

# Oracle® Solaris Studio 12.2 IDE ク イックスタートチュートリアル

2010年9月

- 2 ページの「プロジェクトの作成」
- 8 ページの「プロジェクトの実行」
- 9 ページの「既存のコードからのプロジェクトの作成」
- 10 ページの「リモート開発の実行」
- 12 ページの「アプリケーションのパッケージ作成」
- 14 ページの「ソースファイルの編集」
- 22 ページの「ソースファイルのナビゲーション」
- 28 ページの「ブレークポイントの作成」
- 30 ページの「プロジェクトのデバッグ」
- 32 ページの「機械命令レベルでのデバッグ」
- 34 ページの「実行中のプログラムを接続してデバッグ」
- 35 ページの「既存のコアファイルのデバッグ」

## プロジェクトの作成

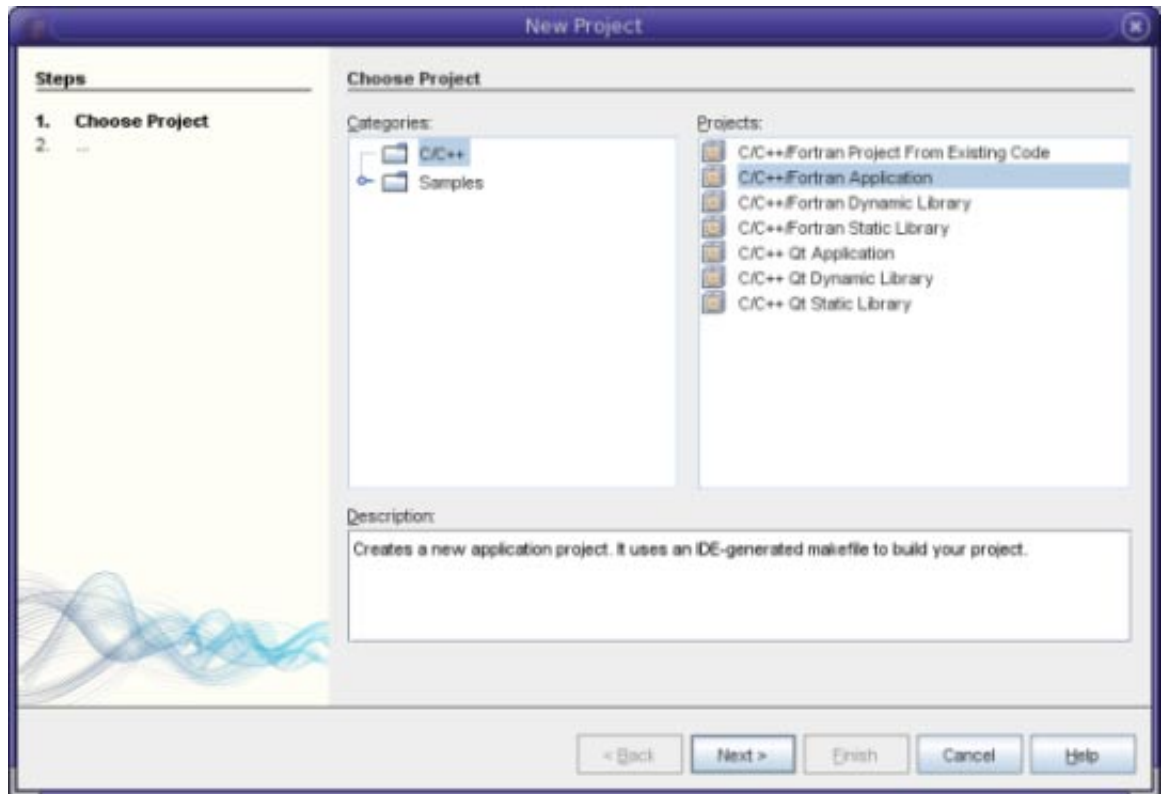
Oracle Solaris Studio では、生成済みメイクファイルを使用して C、C++、および Fortran アプリケーションやライブラリのプロジェクトを作成したり、既存のソースコードを持つプロジェクトを作成したりすることができます。

プロジェクトの構築、実行、およびデバッグは、IDE を起動したローカルホスト上、および Solaris オペレーティングシステムもしくは Linux 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で行うことができます。

C/C++/Fortran アプリケーション、動的ライブラリ、もしくは静的ライブラリプロジェクトでは、IDE はアプリケーションの構築、実行、およびデバッグ方法のあらゆる面を制御します。プロジェクトを作成する際、または「プロジェクトのプロパティ (Project Properties)」ダイアログボックスで、プロジェクト設定を指定します。IDE はメイクファイルを生成し、ここにはすべての設定が保存されます。

### アプリケーションプロジェクトの作成

1. 「ファイル (File)」 > 「新規プロジェクト (New Project)」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++ 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 (C/C++/Fortran Application)」を選択して、「次へ (Next)」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 (Finish)」をクリックしてウィザードを終了します。

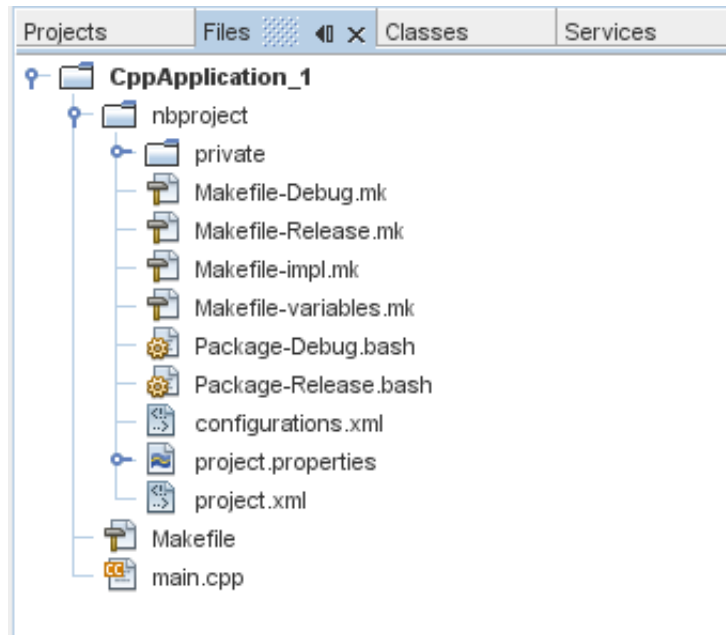
プロジェクトが作成され、いくつかの論理フォルダが作成されます。論理フォルダはディレクトリではありません。ファイルを整理する手段であり、ファイルがディスク上に物理的に保存される場所を示すものではありません。論理フォルダに追加されるファイルは自動的にプロジェクトの一部となり、プロジェクトを構築する際にコンパイルされます。

「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。

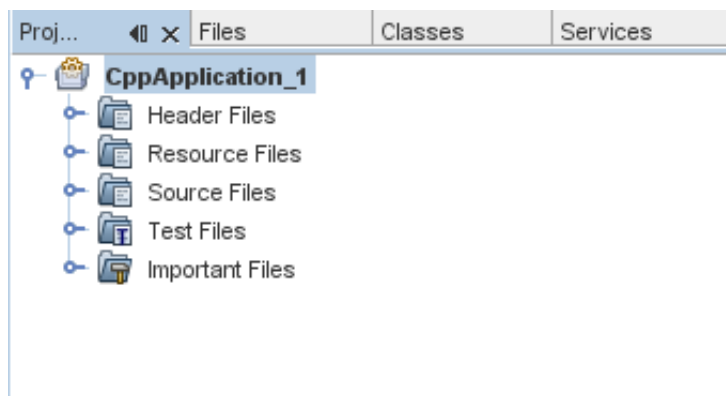
## プロジェクトの論理ビューと物理ビューの切り換え

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 (Files)」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 (Projects)」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。



## ファイルとフォルダのプロジェクトへの追加

論理フォルダを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 (New Logical Folder)」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 (Rename)」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。

## 新規ファイルのプロジェクトへの追加

新しいファイルをプロジェクトに追加できます。

1. 「ソースファイル (Source Files)」フォルダを右クリックして、「新規 (New)」 > 「C main ファイル (Main C File)」を選択します。
2. 「名前と場所 (Name and Location)」ページで、newfile が「ファイル名 (File Name)」フィールドに表示されます。

3. 「完了 (Finish)」をクリックします。

newfile.c 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、「ソースファイル (Source Files)」フォルダに追加されます。このフォルダには、ソースファイルだけでなく、任意の種類ファイルを追加できます。

## その他の新規ファイルのプロジェクトへの追加

1. 「ヘッダーファイル (Header Files)」フォルダを右クリックして、「新規 (New)」> 「C ヘッダーファイル (C Header File)」を選択します。
2. 「名前と場所 (Name and Location)」ページで、newfile が「ファイル名 (File Name)」フィールドに表示されます。
3. 「完了 (Finish)」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 (Header Files)」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 (Source Files)」フォルダを右クリックして、「既存の項目を追加 (Add Existing Item)」を選択します。「項目を選択 (Select Item)」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 (Source Files)」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 (Add Existing Items from Folders)」を選択します。「フォルダの追加 (Add Folders)」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。

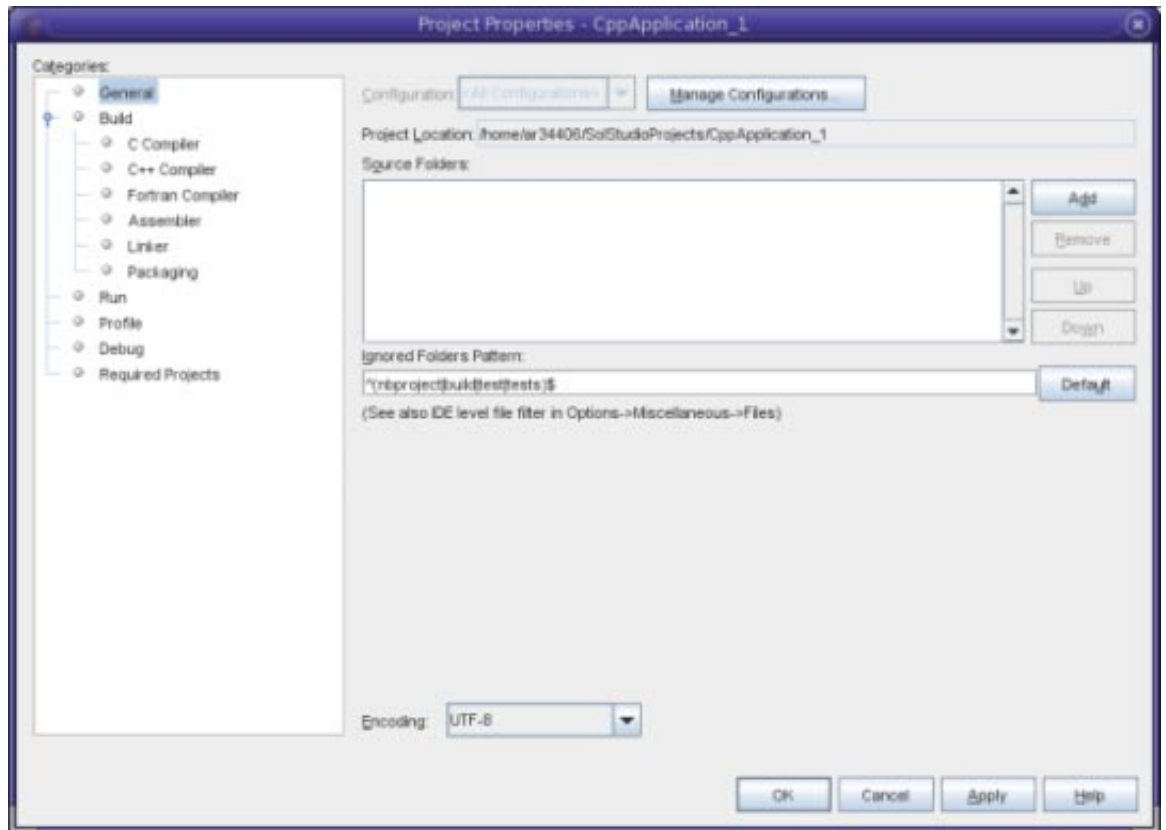
既存の項目の追加に「新規 (New)」メニュー項目を使用しないでください。「名前と場所 (Name and Location)」パネルから、ファイルがすでに存在していることが通知されます。

## プロジェクトプロパティの設定

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、多くのプロパティ設定を一度に簡単に切り換えられます。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 (Project Properties)」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 (Project Properties)」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 (Properties)」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を変更して、「プロジェクトのプロパティ (Project Properties)」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値を選択して、設定できるプロパティに注目します。一般プロパティを設定すると、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を設定すると、現在設定されている構成のプロパティが設定されます。

## 構成の管理

「プロジェクトのプロパティ (Project Properties)」ダイアログボックスで変更されたプロパティは、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 (Project Properties)」ダイアログボックスで「構成を管理 (Manage Configurations)」ボタンをクリックします。
2. 「構成 (Configurations)」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 (Copy)」ボタンをクリックします。その後、「名前を変更 (Rename)」をクリックします。
3. 「名前を変更 (Rename)」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 (Configurations)」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 (Project Properties)」ダイアログボックスで、左側のパネルの「C コンパイラ (C Compiler)」ノードを選択します。PerformanceRelease 構成は「構成 (Configuration)」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シートで、「開発モード (Development Mode)」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。

## ソースファイルのプロパティの設定

プロジェクトにプロジェクトプロパティを設定すると、関連するプロパティ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を設定できます。

1. newfile.c ソースファイルを右クリックして、「プロパティ (Properties)」を選択します。
2. 「カテゴリ (Categories)」パネルの「一般 (General)」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 (C Compiler)」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を上書きできることを確認します。
4. 「プロジェクトのプロパティ (Project Properties)」ダイアログボックスをキャンセルします。

## 主プロジェクトの設定

「プロジェクト (Projects)」ウィンドウのプロジェクトノードを右クリックすると、選択したプロジェクトで実行できるアクションのポップアップメニューが表示されます。同時に複数のプロジェクトを開くと、プロジェクトノードのポップアップメニューが開き、そのプロジェクトで操作していることがわかります。

メニューバーおよびツールバー上のプロジェクト関連のアクションの多くは、主プロジェクトに対して動作します。主プロジェクトノードは「プロジェクト (Projects)」ウィンドウでボールドテキストで表示されます。

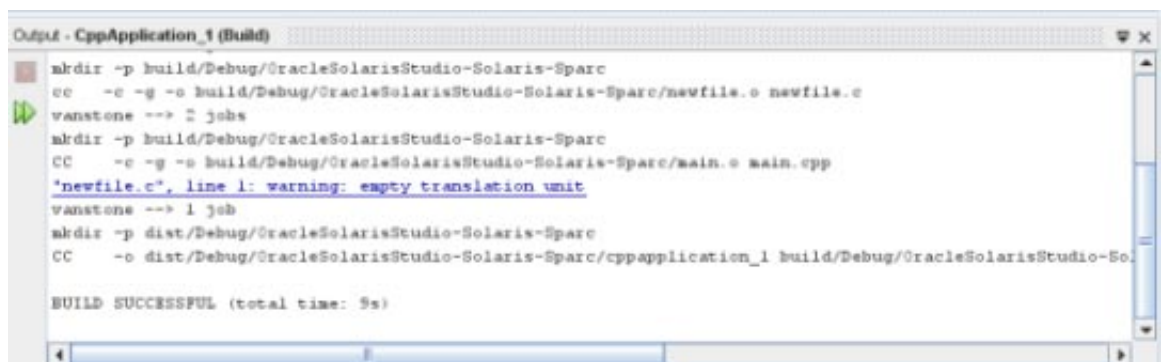
IDE で主プロジェクトを変更するには、次の手順に従います。

- 目的のプロジェクトノードを右クリックして、「主プロジェクトとして設定 (Set as Main Project)」を選択します。このプロジェクトは IDE で主プロジェクトとなり、メニューバーおよびツールバーのアクションはこのプロジェクトを参照するようになります。

## 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 (Build)」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 (Output)」ウィンドウに表示されます。



```
Output - CppApplication_1 (Build)
mkdir -p build/Debug/OracleSolarisStudio-Solaris-Sparc
cc -c -g -o build/Debug/OracleSolarisStudio-Solaris-Sparc/newfile.o newfile.c
vanstone --> 2 jobs
mkdir -p build/Debug/OracleSolarisStudio-Solaris-Sparc
CC -c -g -o build/Debug/OracleSolarisStudio-Solaris-Sparc/main.o main.cpp
'newfile.c', line 1: warning: empty translation unit
vanstone --> 1 job
mkdir -p dist/Debug/OracleSolarisStudio-Solaris-Sparc
CC -o dist/Debug/OracleSolarisStudio-Solaris-Sparc/cppapplication_1 build/Debug/OracleSolarisStudio-So.

BUILD SUCCESSFUL (total time: 9s)
```

2. メインツールバーの「構成 (configuration)」ドロップダウンリストで、構成を Debug から PerformanceRelease に変更します。プロジェクトは PerformanceRelease 構成を使用して構築されるようになります。
3. プロジェクトを右クリックして、「構築 (Build)」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 (Output)」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 (Run)」 > 「主プロジェクトをバッチ構築 (Batch Build Main Project)」を選択して、「バッチ構築 (Batch Build)」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。

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

---

注 - 単体のファイルのコンパイルは、既存のコードからの C/C++/Fortran プロジェクトのプロジェクトタイプではサポートされていません。

---

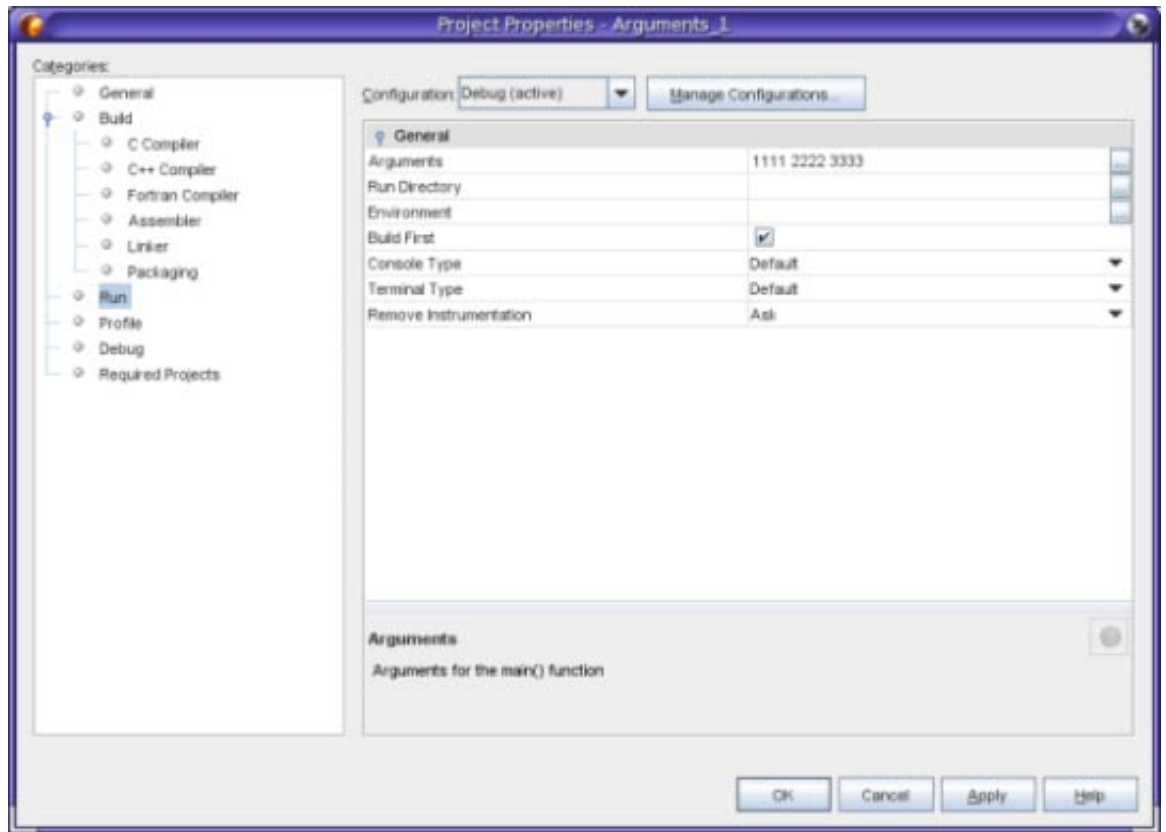
## プロジェクトの実行

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 (File)」 > 「新規プロジェクト (New Project)」を選択します。
2. プロジェクトウィザードで、「サンプル (Samples)」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 (Next)」をクリックして、「完了 (Finish)」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 (Build)」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 (Properties)」を選択します。
6. 「プロジェクトのプロパティ (Project Properties)」ダイアログボックスで、「実行 (Run)」ノードを選択します。
7. 「引数 (Arguments)」テキストフィールドに、1111 2222 3333 と入力します。「OK」をクリックします。





8. 「実行 (Run)」 > 「主プロジェクトを実行 (Run Main Project)」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。

---

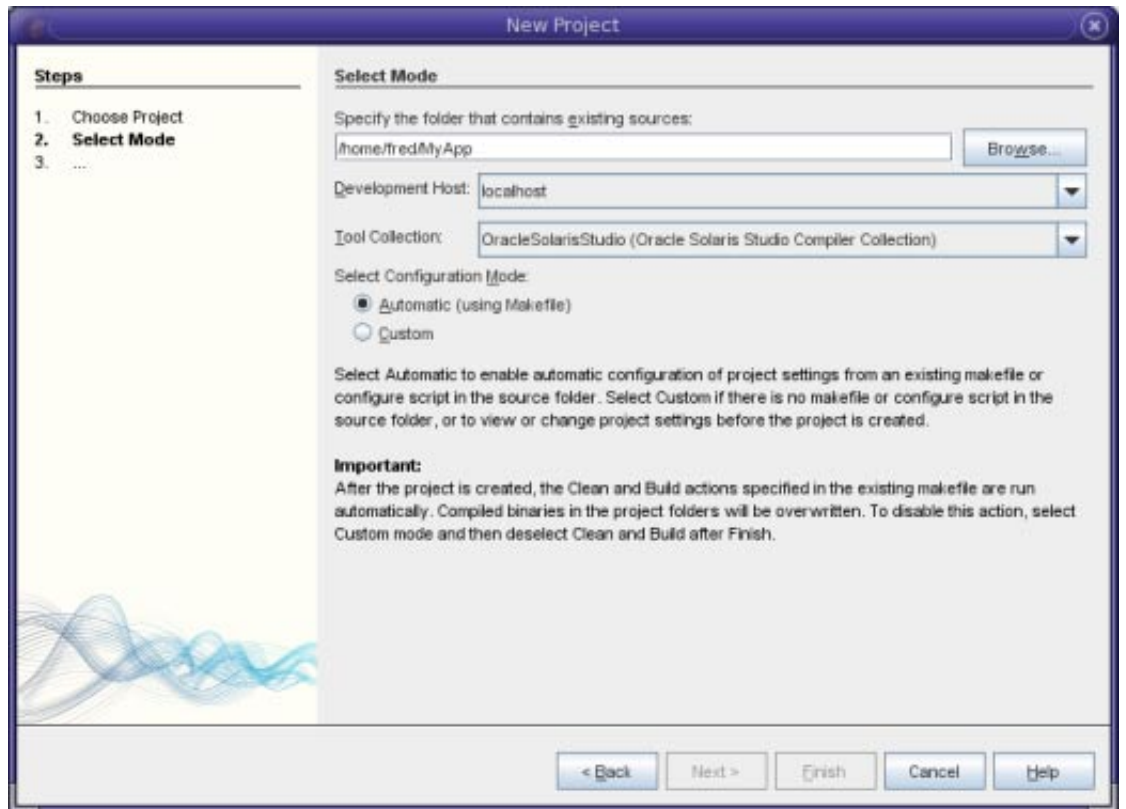
ヒント-プロジェクトを実行すると「モニターの実行 (Run Monitor)」タブが開き、アプリケーションの動作を監視するプロファイルツールが表示されます。プロファイルツールは「プロジェクトのプロパティ (Project Properties)」ダイアログボックスでオフにできます。

---

## 既存のコードからのプロジェクトの作成

「既存のコードからの C/C++/Fortran プロジェクト (C/C++/Fortran Project From Existing Code)」で、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 (File)」 > 「新規プロジェクト (New Project)」を選択します。
2. C/C++ カテゴリを選択します。
3. 「既存のコードからの C/C++/Fortran プロジェクト (C/C++/Fortran Project From Existing Code)」を選択して「次へ (Next)」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 (Select Mode)」ページで、「参照 (Browse)」ボタンをクリックします。「プロジェクトフォルダを選択 (Select Project Folder)」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 (Select)」をクリックします。



5. デフォルトの構成モード「自動(Automatic)」を使用します。「完了(Finish)」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(Projects)」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設定されます。

プロジェクトが作成され、「プロジェクト(Projects)」ウィンドウで開きます。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(Build)」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(Clean and Build)」を選択します。

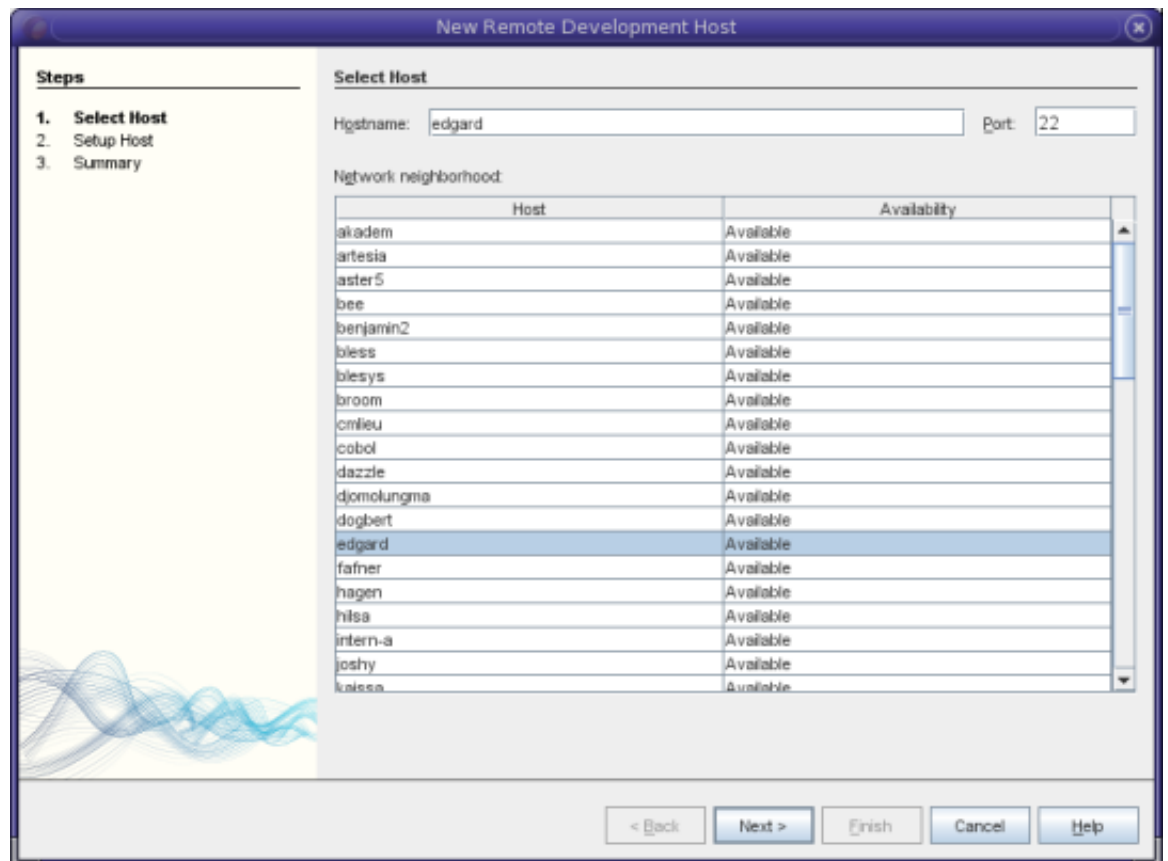
## リモート開発の実行

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。

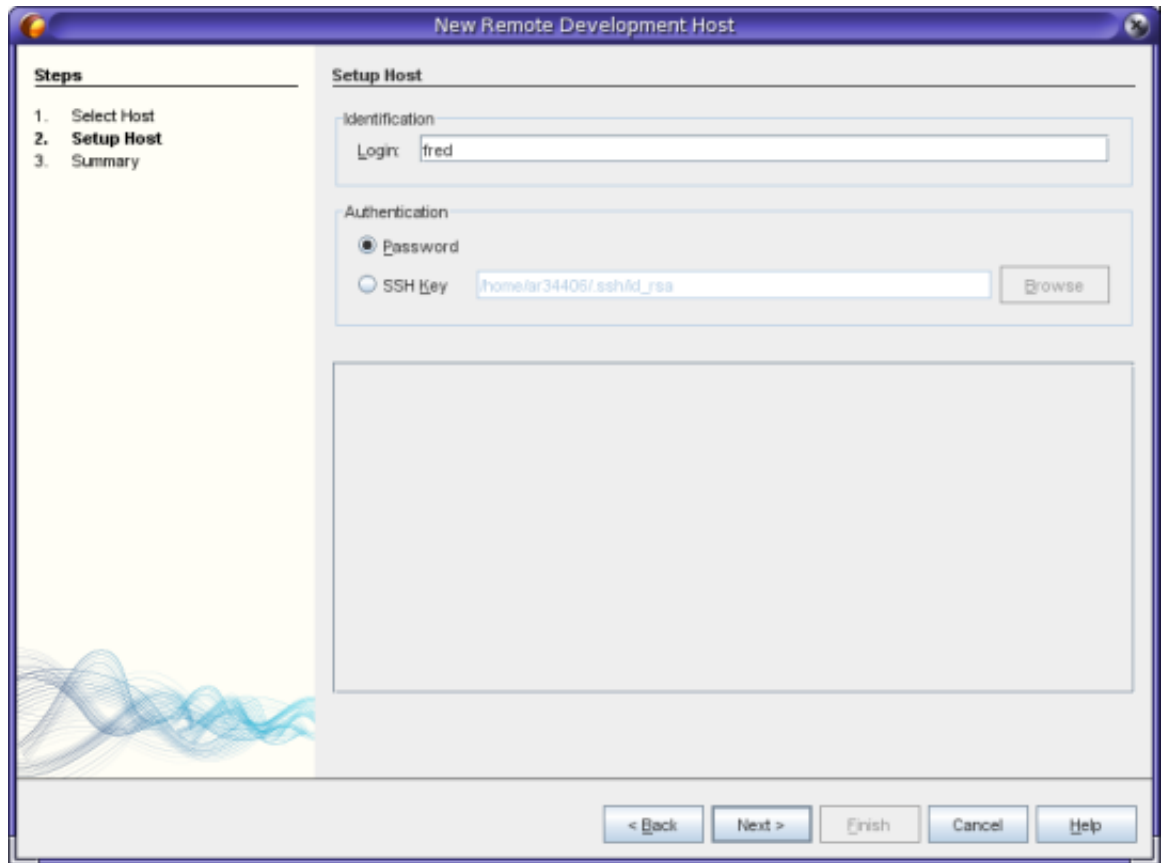
「オプション(Options)」ダイアログボックスの「構築ツール(Build Tools)」タブで、リモート開発ホストを定義できます。リモートホストを追加するには、次の手順に従います。

1. 「ツール(Tools)」>「オプション(Options)」を選択します。
2. 「オプション(Options)」ダイアログボックスの「構築ツール(Build Tools)」タブで、「編集(Edit)」をクリックします。
3. 「開発ホストマネージャー(Development Host Manager)」ダイアログボックスで、「追加(Add)」をクリックします。

- 「新規リモート開発ホスト(New Remote Development Host)」ウィザードの「ホストを選択 (Select Host)」ページで、「ホスト名 (Hostname)」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 (Network neighborhood)」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。



- 「ホストのセットアップ (Setup Host)」ページで、「ログイン (Login)」フィールドにログイン名を入力して「次へ (Next)」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「概要 (Summary)」ページが表示されます。「完了 (Finish)」をクリックします。
7. ホストが「開発ホストマネージャー (Development Hosts Manager)」ダイアログボックスの「開発ホスト (Development Hosts)」リストに追加されたら、「OK」をクリックします。

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。

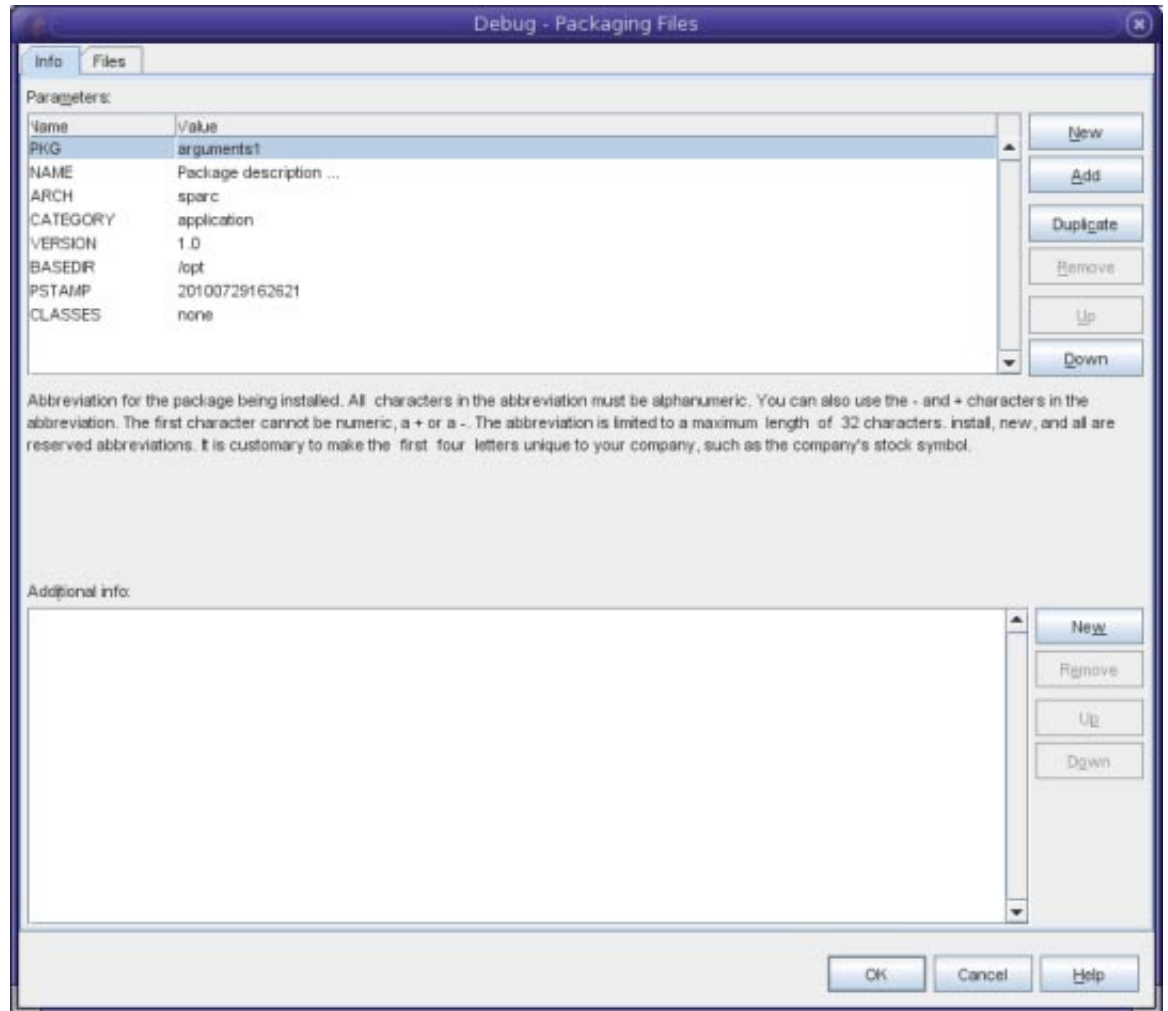
プロジェクトを作成するとき、現在の開発ホストがプロジェクトの開発ホストとして選択されます。「プロジェクトのプロパティ (Project Properties)」ダイアログボックスの「構築 (Build)」パネルで、プロジェクトの開発ホストを変更できます。実行可能ファイルまたはコアファイルをデバッグするときに、開発ホストを指定することもできます。

## アプリケーションのパッケージ作成

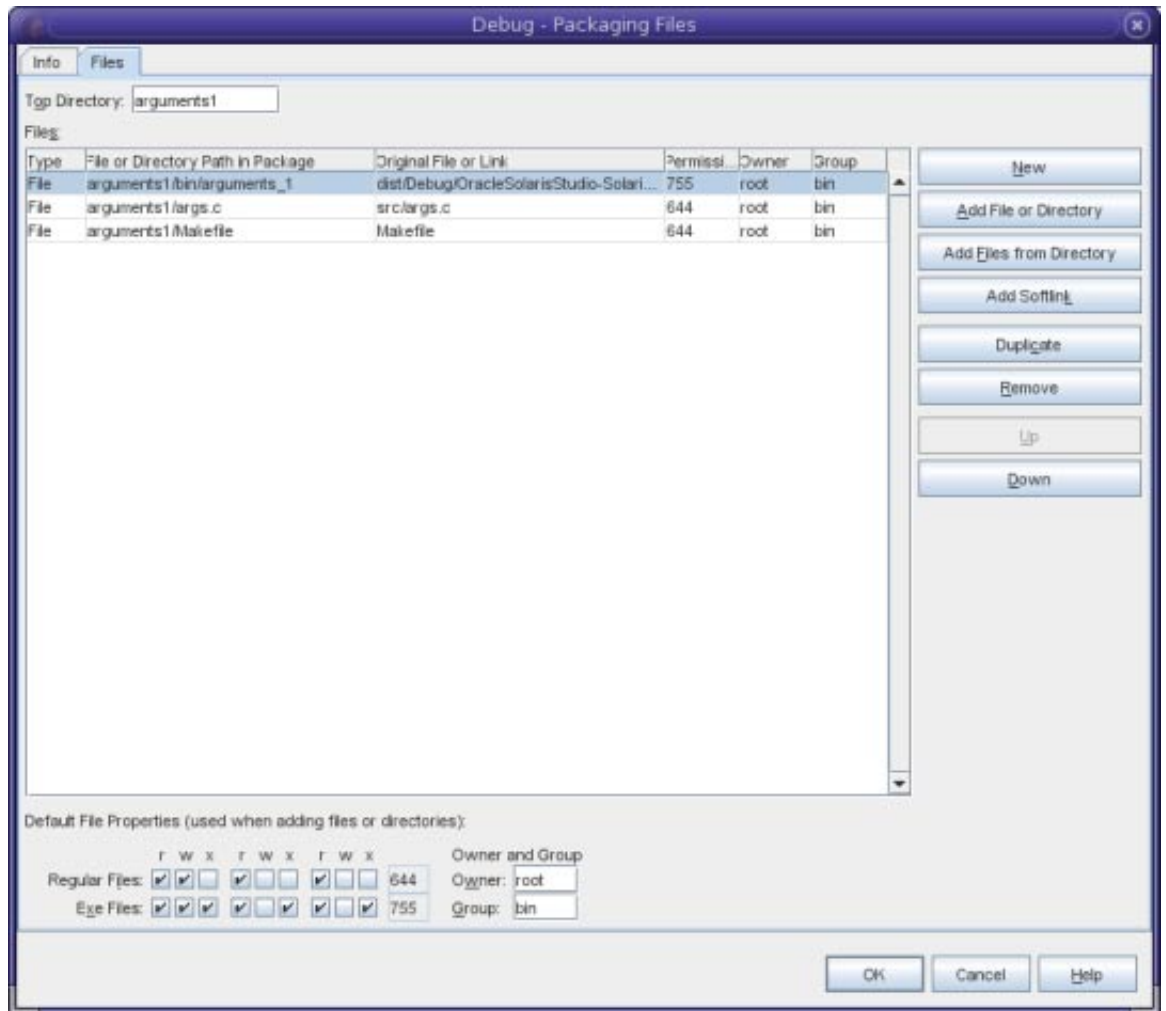
完成したアプリケーションを tar ファイル、zip ファイル、Solaris SVR4 パッケージ、RPM、もしくは Debian パッケージとしてパッケージできます。

1. Arguments\_1 プロジェクトを右クリックして、「プロパティ (Properties)」を選択します。
2. 「プロジェクトのプロパティ (Project Properties)」ダイアログボックスで、「パッケージング (Packaging)」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、出力パスを変更しします。

5. 「ファイルのパッケージング (Packaging Files)」 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 (Packaging Files)」 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 (Info)」 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 (Files)」 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 (Files)」 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 (File or Directory Path in Package)」列に、パッケージ内のパスを指定できます。「ファイル (Files)」 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 (More Build Commands)」>「パッケージの構築 (Build Package)」を選択します。

## 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役に立ちます。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。

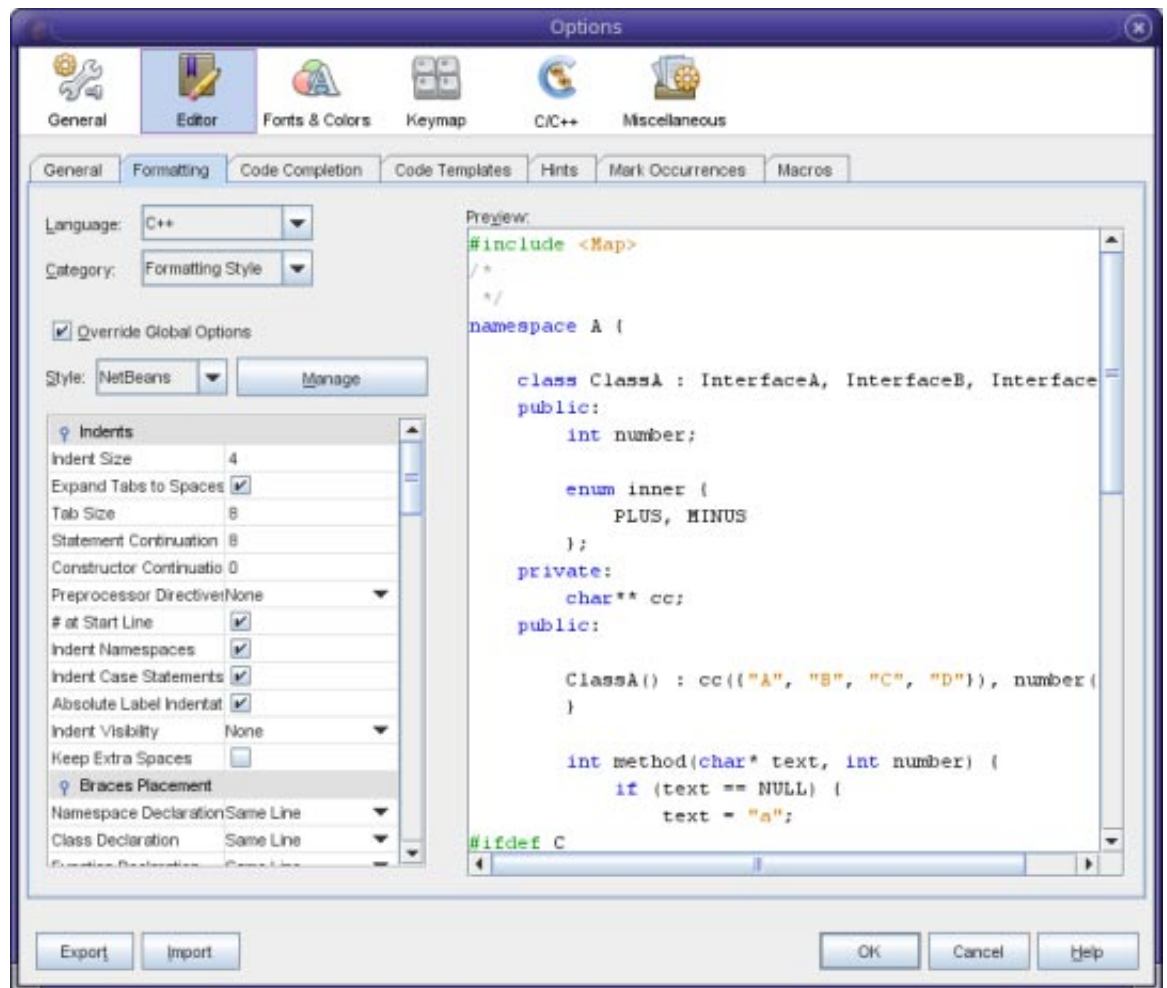
1. 「ファイル (File)」>「新規プロジェクト (New Project)」を選択します。
2. プロジェクトウィザードで、「サンプル (Samples)」カテゴリと「C/C++」サブカテゴリを展開して、Quote プロジェクトを選択します。「次へ (Next)」をクリックして、「完了 (Finish)」をクリックします。

## 書式設定スタイルの設定

「オプション (Options)」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設定スタイルを設定できます。

1. 「ツール (Tools)」>「オプション (Options)」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 (Editor)」をクリックします。
3. 「書式設定 (Formatting)」タブをクリックします。

- 「言語 (Language)」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。
- 「スタイル (Style)」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。



- 必要に応じてスタイルプロパティを変更します。

## 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。

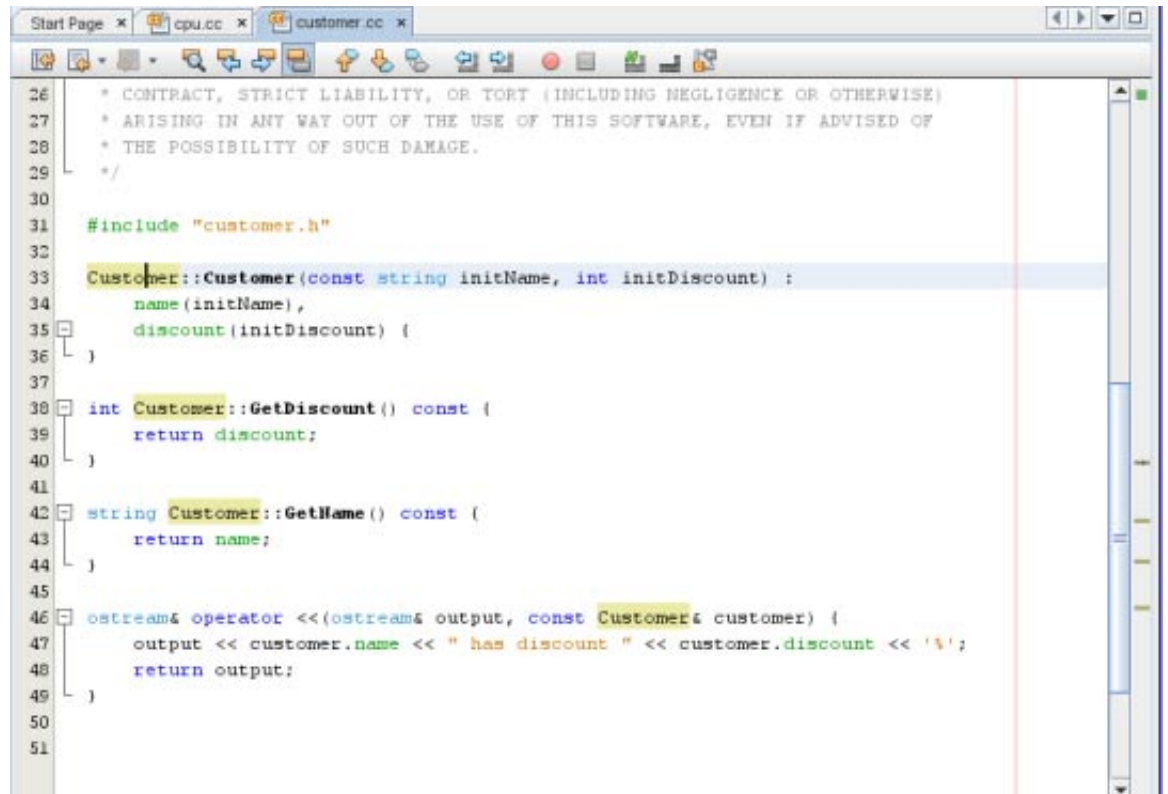
- Quote\_1アプリケーションプロジェクトで、「ソースファイル (Source Files)」フォルダを開き、cpu.cc ファイルをダブルクリックしてソースエディタで開きます。
- 左端の折り畳みアイコン (マイナス記号付きの小さなボックス) をクリックして、メソッドの1つのコードを折り畳みます。
- 折り畳んだブロックの右側の {...} 記号にマウスオーバーして、ブロック内のコードを表示します。

## 意味上の強調表示の使用

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。

- 「ツール (Tools)」 > 「オプション (Options)」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 (Highlighting)」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。



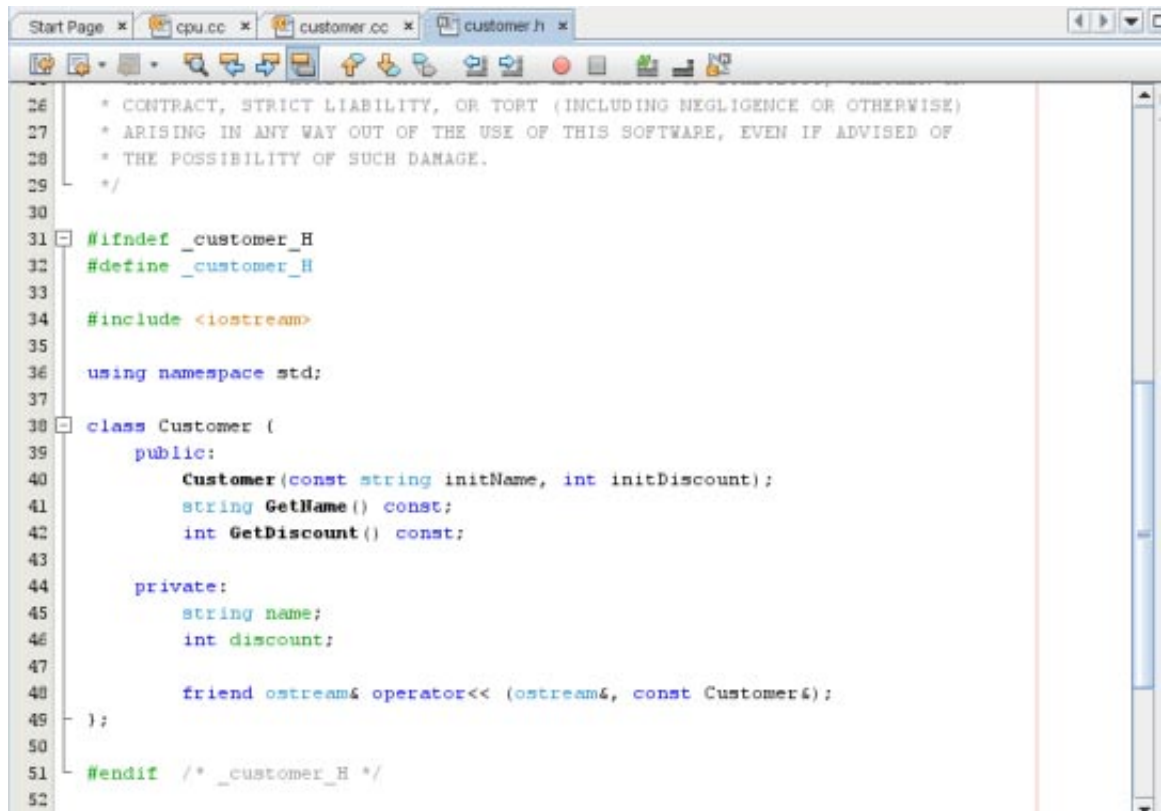
The screenshot shows the Oracle Solaris Studio IDE interface. The main window displays the C++ source file customer.cc. The code is as follows:

```
26  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
27  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
28  * THE POSSIBILITY OF SUCH DAMAGE.
29  */
30
31  #include "customer.h"
32
33  Customer::Customer(const string initName, int initDiscount) :
34      name(initName),
35      discount(initDiscount) {
36  }
37
38  int Customer::GetDiscount() const {
39      return discount;
40  }
41
42  string Customer::GetName() const {
43      return name;
44  }
45
46  ostream& operator <<(ostream& output, const Customer& customer) {
47      output << customer.name << " has discount " << customer.discount << '\n';
48      return output;
49  }
50
51
```

In the screenshot, the class name `Customer` in the constructor definition (line 33), the `GetDiscount` method (line 38), the `GetName` method (line 42), and the `operator <<` (line 46) are highlighted with a yellow background. The IDE window title bar shows 'Start Page', 'cpu.cc', and 'customer.cc'.

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。



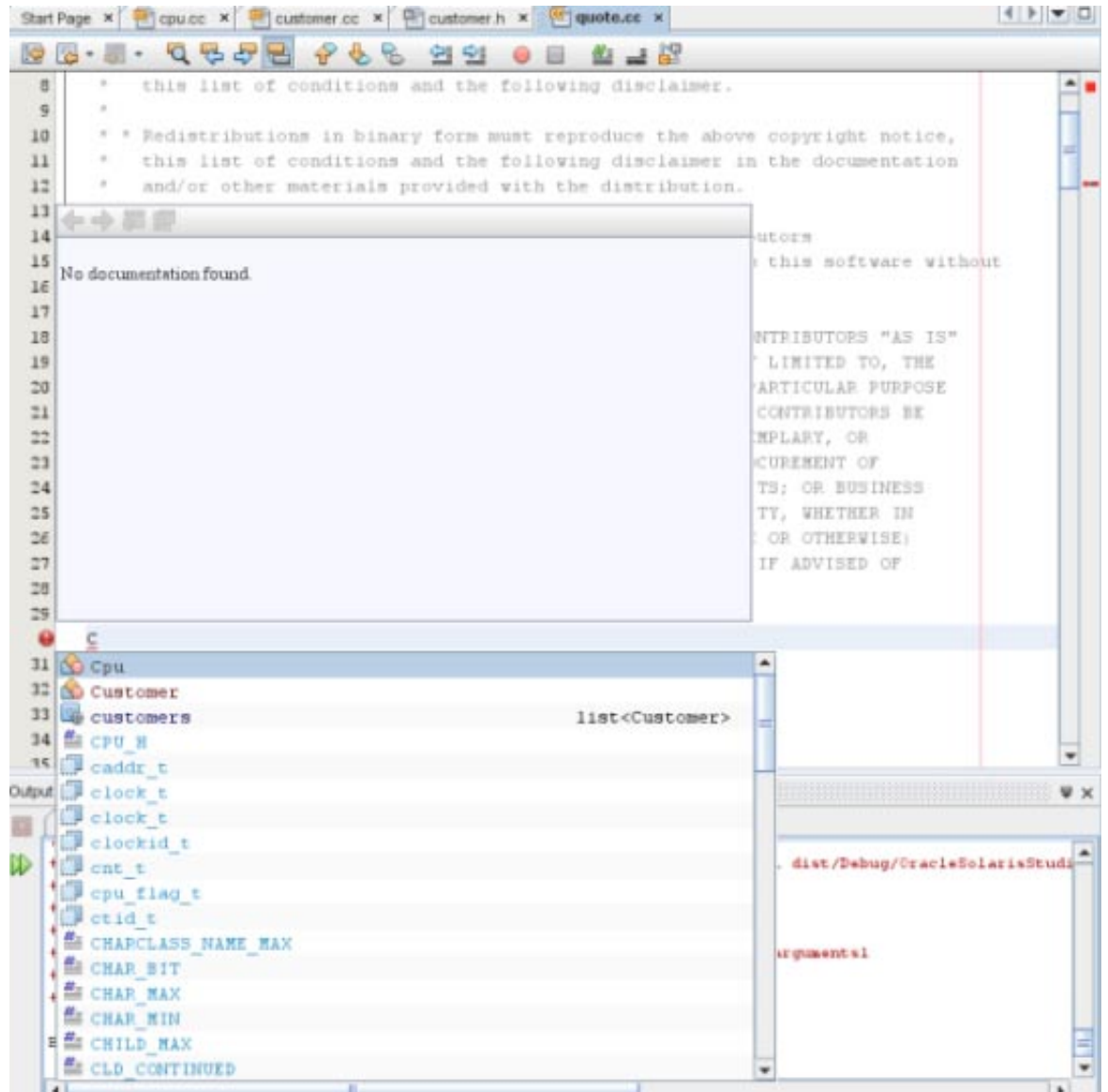


```
26  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
27  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
28  * THE POSSIBILITY OF SUCH DAMAGE.
29  */
30
31  #ifndef _customer_H
32  #define _customer_H
33
34  #include <iostream>
35
36  using namespace std;
37
38  class Customer {
39  public:
40      Customer(const string initName, int initDiscount);
41      string GetName() const;
42      int GetDiscount() const;
43
44  private:
45      string name;
46      int discount;
47
48      friend ostream& operator<< (ostream&, const Customer&);
49  };
50
51  #endif /* _customer_H */
52
```

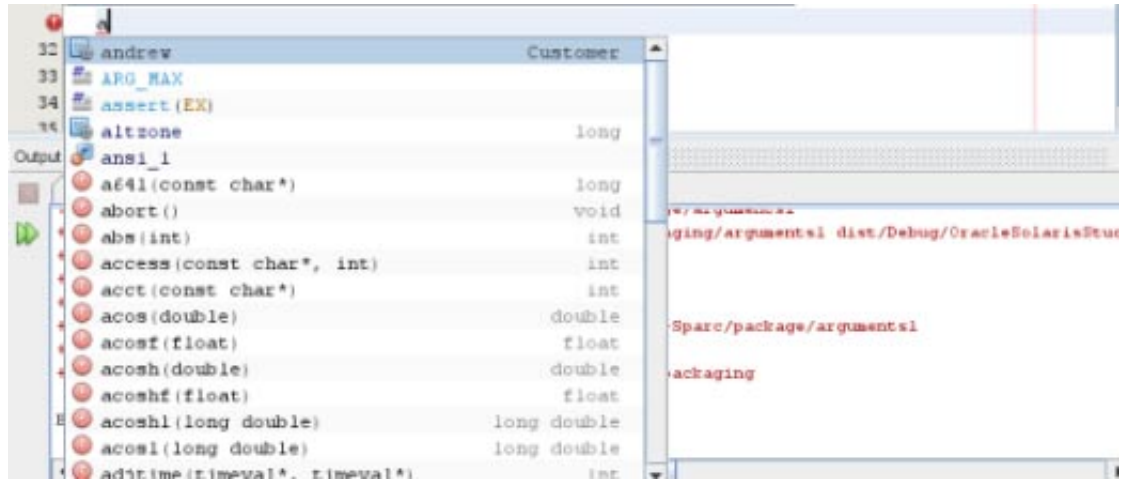
## コード補完の使用

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 (No documentation found)」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、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。



- 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。

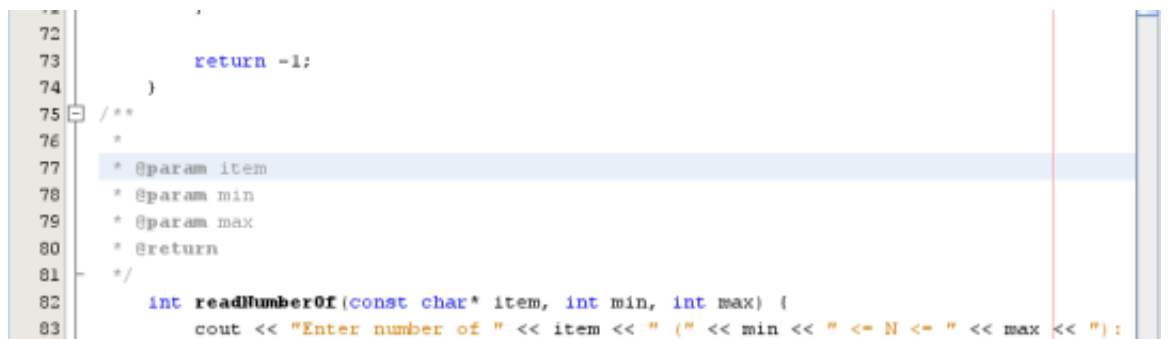


- 追加したコードを削除します。

## 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、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。

- 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。
- スラッシュ1つとアスタリスク2つを入力して Enter を押します。エディタから、Doxygen で書式設定されたコメントが `readNumberOf` クラスに挿入されます。



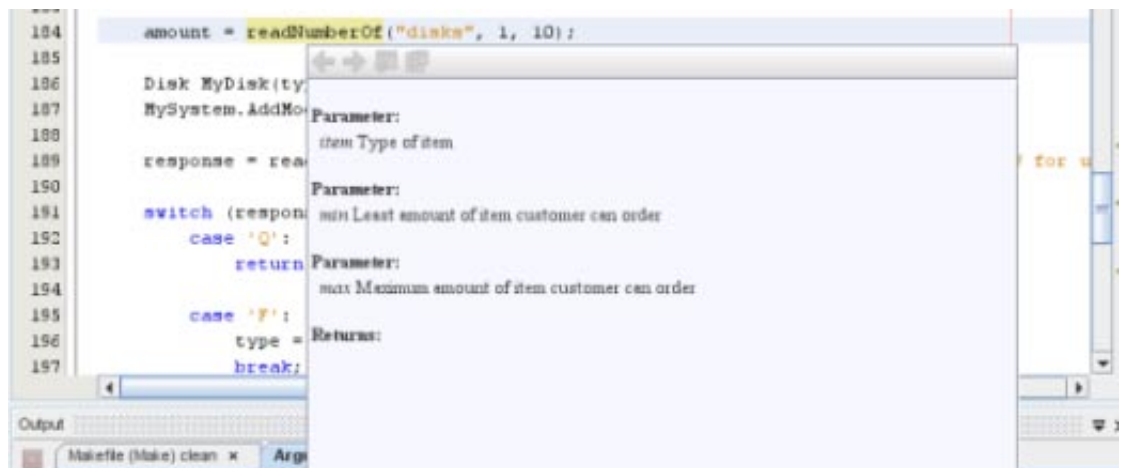
- @param の各行に説明のテキストを追加して、ファイルを保存します。
- `readNumberOf` クラスをクリックして黄色で強調表示し、右側の出現箇所マークの1つをクリックしてクラスが使用されている場所にジャンプします。

```

76 *
77 * @param item Type of item
78 * @param min Least amount of item customer can order
79 * @param max Maximum amount of item customer can order
80 * @return
81 */
82 int readNumberOf(const char* item, int min, int max) {
83     cout << "Enter number of " << item << " (" << min << " <= N <= " << max << "):
84
85     string s;

```

- 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。



- 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。
- 「ソース (Source)」 > 「ドキュメントの表示 (Show documentation)」 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

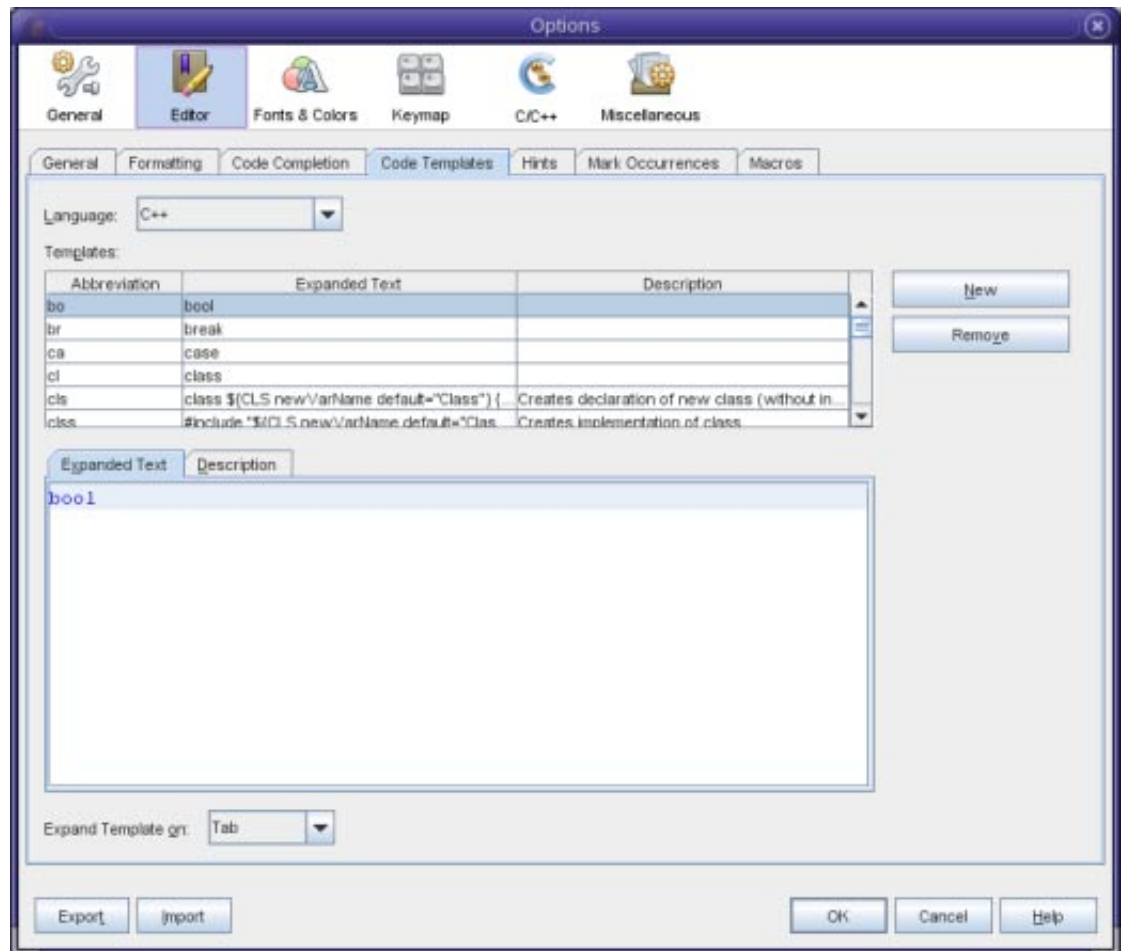
ソースエディタには、C、C++、および Fortran コードの共通スニペット用の、一連のカスタマイズ可能なコードテンプレートがあります。略語を入力して Tab キーを押すと、コードスニペット全体を生成できます。たとえば、Quote\_1 プロジェクトの quote.cc ファイルで、次のようにします。

- 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。

- 「ツール (Tools)」 > 「オプション (Options)」 を選択します。
- 「オプション (Options)」 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 (Code Templates)」 タブをクリックします。

3. 「言語 (Language)」 ドロップダウンリストから言語を選択します。



## 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ファイルの行116の{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。

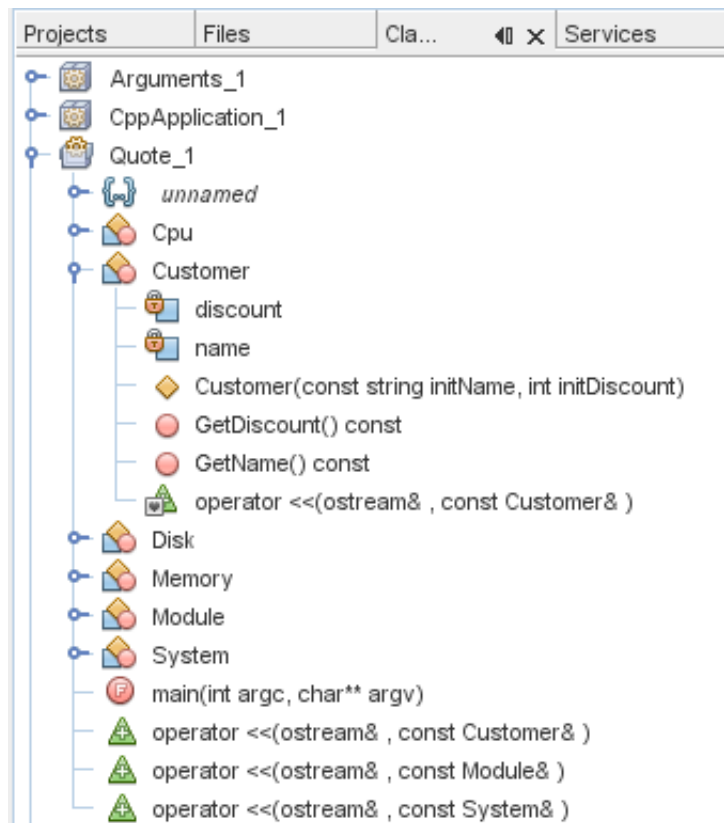
# ソースファイルのナビゲーション

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。

## 「クラス (Classes)」ウィンドウの使用

「クラス (Classes)」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 (Classes)」タブをクリックして、「クラス (Classes)」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。

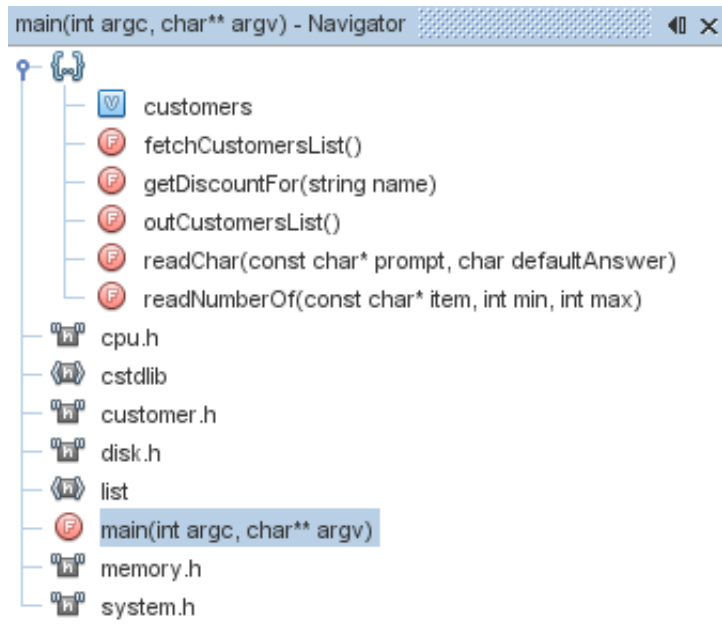


4. name変数をダブルクリックしてcustomer.hヘッダーファイルを開きます。

## 「ナビゲータ (Navigator)」ウィンドウの使用

「ナビゲータ (Navigator)」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 (Navigator)」ウィンドウが開いていない場合、「ウィンドウ (Window)」>「ナビゲート (Navigating)」>「ナビゲータ (Navigator)」の順に選択して開きます。

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 (Navigator)」ウィンドウに表示されます。ウィンドウの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 (Navigator)」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 (Navigator)」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 (Navigator)」ウィンドウにあるアイコンを確認するには、「ヘルプ (Help)」>「ヘルプの目次 (Help Contents)」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 (Search)」タブをクリックして、「検索 (Find)」フィールドにナビゲータアイコンを入力します。

## 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 (Usages)」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。

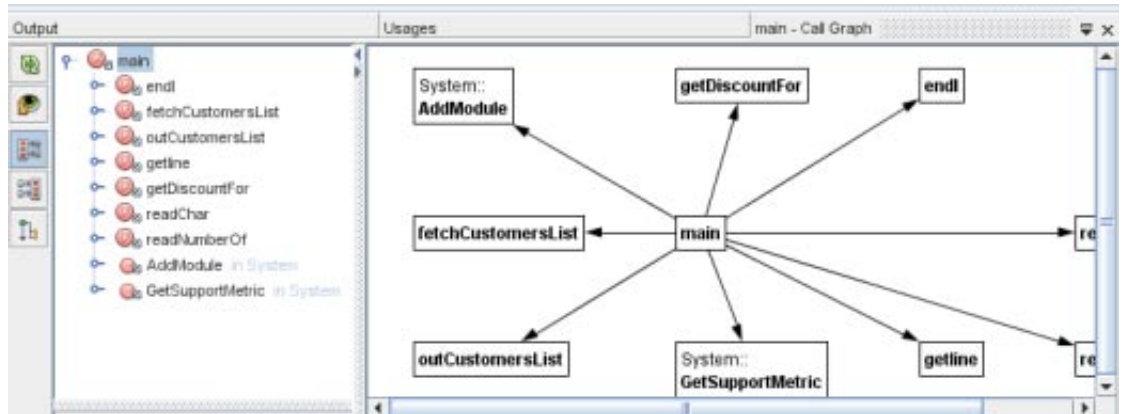
1. customer.cc ファイルで、42 行目の Customer クラスを右クリックして「使用状況を検索 (Find Usages)」を選択します。
2. 「使用状況を検索 (Find Usages)」ダイアログボックスで、「検索 (Find)」をクリックします。
3. 「使用状況 (Usages)」ウィンドウが開き、プロジェクトのソースファイルでの Customer クラスのすべての使用状況が表示されます。



## コールグラフの使用

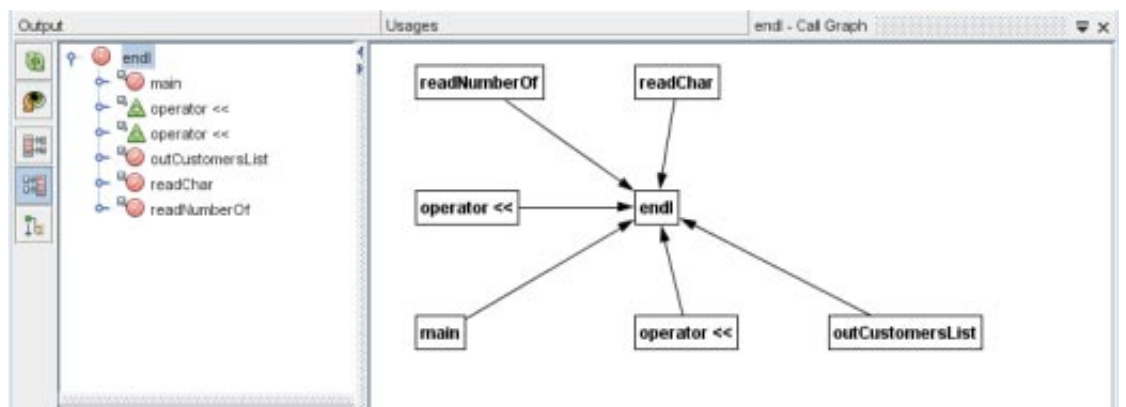
「コールグラフ (Call Graph)」ウィンドウには、クラス内の関数の予呼び出し関係の2つのビューが表示されます。ツリービューに、選択した関数から呼び出された関数、もしくはこの関数を呼び出す関数が表示されます。グラフィカル表示には、呼び出し元および呼び出し先の関数の間に矢印を使用して、呼び出し関係が表示されます。

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 (Show Call Graph)」を選択します。
2. 「コールグラフ (Call Graph)」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。



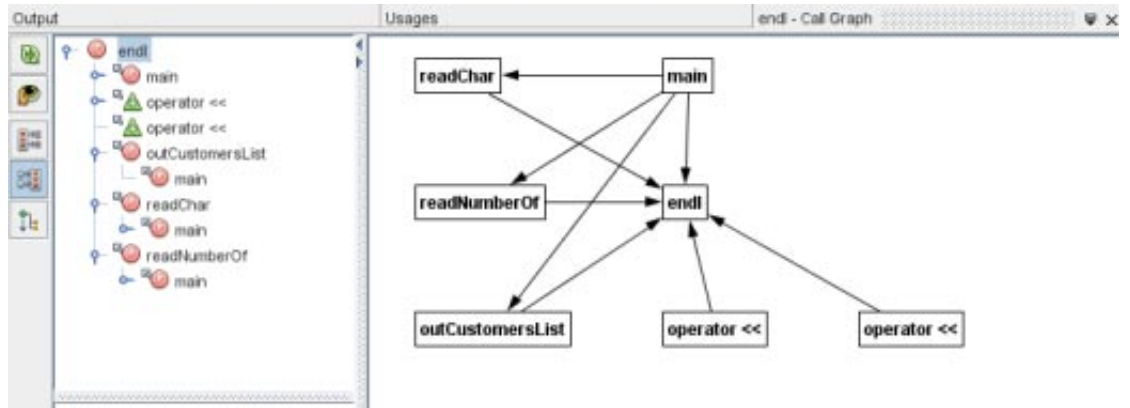
スクリーンショットに示す関数の一部が表示されない場合は、「コールグラフ (Call Graph)」ウィンドウの左側の3番目のボタンをクリックして、この関数から呼び出される関数を表示します。

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2番目のボタンをクリックして、endl 関数にフォーカスし、4番目の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。

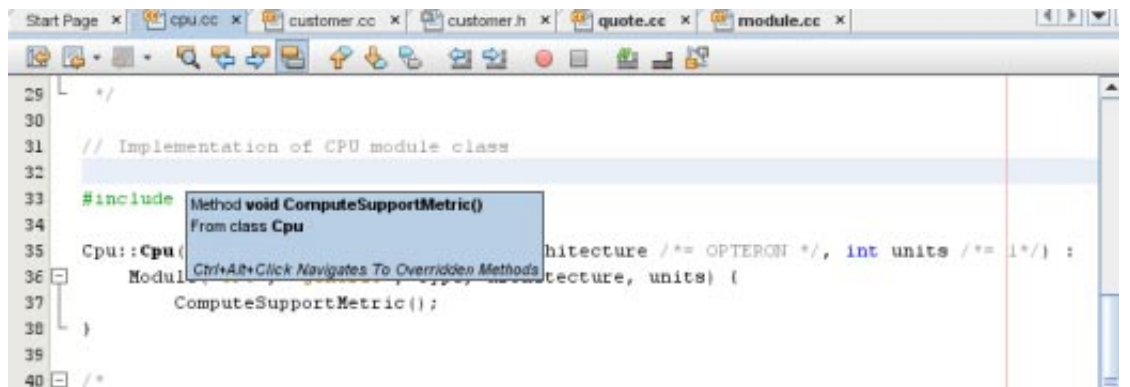




## ハイパーリンクの使用

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。

```

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 |
49 |     switch (GetTypeID()) {
50 |     case MEDIUM:
51 |         metric += 100;
52 |         break;
53 |
54 |     case HIGH:
55 |         metric += 400;

```

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. マウスカーソルを左端の緑の円の上に置くと、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。

```

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 | Overrides Module::ComputeSupportMetric;
46 |
47 | void Cpu::ComputeSupportMetric() {
48 |     int metric = 100 * GetUnits();

```

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。

```

62 virtual const char* GetCategory() const = 0;
63
64 void SetUnits(int u);
65 int GetUnits() const;
66
67 void SetSupportMetric(int m);
68 int GetSupportMetric() const;
69
70 protected:
71 virtual void ComputeSupportMetric() = 0; //metric is defined in derived classes
72     Is Overridden
73     ↓ Cpu::ComputeSupportMetric
74     ↓ Disk::ComputeSupportMetric
75     ↓ Memory::ComputeSupportMetric
76     .s anticipates future functionality
77 int type;

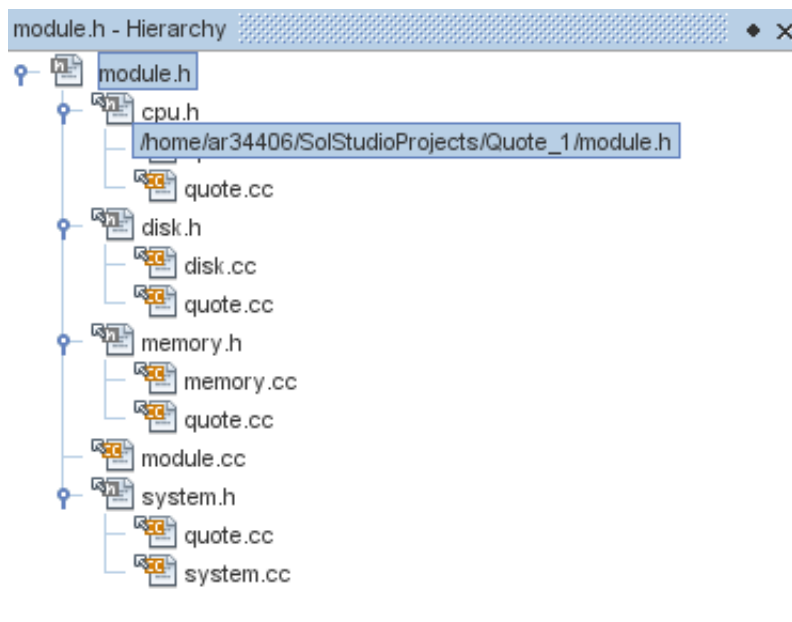
```

- 「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 (Include Hierarchy)」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。

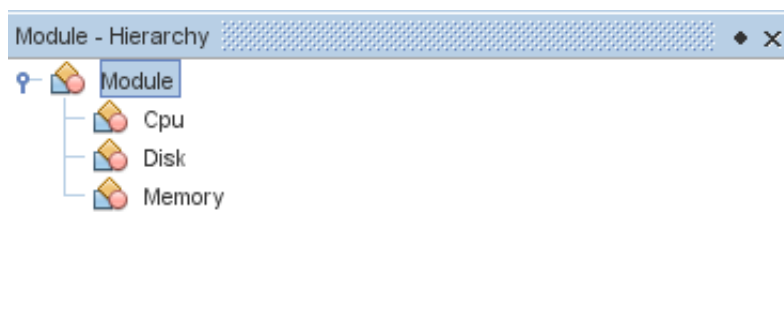
- Quote\_1 プロジェクトで、ソースファイルに module.cc ファイルを開きます。
- ファイルの #include "module.h" 行を右クリックして、「ナビゲート (Navigate)」>「インクルードの階層を表示 (View Includes Hierarchy)」を選択します。
- デフォルトで、「階層 (Hierarchy)」ウィンドウにはヘッダーファイルに直接インクルードされるファイルのプレーンリストが表示されます。ウィンドウ下部の右端のボタンをクリックして、表示をツリービューに変更します。右から2番目のボタンをクリックして、インクルードまたはインクルードされるすべてのファイルに表示を変更します。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 (Type Hierarchy)」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 プロジェクトで、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 (Navigate)」 > 「タイプの階層を表示 (View Type Hierarchy)」を選択します。
3. 「階層 (Hierarchy)」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。

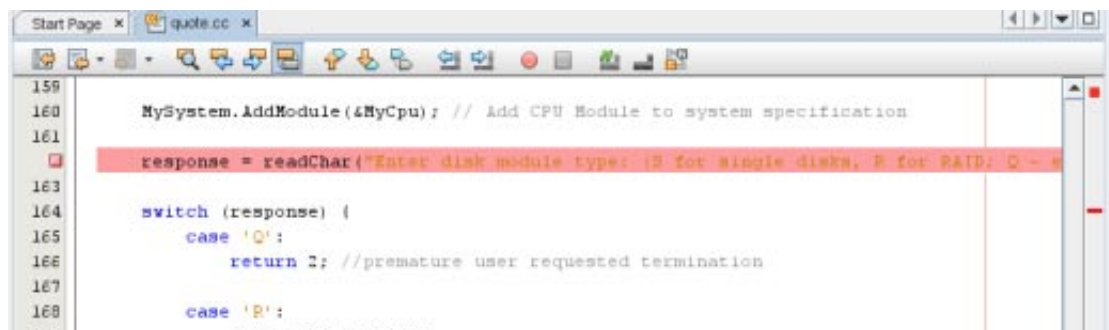


## ブレークポイントの作成

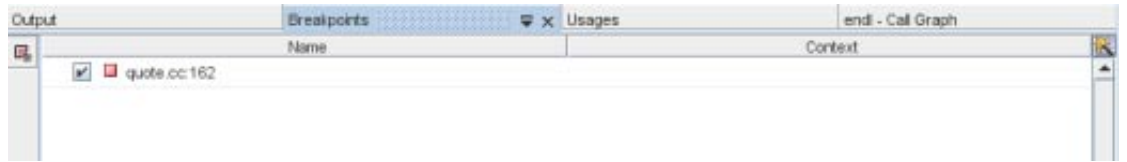
コード内でいつでもブレークポイントを作成し、操作できます。

### 行ブレークポイントの作成と削除

1. Quote\_1 プロジェクトで、quote.cc ファイルを開きます。
2. エディタウィンドウの 173 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。

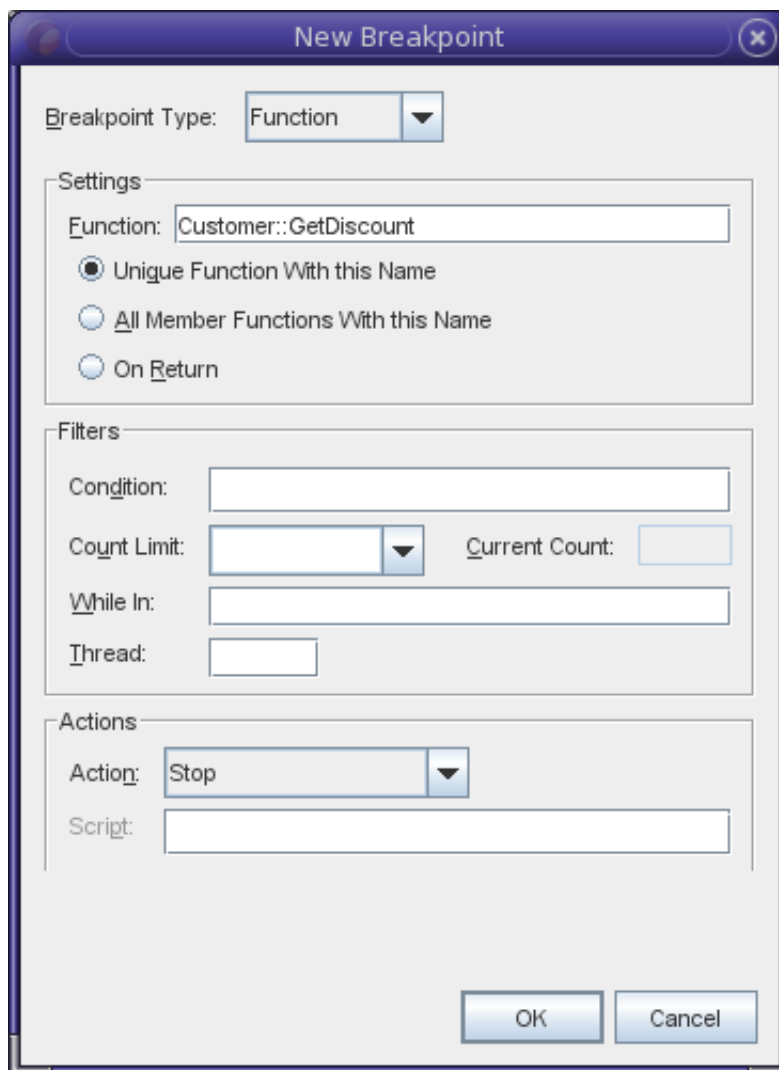


3. 左マージンのアイコンをクリックして、ブレークポイントを削除できます。
4. 「ウィンドウ (Window)」 > 「デバッグ (Debugging)」 > 「ブレークポイント (Breakpoints)」を選択して、「ブレークポイント (Breakpoints)」ウィンドウを開きます。行ブレークポイントがウィンドウに一覧表示されます。

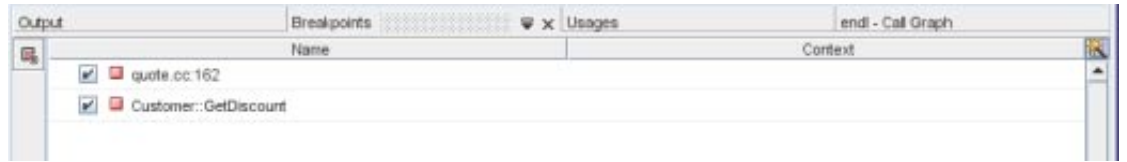


## 関数ブレークポイントの作成

1. 「デバッグ (Debug)」 > 「新規ブレークポイント (New Breakpoint)」 (Ctrl+Shift+f8) を選択して、「新規ブレークポイント (New Breakpoint)」 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 (Breakpoint Type)」 ドロップダウンリストで、タイプを「関数 (Function)」 に設定します。
3. 関数名 `Customer::GetDiscount` を「関数 (Function)」 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 (Breakpoints)」 ウィンドウのリストに追加されます。



## 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dbxデバッガを起動し、デバッガ内部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「出力(Output)」ウィンドウに出力します。

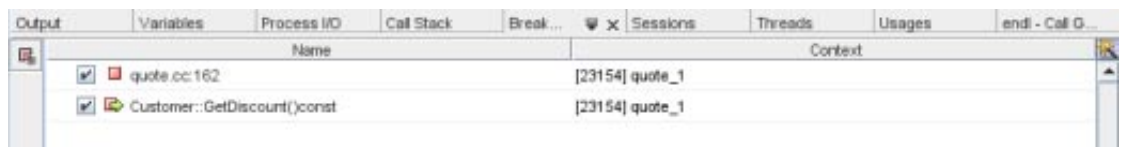
### デバッグセッションを開始する

1. プロジェクトノードを右クリックして「デバッグ(Debug)」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、「デバッグ(Debugging)」ウィンドウに「変数(Variables)」、「呼び出しスタック(Call Stack)」、「スレッド(Threads)」、「プロセスI/O(Process I/O)」、および「Dbxコンソール(Dbx Console)」ウィンドウが開きます。
2. 「ウィンドウ(Window)」>「デバッグ(Debugging)」>「セッション(Sessions)」を選択して、「セッション(Sessions)」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

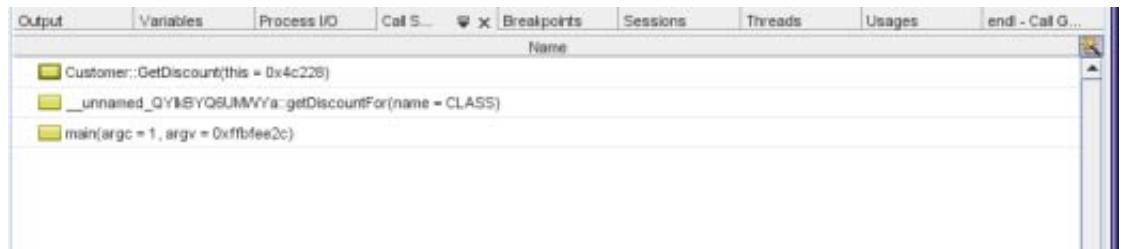
1. Quote\_1アプリケーションから、「プロセスI/O(Process I/O)」ウィンドウに入力するよう通知されます。
2. 「顧客名を入力してください(Enter customer name:)」というメッセージの後に、顧客名を入力します。
3. 以前設定した関数ブレークポイントで、アプリケーションが停止します。「ブレークポイント(Breakpoints)」ウィンドウに、以前設定した2つのブレークポイントが表示されます。関数ブレークポイントのブレークポイントアイコンの上に、緑のプログラムカウンタ矢印が表示されます。



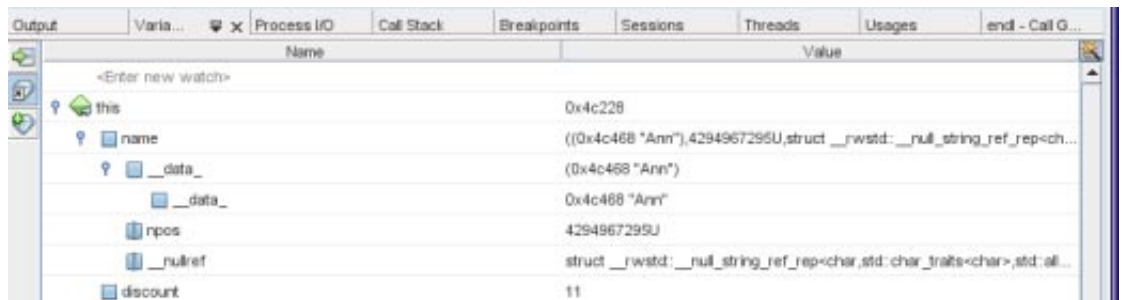
4. customer.ccファイルで、GetDiscount関数の最初の行にあるブレークポイントアイコンの上に、緑のプログラムカウンタ矢印が表示されます。

```
27 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
28 * THE POSSIBILITY OF SUCH DAMAGE.
29 */
30
31 #include "customer.h"
32
33 Customer::Customer(const string initName, int initDiscount) :
34     name(initName),
35     discount(initDiscount) {
36 }
37
38 int Customer::GetDiscount() const {
39     return discount;
40 }
41
42 string Customer::GetName() const {
43     return name;
44 }
```

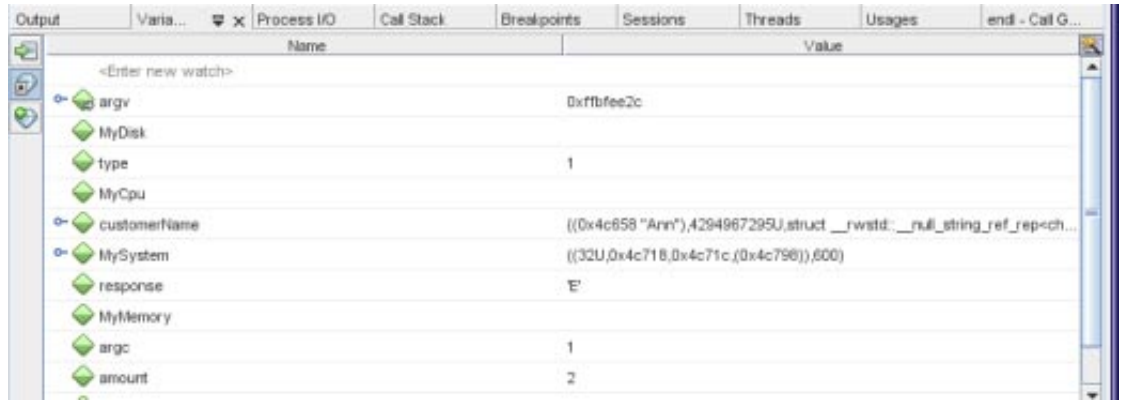
- 「呼び出しスタック (Call Stack)」ウィンドウをクリックします。呼び出しスタックには3つのフレームが表示されます。



- 「変数 (Variables)」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。



- 「続行 (Continue)」ボタンをクリックします。GetDiscount 関数が実行され、「プロセス I/O (Process I/O)」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。
- 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 (Variables)」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。

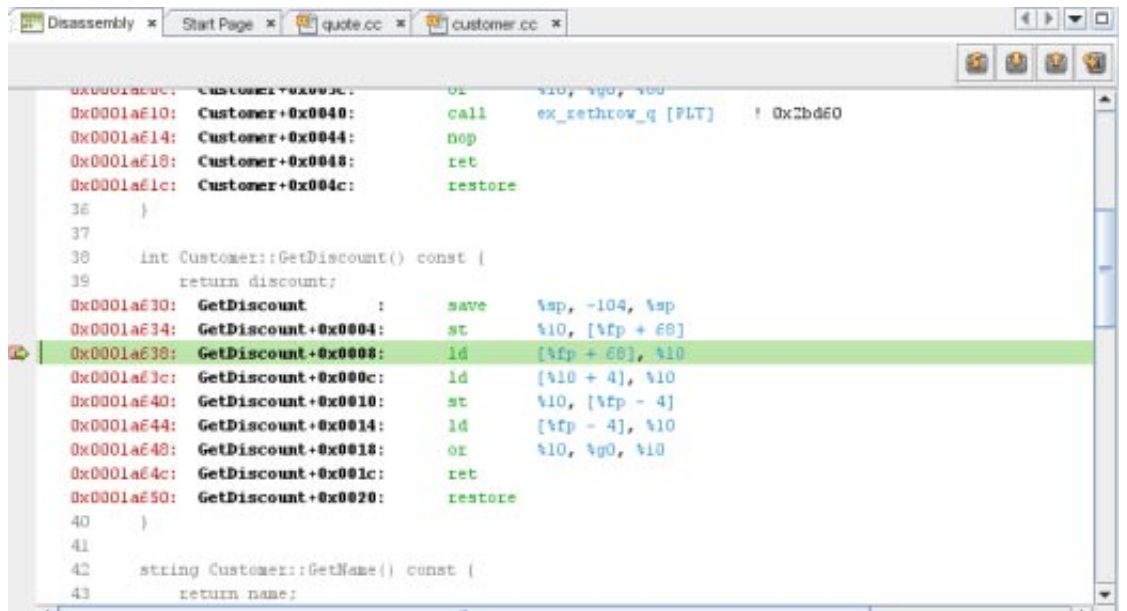


- 「呼び出しスタック (Call Stack)」 ウィンドウをクリックして、スタックにフレームが1つしかないことを確認します。
- 「続行 (Continue)」 ボタンをクリックして、プログラムが完了するまで、「プロセス I/O (Process I/O)」 ウィンドウのプロンプトに従って入力を持続します。プログラムに最後の入力を行うと、デバッグセッションは終了します。プログラムが完了する前にデバッグセッションを終了するには、「セッション (Sessions)」 ウィンドウでセッションを右クリックして「完了 (Finish)」 を選択します。

## 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。

- Quote\_1プロジェクトを右クリックして、「デバッグ (Debug)」 を選択します。
- 「プロセス I/O (Process I/O)」 ウィンドウで、プロンプトに従って顧客名を入力します。
- 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 (Window)」 > 「デバッグ (Debugging)」 > 「逆アセンブリ (Disassembly)」 ウィンドウを選択します。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。



- 「ウィンドウ (Window)」 > 「デバッグ (Debugging)」 > 「レジスタ (Registers)」 を選択して「レジスタ (Registers)」 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。



Name	Value
g0-g1	0x00000000 0x00000000 0x00000000 0x00000000
g2-g3	0x00000000 0x00000010 0x00000000 0xff36e798
g4-g5	0x00000000 0x00000000 0x00000000 0xfefeeaa0
g6-g7	0x00000000 0x00000000 0x00000000 0xff1f2a00
o0-o1	0x00000000 0x00000000 0x00000000 0x00000001
o2-o3	0x00000000 0x00000001 0x00000000 0x00000000
o4-o5	0x00000000 0x0000000e 0x00000000 0x0000000e
o6-o7	0x00000000 0xffbfe80 0x00000000 0xfefcc7a8
l0-l1	0x00000000 0x0004c228 0x00000000 0x0004c228
l2-l3	0x00000000 0x0004c468 0x00000000 0x00000001
l4-l5	0x00000000 0xfefeea78 0x00000000 0x0004c440
l6-l7	0x00000000 0xfefee4b4 0x00000000 0x00000040
l0-l1	0x00000000 0x0004c228 0x00000000 0x00000001
l2-l3	0x00000000 0x0021d38 0x00000000 0x00000000
l4-l5	0x00000000 0x00000001 0x00000000 0xff36e7c
l6-l7	0x00000000 0xffbfe80 0x00000000 0x0017584
y	0x00000000 0x00000000
ect	0x00000000 0x00000000
pc	0x00000000 0x001a638: GetDiscount+0x8 1d [%fp + 68], %l0
npc	0x00000000 0x001a63c: GetDiscount+0xc 1d [%l0 + 4], %l0
f0f1	+0.000000000000000e+00
f2f3	+0.000000000000000e+00
f4f5	+0.000000000000000e+00
f6f7	-NaN
f8f9	-NaN
f10f11	-NaN
f12f13	-NaN
f14f15	-NaN

- 「ウィンドウ (Window)」 > 「デバッグ (Debugging)」 > 「メモリー (Memory)」 を選択して「メモリー (Memory)」 ウィンドウを開きます。ここでは、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。

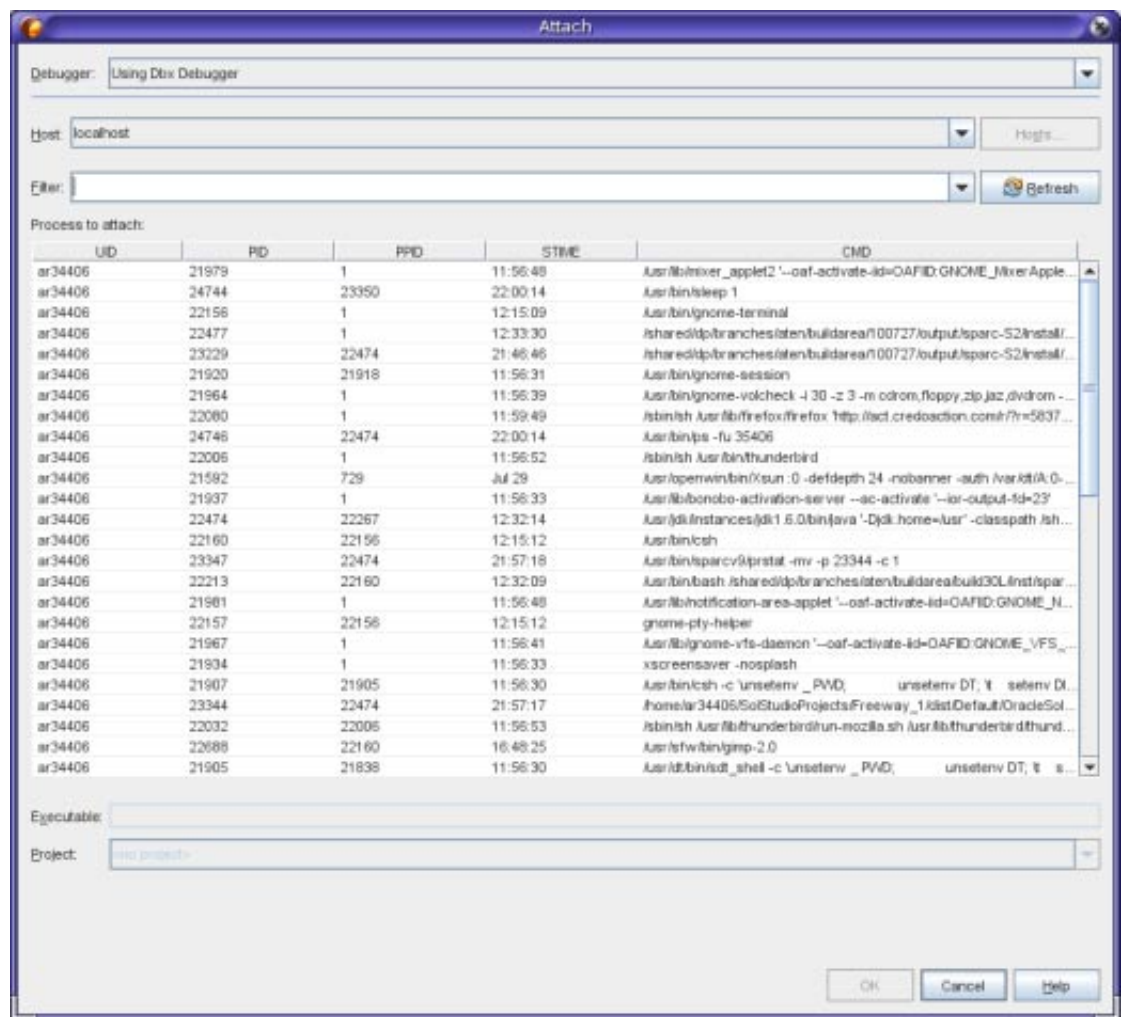
0x00017b38: main	:	0x9de3be98	0xf027a044	0xf227a048	0x210000b0
0x00017b48: main+0x0010:		0xa01421d0	0x2300006e	0xa2146158	0x90140000
0x00017b58: main+0x0020:		0x4000508e	0x52144000	0x21000057	0xa0142350
0x00017b68: main+0x0030:		0x40005090	0x52144000	0x21000057	0xa0142350
0x00017b78: main+0x0040:		0x4000508c	0x52144000	0x7ffffd74	0x01000000
0x00017b88: main+0x0050:		0xa0103fff	0xe027bf68	0xa007bff3	0x400050b5
0x00017b98: main+0x0060:		0x90140000	0xa007bff4	0xa207bff3	0x90140000
0x00017ba8: main+0x0070:		0x4000508c	0x52144000	0x7ffffd0c	0x01000000
0x00017bb8: main+0x0080:		0x210000b0	0xa01421d0	0x2300006e	0xa2146175
0x00017bc8: main+0x0090:		0x50140000	0x52144000	0x40005070	0x01000000
0x00017bd8: main+0x00a0:		0x210000b0	0xa0142238	0xa207bff4	0x90140000
0x00017be8: main+0x00b0:		0x52144000	0x400002cf	0x01000000	0xa007bf74
0x00017bf8: main+0x00c0:		0xa207bff4	0x90140000	0x52144000	0x400050b7
0x00017c08: main+0x00d0:		0x01000000	0xa007bf74	0x90140000	0x7ffffe25
0x00017c18: main+0x00e0:		0x01000000	0xd027bf70	0xe007bf70	0xe027bf68
0x00017c28: main+0x00f0:		0xa007bf74	0x90140000	0x40005094	0x01000000
0x00017c38: main+0x0100:		0xe007bf68	0x80a43fff	0x12800030	0x01000000
0x00017c48: main+0x0110:		0x210000b0	0xa01421d0	0x2300006e	0xa21461b5
0x00017c58: main+0x0120:		0x90140000	0x52144000	0x4000504c	0x01000000
0x00017c68: main+0x0130:		0xd027bf6c	0xe007bf6c	0xa207bff4	0x90140000

Address: main Length: 30 Format: Hexadecimal (4 bytes)

## 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 (File)」 > 「新規プロジェクト (New Project)」 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 (Samples)」 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 (Next)」 をクリックして、「完了 (Finish)」 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 (Run)」 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。
5. 「デバッグ (Debug)」 > 「デバッガを接続 (Attach Debugger)」 を選択します。

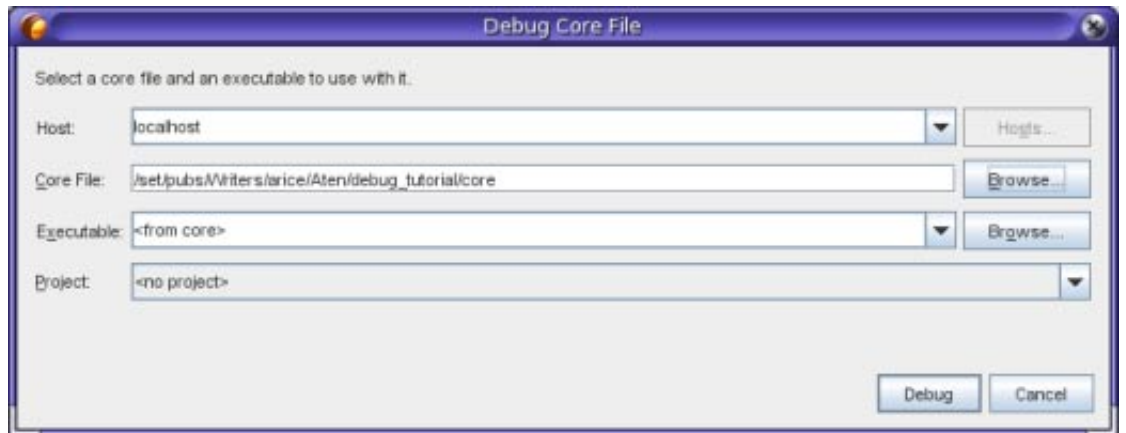


6. 「接続 (Attach)」 ダイアログボックスで、「フィルタ (Filter)」 フィールドに Freeway と入力して、プロセスのリストをフィルタします。
7. フィルタしたリストから Freeway プロセスを選択します。
8. 「OK」 をクリックします。
9. デバッグセッションが開始し、Freeway プロセスの実行がデバッガが接続されたポイントで一時的に停止します。

## 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 (Debug)」 > 「コアファイルのデバッグ (Debug core file)」 を選択します。
2. 「コアファイル (Core File)」 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 (Select Core File)」 ダイアログボックスで「参照 (Browse)」 をクリックしてコアファイルにナビゲートします。



3. 実行可能ファイルで指定したコアファイルにデバッガを接続できない場合、「実行可能ファイルを選択 (Select Executable)」 ダイアログボックスが開き、実行可能ファイルを指定できます。この場合、「実行可能ファイル (Executable)」 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 (Browse)」 ボタンをクリックして「実行可能ファイル (Executable)」 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 (Project)」 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 (Debug)」 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 12.2 dbxtool チュートリアル](#)を参照してください。

Copyright ©2010 このソフトウェアおよび関連ドキュメントの使用と開示は、ライセンス契約の制約条件に従うものとし、知的財産に関する法律により保護されています。ライセンス契約で明示的に許諾されている場合もしくは法律によって認められている場合を除き、形式、手段に関係なく、いかなる部分も使用、複写、複製、翻訳、放送、修正、ライセンス供与、送信、配布、発表、実行、公開または表示することはできません。このソフトウェアのリバース・エンジニアリング、逆アSEMBル、逆コンパイルは互換性のために法律によって規定されている場合を除き、禁止されています。ここに記載された情報は予告なしに変更される場合があります。また、誤りが無いことの保証はいたしかねます。誤りを見つけた場合は、オラクル社までご連絡ください。このソフトウェアまたは関連ドキュメントを、米国政府機関もしくは米国政府機関に代わってこのソフトウェアまたは関連ドキュメントをライセンスされた者に提供する場合は、次の通知が適用されます。

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are “commercial computer software” or “commercial technical data” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

このソフトウェアもしくはハードウェアは様々な情報管理アプリケーションでの一般的な使用のために開発されたものです。このソフトウェアもしくはハードウェアは、危険が伴うアプリケーション（人的傷害を発生させる可能性があるアプリケーションを含む）への用途を目的として開発されていません。このソフトウェアもしくはハードウェアを危険が伴うアプリケーションで使用する際、安全に使用するために、適切な安全装置、バックアップ、冗長性（redundancy）、その他の対策を講じることは使用者の責任となります。このソフトウェアもしくはハードウェアを危険が伴うアプリケーションで使用したことにより起因して損害が発生しても、オラクル社およびその関連会社は一切の責任を負いかねます。

Oracle と Java は Oracle Corporation およびその関連企業の登録商標です。その他の名称は、それぞれの所有者の商標または登録商標です。

AMD、Opteron、AMD ロゴ、AMD Opteron ロゴは、Advanced Micro Devices, Inc. の商標または登録商標です。Intel、Intel Xeon は、Intel Corporation の商標または登録商標です。すべての SPARC の商標はライセンスをもとに使用し、SPARC International, Inc. の商標または登録商標です。UNIX は X/Open Company, Ltd. からライセンスされている登録商標です。

このソフトウェアまたはハードウェア、そしてドキュメントは、第三者のコンテンツ、製品、サービスへのアクセス、あるいはそれらに関する情報を提供することがあります。オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスに関して一切の責任を負わず、いかなる保証もいたしません。オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスへのアクセスまたは使用によって損失、費用、あるいは損害が発生しても一切の責任を負いかねます。

821-2501

Oracle Corporation 500 Oracle Parkway, Redwood City, CA 94065 U.S.A.

