

# Oracle® Solaris Studio 12.3 IDE ク イックスタートチュートリアル

2011年12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「既存のソースからのプロジェクトの作成」
- 10 ページの「バイナリファイルからのプロジェクトの作成」
- 11 ページの「Oracle Database プロジェクトの作成」
- 11 ページの「リモート開発の実行」
- 14 ページの「アプリケーションのパッケージ作成」
- 15 ページの「ソースファイルの編集」
- 23 ページの「ソースファイルのナビゲーション」
- 28 ページの「プロジェクトでのメモリアクセス検査の実行」
- 29 ページの「ブレークポイントの作成」
- 31 ページの「プロジェクトのデバッグ」
- 34 ページの「機械命令レベルでのデバッグ」
- 35 ページの「実行中のプログラムを接続してデバッグ」
- 36 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 IDE 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およびメイクファイルを持つプロジェクトを作成したり、既存のバイナリファイルからプロジェクトを作成したりすることができます。

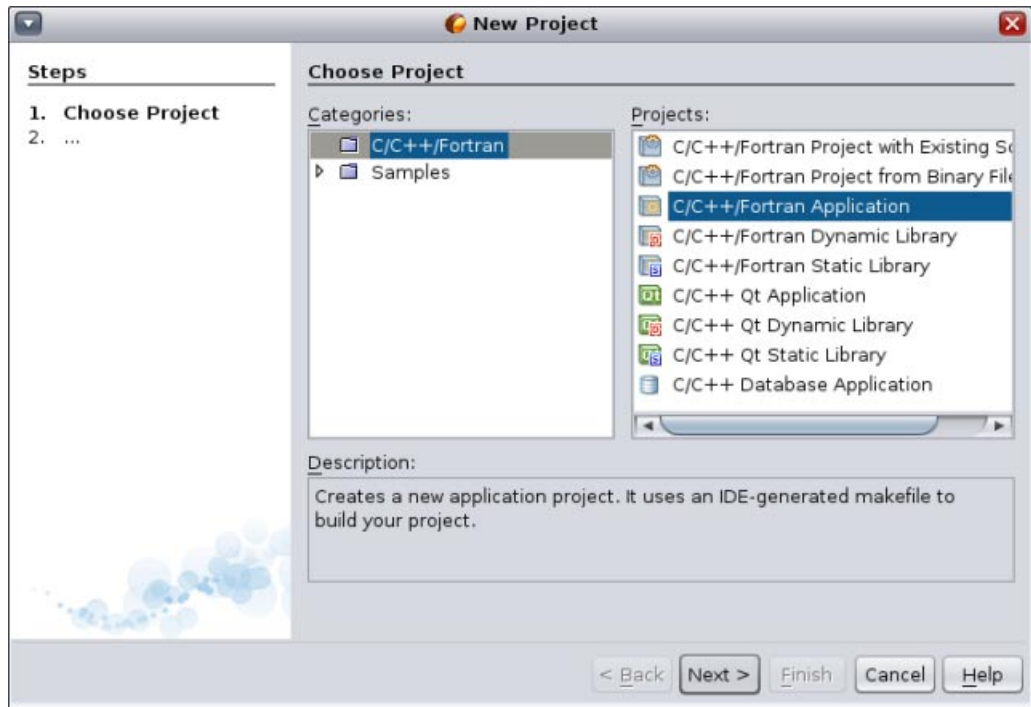
プロジェクトの構築、実行、およびデバッグは、IDE を起動したローカルホスト上、および Solaris オペレーティングシステムもしくは Linux オペレーティングシステムを実行しているリモートホストで行うことができます。

C/C++/Fortran アプリケーション、動的ライブラリ、静的ライブラリ、または Oracle Database 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」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。

既存のソースからのプロジェクトは、メイクファイルを使用して構築されます。

## アプリケーションプロジェクトの作成

1. 「ファイル」 > 「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。

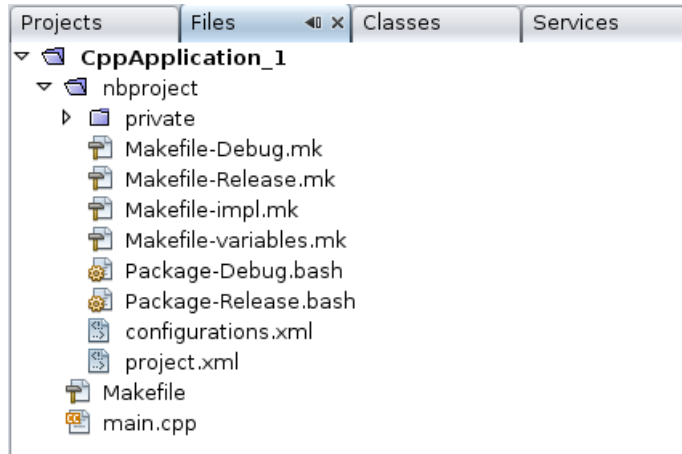
プロジェクトが作成され、いくつかの論理フォルダが作成されます。論理フォルダはディレクトリではありません。ファイルを整理する手段であり、ファイルがディスク上に物理的に保存される場所を示すものではありません。論理フォルダに追加されるファイルは自動的にプロジェクトの一部となり、プロジェクトを構築する際にコンパイルされます。

「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。

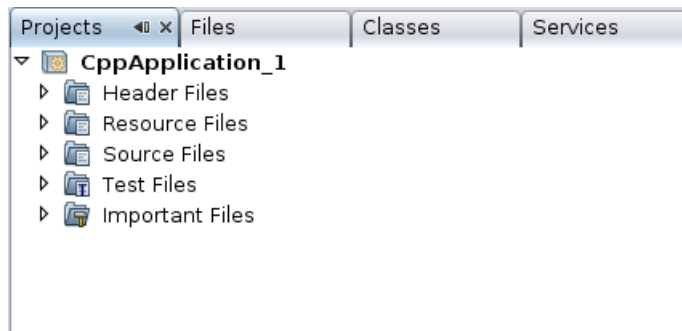
## プロジェクトの論理ビューと物理ビューの切り換え

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」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。



## 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」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。

## 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」フォルダを右クリックして、「新規」>「Cmain 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。

newmain.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」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、任意の種類のファイルを追加できます。

---

注-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。

---

## 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」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。

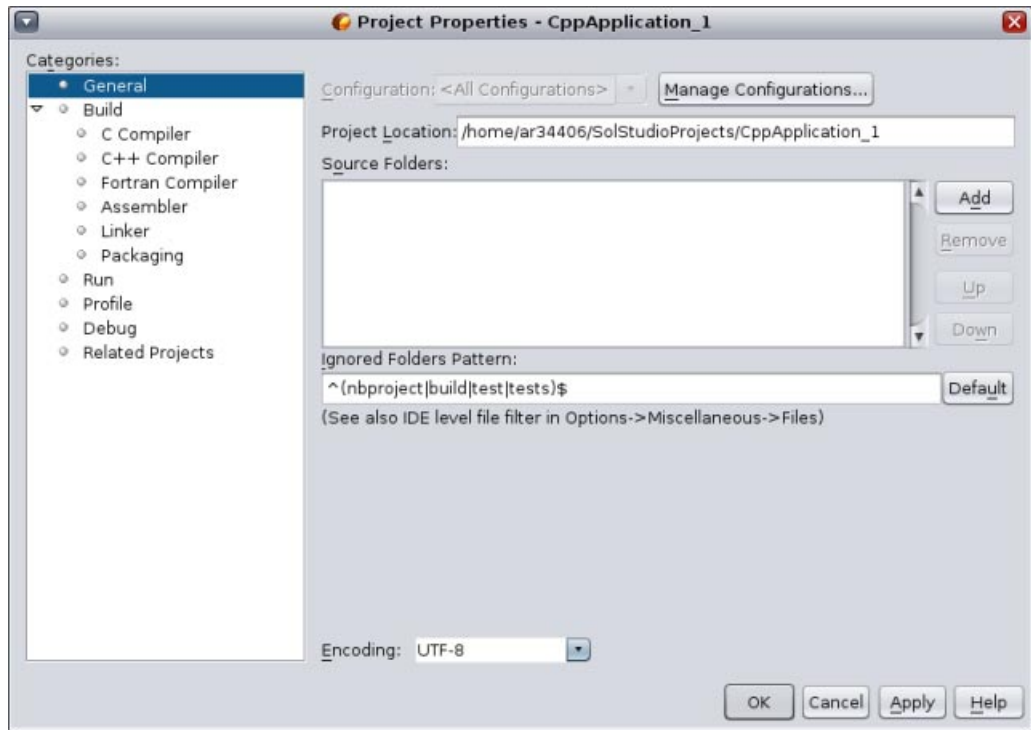
既存の項目の追加に「新規」メニュー項目を使用しないでください。「名前と場所」パネルから、ファイルがすでに存在していることが通知されます。

## 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」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」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」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。

## 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。

- 「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。
- 「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を上書きできることを確認します。
- 「プロジェクトのプロパティ」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」ウィンドウのプロジェクトノードを右クリックすると、IDEでは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」ウィンドウでボールドテキストで表示されます。

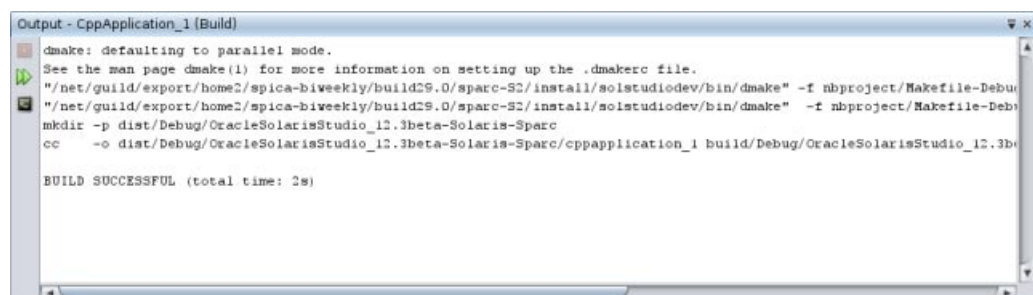
IDEで主プロジェクトを変更するには、次の手順に従います。

- 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」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。

- 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。



```
Output - CppApplication_1 [Build]
dmake: defaulting to parallel mode.
See the man page dmake(1) for more information on setting up the .dmake.rc file.
"/net/guild/export/home2/spica-biveekly/build29.0/sparc-S2/install/solstudiodev/bin/dmake" -f nbproject/Makefile-Debug
"/net/guild/export/home2/spica-biveekly/build29.0/sparc-S2/install/solstudiodev/bin/dmake" -f nbproject/Makefile-Debug
mkdir -p dist/Debug/OracleSolarisStudio_12.3beta-Solaris-Sparc
cc -o dist/Debug/OracleSolarisStudio_12.3beta-Solaris-Sparc/cppapplication_1 build/Debug/OracleSolarisStudio_12.3beta-Solaris-Sparc/cppapplication_1.o
BUILD SUCCESSFUL (total time: 2s)
```

- 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 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。
- 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。

---

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。

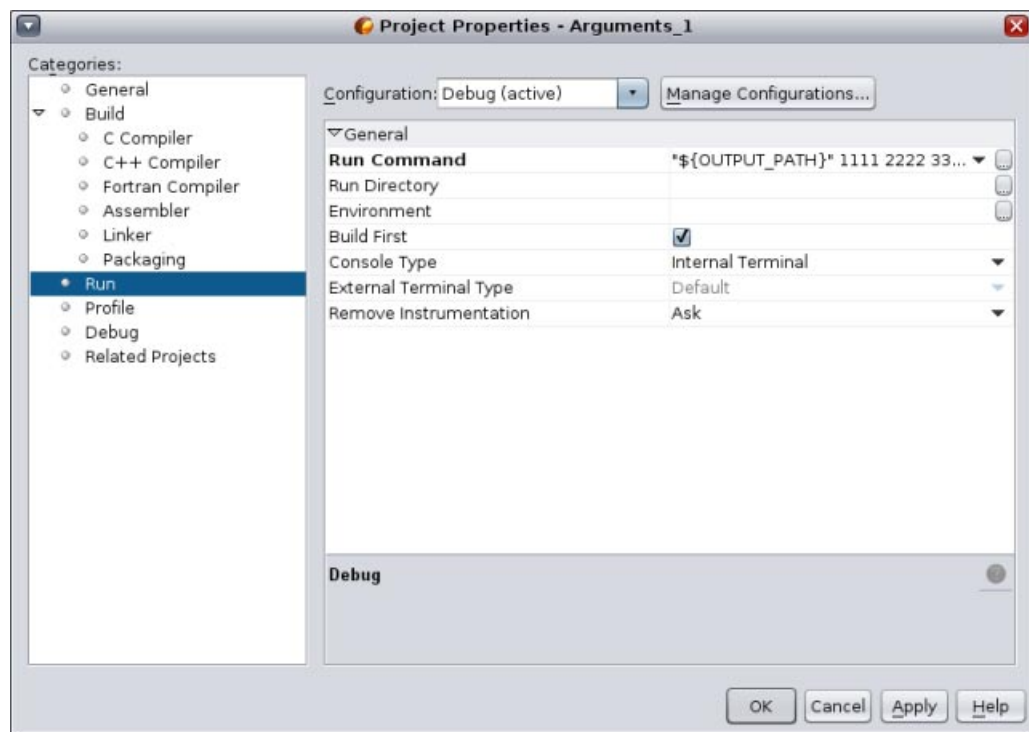
---

## 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」>「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」を選択します。
6. 「プロジェクトのプロパティ」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。



---

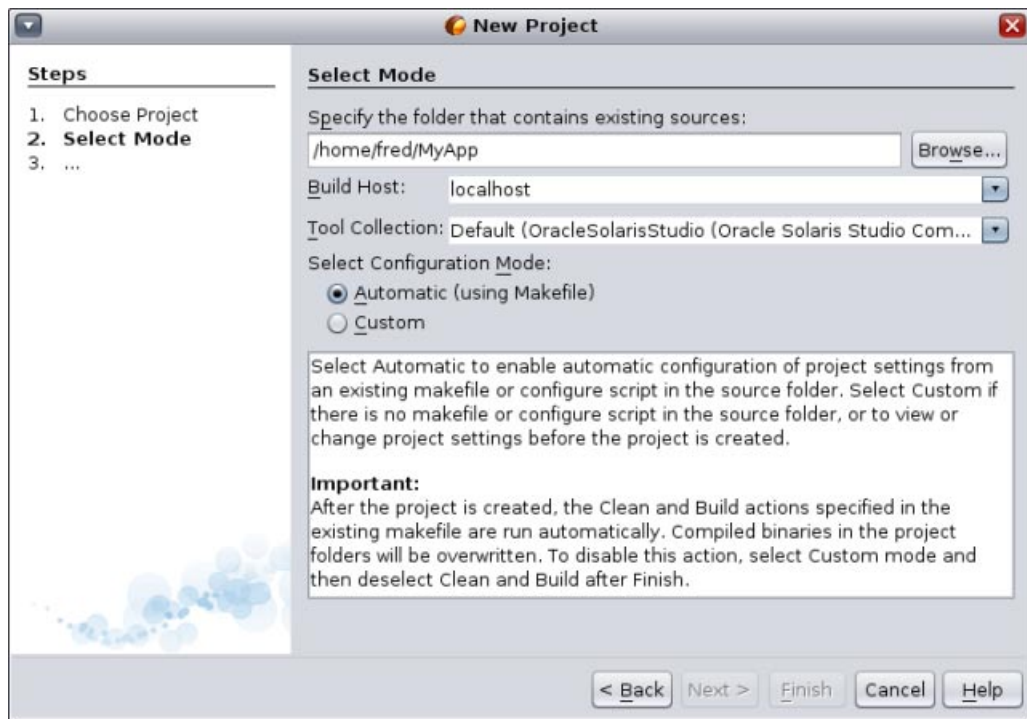
ヒント-プロジェクトを実行すると「モニターの実行」タブが開き、アプリケーションの動作を監視するプロファイルツールが表示されます。プロファイルツールは「プロジェクトのプロパティ」ダイアログボックスでオフにできます。

---

## 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設定されます。

プロジェクトが作成され、「プロジェクト」ウィンドウが開きます。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」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。

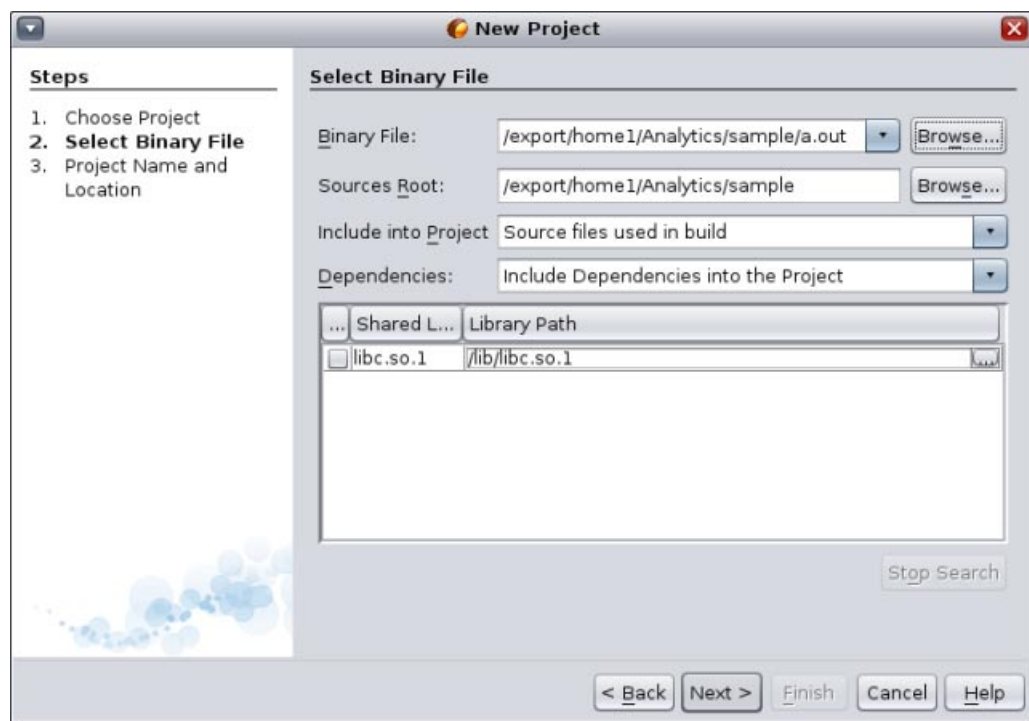
- プロジェクトのプロジェクトノードを右クリックして、「生成物を削除して構築」を選択します。

## バイナリファイルからのプロジェクトの作成

バイナリファイルからの C/C++/Fortran プロジェクトを使用すると、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。

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せん。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。

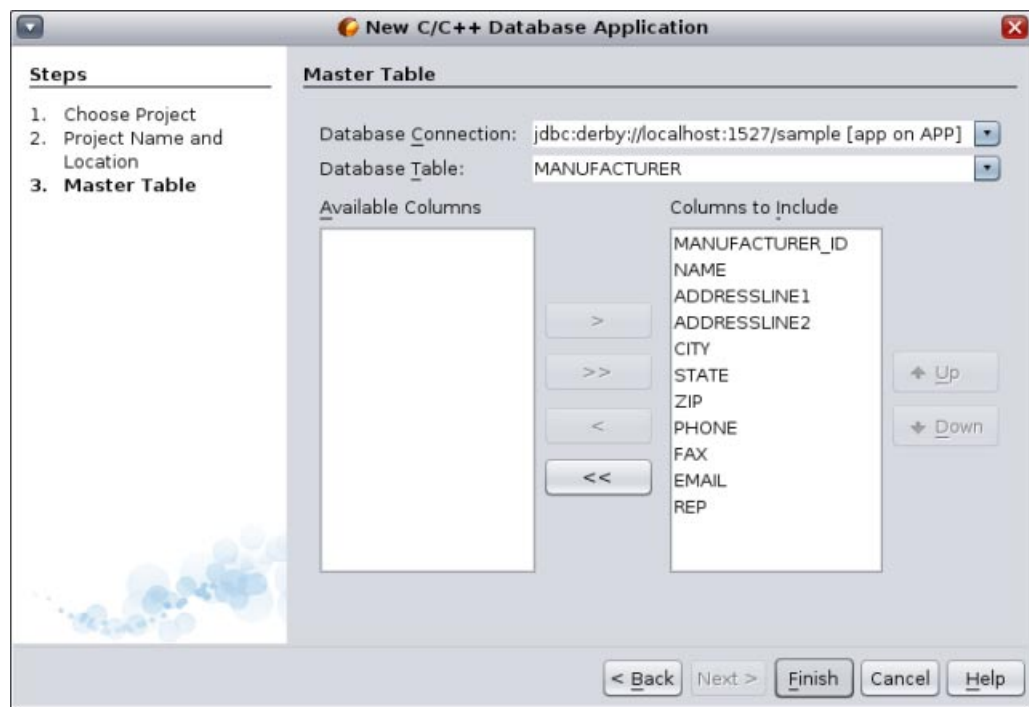


5. 「次へ」をクリックします。
6. 「プロジェクトの名前と場所」ページでは、プロジェクトの名前と場所を選択できます。「完了」をクリックします。

## 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。「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。



5. 「完了」をクリックします。

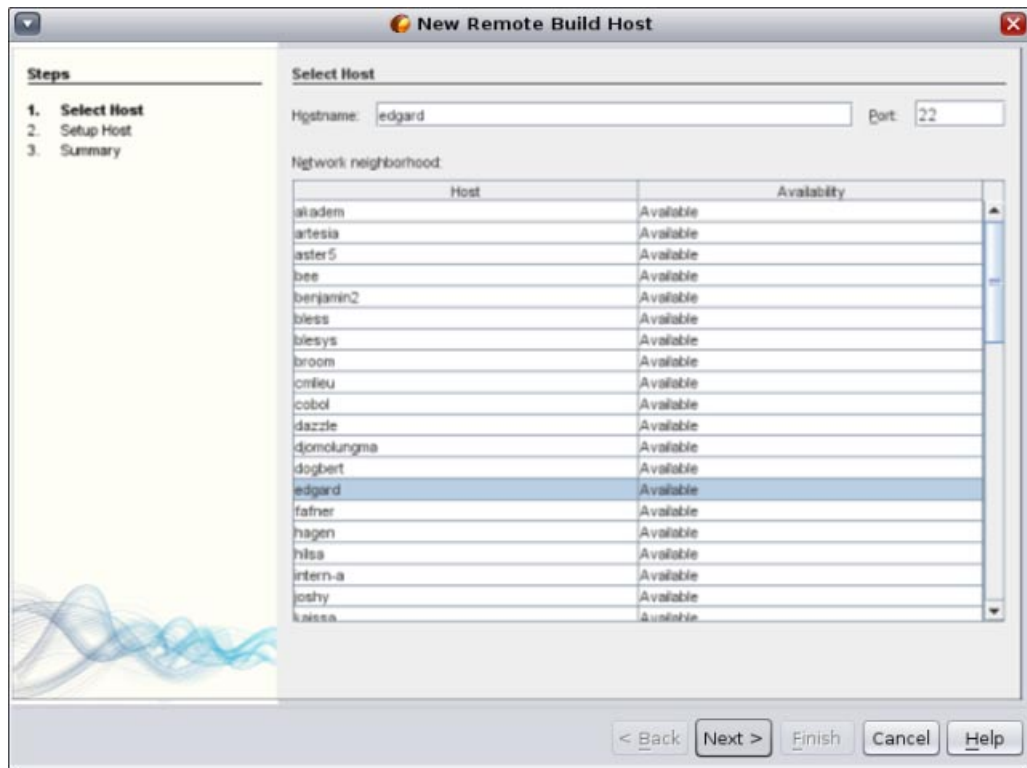
## リモート開発の実行

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。

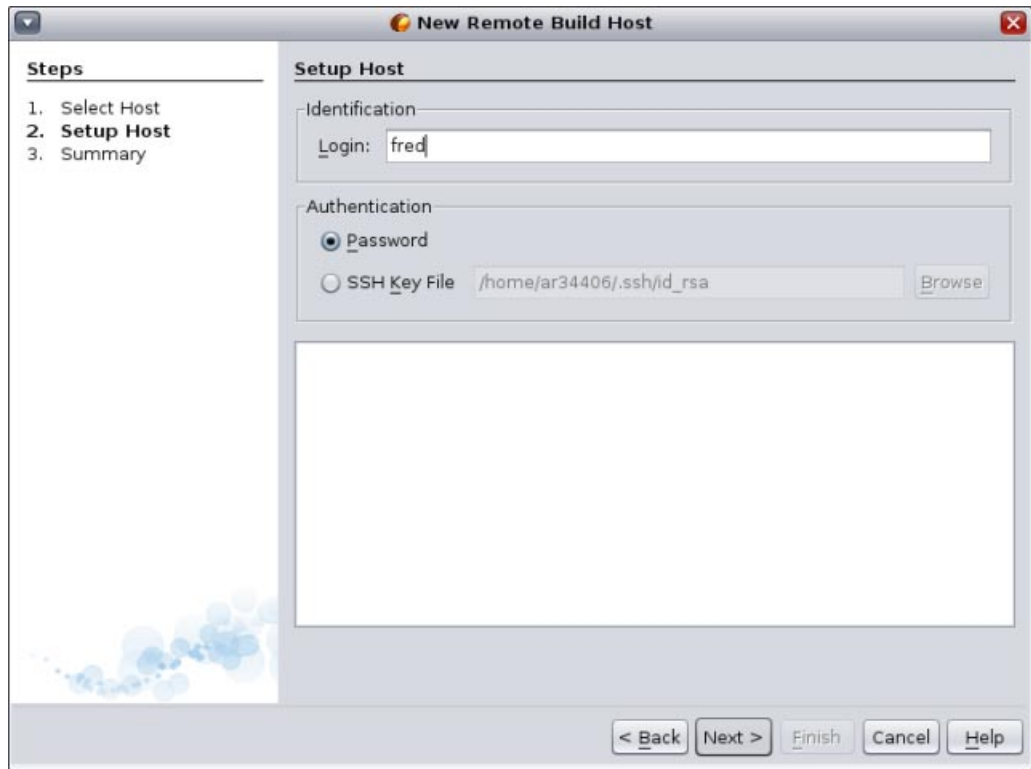
「オプション」ダイアログボックスの「構築ツール」タブで、リモート開発ホストを構成できます。リモートホストを追加するには、次の手順に従います。

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。



- 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」ページが表示されます。「完了」をクリックします。
- 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。
- 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」を選択します。「ホストのプロパティ」ダイアログボックスで目的のプロパティを設定します。
- 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。

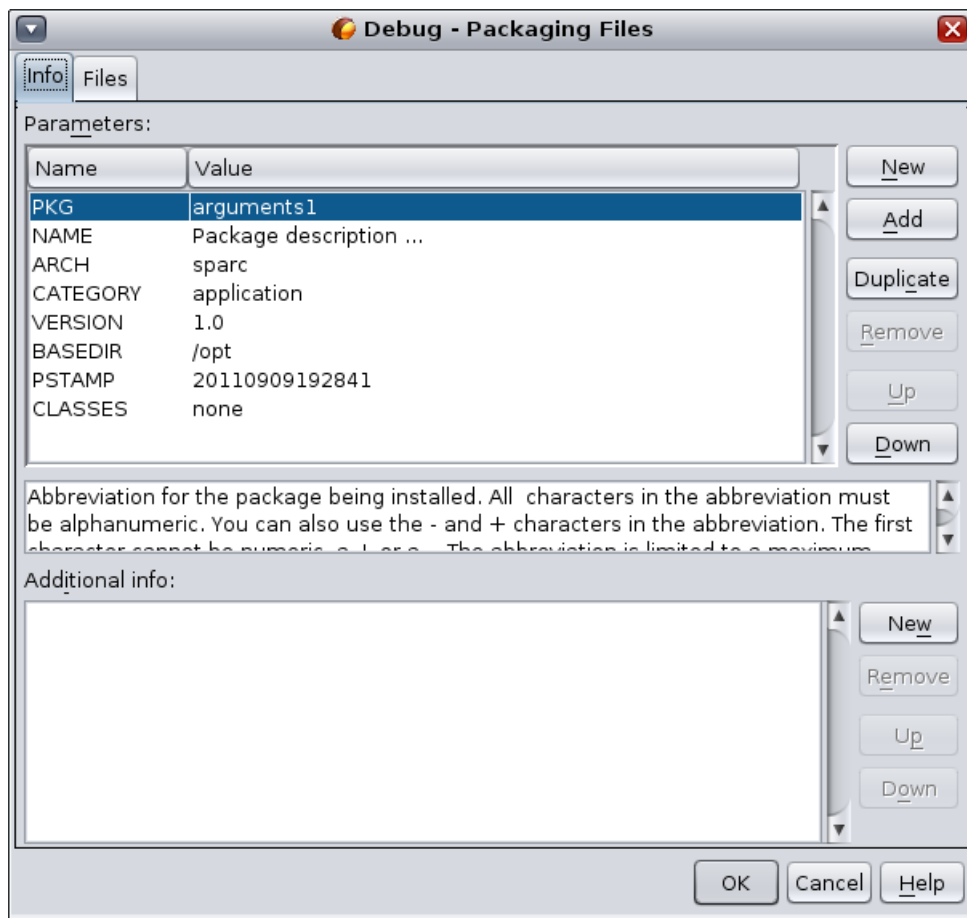
プロジェクトを作成するとき、デフォルトの構築ホストがプロジェクトの構築ホストとして選択されます。「プロジェクトのプロパティ」ダイアログボックスの「構築」パネルで、プロジェクトの構築ホストを変更できます。実行可能ファイルまたはコアファイルをデバッグするときに、構築ホストを指定することもできます。

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。

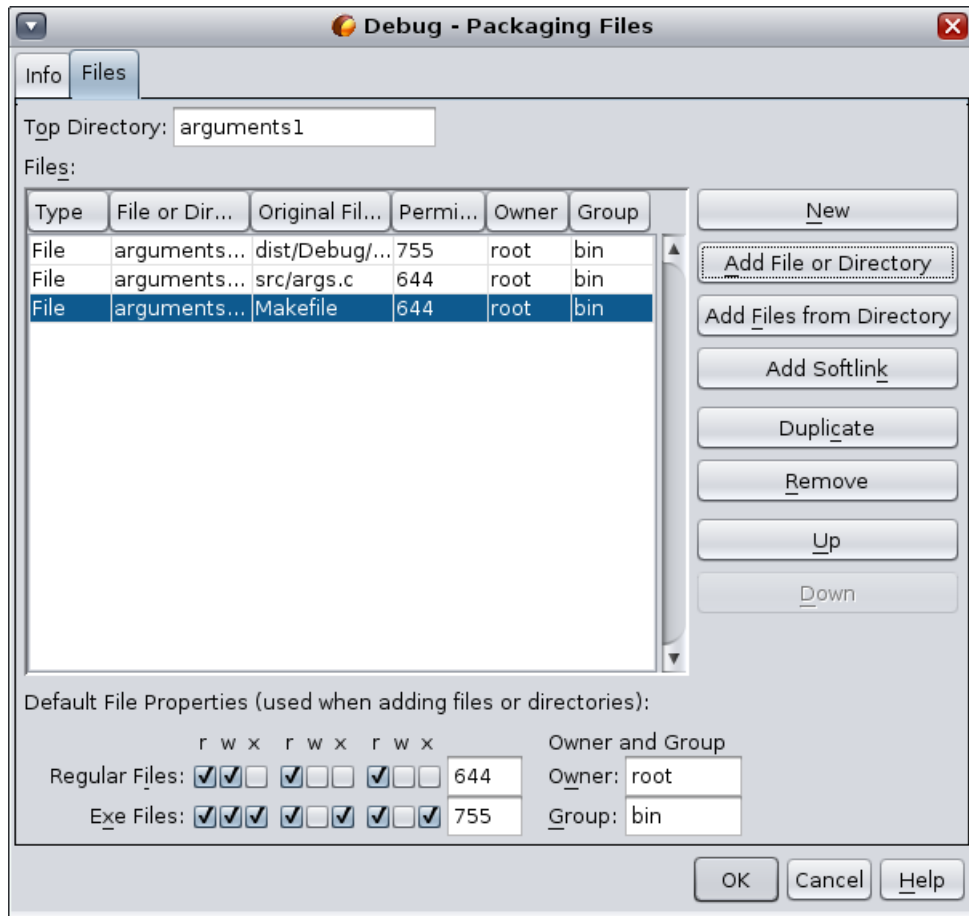
# アプリケーションのパッケージ作成

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」を選択します。
2. 「プロジェクトのプロパティ」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。

## ソースファイルの編集

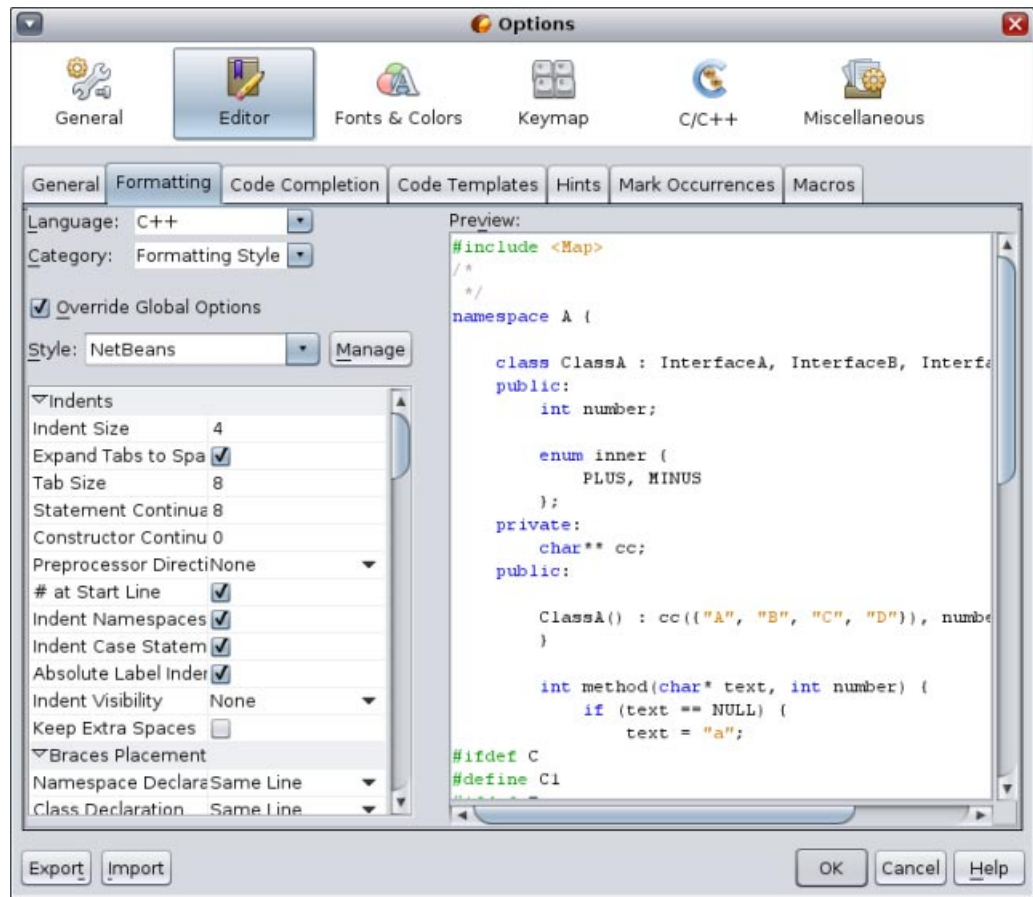
Oracle Solaris Studio IDE には高度な編集機能およびコード支援機能があり、ソースコードの表示と変更役に立ちます。これらの機能を確認するため、Quote プロジェクトを使用します。

1. 「ファイル」>「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設定スタイルを設定できます。

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(マイナスイ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。

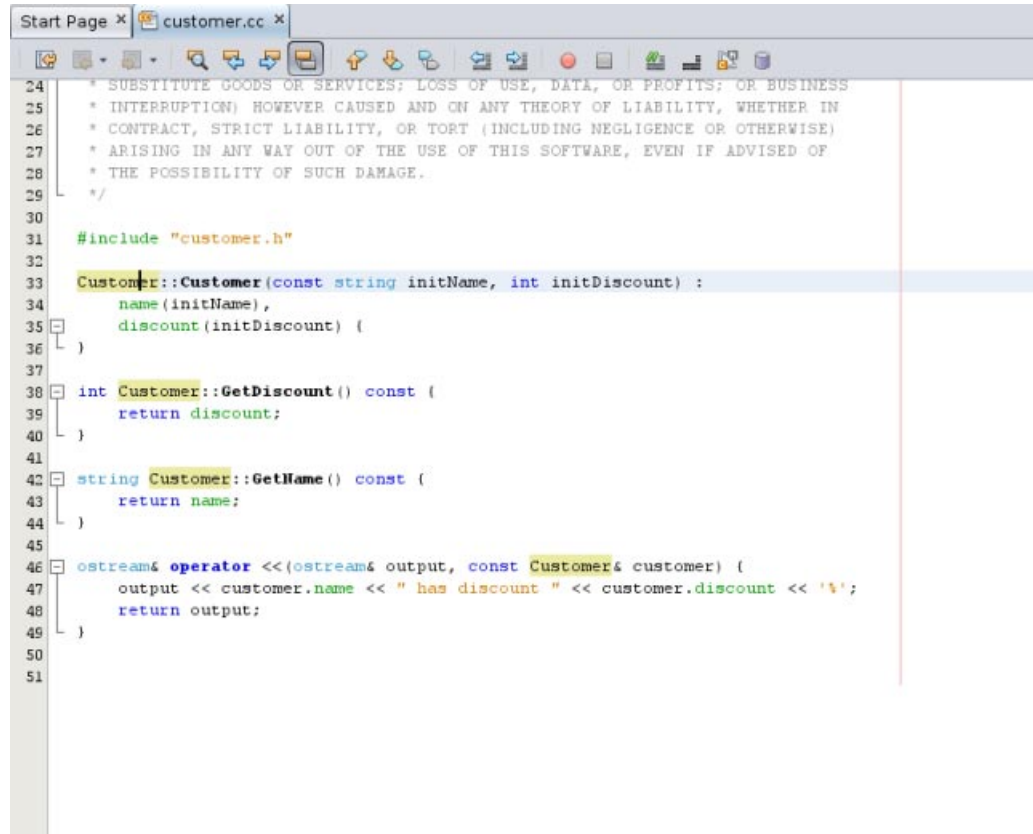
## 意味上の強調表示の使用

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。

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。



- 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。
- Customer クラスの出現箇所をクリックします。
- 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。



```
24  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
25  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
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
```

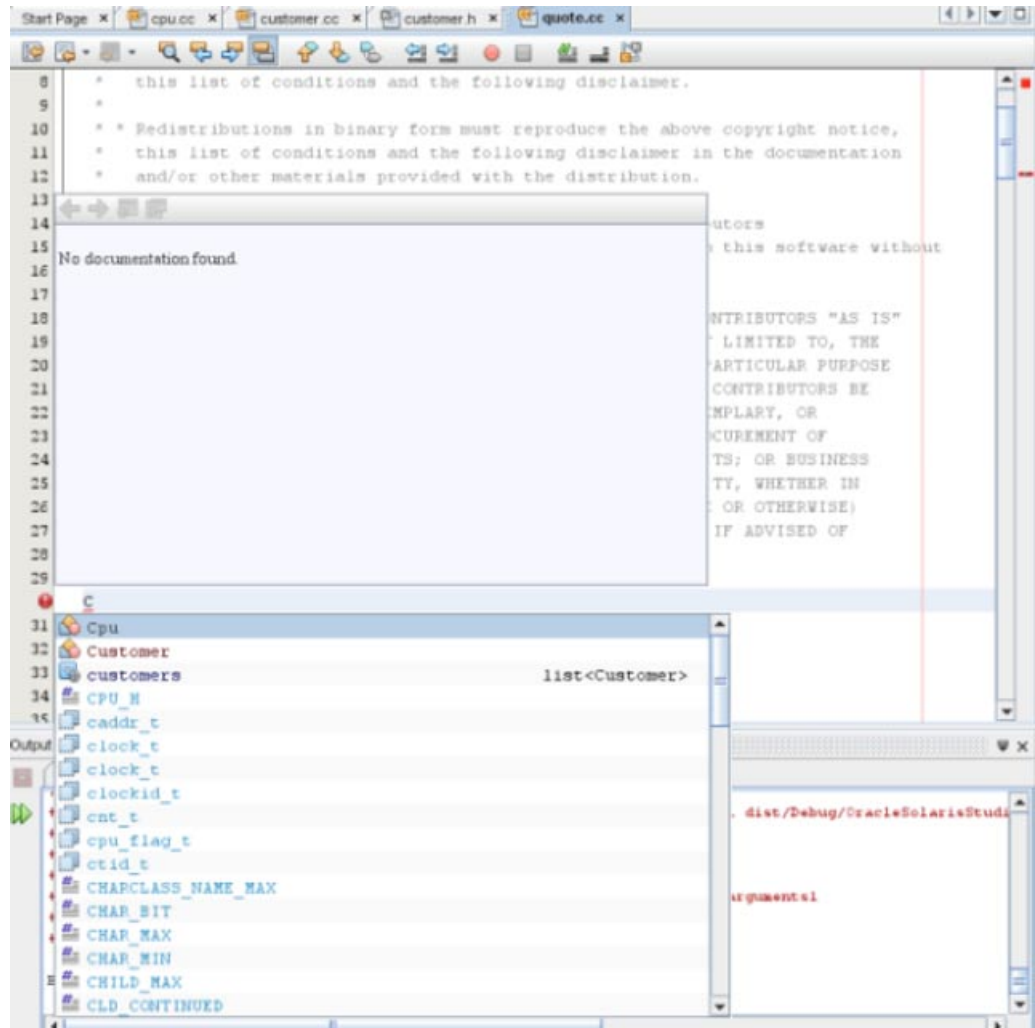
- customer.h ファイルで、クラスフィールドがボールドで強調表示されていることを確認します。

```
Start Page x customer.cc x customer.h x
25  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
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
53
```

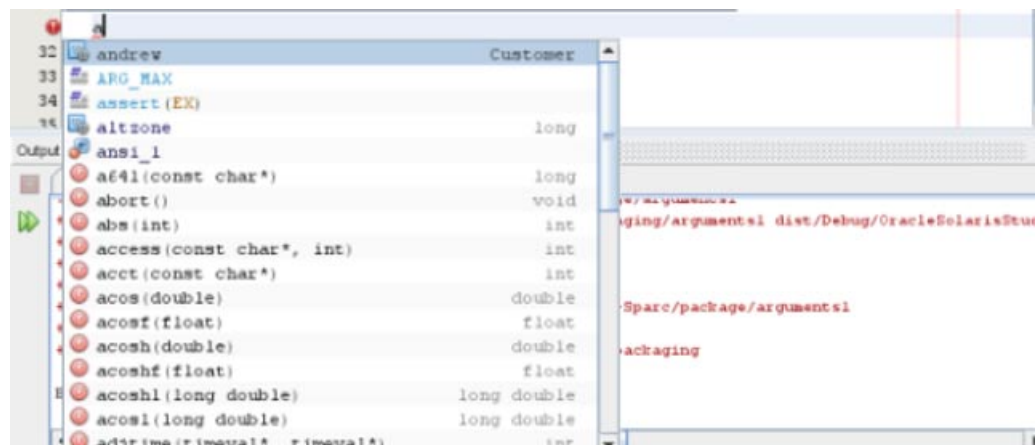
## コード補完の使用

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、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。



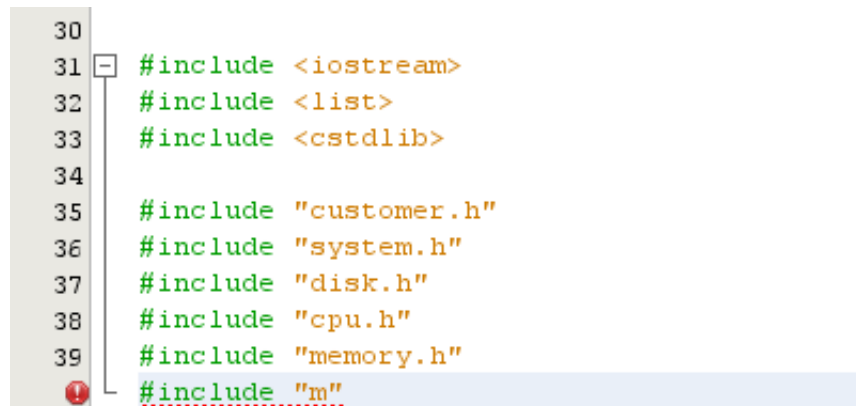
```
30 Customer andrew;  
andrew.  
32 #inc discount int  
33 #inc name string  
34 #inc GetDiscount() int  
35 #inc GetName() string
```

- 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。

- Quote\_1 プロジェクトの quote.cc ファイル内で、40 行目に #include "m と入力すると、エラーアイコンがマージンに表示されることがわかります。



```
30  
31 #include <iostream>  
32 #include <list>  
33 #include <cstdlib>  
34  
35 #include "customer.h"  
36 #include "system.h"  
37 #include "disk.h"  
38 #include "cpu.h"  
39 #include "memory.h"  
40 #include "m"
```

- 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。
- 追加した文を削除します。

## 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。

- 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。エディタによって、Doxygen で書式設定されたコメントが readNumberOf クラス用に挿入されます。

```

72 |         return -1;
73 |     }
74 | }
75 | /**
76 |  *
77 |  * @param item
78 |  * @param min
79 |  * @param max
80 |  * @return
81 |  */
82 | int readNumberOf(const char* item, int min, int max) {
83 |     cout << "Enter number of " << item << " (" << min << " <= N <= " << max << "): ";
84 | }

```

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。

```

75 | /**
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;

```

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。

```

161 |
162 | int amount = readNumberOf("CPUs", 1, 10);
163 |
164 | Cpu MyCpu(type,
165 |
166 | MySystem.AddModu
167 |
168 | response = readC
169 |
170 | switch (response
171 |     case 'Q':
172 |         return 2
173 |
174 |     case 'R':
175 |         type = D
176 |         break;
177 |
178 |     case 'S':
179 |     default :

```

**Parameter:**  
item Type of item

**Parameter:**  
min Least amount of item customer can order

**Parameter:**  
max Maximum amount of item customer can order

**Returns:**

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

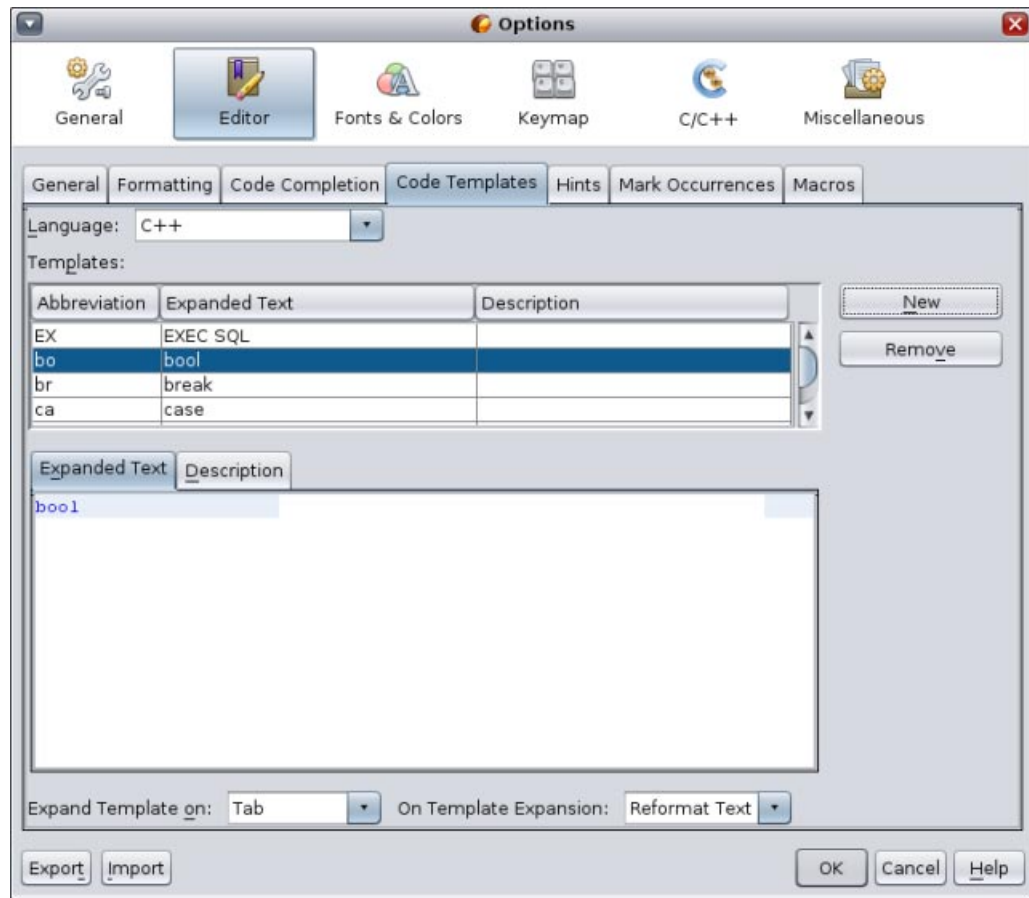
ソースエディタには、C、C++、および Fortran コードの共通スニペット用の、一連のカスタマイズ可能なコードテンプレートがあります。略語を入力して Tab キーを押すと、コードスニペット全体を生成できます。たとえば、Quote\_1 プロジェクトの quote.cc ファイルで、次のようになります。

- 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」 > 「オプション」を選択します。
- 「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。
- 「言語」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。

- 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。
- 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。
- invalid=0, success=1 と入力して、列挙法を補完します。
- 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。
- 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。

- 追加したコードを削除します。

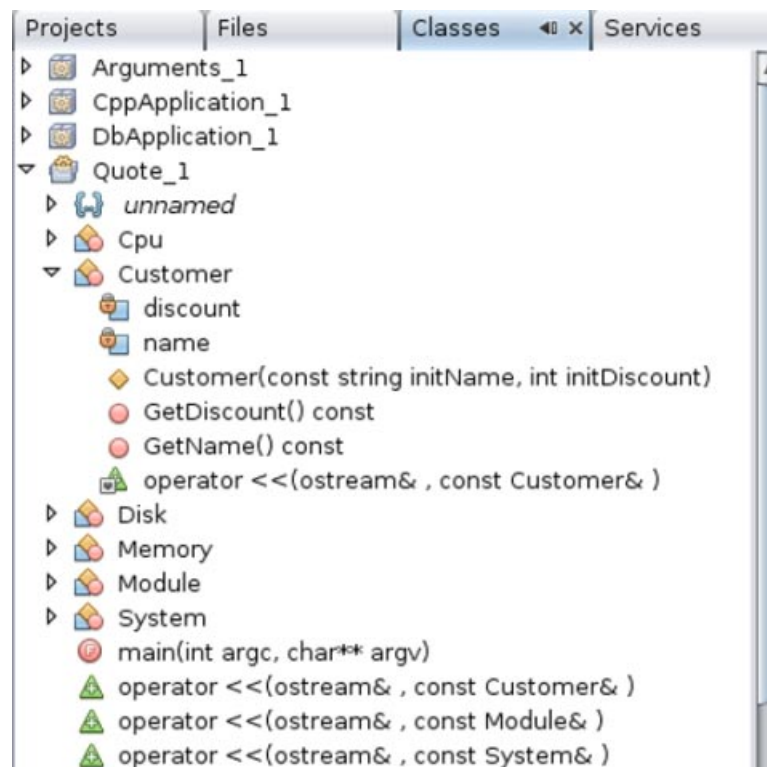
## ソースファイルのナビゲーション

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」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。

- 「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。
- 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。
- Customerクラスを展開します。

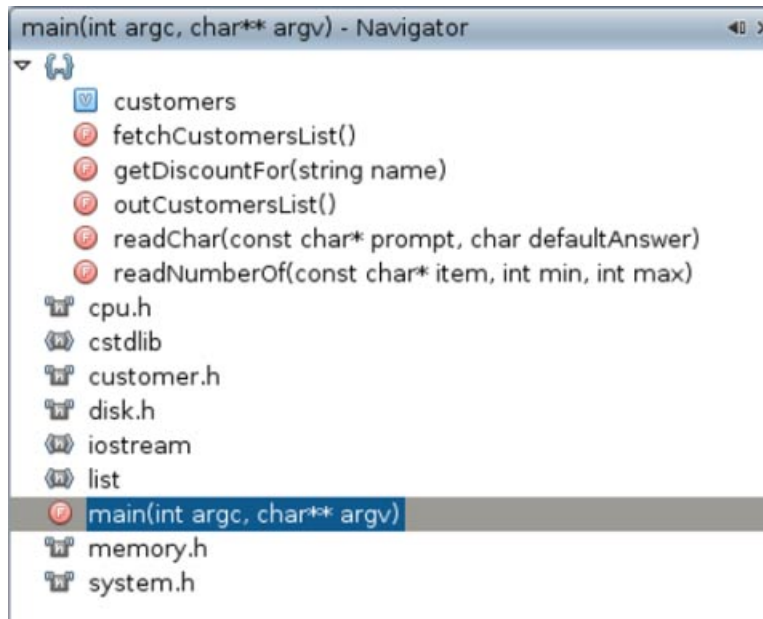


- name変数をダブルクリックしてcustomer.hヘッダーファイルを開きます。

### 「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」の順に選択して開きます。

- 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。
- 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。

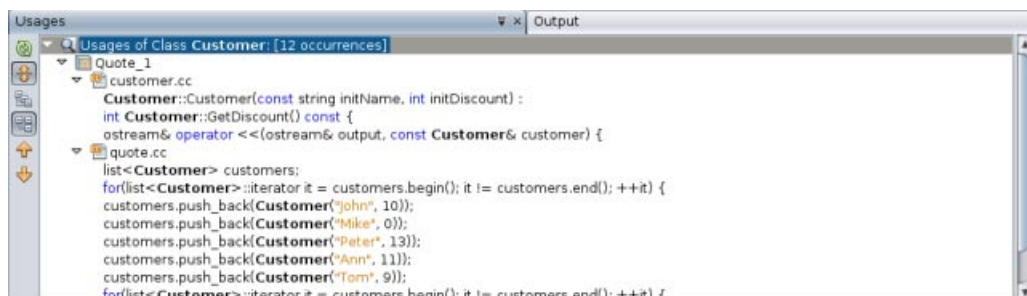


3. ファイルの要素にナビゲートするには、「ナビゲータ」ウィンドウで要素をダブルクリックして、エディタウィンドウのカーソルをその要素に移動させます。
4. 「ナビゲータ」ウィンドウ内を右クリックして、ウィンドウ内の要素のソート、項目のグループ化、フィルタのオプションを表示させます。
5. 「ナビゲータ」ウィンドウにあるアイコンを確認するには、「ヘルプ」>「ヘルプの目次」を選択して、IDE オンラインヘルプを開きます。ヘルプブラウザで、「検索」タブをクリックして、「検索」フィールドにナビゲータアイコンと入力します。

## 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」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。

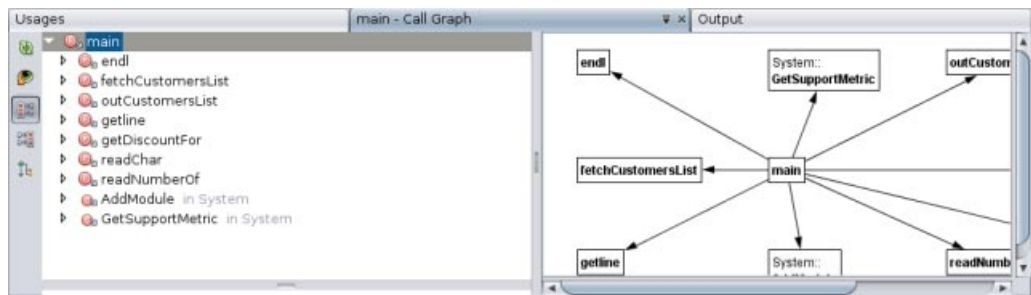


## コールグラフの使用

「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。

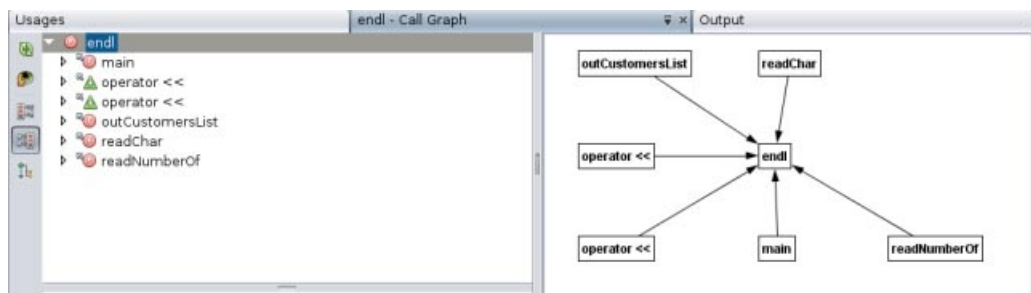


1. quote.cc ファイルで、メイン関数を右クリックして「コールグラフの表示」を選択します。
2. 「コールグラフ」ウィンドウが開き、ツリービューと、main 関数から呼び出されるすべての関数のグラフ表示が表示されます。

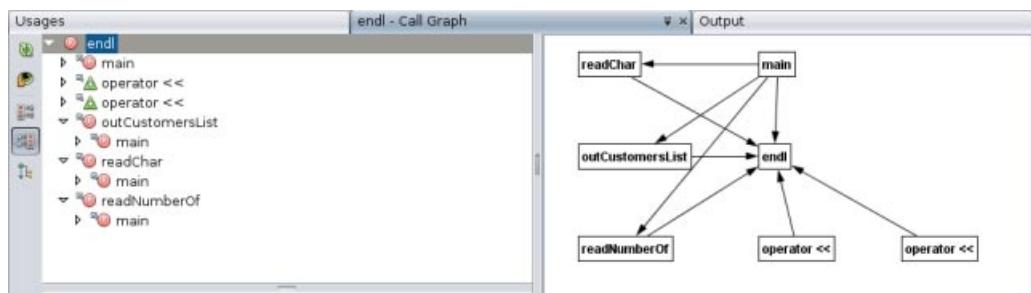


スクリーンショットに示す関数の一部が表示されない場合は、「コールグラフ」ウィンドウの左側の3番目のボタンをクリックして、この関数から呼び出される関数を表示します。

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。



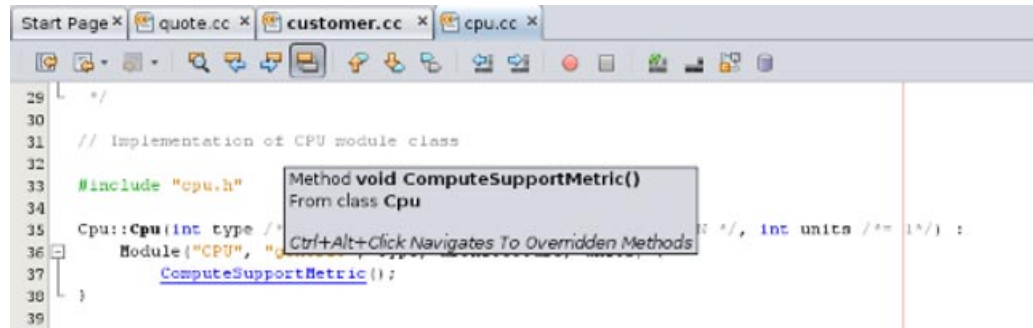
5. ツリー内のいくつかのノードを展開して、その他の関数を表示します。



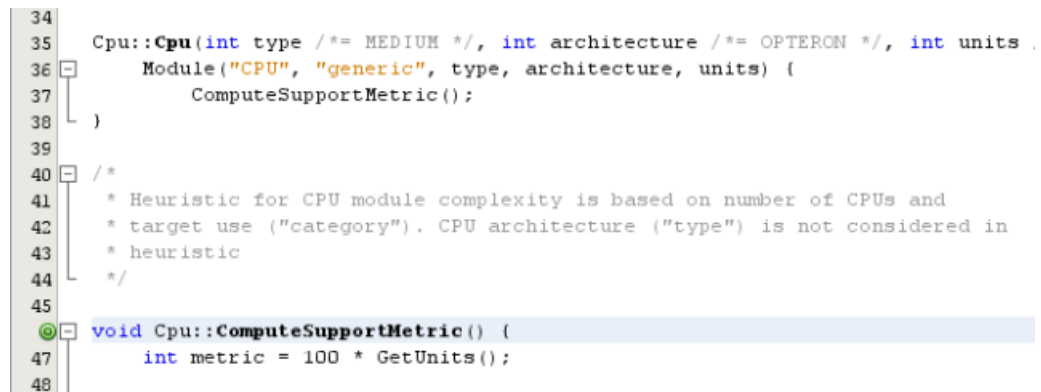
## ハイパーリンクの使用


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。

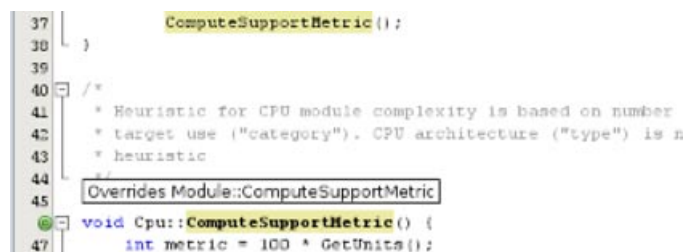
1. Quote\_1プロジェクトのcpu.ccファイルで、Ctrlを押しながら37行目にマウスオーバーします。ComputeSupportMetric関数が強調表示され、関数についての情報を示す注釈が表示されます。



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。



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マージンの緑の円  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。



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。

```

69 protected:
70     virtual void ComputeSupportMetric() = 0; //metric is defined in derived classes
71     Is Overridden
72     Cpu::ComputeSupportMetric
73     Disk::ComputeSupportMetric
74     Memory::ComputeSupportMetric
75     int category;
76     int units;
77

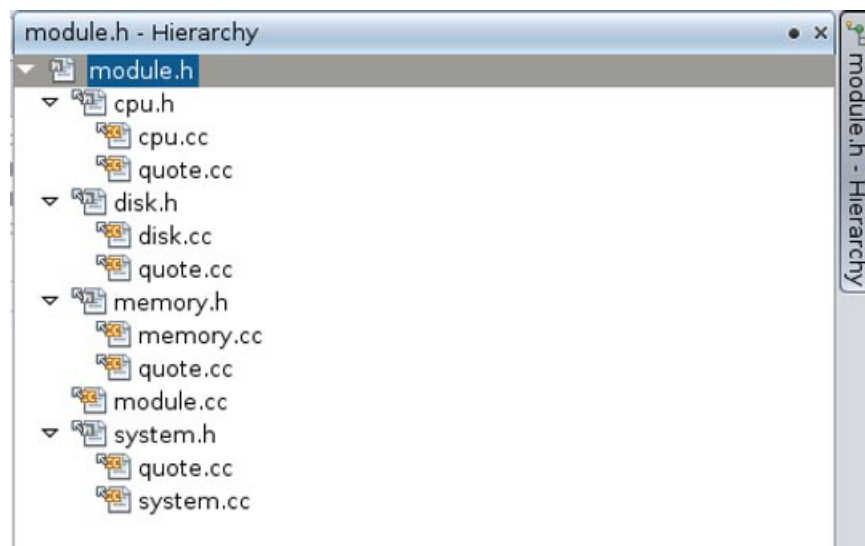
```

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。

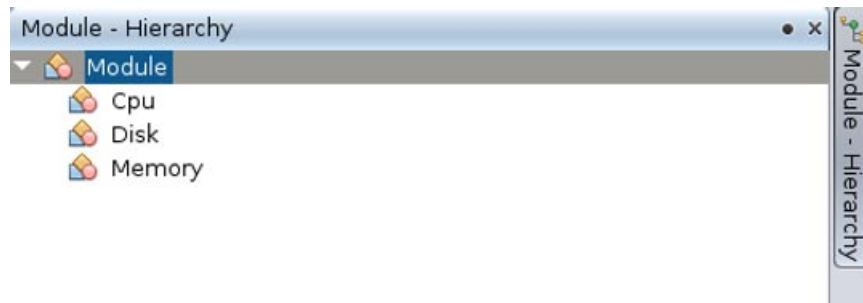
1. Quote\_1 プロジェクトで、ソースファイルに module.cc ファイルを開きます。
2. ファイルの #include "module.h" 行を右クリックして、「ナビゲート」>「インクルードの階層を表示」を選択します。
3. デフォルトで、「階層」ウィンドウにはヘッダーファイルに直接インクルードされるファイルのプレーンリストが表示されます。ウィンドウ下部の右端のボタンをクリックして、表示をツリービューに変更します。右から2番目のボタンをクリックして、インクルードまたはインクルードされるすべてのファイルに表示を変更します。ツリービューのノードを展開して、ヘッダーファイルをインク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

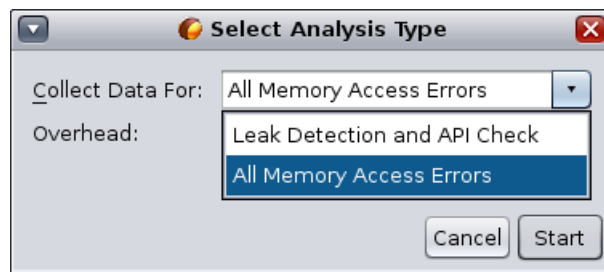


## プロジェクトでのメモリアクセス検査の実行

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指摘することによって、これらのエラーを簡単に見つけることができます。

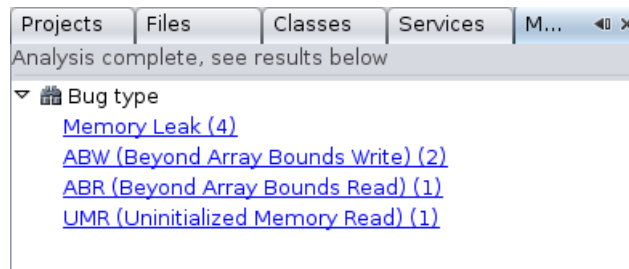
メモリー解析ツールはプログラムの実行中にメモリアクセスエラーを動的に検出して報告するため、コードの一部が実行時に実行されていない場合、その部分のエラーは報告されません。

1. まだ行っていない場合、「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」を選択します。「プロジェクトのプロパティ」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. 次に、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. memorychecks プロジェクトがメインプロジェクトとして設定されていることを確認します。
  - b. 「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「メモリアクセスエラー」を見つけます。
  - c. 「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。

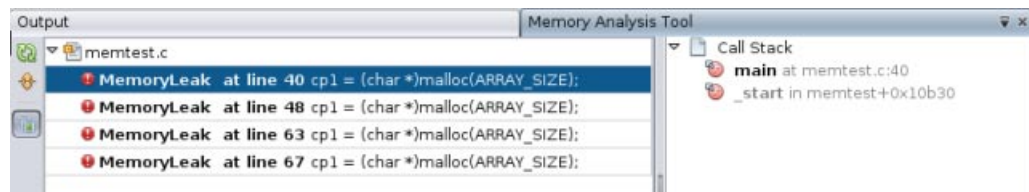


「オーバーヘッド」フィールドにはシステムにかかる負荷を示す「高」または「中」が表示されます。オーバーヘッドが高い場合、システム上で実行中のほかのプログラムのパフォーマンスに影響が出ることがありますが、データの競合とデッドロックの両方が検出されるのはこの場合です。

- d. 「起動」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。エラーの種類のアとの括弧内に、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。

## 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 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。

```
166
167     Cpu MyCpu(type, 0, amount); // Create CPU module object
168
169     MySystem.AddModule(&MyCpu); // Add CPU Module to system specification
170
171     response = readChar("Enter disk module type: S for single disks, R for RAID; Q - es...");
172
173     switch (response) {
174     case 'Q':
175         return 2; //premature user requested termination
176
```

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。

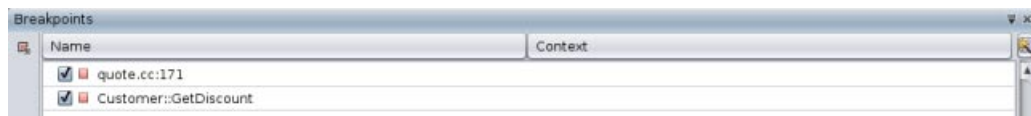


### 関数ブレークポイントの作成

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` 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. 関数ブレイクポイントが設定され、「ブレイクポイント」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。デバッガが開始してアプリケーションが実行し、「変数」および「デバッガコンソール」ウィンドウが開きます。

```

Variables | Debugger Console | Output
Reading libm.so.2
Reading libc.so.1
(dbx) cd "/home/ar34406/SolStudioProjects/Quote_1"
(dbx) runargs
(dbx) intercept -set -unhandled, -unexpected
(dbx) ### dbxenv run_ptty /dev/pts/6
(dbx) ### dbxenv rto_error_log_file_name /dev/null
(dbx) stop at "/home/ar34406/SolStudioProjects/Quote_1/quote.cc":171
(dbx) stop in Customer::GetDiscount
(dbx) run
Running: quote_1
(process id 662)
Reading libc_psr.so.1

```

- 「ウィンドウ」>「デバッグ」>「セッション」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。

Name	Process ID	Process State	Host
quote_1	6336	Running	localhost

## アプリケーションの状態の検査

- 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。
- 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」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。

Name	Context
<input checked="" type="checkbox"/> quote.cc:171	[6336] quote_1
<input checked="" type="checkbox"/> Customer::GetDiscount()const	[6336] quote_1

- 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。

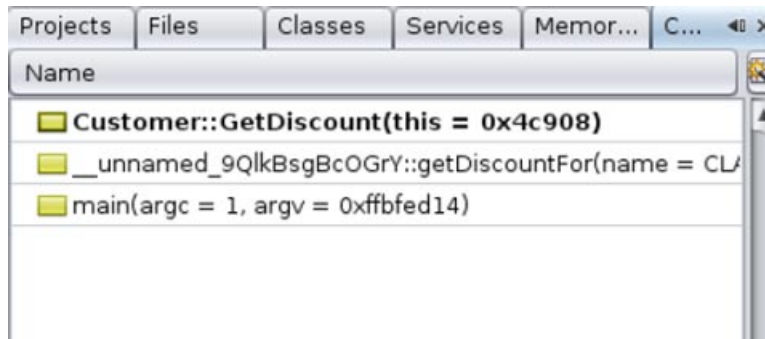
```

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

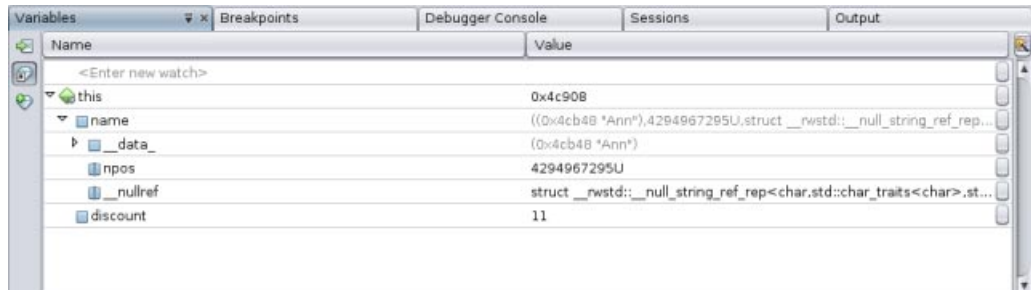
```

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

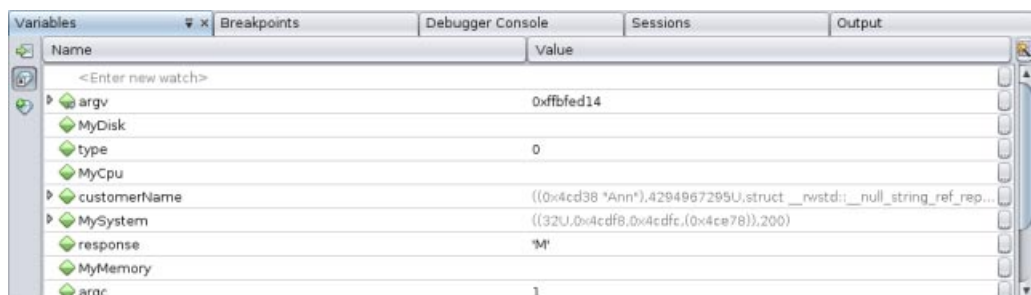





- 「変数」ウィンドウをクリックし、1つの変数が表示されていることに注意します。ノードをクリックして構造を展開します。



- 「続行」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。
- 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。



- 「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。
- 「継続」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行います。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。

# 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」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。

```
1 34     name (initName),
2 35     discount (initDiscount) {
3 0x0001ac48: Customer:      : save    %sp, -96, %sp
4 0x0001ac4c: Customer+0x0004: st     %i0, [%fp + 68]
5 0x0001ac50: Customer+0x0008: st     %i1, [%fp + 72]
6 0x0001ac54: Customer+0x000c: st     %i2, [%fp + 76]
7 0x0001ac58: Customer+0x0010: ld     [%fp + 68], %i0
8 0x0001ac5c: Customer+0x0014: ld     [%fp + 72], %i1
9 0x0001ac60: Customer+0x0018: or     %i0, %i0, %i0
10 0x0001ac64: Customer+0x001c: call   basic_string [PLT] ! 0x2c59c
11 0x0001ac68: Customer+0x0020: or     %i1, %i0, %i1
12 0x0001ac6c: Customer+0x0024: ld     [%fp + 76], %i1
13 0x0001ac70: Customer+0x0028: ld     [%fp + 68], %i0
14 0x0001ac74: Customer+0x002c: ba     Customer+0x48 ! 0x1ac90
15 0x0001ac78: Customer+0x0030: st     %i1, [%i0 + 4]
16 0x0001ac7c: Customer+0x0034: ld     [%fp + 68], %i0
17 0x0001ac80: Customer+0x0038: call   "basic_string [PLT] ! 0x2c5c4
18 0x0001ac84: Customer+0x003c: or     %i0, %i0, %i0
19 0x0001ac88: Customer+0x0040: call   ex_rethrow_q [PLT] ! 0x2c410
20 0x0001ac8c: Customer+0x0044: nop
21 0x0001ac90: Customer+0x0048: ret
22 0x0001ac94: Customer+0x004c: restore
23
24 36     }
25
26 37
27 38     int Customer::GetDiscount() const {
28 39     return discount;
29
30 0x0001aca8: GetDiscount:  : save    %sp, -104, %sp
31 0x0001acac: GetDiscount+0x0004: st     %i0, [%fp + 68]
32 0x0001acb0: GetDiscount+0x0008: ld     [%fp + 68], %i0
33 0x0001acb4: GetDiscount+0x000c: ld     [%i0 + 4], %i0
34 0x0001acb8: GetDiscount+0x0010: st     %i0, [%fp - 4]
35 0x0001acbc: GetDiscount+0x0014: ld     [%fp - 4], %i0
36 0x0001acc0: GetDiscount+0x0018: or     %i0, %i0, %i0
```

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。

Name	Value
g0-g1	0x00000000 0x00000000 0x00000000 0x00103298
g2-g3	0x00000000 0x00000000 0x00000000 0x00000000
g4-g5	0x00000000 0x00000000 0x00000000 0x00000000
g6-g7	0x00000000 0x00000000 0x00000000 0xff382a00
o0-o1	0x00000000 0x00000000 0x00000000 0x00000001
o2-o3	0x00000000 0x00000001 0x00000000 0x00000000
o4-o5	0x00000000 0x00000000 0x00000000 0x0000000e
o6-o7	0x00000000 0xffbfea20 0x00000000 0xff25eb4
i0-i1	0x00000000 0x0004c908 0x00000000 0x0004c908
i2-i3	0x00000000 0x00000000 0x00000000 0x00000000
i4-i5	0x00000000 0x0004cb48 0x00000000 0x00000001
i6-i7	0x00000000 0xf363718 0x00000000 0x00000020

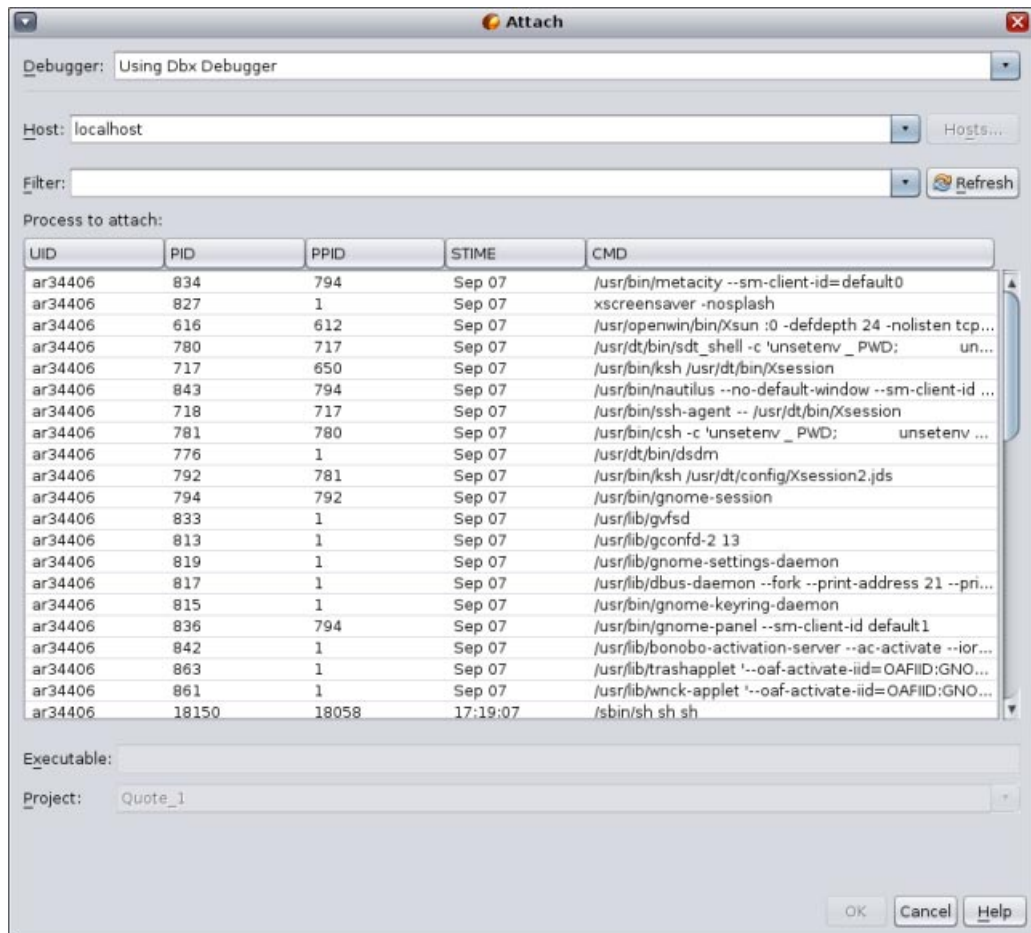
5. 「ウィンドウ」>「デバッグ」>「メモリー」を選択して「メモリー」ウィンドウを開きます。ここには、現在プロジェクトで使用されているメモリーの内容が表示されます。ウィンドウの下部で、参照するメモリーアドレスの指定、メモリー参照の長さの変更、メモリー情報の形式の変更を行えます。




Variables	Breakpoints	Debugger Console	Sessions	Memory	Registers	Output
0x00018120: main :		0x9de3be9000000000	0xf027a04400000000	0xf227a04800000000	0x210000b200000000	
0x00018130: main+0x0010:		0xa01420b000000000	0x2300007000000000	0xa214e00000000000	0x9014000000000000	
0x00018140: main+0x0020:		0x400050c000000000	0x9214400000000000	0x2100005800000000	0xa01423b800000000	
0x00018150: main+0x0030:		0x400050c200000000	0x9214000000000000	0x2100005800000000	0xa01423b800000000	
0x00018160: main+0x0040:		0x400050be00000000	0x9214000000000000	0x7ffffd2c00000000	0x1000000000000000	
0x00018170: main+0x0050:		0xa0103fff00000000	0xe027bfff80000000	0xa007bfff30000000	0x400050e400000000	
0x00018180: main+0x0060:		0x9014000000000000	0xa007bfff40000000	0xa207bfff30000000	0x9014000000000000	
0x00018190: main+0x0070:		0x400050eb00000000	0x9214400000000000	0x7ffffcc400000000	0x1000000000000000	
0x000181a0: main+0x0080:		0x210000b200000000	0xa01420b000000000	0x2300007000000000	0xa214e01d00000000	
0x000181b0: main+0x0090:		0x9014000000000000	0x9214400000000000	0x400050a200000000	0x1000000000000000	
0x000181c0: main+0x00a0:		0x210000b200000000	0xa014211000000000	0xa207bfff40000000	0x9014000000000000	
0x000181d0: main+0x00b0:		0x9214400000000000	0x400002f300000000	0x1000000000000000	0x250000b200000000	

## 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。

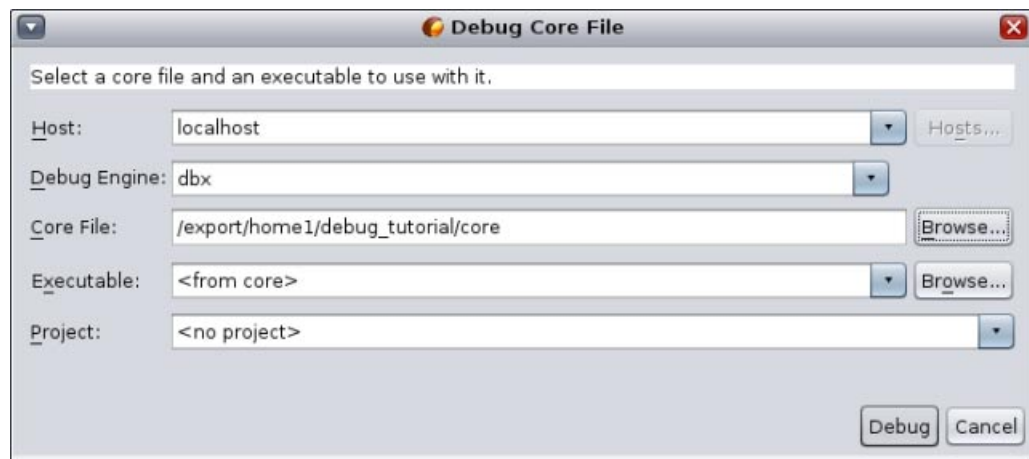


6. 「接続」ダイアログボックスで、「フィルタ」フィールドに Freeway と入力して、プロセスのリストをフィルタします。
7. フィルタしたリストから Freeway プロセスを選択します。
8. 「OK」をクリックします。
9. デバッグセッションが開始し、Freeway プロセスの実行がデバッガが接続されたポイントで一時停止します。
10. 「継続」をクリック  して、現在デバッガの制御下で実行中の Freeway の実行を継続します。「一時停止」  をクリックすると、Freeway の実行が一時停止し、変数や呼び出しスタックなどを検査できます。
11. 「継続」をふたたびクリックして、「デバッグセッションの完了」  をクリックします。デバッグセッションが終了しますが、Freeway プロセスは実行を継続します。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に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 12.3: dbxtool チュートリアル](#)を参照してください。

Copyright ©2011 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、逆アセンブル、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 END USERS:

Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are “commercial computer software” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

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および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。

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。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。UNIXは、The Open Groupの登録商標です。

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。

E26460

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