Oracle® Solaris Studio 12.4: IDE クイックスタートチュートリアル

Part No: E57222
2014 年 10 月

ORACLE

目次

1 Oracle Solaris Studio IDE の概要 7
Oracle Solaris Studio IDE を使用する理由 7

2 新規アプリケーションの作成 9
アプリケーションプロジェクトの作成 9
プロジェクトの論理ビューと物理ビューの切り換え 11
プロジェクトへのファイルの追加 12
プロジェクトの論理ファイルおよびフォルダの作成 12
プロジェクトの新規ソースファイルの作成 13
プロジェクトのヘッダーファイルの作成 13
既存ファイルのプロジェクトへの追加 13
プロパティーと構成の設定 14
プロジェクトのプロパティーの設定 14
構成の管理 15
ソースファイルのプロパティーの設定 16
プロジェクトの構築 16
単体のファイルのコンパイル 17

3 プロジェクトの実行 19
プロジェクトの実行 19
起動ツール 20

4 既存のソースからのプロジェクトの作成 21
既存のソースからのプロジェクトの作成 21
プロジェクトの構築と再構築 22
バイナリファイルからのプロジェクトの作成 23
新規プロジェクトウィザードを使用したバイナリファイルからのプロジェクトの作成 23
「ファイル」ウィンドウでのバイナリファイルからのプロジェクトの作成 24
コード支援 25

コード支援の構成 25
コード支援キャッシュの共有 25
コード支援のプロジェクトのプロパティーのオプション 26
ファイルシステムから C/C++ ヘッダーファイルを検索 26

5 Oracle Database プロジェクトの作成 27
Oracle Database プロジェクトの作成 27
新規データベース接続の作成 28

6 ソースファイルの編集とナビゲート 31
ソースファイルの編集 31
書式設定スタイルの設定 31
C および C++ ファイルでのコードのブロックの折り畳み 32
意味解釈の強調表示の使用 33
コード補完の使用 34
静的コードエラー検査の使用 37
ソースコードドキュメントの追加 38
コードテンプレートの使用 39
ペア補完の使用 40
プロジェクトファイルでのテキストの検索 41
ソースファイルのナビゲーション 44
ウィンドウの管理とグループ化 45
「クラス」ウィンドウの使用 45
「ナビゲータ」ウィンドウの使用 46
クラス、メソッド、およびフィールドの使用状況の検出 47
コールグラフの使用 48
ハイパーリンクの使用 50
インクルード階層の使用 52
タイプの階層の使用 53

7 アプリケーションのデバッグ 55
ブレークポイントの作成 55
行ブレークポイントの作成と削除 55
関数ブレークポイントの作成 56
ブレークポイントのグループ化 58
プロジェクトのデバッグ 58
デバッグセッションを開始する 59
アプリケーションの状態の検査 60
実行可能ファイルのデバッグ 63

機械命令レベルでのデバッグ 64
実行中のプログラムを接続してデバッグ 66
既存のコアファイルのデバッグ 68

8 プロジェクトのモニタリング 69
実行モニターのプロファイリングツールを使用したプロジェクトのプロファイリング 69
プロファイリングプロジェクトの設定 70
ProfilingDemo_1 プロジェクトの構築と実行 72
インジケータコントロールの使用 75
CPU 使用状況の調査 79
メモリー使用状況の調査 82
スレッド使用状況の調査 83
プロジェクトでのメモリーアクセス検査の実行 87

9 リモート開発の実行 91
リモート開発の実行 91
リモート開発ツールバーの使用 93
構築ホスト用のターミナルウィンドウの使用 94
リモートデスクトップ配布 94

10 アプリケーションのパッケージング 95
アプリケーションのパッケージング 95

11 IDE に関する詳細情報 99
IDE の詳細 99
Oracle Database プロジェクト 99
リモート開発 99
バージョン管理システム 100

♦ ♦ ♦ 第 1 章

1 Oracle Solaris Studio IDE の概要

この章では、Oracle Solaris Studio IDE を使用する利点について説明します。

Oracle Solaris Studio IDE を使用する理由

Oracle Solaris Studio は、NetBeans プラットフォーム上に構築され、環境内で直接 Oracle Solaris Studio ツールスイートを使用して構成されたグラフィカルな統合開発環境 (IDE) を提供します。IDE では、次の操作を実行できます。

- **Oracle Solaris Studio C、C++、および Fortran コンパイラ** と `dmake` **分散 make コマンド**を使用してコードをコンパイルおよび構築します。
- `dbx` **デバッガ**を使用してコードをデバッグします。詳細については、第7章「アプリケーションのデバッグ」を参照してください。
- Oracle Solaris Studio 分析ツールと統合するプロファイリングツールを使用してコードをプロファイリングします。
  - **「実行モニター」ツール**では、パフォーマンスアナライザのデータ収集が使用されます。
  - **「メモリー解析」ツール**では `discover` の動的メモリー検査が使用されます。詳細については、87 ページの「プロジェクトでのメモリーアクセス検査の実行」を参照してください。
  - **静的コード検査**では、コードアナライザの静的エラーデータ収集が使用されます。詳細については、37 ページの「静的コードエラー検査の使用」を参照してください。
  - **「データの競合とデッドロック」ツール**では、スレッドアナライザのデータ収集および分析が使用されます。
- **リモート開発**を使用して Oracle Solaris Studio コンパイラおよびツールがリモートサーバーにインストールされている場合に、IDE をローカルに実行します。詳細については、91 ページの「リモート開発の実行」を参照してください。
- **リモートデスクトップ配布**を使用して、リモートビルドサーバーで実行中に IDE の特殊な配布を作成します。

このガイドの以降の項では、IDE でこれらすべての機能を使用する方法を説明します。

♦ ♦ ♦ 第 2 章

2 新規アプリケーションの作成

この章では、IDE で管理対象プロジェクトと呼ばれることがある新しいアプリケーションプロジェクトを作成する手順を説明します。詳細については、9 ページの「アプリケーションプロジェクトの作成」を参照してください。

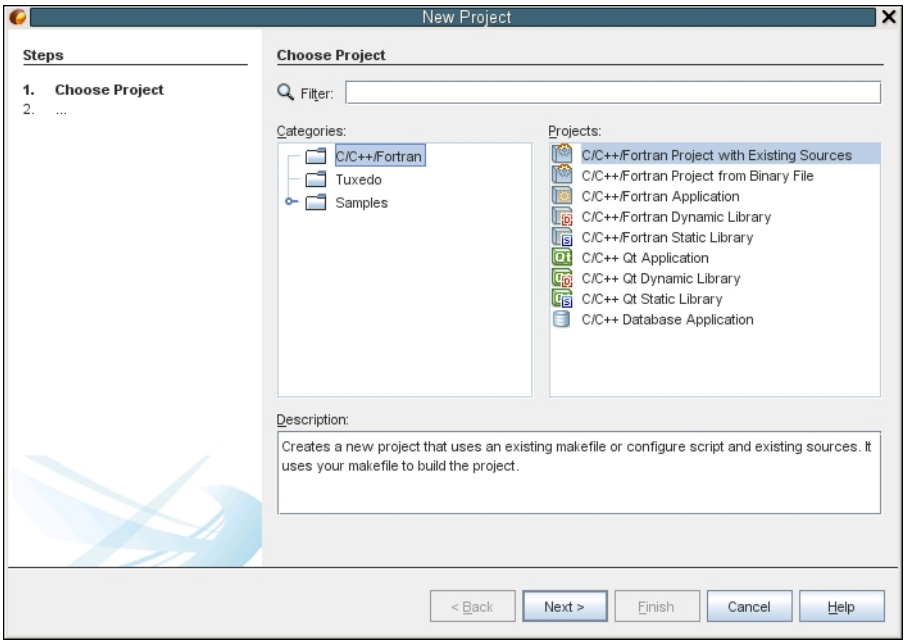
既存のソースを使用してプロジェクトを作成する方法については、第4章「既存のソースからのプロジェクトの作成」を参照してください。

この章では、プロジェクトに対して実行できるその他のタスクについても説明します。

- 11 ページの「プロジェクトの論理ビューと物理ビューの切り換え」
- 12 ページの「プロジェクトへのファイルの追加」
- 14 ページの「プロパティーと構成の設定」
- 16 ページの「プロジェクトの構築」
- 17 ページの「単体のファイルのコンパイル」

アプリケーションプロジェクトの作成

1. 「ファイル」>「新規プロジェクト」(Ctrl+Shift+N) を選択して、新規プロジェクトウィザードを開きます。
2. ウィザードで、C/C++/Fortran カテゴリを選択します。
3. ウィザードでは、新規プロジェクトのタイプを選択できます。「C/C++/Fortran アプリケーション」を選択して、「次へ」をクリックします。



4. デフォルト値を使用して、新しい C/C++/Fortran アプリケーションプロジェクトを作成します。プロジェクトの名前とプロジェクトの場所を選択できます。
5. 「完了」をクリックしてウィザードを終了します。

プロジェクトは、「プロジェクト」ウィンドウに表示される複数の論理フォルダを使用して作成されます。

- ソースファイル
- ヘッダーファイル
- リソースファイル
- テストファイル
- 重要なファイル

既存のソースを使用した C、C++、または Fortran プロジェクトは、「重要なファイル」という 1 つの論理フォルダを使用して作成されます。

論理フォルダはディレクトリではありません。ファイルを整理する手段であり、ファイルがディスク上に物理的に保存される場所を示すものではありません。「ソースファイル」フォルダ、「ヘッ

ダーファイル」フォルダ、その他の論理フォルダに追加されたファイルはプロジェクトの一部となり、プロジェクトの構築時にコンパイルされます。

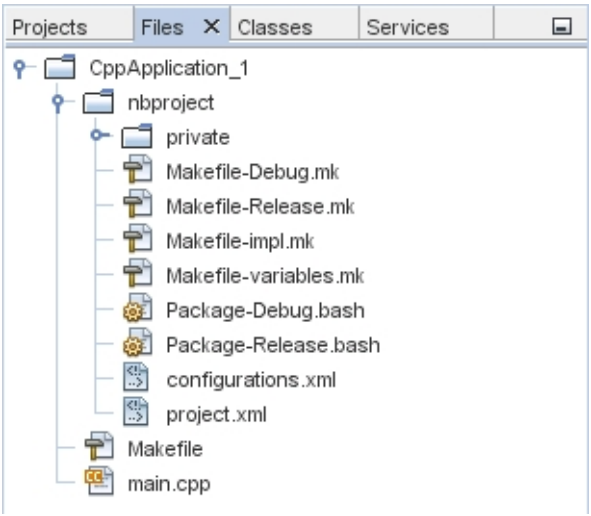
注記 - 「ヘッダーファイル」フォルダに追加されるヘッダーファイルをプログラムにコンパイルする場合、これらのファイルは通常の方法で `#include` 文を使用して、ソースファイル内で参照される必要があります。

「重要なファイル」フォルダに追加されたファイルはプロジェクトの一部ではなく、プロジェクトの構築時にコンパイルされません。これらのファイルは参照用のみで、既存のメイクファイルがプロジェクトにある場合に便利です。

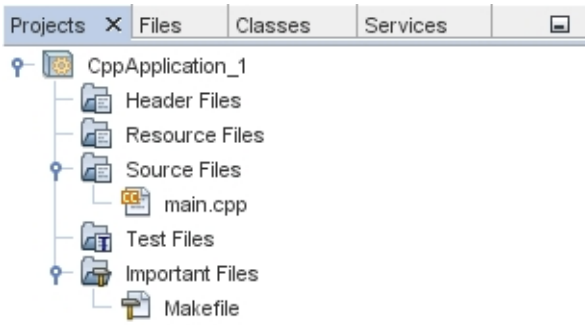
プロジェクトの論理ビューと物理ビューの切り換え

プロジェクトには、論理ビューと物理ビューがあります。プロジェクトの論理ビューと物理ビューを切り換えられます。

1. 「ファイル」タブを選択します。このウィンドウには、プロジェクトの物理ビューが表示されます。ディスクに保存されているファイルとフォルダが表示されます。



2. 「プロジェクト」タブを選択します。このウィンドウには、プロジェクトの論理ビューが表示されます。



プロジェクトへのファイルの追加

以降のセクションでは、各種ファイルとフォルダを現在のプロジェクトに追加する方法 (ファイルとフォルダの作成、または既存のソースの導入) について説明します。

- 12 ページの「プロジェクトの論理ファイルおよびフォルダの作成」
- 13 ページの「プロジェクトの新規ソースファイルの作成」
- 13 ページの「プロジェクトのヘッダーファイルの作成」
- 13 ページの「既存ファイルのプロジェクトへの追加」

プロジェクトの論理ファイルおよびフォルダの作成

論理フォルダを作成して、プロジェクトに追加できます。

1. `CppApplication_1` プロジェクトのプロジェクトノードを右クリックして、「新規論理フォルダ」を選択します。新しい論理フォルダがプロジェクトに追加されます。
2. 新しい論理フォルダを右クリックして、「名前の変更」を選択します。新しいフォルダに付ける名前を入力します。

ファイルとフォルダの両方を既存のフォルダに追加できます。論理フォルダは入れ子にすることができます。

プロジェクトの新規ソースファイルの作成

新規ソースファイルを作成して、プロジェクトに追加できます。

1. 「ソースファイル」フォルダを右クリックして、「新規」>「C main ファイル」を選択します。
2. 「名前と場所」ページで、newmain が「ファイル名」フィールドに表示されます。
3. 「完了」をクリックします。

注記 - また、フォルダからファイルを削除できます。この場合、プロジェクトを作成したときにデフォルトで追加された main.cpp ファイルは必要ありません。このファイルをプロジェクトから削除するには、ファイル名を右クリックして「プロジェクトから削除」を選択します。

プロジェクトのヘッダーファイルの作成

新規ヘッダーファイルを作成して、プロジェクトに追加できます。

1. 「ヘッダーファイル」フォルダを右クリックして、「新規」>「C ヘッダーファイル」を選択します。
2. 「名前と場所」ページで、newfile が「ファイル名」フィールドに表示されます。
3. 「完了」をクリックします。

newfile.h ファイルがプロジェクトディレクトリのディスクに作成され、「ヘッダーファイル」フォルダに追加されます。

既存ファイルのプロジェクトへの追加

2 つの方法で、既存のファイルをプロジェクトに追加できます。

- 「ソースファイル」フォルダを右クリックして、「既存の項目を追加」を選択します。「項目を選択」ダイアログボックスを使用してディスク上の既存ファイルを選択して、ファイルをプロジェクトに追加できます。
- 「ソースファイル」フォルダを右クリックして、「フォルダから既存の項目を追加」を選択します。「フォルダの追加」ダイアログボックスを使用して、既存ファイルを含むフォルダを追加します。

既存の項目の追加に「新規」メニュー項目を使用しないでください。「名前と場所」パネルから、ファイルがすでに存在していることが通知されます。

プロパティーと構成の設定

以降のセクションでは、現在のプロジェクトと個々のファイルの両方のプロパティーを設定および管理する方法について説明します。

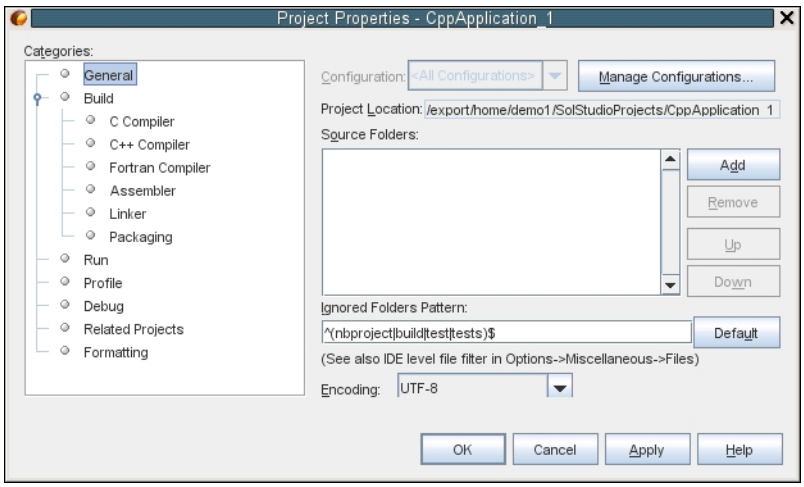
- 14 ページの「プロジェクトのプロパティーの設定」
- 15 ページの「構成の管理」
- 16 ページの「ソースファイルのプロパティーの設定」

プロジェクトのプロパティーの設定

プロジェクトを作成するとき、デバッグとリリースという 2 つの構成があります。構成はプロジェクトに使用される一連の設定であり、多くのプロパティー設定を一度に簡単に切り替えることができます。デバッグ構成では、デバッグ情報を含むバージョンのアプリケーションを構築します。リリース構成では、最適化バージョンを構築します。

「プロジェクトのプロパティー」ダイアログボックスには、プロジェクトの構築情報と構成情報が含まれています。「プロジェクトのプロパティー」ダイアログボックスを開くには、次の手順に従います。

- アプリケーションプロジェクトのプロジェクトノードを右クリックして、「プロパティー」を選択します。



左側のパネルでノードを選択して右側のパネルでプロパティーを変更して、「プロジェクトのプロパティー」ダイアログボックスでコンパイル設定およびその他の構成設定を変更できます。ノードとプロパティー値を選択して、設定できるプロパティーに注目します。一般プロパティーを設定すると、プロジェクトのすべての構成で設定が行われます。構築、実行、もしくはデバッグプロパティーを設定すると、現在設定されている構成のプロパティーが設定されます。

構成の管理

プロジェクトが作成されると、プロジェクトにデバッグ構成とリリース構成が指定されます。構成は、プロジェクトに使用される設定のコレクションです。構成を選択すると、多くの設定を一度に簡単に切り替えることができます。デバッグ構成では、デバッグ情報を含むバージョンのアプリケーションを構築します。リリース構成では、最適化バージョンを構築します。

「プロジェクトのプロパティー」ダイアログボックスで変更されたプロパティーは、現在の構成のメイクファイルに保存されます。デフォルトの構成を編集したり、新しい構成を作成したりできます。

新しい構成を作成するには、次の手順に従います。

1. 「プロジェクトのプロパティー」ダイアログボックスで「構成を管理」ボタンをクリックします。
2. 「構成」ダイアログボックスで、目的の構成にもっとも近い構成を選択します。この場合、Release 構成を選択して、「コピー」ボタンをクリックします。その後、「名前を変更」をクリックします。
3. 「名前を変更」ダイアログボックスで、構成の名前を「PerformanceRelease」に変更します。「OK」をクリックします。
4. 「構成」ダイアログボックスで「OK」をクリックします。
5. 「プロジェクトのプロパティー」ダイアログボックスで、左側のパネルの「C コンパイラ」ノードを選択します。PerformanceRelease 構成は「構成」ドロップダウンリストで選択されています。
6. 右側のパネルのプロパティーシートで、「開発モード」を Release から PerformanceRelease に変更します。「OK」をクリックします。

別のオプションセットでアプリケーションをコンパイルする、新しい構成が作成されました。

ヒント - アクティブな構成は、IDE のツールバーで設定したり、プロジェクトノードを右クリックして「構成を設定」を選択して設定することもできます。

ソースファイルのプロパティーの設定

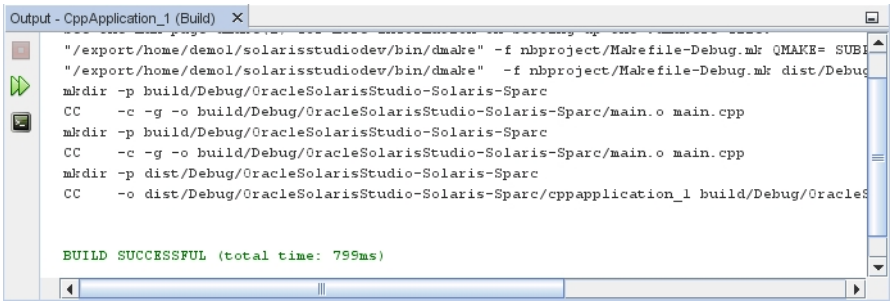
プロジェクトにプロジェクトプロパティーを設定すると、関連するプロパティーがプロジェクト内のすべてのファイルに適用されます。特定のファイルにプロパティーを設定できます。

1. newmain.c ソースファイルを右クリックして、「プロパティー」を選択します。
2. 「カテゴリ」パネルの「一般」ノードをクリックして、このファイルの構築に別のコンパイラまたはその他のツールを選択できることを確認します。チェックボックスを選択して、現在選択されているプロジェクト構成の構築からファイルを除外することもできます。
3. 「C コンパイラ」ノードをクリックして、プロジェクトコンパイラ設定およびこのファイルのその他のプロパティーをオーバーライドできることを確認します。
4. 「プロジェクトのプロパティー」ダイアログボックスをキャンセルします。

プロジェクトの構築

プロジェクトを構築するには、次の手順に従います。

1. プロジェクトを右クリックして、「構築」を選択します。プロジェクトが構築されます。構築生成物が「出力」ウィンドウに表示されます。



2. メインツールバーの「構成」ドロップダウンリストで、構成を「デバッグ」から「パフォーマンス・リリース」に変更します。パフォーマンスリリース構成を使用してプロジェクトが構築されます。
3. プロジェクトを右クリックして、「構築」を選択します。プロジェクトが構築されます。構築生成物が「出力」ウィンドウに表示されます。

プロジェクトの複数の構成を同時に構築するには、「実行」>「主プロジェクトをバッチ構築」を選択して、「バッチ構築」ダイアログボックスで構築する構成を選択します。

プロジェクトを右クリックしてメニューからアクションを選択して、プロジェクトを構築、クリーン、およびクリーンと構築の両方を実行できます。プロジェクトにはオブジェクトファイルと実行可能ファイルが構成ごとに保管されるため、複数の構成でファイルが混在する心配はありません。

単体のファイルのコンパイル

単体のソースファイルをコンパイルするには、次の手順に従います。

- `newmain.c` ファイルを右クリックして、「ファイルのコンパイル」を選択します。このファイルのみがコンパイルされます。

♦ ♦ ♦ 第 3 章

3 プロジェクトの実行

この章は次のセクションで構成されており、IDE でのプロジェクトの実行方法を説明します。

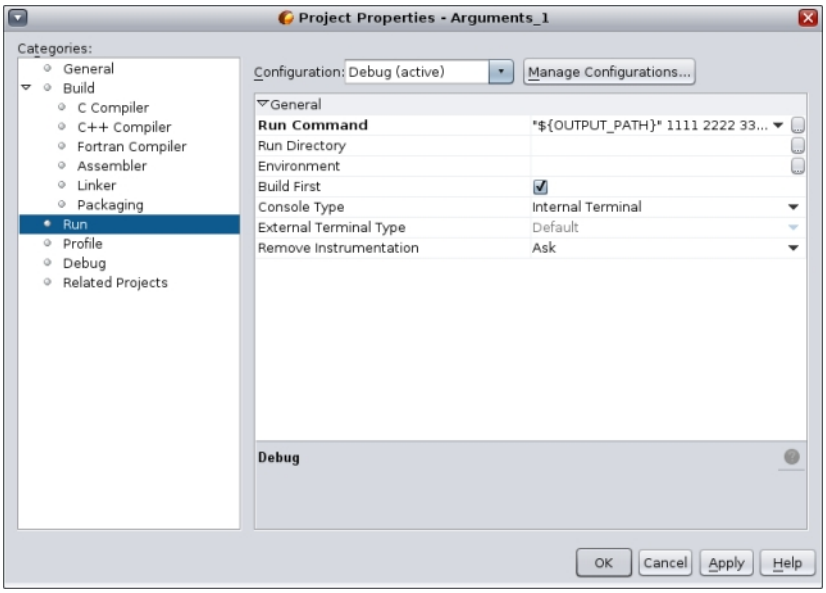
- 19 ページの「プロジェクトの実行」
- 20 ページの「起動ツール」

プロジェクトの実行

`Arguments` サンプルプログラムは、コマンドライン引数を出力します。このプログラムを実行する前に、現在の構成で引数を設定します。その後プログラムを実行します。

`Arguments_1` プロジェクトを作成するには、引数を設定してプロジェクトを実行します。

1. 「ファイル」>「新規プロジェクト」を選択します。
2. プロジェクトウィザードで、「サンプル」カテゴリを展開します。
3. 「C/C++」サブカテゴリを選択して、`Arguments` プロジェクトを選択します。「次へ」をクリックして、「完了」をクリックします。
4. `Arguments_1` プロジェクトノードを右クリックして、「構築」を選択します。プロジェクトが構築されます。
5. `Arguments_1` プロジェクトノードを右クリックして、「プロパティー」を選択します。
6. 「プロジェクトのプロパティー」ダイアログボックスで、「実行」ノードを選択します。
7. 「コマンドを実行」テキストフィールドで、出力パスのあとに `1111 2222 3333` と入力します。「OK」をクリックします。



8. 「実行」>「主プロジェクトを実行」を選択します。アプリケーションが構築され、実行されます。引数は外部ウィンドウに表示されます。

起動ツール

「起動ツール」を作成することで、たとえば簡単にプロジェクトをプロジェクトコンテキストメニューから別の引数を使用して実行したり、プロジェクトをスクリプトから起動したりできます。起動ツールの詳細については、『Oracle Solaris Studio 12.4 リリースの新機能 』の「IDE の新しい起動ツール機能」を参照してください。

♦ ♦ ♦ 第 4 章

4 既存のソースからのプロジェクトの作成

また、Makefile をすでに含んでいたり、`configure` スクリプトを実行するときに Makefile をビルドしたりする既存のソースファイルからプロジェクトを作成することもできます。このタイプのプロジェクトは、管理対象外プロジェクトと呼ばれます。プロジェクトおよび Makefile の詳細については、IDE ヘルプの「C/C++/Fortran プロジェクトおよび Makefile」を参照してください。

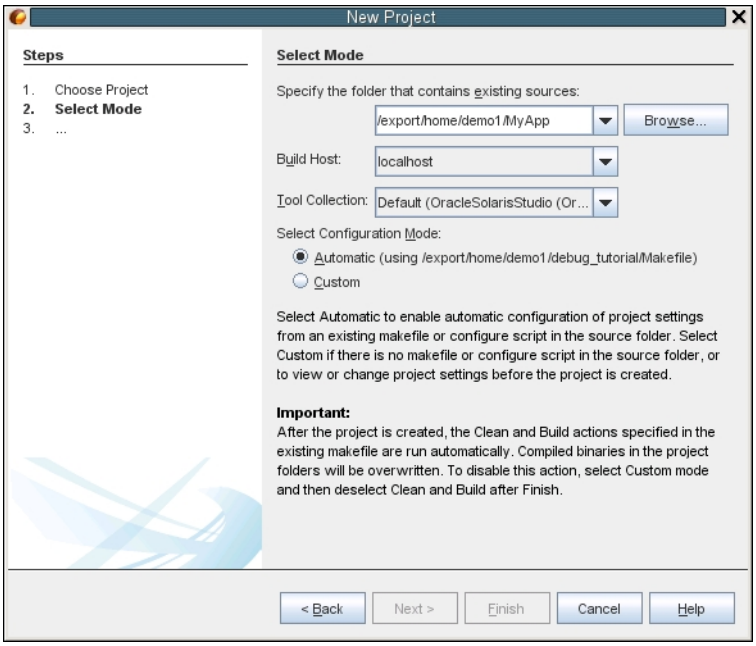
この章は次のセクションで構成されています。

- 21 ページの「既存のソースからのプロジェクトの作成」
- 23 ページの「バイナリファイルからのプロジェクトの作成」
- 25 ページの「コード支援」

既存のソースからのプロジェクトの作成

「既存のソースからの C/C++/Fortran プロジェクト」では、IDE は既存のメイクファイルの命令を使用してアプリケーションをコンパイルおよび実行します。

1. 「ファイル」>「新規プロジェクト」を選択します。
2. C/C++/Fortran カテゴリを選択します。
3. 「既存のソースからの C/C++/Fortran プロジェクト」を選択して「次へ」をクリックします。
4. 新規プロジェクトウィザードの「モードを選択」ページで、「参照」ボタンをクリックします。「プロジェクトフォルダを選択」ダイアログボックスで、ソースコードがあるディレクトリに移動します。「選択」をクリックします。



5. デフォルトの構成モード「自動」を使用します。「完了」をクリックします。
6. プロジェクトが作成され、「プロジェクト」ウィンドウで開きます。また、既存のメイクファイルで指定された Clean セクションと Build セクションを IDE が自動的に実行します。プロジェクトにコード支援が自動的に構成されます。

既存のコードの thin ラッパーとなるプロジェクトが作成されました。

プロジェクトの構築と再構築

プロジェクトを構築するには、次の手順に従います。

- プロジェクトのプロジェクトノードを右クリックして、「構築」を選択します。

プロジェクトを再構築するには、次の手順に従います。

- プロジェクトのプロジェクトノードを右クリックして、「生成物を削除して構築」を選択します。

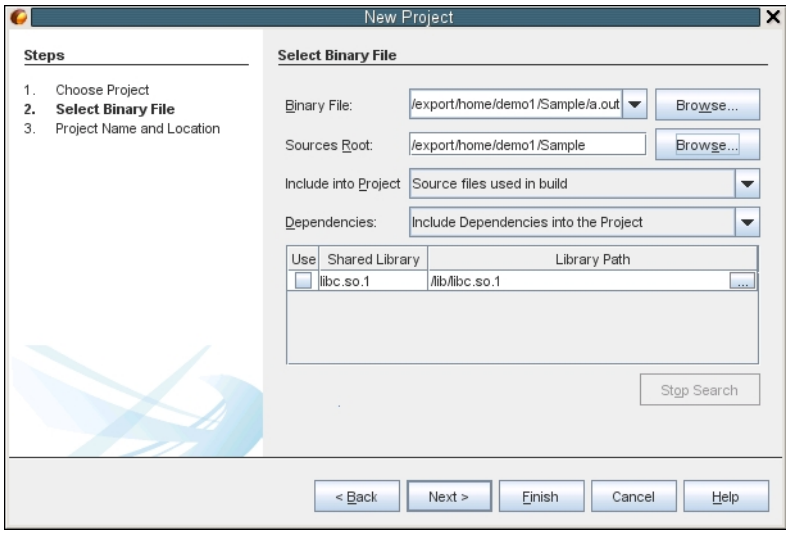
バイナリファイルからのプロジェクトの作成

バイナリファイルの C/C++/Fortran プロジェクトを使用すると、複数の方法で既存のバイナリファイルからプロジェクトを作成できます。

新規プロジェクトウィザードを使用したバイナリファイルからのプロジェクトの作成

1. 「ファイル」>「新規プロジェクト」を選択します。
2. C/C++/Fortran カテゴリを選択します。
3. 「バイナリファイルからの C/C++/Fortran プロジェクト」を選択して「次へ」をクリックします。
4. 新規プロジェクトウィザードの「バイナリファイルを選択」ページで、「参照」ボタンをクリックします。「バイナリファイルを選択」ダイアログボックスで、プロジェクトの作成元となるバイナリファイルに移動します。

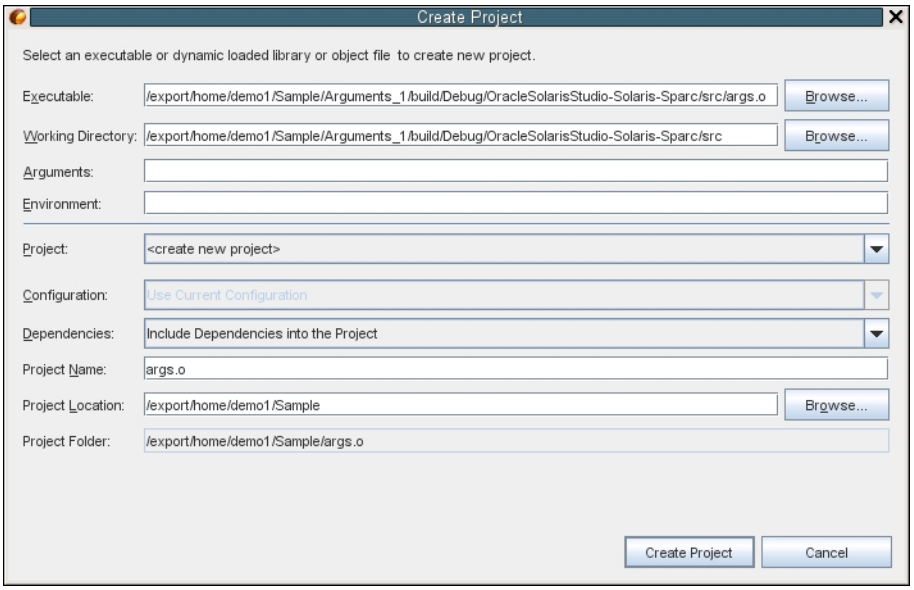
   バイナリの構築元となったソースファイルのルートディレクトリは自動的に入力されます。デフォルトでは、バイナリの構築元となったソースファイルのみがプロジェクトに含まれています。デフォルトでは、依存関係がプロジェクトに含まれます。プロジェクトで必要な共有ライブラリは自動的に一覧表示されます。



5. 「次へ」をクリックします。
6. 「プロジェクトの名前と場所」ページでは、プロジェクトの名前と場所を選択できます。「完了」をクリックします。

「ファイル」ウィンドウでのバイナリファイルからのプロジェクトの作成

1. 「ファイル」ウィンドウで Arguments_1 プロジェクトに移動し、構築ノードを `args.o` まで展開します。
2. 次のいずれかの方法でプロジェクトの作成ウィザードを起動します。
   - `args.o` ファイルを右クリックし、「プロジェクトを作成...」をクリックします。
   - `args.o` ファイルをソースエディタにドラッグします。



「実行可能ファイル」と「作業ディレクトリ」が事前に入力された状態で、プロジェクトの作成ウィザードが表示されます。

3. プロジェクトの実行時に使用するすべての引数と環境変数を入力します。「プロジェクト」ドロップダウンメニューから「<新規プロジェクトを作成>」を選択します。

右下隅にある「プロジェクトの作成」ボタンをクリックすると、「プロジェクト」タブで Args.o という名前の新規プロジェクトが作成されます。

4. 「取消し」をクリックします。

コード支援

コード支援とは、特に管理対象外プロジェクトのソースコードのナビゲートと編集を支援する IDE 機能のセットです。編集とナビゲートの詳細については、第6章「ソースファイルの編集とナビゲート」を参照してください。

管理対象外プロジェクトの場合は、コードの解析方法を指定すると IDE のコード支援機能を有効にできます。このセクションでは、コード支援の構成方法とコード支援キャッシュの共有について説明します。

コード支援の構成

プロジェクトを作成するときに、構成を作成するために組み込みパーサーによって使用されるインクルードファイルとマクロ定義を指定できます。プロジェクトがデバッグ情報付きで構築されている場合、コンパイルされた各ソースファイルのインクルードファイルおよびマクロ定義を、組み込みパーサーで自動的に検索できます。

プロジェクトに対する IDE のコード支援機能の精度を向上させる、追加のコード支援構成情報を指定するには、「コード支援を構成」ウィザードを使用します。ウィザードを起動するには、プロジェクトを右クリックして「コード支援を構成」を選択します。コード支援の構成と「コード支援を構成」ウィザードの詳細については、IDE の該当するヘルプセクションを参照してください。

コード支援キャッシュの共有

C/C++ ソースコードの解析時に、IDE は解析結果をディスク上のコード支援キャッシュに保存します。プロジェクトを開くと、IDE はキャッシュを検証し、キャッシュが最新かどうかを確認します。キャッシュが最新の場合、IDE はプロジェクトを解析せず、コードのナビゲーションに必要なデータをコード支援キャッシュからロードします。

デフォルトでは、IDE はユーザーディレクトリ内の `$`*userdir*`/var/cahche` フォルダに、ユーザーのすべてのプロジェクトを対象とした 1 つのコード支援キャッシュを作成します。ユーザーディレクトリ内のキャッシュを別の場所にコピーまたは共有することはできません。

ただしコード支援キャッシュがプロジェクト内部に配置されている場合、コードが解析されたコンピュータと同一のオペレーティングシステムが稼働しており、かつプロジェクトで使用するツールコレクションが同じ場所にある別のコンピュータに、キャッシュをコピーできます。

コード支援キャッシュの共有の詳細と、IDE に対してプロジェクトメタデータ内のコード支援キャッシュを処理するように指示する方法については、『Oracle Solaris Studio 12.4 リリースの新機能 』の「コード支援キャッシュの共有」および IDE ヘルプのバージョン制御プロジェクトのコード支援キャッシュの移動に関する項目を参照してください。

コード支援のプロジェクトのプロパティーのオプション

管理対象外のプロジェクトをバージョン管理システムで容易に使用できるようにするため、IDE には次のプロジェクトのプロパティーがあります。

一時マクロ	揮発性マクロのリストを指定できます (`-D` オプション)。これらは時間、日付、または環境によって異なります。これらのマクロ環境変数値は、プロジェクトのパブリックメタデータとともに保存されません。
ユーザー環境変数	プロジェクトがシステム固有のパスを渡すときに使用する環境変数のリストを指定できます。これらのマクロ環境変数値は、プロジェクトのパブリックメタデータとともに保存されません。管理対象外プロジェクトの場合は、プロジェクトメタデータの保存時に使用する環境変数のリストを指定できます。IDE がコンパイラオプションを格納し、オプション値が変数値と一致する場合は、代わりにマクロが記述されます。

ファイルシステムから C/C++ ヘッダーファイルを検索

ソースがまだ構築されておらず、デバッグ情報が含まれていない管理対象外プロジェクトを作成すると、IDE ではコード支援の構成で問題が発生することがあります。この場合は、「コード支援を構成」ウィザードで、特殊モード (ファイルシステムから C/C++ ヘッダーファイルを検索) を使用するよう指定できます。このモードでは、IDE はファイルシステムからヘッダーを検索することによって、失敗した include ディレクティブを解決しようとします。ウィザードで、ヘッダーを検索するためのパスを入力するよう求められます。デフォルトでは、このパスはプロジェクトソースルートです。

♦ ♦ ♦ 第 5 章

5 Oracle Database プロジェクトの作成

この章では、IDE で Oracle Database プロジェクトを作成する方法を説明し、Oracle Database プロジェクトと IDE での Oracle Database プロジェクトの使用方法を説明する参照資料を示します。Oracle Database プロジェクトと IDE のその他の参照資料については、99 ページの「Oracle Database プロジェクト」を参照してください。

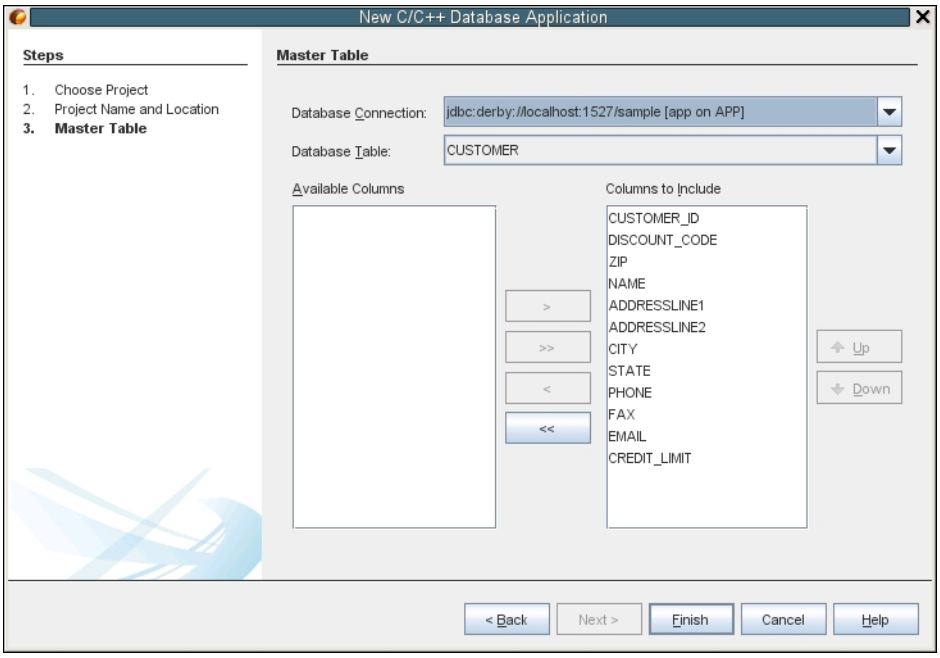
Oracle Database プロジェクトの作成

Oracle Database アプリケーション用のプロジェクトを作成できます。これを実行するには、使用中の Oracle Solaris Studio インストールに、省略可能な Oracle Instant Client コンポーネントが含まれている必要があります。

1. 「ファイル」>「新規プロジェクト」を選択します。
2. 「新規プロジェクト」ダイアログボックスで、「C/C++/Fortran」カテゴリを選択し、「C/C++ データベースアプリケーション」プロジェクトを選択します。「次へ」をクリックします。
3. 「プロジェクトの名前と場所」ページでは、プロジェクトの名前と場所を選択できます。「次へ」をクリックします。
4. 「マスターテーブル」ページで、「データベース接続」ドロップダウンリストから `jdbc:derby://localhost:1527/sample` を選択します。IDE はデータベースに接続します。

   「データベース接続」の下にデータベースが表示されていない場合は、しばらくしてからもう一度実行してください。Derby は JDK 6 以降のバージョンに組み込まれているデータベースです。

   新規データベース接続を作成する場合は、___ を参照してください。
5. 「データベーステーブル」ドロップダウンリストからプロジェクトのマスターテーブルを選択します。
6. 「使用可能な列」と「含める列」の一覧の間にある矢印キーを使用して、プロジェクトに含めるテーブル列を選択します。

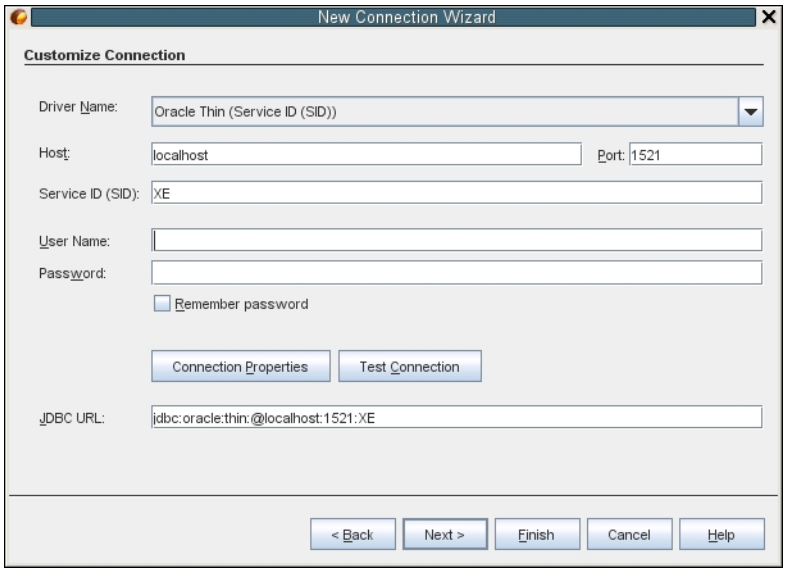


7. 「完了」をクリックします。

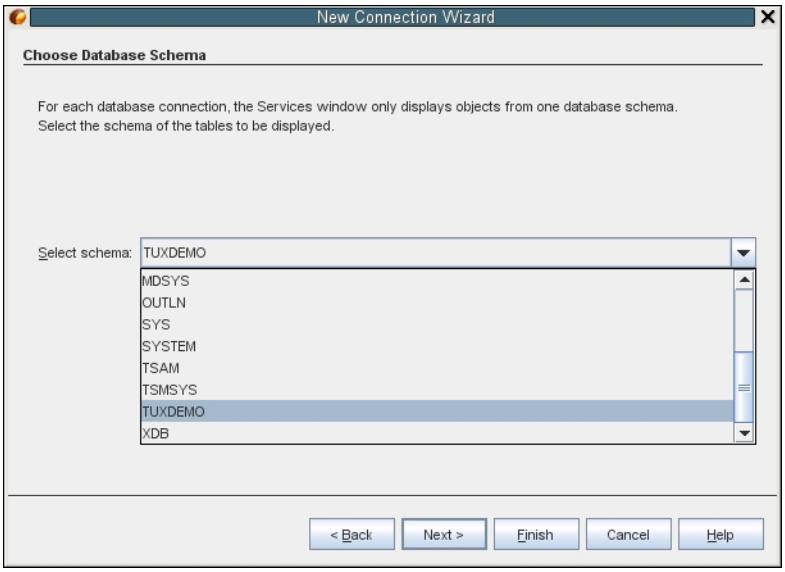
新規データベース接続の作成

Oracle データベースへの新規接続を設定するには、次の手順に従います。

1. 「マスターテーブル」ページで「データベースの新規接続」を選択して、新規データベース接続を作成します。



2. 新規接続ウィザードの「接続をカスタマイズ」ページで、データベースのアドレス、ユーザー名、パスワードを入力し、「次へ」をクリックします。
3. 新規接続ウィザードの「データベース・スキーマを選択」ページで、データベーススキーマを入力するかリストからスキーマを選択します。「完了」をクリックします。


New Connection Wizard
Choose Database Schema
For each database connection, the Services window only displays objects from one database schema.
Select the schema of the tables to be displayed.
Select schema:
TUXDEMO
MDSYS
OUTLN
SYS
SYSTEM
TSAM
TSMSYS
TUXDEMO
XDB
< Back
Next >
Finish
Cancel
Help

♦ ♦ ♦ 第 6 章

6 ソースファイルの編集とナビゲート

IDE では、エディタでソースファイルを編集およびナビゲートできます。この章は次のセクションで構成されています。

- 31 ページの「ソースファイルの編集」
- 44 ページの「ソースファイルのナビゲーション」

ソースファイルの編集

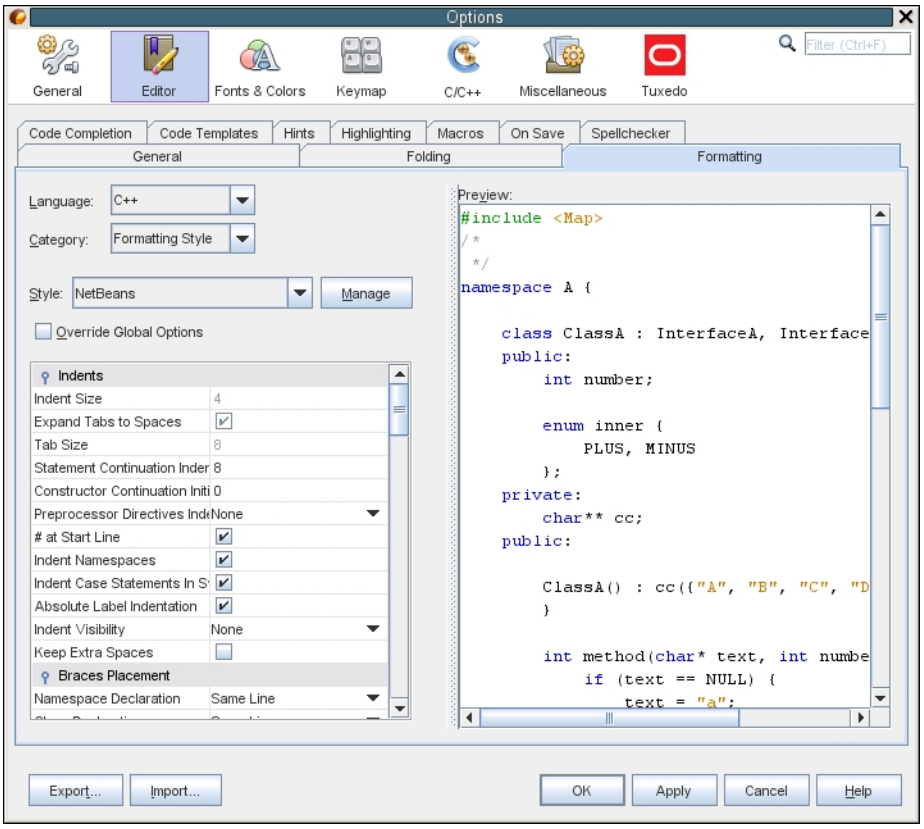
Oracle Solaris Studio IDE には高度な編集機能およびコード支援機能があり、ソースコードの表示と変更に役立ちます。これらの機能を確認するため、`Quote` プロジェクトを使用します。

1. 「ファイル」>「新規プロジェクト」を選択します。
2. プロジェクトウィザードで、「サンプル」カテゴリと「C/C++」サブカテゴリを展開して、`Quote` プロジェクトを選択します。「次へ」をクリックして、「完了」をクリックします。

書式設定スタイルの設定

「オプション」ダイアログボックスを使用して、プロジェクトのデフォルトの書式構成スタイルを構成できます。

1. 「ツール」>「オプション」を選択します。
2. ダイアログボックスの上部ペインの「エディタ」をクリックします。
3. 「書式設定」タブをクリックします。
4. 「言語」ドロップダウンリストから、書式設定スタイルを設定する言語を選択します。
5. 「スタイル」ドロップダウンリストから、設定するスタイルを選択します。



6. 必要に応じてスタイルプロパティーを変更します。

C および C++ ファイルでのコードのブロックの折り畳み

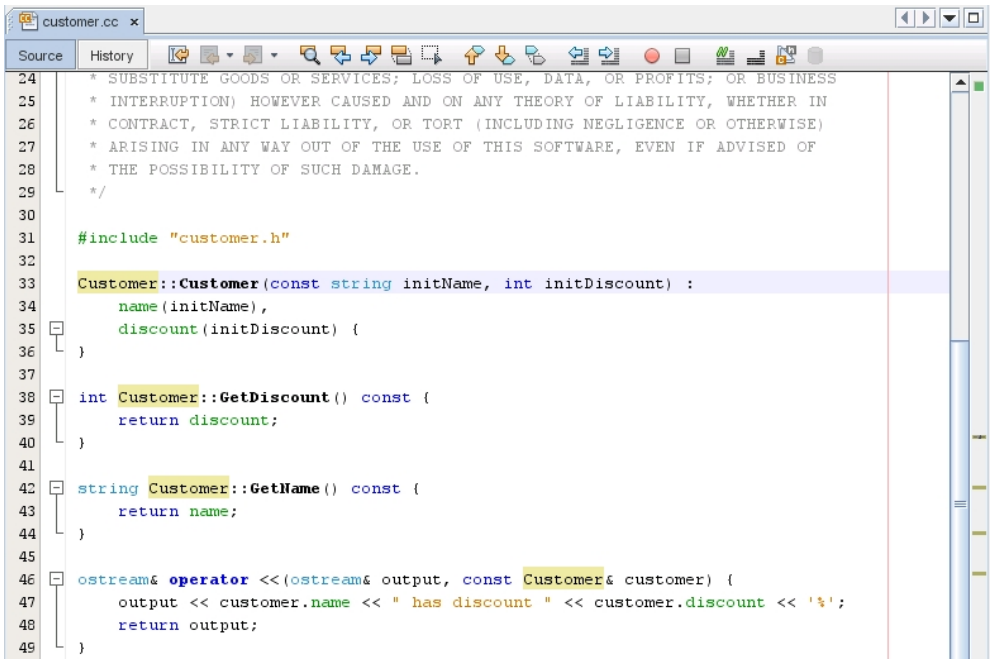
一部のタイプのファイルでは、コード折り畳み機能を使用して、コードのブロックを折りたたんでブロックの最初の行のみをソースエディタに表示できます。

1. `Quote_1` アプリケーションプロジェクトで、「ソースファイル」フォルダを開き、`cpu.cc` ファイルをダブルクリックしてソースエディタで開きます。
2. 左端の折り畳みアイコン (マイナス記号付きの小さなボックス) をクリックして、メソッドの 1 つのコードを折り畳みます。
3. 折り畳んだブロックの右側の {...} 記号にマウスオーバーして、ブロック内のコードを表示します。

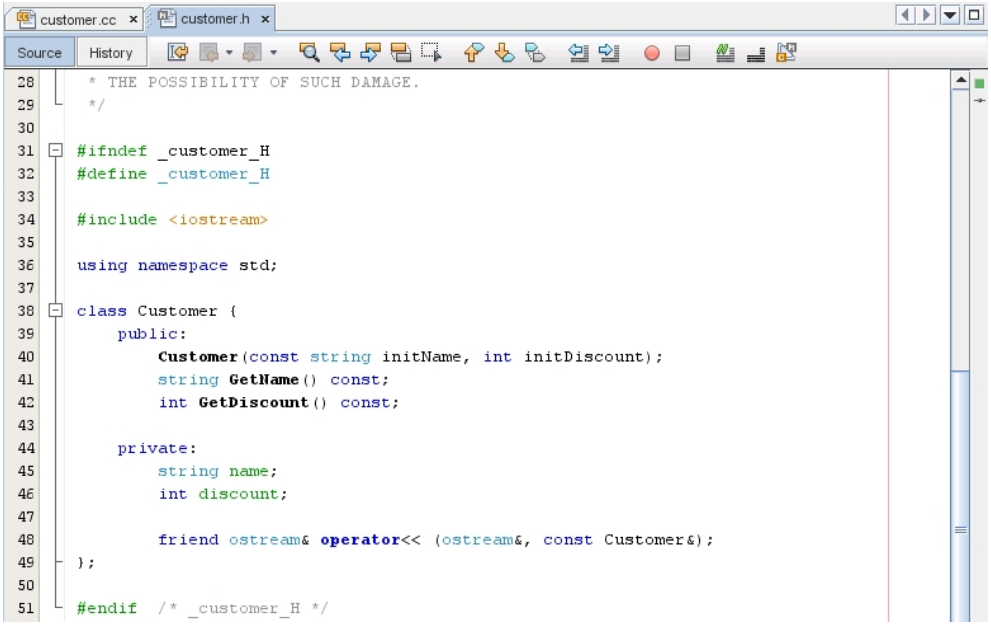
意味解釈の強調表示の使用

オプションを設定して、クラス、関数、変数、もしくはマクロをクリックしたときに、現在のファイル内のこのクラス、関数、変数、もしくはマクロのすべての出現箇所が強調されるようにできます。

1. 「ツール」>「オプション」を選択します。
2. ダイアログボックスの上部ペインの「C/C++」をクリックします。
3. 「強調表示」タブをクリックします。
4. すべてのチェックボックスがチェックされていることを確認します。
5. 「OK」をクリックします。
6. `Quote_1` プロジェクトの `customer.cc` ファイルで、関数名がボールドで強調表示されていることを確認します。
7. `Customer` クラスの出現箇所をクリックします。
8. ファイル内の `Customer` クラスのすべての出現箇所が黄色の背景で強調表示されます。



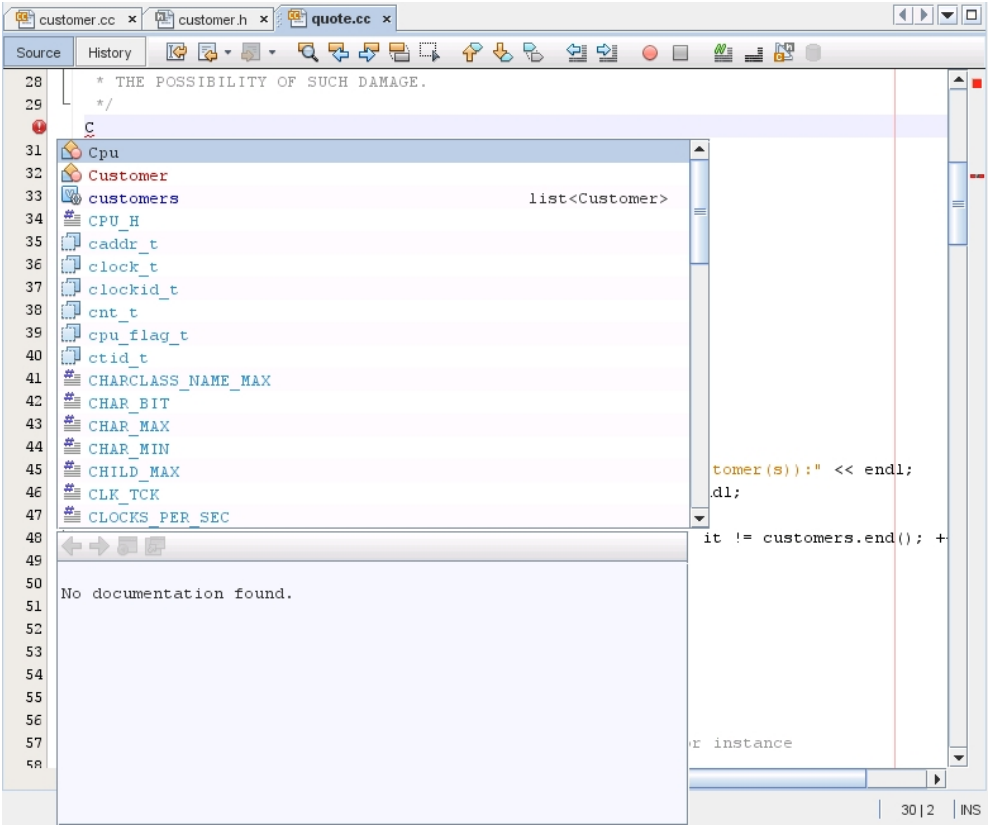
9. `customer.h` ファイルで、クラスフィールドがボールドで強調表示されていることを確認します。



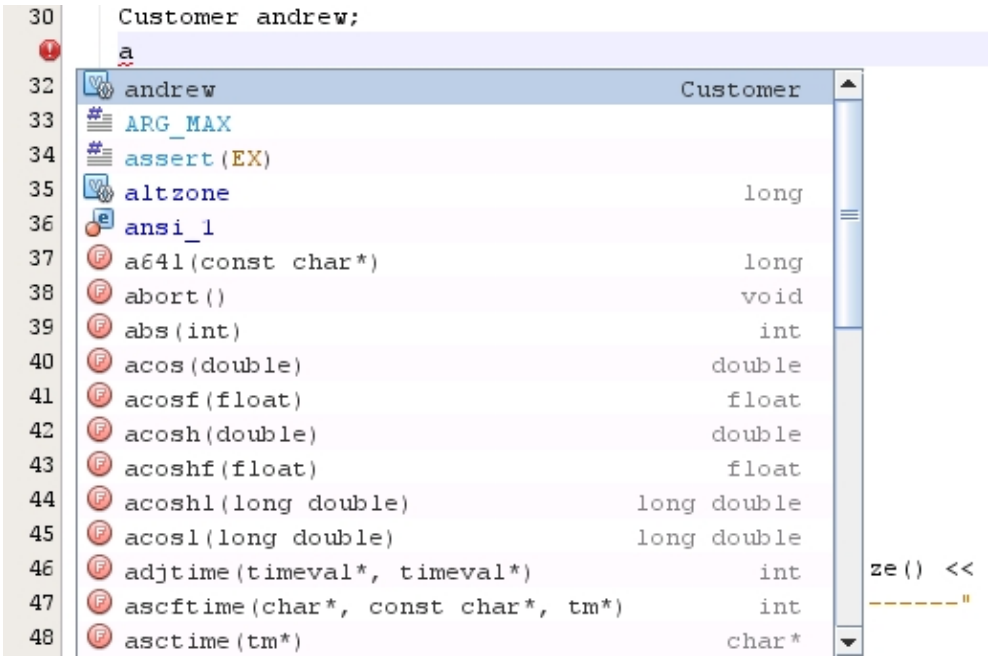
コード補完の使用

IDE には C および C++ の動的コード補完機能があり、1 文字以上を入力すると該当するクラス、メソッド、変数などのリストが表示され、これを使用して式を補完できます。

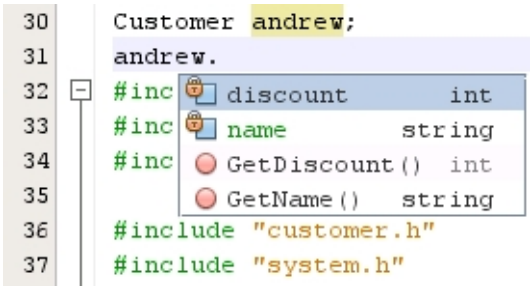
1. `Quote_1` プロジェクトの `quote.cc` ファイルを開きます。
2. `quote.cc` ファイルの最初の空行で、大文字の C を」入力して Ctrl-Space を押します。コード補完ボックスに、`Cpu` および `Customre` クラスを含む短いリストが表示されます。ドキュメントウィンドウも開き、プロジェクトソースコードにドキュメントがないため、「ドキュメントがありません」というメッセージが表示されます。
3. Ctrl-Space をもう一度押して、コード補完リストを展開します。



4. `calloc()` などの標準ライブラリ関数をリストから選択すると、ドキュメントウィンドウにその関数のマニュアルページが表示されます (IDE でマニュアルページにアクセスできる場合)。
5. `Customer` クラスを選択して、Enter を押します。
6. `andrew;` と入力して、`Customer` クラスの新しいインスタンスを完成させます。次の行で、文字 `a` を入力して Ctrl-Space を押します。コード補完ボックスに `andrew` が表示されます。Ctrl-Space をもう一度押すと、コード補完ボックスに、メソッド引数、クラスフィールド、およびグローバル名など、現在のコンテキストでアクセスできる、文字 a で始まる選択対象のリストが表示されます。



7. andrew オプションをダブルクリックして結果を受け入れ、その後にピリオドを入力します。Customer クラスのパブリックメソッドとフィールドのリストが自動的に指定されます。

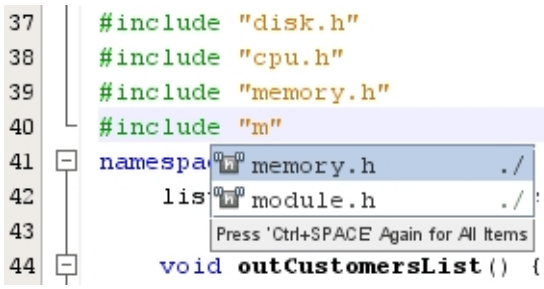


8. 追加したコードを削除します。

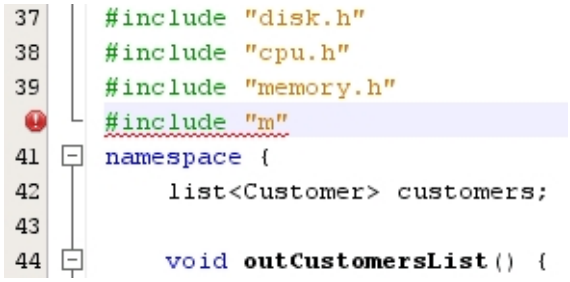
静的コードエラー検査の使用

ソースエディタ内でソースまたはヘッダーファイルにコードを入力するとき、エディタはユーザーの入力中に静的コードエラー検査を実行し、エラーを検出すると、左マージンにエラーアイコン を表示します。

1. Quote_1 プロジェクトの quote.cc ファイル内で、40 行目に #include "m と入力します。コード補完ボックスが表示され、m で始まる 2 つのヘッダーファイルが推奨されます。



2. ソースエディタ内で追加したコードから離れた位置をクリックします。マージンにエラーアイコンが表示されます。



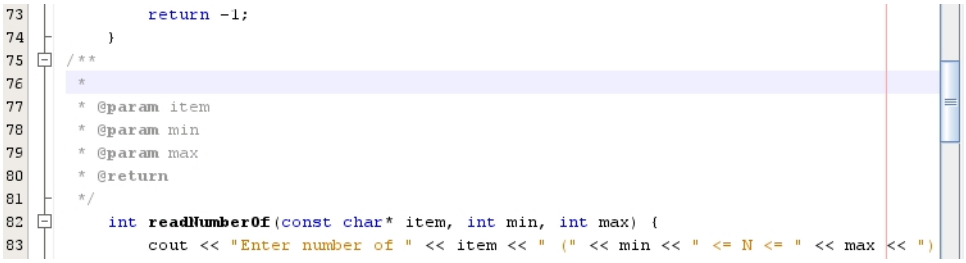
3. 2 番目の引用符をバックスペースで削除し、odule.h" と入力して文を完成させると、文が既存のヘッダーファイルを参照した直後にエラーアイコンが表示されなくなるのがわかります。
4. 追加した文を削除します。

確認するエラーを選択する方法、または静的コードエラー検査を無効にする方法の詳細については、IDE の該当するヘルプページを参照してください。

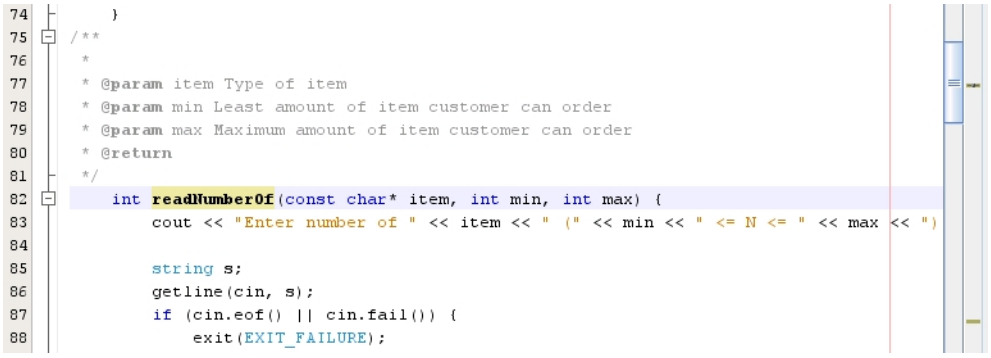
ソースコードドキュメントの追加

コードにコメントを追加して、関数、クラス、およびメソッドのドキュメントを生成できます。IDE は Doxygen 構文を使用するコメントを認識して、ドキュメントを自動的に生成します。また、コメントブロックを自動的に生成して、コメントの下の関数のドキュメントを作成します。

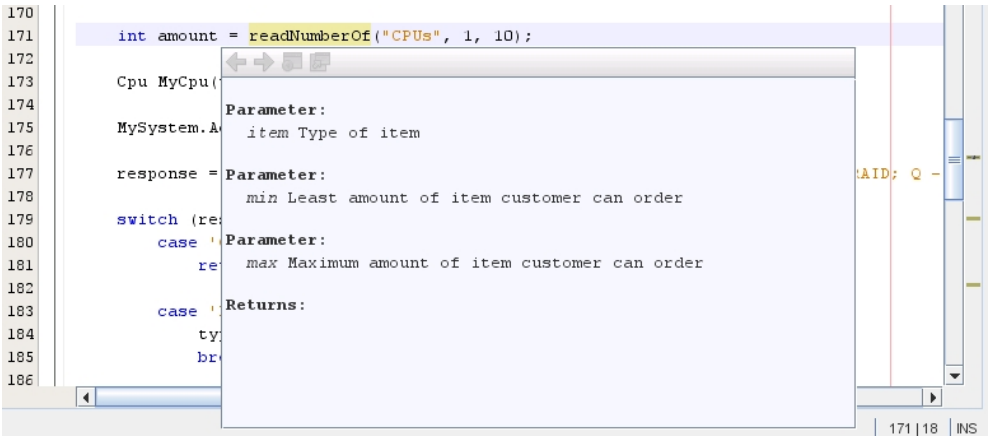
1. `quote.cc` ファイルで、行 `int readNumberOf(const char* item, int min, int max) {` の上の行にカーソルを置きます。
2. スラッシュと 2 つのアスタリスク (/**) を入力し、Enter を押します。エディタによって、`readNumberOf` クラスに Doxygen 形式のコメントが挿入されます。



3. `@param` の各行に説明のテキストを追加して、ファイルを保存します。
4. `readNumberOf` クラスをクリックして黄色で強調表示し、右側の出現箇所マークの 1 つをクリックしてクラスが使用されている場所にジャンプします。



5. ジャンプ先の `readNumberOf` クラスをクリックして、Ctrl-Shift-Space を押してパラメータに追加したドキュメントを表示します。



6. ファイル内の任意の場所をクリックしてドキュメントウィンドウを閉じて、readNumberOF クラスを再度クリックします。
7. 「ソース」>「ドキュメントの表示」を選択して、クラスのドキュメントウィンドウを再度開きます。

コードテンプレートの使用

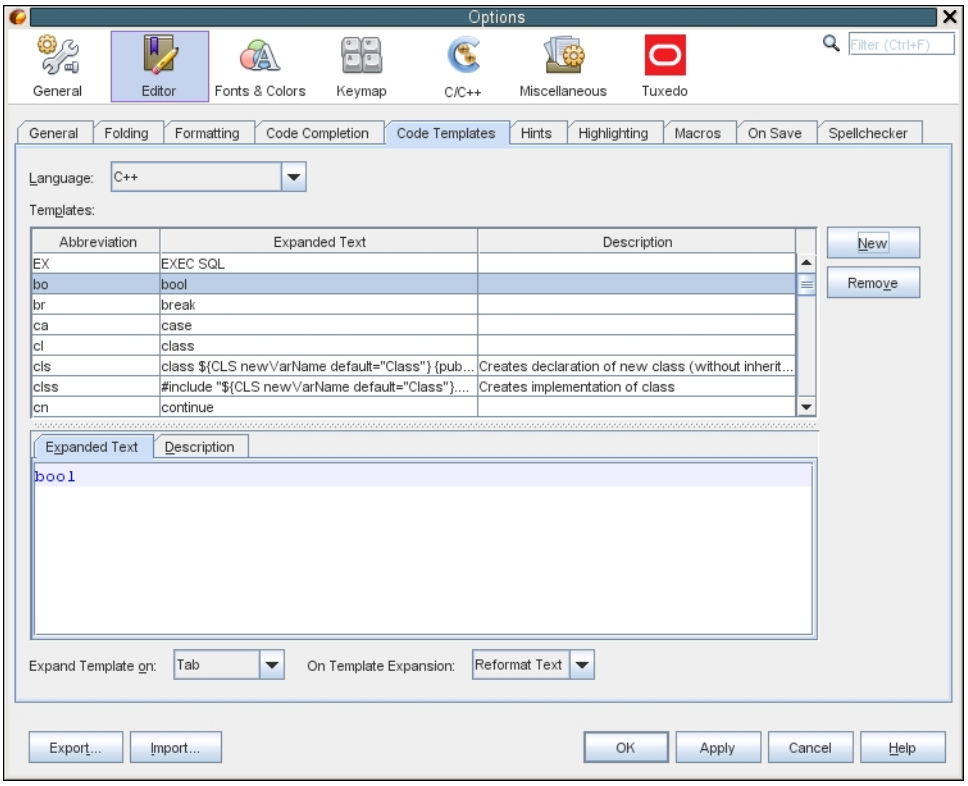
ソースエディタには、C、C++、および Fortran コードの共通スニペット用の、一連のカスタマイズ可能なコードテンプレートがあります。略語を入力して Tab キーを押すと、コードスニペット全体を生成できます。たとえば、Quote_1 プロジェクトの quote.cc ファイルで、次のように操作します。

- uns と入力した後に Tab キーを押して、uns を unsigned に展開します。
- iff と入力した後に Tab キーを押して、iff を if (exp) {} に展開します。
- ifs と入力した後に Tab キーを押して、ifs を if (exp) {} else {} に展開します。
- fori と入力した後に Tab キーを押して、fori expands を for (int i=0; i< size; i++) { Object size = array[i]; } に展開します。

使用できるコードテンプレートすべてを表示するには、テンプレートを変更してユーザー固有のコードテンプレートを作成するか、または別のキーを選択してテンプレートを展開します。

1. 「ツール」>「オプション」を選択します。
2. 「オプション」ダイアログボックスで、「C/C++」を選択して、「コードテンプレート」タブをクリックします。

3. 「言語」ドロップダウンリストから言語を選択します。
4. 「新規」ボタンまたは「削除」ボタンを使用して項目を追加または削除します。「展開されるテキスト」タブで現在のテンプレートを編集したり、「説明」タブでテンプレートの説明を追加したりすることもできます。



ペア補完の使用

C および C++ ソースファイルを編集すると、ソースエディタは角括弧、丸括弧、および引用符など、ペアで使用される文字の「スマート」照合を実行します。これらの文字の片方を入力すると、ソースエディタはもう片方の文字を自動的に挿入します。

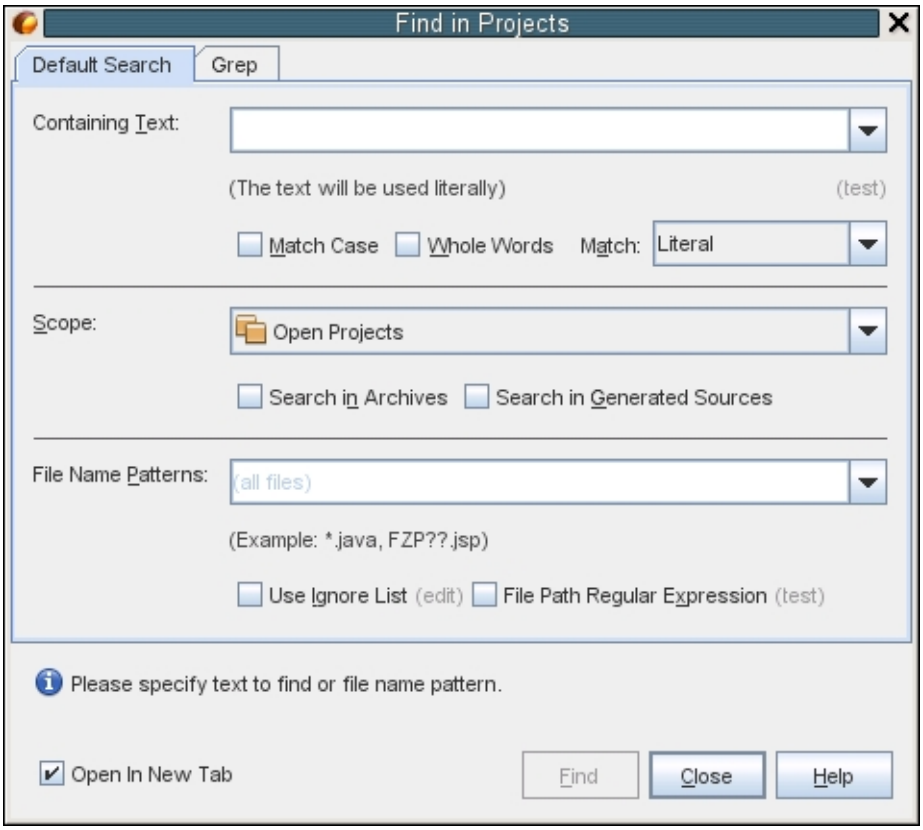
1. `Quote_1` プロジェクトで、`module.cc` ファイルの 116 行目にある `{` の後にカーソルを置き、Return を押して新しい行を開きます。

2. `enum state {` と入力して Return を押します。閉じる大括弧とセミコロンが自動的に追加され、カーソルが括弧の間に置かれます。
3. `invalid=0, success=1` と入力して、列挙法を補完します。
4. 列挙の閉じる `};` の後の行で、`if` と入力します。閉じ括弧が自動的に追加され、カーソルが括弧の間に置かれます。
5. 「`v==null`」と入力します。右側の括弧の後に、`i` と改行を入力します。閉じ括弧が自動的に追加されます。
6. 追加したコードを削除します。

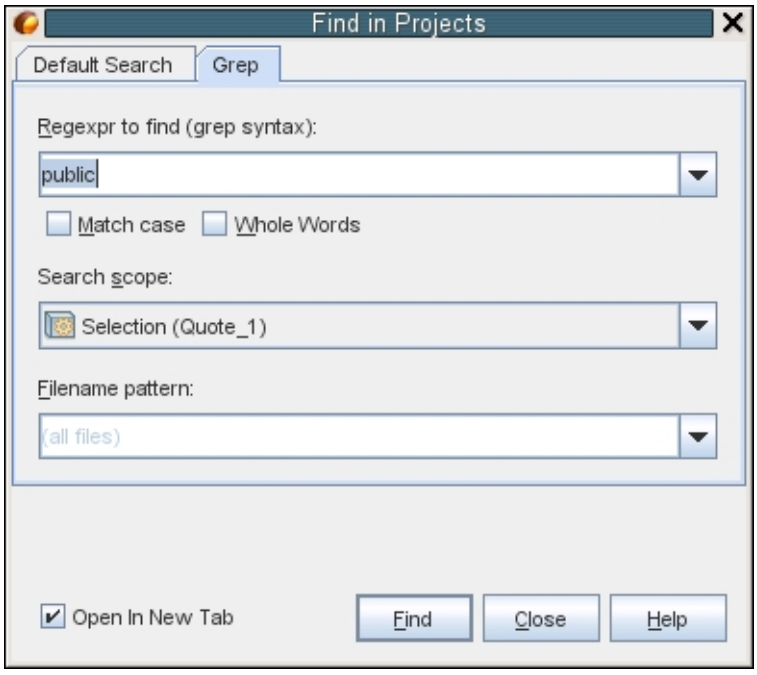
プロジェクトファイルでのテキストの検索

「プロジェクト内を検索」ダイアログボックスを使用すると、プロジェクトで特定のテキストまたは正規表現のインスタンスを検索できます。

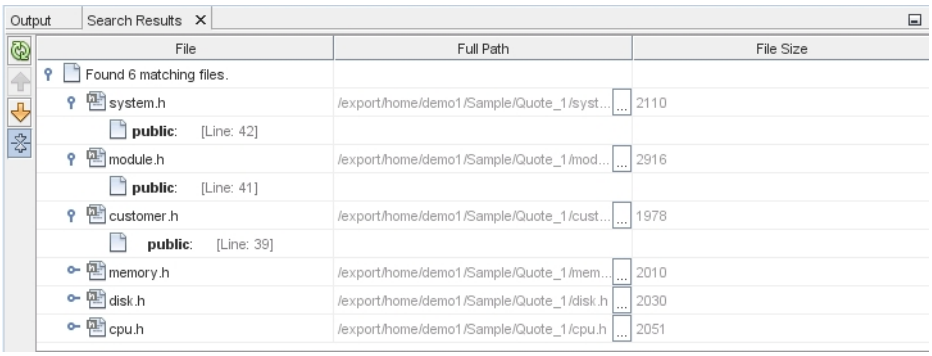
1. 「プロジェクト」ウィンドウでプロジェクトを右クリックして「検索」を選択したり、「編集」>「プロジェクト内を検索」(Ctrl+Shift+F) を選択して、「プロジェクト内を検索」ダイアログボックスを開きます。
2. 「プロジェクト内を検索」ダイアログボックスで、「デフォルト検索」タブまたは「Grep」タブを選択します。「Grep」タブでは、特にリモートプロジェクトに対して高速検索を提供する`grep`ユーティリティーが使用されます。

3. 

4. 「Grep」タブで、検索するテキストまたは正規表現を入力し、検索範囲およびファイル名パターンを指定して、複数の検索を別々のタブで保存できるように「新規タブで開く」チェックボックスを選択します。



5. 「検索」をクリックします。



「検索結果」タブには、該当のテキストまたは正規表現が検出されたファイルが一覧表示されます。

左マージンのボタンを使用すると、検索結果の表示を変更できます。

6. 「展開/縮小」ボタンをクリックして、ファイル名のみが表示されるようにファイルのリストを縮小します。
7. リスト内の項目の 1 つをダブルクリックすると、IDE ではソースエディタ内の該当する場所まで移動できます。

検索および置換

エディタでは検索/置換機能を使用でき、この機能は、置換用の個別のダイアログボックスの代わりに、一貫してエディタウィンドウ下部の「検索」ツールバーで機能します。「置換」フィールドとボタンは、ツールバーの「検索」フィールドとボタンの下に表示されます。「検索」ツールバーをアクティブにするには Ctrl+F を押し、「置換」機能をアクティブにするには Ctrl+H を押します。

クリップボード履歴の使用

必要なテキストが見つかったら、そのテキストをコピーし、ほかのコード領域で使用できます。

デスクトップクリップボードにコピーされたテキストのうち、最新 9 件のテキストバッファーを参照して、貼り付ける行を選択できます。テキストを挿入する位置にカーソルを置いた状態で Ctrl+Shift+D を押すと、クリップボードエントリを表示するポップアップが開きます。矢印キーを使用してクリップボードバッファーをナビゲートし、バッファーのリストの下にあるウィンドウで内容をすべて確認します。バッファーの内容を貼り付けるには、バッファーの番号を入力するか、該当するバッファーが選択されている状態で Enter を押します。このバッファーには、IDE だけではなく、デスクトップ上の任意のウィンドウからコピーされた内容が含まれています。

ヒント - IDE 内で任意のファイルにマウスカーソルを合わせて Alt+Shift+L を押すと、ファイルパスをクリップボードにコピーできます。

ソースファイルのナビゲーション

IDE には、ソースコードを表示する高度なナビゲーション機能があります。これらの機能を確認するため、`Quote_1` プロジェクトを使用します。

ウィンドウの管理とグループ化

個別ウィンドウのほかに、ウィンドウのグループに対してアクションを実行できます。各ウィンドウはグループに属しており、このグループを最小化し、新しい位置にドラッグし、個別ウィンドウにフロートし、IDE ウィンドウに連結できます。

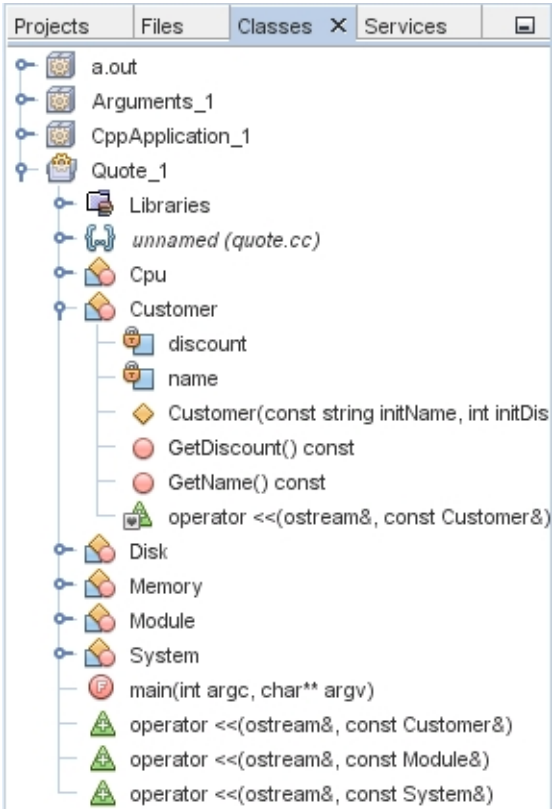
1. グループの右側にある「ウィンドウ・グループの最小化」ボタンをクリックして、左上にある「プロジェクト」、「ファイル」、「クラス」、および「サービス」ウィンドウを最小化します。
2. グループのタブ領域内を右クリックしたり、「ウィンドウ」>「ウィンドウの構成」を選択して「最大化」(Shift+Escape) を選択し、ウィンドウグループを最大化します。

   ウィンドウグループのフロートや連結など、その他のオプションも選択できます。

「クラス」ウィンドウの使用

「クラス」ウィンドウでは、プロジェクトのすべてのクラスと、各クラスのメンバーとフィールドを表示できます。

1. 「クラス」タブをクリックして、「クラス」ウィンドウを表示します。
2. `Quote_1` ノードを展開します。プロジェクト内のすべてのクラスが一覧表示されます。
3. `Customer` クラスを展開します。



4. `name` 変数をダブルクリックして `customer.h` ヘッダーファイルを開きます。

「ナビゲータ」ウィンドウの使用

「ナビゲータ」ウィンドウには、現在選択されているファイルの簡易ビューが表示され、ファイルの異なる部分へのアクセスが簡単になります。「ナビゲータ」ウィンドウが開いていない場合、「ウィンドウ」>「ナビゲート」>「ナビゲータ」(Ctrl-7) を選択して開きます。

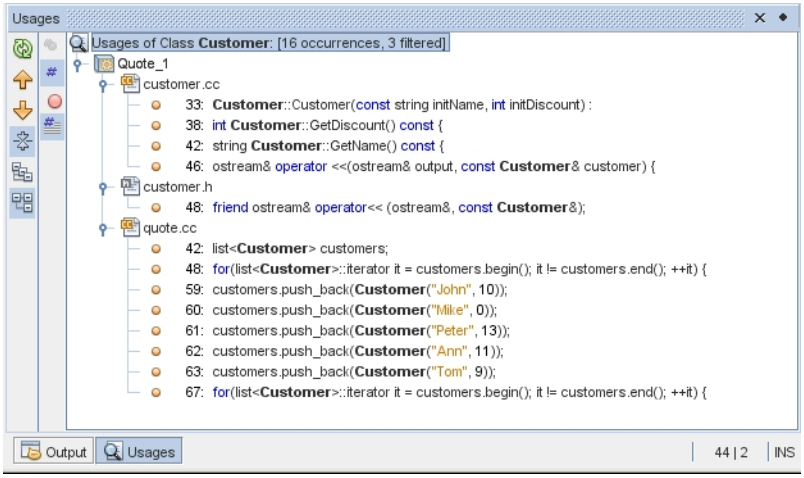
1. エディタウィンドウ内で、`quote.cc` ファイルの任意の場所をクリックします。
2. ファイルの簡易ビューが「ナビゲータ」ウィンドウに表示されます。ウィンドウ上部のノードをクリックして、ビューを展開します。

3. ファイルの要素にナビゲートするには、「ナビゲータ」ウィンドウで要素をダブルクリックして、エディタウィンドウのカーソルをその要素に移動させます。
4. 「ナビゲータ」ウィンドウ内を右クリックして、ウィンドウ内の要素のソート、項目のグループ化、フィルタのオプションを表示させます。
5. 「ナビゲータ」ウィンドウにあるアイコンを確認するには、「ヘルプ」>「ヘルプの目次」を選択して、IDE オンラインヘルプを開きます。ヘルプブラウザで、「検索」タブをクリックして、「検索」フィールドに**ナビゲータアイコン**と入力します。

クラス、メソッド、およびフィールドの使用状況の検出

「使用状況」ウィンドウを使用して、プロジェクトのソースコード内で使用されている、あらゆる場所にあるクラス (構造)、関数、変数、マクロ、もしくはファイルを表示できます。

1. `customer.cc` ファイルで、42 行目の `Customer` クラスを右クリックして「使用状況を検索」(Alt-F7) を選択します。
2. 「使用状況を検索」ダイアログボックスで、「検索」をクリックします。
3. 「使用状況」ウィンドウが開き、プロジェクトのソースファイルでの `Customer` クラスのすべての使用状況が表示されます。

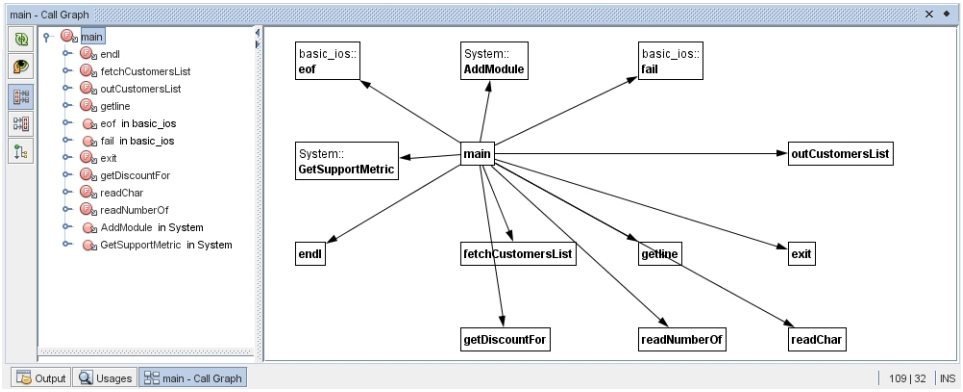


「使用状況を検索」はバックグラウンドで実行されるので、多数のファイルの検索中にほかのタスクを実行できます。検索結果の増加に伴い、「使用状況」ウィンドウに表示される結果が更新されます。進行状況インジケータと検出件数の増分カウントがあります。検索はいつでも停止でき、停止した時点までの検索結果が保存されます。検索一致結果間のナビゲート、論理ビューから物理ビューへの切り替え、異なる設定での「使用状況を検索」の再実行が可能です。

コールグラフの使用

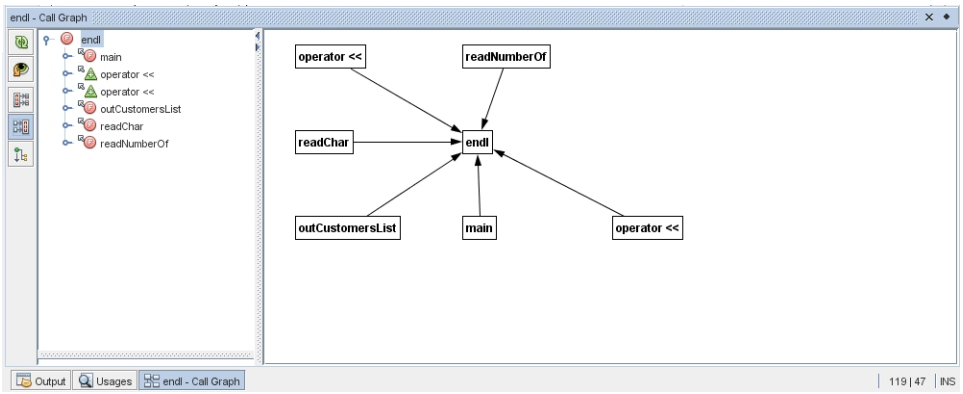
「コールグラフ」ウィンドウには、クラス内の関数の予呼び出し関係の 2 つのビューが表示されます。ツリービューに、選択した関数から呼び出された関数、もしくはこの関数を呼び出す関数が表示されます。グラフィカル表示には、呼び出し元および呼び出し先の関数の間に矢印を使用して、呼び出し関係が示されます。

1. `quote.cc` ファイルで、メイン関数を右クリックして「コールグラフの表示」を選択します。
2. 「コールグラフ」ウィンドウが開き、ツリービューと、`main` 関数から呼び出されるすべての関数のグラフ表示が表示されます。

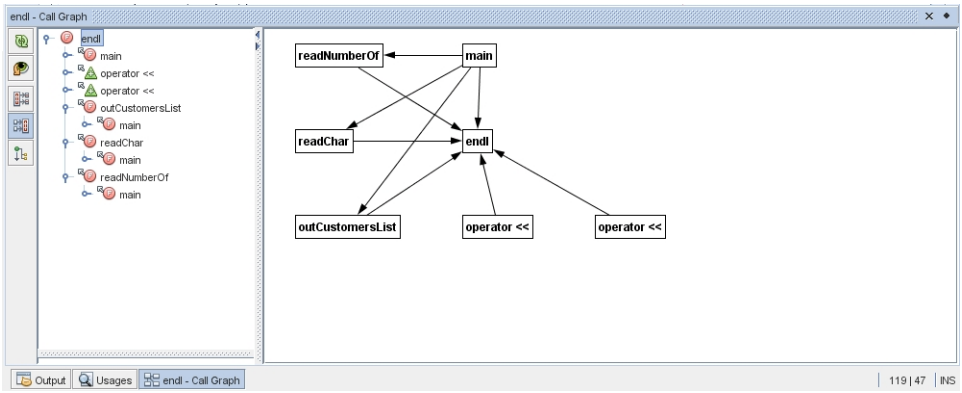


スクリーンショットに示す関数の一部が表示されない場合は、「コールグラフ」ウィンドウの左側の「関数からの呼出し対象」ボタンをクリックして、メイン関数から呼び出される関数を表示します。

3. `endl` ノードを展開して、この関数によって呼び出される関数を表示します。グラフが更新されて、`endl` によって呼び出される関数が追加されます。
4. `endl` ノードを選択してウィンドウの左側の「フォーカス」ボタンをクリックして、`endl` 関数にフォーカスし、「関数の呼出し元」ボタンをクリックして `endl` 関数を呼び出すすべての関数を表示します。



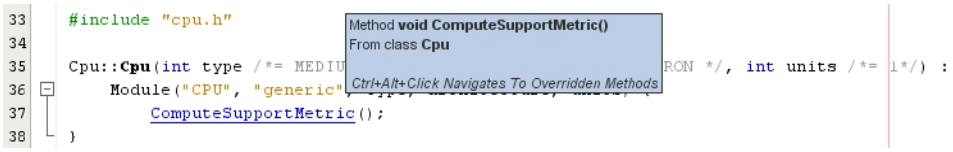
5. ツリー内のいくつかのノードを展開して、その他の関数を表示します。



ハイパーリンクの使用

ハイパーリンクナビゲーションによって、クラス、メソッド、変数、もしくは定数の呼び出しから宣言へジャンプしたり、宣言から定義にジャンプしたりすることができます。ハイパーリンクでは、オーバーライドされるメソッドからオーバーライドするメソッドへ、またはこの逆方向にジャンプすることもできます。

1. `Quote_1` プロジェクトの `cpu.cc` ファイルで、Ctrl を押しながら 37 行目にマウスオーバーします。`ComputeSupportMetric` 関数が強調表示され、関数についての情報を示す注釈が表示されます。



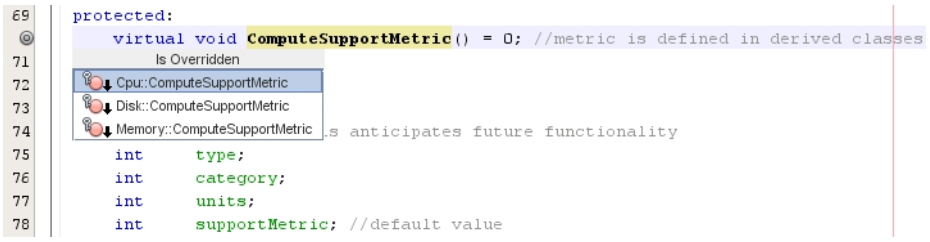
2. ハイパーリンクをクリックすると、エディタで関数の定義にジャンプします。

```
34
35  Cpu::Cpu(int type /*= MEDIUM */, int architecture /*= OPTERON */, int units /*= 1*/) :
36      Module("CPU", "generic", type, architecture, units) {
37          ComputeSupportMetric();
38  }
39
40  /*
41   * Heuristic for CPU module complexity is based on number of CPUs and
42   * target use ("category"). CPU architecture ("type") is not considered in
43   * heuristic
44   */
45
46  void Cpu::ComputeSupportMetric() {
47      int metric = 100 * GetUnits();
48
```

3. Ctrl を押しながら定義にマウスオーバーし、ハイパーリンクをクリックします。エディタで、cpu.h ヘッダーファイルの関数の定義にジャンプします。
4. エディタツールバーの左向き矢印をクリックすると、エディタは cpu.cc 内の定義に戻ります。
5. 左マージンにある緑色の円  の上にマウスカーソルを置くと、このメソッドが別のメソッドをオーバーライドすることを示す注釈が表示されます。

```
37          ComputeSupportMetric();
38  }
39
40  /*
41   * Heuristic for CPU module complexity is based on number of CPUs and
42   * target use ("category"). CPU architecture ("type") is not considered in
43   * heuristic
44   */
45  Overrides Module::ComputeSupportMetric
46  void Cpu::ComputeSupportMetric() {
47      int metric = 100 * GetUnits();
48
49      switch (GetTypeID()) {
50          case MEDIUM:
```

6. 緑の円をクリックしてオーバーライドされたメソッドに移動すると、エディタは module.h ヘッダーファイルにジャンプします。マージンにグレーの円が表示され、メソッドがオーバーライドされていることを示します。
7. グレーの円をクリックすると、エディタにはこのメソッドがオーバーライドするメソッドのリストが表示されます。

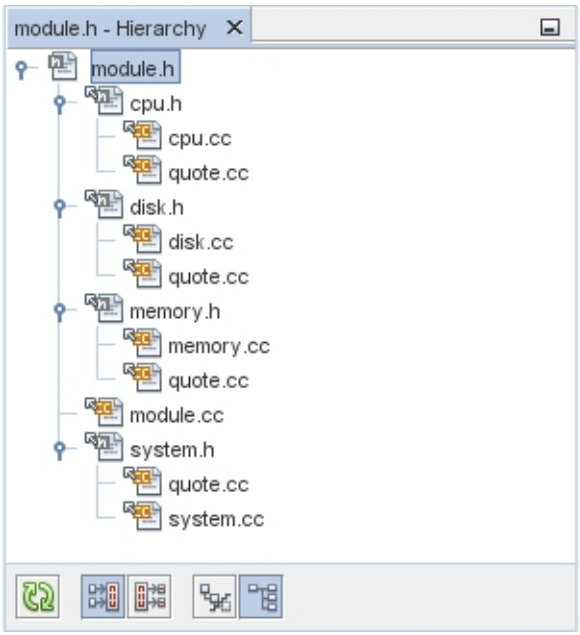


8. 「Cpu::ComputerSupportMetric」項目をクリックすると、エディタは cpu.h ヘッダーファイルのメソッドの宣言に戻ります。

インクルード階層の使用

「インクルードの階層」ウィンドウでは、直接的または間接的にソースファイルにインクルードされたすべてのヘッダーファイルとソースファイル、または直接的または間接的にヘッダーファイルにインクルードされたすべてのソースファイルおよびヘッダーファイルを検査できます。

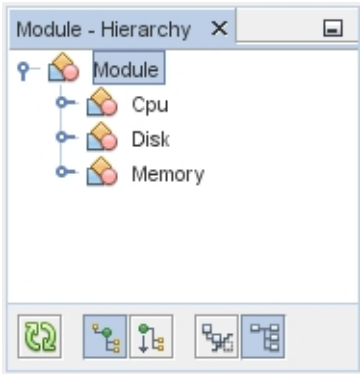
1. Quote_1 プロジェクトの module.cc ファイルをソースエディタで開きます。
2. ファイルの #include "module.h" 行を右クリックして、「ナビゲート」>「インクルードの階層を表示」を選択します。
3. デフォルトで、「階層」ウィンドウにはヘッダーファイルに直接インクルードされるファイルのプレーンリストが表示されます。ウィンドウ下部にある「ツリー・ビューを表示」ボタンをクリックします。「直接的インクルードのみを表示」ボタンをクリックすると、インクルードするファイルまたはインクルードされるファイルがすべて表示されます。ツリービューのノードを展開して、ヘッダーファイルをインクルードするすべてのソースファイルを表示します。



タイプの階層の使用

「タイプの階層」ウィンドウでは、クラスのすべてのサブタイプまたはスーパータイプを検査できます。

1. `Quote_1` プロジェクトの `module.h` ファイルを開きます。
2. `Module` クラスの宣言を右クリックして、「ナビゲート」>「タイプの階層を表示」を選択します。
3. 「階層」ウィンドウに、`Module` クラスのすべてのサブタイプが表示されます。


Module - Hierarchy
Module
Cpu
Disk
Memory

♦ ♦ ♦ 第 7 章

7 アプリケーションのデバッグ

この章では、プロジェクトのブレークポイントを作成する方法と、ブレークポイントを使用してコードをデバッグする方法を説明します。この章は次のセクションで構成されています。

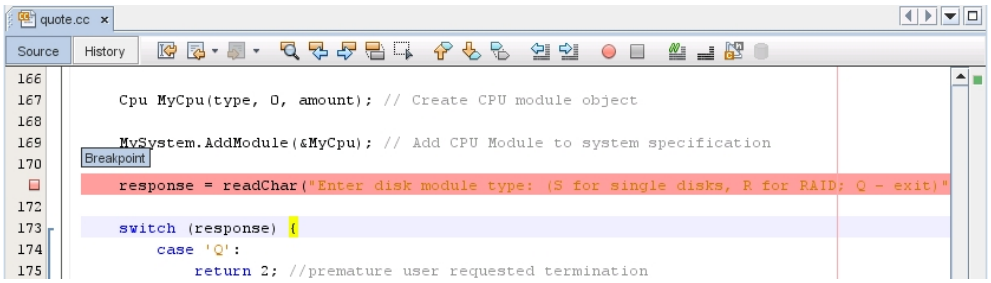
- 55 ページの「ブレークポイントの作成」
- 58 ページの「プロジェクトのデバッグ」
- 63 ページの「実行可能ファイルのデバッグ」
- 64 ページの「機械命令レベルでのデバッグ」
- 66 ページの「実行中のプログラムを接続してデバッグ」
- 68 ページの「既存のコアファイルのデバッグ」

ブレークポイントの作成

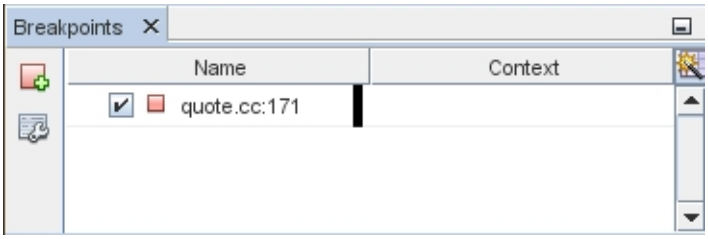
コード内でいつでもブレークポイントを作成し、操作できます。デバッガを実行するには、デバッガが実行を停止する位置を認識できるようにコード内でブレークポイントを設定する必要があり、これによって停止した位置で変数を確認し、コード行をステップ実行し、エラーを修正できます。

行ブレークポイントの作成と削除

1. `Quote_1` プロジェクトの `quote.cc` ファイルを開きます。
2. エディタウィンドウの 171 行目 (`response = readChar("Enter disk module type: (S for single disks, R for RAID; Q - exit)", 'S');`) の横の左マージンをクリックして行ブレークポイントを設定します。行が赤で強調表示され、このブレークポイントが設定されたことを示します。

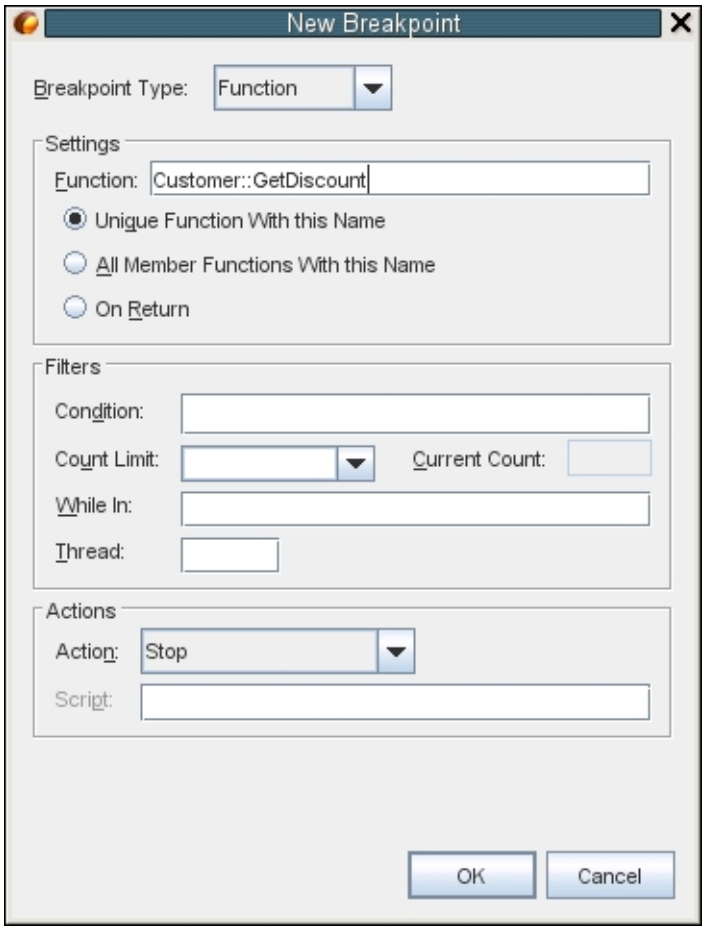


3. 左マージンのアイコンをクリックして、ブレークポイントを削除できます。
4. 「ウィンドウ」>「デバッグ」>「ブレークポイント」を選択して、「ブレークポイント」ウィンドウを開きます。行ブレークポイントがウィンドウに一覧表示されます。

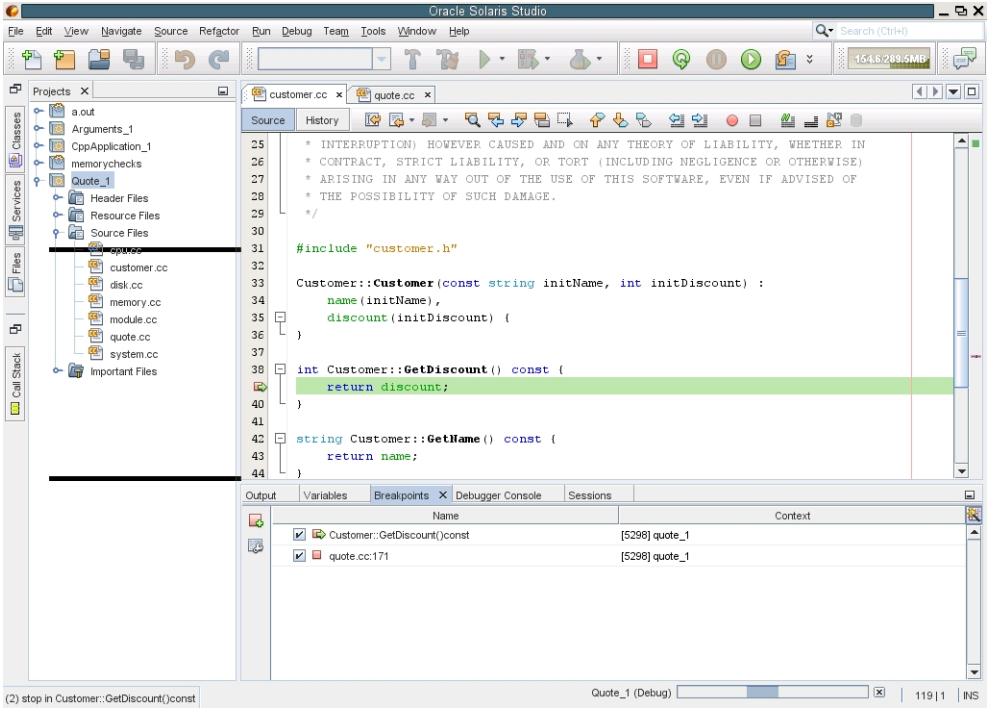


関数ブレークポイントの作成

1. 「デバッグ」>「新規ブレークポイント」(Ctrl+Shift+f8) を選択して、「新規ブレークポイント」ダイアログボックスを開きます。
2. 「ブレークポイントの種類」ドロップダウンリストで、タイプを「関数」に設定します。
3. 関数名 `Customer::GetDiscount` iを「関数」テキストフィールドに入力します。「OK」をクリックします。



4. 関数ブレークポイントが設定され、「ブレークポイント」ウィンドウのリストに追加されます。



ブレークポイントのグループ化

ファイル別、プロジェクト別、タイプ別、言語別といった、複数の異なるカテゴリを使用するとブレークポイントをグループ化できます。

1. 「ウィンドウ」>「デバッグ」>「ブレークポイント」ウィンドウを選択します。
2. 「ブレークポイント・グループを選択」アイコンをクリックします。
3. ブレークポイントグループを選択します。

   ブレークポイントは、選択に応じて配置されます。

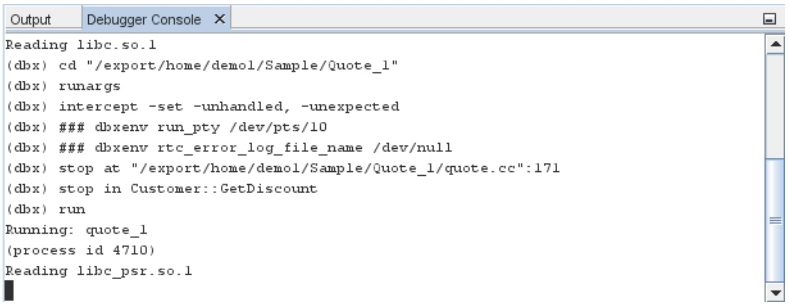
プロジェクトのデバッグ

デバッグセッションを開始すると、IDE はプロジェクトに関連付けられたツールの集合内にあるデバッガ (デフォルトでは `dbx` デバッガ) を開始し、デバッガ内でアプリケーションを実行しま

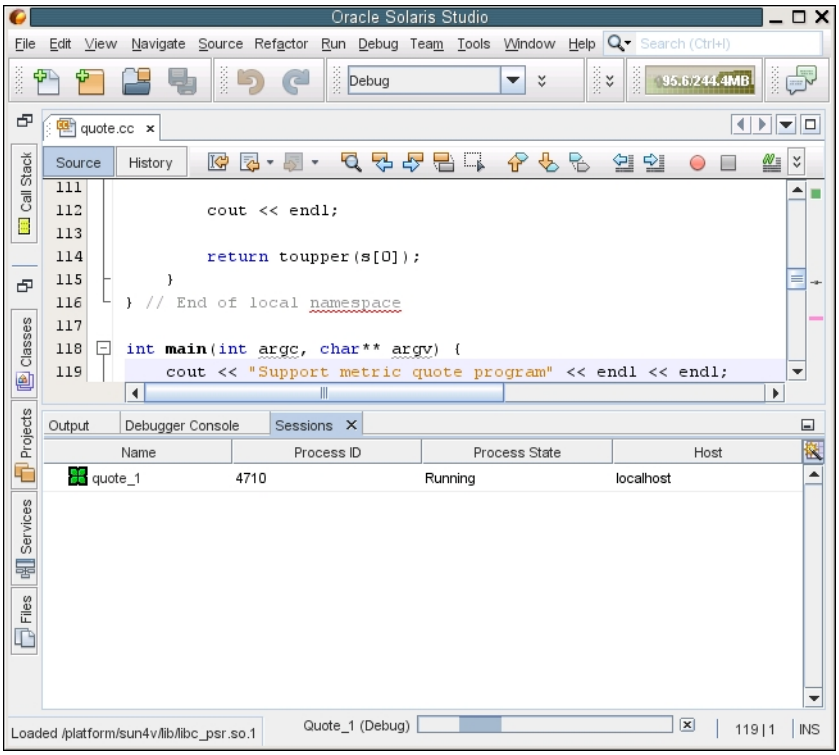
す。IDE はデバッガウィンドウを自動的に開き、デバッガ出力を「デバッガコンソール」ウィンドウに出力します。

デバッグセッションを開始する

1. プロジェクトノードを右クリックして「デバッグ」を選択して、Quote_1 プロジェクトのデバッグセッションを開始します。デバッガが起動してアプリケーションが実行され、「デバッガ・コンソール」ウィンドウが開きます。

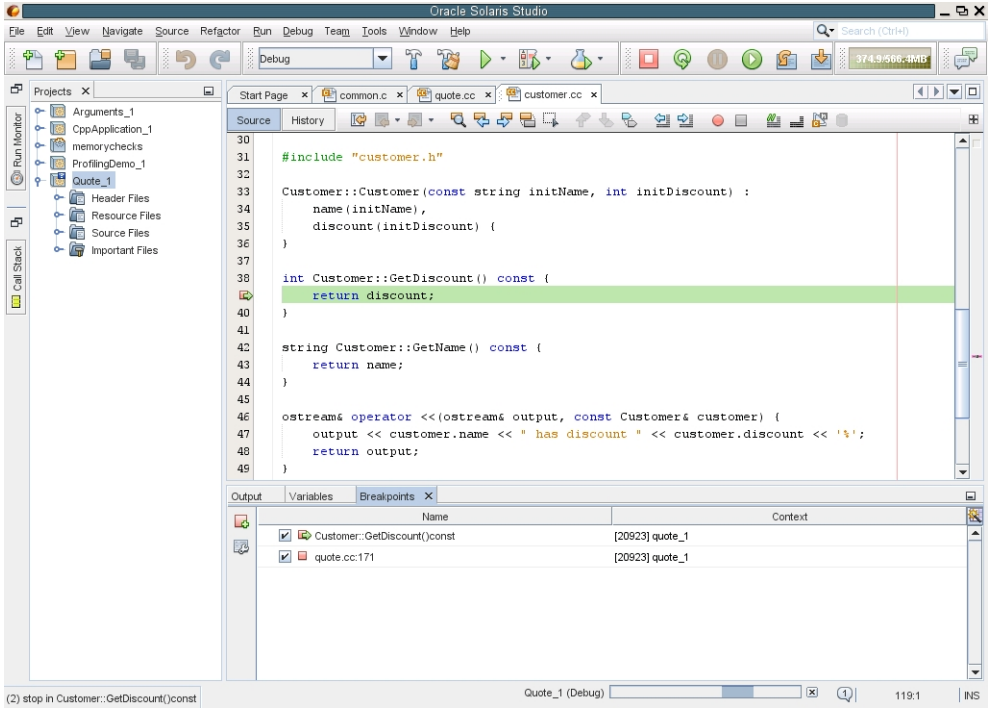


2. 「ウィンドウ」>「デバッグ」>「変数」(Alt+Shift+1) を選択して、「変数」ウィンドウを開きます。
3. 「ウィンドウ」>「デバッグ」>「セッション」(Alt-Shift-6) を選択して、「セッション」ウィンドウを開きます。このウィンドウにデバッグセッションが表示されます。

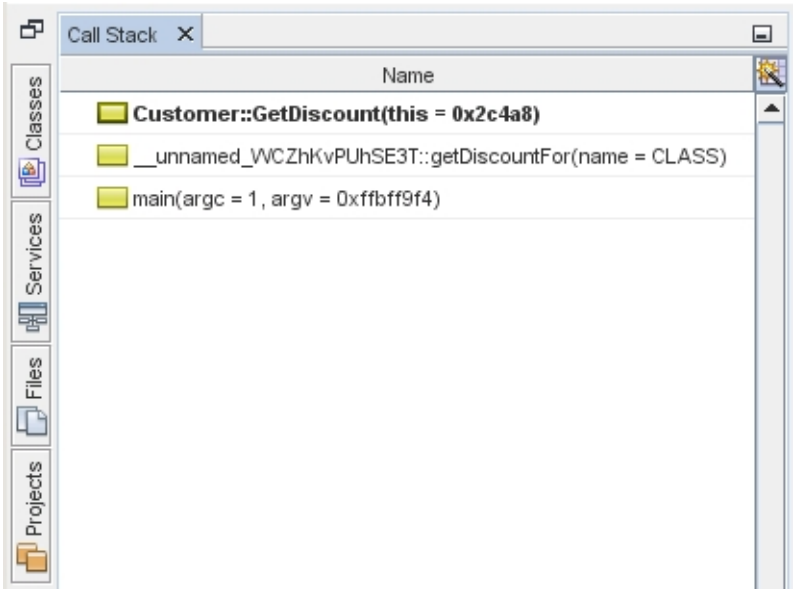


アプリケーションの状態の検査

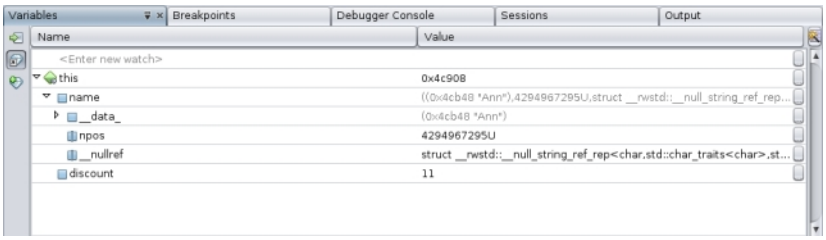
1. Quote_1 アプリケーションから、「出力」ウィンドウに入力するよう求められます。「顧客名を入力してください (Enter customer name:)」というメッセージの後に、顧客名を入力します。
2. customer.cc ファイルで、 GetDiscount 関数の最初の行にあるブレークポイントアイコンの上に、緑のプログラムカウンタ矢印が表示されます。



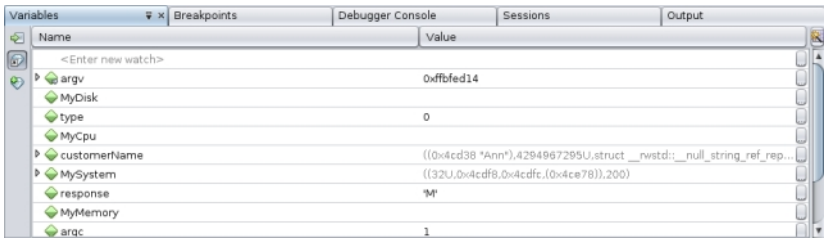
3. 「呼び出しスタック」ウィンドウを開きます (Alt-Shift-3)。呼び出しスタックには 3 つのフレームが表示されます。



4. 「変数」ウィンドウをクリックし、1 つの変数が表示されていることに注意します。ノードをクリックして構造を展開します。



5. 「続行」ボタンをクリックします。`GetDiscount` 関数が実行され、「出力」ウィンドウに顧客割引が出力されます。次に、入力を求められます。
6. プロンプトに従って入力します。プログラムが次のブレークポイント (以前設定した行ブレークポイント) で停止します。「変数」ウィンドウをクリックして、ローカル変数の長いリストを確認します。



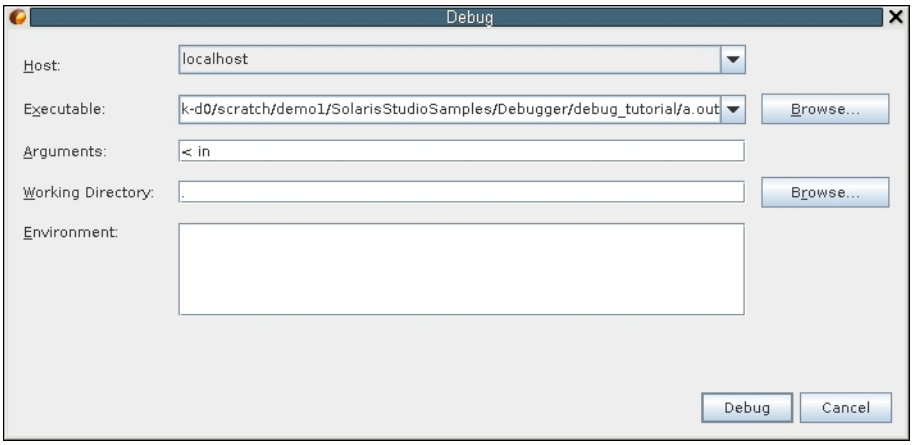
7. 「呼び出しスタック」ウィンドウを見て、スタックにフレームが 1 つしかないことを確認します。
8. 「続行」  をクリックして、プログラムが完了するまで、出力ウィンドウのプロンプトに従って入力を続行します。プログラムに最後の入力を行うと、デバッグセッションは終了します。プログラムが完了する前にデバッグセッションを終了するには、「セッション」ウィンドウでセッションを右クリックして「完了」を選択します。

実行可能ファイルのデバッグ

IDE プロジェクト内にない実行可能バイナリをデバッグできます。ただしデバッガがデバッグ情報を検出できるように、実行可能ファイルは、その構築に使用されたソースコードで検出できる必要があります。これはプロジェクトレスデバッグとも呼ばれます。

実行可能ファイルをデバッグするには:

1. メニューで、「デバッグ」>「実行可能ファイルをデバッグ」を選択します。



2. 「デバッグ」ダイアログボックスで、「ホスト」ドロップダウンリストから使用する開発ホストを選択します。
3. 「参照」ボタンをクリックして実行可能ファイルのパスを見つけるか、または「実行可能ファイル」テキストフィールドにパスを直接入力します。実行可能ファイルのパスを見つけるために「参照」をクリックした場合は、「作業ディレクトリ」テキストフィールドが自動的に生成されます。
4. 「引数」テキストフィールドに引数を追加します。
5. 作業ディレクトリが指定されていない場合は、「作業ディレクトリ」テキストフィールドに作業ディレクトリのパスを直接入力したり、「参照」をクリックして「作業ディレクトリ」ダイアログボックスでディレクトリを選択し、「選択」をクリックします。
6. 「環境」ペインに設定を入力して、環境変数を指定します。
7. 「デバッグ」をクリックします。

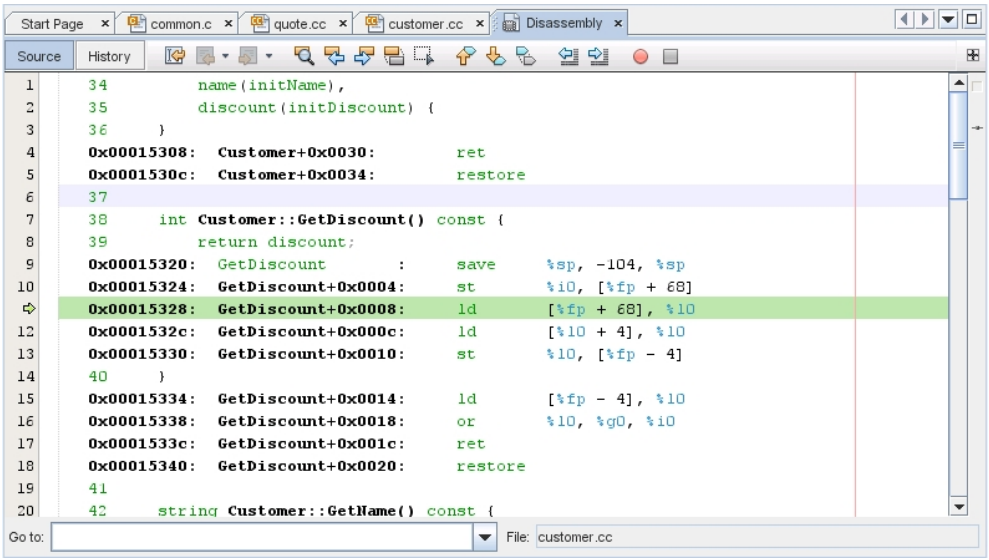
選択したプログラムが IDE にロードされますが、このプログラムは一時停止状態です。「続行」をクリックしてデバッグを続行できます。

実行可能ファイルのデバッグの詳細については、該当する IDE ヘルプページを参照してください。

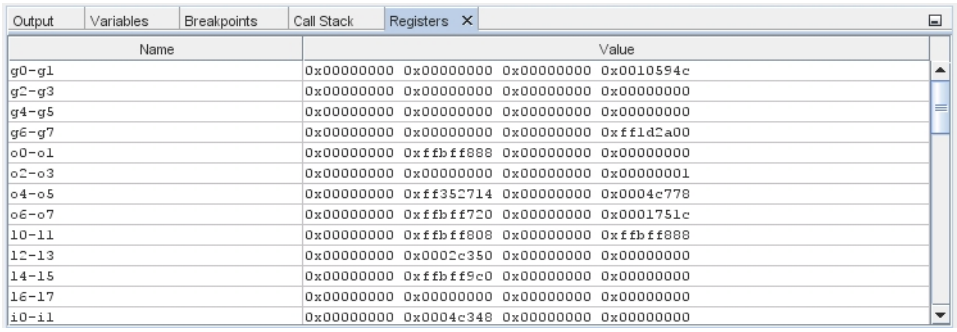
機械命令レベルでのデバッグ

デバッガには、プロジェクトを機械命令レベルでデバッグできるウィンドウがあります。

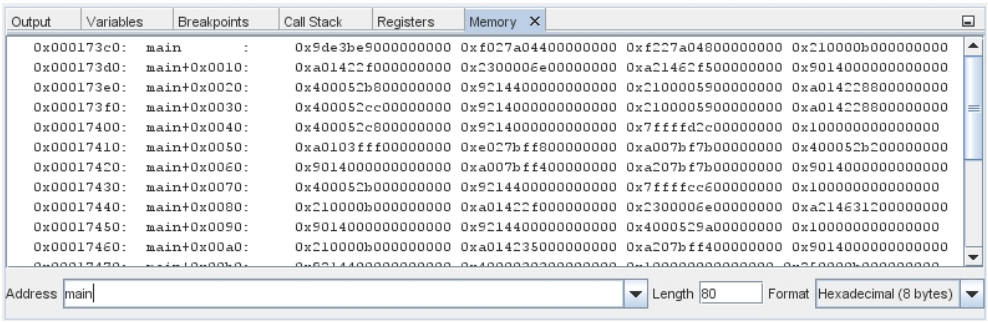
1. Quote_1 プロジェクトを右クリックして、「デバッグ」を選択します。
2. 「出力」ウィンドウで、プロンプトに従って顧客名を入力します。
3. プログラムが GetDiscount 関数のブレークポイントで一時停止したら、エディタウィンドウと同様に、「ウィンドウ」>「デバッグ」>「逆アセンブリ」を選択して「逆アセンブリ」ウィンドウを開きます。プログラムが一時停止した命令のブレークポイントアイコンの上に、緑のプログラムカウンタ矢印が表示されます。



4. 「ウィンドウ」>「デバッグ」>「レジスタ」を選択して「レジスタ」ウィンドウを開きます。ここにはレジスタの内容が表示されます。



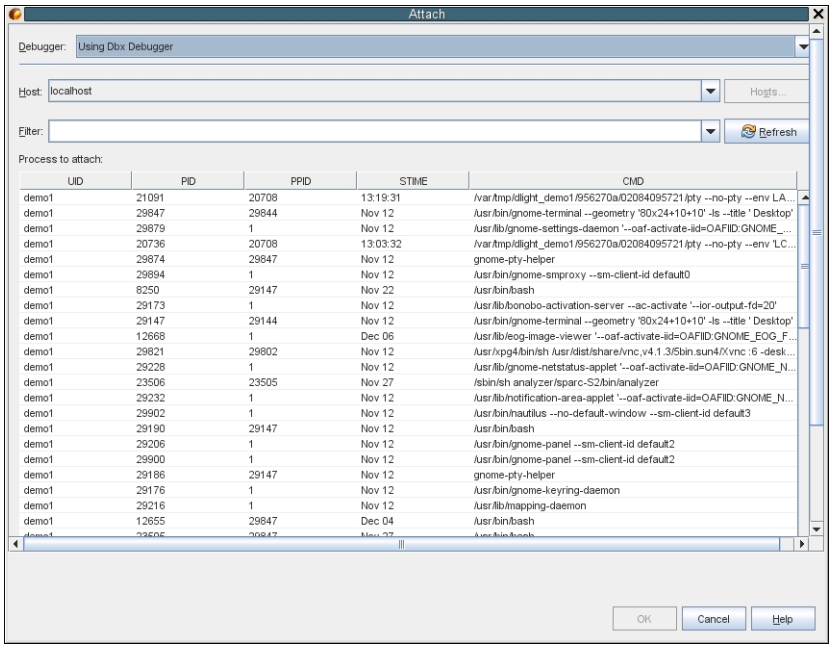
5. 「ウィンドウ」>「デバッグ」>「メモリー」を選択して「メモリー」ウィンドウを開きます。ここには、現在プロジェクトで使用されているメモリーの内容が表示されます。ウィンドウの下部で、参照するメモリーアドレスの指定、メモリー参照の長さの変更、メモリー情報の形式の変更を行えます。



実行中のプログラムを接続してデバッグ

すでに実行されているプログラムをデバッグするため、デバッガを該当するプロセスに接続できます。

1. 「ファイル」>「新規プロジェクト」を選択します。
2. 新規プロジェクトウィザードで、「サンプル」ノードを展開して、「C/C++」カテゴリを選択します。
3. Freeway Simulator プロジェクトを選択します。「次へ」をクリックして、「完了」をクリックします。
4. 作成した `Freeway_1` プロジェクトを右クリックして、「実行する」を選択します。プロジェクトが構築され、`Freeway` アプリケーションが開始されます。`Freeway` GUI ウィンドウで、「アクション」>「開始」を選択します。
5. IDE で、「デバッグ」>「デバッガを接続」を選択します。

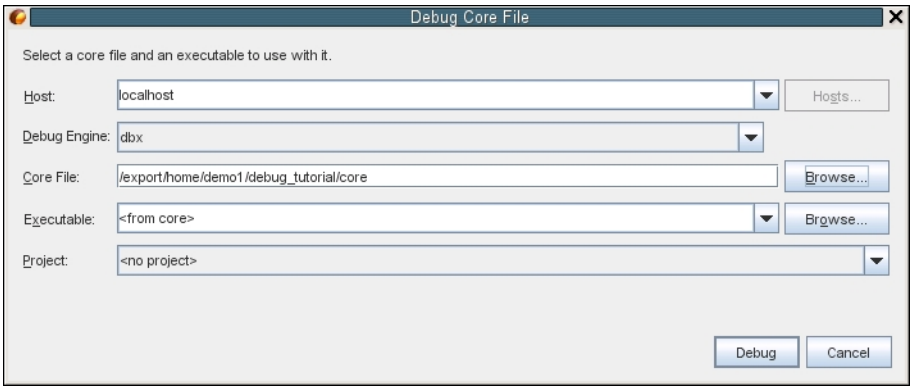


6. 「接続」ダイアログボックスで、「フィルタ」フィールドに Freeway と入力して、プロセスのリストをフィルタします。
7. フィルタしたリストから Freeway プロセスを選択します。
8. 「OK」をクリックします。
9. デバッグセッションが開始し、デバッガが接続されたポイントで Freeway プロセスの実行が一時停止します。
10. 「続行」をクリックして、現在デバッガの制御下で実行中の Freeway の実行を続行します。「一時停止」をクリックすると、Freeway の実行が一時停止し、変数や呼び出しスタックなどを検査できます。
11. 「続行」をもう一度クリックして、「デバッガ・セッションを終了」をクリックします。デバッガセッションが終了しますが、Freeway プロセスは実行を続行します。Freeway GUI で「ファイル」>「終了」を選択して、アプリケーションを終了します。

既存のコアファイルのデバッグ

プログラムがクラッシュする場合、コアファイル (クラッシュしたときのプログラムのメモリーイメージ) をデバッグできます。コアファイルをデバッガにロードするには、次の手順に従います。

1. 「デバッグ」>「コアファイルのデバッグ」を選択します。
2. 「コアファイル」フィールドにコアファイルのフルパスを入力するか、または「コアファイルを選択」ダイアログボックスで「参照」をクリックしてコアファイルにナビゲートします。



3. デバッガによって、指定したコアファイルと実行可能ファイルを関連付けることができない場合、デバッガからエラーメッセージが表示されます。この状況が発生する場合、「実行可能ファイル」テキストボックスに実行可能ファイルのパス名を入力するか、または「参照」ボタンをクリックして「実行可能ファイル」ダイアログボックスを使用して実行可能ファイルを選択します。
4. デフォルトで、「プロジェクト」テキストフィールドには、**<プロジェクトなし>** `(no project)`、または実行可能ファイルの名前と完全に一致する既存のプロジェクトの名前が表示されます。実行可能ファイルに新規プロジェクトを作成するには、**<新規プロジェクトの作成>** `(create new project)` を選択します。
5. 「デバッグ」をクリックします。

デバッグの詳細なチュートリアルは、『Oracle Solaris Studio 12.4: dbxtool チュートリアル 』を参照してください。

♦ ♦ ♦ 第 8 章

8 プロジェクトのモニタリング

IDE には、アプリケーションで実行時の問題を検出できるように、実行中の C/C++/Fortran プロジェクトの動作を調べるためのツールがあります。このような問題は、コードのデバッグ時に検出されないことがあります。プロファイリングツールには次のものがあります。

- CPU 使用率
- スレッド使用量
- メモリー使用
- メモリーアクセスエラー

この章は次のセクションで構成されており、ツールの内容と使用法を説明します。

- 69 ページの「実行モニターのプロファイリングツールを使用したプロジェクトのプロファイリング」
- 87 ページの「プロジェクトでのメモリーアクセス検査の実行」

実行モニターのプロファイリングツールを使用したプロジェクトのプロファイリング

実行モニターのプロファイリング情報は、Oracle Solaris Studio パフォーマンス分析ツールとその基盤となるオペレーティングシステムを使用して収集されます。実行モニターのツールは情報をグラフィカルに表示し、またクリックするとコードの問題領域に関する詳細情報を表示するボタンがあります。このセクションでは、プロファイリングデモプロジェクトの設定、プロジェクトのプロパティーでのプロファイルツールの有効化、および実行モニターのプロファイリングツールの実行結果の確認します。

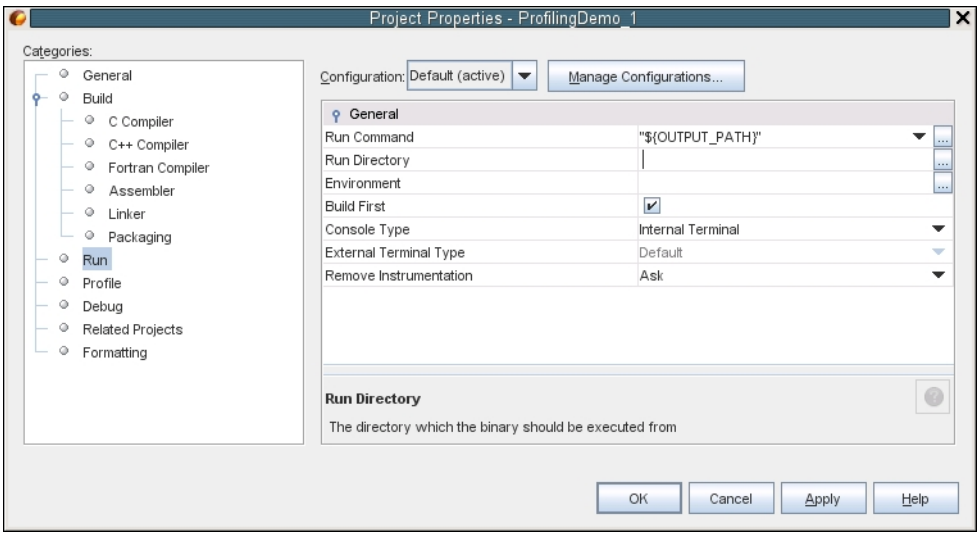
プロファイリングプロジェクトの設定

このチュートリアルでは、ProfilingDemo サンプルプロジェクトを使用します。このデモを作成するには、次の手順に従います。

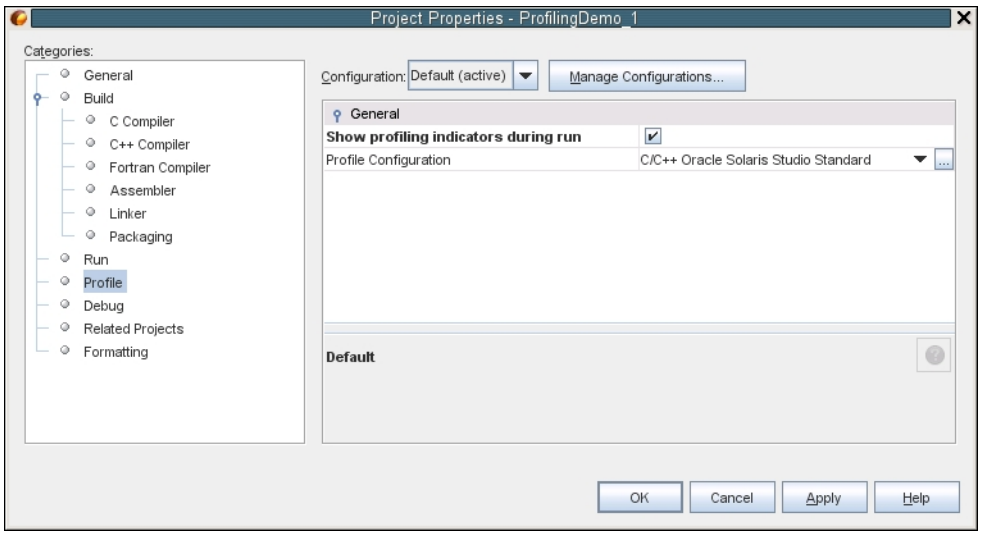
1. 「ファイル」>「新規プロジェクト」(Ctrl+Shift+N) を選択します。
2. 新規プロジェクトウィザードで、「サンプル」ノードを展開して、「C/C++」カテゴリを選択します。
3. 「プロファイリングのデモ」プロジェクトを選択します。「次へ」をクリックして、「完了」をクリックします。

プロジェクトを構成してプロファイリングを有効にするには、次の手順に従います。

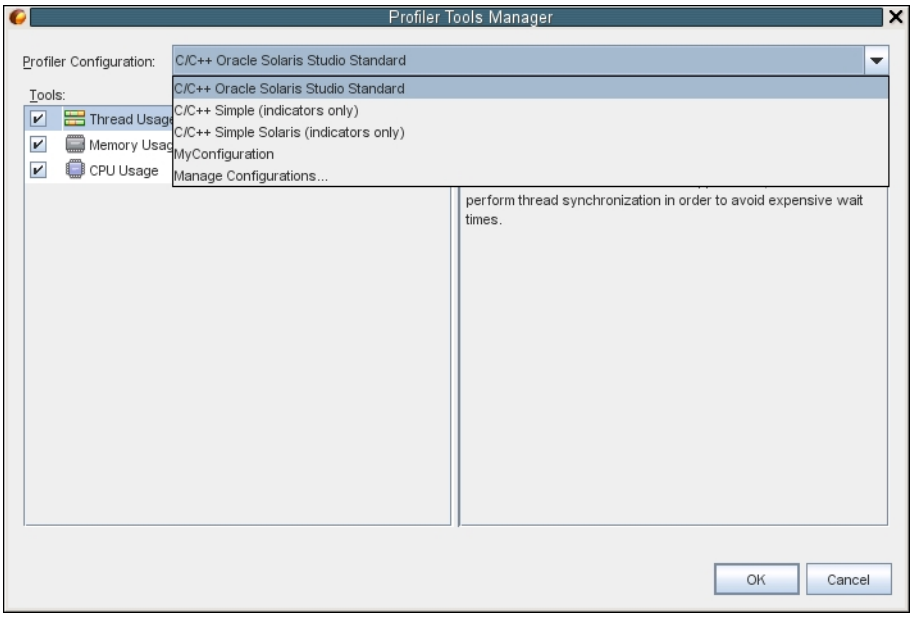
1. 「プロジェクト」タブの「ProfilingDemo_1」プロジェクトノードを右クリックし、「プロパティー」を選択します。
2. 「カテゴリ」パネルで「構築」ノードを選択し、「ツールコレクション」が Oracle Solaris Studio に設定されていることを確認してください。
3. 「カテゴリ」パネルで「実行」ノードを選択し、「コンソールタイプ」が「内部ターミナル」に設定されていることを確認してください。このチュートリアルで示すように、外部ターミナルのウィンドウではなく IDE の出力ウィンドウにプログラムの出力を表示できます。



4. 「カテゴリ」パネルで「プロファイル」ノードを選択します。「実行時にプロファイリングインジケータを表示」オプションがチェックされていることを確認します。プロジェクトでこのオプションをチェックすると、実行モニターツールタブが表示されます。



5. 「プロファイル構成」の「C/C++ Oracle Solaris Studio 標準」を選択します。「プロファイル構成」リストの横にある「...」ボタンをクリックします。
6. 「C/C++ Oracle Solaris Studio 標準」で選択されているツールは、「スレッド使用量」、「メモリー使用」、および「CPU 使用率」であることを確認します。



注記 - 「C/C++ 簡易 (インジケータのみ)」または「C/C++ 簡易 Solaris (インジケータのみ)」を選択すると、「実行モニター」ウィンドウのインジケータだけが表示されます。各インジケータウィンドウの詳細ボタンをクリックすると、次のメッセージが表示されます。「現在のプロファイル構成で詳細情報は使用できません。詳細を表示するには、プロジェクトのプロパティーで別のプロファイル構成を選択し、プロジェクトを再実行してください。」

---

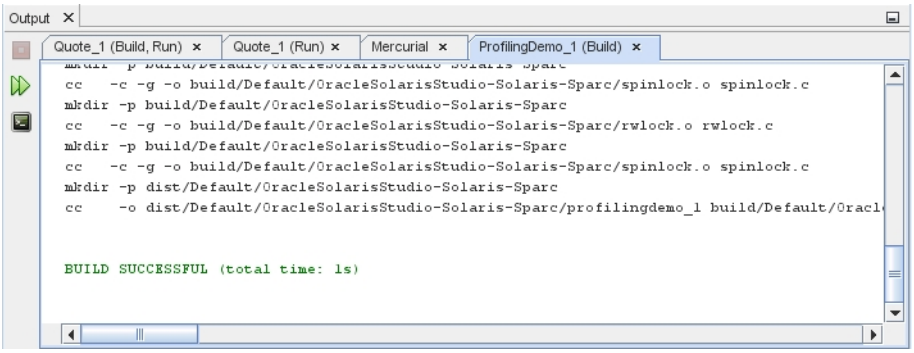
7. 「プロジェクトのプロパティー」ダイアログボックスで「OK」をクリックします。

ProfilingDemo_1 プロジェクトの構築と実行

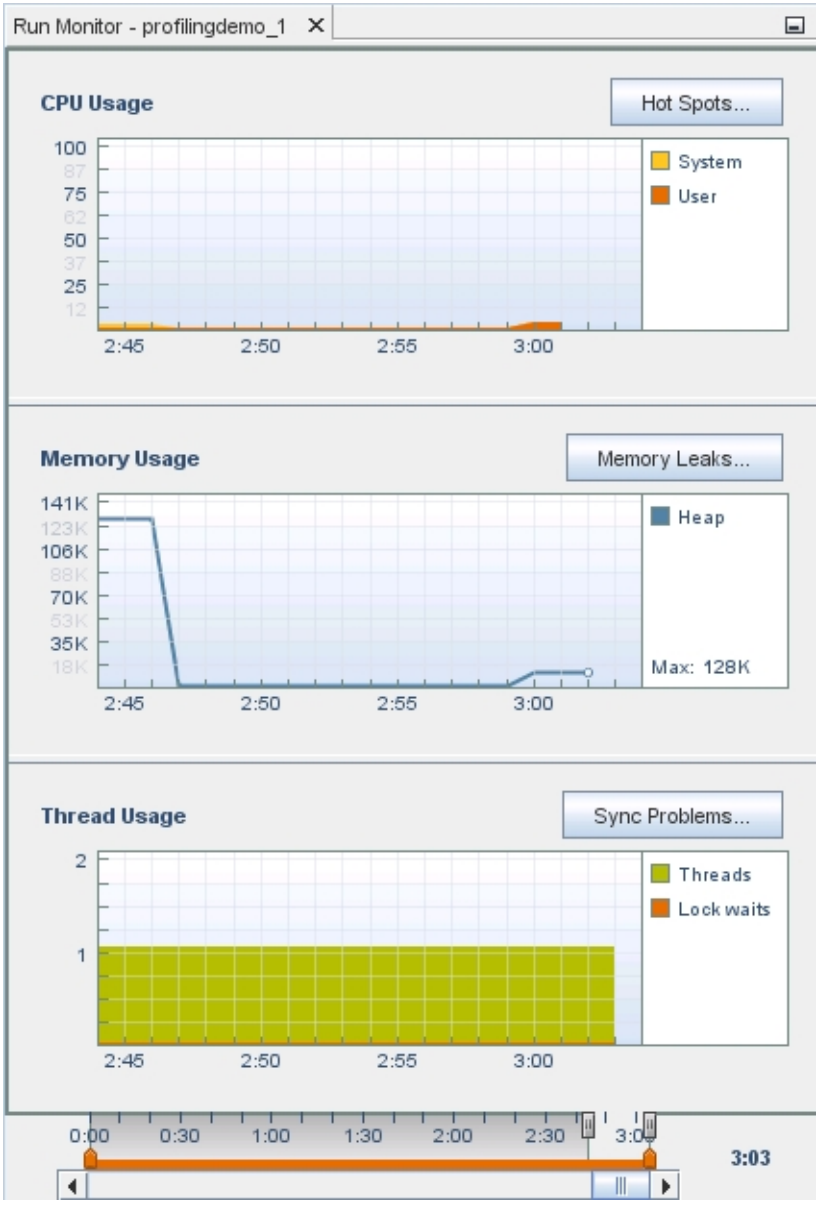
ProfilingDemo_1 プロジェクトを構築および実行するには、次の手順に従います。

1. ProfilingDemo_1 プロジェクトノードを右クリックし、「構築」を選択します。

   「出力」ウィンドウには、構築結果が表示されます。



2. ProfilingDemo_1 プロジェクトノードを右クリックし、「実行」を選択します。「実行モニター」ウィンドウが開き、「CPU 使用率」、「メモリー使用」、「スレッド使用量」の各インジケータと動的グラフが表示されます。



ProfilingDemo_1 プログラムの出力ウィンドウに実行中の処理が表示されるので、IDE によりツールにグラフィカルに表示されているデータと照合できます。たとえば、プログラムで

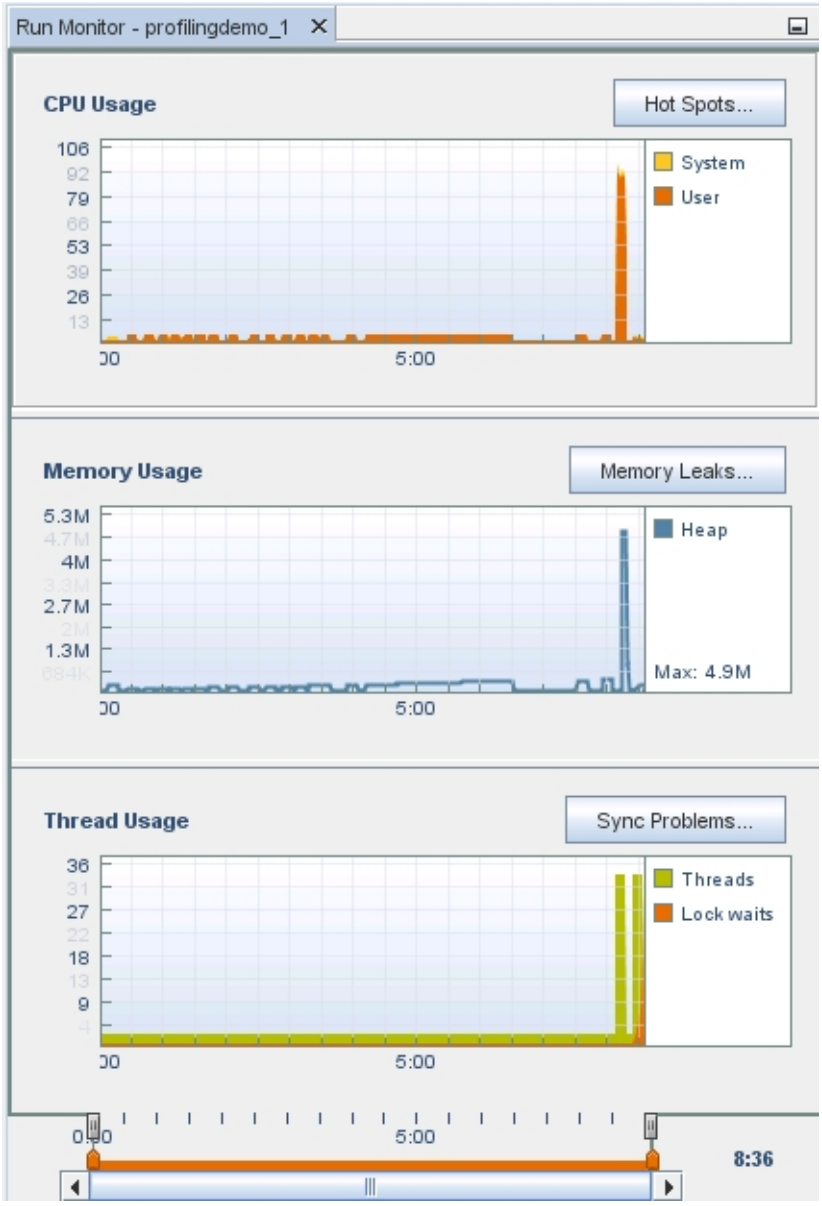
割り当て中のメモリー量が表示され、計算が実行され、メモリーが解放されます。こうした動作がグラフに反映されるところを確認できます。

3. プログラムが終了するまで、リクエストされるたびに Enter を押します。
4. マウスカーソルをインジケータの上に置くと、各グラフについて説明するツールチップが表示されます。各インジケータには、詳細情報を表示するためのボタンがあります (ボタンについては後のセクションで説明します)。

インジケータコントロールの使用

「実行モニター」ウィンドウの下部には、グラフの表示を制御するための表示スライダ、詳細スライダおよびタイムスライダの各スライダがあります。これらのコントロールはデータのフィルタリングにも使用されます。次の図はコントロールの名前を示します。

1. スライダに関する情報を表示するには、マウスカーソルをスライダの終了ポイントの上に置きます。
2. タイムスライダ (下部の水平スクロール バー) でマウスボタンをクリックしたままにします。すべてのグラフが連動してスライドするため、任意の時点での CPU、メモリー、スレッドの状態を確認し、それらの関係を把握できます。



タイムスライダを左から右にドラッグすると、実行の全体を確認できます。

3. マウスを表示スライダ (時間単位で重なったコントロール) に合わせます。表示スライダは、実行時間のどの部分をグラフに表示するかを制御します。

4. 表示スライダの開始ポイントのハンドルをクリックし、実行の開始位置までドラッグします。インジケータに実行全体が表示されます。この効果はズームアウトに似ています。すでにすべてのデータを表示しているため、時実行時間全体を選択すると、時間スクロールバーが最大化されることに注意してください。
5. 表示スライダの開始ポイントを、PTHREAD_MUTEX_DEMO の開始位置にドラッグします。ハンドルをドラッグすると、インジケータはこの領域にフォーカスするようにズームインします。スクロールバーを使用してふたたび実行時間を前後にスクロールできるようになっていることに注意してください。
6. オレンジ色の詳細スライダの終了ポイントの上にマウスカーソルを合わせると、スライダの使用方法の説明が表示されます。詳細スライダコントロールを使用すると、実行時の特定の部分を選択し、詳細情報を確認できます。

詳細スライダの開始ポイントハンドルを、表示スライダの開始ポイントよりも後の時点にドラッグします。表示スライダの開始ポイント以降の領域がインジケータによってグレー表示になることを確認します。これにより、グラフの開始ポイントから終了ポイントまでの間の結

果が強調表示されます。インジケータの詳細ボタンをクリックすると、詳細タブに、強調表示されている領域のデータが表示されます。

7. 詳細スライダの開始ポイントをドラッグして開始位置まで戻し、すべてのデータを表示します。

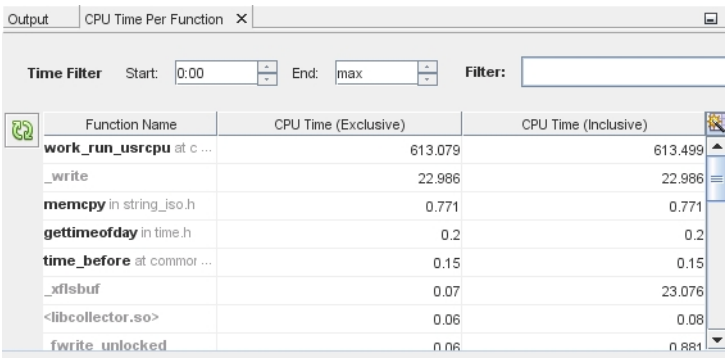
注記 - 表示スライダの開始ハンドルまたは終了ポイントハンドルのいずれかをスライドし、実行時間の任意の時点での詳細情報を表示して確認できます。

CPU 使用状況の調査

「CPU 使用 (CPU Usage)」グラフには、アプリケーションの実行中に使用される合計 CPU 時間がパーセントで表示されます。

1. 「CPU 使用率」の「ホットスポット...」ボタンをクリックし、CPU時間の詳細を表示します。

   「関数あたりの CPU 時間」ウィンドウが開き、プログラムの関数が、各関数によって使用された CPU の時間とともに表示されます。関数は使用された CPU 時間の順に一覧表示され、使用時間がもっとも長い関数が最初に表示されます。プログラムがまだ実行中の場合、初期状態で表示される時間は、グラフをクリックした時点で消費された時間です。



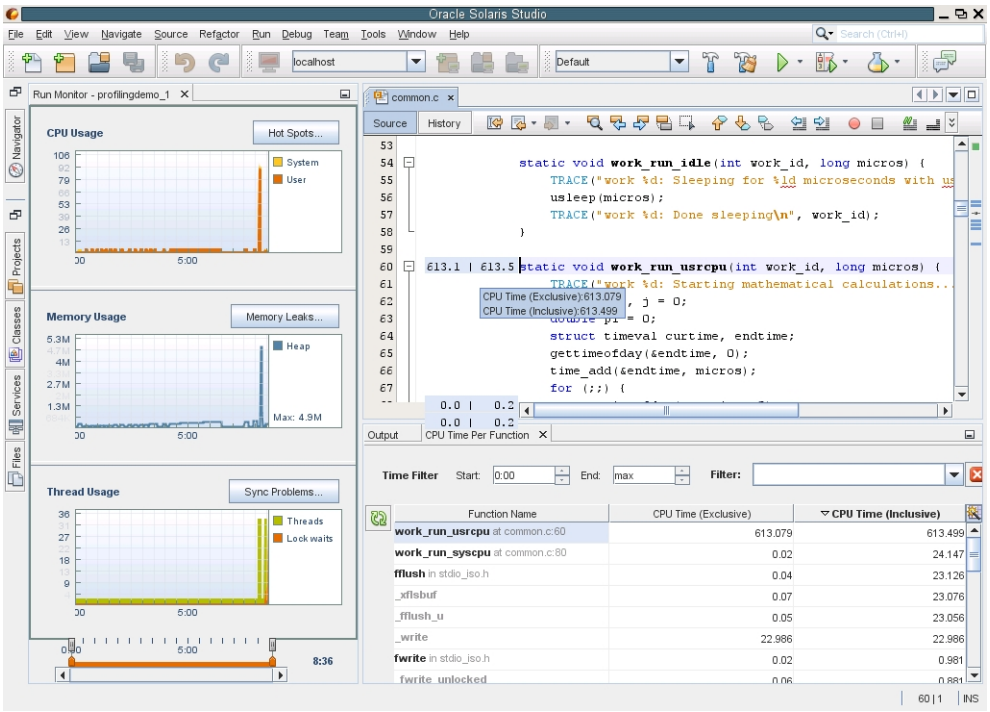
2. 「関数名」列見出しをクリックし、関数をアルファベット順にソートします。

   CPU 時間の 2 つの列の違いを確認します。「CPU 時間 (包括的)」には、関数の実行開始から終了までの間に使用された CPU 時間の合計 (一覧表示された関数によって呼び出されたその他すべての関数の時間も含む) が表示されます。「CPU 時間 (排他的) (CPU Time (Exclusive))」には、特定の関数のみに使用された時間が表示され、その関数から呼び出された関数は含まれません。

3. 「CPU 時間 (包括的)」列見出しをクリックして、使用時間がもっとも長い関数を先頭に戻します。

   この例では `work_run_usrcpu()` 関数の CPU 時間 (排他的) が 613.079、CPU 時間 (包括的) が 613.499 です。つまり、この関数が呼び出すほかの関数が使用している CPU 時間は短いが、`work_run_usrcpu()` 関数自体がもっとも長い時間を消費しています。

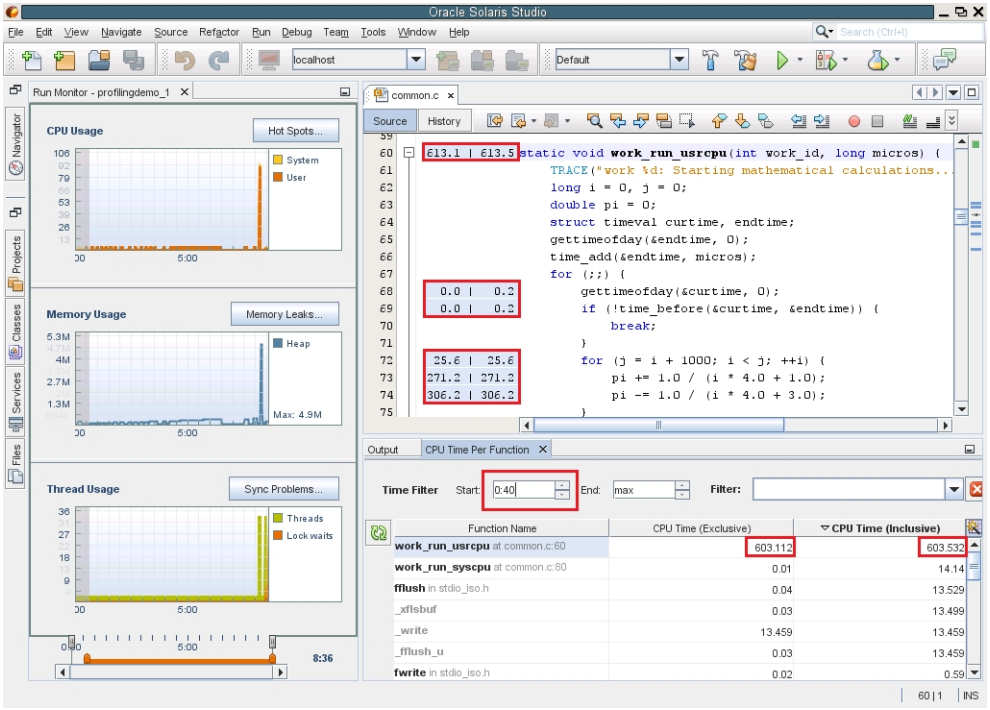
4. 一部の関数がボールドで一覧表示されています。これらの関数を呼び出すソースファイルに移動できます。`work_run_usrcpu()` 関数をダブルクリックします。



   `common.c`ファイルが開き、`work_run_usrcpu()` 関数がある 60 行目にカーソルが置かれます。この行の左マージンに、いくつかの数値が表示されます。

5. 左マージンにある数値の上にカーソルを合わせます。これらの数値は、「関数あたりの CPU 時間」ウィンドウに表示される関数の包括的および排他的 CPU 時間と同じメトリックです。メトリックは小数第 1 位に丸められますが、マウスをメトリックに合わせると、丸める前の値が表示されます。`work_run_usrcpu()` 関数内で計算を行う `for()` ループなど、CPU を消費する行のメトリックは、`common.c` ソースファイルにも表示されます。

6. 「関数あたりの CPU 時間」ウィンドウで、時間を入力して Enter を押すか、矢印を使用して秒をスクロールして、「時間フィルタ」の開始時間を 0:40 に変更します。グラフィックインジケータが、データフィルタリングコントロール上のハンドルを移動したときの状態に変更されます。ハンドルをドラッグすると、「関数あたりの CPU 時間」ウィンドウの「時間フィルタ」の設定が一致するように更新されます。重要なのは、表内の関数に対して表示されているデータがフィルタを反映して更新されるため、その時間内に使用された CPU 時間のみが表示されることです。

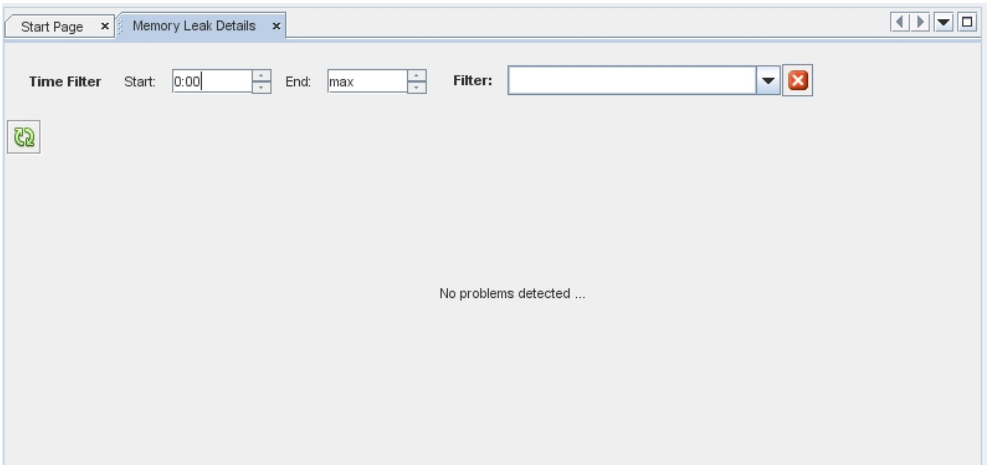


7. 特定のメトリックに一致するデータにフィルタを適用することもできます。`work_run_usrcpu()` の「CPU 時間 (排他的) (CPU Time (Exclusive))」メトリックを右クリックします。この例では、「関数あたりの CPU 時間」ウィンドウで 603.112 を右クリックします。「次の条件の行のみ表示: > CPU 時間 (排他的) == 603.112」を選択します。すべての行がフィルタで除外され、排他的 CPU 時間が 603.112 と等しい行だけが表示されます。

メモリー使用状況の調査

「メモリー使用」インジケータには、プロジェクト実行時のメモリーヒープの経時変化が表示されます。これは、プログラム内で不要になったメモリーの開放に失敗した場所をポイントする、メモリーリークの特定に使用できます。メモリーリークは、プログラムのメモリー消費が増加する原因となります。メモリーリークが発生しているプログラムの実行時間が長くなると、結果的に使用できるメモリーが不足する場合があります。

1. 「実行モニター」の時間スライダを左右に動かし、時間中にメモリーヒープが増減する様子を確認します。ProfilingDemo_1 ではスパイクが 4 回発生しています。最初の 2 回は Sequential Demo 中に発生し、3 回目は Parallel Demo 中に発生し、最後は Pthread Mutex Demo 中に発生しています。
2. 「メモリーリーク」ボタンをクリックし、「メモリーリークの詳細」ウィンドウを表示します。メモリーリークを示している関数に関する詳細情報が表示されます。メモリーリークが発生している関数だけが、表の中に一覧表示されます。ボタンをクリックした時にプログラムが実行中の場合は、ボタンをクリックした時点で存在していたリークの場所が表示されます。時間が経過するとメモリーリークが増加する可能性があるため、「リフレッシュ」ボタンをクリックします。実行の終了時までメモリーリークが検出されなかった場合は、「メモリーリーク詳細 (Memory Leak Detail)」タブにメモリーリークが見つからなかったことが示されます。
3. 開始時間と終了時間を変更してデータをフィルタリングしたり、「CPU 使用状況の調査」セクションと同様に「実行モニター」ウィンドウでオレンジ色の詳細スライダを使用できます。この例ではメモリーリークは発生していません。

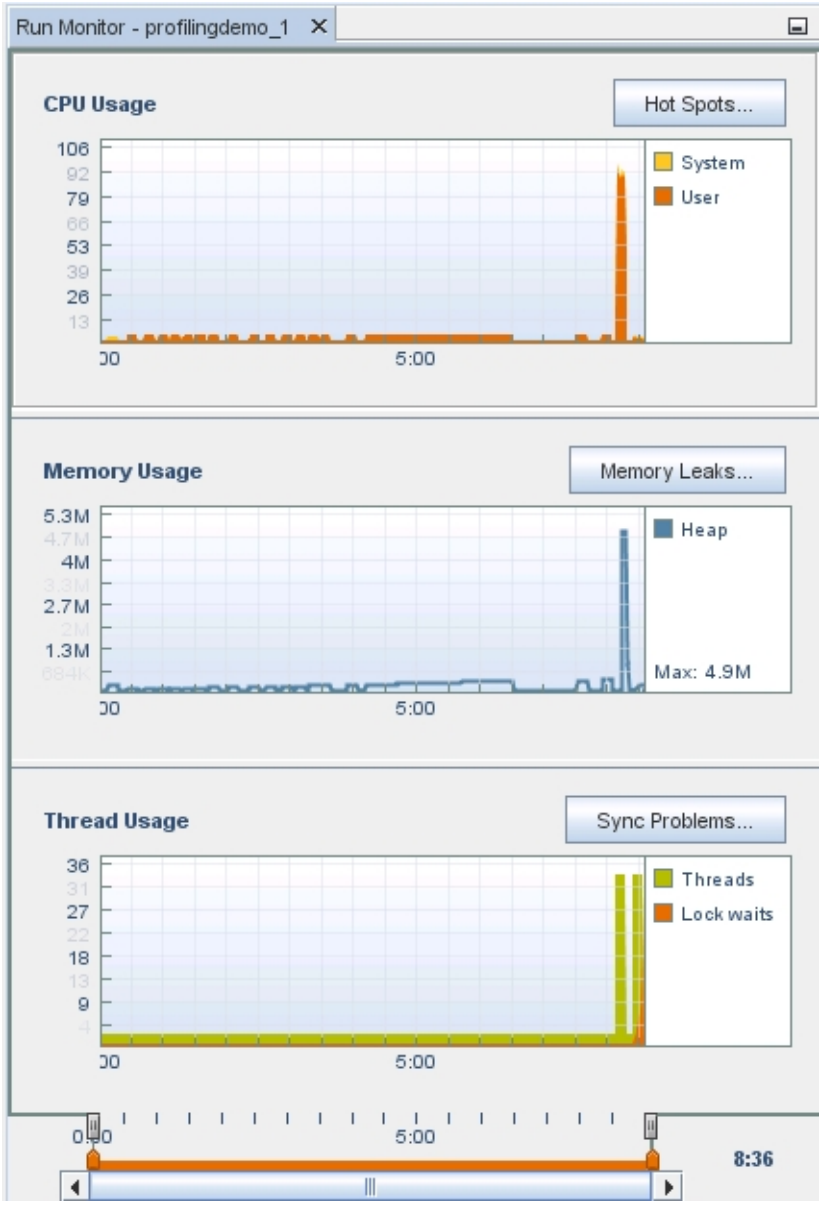


4. 関数をダブルクリックすると、対応するファイルが開き、関数でメモリーリークが発生している行が表示されます。メモリーリークメトリックがソースエディタの左側のマージンに表示されます。
5. CPU 使用率のメトリックの場合と同様に、マウスカーソルをメトリックに合わせて、詳細を表示します。
6. 表内のメトリックを右クリックし、表内のデータにフィルタを適用します。表内のデータへのフィルタの適用は、すべてのプロファイリングツールで可能です。

スレッド使用状況の調査

「スレッド使用量」インジケータには、プログラムで使用されているスレッドの数と、スレッドがタスクを続行するためにロックを取得するまで待機する必要がある時点が示されます。このデータはマルチスレッドのアプリケーションで、コストのかかる待ち時間を回避するためにスレッドの同期を行う必要がある場合に有用です。

1. 「スレッド使用量」インジケータに、プロジェクトの実行時に実行されたスレッドの数が示されます。時間スライダを開始時点まで戻し、Parallel Demo が開始するまでスレッドの数が 1 であることを確認します。次の図ではこれは 8:07 であり、この時点で 1 から 32 スレッドに急増しています。



2. 表示スライダの終了ポイントハンドルを、Parallel Demo の開始位置に移動します。

同期間の「CPU 使用率」インジケータおよび「メモリー使用」インジケータでは、CPU 時間とメモリーを使用する何らかのアクティビティーを実行している 1 つのスレッドが示されています。これは、メインスレッドがファイルへの書き込みを行い、いくつかの計算を連続して行

う、Sequential Demo の部分に相当します。CPU 使用率とメモリー使用は、どちらもユーザーが Enter を押すのをプログラムが待機している間に減少し、スレッド数は 1 のままになります。

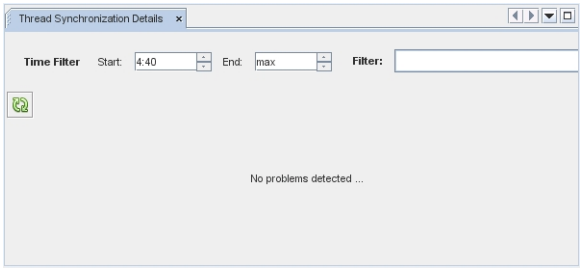
3. タイムスライダを右にスライドさせ、スレッドが増加する 2 か所のポイントを確認します。

   8:07 でのスレッドの増加は、プロジェクト実行の Parallel Demo 部分に対応しています。メインスレッドは、フィルへの書き込みと並列計算実行を処理するため、追加スレッドを開始します。メモリー使用量と CPU 使用率が増加しているが、2 つのタスクがかなり短い期間で完了しています。

4. Parallel Demo スレッドの終了後、スレッド数が 1 に戻り、メインスレッドはユーザーが Enter を押すのを待機します。スレッド数は、プログラムの Pthread Mutex Demo 部分の実行時にふたたび増加します。

   Pthread Mutex Demo 部分でロック待機がオレンジ色で表示されています。Pthread Mutex Demo では、複数のスレッドによる特定の関数へのアクセス競合を防ぐために相互排他ロックが使用されますが、これがスレッドがロックの取得を待機する原因です。

   「同期の問題」ボタンをクリックし、プロジェクトのスレッドロックに関する詳細を表示します。「スレッド同期の詳細」ウィンドウが開き、相互排他ロックを取得するために待機する必要がある関数が一覧表示されます。また、関数が待機に費やすミリ秒での秒数のメトリックと、関数がロックを待機する必要のある回数が表示されます。この例では、同期の問題は発生していません。



   プログラムの実行中に「同期の問題」ボタンをクリックした場合、リフレッシュボタン をクリックして表示を更新し、最新のスレッドロック情報にする必要がある場合があります。

5. 「待ち時間」列をクリックし、待機時間が長い順に関数をソートします。
6. 「ロック待機」列をクリックし、関数内でスレッドが待機した回数の順に関数をソートします。
7. 関数をダブルクリックすると、その関数がメモリー位置をロックしているソースファイルがエディタで開きます。「待ち時間」と「ロック待機」のメトリックがソースファイルの左の列に表示

されます。メトリックの上にマウスポインタを合わせると、「スレッド同期の詳細」ウィンドウの表示内容と一致する詳細が表示されます。

8. メトリックを右クリックして「プロファイラメトリックを表示 (Show Profiler Metrics)」を選択解除します。これでメトリックはどのプロファイルツールのソースエディタにも表示されなくなります。メトリックを再度表示するには、IDE メニューの「表示」を選択し、「プロファイラメトリックを表示」を選択します。

プロジェクトでのメモリーアクセス検査の実行

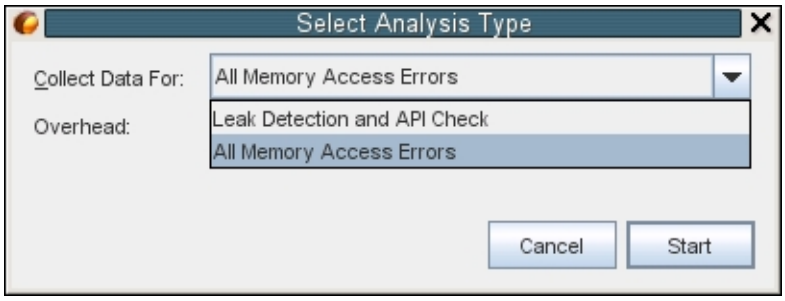
メモリー解析ツールを使用して、プロジェクト内のメモリーアクセスエラーを見つけることができます。このツールを使用すると、ソースコード内で各エラーが発生する場所を正確に特定されるので、これらのエラーを簡単に見つけることができます。

メモリー解析ツールはプログラムの実行中にメモリーアクセスエラーを動的に検出して報告するため、コードの一部が実行時に実行されていない場合、その部分のエラーは報告されません。

1. まだ行なっていない場合は、「Oracle Solaris Studio 12.3 Sample Applications」の Web ページ (http://www.oracle.com/technetwork/server-storage/solarisstudio/downloads/solaris-studio-samples-1408618.html) からサンプルアプリケーション zip ファイルをダウンロードし、任意の場所にファイルを解凍します。`memorychecks` アプリケーションは、`SolarisStudioSampleApplications` ディレクトリの `CodeAnalyzer` サブディレクトリにあります。
2. `memorychecks` アプリケーションを使用して、既存のソースからプロジェクトを作成します。
3. プロジェクトを右クリックし、「プロパティー」を選択します。「プロジェクトのプロパティー」ダイアログボックスで、「実行」ノードを選択し、「コマンドを実行」で、出力パスのあとに Customer.db と入力します。「OK」をクリックします。
4. プロジェクトを実行します。
5. メモリー解析のための計測付きのプロジェクトを構築します。

    a. 「プロジェクトをプロファイル」ボタン  の横にある下矢印をクリックして「プロジェクトをプロファイル」を選択し、ドロップダウンリストから「メモリーアクセスエラー」を見つけます。

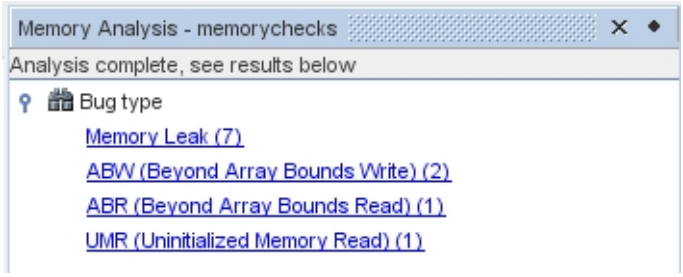
    b. 「解析の種類を選択」ダイアログボックスで、ドロップダウンリストから「すべてのメモリーアクセスエラー」を選択します。



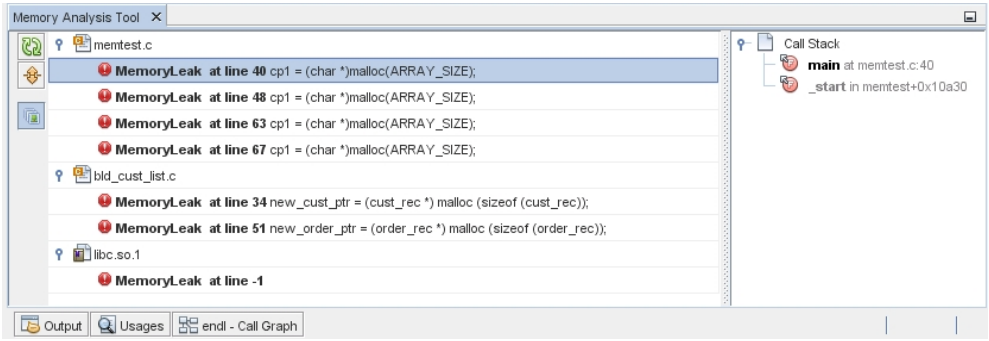
「オーバーヘッド」フィールドにはシステムにかかる負荷を示す「高」または「中」が表示されます。オーバーヘッドが高い場合、システム上で実行中のほかのプログラムのパフォーマンスに影響が出ることがありますが、データの競合とデッドロックの両方が検出されるのはこの場合です。

c. 「開始」をクリックします。

6. 「メモリープロファイルを実行」ダイアログボックスが開き、バイナリが計測されることが通知されます。「OK」をクリックします。
7. プロジェクトが構築され計測されます。アプリケーションが実行を開始し、「メモリー解析」ウィンドウが開きます。プロジェクトの実行が完了すると、プロジェクト内で見つかったメモリーアクセスエラーの種類の一覧が「メモリー解析」ウィンドウに表示されます。エラーの種類のあとの括弧内に、それぞれの種類のエラー数が表示されます。



8. エラーの種類をクリックすると、その種類のエラーが「メモリー解析ツール」ウィンドウに表示されます。



デフォルトでは、エラーはエラーが見つかったソースファイル別にグループ化されます。エラーをクリックすると、そのエラーの呼び出しスタックが表示されます。スタック内の関数呼び出しをダブルクリックすると、ソースファイル内の関連する行が表示されます。

♦ ♦ ♦ 第 9 章

9 リモート開発の実行

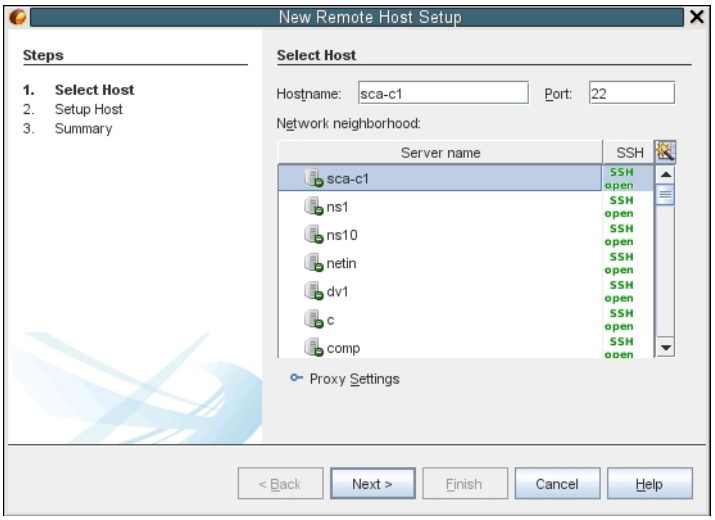
この章では、リモート開発の実行方法を説明します。また、リモート開発ツールバーの使用法についても説明します。また、この章ではリモートデスクトップ配布についても簡単に説明します。

リモート開発の実行

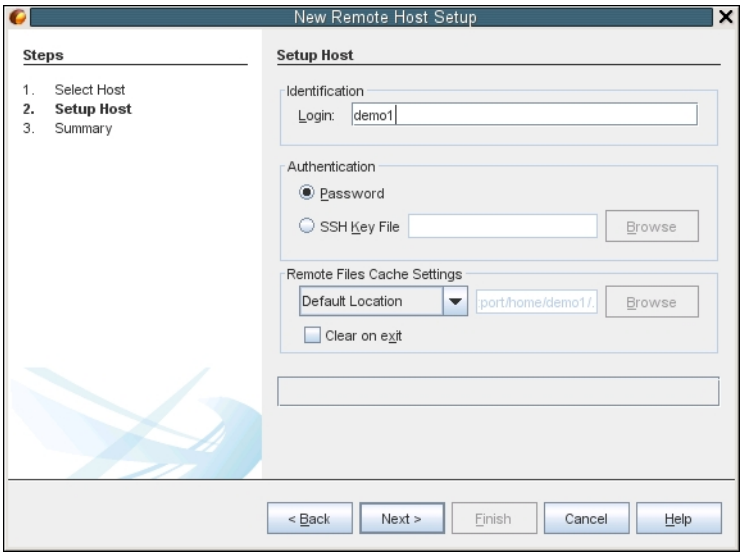
ローカルホスト (IDE を起動したシステム)、または UNIX® オペレーティングシステムを実行しているリモートホスト上で、プロジェクトを構築、実行、デバッグできます。リモート開発では、使い慣れたローカルのデスクトップ環境で IDE を実行しながら、リモートサーバーの処理能力と開発ツールを使用してプロジェクトを構築できます。

「オプション」ダイアログボックスの「構築ツール」タブで、リモート開発ホストを構成できます。リモートホストを追加するには、次の手順に従います。

1. 「ツール」>「オプション」を選択し、「C/C++」カテゴリをクリックします。
2. 「オプション」ダイアログボックスの「構築ツール」タブで、「編集」をクリックします。
3. 「構築ホストマネージャー」ダイアログボックスで、「追加」をクリックします。
4. 「新規リモート構築ホスト」ウィザードの「ホストを選択」ページで、「ホスト名」フィールドにホストのシステム名を入力するか、または「隣接ネットワーク」リスト内の使用できるホストをダブルクリックして選択します。「次へ」をクリックします。



5. 「ホストのセットアップ」ページで、「ログイン」フィールドにログイン名を入力して「次へ」をクリックします。



6. ウィザードからパスワードが要求され、ホストに接続して「サマリー」ページが表示されます。「完了」をクリックします。

7. ホストが「構築ホストマネージャー」ダイアログボックスの「構築ホスト」リストに追加されたら、「OK」をクリックします。
8. IDE がリモートホストを使用する方法を指定するプロパティーを、「サービス」ウィンドウで設定できます。「C/C++ 構築ホスト」ノードを展開し、リモートホストを右クリックして、「プロパティー」を選択します。「ホストのプロパティー」ダイアログボックスで目的のプロパティーを設定します。
9. リモートホストをデフォルトの構築ホストに設定するには、「サービス」ウィンドウの「C/C++ 構築ホスト」ノード内でホストを右クリックし、「デフォルトに設定」を選択します。

リモートホストにプロジェクトを開発するには、プロジェクトはローカルホストとリモートホストの両方で参照できる共有ファイルシステム上に存在する必要があります。通常このようなファイルシステムは、NFS または Samba を使用して共有されます。リモートホストを定義するときに、プロジェクトソースファイルへのローカルパスとリモートパスの間のマッピングを定義できます。

プロジェクトを作成するとき、デフォルトの構築ホストがプロジェクトの構築ホストとして選択されます。「プロジェクトのプロパティー」ダイアログボックスの「構築」パネルで、プロジェクトの構築ホストを変更できます。実行可能ファイルまたはコアファイルをデバッグするときに、構築ホストを指定することもできます。

リモートホスト上に存在するプロジェクトに対してローカルホストで作業するには、「ファイル」>「リモート C/C++ プロジェクトを開く」を選択します。

リモート開発ツールバーの使用

IDE には、リモートホスト上のプロジェクトとファイルへのアクセスを容易にするリモート開発ツールバーがあります。アイコンを使用して、すでに構成したリモートホストを選択し、ローカルの場合と同じようにリモートホスト上のプロジェクトおよびファイルを操作できます。これは、完全リモート開発と呼ばれます。

リモート開発モードの詳細は、IDE の C/C++ リモート開発モードに関するヘルプページを参照してください。

リモートツールバーでは次の操作を実行できます。

- 「接続ステータス」アイコンをクリックし、アイコンの隣のリストで選択されているサーバーに接続または接続解除します。
- リモートプロジェクトの作成アイコンをクリックして、現在接続しているホストに新規プロジェクトを作成します。

- 「リモート・プロジェクトを開く」アイコンをクリックして、現在接続しているホストでプロジェクトを開きます。
- 「リモートファイルを開く」アイコンをクリックして、現在接続しているホストでファイルを開きます。

構築ホスト用のターミナルウィンドウの使用

ローカルホストまたはリモートホストのターミナルウィンドウを IDE で開くことができます。ターミナルウィンドウは、IDE の「出力」領域でタブとして開きます。IDE に対してリモートホストを事前に設定する必要はありませんが、SSH デーモンを実行しておく必要があります。

IDE でターミナルウィンドウを開く方法については、IDE ヘルプで構築ホスト用のターミナルウィンドウのオープンに関する項目を参照してください。ローカルホストのターミナルウィンドウを開くには、「ウィンドウ」>「IDEツール」>「ターミナル」を選択します。

コマンド行ターミナルウィンドウの表示オプションをパーソナライズすることもできます。詳細については、IDE ヘルプのオプションウィンドウ: その他: ターミナルの項目を参照してください。

リモートデスクトップ配布

リモートデスクトップ配布を使用すれば、ほとんどすべてのオペレーティングシステム上で動作し、リモートサーバー上の Oracle Solaris Studio のコンパイラやツールを使用するコードアナライザと IDE の配布を含む ZIP ファイルを生成できます。デスクトップシステムで IDE を実行すると、IDE は、配布の生成元となったサーバーをリモートホストとして認識し、Oracle Solaris Studio インストール内のツールコレクションにアクセスします。

リモートデスクトップ配布の詳細については、Oracle Solaris Studio でのリモートデスクトップからのコードの開発方法に関するページ (http://www.oracle.com/technetwork/articles/servers-storage-dev/howto-use-ide-desktop-1639741.html)を参照してください。

♦ ♦ ♦ 第 10 章

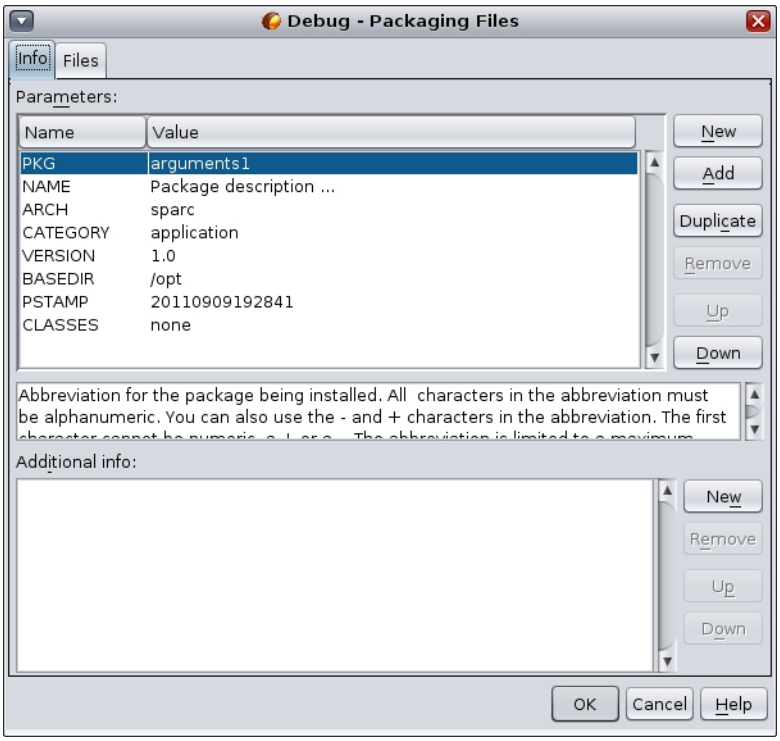
10 アプリケーションのパッケージング

この章では、アプリケーションの作成、コンパイル、デバッグが完了したあとでアプリケーションをパッケージングする方法を説明します。

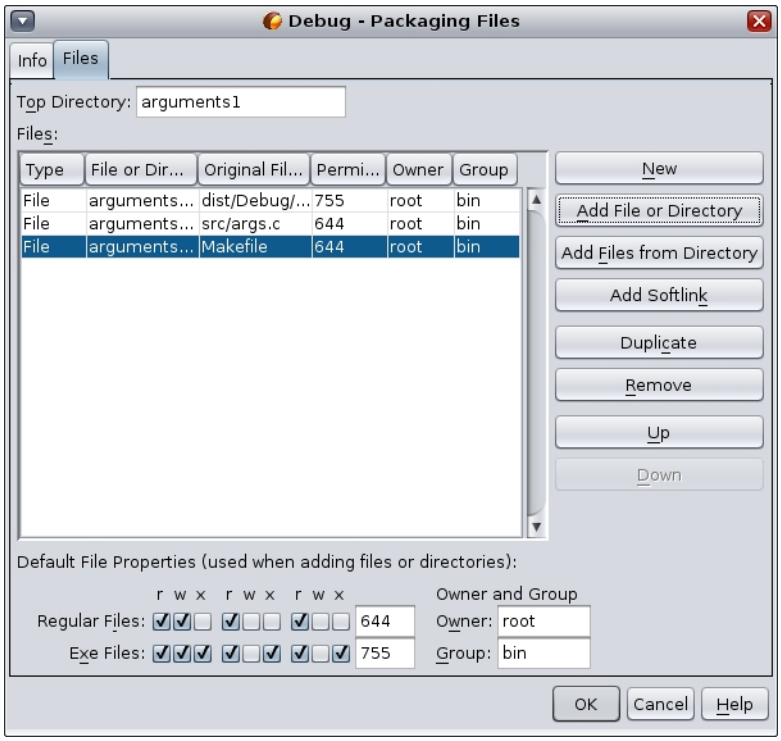
アプリケーションのパッケージング

完成したアプリケーションを tar ファイル、zip ファイル、Solaris SVR4 パッケージ、RPM、もしくは Debian パッケージとしてパッケージできます。

1. `Arguments_1` プロジェクトを右クリックして、「プロパティー」を選択します。
2. 「プロジェクトのプロパティー」ダイアログボックスで、「パッケージング」ノードを選択します。
3. ドロップダウンリストから、Solaris SVR4 パッケージタイプを選択します。
4. パッケージ先に別のディレクトリまたはファイル名を使用する場合は、出力パスを変更します。
5. 「ファイルのパッケージング」参照ボタンをクリックします。「ファイルのパッケージング」ダイアログボックス (SVR4 パッケージの場合) で、必要に応じて「情報」タブのパッケージパラメータを変更します。



6. すべてのパッケージタイプに対して、「ファイル」タブのボタンを使用してファイルをパッケージに追加します。各ファイルについて、「ファイル」リストの「パッケージ内のファイルまたはディレクトリパス」列に、パッケージ内でのパスを指定できます。「ファイル」リストが完成したら、「OK」をクリックします。



7. 必要に応じて、チェックボックスをクリックして冗長モードをオフにします。
8. 「OK」をクリックします。
9. パッケージを構築するには、プロジェクトを右クリックして「その他の構築コマンド」>「パッケージの構築」を選択します。

♦ ♦ ♦ 第 11 章

11 IDE に関する詳細情報

この章では、Oracle Solaris Studio IDE に関するその他のリソースを紹介します。

IDE の詳細

Oracle Solaris Studio IDE の使用方法については、F1 キーを押すかまたは IDE の「ヘルプ」メニューからアクセスできる IDE の統合ヘルプを参照してください。また多くのダイアログボックスに、そのダイアログボックスの使用方法に関する情報を表示するための「ヘルプ」ボタンがあります。

NetBeans IDE C/C++ Learning Trail のチュートリアルも、Oracle Solaris Studio IDE の使用方法を知るために役立ちますが、ユーザーインタフェースおよび機能にいくつか違いがあります。特に、デバッグについての NetBeans のドキュメントは Oracle Solaris Studio IDE には当てはまりません。

Oracle Database プロジェクト

IDE のデータベースエクスプローラの詳細については、Oracle Solaris Studio IDE でのデータベースエクスプローラの使用法に関するページを参照してください。

IDE の Oracle Instant Client については、Oracle Solaris Studio IDE での Oracle Instant Client の使用法に関するページを参照してください。

リモート開発

リモート開発モードの選択については、Oracle Solaris Studio でのリモート開発モードの選択方法に関するページを参照してください。

リモート開発に関する一般情報については、IDE の「C/C++ リモート開発の概要」ヘルプページを参照してください。このページでは、リモート開発に関するその他の有用なページも紹介されています。

バージョン管理システム

IDE は、Mercurial、Subversion、CVS、Git などのさまざまなバージョン管理システムを使用するオプションと統合されているため、プロジェクトのソースコード管理が容易になります。詳細については、バージョン管理下での IDE プロジェクトの使用法に関するページを参照してください。