンに対してや見つかりにくいエラーの検出のために、FinalCheck を使用します。FinalCheck は、ActiveCheck で検出されるすべてのエラーに加えて、以下の表にリストしたエラーを検出する ActiveCheck のスーパーセットです。

表 2-3. FinalCheck で検出されるその他のエラー

メモリ エラー	ポインタ エラーとリーク エラー
• バッファ読み込みオーバーフロー • 未初期化メモリからの読み込み • バッファ書き込みオーバーフロー	• 範囲を超えた配列の読み込み • 有効範囲外を示すポインタのコピー • ダングリング ポインタの演算 • 非関連ポインタの演算 • 関数を示していない関数ポインタ • メモリ領域の解放に伴うメモリ リーク • リークによるリーク • メモリの再割り当てに伴うメモリ リーク • ローカル変数の喪失に伴うメモリ リーク • ローカル変数を指すポインタを返している場合 • アンワインドによるリーク • モジュール アンロードによるリーク • スレッドの終了によるリーク

ActiveCheck と FinalCheck を比較する – 例

DevPartner は、new または malloc を使用するメモリ ブロック割り当てを記録し、ポインタをローカル変数に格納します。このローカル変数に再度別の値を割り当てる場合は、前もってメモリ ブロックを解放するか、またはこのポインタを別の変数に割り当てておかないと、アプリケーション内にリークが発生します。

- **ActiveCheck** を使用する場合 : DevPartner は、malloc または new によって割り当てられたブロックがリークされたことをレポートし、メモリが割り当てられた行を示します。エラーは、アプリケーションの停止時にレポートされます。
- **FinalCheck** を使用する場合 : DevPartner は、ブロックが割り当てられている位置をレポートし、そのブロックを参照している最後の変数に新しい値を割り当てた行を強調表示します。エラーは、発生した時点でレポートされます。

[検出されたプログラム エラー] ダイアログ ボックスを使用する

DevPartner では、アプリケーションに重大なエラーが検出されると、[検出されたプログラム エラー] ダイアログ ボックス（図 2-1 を参照、26 ページ）が表示されます。

[検出されたプログラム エラー] ダイアログ ボックスの上部に、検出されたエラーの説明が表示されます。エラーの説明の下に 1 つまたは複数のタブがあり、各タブはアプリケーション内の場所と対応するコール スタックに関連付けられています。レポートされたエラーとソース情報を確認して、問題の原因特定とその修正に役立てます。

実行できるユーザー操作を理解する

[検出されたプログラム エラー]ダイアログ ボックスには、[説明]、[メモリおよびリソース ビューア]、[デバッグ]、[コピー]、[抑制]の各ボタンが表示されます。

説明

[説明]をクリックすると、各エラーの詳細な説明、サンプル コード、問題を修正するための解決案のリストが表示されます。

メモリおよびリソース ビューア

[メモリ/リソース ビューア]をクリックすると、解放されていないメモリとリソースの詳細な説明が表示されます。詳細については、「[メモリおよびリソース ビューア]ダイアログ ボックスを理解する」（34ページ）および『エラー検出ガイド』を参照してください。

コピー

[コピー]をクリックすると、すべてのウィンドウとタブ（ソース ペインを除く）の内容がクリップボードに転送されます。ユーザーは、この情報を他のアプリケーションに貼り付けできます。

抑制

[抑制]をクリックすると、現在のエラーを抑制できるダイアログ ボックスが開きます。抑制の使用方法と使用理由の詳細については、「[抑制]と[フィルタ]ダイアログ ボックスを理解する」（36ページ）および『エラー検出ガイド』を参照してください。

デバッグ

Visual Studioで作業しているとき、ダイアログ ボックスの下部に[デバッグ]が表示されます。ただし、スタンドアロン アプリケーションでは使用できません。[デバッグ]をクリックすると、Visual Studioデバッガでコードが開きます。

中止

[中止]をクリックすると、アプリケーションが停止します。これにより、効率的にプロセスが終了され、場合によってはアプリケーションを停止するよりも優れた手段となります。

続行

[続行]をクリックすると、エラーの確認後、ダイアログ ボックスが閉じてアプリケーションが引き続き実行されます。エラーは、あとから検証結果ペインで確認できるように、セッション ファイルに保存されます。

[メモリおよびリソース ビューア]ダイアログ ボックスを理解する

メモリおよびリソース ビューアにアクセスするには、[検出されたプログラム エラー]ダイアログ ボックスの[メモリおよびリソース ビューア]ボタンをクリックします。メモリおよびリソース ビューアでは、解放されていないメモリとリソースの割り当てを分析できます。

たとえば、ほとんどのメモリ分析ツールでは、アプリケーションの実行時にメモリへの影響を判定できません。リーク状態のメモリまたはリソースは、アプリケーションの停止後にしかレポートされません。のメモリおよびリソース ビューアは、プログラム実行の任意のポイント

で取得したメモリおよびリソースの「スナップショット」を提示します。また、現在割り当てられているメモリ ブロックまたはリソースを「マーク」して、プログラムの初期化後やトランザクションの処理中に割り当てられたブロックの表示を制限することもできます。

これらの機能は、以下のような状況で特に役に立つ場合があります。

- 定期使用中に24/7サーバー アプリケーションを終了できない
- リソースの消費に応答してアプリケーションが停止する可能性がある
- プログラム終了時に自動的にクリーン アップされる大量のメモリをアプリケーションが消費する可能性がある

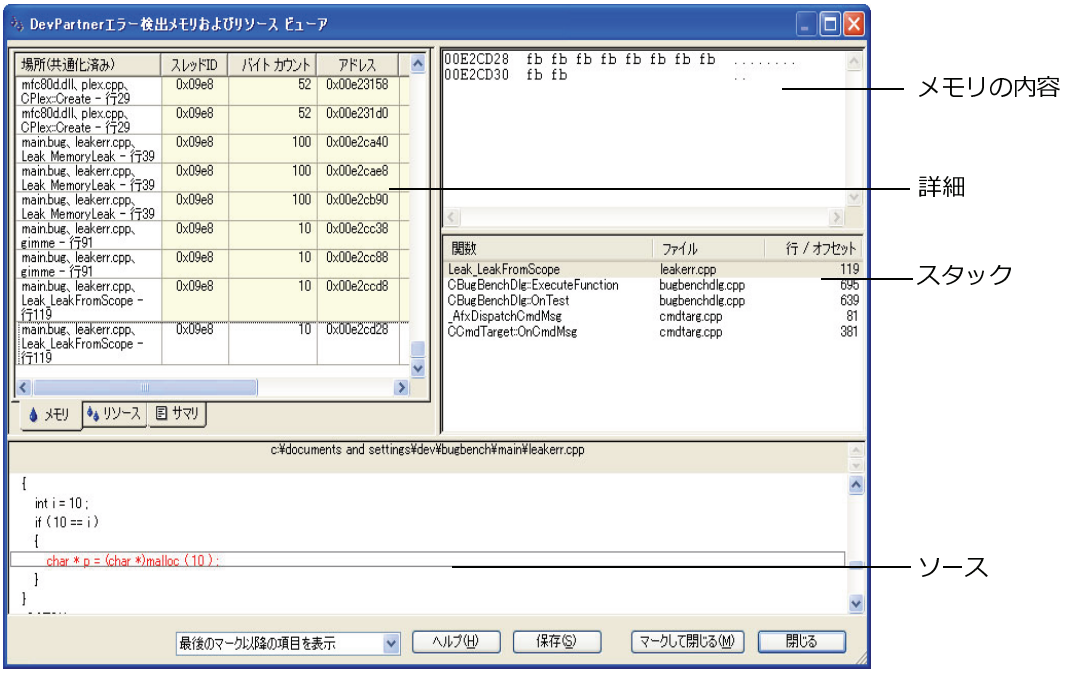


図 2-5 [メモリおよびリソース ビューア]ダイアログ ボックス

メモリおよびリソース ビューアのユーザー インターフェイスを確認する

[メモリおよびリソース ビューア]ダイアログ ボックスにアクセスするには、[検出されたプログラム エラー]ダイアログ ボックスの[メモリ/リソース ビューア]をクリックします。

[メモリおよびリソース ビューア]ダイアログ ボックスは、4つのペインで構成されています。

- メモリの内容ペイン

  さまざまな形式でメモリ ブロックの内容を表示します。リソースに対しては使用できません。

- 詳細ペイン

  [メモリ **]**、**[** リソース**]**、**[** サマリ **]** の各タブがあります。各メモリとリソースの割り当ての詳細が表示されます。

- スタック ペイン

  [メモリ **]** タブにエントリのメモリ ダンプとコールスタックの情報を表示し、**[** リソース**]** タブにエントリの説明とコールスタック情報を表示します。

- ソース ペイン

  コールスタック エントリに対応するソース コード　使用可能な場合）が表示されます。

メモリおよびリソース ビューアの内容を保存する

[保存 **]** をクリックして、あとで確認できるように [メモリおよびリソース ビューア] ダイアログ ボックスの現在の内容をテキスト ファイルとして記録します。

基準点を設定する

[マークして閉じる **]** をクリックすると、メモリとリソース データを記録するための基準点が設定されます。これによって、基準点をマークしたイベントの前後のメモリとリソースの割り当てを比較できるようになります。

[抑制] と [フィルタ] ダイアログ ボックスを理解する

DevPartnerで提供されている [抑制] ダイアログ ボックスと [フィルタ] ダイアログ ボックスを使用すると、収集または表示されるデータを減らすことができます。どちらの方法も、データを制限して管理しやすい分析用のサブセットにすることを目的としています。

たとえば、Kernel32 の `FindResourceA` からのコール バリデーション エラー、または Kernel32 のすべてのコールに対するコール バリデーション エラーを抑制できます。この選択を行ったあとは、アプリケーション内のさまざまな別の選択条件と組み合わせることができます。DevPartner のデフォルトでは、最も制限が低いオプションに設定されています（図 2-6 を参照）。

抑制またはフィルタの適用時には、以下の操作も実行できます。

- 指定した抑制またはフィルタを作成した理由を説明するコメントを入力する。
- 現在の実行または将来の実行に抑制またはフィルタを適用することを選択する。
- 再利用や共有を目的に抑制またはフィルタのインストラクションを保管するための手段として、抑制ファイルまたはフィルタ ファイルを作成する。

エラーを抑制する

エラーを抑制することによって、以降にそれらのエラーが出現してもスキップするように DevPartner を設定します。抑制されたエラーはログに記録されず、[検出されたプログラム エラー] ダイアログ ボックスに表示されません。エラーを抑制するには：

- [検出されたプログラム エラー] ダイアログ ボックスにエラーが表示されたときに、**[** 抑制 **]** をクリックします。

- エラー検出のメイン ウィンドウのいずれかのペインで特定のエラーを右クリックし、**[** 抑制 **]** を選択します。

抑制ファイルを作成して保存する

複数の抑制ファイルを作成し、その際に大規模なアプリケーションを構成する DLL に抑制ライブラリを追加作成できます。開発チームのメンバーの間で、抑制を簡単に再利用したり共有したりできます。

最初にエラー検出で .EXE を開くときに、チェック対象の .EXE と同じフォルダにデフォルトの抑制ファイルが作成されます。

以下のセクションでは、エラー検出で抑制ファイルを作成する方法を説明します。

[抑制ファイル]ダイアログ ボックスから作成する方法

[抑制ファイル]ダイアログ ボックスから抑制ファイルを作成するには、以下の手順に従います。

1. [抑制ファイル]ダイアログ ボックスにアクセスします。
   - ◇ **Visual Studio** の場合：**[DevPartner] > [** エラー検出ルール **] > [** 抑制 **]** を選択します。
   - ◇ スタンドアロンの場合：**[** プログラム **] > [** ルール **] > [** 抑制 **]** を選択します。
2. **[** 追加 **]** をクリックします。
3. 抑制ファイルに割り当てる名前を **[** ファイル名 **]** テキスト ボックスに入力して、**[** 開く **]** をクリックします。
4. **[** はい **]** をクリックして確定します。

   作成した抑制ファイルが [抑制ファイル]ダイアログ ボックス上部のペインの **[** 使用できる抑制ファイル **]** リストに追加されます。
5. **[OK]** をクリックします。

   抑制ファイルは作成済みですが、ユーザーが抑制を追加するまでは空の状態です （「抑制ファイルにエントリを追加する」 38 ページ）を参照）。追加した抑制は、現在のエラー検出セッションを閉じるまで保存されません。

[抑制]ダイアログ ボックスから作成する方法

[抑制]ダイアログ ボックスから抑制ファイルを作成するには、以下の手順に従います。

1. セッションを完了したら、**[** メモリ リーク **]**、**[** その他のリーク **]**、**[** エラー **]**、**[.NET** パフォーマンス **]**、または **[** モジュール **]** タブで特定のエラーを右クリックし、**[** 抑制 **]** を選択します。
2. [抑制]ダイアログ ボックスで、**[** 場所 **]** フィールドの右側の参照ボタン （...）をクリックします。[抑制ファイルを追加]ダイアログ ボックスが開きます。
3. 抑制ファイルに割り当てる名前を **[** ファイル名 **]** テキスト ボックスに入力して、**[** 開く **]** をクリックします。
4. **[** はい **]** をクリックして確定します。
5. **[OK]** をクリックします。

   抑制ファイルには、手順 1 で右クリックしたエラーを抑制するインストラクションが含まれています。追加したすべてのインストラクションおよび抑制は、現在のエラー検出セッションを閉じるまで保存されません。

抑制ファイルにエントリを追加する

抑制ファイルにエントリを追加するには、以下の手順に従います。

1 検証結果ペインで、[メモリ リーク]、[その他のリーク]、[エラー]、[モジュール]、または[通知情報]のいずれかのタブを選択します。

2 そのタブ内で特定のエラー、リーク、またはモジュールを右クリックし、[抑制] を選択します。[抑制]ダイアログ ボックスが開きます（図 2-6、38ページ）を参照）。

3 追加する抑制のタイプを選択します。

上部のペインに、各種の抑制オプションが表示されます。選択できるオプションは、選択したイベント（エラー、リーク、モジュール）と、エラー検出がイベントに遭遇したときのコンテキストに応じて異なります。

4 必要に応じて、抑制エントリを説明するコメントを入力します。

コメントは抑制ファイルをアップデートする際に役立ちます。特に、抑制がアドレス ベースで、サードパーティ ベンダが新規または更新ライブラリを出荷する場合に便利です。

5 この抑制を保存するには、[抑制情報の保存]チェック ボックスをオンにします。

6 このエントリを追加する抑制ファイルの場所を指定するには、[場所]ドロップダウン メニューからファイルを選択します。

- ◇ 選択しなければ、エントリはデフォルトのプログラム抑制ファイルに追加されます。
- ◇ プログラムに抑制ファイルを追加していない場合、選択肢はデフォルトのプログラム抑制ファイルのみです。
- ◇ 別の場所にある抑制ファイルを指定するには、参照ボタン（...）（[場所]ドロップ ダウン メニューの右側）をクリックして、別の抑制ファイルを選択します。

7 **[OK]** をクリックして続行します。追加したエントリは、現在のエラー検出セッションを終了するまでは保存されません。

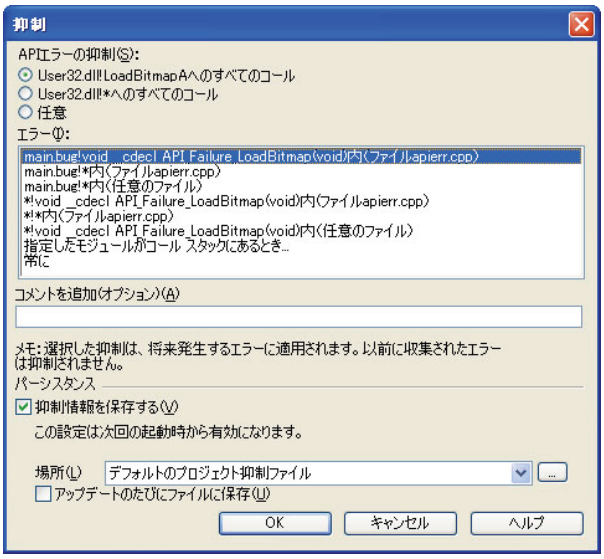


図 2-6 [抑制]ダイアログ ボックス

エラーをフィルタ処理する

フィルタを使用すると、すでに **.dpbc1** ログ ファイルに記録されたイベントを非表示にできます。DevPartner ではこれらのエラーが検出されますが、検証結果ペインでは非表示になるか、**[** フォントと色 **]** で指定した形式で表示されます。フィルタ対象のエラーを選択するには：

- エラー検出のメイン ウィンドウの、いずれかのペインで特定のエラーを右クリックし、**[** フィルタ **]** を選択します。
- エラー検出のメイン ウィンドウの、いずれかのペインで特定のエラーを選択し、ツールバーの **[** フィルタ **]** ボタンをクリックします。

フィルタ処理のインストラクションを削除すると、以降は関連のエラーはフィルタされず、検証結果ペインに表示されます。

フィルタ ファイルを作成する

フィルタ ファイルを作成するには、次の 2 通りの方法があります。

[フィルタ ファイル]ダイアログ ボックスから作成する方法

[フィルタ ファイル] ダイアログ ボックスからフィルタ ファイルを作成するには、以下の手順に従います。

1 [フィルタ ファイル] ダイアログ ボックスを開きます。

2 **[** 追加 **]** をクリックします。

[フィルタ ファイルを追加] ダイアログ ボックスが開きます。

3 フィルタ ファイルに割り当てる名前を **[** ファイル名 **]** テキスト ボックスに入力して、**[** 開く **]** をクリックします。

4 **[** はい **]** をクリックして確定します。

フィルタ ファイルが [フィルタ ファイル] ダイアログ ボックス上部のペインの **[** 使用できるフィルタ ファイル **]** リストに追加されます。フィルタ ファイルは空で、プログラムの検証結果の表示には反映されません。

追加したフィルタは、現在のエラー検出セッションを終了するまでは保存されません。

[フィルタ]ダイアログ ボックスから作成する方法

[フィルタ] ダイアログ ボックスからフィルタ ファイルを作成するには、以下の手順に従います。

1 セッションを完了したら、**[** メモリ リーク **]**、**[** その他のリーク **]**、**[** エラー **]**、**[** モジュール **]**、または **[** 通知情報 **]** タブで特定のイベントまたはエラーを右クリックし、**[** フィルタ **]** を選択します。[フィルタ] ダイアログ ボックスが開きます（図 2-6 38 ページ）を参照）。

2 **[** 場所 **]** フィールドの右側の参照ボタン（...）をクリックします。

[フィルタ ファイルを追加] ダイアログ ボックスが開きます。

3 フィルタ ファイルに割り当てる名前を **[** ファイル名 **]** テキスト ボックスに入力して、**[** 開く **]** をクリックします。

4 **[** はい **]** をクリックして確定します。フィルタ ファイルには、手順 1 で選択したエラーまたはイベントを非表示にするインストラクションが含まれています。追加したフィルタは、現在のエラー検出セッションが閉じるまでは保存されません。

フィルタ ファイルにエントリを追加する

既存のフィルタ ファイルにエントリを追加するには、以下の手順に従います。

1. 検証結果ペインで、**[**メモリ リーク**]**、**[**その他のリーク**]**、**[**エラー**]**、**[**モジュール**]**、または **[** 通知情報 **]** のいずれかのタブを選択します。
2. そのタブ内で特定のエラー、リーク、またはモジュールを右クリックし、**[** フィルタ **]** を選択します。[フィルタ] ダイアログ ボックスが表示されます（図 2-6、38 ページ）を参照）。
3. [オプション] を選択します。

   これらのオプションは、ダイアログ ボックスの上部のペインにリストされます。選択できるオプションは、選択したイベント（エラー、リーク、モジュール）と、エラー検出がイベントに遭遇したときのコンテキストに応じて異なります。
4. 必要に応じて、エントリを説明するコメントを入力します。コメントはフィルタ ファイルをアップデートする際に役立ちます。特に、フィルタがアドレス ベースで、サードパーティ ベンダが新規または更新ライブラリを出荷する場合に便利です。
5. このエントリを保存して再利用する場合は、**[** フィルタ情報を保存 **]** チェック ボックスをオンにします。
6. このエントリを保存するフィルタ ファイルの場所を指定するには、**[** 場所 **]** ドロップダウン メニューからファイルを選択します。
   - ◇ 選択しなければ、エントリはデフォルトのプログラム フィルタ ファイルに追加されます。
   - ◇ プログラムにフィルタ ファイルを追加していない場合、選択肢はデフォルトのプログラム フィルタ ファイルのみです。
   - ◇ 別の場所にあるフィルタ ファイルを指定するには、参照ボタン（...）（**[**場所 **]** ドロップ ダウン メニューの右側）をクリックして、別のフィルタ ファイルを選択します。
7. **[OK]** をクリックして続行します。追加したエントリは、現在のエラー検出セッションを終了するまでは保存されません。

フィルタ処理されたエラーの表示と非表示を切り替える

検証結果ペインでフィルタ処理されたエラーの表示と非表示を切り替えるには、**[** フィルタ処理されたエラーの表示 **]** ツールバー アイコンを使用します。

フィルタ エントリを削除する

不要になったフィルタ エントリを削除するには、**[DevPartner]** > **[** エラー検出ルール **]** > **[** フィルタ **]** を選択します。エントリを含むフィルタ ファイルを選択して、関連のチェック ボックスをオフにします。

コール バリデーションを理解する

コール バリデーションを有効にすると、DevPartner では、5,000 を超える Windows API コールが検証されます。DevPartner のチェック対象は、以下を含む多数のイベントです（以下の項目だけではありません）。

- ◆ ハンドル エラーとポインタ エラー

- フラグ
- 範囲チェック
- API とメソッドのエラー
- 不正な構造体サイズ
- メモリ アクセス エラー

フラグ チェックまたは範囲チェックで、解決しようとしている問題には当てはまらない不要なエラーが生成されていると判断した場合は、**[** フラグ、範囲、および列挙の引数**]** チェック ボックスをオフにします。コール バリデーションでは戻り値のチェックと、さらに重要なハンドルのチェック、および Windows コールに対して受け渡しされるポインタのチェックが続行されます。

メモリ ブロック チェックを有効にする

メモリ ブロック チェックを有効にすると、コール バリデーションでは C ランタイム ライブラリに対するすべてのコールと多数のその他のコールに対して詳細な分析が実行されます。メモリ ブロックのチェックでは全体のパフォーマンスが低下しますが、検出が困難なエラーを診断する場合に役立つことがあります。デフォルトで、この設定は無効化されています。

[設定 **]** ダイアログ ボックスを使用する

ヒント : [設定]ダイアログ ボックスの構成ファイルの管理機能を使用して、構成ファイルとしてエラーチェック パラメータの設定を保存してください。複数のプロジェクトで作業している場合は、それぞれのプロジェクトの構成ファイルをロードし、編集して、関連付けることができます。

DevPartner の設定を使用して、以下の操作を実行できます。

- 特定の問題に必要となる種類のデータ収集だけを選択する
- 主なデータ収集の種類の一部を有効化または無効化する
- プログラムの分析対象となる部分を制御する
- デフォルトの DevPartner 設定を使用して、パフォーマンスに最低限の影響しか与えない最も一般的なエラーを検出する

[設定]ダイアログ ボックスには、以下の方法でアクセスできます。

- スタンドアロンの場合 : **[** プログラム **]** > **[** 設定 **]** を選択する
- **Visual Studio** の場合 : **[**ツール**]** > **[**オプション**]** を選択し、ツリー ビューから **[DevPartner]** > **[** エラー検出 **]** を選択します。

[設定]ダイアログ ボックスには、主な設定のカテゴリを示すツリー ビューがあります。カテゴリを選択すると、ダイアログ ボックスにカテゴリの詳細設定が表示されます。

DevPartner スタンドアロン アプリケーションと統合型の Visual Studio バージョンでは、同じツリー ビューと [設定]ダイアログ ボックスを使用します。

設定のグループはすべて、同じ基本構造に従います。ダイアログ ボックスの最上部のチェック ボックスをオンにして、主なデータ収集の種類を有効化または無効化できます。

最上部の各チェック ボックスの下には、DevPartner によるアプリケーションの分析方法をさらに定義するその他の設定があります。設定を変更して、エラー検出プロセスをカスタマイズします。

たとえば、エラー検出の範囲を広い範囲と狭い範囲で切り替えることができます。

- 広い範囲 – 多数のデータ型と、選択した多数の関連設定を検証します。
  - ◇ 検出されるエラー数が多い
  - ◇ 疑陽性により近い潜在的なエラーを含む
  - ◇ パフォーマンスが低下する（検出するエラー数が多くなるため）
  - ◇ 作成されるログ ファイルが大きくなる
- 制限された範囲 – 少数のデータ型と、選択した少数の関連設定を検証します。
  - ◇ 特定の関数という狭い範囲を対象とする
  - ◇ 検出されるエラー数が少ない
  - ◇ 関連するエラーが検出されない可能性がある
  - ◇ 問題に関連するエラーのみを簡単に確認できる機会が高まる
  - ◇ パフォーマンスが向上する
  - ◇ 作成されるログ ファイルが小さくなる

[全般]プロパティを設定する

[プログラム設定]ダイアログ ボックスにアクセスすると、最初に[全般]プロパティが表示されます。

- イベントをログに記録：オンにすると、イベントのログ記録が有効になります（イベントのログ機能は、DevPartner エラー検出の他の部分からも有効にできます）。
- エラーを表示して一時停止：プログラムの実行を解析して、特定のエラーのポップ アップを表示するように、[検出されたプログラム エラー]ダイアログ ボックスの表示を制御します。
- プログラム検証結果の保存を確認する：オンにすると、プログラムを停止するか、エラー検出セッションを閉じる前に、プログラムの検証結果を保存するようにメッセージが表示されます。
- アプリケーションの終了時にメモリとリソース ビューアを表示する：選択すると、DevPartner エラー検出で、テストしているアプリケーションの終了時に[メモリおよびリソース ビューア]ダイアログ ボックスが開きます。
- ソース ファイルの検索パス：この設定に含めるソース ファイルのフル パスを指定します。

以下の設定は、エラー検出のスタンドアロン アプリケーションでのみ使用できます。

- シンボル パスの上書き：この設定に含めるシンボル ファイルのフル パスを指定します。このフィールドの右側にある参照ボタン（...）をクリックして、[シンボル パス]ダイアログ ボックスを開きます。
- 作業ディレクトリ：ターゲット プロセスの作業フォルダを指定します。

メモ： アプリケーションが起動しない場合は、作業フォルダが読み取り専用の可能性があります。アプリケーションによっては、作業フォルダに書き込み権限が必要です。

- コマンド ライン引数：コマンド ラインでアプリケーションに渡される引数を指定します。

メモ： DevPartner エラー検出でアプリケーションが正しく開始されない場合は、コマンド ライン引数を確認してください。COM サーバー アプリケーションの場合は、これらの引数が特に重要です。

データ収集のプロパティを設定する

データ収集プログラムの設定は、エラー検出の以下のパラメータを制御します。

- コール パラメータのデータ表示の深さ：コール パラメータに関して収集される詳細度を指定します。小さい値を指定すると処理が速くなりますが、ポインタで参照される詳細は深いレベルまではレポートされません。大きい値を指定すると、コールの詳細は深いレベルまでレポートされますが、処理が遅くなり、ログ ファイルのサイズが増えます。
- メモリ割り当ての最大コール スタック数：割り当て時に追跡されるコール スタックの数を指定します。割り当ては頻繁に行われ、エラーになることはほとんどないため、大きい値を選択するとパフォーマンスが低下します。また、大きい値を選択すると、テスト中のアプリケーションの、エラー検出のメモリ使用量が非常に大きくなることがあります。
- エラーの最大コール スタック数：エラーのレポートに使用されるコール スタックの数を指定します。この値を必要なだけ大きく設定しても、十分なログ ファイル領域を確保すれば、パフォーマンスが低下することはありません。
- **NLB** ファイル ディレクトリ：（このフィールドは必須です）生成された NLB（最適化されたタイプ ライブラリ）のファイルが保存されている場所を選択します。通常はプロジェクトと同じ場所に保存されるので、プロジェクトと NLB ファイルの削除が容易になります。存在しない場所を指定すると、アプリケーションの実行時に、有効な場所の選択を求めるメッセージがエラー検出で表示されます。**[** 参照 **]** ボタン（**...**）を使用してシステムを参照し、生成された NLB ファイルを保存するフォルダを指定することもできます。NLB ファイルには、エラー検出に必要な API 記述ファイルの情報がすべて含まれています。

API コール レポーティング プロパティを設定する

ヒント：チェックする必要のない API コールをプログラムが実行している場合は、API 関数の選択を解除します。データ収集を制限すると、パフォーマンスが向上します。

API コール レポーティングを使用して、アプリケーションからシステム関数に対するコールと、パラメータおよび戻り値を記録します。DevPartner エラー検出では、[データ収集] 設定で指定した **[** データ表示の深さ **]** に基づいて、戻り値とパラメータの構造情報を記録します。

API コール レポーティングを有効にして、API のチェック ボックスとモジュールをアクティブにするには、**[API** コール レポーティングを有効にする **]** チェック ボックスをオンにします。以下の設定は、エラー検出の API ログ機能を制御します。

- ウィンドウ メッセージを収集する：オンにすると、API ログの一部として、ウィンドウ制御メッセージが収集されます。
- **API** コールとリターンを収集する：オンにすると、API モジュール ツリーで選択したモジュールの API メソッド コールと戻り値が収集されます。
- このアプリケーションに必要なモジュールだけを表示する：このチェック ボックスをオンにすると、ツリー ビューのプログラムに必要な API モジュールだけが表示されます。チェック ボックスをオフにすると、ツリー ビューが展開され、すべての使用可能な API モジュールが表示されます。
- **API** モジュール ツリー ビュー：プロジェクトに関連付けられた API モジュールが表示されます。アイテムの隣のプラス（+）記号をクリックすると、含まれている関数が表示されます。各アイテムの隣のチェック ボックスをオンにすると、特定のモジュールや関数を API ログの対象として選択できます。

ヒント：**[**ウィンドウ メッセージを収集**]**をオンにすると、ログ ファイルのサイズが増加します。最良の結果を得るには、ウィンドウ メッセージに関する問題をデバッグする場合にだけ、この機能を選択してください。

コール レポーティングを有効にすると、ログ ファイルのサイズが大幅に増加することがあります。ログ ファイルのサイズを最小にするには、アプリケーションの選択した部分のコール レポーティング データのみを収集することを検討してください。チェックする部分を制限するには、以下のような方法があります。

- モジュール ツリーのチェックボックスを使用して、チェックする必要のないAPIモジュールの選択を解除する。
- **[** モジュールとファイル **]** を使用して、ログのスコープを制限する。
- イベントログを有効または無効にするAPIコールをアプリケーションに追加する。`NmApiLib.h` 内のコメントを参照してください。このファイル（DevPartner のインストールの一部）には、DevPartner エラー検出によってエクスポートされたイベント レポーティング API が定義されています。
- イベント ログを無効にする。

コール バリデーション オプションを設定する

コール バリデーションを有効にすると、アプリケーションからオペレーティング システム ライブラリへのコールとCOM メソッド コールが監視されます。渡されたパラメータの有効性が検証され、成功を示す値が返されたかどうかがチェックされます。以下の要素は、エラー検出のコール バリデーションの側面を制御します。

- コール バリデーションを有効にする：このチェック ボックスをオンにすると、コール バリデーションのコンポーネントが有効になります。
- メモリ ブロック チェックを有効にする：このチェック ボックスをオンにすると、メモリを参照するパラメータについて広範囲にわたるメモリ チェックが有効になります。この機能は、**[** プログラムの設定 **]** ツリー ビューの **[** メモリの追跡 **]** で **[** メモリの追跡を有効にする **]** を選択するまでは、有効になりません。

  [メモリ ブロック チェックを有効にする**]**を選択すると、DevPartnerエラー検出によって広範なチェックが実行されます。結果はより正確になり、多数のバグを見つけることができます。このオプションを有効にしたセッションは完了まで時間がかかります。
- コール前に引数で指定されたバッファを既定値で埋める：このチェック ボックスをオンにすると、**[** 確保時にフィルする **]** の [メモリの追跡] 設定で指定したパターンの出力引数が入力されます。
- **COM** 失敗コード：このチェック ボックスをオンにすると、COM メソッドの戻り値のチェックが有効になります。

  通常の使用では、多数のCOMメソッドで「実装されていません」というエラーが報告されます。このチェックを無効にすると、レポートされるエラー数が大幅に減少します。
- **COM** のリターン コード「実装されていません」をチェックする：このチェック ボックスをオンにすると、`HRESULT E_NOTIMPL`（「実装されていません」）リターン コードのチェックが有効になります。DevPartner エラー検出では、このダイアログ ボックスの **[API** エラーをチェックする **DLL**（失敗または不正な引数）**]** で選択された DLL に含まれる COM インターフェイスのみがチェックされます。

- **API** エラー コード: このチェック ボックスをオンにすると、選択した DLL にある API からの戻り値のチェックが有効になります。
- 不正な引数エラー（**COM** または **API**）をチェックする : これらのチェック ボックスのどちらかまたは両方をオンにすると、選択した DLL の API またはエラー検出がサポートする COM インターフェイス、あるいはその両方に対する引数（パラメータ）のチェックが有効になります。

ヒント : パフォーマンスを向上させ、レポートされるエラー数を減らすには、必要に応じてこのオプションを選択します。誤ったコール バリデーション エラーの数を減らすには、**[**ハンドルとポインタの引数**]** をオンにし、**[** フラグ、範囲、および列挙の引数**]** をオフにします。

- カテゴリ : **[**ハンドルとポインタの引数 **]** または **[** フラグ、範囲、および列挙の引数 **]**）**[** 不正な引数エラーのチェック **]** オプションで [COM] と [API] のいずれかまたは両方を選択すると使用できます。これらのチェック ボックスのいずれかまたは両方をオンにすると、引数のタイプに基づいた引数チェックが有効になります。
- 静的にリンクされた**C**ランタイム ライブラリ**API**をチェック: **[API**エラー コード**]**または **[** 不正な引数エラー **]** の [API] を選択した場合に使用可能です。このチェック ボックスをオンにすると、静的な C ランタイム コールのチェックが有効になります。静的な C ランタイム ライブラリを使用しない場合は、これをオフにして、サードパーティ製ライブラリのエラーが表示されないようにします。

ヒント : このリストの DLL を無効にすると、不要なエラーの数が減ります。パフォーマンスを向上させることもできます。

- **API** エラーをチェックする **DLL(**失敗または不正な引数**)**: **[API** エラー コード **]** または **[** 不正な引数エラー **]** の [API] を選択した場合に使用可能です。このチェック ボックスをオンにすると、リストに表示された DLL の API 引数と戻り値のチェックが有効になります。

  Visual Studio に付属の Depends などのツールを使用すると、アプリケーションが使用している DLL から DLL や API を検出できます。

API コールのメモリ上書きの検出を有効にする

API コール（`$trcpy` など）によって起こるメモリ ブロックの破損のチェックは、デフォルトでは無効になっています。API コールのメモリ上書きの検出を有効にするには、以下の手順に従います。

1. **[** メモリの追跡を有効にする **]** チェック ボックスをオンにします。
2. **[** プログラムの設定 **]** ツリー ビューで、**[** コール バリデーション **]** を選択します。
3. **[** コール バリデーションを有効にする **]** チェック ボックスをオンにします。
4. **[** メモリ ブロック チェックを有効にする **]** チェック ボックスをオンにします。

COM コール レポーティング プロパティを設定する

COM コール レポーティングを使用して、COM インターフェイスへのコールと、**[** すべてのインターフェイス **]** ツリーで選択したインターフェイスへの戻り値を記録します。DevPartner エラー検出では、パラメータ値と戻された HRESULT を記録します。

チェックが必要なインターフェイスのみを選択してください。チェックするインターフェイスの数を減らすと、ログ ファイルのサイズが減少し、パフォーマンスが向上します。

COM コール レポーティングを有効にするには、COM インターフェイスのリストを有効にして、**[** 選択したモジュールに実装された **COM** メソッド コールのレポートを有効にする **]** チェック ボックスをオンにします。以下のコントロールを使用して、COM コール レポーティングを設定します。

- リストされていないモジュールに実装された **COM** メソッド コールをレポートする : このチェック ボックスをオンにすると、**[** すべてのインターフェイス **]** ツリーにないインターフェイスに対する COM メソッド コールと戻り値をレポートするように、エラー検出が設定されます。
- すべてのコンポーネント ツリー ビュー : プロジェクトに関連付けられた COM インターフェイスが表示されます。**[** すべてのコンポーネント **]** エントリの隣にあるプラス (+) 記号をクリックすると、COM インターフェイスの全リストが表示されます。各アイテムの隣のチェック ボックスをオンにすると、COM コール レポーティングに特定のインターフェイスを選択できます。

COM オブジェクトの追跡オプションを設定する

[COM オブジェクトの追跡] を使用して、プログラムで COM オブジェクトのリークを監視します。オブジェクトのリークは、検証結果ペインの **[** その他のリーク **]** タブに表示されます。**[** その他のリーク **]** タブでオブジェクト リーク エラーを選択すると、オブジェクトで `AddRef()` および `Release()` へのコールがチェックされ、`Release()` へのコールの不在を検出できます。

パフォーマンスを改善するには、[すべての COM クラス] のサブセットを選択します。アプリケーションの最初のパスを実行する場合か最後の QA のパスを作成する場合にだけ、すべての COM クラスを選択することをお勧めします。

COM オブジェクトの追跡を有効にするには、**[** すべての **COM** クラス **]** ツリー ビューを有効にして、**[COM** オブジェクトの追跡を有効にする **]** チェック ボックスをオンにします。

[すべての **COM** クラス **]** ツリー ビューを使用して、監視対象の COM クラスを選択します。アプリケーションの COM クラスが見つからない場合は、**[** レジストリから更新 **]** をクリックしてリストを更新します。

ヒント : ほとんどのベンダーがオブジェクト名に共通の接頭辞を使用しています。

デッドロック分析オプションを設定する

マルチスレッドのアプリケーションでデッドロックを監視するには、デッドロック分析を使用します。これには次のような分析が含まれます。

- アプリケーションでデッドロックの発生を監視してレポートします。
- アプリケーション内で同期オブジェクトの使用パターンを監視して潜在的なデッドロックを検出します。

デッドロック分析を有効にするには、他のデッドロック分析の制御を有効にして **[** デッドロック分析を有効にする **]** チェック ボックスをオンにします。

以下の設定は、デッドロック分析の動作を制御します。

- シングル プロセスとみなして分析する : オンにすると、アプリケーション内で使用されている名前付きの同期オブジェクトすべてがそのプロセス内でのみ使用されていると エラー検出で想定されます。このチェック ボックスをオフにすると、名前付きの同期オブジェクトに関連のあるデッドロック検出ルールの一部が緩和されます。

- ウォッチャー スレッドを有効にする: このチェック ボックスをオンにすると、アプリケーションで「ウォッチャー」を作成して、局部的なデッドロックが監視されます。エラー検出がアプリケーションを妨害しないように、デフォルトではこの機能が無効になっています。

  アプリケーションが応答しなくなり、デッドロック状態に見える場合は、エラー検出でこの機能を有効にして、アプリケーションの細かい分析を実行してください。

  プロセスで余分のスレッドに遭遇することのない複雑な DLL_THREAD_ATTACH ロジックを作成する場合は、このオプションを有効にしないでください。

- エラーを生成するとき : 以下から、エラー検出でデッドロック エラーをレポートする状況を選択します。

  - クリティカル セクションに再入したとき: スレッドがすでに所有しているクリティカル セクションに再び入ろうとした場合に、警告が生成されます。クリティカル セクションの再入はエラーではありませんが、アプリケーションがクリティカル セクションに入る回数と、そこから出る回数は同じでなければなりません。

  - 所有する **Mutex** に待機が要求されたとき: スレッドがすでに所有している Mutex を待機しようとした場合に、警告が生成されます。

  - リソースごとの過去のイベント数 : エラーや警告でレポートされた同期オブジェクトのそれぞれについて、記録するコール スタックの数を入力します。

    各同期オブジェクトにスタック情報を関連付けておけば、同期オブジェクトが特定の状態にある理由を判断できます。これはデッドロック状況のデバッグに役立ちます。

メモ : 同期オブジェクトのそれぞれに維持するコール スタックの数を増やすと、アプリケーションで消費されるメモリが増えて、パフォーマンスに影響します。

  - 同期 **API** タイムアウトをレポート: 同期オブジェクトの待機が完了せずにタイムアウトになったときに、エラーを報告します。

    Windows の呼び出しすべての API コール レポーティングを有効にしなくても同期オブジェクトの API エラーを監視できるようにするには、このオプションを有効にしてください。

    [待機制限または実際の超過時間 **(**秒 **)** をレポート **]** オプションを使用して、アプリケーション内で最大待機ポリシーを強制することができます。

  - 待機制限または実際の超過時間 **(**秒 **)** をレポート: [レポート タイムアウト] を選択すると、以降はアクティブになります。エラー検出は、同期オブジェクトの待機コールに渡されるタイムアウト値をチェックします。ここで指定した限界をタイムアウト値が超えた場合は、そのコールはエラーとして報告されます。

メモ : `INFINITE` として指定された待機は、エラーとしてのフラグは付けられません。

- 同期オブジェクトのネーミング ルール : 以下のオブジェクト標準から選択します。

  - リソース ネーミングについて警告しない : オンにすると、アプリケーションで検出された名前付きまたは名前なしのリソースについて警告が生成されません。

ヒント : セキュリティ監査を実行している場合は、**[**命名されたリソースに関する警告を表示する **]** オプションを有効にして、予期しない名前付きリソースがプロセス外で表示されていないか確認できます。名前付きリソースはプロセス外で表示可能なため、適切なセキュリティを適用して不正使用を防ぐ必要があります。

- ◇ 名前付きリソースについて警告する：オンにすると、アプリケーション内で名前付きの同期オブジェクトが検出されるたびに警告が生成されます。このチェックを利用すると、アプリケーション外で操作可能な名前付きリソースを探すことができます。
- ◇ 名前なしのリソースについて警告する：オンにすると、アプリケーション内で名前なしの同期オブジェクトが検出されるたびに警告が生成されます。このチェックを利用すると、他のプロセスで使用したり会社の命名規則に従うために名前を付ける必要のある名前なしのリソースを見つけることができます。

エラー検出のデフォルトでは、プログラムの実行中に名前付きのリソースや名前なしのリソースが検出されても、警告は生成されません。

メモリの追跡オプションを設定する

[メモリの追跡] を有効にすると、DevPartner エラー検出では以下の処理を行います。

- メモリの割り当てと解放を行うアプリケーションのすべてのコールを監視します。
- アプリケーションの終了時に解放されていないメモリをレポートします。

また、FinalCheck インストゥルメンテーションでアプリケーションをビルドし、**[FinalCheck** を有効**]** をオンにすると、エラー検出は以下の処理を行います。

- 割り当てられたメモリ ブロックの最後の参照が範囲外になったインスタンスを記録します。
- 実行中、ステートメント レベルでメモリやポインタのエラーをレポートします。

メモリの追跡を有効にするには、メモリの追跡のオプションをすべて有効にして **[** メモリの追跡を有効にする **]** チェック ボックスをオンにします。[コール バリデーション] 設定の **[** メモリ ブロック チェック **]** を有効にするには、事前に **[** メモリの追跡を有効にする **]** チェック ボックスをオンにする必要があります。

以下の設定は、メモリ追跡の動作を制御します。

- リーク分析のみを有効：このチェック ボックスをオンにすると、メモリの追跡内のリークの監視を除くすべての機能が無効になります。メモリの追跡による、オーバーラン、未初期化メモリの使用、またはダグリング ポインタの監視が実施されなくなります。メモリの追跡ではシステム モジュールによって割り当てられたメモリの評価が行われないため、コール バリデーションのメモリ ブロック チェックも無効になります。

  このオプションが有効な場合は、COM インターフェイス フックの一部が処理されません。

- **FinalCheck** を有効にする : このチェック ボックスをオンにすると、FinalCheck が有効になります。オンにした場合、エラー検出によって、FinalCheck がインストゥルメントしたモジュールの追加チェックが実行されます。無効にすると、これらのチェックは実行されません。
- リークしたアロケータ ブロックを表示する：このチェック ボックスをオンにすると、サブアロケーションに使用されるブロックでのリークのレポートが有効になります。サブアロケーション ブロックは通常、`malloc`、`new` のようなメモリ割り当て関数によって作成されます。独自のメモリ アロケータを作成した場合は、この機能を有効にして、`malloc` や `new` などの関数から返されるブロックにサブアロケートされるバッファも含めて、アプリケーション内のすべてのメモリを監視します。DevPartner エラー検出では、**`UserAllocators.dat`** にリストされた後にだけ、ユーザーが作成したメモリ アロケータの監視を行います。**`UserAllocators.dat`** の詳細については、『エラー検出ガイド』の「ユーザーが作成したアロケータの使用」の章を確認してください。

- 厳密な再割り当てセマンティクスを強制する : このチェック ボックスをオンにすると、厳密なセマンティクスの強制が有効になります。エラー検出によって、厳密な再割り当てセマンティクスが強制されると、再割り当てされたメモリへのポインタがダングリング ポインタであるかのように扱われ、そのポインタの使用によってエラーが生成されます。厳密な再割り当てセマンティクスが有効でない場合は、再割り当てされたポインタが新しいポインタと同じメモリの場所を参照し、エラーが生成されていないかぎり、そのポインタを使用できます。例 :

  ```
  char *ptrA = (char *) malloc(17);

  // ptrA は 17 バイトのメモリを正しく参照しています。

  char *ptrB = (char *) realloc(ptrA, 15);

  // ptrB は 15 バイトのメモリを正しく参照しています。

  // 厳密なセマンティクスにより、ptrA は値にかかわらず無効なポインタとなります。

  // 厳密なセマンティクスがないため、ptrA は ptrB に等しいかぎり、引き続き有効です。
  ```

- 保護バイトを有効にする : 有効にすると、割り当てたメモリ ブロックの終わりに保護バイトが挿入され、メモリ オーバーラン エラーが検出されるようになります。オーバーランはヒープ破壊またはスタック破壊の原因となり、ランダム クラッシュや予想しないデータの上書きを引き起こします。

  - パターン : 保護バイト パターンを 16 進で入力します。このパターンを使用して、割り当てたメモリ ブロックがオーバーランかどうかを調べます。

  - カウント : 使用する保護バイトの数を選択します。ランダム ヒープ破壊エラーが発生しても、エラー検出からヒープオーバーラン エラーがレポートされない場合は、保護バイト数を増やすことを検討してください。これによってメモリの使用量が増加する場合がありますが、検出が困難なヒープ破壊を検出できます。

- 実行時のヒープ ブロックをチェックする : 保護バイトが上書きされていないかどうかを調べるためにヒープ全体をチェックする頻度を指定します。DevPartner エラー検出では、各ブロックが解放されるとブロックにオーバーランがないか常にチェックされます。追加のチェックを行うためのオプションは 3 つあります。

  - 解放時
  - 適応分析を使用
  - すべてのメモリ API コール時

- 確保時にフィルする : 有効にした場合、指定された充てんパターンが、割り当てに従ってメモリに適用されます。

  - パターン : 使用するフィル パターンを 16 進数で指定します。

- 未初期化メモリをチェックする : オンにすると、新規に割り当てられるメモリが既知のパターンで初期化され、メモリの参照時にそのパターンでチェックされます。

  - サイズ : フィル パターンをチェックする最小バイト数を選択します。不正確なエラー レポート数を減らすには、この値を大きくします。

- 解放時に無効データをフィルする : このチェック ボックスをオンにすると、メモリ解放時に無効データがフィルされます。

- ◇ パターン：無効データとしてメモリ ロケーションに書き込まれるパターンを入力します。

.NET Framework 分析オプションを設定する

.NET Framework 分析は、アンマネージ コードとマネージ コード、およびアンマネージ リソースが混在するアプリケーションを開発するときに使用します。マネージ コードとアンマネージ コードの両方を使用するアプリケーションでは、パフォーマンスの問題が発生することがあります。ここで収集するデータを使用して、パフォーマンスの問題の範囲と重要度を評価できます。発見したすべての問題を修正する時間がない場合は、この分析によってどれが重大な問題かを判断できます。

このパネルで .NET Framework 分析の制御を有効にするには、**[.NET** 分析を有効にする **]** チェック ボックスをオンにします。以下の制御を使用して、.NET Framework 分析を構成します。

- 例外の監視 : このチェック ボックスをオンにすると、アンマネージ コードまたはレガシー コードがスローした例外が処理されずにマネージ コードに戻るインスタンスが監視されます。

メモ： アンマネージ コードからマネージ コードに例外が渡されると、必要なハンドルがアンマネージ コードになくなるため、エラーが生成される可能性が高くなります。例外の記録を注意して確認してください。エラーの可能性としては、完全に初期化されていないデータ構造体、メモリ リーク、リソース リークなどがあります。

- ファイナライザの監視 : このチェック ボックスをオンにすると、適切な Dispose メソッド（リーク）コールの失敗やアンマネージ リソースをカプセル化するクラスの不正な実装など、アンマネージ リソースの使用の誤りが監視されます。

- **COM** 相互運用性の監視 : このチェック ボックスをオンにすると、マネージ コードとアンマネージ（レガシー）コード間の移行の原因となっているクラス ID が監視されます。この機能により、使用されているインターフェイス ID も識別できます。

  COM 相互運用性の監視を使用すると、頻繁に呼び出されているメソッドを判別できます。何度も呼び出されるメソッドがある場合は、オブジェクトを移植して移行を防ぐことを検討してください。書き直しできない場合は、データをまとめて転送する新しいメソッドを追加して、移行の数を減らしてください。

- **PInvoke** 相互運用性の監視：このチェック ボックスをオンにすると、アンマネージ（レガシー）コードが呼び出された回数がカウントされます（DLL と、可能な場合は API 単位）。このオプションを使用すると、アプリケーションがアンマネージ コードまたはレガシー コードを呼び出す理由を判断する手がかりになります。

  PInvoke 相互運用性の監視では、アプリケーションが PInvoke を呼び出す回数がわかります。PInvoke 相互運用性の監視レポートは、マネージからアンマネージへの移行を監視するときに使用できます。リストを確認して、コールが頻繁に発生していないか確認してください。

- 相互運用性レポートのしきい値 : 以下の説明の **x** は、このフィールドで指定した値を想定しています。アプリケーションの **`call_A`** の回数が **x** 以上の場合は、.NET 分析結果に **`call_A`** を追加します。この処理によって、一定の回数だけ発生するコールをフィルタできます。このしきい値を低くすると、結果に含まれるコールが多くなります。

  相互運用性レポートのしきい値を使用すると、COM 相互運用性および PInvoke 監視レポートから COM 移行を除外できます。DevPartner エラー検出では、移行の数が指定した値以上の場合にだけ移行がレポートされます。

.NET Framework コール レポーティング プロパティを設定する

ヒント：.NET Framework コール レポーティングは、大量のデータを生成して、システムを減速させる可能性があります。フレームワークをデバッグして理解する必要がある場合にだけ .NET Framework コール レポーティングを有効にして、チェックが必要なアセンブリだけを選択します。**[** すべてのタイプ **]** ツリー ビューで選択するアセンブリの数を減らせば、ログ ファイルのサイズが小さくなり、パフォーマンスが向上します。

.NET Framework コール レポーティングを使用して、.NET インターフェイスに対するコールと戻り値を記録します。DevPartner エラー検出は、アセンブリの NLB ファイルが検出された位置に基づき、.NET モジュールの「ユーザー アセンブリ」と「システム アセンブリ」の区別を試みます。

.NET Framework コール レポーティングを有効にして、.NET アセンブリのリストを有効にするには、**[.NET** メソッド コール レポーティングを有効にする **]** チェック ボックスをオンにします。

[すべてのタイプ **]** ツリー ビューには、プロジェクトに関連付けられた .NET アセンブリが表示されます。**[** すべてのタイプ **]** エントリの隣にあるプラス（+）記号をクリックして、ツリーを展開します。ツリーの分岐には .NET ユーザー アセンブリと .NET システム アセンブリの両方が含まれています。各アイテムの隣のチェック ボックスをオンにすると、.NET Framework コール レポーティングに特定のアセンブリを選択できます。

リソースの追跡オプションを設定する

[リソースの追跡] を有効にすると、エラー検出では以下の処理を実行します。

- メモリ以外のシステム リソースの割り当てと解放を行うすべてのコールをアプリケーションで監視します。
- アプリケーションが終了しても解放されないリソースをレポートします。

リソースの追跡を有効にするには、リソースのリストを有効にして **[** リソースの追跡を有効にする **]** チェック ボックスをオンにします。関連のリストでチェック ボックスをオンにすると、エラー検出によってその DLL で作成されたリソースが追跡されます。

1 つまたは複数のリソース解放 API から選択すると、リソースの追跡を特定のリソース セットに絞り込むことができます。たとえば、レジストリ関連のリソースをすべて除外するには、**`advapi.dll`** リソースの **[RegCloseKey]** チェック ボックスをオフにします。

モジュールとファイル オプションを設定する

[モジュールとファイル] 設定を使って、アプリケーションを構成するモジュールを指定します。

モジュールまたはモジュールのコンポーネントを除外しても、インストゥルメンテーションには影響しません。インストゥルメンテーション マネージャを使用した場合にだけ、インストゥルメンテーションを制限することができます。

ヒント：このリストに明示的に追加したモジュールは評価の対象から除外することができます。DevPartner エラー検出には、[モジュールとファイル] に表示されていないモジュールも自動的に含まれます。表示されていないモジュールを除外する必要がある場合は、そのモジュールを追加してから、そのチェック ボックスをオフにして除外対象とします。

DevPartner エラー検出では、プログラム内のすべてのモジュールが自動的に評価されます。[モジュールとファイル] の設定を使用して、以下の操作を実行します。

- モジュールを評価の対象から除外する。

- モジュール内のコンポーネントを評価の対象から除外する。
- 評価するモジュールを追加する。

[モジュールとファイル] には以下の設定があります。

- モジュールとファイルのリスト：チェックするモジュールを表示します。
  - モジュール全体をチェック対象から除外するには、モジュールの隣のチェック ボックスをオフにします。
  - モジュールを展開してモジュールの内容を表示するには、モジュール パスの左にあるプラス記号をクリックします。
  - モジュール内の特定のアイテムを除外するには、モジュールを展開して、除外するアイテムの隣のチェック ボックスをオフにします。

    モジュールまたはモジュール内のアイテムの隣のチェック ボックスをオフにすると、そのモジュールまたはアイテムはリストに表示されますが、アプリケーションでエラー検出を実行するときには分析されません。

    1 つ以上のコンポーネントがクリアされている場合、モジュール名の隣のチェック ボックスは黄色で表示されます。

    [モジュールとファイル] 設定内のすべてのモジュールを無効した場合は、エラーの種類によって報告されるものとされないものがあります。DevPartner エラー検出では、任意のモジュール内のメモリ オーバーランと、**`MFCxxxx.dll`** ライブラリが原因で発生したその他のイベントは必ず報告されます。
- ソース コードが利用できる場合にのみ、エラーとリークを表示する：このチェック ボックスをオンにすると、レポートの対象が使用可能なソース コードがあるリークとエラーに制限されます。有効にした場合、レポートされるリークとエラーの数が減少する可能性があります。無効（デフォルト）にした場合は、すべてのリークとエラーが報告されます。
- モジュールの追加：クリックすると**[**追加するモジュールの選択**]**ダイアログ ボックスが開き、ここでモジュールの選択と追加を行います。
- モジュールの削除：クリックすると選択したモジュールが [モジュールとファイル] リストから削除されます。このボタンは、モジュールが選択されているときにのみ使用可能になります。主要な実行可能ファイルは削除できません。
- システム ディレクトリ：クリックすると、[システム ディレクトリ] ダイアログ ボックスが開きます。

システム ディレクトリ オプションを設定する

チェックする必要のないフォルダ全体を除外するには、[システム ディレクトリ] ダイアログ ボックスを使用します。たとえば、対処済みのエラーを発生させるモジュールが、フォルダに含まれている場合があります。チェックする必要のないこのようなフォルダを除外することによって、エラー検出セッションを高速化することができます。

DevPartner エラー検出では、発生源が未確認のエラー、およびアプリケーションの致命的エラーを引き起こすエラーがすべてレポートされます。このようなエラーは、除外されたフォルダ内のモジュールで発生する可能性がある場合でも、レポートされます。

[システム ディレクトリ] ダイアログ ボックスでは、以下の設定を使用できます。

- 追加 : [追加するシステム ディレクトリ] ダイアログ ボックスが開きます。このダイアログ ボックスで、エラー検出のチェックから除外するフォルダのリストに加えるフォルダを選択します。
- 削除 : 選択したシステム フォルダがリストから削除されます。
- **OK** : 変更内容を保存して [システム ディレクトリ] ダイアログ ボックスを閉じます。
- キャンセル:変更内容を破棄して[システム ディレクトリ]ダイアログ ボックスを閉じます。

ディレクトリ アイコン

[システム ディレクトリ] リストの各パス名の隣に表示されるディレクトリ アイコンは、以下の 2 つの状況を示します。

- 単一ディレクトリ : 1 つのフォルダ アイコンで表します。選択したフォルダの内容のみが含まれます。
- ディレクトリとすべてのサブディレクトリ : 3 つのフォルダ アイコンで表します。選択したフォルダとすべてのサブフォルダが含まれます。

2 つのオプションを切り替えるには、フォルダ名の隣にあるアイコンをクリックします。

メモ : 状況によっては、除外フォルダに格納されている重要なサードパーティ製 DLL を明示的に追加しなければならない場合があります。このようなサードパーティ製 DLL を明示的に追加することによって、そうしなければ特定されなかった問題が明らかになることがあります。DLL を明示的に追加するには、[モジュールとファイル] 設定を使用します。

フォントと色オプションを設定する

[フォントと色] は、エラー検出ウィンドウのタブに表示される項目の外観を制御します。たとえば、頻繁に表示されるエラー データのフォント サイズを大きくしたり、タブに多くの情報を表示できるようにフォント サイズを小さくしたりできます。

以下のコントロールを使用して、フォントと色を選択します。

- 以下の設定を表示 : 検証結果ペインに表示されるさまざまなタブのリストが表示されます。フォントと色を変更するタブを選択します。
- デフォルトを使用 : クリックすると、現在のすべての設定がキャンセルされ、元のフォントと色に戻ります。
- 表示可能な項目 : フォントや色のプロパティを変更する項目をこのリストから選択します。
- フォント:**[**表示可能な項目**]**で、現在選択されている項目に使用するフォントを選択します。
- サイズ:**[** 表示可能な項目 **]** で、現在選択されている項目に使用するフォント サイズを選択します。
- 前景項目 : **[** 表示可能な項目 **]** リストで現在選択されている項目の前景色が表示されます。このドロップダウン メニューから前景色を選択するか、メニューの左側にある **[** カスタム **]** をクリックして、カスタムの前景色を定義します。
- 背景項目 : **[** 表示可能な項目 **]** リストで現在選択されている項目の背景色が表示されます。このドロップダウン メニューから背景色を選択するか、メニューの左側にある **[** カスタム **]** をクリックして、カスタムの背景色を定義します。
- 太字 : 選択すると、表示可能な項目が太字で表示されます。

- タブのサイズ：このコントロールを使用して、ソース コード ペインに表示されるコードのインデント サイズを指定します。

  このコントロールは **[** 以下の設定を表示 **]** の選択が [ソース ペイン] で、**[** 表示可能な項目 **]** の選択が **[** メイン **]** の場合にのみ使用可能になります。

- サンプル テキスト ボックス:[フォントと色] ウィンドウの下部にあるテキスト ボックスに、選択したフォントと色の組み合わせを使うと、現在表示可能な項目がどのように表示されるかが示されます。

構成ファイル管理オプションを設定する

[構成ファイル管理] 設定を使って、構成ファイルを管理します。現在使用している構成ファイルの名前が [プログラム設定] ダイアログ ボックスのタイトル バーに表示されます。

[プログラム設定] ダイアログ ボックスの設定を変更すると、構成ファイル名にアスタリスクが付加されます。このアスタリスクは、プロパティを保存するか、ファイルを再ロードするか、別のファイルをロードするまで表示されます。別のファイルをロードしたり、保存せずにファイルを再ロードした場合、現在のファイルに加えた変更は失われます。

以下のコントロールを使用して、構成ファイルの機能を選択します。

- 構成ファイル名：構成ファイルのフル パスと名前
- 再ロード：すべての変更を破棄して、現在の構成ファイルを再びロードします。この操作では、最後に保存された状態の構成ファイルがロードされます。
- ロード：[ロード元] ダイアログ ボックスを開きます。
  - **[** 内部ユーザー デフォルト **]** を選択して、ユーザーのデフォルト設定をロードします。
  - **[** 構成ファイル **]** を選択すると、[構成ファイルのロード] ダイアログ ボックスが開きます。このダイアログ ボックスで、ロードする別の構成ファイルを選択します。
- 保存：現在ロードされている構成ファイルの変更内容をすべて保存します。
- 名前を付けて保存:[構成ファイルの保存] ダイアログ ボックスを開きます。このダイアログ ボックスから、現在の構成オプションを別のファイル名で保存します。
- リセット：プログラム プロパティ設定をすべて出荷時のデフォルト設定に戻します。
- デフォルトの保存：現在の設定をユーザーのデフォルトとして保存します。新しいプロジェクトのすべてでこの設定が使用されます。
- デフォルトの削除 : ユーザーのデフォルトの構成設定を削除して、出荷時の設定に戻します。新しいプロジェクトのすべてで出荷時の設定が使用されます。

Windows メッセージとイベント ログを追跡する

Windows は、プログラムのほとんどが Windows メッセージとその他のイベントに応答して実行されるイベント駆動型の環境です。DevPartner はイベントが発生するとそれらをインターセプトして、ログに記録します。これらのログを使用して、問題を引き起こしたイベントの詳細な履歴を確認できます。

DevPartner は、以下のイベントをログに記録します。

- Windows メッセージ。

これらのイベントは、Windows メッセージに対応する際のプログラムの動作を示します。

- 引数情報と API コールおよび API の戻り値。これらのイベントは、プログラムでプロシージャが実行される順序を定義します。
- チェック対象のプログラムから出力されたデバッグ文字列メッセージ。
- エラー メッセージ。

データを XML にエクスポートする

DevPartner を使えば、最終的なセッション結果データを XML にエクスポートすることができます。これによって、簡単に、結果データを、レポート フォーマット、電子メール、内部の Web ページなどに移植することができます。

Visual Studio からデータをエクスポートする

Visual Studio でエラー検出を使用している場合は、以下の手順に従って、現在表示されているセッション ファイルからすべてのデータを XML ファイルにエクスポートすることができます。

1. エラー検出セッション ファイルを開きます。
2. **[** ファイル **] > [DevPartner** データのエクスポート **]** を選択します。

   [名前を付けて保存] ダイアログ ボックスが表示されます。
3. エクスポートするデータ ファイルの場所を選択します。
4. **[OK]** をクリックします。

エラー検出スタンドアロン アプリケーションからデータをエクスポートする

エラー検出スタンドアロン アプリケーションを使用している場合は、以下の手順に従って、現在表示されているセッション ファイルからすべてのデータを XML ファイルにエクスポートすることができます。

1. エラー検出セッション ファイルを開きます。
2. **[** ファイル **] > [** データのエクスポート **]** を選択します。

   [名前を付けて保存] ダイアログ ボックスが表示されます。
3. エクスポートするデータ ファイルの場所を選択します。
4. **[OK]** をクリックします。

コマンド ラインからデータをエクスポートする

実行時に適切なフラグを `BC.exe` に渡すか、または `BC.exe` を呼び出して既存のセッション ファイルを指定して、コマンド ラインからエラー検出を実行する際にセッション ファイル データから XML ファイルを生成するように選択できます。

DevPartner エラー検出では、XML 出力の生成時に、エラー検出インストール フォルダに配置されている **`DPSErrorDetection.xsd`** スキーマ ファイルが使用されます。このファイルは編集しないでください。

DevPartnerでセッション データをXMLにエクスポートできなかった場合は、問題の内容を伝えるエラー メッセージが表示されます。

セッションの実行とデータのエクスポート

実行可能ファイルを指定した場合は、エラー検出によって、その実行可能ファイル上のセッションが実行され、その結果からXML出力が生成されます。

```
BC.exe [/B session.DPbcl] [/X[S|D] xmlfile.xml] target.exe [args]
```

/Xと一緒にSフラグとDフラグを指定すると、サマリと詳細情報のどちらかをXMLにエクスポートすることができます。

メモ： 実行可能ファイルを指定した場合は、/Bフラグを使用して対応するセッション ファイルも指定する必要があります。

既存のファイルを変換する

セッション ファイル（**$ession.DPbcl**）だけを指定した場合は、エラー検出によって指定したセッション ファイルがXMLに変換され、その出力が保存されます。

```
BC.exe [/B session.DPbcl]  [/X[S|D] xmlfile.xml]
```

コマンド ラインからエラー検出を実行する

bc.exeまたは**bc.com**を使用して、DOSコマンド ラインからDevPartnerを実行できます。

メモ： 7.xバージョンの従来のサポートでは、DevPartnerエラー検出のスクリプト ファイルにbc7.comを引き続き使用できます。

- **bc.exe**はDevPartnerスタンドアロンのUIを起動します。
- **bc.com**は**bc.exe**を起動する小規模なコンソール プログラムであり、bc.exeが完了するまで待機します。

バッチ スクリプトでは、**bc.exe**と**bc.com**の違いが重要です。**bc.exe**を直接起動するとDevPartnerが開始され、**bc.exe**が完了するまで待機せずに次のコマンドに進みます。スクリプトの次の手順が結果のチェックの場合は、このファイルを使用できません。

bcとだけ入力すると、OSは**bc.exe**ではなく**bc.com**を選択します。

詳細については、DevPartner『エラー検出ガイド』の「コマンド ラインからエラー検出を使用する」を参照してください。

コマンド ライン オプションと構文

[ ]で囲んだコマンドはオプションです。

```
BC.exe [/?]
```

```
BC.exe session.DPbcl
```

```
BC.exe [/B session.DPbcl] [/C configfile.DPbcc] [/M] [/NOLOGO]
[/X[S|D] xmlfile.xml] [/OUT errorfile.txt] [/S] [/W workingdir] target.exe
[args]
```

表2-4. コマンド ライン オプション

オプション	動作
/?	使用状況を表示します。
session.DPbcl	既存のセッション ファイルを開きます。
/B *session*.DPbcl	バッチ モードで実行して、セッション ファイルをログ ファイルの **session.DPbcl** に保存します。
/C *configfile*.DPbcc	**configfile.DPbcc** オプションを使用します。
/M	**bc.exe** を起動し、実行中は最小化します。
/NOLOGO	**bc.exe** のロード中にスプラッシュ画面を表示しないようにします。
/X *xmlfile*.xml	XML 出力を生成して指定されたファイルに保存します。 実行可能ファイルを指定した場合は、エラー検出によって、その実行可能ファイル上のセッションが実行され、その結果から XML 出力が生成されます。 セッション ファイル（**session.DPbcl**）だけを指定した場合は、エラー検出によって指定したセッション ファイルが XML に変換され、その出力が保存されます。 メモ： 実行可能ファイルを指定した場合は、/B スイッチを使用して対応するセッション ファイルも指定する必要があります。
/XS *xmlfile*.xml	/X パラメータと S 修飾子を一緒に使用すると、サマリ データだけが XML ファイルに保存されます。エラー検出セッションの実行に関する情報（セッション データ）はすべてエクスポートされます。
/XD *xmlfile*.xml	/X パラメータと D 修飾子を一緒に使用すると、詳細データだけが XML ファイルに保存されます。エラー検出セッションの実行に関する情報（セッション データ）はすべてエクスポートされます。
/OUT *errorfile*.txt	エラー メッセージをテキスト ファイルに出力します。
/S	サイレント モードで実行します。エラー発生時に [検出されたプログラム エラー] ダイアログ ボックスを表示しないようにします。
/W *workingdirectory*	ターゲットの作業フォルダを設定します。
target.exe [*args*]	起動する実行可能ファイルとその引数

実行可能プログラムが現在のパスにない場合は、プログラムのフル パスを指定する必要があります（実行可能ファイルを探すときにシステムが検索する場所を一覧にする環境変数）。

コマンド ラインから **FinalCheck** を実行する

コマンド ラインから FinalCheck を実行することもできます。詳細については、オンライン ヘルプの「**FinalCheck** でプログラムをチェックする」セクションにある以下のトピックを参照してください。

- コマンド ラインから FinalCheck を実行する
- NMCL オプション
- NMLINK オプション

Visual Studio Team System/Team Foundation Server にデータを送信する

DevPartner Studio では、Team Explorer クライアントがインストールされており、Team Foundation Server に接続可能な場合に、Microsoft Visual Studio Team System がサポートされます。

DevPartner エラー検出の Visual Studio Team System および Team Foundation Server サポート

Visual Studio 2005 および 2008 については、選択した項目に関するデータをバグタイプの作業項目として Visual Studio Team System に送信します。Visual Studio 2010 では、選択した項目に関するデータを問題、バグ、または不具合タイプの作業項目として Team Foundation Server に送信します。以下の [エラー検出] タブで作業項目の送信にアクセスします。

- [エラー] タブ – 選択したエラーの送信
- [メモリ リーク] タブ – 選択したリークの送信
- [モジュール] タブ – 選択したインスタンスの送信
- [その他のリーク] タブ – 選択したリークの送信
- [.NET パフォーマンス] タブ – 選択したインスタンスの送信

バグを送信すると、DevPartner によって、作業項目フォームにタブのデータが入力されます。DevPartner Studio と Visual Studio Team System の統合の詳細については、「Visual Studio Team System のサポート」（20 ページ）を参照してください。

第3章
静的なコード分析

この章には、2つのセクションが含まれています。1つめのセクションでは、はじめてのユーザーを対象として、コード レビューを実行する簡単な手順について説明します。2つめのセクションでは、DevPartner Studioのコード レビュー機能を詳細に把握するための参照情報を示します。

コード レビューに関するタスクベースの追加情報については、DevPartner Studioのオンライン ヘルプを参照してください。

コード レビューとは

DevPartnerコード レビュー機能により、開発者がVisual Studioでベスト プラクティスに準拠したVisual BasicおよびVisual C#コードを記述することができます。DevPartnerコード レビューは、プログラミングやネーミングの違反を識別し、メソッド コール構造を分析して、コード全体の複雑度を追跡します。

メモ： コード レビュー機能はマネージ コードのみを分析し、DevPartner for Visual C++ BoundsChecker Suiteではサポートされていません。

DevPartnerコード レビュー機能では、以下の機能を利用できます。

- 静的なコード分析とレビュー

  DevPartnerコード レビューでは、Visual Studioのソース コードに対する総合的な静的なコード分析を実行し、[DevPartnerコード レビュー]ウィンドウに結果を表示します。

- 自動コマンド ライン バッチ処理

  コマンド ラインからソリューションのバッチ レビューを実行できます。これらの自動バッチ レビューは、毎晩のビルドと組み合わせて実行できます。また、大規模なアプリケーションでは、他のタスクの実行中に自動バッチ レビューを使用することによって、時間を節約することもできます。

- **XML**へのデータ エクスポート

  DevPartnerコード レビューでは、セッションの結果をXML形式にエクスポートできます。これによって、簡単に結果データをレポート フォーマット、電子メール、社内Webページなどに変換できます。データは、セッション実行後のコード レビューから、コマンド ラインから、または自動バッチ処理の一部としてXMLにエクスポートすることができます。

- ルールの管理とカスタマイズ

  ルール マネージャでは、コード レビューで使用されるルールを設定して、設定した標準にコードを準拠させることができます。また、ルールをレビュー セッション内で使用されるセットにグループ化して、独自のカスタム ルールを作成することもできます。

簡単な手順でコード レビューを使用する

以下の準備、設定、実行の手順に従ってDevPartnerコード レビューを使用できます。

すぐに使い始めるには、影付きのボックス内に示された手順に従います。影付きのボックス内に説明されている内容の詳細については、ボックスの下の追加説明を参照してください。

DevPartnerコード レビューでは、ターゲット アプリケーションごとにデータ ファイルが作成されます。コード レビューを開始する前に、ターゲット実行ファイルを含むフォルダへの書き込みアクセス権があることを確認する必要があります。

DevPartner Studioでアプリケーションを分析する場合、昇格されたシステム権限は必要ありません。DevPartnerでのアプリケーションの分析には、アプリケーションの作成とデバッグに使用するシステム権限で十分です。

準備：レビュー実行時の仕様を決定する

DevPartnerコード レビューは非常に柔軟性が高く、セッションに対するさまざまな設定を検討する必要があります。

以降の手順では、以下の事項を前提としています。

- Visual BasicまたはVisual C#の単一開発者のソリューションのレビューを実行しています。
- サポートされているVisual Studioのリリースでコード レビューを実行しています。
- ソリューション内のすべてのプロジェクトが、エラーなしでコンパイルされています。
- レビュー対象のすべてのプロジェクトが、デバッグ情報を出力するように設定されています。

コード レビューでサポートされているすべてのプロジェクト タイプのリストは、「コード レビューでサポートされているプロジェクト タイプ」 174ページ）を参照してください。

- 適用するルールの決定 – 幅広いコード レビュー ルールを使用でき、コードに業界のベスト プラクティスを適用できます。追加の基準を適用する場合は、ルール マネージャを使用してカスタムのルールとルール セットを作成することもできます。
- 適用するネーミング ガイドラインの選択 – DevPartnerコード レビューでは、コードが業界で認可されたネーミング標準に必ず従うように、組み込みのネーミング アナライザを使用できます。
- メトリクス データの収集 – レビュー中にメトリクス データを収集できます。McCabeメトリクスに基づいて、コードの複雑度の結果（複雑度、不良修正の可能性、および理解度）が表示されます。
- コール グラフ データの収集 – レビュー中にコール グラフ データ（潜在的なすべてのインバウンドコールとアウトバウンド コールを表すデータ）を収集できます。
- ソリューション内のプロジェクトの除外 – DevPartnerコード レビューには、デフォルトでソリューション内のすべてのプロジェクトが含まれます。ソリューション内にコード レビューで分析しないプロジェクトがある場合、該当のプロジェクトを除外できます。

メモ： 選択したすべてのプロジェクトでデバッグ情報を出力するように設定する必要があります。選択したプロジェクトで、使用可能ないずれのビルド構成でもデバッグ情報を出力するように設定されていない場合は、コード レビューを実行するとビルド エラーの警告が表示され、そのプロジェクトは以降のセッションから除外されます。

設定：オプションと設定を選択する

DevPartner コード レビューは柔軟性が高く、カスタマイズ可能です。コード レビューをカスタマイズするには **[** 全般 **]** オプション ページを使用します。[全般] オプション ページにアクセスするには、**[DevPartner]** > **[** オプション **]** を選択し、[オプション] ツリー ビューから **[** コード レビュー **]** を選択します。

この手順では、DevPartner のデフォルトのプロパティとオプションを使用できます。設定を変更する必要はありません。

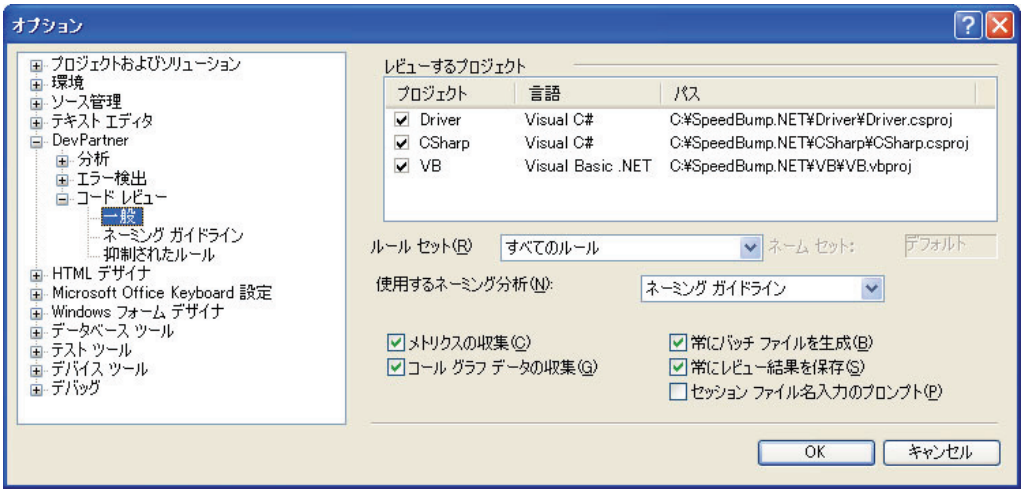


図 3-1 DevPartner コード レビュー [全般] オプション

- ルール セットの選択 – レビューの実行前に **[**ルール セット**]** リストからルール セットを選択できます。デフォルトのルール セットには、コード レビューが提供する優先度が中と高のルールがすべて含まれており、ユーザーは業界共通のベスト プラクティスに準拠できます。表 3-2 (68 ページ) に、コード レビューに備えられている標準ルール セットのリストを示します。

  ルール マネージャ (「コード レビュー ルール マネージャを使用する」 (96 ページ) を参照) を使用して、カスタムのルールとルール セットを作成できます。

- ネーミング ガイドラインの選択 – **[**使用するネーミング分析**]** リストからネーミング ガイドラインを選択できます。デフォルト動作では、コード レビューは Microsoft .NET ネーミング ルールに沿ってモデル化されたネーミング ガイドラインに従います。ただし、代わりにハンガリアン記法ネーミング ルールに従うことや、まったくルールを適用しないことも可能です。

- メトリクス データの収集の有効化または無効化 – **[**メトリクスの収集**]** チェック ボックスをオンにして、McCabe メトリクス データの収集を有効にします (「McCabe メトリクスを収集する」 (69 ページ) を参照)。この機能を無効にするには、チェック ボックスをオフにします。

- コール グラフ データの収集の有効化または無効化 – **[**コール グラフ データの収集**]**チェック ボックスをオンにして、静的メソッド コール データの収集を有効にします。この機能を無効にするには、チェック ボックスをオフにします。

  この機能を有効にしてレビューを実行すると、[結果] ウィンドウの **[** コール グラフ **]** タブに、左端のペインのソリューション ツリーから選択したメソッドまたはプロパティに対応するインバウンド /アウトバウンド コール パスが静的なグラフとして表示されます。

  コール パスは静的に生成されます。つまり、グラフには、プログラム実行時に発生した動的なメソッド コールではなく、コール パスにおける潜在的なメソッド コールが表示されます。

- 不要なプロジェクトの除外 – **[** レビューするプロジェクト **]** テキスト ボックス内の各プロジェクトの横にあるチェック ボックスは、そのプロジェクトがコード レビューの分析対象になるかどうかを制御します。コード レビューの分析対象としないプロジェクトに関連するチェック ボックスはオフにします。

実行：コード レビュー セッションを開始する

DevPartnerのコード分析のプロセスはセッションと呼ばれます。レビューが完了すると、[結果] ウィンドウにセッション データが表示され、コード レビューの停止時にファイルに保存されます。

1. Visual Studioでソリューションを開きます。
2. **[DevPartner]** > **[** コード レビューの実行 **]** を選択します。

   DevPartnerによって、ソリューション内のすべてのプロジェクトに対してコード レビューが実行されます。[結果] ウィンドウが開き、サマリ ペインのステータス バーがセッションの進捗状況を追跡します。

基本のコード レビュー セッションの実行が完了し、[結果] ウィンドウにユーザーの分析対象のデータが集約されます。

結果を分析して違反を修正する

ソリューションのレビューの実行が完了したら、[結果] ウィンドウを確認します。[結果] ウィンドウに表示されるセッション データを利用して、違反の識別、位置の特定、修正を開始します。

1. [結果] ウィンドウの **[** 問題 **]** タブで、レビュー中に発見されたコード違反を確認します。
2. **[** 重要度 **]** カラムの違反が優先度高から低の順にソートされていること（デフォルトの動作）を確認します。必要に応じてカラムの見出しをクリックして、カラムの昇順と降順を切り替えます。

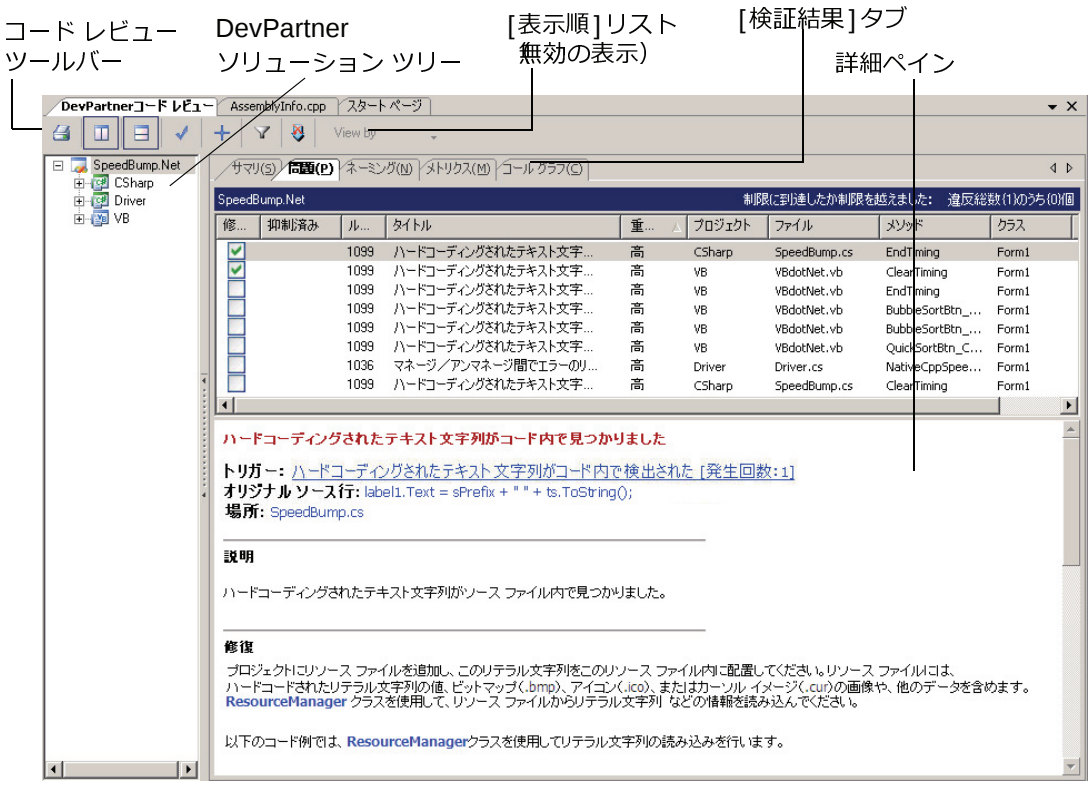


図 3-2 コード レビューの[結果]ウィンドウ

通常は、最も重大なコード違反を最初に修正します。[問題]タブではコード違反が重要度の順にソートされるように設計されているため、簡単に、最優先の違反から先に選択できます。

[結果]ウィンドウでは、セッション データを個々のカテゴリに分類する複数のタブを利用できます。

- レビュー中に発見された各種の違反タイプをまとめたレポートを検証するには、[サマリ]タブを選択します（「サマリ データを表示する」（73 ページ）を参照）。
- レビュー中に発見されたコード違反を表示するには、[問題]タブを選択します。[問題]タブのデフォルト動作では、違反のリストが重要度高から低の順にソートされます（「コード違反を表示する」（75 ページ）を参照）。
- レビュー中に発見されたネーミング違反を表示するには、[ネーミング]タブを選択します。このリストには必要に応じて修正案が提示され、ネーミングを無視するようにレビューが設定されている場合は、空になります（「ネーミング違反を表示する」（76 ページ）を参照）。
- McCabe メトリクスに基づいて、コードの複雑度の結果（複雑度、不良修正の可能性、および理解度）を表示するには[メトリクス]タブを選択します（「収集されたメトリクスを表示する」（79 ページ）を参照）。
- メソッド コールをグラフィカルに表示するには、[コール グラフ]タブを選択します（「コール グラフ データを表示する」（82 ページ）を参照）。

結果をフィルタ処理する

コード レビュー セッションの実行後、結果に大量のデータが含まれているため、修正する1つの領域を絞り込むことが難しい場合があります。コード レビューのソリューション ツリーを使用すると、プロジェクト、ファイル、またはメソッドを選択して結果をフィルタできます。データをフィルタすることにより表示対象を制限し、自身にとって最も重要な結果を集中して確認できます。

コード違反を分析する

デフォルトで、[問題]タブでは、コード レビュー後に[結果]ウィンドウにフォーカスが設定されます。[問題]タブには、現在のソリューションで発見されたコード違反が表示されます。[問題]タブの下にある関連の詳細ペイン（図3-2 63ページ）を参照）では、詳細な説明、例、MSDNへの参照などの問題を説明するソースと、選択したコード違反の修正案（使用可能な場合）が提示されます。

3 [問題]タブで、リストの一番上（最優先）のコード違反を選択します。詳細ペインに、選択したコード違反に関する情報が表示されます。[トリガー]と[場所]の見出しには、コード違反の原因と違反の位置が示されます。

4 下にスクロールして[説明]、コード サンプル（使用可能な場合）、およびコード違反に対する修正案を確認します。さらに情報が必要な場合は、外部リンクをたどって違反の詳細な説明を確認します。

5 [問題]タブのリストでコード違反をダブルクリックします。

DevPartnerにより、Visual Studioエディタとソース コードを表示する新しいウィンドウが開きます。フォーカスは問題が発生しているコード行に設定されています。

6 Visual Studioエディタを使用してコード違反を修正します。

7 コード レビューの[結果]ウィンドウの[問題]タブに戻ります。

8 違反を修正したことを示すために、[修正済み]チェック ボックスをオンにします。

9 [問題]タブで次のコード違反を選択して、このセッションのコード違反に十分に対応したと思われるまで、手順5～8を繰り返します。

メモ： DevPartnerコード レビューは行番号の変更を継続的に追跡し、違反とソース コード間の同期を維持しようとします。大きな修正が行われたあと、結果でソース コードとの同期が失われ、行番号がずれる場合があります。ソース コードに変更を加えた後はコード レビュー セッションを再実行して、新しい[問題]タブのリストで引き続き違反を修正することもできます。

これで、コード レビューを使用してソリューション内のコード違反を解決できました。

ネーミング違反を分析する

[ネーミング]タブには、コード レビューでレビュー中に発見されたネーミング違反がリストされます。[ネーミング]タブの表示内容は、レビュー前に[全般]オプション ページ（図3-4 67ページ）を参照）で選択したネーミング分析のタイプに応じて異なります。[ネーミング]タブの下にある関連の詳細ペインには、図3-2 63ページ）の[問題]タブに関連付けられているペインと同様、選択したネーミング違反の詳細な説明、リソース、および修正案（使用可能な場合）が表示されます。

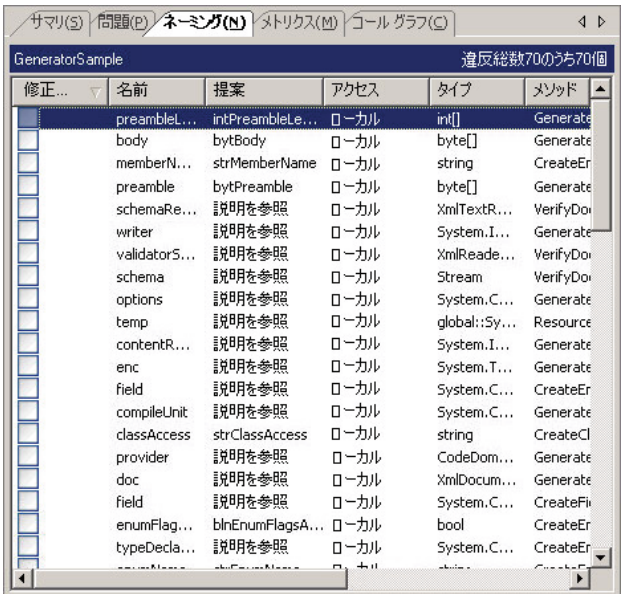


図 3-3 [ネーミング]タブと詳細ペイン

10 **[**ネーミング**]**タブを選択して、セッション中に発見されたネーミング違反を特定します（図 3-3 を参照）。レビュー中に発見されたネーミング違反はすべて、このタブに表示されます。使用可能な場合は、違反の横に正しいネーミングの案が表示されます。

11 **[**ネーミング**]**タブで最初のネーミング違反を選択します。

詳細ペインには、選択したネーミング違反に関するデータが表示されます（図 3-3 を参照）。[ハンガリアン]ネーミングを選択した場合は、詳細ペインは使用できません。

12 詳細な説明と適切なネーミング案（使用可能な場合）を確認します。違反についての情報がさらに必要な場合は、外部リンクをたどります。

13 **[**ネーミング**]**タブのリストでネーミング違反をダブルクリックして、ソースへ移動します。

14 ネーミング違反を修正します。

15 コード レビューの**[**結果**]**ウィンドウの**[**ネーミング**]**タブに戻ります。

16 違反を修正したことを示すために、**[**修正済み**]**チェック ボックスをオンにします。

17 **[**ネーミング**]**タブで次のネーミング違反を選択して、このセッションのネーミング違反に十分に対応したと思えるまで、手順 11 ～ 16 を繰り返します。

これで、コード レビューを使用してソリューション内のネーミング違反を解決できました。

セッション ファイルを保存する

セッション ファイルを保存すると、あとでそれらの結果を再度参照できます。以下の場合に保存したセッション ファイルを開く必要が生じます。

- 後日、セッション データを XML にエクスポートする（「データを XML にエクスポートする」（91 ページ）を参照）。
- このセッションで発見された違反をあとで引き続き修正する。

[全般]オプションにある**[**常にレビュー結果を保存**]**設定をオフにしない限り、コード レビューのデフォルトの動作で、終了時にセッション ファイルが保存されます （「[全般]オプションを設定する」 66ページ）を参照）。

1 [結果]ウィンドウにフォーカスを合わせて、**[**ファイル**]** > **[**コード レビュー セッションに名前を付けて保存**]**を選択します。

2 セッション ファイルの名前を入力して、**[**保存**]**をクリックします。

コード レビューでは、デフォルトでソリューションと同じ場所に**`SolutionName.dpmdb`**というセッション ファイルが保存されます。

DevPartnerでは、セッション ファイルはアクティブなソリューションの一部として保存されます。これらのファイルは、ソリューション エクスプローラのDevPartner Studio仮想フォルダに表示されます。DevPartnerコード レビューのセッション ファイルには拡張子**`.dpmdb`**が付きます。

デフォルトでは、セッション ファイルはプロジェクトの出力フォルダに保存されます。ファイル名は、デフォルト フォルダに配置されているファイル名に自動的に番号を追加する形式（たとえば、**`MyApp.dpmdb`**、**`MyApp1.dpmdb`**）に設定されます。セッション ファイルをデフォルト フォルダ以外の場所に保存した場合は、ファイル名はユーザーが管理することが必要です。

出力フォルダがないプロジェクトの場合（Visual Studio 2005 Webサイト プロジェクトなどの場合）、ファイルはプロジェクト フォルダに物理的に保存されます。

コマンド ラインで生成されたセッション ファイルは、プロジェクトのソリューションに自動的には追加されません。外部で生成されたセッション ファイルは、Visual Studioに開いたソリューションに手動で追加します。

これで、この章の準備、設定、実行のセクションは終了です。コード レビュー セッション実行のメカニズムの基礎が理解できました。詳細情報については、引き続きこの章の残りの部分を参照してください。

オプションを設定する

利用可能な多数のオプションを使用して、コード レビューの動作をカスタマイズします。ユーザーが指定した設定は、システム内のプリファレンス データベースに保存されます。DevPartnerには、コード レビューのオプションを修正するための3つのオプション ページが用意されています。

- [全般]オプション
- [ネーミング ガイドライン]オプション
- [抑制されたルール]

[全般**]**オプションを設定する

[全般]オプション ページでは、コード レビューの設定を行います。これらの設定は、コード レビューの事前に変更できます。[全般]オプションにアクセスするには、**[DevPartner]**メニューで**[**オプション**]**を選択し、ツリー ビューから**[DevPartner]** > **[**コード レビュー**]** > **[**全般**]**を選択します。

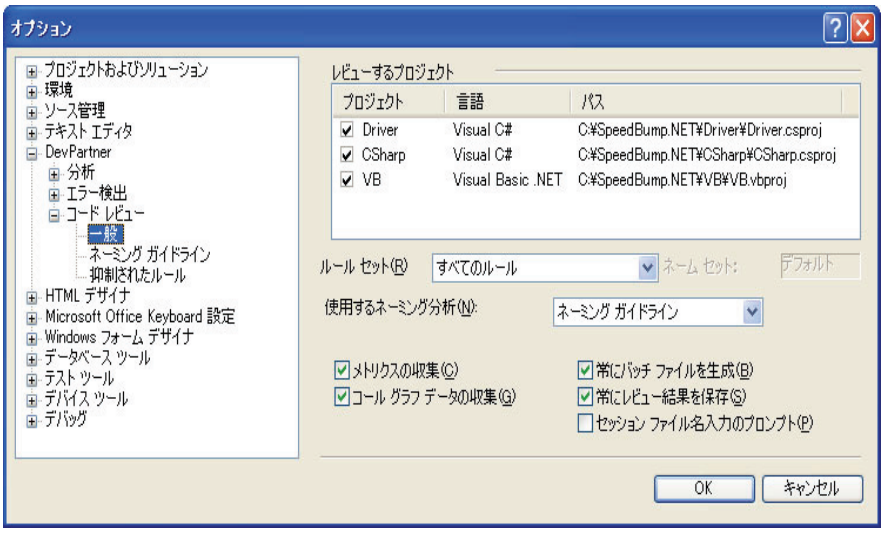


図 3-4 [全般] オプション ページ

レビューするプロジェクトを選択する

[レビューするプロジェクト] リストからソリューション内の一部またはすべてのプロジェクトを選択できます。このリストは、以下の場合に空になります。

- Visual Studio に C# または Visual Basic ソリューションがロードされていない。
- C++ プロジェクトのみを含むソリューションがロードされている。

[レビューするプロジェクト] リストには、以下の情報が含まれています。

表 3-1. レビューするプロジェクト リスト

項目	説明
チェック ボックス	オンにすると、該当のプロジェクトがレビューされる
プロジェクト	プロジェクトの名前
言語	プロジェクトに関連する以下の Visual Studio 言語 • Visual Basic • C# • Web サイト
Path	リスト内のプロジェクトのパスおよび名前

プロジェクトのリストから何も選択しなかった場合、DevPartner はデフォルトで現在のソリューション内のすべてのプロジェクトをレビューします。プロジェクトのリストを編集すると、DevPartner ではこのソリューションを次回操作するときのために、対象または除外のプロジェクト状態を保存します。この場合は、以下の注意が適用されます。

- ソリューションがレビューされるためには、少なくとも 1 つのプロジェクトが選択されている必要があります。この条件が満たされない場合、コード レビューは続行されません。

- 選択したすべてのプロジェクトでデバッグ情報を出力するように設定する必要があります。選択したプロジェクトが有効なビルド構成に関するデバッグ情報を出力するように設定されていない場合は、ビルド エラーが発生し、そのプロジェクトが以降のセッションから除外されます。
- ソリューションには存在しなくなった 1 つまたは複数のプロジェクトを選択すると、まだ存在している残りのプロジェクトがレビューされます。
- 後でレビューを試みるソリューション内のすべてのプロジェクトを誤って削除した場合、ソリューションに存在しなくなったプロジェクトが選択されたことを示すメッセージが表示されると共に、[全般] オプション ページで適切な変更を加えるよう促されます。
- ただし、特定のプロジェクト内のファイル、クラス、またはメソッドを個別に選択することはできません。

ルール セットを選択する

[ルール セット**]** リストからルール セットを選択して、コード レビューに適用できます。**[**ルール セット**]** リストには、DevPartner 提供のルール セットとユーザー定義のルール セットがすべて表示されます。選択したルール セットは保存され、現在のソリューションでセッションを実行するたびに使用されます。

メモ： ルールを保持し、ルール データベースにすでに格納されている有効なルール セットを選択してください。ルール マネージャを介して削除されたルール セットまたは空のルール セットを使用しようとすると、結果が無効になる可能性があります。

ルール マネージャを使用すると、ルール（ルールに違反した場合に、ルール適用の原因となる関連トリガーを含む）を作成してカスタマイズしたり、ルール セットを作成して管理したりできます （「コード レビュー ルール マネージャを使用する」 96 ページ）を参照。

表 3-2. 標準のルール セット

ルール セット名	説明
すべてのルール	簡単な手順で使用でき、以下のような機能を持つマスタ ルール セットを提供します。 • ルール データベース内のすべての DevPartner ルールを含む • ルール データベース内にユーザー定義されたすべてのルールを含む • 総合的なコード レビューを保証する
日付形式	日付の値（特に年を 2 桁で表記した日付）の正しい形式設定および使用方法をチェックするルール
デフォルト	高優先度ルールと中優先度ルールを含む
デザイン タイム プロパティ	適切な UI の設計に役立つデザイン タイム プロパティとフォームおよびコントロールのプロパティ値をチェックするルール
国際化	海外市場向けのローカライズ、文字列の処理、および比較に役立つルール
ロジック	正しいプログラム ロジック、.NET フレームワークの最善のプログラミング方法、エラー処理、タイプ チェック、およびガベージ コレクションをチェックするためのルール
パフォーマンス	パフォーマンスに悪影響を及ぼすコードをチェックするためのルール

表 3-2. 標準のルール セット

ルール セット名	説明
ネーミング ガイドライン	ソース コードで複数の識別子に関わる.NET フレームワーク ネーミングの矛盾を検索するためのルール
Web アプリケーション	適切な ASP.NET 開発、HTML タグの使用、検証、パフォーマンス、キャッシュ、および状態をチェックするルール

ネーミング分析の種類を選択する

メモ： ネーミング分析の種類を選択する必要があります。選択していないと、DevPartner では重要な分析機能が実行されません。プログラミング上のその他の問題に注意を集中する間、ネーミングの異常を一時的に無視するには、**[**なし**]** を選択します。

[使用するネーミング分析**]** リストから、レビューに適用するネーミング分析の種類を選択します。選択肢は以下のとおりです。

- ネーミング ガイドライン（デフォルト）: Visual Studio .NET フレームワークのネーミング ガイドラインを模範としています。

  また、より正確なレビューが行われるようにするには、[ネーミング ガイドライン] オプション ページのオプションを設定する必要があります（「[ネーミング ガイドライン] オプションを設定する」 70 ページ）を参照）。

- ハンガリアン : ハンガリアン記法ネーミング ルールを模範としています（「ハンガリアン ネーミング アナライザを理解する」 95 ページ）を参照）。

  また、有効なハンガリアン ネーム セットを選択する必要があります。ネーミング ガイドライン ネーミング分析には、ネーム セットの選択は適用されません。

- なし : DevPartner では、ネーミング分析が実行されず、コード レビューのあと、**[**ネーミング **]** タブは空になります。

ネーム セットを選択する

ソース コードでハンガリアン ネーミング分析を実行する場合は、有効なハンガリアン ネーム セットも選択してください（デフォルトの DevPartner 提供のルール セットには、あらかじめデフォルト ネーム セットが関連付けられています）。

ルール マネージャを使用して、ネーム セットを作成したり管理したりできます（「コード レビュー ルール マネージャを使用する」 96 ページ）を参照）。

McCabe メトリクスを収集する

[メトリクスの収集**]** を選択した場合、コード レビューでは複雑度、不良修正の可能性、理解度などのコード複雑度の統計を表示するデータが収集されます。これらのメトリクスは、業界標準の McCabe メトリクスに基づいています（「McCabe メトリクスを理解する」 80 ページ）を参照）。[メトリクス] タブには、コード レビュー ソリューション ツリーで選択したノードに関するすべての項目が表示されます。

コール グラフ データを収集する

[コール グラフ データの収集**]**チェック ボックスをオンにした場合、メソッドまたはプロパティに対する潜在的なすべてのインバウンド コールとアウトバウンド コールについて情報が収集され、結果が**[**コール グラフ**]**タブにグラフとして表示されます。コール グラフの個々のノードは、選択されたメソッドまたはプロパティのインバウンド /アウトバウンドのコール パスを表すものです。コール グラフには、プログラム実行時に発生した動的なコールではなく、コール パスにおける潜在的なメソッド コールが静的に表示されます。

バッチ ファイルを生成する

[常にバッチ ファイルを生成**]**を選択すると、Visual Studio で実行される次の対話的コード レビューでバッチ ファイルが生成されます。このバッチ ファイルを使用すると、同じソリューションでコマンド ラインからバッチ レビューを実行できます。

メモ : バッチ ファイルまたはコマンド ラインでレビューを実行するときに /r オプションを使用する場合は、**[**常にバッチ ファイルを生成**]**をオフにするか、バッチ ファイルをバックアップして名前を変更する必要があります。それらの作業を行わない場合は、バッチ ファイルが上書きされます。「コマンド ライン インターフェイスを使用する」（87ページ）を参照してください。

レビュー結果を保存する

[常にバッチ ファイルを生成**]**チェック ボックスをオンにした場合、DevPartner では、コード レビュー後のソリューションと同じ場所に **`SolutionName.dpmdb`** というセッション ファイルが保存されます。Visual Studio のソリューション エクスプローラに保存したセッション ファイルが表示されます。

セッション ファイル名入力のプロンプト

[セッション ファイル名入力のプロンプト**]**チェック ボックスをオンにした場合、レビューを開始する前にセッション ファイルの場所と名前の指定を求めるプロンプトが表示されます。

[ネーミング ガイドライン**]**オプションを設定する

[ネーミング ガイドライン] オプション ページには、より正確なレビューを保証するためのオプションが用意されています。[ネーミング ガイドライン] オプションにアクセスするには、[DevPartner] メニューで **[**オプション**]** を選択し、ツリー ビューから **[DevPartner]** > **[**コード レビュー **]** > **[**ネーミング ガイドライン**]** を選択します。

このページの選択内容は、[全般]オプション ページの[使用するネーミング分析]リストから[ネーミング ガイドライン]ネーミング アナライザを選択するまで無効です （図 3-4 67ページ）を参照。

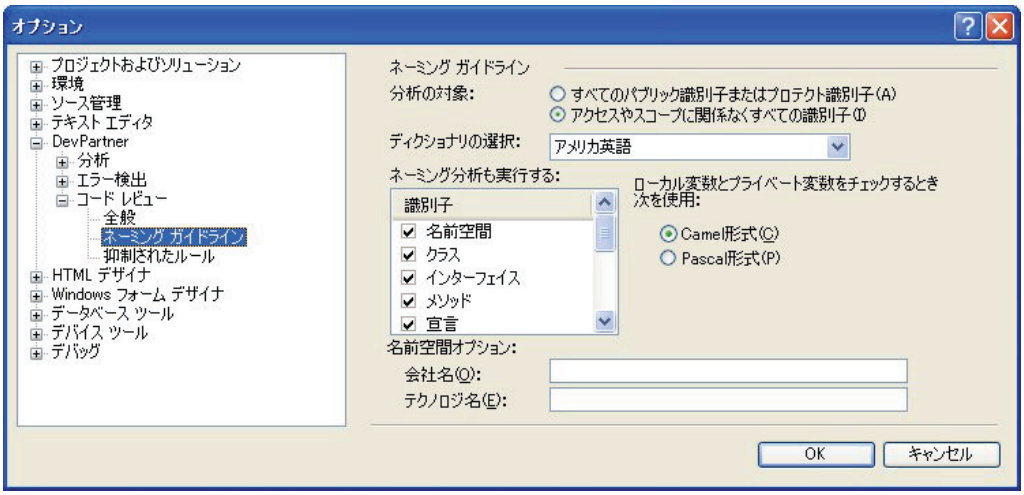


図 3-5 [ネーミング ガイドライン]オプション ページ

分析する識別子を選択する

[分析の対象]セクションで、分析に含める識別子の種類を選択します。

- すべてのパブリック識別子またはプロテクト識別子（デフォルト）：パブリックまたはプロテクト識別子および内部プロテクト識別子を調べます。ただし、このオプションではローカルとプライベートは除外されます。
- アクセスやスコープに関係なくすべての識別子：アクセスやスコープにかかわらず、すべての識別子を調べます。

ディクショナリを選択する

[ディクショナリの選択]リストから、ネーミング分析に適用するディクショナリ データベースを選択します。選択したディクショナリに基づいてネーミング違反が検索されます。[アメリカ英語]がデフォルトのディクショナリです。

ネーミング分析のスコープを選択する

[ネーミング分析も実行する]リストで、DevPartner が分析する 1 つまたは複数の識別子に対応するチェック ボックスを選択します。デフォルトでは、すべての識別子が選択されます。

[変数]チェック ボックスをオンにすると、変数が指定するこのオプション ページの他の選択内容（[分析の対象]など）に影響する可能性があります。

Camel 形式または Pascal 形式を選択する

名前を指定した変数を検証する場合に DevPartner が使用する大文字のプリファレンスを選択します。Camel 形式と Pascal 形式という 2 つのオプションがあります。Camel 形式では、変数の最初の単語を小文字始まりとし、2 番めの単語を大文字始まりとします（`iNtegerBonus` など）。Pascal 形式では、変数名の各単語を大文字始まりにします（`IntegerBonus` など）。

[ネーミング分析も実行する]リストから[変数]を選択済みでない場合、このオプションは無効になります。また、[すべてのパブリック識別子またはプロテクト識別子]を選択した場合も、このオプションを使用できません。

[名前空間]オプションを選択する

[ネーミング分析も実行する]フィールドの[名前空間]チェック ボックスをオンにした場合、追加の名前空間オプションを指定できます。

- 会社名：会社名を表す文字列を入力します。
- テクノロジ名：会社のテクノロジを表す文字列を入力します。

DevPartner コード レビューでは、名前空間における大文字の適切な使用、単語の完全性、予約された単語の有無、数字の使用などを検証します。DevPartner また、各名前空間が推奨される名前空間構文　会社名 . テクノロジ名 [. 機能] [. デザイン]）に従っているかどうかも検証します。会社名やテクノロジ名を入力すると、それらのエントリを使用して名前空間が構成されているかどうかがチェックされます。

抑制されたルールを管理する

[抑制されたルール]オプション ページには、[問題]タブで抑制されたルールのリストが表示されます（図 3-6 を参照）。[抑制されたルール]オプション ページで、抑制されたルールの横にあるチェック ボックスをオンにすると、抑制されたルールは一時的に抑制解除できます。[抑制されたルール]オプションにアクセスするには、[DevPartner]メニューで[オプション]を選択し、ツリー ビューから**[DevPartner]>[**コード レビュー**]>[**抑制されたルール**]**を選択します。

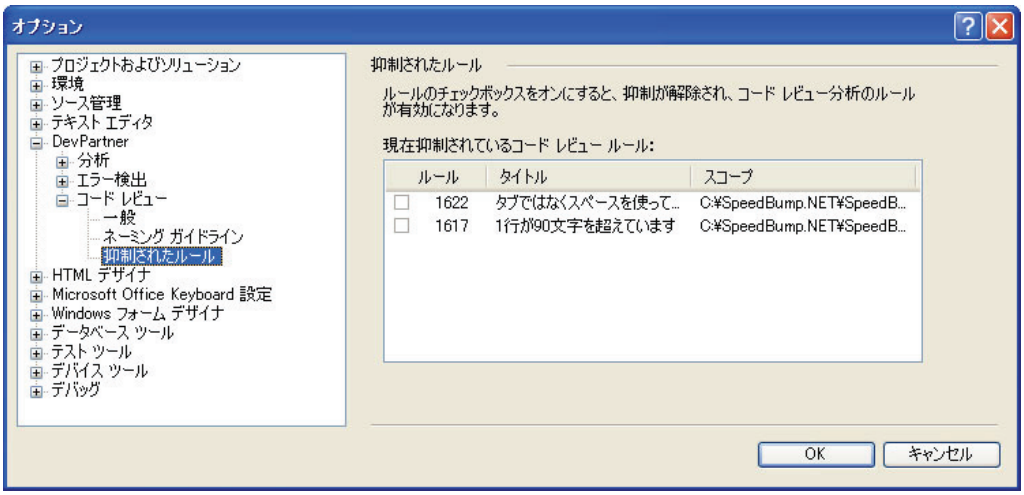


図 3-6 [抑制されたルール]オプション ページ

ルールを抑制する

ルールを抑制すると、コード レビューでは以降のセッションでそのルールを適用しません。ルールの抑制は、以下の点でコード違反のフィルタとは大きく異なります。

- ルールを抑制した場合、そのルールはまったく適用されなくなります。データは収集されず、セッション ファイルには何も保存されません。

- コード違反をフィルタした場合、基準となるルールは引き続き適用されます。データは収集されてセッション ファイルに保存されますが、表示されません。

ルールの抑制は、特定のソリューションでローカルに保存することもできますし、すべてのソリューションにわたって保存することもできます。抑制するルールを選択する前に、まずコード レビューを実行する必要があります。

1. **[**結果**]**ウィンドウの**[**問題**]**タブをクリックします。

   [問題**]**タブにコード違反がリストされます。

2. 基準のルールを抑制するコード違反を選択します。

3. 以下のどちらかの方法で、[ルールの抑制]ダイアログ ボックスを表示します。

   - **[**ルールの抑制**]**ツールバー ボタンをクリックします。
   - 強調表示されたルール行を右クリックし、コンテキスト メニューから**[**ルールの抑制**]**を選択します。

4. 選択されたルールを抑制するスコープを選択します。

   - このソリューションのすべてについてこのルールを抑制：今後行われる現在のソリューションのレビューに影響を与えます。
   - すべてのソリューションに適用：今後行われるすべてのソリューションのレビューに全体的に影響を与えます。

   全面的な抑制を選択した場合、プリファレンス データベースでは、そのルールのソリューションベースの抑制がすべてクリアされ、すべてのソリューションに全面的な抑制の設定が適用されます。

ルールが他のソリューションですでに抑制されている場合に、現在のソリューションでそのルールを抑制しようとすると、[ルールの抑制]ダイアログ ボックスが表示され、全面的な抑制の適用が要求されます。ただし、現在のソリューションのみで抑制するように選択することもできます。

サマリ データを表示する

[サマリ**]**タブでは結果データのサマリを一元管理しており、セッションの各側面の詳細は対応する別のタブに表示されます。**[**サマリ**]**タブのいくつかの項目は動的です。**[**問題**]**または**[**ネーミング**]**タブの項目に**[**修正済み**]**とマークされている場合、**[**サマリ**]**タブには動的に更新が反映されます。レビューに特殊な例外　空のルール セットでのレビュー実行など）が含まれていた場合は、**[**サマリ**]**タブのヘッダー セクションに該当のメッセージが表示されます。**[**サマリ**]**タブを下にスクロールすると、各サマリ テーブルを参照できます。

図3-7 [サマリ]タブ

- **[問題サマリ]**テーブルには、レビューで評価されたルールのカテゴリがリストされます。発見された違反数と修正済みとマークされた違反数を示します。さらに、重要度のカテゴリごとに違反の合計数を示します。
- **[ネーミング ガイドラインのサマリ]**リストには、レビューに含めるように[ネーミング ガイドライン] オプション ページであらかじめ選択したカテゴリが表示されます（図 3-5 71ページ）を参照）。テーブルには [ネーミング ガイドライン] オプション ページで選択したネーミング識別子のサマリが表示され、発見された違反数を示します。

  このテーブルは、レビュー前に[全般]オプション ページで**[ネーミング ガイドライン]**を選択した場合のみ、**[サマリ]**タブに表示されます（図3-4 67ページ）を参照）。ハンガリアン ネーミング アナライザでは、このテーブルは使用できません。
- **[コール グラフ データのサマリ]**では、分析したメソッドとプロパティの合計数、未コール状態の検出数など、レビュー時にキャプチャしたコール グラフ分析の情報を表示します。

  このテーブルは、レビュー前に[全般]オプション ページで**[コール グラフ データのサマリ]**を選択した場合のみ、**[サマリ]**タブに表示されます（図3-4 67ページ）を参照）。
- **[カウント サマリ]**には、実行にかかった時間、ソリューション内の行数、実行された比較回数など、コード レビュー セッション自体について集計された個々の統計も含まれます。
- **[レビュー設定]**には、構成データとレビュー関連データがリストされます。この情報は、記録の保持とトラブルシューティングに役立ちます。
- **[プロジェクト リスト]**には、各プロジェクトにコンパイル エラーがあったか、正常にレビューされたかなど、ソリューション内の各プロジェクトの情報が表示されます。

コード違反を表示する

デフォルトで、[問題]タブでは、コード レビュー後に[結果]ウィンドウにフォーカスが設定されます。[問題]タブには、現在のソリューションで発見されたコード違反が表示されます。特定の違反を選択すると、コード違反リストの下の詳細ペインに詳細な説明、例、および修正案が表示されます。

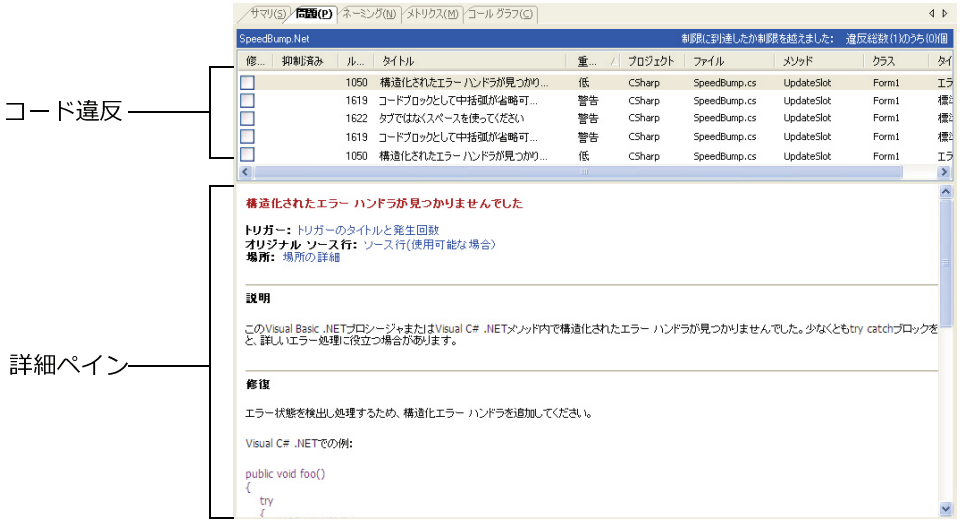


図 3-8 [問題]タブと詳細ペイン

[問題]タブを理解する

以下の表に、[問題]タブに表示される情報を示します。

表 3-3. [問題]タブの内容

カラム	説明
修正済み	コード違反のステータス 修正済みの場合、チェックボックスをオンにする
抑制済み	ルール抑制のステータス 抑制済みまたは空白　抑制されていない場合)
ルール	該当のコード違反に割り当てられた番号
タイトル	ルールのタイトル
重要度	重要度のレベル　高、中、低、警告)
プロジェクト	違反が存在するプロジェクト
ファイル	違反が存在するファイル
メソッド	違反が存在するメソッド
クラス	適用されたルールのクラス
タイプ	ルールのタイプ

詳細ペイン

[問題]タブでコード違反を選択すると、詳細ペインに詳細情報が表示されます（図3-8、75ページ）を参照）。内容は、（システム提供およびユーザー設定の）コード レビュー ルール データベースに格納されたルールから生成されます。以下の表に、詳細ペインに表示される情報を示します。

表3-4. 詳細ペインの内容

見出し	説明
ルール タイトル（赤色で表示）	ルールのタイトル
トリガー	トリガーの名前。オリジナル ソース行にハイパーリンクとして表示される（「トリガーを設定する」、99ページ）を参照）
オリジナル ソース行	ルールが呼び出された原因となるコード行
場所	コード違反の発生元
説明	コード違反の説明
対処方法	問題の修正案
メモ	Microsoft MSDN サポート技術情報の記事への外部リンクなど、追加のコメント

ヒント：各コード違反には、[トリガー]、[オリジナル ソース行]、[場所]への追加のハイパーリンクが含まれる場合があります。

ネーミング違反を表示する

[ネーミング]タブには、コード レビューでレビュー中に発見されたネーミング違反がリストされます。[ネーミング]タブの表示内容は、レビュー前に[全般]オプション ページ（図3-4、67ページ））で選択したネーミング分析のタイプに応じて異なります。各ネーミング アナライザの詳細については、「ネーミング分析を理解する」、92ページ）を参照してください。

[ネーミング]タブには、これらどちらかの結果が表示されますが、両方の結果が同時に表示されることはありません。[なし]を選択していると、コード レビューのあと、[ネーミング]タブは空になります。

ハンガリアンの結果を分析する

図 3-9 に、[全般]オプション ページ（図 3-4 「67ページ」）でハンガリアン ネーミング アナライザを選択した場合の [ネーミング] タブの表示を示します。

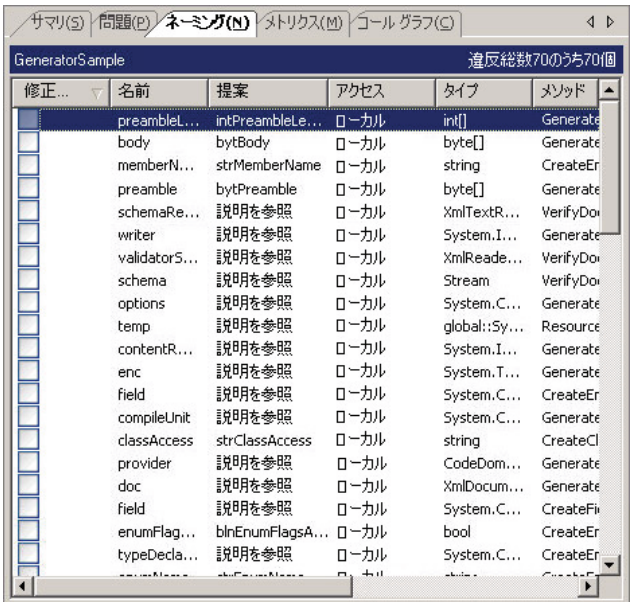


図 3-9 ハンガリアン ネーミング分析の[ネーミング]タブ

ネーミング ガイドラインの結果を分析する

図 3-10 に、[全般]オプション ページ（図 3-4 「67ページ」）でネーミング ガイドライン ネーミング アナライザを選択した場合に 2 つのパネルに表示されるネーミング結果を示します。上のパネルの [ネーミング] タブと、下のパネルのネーミングの詳細ペインを確認します。また、ネーミング ガイドライン分析では、[ネーミング] タブの上の [表示順] リストが有効になります。

図 3-10 ネーミング ガイドライン ネーミング分析の[ネーミング]タブ

以下の表は、選択されたネーミング アナライザに関係なく、[ネーミング]タブに表示される情報を示しています。

表 3-5. [ネーミング]タブの内容

カラム	説明
修正済み	ネーミング違反のステータス 違反を修正した場合に、このチェック ボックスをオンにする。
名前	データ型のユーザー定義名
提案	推奨される名前は、選択したネーミング アナライザに応じて異なります（「ネーミング分析を理解する」 92ページ）を参照）。 • コード レビューでハンガリアン ネーミング規則に基づく名前を提案できない場合は、このカラムに「不明」と表示されます。 • コード レビューでネーミング ガイドラインに基づく名前を提案できない場合は、このカラムにアスタリスクが表示されます。また、ネーミングの詳細ペインに説明が表示されます（図 3-10 78ページ）。
アクセス	現在のソリューション内のアクセスのカテゴリ
タイプ	識別子の種類
メソッド	データ タイプが宣言されているメソッド
クラス	データ タイプが宣言されているクラス
名前空間	データ タイプが宣言されている名前空間
ファイル	データ タイプが宣言されているファイル
プロジェクト	データ タイプが宣言されているプロジェクト

ネーミングの詳細ペインを理解する

[ネーミング ガイドライン]を選択し、[ネーミング ガイドライン]オプション ページ（図 3-5 71ページ））で追加の選択を行った場合、[ネーミング]タブの下に、選択したネーミング違反の詳細を示す詳細ペインが表示されます。

詳細ペインは、ネーミング ガイドライン ネーミング アナライザの場合にのみ使用できます。

表 3-6. ネーミングの詳細ペインの内容

項目	説明
現在の名前	上のパネルで選択された項目に対応します。
スコープ	識別子のスコープを示します。
オリジナル ソース行	上のパネルで選択されたネーミング違反に関連するソース行を表示します。
推奨事項	ネーミング ガイドライン ネーミング アナライザの基準に基づき適切な名前を1つまたは複数提案します （「ネーミング ガイドライン ネーミング アナライザを理解する」 93ページ）を参照)。
説明	この違反が問題としてフラグされた理由を説明します。 コード レビューで名前の修正案を提示できない場合、このペインには説明が表示されます。また、[ネーミング]タブの上のパネルの[提案]カラムには、一連のアスタリスクが表示されます。
メモ	オプションで、「.NET Framework 一般情報リファレンス」の「名前付けのガイドライン」のサポート技術情報へのハイパーリンクが表示されます。

収集されたメトリクスを表示する

[メトリクス]タブ （図 3-11）には、McCabe メトリクスに基づいて、コード複雑度の結果 （複雑度、不良修正の可能性、および理解度）が表示されます （「McCabe メトリクスを理解する」 80ページ）を参照)。

メソッド	ファイル	プロジェクト	複雑度	不良修正確率	理解度	コード行数
QSort	SpeedBump...	CSharp	8	5	Simple to moder...	48
BubbleSortBtn_Click	SpeedBump...	CSharp	5	5	Simple to moder...	17
CSharpBtn_Click	Driver.cs	Driver	1	1	Simple	4
NativeCppBtn_Click	Driver.cs	Driver	1	1	Simple	3
Form1	SpeedBump...	CSharp	1	1	Simple	16
Form1_Load	Driver.cs	Driver	1	1	Simple	2
NativeCppSpeed...	Driver.cs	Driver	1	1	Simple	1
UpdateSlot	SpeedBump...	CSharp	2	1	Simple	5
Dispose	SpeedBump...	CSharp	3	1	Simple	10
Main	Driver.cs	Driver	1	1	Simple	3
Dispose	Driver.cs	Driver	3	1	Simple	10
SwapEm	SpeedBump...	CSharp	2	1	Simple	15
RandomizeBtn_Click	SpeedBump...	CSharp	1	1	Simple	4
QuickSortBtn_Click	SpeedBump...	CSharp	2	1	Simple	9
EndTiming	SpeedBump...	CSharp	1	1	Simple	4
StartTiming	SpeedBump...	CSharp	1	1	Simple	4
Form1	Driver.cs	Driver	1	1	Simple	9
Form1_Load	SpeedBump...	CSharp	1	1	Simple	3
DoRandomize	SpeedBump...	CSharp	3	1	Simple	19
UpdateAll	SpeedBump...	CSharp	2	1	Simple	5
ClearTiming	SpeedBump...	CSharp	1	1	Simple	3

図 3-11 [メトリクス] タブ

[メトリクス] タブには、レビュー前に [全般] オプション ページ（図 3-4 「67ページ」）で [メトリクスの収集] チェック ボックスをオンにした場合のみ、データが表示されます。表 3-7 に、[メトリクス] タブに表示される情報を示します。

表 3-7. [メトリクス] タブの内容

見出し	説明
メソッド	コードの複雑度の問題が発生したメソッドの名前
ファイル	問題が発生したファイルの名前
プロジェクト	問題が発生したプロジェクト
複雑度	特定のコンポーネントに関する複雑度。このメトリックは McCabe 循環複雑度に関連しています。
不良修正確率	既知のバグを修正する際に、コード内に新規のバグが発生する可能性
理解度	コード ロジックの可読性と保守性の程度
コード行	選択したコンポーネント内のコードの合計行。行ごとのカウント数が [サマリ] タブに表示されます。

McCabe メトリクスを理解する

McCabe メトリクスを収集すると、[メトリクス] タブに、複雑度、不良修正の可能性、理解度などのコード複雑度の統計が表示されます。これらのメトリクスは、業界標準の McCabe メトリクスに基づいています。[メトリクス] タブには、コード レビュー ソリューション ツリーで選択したノードに関するすべての項目が表示されます。

複雑度

複雑度（循環的複雑度または McCabe の複雑度とも呼ばれます）は、McCabe メトリクスの一部として確立された業界標準です。複雑度は、プログラムの安定性と信頼性の一般的な尺度です。この尺度として、他のプログラムの複雑度と比較できる単一の序数が提示されます。この序数は、他のソフトウェア メトリクスと調整する場合に頻繁に使用されます。複雑度は、

広く受け入れられているソフトウェア メトリクスの1つとして、言語および言語形式に依存しないように考えられています。複雑度の数値は、コード行数のカウントではなく、プログラム構造の複雑度を測定するより優れた尺度を示します。

複雑度では、プログラム モジュール全体の直線で独立したパスの数を測ることにより、モジュールの条件判断構造の複雑度を測定します。各コンポーネントが個別に分析され、If-Then-Elseステートメントや Select Caseステートメントなど、考えられるすべての条件判断ポイントが計算されます。Select Caseの場合、各ケースが独立した条件判断ポイントです。

McCabeのメトリクスでは、各モジュールの循環的複雑度が以下のように定義されています。

e - n +2

eとnの意味は以下のとおりです。

e：制御フロー図内のエッジ数

n：制御フロー図内のノード数

循環的複雑度では、テストが必要な最小パス数を表します。コードが複雑になるほど、そのコンポーネントのテストには高度な技術が必要になります。

不良修正の可能性

これは、既存のエラーの修正時に誤って新しいエラーが挿入される可能性を表します。不良修正の可能性では、手順を調べて、新しいエラーを引き起こす可能性を計算します。通常、コードが正確に作成されていれば不良修正の可能性は低くなります。不良修正の可能性はMcCabeメトリクスに基づき、理解度と複雑度の結果と相関関係があります。

理解度

理解度は、複雑度と不良修正の可能性と同様に、開発者がコードの解釈や保守を行う際の難度を評価します。理解度では、以下のようにコードを評価します。

表3-8. 理解度メトリック

範囲	理解度
5未満	容易
5以上10以下	容易～中
11以上20以下	中
21以上30以下	中～高
31以上50以下	高
51以上94以下	高～テスト不可能
95以上	テスト不可能

すべてのメトリクスの相関関係

以下の表は、3つのすべてのメトリクスが相互にどのように関連しているかを示しています。

表 3-9. McCabe メトリクスの相関関係

コードの複雑度の範囲	不良修正の可能性の割合	理解度の説明
5未満	1%	容易
5以上10以下	5%	容易～中
11以上20以下	10%	中
21以上30以下	20%	中～高
31以上50以下	30%	高
51以上94以下	40%	高～テスト不可能
95以上	60%	テスト不可能

コール グラフ データを表示する

[コール グラフ]タブには、コード レビュー ソリューション ツリーで選択したメソッドまたはプロパティに対応するインバウンドとアウトバウンドのコール パスの静的なビューが表示されます （図 3-12 82ページ）を参照)。

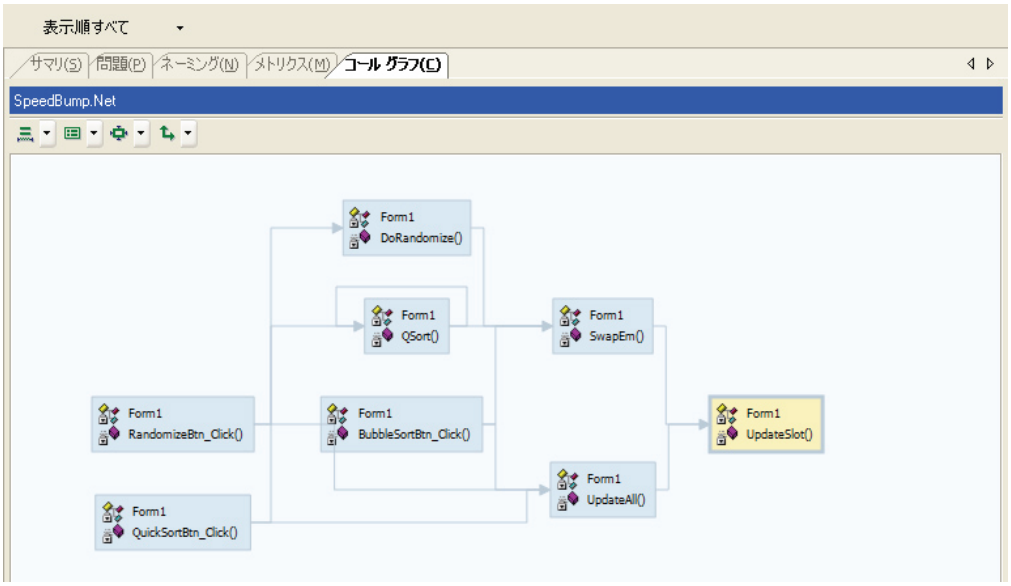


図 3-12 コール グラフを表示した[コール グラフ]タブの例

コール パスは静的に生成されます。つまり、グラフには、プログラム実行中に行われた動的なコールではなく、コール パスにおける潜在的なメソッド コールが表示されます。

[コール グラフ]タブは以下の場合に空になります。

- コード レビューの前に [全般] オプション ページ（図 3-4 67 ページ）で [コール グラフ データの収集] チェック ボックスが選択されていなかった場合。コール グラフ データは

レビュー中に収集されません。コール グラフ分析を実行してコール グラフ データを収集するには、このオプションを選択して別のコード レビューを実行します。

- チェック ボックスをオンにしても、コード レビュー ソリューション ツリー（図 3-2 63ページ）を参照）でメソッドまたはプロパティを選択しなかった場合。データは収集されていますが、コード レビュー ソリューション ツリーでメソッドまたはプロパティ ノードを選択するまでコール グラフは表示されません。

コール グラフの参照を理解する

[コール グラフ**]**タブでは、選択したメソッドまたはプロパティのコール階層を追跡することで、コール パスの潜在的なインバウンド／アウトバウンド コール参照を表します。表示領域は、各メソッドまたはプロパティの潜在的なエントリ ポイントと終了ポイントを示します。コール参照はルート ノードから開始され、ルート ノードを参照して実行されるすべてのコールを対象とします。コール参照は、ルート ノードに制御が戻るか、またはルート ノードからのコールが完了するまで続きます。表示領域には以下のタイプのコール参照が表示されます。

ルート ノード

ルート ノードとは、コール グラフの開始点として選択したメソッドまたはプロパティのことです。その他すべてのノードは、ルート ノードを呼び出すか、ルート ノードによって呼び出されます。ルート ノード（図 3-13）は、表示領域にある他のノードと区別するために、青い境界線で囲まれた黄色の四角形で表示されています。

図 3-13 ルート ノードの例

インバウンド コール

インバウンドとは、ルート ノードを直接的または間接的に呼び出すメソッドやプロパティを指します。ルート ノードと区別するために、インバウンド コール（図 3-14）は薄い青色の四角形のノードとして表示されます。

アウトバウンド コール

アウトバウンドとは、ルート ノードから直接的または間接的に呼び出されるメソッドやプロパティを指します。インバウンド コールと同様に、アウトバウンド コールも（図 3-14）薄い青色の四角形のノードとして表示されます。潜在的なコール パスの方向性を示すために、これらのコールは、ルート以外からの方向を指す一連の矢印で接続されています。

図 3-14 インバウンド／アウトバウンド コール ノードの例

未コールの参照

未コールとは、コードでは定義されているが、アプリケーション コンポーネントを構成するファイル内では1度も参照されなかったメソッドまたはプロパティを指します。[コール グラフ]タブでは、「未コール」というラベルまたは記号 ! のいずれかを使用して、ノード上の未コール メソッドを識別します。

ラベルでの表示

アイコンでの表示

図3-15 未コールを識別する2つの例

再帰コールと循環コールの参照

[コール グラフ]タブでは、選択された実行パス内に存在する、再帰または循環のコール参照のインスタンスをグラフィカルに表示できます。

- 再帰 : 実行パス内で、自らを呼び出すメソッドまたはプロパティ。

  A が B を呼び出す。
  B が B を呼び出す。

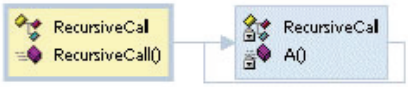


図3-16 再帰コール グラフの例

- 循環 : 実行パス内で前に呼び出されたメソッドまたはプロパティを、間接的に再度呼び出すメソッドまたはプロパティ。

  A が B を呼び出す。
  B が C を呼び出す。
  C が A を呼び返す。

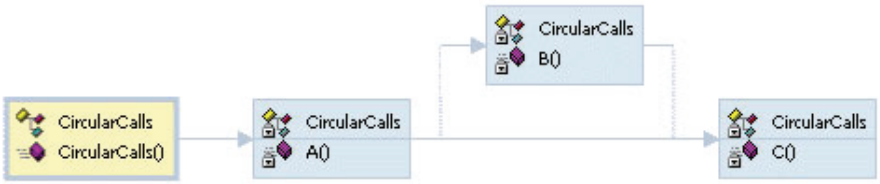


図3-17 循環コール グラフの例

コール グラフの設定オプションを設定する

DevPartner コード レビューには、[コール グラフ]タブに表示されるコール グラフを設定する4つの方法があります。これらのオプションには、[コール グラフ]ツールバーからアクセスするか、[コール グラフ]タブの背景領域を右クリックすることによりアクセスできます。

レベル数

[コール グラフ]タブに表示されるレベル数を選択します。コール グラフには、ルート ノードを呼び出す（インバウンド）またはルート ノードから呼び出される（アウトバウンド）メソッドまたはプロパティが、指定したレベル数で表示されます。1～6のレベル数を選択できます（デフォルトは6）。以下の例ではレベル数2が選択されています。コール グラフの右端のノードに付いているプラス記号（+）は、コール参照のレベルがさらに表示可能であることを示しています。

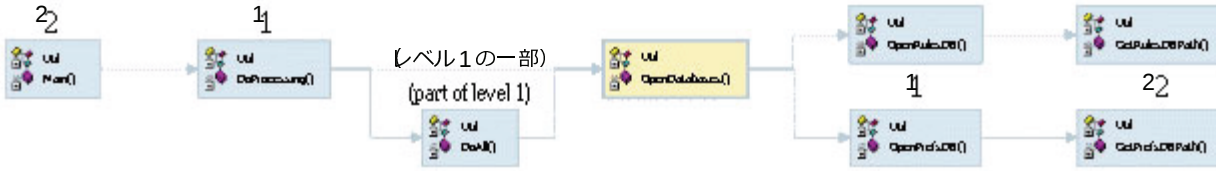


図3-18 レベル数2を設定した場合

ノード スタイル

ユーザーは、[コール グラフ]タブに適用するノード スタイルを選択できます。コール グラフのすべてのノード スタイルで、メソッド名やプロパティ名のほかにクラス名も表示されます。ノード スタイルの中には、クラス、メソッド、またはプロパティのアクセス タイプを示すアイコンが含まれているものもあります。このアクセス タイプには、パブリック、プライベート、内部、およびプロテクトがあります。これらは標準のソリューション ツリーのアイコンです。未コールのメソッドやプロパティを表すその他のアイコンは、コール グラフにだけ表示されます。

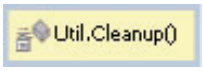
以下の表に、さまざまなノード スタイルの例を示します。

いくつかの例ではルート ノードを示し、それ以外の例では標準ノード（インバウンドまたはアウトバウンド）を使用しています。ノードの区別の詳細については、「コール グラフの参照を理解する」（83ページ）を参照してください。

表3-10. ノード スタイル

ノード スタイル	説明	未コールの表現	例
シングル ラベル	クラス名のあと、ピリオドに続いてメソッド名またはプロパティ名が表示されますが、アイコンは表示されません。	メソッド名またはプロパティ名に - Uncalled という指定が追加されます。	Util.Cleanup() インバウンド／アウトバウンド
ダブル ラベル	上の行にクラス名、下の行にメソッド名またはプロパティ名が表示されますが、アイコンは表示されません。	下の行のメソッド名またはプロパティ名に - Uncalled という指定が追加されます。	Util Cleanup() インバウンド／アウトバウンド

表3-10. ノード スタイル

ノード スタイル	説明	未コールの表現	例
イメージ1つとシングル ラベル	標準のメソッドまたはプロパティの アイコンの他、クラス名、ピリオドに続いてメソッド名またはプロパティ名がすべて同じ行に表示されます。	対応するアイコンには、感嘆符記号 (!) が含まれています。	 ルート
イメージ1つとダブル ラベル	メソッドまたはプロパティ アイコンの他、上の行にクラス名、下の行にメソッド名またはプロパティ名が表示されます。	対応するアイコンには、感嘆符記号 (!) が含まれています。	 ルート
イメージ2つとダブル ラベル	クラスを示す上位のアイコンに続いてクラス名が、そしてメソッドまたはクラスを示す下位のアイコンに続いてその名前が表示されます。	感嘆符記号のアイコン (!) が、データ タイプ アイコンと名前の間に表示されます。	 ルート

拡大／縮小

[コール グラフ]タブでコール グラフの相対サイズを選択します。2つの拡大／縮小オプションを使用できます。

- 空きスペースに合わせる（デフォルト）

  これを選択すると、すべてのノードが表示領域内に収まるようにコール グラフのサイズを調整できます。デフォルトでは、このオプションを選択するとスクロール バーは使用できません。もう1つのオプションを使用してコール グラフを再設定した場合、スクロール バーを含めずに内容のサイズが変更されます。

- フル サイズのパーセント

  これを選択すると、100%、80%、75%、66%、50%のいずれか一定の割合で表示領域の内容を拡大または縮小できます。この選択により、大きな、または複雑なコール シーケンスの一部を拡大表示できます。さらに、内容の再表示の際、選択したメソッドまたはプロパティ（ルート ノード）が表示領域にはっきりと表示されます。スクロール バーも使用できます。

レイアウト

コール グラフ ノードが[コール グラフ]タブでどのように表示されるかを選択します。選択肢は以下のとおりです。

- 水平

  ノードは、表示領域に左から右へ表示されます。選択したノード（ルート ノードとも呼ばれる）を呼び出すメソッドまたはプロパティは、ノードの左に配置されます。選択したノードが呼び出すメソッドまたはプロパティは、ノードの右に分岐します。

- 垂直

  ノードは、表示領域に上から下へ表示されます。選択したルート ノードを呼び出すメソッドまたはプロパティは、ルート ノードの上に配置されます。選択したノードが呼び出すメソッドまたはプロパティは、ノードの下に配置されます。

コマンド ライン インターフェイスを使用する

このコマンド ライン インターフェイスから（**CRBatch.exe** を使用して）バッチ スクリプトを実行して、マネージ プロジェクトを多数含んだ大規模なソリューションをレビューしたり、夜間または自動ビルド プロセスとして大規模なソリューションをレビューしたりできます。コマンド ライン インターフェイスでは、ユーザーの対話をバイパスすることによって、コード レビュー プロセスが合理化されます。

メモ： ソリューション ファイルが読み取り専用に設定されている場合は、Visual Studio からのエラーメッセージが表示され、バッチ レビューが中断されます。

[全般] オプション ページで **[** 常にバッチ ファイルを生成 **]** を選択すると、Visual Studio で実行される次の対話的コード レビューでバッチ ファイルが生成されます。このバッチ ファイルを使用すると、同じソリューションでバッチ レビューを実行できます。

メモ： **/r** オプションを使用する場合は、**[** 常にバッチ ファイルを生成 **]** を無効にするか、バッチ ファイルをバックアップして名前を変更する必要があります。それらの作業を行わない場合は、バッチ ファイルが上書きされます。

コマンド ライン インターフェイスは、レビューの終了後に HTML 形式のサマリ ファイル（**CR_** ソリューション名 **.htm**）をソリューション フォルダに作成します。このファイルは、対話的に生成されたセッション ファイルと内容がまったく同じです。

ソリューションをレビューするバッチ プロシージャを作成すれば、以下のことを実行することができます。

- 生成されたサマリ ファイルとセッション ファイルを別の場所に電子メールで送信する
- サマリ ファイルをローカル イントラネットに保存して、あとで、その場所や外部のインターネット Web サイトから参照する
- **CRExport.exe** を呼び出して XML にデータをエクスポートし、書式設定や表示オプションを追加する （「データを XML にエクスポートする」 91 ページ）を参照）

コード レビューでバッチ レビューが実行できなかった場合は、**CR_** ソリューション名 **.err** というエラー ファイルが作成されます。バッチ ファイルで XML へのエクスポートが失敗した場合は、**CREXPORT_** セッション データベース ファイル名 **.err** という名前のエラー ファイルが作成されます。どちらのエラー ログ ファイルも、セッション ファイルと同じパスに作成されます。

構文とオプション

以下のコマンド ラインの構文とオプションを使用して、コマンド ラインまたはバッチ ファイルからコード レビュー セッションを実行します。

```
CRBatch.exe [/?]/f filename [/v] [/r] [/vs version] [/l XML filename]
```

表3-11. コマンド ライン オプション

オプション	定義
/?	**CRBatch.exe**へのコマンド ライン オプションのリストを表示します。
/f *filename*	レビューで使用する構成ファイルを指定します（必須）。
/vまたは /verbose	/vまたは/verboseエラーをメッセージ ボックスにレポートして、バッチ処理で使用する終了コードを設定するように、コマンド ライン インターフェイスに指示します（オプションですが、構成ファイルを実際にデバッグするときに便利です）。
/rまたは /results	コーディング問題とネーミング違反に関するレビュー結果を調査して、片方または両方のエラー タイプが見つかった場合に特定のエラー コードを返すように、コマンド ラインに指示します（オプション）。
/vs "10.0" /vs "9.0"または /vs "8.0"	バッチ レビューが実行されるVisual Studio .NET Frameworkのバージョンを示します。 10.0 (2010)、9.0 (2008)、または 8.0 (2005)
/l *XML path/file name*	指定されたXMLファイルに示すプロジェクトでのみコード レビューを実行します。 ソリューションのプロジェクトを1つずつ選択し、コード レビューに含めることができます。コード レビュー オプションを使用してプロジェクトを選択します。詳細については、「CRBatchでプロジェクト リスト ファイルを使用する」を参照してください。 ソリューション内でプロジェクトを選択すると、選択されたオブジェクトのリストが入ったXMLファイルが作成されます。/lパラメータを指定してそのXMLファイルを指定し、コード レビューできるようリストのプロジェクトを処理します。/lパラメータは必要ありませんが、使用する場合は、XMLパス/ファイル名を後に続ける必要があります。/lパラメータを使用しながらXMLパス/ファイル名が指定されていない場合やそれらが検出できない場合、エラーが表示（またはログに記録）され、コード レビューは行なわれません。ログに記録されたエラーの詳細については、「エラー ファイルを理解する」を参照してください。

CRBatchでプロジェクト リスト ファイルを使用する

コマンド ラインからコード レビューを実行する場合、ソリューションで処理するいくつかまたはすべてのプロジェクトを選択できます。処理するプロジェクトはプロジェクト リストXMLファイルから抽出されます。

コード レビューがVisual Studioのソリューションに対して最初に実行されると、このファイルが生成されます。このリストは、コード レビュー[全般]オプション ページの[レビューするプロジェクト]テーブルのソリューションに対して選択されたプロジェクトに基づいています。

プロジェクト リスト ファイルが生成されると、名前は**CR_[solutionname].xml**になります（[solution name]はソリューションの名前です）。このファイルは、ソリューション フォルダに格納されています。プロジェクト リストXMLファイルはいったん作成されると、コード レビュー コマンド ライン インターフェイスで使用できます。

コマンド ラインで使用するためプロジェクト リストXMLファイルを作成するのにVisual Studioでコード レビューを実行しない場合、自分のプロジェクト リストXMLファイルを作成できます。以下はプロジェクト リストXMLファイルの構造です。

```
<CRProjectList CRVersion="10.0.0">

    <!-- DevPartner CodeReview Project file-->

    <Projects>

        <Include IncludeType="Include">

            <Project>Project A</Project>

            <Project>Project B</Project>

        </Include>

    </Projects>

</CRProjectList>
```

自分のプロジェクト リストXMLファイルを作成する場合、ファイル名にはCRBatchで使用するXML拡張子がなければなりません。

プロジェクト リストは、包括的または排他的です。デフォルトでは、Visual Studioでコード レビューの実行により作成されたプロジェクト リストXMLファイルは上記のように包括的です。リストを排他的にするには、以下に示すように`Include`指定子の値を`Exclude`に置き換えます。

```
<CRProjectList CRVersion="10.0.0">

    <!-- DevPartner CodeReview Project file-->

    <Projects>

        <Include IncludeType="Exclude">

            <Project>Project C</Project>

            <Project>Project D</Project>

            <Project>Project E</Project>

        </Include>

    </Projects>

</CRProjectList>
```

コマンド ラインからコード レビューを実行している場合にプロジェクト リスト**XML**ファイルを使用する

以下の例では`CRBatch`コマンドで、プロジェクト リストXMLファイルを使用しています。ファイル名には、コマンド構文で`/l`パラメータを続ける必要があります。

```
CRBatch /F "c:¥VS.NET2008¥WindowsApplication2¥CR_WindowsApplication2.CRB" /
vs "9.0" /l "c:¥VS.NET2008¥WindowsApplication2¥CR_WindowsApplication2.xml"
```

/l パラメータが指定されている任意の CRBatch コマンドもバッチ ファイルに使用できます。

エラー ファイルを理解する

コマンド ライン インターフェイスの終了時に、以下のエラー コードが呼び出し元のバッチ プロセスに戻されます。

表 3-12. コマンド ライン エラー コード

エラー番号	メッセージ
0	成功
1	構成ファイルが指定されていません。
2	構成ファイルが存在しません。
3	ソリューション ファイルは指定されませんでした。
4	ソリューション ファイルが存在しません。
5	CRBatch 初期化エラー
6	コマンド ラインの因数が無効です。
7	Visual Studio プロセスの作成に失敗しました。
8	ライセンスのチェックに失敗しました。
9	エラーが原因で Visual Studio が終了しました。
10	Visual Studio のバージョン番号が正しくありません。
11	予期しないエラー
12	コーディング問題が検出されました。
13	ネーミングの違反が検出されました。
14	コーディングの問題とネーミングの違反が検出されました。
70	エラー ファイル （ERR）の作成に失敗しました。

バッチ生成されたレビューでビルド エラーが発生した場合、または、レビュー対象のソリューション内にコンパイル エラーが存在する場合は、セッション ファイルまたはサマリ ファイルが生成されずにバッチ レビューが中断します。エラー ファイルにエラー メッセージが追加されます。

予期しないランタイム エラーの場合は、エラー 11 が返されます。このエラーの詳細 （エラー メッセージとスタック トレース）が、 .ERR ファイルに書き込まれます。

データを XML にエクスポートする

DevPartner コード レビューを使えば、セッション結果データを XML にエクスポートすることができます。これによって、簡単に、結果データをレポート フォーマット、電子メール、内部の Web ページなどに移植することができます。ユーザーは以下の方法でデータを XML にエクスポートできます。

- コード レビュー セッションの実行後に、コード レビューからエクスポートする
- 保存されたセッション ファイルを使用して、コマンド ラインからエクスポートする
- 保存されたセッション ファイルを使用して、自動バッチ処理でエクスポートする

コード レビューのインストール フォルダにある**DPCRExport.xsd**スキーマ ファイルにエクスポート データの内容と XML 形式が記述されています。

DevPartner 内からセッション データをエクスポートする

コード レビューの完了後は、現在のセッション ファイルから XML ファイルにすべてのデータをエクスポートできます。**[** ファイル **]** メニューから **[DevPartner** データのエクスポート **]** を選択して、エクスポート ファイルの名前を指定します。ファイルはデフォルトでソリューションと同じ場所に保存されますが、ソリューション エクスプローラには表示されません。

コード レビュー データをエクスポートするには、コード レビュー セッション ウィンドウにフォーカスを設定する必要があります。

この処理では、コール グラフ データからのインバウンド メソッドを含む、すべてのセッション データが常にエクスポートされます。XML にエクスポートするデータのカテゴリをより詳細に選択するには、コマンド ラインを使用します。

コード レビューでセッション データを XML にエクスポートできなかった場合は、問題の内容を伝えるエラー メッセージが表示されます。

コマンド ラインからセッション データをエクスポートする

DevPartner コード レビューには、XML ファイルにコード レビュー セッションの結果をエクスポートするコマンド ライン ユーティリティ、**CRExport.exe** が含まれています。セッション データをエクスポートするには、必須のコマンド ライン引数を使ってセッション ファイルと出力ファイルを指定する必要があります。例：

```
CRExport.exe /f C:¥MyResults¥WebApp1.DPMDB /e C:¥MyXML¥WebAppData
```

オプションのコマンド ライン引数を使って、セッション データベース ファイルからエクスポートするデータのカテゴリを指定することもできます。

オプションの引数なしで **CRExport.exe** を呼び出した場合は、インバウンド メソッドを含むすべてのセッション データがエクスポートされます。この動作は、**CRExport.exe** に **/a i** 引数を渡した場合および DevPartner 内部からデータ エクスポートを開始した場合と同じです。

エクスポート ユーティリティでエクスポート ファイルが作成できなかった場合は、セッション ファイルと同じパスに **CREXPORT_** セッション データベース ファイル名）**.err** という名前のエラー ログ ファイルが作成されます。

構文とオプション

以下のコマンド ラインの構文とオプションを使用して、コマンド ラインまたはバッチ ファイルからセッション データを XML にエクスポートします。

```
CRExport.exe [/?]/f sessionfile /e xml_exportfile [/a | /a i | /p |
/m | /n | /s | /c | /c i]
```

表3-13. コマンド ライン オプション

オプション	定義
/?	**CRExport.exe**へのコマンド ライン オプションのリストを表示します。
/f *sessionfile*	このエクスポートに使用するセッション データベースを指定します（必須）。
/e *xml_exportfile*	エクスポートされたデータを受信するXMLファイルを指定します（必須）。
/a	コール グラフ データ用のアウトバウンド メソッドを含む、指定されたセッションのすべてのデータをエクスポートします。ただし、インバウンド メソッドはエクスポートされません。
/a i	コール グラフ データ用のインバウンド メソッドとアウトバウンド メソッドを含む、指定されたセッションのすべてのデータをエクスポートします。
/p	指定されたセッションの問題データをエクスポートします。
/m	指定されたセッションのメトリクス データをエクスポートします。
/n	指定されたセッションのネーミング分析データをエクスポートします。
/s	指定されたセッションのコード サイズ データをエクスポートします。
/c	指定されたセッションのコール グラフ データに含まれるアウトバウンド、つまり、呼び出されたメソッドをエクスポートします。
/c i	インバウンド メソッドとアウトバウンド メソッドを含む、指定されたセッションのコール グラフ データをエクスポートします。

バッチ処理からセッション データをエクスポートする

単一のバッチ処理として**CRBatch.exe**と共に**CRExport.exe**を使用してコード レビューを制御し、セッション データをXMLにエクスポートできます。この機能は、バッチ処理を介してすでにコード レビューを実行している以下のような場合に、特に便利です。

- 毎晩のビルド処理の一部として
- 大規模アプリケーションに対して
- 品質制御テストを自動化する目的で

ネーミング分析を理解する

コード レビュー機能には、2種類のネーミング分析機能が組み込まれています。

- ネーミング ガイドライン

  ネーミング分析では、.NET Frameworkがサポートされています。「ネーミング ガイドライン ネーミング アナライザを理解する」（93ページ）を参照してください。

- ハンガリアン

  ハンガリアン ネーミング アナライザは、コード レビューの旧来のネーミング アナライザです。「ハンガリアン ネーミング アナライザを理解する」（95ページ）を参照してください。

また、ネーミング分析全体をバイパスするように、[全般]オプション ページ（図3-4 67ページ）の[使用するネーミング分析]リストから、[なし]を選択することもできます。

ネーミング ガイドライン ネーミング アナライザを理解する

ネーミング ガイドライン ネーミング アナライザは、Visual Studio .NET Frameworkネーミング ガイドラインに基づいて作成されています。これらのネーミング ガイドラインによって、予測と管理が可能な一貫したネーミング方法が、マネージ クラス ライブラリの.NET Frameworkタイプに適用されます。

[全般]オプション ページ（図3-4 67ページ）の[使用するネーミング分析]リストから[ネーミング ガイドライン]を選択し、より正確なレビューが行われるように[ネーミング ガイドライン]オプション ページ（図3-5 71ページ）でさらに追加の選択を行います。

ネーミング ガイドライン ネーミング アナライザでは、以下の要素を検証します。

- パラメータ
- クラス
- 名前空間
- メソッド
- デリゲート
- 列挙
- 構造
- インターフェイス
- 変数

ネーミング アナライザは、大文字の使用、大文字と小文字の区別、略語と頭文字、名前空間の構文、その他の.NET Framework識別子に関連するソース コードのネーミング違反を検索します。

以下のセクションでは、ネーミング ガイドライン ネーミング アナライザが準拠するガイドラインについて説明します。

大文字の使用

ネーミング違反を検出すると、コード レビューでは[ネーミング ガイドライン]オプション ページで選択した大文字使用スタイル（CamelまたはPascal）を使用して、[ネーミング]タブでより適切な名前の提案を試みます。

表3-14. ネーミング ガイドライン ネーミング アナライザで使用される大文字使用スタイル

大文字使用スタイル	連結した最初の単語	連結した後続の単語	名前の提案例
Camel形式	先頭文字を大文字にしない	先頭文字を大文字にする	**r**ed**C**olor
Pascal形式	先頭文字を大文字にする	先頭文字を大文字にする	**R**ed**C**olor

大文字と小文字の区別

DevPartnerコード レビューでは、ソース コードの識別子を区別するために大文字と小文字を区別して使用することは推奨していません。大文字と小文字を区別するプログラミング言語と区別しないプログラミング言語間の相互運用をサポートし、よく似た名前の識別子間での混乱を減らすためにも、大文字と小文字を区別しないことを強く推奨します。開発者は大文

字と小文字を変えただけの名前は避ける必要があります。大文字と小文字を区別するプログラミング言語にも、区別しないプログラミング言語にも有効な名前を使用してください。

略語と頭字語

DevPartner コード レビューでは、一般的に受け入れられている略語と頭字語の使用がサポートされています。DevPartner コード レビューは、以下に基づいて適切なネーミングを決定します。

- 略語または頭字語の文字数
- 識別子名の略語または頭字語の位置

名前空間の構文

DevPartner コード レビューでは、名前空間の .NET Framework ネーミング ルールがサポートされています。名前空間の名前では、先頭に会社名、次にテクノロジ名が続きます。オプションで、末尾に機能名またはデザイン名、あるいはその両方を付与します。構文の例は以下のとおりです。

会社名 . テクノロジ名 [. 機能] [. デザイン]

デフォルトでは、名前空間には Pascal 形式が推奨されます（「Camel 形式または Pascal 形式を選択する」 71 ページ）を参照）。論理的に連結された各単語はピリオド文字 （.） で区切ります。レビューの前に、[ネーミング ガイドライン] オプション ページの **[** 名前空間オプション **]** フィールド （図 3-5 71 ページ）に名前空間の情報を入力します。

その他の .NET Framework 識別子の構文

DevPartner コード レビューでは、ソース コードの .NET Framework 識別子が適切な名前かどうかをチェックします。以下に、コード レビューの検索対象の例をいくつか示します。

- 数字

  識別子名の一部に数字があるかどうかをチェックします。コード レビューでは数字は削除されず、名前は違反としてフラグされます。

- 下線文字

  識別子名での下線文字 （_） のインスタンスが検索されます。下線文字は、ネーミング ガイドライン ネーミング アナライザでは非推奨です。DevPartner コード レビューでは、以下の場合を除いて下線文字を削除します。

  - ◇ 下線文字が先頭文字の場合 （_redColor など）
  - ◇ メソッド名に使用されている場合
  - ◇ 下線文字を削除することによって、別のネーミング違反が発生する場合

- 定数での大文字使用

  DevPartner コード レビューでは、定数の場合、すべて大文字ではなく、Pascal または Camel の大文字使用スタイルに従います （[ネーミング ガイドライン] オプション ページで指定）。たとえば、コード レビューでは以下の HTTP_PORT 定数を変更します。

  ```
  private const int HTTP_PORT = 80
  ```

  - ◇ httpPort （Pascal を適用した場合）
  - ◇ httpPort （Camel を適用した場合）

- **Delegate**

  宣言識別名に1つまたは複数の認識可能な単語と共に単語「delegate」（大文字、小文字は問わない）が含まれている場合、コード レビューでは、削除によって別の違反が発生しないかぎり単語「delegate」を削除します。たとえば、`MyDelegateWord` という名前は、`MyWord` に変更されます。

ハンガリアン ネーミング アナライザを理解する

DevPartner コード レビューでは、ハンガリアン記法ネーミング ルールに基づくハンガリアン ネーミング アナライザが提供されます。

ハンガリアン ネーミングでは、変数名にはその変数の特定のスコープ レベルやデータ型の接頭辞を識別する、特定の文字が含まれます。たとえば、データ型接頭辞の `int` は、整数変数の `Port` などのように整数を意味します。またスコープレベルの接頭辞 `g_` は `g_intPort` のようにグローバルを意味します。

[全般]オプション ページ（図 3-4 「67ページ」）の [使用するネーミング分析] リストから [ハンガリアン] オプションが選択された場合、コード レビュー時にハンガリアン ネーミング アナライザが使用されます。また、DevPartner コード レビューでは、現在選択されているネーム セットが使用されます。コード レビューを開始すると、ネーミング アナライザによってコード内のすべての変数のスコープ レベルの接頭辞とデータ型の接頭辞が評価されます。該当する場合、ネーム セット（「デフォルト」を推奨）に適した提案が行われ、コード レビューのあとに [ネーミング] タブ（図 3-9 「77ページ」）にネーミング結果が表示されます。

ハンガリアン ネーミング アナライザでは、パラメータ名は評価されません。

以下の表に、現在のネーム セットに指定したように、ハンガリアン ネーミング アナライザで評価されるスコープ レベルおよびデータ型の接頭辞の組み合わせ例を示します。

表 3-15. スコープ接頭辞

スコープ	接頭辞
グローバル	g_
メンバー	m_
ローカル	“”

表 3-16. データ型接頭辞

データ型	接頭辞
文字列	str
int	int
int	i
boolean	bool
bool	bln

変数宣言の修飾子によって、変数が存在する境界などのスコープが決まります。たとえば、パブリック ステータスを持つ変数はクラス外にアクセスできるため、コード レビューではこれらの変数はグローバル スコープを持つ変数とみなされます。

デフォルトのネーム セットには、ルール マネージャを使用して編集できるスコープ接頭辞が含まれています。また、ルール マネージャを使用して、ハンガリアン記法に基づいて変数名およびオブジェクト名をカスタマイズすることもできます。

ハンガリアン ネーミングでの提案名の構成

ハンガリアン ネーミング アナライザは、ソース コードに以下に示すような異常が 1 つまたは複数見つかった場合に、適切な提案を行います。

- スコープ接頭辞が間違っている場合や欠落している場合（グローバルに `g_` ではなく `m_` が使われている場合など）
- データ型接頭辞が間違っている場合や欠落している場合（整数タイプに `intShort` ではなく `Short` が使用されている場合など）
- **[** 接頭辞の次の文字が大文字でない場合に警告する **]** チェック ボックス（ルール マネージャの [新規ルール セット] または [ルール セットの編集] ダイアログ ボックス内）をオンにしてハンガリアン ネーム セットに適用しているが、接頭辞の後の変数名の先頭文字が大文字になっていない場合

ネーミング アナライザでは、以下のように組み合わせを行います。スコープ レベル接頭辞 + データ型接頭辞ハンガリアン ネーム セットでスコープレベルの接頭辞が指定されていない場合、推奨される名前はデータ型接頭辞から始まります。

DevPartner コード レビューでは、変数のデータ型を認識できない場合は、以下の理由により、名前は提案されず、**[**ネーミング**]** タブに「不明」と表示されます。

- 現在のハンガリアン ネーム セットにデータ型が存在しない。
- ルール マネージャの[新規ルール セット]または[ルール セットの編集]ダイアログで**[**接頭辞の次の文字が大文字でない場合に警告する **]** チェック ボックスがネーム セットに対して選択されている。

ネーム セットの管理の詳細については、「コード レビュー ルール マネージャを使用する」96 ページ）を参照してください。

コード レビュー ルール マネージャを使用する

DevPartner コード レビューには、Microsoft Visual Studio プログラミング標準に基づく拡張可能なルール データベースが含まれます。ルール データベースはルール マネージャのスタンドアロン アプリケーションに格納され、保守されています。ルール マネージャで、ルール、トリガー、およびルール セットを設定できます。コード レビュー中に、ハンガリアン ネーミング アナライザが使用するハンガリアン ネーム セットも設定できます。ルール マネージャで行われた変更は、コード レビュー ルール データベースに保存されます。これらの変更は、次のコード レビューを設定し、実行するときにすぐに使用できるようになります。

ルール マネージャにアクセスするには、**[** スタート **]** メニューから **[Micro Focus]** > **[DevPartner Studio]** > **[** ユーティリティ **]** > **[** コード レビュー ルール マネージャ **]** を選択します。

ルールを設定する

ルール マネージャを使用して、ルールを作成、編集、および削除します。違反の解決を試みる開発者に詳細情報を提示するために、ルールの説明に HTML リンクを追加することもできます。

ルールを作成する

[新規ルール]ダイアログ ボックスを使用して、新規ルールを作成し、設定します。新しいルールを作成するには、以下の手順を完了します。

1 **[ルール] > [新規ルール]**を選択します。

[新規ルール]ダイアログ ボックスが開き、デフォルトで**[全般]**タブが表示されます。タイトル バーには事前に割り当てられたルール番号が表示されます。ステータス バーには、現在の[所有者]と[最後の編集]の詳細が表示されます (図 3-19 を参照)。

ルールのトリガーと式が作成されるまで、ルールは適用されません。

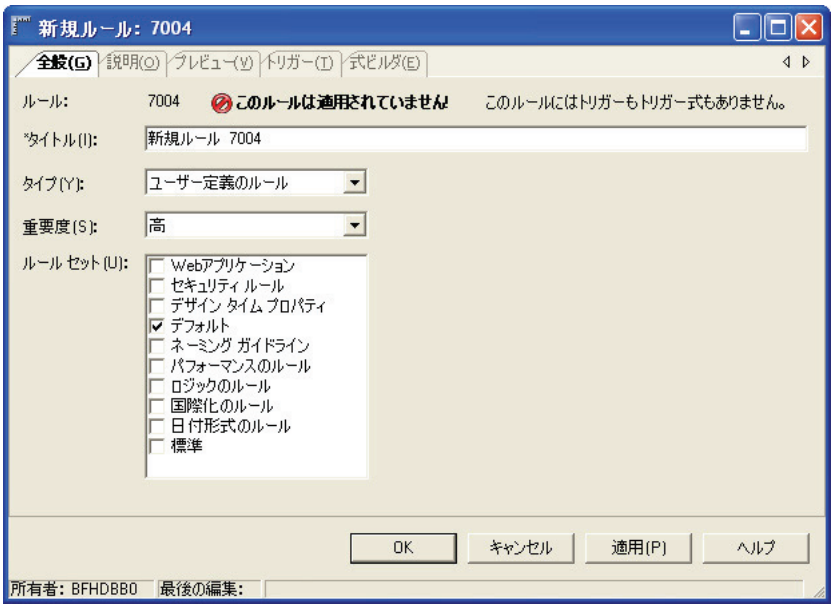


図 3-19 [新規ルール]ダイアログ ボックス

2 各タブを以下の順にクリックして、新しいルールを設定します。

a **[全般]** – ルールの一般的な特性を入力します。

b **[説明]** – ルールの詳細を入力します。

c **[プレビュー]** – 現在のエントリを確認します。

d **[トリガー]** – そのルールに最大 5 つのトリガーを設定します。

e **[式ビルダ]** – 設定した各トリガーに正しい式を作成します。

3 **[説明]**タブをクリックして、ルールの説明を追加します。

[説明]タブを使用して、コーディングの問題解決を支援するために外部リソースへ開発者を誘導する HTML リンクを提示します。入力したリンクは、コード レビュー セッションのあとに、**[問題]**タブの下のパネル (説明ペイン) に表示されます。

4 **[**トリガー **]**タブをクリックして、ルールのトリガーを追加します。

トリガーの作成の詳細については、「トリガーを設定する」（99ページ）を参照してください。

5 **[**式ビルダー **]**タブを選択して、トリガー式をビルドします。

先ほど**[**トリガー **]**タブで設定した各トリガーの式をビルドできます。トリガー式のビルドの詳細については、ルール マネージャのオンライン ヘルプを参照してください。

ルールを編集する

既存のルールのプロパティを編集するには、[ルールの編集]ダイアログ ボックスを使用します。[ルールの編集]ダイアログ ボックスに含まれるフィールドは、[新規ルール]ダイアログ ボックスとすべて同じです（図 3-19 97ページ）を参照）。既存のルールを編集するには、以下の手順を完了します。

1 **[**ルール**]** > **[**ルールの編集**]**を選択します。

[ルールの編集]ダイアログ ボックスが開き、デフォルトで**[**全般**]**タブが表示されます。タイトル バーにはルール番号とタイトルが表示されます。ステータス バーには、現在の[所有者]と[最後の編集]の詳細が表示されます。

2 以下の順序で各タブをクリックして、既存のルールを変更します。

a **[**全般**]** – 既存のルールのプロパティを変更します。

b **[**説明**]** – ルールの詳細を変更します。

c **[**プレビュー**]** – 現在のエントリを確認します。

d **[**トリガー**]** – 既存のトリガーの設定を変更します。

e **[**式ビルダ**]** – 各トリガーの式を変更します。

ルールを削除する

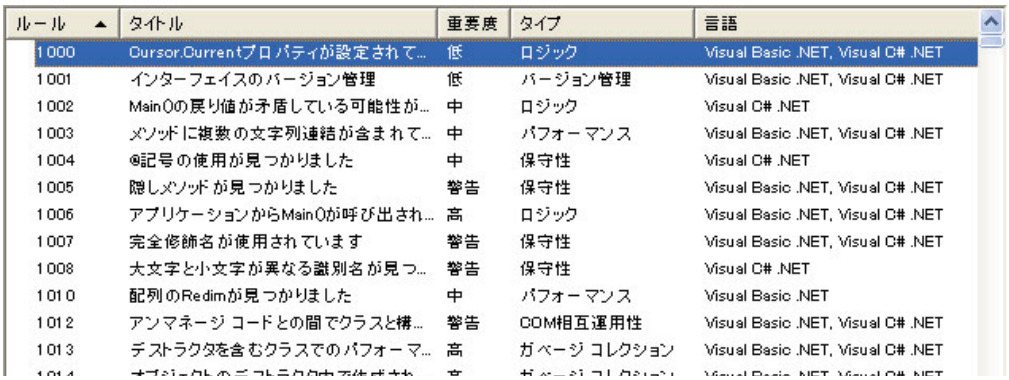
削除できるのは「すべてのルール」にあるユーザー設定のルールのみで、DevPartnerに付属するルールは削除できません。ユーザー定義のルールを「すべてのルール」から削除すると、他のルール セットからもそのルールが自動的に削除されます。

DevPartnerに付属のルールを編集すると、システムにそのルールの所有権がなくなるだけで、ユーザー定義のルールに変更が加えられることはありません。「すべてのルール」からシステム定義のルールを削除することはできません。

ルールを削除するには、以下の手順を完了します。

1 **[**ルール セット**]**リストから**[**すべてのルール**]**を選択します。

「すべてのルール」に含まれているルールがルール リスト ペインに表示されます。



ルール	タイトル	重要度	タイプ	言語
1000	Cursor.Currentプロパティが設定されて...	低	ロジック	Visual Basic .NET, Visual C# .NET
1001	インターフェイスのバージョン管理	低	バージョン管理	Visual Basic .NET, Visual C# .NET
1002	Main()の戻り値が矛盾している可能性が...	中	ロジック	Visual C# .NET
1003	メソッドに複数の文字列連結が含まれて...	中	パフォーマンス	Visual Basic .NET, Visual C# .NET
1004	@記号の使用が見つかりました	中	保守性	Visual C# .NET
1005	隠しメソッドが見つかりました	警告	保守性	Visual Basic .NET, Visual C# .NET
1006	アプリケーションからMain()が呼び出され...	高	ロジック	Visual Basic .NET, Visual C# .NET
1007	完全修飾名が使用されています	警告	保守性	Visual Basic .NET, Visual C# .NET
1008	大文字と小文字が異なる識別名が見つ...	警告	保守性	Visual C# .NET
1010	配列のRedimが見つかりました	中	パフォーマンス	Visual Basic .NET
1012	アンマネージ コードとの間でクラスと構...	警告	COM相互運用性	Visual Basic .NET, Visual C# .NET
1013	デストラクタを含むクラスでのパフォーマ...	高	ガベージ コレクション	Visual Basic .NET, Visual C# .NET

図 3-20 ルール リスト ペイン

2 ルール リスト ペインで、ユーザー定義のルールを 1 つまたは複数選択します。

3 **[**ルール**]** > **[**ルール データベースから選択したルールを削除**]** を選択します。

選択したルールが、デフォルトの「すべてのルール」ルール セットから削除されます。この操作を取り消すことはできません。

[選択したルールをルール データベースから削除**]** は、**[**ルール セット**]** リストから **[**すべてのルール**]** を選択すると **[**ルール**]** メニューでのみ有効になります。

トリガーを設定する

ルールを適用する最大 5 つのトリガーを設定するには、**[**トリガー**]** タブを選択します。

DevPartner に付属するいくつかのルールではマクロを使用しますが、マクロに基づくルールのトリガーを編集または設定することはできません。

設定しているルールにトリガーが関連付けられていない場合は、空の **[**既存のトリガー**]** リスト ボックスのみが表示されます。また、新しいトリガーを設定に追加できるように、**[**追加**]** ボタンが有効になっています。

[既存のトリガー**]** にすでに 1 つまたは複数のトリガーある場合は、トリガーの [タイプ] に該当するその他のフィールドが表示され、必須項目にはアスタリスクが付いています。設定には **[**追加**]** ボタンと **[**削除**]** ボタンを使用できます。

トリガーを追加する

トリガーを追加するには、以下の手順を完了します。

1 **[**追加**]** ボタンをクリックして、新しいトリガーを追加します。

ルール マネージャでは、**[**トリガー名**]** フィールドにデフォルト名の「新規トリガー **n**」が表示されます。たとえば、これが最初のトリガーなら、名前は「新規トリガー**1**」となります。

トリガーを 5 つ設定すると、このペインの **[**追加**]** ボタンが自動的に無効になります。

2 トリガー名を入力または変更します。

左角かっこ [と右角かっこ] は使用しないでください たとえば、[CheckString] など)。角かっこの文字を入力すると、それらのキー入力はルール マネージャで無視されます。角かっこは式ビルダ ペインの **[**式**]** フィールドのトリガー式で複数のトリガーを区切るときに使用します。角かっこを手動で挿入すると、トリガー式が無効になります。

3 [タイプ] リストからトリガー タイプを選択します。

選択するトリガーのタイプによって、設定するトリガーの残りのパラメータが決まります（3-17 を参照）。

表 3-17. トリガー タイプ

タイプ	説明
コード	実際のソース コードの問題を検出します。
Web フォーム ページ	HTML や ASP.NET のタグ構造への準拠を確認します。
デザイン タイム プロパティ	特定の Visual Studio .NET プロパティへのトリガーの適用を検出します。
Web.config	ASP.NET Web.config ファイルの要素への準拠を確認します。

4 選択したトリガー タイプのすべての必須フィールドを設定します。

選択した [タイプ] に応じて、トリガーに異なるパラメータを指定します。特定のトリガー タイプの設定の詳細については、ルール マネージャ オンライン ヘルプを参照してください。

5 [正規表現] テキスト ボックスにルールの正規表現を追加します（「正規表現を使用して新しいルールを作成する」 104 ページ）を参照）。

トリガーを削除する

[既存のトリガー] にリストされたトリガーを削除するには、以下の手順を完了します。

1 トリガーを選択し、[削除] をクリックします。

確認メッセージが表示されます。

2 [はい] をクリックして確定するか、[いいえ] をクリックしてこの操作をキャンセルします。

トリガー式ですでにそのトリガーが使用されている場合は、削除できません。

ルール セットの設定

ルール セットは、コード レビュー セッションで使用できるルールの集合です。DevPartner コード レビューでは、事前に設定されたルール セットを選択できます。また、カスタム ルール セットで作業したり、ルール マネージャを使用してルール セットを作成、編集、または削除したりすることもできます。

ルール セットを作成する

ルール マネージャには、すべてのルールというマスタ ルール セットが付属しています。ただし、プロジェクトに固有の必要条件に合わせて、追加のルール セットを作成することもできます。新しいルール セットを作成するには、以下の手順を完了します。

1 [ファイル] > [新規ルール セット] を選択します。

[新規ルール セット] ダイアログ ボックスが表示されます。

2 [ルール セット名] フィールドにルール セットの名前（最大 30 文字）を入力します。

3 [説明] フィールドに新しいルール セットの短い説明を入力します（オプション）。

4 ダイアログ ボックスの [ハンガリアン ネーム セット] セクションで、[セットの使用] リストからハンガリアン ネーム セットを選択します。

ルール マネージャのネーム セットでは、ハンガリアン ネーミング ルールに従うハンガリアン ネーミング アナライザのみがサポートされています。Visual Studio .NET ネーミング ガイドラインに従うネーミング ガイドライン ネーミング アナライザはサポートされていません。

5 ハンガリアン ネーミング違反を [ネーミング] タブに表示する方法を選択します。

◇ [不明なオブジェクトが検出された場合に警告する] を選択すると、コード レビューで別の名前を提案できないネーミング違反は「不明」と表示されます。

◇ [接頭辞の次の文字が大文字でない場合に警告する] を選択した場合は、コード レビューでネーミング違反が検出されると別の名前が提案されます。

6 **[OK]** をクリックします。

新しいルール セットが検証されます。

7 以下の方法で、ルール セットにルールを設定します。

◇ 新しいルールを作成する（「ルールを作成する」 97ページ）を参照）。

◇ 既存のルール セットを開き、ルールを選択、コピーして新しいルール セットに貼り付ける。

ルール セットを編集する

ルール セットのプロパティを編集するには、以下の手順を完了します。

1 [ルール セット] リストから既存のルール セットを選択します。

2 [ファイル] > [ルール セットのプロパティ] を選択します。

[ルール セットの編集] ダイアログ ボックスが表示されます。[ルール セットの編集] ダイアログ ボックスで使用できるフィールドは、[新規ルール セット] ダイアログ ボックスと同じです。

3 [ルール セット名] フィールドにルール セットの名前（最大 30 文字）を入力します。

4 [説明] フィールドに新しいルール セットの短い説明を入力します（オプション）。

5 ダイアログの [ハンガリアン ネーム セット] セクションに、[セットの使用] リストからハンガリアン ネーム セットを選択します。

ルール マネージャのネーム セットでは、ハンガリアン ネーミング ルールに従うハンガリアン ネーミング アナライザのみがサポートされています。Visual Studio .NET ネーミング ガイドラインに従うネーミング ガイドライン ネーミング アナライザはサポートされていません。

6 ハンガリアン ネーミング違反を [ネーミング] タブに表示する方法を選択します。

◇ [不明なオブジェクトが検出された場合に警告する] を選択すると、コード レビューで別の名前を提案できないネーミング違反は「不明」と表示されます。

◇ [接頭辞の次の文字が大文字でない場合に警告する] を選択した場合は、コード レビューでネーミング違反が検出されると別の名前が提案されます。

7 **[OK]** をクリックします。ルール セットのプロパティに加えた変更が検証されます。

ルール セットを削除する

既存のルール セットを削除するには、以下の手順を完了します。

1 **[**ルール セット**]** リストからルール セットを選択します。ユーザー定義のルール セットは削除できますが、DevPartner に付属するルール セットは削除できません。

2 **[** ファイル **]** > **[** ルール セットの削除 **]** を選択します。[ルール セットの削除] ダイアログ ボックスが表示されます。

3 **[** 削除 **]** をクリックします。この操作を取り消すことはできません。

ハンガリアン ネーム セットの設定

ルール マネージャを使用して、コード レビュー セッション中にハンガリアン ネーミング アナライザで使用されるハンガリアン ネーム セットを作成、編集、複製、または削除します。[ハンガリアン ネーム セット] ダイアログ ボックスにアクセスするには、**[** ファイル **]** > **[** ハンガリアン ネーム セット **]** を選択します。

ハンガリアン ネーム セットを作成する

新しいハンガリアン ネーム セットを作成するには、以下の手順を完了します。

1 **[** 新規作成 **]** をクリックします。[ハンガリアン ネーム セットの新規作成] ダイアログ ボックスが開きます。

2 一番上のフィールドの **[** 無題 **]** をネーム セットの固有の名前に置き換えます。

3 **[** 作成 **]** をクリックします。**[** 作成 **]** をクリックすると、**[** 追加 **]**、**[** 編集 **]**、および **[** 削除 **]** ボタンが有効になります。

4 この新規のネーム セットを適用する適切な言語を選択します。言語を選択すると、ルール マネージャは新しい名前を検証します。

ハンガリアン ネーム セットを編集する

既存のハンガリアン ネーム セットを編集するには、以下の手順を完了します。

1 [ハンガリアン ネーム セット] ダイアログ ボックスでネーム セットを選択します。

2 **[** 編集 **]** をクリックします。[ハンガリアン ネーム セットの編集] ダイアログ ボックスが開きます。

3 目的に合わせて、ネーム セットに関連付けられている言語、変数、およびオブジェクトを編集します。ハンガリアン ネーム セットの名前は編集できません。

ハンガリアン ネーム セットを複製する

ハンガリアン ネーム セットは複製できます。これにより、既存のネーム セットに沿ってテンプレートとして新しいネーム セットを作成できます。ハンガリアン ネーム セットを複製するには、以下の手順を完了します。

1 [ハンガリアン ネーム セット] ダイアログ ボックスでネーム セットを選択します。

2 **[** 複製 **]** をクリックします。[ハンガリアン ネーム セットの複製] ダイアログ ボックスが開きます。

3 一番上のフィールドの「<名前>の複製」を固有の名前に置き換えます。

4 **[**作成**]**をクリックします。

[作成**]**をクリックすると、**[**追加**]**、**[**編集**]**、および**[**削除**]**ボタンが有効になります。ルール マネージャで、ネーム セットが検証されます。

ネーム セットを削除する

ルール セットで現在使用していないユーザー定義のハンガリアンネーム セットのみを削除できます。

ハンガリアン ネーム セットを削除するには、以下の手順を完了します。

1 **[**ファイル**]** > **[**ハンガリアン ネーム セット**]**を選択します。

[ハンガリアン ネーム セット]ダイアログ ボックスがもう一度開きます。

2 削除するネーム セットを選択します。

ハンガリアン ネーム セットに関連付けられている変数とオブジェクトのすべてが、タブ付きペインで強調表示されます。**[**追加**]**、**[**編集**]**、**[**複製**]**ボタンはすべて無効になります。

3 **[**削除**]**をクリックします。

[ハンガリアン ネーム セットの削除]ダイアログ ボックスが開きます。

4 **[OK]**をクリックして、選択したネーム セットを削除します。

[ハンガリアン ネーム セット]ダイアログ ボックスがもう一度開きます。この操作を取り消すことはできません。

5 **[OK]**をクリックします。

ルール リストを操作する

ルール リストに表示されるルールは、以下の2つの方法で操作できます。

ルール リスト ビューをフィルタする

フィルタ ペインは、**[**ルール セット**]**リストの下の、ルール マネージャ ウィンドウの左側に配置され、ルール リスト ペインに表示される内容をフィルタするときに使用します （図 3-20 99ページ）を参照）。

1 **[**ルール セット**]**リストからルール セットを選択します。

[セット内のすべてのルール**]**を選択すると、データベース内のすべてのルールが自動的にリストされます。個々にフィルタ選択を行うには、個々のルール セットを選択します。

2 **[**フィルタ**]**タブをクリックします。

3 **[**フィルタ**]**オプションで、各グループから少なくとも1つ、項目を選択（または）クリアします。

グループのチェック ボックスをオンにするか、[+]をクリックして展開すると、グループ内の個々の項目を選択できます。

各グループから少なくとも1項目を選ぶ必要があります。選択しなければ、処理の必要な領域がマウス ポインタで示されます。グループは以下のとおりです。

- ◇ タイプ – ルールはプログラミング テクノロジに合わせます。
- ◇ 重要度 – 選択肢は、[高]、[中]、[低]、[警告] です。[警告] は、特定のコーディングの問題に注意を促すときに使用します。
- ◇ 言語 – 選択したルール セットに該当する言語のみが表示されます。
- ◇ 所有者 – DevPartner提供のルールは「DevPartner」と示されています。その他のすべてのルールは、そのルールを作成した個人の所有となります。選択したルール セットに該当する所有者のみが表示されます。

4 **[** 適用 **]** をクリックして、現在のビューにフィルタをかけます。

特定ルールを検索する

[検索 **]** タブは、**[** ルール セット **]** リストの下の、ルール マネージャ ウィンドウの左側にあります。これは、1つまたは複数のルールを検索するときに使用します。

1 **[** ルール セット **]** リストから検索するルール セットを選択します。

[セット内のすべてのルール **]** を選択すると、ルール データベースにあるすべてのルールが表示されます。

2 **[** 検索 **]** タブをクリックして、検索オプションを表示します。

3 **[** ルール セットの検索 **]** リストから基準を選択します。**[** 内容 **]** リストから最近使った検索文字列を選択します。

4 **[** 内容 **]** に文字列を入力して、特定の検索条件を定義します。

5 **[** 検索 **]** をクリックします。

続けて検索を実行するには、次の **[** 検索場所 **]** オプションのどちらかを選択します。

- ◆ セット内のすべてのルール – 新しい検索を開始する場合
- ◆ 現在の結果 – 現在の結果内で検索を継続する場合

オプションで上記の検索基準を変更し、もう一度 **[** 検索 **]** をクリックできます。

正規表現を使用して新しいルールを作成する

コード レビューで独自のルールを作成し、それらのルールを使用して多数の疑わしいコーディング プラクティスを特定できます。DevPartner コード レビューのルールに正規表現を活用することによって、テキスト検索の堅牢で汎用的な手段が提供されます。

正規表現は広く使用されており、ドキュメントが充実しているうえ、HTML、Visual Basic、およびVisual C# の構文内のパターンと一致するように作成できます。DevPartner コード レビューでは、Microsoft Visual Studio と同じ正規表現のエンジンを使用しており、同じ構文がサポートされています。

DevPartner コード レビューでは、指定したルールのスコープをコードの特定の部分に制限することによって、ルール内で正規表現をより簡単に使用できるようになります。たとえば、ルールの適用先をファイル全体にしたり、メソッドのみにしたり、または `While` ブロックのみにしたりできます。ルールのスコープを指定できるため、コードのターゲット部分にのみ正規表現を適用することができます。

また、DevPartnerコード レビューでは、コード ブロックからコメントを削除することによって、正規表現の検索を支援することもできます。レビューの実行前にコメントを削除すると、誤検知を減らすことができます。

以下のセクションでは、実際のコード レビュー ルールを例示して、ルールを操作する正規表現について説明します。

コード レビュー ルールの正規表現の記述方法の詳細については、以下のリソースを参照してください。

- Forta, Ben. 『Teach Yourself Regular Expressions in 10 Minutes』Indiana：Sams Publishing、2004
- Friedl, Jeffrey E.F. 『Mastering Regular Expressions』（2nd ed）California：O'Reilly、2002
- Goyvaerts, Jan 『Regex Tutorial, Examples and Reference』 1 Feb. 2006 <http://www.regular-expressions.info>
- Microsoft Corporation.『.NET Frameworkの正規表現』2006.<http://msdn.microsoft.com/library/default.asp?url=/library/hs600312(VS.80).aspx>

90文字を超える行のマッチング

ベスト プラクティスのコーディング標準では、コード行を90文字以内にすることが推奨されています。この標準に準拠するため、コード レビュー ルールでは90文字を超える行を検索します。以下の正規表現によって、行の長さが90文字を超えないことが保証されます。

```
(?-s).{91,}
```

この正規表現ではまず、単一行オプションを**False**に設定します。これにより、式では改行文字（¥n）までのすべての文字（改行は含まない）が単一行として評価されます。この評価では、行の冒頭から改行（¥n）までのコードの各行を個々の別の行として扱います。

次に、このルールには、正規表現の最も基本的な機能である単一文字のマッチングが組み込まれています。このルールでは、メタキャラクタであるピリオド（.）を使用します。このメタキャラクタでは、その行の任意の1文字を検索します。

このルールでは、このピリオド（.）に続いて、繰り返しのマッチング メタキャラクタ{91,}が指定されています。繰り返しのマッチング メタキャラクタでは、マッチングの繰り返し回数が特定の数であるか、または特定の範囲内の数であるかを指定します。このルールでは、任意の単一文字が91回以上一致した場合にのみ、式がTrueになります。このルールでは一致回数が90を超えるかどうかのみを検査するため、2番めの範囲の値は空のままにします。ï 3-18では、基本的な繰り返しマッチングのメタキャラクタを示しています。

表3-18. 繰り返しマッチングのメタキャラクタ

文字	意味
+	直前の文字が1回または複数回あるインスタンスとマッチングします。
*	直前の文字が0回または複数回あるインスタンスとマッチングします。
?	直前の文字が0回または1回あるインスタンスとマッチングします。

表3-18. 繰り返しマッチングのメタキャラクタ

文字	意味
{*n*} {2,6} {*n*,}	指定された回数、先行する文字が出現するインスタンスとマッチングします。nは必要な繰り返し回数を表します。 また、これらの波かっこは、カンマで区切った上限と下限を含めることによって、2から6回などの繰り返し範囲の指定に使用することもできます。 上限を省略すると、上限は設定されずに最小のインスタンス出現回数に対してマッチングが行われます。

スペースの代わりに使用されるタブをマッチングする

ベスト プラクティスのコーディング標準では、タブの代わりにスペースを使用することが推奨されています。タブによって表されるスペースの数はエディタによって異なる場合があります。この相違によって、各エディタでソース コードの表示に差が発生することがあります。ソースの表示に一貫性を持たせるには、スペースを使用する必要があります。以下の正規表現は、メソッドの内部でタブが使用されているかどうかを検索するコード レビュー ルールで使用されます。

```
(?s)¥t.*
```

この正規表現では、単一行オプションを**True**に設定します。これにより、式では、改行文字（¥n）までを含む各行の個々の文字が単一行の一部として評価されます。

次に、このルールでは、メタキャラクタ（¥t）を使用して、タブ文字に対するマッチングを指定します。これ以上表現を変更しない場合、この正規表現ではメソッド内に出現する個々のタブ文字を検出します。インデント行など、1つのメソッド内に複数のタブが使用されているインスタンスでは、そのメソッドの各タブにルールが適用されます。このことは、意図したルール動作とは異なります。

このルールでは、メソッド内で少なくとも1つのタブが使用されている場合、Trueとして評価される必要があります。メソッド内で毎回タブを検出する必要はありません。このようにするには、このルールには、ピリオド（.）メタキャラクタと、そのあとにゼロ回以上のインスタンスを指定する繰り返しマッチングのメタキャラクタを付加する必要があります。ï 3-18（105ページ）では、使用する繰り返しマッチングのメタキャラクタをアスタリスク（*）で表しています。末尾にこれらの2つのメタキャラクタを付加すると、最初のタブ文字を検出した時点で必ずルールがTrueと評価され、以降はメソッド内の個々の後続文字がキャプチャされます。

コードによって**System.Exception**がキャッチされた場合に、インスタンスをマッチングする

`System.Exception`では十分に詳細なレベルでエラーが取得されず、エラー タイプを適切に識別できないため、エラーを処理するためにこの例外をキャッチすることは避けてください。エラー処理のコード ブロックでは、可能なかぎり高い精度でエラーをインターセプトして処理する必要があります。これにより、プログラムがより堅牢になり、クラッシュが発生しにくくなります。以下の正規表現は、コード内で`System.Exception`をキャッチするVisual Basic構文が使用されているインスタンスを検出するように設計された、コード レビュー ルールで使用されます。

```
Catch¥s¥w+¥sAs¥s(System¥.)?Exception
```

この式の最初の部分では、コード内のリテラルの単語 Catch のインスタンスを検索します。このルールでは、より長い単語の最初の一部が Catch であるインスタンスが検出されないように、リテラルのテキストに続いて、空白文字（¥s）が指定されています。

Visual Basic 構文では、単語 Catch のあとに例外オブジェクトの保持に使用される変数名を続けて指定します。この変数のあとには、さらに空白文字とリテラルの単語 As が続きます。

このルールでは、空白文字、および単語「As」があとに続く有効な変数名を検索するために、正規表現の機能が必要となります。メタキャラクタ ¥w は、繰り返しマッチングのメタキャラクタ + とペアで使用され、英数字（大文字または小文字）または下線文字が 1 回または複数回出現するインスタンスを検索します。¥sAs¥s を付加することによって、空白文字と単語「As」があとに続く有効な変数名の検索が完了します。

この正規表現では以下のコードが検出されます。

```
Catch MyExceptionObject As
```

この正規表現では、例外をキャッチするすべてのコードの検索に成功します。ただし、このルールでは、System.Exception をキャッチするコードのみをマッチングする必要があります。さらに正規表現の精度を高める必要があります。

コードが System.Exception をキャッチするインスタンスのみがこの正規表現で検索されるようにするには、System と Exception がピリオドで区切られたリテラルの単語を検索します。ピリオドはメタキャラクタであるため、ルールでは、リテラルのピリオドの前に円記号を指定し、特殊文字というステータスを取り除いてマッチングを指定する必要があります。

これでルールの正規表現の一部分に System¥.Exception が付加されましたが、依然として問題があります。System.Exception をキャッチする場合は、System. がなく、用語 Exception のみを使用する構文が有効です。正規表現の最後の修正点は、System. のマッチングをオプションにすることです。System¥. を丸かっこで囲むことでこの部分を副次式とし、ゼロ個または1個のマッチングを指定するために後ろに?メタキャラクタを続けることができます。

複数の戻り点があるメソッドをマッチングする

ベスト プラクティスのコーディング標準では、メソッドの戻り点を 1 つのみにすることが推奨されています。複数の戻り点があると、コードが難解になります。以下の正規表現は、メソッドに複数の戻り点があるインスタンスを検索するコード レビュー ルールで使用されます。この正規表現を構成するほとんどの部分は前述のルールで使用したものですが、新しい内容を 2 つ説明します。

```
(?s)(¥breturn¥b.*){2,}
```

まず、メソッド全体を対象にするために、(?s) を使用してルールの単一行オプションを True に設定します。メソッドを全体として評価するために、このルールでは、単一行の部分として改行文字（¥n）までを含み、各行の個々の文字を評価する必要があります。

これまでに説明したルールで使用した式の別の部分には、末尾に繰り返しマッチングのメタキャラクタがありました。この式では、メソッド内で 2 箇所以上一致することが必須になるように、{2,} を使用して、直前の副次式（丸かっこで囲まれている）を修飾しています。

この副次式 (¥breturn¥b.*) は、この正規表現の中で最も重要な作業を行う部分です。この式は、ブロック全体を繰り返しマッチングのメタキャラクタで修飾できるように、副次式として記述されています。¥b メタキャラクタは、単語境界です。リテラルのテキスト return を単語境界のメタキャラクタで囲むことによって、正規表現では、より長い単語の一部ではなく単語単体のインスタンスを検索します。

メモ： 前述の例では、常に Catch と完全に一致する単語を持つインスタンスのみを検索するように、リテラルのテキスト Catch のあとに空白文字のメタキャラクタ **¥s** が続きました。このことは、正規表現の柔軟性を表す好例です。前述のルールでも単語境界のメタキャラクタ ¥b を使用することはできましたが、使用しませんでした。

副次式内の最後の .* は任意の文字のゼロ個以上のインスタンスの一致を検索します。これでルールが完成し、ゼロ個以上の文字があとに続く単語 return のインスタンスが 2 個以上あるメソッド全体が検索されます。

変数を定義するとき、変数を強制的に初期化する

簡潔かつ理解しやすいコードを維持するためのベスト プラクティスとして、変数を定義する場合は、常にそれらの変数を初期化することをお勧めします。以下の正規表現は、変数が定義されているが未初期化であるインスタンスを検索するコード レビュー ルールです。

```
(?-s)¥bDim¥b(?!.*=)(?!.*¥bnew¥b)
```

ルールでは、コードの各行を単独で評価する必要があるため、まず単一行オプションを False に設定します。

次に、正規表現は単語 Dim の検索に移ります。完全に一致する単語のみが検討されるように、リテラルのテキスト Dim を単語境界のメタキャラクタ ¥b で囲みます。

副次式によって、前方検索または後方検索の概念を実装できます。正規表現に前方検索または後方検索の機能があることによって、柔軟性が向上します。

前方検索または後方検索は、正規表現の副次式で一致を検索していることを意味します。副次式では、特定のテキスト自体とマッチングしてそのテキストを返すのではなく、単純な文字列一致のように、一致があるかどうかのみを検証します。一致が検出されると、副次式が True と評価され、正規表現の残りの部分が成功するか失敗するかは他の修飾子に従って決まります。一致するテキストが見つかった時点で副次式が True と評価されるため、この種類の前方検索と後方検索は肯定前方参照または肯定後方参照と呼ばれます。

肯定前方参照と肯定後方参照の構文は以下のとおりです。

- 肯定前方参照 (?= 副次式)
- 肯定後方参照 (?<= 副次式)

同様に、否定前方参照と否定後方参照は、ステートメントで指定された副次式と一致しないテキストを検索することで機能します。

否定前方参照と否定後方参照の構文は以下のとおりです。

- 否定前方参照 (?! 副次式)
- 否定後方参照 (?<! 副次式)

このルールの正規表現では、スペースが前に置かれているかに関係なく、Dim キーワードの右側にイコール記号 (=) がない場合を検出する否定前方参照の構造を使用する必要があります。副次式 (?!.*=) では、否定前方参照を処理します。

式の最後の部分（?!.*¥bnew¥b）では、別の否定前方参照を使用して、スペースが前に置かれているどうかに関係なく、Dim キーワードの右側に単語 new が存在しない場合は、True と評価します。

これで単語 Dim のあとにイコール記号 (=) または単語 new が続かないコード行を検出した場合は、常に True と評価される正規表現となり、ルールが完成しました。

1行に複数のステートメントを持つインスタンスをマッチングする

コードの可読性と保守性を高めるために、単一行には1つのステートメントしか配置しないようにする必要があります（ただし、ループ構文は例外）。以下の正規表現は、1行に複数のステートメントを含むインスタンスを検索するコード レビュー ルールです。

```
(?<!for.*);.*;
```

指定した行に複数のステートメントがあるかどうかは、行に複数のセミコロンが含まれているかどうかによって判断するのが最も簡単であるように思われます。事実、このことを検索することは、このルールの正規表現で最も重要です。ただし、セミコロンがforキーワードに関連付けられている可能性も考慮する必要があります。

キーワードforとセミコロンが関連付けられているインスタンスを除外するには、ルールに否定後方参照の構造（?<!for.*）を使用してセミコロンを検出したすべての行で後方検索を行い、単語forがないことを確認します。この正規表現の残りの部分(;.*;)では、セミコロン、任意の数の他の文字、さらに次のセミコロンという並びを検索します。

左波かっこが別の行に配置されているかを確認する

ベスト プラクティスのコーディング標準では、左波かっこは単独で別の行の冒頭に配置して、そのあとにブロックを開始するステートメントを続けることが推奨されています。以下の正規表現は、左波かっこが単独で別の行に配置されていないインスタンスを検索するコード レビュー ルールです。

```
(?m)^¥s*¥w+(?=.*?¥{).*?$
```

この表現には、いくつかの新しい概念が作用しています。ルールではまず、(?m)を使用して複数行オプションをTrueに設定します。この設定により、行の最初（^）と行の最後（$）という他の2つのメタキャラクタの動作が変わります。複数行オプションを有効にすることで、^と$は、検索対象の文字列全体ではなく各行の最初と最後をキャプチャします。

複数行オプションを有効にすると、正規表現では、後ろに1つまたは複数の空白文字（¥s*）と、1つまたは複数の単語文字（¥w+）が続く行頭（^）を検索します。これにより、正規表現によって使用される基準が設定されます。1つまたは複数の単語文字が検出された場合、その行には左波かっこがない必要があります。

肯定前方参照の副次式(?=.*?¥{)が、各行で左波かっこがあとに続く任意の文字を検索します。左波かっこの前の円記号は、メタキャラクタのステータスを削除します。左波かっこがあとに続く文字が行に含まれており、左波かっこが行内に単独で存在しないとルールで判定されると、正規表現の末尾の.*?によって、（$ メタキャラクタによってマッチングされた）行の最後より右側に残っているテキストがキャプチャされます。

ループ本体内部のループ カウンタが変更されていないことを確認する

ループ本体内部のループ カウンタを変更すると、予測不能な結果を引き起こす可能性があり、コードが難解になります。以下の正規表現は、ループ本体内部のループ カウンタが変更されているインスタンスを検索するコード レビュー ルールです。

```
(?s)¥bfor¥b¥s*¥(¥s*¥w+¥s+(?<VARNAME>¥w+).*¥).*¥b¥k<VARNAME>¥b¥s*=
```

ルールを適用するために必要な作業が多いため、この正規表現は非常に長くなっています。ループ カウンタの変数名を特定して保存するには、まず正規表現でforキーワード、左かっこ、およびループ カウンタのタイプをキャプチャする必要があります。

正規表現の前半では、必要な情報を収集しています。for キーワードを含み、その後ろに任意の数の空白文字、左かっこ、さらに空白文字、1 または複数の単語文字、空白文字が続く行を検索し、最後に変数名をキャプチャする副次式を使用します。

副次式（?<VARNAME>¥w+）はループ カウンタ名をキャプチャして変数 VARNAME にその名前を保存します。

この構造では、句読点を含まず数字で始まっていない名前であれば、変数名として使用できます。

ループ カウンタの名前をキャプチャすると、正規表現の前半の最後の部分で残りの文字すべてと右かっこを取得します。その後、以下の構造を使用して、ループ本体でイコール記号があとに続くループ カウンタのインスタンスの検索を開始します。

```
.*¥b¥k<VARNAME>¥b¥s*=
```

式のこの部分がルールの最重要点です。このポイントに到達する前に、正規表現ではループ カウンタの名前を判定し、ループ本体内の検索スコープを特定済みです。式の最後の部分では、値 VARNAME（ループ カウンタ）までのすべての文字をマッチングし、さらにカウンタのあとのイコール記号を検索します。

ループ カウンタのあとにイコール記号が続いているということは、何らかの値を設定しようとしているか、またはループ カウンタを変更しようとしています。ループ本体内でカウンタを変更してはいけないため、ルールは違反を検出したことになります。

Visual Studio Team System/Team Foundation Server にデータを送信する

DevPartner Studio では、Team Explorer クライアントがインストールされており、Team Foundation Server に接続可能な場合に、Microsoft Visual Studio Team System がサポートされます。

DevPartner コード レビューの Visual Studio Team System および Team Foundation Server サポート

Visual Studio 2005 および 2008 については、選択した項目に関するデータをバグタイプの作業項目として Visual Studio Team System に送信します。Visual Studio 2010 では、選択した項目に関するデータを問題、バグ、または不具合タイプの作業項目として Team Foundation Server に送信します。コード レビュー セッション ファイルの以下のタブで作業項目の送信にアクセスします。

- [問題] タブ（「コード違反を表示する」 75 ページ）を参照）
- [ネーミング] タブ（「ネーミング違反を表示する」 76 ページ）を参照）

バグを送信すると、DevPartner によって、作業項目フォームにタブのデータが入力されます。DevPartner Studio と Visual Studio Team System の統合の詳細については、「Visual Studio Team System のサポート」 80 ページ）を参照してください。

第4章

自動コード カバレッジ分析

この章には、2つのセクションが含まれています。1つめのセクションでは、はじめてのユーザーを対象として、カバレッジ分析を実行する簡単な手順について説明します。2 つめのセクションでは、DevPartner Studioのカバレッジ分析機能を詳細に把握するための参照情報を示します。

カバレッジ分析に関するタスクベースの追加情報については、DevPartner Studioのオンラインヘルプを参照してください。

カバレッジ分析とは

DevPartner Studioのカバレッジ分析機能を使用すると、開発者およびテスト エンジニアはアプリケーションのすべてのコードをテストしていることが保証されます。カバレッジ分析を使用してテストを実行すると、テストによってカバーされるコンポーネント、イメージ、メソッド、関数、モジュール、および各コード行のすべてがDevPartnerによって追跡されます。テストが終了すると、DevPartnerは実行されたコードと実行されなかったコードを示す情報を表示します。

DevPartnerはアンマネージ　ネイティブ）C++ アプリケーションだけでなく、Web アプリケーション、ASP.NETアプリケーションなどのマネージ アプリケーションのカバレッジ データを収集できます。

簡単な手順でカバレッジ分析を使用する

以下の準備、設定、実行の手順に従ってDevPartnerを使用して、コード カバレッジを分析できます。

すぐに使い始めるには、影付きのボックス内に示された手順に従います。影付きのボックス内に説明されている内容の詳細については、ボックスの下の追加説明を参照してください。

DevPartner Studioでアプリケーションを分析する場合、昇格されたシステム権限は必要ありません。DevPartnerでのアプリケーションの分析には、アプリケーションの作成とデバッグに使用するシステム権限で十分です。

準備：分析対象を検討する

コード カバレッジを使用する前に、分析対象を検討します。

以降の手順では、以下の事項を前提としています。

- Visual Studio のサポートされているリリースで作業しています。
- 単一プロセスのマネージ アプリケーションをテストします。
- アプリケーションのビルドと実行が可能です。
- ソリューションには、スタートアップ プロジェクトが含まれています。

DevPartner カバレッジ分析でサポートされているすべてのプロジェクト タイプのリストは、「DevPartner Studio サポートされるプロジェクト タイプ」（169 ページ）を参照してください。

アプリケーションを分析する際には、カバレッジ セッションを開始する前に収集する対象データを決定します。セッションを開始する前に、いくつかの手順を実行する必要がある場合があります。たとえば、以下の場合には設定が必要です。

- カバレッジ分析から除外するモジュールがある場合
- 分析対象にアンマネージ モジュールがある場合
- リモート サーバー上で実行されるコードを含める場合

この手順では、アプリケーション内のすべてのローカル マネージ コードが分析されます。

設定：プロパティとオプション

カバレッジ分析に含めるコードを決定したら、集中的に目的のデータの収集を行うようにいくつかのプロパティとオプションを設定できます。

この手順では、DevPartner のデフォルトのプロパティとオプションを使用できます。追加の設定は必要ありません。

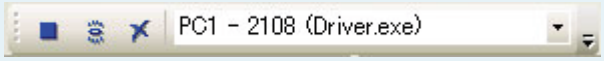
ソリューション プロパティとプロジェクト プロパティを使用して、分析セッション データにアプリケーション外で実行される .NET アセンブリと COM の情報を含めるかどうかを選択できます。DevPartner オプションを使用すると、表示オプションの変更、アプリケーションの一部の分析からの除外、またはセッション コントロール ファイルを作成することによるデータ収集の管理を行うことができます。詳細については、「プロパティとオプションを設定する」（117 ページ）を参照してください。

実行：カバレッジ データを収集する

分析対象を検討し、適切なプロパティとオプションを設定すると、カバレッジ データの収集の準備が整います。

1. Visual Studioで、アプリケーションに関連付けられたソリューションを開きます。
2. **[DevPartner]** > **[カバレッジ分析を選択して開始]**を選択して、カバレッジ分析セッションを開始します。

   セッションの実行中は、セッション コントロール ツールバーのオプションがアクティブになります。

   DevPartnerセッション コントロールを使用すると、カバレッジ分析の対象をアプリケーションの任意のフェーズに集中させることができます。セッション コントロールを使用してデータ収集を停止し、現在収集されているデータのスナップショットを取得して記録を続行するか、または収集済みデータのうちスナップショットにまだ保存されていないデータをクリアすることができます。
3. 分析するコードを実行します。
4. **[スナップショット]**アイコンをクリックします  セッション ウィンドウにフォーカスを移動する必要がある場合は、2回クリックします）。スナップショットを取ると、DevPartnerによって、収集されたデータを含むセッション ファイルと呼ばれるファイルが作成され、セッション ファイルのデータが表示されます。
5. アプリケーションに戻って、テストを引き続き実行します。
6. テストの実行が終了したら、アプリケーションを終了します。Visual Studioに最終的なセッション ファイルが表示されます。

メモ： マネージ アプリケーションのデータ収集を試行しているときにセキュリティ例外のメッセージが表示される場合は、120ページを参照してセキュリティ ポリシーの変更について確認してください。

DevPartnerエラー検出機能と組み合わせてカバレッジを分析できます。テストによってカバーされたコードの範囲を把握すると、エラー検出データを総合的に評価できます。エラー検出とカバレッジ分析の両方を使用したセッションの実行の詳細については、「DevPartnerエラー検出との統合」（133ページ）を参照してください。

データを分析する

スナップショットを取るか、アプリケーションを終了すると、図4-1（114ページ）に示すように、Visual Studioにセッション ファイルが表示されます。以下に、セッション ウィンドウの構成要素を示します。

- フィルタ ペイン。アプリケーションのソース ファイルとイメージをリストし、それぞれでカバーされる行をファイル内の合計行に対する比率として表示します。
- セッション データ ペイン。3つのタブとフィルタ ペインで選択した項目のデータを表示する2つのカバレッジ メーターを含みます。

- カバレッジ メーター。セッション データ ペインのタブの上に表示され、フィルタ ペインで選択した項目の行と関数がカバーされた比率を示します。

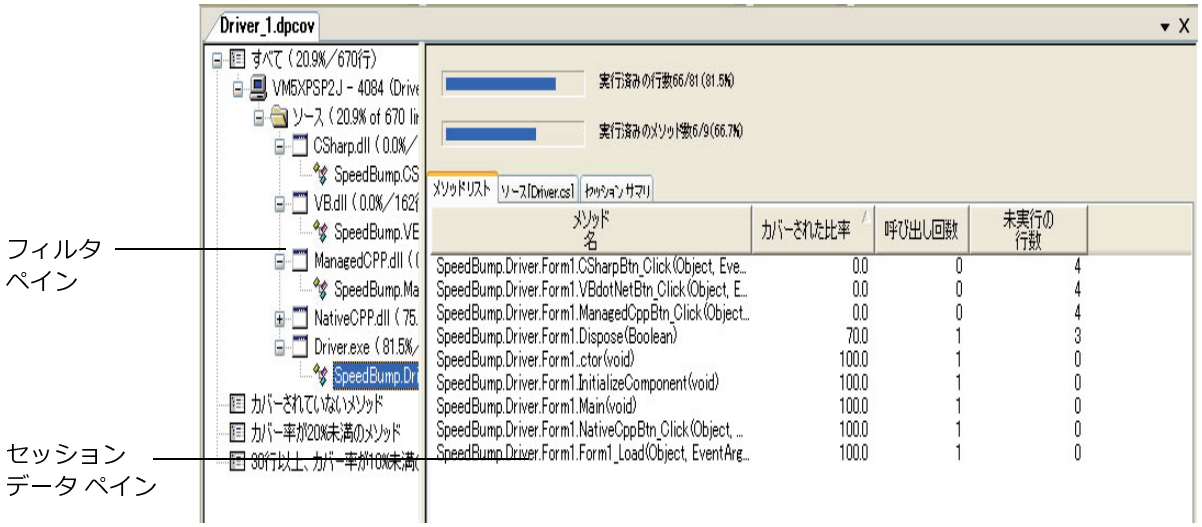


図 4-1 カバレッジ分析セッション ウィンドウ

フィルタ ペインとセッション データ ペインを使用する

フィルタ ペインでは、アプリケーションのファイルとイメージをリストするだけでなく、ユーザーにとって最も重要なデータを集中的に分析するために使用できるフィルタのセットが用意されています。

データの評価を開始するには、フィルタを使用して、表示するデータ量を絞り込むことから開始します。次に、[メソッド リスト]を検証して、テストでのカバー範囲が最も少なかったメソッドを確認します。

1. フィルタ ペインで、[カバー率が **20%** 未満のメソッド]フィルタをクリックします。これで表示されるデータを絞り込み、実行された範囲が最も少なかったメソッドを中心に確認できます。
2. [メソッド リスト]タブのデータを検証して、各メソッドがテストによって適切にカバーされた比率を確認します。

   アプリケーションにテストによるカバーが不適切だった面がある場合は、アプリケーションの機能のカバー率が上がるようにテストを修正できます。

ソース コードを表示する

[ソース]タブには、フィルタ ペインで選択した項目のソース コードが表示されます。[ソース]タブを使用すると、テスト カバレージを上げる必要がある機能の特定に役立ちます。

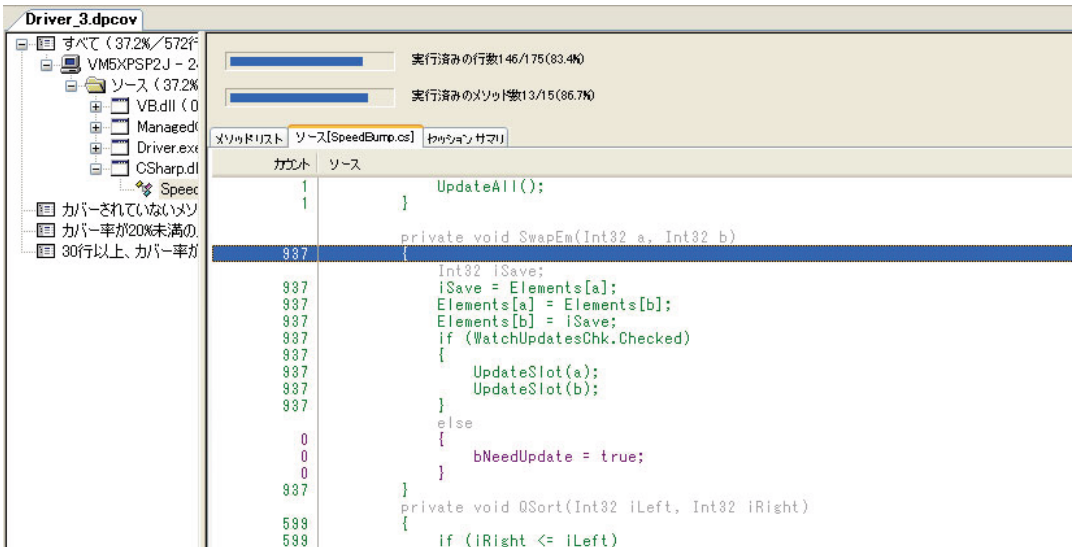


図 4-2 [ソース コード] タブ

[メソッド リスト] にあるメソッドをダブルクリックして、ソース ファイル内の特定のメソッドのコードを表示できます。

3 [メソッド リスト] タブで、[カバーされた比率 [%]] カラムの値が低いメソッドをダブルクリックします。図 4-2 に示すように、該当のメソッドのソース コードが [ソース] タブに表示されます。

[ソース] タブはコードの各行のカバレッジ データを示します。DevPartner では、実行された行（デフォルトで緑色）、実行されなかった行（デフォルトで紫色）、コメントなどの実行できない行（デフォルトで灰色）が強調表示されます。

[カウント] カラムには、行が実行された回数が表示されます。

メモ： DevPartner では、マネージ アプリケーションのソース コード データを表示するには、プログラム データベース ファイル（PDB）の情報が必要です。

[ソース] タブでは行を右クリックして、実行されなかった前後の行に移動できるコンテキスト メニューを表示し、表示するカラムを選択したり別のソース ファイルを表示するように選択したりできます。

セッションのサマリ データを表示する

[セッション サマリ] タブには、カバレッジ分析セッションの概要が表示されます。

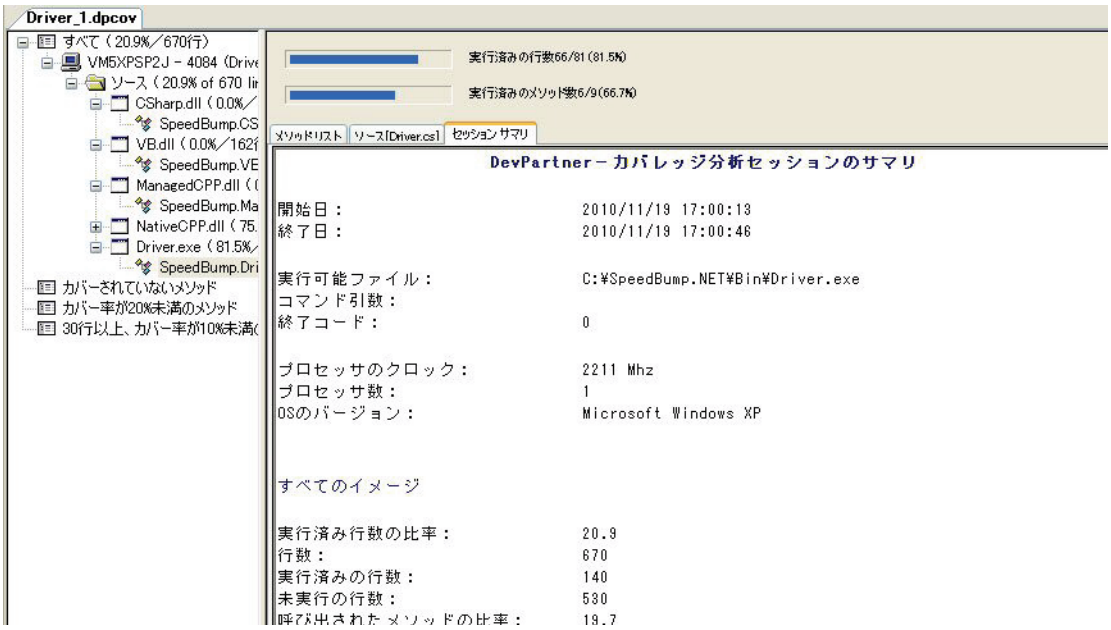


図4-3 [セッション サマリ]タブ

4 **[セッション サマリ]タブをクリックします。**

[セッション サマリ]には、セッションの日時、プロセッサ速度、オペレーティング システムなどのセッションに関するコンテキスト情報が含まれます。この情報は、特に他のユーザーが作成した古いセッション ファイルを参照する場合に役立ちます。

フィルタ ペインおよび[メソッド リスト]タブからのカバレッジ データもサマリに含まれており、分析されたファイルとメソッドの両方のデータが表示されています。

5 タブ内でスクロールして、セッション サマリ データを参照します。

セッション ファイルを保存する

カバレッジ データの確認後、セッション ファイルを保存できます。セッション ファイルを2つ以上作成した場合は、複数のセッション ファイルのカバレッジ データをマージできます。

1 Visual Studioのセッション ファイル ウィンドウを閉じて、セッション ファイルを保存します。メッセージが表示される場合は、デフォルトのファイル名と場所を受け入れます。デフォルトでは、ファイルはプロジェクトの出力フォルダに保存されます。

2 該当のソリューションに複数のカバレッジ セッション ファイルがある場合は、ソリューション プロパティの**[マージ]**設定に応じてファイルをマージするようにユーザーにメッセージが表示されるか、または自動的にマージが行われます。**[セッション ファイルを自動的にマージ]**プロパティの詳細については、「ソリューション プロパティ」 117ページ）を参照してください。

DevPartnerでは、セッション ファイルはアクティブなソリューションの一部として保存されます。これらのファイルは、ソリューション エクスプローラのDevPartnerStudio仮想フォルダに表示されます。カバレッジ セッション ファイルには拡張子 **.dpcov** が付きます。

デフォルトでは、セッション ファイルはプロジェクトの出力フォルダに保存されます。ファイル名は、デフォルト フォルダに配置されているファイル名に自動的に番号を追加する形式（たとえば、**MyApp.dpcov**、**MyApp1.dpcov**）に設定されます。セッション ファイルをデフォルト フォルダ以外の場所に保存した場合は、ユーザーがファイル名と番号を管理する必要があります。

出力フォルダがないプロジェクトの場合（Visual Studio 2005 Web サイト プロジェクトなどの場合）、ファイルはプロジェクト フォルダに物理的に保存されます。

コマンド ラインで生成されたセッション ファイルは、プロジェクトのソリューションに自動的には追加されません。外部で生成されたセッション ファイルは、Visual Studio に開いたソリューションに手動で追加できます。

これで、この章の準備、設定、実行のセクションは終了です。これで、カバレッジ分析セッション実行のメカニズムの基礎が理解できました。追加情報については、引き続きこの章の残りの部分を参照してください。タスクベースの情報については、**DevPartner** のオンライン ヘルプを参照してください。

プロパティとオプションを設定する

多くの場合、カバレッジ分析セッションを開始する前に、特定の種類の情報を含めたり除外したりしてデータ収集を微調整すると便利です。ソリューション プロパティ、プロジェクト プロパティ、および DevPartner オプションを使用して、分析セッションをより目的に適したものにします。

ソリューション プロパティ

ソリューション レベルで使用できるカバレッジ プロパティを表示するには、ソリューション エクスプローラでソリューションを選択し、[F4] を押して [プロパティ] ウィンドウを表示します。

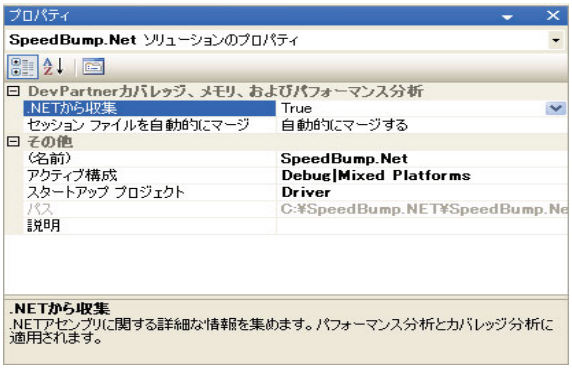


図 4-4 ソリューション プロパティ

以下のソリューション プロパティはカバレッジ分析に影響を与えます。

- セッション ファイルを自動的にマージ – カバレッジ分析セッションのマージ動作（「セッション データをマージする」（128 ページ）で説明）を制御します。
- **.NET** から収集 – マネージ コードのアプリケーションにのみ表示されます。.NET アセンブリの情報を収集しない場合は、このプロパティを [False] に設定します。

このプロパティは、カバレッジ分析とパフォーマンス分析のセッションにのみ影響します。メモリ分析とパフォーマンス エキスパートでは、この値が [False] に設定されていても、常にマネージ アプリケーションからデータが収集されます。

DevPartner for Visual C++ Boundschecker Suite では、**[.NET から収集]** プロパティは使用できません。

- スタートアップ プロジェクト – ソリューションには、スタートアップ プロジェクトを含める必要があります。ソリューションに複数のスタートアップ プロジェクトが含まれている場合は、セッションで使用するスタートアップ プロジェクトを選択するように求めるプロンプトが表示されます。

プロジェクト プロパティ

プロジェクト レベルのプロパティを参照するには、ソリューション エクスプローラでプロジェクトを選択して、ソリューション内のプロジェクトに設定可能なプロパティを参照します。

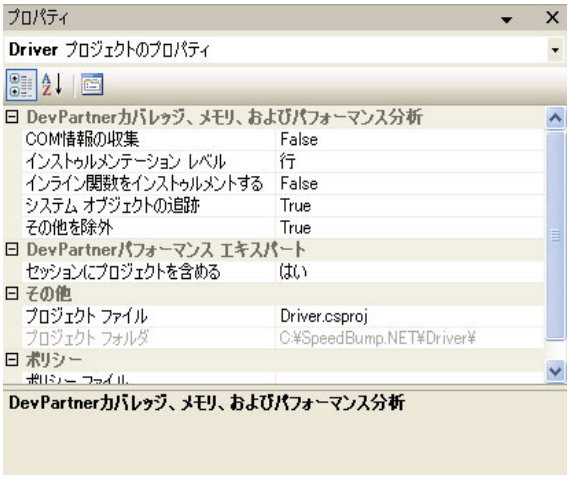


図 4-5 プロジェクト プロパティ

以下のプロジェクト プロパティはカバレッジ分析に影響を与えます。

- **COM 情報の収集** – DevPartner は、DLL エクスポートおよび COM インターフェイスに基づいてメソッド レベルのデータを収集します。アプリケーションの外部で実行される COM の情報を収集しない場合は、[False] を選択します。
- インライン関数をインストゥルメントする – 常にマネージ アプリケーションのインライン関数のカバレッジ データが収集されます。
- アンマネージ コードの場合は、このプロパティを **[True]** に設定してインライン関数をインストゥルメントします。インライン最適化が有効な場合は、デフォルトでインライン関数はインストゥルメントされません。

すべてのプロパティは、明示的に変更しないかぎり保持されます。

オプション

カバレッジ分析セッションに対する DevPartner のオプション設定を確認するには、**[DevPartner] > [オプション] > [分析]** を選択します。

- **[表示]** オプションを使用すると、データ表示時の精度、目盛り、および単位を設定できます。

- **[** 除外 **]** オプションを使用すると、データ収集から 1 つまたは複数のイメージを除外できます。イメージの除外の詳細については、「イメージを除外する」を参照してください。
- **[** セッション コントロール ファイル **]** オプションを使用すると、アプリケーションやモジュールの実行時に DevPartner によって収集されるデータを制御するためのルールとアクションのセットを作成できます。セッション コントロール ファイルの詳細については、「分析セッションの制御」（301 ページ）を参照してください。

[環境 **]** > **[** フォントと色 **]** などのその他の Visual Studio オプションも、DevPartner の機能に影響を与えます。

イメージを除外する

カバレッジ分析でアプリケーションを実行すると、DevPartner はすべてのソースとシステム イメージのデータを収集します。ただし、[除外] を使用することにより、1 つまたは複数のイメージを分析から除外できます。

分析オプションを表示した状態（**[DevPartner]** > **[** オプション **]** > **[** 分析 **]**）で、**[** 除外 - カバレッジ **]** を選択します。

ページ上部にある **[** 表示 **]** リストで、以下のいずれかを選択します。

- グローバルな除外
- 現在のフォルダでのローカルな除外
- 実行可能フォルダでのローカルな除外

[現在のディレクトリ内のローカル除外**]**オプションと**[**実行可能ディレクトリ内のローカル除外 **]** オプションは、ソリューションが開かれており、実行可能フォルダが現在の作業フォルダと異なるときにのみ使用できます。

[挿入 **]** をクリックして、イメージを除外リストに追加します。名前を入力するか、除外するイメージを参照します。除外できるファイル タイプは、**`.exe`**、**`.dll`**、**`.ocx`**、.netmodule です。**[** ファイルの種類 **]** リストを使用して、表示するファイルの種類を制限します。

.NET モジュール （**`netmodule`**）を選択した場合は、モジュールのアンマネージ 部分だけが除外されます。

除外リストからイメージを削除するには、項目を選択して **[** 削除 **]** をクリックします。

他の場所に除外リスト （**`nmexclud.txt`**）のコピーを保存する場合は、**[** 名前を付けて保存 **]** をクリックします。グローバルな除外は、DevPartner のインストール フォルダの **`¥Analysis`** サブフォルダ内の **`nmexclud.txt`** に保存されます。ローカルな除外は、現在の作業フォルダまたはアプリケーション実行可能フォルダにあるアプリケーション用の **`nmexclud.txt`** に保存されます。

除外操作は、[ネイティブ C/C++ インストゥルメンテーション] でコンパイルされたファイルには適用されません。たとえば、インストゥルメントされたアンマネージ C/C++ イメージを除外した場合でも、そのファイルの情報は引き続き収集されます。ただし、システム コール情報は収集されません。データ収集からアンマネージ C/C++ イメージを除外する場合は、そのイメージをインストゥルメントしないでください。

インストゥルメンテーションについて

マネージ アプリケーションを実行する場合、DevPartnerは、コンパイラがバイト コードをロードしてから、各アセンブリのバイト コードにフックを挿入します（このプロセスがインストゥルメントと呼ばれています）。このコードには、アプリケーションの実行時にカバレッジ データの収集に使用されるインストラクションが含まれています。DevPartnerのインストゥルメンテーションでは、ディスク上の実際のファイルは変更されません。実行時にファイルのメモリ内の表現が変更されます。

実行時にインストゥルメントされるマネージ コードとは異なり、アンマネージC/C++コードはコンパイル時にインストゥルメントする必要があります。アンマネージ コードをインストゥルメントする場合、DevPartnerによってソース コードに直接フックが挿入されます。DevPartnerには、使用されるインストゥルメンテーションのタイプ、およびソリューション内のインストゥルメンテーションから除外するプロジェクトを指定できるインストゥルメンテーション マネージャが用意されています（インストゥルメンテーション マネージャの詳細については、「アンマネージ コードのデータを収集する」（121ページ）を参照）。アンマネージ プロジェクトをリビルドすると、フックが挿入されます。フックを削除するには、[DevPartner]メニューの[ネイティブC/C++インストゥルメンテーション]オプションの選択を解除してインストゥルメンテーションをオフにしてから、プロジェクトをリビルドします。

さまざまなタイプのアプリケーションのデータを収集する

このセクションでは、DevPartnerカバレッジ分析を使用して異なるタイプのアプリケーションからデータを収集する方法を説明します。

DevPartnerでは、Visual Studioのすべてのマネージ コードの言語とアンマネージC/C++がサポートされています。また、DevPartnerでは、Internet Explorer（IE）またはInternet Information Services（IIS）を使用しているとき、JScriptおよびVBScript Webアプリケーションのカバレッジ データを収集することもできます。

Visual Studioの各バージョンでサポートされているすべての言語とプロジェクト タイプのリストは、「DevPartner Studio サポートされるプロジェクト タイプ」（169ページ）を参照してください。

マネージ コードのデータを収集する

Visual Studioで開発されたアプリケーションの多くは、C#アプリケーション、Visual Basicアプリケーション、マネージC++アプリケーションなどのマネージ アプリケーションです。

マネージ アプリケーションのデータを収集する場合、セキュリティ ポリシーでDevPartnerによるコードのインストゥルメンテーションが禁止されていると、セキュリティ例外メッセージが表示されます。デフォルトでは、アセンブリをプロファイルするには`SkipVerification`権限が必要です。コードが実行されるポリシーの権限セットからこの権限を削除したり、この権限を無効にするような命令型のセキュリティ宣言をアセンブリに追加した場合、アセンブリをプロファイルできなくなります。

この状態を修正するには、以下の2つの方法のいずれかを使用して、確実にプロファイリングを実行できるようにします。

- 以下のグローバル環境変数を設定して、アプリケーションのプロファイルを再試行します。

  ```
  NM_NO_FAST_INSTR=1
  ```

この方法を使用すると、問題を回避することができますが、パフォーマンスが若干低下します。

- .NET Framework構成ツールのMMCスナップインを使用してアセンブリのポリシーを変更するか、またはアセンブリ内のすべての強制セキュリティ宣言を一時的に削除します。

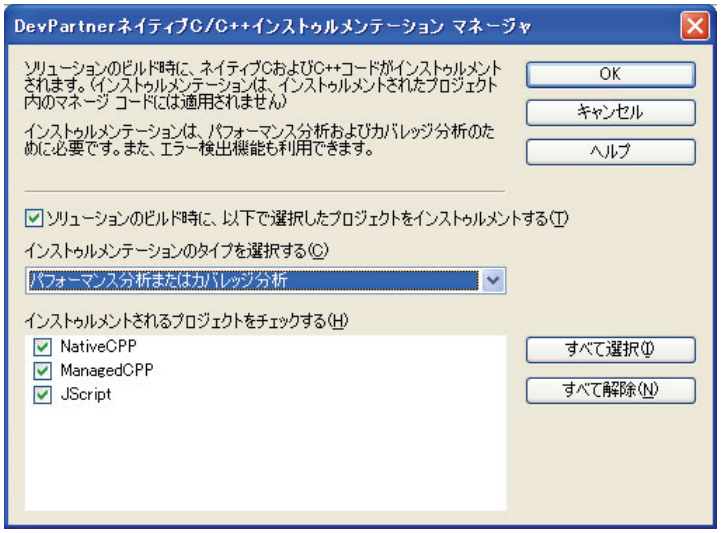
Visual Studioのセキュリティ ポリシーの詳細については、Visual Studioオンライン ヘルプの『.NET Framework Developers Guide』を参照してください。

アンマネージ コードのデータを収集する

カバレッジをプロファイルするためにアンマネージC++アプリケーションをネイティブC/C++インストゥルメンテーションを使用してビルドすると、DevPartnerはコンパイラと連携して、アプリケーション イメージにインストラクションを追加し、実行時にカバレッジ データを収集します。

アンマネージ コードをインストゥルメントするには：

1. データ収集の対象にするアンマネージ C/C++ プロジェクトが含まれるソリューションを開き、**[DevPartner]>[**ネイティブ **C/C++** インストゥルメンテーション マネージャ **]** を選択します。



2. **[**ソリューションのビルド時に、以下で選択したプロジェクトをインストゥルメントする**]** チェック ボックスをオンにして、インストゥルメンテーションのタイプを選択します。選択するインストゥルメンテーションのタイプは、その後実行する分析のタイプと一致している必要があります。

3. インストゥルメントするプロジェクトを選択します。デフォルトで、DevPartnerでは、ソリューション内のすべてのアンマネージ コードがインストゥルメントされます。除外するモジュールの選択を解除します。

4. **[OK]** をクリックして、ソリューションをリビルドします。選択したアンマネージC/C++プロジェクトがインストゥルメントされます。**[**カバレッジ分析を選択して開始**]** を選択して、分析セッションを開始します。

   選択したプロジェクトはソリューションと共にインストゥルメンテーション マネージャに保存されます。一度インストゥルメンテーション マネージャを使用してインストゥルメンテーションを構成すると、[DevPartner] メニューの **[**ネイティブ **C/C++** インストゥル

メンテーション]オプションまたはDevPartnerツールバーの[ネイティブC/C++インストゥルメンテーション]ボタンを使用してインストゥルメンテーションのオンとオフを切り替えできます。インストゥルメンテーション マネージャは、設定を変更する場合にのみ使用してください。

あとでアプリケーションからインストゥルメンテーションを削除するには、[DevPartner]メニューの[ネイティブC/C++インストゥルメンテーション]オプションの選択を解除します。次回カバレッジ分析セッションを開始するとき、またはソリューションをリビルドするときは、Visual Studioによってインストゥルメンテーションなしでソリューションがリビルドされます。

メモ： ご使用のアプリケーションによってアンマネージVisual Studioコンポーネントが呼び出される場合は、Visual Studioでのカバレッジ分析のDevPartnerインストゥルメンテーションでそれらのコンポーネントをコンパイルする必要があります。詳細については、Visual StudioのDevPartner Studioオンライン ヘルプを参照してください。

混合モードのC++ファイル

アンマネージ（ネイティブ）C++で`/clr`オプションを指定すると、アプリケーションをマネージ コードとしてコンパイルできます。その場合は、ネイティブ コードとしてコンパイルする部分を`#pragma`でマークできます。コンパイラは`#pragma`セクションで定義されたメソッドのネイティブ コードを生成します。DevPartnerでは混合モードのC++ファイルはサポートされていません。マネージ セクションとアンマネージ（ネイティブ）セクションの両方が混在するC++ファイルを含むプログラムをプロファイルすると、マネージ コード部分のみのカバレッジ データが収集されて、`#pragma`で囲まれた部分のネイティブ コードのカバレッジ データは収集されません。アンマネージC++コードのデータを収集するには、「アンマネージ コードのデータを収集する」（121ページ）に説明されているように、別のファイルにアンマネージ コードを配置してインストゥルメントします。

複数プロセスのデータを収集する

アプリケーションは複数のプロセスを実行できます。たとえば、ASP.NETアプリケーションをプロファイルすると、ブラウザ プロセス（`iexplore`）、IISプロセス（`inetinfo`）、およびASPワーカー プロセス（`aspnet_wp`または`w3wp`）が実行されていることがわかります。

カバレッジ分析対象として複数プロセス アプリケーションを実行した場合、DevPartnerセッション コントロール ツールバーに、プロセス選択リスト内のアクティブなプロセスが表示されます。

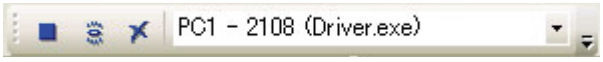


図4-6 プロセス選択リストが表示されたセッション コントロール ツールバー

プロセス選択リストを使用して、データ収集の対象を絞ります。スナップショットを取ると、DevPartnerによって、プロセス選択リストで選択されたプロセスのデータが含まれているセッション ファイルが作成されます。

リモート システムのデータを収集する

DevPartnerを使用して、リモート システムを実行しているアプリケーション コンポーネントのカバレッジ データ収集を有効にできます。たとえば、クライアント/サーバー アプリケーションのクライアント部分とサーバー部分両方のカバレッジ データを収集することもできます。DevPartnerを使用すると、クライアント アプリケーションを実行しているときに、クライアント プロセスとサーバー プロセスのカバレッジ データを収集できます。

クライアント システムとリモート システムから同時にデータを収集するには、クライアント システムに DevPartner を、リモート システムに DevPartner と DevPartner リモート サーバー ライセンスをインストールします。リモート サーバー ライセンスについては、『DevPartner インストール ガイド』および『Distributed License Management ライセンス ガイド』を参照してください。

ターミナル サービス接続経由で接続されるサーバーでは、DevPartner リモート サーバー ライセンスは必要ありません。ターミナル サービスの詳細については、「ターミナル サービスとリモート デスクトップを使用する」（121 ページ）を参照してください。

リモート システムで関連するプロジェクトを選択し、DevPartner プロパティを参照して、クライアント システムで設定したオプションと一致することを確認します。オプションの変更後、IIS などのサーバー プロセスが再起動されます。この再起動は、変更を有効にするために必要です。

アンマネージ C++ アプリケーションを分析する場合は、インストゥルメンテーションを指定します。アプリケーションからアンマネージ C++ コンポーネントを呼び出している場合にそれらのコンポーネントのデータを収集するには、それらのコンポーネントをインストゥルメントする必要があります。「アンマネージ コードのデータを収集する」（121 ページ）を参照してください。

データを関連付ける

IE をブラウザとして使用し、IIS を Web サーバーとして使用している場合、または COM を使用してプロセス内コールを行う場合、クライアント /サーバーの関係はプロセスの間で自動的に認識されます。DCOM オブジェクトのメソッド間、HTTP クライアントとサーバー（IE と IIS）の関係などを保持しておくために、それらのセッションのデータが関連付けられます。次に、そのデータとクライアント セッション データが 1 つのセッション ファイルに結合されます。

関連付けられたセッション ファイルには、アプリケーションのクライアント部分とサーバー部分両方のカバレッジ データが含まれます。関連付けられたセッション ファイルは、**appname_CO.dpcov** のようにファイル名に **_co** が付加されて、他のセッション ファイルと同様に Visual Studio に表示されます。

COM ベースの関係、または IE と IIS 間でクライアント /サーバーの関係がない場合は、**[DevPartner] > [**関連付け**] > [**カバレッジ ファイル**]** を使用して、さまざまなセッション ファイルからのデータを結合できます。`NMCORRELATE` コマンド ライン ユーティリティを使用して手作業でデータを結合することもできます。

.NET Web アプリケーションのデータを収集する

Web フォーム、XML Web サービス、ASP.NET アプリケーションを開発する場合、DevPartner を使用してアプリケーションのクライアント部分とサーバー部分の両方のカバレッジ データを収集できます。ローカル コンピュータまたはリモート サーバーで稼動している IIS と ASP.NET のデータを収集するように、DevPartner を設定できます。

Web アプリケーションがアンマネージ（ネイティブ）C++ コンポーネントを呼び出す場合、Visual Studio の DevPartner コマンドを使用してそれらをインストゥルメントする必要があります。アプリケーションに呼び出されるネイティブ C++ コンポーネントのデータを収集するには、「アンマネージ コードのデータを収集する」（121 ページ）に説明されているように、[ネイティブ C/C++ インストゥルメンテーション] でオブジェクトをインストゥルメントしてリビルドする必要があります。カバレッジ分析のインストゥルメント。DevPartner は、1 つのセッションでは 1 つの分析タイプのデータしか収集しません。

メモ： DevPartnerセッション ファイルは、現在のソリューションと共に保存されます。Visual Studioでプロジェクトを開くのではなく、IISから直接Webプロジェクトを開いた場合は、異なるソリューション ファイルが使用される可能性があります。あるソリューションで作成されたDevPartnerセッション ファイルは、別のソリューションでは表示されません。

前提条件

DevPartnerカバレッジ分析でASP.NETアプリケーションのプロファイル実行を成功させるためには、以下の2つの条件が満たされる必要があります。

- プロジェクトに **`web.config`** ファイルが含まれている。
- **`web.config`**ファイルに、debug属性をtrueに設定したcompilation要素が含まれている。例：

```
<compilation debug=" true" />
```

対象アプリケーションが呼び出すプロセス内部またはプロセス外部のコンポーネントのデータも収集できます。

デバッグなしで **ASP.NET** アプリケーションを分析

最適な結果を得るには、カバレッジ分析をデバッグなしで実行します。

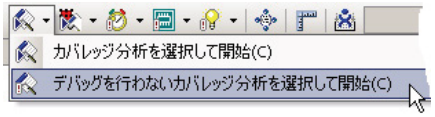


図4-7 デバッグなしで開始するオプション

一度に1つのスクリプト デバッガのみがアクティブになることができます。デバッグありのオプションを指定してWebアプリケーションをデバッグすると、Visual StudioとDevPartnerの両方でスクリプト デバッガのロードが試みられます。その場合、スクリプト デバッガがIEにアタッチできないというメッセージが表示されます。ただし、エラー メッセージが表示されても、セッションは中断されずに続行されます。

エラー メッセージを回避するには、`iexplore`でスクリプトのデバッグを無効にするか、デバッグなしでカバレッジ分析を実行します。

予期せず [プロジェクトにファイルを追加] ダイアログ ボックスが表示される、または予期せずセッション ファイルが保存される

一定の条件のもとでは、ASP.NETアプリケーションを終了したあとで [プロジェクトにファイルを追加] ダイアログ ボックスが予期せず表示されたり、DevPartnerでセッション ファイルを自動的に保存するように設定してある場合に予期せずセッション ファイルが保存されたりする場合があります。

DevPartnerでは、ASP.NETアプリケーションに対してカバレッジ分析を実行すると、IEのデータが一次プロファイル プロセスとして収集されます。また、ASP.NETワーカー プロセス（w3wpまたはaspnet_wp）などの2次プロセスに関するデータも保存されます。一次プロセスが停止すると、DevPartnerではデータの収集が停止されて、クライアント ワーカー プロセスのデータ（IE）とサーバー ワーカー プロセスのデータ（IISとASP.NET）の両方を含む、最終的な相関セッション ファイルが生成されます。セッション コントロール ツールバーでプロセスを選択すると、サーバー プロセスだけのスナップショットを取ることもできます。

ほとんどの場合、クライアント プロセスとサーバー プロセスは、ユーザーのアクションによって終了されます。ただし、ASP.NET ワーカー プロセスは、プロファイル中に自動的にシャットダウンされることもあります。この自動シャットダウンは、以下に示すいずれかの方法でプロセスが実行されているシステム上で、**machine.config** ファイルの「processModel 属性」セクションを編集した場合に発生する可能性があります。

- requestLimit 属性または requestQueueLimit 属性の値を、「Infinite」からセッション中にプロセスをシャットダウンするのに十分低い値に変更した。
- timeout 属性または idleTimeout 属性の値を、Infinite からセッション中にプロセスをシャットダウンするのに十分低い値に変更した。
- memoryLimit 属性の値を、セッション中にプロセスがリサイクルするのに十分低い比率に変更した。

プロセスがシャットダウンすると、DevPartner は最後のスナップショットを取って、セッション ファイルを生成します。DevPartner は、これらのセッション ファイルを以下のいずれかの方法で処理します。

- ASP.NET ワーカー プロセスがセッション コントロール ツールバーで選択されている場合、DevPartner は Visual Studio でセッション ファイルを開き、そのファイルをソリューションに追加します。この動作は、生成および終了される ASP.NET ワーカー プロセスのインスタンスごとに繰り返されます。
- ASP.NET ワーカー プロセスが選択されていない場合、セッション ファイルはキャッシュされます。IE クライアント プロセスが停止するか、IE プロセスのスナップショットが取られると、DevPartner は、IE のセッション ファイルを作成するほか、IE、IIS、およびその時点までに実行および終了されたASP.NETワーカー プロセスのすべてのインスタンスに関するデータを含む相関セッション ファイルを作成します。

分析セッションが終了すると、DevPartner は [プロジェクトにファイルを追加] ダイアログ ボックスを表示したままにするか、実行および終了された ASP.NET ワーカー プロセスのインスタンスに関するセッション ファイルを自動的に保存します。

ASP.NET ワーカー プロセスが頻繁に終了したために余分なセッション ファイルが生成されないようにするには、**machine.config** ファイルを編集して、プロセスの早過ぎる終了を避けるのに十分な高い値に、適切な属性を設定します。

メモ： **machine.config** ファイルを編集する前に、必ずバックアップ コピーを作成してください。

従来の Web スクリプト アプリケーションのデータを収集する

DevPartnerカバレッジ分析を有効にして従来のWebスクリプト アプリケーションを実行すると、HTML ファイル、JScript および VBScript ソース ファイルのデータが収集されます。スクリプト言語が、COM オブジェクトなどのプロセス内部またはプロセス外部のコンポーネントを呼び出す場合、これらのデータも収集できます。

マネージ .NET 言語の場合と同様に、実行時にスクリプト言語のインストゥルメンテーションが行われます。ただし、COM オブジェクトなど監視するアンマネージ コード コンポーネントはインストゥルメントする必要があります。

以下の手順は、従来の Web スクリプト アプリケーション固有の手順です。Visual Studio で開発した Web フォーム、XML Web サービス、および ASP.NET アプリケーションのデータを収集するには、他の任意のマネージ アプリケーションを実行する場合と同様に該当のアプリケーションを実行します。

従来のWebスクリプト アプリケーションのデータを収集するには、**[**スタート**]** > **[**すべてのプログラム**]** > **[Micro Focus]** > **[DevPartner Studio]** > **[**ユーティリティ **]** > **[Web**スクリプトのカバレッジ分析**]**を選択します。

DevPartnerカバレッジ分析がロードされた状態でIEが開きます。IEの他に、データ収集を制御するために使用できるセッション コントロール ツールバーが表示されます。

DevPartnerが有効なIEのインスタンスで、カバレッジ データを収集するHTMLページまたはWebアプリケーションを開き、アプリケーションを実行します。必要に応じて、アプリケーションの実行時にセッション コントロール ツールバーを使用してデータ収集の対象を絞ることも可能です。

IEを終了するか、セッション コントロールを使用している場合は**[**停止**]**アクションを実行します。[セッションを保存]ダイアログ ボックスが表示されて、セッション ファイルが自動的に保存されます。

Webサービス要件

DevPartnerカバレッジ分析でWebサービスを検出させるためには、サービスが以下の要件を少なくとも1つは満たす必要があります。

- Webサービスが、`System.Web.Services.WebService`ベース クラスから派生していること。
- Webサービスに`WebService`属性が含まれること。

DevPartnerカバレッジ分析でWebメソッドを検出させるためには、メソッドに`WebMethod`属性が含まれている必要があります。

NMSourceから一時ファイルを削除する

IEやIISでスクリプトをカバレッジ分析している間に、スクリプト ソースの一時コピーを保存するための**`NMSource`** フォルダが作成されます。このソースは、セッション データの分析時にセッション ウィンドウの[ソース]タブに表示されます。

このソースがいつ必要になるかわからないため、**`NMSource`**フォルダのファイルは削除されません。このフォルダは短期間でサイズが大きくなることがあります。特に、IISでサーバー プログラムを分析する場合には注意が必要です。

`NMSource`フォルダ内のソース ファイルを定期的に参照して、アクティブではなくなったプロジェクトに関するファイルをすべて削除する必要があります。**`NMSource`**は、**`¥Program files¥Internet Explorer`**フォルダにあります。

データ収集のために**IIS**を設定する

リモート サーバーで実行されているIIS/ASP.NETアプリケーションのカバレッジ データを収集するには、後述する設定オプションを設定してください。

メモ : IISがリモート サーバー上で実行されている場合は、リモート システム上にDevPartnerとリモート サーバー ライセンスをインストールして、リモート システムで後述のオプションを設定する必要があります。

スクリプトのデバッグ

以下のオプションは、IIS マネージャの [規定の Web サイトのプロパティ] で、または特定のアプリケーションについては [Web アプリケーションのプロパティ] で設定できます。以下のオプションは IIS 5.0 または 6.0 に適用します。

[ホーム ディレクトリ] タブまたは [ディレクトリ] タブで、**[** 構成 **]** をクリックします。**[** アプリケーションのデバッグ **]** タブで、**[** デバッグのフラッグ **]** を以下のように設定します。

- ASP のサーバー側のスクリプトのデバッグを有効にする
- ASP のクライアント側のスクリプトのデバッグを有効にする

ホスト プロセスの設定

Web アプリケーションが `dllhost` プロセスで実行される場合、[アプリケーション保護] オプションを変更して、DevPartner でのカバレッジ分析データの収集を有効にします。これらのオプションは IIS マネージャの [規定の Web サイトのプロパティ] で、また特定のアプリケーションについては [Web アプリケーションのプロパティ] で設定できます。以下のオプションは IIS 5.0 または 6.0 に適用します。

[ホーム ディレクトリ] または [ディレクトリ] タブの [アプリケーションの設定] セクションで、[アプリケーション保護] を以下のいずれかに設定します。

- 低 (IIS プロセス) : アプリケーションは `inetinfo` プロセスで実行されます。データ収集を有効にすると IIS が再起動され、アプリケーションを実行すると、このプロセスからデータが収集されます。
- 高　分離プロセス) : アプリケーションは、`dllhost` の別のインスタンスとして実行されます。DevPartner により新しいプロセスが認識され、アプリケーションを実行するとデータが収集されます。

データの収集が終了したら、IIS を再起動して、プロセスから DevPartner データ収集を削除します。

Internet Explorer をカバレッジ分析用に設定する

IE からカバレッジ分析データを収集するには、**[** ツール **]** > **[** インターネット オプション **]** を選択します。**[** 詳細設定 **]** タブで、**[** スクリプトのデバッグを使用しない　**(Internet Explorer)** **]** と **[** スクリプトのデバッグを使用しない　その他 **)** **]** をオフに設定します。

サービスのデータを収集する

サービスに対してカバレッジ分析セッションを実行するには、**`DPAnalysis.exe`** を使用します。**`DPAnalysis.exe`** を使用すると、コマンド ラインや XML 構成ファイルからセッションを直接実行できます。

COM および **COM+** アプリケーションからデータを収集する

DevPartner を使用して、COM または DCOM コンポーネントを呼び出すアプリケーションのデータを収集できます。

アンマネージ COM と .NET オブジェクト (COM+) とを混合して使用するアプリケーションをプロファイルする場合は、アプリケーションの .NET 部分の行レベルのデータが収集されます。アンマネージ コード コンポーネントの行レベルのデータが、DevPartner のネイティブ C/C++ インストゥルメンテーションを使用してインストゥルメントされている場合は、この

データが収集されます。カバレッジ データ収集のためにアンマネージ COM オブジェクトの行レベルのデータを最初にインストゥルメントした場合は、それも収集できます。Visual Studio でカバレッジ分析のインストゥルメンテーションを使用してプロジェクトをビルドすることにより、これを実行できます。

C++ オブジェクト、またはインストゥルメントされていないアンマネージ コード コンポーネントをプロファイルする場合は、COM インターフェイスと DLL エクスポートに基づいて、メソッドレベルのデータのみが収集されます。

セッション データをマージする

DevPartner を使用してアプリケーションをテストする場合、すべてのアプリケーション コードが 1 つのセッションでテストされる可能性は低いです。複数のセッションで収集したカバレッジ データを集めて全体のカバレッジ統計を分析できることが重要です。カバレッジ データを蓄積するために、セッション ファイルをマージします。マージとは、複数のセッションのデータを単一のファイルに蓄積する処理のことです。

マージされたセッション データを含むファイルはマージ ファイル （**dpmrg**）と呼ばれます。DevPartner では、単一のプロジェクトに多数のマージ ファイルを関連付けできます。DevPartner では、セッション ファイルはアクティブなソリューションの一部として保存されます。これらのファイルは、ソリューション エクスプローラの DevPartner Studio 仮想フォルダに表示されます。

IE の実行から作成された相関セッション ファイルまたは Web スクリプト セッション ファイルはマージできません。IIS からのサーバー側セッション ファイルはマージできます。

マージ ファイルを作成するには、**[DevPartner] > [** カバレッジ ファイルのマージ **]** を選択して、新しいマージ ファイルを作成するか、または既存のマージ ファイルにデータを追加します。また、「マージ設定」（131 ページ）に説明されているように、マージ ファイルを自動的に作成することもできます。

セッション データをマージすると、DevPartner では以下の処理が実行されます。

- セッション ファイルまたはマージ ファイルでロードされたすべてのイメージおよびメソッドの記録が保存されます。
- カバーされた比率の値が比較され、データの上位集合が返されます。たとえば、30% のメソッドがカバーされたセッションと 20% のメソッドがカバーされたセッションをマージしても、50% のカバレッジには到達しません。実行された関数の一部は、両方のセッションで重複しているからです。
- 最も新しいイメージを実行したセッション ファイルやマージ ファイルからのデータを使用して、メソッドとイメージが新規、変更済み、削除済みのどの状態かを判定します。DevPartner では、イメージのタイム スタンプに基づいて、最も新しいイメージが判定されます。
- 各ソースとイメージの変動率を計算します。変動率は、セッション間のコードで変更されたメソッドの比率を表します。これで、コードの安定性を確認できます。
- マージに関連するファイル、マージをいつ実行したか、およびマージ実行者についての情報を保存します。

マージ データを確認する

DevPartnerでは、マージ データ ウィンドウにマージ データが表示されます。マージ データ ウィンドウには、フィルタとマージ データ ペインがあります。マージ データ ペインには、[メソッド リスト]、[ソース]、[マージ履歴]、および[マージ サマリ]タブが用意されています。

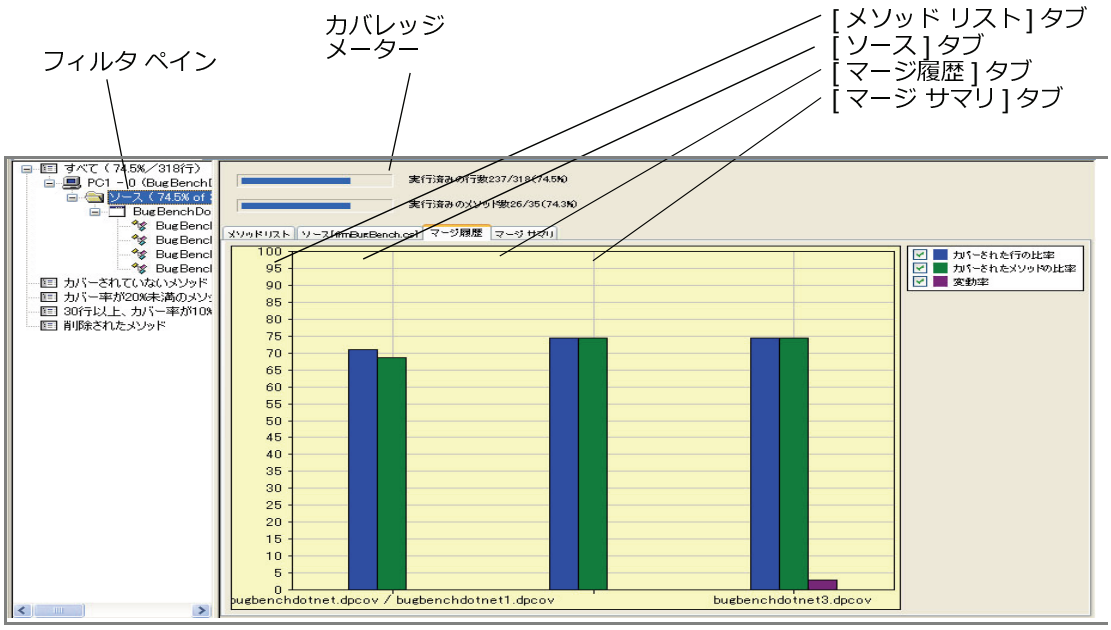


図4-8 マージ データ ウィンドウ

- [メソッド リスト]タブは、マージ ファイルの[状態]カラムを使用します。DevPartnerは、セッション間で新規、変更済み、または削除済みメソッドを区別するために[状態]カラムを使用します。
- [マージ履歴]タブには、現在のマージ ファイルの[カバーされた行の比率]、[カバーされたメソッドの比率]、および[変動率]の値の推移がグラフィカルに表示されます。
  - [カバーされた行の比率]は、ソース コード内の実行された行の比率です。
  - [カバーされたメソッドの比率]は、ソース コード内の呼び出されたメソッドの比率です。
  - [変動率]は最後のマージ以降にソース コードが変更されたメソッドの比率です。

  マージ ファイル内で4回以下のマージを行った場合、[マージ履歴]タブは棒グラフで表示されます。5回以上のマージを行った場合、[マージ履歴]タブは折れ線グラフで表示されます。

  グラフ上のポイントにカーソルを置くと、そのマージのデータが表示されます。

  棒グラフや折れ線グラフを表示または非表示にするには、そのチェック ボックスをオンまたはオフにします。
- [マージ サマリ]タブには、ファイルにマージされたセッションとマージ ファイルのサマリ情報が表示されます。各イメージの変動率など、セッション中に使用されるインストゥルメントされた各イメージの情報も含まれます。

ソース ファイルを変更した場合、カバレッジ セッション ファイルをマージすると、[メソッド リスト] タブに表示されるメソッド データと [ソース] タブに表示される行データの同期に影響が及ぶ点に注意してください。

マージ状態

コードを変更した場合、DevPartnerでは変更部分がトレースされ、トレース結果を基にカバレッジ データが調整されます。変更済み、新規、および削除済みのメソッドとイメージを区別するために、マージ状態が使用されます。これらの状態についての情報は、[メソッド リスト] の [状態] カラムに表示されます。

メソッド

メソッドの状態は新規、変更済み、削除済み、未変更のいずれかです。[状態] カラムには、新規および変更済みメソッドが示されます。[状態] カラムのエントリが空白の場合は、メソッドが変更されていないことを示します。

削除されたメソッドは、[削除されたメソッド] フィルタに表示されます。これらのメソッドは、カバレッジ統計の計算には使用されません。

DevPartnerでは、コード変更の程度は判断されません。たとえば、コメントの追加や削除など、メソッドの行数が変わるような変更を行うと、このメソッドが [変更済み] として表示されます。異なる最適化オプションを持つ実行可能ファイルを使用したセッションをマージすると、DevPartnerでこの違いが変更と解釈され、メソッドが [変更済み] として表示されることがあります。

イメージ

イメージは1つのセッションでロードできますが、他のセッションではロードできません。イメージがロードされていない場合は、DevPartnerではイメージ内にどのメソッドがあるのかを判定したり、イメージとそのメソッドを比較して別のセッションに関連する変更があったかどうかを調べたりすることはできません。

イメージの状態は新規、アクティブ、非アクティブのいずれかです。アクティブなイメージとは、マージ ファイル内の別のセッションに存在し再ロードされたイメージのことです。非アクティブな状態は、いくつかの状況から生じます。

- DevPartnerでは、イメージが削除された場合にそのイメージが非アクティブとしてマークされます。

ヒント：マージした最後のセッション ファイルを迅速に判定するには、マージ ファイルの [マージ サマリ] タブにある [マージ履歴] を確認します。

- ロードされなかったアンマネージ コード イメージの状態は常に [非アクティブ] として表示されます。たとえば、あるアンマネージ DLL がアプリケーションで使用されており、その DLL が 1 回めのセッションではロードされ、2 回めのセッションではロードされなかったものとします。この 2 回めのセッションを 1 回めのセッションとマージすると、この DLL の状態は非アクティブとして表示されます。アンマネージ コード プロジェクトおよびマネージ コード プロジェクトの両方を含むアプリケーションの完全なカバレッジ状態を把握するには、マージ ファイルに追加する最後のカバレッジ セッションでアプリケーションのアンマネージ コード部分を実行する必要があります。
- DevPartnerでは119ページに示すように、[除外] オプションを使用してカバレッジ データ収集からイメージが除外された場合、アンマネージ コード イメージが非アクティブとしてマークされます。

- マネージ アプリケーションでは、アセンブリは、アセンブリ自体とそれに対するすべての参照がアプリケーションから削除された場合にのみ非アクティブとしてマークされます。

非アクティブなイメージは、フィルタ ペインの [非アクティブなソース] フィルタに表示されます。これらのメソッドは、カバレッジ統計の計算には使用されません。そのため、DevPartner では、**[** 非アクティブなソース **]** フィルタに 0% という値が示されます。**[** 非アクティブなソース **]** フィルタを拡張すると、個々の非アクティブ イメージのカバレッジ値が示されます。これらの値には、イメージがアクティブであるセッションのカバレッジ データが反映されます。

マージ ファイルの ASP.NET モジュール

カバレッジ セッションでは、繰り返し可能なアルゴリズムが使用され、**.aspx** ファイル用の名前が生成されてアセンブリにコンパイルされます。このアルゴリズムは繰り返し可能であるため、アセンブリが登録されるごとに同じ名前が割り当てられます。この機能では各アセンブリに一貫性のある名前が指定されるため、アセンブリに対する変更を正確に追跡することができます。

このネーミング処理は、カバレッジ セッションを実行したときにのみ行われます。Visual Studio のデフォルトの動作は、**.aspx** ファイルを含むプロジェクトのビルドやリビルドを行っても変わりません。Visual Studio によって、各 **.aspx** ファイルに対して 8 文字の名前がランダムに割り当てられます。**.aspx** ファイルを編集してアセンブリをリビルドすると、Visual Studio によって、ランダムに作成された新しい 8 文字のファイル名が割り当てられます。

マージ設定

セッション ファイルを生成する際は、ソリューションのマージ プロパティを設定してデフォルトのマージ動作を制御します。

マージ プロパティを設定するには、Visual Studio のソリューション エクスプローラでソリューションを選択して、[Visual Studio プロパティ] ウィンドウを表示します。**DevPartner** カバレッジ、メモリ、およびパフォーマンス分析の各プロパティで **[** セッション ファイルを自動的にマージ **]** のプロパティを選択します。

- カバレッジ データを選択して蓄積し、マージしなかったセッションをマージするための確認メッセージを表示する場合は、**[** マージするかどうかを確認する **]** 設定を使用します。
- カバレッジ データを選択して蓄積し、マージされていないセッションについて確認するメッセージを表示しない場合は、**[** 確認せずに閉じる **]** を使用します。
- 各セッションについてマージ ファイルにカバレッジ データを蓄積する場合は、**[** 自動的にマージする **]** を使用します。

カバレッジ データをエクスポートする

カバレッジ データは XML 形式または CSV 形式にエクスポートできます。XML または CSV 形式でデータをエクスポートすると、ユーザー独自のソフトウェアまたはサードパーティのソフトウェアの使用によるデータの分析、他のツールで作成したデータとのデータ統合、データ ウェアハウス内でのデータのアーカイブに役立ちます。

- DevPartner のカバレッジ セッション ファイル（拡張子 **.dpcov**）とマージされたカバレッジ ファイル（拡張子 **.dpmrg**）を XML 形式にエクスポートできます。保存したカバレッジ セッション ファイルを開くと、**[** ファイル **]** メニューの **[DevPartner** データのエクスポート **]** コマンドを使用できます。XML 形式でのエクスポートの詳細については、「分析データを XML にエクスポートする」（309 ページ）を参照してください。

「分析データを XML にエクスポートする」（309 ページ）に説明されているように、コマンド ラインからデータをエクスポートすることもできます。

- メソッド リスト データをカンマ区切り（CSV）のテキスト ファイルにエクスポートできます。**[** メソッド リスト **]** タブをクリックしてエクスポート対象のカラムを表示し、[メソッド リスト] を右クリックして、コンテキスト メニューから **[** メソッド リストのエクスポート **]** を選択します。カンマ区切りテキスト ファイルは、Microsoft Excel やその他のスプレッドシート アプリケーションで開くことができます。

データ収集を制御する

DevPartner には、アプリケーションの使用中にカバレッジ データを収集するタイミングを制御する 3 つの方法が用意されています。

- セッション コントロール ツールバーを使用して、プログラムの実行中にデータ収集を対話的に制御できます。
- セッション コントロール ファイルを使用して、アプリケーション モジュール内の特定のメソッドに、セッション コントロール アクションを割り当てることができます。
- セッション コントロール API を使用して、プログラム内でデータ収集を制御できます。

セッション コントロール ツールバーまたはセッション コントロール API を使用すると、メソッド内の任意の場所でデータ収集を制御できます。セッション コントロール ファイルを使用する場合は、メソッドの先頭または終端でのみデータ収集を制御できます。

セッション コントロール ファイルの使用およびセッション コントロール API の使用については、「分析セッションの制御」（301 ページ）で説明します。

コマンド ラインから分析する

データ収集を自動化したり、コマンド ラインから分析セッションを実行したりするには、DevPartner コマンド ライン実行ファイル **`DPAnalysis.exe`** を使用します。**`DPAnalysis.exe`** の使用方法の詳細については、「コマンド ラインから分析を起動する」（283 ページ）を参照してください。

カバレッジ分析ビューアを使用する

DevPartner Studio には、Visual Studio とは独立してカバレッジ セッション ファイルを分析するための軽量のカバレッジ分析ビューアが付属しています。ビューアを起動するには、以下のいずれかを実行します。

- [スタート]メニューで、**[**すべてのプログラム**] > [Micro Focus] > [DevPartner Studio] > [** カバレッジ分析ビューア **]** を選択します。
- Windows エクスプローラで **`.dpcov`** セッション ファイルをダブルクリックします。
- **`DPAnalysis.exe`** を使用して、コマンド ラインからカバレッジ分析セッションを実行します。セッション データがカバレッジ分析ビューアに表示されます。

カバレッジ分析ビューアで行える作業

セッション ファイルを開いたまま、カバレッジ セッション データを表示、ソート、保存、印刷できます。さらに以下のことを行えます。

- メソッドのソース コードを表示する。
- [メソッド リスト] タブでデータをソートする。
- ファイルの内容を XML としてエクスポートする。
- メソッド リストの内容を CSV 形式でエクスポートする。

カバレッジ分析ビューアで行えない作業

- カバレッジ分析でのアンマネージ アプリケーションのインストゥルメント
- カバレッジ セッションの開始
- Visual Studio ソリューションにファイルを追加する。

Visual Studio の外部で生成されたセッション ファイルは、自動的にはプロジェクトのソリューションに追加されません。外部で生成されたセッション ファイルは、Visual Studio に開いたソリューションに手動で追加できます。

DevPartner エラー検出との統合

カバレッジ分析と併せて DevPartner エラー検出を使用すると、マネージ アプリケーションまたはアンマネージ C/C++ アプリケーションの実行時に同じセッション中でカバレッジ データを収集してエラーを検出できます。データを収集する前に、ネイティブ C/C++ インストゥルメンテーション マネージャを使用して、[エラー検出とカバレッジ] の対象のアンマネージ C/C++ アプリケーションをインストゥルメントする必要があります。

DevPartner エラー検出の詳細については、「エラー検出」（123 ページ）を参照してください。

Visual Studio Team System にデータを送信する

DevPartner Studio では、Team Explorer クライアントがインストールされており、Team Foundation Server に接続可能な場合に、Microsoft Visual Studio Team System がサポートされます。Team System のサポートの概要については、「Visual Studio Team System のサポート」（10 ページ）を参照してください。

Visual Studio 2005 および 2008 については、選択した項目に関するデータをバグタイプの作業項目として Visual Studio Team System に送信します。Visual Studio 2010 では、選択した項目に関するデータを問題、バグ、または不具合タイプの作業項目として Team Foundation Server に送信します。カバレッジ分析セッション ファイルでは、[メソッド リスト] タブで選択したメソッドの作業項目の送信にアクセスします。作業項目を送信する場合、[メソッド リスト] タブで表示されるカラムからのデータが [作業項目] 形式で入力されます。[作業項目] として送信するメソッド データを変更するには、[メソッド リスト] に表示されるカラムを変更します。

第5章

メモリに関する問題を検出する

この章には、2つのセクションが含まれています。1つめのセクションでは、はじめてのユーザーを対象として、メモリ分析を実行する簡単な手順について説明します。2つめのセクションでは、DevPartnerメモリ分析の使用方法を詳細に理解するための参照情報を示します。

メモリ分析に関するタスクベースの追加情報については、DevPartnerのオンライン ヘルプを参照してください。

メモリ分析の機能

DevPartnerメモリ分析機能を使用すると、マネージVisual Studioアプリケーション内のメモリ割り当てを分析できます。

DevPartnerメモリ分析では、コンテキストに応じてメモリのデータが表示されるため、コード内のオブジェクト参照のチェーンやメソッドのコール シーケンスをたどることができます。これにより、プログラムによるメモリの使用状況をより詳細に理解でき、メモリ使用を最適化するために必要となる重要な情報を入手できます。

メモリ分析を有効にしてアプリケーションを実行すると、オブジェクトまたはクラスによって使用されたメモリの量が示され、オブジェクトをメモリに保持している参照が追跡され、メソッド内でメモリ割り当てを行うソース コード行が識別されます。

DevPartnerメモリ分析には、メモリ リーク、一時オブジェクト、RAMフットプリントの3つの分析タイプが用意されています。1つのメモリ分析セッションで、3つすべてのタイプのメモリ分析を実行できます。

各分析タイプには、リアルタイム グラフ、動的に更新されるクラス リスト、およびいくつかのセッション コントロールが含まれています。これらのセッション コントロールを使用すると、データの収集を制御したり、アクティブなプロセスに対するガベージ コレクションの強制実行やヒープの詳細ビューの作成などのその他のメモリに関するイベントを制御したりできます。

メモ： DevPartnerメモリ分析機能では、マネージ コードのみが分析されるため、この機能はDevPartner for Visual C++ BoundsChecker Suiteではサポートされていません。

メモリ分析はVisual Studioに統合されているため、アプリケーションの開発中にメモリ分析を使用してアプリケーションをテストできます。また、従来のコマンド ライン スイッチまたはXML構成ファイルを指定してDevPartnerコマンド ライン実行ファイル**`DPAnalysis.exe`**を使用することによって、コマンド ラインから、または自動テスト シナリオの一部としてメモリ分析セッションを実行することもできます。詳細については、「コマンド ラインから分析を起動する」(283ページ)を参照してください。

すぐにメモリ分析を使用する

以下の準備、設定、実行手順を使用すると、DevPartner Studio メモリ分析機能の 1 つである メモリ リーク分析を使用できます。メモリ リーク分析。

すぐに使い始めるには、影付きのボックス内に示された手順に従います。影付きのボックス内に説明されている内容の詳細については、ボックスの下の追加説明を参照してください。

DevPartner Studio でアプリケーションを分析する場合、昇格されたシステム権限は必要ありません。DevPartner でのアプリケーションの分析には、アプリケーションの作成とデバッグに使用するシステム権限で十分です。

準備：分析対象を検討する

時間が経過すると、または特定の処理を実行すると、アプリケーションのパフォーマンスが低下することはありませんか。負荷が高い状態、または他のアプリケーションが実行されている場合にアプリケーションのパフォーマンスが低下することはありませんか。アプリケーションでこのような現象が発生する場合は、メモリ関連の問題が発生している可能性があります。

DevPartner メモリ分析では、マネージ アプリケーションのデータのみが収集されます。アプリケーションのメモリ分析データを収集するには、ソリューションに少なくとも 1 つのマネージ コード プロジェクト（たとえば C#、Visual Basic、またはマネージ C++）を含める必要があります。また、ソリューションにスタートアップ プロジェクトを含める必要があります。ソリューションに複数のスタートアップ プロジェクトが含まれている場合は、セッションで使用するスタートアップ プロジェクトを選択するように求めるプロンプトが表示されます。

以降の手順では、以下の事項を前提としています。

- Visual Studio のサポートされているリリースで作業しています。
- 単一プロセスのマネージ アプリケーションをテストします。
- アプリケーションのビルドと実行が可能です。
- ソリューションには、少なくとも 1 つ以上のマネージ コード プロジェクトが含まれています。
- ソリューションには、スタートアップ プロジェクトが含まれています。

DevPartner メモリ分析でサポートされているすべてのプロジェクト タイプのリストは、「DevPartner Studio サポートされるプロジェクト タイプ」（169 ページ）を参照してください。

アプリケーションで使用されるメモリの量は、アプリケーション プラットフォームのパフォーマンスに大きな影響を与えます。割り当てられるメモリの量が多いほど、アプリケーションの実行速度が遅くなり、拡張性も低くなります。

メモリ リーク（再利用されないメモリの割り当て）によって、アプリケーションの RAM フットプリントが大きく増加する可能性があります。自動ガベージ コレクションによって、作成したオブジェクトを明示的に解放する必要がなくなるため、従来の C++ で発生したようなメモリ「リーク」はなくなりますが、プログラムでオブジェクトへの参照を保持したまま再利用しない状態は発生します。

オブジェクトへの参照が存在するかぎり、参照されるオブジェクトはガベージ コレクタによってライブ オブジェクトであるとみなされます。ライブ オブジェクトのメモリは収集されません。この状態は、C++ におけるメモリ リークと同様に好ましくありません。このような参照は追跡することが難しいことがあるため、メモリ分析が役立ちます。

この手順では、単一プロセスのアプリケーションを前提としていますが、DevPartner Studio を使用して複雑な複数プロセスのアプリケーションを分析することもできます。複数プロセスのアプリケーションをプロファイルする方法の詳細については、「複数プロセスのデータを収集する」 (78ページ)を参照してください。

設定：プロパティとオプション

メモリ分析セッションに固有の、最小限の構成設定のセットがあります。

> この手順では、DevPartnerのデフォルトのプロパティとオプションを使用できます。追加の設定は必要ありません。

メモリ分析の実行中にアプリケーションが極端に遅くなる場合は、分析からシステム オブジェクトを除外することによってパフォーマンスが向上することがあります。[システム オブジェクトの追跡] 設定とその他のメモリ分析設定の変更の詳細については、「プロパティとオプションを設定する」 (46ページ)を参照してください。

実行：メモリ分析データを収集する

メモリ リーク分析を開始する前に、分析のワークフローについて理解しておくと役立ちます。

追跡の開始 /停止をクリックすると、新規メモリ割り当ての追跡期間の開始と終了をマークできます。これ以外の期間でのアプリケーションによるすべてのメモリ割り当ては除外されます。

[メモリリークを表示]をクリックした時点では、追跡期間中に割り当てられたオブジェクトの一部またはすべてがタスクを完了して、ガベージ コレクションが可能な状態になっています。

メモリ分析によって、追跡期間中に収集されたすべての割り当てが分析され、ライブ参照を持ち収集不可能なオブジェクトとしてリークが特定されます。

[メモリリークを表示]では、ガベージ コレクションが強制的に実行され、セッション ファイルが作成されて、リークしているオブジェクトが複数のグラフィックとリスト ビューに表示されます。図 5-1、メモリ リーク分析ワークフロー タイムライン (38ページ)に説明されているシナリオでは、メモリ リーク セッション ファイルには、ガベージ コレクション後も残ってリークしている2つのオブジェクトBとCが含まれます。データから、リークしているオブジェクトのうち、想定どおりの動作をしているオブジェクトと、実際のリークであるオブジェクトを判別します。

ガベージ コレクションは、システム ガベージ コレクションまたはガベージ コレクション アイコンや[メモリリークを表示]アイコンを選択することによるユーザー主導のガベージ コレクションを実行できます。

メモリ割り当て
X – 追跡対象外、ガベージ コレクションされない　想定どおり）
Y – 追跡対象外、ガベージ コレクション済み
A – 追跡対象、ガベージ コレクション済み
B – 追跡対象、ガベージ コレクションされない　想定どおり）
C – 追跡対象、ガベージ コレクションされない、問題の可能性あり

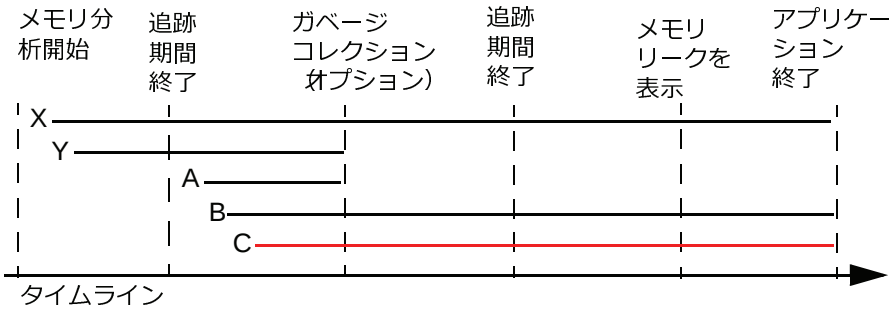


図5-1 メモリ リーク分析ワークフロー タイムライン

これで、メモリ リーク分析を実行する準備が整いました。

1. Visual Studio で、アプリケーションに関連付けられたソリューションを開きます。
2. **[DevPartner]>[**デバッグを実行せずにメモリ分析を選択して開始**]**を選択します。アプリケーションが起動して、[DevPartner メモリ分析] ウィンドウにセッション コントロール ウィンドウが表示されるまで待機します。

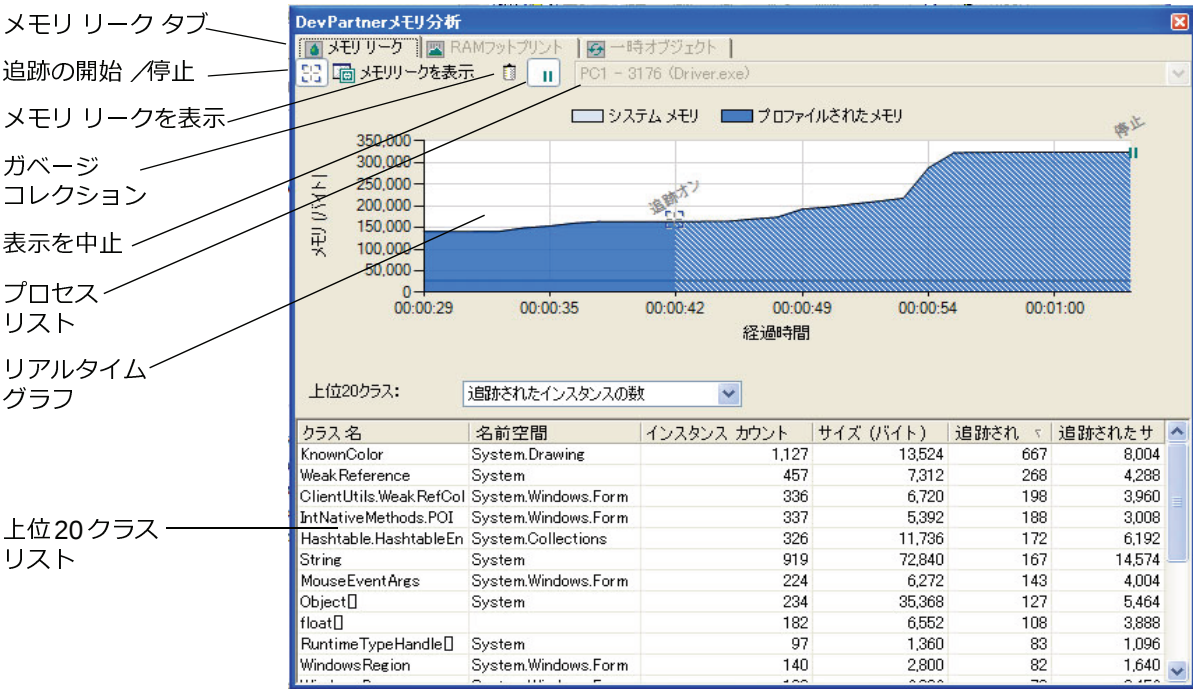


図5-2 メモリ分析セッション コントロール ウィンドウ

3 [メモリ リーク]タブをクリックします。

4 テスト対象の機能を実行して、アプリケーションをウォーム アップします。アプリケーションをウォーム アップすると、追跡期間から初期化時の割り当てが除外されます。

5 追跡の開始 /停止  をクリックして、以前のメモリ割り当てを除いた新規メモリ割り当ての追跡を開始します。

6 データ収集の対象であるアプリケーション機能を一度最後まで実行します。ただし、アプリケーションは終了しません。

この手順を実行する場合は、追跡期間をアプリケーション内の1つの機能を実行する間のみに制限します。これにより、セッション データの複雑度が低下し、パフォーマンスが向上します。

アプリケーションの実行中にセッション コントロール ウィンドウのグラフに表示されるパターンを見ると、アプリケーションがどのようにメモリを使用しているかの初期的な診断が可能になります。メモリに関する問題は、それぞれの種類によって特徴のあるパターンを示すため、リアルタイム グラフを見ると、問題の存在や性質を知るための重要な手がかりを得ることができます。これによって、実行すべきメモリ分析の種類を決定できます。

たとえば、徐々に上昇しているパターンのうち、ベースラインに戻らないか、ガベージ コレクションによって想定どおりにメモリ使用量が減少しないものは、メモリのリークを示している可能性があります。

リアルタイム グラフにおける他の特徴的なパターンの詳細については、「メモリ分析のセッション コントロール ウィンドウを使用する」 (149ページ)を参照してください。

複雑なアプリケーションでは、リストに大量のクラスが表示される場合があります。このような場合は、リストを右クリックし、コンテキスト メニューから[ソースのある上位20クラスを表示]を選択して、クラス リストをアプリケーションのソース コード メソッドに限定します。

7 ガベージ コレクションを実行 をクリックして、アクティブなプロセスに対してガベージ コレクションを実行します。

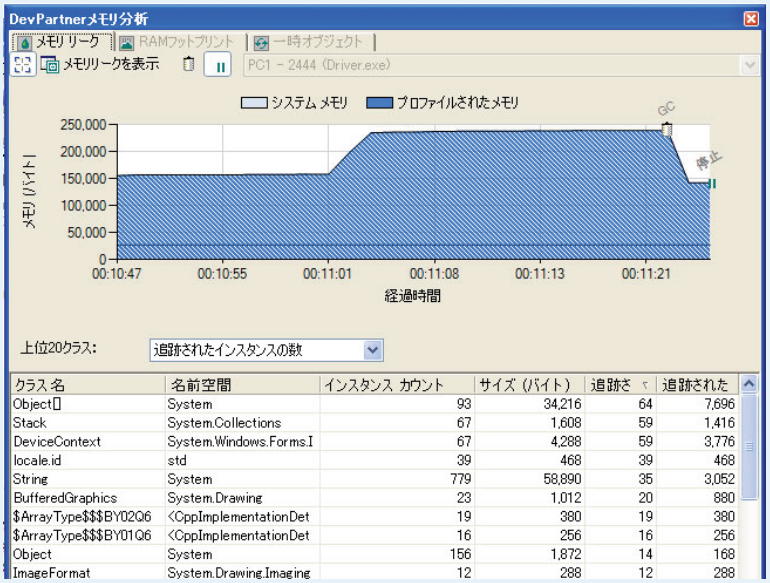


図 5-3 ガベージ コレクション後のセッション コントロール ウィンドウ

8 セッション コントロール ウィンドウの下半分にあるリストを確認します。リストには、ガベージ コレクション後にもアクティブな参照を持つオブジェクトを含むクラスの情報が表示されます。

9 追跡の開始／停止 をクリックして、追跡期間を終了し、以降の新規メモリ割り当てを除外します。アプリケーションがバックグラウンドでアクティブである場合は、リストの値が変化することがありますが、追跡済みのインスタンス数は増加しません。

10 [メモリリークを表示]をクリックして、再度ガベージ コレクションを実行し、メモリ リーク分析セッション ファイルを作成します。

11 アプリケーションを終了します。

メモリ分析によって自動的に2つめのファイル（一時オブジェクト分析セッション ファイル）が作成されて、Visual Studio内でフォーカスが設定されます。

12 **[Leak...Analysis Snap.dpmem]**タブをクリックして、メモリ リーク スナップショット セッション ファイルにフォーカスを移動します。

メモリ分析データを分析する

メモリ リーク分析セッション ファイルには、[メモリリークを表示]をクリックした時点でアクティブな参照を持つ、追跡期間中に割り当てられたすべてのオブジェクトが記録されます。セッション ファイルを使用して、オブジェクト、メソッド、およびクリティカル実行パスを確認し、これらのオブジェクトがメモリに残っている理由を分析します。

メモリ リーク分析は、アプリケーションのコンテキストにおいて不要なオブジェクトを特定して、これらのオブジェクトをメモリに保持している参照を削除するための参照チェーン内の最適な位置を発見する場合に役立ちます。

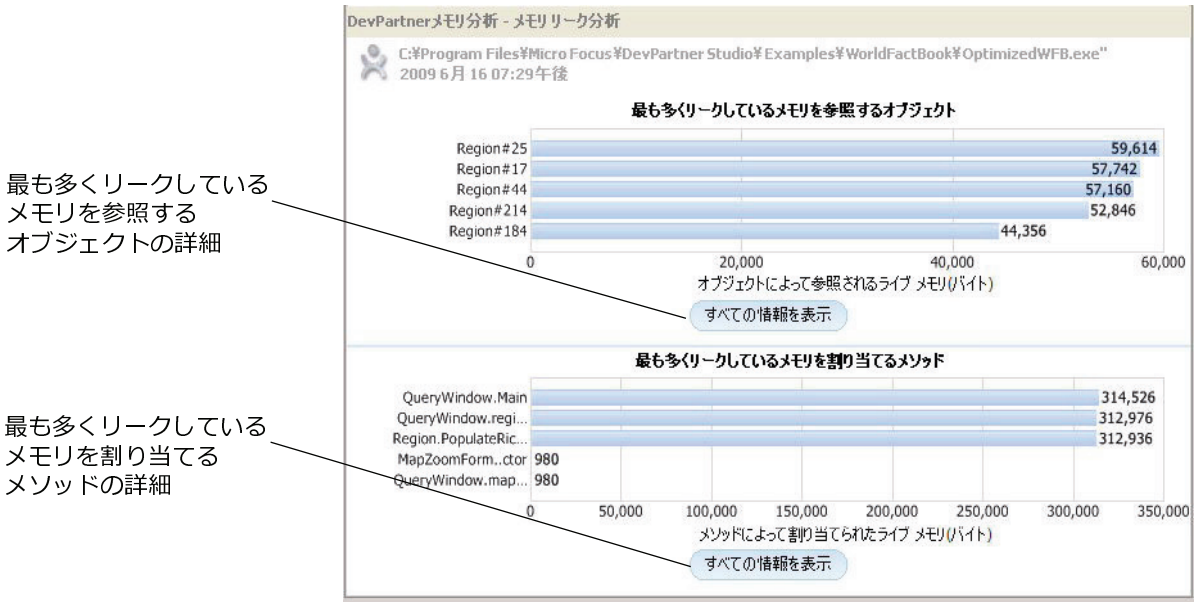


図5-4 DevPartnerDevPartner メモリ分析 - メモリ リーク分析

[DevPartner メモリ分析 - メモリ リーク分析] サマリには、[最も多くリークしているメモリを参照するオブジェクト] と [最も多くリークしているメモリを割り当てるメソッド] の棒グラフが含まれています。

この手順の残りの部分で、これら 2 つのサマリの詳細を調査する方法について説明します。

1 [最も多くリークしているメモリを参照するオブジェクト] の下の [すべての情報を表示] をクリックします。

メモリ分析セッション ファイルの使用方法の詳細については、「オブジェクト参照グラフを使用する」 (153 ページ) を参照してください。

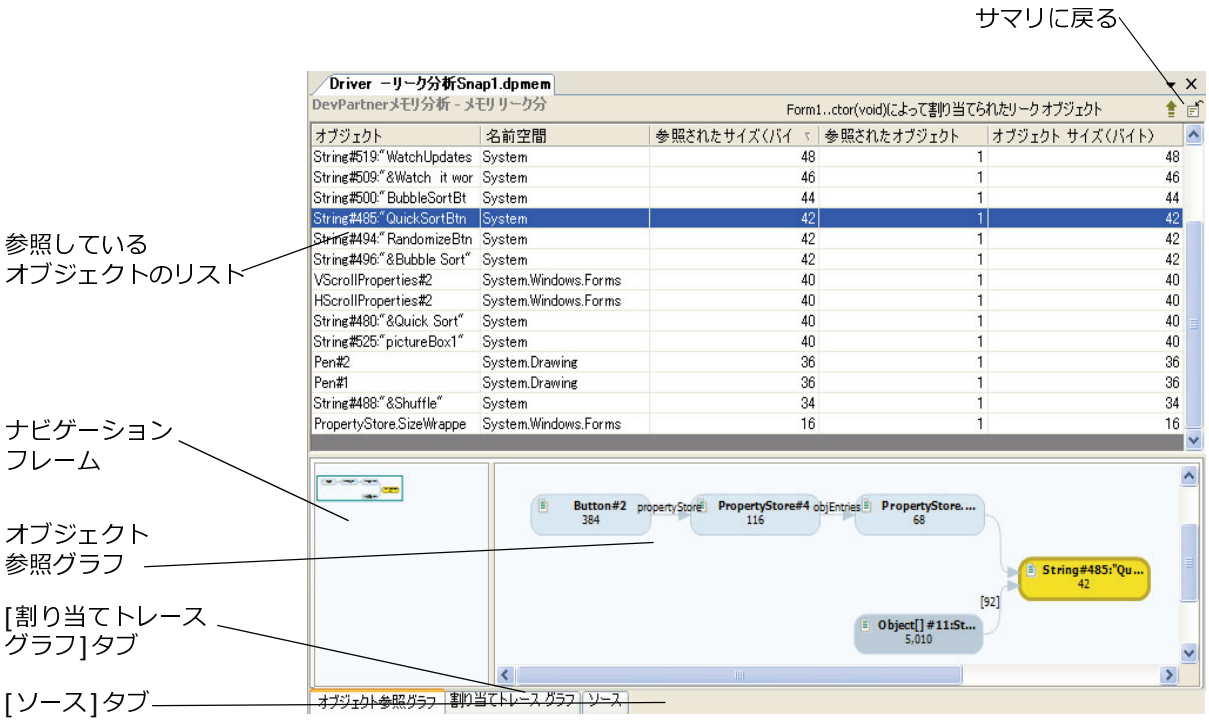


図 5-5 DevPartner メモリ分析 - メモリ リーク オブジェクト参照の詳細

2 [リーク サイズ (バイト)]でソートされた参照しているオブジェクトのリストが表示される[DevPartner メモリ分析 - メモリ リーク]ビューを確認します。最も多いメモリ リークの原因である[参照しているオブジェクト]がリストの一番上に表示されます。

ウィンドウの下部にあるタブには、[オブジェクト参照グラフ]ウィンドウ、[割り当てトレース グラフ]ウィンドウ、および[ソース]ウィンドウが表示されます。

3 [オブジェクト参照グラフ]タブを選択して、参照しているオブジェクトを上部のリストから選択します。

参照しているオブジェクトをリストから選択すると、選択されたオブジェクトがオブジェクト参照グラフで強調表示されます。

4 オブジェクト参照グラフでオブジェクト ノードの上にマウスを置くと、そのオブジェクトに関連付けられたメモリ リークの情報が表示されます。

5 概要ペイン内のナビゲーション フレームをドラッグすると、オブジェクト参照グラフのさまざまな部分にフォーカスを移動できます。

6 リストの[参照しているオブジェクト]を右クリックして、[このオブジェクトによって参照されるリーク オブジェクトを表示]を選択します。デフォルトのソート順序は、オブジェクトを収集した場合に解放できるメモリの量を示す[参照サイズ (バイト)]です。

オブジェクトがまだ参照されている理由を理解する必要があります。この段階では必要に応じて、参照を断ち切る場所をコード内で決定できます。

さらに詳細な情報が必要な場合や、プログラムについてさらに理解する必要がある場合は、タブ付きのビューを使用して、オブジェクト参照を確認し、メモリを割り当てた実行パスを特定して、ソース コード内の実際にメモリを割り当てている行を特定します。

- [オブジェクト参照グラフ] には、オブジェクトと関連するオブジェクト参照がグラフィカルに表示されます。このグラフには、オブジェクト自体とその下位オブジェクトによって使用されているメモリや、オブジェクトによって使用されているメモリの割合などの関連情報と共に各オブジェクトが表示されます。
- 割り当てトレース グラフには、コード内の実行パスがグラフィカルに表示されます。これにより、オブジェクトが割り当てられたコンテキストを把握できます。
- [ソース] ウィンドウには、各オブジェクトに関連するソース コードが表示されます。

コストの高いオブジェクト参照について検討し、オブジェクト参照を異なる方法で管理することによってアプリケーションを最適化できるかどうかを決定します。

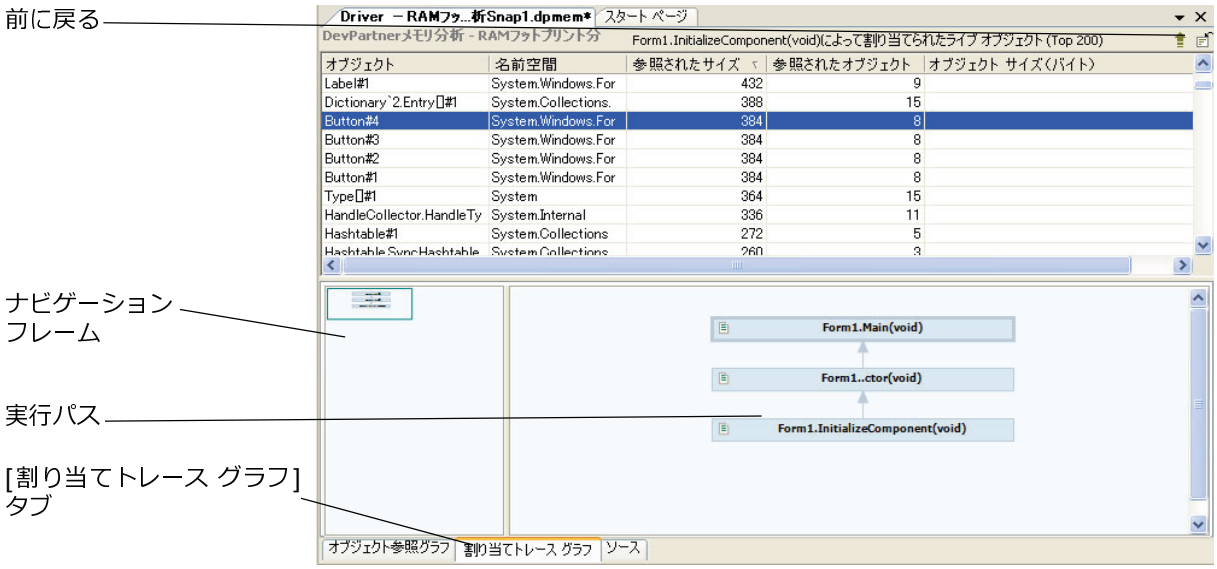


図 5-6 DevPartner メモリ分析 - 割り当てトレース グラフ

7 変更を行うことを決定した場合は、[割り当てトレース グラフ] タブを選択し、オブジェクトを作成してメモリを割り当てた実行パスを確認します。

8 [ソース] タブをクリックし、オブジェクト リスト内のオブジェクトを選択します。異なるオブジェクトを選択すると、ソース コードの参照も変更されます。

9 リスト内のオブジェクトを右クリックし、[ソースの編集] を選択して、Visual Studio エディタ内に関連するソース コード行を表示します。

システム オブジェクトでは、編集可能なソース コードはありません。

10 Visual Studio エディタを閉じます。

詳細な例については、「最も多くリークしているメモリを参照するオブジェクト」(163 ページ) を参照してください。ソース コードにアクセスするためのさまざまな方法については、「[ソース] タブをナビゲートする」(156 ページ) を参照してください。

オブジェクト参照に関連するソース コードを特定したあと、オブジェクトのレベルを超えて、オブジェクトとオブジェクト参照との間の相互関係のレベルでメモリ管理を確認します。以降では、オブジェクト参照を異なる方法で管理することによってアプリケーションのパフォーマンスを向上できるかどうかという観点から決定を行います。

オブジェクト参照管理の変更には、より小さなオブジェクト、弱い参照、異なるオブジェクト参照順序の使用や、抽象化レイヤの数を制限することなどがあります。

Web アプリケーションで拡張性が課題となっている場合、クライアント /サーバー間の関係について理解しておくと、サーバーでガベージ コレクションを有効に活用できる可能性があります。

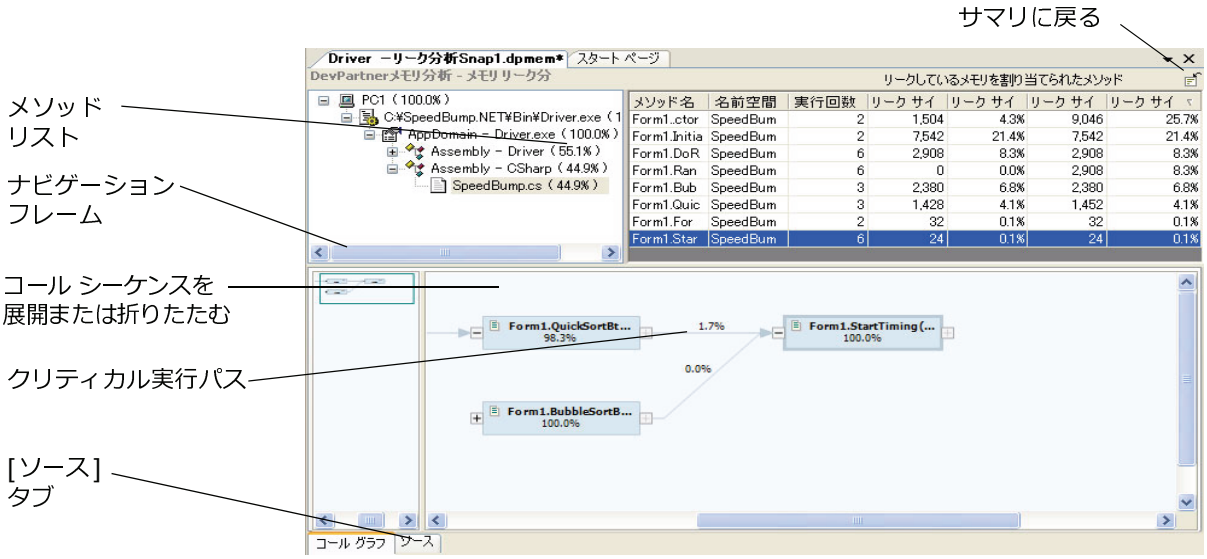


図 5-7 DevPartner メモリ分析 – メソッド リストとコール グラフ

11 [DevPartner メモリ分析 - メモリ リーク分析セッション ファイル] タブを選択し、 をクリックして、サマリ ページに戻ります。

[最も多くリークしているメモリを参照するオブジェクト] 以外に、[最も多くリークしているメモリを割り当てるメソッド] を分析することもできます。

12 サマリ ページから、[最も多くリークしているメモリを割り当てるメソッド] の [すべての情報を表示] を選択します。

メソッド リストに、最も多くリークしているメモリを割り当てたソース メソッドが表示されます。

13 メソッド リストでメソッドを選択して、コール グラフを表示します。

14 [コール グラフ] ウィンドウで、メソッド ノードまたはメソッド ノード間のライン上にマウスを置きます。メソッド ノードとその下位ノードによるリーク サイズを比較します。

クリティカル実行パスは、金色の太線で強調表示されます。

15 [+] と [-] のコントロールを使用して、メソッド ノードのコール シーケンスをさまざまなレベルに展開または折りたたみます。

16 上部のリストでメソッド名を右クリックして、コンテキスト メニューから [ソースを表示] を選択します。

17 リストでメソッド名を右クリックして、[サマリを表示] を選択します。

詳細な例については、「最も多くリークしているメモリを割り当てるメソッド」 (165 ページ) を参照してください。

セッション ファイルを保存する

メモリ分析データを参照し終わったあと、セッション ファイルを保存できます。

1. Visual Studioのセッション ファイル ウィンドウを閉じて、セッション ファイルを保存します。
2. **[OK]**をクリックして、デフォルトの名前と場所でファイルを保存します。

DevPartnerでは、セッション ファイルはアクティブなソリューションの一部として保存されます。これらのファイルは、ソリューション エクスプローラのDevPartner Studio仮想フォルダに表示されます。メモリ分析セッション ファイルには拡張子**.dpmem**が付きます。

デフォルトでは、セッション ファイルはプロジェクトの出力フォルダに保存されます。ファイル名には、デフォルト フォルダに配置されているファイル名に、連続した番号が自動的に追加されます（たとえば、**MyApp-TemporaryObjectSnap1.dpmem**、**MyApp-LeakAnalysisSnap1.dpmem**のようになります）。セッション ファイルをデフォルト フォルダ以外の場所に保存した場合は、ユーザーがファイル名と番号を管理する必要があります。

出力フォルダがないプロジェクトの場合（Visual Studio 2005 Webサイト プロジェクトなどの場合）、ファイルはプロジェクト フォルダに物理的に保存されます。

コマンド ライン ユーティリティで生成されたセッション ファイルは、プロジェクトのソリューションに自動的には追加されません。外部で生成されたセッション ファイルは、Visual Studioに開いたソリューションに手動で追加できます。

この章の残りの部分では、メモリ分析のメモリ リーク、一時オブジェクト、およびRAMフットプリントの各機能の参照情報を示し、各機能について詳細に説明します。

これで、この章の準備、設定、実行のセクションは終了です。これで、メモリ分析セッション実行のメカニズムの基礎が理解できました。追加情報については、引き続きこの章の残りの部分を参照してください。タスクベースの情報については、**DevPartner**のオンライン ヘルプを参照してください。

マネージ **Visual Studio** アプリケーションにおけるメモリに関する問題

マネージVisual Studioアプリケーションでは、ガベージ コレクションを含む高度なメモリ管理環境を利用できます。割り当てるメモリを明示的に解放するアンマネージ（ネイティブ）C++とは異なり、メモリを割り当てられたオブジェクトが使用されなくなると（より正確には、アプリケーションから「アクセス不可能」になると）、そのメモリはガベージ コレクタによって解放されます。

マネージ コード プロジェクトには組込みのメモリ管理機能が備えられているため、多くの開発者は、マネージ言語では従来のようなメモリ管理に関連する問題は発生しなくなったと誤解しています。しかし、マネージVisual Studioプログラムにおけるメモリの割り当てや使用によっても、パフォーマンスのボトルネックになったりリソース不足に陥ったりすることがあります。

アプリケーションで以下のような現象が発生していないか確認してください。

- ◇ 時間がたつにつれてパフォーマンスが低下する。

- ◇ 実行が遅い、または特定の処理を実行すると明らかに速度が遅くなる。
- ◇ 負荷が高い状態でのパフォーマンスが低い。
- ◇ 他のアプリケーションが実行されている場合のパフォーマンスが低い。

これらのいずれの現象も、アプリケーションにパフォーマンスの問題が発生していることを示しています。問題がメモリの使用に関連しているかどうかをよりよく把握するために、以下の質問のリストを参照してください。

- ◆ プログラムで特定の機能を実行する前に、いくつかのアプリケーション クラスをロードする必要があり、各アプリケーション クラスでメモリを使用する場合
  - ◇ プログラムでは、現在のタスクの実行に関連したクラスのみをロードしていますか。
  - ◇ アプリケーションで特定のクラスのインスタンスをいくつ作成しますか。それらはすべて必要ですか。
- ◆ オブジェクトを割り当てたときに、パフォーマンスの問題を発生させる可能性があるメモリの使用も行われる場合
  - ◇ プログラムでオブジェクトが必要以上に割り当てられているか、または効率的に割り当てられているかを把握していますか。
  - ◇ プログラムで割り当てられたオブジェクトは、ガベージ コレクタによってクリアされていますか。
  - ◇ オブジェクトは想定どおりに収集されていますか。あるいは使用後も長時間オブジェクトがメモリに残っていますか。

メモリ分析の機能

DevPartner メモリ分析機能には、マネージ アプリケーションにおけるメモリの使用に関する包括的なビューが備えられています。DevPartner には、3 種類のメモリ分析機能が用意されています。これらは、メモリに関する各種の問題を特定する場合に役立ちます。使用する分析のタイプにかかわらず、すべてのタイプの分析には以下の機能が含まれています。

- ◆ リアルタイム グラフ – 実行中のアプリケーション内での、メモリ使用状況のライブ ビューが表示されます。このビューは、セッション コントロール ウィンドウに表示されます。アプリケーション コード（プロファイルされているコード)、システム、およびその他のアプリケーション コード（除外されているコード）によってどの程度のメモリが使用されているか、マネージ ヒープ用に予約されているメモリと比較して、どの程度のメモリが消費されているかを確認できます。
- ◆ クラスの動的なリスト – プロファイルされているクラスのリストがアプリケーション実行中にリアルタイムに更新されます。このリストには、割り当てられたオブジェクトの数と、各クラスが使用するバイト数がアプリケーション実行中に表示されます。
- ◆ 詳細ヒープ ビュー – プログラム実行中、いつでもマネージ ヒープの詳細なビューのスナップショットをキャプチャできます。

プロパティとオプションを設定する

メモリ分析セッションを開始する前に、データ収集を微調整して、特定のタイプの情報を含めるか、または除外することをお勧めします。ソリューション プロパティ、プロジェクト プロパティ、および DevPartner オプションを使用して、分析セッションをより目的に適したものにします。

ソリューション プロパティ

メモリ分析に影響を与えるソリューション レベルのプロパティを表示するには、ソリューション エクスプローラでソリューションを選択し、[F4]キーを押して、[プロパティ]ウィンドウを表示します。

図5-8 ソリューション プロパティ

以下のソリューション プロパティがメモリ分析に影響を与える可能性があります。

- **.NETから収集** – **[False]**に設定されている場合でも、メモリ分析を有効にしてマネージ アプリケーションを実行すると、このプロパティは上書きされます。メモリ分析では、常にマネージ アプリケーションからデータが収集されます。
- スタートアップ　プロジェクト – ソリューションに複数のプロジェクトが含まれている場合は、スタートアップ プロジェクトを変更できます。スタートアップ プロジェクトのプロジェクト プロパティによって、セッション内のすべてのアクティブなプロジェクトのデータ収集が制御されます。

ソリューションには、スタートアップ プロジェクトを含める必要があります。ソリューションに複数のスタートアップ プロジェクトが含まれている場合は、分析の開始前に、セッションで使用するスタートアップ プロジェクトを選択するように求めるプロンプトが表示されます。

このプロンプト ダイアログには、ソリューション プロパティの**[**共通プロパティ**]** > **[**スタートアップ プロジェクト**]**ページにある**[**アクション**]**が**[**開始**]**に設定されているプロジェクトだけが表示されます。目的のスタートアップ プロジェクトがプロンプト ダイアログに表示されない場合は、ソリューション プロパティ ページを開き、該当プロジェクトの**[**アクション**]**を**[**開始**]**に設定します。後続のセッションに新しいスタートアップ プロジェクトを選択する場合は、新しいスタートアップ プロジェクトのプロパティを見直して、データ収集オプションが正しいことを確認してください。

プロジェクト プロパティ

プロジェクト レベルのプロパティを参照するには、ソリューション エクスプローラでプロジェクトを選択して、ソリューション内のプロジェクトに設定可能なプロパティを参照します。

ここでの変更は、カバレッジ分析、メモリ分析、パフォーマンス分析、およびパフォーマンス エキスパートに影響を与えます。

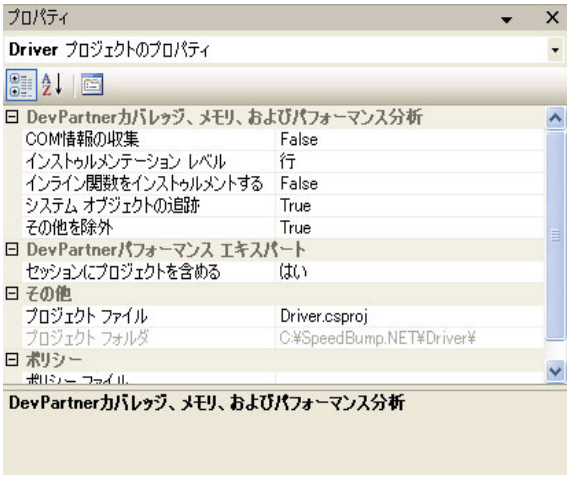


図 5-9 プロジェクト プロパティ

メモリ分析に影響を与えるプロジェクトレベルのプロパティは、以下のとおりです。

- システム オブジェクトの追跡 – このプロパティを **[False]** に設定すると、メモリ分析セッションでの割り当てオブジェクトの追跡時に、システムまたはサードパーティ製オブジェクトの割り当てが無視されます。
  デフォルトの状態は **[True]** になっています。この場合は、システム リソースまたはその他のプロファイルされていないリソースによって行われたメモリの割り当てを確認できます。

オプション

メモリ分析セッションに対する DevPartner のオプション設定を確認するには、**[DevPartner]** > **[オプション]** > **[分析]** を選択します。

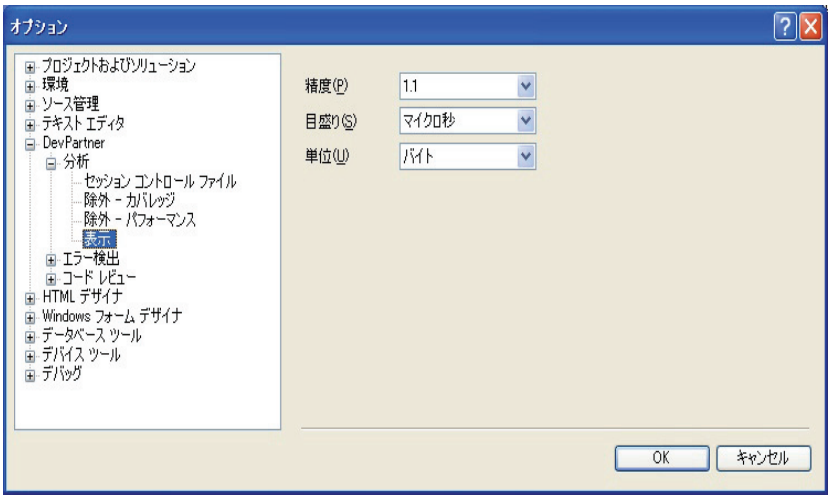


図 5-10 分析オプション

- 精度 – 小数点以下第 1、2、3、4 位から選択できます。
- 単位 – 選択肢には、[バイト]、[KB]、[MB] があります。

- [セッション コントロール ファイル] オプションを使用すると、アプリケーションやモジュールの実行時に DevPartner によって収集されるデータを制御するためのルールとアクションのセットを作成できます。セッション コントロール ファイルの詳細については、「Visual Studio 内でセッション コントロール ファイルを作成する」 (301 ページ) を参照してください。

[環境] > [フォントと色] などのその他の Visual Studio オプションも、DevPartner の機能に影響を与えます。

メモリ分析セッションを開始する

デバッグあり、またはデバッグなしでメモリ分析セッションを実行できます。[DevPartner] メニューからメモリ分析セッションを開始する場合は、デバッグなしのオプションのみを選択できます。プロジェクトまたはソリューションを開いたあと、メモリ分析アイコンの右側で、デバッグあり、またはデバッグなしでのセッションの開始を選択できます。

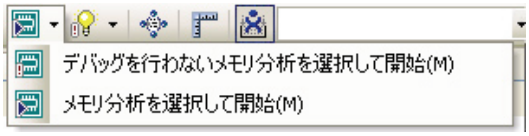


図 5-11 デバッグありまたはデバッグなしでの開始を選択するためのメモリ分析アイコンのオプション

分析結果を理解しやすく、パフォーマンスにも優れているため、メモリ分析のデフォルト設定は [デバッグを実行せずにメモリ分析を選択して開始] になっています。コードにブレーク ポイントを設定して、[メモリ分析を選択して開始] (デバッグあり) を指定すると、コードの特定のセクションのパフォーマンスのみを分析できます。

ブレーク ポイントを設定する代わりに、**`SessionControl.txt`** ファイルまたはセッション コントロール API を使用してプログラム実行中にメモリ分析アクションを実行し、コードの特定のセクションのパフォーマンスのみを分析することもできます。セッション コントロール ファイルの詳細については、「Visual Studio 内でセッション コントロール ファイルを作成する」 (301 ページ) を参照してください。

メモリ分析のセッション コントロール ウィンドウを使用する

新しいメモリ分析セッションを開始すると、セッション コントロール ウィンドウが開きます。セッション コントロール ウィンドウの各タブは、分析可能なメモリに関する問題の各タイプであるメモリ リーク、一時オブジェクト、RAM フットプリントに対応しています。各タブには、リアルタイム グラフのビュー、動的に更新されるクラス リスト、およびいくつかのセッション コントロールが含まれています。これらのセッション コントロールを使用すると、データの収集や、ガベージ コレクションなどのその他のメモリに関するイベントを制御できます。クラス リストに示されるデータや使用可能なセッション コントロールは、選択したタブごとに若干異なります。

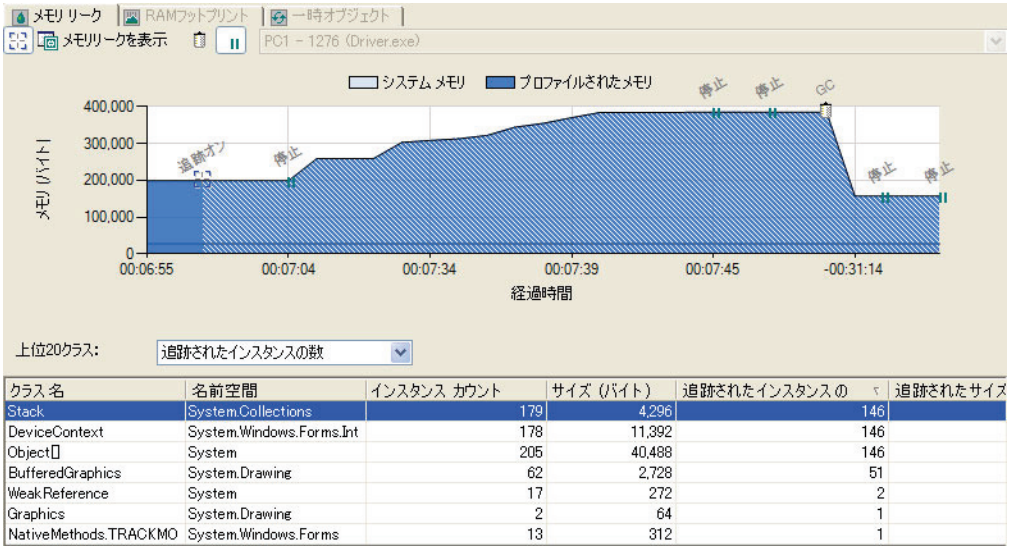


図 5-12 DevPartner メモリ分析セッション コントロール ウィンドウ

リアルタイム グラフのパターン

セッションを開始したときにリアルタイムのグラフを観察します。アプリケーションの実行中にグラフに表示されるパターンを見ると、アプリケーションがどのようにメモリを使用しているかの初期的な診断が可能になります。メモリに関する問題は、それぞれの種類によって特徴のあるパターンを示すため、リアルタイムグラフを見ると、問題の存在や性質を知る手がかりを得ることができます。これによって、どのような種類のメモリ分析を実行すべきかを決定できます。

- 急激に上昇しているパターンのうち、ベースラインに戻らないか、予期されたようにガベージ コレクションに応答しないものは、メモリのリークを示している可能性があります。メモリ リーク分析を実行してください。
- ベースラインに戻らないパターンのうち、メモリの使用が定期的に上昇しているものは、アプリケーションの実行中に多くのオブジェクトが作成されていることを示します。一時オブジェクト分析を実行してください。
- アプリケーションがマネージ ヒープに予約されているシステム メモリのほとんどすべてを絶えず消費しており、その量が、ターゲット ユーザーのシステムに予期されているリソースと比較して大きい場合には、アプリケーションの全体的なメモリ フットプリントを削減することをお勧めします。このような場合は、RAM フットプリントを実行します。

動的クラス リスト

クラス リストには、プロファイルされているクラスのうち、ほとんどのメモリを消費している上位 20 のクラスが示されます。リストは、メモリ分析を有効にしてアプリケーションを実行している間、動的に更新されます。クラス リストを使用して、どのクラスがメモリ消費の増加またはオブジェクト作成の増加に関連しているかを観察します。リストはリアル タイムで更新されるため、アプリケーションの実行中に、問題を引き起こしている領域を特定できる可能性があります。

クラス リストで使用可能なカラムは以下の通りです。データを示すカラムは、**DevPartner**分析表示オプションで設定したオプションに応じて、バイト、キロバイト、またはメガバイトの単位で表示されます。

- クラス リストのソート順序を変更するには、**[**上位**20**クラス**]**リストのカラム見出しを選択します。
- ソース コードにアクセス可能なクラスのみをクラス リストに表示するには、リスト内を右クリックし、コンテキスト メニューから**[**ソースのある上位**20**クラスを表示**]**を選択します。

[ソースのある上位**20**クラスを表示**]**オプションを使用してクラス リストを表示すると、配列要素タイプがソース コード内にある場合、配列クラスがリストに表示されます。

表5-1. メモリ分析の動的クラス リスト内のカラム見出し

カラム	表示データ
クラス名	クラスの名前
名前空間	クラスの名前空間
インスタンスの数	現在メモリに存在するこのクラスのオブジェクト数
サイズ（バイト）	このクラスのインスタンスによって使用されるメモリの量 一時オブジェクト分析とRAMフットプリント分析でのデフォルトのソート
追跡されたインスタンスの数（メモリ リーク分析のみで表示）	現在メモリに存在するこのクラスの追跡オブジェクト数 メモリ リーク分析でのデフォルトのソート
追跡されたサイズ（単位）（メモリ リーク分析のみで表示）	現在メモリに存在するこのクラスの追跡オブジェクトすべてによって使用されているメモリ量

追跡オブジェクトは、ユーザーが追跡の開始／停止をクリックして追跡を開始したあと、再びクリックして追跡を停止するまでに割り当てられるオブジェクトです。

DevPartnerメモリ分析セッション コントロール ウィンドウ

セッション コントロール ウィンドウでは、データの収集と表示をインタラクティブに制御する方法がいくつかあります。

表5-2. メモリ分析のセッション コントロール ウィンドウの機能

セッション コントロール	説明
プロセス	タブ付き領域の右上にあるプロセスのリストを使用して、監視するプロセスを選択します。プロファイル対象として設定されたプロセスの実行が新たに開始されると、そのプロセスがリストに追加されます。デフォルトの選択は、開始プロセスです。
追跡の開始／停止  （メモリ リークのみ）	メモリ割り当ての追跡を開始または停止し（切り替え）ます。このボタンをクリックすると、グラフの色が変化し、追跡の対象となっている部分が示されます。

表5-2. メモリ分析のセッション コントロール ウィンドウの機能

セッション コントロール	説明
メモリをすべてクリア  （一時オブジェクトのみ）	この時点までに収集したメモリ データをすべてクリアします。ガベージ コレクションには影響を与えません。
ガベージ コレクションを実行 	アクティブなプロセスでガベージ コレクションを強制的に実行します。
表示を中止 	表示を停止します（切り替えます）が、データの収集は停止しません。このボタンを再びクリックすると、現在のメモリの使用状況を示すグラフ表示が再描画されます。

セッション結果の表示

アプリケーションの実行中に、該当する[表示]ボタンをクリックすると、メモリの使用状況のスナップショットをキャプチャできます。これによって、メモリの使用状況データを含むセッション ファイルが作成されます。アプリケーションの実行中は、必要なだけの数のセッション ファイルを作成できます。スナップショットをキャプチャしても、データ収集は停止されません。

表5-3. メモリ分析のセッション結果の表示のスナップショット コマンド

スナップショット コマンド	説明
メモリ リークを表示	アクティブなプロセスでガベージ コレクションを実行し、メモリ リーク データの詳細を示すセッション ファイルを開きます。
一時オブジェクトを表示	詳細な一時オブジェクト データを表示するセッション ファイルを作成します。アクティブなプロセスに対してガベージ コレクションは実行されません。
RAMフットプリントを表示	ガベージ コレクションを実行し、RAMフットプリント データの詳細を示すセッション ファイルを開きます。

未保存のセッション ファイルは、作成されたあとに自動的にVisual Studioで開かれます。これらのファイルを保存すると、すべてのセッション ファイルはアクティブなソリューションの一部となります。これらのファイルは、[ソリューション エクスプローラ]ペインのDevPartner Studio仮想フォルダに表示されます。

セッション ファイルは最初は結果サマリの形式で表示されます。結果サマリを使用すると、セッション データにドリルダウンし、ソース コード内の問題の領域を特定できます。

セッション ファイル統合

アプリケーションの実行が停止されると、メモリ分析セッションの結果がVisual Studioのセッション ウィンドウに表示されます。収集されたデータは、拡張子が**.dpmem**のメモリ分析セッション ファイルに保存されます。

セッション ファイルは、DevPartner Studio フォルダに自動的に追加されます。このフォルダは、アクティブなソリューションのソリューション エクスプローラで表示できます。既存のメモリ分析セッション ファイルを参照するには、ソリューション エクスプローラでファイルをダブルクリックします。

開発環境内で、セッション ウィンドウを使用して結果を分析できます。データにドリルダウンしてオブジェクト参照やオブジェクトを割り当てたメソッドのコール関係を確認し、特定のメソッドのソース コードに移動して、任意のメソッドのソース コードを Visual Studio で開いて編集します。

オブジェクト参照グラフを使用する

メモリに残っているオブジェクトを分析するときは、これらのオブジェクトがガベージ コレクタによって収集されない原因を理解する必要があります。オブジェクト参照グラフは、選択されているオブジェクトと、そのオブジェクトをアクティブに保っているガベージ コレクション ルート間のオブジェクトの完全なチェーンを示します。

オブジェクト参照グラフには、すべての参照元オブジェクトが表示されるのではなく、ガベージ コレクション ルートを指している参照元オブジェクトのみが表示されます。完全な詳細を示すため、オブジェクト参照パス内のオブジェクトのうち、ガベージ コレクション ルートの最短パスにないものでも表示される場合があります。

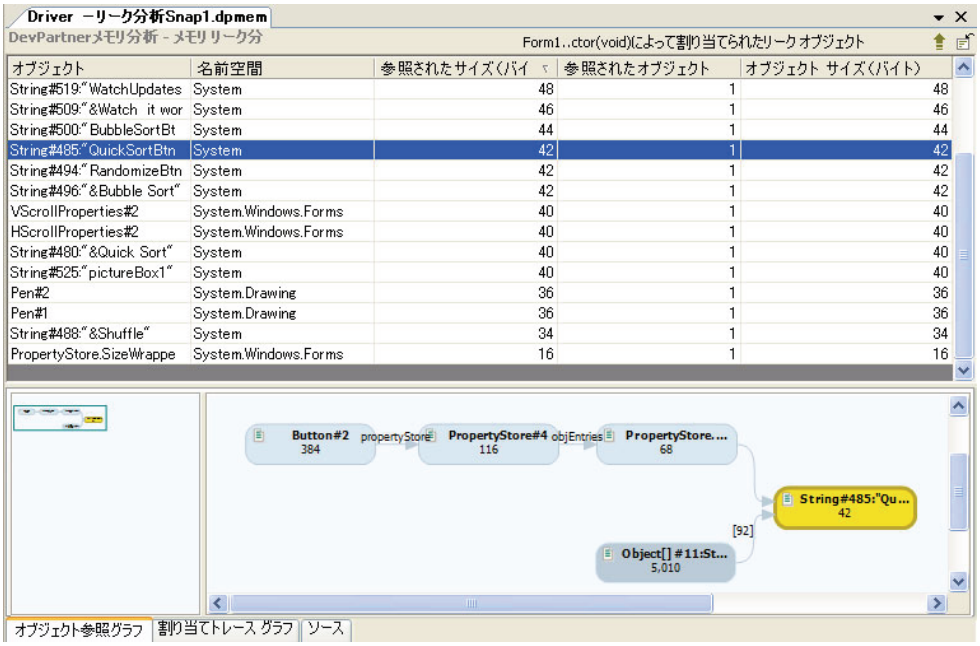


図5-13 メモリ分析のオブジェクト参照グラフ

オブジェクト参照グラフは、オブジェクト リストでオブジェクトを選択したときに自動的に再描画されます。

オブジェクト参照グラフは2つのフレームで構成されています。

- 左のフレームには、オブジェクト参照グラフの概要ペインが表示されます。概要ペインにはナビゲーション フレームが含まれています。これを使用すると、大きいグラフの各領域をすばやく検索して表示できます。
- 右のフレームには、[オブジェクト リスト]で選択したオブジェクトのオブジェクト参照関係が表示されます。

黄色で強調表示されているノードは、選択されているオブジェクトを表します。ノード上の数値データは、コンテキストによって、リークしているサイズか参照されているサイズを示します。オブジェクト参照パスは、参照の順番を表す矢印付きの線で示されます。接続線上のラベルは、参照を保持しているメンバー変数を示します。

コール グラフを使用して実行パスを特定する

コール グラフは、以下の2つのフレームで構成されます。

- 左側のフレームには、コール グラフの概要ペインが表示されます。これは、大きいグラフ内を移動する場合に役立ちます。概要内のナビゲーション フレームを移動すると、右側のフレーム内のビューが動的に変化します。
- 右側のフレームには、コール グラフが表示されます。メソッドはノードとして示されます。ノード間のリンクは呼び出し関係を表します。ノードを拡張すると、プログラムの実行順が表示されます。

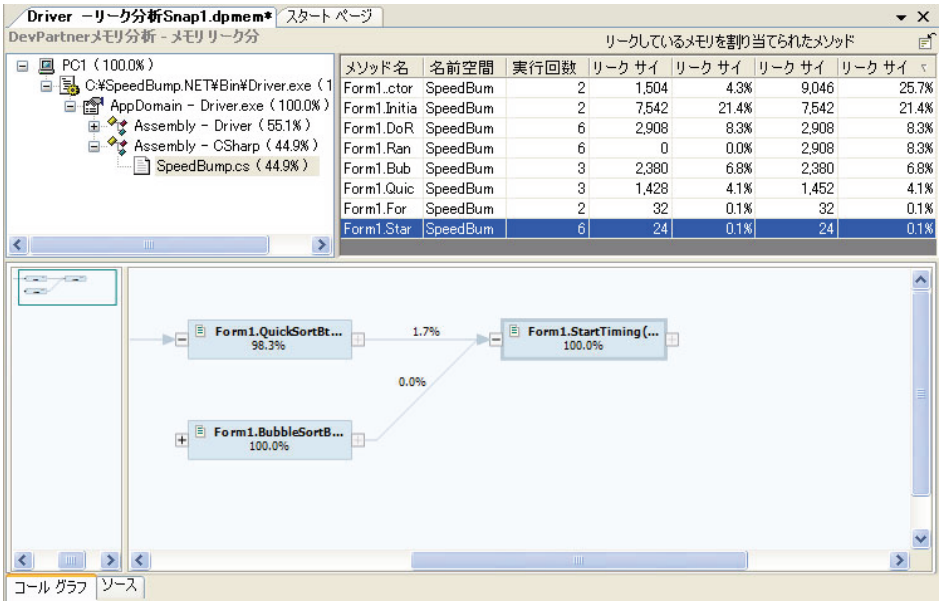


図5-14 メモリ分析のコール グラフ

コール グラフは、左から右に表記されます。コール グラフに最初に示される最初のノードは基本ノードです。これは、[メソッド リスト]内で選択されているメソッドを表します。選択されているノードの左にあるノードは、上位ノード、右にあるノードは下位ノードと呼ばれます。ノードの上半分はノード名を示します。

ノードの上半分には、そのノードによって表示されている関数またはメソッドの名前である、ノード名が表示されます。ノード値の下半分には、ノードに関連付けられた比率値である、ノード値が表示されます。この値は、ノード（およびその下位ノードすべて）によって使用されている合計メモリのうち、そのノードが使用しているメモリの割合を示します。

ノードの左右にある小さな四角形は、リンクと呼ばれます。エッジは、メソッド コールまたはメソッドの呼び出しを表します。ノードを結び付けている線上の割合はリンク値と呼ばれ、リンクに関連付けられている割合の値を示します。リンク値は、上位ノードによって使用されている合計メモリのうち、その下位ノード（さらにその下位のノード）が使用している割合を示します。

上位 ／下位ノードが関連付けられていないノードは、「終端ノード」と呼ばれます。終端ノードは、メソッド コールの順序の開始または終了にある、実行パスの先端を表します。

一時オブジェクトのメソッド名のリストと、それに関連付けられたコール グラフを表示するには、[最も多くのメモリを割り当てるエントリ ポイント]グラフまたは[最も多くのメモリを使用するメソッド]グラフの[完全な詳細の表示]をクリックします。

メソッド名のリストとそれに関連付けられたコール グラフが表示されると、メソッド名リストでメソッドを選択して、そのコール グラフを表示することができます。メソッドのソースコードを表示するには、メソッドを表すノードを選択し、[コール グラフ]ウィンドウの一番下にある[ソース]タブをクリックします。

クリティカル パス

DevPartnerでは、コール グラフを表示すると、選択したメソッドとそのすべての下位メソッドに関するクリティカル パスが自動的に計算されます。クリティカル パスは、最大の累積メモリを割り当てた下位メソッド コールのシーケンスです。クリティカル パスは、金色の太線で強調表示されます。

割り当てトレース グラフを使用する

割り当てトレース グラフには、メモリを割り当てたメソッド コールが示されます。割り当てトレース グラフは、RAM フットプリント セッション ファイルとメモリ リーク セッション ファイルで使用できます。これは、オブジェクト リストのあるどのビューでも表示できます。

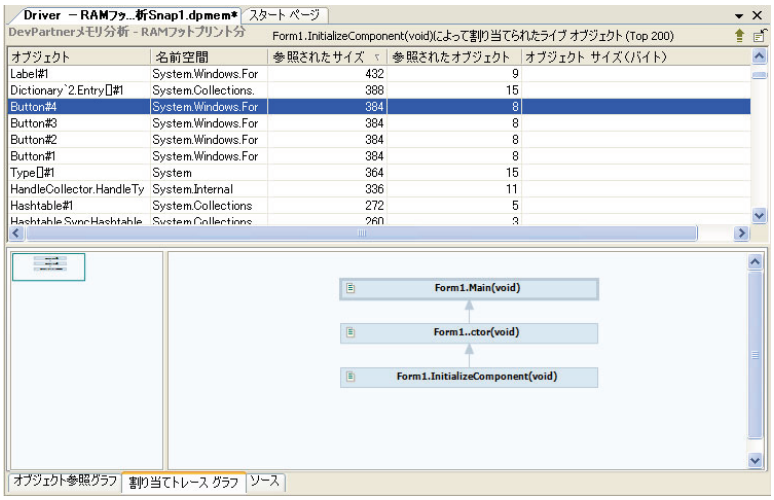


図 5-15 メモリ分析の割り当てトレース グラフ

オブジェクト リストおよび関連する割り当てトレース グラフを表示するには、以下のいずれかの操作を実行します。

- メモリ分析結果サマリで、[最も多くリークしているメモリを参照するオブジェクト] (メモリ リーク) または [最も多く割り当てられたメモリを参照するオブジェクト] (RAM フットプリント) の下にある [完全な詳細の表示] をクリックします。
- メモリ分析結果サマリの[最も多くリークしているメモリを割り当てるメソッド] (メモリ リーク)または [最も多くのメモリを割り当てるメソッド] (RAM フットプリント)ビューからドリルダウンします。

オブジェクトの割り当てトレース グラフを表示するには、以下のいずれかの操作を実行します。

- [オブジェクト リスト] 内のオブジェクトを選択し、セッション ファイル ウィンドウの下部にある [割り当てトレース グラフ] タブをクリックします。
- オブジェクト リスト内のオブジェクトを右クリックし、コンテキスト メニューの [割り当てトレース グラフの表示] を選択します。

選択したオブジェクトの割り当てトレース グラフが再描画されます。

割り当てトレース グラフ内のノードのソース コードを表示して編集するには、ノードを右クリックし、コンテキスト メニューの **[** ソースの編集 **]** を選択します。

ソース コードを表示および編集する

[ソース **]** タブを選択すると、プロファイルされているアプリケーションのソース コードが表示されます。

[ソース **]** タブ ビューには、DevPartner メモリ分析セッション ウィンドウの数多くのポイントから、コンテキスト メニューを経由して、またはセッション ウィンドウの一番下にある **[** ソース **]** タブをクリックすることによってアクセスできます。**[** ソース **]** タブには、ソース コードの他に、ソース コードの各行に関するデータが含まれています。**[** ソース **]** タブに表示されるデータは、実行するメモリ分析のタイプと、データ カラム表示の選択によって異なります。

ソース コードに関するデータを表示するだけでなく、Visual Studio エディタ内のソース コードに直接移動することもできます。移動するには、DevPartner メモリ分析コンテキスト メニューのいずれかから **[** ソースの編集 **]** を選択します。ソース ファイルが開き、**[** ソースを編集 **]** コマンドの実行元である、**[** ソース **]** タブ内のオブジェクト ノード、メソッド ノード、またはコードの行に対応する行を編集できるようになります。

メモ : ソース コードが表示されなかったり文字化けしている場合、ソース ファイルのエンコードが正しく認識されていない可能性があります。エンコードがわかっている場合は、ソース ペインを右クリックし、コンテキスト メニューから **[** エンコード **...]** を選択します。ダイアログで正しいエンコードを選択し、**[OK]** をクリックしてソース ファイルを表示してください。このコンテキスト メニューから、他のソース コードに切り替えることもできます。

[ソース **]** タブは、アプリケーション ソース コードのビューで構成され、アプリケーションが使用するソース メソッドに関する情報のデータ カラムが含まれています。表示されるデータ カラムは、**[** ソース **]** タブのコンテキストに合わせてカスタマイズされています。メモリ リーク分析、一時オブジェクト分析、RAM フットプリント分析の各セッションには、異なるセットのデータ カラムが用意されています。

[ソース **]** タブをナビゲートする

セッション ウィンドウのオブジェクトやメソッド（ソース コードがあるもの）から、**[** ソース **]** タブ上のソース コードの関連する行に直接移動できます。

- メモリ リーク、RAM フットプリント、一時オブジェクトの結果サマリからは、**[** すべての情報を表示 **]** をクリックしてセッション データにドリルダウンできます。
- セッション ウィンドウで、ウィンドウの下部にある **[** ソース **]** タブをクリックします。

ソース コードを表示する

以下の方法を使用して、メモリ分析で関連するソース コードを表示します。

表5-4. ソース コードを表示する

ビューまたはグラフ	ソース コードを表示する
オブジェクト ビュー	オブジェクト リスト内のオブジェクトを選択して、[ソース]タブをクリックします。
オブジェクト参照グラフまたは割り当てトレース グラフ	オブジェクト ノードを右クリックし、コンテキスト メニューの[ソースを表示]を選択します。
メソッド ビュー	メソッド リスト内のメソッドを選択して、[ソース]タブをクリックします。
コール グラフ	コール グラフ内のメソッド ノードを右クリックし、コンテキスト メニューの[ソースを表示]を選択します。

ソース コードを編集する

以下の方法を使用して、メモリ分析で関連するソース コードを編集します。

表5-5. ソース コードを編集する

グラフまたはリスト	ソース コードを編集する
オブジェクト参照グラフまたは割り当てトレース グラフ	オブジェクト ノードを右クリックし、コンテキスト メニューの[ソースの編集]を選択します。Visual Studioでソース ファイルが開き、編集できるようになります。
コール グラフ、オブジェクト リスト、またはメソッド リスト	オブジェクト リストやメソッド リスト内のメソッドを右クリックするか、またはコール グラフ内のノードを右クリックして、コンテキスト メニューの[ソースの編集]を選択します。Visual Studioでソース ファイルが開き、編集できるようになります。

[ソース]タブのデータ カラムをカスタマイズする

- [ソース]タブに表示されるデータ カラムを変更するには、カラム見出しを右クリックし、[項目の選択]を使用します。[ソース]タブのカラムをソートすることはできません。

ソース ファイルを変更する

- [ソース]タブのタイトル バーを右クリックし、[他のソース ファイルの選択]を使用して別のソース ファイルを選択します。これによって、ソース ファイルの新しいマッピングが作成されます。その他のソース パスには影響しません。

メモリに関する問題を特定する

以下のシナリオを考えてみます。

品質保証 (QA) チームが行った最初のテストの結果では、新しいマネージ アプリケーションは想定どおりの動作をしていました。しかし、その後のテストで QA が長時間のテストを実行した結果、アプリケーションを長時間実行するほどパフォーマンスが低下することが判明しました。

これは、想定していなかった動作です。最初にアプリケーションのどの部分を調査すればよいでしょうか。また、問題を発見したあと、どのように修正すればよいでしょうか。

アプリケーション内の問題を発見するには、DevPartner メモリ分析を有効にしてアプリケーションを実行します。メモリに関する問題がその時点で発生していなくても、DevPartner を使用することは有益です。開発プロセスの日常的な作業として、アプリケーションにおけるメモリの使用状況を DevPartner でテストしてください。

DevPartner を使用すると、アプリケーションにおけるメモリ リソースの使用状況を迅速に把握でき、現在発生している問題点や、問題である可能性がある点を明確化できます。

デバッグなしでメモリ分析セッションを開始したあと、セッション コントロール ウィンドウを使用して、プログラムにおけるメモリの使用状況を観察します。

リアルタイム グラフには、メモリの使用状況がビジュアルに表示されます。クラス リストは、プログラム実行中に動的に更新されて、最もメモリを使用しているクラスが表示されます。クラス リストを右クリックすると、上位 **20** クラスとソースのある上位 **20** クラスを切り替えることができます。

ユーザー インターフェイスのセッション コントロール ボタンを使用すると、マネージ ヒープのスナップショットを取ることができ、詳細な分析に使用できます。

メモリ分析セッションを実行する場合、以下の重要な 3 つの潜在的問題点のいずれかを選択して調査できます。

- メモリ リーク
- 一時オブジェクトの作成
- 全体的な RAM フットプリント

表 5-6. 現象と分析ツール

現象	分析ツール
時間と共にパフォーマンスが低下して、再起動すると回復する。アプリケーションを再起動するとパフォーマンスが改善されるが、時間がたつと再び低下する。	メモリ リーク
拡張性の問題、一時的にパフォーマンスが低下する。	一時オブジェクト メモリ リーク
パフォーマンスが低く、アプリケーションを再起動しても改善されない。	RAM フットプリント 一時オブジェクト

最初に、アプリケーションで発生している現象に応じた、適切なメモリ分析機能を選択します。最終的には、3 つすべてのタイプのメモリ分析を有効にしてアプリケーションを実行することもあります。分析によって問題が見つからない場合でも、徹底的に分析することによって、プログラムにおけるメモリ リソースの使用状況をよりよく理解できるようになります。

メモリ分析セッションを実行する

どのメモリ分析セッションを実行した場合でも、セッション コントロール ウィンドウには最初にリアルタイム グラフが表示されます。リアルタイム グラフには、アプリケーションにおけるメモリ リソースの使用状況がビジュアルに表示されます。アプリケーションの実行中に、グラフのパターンを観察します。メモリ使用シナリオは、それぞれの種類によって特徴のあるパターンを示すため、リアルタイム グラフを見ると、問題の存在や性質を知るための最初の手がかりを得ることができます。

ヒント：アプリケーション実行中のリアルタイム グラフの形状に注目してください。多くの場合、グラフのパターンを観察して解釈することによって、メモリに関する問題を即座に診断できます。

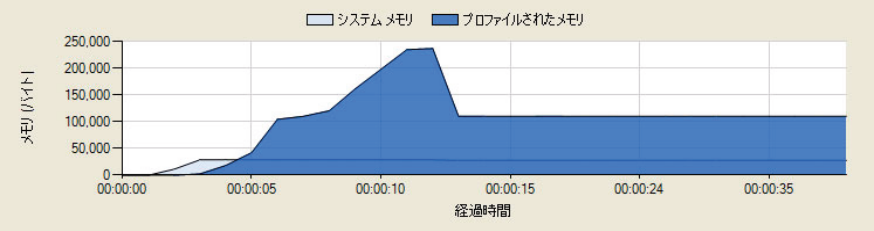


図5-16 メモリ分析セッション コントロール ウィンドウのリアルタイム グラフ

たとえば、図5-16のようにグラフが徐々に上昇し、ベースラインに戻らないパターンである場合は、アプリケーションでメモリがリークしている可能性があります。QA チームが発見したアプリケーションが徐々に遅くなるという問題は、メモリ リークが発生していると考えると整合性がありますが、リアルタイム グラフを観察することによってこの診断を確定できます。

グラフがベースラインに戻る場合で、メモリ使用量が定期的に急増している場合は、アプリケーション実行中にオブジェクトが数多く作成されています。このようなアプリケーションでは、割り当てられたメモリは確かに解放されていますが、負荷が高い状態での拡張性に問題がある可能性があります。

ユーザーや入力が増加するにつれて動作が遅くなるアプリケーションには、拡張性の問題がある可能性があります。ここでも、リアルタイム グラフは問題の性質を示しており、リアルタイム グラフを観察することで即座に正しい診断が可能になります。

一定のパターンが存在しないときでも、リアルタイムのグラフから重要な情報がもたらされることがあります。たとえば、アプリケーションがマネージ ヒープに割り当てられているメモリのほとんどすべてを絶えず消費しており、その量が、ターゲット ユーザーのシステムに予期されているリソースと比較して大きい場合には、アプリケーションの全体的なメモリ フットプリントを削減することをお勧めします。この章の次のセクションでは、これらのケースの詳細とアプリケーションのパフォーマンスに与える影響について説明します。

メモリ リークを検出する

アプリケーションで使用されるメモリの量は、アプリケーション プラットフォームのパフォーマンスに大きな影響を与えます。割り当てられるメモリの量が多いほど、アプリケーションの実行速度が遅くなり、拡張性も低くなります。

メモリ リーク（再利用されないメモリの割り当て）によって、アプリケーションの RAM フットプリントが大きく増加する可能性があります。自動ガベージ コレクションによって、作成したオブジェクトを明示的に解放する必要がなくなるため、従来の C++ で発生したようなメ

モリ「リーク」はなくなりますが、プログラムでオブジェクトへの参照を保持したまま再利用しない状態は発生します。

オブジェクトへの参照が存在するかぎり、参照されるオブジェクトはガベージ コレクタによってライブ オブジェクトであるとみなされます。ライブ オブジェクトのメモリは収集されません。この状態は、C++ におけるメモリ リークと同様に好ましくありません。このような参照は追跡することが難しいことがあるため、メモリ分析が役立ちます。

メモリ リーク分析を検討してください。

メモリ リーク分析セッションを実行する

準備、設定、実行セクション「すぐにメモリ分析を使用する」 136ページ)にも、メモリ リーク機能を使用する手順が説明されています。以下に、このプロセスを簡単にまとめます。

メモリ リーク機能を使用してメモリ リークを特定する

1. メモリ分析を有効にした状態でアプリケーションを起動します。セッション コントロール ウィンドウの [メモリ リーク] タブを使用します。
2. プログラムの関連機能を実行して、初期化を強制的に完了します。アプリケーションのウォームアップを実行することによって、分析から初期化時のメモリ割り当てが除外されます。
3. 追跡の開始 /停止をクリックして、新たに割り当てられるオブジェクトのみの追跡を開始します。
4. テストするプログラム機能を実行します。
5. ガベージ コレクションを実行をクリックして、アクティブなプロセスに対してガベージ コレクションを実行します。
6. 再度追跡の開始 /停止をクリックして、追跡期間を終了し、以降の新規メモリ割り当てを除外します。
7. クラス リスト内の [追跡されたインスタンスの数] カラムと [追跡されたサイズ] カラムをチェックします。オブジェクトが割り当てられているにもかかわらず、収集されていない場合は、[メモリ リークを表示] をクリックし、ガベージ コレクション後に残っている追跡オブジェクトを示すマネージ ヒープのビューをキャプチャします。

   [メモリ リークを表示] は、追跡の開始/停止を最初にクリックしたあとで表示されます。

DevPartner には、マネージ ヒープの状態を示すスナップショットが表示されます。このデータはメモリ リーク結果サマリとして表示されます。結果サマリ ページからは、メモリ使用状況データへのドリルダウン、問題の特定、およびソース コード内の問題のメソッドの検索ができます。

メモ： DevPartner がメモリ リーク セッションまたは RAM フットプリント セッションでほとんどのガベージ コレクション ルートを適切に特定できるようにするには、デバッグを実行せずにメモリ分析を選択して開始するようにしてください。[メモリ分析を選択して開始] (デバッグあり) を有効にした状態で起動したアプリケーションのメモリ リーク データや RAM フットプリント データを収集しようとすると、すべてのガベージ コレクション ルートが「識別できない GC ルート」としてセッション データに表示されます。

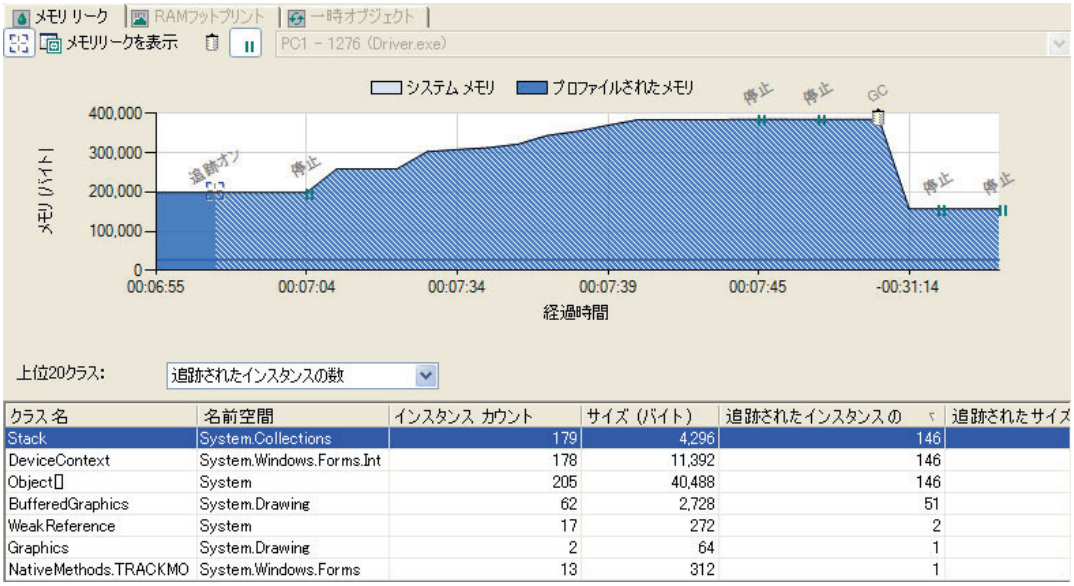
メモリ リーク分析結果を理解する

DevPartner メモリ リーク分析では、メモリ リークは、特定の期間にマネージ ヒープに割り当てられたオブジェクトであり、メモリ データを収集したときに解放されていないオブジェクトとして定義されます。メモリ リーク分析は、アプリケーション内で、解放されている必要があるメモリが保持されている場所を明確化する場合に役立ちます。この情報を使用して、このメモリが解放されるようにするためにコードをどのように変更するかを決定します。

メモリ リークを発見するには、DevPartner メモリ リーク分析機能を有効にしてアプリケーションを実行し、以前に割り当てられたオブジェクトが解放されるようにアプリケーションを実行します。

オブジェクトが解放されるはずであるにもかかわず、ガベージ コレクションを実行してもメモリ使用量が継続的に増加して減少しない場合　または想定どおりに減少しない場合)、アプリケーションでメモリ リークが発生している可能性があります。

たとえば、図 5-17 を参照してください。この図のリアルタイム グラフでは、メモリの使用量が増加して、ガベージ コレクション後もベースラインに戻らない状態が示されています。アプリケーションに属しているクラスの [追跡されたインスタンスの数] カラムを確認すると、ガベージ コレクションによって収集されていない追跡オブジェクトがいくつかあることがわかります。セッション コントロール ウィンドウの [追跡されたインスタンスの数] カラムで、収集されていないインスタンスの数を確認します。



クラス名	名前空間	インスタンス カウント	サイズ (バイト)	追跡されたインスタンスの	追跡されたサイズ
Stack	System.Collections	179	4,296	146	
DeviceContext	System.Windows.Forms.Int	178	11,392	146	
Object[]	System	205	40,488	146	
BufferedGraphics	System.Drawing	62	2,728	51	
WeakReference	System	17	272	2	
Graphics	System.Drawing	2	64	1	
NativeMethods.TRACKMO	System.Windows.Forms	13	312	1	

図 5-17 セッション コントロール ウィンドウのデータ表示

DevPartner によってリークの可能性が指摘されたら、DevPartner によって作成されるメモリ リーク結果サマリ　セッション ファイル）を使用して、リークしている場所を特定して、修正します。メモリ リーク分析結果サマリには、データにドリルダウンする以下の方法が用意されています。

- 最も多くリークしているメモリを参照するオブジェクト
- 最も多くリークしているメモリを割り当てるメソッド

各グラフには、リークしているメモリに関連している上位5つのオブジェクトまたはメソッドが表示されます。上位5つのオブジェクトまたはメソッドの詳細を確認するには、グラフの [すべての情報を表示] をクリックします。

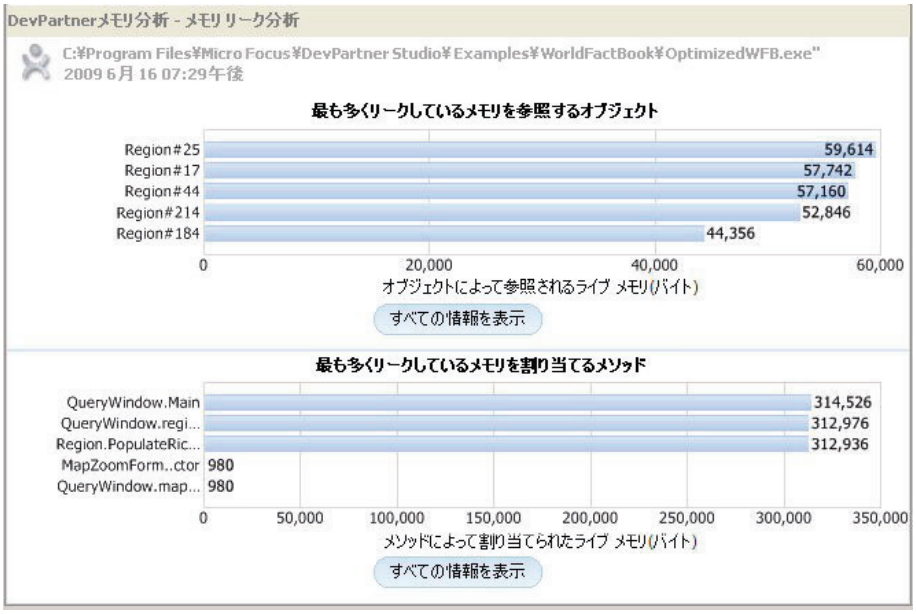


図5-18 [メモリリークを表示] をクリックすると表示される結果サマリ

調査の開始点は、解決する必要がある問題や、問題に対して使用するアプローチに応じて異なります。例：

- 一部の特定のオブジェクトでリークが発生していることに気付いた場合は、[最も多くリークしているメモリを参照するオブジェクト] グラフを使用して、リークしているオブジェクトがどのオブジェクトによって保持されているかをすばやく確認できます。
- 割り当てを行うメソッドのソース コードをよく理解しており、ソース コードを調査することによってリークしているオブジェクトをクリアする必要があるかどうかを判断できる場合は、[最も多くリークしているメモリを割り当てるメソッド] グラフから開始できます。

オブジェクトのグラフとメソッドのグラフのいずれからでも、データを異なる観点から示すビューにすばやく切り替えることができます。

[最も多くリークしているメモリを参照するオブジェクト] のすべての情報を表示している場合は、以下のビューを選択できます。

- オブジェクト参照グラフ
- 割り当てトレース グラフ
- ソース

[最も多くリークしているメモリを割り当てるメソッド] のすべての情報を表示している場合は、以下のビューを選択できます。

- コール グラフ
- ソース

以下の例では、開始点として [最も多くリークしているメモリを参照するオブジェクト] を使用しています。

最も多くリークしているメモリを参照するオブジェクト

以下の例は、一部のオブジェクトによってメモリ リークが発生しているケースについて示しています。また、この例では、問題を診断するための他のアプローチについても説明します。

ガベージ コレクタでは、あるオブジェクトへの参照が少なくとも1つ存在している場合、そのオブジェクトはクリアできません。アプリケーションを実行すると、オブジェクトが作成されます。一部のオブジェクトは、プログラムの実行期間全体にわたって必要です。これらは永続オブジェクト、またはロング ライブ オブジェクトと呼ばれます。ただし、ほとんどのオブジェクトは、他のオブジェクトによって参照されなくなった場合にガベージ コレクションの対象となる必要があります。

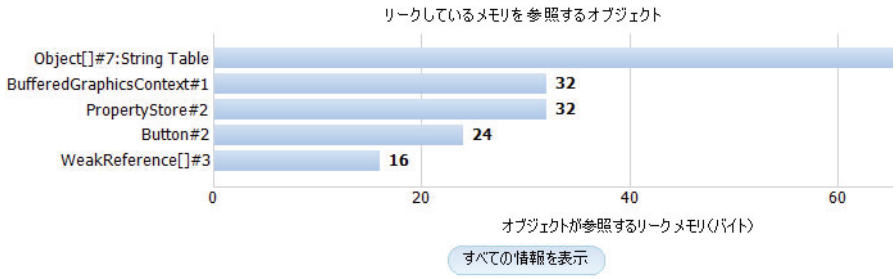


図5-19 最も多くリークしているメモリを参照しているオブジェクト

図5-19のグラフには、最も多くリークしているメモリへの参照を保持する上位5つのオブジェクトが示されています。これらのオブジェクトが原因で、リークしているオブジェクトが解放されなくなっています。最大のメモリがリークしている参照元オブジェクトがグラフの一番上に表示されています。データによって、特定のオブジェクトによってメモリ リークが発生していることがわかります。このグラフを開始点として使用してセッション データにドリルダウンし、メモリ リークの発生場所を特定します。

[すべての情報を表示] 棒グラフの下、図5-20（163ページ）を参照）をクリックすると、リークしているメモリを参照するオブジェクトの詳細が表示されます。

この表示の上部のパネルには、リークしているメモリを参照するすべてのオブジェクトのリストが表示されます（図5-20の図を参照）。このリストには、元の棒グラフに表示されていた上位5つのオブジェクトと、より少量のリークしているメモリを参照するその他のオブジェクトが含まれています。

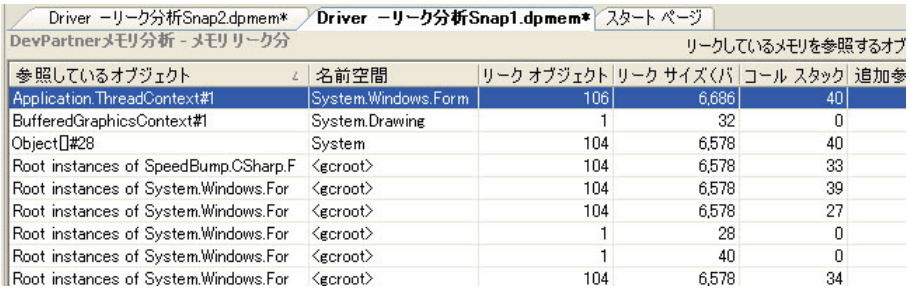


図5-20 最も多くリークしているメモリを参照しているオブジェクトのリスト

デフォルト設定では、オブジェクトは[リーク サイズ] 選択されたオブジェクトによって参照されているリークしているオブジェクトの合計サイズ）カラムでソートされます。他の任意のカラムでリストをソートして、データのパターンを確認することもできます。リストの項目を右クリックして[このオブジェクトによって参照されるリーク オブジェクトを表示]を選択すると、実際にリークしているオブジェクトを確認できます。

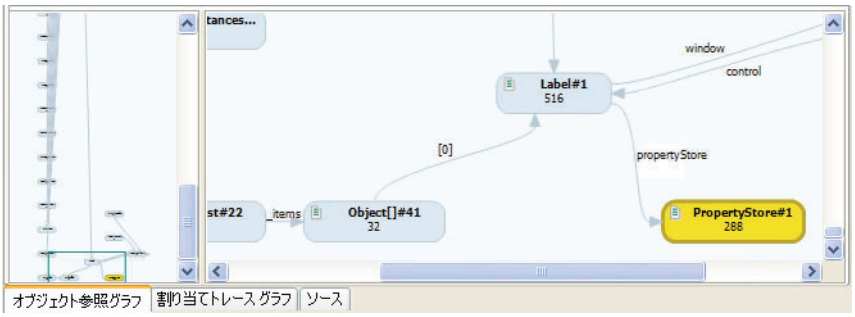


図5-21 オブジェクトがメモリ内に存在する理由を示すオブジェクト参照グラフ

調査対象の参照元オブジェクトを選択します。これらのオブジェクトをメモリに保持している参照のシーケンスについてすばやく理解できるようにすることが重要です。[オブジェクト参照グラフ]タブをクリックして、参照グラフを表示します。オブジェクト参照グラフには、選択されたオブジェクトがガベージ コレクタによってクリアされなかった理由が表示されます。このグラフは、選択されているオブジェクトおよび選択されているオブジェクトをアクティブに保っているガベージ コレクション ルート間のオブジェクトのチェーンを示します。

オブジェクトのリストを下にスクロールして、他のオブジェクトを調査します。オブジェクト参照グラフには、非常に単純なグラフも非常に複雑なグラフもあります。他のオブジェクトを調査することによって、小さいオブジェクトに対する数多くの参照が存在する、または大きなオブジェクトに対する少数の参照が存在するなどの状態を示す証拠を発見できる可能性があります。目的は、このグラフを使用して、参照元オブジェクトのチェーン内でリークを最も効率的に除去できるポイントを特定することにあります。

オブジェクト参照グラフに示される参照元オブジェクトのチェーンには、さまざまな複雑度のチェーンがあります。多くの場合、複数の参照元があり、グラフは非常に複雑になります。概要ペイン内のナビゲーション フレームをドラッグするか、またはノードをクリックして、詳細ペイン内に表示されるノードを変更します。分析で複雑なグラフが表示されたときは、ノードを右クリックして[すべての参照元を表示しない]を選択することによって、ビューを単純化できます。また、グラフ内でノードをドラッグして、見やすくすることもできます。

ノードを接続する矢印に表示されている**[elements]**などのラベルは、グラフの次のクラスで参照されているデータ メンバを表しています。角かっこで囲まれた数字は、配列を示しています。コードをよく理解している場合は、ラベルを参照することによって、潜在的問題点をすばやく特定できます。

また、ノードを右クリックし、[ソースの編集]を選択して、Visual Studio内に関連するソースコードを開いたり、[ソース]タブを選択して関連するソース コードを表示したりすることもできます。

プログラムをよりよく理解するために、グラフ内の各ノードのソースを順番に表示して、リークしたメモリが割り当てられたイベントを確認できます。たとえば、割り当てトレース グラフには、選択されているオブジェクトを割り当てた各メソッドの呼び出し元が表示されます。

リストのオブジェクトからソース コードに直接移動できます。実際に問題解決を行う場合は、解決する問題に適した方法、またはコードに適する方法を使用してドリルダウンする必要があります。

問題を解決するその他の方法

上記の例では、オブジェクト参照パスを使用してリークしている場所を特定する方法について説明しました。メモリがリークしている場所を特定する方法には、他の方法もあります。例:

- 割り当てトレース グラフを参照して、オブジェクトを割り当てたメソッドの呼び出し元を特定します。そこからソース コードに移動します。
- オブジェクトのリストから直接ソース コードに移動します。

[DevPartner メモリ分析 - メモリ リーク分析] サマリには、データのさまざまなビューが表示されます。最初に [最も多くリークしているメモリを参照するオブジェクト] を使用した例を説明しました。ただし、データの複雑度に応じて、または好みに応じて、[DevPartner メモリ分析 - メモリ リーク分析] サマリの以下に示すいずれかのグラフを使用して問題を調査することもできます。

最も多くリークしているメモリを割り当てるメソッド

以下のグラフは [DevPartner メモリ分析 - メモリ リーク分析] サマリの下半分に表示され、リークしたオブジェクトを割り当てた上位5つのメソッドが表示されます。[すべての情報を表示] をクリックすると、オブジェクトのリークが発生したすべてのメソッドのリストが表示され、[コール グラフ] ビューとメソッドのソース コードにアクセスできます　存在する場合)。

図 5-22 最も多くリークしているメモリを割り当てるメソッド

メソッド リストでメソッドを選択して、リークが発生したメソッドによって割り当てられたオブジェクトを確認します。また、メソッドのソース コードを表示することもできます。リークしているオブジェクトを割り当てた行、およびその行でリークしているオブジェクトの数とサイズの統計が表示されます。

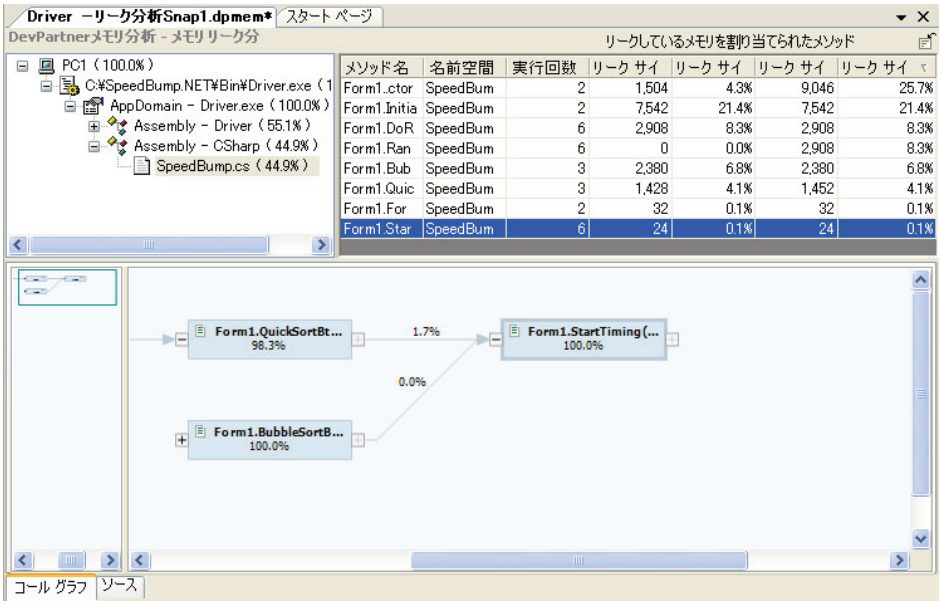


図 5-23 最も多くリークしているメモリを割り当てるメソッドの詳細

たとえば、以下の方法でデータにドリルダウンできます。

- メソッド リストのメソッドを右クリックします。
- 選択したメソッドから、そのメソッドによって割り当てられたオブジェクトのリストに移動するか、またはメソッドのコール グラフに移動します。
- オブジェクト リストのオブジェクトから、参照されているオブジェクトのリスト、オブジェクト参照グラフ、または割り当てトレース グラフを表示します。
- メソッド、コール グラフのノード、または割り当てトレース グラフのノードから、ソース コードと各行のオブジェクト割り当てデータを表示します。
- リストのメソッド、リストのオブジェクト、コール グラフのノード、オブジェクト参照グラフのノード、割り当てトレース グラフのノード、またはソース コードの行から、[ソースの編集] を選択して、ソースの適切な行を開いて編集します。

一時オブジェクト分析を使用して拡張性の問題を解決する

DevPartner でメモリ分析を行う場合、一時オブジェクト分析を使用して、拡張性の問題を診断および修正できます。

拡張性の問題の例

アプリケーションが通常は正常に動作していても、ユーザーが集中的に使用すると拡張性の問題が発生することがあります。クライアント /サーバー アプリケーションでは、ユーザー数が増えるとこのような問題が発生する可能性があります。スタンドアロン アプリケーションでは、数多くのテキスト処理や数学的計算が実行されたときにこのような問題が発生する可能性があります。このような問題は、「拡張性」の問題と呼ぶことができます。アプリケーションによって行われる処理の規模が拡大するにつれて、パフォーマンスが低下します。

考えられる原因：一時オブジェクト

考えられる原因の1つに、一時オブジェクトの過剰な作成が挙げられます。この場合は、オブジェクトの作成がパフォーマンスのボトルネックになります。このような問題は修正する必要があります。

マネージ Visual Studio プログラムにおいて、オブジェクトの作成と使用は重要です。ただし、一部のコーディング手法は、数多くのオブジェクトが作成されるという副作用を伴います。

このような問題の例として、**`String`** クラスによって作成されるオブジェクトなどのオブジェクトの作成があります。オブジェクトを作成したり、あとでこれらのオブジェクトを破棄したりする場合に、処理サイクルが必要になります。作成されるオブジェクトの数を減らすことができると、通常はパフォーマンスが向上します。

オブジェクトの生存期間

DevPartner では、コードによって割り当てられたオブジェクトを追跡し、これらのオブジェクトが収集されるのにかかる時間に基づいて、これらのオブジェクトを分類します。以下の3つの分類があります。

- ショート ライブ – オブジェクトが割り当てられたあとの最初のガベージ コレクションで収集されます (世代 0)。
- ミディアム ライブ – オブジェクトが割り当てられたあとの 2 回めのガベージ コレクションで収集されます (世代 1)。
- ロング ライブ – プログラム実行中に数多く (またはすべて) のガベージ コレクションが行われたあとも残ります (世代 2)。

メモ : Microsoft .NET Framework のガベージ コレクタでは、0、1、2の3世代がサポートされています。最後に実行されたガベージ コレクションのあとに割り当てられたオブジェクトは世代0です。割り当て後に1回ガベージ コレクションが行われたあとにも残っているオブジェクトは世代1のオブジェクトです。さらに1回以上のガベージ コレクションを実行しても収集されなかった世代1のオブジェクトは、世代2のオブジェクトとなります。

DevPartner では、ショート ライブとミディアム ライブのオブジェクト割り当てが一時オブジェクトのカテゴリに分類されます。

ミディアム ライブ オブジェクトがパフォーマンスに最も大きな影響を与え、ガベージ コレクタで必要以上の処理が行われる原因となります。個々のショート ライブ オブジェクトがガベージ コレクションに与える影響は小さいですが、オブジェクトのコンストラクタを呼び出すときにパフォーマンスの問題が発生します。ただし、ショート ライブ オブジェクトを大量に作成すると、ボトルネックとなり、メモリ不足が発生することがあります。

コードに拡張性の問題があることが判明した場合は、DevPartner を使用して、コードの実行中に使用されるメモリを監視します。セッション コントロール ウィンドウのリアルタイム グラフのパターンが上昇と下降を繰り返す波のようなパターンである場合は、アプリケーションで数多くの一時オブジェクトが作成されていることを示しています。この場合は、DevPartner を使用して、アプリケーションにおける一時オブジェクトの作成を分析します。

DevPartner では、一時オブジェクト分析の結果がエントリ ポイントとメソッド別に分類されます。どのような方法を使用してデータにドリルダウンした場合でも、一時オブジェクトによって消費されているメモリの量を確認でき、一時オブジェクトを割り当てているコードの行を特定できます。

一時オブジェクト分析セッションを実行する

アプリケーションで作成される一時オブジェクトによる問題を分析するには、以下の手順を実行します。

1. メモリ分析を有効にした状態でアプリケーションを起動します。セッション コントロール ウィンドウの **[**一時オブジェクト**]** タブを使用します。
2. アプリケーションを実行し、以下のいずれかを実行して、一時オブジェクトを最も多く割り当てたプログラムの部分を確認します。
   a. セッション コントロール ウィンドウでシステムのガベージ コレクション（下降のパターン）を観察する場合は、**[**一時オブジェクトを表示**]** をクリックします。
   b. **[**ガベージ コレクションを強制します**]** アイコンをクリックしてから **[**一時オブジェクトを表示**]** をクリックします。
   c. DevPartner により、ガベージ コレクションが強制され、一時オブジェクト セッション ファイルが作成されます。

メモ： デバッガでアプリケーションを実行している場合、デバッガを使用してアプリケーションを停止しないでください。このアクションに対してセッション ファイルは生成されません。アプリケーションを通常の方法で停止し、セッション ファイルを生成してください。

3. アプリケーションの特定の部分に関する一時オブジェクトの割り当て動作を確認するには、**[**メモリをすべてクリア**]** をクリックし、それまでに収集されている一時オブジェクト割り当てデータをクリアします。次に、アプリケーションの関連する部分を実行し、ガベージ コレクションを強制し、**[**一時オブジェクトを表示**]** をクリックします。

   DevPartner では、メモリ分析が有効な状態で実行されているアプリケーションが終了したときに、一時オブジェクトのセッション ファイルが常に作成されます。さらに、セッション コントロール ファイルまたはセッション コントロール API で実行されるスナップショット アクションに応答して、一時オブジェクト セッション ファイルが作成されます。

DevPartner には、マネージ ヒープの状態を示すスナップショットが表示されます。このデータは一時オブジェクト結果サマリとして表示されます。結果サマリ ページからは、オブジェクト作成データへのドリルダウン、問題の特定、およびソース コード内の問題のメソッドの検索ができます。

拡張性の問題を特定する

DevPartner を使用すると、潜在的な問題点を検出し、アプリケーションにおいて一時オブジェクトが使用されている場所にドリルダウンして、問題の特定およびコードの全体的な品質の改善を行うことができます。

リアルタイム グラフ

リアルタイム グラフに表示される高レベルのビューを使用して、問題点を特定できます。

アプリケーションによってショート ライブ オブジェクトやミディアム ライブ オブジェクトが数多く作成される場合は、プロファイルされたメモリのピークがリアルタイム グラフに出現し、その後ガベージ コレクションが実行されると消滅します。ガベージ コレクション後に再度同じ機能を実行すると、新たに作成された一連の一時オブジェクトによるピークが再度現れます。

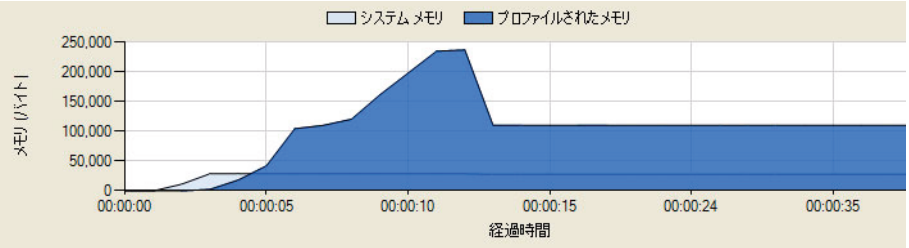


図 5-24 一時オブジェクトが過剰に作成されていることを示唆するリアルタイム グラフ

最も多くオブジェクトを作成するクラスが、プロファイルされたクラスのリストに表示されます。このリストは、[サイズ] カラムでソートされています。ここでは、最も一時メモリを消費しているオブジェクトを持つクラスが表示されます。また、[インスタンス カウント] には、各クラスで作成されたオブジェクトのインスタンス数が表示されます。

図 5-24 に、過剰な一時オブジェクトの作成を示唆するリアルタイム グラフを示します。グラフが急激に上昇している箇所は、アプリケーションによって数多くのオブジェクトが作成されている時点を示しています。オブジェクトを過剰に作成すると、マネージ アプリケーション、特にサーバー アプリケーションで、重大なパフォーマンスの問題や拡張性の問題が発生することがあります。拡張性が問題ではない場合でも、ショート ライブ オブジェクトを多く割り当てるメソッドは、修正の簡単なパフォーマンスの問題を引き起こしていることがよくあります。

一時オブジェクトを表示する

[一時オブジェクトを表示] をクリックして、特定の時点におけるアプリケーションのデータを収集します。一時オブジェクト分析のページが表示されます。このページでは、最も多く一時オブジェクトを作成したエントリ ポイントまたはメソッド別にデータが分類されています。

エントリ ポイントとは、除外されている（システムまたはサードパーティ製の）コードによって呼び出される、プロファイルされたメソッドです。アプリケーションを実行すると、プロファイルされている、ユーザーコード メソッドへの最初のコール以降の監視が開始されます（ユーザーコード メソッドとは、アプリケーションのソース コード内にあるメソッドです）。これは、エントリ ポイントと呼ばれます。そのポイントからの他のユーザーコード メソッドのすべてのコールは、そのエントリ ポイントの一部とみなされます。

他のユーザーコード メソッドからのみ呼び出されるメソッドはエントリ ポイントではありません。ただし、そのようなメソッドで大量の一時メモリが使用されることがあります。結果サマリの 2 つめのグラフには、エントリ ポイント メソッド以外のメソッドも含め、大量の一時メモリを割り当てるメソッドが示されます。このように、エントリ ポイントから呼び出される下位メソッドがアプリケーション内で多くのメモリを割り当てるメソッドである場合は、呼び出し元のエントリ ポイント メソッドのコール グラフをたどることなく、[最も多くのメモリを使用するメソッド] でそのメソッドを検索できます。

結果サマリ ビューからデータにドリルダウンして、これらのメソッドによって割り当てられたオブジェクトが消費しているメモリの量を把握し、ショート ライブ オブジェクトやミディアム ライブ オブジェクトが作成されているコードの行を特定できます。

一時オブジェクト データを分析する

いずれかのグラフの下の [すべての情報を表示] をクリックすると、アプリケーション内の一時メモリを割り当てたすべてのエントリ ポイントまたはすべてのメソッドの詳細ビューが表示されます。すべてのメソッドのリスト以外に、このビューには [コール グラフ] タブと [ソース] タブが含まれています。

メソッド リスト内のデータ カラムには、アプリケーションのメソッドについて、セッション コントロール ウィンドウのプロファイルされているクラスのリストよりも広範な情報が表示されます。

コール グラフ

エントリ ポイント リストのエントリ ポイントまたはメソッド リストのメソッドをクリックして、メソッドのコール グラフを表示します。コール グラフには、選択したメソッドとその下位メソッドが表示されて、クリティカル パスが金色の太線で強調表示されます。クリティカル パスは、選択したメソッドに対して最大の累積メモリを割り当てた下位メソッド コールのシーケンスです。

コール グラフでは、メソッドはノードとして表示されます。各ノードには、そのメソッドによって割り当てられたメモリについてのデータが表示されます。また、ノード間のリンクには、グラフのその分岐によって割り当てられたメモリについてのデータが表示されます。このデータは、割り当てられたメモリの割合として表示されます。

- ノード – メソッドの本体自体によって割り当てられたメモリの割合
- リンク – 該当する分岐で実行された下位メソッドによって割り当てられたメモリの割合

DevPartner ではこのように、アプリケーションによって作成される一時オブジェクトがどのメソッドによって作成されるかのみではなく、実行パスのどこで具体的に割り当てられるかが示されます。

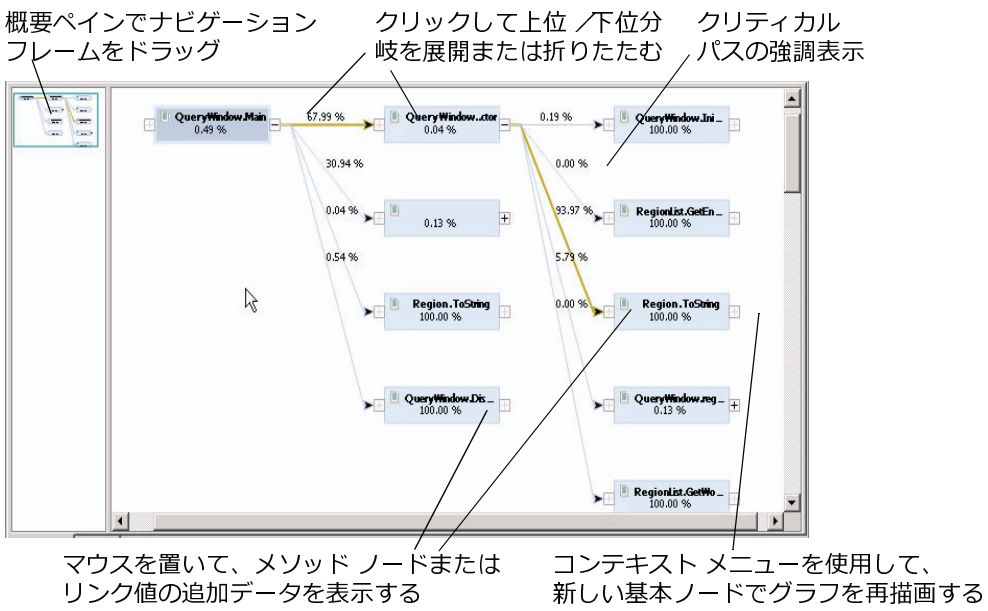


図 5-25 クリティカル パスが表示されたエントリ ポイント メソッドのコール グラフ

コール グラフ内の任意のノードを右クリックして、以下の操作を実行できます。

- 選択したノードでコール グラフを再描画します。
- 選択したノードのソース コードを表示します。
- 選択したノードのソース コードを編集します。

ソース ビュー

[最も多くのメモリを割り当てるエントリ ポイント] または [最も多くのメモリを使用するメソッド] メソッド リスト内のエントリ ポイントまたはメソッドのソース コードを表示する場合は、選択したメソッドに対して [ソース] タブ ビューが開きます。ソース コード以外にも、アプリケーション コード内の各行によって割り当てられたメモリの詳細情報が表示されます。詳細情報には、その行が実行された頻度、その行で割り当てられたショート ライブ オブジェクト、ミディアム ライブ オブジェクト、ロング ライブ オブジェクトの数（下位オブジェクトを含む数または含まない数）、およびこれらのオブジェクトの蓄積サイズ（メモリ負荷）が含まれています。

結果を解釈して拡張性の問題を修正する

以下に、メモリ分析結果を解釈して、メモリ関連の拡張性の問題を修正するための方法をいくつか示します。

- 一時オブジェクト分析のページを参照して、エントリ ポイント メソッドまたは非エントリ ポイント メソッドのどちらによって最も多く一時メモリが消費されているかを特定します。
- エントリ ポイントによって最も多くの一時オブジェクトが消費されている場合は、[最も多くのメモリを割り当てるエントリ ポイント] ビューを使用してドリルダウンし、エントリ ポイントの実行パス内のどのメソッドで最も多くの一時領域が必要となっており、修正または呼び出し頻度を少なくする必要があるかを特定します。
- 非エントリ ポイント メソッドによって最も多くの一時オブジェクトが消費されている場合は、[最も多くのメモリを使用するメソッド] ビューを使用してドリルダウンし、コードのどの部分を修正するかを特定します。
- ショート ライブ オブジェクトとミディアム ライブ オブジェクトの数、およびそれらのオブジェクトによって消費されている一時領域のサイズを比較します。この情報を使用して、コードのどの部分を修正するかを特定します。
- ショート ライブ オブジェクトとミディアム ライブ オブジェクトが同程度のサイズの一時領域を消費している場合は、パフォーマンス分析を実行して、コンストラクタで一時オブジェクトの作成にかかっている時間を測定します。
- コール グラフを使用して、一時メモリを割り当てるメソッド間の関係を理解します。異なるメソッドの特性（消費メモリの割合、実際に使用されているバイト数、作成された一時オブジェクトの数）を調査します。この情報を使用して、どのメソッドを修正するかを特定します。
- [ソース] タブを使用して、コード内で一時オブジェクトが割り当てられている具体的な行を特定します。作成されるオブジェクトの種類やサイズ、およびその行が実行される頻度を調査します。この情報を使用して、オブジェクトをより効率的に使用できる方法を特定します。

RAM フットプリントを使用してパフォーマンスを向上させる

一部のマネージ アプリケーションでは、実行中に数百 MB の RAM が消費されます。この章では、メモリ リークの問題（アプリケーション実行時にメモリが徐々に消費され、最終的にはヒープが使い果たされる可能性がある）や、一時オブジェクトが過剰に作成されることが原因で、定期的にメモリ使用量が急激に上昇する問題（拡張性の問題を発生させる可能性がある）などの、特定のメモリに関する問題について検証します。これらの問題は、アプリケー

ションのメモリ使用に悪影響を与えます。また、これらの問題は、アプリケーションのメモリ フットプリントを増加させます。アプリケーションが適切に動作しており、これらのエラーが発生していない場合でも、特にさまざまなエンド ユーザー環境で実行された場合に実行速度が遅くなることもあります。

パフォーマンスが低下する原因の1つに、アプリケーション実行時にメモリを過剰に使用していることが考えられます。過剰とはどの程度の量を指すのでしょうか。その基準は、アプリケーションが使用される環境 (ハードウェアやソフトウェア) に応じて異なります。エンド ユーザー環境をよく理解している場合でも、環境は変化することがあります。たとえば、アプリケーション実行時にエンド ユーザーが同時に複数の他のアプリケーションを実行して、これらのアプリケーション間でメモリ リソースの競合が発生することがあります。また、アプリケーションの新しいバージョンをリリースするたびに、ユーザーに対してハードウェアのアップグレードを強制することもできません。このような理由から、アプリケーションのメモリ フットプリントを小さく保つことは非常に重要です。

より具体的には、ここでのメモリ使用とは、全体的なメモリの使用ではなく、RAM フットプリントを指しています。アプリケーションのパフォーマンス、およびエンド ユーザーのアプリケーションに対する印象に最も大きな影響を与える要因は、アプリケーションをオペレーティング システムの仮想メモリ システムに依存させることです。マネージ オブジェクトを仮想メモリにページングすることは、アプリケーションのパフォーマンスを大きく低下させます。

アプリケーションによるRAMリソースの使用を最適化するために何ができるでしょうか。DevPartnerには、メモリ分析機能の一部としてRAM フットプリント分析が用意されています。アプリケーションの開発中に、定期的にRAM フットプリント分析を実行してください。アプリケーションによるRAMリソースの使用は、アプリケーションの設計とアーキテクチャに大きく依存しています。アプリケーションのベータ版リリースの段階よりも、開発プロセスの初期の方が、機能をより簡単に再設計できます。

RAM フットプリントを測定する

DevPartnerを使用することで、RAMの消費に最も大きな影響を与える領域で集中的にパフォーマンスを調整できます。RAM フットプリント分析を有効にしてアプリケーションを実行すると、以下のことが可能になります。

ヒント : RAM フットプリントの測定に関する手順については、DevPartner Studioのオンライン ヘルプを参照してください。

- アプリケーションのRAM消費のリアルタイム グラフを表示して、最も多いバイト数のメモリを消費するプロファイルされているクラスのリアルタイム リストを表示します。
- 最もメモリを使用しているオブジェクトを調査するために使用する、マネージ ヒープのスナップショットを取ります。

RAM フットプリントを測定するには、以下の操作を実行します。

1. メモリ分析を有効にした状態でアプリケーションを起動します。セッション コントロール ウィンドウの**[RAM** フットプリント**]** タブを使用します。
2. アプリケーションを実行し、メモリの使用を確認するための安定した状態に導きます。
3. **[RAM** フットプリントを表示**]** をクリックし、その時点でのマネージ ヒープの詳細スナップショットを表示します。
4. ガベージ コレクタは使用可能なメモリが消耗したときにだけ実行されるため、メモリ グラフには特定の時期のメモリ量が正確に表示されない場合があります。プログラムが安定

したアイドル状態にあるときに、ガベージ コレクションを実行をクリックしてガベージ コレクタを強制的に実行し、メモリ グラフを更新します。

DevPartnerには、マネージ ヒープの状態を示すスナップショットが表示されます。このデータは**RAM**フットプリント結果サマリとして表示されます。結果サマリ ページからは、セッション データにドリルダウンし、メモリを最も多く使用しているオブジェクトやメソッドを突き止めることができます。

DevPartnerがメモリ リーク セッションまたはRAMフットプリント セッションでほとんどのガベージ コレクション ルートを適切に特定できるようにするには、デバッグを実行せずにメモリ分析を選択して開始するようにしてください。**[**メモリ分析を選択して開始**]** （デバッグあり）を有効にした状態で起動したアプリケーションのメモリ リーク データやRAM フットプリント データを収集しようとすると、すべてのガベージ コレクション ルートが「識別できない**GC**ルート」としてセッション データに表示されます。

RAM フットプリント分析のページを使用して、アプリケーションにおけるメモリの使用状況を詳細に把握します。**RAM** フットプリント結果サマリには、データを調査し、データにドリルダウンする以下の方法が用意されています。

- オブジェクトの配布
- 最も多く割り当てられているメモリを参照しているオブジェクト
- 最も多くのメモリを割り当てるメソッド

どの方法を最初に使用するかは、表示されるデータ、およびアプリケーションに対する認識に応じて異なります。

オブジェクトの配布

DevPartnerでは、メモリ内のオブジェクトの配布が円グラフとして表示されるため、アプリケーションによって使用されているメモリの割合（プロファイルされたオブジェクト）と、システム コードによって使用されているメモリの割合（システム オブジェクト）を即座に確認できます。

以下のように**[**オブジェクトの配布**]**グラフを解釈します。

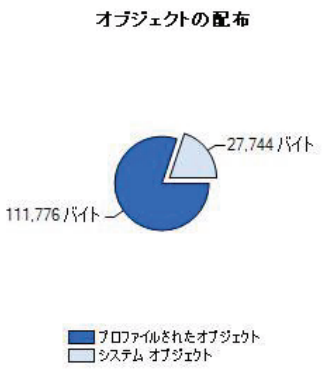


図 5-26 DevPartner メモリ分析の[オブジェクトの配布]グラフ

- 円グラフでアプリケーション（プロファイルされたオブジェクト）が最も大きな割合を占めており、ターゲットの運用環境において想定されているリソースと比較してメモリ使用が中程度から高程度である場合は、アプリケーションのどの部分で最も多くのメモリが割り当てられているかを特定する必要があります。これを行うには、**[** 最も多く割り当てら

れたメモリを参照するオブジェクト] グラフまたは [最も多くのメモリを割り当てるメソッド] グラフを使用してデータにドリルダウンします。最終的には、メモリの使用量を少なくするために、アプリケーションのソース コードのどの部分を変更または再構築できるかを特定する必要があります。

- 円グラフの [プロファイルされたオブジェクト] 部分が小さい場合は、メモリは主にアプリケーション以外の部分によって割り当てられています。これはよいことです。ただし、それでもアプリケーションの速度が遅い場合や、全体的なメモリ使用が多い場合には、アプリケーションにおけるアンマネージ コードまたはシステム リソースの使用状況を調査する必要があります。アンマネージ コードでは、メモリ内にオブジェクトが永続的に保持されることがあります。Visual Studio アプリケーションでは、.NET Framework 内での実行時間が非常に長いことがよくあります。そのような場合は、.NET Framework メソッドをより効率的に呼び出すか、または呼び出し頻度を低くできる場合があります。

以下の 2 つの分析方法のいずれかを使用して、RAM フットプリント データにドリルダウンします。

- 最も多く割り当てられているメモリを参照しているオブジェクト
- 最も多くのメモリを割り当てるメソッド

最も多く割り当てられたメモリを参照するオブジェクト

[最も多く割り当てられたメモリを参照するオブジェクト] には、セッション ファイルの作成時にライブ オブジェクトへの参照を保持していたオブジェクトが示されます。表示されるサイズは、このインスタンスから参照されるオブジェクトすべての合計です。

- [すべての情報を表示] をクリックして、これらのオブジェクトのデータにドリルダウンします。

[最も多く割り当てられたメモリを参照するオブジェクト] を使用すると、最も多くのメモリを参照するオブジェクトのインスタンスに焦点を絞って調査できます。割り当てられたメモリへの参照を持つオブジェクトのインスタンスごとにデータを整理すると、大きなオブジェクト、つまり、ガベージ コレクタがオブジェクトを収集できる場合に、最大のメモリ量を再利用できるオブジェクトに注意を向けることができます。

個々のオブジェクトが小さい場合でも、そのオブジェクトの参照先のオブジェクトによって消費されているメモリを含めると、そのオブジェクトが拡大し、大きなオブジェクトとなります。ガベージ コレクタを実行したとき、他のオブジェクトによって参照されているオブジェクトは収集できません。このため、他の多くのオブジェクトを参照しているオブジェクトは、非常に多くのメモリを拘束する場合があります。このようなオブジェクトを収集できると、そのオブジェクトが固有の参照を保持している他のすべてのオブジェクトも収集できます。このような大きなオブジェクトは、アプリケーションの RAM フットプリントを削減する場合の対象となります。

[最も多く割り当てられたメモリを参照するオブジェクト] ビューには、セッション ファイルの作成時におけるライブ オブジェクト インスタンスのリスト、および各オブジェクトのメモリに対する影響についてのデータが含まれています。また、[オブジェクト参照グラフ] ビュー、[割り当てトレース グラフ] ビュー、および [ソース] ビューを表示できるタブ付きウィンドウも含まれています。

このビューは、メモリ内の大きなオブジェクトを特定する場合に役立ちます。[参照サイズ] データには、そのオブジェクトが唯一の親であるすべての下位オブジェクトに割り当てられたメモリが含まれています。多くの場合、個々のオブジェクトは小さいサイズです。ただし、オブジェクトにいくつかの下位オブジェクトがあり、下位オブジェクトのそれぞれにさらに

下位オブジェクトがあり、上位オブジェクトと下位オブジェクトにオブジェクトごとのオーバーヘッドが存在する場合には、大量のメモリが消費されることがあります。

DevPartnerでは、オブジェクト参照パスを使用して下位オブジェクトに関連するバイト数が集計されて、その値が上位オブジェクトの使用量として表示されます。このビューの利点は、割り当て方法を変更した場合に最も大きな効果を得ることができるオブジェクトに焦点を絞って調査できることです。

最も多くのメモリを消費しているオブジェクトに照準を絞ることによって、メモリ消費を減少させるためにどのような変更を行うことができるかを即座に明確にできる可能性があります。ただし、さらに詳細に調査して、特定のオブジェクトを解放したり、特定のオブジェクトの使用方法を変更したりした場合の影響を把握することもできます。

- インスタンス リスト内で選択したオブジェクトをダブルクリックするか、またはコンテキスト メニューを使用して、選択したオブジェクトによって参照されているライブ オブジェクトを表示します。

<object name> によって参照されるライブ オブジェクト

[オブジェクトによって参照されるライブ オブジェクト**]** ビューには、メモリ内で参照されており、選択した上位オブジェクトによって参照されているすべてのライブ オブジェクトが表示されます。つまり、これらの下位オブジェクトは、上位オブジェクトが収集されると収集できるようになります。

<object name> によって参照されるすべてのオブジェクト

[オブジェクトによって参照されるすべてのオブジェクト**]** ビューには、**[**オブジェクトによって参照されるライブ オブジェクト**]** ウィンドウで選択されたオブジェクトによって参照されているオブジェクトのインスタンス リストが表示されます。

その上位のウィンドウと同様に、**[**オブジェクトによって参照されるすべてのオブジェクト**]** に表示されるデータは、割り当てられているメモリへの参照を持つオブジェクトのインスタンス別に整理されています。このビューを使用すると、オブジェクトをメモリ内に保持している参照のチェーンをさらに調査できます。**[**オブジェクトによって参照されるすべてのオブジェクト**]** ウィンドウでは、**[**オブジェクトによって参照されるライブ オブジェクト**]** 内のすべての下位オブジェクトによって参照されているオブジェクトのチェーン全体を調査できます。

オブジェクト参照のシーケンス全体の中で、引き続き任意のオブジェクトからドリルダウンして、そのオブジェクトが参照を保持しているオブジェクトをすべて表示できます。

オブジェクト参照グラフと割り当てトレースグラフ

上記すべてのオブジェクトのビューには、オブジェクト参照グラフと割り当てトレース グラフが含まれています。

オブジェクト参照グラフには、セッション ファイルが作成された時点でメモリ内に存在したライブ オブジェクトが表示されます。ライブ オブジェクトとは、メソッドを呼び出すことができるオブジェクトです。ガベージ コレクタが実行されると、ガベージ コレクタによって有効な参照を持つオブジェクトが特定されます。有効な参照とは、オブジェクトがアプリケーションのガベージ コレクション ルートからアクセス可能であることを指します。アクセス可能なオブジェクトはライブ オブジェクトとしてマークされ、収集できません。オブジェクト参照グラ

フには、これらのオブジェクト参照が表示され、オブジェクトがメモリ内に存在する理由を明確にする場合に役立ちます。

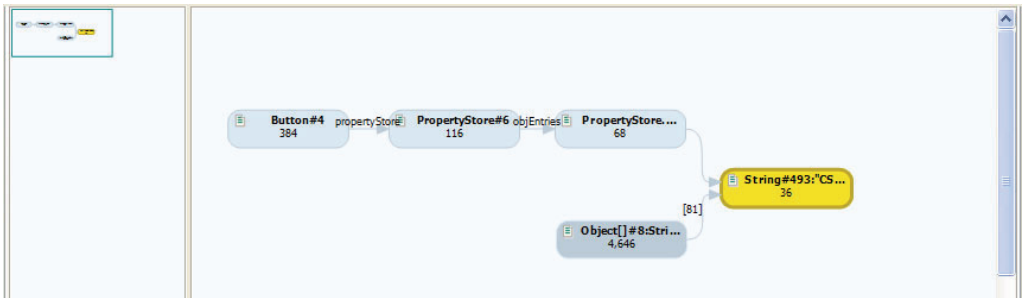


図 5-27 オブジェクト参照グラフ

アプリケーション内のメソッドによってオブジェクトが割り当てられて、オブジェクトが使用するメモリが割り当てられます。メモリを割り当てたメソッド コールのシーケンスを把握することは有用です。割り当てトレース グラフには、オブジェクトを割り当てたメソッド コールが示されます。

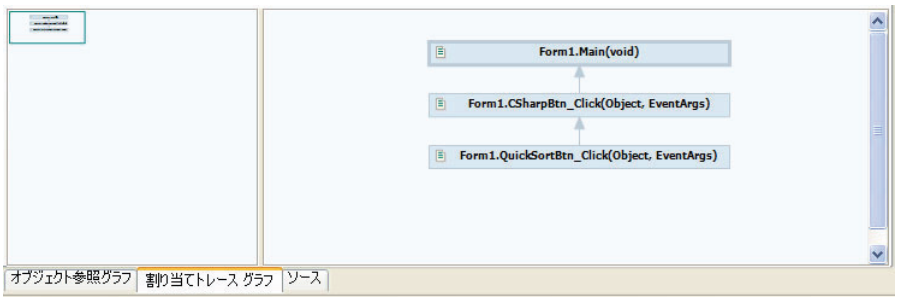


図 5-28 割り当てトレース グラフ

最も多くのメモリを割り当てるメソッド

[最も多くのメモリを割り当てるメソッド**]**ビューには、アプリケーションに最も多くのライブ メモリを割り当てたソース メソッドを示すメソッド リストが表示されます。このビューには、アプリケーションが現在の状態にある間はガベージ コレクションによって解放できないメモリを最も多くマネージ ヒープ内に割り当てるメソッドが表示されます。

[ライブ サイズ / 下位も含む　%）**]**カラムは、セッション ファイルの作成時にマネージ ヒープ内に割り当てられた合計メモリに対し、このメソッド　およびその下位メソッド）によって使用されているメモリの比率を示します。この表示によって、最もメモリを使用しているメソッドに注目できます。

このビューには、ソース コード以外に、メモリ割り当てが行われた実行パスを示すコール グラフも含まれています。詳細については、「コール グラフ」 170 ページ)を参照してください。

<method name> によって割り当てられたライブ オブジェクト

[<method name> によって割り当てられたライブ オブジェクト**]**ビューには、**[**最も多くのメモリを割り当てるメソッド**]**ビューで選択したメソッドによって割り当てられたライブ オブジェクト インスタンスのリストが表示されます。この場合、ビューは、前のウィンドウで選択されているメソッドによって割り当てられたライブ オブジェクトに限定されます。これによって、ライブ メモリを最も多く割り当てていたアプリケーション内のメソッドからドリルダウンで

きるようになります。ここから、RAM フットプリント スナップショットが取られた時点でガベージ コレクションできなかったオブジェクトを調査できます。

割り当てられたオブジェクトのリストには、[最も多くのメモリを割り当てるメソッド] ビューで選択したユーザーコード メソッドによって呼び出された、プロファイルされていない（システム）メソッドによって作成されたオブジェクトが含まれています。たとえば、メソッド内で WinForms ライブラリ内のメソッドが使用されている場合、これらのメソッドによって割り当てられたオブジェクトが、割り当てられたオブジェクトのリストに表示されます。

アプリケーションによるオブジェクトの割り当て状況を理解するためには、ドリルダウンして、調査対象のメソッドによって割り当てられているすべてのライブ オブジェクトによって参照されているすべてのオブジェクトを調査します。[このインスタンスから参照されるすべてのオブジェクト] ビューは、「<object name> によって参照されるすべてのオブジェクト」 (175 ページ) で説明されているビューと同一です。

オブジェクト参照のシーケンス全体の中で、引き続き任意のオブジェクトからドリルダウンして、そのオブジェクトが参照を保持しているオブジェクトをすべて表示できます。

メモリ使用を最適化する

アプリケーションにおけるメモリの使用状況を理解したら、メモリ使用の最適化を開始できます。通常メモリを最も多く消費するのはオブジェクトであるため、オブジェクトから分析を開始します。

アプリケーションでは、実行時に複数のオブジェクトが作成されることがあります。パフォーマンスを最適化するには、単純に作成されるオブジェクト数を減らせばよいでしょうか。どの部分で集中的にパフォーマンスの調整作業を行えばよいでしょうか。

DevPartner では、費用便益計算の大部分が自動的に計算されます。個々のオブジェクトが小さい場合でも、それらの下位オブジェクトを含めると他のオブジェクトよりもはるかにサイズが大きくなるオブジェクトも存在します。DevPartner では、ラージ オブジェクトの概念を使用して、下位オブジェクトも含めて多くのメモリを消費しているオブジェクトについての警告が表示されます。このようなオブジェクトの割り当てに集中してパフォーマンスを調整することによって、最も迅速に RAM フットプリントを削減できます。

ミディアム ライブ オブジェクトには注意が必要です。ミディアム ライブ オブジェクトとは、最初のガベージ コレクションのあとも残り、世代 1 に移行したオブジェクトです。これらのオブジェクトは、移行完了後の 2 回めのガベージ コレクションで収集されます。このため、このメモリの量がトランザクションで新たに必要となります。割り当てられるオブジェクトの数を減らすことができると、通常はパフォーマンスが向上します。

アプリケーション実行中に、複数の時点でのライブ オブジェクトを確認します。以降のトランザクションで不要なオブジェクトが割り当てられていませんか。複数のトランザクションで共有できるライブ オブジェクトはありませんか。アプリケーションでしばらく使用されないオブジェクトが割り当てられていませんか。これらに該当する場合は、アプリケーションによるオブジェクトの割り当て方法を変更できます。これにより、RAM フットプリントを削減して、パフォーマンスを向上できる可能性があります。

メモリ分析を使用して Web アプリケーションを分析する

DevPartner Studio では、Web フォーム、XML Web サービス、ASP.NET を使用するアプリケーションなど、Visual Studio で開発されたマネージ Web アプリケーションのメモリ使用状況を分析できます。サーバー側データを収集するには、DevPartner Studio をサーバー システム上にインストールする必要があります。

サーバー アプリケーションがリモート コンピュータで実行される場合にサーバー データを収集するには、リモート システムに DevPartner Studio と DevPartner Studio リモート サーバー ライセンスをインストールします。詳細は、『DevPartner Studio インストール ガイド』と『Distributed Licensing Managemen ライセンス ガイド』を参照してください。サーバー上でのデータ収集を設定するには、Visual Studio の DevPartner メモリ分析プロパティを使用します。

メモ： DevPartner セッション ファイルは、現在のソリューションと共に保存されます。Visual Studio でプロジェクトを開くのではなく、IIS から直接 Web プロジェクトを開いた場合は、異なるソリューション ファイルが使用される可能性があります。あるソリューションで作成された DevPartner セッション ファイルは、別のソリューションでは表示されません。

サーバー側メモリ データを収集する

Web アプリケーションまたはクライアント /サーバー アプリケーションの一部でメモリ分析データを収集する必要がある場合があります。DevPartner を使用すると、クライアント アプリケーションの実行中に、任意のプロセス内のマネージ コードのメモリ データを収集できます。

リモート プロセスのデータを収集するには、クライアントに DevPartner Studio をインストールして、リモート コンピュータに DevPartner Studio と DevPartner リモート サーバー ライセンスをインストールします。この構成を使用して、分散アプリケーションが実際に配置された状態でデータを収集できます。詳細は、『DevPartner Studio インストール ガイド』と『Distributed Licensing Managemen ライセンス ガイド』を参照してください。

複数プロセスのデータを収集する

Web アプリケーションやクライアント /サーバー アプリケーションでは、複数のプロセスが実行されることがありますが、DevPartner では、マネージ アプリケーションのメモリ分析データのみが収集されます。たとえば、ASP.NET アプリケーションをプロファイルする場合、DevPartner ではブラウザ プロセス（`iexplore`）のデータは収集されません。ただし、`aspnet_wp` プロセスや `w3wp` プロセスで実行されるマネージ コードのデータは収集されます。

メモリ分析を有効にしてこのようなアプリケーションを実行する場合、Visual Studio のメモリ分析セッション コントロール ウィンドウのプロセス選択リストには、サーバー プロセスとサロゲート プロセスが表示されます。プロセス リストを使用して、データ収集の対象を絞ります。

`DPAnalysis.exe` と XML 構成ファイルを使用した複数プロセス アプリケーションのプロファイルの詳細については、「コマンド ラインから分析を起動する」（183 ページ）を参照してください。

Web アプリケーション分析の前提条件

DevPartner メモリ分析で ASP.NET アプリケーションのプロファイル実行を成功させるためには、以下の 2 つの条件が満たされる必要があります。

- プロジェクトに **`web.config`** ファイルが含まれている。
- プロジェクトがデバッグ用に構成されている。これを実行するためには、**`web.config`** ファイルに、debug 属性を true に設定した compilation 要素を含める必要があります。例：
  `<compilation debug="true" />`

Webアプリケーションでメモリ分析セッションを実行する

Webアプリケーションでメモリ使用状況を分析するには、以下の手順を実行します。

1 Visual Studioで、アプリケーションのプロジェクトを含むソリューションを開きます。

2 ソリューション内にあるプロジェクトのDevPartnerのカバレッジ、メモリ、およびパフォーマンス分析の各プロパティを確認します。

3 ソリューション エクスプローラでプロジェクトを選択します。

4 [プロパティ]ウィンドウを表示するには、[表示] > [プロパティ ウィンドウ]を選択します。

メモ： システムをログオフしたり再起動したりすると、ターミナル サービス クライアントを介してターミナル サーバーに接続しているか、リモート デスクトップを介してシステムに接続されている場合にのみ、分析オプションが変更されます。

メモリ分析を有効にした状態で低速のコンピュータを再起動または起動すると、サービス コントロール マネージャによって、SMTP、FTP、またはWWWPの各サービスの起動時に無応答が報告される場合があります。これらのメッセージは無視して差し支えありません。すべてのサービスは正常に起動しています。応答がないと報告されるのは、DevPartnerはメモリ分析が有効になっている場合にIISをインストゥルメントし、これらのサービスがIISに依存するためです。

5 ローカル コンピュータ上でサーバー コンポーネントを実行していない場合は、DevPartnerデータ収集プロパティを使用してリモート データの収集オプションを設定します。

6 収集するデータの種類とサーバー上のIISのバージョンによっては、IISの構成を変更する必要が生じる場合があります。

ヒント：IISの構成変更の詳細については、オンライン ヘルプを参照してください。

7 Visual Studioで、[DevPartner] > [デバッグを実行せずにメモリ分析を選択して開始]を選択するか、DevPartnerツールバーの[メモリ分析を選択して開始]をクリックします。

メモリ分析セッション コントロール ウィンドウを使用して、実行する分析タイプを以下のいずれかから選択します。

- ◇ メモリ リーク
- ◇ 一時オブジェクト
- ◇ RAMフットプリント

実行する分析タイプの選択の詳細については、「メモリに関する問題を特定する」 158ページ)を参照してください。

8 セッション コントロール ウィンドウで、データを収集するサーバー プロセスを選択します。クライアントからアプリケーションを実行し、[表示]をクリックして、必要なマネージ ヒープのスナップショットを取ります。必要に応じて、同じセッションにある別のサーバー プロセスを選択し、別のスナップショットを取ることもできます。

9 クライアントからアプリケーションを実行すると、ASP.NETプロセスまたはIISプロセスに関するメモリ データが収集されます。Internet Explorer　クライアント）プロセスに関するデータは収集されません。

10 メモリ分析セッション コントロール ウィンドウの[セッション コントロール]ボタンを使用して、データ収集を制御します。セッション コントロール ファイルまたはセッション コントロールAPIを使用して、データ収集を自動化することもできます。

予想外のファイルの保存ダイアログ ボックスや保存済みセッション ファイルが表示される場合

一定の条件のもとでは、ASP.NET アプリケーションを終了したあとで [プロジェクトにファイルを追加] ダイアログが予期せず表示されたり、DevPartner でセッション ファイルを自動的に保存するように設定してある場合に予期せずセッション ファイルが保存されたりする場合があります。

メモリ分析セッションでは、Internet Explorer に関するデータは収集されません (DevPartner では、マネージ コードに関するメモリ分析データのみ収集されます）。このため、ASP.NET アプリケーションのメモリ分析を実行しているときは、ASP.NET ワーカー プロセス（w3wp または aspnet_wp）が一次プロファイル プロセスとなります。一次プロセスが終了すると、DevPartner はデータの収集を停止して、最終的なセッション ファイルを生成します。ほとんどの場合、これはユーザーのアクションに対する動作として行われます。ただし、ASP.NET ワーカー プロセスが実行されるシステムで **machine.config** ファイルのプロセスのモデル属性」クションを以下に示すいずれかの方法で編集しておくと、これらのプロセスをプロファイリング中に自動的にシャットダウンすることができます。

- requestLimit 属性または requestQueueLimit 属性の値を「Infinite」から、セッション中にプロセスをシャットダウンするのに十分な低い値に変更する
- timeout 属性または idleTimeout 属性の値を「Infinite」から、セッション中にプロセスをシャットダウンするのに十分な低い値に変更する
- memoryLimit属性の値をセッション中にプロセスがリサイクルするのに十分な低い比率に変更する

プロセスがシャットダウンすると、DevPartner は最後のスナップショットを取って、セッション ファイルを生成し、セッションを終了します。IE（ユーザー起動のクライアント プロセス）が依然としてアクティブである場合は、ASP.NET ワーカー プロセスの新しいインスタンスが発生します。ASP.NET ワーカー プロセスでは、その終了時に個別にセッション ファイルが作成されます。ファイルの内容は、セッション ファイルに保存されるか、[プロジェクトにファイルを追加] ダイアログ ボックスに表示されます。ただし、このセッション データは元のメモリ分析セッションの一部ではないため、通常、有用性は高くありません。

この問題を解決するには、**machine.config** ファイルを編集し、プロセスの早すぎる終了を避けるのに十分な高い値に、適切な属性を設定します。

メモ： **machine.config** ファイルを編集する前に、必ずバックアップ コピーを作成してください。

DevPartner は、Visual Studio プロパティ ウィンドウの DevPartner カバレッジ、メモリ、およびパフォーマンス分析の設定で分析が明示的に無効にされるまで、ASP.NET ワーカー プロセスが実行して終了するたびに分析データの収集を続けます。

セキュリティ例外が発生した場合

マネージ アプリケーションのデータを収集する場合、セキュリティ ポリシーで DevPartner によるコードのインストゥルメンテーションが禁止されていると、セキュリティ例外メッセージが表示されます。デフォルトでは、アセンブリをプロファイルするには SkipVerification 権限が必要です。コードが実行されるポリシーの権限セットからこの権限を削除したり、この権限を無効にするような命令型のセキュリティ宣言をアセンブリに追加した場合、アセンブリをプロファイルできなくなります。

この状態を修正するには、以下の 2 つの方法のいずれかを使用して、確実にプロファイリングを実行できるようにします。

- 以下のグローバル環境変数を設定して、アプリケーションのプロファイルを再試行します。

  ```
  NM_NO_FAST_INSTR=1
  ```

  この方法を使用すると、問題を回避することができますが、パフォーマンスが若干低下します。

- .NET Framework構成ツールのMMCスナップインを使用してアセンブリのポリシーを変更するか、またはアセンブリ内のすべての強制セキュリティ宣言を一時的に削除します。

Visual Studioのセキュリティ ポリシーの詳細については、Visual Studioオンライン ヘルプの『.NET Framework Developers Guide』を参照してください。

開発サイクルにおいてメモリ分析を使用する

問題があるのではないかと疑いを持つ前からテストを開始してかまいません。DevPartnerメモリ分析を早期から頻繁に実行して、アプリケーション分析時の分析対象を理解しておくと、問題の特定が容易で、修正によるリスクも少ないうちに問題を早期に修正できます。

多くの場合、マネージ アプリケーションにおけるメモリに関する問題は、単純なコーディング エラーではなく、より大きな設計上およびアーキテクチャ上の決定が原因です。たとえば、解放されない古い参照が原因で、あるオブジェクトが収集されず、そのためにメモリ ロスが発生することがあります。これは、コードの他の部分を修正した結果である可能性があります。これらの問題を特定したときに開発サイクルが進んでいるほど、より問題の修正が難しく、リスクも高くなります。

そのため、開発サイクル全体にわたる継続的なテスト プログラムの一部としてメモリ分析を使用することは有用です。単体テスト中にメモリ分析を使用することで、個々のモジュールにおけるメモリの処理方法を理解できます。改善が必要な領域を特定して修正したあと、再度テストして修正内容を確認します。その後、アプリケーションにモジュールを統合するときにも、メモリ テストを再度繰り返して、メモリに関する新たな問題が発生しないことを確認します。

Visual Studio Team System にデータを送信する

Visual Studio 2005および2008については、選択したメソッドに関するデータをバグタイプの作業項目としてVisual Studio Team Systemに送信します。Visual Studio 2010では、選択したメソッドに関するデータを問題、バグ、または不具合タイプの作業項目として Team Foundation Serverに送信します。

DevPartnerメモリ分析セッション ファイルのメソッド リスト ビューで選択したメソッドのデータを作業項目として送信できます。有効な作業項目には、以下の分析タイプで選択されたメソッドが含まれます。

- メモリ リーク – 最も多くリークしているメモリを割り当てるメソッド
- RAM フットプリント – 最も多くのメモリを割り当てるメソッド
- 一時オブジェクト – 最も多くのメモリを使用するメソッドと、最も多くのメモリを割り当てるエントリ ポイント

作業項目を送信すると、ビューの表示カラムのデータが該当する[作業項目のタイプ]フォームに挿入されます。**[**作業項目**]**で送信するメソッド データを変更するには、カラム見出しを右クリックし、コンテキスト メニューから**[**項目の選択**...]**を選択します。

DevPartner Studio と Visual Studio Team System の統合の詳細については、「Visual Studio Team System のサポート」 (10 ページ) を参照してください。

第6章

自動パフォーマンス分析

この章には、2つのセクションが含まれています。1つめのセクションでは、はじめてのユーザーを対象として、パフォーマンス分析を使用する簡単な手順について説明します。2つめのセクションでは、DevPartner Studioのパフォーマンス分析機能を詳細に把握するための参照情報を示します。

パフォーマンス分析のタスクに関する情報については、DevPartner Studioのオンラインヘルプを参照してください。

パフォーマンス分析とは

DevPartner Studioのパフォーマンス分析機能を使用すると、アプリケーションのパフォーマンス低下の原因となるボトルネックについて、そのボトルネックがコード、サードパーティのコンポーネント、またはオペレーティング システムのどこにあるかにかかわらず、検出できます。

DevPartnerのパフォーマンス分析：

- コンポーネントが実際に使用されているときのパフォーマンスを分析します。このことは、コンポーネントが分散システム上にある場合も同様です。
- アプリケーションの特定のフェーズを対象としてデータを収集できるため、パフォーマンスの調整作業の無駄を省くことができます。
- 対象のアプリケーションのスレッドの経過時間と、実行中の他のアプリケーションのスレッドの経過時間を区別できるため、外部の影響を受けない、正確で再現可能な結果を得ることができます。

パフォーマンス分析を今すぐ使用する

以下に示す準備、設定、実行の手順では、DevPartnerを使用してコードのパフォーマンスを分析する方法について説明します。

すぐに使い始めるには、影付きのボックス内に示された手順に従います。影付きのボックス内に説明されている内容の詳細については、ボックスの下の追加説明を参照してください。

メモ： DevPartner Studioでアプリケーションを分析する場合、昇格されたシステム権限は必要ありません。DevPartnerでのアプリケーションの分析には、アプリケーションの作成とデバッグに使用するシステム権限で十分です。

準備：分析対象を検討する

パフォーマンス分析を使用する前に、分析対象を検討します。

以降の手順では、以下の事項を前提としています。

- 単一プロセスのマネージ アプリケーションをテストします。
- アプリケーションのビルドと実行が可能です。
- ソリューションには、スタートアップ プロジェクトが含まれています。

メモ： DevPartnerパフォーマンス分析でサポートされているすべてのプロジェクト タイプのリストは、「DevPartner Studio サポートされるプロジェクト タイプ」（269ページ）を参照してください。

アプリケーションの分析では、パフォーマンス セッションを開始する前に、どのようなデータを収集するかを決定します。セッションを開始する前に、いくつかの手順を実行する必要がある場合があります。たとえば、以下の場合には設定が必要です。

- パフォーマンス分析から除外するモジュールがある場合
- アンマネージ モジュールを分析する場合
- リモート サーバー上で実行されるコードを含める場合

この手順では、アプリケーション内のすべてのローカル マネージ コードが分析されます。

設定：プロパティとオプション

分析するコードを決定したあとは、データ収集の対象にするプロパティとオプションを設定できます。

この手順では、DevPartnerのデフォルトのプロパティとオプションを使用できます。追加の設定は必要ありません。

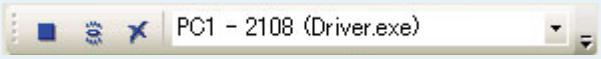
ソリューション プロパティとプロジェクト プロパティを使用して、.NETアセンブリの情報、アプリケーション外部で実行されるCOM、実行中の他のアプリケーションのスレッドの経過時間、行レベル分析またはメソッド レベル分析などを分析セッション データに含める必要があるかどうかを選択できます。DevPartnerオプションを使用すると、表示オプションの変更、アプリケーションの一部の分析からの除外、またはセッション コントロール ファイルを作成することによるデータ収集の管理を行うことができます。設定のカスタマイズの詳細については、「プロパティとオプションを設定する」（190ページ）を参照してください。

実行：パフォーマンス データを収集する

分析対象を検討し、適切なプロパティとオプションを設定したあとは、パフォーマンス データの収集を開始できます。

1. Visual Studioで、アプリケーションに関連付けられたソリューションを開きます。
2. **[DevPartner]** > **[**パフォーマンス分析を選択して開始**]**を選択して、パフォーマンス分析セッションを開始します。

   セッションの実行中は、セッション コントロール ツールバーのオプションがアクティブになります。

   DevPartnerのセッション コントロールを使用すると、アプリケーションの任意のフェーズにパフォーマンス分析の焦点を絞ることができます。セッション コントロールを使用してデータ収集を停止し、現時点で収集されたデータのスナップショットを取ってからレコーディングを継続することも、収集されたがまだスナップショットに保存されていないデータをクリアすることもできます。
3. 分析するコードを実行します。
4. **[**スナップショット**]**アイコンをクリックします セッション ウィンドウにフォーカスを移動する必要がある場合は、2回クリックします）。スナップショットを取ると、DevPartnerによって、収集されたデータを含むセッション ファイルと呼ばれるファイルが作成され、セッション ファイルのデータが表示されます。
5. アプリケーションに戻って、テストを引き続き実行します。
6. テストの実行が終了したら、アプリケーションを終了します。Visual Studioに最終的なセッション ファイルが表示されます。

メモ： マネージ アプリケーションのデータを収集しようとしてセキュリティ例外メッセージが表示された場合は、194ページのセキュリティ ポリシーの変更方法を参照してください。

パフォーマンス分析を実行するとき、必要に応じてデバッグを実行することができます。デバッグを実行しないセッションの結果の方が容易に解釈できるため、通常のパフォーマンス分析はデバッグなしで実行してください。アプリケーションをデバッガで実行した場合は、セッション中にブレークポイントにヒットした場合など、一部のタイミング値が予想より大きくなることがあります。

データを分析する

スナップショットを取るか、アプリケーションを終了すると、図 6-1に示すように、Visual Studioにセッション ファイルが表示されます。以下に、セッション ウィンドウの構成要素を示します。

- フィルタ ペイン。アプリケーションに含まれているソース ファイルとイメージがリストされます。フィルタ ペインには、セッションの経過時間に対する各ファイルの経過時間の割合が表示されます。フィルタ ペインには、最も重要なデータに焦点を絞るために使用できるフィルタ セットも表示されます。

- セッション データ ペイン。[メソッド リスト] タブ、[ソース] タブ、[セッション サマリ] タブがあります。セッション データ ペインには、フィルタ ペインで選択されているファイルまたはフィルタのデータが表示されます。

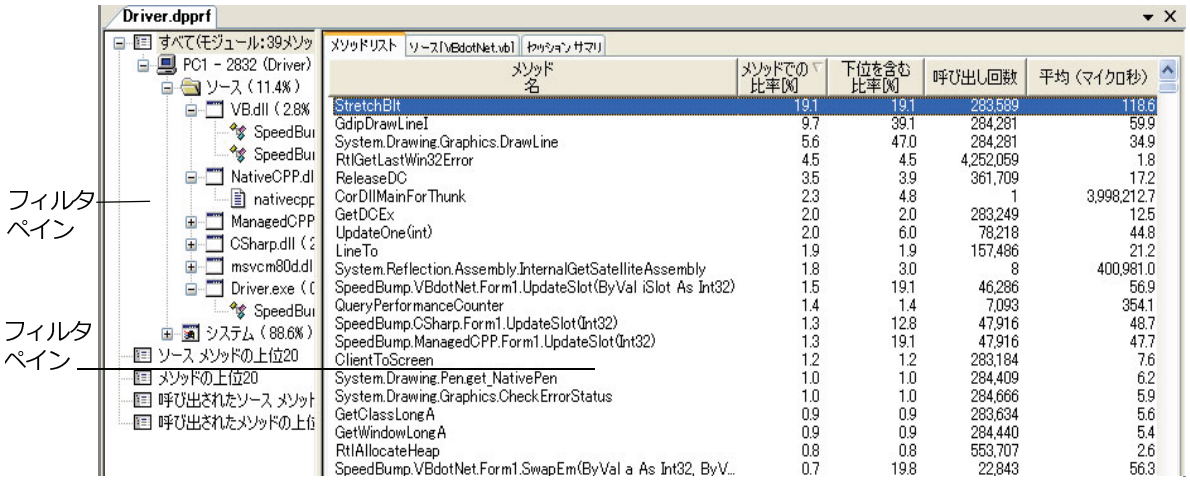


図 6-1 パフォーマンス分析のセッション ウィンドウ

フィルタ ペインとセッション データ ペインを使用する

データの評価を開始するには、フィルタを使用して開始し、[メソッド リスト] を調べて、プログラムの処理時間において大きな割合を占めているメソッドを探します。

1. フィルタ ペインで、[ソース メソッドの上位 **20**] フィルタをクリックします。表示されるデータが少なくなり、対象にするソース メソッドに焦点を絞ることができます。

   システム ファイルの経過時間を把握することは、パフォーマンスを分析する場合には役立ちますが、このフィルタを使用してシステム ファイルを表示しないようにすると、パフォーマンスの調整作業の焦点を絞ることができます。

2. [メソッド リスト] タブのデータを調べます。[メソッド リスト] タブには、各メソッドの経過時間に関する情報が表示されています。

   1 つまたは複数のカラムに大きな値があるメソッドを [メソッド リスト] で検索すると、パフォーマンス向上のために作業対象にする特定の領域がわかります。

3. [メソッド リスト] タブで、[メソッドでの比率 **[%]**] を確認します。このカラムには、セッションの経過時間に対するメソッドの経過時間の割合が表示されています（デフォルトでは、各データは [メソッドでの比率 **[%]**] で降順にソートされています。そうでない場合は、カラム見出しをクリックしてください）。

4. [下位を含む比率 **[%]**] を確認します。このカラムには、セッションの経過時間に対するメソッドとその下位メソッドの経過時間の割合が表示されています。

5. [平均] を確認します。このカラムには、メソッドの平均実行時間が表示されています。

これらのカラムの値を調べると、パフォーマンス向上のために作業対象にするコードの領域がわかります。

コール グラフを表示する

パフォーマンス上の問題の中には、上位メソッドと下位メソッドの間でコールが行われるコンテキストでのみ確認できるものがあります。このような場合は、コール グラフを調べることが役立ちます。コール グラフとは、アプリケーションの複数のメソッドのコール関係を表したグラフのことです。

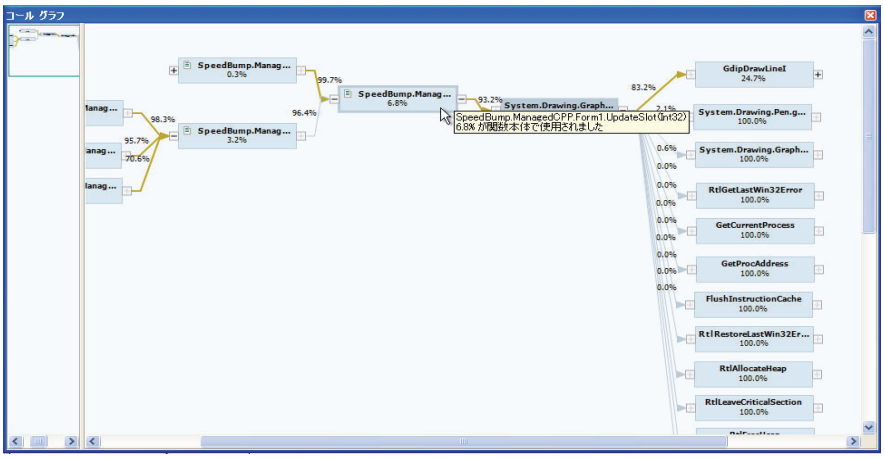


図 6-2 コール グラフ

コール グラフには、[メソッド リスト]タブか[ソース]タブのコンテキスト メニュー、または[コール グラフ]アイコン  からアクセスできます。

6 [メソッド]リストでメソッドを右クリックし、コンテキスト メニューの[コール グラフの表示]を選択します。メソッドのコール グラフが表示されます。クリティカル パスは、システムのデフォルトの色で強調表示されています。

ヒント：以下のステップは、コール グラフの使用方法の概要です。コール グラフの詳細については、「コール グラフを分析する」（202ページ）を参照してください。

7 任意のノードの両端にあるプラスまたはマイナスのアイコンをクリックすると、上位ノード（左）または下位ノード（右）の表示が展開されるか、折りたたまれます。ノードに表示されている割合の値と、下位ノードに続く線の上に表示された割合とを比較して、パフォーマンス上の問題の原因になっている可能性があるパスを探します。

8 ノードの上、またはメソッド ノード間のリンクに示された割合の値の上にマウス ポインタを置くと、詳細情報が表示されます。

9 パフォーマンスの向上に関係する可能性のあるメソッドを識別します。右クリックし、コンテキスト メニューから[メソッドのソース]を選択します（システム メソッドを選択した場合、ソースは表示されません）。

メソッドのソース コードがセッション ウィンドウの[ソース]タブに表示されますが、フォーカスはまだコール グラフにあります。

10 ソース コードの作業を始めるには、コール グラフを閉じます。

ソース コードを表示する

[ソース]タブには、選択したファイルまたはメソッドのソース コードが表示されます。[ソース]タブは、パフォーマンス上の問題の原因となっている可能性のあるコード行の識別に役立ちます。

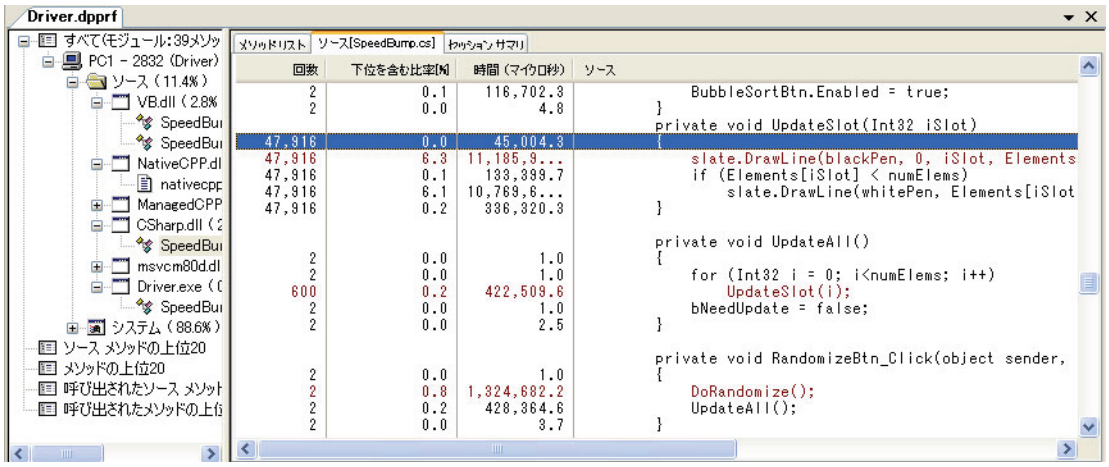


図6-3 [ソース]タブ

11 図 6-3に示すように、選択したメソッドのソース コードが[ソース]タブに表示されます。[ソース]タブには、実行済みのコード行のそれぞれに関するパフォーマンス セッション データが表示されます。

各メソッドで最も実行速度が遅いコード行が強調表示されます。パフォーマンスが向上する可能性があるかどうかを判断し、それに応じてコードを変更します。

セッションを比較する

パフォーマンス上の問題を特定して修正したあとは、別のパフォーマンス セッションを実行して、変更の前後のセッション ファイルを比較できます。DevPartnerにより、セッション間の違いを表示する比較ウィンドウが表示されます。比較セッションの詳細については、「セッションを比較する」（205ページ）を参照してください。

セッションのサマリ データを表示する

[セッション サマリ]タブには、パフォーマンス分析セッションの概要が表示されます。

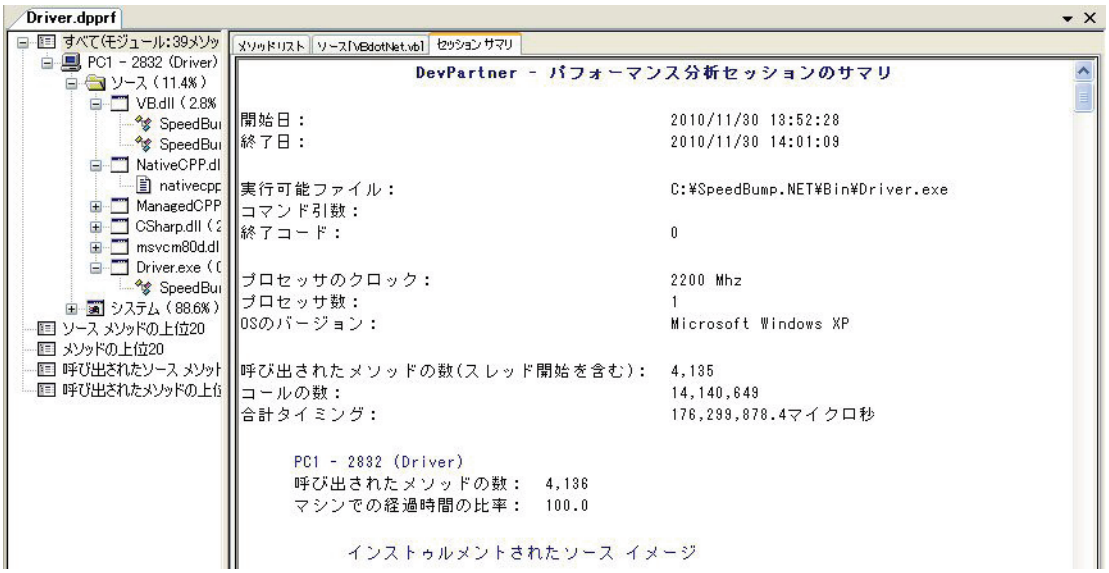


図6-4 [セッション サマリ]タブ

12 **[セッション サマリ]**タブをクリックします。

[セッション サマリ]には、セッションの日時、プロセッサ速度、オペレーティング システムなどのセッションに関するコンテキスト情報が含まれます。この情報は、特に他のユーザーが作成した古いセッション ファイルを参照する場合に役立ちます。

また、サマリには、フィルタ ペインと**[メソッド リスト]**タブのパフォーマンス データも含まれており、分析されたファイルとメソッドの両方のデータが表示されています。

13 タブ内でスクロールして、セッション サマリ データを参照します。

セッション ファイルを保存する

パフォーマンス データの確認後、セッション ファイルを保存できます。

1 Visual Studioのセッション ファイル ウィンドウを閉じて、セッション ファイルを保存します。

2 デフォルトのファイル名と場所を受け入れる場合は、**[OK]**をクリックします。

DevPartnerでは、セッション ファイルはアクティブなソリューションの一部として保存されます。これらのファイルは、ソリューション エクスプローラの**DevPartner Studio**仮想フォルダに表示されます。パフォーマンス分析のセッション ファイルには、**.dpprf**という拡張子が付きます。

デフォルトでは、セッション ファイルはプロジェクトの出力フォルダに保存されます。ファイル名は、デフォルト フォルダに配置されているファイル名に自動的に番号を追加する形式（たとえば、**MyApp.dpprf**、**MyApp1.dpprf**）に設定されます。セッション ファイルをデフォルト フォルダ以外の場所に保存した場合は、ユーザーがファイル名と番号を管理する必要があります。

出力フォルダがないプロジェクトの場合（Visual Studio 2005 Webサイト プロジェクトなどの場合）、ファイルはプロジェクト フォルダに物理的に保存されます。

コマンド ラインで生成されたセッション ファイルは、プロジェクトのソリューションに自動的には追加されません。外部で生成されたセッション ファイルは、Visual Studio に開いたソリューションに手動で追加できます。

これで、この章の準備、設定、実行のセクションは終了です。これで、パフォーマンス分析セッション実行のメカニズムの基礎が理解できました。追加情報については、引き続きこの章の残りの部分を参照してください。タスクベースの情報については、DevPartner のオンライン ヘルプを参照してください。

プロパティとオプションを設定する

パフォーマンス分析セッションを開始する前に、データ収集を微調整して、特定のタイプの情報を含めるか除外しておくと、多くの場合に役立ちます。ソリューション プロパティ、プロジェクト プロパティ、および **DevPartner** オプションを使用して、分析セッションをより目的に適したものにします。

ソリューション プロパティ

ソリューション レベルで使用可能なパフォーマンス プロパティを表示するには、ソリューション エクスプローラでソリューションを選択してから [F4] キーを押して、プロパティ ウィンドウを表示します。

図 6-5 ソリューション プロパティ

パフォーマンス分析に影響を与えるソリューション プロパティは、以下のとおりです。

- **.NET** から収集 – マネージ コードのアプリケーションにのみ表示されます。.NET アセンブリの情報を収集しない場合は、このプロパティを [False] に設定します。

  このプロパティは、カバレッジ分析とパフォーマンス分析のセッションにのみ影響します。メモリ分析とパフォーマンス エキスパートでは、この値が [False] に設定されていても、常にマネージ アプリケーションからデータが収集されます。

  DevPartner for Visual C++ Boundschecker Suite では、**[.NET** から収集**]** プロパティは使用できません。

- スタートアップ プロジェクト – ソリューションには、スタートアップ プロジェクトを含める必要があります。ソリューションに複数のスタートアップ プロジェクトが含まれている場合は、分析の開始前に、セッションで使用するスタートアップ プロジェクトを選択するように求めるプロンプトが表示されます。

プロジェクト プロパティ

プロジェクト レベルのプロパティを参照するには、ソリューション エクスプローラでプロジェクトを選択して、ソリューション内のプロジェクトに設定可能なプロパティを参照します。

図 6-6 プロジェクト プロパティ

パフォーマンス分析に影響を与えるプロジェクト プロパティは、以下のとおりです。

- **COM**情報の収集 – DevPartnerは、DLLエクスポートおよびCOMインターフェイスに基づいてメソッド レベルのデータを収集します。アプリケーションの外部で実行されるCOM の情報を収集しない場合は、[False] を選択します。
- その他を除外 – 他のアプリケーションのスレッドの経過時間は除外します。アプリケーションのスレッドに費やされた時間だけが測定されます。

  パフォーマンス情報の収集中に、コンテキスト スイッチが監視され、アプリケーションの外部にあるスレッドの処理に要したCPU 時間が計測されます。タイミング データの収集後、クロック時間から他のスレッドでの経過時間が引かれ、アプリケーションの正確な経過時間が計算されます。

  この機能を有効にする場合は [True]、無効にする場合は [False] を選択します。
- インライン関数をインストゥルメントする – インライン関数をインストゥルメントする場合は、このプロパティを [True] に設定します (インストゥルメンテーションについては、「インストゥルメンテーションについて」 (193 ページ) を参照してください)。インライン最適化が有効な場合は、デフォルトでインライン関数はインストゥルメントされません。

  マネージ C++ アプリケーションの場合、**[**インライン関数をインストゥルメントする**]** プロパティを **[True]** に設定しても、DevPartnerは `__forceinline` キーワードで明示的にインラインされている関数のデータを収集しません。
- インストゥルメンテーション レベル – **[**メソッド**]**または**[**行**]**を選択します (インストゥルメンテーションについては、「インストゥルメンテーションについて」 (193 ページ) を参照してください)。
  - ◇ メソッド : メソッドレベルのインストゥルメンテーションでは、パフォーマンス分析を短時間で実行できますが、メソッドレベルのデータしか提供されません。
  - ◇ 行 : 行レベルのインストゥルメンテーションでは、ソース コードの特定の行までドリルダウンすることができます。.NET アプリケーションで作業する場合、[インストゥルメンテーション レベル] として [行] を選択し、グローバル アセンブリ キャッシュ (GAC) に JIT コンパイルのアセンブリをインストールすると、DevPartnerパフォー

マンス分析は、アセンブリに関する行レベルのデータを提供できません。DevPartner は、JIT コンパイルのアセンブリをインストゥルメントできません。行レベルのデータを収集するには、パフォーマンス分析の実行の際、事前に JIT アセンブリを行わないでください。

すべてのプロパティ設定は、明示的に変更しないかぎり保持されます。

オプション

パフォーマンス分析セッションの DevPartner オプション設定を確認するには、**[DevPartner]** > **[** オプション **]** > **[** 分析 **]** を選択します。

- **[** 表示 **]** オプションを使用すると、データ表示時の精度、目盛り、および単位を設定できます。
- **[** 除外 **]** オプションを使用すると、データ収集から 1 つまたは複数のイメージを除外できます。除外の詳細については、「イメージを除外する」 192 ページ）を参照してください。
- **[** セッション コントロール ファイル **]** オプションを使用すると、アプリケーションやモジュールの実行時に DevPartner によって収集されるデータを制御するためのルールとアクションのセットを作成できます。セッション コントロール ファイルの詳細については、「分析セッションの制御」 301 ページ）を参照してください。

[環境 **]** > **[** フォントと色 **]** などのその他の Visual Studio オプションも、DevPartner の機能に影響を与えます。

イメージを除外する

パフォーマンス分析でアプリケーションを実行すると、DevPartner によって、すべてのソース イメージとシステム イメージが収集されます。この場合に、除外オプションを使用すると、1 つまたは複数のイメージを分析から除外できます。

分析オプションを表示した状態（**[DevPartner]** > **[** オプション **]** > **[** 分析 **]**）で、**[** 除外 - パフォーマンス **]** を選択します。

ページ上部にある **[** 表示 **]** リストで、以下のいずれかを選択します。

- グローバルな除外
- 現在のフォルダでのローカルな除外
- 実行可能フォルダでのローカルな除外

[現在のディレクトリ内のローカル除外**]**オプションと**[**実行可能ディレクトリ内のローカル除外 **]** オプションは、ソリューションが開かれており、実行可能フォルダが現在の作業フォルダと異なるときにのみ使用できます。

[挿入 **]** をクリックして、イメージを除外リストに追加します。名前を入力するか、除外するイメージを参照します。除外できるファイル タイプは、**`.exe`**、**`.dll`**、**`.ocx`**、**`.netmodule`** です。**[** ファイルの種類 **]** リストを使用して、表示するファイルの種類を制限します。

.NET モジュール（**`netmodule`**）を選択した場合は、モジュールのアンマネージ部分のみが除外されます。

除外リストからイメージを削除するには、項目を選択して **[** 削除 **]** をクリックします。

[システム イメージを除外する **]** チェック ボックスをオンにすると、インストゥルメントされていないシステム DLL が DevPartner パフォーマンス プロファイリングから除外されます。

グローバルな除外は、DevPartnerのインストール フォルダの**¥Analysis**サブフォルダ内の**nmexclud.txt**に保存されます。ローカル除外は、アプリケーションの実行可能フォルダまたは現在の作業フォルダの**nmexclud.txt**に保存されます。他の場所に除外リスト（**nmexclud.txt**）のコピーを保存する場合は、**[**名前を付けて保存**]**をクリックします。

メモ： 実行中のアプリケーションを完全に監視するために、DevPartnerではいくつかの特定のWin32 APIが常にプロファイルされます。したがって、一部のシステムDLLは個別に除外できず、**[**システム イメージを除外する**]**をオンにしてすべてのシステム イメージを除外しないかぎり、セッション ファイルのシステム イメージ リストに常に表示されます。

除外操作は、**[**ネイティブ**C/C++**インストゥルメンテーション**]**でコンパイルされたファイルには適用されません。たとえば、インストゥルメントされたアンマネージC/C++イメージを除外した場合でも、そのファイルの情報は引き続き収集されます。ただし、システム コール情報は収集されません。データ収集からアンマネージC/C++イメージを除外する場合は、そのイメージをインストゥルメントしないでください。

インストゥルメンテーションについて

マネージ アプリケーションを実行すると、DevPartnerによって、アセンブリがコンパイラにロードされるときに、各アセンブリのバイト コードにフックが挿入されます。これがインストゥルメンテーションと呼ばれるプロセスです。このコードには、アプリケーションの実行時にパフォーマンス データの収集に使用されるインストラクションが含まれています。DevPartnerのインストゥルメンテーションでは、ディスク上の実際のファイルは変更されません。実行時にファイルのメモリ内の表現が変更されます。

実行時にインストゥルメントされるマネージ コードとは異なり、アンマネージC/C++コードはコンパイル時にインストゥルメントする必要があります。アンマネージ コードをインストゥルメントする場合、DevPartnerによってソース コードに直接フックが挿入されます。使用されるインストゥルメンテーションのタイプ、およびソリューション内のインストゥルメンテーションから除外するプロジェクトを指定できるインストゥルメンテーション マネージャを使用します（インストゥルメンテーション マネージャの詳細については、「アンマネージ コードのデータを収集する」（194ページ）を参照）。アンマネージ プロジェクトをリビルドすると、フックが挿入されます。フックを削除するには、[DevPartner]メニューの[ネイティブC/C++インストゥルメンテーション]オプションの選択を解除してインストゥルメンテーションをオフにしてから、プロジェクトをリビルドします。

さまざまなタイプのアプリケーションのデータを収集する

このセクションでは、DevPartnerパフォーマンス分析を使用してさまざまなタイプのアプリケーションからデータを収集する方法の詳細について説明します。

DevPartnerでは、Visual Studioのすべてのマネージ コードの言語とアンマネージC/C++がサポートされています。さらに、Internet ExplorerまたはIISを使用するときのJScriptおよびVBScript Webアプリケーションのパフォーマンス データも収集できます。

Visual Studioの各バージョンでサポートされているすべての言語とプロジェクト タイプのリストは、付録A「DevPartner Studioサポートされるプロジェクト タイプ」を参照してください。

マネージ コードのデータを収集する

Visual Studioで開発されたアプリケーションの多くは、C#アプリケーション、Visual Basicアプリケーション、マネージC++アプリケーションなどのマネージ アプリケーションです。

DevPartnerでは、マネージ アプリケーション ソース コードに関する詳細情報を収集するには、PDB（プログラム データベース ファイル）の情報が必要です。[ソース]タブにソースデータが表示されない場合、またはフィルタ ペインにソース ファイルが表示されない場合は、**.pdb**ファイルが生成されることを確認してください。

PDB情報のないマネージ アプリケーション ファイルは、フィルタ ペインのシステム フォルダに表示されます。

マネージ アプリケーションのデータを収集する場合、セキュリティ ポリシーでDevPartnerによるコードのインストゥルメンテーションが禁止されていると、セキュリティ例外メッセージが表示されます。デフォルトでは、アセンブリをプロファイルするには`SkipVerification`権限が必要です。コードが実行されるポリシーの権限セットからこの権限を削除したり、この権限を無効にするような命令型のセキュリティ宣言をアセンブリに追加した場合、アセンブリをプロファイルできなくなります。

この状態を修正するには、以下の2つの方法のいずれかを使用して、確実にプロファイリングを実行できるようにします。

- 以下のグローバル環境変数を設定して、アプリケーションのプロファイルを再試行します。

  ```
  NM_NO_FAST_INSTR=1
  ```

  この方法を使用すると、問題を回避することができますが、パフォーマンスが若干低下します。

- .NET Framework構成ツールのMMCスナップインを使用してアセンブリのポリシーを変更するか、またはアセンブリ内のすべての強制セキュリティ宣言を一時的に削除します。

Visual Studioのセキュリティ ポリシーの詳細については、Visual Studioオンライン ヘルプの『.NET Framework Developers Guide』を参照してください。

アンマネージ コードのデータを収集する

パフォーマンスをプロファイルするためにアンマネージVisual C++アプリケーションをネイティブ**C/C++**インストゥルメンテーションを使用してビルドすると、DevPartnerはコンパイラと連携して、アプリケーション イメージにインストラクションを追加し、実行時にパフォーマンス データを収集します。たとえば、メソッドの開始や終了のたびにDevPartnerが呼び出されます。この情報を使用して、メソッドの実行時間が計測されます。

アンマネージ コードをインストゥルメントするには、データ収集の対象にするアンマネージC/C++プロジェクトが含まれるソリューションを開き、**[DevPartner]** > **[**ネイティブ**C/C++**インストゥルメンテーション マネージャ**]**を選択します。

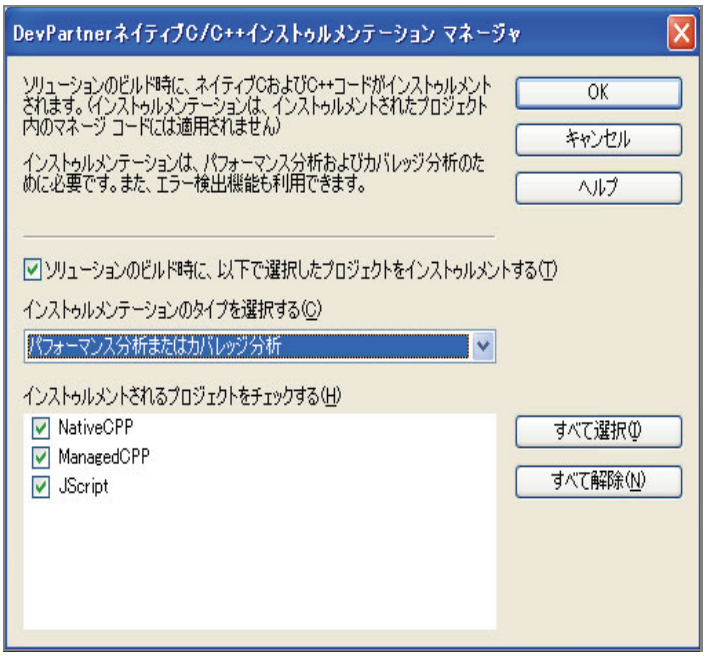


図 6-7 インストゥルメンテーション マネージャ

[ソリューションのビルド時に、以下で選択したプロジェクトをインストゥルメントする**]** チェック ボックスをオンにして、インストゥルメンテーションのタイプを選択します。選択するインストゥルメンテーションのタイプは、その後実行する分析のタイプと一致している必要があります。

インストゥルメントするプロジェクトを選択します。デフォルトで、DevPartner では、ソリューション内のすべてのアンマネージ コードがインストゥルメントされます。除外するモジュールの選択を解除します。

[OK] をクリックして、ソリューションをリビルドします。選択したアンマネージ C/C++ プロジェクトがインストゥルメントされます。DevPartner ツールバーの **[**パフォーマンス分析を選択して開始**]** をクリックして、分析セッションを開始します。

選択したプロジェクトはソリューションと共に **Native C/C++** インストゥルメンテーション マネージャに保存されます。いったんインストゥルメンテーション マネージャを使用してインストゥルメンテーションを設定すると、[DevPartner] メニューの **[**ネイティブ **C/C++** インストゥルメンテーション**]** オプションまたは DevPartner ツールバーの **[**ネイティブ **C/C++** インストゥルメンテーション**]** ボタンで、インストゥルメンテーションをオンまたはオフにすることができます。ネイティブ **C/C++** インストゥルメンテーション マネージャは、設定を変更する場合にのみ使用してください。

アプケーションからインストゥルメンテーションを削除するには、[DevPartner] メニューの **[**ネイティブ **C/C++** インストゥルメンテーション**]** オプションの選択を解除します。次にパフォーマンス分析セッションを開始またはソリューションをリビルドするときは、Visual Studio によってインストゥルメンテーションなしでソリューションがリビルドされます。

メモ： アプリケーションによって Visual Studio コンポーネントが呼び出される場合は、Visual Studio でパフォーマンス分析を行うために、これらのコンポーネントを DevPartner インストゥルメンテーションでコンパイルする必要があります。詳細については、Visual Studio の DevPartner Studio オンライン ヘルプを参照してください。

混合モードの C++ ファイル

アンマネージ（ネイティブ）C++ で /clr オプションを指定すると、アプリケーションをマネージ コードとしてコンパイルできます。その場合は、ネイティブ コードとしてコンパイルする部分を #pragma でマークできます。コンパイラは #pragma セクションで定義されたメソッドのネイティブ コードを生成します。DevPartner では混合モードの C++ ファイルはサポートされていません。マネージ セクションとアンマネージ（ネイティブ）セクションの両方が混在する C++ ファイルを含むプログラムをプロファイルすると、マネージ コード部分のみのカバレッジ データが収集されて、#pragma で囲まれた部分のネイティブ コードのカバレッジ データは収集されません。アンマネージ C++ コードのデータを収集するには、そのアンマネージ コードを別のファイルに配置してインストゥルメントします。「アンマネージ コードのデータを収集する」（194ページ）を参照してください。

複数プロセスのデータを収集する

1 つのアプリケーションで 1 つ以上のプロセスが実行される場合があります。たとえば、ASP.NET アプリケーションをプロファイルすると、ブラウザ プロセス（iexplore）、IIS プロセス（inetinfo）、および ASP ワーカー プロセス（aspnet_wp または w3wp）が実行されていることがわかります。

パフォーマンス分析で複数プロセスのアプリケーションを実行すると、DevPartner のセッション コントロール ツールバーのプロセス選択リストにアクティブなプロセスが表示されます。

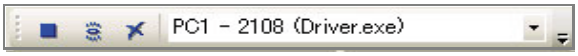


図 6-8 プロセス選択リストが表示されたセッション コントロール ツールバー

プロセス選択リストを使用して、データ収集の対象を絞ります。スナップショットを取ると、DevPartner によって、プロセス選択リストで選択されたプロセスのデータが含まれているセッション ファイルが作成されます。

リモート システムのデータを収集する

リモート システムで実行中のアプリケーション コンポーネントのパフォーマンス データを収集できます。たとえば、クライアント /サーバー アプリケーションのクライアント部分とサーバー部分の両方のパフォーマンス データを収集できます。DevPartner を使用すると、クライアント アプリケーションを実行しながら、クライアント プロセスとサーバー プロセスのパフォーマンス データを収集できます。

クライアント システムとリモート システムから同時にデータを収集するには、クライアント システムに DevPartner を、リモート システムに DevPartner と DevPartner リモート サーバー ライセンスをインストールします。リモート サーバー ライセンスについては、『DevPartner Studio インストール ガイド』および『Distributed License Management ライセンス ガイド』を参照してください。

ターミナル サービス接続経由で接続されるサーバーでは、DevPartner リモート サーバー ライセンスは必要ありません。ターミナル サービスの詳細については、「ターミナル サービスとリモート デスクトップを使用する」（211ページ）を参照してください。

リモート システムで関連するプロジェクトを選択し、DevPartner プロパティを参照して、クライアント システムで設定したオプションと一致することを確認します。オプションの変更後、IIS などのサーバー プロセスが再起動されます。この再起動は、変更を有効にするために必要です。

アンマネージ C++ アプリケーションを分析する場合は、インストゥルメンテーションを指定します。「アンマネージ コードのデータを収集する」（194ページ）に説明されているように、アプリケーションからアンマネージ C++ コンポーネントを呼び出している場合にそれらのコンポーネントのデータを収集するには、それらのコンポーネントをインストゥルメントする必要があります。

データを関連付ける

Internet Explorer（IE）をブラウザとして使用し、Internet Information Server（IIS）を Web サーバーとして使用している場合、または COM を使用してインタープロセス コールを行う場合、クライアント／サーバーの関係はプロセス間で自動的に認識されます。DCOM オブジェクトのメソッド間、HTTP クライアントとサーバー（IE と IIS）の関係などを保持しておくために、それらのセッションのデータは自動的に関連付けられます。次に、そのデータとクライアント セッション データが 1 つのセッション ファイルに結合されます。

関連付けられたセッション ファイルには、アプリケーションのクライアント部分とサーバー部分の両方のパフォーマンス データが含まれています。関連付けられたセッション ファイルは、Visual Studio では他のセッション ファイルと同様に、ファイル名に **_co** を追加して（たとえば **appname_CO.dpprf**）表示されます。

関連付けられたセッション ファイルを **[** コール グラフ **]** で表示すると、呼び出し側メソッドから呼び出されたメソッドまでの COM コール スタックを追跡できます。DevPartner は、クライアント システムのクロック速度に一致するように、サーバー側データの単位を変更します。

COM ベースの関係、または IE と IIS 間でクライアント／サーバーの関係がない場合は、**[DevPartner]** > **[** 関連付け **]** > **[** パフォーマンス ファイル **]** を使用して、さまざまなセッション ファイルからのデータを手動で結合できます。また、「コマンド ラインから分析を起動する」（183ページ）に説明されているように、`NMCORRELATE` コマンド ライン ユーティリティを使用して、データを手動で結合することもできます。

.NET Web アプリケーションのデータを収集する

Web フォーム、XML Web サービス、または ASP.NET アプリケーションを開発する場合は、DevPartner を使用して、アプリケーションのクライアント部分とサーバー部分の両方のパフォーマンス データを収集できます。ローカル コンピュータまたはリモート コンピュータで実行されている IIS と ASP.NET のデータを収集するように、DevPartner を設定できます。

アプリケーションによって呼び出されたアンマネージ C++ コンポーネントのデータを収集するには、「アンマネージ コードのデータを収集する」（194ページ）に説明されているように、ネイティブ **C/C++** インストゥルメンテーションを使用して、対象となるコンポーネントをインストゥルメントおよびリビルドする必要があります。Web アプリケーションによって C++ コンポーネントが呼び出される場合は、Visual Studio で DevPartner コマンドを使用して、そのコンポーネントをインストゥルメントする必要があります。必ずパフォーマンス分析でインストゥルメントしてください。DevPartner は、1 つのセッションでは 1 つの分析タイプのデータしか収集しません。

メモ： DevPartner セッション ファイルは、現在のソリューションと共に保存されます。Visual Studio でプロジェクトを開くのではなく、IIS から直接 Web プロジェクトを開いた場合は、異なるソリューション ファイルが使用される可能性があります。あるソリューションで作成された DevPartner セッション ファイルは、別のソリューションでは表示されません。

前提条件

DevPartnerパフォーマンス分析でASP.NETアプリケーションのプロファイル実行を成功させるためには、以下の2つの条件が満たされる必要があります。

- プロジェクトに **web.config** ファイルが含まれている。
- **web.config** ファイルに、debug属性をtrueに設定したcompilation要素が含まれている。例:

```
<compilation debug=" true" />
```

対象アプリケーションが呼び出すプロセス内部またはプロセス外部のコンポーネントのデータも収集できます。

デバッグなしで **ASP.NET** アプリケーションを分析

最適な結果を得るには、パフォーマンス分析をデバッグなしで実行します。

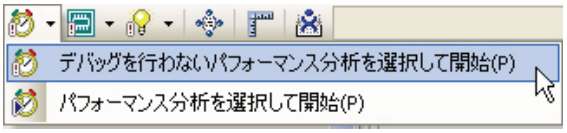


図6-9 デバッグなしで開始するオプション

一度に1つのスクリプト デバッガのみがアクティブになることができます。デバッグありのオプションを指定してWebアプリケーションをデバッグすると、Visual StudioとDevPartnerの両方でスクリプト デバッガのロードが試みられます。その場合、スクリプト デバッガがIEにアタッチできないというメッセージが表示されます。ただし、エラー メッセージが表示されても、セッションは中断されずに続行されます。

エラー メッセージが表示されないようにするには、iexploreでスクリプトのデバッグを無効にするか、デバッグなしでパフォーマンス分析を実行します。

予期せず [プロジェクトにファイルを追加] ダイアログ ボックスが表示される、または予期せずセッション ファイルが保存される

一定の条件のもとでは、ASP.NETアプリケーションを終了したあとで [プロジェクトにファイルを追加] ダイアログ ボックスが予期せず表示されたり、DevPartnerでセッション ファイルを自動的に保存するように設定してある場合に予期せずセッション ファイルが保存されたりする場合があります。

DevPartnerでは、ASP.NETアプリケーションに対してパフォーマンス分析を実行すると、Internet Explorerのデータが一次プロファイル プロセスとして収集されます。また、ASP.NETワーカー プロセス（w3wpまたはaspnet_wp）などの2次プロセスに関するデータも保存されます。一次プロセスが終了すると、DevPartnerではデータの収集が停止されて、クライアント ワーカー プロセスのデータ（IE）とサーバー ワーカー プロセスのデータ（IISとASP.NET）の両方を含む、最終的な相関セッション ファイルが生成されます。セッション コントロール ツールバーでプロセスを選択すると、サーバー プロセスだけのスナップショットを取ることもできます。

ほとんどの場合、クライアント プロセスとサーバー プロセスは、ユーザーのアクションによって終了されます。ただし、ASP.NETワーカー プロセスは、プロファイル中に自動的にシャットダウンされることもあります。これは、プロセスが実行されるシステムで **machine.config** ファイルの「processModel属性」セクションを以下に示すいずれかの方法で編集した場合に発生します。

- `requestLimit` 属性または `requestQueueLimit` 属性の値を、「Infinite」からセッション中にプロセスをシャットダウンするのに十分低い値に変更した。
- `timeout` 属性または `idleTimeout` 属性の値を、Infinite からセッション中にプロセスをシャットダウンするのに十分低い値に変更した。
- `memoryLimit` 属性の値を、セッション中にプロセスがリサイクルするのに十分低い比率に変更した。

プロセスがシャットダウンすると、DevPartner は最後のスナップショットを取って、セッション ファイルを生成します。DevPartner は、これらのセッション ファイルを以下のいずれかの方法で処理します。

- ASP.NET ワーカー プロセスがセッション コントロール ツールバーで選択されている場合、DevPartner は Visual Studio でセッション ファイルを開き、そのファイルをソリューションに追加します。この動作は、生成および終了される ASP.NET ワーカー プロセスのインスタンスごとに繰り返されます。
- ASP.NET ワーカー プロセスが選択されていない場合、セッション ファイルはキャッシュされます。IE クライアント プロセスが終了するか、IE プロセスのスナップショットが取られると、DevPartner は、IE のセッション ファイルを作成するほか、IE、IIS、およびその時点までに実行および終了されたASP.NETワーカー プロセスのすべてのインスタンスに関するデータを含む相関セッション ファイルを作成します。

分析セッションが終了すると、DevPartner は [プロジェクトにファイルを追加] ダイアログボックスを表示したままにするか、実行および終了された ASP.NET ワーカー プロセスのインスタンスに関するセッション ファイルを自動的に保存します。

ASP.NET ワーカー プロセスが頻繁に終了したために余分なセッション ファイルが生成されないようにするには、**`machine.config`** ファイルを編集して、プロセスの早過ぎる終了を避けるのに十分な高い値に、適切な属性を設定します。

メモ： **`machine.config`** ファイルを編集する前に、必ずバックアップ コピーを作成してください。

従来の **Web** スクリプト アプリケーションのデータを収集する

DevPartnerパフォーマンス分析を有効にして従来のWebスクリプト アプリケーションを実行すると、HTML ファイル、JScript ソース ファイル、VBScript ソース ファイルのデータが収集されます。スクリプト言語が、COM オブジェクトなどのプロセス内部またはプロセス外部のコンポーネントを呼び出す場合、これらのデータも収集できます。

マネージ .NET 言語の場合と同様に、実行時にスクリプト言語のインストゥルメンテーションが行われます。ただし、監視対象にするアンマネージ コンポーネント （COM オブジェクトなど）をインストゥルメントする必要があります。

メモ： 以下の手順は、従来の Web スクリプト アプリケーション固有の手順です。Visual Studio で開発した Web フォーム、XML Web サービス、および ASP.NET アプリケーションのデータを収集するには、その他の .NET アプリケーションと同様の方法で、該当するアプリケーションを実行します。

従来の Web スクリプト アプリケーションのデータを収集するには、**[** スタート **] > [** すべてのプログラム **] > [Micro Focus] > [DevPartner Studio] > [** ユーティリティ **] > [Web** スクリプトのパフォーマンス分析 **]** を選択します。

DevPartnerパフォーマンス分析がロードされた状態でInternet Explorer（IE）が開きます。IEの他に、データ収集を制御するために使用できるセッション コントロール ツールバーが表示されます。

DevPartnerを有効にしたIEのインスタンスで、パフォーマンス データを収集するHTMLページまたはWebアプリケーションを開き、アプリケーションを実行します。必要に応じて、アプリケーションの実行時にセッション コントロール ツールバーを使用してデータ収集の対象を絞ることも可能です。

Internet Explorerを終了するか、セッション コントロールを使用している場合は停止アクションを実行します。[セッションを保存]ダイアログ ボックスが表示されて、セッション ファイルが保存されます。

Webアプリケーションのデータ収集のヒント

分析用のデータを収集する前に、以下の操作を行うことをお勧めします。

- アプリケーションを数分間実行してウォームアップします。分析対象にするアプリケーションの部分が含まれるようにしてください。
- セッション コントロールのクリア アクションを実行して、その時点までに収集されたデータを破棄します。これにより、アプリケーションの起動時に発生した多くの一時的な初期化に対するデータ収集が排除されます。
- 分析するモジュールを実行します。
- セッション コントロール ツールバーの **[スナップショット]** をクリックします。これにより、コードの代表的なサンプルのパフォーマンス データを取得できます。
- HTML ページが完全にロードされる時間を考慮します。手動でテストする場合、ページがロードされるのを待ちます。自動化されたテスト用のスクリプトを作成する場合、ページが完全にロードされるまでの待機時間を組み込みます。ページが完全にロードされる前にページでコードが実行されると、プロファイリング データが歪曲されることがあります。
- キャッシュに注意してください。Web アプリケーションは、アプリケーション コードを実行する代わりに、キャッシュからページを返すことがあります。テストで同じ入力データを繰り返して使用する場合は、キャッシュにより結果が歪曲されます。キャッシュの影響を測定しないようにするには、**`machine.config`** ファイルを編集して次の行をコメントアウトすることにより、キャッシュを無効にします。

  ```
  <add name="OutputCache" type="System.Web.Caching.Output-
  CacheModule"/>
  ```

メモ： **`machine.config`** ファイルを編集する前に、必ずバックアップ コピーを作成してください。

Webサービス要件

DevPartnerパフォーマンス分析でWebサービスを検出させるためには、サービスが以下の要件を少なくとも1つは満たす必要があります。

- Webサービスが、**`System.Web.Services.WebService`** ベース クラスから派生していること。
- Web サービスに `WebService` 属性が含まれること。

DevPartner パフォーマンス分析で Web メソッドを検出させるためには、メソッドに `WebMethod` 属性が含まれている必要があります。

NMSourceから一時ファイルを削除する

IEやIISでスクリプトをパフォーマンス分析している間に、スクリプト ソースの一時コピーを保存するための**NMSource**フォルダが作成されます。このソースは、セッション データの分析時にセッション ウィンドウの[ソース]タブに表示されます。

このソースがいつ必要になるかわからないため、**NMSource**フォルダのファイルは削除されません。このフォルダは短期間でサイズが大きくなることがあります。特に、IISでサーバー プログラムを分析する場合には注意が必要です。

NMSourceフォルダ内のソース ファイルを定期的に参照して、アクティブではなくなったプロジェクトに関するファイルをすべて削除する必要があります。**NMSource**は、**¥Program files¥Internet Explorer**フォルダにあります。

IISでデータ収集を設定する

ローカル コンピュータまたはリモート サーバー上で実行中のIISまたはASP.NETアプリケーションのパフォーマンス データを収集するには、以下に説明する設定オプションを設定します。

IISがローカル システムで実行されている場合は、以下に説明するオプションをローカル システムで設定します。IISがリモート サーバー上で実行されている場合は、リモート システム上にDevPartnerと（リモート サーバー ライセンス）をインストールして、リモート システムで後述のオプションを設定する必要があります。

スクリプトのデバッグ

以下のオプションは、Internet Information Servicesマネージャの[規定のWebサイトのプロパティ]（特定のアプリケーションについては[Webアプリケーションのプロパティ]）で設定できます。以下のオプションはIIS 5.0または6.0に適用します。

[ホーム ディレクトリ]タブまたは[ディレクトリ]タブで、**[構成]**をクリックします。**[アプリケーションのデバッグ]**タブで、**[デバッグのフラグ]**を以下のように設定します。

- ASP のサーバー側のスクリプトのデバッグを有効にする
- ASPのクライアント側のスクリプトのデバッグを有効にする

ホスト プロセスの設定

Webアプリケーションが`dllhost`プロセスで実行される場合は、[アプリケーション保護]オプションを変更して、DevPartnerでのパフォーマンス分析データの収集を有効にします。これらのオプションは、Internet Information Servicesマネージャの[規定のWebサイトのプロパティ]（特定のアプリケーションについては[Webアプリケーションのプロパティ]）で設定できます。以下のオプションはIIS 5.0または6.0に適用します。

[ホーム ディレクトリ]または[ディレクトリ]タブの[アプリケーションの設定]セクションで、[アプリケーション保護]を以下のいずれかに設定します。

- 低（IISプロセス）：アプリケーションは`inetinfo`プロセスで実行されます。データ収集を有効にするとIISが再起動され、アプリケーションを実行すると、このプロセスからデータが収集されます。
- 高（分離プロセス）：アプリケーションは、`dllhost`の別のインスタンスとして実行されます。DevPartnerにより新しいプロセスが認識され、アプリケーションを実行するとデータが収集されます。

データの収集が終了したら、IIS を再起動して、プロセスから DevPartner データ収集を削除します。

Internet Explorer でデータ収集を設定する

Internet Explorer からパフォーマンス分析データを収集するには、**[** ツール **]** > **[** インターネット オプション **]** を選択します。**[** 詳細設定 **]** タブで、**[** スクリプトのデバッグを使用しない（**Internet Explorer**）**]** と **[** スクリプトのデバッグを使用しない（その他）**]** をオフに設定します。

サービスのデータを収集する

サービスに対してパフォーマンス分析セッションを実行するには、**DPAnalysis.exe** を使用します。**DPAnalysis.exe** を使用すると、コマンド ラインや XML 構成ファイルからセッションを直接実行できます。**DPAnalysis.exe** の詳細については、「コマンド ラインから分析を起動する」 283ページ）を参照してください。

COM および COM+ アプリケーションからデータを収集する

DevPartner を使用して、COM または DCOM コンポーネントを呼び出すアプリケーションのデータを収集できます。

アンマネージ COM と .NET オブジェクト（COM+）とを混合して使用するアプリケーションをプロファイルする場合は、アプリケーションの .NET 部分の行レベルのデータが収集されます。アンマネージ コード コンポーネントの行レベルのデータが、DevPartner のネイティブ C/C++ インストゥルメンテーションを使用してインストゥルメントされている場合は、このデータが収集されます。パフォーマンス データ収集のために Visual Basic COM オブジェクトの行レベルのデータを最初にインストゥルメントした場合は、このデータも収集できます。この場合は、パフォーマンス分析のインストゥルメンテーションを使用してプロジェクトをビルドします。

C++ オブジェクト、またはインストゥルメントされていないアンマネージ コード コンポーネントをプロファイルする場合は、COM インターフェイスと DLL エクスポートに基づいて、メソッドレベルのデータのみが収集されます。

再帰関数のデータを収集する

再帰を使用するアプリケーションのリテラル プロファイルでは、再帰関数が重複してカウントされます。DevPartner では、関数の時間をすでに測定しているかどうかを検出することによって、再帰関数のカウントの重複を防ぎます。再帰が検出されると、DevPartner は 1 回めの関数コールの時間測定を停止し、新規に累積を開始します。再帰関数のデータ収集が DevPartner によって処理される方法の詳細については、DevPartner Studio オンライン ヘルプを参照してください。

コール グラフを分析する

コール グラフとは、アプリケーションの複数のメソッドのコール関係を表したグラフのことです。コール グラフの使用については、すでにこの章の準備、設定、実行手順で説明しました。このセクションでは、コール グラフの使用に関する詳細について説明します。

セッション ファイルからコール グラフを表示するには、**[** コール グラフの表示 **]** ボタンをクリックするか、**[** メソッド リスト **]** タブでメソッドを選択して右クリックし、**[** コール グラフの表示 **]** を選択します。個別の [コール グラフ] ウィンドウが表示されます。

DevPartnerでは、特定のメソッド コールまでの一連のコールを示すコール グラフが、そのメソッドによって呼び出されるメソッドと共に表示されます。

ノードは、呼び出される順に左から右に順番に表示されます。コール グラフに最初に示される最初のノードは基本ノードです。これは選択したメソッドまたはオブジェクトを表します。ノードの左側のノードは、「上位ノード」と呼ばれます。ノードの右側のノードは、「下位ノード」と呼ばれます。

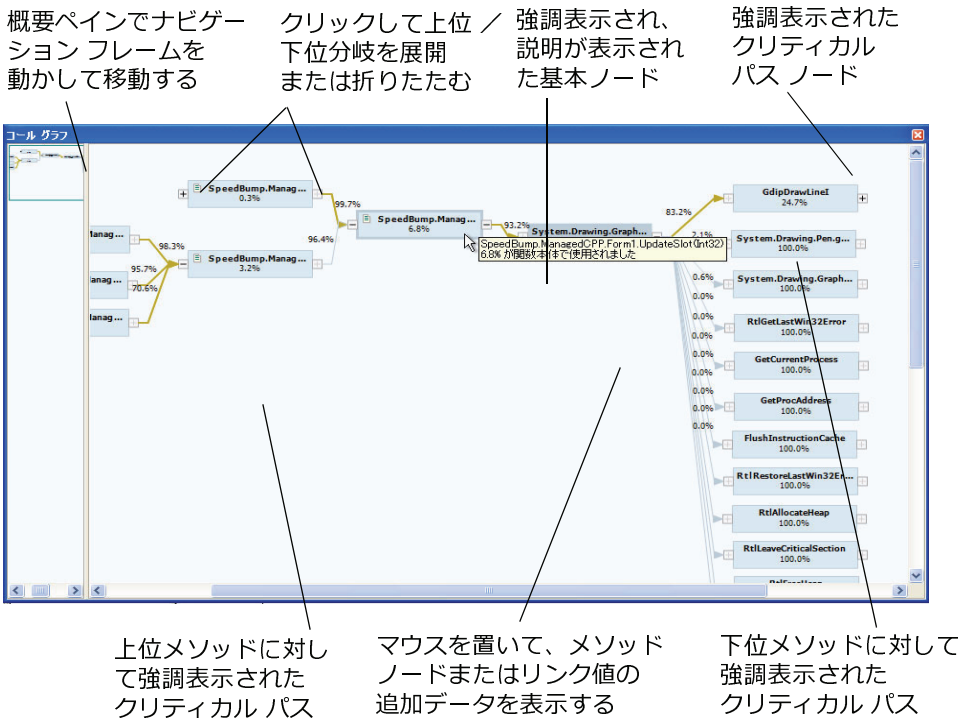


図6-10 コール グラフ

コール グラフは、以下の2つのフレームで構成されます。

- 左側のフレームには、コール グラフの概要が表示されます。これは、右側のフレームでコール グラフのノードが多すぎてスクロールしないと表示できない場合に、コール グラフ全体を表示できるので便利です。右フレームのノードを展開するか折りたたむと、概要が自動的に更新され、現在のビューが表示されるようになります。また、概要ペインでナビゲーション フレームを移動すると、右側のフレームに表示されたグラフの部分が変化します。概要を閉じるには、右フレーム内の任意の場所を右クリックして、**[** 概要を表示 **]** オプションの選択を解除します。

- 右フレームには、基本メソッド ノードの他、その呼び出し元または呼び出し先であるすべてのメソッドが表示されます。選択したノードの左右にあるノードを表示または非表示にするには、展開 ／折りたたみボックスを使用します。

  各ノードに表示される割合は、そのノードに費やされている時間の割合を表します。それぞれの下位ノードに続く線に表示される値は、その下位パスに費やされている時間を、上位ノードに費やされている合計時間に対する割合で表しています。

クリティカル パス

DevPartnerでは、コール グラフを表示すると、選択したメソッドとそのすべての下位メソッドに関するクリティカル パスが自動的に計算されます。クリティカル パスは、メソッドとそのすべての下位メソッドに費やされた時間の最大比率を表す一連のメソッド コールです。

コール グラフをナビゲートする

ノードをウィンドウの別の場所にドラッグすると、コール グラフの線は自動的に再描画されます。この機能は、画面上のメソッド数が多すぎる場合や、基本ノード、上位ノード、および下位ノードを最初に表示するときに使用される画面上の領域を削減したい場合に便利です。

デフォルトでは、下位ノードのみが展開されて表示されます。基本ノードの上位ノードは表示されません。基本ノードの上位ノードを表示するには、基本ノードの左側にあるプラス記号のアイコンをクリックします。フル パスを表示するには、プログラムによって最初に実行されるメソッド　通常は`Program Start`という名前）に達するまで、各上位ノードの左にあるアイコンを展開します。

ノードは、個別に選択することも複数個をまとめて選択することもできます。複数のノードをまとめて選択するには、あるノードを選択し、[Ctrl]キーまたは[Shift]キーを押しながら残りのノードを選択します。選択したノードはまとめてドラッグできます。

ソース コードを表示する

基本ノードのソース コードを表示するには、基本ノードを右クリックして、コンテキスト メニューの**[**メソッドの表示**]**オプションを選択します。表示できるのは基本ノードのソース コードのみです。

下位側の分析

コール グラフの下位　右）側を分析して、何を最適化すればよいかを調べます。

下位ノードを展開し、時間の多くを占めているのが基本メソッドであるか、下位メソッドであるかを分析します。

- 基本ノードに複数の並行する分岐がある場合は、1つめの下位メソッドに続くリンクに示される値が最も大きい分岐を探します。値が大きい分岐を最適化する方が、パフォーマンスの点で大きな効果が得られる可能性は高くなります。
- 基本メソッド自体の値が大きい場合は、基本メソッドの最適化を検討します。
- 基本メソッドに費やされる時間の多くが下位分岐によって費やされている場合は、その分岐で割合の値が大きい下位ノードを探します。

上位側の分析

コール グラフの上位　左）側を分析して、基本ノード分岐を最適化する価値があるかどうか、または基本ノードが呼び出される回数を削減またはゼロにできるどうかを判断します。

基本ノードの左にある上位ノードを展開します。特に、上位分岐に費やされる時間のうちのどの程度が基本ノードによって費やされているかを調べます。このことは、基本ノードまたはその下位メソッドを最適化する価値があるかどうかを判断する場合に役立ちます。複数の上位ノード、またはプログラムの実行全体から見て重要な上位ノードの時間の多くが基本ノードによって費やされている場合は、基本ノードの最適化を検討する価値があると考えられます。

基本ノードとその上位ノードの間のリンクに表示されている値はそれぞれ独立しており、加算されません。それぞれの割合の値は、基本ノードがその親に費やされる時間のどの程度を費やしているかを表します。

- 基本ノードに複数の上位ノードがあり、基本ノードに続くリンクの 1 つまたは複数の値が大きい場合は、その基本ノードを最適化する価値があると考えられます。
- 上位ノードに続くリンクの値が非常に小さい場合、基本ノードの分岐を最適化しても、上位メソッドのパフォーマンスにはほとんど影響ありません。
- 基本ノードが最適化に最もふさわしいかどうかを判断するには、その上位ノードを基本ノードとして選択した新しいコール グラフを表示します。このグラフには、元の基本ノードがその上位ノードのパフォーマンスに対して持つ重要性が、その上位メソッドの他の下位ノードと比較して示されます。

コール グラフの上位側または下位側のどちらを分析する場合でも、ノードを右クリックし、コンテキスト メニューを使用してそのメソッドのソース コードを表示すると、多くの時間が費やされている理由を判断できます。

セッションを比較する

プログラムのパフォーマンスを微調整するには、はじめに、実行に最も長い時間が費やされる場所を特定して、最もコスト高なコード部分を調整できるようにする必要があります。そのあと、その調整がパフォーマンスにどのような影響を及ぼすかを比較できます。

DevPartner では、1つのパフォーマンス セッションの結果を別の結果と比較して、実行する最適化が個々のメソッドおよびアプリケーションのパフォーマンス全体に与える影響を確認できます。

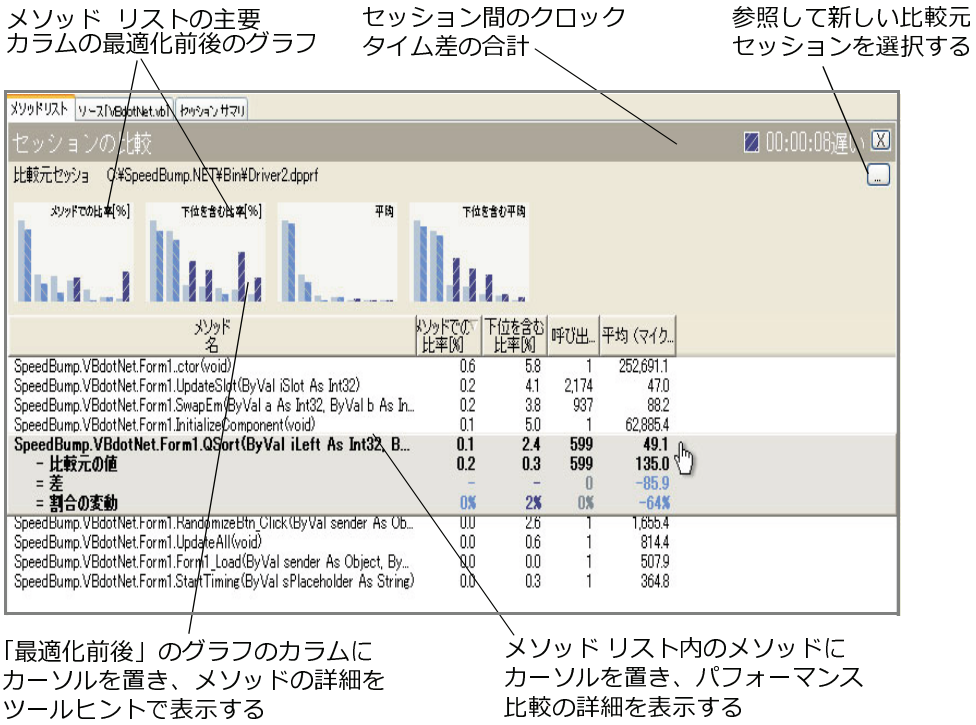


図 6-11 パフォーマンス セッションを比較する

セッションの比較を呼び出すには、セッション ウィンドウを開いたまま **[**比較**]** コマンドを切り替えます。**[**比較**]** コマンドは以下の場所で選択できます。

- ツールバーのボタン 
- パフォーマンス セッション ウィンドウのコンテキスト メニューのメニュー項目

[比較**]** コマンドを初めて呼び出すと、比較元セッションとするセッション ファイルを選択するよう求めるプロンプトが表示されます。デフォルトで、現在アクティブなセッションが現行セッションとして追跡されます。比較元セッションを選択すると、セッション ウィンドウにフレームが表示されます。このフレームはメソッド リスト内の任意のメソッド上に配置できます。フレームには、そのメソッドの比較元セッションと現行セッションのバージョン間の比較が表示されます。

比較するセッションを選択するときは、実行方法ができるかぎり同じものを選択するようにしてください。たとえば、デバッガで開始したセッションとデバッガの外部で実行したセッションを比較したりしないでください。比較をさらに正確にするには、セッション コントロール ファイルまたはセッション コントロール API を使用してデータ収集を制御してください。

比較ウィンドウの右上には、現行セッションと比較元セッションとの全般的な時間の差異が表示されます。その左側のグラフィックには、現行セッションの所要時間が比較元セッションのそれより長いか短いかが示されます。これは、同じ方法で実行されたアプリケーションのセッションをすばやく比較でき便利です。

比較では、選択したメソッドの現在の値、基本の値、2 つのバージョン間の差と差率が示されます。DevPartner のパフォーマンス フィルタを使用すると、セッション データのビューを変更できます。

セッション ウィンドウの上部の 4 つの棒グラフには、現行セッションの上位メソッドについて同じ情報のグラフ ビューが表示されます。

メソッド リスト内でコンテキスト メニューの **[**比較データのコピー**]** を選択すると、セッション比較データ ボックスの情報をコピーできます。このコマンドにより、データはクリップボードにコピーされます。

比較を終了するときは、**[Esc]** キーを押すか、**[**比較**]** アイコンをクリックします。

セッション比較の結果を解釈する

セッション比較には、現行セッションのメソッドと比較元セッションの同一メソッドの現在の値や基本の値の他、これらのメソッド間の差や差率が示されます。DevPartner では、色が使用されるため、現行セッションの値が比較元セッションの値より大きいか小さいかが一目でわかります。差と差率の値が濃い青色で示される場合、現行セッションの値は比較元セッションの値よりも優れています　高速です)。薄い青色で示される場合、現行セッションのパフォーマンスの値の方が低速になっています。

指定したメソッドのセッション間でコード変更によってどのような結果が生じたかがわかったら、セッションの他のメソッドを調べて、最初のコード変更による副作用を探します。個々のメソッドのパフォーマンスは向上したとしても、サイズの大きなプログラムのパフォーマンスが劣化している可能性があります。パフォーマンスの調整では、コードの構造に関する詳細な知識に代わるツールはありません。

セッション比較の結果を調べるときは、以下の点に注意してください。

- 割合とは 2 つの数字から得られる比率です。割合は、同じ合計値に比較して計算されている場合にのみ加算できます。

- 割合の値の 1 つが減少すると、割合の他のすべての値は必ず増加します。複雑なプログラムでは、このことに気付きにくい場合があります。割合の増加は、プログラムの他のすべてのメソッド間で平均する必要があるためです。
- サブプログラムのタイミングを解釈するには、そのサブプログラムが含まれるプログラムにおける役割を理解しておく必要があります。
- パフォーマンス測定は、その測定を実行したプログラム以外の状況では意味がありません。プログラムの動作を理解していなければ、プログラム変更の影響を総括することはできません。

プログラム内で最もコスト高なメソッドに対する変更が完了すると、他のコスト高なメソッドに取り組むことができます。

パフォーマンス データをエクスポートする

パフォーマンス データは、XML 形式または CSV 形式でエクスポートできます。データを XML または CSV 形式でエクスポートすると、独自またはサードパーティのソフトウェアを使用した分析、他のツールで作成したデータへの統合、データ ウェアハウスへのアーカイブなどが容易になります。

- DevPartner パフォーマンス セッション ファイル（拡張子 **.dpprf**）は XML 形式にエクスポートできます。保存されているパフォーマンス セッション ファイルが開いているときは、**[** ファイル **]** メニューで **[DevPartner** データのエクスポート **]** コマンドを使用できます。XML 形式でのエクスポートの詳細については、「分析データを XML にエクスポートする」（309 ページ）を参照してください。

  「コマンド ラインから分析データを XML にエクスポートする」（310 ページ）に説明されているように、コマンド ラインからデータをエクスポートすることもできます。
- メソッド リストのデータはカンマ区切りの CSV テキスト ファイルにエクスポートできます。**[** メソッド リスト **]** タブをクリックしてアクティブにし、エクスポートするカラムを表示して、メソッド リスト内を右クリックしてから、コンテキスト メニューで **[** メソッド リストのエクスポート **]** を選択します。カンマ区切りテキスト ファイルは、Microsoft Excel やその他のスプレッドシート アプリケーションで開くことができます。

データ収集を制御する

DevPartner で、アプリケーションの使用中にパフォーマンス データを収集するときの制御には、以下の 3 つの方法があります。

- セッション コントロール ツールバーを使用して、プログラムの実行中にデータ収集を対話的に制御できます。
- セッション コントロール ファイルを使用して、アプリケーション モジュール内の特定のメソッドに、セッション コントロール アクションを割り当てることができます。
- セッション コントロール API を使用して、プログラム内でデータ収集を制御できます。

セッション コントロール ツールバーまたはセッション コントロール API を使用すると、メソッド内の任意の場所でデータ収集を制御できます。セッション コントロール ファイルを使用する場合は、メソッドの先頭または終端でのみデータ収集を制御できます。

セッション コントロール ファイルとセッション コントロール API の使用については、「分析セッションの制御」（301 ページ）を参照してください。

コマンド ラインから分析する

データ収集を自動化したり、コマンド ラインから分析セッションを実行したりするには、DevPartner コマンド ライン実行ファイル **DPAnalysis.exe** を使用します。**DPAnalysis.exe** の使用方法の詳細については、「コマンド ラインから分析を起動する」（283 ページ）を参照してください。

パフォーマンス分析ビューアを使用する

DevPartner Studio には、Visual Studio とは独立してパフォーマンス セッション ファイルを分析するための軽量のパフォーマンス分析ビューアが付属しています。ビューアを起動するには、以下のいずれかを実行します。

- [スタート] メニューで、**[** すべてのプログラム **]** > **[Micro Focus]** > **[DevPartner Studio]** > **[** パフォーマンス分析ビューア **]** を選択します。
- Windows エクスプローラで、**.dpprf** セッション ファイルをダブルクリックします。
- コマンド ラインで **DPAnalysis.exe** を使用して、パフォーマンス分析セッションを実行します。セッション データがパフォーマンス分析ビューアに表示されます。

パフォーマンス分析ビューアで行える作業

セッション ファイルを開いたまま、パフォーマンス セッション データを表示、ソート、保存、印刷できます。さらに以下のことを行えます。

- メソッドのソース コードを表示する。
- [メソッド リスト] タブでデータをソートする。
- メソッドのコール グラフを表示する。
- セッション データを比較する。
- ファイルの内容を XML としてエクスポートする。
- メソッド リストの内容を CSV 形式でエクスポートする。

パフォーマンス分析ビューアで行えない作業

- パフォーマンスのためにアンマネージ アプリケーションをインストゥルメントする。
- パフォーマンス セッションを開始する。
- Visual Studio ソリューションにファイルを追加する。

コマンド ラインで生成されたセッション ファイルは、プロジェクトのソリューションに自動的には追加されません。外部で生成されたセッション ファイルは、Visual Studio に開いたソリューションに手動で追加できます。

.NET アプリケーションのパフォーマンス分析のヒント

パフォーマンス分析プロセスの生産性を高めるための戦略を以下に示します。

- ソース コードを分析する

[ソース メソッドの上位 **20]** フィルタを使用して、アプリケーションで問題のある部分を分離します。

すべてのシステム ファイル（ソース ファイル以外のファイル）を収集しないようにするには、**DevPartner** オプション ページの **[** 除外 - パフォーマンス **]** で **[** システム イメージを除外する **]** をオンにします。ソース コードを最適化したあと、このオプションをオフにすると、特に .NET Framework においてアプリケーションでシステム コードが使用される方法を調べることができます。

コール グラフを使用して、最もコスト高のメソッドを調べ、呼び出される下位メソッドに関連するコストを把握します。

複数のパフォーマンス セッションを実行して、異なるアルゴリズムやロジック変更の影響を比較します。

- **Framework** のコストを把握する

  [メソッド リスト **]** タブまたは **[** ソース **]** タブの **[** 下位を含む比率 **[%]]** を使用して、.NET Framework に費やされる時間を確認します。

  コール グラフの下位メソッドを調べて .NET Framework を分析し、どのコールのコストが高く、それがなぜかを解明します。

  作業を削減するか、.NET Framework の呼び出しが少なくなるようにアプリケーションを修正します。

- スタートアップ コストを把握する

  パフォーマンス データを収集する前に、**[** クリア **]** セッション コントロールを使用します。.NET Framework では、一時的な初期化が多数実行されます。この初期化によってパフォーマンスの結果が歪曲されないようにするには、プロファイルするすべての機能を実行してアプリケーションをウォームアップしてから、そのデータをクリアします。次に、同じ機能を実行するテストを行い、パフォーマンスをより正確に把握します。

- 測定対象を理解する

  パフォーマンス データの収集を開始する前に、アプリケーションがどのように動作するかを考慮します。たとえば、Web サービスまたは ASP.NET アプリケーションをプロファイルする場合は、Web キャッシュが結果に与える影響について考えます。テストの実行で同じデータが繰り返し入力される場合は、アプリケーションによってキャッシュからページがフェッチされるため、パフォーマンス データが歪曲されます。このような場合は、手間をかけて可変入力データを使用するか、簡単な方法としては、**`machine.config`** ファイルを編集して、テスト中はキャッシュをオフにします。以下の行をコメントアウトしてください。

  ```
  <add name=”OutputCache” type=System.Web.Caching.
  OutputCacheModule”/>
  ```

- 混合モードのアプリケーションのパフォーマンスを測定する

  アンマネージ C/C++ の .NET アプリケーションの部分を書き込むように選択できます。DevPartner では、アプリケーションのマネージ部分とアンマネージ部分の両方のパフォーマンス データを 1 回の実行で収集できます。ただし、アンマネージ コードは別個のファイル内にあり、データの収集前にこのコードをインストゥルメントする必要があります。この結果、パフォーマンス セッションを比較することにより、アプリケーション全体におけるアンマネージ コードとマネージ コードとの効果を比較できます。

- 分散アプリケーションに対してすべてのデータを収集する

ヒント：セッション コントロール ツールバーのプロセス リストを使用して、複数プロセスの分散アプリケーションにおける各プロセスのパフォーマンス スナップショットを取ります。

Web アプリケーション、複数層のクライアント／サーバー アプリケーション、または Web サービスを使用するアプリケーションのパフォーマンスを分析する場合は、すべてのリモート アプリケーション コンポーネントを分析に含めます。DevPartner のインストールを使用して、パフォーマンス データ収集をリモート システム上で設定します。アプリケーションでアンマネージ C/C++ コンポーネントが使われている場合は、データを収集する前に、そのコンポーネントをパフォーマンス分析でインストゥルメントします。スタートアップ コスト、.NET Framework のコスト、およびアプリケーション動作の認識に関する推奨事項は、サーバー側のコンポーネントのデータ収集にも同様に適用できます。

- マイクロプロファイリングの制限を理解する

アプリケーションのボトルネックを識別したあと、メイン アプリケーションの問題がある領域を複製した小規模のコード サンプルを作成すると便利です。パフォーマンスの比較を繰り返してサンプルのパフォーマンスを向上させてから、メイン アプリケーションにコードを戻します。この処理によって、アプリケーションの動作が速くなる場合があります。ただし、元のパフォーマンス テストに戻るまで、確実なことはわかりません。

- 実際の実行状態をシミュレーションする

コードが実行される方法は、アプリケーション メモリ フットプリント、マルチスレッド、スレッドの優先順位、プロセスのセキュリティ、ネットワーク待ち時間、サーバー負荷などの不測の事態の影響を受けるため、1 つのコンポーネントでパフォーマンスをテストしてもわからない可能性があります。アプリケーションのパフォーマンスを測定するには、実際にアプリケーションが使用されるのとなるべく近い状態でシミュレーションする必要があります。

Visual Studio Team System にデータを送信する

Visual Studio 2005 および 2008 については、選択した項目に関するデータをバグタイプの作業項目として Visual Studio Team System に送信します。Visual Studio 2010 では、選択した項目に関するデータを問題、バグ、または不具合タイプの作業項目として Team Foundation Server に送信します。カバレッジ分析セッション ファイルでは、[メソッド リスト] タブで選択したメソッドの作業項目の送信にアクセスします。作業項目を送信する場合、**[** メソッド リスト **]** タブで表示されるカラムからのデータが **[** 作業項目 **]** 形式で入力されます。**[** 作業項目 **]** として送信するメソッド データを変更するには、**[** メソッド リスト **]** に表示されるカラムを変更します。DevPartner Studio では、Team Explorer クライアントがインストールされており、Team Foundation Server に接続可能な場合に、Microsoft Visual Studio Team System がサポートされます。Team System のサポートの概要については、「Visual Studio Team System のサポート」 210 ページ）を参照してください。

パフォーマンス分析セッション ファイルでは、DevPartner パフォーマンス分析セッション ファイルの **[** メソッド リスト **]** タブで選択したメソッドのデータを、作業項目として Visual Studio Team System に送信できます。

作業項目を送信する場合、**[** メソッド リスト **]** タブの表示カラムのデータが **[** 作業項目 **]** フォームに挿入されます。**[** 作業項目 **]** として送信するメソッド データを変更するには、[メソッド リスト] に表示されるカラムを変更します。

第7章

詳細パフォーマンス分析

この章には、2つのセクションが含まれています。1つめのセクションでは、はじめてのユーザーを対象として、パフォーマンス エキスパートを使用する簡単な手順について説明します。2つめのセクションでは、DevPartner Studioパフォーマンス エキスパート コンポーネントを詳細に把握するための参照情報を示します。

パフォーマンス エキスパートに関するタスクベースの追加情報については、DevPartner Studioのオンライン ヘルプを参照してください。

パフォーマンス エキスパートとは

DevPartner Studioには、コード内のボトルネックの特定を容易にするパフォーマンス アナライザなどのアプリケーション開発を支援するために設計された多くの機能が含まれています。パフォーマンス エキスパートによって、以下のような解決が困難な問題のより詳細な分析を通して、マネージVisual Studioアプリケーションのパフォーマンス分析が一歩前進します。

- CPU ／スレッドの使用
- ファイルとディスクの I/O
- ネットワーク I/O
- 同期待機時間

メモ： パフォーマンス エキスパートではマネージ コードのみが分析されるため、この機能はDevPartner for Visual C++ BoundsChecker Suiteではサポートされていません。

パフォーマンス エキスパートでは、実行時にアプリケーションが分析され、コード内の問題があるメソッドが検出されます。メソッド内の個別の行の詳細を表示したり、呼び出しの親子関係を調査したりすることができるため、問題を修正するための最善の方法を決定する場合に役立ちます。アプローチ方法を決定する場合には、パフォーマンス エキスパートからソース コード内の関連がある行に直接ジャンプできるため、問題を迅速に修正できます。

パフォーマンス エキスパートはVisual Studioに統合されているため、アプリケーションの開発中にパフォーマンス エキスパートを使用してアプリケーションをテストできます。また、従来のコマンド ライン スイッチまたはXML構成ファイルを指定してDevPartnerコマンド ライン実行ファイル**`DPAnalysis.exe`**を使用することによって、コマンド ラインから、または自動テスト シナリオの一部としてパフォーマンス エキスパートを実行することもできます。詳細については、「コマンド ラインから分析を起動する」（283ページ）を参照してください。

パフォーマンス エキスパートは、ソフトウェア設計者、ソフトウェア開発者、および品質保証（QA）エンジニアによる使用を前提として設計されています。開発管理担当者が進行中のプロジェクトにおける問題を特定するために使用することもできます。

パフォーマンス エキスパートとパフォーマンス分析

パフォーマンス分析は、従来のパフォーマンス プロファイリングを補完するものであると考えてください。最初に、パフォーマンス分析を有効にしてアプリケーションを実行し、パフォーマンスの基本的な傾向を把握します。次に、パフォーマンス エキスパートを有効にして同じセッションを実行し、困難な問題、特に、ディスクI/O、ネットワークI/O、同期に関する問題の特性を理解します。問題が解決したら、パフォーマンス分析を有効にして再度アプリケーションを実行し、パフォーマンス分析のセッション比較機能を使用して性能の向上を確認します。パフォーマンス分析セッションの比較の詳細については、「セッションを比較する」 205ページ）を参照してください。パフォーマンス エキスパートをパフォーマンス分析と共に使用する詳細については、「パフォーマンス エキスパートとパフォーマンス分析を併用する」 141ページ）を参照してください。

すぐにパフォーマンス エキスパートを使用する

以下の準備、設定、実行の手順に従ってパフォーマンス エキスパートを使用できます。

すぐに使い始めるには、影付きのボックス内に示された手順に従います。影付きのボックス内に説明されている内容の詳細については、ボックスの下の追加説明を参照してください。

パフォーマンス エキスパートでアプリケーションを分析する場合、昇格されたシステム権限は必要ありません。パフォーマンス エキスパートでのアプリケーションの分析には、アプリケーションの作成とデバッグに使用するシステム権限で十分です。

準備：分析対象を検討する

分析中のアプリケーションの種類を判断するには、パフォーマンス エキスパート セッションを開始する前に実行する必要がある手順について検討します。

パフォーマンス エキスパートでは、マネージ アプリケーションからのみデータが収集されます。アプリケーションのパフォーマンス エキスパート データを収集するには、ソリューションに少なくとも1つのマネージ コード プロジェクト（たとえばC#、Visual Basic、またはマネージC++）を含める必要があります。また、ソリューションには、スタートアップ プロジェクトが含まれている必要があります。ソリューションに複数のスタートアップ プロジェクトが含まれている場合は、セッションで使用するスタートアップ プロジェクトを選択するように求めるプロンプトが表示されます。

以降の手順では、以下の事項を前提としています。

- 単一プロセスのマネージ アプリケーションをテストします。
- アプリケーションのビルドと実行が可能です。
- ソリューションには、少なくとも 1 つ以上のマネージ コード プロジェクトが含まれています。
- ソリューションには、スタートアップ プロジェクトが含まれています。

DevPartner メモリ分析でサポートされているすべてのプロジェクト タイプのリストは、「DevPartner Studio サポートされるプロジェクト タイプ」 169ページ）を参照してください。

パフォーマンス エキスパートでは、Visual Studio から実行された単一プロセス、または従来のコマンド ライン スイッチを使用して **DPAnalysis.exe** で実行された単一プロセスが監視されます。XML 構成ファイルを指定して **DPAnalysis.exe** を使用すると、セッション内の複数のプロセスやサービスのパフォーマンス エキスパート データを収集できますが、通常パフォーマンス エキスパート セッションでは単一プロセスのみを対象とすることをお勧めします。アプリケーションが複数のプロセスで実行される場合は、2 つめのプロセスを対象にしてアプリケーションを再実行します。**DPAnalysis.exe** の使用の詳細については、「データ収集を自動化する」（235 ページ）を参照してください。

パフォーマンス エキスパートを使用して、以下のような任意のマネージ Visual Studio アプリケーションのパフォーマンスを向上できます。

- ASP.NET Web アプリケーション
- ASP.NET Web サービス アプリケーション
- .NET Remoting サーバー アプリケーション
- Windows Forms クライアント アプリケーション
- COM+ などのサービス コンポーネント

パフォーマンス エキスパート セッションを開始する前に、収集するデータを決定します。アプリケーションのパフォーマンスについて検討します。特定の機能を使用すると、そのアプリケーションの速度が低下しますか。その場合は、パフォーマンス エキスパートを有効にしてアプリケーションを実行し、その機能を実行します。従来のパフォーマンス分析によって、データを読み書きするメソッド、またはネットワーク リソースにアクセスするメソッドで想定以上に時間がかかっていることが示されましたか。パフォーマンス エキスパートではディスク I/O とネットワーク I/O に関する詳細情報が提供されるため、パフォーマンス エキスパート セッションでその機能を対象にします。

アプリケーションにローカル クライアント プロセスとリモート サーバー プロセスが含まれている場合に、両方のプロセスを対象に分析を実行しますか。その場合、サーバー データを収集するには、最初にリモート コンピュータに DevPartner Studio と DevPartner リモート サーバー ライセンスをインストールする必要があります。サーバー側データを収集する前に、一部の IIS 設定が必要になる場合があります。

設定：プロパティとオプション

パフォーマンス エキスパート セッションに含めるコードを決定したら、いくつかのプロパティとオプションを設定して、データ収集を絞り込むことができます。

> この手順では、DevPartner のデフォルトのプロパティとオプションを使用できます。追加の設定は必要ありません。

ソリューションに複数のプロジェクトが含まれている場合は、ソリューション プロパティまたはプロジェクト プロパティを使用して、セッションのスタートアップ プロジェクトを選択したり、セッションから特定のプロジェクトを除外したりできます。DevPartner オプションを使用すると、表示オプションの変更、またはセッション コントロール ファイルを作成することによるデータ収集の管理を行うことができます。分析セッションの設定については、「プロパティとオプションを設定する」（225 ページ）を参照してください。

実行：パフォーマンス エキスパートのデータを収集する

分析対象を検討し、適切なプロパティとオプションを設定すると、パフォーマンス エキスパート データ収集の準備が完了します。

DevPartnerでは、Visual Studioの起動モデルがサポートされます。パフォーマンス エキスパート アイコンをクリックするか、[DevPartner] メニューの [デバッグを実行せずにパフォーマンス エキスパートを選択して開始] を選択すると、ソリューションがリビルドされ、アプリケーションのスタートアップ プロジェクトが起動されて、パフォーマンス エキスパート データの収集が開始されます。

図 7-1 パフォーマンス エキスパート ウィンドウによるデータ収集の制御

1 パフォーマンス エキスパート ウィンドウで左上のクリア セッション コントロールをクリックすると、スタートアップ データと初期化データがクリアされて、問題のある機能のみのデータが収集されます。

2 アプリケーションの速度が遅い部分を実行します。

3 アプリケーションの実行中にパフォーマンス エキスパート ウィンドウを観察します。グラフには、CPU処理時間のライン、および存在する場合にはディスク動作とネットワーク動作のラインが表示されます。これらのラインで急激な上昇が発生している場合は、潜在的な問題点を示している可能性があります。

4 気になる点がある場合は、スナップショット セッション コントロールをクリックします。パフォーマンス エキスパートのセッション ファイルが生成されて、Visual Studioに表示されます。

パフォーマンス エキスパート ウィンドウを使用する

リアルタイム グラフの使用

パフォーマンス エキスパートのリアルタイム グラフでは、アプリケーションの最新の30秒間の動作が提供されます。このグラフには、CPUの使用状況を反映したラインが常に表示されます。アプリケーションでディスクやネットワークへの読み書きが行われる場合、グラフにはディスクI/Oとネットワーク I/O用の別のラインが表示されます。

リアルタイム グラフを使用して、アプリケーションの動作を監視します。興味深い動き、たとえば、グラフ内の動作に急激な上昇が見られる場合は、スナップショット ボタンを使用してその時点での動作のスナップショットを取得することができます。逆に、興味深い動きが見られない場合は、[クリア]ボタンを使用して、その時点で収集されたデータをクリアします。

クリアとスナップショットの使用

クリア ボタンとスナップショット ボタンは、リアルタイム グラフの上、パフォーマンス エキスパート ウィンドウの左上に設置されています。これらのセッション コントロールを使用して、以下の動作を実行します。

- クリア – その時点までに収集されたデータまたは最後のクリア処理以降に収集されたデータをクリアします。クリアを使用して、データ収集に専念したり、セッション結果ファイルのサイズを最小化します。
- スナップショット – その時点までに収集されたデータまたは最後のクリア処理以降に収集されたデータを含むセッション結果ファイルを作成します。データ収集は継続されます。アプリケーションを実行させながら、複数のスナップショットを取得することができます。

カバレッジ メーターの使用

カバレッジ メーターは、リアルタイム グラフの下、セッション コントロール ウィンドウの左下に設置されています。カバレッジ メーターには、セッション内のその時点まで実行されたアプリケーション メソッドの割合が表示されます。カバレッジ メーターを使用して、パフォーマンス エキスパート下ですべてのコードがテストされたことを確認します。また、カバレッジ メーターをセッション コントロール処理と組み合わせて使用して、アプリケーションの特定の部分に特化したデータ収集を実施することができます。

メモ : 通常、パフォーマンス エキスパート セッションはデバッグなしで実行してください。デバッグなしのセッションの結果は解釈しやすく、デバッガによる処理のオーバーヘッドが含まれていません。アプリケーションをデバッガで実行した場合は、セッション中にブレークポイントにヒットした場合など、一部のタイミング値が予想より大きくなることがあります。このようなセッション ファイル内では、トレースなどのデバッグのみの機能の値が高くなる可能性があります。

5 データ収集が終了したら、アプリケーションを終了します。

アプリケーションを終了すると、DevPartner によって最終的なパフォーマンス エキスパート セッション ファイルが生成されます。1 つのセッション ファイルにすべてのセッション データをキャプチャする場合は、スナップショット セッション コントロールを使用する必要はありません。単純にアプリケーションを終了してください。

データを分析する

データのスナップショットを取る場合、またはデータの収集を終了してアプリケーションを終了する場合、DevPartner ではパフォーマンス エキスパート セッション ファイルが作成されます。アプリケーションで収集されたデータの最初のビューは、結果サマリの形式で表示されます。

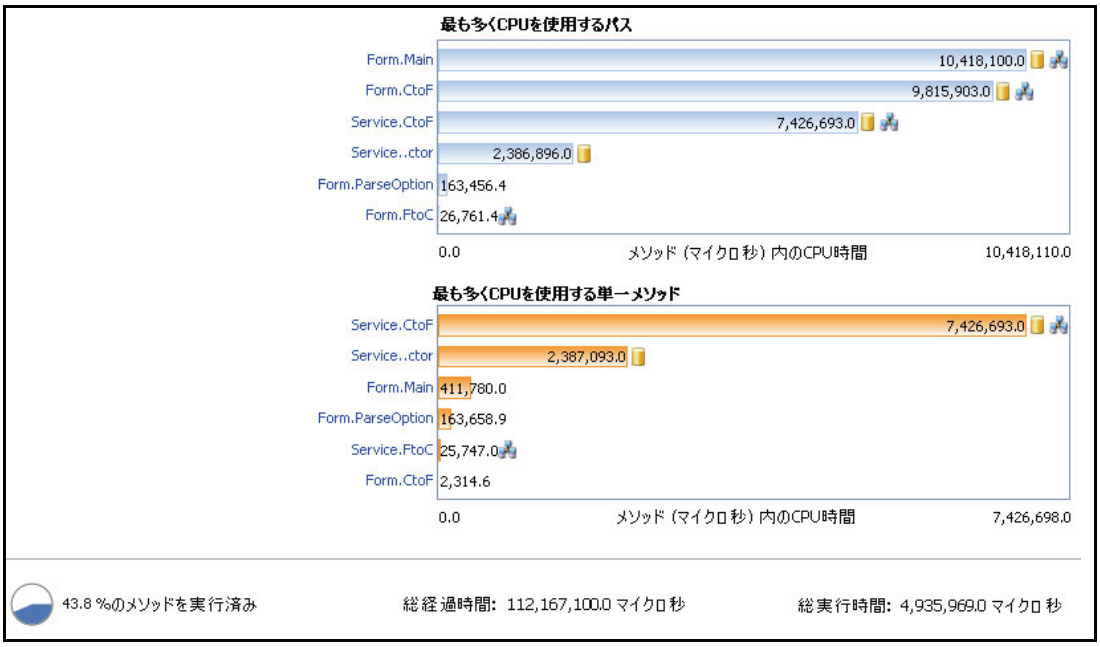


図 7-2 パフォーマンス エキスパートの結果サマリ

結果サマリには2つの棒グラフが含まれており、これらはアプリケーションの問題を解決するための2つのデータ分析方法を示しています。

ヒント: エントリ ポイント メソッドとは、他のソース コード メソッドによって呼び出されなかったソース コード メソッド、つまりソース コード実行のエントリ ポイントです。

- **[最も多く CPU を使用するパス]** には、セッション中に最も CPU サイクルを消費した上位のパス、またはメソッド コールのチェーンが表示されます。パスを分析することによって、メソッド実行の最もコストの高いパスを容易に特定することができます。以下の処理が可能です。
  - パフォーマンス低下の原因になっている下位メソッドを修正する
  - コストの高い下位メソッドを呼び出さないようにコーリング シーケンス内のその他のメソッドを変更する
- **[最も多く CPU を使用する単一メソッド]** には、CPU サイクルを消費した上位のメソッドが表示されます。メソッドを分析することによって、容易に個別の問題メソッドを特定して、それらを解決することができます。

棒グラフの端には、図 7-2 216ページ）のようにアイコンが表示されます。これらのアイコンは、メソッドによってディスク動作 またはネットワーク動作 が行われたことを示しています。

分析の開始点を決定する

結果サマリの2つの棒グラフを比較することから、セッション データの評価を開始します。

- **[最も多く CPU を使用するパス]** グラフの一番上のパスは、グラフ内の他のパスよりも大幅に長くなっていますか。
- **[最も多く CPU を使用する単一メソッド]** グラフの一番上のメソッドは、グラフ内の他のメソッドよりも大幅に長くなっていますか。

- メソッドの時間の値が、メソッドで実行される処理から考えて必要以上に長くなっていますか。
- 両方のグラフで、コストの高いメソッドとして同じメソッドが表示されていますか。

これらのことが該当する場合は、そのメソッドを調査します。

データを分析する前に、結果サマリからアクセス可能なデータ ビューへの移動方法を学習します。

1 結果サマリで、**[** 最も多く **CPU** を使用するパス **]** グラフの一番上のパスをクリックします。**[** パス分析 **]** ウィンドウが開きます。

[パス分析 **]** ウィンドウには、**[** コール グラフ **]** タブと **[** コール ツリー **]** タブが含まれており、下部に **[** ソース **]** タブと **[** コール スタック **]** タブが含まれています。

2 **[** サマリに戻る **]** をクリックして、結果サマリに戻ります。

3 結果サマリで、**[** 最も多く **CPU** を使用する単一メソッド **]** グラフの一番上のメソッドをクリックします。[メソッド] ウィンドウが開きます。

[メソッド] ウィンドウには、セッション内で実行されたメソッドのリストが含まれており、下部に **[** ソース **]** タブと **[** コール スタック **]** タブが含まれています。

4 **[** サマリに戻る **]** をクリックして、結果サマリに戻ります。

最も多く CPU を使用するパスの分析

[最も多く **CPU** を使用するパス **]** グラフからドリルダウンすると、セッション データをコール グラフとコール ツリーで表示できます。コール グラフには、エントリ ポイント メソッドによって呼び出された下位メソッドと、パス内の各メソッドの相対的な消費時間が表示されます。コール ツリーには、同じデータがツリー ビューで表示されますが、ユーザーが構成可能なデータ カラムの形式で各メソッドについての追加データが表示されます。

[最も多く **CPU** を使用する単一メソッド **]** グラフでメソッドを調査する場合、ユーザーが構成可能なデータ カラムを持つ、トラブルシューティングに役立つ **[** メソッド **]** テーブルが表示されます。**[** パス分析 **]** テーブル ビューと **[** メソッド **]** テーブル ビューを切り替えるには、任意の詳細ビューで **[** サマリに戻る **]** をクリックします。

パフォーマンス エキスパート セッション データの計算方法は、**[** 最も多く **CPU** を使用するパス **]** ビューと **[** 最も多く **CPU** を使用する単一メソッド **]** ビューとでは異なります。**[** 最も多く **CPU** を使用する単一メソッド **]** ビューにおける CPU 時間、ディスク I/O、ネットワーク I/O、および同期ロック待機時間の計算では、ソース コード下位メソッドの測定結果が除外されます。ソース コード下位メソッドを除外することによって、そのメソッド自体で長時間の CPU 時間を消費するメソッドに注目できます。一方、**[** 最も多く **CPU** を使用するパス **]** ビューでは、最もコストの高い実行パスを示すために、上位メソッドにソース コード下位メソッドによる影響が含まれます。

両方のビューにおけるすべての計算には、ソース コード メソッドから呼び出されたシステム メソッドまたは .NET Framework メソッドによる時間またはスループットが含まれています。通常、マネージ アプリケーションは、.NET Framework コードの実行に多くの時間を費やします。パフォーマンス エキスパートでは、システム データは呼び出し元のソース コードの行

に加算されます。これによって、コードにおいて.NET Frameworkとの対話を行う部分、つまりアプリケーション内の変更可能な部分に焦点が当てられます。

この手順では、最初に[パス分析]を使用して、最もコストの高いパスの実行中に呼び出された下位メソッドの相対的寄与率を分析します。

5 結果サマリで、[最も多くCPUを使用するパス]グラフ内のメソッドをクリックして、[パス分析]ビューにドリルダウンします。コール グラフが表示されていない場合は、左側の[コール グラフ]タブをクリックします。

クリティカルな（最もコストの高い）実行パスが強調表示されます。ここから、トラブルシューティングが始まります。

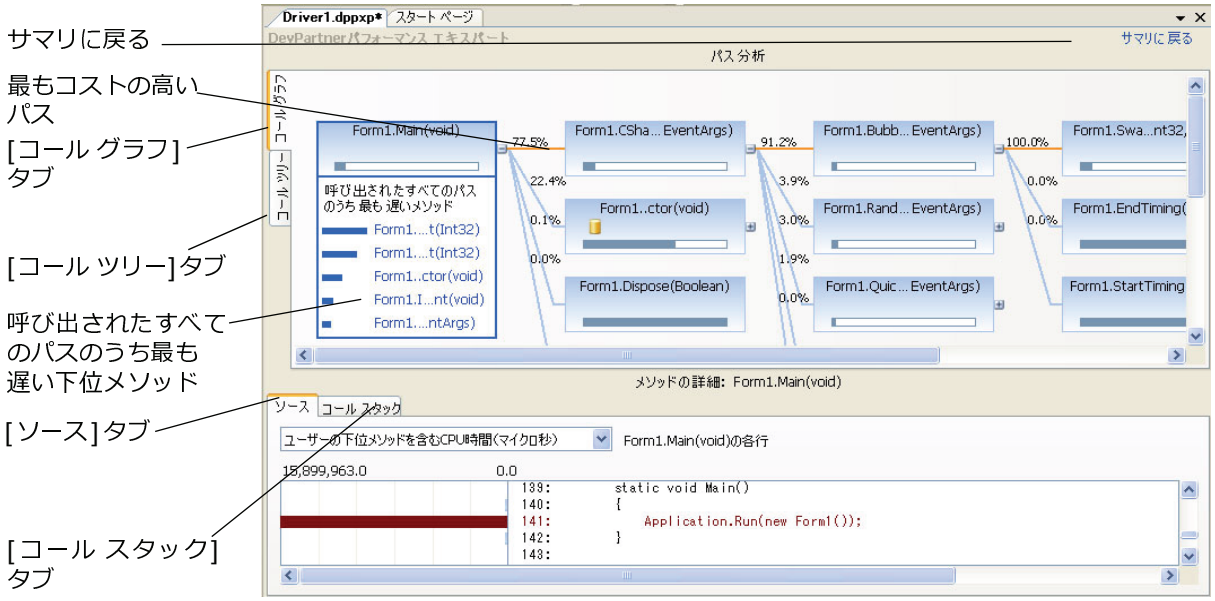


図 7-3 [パス分析]ウィンドウでのコストの高い実行パスの特定

コール グラフで、以下の操作を実行します。

6 パスを調査するには、ノード上のプラス記号をクリックして、パスを右側に展開します。

7 任意のメソッドをクリックして、パスに関係なく呼び出した下位メソッドのうち最も遅いメソッドのリストを表示します。このリストには、クリティカル パスには含まれない遅いメソッドも表示されます。

8 同じメソッドで実行されたパスごとの相対的寄与率を判断するには、選択されたメソッドをその下位パスに接続しているライン上の割合値を比較します。最もコストの高い（割合が高い）パスから調査します。

9 各ノードの下にある水平の棒グラフ上にマウス ポインタを置き、メソッド自体でかかった時間と下位メソッドの実行でかかった時間の割合を確認します。たとえば、図 7-4 (219ページ）を参照してください。

ほとんどの時間が下位メソッドでかかっている場合は、そのパスの調査を続行します。ほとんどの時間がメソッド自体でかかっている場合は、そのメソッドに焦点を絞って調査します。

コール グラフを使用すると、コール シーケンス中でコストの高いメソッドをすばやく検索でき、それらのメソッドに焦点を絞って調整作業を行うことができます。コール グラフ内のノードでは、下位メソッドによる影響が表示される以外に、メソッドにおける実行内容の理解に役立つ情報が提供されます。コール グラフの使用方法の詳細については、「コール グラフ」（231 ページ）を参照してください。

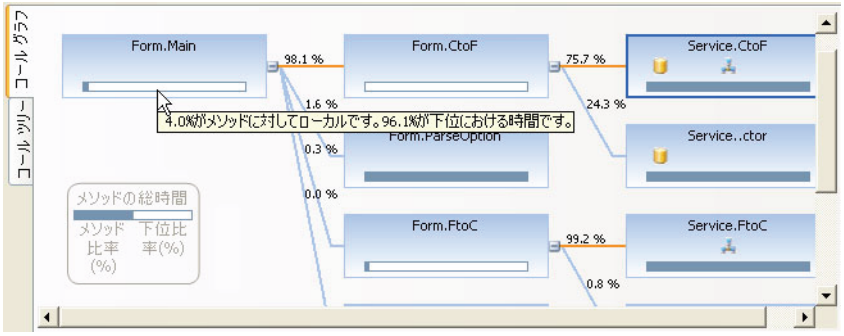


図 7-4 下位メソッドの影響の評価

10 調査予定のノードには、以下のアイコンが表示されていますか。

ディスク動作を示します。

ネットワーク動作を示します。

同期待機時間を示します。

11 表示されている場合は、これらのアイコンにマウス ポインタを移動して、動作の規模を確認します。調査に値する程度に動作の規模が大きい場合には、**[**コール ツリー**]** タブに切り替えて、より詳細に診断します。

12 コール ツリーを表示するには、セッション ファイル ウィンドウの左側の **[**コール ツリー**]** タブをクリックします。

ヒント : 「ユーザーの下位メソッド」という用語は、アプリケーション コードから呼び出されるシステム コードや .NET Framework メソッドではなく、アプリケーション自体のソース コード メソッドを指しています。

コール ツリーには、コール グラフと同様の情報が表示されますが、ツリー ビューの形式で表示されます。最もコストの高いパスが、テーブルのソート順序の位置によって示されます。デフォルトのソート カラムは、**[**ユーザーの下位メソッドを含む **CPU** 時間 **]** です。

前述のように、コール グラフでは、上位メソッドに対する下位メソッドの相対的寄与率に関する情報が提供されます。一方、コール ツリーでは、アプリケーション内のメソッドの実行内容について、より詳細なデータが表示されます。データは、ソート可能で、ユーザーが構成可能なデータ カラムの形式で表示されます。これらのデータ カラムは、カラム見出しを右クリックして、コンテキスト メニューから **[** 項目の選択 **...]** を選択することによって、[コール ツリー] ビューに追加できます。データ カラムを追加する前に、Visual Studio の [プロパティ] ウィンドウで、カラムに含まれるデータをプレビューできます。[プロパティ] ウィンドウを表示するには、**[**表示**]** > **[**プロパティ ウィンドウ**]** を選択します。

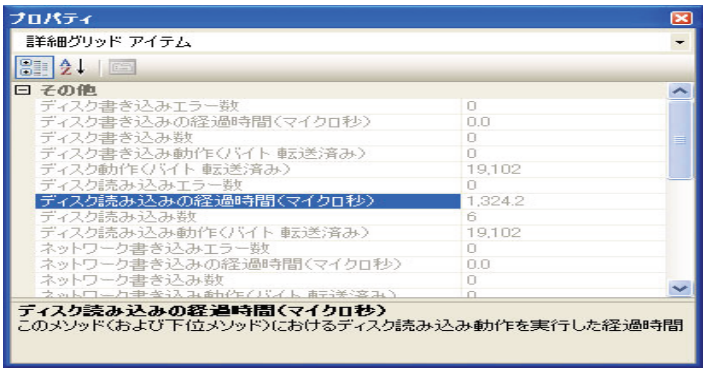


図 7-5 [プロパティ]ウィンドウでのメソッド データの表示

コール ツリーで、以下の操作を実行します。

13 同じメソッドで実行されたパスごとの相対的寄与率を判断するには、ユーザーの下位メソッドを含む **CPU** 時間カラムに表示されている下位パスごとの値を比較します。このカラムでソートされた場合（デフォルトのソート）、最もコストの高いパスがツリー ビューの一番上に表示されます。

14 コール ツリーはコール グラフと組み合わせて使用します。たとえば、コール グラフ内のコストの高いノードにネットワーク I/O アイコンが含まれている場合は、コール ツリーに切り替えて、ビューにネットワーク関連のデータ カラムを追加します。

[コール ツリー]ビューにデータ カラムを追加するには、カラム見出しを右クリックして、コンテキスト メニューから **[** 項目の選択 **...]** を選択します。

これらのデータ カラムには、ネットワーク経由の読み書きの回数、ネットワーク経由でのデータの読み書きにかかった時間、読み書きデータ量、および読み書きエラーの数が表示されます。

コール グラフ内のノードにディスク I/O または待機時間のアイコンが含まれている場合は、[コール ツリー]ビューにこれらのデータ カラムを追加します。このようにして、問題のあるノードでコストが高い理由をすばやく特定できます。

[コール グラフ]ウィンドウと [コール ツリー]ウィンドウの両方の下部に **[** ソース **]** タブと **[** コール スタック **]** タブが含まれています。

[ソース **]** タブによって、セッション中に実行されたコード行のコストを示すメトリクスと共に、アプリケーションのメソッドのソース コードを表示することができます。このタブを使用すれば、コンテキスト内のコストの高いコード行を表示して、改善が必要なコード行を容易に特定することができます。**[** ソース **]** タブには、図 7-6 （121 ページ）に示すようにメトリクス セレクタが含まれています。**[** パス分析 **]** ビュー内のデフォルトのメトリクスは、**[** ユーザーの下位メソッドを含む **CPU** 時間 **]** です。メソッドの処理内容に応じて、ディスク I/O、ネットワーク I/O、待機時間などの他のメトリクスを使用できる場合もあります。セレクタで新たなメトリクスを選択すると、[ソース]ペインが更新されて、そのメトリクスについて最もコストの高い行をソース コード内で特定できます。

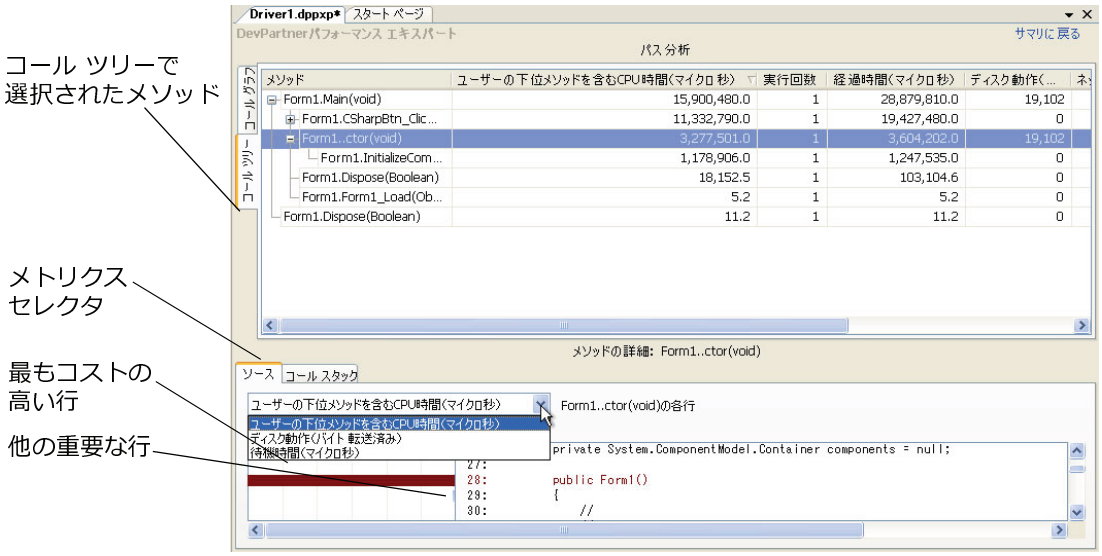


図 7-6 [ソース] タブにおける最もコストの高い行の特定

[ソース] タブはコール グラフおよびコール ツリーと組み合わせて使用します。

15 [コール ツリー] タブで、目的のメソッドを選択します（[コール グラフ] タブに戻っている場合は、メソッド ノードを選択します）。[ソース] タブを選択します。最もコストの高い行（[ユーザーの下位メソッドを含む CPU 時間] によって測定）が濃い赤で強調表示されます。[ソース] ペインをスクロールして、コストの高い他の行が青色で強調表示されていることを確認します。

16 選択したメソッドには、ディスク動作、ネットワーク動作、または待機時間動作がありましたか。コール ツリーでこのようなメソッドをすばやく検索するには、これらのデータ カラムの値が高いメソッドを検索します（コール グラフでは、メソッド ノードに表示されているディスク アイコン、ネットワーク アイコン、または待機時間アイコンを検索します）。

17 [ソース] タブの左上にあるメトリック セレクタ（図 7-6、121 ページ）を参照）を展開します。選択されたメソッドでディスク I/O、ネットワーク I/O、または待機時間が発生している場合は、リストに対応するメトリクスが表示されます。新しいメトリクスを選択し、ソース表示をスクロールして、そのメトリクスについて最もコストの高い行を検索します。コストの高いメソッドには、改善可能な行が複数存在する場合があります。

18 [ソース] タブ内で、修正する行を特定します。その行をダブルクリックして Visual Studio にソース ファイルを開き、編集します。

[コール スタック] タブによって、アプリケーションのコストの高いメソッドの複数のインスタンスまたは使用状況を表示することができます。各コール スタックは一意です。同じメソッド コーリング シーケンスを含むコール スタックが存在する場合があります。ただし、一部のコールは少なくとも 1 つのメソッド内の複数の行から行なわれました。

コール グラフまたはコール ツリーで異なるメソッドを選択すると、各メソッド内で最もコストの高い行まで [ソース] タブがスクロールされます。同様に、異なるメソッドを選択すると、[コール スタック] タブも更新されます。

たとえば、図 7-6　121 ページ）では、コール ツリーで下位メソッドが選択されています。この下位メソッド自体を修正することによってパフォーマンスの問題を解決する場合は、[ソース] タブを参照します。[ソース] タブでは、メソッド内で最もコストの高い行が強調表示されています。メソッドでディスク I/O やネットワーク I/O が行われた場合、または長時間の待機時間が発生した場合は、メトリクス セレクタを使用して、選択したメトリクスについて最もコストの高い行を検索します。修正内容を決定したら、そのソース行をダブルクリックして、Visual Studio で編集します。

また、アプリケーションで下位メソッドを呼び出す方法を変更することによってパフォーマンスの問題を解決する場合は、[コール スタック] タブに切り替えます。

[コール スタック] タブはコール グラフまたはコール ツリーと組み合わせて使用します。[コール スタック] タブには、選択されたメソッドを呼び出したすべてのパスが表示されるため、アプリケーション内でメソッドを使用するすべての方法に照らして、メソッドに加える変更を評価することができます。[コール スタック] タブを使用して、メソッドの中の最もコストの高いインスタンス　使用）を特定します。

19 [コール ツリー] タブで、目的のメソッドを選択します　[コール グラフ] タブに戻っている場合は、メソッドを選択します）。[コール スタック] タブを選択します。選択されたメソッドを呼び出している行が強調表示されます。

20 [コール スタック] タブの左上にあるスタック セレクタを展開します。スタック セレクタを使用して、最もコストの高いメソッド使用部分を特定します。

21 [コール スタック] タブ内で、修正する行を特定します。その行をダブルクリックして Visual Studio にソース ファイルを開き、編集します。

22 コストの高いメソッドを直接修正できない場合は、そのメソッドの呼び出し回数を減らすか、または全く呼び出さないようにコードを変更します。

[コール スタック] タブでは、コール ツリーで選択したメソッドが呼び出されたすべてのコール シーケンスまたはパスを調査できます。図 7-7　123 ページ）では、スタック セレクタに、各コール スタックで費やされた時間の割合が表示されています。これにより、最もコストの高い実行パスをすばやく検索できます。コール スタックを選択すると、スタックを構成するすべてのメソッドが表示され、各メソッド内でスタック上の次のメソッドを呼び出した行番号が表示されます。コール スタック内の任意のメソッドを選択すると、ソース ペインが更新されて、次の下位メソッドが呼び出された行が強調表示されます。呼び出し元のソース行をダブルクリックして、Visual Studio で編集します。

下位メソッドの呼び出し方ではなく、遅い下位メソッド自体を修正する場合でも、メソッドの [コール スタック] タブを調査してください。メソッドを変更する前に、アプリケーションでメソッドがどのように使用されているかをすべて理解することをお勧めします。パフォーマンス エキスパートを使用すると、この作業は簡単です。

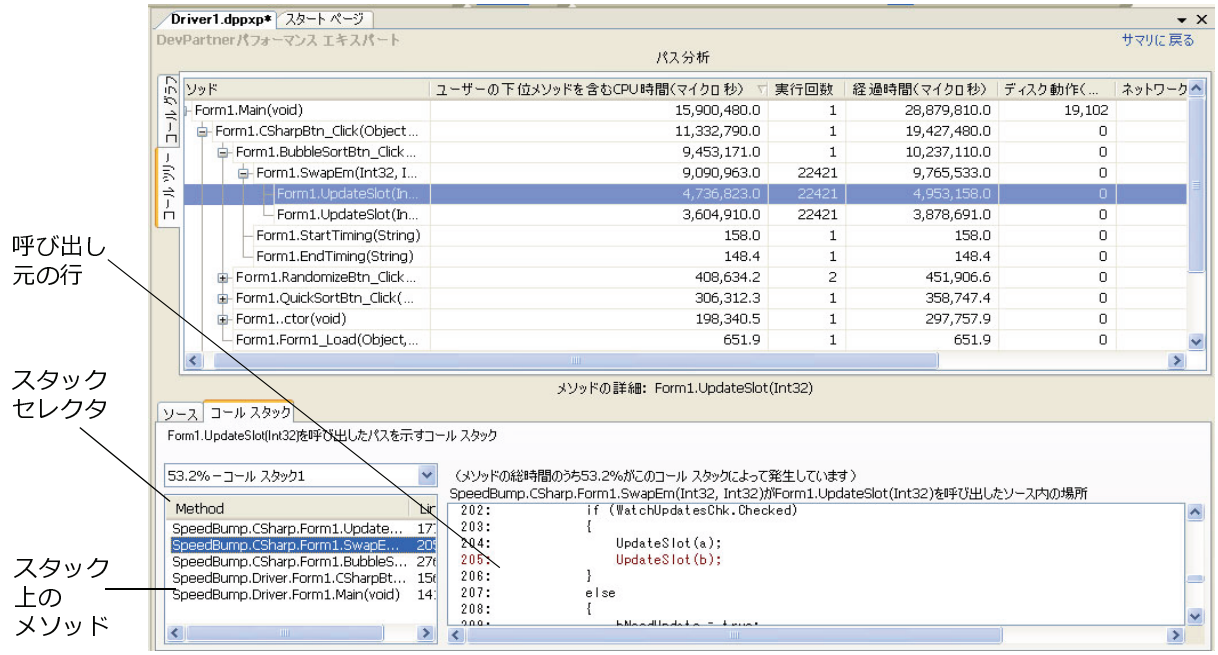


図 7-7 メソッドを使用したパスのうち最もコストの高いコール パスの特定

最も多く CPU を使用する単一メソッドの分析

結果サマリで [最も多く **CPU** を使用するパス] 棒グラフのセッション データをドリルダウンするだけでなく、[最も多く **CPU** を使用する単一メソッド] 棒グラフを使用してパフォーマンス エキスパート データが分析されます。たとえば、このグラフの一番上のメソッドで想定以上の時間がかかっている場合は、グラフ内でメソッドをクリックして、そのメソッドを即座に調査できます。

メソッドをクリックすると、アプリケーションを実行したときに実行されたメソッドが一覧表示される [メソッド] テーブルが表示されます。

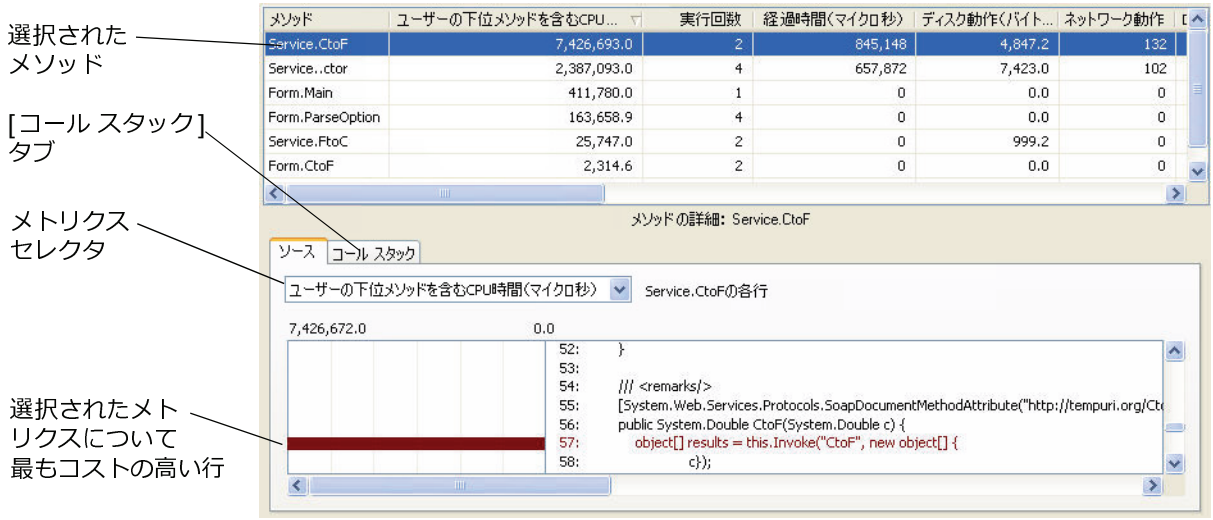


図 7-8 個別メソッドの影響の分析

23 **[**最も多く **CPU** を使用する単一メソッド **]** ビューにアクセスするには、**[**サマリに戻る **]** をクリックします。

24 結果サマリで、**[** 最も多く **CPU** を使用する単一メソッド **]** グラフ内のメソッドをクリックして、**[** メソッド **]** テーブルにドリルダウンします。

アプリケーション内で最もコストの高い個別メソッドが強調表示されます。デフォルトで、メソッドは、そのメソッドでかかったCPU時間でソートされます。この時間には、ユーザーの下位メソッドは含まれていませんが、システム コールは含まれています。

25 **[** メソッド **]** テーブルのカラムの選択をカスタマイズするには、カラム見出しを右クリックし、コンテキスト メニューから **[** カラムの選択 **...]** を選択します。

データ カラムを使用して、メソッドのパフォーマンスの最もコストの高い側面を特定します。

デフォルトで、[メソッド] テーブルは **[** ユーザーの下位メソッドを除く **CPU** 時間 **]** でソートされます。このメトリクスでは、メソッド自体のパフォーマンスに焦点が当てられています。一方、**[** 最も多く **CPU** を使用するパス **]** ビューでは、計算にユーザー　またはソース コード）下位メソッドが含まれています。

[ソース **]** タブは [メソッド] テーブルと組み合わせて使用します。メソッドを選択すると、**[** ソース **]** タブから直接最もコストの高いメソッド内のコード行に移動して、他の行の相対的なコストを表示することができます。最もコストの高い行は濃い赤で表示されます。メソッド内で時間が費やされた他の行は、薄い青で表示されます。

26 **[** ソース **]** タブのメトリクス セレクタを使用して、利用可能な各メトリクスについて最もコストの高い行を特定します。問題があるメソッドには、改善可能な行が複数存在する場合があります。

[コール スタック **]** タブは **[** メソッド **]** テーブルと組み合わせて使用します。**[** コール スタック **]** タブには、選択されたメソッドを呼び出したすべてのパスが表示されるため、アプリケーション内でメソッドを使用するすべての方法に照らして、メソッドに加える変更を評価することができます。**[** コール スタック **]** タブを使用して、メソッドの中の最もコストの高いインスタンス　使用）を特定します。

27 **[** ソース **]** タブまたは **[** コール　スタック **]** タブ内で、修正する行を特定します。速度の遅い行をダブルクリックして、Visual Studioで編集するためにそのソースを開きます。

28 コストの高いメソッドを直接修正できない場合は、そのメソッドの呼び出し回数を減らすか、または全く呼び出さないようにコードを変更します。

[最も多く **CPU** を使用する単一メソッド **]** テーブルおよび [メソッド] テーブルでの時間の割合の計算には、ソース コード下位メソッドで費やされた時間は含まれていませんが、システム下位メソッドで費やされた時間は含まれています。マネージ アプリケーションでは、.NET Framework内で非常に長い時間が費やされます。計算にシステム下位メソッドが含まれることによって、システム コードとの相互作用に問題のあるソース コード内のメソッドに焦点が当てられます。これは、マネージ コード アプリケーションにおいて特に重要な場合があります。

セッション ファイルを保存する

パフォーマンス エキスパート データを参照し終わったら、セッション ファイルを保存するか、または破棄できます。

1. Visual Studioで、保存されていないセッション ファイルを選択します。**[** ファイル **]** > **[**<ファイル名>**.dppxp** の保存 **]** を選択します。
2. セッションを保存する前にVisual Studioでセッション ファイル ウィンドウを閉じようとすると、開いているセッション ファイルを保存するように求めるプロンプトがDevPartnerによって表示されます。

DevPartnerでは、セッション ファイルはアクティブなソリューションの一部として保存されます。これらのファイルは、ソリューション エクスプローラのDevPartner Studio仮想フォルダに表示されます。パフォーマンス エキスパートのセッション ファイルには拡張子 **.dppxp** が付きます。

デフォルトでは、セッション ファイルはプロジェクトの出力フォルダに保存されます。ファイル名は、デフォルト フォルダに配置されているファイル名に自動的に番号を追加する形式（たとえば、**MyApp.dppxp**、**MyApp1.dppxp**）に設定されます。セッション ファイルをデフォルト フォルダ以外の場所に保存した場合は、ファイル名はユーザーが管理することが必要です。

出力フォルダがないプロジェクトの場合（Visual Studio 2005 Webサイト プロジェクトなどの場合）、ファイルはプロジェクト フォルダに物理的に保存されます。

Visual Studioの外部で生成されたセッション ファイルは、自動的にはプロジェクトのソリューションに追加されません。外部で生成されたセッション ファイルは、Visual Studioに開いたソリューションに手動で追加できます。

これで、この章の準備、設定、実行のセクションは終了です。これで、メモリ分析セッション実行のメカニズムの基礎が理解できました。追加情報については、引き続きこの章の残りの部分を参照してください。タスクベースの情報については、**DevPartner** のオンライン ヘルプを参照してください。

プロパティとオプションを設定する

パフォーマンス エキスパート セッションを開始する前に、データ収集を微調整して、特定のタイプの情報を含めるか、または除外することをお勧めします。ソリューション プロパティ、プロジェクト プロパティ、およびDevPartnerオプションを使用して、分析セッションをより目的に適したものにします。

ソリューション プロパティ

パフォーマンス エキスパートに影響を与えるソリューション レベルのプロパティを表示するには、ソリューション エクスプローラでソリューションを選択し、[F4]キーを押して、[プロパティ]ウィンドウを表示します。

図 7-9 ソリューション プロパティ

以下のソリューション プロパティがパフォーマンス エキスパートに影響を与える可能性があります。

- **.NETから収集 – [False]**に設定されている場合でも、パフォーマンス エキスパートを有効にしてマネージ アプリケーションを実行すると、このプロパティは上書きされます。パフォーマンス エキスパートでは、常にマネージ アプリケーションからデータが収集されます。
- スタートアップ プロジェクト – ソリューションに複数のプロジェクトが含まれている場合は、スタートアップ プロジェクトを変更できます。スタートアップ プロジェクトのプロジェクト プロパティによって、セッション内のすべてのアクティブなプロジェクトのデータ収集が制御されます。

ソリューションには、スタートアップ プロジェクトを含める必要があります。ソリューションに複数のスタートアップ プロジェクトが含まれている場合は、分析の開始前に、セッションで使用するスタートアップ プロジェクトを選択するように求めるプロンプトが表示されます。

このプロンプト ダイアログには、ソリューション プロパティの **[**共通プロパティ**]** > **[**スタートアップ プロジェクト**]** ページにある **[**アクション**]** が **[**開始**]** に設定されているプロジェクトだけが表示されます。目的のスタートアップ プロジェクトがプロンプト ダイアログに表示されない場合は、ソリューション プロパティ ページを開き、該当プロジェクトの **[**アクション**]** を **[**開始**]** に設定します。後続のセッションに新しいスタートアップ プロジェクトを選択する場合は、新しいスタートアップ プロジェクトのプロパティを見直して、データ収集オプションが正しいことを確認してください。

プロジェクト プロパティ

プロジェクト レベルのプロパティを参照するには、ソリューション エクスプローラでプロジェクトを選択して、ソリューション内のプロジェクトに設定可能なプロパティを参照します。

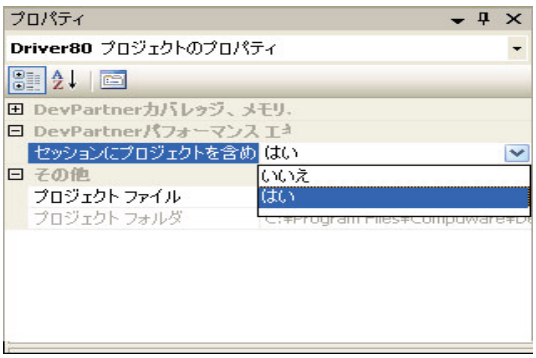


図 7-10 パフォーマンス エキスパートのプロジェクト プロパティ

パフォーマンス エキスパートに影響を与えるプロジェクトレベルのプロパティは、以下のとおりです。

- セッションにプロジェクトを含める – パフォーマンス エキスパート データ収集からプロジェクトを除外するには、**[** いいえ **]** を選択します。

オプション

パフォーマンス エキスパート セッションに対する DevPartner のオプション設定を確認するには、**[**DevPartner**]** > **[** オプション **]** > **[** 分析 **]** を選択します。

- **[**表示**]**オプションを使用すると、データ表示時の精度、目盛り、および単位を設定できます。
- **[** セッション コントロール ファイル **]** オプションを使用すると、アプリケーションやモジュールの実行時に DevPartner によって収集されるデータを制御するためのルールとアクションのセットを作成できます。セッション コントロール ファイルの詳細については、「Visual Studio 内でセッション コントロール ファイルを作成する」（301 ページ）を参照してください。

[環境 **]** > **[** フォントと色 **]** などのその他の Visual Studio オプションも、DevPartner の機能に影響を与えます。

パフォーマンス エキスパートを使用してアプリケーションの問題を検出する

パフォーマンス エキスパートを使用すると、以下の重要な領域において、マネージ Visual Studio アプリケーションの問題を特定できます。

- CPU ／スレッドの使用（待機の問題や同期の問題を含む）
- ファイルとディスクの I/O
- ネットワーク I/O
- 同期待機時間

Visual Studio から実行する場合、パフォーマンス エキスパートでは、一度に 1 つのプロセスのみが分析されます。選択したプロセス内で実行されるすべてのマネージ スレッドのデータがレポートされます。他のプロセスを分析するには、2 つめのプロセスを選択して、パフォーマンス エキスパートを再実行します。パフォーマンス エキスパートでは、複数のコンピュータにまたがる分散アプリケーションを分析することもできます。リモート データ収集の詳細については、「分散アプリケーションのデータを収集する」（237 ページ）を参照してください。

DevPartner では、Visual Studio の起動モデルがサポートされます。パフォーマンス エキスパート アイコンをクリックするか、**[**DevPartner**]** メニューの **[** デバッグを実行せずにパフォーマンス エキスパートを選択して開始 **]** を選択すると、アプリケーションのスタートアップ プロジェクトがすぐに起動され、パフォーマンス エキスパート データの収集が開始されます。

アプリケーションのパフォーマンス エキスパート データを収集するには、ソリューションに少なくとも 1 つのマネージ コード プロジェクト（たとえば C#、Visual Basic、またはマネージ C++）を含める必要があります。また、ソリューションには、スタートアップ プロジェクトが含まれている必要があります。詳細については、「プロパティとオプションを設定する」（125 ページ）を参照してください。

セキュリティ例外が発生した場合

マネージ アプリケーションのデータ収集を試みるときにセキュリティ例外メッセージが表示される場合は、セキュリティ ポリシーによりコードの DevPartner インストゥルメンテーションが行われていないことを示しています。デフォルトでは、アセンブリをプロファイルするに

はSkipVerification権限が必要です。コードが実行されるポリシーの権限セットからこの権限を削除したり、この権限を無効にするような命令型のセキュリティ宣言をアセンブリに追加した場合、アセンブリをプロファイルできなくなります。

この状態を修正するには、以下の2つの方法のいずれかを使用して、確実にプロファイリングを実行できるようにします。

- 以下のグローバル環境変数を設定して、アプリケーションのプロファイルを再試行します。

  ```
  NM_NO_FAST_INSTR=1
  ```

  この方法を使用すると、問題を回避することができますが、パフォーマンスが若干低下します。

- .NET Framework構成ツールのMMCスナップインを使用してアセンブリのポリシーを変更するか、またはアセンブリ内のすべての強制セキュリティ宣言を一時的に削除します。

Visual Studioのセキュリティ ポリシーの詳細については、Visual Studioオンライン ヘルプの『.NET Framework Developers Guide』を参照してください。

下位メソッドの集計

パフォーマンス エキスパート セッション データの計算方法は、**[**最も多く**CPU**を使用するパス**]**ビューと**[**最も多く**CPU**を使用する単一メソッド**]**ビューとでは異なります。[最も多くCPUを使用する単一メソッド]ビューにおけるCPU時間、ディスクI/O、ネットワークI/O、および同期ロック待機時間のデータの計算では、ソース コード下位メソッドの測定結果が除外されます。一方、[最も多くCPUを使用するパス]ビューでは、上位メソッドにソース コード下位メソッドによる影響が含まれます。

両方のビューにおけるすべての計算には、ソース コード メソッドから呼び出されたシステム メソッドまたは.NET Frameworkメソッドによる時間またはスループットが含まれています。通常、マネージ アプリケーションは、.NET Frameworkコードの実行に多くの時間を費やします。パフォーマンス エキスパートでは、システム データは呼び出し元のソース コードの行に加算されます。これによって、コードにおいて.NET Frameworkとの対話を行う部分、つまりアプリケーション内の変更可能な部分に焦点が当てられます。

セッション データの収集と分析の詳細は、以下の「使用シナリオ」を参照してください。

使用シナリオ

パフォーマンスの問題を解決するための一般的な方法は、以下の手順で構成されています。

1 問題のあるメソッド内で最も遅い行を特定して、最適化します。

2 その行を最適化できない場合は、その行を削除するか、または実行回数を減らします。

最も単純な場合は、たとえばパフォーマンス分析を使用してメソッド内の最も遅い行を特定し、その行を最適化するか、またはコール回数を減らすことができます。ただし、実際のアプリケーション開発では、より複雑な原因が多くの問題に存在します。最も遅いメソッドを特定することはできても、実行速度が遅くなる原因がメソッド内の複数の行の組み合わせであることもあります。このような場合は、より対象を絞ったデータを使用することによって、問題をすばやく解析できます。

たとえば、アプリケーション内の最も遅い部分で大量のネットワーク I/O が発生している場合は、以下のメトリクスを使用すると、問題の性質の理解に役立つ可能性があります。

- ネットワーク経由の読み書きの合計回数
- 読み書きバイト数
- 読み書きエラーの数
- 読み書き動作の経過時間

アプリケーションで大量のディスク I/O が行われている場合は、読み書きデータ量や、読み書き動作の効率性に関するメトリクスが役立ちます。パフォーマンス エキスパートでは、これらのデータがレポートされます。

パフォーマンス エキスパートを使用して、CPU やスレッドのパフォーマンス、ディスク I/O、ネットワーク I/O、および同期待機時間を分析できます。以下の例では、パフォーマンス エキスパートを使用してアプリケーションのパフォーマンスを向上する方法について説明します。

特定可能なパフォーマンスの問題

シナリオ：ユーザビリティのテスターから、アプリケーション内の特定の操作の実行速度が遅すぎるというレポートがありました。開発者であるあなたは、完了に長時間かかる遅い操作の原因となっているソース コード内の部分を特定して、その部分を修正する必要があります。

「実行：パフォーマンス エキスパートのデータを収集する」（213 ページ）に説明されているように、パフォーマンス エキスパートを有効にしてアプリケーションの最も遅い部分を実行したとします。セッション ファイルを調査すると、**[** 最も多く **CPU** を使用する単一メソッド **]** グラフの一番上で、実行に最も長い時間がかかったメソッドを即座に確認できます。ただし、複雑なアプリケーションでは、最も遅いメソッドではなく、そのメソッドよりも速度が速い複数のメソッドのシーケンスでよりパフォーマンスが低下している場合があります。最も遅いコール シーケンスは、**[** 最も多く **CPU** を使用するパス **]** グラフに表示されます。両方のグラフに表示されているメソッドはありますか。両方のグラフに表示されているメソッドは、調査する価値があります。

また、グラフ内の一部のメソッドには、メソッド内でのディスク I/O 動作またはネットワーク I/O 動作を示すアイコンが表示されています。これらのインジケータによって、メソッド内で行われている処理の種類の概要を理解できます。

 ディスク動作

 ネットワーク動作

パフォーマンス エキスパートの結果サマリの下部には、**[** 総経過時間 **]** と **[** 総実行時間 **]** が表示されます。実行時間が経過時間と比較して非常に短く、アプリケーションの実行方法から考えてこの差異が単にユーザー入力を待機したことが原因でないことが明らかである場合には、アプリケーション内に想定した以上の時間ロックを待機しているメソッドがないかどうかをチェックします。

最初に **[** 最も多く **CPU** を使用する単一メソッド **]** グラフの一番上のメソッドを調査するとします。CPU 使用率には、プロセッサを集中的に使用する計算、ディスク I/O、ネットワーク I/O、非効率な同期オブジェクトの使用など、数多くの要因が影響します。同様に、待機時間にも、メソッドが待機しているリソースが同一プロセス内で共有されている、または外部プロセスと共有されているなどの複数の原因があります。アプリケーション内で何が行われているかをすばやく判断するにはどうすればよいでしょうか。

[最も多く CPU を使用する単一メソッド]グラフの一番上のメソッドをクリックして、そのメソッドの[メソッド]詳細ビューを開きます。[メソッド]テーブル内のカラムのデータを確認します。この情報は、メソッドの処理内容を判断する場合に役立ちます。グラフでこのメソッドにディスク動作のアイコンが表示されていた場合は、テーブルを右クリックし、[項目の選択...]ダイアログ ボックスを使用して、テーブルにディスク関連のすべてのカラムを追加します。これにより、メソッドで読み書きエラーが発生している、メソッドで少量のデータを書き込むのに長時間かかっている、メソッドが何度も実行されている、などの原因を発見できます。

[メソッド]ウィンドウの下半分にある[ソース]タブには、テーブルで選択した任意のメソッドのソース コードが表示されます。テーブル内でメソッドをクリックすると、CPU 時間を最も長く消費した行までソースが自動的にスクロールされて、その行でかかった時間が表示されます。このビューには、CPU 時間を使用した他の行もグラフィカルに表示されます。

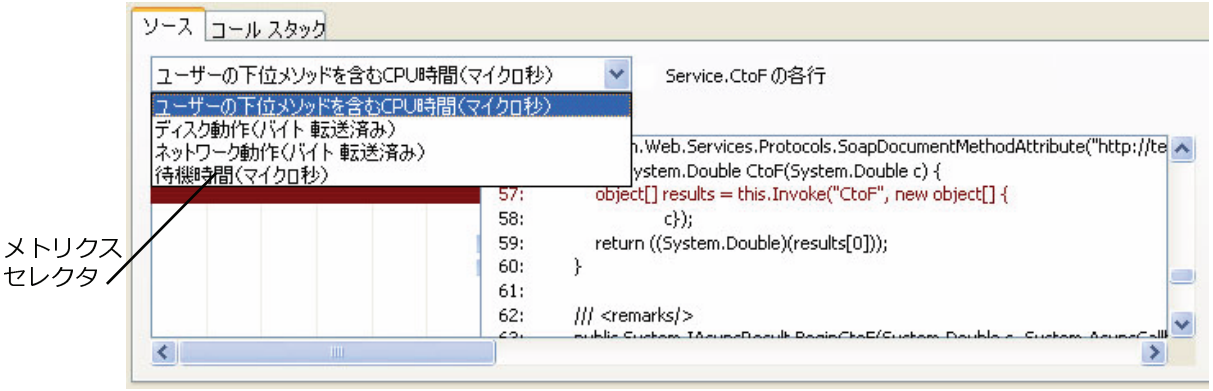


図 7-11 [ソース]タブでの問題のある行の特定

メソッドでディスク I/O またはネットワーク I/O が実行されたり、待機時間が発生したりしている場合には、左上のメトリクス セレクタを展開し、表示されるメトリクスを選択して、そのメトリクスについて最もコストの高い行を即座に特定できます。たとえば、ドロップダウン リストから[ディスク動作]を選択すると、最も多くのバイト数が転送された行に即座に移動して、メソッド内の他の行における相対的なディスク動作を確認できます。メソッドで待機時間が発生している場合は、[待機時間]のビューもチェックします。長時間の待機時間が発生している行を確認します。各ビューでは、デフォルトで最もコストの高い行が選択されます。メソッド内の行についてこれらのビューを比較することによって、従来のデバッグ方法よりもはるかに迅速に、修正対象とする箇所を絞り込むことができます。

修正する適切な行を特定したら、その行をダブルクリックして、Visual Studio 内でソース コードのその行に移動します。

問題の修正方法が明らかでない場合は、[コール スタック]タブをクリックして、アプリケーション実行中にメソッドがどのように使用されているかをすべて確認します。問題があるメソッドは、複数のパスから呼び出されていますか。複数のパスから呼び出されている場合は、メソッド内で最も時間がかかっているコール スタックを調査します。

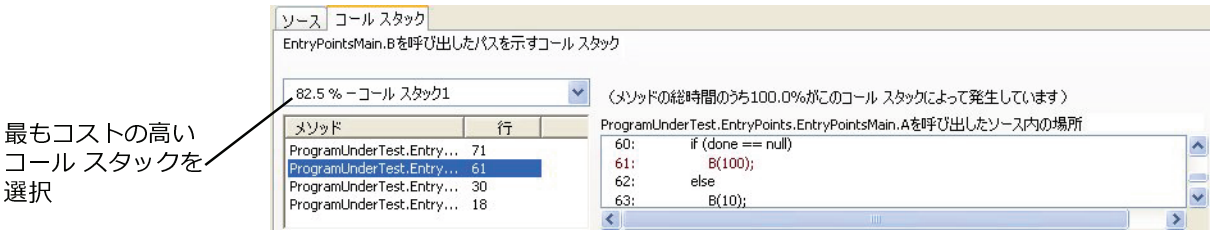


図 7-12 最もコストの高いコール スタックの選択

ヒント : パフォーマンス エキスパートでは、コール スタック内のいずれかのメソッドが異なる場合 (または同じメソッド内の呼び出し元の行が異なる場合) は、それぞれ独自の上位分岐として記録されます。

最初に、最も高い割合で呼び出した上位パスを確認します。コードを変更して、コールを削除するか、またはコール回数を減らします。[コール スタック] タブには、ソース コードのビューが含まれています。スタック内のメソッドを選択すると、スタック内の次のメソッドが呼び出された行まで自動的にソースがスクロールされます。ダブルクリックするとその行が Visual Studio に開かれるため、必要に応じてすばやくコール シーケンスを変更できます。コードを変更したら、再度パフォーマンス エキスパートを有効にしてアプリケーションを実行し、性能の向上を確認します。

アプリケーションにおける拡張性の問題

シナリオ : 新しい **Web** アプリケーションは、自分のコンピュータでテストしたときには問題なく動作します。ただし、他のユーザーもアプリケーションにアクセスするようになると、動作が非常に遅くなります。開発終了期限が迫っています。問題がある点をすばやく特定するにはどのようにすればよいでしょうか。

ロード テスト ツールを使用して、アプリケーションに負荷をかけた状態でパフォーマンス エキスパート データを収集できます。そのためには、コマンド ライン ツールやスクリプトを使用してアプリケーションを開始および停止します。DevPartner には、この目的のために、**DPAnalysis.exe** というコマンド ライン ユーティリティが用意されています。コマンド ラインからのパフォーマンス エキスパート セッションの実行の詳細については、「データ収集を自動化する」 (235 ページ) を参照してください。たとえば、以下の手順を実行できます。

1. **DPAnalysis.exe** を使用し、パフォーマンス エキスパートを有効にして、アプリケーションを実行します。
2. ロード テスト アプリケーションを実行します。
3. アプリケーションを停止します。
4. パフォーマンス エキスパート セッション データを調査します。

セッション ファイルを確認しても、[最も多く **CPU** を使用する単一メソッド] グラフのどのメソッドも突出した原因になっていないとします。これは複雑なアプリケーションであり、複数のメソッドがパフォーマンス低下の原因である可能性があります。このような場合は、結果サマリの [最も多く **CPU** を使用するパス] グラフから分析を開始します。このグラフにはメソッドのリストが表示されますが、この場合、各メソッドはエントリ ポイントを表しています。エントリ ポイント メソッドとは、他のソース コード メソッドによって呼び出されないソース コード メソッドです。つまり、作成したコードを実行するときのエントリ ポイントとなるメソッドです。エントリ ポイント メソッドは実行パスの開始点であり、メソッドを変更したり、メソッドの呼び出し方を変更したりすることによって変更できます。アプリケーション内で最もコストの高い実行パスに対応するエントリ ポイント メソッドがグラフの一番上に表示されます。メソッドをクリックして、[パス分析] ビューを開きます。

コール グラフ

結果サマリからコール グラフを開くと、コール グラフの一番上には最もコストの高いパスが配置され、パスの分岐点では最もコストの高い下位パスが強調表示されます。データを調査する場合は、最初に、最もコストの高い下位パスを調査します。パスを調査するには、そのノードを右に展開します。

ヒント：メソッドと、そのメソッドから呼び出されたすべての下位メソッドを結ぶライン上の割合を合計すると100%になります。1つのパス内のメソッドのチェーンを接続するライン上の割合は合計しても100%になりません。

同じメソッドで実行されたパスごとの相対的寄与率を判断するには、選択されたメソッドをその下位パスに接続しているライン上の割合値を比較します。各リンク上の値は、上位メソッドの実行時間のうち、そのパスで呼び出された下位メソッドの割合を示しています。したがって、図7-13（232ページ）では、メソッド`Form.Main`は`Form.CtoF`、`Form.ParseOption`、および`Form.FtoC`を呼び出しています。`Form.Main`から`Form.CtoF`にリンクするライン上の値は98.1%であり、残りの1.9%は他のコール パスで配分されます。つまり、`Form.Main`で下位メソッドの実行に費やされたCPU時間の98.1%が、`Form.Main`から`Form.CtoF`へのコール パスによって費やされたことになります。このパスからトラブルシューティングを開始します。

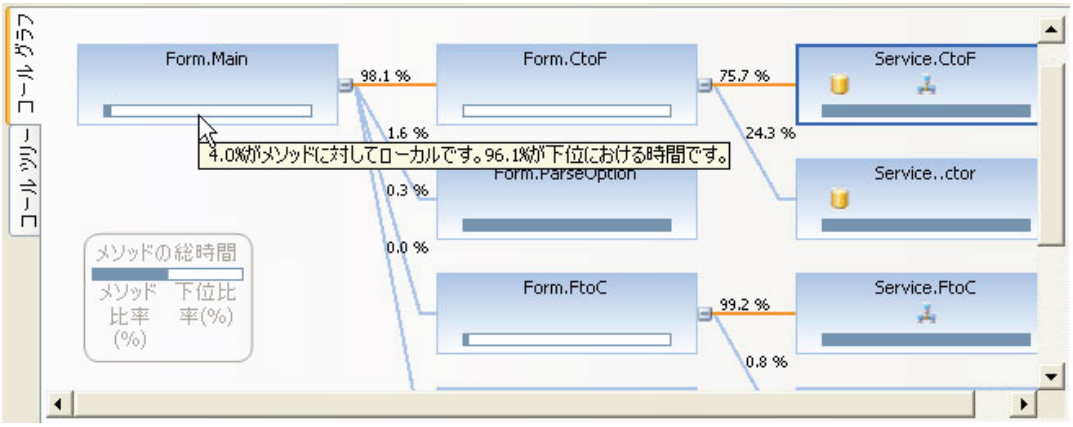


図7-13 下位メソッドの影響の理解

コール パスを調査する場合、各ノードの下にある水平の棒グラフを確認してください。この棒グラフは、メソッド本体とそれが呼び出した下位メソッドに起因するメソッドの実行時間の相対的割合を示しています。このバーの上にマウスを移動して、実際の割合を確認します。このバーを使用して、調整方法を検討します。たとえば、メソッド本体で4%の時間が消費され、下位メソッドに起因する時間が96%の場合は、最もコストの高いコール パスの調査を継続して、パフォーマンスに影響を与えている下位メソッドを特定します。これらのメソッドを修正するか、これらのメソッドの呼び出しを減らすようにコードを変更します。一方、時間の96%がメソッド本体で消費されていた場合は、以下を集中的に実行します。

また、コストの高いノードにディスク動作、ネットワーク動作、または待機時間のアイコンが含まれているかどうかを確認します。これらのアイコンにマウスを移動して、動作の規模を表示します。ノードにこれらのアイコンが1つまたは複数含まれている場合は、[コール ツリー]ビューに切り替え、適切なデータ カラムを追加して、問題の診断に役立つ情報を入手します。

コール ツリー

[コール ツリー] テーブルは、デフォルトで **[** ユーザーの下位メソッドを含む **CPU** 時間 **]** でソートされています。多くの時間がどこで費やされているかを確認するために、他のカラムの値も確認します。それにより、待機時間、ディスク I/O、ネットワーク I/O、または CPU を集中的に使用する処理のいずれが主な要因であるかを判断できます。さらに詳細な情報が必要な場合は、ディスクやネットワークの読み書き数やエラー数などの他のカラムを表示に追加できます。

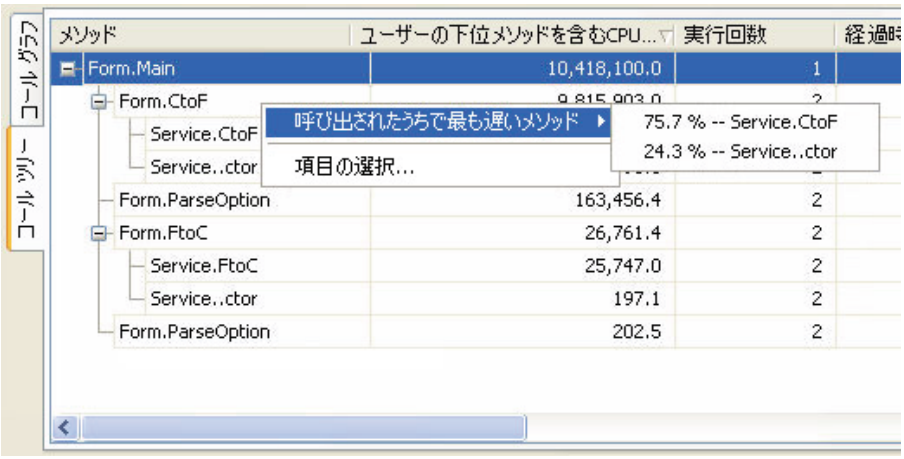


図 7-14 コール ツリー内の選択したメソッドの追加データの表示

たとえば、コール グラフ内のコストの高いメソッドにネットワーク I/O が含まれている場合は、そのメソッドを選択し、コール ツリーに切り替えて、テーブルにネットワーク関連のすべてのデータ カラムを追加します。カラムを追加するには、[コール ツリー] テーブルを右クリックして、コンテキスト メニューから **[** 項目の選択 **...]** を選択します。各カラムにレポートされたデータの詳細な説明については、パフォーマンス エキスパートのオンライン ヘルプを参照してください。

ヒント : 「ユーザー下位メソッド」や「ユーザー メソッド」で使用されている「ユーザー」という用語は、ソース コード メソッドを指しています。

コール グラフまたはコール ツリーのいずれを使用する場合も、セッション ファイル ウィンドウには **[** ソース **]** タブと **[** コール スタック **]** タブが含まれています。これらのタブは、[メソッド] テーブルのタブと同様の機能を持っていますが、計算されるデータにユーザー (または ソース コード) 下位メソッドのデータが含まれている点が異なります。**[** ソース **]** タブは、コール グラフまたはコール ツリーで選択した任意のメソッド内で最もコストの高い行を即座に特定する場合に使用します。**[** コール スタック **]** タブは、メソッドを呼び出した他のパスの相対的な影響を確認し、スタック内で選択したメソッドを呼び出した行を特定する場合に使用します。**[** ソース **]** タブまたは **[** コール スタック **]** タブのどちらかのコードの行をダブルクリックし、Visual Studio でその行に移動して編集します。

パフォーマンスは低いが具体的な問題が見つからない場合

アプリケーションの速度が全体的に遅いが、具体的な問題が見つからないとします。パフォーマンスの調整は反復的なプロセスです。このような場合でも、引き続き前述の方法を使用して、パフォーマンスの向上を試みることができます。

- パフォーマンス エキスパートを有効にしてアプリケーションを実行します。

- **[**最も多く **CPU** を使用するパス**]** を確認して、各クリティカル パスで最もコストの高い分岐を最適化します。
- 同様に、**[**最も多く **CPU** を使用する単一メソッド**]** のリストを確認して、リスト内の上位メソッドを最適化します。
- 再度テストを行って、性能の向上を確認します。

Web アプリケーションのデータを収集する

Web アプリケーションを含む任意のマネージ アプリケーションに関するパフォーマンス エキスパート データを収集できます。パフォーマンス エキスパートで Web アプリケーションを実行する場合は、以下のことに注意してください。

マネージ コードのみ

その他の DevPartner 機能と違って、パフォーマンス エキスパートでは、マネージ アプリケーションに関するデータだけが収集されます。そのため、アプリケーションで Internet Explorer がクライアントとして使用されている場合は、セッション ファイルに Internet Explorer のデータは含まれません。ASP.NET または Web サービス アプリケーションに関するサーバー側データが表示されます。

web.config 要件

パフォーマンス エキスパートで ASP.NET アプリケーションを正しくプロファイルするには、以下の 2 つの条件が満たされる必要があります。

- プロジェクトに **`web.config`** ファイルが含まれている。
- プロジェクトがデバッグ用に構成されている。これを実行するためには、**`web.config`** ファイルに、debug 属性を true に設定した compilation 要素を含める必要があります。以下に例を示します。

```
<compilation debug="true" />
```

複数プロセスのプロファイリング

Visual Studio IDE から、または、DevPartner のコマンド ライン スイッチを指定してコマンド ラインからパフォーマンス エキスパートを起動した場合は、セッションごとに 1 つのプロセスまたはサービスに関するデータが収集されます。複数のプロセスでアプリケーションが実行されている場合、または、ターゲット アプリケーションが実行されているプロセスだけでなく、IIS などのサービスに関するデータも収集する場合は、**`DPAnalysis.exe`**（DevPartner の分析ツールのコマンド ライン実行ファイル バージョン）を使用して、XML 構成ファイルを対象としてセッションを管理します。詳細については、「DPAnalysis.exe で XML 構成ファイルを使用する」 285 ページ）を参照してください。

メモ： **`DPAnalysis.exe`** と XML 構成ファイルを使用すれば、同時に、複数のプロセスまたはサービスから 別々のセッション ファイルに）データを収集することができますが、通常、パフォーマンス エキスパートは、一度に 1 つのプロセスに対して実行するように最適化されています。複数プロセスに対するデータ収集のオーバーヘッドは、アプリケーションが低速化して経過時間が長くなるだけでなく、プロセスの相互作用に影響を及ぼす可能性があります。同時に複数のプロセスに対してパフォーマンス エキスパート データを収集する場合は、ディスク I/O、ネットワーク I/O、または同期待機時間のタイミング値を増やすと、プロファイリングのオーバーヘッドによる性能の

低下につながる可能性があります。1つのプロセスを対象としたセッションを実行して、調査に適したタイミング値を確認してください。

IIS6.0上の単一プロセスのプロファイリング

IIS 6.0上では、1つのワーカー プロセスに関するパフォーマンス エキスパート データが収集されます。IIS上には、アプリケーション プールにつき1つのワーカー プロセスが存在します。そのため、システム上でWebサービスとWebサービス クライアントの両方が1つのアプリケーション プール内で実行されている場合は、パフォーマンス エキスパートを有効にしてサービスを開始し、Visual Studioの別のインスタンスでパフォーマンス エキスパートを有効にしないでクライアントを開始した場合でも、パフォーマンス エキスパートは両方のデータを収集します。クライアントが別のアプリケーション プール内で実行されるようにアプリケーションを変更すれば、パフォーマンス エキスパートは、パフォーマンス エキスパートを使用して起動されたアプリケーション　(この場合はサービス）に関するデータのみを収集します。

DLLHOST下で実行中のコンポーネントに関するリモート セッション ファイル

リモート システム上の**dllhost.exe**と対話するプロセスに対してパフォーマンス エキスパートを実行している場合は、**dllhost.exe**が終了したときに、最終セッション ファイルがリモート システム上に生成されません。

リモート コンピュータのソース コード

DevPartner Studioでは、ソース ファイルは開いたセッション ファイルと同じコンピュータにあるとみなされます。

- ソース コードを表示しようとすると**[**ファイルを開く**]**ダイアログ ボックスが表示される場合は、そのダイアログからリモート コンピュータ上の正しい場所を参照します。
- リモートASP.NETアプリケーションのデータを収集した場合、ソース ファイルを参照するには、ターゲットWebサイトのIIS設定の **[** 仮想ディレクトリ **]** タブで **[** ローカル パス **]** エントリの値を参照する必要がある場合があります。

開いているソリューションに保存されたセッション ファイル

DevPartnerセッション ファイルは、現在のソリューションと共に保存されます。Visual Studioでプロジェクトを開くのではなく、IISから直接Webプロジェクトを開いた場合は、異なるソリューション ファイルが使用される可能性があります。あるソリューションで作成されたDevPartnerセッション ファイルは、別のソリューションでは表示されません。

データ収集を自動化する

パフォーマンス エキスパートでは、**DPAnalysis.exe**という実行ファイルからのコマンド ライン実行がサポートされています。このファイルは、**¥Program Files¥Micro Focus¥DevPartner Studio¥Analysis¥**フォルダにあります。

メモ： 64ビットバージョンのWindowsでは、DevPartner Studioは以下の場所にインストールされます。**¥Program Files (x86)¥Micro Focus¥DevPartner Studio¥Analysis¥**

パフォーマンス エキスパートを有効にしてコマンド プロンプトからアプリケーションを実行したり、データ収集を自動化するためのバッチ ファイルを作成したりできます。パフォーマンス エキスパート セッションは、以下の2通りの方法で起動できます。

- 標準のMS-DOSコマンド ライン構文でターゲットと引数を指定します。

- セッションのターゲットと引数を含む XML 構成ファイルを指定します。

コマンド ライン スイッチを使用する

セクション「アプリケーションにおける拡張性の問題」（231ページ）の例を挙げます。品質保証エンジニアは、アプリケーションに対して自動テスト（テスト スイート）が毎晩実行されるように設定することによって、拡張性（またはアプリケーションのその他の側面）を毎日監視します。テストを自動化するには、以下の処理を行うバッチ ファイルを設定します。

1. パフォーマンス エキスパートを有効にしてアプリケーションを実行する。
2. ロード テスト アプリケーションと実行するその他のテストを開始する。
3. テストが完了したらアプリケーションを停止する。

アプリケーションが終了すると、DevPartner によって自動的にセッション ログ ファイルが生成されます。

セッションを起動するコマンド ライン構文は以下のとおりです。

```
DPAnalysis.exe /Exp /E /O /W /H [/P or /S] target {target arguments}
```

/Exp は、分析タイプを [パフォーマンス エキスパート] に設定します。

/E は、指定されたプロセス／サービスに対するデータ収集を有効にします。

/O は、セッション ファイルの出力フォルダとファイル名のいずれかまたは両方を指定します

/W は、プロセスの作業フォルダを指定します。

/H は、ターゲットを実行するホスト コンピュータを指定します。

/P または /S は、ターゲットがプロセスであるか、またはサービスであるかを指定します。いずれか一方のみを使用します。

スイッチの指定順序には制限事項が 1 つあります。/P または /S スイッチは、最後に指定してください。これらのスイッチのあとに指定されているものは、すべて、プロセスまたはサービスへの引数として解釈されます。

XML 構成ファイルを使用する

XML 構成ファイルを使用する場合のコマンド ラインはより単純です。

```
dpanalyis.exe /C [path]configuration_file.xml
```

構成ファイルには、コマンド ライン スイッチを使用する場合には利用できないオプションも含めて、実行する DevPartner 分析のタイプに必要なパラメータが含まれます。たとえば、パ

フォーマンス エキスパート セッションからアプリケーション コンポーネントを除外する場合は、構成ファイルで ExcludeImages 要素を使用する必要があります。

```
<?xml version="1.0" ?>
<ProductConfiguration xmlns="http://www.microfocus.com/products">
        <RuntimeAnalysis Type="Expert" MaximumSessionDuration="1000"/>
        <Targets RunInParallel="true">
                <Process CollectData="true" Spawn="true" NoWaitForCompletion="false">
                        <AnalysisOptions NO_QUANTUM="1" NM_METHOD_GRANULARITY=""
                        SESSION_DIR="c:\Sessions" SESSION_FILENAME="ClientApp.dppxp" />
                        <Path>ClientApp.exe</Path>
                        <Arguments>/arg1 /agr2 /arg3</Arguments>
                        <WorkingDirectory>c:\temp</WorkingDirectory>
                        <ExcludeImages>
                                <Image>ClassLibrary1.dll</Image>
                                <Image>ClassLibrary2.dll</Image>
                        </ExcludeImages>
                </Process>
                <Service CollectData="false" Start="true" RestartIfRunning="true"
                RestartAtEndOfRun="true">
                        <AnalysisOptions NM_METHOD_GRANULARITY="0" SESSION_DIR=""
                        SESSION_FILENAME="" />
                        <Name>iisadmin</Name>
                        <Host>remotemachine</Host>
                </Service>
        </Targets>
</ProductConfiguration>
```

図 7-15 XML 構成ファイルでのセッション詳細の指定

リモート コンピュータで実行されるプロセスのデータを収集するには、フォルダとファイル名を指定する必要があります。構成ファイルの分析オプションで SESSION_FILENAME 要素と SESSION_DIR 要素を使用します。

構成ファイルを使用したデータ収集の管理の詳細については、「DPAnalysis.exe で XML 構成ファイルを使用する」（185 ページ）を参照してください。

QA エンジニアは、翌朝セッション ログ ファイルを確認します。パフォーマンスの値が低下した場合、QA はセッション ログを適切な開発者に送信します。このようにして、QA は開発サイクル全体を通して、アプリケーションの状態を追跡します。問題が発生した場合、開発チームはセッション ログ ファイルを使用して、問題の性質をすばやく特定できます。さらに、テストを毎晩実行した場合は、問題が前日に変更したコードに起因するものであることがわかるため、問題の修正時にレビューするコード量を大幅に削減できます。

DPAnalysis.exe の使用の詳細については、付録 B「コマンド ラインから分析を起動する」を参照してください。

分散アプリケーションのデータを収集する

DevPartner では、リモート システムにリモート データ収集用のライセンスが適切に設定されている場合には、リモート システムで実行される分散アプリケーション コンポーネントのパフォーマンス エキスパート データを収集できます。リモート セッションを起動するときは、Visual Studio から実行する場合や、従来のコマンド ライン構文を使用してコマンド ラインから **DPAnalysis.exe** を実行する場合には、パフォーマンス エキスパート セッションの 1 回の実行で 1 つのプロセスのみが監視されることに注意してください。XML 構成ファイルを使用

すると、アプリケーションの1回の実行で複数のプロセスやサービスを対象とすることができますが、通常はパフォーマンス エキスパート セッションでは1つのプロセスを対象とすることをお勧めします。アプリケーションが複数のプロセスで実行される場合は、2つめのプロセスを対象にしてアプリケーションを再実行します。アプリケーションをスクリプトやバッチファイルで起動すると、両方のセッションで同じようにアプリケーションが実行されることが保証されます。この概要については、「データ収集を自動化する」（235ページ）を参照してください。

XML構成ファイル オプションを指定して**DPAnalysis.exe**を使用すると、必要に応じて、アプリケーションの1回の実行で2つめのプロセスやサービスのデータを別のセッション ファイルに収集できます。同時に複数のプロセスやサービスのデータを収集できますが、複数のプロセスを対象にしたデータ収集のオーバーヘッドによって、プロセス間の相互作用に影響があったり、アプリケーションの実行速度が遅くなることによって経過時間の値が大きくなったりする可能性があります。同時に複数のプロセスに対してパフォーマンス エキスパート データを収集する場合は、ディスクI/O、ネットワークI/O、または同期待機時間のタイミング値を増やすと、プロファイリングのオーバーヘッドによる性能の低下につながる可能性があります。1つのプロセスを対象としたセッションを実行して、調査に適したタイミング値を確認してください。

DPAnalysis.exeでのリモート データ収集の有効化

DPAnalysis.exeは、リモート プロセスを実行するために使用することはできません。DPAnalysisはリモート コンピュータ上のプロセスのデータ収集を有効にすることのみを目的としています。以下に例を示します。

```
DPAnalysis.exe /host remotemachine /p c:¥MyDir¥target.exe
```

このコマンド ラインでは、**DPAnalysis.exe**は**target.exe**のプロファイルを有効にするだけで、リモート コンピュータでこのアプリケーションを起動するわけではありません。**target.exe**がリモート コンピュータ上で（何らかの方法によって）起動されたときに、プロファイルが開始されます。

ただし、リモート サービスはローカル マシンから開始することができます。例：

```
DPAnalysis.exe /host remotemachine /s servicename
```

このコマンドはプロファイルを有効にし、**remotemachine**上の**servicename**を起動します。

必要に応じて、XML構成ファイルを使用し、上記のコマンド ラインにパラメータを指定することができます。**DPAnalysis.exe**の詳細については、付録B「コマンド ラインから分析を起動する」を参照してください。

リモート コンピュータ上のセッション ファイルを保存する

リモート プロファイル シナリオでは、いずれの分析タイプ（カバレッジ、メモリ、パフォーマンス、パフォーマンス エキスパート）でも、セッション ファイルはリモート コンピュータに保存されます。保存先のフォルダとセッション ファイル名をコマンド ラインに指定するか、リモート プロセスまたはリモート サービスの XML 構成ファイルに追加する必要があります。指定するフォルダは、リモート コンピュータ上にすでに存在していることが必要となります。フォルダまたはファイル名を指定しなかった場合は、リモート コンピュータに [名前を付けて保存] が表示されます。

セッション ファイルを表示する

DevPartner Studio がインストールされているコンピュータ（プロファイルが開始され、クライアント ファイルが保存されているコンピュータなど）にセッション ファイルをコピーします。

コマンド ラインまたは XML 構成ファイルに、DevPartner Studio がインストールされている別のコンピュータ（プロファイルが開始されているコンピュータなど）にセッション ファイルを保存するための、リモート コンピュータ上でマップされたドライブを指定します。

ターミナル サービスまたはリモート デスクトップを使用してデータを収集する

DevPartner Studio では Windows のターミナル サービスがサポートされています。DevPartner Studio をターミナル サービスを使用する方法の詳細については、「ターミナル サービスとリモート デスクトップを使用する」（221 ページ）を参照してください。

リモート プロファイルと **Windows XP Service Pack 2**（**SP2**）以降

Windows XP SP 2 では、リモート アプリケーションのセキュリティ レベルが引き上げられています。そのため、Visual Studio からサーバー側のアプリケーション コンポーネントをプロファイルする場合、セキュリティ設定が原因でデータを収集できないことがあります。リモート コンピュータ上のアプリケーション コンポーネントからデータを収集するためには、セッションに参加するすべての Windows XP SP2 以降のオペレーティング システム（プロファイルが開始されているリモート コンピュータとクライアント コンピュータの両方）のセキュリティ設定を変更する必要があります

ここでは、これらのセキュリティ設定をリモート プロファイルが可能な設定に変更する 3 通りの方法について説明します。

Windows ファイアウォールの例外リストに **DevPartner** コントロール サービスを追加する

Windows ファイアウォール サービスを有効にしている場合は、ファイアウォールの例外リストに DevPartner コントロール サービスを追加します。以下の手順に従ってください。

1. **[** スタート **]** メニューから **[** コントロール パネル **]** を選択します。
2. コントロール パネルの **[Windows** ファイアウォール **]** を選択し、**[** 例外 **]** タブをクリックします。
3. **[** 例外 **]** タブの **[** プログラムの追加 **]** をクリックします。
4. **[** プログラムの追加 **]** ダイアログ ボックスの **[** 参照 **]** をクリックし、**NCS.exe** を探します。この実行可能ファイルは、デフォルトでは以下の場所にあります。

```
C:¥Program Files¥Micro Focus¥DevPartner Studio¥Analysis¥NCS.exe
```

メモ： 64ビット バージョンのWindowsでは、この実行ファイルは以下の場所にインストールされます。**¥Program Files (x86)¥Micro Focus¥DevPartner Studio¥Analysis¥NCS.exe**

5 **[**参照**]**ダイアログ ボックスの**[**開く**]**をクリックして**NCS.exe**を選択し、**[OK]**をクリックして**[**プログラムの追加**]**ダイアログ ボックスを閉じます。

6 **[Windows** ファイアウォール**]**コントロール パネルの**[**全般**]**タブをクリックし、**[**例外を許可しない**]**チェック ボックスをオフにします。

リモート（サーバー）とローカル（クライアント）コンピュータ両方でのセキュリティ設定の変更

以下の手順に従ってセキュリティ設定を変更します。

1 コントロール パネルで、**[**管理ツール**] > [**ローカル セキュリティ ポリシー**] > [**ローカル ポリシー **] > [**セキュリティ オプション**]**を選択します。

2 **[DCOM:**セキュリティ記述子定義言語 **(SDDL)** でのコンピュータ アクセス制限**]**の[プロパティ]ページを開きます。

3 **[**セキュリティの編集**]**を選択します。

4 匿名ログオンユーザーを追加します　追加していない場合）。

5 **ANONYMOUS LOGON**ユーザーにローカル アクセスとリモート アクセスの両方を割り当てます。

Visual Studioを実行した状態でセキュリティ設定を変更した場合は、変更後の設定を有効にするためにVisual Studioを再起動する必要があります。

クライアント コンピュータで **COM** セキュリティを緩和する

COMセキュリティを緩和するには、プロファイリングを開始したクライアント コンピュータで以下の手順を実行します。

1 **[**スタート**]**メニューから**[**コントロール パネル**]**を選択します。

2 **[**コントロール パネル**]**で**[**管理ツール**]**を選択し、**[**管理ツール**]**ウィンドウで**[**コンポーネント サービス**]**を開きます。

3 **[**コンポーネント サービス**]**ウィンドウで、**[**マイ コンピュータ**]**へ移動し、**[**マイ コンピュータ**]**を右クリックして**[**プロパティ **]**を選択します。

4 **[**マイ コンピュータのプロパティ **]**の**[COM**セキュリティ **]**タブをクリックします。

5 **[COM**セキュリティ**]**タブの**[**起動とアクティブ化のアクセス許可**]**にある**[**制限の編集**]**をクリックし、以下の変更を行います。

6 **[**追加**]**をクリックし、NETWORKと入力します。

7 **[**ローカルからの起動**]**、**[**リモートからの起動**]**、**[**ローカルからのアクティブ化**]**、**[**リモートからのアクティブ化**]**のいずれにおいても**[**許可**]**チェック ボックスがオンになっていることを確認してください。

8 **[COM**セキュリティ **]**タブの**[**起動とアクティブ化のアクセス許可 **]**にある**[**既定値の編集**]**をクリックし、以下の変更を行います。

9 **[**追加**]**をクリックし、NETWORKと入力します。

10 **[** ローカルからの起動 **]**、**[** リモートからの起動 **]**、**[** ローカルからのアクティブ化 **]**、**[** リモートからのアクティブ化 **]** のいずれにおいても **[** 許可 **]** チェック ボックスがオンになっていることを確認してください。

ファイアウォールとリモート データ収集

DevPartner は、Visual Studio で実行されている場合でも **DPAnalysis.exe** から実行されている場合でも、リモート コンピュータからセッション データを収集するために以前にインストールされたサービスに接続します。このサービスは、インタープロセス通信トラフィックをリスンします（インターネット アドレス `0.0.0.0`、ポート `18441`）。一部のファイアウォールでは、このサービス接続によりアラームがトリガーされます。このアドレスを信頼できるアドレスとしてファイアウォールを設定すると、アラームがトリガーされなくなります。ファイアウォールのセキュリティ レベルを最高に設定している場合は、DevPartner がリモート データ収集を実行できないことがあります。この場合はファイアウォールを再構成し、アドレス `0.0.0.0`、ポート `18441` でのデータ交換を有効にしてください。

DevPartner データを XML 形式にエクスポートする

パフォーマンス エキスパート データを XML 形式にエクスポートできます。データを XML 形式でエクスポートすることによって、独自のソフトウェアやサードパーティ製ソフトウェアを使用したデータの分析、他のツールによって生成されたデータとの統合、データ ウェアハウスへのデータのアーカイブがより容易に実行できるようになります。

パフォーマンス エキスパート セッション ファイル（拡張子 **.dppxp**）を XML 形式にエクスポートできます。保存されたパフォーマンス エキスパート セッション ファイルを開くと、**[** ファイル **]** メニューの **[**DevPartner データのエクスポート **]** コマンドが使用可能になります。

「分析データを XML にエクスポートする」（309 ページ）に説明されているように、コマンドラインから XML データをエクスポートすることもできます。

DevPartner Studio インストール フォルダにあるファイル **DevPartnerPerformanceExpert***xx*`.xsd` に、セッション ファイルのエクスポート時にパフォーマンス エキスパートによって使用される XML スキーマが記述されています。

パフォーマンス エキスパートとパフォーマンス分析を併用する

パフォーマンスの調整は反復的なプロセスです。パフォーマンス エキスパートを DevPartner Studio のパフォーマンス分析と組み合わせて使用します。最初に、パフォーマンス分析を有効にしてアプリケーションを実行し、セッション ファイルを保存して、パフォーマンスの基本的な傾向を把握します。次に、パフォーマンス エキスパートを使用して、困難な問題、特に、ディスク I/O、ネットワーク I/O、同期に関する問題のトラブルシューティングを行います。問題が解決したら、パフォーマンス分析セッションでアプリケーションを実行し、パフォーマンス分析のセッション比較機能を使用して性能の向上を確認します。例：

1 パフォーマンス分析を使用してアプリケーションを実行します。

2 性能を低下させているメソッドを発見します。

3 問題のメソッドを解決する方法がすぐに分からない場合は、パフォーマンス エキスパートを使用して同じセッションを実行します。

4 問題のメソッドが、**[** 最も多く **CPU** を使用するパス **]** グラフまたは **[** 最も多く **CPU** を使用する単一メソッド **]** グラフに表示されていないかどうかチェックします。

5 **[**最も多く **CPU** を使用するパス**]**グラフ内のそのメソッドをクリックして、コール グラフを開きます。コール グラフでは、コンテキスト内にそのメソッドが表示され、そのメソッドとそのメソッドの下位メソッドのどちらがパフォーマンス問題に関与しているかが示されます。

6 問題のメソッドが、ディスク アイコン、ネットワーク アイコン、または待機時間アイコンでマークされているかどうか確認します。

たとえば、メソッドがネットワーク動作を示している場合は、**[**コール ツリー **]**タブに切り替えて、コンテキスト メニューを使用してネットワークに関連するデータ項目を表示要素に追加します。追加したデータによって、問題が、読み込み動作、書き込み動作、および読み書きエラーのどれに起因しているかを容易に把握することができます。**[**最も多く **CPU** を使用する単一メソッド**]**グラフからさらにデータを分析する場合は、データ項目を [メソッド] テーブルに追加することができます。

7 **[**コール スタック**]**タブを使用して、問題のメソッドが呼び出された回数と、最もコストが高いコール スタックを表示します。

8 **[**ソース**]**タブを使用して、問題のコード行を特定し、Visual Studio で編集するソース ファイルに移動します。

問題が解決したら、再度、パフォーマンス分析セッションでアプリケーションを実行します。以前のパフォーマンス分析のセッション ファイルをベースラインとして使用して、パフォーマンス分析のセッション比較機能でセッションを比較し、性能の向上を確認します。

パフォーマンス エキスパートとパフォーマンス分析は相補的な機能ですが、タイミング データの計算方法には差異があります。同じアプリケーション上で、システム イメージを含むパフォーマンス分析セッションと、パフォーマンス エキスパート セッションを実行したときに、パフォーマンス分析の **[** ソース メソッドの上位 **20]** とパフォーマンス エキスパートの **[**最も多く **CPU** を使用する単一メソッド **]** に異なるメソッドが含まれていたり、メソッドの順序が違う場合があります。

パフォーマンス分析セッションでは、メソッド内で消費された時間の割合 **[**（メソッドでの比率 **[%]]** カラム）が、ユーザーまたはシステムの下位メソッドを含まずに計算されます。パフォーマンス エキスパート セッションでは、**[**最も多く **CPU** を使用する単一メソッド **]** テーブルと **[** メソッド **]** テーブルに表示されるメソッド内で消費された時間の割合に、システムの下位メソッドで消費された時間が含まれます。

パフォーマンス プロファイリングを実行すると、マネージ アプリケーションでは .NET Framework 内のメソッドの実行に多くの時間が費やされていることがわかります。パフォーマンス エキスパートの結果にシステムの下位メソッドを含むことによって、メソッド自体の実行に時間がかかるメソッドよりも、システム コードとの相互作用に問題のあるソース コード内のメソッドに焦点が当てられます。システム コードが実行を開始すると、その時間消費を制御することはできませんが、アプリケーション コードでシステム コードを呼び出す方法やタイミングを変更することはできます。パフォーマンス エキスパートによって、このような問題の特定が容易になります。

パフォーマンス分析のセッション ファイルとパフォーマンス エキスパートのセッション ファイルを直接比較することはできません。パフォーマンス分析のセッション ファイル同士を比較できるだけです。

開発サイクルにおけるパフォーマンス エキスパート

ソフトウェア開発サイクル全体にわたって、パフォーマンス エキスパートを使用します。ソフトウェア プロジェクト ライフ サイクル内の複数の時点でパフォーマンス エキスパートを使用することは、エンジニアリング チームの多くのメンバにとって有用です。

ソフトウェア設計者

ソフトウェア設計者は、多くの場合、応答時間や拡張性などの特定の要件を満たすプロトタイプを開発する必要があります。設計者は、最終的な設計を作成する前に、プロトタイプがパフォーマンス要件を満たさない原因となっている操作と、可能な場合にはメソッドを特定する必要があります。修正した場合に大幅にパフォーマンスが向上するようなメソッドをいくつか特定できることが理想的です。

ソフトウェア設計者は、設計フェーズやプロトタイプ フェーズでパフォーマンス エキスパートを使用して、コードの速度や効率性を向上できます。設計作業を進める間に定期的にテストを行うことで、プロトタイプのコードが最低限のパフォーマンス要件を満たしていることを確認できます。また、プロトタイプでいくつかの重要なパフォーマンス要件のテストが行われていることがわかっているため、開発チームはプロトタイプを受け取ったあと、安心してプロトタイプの一部を再利用できます。

ソフトウェア開発者

ソフトウェア開発者は、開発中に頻繁にパフォーマンス エキスパートを使用する必要があります。コードをチェックインする前に、単体テスト以外にパフォーマンス エキスパートを実行してください。単体テストを行うと、他のコンポーネントをブレークすることなく、コンポーネントが想定されたとおりに動作することを確認できます。それと同様に、パフォーマンス エキスパートを使用すると、コンポーネントがアプリケーションに完全に統合されて修正が困難になる前に、パフォーマンス上の潜在的な問題について早期に警告を受け取ることができます。

ソフトウェア開発チームは、設計者のプロトタイプと仕様に基づいてアプリケーションを構築します。アプリケーション（またはアプリケーション コンポーネント）がテストおよび実行できる状態になると、開発者は、コーディングとデバッグを行いながら CPU の使用、ファイル I/O、ネットワーク I/O に関する潜在的な問題を特定するために、パフォーマンス エキスパートを自動テスト ルーチンに統合できます。開発者は毎朝パフォーマンス エキスパート セッション ログを参照して、前日に行ったコーディングによってパフォーマンスの問題が新たに発生していないかを確認し、問題があれば即座に解決できます。コーディングが完了したら、開発チームは最終的なパフォーマンス エキスパート セッション ログを提出して、パフォーマンスに関する目標が達成されたことを文書化できます。

品質保証エンジニア

品質保証チームは、パフォーマンス エキスパートを使用して、継続的にアプリケーションのパフォーマンスを監視できます。QA は簡単にパフォーマンス エキスパートを自動テスト スイートに統合して、重要な領域におけるアプリケーションのパフォーマンスを毎日確認できます。問題が発生した場合、QA チームはセッション ログを開発チームに送信するか、または障害管理システムへのバグ レポートにログを添付できます。

指定されたエンジニアは、毎日セッション ログ ファイル内の重要なメトリクスを参照できます。セッション ログに問題が見つかった場合、QA エンジニアはそのログ ファイルを担当の開発者に送信して、問題を即座に解決できます。

このように、設計フェーズから最終的な品質保証テストに至るまで、パフォーマンス エキスパートを実行することによって、ソフトウェア開発チームのすべてのメンバに利点がもたらされます。パフォーマンス エキスパートは、製品管理にも役立ちます。重要な各マイルストーンで、パフォーマンス エキスパート セッション ログ、および修正前後のパフォーマンス分析セッション ファイルを使用して、製品が目標パフォーマンスを満たしていることを文書化できます。

Visual Studio Team System にデータを送信する

Visual Studio 2005 および 2008 については、選択した項目に関するデータをバグタイプの作業項目として Visual Studio Team System に送信します。Visual Studio 2010 では、選択した項目に関するデータを問題、バグ、または不具合タイプの作業項目として Team Foundation Server に送信します。カバレッジ分析セッション ファイルでは、[メソッド リスト] タブで選択したメソッドの作業項目の送信にアクセスします。作業項目を送信する場合、**[** メソッド リスト **]** タブで表示されるカラムからのデータが **[** 作業項目 **]** 形式で入力されます。**[** 作業項目 **]** として送信するメソッド データを変更するには、**[** メソッド リスト **]** に表示されるカラムを変更します。DevPartner Studio では、Team Explorer クライアントがインストールされており、Team Foundation Server に接続可能な場合に、Microsoft Visual Studio Team System がサポートされます。

パフォーマンス エキスパートのセッション ファイルから、メソッドレベルのデータを、Visual Studio Team System のバグタイプの作業項目として送信できます。**[** 作業項目の提出 **]** コマンドは、以下のパフォーマンス エキスパート ビューで選択したメソッドのコンテキスト メニューで使用できます。

- [メソッド] 詳細ビューの **[** メソッド **]** テーブル
- [パス分析] ビューの [コール ツリー]

作業項目を送信する場合、**[** メソッド **]** テーブルまたは **[** コール ツリー **]** ビューで表示されるカラムからのデータが **[** 作業項目 **]** 形式で入力されます。**[** 作業項目 **]** として送信するメソッド データを変更するには、メソッド ビューに表示されるカラムを変更します。

DevPartner Studio と Visual Studio Team System の統合の詳細については、「Visual Studio Team System のサポート」（210 ページ）を参照してください。

第8章

System Comparison

この章には、2つのセクションが含まれています。1つめのセクションでは、はじめてのユーザーを対象として、System Comparisonを使用する簡単な手順について説明します。2つめのセクションでは、DevPartnerのSystem Comparison機能を詳細に把握するための参照情報を示します。

システムの比較に関するタスクベースの追加情報については、System Comparisonのオンライン ヘルプを参照してください。

System Comparison とは

System Comparisonでは、2つのコンピュータ システムを比較したり、コンピュータの現在の状態と過去の状態を比較したりすることで、アプリケーションが以下の動作をする原因を突き止めることができます。

- 特定のコンピュータでは動作するのに、別のコンピュータでは動作しない。
- コンピュータによって動作が異なる。
- 以前動作したコンピュータで動作しなくなった。

システムを比較するために、System Comparisonでは、コンピュータ システムに関する情報（インストールされている製品、システム ファイル、ドライバ、その他の数多くのシステム特性など）を含むスナップショット ファイルと呼ばれるXML ファイルが作成されます。スナップショット ファイルが比較されて、それらの差異がレポートされます。

他のDevPartnerコンポーネントとは異なり、System ComparisonはVisual Studio環境に統合されていません。System Comparisonはスタンドアロン ユーティリティとして動作するため、ターゲット システムへの影響が最小限に抑えられます。

System Comparisonは以下の内容で構成されます。

- サービス。システムのスナップショットを夜間に取得します。
- ユーザー インターフェイス。手動でスナップショットを取り、比較して相違点を見つけることができます。
- コマンド ライン インターフェイス
- ソフトウェア開発キット (SDK) ソフトウェア開発者は、SDK を使用して比較についての追加情報を収集して、配置されるアプリケーションにスナップショット機能を埋め込むことができます。

図 8-1 System Comparisonのユーザー インターフェイス

System Comparison を今すぐ使用する

以下に示す準備、設定、実行の手順では、System Comparison を使用する方法について説明します。

すぐに使い始めるには、影付きのボックス内に示された手順に従います。詳細については、ボックスの下の追加説明を参照してください。

System Comparisonでシステムを分析する場合、昇格されたシステム権限は必要ありません。DevPartnerでのシステムの分析には、システム上でファイルを作成し、アプリケーションを操作するシステム権限で十分です。

以下の手順では、コンピュータに小さな変更を行い、コンピュータの現在の状態を過去の状態と比較します。

準備：比較対象を検討する

システムの比較を実行する前に、比較の目標を理解しておきます。

以降の手順では、以下の事項を前提としています。

- System Comparison がインストールされています。
- System Comparison サービスが実行中で、すでにスナップショットが取られています。

  System Comparison をインストールすると、サービスが自動的に開始し、開始から数分以内に最初のスナップショットが取られます。このサービスは、システムのサービス リストに DevPartner `Differ` という名前でリストされています。
- 1 台のコンピュータの異なる状態を比較できます。

比較対象を正確に特定することで、Comparison を正しく設定することができます。たとえば、以下のような目標を設定できます。目標によっては、追加の設定手順が必要な場合があります。

- 製品のインストールまたは削除がコンピュータのサービス、設定、レジストリ キー、またはファイルに及ぼす影響を調べる（レジストリ キーまたはファイルを調べるには、XML ファイルを変更する追加設定が必要です）。
- 製品が以前動作したシステム上で動作しなくなった原因と思われるシステムの変更を判断する。
- 製品への変更による影響の範囲（たとえば、自動テストへの影響）を判断する。
- 以前の開発システムで利用可能であったすべてのツールが新しい開発システムにあることをチェックする。
- 特定のシステム上で製品が動作しない、または製品の動作が異なる原因を判断する。
- エンド ユーザー側への配置後に製品のトラブルシューティングを実行する。

設定：**System Comparison** を準備する

比較の目標を決定したあとに、いくつかの設定タスクの実行が必要になる場合があります。

この手順では、System Comparison のデフォルト オプションを使用できます。追加の設定は必要ありません。

状況例のなかには、以下のような設定タスクが必要な場合があります。

- レジストリ キーまたは特定のファイルを比較する場合は、設定タスクで **`RegistrySections.xml`** ファイルまたは **`FileSections.xml`** ファイルを変更する必要があります（255 ページと 256 ページを参照）。
- デフォルトで収集されないデータを比較する場合は、設定タスクでカスタム プラグインを記述する必要があります（262 ページを参照）。デフォルトで収集されるデータのカテゴリについては、表 8-1（252 ページ）を参照してください。
- 2つのシステムを比較する場合は、設定タスクで2台めのコンピュータにSystem Comparisonをインストールし、スナップショットを取り、そのスナップショット ファイルを比較に使用できるようにする必要があります（258 ページを参照）。

実行：変更を加え、スナップショットを作成する

これで、システムの比較を開始する準備ができました。以下の手順では、コンピュータに変更を行い、コンピュータの現在の状態を過去の状態と比較します。

システムの相違点が報告される様子を実演するには、スナップショットを作成する前に、コンピュータ システムにいくつかの変更を行う必要があります。

1 **[**コントロール パネル**]** > **[**管理ツール**]** > **[**サービス**]**ウィンドウにナビゲートして、作業環境に影響を与えないいくつかのサービスを停止または開始します。たとえば、[Automatic Updates]サービスを停止することができます　あとで再起動できるように、変更したサービスのメモを取ります）。

2 [スタート]メニューから、**[**すべてのプログラム**]** > **[Micro Focus]** > **[DevPartnerSystem Comparison]**を選択します。

3 [System Comparison]ウィンドウで、**[**このコンピュータの現在の状態を以前の状態と比較**]**をクリックします。

スナップショット ファイルのリストが表示されます。System Comparison サービス（150ページを参照）では、コンピュータの状態のスナップショットが毎日自動的に取られ、そのファイルの日付と時間がリストされています。

メモ： リストには少なくともすでにファイルが1つあるはずです。ファイルが1つもリストされていない場合は、System Comparisonサービスが実行中であることを確認してください。このサービスはサービス リストでDevPartner `Differ`という名前です。

4 リストから、比較の基本として使用するスナップショットの日付と時間を選択し、**[**比較**]**をクリックします。

System Comparisonにより、結果ウィンドウが表示されます。結果ウィンドウの内容については、「結果を分析する」を参照してください。

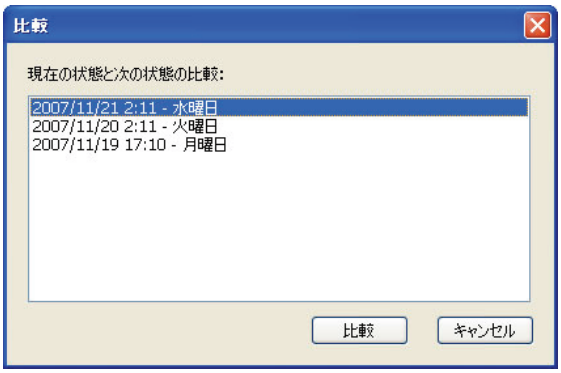


図8-2 スナップショット ファイルのリスト

結果を分析する

System Comparisonで2つのスナップショットを比較すると、その相違点とすべての項目が結果ウィンドウ（図8-3）に表示されます（ここで表示される準備、設定、実行手順の結果ウィンドウの情報は、図に表示されている結果よりもかなり少ない場合があります）。

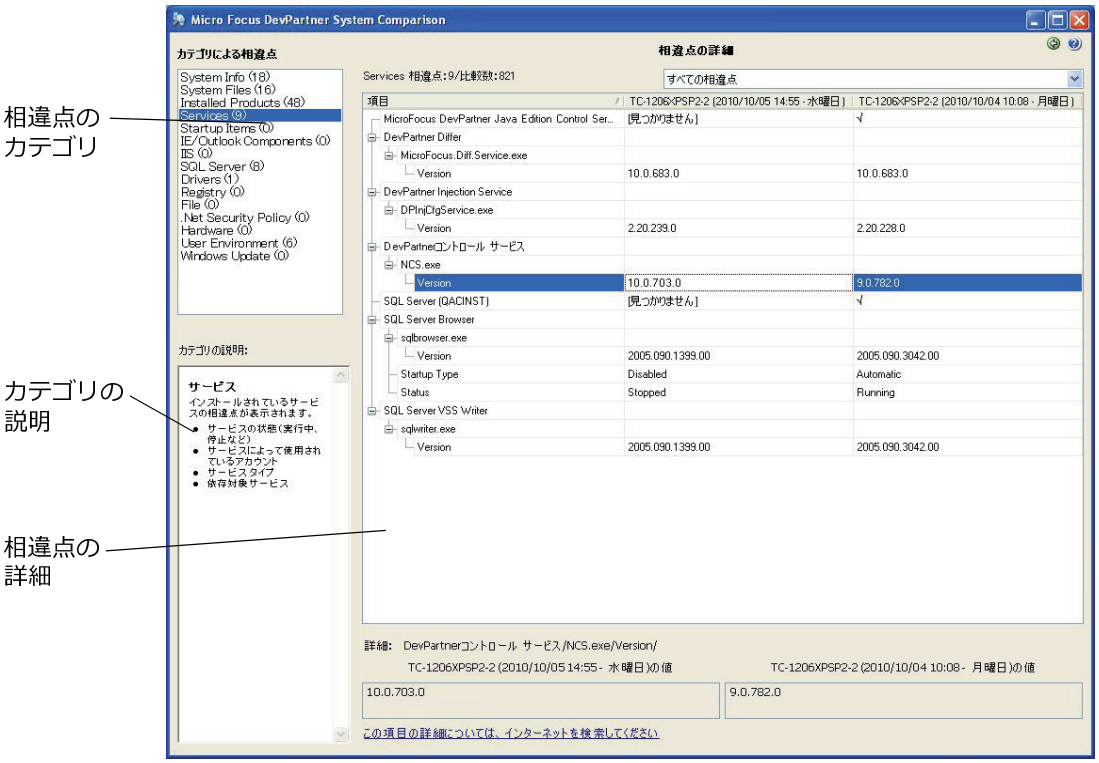


図8-3 結果ウィンドウ

左上のペインには、比較されたカテゴリと、各カテゴリで見つかった相違点の数がリストされます。ウィンドウを最初に開いたときは、相違点の数がゼロ (0) ではない1つめのカテゴリが選択されています。

左下のペインには、選択したカテゴリについての説明が表示されます。

右側のペインには、選択したカテゴリで見つかった相違点の詳細が表示されます。

1. カテゴリをいくつかクリックして、その説明を表示してください。
2. [サービス]カテゴリをクリックすると、[相違点の詳細]ペインに相違点が表示されます。

   [相違点の詳細]ペインでは、最初のカラムに項目の名前が表示されます。2番めと3番めのカラムには、スナップショットから取得した情報がリストされます。ヘッダー行には、コンピュータ名と、比較が実行された完全なタイムスタンプ情報が表示されます。

   スナップショットにない項目は「見つからない項目」としてリストされます。コンピュータ上の項目は、チェック マークまたは「インストールされた日時」で提示されます。
3. 詳細ペインの下部にある2つのカラムには、選択した項目の1つめと2つめのスナップショットから取得した実際のデータがリストされます。

4 画面の下部近くには、[この項目の詳細については、インターネットを検索してください] というリンクがあります。このリンクをクリックすると、現在選択されている相違点（たとえば、「Windows システム環境変数」）に関連する項目のインターネット検索が開始されます。

5 [相違点の詳細] ペインの右上で、[表示する:] のリストをクリックします。このオプションを使用して、表示する相違点をフィルタできます。

6 相違点の確認が終わったあとは、ウィンドウの右上隅にある [戻る]  ボタンをクリックして、DevPartner System Comparison のメイン ウィンドウに戻ります。

結果ウィンドウには、[表示する:] リストに表示したオプションに応じて、相違点が表示され、両方のスナップショットで同じであるすべての項目がリストされます。

System Comparison では、相違点を評価するときにバージョン番号が考慮されることに注意してください。そのため、バージョン番号が異なるコンポーネントは、別のコンポーネントとみなされます。あるコンポーネントが 2 つのスナップショットに存在している場合でも、そのコンポーネントのバージョン番号が異なるときは、見つからないコンポーネントしてリストされます。

現在の状態を過去の別の状態と比較するには、結果ウィンドウの [現在の状態の、次の状態との相違点詳細：] リストから別のスナップショットを選択します。

System Comparison の試用を終了したあとは、はじめに停止したサービスを再起動することを忘れないでください。

これで、この章の準備、設定、実行のセクションは終了です。これで、システムの比較を実行するメカニズムの基礎が理解できました。追加情報については、引き続きこの章の残りの部分を参照してください。タスクベースの情報については、**System Comparison** のオンラインヘルプを参照してください。

System Comparison サービス

DevPartner `Differ` と呼ばれる System Comparison サービスは、対象のコンピュータが起動している場合に、毎日午前 2:10 に自動的にコンピュータの状態のスナップショットを取ります。コンピュータの電源がオフの場合は、次回に起動してから 5 分後にスナップショットを取ります。System Comparison をインストールすると、System Comparison サービスが開始してから数分後にスナップショットが取られます。

夜間にスナップショットを取るサービスは、最大で 21 夜分のスナップショットを保存でき、それ以降は古いスナップショットを削除して新しく取るようになります。保存されるスナップショットの数は、System Comparison ユーティリティの設定ファイルで値を変えることによって変更できます（「保存されるスナップショット数を変更する」 251 ページ）を参照）。スナップショット ファイルのサイズは収集するデータの量によって異なります。通常は、1 メガバイト以下です。

System Comparisonサービスの実行優先度は最低に設定されていますが、実行中は数分間にわたってシステム リソースを消費します。必要に応じて、System Comparisonサービスのスタートアップの種類を手動に設定することもできます。ただし、この場合にはスナップショットが自動的に作成されなくなります。

自動スナップショット設定を変更する

System Comparisonサービスによって取得される自動スナップショットのタイミングと保存されるスナップショット数は、System Comparisonユーティリティの設定ファイルの値で指定します。設定ファイル（**MicroFocus.Diff.Settings.xml**）は**Program Files¥Micro Focus¥DevPartner Studio¥System Comparison¥bin**フォルダにあります。

64ビット バージョンのWindowsでは、設定ファイルは以下の場所にインストールされます。**¥Program Files (x86)¥Micro Focus¥**DevPartner Studio**¥System Comparison¥bin**

保存されるスナップショット数を変更する

デフォルトでは、System Comparisonで21個まで自動でスナップショット ファイルが保存されます。それを過ぎると、最も古いファイルから削除されます。保存されるスナップショット ファイル数を変更するには、設定ファイルの`SnapshotsToKeep`キーを変更します。たとえば、以下のようにキーを変更すると、保存されるスナップショット数が30個までに変更されます。

```
<add key="SnapshotsToKeep" value="30" />
```

スナップショットを取る時刻を変更する

System Comparisonサービスでは、毎日午前2:10にコンピュータのスナップショットが自動で取得されます（コンピュータの電源がオフの場合は、次に起動して5分後にスナップショットが取られます）。このデフォルト時刻を変更するには、`SnapshotHour0To23`キーと`SnapshotMinute0To59`キーを使用して設定ファイルに変更する時刻を指定します。たとえば、以下のようにキーを変更すると、自動スナップショットの時刻が午前3:42になります。

```
<add key="SnapshotHour0To23" value="3" />

<add key="SnapshotMinute0To59" value="42" />
```

「時」に設定できる値は0～23で、「分」に設定できる値は0～59です。

新しい設定を有効にするには、System Comparisonサービスを再起動する必要があります。当日の自動スナップショットがすでに取られている場合、新しい設定は翌日から有効になります。System Comparisonで毎日取得される自動スナップショットは1つだけです。

相違点のカテゴリ

System Comparisonユーティリティでは、スナップショットを取るときに、以下の表にリストされている項目の存在、バージョン、および状態が記録されます。

System Comparisonプラグインを記述することによって別のカテゴリを追加して、データ取得をカスタマイズすることができます（「プラグインを記述する」 262ページ）を参照）。

表8-1. 相違点のカテゴリ

カテゴリ	検出される相違点
システム情報	• オペレーティング システム • .NET Framework • グローバル アセンブリのキャッシュ • Java Runtime • システム環境変数 • ファイル システムの大文字と小文字の区別
システム ファイル	• オペレーティング システム ファイル（**¥Windows¥System32**） • Windows ファイル保護のキャッシュ（**¥Windows¥System32¥dllcache**）– このフォルダには、オペレーティング システム ファイルが破損した場合に Windows を保守するためのオペレーティング システム ファイルが含まれます。ファイルが破損するか見つからない場合、このフォルダにあるファイルで自動的に置換されます（ユーザーの介入はありません）。 • サイドバイサイド アセンブリ（**¥Windows¥WinSxS**）
インストールされている製品	検出された製品。バージョン番号が検出された場合は、その番号も表示されます。 この情報は、レジストリの Add/Remove Programs セクションから抽出されます。
サービス	インストールされているサービスの以下についての相違点 • サービスの状態（実行中、停止中など） • サービスが使用しているアカウント • サービスの種類 • サービスの依存関係
スタートアップ アイテム	スタートアップの相違点。この情報は、以下から抽出されます。 • Windows フォルダにある **Win.ini** ファイル • レジストリ キー： **HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Run** • 可能な場合には、プログラム ファイルから取得するバージョン情報

表8-1. 相違点のカテゴリ

カテゴリ	検出される相違点
IE/Outlook コンポーネント	Internet ExplorerとOutlookについての相違点 • アクティブ セットアップに、更新された、またはシステムに存在しないOutlook ／Internet Explorerのコンポーネントが表示されます。この情報は、レジストリ キー **HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Active Setup¥Installed Components**から抽出されます。 • Browser Helper Object。この情報は、レジストリ キー **HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Explorer¥Browser Helper Objects**から抽出されます。 • MIME マッピング （MIMEタイプとMIMEを扱うアプリケーション間のマッピング）。この情報は、レジストリ キー **HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Active Setup¥MimeFeature objects**から抽出されます。 • Internet Explorer拡張機能。この情報は、レジストリ キー **HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Internet Explorer¥Extensions**から抽出されます。 • インターネット設定。この情報は、レジストリ キー **HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥InternetSettings**から抽出されます。
IIS	IISメタベースから入手可能なMicrosoft IISインストールでの相違点には、インストールされているすべてのWebアプリケーションとそれらの設定での以下の相違点が含まれます。 • Webサーバーの相違点 • SMTPサーバーの相違点 • FTPサーバーの相違点
SQL Server	Microsoft SQLインストールでの相違点 • レジストリから抽出されたMicrosoft SQL設定 • Microsoft SQLサービスおよび関連サービスに関するデータ • インストールされているすべてのインスタンスについての、マスタ データベースにある`syscurconfigs`テーブルと`sysconfigures`テーブルの設定。System Comparisonユーティリティは、統合セキュリティを使用してSQL Serverへの接続を試みます。SQL Serverを実行していない場合は、マスタ データベースでの相違点は収集されません。 メモ：マスタ データベースでの相違点を収集するには、実行しているアカウントに、それらの2つのテーブルにアクセスするために十分な権限が付与されていることが必要です。
ドライバ	すべてのドライバの相違点 • インストールされているドライバ • ドライバのステータス

表8-1. 相違点のカテゴリ

カテゴリ	検出される相違点
レジストリ	レジストリの特定のセクションでの相違点。デフォルトでは、レジストリ セクションは収集されませんが、以下のレジストリ セクションの相違点を収集できます。 HKEY_CLASSES_ROOT HKEY_LOCAL_MACHINE **RegistrySections.xml**ファイルを編集して、収集するレジストリのセクションをカスタマイズできます。このファイルは**Program Files¥Micro Focus¥**DevPartner Studio**¥System Comparison¥data**フォルダにあります。 メモ： 64ビット バージョンのWindowsでは、このファイルは以下の場所にインストールされます。**¥Program Files (x86)¥Micro Focus¥**DevPartner Studio**¥System Comparison¥data** レジストリ キーのデータを収集できる権限を持っている必要があります。
ファイル	特定のパスから抽出したフォルダやファイルのプロパティの相違点。デフォルトでは、ファイルは収集の対象には含まれません。**FileSections.xml**ファイルを編集して収集するパスをカスタマイズできます。このファイルは**Program Files¥Micro Focus¥DevPartner Studio¥System Comparison¥data**フォルダにあります。 メモ： 64ビット バージョンのWindowsでは、このファイルは以下の場所にインストールされます。**¥Program Files (x86)¥Micro Focus¥DevPartner Studio¥System Comparison¥data**
.NETセキュリティ ポリシー	2つの個別のシステム構成でのセキュリティ ポリシーの相違点、または1つのコンピュータでの時間の経過によるセキュリティ ポリシーの変更を判断します。 • エンタープライズ • マシン • ユーザー
ハードウェア	• システム　メーカー、モデル、プロセッサの数、システム タイプ） • メモリ　MB単位） • 各プロセッサの詳細な説明　説明、動作速度、役割、状態）
ユーザー環境	プログラムの実行に影響を与える可能性のあるユーザー環境の相違点。スナップショットを取ったユーザーに依存します。 • 環境変数 • アクセシビリティ設定 • 海外向け設定
Windows Update	Windows Updateサービスの状態の相違点。この情報は、疑いのある更新で基本的なコンポーネントが変更された可能性があるかを判断するときなどに役立ちます。

レジストリ キーを比較する

システムの比較時には、レジストリ設定がしばしば重要になりますが、システムには数千ものレジストリ キーが登録されていることがあります。そのようなときは、比較するキーの範囲を絞ると便利です。比較するレジストリのセクションを指定するには、インストール パス（デフォルトでは、**Program Files¥Micro Focus¥DevPartner Studio¥System Comparison¥data**）の data フォルダにあるファイル**RegistrySections.xml**を使用します。

64 ビット バージョンの Windows では、このファイルは以下の場所にインストールされます。
¥Program Files (x86)¥Micro Focus¥DevPartner Studio¥System Comparison¥data

デフォルトでは、スナップショットにはレジストリ キーは含まれません。

メモ： System Comparison ユーティリティのスナップショット API（アプリケーション プログラム インターフェイス）でこのファイルを使用する場合、このファイルは、アプリケーションの実行可能ファイルよりも 1 レベル上の**¥data** フォルダに格納されている必要があります。たとえば、実行可能ファイルのパスが **...¥App¥bin¥MyApp.exe** である場合、このファイルのパスを **...App¥data¥RegistrySections.xml** にする必要があります。

HKEY_LOCAL_MACHINE と HKEY_CLASSES_ROOT にあるレジストリ エントリを比較できます。その他のレジストリ キーの比較はサポートされていません。

必要なセクションをいくつでも指定できます。

レジストリ キーのデータを収集できる権限を持っている必要があります。

構文

```
<Section categoryName="XXX">YYY</Section>
```

パラメータ

XXX　ユーザー インターフェイスに表示されるカテゴリ名です。この属性はオプションです。レジストリ キーを指定していない場合は、カテゴリ名として使用されます。

YYY　収集を再帰的に開始するレジストリ キーです。このキーには、接頭辞 HKEY_LOCAL_MACHINE または HKEY_CLASSES_ROOT は指定しません。たとえば、**KEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Rpc** のすべてを収集するには、以下のように指定します。
<Section categoryName="Microsoft RPC">SOFTWARE¥Microsoft¥Rpc</Section>
LOCAL_MACHINE キーまたは CLASSES_ROOT キーのすべてを収集するには、特殊文字「¥」を指定します。たとえば、**<Section categoryName="All">¥</Section>** のようにします。ただし、すべてのレジストリ キーの収集には長時間を要することに注意してください。
REG_BINARY タイプのキーでは、各キーの最初の 20 バイトだけが収集されます。

例

以下に、**RegistrySections.xml** ファイルの例を示します。

```
<RegistrySections>
<LocalMachine>
<!-- RPC 下ですべてのレジストリ キーを収集した場合の例 -->

<Section categoryName="Microsoft RPC">SOFTWARE¥Microsoft¥Rpc</
Section>

</LocalMachine>
<ClassesRoot>
<!-- ClassesRoot 下にあるすべてのデータを収集した場合の例（メガバイト単位のデータ）-->

<Section categoryName="All">¥</Section>
<Section categoryName="Shell Extensions">*¥shellex</Section>

</ClassesRoot>
</RegistrySections>
```

特定のファイルを比較する

デフォルトでは、個々のファイルの相違点は収集されません。システムの比較時には、特定のファイルを比較することがしばしば重要になりますが、比較するファイルの範囲を絞ると便利です。比較するファイルを指定するには、インストール パス（デフォルトでは、**Program Files¥Micro Focus¥DevPartner Studio¥System Comparison¥data**）のデータ フォルダにあるファイル**FileSections.xml** を使用します。

64 ビット バージョンの Windows では、このファイルは以下の場所にインストールされます。
¥Program Files (x86)¥Micro Focus¥DevPartner Studio¥System Comparison¥data

---

Required: System Comparison ユーティリティのスナップショット API（アプリケーション プログラム インターフェイス）でこのファイルを使用する場合、このファイルは、アプリケーションの実行可能ファイルよりも 1 レベル上の **¥data** フォルダに格納されている必要があります。たとえば、実行可能ファイルのパスが **...¥App¥bin¥MyApp.exe** である場合、このファイルのパスを **...App¥data¥FileSections.xml** にする必要があります。

比較対象となるファイルの各カテゴリは、**FileSections.xml** の独立したセクションに指定します。必要なセクションをいくつでも指定できます。

構文

```
<Section [categoryName="XXX"]
[filterPattern="{*,?}"][attributes="{yes,no}"]
[programAttributes="{yes,no}"]
[recurseSubDirectories="{yes,no}"]>YYY</Section>
```

パラメータ

categoryName	オプションの属性。XXXは、サブカテゴリとして表示される名前です。この属性を指定しないと、デフォルトで、カテゴリ名がフォルダ パスとして使用されます。
filterPattern	オプションの属性。ワイルドカード文字*（ゼロ個以上の文字）と?（1文字）を使用して、ファイル フィルタを指定します。この属性を指定しないと、「filter="*.*"」を指定したことと同じになります。
attributes	オプションのXML属性。この属性を指定しないと、「attributes="yes"」を指定したことと同じになります。この属性を「yes」と同等に設定すると、ユーティリティでは、以下が収集されます。 読み取り専用フラグ 暗号化 ファイル長 変更された日付 また、設定しないかぎり、Company属性とProduct属性は収集されません。読み取り専用やデバッグなどのブール値ファイル属性も同様です。
programAttributes	オプションの属性。この属性を指定しないと、「programAttributes=" yes"」を指定したことと同じになります。この属性を「yes」と同等に設定し、ファイル名の拡張子が「**.exe**」、「**.dll**」、「**.ocx**」、「**.cpl**」のいずれかであると、ユーティリティでは以下のプログラム バージョン情報が収集されます。 バージョン 言語 programAttributesは、「no」に設定すると便利です。たとえば、品質保証環境で、製品のインストール中に何らかのファイルの削除や追加が行われたけれども、一部のファイルのプロパティ（プログラム バージョン）などが変更された可能性がある場合などに役立ちます。
recurseSubDirectories	オプションのXML属性。この属性を指定しないと、「recurseSubDirectories=" yes"」を指定したことと同等になります。この属性を「yes」と同等に設定すると、ユーティリティでは、すべてのフォルダに関するファイル情報が再帰的に収集されます。
YYY	ファイル情報の収集を再帰的に開始するときのパス。

例

以下に、**FileSections.xml**ファイルの例を示します。

```
-->
- <FileSections>
- <!-- ファイル セクションの例 -->

<Section categoryName="My Product">c:¥somedir¥somesubdir</Section>
```

```
<Section categoryName="My bat files" attributes="yes"
filterPattern="*.bat" programAttributes="no"
recurseSubDirectories="no">c:¥diff</Section>

<Section categoryName="My Test Files" attributes="yes"
programAttributes="yes" recurseSubDirectories="yes">D:¥test</
Section>

</FileSections>
```

DevPartner Studio なしでインストールする

System Comparison は DevPartner Studio とは別にインストールされます。これは、2 台の異なるコンピュータを比較して、システムが異なるとアプリケーションの動作がなぜ異なるのかを判断する場合などに便利です。システムを比較して両者間の不一致を見つける場合、対象となるシステムに加える変更は最小限にすることが重要です。Visual Studio のオーバーヘッドや残りの DevPartner 機能なしで System Comparison のみをインストールすることにより、比較するシステム間に存在する重要な相違点に焦点を絞りやすくなります。

System Comparison をインストールするには、DevPartner Studio のインストール設定画面から、**[**DevPartner**System Comparison** のインストール **]** を選択して、インストール手順に従います。

System Comparison は DevPartner のライセンス契約に含まれているため、System Comparison の使用には DevPartner ライセンスが使用されます。ライセンスの問題の詳細については、『DevPartner Studio インストール ガイド』を参照してください。ただし、以下の点に注意が必要です。

- ノードロック（シングルシート）ライセンスまたはコンカレント ライセンスの場合、System Comparison による比較の実行中は 1 つのライセンスが使用されます。Comparison サービスを開始し、サービスでスナップショットを取る場合には、ライセンスは使用されません。
- 14 日間の評価期間で DevPartner Studio を実行している場合、この 14 日間は、System Comparison のユーザー インターフェイスを使用して比較を実行したときに開始します。Comparison サービスをインストールして開始し、スナップショットを取った時点では、評価期間は開始しません。

コマンド ラインから System Comparison ユーティリティを実行する

2 つのコマンド ライン インターフェイス（**`MicroFocus.Diff.CommandLine.exe`** と **`CommandLineDiff.exe`**）を使用して、データの収集と比較を自動化することができます。

- **`MicroFocus.Diff.CommandLine.exe`** は、コンピュータ システムの現在の状態を示すスナップショットを取得します。デフォルトでは、このスナップショットは、スナップショットの保存に使用した最後のフォルダに格納されます。ただし、コマンド ラインでパラメータを指定して、別のフォルダを指定することもできます。

以下に例を示します。

```
C:¥Program Files¥Micro Focus¥DevPartner Studio¥System
Comparison¥bin>MicroFocus.Diff.CommandLine.exe
```

```
C:¥Program Files¥Micro Focus¥DevPartner Studio¥System
Comparison¥bin>MicroFocus.Diff.CommandLine.exe c:¥MySnaps
```

64ビットバージョンのWindowsにインストールされている場合は、デフォルトのインストール フォルダは以下の場所になります。**¥Program Files (x86)¥Micro Focus¥DevPartner Studio¥System Comparison¥**

- **MicroFocus.Diff.CommandLineDiff.exe** は、既存の 2 つのスナップショット ファイルの値を比較して、結果としての相違点を出力ファイルに書き込みます。

  WIndows 7またはWindows Vistaで実行しているときは、出力フォルダに書き込む十分な権限があることを確認してください。

  必須のパラメータは、「computers 」 プレースホルダ）と、比較するファイルの名前です。オプションで、出力ファイルを書き込むフォルダを指定できます。

  以下に例を示します。

  ```
  C:¥Program Files¥Micro Focus¥DevPartner Studio¥System
  Comparison¥bin>MicroFocus.Diff.CommandLineDiff.exe computers
  SnapFile1 SnapFile2
  ```

  ```
  C:¥Program Files¥Micro Focus¥DevPartner Studio¥System
  Comparison¥bin>MicroFocus.Diff.CommandLineDiff.exe computers
  SnapFile1 SnapFile2 C:¥MyResults
  ```

  出力ファイルはXMLファイルであり、プログラムで読み取って比較の結果をチェックすることができます。この出力ファイルをSystem Comparisonユーティリティのユーザー インターフェイスで開くことはできません。

コマンド ライン プログラムは、System Comparisonユーティリティの**¥bin**フォルダ ォルダ（デフォルトでは、**¥Program Files¥Micro Focus¥DevPartner Studio¥System Comparison¥bin**）にあります。

ソフトウェア開発キット

System Comparisonには、ソフトウェア開発キット $DK）が含まれています。これは、以下のようなソフトウェア開発者向けの機能を提供します。

- スナップショット API アプリケーション プログラム インターフェイス）を使用して、関数コールをアプリケーションに埋め込み、アプリケーションの配置後にスナップショットを開始できる機能

  アプリケーション開発者はスナップショットAPIを使用して、配置されたアプリケーションからのスナップショット機能を制御することができます。アプリケーションの配置後に問題が発生した場合、埋め込まれたAPIコールを使用して、問題の診断を支援するスナップショットを開始できます。

- System Comparisonプラグインを記述することで、スナップショットを取るときに収集する追加情報を指定できる機能

ほとんどの比較では、System Comparisonユーティリティによって収集される情報カテゴリ（$paranumonly[TableTitle]> 252ページ）を参照）で十分です。システムを十分に比較するために追加情報が必要な場合は、データ取得用のプラグインを記述してSystem Comparisonユーティリティをカスタマイズすることができます。

API とプラグイン機能については、以下のセクションで説明します。

System Comparison のスナップショット API

System Comparisonのスナップショット API を使用すると、配置されたアプリケーション内部からスナップショット機能を制御できます。スナップショット API を使用して、以下の項目を指定することができます。

- スナップショットを保存する場所
- メッセージまたはエラーの処理方法
- 進行状況の報告方法
- カスタム プラグインを使用する場合は、そのプラグインの場所

スナップショット API の情報は、System Comparison インストール フォルダの以下の2つのフォルダにあります。

- **System Comparison¥redistributable** フォルダ。ユーザーがアプリケーションのインストールに含めるためにライセンスを取得したアセンブリが格納されます。

  スナップショット API アセンブリは、Micro Focus ソフトウェア ライセンス契約の条項に従って再配布できます。スナップショットを取るときには、ソフトウェアのライセンスは必要ありません。スナップショットを表示して比較するには、ライセンスを取得したDevPartnerシステムに送信する必要があります。

- **¥System Comparison¥sdk¥SnapshotAPI** フォルダ。API をアプリケーションで使用する方法を示したサンプル アプリケーション（**SampleSnapshotAPI.cs**）が格納されています。

  この API の使用方法を理解するには、**SampleSnapshotAPI.cs** を参照してください。

スナップショット API は、VB.NET、C#、およびマネージ C++ からアクセス可能で、.NET Framework 1.1 と 2.0 で作成したアプリケーションで使用することができます。

以下では、スナップショット API 用に実装されているクラスとメソッドを、**SampleSnapshotAPI.cs** アプリケーションで示されるとおりに説明します。

メモ： スナップショット API を使用する場合、アプリケーションのパスに、アプリケーションの実行可能ファイルよりも1レベル上の **¥data** フォルダを含める必要があります。**RegistrySections.xml** ファイルと **FileSections.xml** ファイルは、たとえ使用されていない場合でも、**¥data** フォルダに存在する必要があります。たとえば、実行可能ファイルのパスが **...¥App¥bin¥MyApp.exe** である場合は、**...App¥data¥RegistrySections.xml** と **...App¥data¥FileSections.xml** が存在する必要があります。

スナップショットを取る

`MicroFocus.Diff.Collector` によりクラス `SnapshotAPI` が実装されます。この `SnapshotAPI` クラスを使用してスナップショットを取ることができます。

Public SnapshotAPI （ILoggable logger、 IProgressStatus progressStatusInterestedParty、 String pluginsSubDirectoryName）	**Logger**：イベントとエラーを処理するクラスのインスタンス。オプションとしてNULLを渡すことができますが、ロガーによって問題のトラブルシューティングが大幅に容易になるため、実装することをお勧めします。 **progressStatusInterestedParty**：進行状況メッセージを処理するクラスのインスタンス。スナップショット処理は時間がかかることがあるため、アプリケーションではユーザーにフィードバックを提供することが重要な場合があります。オプションとして、NULLを渡します。 **pluiginsSubDirectoryName**：データ取得用のアセンブリが格納されている実行可能ファイルのフォルダのサブフォルダの名前。プラグインを使用しない場合は、既存の空のフォルダを指定するか、NULLを渡すことができます。

`SnapshotAPI` クラスにより、以下の3つのメソッドが実装されます。

public string TakeSnapshot （string snapshotDirectory）	スナップショットを取り、指定されたフォルダに保存します。
public int GetNumberSteps ()	進行状況オブジェクトが受け取るステップの合計数を指定します。その後、この数に従って進行状況を設計できます（`ProgressStatus` インターフェイスの詳細については、262ページを参照してください）。
public void Dispose ()	スナップショット オブジェクトはDispose可能オブジェクトです。

以下の例は、最も基本的なスナップショット機能を表しています。

```
using ( SnapshotAPI snapshoter = new SnapshotAPI( null, null, null ) )
{
string snapFile = snapshoter.TakeSnapshot ( userSnapshotDirectory );}
```

この例ではスナップショットを取りますが、エラー、メッセージ、および進行状況が追跡されないため、実稼働設定ではあまり役に立ちません。

メッセージをログに記録する

`MicroFocus.Diff.LoggableInterface` を使用して、スナップショット処理中に戻されたエラーとメッセージの報告を制御することができます。アプリケーションで、ログ メカニズムを作成してこのインターフェイスを実装し、メッセージを適切な出力デバイスに送信します。たとえば、サンプル アプリケーションでは、メッセージをコンソールのログに記録する `ConsoleLogger` クラスが実装されます。

このインターフェイスは、以下の2つのメソッドで構成されます。

`void Log( string message )`	通常のステータス メッセージをログに記録するときに呼び出します。
`void LogError( string message )`	エラー メッセージをログに記録するときに呼び出します。

進行状況を報告する

`MicroFocus.Diff.ProgressStatus` インターフェイスを実装することにより、スナップショットの進行状況の報告と表示が可能になります。このインターフェイスは、以下の3つのメソッドで構成されます。

`void OneStep ()`	進行状況表示を1ステップ進めるように関係先に通知するために呼び戻すメソッド
`void MultiSteps ( int nbrSteps )`	進行状況表示を数ステップ進めるように関係先に通知するために呼び戻すメソッド
`void UpdateStatus (String newStatus )`	新しい状況を処理するように関係先に通知するために呼び戻すメソッド newStatus 文字列は新しい状況を表します。通常は、進行状況 UI 要素の一部として表示されます。

プラグインを記述する

ほとんどの比較では、System Comparison ユーティリティによって収集される情報カテゴリ（<$paranumonly[TableTitle]> 252ページ）を参照）で十分です。システムを十分に比較するために追加情報が必要な場合は、データ取得用のプラグインを記述して System Comparison ユーティリティをカスタマイズすることができます。このセクションでは、データ取得用プラグインを定義し、提供されているサンプルを使用してプラグインの動作を実演し、独自のデータ取得用プラグインを作成する方法について説明します。

Required: DevPartner Studio のバージョン 9.0 よりも前のバージョンで作成した既存のプラグインは、リビルドしなければ DevPartner Studio9.0 では使用できません。

プラグインとは

プラグインとは、`MicroFocus.Diff.PluginInterface.IPluggableDataExtractor` インターフェイスを実装する1つまたは複数のタイプが含まれる .Net アセンブリです。プラグインにより、比較のために収集されるデータの高レベルのカテゴリが定義されます。プラグインによってデータが抽出され、基本要素に階層的に XML 要素を追加することによって、このデータが呼び出し側に XML 形式で渡されます。

プラグインは、製品のインストール フォルダの **`bin/plugins`** にあります。このフォルダ内のすべての .NET アセンブリは Comparison サービスによってロードされ、`IPluggableDataExtractor` インターフェイスを実装するすべてのタイプがインスタンス化され、プラグインのリストに配置されて、データの抽出時に呼び出しが行われます。

プラグインを記述するメカニズムの基礎に習熟するため、System Comparison には以下の2つのサンプル ファイルが含まれています。

- サンプル プラグイン **SamplePlugin.cs**。単純なプラグインの構造を示します。このサンプルでは重要なデータは収集されませんが、最初のサブカテゴリの、2 つめのデータ ポイントでのタイムスタンプの違いが常に表示されます。プラグインに実装されているメソッドの詳細については、**¥SDK¥Plugin** 内のファイル **IPluggableDataExtractor.cs** を参照してください。
- サンプル プラグインを実行するプログラム **TestDriver.cs**。プラグインのメカニズムの基礎に習熟するために使用できます。その後は、独自にカスタマイズしたプラグインを実行するために使用できます。期待した情報をプラグインで取得したあとは、System Comparison の **bin/plugins** フォルダに配置しておくと、System Comparison のユーザー インターフェイスまたはコマンド ライン インターフェイスを使用して実行できます。

サンプル プラグインのステップごとの手順

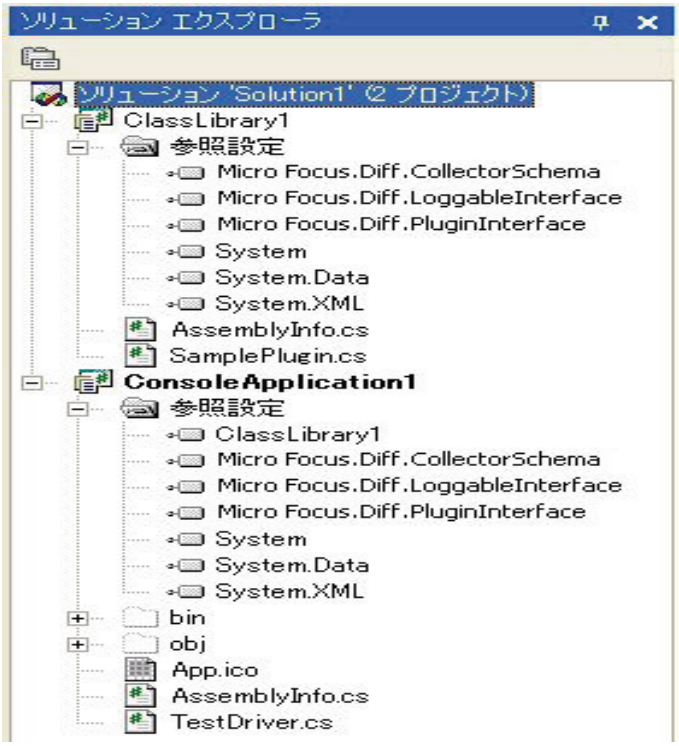
プラグインの使用に習熟するには、**TestDriver.cs** サンプル ファイルと **SamplePlugin.cs** サンプル ファイルを使用します。どちらのファイルも **¥sdk¥Plugin** フォルダ (デフォルトでは、**C:¥Program Files¥Micro Focus¥DevPartner Studio¥System Comparison¥sdk¥Plugin**) にあります。

メモ : 64 ビット バージョンの Windows にインストールされている場合は、デフォルト フォルダは以下の場所になります。**¥Program Files (x86)¥Micro Focus¥DevPartner Studio¥System Comparison¥sdk¥Plugin**

ヒント : プラグインは、Visual Studio の現在サポートされている任意のバージョンで作成できます。使用する予定の機能が含まれている .NET Framework のバージョンに一致するバージョンの Visual Studio を使用してください。

サンプル ファイルをビルドしてテストするには :

1. Visual Studio を使用してソリューションを作成します。
2. このソリューションに、以下の 2 つの C# プロジェクトを追加します。
   - ◇ `ClassLibrary1` (タイプ クラス ライブラリ) : このプロジェクトはプラグインの開発に使用します。
   - ◇ `ConsoleApplication1` (タイプ コンソール) : このプロジェクトはプラグインのデバッグに使用します。



3 ClassLibrary1プロジェクトで、以下の操作を実行します。

a **SamplePlugin.cs**を追加します（デフォルトでは、**C:¥Program Files¥Micro Focus¥DevPartner Studio¥System Comparison¥sdk¥Plugin**にあります）。

64ビット バージョンのWindowsにインストールされている場合は、デフォルト フォルダは以下の場所になります。**¥Program Files (x86)¥Micro Focus¥DevPartner Studio¥System Comparison¥**

b 以下のアセンブリに参照を追加します（デフォルトでは、再配布可能なフォルダ**C:¥Program Files¥Micro Focus¥DevPartner Studio¥System Comparison¥redistributable**から行います）。

MicroFocus.Diff.PluginInterface.dll

MicroFocus.Diff.LoggabcomputerleInterface.dll

MicroFocus.Diff.CollectorSchema.dll

4 ConsoleApplication1プロジェクトで、以下の操作を実行します。

a ソリューション エクスプローラから**Class1.cs**を削除します（存在する場合）。

b プロジェクトに**TestDriver.cs**ファイルを追加します（デフォルトでは、**C:¥Program Files¥Micro Focus¥DevPartner Studio¥System Comparison¥sdk¥Plugin**にあります）。

c 以下のアセンブリに参照を追加します（再配布可能なフォルダから行います）。

MicroFocus.Diff.PluginInterface.dll

MicroFocus.Diff.LoggableInterface.dll

MicroFocus.Diff.CollectorSchema.dll

d ClassLibrary1に参照を追加します。

e このプロジェクトをスタートアップ プロジェクトとして設定します。

5 ソリューションをビルドして実行します。デバッグ モードで実行して、サンプルをステップごとに実行すると、プラグインの基本動作を理解することができます。サンプルのプラグイン データが含まれるXML出力ファイルが生成されます。このファイルの名前は**pluginOutput.xml**で、テスト ドライバを実行したフォルダに置かれます。

```
-<testPlugin>

- <c n="Sample Data Extractor Plug-in">
- <c n="sampleSubCategory1">
<s n="data1">data1 actual value</s>
<s n="data2">data2 actual value 4/3/2006 10:42:34 AM</s>
</c>
- <c n="sampleSubCategory2">
<s n="data1">data1 actual value</s>
<s n="data2">data2 actual value</s>
</c>
</c>

</testPlugin>
```

TestDriverを使用してサンプル プラグインを実行したあとは、以下の手順に従って、System Comparisonユーティリティのユーザー インターフェイスまたはコマンド ライン インターフェイスで使用することができます。

1 **ClassLibrary1.dll**をプラグインのサブフォルダにコピーします。

2 System Comparisonのユーザー インターフェイスまたはコマンド ライン インターフェイスを使用してスナップショットを取り、さらに2つめのスナップショットを取ります。

3 この2つのスナップショットを比較します。SamplePluginではタイムスタンプ データが収集されるため、2つのスナップショットによってこの違いが示されます。

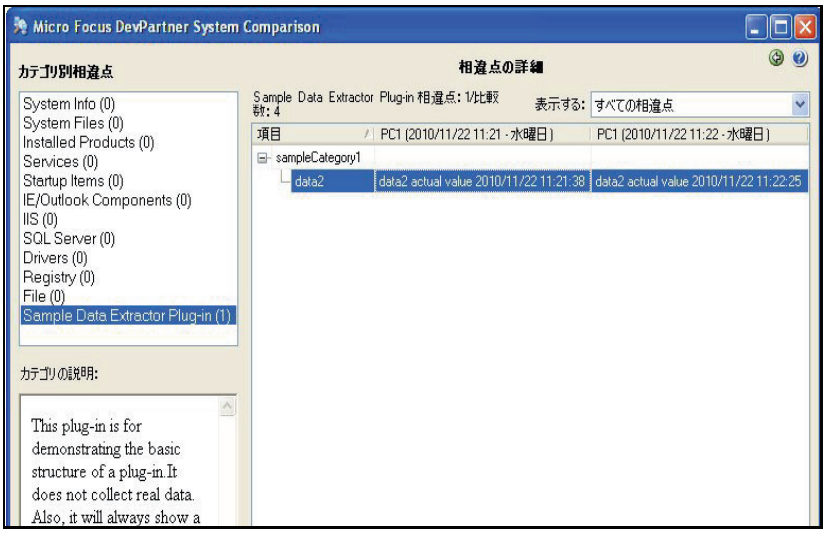


図8-4 サンプル プラグインの結果ウィンドウ

独自のプラグインを作成してテストする

プラグインのメカニズムの基礎に習熟したあとは、独自にカスタマイズしたプラグインを設計し、関心のあるデータの収集を開始できます。

プラグインを設計するときは、収集したデータの階層に特に注意してください。関心のある値を判断できるような階層に設計してください。データ階層に一致しない値が見つかると、その階層の残りのデータは比較されません（プラグインの後続バージョンでデータ階層を変更する詳細については、「配置したプラグインを変更する」（266ページ）を参照）。

トラブルシューティングを容易にするために、プラグインを`TestDriver`で実行することができます。プラグインの出力に満足したあとは、以下の手順に従って、System Comparisonのコマンド ライン インターフェイスでテストできます。

1. 作成したプラグインを、製品のインストール フォルダ**`plugins`**にコピーします（**`TestDriver.exe`**ファイルをコピーする必要はありません。このファイルはプラグインのテストのみに使用します）。
2. プラグインによって収集される領域に相違点がある2台のコンピュータで、コマンド ライン プログラム（**`<product dir>¥bin¥MicroFocus.Diff.CommandLine.exe`**）を実行します。
3. System Comparisonのユーザー インターフェイスを使用して、2つのスナップショットを比較します。相違点が表示されます。
4. System Comparisonサービスを再起動し、プラグインでスナップショットが作成されるときに指定されたデータが含まれるようにします。

一時フォルダで**`DifferEvent.log`**を確認（正確な場所はTEMP環境変数を参照）し、発生したすべての問題のトラブルシューティングを実行します。プラグインが見つかり、インスタンス化されると、イベントのログが記録されます。ロード、アンロード、データ取得などの呼び出し中に発生したその後のエラーでも、ログにイベントが生成されます。

さらに、`IPluggableDataExtractor.GetData`呼び出しの`ILoggable traceLogger`パラメータを使用してログに記録したエラーもこのファイルに書き込まれます。**`IPluggableExtractor.cs`**を参照してください。

配置したプラグインを変更する

プラグインを配置したあとに、収集されるデータの変更が決定される場合があります。古いスナップショット ファイルを、プラグインの新しいバージョンで作成されたスナップショットと比較すると、収集されたデータが一致しません。System Comparisonユーティリティでは、このような不一致が相違点として識別されるため、混乱の原因になる場合があります。

バージョンのメジャー番号とマイナー番号を使用することによって、不一致の処理方法を制御できます。プラグインのバージョンのメジャー番号がスナップショット間で異なる場合は、System Comparisonユーティリティによって「プラグイン スキーマに互換性がありません」というメッセージが報告されます。プラグインのバージョンのマイナー番号が異なる場合は、System Comparisonユーティリティによって、新しいデータの状態は古いスナップショットでは「不明」であると識別されます。

データを削除したり、収集したデータの階層を変更したりするためにプラグインを変更するときは、バージョンのメジャー番号を変更することをお勧めします。データを追加するためにプラグインを変更する場合、通常は、マイナーバージョンを変更するのみで十分です。

このメカニズムに習熟するために、以下に示すように`SamplePlugin`でバージョン番号を変更して試すことができます。バージョン番号の初期設定は1.0です。

```
public PluginSchemaVersion PluginVersion
{
get { return new PluginSchemaVersion( 1, 0 );}
}
```

メモ： プラグインの置き換えまたは削除が必要な場合は、はじめにSystem　Comparisonサービスを停止し、System Comparisonのユーザー インターフェイスを終了して、ファイルがオペレーティング システムにロックされないようにする必要があります。

プラグイン スキーマに関する重要事項

プラグイン スキーマに習熟するために、任意のスナップショットをチェックできます。スナップショットには、プラグインからのデータが含まれています。詳細については、**¥sdk¥Plugin**内の**diff-plugin-schema.xsd**ファイルを参照してください。使用可能な要素と属性の一部を示したサンプルXMLの一部を注釈付きで以下に示します。

`<c`	プラグインの最も外側のカテゴリ ノード
`n="MyApplication"`	プラグインの名前。このテキストは、カテゴリのリスト　(UIの左側)に表示されます。
`descrip="text"`	このテキストは、カテゴリを選択したときに、UIの左下に表示されます。
`schema="1">`	「1」に設定します。プラグインの新しいバージョンと、それまでのリリースとの間に互換性がないときは変更します。
`<c`	ネストされているすべてのカテゴリは、UIのメイン ウィンドウに表示されます。
`n="MyCategory"`	UIのメイン ウィンドウに表示される名前。比較に使用されます。
`missing="text"`	オプションです。このカテゴリが他のスナップショットで見つからないときに表示するテキストです。バージョン情報にすることも、「インストール済み」のような簡単なテキストにすることもできます。
`error="text">`	オプションです。このカテゴリのデータをフェッチするときにプラグインでエラーが発生した場合は、そのエラーをここに含めて、UIに表示することができます。
`<s`	すべてのデータは文字列データです。
`n="MyData"`	UIのメイン ウィンドウに表示される名前。比較に使用されます。
`search="t1 t2"`	オプションです。UIの「検索」リンクに使用される検索語。Googleに渡されます。
`error="text">`	オプションです。データをフェッチするときにプラグインでエラーが発生した場合は、そのエラーをここに含めて、UIに表示することができます。
`Actual Data Value`	レジストリまたはその他の設定から得た対象データ
`</s>`	
`</c>`	
`</c>`	

再配布可能なアセンブリについて

MicroFocus.Diff.PluginInterface.dll、**MicroFocus.Diff.LoggableInterface.dll**、および**MicroFocus.Diff.CollectorSchema.dll**の現在のバージョンは1.0.0.0ですこれらのアセンブリのバージョン番号が変わらないかぎり、カスタマイズしたプラグインはSystem Comparisonユーティリティの今後のバージョンでも引き続き機能します。アセンブリに大きな変更が行われた場合はバージョン番号が増加するため、新しいアセンブリに対してプラグインをリビルドする必要があります。

付録 **A**

DevPartner Studio サポートされるプロジェクト タイプ

この章では、上記の DevPartner Studio 機能でそれぞれサポートされているプロジェクト タイプを一覧にした表を記載します。

サポートされるプロジェクト タイプ

DevPartner Studio は、Visual Studio 統合開発環境、マネージ プロジェクト タイプおよびアンマネージ プロジェクト タイプ、およびプログラミング言語を含む多数のソフトウェア開発環境で動作します。

表 **A-1** サポートされる Visual Studio バージョンおよび言語リファレンス

統合開発環境	マネージ／アンマネージ	言語
Visual Studio 2010 (VS2010)	マネージ	Visual Basic Visual C++ Visual C#
	アンマネージ	Visual C++
Visual Studio 2008 (VS2008)	マネージ	Visual Basic Visual C++ Visual C#
	アンマネージ	Visual C++
Visual Studio 2005 (VS2005)	マネージ	Visual Basic Visual C++ Visual C# Visual J#
	アンマネージ	Visual C++

以降のページでは、DevPartner Studio の各個別機能でサポートされる IDE、プロジェクト タイプ、および言語について説明します。

エラー検出でサポートされているプロジェクト タイプ

アプリケーション プロジェクトでは、x86 実行ファイルが構築されます。DevPartner エラー検出では、以下のプロジェクト タイプがサポートされます。

表 **A-2** エラー検出でのマネージ プロジェクト タイプのサポート

Visual Studio のバージョン	アプリケーション プロジェクト タイプ	サポートされる言語
VS2010	ATL プロジェクト CLR コンソール アプリケーション 空の CLR プロジェクト カスタム ウィザード Makefile プロジェクト MFC ActiveX コントロール MFC アプリケーション MFC DLL Win32 コンソール アプリケーション Win32 プロジェクト	C++
	アクティビティ ライブラリ アクティビティ デザイナー ライブラリ ASP.NET AJAX サーバー コントロール[1] ASP.NET AJAX サーバー コントロール エクステンダ[1] ASP.NET MVC 2 Empty Web アプリケーション ASP.NET MVC 2 Web アプリケーション ASP.NET サーバー コントロール[1] コンソール アプリケーション Crystal Reports アプリケーション 空のワークフロー プロジェクト[3] シーケンシャル ワークフロー コンソール アプリケーション[3] ステート マシン ワークフロー サービス ライブラリ シンジケーション サービス ライブラリ Visual Basic SQL CLR データベース プロジェクト Visual C# SQL CLR データベース プロジェクト Windows フォーム アプリケーション Windows フォーム コントロール ライブラリ Windows サービス WPF アプリケーション[2] WPF ブラウザ アプリケーション[2] WPF カスタム コントロール ライブラリ[2] WPF ユーザー コントロール ライブラリ[2]	C#、VB

表 **A-2** エラー検出でのマネージ プロジェクト タイプのサポート

Visual Studio のバージョン	アプリケーション プロジェクト タイプ	サポートされる言語
	クラス ライブラリ エンプティ プロジェクト 共有アドイン テスト プロジェクト[3] Visual Studio アドイン Visual Studio Package	C++、C#、VB
[1] JavaScript、非同期 XML [2] 標準 VB または C# アプリケーション [3] 生成コード、.NET Framework 3.0 以降		
VS2008	MFC アプリケーション MFC DLL Win32 コンソール アプリケーション Win32 プロジェクト Win32 スマート デバイス プロジェクト　(メモを参照)	C++
	Crystal Reports アプリケーション テスト プロジェクト Windows コントロール ライブラリ Windows アプリケーション Windows フォーム アプリケーション Windows フォーム コントロール ライブラリ WPF アプリケーション[1] WPF ユーザー コントロール ライブラリ[1] WPF カスタム コントロール ライブラリ[1] WPF ブラウザ アプリケーション[1]	C#、VB
	コンソール アプリケーション Windows サービス	C++、C#、VB
[1] 生成コード、.NET Framework 3.0 以降 [2] VS 2008 SP1 以降		

表 **A-2** エラー検出でのマネージ プロジェクト タイプのサポート

Visual Studio のバージョン	アプリケーション プロジェクト タイプ	サポートされる言語
VS2005	MFC アプリケーション MFC DLL MFC ActiveX コントロール MFC ISAPI Extension DLL Win32 コンソール アプリケーション Win32 プロジェクト Win32 スマート デバイス プロジェクト （メモを参照） Windows フォーム コントロール ライブラリ	C++
	電卓スタート キット	J#
	Crystal Reports アプリケーション Windows コントロール ライブラリ Windows アプリケーション	C#、J#、VB
	コンソール アプリケーション Windows サービス	C++、C#、J#、VB
MFC - Microsoft Foundation Class		

メモ： Win32 スマート デバイス プロジェクト タイプ（ソリューション プラットフォームが Win32 に設定されたスマート デバイス プロジェクト タイプ）を DevPartner エラー検出でサポートするためには、エミュレータではなく、開発コンピュータ上で実行する必要があります。

ホスト型プロジェクトでは、x86 DLL が構築されます。この DLL をテストする場合は、アプリケーション内でホストする必要があります。DevPartner エラー検出では、他の実行ファイル内でホストされている場合にだけ以下のプロジェクト タイプがサポートされます。

表 **A-3** 他の実行ファイル内でプロジェクトがホストされる場合にサポートされる

Visual Studio のバージョン	他の実行ファイル内でホストされる場合にサポートされるプロジェクト タイプ	サポートされる言語
VS2010	ATL プロジェクト 拡張ストアド プロシージャ DLL MFC ActiveX コントロール MFC DLL Windows フォーム コントロール ライブラリ	C++
	Web コントロール ライブラリ	C#、VB
	クラス ライブラリ Windows コントロール ライブラリ	C++、C#、VB

Visual Studio のバージョン	他の実行ファイル内でホストされる場合にサポートされるプロジェクト タイプ	サポートされる言語
VS2008	ATL プロジェクト ATL サーバー プロジェクト ATL スマート デバイス プロジェクト 拡張ストアド プロシージャ DLL MFC ActiveX コントロール MFC DLL MFC ISAPI Extension DLL MFC スマート デバイス ActiveX コントロール MFC スマート デバイス アプリケーション MFC スマート デバイス DLL Windows フォーム コントロール ライブラリ	C++
	Web コントロール ライブラリ	C#、VB
	クラス ライブラリ Windows コントロール ライブラリ	C++、C#、VB
VS2005	ATL プロジェクト ATL サーバー プロジェクト ATL スマート デバイス プロジェクト 拡張ストアド プロシージャ DLL MFC ActiveX コントロール MFC DLL MFC ISAPI Extension DLL MFC スマート デバイス ActiveX コントロール MFC スマート デバイス アプリケーション MFC スマート デバイス DLL Windows フォーム コントロール ライブラリ	C++
	Web コントロール ライブラリ	C#、J#、 VB
	クラス ライブラリ Windows コントロール ライブラリ	C++、C#、J#、 VB
ATL - Active Template Library MFC - Microsoft Foundation Class		

コード レビューでサポートされているプロジェクト タイプ

以下の表に、DevPartner Studio コード レビューでサポートされるプロジェクト タイプのリストを示します。

表 **A-4** コード レビューでのマネージ プロジェクト タイプのサポート

Visual Studio バージョン	マネージ プロジェクト タイプ	サポートされる言語
VS2010	アクティビティ ライブラリ アクティビティ デザイナー ライブラリ ASP.NET MVC 2 Empty Web アプリケーション ASP.NET MVC 2 Web アプリケーション ASP.NET Dynamic Data Entities Web アプリケーション ASP.NET Dynamic Data Linq to SQL Web アプリケーション ASP.NET Empty Web アプリケーション ASP.NET Web アプリケーション ASP.NET Web サービス ASP.NET Web サービス アプリケーション ASP.NET Web サイト ASP.NET AJAX サーバー コントロール[1、4] ASP.NET AJAX サーバー コントロール エクステンダ[1] ASP.NET サーバー コントロール[1] クラス ライブラリ コンソール アプリケーション Crystal Reports アプリケーション エンプティ プロジェクト 空のワークフロー プロジェクト[2] Crystal Reports アプリケーション シーケンシャル ワークフロー コンソール アプリケーション[2] シーケンシャル ワークフロー ライブラリ[2] シーケンシャル ワークフロー サービス ライブラリ ステート マシンのワークフロー コンソール アプリケーション[2] ステート マシンのワークフロー ライブラリ[2] ステート マシン ワークフロー サービス ライブラリ シンジケーション サービス ライブラリ テスト プロジェクト[2] WCF サービス アプリケーション WCF ワークフロー サービス アプリケーション Windows アプリケーション Windows フォーム アプリケーション	C#、VB

表 **A-4** コード レビューでのマネージ プロジェクト タイプのサポート

Visual Studio バージョン	マネージ プロジェクト タイプ	サポートされる言語
VS2010 （続き）	Windows フォーム コントロール ライブラリ Windows サービス ワークフロー アクティビティ ライブラリ[2] Workflow コンソール アプリケーション WPF フォーム アプリケーション[3] WPF ブラウザ アプリケーション[3] WPF カスタム コントロール ライブラリ[3] WPF ユーザー コントロール ライブラリ[3]	C#、VB
[1]ASP .NET Web アプリケーション [2] 標準 VB または C# アプリケーション [3] 生成コード、.NET Framework 3.0 以降		
VS2008	ASP.NET Web アプリケーション ASP.NET Web サービス ASP.NET Web サイト ASP.NET AJAX サーバー コントロール[1] ASP.NET AJAX サーバー コントロール エクステンダ[1] ASP.NET サーバー コントロール[1] クラス ライブラリ コンソール アプリケーション Crystal Reports アプリケーション モバイル Web アプリケーション テスト プロジェクト[2] 空のワークフロー プロジェクト[2] シーケンシャル ワークフロー コンソール アプリケーション[2] シーケンシャル ワークフロー ライブラリ[2] ステート マシンのワークフロー コンソール アプリケーション[2] ステート マシンのワークフロー ライブラリ[2] ワークフロー アクティビティ ライブラリ[2] Web コントロール ライブラリ Windows アプリケーション Windows コントロール ライブラリ Windows サービス WPF アプリケーション[3] WPF ユーザー コントロール ライブラリ[3] WPF カスタム コントロール ライブラリ[3]	C#、VB

表 **A-4** コード レビューでのマネージ プロジェクト タイプのサポート

Visual Studio バージョン	マネージ プロジェクト タイプ	サポートされる言語
	WPF ブラウザ アプリケーション[3] WCF サービス アプリケーション	
[1]ASP .NET Web アプリケーション [2] 標準 VB または C# アプリケーション [3] 生成コード、.NET Framework 3.0 以降		
VS2005	ASP.NET Web アプリケーション ASP.NET Web サービス ASP.NET Web サイト クラス ライブラリ コンソール アプリケーション Crystal Reports アプリケーション モバイル Web アプリケーション Web コントロール ライブラリ Windows アプリケーション Windows コントロール ライブラリ Windows サービス	C#、VB

カバレッジ分析、パフォーマンス分析、メモリ分析、およびパフォーマンス エキスパートでサポートされているプロジェクト タイプ

以下の表に、DevPartner Studio 分析でサポートされるプロジェクトの一覧を示します。

表 **A-5** カバレッジ分析、パフォーマンス分析、メモリ分析、およびパフォーマンス エキスパートでサポートされているプロジェクト タイプ

Visual Studio のバージョン	プロジェクト タイプ	サポートされる言語
VS2010	ATL プロジェクト CLR コンソール アプリケーション 空の CLR プロジェクト カスタム ウィザード Makefile プロジェクト MFC ActiveX コントロール MFC アプリケーション MFC DLL Win32 コンソール アプリケーション Win32 プロジェクト	C++
	アクティビティ ライブラリ アクティビティ デザイナー ライブラリ ASP.NET AJAX サーバー コントロール[1、4] ASP.NET AJAX サーバー コントロール エクステンダ[1] ASP.NET MVC 2 Empty Web アプリケーション ASP.NET MVC 2 Web アプリケーション ASP.NET Dynamic Data Entities Web アプリケーション ASP.NET Dynamic Data Linq to SQL Web アプリケーション ASP.NET Empty Web アプリケーション ASP.NET サーバー コントロール[1] ASP.NET Web アプリケーション ASP.NET Web サービス アプリケーション コンソール アプリケーション Crystal Reports アプリケーション 空のワークフロー プロジェクト[3] Crystal Reports アプリケーション シーケンシャル ワークフロー コンソール アプリケーション[3] シーケンシャル ワークフロー ライブラリ[3] シーケンシャル ワークフロー サービス ライブラリ ステート マシンのワークフロー コンソール アプリケーション[3]	C#、VB

表 **A-5** カバレッジ分析、パフォーマンス分析、メモリ分析、およびパフォーマンス エキスパートでサポートされているプロジェクト タイプ

Visual Studio のバージョン	プロジェクト タイプ	サポートされる言語
VS2010 （続き）	ステート マシンのワークフロー ライブラリ[3] ステート マシン ワークフロー サービス ライブラリ シンジケーション サービス ライブラリ Visual Basic SQL CLR データベース プロジェクト Visual C# SQL CLR データベース プロジェクト WCF ワークフロー サービス アプリケーション WCF サービス アプリケーション[3] WCF サービス ライブラリ Windows フォーム アプリケーション Windows フォーム コントロール ライブラリ Windows サービス ワークフロー アクティビティ ライブラリ[3] Workflow コンソール アプリケーション WPF アプリケーション[2] WPF ブラウザ アプリケーション[2] WPF カスタム コントロール ライブラリ[2] WPF ユーザー コントロール ライブラリ[2]	C#、VB
	クラス ライブラリ エンプティ プロジェクト 共有アドイン テスト プロジェクト[3] Visual Studio アドイン Visual Studio Package	C++、C#、VB
	クラス ライブラリ コンソール アプリケーション エンプティ プロジェクト リンク ライブラリ[6] シンジケーション サービス ライブラリ WCF サービス ライブラリ Windows アプリケーション[6] Windows フォーム アプリケーション Windows フォーム コントロール ライブラリ Windows サービス WPF アプリケーション WPF ユーザー コントロール ライブラリ	COBOL for .NET （Micro Focus Studio Enterprise Edition）[5]

表 **A-5** カバレッジ分析、パフォーマンス分析、メモリ分析、およびパフォーマンス エキスパートでサポートされているプロジェクト タイプ

Visual Studio のバージョン	プロジェクト タイプ	サポートされる言語
[1] JavaScript、非同期 XML [2] 生成コード、.NET Framework 3.0 以降 [3] 標準 VB または C# アプリケーション [4] カバレッジ分析とパフォーマンス分析のみ [5] Visual Studio 2008 Service Pack 2 以降 [6] ネイティブ (インストゥルメンテーションがないエラー検出、カバレッジ分析、パフォーマンス分析作業) MFC - Microsoft Foundation Class		
VS2008	ATL プロジェクト ATL サーバー プロジェクト ATL Server Web サービス ASP.NET Web サービス CLR コンソール アプリケーション 空の CLR プロジェクト 共有アドイン MFC ActiveX コントロール MFC アプリケーション MFC DLL Win32 コンソール アプリケーション Win32 プロジェクト	C++
	ASP.NET Web サイト ASP.NET AJAX サーバー コントロール[1、4] ASP.NET AJAX サーバー コントロール エクステンダ[1] ASP.NET サーバー コントロール[1] ASP.NET Web アプリケーション ASP.NET Web サービス アプリケーション コンソール アプリケーション Crystal Reports アプリケーション Crystal Reports アプリケーション SQL Server プロジェクト WPF アプリケーション[2] WPF ブラウザ アプリケーション[2] WPF カスタム コントロール ライブラリ[2] WPF ユーザー コントロール ライブラリ[2] WCF サービス アプリケーション[3] テスト プロジェクト[3]	C#、VB

表 **A-5** カバレッジ分析、パフォーマンス分析、メモリ分析、およびパフォーマンス エキスパートでサポートされているプロジェクト タイプ

Visual Studio のバージョン	プロジェクト タイプ	サポートされる言語
VS2008 （続き）	空のワークフロー プロジェクト[3] シーケンシャル ワークフロー コンソール アプリケーション[3] シーケンシャル ワークフロー ライブラリ[3] ステート マシンのワークフロー コンソール アプリケーション[3] ステート マシンのワークフロー ライブラリ[3] ワークフロー アクティビティ ライブラリ[3] Visual Studio アドイン Visual Studio 統合パッケージ Web コントロール ライブラリ Windows アプリケーション Windows コントロール ライブラリ	C#、VB
	クラス ライブラリ Windows フォーム アプリケーション Windows フォーム コントロール ライブラリ Windows サービス	C++、C#、VB
	Windows フォーム Windows フォーム コントロール ライブラリ シンジケーション サービス ライブラリ WPF アプリケーション クラス ライブラリ コンソール アプリケーション Windows サービス WCF サービス ライブラリ WPF ユーザー コントロール ライブラリ	COBOL for .NET （Micro Focus Studio Enterprise Edition）[5]

[1] JavaScript、非同期 XML

[2] XAML 生成コード、.NET Framework 3.0 以降

[3] 標準 VB または C# アプリケーション

[4] カバレッジ分析とパフォーマンス分析のみ

[5] Visual Studio 2008 Service Pack 2 以降

MFC - Microsoft Foundation Class

表 **A-5** カバレッジ分析、パフォーマンス分析、メモリ分析、およびパフォーマンス エキスパートでサポートされているプロジェクト タイプ

Visual Studio のバージョン	プロジェクト タイプ	サポートされる言語
VS2005	ATL プロジェクト ATL サーバー プロジェクト ATL Server Web サービス ASP.NET Web サービス CLR コンソール アプリケーション 空の CLR プロジェクト 共有アドイン MFC ActiveX コントロール MFC アプリケーション MFC DLL Win32 コンソール アプリケーション Win32 プロジェクト	C++
	ASP.NET Web アプリケーション ASP.NET Web サービス アプリケーション SQL Server プロジェクト	C#、VB
	ASP.NET Web サイト コンソール アプリケーション Crystal Reports アプリケーション Visual Studio アドイン Visual Studio 統合パッケージ Web コントロール ライブラリ Windows アプリケーション Windows コントロール ライブラリ	C#、J#、VB
	クラス ライブラリ Windows フォーム アプリケーション Windows フォーム コントロール ライブラリ Windows サービス	C++、C#、J#、VB
ATL - Active Template Library MFC - Microsoft Foundation Class		

メモ： DevPartner Studio カバレッジ分析およびパフォーマンス分析では、Classic ASP とクライアント側 Web スクリプトの両方で VBScript と JScript がサポートされます。

付録 B
コマンド ラインから分析を起動する

この付録では、カバレッジ分析、メモリ分析、パフォーマンス分析、およびパフォーマンス エキスパートに有効な **DPAnalysis.exe** コマンド ライン ツールに関する情報を示します。

DPAnalysis.exe の概要

Visual Studio でプログラムを実行しているときに分析データを収集する以外に、**DPAnalysis.exe** を使用すると、Visual Studio を起動することなくプロファイル情報を収集することができます。**DPAnalysis.exe** では、オプション スイッチと組み合わせるか、または XML 構成ファイルを参照して、アプリケーション データを収集します。

コマンド ラインから DPAnalysis.exe を実行する

DPAnalysis.exe をコマンド ラインから使用することができます。その場合、スイッチまたは XML 構成ファイルを使用して分析セッションを制御できます。以下のコマンド ラインは、アプリケーション **target.exe** のパフォーマンス分析セッションを開始し、セッション ファイル （**dpprf**）を **c:¥output** フォルダに保存します。

```
DPAnalysis.exe /perf /output c:¥output¥target.dpprf /p target.exe
```

DPAnalysis.exe をコマンド ラインから実行する場合は、データ収集を有効にし、1 つのプロセスまたはサービスを実行できます。**DPAnalysis.exe** で複数のプロセスを起動するには、「DPAnalysis.exe で XML 構成ファイルを使用する」 285 ページ）を参照してください。

DPAnalysis.exe では、アンマネージ コードはインストゥルメントされません。アンマネージ アプリケーションのパフォーマンスまたはカバレッジの分析データを収集するには、まずアプリケーションをインストゥルメントする必要があります。カバレッジ分析については「アンマネージ コードのデータを収集する」 121 ページ）を、パフォーマンス分析については「アンマネージ コードのデータを収集する」 194 ページ）を参照してください。

DevPartner Studio の分析コンポーネントをコマンド ラインから実行するには、以下の構文およびスイッチを使用します。

```
DPAnalysis.exe [/Perf|/Cov|/Mem|/Exp] [/E|/D|/R]
[/O outputfilename] [/W workingdirectory] [/PROJ_DIR]
[/H hostmachine] [/NOWAIT] [/NO_UI_MSG] [/N] [/F]
[/NO_QUANTUM /NM_METHOD_GRANULARITY /EXCLUDE_SYSTEM_DLLS
/NM_ALLOW_INLINING /NO_OLEHOOKS
/NM_TRACK_SYSTEM_OBJECTS] {/P|/S} target.exe [target arguments]
```

分析タイプ スイッチ

ランタイム分析タイプを設定します。デフォルトはパフォーマンス分析です。

/Cov[erage]	分析タイプを[カバレッジ分析]に設定します。
/Exp[ert]	分析タイプを[パフォーマンス エキスパート]に設定します。
/Mem[ory]	分析タイプを[メモリ分析]に設定します。
/Perf[ormance]	分析タイプを[パフォーマンス分析]に設定します。

データ収集スイッチ

指定されたターゲットのデータ収集を有効または無効にします。ただし、指定されたターゲットは起動しません。これらのスイッチはオプションです。

/E[nable]	指定されたプロセスまたはサービスのデータ収集を有効にします。
/D[isable]	指定されたプロセスまたはサービスのデータ収集を無効にします。
/R[epeat]	/Dスイッチを使用してプロファイリングを無効にするまで、指定されたプロセスを実行するとプロファイリングが発生します。

その他のスイッチ

これらのスイッチはオプションです。

/O[utput]	セッション ファイルの出力フォルダまたはフォルダとファイル名を指定します。
/W[orkingDir]	ターゲット プロセスまたはサービスの作業フォルダを指定します。
/PROJ_DIR	プレイリストの検出などに使用される、DevPartner Studioプロジェクトのフォルダを指定します。
/H[ost]	ターゲットのホスト コンピュータを指定します。

/NOWAIT	バッチ ファイルで複数のターゲットに/NOWAITを使用すると、**DPAnalysis.exe**は、**process1**の起動後すぐに**process2**を起動します。
	例：
	`DPAnalysis.exe /Exp /NOWAIT /P c:¥temp¥process1.exe`
	`DPAnalysis.exe /Exp /NOWAIT /P c:¥temp¥process2.exe`
	オプションの/NOWAITスイッチを省略すると、**DPAnalysis.exe**は**process1**を終了するまで**process2**の起動を待ちます(デフォルトの動作)。

/NO_UI_MSG	このスイッチをtrueに設定し、UIエラー メッセージを抑制します。デフォルトはfalseです。
/N[ewconsole]	プロセスを独立したコマンド ウィンドウで実行します。 **DPAnalysis.exe**を使用してキーボード入力が必要なコンソール アプリケーションを分析する場合、/NewConsoleスイッチを使用して、入力を受け入れるコンソール ウィンドウを開く必要があります。

/F[orce]	カバレッジ分析またはパフォーマンス分析で、ネイティブ C/C++ インストゥルメンテーションでインストゥルメントされていないアンマネージ コード アプリケーションのプロファイリングを実行します。

二重引用符で囲まれたパスと /O[utput] スイッチ

出力 (/o) スイッチのパラメータとして二重引用符で囲んだパスを指定し、ファイル名を省略する場合は、パスの末尾を次のいずれかの形式とする必要があります。

/o "c:¥test directory"	パスの末尾に二重引用符を付けます。
/o "c:¥test directory¥."	パスの末尾に円記号を付け、その後にピリオドを追加します。
/o "c:¥test directory/"	パスの末尾にスラッシュを付けます。

分析オプション

これらのスイッチはオプションです。

/NO_QUANTUM	他のスレッドの経過時間の除外を無効にします。
/NM_METHOD_GRANULARITY	データ収集の精度をメソッド レベルに設定します。デフォルトは行レベルです (パフォーマンス分析のみ)。
/EXCLUDE_SYSTEM_DLLS	システム DLL に対するデータ収集を除外します (パフォーマンス分析のみ)。
/NM_ALLOW_INLINING	インライン メソッドの実行時インストゥルメンテーションを有効にします。
/NO_OLEHOOKS	COM の収集を無効にします。
/NM_TRACK_SYSTEM_OBJECTS	システム オブジェクトを追跡します (DevPartner メモリ分析の場合のみ)。

ターゲット スイッチ

必須の要素です。1つだけ選択します。ターゲットがプロセスかサービスかを識別します。ターゲットの名前またはパスのあとに指定されたすべての引数がターゲットに対する引数とみなされます。

/P[rocess]	ターゲット プロセスを指定します (後ろにプロセスへの引数が続きます)。
/S[ervice]	ターゲット サービスを指定します (後ろにサービスへの引数が続きます)。
/C[onfig]	構成ファイルとパスを指定します。

DPAnalysis.exe で XML 構成ファイルを使用する

XML 構成ファイルを使用して分析セッションを管理するには、コマンド ラインから /config スイッチと適切に構成された XML 構成ファイルをターゲットとして使用して **DPAnalysis.exe** を実行します。例：

```
DPAnalysis.exe /config c:¥temp¥configuration_file.xml
```

構成ファイルを使用することによって、複数のプロセスまたはサービスのプロファイルと管理が可能となります。複数のプロセスをプロファイルできることは、特に Web アプリケーションの分析を実行する場合に有益です。

DPAnalysis.exe を使用してセッションを開始すると、**DPAnalysis.exe** を起動したシステム上のプロファイル プロセスごとに、セッション コントロール ツールバーが起動されます。ツール バーの適切なインスタンスを使用して、プロセスごとにセッション コントロールのアクションを実行します。

構成ファイルのサンプルを以下に示します。

```
<?xml version="1.0" ?>

<ProductConfiguration xmlns="http://www.microfocus.com/products"

  <RuntimeAnalysis Type="Performance"

   MaximumSessionDuration="1000" NoUIMsg="true" />

  <Targets RunInParallel="true">

    <Process CollectData="true" Spawn="true"

     NoWaitForCompletion="true">

      <AnalysisOptions NO_QUANTUM="1" NM_METHOD_GRANULARITY="1"

       SESSION_DIR="c:¥temp" />

      <Path>ClientApp.exe</Path>

      <Arguments>/arg1 /agr2 /arg3</Arguments>

      <WorkingDirectory>c:¥temp</WorkingDirectory>

      <ExcludeImages>

        <Image>ClassLibrary1.dll</Image>

        <Image>ClassLibrary2.dll</Image>

      </ExcludeImages>

    </Process>

    <Service CollectData="true" Start="true"

     RestartIfRunning="true"

     RestartAtEndOfRun="true">

      <AnalysisOptions NM_METHOD_GRANULARITY="0"

       EXCLUDE_SYSTEM_DLLS="1" />

      <Name>IISadmin</Name>

      <Host>remotemachine</Host>
```

```
    </Service>

  </Targets>

</ProductConfiguration>
```

XML 構成ファイル要素のリファレンス

以下に、XML 構成ファイルの要素について説明します。

RuntimeAnalysis 要素

```
<RuntimeAnalysis Type = "type of analysis"
MaximumSessionDuration = "number of seconds"
NoUIMsg = "allow or suppress UI error messages" />
```

属性

なし

タイプ

必須の要素です。指定できる値は、**Performance**、**Coverage**、**Memory**、**Expert** です。これらの値は、表示されたすべてのターゲットに適用する分析タイプを指定します。

MaximumSessionDuration

オプションです。省略した場合、デフォルトは仮定されません。指定すると、**DPAnalysis.exe** はセッションの実行をこの秒数に制限します。たとえば、`MaximiumSessionDuration="60"` と指定し、サービスのプロファイルを開始すると　そのサービスに対して `RestartAtEndOfRun="true"` と指定します)、60 秒後に、**DPAnalysis.exe** によってサービスが停止され、再起動されます。

NoUIMsg

オプションです。省略した場合、デフォルトの false が使用されます。true に設定すると、**DPAnalysis.exe** は、セッションの期間中に表示される可能性があるすべての UI エラー メッセージを抑制します。セッションが自動実行される場合、または非常に多くのテストを連続して実行する場合は、true に設定すると便利です。

要素情報

発生回数	1 回
上位要素	ProductConfiguration
目次	なし

注釈

分析のタイプと最長のセッション時間を定義します。

例

以下は、`ProductConfiguration` タグのあとに `RuntimeAnalysis` を指定した構造の例です。この例では、`Type` 属性によって分析タイプとしてパフォーマンスを指定し、最大セッション時間を 1000 秒に設定して UI エラー メッセージを抑制しています。

```
<?xml version="1.0" ?>
<ProductConfiguration xmlns="http://www.microfocus.com/products">
<RuntimeAnalysis Type="Performance" MaximumSessionDuration="1000"
NoUIMsg="true" />
```

Targets 要素

```
<Targets RunInParallel=" true or false" >

    ...

  </Targets>
```

属性

RunInParallel: オプションです。**true** または **false** のいずれかを指定します。省略した場合は、デフォルトの **true** になります。複数のターゲットを指定する場合は、それらのターゲットの実行方法を定義します。RunInParallel を **true** に設定すると、**DPAnalysis.exe** によってターゲット プロセスおよびターゲット サービスが次々に起動されます。つまり、同時に複数のターゲットが実行されます（並行）。false に設定すると、**DPAnalysis.exe** によって、プロセスターゲットNが起動および停止してからターゲットN + 1が起動されます。つまり、ターゲットは一度に1つずつ実行されます（逐次）。

要素情報

発生回数	1回
上位要素	RuntimeAnalysis
目次	Process、Service

注釈

必須の要素です。1つまたは複数の<Process>エントリまたは<Service>エントリのブロックを開始します。ターゲットのプロセスとサービスは、構成ファイルに指定されている順に開始されます。

例

以下は、Targets を使用して1つの<Service>要素と2つの<Process>要素の分析を指定する構造の例です。この例では、RunInParallel が **true** に設定されているため、ターゲットは並行して実行されます。

```
<Targets RunInParallel="true">

   <Service CollectData="true" Start="true">

      <AnalysisOptions NM_METHOD_GRANULARITY="0"

         EXCLUDE_SYSTEM_DLLS="1" />

      <Name>ServiceApp</Name>

      <Host>remotemachine</Host>

   </Service>
```

```
   <Process CollectData="true" Spawn="true"
         NoWaitForCompletion="true">
      <AnalysisOptions NO_QUANTUM="1"
         NM_METHOD_GRANULARITY="1"
         SESSION_DIR="c:¥MyDir" />
      <Path>ClientApp.exe</Path>
      <WorkingDirectory>c:¥temp</WorkingDirectory>
   </Process>
   <Process CollectData="true" Spawn="true"
         NoWaitForCompletion="true">
      <AnalysisOptions NO_QUANTUM="1"
         NM_METHOD_GRANULARITY="1"
         SESSION_DIR="c:¥MyDir" />
      <Path>TestApp02.exe</Path>
      <WorkingDirectory>c:¥temp</WorkingDirectory>
   </Process>
</Targets>
```

Process 要素

```
<Process
CollectData = "true or false"
Spawn = "true or false"
NoWaitForCompletion = "true or false"
NewConsole = "true or false"
RepeatInjection = "true or false"
>
...
</Process>
```

属性

/D スイッチを使用してプロファイリングを無効にするまで、指定されたプロセスを実行するとプロファイリングが発生します。

CollectData: オプションです。**true** または **false** のいずれかを指定します。省略した場合は、デフォルトの **true** になります。ターゲット プロセスのプロファイルを有効にするかどうかを指定します。

Spawn: オプションです。**true** または **false** のいずれかを指定します。省略した場合は、デフォルトの **true** になります。**DPAnalysis.exe** が指定されたターゲットを実行するかどうかを指定します。**aspnet_wp.exe** または **w3wp.exe** に対しては、この属性を **true** に設定しないでください。DevPartner では、ASP.NET ワーカー プロセスを直接実行することはできません。ターゲット Web ページを開くことによって、ASP.NET ワーカー プロセスを起動する必要があります。

NoWaitForCompletion: オプションです。**true** または **false** のいずれかを指定します。省略した場合は、デフォルトの **false** になります。デフォルトでは、プロセスが完了するまで待機します。**true** に設定した場合、**DPAnalysis.exe** はターゲットが起動するまで待機します。この場合、**DPAnalysis.exe** はリモート コンピュータ上のプロセスが完了するまで待機するわけではありません（**Host** 要素を使用します）。RuntimeAnalysis 要素に MaximumSessionDuration 属性を指定すると、NoWaitForCompletion は無効となります。

NewConsole: オプションです。**true** または **false** のいずれかを指定します。省略した場合は、デフォルトの **false** になります。true に設定すると、ターゲットは独立したコンソール ウィンドウで実行されます。デフォルトでは、**DPAnalysis.exe** コマンド ラインを入力したコンソールと同じコンソールが使用されます。**DPAnalysis.exe** を使用して、キーボード入力が必要なコンソール アプリケーションを分析する場合、/NewConsole スイッチを使用して、入力を受け入れるコンソール ウィンドウを開く必要があります。

RepeatInjection: オプションです。**true** または **false** を指定します。省略した場合は、デフォルトの **false** になります。明示的に **false** を指定するまで、**DPAnalysis.exe** によって、ターゲットを実行するたびにプロファイルされます。

要素情報

発生回数	1 回以上
上位要素	Target
目次	AnalysisOptions、Path, Arguments、WorkingDirectory、ExcludeImages

注釈

ターゲットの実行可能ファイルを指定します。

例

以下は、Process を使用した構造例です。**AnalysisOptions**、**Path**、**Arguments**、**WorkingDirectory** のタグが使用されています。

```
<Targets RunInParallel="true">

<Process CollectData="true" Spawn="true"

 NoWaitForCompletion="true" NewConsole="true">

  <AnalysisOptions NO_QUANTUM=Åh1Åh NM_METHOD_GRANULARITY=Åh1Åh

   SESSION_DIR="c:¥MyDir" />
```

```
  <Path>ClientApp.exe</Path>

  <Arguments>/arg1 /agr2 /arg3</Arguments>

  <WorkingDirectory>c:¥temp</WorkingDirectory>

</Process>

</Targets>
```

AnalysisOptions 要素

AnalysisOptions で使用できる属性は、実行する分析セッションのタイプによって異なります。このトピックの末尾にある表を参照してください。**DPAnalysis.exe** では、分析タイプに適合しない属性は無視されます。

```
<AnalysisOptions

SESSION_DIR = "c:¥MyDir"

SESSION_FILENAME = "myfile.dpcov"

NM_METHOD_GRANULARITY = "1"

EXCLUDE_SYSTEM_DLLS = "1"

NM_ALLOW_INLINING = "1"

NO_OLEHOOKS = "1"

NM_TRACK_SYSTEM_OBJECTS = "1"

NO_QUANTUM = Åg1Åh

FORCE_PROFILING = "1"

/>
```

属性

SESSION_DIR: オプションです。カバレッジ分析、メモリ分析、パフォーマンス分析、およびパフォーマンス エキスパートで使用します。プロファイルされたターゲットが生成するセッション ファイルの保存先フォルダを指定します。この属性を指定しない場合、生成されたセッション ファイルはユーザーの **My Documents** または **Documents** フォルダに保存されます。SESSION_DIR と SESSION_FILENAME のいずれも指定せずに、**DPAnalysis.exe** を実行すると、セッションの終了時に保存先の指定を求めるメッセージが表示されます。

SESSION_FILENAME: オプションです。カバレッジ分析、メモリ分析、パフォーマンス分析、およびパフォーマンス エキスパートで使用します。このターゲットについて生成されるセッション ファイルのセッション名を指定します。この属性を指定せずに **DPAnalysis.exe** を実行すると、ターゲットのイメージ名に番号が付いた固有の名前（例 : **iexplore1.dpprf**）が作成されます。名前だけを指定し、フォルダを指定しなかった場合、ファイルはユーザーの **My Documents** フォルダに保存されます。SESSION_FILENAME と SESSION_DIR のいずれも指定せずに、**DPAnalysis.exe** を実行すると、セッションの終了時に保存先の指定を求めるメッセージが表示されます。

NM_METHOD_GRANULARITY: オプションです。パフォーマンス分析で、データ収集の精度をメソッド レベルに設定するために使用します（デフォルトは行レベルです）。この属性を設定するには、**1**を指定します。省略すると、この属性は無効となります。

EXCLUDE_SYSTEM_DLLS: オプションです。パフォーマンス分析でシステム イメージを除外する場合に使用します。この属性を設定するには、1を指定します。省略すると、この属性は無効となります。

NM_ALLOW_INLINING: オプションです。カバレッジ分析およびパフォーマンス分析で使用し、分析の詳細度を指定します。インライン メソッドの実行時インストゥルメンテーションを有効にします。この属性は**Instrument Inline Functions**プロパティと同じ働きをします。インライン関数をインストゥルメントする場合は、1を指定します。省略すると、この属性は無効となります。

NO_OLEHOOKS: オプションです。パフォーマンス分析で使用し、システム オブジェクトの追跡を有効にします。この属性を設定するには、1を指定します。省略すると、この属性は無効となります。

NM_TRACK_SYSTEM_OBJECTS: オプションです。メモリ分析で使用し、割り当てられたオブジェクトの追跡時にシステムまたはサードパーティ オブジェクトの割り当てを無視します。この属性を設定するには、1を指定します。省略すると、この属性は無効となります。デフォルトの状態　無効）では、システム リソースまたはその他のプロファイルされていないリソースがアプリケーションによって使用されたときに行われたメモリの割り当てを確認できます。

NO_QUANTUM: オプションです。パフォーマンス分析およびパフォーマンス エキスパート分析で使用し、他のアプリケーションのスレッドの経過時間を除外します。この属性を設定するには、1を指定します。省略すると、この属性は無効となります。

FORCE_PROFILING: オプションです。カバレッジ分析とパフォーマンス分析で使用され、マネージ コードまたはネイティブC/C++インストゥルメンテーションなしで作成されたアプリケーションのプロファイリングを実行します。この属性を設定するには、1を指定します。省略すると、この属性は無効となります。

属性	カバレッジ	メモリ	パフォーマンス	パフォーマンス エキスパート
NM_METHOD_GRANULARITY			X	
EXCLUDE_SYSTEM_DLLS			X	
NM_ALLOW_INLINING	X		X	
NO_OLEHOOKS			X	
NM_TRACK_SYSTEM_OBJECTS		X		
NO_QUANTUM			X	X
FORCE_PROFILING	X		X	

要素情報

発生回数	1つのプロセスまたはサービスにつき1回または発生なし
上位要素	Process、Service
目次	なし

注釈

オプションです。特定のターゲット プロセスまたはターゲット サービスにランタイム属性を定義します。カバレッジ分析、メモリ分析、パフォーマンス分析の各プロパティに対応する属性には、Visual Studioのプロパティ ウィンドウからアクセスできます。

例

以下は、Service内に**AnalysisOptions** を指定した構造の例です。

```
<Service CollectData="true">
  <AnalysisOptions NM_METHOD_GRANULARITY="1"
   EXCLUDE_SYSTEM_DLLS="1" NM_ALLOW_INLINING="1"
   NO_OLEHOOKS="1">
</Service>
```

Path要素

```
<Path> c:¥MyDir¥target.exe </Path>
```

属性

なし

要素情報

発生回数	1回
上位要素	Process
目次	実行可能ファイルのパス

注釈

必須の要素です。実行可能ファイルの完全修飾パスまたは相対パスを指定します。実行可能ファイルが現在のフォルダにある場合は、パスを指定せずに実行可能ファイル名を指定できます。

例

以下は、Process要素内にPathを指定した構造の例です。

```
<Process CollectData="true">
  <Path>ClientApp.exe</Path>
</Process>
```

Arguments 要素

```
<Arguments>/arg1 /arg2 /arg3</Arguments>
```

属性

なし

要素情報

発生回数	1 つのプロセスまたはサービスにつき 0 回または 1 回
上位要素	Process、Service
目次	なし

注釈

オプションです。省略した場合のデフォルトはありません。ターゲットには、引数として `process` または `service` のいずれかを渡します。

例

以下は、`Process` 要素内に `Arguments` を指定した構造の例です。

```
<Process CollectData="true">
  <Arguments>/arg1 /agr2 /arg3</Arguments>
</Process>
```

WorkingDirectory 要素

```
<WorkingDirectory> c:¥MyWorkingDir </WorkingDirectory>
```

属性

なし

要素情報

発生回数	1 つの Process 要素または Service 要素につき 1 回
上位要素	Process、Service
目次	ターゲット フォルダのパス

注釈

オプションです。省略した場合のデフォルトはありません。ターゲット プロセスまたはターゲット サービスの作業フォルダを設定します。

例

以下は、親 `Process` 要素内にネストされた `WorkingDirectory` を指定した構造の例です。

```
<Process CollectData="true">
  <WorkingDirectory>c:¥temp</WorkingDirectory>
```

```
</Process>
```

ExcludeImages 要素

```
<ExcludeImages>

<Image>ClassLibrary1.dll</Image>

<Image>ClassLibrary2.dll</Image>

</ExcludeImages>
```

属性

なし

要素情報

発生回数	1 つのプロセスまたはサービスにつき 0 回または 1 回
上位要素	Process、Service
目次	イメージ

注釈

オプションです。省略した場合のデフォルトはありません。ターゲット プロセスまたはターゲット サービスによってロードされた場合にプロファイルされない、1 つ以上のイメージ 数は無制限）のリストを表示します。

例

以下は、`Process` 要素内に `ExcludeImages` を指定した構造の例です。`ExcludeImages` 内に `Image` 要素が含まれていることに注意してください。

```
<Process CollectData="true">

  <ExcludeImages>

    <Image>ClassLibrary1.dll</Image>

    <Image>ClassLibrary2.dll</Image>

  </ExcludeImages>

</Process>
```

Service 要素

```
<Service>

CollectData = "true or false"

Start = "true or false"

RestartIfRunning = "true or false"

RestartAtEndOfRun = "true or false"
```

```
RepeatInjection = "true or false"

>

...

</Service>
```

属性

CollectData: オプションです。**true** または **false** を指定します。省略した場合は、デフォルトの **true** になります。ターゲット プロセスのプロファイルを有効にするかどうかを指定します。

Start: オプションです。**true** または **false** のいずれかを指定します。省略した場合は、デフォルトの **true** になります。**DPAnalysis.exe** が指定されたターゲットを起動するかどうかを指定します。**false** に設定すると、そのターゲットのプロファイルは有効になりますが、ターゲットは起動されません。プロファイルは、サービスが次に　何らかの方法で）起動されたときに開始されます。

RestartIfRunning: オプションです。**true** または **false** のいずれかを指定します。省略した場合は、デフォルトの **false** になります。RestartIfRunning を **true** に設定すると、指定したサービスがホスト コンピュータ上で実行されている場合、そのサービスが再起動されます。

RestartAtEndOfRun: オプションです。**true** または **false** のいずれかを指定します。省略した場合は、デフォルトの **false** になります。**true** を指定すると、サービスが実行完了時に再起動されます　セッション ファイルが生成されます）。

RepeatInjection: オプションです。**true** または **false** のいずれかを指定します。省略した場合は、デフォルトの **false** になります。明示的に **false** を指定するまで、**DPAnalysis.exe** によって、ターゲットを実行するたびにプロファイルされます。

要素情報

発生回数	構成ファイルには、少なくとも1つのProcess要素またはService要素を指定する必要があります。
上位要素	Targets
目次	AnalysisOptions、Path、Arguments、Working Directory、ExcludeImages、Name、Host

注釈

ターゲット サービスを指定します。

例

以下は、Targets要素内にServiceを指定した構造の例です。

```
<Targets RunInParallel="true">
  <Service CollectData="true" Start="true"
   RestartIfRunning="true" RestartAtEndOfRun="true">
   <Name>ServiceApp</Name>
  </Service>
```

```
</Targets>
```

Name 要素

```
<Name>MyServiceName</Name>
```

属性

なし

要素情報

発生回数	1 回
上位要素	Service
目次	サービス名

注釈

必須の要素です。サービス コントロール マネージャに登録されたサービスの名前を指定します。これは、NET START コマンドを使用するときと同じ名前です。

例

以下は、Service 内に Name を指定した構造の例です。

```
<Service CollectData="true">
  <Name>ServiceApp</Name>
</Service>
```

Host 要素

```
<Host>hostmachine</Host>
```

属性

なし

要素情報

発生回数	1 つのプロセスまたはサービスにつき 0 回または 1 回
上位要素	Process、Service
目次	ホスト コンピュータの名前

注釈

オプションです。省略した場合のデフォルトはありません。ターゲット プロセスまたはターゲット サービスのホスト コンピュータを設定します。

例

以下は、Service 内に Host を指定した構造の例です。必須の Name 要素が含まれていることに注意してください。

```
<Service CollectData="true">
  <Name>ServiceApp</Name>
  <Host>remotemachine</Host>
</Service>
```

XML構成ファイルによるWebアプリケーションのプロファイル

Webプロファイルの対象となる主なプロセスとして、ブラウザ、Webサーバー、ASP.NETワーカー プロセスがあります。これら3つのエントリは、いずれも1つの構成ファイルに指定することができます。ブラウザとASP.NETワーカー プロセスは<Process>要素内に指定し、Webサーバーは<Service>要素内に指定します。このとき、サービス名は<Name>要素に指定します。IISの場合、サービス名はiisadminとなります。

例：

```
<?xml version="1.0" ?>
<ProductConfiguration xmlns="http://www.microfocus.com/products">
  <RuntimeAnalysis Type="Expert"/>
  <Targets>
     <Process CollectData="true">
       <AnalysisOptions
         SESSION_DIR="z:¥SessionFiles"/>
       <Path>aspnet_wp.exe</Path>
       <Host>remotemachine</Host>
     </Process>
     <Service CollectData="true" Start="true"
       RestartIfRunning="true"
       RestartAtEndOfRun="true">
       <AnalysisOptions
         SESSION_DIR="z:¥SessionFiles"/>
       <Name>iisadmin</Name>
       <Host>remotemachine</Host>
     </Service>
     <Process CollectData="true" Spawn="true">
       <AnalysisOptions
```

```
        SESSION_DIR="c:¥SessionFiles"/>
      <Path>iexplore.exe</Path>
      <Arguments>
        http://remotemachine/WebApplication/
        StartPage.aspx
      </Arguments>
    </Process>
  </Targets>
</ProductConfiguration>
```

上記の構成ファイルには以下の操作が指定されています。

- remotemachine において ASP.NET ワーカー プロセスのデータ収集を有効にします。
- リモート コンピュータにおいて **inetinfo.exe**（isadmin）のデータ収集を有効にし、リモート マシンを再起動してプロファイルを開始します。
- ローカル コンピュータ上でローカル ブラウザを開き、リモート コンピュータ上の Web ページに接続します。これによって、リモート コンピュータ上で **aspnet_wp.exe** が実行され、プロファイリングが開始します。

ローカル コンピュータのブラウザを閉じると、リモート コンピュータ上で IIS が再起動され（aspnet_wp が終了します）、セッション ファイルがそれぞれの保存先フォルダに保存されます。必要に応じて、リモート コンピュータ上の既存のマップされたドライブを使用し、プロファイルが開始されたマシンにセッション ファイルを保存することもできます。上記の例では、<Process> 要素および <Service> 要素に **z:¥** ドライブを指定しています。

サンプル構成ファイル

DevPartner Studio では、読み取り専用サンプル構成ファイルが提供されます。これらをカスタム構成ファイルのモデルとして使用します。

Sample.Process.Config.xml
Sample.Service.Config.xml
Sample.WebApp.Config.xml
Sample.DCOM.Config.xml
Sample.ClassicASP_IIS_High_Isolation.Config.xml
Sample.ClassicASP_IIS_Low_Isolation.Config.xml
Sample.Multi_Process.Config.xml

デフォルトのインストールをした場合、これらのファイルは以下のフォルダに入っています。

<install drive>:¥Program Files¥Micro Focus¥DevPartner Studio¥Analysis¥SampleConfigs¥

64ビット バージョンのWindowsでは、このファイルは**¥Program Files (x86)¥Micro Focus¥DevPartner Studio¥Analysis¥SampleConfigs¥**にインストールされます。

DPAnalysis.exe では、アンマネージ コードはインストゥルメントされません。アンマネージ アプリケーションのパフォーマンスまたはカバレッジの分析データを収集するには、まずアプリケーションをインストゥルメントする必要があります。カバレッジ分析については「アンマネージ コードのデータを収集する」（121ページ）を、パフォーマンス分析については「アンマネージ コードのデータを収集する」（194ページ）を参照してください。

リモート コンピュータの分析データを収集する

DPAnalysis.exe を使用してリモート コンピュータで実行するアプリケーションのデータを収集する場合、以下の点に注意してください。

- **DPAnalysis.exe** を使用してリモート システムでアプリケーションを実行するときは、コマンド ラインかXML構成ファイルを使って、セッション ファイルを保存するためのフォルダとファイル名を指定します。
- 書き込みアクセス権のあるフォルダであれば、任意のフォルダを指定できます。これには、プロファイルを開始したローカル（クライアント）コンピュータにマップされているフォルダも含まれます。
- フォルダとファイル名を指定しないと、リモート コンピュータ上に [プロジェクトにファイルを追加] ダイアログが表示されます。このダイアログ ボックスで操作するには、対象コンピュータへの物理的なアクセス、ターミナル サービス接続またはリモート デスクトップ接続が必要です。[プロジェクトにファイルを追加] ダイアログのデフォルト保存場所は、アクティブなユーザー アカウントの **My Documents** または **Documents** フォルダです。

付録 C

分析セッションの制御

この付録では、DevPartner カバレッジ分析、メモリ分析、パフォーマンス分析、およびパフォーマンス エキスパートで使用できるセッション コントロール ファイルとセッション コントロール API に関する情報を示します。

セッション コントロール ファイルの概要

[セッション コントロール ファイル] オプションを使用すると、アプリケーションを実行したままで、DevPartner で収集したデータをコントロールする一連のルールやアクションを作成できます。DevPartner では、これらのルールやアクションは、アプリケーションのソリューション フォルダにあるセッション コントロール ファイル（**`$essionControl.txt`**）に格納されます。

セッション コントロール ファイルには選択したメソッドに対するデータ収集アクションが含まれており、ユーザーは以下の操作を実行できます。

- メソッドの開始時または終了時のデータ収集アクションを指定する。
- セッション間でセッション コントロールファイルを維持する。
- カバレッジ分析、メモリ分析、パフォーマンス分析、およびパフォーマンス エキスパートのセッションに影響するセッション コントロール ファイルのエントリを作成する。

Visual Studio 内でセッション コントロール ファイルを作成する

サポートされている Visual Studio のリリースから、下記で説明されているように、**[DevPartner]** > **[** オプション **]** メニューからファイルを作成できます。テキスト エディタを使用したセッション コントロール ファイルの作成については、「セッション コントロール API を使用する」（302 ページ）を参照してください。

セッション コントロール ファイルを作成するには

1. **[DevPartner]** > **[** オプション **]** を選択します。
2. **[** オプション **]** で、**[DevPartner]** > **[** 分析 **]** > **[** セッション コントロール ファイル **]** を選択します。はじめてセッション コントロール ファイルのオプションを設定するときは、空のセッション コントロール ファイル（**`$essionControl.txt`**）にアクセスします。
3. **[** 追加 **]** をクリックします。
4. [モジュール] テキスト ボックスで、選択または参照を行って、データ収集を行うモジュールを指定します。モジュールのインストゥルメンテーション ステータスが表示されます。

   マネージ コードのすべてのモジュールには、「インストゥルメントされていない」ということを示すステータスが表示されます。「インストゥルメントされている」ということを

示すステータスが表示されるのは、ネイティブ C/C++ インストゥルメンテーションでビルドされているアンマネージ　ネイティブ）C/C++ モジュールだけです。

5 [メソッド リスト] から、データを記録するメソッドを選択します。

.NET モジュール（**.netmodule**）からメソッドを選択する場合、[メソッド リスト] のメソッドは **namespace.classname.method** の形式で表示されます。DevPartner Studio は、セッション コントロール ファイルで最大 512 文字の修飾メソッド名をサポートしています。512 文字より長い名前は無視され、そのメソッドにセッション コントロール アクションは発生しません。

6 セッション コントロール アクションの開始時期を選択します。

7 以下のいずれかから適用するアクションを選択します。

- ◇ 記録の停止　最後のスナップショットの取得）
- ◇ スナップショットの取得
- ◇ 記録したすべてのデータのクリア
- ◇ 追跡の開始　メモリ リーク分析）
- ◇ 追跡の停止　メモリ リーク分析）
- ◇ GC の実行　メモリ分析）

8 **[OK]** をクリックします。

9 含めるメソッドをすべて選択するまで、手順 3 ～ 8 を繰り返します。

10 **[OK]** をクリックし、セッション コントロール ファイルを保存します。

Visual Studio でソリューションを開いている場合は、ソリューション フォルダに、セッション コントロール ファイルが保存されます。

メモ： **SessionControl.txt** ファイルは、プロファイリング中の実行可能なアプリケーションを含むソリューション フォルダで検索されます。ソリューション フォルダにこのファイルが見つからない場合、アプリケーションの実行可能ファイルがビルドされた出力フォルダが検索されます。**SessionControl.txt** ファイルをこれ以外の場所に置くと、セッション コントロール コマンドが認識されなくなります。

セッション コントロール ファイルのエントリは、カバレッジ分析、メモリ分析、パフォーマンス分析、およびパフォーマンス エキスパートの分析セッションに影響を与えます。

セッション コントロール API を使用する

ソース コード内の任意の場所からセッション コントロール API を呼び出して、いずれかの Visual Studio アプリケーションに対するデータ収集を制御します。セッション コントロール テキスト ファイルを使用すると、メソッドの開始時または終了時にのみ DevPartner セッション コントロールのアクションを許可できます。

DevPartner セッション コントロール **API** 関数

Clear	この時点までに収集されたデータをクリアします。データ収集は継続されます。データがクリアできた場合は`NMStatusSuccess`を、クリアできなかった場合は`NMStatusFailure`を返します。
Snap	記録されたデータのスナップショットを取ります。スナップショットが保存された場合は`NMStatusSuccess`を、保存されなかった場合は`NMStatusFailure`を返します。
SaveNow	収集データのスナップショットを取り、データ収集を停止します。指定されている場合は、メソッドのファイル名を取得します。`NMStatusSuccess`または`NMStatusFailure`を返します。
StartTrackingForLeakAnalysis	割り当てられたオブジェクトの追跡を開始します。（メモリ リーク分析のみ）
StopTrackingForLeakAnalysis	割り当てられたオブジェクトの追跡を停止します。（メモリ リーク分析のみ）
RunGC	システムのガベージ コレクタを実行します（メモリ分析）。

メモ： **`SaveNow`**は、コードでの最後の API 関数コールになるようにしてください。このコールによってプロセスのデータ収集が停止されるため、後続のすべての API コールが無視されます。

`Snap Session Control`API コールによって、メモリ分析セッション中に一時オブジェクト セッション ファイルが作成されます。最後のガベージ コレクション後に割り当てられたオブジェクトのメモリ サイズ データをキャプチャするには、`Snap` API コールの前に`RunGC` API コールを挿入します。

API の場所

以下のファイルには、それぞれ、マネージ コードとアンマネージ コードで使用するセッション コントロール API 関数が含まれています。これらのファイルはすべて、DevPartner Studio の **`¥Analysis`** フォルダにインストールされています。

マネージ コード Visual Studio アプリケーション	**`DevPartner.Analysis.SessionControl.dll`**
アンマネージ（ネイティブ）コードの C/C++ または C++ アプリケーション	**`NmTxApi.h`**

マネージ アプリケーションでセッション コントロール **API** を使用する方法

マネージ コード Visual Studio アプリケーションでセッション コントロール API 関数を使用するには、プロジェクト内の **`DevPartner.Analysis.SessionControl.dll`** を参照する必要があります。

それによって、DevPartner 名前空間のセッション コントロール API にアクセスできるようになります。以下の構文を使用して、コード内の該当するポイントで API へのコールを挿入できます。

`Clear`

DevPartner.`Analysis.SessionControl.Clear()`

`Snap`

DevPartner.`Analysis.SessionControl.Snap(<セッション ファイル名>.dpxxx)`

dpxxx は分析タイプの拡張子です (dpcov、dpmem、dpprf、または dppxp)。

`SaveNow`

DevPartner.`Analysis.SessionControl.SaveNow(<セッション ファイル名>.dpxxx)`

dpxxx は分析タイプの拡張子です (dpcov、dpmem、dpprf、または dppxp)。

`StartTrackingForLeakAnalysis`

DevPartner.`Analysis.SessionControl.StartTrackingForLeakAnalysis()`

`StopTrackingForLeakAnalysis`

DevPartner.`Analysis.SessionControl.StopTrackingForLeakAnalysis()`

`RunGC`

DevPartner.`Analysis.SessionControl.RunGC()`

`Snap` と **`SaveNow`** API 関数には、以下の入力を指定できます。

- ファイル名
- フォルダへの完全修飾パス (最後に「¥」円記号を付けます)
- 完全修飾パス (ファイル名を含めます)
- 指定なし (Null)

DevPartner でファイル情報とパス情報がどのように扱われるかについては、「セッション コントロール API を使用してファイルを保存する」 (306 ページ)を参照してください。

セキュリティ例外が発生した場合

セッション コントロール API を使用してマネージ コード アプリケーションをプロファイルしようとしたときにセキュリティ例外が発生した場合、それはセキュリティ ポリシーによって、通常のコードのランタイム時 DevPartner インストゥルメンテーションが禁止されていることを意味します。これを修復するには、安全なプロファイリングを有効にする必要があります。

以下のグローバル環境変数を設定します。

`NM_NO_FAST_INSTR=1`

アプリケーションをもう一度プロファイルします。

デフォルトで、アセンブリをプロファイルするには、**`SkipVerification`** 権限が必要になります。コードが実行されるポリシーの権限セットからこの権限を削除したり、この権限を無効にするような命令型のセキュリティ宣言をアセンブリに追加した場合、アセンブリをプロファイルできません。前述のソリューションを使用すると、この問題を回避できますが、パ

フォーマンスが若干低下します。前述のソリューションを導入しない場合は、DevPartner Studio を使用してこのようなアセンブリのプロファイリングを有効にすることもできます。有効にするには、.NET Framework Configuration ツールの MMC スナップインを使用してアセンブリのポリシーを変更するか、アセンブリ内の命令的セキュリティ宣言を一時的に削除します。

Visual Studio セキュリティ ポリシーの詳細については、Visual Studio オンライン ヘルプの『.NET Framework Developers Guide』を参照してください。

アンマネージ アプリケーションでセッション コントロール API を使用する

セッション コントロール API を使用して、カバレッジ分析を制御したりアンマネージ C/C++ の分析セッションを実行したりできます。

アンマネージ（ネイティブ）C/C++ プロジェクト

ネイティブ C/C++ アプリケーションのカバレッジ データを収集するには、ソリューション（または ネイティブ C/C++ プロジェクト）をネイティブ C/C++ インストゥルメンテーションでリビルドする必要があります。

ネイティブ C/C++ でセッション コントロール API 関数を使用するには

1. **NmTxApi.h** をセッション コントロール API のコールを追加するファイルに含めます。リンク ライブラリの一覧に **TxInterf.lib** を追加します。
2. コード内の該当するポイントにセッション コントロール API 関数へのコールを挿入します。「アンマネージ プロジェクトでのセッション コントロール API の構文」（305 ページ）を参照してください。
3. ネイティブ C/C++ インストゥルメンテーションでソリューションまたは 1 つのネイティブ C/C++ プロジェクトをリビルドします。

アンマネージ（ネイティブ）C++ プロジェクト

ネイティブ C++ アプリケーションのカバレッジ データを収集する前に、Visual Studio のインストゥルメンテーションでプロジェクトをリビルドする必要があります。

アンマネージ プロジェクトでのセッション コントロール API の構文

アンマネージ プロジェクトのセッション コントロール API の構文については、以下の情報を参照してください。

`Clear`	`Clear()`
`Snap`	`Snap(" <セッション ファイル名>.dpxxx" )` `dpxxx` は分析タイプの拡張子です **`.dpcov`** または **`.dpprf`**
`SaveNow`	`SaveNow(" <セッション ファイル名>.dpxxx" )` `dpxxx` は分析タイプの拡張子です **`.dpcov`** または **`.dpprf`**

セッション コントロール API を使用してファイルを保存する

セッション コントロール API を使用してデータのスナップショットを取ったり最後のセッション ファイルを作成したりする場合、API コールでセッション ファイル名とフォルダを指定できます。

セッション コントロール API コールで指定したファイル名とフォルダは、その他の方法（コマンド ラインの `/output` スイッチ、XML 構成ファイルの `SESSION_FILENAME` 属性や `SESSION_DIR` 属性など）で指定したファイル名とフォルダよりも優先されます。

- セッション コントロールの `Snap` または `SaveNow` API コールでファイル名とフォルダを指定すると、それに応じてファイルが保存されます。このフォルダに同じ名前のファイルが存在している場合は上書きされます。
- フォルダだけを指定すると、セッションは、ターゲット プロセスの名前に基づいて一意のファイル名で保存されます。既存のファイルが上書きされることのないように、ファイル名には連続した番号が自動的に追加されます。
- ファイル名だけを指定すると、セッションは指定の名前で保存されます。保存先は、アプリケーションの起動方法に応じてターゲット フォルダが決定されます。たとえば、アプリケーションを Visual Studio から起動した場合、ファイルは現在のプロジェクトのソリューション フォルダに保存されます。**`DPAnalysis.exe`** を使用してコマンド ラインからアプリケーションを起動すると、ファイルはアクティブなユーザー アカウントの **`Documents`**（Windows XP の場合は **`My Documents`**）フォルダに保存されます。このフォルダに同じ名前のファイルが存在している場合は上書きされます。
- ファイル名とフォルダのいずれも指定しないと、ファイルは一意のファイル名で保存されます。保存先は、前述したように、アプリケーションの起動方法に応じてターゲット フォルダが決定されます。ファイルが上書きされることのないように、ファイル名には連続した番号が自動的に追加されます。

メモ： 出力フォルダがないプロジェクトの場合は（たとえば、Visual Studio 2005 Web サイト プロジェクトなど）、ファイルはプロジェクト フォルダに保存されます。

パスの指定時には、以下の点に注意してください。

- ファイル処理するプロセスの現在の作業フォルダに基づいてパス情報が評価されます。アプリケーションが実行されると、作業フォルダが変更される場合もあります。
- セッション ファイルの場所の特定を容易にするには、フル パスを指定することをお勧めします。
- ローカル コンピュータ上では、フル パスが存在しない場合はフル パスが作成されます。リモート コンピュータ上でデータを収集している場合は、既存のフォルダを指定する必要があります。
- パスを指定してファイル名を指定しない場合は、パスの最後に「¥」円記号を付けます。DevPartner では、最後の円記号のあとに続く文字がファイル名として扱われます。
- パスに無効なデータが含まれている場合は、フォルダは指定されていないとみなされてファイルが保存されます。

相互関係と優先順位

セッション コントロール API コールに指定したファイル名とフォルダは、その他の方法で指定したファイル名とフォルダよりも優先されます。

推奨：ファイル名とフォルダの設定は、API コールとコマンド ラインの　両方ではなく）いずれかで行うようにしてください。

例:セッション コントロール API でフォルダ（またはファイル名）だけを指定し、**DPAnalysis.exe** コマンド ラインまたは XML 構成ファイルでファイル名（またはフォルダ）を指定すると、これらの情報が組み合わされてファイルが保存されます。この例で、DevPartner で一意なファイル名が作成されることを意図していた場合、期待どおりには処理されません。

推奨：ファイル管理を簡素化するため、API コールのときにスナップショットと最後のセッションを両方とも指定します。

例：セッション コントロール API で最後のスナップショット　**SaveNow**）を指定しないと、最後のスナップショットはプロセスの終了時に取られることになります。**DPAnalysis.exe** を使ってアプリケーションを起動した場合、コマンド ラインまたは XML 構成ファイルで指定されたオプションに応じて、最後のセッション ファイルが保存されます。Visual Studio からアプリケーションを起動すると、保存されていないセッション データが表示されます。

付録 D
分析データを XML にエクスポートする

この付録では、DevPartner カバレッジ分析、パフォーマンス分析、およびパフォーマンス エキスパートで使用できる、分析データの XML へのエクスポートに関する情報を示します。

DevPartner データのエクスポートの概要

DevPartner を使用すると、カバレッジ分析、パフォーマンス分析、およびパフォーマンス エキスパート データから XML へ保存済みのセッション ファイルをエクスポートできます。XML データは、Visual Studio またはコマンド ラインからエクスポートできます。

ユーザー独自のソフトウェアまたはサードパーティ製のソフトウェアを使用して、エクスポートした XML データを分析することができます。例：

- 単体テスト、機能テスト、回帰テストを行う開発ビルド サーバーまたは QA サーバーで **[**DevPartner データのエクスポート **]** を使用します。エクスポートした XML データを分析して日々の進捗を監視する。
- **[**DevPartner データのエクスポート **]** を使用して、長期的な分析用のデータを収集します。データベースやデータ ウェアハウスに XML データを蓄積して以下の目的に使用することができます。
  - データと、開発方法、QA 方法、ツール、およびインフラストラクチャの統合
  - データに対するカスタム分析の実施
  - 履歴または監査目的のデータのアーカイブ

分析データを XML にエクスポートする

Visual Studio 内から、保存した DevPartner カバレッジ分析 （**.dpcov**）、カバレッジ分析マージ （**.dpmrg**）、パフォーマンス分析 （**.dpprf**）、およびパフォーマンス エキスパート （**.dppxp**）のデータを XML 形式にエクスポートできます。

Visual Studio で XML にエクスポートするには

1 保存したセッション ファイル （上記を参照）を開きます。

2 **[** ファイル **]** > **[**DevPartner データのエクスポート **]** を選択します。

DevPartner はデフォルトで、保存済みのセッション ファイル名に **.xml** 拡張子を付与して、セッション ファイルが保存されているフォルダ内に XML ファイルを保存します（**Chart1.dpcov.xml** など）。

ファイル DevPartner**PerformanceCoverage***xx***.xsd** では、カバレッジ分析データとパフォーマンス分析データをエクスポートするために DevPartner が使用する XML スキーマを定義します。ファイル DevPartner**PerformanceExpert***xx***.xsd** では、パフォーマンス エキスパート データを

エクスポートするために DevPartner が使用する XML スキーマを定義します。どちらのスキーマも **C:¥Program Files¥Micro Focus¥DevPartner Studio¥Analysis** に配置されます。

64 ビット バージョンの Windows では、DevPartner Studio は以下の場所にインストールされます。**¥Program Files (x86)¥Micro Focus¥DevPartner Studio¥Analysis¥**

コマンド ラインから分析データを XML にエクスポートする

Visual Studio を使用する代わりに、コマンド ラインから DevPartner**.Analysis.DataExport.exe** を使用してカバレッジ分析、カバレッジ分析マージ、パフォーマンス分析、パフォーマンス エキスパートのデータを XML にエクスポートできます。

このユーティリティは **C:¥Program Files¥Micro Focus¥DevPartner Studio¥Analysis** に配置されます。

64 ビット バージョンの Windows では、DevPartner Studio は以下の場所にインストールされます。**¥Program Files (x86)¥Micro Focus¥DevPartner Studio¥Analysis**

説明

```
DevPartner.Analysis.DataExport.exe [ セッション ファイル名 | ディレクトリのパス ] { オプション }
```

オプション

/out[put]=<String>	エクスポートする XML ファイルのローカルまたはリモートの出力フォルダを指定します。フォルダが存在しない場合、フォルダが作成されます。
/r[ecurse]	DevPartner セッション ファイルのサブフォルダを検索します。
/f[ilename]=<String>	XML 出力ファイルの名前を指定します。指定した名前に .xml が付加されます。
/showAll	パフォーマンス分析またはカバレッジ分析のセッション ファイルで利用可能な、すべてのパフォーマンス分析およびカバレッジ分析のセッション ファイル データが表示されます。 たとえば、このオプションを指定してパフォーマンス セッション ファイルをエクスポートすると、結果の XML ファイルにはパフォーマンスとカバレッジの両方のデータ フィールドが含まれます。 このオプションは、パフォーマンス エキスパート ファイルには利用できません。
/w[ait]	ユーザーの入力を待機して、コンソール ウィンドウを閉じます。
/nologo	ロゴや著作権情報を表示しません。
/help または /?	コンソール ウィンドウにヘルプを表示します。

/summary	パフォーマンス エキスパートのサマリ データをエクスポートします。CPU リソースを最も多く使用したコール パスとメソッドをエクスポートします。デフォルトでは、最大で上から 10 番めまでのコール パスとメソッドがエクスポートされます。 最大値を上書きするには、**/maxpaths** オプションと /maxmethods オプションを使用します。 デフォルトでサマリ データが表示されます。
/method	パフォーマンス エキスパート メソッド データをエクスポートします。
/calltree	パフォーマンス エキスパートのコール ツリー データをエクスポートします。
/maxpaths=<integer>	パフォーマンス エキスパートでのみ使用します。CPU リソースを最も多く使用した上位のコール パスを、指定の数だけエクスポートします。
/maxmethods=<integer>	パフォーマンス エキスパートでのみ使用します。CPU リソースを最も多く使用した上位のメソッドを、指定の数だけエクスポートします。

指定するオプションとオプション値を区切るには、等号、コロン、スペースのいずれかを使用します。

DevPartner.Analysis.Export.exe の使用例

以下に、**DevPartner.Analysis.DataExport.exe** の使用例を示します。

例 **1** : カバレッジ分析セッション ファイルを XML ファイルにエクスポートし、同じフォルダに保存します。

```
DevPartner.Analysis.DataExport.exe
“ c:¥WindowsApplication1¥WindowsApplication1.dpcov”
```

出力は、以下のファイルに保存されます。

c:¥windowsApplication1¥WindowsApplication1.dpcov.xml

例 **2** : ある場所に保存されているパフォーマンス分析セッション ファイルを、別のフォルダにエクスポートします。

```
DevPartner.Analysis.DataExport.exe
“ c:¥WindowsApplication1¥WindowsApplication1.dpprf”
/output=” c:¥temp”
```

出力は、以下のファイルに保存されます。

c:¥temp¥WindowsApplication1.dpprf.xml

例 **3** : 同じフォルダに保存されている複数のパフォーマンス エキスパート セッション ファイルをエクスポートします。

この例では、2 つのパフォーマンス エキスパート セッション ファイル（**WindowsApplication1.dppxp** と **WindowsApplication2.dppxp**）が同じフォルダに保存されていることを前提としています。

```
DevPartner.Analysis.DataExport.exe “c:\WindowsApplication1\*.dppxp”
```

出力は、以下のファイルに保存されます。

c:¥WindowsApplication1¥WindowsApplication1.dppxp.xml
c:¥WindowsApplication1¥WindowsApplication2.dppxp.xml

例 **4** : 同じフォルダに保存されている複数のカバレッジ分析、パフォーマンス分析、およびパフォーマンス エキスパートのセッション ファイルをエクスポートします。

この例では、以下の 3 つのセッション ファイルが同じフォルダに保存されていることを前提としています。

WindowsApplication1.dpprfÅAWindowsApplication2.dpcov、および
WindowsApplication3.dppxp

```
DevPartner.Analysis.DataExport.exe “ c:¥WindowsApplication1”
```

出力は、以下の 3 つのファイルに保存されます。

c:¥WindowsApplication1¥WindowsApplication1.dpprf.xml
c:¥WindowsApplication1¥WindowsApplication2.dpcov.xml
c:¥WindowsApplication1¥WindowsApplication3.dppxp.xml

例 **5** : パフォーマンス エキスパートのサマリをエクスポートし、CPU リソースを最も多く使用する上位 20 番めまでのメソッドを出力するように変更します。デフォルトでは、上位 10 番めまでのメソッドが出力されます。

```
DevPartner.Analysis.DataExport.exe
“ c:WindowsApplication1WindowsApplication1.dppxp”
/summary /maxmethods=20
```

出力は、以下のファイルに保存されます。

c:WindowsApplication1WindowsApplication1.dppxp.xml

索引

I

M

N

P

R

S

T

V

W

X

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

つ

て

と

な

ね

は

ひ

ふ

ほ

ま

み

め

ゆ

よ

ら

り

る

れ

ろ

わ