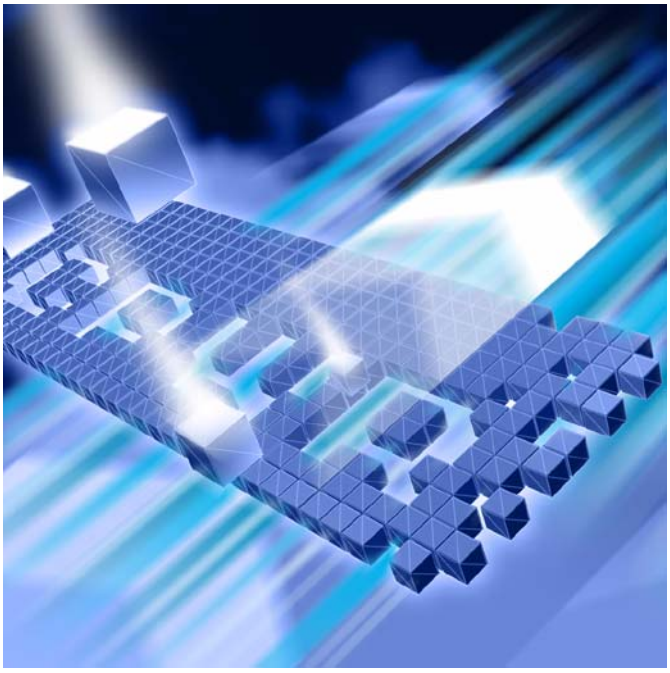


DevPartner®
ユーザー ガイド

DevPartner Studio Professional Edition
DevPartner for Visual C++ BoundsChecker Suite

リリース 8.2.1

COMPUWARE

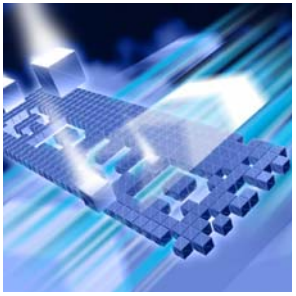
技術に関するお問い合わせは、電子メールでお寄せください。

call.center.japan@compuware.com

米国特許番号：5,987,249、6,332,213、6,186,677、6,314,558、6,760,903 B1、6,016,466

発行日：2007年10月1日

目次



はじめに

第 1 章

DevPartner の概要

第 2 章
エラー検出

第 3 章

静的なコード分析

第 4 章

自動コード カバレッジ分析

第 5 章
メモリ問題の検出

第 6 章

パフォーマンスの自動分析

第 7 章

詳細なパフォーマンス分析

第 8 章

システムの比較

付録 A

DevPartner Studio Enterprise Edition と TrackRecord

付録 B

DevPartner Studio でサポートされるプロジェクト タイプ

付録 C

コマンド ラインからの分析の開始

付録 D

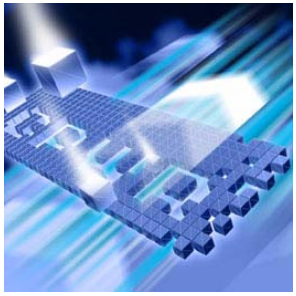
分析のセッション コントロール

付録 E

分析データの XML へのエクスポート

索引

はじめに



このマニュアルでは、Compuware® DevPartner® Studio ソフトウェアの使用方法の概要を説明します。

対象読者

このマニュアルは DevPartner Studio の新規ユーザーの方を対象に作成されています。第 1 章では DevPartner Studio のコンセプトの概要について説明し、以降の章では個々の DevPartner 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の冒頭には、新規ユーザーの方が DevPartner Studio を使用するための準備、設定、実行手順の概略が記載されています。

以前のバージョンから DevPartner を使用している場合は、『DevPartner インストール ガイド』の「はじめに」を読み、以前のバージョンからの相違点を把握してください。

このマニュアルには、Professional、Enterprise Edition、DevPartner for Visual C++ BoundsChecker Suite などの DevPartner Studio の全製品に関連する情報を記載しています。

メモ： DevPartner for Visual C++ BoundsChecker Suite で分析されるのはアンマネージ コードのみです。DevPartner のメモリ分析、静的なコード分析、パフォーマンス エキスパート機能で分析されるのはマネージ コードのみなので、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、Windows のインターフェイス、Microsoft Visual Studio、およびソフトウェア開発のコンセプトを理解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

このマニュアルに含まれる内容は以下のとおりです。

- **第1章**の「DevPartnerの概要」では、DevPartnerのコンセプトとコンポーネントについて説明します。
- **第2章**の「エラー検出」では、DevPartnerを使用してCコード、マネージC++コード、アンマネージC++コード内のエラーを検出する方法について説明します。
- **第3章**の「静的なコード分析」では、DevPartnerを使用してVisual Basicコードと Visual C#コード内のエラーを検出する方法について説明します。
- **第4章**の「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たかを追跡する方法について説明します。
- **第5章**の「メモリ問題の検出」では、DevPartnerを使って、メモリやオブジェクトの誤使用によって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**の「パフォーマンスの自動分析」では、DevPartnerを使用して、ボトルネックと、最適化が必要なコードを検出する方法について説明します。
- **第7章**の「詳細なパフォーマンス分析」では、DevPartnerを使用して、さまざまなパフォーマンス上の問題を分析する方法について説明します。
- **第8章**の「システムの比較」では、アプリケーション開発の問題を解決するために、DevPartnerを使用してコンピュータ システム間の違いを特定する方法について説明します。
- **付録A**の「DevPartner Studio Enterprise Editionと TrackRecord」では、DevPartner Studioの Enterprise Editionについて説明します。
- **付録B**の「DevPartner Studioでサポートされるプロジェクト タイプ」では、DevPartner Studioの各機能がサポートするプロジェクトの種類を表で示します。
- **付録C**の「コマンド ラインからの分析の開始」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D**の「分析のセッション コントロール」では、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 セッションのために、セッション コントロール ファイルを作成する方法について説明します。
- **付録E**の「分析データのXMLへのエクスポート」では、カバレッジ、パフォーマンス、パフォーマンス エキスパートの各データを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 コマンドとメニューの名前は**[ ]**で示します。以下に例を示します。**[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。以下に例を示します。

  『DevPartnerユーザー ガイド』（**Understanding DevPartner.pdf**）で説明します。

- 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。以下に例を示します。

  [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の特定のタスクの手順については、機能レベルのオンライン ヘルプを参照してください。

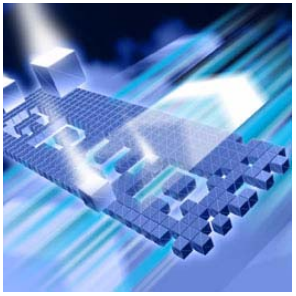
DevPartner Studioコンポーネントの詳細については、**[スタート]>[DevPartner]**メニューを選択して、DevPartner InfoCenterページを参照してください。

このマニュアルの他に、DevPartner Studioのドキュメント セットには以下の情報も記載されています。

- 『DevPartnerインストール ガイド』には、新機能情報、詳細なシステム要件リスト、インストール手順に関する説明が記載されています。
- 『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』には、CompuwareのDevPartnerエラー検出ソフトウェアの使用方法を理解する上で役立つコンセプトと手順が記載されています。
- Known Issues（既知の問題）のファイルには、DevPartner Studioの既知の問題とテクニカル ノートが入っています。このファイルはインストール ディレクトリに保存されます。また、Webの**ReadMe.htm**にKnown Issues（既知の問題）のファイルへのリンクが含まれています。

第 1 章

DevPartner の概要



- DevPartner Studio の機能
- DevPartner と Visual Studio
- Visual Studio Team System のサポート
- ターミナル サービスとリモート デスクトップの使用
- ソフトウェア開発サイクルの中の DevPartner

この章では、DevPartner Studio Professional Edition と DevPartner for Visual C++ BoundsChecker Suite の概要を説明します。このマニュアルを使用すると、2 つの DevPartner 製品の基本的なコンセプトを理解することができます。

メモ： DevPartner for Visual C++ BoundsChecker Suite で分析されるのはアンマネージ コードのみです。DevPartner のメモリ分析、静的なコード分析、パフォーマンス エキスパート機能で分析されるのはマネージ コードのみなので、DevPartner for Visual C++ BoundsChecker Suite ではサポートされていません。

DevPartner Studio の機能

DevPartner Studio は、多様なプログラマの生産性向上支援機能を備えたソフトウェア ツールです。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、パフォーマンスのプロファイリング、システム パフォーマンス分析、システム比較などの機能があります。

DevPartner を使用すると、多様な言語で記述された幅広いマネージ アプリケーションとアンマネージ アプリケーションを分析できます。サポートされているプロジェクトの種類と言語のリストについては、付録 B「DevPartner Studio で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。

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

エラー検出

DevPartner Studio にはエラーの自動検出機能があり、マネージ C++ プログラムとアンマネージ C++ プログラムの両方に使用できます。DevPartner のエラー検出は BoundsChecker ™ テクノロジに基づいて構築されており、Windows ベースのアプリケーション内で、検出の困難な以下のエラーを短時間で検出できます。

- メモリ、リソース、および COM インターフェイスのリーク
- Windows API コールの誤った使用
- メモリまたはポインタの誤った使用
- 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 エラー
- .NET ファイナライザ内のエラー

DevPartner のエラー検出機能によって、作成の瞬間から、メモリからプロセスをアンロードする直前の最後の瞬間まで、アプリケーションが監視されます。アプリケーションの通常のフローだけでなく、すべての DLL のロードとアンロード、静的なコンストラクタやデストラクタも監視できます。また、DevPartner のエラー検出機能を調整し、アプリケーションの特定ファイルまたは特定部分をフィルタすることで、特定の問題を解決するために必要な情報のみを収集することもできます。

DevPartner のエラー検出機能の詳細については、「エラー検出」（13 ページ）を参照してください。

静的なコード分析

DevPartner は、Visual Studio 内で、Visual Basic .NET および Visual C# と互換性のあるコードの作成を支援します。DevPartner では、.NET Framework におけるプログラミングとネーミングの違反の識別、メソッドコール構造の分析、コード全体の複雑度の追跡を実行できます。

DevPartner ソフトウェアは多様なコーディング エラーを検出します。

- 一貫性のない変数のネーミング
- コーディング基準の違反
- Win32 API バリデーション エラー
- 共通ロジック エラー
- .NET の移植性の問題
- 構造化された例外処理エラー

また、DevPartner と拡張されたルール セットや拡張可能なルール セットを使用すると、.NET 環境で動作しないコンストラクトを特定できるので、既存の Visual Basic コード資産の移植に役立ちます。

DevPartner の静的コード レビューの詳細については、「静的なコード分析」（57 ページ）を参照してください。

カバレッジ分析

開発者とテスト エンジニアは、DevPartnerのカバレッジ分析機能を使用するとアプリケーション コード全体を確実にテストできます。カバレッジ分析を使用してテストを実行すると、テストでカバーされた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、コードの各行が追跡されます。テストが終了すると、テストされたコードとテストされなかったコードに関する情報が表示されます。

WebやASP.NETのアプリケーションなどのマネージ コードのアプリケーションと、アンマネージのVisual C++アプリケーションとVisual Basicアプリケーションの両方について、カバレッジ データを収集できます（サポートされているテクノロジのリストについては、付録B「DevPartner Studioで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。テストでカバーされたコードの割合がわかると、エラー検出結果が信頼できるものであることを確認できます。

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」（121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アプリケーションによるメモリの割り当て状況を分析します。DevPartner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、オブジェクトやクラスによって消費されたメモリ量を表示したり、メモリにオブジェクトを保持している参照を追跡したり、メソッド内のソースコードからメモリを割り当てたコード行を特定したりすることができます。

さらに重要な点は、メモリ データがコンテキストで表示されることです。これにより、コード内のオブジェクト参照チェーンとメソッドのコール シーケンスを移動できます。メモリ データをコンテキストで表示することで、プログラムでメモリがどのように使用されているかを詳しく理解し、メモリの使用を最適化するのに必要な重要情報が得られます。

メモリ分析の詳細については、「メモリ問題の検出」（151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、パフォーマンスのボトルネックについてコードを分析します。この機能によって、これらのボトルネックがソース コードの各行で特定できます。また、サード パーティ製コンポーネントや、オペレーティングシステム内、さらには最も重要な.NET Framework内のメソッドレベルの理解にも役立ちます。

また、Microsoft Visual Basic 6.0とMicrosoft Visual C++ 6.0のパフォーマンス プロファイリングと、.NETアプリケーションで使用される古いVisual Studio 6.0コンポーネントのプロファイリングもサポートしています（サポートされているテクノロジのリストについては、付録B「DevPartner Studioで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改善するために、DevPartnerパフォーマンス分析でパフォーマンスのボトルネックを検出し、パフォーマンスが実際に改善された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、「パフォーマンスの自動分析」（207ページ）を参照してください。

詳細なパフォーマンス分析

DevPartner Studioパフォーマンス エキスパート機能は、DevPartnerのパフォーマンス分析機能よりもさらに進んだパフォーマンス プロファイリングを行います。マネージコードのVisual Studioアプリケーションの場合、パフォーマンス エキスパートには、以下のように解決が困難な問題を詳しく分析できる機能があります。

- CPU／スレッドの使用状況
- ファイルまたはディスクのI／O
- ネットワークI／O
- 同期の待機時間

パフォーマンス エキスパートは、実行時にアプリケーションを分析し、コード内で問題のあるメソッドを特定します。ユーザーは、メソッド内の行に関する詳細を個別に表示したり、親子の呼び出し関係を調べて問題解決に最良の方法を判断したりすることができます。解決へのアプローチを決定したら、パフォーマンス エキスパートを使用して、ソース コードの問題のある行まで直接ジャンプできます。このように問題の修正を迅速に行うことが可能です。

詳細については、「詳細なパフォーマンス分析」（243ページ）を参照してください。

システム比較

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次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

System Comparison では、システムを比較するために、スナップショット ファイルという XML ファイルが作成されます。このファイルには、インストールされた製品、システム ファイル、ドライバ、その他多くのシステム特性など、コンピュータ システムに関する情報が記載されます。ファイルの作成後、スナップショット ファイルが比較され、相違点が報告されます。

他の DevPartner 機能とは異なり、System Comparison は Visual Studio 環境に統合されません。ターゲット システムへの影響を最小限に抑えるために、スタンドアロン ユーティリティとして実行されます。

System Comparison は、システムのスナップショットを毎晩作成するサービスと、手動でスナップショットを作成し、スナップショットを比較して相違点を見つけるユーザー インターフェイスから構成されます。System Comparison には、コマンド ライン インターフェイスと Software Development Kit（SDK）も含まれます。SDK を使用してソフトウェアを開発すると、比較に関する追加情報を集めたり、運用アプリケーションにスナップショット機能を埋め込んだりすることができます。

System Comparison ユーティリティの詳細については、「システムの比較」（283 ページ）を参照してください。

DevPartner と Visual Studio

DevPartner は、Visual Studio 環境にシームレスに統合されます。この統合によって、アプリケーションの作成やデバッグを行う場合に、製品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、これらのコードの分析を何回も実行できます。

DevPartner Studio ソフトウェアは、Visual Studio 2003 環境内と Visual Studio 2005 環境内のアプリケーション開発をサポートしています。また、Visual Studio 6.0 での Visual C++ 開発と Visual Basic 開発をサポートしています。これは、開発者が古い Microsoft 環境から最新の .NET Framework にコードを移行するときに役立ちます。

各 Visual Studio 環境で使用できる DevPartner 機能を以下の表に示します。

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

Microsoft Visual Basic 6.0	Microsoft Visual C++ 6.0	Microsoft Visual Studio 2003 または 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
		静的なコード分析
		メモリ分析
		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 Basic 6.0	Microsoft Visual C++ 6.0	Microsoft Visual Studio 2003 または 2005
サポートされていません	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
	エラー検出	エラー検出

Visual Studio のメニューとツールバー

DevPartner によって、Visual Studio にメニューと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）メニューなどのメニュー コマンドが Visual Studio の複数のメニューに追加されます。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、静的コード レビューのルール、オプション ダイアログ ボックス、およびインストゥルメンテーション制御にアクセスできます。

Visual Studio にツールバーが追加されるので、DevPartner 機能に簡単にアクセスできます。Visual Studio 2005 の DevPartner ツールバーについては、図 1-1（7 ページ）で説明します。

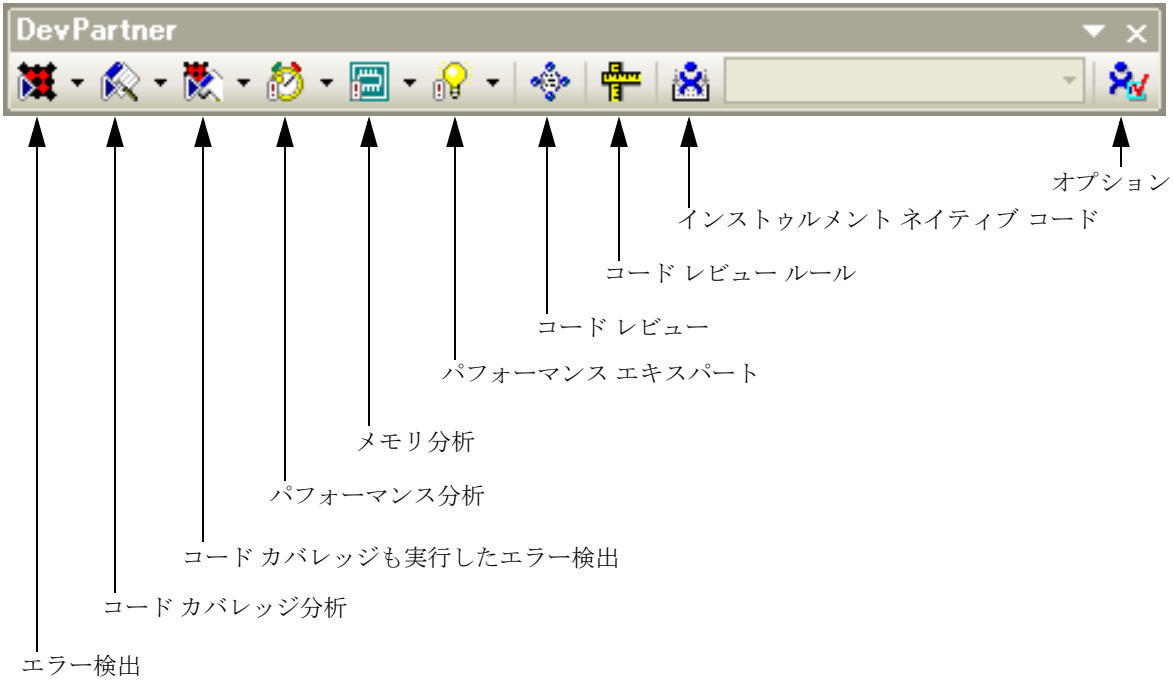


図 1-1. DevPartner ツールバー

また、IDE にセッション コントロール ツールバーも配置されます。カバレッジ分析、メモリ分析、パフォーマンス分析、パフォーマンス エキスパート機能がアクティブな場合、セッション コントロール ツールバーがアクティブになります。

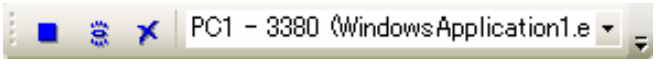


図 1-2. DevPartner のセッション コントロール ツールバー

DevPartner のセッション コントロール ツールバーは、3 つのアイコンと 1 つのプロセス リストから構成されます。

- データ収集を**停止**し、最終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。
- クリア アクションが実行される時点までに収集されたデータを**クリア**します。

複数のプロセスを使用するアプリケーションの場合、プロセス リストは、単一プロセスでのデータ収集のみを扱います。

DevPartner では、メニューとツールバーの他に、Visual Studio 2003 ユーザーと 2005 ユーザー向けに、Visual Studio のドック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を表示します。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 ファイル名を表示します。さらに、DevPartner によって、DevPartner のコード分析動作を設定するためのページが、Visual Studio オプション、ソリューション プロパティ、プロジェクト プロパティに追加されます。

Visual Studio での DevPartner の使用

Visual Studio 内で DevPartner を使用するための一般的なワークフローは、以下に挙げる一般的なタスクのいずれかまたは複数のタスクで構成されます。

- Visual Studio でソリューションを作成するか、または開く。
- コード分析操作のためのオプションを設定する。
- DevPartner のメニューまたはツールバーから、実行する分析をオンにする。
- アプリケーションを実行する。
- DevPartner から返されたセッションの結果を表示する。

DevPartner では、監視するアプリケーションの部分の選択、表示するデータの選択、不要な情報を除外するフィルタの作成などを柔軟に設定できます。

また、DevPartner には、コマンド ラインから多くの機能を実行するオプションもあります。この機能によって、夜間構築のスモーク テストなどの自動化されたバッチ プロセスの動作でも、DevPartner の機能を活用できます。

統合されたオンライン ヘルプ

DevPartner には、機能についての広範なオンライン ヘルプがあります。使用方法に関する情報や参照情報が必要な場合は、このヘルプを最初に参照してください。

DevPartner オンライン ヘルプは Visual Studio のその他のヘルプと同じ形式で提供され、Visual Studio ヘルプ コレクション内の別個のブックとして表示できます。DevPartner Studio ブックには、DevPartner 機能ごとに 1 つのボリュームが含まれます。

Visual Studio Team System のサポート

Visual Studio 2005 Team System は、Visual Studio 2005 ソフトウェア開発プロジェクト向けのバージョン管理、バグ追跡、プロセス管理を行う Microsoft のソフトウェアです。Microsoft Visual Studio Team System のクライアント ソフトウェアがインストールされ、Team Foundation Server が使用可能になっている場合に、DevPartner Studio は Team System をサポートします。

DevPartner Studio では、種類が**バグ**の**作業項目**を Visual Studio Team System へ送信できます。**バグ**を送信すると、**[作業項目]** フォームに選択したセッション データが自動的に挿入されます。DevPartner Studio から**バグ**を送信するには、種類が**バグ**の**作業項目**がアクティブな Team System プロジェクトでサポートされている必要があります。DevPartner Studio では、この種類の**作業項目**にのみ、データが自動的に追加されます。

DevPartner セッション ファイルの以下に示すビューのいずれかから、DevPartner データを含む**作業項目**を送信できます。

- カバレッジ、メモリ、またはパフォーマンス分析のセッション ファイル、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]**または**[ネーミング]**タブ
- エラー検出の任意のタブのエラー リストまたはリーク リスト、またはエラー検出の**[モジュール]**タブまたは**[.NET パフォーマンス]**タブのインスタンス リスト

DevPartner Studio から Team System の**作業項目**を送信するには、DevPartner セッション ファイルのメソッドや他の対象項目を右クリックし、**[作業項目の提出]**を選択します。[Title]、[Description]、または [Symptom] フィールドにデータが挿入されます。他の必要なデータを入力し、**作業項目**を保存します。

メモ： **Visual Studio で Team Explorer コンテキスト メニューを使用した場合は、[作業項目] にセッション データが自動的に挿入されません。**

DevPartner Studio から Team System へのデータ送信の詳細については、このマニュアルの各章に記載されている Visual Studio Team System のセクションを参照してください。Team System を使用して開発作業とプロジェクト管理作業を行う方法の詳細については、Microsoft Visual Studio 2005 Team System のドキュメントを参照してください。

ターミナル サービスとリモート デスクトップの使用

DevPartner Studio は Windows ターミナル サービスをサポートしています。ターミナル サービスを使用すると、コンピュータを直接使用して実行できる以下のような処理をすべて実行できます。

- リモート システムで DevPartner オプションを設定します。
- リモート システムで分析を有効または無効にします。
- リモート システムで実行するアプリケーションをプロファイルします。

ライセンス処理

ターミナル サービス接続には、1 台のユーザー ディスプレイごとに 1 つの DevPartner コンカレント 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するサーバーには、DevPartner Studio Remote Server ライセンスは必要ありません。

ターミナル サービスでの複数セッションの実行

複数の DevPartner セッションをターミナル サーバーで同時に実行できます。単一ユーザーまたは複数ユーザーが複数セッションを起動できます。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、どちらのインスタンスも同じワークスペース設定を共有します。これは、DevPartner がユーザーごとにワークスペース設定を格納するためです。異なるユーザーがカバレッジ分析のインスタンスを起動する場合、インスタンスごとに別のワークスペース設定を設定できます。

収集データには、ターミナル サーバー上のすべてのユーザーからのサーバー プロセスのアクティビティが含まれます。セッション中はデータ収集に集中できるように、

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したりする余分なアプリケーションのアクティビティは除外または制限してください。

ソフトウェア開発サイクルの中の DevPartner

ソフトウェア開発プロジェクトはいくつかのフェーズで構成され、ソフトウェア開発ライフ サイクルと呼ばれます。ソフトウェア開発ライフ サイクルは開発部門によって異なりますが、DevPartner は実質的にどのような開発ライフ サイクル モデルにも適用できます。

ヒント: 各フェーズ間のアクションをプロジェクトのマイル ストーンと定義することもあります。

一般的な開発ライフ サイクルのフェーズを以下の図に示します。

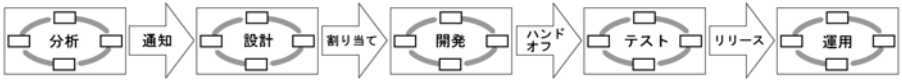


図 1-3. 一般的なソフトウェア開発ライフ サイクルのフェーズ

これらの各フェーズの中で、プロジェクト マネージャ、開発リーダー、開発者、テスターが、エラーや不規則なコーディング、パフォーマンスのボトルネックやメモリの問題を最小限に抑えたコードを作成できるように DevPartner は支援します。

分析フェーズと設計フェーズの間に、DevPartner Enterprise Edition の要件定義機能とプロジェクト追跡機能を使用すると、プロジェクト チーム全体のコミュニケーションが明確になります。

開発フェーズとテスト フェーズは、多くの場合、最も時間がかかり不安定なフェーズですが、DevPartner を使用すると、開発者はソフトウェアのバグを見つけて解決できるだけでなく、開発中のアプリケーションを調整してテストできます。DevPartner の機能によって生成された情報は、開発チームのメンバーで共有できます。

テスト フェーズでは、開発部門は内部的なロード テストとシナリオに基づくテストを使用して、開発中のアプリケーションで機能の動作を検証します。この内部的なテストは、開発ライフ サイクルが終了するまで継続します。DevPartner は、開発サイクルのテスト フェーズで多くの利点を提供します。アクティブな分析テクノロジを使用すると、DevPartner のエラー検出機能とパフォーマンス分析機能を、コンピュウェアの *QA*Run や *QA*Load などの *QA*Center テスト ツールと統合して、アプリケーション テスト プロセスを合理化できます。

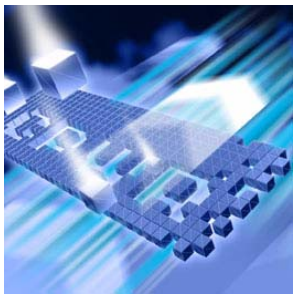
最後に、運用フェーズで DevPartner を使用すると、開発チームは最終的な製品リリースに大きな自信を持ってアプリケーションのビルドおよびリリースを行うことができます。しかし、どんなに最新のテクノロジを利用したとしても発見できないエラーや問題があります。このような問題はエンド ユーザーに悪影響を与える場合があるため、問題が確認されたらすぐに対処する必要があります。DevPartner は、エラーの検出、確認、および解決を行い、この運用段階でもお役に立つことができます。

また、開発チームが大きくなったり、アプリケーションが複雑化した場合は、DevPartner Enterprise Edition（日本語版は販売されていません）にある障害管理および統合テクノロジを利用すると、運用段階における生産性をさらに向上させるこ

とができます。DevPartner Enterprise Edition は、DevPartner と、Compuware TrackRecord アプリケーションおよびReconcile アプリケーションと統合できます。エンタープライズ環境でのDevPartner の使用の詳細については、「DevPartner Studio Enterprise Edition と TrackRecord」（311 ページ）を参照してください。

第 2 章

エラー検出



- エラー検出の機能
- すぐにエラー検出を使用するには
- ActiveCheck と FinalCheck を使用する場合の判断
- [検出されたプログラム エラー] ダイアログ ボックスの使用
- [メモリおよびリソース ビューア] ダイアログ ボックス
- [抑制] および [フィルタ] ダイアログ ボックス
- コール バリデーション
- [設定] ダイアログ ボックスの使用
- Windows メッセージとイベント ログの追跡
- データの XML へのエクスポート
- コマンド ラインからのエラー検出の実行
- Visual Studio Team System へのデータの送信

この章には 2 つのセクションがあります。1 つめのセクションには、はじめてのユーザーがエラー検出機能を利用できるように、簡単な操作手順が記載されています。2 つめのセクションには、DevPartner のエラー検出機能を詳しく理解するための参考情報が記載されています。

エラー検出に関するその他のタスクに基づく情報については、DevPartner のオンライン ヘルプを参照してください。より高度な機能については、DevPartner ソフトウェアに付属の PDF 形式のガイド、『エラー検出ガイド』を参照してください。

エラー検出の機能

DevPartnerエラー検出はCおよびC++の開発環境で利用できる総合的なデバッグ ソリューションです。DevPartnerエラー検出を使用した高頻度の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で、安定性が高くエラーがないコードを作成できます。DevPartnerエラー検出は、開発プロセスの時間を増やすことなく、エラーの検出と分析を自動化します。以下の機能を使用すると、従来のデバッグ技術やテスト技術では特定できなかったわかりづらいバグを特定できます。

- 徹底したエラー検出
- 各種の開発環境への対応
- Visual Studioデバッガとの統合
- Microsoft Visual C++6.0およびVisual Studioとの統合
- 一歩進んだエラー分析
- 拡張性のあるエラー検出のアーキテクチャ

すぐにエラー検出を使用するには

以下の準備、設定、実行手順では、DevPartnerエラー検出の使用方法を紹介します。

機能をすぐに使用したい場合は、色付きの枠内に記載されている手順に従ってください。色付きの枠内に記載されているトピックの詳細な情報については、枠の下に記載されている文章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：エラー検出の分析範囲の決定

コードに対する DevPartner エラー検出の実行方法、特定する必要があるエラーとメモリ リークのタイプについて検討します。

メモ： **DevPartner エラー検出では、ターゲット ファイルごとにデータ ファイルが作成されます。エラー検出を開始する前に、ターゲット実行ファイルが保存されたディレクトリに対する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。**

この手順では以下を前提としています。

- ソリューションにアンマネージ ソース コードが含まれています。
- Visual Studio 2003 または Visual Studio 2005 でエラー検出を実行しています。

メモ： **DevPartner エラー検出でサポートされているプロジェクト タイプのリストについては、****「エラー検出でサポートされるプロジェクト タイプ」（323 ページ）を参照してください。**

セッションの実行方法の決定

状況の必要に応じて、DevPartner エラー検出の実行方法がいくつかあります。

- Microsoft Visual Studio 内から定期的なコード バリデーション プロセスの一部として、またはスタンドアロン アプリケーションを使用して、インタラクティブ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。また、DevPartner エラー検出の設定の変更、プログラムのチェック、検出されたエラーの表示を行うことができます。
  - DevPartner エラー検出は、Visual Studio から完全に独立したアプリケーションとして実行できますが、Visual Studio エディタにアクセスしてコードを編集することはできません。
- `bc.exe` を使用して、バッチ ファイルまたはコマンド ラインからエラー検出を自動実行します。
  - DevPartner エラー検出を DOS コマンド ラインから実行する場合は、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 ラインからのエラー検出の実行」（54 ページ）を参照してください。
- 開発の主要マイルストーンでコードに FinalCheck をインストゥルメントして、全体的なバリデーションを行います（Visual Studio 内でのみ）。

特定するエラー タイプの決定

エラー検出は、コードで発生するさまざまなエラーとリークを特定するだけでなく、問題が疑われる部分を追跡するためにも使用できます。

- COM オブジェクトを正しく使用しているはずでも、エラーがないことを確認したい場合には、COM オブジェクトの追跡を有効にします。
- 同期オブジェクトを正しく使用していることを確認したい場合、またはアプリケーションのデッドロックがときどき発生し、理由がわからない場合は、デッドロック分析を有効にします。
- 実装にリークやエラーがないことを確認するには、メモリの追跡システムを拡張してカスタム アロケータを含めます。カスタム アロケータについて説明するには、アロケータに関する説明情報を `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：オプションと設定の選択

DevPartner エラー検出をカスタマイズすると、不要な「ノイズ」を無視またはフィルタして、特定のエラー タイプを報告できます。

この手順では、デフォルトの DevPartner 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。

メモ： **DevPartner エラー検出の実行方法に応じて、[設定] ダイアログ ボックスにアクセスするメニュー オプションは複数あります（「[設定] ダイアログ ボックスの使用」（37 ページ）を参照）。**

デフォルトでは、エラー検出によって単純なメモリ リーク、一部のメモリ エラー、リソース リークが検出されます。また、デフォルト構成を編集すると、以下のタイプのエラー、リーク、イベントをすべて検出することができます。

- 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
- メモリの割り当てと解放
- Windows のメッセージと他の重要なイベント（「Windows メッセージとイベント ログの追跡」（52 ページ）を参照）

また、FinalCheck を使用するように DevPartner エラー検出を構成できます。FinalCheck を使用すると、エラー検出によって C または C++ アプリケーションがインストゥルメントされ、エラーが発生したステートメントまで正確に特定できます。FinalCheck の実行には時間がかかり、使用するリソースも増えますが、検出が困難なメモリ エラー、ポインタ エラー、リーク エラーを正確に特定できます。

特定のエラーやリークのためにエラー検出を構成する以外に、以下の設定が可能です。

- 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
- 再利用するためにパラメータを構成ファイルに保存する
- 別の構成ファイルを現在のセッションにロードする

実行：エラー検出を使用したソリューションの実行

DevPartner エラー検出を有効にしてソリューションを実行する準備が整いました。

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. エラーをチェックするプログラムの部分を実行します。重大なエラーが発生するたびに、**[検出されたプログラム エラー]** ダイアログ ボックスが表示されます（図 2-1 を参照）。他のエラーはあまり重大ではないか一般的なもので、記録され、**結果**ペイン（エラー検出のメイン ウィンドウの左上にあります）に表示されるため、あとで対処できます。

[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 ボックスの使用」（27 ページ）を参照）には、エラーの説明が表示されます。また、続けてコール スタック情報と、エラーが検出されたコードのセグメント（可能な場合）が表示されます。検出されたエラーの詳細な説明については、**[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る場合、以下のいずれかの方法で対応し、プログラムを続行します。

メモ： **[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、この手順をスキップできます。**

- ◇ **[説明]**－エラーの詳細な説明が表示されます。
- ◇ **[抑制]**－[抑制]ダイアログ ボックスを開き、このエラーを事前にコピーします。エラーを抑制すると、以降、そのエラーはエラー検出によって処理されなくなります。「[抑制]および[フィルタ]ダイアログ ボックス」（30ページ）を参照してください。
- ◇ **[デバッグ]**－Visual Studioデバッガで、コードのエラーが発生した行を開きます。
- ◇ **[中止]**－プログラムを終了し、**結果**ペインにフォーカスを置きます。
- ◇ **[続行]**－エラーを承認し、次に進みます。エラーは**結果**ペインに表示されるので、セッションの完了後にレビューできます。

5 プログラムのチェックが終わったら、プログラムを終了します。プログラムが自然に終了しないことがよくありますが、十分なデータを収集した場合には終了する必要があります。プログラムを終了するには、以下の3つの方法があります。

- ◇ **[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]**メニューの**[デバッグの停止]**を選択します。
- ◇ プログラムを終了します。

基本的なエラー検出セッションの実行が終わりました。**結果**ペインを確認して、検出したエラーとリークのデータを分析します。

結果ペインのデータの分析

エラー検出のメイン ウィンドウ（図 2-2 を参照）の左上にある**結果**ペインには、さまざまな情報を表示するタブがあります。

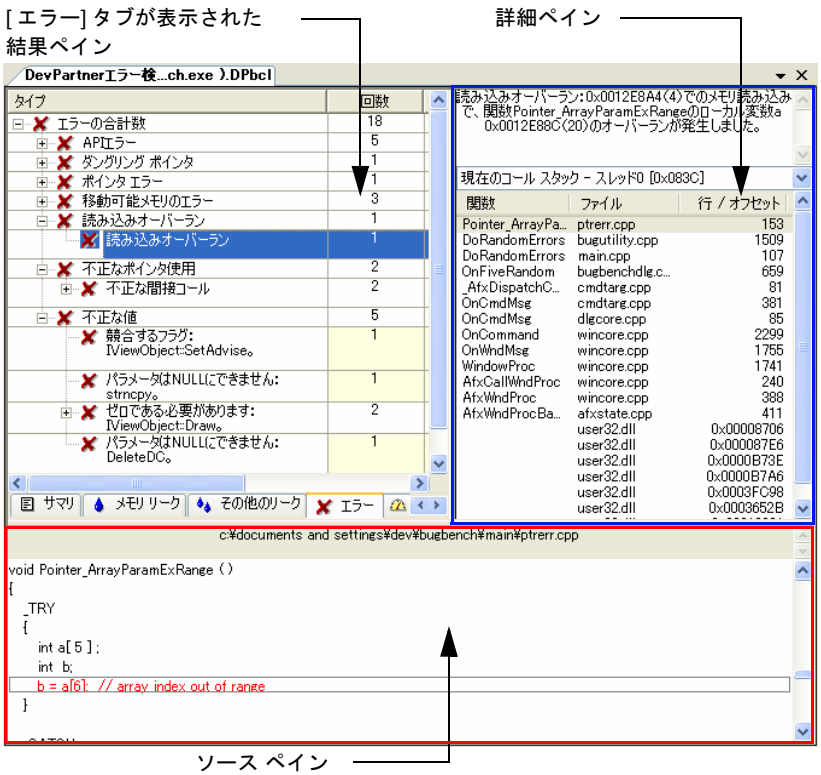


図 2-2. DevPartner エラー検出のメイン ウィンドウ

セッションが終了したら、**結果**ペインの**[サマリ]**タブにフォーカスを移動して、データのレビューを開始します（図 2-3 を参照）。

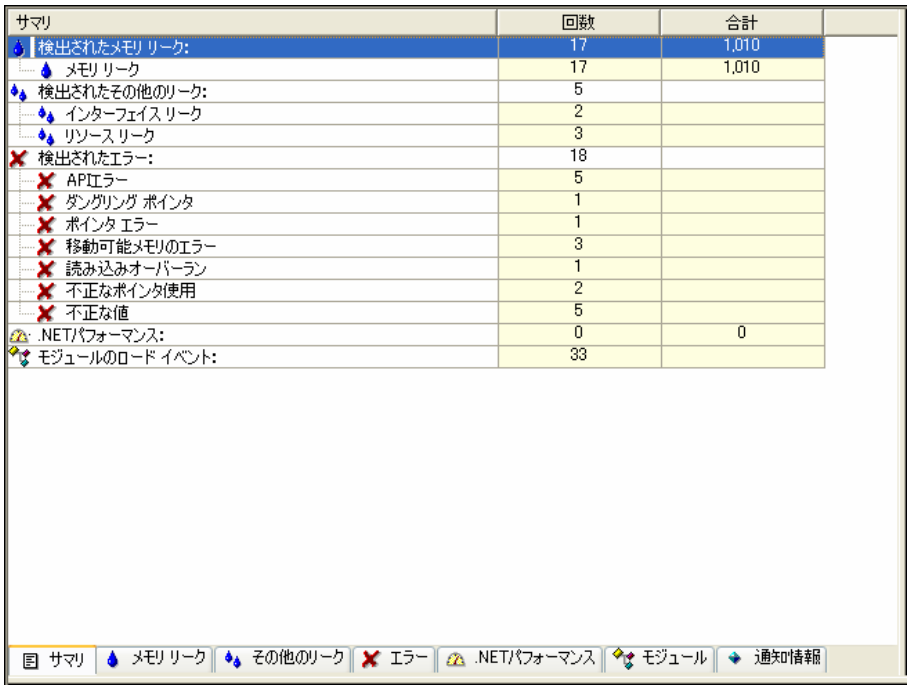


図 2-3.　　[サマリ]タブが表示された結果ペイン

[サマリ]タブには、セッションで検出されたすべてのエラーとリークの概要が表示されます。特定のイベントをダブルクリックすると、選択したイベントの詳細が表示されたタブに移動します。

1. 検出されたエラーとリークの概要については**結果**ペインの**[サマリ]**タブを確認します。
2. **[サマリ]**ペインに表示されている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応じたタブにフォーカスが切り替わります。
   タブではエラーがカテゴリに分類され、再発するエラーにフォーカスしやすくなっているため、エラーを診断して修正できます。
3. 特定のカテゴリをすべて展開し、特定のリーク、エラー、またはイベントを選択します。
   リストの最上位には、リーク、エラー、イベントのカテゴリが表示されます。
   特定のカテゴリをすべて展開すると、検出された個々のエラー、リーク、イベントが表示されます（図 2-4（21 ページ）を参照）。

図 2-4.　　選択したエラーが表示された[エラー]タブ

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]**、**[通知情報]**というタブがあります（図2-4（21ページ）を参照）。これらのタブから、表示されているデータの分類または評価のために、他の操作を実行できます。

ヒント: また、**結果**ペインの各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。この操作で**[ソース]**ペインにソース ファイルが開き、エラーが発生したコードの行にフォーカスが移動します。

- これらのタブに表示されるデータをソートするには、カラム見出し（**[その他のリーク]**タブの**[タイプ]**、**[回数]**、**[デアロケータ]**など）をクリックします。
- イベントの詳細を表示するには、イベントを右クリックし、**[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のイベントを表示するには、イベントを右クリックし、**[通知情報で検索]**を選択します。**[通知情報]**タブには、アプリケーション内で発生したすべてのイベントが日付順にリストされます。

4　**詳細**ペインを確認します（図2-2（19ページ）を参照）。**詳細**ペインの上部には、選択したエラーの詳細が表示されます。詳細の下には現在のコール スタックが表示されます。

エラー検出のメイン ウィンドウの右上のセクションが**詳細**ペインです（図 2-2（19 ページ）を参照）。**詳細**ペインに表示される情報のタイプは、現在選択しているイベントによって変わります。**詳細**ペインには、常にエラーまたはイベントの詳細が表示されますが、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されることがあります。

5 コール スタックを確認します。
エラー タイプに応じて、コール スタックにはエラーまたはリークが検出された場所または割り当てられた場所が表示されます。複数のコール スタックが使用できる場合、ドロップダウン リストを使用して切り替えることができます。

エラー検出のメイン ウィンドウの下部は**ソース**ペインです（図 2-2（19 ページ）を参照）。**ソース** ペインには、現在選択しているコール スタックと関連付けられているソース ファイルが表示されます。したがって、**詳細**ペインで別のコール スタックを選択すると、ソース コードも変わります。

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、エラーまたは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ます。

7 ソース コードをレビュー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。
Visual Studio エディタでソース ファイルが開かれ、**ソース** ペインに表示された同じ場所が表示されます。

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。

ここまでで、DevPartner エラー検出を使用してコードのエラーまたはリークを特定し、エディタに移動して修正を行いました。

セッション ファイルの保存

セッション ファイルを保存すると、結果を再度参照することができます。以下のような理由から保存したセッション ファイルを開くことがあります。

ヒント：[全般]設定を使用して[ファイル]>[閉じる]を選択したときに、セッション ファイルを保存することを確認するメッセージが表示されるようにエラー検出を構成できます。

- 以前に発生した種類のリークやエラーを再確認する場合
- 以下の目的で、セッション データを XML にエクスポートする予定の場合（「データの XML へのエクスポート」（53 ページ）を参照）
  - データを他の関係者に公表する
  - 複数のセッション間でデータを比較する
  - 傾向のデータベースを構築する
- このセッションで検出されたエラーの修正をあとで続行する場合

メモ： セッション ファイルは `.dpbcl` という拡張子で保存されます。セッション ファイルのデフォルトの保存場所は、実行ファイルと同じディレクトリです。

10 **[ファイル]>[選択したファイルに名前を付けて保存]**を選択してセッション ファイルを保存します。

11 **[ファイルに名前を付けて保存]**ダイアログ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。

Visual Studio C++ 6.0 でエラー検出を実行している場合と、スタンドアロン アプリケーションを実行している場合とで、セッション ファイルを保存する手順は変わります。

Visual C++ 6.0 からのセッション ファイルの保存

1 **[ファイル]>[名前を付けて保存]**を選択します。

2 **[名前を付けて保存]**ダイアログ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のセッション ファイルの保存

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 **[名前を付けて保存]**ダイアログ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。

この章の準備、設定、実行セクションはこれで終了です。エラー検出セッションの実行方法について基本的な知識が習得できたはずです。詳細については、この章の残りを続けて読んでください。高度なトピックの詳細な説明については『エラー検出ガイド』を参照してください。また、タスクごとの情報については DevPartner のオンライン ヘルプを参照してください。

ActiveCheck と FinalCheck を使用する場合の判断

DevPartner エラー検出では、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 について

ActiveCheck テクノロジとは、ソース コードをインストゥルメントすることなくエラー、リーク、イベントをチェックする標準的操作のことです。コードのインストゥルメントが不要なので、プログラムを再コンパイルまたは再リンクすることなく、ActiveCheck でエラーを検出します。ActiveCheck はすべてのエラー検出セッションで有効です。

ActiveCheck を使用すると、以下のことを実行できます。

- 実行時の API 検証エラーのレポート
- プログラム終了時のメモリ リークとリソース リークのレポート
- メモリやリソースが割り当てられた行や、エラーが発生した行の特定
- 潜在的なデッドロックの特定

DevPartner エラー検出で ActiveCheck を使用してプログラムを実行すると、実行時にプログラムが自動的に分析されます。DevPartner エラー検出によって、プログラムの API コール、メモリの割り当てと解放、メッセージ、重要なイベントなどが監視され、このデータを使ってエラーの検出とプログラム実行の完全なトレースが行われます。また、ソース コードがないプログラムのエラーも検出できます。

ActiveCheck では再コンパイルと再リンクが必要ないため、連続して使用できます。ActiveCheck をソフトウェア開発工程全体にわたって使用すると、API 検証エラー、デッドロック、リソース リーク、COM インターフェイス リークなどを検出できます。

表 2-1 と表 2-2 は、ActiveCheck で検出されるエラーを示します。

表 2-1. ActiveCheck によって検出される API エラー、COM エラー、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 メソッドの引数 • パラメータ範囲エラー • スレッドの不正使用 • 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 • メモリ割り当ての不一致 • アンロックされたメモリ ブロックをポインタが参照する場合 • 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によって検出されるデッドロック関連エラー、.NETエラー、ポインタとリーク 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が呼び出されない • 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について

FinalCheckは、コンパイル時に診断ロジックをコードに挿入する特許取得済みのテクノロジです。FinalCheckを使用すると、DevPartnerエラー検出によって、エラーが発生したステートメントを正確に特定することができます。

FinalCheckは、重要なプロジェクトのイベントや検出困難なエラーの検出に使用してください。FinalCheckによって、ActiveCheckで検出されるエラーだけでなく、表2-3に示すエラーも検出されます。

表 2-3. FinalCheck で検出されるエラー

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 の比較ー例

new または **malloc** を使用してメモリ ブロックを割り当て、ポインタをローカル変数に格納すると、DevPartner エラー検出によってこの情報が記録されます。あとで別の値をこのローカル変数に再度割り当てる場合は、あらかじめメモリ ブロックの割り当てを解除するか、またはこのポインタの割り当て先を別の変数に変更します。これを行わないとアプリケーション内でリークが発生します。

- **ActiveCheck を使用する場合：** DevPartner エラー検出によって、**malloc** または **new** によって割り当てられたブロックでリークしたことがレポートされ、このメモリの割り当てられた行が指摘されます。エラーは、アプリケーションの終了時にレポートされます。
- **FinalCheck を使用する場合：** DevPartner エラー検出によって、ブロックの割り当て先がレポートされ、このブロックを参照する最後の変数に新しい値を割り当てている行が強調表示されます。エラーは、発生時にレポートされます。

[検出されたプログラム エラー]ダイアログ ボックスの使用

DevPartnerエラー検出は、アプリケーションに重大なエラーを検出すると**[検出されたプログラム エラー]**ダイアログ ボックス(図 2-5 (27ページ) を参照) を表示します。

[検出されたプログラム エラー]ダイアログ ボックスの上部には検出されたエラーの説明が表示されます。また、エラーの説明の下には1つまたは複数のタブが表示されます。各タブはアプリケーション内の場所に対応するコール スタックに関連付けられています。報告されたエラーとソース情報を確認し、問題の原因を特定して修正します。

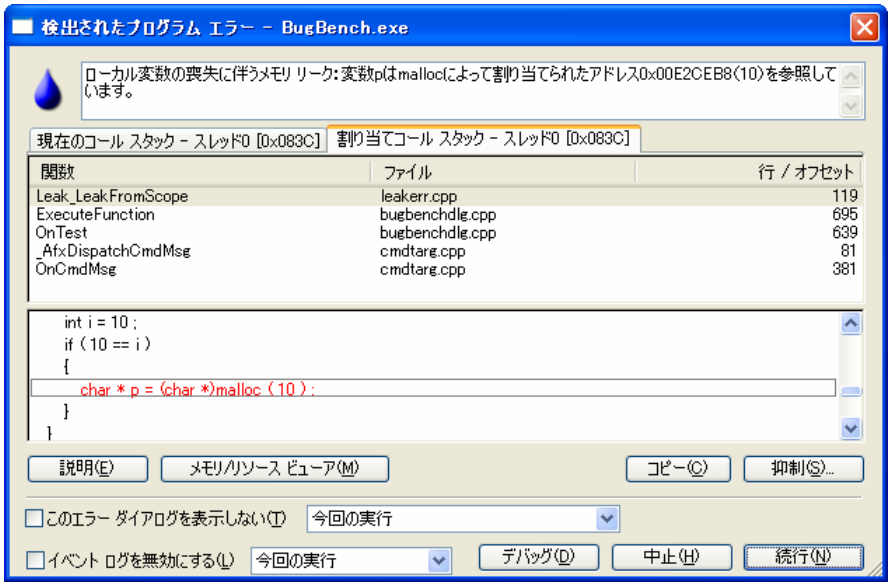


図 2-5. **[検出されたプログラム エラー]**ダイアログ ボックス

実行できる操作

[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]**ボタンがあります。DevPartner Studio Enterprise Edition を TrackRecord 統合機能と共にインストールしている場合は、**[提出]**ボタンも表示されます。

説明

[説明]ボタンをクリックすると、各エラーの詳細な説明と共にサンプルコードとその問題の解決法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」（29ページ）と『エラー検出ガイド』を参照してください。

送信

[提出]は、DevPartnerの一部としてTrackRecordがインストールされているときにのみ使用できます。**[提出]**をクリックすると、TrackRecordで不具合ページとタスクページのいずれかの新しいページが開きます。

コピー

[コピー]をクリックすると、**ソース** ペインを除くすべてのウィンドウとタブの内容がクリップボードにコピーされ、その情報を他のアプリケーションに貼り付けることができます。

抑制

[抑制]をクリックすると、ダイアログ ボックスが開き、現在のエラーを抑制できます。抑制を使用する方法と理由の詳細については、「[抑制]および[フィルタ]ダイアログ ボックス」（30ページ）と『エラー検出ガイド』を参照してください。

デバッグ

[デバッグ]は、Visual StudioまたはVisual C++で作業している場合、ダイアログボックスの下部に表示されますが、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デバッガにコードが表示されます。

中止

[中止]をクリックすると、アプリケーションが終了します。この操作でプロセスは実質的に終了しますが、アプリケーションを終了する方法は他にもあります。

続行

[続行]をクリックすると、エラーが承認され、ダイアログ ボックスが閉じ、アプリケーションの実行が続行されます。エラーはセッション ファイルに保存され、あとで**結果**ペインで確認することもできます。

[メモリおよびリソース ビューア]ダイアログ ボックス

[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 ボックスで**[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]**では、解放されていないメモリとリソース割り当てを分析することができます。

たとえば、アプリケーションの実行中にメモリの状況を判断できるメモリ分析ツールはほとんどありません。リークしたメモリまたはリソースが、アプリケーションの停止後に報告されるだけです。DevPartner エラー検出の**[メモリおよびリソース ビューア]**にはメモリとリソースの「スナップショット」機能があり、プログラム実行中の任意の時点でスナップショットを作成できます。また、現在割り当てられているメモリ ブロックやリソースを「マーク」して、プログラムの初期化後またはトランザクションの実行中に割り当てられたブロックの表示を制限することができます。

これらの機能は、以下のような場合に特に役立ちます。

- 1 日 24 時間週 7 日体制のサーバーやアプリケーションで、通常は決して終了しない場合
- アプリケーションがリソース不足でハングする可能性がある場合
- アプリケーションが、大量のメモリを消費するが、プログラム終了時に自動的にクリーン アップされる場合

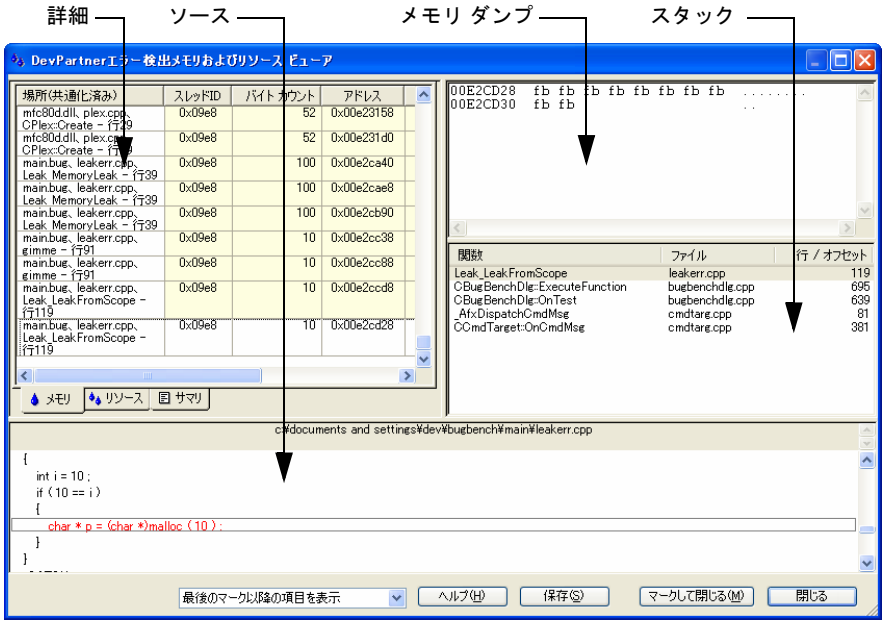


図 2-6. **[メモリおよびリソース ビューア]**ダイアログ ボックス

[メモリおよびリソース ビューア]のユーザー インターフェイスの操作

[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 ボックスで**[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 ボックスには、以下の4つのペインがあります。

- **メモリの内容ペイン**
  さまざまな形式でメモリ ブロックのダンプを表示します。リソースのダンプはありません。
- **詳細ペイン**
  [メモリ]タブ、**[リソース]**タブ、**[サマリ]**タブがあります。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]タブにはエントリのメモリ ダンプとコールスタック情報を表示し、**[リソース]**タブには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 コードがある場合、ここに表示されます。

[メモリおよびリソース ビューア]の内容の保存

[保存]をクリックすると、**[メモリおよびリソース ビューア]**ダイアログ ボックスの内容が、テキスト ファイルとして記録され、あとで参照することができます。

基準点の設定

[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]および[フィルタ]ダイアログ ボックス

DevPartnerエラー検出には**[抑制]**と**[フィルタ]**というダイアログ ボックスがあり、収集または表示するデータの量を減らすことができます。抑制または収集は、データを制限し、分析のサブセットを管理しやすくするために使用します。

たとえば、Kernel32の**`FindResourceA`**からのコール バリデーション エラーや、Kernel32のすべてのコールを抑制できます。このように指定すると、これはアプリケーション内のさまざまな選択条件に適用されます。DevPartnerエラー検出は、デフォルトで、制限が最もゆるいオプションに設定されています（図2-7を参照）。

抑制またはフィルタを適用する場合、以下の操作も可能です。

- 抑制またはフィルタが作成された理由を説明するコメントを入力する。
- 現在または将来の実行に抑制またはフィルタを適用するように選択する。
- 抑制またはフィルタの指示を再利用または共有のために保存する方法として、抑制ファイルまたはフィルタ ファイルを作成する。

エラーの抑制

エラーを抑制すると、DevPartnerエラー検出に対してそのエラーをスキップするように指示したことになります。抑制されたエラーがログに記録されたり、**[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たりすることはありません。エラーを抑制するには、以下の操作を行います。

- **[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 ウィンドウに表示される各ペインで、特定のエラーを右クリックし、**[抑制]**を選択します。

抑制ファイルの作成と保存

複数の抑制ファイルを作成できます。この機能を使用して、大規模なアプリケーションを構成するさまざまなDLLの抑制ライブラリを作成して追加できます。これによって、開発チームのメンバー間で抑制を簡単に再利用したり共有したりできます。

メモ： **はじめてDevPartnerエラー検出で.EXEを開くと、チェックする.EXEと同じディレクトリにデフォルトの抑制ファイルが作成されます。**

以降のセクションでは、DevPartnerエラー検出で抑制ファイルを作成する方法について説明します。

[抑制ファイル]ダイアログ ボックスから

[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、以下の操作を行います。

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：[DevPartner]>[エラー検出ルール]>[抑制]**を選択します。
   - ◇ **Visual C++：[DevPartner]>[エラー検出ルール]>[抑制]**を選択します。
   - ◇ **スタンドアロン：[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. 抑制ファイルに付ける名前を**[ファイル名]**テキスト ボックスに入力し、**[開く]**をクリックします。
4. **[はい]**をクリックして承認します。
   作成した抑制ファイルは、**[抑制ファイル]**ダイアログ ボックスの上部ペインにある**[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。
この時点で抑制ファイルは作成されますが、何らかの抑制を追加しないとファイルは空です（「抑制ファイルへのエントリの追加」（33ページ）を参照）。

メモ： 現在のエラー検出セッションを閉じるまで、追加した抑制は保存されません。

[抑制] ダイアログ ボックスから

[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、以下の操作を行います。

1 セッションの終了後、**[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]** の各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]** フィールドの右にある参照ボタン（**[...]**）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 抑制ファイルに付ける名前を **[ファイル名]** テキスト ボックスに入力し、**[開く]** をクリックします。

4 **[はい]** をクリックして承認します。

5 **[OK]** をクリックします。
この時点で、抑制ファイルには、ステップ1で右クリックしたエラーを抑制する指示が含まれています。

メモ： 追加した指示と、以降に追加する抑制は、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。

抑制ファイルへのエントリの追加

抑制ファイルにエントリを追加するには、以下の操作を行います。

1 **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]**のタブを選択します。

2 タブ内で特定のエラー、リーク、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 2-7（34 ページ）を参照）。

3 追加する抑制のタイプを選択します。
上部ペインにはさまざまな抑制のオプションが表示されます。選択するオプションは、選択したイベント（エラー、リーク、またはモジュール）や、検出された状況に応じて変わります。

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。

メモ： 抑制ファイルを更新するとき、特に抑制がアドレスに基づく場合やサードパーティのベンダが新規または更新ライブラリを公開した場合などに、コメントが役に立ちます。

5 この抑制を再利用するために保存するには、**[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。

- ◇ 選択しない場合、デフォルトのプログラム抑制ファイルにエントリが追加されます。
- ◇ 抑制ファイルをプログラムに追加したことがない場合、デフォルトのプログラム抑制ファイルが唯一の選択肢です。
- ◇ 抑制ファイルを別の場所に保存するには、参照ボタン（**[...]**）（**[場所]**ドロップダウン メニューの右にあります）をクリックし、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。

メモ： 現在のエラー検出セッションを閉じるまで、追加したエントリは保存されません。

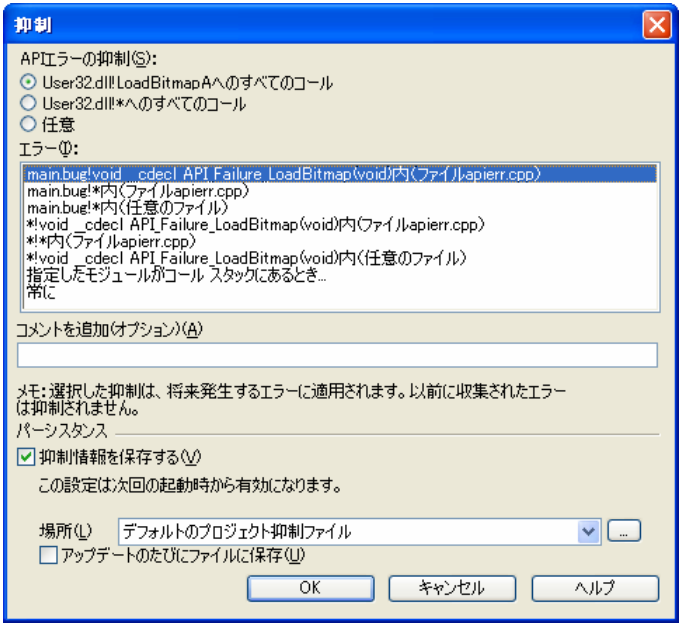


図 2-7. [抑制]ダイアログ ボックス（[フィルタ]ダイアログ ボックスも同じデザインを使用）

エラーのフィルタ

フィルタを実行すると、.DPBCL ログ ファイルに記録されているイベントを非表示にできます。これらのエラーはDevPartnerエラー検出で検出されますが、**結果**ペインのビューには表示されないか、または**[フォントと色]**で指定したとおりに表示されます。フィルタするエラーを選択するには、以下の操作を行います。

- エラー検出のメイン ウィンドウに表示される各ペインで、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 ウィンドウに表示される各ペインで、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。

フィルタの指示を削除すると、関連付けられたエラーはフィルタされなくなり、**結果**ペインに表示されるようになります。

フィルタ ファイルの作成

フィルタ ファイルを作成するには2つの方法があります。

[フィルタ ファイル] ダイアログ ボックスから

[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、以下の操作を行います。

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 ファイルに付ける名前を**[ファイル名]**テキスト ボックスに入力し、**[開く]**をクリックします。
4. **[はい]**をクリックして承認します。
   作成したフィルタ ファイルは、**[フィルタ ファイル]**ダイアログ ボックスの上部ペインにある**[使用できるフィルタ ファイル]**リストに追加されます。この時点でフィルタ ファイルは空なので、プログラム結果の表示には影響がありません。

メモ： **現在のエラー検出セッションを閉じるまで、追加したフィルタは保存されません。**

[フィルタ]ダイアログ ボックスから

[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、以下の操作を行います。

1. セッションの終了後、**[メモリ リーク]**、**[その他のリーク]**、**[エラー]**、**[モジュール]**、**[通知情報]**の各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 2-7（34ページ）を参照）。
2. **[場所]**フィールドの右にある参照ボタン（**[...]**）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 ファイルに付ける名前を**[ファイル名]**テキスト ボックスに入力し、**[開く]**をクリックします。
4. **[はい]**をクリックして承認します。
   この時点で、フィルタ ファイルには、ステップ1で選択したエラーまたはイベントを非表示にする指示が含まれています。

メモ： **追加したフィルタと、以降に追加するフィルタは、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。**

フィルタ ファイルへのエントリの追加

既存のフィルタ ファイルにエントリを追加するには、以下の操作を行います。

1 **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]**のタブを選択します。

2 タブ内で特定のエラー、リーク、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 ボックスが開きます（図 2-7（34 ページ）を参照）。

3 オプションを選択します。
オプションはダイアログ ボックスの上部ペインに表示されます。選択するオプションは、選択したイベント（エラー、リーク、またはモジュール）や、検出された状況に応じて変わります。

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。

メモ： **フィルタ ファイルを更新するとき、特にフィルタがアドレスに基づく場合やサードパーティのベンダが新規または更新ライブラリを公開した場合などに、コメントが役に立ちます。**

5 このエントリを再利用するために保存するには、**[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。

- ◇ 選択しない場合、デフォルトのプログラム フィルタ ファイルにエントリが追加されます。
- ◇ フィルタ ファイルをプログラムに追加したことがない場合、デフォルトのプログラム フィルタ ファイルが唯一の選択肢です。
- ◇ フィルタ ファイルを別の場所に保存するには、参照ボタン（**[...]**）（**[場所]**ドロップダウン メニューの右にあります）をクリックし、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。

メモ： **現在のエラー検出セッションを閉じるまで、追加したエントリは保存されません。**

フィルタされたエラーの表示と非表示

[フィルタされたエラーの表示]ツールバー アイコンをクリックすると、**結果**ペインでフィルタされたエラーの表示と非表示が切り替わります。

フィルタ エントリの削除

不要なフィルタ 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。削除するエントリを含むフィルタ ファイルを選択し、対応するチェック ボックスをオフにします。

コール バリデーション

[コール バリデーション]を有効にすると、DevPartnerエラー検出によって5,000を超えるWindows APIコールが検証されます。また、DevPartnerエラー検出で以下をはじめとして多数のイベントもチェックできます。

- ハンドルおよびポインタ エラー
- フラグ
- 範囲のチェック
- APIとメソッド エラー
- 不正な構造体のサイズ
- メモリ アクセスの失敗

フラグまたは範囲のチェックで、問題の解消とは無関係のエラーが生成されると判断した場合は、**[フラグ、範囲、および列挙引数]**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**によって、戻り値だけでなく、Windowsコールとの間でやり取りされたハンドルおよびポインタもチェックされます。

メモリ ブロック チェックの有効化

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**によって、Cランタイム ライブラリへのすべてのコールと他の多くのコールが詳細に分析されます。メモリ ブロック チェックによって全体のパフォーマンスは低下しますが、検出が困難なエラーを診断する場合には有効です。デフォルトでは、この設定はオフです。

[設定]ダイアログ ボックスの使用

ヒント: **[設定]**ダイアログ ボックスで構成ファイルの管理機能を使用して、エラーチェック パラメータのセットを構成ファイルに保存します。複数のプロジェクトで作業している場合、構成ファイルのロード、編集、作業している他のプロジェクトとの関連付けを行うことができます。

DevPartnerエラー検出の[設定]ダイアログ ボックスでは、以下を実行できます。

- 特定の問題に必要なデータ収集タイプだけを選択できます。
- 主なデータ収集タイプを部分的に有効または無効にできます。
- プログラムのどの部分を分析するかを制御できます。
- DevPartnerエラー検出のデフォルト設定を使用すると、パフォーマンスにそれほど影響を与えずにエラーを検出できます。

[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：[プログラム]>[設定]**を選択します。
- **Visual Studio：[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。
- **Visual C++：[DevPartner]>[エラー検出設定]**を選択します。

[設定]ダイアログ ボックスには、主な設定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、そのカテゴリの詳細な設定がダイアログ ボックスに表示されます。

DevPartnerエラー検出のスタンドアロン アプリケーションと統合されたVisual Studioバージョンは、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。このダイアログ ボックスの最上位のチェック ボックスをオンにすると、主なデータ収集タイプを有効または無効にできます。

最上位の各チェック ボックスの下には、DevPartnerエラー検出でのアプリケーションの分析方法を詳細に定義する他の設定項目があります。独自のエラー検出プロセスをカスタマイズするには、この設定を変更します。

たとえば、広い範囲や制限された範囲でのエラーを検出できます。

- 広い範囲－選択したデータ タイプや関連する設定が多い場合
  - 多くのエラーが検出されます。
  - 不正確な結果が出る可能性が高くなります。
  - 検出されるエラーの数が多くなるため、パフォーマンスが低下します。
  - 多数のログ ファイルが作成されます。
- 制限された範囲－選択したデータ タイプや関連する設定が少ない場合
  - 特定の機能を重点的に検出できます。
  - 検出されるエラーの数は少なくなります。
  - 特定のエラーを検出しないようにすることができます。
  - 当面の問題に関連するエラーだけを表示できます。
  - パフォーマンスが向上します。
  - 作成されるログ ファイルの数が少なくなります。

全般的なプロパティの設定

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

- **イベントをログに記録：**イベント ログを有効にするにはオンにします(DevPartnerエラー検出の他の画面からもイベント ログを有効にすることができます)。
- **エラーを表示して一時停止：[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。このダイアログ ボックスは特定のエラーに対して表示され、プログラムの実行が一時停止されます。
- **プラグラム検証結果の保存を確認する：**オンにすると、プログラムを終了する前、またはエラー検出セッションを閉じる前に、プログラムの結果を保存するように確認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：**オンにすると、テスト対象のアプリケーションを終了するときに、**[メモリ/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：**この構成に含めるソース ファイルのフル パスを指定します。

以下の設定を使用できるのは、エラー検出のスタンドアロン アプリケーションのみです。

- **シンボル パスの上書き：**この構成に含めるシンボル ファイルのフル パスを指定します。このフィールドの右にある省略ボタン（**[...]**）をクリックし、**[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 プロセスの作業ディレクトリを指定します。

メモ： **アプリケーションが起動しない場合、読み取り専用の作業ディレクトリのために問題が発生している可能性があります。一部のアプリケーションには作業ディレクトリへの書き込み権限が必要です。**

- **コマンド ライン引数：**コマンド ライン経由でアプリケーションに渡す引数を指定します。

メモ： **DevPartner エラー検出がアプリケーションの正常起動に失敗した場合、コマンド ラインの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、コマンド ラインの引数は特に重要です。**

データ収集プロパティの設定

データ収集のプログラム設定は、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：**コールのパラメータに関して収集するデータの量を指定します。小さな値にすると処理は速くなりますが、ポインタから参照される深いレベルの詳細は報告されません。大きな値にすると深いレベルのコールの詳細が報告されますが、処理速度は低下し、ログ ファイルのサイズも増えます。
- **メモリ割り当ての最大コール スタック数：**各割り当てについて、追跡するコール スタックの最大数を指定します。割り当てはひんぱんに実行され、エラーの原因になることはあまりないので、大きな値を選択するとパフォーマンスが低下する可能性があります。また、大きな値を選択すると、テスト対象のアプリケーションで、エラー検出のメモリ使用量が大幅に増加する可能性があります。
- **エラーの最大コール スタック数：**報告されたエラーについて検証するコール スタックの最大数を指定します。この値は必要に応じた高い値に設定できます。その場合でも、ログ ファイルを保存するための空き容量が十分であれば、パフォーマンスに悪影響が及ぶことはありません。
- **NLB ファイル ディレクトリ：**(必須フィールド）生成された NLB（最適化されたタイプ ライブラリ）ファイルを保存する場所を選択します。プロジェクト ファイルと NLB ファイルの削除を簡単に行えるように、通常はプロジェクトと同じ場所にします。存在しないディレクトリを指定すると、アプリケーションの実行時に有効なディレクトリを選択するように確認メッセージが表示されます。また、**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ディレクトリを指定することもできます。

メモ： **NLB ファイルには、エラー検出に必要なすべての API の説明ファイル情報が含まれます。**

API コール レポーティング プロパティの設定

ヒント： プログラムでチェックする必要がないAPIコールを行っていることがわかっている場合、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の向上に役立ちます。

ヒント： **[ウィンドウ メッセージを収集する]**を選択すると、ログ ファイルのサイズが大幅に増えます。最適な結果を得るには、ウィンドウ メッセージの問題をデバッグする場合にのみ、この機能を選択します。

アプリケーションから行ったシステム関数のコール、パラメータ、戻り値を記録するには、API コール レポーティングを使用します。DevPartner のエラー検出では、**データ収集**設定で指定した**データ表示の深さ**の設定に基づいて、戻り値とパラメータの構造情報が記録されます。

API コール レポーティングを有効にして、API 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。以下の設定で、エラー検出に合わせて API のログを制御します。

- **ウィンドウ メッセージを収集する：** API ログの一部としてウィンドウ コントロール メッセージを収集するには、オンにします。
- **API メソッドのコールとリターンを収集する：** API モジュールのツリー ビューで選択したモジュールについて、API メソッドのコールとリターンを収集するには、オンにします。
- **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：** ツリー ビューでプログラムに必要な API モジュールのみを表示するには、このチェック ボックスをオンにします。このツリー ビューを展開し、すべての API モジュールを表示するには、チェック ボックスをオフにします。
- **API モジュールのツリー 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。項目の横にあるプラス（+）記号をクリックすると、その項目に含まれる関数が表示されます。各項目の横にあるチェック ボックスをクリックすると、特定のモジュールまたは関数を API ログの対象として選択できます。

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える可能性があります。ログ ファイル サイズを最小限に抑えるには、アプリケーションの選択した部分についてのみ、コール レポーティング データ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方法をいくつか紹介します。

- **[モジュール]** ツリー ビューのチェック ボックスをクリックして、チェックする必要がない API モジュールの選択を解除します。
- **[モジュールとファイル]** を使用してログの範囲を制限します。
- アプリケーションに対するイベント ログを有効または無効にする API コールを追加します。`NmApiLib.h` に記載されているコメントを参照してください。このファイルは DevPartner ソフトウェアのインストール ファイルに含まれ、DevPartner エラー検出からエクスポートされるイベント レポートの API が定義されています。
- イベント ログをオフにします。

[コール バリデーション]オプションの設定

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 システムのライブラリへのコールと、COM メソッドのコールが監視されます。渡されるパラメータが検証され、コールから返された値が正常な終了を示すかどうかがチェックされます。以下の要素で、エラー検出のコール バリデーションの動作を制御します。

- **コール バリデーションを有効にする：**コール バリデーション コンポーネントを有効にするには、このチェック ボックスをオンにします。
- **メモリ ブロック チェックを有効にする：**メモリを参照するパラメータに対してより広範囲なメモリ チェックの検証を有効にするには、このチェック ボックスをオンにします。**[プログラム設定]**ツリー ビューの**[メモリの追跡]**にある**[メモリの追跡を有効にする]**をオンにするまで、この機能はアクティブになりません。

メモ： **[メモリ ブロック チェックを有効にする]**をオンにすると、より広範囲のチェックが実行されます。それによって結果がより正確になり、多くのバグを検出できることがあります。この機能を有効にするとセッションの終了までにかかる時間が長くなります。

- **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：[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で出力引数に値を入力するには、このチェック ボックスをオンにします。
- **COM 失敗コード：**COM メソッドのすべての戻り値をチェックするには、このチェック ボックスをオンにします。

ヒント：通常の使用方法では、多くの COM メソッドによって "Not Implemented" エラーが報告されます。このチェックを無効にすると、報告されるエラー数が大幅に減る可能性があります。

- **COM のリターン コード "Not Implemented" をチェックする：**`HRESULT E_NOTIMPL`（"Not Implemented"）のリターン コードのチェックを有効にするには、このチェック ボックスをオンにします。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)]**で選択した 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 失敗コード:** 選択した DLL にある API からの戻り値のチェックを有効にするには、このチェック ボックスをオンにします。
- **不正なパラメータ エラーのチェック（COM または API）：**エラー検出がサポートする選択した DLL または COM インターフェイスの API に対して、引数（パラメータ）のチェックを有効にするには、以下のチェック ボックスの一方または両方をオンにします。

ヒント：パフォーマンスを向上させ、エラーの報告数を減らすには、必要な場合にのみこの機能をオンにします。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])[不正なパラメータ エラーのチェック（COM または API）]**の一方または両方をオンにすると使用可能になります。引数のタイプに応じて引数のチェックを有効にするには、チェック ボックスの一方または両方をオンにします。
- **静的リンクされた C ランタイム ライブラリ API をチェックする：[API 失敗コード]**または**[不正なパラメータ エラーのチェック：API]**をオンにすると、使用可能になります。. 静的 C ランタイム コールのチェックを有効にするには、このチェック ボックスをオンにします。静的 C ランタイム ライブラリを使用していない場合、このチェック ボックスをオフにして、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、不要なエラーを大幅に減らすことができます。また、パフォーマンスも上がります。

- **APIエラーをチェックするDLL(失敗または不正な引数): [API失敗コード]** または **[不正なパラメータ エラーのチェック : API]** をオンにすると、使用可能になります。表示されている DLL に含まれる API の引数と戻り値のチェックを有効にするには、このチェック ボックスをオンにします。

メモ: アプリケーションが使用している DLL や DLL 内の API を検出するには、ツール（Visual Studio に付属する Depends など）を使用できます。

API コールに対するメモリ上書きの検出の有効化

API コール（`strcpy` など）によるメモリ ブロックの破損のチェックは、デフォルトで無効になっています。API コールによるメモリ上書きの検出を有効にするには、以下の操作を行います。

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 プロパティの設定

[すべてのインターフェイス] ツリーで選択した COM インターフェイスに対するコールとその戻り値を記録するには、COM コール レポーティングを使用します。DevPartner エラー検出では、パラメータ値および返された `HRESULT` が記録されます。

ヒント: チェックする必要があるインターフェイスのみを選択します。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小さくなり、パフォーマンスも向上します。

COM コール レポーティングを有効にし、COM インターフェイスのリストをアクティブにするには、**[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。COM コール レポーティングを構成するには、以下のコントロールを使用します。

- **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 : [すべてのインターフェイス]** ツリーに表示されていないインターフェイスについて、COM メソッド コールと戻り値を報告するようにエラー検出を構成するには、このチェック ボックスをオンにします。
- **[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]** エントリの横にあるプラス（+）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。各項目の横にあるチェック ボックスをクリックすると、COM コール レポーティングを行う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] オプションの設定

COM オブジェクトの追跡を使用すると、プログラムのリーク COM オブジェクトを監視できます。オブジェクト リークは、**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、オブジェクトに関して `AddRef()` と `Release()` のコールを確認できるので、`Release()` のコールがない場所を特定できます。

ヒント：パフォーマンスを上げるには、[すべてのCOMクラス]の一部を選択します。すべてのCOMクラスを選択するのは、アプリケーションをはじめて実行するときと、最後のQAで実行するときだけにするようにしてください。

COMオブジェクトの追跡を有効にし、**[すべてのCOMクラス]**ツリー ビューをアクティブにするには、**[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、監視するCOMクラスを選択します。アプリケーションのCOMクラスが表示されない場合、**[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。

メモ： **一部のCOMクラスを選択する場合、ほとんどのベンダが共通の接頭辞をオブジェクト名に付けていることに注意してください。**

[デッドロック分析]オプションの設定

デッドロック分析を使用すると、マルチスレッド アプリケーションのデッドロックを監視できます。これには以下のような分析が含まれます。

- アプリケーション内で発生するデッドロックを監視し、レポートする
- アプリケーション内での同期オブジェクトの使用パターンを監視し、潜在的なデッドロックを検出する

デッドロック分析を有効にし、他のデッドロック分析コントロールをアクティブにするには、**[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。

以下の設定によって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すべての名前付き同期オブジェクトは、そのプロセス内でのみ使用されるとみなされます。名前付き同期オブジェクトと関連付け付けられたデッドロックの検出ルールの一部を緩和するには、このチェック ボックスをオフにします。
- **ウォッチャー スレッドを有効にする：**アプリケーションにウオッチャー スレッドを作成して、ローカライズされたデッドロックについて監視するには、このチェック ボックスをオンにします。デフォルトでこの機能は無効になっているので、アプリケーションに干渉が発生することはありません。

  アプリケーションが応答しなくなり、デッドロックしたように考えられる場合、この機能を有効にすると、アプリケーションの詳細な分析を実行できます。

メモ： **プロセス内で余計なスレッドが検出されないように、複雑なDLL_THREAD_ATTACHロジックを作成する場合、このオプションを有効にする必要はありません。**

- **エラーを生成するとき：**以下の選択を使用して、エラー検出でデッドロック エラーを報告するタイミングを指定します。
  - **クリティカル セクションに再入したとき：**スレッドにすでに含まれるクリティカル セクションに再入するときに警告を生成する場合に選択します。クリティカル セクションへの再入はエラーではありませんが、アプリケーションのクリティカル セクションへの入りと出は必ず同じ回数発生する必要があります。

◇ **所有するミューテックスに待機が要求されたとき：**スレッドにすでに含まれるミューテックスで待機するときに警告を生成する場合に選択します。

◇ **リソースごとの過去のイベント数：**エラーまたは待機で報告する、同期オブジェクトごとに記録されるコール スタックの数を入力します。

各同期オブジェクトに関連付けられているスタック情報によって、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。これによって、デッドロック状況をデバッグできます。

メモ： **同期オブジェクトごとに維持するコール スタックの数が増えると、アプリケーションのメモリ使用量が増え、アプリケーションのパフォーマンスに影響があります。**

◇ **同期APIタイムアウトをレポート：**待機が正常終了せずに同期オブジェクトの待機がタイムアウトしたときに、エラーを報告する場合に選択します。

すべてのWindowsコールについてAPIコール レポーティングを有効にせずに、同期オブジェクトのAPIエラーを監視するには、このオプションを有効にします。

ヒント：[待機制限または実際の超過時間をレポート]機能を使用すると、アプリケーション内の最長待機ポリシーを指定できます。

◇ **待機制限または実際の超過時間（秒）をレポート：**レポートのタイムアウトを選択したあとにアクティブになります。エラー検出では、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がチェックされます。タイムアウト値がここで指定した制限を超える場合、コールがエラーとして報告されます。

メモ： **`INFINITE`と指定された待機には、エラーのフラグは付きません。**

◆ **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。

◇ **リソースの命名に関する警告を表示しない：**選択すると、アプリケーションに含まれる名前付きリソースと名前なしリソースについて警告は表示されません。

ヒント：セキュリティ監査を実行している場合、予期しない名前付きリソースがプロセスの外部で見えるかどうかを判断するには、**[命名されたリソースに関する警告を表示する]**機能を有効にしてみてください。名前付きリソースはプロセスの外部でも見ることができるため、不正利用を防ぐには、適切なセキュリティを適用する必要があります。

◇ **命名されたリソースに関する警告を表示する：**アプリケーションに含まれる名前付き同期リソースごとに警告を生成する場合に選択します。このチェックを使用すると、アプリケーションの外部で操作できる名前付きリソースを特定できます。

◇ **名前のないリソースに関する警告を表示する：**アプリケーションに含まれる名前なし同期リソースごとに警告を生成する場合に選択します。このチェックを使用すると、名前なしリソース（他のプロセスで使用するため、または企業のネーミング ルールに合わせるために、命名が必要なリソースなど）を検出できます。

メモ： **デフォルトで、プログラムの実行中に検出された名前付きリソースと名前なしリソースのどちらについても、警告は表示されません。**

[メモリの追跡]オプションの設定

メモリの追跡を有効にすると、DevPartnerエラー検出によって以下が実行されます。

- アプリケーションでメモリの割り当てと解放を行うすべてのコールが監視されます。
- アプリケーションの終了時に解放されないメモリについて報告されます。

さらに、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を有効にする]**をオンにした場合、エラー検出では以下が実行されます。

- メモリの割り当て済みブロックに対する最後の参照がスコープから外れているインスタンスが記録されます。
- 実行中のステートメント レベルのメモリ エラーとポインタ エラーが報告されます。

メモリの追跡を有効にし、すべてのメモリの追跡オプションをアクティブにするには、**[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。

メモ： [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、その前に[メモリの追跡を有効にする]チェック ボックスを選択する必要があります。

以下の設定によってメモリの分析の動作を制御します。

- **リーク分析のみを有効にする**：リークの監視を除き、メモリの追跡のすべてを無効にするには、このチェック ボックスをオンにします。これにより、オーバーラン、初期化されていないメモリの使用、またはダングリング ポインタは追跡されません。また、システム モジュールから割り当てたメモリがメモリの追跡で評価されないため、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。

メモ： この機能を有効にした場合、COMインターフェイスのフックの一部は完全には処理されません。

- **FinalCheckを有効にする**：FinalCheckを有効にするには、このチェック ボックスをオンにします。オンにすると、FinalCheckによってインストゥルメントされたモジュールについて、追加のチェックが実行されます。オフにすると、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**：サブアロケーションに使用されているブロック上のリークの報告を有効にするには、このチェック ボックスをオンにします。サブアロケーションに使用されたブロックは、通常、`malloc`または`new`などの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ている場合、`malloc`または`new`などの関数から返されるブロックにサブアロケートしたバッファを含め、この機能でアプリケーションのすべてのメモリを監視できるようにします。カスタム メモリ アロケータは、`UserAllocators.dat`にリストしないと監視されません。`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」を参照してください。
- **厳密な再割り当てセマンティクスを強制する**：セマンティクスの厳密な適用を有効にするには、このチェック ボックスをオンにします。厳密な再割り当てセマンティクスを強制している場合、再割り当てされたメモリへのポインタはダングリング ポインタのように扱われ、そのポインタを使用すると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を有効にしていない場合、再割り当てされたポインタは新しいポインタと同じメモリ位置を指しているかぎり使用でき、エラーは生成されません。次に例を示します。

```
char *ptrA = (char *) malloc(17);
// ptrA は 17 バイトのメモリを指しており、これは有効です。
char *ptrB = (char *) realloc(ptrA, 15);
// ptrB は 15 バイトのメモリを指しており、これは有効です。
// 厳密なセマンティクスを適用すると、値に関係なく ptrA は無効なポインタになります。
// 厳密なセマンティクスを適用しなければ、ptrA は ptrB と同じであるかぎり有効です。
```

- **保護バイトを有効にする：** 有効にすると、割り当てられたメモリ ブロックの末尾に保護バイトが挿入され、メモリ オーバーラン エラーが検出されます。オーバーランによってヒープまたはスタックの破損が発生することがあり、さらにランダムなクラッシュや予期しないデータの上書きにつながる可能性があります。
  - **パターン：** 16進の保護バイト パターンを入力します。このパターンは、割り当てられたブロックがオーバーランされたかどうかを判断するために使用されます。
  - **カウント：** 使用する保護バイト数を選択します。ランダムなヒープの破損エラーが発生してもヒープのオーバーラン エラーが報告されていない場合、保護バイト数を増やしてみてください。それによってメモリの使用量は増えますが、検出が困難なヒープの破損エラーが見つかることがあります。
- **実行時のヒープ ブロック チェック：** ヒープ全体をチェックして、保護バイトが上書きされたかどうかを確認する頻度を指定します。各ブロックは、解放時に必ずオーバーランについてチェックされます。追加のチェックについては3つのオプションがあります。
  - **解放時**
  - **適応分析の使用**
  - **すべてのメモリ API コール時**
- **確保時にフィルする：** 有効にすると、メモリの割り当て時に指定したフィル パターンが適用されます。
  - **パターン：** 使用する16進のフィル パターンを指定します。
- **未初期化メモリをチェックする：** 選択すると、新しく割り当てられたメモリは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がチェックされます。
  - **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。不正なエラー レポートの数を減らすには、この値を増やします。
- **解放時に無効データをフィルする：** メモリの解放時に無効データをフィルするには、このチェック ボックスをオンにします。
  - **パターン：** 無効データをフィルするメモリ位置へ書き込むパターンを入力します。

[.NET Framework 分析]オプションの設定

アンマネージ コード、マネージ コード、アンマネージ リソースが混在しているアプリケーションを開発する場合、.NET Framework 分析を使用します。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は、パフォーマンスが低下する可能性があります。この分析で収集されるデータは、低下の範囲や重大度を評価するときに役立ちます。問題が見つかってもすべての問題を修正する時間がない場合、この分析によって最も重大度が高い問題を判断できます。

このパネルで.NET Framework 分析のコントロールを有効にするには、**[.NET 分析を有効にする]**チェック ボックスをオンにします。.NET Framework 分析を構成するには、以下のコントロールを使用します。

- **例外の監視：**アンマネージ（レガシ）コードがハンドルされない例外をスローし、それがマネージ コードに渡されるようなインスタンスを監視するには、このチェック ボックスをオンにします。

メモ： **アンマネージ コードからマネージ コードへ渡される例外はエラーの原因になることがよくあります。これは、必要なハンドルがアンマネージ コードに存在しなくなるためです。通知される例外についてはよく確認してください。考えられるエラーとして、部分的に初期化されたデータ構造、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：**適切な dispose メソッドの呼び出しエラー（リーク）や、アンマネージ リソースをカプセル化するクラスの不適切な実装など、アンマネージ リソースの不正使用を監視するには、このチェック ボックスをオンにします。
- **COM 相互運用性の監視：**マネージ コードとアンマネージ（レガシ）コードの間で移行を発生させているクラス ID を監視するには、このチェック ボックスをオンにします。この機能によって、使用されているインターフェイス ID も特定されます。

メモ： **COM 相互運用性の監視は、ひんぱんに呼び出されているメソッドを判断するときにも使用できます。メソッドが何度も呼び出されていることがわかった場合、移行を回避するようにオブジェクトを移植してみてください。書き換えが選択肢にない場合、移行の数を減らすために、データをひとかたまりで転送する新しいメソッドの追加を検討してください。**

- **PInvoke 相互運用性の監視：**アンマネージ（レガシ）コードが呼び出される（DLL と、可能な場合は API 別の）回数をカウントするには、このチェック ボックスをオンにします。オンにすると、アプリケーションがアンマネージ（レガシ）コードに移行する理由を判断できます。

メモ： **PInvoke 相互運用性の監視によって、アプリケーションによる PInvoke の呼び出し回数がわかります。また、PInvoke 相互運用性の監視レポートを使って、マネージからアンマネージへの移行を監視できます。リストを確認して、過度の呼び出しが行われているかどうかを判断してください。**

- **相互運用性レポートのしきい値：***x*はこのフィールドで指定する値という前提です。アプリケーションが *call_A* を行う回数が *x* 以上の場合、*call_A* を .NET 分析の結果に追加します。これによって、限定された回数のみ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の数が増えます。

メモ： 相互運用性レポートのしきい値を指定することで、COM 相互運用性監視レポートとPInvoke相互運用性監視レポートからCOMの移行を除外できます。移行回数が指定した値以上の場合だけ、移行が報告されます。

.NET Framework コール レポーティング プロパティの設定

ヒント： .NET Framework コール レポーティングによって大量のデータが生成され、システムがシャットダウンする可能性があります。.NET Framework コール レポーティングは、.NET Framework のデバッグや把握が必要な場合にのみ有効にし、その場合でもチェックが必要なアセンブリのみを選択するようにしてください。**[すべてのタイプ]** ツリー ビューで選択するアセンブリ数を制限すると、ログ ファイルのサイズが減り、パフォーマンスが向上します。

.NETインターフェイスの呼び出し、または.NETインターフェイスからの戻り値を記録するには、.NET Framework コール レポーティングを使用します。アセンブリのNLBファイルが見つかった場所に基づき、.NETモジュールについて「ユーザー アセンブリ」と「システム アセンブリ」が区別されます。

.NET Framework コール レポーティングを有効にし、.NETアセンブリのリストをアクティブにするには、**[.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ている.NETアセンブリが表示されます。**[すべてのタイプ]** エントリの横にあるプラス (+) 記号をクリックすると、ツリーが展開されます。ツリーには、.NETユーザー アセンブリと.NETシステム アセンブリの両方について分岐があります。各項目の横にあるチェックボックスをクリックすると、.NET Framework コール レポーティングを行うアセンブリを選択できます。

[リソースの追跡]オプションの設定

リソースの追跡を有効にすると、エラー検出によって以下が実行されます。

- アプリケーションで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が監視されます。
- アプリケーションの終了時に解放されなかったリソースが報告されます。

リソースの追跡を有効にし、リソースのリストをアクティブにするには、**[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。対応するリストでチェックボックスをオンにすると、そのDLLによって作成されたリソースが追跡されます。

リソースの追跡をさらに調整して特定のリソースに限定するには、1つまたは複数のリソースの割り当て解除APIから選択します。たとえば、すべてのレジストリ関連のリソースを除外するには、**ADVAPI.DLL** リソースの下の **[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]オプションの設定

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

メモ： **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がありません。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のみ、インストゥルメンテーションを制限できます。**

ヒント： 場合によっては、評価から除外できるように、このリストにモジュールを明示的に追加する必要があります。DevPartner エラー検出では、**[モジュールとファイル]**に表示されていないモジュールが自動的に含まれます。表示されないモジュールを除外する場合、そのモジュールを追加してから、チェック ボックスをオフにして除外する必要があります。

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]の設定を使用すると、以下のことが可能です。

- モジュールを評価から除外する
- モジュール内のコンポーネントを評価から除外する
- 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]リスト：**チェック対象のモジュールが表示されます。
  - モジュール全体をチェック対象から除外するには、そのモジュールの横にあるチェック 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項目を除外するには、モジュールを展開し、除外する項目の横にあるチェック ボックスをオフにします。

メモ： **モジュールまたはモジュール内の項目の横にあるチェック ボックスをオフにすると、リストには表示されますが、アプリケーションでエラー検出を実行しても分析されません。**

1つまたは複数のコンポーネントがオフになっている場合、モジュール名の横にあるチェック ボックスが黄色で表示されます。

[モジュールとファイル]の設定ですべてのモジュールを無効にしても、一部のエラー タイプのレポートは回避されません。モジュール内のメモリ オーバーランと、`MFCxxxx.dll` ライブラリが原因の他の種類のイベントは常に報告されます。

- **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：**報告するリークとエラーをソース コードが使用可能なものに制限するには、このチェック ボックスをオンにします。このオプションを有効にすると、報告されるリーク数とエラー数が減ることがあります。無効にすると（デフォルト）、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 ボックスが開きます。このダイアログ ボックスを使って、モジュールを選択して追加します。

- **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。モジュールが選択されている場合にのみアクティブになります。

メモ： **メインの実行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]オプションの設定

チェックの必要がないディレクトリ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生成するモジュールがディレクトリに含まれている場合があります。チェックする必要がないディレクトリを除外すると、エラー検出セッションの速度が向上します。

メモ： **DevPartner エラー検出では、原因が特定されていないエラーと、アプリケーションの壊滅的な障害を引き起こすすべてのエラーが報告されます。このようなエラーは、除外したディレクトリ内のモジュールで発生した場合でも報告されます。**

[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が使用できます。

- **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。エラー検出のチェックから除外するディレクトリのリストに追加するディレクトリを選択するには、このダイアログ ボックスを使用します。
- **削除：**クリックすると、選択したシステム ディレクトリがリストから削除されます。
- **OK:** クリックすると**[システム ディレクトリ]**ダイアログ ボックスが閉じ、変更内容が保存されます。
- **キャンセル：**クリックすると**[システム ディレクトリ]**ダイアログ ボックスが閉じ、変更内容が破棄されます。

ディレクトリ アイコン

[システム ディレクトリ]リストの各パスの横にあるディレクトリ アイコンは、2つの異なる条件を示します。

- **単一のディレクトリ：**単一のフォルダ アイコンで示されます。選択したディレクトリの直下にあるコンテンツのみが含まれます。
- **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 アイコンで示されます。選択したディレクトリとすべてのサブディレクトリが含まれます。

2つのオプションを切り替えるには、ディレクトリ名の横にあるアイコンをクリックします。

メモ： **場合によっては、除外されるディレクトリに含まれる重要なサードパーティ製 DLL を明示的に追加する必要があります。サードパーティ製 DLL を明示的に追加することで、特定できなかった問題が判明することがあります。明示的に DLL を追加するには、[モジュールとファイル]の設定を使用します。**

[フォントと色]オプションの設定

[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、よく表示するエラー データのフォント サイズを大きくしたり、タブに表示する情報量を増やすためにフォント サイズを小さくしたりすることができます。

フォントと色を定義するには、以下のコントロールを使用します。

- **以下の設定を表示：結果**ペインに表示される各タブが表示されます。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、現在の設定がすべて破棄され、最初のフォントと色が復元されます。
- **表示可能な項目：**フォントと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ついて、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ついて、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について、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について、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、表示可能な項目のテキストが太字で表示されます。
- **タブのサイズ：ソース コード** ペインに表示するコードのインデント サイズを指定するには、このコントロールを使用します。

メモ： **このコントロールを使用できるのは、[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]の場合のみです。**

- **サンプルのテキスト 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 ボックスには、選択したフォントと色の組み合わせで現在の表示可能な項目が表示されます。

[構成ファイル管理]オプションの設定

構成ファイルを管理するには、[構成ファイル管理]設定を使用します。**[プログラム設定]**ダイアログ ボックスのタイトル バーには、現在使用されている構成ファイルが表示されます。

メモ： **[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、構成ファイル名のあとにアスタリスクが表示されます。プロパティの保存、ファイルの再ロード、または別のファイルのロードを行うと、アスタリスクは消えます。保存せずに別のファイルのロードやファイルの再ロードを行うと、現在のファイルに加えた変更は失われます。**

構成ファイルの機能を定義するには、以下のコントロールを使用します。

- **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：**現在の構成ファイルをロードし直します。変更は破棄されます。これにより、現在の構成ファイルの前回保存されたバージョンに戻ります。
- **ロード：[ロード元]**ダイアログ ボックスが開きます。
  - **[内部ユーザー デフォルト]**を選択して、ユーザー デフォルト設定をロードします。
  - **[構成ファイル]**を選択した場合、**[構成ファイルのロード]**ダイアログが開きます。このダイアログを使用して、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に加えられたすべての変更を保存します。
- **名前を付けて保存: [構成ファイルの保存]**ダイアログ ボックスが開きます。このダイアログ ボックスを使用して、現在の構成設定を別のファイル名で保存します。
- **リセット：**すべてのプログラム プロパティ設定を工場出荷時のデフォルト設定にリセットします。
- **デフォルトの保存：**現在の設定をユーザー デフォルトとして保存します。すべての新規プロジェクトで、これらの設定が使用されるようになります。
- **デフォルトの削除：**ユーザー デフォルト構成設定を削除し、工場出荷時の設定に戻します。すべての新規プロジェクトで工場出荷時の設定が使用されるようになります。

Windows メッセージとイベント ログの追跡

Windows はイベントドリブンの環境であるため、ほとんどのプログラムは Windows のメッセージやイベントに対応して実行されます。イベントが発生すると、DevPartner エラー検出によってイベントがインターセプトされ、ログに記録されます。これにより、イベントのすべての履歴を表示して、どのイベントによってエラーが発生したかを確認できます。

DevPartner エラー検出によって、以下のイベントが記録されます。

- Windows メッセージ。
  これらのイベントを参照すると、プログラムが Windows のメッセージにどのように反応したかがわかります。
- API コールと API リターン（引数の情報を含む）。
  これらのイベントを参照すると、プログラムのどの部分が実行されていたのかがわかります。
- チェック中のプログラムで表示されるデバッグ文字列メッセージ。
- エラー メッセージ。

データの XML へのエクスポート

DevPartner を使用すると、包括的なセッションの結果データを XML にエクスポートできます。そのため、結果データをレポート形式、電子メール、社内の Web ページなどへ簡単にコピーできます。

Visual Studio 内からのデータのエクスポート

Visual Studio 内からエラー検出を使用して、現在表示されているセッション ファイルの全データを XML ファイルにエクスポートするには、以下の操作を行います。

1. エラー検出のセッション 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。

エラー検出のスタンドアロン アプリケーションからのデータのエクスポート

エラー検出のスタンドアロン アプリケーションを使用して、現在表示されているセッション ファイルの全データを XML ファイルにエクスポートするには、以下の操作を行います。

1. エラー検出のセッション 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 ラインからのデータのエクスポート

コマンド ラインからエラー検出を実行し、セッション ファイル データから XML ファイルを生成するには、実行時に `BC.exe` に適切なフラグを渡すか、`BC.exe` を呼び出して既存のセッション ファイルを指定します。

DevPartner エラー検出では、XML の出力ファイルを生成するときに、エラー検出のインストール ディレクトリにある `DPSErrorDetection.xsd` スキーマ ファイルを使用します。このファイルは編集しないでください。

セッション データの XML へのエクスポートに失敗した場合、発生した問題を説明するエラー メッセージが生成されます。

セッションの実行とデータのエクスポート

実行ファイルを指定すると、そのファイルに対してエラー検出のセッションが実行され、結果からXMLの出力ファイルが生成されます。

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

/XにSまたはDのフラグを使用すると、サマリ情報または詳細情報をXMLにエクスポートできます。

メモ： **実行ファイルを指定する場合には、/Bフラグを使用して、それに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。**

既存のファイルの変換

セッション ファイル（*session*.DPbcl）のみを指定すると、指定したセッション ファイルがXMLに変換され、出力ファイルが保存されます。

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

コマンド ラインからのエラー検出の実行

DevPartnerエラー検出をDOSコマンド ラインから実行する場合は、**bc.exe**または**bc.com**を使用します。

メモ： **従来の7.xバージョンのDevPartnerエラー検出機能では、スクリプト ファイルで継続してbc7.comを使用できます。**

- **bc.exe**は、DevPartnerエラー検出スタンドアロンのユーザー インターフェイスを起動します。
- **bc.com**は、**bc.exe**を実行し、これが終了するまで待機する小さなコンソール プログラムです。

bc.exeと**bc.com**の違いは、バッチ スクリプトで重要となります。**bc.exe**を直接起動すると、DevPartnerエラー検出が起動し、**bc.exe**の終了を待たずに次のコマンドを継続して実行します。スクリプトの次のステップが結果を確認するものである場合、そのステップは無効になります。

メモ： **「bc」とだけ入力すると、OSによって**bc.exe**ではなく**bc.com**が選択されます。**

詳細については、『エラー検出ガイド』の コマンド ラインからのエラー検出の使用を参照してください。

コマンド ラインのオプションと構文

角かっこ [ ] は、コマンドが任意選択であることを示します。

```
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
/?	使用法情報を表示します。
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、セッション ファイルをログ ファイル *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、指定したファイルに保存します。 実行ファイルを指定すると、そのファイルに対してエラー検出のセッションが実行され、結果から XML の出力ファイルが生成されます。 セッション ファイル（*session*.DPbcl）のみを指定すると、指定したセッション ファイルが XML に変換され、出力ファイルが保存されます。 **メモ：** 実行ファイルを指定する場合には、/B スイッチを使用して、それに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 フラグに S 修飾子を使用すると、サマリ データのみが 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）は常にエクスポートされます。
/XD *xmlfile*.xml	/X フラグに D 修飾子を使用すると、詳細データのみが 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）は常に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。エラー時に **[検出されたプログラム エラー]** ダイアログ ボックスが開きません。
/W *workingdirectory*	ターゲットの作業ディレクトリを設定します。
target.exe [*args*]	起動する実行可能ファイルとその引数です。

メモ： 使用する実行可能プログラムが現在のパスにない場合は、ディレクトリのフル 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ディレクトリを一覧にする環境変数）。

コマンド ラインからの FinalCheck の実行

FinalCheck は、コマンド ラインから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 でのプログラムのチェック」セクションの以下のトピックを参照してください。

- コマンド ラインからの FinalCheck の実行
- NMCL オプション
- NMLINK オプション

Visual Studio Team System へのデータの送信

Microsoft Visual Studio Team Explorer クライアントがインストールされ、Team Foundation Server の接続が使用可能になっている場合に、DevPartner Studio は Microsoft Visual Studio Team System をサポートします。

DevPartner エラー検出の Visual Studio Team System サポート

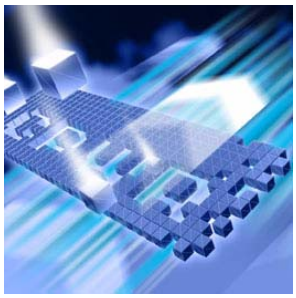
以下のエラー検出タブで選択した項目のデータを、**バグ** タイプの**作業項目**として送信できます。

- [エラー] タブー選択したエラーを送信します。
- [メモリ リーク] タブー選択したリークを送信します。
- [モジュール] タブー選択したインスタンスを送信します。
- [その他のリーク] タブー選択したリークを送信します。
- [.NET パフォーマンス] タブー選択したインスタンスを送信します。

バグを送信すると、タブのデータが**作業項目**フォームにコピーされます。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」（8ページ）を参照してください。

第 3 章

静的なコード分析



- コード レビューの機能
- すぐにコード レビューを使用するには
- 設定オプション
- ルールの抑制
- サマリ データの表示
- コード違反の表示
- ネーミング違反の表示
- 収集したメトリクスの表示
- コール グラフ データの表示
- コマンド ライン インターフェイスの使用
- データの XML へのエクスポート
- ネーミング分析
- コード レビュー ルール マネージャの使用
- 正規表現を使用した新しいルールの作成
- Visual Studio Team System へのデータの送信

この章には 2 つのセクションがあります。1 つめのセクションには、はじめてのユーザーがコード レビュー機能を利用できるように、簡単な操作手順が記載されています。2 つめのセクションには、DevPartner Studio のコード レビュー機能を詳しく理解するための参考情報が記載されています。

コード レビューに関するその他のタスクに基づく情報については、DevPartner Studio のオンライン ヘルプを参照してください。

コード レビューの機能

DevPartner コード レビューは、Visual Studio においてベスト プラクティスに準拠した Visual Basic コードおよび Visual C# コードを開発することを支援します。DevPartner コード レビューには、プログラミングとネーミングの違反の識別、メソッド コール構造の分析、コード全体の複雑度の追跡を行う機能があります。

メモ： **コード レビュー機能で分析されるのはマネージ コードのみなので、DevPartner for Visual C++ BoundsChecker Suite ではサポートされていません。**

DevPartner コード レビュー機能には、以下の機能が備わっています。

- 静的なコード分析とレビュー

  DevPartner コード レビューを使用すると、Visual Studio のソース コードについて広範な静的コード分析が実行され、**[DevPartner コード レビュー] ウィンドウ**にその結果が表示されます。

- 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。この自動バッチ レビューは夜間のビルドと組み合わせて実行できます。また、他のタスクを実行しながら、大規模なアプリケーションについて自動バッチ レビューを使用できるので、時間を節約できます。

- XML へのデータ エクスポート

  DevPartner コード レビューを使用すると、セッションの結果を XML 形式にエクスポートできます。そのため、結果データをレポート形式、電子メール、社内の Web ページなどへ簡単に変換できます。コード レビューから XML へのデータのエクスポートは、セッションの実行後、コマンド ラインから、または自動バッチ プロセスの一部として行えます。

- ルールの管理とカスタマイズ

  ルール マネージャを使用して、設定した標準にコードが準拠するように、コード レビューが使用するルールを構成することができます。また、レビュー セッションで使用するために複数のルールをグループ化し、独自のカスタム ルールを作成することもできます。

すぐにコード レビューを使用するには

以下の準備、設定、実行手順では、DevPartner コード レビューの使用方法を紹介します。

機能をすぐに使用したい場合は、色付きの枠内に記載されている手順に従ってください。色付きの枠内に記載されているトピックの詳細な情報については、枠の下に記載されている文章を参照してください。

メモ： DevPartner コード レビューでは、ターゲット ファイルごとにデータ 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、ターゲットの実行可能ファイルを含むディレクトリ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：レビューの実行方法の決定

DevPartner コード レビューは柔軟性が高く、どのセッションでも考慮が必要なさまざまな構成があります。

この手順では以下を前提としています。

- Visual Basic または Visual C# のシングルデベロッパー ソリューションのレビューを実行しています。
- Visual Studio 2003 または Visual Studio 2005 でコード レビューを実行しています。
- ソリューションに含まれるすべてのプロジェクトはエラーなしでコンパイルされます。
- レビューするすべてのプロジェクトは、デバッグ情報を出力するように設定されています。

メモ： コード レビューでサポートされているプロジェクト タイプのリストについては、「コード レビューがサポートするプロジェクト タイプ」（325 ページ）を参照してください。

- 適用するルールの決定－業界のベスト プラクティスがコードに適用されるように、さまざまなコード レビュー ルールを使用できます。また、他にも実施する標準がある場合、ルール マネージャを使用してカスタム ルールとルール セットを作ることもできます。
- 適用するネーミング ガイドラインの選択－DevPartner コード レビューには、業界で受け入れられているネーミング標準にコードが準拠するように、組み込みのネーミング アナライザが用意されています。
- 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、これにより McCabe のメトリクスに基づいた、コードの複雑度の結果（複雑度、不良修正の確率、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 データを収集できます（すべての潜在的なインバウンド コールとアウトバウンド コールが表示されます）。
- ソリューション内のプロジェクトの除外－DevPartner コード レビューでは、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。コード レビューで分析したくないプロジェクトがソリューションに含まれる場合、除外することができます。

メモ： すべての選択プロジェクトは、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。使用できるビルド構成についてデバッグ情報を出力するように選択プロジェクトが設定されていない場合、コード レビューを実行するとビルド エラーに関する警告が表示され、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。

設定：オプションと設定の選択

DevPartner コード レビューは柔軟性が高くカスタマイズできます。コード レビューをカスタマイズするには、**[一般]** オプション ページ（図 3-1（60 ページ）を参照）を使用します。**[一般]** 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 **[DevPartner]>[コード レビュー]** を選択します。

この手順では、デフォルトの DevPartner 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。

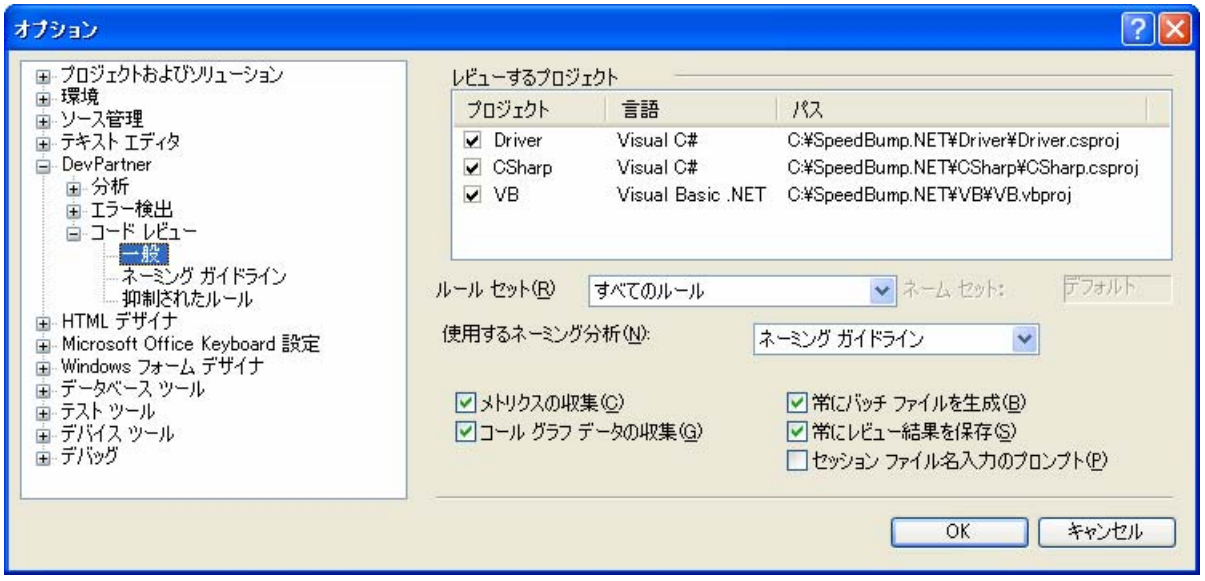


図 3-1. DevPartner コード レビューの [一般] オプション

- ルール セットの選択－レビューを実行する前に、**[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]**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]および[高]のルールがすべて含まれます。このルール セットを使用すると、業界で一般的なベスト プラクティスを実現できます。表 3-2（69 ページ）は、コード レビューに付属する標準のルール セット リストです。

メモ： **ルール マネージャ（「コード レビュー ルール マネージャの使用」（100 ページ）を参照）を使用すると、カスタム 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。デフォルトの動作では、Microsoft .NET のネーミング ルールをモデルにしたネーミング ガイドラインが適用されます。ただし、ハンガリアン記法ネーミング ルールを代わりに適用したり、ルールをまったく適用しないこともできます。

- メトリクス データ収集の有効化または無効化－McCabe のメトリクス データの収集を有効にするには、**[メトリクスの収集]**チェック ボックスをオンにします（「McCabe のメトリクスの収集」（70 ページ）を参照してください）。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 データ収集の有効化または無効化－静的メソッド コールのデータ収集を有効にするには、**[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、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、静的なインバウンドおよびアウトバウンドのコール パスの図が、結果ウィンドウの**[コール グラフ]**タブに表示されます。

メモ： **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メソッド コールではなく、コール パスの潜在的なメソッド 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 ボックスの各プロジェクトの横にあるチェック ボックスによって、プロジェクトをコード レビューで分析するかどうかを制御します（図 3-1（60 ページ）を参照）。コード レビューで分析しないプロジェクトに対応するチェック ボックスはオフにします。

メモ： **すべての選択プロジェクトは、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。使用できるビルド構成についてデバッグ情報を出力するように選択プロジェクトが設定されていない場合、コード レビューを実行するとビルド エラーに関する警告が表示され、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。**

実行：コード レビュー セッションの開始

コードを分析する DevPartner の処理は、セッションと呼ばれます。レビューが完了すると、セッション データが結果ウィンドウに表示され（図 3-2（63 ページ）を参照）、コード レビューの終了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。
   ソリューションの全プロジェクトについてコード 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 バーにセッションの進捗が表示されます。

基本的なコード レビュー セッションの実行が終わると、データが結果ウィンドウにまとめられ、分析可能になります。

結果の分析と違反の修正

ソリューションのレビューが完了すると、結果ウィンドウが操作の中心になります。結果ウィンドウに表示されたセッション データを使用して、違反の識別、場所の特定、修正を開始します。

1. 結果ウィンドウの **[問題]** タブ（図 3-2 を参照）で、レビュー中に検出されたコード違反を確認します。
2. **[重要度]** カラムには、重要度が高い順から違反が表示されます（デフォルトの動作）。必要に応じてカラムの昇順と降順を切り替えるには、カラム見出しをクリックします。

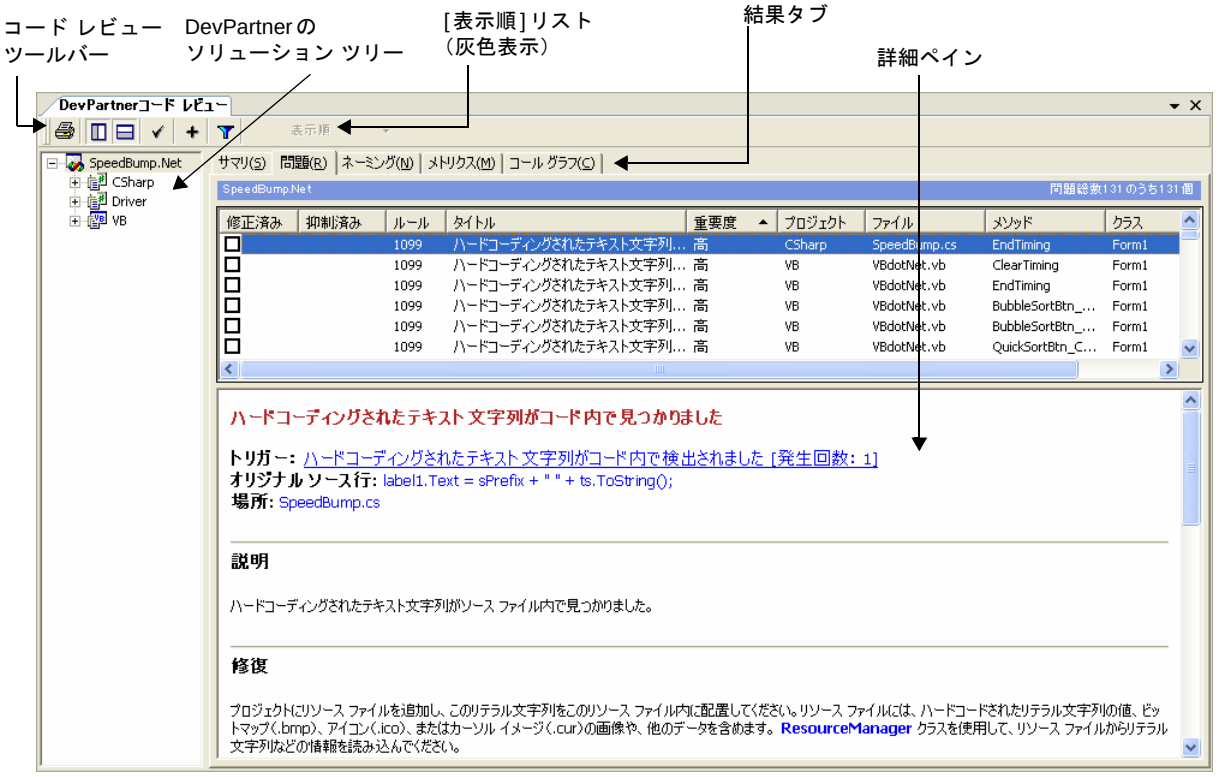


図 3-2. DevPartner コードレビューの結果ウィンドウ

ヒント: 重要なコード違反から修正したいと考えるのは一般的です。**[問題]** タブは、コード違反を重要度順に並べ替えることができるように設計されています。そのため、最も高い重要度の違反から簡単に選択することができます。

結果ウィンドウには、セッション データをカテゴリに分類するタブがいくつかあります。

- レビュー中に検出されたさまざまなタイプの違反をまとめたレポートを確認するには、**[サマリ]** タブを選択します（「サマリ データの表示」（76 ページ）を参照）。
- レビュー中に検出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]** タブでは、デフォルトで、違反リストが重要度の高い順に並べられています（「コード違反の表示」（77 ページ）を参照）。
- レビュー中に検出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、可能な場合修正案も表示されます。ネーミングを無視するようにレビューを設定していた場合（「ネーミング違反の表示」（79 ページ）を参照）は、空欄になります。
- McCabe のメトリクスに基づいた、コードの複雑度の結果（複雑度、不良修正の確率、理解度）を表示するには、**[メトリクス]** タブを選択します（「収集したメトリクスの表示」（82 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 データの表示」（85 ページ）を参照）。

結果のフィルタ

コード レビュー セッションの実行後、結果に多数のデータが含まれているために、修正する1つの領域に的を絞るのが難しいことがあります。コード レビューのソリューション ツリー（図 3-2（63ページ）を参照）でプロジェクト、ファイル、メソッドを選択すると、結果をフィルタできます。データをフィルタすると表示される内容が制限されるので、自分にとって最も重要な結果にフォーカスすることができます。

コード違反の分析

コード レビュー後の結果ウィンドウでは、デフォルトで、**[問題]**タブにフォーカスがあります。**[問題]**タブには、現在のソリューションで検出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する詳細ペイン（図 3-2（63ページ）を参照）には、詳細な説明、例、MSDNへの参照など問題を説明するソースと、選択したコード違反に対して提案される修正（可能な場合）が表示されます。

3. **[問題]**タブに表示されている最初のコード違反（最も高い重要度）を選択します。詳細ペインには、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]**の見出しを見ると、コード違反の理由と違反が発生した場所がわかります。
4. スクロールして、コード違反に関する**[説明]**、コード サンプル（可能な場合）、提案される**[修正]**を確認します。詳細については、違反に関する詳細説明への外部リンクをクリックします。
5. **[問題]**タブに表示されているコード違反をダブルクリックします。Visual Studioエディタとソース コードを含む新しいウィンドウが開きます。問題が発生したコードの行にフォーカスが配置されます。
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には、**[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、手順5～8を繰り返してそのセッションのコード違反を修正します。

メモ： **DevPartnerコード レビューでは行番号の変更が継続的に追跡され、違反とソース コードの同期が維持されます。ただし、変更量が多くなると、結果とソース コード間の同期が失われ、行番号が変わることがあります。このような場合、ソース コードの大部分を変更後にコード レビュー セッションを再実行し、新しい[問題]タブ リストで違反の修正を続けることができます。**

ここまでで、コード レビューを使用してソリューションのコード違反を解決しました。

ネーミング違反の分析

[ネーミング]タブには、コード レビュー中に検出されたネーミング違反が表示されます。**[ネーミング]**タブの表示内容は、レビュー前に**[一般]**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種類によって変わります。**[ネーミング]**タブの下に関連する詳細ペインが表示されます（図 3-2（63 ページ）の**[問題]**タブに関連する詳細ペインと同様です）。詳細ペインには、選択したネーミング違反の詳細な説明、リソース、修正案（可能な場合）が表示されます。

メモ：　ハンガリアン ネーミングを使用しているときは詳細ペインを使用できません。

図 3-3.　[ネーミング]タブと詳細ペイン

10 セッション中に検出されたネーミング違反を確認するには、**[ネーミング]**タブを選択します（図 3-3 を参照）。レビュー中に検出されたすべてのネーミング違反がこのタブに表示されます。可能な場合、正しいネーミングの提案が違反の横に表示されます。

11 **[ネーミング]**タブの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情報が表示されます（図 3-3 を参照）。

メモ：　ハンガリアン ネーミングを選択した場合、詳細ペインは使用できません。

12 詳細な説明や正しいネーミングの提案（表示されている場合）を確認します。違反の詳細情報を確認するには、外部リンクをクリックします。

13 **[ネーミング]**タブのネーミング違反をダブルクリックすると、該当のソースが表示されます。

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には、**[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、手順 11 ～ 16 を繰り返してそのセッションのネーミング違反を修正します。

ここまでで、コード レビューを使用してソリューションのネーミング違反を解決しました。

セッション ファイルの保存

セッション ファイルを保存すると、結果を再度参照することができます。以下のような理由から保存したセッション ファイルを開くことがあります。

- セッション データを XML にエクスポートする予定の場合（「データの XML へのエクスポート」（93 ページ）を参照）
- このセッションで検出された違反の修正をあとで続行する場合

メモ： [一般]オプションで[常にレビュー結果を保存]設定をオフにした場合を除き（「[一般]オプションの設定」（67 ページ）を参照）、終了時にデフォルトでセッション ファイルが保存されます。

1 結果ウィンドウにフォーカスを置き、**[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、**[保存]**をクリックします。
デフォルトでは、セッション ファイルはソリューションと同じ場所に *SolutionName*.dpmdb という名前で保存されます。

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。保存されたファイルは、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 セッション ファイルの拡張子は .dpmdb です。

デフォルトで、セッション ファイルはプロジェクトの出力フォルダに物理的に保存されます。また、デフォルト ディレクトリの内容に基づいて、ファイル名の番号が自動的に 1 つ増えます（たとえば、MyApp.dpmdb、MyApp1.dpmdb など）。セッション ファイルをデフォルト ディレクトリとは別の場所に保存する場合は、自分でファイルのネーミングを管理する必要があります。

Visual Studio 2005 の Web サイト プロジェクトなど、出力ディレクトリがないプロジェクトの場合、ファイルはプロジェクト ディレクトリに物理的に保存されます。

コマンド ラインから生成されたセッション ファイルは、自動的にプロジェクトのソリューションへ追加されません。外部で生成したセッション ファイルは、Visual Studio で開いているソリューションに手動で追加します。

この章の準備、設定、実行セクションはこれで終了です。コード レビュー セッションの実行方法について基本的な知識が習得できたはずです。詳細については、この章の残りを続けて読んでください。

設定オプション

コード レビューの動作のカスタマイズに使用できるオプションは多数あります。ユーザー固有の設定はユーザーのシステムのプリファレンス データベースに保存されます。コード レビューのオプションを変更するオプション ページは3つあります。

- [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]オプション

[一般]オプションの設定

[一般]オプション ページにはコード レビュー設定が含まれており、コード レビューの開始前に変更可能です。**[一般]**オプションにアクセスするには、**[DevPartner]**メニューの**[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]>[一般]**を選択します。

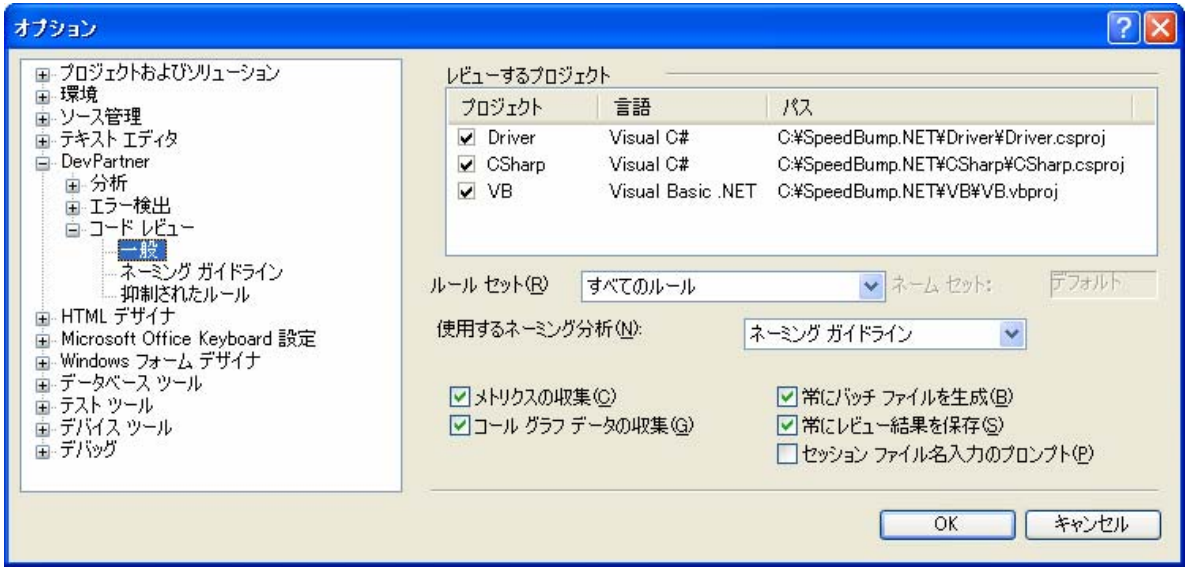


図 3-4. [一般]オプション ページ

レビューするプロジェクトの選択

[レビューするプロジェクト]リストから、ソリューションに含まれるプロジェクトの一部または全部を選択できます。以下の場合、リストは空です。

- Visual Studio で Visual C# または Visual Basic ソリューションをロードしていない
- Visual C++ プロジェクトのみを含むソリューションをロードした

[レビューするプロジェクト]リストには以下の情報が表示されます。

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	オンにすると対応するプロジェクトがレビューされます。
プロジェクト	プロジェクト名
言語	プロジェクトに関連付けられている Visual Studio の言語 • Visual Basic .NET • Visual C# • Web サイト **メモ：** Web サイトという言語の種類は、Visual Studio 2005 でのみ使用できます。ASP.NET テクノロジを使用する、言語に依存しないプロジェクトに関係があります。
パス	パスと表示されているプロジェクトの名前

プロジェクト リストから 1 つも選択しないと、デフォルトで現在のソリューションに含まれるすべてのプロジェクトがレビューされます。プロジェクト リストを編集すると、レビュー対象に含めたプロジェクトと除外したプロジェクトの状態が保存され、このソリューションを次に処理するときに利用できます。以下の点に注意してください。

- ソリューションをレビューするには、1 つ以上のプロジェクトを選択する必要があります。選択しないと、コード レビューは続行されません。
- デバッグ情報を出力するようにすべての選択プロジェクトを設定する必要があります。使用できるビルド構成についてデバッグ情報を出力するように選択プロジェクトが設定されていない場合、ビルド エラーに関する警告が表示され、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。
- ソリューションから削除されたプロジェクトを 1 つまたは複数選択すると、削除したプロジェクト以外がレビューされます。
- 誤ってソリューションのプロジェクトをすべて削除したあとにレビューを実行すると、ソリューション内に存在しないプロジェクトを選択したという警告が表示され、**[一般]**オプション ページの設定を適宜変更するように提案されます。

メモ： プロジェクト内の特定のファイル、クラス、またはメソッドを選択することはできません。

ルール セットの選択

[ルール セット] リストからコード レビューに適用するルール セットを選択します。**[ルール セット]** リストには、DevPartner に付属のルール セットとユーザー設定ルール セットがすべて表示されます。選択したルール セットの情報が保存され、現在のソリューションに対してセッションを実行するたびに使用されます。

メモ： **ルールが登録され、ルール データベースに存在している有効なルール セットを選択してください。**ルール マネージャで削除したルール セットまたは空のルール セットを使用しようとすると、結果が無効になることがあります。

ルールとルールに関連付けるトリガー（トリガーに違反するとルールが適用されます）の作成とカスタマイズ、ルール セットの作成と管理は、**ルール マネージャ**で実行できます（「コード レビュー ルール マネージャの使用」（100 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	すぐに使用できるマスタ ルール セットが表示されます。 • ルール データベースのすべての DevPartner ルールが含まれます。 • ルール データベースのユーザー設定ルールが含まれます。 • 総合的なコード レビューが可能です。
日付形式	日付の値の正しい形式と使用方法（特に 2 桁の年形式の日付）がチェックされます。
デフォルト	優先度が高と中のルールが含まれます。
デザイン タイム プロパティ	適切なユーザー インターフェイス設計になるように、フォームとコントロールのデザイン タイム プロパティとその値がチェックされます。
国際化	国際市場向けにローカライゼーション、文字列処理、比較を行うときに役立ちます。
ロジック	正しいプログラム ロジック、適切な .NET Framework プログラミング、エラー処理、型チェック、ガベージ コレクションがチェックされます。
パフォーマンス	パフォーマンスに悪影響を与えるコードがチェックされます。
ネーミング ガイドライン	ソース コードに複数の識別子を含む、.NET Framework のネーミングとの相違点が検索されます。
Web アプリケーション	適切な ASP.NET 開発、HTML タグの使用方法、バリデーション、パフォーマンス、キャッシング、状態がチェックされます。

ネーミング分析の種類の選択

ヒント: ネーミング分析の種類を選択する必要があります。選択しないと、重要な分析機能が省略されます。他のプログラミングの問題に集中するために、ネーミングの問題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] リストを使用して、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 Framework のネーミング ガイドラインをモデルにしています。

メモ： レビューをより正確にするには、**[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も設定する必要があります（「[ネーミング ガイドライン]オプションの設定」（71 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 ルールをモデルにしています（「ハンガリアン ネーミング アナライザ」（98 ページ）を参照）。

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

- **なし**：ネーミング分析は省略され、コード レビューの実行後 **[ネーミング]** タブは空になります。

ネーム セットの選択

ソース コードにハンガリアン ネーミング分析を実行する場合、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（デフォルトでは、**[デフォルト]** ネーム セットがデフォルトの DevPartner 付属のルール セットに関連付けられます）。

ネーム セットの作成と管理は、**ルール マネージャ** で行うことができます。「コード レビュー ルール マネージャの使用」（100 ページ）を参照してください。

McCabe のメトリクスの収集

[メトリクスの収集] を選択すると、複雑度、不良修正確率、理解度など、コードの複雑度の統計情報を示す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 のメトリクスに従います（「McCabe のメトリクス」（83 ページ）を参照）。[メトリクス] タブには、コード レビューのソリューション ツリーで選択したノードに関連するすべての項目が表示されます。

コール グラフ データの収集

[コール グラフ データの収集]チェック ボックスをオンにすると、メソッドまたはプロパティに対するすべての潜在的なインバウンド コールとアウトバウンド コールに関する情報が収集され、結果のグラフが**[コール グラフ]**タブに表示されます。コール グラフの各ノードは、選択したメソッドまたはプロパティのインバウンドまたはアウトバウンドのコール パスを表します。コール グラフには、プログラム実行中に行われる動的なコールではなく、コール パスの潜在的メソッド コールの静的表現が示されます。

バッチ ファイルの生成

[常にバッチ ファイルを生成]を選択すると、Visual Studio で次にインタラクティブ コード レビューを実行したときにバッチ ファイルが生成されます。このバッチ ファイルは、同じソリューションに対してコマンド ラインからバッチ レビューを実行するときに使用できます。

メモ： **バッチ ファイルまたはコマンド ラインから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 ファイルのバックアップを作成してファイル名を変更する必要があります。そうしないとバッチ ファイルは上書きされます。****「コマンド ライン インターフェイスの使用」（91 ページ）を参照してください。**

レビュー結果の保存

[常にレビュー結果を保存]チェック ボックスをオンにすると、コード レビューの実行後にソリューションと同じ場所に *SolutionName*.dpmdb という名前でセッション ファイルが保存されます。保存したセッション ファイルは Visual Studio の**ソリューション エクスプローラ**に表示されます。

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 ボックスをオンにすると、レビューの開始前に、セッション ファイルの場所と名前を指定するように求められます。

[ネーミング ガイドライン]オプションの設定

[ネーミング ガイドライン]オプション ページには、より正確にレビューを実行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]**メニューの**[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。

メモ： **このページでの設定内容は、[一般]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]リストで[ネーミング ガイドライン]のネーミング アナライザを選択するまで有効になりません。**

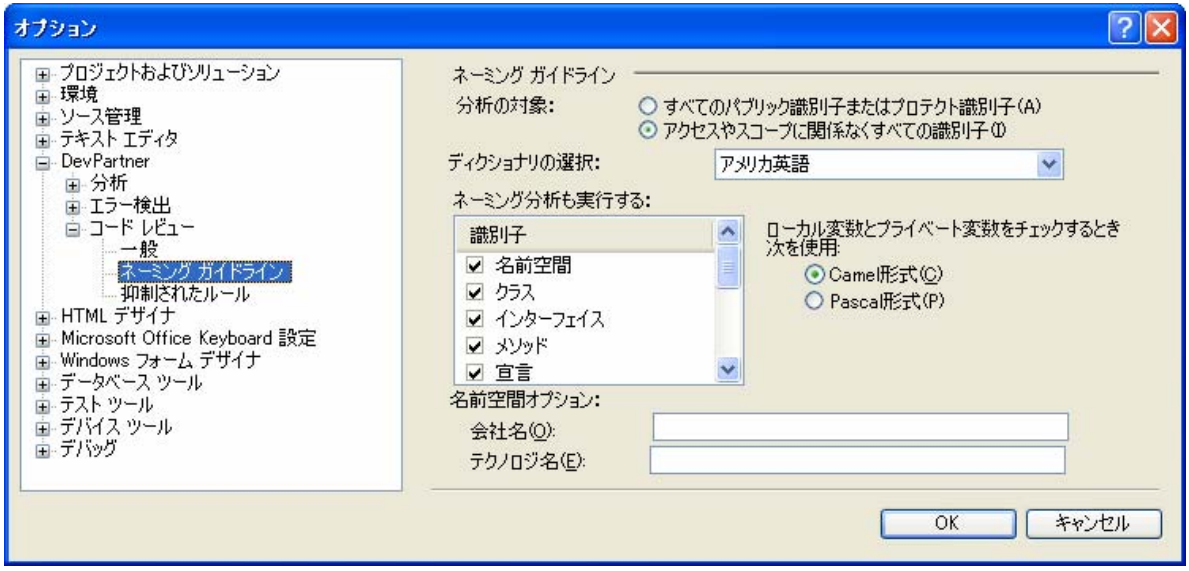


図 3-5. [ネーミング ガイドライン]オプション ページ

分析する識別子の選択

[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）**：パブリック識別子またはプロテクト識別子、および内部プロテクト識別子がチェックされます。ローカル識別子とプライベート識別子は除外されます。
- **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関係なくすべての識別子がチェックされます。

ディクショナリの選択

[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の選択

[ネーミング分析も実行する]リストから、分析する 1 つまたは複数の識別子に対応するチェック ボックスをオンにします。デフォルトではすべての識別子が選択されています。

メモ： **[変数]**チェック ボックスをオンにすると、このオプション ページに表示されているその他の変数固有の選択（**[分析の対象]**など）に影響が及ぶ可能性があります。

Camel 形式または Pascal 形式の選択

名前付き変数を検証するときに使用する大文字の設定を選択します。オプションは Camel 形式と Pascal 形式の 2 つです。Camel 形式は、最初の単語は小文字ですが 2 つめの単語は頭文字が大文字の変数を指します（たとえば、`integerBonus`）。Pascal 形式は、名前の各単語を大文字にした変数を指します（たとえば、`IntegerBonus`）。

メモ： **[ネーミング分析も実行する]** リストから **[変数]** を選択していない場合、このオプションは使用できません（灰色表示されます）。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]** をオンにした場合も使用できません。

名前空間オプションの選択

[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 ボックスをオンにすると、追加の名前空間オプションを指定できます。

- **会社名**：会社名を入力します。
- **テクノロジ名**：会社のテクノロジを入力します。

名前空間については、大文字の適切な使用、単語が完成しているかどうか、予約語の存在、数字の使用などがチェックされます。また、各名前空間が、`CompanyName.TechnologyName[.Feature][.Design]` という推奨する名前空間構文に従っているかどうかもチェックされます。会社名やテクノロジ名を入力すると、名前空間がそれらのエントリを使用して構成されているかどうかがチェックされます。

抑制されたルールの管理

[抑制されたルール] オプション ページには、**[問題]** タブで抑制したルール リストが表示されます（図 3-6 を参照）。抑制されたルールは、**[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 ボックスを選択して、一時的に抑制解除することができます。**[抑制されたルール]** オプションにアクセスするには、**[DevPartner]** メニューの **[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。

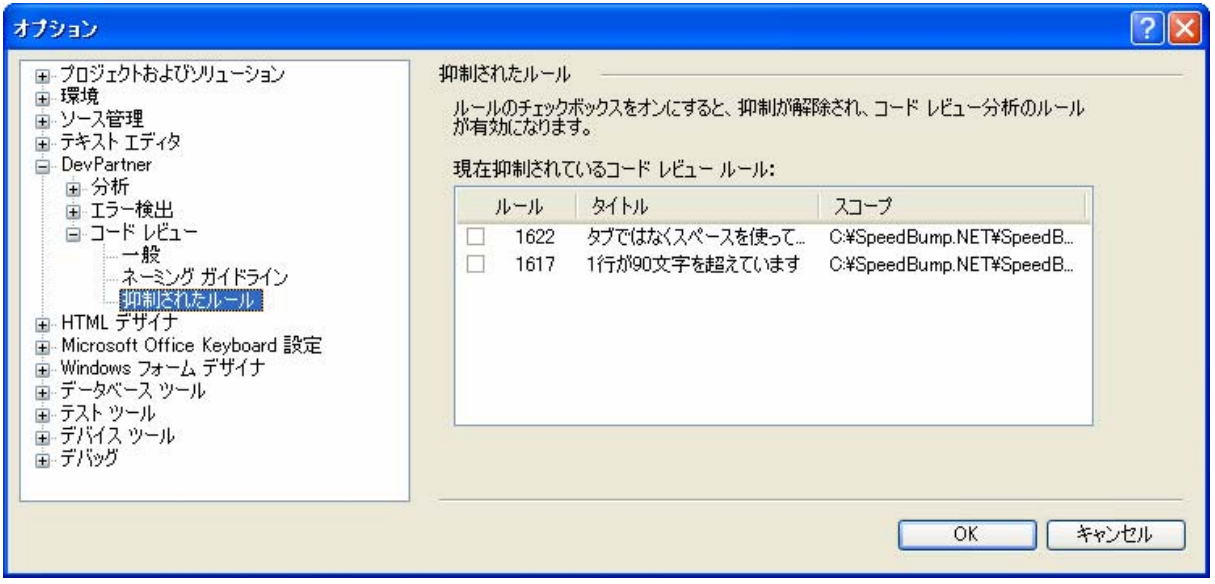


図 3-6. [抑制されたルール]オプション ページ

ルールの抑制

ルールを抑制すると、以降のセッションでそのルールが適用されなくなります。ルールの抑制はコード違反のフィルタとはまったく異なります。

- ルールを抑制すると、ルールは適用されず、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ても、基礎となるルールは適用されるので、データが収集され、セッション ファイルに保存されますが、表示はされません。

ルールの抑制は、個々のソリューションまたはすべてのソリューション共通に保存できます。抑制するルールを選択する前に、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表示されます。
2. コード違反を選択して基礎となるルールを抑制します。
3. 以下のいずれかの方法で**[ルールを抑制]**ダイアログ ボックスにアクセスします。
   - **[ルールを抑制]**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**[ルールを抑制]**を選択します。
4. 選択し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**：現在のソリューションのこれ以降のレビューについて有効です。
   - **すべてのソリューションに適用**：すべてのソリューションのこれ以降のレビューについて有効です。

メモ： **すべての抑制を選択すると、プリファレンス データベースではソリューションに基づくルールの抑制が解除され、すべてのソリューションに共通の設定が適用されます。**

現在のソリューションについてルールを抑制しようとしたときに、他のソリューションでそのルールがすでに抑制されていることが検出された場合、**[ルールを抑制]**ダイアログ ボックスにプロンプトが表示され、全体の抑制を適用するかどうかが確認されます。ここで、現在のソリューションでだけ抑制するよう選択することもできます。

サマリ データの表示

[サマリ]タブには、結果の概要データが統合して表示されます。セッションの各側面に関する詳細は、他の対応するタブに表示されます。**[サマリ]タブ**の項目の一部は、動的に更新されます。**[問題]**タブまたは**[ネーミング]**タブの項目を[修正済み]にすると、**[サマリ]タブ**にはその更新が動的に反映されます。レビューに特殊な例外が含まれる場合（たとえば、空のルール セットでレビューを実行するなど）、**[サマリ]**タブの見出しセクションにはそのメッセージが反映されます。各サマリ テーブルを表示するには、**[サマリ]**タブをスクロールします。

タイプ	問題		重要度			
名前	合計	修正済み	高	中	低	警告
COM相互運用性	1	0	0	0	0	1
Windows API	0	0	0	0	0	0
エラー/例外処理	19	0	0	1	18	0
ガベージ コレクション	0	0	0	0	0	0
システム	0	0	0	0	0	0
セキュリティ	3	0	3	0	0	0
データベース	0	0	0	0	0	0
デザイン タイム プロパティ	0	0	0	0	0	0
バージョン管理	0	0	0	0	0	0
パフォーマンス	1	0	1	0	0	0

図 3-7. **[サマリ]タブ**

- **[問題サマリ]**テーブルには、レビューで評価したルールのカテゴリが表示されます。また、検出された違反の数と、修正済みにした数が表示されます。さらに、重要度のカテゴリごとに、違反の合計数が表示されます。
- **[ネーミング ガイドラインのサマリ]**には、**[ネーミング ガイドライン]**オプション ページでレビュー対象に含めたカテゴリが表示されます（図 3-5（72 ページ））。このテーブルには、**[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と、検出された違反の数が表示されます。

メモ： **このテーブルが[サマリ]タブに表示されるのは、レビュー前に[一般]オプション ページの[ネーミング ガイドライン]を選択した場合のみです（図 3-4（67 ページ））。このテーブルは、ハンガリアン ネーミング アナライザには利用できません。**

- **[コール グラフ データのサマリ]**には、レビュー中に取得されたコール グラフ分析の概要が表示されます。たとえば、分析したメソッドとプロパティの合計数、未コールの可能性がある数などです。

メモ： このテーブルが**[サマリ]**タブに表示されるのは、レビュー前に**[全般]**オプション ページの**[コール グラフ データの収集]**を選択した場合のみです（図 3-4（67 ページ））。

- **[カウント サマリ]**には、コード レビュー セッション自体について収集された個々の統計情報が表示されます。たとえば、実行にかかった時間、ソリューションの行数、実行された比較の数などです。
- **[レビュー設定]**には、設定値とレビュー関連データが表示されます。この情報は、記録の保存や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、ソリューションの各プロジェクトに関する情報が表示されます。たとえば、各プロジェクトにコンパイル エラーがあったかどうか、レビューが正常終了したかどうかなどです。

コード違反の表示

コード レビュー後の結果ウィンドウでは、デフォルトで、**[問題]**タブにフォーカスがあります。**[問題]**タブには、現在のソリューションで検出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、修正案が表示されます。

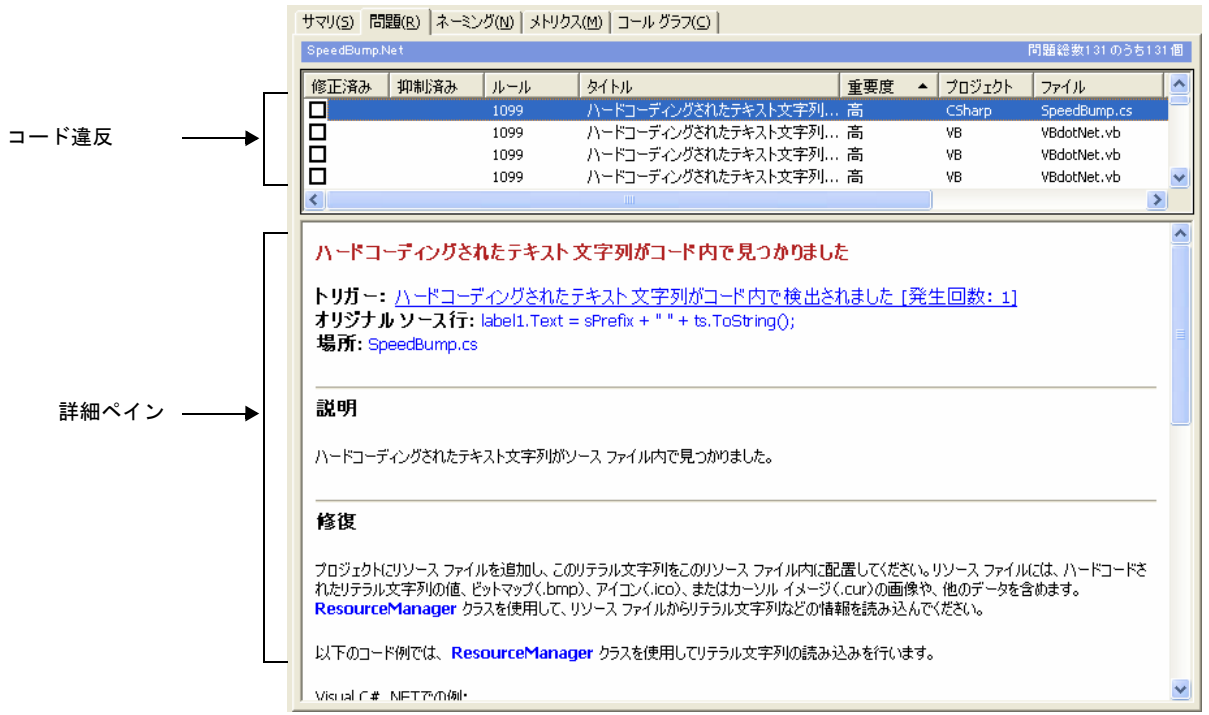


図 3-8. **[問題]**タブと詳細ペイン

[問題]タブ

[問題]タブに表示される情報について以下の表で説明します。

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 ボックスはオンです。
抑制済み	ルール抑制のステータス 抑制されていれば[抑制済み]、抑制されていない場合は空欄になります。
ルール	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

ヒント: 各コード違反には、トリガー、オリジナル ソース行、場所へのハイパーリンクが含まれることがあります。

[問題]タブでコード違反を選択すると、より詳細な情報が詳細ペインに表示されます（図 3-8（77 ページ）を参照）。この内容は、コード レビューのルール データベースに保存されているルールから生成されます（システム付属のルールとユーザー設定ルール）。詳細ペインに表示される情報を以下の表に示します。

表 3-4. 詳細ペインの内容

見出し	説明
ルールのタイトル （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 ソース行へ移動できるハイパーリンクとして表示されます（「トリガーの設定」（104 ページ）を参照）。
オリジナル ソース行	ルールが適用されたコード行
場所	コード違反の発生場所
説明	コード違反の説明

表 3-4.　詳細ペインの内容 （続き）

見出し	説明
修正	問題を修正するための提案
メモ	補足コメント（Microsoft MSDN サポート技術情報への外部リンクなど）

ネーミング違反の表示

[ネーミング] タブには、コード レビュー中に検出されたネーミング違反が表示されます。[ネーミング] タブの表示内容は、レビュー前に [一般] オプション ページで選択したネーミング分析の種類によって変わります（図 3-4（67 ページ））。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」（96 ページ）を参照してください。

メモ：　[ネーミング] タブには、2 つのネーミング アナライザの（両方ではなく）いずれかの結果が表示されます。[なし] を選択した場合、コード レビュー後 [ネーミング] タブは空になります。

ハンガリアン ネーミング アナライザの結果の分析

[一般] 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ときに表示される [ネーミング] タブを図 3-9 に示します。

図 3-9.　ハンガリアン ネーミング分析用の [ネーミング] タブ

ネーミング ガイドラインの結果の分析

[一般]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ときに表示される、ネーミング結果の 2 つのパネルを図 3-10 に示します。上部パネルの**[ネーミング]**タブと、下部パネルのネーミング詳細ペインに注目してください。ネーミング ガイドライン分析では、**[ネーミング]**タブの上にある**[表示順]**リストも使用できます。

図 3-10.　ネーミング ガイドライン分析用の[ネーミング]タブ

[ネーミング]タブに表示される情報を以下の表に示します。これらの情報は、選択したネーミング アナライザの種類にかかわらず表示されます。

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 違反を修正済みの場合はこのチェック ボックスをオンにします。
名前	データ型のユーザー定義の名前

表 3-5. [ネーミング]タブの内容 （続き）

カラム	説明
提案	提案される名前 提案される名前は、選択したネーミング アナライザの種類によって異なります（「ネーミング分析」（96 ページ）を参照）。 • ハンガリアン ネーミング ルールに基づいて名前を提案することができない場合、このカラムには「不明」と表示されます。 • ネーミング ガイドラインに基づいて名前を提案することができない場合、このカラムにはアスタリスクが表示されます。その場合、ネーミング詳細ペイン（図 3-10（80 ページ））に説明が表示されます。
アクセス	現在のソリューション内にあるアクセスのカテゴリ
タイプ	識別子のタイプ
メソッド	データ型が宣言されているメソッド
クラス	データ型が宣言されているクラス
名前空間	データ型が宣言されている名前空間
ファイル	データ型が宣言されているファイル
プロジェクト	データ型が宣言されているプロジェクト

ネーミング詳細ペイン

[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（72 ページ））で追加の選択を行うと、**[ネーミング]**タブの下に詳細ペインが表示され、選択したネーミング違反の詳細が表示されます。

メモ： 詳細ペインは、ネーミング ガイドライン ネーミング アナライザが使用されている場合にだけ表示されます。ハンガリアンでは表示されません。

表 3-6. ネーミング詳細ペインの内容

項目	説明
現在の名前	上部パネルで選択した項目に対応する名前
スコープ	識別子の範囲
オリジナル ソース行	上部パネルで選択したネーミング違反に関連するソース行
推奨	ネーミング ガイドライン ネーミング アナライザに基づき、1 つまたは複数の名前を提案します（「ネーミング ガイドライン ネーミング アナライザ」（96 ページ）を参照）。

表 3-6.　　ネーミング詳細ペインの内容 （続き）

項目	説明
説明	問題であるとのフラグがこの違反に付いた理由 **メモ**：より適切な名前をコード レビューが提案できない場合、このペインに説明が表示されます。また、**[ネーミング]**タブの上部パネルにある**[提案]**カラムには一連のアスタリクスが表示されます。
メモ	オプションで提供される、『.NET Framework General Reference』内のトピックへのハイパーリンクです。

収集したメトリクスの表示

[メトリクス]タブ（図 3-11）には、McCabe のメトリクス（「McCabe のメトリクス」（83 ページ）を参照）に基づいて、コードの複雑度の結果（複雑度、不良修正の確率、理解度）が表示されます。

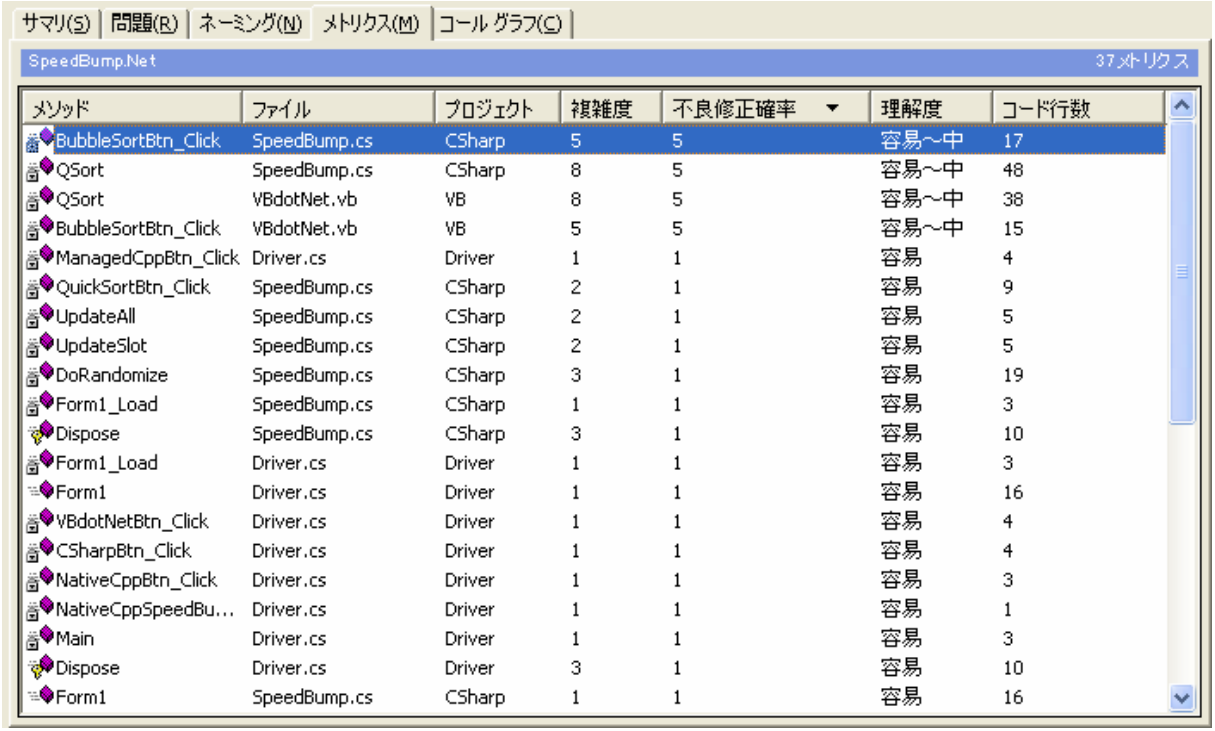


図 3-11.　　[メトリクス]タブ

[メトリクス]タブが表示されるのは、レビュー前に[全般]オプション 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です（図 3-4（67ページ））。[メトリクス]タブに表示される情報を表 3-7 に示します。

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検出されたメソッド名
ファイル	問題が検出されたファイル名
プロジェクト	問題が検出されたプロジェクト
複雑度	特定のコンポーネントに関する複雑度。このメトリクスは、McCabe の循環的複雑度を基準にしています。
不良修正確率	既存のバグの修正中に新しいバグが生じる可能性
理解度	コード ロジックの解読や保守の容易さのレベル
コード行数	選択したコンポーネント内のコード行の合計。各行のカウントの内訳は、[サマリ]タブに表示されます。

McCabe のメトリクス

McCabe のメトリクスを収集すると、[メトリクス]タブに、複雑度、不良修正確率、理解度など、コードの複雑度の統計情報が表示されます。これらのメトリクスは業界標準の McCabe のメトリクスに従います。[メトリクス]タブには、コード レビューのソリューション ツリーで選択したノードに関連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 のメトリクスの一部として確立した業界標準です。複雑度は、プログラムの安定性と信頼性を測る一般的な基準です。この基準は、他のプログラムの複雑度と比較できる 1 つの序数を提供します。多くの場合、他のソフトウェアのメトリクスと併せて使用されます。また、より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、言語や言語形式に依存しないように作成されています。複雑度の数値は、プログラムの構造的な複雑度を測る上で、コードの行数をカウントするよりも強力な基準になります。

複雑度は、プログラム モジュール全体で直線的に独立したパスの数を測定することで、モジュールの決定構造の複雑度を測定します。各コンポーネントが個々に分析され、すべての決定ポイント（たとえば、If-Then-Else ステートメントや Select Case ステートメント）が計算されます。Select Case の場合、各ケースが別個の決定ポイントになります。

McCabe のメトリクスでは、各モジュールに関する循環的複雑度を以下のように定義しています。

```
e - n + 2
```

e：コントロール フロー図のエッジ数

n：コントロール フロー図のノード数

循環的複雑度とは、テストが必要な最小パス数です。コードが複雑になるほど、そのコンポーネントのテスト量は増えます。

不良修正確率

不良修正確率とは、既知のバグを修正するときに、誤って新しいバグを増やす可能性です。不良修正確率ではプロシージャに注目し、新しいバグが追加される可能性を評価します。優れたコードであれば、一般的に不良修正確率の数値は低くなります。不良修正確率は、McCabe のメトリクスから派生したもので、理解度と複雑度の結果と相関関係があります。

理解度

複雑度と不良修正確率と同様に、理解度は、開発者によるコードの解釈と保守の容易さを評価します。理解度は、以下のようにコードを評価します。

表 3-8. 理解度のメトリクス

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

すべてのメトリクスの関連付け

3 つのすべてのメトリクスについて相関関係を以下の表に示します。

表 3-9. McCabe のメトリクスの相関関係

コードの複雑度の範囲	不良修正確率	理解度の評価
5 未満	1%	容易
5 ～ 10	5%	容易～中
11 ～ 20	10%	中
21 ～ 30	20%	中～高
31 ～ 50	30%	高度

表 3-9.　McCabe のメトリクスの相関関係 （続き）

コードの複雑度の範囲	不良修正確率	理解度の評価
51 ～ 94	40%	高～テスト不可能
95 以上	60%	テスト不能

コール グラフ データの表示

[コール グラフ] タブには、**コード レビューのソリューション ツリー**（図 3-12（85 ページ）参照）で選択したメソッドまたはプロパティに対応する、インバウンドおよびアウトバウンドのコール パスの静的ビューが表示されます。

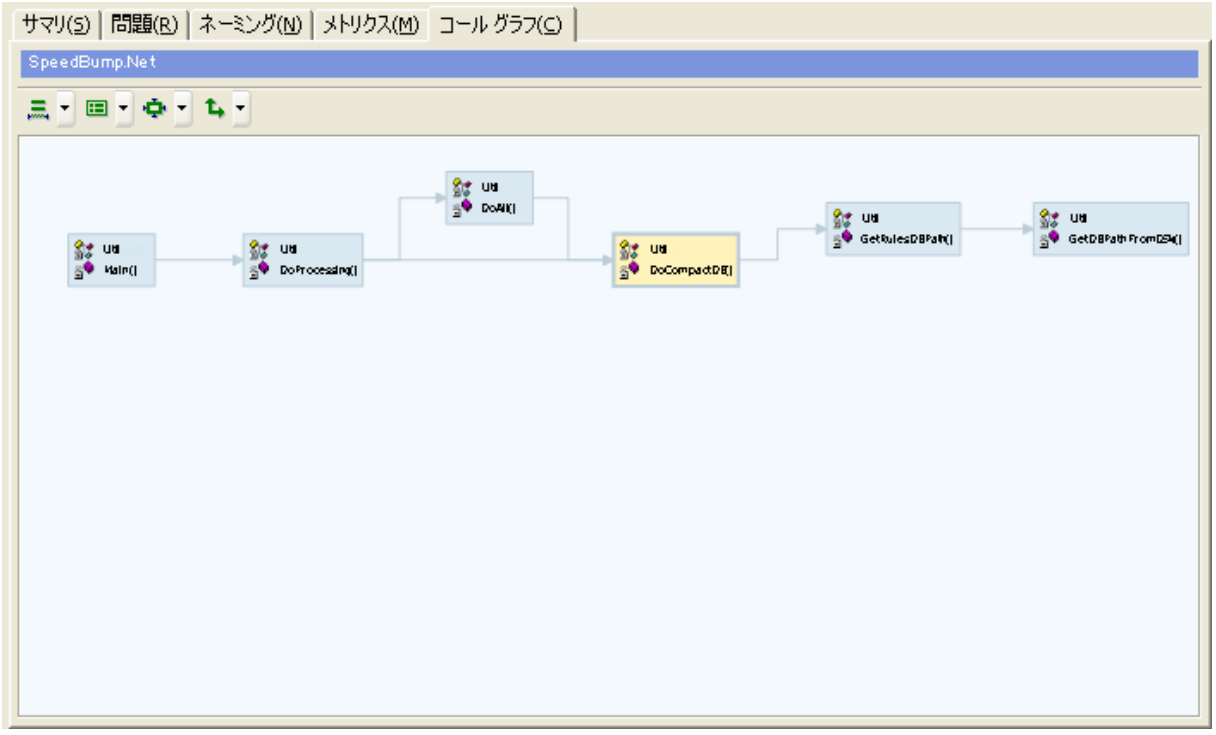


図 3-12.　コール グラフの例が表示された [コール グラフ] タブ

メモ： コール パスは、動的ではなく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メソッド コールではなく、コール パスの潜在的なメソッド コールが表示されます。

以下の場合、**[コール グラフ]** タブは空です。

- コード レビュー前に **[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 ボックスをオンにしなかった場合。レビュー時にコール グラフ データが収集されていません。コール グラフ分析を実行し、

コール グラフ データを収集するには、このオプションをオンにしてからコード レビューを実行します。

- チェック ボックスはオンにしたが、**コード レビューのソリューション ツリー**でメソッドまたはプロパティを選択しなかった場合（図 3-2（63 ページ）を参照）。データは収集されますが、**コード レビューのソリューション ツリー**でメソッド ノードまたはプロパティ ノードを選択しないと、コール グラフは表示されません。

コール グラフの参照

選択したメソッドまたはプロパティのコール階層が追跡され、**[コール グラフ]**タブにコール パスの潜在的なインバウンド／アウトバウンド コール参照が表示されます。表示領域には、各メソッドとプロパティの潜在的な開始点と終了点が表示されます。コール参照は、すべてのコールが参照するルート ノードから開始され、制御がルート ノードに戻るか、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です。他のすべてのノードは、ルート ノードを呼び出すか、ルート ノードにより呼び出されます。ルート ノード（図 3-13）は、青色の太い罫線に薄い黄色の四角形で表示され、他のノードとは区別されます。

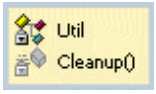


図 3-13.　ルート ノードの例

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コールするメソッドまたはプロパティです。インバウンド コールのノード（図 3-14）は、水色の四角形で表示され、ルート ノードと区別されます。

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コールされるメソッドまたはプロパティです。インバウンド コールと同様に、アウトバウンド コールのノード（図 3-14）は水色の四角形で表示されます。アウトバウンド コールは、ルートから出ている、コール パスの方向を示す一連の矢印で接続されます。

図 3-14.　インバウンドとアウトバウンドのコール ノードの例

未コールの参照

未コールとは、コードに定義されていても、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参照されていないメソッドまたはプロパティです。**[コール グラフ]** タブでは、未コール メソッドのノードを **[未コール]** というラベルまたは記号（!）で識別しています。

ラベルによる識別

アイコンによる識別

図 3-15.　未コール識別の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、選択した実行パスに存在する再帰コール参照または循環コール参照のインスタンスが示されることがあります。

- 再帰：実行パスで自身を呼び出すメソッドまたはプロパティ

  A は B を呼び出す。
  B は B を呼び出す。

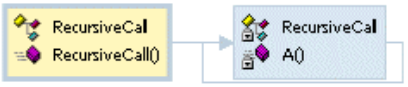


図 3-16.　再帰コール グラフの例

- 循環：実行パスで以前に呼び出し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 は B を呼び出す。
  B は C を呼び出す。
  C は A を呼び出す。

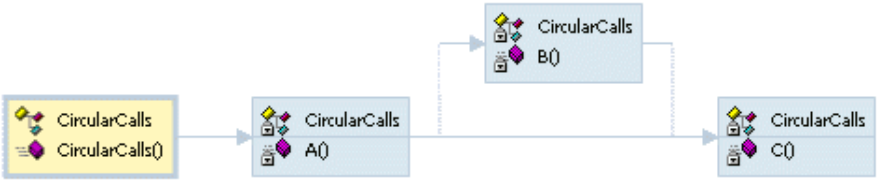


図 3-17.　循環コール グラフの例

コール グラフの設定オプションの設定

DevPartner コード レビューには、**[コール グラフ]** タブでのコール グラフの表示方法を設定するオプションが 4 つ用意されています。このオプションには、[コール グラフ] ツールバー、または **[コール グラフ]** タブの背景を右クリックして表示されるメニューからアクセスします。

レベル数

[コール グラフ] タブに表示するレベル数を選択します。コール グラフには、指定したレベル数のメソッドまたはプロパティが表示されます。これらのメソッドやプロパティは、ルート ノードを呼び出す（インバウンド）メソッドまたはプロパティ、およびルート 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です。選択可能なレベル数は、1～6 です（デフォルトは 6）。以下の例は、レベル数に 2 を選択した場合を示しています。コール グラフの右にある右端のノードに付いているプラス記号（+）は、コール参照をさらに表示可能であることを示しています。

図 3-18.　2 レベルを設定した場合

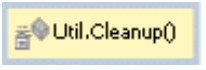
ノード スタイル

[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。すべてのコール グラフ ノード スタイルで、クラス名、およびメソッド名またはプロパティ名が表示されます。一部のノード スタイルでは、クラス、メソッド、またはプロパティのアクセス タイプ（パブリック、プライベート、内部、プロテクト）を示すアイコンも表示されます。これらは、標準のソリューション ツリー アイコンです。未コール メソッドやプロパティを表す他のアイコンは、コール グラフだけで表示されます。

以下の表に、さまざまなノード スタイルの例を示します。

メモ： **ルート ノードを表示した例と、標準ノード（インバウンドまたはアウトバウンド）を使用した例があります。**ノードの見分け方の詳細については、「コール グラフの参照」（86 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	未コール表示	例
シングル ラベル	クラス名、次にピリオド、続いてメソッドまたはプロパティ名が表示されます。アイコンはありません。	- Uncalled という文字列がメソッド名またはプロパティ名に付加されます。	 インバウンド／アウトバウンド
上下のラベル	最初の行にクラス名、次の行にメソッドまたはプロパティ名が表示されます。アイコンはありません。	- Uncalled という文字列が 2 行めのメソッド名またはプロパティ名に付加されます。	 インバウンド／アウトバウンド
1 つのイメージとラベル	標準のメソッドまたはプロパティ アイコンの後ろに、クラス名、ピリオド、メソッドまたはプロパティ名が表示されます（すべて同じ行）。	対応するアイコンには感嘆符アイコン（!）が含まれます。	 ルート
1 つのイメージと 2 つのラベル	メソッドまたはプロパティのアイコンの他に、1 行めにクラス名、2 行め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アイコン（!）が含まれます。	 ルート
2 つのイメージと 2 つのラベル	クラスの上位レベル アイコンに続いてクラス名が表示され、メソッドまたはクラスの下位レベル アイコンに続いてその名前が表示されます。	感嘆符アイコン（!）は、データ型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] タブに対するコール グラフの相対的なサイズを選択できます。使用できる拡大／縮小のオプションは以下の 2 つです。

- 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に収まるようにコール グラフが拡大／縮小されます。デフォルトでは、このオプションが選択されている場合はスクロール バーを使用できません。他のオプションを使用してコール グラフを設定し直すと、スクロール バーを含ま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、次の固定比率のいずれかを使って表示領域で内容を拡大/縮小できます：100%、80%、75%、66%、50%。このオプションにより、大規模なコール シーケンスまたは複雑なコール シーケンスの特定のセクションを拡大できます。さらに、コンテンツを再描画すると、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 バーも利用できます。

レイアウト

コール グラフ ノードを **[コール グラフ]** タブに表示する方法を選択します。以下のオプションがあります。

- 水平

  ノードは、表示領域で左から右の方向に表示されます。選択したノード（ルート ノード）を呼び出すメソッドまたはプロパティは、選択したノードの左に表示されます。選択したノードによって呼び出されるメソッドまたはプロパティは、選択したノードの右側に分岐します。

- 垂直

  ノードは、表示領域で上から下の方向に表示されます。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 ノードの上に表示されます。選択したノードによって呼び出されるメソッドまたはプロパティは、その下に表示されます。

コマンド ライン インターフェイスの使用

コマンド ライン インターフェイスからバッチ スクリプトを実行して（CRBatch.exe を使用）、マネージ コード 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、ユーザー操作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]** をオンにすると、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] をオフにするか、バッチ ファイルのバックアップを作成してファイル名を変更する必要があります。そうしないとバッチ ファイルは上書きされます。

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_*solution-name*.htm）をソリューション ディレクトリに作成します。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し、以下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 を呼び出してデータを XML にエクスポートし、書式設定や表示オプションを追加する行を含める（「データの XML へのエクスポート」（93 ページ）を参照）

バッチ レビューを実行できない場合は、CR_*solution-name*.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_*sessionfiledatabasename*.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

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 セッションを実行するには、以下のコマンド ライン構文とオプションを使用します。

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

表 3-11.　コマンド ライン オプション

オプション	定義
/?	CRBatch.exeのコマンド ライン オプションがリスト表示されます。
/f *filename*	レビューに使用する構成ファイルを指定します（必須）。
/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 プロセスで使用する終了コードを設定するように指示します（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指示します（オプション）。
/vs "7.1" または /vs "8.0"	バッチ レビューが実行されるVisual Studio .NET Frameworkのバージョンを指定します。 7.1 (2003) または 8.0 (2005)

エラー ファイル

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 ラインのエラー コード

エラー番号	メッセージ
0	Successful（成功）
1	No configuration file specified（構成ファイルが指定されていません）
2	Configuration file does not exist（構成ファイルが存在しません）
3	No solution file was specified（ソリューション ファイルが指定されていません）
4	Solution file does not exist（ソリューション ファイルが存在しません）
5	CRBatch initialization error（CRBatch初期化エラー）
6	Invalid command line argument（コマンド ラインの引数が無効です）
7	Create Visual Studio process failed（Visual Studioプロセスの作成に失敗しました）
8	License check failed（ライセンスのチェックに失敗しました）

表 3-12.　コマンド ラインのエラー コード

エラー番号	メッセージ
9	Visual Studio exited with an error（Visual Studio がエラーで終了しました）
10	Visual Studio version number incorrect（Visual Studio のバージョン番号が正しくありません）
11	Unexpected error（予期しないエラー）
12	Coding problems found（コーディング問題が検出されました）
13	Naming violations found（ネーミングの違反が検出されました）
14	Coding problems and naming violations found（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	Attempt to create error file (.ERR) failed（エラー ファイル（.ERR）を作成できませんでした）

バッチで生成したレビューでビルド エラーが発生するか、レビューするソリューションにコンパイル エラーが存在する場合、バッチ レビューは停止し、セッション ファイルとサマリ ファイルは生成されません。エラー ファイルにエラー メッセージが付加されます。

メモ：　予期しない実行時エラーの場合、エラー 11 が返されます。エラーの詳細（エラー メッセージとスタック トレース）は .ERR ファイルに出力されます。

データの XML へのエクスポート

DevPartner コード レビューを使用すると、セッションの結果データを XML にエクスポートできます。そのため、結果データをレポート形式、電子メール、社内の Web ページなどへ簡単にコピーできます。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
- 保存したセッション ファイルを使用してコマンド ラインから
- 保存したセッション ファイルを使用して自動バッチ プロセスで

コード レビューのインストール ディレクトリにある DPCRExport.xsd スキーマ ファイルには、エクスポート データのコンテンツと XML 形式が記述されています。

DevPartner からのセッション データのエクスポート

コード レビューの完了後、現在のセッション ファイルの全データを XML ファイルにエクスポートできます。[ファイル] メニューの [DevPartner データのエクスポート] を選択し、エクスポート ファイルの名前を指定します。デフォルトで、このファイルは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。

メモ： 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置いておく必要があります。

エクスポートでは、コール グラフ データのインバウンド メソッドを含め、常にすべてのセッション データが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選択するには、コマンド ラインを使用します。

セッション データの XML へのエクスポートに失敗した場合、発生した問題を説明するエラー メッセージが生成されます。

コマンド ラインからのセッション データのエクスポート

DevPartner コード レビューには、CRExport.exe というコマンド ライン ユーティリティが付属しており、コード レビュー セッションの結果を XML ファイルにエクスポートできます。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 ライン引数にセッション ファイルと出力ファイルを指定します。次に例を示します。

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

オプションのコマンド ライン引数を使用して、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。

メモ： オプションの引数を指定せずに CRExport.exe を呼び出すと、インバウンド メソッドを含め、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 引数を指定した場合、または DevPartner からデータ エクスポートを実行した場合と同じです。

エクスポート ファイルの作成に失敗した場合、CREXPORT_*sessionfiledatabasename*.err というエラー ログ ファイルがセッション ファイルと同じパスに生成されます。

構文とオプション

コマンド ラインまたはバッチ ファイルでセッション データをXMLにエクスポートするには、以下のコマンド ライン構文とオプションを使用します。

```
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のコマンド ライン オプションがリスト表示されます。
/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 データを受け取るXML ファイルを指定します（必須）。
/a	コール グラフ データのアウトバウンド メソッドを含め、指定したセッションのすべてのデータがエクスポートされます（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 メソッドを含め、指定したセッションのすべてのデータがエクスポートされます。
/p	指定したセッションの問題データがエクスポートされます。
/m	指定したセッションのメトリクス データがエクスポートされます。
/n	指定したセッションのネーミング分析データがエクスポートされます。
/s	指定したセッションのコード サイズ データがエクスポートされます。
/c	指定したセッションのコール グラフ データのアウトバウンド メソッド（呼び出されたメソッド）がエクスポートされます。
/c i	指定したセッションのインバウンド メソッドとアウトバウンド メソッドを含め、コール グラフ データがエクスポートされます。

バッチ プロセスからのセッション データのエクスポート

1つのバッチ プロセスとしてCRExport.exeをCRBatch.exeと一緒に使用して、コード レビューを実行したあとに、セッション データをXMLにエクスポートすることができます。この機能は、以下のようなバッチ プロセスでコード レビューを実行しているときに特に便利です。

- 夜間のビルド プロセスの一部として
- 大規模なアプリケーションに対して
- 品質管理テストを自動化するため

ネーミング分析

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

  このネーミング アナライザは、.NET Frameworkをサポートしています。「ネーミング ガイドライン ネーミング アナライザ」（96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 レビューの古いネーミング アナライザです。「ハンガリアン ネーミング アナライザ」（98ページ）を参照してください。

メモ： **[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]を選択して、ネーミング分析を省略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

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のネーミング ガイドラインに従って作られています。これらのネーミング ガイドラインにより、マネージ クラス ライブラリにおける.NET Frameworkタイプに対して、一貫性があり、予測可能な、管理しやすいネーミングを確実に適用できるようになります。

メモ： **レビューをより正確にするには、[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、さらに[ネーミング ガイドライン]オプション ページ（図3-5（72ページ））で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、以下が分析されます。

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

また、大文字の使用、大文字と小文字の区別、省略形と頭字語、名前空間や他の.NET Framework識別子の構文に関するネーミング違反がソース コードから検索されます。

以降のセクションでは、ネーミング ガイドライン ネーミング アナライザが従うガイドラインについて説明します。

大文字の使用

コード レビューでは、ネーミング違反が検出されると、**[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の使用スタイル（CamelまたはPascal）に応じた適切な名前が提案され、**[ネーミング]** タブに表示されます。

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、大文字の使用スタイル

大文字の使用スタイル	最初の連結語	続く連結語	推奨される名前の例
Camel 形式	頭文字は大文字でない	頭文字は大文字	**redC**olor
Pascal 形式	頭文字は大文字	頭文字は大文字	**RedC**olor

大文字と小文字の区別

ソース コード内の識別子を区別するために大文字と小文字を区別することは推奨できません。大文字と小文字を「区別しない」ようにしてください。これは、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との相互運用をサポートし、よく似た名前の識別子の混同を避けるために重要です。大文字か小文字かが違うだけで他は同じである名前を使うべきではありません。プログラミング言語が大文字と小文字を区別しても区別しなくても機能する名前を使用してください。

省略形と頭字語

DevPartnerコード レビューでは、一般に認められている省略形と頭字語の使用をサポートしています。以下に基づいて正しいネーミングが判断されます。

- 省略形と頭字語の文字数
- 識別子名内の省略形または頭字語の位置

名前空間構文

DevPartnerコード レビューは、名前空間に.NET Frameworkネーミング ルールを適用します。そのため、名前空間名は、会社名で始まり、次にテクノロジ名が付加され、オプションとして機能や設計名で終わります。構文の例を以下に示します。

```
CompanyName.TechnologyName[.Feature][.Design]
```

デフォルトでは、名前空間にPascal形式を推奨しています（「Camel形式またはPascal形式の選択」（73ページ）を参照）。ピリオド文字（.）で各論理連結語を区切ります。レビュー開始前に、**[ネーミング ガイドライン]** オプション ページ（図 3-5（72ページ））にある **[名前空間オプション]** フィールドに、名前空間情報を入力します。

他の .NET Framework 識別子の構文

ソース コードで .NET Framework 識別子が正しい名前になっているかどうかが確認されます。検索される内容の例を以下に示します。

- 数字

  識別子名の一部に数字が使われているかどうかがチェックされます。数字が検出された場合、削除はしませんが、違反であることを示すフラグが付加されます。

- アンダースコア

  識別子名にアンダースコア（_）を用いているインスタンスが検索されます。アンダースコアは、ネーミング ガイドライン ネーミング アナライザで使用することは避けてください。アンダースコアが検出された場合は、以下の場合を除き削除されます。

  - アンダースコアが先頭文字である場合（_redColor など）
  - メソッド名に使用されている場合
  - 削除すると他のネーミング違反が発生する場合

- 定数の形式

  DevPartner コード レビューでは、定数はすべて大文字ではなく、Pascal 形式または Camel 形式に準拠しています。どちらの形式であるかは、**[ネーミング ガイドライン]** オプション ページで選択した形式によります。たとえば、定数 HTTP_PORT は以下のように変更されます。

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

  - HttpPort（Pascal 形式）
  - httpPort（Camel 形式）

- デリゲート

  デリゲート識別子名に、delegate という語（大文字小文字は区別しない）が識別可能な 1 つまたは複数の語と一緒に含まれている場合、他の違反を引き起こさないかぎり delegate という語は削除されます。たとえば、`MyDelegateWord` という名前は、`MyWord` という名前に変更されます。

ハンガリアン ネーミング アナライザ

DevPartner コード レビューには、ハンガリアン記法ネーミング ルールに従って作られたハンガリアン ネーミング アナライザが付属しています。

ハンガリアン ネーミングでは、変数名には、その変数のスコープレベルまたはデータ型を特定する接頭辞が含まれます。たとえば、データ型接頭辞の 1 つで整数を示す `int` は `Port` のように整数の変数を示し、スコープレベルの 1 つである `g_` は `g_intPort` のようにスコープがグローバルであることを示します。

コード レビューでハンガリアン ネーミング アナライザが使用されるのは、**[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]**リストで**[ハンガリアン]**オプションを選択した場合です。また、現在選択しているネーム セットが使用されます。コード レビューを開始すると、コード内の各変数について、スコープレベル接頭辞とデータ型接頭辞がネーミング アナライザで評価されます。可能な場合は、ネーム セット（デフォルト優先）と整合性のある「推奨される名前」が作成され、コード レビュー後に**[ネーミング]**タブ（図 3-9（79 ページ））にネーミング結果が表示されます。

メモ： ハンガリアン ネーミング アナライザは、パラメータ名を評価しません。

以下の表に、現在のネーム セットに示されている、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接頭辞とデータ型接頭辞の組み合わせの例を示します。

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
string	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 ステータスの変数は、クラス外からアクセスできるため、グローバル スコープとして扱われます。

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ます。また、ハンガリアン記法に基づいて変数やオブジェクト名をルール マネージャでカスタマイズすることもできます。

ハンガリアン ネーミングによる提案

ハンガリアン ネーミング アナライザでは、ソース コードに以下のような問題が検出された場合に、より適切なネーミングが提案されます。

- スコープ接頭辞が正しくない、または欠落している（たとえば、グローバルなのに `g_` ではなく `m_` の場合）。
- データ型接頭辞が正しくない、または欠落している（たとえば、整数型なのに `intShort` ではなく `Short` の場合）。
- **[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]** ダイアログ ボックスまたは **[ルール セットの編集]** ダイアログ ボックス）をオンにしてハンガリアン ネーム セットに適用したが、接頭辞に続く変数名の先頭文字が大文字になっていない。

ネーミング アナライザでは、接頭辞が**スコープレベル接頭辞 + データ型接頭辞**のように組み合わされます。ハンガリアン ネーム セットでスコープレベル接頭辞を指定していない場合、提案される名前はデータ型接頭辞で始まります。

以下の理由からコード レビューで変数のデータ型が認識できない場合は、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。

- データ型が現在のハンガリアン ネーム セットに存在しない
- **ルール マネージャ**の **[新規ルール セット]** ダイアログ ボックスまたは **[ルール セットの編集]** ダイアログ ボックスで **[不明なオブジェクトが検出された場合に警告する]** チェック ボックスをオンにして、ネーム セットに適用した

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 マネージャの使用」（100 ページ）を参照してください。

コード レビュー ルール マネージャの使用

DevPartner コード レビューには、Microsoft Visual Studio プログラミングの標準に基づいた、拡張可能なルール データベースがあります。ルール データベースは、スタンドアロン アプリケーションのルール マネージャで保守および保存します。ルール マネージャを使用すると、ルール、トリガー、ルール セットを設定できます。また、コード レビュー中にハンガリアン ネーミング アナライザで使用されるハンガリアン ネーム セットも設定できます。ルール マネージャは、コード レビュー ルール データベースに加えた変更を自動的に保存します。そのため、次にコード レビューを設定および実行するときに、変更したものをすぐに利用できます。

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 [Compuware 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。

ルールの設定

ルール マネージャを使用してルールの作成、編集、削除を行います。また、ルールの説明にHTMLリンクを追加して、開発者が違反を解決するときに役立つ詳細情報を提供することもできます。

ルールの作成

新しいルールを作成して設定するには、**[新規ルール]**ダイアログ ボックスを使用します。新しいルールを作成するには、以下の操作を行います。

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 ボックスが開きます。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]**のデータが表示されます(図 3-19 を参照)。

メモ： ルールの[トリガー]と[式]を作成するまで、ルールは適用されません。

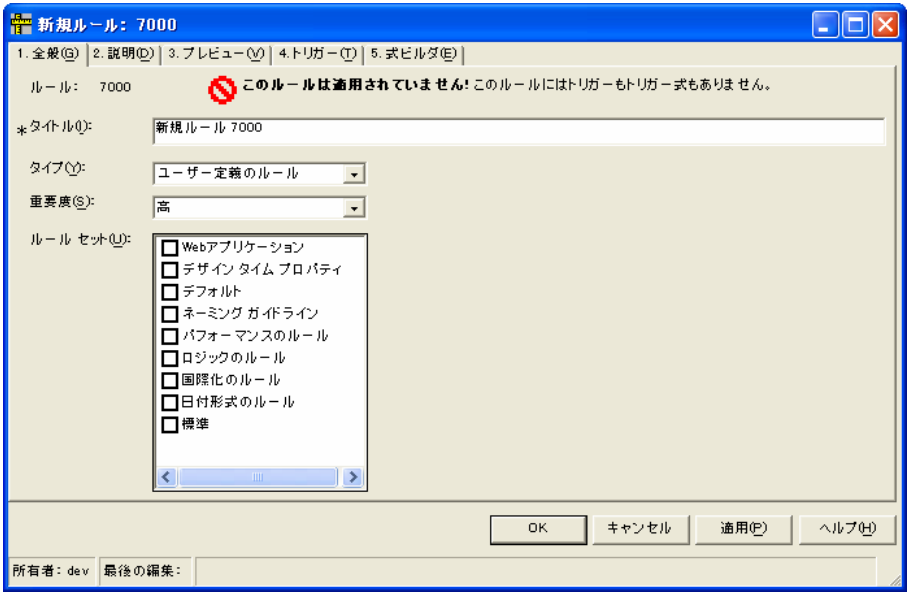


図 3-19. [新規ルール]ダイアログ ボックス

2 以下のタブを使用して新規ルールを設定します（ここに示す順番で）。

a **[全般]**－全般的なルール プロパティを入力します。

b **[説明]**－ルールの説明を入力します。

c **[プレビュー]**－現在の内容を確認します。

d **[トリガー]**－ルールのトリガーを最大5つまで設定します。

e **[式ビルダ]**－各トリガーのトリガー式を作成します。

3 **[説明]**タブをクリックし、ルールの説明を追加します。
[説明]タブにHTMLリンクを入力して、開発者が外部リソースを参考にしてコーディングの問題を解決できるようにすることもできます。このリンクは、コード レビュー セッション後に、**[問題]**タブの下部パネル（**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、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、「トリガーの設定」（104ページ）を参照してください。

5 **[式ビルダ]**タブをクリックし、トリガー式を作成します。
[トリガー]タブで設定した各トリガーについて、式を作成できます。トリガー式の作成方法の詳細については、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。

ルールの編集

[ルールの編集]ダイアログ ボックスを使用して、既存のルール プロパティを変更します。[ルールの編集]ダイアログ ボックスのフィールドは、[新規ルール]ダイアログ ボックスとまったく同じです（図3-19（101ページ）を参照）。既存のルールを編集するには、以下の操作を行います。

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 ボックスが開きます。デフォルトで**[全般]**タブが表示されます。タイトル バーにはルールの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]**のデータが表示されます。

2 以下のタブを使用して既存のルールを変更します（ここに示す順番で）。

a **[全般]**－既存のルール プロパティを変更します。

b **[説明]**－ルールの説明を変更します。

c **[プレビュー]**－現在の内容を確認します。

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]**－各トリガーのトリガー式を変更します。

ルールの削除

[すべてのルール] に表示されるユーザー設定のルールのみを削除できます。DevPartner 付属の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、そのルールが設定されていた他のルール セットからも自動的に削除されます。

メモ： DevPartner 付属のルールを編集すると、システムの所有から削除されますが、ユーザー定義のステータスには変更されません。DevPartner 付属のルールを **[すべてのルール]** から削除することはできません。

ルールを削除するには、以下の操作を行います。

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] のルールが**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
1014	オブジェクトのデストラクタ内で作成され...	高	ガベージ コレクション	Visual Basic .NET, Visual C# .NET
1015	デストラクタ内でオブジェクトの復活が行...	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** ペインから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。
[すべてのルール] ルール セットから選択したルールが削除されます。

メモ： この操作は元に戻せません。

メモ： **[ルール]** メニューの **[ルール データベースから選択したルールを削除]** を有効にするには、**[ルール セット]** リストから **[すべてのルール]** を選択する必要があります。

トリガーの設定

[トリガー]タブを選択し、ルールを適用させるトリガー（最大5つ）を設定します。

メモ：　一部のDevPartner付属のルールではマクロが使用されていますが、マクロベースのルールのトリガーは編集または設定できません。

ルールにトリガーが関連付けられていない場合、表示されるフィールドは空の[既存のトリガー]リスト ボックスのみです。また、設定するトリガーを追加できるように、[追加]ボタンが有効になります。

1つまたは複数のトリガーが[既存のトリガー]に表示されている場合、トリガーの[タイプ]に適用できる他のフィールドが表示され、必須のフィールドにはアスタリスクが表示されます。また、[追加]ボタンと[削除]ボタンが有効になります。

トリガーの追加

トリガーを追加するには、以下の操作を行います。

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。
[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、最初のトリガーの場合、名前は「**新規トリガー 1**」になります（図 3-21）。

メモ：　トリガーの数が上限の5つに達すると、このペインの[追加]ボタンは自動的に使用できなくなります。

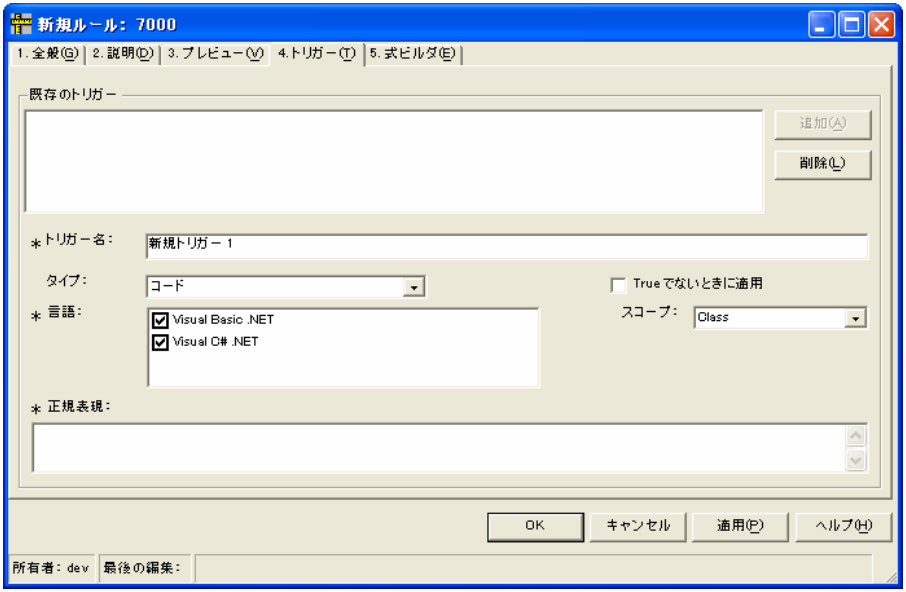


図 3-21.　新規トリガータブ

2 トリガー名を入力または変更します。
[CheckString] のように、左の角かっこ [と右の角かっこ] は使用しないでください。角かっこ文字を入力すると、そのキー入力は無視されます。**[式ビルダ]** ペインの **[式]** フィールド内では、角かっこはトリガー式の複数のトリガーを区切るために使用されます。手動で角かっこを入力しようとすると、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のタイプを選択します。
トリガーのタイプを選択すると、そのトリガーの他のパラメータが決まります（表 3-17 を参照）。

表 3-17. トリガーのタイプ

タイプ	説明
コード	実際のソース コードの問題が検出されます。
Web Form Page	HTML や ASP.NET のタグ構造との準拠がチェックされます。
デザイン タイム プロパティ	トリガーの起動を特定の Visual Studio .NET プロパティに制限します。
Web.config	ASP.NET の Web.config ファイルに含まれる要素との準拠がチェックされます。

4 選択したトリガーのタイプについてすべての必須フィールドを設定します。
選択した **タイプ** によって、トリガーに指定する必須のパラメータは異なります。特定のトリガーのタイプを設定する方法の詳細については、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。

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（「正規表現を使用した新しいルールの作成」（113 ページ）を参照）。

トリガーの削除

[既存のトリガー] に表示されているトリガーを削除するには、以下の操作を行います。

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 この操作を実行するには **[はい]**、中止するには **[いいえ]** をクリックします。

メモ： トリガー式に使用されているトリガーは削除できません。

ルール セットの設定

ルール セットは、コード レビュー セッションに使用できるルールの集合です。DevPartner コード レビューには、事前に設定済みのルール セットが付属しています。カスタムのルール セットが必要な場合は、ルール マネージャを使用してルール セットの作成、編集、削除を行えます。

ルール セットの作成

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 セットがあります。ただし、プロジェクト固有の要件に合わせた新しいルール 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、以下の操作を行います。

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（図 3-22）。

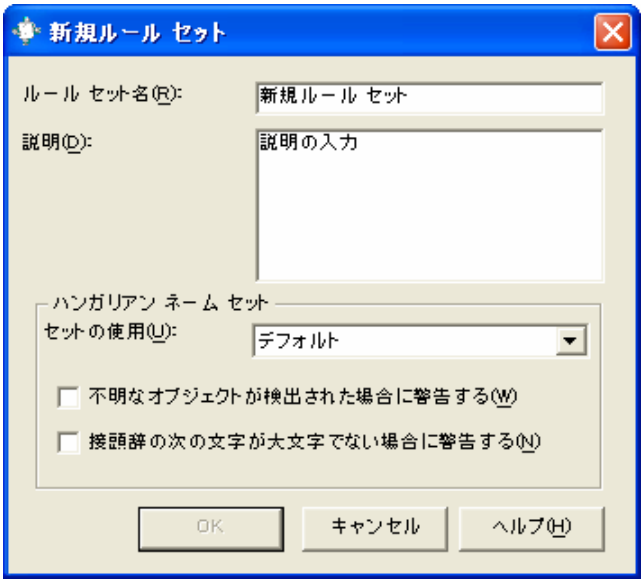


図 3-22. [新規ルール セット] ダイアログ ボックス

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

3 **[説明]** フィールドに新しいルール セットの簡単な説明を入力します（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]** セクションの **[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。

メモ： ルール マネージャのネーム セットは、ハンガリアン ネーミング ルールをモデルに作成されたハンガリアン ネーミング アナライザのみをサポートしています。Visual Studio .NET ネーミング ガイドラインをモデルに作成されたネーミング ガイドライン ネーミング アナライザはサポートしていません。

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、提案が不可能なネーミング違反に対しては**[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]**を選択すると、提案が作成されます。

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 以下の手順でルール セットにルールを設定します。

- ◇ 新しいルールを作成します（「ルールの作成」（101ページ）を参照）。
- ◇ 既存のルール セットを開き、ルールを選択してコピーし、新しいルール セットに貼り付けます。

ルール セットの編集

ルール セットのプロパティを編集するには、以下の操作を行います。

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 ボックスのフィールドは、[新規ルール セット]ダイアログ ボックスと同じです（図 3-22（106ページ）を参照）。

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

4 **[説明]**フィールドに新しいルール セットの簡単な説明を入力します(オプション)。

5 ダイアログ ボックスの**[ハンガリアン ネーム セット]**セクションの**[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。

メモ： **ルール マネージャのネーム セットは、ハンガリアン ネーミング ルールをモデルに作成されたハンガリアン ネーミング アナライザのみをサポートしています。Visual Studio .NET ネーミング ガイドラインをモデルに作成されたネーミング ガイドライン ネーミング アナライザはサポートしていません。**

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、提案が不可能なネーミング違反に対しては**[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]**を選択すると、提案が作成されます。

7 **[OK]**をクリックします。
ルール セット プロパティの変更内容が検証されます。

ルール セットの削除

既存のルールを削除するには、以下の操作を行います。

1 **[ルール セット]**リストからルール セットを選択します。

メモ： ユーザー定義のルール セットは削除できますが、DevPartner 付属のルール 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。

メモ： この操作は元に戻せません。

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 アナライザが使用するハンガリアン ネーム セットの作成、編集、複製、削除を行います。**[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。

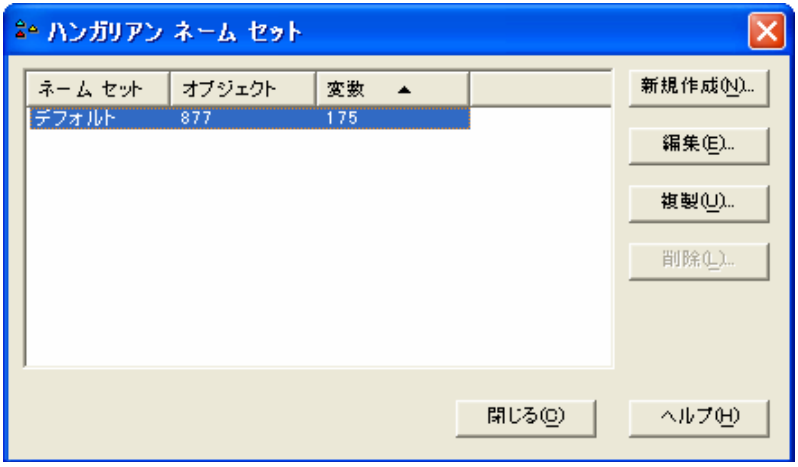


図 3-23. [ハンガリアン ネーム セット]ダイアログ ボックス

ハンガリアン ネーム セットの作成

新しいハンガリアン ネーム セットを作成するには、以下の操作を行います。

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 ボックスが表示されます（図 3-24（109 ページ））。

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で置き換えます。

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]**、**[削除]**の各ボタンが有効になります。

4 この新しいネーム セットに適用する言語を選択します。
選択肢には**[Visual Basic .NET]**と**[Visual C# .NET]**があります。言語を選択すると、新しい名前が検証されます。

図 3-24. **[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**

ハンガリアン ネーム セットの編集

既存のハンガリアン ネーム セットを編集するには、以下の操作を行います。

1 **[ハンガリアン ネーム セット]**ダイアログ ボックスのネーム セットを選択します（図 3-23（108ページ）を参照）。

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 ボックスが表示されます。**[ハンガリアン ネーム セットの編集]**ダイアログ ボックスは、**[ハンガリアン ネーム セットの新規作成]**ダイアログ ボックスと似ています（図 3-24（109ページ）を参照）。

3 ネーム セットに合わせて、関連付ける言語、変数、オブジェクトを編集します。

メモ： 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 セットの複製

ハンガリアン ネーム セットを複製してテンプレートとして使用し、新しいネーム セットを作成することができます。ハンガリアン ネーム セットを複製するには、以下の操作を行います。

1 **[ハンガリアン ネーム セット]**ダイアログ ボックスのネーム セットを選択します（図 3-23（108 ページ）を参照）。

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 ボックスが表示されます（図 3-25 を参照）。

3 一番上のフィールドの「**コピー～<name>**」を固有の名前で置き換えます。

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]**、**[削除]**の各ボタンが有効になります。ネーム セットが検証されます。

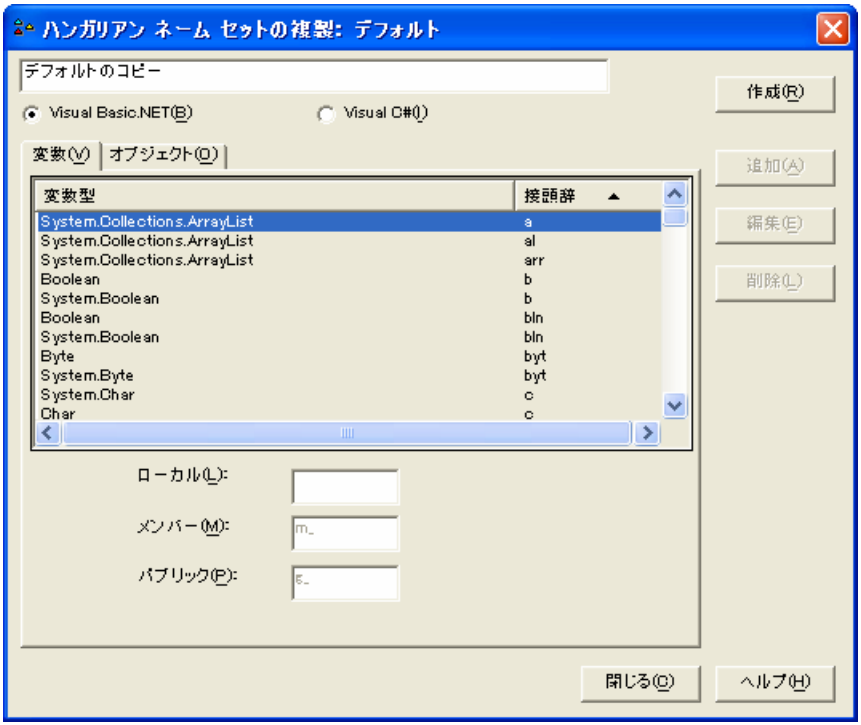


図 3-25.　[ハンガリアン ネーム セットの複製]ダイアログ ボックス

ネーム セットの削除

ハンガリアン ネーム セットを削除するには、以下の操作を行います。

メモ：　削除できるのは、現在ルール セットに使用されていないユーザー定義のハンガリアン ネーム セットのみです。

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 ボックスが表示されます（図 3-23（108ページ）を参照）。
2. 削除するネーム セットを選択します。
   各タブのペインでは、そのハンガリアン ネーム セットと関連付けられたすべての変数とオブジェクトが強調表示され、**[追加]**、**[編集]**、**[複製]**の各ボタンは無効になります。
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 ボックスが表示されます。
4. 選択したネーム セットを削除するには、**[OK]**をクリックします。
   [ハンガリアン ネーム セット]ダイアログ ボックスが再表示されます。

メモ：　この操作は元に戻せません。

5. **[OK]**をクリックします。

ルール リストの操作

ルール リストに表示されるルールを操作する方法は2つあります。

ルール リスト ビューのフィルタ

ルール マネージャ ウィンドウの左側にあり、**[ルール セット]**リストの下に表示される**[フィルタ]**タブを使用すると、**ルール リスト** ペインに表示する内容をフィルタできます（図 3-20（103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表示されます。選択内容をフィルタするには、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（図 3-26（112ページ））。

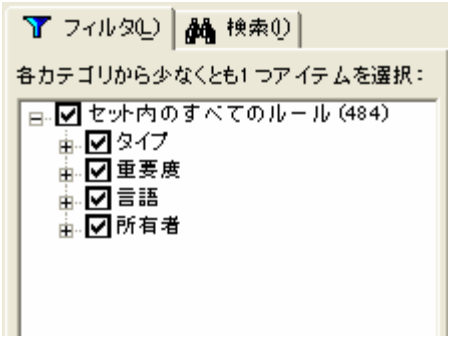


図 3-26.　[フィルタ]タブと[検索]タブ

3　**[フィルタ]**オプションの各グループから1つ以上の項目を選択(または選択解除)します。
グループ チェック ボックスを選択するか、[+]をクリックして展開し、グループ内から個々に選択します。

各グループから1つ以上の項目を選択する必要があります。選択していない場合、注意が必要な領域にマウス ポインタが移動します。以下のグループがあります。

- ◇ **[タイプ]**－ルールはプログラミング テクノロジと一致します。
- ◇ **[重要度]**－選択肢には、**[高度]**、**[中]**、**[低]**、**[警告]**があります。**[警告]は、特定のコーディングの問題に注意を向けるために使用されます。**
- ◇ **[言語]**－選択したルール セットに適用される言語のみが表示されます。
- ◇ **[所有者]**－DevPartner付属のルールにはDevPartnerという識別子が付けられています。他のすべてのルールは、そのルールを作成した個人が所有者になります。選択したルール セットに適用される所有者のみが表示されます。

4　**[適用]**をクリックして、現在のビューをフィルタします。

特定のルールの検索

ルール マネージャ ウィンドウの左側にあり、**[ルール セット]**リストの下に表示される**[検索]**タブを使用すると、1つまたは複数のルールを検索できます。

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 データベース内のすべてのルールが表示されます。

2　**[検索]**タブをクリックすると、検索オプションが表示されます（図 3-26（112 ページ）を参照)。

3　**[ルール セットの検索]**リストから基準を選択します。

ヒント: **[内容]**リストから最近使用した検索文字列を選択することもできます。

4　特定の検索条件を定義するには、**[内容]**に文字列を入力します。

5　**[検索]**をクリックします。

続けて検索を実行するには、**[検索場所]**オプションのいずれかを選択します。

- **[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

上記のように、オプションで検索基準を変更し、もう一度**[検索]**をクリックすることもできます。

正規表現を使用した新しいルールの作成

コード レビューに独自のルールを作成し、それを使用して問題がありそうなコーディングを特定することもできます。DevPartner コード レビューのルールでは正規表現をさまざまに利用できます。これにより、堅牢で汎用的なテキスト検索の手段が提供されます。

正規表現は広く使用されており、ドキュメントも豊富で、HTML、Visual Basic、Visual C# の構文内のパターン照合に使用できます。DevPartner コード レビューは Microsoft Visual Studio と同じ正規表現エンジンを使用し、同じ構文をサポートしています。

DevPartner コード レビューでは、ルールのスコープをコードの特定の部分に制限できるため、ルールで簡単に正規表現を使用できます。たとえば、ルールは、ファイル全体、メソッドのみ、または`While`ブロックのみに適用できます。ルールでスコープを指定できるため、正規表現ではコードの対象となる部分だけに焦点を当てることができます。

また、コード ブロックからコメントを削除して、正規表現検索を補助することもできます。レビュー前にコメントを削除すると、不正確な結果が減ります。

以降のセクションでは、実際のコード レビュー ルールの例と、それに使用できる正規表現について説明します。

メモ：　コード レビュー ルールに正規表現を記述する方法の詳細については、以下の参考文献を参照してください。

- ◇ Forta, Ben 著『Teach Yourself Regular Expressions in 10 Minutes』インディアナ：Sams Publishing（2004 年）
- ◇ Friedl, Jeffrey E.F 著『Mastering Regular Expressions』（第 2 版）カリフォルニア：O'Reilly（2002 年）
- ◇ Goyvaerts, Jan 著『Regex Tutorial, Examples and Reference』（2006 年 2 月 1 日）<http://www.regular-expressions.info>
- ◇ Microsoft Corporation 「.NET Framework の正規表現」<http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/cpguide/html/cpconCOMRegularExpressions.asp en-us/cpguide/html/cpconcomregularexpressions.asp>

90 文字を超える行の検索

ベスト プラクティスのコーディング標準では、コードの 1 行は 90 文字以内にすることが推奨されています。コード レビューのルールでは、90 文字を超える行を検索することでこの標準を適用しています。以下の正規表現で、行の文字数が 90 文字を超えないようにすることができます。

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

この正規表現では、まず**単一行**オプションを **False** に設定して、改行文字（¥n）の前までのすべての文字列を 1 行として評価します。この評価では、開始文字から改行文字（¥n）までの各コード行を別個の行として扱います。

次に、正規表現の最も基本的な機能である単一文字の照合が指定されています。このルールでは、ピリオド（.）のメタ文字を使って、任意の単一文字を照合します。

ピリオド（.）のあとに繰り返し照合のメタ文字 {91,} が続きます。繰り返し照合のメタ文字は、指定文字が特定回数繰り返されること、または特定の範囲の繰り返し回数繰り返されることを示します。このルールの式が true になるのは、任意の単一文字が 91 回以上繰り返された場合だけです。範囲の 2 つめの値は空になっています。これは、このルールでは文字が 90 文字を超えるかどうかだけを問題としているからです。基本的な繰り返し照合のメタ文字を表 3-18 に示します。

表 3-18.　繰り返し照合のメタ文字

文字	意味
+	前の文字の 1 つまたは複数の繰り返しと一致します。
*	前の文字が存在しないか、任意の数の繰り返しと一致します。
?	前の文字が存在しないか、1 つだけ存在する場合に一致します。
{*n*} {2,6} {*n*,}	前の文字が指定した回数だけ繰り返された場合に一致します。*n* は繰り返し回数を表します。 この中かっこは、2 ～ 6 回など、繰り返し回数の範囲を指定するときにも使用されます。上限と下限をカンマで区切って指定します。 上限を省略すると、上限なしで最小限の繰り返し数を満たすものが一致します。

スペースの代わりに使用されたタブの検索

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。タブが表すスペース数はエディタによって異なるため、その違いによってソース コードの表示がエディタによって変わる可能性があります。ソースの表示を統一するためには、スペースを使用する必要があります。以下の正規表現は、メソッド内に使用されているタブを検索するときに使用されます。

```
(?s)¥t.*
```

この正規表現では、**単一行**オプションを **True** に設定して、改行文字（¥n）までの各行のすべての文字を単一行の一部として評価します。

次に、メタ文字（¥t）を使用してタブ文字を検索します。これだけでは、この正規表現でメソッド内のすべてのタブ文字が検索されてしまいます。行のインデントなどのためにメソッドに複数のタブが使用されている場合は、そのメソッドの各タブについてルールが適用されてしまいます。これはこのルールが意図している動作ではありません。

このルールは、メソッドで最低1つのタブが使用されている場合にtrueと評価されるべきで、メソッド内にタブが検出されるたびにtrueになるべきではありません。このためには、ピリオド（.）と、その後ろに0以上のインスタンスを指定する繰り返し照合のメタ文字を付ける必要があります。表3-18（114ページ）に示すように、使用する繰り返し照合のメタ文字はアスタリスク（*）です。末尾にこの2文字のメタ文字を追加すると、タブ文字が最初に検出されたときにtrueと評価され、そのメソッド内の以降のすべての文字がキャプチャされます。

System.Exceptionを受け取るコードの検索

エラーを処理するためにSystem.Exceptionを受け取るようにすることは避けてください。これは、エラー タイプを適切に区別するために十分な詳細レベルでエラーが取得されないためです。エラー処理のコード ブロックでは、可能なかぎり細かい粒度でエラーをインターセプトし、処理する必要があります。そうすることで、プログラムはより堅牢になり、クラッシュの可能性が低くなります。以下の正規表現は、コードでSystem.Exceptionを受け取るVisual Basic構文が使用されている箇所を検索するために使用されます。

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

この表現の最初の部分で、コード内に含まれるCatchというリテラルが検索されます。このルールでは、Catchが長い単語の先頭部分として使用されている場合は除外すべきです。そのため、リテラル テキストの後ろに、空白スペース（¥s）のメタ文字を付けます。

Visual Basic構文では、Catchという単語の後ろに変数名が続きます（例外オブジェクトを保持するために使用されます）。この変数の後ろには、空白スペースとリテラル「As」が続きます。

このルールの正規表現では、有効な変数名、次に空白スペース、次に「As」という単語が続く箇所を検索する必要があります。メタ文字¥wは、繰り返し照合のメタ文字+と組み合わせて、英数字（大文字または小文字）またはアンダースコア文字の1つまたは複数のインスタンスを検索します。¥sAs¥sを追加すると、有効な変数名、次に空白スペース、次に単語「As」が続く箇所の検索が完了します。

ここまででではこの正規表現で、以下のようなコードが検索されます。

```
Catch MyExceptionObject As
```

この正規表現を使用すると、例外を受け取るすべてのコードが検索されるはずです。ただし、このルールではSystem.Exceptionを受け取るコードだけを検索する必要があります。そのため、さらにこの正規表現を精巧にする必要があります。

正規表現が System.Exception を受け取るコードだけに一致するようにするには、ピリオドで区切られた System と Exception というリテラルを検索します。ピリオドはメタ文字なので、円マークを前に付けて特殊文字のステータスを除外する必要があります。

正規表現の一部に System¥.Exception を指定しても、まだ問題があります。System.Exception を受け取るための構文では、System. を省略して Exception だけを使用することができます。そのため、最後に System. の一致をオプションにする変更が必要です。System¥. をかっこで囲むとこれが副次式になります。この後ろに、0 以上の一致を指定する ? メタ文字を続けます。

複数のリターン ポイントがあるメソッドの検索

ベスト プラクティスのコーディング標準では、メソッドにリターン ポイントを 1 つだけ含めることが推奨されています。複数のリターン ポイントがあるとコードがわかりづらくなる可能性があります。以下の正規表現は、コード レビューのルールで、メソッドに複数のリターン ポイントがあるインスタンスを検索するときに使用されます。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されたものですが、確認が必要な点がいくつかあります。

```
(?s)(¥breturn¥b.*){2,}
```

まず、このルールでは、メソッド全体にフォーカスするように、(?s) を使用して**単一行**オプションを true に設定しています。メソッドをひとかたまりとして扱うには、改行文字 (¥n) までの各行のすべての文字を単一行の一部として評価する必要があります。

前述のルールで使用した正規表現では、末尾に繰り返し照合のメタ文字が使用されています。この式では {2,} を使用して前述の副次式（かっこの内容）を変更します。この場合、メソッド内に 2 つ以上の一致が存在する必要があります。

(¥breturn¥b.*) という副次式は、この正規表現でほとんどの処理を実行する部分です。繰り返し照合のメタ文字でブロック全体を変更できるように、副次式として記述しています。メタ文字 ¥b は単語の境界です。リテラル テキスト return を単語の境界のメタ文字で囲むことで、長い単語の一部として使用されている return ではなく、return 単体が検索されます。

メモ： **前述の例では、リテラル テキスト Catch の後ろに空白スペースのメタ文字 ¥s を続けて、Catch そのものが単語であるインスタンスのみが検索されるようにしました。これは柔軟な正規表現を示すよい例です。このルールに単語の境界のメタ文字 ¥b を使用することもできましたが、使用しませんでした。**

副次式内の最後の .* で、任意の文字の 0 インスタンス以上の一致が検索されます。これでルールは完成です。return、次に 0 または任意の数の文字が続くインスタンスが 2 つ以上あるかどうかがメソッド全体で検索されます。

定義時の変数の初期化の強制

ベスト プラクティスとして、コードの簡潔さとわかりやすさを維持するには、変数を定義時に必ず初期化する必要があります。以下の正規表現は、コード レビューのルールで、変数が定義されていても初期化されていないインスタンスを検索するために使用されます。

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

コードの各行をそれぞれ別個に評価する必要があるため、最初に必要な操作は**単一行**オプションをfalseに設定することです。

次に、Dimという単語を検索します。リテラル テキストDimを単語の境界のメタ文字¥bで囲み、1単語としてのDimのみを検索するようにします。

副次式で、先読みまたは後読みの概念を実装します。先読みまたは後読みを行う正規表現の機能によって、さらに柔軟性が高くなります。

先読みまたは後読みとは、正規表現の副次式で一致を検索することです。副次式では、単純な文字列照合のように指定したテキスト自体の一致を検索して返すのではなく、一致が存在することだけを検証します。一致が見つかると副次式がtrueと評価され、正規表現の残りの部分が他の修飾子に従って評価されます。このような先読みと後読みは、一致するテキストが検出されたときに副次式がtrueと評価されるため、**肯定の先読み**または**肯定の後読み**とも呼ばれます。

肯定の先読みと肯定の後読みの構文を以下に示します。

- 肯定の先読み：(?=**副次式**)
- 肯定の後読み：(?<=**副次式**)

同様に、**否定の先読み**と**否定の後読み**は、ステートメントに指定した副次式に一致しないテキストを検索します。

否定の先読みと否定の後読みの構文を以下に示します。

- 否定の先読み：(?!*subexpression*)
- 否定の後読み：(?<!*subexpression*)

このルールの正規表現には、Dimキーワードの右に（先行するスペースの有無にかかわらず）等号（=）がないことを確認する否定の先読み構造を使用する必要があります。副次式(?!.*=)は、この否定の先読みを処理します。

式の末尾部分にある(?!.*¥bnew¥b)では、否定の先読みをもう1つ使用して、Dimキーワードの右に（先行するスペースの有無にかかわらず）newという単語が存在しない場合にtrueと評価しています。

これで、Dimの後ろに等号（=）またはnewという単語がないコードの行が検出されるとtrueに評価される、ルールの正規表現が完成しました。

1行に複数のステートメントがあるインスタンスの検索

コードの読みやすさと保守性を高めるためには、1行に入れるステートメントは1つにする必要があります（ただし、ループ構文は除きます）。以下の正規表現は、コード レビューのルールで、1行に複数のステートメントが含まれるインスタンスを検索するために使用されます。

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

ある行に複数のステートメントがあることを検出するには、最も簡単な方法として、行に複数のセミコロンが含まれるかどうかを調べればよいと思われるかもしれません。実際、この検索はこのルールの正規表現の基本部分となりますが、for キーワードと関連付けられているセミコロンの可能性も考慮に入れる必要があります。

キーワード for がセミコロンに関連付けられているインスタンスを除外するには、否定の後読み構造 (?<!for.*) を使用して、セミコロンがある行を後読みし、for という単語が存在しないことを確認します。この正規表現の残りの部分（;.*;）で、セミコロンの後ろに、任意の数の他の文字、次にもう1つのセミコロンが続く文字列が検索されます。

開始の中かっこが別の行に配置されていることの確認

コーディングのベスト プラクティスでは、開始の中かっこは、ブロックを開始するステートメントの次行の行頭に配置することが推奨されています。以下の正規表現は、コード レビューのルールで、開始の中かっこが別の行に配置されていないインスタンスを検索するために使用されます。

```
(?m)^¥s*¥w+(?=.*?¥{).*?$
```

この正規表現には、いくつかの新しい概念が含まれます。このルールでは、まず (?m) で**複数行**オプションを true に設定します。この設定で、**行頭**（^）と**行末**（$）の2つのメタ文字の動作が変わります。複数行オプションを有効にすると、^ と $ は、検索対象の文字列全体ではなく、各行の行頭と行末を取得します。

複数行モードを有効にすると、この正規表現で、行頭（^）の後ろに1つまたは複数の空白スペース文字（¥s*）、次に1つまたは複数の単語文字（¥w+）が続く文字列が検索されます。これで正規表現の基礎が作成されました。1つまたは複数の単語文字が検索された場合、その行には開始の中かっこは存在すべきではありません。

肯定の先読みの副次式 (?=.*?¥{) では、任意の文字に開始の中かっこが続く文字列が各行で検索されます。中かっこの前の円マークによって、メタ文字のステータスが除外されます。文字の後ろに開始の中かっこが続く行が検出され、開始の中かっこが独立した行にないことがわかると、正規表現の末尾にある .*? で、行末（$ メタ文字で検索）までの残りのテキストが取得されます。

ループ本体内でループ カウンタが変更されないことの確認

ループ本体内でループ カウンタを変更すると、予期しない結果が発生し、コードがわかりづらくなる可能性があります。以下の正規表現は、コード レビューのルールで、ループ カウンタがループ本体内で変更されているインスタンスを検索するために使用されます。

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

ルールを実施するために必要な処理が多いため、この正規表現は非常に長くなります。ループ カウンタの変数名を識別して格納するために、この正規表現ではまず、for キーワード、左かっこ、ループ カウンタの型を取得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 キーワードの後ろに、任意の数の空白スペース文字、1 つの左かっこ、さらに空白スペース文字、1 つまたは複数の単語文字、さらに空白スペースが続く行を検索し、最後に副次式を使用して変数名を取得しています。

副次式（?<VARNAME>¥w+）では、ループ カウンタの名前を取得し、変数 VARNAME に格納します。

メモ：　この構造には、句読点を含む変数名や数字から始まる変数名でなければ、任意の変数名を使用できます。

ループ カウンタの名前を取得したあと、正規表現の後半で残りの文字と右のかっこを取得します。次に、以下の構造を使用して、ループ カウンタのインスタンス、次に等号が続く文字列がないか、ループ本体を検索します。

```
.*¥b¥k<VARNAME>¥b¥s*=
```

正規表現のこの部分が、このルールの基本部分です。この時点までに、ループ カウンタ名が判断され、ループ本体の検索のスコープが設定されました。正規表現の最後の部分は、VARNAME（ループ カウンタ）の値までのすべての文字に一致し、カウンタの後ろに等号が続く文字列が検索されます。

ループ カウンタの後ろに等号が続くということは、何らかの値に設定されているか、変更されたことを示します。ループ本体内ではカウンタを変更すべきではないので、違反が検出されます。

Visual Studio Team System へのデータの送信

Microsoft Visual Studio Team Explorer クライアントがインストールされ、Team Foundation Server の接続が使用可能になっている場合に、DevPartner Studio は Microsoft Visual Studio Team System をサポートします。

DevPartner コード レビューの Visual Studio Team System サポート

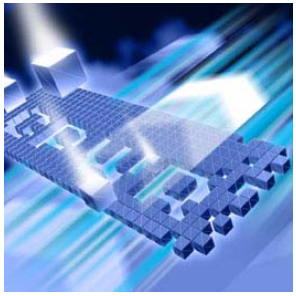
コード レビュー セッション ファイルの以下のタブで選択した項目のデータは、**バグ** タイプの**作業項目**として Visual Studio Team System に送信できます。

- [問題] タブ（「コード違反の表示」（77 ページ）を参照）
- [ネーミング] タブ（「ネーミング違反の表示」（79 ページ）を参照）

バグを送信すると、タブのデータが**作業項目**フォームにコピーされます。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」（8 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



- カバレッジ分析の機能
- すぐにカバレッジ分析を使用するには
- プロパティとオプションの設定
- インストゥルメンテーションについて
- さまざまなアプリケーションからのデータの収集
- セッション データのマージ
- カバレッジ データのエクスポート
- データ収集の制御
- コマンド ラインからの分析
- カバレッジ分析ビューアの使用
- DevPartner エラー検出との統合
- Visual Studio Team System へのデータの送信

この章には 2 つのセクションがあります。1 つめのセクションには、はじめてのユーザーがカバレッジ分析機能を利用できるように、簡単な操作手順が記載されています。2 つめのセクションには、DevPartner Studio のカバレッジ分析機能を詳しく理解するための参考情報が記載されています。

カバレッジ分析に関するその他のタスクに基づく情報については、DevPartner Studio のオンライン ヘルプを参照してください。

カバレッジ分析の機能

開発者とテスト エンジニアは、DevPartner Studio のカバレッジ分析機能を使用するとアプリケーション コード全体を確実にテストできます。カバレッジ分析を使用してテストを実行すると、テストでカバーされたすべてのコンポーネント、イメージ、メ

ソッド、関数、モジュール、コードの各行が追跡されます。テストが終了すると、テストされたコードとテストされなかったコードに関する情報が表示されます。

Web や ASP.NET のアプリケーションなどのマネージ アプリケーションと、アンマネージ（ネイティブ）の Visual C++ アプリケーションと Visual Basic アプリケーションの両方について、カバレッジ データを収集できます。

すぐにカバレッジ分析を使用するには

以下の準備、設定、実行手順では、DevPartner を使用してコード カバレッジを分析する方法を紹介します。

機能をすぐに使用したい場合は、色付きの枠内に記載されている手順に従ってください。色付きの枠内に記載されているトピックの詳細な情報については、枠の下に記載されている文章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：分析内容の検討

コード カバレッジを使用する前に、分析内容を検討します。

この手順では以下を前提としています。

- Visual Studio 2003 または 2005 で作業しています。
- シングルプロセスのマネージ アプリケーションをテストしています。
- アプリケーションのビルドと実行が可能です。
- ソリューションにスタートアップ プロジェクトが含まれます。

メモ： DevPartner カバレッジ分析でサポートされているプロジェクト タイプのリストについては、「DevPartner Studio でサポートされるプロジェクト タイプ」（321 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場合、カバレッジ セッションを開始する前に、収集対象のデータを決定します。場合によっては、セッションの開始前に必要な手順があります。たとえば、以下の場合、何らかの設定が必要です。

- カバレッジ分析から除外するモジュールがある場合
- 分析対象にするアンマネージ モジュールがある場合
- リモート サーバーで実行されているコードを含める場合

この手順では、アプリケーションに含まれるローカルのマネージ コードがすべて分析されます。

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、データ収集を制限するために、いくつかのプロパティとオプションを設定します。

この手順では、デフォルトの DevPartner のプロパティとオプションを使用できます。その他の設定は必要ありません。

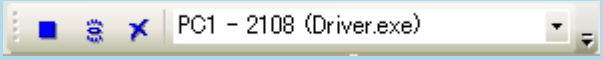
ソリューション プロパティとプロジェクト プロパティを使用すると、アプリケーションの外部で実行されている .NET アセンブリと COM の情報を、分析セッションデータに含めるかどうかを選択できます。DevPartner のオプションを使用すると、表示オプションを変更したり、アプリケーションの一部を分析から除外したり、データ収集管理のためにセッション コントロール ファイルを作成したりできます。設定のカスタマイズの詳細については、「プロパティとオプションの設定」（129 ページ）を参照してください。

実行：カバレッジ データの収集

分析する内容を検討し、適切なプロパティとオプションを設定すると、カバレッジデータを収集する準備が整います。

1. Visual Studio で、アプリケーションに関連付けられている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、アプリケーションの任意の部分にフォーカスしてカバレッジ分析を行えます。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たり、現在収集されているデータのスナップショットを作成してから記録を続行したり、収集しただけでスナップショットに保存していないデータをクリアしたりできます。
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  (必要に応じて2回クリックしてセッション ウィンドウにフォーカスを移動します)。スナップショットを作成すると、収集データを含むファイル(セッション ファイルと呼ばれます)が作成され、セッション ファイル データが表示されます。

5 アプリケーションに戻り、テストの実行を継続します。

6 テストの実行が完了したら、アプリケーションを終了します。最後のセッション ファイルがVisual Studioに表示されます。

メモ： マネージ アプリケーションのデータを収集するときにセキュリティ エラー メッセージが表示される場合、セキュリティ ポリシーの変更方法について **133ページを参照してください。**

カバレッジ分析はDevPartnerエラー検出機能と併せて使用できます。テストでどのくらいのコードがカバーされたかがわかると、エラー検出データの包括性を評価するうえで役立ちます。エラー検出とカバレッジ分析の両方を使用してセッションを実行する方法については、「DevPartnerエラー検出との統合」(149ページ)を参照してください。

データの分析

スナップショットを作成するか、またはアプリケーションを終了すると、図4-1(125ページ)のようにVisual Studioにセッション ファイルが表示されます。セッション ウィンドウは以下の要素で構成されます。

- 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がリストされ、それぞれでカバーされた行がファイルの合計行数に対する割合で表示されます。
- セッション データ ペイン。3つのタブと2つのカバレッジ メーターがあり、フィルタ ペインで選択した項目のデータが表示されます。
- 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ます。フィルタ ペインで選択した項目に対して、行と関数のカバレッジが要約されます。

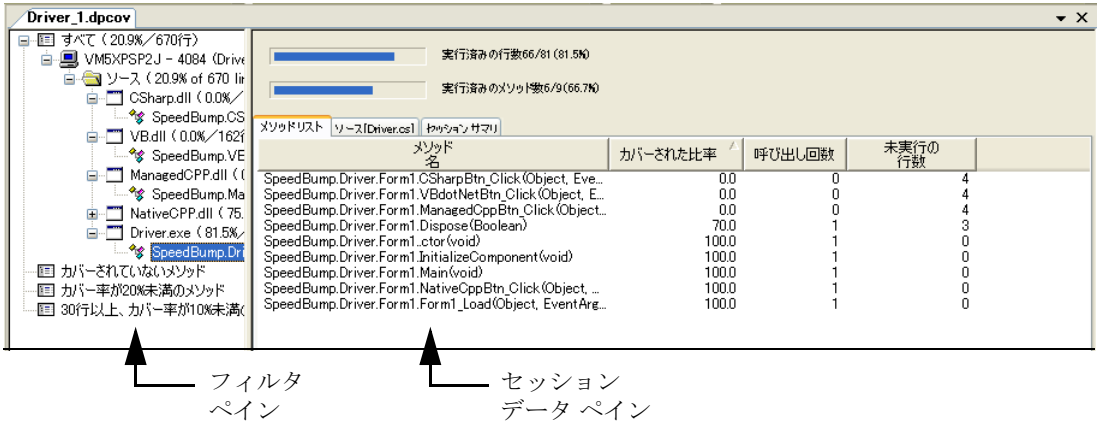


図 4-1.　　カバレッジ分析セッション ウィンドウ

フィルタ ペインとセッション データ ペインの使用

フィルタ ペインには、アプリケーションのファイルとイメージの他に、最も重要なデータにフォーカスするために使用するフィルタのセットも表示されます。

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減らすことから始めます。次に、**メソッド リスト**を確認して、テストのカバー率が最も低いメソッドを確認します。

1. **フィルタ** ペインの**[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。これで表示データが少なくなるので、最も実行率が少なかったメソッドに集中できます。
2. **[メソッド リスト]**タブのデータで、各メソッドのどのくらいがテストで十分にカバーされたかを確認します。

   アプリケーションの一部が十分にカバーされていない場合は、アプリケーションの機能をより多くカバーするようにテストを修正できます。

ソース コードの参照

[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]タブは、テストでさらにカバーする必要がある機能を特定するときに役立ちます。

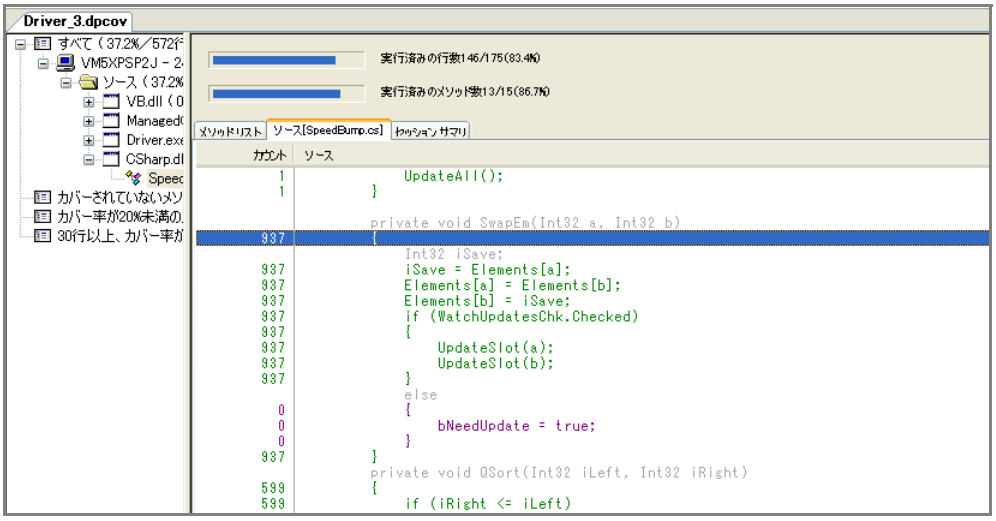


図 4-2.　[ソース コード]タブ

ソース ファイル内の特定のメソッドのコードを表示するには、**メソッド リスト**のメソッドをダブルクリックします。

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 のように、そ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]タブには各コード行のカバレッジ データが表示されます。実行された行（デフォルトで緑色）、未実行の行（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、行の実行回数が表示されます。

メモ：　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）情報が必要です。

[ソース]タブの行を右クリックして表示されるコンテキスト メニューから、前の未実行の行または次の未実行の行へ移動したり、表示するカラムを選択したり、表示する別のソース ファイルを選択したりできます。

セッション サマリ データの表示

[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。

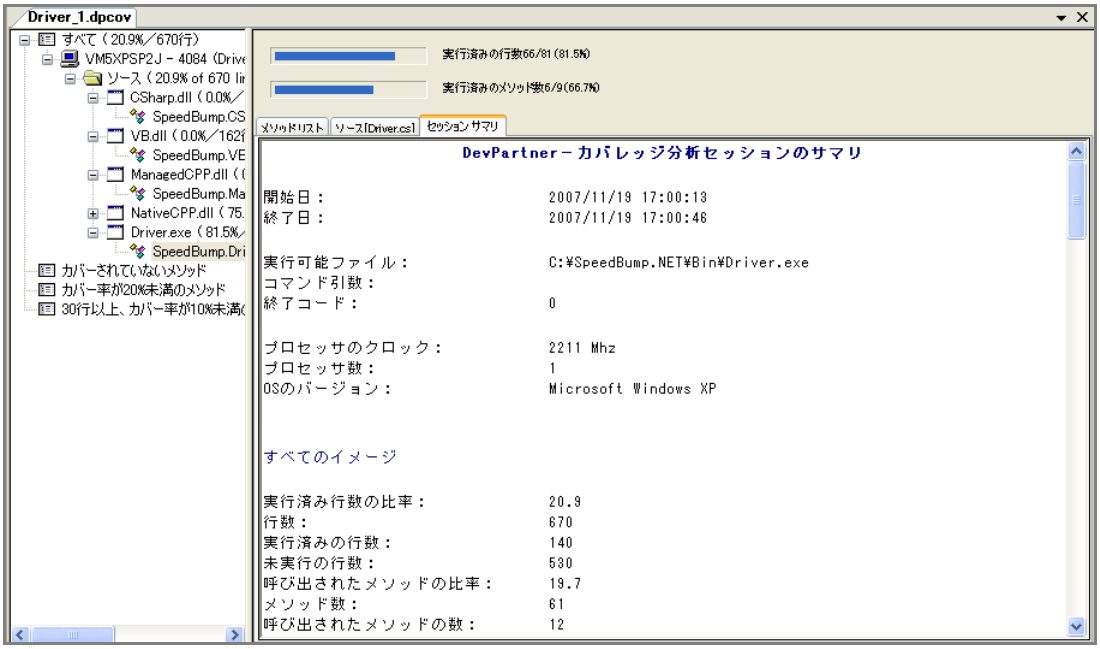


図 4-3.　[セッション サマリ]タブ

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、セッションの状況に関する情報が表示されます。たとえば、セッションの日時、プロセッサ速度、オペレーティング システムなどです。この情報は、古いセッション ファイル、特に他の人が作成したセッション ファイルを表示したときに便利です。

サマリにはフィルタ ペインと[メソッド リスト]タブのカバレッジ データも含まれ、分析したファイルとメソッド両方のデータが表示されます。

5　タブ内をスクロールすると、セッション サマリ データが表示されます。

セッション ファイルの保存

カバレッジ データの確認が終わったら、セッション ファイルを保存できます。複数のセッション ファイルを作成した場合、複数のセッション ファイルからカバレッジ データをマージできます。

1. Visual Studio のセッション ファイル ウィンドウを閉じると、セッション ファイルが保存されます。メッセージが表示されたら、デフォルトのファイル名と保存場所をそのまま使用します。デフォルトでは、ファイルはプロジェクトの出力ディレクトリに保存されます。
2. そのソリューションに複数のカバレッジ セッション ファイルが存在する場合、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確認メッセージが表示されるか、マージが自動的に実行されることがあります。**[セッション ファイルを自動的にマージ]** プロパティの情報については、「ソリューションのプロパティ」（129 ページ）を参照してください。

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。保存されたファイルは、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。カバレッジ セッション ファイルの拡張子は **.dpcov** です。

デフォルトで、セッション ファイルはプロジェクトの出力フォルダに物理的に保存されます。また、デフォルト ディレクトリの内容に基づいて、ファイル名の番号が自動的に 1 つ増えます（たとえば、**MyApp.dpcov**、**MyApp1.dpcov** など）。セッション ファイルをデフォルト ディレクトリとは別の場所に保存する場合は、自分でファイルのネーミングとナンバリングを管理する必要があります。

Visual Studio 2005 の Web サイト プロジェクトなど、出力ディレクトリがないプロジェクトの場合、ファイルはプロジェクト ディレクトリに物理的に保存されます。

コマンド ラインから生成されたセッション ファイルは、自動的にプロジェクトのソリューションへ追加されません。外部で生成したセッション ファイルは、Visual Studio で開いているソリューションに手動で追加します。

この章の準備、設定、実行セクションはこれで終了です。ここまでで、カバレッジ分析セッションの実行方法について基本的な知識が習得できたはずです。詳細情報については、この章の残りを続けて読んでください。また、タスクに基づく情報については 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の設定

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、除外したりするために、データ収集を調整すると便利です。ソリューションのプロパティ、プロジェクトのプロパティ、DevPartner のオプションを使用すると、分析セッションのフォーカスを調整できます。

ソリューションのプロパティ

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。

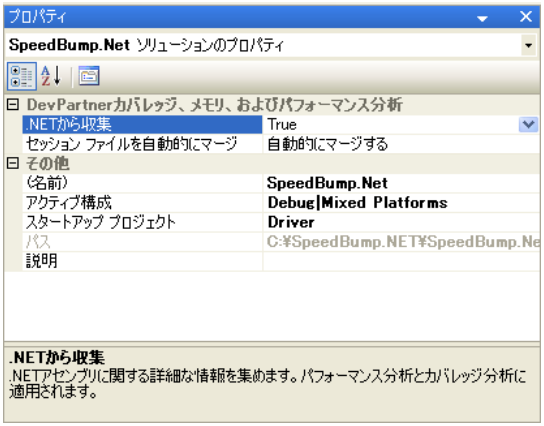


図 4-4.　　ソリューションのプロパティ

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を制御します（「セッション データのマージ」（143 ページ）を参照してください）。
- **[.NET から収集]** ― .NET アセンブリの情報を収集しない場合、このプロパティを [False] に設定します。
- **[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、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メッセージが表示されます。

プロジェクトのプロパティ

プロジェクト レベルのプロパティを確認するには、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、ソリューション内のプロジェクトについて設定できるプロパティを確認します。

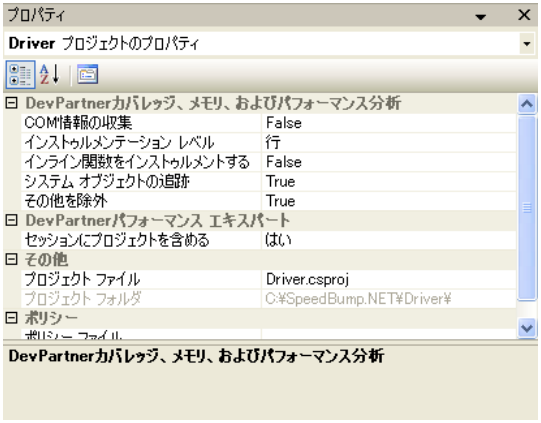


図 4-5. プロジェクトのプロパティ

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]** – DLLのエクスポートと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に関する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]**–マネージ アプリケーションのインライン関数についてのカバレッジ データを常に収集します。
- アンマネージ コードの場合、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。インラインの最適化が有効な場合、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。

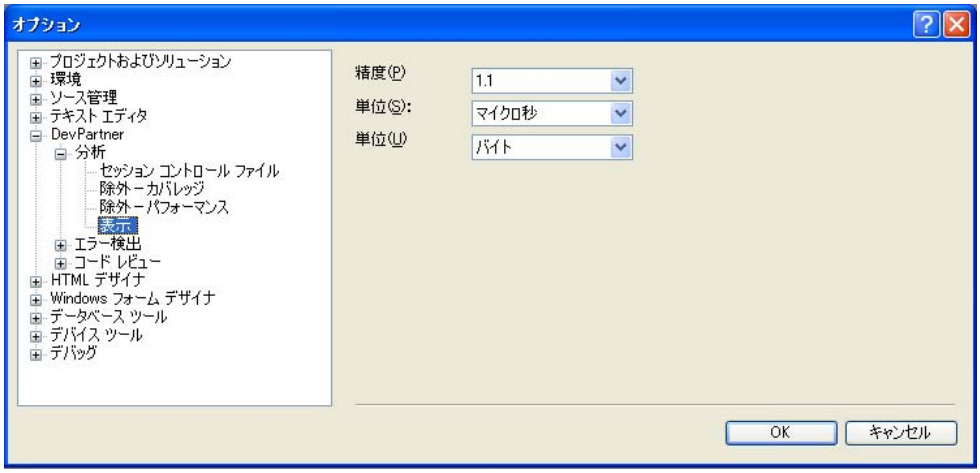


図 4-6.　　分析オプション

- **[表示]**オプションを使用すると、データの表示に使用される精度、位取り、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省略できます。イメージの除外の詳細については、「イメージの除外」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、ルールとアクションのセットを作成し、アプリケーションまたはモジュールの実行時にDevPartnerで収集するデータを制御できます。セッション コントロール ファイルの詳細については、「分析のセッション コントロール」（353ページ）を参照してください。

[環境]>[フォントと色]オプションなど、他のVisual Studioオプションも DevPartner機能に影響があります。

イメージの除外

カバレッジ分析でアプリケーションを実行すると、すべてのソースとシステム イメージのデータが収集されます。ただし、[除外]を使用すると分析から1つまたは複数のイメージを省略できます。

[分析オプション]（**[DevPartner]>[オプション]>[分析]**）の表示中に、**[除外－カバレッジ]**を選択します。

ページの上部にある**[表示]**リストから以下のいずれかを選択します。

- グローバル除外
- 現在のディレクトリ内のローカル除外
- 実行可能ディレクトリ内のローカル除外

[実行可能ディレクトリ内のローカル除外]オプションを使用できるのは、ソリューションが開いていて、実行可能ファイルのディレクトリが現在の作業ディレクトリと異なる場合のみです。

[挿入]  をクリックし、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。除外に指定できるファイルの種類は**.exe**、**.dll**、**.ocx**、.netmoduleです。表示するファイルの種類を制限するには、**[ファイルの種類]**リストを使用します。

.NETモジュール（**.netmodule**）を選択すると、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、**[削除]**  をクリックします。

除外リスト（**nmexclud.txt**）のコピーを別の場所に保存するには、**[保存場所]**をクリックします。グローバル除外は、DevPartnerインストール ディレクトリの**¥Analysis**サブディレクトリの**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現在の作業ディレクトリまたはアプリケーションの実行可能ファイル ディレクトリの**nmexclud.txt**に保存されます。

[ネイティブC/C++インストゥルメンテーション]でコンパイルしたファイルには除外は適用されません。たとえば、インストゥルメントされたアンマネージC/C++イメージを除外しようとしても、そのファイルの情報は収集されます。ただし、システム コール情報は収集されません。アンマネージC/C++イメージをデータ収集から除外するには、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、コンパイラから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。このプロセスはインストゥルメンテーションと呼ばれます。このコードには、アプリケーションの実行中、カバレッジ データの収集に使用される指示が含まれます。DevPartnerのインストゥルメンテーションによって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るだけです。

DevPartnerによって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ユーザーが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ために、DevPartnerはソース コードに直接フックを挿入します。DevPartnerが提供するインストゥルメンテーション マネージャ機能を使うと、使用するインストゥルメンテーションの種類を指定し、インストゥルメンテーションから除外するソリューション内のプロジェクトを指定することができます（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 コードのデータの収集」（134ページ）を参照してください）。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メニューから[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無効にしてから、プロジェクトをリビルドします。

さまざまなアプリケーションからのデータの収集

このセクションでは、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ときにDevPartnerカバレッジ分析を使用する方法について、情報を提供します。

DevPartnerでは、Visual Studioのすべてのマネージ コード言語とアンマネージC/C++をサポートしています。Visual BasicアプリケーションやVisual C++ 6.0アプリケーションのカバレッジ データを収集できるだけでなく、Internet Explorer（IE）またはInternet Information Services（IIS）を使用するとJScriptアプリケーションやVBScript Webアプリケーションのカバレッジ 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言語とプロジェクト タイプの詳細については、「DevPartner Studioでサポートされるプロジェクト タイプ」（321ページ）を参照してください。

マネージ コードからのデータの収集

Visual Studioで開発する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Visual C++アプリケーションのようにマネージ アプリケーションになります。

マネージ アプリケーションのデータを収集しようとしたときに、セキュリティ ポリシーによってコードのDevPartnerインストゥルメントが回避された場合、セキュリティ エラー 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。コードの実行に使用しているポリシーの権限セットからこの権限を削除した場合、またはこの権限を取り消すような厳しいセキュリティ宣言をアセンブリに追加した場合、アセンブリはプロファイルできません。

この状況を修正するには、以下のいずれかの方法で、安全なプロファイルを有効にします。

- 以下のグローバル環境変数を設定し、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
  ```

  この方法で問題を回避できますが、パフォーマンスがやや低下します。

- .NET Framework構成ツールのMMCスナップインを使用するか、アセンブリに含まれる厳しいセキュリティ宣言を一時的に削除することで、アセンブリのポリシーを変更します。

Visual Studio のセキュリティ ポリシーの詳細については、Visual Studio のオンライン ヘルプに含まれる「.NET Framework 開発者ガイド」を参照してください。

アンマネージ コードのデータの収集

カバレッジのプロファイルのために**[ネイティブ C/C++ インストゥルメンテーション]**を使用してアンマネージ Visual C++ アプリケーションをビルドすると、DevPartner とコンパイラが連携して、実行時にカバレッジ データを収集する指示がアプリケーション イメージに追加されます。

アンマネージ コードをインストゥルメントするには、データを収集するアンマネージ 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。

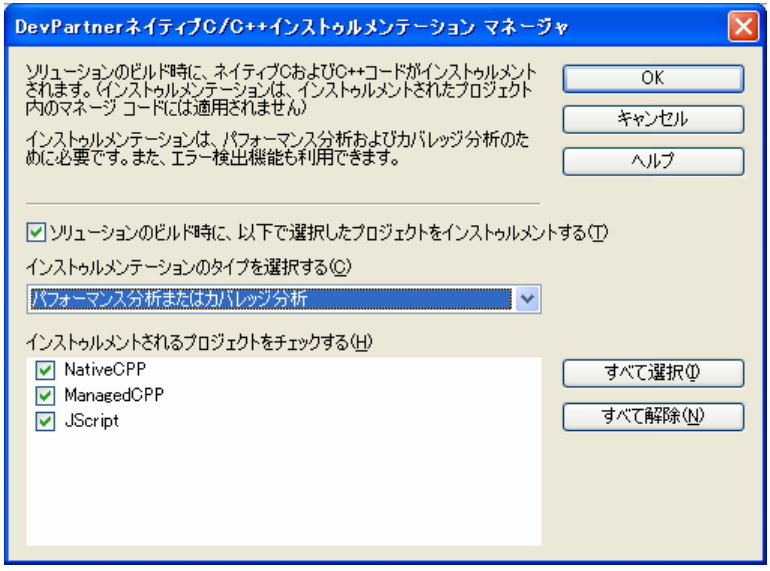


図 4-7. インストゥルメンテーション マネージャ

[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、インストゥルメンテーションの種類を選択します。選択するインストゥルメンテーションの種類は、このあと実行する分析の種類と一致させる必要があります。

インストゥルメントするプロジェクトを選択します。デフォルトでは、ソリューションに含まれる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。省略するモジュールの選択を解除します。

[OK]をクリックし、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]**を選択すると、分析セッションが開始されます。

インストゥルメンテーション マネージャで選択したプロジェクトがソリューションと共に保存されます。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したあと、インストゥルメンテーションのオン／オフを切り替えるには、[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、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]** ボタンを使用します。**インストゥルメンテーション マネージャ**を使用するのは、設定を変更する場合のみです。

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]** オプションをオフにします。次にカバレッジ分析セッションを開始したとき、またはソリューションをリビルドしたときは、インストゥルメントなしでソリューションがリビルドされます。

メモ： **アプリケーションから Visual Studio 6.0 コンポーネントを呼び出している場合、Visual Studio 6.0 でカバレッジ分析のための DevPartner インストゥルメンテーションを使って、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 6.0 で DevPartner Studio のオンラインヘルプを参照してください。**

サポートされない混合コード

Visual C++ では、**/clr** オプションを使用して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が、コードのセクションに **#pragma**（ネイティブ）のマークが付きます。コンパイラは、**#pragma** セクションに定義されているすべてのメソッドについて、ネイティブ コードを生成します。DevPartner では混合モードの C++ ファイルをサポートしていません。マネージ セクションとアンマネージ セクションの両方を含むプログラムをプロファイルしようとすると、マネージ コード部分についてのみカバレッジ データが収集され、アンマネージ コード部分についは収集されません。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 コードを別のファイルに配置し、そのファイルをインストゥルメントします。詳細については、「アンマネージ コードのデータの収集」（134 ページ）を参照してください。

DevPartner for Visual C++ BoundsChecker Suite での混合コード

DevPartner for Visual C++ BoundsChecker Suite を使用して、マネージ プロジェクトとアンマネージ プロジェクトが含まれるソリューションを開くと、[DevPartner] メニューの以下のコマンドが使用できなくなります。

カバレッジ分析を選択して開始
エラー検出とカバレッジ分析を選択して開始
デバッグを実行せずにパフォーマンス分析を選択して開始
デバッグを実行せずにメモリ分析を選択して開始
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

Visual Studio 内からアンマネージ コードを分析するには、混合コード ソリューションからアンマネージ プロジェクトのみを含む新しいソリューションを作成し、**[DevPartner]>[ネイティブ C/C++ インストゥルメンテーション]** を選択してソリューションをリビルドし、分析セッションを実行します。

複数のプロセスからのデータ収集

アプリケーションでは、複数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）、ASP ワーカー プロセス（aspnet_wp または w3wp）が含まれている場合があります。

カバレッジ分析でマルチプロセス 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。

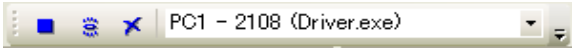


図 4-8.　**プロセス選択リストが含まれるセッション コントロール ツールバー**

このプロセス選択リストを使用して、データ収集のフォーカスを設定できます。スナップショットを作成すると、プロセス選択リストで選択したプロセスのデータを含むセッション ファイルが作成されます。

リモート システムからのデータの収集

DevPartner を使用すると、リモート システムで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側とサーバー側の両方のカバレッジ データを収集できます。DevPartner では、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、クライアントに DevPartner をインストールし、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 ライセンスの詳細については、『DevPartner インストール ガイド』（**DPS Install.pdf**）および『Distributed License Management ライセンス ガイド』（**compuware_licensing_guide_J.pdf**）を参照してください。

メモ：　**ターミナル サービス接続経由で接続するサーバーには、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 デスクトップの使用」（9 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 プロパティがクライアント 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。オプションを変更すると、DevPartner によって、IIS などのサーバー プロセスが再起動されます。再起動は、変更内容を有効にするために必要な処理です。

アンマネージ C++ アプリケーションを分析する場合、必ずインストゥルメンテーションを指定してください。アプリケーションからアンマネージ C++ コンポーネントが呼び出される場合、そのコンポーネントからデータを収集するには、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。詳細については「アンマネージ コードのデータの収集」（134 ページ）を参照してください。

相関データ

IE と IIS をブラウザと Web サーバーとして使用する場合、または COM を使用してプロセス間の呼び出しを行う場合、DevPartner はプロセス間のクライアント／サーバーの関係を自動的に認識します。DevPartner は、DCOM オブジェクトのメソッド間の関係、または HTTP クライアントとサーバー（IE と IIS）の関係を維持するために、そのセッションからのデータを自動的に関連付けます。そして、相関データとクライアントのセッション データが結合されて、1 つのセッション ファイルが作成されます。

相関セッション ファイルには、アプリケーションのクライアント側とサーバー側の両方のカバレッジ データが含まれています。相関セッション ファイルは、他のセッション ファイルと同様に Visual Studio に表示されますが、**appname_CO.dpcov** のようにファイル名に **_co** が付加されます。

COM に基づく関係や、IE と IIS 間のクライアント／サーバー関係がない場合に異なるセッション ファイルからデータを手動で結合するには、**[DevPartner]>[関連付け]>[カバレッジ ファイル]** を選択します。また、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 アプリケーションからのデータの収集

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側とサーバー側の両方のカバレッジ データを収集できます。ローカル マシンまたはリモート マシンで実行されている IIS と ASP.NET のデータを収集するように DevPartner を設定できます。

Web アプリケーションから Visual Basic 6.0 または Visual C++ 6.0 コンポーネントを呼び出す場合、Visual Studio 6.0 で DevPartner コマンド ラインを使用して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。アプリケーションから呼び出されるアンマネージ C++ コンポーネントのデータを収集するには、[ネイティブ C/C++ インストゥルメンテーション] を使用してオブジェクトのインストゥルメントとリビルドを行う必要があります。詳細については、「アンマネージ コードのデータの収集」（134 ページ）を参照してください。カバレッジ分析のためには、必ずインストゥルメントしてください。DevPartner では、1 つのセッションで 1 種類の分析についてのみデータが収集されます。

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。IIS から Web オブジェクトを直接開くと、Visual Studio でプロジェクトを開くときとは異なり、別のソース ファイルが使用されることがあります。最初のソリューションで作成された DevPartner セッション ファイルは、2 回め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を正常に実行するには、以下の条件を満たす必要があります。

- プロジェクトに **web.config** ファイルが含まれる必要があります。
- **web.config** ファイルには、debug 属性を true に設定した compilation エレメントを含める必要があります。次に例を示します。
  ```
  <compilation debug="true"/>
  ```

また、アプリケーションから呼び出されるプロセス内のコンポーネントまたはプロセス外のコンポーネントのデータも収集できます。

デバッグを実行しない ASP.NET アプリケーションの分析

最適な結果を得るためには、カバレッジ分析をデバッグなしで実行します

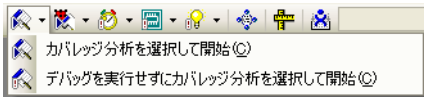


図 4-9. [デバッグを実行せずに ... を選択して開始] オプション

同時にアクティブにできるのは 1 つのスクリプト デバッガのみです。デバッグ機能を使用して Web アプリケーションをデバッグする場合、Visual Studio と DevPartner はスクリプト デバッガをロードしようとします。スクリプト デバッガが IE へのアタッチに失敗したというメッセージが表示されます。エラー メッセージがあってもセッションは中断することなく継続します。

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、デバッグを実行せずにカバレッジ分析を実行します。

予期しない [ファイルの保存] ダイアログとセッション ファイルの保存

場合によっては、ASP.NET アプリケーションの終了後に予期しない **[ファイルの保存]** ダイアログ ボックスが表示されたり、セッション ファイルを自動的に保存するように DevPartner を設定している場合、予期しないセッション ファイルの保存が行われたりすることがあります。

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）などの二次プロセスに関するセッション データも保存されます。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、IE のクライアント データと IIS および ASP.NET ワーカー プロセスのサーバー データの両方を含む、最終的な相関セッション ファイルを生成します。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作成することもできます。

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操作で終了します。ただし、プロファイル中に、ASP.NET ワーカー プロセスが自動的にシャットダウンすることもあります。これは、プロセスを実行しているシステムで、**machine.config** ファイルの processModel Attributes セクションを以下のように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のシャットダウンが発生するくらい低い値に変更した場合
- 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のシャットダウンが発生するくらい低い値に変更した場合
- memoryLimit 属性の値を、セッション中にプロセスのリサイクルが発生するくらい低いパーセント値に変更した場合

プロセスがシャットダウンすると、最後のスナップショットが作成され、セッション ファイルが生成されます。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、そのセッション ファイルが Visual Studio で開かれ、ソリューションに追加されます。この動作は、ASP.NET ワーカー プロセスが実行され、終了するたびに繰り返されます。
- 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作成されると、IE のセッション ファイルと相関セッション ファイルが作成されます。相関セッション ファイルには、この時点までに実行および終了された IE、IIS、および 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が含まれます。

分析セッションが終了すると、DevPartner は **[ファイルの保存]** ダイアログ ボックスを表示するか、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することで余分なセッション ファイルが生成されないように、**machine.config** ファイルを編集し、プロセスが早期に終了しないような高い値に制限属性を設定します。

注意： **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 アプリケーションからのデータの収集

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 と VBScript のソース 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またはプロセス外のコンポーネントを呼び出す場合、そのデータも収集されます。

スクリプト言語のインストゥルメンテーションは、マネージ .NET 言語の場合と同様に、実行時に発生します。ただし、監視するアンマネージ コード コンポーネント（COM オブジェクトなど）は、すべて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 アプリケーションに独自のものです。Visual Studio で開発した Web フォーム、XML Web サービス、ASP.NET アプリケーションのデータを収集するには、他のマネージ アプリケーションを実行する場合と同様に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]>[Compuware DevPartner Studio]>[ユーティリティ]>[Web Script 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、セッション コントロール ツールバーが表示されます。このツールバーを使用してデータの収集を制御できます。

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。セッション コントロール ツールバーを使用すると、アプリケーションを実行しながらデータ収集のフォーカスを制御できます。

IE を終了します。またセッション コントロールを使用している場合、**[停止]** アクションを実行します。セッションの保存を促すダイアログ ボックスが表示され、セッション ファイルが自動的に保存されます。

Web サービスの要件

DevPartner カバレッジ分析で Web サービスを検出する場合、そのサービスは以下の要件の少なくとも 1 つを満たす必要があります。

- Web サービスは `System.Web.Services.WebService` 基本クラスから派生する必要があります。
- Web サービスには `WebService` 属性が含まれる必要があります。

DevPartner カバレッジ分析で Web メソッドを検出する場合、そのメソッドには `WebMethod` 属性が含まれる必要があります。

NMSource からの一時ファイルの削除

IE または IIS でスクリプトのカバレッジを分析している間、スクリプト ソースの一時的なコピーを保持するために **NMSource** ディレクトリが作成されます。セッション データの分析中、このソースはセッション ウィンドウの [ソース] タブに表示されます。

このソースはいつか必要になることもあるため、DevPartner は **NMSource** からこのファイルを削除しません。特に IIS でサーバー プログラムを分析している場合、このディレクトリのサイズは急速に増える可能性があります。

定期的に **NMSource** ディレクトリのソース ファイルを確認し、アクティブでないプロジェクトに関連するファイルを削除してください。**NMSource** は **¥Program files¥Internet Explorer** ディレクトリにあります。

データ収集のための IIS の設定

リモート サーバーで実行されている IIS アプリケーションまたは 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、以下の設定オプションを設定します。

メモ： **IIS がリモート サーバーで実行されている場合、そのシステムに DevPartner（およびリモート サーバー ライセンス）をインストールし、リモート システムに以下に示す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

以下のオプションを、IIS マネージャの [規定の Web サイトのプロパティ] 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 に適用されます。

[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ているとき、場合によっては、[アプリケーション保護] オプションを変更して DevPartner によるカバレッジ分析データの収集を有効にする必要があります。このオプションを、IIS マネージャの [規定の Web サイトのプロパティ] 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 に適用されます。

[ホーム ディレクトリ] タブまたは [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。

カバレッジ分析のためのInternet Explorerの設定

IEからカバレッジ 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）]**をオフにします。

サービスからのデータの収集

サービスの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／COM+アプリケーションからのデータの収集

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および.NETオブジェクト（COM+）を混合して使用するアプリケーションをプロファイルした場合、アプリケーションの.NET部分については行レベルのデータが収集されます。アンマネージ コードのコンポーネントについては、**[ネイティブC/C++インストゥルメンテーション]**で事前にインストゥルメントしている場合は、行レベルのデータが収集されます。また、Visual 6 COMオブジェクトについても、カバレッジ データ収集用に事前にインストゥルメントしている場合は、行レベルのデータを収集できます。これを行うには、Visual Studio 6で、プロジェクトをカバレッジ分析用のインストゥルメンテーションを使ってビルドします。

Visual Studio 6.0オブジェクト、Visual C++ 6.0オブジェクト、またはインストゥルメントされていない任意のアンマネージ コード コンポーネントをプロファイルした場合、COMインターフェイスとDLLの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 データのマージ

DevPartner でアプリケーションをテストする場合、1 つのセッションですべてのコードを実行することはまず不可能です。複数のセッションから収集されたカバレッジ データを蓄積し、総合的なカバレッジ統計値を分析できるということが重要です。カバレッジ分析を蓄積するために、セッション ファイルをマージできます。マージとは、複数のセッションから収集したデータを、1 つ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 データを含んでいるファイルを**マージ ファイル**（**.dpmrg**）といいます。DevPartner では、多数のマージ ファイルを 1 つのプロジェクトに関連付けることもできます。マージ 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。保存されたファイルは、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：　相関セッション ファイル、または IE を実行して作成した Web Script セッション ファイルは、マージできません。IIS のサーバー側セッション ファイルをマージすることはできます。

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、新しいマージ ファイルを作成するか、既存のマージ ファイルにデータを追加します。マージ ファイルを自動的に作成することもできます。「マージ設定」（147 ページ）を参照してください。

セッション データをマージすると、DevPartner では以下の処理が行われます。

- セッション ファイルやマージ ファイルでロードされたすべての実行イメージやメソッドの記録を保持します。
- カバーされた値（%）を比較し、データの上位集合を返します。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が、両方のセッションで重複しているからです。
- 最新のイメージを実行したセッション ファイルまたはマージ ファイルのデータを使用して、メソッドとイメージが新しいものか、変更されているか、または削除されているかを判断します。DevPartner では、イメージのタイム スタンプを使用して最新のイメージを判断します。
- 各ソースと実行イメージのメソッド変動率を計算します。メソッド変動率は、セッション間で変更されたコード内のメソッドの割合を表します。これで、コードの安定性を確認できます。
- マージに関係するファイル、マージが行われた時間、マージの作成者名などの情報を管理します。

マージ データの表示

マージ データはマージ データ ウィンドウに表示されます。マージ データ ウィンドウは、フィルタ ペインとマージ データ ペインで構成されています。マージ データ ペインには、**[メソッド リスト]**タブ、**[ソース]**タブ、**[マージ履歴]**タブ、および**[マージ サマリ]**タブがあります。

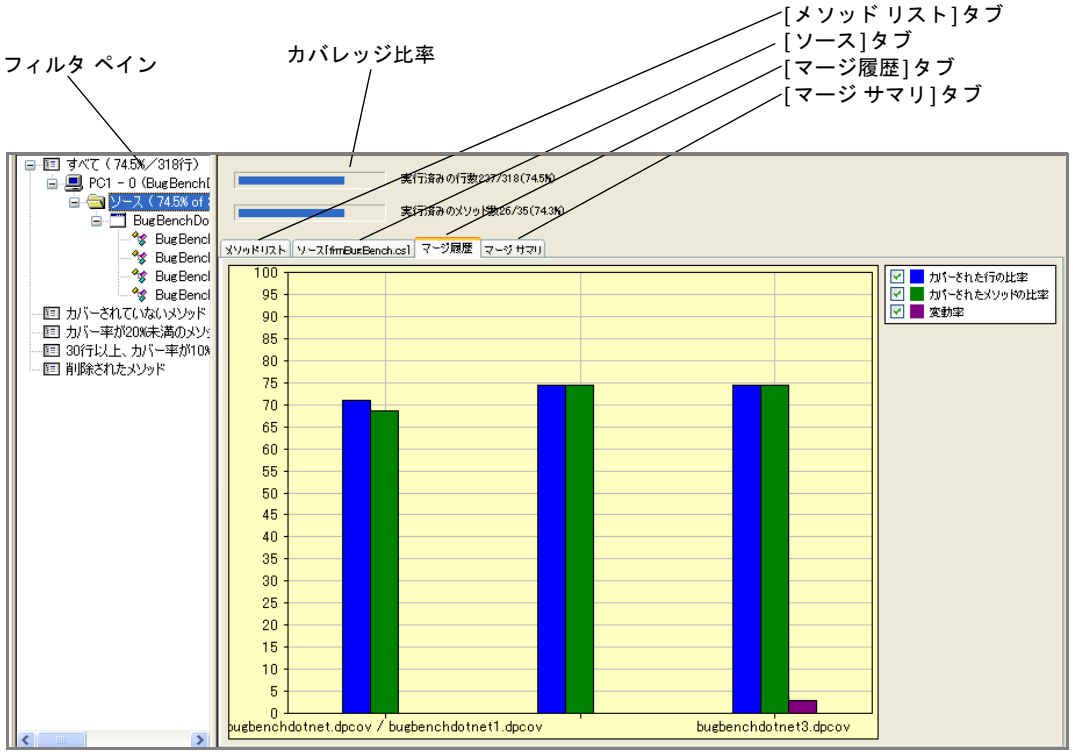


図 4-10.　　マージ データ ウィンドウ

- **[メソッド リスト]**タブでは、マージ ファイルの[状態]カラムが使用されます。[状態]カラムには、セッション間におけるメソッドの状態（[新規]、[変更済み]、[削除済み]）が表示されます。
- **[マージ履歴]**タブには、現在のマージ ファイルについて、**[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[%]]**の推移がグラフに表示されます。
  - **[カバーされた行の比率[%]]**は、ソース コード全体に占める実行されたコード行の割合です。
  - **[カバーされたメソッドの比率[%]]**は、ソース コード全体に占める呼び出されたメソッドの割合です。
  - **[変動率[%]]**は、前回のマージ以降にソース コードが変更されたメソッドの割合です。

1つのマージ ファイルで実行したマージが5つ未満の場合、**[マージ履歴]**タブに棒グラフが表示されます。マージ ファイルで実行したマージが5つ以上の場合、**[マージ履歴]**タブに折れ線グラフが表示されます。

マージの特定のデータを確認するには、グラフのポイントにカーソルを置きます。

棒または折れ線の表示／非表示を切り替えるには、キーのチェック ボックスをオンまたはオフにします。

- **[マージのサマリ]**タブには、ファイルにマージされたセッション ファイルとマージ ファイルについての概要が表示されます。表示される内容には、各イメージに関する**[変動率]**など、セッション中に使用されたインストゥルメント済みの実行イメージについての情報も含まれます。

ソース ファイルに変更があった場合、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あります。

マージの状態

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。マージの状態は、新規、変更済み、削除済みのメソッドおよびイメージとして表されます。これらの状態に関する情報は、**メソッド リスト**の**[状態]**カラムに表示されます。

メソッド

メソッドの状態には、新規、変更済み、削除済み、変更なしがあります。**[状態]**カラムには新規メソッドと変更済みメソッドが表示されます。**[状態]**カラムが空の場合、そのメソッドは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。これらはカバレッジ統計値の計算には使用されません。

DevPartnerは、コード変更の程度を区別しません。たとえば、メソッドの行数が変わるような変更を行うと（コメントの追加や削除など）、このメソッドは**[変更済み]**として表示されます。最適化オプションが異なる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、その違いが変更と解釈され、一部のメソッドが**[変更済み]**として表示される場合があります。

イメージ

あるセッションでロードしたイメージは、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、イメージに含まれるメソッドを判断したり、イメージとそのメソッドを比較して他のセッションからの変更を判断したりすることができません。

イメージの状態には、新規、アクティブ、非アクティブがあります。アクティブなイメージとは、マージ ファイルの他のセッションに含まれ、リロードされたイメージです。非アクティブな状態は、いくつかの条件から発生します。

- イメージが削除されている場合、そのイメージ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 イメージがロードされていない場合、そのイメージは非アクティブとして表示されます。たとえば、アプリケーションでアンマネージ DLL が使用されているのにセッション中にロードしなかった場合、そのセッションと、DLL をロードした前回のセッションとをマージすると、非アクティブとして表示されます。アンマネージ コード プロジェクトとマネージ コード プロジェクトが両方とも含まれているアプリケーションに関して完全なカバレッジを把握するためには、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、必ず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ようにしてください。
- **[除外]** オプション（131 ページを参照）を使用してカバレッジ データの収集からアンマネージ コード イメージを除外した場合、そのイメージは非アクティブとして表示されます。
- マネージ アプリケーションでは、アセンブリ（およびそのすべての参照）がアプリケーションから削除された場合のみ、そのアセンブリが非アクティブとして表示されます。

ヒント： 最後にマージしたセッション ファイルをすばやく確認するには、マージ ファイルの [マージ サマリ] タブの [マージ履歴] を調べてください。

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。これらはカバレッジ統計値の計算には使用されません。**[非アクティブなソース]** フィルタには 0% の値が表示されます。**[非アクティブなソース]** フィルタを展開すると、個々の非アクティブなイメージのカバレッジ値が表示されます。これらの値は、イメージがアクティブだったセッションのカバレッジ データを反映しています。

マージ ファイルの ASP.NET モジュール

カバレッジ セッションを実行すると、繰り返し可能なアルゴリズムによって、アセンブリにコンパイルされる **.aspx** ファイルの名前が生成されます。このアルゴリズムは繰り返し可能なので、アセンブリを登録するたびに同じ名前が割り当てられます。この機能によって、各アセンブリに一貫した名前が付けられるため、アセンブリの変更を正確に追跡できます。

このネーミング操作は、カバレッジ セッションを実行したときにのみ実行されます。**.aspx** ファイルを含むプロジェクトをビルドまたはリビルドする場合、デフォルトの Visual Studio の動作は変わりません。Visual Studio では、ランダムに生成された 8 文字の名前が各 **.aspx** ファイルに割り当てられます。**.aspx** ファイルを編集し、アセンブリをリビルドすると、Visual Studio によって新しいランダムな 8 文字の名前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ときにデフォルトのマージ動作を制御するには、ソリューションのマージ プロパティを設定します。

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、Visual Studio の [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]** プロパティの **[セッション ファイルを自動的にマージ]** の下にあるプロパティを選択します。

- **[マージするかどうかを確認する]**：カバレッジ データを選択して蓄積します。セッションをマージせずに閉じようとすると、マージするかどうかを問い合わせるダイアログ ボックスが表示されます。
- **[確認せずに閉じる]**：カバレッジ データを選択して蓄積します。セッションをマージしないで閉じようとしても、マージを問い合わせるダイアログ ボックスは表示されません。
- **[自動的にマージする]**：すべてのセッションについて、カバレッジ データがマージ ファイルに自動的に蓄積されます。

カバレッジ データのエクスポート

カバレッジ データは XML 形式または CSV 形式で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、自社製またはサードパーティ製のソフトウェアを使用して、データの分析、他のツールで作成したデータとの統合、データウェアハウスへのデータのアーカイブが容易になります。

- DevPartner カバレッジ セッション ファイル（拡張子 **.dpcov**）とマージしたカバレッジ ファイル（拡張子 **.dpmrg**）は XML 形式でエクスポートできます。保存したカバレッジ セッション ファイルを開いているとき、**[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については、「分析データの XML へのエクスポート」（363 ページ）を参照してください。

  また、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。「分析データの XML へのエクスポート」（363 ページ）を参照してください。
- **メソッド リスト**のデータは、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するカラムを表示します。メソッド リストを右クリックし、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。カンマ区切りのテキスト ファイルは Microsoft Excel や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。

データ収集の制御

アプリケーション実行中に収集されるカバレッジ データを制御するには、以下の3つの方法があります。

- プログラムの実行時に、セッション コントロール ツールバーでデータ収集を対話的に制御します。
- 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 モジュールの特定のメソッドに対してセッション コントロールの動作を指定します。
- セッション コントロールAPIを使用して、プログラムのデータ収集を制御します。

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を使用した場合、メソッド内のどのポイントでもデータ収集を制御できます。セッション コントロール ファイルを使用した場合は、メソッドの開始点または終了点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の使用方法については、「分析のセッション コントロール」（353ページ）を参照してください。

コマンド ラインからの分析

データ収集を自動化する場合、またはコマンド ラインから分析セッションを実行する場合、DevPartnerのコマンド ライン実行ファイルである**`DPAnalysis.exe`**を使用します。**`DPAnalysis.exe`**の使用方法については、「コマンド ラインからの分析の開始」（333ページ）を参照してください。

カバレッジ分析ビューアの使用

DevPartner Studioには、Visual Studio 2003やVisual Studio 2005から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]メニューの**[すべてのプログラム]>[Compuware 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。
- コマンド ラインで**`DPAnalysis.exe`**を使用して、カバレッジ分析セッションを実行します。カバレッジ分析ビューアにセッション データが表示されます。
- Visual C++ 6.0とVisual Basic 6.0の場合は、**[DevPartner]>[カバレッジ分析を選択して開始]**コマンドからカバレッジ分析セッションを実行します。カバレッジ分析ビューアにセッション データが表示されます。

カバレッジ分析ビューアの機能

セッション ファイルを開くと、カバレッジ セッション データの表示、ソート、保存、または印刷を行うことができます。また、以下の機能もあります。

- メソッドのソース コードを表示する
- **[メソッド リスト]**タブのデータをソートする
- ファイルの内容をXMLにエクスポートする
- **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実行できない機能

- カバレッジ分析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
- カバレッジ セッションを開始する
- ファイルをVisual Studioソリューションに追加する

メモ： **Visual Studio以外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で開いている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

カバレッジ分析と共にDevPartnerエラー検出を使用すると、デバッガでのマネージ 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、同じセッションでカバレッジ データを収集し、エラーをチェックできます。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分析**を行うアンマネージ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のエラー検出機能の詳細については、「エラー検出」（13ページ）を参照してください。

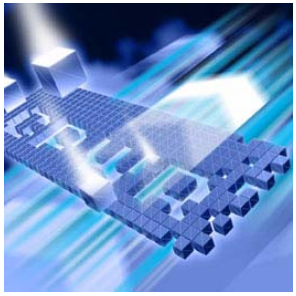
Visual Studio Team Systemへのデータの送信

Microsoft Visual Studio Team Explorerクライアントがインストールされ、Team Foundation Serverの接続を使用可能になっている場合に、DevPartner Studioは Microsoft Visual Studio Team Systemをサポートします。Team Systemのサポートに関する全般的な情報については、「Visual Studio Team Systemのサポート」（8ページ）を参照してください。

DevPartnerカバレッジ分析セッション 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。

バグを送信すると、**[メソッド リスト]**タブの表示カラムのデータが**作業項目**フォームにコピーされます。**作業項目**に送信するメソッド データを変更するには、**メソッド リスト**に表示するカラムを変更してください。

第５章
メモリ問題の検出



- メモリ分析の機能
- すぐにメモリ分析を使用するには
- マネージ Visual Studio アプリケーションにおけるメモリ問題
- プロファイルとオプションの設定
- メモリ分析セッションの開始
- メモリ分析でのセッション コントロール ウィンドウの使用
- メモリ問題の識別
- メモリ分析セッションの実行
- メモリ リークの検出
- 一時オブジェクトを使用したスケーラビリティ問題の解決
- RAM フットプリントを使用したパフォーマンスの改善
- メモリ分析を使用した Web アプリケーションの分析
- 開発サイクルにおけるメモリ分析の使用法
- Visual Studio Team System へのデータの送信

この章には 2 つのセクションがあります。1 つめのセクションには、はじめてのユーザーがメモリ分析機能を利用できるように、簡単な操作手順が記載されています。2 つめのセクションには、DevPartner メモリ分析機能を詳しく理解するための参考情報が記載されています。

メモリ分析に関するその他のタスクに基づく情報については、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 アプリケーションのメモリ割り当て状況を分析できます。

DevPartner では状況に応じたメモリ データを表示できるため、コード内のオブジェクト参照のチェーンやメソッドの呼び出しシーケンスをナビゲートすることが可能になります。これによって、プログラムでメモリがどのように使用されているかを詳しく理解し、メモリの使用を最適化するのに必要な重要情報が得られます。

DevPartner 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、オブジェクトやクラスによって消費されたメモリ量を表示したり、メモリにオブジェクトを保持している参照を追跡したり、メソッド内のソースコードからメモリを割り当てたコード行を特定したりすることができます。

DevPartner メモリ分析には、メモリ リーク、一時オブジェクト、RAM フットプリントという 3 種類の分析機能があります。1 回のメモリ分析セッションで、3 種類のメモリ分析をすべて実行できます。

各分析タイプには、リアルタイムのグラフ、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 コントロールがあります。セッション コントロールを使用すると、データの収集やその他のメモリ関連のイベントを制御できます。たとえば、アクティブなプロセスにガベージ コレクションを強制したり、ヒープの詳細表示を作成することができます。

メモ： **DevPartner メモリ分析機能で分析されるのはマネージ コードのみなので、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、アプリケーションを開発しながら、メモリ分析を使用してアプリケーションのテストを行えます。また、メモリ分析セッションは、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実行することもできます。この場合は、DevPartner のコマンド ライン実行可能ファイル `DPAnalysis.exe` を従来のコマンド ライン スイッチや XML 構成ファイルと一緒に使用します。詳細については、「コマンド ラインからの分析の開始」（333 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には

以下の準備、設定、実行手順では、DevPartner Studio の 3 つのメモリ分析機能について、それぞれの使用方法を紹介します。メモリ リーク分析

機能をすぐに使用したい場合は、色付きの枠内に記載されている手順に従ってください。色付きの枠内に記載されているトピックの詳細な情報については、枠の下に記載されている文章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：分析内容の検討

長時間使用しているときや、特定の操作を実行したりすると、アプリケーションのパフォーマンスが低下しますか。負荷がかかっているときや、他のアプリケーションが実行中のとき、アプリケーションのパフォーマンスが低下しますか。アプリケーションにこのような現象が見られる場合、メモリ関連の問題が発生している可能性があります。

DevPartner メモリ分析は、マネージ アプリケーションのデータのみを収集します。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（C#、Visual Basic、マネージ C++ など）が含まれる必要があります。また、スタートアップ プロジェクトも含まれる必要があります。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、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メッセージが表示されます。

この手順では以下を前提としています。

- Visual Studio 2003 または Visual Studio 2005 で作業しています。
- シングルプロセスのマネージ アプリケーションをテストしています。
- アプリケーションのビルドと実行が可能です。
- ソリューションに少なくとも1つのマネージ コード プロジェクトが含まれます。
- ソリューションにスタートアップ プロジェクトが含まれます。

メモ： DevPartner メモリ分析でサポートされているプロジェクト タイプのリストについては、「DevPartner Studio でサポートされるプロジェクト タイプ」（321 ページ）を参照してください。

アプリケーションによって消費されるメモリの量は、アプリケーションのパフォーマンスに大きく影響します。メモリの割り当て量が多いほど、アプリケーションの実行速度が遅くなり、うまくスケーリングできない可能性が高くなります。

リーク メモリ、つまり返還されないメモリの割り当てがあると、アプリケーションの RAM フットプリントが極端に大きくなることがあります。自動ガベージ コレクション機能があるので、作成したオブジェクトを明示的に解放する必要はありません。そのため、従来の C++ で言うところのメモリ「リーク」は発生しませんが、プログラムが二度と使用しないオブジェクトの参照が残っている可能性はあります。

オブジェクトへの参照が存在するかぎり、その参照先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**とみなされるため、そのオブジェクトが処理されることはありません。これは C++ のリーク メモリと同様に望ましい状態ではありません。このような参照は追跡が難しいため、メモリ分析を使用すると役に立ちます。

この手順ではシングル　プロセス　アプリケーションを前提としていますが、DevPartner Studio を使用して複雑なマルチプロセス アプリケーションを分析することもできます。マルチプロセス アプリケーションのプロファイル方法の詳細については、「複数のプロセスからのデータ収集」（202 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構成設定には、最小限のセットがあります。

> この手順では、デフォルトの DevPartner のプロパティとオプションを使用できます。その他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の速度が極端に遅くなる場合、分析からシステム オブジェクトを除外することでパフォーマンスを改善できます。**[システム オブジェクトの追跡]** 設定やその他のメモリ分析設定を変更する方法の詳細については、「プロファイルとオプションの設定」（165 ページ）を参照してください。

実行：メモリ分析データの収集

メモリ リーク分析を開始する前に、分析のワークフローを理解しておくと役に立ちます。

[開始／停止] をクリックすると、新しいメモリ割り当ての追跡期間の開始／終了がマークされ、アプリケーションによる他のメモリ割り当ては除外されます。

[メモリ リークを表示] をクリックすると、追跡期間中に割り当てられた一部またはすべてのオブジェクトとそのタスクが表示され、ガベージ コレクションを開始できます。

メモリ分析では、追跡期間中に収集されたすべての割り当てが分析され、まだアクティブな参照があり、ガベージ コレクションで処理できないオブジェクトがリークとみなされます。

[メモリ リークを表示] を選択すると、ガベージ コレクションが実行され、いくつかのグラフィックとリスト ビューにリーク オブジェクトを示すセッション ファイルが作成されます。図 5-1「メモリ リーク分析ワークフローのタイムライン」（155 ページ）に示すシナリオでは、メモリ リークのセッション ファイルに、ガベージ コレク

ション後も残ったBとCという2つのリーク オブジェクトが含まれます。このデータから、予期されたリーク オブジェクトと本当のリークとを判断する必要があります。

メモ： **ガベージ コレクションには、[ガベージ コレクション]アイコンまたは[メモリ リークを表示]アイコンを選択して実行されるシステム ガベージ コレクションまたはユーザー開始のガベージ コレクションがあります。**

メモリ割り当て
X－追跡されない／ガベージ コレクションされない（予期されたもの）
Y－追跡されない／ガベージ コレクションされる
A－追跡される／ガベージ コレクションされる
B－追跡される／ガベージ コレクションされない（予期されたもの）
C－追跡される／ガベージ コレクションされない／問題の可能性あり

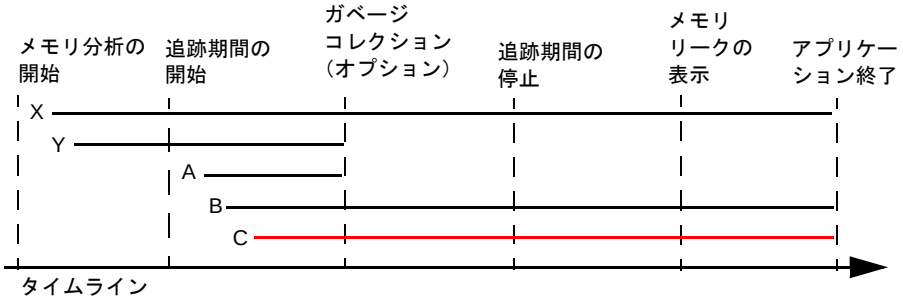


図 5-1. メモリ リーク分析ワークフローのタイムライン

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ている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、**DevPartnerメモリ分析**ウィンドウが開くと、**セッション コントロール ウィンドウ**が表示されます。

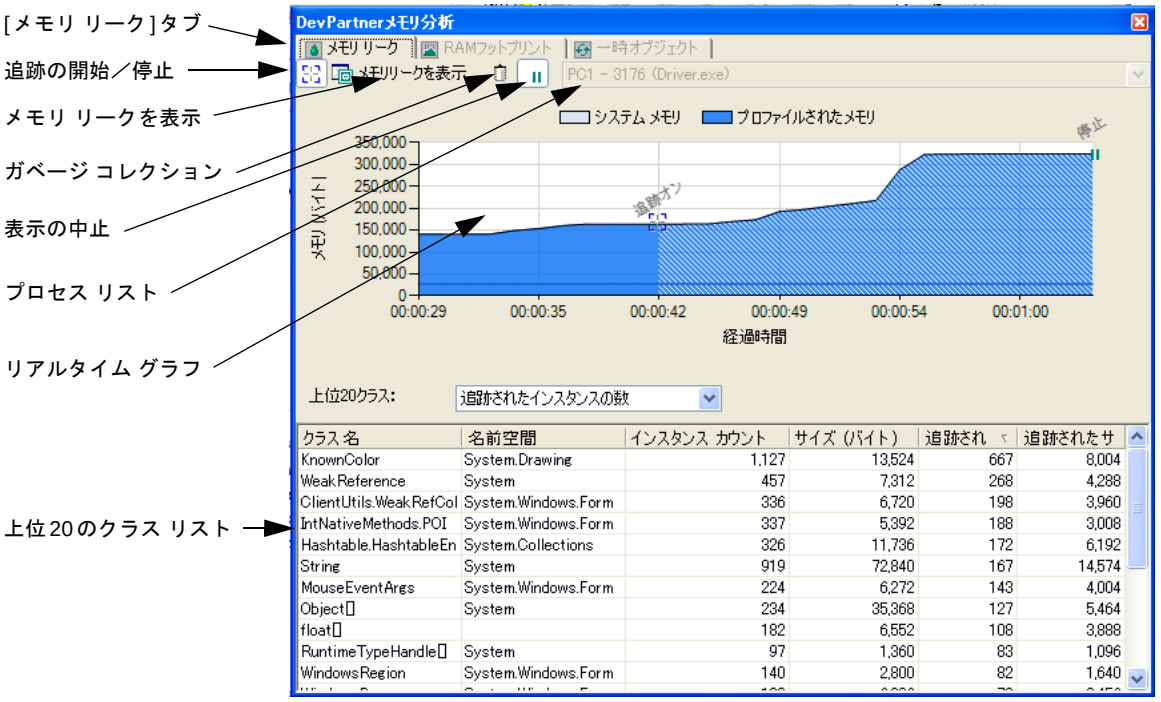


図 5-2. メモリ分析のセッション コントロール ウィンドウ

3 **[メモリ リーク]**タブをクリックします。

4 テスト予定の機能を実行して、アプリケーションをウォーム アップします。アプリケーションのウォーム アップによって、追跡期間から初期化の割り当てが除外されます。

5 **[開始／停止]** をクリックして、新しいメモリ割り当ての追跡を開始し、以前のメモリ割り当てを除外します。

6 データを収集するアプリケーション機能を実行し、サイクル全体を実行しますが、アプリケーションは停止しません。

この手順では、アプリケーション内の単一機能の実行に追跡期間を制限します。これによって、セッション データの複雑さが緩和され、パフォーマンスが上がります。

アプリケーションの実行時にセッション 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、アプリケーションのメモリ使用状況に関する初期診断が可能です。パターンの特徴は、発生しているメモリ問題の種類によって異なります。したがって、リアルタイム グラフを見ることによって、問題の存在と特徴に関する重要なヒントを得ることができます。また、実行すべきメモリ分析の種類を判断するときに役立ちます。

たとえば、徐々に上昇するパターンで、ベースラインに戻らない場合、またはガベージ コレクションに対して予想どおりの反応がない場合、メモリ リークの可能性があります。

リアルタイム グラフの他の特徴的なパターンについては、「メモリ分析でのセッション コントロール ウィンドウの使用」（168ページ）を参照してください。

複雑なアプリケーションの場合、リストに表示されるクラス数が多くなる可能性があります。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]**を選択すると、クラス リストを、アプリケーションのソース コード メソッドに制限することができます。

7 アクティブなプロセスにガベージ コレクションを実行するには、**[ガベージ コレクションを強制します]** をクリックします。

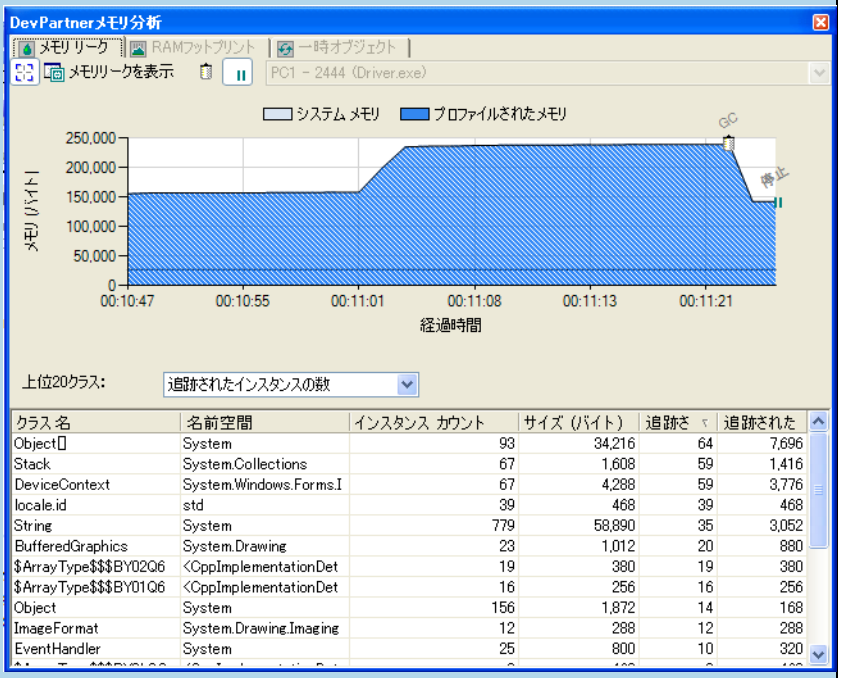


図 5-3. ガベージ コレクション後のセッション コントロール ウィンドウ

8 **セッション コントロール ウィンドウ**の下部にあるリストを確認します。このリストには、ガベージ コレクション後にアクティブな参照があるオブジェクトを含むクラスに関して、情報が表示されます。

9 **[開始／停止]** をクリックして、追跡を停止し、新しい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な場合、リストに含まれる値は変わる可能性がありますが、追跡されるインスタンスは増えません。

10 **[メモリ リークを表示]**をクリックして、もう一度ガベージ 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閉じます。

2つめのファイルである**一時オブジェクト**分析セッション ファイルが自動的に作成され、Visual Studio でそのファイルにフォーカスが移動します。

12 **[Leak...Analysis Snap.dpmem]**タブをクリックして、メモリ リークのスナップショット セッション ファイルにフォーカスを移動します。

メモリ分析データの分析

メモリ リーク分析セッション ファイルには、追跡期間中に割り当てられたオブジェクトのうち、**[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があっ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、クリティカル実行パスを確認すると、オブジェクトがメモリに残っている理由がわかります。

メモリ リーク分析を使用すると、アプリケーションのコンテキストで不要なオブジェクトを特定したり、参照チェーンの中で、オブジェクトがメモリに残る原因となっている参照を削除する最適なポイントを見つけることができます。

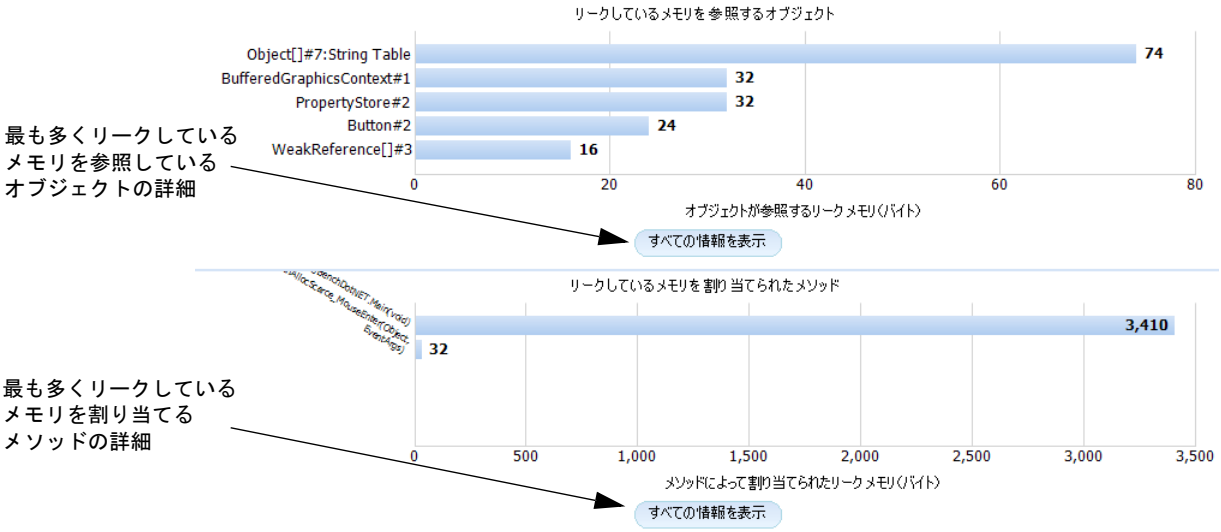


図 5-4. DevPartner メモリ分析－メモリ リーク分析のサマリ

[DevPartner メモリ分析－メモリ リーク分析] のサマリには、**[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ます。

以降の手順では、両方のサマリで詳細を調べ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： メモリ分析セッション ファイルの操作方法の詳細については、「[オブジェクト参照グラフ] の使用」（172 ページ）を参照してください。

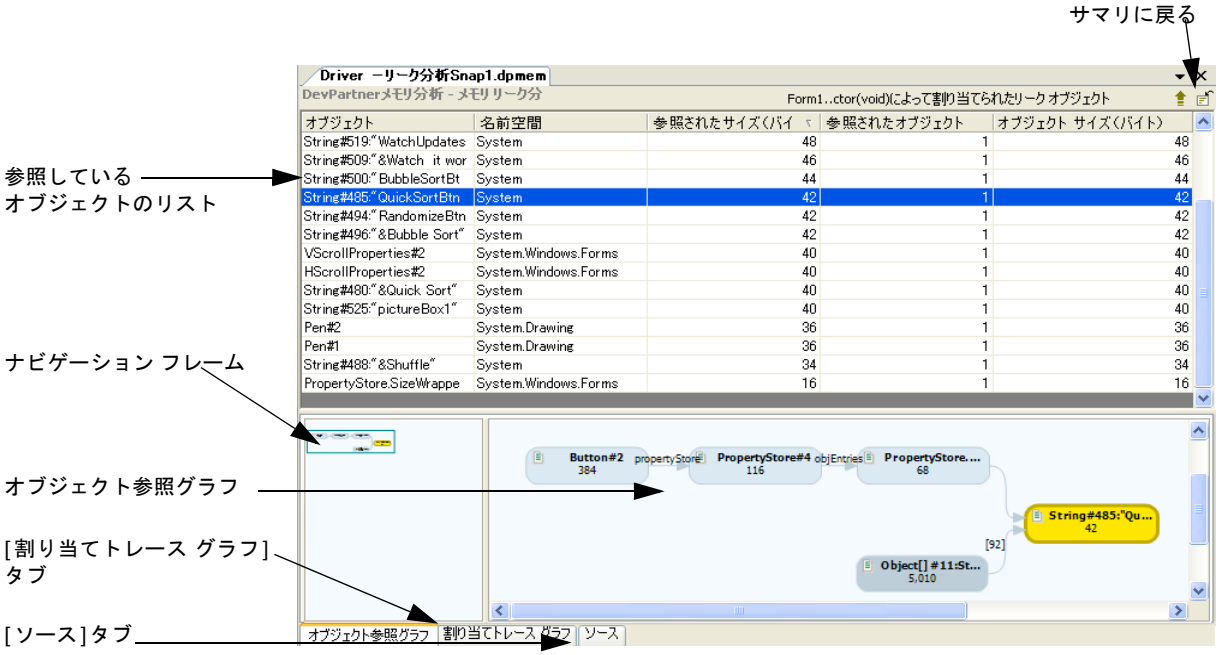


図 5-5. DevPartner メモリ分析－メモリ リーク オブジェクト参照の詳細

2 **[DevPartner メモリ分析－メモリ リーク]**を確認します。この画面には**[リーク サイズ（バイト）]**でソートされた参照元オブジェクトのリストが表示されます。最も多くリークされているメモリの原因となっている**[参照しているオブジェクト]**は、リストの一番上に表示されます。

ウィンドウの下部には、**[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]**の各タブが表示されます。

3 **[オブジェクト参照グラフ]**タブを選択し、上部のリストにある参照元オブジェクトを選択します。

リストから参照元オブジェクトを選択すると、選択したオブジェクトが**[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]**のオブジェクト ノードにマウスを移動して、そのオブジェクトに関連するリーク メモリの情報を参照します。

5 **概要ペイン**の**ナビゲーション フレーム**をドラッグして、**[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します。

6 **リストの[参照しているオブジェクト]**を右クリックし、**[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は、**[参照サイズ（バイト）]**順で、オブジェクトをガベージ コレクションした場合に解放されるはずのメモリ量が強調表示されます。

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。また、この段階で、コード内のどこで参照を絶つかを決めることができます（必要な場合）。

より詳細な情報やプログラムの理解が必要な場合、各タブ表示をクリック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、ソース コードで該当する行の場所を特定します。

- **[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図示されます。このグラフには、オブジェクトと下位オブジェクトが使用するメモリ、オブジェクトが使用するメモリのパーセンテージなど、各オブジェクトと関連情報が表示されます。
- **割り当てトレース グラフ**には、コードの実行パスが図示されます。このグラフでオブジェクトが割り当てられたコンテキストがわかります。
- **[ソース]**タブには、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。

高いコストの原因となるオブジェクト参照について検討し、オブジェクト参照の管理方法を変えることでアプリケーションを最適化できるかどうかを判断します。

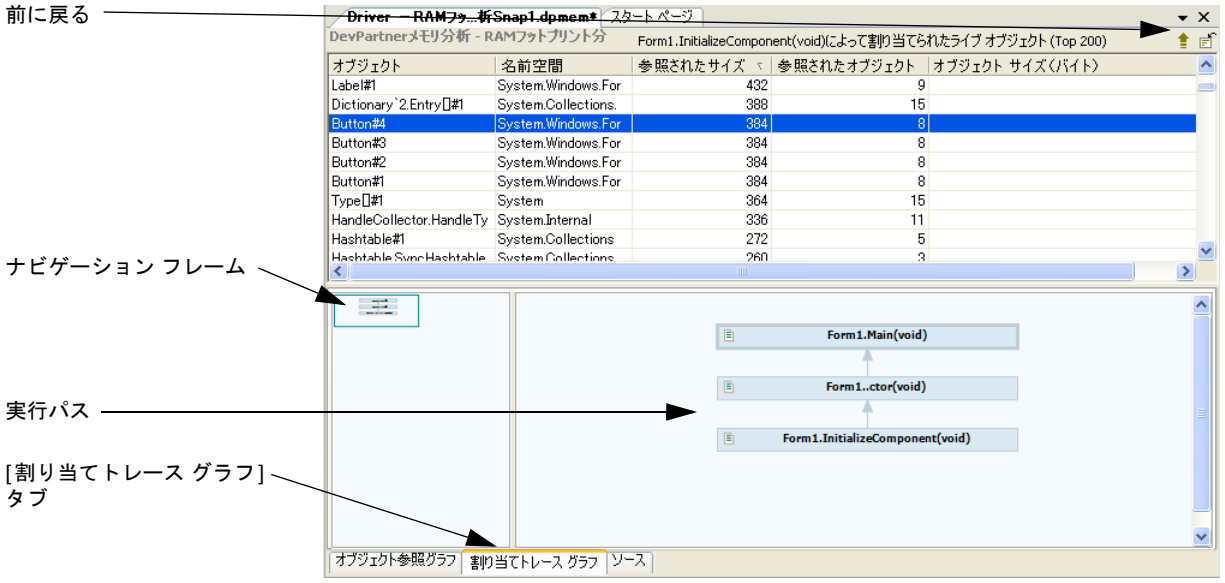


図 5-6. DevPartner メモリ分析ー[割り当てトレース グラフ]

7 変更するには、**[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、メモリを割り当てた実行パスを確認します。

8 **[ソース]** タブを選択し、**オブジェクト リスト**のオブジェクトを選択します。選択した各オブジェクトについて、ソース コードの参照が変わります。

9 リスト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 エディタに関連するソース コード行を表示します。

メモ： システム オブジェクトの場合、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しているオブジェクト」（184 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さまざまな方法については、「[ソース] タブのナビゲート」（177 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 コードを特定したら、オブジェクト レベルを越えて、オブジェクトとオブジェクト参照間の関係までメモリ管理を調べます。ここから、オブジェクト参照の管理方法を変更することでアプリケーションのパフォーマンスを改善できるかどうかを判断していきます。

オブジェクト参照の管理の変更には、より小さなオブジェクトの使用、弱い参照の使用、オブジェクト参照のシーケンスの変更、抽象化の層数の制限などが含まれます。

Web アプリケーションの場合、スケーラビリティが問題のときは、クライアントとサーバー間の関係を知ることにより、サーバー上でガベージ コレクションを利用できるかもしれません。

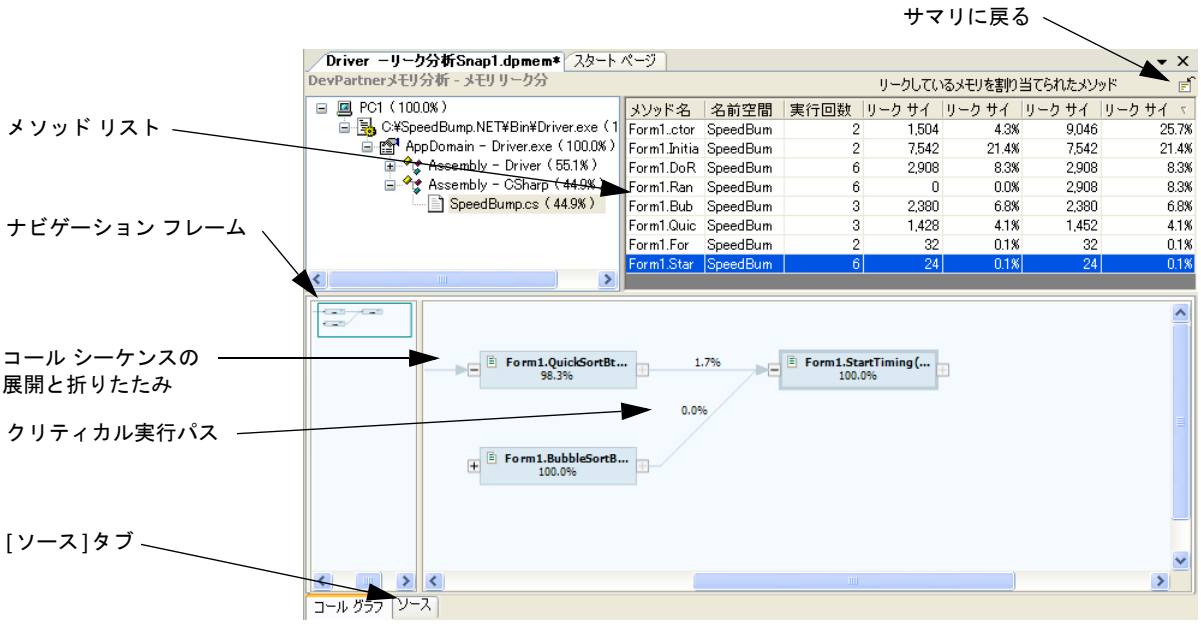


図 5-7. DevPartner メモリ分析ーメソッド リストのコール グラフ

11 **[DevPartner メモリ分析ーメモリ 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]の他に、**[最も多くリークしているメモリを割り当てるメソッド]**も分析できます。

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 リストには、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、**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 ノード間の線にマウスを移動します。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太い金色の線で強調表示されます。

15 コール シーケンスをメソッド ノードのさまざまなレベルに展開したり、折りたたんだりするには、[+]と[-]のコントロールをクリックします。

16 前述のリストでメソッド名を右クリックし、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 リストのメソッド名を右クリックし、**[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」（186 ページ）を参照してください。

セッション ファイルの保存

メモリ分析データの確認が終わったら、セッション ファイルを保存できます。

1. Visual Studio で両方のセッション ファイル ウィンドウを閉じます。
   セッション ファイルの保存を確認するメッセージが表示されます。
2. [OK] をクリックし、デフォルトのファイル名と場所でファイルを保存します。

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。保存されたファイルは、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 ファイルの拡張子は **.dpmem** です。

デフォルトで、セッション ファイルはプロジェクトの出力フォルダに物理的に保存されます。また、デフォルト ディレクトリの内容に基づいて、ファイル名の番号が自動的に 1 つ増えます（たとえば、**MyApp-TemporaryObjectSnap1.dpmem**、**MyApp-LeakAnalysisSnap1.dpmem** など）。セッション ファイルをデフォルト ディレクトリとは別の場所に保存する場合は、自分でファイルのネーミングとナンバリングを管理する必要があります。

Visual Studio 2005 の Web サイト プロジェクトなど、出力ディレクトリがないプロジェクトの場合、ファイルはプロジェクト ディレクトリに物理的に保存されます。

コマンド ラインから生成されたセッション ファイルは、自動的にプロジェクトのソリューションへ追加されません。外部で生成されたセッション ファイルは、Visual Studio で開いているソリューションに手動で追加できます。

この章の残りのセクションでは、メモリ リーク、一時オブジェクト、RAM フットプリントという DevPartner の各メモリ分析機能について、参考情報を提供し、詳細を説明します。

この章の準備、設定、実行セクションはこれで終了です。ここまでで、メモリ分析セッションの実行方法について基本的な知識が習得できたはずです。詳細情報については、この章の残りを続けて読んでください。また、タスクに基づく情報については 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問題

マネージVisual Studioアプリケーションは、ガベージ コレクションによってメモリを精巧に管理する環境で実行されます。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中でなくなると（つまり、アプリケーション側から見て「到達可能」でなくなると）、ガベージ コレクタによってそのメモリが解放されます。

マネージ コード プロジェクトにはメモリ管理が組み込まれているため、多くの開発者は、マネージ言語のおかげでメモリ管理に関する従来の問題から解放されたと思い込んでいます。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が、パフォーマンスのボトルネックになったり、リソースの消耗を引き起こしたりする可能性は、依然としてあります。

開発中のアプリケーションに、以下の現象はありませんか。

- 時間の経過と共にパフォーマンスが低下する
- 実行速度が遅い、または特定の操作を実行するときに処理速度が著しく低下する
- 負荷をかけると、パフォーマンスが低下する
- 他のアプリケーションを実行すると、パフォーマンスが低下する

以上の現象は、いずれもアプリケーションにパフォーマンスの問題があることを示しています。問題がメモリの使用に関連するかどうかを理解するために、以下の質問リストを参照してください。

- サーバー アプリケーション クラスは、プログラムが特定の関数を実行する前、および各アプリケーション クラスがメモリを使用する前にロードする必要がありますか。
  - 現在のタスクの実行に関連するクラスのみをロードしていますか。
  - 特定のクラスについて、アプリケーションによって作成されるインスタンスはいくつですか。またすべてのインスタンスが必要ですか。
- オブジェクト割り当てによってもメモリ使用が発生し、これがパフォーマンスの問題につながることもあります。
  - プログラムが割り当てるオブジェクトの数が多すぎないか、またはプログラムがオブジェクトを効率的に割り当てているかどうか、認識していますか。
  - プログラムによって割り当てられたオブジェクトをガベージ コレクタがクリアしていますか。
  - オブジェクトは期待したとおりにガベージ コレクトされていますか。不要になってからも長時間メモリにオブジェクトが残っていませんか。

メモリ分析の利点

DevPartnerのメモリ分析機能には、マネージ アプリケーションによるメモリの使用状況を総合的に示すビューがあります。DevPartnerでは、メモリに関連するさまざまな種類の問題を特定するのに役立つ、3種類のメモリ分析を実行できます。以下の機能は、使用するメモリ分析の種類にかかわらず、すべての種類の分析に含まれています。

- **リアルタイム グラフ**－アプリケーションの実行中にそのメモリ使用状況をライブで表示します。これは**セッション コントロール ウィンドウ**内に表示されます。アプリケーション コード（プロファイル対象コード）、システム、および他のアプリケーション コード（除外コード）によって使用されているメモリの量や、マネージ ヒープに予約されているメモリに対するメモリ消費量を確認できます。
- **クラスの動的リスト**－アプリケーションの実行中、リアルタイムでプロファイルされたクラスのリストが更新されます。リストには、アプリケーションの実行中に割り当てられたオブジェクト数、各クラスに使用されたバイト数が表示されます。
- **ヒープの詳細ビュー**－プログラムを実行しているときであれば、いつでもマネージ ヒープの詳細ビューのスナップショットをキャプチャできます。このデータはセッション ファイルに保存され、あとでメモリ問題を詳しく分析する際に使用できます。DevPartner では、複数の方法でセッション データにドリルダウンできるため、アプリケーションのメモリ使用状況を調べ、メモリを最も多く使用しているメソッドまたはコード行を見つけることができます。

プロファイルとオプションの設定

メモリ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、除外したりするために、データ収集を調整すると便利です。ソリューションのプロパティ、プロジェクトのプロパティ、DevPartner のオプションを使用すると、分析セッションのフォーカスを調整できます。

ソリューションのプロパティ

ソリューション レベルでメモリ分析に影響がある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.　ソリューションのプロパティ

以下のソリューション プロパティはメモリ分析に影響を与えます。

- **.NET から収集**－メモリ分析を使用してマネージ アプリケーションを実行すると、**[False]** に設定されていた場合、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る場合、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** プロパティは、そのセッション内でアクティブなすべてのプロジェクトについてのデータ収集を制御します。

ソリューションにスタートアップ プロジェクトが含まれる必要があります。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、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メッセージが表示されます。

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]**ページに表示される**[アクション]**が**[開始]**に設定されているプロジェクトのみが、プロンプト ダイアログに含まれます。目的のスタートアップ プロジェクトがプロンプトに表示されない場合、ソリューション プロパティ ページを開き、プロジェクトの**[アクション]**を**[開始]**に設定します。以降のセッションで新しいスタートアップ プロジェクトを選択する場合、新しいスタートアップ プロジェクトのプロパティについて、データ収集のオプションが正しいことを確認します。

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確認するには、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、ソリューション内のプロジェクトについて設定できるプロパティを確認します。

ここで行った変更は、カバレッジ分析、メモリ分析、パフォーマンス分析、パフォーマンス エキスパートに影響があります。

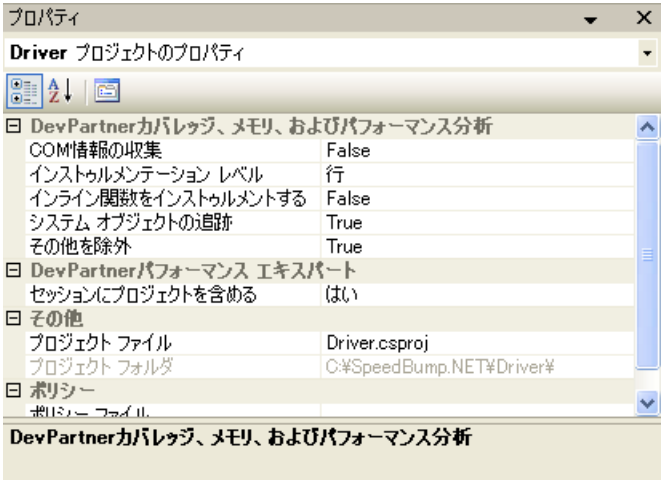


図 5-9.　　プロジェクト プロパティ

以下のプロジェクトレベル プロパティはメモリ分析に影響を与えます。

- **システム オブジェクトの追跡**ーメモリ分析セッションでメモリ内に割り当てられたオブジェクトを追跡するときに、システム オブジェクトまたはサードパーティ オブジェクトの割り当てを無視するには、このプロパティを **[False]** に設定します。
  デフォルトの状態の **[True]** では、システムまたは他のプロファイルされていないリソースによるメモリ割り当ても確認できます。

オプション

メモリ分析セッションの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。

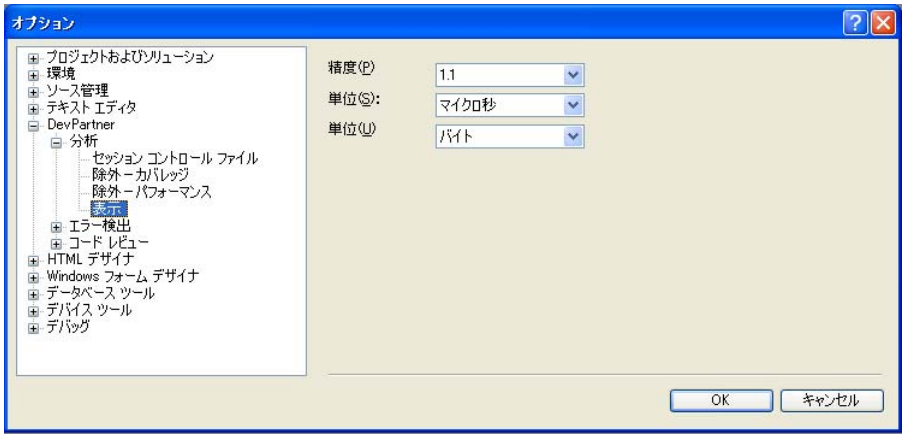


図 5-10.　分析オプション

- **精度**ー選択肢は、1、2、3、または4の小数桁数です
- **単位**ー選択肢は、バイト、キロバイト、メガバイトです。
- **[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、ルールとアクションのセットを作成し、アプリケーションまたはモジュールの実行時にDevPartnerで収集するデータを制御できます。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内でのセッション コントロール ファイルの作成」（354ページ）を参照してください。

[環境]>[フォントと色] オプションなど、他のVisual StudioオプションもDevPartner機能に影響があります。

メモリ分析セッションの開始

メモリ分析セッションを実行するときデバッグを行うか行わないかを選択できます。DevPartnerメニューからは、デバッグなしでメモリ分析セッションを開始するオプションしかありません。プロジェクトまたはソリューションを開いたら、メモリ分析ア

イコンの右のオプションを選択して、デバッグするかしないかを選択してセッションを開始できます。

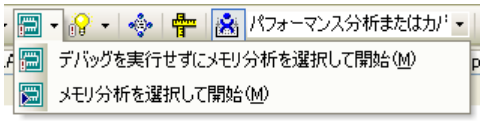


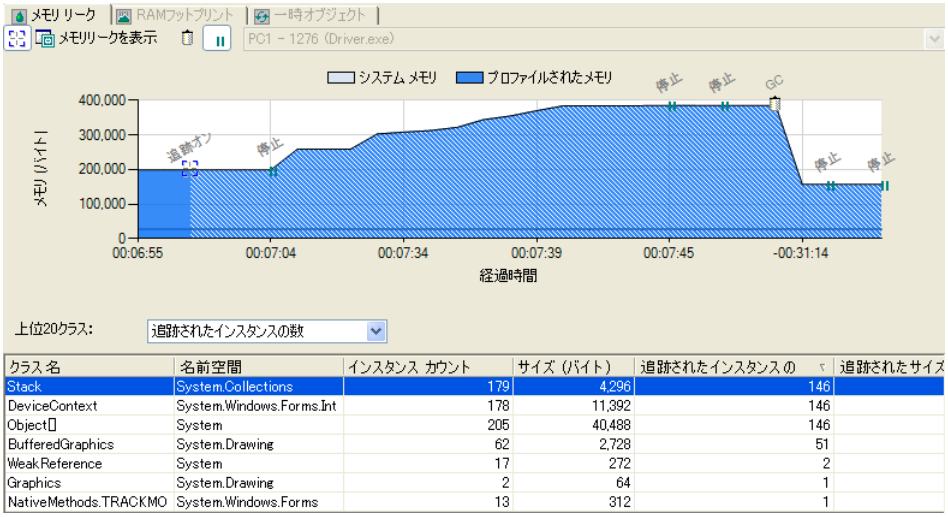
図 5-11. メモリ分析アイコンとデバッグあり／なしの開始オプション

分析結果のわかりやすさとパフォーマンスを考慮して、[デバッグを実行せずにメモリ分析を選択して開始]がメモリ分析のデフォルト設定になっています。コードにブレーク ポイントを設定し、**[メモリ分析を選択して開始]（デバッグを使用）**を選択すると、コード内の特定セクションのパフォーマンスを分離できます。

コードのセクションを分離するためにブレーク 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、プログラムの実行中にメモリ分析機能を実行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 内でのセッション コントロール ファイルの作成」（354 ページ）を参照してください。

メモリ分析でのセッション コントロール ウィンドウの使用

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、メモリ リーク、一時オブジェクト、RAM フットプリントという分析可能なメモリ問題の種類に対応しています。各タブには、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 コントロールが表示されます。セッション コントロールを使用すると、データの収集やその他のメモリ関連のイベント（ガベージ コレクションなど）を制御できます。クラス リストに表示されるデータと使用できるセッション コントロールは、選択したタブによって少し異なります。



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 5-12. 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ら、リアルタイム グラフの変化を観察します。アプリケーションの実行時にグラフに表示されるパターンを見ると、アプリケーションのメモリ使用状況の初期診断が可能です。パターンの特徴は、発生しているメモリ問題の種類によって異なります。したがって、リアルタイム グラフを見ることによって、問題の存在と特徴に関する最初のヒントを得ることができます。また、実行すべきメモリ分析の種類を判断するときに役立ちます。

- 徐々に上昇するパターンで、ベースラインに戻らない場合、またはガベージ コレクションに対して予想どおりの反応がない場合、メモリ リークの可能性があります。**メモリ リーク**分析を実行します。
- ベースラインに戻っても、メモリの使用量が周期的に急増するようなパターンの場合は、アプリケーションの実行に従い多数のオブジェクトが作成されています。**一時オブジェクト**分析を実行します。
- アプリケーションがマネージ ヒープに予約されたシステム メモリのほぼすべてを継続して消費しており、ターゲット ユーザーのシステムにあると考えられるリソースに対し、その消費量が多すぎる場合には、アプリケーションの全体的なメモリ フットプリントを小さくしたいと思うかもしれません。**RAM フットプリント**を実行します。

動的なクラス リスト

クラス リストには、最も多くのメモリを消費している 20 個のプロファイル対象クラスが表示されます。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、このリストは動的に更新されます。クラス リストを使用して、メモリ消費量の増加やオブジェクト作成数の増加にどのクラスが関わっているのかを確認します。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、問題が発生している可能性がある領域を特定できます。

クラス リストには以下のカラムが表示されます。データを示すカラムには、DevPartner 分析の**[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、またはメガバイト単位でデータが表示されます。

- クラス リストのソート順を変更するには、**[上位 20 クラス]**リストでカラム見出しを選択します。
- ソース コードが使用可能なクラスだけをクラス リストに表示するには、リストの任意の場所を右クリックし、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。

メモ： **[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、配列エレメント型がソース コードにある場合、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 リストのカラム見出し

カラム	表示されるデータ
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 カウント	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（単位）	このクラスのインスタンスによって使用されているメモリの量。一時オブジェクト分析と RAM フットプリント分析のデフォルトのソート順。
追跡されたインスタンスの数（メモリ リーク分析の場合のみ表示）	現在メモリに存在するこのクラスの追跡されたオブジェクト数。メモリ リーク分析のデフォルトのソート順。
追跡されたサイズ（単位）（メモリ リーク分析の場合のみ表示）	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。

メモ： **追跡されたオブジェクトとは、ユーザーが[追跡の開始]をクリックしてから、[追跡の停止]に切り替えるまでの間に割り当てられたオブジェクトです。**

DevPartner メモリ分析のセッション 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、さまざまな方法で対話的にデータ収集とデータ表示を制御できます。

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	タブ領域の右上にあるプロセス 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してある) 新しいプロセスの実行を開始すると、そのプロセスがプロセス リストに追加されます。デフォルトの選択は、起動 (スタートアップ) 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ます (切り替え式ボタン)。このボタンをクリックすると、グラフの色が変わり、追跡された部分を示します。
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されたすべてのメモリ データをクリアします。ガベージ コレクションには影響がありません。
ガベージ コレクションを実行 	アクティブなプロセスに対してガベージ コレクションを強制実行します。
表示を中止する 	表示を一時停止しますが、データの収集は停止しません (切り替え式ボタン)。もう一度このボタンをクリックすると、グラフ表示で現在のメモリ使用状況が再描画されます。

セッション結果の表示

アプリケーションの実行中に、対応する **[... を表示]** ボタンをクリックすると、メモリ使用状況のスナップショットをキャプチャできます。この結果、メモリ使用データが含まれるセッション ファイルが作成されます。アプリケーションの実行中に、必要に応じていくつでもセッション 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しません。

表 5-3. メモリ分析のセッション結果の表示に使用できるスナップショット 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に対するガベージ コレクションが強制実行され、セッション ファイルが開いて詳細なメモリ リーク データが表示されます。
一時オブジェクトを表示	セッション ファイルが作成され、詳細な一時オブジェクト データが表示されます。アクティブなプロセスに対するガベージ コレクションは強制実行されません。

表 5-3. メモリ分析のセッション結果の表示に使用できるスナップショット コマンド

スナップショット コマンド	説明
RAM フットプリントを表示	ガベージ コレクションが強制実行され、セッション ファイルが開いて詳細な RAM フットプリント データが表示されます。

作成された保存されていないセッション ファイルは Visual Studio で自動的に開かれます。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になります。保存されたファイルは、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、結果サマリの形式で表示されます。結果サマリを使用して、セッション データにドリルダウンし、ソース コードの問題がある領域を特定します。

セッション ファイルの統合

アプリケーションの実行を停止すると、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。収集データは、メモリ分析セッション ファイルとして保存されます。このファイルには **.dpmem** の拡張子が付いています。

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、アクティブなソリューションのソリューション エクスプローラで参照できます。既存のメモリ分析セッション ファイルをレビューするには、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。

セッション ウィンドウでは、開発環境で結果を分析できます。データにドリルダウンして、オブジェクト参照を調べたり、オブジェクトを割り当てたメソッドの呼び出し関係を調べたり、特定のメソッドのソース コードにジャンプしたりできます。また、任意のメソッドのソース コードを開いて Visual Studio で編集することも可能です。

[オブジェクト参照グラフ] の使用

メモリに残っているオブジェクトを分析するときに、ガベージ コレクタによって処理されなかった理由を理解したいと思うでしょう。**[オブジェクト参照グラフ]** には、選択したオブジェクトと、それをアクティブにしているガベージ コレクション ルートとの間のオブジェクト チェーンが表示されます。

メモ： **[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わけではなく、ガベージ コレクション ルートを指す参照元オブジェクトが表示されます。完全性のために、このグラフには、ガベージ コレクション ルートへの最短パスにない、オブジェクト参照パスのオブジェクトも表示されることがあります。**

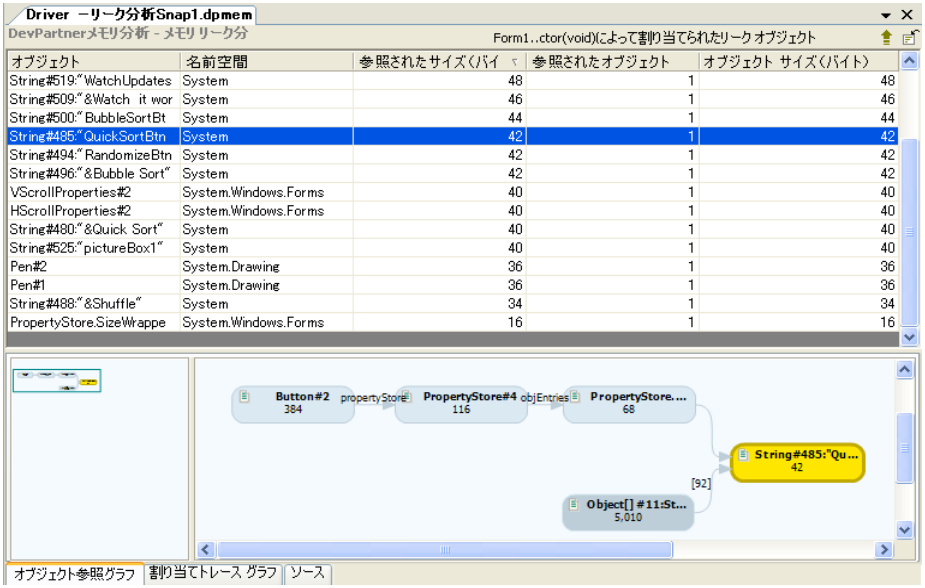


図 5-13. メモリ分析の[オブジェクト参照グラフ]

オブジェクト リストのオブジェクトを選択すると、**[オブジェクト参照グラフ]**が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ます。

- 左側のフレームには**[オブジェクト参照グラフ]**の**概要ペイン**があります。**概要ペイン**には、大きなグラフのさまざまな部分をすばやく特定し、表示できる**ナビゲーション フレーム**が含まれます。
- 右側のフレームには、**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、選択したオブジェクトを示します。ノードの数値データは、コンテキストに応じて、リークしたサイズまたは参照されたサイズを示します。オブジェクト参照パスは、参照順を示す矢印の付いた線で示されます。接続線のラベルは、参照を保持しているメンバー変数を示します。

コール グラフを使用した実行パスの識別

コール グラフは2つのフレームで構成されます。

- 左側のフレームには、大きなグラフのナビゲートに便利な**コール グラフ**の**概要ペイン**が表示されます。概要ペインの**ナビゲーション フレーム**を動かすと、右側のフレームの表示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表示されます。ノード間のリンクは、呼び出しの関係を示します。ノードを展開すると、プログラムの実行順が表示されます。

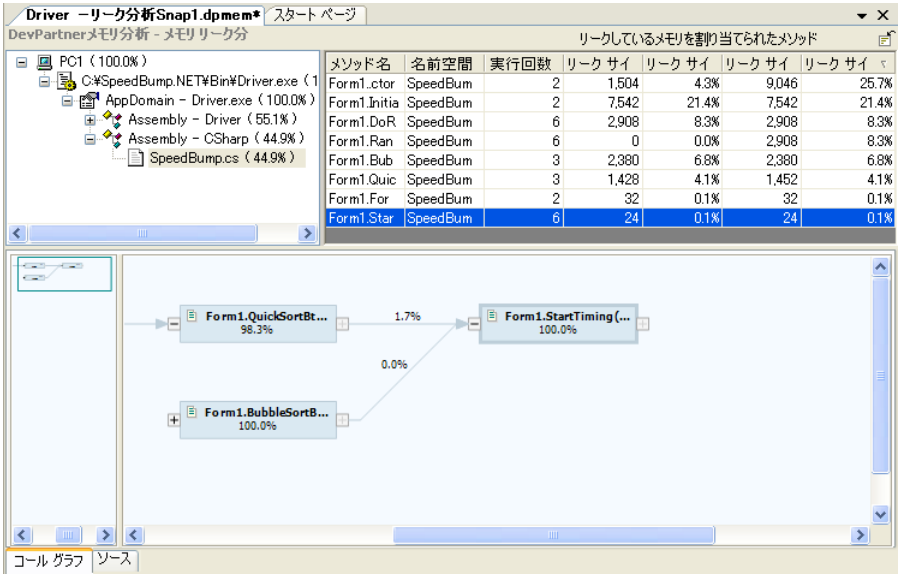


図 5-14.　　メモリ分析のコール グラフ

コール グラフは左から右に読みます。初期状態で**コール グラフ**に表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**で選択したメソッドを示します。選択したノードの左にあるノードは、上位ノードと呼ばれます。ノードの右側にあるノードは下位ノードと呼ばれます。

ノードの上部にはノード名が表示されます。これは、ノードによって表されている関数またはメソッドの名前です。下部にはノードの値が表示されます。これは、ノードに関連付けられたパーセント値です。この値は、ノード（とその下位ノード）が使用し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が占めるパーセント値です。

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。これは、メソッド コールまたはメソッドの起動を示します。ノード間を結ぶ線に表示されるパーセント値はリンク値と呼ばれ、そのリンクに関連付けられたパーセント値を示します。このリンク値は、上位ノードが使用している合計メモリのうち、下位ノード（およびその下位ノード）が使用しているメモリが占めるパーセント値を示します。

関連する上位／下位ノードがないノードは、「デッド エンド ノード」と呼ばれます。このノードは実行パスの端、つまりメソッド コール順の始点または終点を示します。

一時オブジェクトの**メソッド名リスト**と関連する**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]**グラフから**[すべての情報を表示]**をクリックします。

メソッド名リストと関連する**コール グラフ**が表示されると、**メソッド名リスト**のメソッドを選択することで、そのメソッドの**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、メソッドを示すノードを選択し、**[コール グラフ]** ウィンドウの下部にある **[ソース]** タブをクリックします。

クリティカル パス

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項目についてクリティカル パスが計算されます。クリティカル パスは、メモリ割り当て量の合計が最も多くなる下位メソッドのコール シーケンスです。クリティカル パスは太い金色の線で強調表示されます。

[割り当てトレース グラフ]の使用

[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 コールが表示されます。**[割り当てトレース グラフ]** は、RAM フットプリント分析とメモリ リーク分析のセッション ファイルで使用できます。オブジェクト リストを含む任意のビューに表示できます。

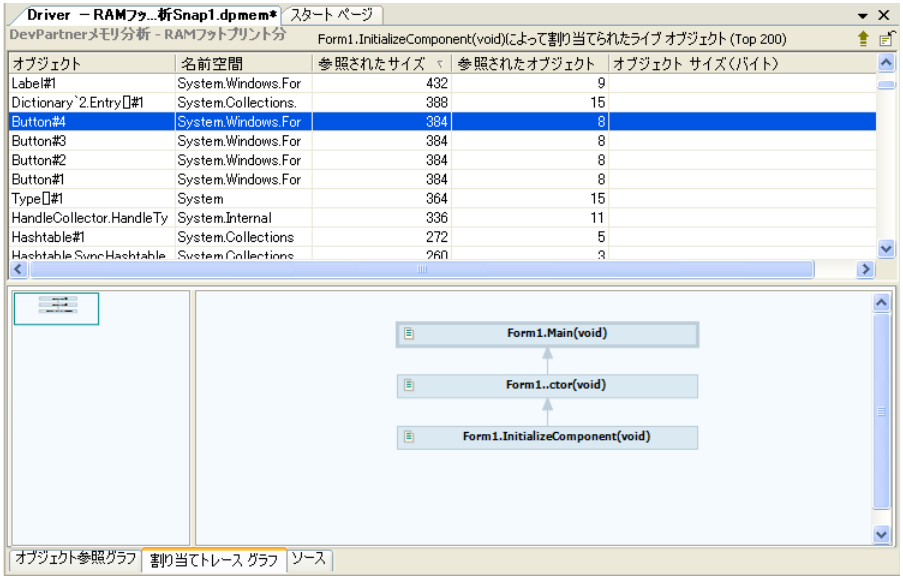


図 5-15. メモリ分析の [割り当てトレース グラフ]

オブジェクト リストと、関連する **[割り当てトレース グラフ]** を表示するには、以下のいずれかを実行します。

- メモリ分析の**結果サマリ**の **[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）グラフの下にある **[すべての情報を表示]** をクリックします。

- **メモリ分析の結果サマリ**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、以下のいずれかを実行します。

- **オブジェクト リスト**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**のオブジェクトを右クリックし、コンテキスト メニューから**[割り当てトレース グラフを表示する]**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]の任意のノードについてソース コードの表示と編集を行うには、ノードを右クリックし、コンテキスト メニューから**[ソースの編集]**を選択します。編集するソース コードが開き、選択したメソッドの部分が表示されます。

ソース コードの表示と編集

[ソース]タブを選択すると、プロファイル対象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]タブ表示には、DevPartner メモリ分析セッション ウィンドウのさまざまな場所からアクセスできます。たとえば、コンテキスト メニューから選択したり、セッション 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]**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表示されます。**[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種類やデータ カラムの選択オプションによって変わります。

ソース コードに関するデータを表示する他に、Visual Studio エディタのソース コードに直接ジャンプすることもできます。これには、DevPartner メモリ分析のコンテキスト メニューから**[ソースの編集]**を選択します。編集するソース ファイルが開き、**[ソースの編集]**コマンドを実行した**[ソース]**タブのオブジェクト ノード、メソッド ノード、またはコード行に対応する行が表示されます。

メモ： **ソース コードが表示されない場合、または不明な文字がソース コードに表示される場合、DevPartner がソース ファイルのエンコードを判断できなかった可能性があります。**
エンコードがわかる場合、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] をクリックするとソース ファイルが表示されます。
このコンテキスト メニューから別のソース ファイルに変更することもできます。

[ソース]タブは、アプリケーション ソース コードの表示で構成され、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を含むデータ カラムが表示されます。表示されるデータ カラムは、**[ソース]**タブを表示した状況によって変わります。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セッションごとに、表示されるデータ カラムは異なります。

[ソース]タブのナビゲート

セッション ウィンドウの（ソース コードがある）オブジェクトまたはメソッドから、**[ソース]**タブに表示されるソース コードの関連する行にジャンプできます。

- メモリ リーク、RAM フットプリント、または一時オブジェクトの結果サマリから、**[すべての情報を表示]**をクリックし、セッション データにドリル ダウンします。
- セッション 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 コードの参照

メモリ分析で関連するソース コードを参照するには、以下の方法を使用します。

表 5-4. ソース コードの参照

表示またはグラフ	ソース コードの参照
オブジェクトの表示	**オブジェクト リスト**のオブジェクトを選択し、**[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。
メソッドの表示	**メソッド リスト**のメソッドを選択し、**[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**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 コードの編集

メモリ分析で関連するソース コードを編集するには、以下の方法を使用します。

表 5-5. ソース コードの編集

グラフまたはリスト	ソース コードの編集
[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 で編集対象のソース ファイルが開きます。
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**のメソッド、または**コール グラフ**のノードを右クリックし、コンテキスト メニューから**[ソースの編集]**を選択します。Visual Studio で編集対象のソース ファイルが開きます。

[ソース]タブのデータ カラムのカスタマイズ

- **[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]**タブのカラムはソートできません。

ソース ファイルの変更

別のソース ファイルを選択するには、**[ソース]**タブ ウィンドウのタイトル バーを右クリックし、**[別のソース ファイルを選択してください]**を選択します。ソース ファイルの新しいマッピングが作成されます。この操作は他のソース パスに影響を与えません。

メモリ問題の識別

以下のシナリオについて考えてみましょう。

QAチームから新しいマネージ アプリケーションの最初のテスト結果が報告されたとき、あなたはその期待通りの結果に満足でした。ところが、その後QAチームがさらに長いサイクルでテストを実施したところ、アプリケーションの実行時間が長くなるとパフォーマンスが低下するという結果が得られました。

これは、期待していた結果ではありません。アプリケーションのどの部分を最初に調べればよいのでしょうか。この問題を修正するにはどうすればよいのでしょうか。

アプリケーション内の問題を見つけるには、DevPartnerメモリ分析を有効にしてそのアプリケーションを実行します。DevPartnerを使用するのに、メモリ問題だと思われる現象の発生を待つ必要はありません。アプリケーションのメモリ使用についてDevPartnerを使用してテストすることを、開発プロセスの一部としてルーチン化します。

DevPartnerは、アプリケーションによるメモリ リソースの使用状況を判断し、現在の問題領域または今後問題が発生する可能性がある領域を見つけるのに役立ちます。

デバッグなしでメモリ分析セッションを開始したあとに、プログラムによるメモリの使用状況を観察するには、**セッション コントロール ウィンドウ**を使用します。

リアルタイム グラフには、メモリの使用状況がグラフ形式で表示されます。クラス リストには、最も多くのメモリを使用しているクラスが表示されます。このリストは、プログラムの実行と共に自動的に更新されます。クラス リストを右クリックすると、**[上位20クラスを表示]**と**[ソースのある上位20クラスを表示]**と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]**ボタンをクリックすると、マネージ ヒープのスナップショットを作成して、詳細な分析を行えます。

メモリ分析セッションを実行する際には、次の3つの重要な問題領域の中から1つを選択できます。

- 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
時間の経過と共にパフォーマンスが低下し、再起動すると回復します。パフォーマンスはアプリケーションの再起動後に回復しますが、再び低下します。	メモリ リーク
スケーラビリティ問題。一時的なパフォーマンスの低下。	一時オブジェクト メモリ リーク
パフォーマンスが低下し、アプリケーションを再起動しても回復しません。	RAMフットプリント 一時オブジェクト

まず、発生している現象について、適切なメモリ分析機能を選択します。最終的にはアプリケーションに対して3種類すべてのメモリ分析を行うことになるかもしれません。そうすれば、問題が見つからなかったとしても、徹底的な分析によってプログラムによるメモリ リソースの使用状況をより理解することができます。

メモリ分析セッションの実行

どのメモリ分析セッションの場合でも、実行したときに最初に目にするのは、**セッション コントロール ウィンドウ**内のリアルタイム グラフです。このリアルタイム グラフは、アプリケーションによるメモリ リソースの使用状況をビジュアルに表現したものです。アプリケーションの実行に従って変化するグラフのパターンを観察します。パターンの特徴は、メモリの使用シナリオによって異なります。そのため、リアルタイム グラフを見ることによって、メモリ問題の存在と特徴に関する最初のヒントを得ることができます。

ヒント: アプリケーションの実行と共に変化する、リアルタイム グラフの形状を注意深く観察してください。グラフのパターンを観察し理解することによってメモリ問題をすばやく診断できることがよくあります。

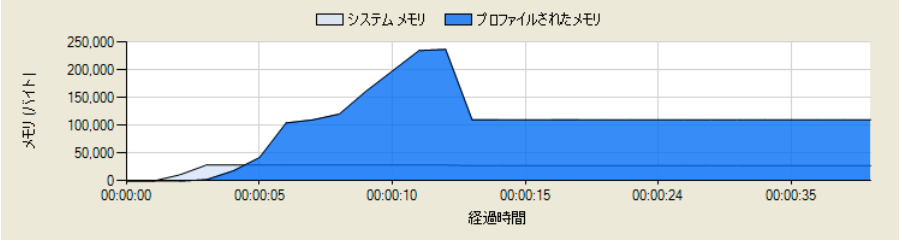


図 5-16. メモリ分析**セッション コントロール ウィンドウ**のリアルタイム グラフ

たとえば、図 5-16 に示すように、グラフが上昇したままベースラインに戻らないパターンを示す場合は、おそらくアプリケーションにメモリ リークが発生しています。QA チームからアプリケーションの速度が次第に低下するという報告を受けたとき、それがメモリ リークを示していると推測することはできますが、リアルタイム グラフを見ることによって、その診断の妥当性を確信できます。

グラフはベースラインに戻っても、メモリの使用量が周期的に急増するようなパターンの場合は、アプリケーションの実行に従い多数のオブジェクトが作成されています。その場合、割り当てられたメモリは解放されますが、そのようなアプリケーションは負荷がかかった状態でうまくスケーリングできない可能性があります。

ユーザー数または入力数の増加に応じてアプリケーションの速度が低下した場合は、スケーラビリティに問題があるかもしれません。このように、リアルタイム グラフには問題の特徴が示されるため、どのような対応が必要であるかをただちに判断できます。

特徴的なパターンが見られない場合でも、リアルタイム グラフから重要な情報を読み取ることができます。たとえば、アプリケーションがマネージ ヒープ用に割り当てられたメモリのほぼすべてを継続して消費しており、ターゲット ユーザーのシステムにあると考えられるリソースに対し、その消費量が多すぎる場合には、アプリケーションの全体的なメモリ フットプリントを小さくしたいと思うかもしれません。この章の以降のセクションでは、そのようなケースに関する詳細な情報を提供し、それがアプリケーション 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 リークの検出

アプリケーションによって消費されるメモリの量は、アプリケーションのパフォーマンスに大きく影響します。メモリの割り当て量が多いほど、アプリケーションの実行速度が遅くなり、うまくスケーリングできない可能性が高くなります。

リーク メモリ、つまり返還されないメモリの割り当てがあると、アプリケーションの RAM フットプリントが極端に大きくなることがあります。自動ガベージ コレクション機能があるので、作成したオブジェクトを明示的に解放する必要はありません。そのため、従来の C++ で言うところのメモリ「リーク」は発生しませんが、プログラムが二度と使用しないオブジェクトの参照が残っている可能性はあります。

オブジェクトへの参照が存在するかぎり、その参照先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**とみなされるため、そのオブジェクトが処理されることはありません。これは C++ のリーク メモリと同様に望ましい状態ではありません。このような参照は追跡が難しいため、メモリ分析を使用すると役に立ちます。

メモリ リーク分析について考えてみましょう。

メモリ リーク分析セッションの実行

準備、設定、実行セクションの「すぐにメモリ分析を使用するには」（153ページ）にも、メモリ リーク機能の使用手順が記載されています。以下にこのプロセスの概要を示します。

メモリ リーク機能によるメモリ リークの特定

1. メモリ分析を有効にして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]**タブを選択します。
2. プログラムの関連する機能を実行して、スタートアップの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により、分析から初期化のメモリ割り当てを除外することができます。
3. **[開始／停止]**をクリックして、新しく割り当てられたオブジェクトだけを追跡開始します。
4. テストするプログラムの機能を実行します。
5. **[ガベージ コレクションを強制します]**をクリックして、アクティブなプロセスについてガベージ コレクションを強制実行します。
6. もう一度**[開始／停止]**をクリックして、追跡期間を終了し、以降のメモリ割り当てを除外します。
7. **クラス リスト**の**[追跡されたインスタンスの数]**と**[追跡されたサイズ]**を確認します。割り当てられたあとガベージ コレクションで処理されていないオブジェクトがある場合、**[メモリ リークを表示]**をクリックして、ガベージ コレクション後も残った追跡対象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： **[メモリ リークを表示]が表示されるのは、追跡の[開始／停止]をはじめてクリックしたあとです。**

マネージ ヒープの状態のスナップショットが表示されます。このデータは、**メモリ リークの結果サマリ**として表示されます。結果サマリ ページから、メモリ使用データにドリル ダウンし、問題を特定し、ソース コードの該当するメソッドを特定できます。

メモ： **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 ルートを正しく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]を選択します。[メモリ分析を選択して開始]（デバッグを使用）を選択して開始したアプリケーションについて、メモリ リークまたは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 ルートがセッション データで「未定義のGCルート」として表示されます。**

メモリ リーク分析の結果について

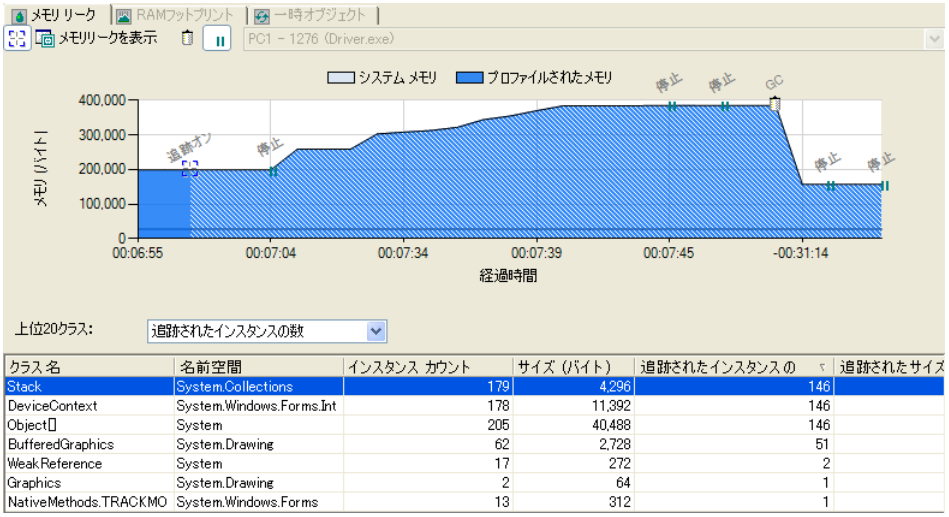
DevPartnerメモリ リーク分析では、特定の期間内にマネージ ヒープ上に割り当てられ、かつメモリ データの収集時に解放されなかったオブジェクトをメモリ リークとして定義しています。メモリ リーク分析は、解放されるべきメモリがアプリケーションの

どこで保持されているかを明らかにするうえで役に立ちます。この情報を使用すれば、コードをどのように変更すればよいかがわかるため、このメモリを解放することができます。

メモリ リークを検出するには、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、前に割り当てられたオブジェクトが解放されるようにします。

メモリの使用量は増加するばかりで、ガベージ コレクションが実行されても減少しない（または期待通りに減少しない）場合は、おそらくアプリケーションでメモリ リークが発生しています。

例については、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、メモリの使用量が時間の経過と共に増加していますが、ガベージ コレクションの実行後もメモリの使用量はベースラインに戻っていません。このアプリケーションに属するクラスの **[追跡されたインスタンスの数]** カラムを見ると、ガベージ コレクションで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]** で、未収集のインスタンス数を確認します。



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**

リークの可能性がある場合は、DevPartner によって作成される**メモリ リークの結果サマリ**（セッション ファイル）を使用してリークの原因を見つけ、その問題を修正します。メモリ リーク分析の結果サマリでは、次の方法で詳細データにドリルダウンできます。

- **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**
- **最もメモリ リークが多いメソッド**

各グラフには、それぞれリーク メモリに関連する上位 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表示するには、そのグラフの **[すべての情報を表示]** をクリックします。

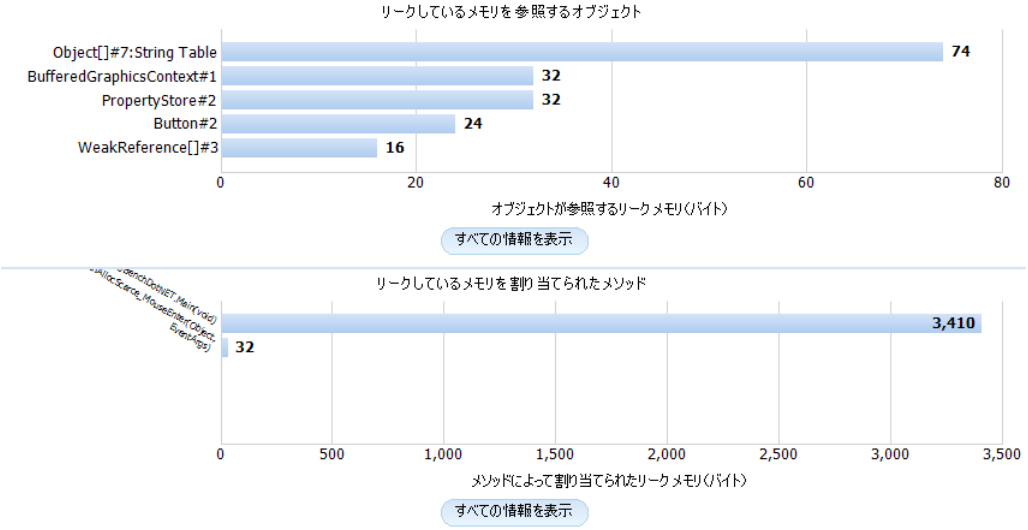


図 5-18. **[メモリ リークを表示]**をクリックしたときに表示される結果サマリ

解決しようとしている問題と、その問題の解決に使用したいアプローチに応じて、どちらの情報をドリルダウンするかを選択します。次に例を示します。

- いくつかの特定のオブジェクトがリークしていると考えられる場合は、**[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]**グラフを使用すると、リーク オブジェクトへの参照を保持しているオブジェクトをすばやく確認できます。
- 割り当てメソッドのソース コードに詳しく、ソース コードを調べることによってリークしているオブジェクトがクリアされるべきであった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]**グラフから始めてもかまいません。

オブジェクトとメソッドのどちらのグラフからでも、すばやくビューを切り替えて、データを別の面から表示することができます。

[最も多くリークしているメモリを参照しているオブジェクト]の完全な詳細を表示した場合は、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]の完全な詳細を表示した場合は、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、**[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]**からドリルダウンを開始しています。

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

以下の例では、いくつかの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ケースを示します。また、問題診断に利用できる他のアプローチも提示しています。

オブジェクトへの参照が 1 つでも存在するかぎり、ガベージガベージ コレクタはそのオブジェクトをクリアできません。アプリケーションを実行すると、そのアプリケーションによってオブジェクトが作成されます。一部のオブジェクトは、プログラムが実行されている間ずっと必要とされます。これらは永久的なオブジェクト（ロング ライブ オブジェクト）です。しかし、ほとんどのオブジェクトは、別のオブジェクトによって参照されなくなると、ガベージ コレクションの対象になる必要があります。

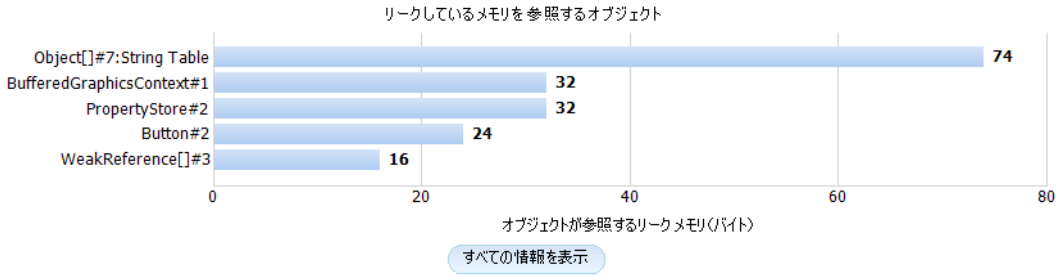


図 5-19.　最も多くリークしているメモリを参照しているオブジェクト

図 5-19 のグラフは、最も多くリークしているメモリを参照している上位 5 つのオブジェクトを示しています。リーク オブジェクトが解放されないのは、これらのオブジェクトがあるためです。このグラフでは、参照元のオブジェクトがメモリ リークの多い順に表示されます。このデータは、いくつかの特定のオブジェクトがメモリ リークの原因になっていることを示しています。このグラフから始めてセッション データにドリルダウンすることにより、メモリ リークの原因を見つけ出します。

グラフの下にある**[すべての情報を表示]**（図 5-20（184 ページ）を参照）をクリックすると、リーク メモリを参照しているオブジェクトの詳細表示が表示されます。

図 5-20 に示すように、この表示の上部にあるパネルにはリーク メモリを参照しているすべてのオブジェクトのリストがあります。このリストには、最初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、リークが少ないリーク メモリを参照しているその他のオブジェクトが含まれています。

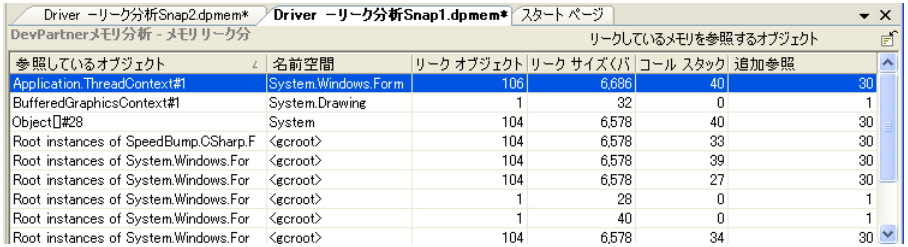


図 5-20.　最も多くリークしているメモリを参照しているオブジェクトのリスト

デフォルト設定では、**[リーク サイズ]**（選択したオブジェクトによって参照されているリーク オブジェクトの合計サイズ）カラムを基準にしてオブジェクトがソートされています。データのパターンを調べるために、他のカラムを基準にしてリストをソートすることも可能です。リスト内の項目を右クリックし、**[オブジェクトによって参照されるリーク オブジェクト]**を選択すると、実際にリークしていたオブジェクトが表示されます。

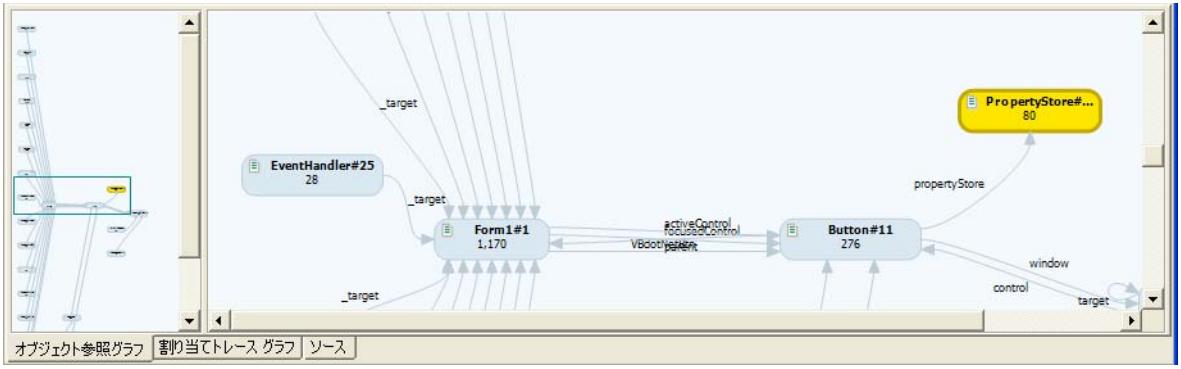


図 5-21.　**オブジェクトがメモリ内に残っている理由を示す[オブジェクト参照グラフ]**

調べたい参照元の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内に保持している参照のシーケンスをすばやく理解できるということは、とても大切なことです。**[オブジェクト参照グラフ]**タブをクリックすると、参照グラフが表示されます。**[オブジェクト参照グラフ]**には、ガベージ コレクタが選択したオブジェクトをクリアしなかった理由が示されます。つまり、選択したオブジェクトとそれをメモリ内に維持しているガベージ コレクション ルートとの間のオブジェクト チェーンが表示されます。

他のオブジェクトを評価するには、オブジェクトのリストを下方向にスクロールします。[オブジェクト参照グラフ]の中には、極めて簡単なものもあれば、非常に複雑なものもあります。また、小さなオブジェクトへの参照が数多く含まれている場合や、大きなオブジェクトへの参照が2～3個しか含まれていない場合など、その状態はさまざまです。このグラフの使用目的は、参照元オブジェクトのチェーンの中でリークを排除する最も効果的な場所を決定することです。

[オブジェクト参照グラフ]に表示される参照元オブジェクトのチェーンの複雑さは、場合によってさまざまです。多くの場合は、複数の参照元があるため、グラフは非常に複雑になります。詳細ペインに表示されているノードを変更するには、**概要ペイン**の**ナビゲーション フレーム**をドラッグするか、ノードをクリックします。複雑なグラフが表示された場合は、ノードを右クリックし、**[すべての参照元を表示しない]**を選択することによって、ビューをシンプルにできます。また、グラフ内でノードをドラッグして表示を見やすくすることもできます。

接続矢印の上に表示されている**エレメント**などのラベルは、グラフ内における次のクラスの参照データ メンバです。角かっこ内の数字は、配列を示しています。コードをよく理解していれば、ラベルを見て、問題が発生している可能性がある領域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内で関連するソース 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できます。グラフ内のオブジェクトを割り当てたメソッドの行が強調表示されます。

プログラムに関する理解を深めるために、グラフ内の各ノードのソースを順番に表示し、リークしたメモリの割り当ての原因となったイベントを確認できます。DevPartnerでは、別の方法でこれらのプログラム イベントを表示することもできます。たとえば、**[割り当てトレースグラフ]**には、選択した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。

リスト内のオブジェクトから直接ソース コードに移動できます。実際に問題を解決する際には、解決しようとしている問題に合った方法、またはコードに見合った方法でドリルダウンする必要があります。

別の問題解決方法

前述の例では、リークの原因を見つけるためにオブジェクト参照パスを使用する方法に焦点をあてました。メモリ リーク ソースの対処方法は他にもあります。次に例を示します。

- **[割り当てトレース グラフ]**を参照し、オブジェクトを割り当てたメソッドの呼び出し元を確認します。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メモリ分析ーメモリ リーク分析のサマリには、さまざまなデータの表示方法があります。最初の例では、**[最もメモリ リークが多いオブジェクト]**を使用しましたが、データの複雑さやユーザーの好みに応じて、**DevPartnerメモリ分析ーメモリ リーク分析**サマリの以下のどのグラフからでも問題を調べることができます。

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メモリ分析ーメモリ リーク分析**サマリの下半分に表示されるもので、リーク オブジェクトを割り当てた上位5つのメソッドが示されます。**[すべての情報を表示]**をクリックすると、リーク オブジェクトを割り当てたすべてのメソッドのリストが表示されます。また、**コール グラフ** ビューや、可能な場合にはメソッドのソース コードにアクセスすることもできます。

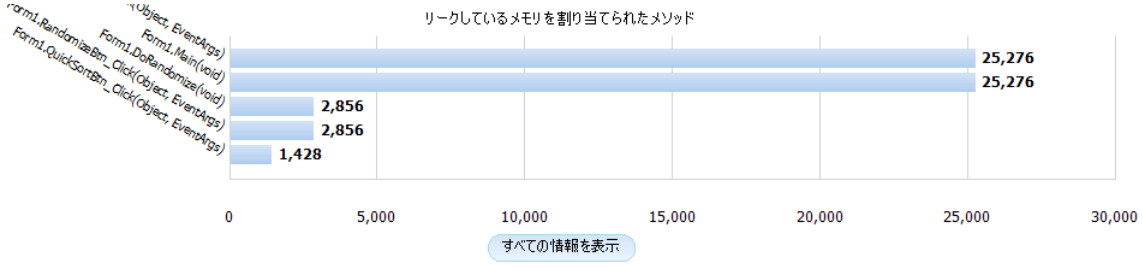


図 5-22.　最も多くリークしているメモリを割り当てるメソッド

メソッド リストのメソッドを選択すると、リークしたメソッドから割り当てたオブジェクトが表示されます。また、メソッドのソース コードを開いて、リークしたオブジェクトを割り当てた行を表示することもできます。その際、その行でリークしたオブジェクトの数とサイズに関する統計情報も表示されます。

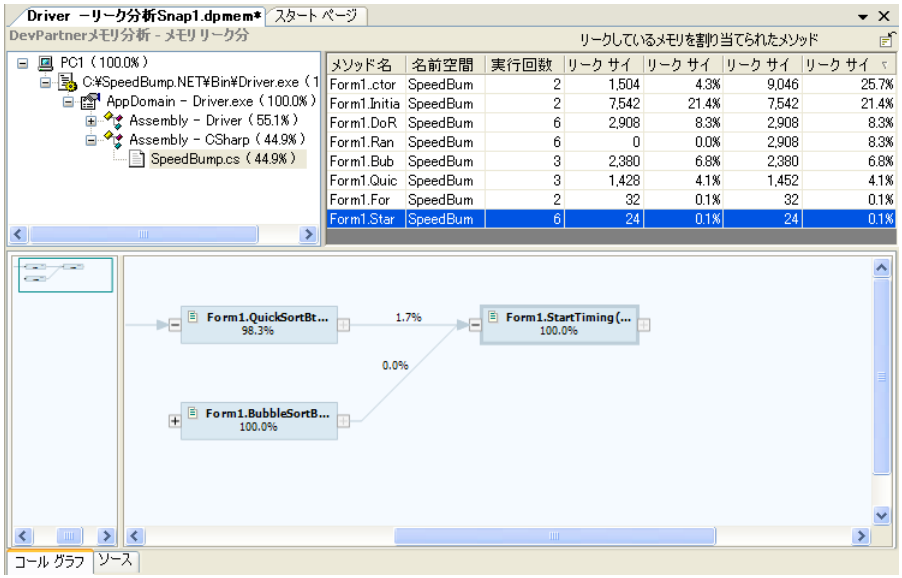


図 5-23. 最も多くリークしているメモリを割り当てるメソッドの詳細

たとえば、データをドリルダウンするには次の操作を行います。

- **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、またはそのメソッドの**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、参照先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、あるいは**コール グラフ**や**[割り当てトレース グラフ]**のノードから、オブジェクト割り当てデータがあるソースコードを表示して、個々の行を参照します。
- リスト内のメソッドまたはオブジェクトから、あるいは**コール グラフ**、**[オブジェクト参照グラフ]**、**[割り当てトレース グラフ]**のノードから、またはソース コードの行から**[ソースの編集]**を選択してソースを開き、適切な行を編集します。

一時オブジェクトを使用したスケーラビリティ問題の解決

DevPartner でメモリ分析を実行する際には、一時オブジェクト分析を使用してスケーラビリティの問題を診断および修正できます。

スケーラビリティ問題の例

普段は正常に動作しているアプリケーションでも、集中して使用すると問題が発生する場合は、そのアプリケーションにスケーラビリティの問題があると言えます。クライアント／サーバー アプリケーションの場合は、ユーザーの数が増加したときにこの問題が発生します。また、スタンドアロンのアプリケーションの場合は、多数のテキスト操作または数値計算のあとでこの問題が発生することがあります。これを「スケーラビリティ」の問題と言います。アプリケーションによって実行される作業の規模が大きくなるにつれ、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

スケーラビリティ問題の原因の 1 つとして、作成される一時オブジェクトの数が多すぎることが考えられます。この場合、オブジェクトの作成はパフォーマンスのボトルネックとなるため、この問題を修正する必要があります。

オブジェクトの作成と使用は、マネージ Visual Studio プログラム内では重要です。しかし、一部のコーディング方法には、多くのオブジェクトが作成されるという副作用を持つものがあります。

問題の一部は、オブジェクトの作成にあります。たとえば、**`String`** クラスで作成されるオブジェクトなどです。これらのオブジェクトは、作成してあとで破棄するために処理サイクルを必要とします。一般に、作成されるオブジェクトの数を減らすことができれば、パフォーマンスの向上を期待できます。

オブジェクトのライフ スパン

DevPartner は、アプリケーション コードによって割り当てられたオブジェクトを追跡し、ガベージ コレクションで収集されるまでの時間の長さに基づいてオブジェクトを分類します。以下の 3 種類のカテゴリがあります。

- ショート ライブー割り当てられたあと、最初のガベージ コレクションで収集されるオブジェクト（世代 0）
- ミディアム ライブー割り当てられたあと、2 回めのガベージ コレクションで収集されるオブジェクト（世代 1）
- ロング ライブープログラムが実行されている間の多く（またはすべて）のガベージ コレクションで収集されないオブジェクト（世代 2）

メモ： **Microsoft .NET Framework のガベージ コレクタは、3 つの世代（0、1、2）をサポートします。前回行われたガベージ 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。割り当てられてから最初のガベージ コレクションを生き残ったオブジェクトは世代 1 となります。さらにその後のガベージ コレクションを生き残った世代 1 のオブジェクトは世代 2 となります。**

DevPartner では、一時オブジェクト カテゴリで、ショート ライブ オブジェクトとミディアム ライブ オブジェクトの組み合わせが行われます。

ミディアム ライブ オブジェクトは、パフォーマンスに対する影響が最も大きく、ガベージ コレクタが必要以上に動作する原因になります。ショート ライブ オブジェクトの場合、個別に見ると、オブジェクトのコンストラクタを呼び出すときにはパフォーマンスの問題が発生するものの、ガベージ コレクションに及ぼす影響はミディアム ライブ オブジェクトの場合よりも小さくなります。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、ボトルネックやメモリ不足が生じることがあります。

コードにスケーラビリティの問題があると考えられる場合は、DevPartner を使用して、実行時にコードが使用するメモリを監視してください。**セッション コントロール ウィンドウ**に表示されるリアルタイム グラフには、上下に変動する波様のパターンが示されます。これはアプリケーションで多数の一時オブジェクトが作成されていることを示すものです。DevPartner を使用して、アプリケーションの一時オブジェクト作成状況を分析できます。

DevPartner は、エントリ ポイント別、メソッド別に、一時オブジェクト分析の結果を分類します。DevPartner では、データのドリルダウンに使用した方法に関係なく、一時オブジェクトが消費したメモリ量を表示し、該当する一時オブジェクトの割り当てを行うコード行を特定できます。

一時オブジェクト分析セッションの実行

以下の手順を使用して、一時オブジェクトの作成によって発生した問題がないか、アプリケーションを分析します。

1. メモリ分析を有効にして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]**タブを選択します。
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、最も多く一時オブジェクトを割り当てたプログラム部分を確認します。
   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（下降パターン）を確認したら、**[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、すぐに**[一時オブジェクトを表示]**をクリックします。
   c. プログラムを終了します。ガベージ コレクションが強制実行され、一時オブジェクトのセッション ファイルが作成されます。

メモ： **デバッガを有効にしてアプリケーションを実行している場合、デバッガを使ってアプリケーションを終了しないでください。この場合、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どおり終了すると、セッション ファイルが生成されます。**

3. アプリケーションの特定部分について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]**をクリックして収集した一時オブジェクトの割り当てデータをクリアします。それから、アプリケーションの関連部分を

実行し、ガベージ コレクションを強制実行し、**[一時オブジェクトを表示]**をクリックします。

メモ： **メモリ分析を有効にして実行したアプリケーションを終了すると、常に一時オブジェクトのセッション ファイルが作成されます。また、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 で実行されたスナップショット動作に対しても、一時オブジェクト セッション ファイルが作成されます。**

マネージ ヒープの状態のスナップショットが表示されます。このデータは、**一時オブジェクトの結果サマリ**として表示されます。結果サマリ ページから、オブジェクト作成データにドリル ダウンし、問題を特定し、ソース コードの該当するメソッドを特定できます。

スケーラビリティ問題の識別

DevPartner を使用すると、潜在的な問題箇所を見つけ出し、アプリケーションにドリルダウンして一時オブジェクトが使用されている部分を確認できます。これにより、問題を見極め、コードの全体的な品質を上げることができます。

リアルタイム グラフ

リアルタイム グラフはハイレベルなビューを提供するもので、問題がある領域を識別できます。

アプリケーションによって多数のショート ライブおよびミディアム ライブのオブジェクトが作成されている場合は、リアルタイム グラフのプロファイル対象メモリにピークが見られます。ガベージ コレクタが機能すると、このピークは低くなります。ガベージ コレクションの実行後にもう一度機能を実行すると、別のピークが見られます。これは、一時オブジェクトの新しいグループが作成されることによるものです。

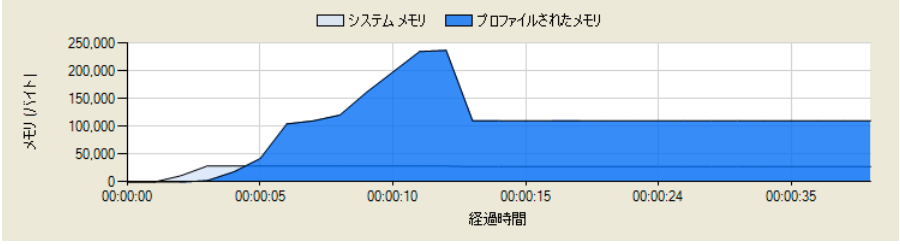


図 5-24. **一時オブジェクトが過剰に作成されていることを示すリアルタイム グラフ**

最も多くの一時オブジェクトの作成に関与したクラスが、**[サイズ]**カラムでソートされて、プロファイル対象クラスのリストに表示されます。このリストを見ると、一時メモリを最も多く消費しているオブジェクトのクラスがわかります。また、**[インスタンス カウント]**カラムには、各クラスに対して作成されたオブジェクト インスタンスの数が表示されます。

図 5-24 は、一時オブジェクトが過剰に作成されていることを示すリアルタイム グラフです。グラフのスパイクは、アプリケーションで多数のオブジェクトが作成されたところを示します。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）では、オブジェクトが過剰に作成されると、パフォーマンスまたはスケーラビリティに関する重大な問題が発生する可能性があります。たとえスケーラビリティが問題にならない場合でも、ショート ライブ オブジェクトを多数割り当てるメソッドはパフォーマンス問題の原因となることがよくあります。

一時オブジェクトを表示

アプリケーションである時点のデータを収集するには、**[一時オブジェクトを表示]** をクリックします。すると **一時オブジェクト分析** ページが表示されます。ここでは、最も多くの一時オブジェクトを作成したものがエントリ ポイント別、メソッド別に分類して表示されます。

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）コードによって呼び出される、プロファイル対象メソッドです。アプリケーションを実行すると、プロファイル対象メソッドまたはユーザーコード メソッドが最初に呼び出された時点から監視が始まります（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 コード内のメソッドのことです）。これは、「エントリ ポイント」と呼ばれます。また、そのポイントから他のユーザーコード メソッドに対して行われる呼び出しもすべて、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 メソッドによって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。しかし、これらのメソッドが原因となって、一時メモリが大量に使用されることもあります。**結果サマリ** の 2 つめのグラフには、多くの一時メモリを割り当てるメソッドが表示されますが、これらが必ずしもエントリ ポイント メソッドだとは限りません。このように、エントリ ポイントによって呼び出された下位メソッドが、そのアプリケーションにおいて多くのメモリを割り当てているメソッドである場合は、そのメソッドを呼び出したエントリ ポイント メソッドの **コール グラフ** をたどらなくても、**[最も多くのメモリを使用するメソッド]** でそのメソッドを特定することができます。

結果サマリ ページからさらに詳細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のメモリ消費状況を把握することもできます。さらに、ショート ライブ オブジェクトやミディアム ライブ オブジェクトを作成しているコード行も見つけることができます。

一時オブジェクト データの分析

各グラフの下にある **[すべての情報を表示]** をクリックすると、すべてのエントリ ポイントの詳細ビュー、または一時メモリを割り当てたすべてのメソッドの詳細ビューが開きます。このビューには、詳細なメソッド リストの他、**[コール グラフ]** タブと **[ソース]** タブもあります。

メソッド リストのデータ カラムには、**セッション コントロール ウィンドウ**のプロファイル対象クラス リストに比べ、アプリケーションのメソッドに関するより広範囲なデータが表示されます。

コール グラフ

エントリ ポイント リスト内のエントリ ポイント、またはメソッド リスト内のメソッドをクリックすると、そのメソッドの**コール グラフ**が表示されます。**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ると共に、**クリティカル パス**が太い金色の線で強調表示されます。**クリティカル パス**として表示されるのは、選択したメソッドに関して割り当てたメモリの合計が最も多い下位メソッドの呼び出しシーケンスです。

コール グラフでは、メソッドが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関するデータが表示されます。また、ノード間のリンクには、その分岐によって割り当てられたメモリに関するデータが表示されます。これらのデータは、割り当てられたメモリの割合（%）として表されます。

- **ノード** – メソッドによって割り当てられた、メソッドの本体そのものに対するメモリの割合（%）。
- **リンク**–メソッドによって割り当てられた、分岐で実行された下位メソッドに対するメモリの割合（%）。

このようにして、アプリケーションの一時オブジェクト作成の原因となったメソッドだけでなく、その割り当てが発生した実行パス上の正確な位置も表示されます。

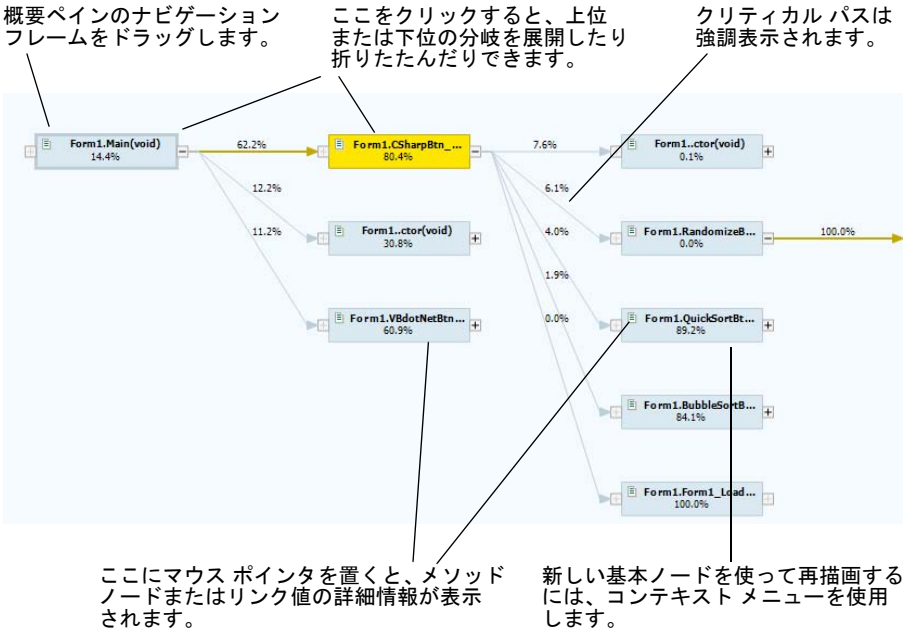


図 5-25. クリティカル パスを示すエントリ ポイント メソッドのコール グラフ

コール グラフ内のノードを右クリックすると、以下の操作を実行できます。

- 選択したノードの**コール グラフ**を再描画する
- 選択したノードのソース コードを表示する
- 選択したノードのソース コードを編集する

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]のエントリ ポイント または**[最も多くのメモリを割り当てるメソッド]**メソッド リストのメソッドのソース コードを確認する際には、選択したメソッドに対する**[ソース]**タブが開きます。ソース コードの他に、アプリケーション コードの各行が割り当てたメモリに関する詳細情報が表示されます。詳細情報には、その行が実行される頻度、その行で割り当てられたショート ライブ／ミディアム ライブ／ロング ライブ オブジェクトの数（下位オブジェクトを含める場合と含めない場合）、これらのオブジェクトの合計サイズ（メモリ負荷）などが含まれます。

スケーラビリティ問題を修正するための結果の解釈

以下に、メモリ関連のスケーラビリティ問題を修正するのに役立つメモリ分析結果の解釈方法をいくつか紹介します。

- **一時オブジェクト分析**ページを参照し、最も多くのメモリを消費しているメソッドがエントリ ポイントであるかどうかを確認します。
- 最も多くの一時オブジェクトを消費しているのがエントリ ポイント メソッドである場合は、**[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、そのエントリ ポイントの実行パス上で最も多くの一時領域を必要としているため、変更を加えたり、呼び出し頻度を少なくする必要があるメソッドを判断します。
- 最も多くのメモリを消費しているのが非エントリ ポイント メソッドである場合は、**[最も多くのメモリを割り当てるメソッド]**ビューを使用してドリルダウンし、変更すべきコード部分を判断します。
- 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が消費する一時領域の量を比較します。この情報を使用して、コード内の変更すべき箇所を判断します。
- ショート ライブ オブジェクトとミディアム ライブ　オブジェクトの両方が同じくらいの量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、コンストラクタが一時オブジェクトを作成するのに費やす時間を調べます。
- 一時メモリを割り当てているメソッド間の関係を理解するには、**コール グラフ**を使用します。消費されるメモリの割合（%）、実際に使用されるバイト数、作成される一時オブジェクトの数など、さまざまなメソッドの特性を調べます。この情報を使用して、変更すべきメソッドを見つけます。
- 一時オブジェクトを割り当てているコード内の特定の行を見つけるには、**[ソース]**タブを使用します。作成されるオブジェクトの種類やサイズ、およびその行の実

行頻度を調べます。この情報を使用して、最も有効なオブジェクトの使用方法を見つけます。

RAM フットプリントを使用したパフォーマンスの改善

マネージ アプリケーションの中には、実行時に数百メガバイトの RAM を消費するものがあります。この章では、特定のメモリ問題について調べています。たとえば、メモリ リーク問題があると、アプリケーションの実行時にヒープを使い切るまでメモリが消費されます。また、一時オブジェクトが過剰に作成されることによってメモリ使用量が周期的に急増していると、スケーラビリティ問題に発展する可能性があります。このような問題は、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、アプリケーションのメモリ フットプリントも増加します。このようなエラーに対してアプリケーションは正常に動作していても、特に多様なエンド ユーザー環境で実行しているときは、パフォーマンスが遅く見えることがあります。

パフォーマンスが悪くなる場合、その原因としては、アプリケーションが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。それでは一体どのくらいの量を過剰と言うのでしょうか。それは、アプリケーションを使用している環境（ハードウェアとソフトウェア）によって異なります。エンド ユーザー環境についてよい考えを持っていても、環境は変化する可能性があります。たとえば、エンド ユーザーは、このアプリケーションと同時に、メモリ リソースが競合する他の複数のアプリケーションを実行することがあります。また、新しいバージョンのアプリケーションをリリースするたびにエンド ユーザーにハードウェアのアップグレードを強要することも不可能です。これらの理由から、アプリケーションのメモリ フットプリントは小さくするのが好ましいとされています。

より具体的に言うと、ここで対処するメモリ使用は、単に全体のメモリ使用ではなく、RAM フットプリントです。アプリケーション パフォーマンス（およびエンド ユーザーがこのアプリケーションについて持つ認識）に対して最も影響が大きくなるのは、オペレーティング システムの仮想メモリ システムに依存するように強制することです。マネージ オブジェクトを仮想メモリにページングすると、アプリケーションのパフォーマンスが大幅に低下します。

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にはどうすればよいのでしょうか。DevPartner は、メモリ分析機能の一部として RAM フットプリント分析を備えています。アプリケーションの開発中に RAM フットプリント分析を定期的に実行します。アプリケーションの設計およびアーキテクチャによって、アプリケーションが RAM リソースをどのように使用するかがほぼ決まります。ベータ リリースの準備ができてから機能を再設計するよりは、開発プロセスの初期段階で再設計する方がはるかに簡単です。

RAM フットプリントの測定

DevPartner を使用すると、RAM の使用に最も大きな影響がある領域にパフォーマンスの調整作業を集中させることができます。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、以下の操作が可能になります。

ヒント : RAM フットプリントの測定に関連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 消費状況のリアルタイム グラフを表示したり、最大メモリ バイトに関連するプロファイル対象クラスのリアルタイム リストを表示します。
- マネージ ヒープのスナップショットを取得します。このスナップショットを使用して、最もメモリを消費しているオブジェクトを調べます。

RAM フットプリントを測定するには、以下の操作を行います。

1. メモリ分析を有効にして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]** タブを選択します。
2. メモリ使用を調べるために、安定した状態になるまでアプリケーションを実行します。
3. **[RAM フットプリントを表示]** をクリックし、その時点のマネージ ヒープの詳細なスナップショットを表示します。
4. ガベージ コレクタが実行されるのは使用できるメモリがすべて消費されたときだけなので、メモリ グラフは、その時点で使用中のメモリ量を正確に表していない可能性があります。プログラムがアイドルの安定した状態にある場合、**[ガベージ コレクションを強制します]** をクリックしてガベージ コレクタを強制実行して、メモリ グラフを更新します。

マネージ ヒープの状態のスナップショットが表示されます。このデータは、**RAM フットプリントの結果サマリ** として表示されます。結果サマリ ページから、セッション データにドリルダウンし、最もメモリを使用しているオブジェクトとメソッドを特定します。

メモ : 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 ルートを正しく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]** を選択します。**[メモリ分析を選択して開始]**（デバッグを使用）を選択して開始したアプリケーションについて、メモリ リークまたは 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 ルートがセッション データで「未定義の GC ルート」として表示されます。

RAM フットプリントの分析ページを使用して、アプリケーションによるメモリの使用状況を詳しく理解することができます。**RAM フットプリントの結果サマリ** から詳細データにドリルダウンして調べるには、以下のような複数の方法があります。

- **オブジェクトの分布**
- **最も多くの割り当てメモリを参照するオブジェクト**
- **最も多くのメモリを割り当てるメソッド**

どれを最初に使用するかは、提示されるデータによって異なります。また、アプリケーションに対する考え方によっても変わります。

オブジェクトの分布

DevPartner では、アプリケーションによって使用されているメモリ（**プロファイル対象オブジェクト**）と、システム コードで使用されるメモリ（**システム オブジェクト**）のそれぞれの割合が一目でわかるように、メモリにおけるオブジェクトの分布状況を円グラフで示します。

オブジェクトの分布グラフは、以下のように解釈します。

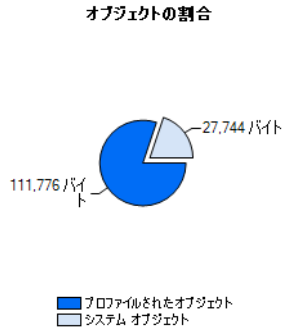


図 5-26. DevPartner メモリ分析のオブジェクトの分布グラフ

- アプリケーション（**プロファイル対象オブジェクト**）が円グラフで最も大きな領域を占めていて、ターゲットとなる運用環境で予測されるリソースに対してメモリの使用が中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メモリが最も割り当てられているかを判断する必要があります。これには、**[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]**または**[最も多くのメモリを割り当てるメソッド]**チャートを使って、データにドリルダウンします。最終的には、アプリケーションの問題の部分のソース コードを特定して、コードを変更したり、より少ないメモリを使用するように作成し直したりする必要があるでしょう。
- 円グラフの「**プロファイル対象オブジェクト**」領域が小さい場合は、アプリケーションはメモリのメイン アロケータではありません。これは、良い結果といえます。ただし、アプリケーションの動作速度が遅いと感じる場合や、全体的なメモリの使用量が高い場合は、アプリケーションでアンマネージ コードやシステム リソースがどのように使用されているかを検証する必要があるかもしれません。アンマネージ コードは、オブジェクトをメモリに固定することがあります。Visual Studio アプリケーションは、多くの場合、.NET Framework で多くの時間を費やします。そのため、Framework メソッドをより効率的に呼び出したり、呼び出す回数を減らすことでこの時間を短縮できる場合があります。

以下の 2 つの分析パス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たメモリを参照するオブジェクト**
- **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]は、セッション ファイルが生成された時点でライブ オブジェクトへの参照を保持していたオブジェクトを示します。表示されるサイズは、このインスタンスから参照される全オブジェクトの合計になります。

- **[すべての情報を表示]**をクリックすると、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]を使うと、メモリの最大使用に関与しているオブジェクトのインスタンスに焦点を当てることができます。割り当てられているメモリへの参照を保持するオブジェクトのインスタンスによってデータを整理すると、「**大きなオブジェクト**」、つまり、オブジェクトがガベージ コレクションによって収集された場合に、最大のメモリ量が回復するであろうオブジェクトが強調表示されます。

個々のオブジェクトは小さい場合でも、参照先のオブジェクトによって消費されるメモリを含めるとはるかに大きくなり、最終的に「大きなオブジェクト」になります。ガベージ コレクタを実行した場合、他のオブジェクトによって参照されるオブジェクトは収集できません。このため、他のオブジェクトを多数参照するオブジェクトは、かなりの量のメモリを占める場合があります。このようなオブジェクトを収集できれば、固有の参照を保持するその他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フットプリントを小さくしようとする場合に、重要なターゲットとなります。

[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、ライブ オブジェクト インスタンスのリストが表示され、セッション ファイルが作成された時点で各オブジェクトがメモリに与えていた影響に関するデータが表示されます。また、**[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]**ビューを表示するタブ付きのウィンドウも表示されます。

このビューにより、メモリ内の最大のオブジェクトを識別できます。**参照サイズ**のデータには、そのオブジェクトが唯一の上位オブジェクトであるすべての下位オブジェクトに起因するメモリが含まれています。個別に考えた場合、オブジェクトは通常小さいものです。ただし、複数の下位オブジェクトを持つオブジェクトは、さらにその下位オブジェクトを持つ場合があります。さらに、上位オブジェクトや下位オブジェクトのオブジェクトごとのオーバーヘッドによって、実質的に大量のメモリを消費していることがあります。

DevPartner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付けられているバイトを集めて、上位オブジェクトに結び付けます。このビューの利点は、割り当て方法を変更した場合に、最も効果のあるオブジェクトに焦点を当てられることです。

メモリを最も消費するオブジェクトを特定したら、メモリ消費を減らすために変更を加えて、それをただちに確認することができます。ただし、さらに調査して、メモリを解放したり、アプリケーションが特定のオブジェクトを使う方法を変更した場合の影響を把握したいと思うかもしれません。

- インスタンスのリストで選択したオブジェクトをダブルクリックするか、コンテキスト メニューを使って、選択したオブジェクトによって参照されるライブ オブジェクトを表示します。

<オブジェクト名>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、選択した上位オブジェクトによって参照され、メモリ内で参照されているライブ オブジェクトがすべて表示されます。つまり、上位オブジェクトを収集できる場合、これらの下位オブジェクトも収集できます。

<オブジェクト名>によって参照されてい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ているオブジェクトによって参照されるオブジェクトのインスタンス リストが表示されます。

その上位ウィンドウと同じく、**[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]**に示されるデータは、割り当てられたメモリへの参照を保持するオブジェクトのインスタンスごとに整理されます。このビューを使うと、メモリにオブジェクトを保持している参照のチェーンをさらに検証することができます。**[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]**にある任意の下位オブジェクトによって参照されるオブジェクトのチェーン全体を検証できます。

任意のオブジェクトからドリルダウンを続け、オブジェクト参照のシーケンス全体を通して、参照を保持す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

前述のオブジェクト ビューには、すべて**[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にあったライブ オブジェクトが表示されます。ライブ オブジェクトとは、それに対してメソッドを呼び出せるオブジェクトです。ガベージ コレクタを実行すると、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 ルートから到達可能であることを意味します。到達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、収集することはできません。**[オブジェクト参照グラフ]**にはこのようなオブジェクト参照が表示されるので、オブジェクトがメモリに残っている理由を判断できます。

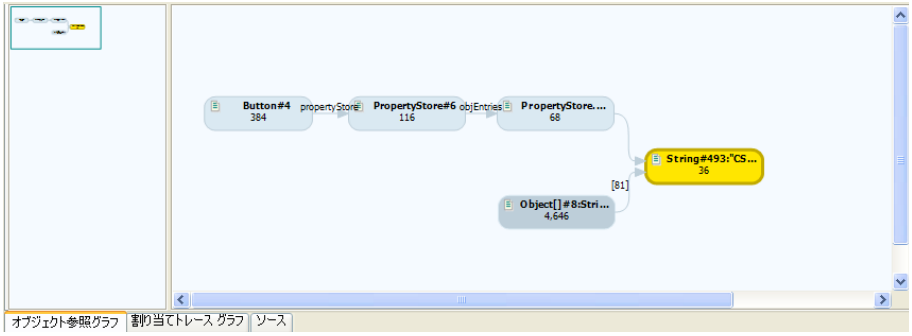


図 5-27.　　オブジェクト参照グラフ

アプリケーションのメソッドによって、オブジェクトと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知ることは重要です。**[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 コールが表示されます。

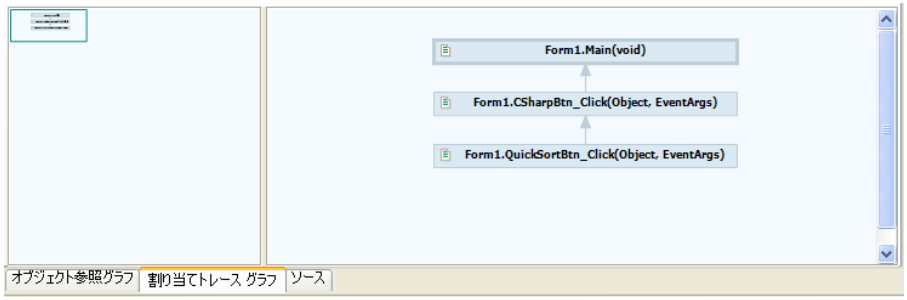


図 5-28.　　割り当てトレース グラフ

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、アプリケーションで最も多いライブ 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、マネージ ヒープで最も多くのメモリを割り当てたメソッドが表示されますが、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ません。

[ライブ サイズ/下位も含む(%)]カラムには、セッション ファイルの作成時にマネージ ヒープに割り当てられていた総メモリ量に対して、そのメソッド（およびその下位メソッド）が使用したメモリの割合が示されます。これにより、メモリ使用量の最も多いメソッドに注意を向けることができます。

ソース コードのビューに加え、このビューには、**コール グラフ**も含まれます。コール グラフには、メモリ割り当てを行った実行パスが示されます。詳細については、「コール グラフ」（192 ページ）を参照してください。

<メソッド名>によって割り当てられているライブ オブジェクト

[<メソッド名>によって割り当てられているライブ 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、このビューには、前のウィンドウで選択したメソッドによって割り当てられたライブ オブジェクトだけが表示されています。これによって、アプリケーション内でライブ メモリを最も多く割り当てたメソッドからドリルダウンできます。ここから、RAM フットプリント スナップショットの作成時にガベージ コレクションで処理できなかったオブジェクトを調べます。

メモ： **割り当てられたオブジェクトのリストには、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は含まれないことに注意してください。「プロファイルされていない（システム）メソッド」とは、[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メソッドを指します。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も、割り当てられたオブジェクトのリストに表示されます。**

ドリルダウンし、メソッドによって割り当てられたライブ オブジェクトが参照するすべてのオブジェクトを調べることで、アプリケーションがオブジェクトの割り当てをどのように行っているかを理解できます。**[このインスタンスによって参照されるすべてのオブジェクト]**ビューは、「<オブジェクト名>によって参照されているすべてのオブジェクト」（198ページ）で説明したビューと同じです。

任意のオブジェクトからドリルダウンを続け、オブジェクト参照のシーケンス全体を通して、参照を保持するオブジェクトをすべて表示できます。

メモリ使用の最適化

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を理解したら、メモリ使用の最適化を開始できます。通常、最も多くメモリを使用しているのはオブジェクトなので、オブジェクトの分析から始めます。

アプリケーションは、おそらく実行時にいくつかのオブジェクトを作成しています。パフォーマンスを最適化するために、単に作成するオブジェクト数を減らしますか。どこに焦点を当ててパフォーマンスの調整を行えばよいでしょうか。

幸いにも、DevPartner はユーザーにとって有益なさまざまな計算を行います。1つ1つのオブジェクトは小さいかもしれませんが、下位オブジェクトと一緒にすると他のオブジェクトよりも大きくなるオブジェクトがあることを念頭に置いてください。DevPartner は、大きなオブジェクトについての考え方に基づき、下位オブジェクトと一緒になるとオブジェクトがメモリを大量に消費することをユーザーに警告します。警告されたオブジェクトの割り当てを中心にパフォーマンス調整を行うと、RAM フットプリントを減らすための最短の近道になります。

ミディアム ライブ オブジェクトに注意してください。ミディアム ライブ オブジェクトは最初のガベージ コレクション後も残り、世代1に移行します。そして、トランザク

ションの終了後に2回めのガベージ コレクション中に収集されます。これは、トランザクションが必要とする新しいメモリ量です。割り当てるオブジェクトの数を減らすことができると、パフォーマンスの向上につながる可能性があります。

アプリケーション実行中のいくつかの時点で、ライブ オブジェクトに注目してください。トランザクションの残りの部分で必要とされないオブジェクトを割り当てていませんか。複数のトランザクションでライブ オブジェクトを共有できますか。あとになるまでアプリケーションで必要とされないオブジェクトを割り当てていませんか。これらの質問に対する答えが「イエス」で、アプリケーションがオブジェクトを割り当てる方法を変更できる場合は、RAM フットプリントを減らし、パフォーマンスを向上させることが可能です。

メモリ分析を使用したWebアプリケーションの分析

DevPartner Studioを使用すると、Visual Studioで開発したマネージWebアプリケーションのメモリ使用を分析できます。たとえば、Web フォーム、XML Web サービス、およびASP.NET を使用するアプリケーションなどです。サーバー側のデータを収集するには、DevPartner Studioをサーバー システム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 マシンで実行されている場合、そのサーバーのデータを収集するには、DevPartner Studio と DevPartner Studio リモート サーバー ライセンスをリモート システムにインストールします。詳細については、『DevPartner Studioインストール ガイド』（**DPS Install.pdf**）と『Distributed Licensing Management ライセンス ガイド』（**compuware_licensing_guide_J.pdf**）を参照してください。サーバーでのデータ収集を設定するには、Visual Studioでメモリ分析の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。IISからWebオブジェクトを直接開くと、Visual Studioでプロジェクトを開くときとは異なり、別のソース ファイルが使用されることがあります。最初のソリューションで作成されたDevPartnerセッション ファイルは、2回め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 データの収集

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に対してメモリ分析を行い、データを収集したいことがあります。DevPartnerでは、クライアント アプリケーションの実行中に、任意のプロセスにおけるマネージ コードのメモリ データを収集できます。

リモートのプロセス データを収集するには、クライアントにDevPartnerをインストールし、リモート マシンに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。この構成を使用して、分散アプリケーションを実際に運用しながらそのデータを収集できます。詳細については、『DevPartnerインストール ガイド』（**DPS Install.pdf**）および『Distributed License Management ライセンス ガイド』（**compuware_licensing_guide_J.pdf**）を参照してください。

複数のプロセスからのデータ収集

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はマネージ アプリケーションについてのみ、メモリ分析データを収集します。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、ブラウザ プロセス（iexplore）のデータは収集されませんが、aspnet_wp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： **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。IISからWebオブジェクトを直接開くと、Visual Studioでプロジェクトを開くときとは異なり、別のソース ファイルが使用されることがあります。最初のソリューションで作成されたDevPartnerセッション ファイルは、2回めのソリューションでは表示されません。**

メモリ分析中にこのようなアプリケーションを実行すると、Visual Studioのメモリ分析セッション コントロール ウィンドウにあるプロセス選択リストにサーバー プロセスとサロゲート プロセスが表示されます。このプロセス リストを使用して、データ収集のフォーカスを設定できます。

DPAnalysis.exeとXML構成ファイルを使用して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については、「コマンド ラインからの分析の開始」（333ページ）を参照してください。

Webアプリケーションを分析するための前提条件

DevPartnerメモリ分析でASP.NETアプリケーションのプロファイルを正常に実行するには、以下の条件を満たす必要があります。

- プロジェクトに**web.config**ファイルが含まれる必要があります。
- プロジェクトの設定でデバッグを有効にする必要があります。このために、**web.config**ファイルには、debug属性をtrueに設定したcompilationエレメントを含める必要があります。次に例を示します。

```
<compilation debug="true" />
```

Webアプリケーションでのメモリ分析セッションの実行

Webアプリケーションでメモリ使用を分析するには、以下の操作を行います。

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. ソリューション内のプロジェクトについて、**DevPartnerのカバレッジ、メモリ、パフォーマンス**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： システムのログオフまたはリブートによって分析オプションが変わるのは、ターミナル サービス クライアント経由でターミナル サーバーに接続している場合、またはリモート デスクトップ経由でシステムに接続している場合のみです。

メモリ分析を有効にして速度が遅いコンピュータをリブートまたは起動すると、SMTP サービス、FTP サービス、および WWWP サービスの起動時にハングしたことがサービス コントロール マネージャから報告されることがあります。このメッセージは無視しても問題ありません。すべてのサービスは正常に起動します。ハングしたと報告されるのは、メモリ分析を有効にすると、これらのサービスが IIS に依存するときに DevPartner が IIS をインストゥルメントするためです。

5 サーバー コンポーネントがローカル マシンで実行されない場合、DevPartner の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やサーバー上の IIS のバージョンによっては、IIS の構成を変更する必要があります。

ヒント： IIS の構成変更については、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]**、または 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。

8 メモリ分析のセッション コントロール ウィンドウを使用して、以下から実行する分析の種類を選択します。

メモリ リーク
一時オブジェクト
RAM フットプリント

実行する分析の種類を選択するための詳細な手順については、「メモリ問題の識別」（178 ページ）を参照してください。

9 **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、**[... を表示]** をクリックして、目的のマネージ ヒープのスナップショットを作成します。必要に応じて、同じセッションで別のサーバー プロセスを選択し、追加のスナップショットを作成することもできます。

10 クライアントの実行中に、ASP.NET または IIS プロセスのメモリ データが収集されます。Internet Explorer（クライアント）プロセスのデータは収集されません。

11 メモリ分析の**セッション コントロール ウィンドウ**にある **[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。または、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、データ収集を自動化することもできます。

予期しない[ファイルの保存]ダイアログとセッション ファイルの保存

場合によっては、ASP.NETアプリケーションの終了後に予期しない**[ファイルの保存]**ダイアログが表示されたり、セッション ファイルを自動的に保存するようにDevPartnerを設定している場合、予期しないセッション ファイルの保存が行われたりすることがあります。

メモリ分析セッションでは、Internet Explorerのデータは収集されません（マネージ コードのメモリ分析データのみが収集されます）。そのため、ASP.NETアプリケーションでメモリ分析を実行すると、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 プロセスになります。一次プロファイル 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、この処理はユーザーの操作に応じて発生します。ただし、プロセスを実行しているシステムで、**`machine.config`** ファイルの**process Model Attributes**セクションを以下のように編集した場合、プロファイル中にASP.NETワーカー プロセスが自動的にシャット ダウンすることもあります。

- `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のシャットダウンが発生するくらい低い値に変更した場合
- `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のシャットダウンが発生するくらい低い値に変更した場合
- `memoryLimit` 属性の値を、セッション中にプロセスのリサイクルが発生するくらい低いパーセント値に変更した場合

プロセスがシャット ダウンすると、最後のスナップショットが作成され、セッション ファイルが生成され、セッションが終了します。IE（ユーザーが起動したクライアント プロセス）がまだアクティブな場合、そのIEプロセスによってASP.NET ワーカー プロセスの新しいインスタンスが生成されることがあります。これらのASP.NET ワーカー プロセスが終了するとき、それぞれセッション ファイルが生成され、その結果、セッション ファイルが保存されたり、[ファイルの保存]ダイアログが表示されたりします。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く、通常は小さな値です。

この状況を修正するには、**`machine.config`** ファイルを編集し、プロセスが早期に終了しないように、制限属性を高い値に設定します。

注意： **`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。

ASP.NET ワーカー プロセスの起動から終了まで、分析データの収集は継続されます。ただし、Visual Studioプロパティ ウィンドウの**DevPartnerのカバレッジ分析、メモリ分析、パフォーマンス分析**で分析を明示的に無効にした場合、収集は終了します。

セキュリティ エラーが表示された場合

マネージ アプリケーションのデータを収集しようとしたときに、セキュリティ ポリシーによってコードの DevPartner インストゥルメントが回避された場合、セキュリティ エラー 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。コードの実行に使用しているポリシーの権限セットからこの権限を削除した場合、またはこの権限を取り消すような厳しいセキュリティ宣言をアセンブリに追加した場合、アセンブリはプロファイルできません。

この状況を修正するには、以下のいずれかの方法で、安全なプロファイルを有効にします。

- 以下のグローバル環境変数を設定し、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
  ```

  この方法で問題を回避できますが、パフォーマンスがやや低下します。

- .NET Framework 構成ツールの MMC スナップインを使用するか、アセンブリに含まれる厳しいセキュリティ宣言を一時的に削除することで、アセンブリのポリシーを変更します。

Visual Studio のセキュリティ ポリシーの詳細については、Visual Studio のオンライン ヘルプに含まれる「.NET Framework 開発者ガイド」を参照してください。

開発サイクルにおけるメモリ分析の使用法

テストを開始して、問題があるのではないかと疑問を抱くまで待つ必要はありません。DevPartner のメモリ分析を早い段階で、高い頻度で実行し、アプリケーションを分析するときに何に注目するのかということを認識すると、早期に問題を修正することができます。この段階では、識別も容易であり、修正を行ううえでのリスクも低くなります。

マネージ アプリケーションにおけるメモリの問題は、多くの場合、単純なコーディング エラーではなく、より広範な設計上またはアーキテクチャ上の決定の結果であることがあります。たとえば、メモリ損失の原因の 1 つが、解放されていない古い参照のためにガベージ コレクションで収集されないオブジェクトにあるとします。これは、コードの別の部分を変更した結果であることがあります。このような問題が識別されるのが開発サイクルのあとの段階になればなるほど、修正が難しくなり、コストも高くなります。

結論としては、開発サイクルを通して、継続的なプログラム テストの一部としてメモリ分析を行うことが重要です。単体テストでメモリ分析を使用し、各モジュールがメモリをどのように処理しているかを理解することも有益です。改善が必要な領域を特定して修正したら、もう一度テストして修正内容を検証します。次にモジュールをアプリケーションに統合したら、再びメモリ テストを繰り返して、新たなメモリ問題が発生していないことを確認します。

Visual Studio Team System へのデータの送信

DevPartner メモリ分析セッション ファイルの任意のメソッド 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 Visual Studio Team System へ送信できます。以下の分析で選択したメソッドが、作業項目として有効です。

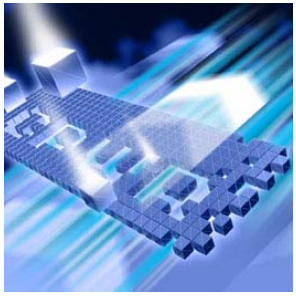
- メモリ リーク ー最も多くリークしているメモリを割り当てるメソッド
- RAM フットプリント ー最も多くのメモリを割り当てるメソッド
- 一時オブジェクト ー最も多くのメモリを使用するメソッドと、最も多くのメモリを割り当てるエントリ ポイント

バグを送信すると、ビューの表示カラムのデータが**作業項目**フォームにコピーされます。**作業項目**に送信するメソッド 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」（8ページ）を参照してください。

第 6 章

パフォーマンスの自動分析



- パフォーマンス分析の機能
- すぐにパフォーマンス分析を使用するには
- プロパティとオプションの設定
- さまざまなアプリケーションからのデータの収集
- コール グラフの分析
- セッションの比較
- パフォーマンス データのエクスポート
- データ収集の制御
- コマンド ラインからの分析
- パフォーマンス分析ビューアの使用
- .NET アプリケーションのパフォーマンス分析のヒント
- Visual Studio Team System へのデータの送信

この章には 2 つのセクションがあります。1 つめのセクションには、はじめてのユーザーがパフォーマンス分析を利用できるように、簡単な操作手順が記載されています。2 つめのセクションには、DevPartner Studio のパフォーマンス分析機能を詳しく理解するための参考情報が記載されています。

パフォーマンス分析に関するその他のタスクに基づく情報については、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の機能

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を低下させる、ユーザー コード、サードパーティのコンポーネント、オペレーティング システム内のボトルネックを見つけることができます。

DevPartner パフォーマンス分析には、以下の機能があります。

- コンポーネントが分散システム上にあっても、そのコンポーネントを実際に使用しているときのパフォーマンスを分析します。
- アプリケーションの特定の面を対象にデータ収集を行えます。これにより、パフォーマンス調整作業の焦点を絞ることができます。
- 対象のアプリケーションのスレッドに使用された時間と、他の実行アプリケーションのスレッドに使用された時間を区別できます。このため、外部の影響と関係なく、正確で再現可能な結果が得られます。

すぐにパフォーマンス分析を使用するには

以下の準備、設定、実行手順では、DevPartner を使用してコードのパフォーマンスを分析する方法を紹介します。

機能をすぐに使用したい場合は、色付きの枠内に記載されている手順に従ってください。色付きの枠内に記載されているトピックの詳細な情報については、枠の下に記載されている文章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：分析内容の検討

パフォーマンス分析を使用する前に、分析内容を検討します。

この手順では以下を前提としています。

- Visual Studio 2003 または 2005 で作業しています。
- シングルプロセスのマネージ アプリケーションをテストしています。
- アプリケーションのビルドと実行が可能です。
- ソリューションにスタートアップ プロジェクトが含まれます。

メモ： DevPartner パフォーマンス分析でサポートされているプロジェクト タイプのリストについては、「DevPartner Studio でサポートされるプロジェクト タイプ」（321 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場合、パフォーマンス セッションを開始する前に、収集対象のデータを決定します。場合によっては、セッションの開始前に必要な手順があります。たとえば、以下の場合、何らかの設定が必要です。

- 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
- 分析対象にするアンマネージ モジュールがある場合
- リモート サーバーで実行されているコードを含める場合

この手順では、アプリケーションに含まれるローカルのマネージ コードがすべて分析されます。

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ら、データ収集を制限するために、いくつかのプロパティとオプションを設定します。

この手順では、デフォルトの DevPartner のプロパティとオプションを使用できます。その他の設定は必要ありません。

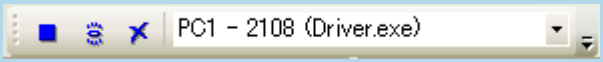
ソリューション プロパティとプロジェクト プロパティを使用して、.NET アセンブリ、アプリケーションの外部で実行される COM、他の実行アプリケーションのスレッドに使用される時間、行レベルの分析、メソッドレベルの分析などに関する情報を分析セッション データに含めるかどうかを選択できます。DevPartner のオプションを使用すると、表示オプションを変更したり、アプリケーションの一部を分析から除外したり、データ収集管理のためにセッション コントロール ファイルを作成したりできます。設定のカスタマイズの詳細については、「プロパティとオプションの設定」（217 ページ）を参照してください。

実行：パフォーマンス データの収集

分析する内容を検討し、適切なプロパティとオプションを設定すると、パフォーマンスジ データを収集する準備が整います。

1. Visual Studio で、アプリケーションに関連付けられているソリューションを開きます。
2. **[DevPartner]>[デバッグを実行せずに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、アプリケーションの任意の部分にフォーカスしてパフォーマンス分析を行えます。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たり、現在収集されているデータのスナップショットを作成してから記録を続行したり、収集しただけでスナップショットに保存していないデータをクリアしたりできます。
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  (必要に応じて2回クリックしてセッション ウィンドウにフォーカスを移動します)。スナップショットを作成すると、収集データを含むファイル(セッション ファイルと呼ばれます)が作成され、セッション ファイル データが表示されます。
5. アプリケーションに戻り、テストの実行を継続します。
6. テストの実行が完了したら、アプリケーションを終了します。最後のセッションファイルが Visual Studio に表示されます。

メモ： **マネージ アプリケーションのデータを収集するときにセキュリティ エラー メッセージが表示される場合、セキュリティ ポリシーの変更方法について** 222 **ページを参照してください。**

パフォーマンスを分析するときデバッグを行うか行わないかを選択できます。一般的には、デバッグを実行しないセッションの結果の方がわかりやすいため、デバッグを実行せずにパフォーマンス分析を実行します。アプリケーションをデバッガで実行すると、セッション中にブレークポイントに達した場合など、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。

スナップショットを作成するか、またはアプリケーションを終了すると、図6-1のようにVisual Studioにセッション ファイルが表示されます。セッション ウィンドウは以下の要素で構成されます。

- 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が表示されます。フィルタ ペインには、各ファイルに使用された時間が、セッションに使用された時間に占めるパーセント値で表示されます。また、フィルタ ペインには、最も重要なデータにフォーカスするために使用できるフィルタのセットも用意されています。
- 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 ペインで選択したファイルまたはフィルタのデータが表示されます。

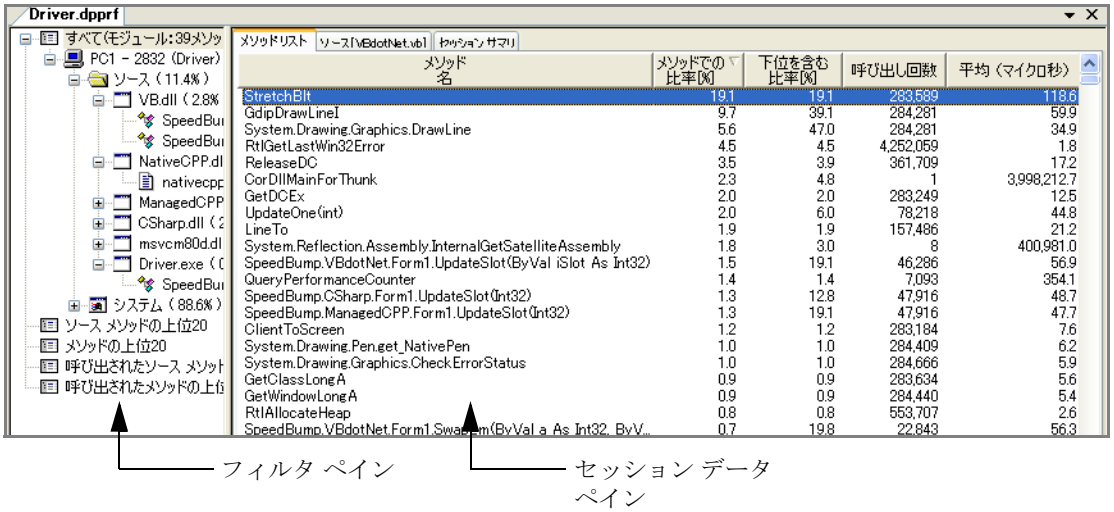


図 6-1. パフォーマンス分析のセッション ウィンドウ

フィルタ ペインとセッション データ ペインの使用

データの評価を開始するには、フィルタを使用し、**メソッド リスト**を確認してプログラムの処理時間のうち大きな割合を占めるメソッドを見つけることから始めます。

1 **フィルタ** ペインの**[ソース メソッドの上位20位]**フィルタをクリックします。これで表示データが少なくなるので、ソースのメソッドに集中できます。

システム ファイルに使用された時間を知ることはパフォーマンスの評価時には役立ちますが、このフィルタを使用して表示からシステム ファイルを除外すると、パフォーマンスの調整作業の対象を絞りやすくなります。

2 **[メソッド リスト]**タブのデータを確認します。**[メソッド リスト]**タブには、各メソッドに使用された時間の合計に関する情報が表示されます。

メソッド リストを確認して1つまたは複数のカラムで高い値のメソッドを探すと、パフォーマンス改善のターゲットを特定の領域に絞ることができます。

3 **[メソッド リスト]**タブの**[メソッドでの比率（%）]**カラムに注目します。このカラムには、メソッドに使用された時間が、そのセッションに使用された時間に占めるパーセント値で表示されます（デフォルトで、データは**[メソッド比率（%）]**カラムの降順でソートされます。ソート順を変更するには、カラム見出しをクリックします）。

4 **[下位を含む比率[%]]**カラムに注目します。このカラムには、メソッドとその下位メソッドに使用された時間が、そのセッションに使用された時間に占めるパーセント値で表示されます。

5 **[平均]**カラムに注目します。このカラムには、メソッドの平均実行時間が表示されます。

これらのカラムの値を確認すると、改善のターゲットとするコードの領域がわかります。

コール グラフの表示

上位メソッドと下位メソッド間で行われた呼び出しのコンテキストでのみ明らかになるパフォーマンス問題があります。このような場合、**コール グラフ**を確認すると役に立ちます。**コール グラフ**は、アプリケーションのメソッドを呼び出す関係を表したグラフィックです。

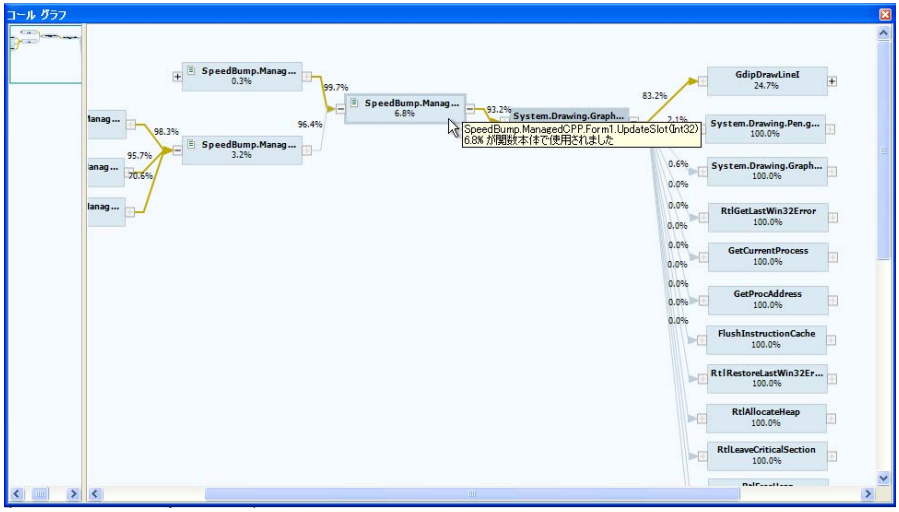


図 6-2. コール グラフ

コール グラフには、**[メソッド リスト]**タブまたは**[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**の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、クリティカル パスがデフォルトのシステム カラーで強調表示されます。

メモ： **コール グラフの基本的な使用方法について以下の手順で説明します。コール グラフの詳細については、「コール グラフの分析」（233 ページ）を参照してください。**

7 ノードの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（左）または下位（右）ノードが展開されるか、折りたたまれます。ノードに表示されるパーセント値と、下位ノードへ伸びる線に表示されるパーセント値を比較して、パフォーマンス問題を引き起こしている可能性があるパスをたどります。

8 ノードまたはメソッド ノード間のリンク上のパーセント値にマウスを移動すると、詳細情報が表示されます。

9 パフォーマンス改善のターゲットとなるメソッドを特定します。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、ソースは使用できません）。

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、フォーカスはコール グラフのままです。

10 **コール グラフ**を閉じてソース コードの編集を開始します。

ソース コードの表示

[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]タブを使用して、パフォーマンス問題を引き起こしている可能性があるコード行を特定します。

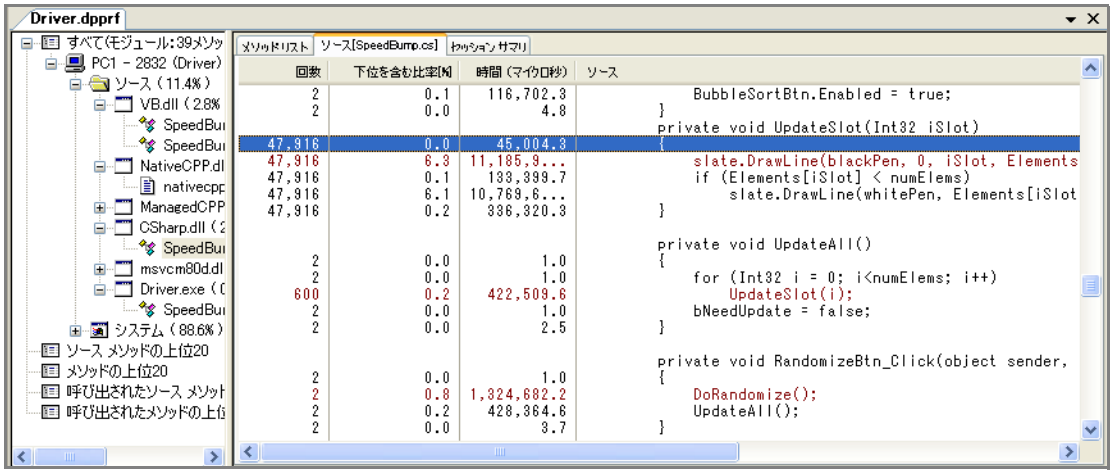


図 6-3. [ソース]タブ

11 図6-3のように、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、実行済みの各コード行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。

各メソッドで最も遅い行が強調表示されます。パフォーマンス改善の余地があるかどうかを判断し、その判断に従ってコードを変更します。

セッションの比較

パフォーマンス問題を特定し、修正したら、もう一度パフォーマンス セッションを実行して、変更前と変更後のセッション ファイルを比較します。セッション間の差異を示す比較ウィンドウが表示されます。セッションの比較の詳細については、「セッションの比較」（236ページ）を参照してください。

セッション サマリ データの表示

[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。

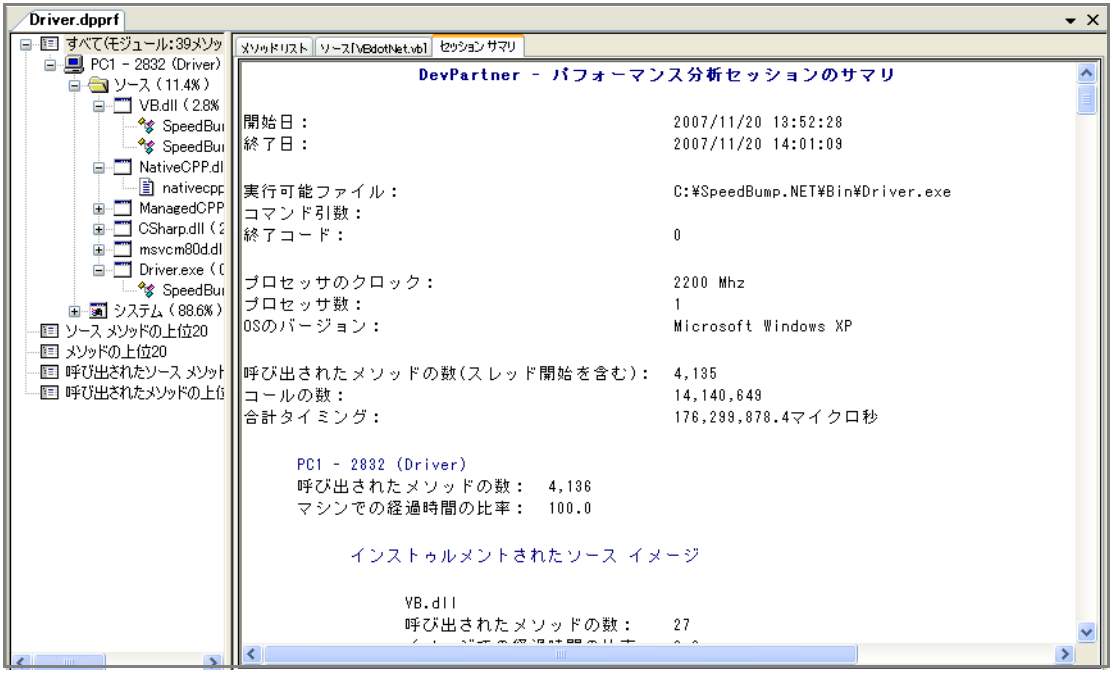


図 6-4. [セッション サマリ]タブ

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、セッションの状況に関する情報が表示されます。たとえば、セッションの日時、プロセッサ速度、オペレーティング システムなどです。この情報は、古いセッション ファイル、特に他の人が作成したセッション ファイルを表示したときに便利です。

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、分析したファイルとメソッド両方のデータが表示されます。

13 タブ内をスクロールすると、セッション サマリ データが表示されます。

セッション ファイルの保存

パフォーマンス データの確認が終わったら、セッション ファイルを保存できます。

1. Visual Studio でセッション ファイル ウィンドウを閉じます。セッション ファイルの保存を確認するメッセージが表示されます。
2. **[OK]** をクリックしてデフォルトのファイル名と場所を受け入れます。

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。保存されたファイルは、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セッション ファイルの拡張子は **.dpprf** です。

デフォルトで、セッション ファイルはプロジェクトの出力フォルダに物理的に保存されます。また、デフォルト ディレクトリの内容に基づいて、ファイル名の番号が自動的に 1 つ増えます（たとえば、**MyApp.dpprf**、**MyApp1.dpprf** など）。セッション ファイルをデフォルト ディレクトリとは別の場所に保存する場合は、自分でファイルのネーミングとナンバリングを管理する必要があります。

Visual Studio 2005 の Web サイト プロジェクトなど、出力ディレクトリがないプロジェクトの場合、ファイルはプロジェクト ディレクトリに物理的に保存されます。

コマンド ラインから生成されたセッション ファイルは、自動的にプロジェクトのソリューションへ追加されません。外部で生成されたセッション ファイルは、Visual Studio で開いているソリューションに手動で追加できます。

この章の準備、設定、実行セクションはこれで終了です。ここまでで、パフォーマンス分析セッションの実行方法について基本的な知識が習得できたはずです。詳細情報については、この章の残りを続けて読んでください。また、タスクに基づく情報については 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の設定

パフォーマンス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、除外したりするために、データ収集を調整すると便利です。**ソリューションのプロパティ**、**プロジェクトのプロパティ**、**DevPartner のオプション**を使用すると、分析セッションのフォーカスを調整できます。

ソリューションのプロパティ

ソリューション レベルで使用できるパフォーマンス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 6-5.　ソリューションのプロパティ

以下のソリューション プロパティはパフォーマンス分析に影響を与えます。

- **[.NET から収集]** – .NET アセンブリの情報を収集しない場合、このプロパティを [False] に設定します。
- **[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、分析の開始前に、その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メッセージが表示されます。

プロジェクトのプロパティ

プロジェクト レベルのプロパティを確認するには、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、ソリューション内のプロジェクトについて設定できるプロパティを確認します。

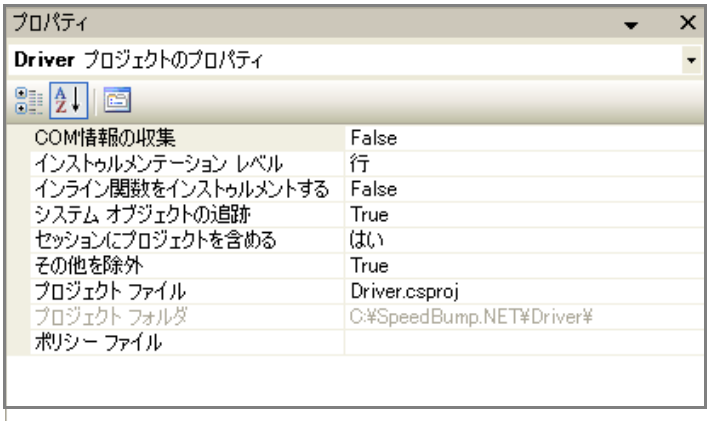


図 6-6. プロジェクトのプロパティ

以下のプロジェクト プロパティはパフォーマンス分析に影響を与えます。

- **[COM情報の収集]** – DLLのエクスポートと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に関する情報をDevPartnerで収集しない場合、[False]を選択します。
- **[その他を除外]** – 実行中の他のアプリケーションのスレッドで使用された時間を除外します。結果のセッション データには、アプリケーションのスレッドに使用された時間のみが含まれます。

  パフォーマンス情報を収集している間、DevPartnerはコンテキストの切り替えを監視して、アプリケーション外部のスレッドで行われる作業に使用されるCPU時間が追跡されます。タイミング データの収集後、DevPartnerはクロック タイムから他のスレッドに使用された時間を差し引いて、アプリケーションに使用された正確な時間を決定します。

  この機能を有効にするには[True]、無効にするには[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には、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」（221ページ）を参照してください）。インラインの最適化が有効な場合、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++アプリケーションを分析しているとき、**[インライン関数をインストゥルメントする]**プロパティで**[True]**を選択した場合でも、`__forceinline`キーワードで明示的にインライン化された関数のデータは収集されません。

- **[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」(221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を使用すると、パフォーマンス分析セッションの実行速度は速くなりますが、メソッドレベルのデータしか表示されません。
  - ◇ **行**：行レベルのインストゥルメンテーションを使用すると、ソース コードの特定の行までドリルダウンできます。.NETアプリケーションを分析している場合、[インストゥルメンテーション レベル]に[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、アセンブリに関する行レベル データが提供されません。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 データを収集するには、パフォーマンス分析を実行する前にアセンブリをJITコンパイルしないでください。

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。

図 6-7. 分析オプション

- **[表示]**オプションを使用すると、データの表示に使用される精度、位取り、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省略できます。除外の詳細については、「イメージの除外」(220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、ルールとアクションのセットを作成し、アプリケーションまたはモジュールの実行時にDevPartnerで収集するデータを制御できます。セッション コントロール ファイルの詳細については、「分析のセッション コントロール」（353ページ）を参照してください。

[環境]>[フォントと色]オプションなど、他のVisual StudioオプションもDevPartner機能に影響があります。

イメージの除外

パフォーマンス分析でアプリケーションを実行すると、すべてのソースとシステム イメージのデータが収集されます。ただし、[除外]オプションを使用すると分析から1つまたは複数のイメージを省略できます。

[分析オプション]（**[DevPartner]>[オプション]>[分析]**）を表示中に、**[除外－パフォーマンス]**を選択します。

ページの上部にある**[表示]**リストから以下のいずれかを選択します。

- グローバル除外
- 現在のディレクトリ内のローカル除外
- 実行可能ディレクトリ内のローカル除外

[実行可能ディレクトリ内のローカル除外]オプションを使用できるのは、ソリューションが開いていて、実行可能ファイルのディレクトリが現在の作業ディレクトリと異なる場合のみです。

[挿入]  をクリックし、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。除外に指定できるファイルの種類は**.exe**、**.dll**、**.ocx**、**.netmodule**です。表示するファイルの種類を制限するには、**[ファイルの種類]**リストを使用します。

.NETモジュール（**.netmodule**）を選択すると、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、**[削除]**  をクリックします。

インストゥルメントされていないシステムDLLをDevPartnerのパフォーマンス プロファイルから除外するには、**[システム イメージを除外する]**チェック ボックスをオンにします。

グローバル除外は、DevPartnerインストール ディレクトリの**¥Analysis**サブディレクトリの**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ァイル ディレクトリ、または現在の作業ディレクトリの**nmexclud.txt**に保存されます。除外リスト（**nmexclud.txt**）のコピーを別の場所に保存するには、**[保存場所]**をクリックします。

メモ： 実行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 を常にプロファイルしています。そのため、特定のシステム DLL を個々に除外することはできません。また、**[システム イメージを除外する]** を選択してすべてのシステム イメージを除外しないかぎり、システム DLL はセッション 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。

[ネイティブ C/C++ インストゥルメンテーション] でコンパイルしたファイルには除外は適用されません。たとえば、インストゥルメントされたアンマネージ C/C++ イメージを除外しようとしても、そのファイルの情報は収集されます。ただし、システム コール情報は収集されません。アンマネージ C/C++ イメージをデータ収集から除外するには、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、コンパイラから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。このプロセスはインストゥルメンテーションと呼ばれます。このコードには、アプリケーションの実行中、パフォーマンス データの収集に使用される指示が含まれます。DevPartner のインストゥルメンテーションによって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るだけです。

DevPartner によって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ユーザーが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ために、DevPartner はソース コードに直接フックを挿入します。DevPartner が提供するインストゥルメンテーション マネージャ機能を使うと、使用するインストゥルメンテーションの種類を指定し、インストゥルメンテーションから除外するソリューション内のプロジェクトを指定することができます（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 コードからのデータ収集」（222 ページ）を参照してください）。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 メニューから [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無効にしてから、プロジェクトをリビルドします。

さまざまなアプリケーションからのデータの収集

このセクションでは、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ときに DevPartner パフォーマンス分析を使用する方法について、情報を提供します。

DevPartner では、Visual Studio のすべてのマネージ コード言語とアンマネージ C/C++ をサポートしています。Visual Basic アプリケーションや Visual C++ 6.0 アプリケーションのパフォーマンス データを収集できるだけでなく、Internet Explorer または IIS を使用すると JScript アプリケーションや 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言語とプロジェクト タイプの詳細については、付録B「DevPartner Studioで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 コードからのデータ収集

Visual Studioで開発する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Visual C++アプリケーションのようにマネージ アプリケーションになります。

マネージ アプリケーションのソース コードに関する詳細情報を収集するために、DevPartnerはPDB（プログラム データベース ファイル）情報を必要とします。ソース データが[ソース]タブに表示されない場合、またはソース ファイルが**フィルタ** ペインに表示されない場合、**.pdb**ファイルが生成されていることを確認します。

PDB情報がないマネージ 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しているときに、セキュリティ ポリシーによってコードのDevPartnerインストゥルメントが回避された場合、セキュリティ エラー 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。コードの実行に使用しているポリシーの権限セットからこの権限を削除した場合、またはこの権限を取り消すような厳しいセキュリティ宣言をアセンブリに追加した場合、アセンブリはプロファイルできません。

この状況を修正するには、以下のいずれかの方法で、安全なプロファイルを有効にします。

- 以下のグローバル環境変数を設定し、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
  ```

  この方法で問題を回避できますが、パフォーマンスがやや低下します。

- .NET Framework構成ツールのMMCスナップインを使用するか、アセンブリに含まれる厳しいセキュリティ宣言を一時的に削除することで、アセンブリのポリシーを変更します。

Visual Studioのセキュリティ ポリシーの詳細については、Visual Studioのオンライン ヘルプに含まれる「.NET Framework開発者ガイド」を参照してください。

アンマネージ コードからのデータ収集

パフォーマンスのプロファイルのために**[ネイティブC/C++インストゥルメンテーション]**を使用してアンマネージVisual C++アプリケーションをビルドすると、DevPartnerとコンパイラが連携して、実行時にパフォーマンス データを収集する指示がアプリケーション イメージに追加されます。たとえば、メソッドの開始時と終了時に毎回DevPartnerが呼び出されます。この情報は、メソッドの実行時間を決定するときに使用されます。

アンマネージ コードをインストゥルメントするには、データを収集するアンマネージ 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。

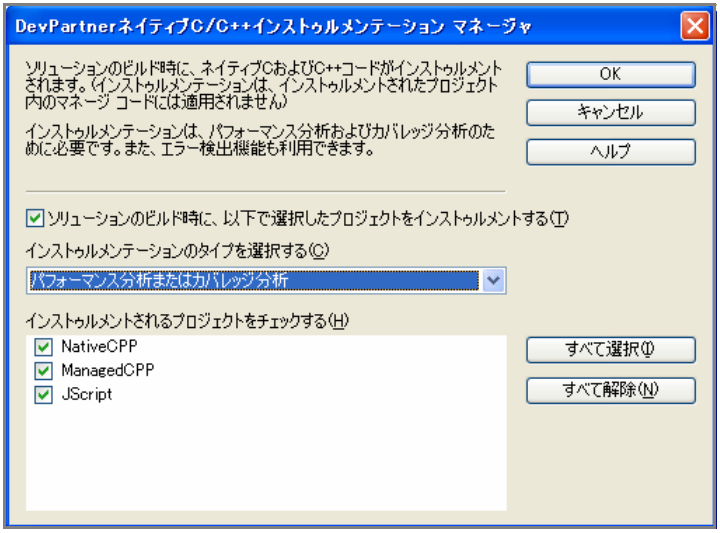


図 6-8. インストゥルメンテーション マネージャ

[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、インストゥルメンテーションの種類を選択します。選択するインストゥルメンテーションの種類は、このあと実行する分析の種類と一致させる必要があります。

インストゥルメントするプロジェクトを選択します。デフォルトでは、ソリューションに含まれる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。省略するモジュールの選択を解除します。

[OK] をクリックし、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。

[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] で選択したプロジェクトがソリューションと共に保存されます。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したあと、インストゥルメンテーションのオン／オフを切り替えるには、[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、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]** を使用するのは、設定を変更する場合のみです。

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]** オプションをオフにします。次にパフォーマンス分析セッションを開始したとき、またはソリューションをリビルドしたときは、インストゥルメントなしでソリューションがリビルドされます。

メモ： アプリケーションから Visual Studio 6.0 コンポーネントを呼び出している場合、Visual Studio 6.0 でパフォーマンス分析のための DevPartner インストゥルメンテーションを使って、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 6.0 で DevPartner Studio のオンライン ヘルプを参照してください。

サポートされない混合コード

Visual C++ では、`/clr` オプションを使用して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が、コードのセクションに `#pragma`（ネイティブ）のマークが付きます。コンパイラは、`#pragma` セクションに定義されているすべてのメソッドについて、ネイティブ コードを生成します。DevPartner では混合モードの C++ ファイルをサポートしていません。マネージ セクションとアンマネージ セクションの両方を含むプログラムをプロファイルしようとすると、マネージ コード部分についてのみパフォーマンス データが収集され、アンマネージ コード部分についは収集されません。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 コードを別のファイルに配置し、そのファイルをインストゥルメントします。詳細については、「アンマネージ コードからのデータ収集」（222 ページ）を参照してください。

DevPartner for Visual C++ BoundsChecker Suite での混合コード

DevPartner for Visual C++ BoundsChecker Suite を使用して、マネージ プロジェクトとアンマネージ プロジェクトが含まれるソリューションを開くと、[DevPartner] メニューの以下のコマンドが使用できなくなります。

カバレッジ分析を選択して開始
エラー検出とカバレッジ分析を選択して開始
デバッグを実行せずにパフォーマンス分析を選択して開始
デバッグを実行せずにメモリ分析を選択して開始
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

Visual Studio 内からアンマネージ コードを分析するには、混合コード ソリューションからアンマネージ　プロジェクトのみを含む新しいソリューションを作成し、**[DevPartner]>[ネイティブ C/C++ インストゥルメンテーション]** を選択してソリューションをリビルドし、分析セッションを実行します。

複数のプロセスからのデータ収集

アプリケーションでは、複数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）、ASP ワーカー プロセス（aspnet_wp または w3wp）が含まれている場合があります。

パフォーマンス分析でマルチプロセス 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。

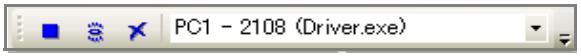


図 6-9.　**プロセス選択リストが含まれるセッション コントロール ツールバー**

このプロセス選択リストを使用して、データ収集のフォーカスを設定できます。スナップショットを作成すると、プロセス選択リストで選択したプロセスのデータを含むセッション ファイルが作成されます。

リモート システムからのデータの収集

リモート システムで実行されているアプリケーション 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側とサーバー側の両方のパフォーマンス データを収集できます。DevPartner では、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、クライアントに DevPartner をインストールし、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 ライセンスの詳細については、『DevPartner インストール ガイド』（**DPS Install.pdf**）および『Distributed License Management ライセンス ガイド』（**compuware_licensing_guide_J.pdf**）を参照してください。

メモ：　**ターミナル サービス接続経由で接続するサーバーには、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 デスクトップの使用」（9 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 プロパティがクライアント 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。オプションを変更すると、DevPartner によって、IIS などのサーバー プロセスが再起動されます。再起動は、変更内容を有効にするために必要な処理です。

アンマネージ C++ アプリケーションを分析する場合、必ずインストゥルメンテーションを指定してください。アプリケーションからアンマネージ C++ コンポーネントが呼び出される場合、そのコンポーネントからデータを収集するには、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。詳細については「アンマネージ コードからのデータ収集」（222 ページ）を参照してください。

データの関連付け

Internet Explorer (IE) と Internet Information Server (IIS) をブラウザと Web サーバーとして使用する場合、または COM を使用してプロセス間の呼び出しを行う場合、DevPartner はプロセス間のクライアント／サーバーの関係を自動的に認識します。DevPartner は、DCOM オブジェクトのメソッド間の関係、または HTTP クライアントとサーバー（IE と IIS）の関係を維持するために、そのセッションからのデータを自動的に関連付けます。そして、相関データとクライアントのセッション データが結合されて、1 つのセッション ファイルが作成されます。

相関セッション ファイルには、アプリケーションのクライアント側とサーバー側の両方のパフォーマンス データが含まれています。相関セッション ファイルは、他のセッション ファイルと同様に Visual Studio に表示されますが、**appname_CO.dpprf** のようにファイル名に **_co** が付加されます。

[コール グラフ] ウィンドウに相関セッション ファイルを表示すると、呼び出し元のメソッドから呼び出し先のメソッドへ COM コール スタックをたどることができます。サーバー側のデータは、クライアント システムのクロック速度に合わせてスケーリングされます。

COM に基づく関係や、IE と IIS 間のクライアント／サーバー関係がない場合に異なるセッション ファイルからデータを手動で結合するには、**[DevPartner]>[関連付け]>[パフォーマンス ファイル]** を選択します。また、「コマンド ラインからの分析の開始」（333 ページ）のように、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 アプリケーションからのデータの収集

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側とサーバー側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 マシンまたはリモート マシンで実行されている IIS と ASP.NET のデータを収集するように DevPartner を設定できます。

アプリケーションから呼び出されるアンマネージ C++ コンポーネントのデータを収集するには、**[ネイティブ C/C++ インストゥルメンテーション]** を使用してオブジェクトのインストゥルメントとリビルドを行う必要があります。詳細については、「アンマネージ コードからのデータ収集」（222 ページ）を参照してください。Web アプリケーションから Visual Basic 6.0 または Visual C++ 6.0 コンポーネントを呼び出す場合、Visual Studio 6.0 で DevPartner コマンド ラインを使用して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。パフォーマンス分析のためには、必ずインストゥルメントしてください。DevPartner では、1 つのセッションで 1 種類の分析についてのみデータが収集されます。

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。IIS から Web オブジェクトを直接開くと、Visual Studio でプロジェクトを開くときとは異なり、別のソース ファイルが使用されることがあります。最初の

ソリューションで作成されたDevPartnerセッション ファイルは、2回め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を正常に実行するには、以下の条件を満たす必要があります。

- プロジェクトに**web.config**ファイルが含まれる必要があります。
- **web.config**ファイルには、debug属性をtrueに設定したcompilationエレメントを含める必要があります。以下に例を示します。
  `<compilation debug="true"/>`

また、アプリケーションから呼び出されるプロセス内のコンポーネントまたはプロセス外のコンポーネントのデータも収集できます。

デバッグを実行しないASP.NETアプリケーションの分析

最適な結果を得るためには、パフォーマンス分析をデバッグなしで実行します。

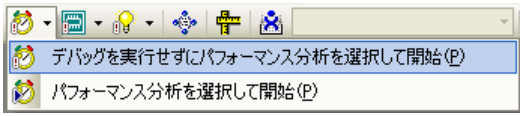


図 6-10. [デバッグを実行せずに... を選択して開始]オプション

同時にアクティブにできるのは1つのスクリプト デバッガのみです。デバッグ機能を使用してWebアプリケーションをデバッグする場合、Visual StudioとDevPartnerはスクリプト デバッガをロードしようとします。スクリプト デバッガがIEへのアタッチに失敗したというメッセージが表示されます。エラー メッセージがあってもセッションは中断することなく継続します。

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、デバッグを実行せずにパフォーマンス分析を実行します。

予期しない[ファイルの保存]ダイアログとセッション ファイルの保存

場合によっては、ASP.NETアプリケーションの終了後に予期しない**[ファイルの保存]**ダイアログ ボックスが表示されたり、セッション ファイルを自動的に保存するようにDevPartnerを設定している場合、予期しないセッション ファイルの保存が行われたりすることがあります。

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）などの二次プロセスに関するセッション データも保存されます。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、IEのクライアント データとIISおよびASP.NETワーカー プロセスのサーバー データの両方を含む、最終的な相関セッション ファイルを生成します。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作成することもできます。

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操作で終了します。ただし、プロファイル中に、ASP.NET ワーカー プロセスが自動的にシャットダウンすることもあります。これは、プロセスを実行しているシステムで、**machine.config** ファイルの processModel Attributes セクションを以下のように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のシャット ダウンが発生するくらい低い値に変更した場合
- 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のシャットダウンが発生するくらい低い値に変更した場合
- memoryLimit 属性の値を、セッション中にプロセスのリサイクルが発生するくらい低いパーセント値に変更した場合

プロセスがシャットダウンすると、最後のスナップショットが作成され、セッション ファイルが生成されます。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、そのセッション ファイルが Visual Studio で開かれ、ソリューションに追加されます。この動作は、ASP.NET ワーカー プロセスが実行され、終了するたびに繰り返されます。
- 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作成されると、IE のセッション ファイルと相関セッション ファイルが作成されます。相関セッション ファイルには、この時点までに実行および終了された IE、IIS、および 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が含まれます。

分析セッションが終了すると、DevPartner は **[ファイルの保存]** ダイアログ ボックスを表示するか、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することで余分なセッション ファイルが生成されないように、**machine.config** ファイルを編集し、プロセスが早期に終了しないような高い値に制限属性を設定します。

注意： **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 アプリケーションからのデータの収集

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 と VBScript のソース 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またはプロセス外のコンポーネントを呼び出す場合、そのデータも収集されます。

スクリプト言語のインストゥルメンテーションは、マネージ .NET 言語の場合と同様に、実行時に発生します。ただし、監視するアンマネージ コンポーネント（COM オブジェクトなど）は、すべて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 アプリケーションに独自のものです。Visual Studio で開発した Web フォーム、XML Web サービス、ASP.NET アプリケーションのデータを収集するには、他の .NET アプリケーションを実行する場合と同様に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]>[Compuware DevPartner Studio]>[ユーティリティ]>[Web Script 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、セッション コントロール ツールバーが表示されます。このツールバーを使用してデータの収集を制御できます。

DevPartner が有効な 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 アプリケーションが開き、アプリケーションを実行します。セッション コントロール ツールバーを使用すると、アプリケーションを実行しながらデータ収集のフォーカスを制御できます。

Internet Explorer を終了します。またセッション コントロールを使用している場合、**[停止]** アクションを実行します。セッションの保存を促すダイアログ ボックスが表示され、セッション ファイルが自動的に保存されます。

Web アプリケーションのデータ収集のヒント

分析データの収集を開始する前に、以下のことを行います。

- アプリケーションを実行して、数分間「ウォームアップ」します。対象となるアプリケーション部分が含まれていることを確認します。
- **クリア** セッション コントロール 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、アプリケーションを起動したときに行われる 1 回限りの初期化のデータを除外できます。
- 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 データが提供されます。
- HTML ページを完全にロードする時間を考慮に入れます。手動でテストするときは、ページがロードされるまで待ちます。自動テストのためにスクリプトを

作成する場合は、ページが完全にロードされるように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を実行すると、プロファイル データが不正になる可能性があります。

- キャッシュには注意してください。Web アプリケーションは、アプリケーション コードを実行するのではなく、キャッシュからページを返すことがあります。テストに同じ入力データを繰り返し使用する場合、キャッシュによって結果が不正になります。キャッシュの影響を排除したい場合、**machine.config** ファイルを編集して以下の行をコメント アウトすることで、キャッシュを無効にすることができます。

```
<add name="OutputCache"
type="System.Web.Caching.OutputCacheModule"/>
```

注意： **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 サービスの要件

DevPartner パフォーマンス分析で Web サービスを検出する場合、そのサービスは以下の要件の少なくとも 1 つを満たす必要があります。

- Web サービスは **System.Web.Services.WebService** 基本クラスから派生する必要があります。
- Web サービスには WebService 属性が含まれる必要があります。

DevPartner パフォーマンス分析で Web メソッドを検出する場合、そのメソッドには WebMethod 属性が含まれる必要があります。

NMSource からの一時ファイルの削除

IE または IIS でスクリプトのパフォーマンスを分析している間、スクリプト ソースの一時的なコピーを保持するために **NMSource** ディレクトリが作成されます。セッション データの分析中、このソースはセッション ウィンドウの [ソース] タブに表示されます。

このソースはいつか必要になることもあるため、DevPartner は **NMSource** からこのファイルを削除しません。特に IIS でサーバー プログラムを分析している場合、このディレクトリのサイズは急速に増える可能性があります。

定期的に **NMSource** ディレクトリのソース ファイルを確認し、アクティブでないプロジェクトに関連するファイルを削除してください。**NMSource** は **¥Program files ¥Internet Explorer** ディレクトリにあります。

データ収集のための IIS の設定

ローカル マシンまたはリモート サーバーで実行されている IIS アプリケーションまたは 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、以下の設定オプションを設定します。

メモ： **IIS をローカル システムで実行している場合、ローカル システムに以下のオプションを設定します。IIS がリモート サーバーで実行されている場合、そのシステムに DevPartner（およびリモート サーバー ライセンス）をインストールし、リモート システムに以下に示す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

以下のオプションを、Internet Information Services マネージャの [規定の Web サイトのプロパティ] 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 に適用されます。

[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ているとき、場合によっては、[アプリケーション保護] オプションを変更して DevPartner によるパフォーマンス分析データの収集を有効にする必要があります。以下のオプションを、Internet Information Services マネージャの [規定の Web サイトのプロパティ] 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 に適用されます。

[ホーム ディレクトリ] タブまたは [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。

データ収集のための Internet Explorer の設定

Internet Explorer からパフォーマンス分析データを収集するには、以下を選択します。**[ツール]>[インターネット オプション ...] [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）]** をオフにします。

サービスからのデータの収集

サービスの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の分析の開始」（333 ページ）を参照してください。

COM／COM+ アプリケーションからのデータの収集

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 および .NET オブジェクト（COM+）を混合して使用するアプリケーションをプロファイルした場合、アプリケーションの .NET 部分については行レベルのデータが収集されます。アンマネージ コードのコンポーネントについては、[ネイティブ C/C++ インストゥルメンテーション] で事前にインストゥルメントしている場合は、行レベルのデータが収集されます。また、Visual Studio 6 COM オブジェクトについても、パフォーマンス データ収集用に事前にインストゥルメントしている場合は、行レベルのデータを収集できます。これを行うには、Visual Studio 6 で、プロジェクトをパフォーマンス分析用のインストゥルメンテーションを使ってビルドします。

Visual Studio 6.0 オブジェクト、Visual C++ 6.0 オブジェクト、またはインストゥルメントされていない任意のアンマネージ コード コンポーネントをプロファイルした場合、COM インターフェイスと DLL の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。

再帰関数のデータの収集

再帰を使用するアプリケーションのリテラル プロファイルには、再帰関数の二重のカウントが含まれます。DevPartner では、すでに関数が測定されていることを検出することでこの重複を排除します。つまり、最初の関数呼び出しの測定を停止し、2 回めの呼び出しに対して新たに累積を開始します。再帰関数に関する DevPartner のデータ収集処理の詳細については、DevPartner Studio のオンライン ヘルプを参照してください。

コール グラフの分析

コール グラフは、アプリケーションのメソッドを呼び出す関係を表したグラフィックです。コール グラフの使用については、この章の準備、設定、実行手順で概要を説明しました。このセクションでは、**コール グラフ**の使用に関して詳細に説明します。

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]**タブからメソッドを選択し、右クリックして**[コール グラフの表示]**を選択します。独立した**[コール グラフ]**ウィンドウが表示されます。

コール グラフには、特定のメソッド コールまでのコール チェーンと、そのメソッドから順に呼び出されるメソッドが表示されます。

ノードは、呼び出された順に左から右に表示されます。初期状態で**コール グラフ**に表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示します。あるノードの左にあるノードは「上位ノード」と呼ばれます。ノードの右にあるノードは「下位ノード」と呼ばれます。

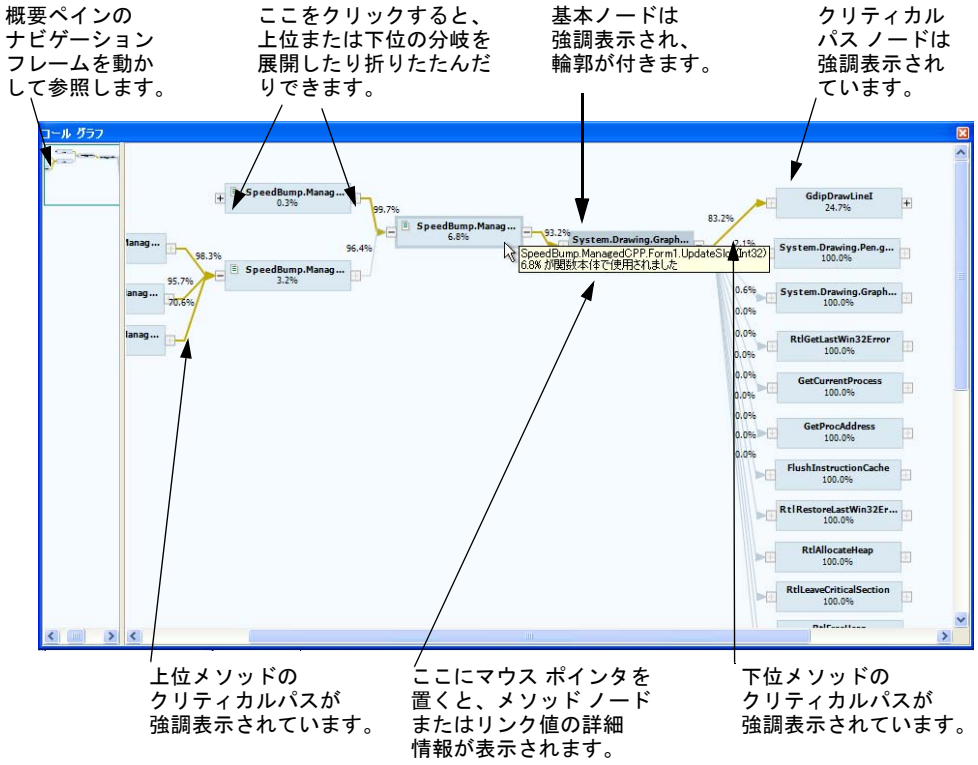


図 6-11.　　コール グラフ

コール グラフは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。**コール グラフ**のノード数が多すぎてスクロールしないと右側のフレームに表示しきれない場合、全体の**コール グラフ**を確認できます。右側のフレームのノードを展開したり折りたたんだりすると、概要が自動的に更新され、現在のビューが表示されます。また、概要のナビゲーション フレームを動かすと、右側に表示されるグラフの部分が変化します。右側のフレームの任意の場所を右クリックし、**[概要を表示]** オプションをオフにすると、概要を閉じることができます。
- 右側のフレームには、基本メソッド ノードと、そのノードから呼び出されるかそのノードを呼び出すすべてのメソッドが表示されます。展開／折りたたみボックスを使用して、選択したノードの右側または左側にあるノードの表示／非表示を切り替えます。

  各ノードに表示されるパーセント値は、ノードが使用している時間のパーセント値を示します。各下位ノードに伸びている線に表示される値は、下位パスが使用している時間を、上位ノードが使用している合計時間に占めるパーセント値で示したものです。

クリティカル パス

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についてクリティカル パスが計算されます。クリティカル パスとは、そのメソッドとすべての下位メソッドに起因する、時間の最も多い割合を占めるメソッド コールのシーケンスです。

コール グラフのナビゲート

ノードをウィンドウの別の位置にドラッグすると、**コール グラフ**の線が自動的に再描画されます。メソッド数が多すぎて画面が見づらい場合、または最初の基本ノード、上位ノード、下位ノードの表示が占める画面の領域を小さくする場合、この操作は便利です。

デフォルトで、下位ノードのみが展開された形式で表示されます。基本ノードの上位ノードは表示されません。基本ノードの左側にあるプラス アイコンをクリックすると、基本ノードの上位ノードが表示されます。完全なパスを表示するには、プログラムが最初に実行したメソッド（通常、`Program Start` という名前）に到達するまで、各上位ノードの左にあるアイコンを展開する必要があります。

個々のノードまたはノードのグループを選択できます。複数のノードを選択するには、1つのノードを選択し、[Ctrl] キーまたは [Shift] キーを押しながら他のノードを選択します。選択した複数のノードはグループとしてドラッグできます。

ソース コードの表示

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、コンテキスト メニューの **[メソッドの表示]** を選択します。基本ノードのソース コードのみを表示できます。

下位側の分析

コール グラフの下位（右）側を分析して、何を最適化すべきかを検討します。

下位ノードを展開し、基本メソッドと下位メソッドのどちらが長い経過時間の原因となっているのかを分析します。

- 基本ノードに複数の並列分岐がある場合、最初の下位メソッドへのリンク上の値が最大である分岐を探します。値が大きい分岐を最適化すると、パフォーマンスの向上に効果的です。
- 基本メソッド自体の値が大きい場合は、基本メソッドを最適化することを検討してください。
- 基本メソッドによる経過時間のうち下位分岐の割合が大きい場合は、パーセント値の高い分岐にある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、基本分岐を最適化する価値があるか、または基本ノードの呼び出しをなくすか回数を減らすことができるかを検討します。

基本ノードの左側にある上位ノードを展開します。特に、上位分岐における基本ノードの経過時間を入念に調べてください。これは、基本ノードまたは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のに役立ちます。プログラム実行時間全体において、複数の上位メソッドまたは主要上位メソッドの時間のうち基本メソッドによる経過時間が占める割合が大きい場合、その基本メソッドの最適化を検討する価値があると考えられます。

メモ： **基本ノードとその各上位ノードとのリンク上の値は互いに依存しない独立した値であり、加算値ではありません。各パーセント値は、その上位ノードが費やした時間に対する基本ノードの時間を示します。**

- 上位ノードが複数あり、その基本ノードへのリンクのうち1つまたは複数の値が大きい場合、その基本ノードは最適化の候補になります。
- 基本ノードから上位ノードへのリンク上の値が非常に小さい場合は、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効果はありません。
- その基本ノードが最適化の対象として最適かどうかを判断するために、上位メソッ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。これによって、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どのくらい重要かが、他の下位ノードとの対比でわかります。

コール グラフの上位または下位側の分析では、ノードを右クリックして開くコンテキスト メニューから、そのメソッドのソース コードにアクセスして、実行時間が長い理由を随時調べることができます。

セッションの比較

プログラムのパフォーマンスを調整するには、コストが一番高いコードの部分を調整できるように、まず最も長い実行時間がかかっている場所を特定する必要があります。次に、調整がパフォーマンスに与える影響を比較します。

DevPartner を使用すると、パフォーマンス セッション間の結果を比較できるため、最適化によって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が受ける影響を確認できます。

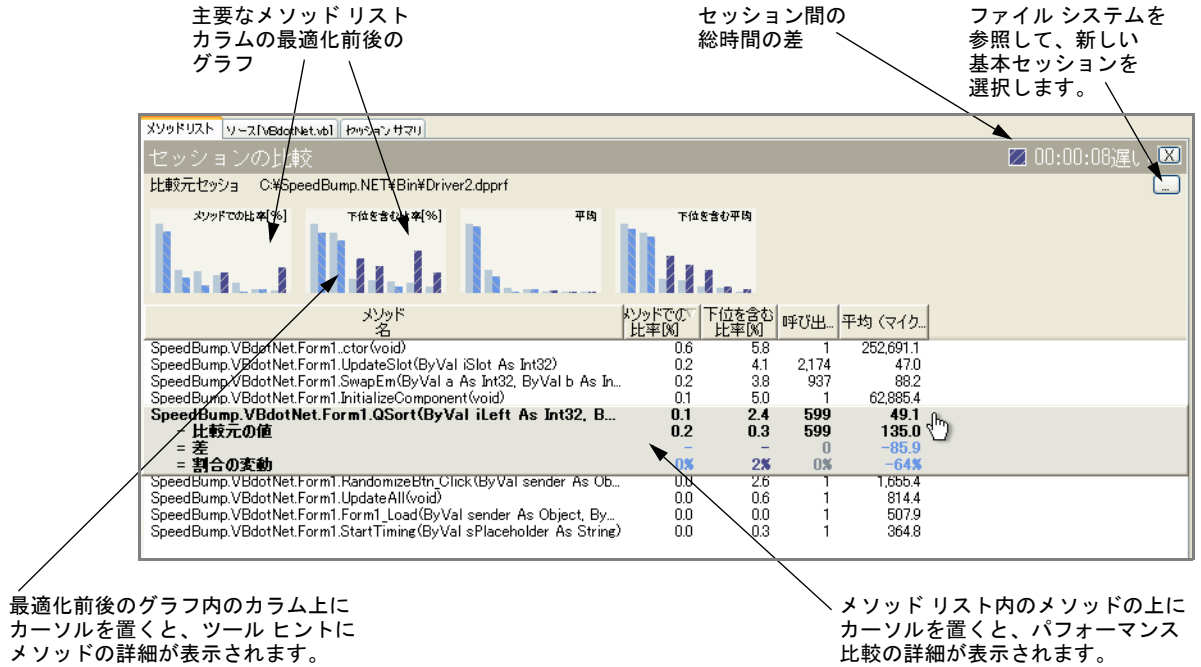


図 6-12. パフォーマンス セッションの比較

セッションの比較機能を呼び出すには、セッション ウィンドウを開いているときに **[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から使用できます。

- ツールバー 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 メニューにあるメニュー項目

はじめて **[比較]** コマンドを呼び出すと、基本セッションにするセッション ファイルを選択するようにメッセージが表示されます。デフォルトで、現在アクティブなセッションが現在のセッションとして追跡されます。基本セッションを選択すると、セッション ウィンドウが変化して、メソッド リストの任意のメソッドを配置できるフレームが表示されます。フレームには、基本セッションのメソッドと現在のセッションのメソッドの比較結果が表示されます。

比較するセッションを選択するときは、セッションができるだけ同じ形式で実行されたことを確認します。たとえば、デバッガで起動したセッションと、デバッガ以外で実行したセッションとを比較しないでください。より正確な比較結果を得るためには、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することを検討してください。

比較ウィンドウの右上には、現在のセッションと基本セッションの全体的な時間の差が表示されます。時間データの左側の図には、現在のセッションが基本セッションより時間がかかっているかどうかが表示されます。この図は、同様に実行したアプリケーションのセッションをすばやく比較する場合に便利です。

比較結果には、2つのバージョンの選択したメソッドについて、現在の値、基本の値、差、パーセントの差が表示されます。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 ウィンドウの上部にある4つの棒グラフには、現在のセッションの上位メソッドに関する同じ情報が示されます。

セッション比較データ ボックスの情報はコピーできます。これには、**メソッド リスト**のコンテキスト メニューから**[比較データのコピー]**を呼び出します。このコマンドで、データがクリップボードにコピーされます。

比較を終了するには、**[Esc]**キーを押すか、**[比較]**アイコンをクリックします。

セッション比較結果の解釈

セッション比較は、現在のセッション内のメソッドと基本セッション内でのそのメソッド間の**[現在の値]**、**[基本値]**、**[差]**、**[差の割合]**を表示します。現在のセッションの値が基本セッションの値より大きいか小さいかがひと目でわかるように、色が使用されています。差とパーセントの差の値が濃い青の場合、現在のセッションの値の方は基本セッションよりよい（速い）ことを示しています。薄い青は、現在のセッションのパフォーマンス値の方が遅いことを示しています。

コードの変更が任意のメソッドのセッション間に与えた効果を確認したら、そのセッション内の別のメソッドを探して、コード変更による副次的な影響がないか調べます。個々のメソッドのパフォーマンスが向上しても、プログラムとしてのパフォーマンスが低下する場合があります。パフォーマンスの調整においては、コードの構造をよく知ることが何よりも重要です。

セッション比較結果を確認する際、以下の事項に注意してください。

- パーセントは2つの数値間の比率です。パーセントは、同じ合計値を基準にして計算した場合のみ、加算できます。
- 1つのパーセント値が減少すると、他のパーセント値は増加するはずです。複雑なプログラムではパーセントの増加はプログラム内の他のすべてのメソッド間で平均されるので、簡単には気付かない場合があります。

- サブプログラムのタイミングを解釈するには、上位プログラムにおけるサブプログラムの役割を理解する必要があります。
- パフォーマンス測定値は、その値が計算されたプログラム外では意味を持ちません。プログラムの動作を理解しないで、プログラム変更の効果について一般論を語ることはできません。

プログラム内で最もパフォーマンスの低いメソッドに対する変更が終了したら、その他のパフォーマンスの低いメソッドに注目します。

パフォーマンス データのエクスポート

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。XMLまたはCSV形式でデータをエクスポートすると、自社製またはサードパーティ製のソフトウェアを使用して、データの分析、他のツールで作成したデータとの統合、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（拡張子は`.dpprf`）はXML形式にエクスポートできます。保存したパフォーマンス セッション ファイルを開いているとき、**[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については、「分析データのXMLへのエクスポート」（363ページ）を参照してください。

  また、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。「コマンド ラインからの分析データのXMLへのエクスポート」（364ページ）を参照してください。
- **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ます。**メソッド リスト**を右クリックし、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。カンマ区切りのテキスト ファイルはMicrosoft Excelや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。

データ収集の制御

アプリケーション実行中に収集するデータを制御するには、以下の3つの方法があります。

- プログラムの実行時に、セッション コントロール ツールバーでデータ収集を対話的に制御します。
- 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 モジュールの特定のメソッドに対してセッション コントロールの動作を指定します。
- セッション コントロールAPIを使用して、プログラムのデータ収集を制御します。

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を使用した場合、メソッド内のどのポイントでもデータ収集を制御できます。セッション コントロール ファイルを使用した場合は、メソッドの開始点または終了点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の使用方法については、「分析のセッション コントロール」（353ページ）を参照してください。

コマンド ラインからの分析

データ収集を自動化する場合、またはコマンド ラインから分析セッションを実行する場合、DevPartnerのコマンド ライン実行ファイルである**DPAnalysis.exe**を使用します。**DPAnalysis.exe**の使用方法については、「コマンド ラインからの分析の開始」（333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の使用

DevPartner Studioには、Visual Studio 2003やVisual Studio 2005から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]メニューの**[すべてのプログラム]>[Compuware 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。**パフォーマンス分析ビューア**にセッション データが表示されます。
- Visual C++ 6.0とVisual Basic 6.0の場合は、**[DevPartner]>[パフォーマンス分析を選択して開始]**コマンドからパフォーマンス分析セッションを実行します。パフォーマンス分析ビューアにセッション データが表示されます。

パフォーマンス分析ビューアの機能

セッション ファイルを開くと、パフォーマンス セッション データの表示、ソート、保存、または印刷を行うことができます。また、以下の機能もあります。

- メソッドのソース コードを表示する
- **[メソッド リスト]**タブの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にエクスポートする
- **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実行できない機能

- 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
- ファイルを Visual Studio ソリューションに追加する

メモ： **コマンド ラインから生成されたセッション ファイルは、自動的にプロジェクトのソリューションへ追加されません。外部で生成されたセッション ファイルは、Visual Studio で開いている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をより効率的にするための方法を以下に示します。

- ソース コードを分析する

  [ソース メソッドの上位 20 位] フィルタを使用して、アプリケーションのホットスポットを識別します。

  コール グラフ を使用して、最もコストの高いメソッドを調べ、呼び出した下位メソッドのコストを調べます。

  複数のパフォーマンス セッションを実行し、異なるアルゴリズムを使用したりロジックを変更したりした場合の影響を比較します。

- Framework のコストを調べる

  [メソッド リスト] または **[ソース]** タブの **[下位を含む比率 [%]]** で、.NET Framework に費やす時間を調べます。

  コール グラフ で下位メソッドを検証することによって .NET Framework にドリルダウンし、コストの高いコールとその理由を調べます。

  アプリケーションを手直しして、処理を少なくしたり、.NET Framework の呼び出し回数を減らします。

- スタートアップ コストを調べる

  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、1 回限りの初期化が何度も行われます。パフォーマンス データが初期化の影響を受けないようにするには、プロファイルするすべての機能を試してウォームアップし、そのあとにデータを **クリア** します。さらに、同じ機能をテストし、より正確なパフォーマンス データを取得します。

- 測定対象を調べる

  アプリケーションの動作を考慮してから、パフォーマンス データを収集します。たとえば、Web サービスや ASP.NET アプリケーションをプロファイルする場合は、Web キャッシュがパフォーマンスに与える影響を考えます。同じデータを繰り返し入力してテストすると、アプリケーションはキャッシュからページを読み出すので、パフォーマンス データに影響を与えます。このような場合、わざわざ入力データを変えることもできますが、**machine.config** ファイルを編集

ヒント： システム（ソースなし）ファイルすべてについてデータ収集を行わないようにするには、DevPartner で **[除外 - パフォーマンス]** オプション ページの **[システムの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ら、このオプションをオフにします。アプリケーションでシステム コード、特に .NET Framework がどのように使用されるかを試すことができます。

して、テスト時にキャッシュをオフにすれば簡単です。以下の行をコメントアウトします。

```
<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

  .NET アプリケーションの一部をアンマネージ C/C++ で作成できます。DevPartner では、アプリケーションを一度実行すると、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 データを収集できます。ただし、アンマネージ コードが別のファイルに保存され、データを収集する前にコードをインストゥルメントすることが前提です。このように、パフォーマンス セッションを比較することによって、アプリケーション全般でアンマネージ コードとマネージ コードの効果を比較できます。

- 分散環境アプリケーションの詳細なデータを収集する

  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、分散マルチプロセス アプリケーションの各プロセスのパフォーマンス スナップショットを作成します。

  Web アプリケーションや多層のクライアント／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、すべてのリモート アプリケーション コンポーネントも分析します。DevPartner で、リモート システムでのパフォーマンス データ収集について設定します。アプリケーションでアンマネージ C/C++ コンポーネントを使用する場合は、データを収集する前にコンポーネントをパフォーマンス分析用に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を考慮するという推奨事項は、サーバー側コンポーネントのデータを収集する場合にも同様に適用されます。

- マイクロプロファイルの限度を調べる

  アプリケーションのボトルネックを特定したら、メイン アプリケーションで同じ問題を発生させるような簡単なコード サンプルを作成します。サンプルのパフォーマンスを繰り返し比較して、パフォーマンスを改善したあとに、メイン アプリケーションにコードを戻します。アプリケーションの実行速度は速くなりましたか。多分、速くなったでしょう。しかし、最初のパフォーマンス テストをもう一度実施してみるまではわかりません。

- 実際の実行条件のシミュレート

  アプリケーションのメモリ フットプリント、マルチスレッド、スレッドの優先度、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、その他不測の事態は、コードの実行方法に影響を及ぼす可能性があり、これは単一コンポーネントのパフォーマンス テストではわからないことがあります。アプリケーションが使用される条件に可能なかぎり近づけてシミュレーションを行うまで、アプリケーションのパフォーマンスを測定したことにはなりません。

Visual Studio Team System へのデータの送信

Microsoft Visual Studio Team Explorer クライアントがインストールされ、Team Foundation Server の接続を使用可能になっている場合に、DevPartner Studio は Microsoft Visual Studio Team System をサポートします。Team System のサポートに関する全般的な情報については、「Visual Studio Team System のサポート」（8 ページ）を参照してください。

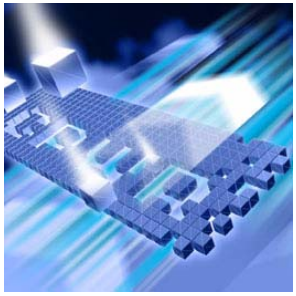
DevPartner パフォーマンス分析セッション ファイルの **[メソッド リスト]** タブで選択したメソッドのデータを、**作業項目**として Visual Studio Team System に送信できます。

バグを送信すると、**[メソッド リスト]** タブの表示カラムのデータが**作業項目**フォームにコピーされます。**作業項目**に送信するメソッド データを変更するには、**メソッド リスト**に表示するカラムを変更してください。

メモ： **この機能は、DevPartner パフォーマンス分析 Community Edition では使用できません。**

第 7 章

詳細なパフォーマンス分析



この章には 2 つのセクションがあります。1 つめのセクションには、はじめてのユーザーがパフォーマンス エキスパートを利用できるように、簡単な操作手順が記載されています。2 つめのセクションには、DevPartner Studio のパフォーマンス エキスパート機能を詳しく理解するための参考情報が記載されています。

パフォーマンス エキスパートに関するその他のタスクに基づく情報については、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 エキスパートの機能

DevPartner Studio は、アプリケーション開発を支援するように設計された数多くの機能を備えています。たとえば、コード内のボトルネックを特定するのに役立つパ

フォーマンス アナライザ（パフォーマンス エキスパート）もその1つです。パフォーマンス エキスパートは、以下のような解決困難な問題をより詳細に分析することによって、マネージVisual Studioアプリケーションの一歩進んだパフォーマンス分析を実行します。

- CPU／スレッドの使用状況
- ファイルまたはディスクのI/O
- ネットワークI/O
- 同期の待機時間

メモ： **パフォーマンス エキスパートで分析されるのはマネージ コードのみなので、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 エキスパートは、実行時にアプリケーションを分析し、コード内で問題のあるメソッドを特定します。ユーザーは、メソッド内の行に関する詳細を個別に表示したり、上位／下位の呼び出し関係を調べて問題解決に最良の方法を判断したりすることができます。解決へのアプローチを決定したら、パフォーマンス エキスパートを使用して、ソース コードの関連する行まで直接ジャンプできます。このように問題の修正を迅速に行うことが可能です。

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、アプリケーションを開発しながらアプリケーションのテストを行えます。また、パフォーマンス エキスパート セッションは、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実行することもできます。この場合は、DevPartnerのコマンド ライン実行可能ファイル**`DPAnalysis.exe`**を従来のコマンド ライン スイッチやXML構成ファイルと一緒に使用します。詳細については、「コマンド ラインからの分析の開始」（333ページ）を参照してください。

DevPartner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、品質保証（QA）エンジニア向けに設計されています。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ときにも使用できます。

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 エキスパートが必要な理由この機能は、従来のパフォーマンス プロファイルを補完する機能と考えてください。まず、パフォーマンス分析でアプリケーションを実行して、パフォーマンスのベースラインを把握します。次に、パフォーマンス エキスパートで同じセッションを実行し、特にディスクI/Oやネットワーク I/O、同期の問題が関係する問題など、困難な問題の本質をより深く理解します。問題を修正したら、パフォーマンス分析で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**機能を使用して状況が改善されていることを検証します。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、「セッションの比較」（236ページ）を参照してください。パフォーマンス分析と一緒にパフォーマンス エキスパートを使用する方法については、「パフォーマンス エキスパートとパフォーマンス分析の使用」（279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には

以下の準備、設定、実行手順では、DevPartner Studio のパフォーマンス エキスパート機能を使用する方法を紹介します。

機能をすぐに使用したい場合は、色付きの枠内に記載されている手順に従ってください。色付きの枠内に記載されているトピックの詳細な情報については、枠の下に記載されている文章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：分析内容の検討

どのような種類のアプリケーションを分析しますか。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、どのようなステップが必要か考えてください。

パフォーマンス エキスパートでは、マネージ アプリケーションのデータだけが収集されます。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（C#、Visual Basic、マネージ C++ など）が含まれる必要があります。また、スタートアップ プロジェクトも含まれる必要があります。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、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メッセージが表示されます。

この手順では以下を前提としています。

- Visual Studio 2003 または Visual Studio 2005 で作業しています。
- シングルプロセスのマネージ アプリケーションをテストしています。
- アプリケーションのビルドと実行が可能です。
- ソリューションに少なくとも1つのマネージ コード プロジェクトが含まれます。
- ソリューションにスタートアップ プロジェクトが含まれます。

メモ： **DevPartner メモリ分析でサポートされているプロジェクト タイプのリストについては、「DevPartner Studio でサポートされるプロジェクト タイプ」（321 ページ）を参照してください。**

Visual Studio からパフォーマンス エキスパートを実行した場合、または従来のコマンド ライン スイッチを使用して **DPAnalysis.exe** で実行した場合、単一のプロセスが監視されます。**DPAnalysis.exe** と XML 構成ファイルを使用することで、1 つのセッションに含まれる複数のプロセスまたはサービスからパフォーマンス 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、単一のプロセスをターゲットにするのが最適です。アプリケーションを複数のプロセスで実行している場合は、2 番めのプロセスをターゲットとしてアプリケーションを

再実行します。**DPAnalysis.exe** の使用方法の詳細については、「データ収集の自動化」（273 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すると、以下のようなマネージ Visual Studio アプリケーションのパフォーマンスを向上させることができます。

- 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 フォーム クライアント アプリケーション
- サービス コンポーネント（COM+ など）

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、収集するデータの内容を決定します。また、アプリケーションのパフォーマンスについて考えます。特定の機能を使用するときにアプリケーションの速度が遅くなりますか。その場合、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ときに、その機能を実行します。従来のパフォーマンス分析によると、データの読み取りまたは書き込みを行うメソッド、またはネットワーク リソースにアクセスするメソッドに通常より長い時間がかかっていましたか。パフォーマンス エキスパートを使用するとディスク I/O やネットワーク I/O に関する追加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ターゲットにします。

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 プロセスが含まれる場合、両方のプロセスのデータが必要ですか。その場合、まず DevPartner Studio と DevPartner リモート サーバー ライセンスをリモート マシンにインストールし、サーバー データを収集します。サーバー側のデータを収集する前に、場合によっては IIS の設定が一部必要になります。

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、データ収集を制限するために、いくつかのプロパティとオプションを設定します。

この手順では、デフォルトの DevPartner のプロパティとオプションを使用できます。その他の設定は必要ありません。

ソリューションのプロパティまたはプロジェクトのプロパティを使用すると、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、ソリューションに複数のプロジェクトが含まれる場合にセッションから特定のプロジェクトを除外したりすることができます。DevPartner のオプションを使用すると、表示オプションを変更したり、セッション コントロール ファイルを作成してデータ収集を管理したりすることができます。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の設定」（260 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 データの収集

分析する内容を検討し、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 データを収集する準備が整います。

DevPartner は Visual Studio の起動モデルをサポートしています。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し、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。

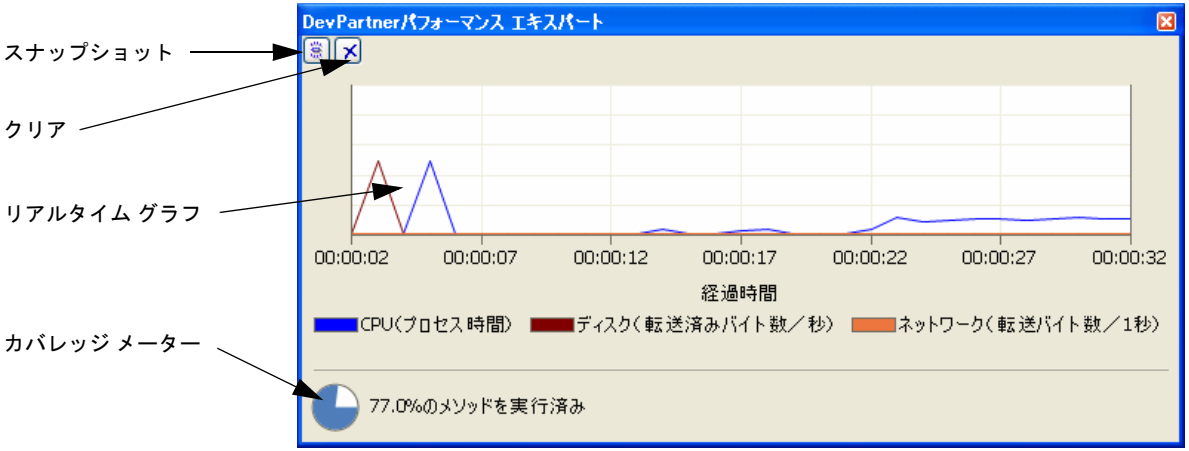


図 7-1. パフォーマンス エキスパート ウィンドウを使用したデータ収集の制御

1. **パフォーマンス エキスパート** ウィンドウの左上にある **[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をクリアして、問題に関連するデータの収集にフォーカスすることができます。
2. アプリケーションの低速な部分を実行します。
3. アプリケーションの実行中、**パフォーマンス エキスパート** ウィンドウを監視します。グラフでは、CPU プロセス時間が折れ線で表示されます。また、ディスクとネットワークの動作があれば、それらについても折れ線が表示されます。これらの折れ線のいずれかにスパイクがある場合、そこで問題が発生している可能性があります。
4. 問題が疑われるところが見つかったら、**[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 セッション ファイルが生成され、Visual Studio に表示されます。

パフォーマンス エキスパート ウィンドウの使用

リアルタイム グラフの使用

アプリケーションを実行すると、パフォーマンス エキスパートのリアルタイム グラフに最新の30秒間の動作が表示されます。また、CPU使用量を反映したグラフが常に描画されます。アプリケーションでディスクまたはネットワークの読み取りまたは書き込みを行っている場合、ディスクI/Oとネットワーク I/Oについても別の折れ線が表示されます。

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。たとえば、グラフの動作にスパイクがあるなど、問題が疑われるところが見つかったら、**[スナップショット]**ボタンを使用してその時点までの動作のスナップショットを作成できます。反対に、特に問題がない場合は、[クリア]ボタンを使用してその時点までに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、パフォーマンス エキスパート ウィンドウの左上、リアルタイム グラフの上にあります。これらのセッション コントロールを使用して、以下の動作を実行できます。

- **[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動作以降のデータをクリアします。[クリア]を使用すると、データの収集を制限して、セッション結果ファイルのサイズを最小限に抑えることができます。
- **[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動作以降のデータを含むセッション結果ファイルを作成します。データ収集は続行します。アプリケーションの実行中に複数のスナップショットを作成できます。

カバレッジ メーターの使用

カバレッジ メーターは、セッション コントロール ウィンドウの左下、リアルタイム グラフの下にあります。カバレッジ メーターには、セッションのその時点までに実行されたアプリケーション メソッドのパーセント値が表示されます。カバレッジ メーターを使用して、パフォーマンス エキスパートですべてのコードをテストしたことを確認できます。また、カバレッジ メーターとセッション コントロールの操作を組み合わせると、アプリケーションの特定部分のデータだけを収集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ます。デバッグを実行しないセッションの結果の方がわかりやすく、デバッガによる処理のオーバーヘッドも発生しません。アプリケーションをデバッガで実行すると、セッション中にブレークポイントに達した場合など、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。追跡機能などのデバック専用機能がこのようなセッションファイルに高い数値で示されることが予想されます。**

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、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のデータを入れたい場合は、**[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。そのままアプリケーションを終了してください。

データの分析

データのスナップショットを作成するか、データ収集を終了してアプリケーションを終了すると、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。アプリケーションについて収集されたデータは、最初結果サマリの形式で表示されます。

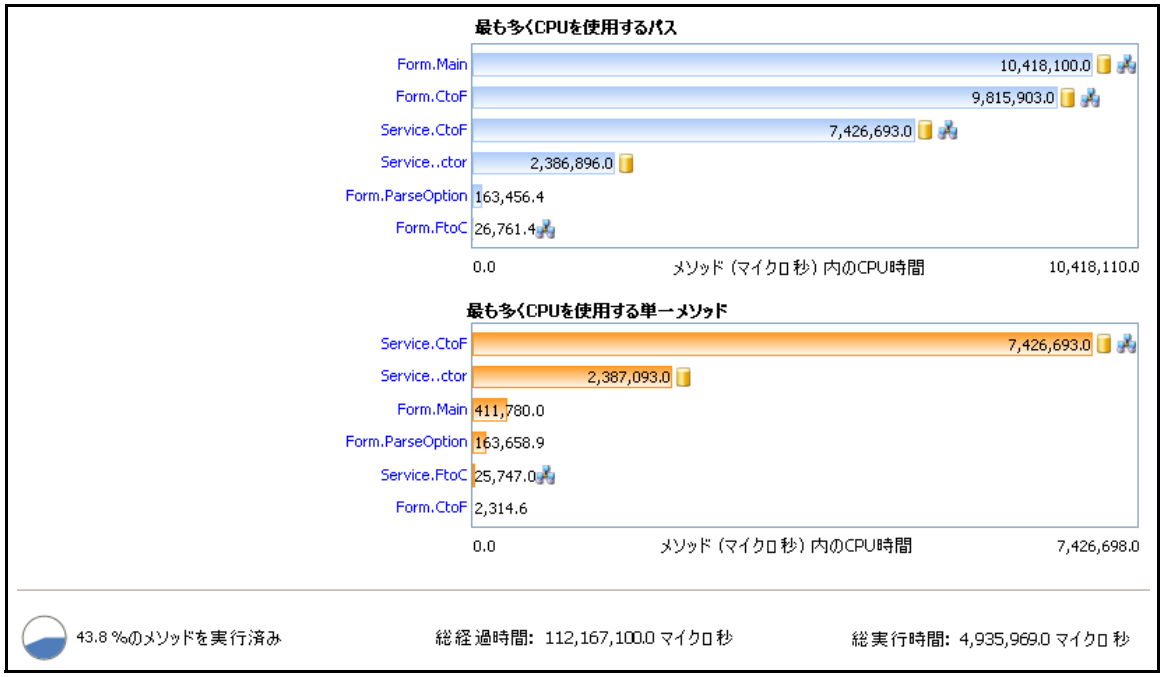


図 7-2. パフォーマンス エキスパートの結果サマリ

結果サマリには2つの棒グラフがあり、データを分析してアプリケーションの問題を解決できる2つの方法が反映されます。

ヒント: エントリ ポイント メソッドは、別のソース コード メソッド（つまりエントリ ポイント）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 コード実行へのエントリ ポイントです。

- **[最も多くCPUを使用するパス]**には、セッションで最も多くCPUサイクルを使用した上位パス、またはメソッド コール チェーンのエントリ ポイント メソッドが表示されます。パス分析によって、最もコストが高いメソッドの実行パスをすばやく特定できます。それによって、以下のことが可能です。
  - ◇ 低いパフォーマンスの原因になっている下位メソッドを修正する
  - ◇ コストが高い下位メソッドの呼び出し回数が少なくなるように、コール シーケンス内の他のメソッドを修正する
- **[最も多くCPUを使用する単一メソッド]**には、消費したCPUサイクルに関して上位のメソッドが表示されます。メソッド分析によって、問題となる各メソッドをすばやく特定して、修正できます。

棒グラフの端にはアイコンが表示されます（図 7-2（249ページ））。これらのアイコンは、メソッドによってディスク  またはネットワーク  の動作が行われたことを示します。

開始する場所の決定

セッション データの評価を開始するには、結果サマリの2つの棒グラフを比較します。

- **[最も多く CPU を使用するパス]**グラフの最上位パスは、グラフの他のパスと比べてかなり長いですか。
- **[最も多く CPU を使用する単一メソッド]**グラフの最上位メソッドは、グラフの他のメソッドと比べて際立っていますか。
- メソッドの時間値は、メソッドが実行する内容にしては長いと思われるでしょうか。
- 同じメソッドがどちらのグラフでも同じくらいのコストに見えるでしょうか。

以上の質問に対する答えが「イエス」の場合、そのメソッドを調べます。

データを分析する前に、結果サマリからアクセスできるデータ ビューのナビゲート方法を確認します。

1. 結果サマリの**[最も多く CPU を使用するパス]**グラフの最上位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、上部に**[コール グラフ]**タブと**[コール ツリー]**タブ、下部に**[ソース]**タブと**[コール スタック]**タブがあります。

2. **[サマリに戻る]**をクリックすると、結果サマリに戻ります。
3. 結果サマリの**[最も多く CPU を使用する単一メソッド]**グラフの最上位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、上部にセッションで実行されたメソッドのリスト、下部に**[ソース]**タブと**[コール スタック]**タブがあります。

4. **[サマリに戻る]**をクリックすると、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 データの**コール グラフ**と**コール ツリー**を表示できます。**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が、そのパスに費やされた時間に対する割合と共に表示されます。**コール ツリー**は同じデータのツリー ビューですが、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関する追加データも表示されます。

[最も多く CPU を使用する単一メソッド]グラフのメソッドを調べることにした場合、ユーザーが構成可能なデータ カラムがある**メソッド** テーブルが表示され、トラブルシューティングに使用できます。**パス分析**と**メソッド** テーブルの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]**ビューとで異なります。**[最も多く CPU を使用する単一メソッド]**ビューでは、CPU 時間、ディスク I/O、ネットワーク I/O、同期ロックの待機時間に関するデータを計算する際に、ソース コード

の下位メソッドの測定が除外されます。ソース コードの下位メソッドを除外することにより、最も長時間の CPU 時間を消費するメソッドに注意を向けられます。反対に、**[最も多く CPU を使用するパス]** ビューでは、最もコストが高い実行パスを特定するために、ソース コードの下位メソッドの影響が、その上位メソッドの計算に含まれます。

いずれのビューでも、ユーザーのソース コード メソッドによって呼び出されたシステムまたは .NET Framework のメソッドに起因する時間やスループットは計算に含まれます。マネージ アプリケーションでは、一般的に、.NET Framework コードの実行にかなりの時間が費やされます。パフォーマンス エキスパートは、呼び出しを行うソース コードの行にシステム データを割り当てて、コードと .NET Framework との対話、つまりアプリケーションでユーザーが変更可能な部分を調べられるようにします。

この手順では、まず**パス分析**を使用して、最もコストが高い実行パスで呼び出された下位メソッドによる相対的な影響を分析します。

5 結果サマリの **[最も多く CPU を使用するパス]** グラフ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、左の **[コール グラフ]** タブをクリックします。

重要な、つまり最もコストが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を始めます。

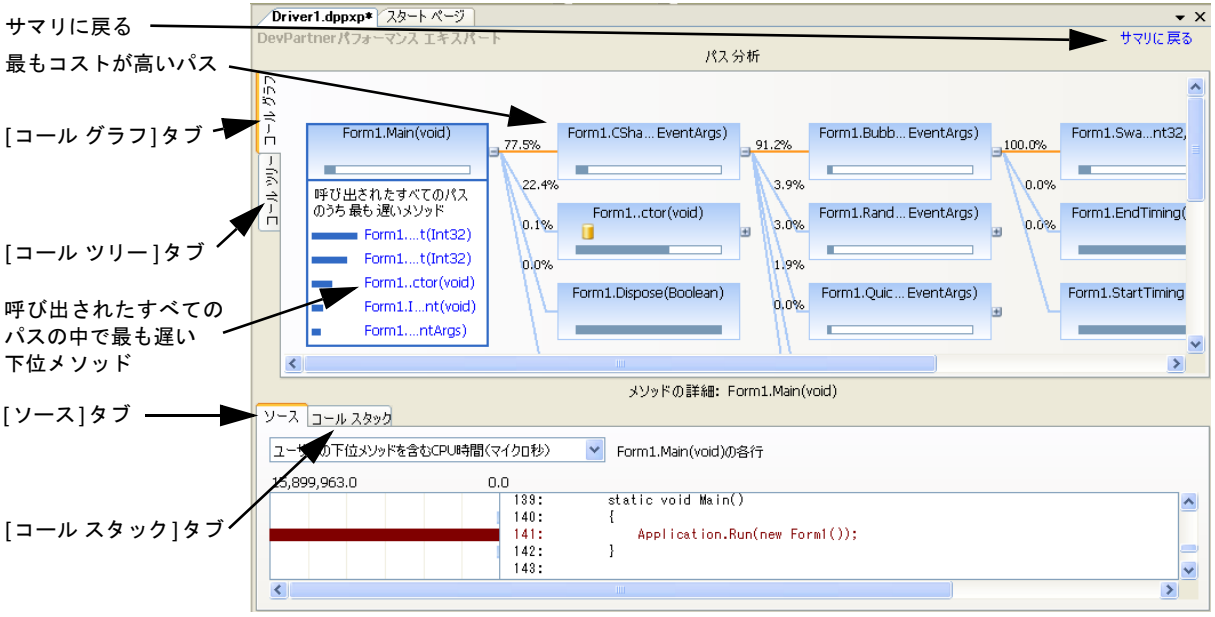


図 7-3. [パス分析]ウィンドウでの最もコストが高い実行パスの特定

コール グラフで、以下を実行します。

6 パスを調べるために、ノードのプラス記号をクリックして、右側のパスを展開します。

7 任意のメソッドをクリックして、パスに関係なく、そのメソッドが呼び出した中で最も遅い下位メソッドのリストを表示します。このリストによって、クリティカル パスに含まれていないものも含め、遅いメソッドがわかります。

8 同じメソッドから派生するそれぞれのパスの相対的な影響を判断するために、選択したメソッドと各下位パスをつなぐライン上のパーセント値を比較します。最もコストが高い（最も高いパーセント値の）パスから調べます。

9 各ノードの下部にある水平バーにマウス ポインタを移動して、そのメソッドに使用された時間に対する下位メソッドの実行に使用された時間のパーセント値を表示します。たとえば、図 7-4（252 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に使用されている場合、そのパスの調査を続行します。ほとんどの時間があるメソッドに使用されていた場合、そのメソッドに焦点を当てます。

コール グラフを見ると、コール シーケンスの中でコストが高いメソッドを迅速に特定できるため、パフォーマンス調整作業の焦点を絞ることができます。下位メソッドの影響の他に、**コール グラフ**のノードによってメソッドの実行内容がわかります。コール グラフの使用に関する詳細については、「コール グラフ」（269 ページ）を参照してください。

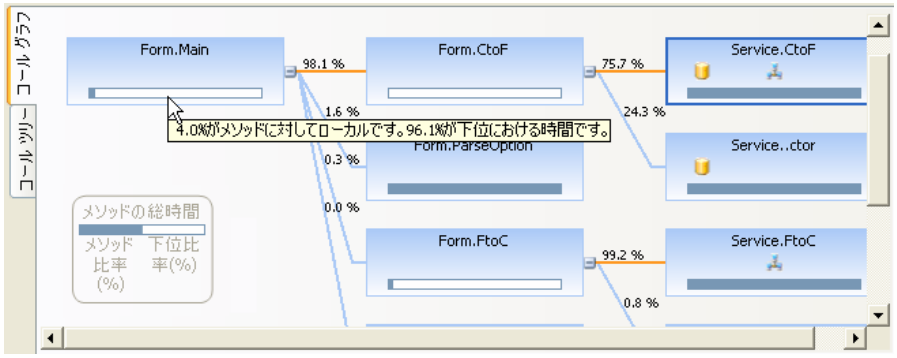


図 7-4. 下位メソッドの影響の評価

10 調べようとしているノードには、以下のアイコンの 1 つまたは複数が含まれていますか。

 ディスク動作を示します。

 ネットワーク動作を示します。

 同期の待機時間を示します。

11 含まれる場合、アイコンの上にマウス ポインタを移動して、動作の重要性を表示します。動作の重要性が高く、詳細に調べる利点がある場合、**[コール ツリー]** タブに切り替えて、詳細な診断ヘルプを参照します。

12 **コール ツリー**を表示するには、セッション ファイル ウィンドウの左側にある **[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 コードまたは .NET Framework メソッドとは対照的な、ユーザーのアプリケーション ソース コードのメソッドです。

コール ツリーには、**コール グラフ**と同様の情報がツリー ビューの形式で表示されます。最もコストが高いパスは、テーブルのソート順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**には、上位メソッドに対する下位メソッドの相対的な影響に関する情報が表示されます。対照的に、**コール ツリー**には、アプリケーションのメソッドが実際に実行している内容に関する詳細データが表示されます。このデータは、ソート／設定可能なデータ カラムの形式で表示されます。**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、コンテキスト メニューから **[項目の選択 ...]** を選択します。データ カラムを追加する前に、Visual Studio の [プロパティ] ウィンドウでデータ 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。

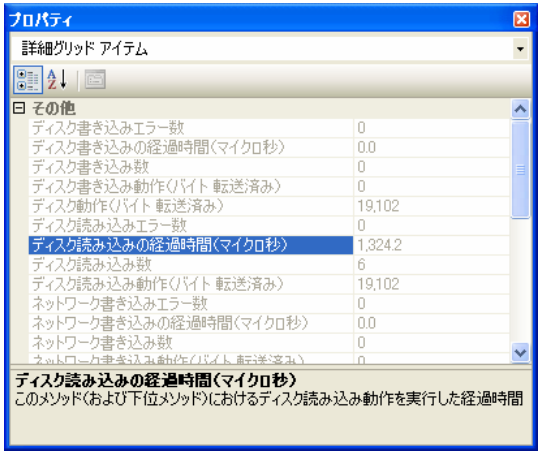


図 7-5. [プロパティ] ウィンドウでのメソッド データの表示

コール ツリーで、以下を実行します。

13 同じメソッドから派生するそれぞれのパスの相対的な影響を判断するために、各下位パスの**[ユーザーの下位メソッドを含む CPU 時間]**カラムの値を比較します。このカラムでソートすると（デフォルトのソート順）、最もコストが高いパスがツリー ビューの上位に表示されます。

14 **コール ツリー**を**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**のコストが高いノードにネットワーク I/O アイコンが含まれる場合、**コール ツリー**に切り替えて、ネットワーク関連データのカラムをビューに追加します。

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、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、ネットワークの読み取り数と書き込み数、ネットワーク内でデータの読み取りと書き込みに使用された時間、データの読み取りと書き込みのサイズ、読み取りエラーと書き込みエラーの数が表示されます。

コール グラフ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る場合、そのデータ カラムを**コール ツリー** ビューに追加します。このように操作すると、問題のノードのコストが高い理由を迅速に特定できます。

[コール グラフ]ウィンドウと**[コール ツリー]**ウィンドウのどちらにも、ウィンドウの下部に**[ソース]**タブと**[コール スタック]**タブがあります。

[ソース]タブを使用すると、アプリケーションのメソッドのソース コードと、セッション中に実行された行のコストを示すメトリクスを確認できます。これにより、ソース コード内でコストが高い行を確認し、改善すべき行を迅速に特定できます。**[ソース]**タブには図 7-6（255 ページ）に示すようにメトリクス リストがあります。**[パス分析]**ビュー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。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は、メソッドの実行内容に応じて使用可能になります。このリストで新しいメトリクスを選択すると、ソース ペインが更新され、そのメトリクスに関して、ソース コードで最もコストが高い行を特定できます。

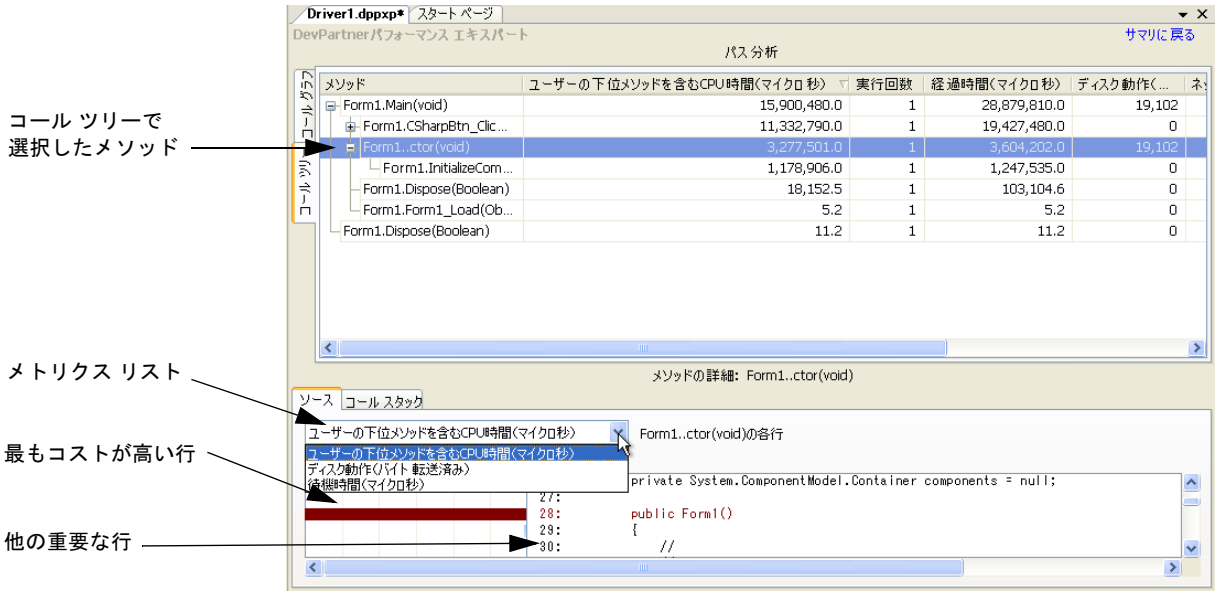


図 7-6. [ソース]タブでの最もコストが高い行の特定

[ソース]タブを**コール グラフ**、**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]**タブで問題がありそうなメソッドを選択します（**[コール グラフ]**タブに戻った場合は、メソッド ノードを選択できます）。**[ソース]**タブを選択します。（**[ユーザーの下位メソッドを含む CPU 時間]**で測定された）最もコストが高い行が茶色で強調表示されます。**[ソース]**ペインをスクロールすると、他のコストの高い行が青色で強調表示されているのがわかります。

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はありましたか。**コール ツリー**でそのようなメソッドをすばやく特定するには、該当するデータ カラムの値が高いメソッドを探します（または、**コール グラフ**で、メソッド ノードのディスク 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探します）。

17 **[ソース]**タブの左上にあるメトリクス リスト（図 7-6（255 ページ）を参照）を展開します。選択したメソッドにディスク I/O、ネットワーク I/O、または待機時間が含まれていた場合、そのメトリクスがリストに表示されます。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、そのメトリクスに関して最もコストが高い行を特定します。コストが高いメソッドには、改善できる点が複数存在することがあります。

18 **[ソース]**タブ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 でソース ファイルを開き、編集します。

[コール スタック]タブを使用すると、アプリケーションのコストが高いメソッドのさまざまなインスタンスや使用方法を参照できます。各コール スタックは固有です。場合によっては、同じメソッド コール シーケンスを含むコール スタックがあるかもしれません。ただし、注意深く見ると、少なくとも1つのメソッドで、別の行から呼び出しが行われているはずです。

コール グラフまたは**コール ツリー**で別のメソッドを選択すると、**[ソース]**タブがスクロールし、各メソッドで最もコストが高い行が表示されます。同様に、別の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（255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。その下位メソッドを修正することでパフォーマンスを改善しようとする場合、**[ソース]**タブを表示します。**[ソース]**タブでは、そのメソッドで最もコストが高い行が強調表示されます。そのメソッドでディスク I/O またはネットワーク I/O を実行した場合、または長い待機時間があった場合、メトリクス リストを使用して選択したメトリクスについて最もコストが高い行を特定します。修正対象を決定したら、そのソース行をダブルクリックし、Visual Studio で編集します。

一方、アプリケーションから下位メソッドを呼び出す方法を変更することでパフォーマンスを改善しようとする場合は、**[コール スタック]**タブに切り替えます。

[コール スタック]タブを**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]**タブには選択し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、アプリケーションでそのメソッドが使用されているすべての方法について、メソッドに対する変更を評価できます。**[コール スタック]**タブを使用して、メソッドの最もコストが高いインスタンス（使用方法）を迅速に特定します。

19 **[コール ツリー]**タブで問題がありそうなメソッドを選択します（**[コール グラフ]**タブに戻った場合は、そこでメソッドを選択できます）。**[コール スタック]**タブを選択します。選択したメソッドを呼び出した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 リストを展開します。スタック リストを使用して、メソッドの最もコストが高い使用方法を特定します。

21 **[コール スタック]**タブ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 でソース ファイルを開き、編集します。

22 コストが高いメソッドを直接修正できない場合、そのメソッドを呼び出す回数を減らしたり、まった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を呼び出したすべてのコール シーケンスまたはパスを確認できます。図 7-7（257ページ）では、スタック リストに各コール スタックに起因する時間のパーセント値が表示されています。これにより、最もコストが高い実行パスを迅速に特定できます。コール スタックを選択すると、そのスタックを構成するすべてのメソッドと、スタックで次のメソッドを呼び出した各メソッドの行番号が表示されます。コール スタックでメソッドを選択すると、ソース ペインが更新され、次の下位メソッドを呼び出し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、Visual Studio で編集します。

呼び出し方法を変えるのではなく、遅い下位メソッドを修正する場合でも、そのメソッドについて**コール スタック**を確認してください。メソッドを変更する前に、アプリケーションでそのメソッドがどのように使用されているかをすべて把握することは重要なことです。パフォーマンス エキスパートを使用すると、これを簡単に行えます。

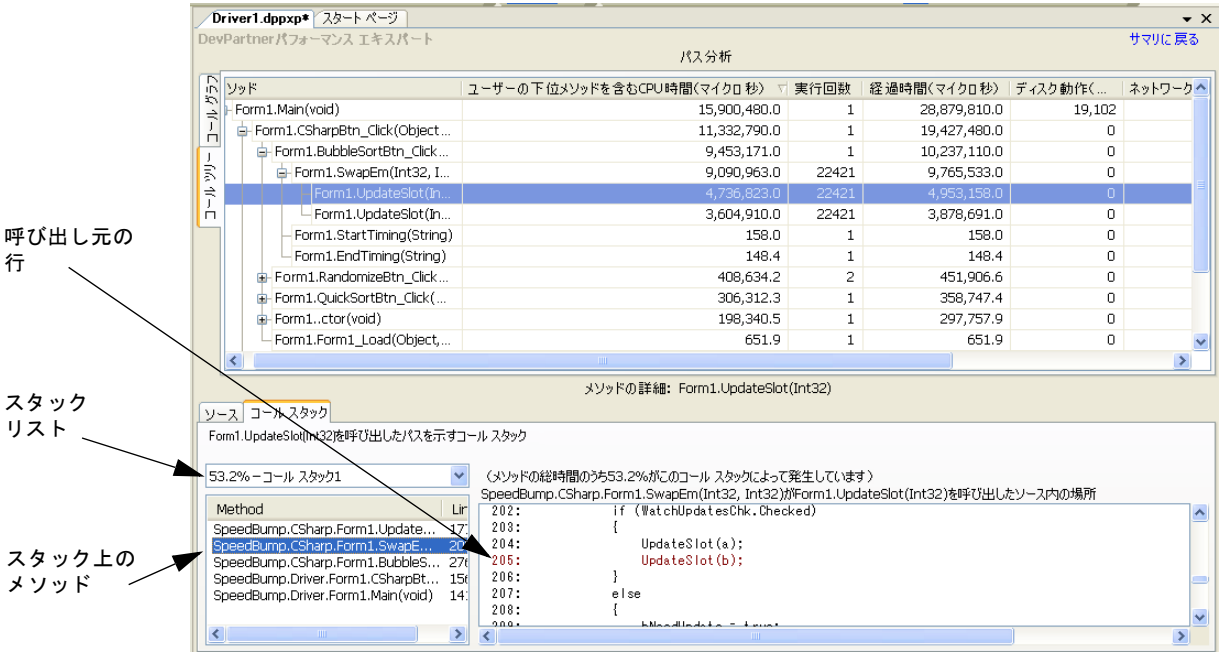


図 7-7. 特定のメソッドを使用した最もコストが高いコール パスの特定

最も多く CPU を使用するメソッドの分析

ここまでは、結果サマリの**[最も多く CPU を使用するパス]**棒グラフからセッション データにドリル ダウンすることに重点を置きました。パフォーマンス エキスパート データの分析は、**[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行うこともできます。たとえば、このグラフの上位メソッドに予想よりも多くの時間がかかっていることに気づいた場合、グラフのメソッドをクリックして、そのメソッドをすぐに調べることができます。

メソッドをクリックすると、**メソッド**テーブルが開きます。このテーブルには、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表示されます。

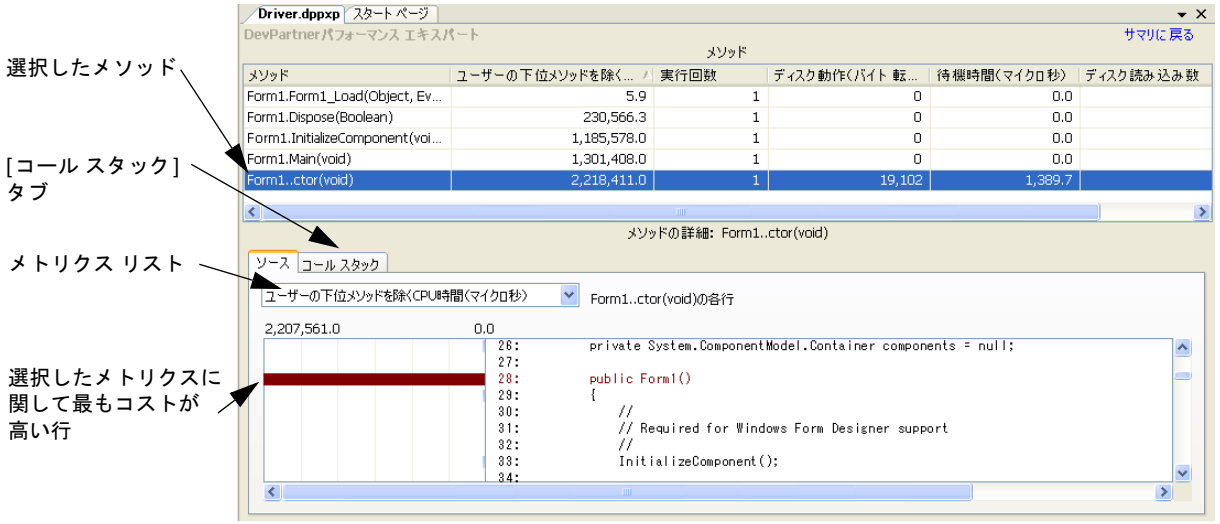


図 7-8. 各メソッドの影響の分析

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ために、**[サマリに戻る]** をクリックします。

24 結果サマリの **[最も多く CPU を使用する単一メソッド]** グラフ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。

アプリケーションで最もコストが高いメソッドが強調表示されます。デフォルトでは、メソッドに使用された CPU 時間によってメソッドがソートされます。このとき、ユーザーの下位メソッドは含まれませんが、システム コールは含まれます。

25 **メソッド** テーブルのカラム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 **[項目の選択 ...]** 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に関して最もコストが高い面を判断します。

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。このメトリクスを選択すると、そのメソッド自体のパフォーマンスに焦点を絞れます。対照的に、**[最も多く CPU を使用するパス]** ビューでは、ユーザー コード、ソース コード、下位メソッドが計算に含まれます。

[ソース]タブを**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]**タブにはそのメソッドの最もコストが高い行が直接表示され、他の行の相対的なコストも表示されます。最もコストが高い行は茶色で表示されます。メソッドの時間消費に影響があった他の行は、淡い青色で表示されます。

26 **[ソース]**タブのメトリクス リストを使用して、使用できる各メトリクスで最もコストが高い行を特定します。問題のメソッドには、改善できる点が複数存在することがあります。

[コール スタック]タブを**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]**タブには選択し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、アプリケーションでそのメソッドが使用されているすべての方法について、メソッドに対する変更を評価できます。**[コール スタック]**タブを使用して、メソッドの最もコストが高いインスタンス（使用方法）を迅速に特定します。

27 **[ソース]**タブまたは**[コール スタック]**タブで修正する行を特定します。遅い行をダブルクリックして、Visual Studio でソース ファイルを開き、編集します。

28 コストが高いメソッドを直接修正できない場合、そのメソッドを呼び出す回数を減らしたり、まった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]と**メソッド**テーブルの時間計算のパーセント値では、ソース コードの下位メソッドに使用された時間は除外されますが、**システム**の下位メソッドに使用された時間は含まれます。多くの場合、マネージ アプリケーションは .NET Framework でかなりの時間を費やします。計算にシステムの下位メソッドを含めることにより、システム コードとの対話の仕方に問題が発生しているユーザーのソース コードのメソッドに焦点を絞ることができます。これはマネージ アプリケーションの場合、特に重要です。

セッション ファイルの保存

パフォーマンス エキスパート データのレビューが終わったら、セッション ファイルを保存するか破棄することができます。

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]>[<filename>.dppxp の保存]**を選択します。

2 セッションを保存せずに Visual Studio でセッション ファイル ウィンドウを閉じた場合、開いているセッション ファイルの保存を確認するダイアログ ボックスが表示されます。

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。保存されたファイルは、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 セッション ファイルの拡張子は **.dppxp** です。

デフォルトで、セッション ファイルはプロジェクトの出力フォルダに物理的に保存されます。また、デフォルト ディレクトリの内容に基づいて、ファイル名の番号が自動的に1つ増えます（たとえば、**`MyApp.dppxp`**、**`MyApp1.dppxp`** など）。セッション ファイルをデフォルト ディレクトリとは別の場所に保存する場合は、自分でファイルのネーミングを管理する必要があります。

Visual Studio 2005 の Web サイト プロジェクトなど、出力ディレクトリがないプロジェクトの場合、ファイルはプロジェクト ディレクトリに物理的に保存されます。

Visual Studio 以外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 で開いているソリューションに手動で追加できます。

この章の準備、設定、実行セクションはこれで終了です。ここまでで、パフォーマンス分析セッションの実行方法について基本的な知識が習得できたはずです。詳細情報については、この章の残りを続けて読んでください。また、タスクに基づく情報については 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の設定

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、特定の種類の情報を含めたり、除外したりするために、データ収集を調整すると便利です。ソリューションのプロパティ、プロジェクトのプロパティ、DevPartner のオプションを使用すると、分析セッションのフォーカスを調整できます。

ソリューションのプロパティ

ソリューション レベルでパフォーマンス エキスパートに影響がある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.　ソリューションのプロパティ

以下のソリューション プロパティはパフォーマンス 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**－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、**[False]** に設定されていた場合、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 アプリケーションの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る場合、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** プロパティは、そのセッション内でアクティブなすべてのプロジェクトについてのデータ収集を制御します。

ソリューションにスタートアップ プロジェクトが含まれる必要があります。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、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メッセージが表示されます。

ソリューション プロパティの **[共通のプロパティ]>[スタートアップ プロジェクト]** ページに表示される **[アクション]** が **[開始]** に設定されているプロジェクトのみが、プロンプト ダイアログに含まれます。目的のスタートアップ プロジェクトがプロンプトに表示されない場合、ソリューション プロパティ ページを開き、プロジェクトの **[アクション]** を **[開始]** に設定します。以降のセッションで新しいスタートアップ プロジェクトを選択する場合、新しいスタートアップ プロジェクトのプロパティについて、データ収集のオプションが正しいことを確認します。

プロジェクトのプロパティ

プロジェクト レベルのプロパティを確認するには、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、ソリューション内のプロジェクトについて設定できるプロパティを確認します。

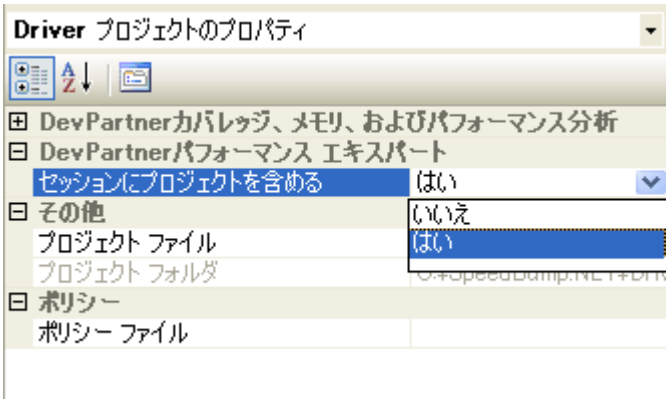


図 7-10. パフォーマンス エキスパートのプロジェクト プロパティ

以下のプロジェクトレベル プロパティはパフォーマンス エキスパートに影響を与えます。

- **セッションにプロジェクトを含める**ーパフォーマンス エキスパートの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、データの表示に使用される精度、位取り、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、ルールとアクションのセットを作成し、アプリケーションまたはモジュールの実行時に DevPartner で収集するデータを制御できます。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 内でのセッション コントロール ファイルの作成」（354 ページ）を参照してください。

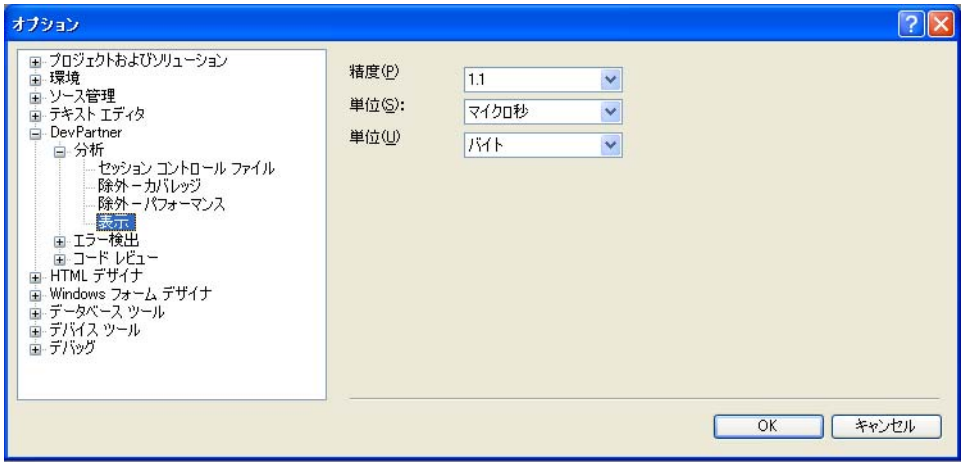


図 7-11.　分析オプション

[環境]>[フォントと色]オプションなど、他のVisual Studioオプションも DevPartner機能に影響があります。

パフォーマンス エキスパートを使用したアプリケーション問題の検出

DevPartnerパフォーマンス エキスパートは、以下のような重要な領域で発生したマネージVisual Studioアプリケーションの問題特定を支援します。

- CPU／スレッドの使用（待機や同期に関する問題を含む）
- ファイルI/Oやディスク I/O
- ネットワーク I/O
- 同期の待機時間

パフォーマンス エキスパートは、Visual Studioから実行した場合一度に1つのプロセスを分析します。選択したプロセスで実行されているすべてのマネージ スレッドに関するデータをレポートします。追加のプロセスを分析するには、2つめのプロセスを選択し、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、複数のマシンで実行され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からのデータの収集」（275ページ）を参照してください。

DevPartnerはVisual Studioの起動モデルをサポートしています。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 プロジェクトが直ちに起動し、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（C#、Visual Basic、マネージ C++ など）が含まれる必要があります。また、スタートアップ プロジェクトも含まれる必要があります。詳細については、「プロパティとオプションの設定」（260 ページ）を参照してください。

セキュリティ エラーが表示された場合

マネージ アプリケーションのデータを収集しようとしたときに、セキュリティ例外メッセージが表示された場合、DevPartner によるコードのインストゥルメントがセキュリティ ポリシーによって拒否されたことを示します。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。コードの実行に使用しているポリシーの権限セットからこの権限を削除した場合、またはこの権限を取り消すような厳しいセキュリティ宣言をアセンブリに追加した場合、アセンブリはプロファイルできません。

この状況を修正するには、以下のいずれかの方法で、安全なプロファイルを有効にします。

- 以下のグローバル環境変数を設定し、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
  ```
  この方法で問題を回避できますが、パフォーマンスがやや低下します。
- .NET Framework 構成ツールの MMC スナップインを使用するか、アセンブリに含まれる厳しいセキュリティ宣言を一時的に削除することで、アセンブリのポリシーを変更します。

Visual Studio のセキュリティ ポリシーの詳細については、Visual Studio のオンライン ヘルプに含まれる「.NET Framework 開発者ガイド」を参照してください。

下位メソッドについての計算

パフォーマンス エキスパート セッション データの計算は、**[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]** ビューとで異なります。単一メソッドの分析ビューでは、CPU 時間、ディスク I/O、ネットワーク I/O、同期ロックの待機時間などのデータの計算で、ソース コードの下位メソッドの測定値は除外されます。対照的に、パスの分析ビューでは、ソース コードの下位メソッドの影響がその上位メソッドの計算に含まれます。

いずれのビューでも、ユーザーのソース コード メソッドによって呼び出されたシステムまたは .NET Framework のメソッドに起因する時間やスループットは計算に含まれます。マネージ アプリケーションでは、一般的に、Framework コードの実行に長い時間が費やされます。パフォーマンス エキスパートは、呼び出しを行うソース

コードの行にシステム データを割り当てて、コードと Framework との対話、つまりアプリケーションでユーザーが変更可能な部分を調べられるようにします。

セッション データの収集と分析に関するヒントについては、後述の「使用シナリオ」を参照してください。

使用シナリオ

パフォーマンス問題を解決するための一般的な手順は、以下のステップで構成されます。

1. 問題のメソッドで最も処理の遅い行を特定し、最適化する。
2. 該当する行を最適化できない場合は、その行を削除するか、実行する頻度を少なくする。

最も単純なケースでは、DevPartner パフォーマンス分析機能などを使用して、メソッドで最も処理の遅い行を特定し、最適化するか、実行する頻度を少なくします。しかし、現実のアプリケーション開発では、多くの問題の原因はより複雑です。最も処理の遅いメソッドを特定できたとしても、そのメソッド内のいくつかの行の組み合わせで実行速度が遅くなっていることが判明するだけかもしれません。そのようなケースでは、対象データを追加することで、問題を迅速に分析できるかもしれません。

たとえば、アプリケーションで最も処理の遅い部分が大量のネットワーク I/O を行っている場合、以下のようなメトリクスを使用すると、問題の本質を理解するのに役立ちます。

- ネットワークの読み取り／書き込みの合計数
- 読み取り／書き込みのバイト数
- 読み取り／書き込みのエラー数
- 読み取り／書き込み処理に要した経過時間

アプリケーションが大量のディスク I/O を行った場合、読み取り／書き込みの量を反映するメトリクスや、それらの処理の効率性を調べる必要があるかもしれません。DevPartner パフォーマンス エキスパートは、このようなデータを正確にレポートします。

DevPartner パフォーマンス エキスパートを使用して、CPU パフォーマンス、スレッド パフォーマンス、ディスク I/O、ネットワーク I/O、同期の待機時間に関するアプリケーション パフォーマンスを分析することができます。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 パフォーマンスを向上させるためのいくつかの方法について、以下に例を挙げて説明します。

特定可能なパフォーマンス問題

シナリオ : ユーザビリティのテスト担当者から、アプリケーションの特定の処理が極端に遅いという報告を受けました。開発者として、処理速度が遅くなる原因となっているソース コード部分を特定し、修正したいと考えています。

「実行：パフォーマンス エキスパート データの収集」（247ページ）で説明したように、パフォーマンス エキスパートを使用して、アプリケーションの最も処理の遅い部分を実行したとします。セッション ファイルを調べると、**[最も多く CPU を使用する単一メソッド]**グラフの上部に、実行時間の最も長いメソッドが表示されています。ただし、複雑なアプリケーションでは、処理の遅い1つのメソッドがパフォーマンスに及ぼす影響は、処理がやや遅いメソッドが連続して実行される場合と比較して、その度合いが小さいことがあります。処理が最も遅いコール 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るメソッドはありますか。ある場合、それらは明らかに調査対象とすべきメソッドです。

また、グラフのいくつかのメソッドには、メソッド内のディスク I/O 動作やネットワーク I/O 動作を示すアイコンが付いています。これらのアイコンは、そのメソッドによって実行された処理の種類を示します。


ディスク動作


ネットワーク動作

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。経過時間と比較して実行時間が非常に短く、その時間差は単にユーザーからの入力を待機することに起因するものではないことが確実である場合は、アプリケーションの一部のメソッドのロック待機時間が必要以上に長くなっていないかどうかを確認します。

まず最初に、**[最も多く CPU を使用する単一メソッド]**グラフの最上位メソッドを調べることにしたとします。CPU 使用量には、プロセッサ使用量の多い計算、ディスク I/O、ネットワーク I/O、非効率に使用されている同期オブジェクトなど、数多くの要因が影響を及ぼす可能性があります。同様に、待機時間が生じる原因もさまざまです。たとえば、メソッドが待機しているリソースが、同じプロセス内で共有されていたり、外部のプロセスと共有されている場合などがあります。しかし、アプリケーションで起こっていることを迅速に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]グラフで、最上位メソッドをクリックし、そのメソッドの詳細ビューを開きます。**メソッド**テーブルのカラムのデータに注目します。この情報は、メソッドが実行している内容を判断するのに役立つものでなければなりません。グラフ内で、そのメソッドにディスク動作を示すアイコンが付いていた場合、テーブルを右クリックし、**[項目の選択 ...]**ダイアログを使用して、ディスクに関連するすべてのカラムをテーブルに追加します。そのメソッドで読み取り／書き込みエラーが発生していたり、小さなデータを書き込むのに長い時間を使用していたり、何回も実行されていることがわかるかもしれません。

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メソッドのソース コードが表示されます。テーブルでメソッドをクリックすると、ソースが自動的にスクロールされて、最も長く CPU 時間を使用した行が表示され、その行に

起因する時間が示されます。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。

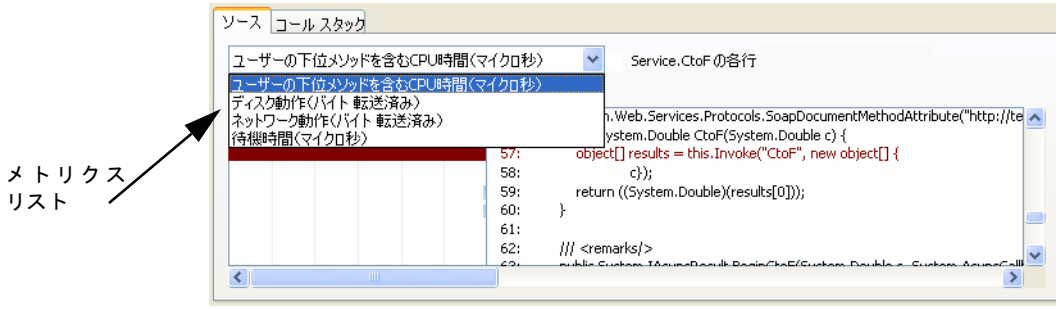


図 7-12.　[ソース] タブでの問題の行の特定

メソッドがディスク I/O やネットワーク I/O を実行している場合、またはメソッドに待機時間がある場合、ウィンドウの左上にあるメトリクス リストを展開してそれらのメトリクスを選択すると、そのメトリクスについてメソッド内で最も重要な行を直ちに特定できます。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、最も多いバイト数を転送した行に即座に移動して、そのメソッド内での他の行について関連するディスク動作があるかどうかを調べることができます。メソッドに**待機時間**がある場合は、そのビューも調べます。長い待機時間に関連のある行に注目します。デフォルトでは、各ビュー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と、従来のデバッグ方法を使用するよりもはるかに迅速に、取り組むべき箇所を見つけることができます。

修正すべき行を特定したら、その行をダブルクリックします。すると、Visual Studio でソース コードの該当する行にジャンプします。

問題を修正する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時におけるそのメソッドのすべての使用方法を表示します。問題となるメソッドは、複数のパスから呼び出されているでしょうか。その場合は、メソッド内で最も長い時間を使用しているコール スタックを調べます。

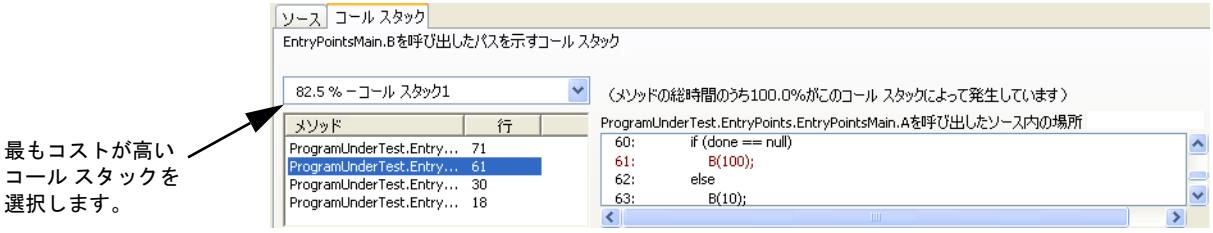


図 7-13.　最もコストが高いコール スタックの検索

ヒント: コール スタック内のいずれかのメソッド（または同じメソッド内の呼び出し行）が異なる場合、パフォーマンス エキスパートは、その上位分岐を固有のものとして記録します。

当然のことながら、最初に、最も高いパーセント値を持つコールを含む上位パスを調べます。コードを変更して、そのコールをなくすか、呼び出す頻度を少なくするようにします。**[コール スタック]** タブには、ソース コードのビューも含まれます。スタック内のメソッドを選択すると、ソースが自動的にスクロールされ、スタック内で次のメソッドを呼び出した行が表示されます。ダブルクリックすると Visual Studio でその行が表示されるので、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。コードに変更を加えたら、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再実行し、改善を確認します。

アプリケーションのスケーラビリティ問題

シナリオ : 新しい Web アプリケーションを自分のマシンでテストしたときは、申し分なく動作しました。ところが、他のユーザーがアプリケーションにアクセスできるようにすると、速度が極端に遅くなりました。納期は迫っています。何がいけないのかをすばやく判断するには、どう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ながら、パフォーマンス エキスパートでデータを収集します。このためには、コマンド ライン ツールかスクリプトを使用して、アプリケーションを開始し、終了する必要があるでしょう。DevPartner には、この用途のために **`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 ラインからパフォーマンス エキスパート セッションを実行する詳細については、「データ収集の自動化」（273 ページ）を参照してください。たとえば、以下のような処理を実行できます。

1. **`DPAnalysis.exe`** を使用して、パフォーマンス エキスパートでアプリケーションを開始します。
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終了します。
4. パフォーマンス エキスパート セッション データを調べます。

セッション ファイルを調べたところ、**[最も多く CPU を使用する単一メソッド]** グラフには問題の原因となるような単一のメソッドがなかったとします。これは複雑なアプリケーションで、いくつかのメソッドが組み合わさってパフォーマンスの低下を招いていると考えられます。結果サマリの **[最も多く CPU を使用するパス]** グラフを使用して、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、このケースでは、各メソッドは**エントリ ポイント**を示しています。エントリ ポイント メソッドとは、他のソース コード メソッドから呼び出されなかったメソッドです。つまり、ユーザーが作成したコードの実行へのエントリ ポイントです。最も重要な点は、メソッドやメソッドの呼び出し方法を変更することで、変更できる実行パスの開始点を示しているということです。アプリケーション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、グラフの上部に表示されます。そのメソッドをクリックし、**[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、最もコストの高いパスが**コール グラフ**の上部に表示されます。パスに分岐があるときは、最もコストの高い下位パスが強調表示されます。データを調べるときには、最初に、最もコストの高い下位パスを調べます。パスを調べるには、右側のノードを展開します。

ヒント: メソッドと、そのメソッドが呼び出す下位メソッドをつなぐライン上に表示されるパーセントは、加算できる値ですが、単一のパス上のメソッドのチェーンをつなぐライン上に表示されるパーセントは加算できる値ではありません。

同じメソッドから派生するそれぞれのパスの相対的な影響を判断するために、選択したメソッドと各下位パスをつなぐライン上のパーセント値を比較します。各ライン上の値は、上位メソッドの時間に占める、そのパスで呼び出される下位メソッドの時間のパーセントです。図 7-14 (269 ページ) を例に挙げて説明すると、`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` へのライン上の値は 98.1% です。残りの 1.9% は、呼び出された他のパスに分散しています。これは、`Form.Main` が `Form.CtoF` を呼び出すパスが、`Form.Main` で費やされた CPU 時間の 98.1% を占め、これが下位メソッドの実行に起因していることを意味します。このパスからトラブルシューティングを開始します。

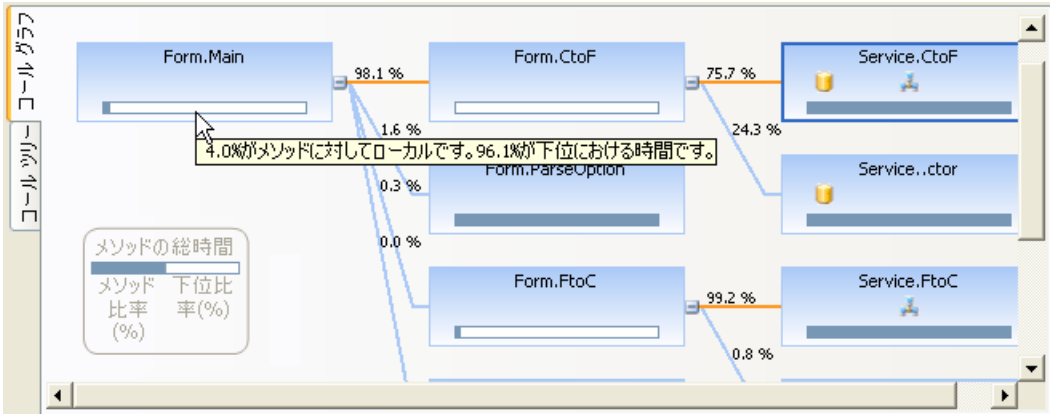


図 7-14.　下位メソッドの影響の理解

呼び出されたパスを調べるときは、各ノードの下部にある水平バーに注目します。このバーは、メソッドが呼び出した下位メソッドと比較して、メソッドの本体で費やされた時間の相対的な割合を示すものです。バーの上にマウスを移動すると、実際のパーセンテージが表示されます。このバーを、問題解決の指針として活用します。たとえば、メソッドの本体で費やされた時間が 4% であり、下位メソッドに対して費やされた時間が 96% である場合、最もコストの高いコール パスを調査して、パフォーマンスに影響を及ぼす下位メソッドを特定します。該当するメソッドを修正するか、そのメソッドの呼び出し回数を少なくするようにコードを変更します。一方、メソッドの本体で 96% の時間が費やされている場合は、メソッド本体で問題を解決します。

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、待機時間のいずれかのアイコンが示されているかどうかに注目します。アイコンの上にマウスを移動すると、動作の重要性が表示されます。1 つのノードに、1 つまたは複数のアイコンが示されている場合は、**[コール ツリー]** ビューに切り替えて、問題の診断に役立つ適切なデータ カラムを追加することを検討します。

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]**でソートされます。大半の時間が費やされているところを見つけるには、その他のカラムの値を細かく調べます。そうすることで、待機時間、ディスク I/O、ネットワーク I/O、CPU 使用量の多い処理のいずれが主な要因になっているかを判断できます。より詳細な情報が必要な場合は、ディスクまたはネットワークに関する読み取り、書き込み、エラーなどのカラムを追加して表示することができます。

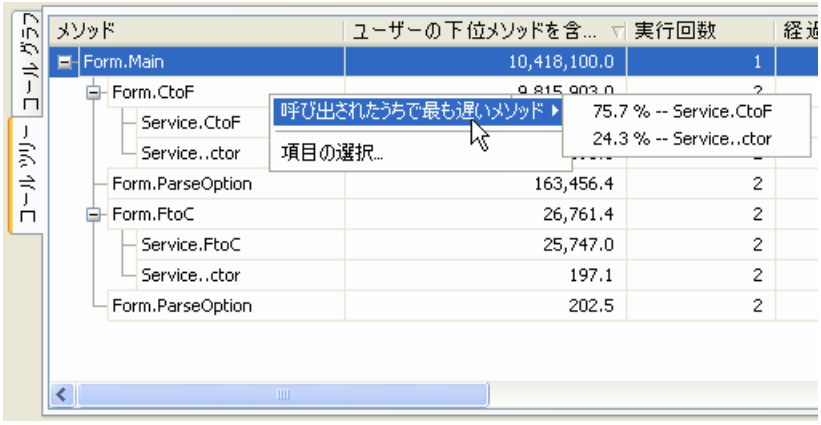


図 7-15.　コール ツリーで選択したメソッドに関する追加データの表示

たとえば、**コール グラフ**でパフォーマンスの低いメソッドがネットワーク I/O を行っているのがわかったら、**コール ツリー**に切り替えて、ネットワーク関連のすべてのデータ カラムをテーブルに追加します。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、コンテキスト メニューから**[項目の選択...]**を選択します。各カラムに示される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：「ユーザーの下位メソッド」または「ユーザー メソッド」とは、ユーザーのソース コードのメソッドを指します。

コール グラフと**コール ツリー**のどちらを使用している場合でも、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]**タブが表示されます。これらのタブは、**メソッド** テーブルの場合と同様に機能します。ただし、表示されるデータの計算に、ユーザー コード、ソース コード、下位メソッドに起因するデータが含まれる点が異なります。**[ソース]**タブを使用すると、**コール グラフ**や**コール ツリー**で選択した任意のメソッドにおいて、最もパフォーマンスの低い行を直ちに特定できます。**[コール スタック]**タブを使用すると、その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たり、スタック内で選択したメソッドを呼び出した行を特定できます。**[ソース]**タブまたは**[コール スタック]**タブでコード行をダブルクリックすると、Visual Studio が開いて該当する行までジャンプするので、そこで行を編集できます。

パフォーマンスは低いが特定の問題がない場合

アプリケーションのパフォーマンスが全般的に低いのに、特定の問題を特定できない場合を考えてみます。パフォーマンスの調整は、反復プロセスです。前述した方法を使用して、パフォーマンスの向上を図ることができます。

- パフォーマンス エキスパートを使用して、アプリケーションを実行します。
- **[最も多く CPU を使用するパス]** グラフを調べ、クリティカル パスごとに、最もコストの高い分岐を最適化するようにします。
- 同様に、**[最も多く CPU を使用する単一メソッド]** リストを細かく調べ、リスト内の上位メソッドを最適化するようにします。
- 再度テストを実行し、改善を確認します。

Web アプリケーションからのデータの収集

Web アプリケーションを含め、マネージ 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 Web アプリケーションを実行するときは、以下の点に注意してください。

マネージ コードのみ

他のいくつかの DevPartner 機能と異なり、パフォーマンス エキスパートではマネージ アプリケーションのデータだけを収集します。したがって、アプリケーションで Internet Explorer をクライアントとして使用している場合、セッション ファイルに Internet Explorer のデータは表示されません。ASP.NET または Web サービス アプリケーションの場合は、サーバー側データが表示されます。

web.config の要件

DevPartner パフォーマンス エキスパートで ASP.NET アプリケーションのプロファイルを正常に実行するには、以下の条件を満たす必要があります。

- プロジェクトに **web.config** ファイルが含まれる必要があります。
- プロジェクトの設定でデバッグを有効にする必要があります。このために、**web.config** ファイルには、debug 属性を true に設定した compilation エレメント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ル

パフォーマンス エキスパートを Visual Studio IDE から、または DevPartner コマンド ライン スイッチを使用してコマンド ラインから実行すると、セッションごとに単一のプロセス（またはサービス）のデータが収集されます。複数プロセスでアプリケーションが実行される場合、または IIS および対象のアプリケーションが実行されるプ

ロセスなどのサービスに関するデータを収集する必要がある場合、**DPAnalysis.exe**（DevPartner 分析ツールのコマンド ライン実行可能ファイル バージョン）と XML 構成ファイルを使用して、セッションを管理することができます。詳細については、「DPAnalysis.exe と XML 構成ファイルの使用」（337 ページ）を参照してください。

注意： **DPAnalysis.exe と XML 構成ファイルを使用すると、複数のプロセスまたはサービスのデータを同時に収集できますが、一般的には、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のが最適です。複数プロセスの場合、データ収集のオーバーヘッドがプロセス間の通信に影響を及ぼす可能性があり、また、アプリケーションの速度が遅くなって経過時間値が大きくなることがあります。複数のプロセスについてパフォーマンス エキスパート データを同時に収集した場合は、ディスク I/O、ネットワーク I/O、同期の待機時間のタイミング値が大きくても、それはプロファイルによるオーバーヘッドの増加を反映したものかもしれません。1 つのプロセスを対象にセッションを再度実行し、調査する価値があるくらいタイミング値が大きいかどうかを確認してください。**

IIS 6.0 での単一プロセスのプロファイル

IIS 6.0 では、1 つのワーカー プロセスについてのみ、パフォーマンス エキスパート データが収集されます。IIS では、アプリケーション プールごとに 1 つのワーカー プロセスがあります。したがって、システムで Web サービスと Web サービス クライアントを実行し、どちらも同じアプリケーション プールで実行している場合、サービスをパフォーマンス エキスパートで開始し、クライアントを Visual Studio の別のインスタンスでパフォーマンス エキスパートなしで開始したとしても、両方のデータが収集されます。クライアントを別のアプリケーション プールで実行するようにアプリケーションを変更した場合、パフォーマンス エキスパートを有効にして起動したアプリケーション（この場合はサービス）のみのデータが収集されます。

DLLHOST で実行されるコンポーネントのリモート セッション ファイルは生成されない

リモート システム上にある **dllhost.exe** と通信するプロセスについてパフォーマンス エキスパートを実行すると、**dllhost.exe** の終了時にリモート システムに関する最終的なセッション ファイルが生成されません。

リモート マシン上のソース コード

DevPartner Studio は、開いているセッション ファイルと同じマシンにソース ファイルが存在することを前提とします。

- ソース コードを表示するときに **[ファイル]>[開く]** ダイアログを開いた場合、そのダイアログでリモート システム上の正しい場所を参照します。
- リモートの 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ために、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]** エントリの値を調べる必要があるかもしれません。

セッション ファイルは開いているソリューションに保存される

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。IISからWebオブジェクトを直接開くと、Visual Studioでプロジェクトを開くときとは異なり、別のソース ファイルが使用されることがあります。最初のソリューションで作成されたDevPartnerセッション ファイルは、2回めのソリューションでは表示されません。

データ収集の自動化

DevPartnerパフォーマンス エキスパートは、**DPAnalysis.exe**という実行可能ファイルを使用して、コマンド ラインから実行することができます。このファイルは、**¥Program Files¥Compuware¥DevPartner Studio¥Analysis¥**ディレクトリにあります。コマンド プロンプトからパフォーマンス エキスパートでアプリケーションを実行するか、バッチ ファイルを作成してパフォーマンス エキスパートでアプリケーションを実行して、データ収集を自動化することができます。パフォーマンス エキスパート セッションを起動するには、2つの方法があります。

- 標準的なMS-DOSコマンド構文でターゲットと引数を指定する
- 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する

コマンド ライン スイッチの使用

「アプリケーションのスケーラビリティ問題」（268ページ）で説明した例を考えてください。品質保証エンジニアは、アプリケーションを毎晩実行するように自動テスト（またはテスト スイート）を設定することで、日常的にスケーラビリティ（または、アプリケーションの他の局面）を監視することができます。テストを自動化するには、バッチ ファイルを以下のように設定します。

1. 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開始する
2. ロード テスト アプリケーションを開始し、実行したい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終了する

アプリケーション終了時に、DevPartnerによって、セッション ログ ファイルが自動的に作成されます。

セッションを起動するときのコマンド 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 DevPartnerパフォーマンス エキスパートに分析のタイプを設定します。

/E 指定し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。

/O セッション ファイルの出力ディレクトリとファイル名のいずれかまたは両方を指定します。

/W プロセスの作業ディレクトリを指定します。

/H ターゲットを実行するホスト マシンを指定します。

/P または /S ターゲットがプロセスとサービスのどちらであるかを指定します。いずれか 1 つのみ指定します。

スイッチを指定する順序については、1 つだけ制限事項があります。/P または /S スイッチは、最後に指定することが必要です。/P（または /S）スイッチの後ろにスイッチを指定すると、そのプロセス（またはサービス）への引数として解釈されます。

XML 構成ファイルの使用

XML 構成ファイルを使用する場合、`dpanalyis.exe /C [path]configuration_file.xml` のようにコマンド ラインはずっとシンプルになります。

構成ファイルには、DevPartner 分析のタイプに必要なパラメータを含めることができます。これには、コマンド ライン スイッチを使用するときには使用できないオプションも含まれます。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、構成ファイルでは `ExcludeImages` 要素を使用する必要があります。

```
<?xml version="1.0" ?>
<ProductConfiguration xmlns="http://www.compuware.com/products">
        <RuntimeAnalysis Type="Expert" MaximumSessionDuration="1000"/>
        <Targets RunInParallel="true">
                <Process CollectData="true" Spawn="true" NoWaitForCompletion="false">
                        <AnalysisOptions NO_MACH5="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 7-16.　XML 構成ファイルでのセッション詳細の指定

リモート マシン上で実行するプロセスのデータを収集するには、ディレクトリ名とファイル名を指定することが必要です。構成ファイルの AnalysisOptions で SESSION_FILENAME 要素と SESSION_DIR 要素を使用します。

データ収集を管理するために構成ファイルを使用する方法の詳細については、「DPAnalysis.exe と XML 構成ファイルの使用」（337 ページ）を参照してください。

QA エンジニアは、翌朝に、セッション ログ ファイルを細かく調べます。パフォーマンスを示す数値が悪化している場合、適切な開発者に対してセッション ログを送信します。QA は、このような方法で、開発サイクルを通してアプリケーションの健全性を追跡します。問題が明らかになると、開発チームはセッション ログ ファイルを活用し、その問題の本質を迅速に判断します。さらに、開発チームは、その問題は前日のコード変更に起因していることを把握しているため、問題を修正するために確認する必要のあるコードの分量を大幅に減少させることができます。

DPAnalysis.exe の使用方法の詳細については、付録 C「コマンド ラインからの分析の開始」を参照してください。

分散アプリケーションからのデータの収集

DevPartner では、リモート システムで実行している分散アプリケーション コンポーネントからパフォーマンス エキスパート データを収集することができます。この場合、リモート システムがリモートのデータ収集に適切なライセンスを持っていることが必要です。Visual Studio から実行する場合、または従来のコマンド ライン構文を使ってコマンド ラインから **DPAnalysis.exe** を実行して実行する場合は、リモート セッションを起動する前に、パフォーマンス エキスパート セッションが 1 回の実行で 1 つのプロセスを監視することを確認してください。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ターゲットとすることができますが、パフォーマンス エキスパート セッションでは、一般的に単一プロセスをターゲットとするのが最適です。アプリケーションを複数のプロセスで実行している場合は、2 番めのプロセスをターゲットとしてアプリケーションを再度実行するようにしてください。スクリプト ファイルやバッチ ファイルを使用してアプリケーションを実行すると、アプリケーションを両方のセッションでまったく同じに実行できます。概要については、「データ収集の自動化」（273 ページ）を参照してください。

XML 構成ファイル オプションを使用して **DPAnalysis.exe** を実行する場合は、アプリケーションの 1 回の実行で、2 番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）収集することもできます。2 つ以上のプロセスやサービスについて同時にデータを収集することはできますが、複数プロセスによって生じるデータ収集のオーバーヘッドがプロセス間の通信に影響を及ぼす可能性があり、アプリケーションの速度が遅くなって経過時間値が大きくなることがあります。複数のプロセスについてパフォーマンス エキスパート データを同時に収集した場合は、ディスク I/O、ネットワーク I/O、同期の待機時間のタイミング値が大きくても、それはプロファイルによるオーバーヘッドの増加を反映したものかもしれません。1 つのプロセ

スを対象にセッションを再度実行し、調査する価値があるくらいタイミング値が大きいかどうかを確認してください。

DPAnalysis.exeによるリモート データの収集

DPAnalysis.exeを使ってリモート プロセスを派生させることはできません。できるのは、リモート マシン上のプロセスについてデータ収集を有効にすることだけです。たとえば、以下のコマンド ラインがあります。

```
DPAnalysis.exe /host remotemachine /p c:¥MyDir¥target.exe
```

DPAnalysis.exeは**target.exe**のプロファイルを設定しますが、リモート マシン上でそれを起動することはしません。**target.exe**を（何らかの手段で）リモート システムで起動すると、プロファイルが開始されます。

これは、リモート サービスについては当てはまりません。リモート サービスはリモートで起動できます。以下に例を示します。

```
DPAnalysis.exe /host remotemachine /s servicename
```

このコマンド ラインを使用すると、プロファイルが有効になり、**remotemachine**上で**servicename**サービスが起動されます。

オプションで、XML構成ファイルを使用して、上記のコマンド ライン例にパラメータを指定できます。**DPAnalysis.exe**の詳細については、付録C「コマンド ラインからの分析の開始」を参照してください。

リモート マシンへのセッション ファイルの保存

リモート プロファイルのシナリオでは、4種類のすべての分析（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）のセッション ファイルがリモート マシンに保存されます。リモート プロセスまたはリモート サービスの場合、コマンド ラインまたはXML構成ファイルにディレクトリとセッション ファイル名を指定する必要があります。指定したディレクトリは、リモート マシンに存在している必要があります。ディレクトリまたはファイル名を指定しない場合、リモート マシンに**[名前を付けて保存]**ダイアログが表示されます。

セッション ファイルの表示

DevPartner Studioがインストールされているマシン（プロファイルを開始した、クライアント ファイルが保存されているマシンなど）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構成ファイルで、リモート マシンのマッピングしたドライブを指定し、DevPartner Studioがインストールされている別のマシン（たとえばプロファイルを開始したマシンなど）にセッション ファイルを保存します。

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たデータの収集

DevPartner Studio は Windows ターミナル サービスをサポートしています。DevPartner とターミナル サービスを使用する方法については、「ターミナル サービスとリモート デスクトップの使用」（9 ページ）を参照してください。

リモートのプロファイルと Windows XP Service Pack 2

Windows XP Service Pack 2 によって、リモート アプリケーションのセキュリティ レベルは高くなりました。Visual Studio からプロファイルする場合、この新しいセキュリティ設定によって、一部のサーバー側アプリケーション コンポーネントのデータ収集が妨げられることがあります。リモート マシン上にあるアプリケーション コンポーネントからデータを収集するには、セッションに参加している Windows XP SP 2 マシン（リモート マシンとプロファイルを開始したクライアント マシンの両方）上のセキュリティ設定を変更する必要があります。

以下の手順では、リモートのプロファイルを可能にするために Windows XP Service Pack 2 のセキュリティ設定を変更する 3 つ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 サービスが有効な場合、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。以下の操作を行います。

1. **[スタート]** メニューの **[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]** タブを選択します。
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のデフォルトの場所は以下のとおりです。

   ```
   C:¥Program Files¥Compuware¥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、**NCS.exe** を選択してから **[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]** タブにある **[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）マシンとローカル（クライアント）マシンの両方でセキュリティ設定を変更する

以下の手順に従い、セキュリティ設定を変更します。

1. **[コントロール パネル]**の**[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]**を開きます。
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]**構文の**[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. **[ANONYMOUS LOGON]**ユーザーがなければ、追加します。
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]**の両方を許可します。

設定を変更したときにVisual Studioが実行中の場合、新しい設定を有効にするにはVisual Studioを再起動する必要があります。

クライアント マシン上の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、プロファイルを開始するクライアント マシンで以下の手順に従います。

1. **[スタート]**メニューの**[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**[管理ツール]**を選択します。**[管理ツール]**ウィンドウから**[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]**を探し、**[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]**で**[COMセキュリティ]**タブを選択します。
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、以下のように変更します。
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**[許可]**チェック ボックスをオンにします。
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、以下のように変更します。
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**[許可]**チェック ボックスをオンにします。

ファイアウォールとリモートのデータ収集

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 マシンからセッション 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、プロセス間通信トラフィックをインターネット アドレス**0.0.0.0**、ポート**18441**でリスンします。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の設定を変更し、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。

XML 形式への DevPartner データのエクスポート

パフォーマンス エキスパート データはXML形式にエクスポートできます。XML形式でデータをエクスポートすると、自社製またはサードパーティ製のソフトウェアを使用して、データの分析、他のツールで作成したデータとの統合、データ ウェアハウスへのデータのアーカイブが容易になります。

DevPartnerパフォーマンス エキスパート セッション ファイル（拡張子は **.dppxp**）はXML形式にエクスポートできます。保存したパフォーマンス エキスパート セッション ファイルを開いているとき、**[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。

また、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。「分析データのXMLへのエクスポート」（363ページ）を参照してください。

DevPartnerのインストール ディレクトリにある **DevPartnerPerformanceExpert82.xsd** には、パフォーマンス エキスパートがセッション ファイルのエクスポートに使用する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の使用

パフォーマンスの調整は、反復プロセスです。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機能と組み合わせて使用します。まず、パフォーマンス分析で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、パフォーマンスのベースラインを把握します。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やネットワーク I/O、同期の問題が関係する問題）を解決します。問題を修正したら、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**機能を使用して状況が改善されていることを検証します。以下に例を示します。

1. 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ます。
2. パフォーマンスが遅く見えるメソッドに注意します。
3. 問題のメソッドを修正する方法がすぐにわからない場合、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]** グラフに表示されるかどうかを確認します。

5 **[最も多く CPU を使用するパス]** グラフのメソッドをクリックすると、**コール グラフ**が開きます。**コール グラフ**には状況に応じたメソッドが表示され、メソッドまたはその下位メソッドがパフォーマンス問題の原因かどうかが示されます。

6 問題のメソッドには、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が表示されます。

たとえば、メソッドがネットワーク動作を示す場合、**[コール ツリー]** タブに切り替え、コンテキスト メニューを使用してネットワーク関連データのカラムをビューに追加します。追加データによって、問題の原因が読み取り動作、書き込み動作、読み取りエラー、書き込みエラーのいずれであるかを判断できます。**[最も多く CPU を使用する単一メソッド]** グラフからドリルダウンした場合、**メソッド** テーブルにデータ カラムを追加できます。

7 **[コール スタック]** タブを使用すると、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 スタックがわかります。

8 **[ソース]** タブを使用して原因のコード行を特定し、Visual Studio のソース ファイルにジャンプして編集します。

問題を修正したら、2 回めの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。前回のパフォーマンス分析セッション 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を使用してセッションを比較し、改善を確認します。

パフォーマンス エキスパートとパフォーマンス分析は互いに補完し合う機能ですが、タイミング データを計算する方法には違いがあります。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を実行し、同じアプリケーションに対してパフォーマンス エキスパート セッションを実行すると、パフォーマンス分析の **[ソース メソッドの上位 20 位]** 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]** グラフに含まれるメソッドが一致しないか、メソッドが同じ順序で表示されないことがあります。

パフォーマンス分析セッションでは、あるメソッドに使用された時間のパーセント値 (**[メソッド比率 (%)]** カラム) は、ユーザーまたはシステムの下位メソッドを除外して計算されます。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]** グラフと **メソッド** テーブルに表示されるメソッドに使用された時間のパーセント値に、システムの下位メソッドに使用された時間も含まれます。

パフォーマンスのプロファイルを実行した場合、マネージ アプリケーションが .NET Framework でのメソッドの実行に多くの時間を費やしていることに気付くかもしれません。システムの下位メソッドをパフォーマンス エキスパートの結果に含めることにより、それ自体の実行に長い時間がかかるメソッドよりも、システム コードと通信する方法に問題があるソース コードのメソッドに注目することができます。システム コードの実行が開始されたあとは、そのシステム コードに費やされる時間について何もできませんが、ユーザー コードからシステム コードを呼び出す方法とタイミングについては変更できます。パフォーマンス エキスパートを使用すると、こうした問題の領域を迅速に特定できます。

メモ： パフォーマンス分析セッション ファイルとパフォーマンス エキスパート セッション ファイルを直接比較することはできません。比較できるのはパフォーマンス分析セッション ファイル間のみです。

開発サイクルにおけるパフォーマンス エキスパート

DevPartnerパフォーマンス エキスパートは、ソフトウェアの開発サイクルを通して使用します。ソフトウェア プロジェクトのライフ サイクルにおけるいくつかのポイントでパフォーマンス 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ーにとってメリットがあります。

ソフトウェア設計者

ソフトウェア設計者は、多くの場合、応答時間やスケーラビリティなどの特定の要件を満たすためのプロトタイプを開発する必要があります。最終的な設計を作成する前に、設計者は、プロトタイプがパフォーマンス要件を満たすことを妨害するような動作を特定しなければなりません。可能であれば、メソッドも特定することが必要です。この場合、それらを修正することでパフォーマンスが格段に向上するような、2～3個のメソッドを特定することができれば理想的です。

ソフトウェア設計者は、設計フェーズとプロトタイプ フェーズでパフォーマンス エキスパートを使用して、コードの実行速度と効率性を高めることができます。設計が進むにつれて、定期的にテストを実行することで、プロトタイプ コードが確実に最低限のパフォーマンス要件を満たすようにすることができます。プロトタイプが開発チームに引き渡されると、開発チームは、重要なパフォーマンス問題についてはテスト済みであるという理解のうえで、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

ソフトウェアの開発者は、開発作業中に、高い頻度でパフォーマンス エキスパートを使用する必要があります。コードのチェックイン前の単体テストに加えて、パフォーマンス エキスパートを実行することを検討してください。単体テストでは、コンポーネントが他のコンポーネントの処理を妨げることなく、想定された処理を行っていることを確認します。それと同時にパフォーマンス エキスパートを実行することで、潜在的なパフォーマンス問題の早期警告を得ることができます。コンポーネントをアプリケーションに完全に統合してからだと、問題を修正するのはより困難になります。

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）のテスト アンド ランが可能になったら、開発者はパフォーマンス エキスパートを自動テスト ルーチンに組み込むことができます。これは、コーディングしてデバッグするときに、潜在的なCPUの使用量、ファイルI/O、ネットワークI/O問題を特定することを目的とします。開発者は、毎朝、パフォーマンス エキスパート セッション ログを調べ、前日のコーディングによって新たなパフォーマンス問題が発生していないかどうかを確認します。発生が確認された場合は、ただちにその問題に対処します。コーディング完了後、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 ログを出力し、パフォーマンスの目標が達成されていることを記録します。

品質保証エンジニア

品質保証（QA）チームは、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 パフォーマンスを継続的に監視できます。QA チームは、パフォーマンス エキスパートを自動テスト スイートに簡単に統合して、重要な領域でのアプリケーション パフォーマンスの数値を毎日取得することができます。問題が発見された場合、QA チームは開発チーム宛てにセッション ログを送信します。または、ログをバグ レポートに添付して Compuware TrackRecord などのバグ追跡システムに登録することができます。

担当のエンジニアは、日常的にセッション ログ ファイルで重要なメトリクスを確認できます。セッション ログに問題が示唆されていた場合、QA エンジニアは、即座にその問題に取り組むことができるように、担当の開発者にログ ファイルを送信します。

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で、ソフトウェア開発チームのすべてのメンバーが恩恵を受けることができます。製品管理の面でもメリットがあります。重要なマイルストーンにおいて、パフォーマンス エキスパート セッション ログを、事前事後のパフォーマンス分析セッション ファイルと組み合わせて使用して、製品がパフォーマンス面での期待値を満たすことを記録できます。

Visual Studio Team System へのデータの送信

Microsoft Visual Studio Team Explorer クライアントがインストールされ、Team Foundation Server の接続が使用可能になっている場合に、DevPartner Studio は Microsoft Visual Studio Team System をサポートします。

DevPartner パフォーマンス エキスパートのセッション ファイルの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。

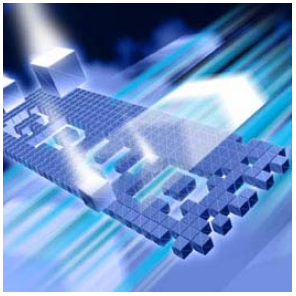
- **メソッド**詳細ビューの**メソッド** テーブル
- **パス**分析ビューの**コール ツリー**

バグを送信すると、**メソッド** テーブルまたは **[コール ツリー]** ビューの表示カラムのデータが**作業項目**フォームにコピーされます。**作業項目**に送信するメソッド 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てください。

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」（8 ページ）を参照してください。

第 8 章

システムの比較



- System Comparison の機能
- すぐに System Comparison を使用するには
- System Comparison サービス
- 相違点のカテゴリ
- レジストリ キーの比較
- 特定ファイルの比較
- DevPartner から独立したインストール
- コマンド ラインからの比較ユーティリティの実行
- Software Development Kit
- System Comparison Snapshot API
- プラグインの記述

この章には 2 つのセクションがあります。1 つめのセクションには、はじめてのユーザーが System Comparison 機能を利用できるように、簡単な操作手順が記載されています。2 つめのセクションには、DevPartner の System Comparison 機能を詳しく理解するための参考情報が記載されています。

システムの比較に関するその他のタスクに基づく情報については、DevPartner System Comparison のオンライン ヘルプを参照してください。

System Comparison の機能

DevPartner System Comparison 機能では、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次のような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

System Comparison では、システムを比較するために、スナップショット ファイルという XML ファイルが作成されます。このファイルには、インストールされた製品、システム ファイル、ドライバ、その他多くのシステム特性など、コンピュータ システムに関する情報が記載されます。ファイルの作成後、スナップショット ファイルが比較され、相違点が報告されます。

他の DevPartner 機能とは異なり、System Comparison は Visual Studio 環境に統合されません。ターゲット システムへの影響を最小限に抑えるために、スタンドアロン ユーティリティとして実行されます。

System Comparison は以下の要素で構成されます。

- サービス。システムのスナップショットを毎晩作成します。
- ユーザー インターフェイス。これを使って、スナップショットを手動で作成し、スナップショットを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。
- Software Development Kit（SDK）。SDK を使用してソフトウェアを開発すると、比較に関する追加情報を集めたり、運用アプリケーションにスナップショット機能を埋め込んだりすることができます。

図 8-1. System Comparison のユーザーインターフェース

すぐに System Comparison を使用するには

以下の準備、設定、実行手順では、DevPartner System Comparison の使用方法を紹介します。

機能をすぐに使用したい場合は、色付きの枠内に記載されている手順に従ってください。詳細な情報については、枠の下に記載されている説明を参照してください。

メモ： DevPartner System Comparison を使用してシステムを分析するために、昇格したシステム特権は必要ありません。DevPartner でのシステムの分析には、ユーザーのシステム上でファイルを作成したり、アプリケーションを操作したりするために使用するシステム権限で十分です。

以下の手順で、コンピュータに細かい変更を加え、コンピュータの現在の状態と前の状態を比較してみます。

準備：比較内容の検討

System Comparison を実行する前に、比較の目的を把握します。

この手順では以下を前提としています。

- DevPartner 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、スナップショットが 1 つ作成されています。

  System Comparison をインストールすると、サービスが自動的に開始され、開始から数分以内に最初のスナップショットが作成されます。このサービスは、システムのサービス リストに [DevPartner Differ] として表示されます。
- 1 台のコンピュータの異なる状態を比較します。

比較する内容を正確に特定することで、適切に比較を設定できます。たとえば、以下のような目的が考えられます。目的によっては、追加の設定が必要になることがあります。

- 製品のインストールと削除が、コンピュータのサービス、設定、レジストリ キー、ファイルに与える影響を確認する（レジストリ キーまたはファイルの確認には、XML ファイルを変更する追加の設定が必要です）。
- システムの変更によって、以前は機能していたシステム上の製品が機能しなくなった可能性があるかどうかを確認する。
- ある製品への変更による影響の程度を確認する（たとえば、自動化されたテストへの影響）。
- 新しい開発システムに、以前の開発システムで使用していたツールがすべてあるかどうかを確認する。
- あるシステムで、製品が機能しない理由、または機能が異なる理由を判断する。
- エンドユーザー サイトに展開済みの製品の問題を解決する。

設定：System Comparison の準備

比較の目的を決定したら、場合によってはいくつかの設定タスクを実行する必要があります。

この手順では、デフォルトの DevPartner System Comparison オプションを使用できます。その他の設定は必要ありません。

設定タスクが必要な状況として、以下のような例があります。

- レジストリ キーまたは特定のファイルを比較する場合、設定タスクには **`RegistrySections.xml`** ファイルまたは **`FileSections.xml`** ファイルの変更が含まれます（294 ページと 296 ページを参照してください）。

- デフォルトで収集されないデータを比較する場合、設定タスクにはカスタム プラグインの記述が含まれます（303ページを参照してください）。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 (291 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、設定タスクには、2台めのコンピュータへの System Comparison のインストール、スナップショットの作成、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手順が含まれます（298ページページを参照してください）。

実行：変更とスナップショットの作成

システム比較を開始する準備が整いました。この手順では、コンピュータに変更を加え、現在の状態と以前の状態を比較します。

システムの相違点がどのように報告されるかを示すために、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加えます。

1. **[コントロール パネル]>[管理ツール]>[サービス]** ウィンドウを開き、作業環境に影響が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 サービスを停止します（あとで再開できるように、変更したサービスをメモしておきます）。
2. [スタート] メニューから **[すべてのプログラム]>[Compuware DevPartner System Comparison]** を選択します。
3. **[System Comparison]** ウィンドウの **[このコンピュータの現在の状態と過去の状態を比較]** 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（290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自動的に毎日作成され、スナップショット ファイルの日時が表示されます。

メモ： System Comparison のインストールから数分経過していれば、リストには少なくとも1つのファイルが表示されます。ファイルが表示されない場合、System Comparison サービスが実行中かどうか確認してください。このサービスは、サービス リストに `[DevPartner Differ]` として示されます。

4. このリストから、比較の基準として使用するスナップショットの日時を選択し、**[比較]** をクリックします。

   System Comparison に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、「結果の分析」で説明します。

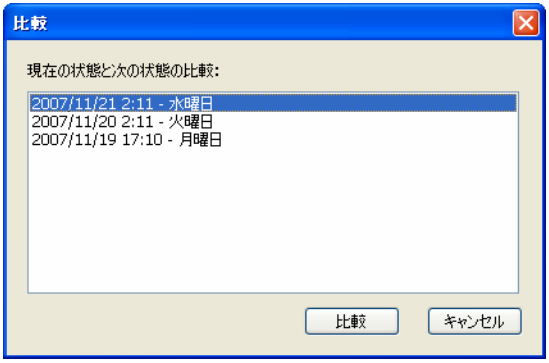


図 8-2.　　スナップショット ファイルのリスト

結果の分析

System Comparison で2つのスナップショットを比較すると、図8-3のように結果ウィンドウに相違点が表示されます（準備、設定、実行手順の結果ウィンドウに表示される情報は、この図よりもはるかに少ないかもしれません）。

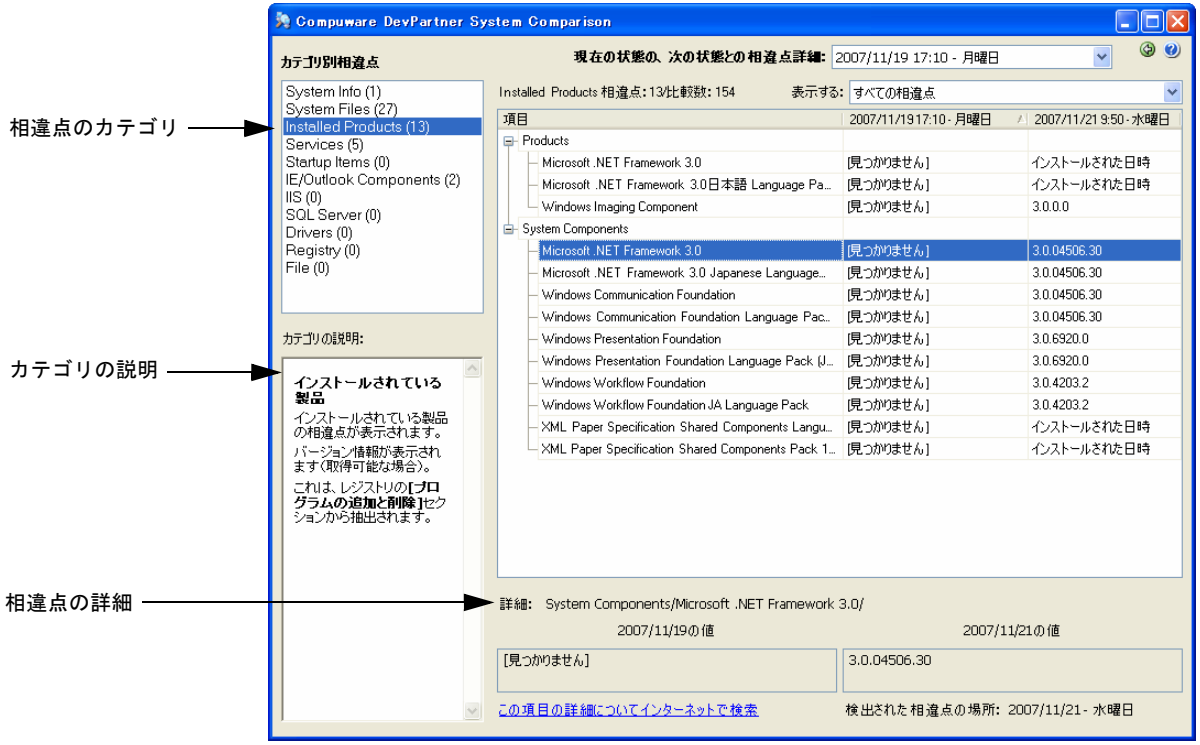


図 8-3.　　結果ウィンドウ

左上ペインには、比較したカテゴリと、各カテゴリで見つかった相違点の数が表示されます。ウィンドウを開くと、相違点の数がゼロ以外の最初のカテゴリが選択されます。

左下ペインには、選択したカテゴリの説明が表示されます。

右ペインには、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. 説明を表示するカテゴリをクリックします。
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]ペインの1列めには項目名が表示されます。2列めと3列めにはスナップショットからの情報が表示されます。見出し行にはコンピュータ名と比較の完全なタイムスタンプが表示されます。

   そのスナップショットにない項目は、[見つかりません]と表示されます。コンピュータ上の項目には、チェック マークまたは「installed」(インストール済み)という単語が表示されます。
3. 詳細ペインの下部にある2つの列には、選択した項目について、1つめと2つめのスナップショットの実際のデータが表示されます。
4. 画面の下部には、**[この項目の詳細についてインターネットで検索]**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に関係がある項目についてインターネット検索が実行されます(たとえば、「Windowsのシステム環境変数」)。
5. **[相違点の詳細]**ペインの右上の**[表示]**リストをクリックします。表示する相違点をフィルタするにはこれらのオプションを使用します。
6. 相違点の確認が終わったら、ウィンドウの右上にある戻る  ボタンをクリックして、メインの**[DevPartner System Comparison]**ウィンドウに戻ります。

結果ウィンドウには相違点のみが表示されます。2つのスナップショットで同じだった項目は表示されません。

System Comparisonでは、相違点の評価時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。コンポーネントのバージョン番号が異なる場合、異なるコンポーネントと判断されます。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もコンポーネントのバージョン番号が異なる場合、コンポーネントは[見つかりません]と表示されます。

現在の状態を別の以前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の操作の練習が終了したら、忘れずに停止したサービスを再開します。

この章の準備、設定、実行セクションはこれで終了です。ここまでで、System Comparison の実行方法について基本的な知識が習得できたはずです。詳細情報については、この章の残りを続けて読んでください。また、タスクに基づく情報については 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

System Comparison サービス（DevPartner Differ）では、コンピュータが起動している場合、毎日午前 2 時 10 分にコンピュータの状態のスナップショットが自動的に作成されます。コンピュータの電源がオフの場合、次回の起動 5 分後にスナップショットが作成されます。System Comparison をインストールすると、System Comparison サービスの開始数分後にスナップショットが作成されます。

夜間スナップショット サービスでは 21 日間毎晩スナップショットが作成され、それ以降は古いものから削除されます。System Comparison ユーティリティの設定ファイルの値を変更すると、保持するスナップショットの数を変更できます。「保持するスナップショットの数の変更」（290 ページ）を参照してください。スナップショット ファイルのサイズは収集するデータ量によって変わります。一般的なファイル サイズは 1 メガバイト未満です。

System Comparison サービスは最低の優先度で実行されますが、実行中の数分間はある程度のシステム リソースが消費されます。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できますが、自動的にスナップショットを作成する機能は無効になります。

自動スナップショット設定の変更

System Comparison サービスで作成される自動スナップショットのタイミングと、保持す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の値で決まります。設定ファイル（**Compuware.Diff.Settings.xml**）は **Program Files¥Compuware¥DevPartner Studio¥System Comparison¥bin** ディレクトリにあります。

保持するスナップショットの数の変更

System Comparison では、デフォルトで 21 個の自動スナップショット ファイルが保持され、その後は古いファイルから削除されます。保持するスナップショットの数を変更するには、設定ファイルの **SnapshotsToKeep** キーを変更します。たとえば、以下のキーに変更すると、保持するスナップショットの数が 30 個になります。

```
<add key="SnapshotsToKeep" value="30" />
```

スナップショット時間の変更

System Comparison サービスでは、毎日午前 2 時 10 分にコンピュータの自動スナップショットが作成されます（コンピュータの電源がオフの場合、次回の起動 5 分後にスナップショットが作成されます）。このデフォルト時間を変更するには、設定ファイルの SnapshotHour0To23 キーと SnapshotMinute0To59 キーを使用して時間を指定します。たとえば、以下のキーに変更すると、自動スナップショットの時間は午前 3 時 42 分になります。

```
<add key="SnapshotHour0To23" value="3" />
<add key="SnapshotMinute0To59" value="42" />
```

時間の有効な設定は 0 ～ 23 です。分の有効な設定は 0 ～ 59 です。

サービスの新しい設定を有効にするには、再起動する必要があります。その日の自動スナップショットが作成済みの場合、新しい設定は次の日から有効になります。System Comparison で自動スナップショットが作成されるのは 1 日に 1 回のみです。

相違点のカテゴリ

System Comparison ユーティリティでは、スナップショットの作成時に、以下の表に示す項目の存在、バージョン、ステータスが記録されます。

System Comparison プラグインを記述することで、新しいカテゴリを追加し、データ取得をカスタマイズできます。「プラグインの記述」（303 ページ）を参照してください。

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 • G 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	• Windows¥System32 のオペレーティング システム ファイル • Windows¥System32¥dllcache の Windows ファイル保護のキャッシュ－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 の保守に使用される、オペレーティング システム ファイルが含まれています。ファイルが破損するかなくなると、このフォルダのファイルで自動的に置換されます。 • Windows¥WinSxS のサイドバイサイド アセンブリ

表 8-1.　相違点のカテゴリ

カテゴリ	検出される相違点
インストールされている製品	検出された製品。バージョン番号がある場合、それも表示されます。 この情報はレジストリの Add/Remove Programs セクションから読み取られます。
サービス	インストールされているサービスの相違点： • サービスのステータス（実行中、停止など） • サービスが使用するアカウント • サービスの種類 • 依存するサービス
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読み取られます。 • Windows ディレクトリの `Win.ini` ファイル。 • 以下のレジストリ キー。 `HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Run` • 可能な場合、プログラム ファイルのバージョン情報が含まれます。
IE/Outlook コンポーネント	Internet Explorer と Outlook の相違点。 • Active Setup は、レジストリ キー `HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Active Setup¥Installed Components` から抽出された、更新済みまたは見つからない Outlook／Internet Explorer コンポーネントを示します。 • レジストリ キー `HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Explorer¥Browser Helper Objects` から抽出された Browser Helper オブジェクト • 以下のレジストリ キーから抽出された MIME マッピング（MIME タイプと MIME を処理するアプリケーションとのマッピング）`HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Active Setup¥MimeFeature` オブジェクト • レジストリ キー `HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Internet Explorer¥Extensions` から抽出された Internet Explorer 拡張 • レジストリ キー `HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥InternetSettings` から抽出されたインターネット設定

表 8-1.　相違点のカテゴリ

カテゴリ	検出される相違点
IIS	以下のように、インストール済みの全 Web アプリケーションとその設定の相違点など、IIS メタベースから取得できる Microsoft IIS インストールの相違点 • 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 インストールの相違点 • レジストリの Microsoft SQL の設定 • Microsoft SQL サービスと関連サービスに関するデータ • インストール済みの全インスタンスに関するマスタ データベースの syscurconfigs テーブルと sysconfigures テーブルの設定。System Comparison ユーティリティは、統合されたセキュリティを使用して SQL Server に接続を試みます。SQL Server が起動されていない場合、マスタ データベースの相違点は収集されません。 **メモ**：マスタ データベースの相違点を収集する場合、実行するアカウントには、これら 2 つのファイルにアクセスするための権限が必要です。
ドライバ	検出されたドライバの相違点 • インストール済みドライバ • ドライバのステータス
レジストリ	レジストリの特定セクションの相違点。デフォルトで収集されるレジストリのセクションはありませんが、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 収集レジストリのセクションをカスタマイズするには、**Program Files¥Compuware¥DevPartner Studio¥System Comparison¥data** ディレクトリの **RegistrySections.xml** ファイルを編集します。 レジストリ キー データを収集するために十分な権限が必要です。
ファイル	特定のパスのディレクトリの内容とファイル プロパティの相違点。デフォルトで収集されるファイルはありません。収集するパスをカスタマイズするには、**Program Files¥Compuware¥DevPartner Studio¥System Comparison¥data** ディレクトリの **FileSections.xml** ファイルを編集します。

レジストリ キーの比較

システムの比較時にレジストリ設定を対象にすることはよくありますが、システムには数千単位のレジストリ キーが存在する可能性があるため、比較するキーの範囲を絞ると便利です。この場合、インストール パスの data ディレクトリ（デフォルトで **Program Files¥Compuware¥DevPartner Studio¥System Comparison¥data**）にある **RegistrySections.xml** ファイルに、比較するレジストリのセクションを指定します。デフォルトでスナップショットに含まれるレジストリ キーはありません。

メモ： System Comparison ユーティリティの Snapshot Application Program Interface（API）でこのファイルを使用する場合、アプリケーションの実行可能ファイルの 1 つ上位にある **¥data** ディレクトリにこのファイルを置く必要があります。たとえば、実行可能ファイルが **...¥App¥bin¥MyApp.exe** である場合、このファイルのパスは **...App¥data¥RegistrySections.xml** となります。

HKEY_LOCAL_MACHINE と HKEY_CLASSES_ROOT のレジストリ エントリを比較できます。他のレジストリ キーの比較はサポートされていません。

必要に応じて複数のセクションを指定できます。

レジストリ キー データを収集するために十分な権限が必要です。

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。この属性はオプションです。未指定の場合、カテゴリ名としてレジストリ キーが使用されます。

YYY　再帰的に収集を開始するレジストリ キー。このキーには接頭辞の HKEY_LOCAL_MACHINE または HKEY_CLASSES_ROOT を指定しません。たとえば、**KEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Rpc** のすべてのキーを収集するには、以下の構文を使用します。

<Section categoryName="Microsoft RPC">SOFTWARE¥Microsoft¥Rpc</Section>

LOCAL_MACHINE キーまたは CLASSES_ROOT のすべてのキーを収集するには、特殊文字の「¥」を指定します。たとえば、**<Section categoryName="All">¥</Section>** と指定します。ただし、すべてのレジストリ キーを収集すると時間がかかるので注意してください。

キーの種類が REG_BINARY の場合、各キーの先頭 20 バイトのみが収集されます。

例

RegistrySections.xml ファイルの例を以下に示します。

```
<RegistrySections>
<LocalMachine>
<!-- これは RPC 以下のすべてのレジストリ キーを収集する例です -->
<Section categoryName="Microsoft RPC">SOFTWARE¥Microsoft¥Rpc</
Section>
</LocalMachine>
<ClassesRoot>
<!-- これは ClassesRoot 以下のすべてを収集する例で、メガバイト単位の大きなデータになります -->
<Section categoryName="All">¥</Section>
<Section categoryName="Shell Extensions">*¥shellex</Section>
</ClassesRoot>
</RegistrySections>
```

特定ファイルの比較

デフォルトで、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特定ファイルを対象にすることはよくありますが、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 パスの data ディレクトリ（デフォルトで **Program Files¥Compuware¥DevPartner Studio¥System Comparison¥data**）にある **FileSections.xml** を使用します。

メモ： System Comparison ユーティリティの Snapshot Application Program Interface（API）でこのファイルを使用する場合、アプリケーションの実行可能ファイルの 1 つ上位にある **¥data** ディレクトリにこのファイルを置く必要があります。たとえば、実行可能ファイルが **...¥App¥bin¥MyApp.exe** である場合、このファイルのパスは **...App¥data¥FileSections.xml** となります。

比較に含めるファイルの各カテゴリは、**FileSections.xml** の別のセクションに指定します。必要に応じて複数のセクションを指定できます。

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場合、デフォルトでディレクトリ パスがカテゴリ名になります。
`filterPattern`	オプションの属性。ワイルドカード文字の * （0 字以上の文字）および ?（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場合、`filter="*.*"` と同様に処理されます。
`attributes`	オプションの XML 属性。指定しない場合、`attributes="yes"` と同様に処理されます。この属性が "yes" の場合、以下の情報が収集されます。 flag read only（読み取り専用フラグ） encrypted（暗号化） file length（ファイル長） modified date（更新日） 未設定の場合、Company 属性、Product 属性、read-only や debug などのブール型ファイル属性は収集されません。

programAttributes	オプションの属性。指定しない場合、programAttributes="yes" と同様に処理されます。この属性が "yes" で、ファイル名の拡張子が **.exe**、**.dll**、**.ocx**、***.cpl** のいずれかの場合、以下のプログラム バージョン情報が収集されます。 version（バージョン） language（言語） たとえば、品質保証環境で、製品のインストール中に削除または追加されたファイルがあるかどうかを検証する必要があるが、一部のファイル プロパティ（プログラムのバージョンなど）はインストールごとに変更されていることが予想される場合などに、programAttributes を "no" に設定すると便利です。
recurseSubDirectories	オプションの XML 属性。この属性を指定しない場合、recurseSubDirectories="yes" と同様に処理されます。この属性が "yes" の場合、すべてのサブディレクトリのファイル情報が再帰的に収集されます。
YYY	ファイル情報の再帰的な収集を開始するディレクトリ パス。

例

FileSections.xml ファイルの例を以下に示します。

```
-->
- <FileSections>
- <!-- これらはファイル セクションの例です -->
<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 から独立したインストール

DevPartner System Comparison は、他の DevPartner 機能とは別にインストールされます。2 つの異なるコンピュータを比較して、異なるシステム上でアプリケーションの動作が異なる理由を確認する必要があるとき、このオプションが有効かもしれません。システムを比較して相違点を見つける場合、システムに加える変更を最小限に抑えることが重要です。Visual Studio や他の DevPartner 機能をインストールせずに、System Comparison のみをインストールすると、比較対象のシステム間の重要な相違点を特定するのが容易になります。

System Comparison をインストールするには、DevPartner のインストール設定画面で **[DevPartner System Comparison のインストール]** を選択し、インストール手順に従います。

System Comparison は DevPartner ライセンス契約に含まれるため、System Comparison を使用すると DevPartner の 1 ライセンスが使用されます。ライセンスの詳細については『DevPartner Studio インストール ガイド』に記載されていますが、以下の点に注意してください。

- 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 ライセンスを持っている場合、System Comparison の実行中に 1 ライセンスが使用されます。System Comparison サービスの開始とこのサービスによるスナップショットの作成では、ライセンスは使用されません。
- 14 日の評価期間で DevPartner を実行する場合、System Comparison 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、この 14 日は開始されます。System Comparison サービスのインストール時、起動時、スナップショットの作成時に開始されるわけではありません。

コマンド ラインからの比較ユーティリティの実行

データ収集と比較を自動化するには、**CommandLine.exe** と **CommandLineDiff.exe** という 2 つのコマンド ライン インターフェイスを使用できます。

- **Compuware.Diff.CommandLine.exe** では、コンピュータ システムの現状のスナップショットが作成されます。デフォルトでは、スナップショットの格納に使用された最後のディレクトリにスナップショットが格納されますが、コマンドラインのパラメータでディレクトリを指定することもできます。

  例：

  ```
  C:¥Program Files¥Compuware¥DevPartner Studio¥System
  Comparison¥bin>Compuware.Diff.CommandLine.exe
  C:¥Program Files¥Compuware¥DevPartner Studio¥System
  Comparison¥bin>Compuware.Diff.CommandLine.exe c:¥MySnaps
  ```

- **Compuware.Diff.CommandLineDiff.exe** では、2 つの既存のスナップショット ファイルに含まれる値が比較され、結果の相違点が出力ファイルに書き出されます。

  メモ： Windows Vista で実行している場合、出力ディレクトリへの書き込み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（これはプレースホルダです）と、比較するファイル名です。オプションで、出力ファイルを書き込むディレクトリを指定できます。

  例：

  ```
  C:¥Program Files¥Compuware¥DevPartner Studio¥System
  Comparison¥bin>Compuware.Diff.CommandLineDiff.exe
  computers SnapFile1 SnapFile2
  C:¥Program Files¥Compuware¥DevPartner Studio¥System
  Comparison¥bin>Compuware.Diff.CommandLineDiff.exe
  computers SnapFile1 SnapFile2 C:¥MyResults
  ```

  出力ファイルは XML ファイルです。プログラムで読み取って比較結果を確認できます。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** ディレクトリ（デフォルトで **¥Program Files¥Compuware¥DevPartner Studio ¥System Comparison¥bin**）にあります。

Software Development Kit

System Comparison には Software Development Kit（SDK）が含まれます。SDK にはソフトウェア開発者向けの以下のような機能があります。

- Snapshot Application Program Interface（API）を使用して、アプリケーションに関数コールを埋め込み、アプリケーションの運用中にスナップショットをトリガする機能

  Snapshot API を使用すると、アプリケーション開発者は、運用アプリケーション内からスナップショット機能を制御できます。アプリケーションの運用中に問題が発生した場合、埋め込まれた API コールによってスナップショットがトリガされるので、これを問題の診断に利用できます。

- System Comparison プラグインを作成して、スナップショット中に収集する追加情報を指定する機能

  ほとんどの比較の場合、System Comparison ユーティリティが収集する 11 カテゴリの情報（表 8-1（291 ページ）を参照）があれば十分ですが、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、データ取得プラグインを作成して System Comparison ユーティリティをカスタマイズできます。

API とプラグインの機能については、以降のセクションで説明します。

System Comparison Snapshot API

System Comparison Snapshot API を使用すると、運用アプリケーション内からスナップショット機能を制御できます。Snapshot API を使用すると、以下を指定できます。

- スナップショットの格納場所
- メッセージまたはエラーの処理方法
- 進捗ステータスの報告方法
- プラグインの場所（カスタム プラグインを使用する場合）

Snapshot API 情報は、System Comparison インストール ディレクトリ内の 2 つのサブディレクトリに格納されます。

- **`System Comparison¥redistributable`** サブディレクトリには、お客様がアプリケーションのインストールに含めることのできるライセンス取得済みのアセンブリが格納されています。

  Snapshot API アセンブリは、弊社とのソフトウェア ライセンス契約の条項に従って再配布できます。スナップショットの作成にソフトウェアのライセンスを取得する必要はありません。スナップショットの表示と比較には、ライセンス取得済みの DevPartner システムにスナップショットを送信する必要があります。

- **¥System Comparison¥sdk¥SnapshotAPI** サブディレクトリには、アプリケーションでの API の使用法を示すサンプル アプリケーション（**SampleSnapshotAPI.cs**）が格納されています。

  API の使用方法を理解するには、**SampleSnapshotAPI.cs** を参照してください。

Snapshot API は VB.NET、C#、マネージ C++ からアクセスでき、.NET Framework 1.1 と 2.0 でビルドされたアプリケーションで使用できます。

SampleSnapshotAPI.cs アプリケーションの内容に従って、Snapshot API 用に実装されたクラスとメソッドについて説明します。

メモ： Snapshot API を使用する場合、アプリケーションの実行可能ファイルの 1 レベル上の **¥data** ディレクトリをアプリケーションのディレクトリ パスに含める必要があります。**RegistrySections.xml** ファイルと **FileSections.xml** ファイルは、使用しない場合でも、**¥data** ディレクトリに置く必要があります。たとえば、実行可能ファイルが **...¥App¥bin¥MyApp.exe** である場合、**...App¥data¥RegistrySections.xml** と **...App¥data¥FileSections.xml** が存在する必要があります。

スナップショットの作成

Compuware.Diff.Collector はクラス SnapshotAPI を実装します。
クラス SnapshotAPI を使用するとスナップショットを作成できます。

Public SnapshotAPI

(ILoggable logger,
IProgressStatus
progressStatusInterestedParty,
String pluginsSubDirectoryName)

Logger：イベントとエラーの処理を行うクラスのインスタンス。オプションで null を渡すこともできますが、問題の解決が容易になるため、logger の実装をお勧めします。

progressStatusInterestedParty：進捗ステータス メッセージを処理するクラスのインスタンス。スナップショット操作は長時間かかる可能性があるため、ユーザーにフィードバックを送ることが重要な場合があります。オプションで null を渡します。

pluiginsSubDirectoryName：データ取得のアセンブリが格納された実行可能ファイルのディレクトリのサブディレクトリ名。プラグインがない場合、空の既存ディレクトリを指定するか、null を渡すことができます。

SnapshotAPI クラスは以下の 3 つのメソッドを実装します。

public string TakeSnapshot (String snapshotDirectory)	スナップショットを作成し、指定したディレクトリに格納します。
public int GetNumberSteps ()	進捗ステータス オブジェクトを受け取るステップの合計数を指定します。また、それに応じて進捗ステータスを設計できます（ProgressStatus インターフェイスの詳細については303 ページを参照してください。）
public void Dispose ()	スナップショット オブジェクトは破棄可能なオブジェクトです。

以下の例で、最も基本的なスナップショット機能について説明します。

```
using ( SnapshotAPI snapshoter = new SnapshotAPI( null, null,
null ) )
{
string snapFile = snapshoter.TakeSnapshot ( userSnapshotDirectory
);}
```

これでスナップショットは作成されますが、エラー、メッセージ、進捗が追跡されないため、実稼働の設定ではあまり有効ではありません。

メッセージのログ機能

Compuware.Diff.LoggableInterface を使用すると、スナップショット処理中に返されるエラーとメッセージの報告を制御できます。アプリケーションにログ機能を作成して、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書き出します。たとえば、サンプル アプリケーションでは、メッセージのログをコンソールに出力する ConsoleLogger クラスを実装しています。

このインターフェイスは以下の 2 つのメソッドから構成されます。

void Log(string message)	このメソッドを呼び出すと、通常のステータス メッセージのログが出力されます。
void LogError(string message)	このメソッドを呼び出すと、エラー メッセージのログが出力されます。

進捗の報告

スナップショットの進捗を報告および表示するには、Compuware.Diff.ProgressStatus インターフェイスを実装します。このインターフェイスは以下の 3 つのメソッドから構成されます。

`void OneStep ()`	関係者に通知し、進捗表示を 1 ステップずつインクリメントするコール バック メソッド。
`void MultiSteps ( int nbrSteps )`	関係者に通知し、進捗表示を複数ステップずつインクリメントするコール バック メソッド。
`void UpdateStatus (String newStatus )`	関係者に通知し、新しいステータスを処理するコール バック メソッド。 newStatus の文字列は新しいステータスを表します。一般的に、これは進捗ステータスの UI 要素の一部として表示されます。

プラグインの記述

ほとんどの比較の場合、System Comparison ユーティリティが収集する 11 カテゴリの情報（表 8-1（291 ページ）を参照）があれば十分ですが、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、データ取得プラグインを作成して System Comparison ユーティリティをカスタマイズできます。このセクションでは、データ取得プラグインを定義し、付属のサンプルを使用してプラグインがどのように機能するかを示し、独自のデータ取得プラグインを作成する方法について説明します。

プラグインとは

プラグインとは、インターフェイス Compuware.Diff.PluginInterface.IPluggableDataExtractor を実装する 1 つまたは複数のタイプを含む .NET アセンブリです。プラグインでは、比較のために収集するデータの高レベルのカテゴリを定義します。プラグインはデータを抽出し、階層的な方法で基本要素に XML 要素を追加することで、呼び出し元に XML 形式でデータを渡します。

プラグインは製品のインストール ディレクトリの **bin/plugins** に格納されます。このディレクトリのすべての .NET アセンブリが System Comparison サービスによって自動的にロードされ、インターフェイス IPluggableDataExtractor を実装するすべてのタイプがインスタンス化されて、データの抽出時に呼び出すプラグインのリストに入れられます。

プラグインの作成方法を習得するために、System Comparison には以下の 2 つのサンプル ファイルがあります。

- サンプル プラグインの **SamplePlugin.cs**。シンプル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を収集しませんが、最初のサブカテゴリの 2 番めのデータ ポイントに、常にタイムスタンプの相違点を示します。プ

ラグインに実装されるメソッドの詳細については、**¥SDK¥Plugin** のファイル **IPluggableDataExtractor.cs** を参照してください。

- サンプル プラグインで演習を行うためのプログラム **TestDriver.cs**。これを使用すると、サンプル プラグインのメカニズムがわかります。また、このプログラムを使用して、自分でカスタマイズしたプラグインの演習も行えます。プラグインで目的の情報を取得できたら、System Comparison の **bin/plugins** ディレクトリにそのプラグインを格納し、System Comparison 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って演習を行えます。

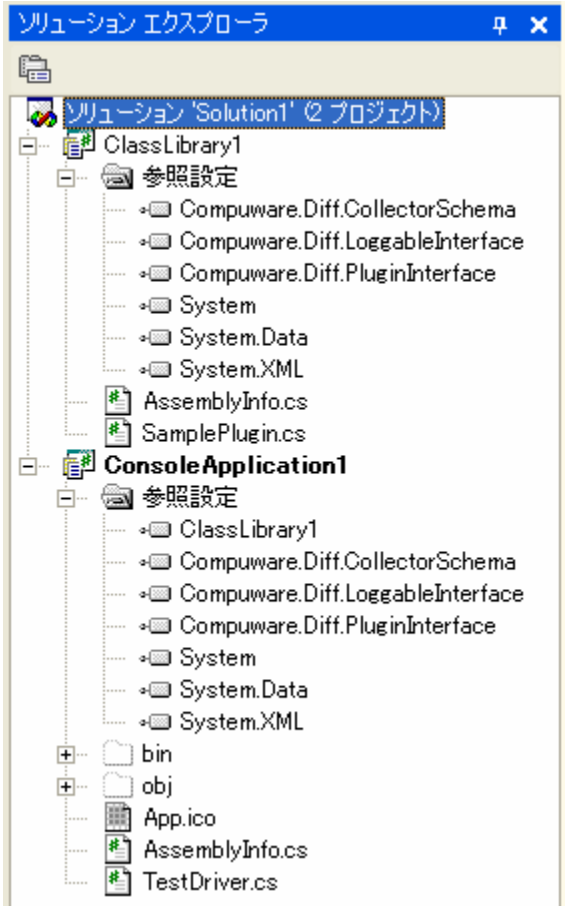
プラグイン サンプルの操作手順

プラグインの使用に慣れるために、**TestDriver.cs** と **SamplePlugin.cs** のサンプル ファイルを使用します。どちらのファイルも **¥sdk¥Plugin** ディレクトリ（デフォルトで **C:¥Program Files¥Compuware¥DevPartner Studio¥System Comparison¥sdk¥Plugin**）に格納されています。

メモ： **プラグインは Visual Studio .NET 2003 で作成する必要があります。** System Comparison ユーティリティではバージョン 1.1 の .NET Framework を使用します。それより新しいバージョンの Framework ではプラグインをビルドできません。Framework 2.0 の機能を使用したい場合、外部プロセスの呼び出しをプラグインに追加して 2.0 の機能を使用できます。

サンプル ファイルのビルドとテストを行うには、以下の手順に従います。

1. Visual Studio .NET 2003 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、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されます。
   - ◇ 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されます。



3 ClassLibrary1 プロジェクトで以下を実行します。

a **Class1.cs** がある場合、ソリューション エクスプローラから削除します。VS 2003 を使用して C# プロジェクトを作成すると、このファイルが自動的に生成され、スタートアップ ファイルとして使用されます。これを削除せずに SamplePlugin を追加すると、エラーが発生します。

b **SamplePlugin.cs**
（デフォルトで **C:¥Program Files¥Compuware¥DevPartner Studio¥System Comparison¥sdk¥Plugin** にあります）を追加します。

c 再配布可能なディレクトリ
（デフォルトで **C:¥Program Files¥Compuware¥DevPartner Studio¥System Comparison¥redistributable**）の以下のアセンブリへの参照を追加します。

Compuware.Diff.PluginInterface.dll
Compuware.Diff.LoggableInterface.dll
Compuware.Diff.CollectorSchema.dll

4. ConsoleApplication1プロジェクトで以下を実行します。
   a. **Class1.cs**がある場合、ソリューション エクスプローラから削除します。
   b. **TestDriver.cs**ファイル（デフォルトで**C:¥Program Files¥Compuware¥DevPartner Studio¥System Comparison¥sdk¥Plugin**にあります）をプロジェクトに追加します。
   c. 再配布可能なディレクトリの以下のアセンブリへの参照を追加します。

      Compuware.Diff.PluginInterface.dll
      Compuware.Diff.LoggableInterface.dll
      Compuware.Diff.CollectorSchema.dll

   d. ClassLibrary1への参照を追加します。
   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、サンプルをステップ実行すると、プラグインの基本的な機能を理解できます。また、サンプル プラグイン データを含むXML出力ファイルが作成されます。このファイル名はpluginOutput.xmlで、テスト ドライバを実行したディレクトリに格納されます。

```
-<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を使ってサンプル プラグインの演習を行ったら、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このプラグインを使用できます。

1. **ClassLibrary1.dll**をプラグイン サブディレクトリにコピーします。
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作成し、続けて2つめのスナップショットを作成します。
3. 2つのスナップショットを比較します。SamplePluginはタイムスタンプ データを収集するため、2つのスナップショットにはこの相違点が表示されます。

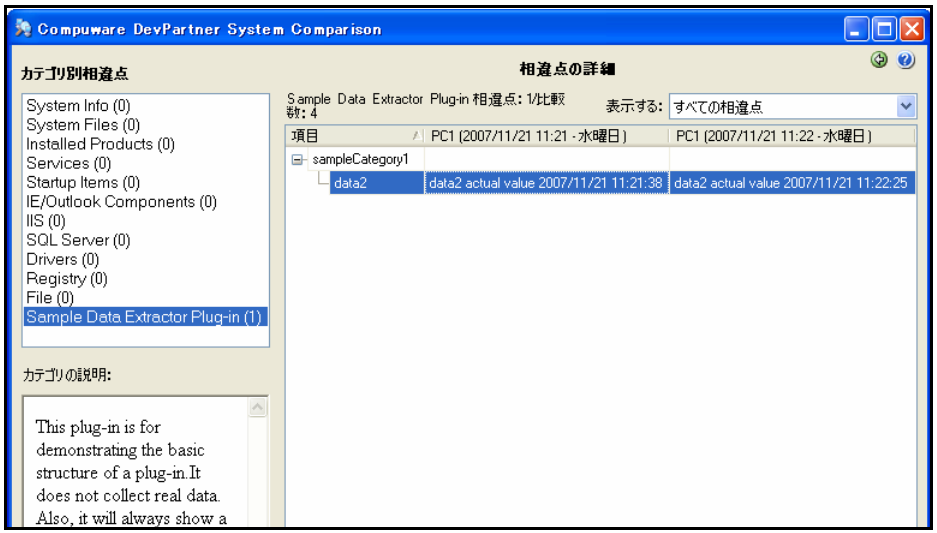


図 8-4.　　サンプル プラグインの結果ウィンドウ

プラグインの作成とテスト

プラグインのメカニズムを理解したら、カスタム プラグインを設計して、必要なデータを収集できます。

プラグインを設計するときは、収集するデータの階層に特に注意してください。必要な値を把握できるように階層を設計します。データ階層に不一致の値が見つかった場合、その階層の残りのデータは比較されません（以降のバージョンのプラグインでデータ階層を変更する方法については、「運用プラグインの変更」（308 ページ）を参照してください）。

TestDriver を使用して作成したプラグインを実行すると、問題解決が簡単になります。プラグインの出力が適切であることが確認できたら、System Comparison コマンドライン インターフェイスを使用してテストします。

1. プラグインを製品インストール ディレクトリの plugins サブディレクトリにコピーします（**TestDriver.exe** ファイルはプラグインのテスト用なので、コピーする必要はありません）。
2. プラグインがデータを収集する領域に相違点がある 2 台のコンピュータ上で、コマンド ライン プログラム（**<product dir>¥bin¥Compuware.Diff.CommandLine.exe**）を実行します。
3. System Comparison 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。相違点が確認できるはずです。
4. スナップショットを作成するときにプラグインで指定したデータが含まれるように、System Comparison サービスを再起動します。

一時ディレクトリ（正確な場所については、temp 環境変数を参照）の **DifferEvent.log** を確認し、発生した問題を解決します。プラグインが検出され、インスタンス化されると、イベントのログが出力されます。そのあと、データのロード、アンロード、または取得コールの間に発生したエラーでも、ログにイベントが生成されます。

また、IPluggableDataExtractor.GetData コールの ILoggable traceLogger パラメータによってログに出力したエラーも、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。

運用プラグインの変更

プラグインを配置したあとに、収集するデータを変更する場合があります。古いスナップショット ファイルを新しいバージョンのプラグインで作成したスナップショットと比較すると、収集したデータは一致しません。System Comparison ユーティリティは、この不一致を相違点と特定するため、混乱する可能性があります。

メジャーおよびマイナー バージョン番号を使用することで、不一致の処理方法を制御できます。プラグインのメジャー バージョン番号がスナップショット間で異なる場合、「プラグイン スキーマに互換性がありません」と報告されます。マイナー バージョンが異なる場合は、新しいデータのステータスが古いスナップショットで「不明」と示されます。

収集するデータを削除したり、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場合、メジャー バージョン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、一般的にはマイナー バージョンを変更するだけで十分です。

この仕組みを理解するには、SamplePlugin のバージョン番号（最初は 1.0）を変更して練習します。

```
public PluginSchemaVersion PluginVersion
{
get { return new PluginSchemaVersion( 1, 0 );}
}
```

メモ： プラグインの置換または削除が必要な場合、まず System Comparison サービスを停止し、System Comparison ユーザー インターフェイスを終了して、オペレーティング システムがファイルをロックしないようにします。

プラグイン スキーマの主要な要素

プラグイン スキーマを理解するには、スナップショットを確認します。スナップショットにはプラグインが収集したデータが含まれます。詳細については、**¥sdk¥Plugin**の**diff-plugin-schema.xsd**ファイルを参照してください。注釈を付けたXMLサンプルの抜粋を以下に示します。使用できる要素と属性の一部がわかります。

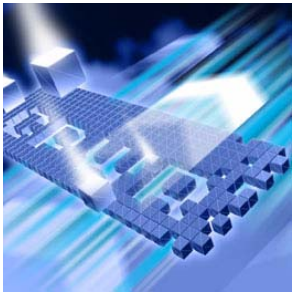
`<c`	最も外側のCategoryノードはプラグイン用です。
`n="MyApplication"`	プラグインの名前。このテキストはカテゴリ リストに表示されます（UIの左側）。
`descrip="text"`	このテキストは、カテゴリを選択したときにUIの左下に表示されます。
`schema="1">`	これを"1"に設定します。新しいバージョンのプラグインが以前のリリースと互換性がない場合、この値を変更します。
`<c`	ネストされているすべてのカテゴリがUIのメイン ウィンドウに表示されます。
`n="MyCategory"`	この名前はUIのメイン ウィンドウに表示され、比較に使用されます。
`missing="text"`	オプション。このカテゴリがもう一方のスナップショットに見つからない場合に表示するテキストです。たとえば、バージョン情報や「インストール済み」などのシンプルなテキストです。
`error="text">`	オプション。このテキストを指定すると、このカテゴリのデータを読み取り中にエラーが発生した場合に、このテキストがUIに表示されます。
`<s`	すべてのデータは文字列データです。
`n="MyData"`	この名前はUIのメイン ウィンドウに表示され、比較に使用されます。
`search="t1 t2"`	オプション。UIの「検索」リンクの用語を検索します。用語はGoogleに渡されます。
`error="text">`	オプション。このテキストを指定すると、このカテゴリのデータを読み取り中にエラーが発生した場合に、このテキストがUIに表示されます。
`Actual Data Value`	レジストリまたは他の設定のデータです。
`</S>`	
`</c>`	
`</c>`	

再配布可能なアセンブリについて

最新バージョンの **Compuware.Diff.PluginInterface.dll**、**Compuware.Diff.LoggableInterface.dll**、**Compuware.Diff.CollectorSchema.dll** 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今後のバージョンの System Comparison ユーティリティでも機能します。アセンブリに大きな変更が加えられた場合、バージョン番号が増えます。この場合、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 Enterprise Edition と TrackRecord



DevPartner Studio Enterprise Edition の概要

DevPartner Studio Enterprise Edition（EE）を使用すると、管理者は、プロジェクトで特定の品質レベルや運用ステータスなどの目標を達成する時期をより正確に予測できるようになります。DevPartner Studio EE を使用すると、プロジェクト管理者はソフトウェア プロジェクトを予定どおりに進めるために必要な具体的なプロジェクト詳細を把握でき、開発チームのメンバーは目標達成に必要なツールを利用できます。

DevPartner Studio EE は複数の既存ソフトウェア ソリューションの機能を組み合わせて統合し、新しい機能を提供します。Enterprise Edition にはこのマニュアルで説明する DevPartner の機能の他に、以下のコンポーネントも入っています。

TrackRecord	開発チームのための優れた変更要求管理、タスク管理、およびワークフロー サポート
Reconcile	ソフトウェア開発チームのための実用的な要件管理

DevPartner Studio EE では、以下の方法でプロジェクト データの取得、修正、表示、および追跡を行います。

- クリティカルパス プロジェクト データを解釈して理解するためのマイルストーン関連のサマリ
- 各社の開発プロセスに適した方法でデータを追跡するカスタマイズ可能なワークフロー
- Web インターフェイスによるプロジェクト情報へのリモート アクセス
- 重要なプロジェクト情報への変更を通知する電子メール

開発プロセス

ソフトウェア開発グループはそれぞれ独自のプロセスを定義します。プロセスとは、プロジェクトの発案および設計段階から、実装および納品段階までを実現するための一連の工程です。DevPartner Studio EE には、チームの現在のプロセスに合わせて社内開発手順の微調整を支援する機能があります。

プロセスの例は以下のとおりです。

- 要件の記述
- 体系的な変更管理
- 技術レビュー
- 品質保証プランニング
- 実装プランニング
- ソース コード自動管理
- 主要マイルストーンでの予測更新

プロセスを使用しないプロジェクトでは、以下の問題がたびたび発生します。

- テスト時のアプリケーションの再設計や再作成
- 統合の問題
- 開発ライフサイクルの後半での莫大な費用がかかる不具合修正
- 過大な要求の問題（「スラッシング」と呼ばれる）

プロセスが十分に管理されているプロジェクトでは、計画に沿ったプロジェクトの進行状況の信頼性が高くなります。また、プロセスによって開発チームの志気も高まります。50 社に行った調査によると、士気が高いと評定された開発者のうち、プロセス志向ではない会社で働いていた開発者は 20% だったのに対し、プロセスを重要視した会社で働いていた開発者は 60% でした。

DevPartner Studio EE は会社の既存のソフトウェア開発プロセスに適応し、必要に応じてチームによるプロセス強化を支援するツールを提供します。また、チームのワークフローを形式化し、メンバーにそのワークフローに対する責任を持たせ、プロセス全体を監視させる機能もあります。カスタマイズ可能なワークフローとエラーの自動検出機能を組み合わせると、ソフトウェアの品質が向上し、開発プロセスが合理化されます。

DevPartner Studio EE のソリューション

DevPartner Studio EE は、一般的にソフトウェア開発チームが直面する問題に対するソリューションを提供します。

- プロジェクト管理の改善
- コードの信頼性を高めることによる、ソフトウェア品質の向上
- 生産性の向上

プロジェクト管理の改善

プロジェクトを管理するために、以下の点について簡単に調べる機能があります。

- チームがすでに完了したタスク
- まだ完了していないタスク
- アプリケーションのコードの変動性
- アプリケーション テストの徹底性
- アプリケーションの信頼性

プロジェクト情報の動的な追跡

DevPartner Studio EE には、TrackRecord を使用して動的にプロジェクト情報を追跡する優れた機能があります。

ソフトウェア プロジェクトの計画には、多数のツールが存在します。これらのツールは、リソースの割り当て、スケジュール、クリティカルパス タスクなどの不可欠な情報の決定に役立ちます。DevPartner Studio EE 以前は、承認されたプロジェクト計画が静的なデータ ポイントになりました。実際のプロジェクトでは、スケジュールが変更されたり、プログラマがプロジェクトを離れて他のプロジェクトの大きな問題の処理に追われたり、納品状況が変更されたりすることがあります。プロジェクト プランニング ツールだけでは状況の変化に簡単に対処することはできませんが、Microsoft Project と TrackRecord の間に DevPartner Studio EE を接続することによって、スケジュールを動的に再計算できるようになりました。

ソフトウェア品質の向上

開発者とテスト エンジニアは、ソフトウェア開発サイクル全体を通して DevPartner を使用できます。

ユーザーが発見する前に問題を発見する

DevPartner Studio EE は他のソフトウェア プロジェクト強化ツールとは異なり、事前に問題を察知する系統立ったアプローチにより、バグからコードのボトルネックまで、プログラムの異常を検出し、修正できます。高品質の製品を開発するには、プログラムの問題を早期発見することが重要になります。DevPartner Studio EE のデバッグ機能は、開発サイクルを通じて、個別にも全体的にも開発者を支援します。このように DevPartner を使用することで、エラーの発見や修正を簡単に行い、開発時間を大

幅に短縮できます。また、レポートを簡単に作成できるため、開発者が不具合および機能レポートを入力する機会が増えます。

エラーの発見はプロセスのスタート地点にすぎません。エラーを修正するには、エラーの検出、記録、および再現を行い、修正の優先順位を決める必要があります。TrackRecord を使用すると、このプロセスの大部分が自動化されます。これにより開発者が多くの作業から解放されるので生産性が向上し、また多忙なスケジュールの中問題が未解決のまま残るのを防ぐこともできます。

生産性の向上

プロジェクトのマイルストーン（コード凍結や運用日付などの重要な日付）アプローチとして、未解決の不具合の数、コード変動率、チーム全体のコード カバレッジ統計値などのプロジェクト データを動的に表示することにより、チームのメンバー全員が目標達成度を把握することができます。

DevPartner Studio EE ユーザーは、それぞれプロジェクト データベースに固有の情報のビューを作成できます。

- 管理者はプロジェクトの全体像を把握し、重要なテストが実行されているかどうかを追跡して、品質をさらに厳しく管理できます。
- 開発者は、早急に対処が必要なタスクのリストの作成、優先度に従ったタスクの分類、コードでのエラー検出とパフォーマンス テストを行い、日常作業に集中することができます。
- テスターは、バグと既知の問題のステータスの追跡、カバレッジの実行で生成されたデータのマージ、テスト プランの実行、および毎日の作業の整理を行うことができます。
- ライターは、仕様書の発行時期、機能の実装時期、およびユーザー インターフェイスの変更時期を追跡できます。
- サポート コーディネータは、既知の不具合やテストされる構成などの情報をすばやく検索し、ユーザーの問題解決を支援できます。

このように、個々のユーザーが必要な情報だけをすばやく検索できるので、生産性が向上します。[Milestone Summary] などのビューには、この情報を表示するためのコンテキストがあります。

ソフトウェア開発プロジェクトは多様で、会社によってニーズも異なります。同じ会社内でも、進行中のプロジェクトに関して必要な情報は、部門ごとに違います。DevPartner Studio EE は、プロジェクトの設計、特に追跡する情報のタイプに柔軟性を持たせてこれらの要件を満たします。

DevPartner Studio EE にはあらかじめ組み込まれた多数のデータベース情報のビューがありますが、プロジェクト情報の保存と表示に対するユーザーの要求はそれぞれ異なります。DevPartner Studio EE では、レポート、フォーム、ワークフロー、プロジェクト、および情報のタイプをユーザーのニーズに合わせて柔軟にカスタマイズできます。

機能の概要

DevPartner Studio EEには、ソフトウェア開発プロジェクト情報を蓄積して共有するためのツールがあります。開発時のプロジェクトの追跡プロセスを容易にする豊富な機能が用意されています。

要件管理

アプリケーション開発プロジェクトで重要な最初の段階は、エンド ユーザーが求める要件を収集することです。次に、それらの要件を開発チームとテスト チームに正確に伝える必要があります。Reconcileは、プロジェクト要件の収集、整理、および配布を行います。

Reconcileを使用すると、Microsoft Wordをエディタとして使用し、要件を収集して絞り込むことができます。開発チームはReconcile Project Explorerを使用して要件の関係を確認できます。Reconcile要件とQADirectorの同期をとることによって、テスト プロシージャを自動作成し、テスト結果とテスト プランを関連付けることができます。

TrackRecordとReconcileの統合によって、不具合と問題をプロジェクト要件に関連付けることができるようになります。このような方法でReconcileとTrackRecordを使用すると、開発チームのメンバー全員が最新のプロジェクト目標を常に把握できます。

カバレッジ データのマージ

DevPartner Studio EEのカバレッジ機能によって、実行またはテストされたコンポーネントのコードの量に関する情報が生成されます。個々の開発者がそれぞれ異なるコンポーネントを担当する場合があるので、この個々のカバレッジ データはアプリケーション プロジェクト全体についてのデータを示すものではありません。プロジェクト全体のコードに対してどれだけのコードがテストされたかという割合を把握するには、各開発者のローカル カバレッジ レポートを他のカバレッジ結果とマージする必要があります。

DevPartner Studio EEでは、ビルド、構成、ユーザー、またはその他の基準に従ってカバレッジ データのセットをマージできます。

プロジェクト作業の追跡

ソフトウェア プロジェクトのさまざまなタスクとコンポーネントを追跡すると、複雑な問題に対処する際に役立ちます。チームのメンバーが特定のタスクに取り組んでいる場合、今ではなくあとで注意が必要な新しいタスクが発生することがあります。DevPartner Studio EEを使用すると、こうしたタスクを忘れないように記録して追跡できます。各開発者の作業を統合するには、あとで検討が必要な問題を詳しくまとめ、正確に記録しておく必要があります。

実行中のテストのレベル、発見されている不具合の数、およびカバレッジ作業の量を追跡すると、プロジェクト管理者は問題を事前に予測して回避することができます。DevPartner Studio EE と Microsoft Project 間の双方向通信により、スケジュール変更を自動化することもできます。

変更の自動通知

適切なタイミングで作業することは生産性を向上させます。たとえば、以下のような通知を設定できます。

- 新しく発見された優先度の高いバグの通知：管理者はタスクの優先度の変化に応じてリソースを割り当て直すことができます。
- 新しく割り当てられたタスクの通知：開発者がより効率的に予定を立てるのに役立ちます。

プロジェクト データへの変更をユーザーに通知するための主な方法として、動的に生成されるオンライン レポートが使用されますが、もう 1 つの方法として DevPartner Studio EE AutoAlert 機能があります。この機能を使用すると、追跡しているイベントが発生したときに 1 人または複数のユーザーに通知できます。AutoAlert では、基準を柔軟に定義することができるので、発生した変更内容に関係するリモート ユーザーや少数のユーザーに、その変更を通知することができます。

自動電子メール通知を受信する場合は、各ユーザーが特殊な DevPartner Studio EE 電子メール クエリを作成して通知基準を設定します。AutoAlert は、新しいアイテムが電子メール クエリに一致するかどうかを定期的にチェックして、DevPartner Studio EE データベースを監視します。

あるアイテムが入力または変更され、それがいずれかの電子メール クエリと一致すると、電子メール クエリの所有者に電子メール メッセージが自動送信されます。TrackRecord ソフトウェアの柔軟なクエリ エンジンを使用すると、AutoAlert によって、あらゆる基準に従って電子メール通知を受信できるようになります。

カスタマイズ可能なワークフロー

ソフトウェア開発チームには、それぞれ、あるタスクを確実に、通常は特定の順番で、完了させるための方法が必要です。たとえば QA では、修正が開発中のアプリケーションの残りの部分と統合されたと記録されるまで、修正済みの問題をテストすることはできません。DevPartner Studio EE では、必須ではありませんが、この方法で動作するワークフローを設定できます。

DevPartner Studio EE には、順番にワークフローを実装するメカニズムがあります。このワークフローでは、プロジェクト データのあるアイテムをワークフロー ライフサイクルの 1 つの段階から次の段階に移行させることができるユーザーを制限するように設計できます。ワークフローとその実施規則には、指定された条件下での特定の情報が必要になることがあります。これらの規則によって、プロジェクトで使用されるプロセスに各チーム メンバーが責任を持つようにすることができます。

Web 経由のリモート アクセス

開発チームのメンバーは、離れた場所で作業していても、プロジェクト データにアクセスできます。DevPartner Studio EE WebServer には、標準の Web 接続を介したリモート アクセス機能が備わっているので、重要なプロジェクト データの表示、入力、変更を行うことができます。

共有情報の集中保存

DevPartner Studio EE には、情報共有に使用するクライアント／サーバーベースの高機能リポジトリがあります。このリポジトリではオブジェクト指向データベースが使用されており、ActiveX（以前の OLE オートメーション）インターフェイスを介したプログラムによるアクセスが可能です。このリポジトリを使用することにより、各メンバーが別々に作業しながらもグループが連携して作業できる基本的なインフラが提供されます。

情報のタイプに基づいた拡張可能で柔軟なデータベース構造は、DevPartner Studio EE のレポジトリの中核を成し、その威力を発揮します。

TrackRecord と DevPartner Studio

TrackRecord は、ソフトウェア開発ツール DevPartner Studio Enterprise Edition スイートの一部です。これらのアプリケーションは、ソフトウェアの問題の検出、診断、解決に関する情報を自動的に生成し、保存します。

開発者は TrackRecord を使って、この情報をマイルストーンの日付などその他のプロジェクト情報と同時にキャプチャし、迅速で整合性のとれた問題解決に役立てることができます。

メモ： TrackRecord と DevPartner Studio の統合はバージョンによって異なります。各ツールを別の時期に購入した場合は、DevPartner Studio または TrackRecord ソフトウェアのアップグレードが必要な場合があります。

DevPartner Studio の TrackRecord との通信

DevPartner Studio では、ツールバー ボタンとメニューの選択によって、TrackRecord データベースにバグ レポートを送信できます。

DevPartner ツールバー ボタン

DevPartner ツールバー ボタンを使用して、バグを入力できます。これらのボタンをクリックすると、バグ フォームが開き、DevPartner Studio データベースに情報を入力できるようになります。

バグの送信

バグを送信するには、まずDevPartner Studioのデバッグ表示で項目を強調表示します。

DevPartnerからのバグの入力

DevPartnerからバグを入力するには、以下の操作を行います。

1 **DevPartner**メニューまたはツールバーで**[バグの提出]**を選択します。
または、DevPartnerの機能ツールバーで[バグの提出]ボタンを選択します。

TrackRecordにより、空の**バグ** フォームまたは該当データが一部入力された**バグ** フォームが開かれます。

2 バグ レポートにその他の必要な情報を入力します。

3 **[Save]**をクリックしてから、**[Close]**をクリックします。

TrackRecordとDevPartner Studioのカバレッジ分析

DevPartnerカバレッジ分析のユーザーは、自分のワーク スペース内に蓄積したセッション ファイルをマージできます。マージされたセッションは、時間の経過と共に開発者のコードがどのくらいテストされたかを示します。

DevPartner StudioとTrackRecordを使用して、ユーザー間、環境間でカバレッジ セッションをマージしてフィルタできます。同じアプリケーションを担当しているすべての開発者からのカバレッジ セッションをマージすると、管理者やテスト コーディネータは、アプリケーションの全コード ベースがテストプログラムによってどのくらい検証されたかを調べることができます。

DevPartnerカバレッジ分析の使用方法については、カバレッジ分析のマニュアルとオンライン ヘルプを参照してください。

カバレッジ セッションをマージするには、まずカバレッジ マージ セットを作成し、次にそのセッションをマージします。開発者は通常、マージする必要があるカバレッジ セッションと除外するセッションを選択します。マージするセッションを識別する基準に、以下の項目を含めることができます。

- アプリケーション コンポーネント
- 日付
- メモリ
- マイルストーン
- オペレーティング システム
- 開発者
- プロジェクト

これらのいずれかの項目を特定の値、任意の値、または指定した値を除く値と照合できます。

カバレッジ マージ処理の基準の作成

カバレッジ マージ処理の基準を作成するには、以下の操作を行います。

1. TrackRecord で、**[Tools]** メニューから **[Merge Coverage Sessions]** を選択します。
2. 左側のリスト ボックスからターゲットを選択します。
3. 右側のリスト ボックスから一致基準を選択します。
4. ステップ 2 で [is equal to] を選択した場合は、下のリスト ボックスから値を選択します。

   たとえば、上部の 2 つのリストで [Operating System] と [is equal to] を選択した場合は、下部のリストから [Windows 98] などの値を選択します。
5. **[Add]** をクリックします。
6. **[Next]** をクリックすると、基準に一致したセッションが表示されます。

カバレッジ セッションのマージ

カバレッジ セッションをマージするには、以下の操作を行います。

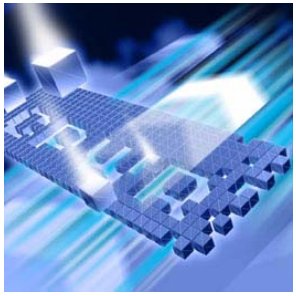
1. セッションの隣にあるチェック ボックスをクリックしてオンまたはオフにします。

   オンにすると、そのセッションは選択されている他のファイルとマージされます。オフにすると、そのセッションはマージされません。
2. **[Merge]** をクリックします。

   DevPartner カバレッジ分析のメイン ダイアログ ボックスが開き、単体テストで検証された行と関数の量に関する棒グラフと統計が表示されます。

付録 B

DevPartner Studio でサポートされるプロジェクト タイプ



この章には、上記の DevPartner Studio 機能がサポートするプロジェクトのタイプを示す表があります。

サポートされるプロジェクト タイプ

DevPartner Studio は、さまざまな Visual Studio 統合開発環境（IDE）、マネージおよびアンマネージのプロジェクト タイプ、プログラミング言語を含め、多くのソフトウェア開発環境で動作します。

表 B-1　サポートされる Visual Studio のバージョンと言語

統合開発環境	マネージまたはアンマネージ	言語
Visual Studio 2005（VS2005）	マネージ	Visual Basic 2005 Visual C++ 2005 Visual C# 2005 Visual J# 2005
	アンマネージ	Visual C++ 2005
Visual Studio 2003（VS2003）	マネージ	Visual Basic .NET Visual C++ .NET Visual C# .NET Visual J# .NET
	アンマネージ	Visual C++ .NET
Visual Studio 6.0 Visual Basic（VS 6.0 VB）	アンマネージ	Visual Basic 6.0
Visual Studio 6.0 C++（VS 6.0 C++）	アンマネージ	Visual C++ 6.0

以降のページでは、各 DevPartner Studio 機能でサポートされる IDE、プロジェクトタイプ、言語について説明します。

エラー検出でサポートされるプロジェクト タイプ

アプリケーション プロジェクトはx86実行ファイルにビルドされます。DevPartnerエラー検出は以下のプロジェクト タイプをサポートします。

表 B-2　エラー検出がサポートするマネージ プロジェクト タイプ

Visual Studioのバージョン	アプリケーションのプロジェクト タイプ	サポートされる言語
VS2003	コンソール アプリケーション（.NET） Windows フォーム アプリケーション（.NET） Windowsサービス（.NET）	C++
	コンソール アプリケーション Windowsサービス	C#、J#、 VB
VS2003、VS2005	MFCアプリケーション Win32コンソール アプリケーション Win32プロジェクト	C++
	Windowsアプリケーション	C#、J#、 VB
VS2005	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++
	Calculator Starter Kit	J#
	Crystal Reportsアプリケーション Windowsコントロール ライブラリ	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 デバイスのプロジェクト タイプ（ソリューション プラットフォームがWin32に設定されているスマート デバイスのプロジェクト タイプ）は、DevPartnerエラー検出でサポートするには、エミュレータではなく開発マシンで実行する必要があります。

ホストするプロジェクトはx86 DLLにビルドされます。このプロジェクトをテストするにはアプリケーションでホストする必要があります。別の実行可能ファイル内でホストされる場合のみ、DevPartnerエラー検出は以下のプロジェクト タイプをサポートします。

表 B-3　プロジェクトが別の実行ファイル可能内でホストされる場合のサポート

Visual Studio のバージョン	別の実行可能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
VS2003	クラス ライブラリ（.NET）	C++
	クラス ライブラリ	C#、J#、VB
VS2003、VS2005	ATL プロジェクト 拡張ストアド プロシージャ DLL MFC Active X コントロール MFC DLL MFC ISAPI 拡張 DLL	C++
	Web コントロール ライブラリ	C#、J#、VB
	Windows コントロール ライブラリ	C++、C#、J#、VB
VS2005	ATL プロジェクト ATL Server プロジェクト ATL スマート デバイス プロジェクト MFC スマート デバイスの 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++
	クラス ライブラリ	C++、C#、J#、VB
ATL ー Active Template Library MFC ー Microsoft Foundation Class		

コード レビューがサポートするプロジェクト タイプ

以下の表に、DevPartner Studio コード レビュー機能がサポートするプロジェクト タイプを示します。

表 B-4　コード レビューがサポートするマネージ プロジェクト タイプ

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
VS2003、VS2005	ASP.NET Web アプリケーション ASP.NET Web サービス クラス ライブラリ コンソール 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
VS2005	ASP.NET Web サイト Crystal Reports アプリケーション	

カバレッジ分析とパフォーマンス分析がサポートするプロジェクト タイプ

以下の表に、DevPartner Studio カバレッジ分析機能とパフォーマンス分析機能がサポートするプロジェクト タイプを示します。

表 B-5　カバレッジ分析とパフォーマンス分析がサポートするマネージ プロジェクト タイプ

Visual Studio のバージョン	マネージ プロジェクト タイプ	サポートされる言語
VS2003	コンソール アプリケーション（.NET） クラス ライブラリ（.NET） Windows コントロール ライブラリ（.NET） Windows フォーム アプリケーション（.NET） Windows サービス（.NET）	C++
	ASP.NET モバイル Web アプリケーション ASP.NET Web 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 Visual Studio .NET アドイン Windows コントロール ライブラリ Windows サービス	C#、J#、VB
	ASP.NET Web サービス	C++、C#、J#、 VB
VS2003、VS2005	Shared アドイン Web コントロール ライブラリ Windows アプリケーション	C#、J#、VB

表 B-5　カバレッジ分析とパフォーマンス分析がサポートするマネージ プロジェクト タイプ

Visual Studio のバージョン	マネージ プロジェクト タイプ	サポートされる言語
VS2005	ASP.NET Web サービス CLR コンソール 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	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	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 Windows コントロール ライブラリ	C#、J#、VB
	クラス ライブラリ Windows サービス	C++、C#、J#、 VB

表 B-6　カバレッジ分析とパフォーマンス分析がサポートするアンマネージ プロジェクト タイプ

Visual Studio のバージョン	アンマネージ プロジェクト タイプ	サポートされる言語
C++ 6.0	ATL COM AppWizard ISAPI 拡張ウィザード MFC ActiveX ControlWizard MFC AppWizard（DLL） MFC AppWizard（EXE） Win32 アプリケーション Win32 ダイナミック リンク ライブラリ Win32 静的ライブラリ	C++ 6.0
VB 6.0	ActiveX コントロール ActiveX DLL ActiveX EXE データ プロジェクト 標準の EXE VB アプリケーション ウィザード VB Pro Edition のコントロール	VB 6.0
VS2003	拡張ストアド プロシージャ DLL MFC ISAPI 拡張 DLL	C++
	Win32 コンソール プロジェクト	C++
C++ 6.0、VS2005	Win32 コンソール アプリケーション	C++
VS2003、VS2005	ATL プロジェクト ATL Server プロジェクト ATL Server Web サービス MFC ActiveX コントロール MFC アプリケーション MFC DLL Win32 プロジェクト	C++
ATL ー Active Template Library MFC ー Microsoft Foundation Class		

メモ：　DevPartner Studio カバレッジ分析とパフォーマンス分析は、従来の ASP とクライアント側の Web スクリプトの両方で VBScript と JScript をサポートしています。

メモリ分析がサポートするプロジェクト タイプ

以下の表に、DevPartner Studio メモリ分析機能がサポートするプロジェクト タイプを示します。

表 B-7　メモリ分析がサポートするマネージ プロジェクト タイプ

Visual Studio のバージョン	マネージ プロジェクト タイプ	サポートされる言語
VS2003	コンソール アプリケーション（.NET） クラス ライブラリ（.NET） Windows コントロール ライブラリ（.NET） Windows フォーム アプリケーション（.NET） Windows サービス（.NET）	C++
	ASP.NET モバイル Web アプリケーション ASP.NET Web アプリケーション クラス ライブラリ Visual Studio .NET アドイン Windows コントロール ライブラリ Windows サービス	C#、J#、 VB
	ASP.NET Web サービス	C++、C#、 J#、 VB
VS2003、VS2005	Shared アドイン Web コントロール ライブラリ Windows アプリケーション	C#、J#、VB
VS2005	ASP.NET Web サービス CLR コンソール 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	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	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 Windows コントロール ライブラリ	C#、J#、VB
	クラス ライブラリ Windows サービス	C++、C#、 J#、 VB

パフォーマンス エキスパートがサポートするプロジェクト タイプ

以下の表に、DevPartner Studio パフォーマンス エキスパート機能がサポートするプロジェクト タイプを示します。

表 B-8　パフォーマンス エキスパートがサポートするマネージ プロジェクト タイプ

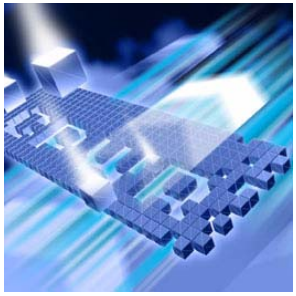
Visual Studio のバージョン	マネージ プロジェクト タイプ	サポートされる言語
VS2003	クラス ライブラリ（.NET） コンソール アプリケーション（.NET） Windows コントロール ライブラリ（.NET） Windows フォーム アプリケーション（.NET） Windows サービス（.NET）	C++
	ASP.NET モバイル Web アプリケーション ASP.NET Web アプリケーション クラス ライブラリ コンソール アプリケーション Visual Studio .NET アドイン Windows コントロール ライブラリ Windows サービス	C#、J#、VB
	ASP.NET Web サービス	C++、C#、J#、 VB
VS2003、VS2005	Shared アドイン Web コントロール ライブラリ Windows アプリケーション	C#、J#、VB

表 B-8　パフォーマンス エキスパートがサポートするマネージ プロジェクト タイプ

Visual Studio のバージョン	マネージ プロジェクト タイプ	サポートされる言語
VS2005	ASP.NET Web サービス CLR コンソール 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	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	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 Windows コントロール ライブラリ	C#、J#、VB
	クラス ライブラリ Windows サービス	C++、C#、 J#、 VB

付録 C

コマンド ラインからの分析の開始



- DPAnalysis.exe の概要
- コマンド ラインからの DPAnalysis.exe の実行
- DPAnalysis.exe と XML 構成ファイルの使用
- リモート コンピュータからの分析データの収集

この付録には、カバレッジ分析、メモリ分析、パフォーマンス分析、パフォーマンス エキスパートに使用できる **DPAnalysis.exe** コマンド ライン ツールに関する情報が記載されています。

DPAnalysis.exe の概要

DPAnalysis.exe を使用すると、Visual Studio でプログラムを実行中に分析データを収集できるだけでなく、Visual Studio を実行しなくてもプロファイル情報を収集できます。**DPAnalysis.exe** をオプション スイッチと組み合わせたり、XML 構成ファイルを指定したりして、アプリケーション データを収集します。

コマンド ラインからの DPAnalysis.exe の実行

コマンド ラインから、分析セッションを指示するスイッチやXML構成ファイルを指定して**DPAnalysis.exe**を使用できます。以下のコマンド ライン例では、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実行し、セッション ファイル（**.dpprf**）を**c:¥output**ディレクトリに保存します。

```
DPAnalysis.exe /perf /output c:¥output¥target.dpprf /p target.exe
```

コマンド ラインから**DPAnalysis.exe**を使用すると、データ収集を有効にし、単一のプロセスまたはサービスを実行できます。**DPAnalysis.exe**を使用して複数のプロセスを実行するには、「DPAnalysis.exe と XML 構成ファイルの使用」（337 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（Visual C++ 6.0 アプリケーションや Visual Basic 6.0 アプリケーションなど）をインストゥルメントできません。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 コードのデータの収集」（134 ページ）、パフォーマンス分析については「アンマネージ コードからのデータ収集」（222 ページ）を参照してください。

コマンド ラインから DevPartner Studio の 4 つの分析ツールを実行するには、以下の構文と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] [/N] [/F]
[/NO_MACH5 /NM_METHOD_GRANULARITY /EXCLUDE_SYSTEM_DLLS
/NM_ALLOW_INLINING /NO_OLEHOOKS
/NM_TRACK_SYSTEM_OBJECTS] {/P|/S} target.exe [target arguments]
```

分析タイプのスイッチ

実行時の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 DevPartner カバレッジ分析に設定します。
/Exp[ert]	分析タイプを DevPartner 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 DevPartner メモリ分析に設定します。
/Perf[ormance]	分析タイプを DevPartner パフォーマンス分析に設定します。

データ収集のスイッチ

指定し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、ターゲットを実行することはできません。これらのスイッチはオプションです。

/E[nable]	指定し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]	指定し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ルを無効にするまで、指定したプロセスを実行するたびにプロファイルが行われます。

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 ファイルの出力ディレクトリ、またはディレクトリと名前を指定します。
/W[orkingDir]	ターゲットのプロセスまたはサービスの作業ディレクトリを指定します。
/PROJ_DIR	DevPartner Studio プロジェクトのディレクトリを指定します。プレイリストなどの場所を特定するときに使用されます。
/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 を指定して DPAnalysis.exe を使用すると、process1 の開始直後に process2 が起動されます。 以下に例を示します。 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 スイッチを省略して DPAnalysis.exe を使用すると、process1 が終了してから process2 が実行されます（デフォルトの動作）。
/N[ewconsole]	新しいコマンド ウィンドウでプロセスを実行します。 DPAnalysis.exe を使用してキーボードの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]	[DevPartner 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していないアンマネージ コード アプリケーションについて、カバレッジ分析またはパフォーマンス分析のプロファイルを強制的に実行します。

引用符付きのパスと /O[utput] スイッチ

出力（/o）スイッチのパラメータとして引用符付きのパスを使用し、ファイル名を含めない場合、以下に示すいずれかの方法でパスを終了する必要があります。

/o "c:¥test directory"	引用符で終了します。
/o "c:¥test directory¥"	円マークに続けてピリオドを指定して終了します。
/o "c:¥test directory/"	フォワード スラッシュで終了します。

分析オプション

これらのスイッチはオプションです。

/NO_MACH5	他のスレッドで費やされた時間の除外を無効にします。
/NM_METHOD_GRANULARITY	データ収集の粒度をメソッドレベルに設定します。行レベルがデフォルトです（パフォーマンス分析のみ）。
/EXCLUDE_SYSTEM_DLLS	システム dll の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 オブジェクトの割り当てを追跡します（メモリ分析のみ）。

ターゲット スイッチ

必須。1つだけ選択します。プロセスまたはサービスとして、ターゲットを指定します。ターゲットの名前またはパスのあとに指定するすべての引数は、ターゲットに対する引数です。

/P[rocess]	ターゲット プロセスを指定します（プロセスに対する引数が続きます）。
/S[ervice]	ターゲット サービスを指定します（サービスに対する引数が続きます）。
/C[onfig]	構成ファイルとパスを指定します。

DPAnalysis.exe と XML 構成ファイルの使用

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、**/config** スイッチを付け、そのターゲットとして正しい構造の XML 構成ファイルを指定して、コマンドラインから **DPAnalysis.exe** を実行します。次に例を示します。

```
DPAnalysis.exe /config c:¥temp¥configuration_file.xml
```

構成ファイルを使用すると、複数のプロセスやサービスをプロファイルおよび管理できます。複数のプロセスをプロファイルする機能は、Web アプリケーションを分析するときに特に役立ちます。

DPAnalysis.exe を使用してセッションを開始すると、**DPAnalysis.exe** を呼び出したシステム上のプロファイル対象プロセスごとにセッション コントロール ツールバーが起動します。ツールバーの適切なインスタンスを使用して各プロセスのセッション コントロール アクションを実行します。

構成ファイルの例を以下に示します。

```
<?xml version="1.0" ?>
<ProductConfiguration xmlns="http://www.compuware.com/products">
  <RuntimeAnalysis Type="Performance"
   MaximumSessionDuration="1000"/>
  <Targets RunInParallel="true">
    <Process CollectData="true" Spawn="true"
     NoWaitForCompletion="true">
      <AnalysisOptions NO_MACH5="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 構成ファイルの要素

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" />
```

属性

なし

Type

必須。使用できる選択肢は、**Performance**、**Coverage**、**Memory**、または **Expert** です。これらの選択肢で、リストするすべてのターゲットに対して分析タイプを指定します。

MaximumSessionDuration

オプション。省略した場合、デフォルトはありません。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"** を指定し、（サービスに **RestartAtEndOfRun="true"** を使用して）サービスのプロファイルを開始すると、60 秒後に **DPAnalysis.exe** はサービスを停止し、サービスを再起動します。

要素の情報

オカレンス数	1
親要素	ProductConfiguration
内容	**なし**

注意

分析のタイプと最長のセッション時間を定義します。

例

ProductConfiguration タグに続けて **RuntimeAnalysis** を使用する例を以下に示します。この例では、**Type** 属性にパフォーマンス分析を指定し、最大 1000 秒の時間を指定しています。

```
<?xml version="1.0" ?>
<ProductConfiguration xmlns="http://www.compuware.com/products">
<RuntimeAnalysis Type="Performance"
MaximumSessionDuration="1000"/>
```

Targets 要素

```
<Targets RunInParallel="true or false">
    ...
  </Targets>
```

属性

RunInParallel

オプション。**true** または **false** を指定します。省略するとデフォルトは **true** です。複数のターゲットを指定する場合、ターゲットの実行方法を定義します。**RunInParallel** が **true** の場合、**DPAnalysis.exe** はターゲットのプロセスとサービスを順番に続けて開始するので、複数のターゲットが同時に（並行して）実行されます。それ以外の場合、**DPAnalysis.exe** は、前のターゲットを起動して終了してから、次のターゲットを起動するので、ターゲットは一度に 1 つずつ（順番に）実行されます。

要素の情報

オカレンス数	1
親要素	RuntimeAnalysis
内容	Process、Service

注意

必須。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

Targets 要素を使用し、1 つの **<Service>** 要素と 2 つの **<Process>** 要素を指定する例を以下に示します。この例ではターゲットが並行して実行されるように、**RunInParallel** が **true** になっていることに注意してください。

```
<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_MACH5="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_MACH5="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ルを無効にするまで、指定したプロセスを実行するたびにプロファイルが行われます。

CollectData

オプション。**true** または **false** を指定します。省略するとデフォルトは **true** です。ターゲット プロセスについて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。**true** または **false** を指定します。省略するとデフォルトは **true** です。**DPAnalysis.exe** が指定し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** の場合、**true** に設定しないでください。DevPartner から 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。ターゲットの Web ページを開くことで ASP.NET ワーカー プロセスを起動します。

NoWaitForCompletion

オプション。**true** または **false** を指定します。省略するとデフォルトは **false** です。デフォルトでは、プロセスが完了するまで待機します。**true** に設定すると、ターゲットが実行を開始するまでしかは待機しません。**DPAnalysis.exe** は、（**Host** 要素を使用した）リモート コンピュータ上のプロセスを待機しません。

RuntimeAnalysis 要素の **MaximumSessionDuration** 属性は、**NoWaitForCompletion** を上書きします。

NewConsole

オプション。**true** または **false** を指定します。省略するとデフォルトは **false** です。**DPAnalysis.exe** は新しいコンソール ウィンドウでターゲットを実行します。デフォルトでは、**DPAnalysis.exe** コマンド ラインを入力した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の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。**true** または **false** を指定します。省略するとデフォルトは **false** です。**false** を明示的に指定するまで、ターゲットは実行されるたびにプロファイルされます。

要素の情報

オカレンス数	**1つまたは複数**
親要素	Target
内容	AnalysisOptions、Path、Arguments、WorkingDirectory、ExcludeImages

注意

ターゲットの実行可能ファイルを指定します。

例

Process の使用例を以下に示します。**AnalysisOptions**、**Path**、**Arguments**、**WorkingDirectory** の各タグも使用します。

```
<Targets RunInParallel="true">
<Process CollectData="true" Spawn="true"
 NoWaitForCompletion="true" NewConsole="true">
  <AnalysisOptions NO_MACH5="1" NM_METHOD_GRANULARITY="1"
   SESSION_DIR="c:¥MyDir" />
  <Path>ClientApp.exe</Path>
  <Arguments>/arg1 /agr2 /arg3</Arguments>
  <WorkingDirectory>c:¥temp</WorkingDirectory>
</Process>
</Targets>
```

AnalysisOptions 要素

AnalysisOptions と一緒に使用する属性は、実行する分析セッションの種類によって変わります。この説明の末尾にある表を参照してください。**DPAnalysis.exe** は分析の種類が一致しない属性を無視します。

```
<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_MACH5 = "1"
        FORCE_PROFILING = "1"
/>
```

属性

SESSION_DIR

オプション。カバレッジ分析、メモリ分析、パフォーマンス分析、パフォーマンス エキスパートで使用します。プロファイル対象のターゲットによって生成されたセッション ファイルの保存ディレクトリを指定します。この属性を指定しない場合、結果のセッション ファイルはユーザーの My Documents または Documents ディレクトリに格納されます。**SESSION_DIR** と **SESSION_FILENAME** のどちらも指定しない場合、**DPAnalysis.exe** はセッションの終了時に保存場所の入力を求めます。

SESSION_FILENAME

オプション。カバレッジ分析、メモリ分析、パフォーマンス分析、パフォーマンス エキスパートで使用します。このターゲットに対して生成されたセッション ファイルのセッション名を指定します。この属性を指定しない場合、**DPAnalysis.exe** はターゲットのイメージ名と数字を組み合わせて（**iexplore1.dpprf** など）、一意な名前を作成します。名前は指定してもディレクトリは指定しない場合、ファイルはユーザーの **My Documents** ディレクトリに格納されます。**SESSION_FILENAME** と **SESSION_DIR** のどちらも指定しない場合、**DPAnalysis.exe** はセッションの終了時に保存場所の入力を求めます。

NM_METHOD_GRANULARITY

オプション。パフォーマンス分析で使用し、データ収集の粒度をメソッド レベルに設定します（デフォルトは行レベル）。この属性を設定するには、**1** の値を指定します。この属性を省略すると、無効になります。

EXCLUDE_SYSTEM_DLLS

オプション。パフォーマンス分析で使用し、システム イメージを除外します。この属性を設定するには、1 の値を指定します。この属性を省略すると、無効になります。

NM_ALLOW_INLINING

オプション。カバレッジ分析とパフォーマンス分析で使用し、分析の詳細レベルを指定します。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。[インライン関数をインストゥルメントする]プロパティと同等です。インライン関数をインストゥルメントするには、1 の値を指定します。この属性を省略すると、無効になります。

NO_OLEHOOKS

オプション。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アクティブにします。この属性を設定するには、1 の値を指定します。この属性を省略すると、無効になります。

NM_TRACK_SYSTEM_OBJECTS

オプション。メモリ分析で使用し、割り当て済みオブジェクトを追跡するときに、システムまたはサードパーティのオブジェクト割り当てを無視します。この属性を設定するには、1 の値を指定します。この属性を省略すると、無効になります。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をアプリケーションが使用するときに割り当てられたメモリ割り当てを確認できます。

NO_MACH5

オプション。パフォーマンス分析とパフォーマンス エキスパートで使用し、他の実行アプリケーションのスレッドに使用された時間を除外します。この属性を設定するには、1 の値を指定します。この属性を省略すると、無効になります。

FORCE_PROFILING

オプション。カバレッジ分析とパフォーマンス分析で使用し、マネージ コードまたは DevPartner ネイティブ C/C++ インストゥルメンテーションを使用せずに作成されたアプリケーションのプロファイルを強制します。この属性を設定するには、1 の値を指定します。この属性を省略すると、無効になります。

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			○	
EXCLUDE_SYSTEM_DLLS			○	
NM_ALLOW_INLINING	○		○	
NO_OLEHOOKS			○	
NM_TRACK_SYSTEM_OBJECTS		○		
NO_MACH5			○	○
FORCE_PROFILING	○		○	

要素の情報

オカレンス数	Process または Service ごとに 1 または 0
親要素	Process、Service
内容	なし

注意

オプション。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

Service 内で **AnalysisOptions** を使用する例を以下に示します。

```
<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

要素の情報

オカレンス数	1
親要素	Process
内容	実行可能ファイルへのパス

注意

必須。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ディレクトリにある場合は、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

Process 要素内で **Path** を使用する例を以下に示します。

```
<Process CollectData="true">
  <Path>ClientApp.exe</Path>
</Process>
```

Arguments 要素

```
<Arguments>/arg1 /arg2 /arg3</Arguments>
```

属性

なし

要素の情報

オカレンス数	Process または Service ごとに 0 または 1
親要素	Process、Service
内容	なし

注意

オプション。省略した場合のデフォルトはありません。ターゲットの **process** または **service** に渡す属性。

例

Process 要素内で **Arguments** を使用する例を以下に示します。

```
<Process CollectData="true">
  <Arguments>/arg1 /agr2 /arg3</Arguments>
</Process>
```

WorkingDirectory 要素

```
<WorkingDirectory> c:¥MyWorkingDir </WorkingDirectory>
```

属性

なし

要素の情報

オカレンス数	Process 要素または Service 要素ごとに 1
親要素	Process、Service
内容	ターゲット ディレクトリへのパス

注意

オプション。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 サービスの作業ディレクトリを設定します。

例

親の **Process** 要素内にネストして **WorkingDirectory** を使用する例を以下に示します。

```
<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

要素の情報

オカレンス数	Process または Service ごとに 0 または 1
親要素	Process、Service
内容	Image

注意

オプション。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 サービスでロードされた場合に、プロファイルしない 1 つ以上のイメージ（上限なし）を定義します。

例

Process 要素内で **ExcludeImages** を使用する例を以下に示します。**Image** 要素は **ExcludeImages** 内に含まれ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

なし

CollectData

オプション。**true** または **false** を指定します。省略するとデフォルトは **true** です。ターゲット サービスについて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。**true** または **false** を指定します。省略するとデフォルトは **true** です。**DPAnalysis.exe** が指定したターゲットを開始するかどうかを指定します。**false** に設定すると、このターゲットに対するプロファイルは有効になりますが、ターゲットは開始されません。何らかの手段で次にサービスが開始されたときにプロファイルが開始されます。

RestartIfRunning

オプション。**true** または **false** を指定します。省略するとデフォルトは **false** です。**RestartIfRunning** を **true** に設定すると、指定したサービスがホスト コンピュータで実行されている場合、**DPAnalysis.exe** はそのサービスの再起動を試行します。

RestartAtEndOfRun

オプション。**true** または **false** を指定します。省略するとデフォルトは **false** です。**true** を指定すると、実行の終了時に **DPAnalysis.exe** は（セッション ファイルを生成して）サービスの再起動を試行します。

RepeatInjection

オプション。**true** または **false** を指定します。省略するとデフォルトは **false** です。**false** を明示的に指定するまで、ターゲットは実行されるたびにプロファイルされます。

要素の情報

オカレンス数	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。
親要素	Targets
内容	AnalysisOptions、Path、Arguments、Working Directory、ExcludeImages、Name、Host

注意

ターゲット サービスを指定します。

例

Targets 要素内で **Service** を使用する例を以下に示します。

```
<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

要素の情報

オカレンス数	1
親要素	Service
内容	サービス名

注意

必須。サービス コントロール マネージャに登録されているサービスの名前。これは、NET START コマンドを使用するときに使用する名前と同じです。

例

Service 内で **Name** を使用する例を以下に示します。

```
<Service CollectData="true">
  <Name>ServiceApp</Name>
</Service>
```

Host 要素

```
<Host>hostmachine</Host>
```

属性

なし

要素の情報

オカレンス数	Process または Service ごとに 0 または 1
親要素	Process、Service
内容	**ホスト コンピュータの名前**

注意

オプション。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 マシンを設定します。

例

Service 内で **Host** を使用する例を以下に示します。この例には必須の **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 構成ファイルを使用した Web アプリケーションのプロファイル

一般的に、Web プロファイルに関係があるプロセスは、ブラウザ、Web サーバー、ASP.NET ワーカー プロセスの 3 つです。これら 3 つのエントリは 1 つの構成ファイルに含めることができます。ブラウザと ASP.NET プロセスは **<Process>** 要素内に指定します。Web サーバーは **<Service>** 要素内で指定し、**<Name>** 要素でサービス名を識別します。IIS の場合、これは **iisadmin** になります。

次に例を示します。

```
<?xml version="1.0" ?>
<ProductConfiguration xmlns="http://www.compuware.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は以下を実行します。

- **remotemachine**上の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 コンピュータ上の**inetinfo.exe**（**iisadmin**）のデータ収集を有効にし、プロファイルが開始されるように再起動します。
- リモート コンピュータ上のWebページで指示されたローカル コンピュータに対して、ローカル ブラウザを開きます。**aspnet_wp.exe**がリモート コンピュータで実行され、プロファイルが開始されます。

ローカル コンピュータのブラウザを閉じると、(**aspnet_wp**を停止して）リモート コンピュータ上のIISはリモート コンピュータで再起動されます。セッション ファイルは自動的に各保存ディレクトリに保存されます。必要に応じて、リモート コンピュータ上にある既存のマッピング済みドライブを使用して、セッション ファイルをプロファイルを開始したコンピュータに保存します。この例の**<Process>**要素と**<Service>**要素の**z:¥**ドライブに従います。

サンプル構成ファイル

DevPartner Studio のインストールには、これらのサンプルと読み取り専用の構成ファイルが含まれます。これらをモデルとして使用してカスタムの構成ファイルを作成します。

```
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では、以下のディレクトリにファイルが格納されます。

```
<install drive>:¥Program Files¥Compuware¥DevPartner Studio
¥Analysis¥SampleConfigs¥
```

メモ： **DPAnalysis.exe** ではアンマネージ コード（Visual C++ 6.0 アプリケーションや Visual Basic 6.0 アプリケーションなど）をインストゥルメントできません。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 コードのデータの収集」（134 ページ）、パフォーマンス分析については「アンマネージ コードからのデータ収集」（222 ページ）を参照してください。

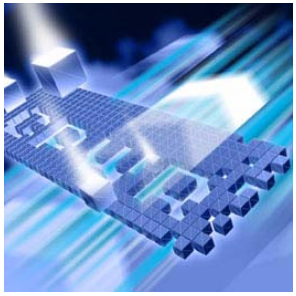
リモート コンピュータからの分析データの収集

DPAnalysis.exe を使用してリモート コンピュータ上で実行され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 システム上のアプリケーションを実行する場合、コマンド ラインまたは XML 構成ファイルを使用して、セッション ファイルを保存するためのディレクトリとファイル名を指定します。
- プロファイルを開始するローカル（クライアント）コンピュータへの既存のマッピング済みディレクトリなど、書き込み許可を持っている任意のディレクトリを指定できます。
- ディレクトリとファイル名を指定しない場合、リモート コンピュータで [ファイルの保存] ダイアログが表示されます。このダイアログを使用するには、コンピュータに対する物理的なアクセス権、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。[ファイルの保存] ダイアログのデフォルト ディレクトリは、アクティブなユーザー アカウントの **My Documents** ディレクトリまたは **Documents** ディレクトリです。

付録 D

分析のセッション コントロール



この付録には、DevPartner のカバレッジ分析、メモリ分析、パフォーマンス分析、パフォーマンス エキスパートに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 に関する情報が記載されています。

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、ルールとアクションのセットを作成し、アプリケーションの実行時に DevPartner で収集するデータを制御できます。このルールとアクションは、アプリケーションのソリューション ディレクトリにあるセッション コントロール ファイル（**`SessionControl.txt`**）に保存されます。

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関するデータ収集アクションが保存されるため、以下を実行できます。

- メソッドの開始点または終了点でデータ収集アクションを指定します。
- セッション コントロール ファイルを保持して他のセッションに使用します。
- カバレッジ分析、メモリ分析、パフォーマンス分析、パフォーマンス エキスパートのセッションに影響を与えるエントリをセッション コントロール ファイルに作成します。

Visual Studio の [DevPartner] メニューからセッション コントロール ファイルを作成できます。また、Visual Studio 6.0 を使用している場合は、テキスト エディタでセッション コントロール ファイルを作成できます。DevPartner には、データ収集をさらに制御できるセッション コントロール API も用意されています。

Visual Studio 内でのセッション コントロール ファイルの作成

Visual Studio 2003 または 2005 では、以下のように **[DevPartner]>[オプション]** メニューからファイルを作成できます。セッション コントロール ファイルをテキスト エディタで作成する方法については、「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」（355 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、以下の操作を行います。

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。
   セッション コントロール ファイルのオプションをはじめて設定するときは、空のセッション コントロール（`SessionControl.txt`）ファイルにアクセスします。
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、データを収集するモジュールの場所を選択または参照します。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。

メモ： すべてのマネージ コード モジュールには、「インストゥルメントされていない」ステータスが表示されます。「インストゥルメント済み」のステータスが表示されるのは、ネイティブの C/C++ インストゥルメンテーションでビルドしたアンマネージ（ネイティブ）Visual C++ モジュールのみです。

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。

メモ： .NET モジュール（`.netmodule`）からメソッドを選択する場合、メソッド リストには、`namespace.classname.method` という形式でメソッドが表示されます。セッション コントロール ファイルでは、512 文字以内の修飾されたメソッド名がサポートされます。512 文字を超える名前は無視され、そのメソッドについてセッション コントロール アクションは発生しません。

6. 開始するセッション コントロール アクションを選択します。
7. 以下の中から適用するアクションを選択します。
   - **記録の停止（最終のスナップショットを取得）**
   - **スナップショットの取得**
   - **記録されているすべてのデータをクリア**
   - **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、ステップ 3 ～ 8 を繰り返します。

10 **[OK]** をクリックして、セッション コントロール ファイルが閉じ、保存します。

ソリューションを Visual Studio で開いている場合、セッション コントロール ファイルはソリューション ディレクトリに保存されます。

メモ： **プロファイル対象のアプリケーション実行可能ファイルが保存されているソリューション ディレクトリで、`SessionControl.txt` ファイルが検索されます。ソリューション ディレクトリにファイルが見つからなかった場合、アプリケーション実行ファイルをビルドした出力ディレクトリが検索されます。`SessionControl.txt` を別の場所に保存した場合、DevPartner はセッション コントロール コマンドを認識することができません。**

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、パフォーマンス エキスパートの分析セッションに影響があります。

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

Visual Studio 6.0 を使用している場合、テキスト エディタを使用してセッション コントロール ファイルを作成できます（Visual Studio 2003 または 2005 を使用している場合、「Visual Studio 内でのセッション コントロール ファイルの作成」（354 ページ）の説明に従って、DevPartner の [オプション] メニューからファイルを作成できます）。

セッション コントロール ファイルの一般的なフォーマットは以下のとおりです。

```
[NMPLAYLIST]
Action1=MyClass.CSharp.Form1;.ctor;Entry;Snap;2;
C:¥myPath¥ModuleName.dll
Action2=MyClass.VBdotNet.Form1;Dispose;Exit;Clear;2;
C:¥myPath¥ModuleName.dll
Action3=C:¥myPath¥NativeCPP.dll;NativeDialog;Entry;Stop;0;
C:¥myPath¥NativeCPP.dll
```

セッション コントロールのテキスト ファイルをテキスト エディタで作成するには、以下の操作を行います。

1 空のテキスト ファイルをテキスト エディタで開きます。

2 ファイルの 1 行めに **`[NMPLAYLIST]`** と入力します。

3 **`ActionN`** と入力します。この **`N`** は 1 を基本にして、ファイルのアクションごとに 1 ずつ増えるシーケンス番号です。シーケンス番号は反復を見込んで連続番号にする必要があります。

4 以下の情報を参照して、等号記号の右に、セミコロン区切りで以下の必須の値を入力します。

モジュール名またはクラス名	アンマネージ コードの場合、バイナリのフル パス名を入力します。マネージ コードの場合、メソッドが属するクラスの完全修飾名を入力します。 セッション コントロール ファイルでは、512 文字以内の修飾されたメソッド名がサポートされます。512 文字を超える名前は無視され、そのメソッドについてセッション コントロール アクションは発生しません。
メソッド名	アンマネージ コードの場合、(**?Method@@YAXXZ** のように) 名前を入力します。マネージ コードの場合、修飾されていないメソッド名を入力します。
アクションを実行するタイミング	**Entry** または **Exit**。
必要なアクション	**Snap** (スナップショット)、**Clear**、**Stop**、**StartLeak**、**StopLeak**、または **GC**。**Stop** コマンドによって、最終的なデータのスナップショットが作成され、データ収集は停止します。セッション コントロール ファイルで指定した他のすべてのメソッドが実行されたあとに実行されることがわかっているメソッドの開始点または終了点にのみ、このコマンドを使用します。**StartLeak**、**StopLeak**、**GC** は、マネージ コード アプリケーションのメモリ分析にのみ使用されます。**StartLeak** と **StopLeak** は、メモリ リーク分析の場合にのみ機能します。つまり、マネージ ヒープに割り当てられ、クリアされたメモリの追跡を開始および終了します。
タイプ	アンマネージ コードの場合は 0、マネージ コードの場合は 2 を入力します。
モジュールのフル パス名	アンマネージ モジュールとマネージ モジュールのどちらの場合でも、ドライブ、ディレクトリ、モジュール名、拡張子を含むフル パス名。

メモ： プロファイル対象のアプリケーション実行可能ファイルが保存されているソリューション ディレクトリで、`SessionControl.txt` ファイルが検索されます。`SessionControl.txt` を別の場所に保存した場合、DevPartner はセッション コントロール コマンドを認識することができません。

5 プロファイル対象のアプリケーション実行可能ファイルのソリューション ディレクトリに、**SessionControl.txt** ファイルを保存してください。

セッション コントロール API の使用

Visual Studio アプリケーションのデータ収集を制御するには、ソース コードの任意の場所からセッション コントロール API を呼び出します。セッション コントロールのテキスト ファイルを使用した場合、DevPartner セッション コントロール アクションはメソッドの開始点と終了点でのみ使用できます。

DevPartner セッション コントロール API の関数

`Clear`	現時点までに収集されたデータをクリアします。データ収集は続行します。データが正常にクリアされは場合は `NMStatusSuccess`、クリアされなかった場合は `NMStatusFailure` を返します。
`Snap`	記録されているデータのスナップショットを作成します。スナップショットが正常に保存されは場合は `NMStatusSuccess`、保存されなかった場合は `NMStatusFailure` を返します。
`SaveNow`	スナップショットを作成し、データ収集を終了します。メソッドにファイル名を指定すると、そのファイル名が使用されます。`NMStatusSuccess` または `NMStatusFailure` を返します。
`StartTrackingForLeakAnalysis`	割り当て済み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`	割り当て済みオブジェクトの追跡を終了します。（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` がコードに使用される最後の API 関数コールであることを確認します。この関数はそのプロセスのデータ収集を終了させるため、以降のすべての API コールは無視されます。

メモ： `Snap Session Control` の API コールによって、メモリ分析セッションに一時オブジェクトのセッション ファイルが作成されます。最後のガベージ コレクション以降に割り当てられたオブジェクトについて、メモリ サイズ データを取得するには、`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、マネージ コードとアンマネージ コードにそれぞれ使用できるセッション コントロールAPI関数が含まれます。すべてのファイルは、DevPartner Studioインストールの**¥DevPartner Studio¥Analysis**ディレクトリにインストールされます。

マネージ コードの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Visual C/C++またはC++アプリケーション	**NmTxApi.h**
アンマネージ（ネイティブ）コードのVisual Basicアプリケーション	**nmtxapi.bas**

マネージ アプリケーションでのセッション コントロールAPIの使用

マネージ コードの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、プロジェクトで**DevPartner.Analysis.SessionControl.dll**を参照する必要があります。

これによって、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、コードの適切な位置にAPIコールを挿入できます。

Clear

```
DevPartner.Analysis.SessionControl.Clear()
```

Snap

```
DevPartner.Analysis.SessionControl.Snap
(<your session file name>.dpxxx)
```

このdpxxxは、dpcov、dpmem、dpprf、またはdppxpという分析タイプの拡張子です。

SaveNow

```
DevPartner.Analysis.SessionControl.SaveNow
(<your session file name>.dpxxx)
```

このdpxxxは、dpcov、dpmem、dpprf、またはdppxpという分析タイプの拡張子です。

StartTrackingForLeakAnalysis

```
DevPartner.Analysis.SessionControl.StartTrackingForLeakAnalysis()
```

StopTrackingForLeakAnalysis

```
DevPartner.Analysis.SessionControl.StopTrackingForLeakAnalysis()
```

RunGC

```
DevPartner.Analysis.SessionControl.RunGC()
```

Snap と **SaveNow** API 関数の有効な入力値は以下のとおりです。

- ファイル名
- 「¥」（円マーク）で終了するディレクトリの完全修飾パス
- ファイル名を含む完全修飾パス
- なし（Null）

DevPartner がファイルとパスの情報を扱う方法については、「セッション コントロール API を使用したファイルの保存」（361 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 コード アプリケーションのプロファイルにセッション コントロール API を使用したときにセキュリティ例外が発生した場合、コードの通常の DevPartner インストゥルメンテーションが、セキュリティ ポリシーのために実行時に回避されたことを示します。これを修正するには、安全なプロファイルを有効にする必要あります。

以下のグローバル環境変数を設定します。

```
NM_NO_FAST_INSTR=1
```

アプリケーションのプロファイルを再試行します。

メモ： **デフォルトでは、アセンブリをプロファイルするためには SkipVerification 権限が必要です。コードの実行に使用しているポリシーの権限セットからこの権限を削除した場合、またはこの権限が取り消すような厳しいセキュリティ宣言をアセンブリに追加した場合、アセンブリはプロファイルできません。上記の方法で問題を回避できますが、パフォーマンスがやや低下します。上記の解決法を実装しない場合、DevPartner Studio で .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、アセンブリに含まれる厳しいセキュリティ宣言を一時的に削除することで、アセンブリのプロファイルを有効にすることもできます。**

Visual Studio のセキュリティ ポリシーの詳細については、Visual Studio のオンライン ヘルプに含まれる「.NET Framework 開発者ガイド」を参照してください。

アンマネージ アプリケーションでのセッション コントロール API の使用

セッション コントロール API を使用して、C/C++、Visual C++ 6.0、Visual Basic 6.0 のアンマネージ アプリケーションに対するカバレッジ分析セッションとパフォーマンス分析セッションを制御できます。

アンマネージ 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 データを収集する前に、[ネイティブ C/C++ インストゥルメンテーション] を使用してソリューション（またはネイティブ C/C++ プロジェクト）をリビルドする必要があります。

ネイティブ（アンマネージ）C/C++でセッション コントロールAPI関数を使用するには、以下の操作を行います。

1 セッション コントロールのAPIコールを追加するファイルに**NmTxApi.h**を含めます。リンク ライブラリ リストに**TxInterf.lib**を追加します。

2 セッション コントロールAPI関数コールを、コードの適切な位置に挿入します。「アンマネージ プロジェクト用のセッション コントロールAPIの構文」(361ページ）を参照してください。

3 ソリューションまたは個々のネイティブ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]を使用してリビルドします。

アンマネージVisual C++ 6.0プロジェクト

Visual C++ 6.0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 C++ 6.0で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リビルドする必要があります。

Visual C++ 6.0でセッション コントロールAPIを使用するには、以下の操作を行います。

1 セッション コントロールAPIコールを追加するファイルに`NmTxApi.h`を含めます。リンク ライブラリ リストに`TxInterf.lib`を追加します。

2 セッション コントロールAPI関数コールを、コードの適切な位置に挿入します。「アンマネージ プロジェクト用のセッション コントロールAPIの構文」(361ページ）を参照してください。

3 Visual C++ 6.0で**[DevPartner]>[ビルド]**（または**[リビルド]**か**[バッチ ビルド]**）**>[カバレッジ]または[パフォーマンス]**コマンドを使用して、ソリューションをリビルドします。

アンマネージVisual Basic 6.0プロジェクト

Visual Basic 6.0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 Basic 6.0で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リビルドする必要があります。

Visual Basic 6.0でセッション コントロールAPIを使用するには、以下の操作を行います。

1 セッション コントロールAPIコールを追加するファイルに`nmtxapi.bas`を含めます。

2 セッション コントロールAPI関数コールを、コードの適切な位置に挿入します。「アンマネージ プロジェクト用のセッション コントロールAPIの構文」(361ページ）を参照してください。

3 Visual Basic 6.0で**[DevPartner]>[カバレッジ分析でメイク]**または**[パフォーマンス分析でメイク]**を使用してソリューションをリビルドします。

アンマネージ プロジェクト用のセッション コントロール API の構文

アンマネージ プロジェクト用のセッション 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("<your session file name>.dpxxx")` この dpxxx は、dpcov または dpprf という分析タイプの拡張子です。
`SaveNow`	`SaveNow("<your session file name>.dpxxx")` この dpxxx は、dpcov または dpprf という分析タイプの拡張子です。

セッション コントロール API を使用したファイルの保存

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作成する場合、または最終的なセッション ファイルを作成する場合、API コールにセッション ファイル名とディレクトリを指定できます。

セッション コントロール API コールにファイル名とディレクトリを指定すると、コマンド ラインの **/output** スイッチ、XML 構成ファイルの **SESSION_FILENAME** 属性または **SESSION_DIR** 属性など、他の方法で指定したファイル名とディレクトリが上書きされます。

- セッション コントロールの **Snap** または **SaveNow** の API コールにファイル名とディレクトリを指定すると、指定に従ってファイルが保存されます。同じ名前のファイルが同じディレクトリに存在する場合、上書きされます。
- ディレクトリのみを指定すると、ターゲット プロセスの名前に基づいて一意なファイル名でセッションが保存されます。既存のファイルが上書きされないように、ファイル名の番号が自動的に 1 つ増えます。
- ファイル名のみを指定すると、指定した名前でセッションが保存されます。保存先フォルダは、アプリケーションの起動に使用した方法によって決まります。Visual Studio からアプリケーションを起動した場合、ファイルは現在のプロジェクトのソリューション ディレクトリに保存されます。**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した場合は、アクティブなユーザー アカウントの **My Documents** ディレクトリまたは **Documents** ディレクトリにファイルが保存されます。同じ名前のファイルが同じディレクトリに存在する場合、上書きされます。
- ファイル名もディレクトリも指定しないと、一意なファイル名でセッションが保存されます。保存先フォルダは、前述のようにアプリケーションの起動に使用した方法によって決まります。ファイルが上書きされないように、ファイル名の番号が自動的に 1 つ増えます。

メモ： Visual Studio 2005 の Web サイト プロジェクトなど、出力ディレクトリがないプロジェクトの場合、ファイルはプロジェクト ディレクトリに物理的に保存されます。

パスを指定するときは以下の点に注意してください。

- DevPartnerでは、プロファイル対象プロセスの現在の作業ディレクトリに対する相対パスとしてパス情報が評価されます。場合によっては、作業ディレクトリがアプリケーションの実行時に変わることがあるので注意してください。
- セッション ファイルの場所を簡単に特定できるように、フル パスを指定することをお勧めします。
- ローカル マシンに指定したフル パスが存在しない場合、そのパスが作成されます。リモート マシンのデータを収集する場合、既存のディレクトリを指定する必要があります。
- パスを指定してもファイル名は指定しない場合、パスを「¥」（円マーク）で終了してください。DevPartnerでは、最後の円マークに続く文字はファイル名とみなされます。
- パスに無効なデータが含まれる場合、ディレクトリが指定されなかった場合と同様にファイルが保存されます。

相互作用と優先

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ディレクトリは、他の方法で指定したファイル名とディレクトリを上書きします。

推奨：APIコールまたはコマンド ラインのいずれかでファイル名とディレクトリを設定します。両方には指定しません。

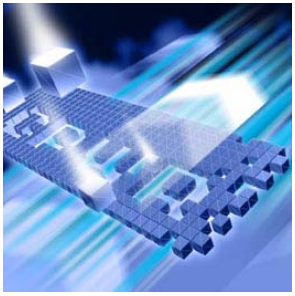
例：セッション コントロールAPIでディレクトリ（またはファイル名）のみを指定して、**DPAnalysis.exe**コマンド ラインまたはXML構成ファイルでファイル名（またはディレクトリ）を指定した場合、両方の情報を組み合わせてファイルに名前が付けられ、保存されます。この例では、一意なファイル名が自動的に作成されるように意図していた場合でも、その目的を達成できません。

推奨：ファイルの管理を単純にするために、スナップショットと最終的なセッション ファイルの両方をAPIコールで指定します。

例：セッション コントロールAPIによって最終的なスナップショット（**SaveNow**）を指定しない場合、プロセスの終了時に最終的なスナップショットが作成されます。**DPAnalysis.exe**で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したオプションに従って最終的なセッション 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した場合、保存されていないセッション データがDevPartnerに表示されます。

付録 E

分析データの XML へのエクスポート



この付録には、DevPartner のカバレッジ分析、パフォーマンス分析、パフォーマンス エキスパートに使用できる、分析データの XML へのエクスポートに関する情報が記載されています。

DevPartner データのエクスポートの概要

DevPartner では、カバレッジ分析、パフォーマンス分析、パフォーマンス エキスパート データから保存したセッション ファイルを XML にエクスポートできます。Visual Studio またはコマンド 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。

独自のソフトウェアや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 ファイルのデータを分析できます。以下に例を示します。

- 単体テスト、機能テスト、または回帰テストを行っている開発ビルド サーバーや QA サーバー上で **[DevPartner データのエクスポート]** を使用します。エクスポートされた XML データを分析して、毎日の進行状況を監視できます。
- 長期間の分析のためにデータを収集するには、**[DevPartner データのエクスポート]** を使用します。以下を実行するために、データベースまたはデータ ウェアハウスに XML データを蓄積できます。
  - 開発および QA の手法、ツール、インフラストラクチャとデータを統合する
  - データに対してカスタム分析を実行する
  - 履歴または監査の目的でデータをアーカイブする

分析データの XML へのエクスポート

Visual Studio 内から、保存した DevPartner カバレッジ分析（***.dpcov**）、カバレッジ分析のマージ（***.dpmrg**）、パフォーマンス分析（***.dpprf**）、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、以下の操作を行います。

1 保存したセッション ファイルを開きます（上記を参照）。

2 **[ファイル]>[DevPartner データのエクスポート]** を選択します。

デフォルトで、XML ファイルはセッション ファイルと同じフォルダに保存され、保存済みのセッション ファイル名に **.xml** の拡張子が付加されます。たとえば、**Chart1.dpcov.xml** のようになります。

DevPartnerPerformanceCoverage82.xsd のファイルには、カバレッジ分析データとパフォーマンス分析データのエクスポートに使用される XML スキーマが定義されています。**DevPartnerPerformanceExpert82.xsd** のファイルには、パフォーマンス エキスパート データのエクスポートに使用される XML スキーマが定義されています。どちらのスキーマも、**C:¥Program Files¥Compuware¥DevPartner Studio¥Analysis** に保存されています。

コマンド ラインからの分析データの XML へのエクスポート

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、カバレッジ分析のマージ、パフォーマンス分析、およびパフォーマンス エキスパートのデータを XML にエクスポートすることもできます。

このユーティリティは以下のディレクトリに保存されています。
C:¥Program Files¥Compuware¥DevPartner Studio¥Analysis

使用方法

```
DevPartner.Analysis.DataExport.exe [ sessionfilename |
pathtodirectory ] { options }
```

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ディレクトリを指定します。ディレクトリが存在しない場合、ディレクトリが作成されます。
/r[ecurse]	DevPartner セッション ファイルのサブディレクトリを検索します。
/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。 **/maxpaths** オプションと /maxmethods オプションを使用すると、最大数を上書きできます。
/method	パフォーマンス エキスパートのメソッド 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 エキスパートの **/summary** オプションでのみ使用します。最も多く CPU リソースを使用する上位のコール パスのうち、指定したパス数をエクスポートします。
/maxmethods=<integer>	パフォーマンス エキスパートの /summary オプションでのみ使用します。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

DevPartner.Analysis.DataExport.exe の使用例の一部を以下に示します。

例 1：カバレッジ分析のセッション ファイルを同じディレクトリの XML ファイルにエクスポートします。

```
DevPartner.Analysis.DataExport.exe
"c:¥WindowsApplication1¥WindowsApplication1.dpcov"
```

c:¥windowsApplication1¥WindowsApplication1.dpcov.xml の出力ファイルに保存されます。

例 2：ある場所に保存されているパフォーマンス分析のセッション ファイルを別のディレクトリにエクスポートします。

```
DevPartner.Analysis.DataExport.exe
"c:¥WindowsApplication1¥WindowsApplication1.dpprf"
/output="c:¥temp"
```

c:¥temp¥WindowsApplication1.dpprf.xml の出力ファイルに保存されます。

例 3: 同じディレクトリに保存されているパフォーマンス 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、同じディレクトリに、**WindowsApplication1.dppxp** と **WindowsApplication2.dppxp** という 2 つのパフォーマンス エキスパートのセッション ファイルが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"
```

以下の出力ファイルに保存されます。

c:¥WindowsApplication1¥WindowsApplication1.dppxp.xml
c:¥WindowsApplication1¥WindowsApplication2.dppxp.xml

例 4：同じディレクトリ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、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、同じディレクトリに、**WindowsApplication1.dpprf**、**WindowsApplication2.dpcov**、**WindowsApplication3.dppxp** という複数のセッション ファイルが保存されていることを前提としています。

```
DevPartner.Analysis.DataExport.exe "c:¥WindowsApplication1"
```

出力ファイルは以下の 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

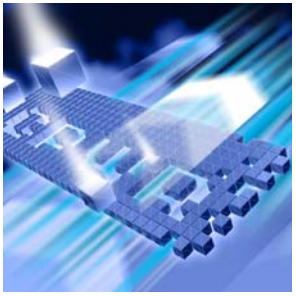
例 5: パフォーマンス エキスパートのサマリをエクスポートし、デフォルトの出力を、最も多く CPU リソースを使用している上位 10 位のメソッドから 20 位のメソッドに変更します。

```
DevPartner.Analysis.DataExport.exe
"c:WindowsApplication1WindowsApplication1.dppxp"
/summary /maxmethods=20
```

以下の出力ファイルに保存されます。

c:WindowsApplication1WindowsApplication1.dppxp.xml

索引



A

C

D

E

F

H

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

ふ

へ

ま

み

め

ゆ

よ

ら

り

る

れ

ろ

わ