アプリケーションのリソースリークを解決

インテル® Parallel Studio XE

リソースリークとは、リソース消費の 1 つで、プログラムが割り当てたリソースを解放していない状態を指します。通常、一般的なリソース問題 (メモリーリークなど) やコードの不具合による結果として起こりますが、非常に限られた状況や、長い間アプリケーションを使用した場合に問題を引き起こします。

インテル® Parallel Inspector XE は、シリアルおよび並列アプリケーションをサポートし、メモリー/スレッドエラー検証機能の両方を備えた動的解析ツールです。リソースリークを 1 回の発生で検出できるため、アプリケーションで不具合が再現できない場合でも、エラーの場所を特定できます。インテル® Parallel Inspector XE を実行してアプリケーションをチェックすることで、リソースリークをピンポイント検出し修正して、問題を未然に防ぐことができます。

概要

スレッドエラーとメモリーエラーを検出し解決した後にも、プログラムで原因の分からない不具合が断続的に起こる場合は、リソースリークが発生している可能性があります。リソースリークのほとんどは大きな問題ではありません。リソースにハンドルを割り当ててプログラムが終了するまで保持するようにすると、プログラムの終了時にそのリソースは自動で解放されます。

通常はこの方法で問題ありませんが、ファイルやハッチブラシなど、使用するリソースの数が増えるに従って問題となります。GDI ブラシなど、リソースの種類によっては利用が制限されているものもありますが、すべてのリソースはある程度のシステムメモリーを消費します。その結果、システムのパフォーマンスが低下したり、予期しないシステム API 障害が発生することがあります。

インテル® Inspector XE はシングルおよびマルチスレッドのエラー検証ツールです。26 種類のリソースを追跡し、コード内で割り当てられているリソースのうち解放されていないものを特定します。Linux* 版、Windows* 版 (Microsoft* Visual Studio* に統合) があり、C/C++、C#、.NET、Fortran で作成されたアプリケーションをサポートします。この統合された開発支援ツールは、エラーを正確に示し、アプリケーションの信頼性と品質を保証するためのガイダンスを提供します。

本ガイドは、インテル® Inspector XE を使用してプログラム中のリソースリークのエラーを検出、修正して、問題を未然に防ぐ方法について説明します。

手順: リソースエラーの特定、解析、解決

インテル® Inspector XE を使って、一連のステップを実行することで、シリアルまたは並列プログラムのリソースエラーを特定、解析、解決できます。このチュートリアルでは、「Colors」という名前のサンプルプログラムを使用して、順に説明します。

注: インテル® Inspector XE は Microsoft* Visual Studio* 2005/2008/2010 に統合されます。このチュートリアルでは、Microsoft* Visual* Studio* 2005 開発環境 (IDE) を使用しています。メニュー項目の異なる IDE を使用する場合はご注意ください。

ステップ 1: インテル® Parallel Studio XE のインストールと設定

推定所要時間: 15-30 分

1. インテル® Parallel Studio XE の評価版をダウンロードします。
2. **parallel_studio_xe_2011_setup.exe** をクリックしてインテル® Parallel Studio XE をインストールします (システムにより異なりますが、約 15-30 分かかります)。

Colors サンプル・アプリケーションのインストール

1. Colors_conf.zip サンプルファイルをローカルマシンにダウンロードします。このサンプルは、Microsoft* Visual Studio* 2005 を使用して作成された C++ GUI アプリケーションです。
2. **Colors_conf.zip** ファイルをシステムの書き込み可能なフォルダーに展開します。

サンプル・アプリケーションの実行

1. Microsoft* Visual Studio* でサンプルを開きます。**[ファイル] > [開く] > [プロジェクト/ソリューション]** を選択して、**colors_conf\vc8\colors.sln** ファイルを開きます。図 1

[ソリューション エクスプローラ] ペインに colors ソリューションが表示されます。図 2

図 1

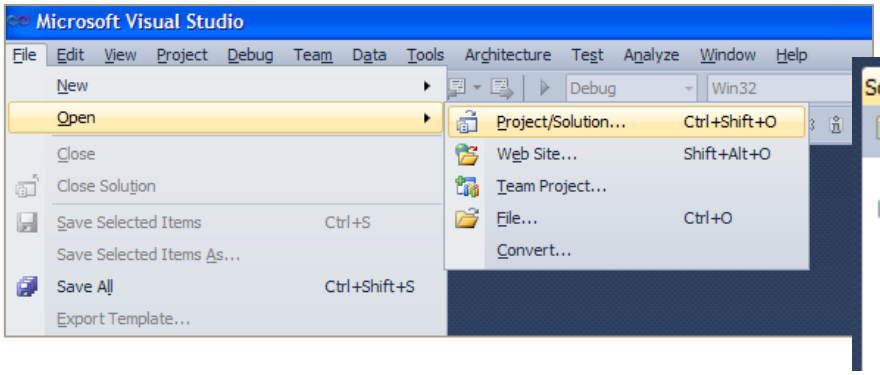
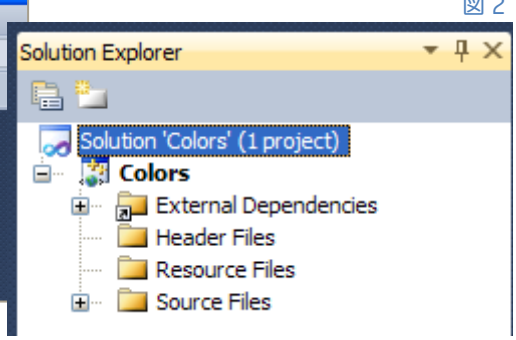


図 2



2. **[ビルド] > [ソリューションのビルド]** を選択してアプリケーションをビルドします。図 3
3. **[デバッグ] > [デバッグなしで開始]** をクリックしてアプリケーションを実行します。図 4

図 5 のようにアプリケーションが表示されます。

ウィンドウのサイズを 2 回変更してみてください。1 回目のサイズ変更後は正しく表示されますが、2 回目のサイズ変更後は 図 6 のように正しく表示されません。

もう 1 度サイズを変更すると、カラーの部分が全く表示されず、ウィンドウのコントロール・ボタンも正しく描画されません。この問題は、プログラム中のリソースリークにより発生します。

図 3

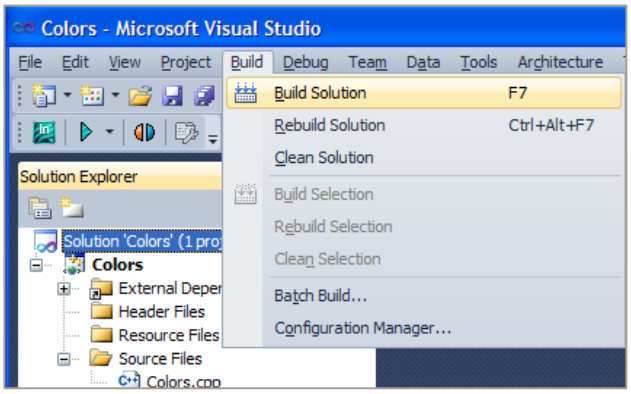


図 4

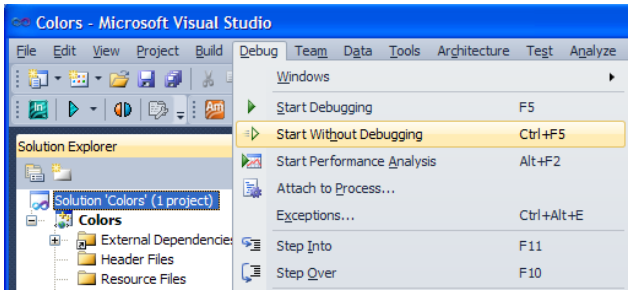


図 5

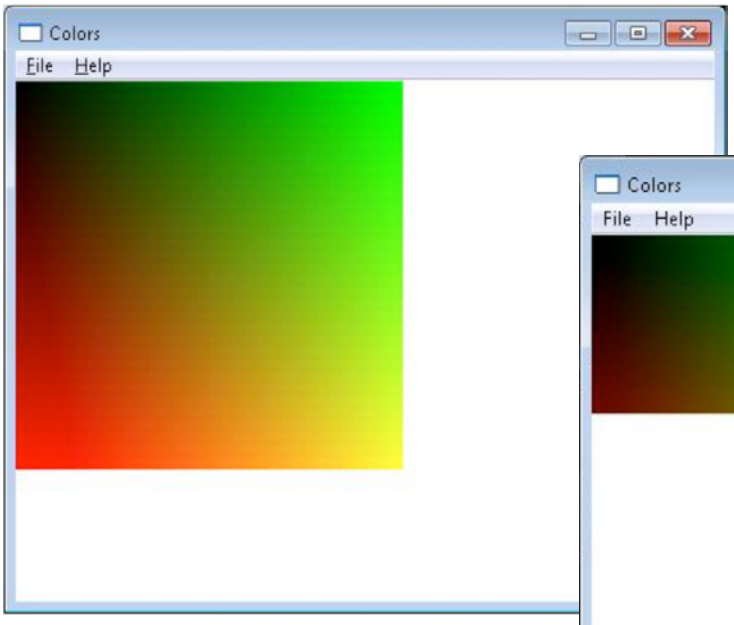
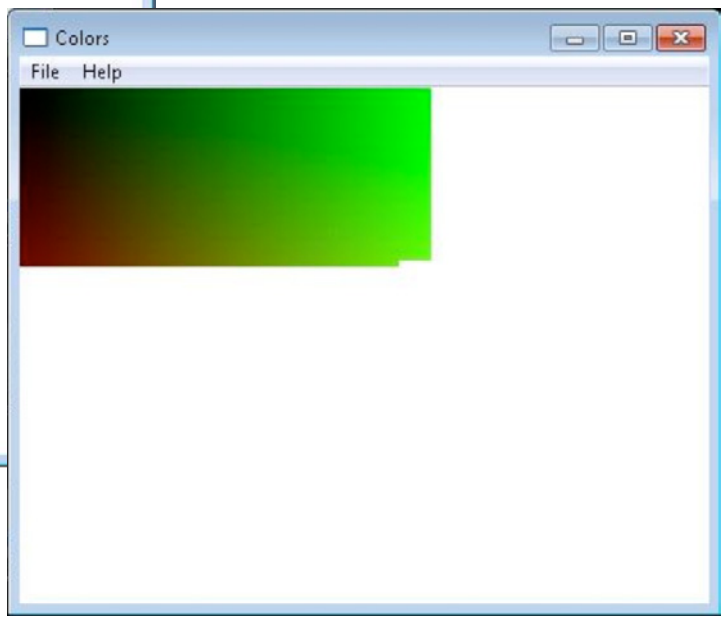


図 6



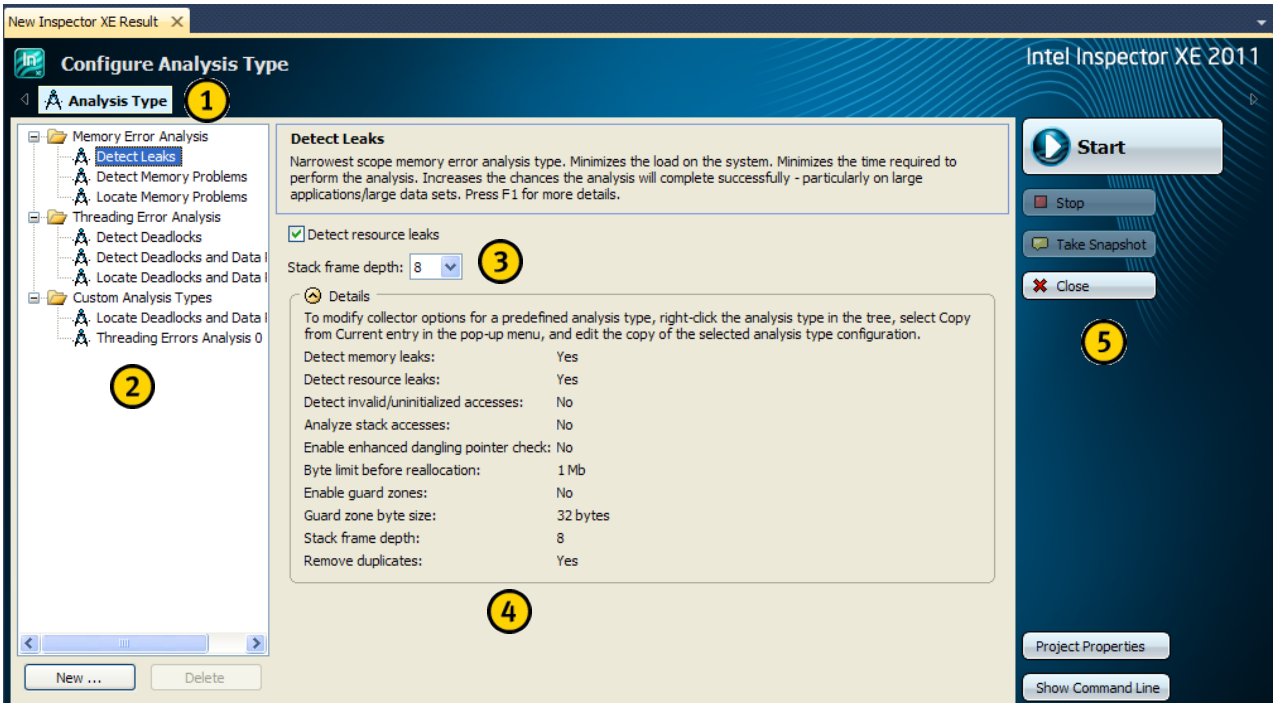
解析の設定と実行

メモリーエラー解析の範囲と実行時間に合ったプリセット設定を選択します。

メモリーエラー解析を設定するには:

1. Microsoft* Visual Studio* で、**[ツール] > [Intel Inspector XE 2011 (インテル(R) Inspector XE 2011] > [New Analysis (新しい解析)]** を選択して、[Analysis Type (解析タイプ)] ウィンドウを表示します。
2. **[Detect Leaks (リークの検出)]** 解析タイプを選択して、次のようなウィンドウを表示します。図 7

図 7



1. **ナビゲーション・ツールバー**を使用して、インテル® Inspector XE の各ウィンドウを切り替えます。ツールバーのボタンは、表示されているウィンドウによって変わります。

2. 解析タイプツリーに利用可能な解析タイプが表示されます。

   このチュートリアルでは、メモリーエラー解析を説明します。メモリーエラー解析は、GDI リソースリーク、不正な memcpy の呼び出し、不正な割り当て解除、カーネル・リソース・リーク、不正なメモリーアクセス、不正な部分メモリーアクセス、メモリーリーク、割り当てと解放の不一致、不明な割り当て、初期化されていないメモリーアクセス、初期化されていない部分メモリーアクセスの検出に使用できます。

   スレッドエラー解析タイプは、データ競合、デッドロック、ロック階層違反、スレッド間スタック・アクセスの検出に使用できます。

   [New (新規)] ボタンをクリックして、既存の解析タイプからカスタムの解析タイプを作成することもできます。

3. チェックボックスとドロップダウン・リストを使用して、解析タイプの設定を調整します (すべての設定を調整できるわけではありません)。その他の設定を調整したい場合は、別のプリセット解析タイプを選択するか、カスタム解析タイプを作成してください。

4. **[Details (詳細)]** 領域では現在のすべての解析タイプ設定が表示されます。別のプリセット解析タイプ、またはチェックボックス/ドロップダウン・リストの値を選択して、**[Details (詳細)]** 領域がどう変わるかを確認してみてください。

5. **コマンド**・ツールバーで、解析の実行を制御したり、他の機能を実行します。例えば、**[Project Properties (プロジェクト・プロパティー)]** ボタンを使用して、**[Project Properties (プロジェクト・プロパティー)]** ダイアログボックスを表示します。このダイアログボックスでは、デフォルトの結果ディレクトリーの場所を変更したり、解析を高速化するためにパラメーターを設定したり、他のプロジェクト構成機能を実行できます。

3. [Start (開始)] ボタンをクリックして、アプリケーションの解析を開始します。図 5 と同じ結果が表示された後、先程と同様に、ウィンドウのサイズを何回か変更して正しく表示されない問題を発生させます。その後アプリケーションを閉じてください。インテル® Inspector XE によって最終解析が行われます。この処理にはしばらく時間がかかります。解析中は、Microsoft* Visual Studio* IDE に 図 8 のようなウィンドウが表示されます。

インテル® Inspector XE の解析が終了すると、図 9 のようなウィンドウが表示されます。

図 8

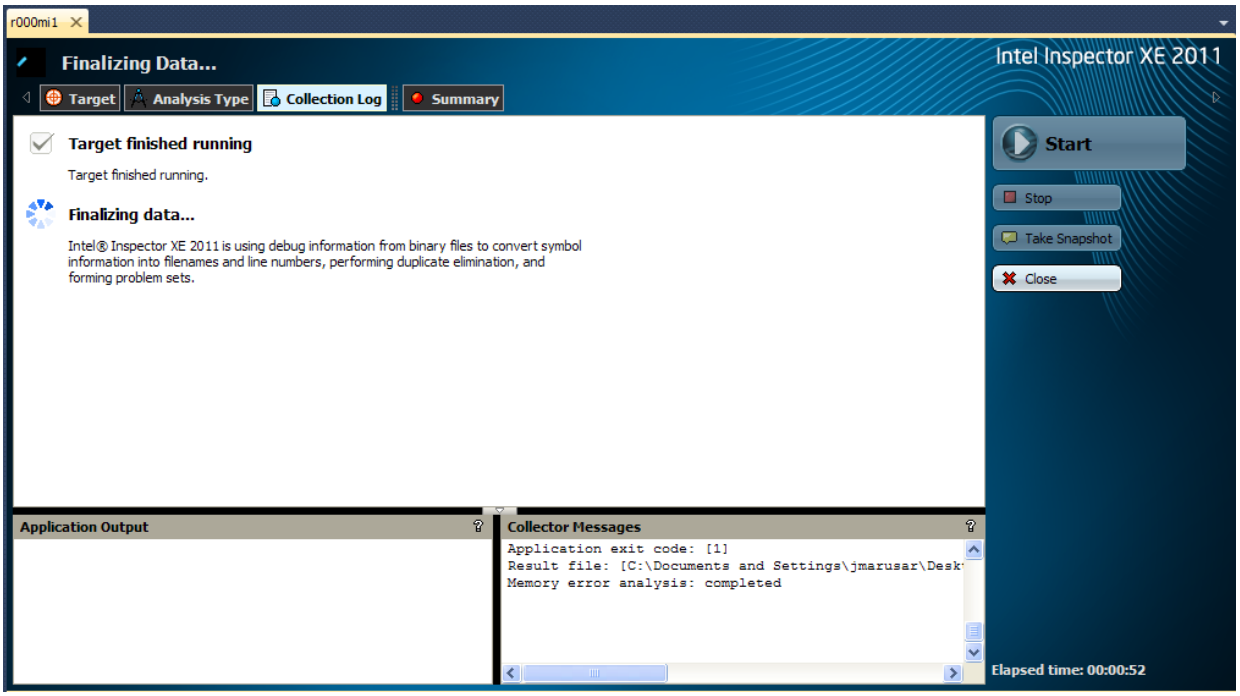
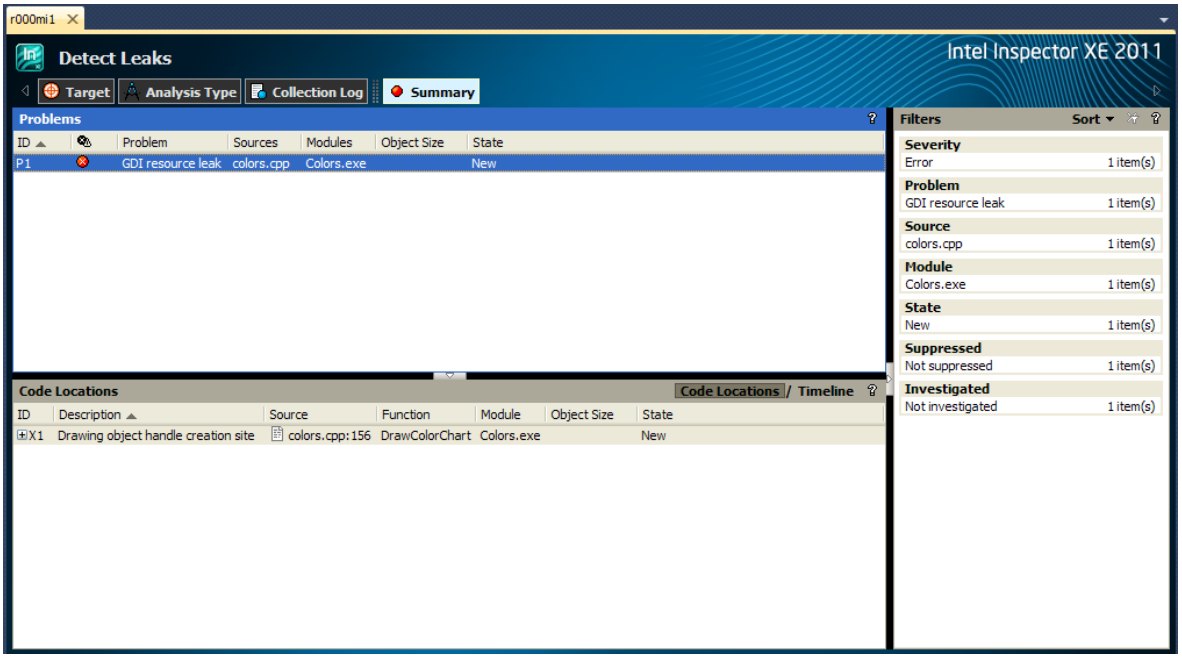


図 9



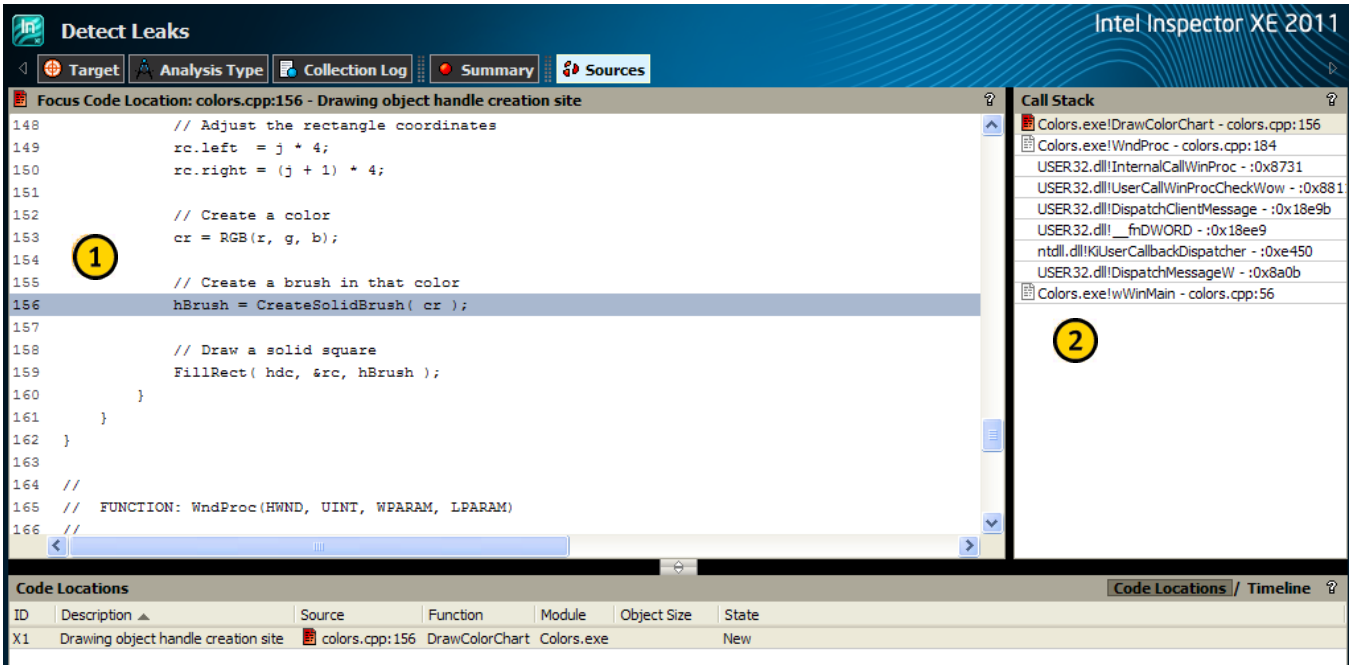
結果の解釈

図 9 のように、GDI リソースリークが 1 件表示されます。下のペインから、これは colors.cpp ファイルの 156 行目にある DrawColorChart 関数の描画オブジェクト・ハンドルのリークであることが分かります。[Problem (問題)] リストでこの問題をクリックすると、[Sources (ソース)] ビューが表示されます。

このビューでは、①問題のハンドルが作成されたソースコードの場所と、②エラーを引き起こしたコールスタックを確認できます。図 10

ソースコードから、GDI ブラシが 156 行目で作成され、このブラシが解放されていないことが分かります。

図 10



リソースリークのレポートでは、プログラムでリソースのハンドルが保持され解放されていない場合と、プログラムがすべてのハンドルを失った場合が区別されていません。問題を修正する前に、ソースコードを調べて、割り当てられるリソースの存続期間を確認する必要があります。

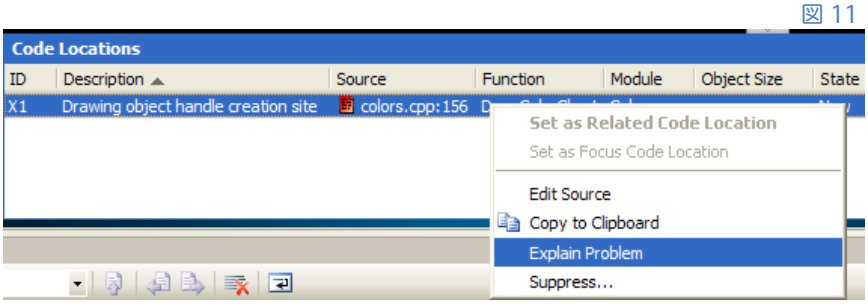
この例では、問題のハンドルは作成されたスコープの範囲外では使用されないため、使用後直ちに削除することができます。リソースを解放するための適切な関数が分からない場合は、左下のペインでコードの場所を右クリックして **[Explain Problem (問題の説明)]** を選択すると、そのリソースを解放するための関数を含むリソースリークの説明が表示されます。図 11

問題を修正する場合は、[Sources (ソース)] ビューで修正する行をダブルクリックすると、ソースファイルが開いて選択した行を編集できます。サンプル・アプリケーションで見つかった問題を解決するには、FillRect() の呼び出しの下に次のコードを追加します。

```
DeleteObject( hBrush );
```

そして、ソリューションをリビルドして再度実行してみます。ウィンドウのサイズを繰り返し変更しても、問題が発生しないことを確認できるでしょう。

図 11



結果

この例では、目に見える影響があり、簡単に再現できるリソース・リークのエラーについて説明しました。しかし、リソースリークは常にこのように容易に気が付くものであるとは限りません。非常に特殊な状況下でのみ発生したり、長期間にわたってアプリケーションを使用した後に見つかることはよくあることです。

インテル® Inspector XE は、リソースリークを 1 回の発生で検出できます。そのため、アプリケーションで不具合が再現できない場合でも、エラーの場所を特定します。インテル® Inspector XE を定期的に実行してアプリケーションをチェックすることで、開発サイクルの早期にリソースリークのエラーを発見し、修正することが可能です。

大規模なアプリケーション/複雑なアプリケーションの場合のヒント

重要な概念: 小さく代表的なデータセットを選択する

解析を実行するとき、インテル® Inspector XE はデータセットに応じてターゲットを実行します。データセットのサイズはターゲットの実行時間と解析速度に直接影響します。

例えば、1000x1000 ピクセルのイメージのほうが、100x100 ピクセルのイメージよりも処理は長くなります。大きなイメージではループで 1…1000 の反復空間が必要になるのに対して、小さなイメージでは 1…100 でかまわないことも理由の 1 つです。完全に同じコードパスを両方のケースで実行します。違いは、これらのコードパスを繰り返す回数だけです。

ターゲットから冗長な処理を省くことで、完全性を損なうことなく、解析時間を制御できます。大きな繰り返し型のデータセットの代わりに、小さく典型的なデータセットを選択してください。数秒で実行できるデータセットが理想的です。すべてのコードが確実に検査されるように、他のデータセットを作成することもできます。

スレッドエラーの管理

インテル® Inspector XE は、並列プログラムの隠れたデータ競合やデッドロックのようなスレッドエラーも特定、解析、解決します。発見、再現、修正が非常に困難な、再現性がなく、異なる結果となるようなエラーも検出できます。

コマンドラインを使用したテストの自動化

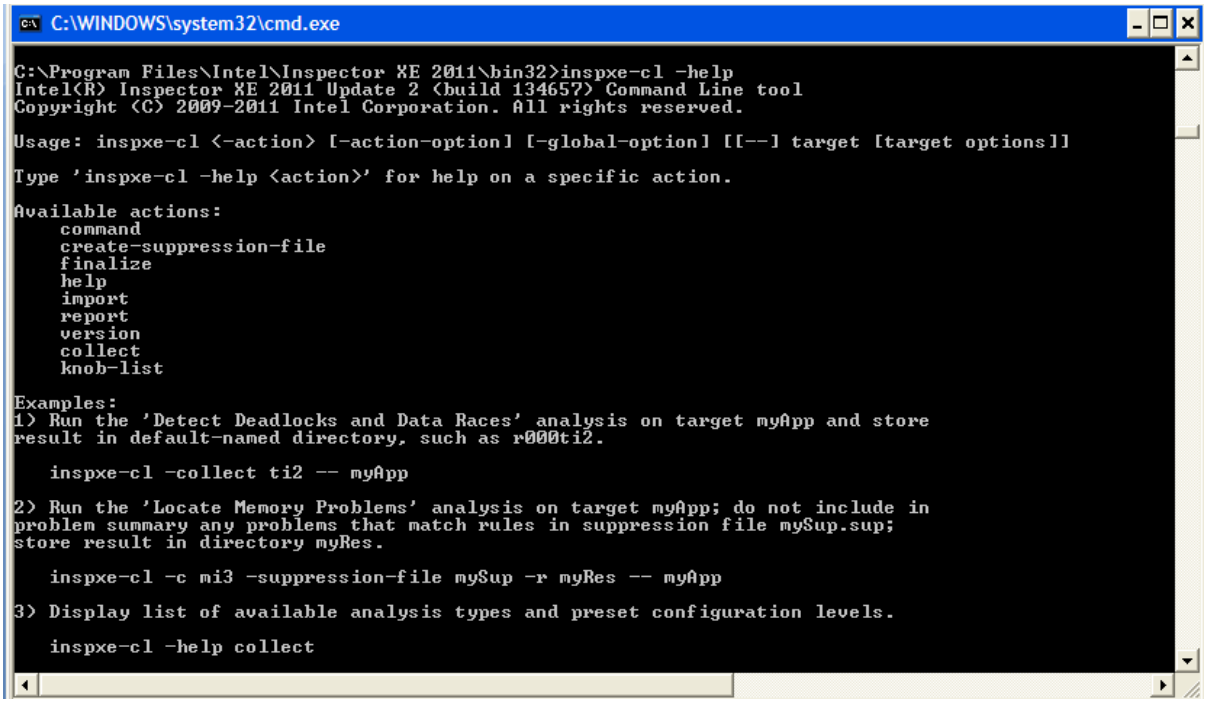
インテル® Inspector XE は、エラーを検出するためにコードパスを実行する必要があるため、異なるコードパスや異なるワークロードを考慮して、コードを複数回実行することになります。このため、コード検査ツールが十分な時間をかけてテストできるように、これらのテストを一晩中、あるいはリグレッション・テストの一部として実行し、コンピューターに作業させるほうがより効率的です。翌日に複数のテストの結果を確認するだけで済みます。

インテル® Inspector XE のコマンドライン・バージョン (inspxe-cl) は、コマンドウィンドウから使用します ([スタート] > [ファイル名を指定して実行] を選択し、cmd と入力して [OK] をクリックした後、インテル® Inspector XE をインストールしたフォルダーのパスを入力して起動します)。図 12

inspxe-cl でヘルプを表示するには、-help コマンドライン・オプションを使用します。

```
> c:\Program Files\Intel\Inspector XE 2011\bin32\inspxe-cl -help
```

図 12



関連情報

ラーニング・ラボ
テクニカルビデオ、ホワイトペーパー、Webinar の再生など

インテル® Parallel Studio XE 製品ページ
HOW TO ビデオ、入門ガイド、ドキュメント、製品の詳細情報、サポートなど

評価ガイド
さまざまな機能の使用法を紹介する評価ガイド

インテル® ソフトウェア・ネットワーク・フォーラム
開発者のコミュニティー

インテル® ソフトウェア製品ナレッジベース
製品やライセンスに関する情報を掲載

30 日間の評価版のダウンロード

最適化に関する注意事項

インテル® コンパイラー、関連ライブラリーおよび関連開発ツールには、インテル製マイクロプロセッサーおよび互換マイクロプロセッサーで利用可能な命令セット (SIMD 命令セットなど) 向けの最適化オプションが含まれているか、あるいはオプションを利用している可能性がありますが、両者では結果が異なります。また、インテル® コンパイラー用の特定のコンパイラー・オプション (インテル® マイクロアーキテクチャーに非固有のオプションを含む) は、インテル製マイクロプロセッサー向けに予約されています。これらのコンパイラー・オプションと関連する命令セットおよび特定のマイクロプロセッサーの詳細は、『インテル® コンパイラー・ユーザー・リファレンス・ガイド』の「コンパイラー・オプション」を参照してください。インテル® コンパイラー製品のライブラリー・ルーチンの多くは、互換マイクロプロセッサーよりもインテル製マイクロプロセッサーでより高度に最適化されます。インテル® コンパイラー製品のライブラリー・ルーチンの多くは、互換マイクロプロセッサーよりもインテル製マイクロプロセッサーでより高度に最適化されます。インテル® コンパイラー製品のコンパイラーとライブラリーは、選択されたオプション、コード、およびその他の要因に基づいてインテル製マイクロプロセッサーおよび互換マイクロプロセッサー向けに最適化されますが、インテル製マイクロプロセッサーにおいてより優れたパフォーマンスが得られる傾向にあります。

インテル® コンパイラー、関連ライブラリーおよび関連開発ツールは、互換マイクロプロセッサー向けには、インテル製マイクロプロセッサー向けと同等レベルの最適化が行われない可能性があります。これには、インテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2)、インテル® ストリーミング SIMD 拡張命令 3 (インテル® SSE3)、ストリーミング SIMD 拡張命令 3 補足命令 (SSSE3) 命令セットに関連する最適化およびその他の最適化が含まれます。インテルでは、インテル製ではないマイクロプロセッサーに対して、最適化の提供、機能、効果を保証していません。本製品のマイクロプロセッサー固有の最適化は、インテル製マイクロプロセッサーでの使用を目的としています。

インテルでは、インテル® コンパイラーおよびライブラリーがインテル製マイクロプロセッサーおよび互換マイクロプロセッサーにおいて、優れたパフォーマンスを引き出すのに役立つ選択肢であると信じておりますが、お客様の要件に最適なコンパイラーを選択いただくよう、他のコンパイラーの評価を行うことを推奨しています。インテルでは、あらゆるコンパイラーやライブラリーで優れたパフォーマンスが引き出され、お客様のビジネスの成功のお役に立ちたいと願っております。お気づきの点がございましたら、お知らせください。

改訂 #20110307