# Red Hat Enterprise Linux 6

## 開発者ガイド

Red Hat Enterprise Linux 6 のアプリケーション開発ツールの概要

Last Updated: 2017-10-12

# Red Hat Enterprise Linux 6 開発者ガイド

Red Hat Enterprise Linux 6 のアプリケーション開発ツールの概要

Jacquelynn East
Red Hat Customer Content Services
jeast@redhat.com

Don Domingo
Red Hat Customer Content Services
ddomingo@redhat.com

Robert Krátký
Red Hat Customer Content Services
rkratky@redhat.com

## 法律上の通知



## 概要

本書では、Red Hat Enterprise Linux 6 をアプリケーション開発に最適なエンタープライズプラットフォームにするさまざまな機能とユーティリティーについて説明します。本ガイドでは、エントツーエンドの総合開発環境 (IDE) としての Eclipse にフォーカスしますが、Eclipse 外のコマンドラインツールや他のユーティリティーについても説明しています。

# 目次

# 第1章 ECLIPSE の開発環境

Eclipse は、開発プロセスの各フェーズにツールを提供する強力な開発環境です。容易に使用できるようにするために完全に設定済の単独ユーザーインターフェースに統合されています。そして各種方法で拡張を可能にするプラグ可能なアーキテクチャーを特徴としています。

Eclipse は、一元管理された環境にさまざまな異種ツールを統合してリッチな開発環境を構築します。たとえば、Valgrind プラグインでは、プログラマーは Eclipse ユーザーインターフェースで (通常はコマンドラインで実行される) メモリープロファイリングを実行できます。この機能は、Eclipse 専用のものではありません。

Eclipse はグラフィカルなアプリケーションなので、コマンドラインインターフェースになじめず難しいと感じていた開発者には歓迎されるものとなります。また、Eclipse のビルトインの **ヘルプ** システムは、各統合機能およびツールの幅広いドキュメントを提供します。これにより、新しい開発者が使い方に慣れるまでに必要な時間が大幅に短縮されます。

従来型の (つまり、ほとんどコマンドラインベースの) Linux ツールスイート (**gcc** や **gdb** など) と Eclipse は別個の 2 つのアプローチをプログラミングに提供します。ほとんどの従来型 Linux ツールは、Eclipse ベースのものよりはるかに柔軟性があり繊細で (トータルでは) 強力なものです。一方でこれらの従来型 Linux ツールは熟達するのが難しく、ほとんどのプログラマーやプロジェクトが必要とする以上の機能を提供します。対照的に Eclipse は、これらの機能の一部を犠牲にして統合型環境を優先します。これは、単一のグラフィカルなインターフェースでアクセス可能なツールを好むユーザーには適しています。

## 1.1. ECLIPSE プロジェクトの開始

以下のコマンドで Eclipse をインストールします。

```
# yum install eclipse
```

インストール後に Eclipse を開始するには、手動で **/usr/bin/eclipse** を実行するか、インストール時に作成されたシステムメニューを使用します。

Eclipse は、指定された *ワークスペース* にすべてのプロジェクトおよびユーザーファイルを保存します。複数のワークスペースを用意して、切り替えることも可能です。ただし、Eclipse はその時点でアクティブなワークスペースからしかプロジェクトを読み込むことができません。アクティブなワークスペースの間で切り替えるには、**File > Switch Workspace > *</path/to/workspace>*** に進みます。また、**Workspace Launcher** ウィザードを使って新たなワークスペースを追加することもできます。このウィザードを開くには、**File > Switch Workspace > Other** に進みます。

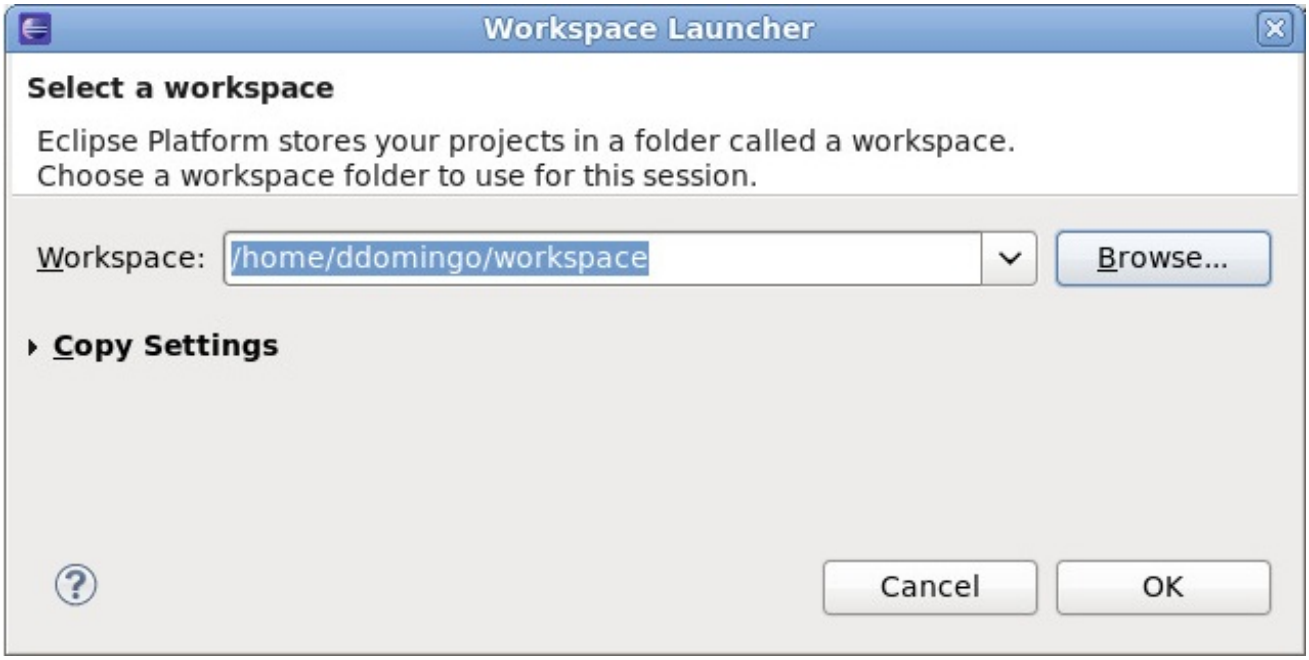



**図1.1 Workspace Launcher**

ワークスペース設定の詳細情報は、『Workbench User Guide』 (**Help Contents**) の 『Reference』 > 『Preferences』 > 『Workspace』 を参照してください。

プロジェクトに必要な Eclipse メタファイルがプロジェクトに含まれている場合は、Eclipse に直接インポートすることができます。Eclipse はこれらのファイルを使って実装するパースペクティブ、ツール、その他のユーザーインターフェース設定の種類を決定します。

このため、Eclipse で使用されたことのないプロジェクトをインポートしようとする際は、**Import** ウィザードではなく **New Project** ウィザードを使用する必要があります。これによりプロジェクトに必要な Eclipse メタファイルが作成され、プロジェクトをコミットする際にこれらのファイルを含めることもできます。

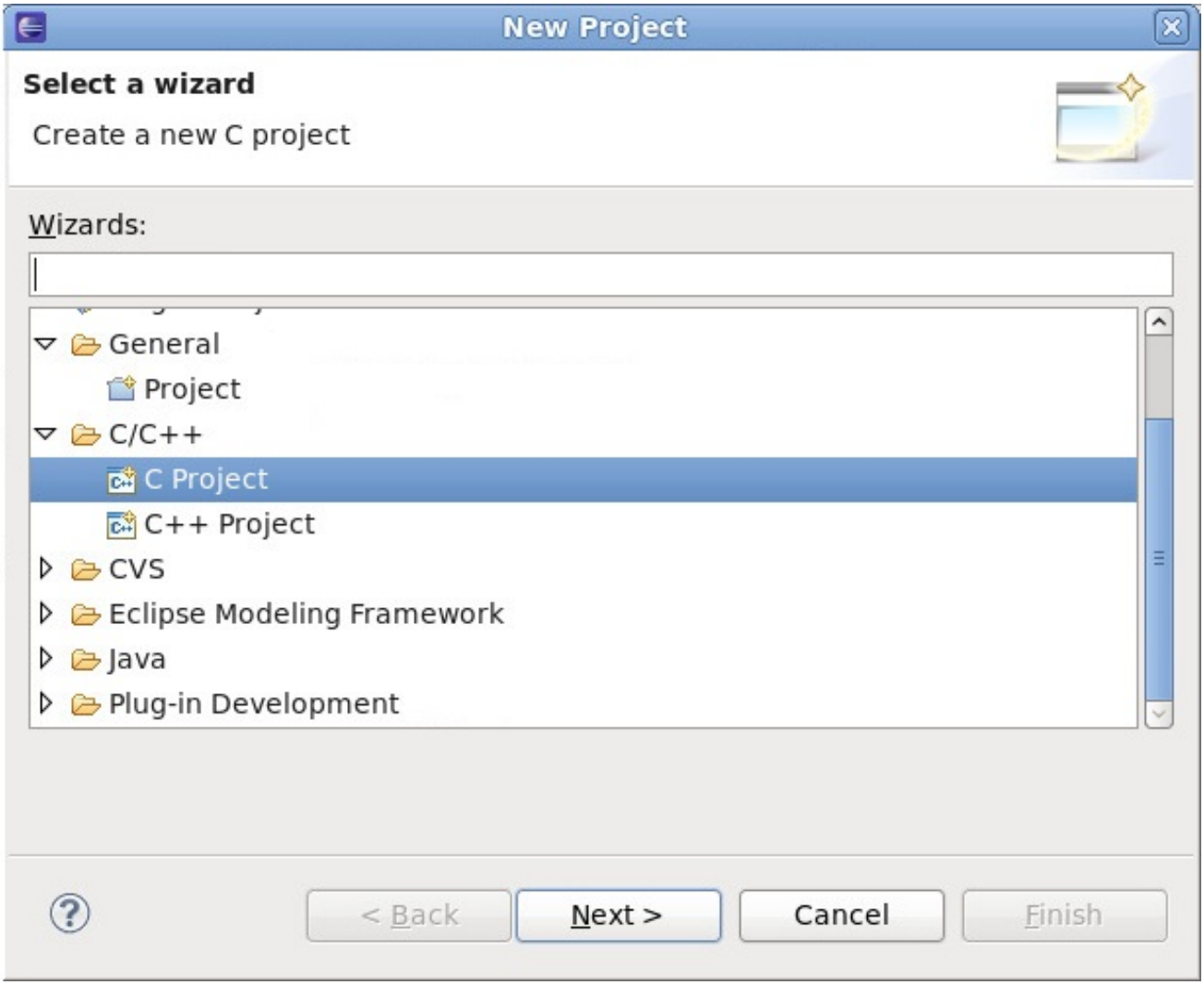



**図1.2 New Project ウィザード**

**Import** ウィザードは、**Eclipse** で作成もしくは以前に編集されたほとんどのプロジェクトに適しています。これらのプロジェクトには、必要な **Eclipse** メタファイルが含まれています。

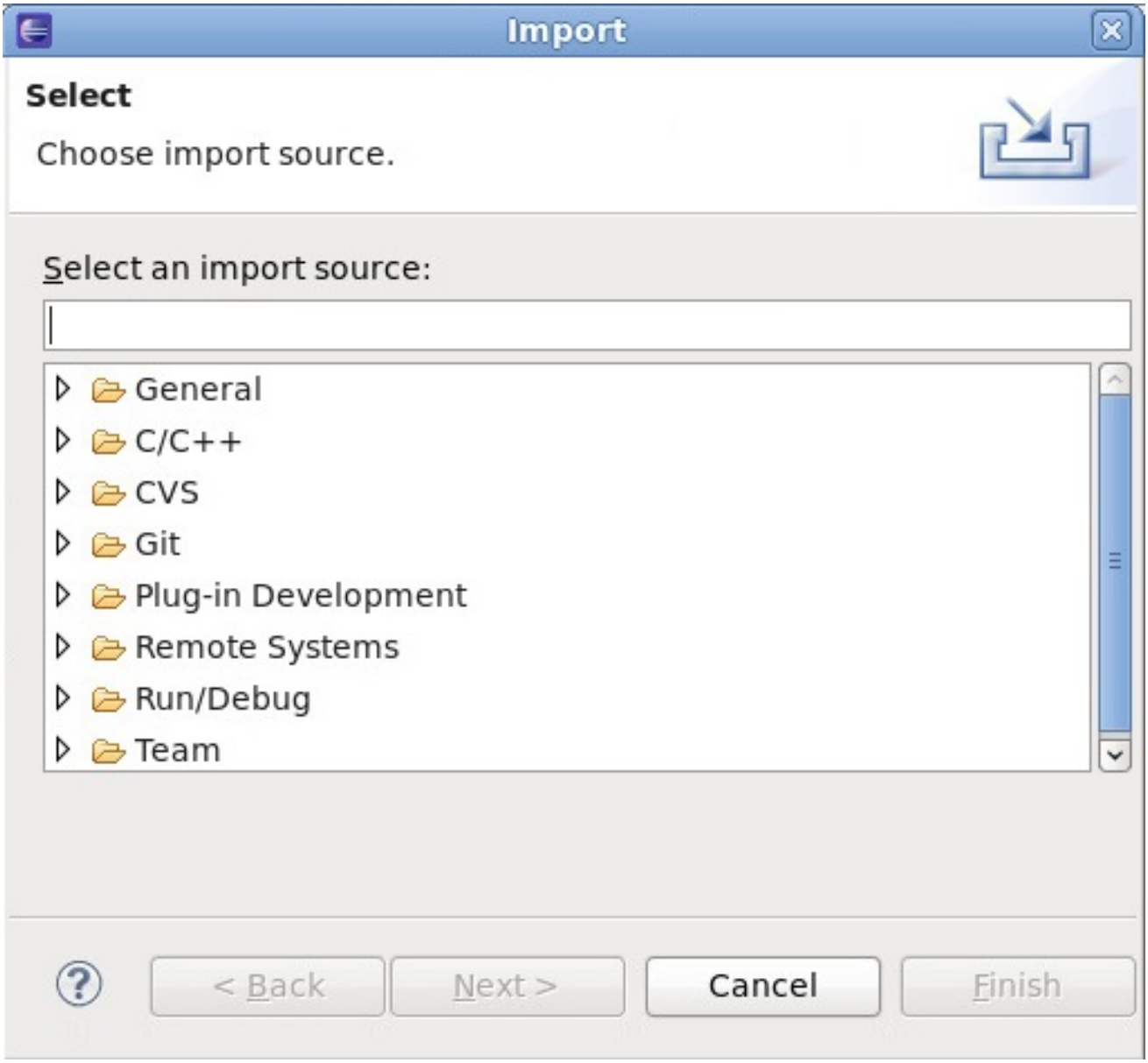



**図1.3 Import ウィザード**

## 1.2. ECLIPSE ユーザーインターフェース

図1.4「Eclipse ユーザーインターフェース (デフォルト)」 内のユーザーインターフェース全体は、Eclipse *ワークベンチ*と呼ばれています。通常これは、コードの**Editor**、**Project Explorer** ウィンドウ、いくつかのビューで構成されます。Eclipse ワークベンチ内のすべての要素は設定可能で、『Workbench User Guide』 (**Help Contents**) に詳細があります。ユーザーインターフェースのカスタマイズに関する概要は、「パースペクティブのカスタマイズ」を参照してください。

Eclipse には異なる *パースペクティブ*の機能があります。パースペクティブは、ビューとエディターのセットで、特定の種類のタスクやプロジェクトに有用なものです。Eclipse ワークベンチには 1 つ以上のパースペクティブを入れることができます。図1.4「Eclipse ユーザーインターフェース (デフォルト)」 は C/C++ 向けのデフォルトのパースペクティブを表示しています。

また Eclipse は多くの機能をいくつかのクラスに分け、個別の*メニューアイテム* 内にこれらを格納します。たとえば、**プロジェクト** メニューにはプロジェクトのコンパイル/構築関連の機能が格納されます。**ウィンドウ** メニューにはパースペクティブやメニューアイテム、その他のユーザーインターフェース要素を作成/カスタマイズするオプションが格納されます。各メインメニューの簡単な概要については、『C/C++ Development User Guide』 の **Reference> → C/C++ Menubar** か 『Java Development User Guide』 の **Reference → Menus and Actions** を参照してください。

以下のセクションでは、Eclipse *統合開発環境* (IDE) のデフォルトのユーザーインターフェースで表示される異なる要素の高レベルな概要を説明します。

Eclipse ワークベンチは、開発プロセスの各フェーズで必須の数多くの機能やツールにおけるユーザーインターフェースを提供します。このセクションでは、Eclipse の主要ユーザーインターフェースの概要を説明します。

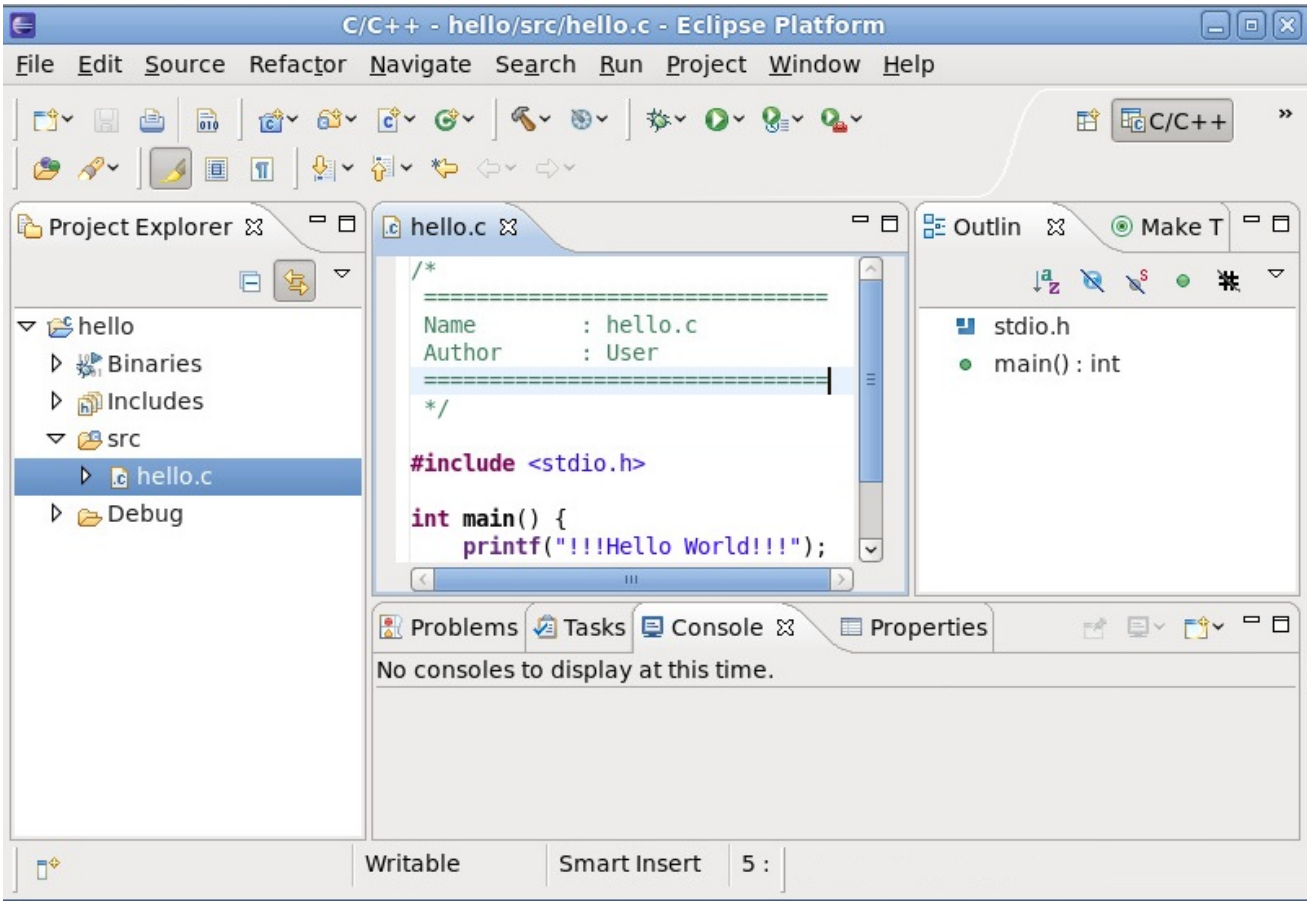



**図1.4 Eclipse ユーザーインターフェース (デフォルト)**

図1.4「Eclipse ユーザーインターフェース (デフォルト)」では C/C++ プロジェクト用のデフォルトのワークベンチが示されています。ワークベンチ内で利用可能なパースペクティブの切り替えを行うには、**Ctrl+F8** を押します。パースペクティブのカスタマイズに関するヒントについては、「パースペクティブのカスタマイズ」を参照してください。以下に続く図は、デフォルトの C/C++ パースペクティブで表示される基本的な要素です。

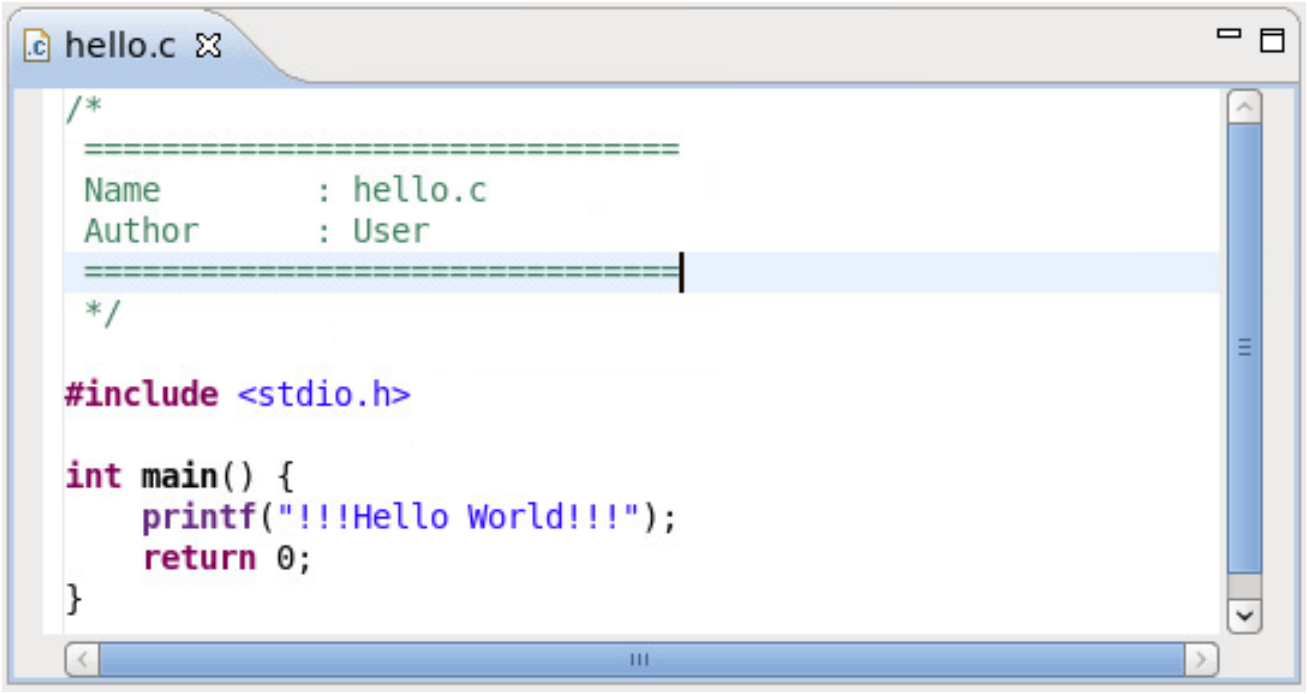



**図1.5 Eclipse Editor**

**Editor** はソースファイルの書き込みおよび編集に使用します。Eclipse はほとんどのタイプのソース言語で、適正な言語エディターを自動検出し読み込みます (たとえば、**.c** で終わっているファイルには C Editor )。**Editor** を設定するには、**Window > Preferences > *language (たとえば、Java、C++)*> Code Style** に移動します。

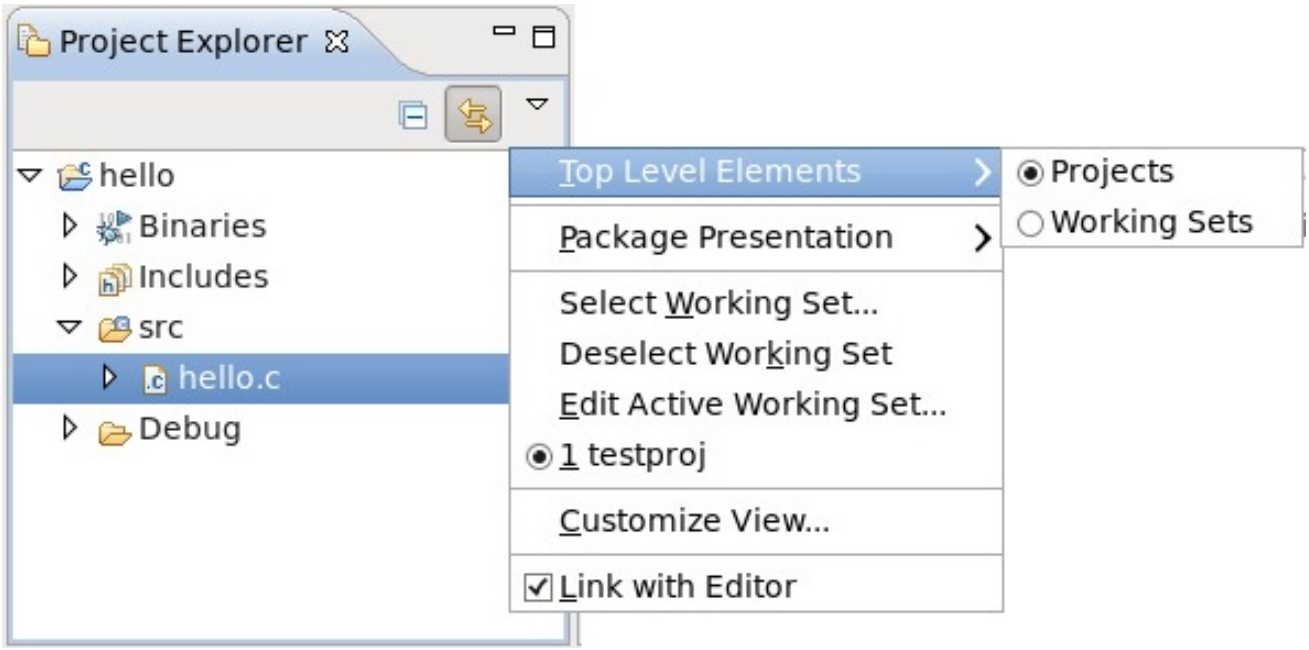



**図1.6 Project Explorer**

**Project Explorer View** は、全プロジェクトリソースの階層ビューを提供します (バイナリー、ソースファイルなど)。このビューからファイルを開いたり、削除したり、編集したりすることができます。

**Project Explorer View** の **View Menu** ボタンを使うと、**Project Explorer View** のトップレベルアイテムをプロジェクトとするか *working sets* とするかを設定できます。**working set** は一つのセットとして任意に分類されたプロジェクトのグループです。これは、関連またはリンクしているプロジェクトの編成に便利なものです。


**図1.7 Outline Window**

**Outline** ウィンドウは、ソースファイル内のコードの圧縮ビューを提供します。**Editor** 内の選択ファイルからの異なる変数や関数、ライブラリー、その他の構造体の要素を詳細表示します。これらはすべて、エディター固有のものです。

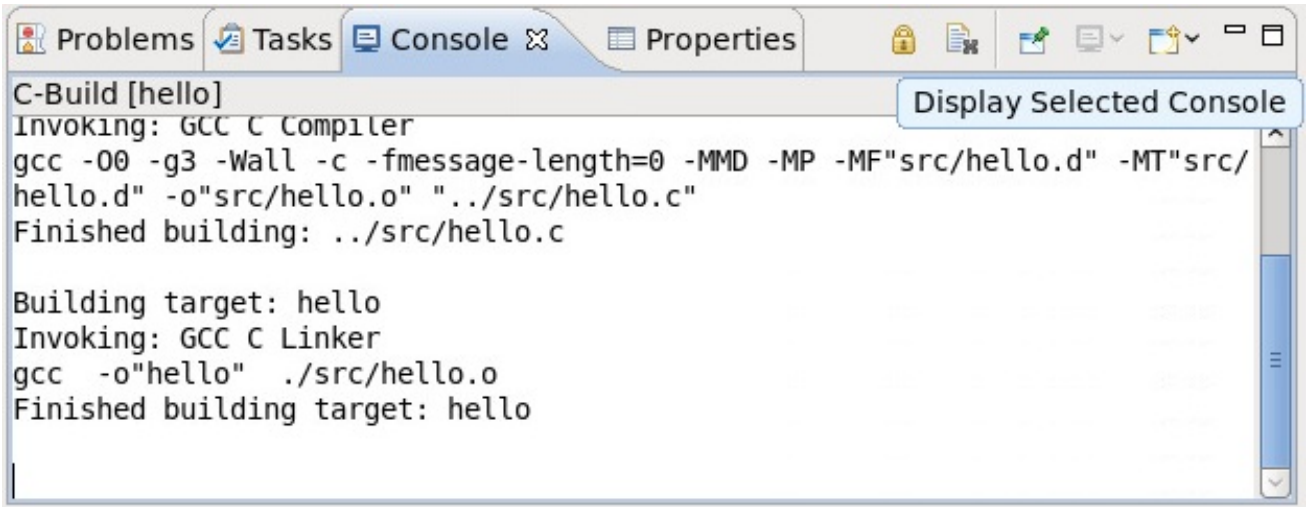



**図1.8 Console View**

Eclipse の機能やプラグインプログラムのなかには出力を **Console** ビューに送信するものもあります。このビューの **`Display Selected Console`** を使うと異なるコンソール間の切り替えができます。

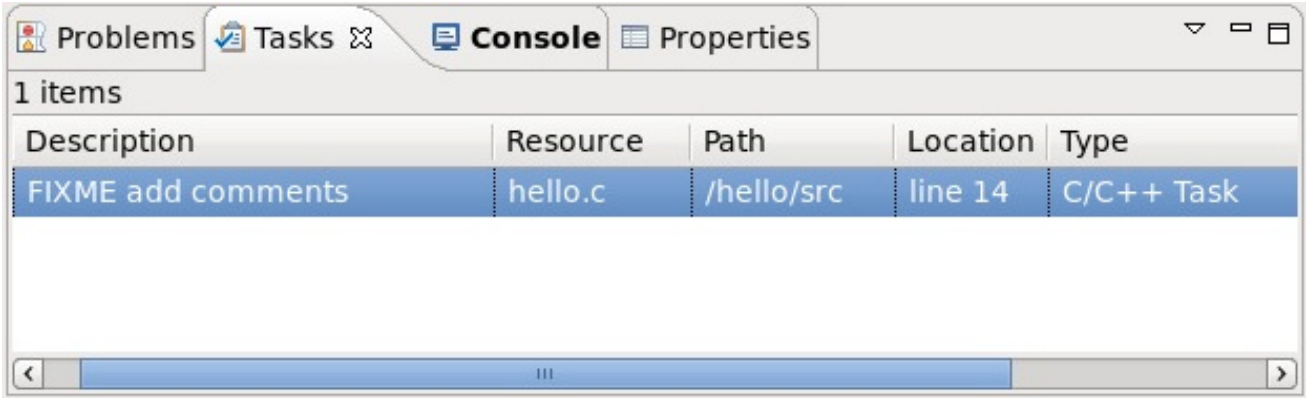



**図1.9 Tasks View**

**Tasks** ビューでは、コード内で特別にマークされたリマインダのコメントを追跡できます。このビューでは、各タスクコメントの場合が表示され、いくつかの方法でこれらを分類できます。


**図1.10 追跡されたコメントの例**

ほとんどの Eclipse エディターは **//FIXME** または **//TODO** タグでマークされたコメントを追跡します。*タスクタグ*と呼ばれるこれらの追跡されたタグは、他の言語で記述されたソースファイルによって異なります。タスクタグを追加または設定するには、**Window > Preferences** に移動し、**task tags** のキーワードで特定のエディター/言語のタスクタグ設定メニューを表示します。

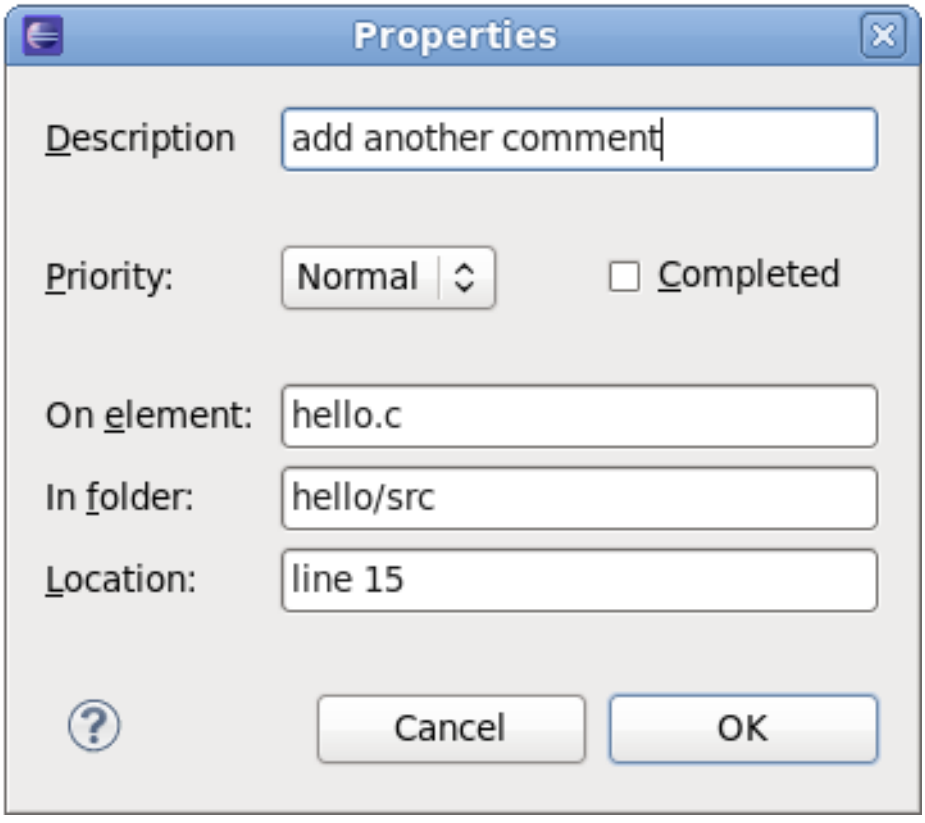



**図1.11 Task Properties**

別の方法では、**Edit > Add Task** に移動して、タスクの **Properties** メニュー (図1.11「Task Properties」) を開くこともできます。この方法だと、タスクタグを使わずにソースファイルの特定の場所にタスクを追加することができます。

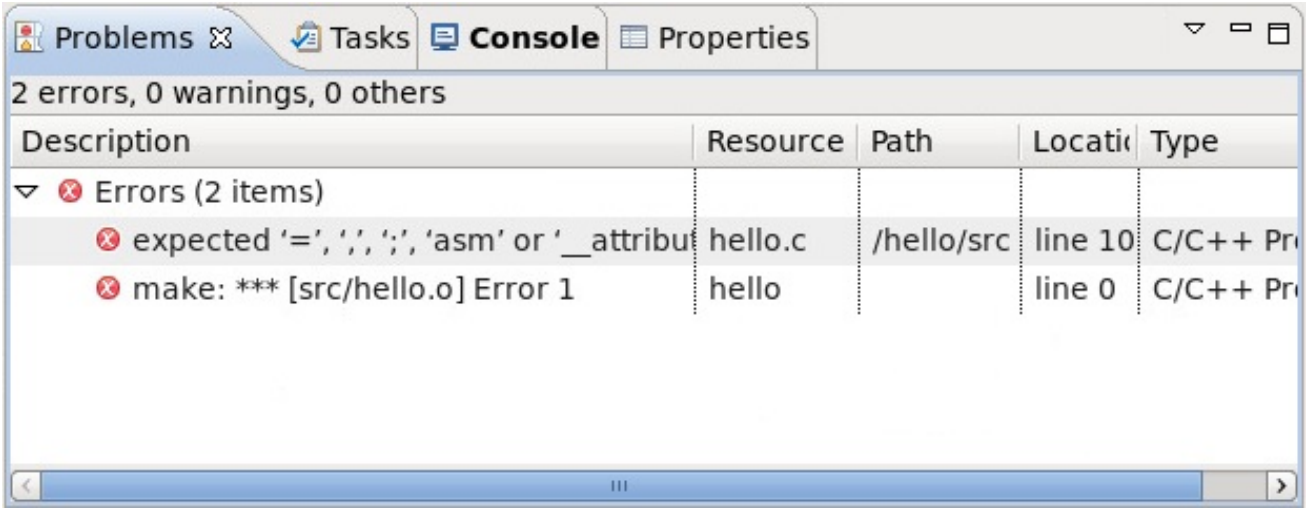



**図1.12 Problems View**

**Problems** ビューは、ビルドやクリーン、プロファイルの実行など特定のアクションの実行中に発生したエラーや警告を表示します。特定の問題に対する **"quick fix (緊急措置) "** の例を表示するには、問題を選択して **`Ctrl+1`** を押します。

### 1.2.1. クイックアクセスメニュー

Eclipse で最も便利な機能の一つが、**quick access** メニューです。**quick access** メニューに単語を入力すると、その単語に関連したビューやコマンド、ヘルプファイル、その他のアクションの一覧が表示されます。このメニューを開くには、**`Ctrl+3`** を押します。

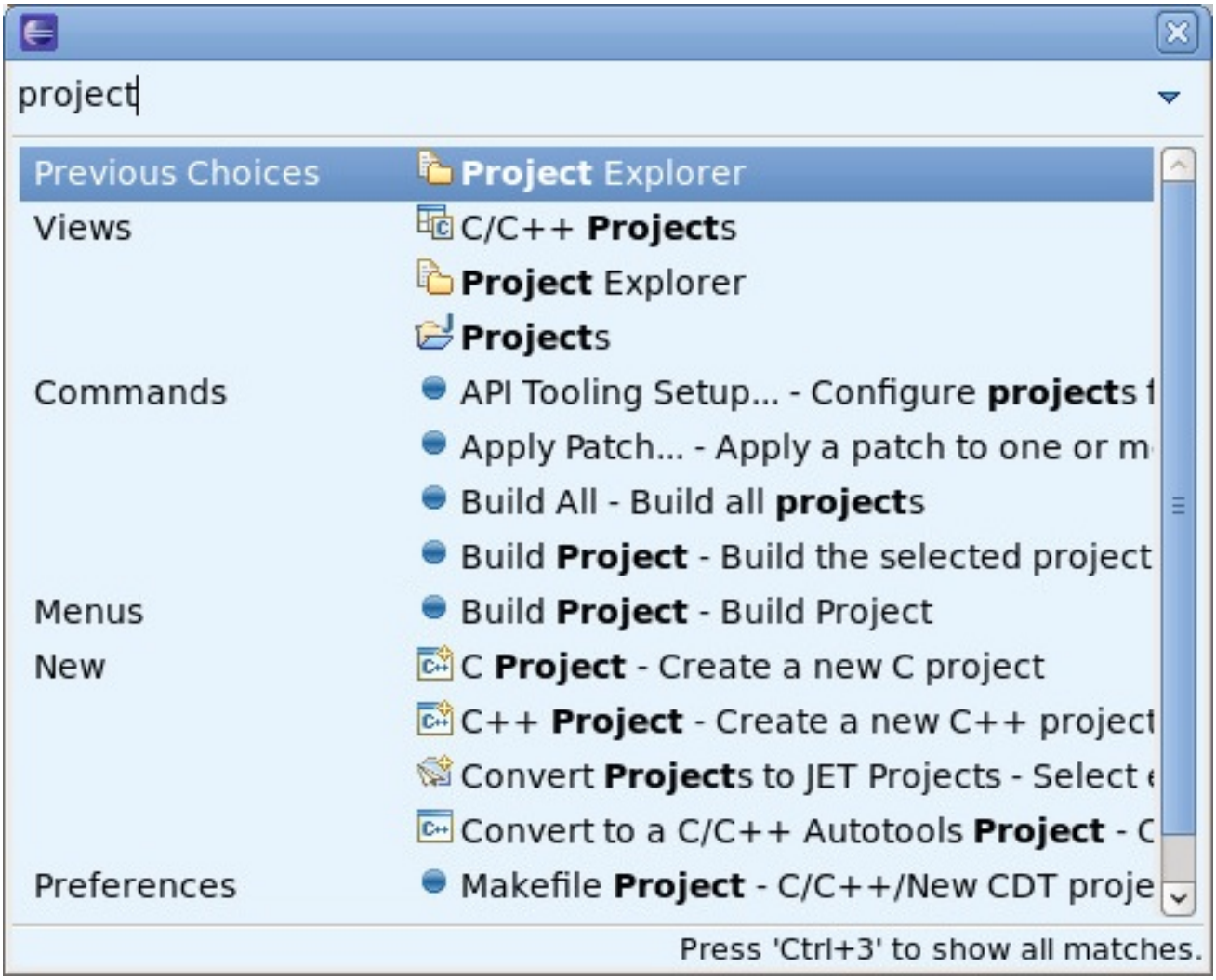



**図1.13 クイックアクセスメニュー**

図1.13「クイックアクセスメニュー」で **Views > Project Explorer** をクリックすると、**Project Explorer** ウィンドウが表示されます。**Commands**、**Menus**、**New**、**Preferences** カテゴリーからアイテムを選択して、これをクリックして実行します。これは、メニューオプションやタスクバーのアイコンをクリックするのと同様の動作です。また、矢印キーを使って **quick access** メニューを移動することもできます。

### 1.2.2. キーボードショートカット

すべてのキーボードショートカットの完全一覧を表示することもできます。**Shift+Ctrl+L** を押すと表示されます。

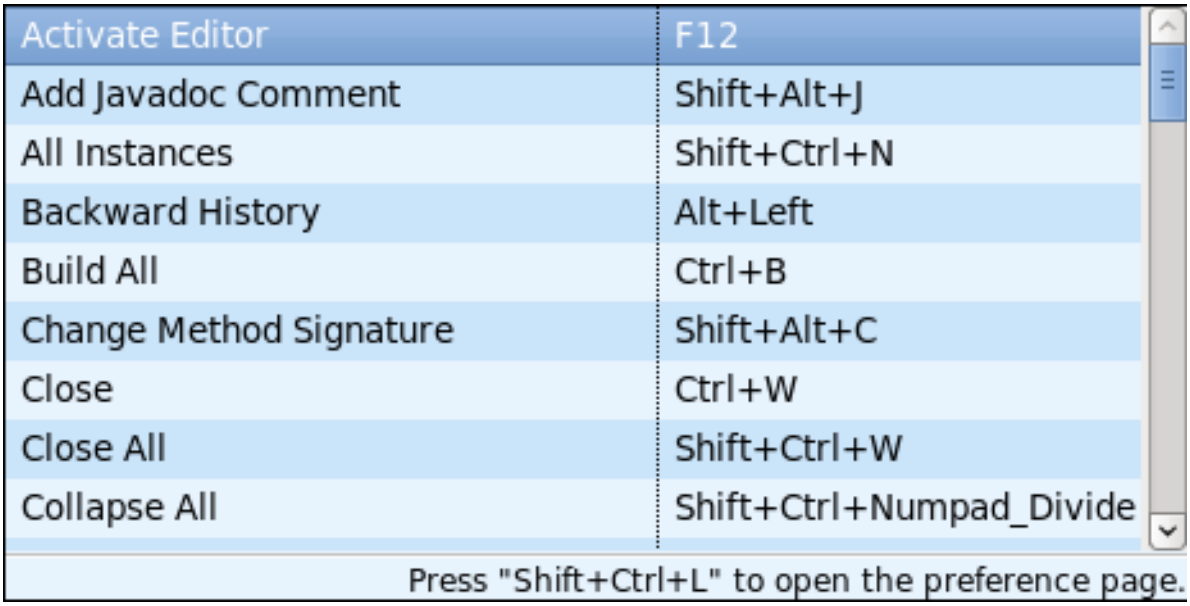



**図1.14 キーボードショートカット**

Eclipse のキーボードショートカットを設定するには、**Keyboard Shortcuts** リストの表示中に **`Shift+Ctrl+L`** を再度押します。

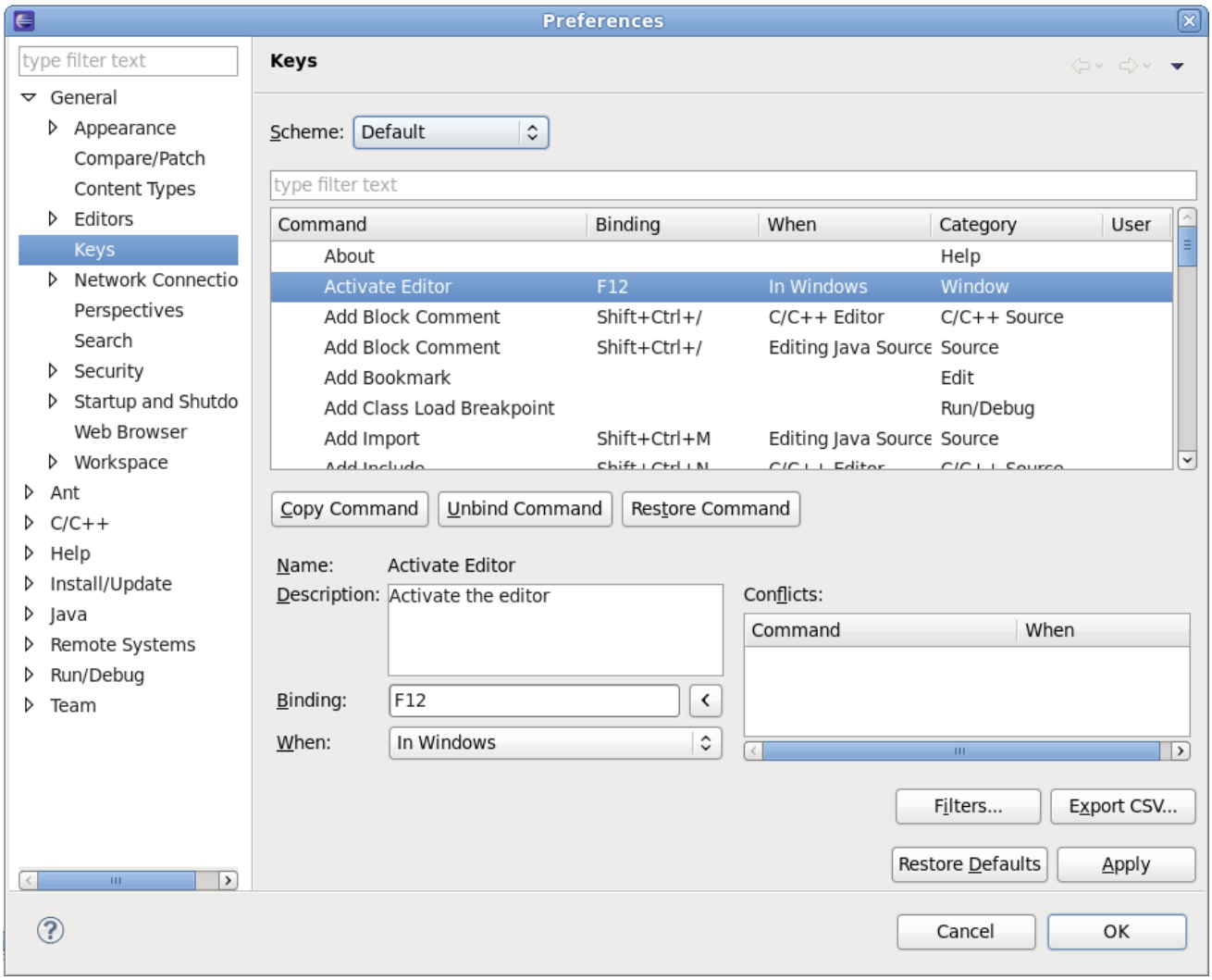



**図1.15 キーボードショートカットの設定**

### 1.2.3. パースペクティブのカスタマイズ

現在のパースペクティブをカスタマイズするには、**Window > Customize Perspective** を選択します。これで **Customize Perspective (パースペクティブのカスタマイズ)** メニューが開き、表示可能なツールバー、メインメニューアイテム、コマンドグループ、ショートカットが設定可能となります。

ワークベンチ内の各ビューの場所は、ビューのタイトルをクリックして移動した場所にドラッグすることでカスタマイズできます。

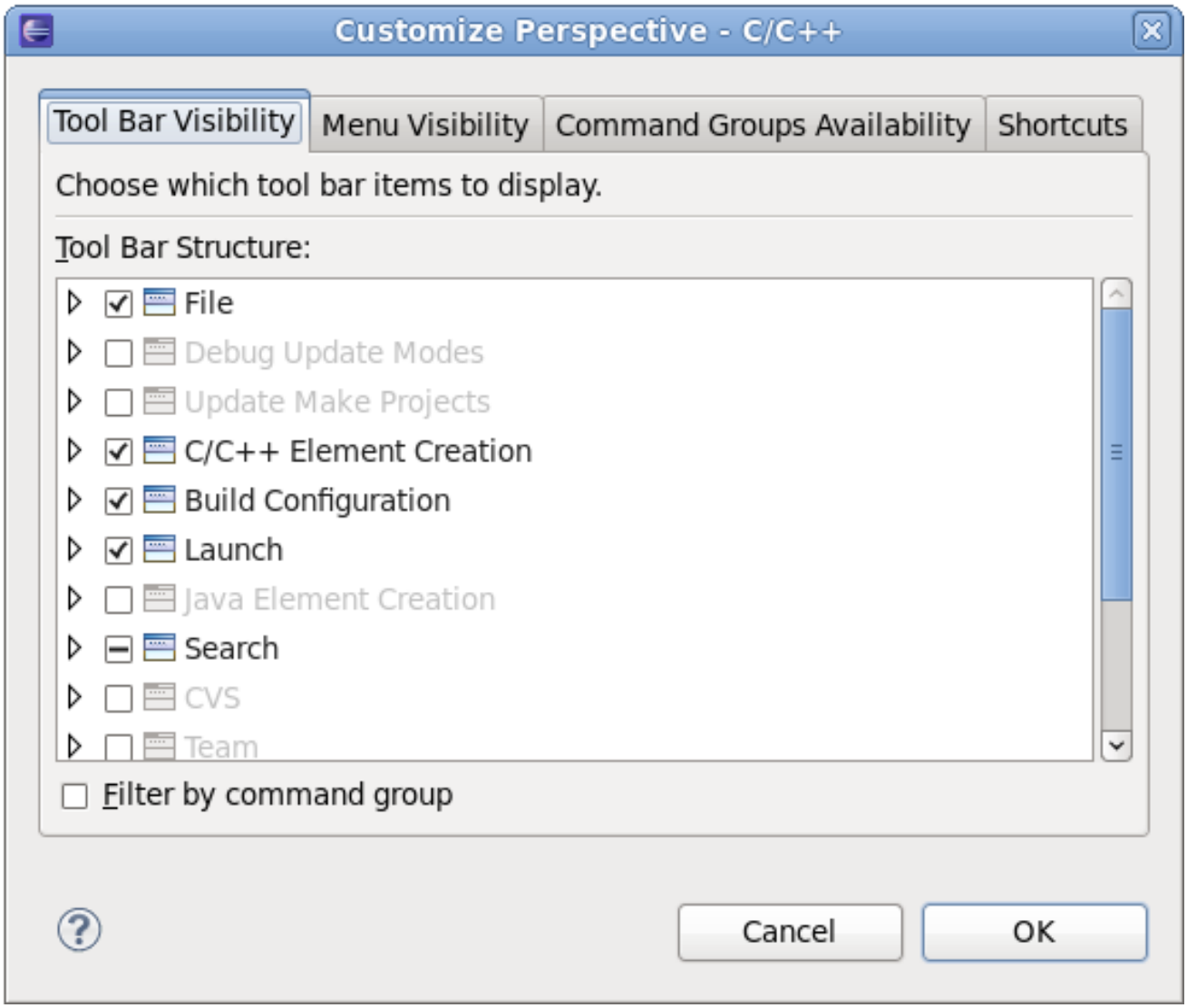



**図1.16 パースペクティブメニューのカスタマイズ**

図1.16「パースペクティブメニューのカスタマイズ」は **Tool Bar Visibility** タブを表示しています。名前が示すようにこのタブではツールバー表示を変更できます (図1.17「ツールバー」)。

**図1.17 ツールバー**

以下の図では **Customize Perspective Menu** の他のタブを表示しています。

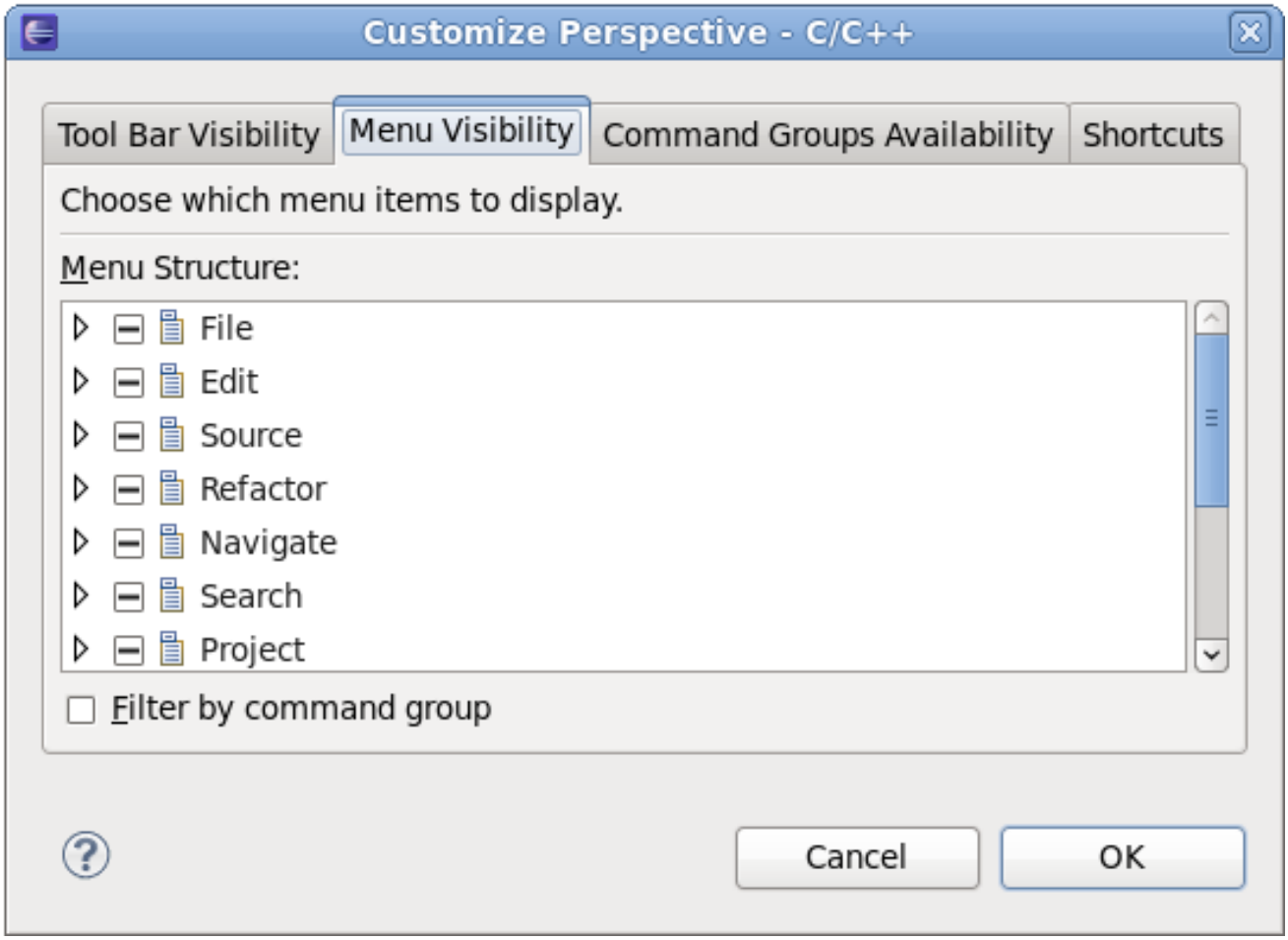



**図1.18 Menu Visibility タブ**

**Menu Visibility** タブでは、各メインメニューアイテムで表示される機能を設定します。メインメニューアイテムの概要については、『C/C++ Development User Guide』 の 『Reference』 > 『C/C++ Menubar』 か 『Java Development User Guide』 の 『Reference』 > 『Menus and Actions』 を参照してください。

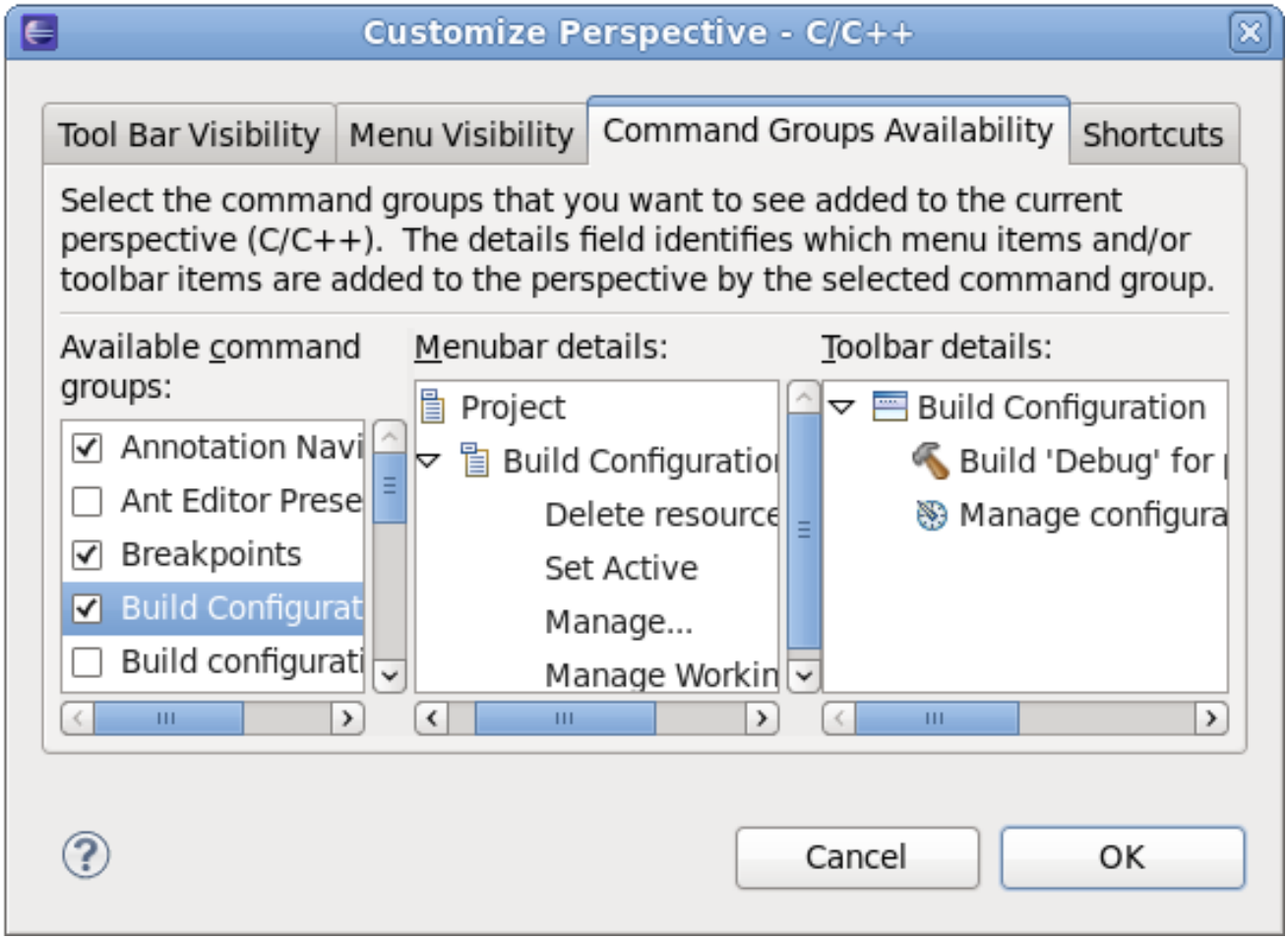



**図1.19 Command Group Availability タブ**

コマンドグループでは、機能またはオプションをメインメニューもしくはツールバーのエリアに追加します。**Command Group Availability** タブを使ってコマンドグループを追加もしくは削除します。**Menubar details** と **Toolbar details** フィールドでは、コマンドグループがそれぞれメインメニューとツールバーエリアに追加した機能もしくはオプションが表示されます。

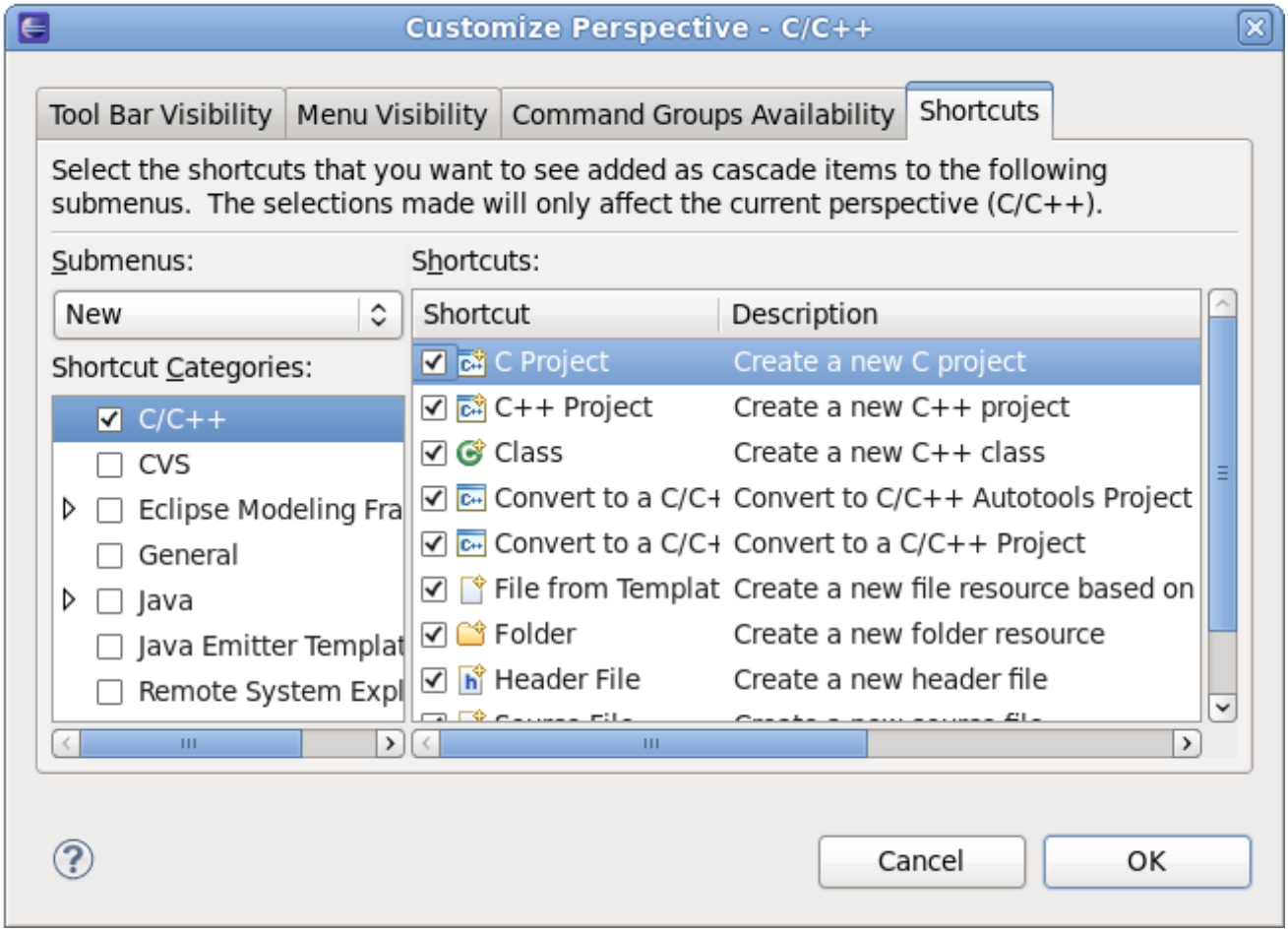



**図1.20 Shortcuts タブ**

**Shortcuts タブ** タブでは、以下のサブメニューで利用可能なメニューアイテムを設定します。

- **File > New**
- **Window > Open Perspective**
- **Window > Show View**

## 1.3. ECLIPSE での C/C++ ソースコードの編集

Red Hat Enterprise Linux 6 は C/C++ 開発用の Eclipse プラグイン CDT を提供します。C/C++ ソースコード用の特別エディターやメイクファイル、GNU Autotools ファイルも含まれています。プログラムの実行およびデバッグの機能も利用できます。

Eclipse テキストエディターはカット、コピー、ペースト、ブロック選択 (**Ctrl+Shift+A**) といったテキストエディターに通常備わっているほとんどの機能をサポートします。また、選択先の移動 (**Alt+Up/Down Arrow**) といった比較的独特な機能もあります。

C/C++ プログラマーにとって特に興味深いのは、*コンテンツアシスト*機能です。この機能はポップアップウィンドウで現行ファイル/場所で実行可能な関数、変数、テンプレートを提示します。これを起動するには、カーソルを希望の場所において、**Ctrl+Space** を押します。

ライブラリーからの関数呼び出しの詳細に関しては、「**libhover** プラグイン」 を参照してください。

Eclipse C/C++ コードエディターにはエラーの強調表示およびリファクタリングもあります。

コードエラーや警告は、色付きの波型下線で示されます。このようなエラーや警告は、コードをエディターに入力する際、もしくはビルドが発生しコンパイラー出力がこのようなマークに変換されて表示される場合があります。

提供されるリファクタリングツールには、コード要素の名前を変更する機能があります。

詳細は 「Eclipse による C/C++ アプリケーションのデバッグ」 または 「Eclipse 用の Autotools プラグイン」 を参照するか、ヘルプコンテンツ内にある 『C/C++ Development User Guide』 で**Concepts** → **Coding aids** と **Concepts** → **Editing C/C++ Files**、**Tasks** → **Write code** を参照してください。

### 1.3.1. `libhover` プラグイン

Eclipse 用の **`libhover`** プラグインは、GNU C ライブラリおよび GNU C++ 標準ライブラリにプラグアンドプレイのホバーヘルプサポートを提供します。これにより開発者は、*ホバーヘルプ*および *code completion* を使ってよりシームレスで便利な方法で Eclipse IDE 内で**`glibc`** および **`libstdc++`** ライブラリに関する既存ドキュメントを参照できます。

#### C++ 言語

メンバー関数の入力補完のドキュメントは C++ に対応していません。ヘッダーファイルからのプロトタイプのみが提供されます。また、C++ メンバー関数では、ソースファイルにヘッダーファイルを追加する機能は対応していません。

C++ ライブラリーリソースでは、**`libhover`** は CDT インデクサーを使ってファイルの**インデックス化**をする必要があります。インデックス化は、ビルドのコンテキストで対象ファイルを解析します。ビルドコンテキストは、ヘッダーファイルがどこから来るか、またタイプやマクロ、同様のアイテムがどのように解決されるかを決定します。**`libhover`** はヘッダーファイルの場所が分かっている場合もありますが、C++ ソースファイルのインデックス化ができるようになるには、通常実際のビルドが先に実行される必要があります。

C++ メンバー関数名はドキュメントの検索に十分な情報ではないので、**`libhover`** プラグインは C++ ソースのインデックス化を必要とする場合があります。C++ は、クラス内においても異なるクラスが同一名のメンバーを持つことを可能とし、メンバーは異なるメソッドシグネチャの同一名を持つ場合があります。これには、クラス名と関数のパラメーターの署名が提供されて正確にどのメンバーが参照されているかを判断する必要があります。

また、C++ にはタイプ定義とテンプレート化されたクラスもあります。この様な情報は、ファイル全体と関連の **`include`** ファイルの解析が必要になります。**`libhover`** はインデックス化によってのみ、これができます。

#### C 言語

C 関数では、入力補完 (**`Ctrl+Space`**) を実行することで、潜在的なソースに追加される C 関数の一覧が提供され (たとえば、prin と入力して **`Ctrl+Space`** を押すと入力補完候補の 1 つとして**`printf`** が一覧表示されます)、新たなウィンドウでドキュメントが表示され、厳密にどの C 関数が必要かを判断します。

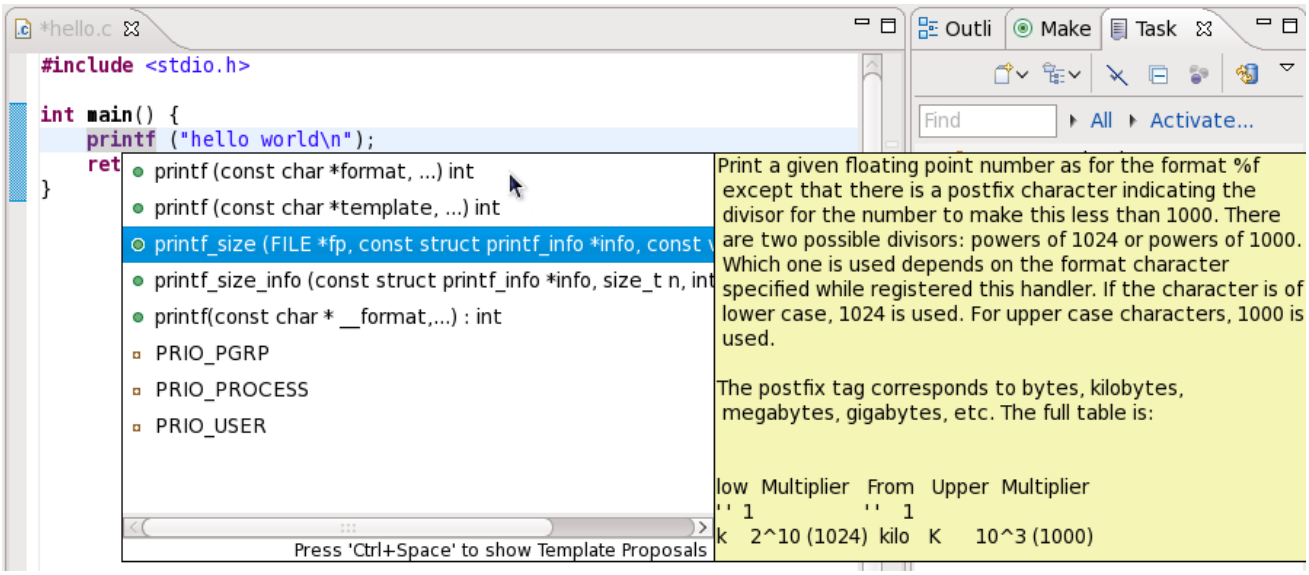



**図1.21 Code Completion (入力候補) の使用**

C 関数は名前だけでドキュメント内での参照が可能です。このため、**libhover** はホバーヘルプや code 入力候補を提供するために C ソースファイルをインデックス化する必要がありません。C ライブラリー関数用の適切なヘッダーファイルインクルードステートメントは、すでに存在していない場合は自動的に追加されます。

ファイル内で C 関数を選択し、右クリック > **Source** > **Add Include** と進んで、ソースに必要なヘッダーファイルを自動的に追加します。これは、**Shift+Ctrl+N** を押すことでも実行できます。

#### 1.3.1.1. 設定および使用方法

すべてのインストール済み **libhover** ライブラリー用のホバーヘルプはデフォルトで有効となっており、プロジェクトごとに無効とすることができます。特定のプロジェクトごとにホバーヘルプを無効または有効にするには、プロジェクト名を右クリックして **Properties** をクリックします。表示されるメニューで **C/C++ General** > **Documentation** と進みます。**Help books** セクションのライブラリーにチェックを入れるか外すかして、そのライブラリーのホバーヘルプを有効/無効にします。

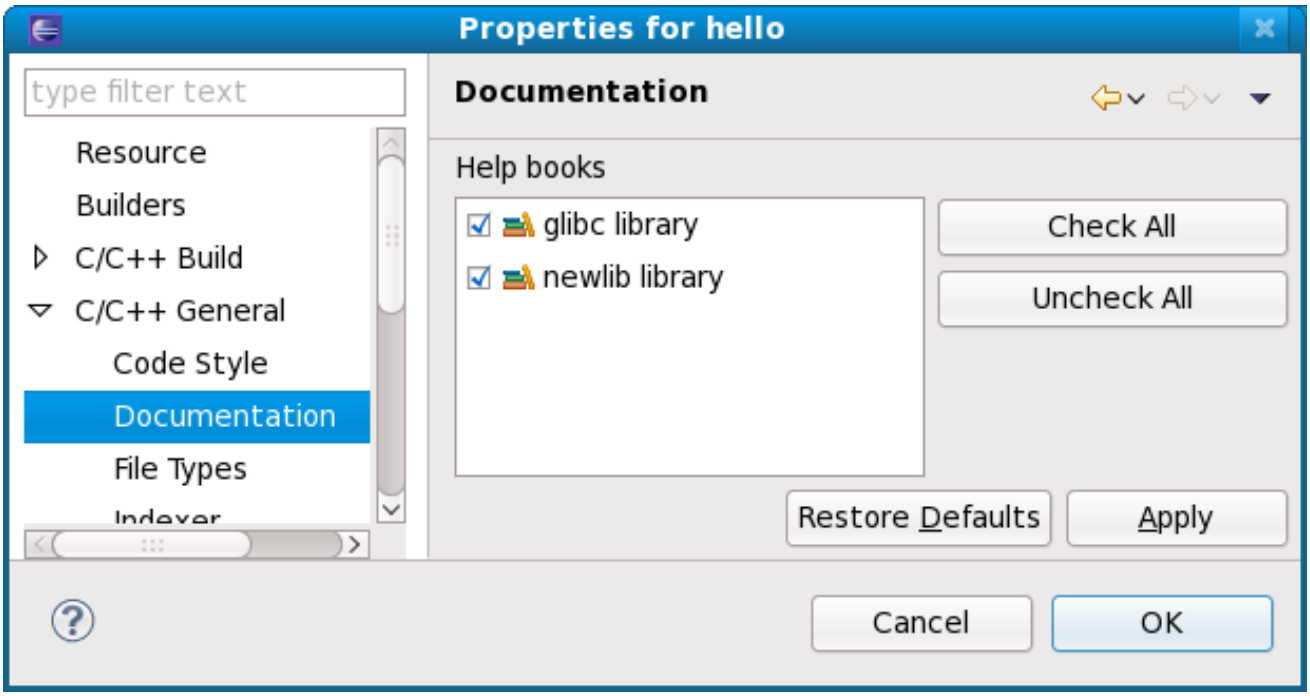



**図1.22 ホバーヘルプの有効/無効化**

複数の **libhover** ライブラリが機能上で重複している場合は特に、特定のライブラリのホバーヘルプを無効にすることをお勧めします。例えば、**newlib** C ライブラリのような C ライブラリ用の **libhover** プラグインが手動でインストールされたとします (**newlib** C ライブラリプラグインは Red Hat Enterprise Linux 6 で提供されていないことに注意してください)。ホバーヘルプには C 関数が含まれており、その名前が GNU C ライブラリ内のものと重複します (デフォルトで提供) 。これらのホバーヘルプの両方が同時にアクティブになるのはユーザーに好ましくないので、どちらかを無効にすることが実用的です。

複数の **libhover** ライブラリーが有効となっていてライブラリー間で関数の重複がある場合、**Help books** セクションで **最初に** 一覧表示されているライブラリーからの関数のヘルプコンテンツがホバーヘルプに表示されます (つまり、図1.22「ホバーヘルプの有効/無効化」では **glibc** がこれに当たります)。code 入力候補では、**libhover** が有効なすべての **libhover** ライブラリーから考えられるすべての代替案を提供します。

ホバーヘルプを使うには、 **C/C++ Editor** で関数名またはメンバー関数名の上にカーソルを持っていきます。2、3秒のうちに、**libhover** が選択した C 関数もしくは C++ メンバー関数に関するライブラリードキュメントが表示されます。

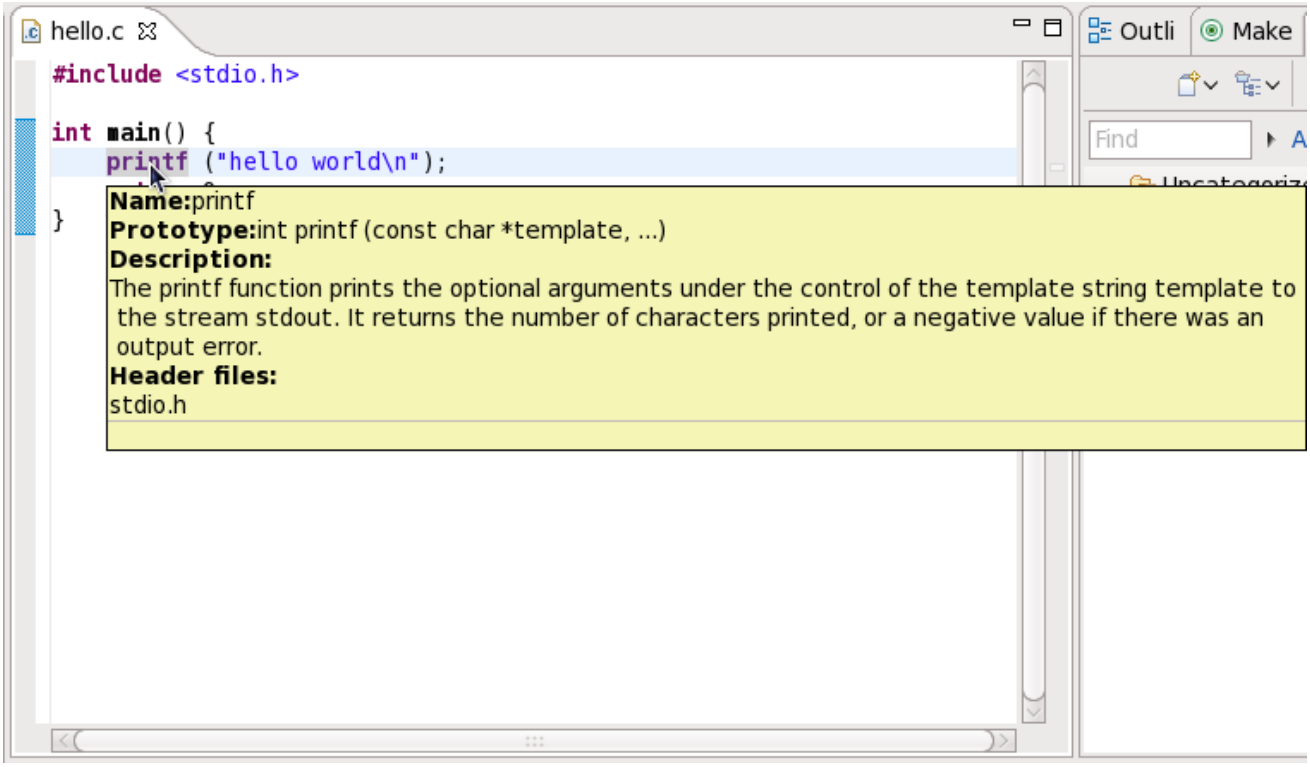



**図1.23 ホバーヘルプの使用**

## 1.4. ECLIPSE での JAVA ソースコードの編集

Red Hat Enterprise Linux 6 は Java (Java SE) 開発用の Eclipse プラグイン JDT を提供します。Java ソースコードや ant **build.xml** 用の特別エディターも含まれています。プログラムの実行およびデバッグの機能も利用できます。

Eclipse は Java 開発者用にフル機能搭載のインタラクティブな開発環境を提供します。

### New Project ウィザード

Eclipse の **New Project ウィザード** は、Java プロジェクトの開始に必要な標準的セットアップのほとんどを実行します。これによりユーザーは、どの Java ランタイム環境を使用するかや望ましいプロジェクトファイルツリーのレイアウトなどの様々なオプションを選択したり、カスタマイズしたりすることができます。

既存プロジェクトのエクスポートと同じ手順を行い、場所を求められたら、代わりに既存プロジェクトの場所を入力します。

新規 Java プロジェクトの設定に関する詳細は、**Help > Help Contents > Java Development > Getting Started > Basic Tutorial > Creating Your First Java Project** を参照してください。

#### コンテンツアシスタンス

Eclipse の Java 開発環境 (JDT) は、豊富なコンテンツアシスタンス機能を提供することで、生産性を高め、エラーを減らします。これは通常、**Ctrl + Space** を押すと起動します。この機能に含まれるのは、コードやライブラリーでのメソッド名の完成や、Javadoc でのパラメーター名の挿入、メソッド呼び出しの際のパラメーターの記入などです。これは完全にカスタマイズ可能で、特定候補を禁止したり、コード記入の際に使用するカスタムコードテンプレートを追加したりすることができます。

これらの機能の概要については、**Help > Help Contents > Java Development User Guide > Tips and Tricks** を参照してください。

#### コードの書式設定

コードの書式設定は **Ctrl + Shift + F** でアクセスできる JDT の別の便利な機能です。書式設定は **Window > Preferences > Java > Code Styler > Formatter** に移動し、インストール済みの書式設定プロファイルを使用するか、プロジェクトのスタイルに一致するように新たなプロファイルを作成するかを選択することができます。

#### デバッグ機能

JDT にはデバッグ機能もいくつかあります。対象のコードの行の左マージンでダブルクリックすると、ブレークポイントが作成されます。デバッガーの実行中は、そのコード行でプログラムは停止するので、これはエラーの場所の検出に役立ちます。

デバッガーの初回実行時に設定されるデバッグパースペクティブは、デバッグに関連するビューをよりすぐれたものにする別のレイアウトです。たとえば、**Expressions** ビューは、現行フレームのコンテキスト内での Java 演算の評価を可能にします。

デバッグパースペクティブを構成するビューは、すべてのビューがそうであるように、**Window > Show View** でアクセスでき、これらのビューへのアクセスにデバッグする必要はありません。

デバッグ中に変数の上にカーソルを持って行くと、その値が表示されます。または、**Variables** ビューを使用します。デバッグビューを使用すると、プログラムの実行を管理し、スタック上の様々なフレームを参照することができます。

JDT でのデバッグに関する詳細は、**Help > Help Contents > Java Development > Getting Started > Basic Tutorial > Debugging Your Programs** を参照してください。

#### JDT 機能

JDT は高度にカスタマイズ可能で、幅広い機能があり、その一覧は、**Window > Preferences > Java** と **Project > Properties** の Java 設定で表示されます。JDT の詳細なドキュメントとその機能については、**Help > Help Contents > Java Development User Guide** にある 『Java Development User Guide』 を参照してください。

## 1.5. ECLIPSE RPM ビルディング

Eclipse 用の Specfile Editor プラグインは、開発者が **.spec** ファイルを管理する際に役立つ機能を提供します。このプラグインを使うと、ユーザーは **.spec** ファイルの編集の際に、オートコンプリート機能やハイライト、ファイルのハイパーリンク、折り曲げなどのいくつかの Eclipse GUI 機能を活用することができます。

また、Specfile Editor プラグインは **rpmlint** ツールを Eclipse インターフェースに統合します。**rpmlint** はコマンドラインツールで、開発者が一般的な RPM パッケージエラーを検出する際に役立ちます。Eclipse インターフェースが提供する豊富な仮想化は、**rpmlint** がレポートするミスを開発者が迅速に検出、表示、訂正する際に役立ちます。

Eclipse の **.spec file editor** プラグインは、RPM プロジェクトからの RPM ファイルのビルドもサポートします。この機能は、エクスポートウィザード **(Import → RPM → Source/Binary RPM)** を利用することで使用可能になり、ソース RPM **(src.rpm)** かバイナリー RPM、もしくはその両方が必要かどうかを選択することが可能になります。

ビルド出力は Eclipse コンソールビューにあります。一定数のビルド失敗には、ハイパーリンクのサポートがあります。つまり、ビルド失敗の特定部分は Eclipse コンソールビューでハイパーリンクに変更され **(Ctrl+Click)**、これがユーザーを問題を発生させている **.spec** ファイルの実際の行に向けます。

また、ソース RPM **(.src.rpm)** ファイルのインポートウィザードも重要です。これは、**Import → RPM → Source RPM** にあります。これを使用すると、ソース RPM が既に作成されている場合、設定なしでユーザーは簡単に開始することができます。するとこのプロジェクトは、spec ファイルの編集と、ソース/バイナリー RPM へのビルド (エクスポート) の準備ができたことになります。

詳細はヘルプコンテンツにある 『Specfile Editor User Guide』 の **Specfile Editor User Guide → Import src.rpm and export rpm and src.rpm** セクションを参照してください。

## 1.6. ECLIPSE のドキュメント

Eclipse には包括的な内部ヘルプライブラリーがあり、統合開発環境 **(IDE)** のほとんどすべての側面がカバーされています。Eclipse のドキュメントプラグインはすべて、そのコンテンツをこのライブラリーにインストールし、そこでインデックス化されます。このライブラリーにアクセスするには、**Help** メニューを使います。

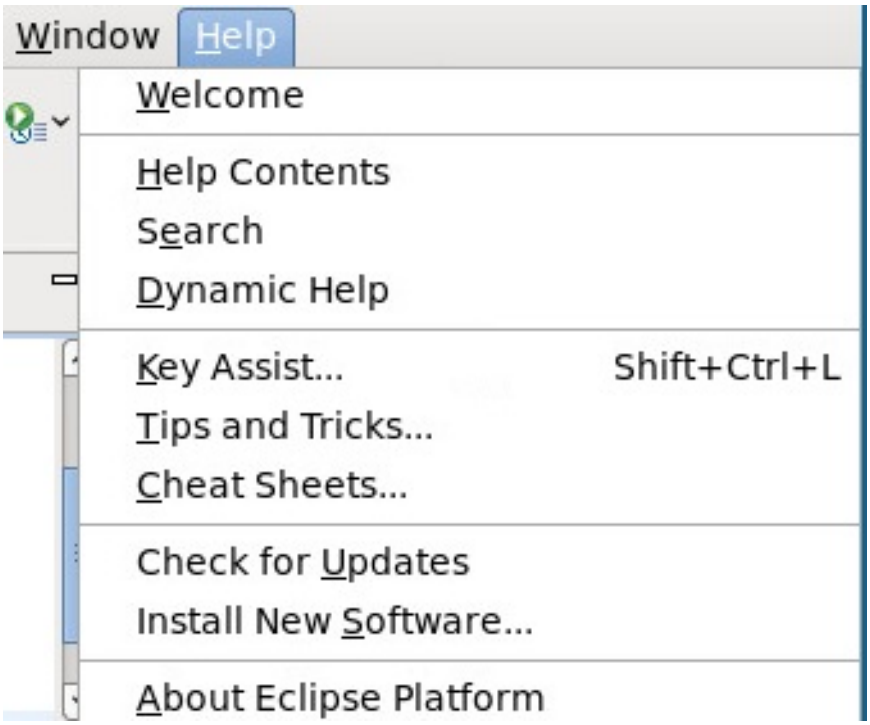



**図1.24 Help**

メインの **Help** メニューを開くには、**Help > Help Contents** に移動します。**Help** メニューでは、**Contents** フィールドにインストール済みドキュメントプラグインが提供するすべての利用可能なコンテンツが表示されます。

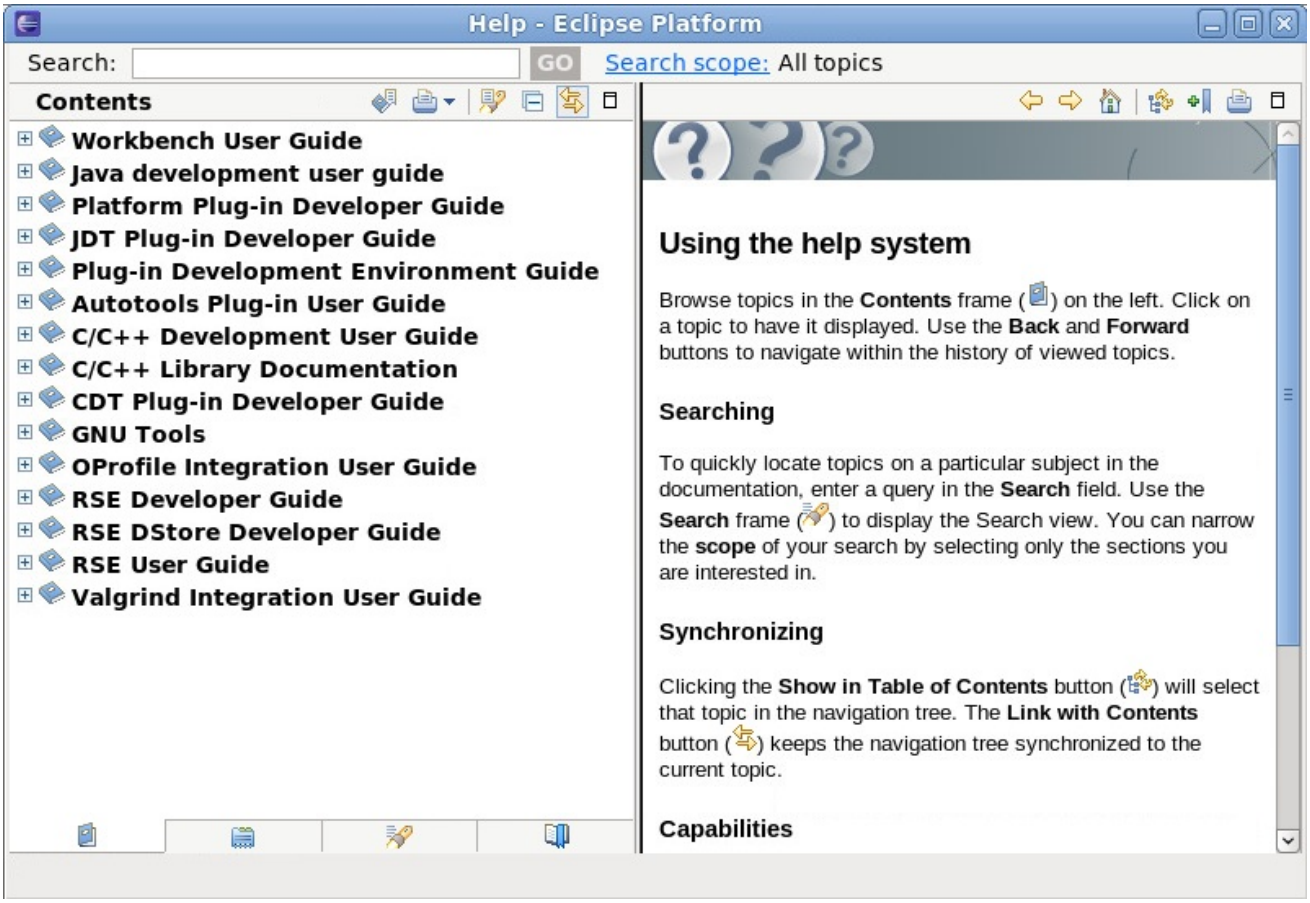



**図1.25 Help Menu**

**Contents** フィールド下部のタブは、Eclipse ドキュメントにアクセスするオプションになります。セクション/ヘッダーごと、または **Search** フィールドで検索して各「ブック」に移動できます。各ブックにブックマークを付けて、**Bookmarks** タブからその箇所にアクセスすることもできます。

『Workbench User Guide』 は Eclipse ユーザーインターフェースの全側面を幅広くカバーしています。Eclipse の機能方法を理解するために便利な Eclipse ワークベンチ、パースペクティブ、異なるコンセプトの非常に低レベルな情報が含まれています。『Workbench User Guide』 は、Eclipse や IDE 全般に関する経験がほとんどない、またはある程度あるユーザーに理想的なリソースとなります。このドキュメントプラグインはデフォルトでインストールされます。

Eclipse ヘルプシステムには、*動的ヘルプ*機能も含まれています。この機能はワークベンチ内で新たなウィンドウを開き、選択されたインターフェース要素に関連のあるドキュメントを表示します。動的ヘルプをアクティベートするには、**Help > Dynamic Help** に移動します。

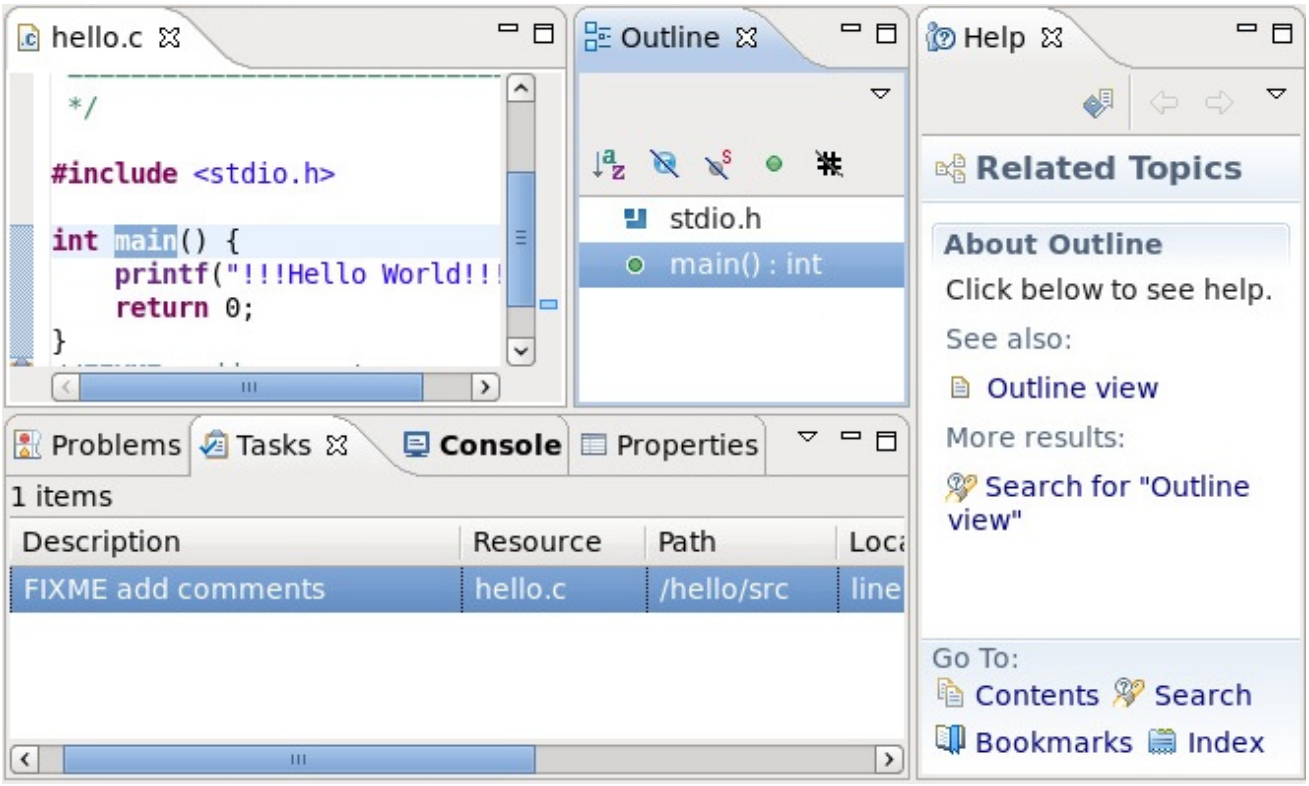



**図1.26 動的ヘルプ**

図1.26「動的ヘルプ」の右端のウィンドウには **Outline** ビューに関連するヘルプトピックが表示されています。これが選択されたユーザーインターフェースの要素だからです。

# 第2章 協同作業

複数の開発者が関わるすべてのプロジェクトでは、効果的な改訂管理が必須になります。改訂管理を効果的に行うことで、チーム内の開発者全員が組織的かつ規則的な方法でコードを作成、見直し、改訂、記録できるようになります。Red Hat Enterprise Linux 6 は、オープンソースの改訂管理システムのなかで最も人気のある 3 つのシステム (CVS、SVN、Git) をサポートしています。これらの改訂管理システムのツールは、個別の内部コードリポジトリーを設定する機能や、公開されているさまざまなオープンソースコードを使用できるようにします。

以下のセクションでは、各ツールの関連資料の概要と参照先を説明します。

## 2.1. CONCURRENT VERSIONS SYSTEM (CVS)

通常 **CVS** と略される **Concurrent Versions System** は、クライアントサーバーアーキテクチャーで構成される *集中型バージョン管理システム* です。CVS は、以前の Revision Control System (RCS) の後継で、リビジョン管理が行われるファイルへの変更をすべてトラッキングしつつ、複数の開発者が同じプロジェクトで協同で作業できるようにします。

### 2.1.1. CVS のインストールおよび設定

**cvs パッケージのインストール**
**CVS** は、Red Hat Enterprise Linux 6 では cvs パッケージで提供されます。cvs パッケージとすべての依存関係をお使いのシステムにインストールするには、**root** として、シェルプロンプトで以下のコマンドを入力します。

```
yum install cvs
```

これにより、コマンドライン CVS クライアントとその他の関連ツールがシステムにインストールされます。

**デフォルトのエディターの設定**
コマンドラインで CVS を使用する場合に、**cvs import** または **cvs commit** など、特定のコマンドでは短いログメッセージを記述する必要があります。**cvs** クライアントアプリケーションはまず環境変数 **$CVSEDITOR** のコンテンツを読み込み、次に、より一般的な環境変数**$EDITOR** を読み込み、どのテキストエディターを起動するかを決定します。どちらの変数も指定されていない場合には、**vi** が起動します。

環境変数 **$CVSEDITOR** の値を永続的に変更する場合は、以下のコマンドを実行します。

```
echo "export CVSEDITOR=command" >> ~/.bashrc
```

これにより、**export CVSEDITOR=***command* の行を **~/.bashrc** に追加します。*command* は、任意のエディターを実行するコマンドに置き換えます (例: **emacs**)。現在の shell セッションで、この変更を有効にするには、シェルプロンプトで以下のコマンドを入力して **~/.bashrc** のコマンドを実行する必要があります。

```
. ~/.bashrc
```

**例2.1 デフォルトのテキストエディターの設定**

CVS クライアントがテキストエディターとして **Emacs** を使用するように設定するには、以下を入力します。

```
~]$ echo "export CVSEDITOR=emacs" >> ~/.bashrc
~]$ . ~/.bashrc
```

### 2.1.2. 新規リポジトリーの作成

CVS *リポジトリー*は、リビジョン管理がされているファイルやディレクトリー、完全な変更履歴や変更者や変更時間の情報などの追加データを保存する中央の場所となっています。一般的な CVS リポジトリーは、*モジュール* と呼ばれる個別のサブディレクトリーで複数のプロジェクトを保存します。公開されている場合には、複数の開発者がモジュールの *作業コピー* を作成したり変更したりできるだけでなく、リポジトリーに *コミット* することで他の作業者と変更を共有することができます。

#### 空のリポジトリーの初期化

任意のディレクトリーに空の CVS リポジトリーを新規作成するには、以下のコマンドを実行します。

```
cvs -d path init
```

*path* は、リポジトリーの保存先のディレクトリーへの絶対パスを指定する必要がある点を念頭に置いてください (例: **/var/cvs/**)。または、**$CVSROOT** の環境変数の値を変更して、このパスを指定することもできます。

```
export CVSROOT=path
```

これにより **cvs init** やその他の CVS 関連のコマンドの実行時にパスを省略することができます。

```
cvs init
```

**例2.2 新規 CVS リポジトリーの初期化**

**~/cvs/** ディレクトリーに空の CVS リポジトリーを作成するには、以下を入力します。

```
~]$ export CVSROOT=~/cvs
~]$ cvs init
```

#### リポジトリーへのデータのインポート

既存のプロジェクトでリビジョン管理を行うように設定するには、プロジェクトの保存されているディレクトリーに移動して、以下のコマンドを実行します。

```
cvs [-d cvs_repository] import [-m "commit message"] module vendor_tag
release_tag
```

*cvs_repository* は CVS リポジトリーへのパス、*module* はプロジェクトのインポート先のサブディレクトリー (例: **project**)、*vendor_tag* と *release_tag* はベンダーとリリースタグを指す点に注意してください。

**例2.3 CVS リポジトリーへのプロジェクトのインポート**

プロジェクトのディレクトリーに以下のコンテンツが含まれていると仮定します。

```
~]$ ls myproject
AUTHORS  doc  INSTALL  LICENSE  Makefile  README  src  TODO
```

また、**~/cvs/** に空の CVS リポジトリーがあるとします。このリポジトリーの**project** に、ベンダータグ **mycompany** と、リリースタグ **init** を付けて、プロジェクトをインポートするには、以下を実行します。

```
myproject]$ export CVSROOT=~/cvs
myproject]$ cvs import -m "Initial import." project mycompany init
N project/Makefile
N project/AUTHORS
N project/LICENSE
N project/TODO
N project/INSTALL...
```

### 2.1.3. 作業コピーのチェックアウト

CVS リポジトリーにあるプロジェクトの作業中のコピーをチェックアウトするには、以下のコマンドを実行します。

```
cvs -d cvs_repository checkout module
```

このコマンドでは、プロジェクトの作業コピーが含まれた *module* と呼ばれる新規ディレクトリーが作成されます。*cvs_repository* は、CVS リポジトリーの URL で、*module* はプロジェクトの保存先のサブディレクトリー (例: **project**) である点に注意してください。また **$CVSROOT** の環境変数は以下のように設定できます。

```
export CVSROOT=cvs_repository
```

次に、**-d** オプションなしで **cvs checkout** コマンドを使用することもできます。

```
cvs checkout module
```

**例2.4 作業コピーのチェックアウト**

CVS リポジトリーに **~/cvs/** があり、このリポジトリーに**project** という名前のモジュールが含まれていると仮定します。このモジュールの作業コピーをチェックアウトするには、以下を入力します。

```
~]$ export CVSROOT=~/cvs
~]$ cvs checkout project
cvs checkout: Updating project
U project/AUTHORS
U project/INSTALL
U project/LICENSE
U project/Makefile
U project/TODO
```

### 2.1.4. ファイルの追加および削除

**ファイルの追加**

CVS リポジトリーへ既存のファイルを追加して、リビジョン管理を行うには、作業コピーのあるディレクトリーに移動して、以下のコマンドを実行します。

```
cvs add file…
```

これにより、CVS リポジトリーへファイルを追加するようにスケジュールします。先に進め、実際にリポジトリーにファイルを追加するには、「変更のコミット」に説明されているように**cvs commit** コマンドを実行します。

**例2.5 CVS リポジトリーへのファイルの追加**

CVS リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  ChangeLog  CVS  doc  INSTALL  LICENSE  Makefile  README  src
TODO
```

このディレクトリーに含まれる **ChangeLog** 以外のファイルとディレクトリーはすべてリビジョンが管理されています。このファイルを CVS リポジトリーに追加するようにスケジュールするには、以下を入力します。

```
project]$ cvs add ChangeLog
cvs add: scheduling file `ChangeLog' for addition
cvs add: use 'cvs commit' to add this file permanently
```

### ファイルの削除

CVS リポジトリーからファイルを削除するには、作業コピーが含まれるディレクトリーに移動して、ローカルでそのファイルを削除しあｍす。

```
rm file…
```

以下のコマンドを使用して、このファイルの削除をスケジュールします。

```
cvs remove file…
```

実際にリポジトリーからファイルを削除するように進めるには、「変更のコミット」の説明のとおりに、**cvs commit** コマンドを実行します。

**例2.6 CVS リポジトリーからのファイルの削除**

CVS リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  ChangeLog  CVS  doc  INSTALL  LICENSE  Makefile  README  src
TODO
```

このディレクトリーのファイルはすべてリビジョン管理されています。**TODO** ファイルを CVS リポジトリーから削除するようにスケジュールするには、以下を入力します。

```
project]$ rm TODO
```

```
project]$ cvs remove TODO
cvs remove: scheduling `TODO' for removal
cvs remove: use 'cvs commit' to remove this file permanently
```

### 2.1.5. 変更の表示

#### ステータスの表示

作業コピーの現在の状況を確認するには、作業コピーが含まれるディレクトリーに移動して、以下のコマンドを実行します。

```
cvs status
```

このコマンドにより、現在のステータス (**Up-to-date**、**Locally Added**、**Locally Removed** または **Locally Modified**) およびリビジョンなど、リビジョン管理が行われるファイルごとに詳細情報が表示されます。ただし、作業コピーの変更内容だけを確認するには、シェルプロンプトで以下を入力して、出力を簡素化することができます。

```
cvs status 2>/dev/null | grep Status: | grep -v Up-to-date
```

**例2.7 作業コピーのステータスの表示**

CVS リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  ChangeLog  CVS  doc  INSTALL  LICENSE  Makefile  README  src
```

CVS リポジトリーへの追加が予定されている、このディレクトリー内のファイルとディレクトリーはすべて (**ChangeLog** 以外の)、すでにリビジョン管理がされています。リビジョン管理がされている **TODO** ファイルは、削除のスケジューリングがされており、この作業コピーはすでに存在しません。最後に **Makefile** にはローカルの変更内容が含まれます。このような作業コピーのステータスを表示するには、以下を入力します。

```
project]$ cvs status 2>/dev/null | grep Status: | grep -v Up-to-date
File: ChangeLog         Status: Locally Added
File: Makefile          Status: Locally Modified
File: no file TODO              Status: Locally Removed
```

#### 差異の表示

作業コピーとチェックアウトしたコンテンツの差異を表示するには、作業コピーが含まれるディレクトリーに移動して、以下のコマンドを実行します。

```
cvs diff [file…]
```

このコマンドにより、作業コピーにあるすべてのファイルへの変更が表示されます。特定のファイルに対する変更のみを表示するには、コマンドラインでそのファイル名を指定します。

**例2.8 作業コピーへの変更の表示**

CVS リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  ChangeLog  CVS  doc  INSTALL  LICENSE  Makefile  README  src
```

このディレクトリー内のすべてのファイルはリビジョン管理されており、**Makefile** にはローカルでの変更が含まれます。これらの変更を表示するには以下を入力します。

```
project]$ cvs diff
cvs diff: Diffing .
cvs diff: ChangeLog is a new entry, no comparison available
Index: Makefile
===================================================================
RCS file: /home/john/cvs/project/Makefile,v
retrieving revision 1.1.1.1
diff -r1.1.1.1 Makefile
156c156
<       -rm -f $(MAN1)
---
>       -rm -f $(MAN1) $(MAN7)
cvs diff: TODO was removed, no comparison available
cvs diff: Diffing doc...
```

### 2.1.6. 変更のコミット

他の人と変更の共有や CVS リポジトリーへの変更のコミットを行うには、作業コピーのあるディレクトリーに移動して、以下のコマンドを実行します。

```
cvs commit [-m "commit message"]
```

コマンドラインにコミットメッセージを指定しない限り、CVS により、ログの記述用に外部テキストエディター (デフォルトでは **vi**) が開きます。どのエディターを起動するか決定する方法については「CVS のインストールおよび設定」を参照してください。

**例2.9 CVS リポジトリーへの変更のコミット**

CVS リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  ChangeLog  CVS  doc  INSTALL  LICENSE  Makefile  README  src
```

作業コピーでは、**ChangeLog** は CVS リポジトリーへの追加がスケジューリングされており、**Makefile** はすでにリビジョン管理がされ、ローカルの変更が含まれます。また、リビジョンが管理されている **TODO** ファイルは削除するようにスケジューリングされており、作業コピーには存在しません。CVS リポジトリーにこれらの変更をコミットするには、以下を入力します。

```
project]$ cvs commit -m "Updated the makefile."
cvs commit: Examining .
cvs commit: Examining doc...
RCS file: /home/john/cvsroot/project/ChangeLog,v
```

```
done
Checking in ChangeLog;
/home/john/cvsroot/project/ChangeLog,v  <--  ChangeLog
initial revision: 1.1
done
Checking in Makefile;
/home/john/cvsroot/project/Makefile,v  <--  Makefile
new revision: 1.2; previous revision: 1.1
done
Removing TODO;
/home/john/cvsroot/project/TODO,v  <--  TODO
new revision: delete; previous revision: 1.1.1.1
done
```

### 2.1.7. 作業コピーの更新

作業コピーを更新して、CVS リポジトリーから最新の変更を取得するには、作業コピーが含まれるディレクトリーに移動して、以下のコマンドを実行します。

```
cvs update
```

**例2.10 作業コピーの更新**

CVS リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  CVS  doc  INSTALL  LICENSE  Makefile  README  src TODO
```

このリポジトリーに **ChangeLog** を追加し、**TODO** ファイルを削除し、さらに **Makefile** に変更を加えたと仮定します。この作業コピーを更新するには、以下を入力します。

```
myproject]$ cvs update
cvs update: Updating .
U ChangeLog
U Makefile
cvs update: TODO is no longer in the repository
cvs update: Updating doc
cvs update: Updating src
```

### 2.1.8. その他のリソース

サポートされる全機能に関する詳しい説明は、本書の対象外となっています。詳細情報は、以下に記載のリソースを参照してください。

#### インストールされているドキュメント

- **cvs(1) – cvs** クライアントプログラムの man ページでは、使用方法に関する詳しい情報が提供されます。

## 2.2. APACHE SUBVERSION (SVN)

通常 **SVN** と略される **Apache Subversion** は、クライアントサーバーで構成される*集中型バージョン管理システム* です。SVN は以前の Concurrent Versions System (CVS) の後継で、同じ開発モデルを保持しており、CVS で頻繁に発生していた問題に対応しています。

### 2.2.1. Subversion のインストールおよび設定

**subversion パッケージのインストール**
**Subversion** は、Red Hat Enterprise Linux 6 では subversion パッケージで提供されます。subversion パッケージとすべての依存関係をお使いのシステムにインストールするには、**root** として、シェルプロンプトで以下のコマンドを入力します。

```
yum install subversion
```

これにより、コマンドライン Subversion クライアントと Subversion サーバー、その他の関連ツールがシステムにインストールされます。

**デフォルトのエディターの設定**
コマンドラインで Subversion を使用する場合に、**svn import** または **svn commit** など、特定のコマンドでは短いログメッセージを記述する必要があります。**svn** クライアントアプリケーションはまず環境変数 **$SVN_EDITOR** のコンテンツを読み込み、次に、より一般的な環境変数**$VISUAL** と **$EDITOR** を読み込み、どのテキストエディターを起動するかを決定します。どちらの変数も指定されていない場合には、エラーが報告されます。

環境変数 **$SVN_EDITOR** の値を永続的に変更する場合は、以下のコマンドを実行します。

```
echo "export SVN_EDITOR=command" >> ~/.bashrc
```

これにより、**export SVN_EDITOR=*command*** の行を **~/.bashrc** に追加します。*command* は、任意のエディターを実行するコマンドに置き換えます (例: **emacs**)。現在の shell セッションで、この変更を有効にするには、シェルプロンプトで以下のコマンドを入力して **~/.bashrc** のコマンドを実行する必要があります。

```
. ~/.bashrc
```

**例2.11 デフォルトのテキストエディターの設定**

Subversion クライアントがテキストエディターとして **Emacs** を使用するように設定するには、以下を入力します。

```
~]$ echo "export SVN_EDITOR=emacs" >> ~/.bashrc
~]$ . ~/.bashrc
```

### 2.2.2. 新規リポジトリーの作成

Subversion *リポジトリー*は、リビジョン管理がされているファイルやディレクトリー、完全な変更履歴や変更者や変更時間の情報などの追加データを保存する中央の場所となっています。一般的な Subversion リポジトリーは、個別のサブディレクトリーで複数のプロジェクトを保存します。公開されている場合には、複数の開発者がサブディレクトリーの *作業コピー*を作成したり変更したりできるだけでなく、リポジトリーに *コミット* することで他の作業者と変更を共有することができます。

### 空のリポジトリーの初期化

任意のディレクトリーに空の Subversion リポジトリーを新規作成するには、以下のコマンドを実行します。

```
svnadmin create path
```

*path* は、リポジトリーの保存先のディレクトリーへの絶対パスを指定する必要がある点を念頭に置いてください (例: **/var/svn/**)。ディレクトリーがない場合には、**svnadmin create** を実行すると作成されます。

**例2.12 新規 Subversion リポジトリーの初期化**

**~/svn/** ディレクトリーに Subversion リポジトリーを作成するには、以下を入力します。

```
~]$ svnadmin create svn
```

### リポジトリーへのデータのインポート

既存のプロジェクトでリビジョン管理を行うように設定するには、以下のコマンドを実行します。

```
svn import local_path svn_repository/remote_path [-m "commit message"]
```

*local_path* は、プロジェクトを保存するディレクトリーへの絶対パスに (現在作業中のディレクトリーを指定するには **.** を使用)、*svn_repository* は、Subversion リポジトリーの URL に、*remote_path* は Subversion リポジトリーのターゲットディレクトリー (例: **project/trunk**) に置き換えます。

**例2.13 Subversion リポジトリーへのプロジェクトのインポート**

プロジェクトのディレクトリーに以下のコンテンツが含まれていると仮定します。

```
~]$ ls myproject
AUTHORS  doc  INSTALL  LICENSE  Makefile  README  src  TODO
```

**~/svn/** (今回の例では **/home/john/svn/**) に空の Subversion リポジトリーがあると仮定します。このリポジトリーの **project/trunk** るプロジェクトをインポートするには、以下を入力します。

```
~]$ svn import myproject file:///home/john/svn/project/trunk -m "Initial
import."
Adding         project/AUTHORS
Adding         project/doc
Adding         project/doc/index.html
Adding         project/INSTALL
Adding         project/src...
```

## 2.2.3. 作業コピーのチェックアウト

Subversion リポジトリーにあるプロジェクトの作業中のコピーをチェックアウトするには、以下のコマンドを実行します。

```
svn checkout svn_repository/remote_path [directory]
```

このコマンドでは、プロジェクトの作業コピーが含まれた *directory* と呼ばれる新規ディレクトリーが作成されます。*svn_repository* は、Subversion リポジトリーの URL で、*remote_path* はプロジェクトの保存先のサブディレクトリーである点に注意してください。

**例2.14 作業コピーのチェックアウト**

**~/svn/** ディレクトリー (今回の例では **/home/john/svn/**) に Subrevision リポジトリーがあり、このリポジトリーの **project/trunk** サブディレクトリーに最新版のプロジェクトが含まれると仮定します。このプロジェクトの作業コピーをチェックアウトするには、以下を入力します。

```
~]$ svn checkout svn:///home/john/svn/project/trunk project
A    project/AUTHORS
A    project/doc
A    project/doc/index.html
A    project/INSTALL
A    project/src...
```

### 2.2.4. ファイルの追加、名前変更、削除

#### ファイルまたはディレクトリーの追加

Subversion リポジトリーへ既存のファイルを追加して、リビジョン管理を行うには、作業コピーのあるディレクトリーに移動して、以下のコマンドを実行します。

```
svn add file…
```

同様に、ディレクトリーとその中にあるファイルをすべて追加するには、以下を入力します。

```
svn add directory…
```

これにより、Subversion リポジトリーへファイルとディレクトリーを追加するようにスケジュールします。先に進め、実際にリポジトリーにこのコンテンツを追加するには、「変更のコミット」に説明されているように **cvs commit** コマンドを実行します。

**例2.15 Subversion リポジトリーへのファイルの追加**

Subversion リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  ChangeLog  doc  INSTALL  LICENSE  Makefile  README  src  TODO
```

このディレクトリーに含まれる **ChangeLog** 以外のファイルとディレクトリーはすべてリビジョンが管理されています。このファイルを Subversion リポジトリーに追加するようにスケジュールするには、以下を入力します。

```
project]$ svn add ChangeLog
A         ChangeLog
```

#### ファイルまたはディレクトリーの名前変更

Subversion リポジトリーにある既存のファイルまたはディレクトリーの名前を変更するには、作業コピーのあるディレクトリーに移動して、以下のコマンドを実行します。

```
svn move old_name new_name
```

これにより、オリジナルのファイルまたはディレクトリーの複製が作成され、この複製の追加がスケジューリングされ、自動的にオリジナルのファイルまたはディレクトリーが削除されます。先に進め、Subversion のリポジトリーのコンテンツの名前を実際に変更するには、「変更のコミット」の説明のとおりに **svn commit** コマンドを実行します、

**例2.16 Subversion リポジトリーのファイル名の変更**

Subversion リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  ChangeLog  doc  INSTALL  LICENSE  Makefile  README  src  TODO
```

このディレクトリー内のファイルはすべて、リビジョンの管理が行われています。**LICENSE** ファイルの名前を **COPYING** に変更するには、以下を入力します。

```
project]$ svn move LICENSE COPYING
A         COPYING
D         LICENSE
```

**svn move** は自動的に、作業コピーのファイルの名前も変更する点に注意してください。

```
project]$ ls
AUTHORS  ChangeLog  COPYING  doc  INSTALL  Makefile  README  src TODO
```

### ファイルまたはディレクトリーの削除

Subversion リポジトリーからファイルを削除するには、作業コピーのあるディレクトリーに移動して、以下のコマンドを実行します。

```
svn delete file…
```

同様に、ディレクトリーとその中にあるファイルをすべて削除するには、以下を入力します。

```
svn delete directory…
```

これにより、Subversion リポジトリーからファイルとディレクトリーを削除するようにスケジュールします。先に進め、実際にリポジトリーにこのコンテンツを削除するには、「変更のコミット」に説明されているように **cvs commit** コマンドを実行します。

**例2.17 Subversion リポジトリーからのファイルの削除**

Subversion リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  ChangeLog  COPYING  doc  INSTALL  Makefile  README  src TODO
```

このディレクトリーのファイルはすべてリビジョン管理されています。**TODO** ファイルを SVN リポジトリーから削除するようにスケジュールするには、以下を入力します。

```
project]$ svn delete TODO
D         TODO
```

**svn delete** は自動的に、作業コピーのファイルを削除する点に注意してください。

```
project]$ ls
AUTHORS  ChangeLog  COPYING  doc  INSTALL  Makefile  README  src
```

### 2.2.5. 変更の表示

#### ステータスの表示

作業コピーの現在の状況を確認するには、作業コピーが含まれるディレクトリーに移動して、以下のコマンドを実行します。

```
svn status
```

This displays information about all changes to the working copy (**A** for a file that is scheduled for addition, **D** for a file that is scheduled for removal, **M** for a file that contains local changes, **C** for a file with unresolved conflicts, **?** for a file that is not under revision control).

**例2.18 作業コピーのステータスの表示**

Subversion リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  ChangeLog  COPYING  doc  INSTALL  Makefile  README  src
```

Subversion リポジトリーへの追加が予定されている、このディレクトリー内のファイルとディレクトリーはすべて (**ChangeLog** 以外の)、すでにリビジョン管理がされています。リビジョン管理がされている **TODO** ファイルは、削除のスケジューリングがされており、この作業コピーはすでに存在しません。**LICENSE** ファイルの名前は **COPYING** に変更され、**Makefile** にはローカルの変更内容が含まれます。このような作業コピーのステータスを表示するには、以下を入力します。

```
project]$ svn status
D       LICENSE
D       TODO
A       ChangeLog
A  +    COPYING
M       Makefile
```

#### 差異の表示

作業コピーとチェックアウトしたコンテンツの差異を表示するには、作業コピーが含まれるディレクトリーに移動して、以下のコマンドを実行します。

```
svn diff [file…]
```

このコマンドにより、作業コピーにあるすべてのファイルへの変更が表示されます。特定のファイルに対する変更のみを表示するには、コマンドラインでそのファイル名を指定します。

**例2.19 作業コピーへの変更の表示**

Subversion リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  ChangeLog  COPYING  CVS  doc  INSTALL  Makefile  README  src
```

このディレクトリー内のすべてのファイルはリビジョン管理されており、**Makefile** にはローカルでの変更が含まれます。これらの変更を表示するには以下を入力します。

```
project]$ svn diff Makefile
Index: Makefile
===================================================================
--- Makefile    (revision 1)
+++ Makefile    (working copy)
@@ -153,7 +153,7 @@
      -rmdir $(man1dir)

clean:
-       -rm -f $(MAN1)
+       -rm -f $(MAN1) $(MAN7)

%.1: %.pl
      $(POD2MAN) --section=1 --release="Version $(VERSION)" \
```

### 2.2.6. 変更のコミット

他の人と変更の共有や Subversion リポジトリーへの変更のコミットを行うには、作業コピーのあるディレクトリーに移動して、以下のコマンドを実行します。

```
svn commit [-m "commit message"]
```

コマンドラインにコミットメッセージを指定しない限り、Subversion により、ログの記述用に外部テキストエディターが開きます。どのエディターを起動するか決定する方法については「Subversion のインストールおよび設定」を参照してください。

**例2.20 Subversion リポジトリーへの変更のコミット**

Subversion リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  ChangeLog  COPYING  doc  INSTALL  Makefile  README  src
```

作業コピーでは、**ChangeLog** は Subversion リポジトリーへの追加がスケジューリングされており、**Makefile** はすでにリビジョン管理がされ、ローカルの変更が含まれます。リビジョンが管理されている **TODO** ファイルは削除するようにスケジューリングされており、作業コピーには存在し

ません。さらに、**LICENSE** ファイルの名前は **COPYING** に変更されました。Subversion リポジトリーにこれらの変更をコミットするには、以下を入力します。

```
project]$ svn commit -m "Updated the makefile."
Adding         COPYING
Adding         ChangeLog
Deleting       LICENSE
Sending        Makefile
Deleting       TODO
Transmitting file data ..
Committed revision 2.
```

### 2.2.7. 作業コピーの更新

作業コピーを更新して、Subversion リポジトリーから最新の変更を取得するには、作業コピーが含まれるディレクトリーに移動して、以下のコマンドを実行します。

```
svn update
```

**例2.21 作業コピーの更新**

Subversion リポジトリーの作業コピーが含まれるディレクトリーに以下のコンテンツが含まれていると仮定します。

```
project]$ ls
AUTHORS  doc  INSTALL  LICENSE  Makefile  README  src TODO
```

このリポジトリーに対して **ChangeLog** の追加と **TODO** ファイルの削除を行い、**LICENSE** から **COPYING** に名前を変更したうえに、**Makefile** に変更を加えたと仮定します。この作業コピーを更新するには、以下を入力します。

```
myproject]$ svn update
D    LICENSE
D    TODO
A    COPYING
A    Changelog
M    Makefile
Updated to revision 2.
```

### 2.2.8. その他のリソース

サポートされる全機能に関する詳しい説明は、本書の対象外となっています。詳細情報は、以下に記載のリソースを参照してください。

#### インストールされているドキュメント

- **svn help** – **svn help** コマンドにより、詳しい **svn** の使用方法が出力されます。
- **svnadmin help** – **svnadmin help** コマンドにより **svnadmin** の使用方法の詳細情報が出力されます。

### オンラインのドキュメント

- Subversion によるバージョン管理 – Subversion の公式ウェブサイトで、『Subversion によるバージョン管理』 マニュアルを参照してください。このマニュアルでは、Subversion、管理、使用方法について詳しく説明されています。

## 2.3. GIT

**Git**は、ピアツーピアアーキテクチャーで構成される**分散型リビジョン管理システム**です。クライアントサーバーモデルの集中型バージョン管理システムとは異なり、**Git** では **Git** リポジトリーの各作業コピーが、完全な改訂履歴を利用することで、作業コピーと全く同じ状態となるようにします。これにより、正式なリポジトリーに変更をプッシュするパーミッションなしにプロジェクトの作業および参加が可能となるだけではなく、ネットワーク接続のない状態でも作業することができます。

### 2.3.1. Git のインストールおよび設定

#### git パッケージのインストール

**Git** は、Red Hat Enterprise Linux 6 では **git** パッケージで提供されます。**cvs** パッケージとすべての依存関係をお使いのシステムにインストールするには、**root** として、シェルプロンプトで以下のコマンドを入力します。

```
~]# yum install git
```

#### デフォルトのテキストエディターの設定

**git commit** などの **Git** コマンドでは、外部テキストエディターで短いメッセージを記述するか、変更を加える必要があります。**Git** は、**GIT_EDITOR** の環境変数、**core.editor** の設定オプション、**VISUAL** 環境変数の値をこの順番に読み込み、最後に **EDITOR** の環境変数の値の読み込みを使用してどのテキストエディターを起動するか決定します。これらのオプションや変数が指定されていない場合には、**git** コマンドにより **vi** が起動されます。

**core.editor** の設定オプションの値を変更して別のテキストエディターを指定するには、シェルプロンプトで以下を入力します。

```
git config --global core.editor command
```

*command* は、選択したテキストエディターの起動に使用するコマンドに置き換えます。

**例2.22 デフォルトのテキストエディターの設定**

**Git** がデフォルトのテキストエディターとして **vim** を使用するように設定するには、シェルプロンプトで以下を入力します。

```
~]$ git config --global core.editor vim
```

#### ユーザー情報の設定

**Git** では、各コミット (またはリビジョン) は、そのコミットを行なった人のフルネームとメールに関連付けられます。デフォルトでは、**Git** はユーザー名とホスト名をベースにした ID を使用します。

**Git** コミットに関連付けられたフルネームを変更するには、シェルプロンプトで以下を入力します。

```
git config --global user.name "full name"
```

**Git** コミットで関連付けられたメールアドレスを変更するには、以下を入力します。

```
git config --global user.email "email_address"
```

**例2.23 ユーザー情報の設定**

**Git** で **John Doe** をフルネームとして、**john@example.com** をメールアドレスとして使用するように設定するには、以下をシェルプロンプトに入力します。

```
~]$ git config --global user.name "John Doe"
~]$ git config --global user.email "john@example.com"
```

### 2.3.2. 新規リポジトリーの作成

*リポジトリー*とは、**Git** はリビジョン管理がされている全ファイルと、完全な変更履歴や変更日時や変更者の情報などそのファイルに関連する追加データを保存する場所のことです。**Subversion** または **CVS** などの集中型のリビジョン管理システムとは異なり、通常 **Git** リポジトリーと *作業ディレクトリー*は同じです。また、一般的な **Git** リポジトリーでは、単一のプロジェクトだけを保存し、公開されている場合には、誰でも、完全な改訂履歴を使用してクローンを作成することができます。

**空のリポジトリーの初期化**
空の **Git** リポジトリーを新規作成するには、シェルリポジトリーで、保存する先のディレクトリーに移動して、以下を入力します。

```
git init
```

このコマンドにより、リポジトリー情報がすべて保存されている **.git** と呼ばれる隠しディレクトリーが作成されます。

**リポジトリーへのデータのインポート**
既存のプロジェクトでリビジョン管理を行うように設定するには、プロジェクトのディレクトリーに **Git** リポジトリーを作成して、以下のコマンドを実行します。

```
git add .
```

このコマンドは、現在の作業ディレクトリーにある全ファイルとディレクトリーに対して、**Git** リポジトリーへの追加の準備ができているとマークします。実際にリポジトリーへこのコンテンツを追加するように進めるには、シェルプロンプトで以下を入力して変更をコミットします。

```
git commit [-m "commit message"]
```

*commit message*は、リビジョンに関する短い説明に置き換えてください。**-m** オプションを省略すると、このコマンドにより、外部テキストエディターでコミットのメッセージを入力できるようになります。デフォルトのテキストエディターの設定方法に関する情報は 「デフォルトのテキストエディターの設定」 を参照してください。

### 2.3.3. 既存のリポジトリーのクローン作成

既存の **Git** リポジトリーのクローンを作成するには、シェルプロンプトで以下を入力します。

```
git clone git_repository [directory]
```

*git_repository* は、クローンする **Git** リポジトリーへのパスまたは URL に、*directory* はクローンの保存先のリポジトリーに置き換えます。

### 2.3.4. ファイルの追加、名前変更、削除

#### ファイルおよびディレクトリーの追加

既存のファイルを **Git** リポジトリーに追加して、リビジョン管理の設定を行うには、シェルプロンプトでローカルの **Git** リポジトリーがあるディレクトリーに移動して、以下を入力します。

```
git add file...
```

*file* は、追加するファイルに置き換えます。このコマンドは、選択したファイルに対して**Git** リポジトリーへの追加準備ができているとマークします。同様に、特定のディレクトリーに保存されている全ファイルを **Git** リポジトリーに追加するには、以下を入力します。

```
git add directory...
```

*directory* は、追加するディレクトリーに置き換えます。このコマンドは、選択したディレクトリーの中にある全ファイルに対して、**Git** リポジトリーに追加する準備ができているとマークします。

このコンテンツをリポジトリーに実際に追加するように進めるには、「変更のコミット」の説明のように変更をコミットします。

#### ファイルとディレクトリーの名前の変更

**Git** リポジトリーにある既存のファイルまたはディレクトリーの名前を変更するには、シェルプロンプトでローカルの **Git** リポジトリーがあるディレクトリーに移動して、以下を入力します。

```
git mv old_name new_name
```

*old_name* は、ファイルまたはディレクトリーの現在の名前に、*new_name* は新しい名前に変更します。このコマンドは、選択したファイルまたはディレクトリーの名前を変更し、**Git** リポジトリーで名前変更の準備ができているとマークします。

実際にリポジトリーのコンテンツの名前を変更するように進めるには、「変更のコミット」の説明のように変更をコミットします。

#### ファイルおよびディレクトリーの削除

**Git** リポジトリーから既存のファイルを削除するには、シェルプロンプトでローカルの**Git** リポジトリーがあるディレクトリーに移動して、以下を入力します。

```
git rm file...
```

*file* は、削除するファイルに置き換えます。このコマンドは、選択した全ファイルに対して**Git** リポジトリーへの削除の準備ができているとマークします。同様に、特定のディレクトリーに保存されている全ファイルを **Git** リポジトリーから削除するには、以下を入力します。

```
git rm -r directory...
```

*directory* は、削除するディレクトリーに置き換えます。このコマンドは、選択したディレクトリーをすべて削除して、それらの対して **Git** リポジトリーから削除する準備ができているとマークします。

リポジトリーからこのコンテンツを実際に削除するように進めるには、「変更のコミット」の説明のように変更をコミットします。

### 2.3.5. 変更の表示

#### 現在のステータスの表示

ローカルの **Git** リポジトリーにおける現在のステータスを確認するには、シェルプロンプトで、リポジトリーのあるディレクトリーに移動して、以下のコマンドを入力します。

```
git status
```

このコマンドにより、リポジトリー内のコミット前の全変更に関する情報 (**new file**、**renamed**、**deleted** または **modified**) を表示して、次回のコミットの差異にどの変更が適用されるかが表示されます。変更のコミット方法に関する情報は、「変更のコミット」を参照してください。

#### 差異の表示

**Git** リポジトリーの全変更を表示するには、シェルプロンプトでリポジトリーのあるディレクトリーに移動して以下を入力します。

```
git diff
```

このコマンドにより、リポジトリー内のファイルと最新のリビジョンの間の変更が表示されます。特定のファイルの変更のみを表示するには、コマンドラインで以下のようにファイルの名前を指定します。

```
git diff file...
```

*file* は、表示するファイルに置き換えます。

### 2.3.6. 変更のコミット

**Git** リポジトリーに変更を適用して、新規リビジョンを作成するには、シェルプロンプトで、リポジトリーのあるディレクトリーに移動して、以下のコマンドを入力します。

```
git commit [-m "commit message"]
```

*commit message* は、リビジョンの短い説明に置き換えます。このコマンドは、コミットの準備ができていると明示的にマークされているファイルの変更をすべてコミットします。リビジョン管理がされている全ファイルの変更をコミットするには、以下のように **-a** のコマンドラインオプションを追加します。

```
git commit -a [-m "commit message"]
```

**-m** オプションを省略すると、このコマンドにより、外部テキストエディターでコミットのメッセージを入力できるようになる点に注意してください。デフォルトのテキストエディターの設定方法に関する情報は 「デフォルトのテキストエディターの設定」を参照してください。

### 2.3.7. 変更の共有

CVS または Subversion などの集中型バージョン管理システムと違って、プロジェクトのコントリビューターは **Git** で作業する場合には、通常、単一の中央リポジトリーに変更を加えるわけではなく、ローカルのリポジトリーのクローンを作成して公開するか、パッチとしてメールで変更を送信します。

#### 公開リポジトリーへの変更のプッシュ

公開されている **Git** リポジトリーに変更をプッシュするには、シェルプロンプトで、ローカルリポジトリーのあるディレクトリーに移動して、以下を入力します。

```
git push remote_repository
```

*remote_repository* は、変更をプッシュする先のリモートのリポジトリー名に置き換えます。ローカルのコピーをクローンした元のリポジトリーは自動的に **origin** という名前が指定される点にご注意ください。

**個別のコミットからのパッチ作成**
コミットからパッチを作成するには、シェルプロンプトでローカルの **Git** リポジトリーのあるディレクトリーに移動して、以下を入力します。

```
git format-patch remote_repository
```

*remote_repository* は、ローカルのコピーを作成した元のリモートリポジトリーの名前に置き換えます。これにより、リモートリポジトリーにない各コミットのパッチが作成されます。

### 2.3.8. リポジトリーの更新

**Git** リポジトリーのローカルコピーを更新して、リモートのリポジトリーからの最新の変更を取得するには、シェルプロンプトでローカルの **Git** リポジトリーのあるディレクトリーに移動して、以下を入力します。

```
git fetch remote_repository
```

*remote_repository* は、リモートのリポジトリー名に置き換えます。このコマンドにより、リモートリポジトリーの現在のステータスに関する情報が取得され、ローカルコピーに変更を加える前に、変更内容を確認することができます。ローカルの **Git** リポジトリーにある変更のマージを続行するには、以下を入力します。

```
git merge remote_repository
```

または、以下のコマンドを使用して、これらの両ステップを同時に実行します。

```
git pull remote_repository
```

### 2.3.9. その他のリソース

**Git** およびその機能に関する詳しい説明は、本書の対象外となっています。このリビジョン管理システムに関する詳しい情報は、以下に記載のリソースを参照してください。

**インストールされているドキュメント**

- **gittutorial(7) – gittutorial** と呼ばれる man ページには **Git** とその使用方法に関する簡単な説明が記載されています。
- **gittutorial-2(7) – gittutorial-2** と呼ばれる man ページには **Git** とその使用方法に関する簡単な説明 (パート 2) が記載されています。
- 『Git ユーザーマニュアル』 – **Git** の HTML ドキュメントは **/usr/share/doc/git-1.7.1/user-manual.html** にあります。

**オンラインのドキュメント**

- Pro Git – 『Pro Git』 ブックのオンライン版は、**Git**、コンセプト、使用方法が詳しく記載しています。

# 第3章 ライブラリーおよびランタイムのサポート

Red Hat Enterprise Linux 6 は、実績のある業界仕様のツールを使用して幅広いプログラム言語でカスタムアプリケーションの開発をサポートします。本章では、Red Hat Enterprise Linux 6 で提供されているランタイムサポートのライブラリーについて説明します。

## 3.1. バージョン情報

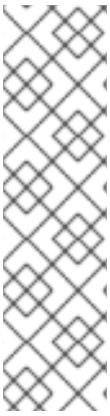
以下の表では、Red Hat Enterprise Linux 6、Red Hat Enterprise Linux 5、および Red Hat Enterprise Linux 4 の間でのサポート対象のプログラミング言語におけるランタイムサポートパッケージのバージョン情報を比較しています。

この表は完全なリストではなく、標準言語ランタイムの概略と Red Hat Enterprise Linux 6 で開発されたソフトウェアの主要依存関係を示しています。

**表3.1 言語およびランタイムライブラリーのバージョン**

| パッケージ名 | Red Hat Enterprise 6 | Red Hat Enterprise 5 | Red Hat Enterprise 4 |
|---|---|---|---|
| **glibc** | 2.12 | 2.5 | 2.3 |
| **libstdc++** | 4.4 | 4.1 | 3.4 |
| **boost** | 1.41 | 1.33 | 1.32 |
| **java** | 1.5 (IBM)、1.6 (IBM、OpenJDK、Oracle Java) | 1.4、1.5、1.6 | 1.4 |
| **python** | 2.6 | 2.4 | 2.3 |
| **php** | 5.3 | 5.1 | 4.3 |
| **ruby** | 1.8 | 1.8 | 1.8 |
| **httpd** | 2.2 | 2.2 | 2.0 |
| **postgresql** | 8.4 | 8.1 | 7.4 |
| **mysql** | 5.1 | 5.0 | 4.1 |
| **nss** | 3.12 | 3.12 | 3.12 |
| **openssl** | 1.0.0 | 0.9.8e | 0.9.7a |
| **libX11** | 1.3 | 1.0 | |
| **firefox** | 3.6 | 3.6 | 3.6 |
| **kdebase** | 4.3 | 3.5 | 3.3 |

| パッケージ名 | Red Hat Enterprise 6 | Red Hat Enterprise 5 | Red Hat Enterprise 4 |
|---|---|---|---|
| gtk2 | 2.18 | 2.10 | 2.04 |



注記

**compat-glibc** RPM は Red Hat Enterprise Linux 6 に含まれていますが、ランタイムパッケージではないので、何かの実行に必要というわけではありません。これはヘッダーファイルやリンク用のダミーライブラリーを含む、単なる開発パッケージです。これを使用することで、Red Hat Enterprise Linux の古いバージョンでコンパイルおよびリンクのパッケージを実行することが可能になります (それらのヘッダーおよびライブラリーに **compat-gcc-*** を使用)。**rpm -qpi compat-glibc-*** を実行すると、このパッケージの使用方法に関する情報が提供されます。

**compat-glib** の詳細については、「**compat-glibc**」 を参照してください。

## 3.2. 互換性

互換性により、コンピューターの稼働環境が異なるインスタンスにおいてバイナリーオブジェクトとソースコードの移植性があるかどうかが分かります。公式には、Red Hat は現行バージョンとそれ以前の 2 つのバージョンをサポートします。つまり、Red Hat Enterprise Linux 4 および Red Hat Enterprise Linux 5 で作成されたアプリケーションは、それが Red Hat ガイドラインに準拠している限り (例えば、ホワイトリスト化された記号を使用)、Red Hat Enterprise Linux 6 で稼働します。

Red Hat は、エンタープライズプラットフォームとしてお客様がそのアプリケーションの長期的開発に依存していることを理解しています。このため、互換性ライブラリの助けを用いて C/C++ ライブラリに対して構築されたアプリケーションは、10 年間サポートが継続されます。

互換性には以下の 2 種類があります。

**ソースの互換性**

ソースの互換性は、コードが稼働環境の異なるインスタンスにおいて一貫性があり予測可能な方法でコンパイルおよび実行することを指定します。この種類の互換性は、特定の Application Programming Interfaces (API) との適合性で定義されます。

**バイナリーの互換性**

バイナリーの互換性は、実行可能ファイルおよび *Dynamic Shared Objects* (動的共有オブジェクト: DSO) の形式でコンパイル済みバイナリーが稼働環境の異なるインスタンスで正確に実行することを指定します。この種類の互換性は、特定の Application Binary Interfaces (ABI) との適合性で定義されます。

この点と、コアおよび非コアライブラリー間の全レベルでの互換性についての詳細情報は、https://access.redhat.com/support/policy/updates/errata/から Red Hat Enterprise Linux のサポート対象リリースを参照してください。また、Red Hat Enterprise Linux の互換性ポリシー全般は、https://access.redhat.com/solutions/5154 を参照してください。

### 3.2.1. 静的リンク

静的リンクは、すべての Red Hat Enterprise Linux リリースで使用しないことが強く推奨されます。静的リンクは解決するよりもはるかに多くの問題を生み出し、ぜひとも避けるべきです。

静的リンクの主な欠点は、それが構築されたシステム上での動作のみが保証されており、それも glibc もしくは libstdc++ (C++ の場合) の次回リリースまでです。静的構築では、前方もしくは後方互換性がありません。さらに、以降のライブラリーの更新におけるセキュリティ修正は、影響のある静的にリンクされた実行可能ファイルが再度リンク化されない限り、利用可能にはなりません。

静的リンクを避ける他の理由は以下のとおりです。

- メモリフットプリントが大きい。
- アプリケーションの起動時間が長い。
- 静的リンクだと glibc 機能が限定される。
- ロードアドレスのランダム化などのセキュリティ対策が使用できない。
- glibc 外の共有オブジェクトの動的ローディングがサポートされていない。

静的リンクを避けるさらなる理由については、Static Linking Considered Harmful を参照してください。

## 3.3. ライブラリーおよびランタイムの詳細

### 3.3.1. compat-glibc

**compat-glibc** は、以前のバージョンの Red Hat Enterprise Linux からの共有静的ライブラリーのサブセットを提供します。Red Hat Enterprise Linux 6 では、以下のライブラリーが提供されます。

- **libanl**
- **libcidn**
- **libcrypt**
- **libc**
- **libdl**
- **libm**
- **libnsl**
- **libpthread**
- **libresolv**
- **librt**
- **libthread_db**
- **libutil**

このライブラリーセットにより、アプリケーションが上記のライブラリーのみを使用するという条件で、開発者はRed Hat Enterprise Linux 6 で Red Hat Enterprise Linux 5 のアプリケーションを作成することができます。これを行うには、以下のコマンドを実行してください。

```
# gcc -fgnu89-inline -I /usr/lib/x86_64-redhat-linux5E/include -B
/usr/lib/x86_64-redhat-linux5E/lib64/ -lc_nonshared
```

### 3.3.2. GNU C++ 標準ライブラリー

**libstdc++** パッケージには GNU C++ 標準ライブラリーが含まれています。これは ISO 14882 標準 C++ ライブラリーを実装するための進行中のプロジェクトです。

**libstdc++** パッケージをインストールすると、リンクの依存関係が十分に満たされます (つまり、共有ライブラリーファイルのみ)。C++ 開発で使用可能なライブラリーおよびヘッダーファイルのすべてを活用するには、**libstdc++-devel** もインストールする必要があります。**libstdc++-devel** には、GNU 固有の標準テンプレートライブラリー (STL) の実装も含まれています。

Red Hat Enterprise Linux 4、5、6 では、C++ 言語およびランタイムの実装は安定しているため、**libstdc++** に互換性ライブラリーは必要ありません。ただし、Red Hat Enterprise Linux 2 および 3 では、これは該当しません。Red Hat Enterprise Linux 2 では、**compat-libstdc++-296** のインストールが必要になります。Red Hat Enterprise Linux 3 では、**compat-libstdc++-33** のインストールが必要になります。どちらもデフォルトではインストールされていないので、別個に追加する必要があります。

#### 3.3.2.1. GNU C++ 標準ライブラリー更新

Red Hat Enterprise Linux 6 バージョンの GNU C++ 標準ライブラリーでは、Red Hat Enterprise Linux 5 バージョンと比べて以下の機能が向上されています。

- ISO C++ TR1 要素でサポートが強化。具体的には以下の通りです。
  - **<tr1/array>**
  - **<tr1/complex>**
  - **<tr1/memory>**
  - **<tr1/functional>**
  - **<tr1/random>**
  - **<tr1/regex>**
  - **<tr1/tuple>**
  - **<tr1/type_traits>**
  - **<tr1/unordered_map>**
  - **<tr1/unordered_set>**
  - **<tr1/utility>**
  - **<tr1/cmath>**
- 今後の ISO C++ 標準、C++0x の要素のサポート強化。以下の要素が含まれます。
  - **<array>**

- **<chrono>**
- **<condition_variable>**
- **<forward_list>**
- **<functional>**
- **<initalizer_list>**
- **<mutex>**
- **<random>**
- **<ratio>**
- **<regex>**
- **<system_error>**
- **<thread>**
- **<tuple>**
- **<type_traits>**
- **<unordered_map>**
- **<unordered_set>**

- **-fvisibility** コマンドのサポート強化。
- 以下の拡張子を追加。
  - **__gnu_cxx::typelist**
  - **__gnu_cxx::throw_allocator**

Red Hat Enterprise Linux 6 の **libstdc++** の更新に関する詳細情報は、以下のドキュメントの 『C++ Runtime Library』 セクションを参照してください。

- 『GCC 4.2 Release Series Changes, New Features, and Fixes 』: http://gcc.gnu.org/gcc-4.2/changes.html
- 『GCC 4.3 Release Series Changes, New Features, and Fixes 』: http://gcc.gnu.org/gcc-4.3/changes.html
- 『GCC 4.4 Release Series Changes, New Features, and Fixes 』: http://gcc.gnu.org/gcc-4.4/changes.html

#### 3.3.2.2. GNU C++ 標準ライブラリードキュメント

ライブラリーコンポーネントの **man** を使用するには、**libstdc++-docs** パッケージをインストールします。これで、ライブラリーが提供するほとんどすべてのリソースの **man** ページ情報が提供されます。たとえば、**vector** コンテナについての情報を表示するには、その完全修飾コンポーネント名を使用します。

**man std::vector**

これで以下の情報が表示されます (一部省略)

```
std::vector(3)
std::vector(3)

NAME
       std::vector -

       A standard container which offers fixed time access to individual
       elements in any order.

SYNOPSIS
       Inherits std::_Vector_base< _Tp, _Alloc >.

   Public Types
       typedef _Alloc allocator_type
       typedef __gnu_cxx::__normal_iterator< const_pointer, vector >
           const_iterator
       typedef _Tp_alloc_type::const_pointer const_pointer
       typedef _Tp_alloc_type::const_reference const_reference
       typedef std::reverse_iterator< const_iterator >
```

**libstdc++-docs** パッケージは、以下のディレクトリーで HTML 形式のマニュアルおよび参照情報を提供します。

**file:///usr/share/doc/libstdc++-docs-*version*/html/spine.html**

libstdc++ 開発のメインサイトは gcc.gnu.org です。

### 3.3.3. Boost

**boost** パッケージには、ピア・レビューされた多くの無償の移植可能な C++ ソースライブラリーがあります。これらのライブラリーは、移植可能なファイルシステムおよび時間/日付抽象化、シリアル化、ユニットテスト、スレッド作成およびマルチプロセス同期化、解析、グラフ化、正規表現の操作など、その他多くのタスクに適しています。

**boost** パッケージのインストールは、リンクの依存関係を十分に満たします (つまり、共有ライブラリーファイルのみ)。C++ 開発で使用可能なライブラリーおよびヘッダーファイルのすべてを活用するには、**boost-devel** もインストールする必要があります。

**boost** は実際にはメタパッケージで、多くのライブラリーのサブパッケージが含まれています。これらのサブパッケージは個別にインストールして、より詳細なパッケージ間の依存関係の追跡を提供することもできます。メタパッケージには、以下のサブパッケージすべてが含まれます。

- **boost-date-time**
- **boost-filesystem**
- **boost-graph**
- **boost-iostreams**
- **boost-math**

- **boost-program-options**
- **boost-python**
- **boost-regex**
- **boost-serialization**
- **boost-signals**
- **boost-system**
- **boost-test**
- **boost-thread**
- **boost-wave**

静的リンク用のパッケージまたは基礎的 Message Passing Interface (MPI) サポートに依存するパッケージは、メタパッケージに含まれません。

MPI サポートは 2 つの形式で提供されます: ひとつはデフォルトの Open MPI 実装で [1]、もうひとつは代替の MPICH2 実装です。使用する基礎的 MPI ライブラリーはユーザーが決定でき、特定のハードウェア詳細とユーザーの好みによります。インストール済みファイルは一意のディレクトリーの場所にあることから、これらのパッケージは並行してインストールできることに注意してください。

Open MPI:

- **boost-openmpi**
- **boost-openmpi-devel**
- **boost-graph-openmpi**
- **boost-openmpi-python**

MPICH2:

- **boost-mpich2**
- **boost-mpich2-devel**
- **boost-graph-mpich2**
- **boost-mpich2-python**

静的リンクが避けられない場合は、**boost-static** パッケージが必要な静的ライブラリーをインストールします。スレッド対応と単一スレッドの両方のライブラリーが提供されます。

#### 3.3.3.1. Boost 更新

Red Hat Enterprise Linux 6 バージョンの Boost 機能には、多くのパッケージ関連の改善と新機能が含まれています。

**boost** パッケージのいくつかの側面が変更されています。上記のように、モノリシックな**boost** パッケージは、より小型の別個のサブパッケージで増強されてきました。これでユーザーは依存関係の管理

がより広くできるようになり、Boost を使用するカスタムアプリケーションをパッケージする際はより小型のバイナリーパッケージが可能になっています。

さらに、全ライブラリーのシングルスレッドとマルチスレッドバージョンがパッケージ化されています。マルチスレッドバージョンには、通常の Boost 慣習の通りに **mt** 接尾辞が含まれています。

Boost には以下の新たなライブラリー機能もあります。

- Foreach
- Statechart
- TR1
- Typeof
- Xpressive
- Asio
- Bitmap
- Circular Buffer
- Function Types
- Fusion
- GIL
- Interprocess
- Intrusive
- Math/Special Functions
- Math/Statistical Distributions
- MPI
- System
- Accumulators
- Exception
- Units
- Unordered
- Proto
- Flyweight
- Scope Exit
- Swap

- Signals2
- Property Tree

既存ライブラリーの多くは改善されバグ修正されているか、強化されています。

#### 3.3.3.2. Boost ドキュメント

**boost-doc** パッケージは、以下のディレクトリーで HTML 形式のマニュアルおよび参照情報を提供します。

**file:///usr/share/doc/boost-doc-*version*/index.html**

Boost 開発のメインサイトは boost.org です。

### 3.3.4. Qt

**qt** パッケージは、GUI プログラム開発に使用される Qt (「cute」と発音) クロスプラットフォームアプリケーション開発フレームワークを提供します。人気のあるウィジェットツールキットというだけでなく、Qt はコンソールツールやサーバーといった非 GUI プログラムの開発にも使用されます。Qt は、Google Earth や KDE、Opera、OPIE、VoxOx、Skype、VLC メディアプレーヤー、VirtualBox といったすぐれたプロジェクトの開発に使用されました。これは、Nokiaの Qt 開発フレームワーク部門にが開発しました。この部門は、Qt の元々の作成者であるノルウェーの会社 Trolltech を Nokia が 2008 年 6 月 17 日に買収した後に組織されました。

Qt は標準 C++ を使用しますが、*Meta Object Compiler*(MOC) と呼ばれる特別なプリプロセッサを大幅に使用して言語を豊かなものにします。Qt は言語バインディングで他のプログラミング言語でも使用できます。すべての主要プラットフォームで作動し、広範囲の国際的サポートがあります。非 GUI Qt 機能には、SQL データベースアクセス、XML 解析、スレッド管理、ネットワークサポート、ファイル処理用の統一のクロスプラットフォーム API が含まれます。

Qt は GNU 一般公衆ライセンス (GPL) の下で配布される無償かつオープンソースのソフトウェアです。Qt の Red Hat Enterprise Linux 6 バージョンは、GCC C++ コンパイラーおよび Visual Studio スイートを含む幅広いコンパイラーをサポートします。

#### 3.3.4.1. Qt 更新

Red Hat Enterprise Linux 6 バージョンの Qt で改善された機能は以下のとおりです。

- 高度なユーザーエクスペリエンス
  - **Advanced Graphics Effects:** 不透明度やドロップシャドウ、ぼかし、彩色、その他の類似の効果のオプション
  - **Animation and State Machine:** 複雑なコード管理の混乱なしにシンプルまたは複雑なアニメーションを作成
  - ジェスチャーおよびマルチタッチのサポート
- 新たなプラットフォームのサポート
  - Windows 7、Mac OSX 10.6、およびその他のデスクトッププラットフォームをサポート
  - モバイル開発用の新たなサポート; Qt は、予定されている Maemo 6 プラットフォームに最適化されており、まもなく Maemo 5 にポートされます。さらに、Qt は今では S60 フレームワーク向けの統合で Symbian プラットフォームをサポートしています。

    - QNX や VxWorks などのリアルタイムのオペレーティングシステム向けの新たなサポート

- (他のレンダリング更新に加えて) ハードウェアアクセラレータによるレンダリングの新たなサポートを加えたことによるパフォーマンスの改善。
- クロスプラットフォーム IDE の更新

Red Hat Enterprise Linux 6 に含まれる Qt の更新に関する詳細は、以下のリンクを参照してください。

- http://doc.qt.nokia.com/4.6/qt4-6-intro.html
- http://doc.qt.nokia.com/4.6/qt4-intro.html

#### 3.3.4.2. Qt Creator

**Qt Creator** は、Qt 開発者の要件に合わせたクロスプラットフォーム IDE です。以下のグラフィカルツールが含まれます。

- 高度な C++ コードエディター
- 統合 GUI レイアウトおよびフォームデザイナー
- プロジェクトおよびビルド管理ツール
- コンテキストを感知する統合ヘルプシステム
- 視覚デバッガー
- ラピッドコードナビゲーションツール

#### 3.3.4.3. Qt ライブラリードキュメント

**qt-doc** パフォーマンスは、**/usr/share/doc/qt4/html/** にある HTML マニュアルおよびリファレンスを提供します。このパッケージは、『Qt Reference Documentation』 も提供し、これは Qt フレームワーク内での開発に絶好の出発点となります。

**qt-demos** や **qt-examples** からさらなるデモや例をインストールすることもできます。Qt フレームワークの機能の概要については、**/usr/bin/qtdemo-qt4** (**qt-demos** が提供) を参照してください。

### 3.3.5. KDE 開発フレームワーク

**kdelibs-devel** パッケージは、KDE ライブラリーを提供します。これは、Qt 上でビルドしてアプリケーション開発を容易にするフレームワークを提供します。KDE 開発フレームワークは、KDE デスクトップ環境にわたって一貫性を提供する手助けにもなります。

#### 3.3.5.1. KDE4 アーキテクチャー

Red Hat Enterprise Linux 6 の KDE 開発フレームワークのアーキテクチャーは KDE4 を使用しており、これは以下のテクノロジーで構築されています。

**Plasma**

**Plasma** は KDE4 で KDesktop の代わりとなるものです。この実装は **Qt Graphics View Framework** に基づいており、これは Qt 4.2 で導入されました。**Plasma** についての詳細は、http://techbase.kde.org/Development/Architecture/KDE4/Plasma を参照してください。

### Sonnet

**Sonnet** は、自動言語検出やプライマリ/バックアップ辞書、その他の便利な機能をサポートする多言語スペルチェックアプリケーションです。これは KDE4 で **kspell2** に代わるものです。

### KIO

KIO ライブラリーは、ネットワーク透過ファイル処理用のフレームワークを提供し、ユーザーはネットワーク透過プロトコルによるファイルアクセスが容易になります。また、標準ファイルダイアログの提供にも役立ちます。

### KJS/KHTML

KJS and KHTML は、(**konqueror** のように) KDE4 にネイティブな異なるアプリケーションが使用する、本格的な JavaScript および HTML エンジンです。

### Solid

**Solid** は、ハードウェアおよびネットワーク認識フレームワークで、これを使用することでハードウェア対話機能でアプリケーションを開発できます。その包括的 API が必要な抽象化を提供し、クロスプラットフォームアプリケーションの開発をサポートします。詳細情報については、http://techbase.kde.org/Development/Architecture/KDE4/Solid を参照してください。

### Phonon

**Phonon** は、マルチメディア機能でアプリケーション開発をサポートするマルチメディアフレームワークです。KDE 内のメディア機能の使用を容易にします。詳細については、http://techbase.kde.org/Development/Architecture/KDE4/Phonon を参照してください。

### Telepathy

**Telepathy** は、KDE4 内でのリアルタイムの通信および共同作業のフレームワークを提供します。主要機能は、KDE 内の異なるコンポーネント間の融合を高めることです。このプロジェクトに関する概要は、http://community.kde.org/Real-Time_Communication_and_Collaboration を参照してください。

### Akonadi

**Akonadi** は、*Parallel Infrastructure Management* (PIM) コンポーネントの一元保存用のフレームワークを提供します。詳細は、http://techbase.kde.org/Development/Architecture/KDE4/Akonadi を参照してください。

### KDE4 内のオンラインヘルプ

KDE4 には、アプリケーションにオンラインヘルプ機能を追加する Qt ベースの操作が容易なフレームワークもあります。この機能に含まれるのは、ツールヒントやホバーヘルプ情報、**khelpcenter** マニュアルなどです。KDE4 内のオンラインヘルプの概要については、http://techbase.kde.org/Development/Architecture/KDE4/Providing_Online_Help を参照してください。

### KXMLGUI

**KXMLGUI** は、XML でユーザーインターフェース設計するためのフレームワークです。このフレームワークを使用すると、ソースコードを改訂することなく (開発者が定義する)「アクション」に基づいて UI 要素を定義することができます。詳細情報は、https://techbase.kde.org/Development/Architecture/KDE3/XMLGUI_Technologyを参照してください。

### Strigi

**Strigi** は、多くのデスクトップ環境およびオペレーティングシステムに互換性のあるデスクトップ検索デーモンです。自身の **jstream** システムを使用することで深い階層にまでのファイルのインデックス化が可能になります。**Strigi** 開発の詳細情報は、http://www.vandenoever.info/software/strigi/を参照してください。

**KNewStuff2**

**KNewStuff2** は、多くの KDE4 アプリケーションが使用する共同作業データ共有ライブラリーです。詳細情報は、http://techbase.kde.org/Projects/KNS2を参照してください。

#### 3.3.5.2. kdelibs ドキュメント

**kdelibs-apidocs** パッケージは、**/usr/share/doc/HTML/en/kdelibs4-apidocs/** で KDE 開発フレームワーク用の HTML ドキュメントを提供します。以下のリンクも KDE 関連のプログラミングタスクに関する詳細情報を提供します。

- http://techbase.kde.org/
- http://techbase.kde.org/Development/Tutorials
- http://techbase.kde.org/Development/FAQs
- http://api.kde.org

### 3.3.6. GNOME の電源管理

GNOME 電源管理インフラストラクチャーのバックエンドプログラミングは、**gnome-power-manager** です。これは、Red Hat Enterprise Linux 5 で導入されており、GNOME デスクトップ環境下で完全かつ統合された電源管理のソリューションを提供します。Red Hat Enterprise Linux 6 では、**hal** のストレージ処理部分は **udisks** に、**libgnomeprint** スタックは **gtk2** のプリントサポートに代わっています。

#### 3.3.6.1. GNOME の電源管理バージョンガイド

本セクションでは、どのバージョンの **gnome-power-management** が Red Hat Enterprise Linux のどのバージョンに同梱されているかを説明します。

ただし、一般的に Red Hat Enterprise Linux 4 には GNOME 2.8 が、RRed Hat Enterprise Linux 5 には GNOME 2.16 が、Red Hat Enterprise Linux 6 には GNOME 2.28 が同梱されています。

**表3.2 デスクトップコンポーネントの比較**

| GNOME の電源管理のデスクトップコンポーネント | Red Hat Enterprise Linux バージョン | | |
|---|---|---|---|
| | 4 | 5 | 6 |
| hal | 0.4.2 | 0.5.8 | 0.5.14 |
| udisks | 該当なし | 該当なし | 1.0.1 |
| glib2 | 2.4.7 | 2.12.3 | 2.22.5 |

| GNOME の電源管理のデスクトップコンポーネント | Red Hat Enterprise Linux バージョン 4 | 5 | 6 |
|---|---|---|---|
| gtk2 | 2.4.13 | 2.10.4 | 2.18.9 |
| gnome-vfs2 | 2.8.2 | 2.16.2 | 2.24.2 |
| libglade2 | 2.4.0 | 2.6.0 | 2.6.4 |
| libgnomecanvas | 2.8.0 | 2.14.0 | 2.26.0 |
| gnome-desktop | 2.8.0 | 2.16.0 | 2.28.2 |
| gnome-media | 2.8.0 | 2.16.1 | 2.29.91 |
| gnome-python2 | 2.6.0 | 2.16.0 | 2.28.0 |
| libgnome | 2.8.0 | 2.16.0 | 2.28.0 |
| libgnomeui | 2.8.0 | 2.16.0 | 2.24.1 |
| libgnomeprint22 | 2.8.0 | 2.12.1 | 該当なし |
| libgnomeprintui22 | 2.8.0 | 2.12.1 | 該当なし |
| gnome-session | 2.8.0 | 2.16.0 | 2.28.0 |
| gnome-power-manager | 該当なし | 2.16.0 | 2.28.3 |
| gnome-applets | 2.8.0 | 2.16.0 | 2.28.0 |
| gnome-panel | 2.8.1 | 2.16.1 | 2.30.2 |

### 3.3.6.2. glib 向けの API 変更

バージョン間では多くの glib 向け API 変更があります。

#### バージョン 2.4 からバージョン 2.12

バージョン 2.4 とバージョン 2.12 間 (もしくは Red Hat Enterprise Linux 4 と Red Hat Enterprise Linux 5) での glib の違いは以下の通りです。

- GOption (コマンドラインオプション解析)
- GKeyFile (key/ini ファイル解析)
- GObject トグルリファレンス

- GMappedFile (マップラッパー)
- GSlice (高速メモリアロケータ)
- GBookmarkFile (ブックマークファイル解析)
- Base64 エンコーディングサポート
- s390 上のネイティブアトミックオプション
- 5 へのユニコードサポートの更新
- GObject 用のアトミックな参照カウント

**バージョン 2.12 からバージョン 2.22**

バージョン 2.12 とバージョン 2.22 間 (または Red Hat Enterprise Linux 5 と Red Hat Enterprise Linux 6) での glib の相違点は以下の通りです。

- GSequence (バランスツリーとして実装されたリストデータ構造)
- GRegex (PCRE regex ラッパー)
- 単調クロックのサポート
- XDG ユーザーディレクトリーのサポート
- GIO (gnome-vfs の代わりとなる VFS ライブラリー)
- GChecksum (MD5 や SHA-256 などのハッシュアルゴリズムのサポート)
- GTest (テストフレームワーク)
- GIO におけるソケットおよびネットワーク IO のサポート
- GHashTable パフォーマンスの改善
- GMarkup パフォーマンスの改善

新規および廃止予定の API のインデックスを含む glib のドキュメントは、glib2-devel パッケージに同梱されています。

### 3.3.6.3. GTK+での API 変更

バージョン間では多くの GTK+ 向け API 変更があります。

**バージョン 2.4 からバージョン 2.10**

バージョン 2.4 とバージョン 2.10 間 (もしくは Red Hat Enterprise Linux 4 と Red Hat Enterprise Linux 5) での GTK+ の相違点は以下の通りです。

- GtkIconView
- GtkAboutDialog
- GtkCellView
- GtkFileChooserButton

- GtkMenuToolButton
- GtkAssistant
- GtkLinkButton
- GtkRecentChooser
- GtkCellRendererCombo
- GtkCellRendererProgress
- GtkCellRendererAccel
- GtkCellRendererSpin
- GtkStatusIcon
- 印刷のサポート
- メモ帳タブ DND サポート
- ラベル、進捗状況バー、ツリービューでの Ellipsisation サポート
- 回転したテキストのサポート
- Themability の改善

**バージョン 2.10 からバージョン 2.18**

バージョン 2.10 とバージョン 2.18 間 (もしくは Red Hat Enterprise Linux 4 と Red Hat Enterprise Linux 5) での GTK+ の相違点は以下の通りです。

- GtkScaleButton
- GtkVolumeButton
- GtkInfoBar
- libglade に代わる GtkBuilder
- 新たなツールヒント API
- GtkMountOperation
- gtk_show_uri
- スケールマーク
- ラベルのリンク
- ランタイムフォント設定変更のサポート
- GIO の使用

新規および廃止予定の API のインデックスを含む GTK+ のドキュメントは、gtk2-devel パッケージに同梱されています。

### 3.3.7. NSS 共有データベース

NSS 3.12 で導入された NSS 共有データベース形式は、Red Hat Enterprise Linux 6 で利用可能となりました。これには、アクセスおよびユーザビリティーを改善する数多くの機能とコンポーネントが含まれています。

これには NSS 認証およびキーデータベースが含まれており、これらは SQLite ベースで同時アクセスを可能にします。レガシーの **key3.db** と **cert8.db** も **key4.db** と **cert9.db** と呼ばれる新たな SQL データベースで置換されています。これらの新規データベースは PKCS #11 トークンオブジェクトを保存します。これらは、現在 **cert8.db** および **key3.db** に保存されているものと同一のものです。

共有データベースのサポートがあることで、システムワイドの NSS データベースが可能になります。これは **/etc/pki/nssdb** に配置されており、ここではグローバルに信頼された CA 認証がすべてのアプリケーションでアクセス可能になります。**rv = NSS_InitReadWrite("sql:/etc/pki/nssdb");** コマンドは、アプリケーション用に NSS を初期化します。アプリケーションを root 権限で実行中の場合は、システムワイドのデータベースが読み込みおよび書き込みで利用可能になります。しかし、アプリケーションが通常のユーザー権限で実行中の場合は、読み込み可能のみとなります。

また、NSS 用の PEM PKCS #11 モジュールは、PEM 形式ファイルに保存されているメモリ認証書およびキーへのアプリケーションのロードを可能にします (たとえば、openssl 作成のものなど)。

#### 3.3.7.1. 後方互換性

NSS アップストリームで保証されているバイナリー互換性は、Red Hat Enterprise Linux 6 の NSS でも確保されます。この保証内容は、NSS 3.12 はすべての古い NSS 3.x の共有ライブラリーと後方互換性があるとしています。このため、古い NSS 3.x の共有ライブラリーとリンクしているプログラムは再コンパイルや再リンクなしで機能します。また、NSS API の使用を NSS 公開機能に限定するアプリケーションは、NSS 共有ライブラリーの今後のバージョンと互換性を保ちます。

#### 3.3.7.2. NSS 共有データベース のドキュメント

Mozilla の wiki ページでは、システムワイドのデータベースの原理を詳細にわたって説明しており、http://wiki.mozilla.org/NSS_Shared_DB_And_LINUX からアクセスできます。

### 3.3.8. Python

**python** パッケージは、Python プログラミング言語のサポートを追加します。このパッケージは、基本的な Python プログラムのランタイムサポートの有効化に必要なオブジェクトとキャッシュバイトコードファイルを提供します。また、**python** インタープリターと **pydoc** ドキュメントツールも含まれています。**python-devel** パッケージには、Python 拡張の開発に必要なライブラリーおよびヘッダーファイルが含まれています。

Red Hat Enterprise Linux には多くの **python** 関連パッケージも同梱されています。慣習として、これらパッケージの名前には、**python** 接頭辞もしくは接尾辞が付いています。このようなパッケージは、ライブラリの拡張機能か既存ライブラリーへの Python バインディングのどちらかです。例えば、**dbus-python** は D-Bus 用の Python 言語バインディングです。

キャッシュされたバイトコード (***.pyc**/***.pyo** ファイル) とコンパイルされた拡張モジュール (***.so** ファイル) の両方が Python 2.4 (Red Hat Enterprise Linux 5 で使用) と Python 2.6 (Red Hat Enterprise Linux 6 で使用) の間で互換性がないことに注意してください。このため、Red Hat Enterprise Linux の一部でない拡張モジュールは再構築する必要があります。

#### 3.3.8.1. Python 更新

Red Hat Enterprise Linux 6 バージョンの Python には、多くの言語変更機能があります。これらの変更についての情報は、以下のプロジェクト資料を参照してください。

- What's New in Python 2.5: http://docs.python.org/whatsnew/2.5.html
- What's New in Python 2.6: http://docs.python.org/whatsnew/2.6.html

どちらのサイトにも、Python の以前のバージョンを使用して開発したコードのポートに関するアドバイスが含まれています。

#### 3.3.8.2. Python ドキュメント

Python に関する詳細情報は、**man python** を参照してください。また、**python-docs** をインストールすることもできます。これは、以下の場所で HTML マニュアルとリファレンスを提供します。

**file:///usr/share/doc/python-docs-*version*/html/index.html**

ライブラリーおよび言語コンポーネントについては、**pydoc *component_name*** を使用してください。たとえば、**pydoc math** は **math** Python モジュールについて以下の情報を表示します。

```
Help on module math:

NAME
 math

FILE
 /usr/lib64/python2.6/lib-dynload/mathmodule.so

DESCRIPTION
 This module is always available.  It provides access to the
 mathematical functions defined by the C standard.

FUNCTIONS
 acos[...]
  acos(x)

  Return the arc cosine (measured in radians) of x.

 acosh[...]
  acosh(x)

  Return the hyperbolic arc cosine (measured in radians) of x.

 asin(...)
  asin(x)

  Return the arc sine (measured in radians) of x.

 asinh[...]
  asinh(x)

  Return the hyperbolic arc sine (measured in radians) of x.
```

Python 開発プロジェクトのメインサイトは、python.org になります。

### 3.3.9. Java

**java-1.6.0-openjdk** パッケージは、Java プログラミング言語のサポートを追加します。このパッケージは **java** インタープリターを提供します。**java-1.6.0-openjdk-devel** パッケージには、Java 拡張の開発に必要なライブラリーおよびヘッダーファイルのほか、**javac** コンパイラーが含まれています。

Red Hat Enterprise Linux には多くの **java** 関連パッケージも同梱されています。慣習として、これらパッケージの名前には、**java** 接頭辞もしくは接尾辞が付いています。

#### 3.3.9.1. Java ドキュメント

Java についての詳細情報は、**man java** を参照してください。関連のユーティリティーにはそれぞれ、**man** ページがあるものもあります。

特定の Java ユーティリティーについての詳細情報のために他の Java ドキュメントをインストールすることもできます。慣習として、これらパッケージの名前には、**javadoc** 接尾辞が付いています (たとえば、**dbus-java-javadoc**)。

Java 開発のメインサイトは http://openjdk.java.net/ です。Java のライブラリーランタイムのメインサイトは http://icedtea.classpath.org です。

### 3.3.10. Ruby

**ruby** パッケージは、Ruby インタープリターを提供し、Ruby プログラミング言語のサポートを追加します。**ruby-devel** パッケージには、Ruby 拡張開発に必要なライブラリーおよびヘッダーファイルが含まれています。

Red Hat Enterprise Linux には多くの **ruby** 関連パッケージも同梱されています。慣習として、これらパッケージの名前には、**ruby** もしくは **rubygem** の接頭辞または接尾辞が付いています。このようなパッケージは、ライブラリーの拡張機能か既存ライブラリーへの Ruby バインディングのどちらかです。

**ruby** 関連パッケージの例は以下のとおりです。

- ruby-flexmock
- rubygem-flexmock
- rubygems
- ruby-irb
- ruby-libguestfs
- ruby-libs
- ruby-qpid
- ruby-rdoc
- ruby-ri
- ruby-saslwrapper
- ruby-static

- ruby-tcltk

Red Hat Enterprise Linux 6 の Ruby 言語への更新に感する情報は、以下のリンクを参照してください。

- **file:///usr/share/doc/ruby-*version*/NEWS**
- **file:///usr/share/doc/ruby-*version*/NEWS-*version***

#### 3.3.10.1. Ruby ドキュメント

Ruby に関する詳細情報は、**man ruby** を参照してください。また、**ruby-docs** をインストールすることもできます。これは、以下の場所で HTML マニュアルとリファレンスを提供します。

**file:///usr/share/doc/ruby-docs-*version*/**

Ruby 開発のメインサイトは、http://www.ruby-lang.org です。http://www.ruby-doc.org こちらのサイトにも Ruby ドキュメントが含まれています。

### 3.3.11. Perl

**perl** パッケージは、Perl プログラミング言語のサポートを追加します。このパッケージは、Perl コアモジュール、Perl Language Interpreter (Perl 言語インタープリター)、PerlDoc ツールを提供します。

Red Hat は、パッケージ形式でも多くの perl モジュールを提供します。これらパッケージの名前には、**perl-*** 接頭辞が付いています。これらのモジュールは、スタンドアロンのアプリケーションや言語拡張、Perl ライブラリー、外部ライブラリーバインディングを提供します。

#### 3.3.11.1. Perl 更新

Perl バージョン間の違いについては、以下の資料を参照してください。

**Perl 5.12 更新**

Perl 5.12 には以下の更新があります。

- Perl はユニコード標準により適合します。
- 実験的な API により、Perl を「プラグ可能」なキーワードおよび構文で拡張することができます。
- Perl は 「2038 年」の問題以降も正確な時間を維持できます。
- パッケージのバージョン番号は、直接「パッケージ」ステートメントに指定することができます。
- Perl はデフォルトで、廃止予定の機能の使用をユーザーに警告します。

Perl 5.12 delta は、http://perldoc.perl.org/perl5120delta.html でアクセス可能です。

**Perl 5.14 更新**

Perl 5.14 には以下の更新があります。

- ユニコード 6.0 をサポート
- IPv6 のサポートが改善

- CPAN クライアントの自動設定の容易化
- 新たな /r フラグが s/// 置換を非破壊的なものにします。
- 新規の正規表現フラグが、マッチしたストリングを ASCII またはユニコードとして扱うかを管理します。
- 新たな **package** ***Foo*** **{ }** 構文
- 以前のリリースに比べてメモリおよび CPU 使用量が少ない。
- 多くのバグを修正

Perl 5.14 delta は http://perldoc.perl.org/perl5140delta.html でアクセスできます。

### Perl 5.16 更新

Perl 5.14 には以下の更新があります。

- Unicode 6.1 のサポート
- **$$** 変数が書き込み可能。
- デバッガーの改善
- Unicode データベースファイルへの直接アクセスは廃止予定。代わりに Unicode::UCD を使用。
- *Version*::*Requirements* が廃止予定。代わりに CPAN::*Meta*::*Requirements* を使用。
- 多くの perl4 ライブラリーを削除:
  - abbrev.pl
  - assert.pl
  - bigfloat.pl
  - bigint.pl
  - bigrat.pl
  - cacheout.pl
  - complete.pl
  - ctime.pl
  - dotsh.pl
  - exceptions.pl
  - fastcwd.pl
  - flush.pl
  - getcwd.pl

- getopt.pl
- getopts.pl
- hostname.pl
- importenv.pl
- lib/find{,depth}.pl
- look.pl
- newgetopt.pl
- open2.pl
- open3.pl
- pwd.pl
- hellwords.pl
- stat.pl
- tainted.pl
- termcap.pl
- timelocal.pl

Perl 5.16 delta は http://perldoc.perl.org/perl5160delta.html でアクセス可能です。

### 3.3.11.2. インストール

Perl の機能は追加モジュールをインストールすることで拡張されます。モジュールは以下の形式になります。

**公式の Red Hat RPM**

公式モジュールパッケージは、Red Hat Enterprise Linux リポジトリーから **yum** または **rpm** でインストールできます。これらは **/usr/share/perl5** と、32 ビットアーキテクチャーの場合は **/usr/lib/perl5** に、64 ビットアーキテクチャーの場合は **/usr/lib64/perl5** にインストールされます。

**CPAN からのモジュール**

perl-CPAN パッケージが提供する **cpan** ツールを使用して CPAN Web サイトから直接モジュールをインストールします。これは **/usr/local/share/perl5** と、32 ビットアーキテクチャーの場合は **/usr/local/lib/perl5** に、64 ビットアーキテクチャーの場合は **/usr/local/lib64/perl5** にインストールされます。

**サードパーティのモジュールパッケージ**

サードパーティのモジュールは **/usr/share/perl5/vendor_perl** と、32 ビットアーキテクチャーの場合は **/usr/lib/perl5/vendor_perl** に、64 ビットアーキテクチャーの場合は **/usr/lib64/perl5/vendor_perl** にインストールされます。

**カスタムのモジュールパッケージ/手動でインストールするモジュール**

これらのモジュールは、サードパーティの場合と同じディレクトリーに置かれます。つまり、**/usr/share/perl5/vendor_perl** と、32 ビットアーキテクチャーの場合は **/usr/lib/perl5/vendor_perl** に、64 ビットアーキテクチャーの場合は **/usr/lib64/perl5/vendor_perl** にインストールされます。

**警告**

モジュールの公式バージョンが既にインストールされている場合は、非公式バージョンをインストールすると、**/usr/share/man** ディレクトリーで競合が発生する場合があります。

### 3.3.11.3. Perl のドキュメント

**perldoc** ツールは、言語およびコアモジュールに関するドキュメントを提供します。モジュールについてのさらなる情報は、**perldoc module_name** を使用してください。たとえば、**perldoc CGI** は CGI コアモジュールについて以下の情報を表示します。

```
        NAME
        CGI - Handle Common Gateway Interface requests and responses

        SYNOPSIS
        use CGI;

   my $q = CGI->new;

   [...]

   DESCRIPTION
   CGI.pm is a stable, complete and mature solution for processing and
preparing HTTP requests and responses.  Major features including
processing form submissions, file uploads, reading and writing cookies,
query string generation and manipulation, and processing and preparing
HTTP headers. Some HTML generation utilities are included as well.

   [...]

   PROGRAMMING STYLE
   There are two styles of programming with CGI.pm, an object-oriented
style and a function-oriented style.  In the object-oriented style you
create one or more CGI objects and then use object methods to create the
various elements of the page.  Each CGI object starts out with the list of
named parameters that were passed to your CGI script by the server.

   [...]
```

Perl 機能の詳細については、**perldoc -f *function_name*** を使用してください。たとえば、perldoc -f split は split 機能について以下の情報を表示します。

-

```
      split /PATTERN/,EXPR,LIMIT
      split /PATTERN/,EXPR
      split /PATTERN/
      split   Splits the string EXPR into a list of strings and returns
that list.  By default, empty leading fields are preserved, and empty
trailing ones are deleted.  (If all fields are empty, they are considered
to be trailing.)

      In scalar context, returns the number of fields found. In scalar and
void context it splits into the @_ array.  Use of split in scalar and void
context is deprecated, however, because it clobbers your subroutine
arguments.

      If EXPR is omitted, splits the $_ string.  If PATTERN is also
omitted, splits on whitespace (after skipping any leading whitespace).
Anything matching PATTERN is taken to be a delimiter separating the
fields.  (Note that the delimiter may be longer than one character.)

      [...]
```

最新の PerlDoc ドキュメントは perldoc.perl.org にあります。

コアおよび外部モジュールは Comprehensive Perl Archive Network で提供されています。

---

[1] MPI サポートは、IBM System Z マシン (ここでは Open MPI が利用不可) では利用できません

# 第4章 コンパイルおよびビルド

Red Hat Enterprise Linux 6 には、ソースコードのコンパイルおよびビルド用のツールなど、ソフトウェア開発に使用する多くのパッケージが含まれています。本章では、ソースコードのコンパイルに使用するパッケージおよびツールのいくつかを説明します。

## 4.1. GNU コンパイラーコレクション (GCC)

GNU コンパイラーコレクション (GCC) は、様々なプログラミング言語 (C、C++、ObjectiveC、ObjectiveC++、Fortran、Ada を含む) を非常に最適化されたマシンコードにコンパイルするツールセットです。これらのツールには、様々なコンパイラー (**gcc** や **g++** など)、ランタイムライブラリー (**libgcc**、**libstdc++**、**libgfortran**、**libgomp** など)、その他のユーティリティーが含まれます。

### 4.1.1. GCC ステータスおよび機能

Red Hat Enterprise Linux 6 用の GCC は 4.4.x リリースシリーズをベースとしており、予定されているリリースからのバックポート (GCC 4.5 を含む)、バグ修正や機能強化が含まれていますが、Red Hat Enterprise Linux 6 の機能が凍結された際に、GCC 4.5 の成熟度はエンタープライズディストリビューションには不十分とみなされました。

この標準化は、4.4 シリーズの更新が利用可能になると (4.4.1、4.4.2 など)、Red Hat Enterprise Linux 6 に含まれているコンパイラーに更新として取り込まれることを意味します。Red Hat は 4.4 シリーズ以外の予定されているリリースから新たなバックポートや機能強化をインポートする可能性がありますが、Enterprise Linux リリース内での互換性を断つものではありません。場合によっては、標準に準拠していないコードがコンパイルに失敗することや、バグ修正や標準準拠のプロセス中に機能が変更することあります。

前回の Red Hat Enterprise Linux リリース以降、GCC は 4.2.x、4.3.x、4.4.x の 3 つのメジャーリリースを行いました。以下は、変更点の概要になります。

- inliner、実行されないコードの除外ルーチン、コンパイル時間、メモリ使用コードが改善されました。このリリースでは、新たなレジスタアロケータ、指示スケジューラ、ソフトウェアパイプラインの機能があります。
- OpenMP のバージョン 3.0 仕様が C、C++、Fortran コンパイラー対応となっています。
- 予定されている ISO C++ 標準 (C++0x) の体験サポートが含まれています。これには、自動/inline ネームスペース、文字タイプ、スコープ列挙のサポートがあります。これを有効にするには、コンパイラーオプションの **-std=c++0x** (GNU 拡張機能を無効化) か **-std=gnu++0x** を使用します。

  C++0x の改善点の詳細リストについては、以下のリンク先を参照してください。

  http://gcc.gnu.org/gcc-4.4/cxx0x_status.html
- GCC は *Variable Tracking at Assignments*(VTA) インフラストラクチャーを組み込むようになりました。これにより、最適化中の GCC の変数追跡機能が向上され、GNU Project Debugger や SystemTap、その他のツール用に作成されるデバッグ情報 (つまり、DWARF) が改善されます。VTA の概要については「Variable Tracking at Assignments」を参照してください。

  VTA を使うと、以前の GCC リリースと比べてはるかに最適化されたコードのデバッグがうまく実行でき、優れたデバッグ体験を提供するために -O0 でコンパイルする必要がなくなります。
- Fortran 2003 のサポートが延長され、Fortran 2008 がサポート対象となりました。

GCC の改善点の詳細なリストは、以下を参照してください。

- 『Updates in the 4.2 Series』: http://gcc.gnu.org/gcc-4.2/changes.html
- 『Updates in the 4.3 Series』: http://gcc.gnu.org/gcc-4.3/changes.html
- 『Updates in the 4.4 Series』: http://gcc.gnu.org/gcc-4.4/changes.html

GCC 4.4 再ベースで導入された変更点に加え、Red Hat Enterprise Linux 6 バージョンの GCC はアップストリームソース (バージョン 4.5 およびそれ以降) からバックポートされた修正および機能強化も備えています。これらの改善点には以下のものが含まれます。

- 最適化された C++ コードのデバッグ用の DWARF3 デバッグの改善
- Fortran 最適化の改善
- ix86、Intel 64 および AMD64、s390 の指示の長さ情報の精度改善
- Intel Atom のサポート
- POWER7 のサポート
- C++ raw 文字列サポート、u/U/u8 文字列リテラルサポート

### 4.1.2. 言語の互換性

GNU C、C++、Fortran、Java Compiler が指定する アプリケーションバイナリーインターフェース (ABI) には、以下が含まれます。

- 呼び出し規約。これは、引数の関数への渡され方と関数からの結果の返し方を指定します。
- レジスタ使用量規約。これはプロセッサーレジスタの割り当ておよび使用方法を指定します。
- オブジェクトファイル形式。これは、バイナリーオブジェクトコードの表示方法を指定します。
- サイズ、レイアウト、データ配置のタイプ。これは、メモリでのデータの配置方法を指定します。
- ランタイム環境が提供するインターフェース。記録されたセマンティクスがあるバージョンから別のバージョンで変更されない部分では、これらは利用可能である必要があり、常に同じ名前を使う必要があります。

Red Hat Enterprise Linux 6 に含まれているデフォルトのシステム C コンパイラーは、C99 ABI 標準と幅広い互換性があります。GCC 4.4 の C99 標準との偏差は、online で追跡されます。

C ABI に加えて、GNU C++ コンパイラーのアプリケーションバイナリーインターフェースは、C++ 言語のサポートに必要な以下のバイナリーインターフェースを指定します。

- 名前のマングル化およびデマングル化
- 例外の生成および伝達
- ランタイムタイプ情報のフォーマット
- コンストラクターおよびデストラクター

- クラスおよび派生クラスのレイアウト、配置、パディング
- 仮想テーブルのレイアウトおよび配置といった仮想機能実装の詳細

Red Hat Enterprise Linux 6 に含まれるデフォルトのシステム C++ コンパイラーは、Itanium C++ ABI (1.86) が定義する C++ ABI に準拠します。

GCC の各バージョンは以前のリリースと互換性を保つようにできる限りの努力がなされていますが、いくつかの非互換性が存在します。

**Red Hat Enterprise Linux 6 と Red Hat Enterprise Linux 5 の間における ABI の非互換性**

以下は、Red Hat Enterprise Linux 6 と 5 のツールチェーン間での既知の非互換性です。

- 柔軟性のある配列メンバーの構造体を値でパス/返却すると、Intel 64 および AMD64 で変更する場合がある。
- long double メンバーのユニオンを値でパス/返却すると、Intel 64 および AMD64 で変更する場合がある。
- 複雑な浮動メンバーの構造体を値でパス/返却すると、Intel 64 および AMD64 で変更する場合がある。
- **-mavx** を使用する場合、x86、Intel 64、AMD64 のプラットフォームでの 256 ビットのベクターのパスが変更される。
- _Decimal{32,64,128} タイプおよびいくつかのターゲットで値ごとのものを含む総計のパスに複数の変更がある。
- パックされた char ビットフィールドの圧縮が変更されるケースがある。

**Red Hat Enterprise Linux 5 と Red Hat Enterprise Linux 4 の間における ABI の非互換性**

以下は、Red Hat Enterprise Linux 5 と 4 のツールチェーン間での既知の非互換性です。

- C++ ABI が指定するライブラリーインターフェースで、機能スコープの静的変数のスレッドセーフな初期化に変更があります。
- Intel 64 および AMD64 では、データセグメントが 4GB を超えるアプリケーション構築用の中型モデルは、当時の最新 ABI ドラフトに一致するように再設計されました。ABI の変更が中型モデルのオブジェクト間での非互換性につながっています。

コンパイラーのフラグ **-Wabi** は、これらの構造文がソースコードのどこに現れるかを示す診断を得るために使用できますが、すべてのケースを捕捉するわけではありません。このフラグは、ベンダー中立の C++ ABI と互換性がないと分かっているコードをコンパイラー生成する際はいつでも警告するので、C++ コードでは特に便利です。

上記に一覧表示した非互換性を除くと、GCC C と C++ 言語 ABI は**ほとんど** ABI と互換性があります。ほとんどのソースコードは既知の問題に遭遇することはなく、互換性があると考えて問題ありません。

ABI の互換性がある場合には、ソースコードのコンパイルで作成されたオブジェクトを他のシステムに移動できるようになります。特に Red Hat Enterprise Linux では、**これで上位** 互換性が可能になります。上位互換性とは、特定の Red Hat Enterprise Linux リリースのコンパイラーのバージョンで作成した共有ライブラリーおよびオブジェクトを問題なくリンクできる機能のことです。これには、以降の Red Hat Enterprise Linux でコンパイルされた新たなオブジェクトも含まれます。

C ABI は少なくとも Red Hat Enterprise Linux 3 以降 (上記リストで説明した非互換性を除く)、安定性があると考えられています。Red Hat Enterprise Linux 3 およびそれ以降で構築されたライブラリー

は、それ以降の環境 (Red Hat Enterprise Linux 4、Red Hat Enterprise Linux 5、Red Hat Enterprise Linux 6) で作成されたオブジェクトにリンクさせることができます。

C++ ABI は安定性があるものの、C ABI ほどの安定性はありません。また Red Hat Enterprise Linux 4 以降でのみ安定しています (GCC バージョン 3.4 およびそれ以上に対応)。C の場合と同様に、上位互換性のみになります。Red Hat Enterprise Linux 4 およびそれ以降で構築されたライブラリーは、それ以降の環境 (Red Hat Enterprise Linux 5 および Red Hat Enterprise Linux 6) で作成されたオブジェクトにリンクさせることができます。

Red Hat Enterprise Linux 4 より前の Red Hat Enterprise Linux の C++ ABI と互換性のあるコードを強制的に GCC に生成させるため、開発者によっては **-fabi-version=1** オプションを使用していました。この方法は、推奨されません。この方法で作成されるオブジェクトは、最新の安定的な ABI に適合するオブジェクトと区別がつかず、異なる ABI の間で (誤って) リンクされる可能性があります。特に、新しいコンパイラーを使ってコードを作成し、これを Red Hat Enterprise Linux 4 より前のツールで構築された古いライブラリーにリンクする場合などです。

**警告**

特に、コアライブラリー外で複数の依存関係によりカスタムライブラリを開発する場合など、上記の非互換性が原因でリリース間の ABI 共有ライブラリーを適切に維持することは非常に困難になります。このため、共有ライブラリーを開発する際は、Red Hat Enterprise Linux の各リリースで新たなバージョンを構築することを **強く** 推奨します。

### 4.1.3. オブジェクトの互換性と相互運用性

2 つの重要なアイテムは、コンパイラーが使用する基礎的ツールにおける変更と強化、またある言語のコンパイラーの異なるバージョン間での互換性です。

**ld** (**binutils** パッケージの一部として配布) や動的ローダー (**glibc** パッケージの一部として配布される **ld.so**) などのツールの変更や新機能は、コンパイラーが作成するオブジェクトファイルをわずかに変更する可能性があります。これらの変更では、Red Hat Enterprise Linux の以前のリリースから最新リリースに移動するオブジェクトファイルは機能を失い、ランタイムで異なる動作をし、相互運用性が減少する可能性があることになります。既知の問題分野は以下のとおりです。

- **ld --build-id**

  Red Hat Enterprise Linux 6 では、これはデフォルトで **ld** に渡されます。一方、Red Hat Enterprise Linux 5 では、**ld** は認識されません。

- **as .cfi_sections** サポート

  Red Hat Enterprise Linux 6 では、このディレクティブにより、**.cfi*** ディレクティブから **.debug_frame** か **.eh_frame**、または両方を省略できるようになります。Red Hat Enterprise Linux 5 で省略可能なのは、**.eh_frame** のみです。

- **as**、**ld**、**ld.so**、**gdb**、**STB_GNU_UNIQUE**、**%gnu_unique_symbol** サポート

  Red Hat Enterprise Linux 6 では、より多くのデバッグ情報が生成されてオブジェクトファイルに保存されます。この情報は、**DWARF** 標準で詳述されている新機能と標準化されていない新たな拡張機能にも依存しています。Red Hat Enterprise Linux 5 では、**as** や

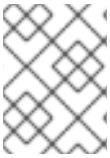
**ld**、**gdb**、**objdump**、**readelf** といったツールはこの新情報に対応できていない可能性があり、新規ツールが作成したオブジェクトとの相互運用に失敗する可能性があります。また、Red Hat Enterprise Linux 5 で作成されたオブジェクトファイルは、これらの新機能に対応していません。これらのオブジェクトファイルは Red Hat Enterprise Linux 6 ツールでは最適でない方法で処理される可能性があります。

このようにデバッグの強化情報が増えたことで、システムライブラリーと共に出荷される **debuginfo** パッケージがインストールされていれば、システムライブラリーへの有益なソースレベルでのデバッグが可能になります。**debuginfo** パッケージに関する詳細情報は、「Debuginfo パッケージのインストール」を参照してください。

上記記載のようなオブジェクトファイルの変更は、**prelink** のポータブルな使用を妨げる可能性があります。

### 4.1.4. 後方互換性パッケージ

Red Hat Enterprise Linux の古いバージョンから最新リリースにソースコードや実行可能ファイルを移動するためのパッケージがいくつか提供されています。これらのパッケージは、ソースを動作が変更された新しいコンパイラーに移動する一時的な手助けとして、もしくはコンパイル環境からシステム環境の違いを切り離す便利な方法として使用するためのものです。



**注記**

Red Hat は、これらのパッケージを今後のRed Hat Enterprise Linux リリースからは削除する可能性があることに留意してください。

以下のパッケージは、**Red Hat Enterprise Linux 4 で古いコンパイラーを使用しているかのように** Red Hat Enterprise Linux 6 上で Fortran または C++ ソースコードをコンパイルするための互換性ツールを提供します。

- **compat-gcc-34**
- **compat-gcc-34-c++**
- **compat-gcc-34-g77**

以下のパッケージは、**Red Hat Enterprise Linux 5 でコンパイルされた** Fortran 実行可能ファイルの互換性ランタイムライブラリーを提供し、Red Hat Enterprise Linux 6 の最新リリースで再コンパイルなしで実行できるようにします。

- **compat-libgfortran-41**

後方互換性ライブラリーパッケージは、サポートされているシステムライブラリーすべてに提供されているわけではありません。コンパイラーおよび C/C++ 標準ライブラリーに関連するシステムライブラリーのみになります。

後方互換性ライブラリーパッケージに関する詳細は、Red Hat Enterprise Linux 6 『移行計画ガイド』の 『アプリケーションの互換性』 のセクションを参照してください。

### 4.1.5. Red Hat Enterprise Linux 5 での Red Hat Enterprise Linux 6 コンパイラー機能のプレビュー

Red Hat Enterprise Linux 5 では、更新としてパッケージ **gcc44** が含まれています。これは、Red Hat Enterprise Linux 6 コンパイラーのバックポートで、Red Hat Enterprise Linux 5 を実行するユーザーが Red Hat Enterprise Linux 6 コンパイラーでコードをコンパイルして新機能や最適化を体験してから、

次のメジャーリリースの際にシステムをアップグレードすることができます。作成されたバイナリーは Red Hat Enterprise Linux 6 と前方互換があるため、Red Hat Enterprise Linux 5 上で **gcc44** を使用してコンパイルして、Red Hat Enterprise Linux 5、Red Hat Enterprise Linux 6 以降で実行するｋとおができます。

Red Hat Enterprise Linux 5 **gcc44** コンパイラーは、Red Hat Enterprise Linux 6 と同梱の GCC 4.4.x と適度に調和され、移行を容易にします。ただし、最新機能を使用するには、開発に Red Hat Enterprise Linux 6 を使用することが推奨されます。**gcc44** は変換プロセスの支援としてのみ提供されています。

### 4.1.6. GCC の実行

GCC ツールを使ってコンパイルするには、まず **binutils** と **gcc** をインストールします。こられでいくつかの依存関係もインストールされます。

簡単に言うと、ツールは **gcc** コマンドで動作します。これがコンパイラーの主要なドライバーです。コマンドラインから、ソースファイルの前処理またはコンパイル、オブジェクトファイルおよびライブラリーのリンク、またはこれらの組み合わせを実行できます。デフォルトでは、**gcc** が提供されている **libgcc** ライブラリー内の詳細とリンクの処理をします。

GCC が提供するコンパイラー機能は、**CDT** の一部として Eclipse IDE にも統合されます。これにより多くの利点がもたらされます。グラフィカルインターフェースと完全統合型の環境を好む開発者の場合は特にそうです。

逆に、コマンドラインインターフェースから GCC ツールを使うと、消費するシステムリリースが少なくすみます。これだと、コンパイラーに対する粒度の細かい管理が可能になります。GCC のコマンドラインツールは、グラフィカルモード (ランレベル 5) 外でも使用できます。

#### 4.1.6.1. シンプルな C の使用

GCC を使用した C 言語プログラムの基本的なコンパイルは簡単なものです。次のシンプルなプログラムで開始します。

**hello.c**

```
#include <stdio.h>

int main ()
{
  printf ("Hello world!\n");
  return 0;
}
```

以下の手順は、最も基本的な形での C のコンパイルプロセスになります。

**手順4.1 'Hello World' C プログラムのコンパイル**

1. 次のコマンドを使用して hello.c を実行可能ファイルにコンパイルします。

   ```
   gcc hello.c -o hello
   ```

   作成されるバイナリー **hello** が **hello.c** と同じディレクトリーにあることを確認します。

2. **hello** バイナリー、つまり **hello** を実行します。

#### 4.1.6.2. シンプルな C++ の使用

GCC を使用した C++ 言語プログラムの基本的なコンパイルは同様のものです。次のシンプルなプログラムで開始します。

**hello.cc**

```
#include <iostream>

using namespace std;

int main(void)
{
  cout << "Hello World!" << endl;
  return 0;
}
```

以下の手順は、最も基本的な形での C++ のコンパイルプロセスになります。

**手順4.2 'Hello World' C++ プログラムのコンパイル**

1. 次のコマンドを使用して hello.cc を実行可能ファイルにコンパイルします。

   ```
   g++ hello.cc -o hello
   ```

   生成されるバイナリー **hello** が **hello.cc** と同じディレクトリーにあることを確認します。

2. **hello** バイナリー、つまり **hello** を実行します。

#### 4.1.6.3. シンプルなマルチファイルの使用

マルチファイルもしくはオブジェクトファイルの基本的なコンパイルを行うには、以下の 2 つのソースファイルで開始します。

**one.c**

```
#include <stdio.h>
void hello()
{
  printf("Hello world!\n");
}
```

**two.c**

```
extern void hello();

int main()
{
  hello();
  return 0;
}
```

以下の手順は、最も基本的な形式でシンプルなマルチファイルのコンパイルプロセスを説明しています。

**手順4.3 マルチソースファイルによるプログラムのコンパイル**

1. 次のコマンドを使用して one.c を実行可能ファイルにコンパイルします。

   ```
   gcc -c one.c -o one.o
   ```

   生成されるバイナリー **one.o** が **one.c** と同じディレクトリーにあることを確認します。

2. 次のコマンドを使用して two.c を実行可能ファイルにコンパイルします。

   ```
   gcc -c two.c -o two.o
   ```

   生成されるバイナリー **two.o** が **two.c** と同じディレクトリーにあることを確認します。

3. 次のコマンドを使用して **one.o** および **two.o** を単一の実行可能ファイルにコンパイルします。

   ```
   gcc one.o two.o -o hello
   ```

   生成されるバイナリー **hello** が **one.o** および **two.o** と同じディレクトリーにあることを確認します。

4. **hello** バイナリー、つまり **hello** を実行します。

#### 4.1.6.4. 推奨される最適化オプション

プロジェクトによって必要な最適化オプションは異なります。最適化では万能型のアプローチはありませんが、以下に留意すべきガイドラインを挙げます。

**指示の選択およびチューニング**

指示のスケジューリングに正しいアーキテクチャーを選択することは、非常に重要です。デフォルトでは、GCC はほとんどの一般的なプロセッサーに最適化されたコードを作成しますが、コードが動作する CPU が分かっている場合は、それに対応する指示スケジューリングを最適化する **-mtune=** オプションと、指示選択を最適化する **-march=** オプションを使用すべきです。

オプション **-mtune=** は、指示スケジューリングを最適化して、ABI および利用可能な指示セットを除きすべてをチューニングすることでアーキテクチャーに適合させます。このオプションは特定の指示を選択しませんが、その代わりにプログラムのチューニングを行い、特定のアーキテクチャー上での実行を最適化します。たとえば、大体において Intel Core2 CPU を使うのであれば、**-mtune=core2** を選択します。選択が間違っている場合は、プログラムは実行しますが、そのアーキテクチャー上での最善の動作にはなりません。そのプログラムが実行する可能性の最も高いアーキテクチャーを常に選択するようにします。

オプション **-march=** は、指示選択を最適化します。間違った選択をするとプログラムが失敗するので、正確な選択が重要になります。このオプションは、コード生成時に使用される指示セットを選択します。たとえば、プログラムが AMD K8 コアベースの CPU で実行されるのであれば、**-march=k8** を選択します。このオプションでアーキテクチャーを指定することで、**-mtune=** が暗示されます。

**-mtune=** と **-march=** コマンドは、異なるコード生成 (クロスコンパイル) のためではなく、あるアーキテクチャー内でのチューニングと指示選択のみに使用するようにします。たとえば、Intel 64 や AMD64 プラットフォームからの PowerPC コード生成には使用しません。

**-march=** および **-mtune=** の両方で利用可能なオプションの完全一覧については、GCC 4.4.4 Manual: Hardware Models and Configurations のリンクから取得可能な GCC ドキュメントを参照してください。

#### 一般目的の最適化フラグ

コンパイラーフラグ **-O2** は、高速コード生成のすぐれた穏健なオプションです。作成されるコードサイズが大き過ぎない場合に、最適化されたコードが作成されます。最善のものがわからない場合は、このオプションを使用してください。

コードサイズが問題とならない場合は、**-O3** が推奨されます。このオプションはやや大きめのコードを作成しますが、関数のインラインの頻度が高いため、実行速度が速くなります。浮動小数点集約型コードではこれが理想的なものになります。

他の一般目的の最適化フラグは、**-Os** です。このフラグはサイズの最適化も行い、より小さいフットプリントがコードのローカリティを増やし、キャッシュミスを減らすような状況で高速コードを作成します。

オブジェクトのコンパイル時には **-frecord-gcc-switches** を使用します。これにより、オブジェクトをオブジェクト自体にビルドする際に使用されるオプションが記録されます。オブジェクトがビルドされた後で、そのビルドに使用されたオプションセットを見つけ出します。このオプションセットはその後、オブジェクト内の **.GCC.command.line** と呼ばれるセクションに記録され、以下のように確認可能になります。

```
$ gcc -frecord-gcc-switches -O3 -Wall hello.c -o hello
$ readelf --string-dump=.GCC.command.line hello

String dump of section '.GCC.command.line':
  [     0]  hello.c
  [     8]  -mtune=generic
  [    17]  -O3
  [    1b]  -Wall
  [    21]  -frecord-gcc-switches
```

典型的なデータセットを使用して、さまざまなオプションをテストして試すことは非常に重要です。異なるモジュールやオブジェクトを異なる最適化フラグでコンパイルすると、最適な結果が得られる場合がしばしばあります。さらなる最適化チューニングについては、「プロファイルフィードバックを使用した最適化ヒューリスティックのチューニング」 を参照してください。

### 4.1.6.5. プロファイルフィードバックを使用した最適化ヒューリスティックのチューニング

典型的なソースコードを実行可能ファイルに変換する間、コードの一部と別の部分での速度の重要性やコード速度に対するコードサイズについてなど、数万もの選択をする必要があります。デフォルトでは、これらの選択はコンパイラーが妥当なヒューリスティックを使用して行い、最適なランタイムパフォーマンスを生み出すために長期的にチューニングされます。しかしGCC には、特定の本番環境において特定のマシンで実行可能ファイルを最適化するようにコンパイラーに教える方法があります。この機能は、プロファイルフィードバックと呼ばれます。

プロファイルフィードバックは、以下のような最適化をチューニングするために使用されます。

- インライン
- 分岐予測
- 指示のスケジューリング
- 手順間の不断の伝達
- ホットもしくはコールド機能の区別

プロファイルフィードバックは最初にプログラムをコンパイルし、実行、分析されるプログラムを生成してから、収集したデータで最適化を行います。

**手順4.4 プロファイルフィードバックの使用**

1. プロファイル情報を作成するには、アプリケーションを **-fprofile-generate** でコンパイルしてインストルメント化する必要があります。
2. アプリケーションを実行してプロファイル情報を蓄積し、保存します。
3. アプリケーションを **-fprofile-use** で再コンパイルします。

ステップ 3 ではステップ 1 で収集されたプロファイル情報を使ってコンパイラーのヒューリスティックをチューニングする一方で、コードを最終的な実行可能ファイルに最適化します。

**手順4.5 プロファイリングフィードバックを使ったプログラムのコンパイル**

1. **source.c** をコンパイルし、プロファイリングインストルメンテーションを含めます。

   ```
   gcc source.c -fprofile-generate -O2 -o executable
   ```

2. **executable** を実行してプロファイリング情報を収集します。

   ```
   ./executable
   ```

3. ステップ 1 で収集したプロファイリング情報を使って **source.c** を再コンパイル、最適化します。

   ```
   gcc source.c -fprofile-use -O2 -o executable
   ```

ステップ 2 にあるように、複数のデータ収集の実行はデータを置き換えるのではなく、プロファイリングファイルに蓄積します。これにより、ステップ 2 の実行可能ファイルが追加された対応するデータで複数回実行することが可能になり、より多くの情報が収集できます。

実行可能ファイルは、使用されるマシンと必要な入力に十分な大きさのある対応データセットの両方の典型的なレベルで実行される必要があります。これにより、最善の結果が達成されるようになります。

デフォルトでは、GCC はプロファイルデータをステップ 1 が実行されたディレクトリーに生成します。この情報を別の場所に生成するには、**-fprofile-dir=DIR** でコンパイルします。この**DIR** は作成先となるディレクトリーです。

**警告**

コンパイラーフィードバックデータファイルのフォーマットは、コンパイラーのバージョンにより異なります。コンパイラーの各バージョンでプログラムのコンパイルを繰り返すことが必要となります。

#### 4.1.6.6. 64 ビットホスト上での 32 ビットコンパイラーの使用

64 ビットのホスト上では、GCC は 64 ビットのホスト上でのみ実行可能な実行可能ファイルをビルドします。しかし、GCC を使って 64 ビットと 32 ビットの両方のホスト上で実行可能な実行可能ファイルをビルドすることもできます。

64 ビットのホスト上で 32 ビットのバイナリーをビルドするには、まず実行可能ファイルが必要とするサポート用ライブラリーの 32 ビットバージョンをインストールします。プログラムが C++ プログラムの場合、これは少なくとも **glibc**、**libgcc**、**libstdc++** のサポート用ライブラリーを含める必要があります。Intel 64 および AMD64 では、以下のコマンドを実行します。

```
yum install glibc-devel.i686 libgcc.i686 libstdc++-devel.i686
```

プログラムが必要とする新たな 32 ビットライブラリーをインストールすることが便利な場合もあります。たとえば、プログラムがビルドに **db4-devel** を使用する場合、このライブラリーの 32 ビットバージョンは以下のコマンドでインストールできます。

```
yum install db4-devel.i686
```

### 注記

(**x86-64** ではなく) x86 プラットフォーム上の **.i686** 接尾辞は、当該パッケージの 32 ビットバージョンを指定します。PowerPC アーキテクチャーの場合、接尾辞は (**ppc64** ではなく) **ppc** になります。

32 ビットライブラリーのインストール後は、**-m32** オプションをコンパイラーとリンカーにパスして 32 ビットの実行可能ファイルが作成できます。64 ビットシステム上にサポート用の 32 ビットライブラリーがインストールされていれば、この実行可能ファイルは 32 ビットシステムと 64 ビットシステムの両方で実行できることになります。

**手順4.6 64 ビットホスト上での 32 ビットプログラムのコンパイル**

1. 64 ビットシステム上では、以下のコマンドで **hello.c** を 64 ビットの実行可能ファイルにコンパイルします。

   ```
   gcc hello.c -o hello64
   ```

2. 作成された実行可能ファイルが 64 ビットのバイナリーであることを確認します。

   ```
   $ file hello64
   hello64: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux),
   dynamically linked (uses shared libs), for GNU/Linux 2.6.18, not
   stripped
   $ ldd hello64
   linux-vdso.so.1 =>  (0x00007fff242dd000)
   libc.so.6 => /lib64/libc.so.6 (0x00007f0721514000)
   /lib64/ld-linux-x86-64.so.2 (0x00007f0721893000)
   ```

   64 ビットの実行可能ファイル上では、コマンド **file** の出力に **ELF 64-bit** が含まれ、**ldd** はリンクされたメイン C ライブラリーとして **/lib64/libc.so.6** をリスト表示します。

3. 64 ビットシステム上では、以下のコマンドで **hello.c** を 32 ビットの実行可能ファイルにコンパイルします。

   ```
   gcc -m32 hello.c -o hello32
   ```

4. 作成された実行可能ファイルが 32 ビットのバイナリーであることを確認します。

```
$ file hello32
hello32: ELF 32-bit LSB executable, Intel 80386, version 1
(GNU/Linux), dynamically linked (uses shared libs), for GNU/Linux
2.6.18, not stripped
$ ldd hello32
linux-gate.so.1 =>  (0x007eb000)
libc.so.6 => /lib/libc.so.6 (0x00b13000)
/lib/ld-linux.so.2 (0x00cd7000)
```

   32 ビットの実行可能ファイル上では、コマンド **file** の出力に **ELF 32-bit** が含まれ、**ldd** はリンクされたメイン C ライブラリとして **/lib/libc.so.6** をリスト表示します。

32 ビットのサポート用ライブラリがインストールされていなければ、C コードに対して以下のようなエラーが返されます。

```
$ gcc -m32 hello32.c -o hello32
/usr/bin/ld: crt1.o: No such file: No such file or directory
collect2: ld returned 1 exit status
```

C++ コードの場合は、以下のようなエラーが返されます。

```
$ g++ -m32 hello32.cc -o hello32-c++
In file included from /usr/include/features.h:385,
     from /usr/lib/gcc/x86_64-redhat-
linux/4.4.4/../../../../include/c++/4.4.4/x86_64-redhat-
linux/32/bits/os_defines.h:39,
     from /usr/lib/gcc/x86_64-redhat-
linux/4.4.4/../../../../include/c++/4.4.4/x86_64-redhat-
linux/32/bits/c++config.h:243,
     from /usr/lib/gcc/x86_64-redhat-
linux/4.4.4/../../../../include/c++/4.4.4/iostream:39,
     from hello32.cc:1:
/usr/include/gnu/stubs.h:7:27: error: gnu/stubs-32.h: No such file or
directory
```

これらのエラーは、本セクションの最初の部分で説明されたようにサポート用の 32 ビットライブラリが適切にインストールされていないことを示しています。

また、**-m32** でビルドしても、プログラムを適応もしくは変換して 32/64 ビットの互換性から発生する問題の解決にはならないことに注意してください。移動可能なコードの書き込みと、32 ビットから 64 ビットへの変換に関するヒントは、Proceedings of the 2003 GCC Developers Summit の **Porting to 64-bit GNU/Linux Systems** を参照してください。

### 4.1.7. GCC のドキュメント

GCC コンパイラーについての詳細情報は、**cpp**、**gcc**、**g++**、**gcj**、**gfortran** の各コマンドの **man** ページを参照してください。

また、以下のオンラインユーザーマニュアルも利用可能です。

- GCC 4.4.4 Manual

- GCC 4.4.4 GNU Fortran Manual
- GCC 4.4.4 GCJ Manual
- GCC 4.4.4 CPP Manual
- GCC 4.4.4 GNAT Reference Manual
- GCC 4.4.4 GNAT User's Guide
- GCC 4.4.4 GNU OpenMP Manual

GCC 開発のメインサイトは gcc.gnu.org です。

## 4.2. 分散コンパイル

Red Hat Enterprise Linux 6 は *分散コンパイル* もサポートします。分散コンパイルでは、1 つのコンパイルジョブをいくつもの小さいジョブに変換します。これらのジョブがマシン群に分散されるので、(特に大型のコードベースのプログラムでは) ビルド時間が短縮されます。この機能は、**distcc** パッケージにより提供されます。

分散コンパイルを設定するには、以下のパッケージをインストールします。

- **distcc**
- **distcc-server**

分散コンパイルの詳細情報は、**distcc** および **distccd** の **man** ページを参照してください。以下のリンクでも **distcc** の開発についての詳細情報を提供しています。

http://code.google.com/p/distcc

## 4.3. AUTOTOOLS

GNU Autotools はコマンドラインツールのスイートで、開発者はこれを使うことでインストール済みパッケージや Linux ディストリビューションに関係なく異なるシステム上でアプリケーションをビルドできます。このツールは、開発者が **configure** スクリプトを作成する際の手助けとなります。このスクリプトはビルド前に実行され、アプリケーションのビルドに必要なトップレベルの **Makefile** を作成します。**configure** スクリプトは、現行システム上でテストを実行し、追加ファイルを作成し、ビルダーが提供するパラメーターのように他のディレクティブを実行することができます。

Autotools スイートの中で最も一般的に使用されるものは、以下のとおりです。

**autoconf**

入力ファイル (例: **configure.ac**) から **configure** スクリプトを生成します。

**automake**

特定システム上でプロジェクト用の **Makefile** を作成します。

**autoscan**

予備入力ファイル (つまり、**configure.scan**) を作成します。これは、**autoconf** が使用する最終的な **configure.ac** を作成するために編集可能なものです。

Autotools スイートの全ツールは、**Development Tools** グループパッケージの一部です。このグループパッケージをインストールして Autotools スイート全体をインストールすることも可能ですし、または **yum** を使用してスイートの好みのツールのみをインストールすることもできます。

### 4.3.1. Eclipse 用の Autotools プラグイン

Autotools スイートは、Autotools プラグインで Eclipse IDE にも統合されます。このプラグインは、Autotools 用の Eclipse グラフィカルユーザーインターフェースを提供し、これはほとんどの C/C++ プロジェクトに適したものです。

Red Hat Enterprise Linux 6 では、このプラグインは 以下の 2 つの新規 C/C++ プロジェクト用テンプレートのみをサポートします。

- 空のプロジェクト
- "hello world" アプリケーション

空のプロジェクトのテンプレートは、すでに Autotools をサポートしている C/C++ 開発ツールキットにプロジェクトをインポートする際に使用します。Autotools プラグインへの将来的な更新には、共有ライブラリーや他の複雑なシナリオを作成するための新たなグラフィカルユーザーインターフェース (たとえば、ウィザード) が含まれます。

Autotools プラグインの Red Hat Enterprise Linux 6 バージョンでは、**git** や **mercurial** は Eclipse に統合されません。このため、**git** リポジトリーを使用する Autotools プロジェクトは、Eclipse のワークスペース **外** でチェックアウトする必要があります。その後に Eclipse 内のそのようなプロジェクトのソースの場所を特定することができます。すべてのリポジトリー操作 (コミット、更新など) は、コマンドラインで行います。

### 4.3.2. 設定スクリプト

Autotools の最も重要な機能は、**configure** スクリプトの作成です。このスクリプトは、ツール、入力ファイル、その他の機能がプロジェクトの構築に使用できるようにシステムをテストします。[2]。**configure** スクリプトは **Makefile** を生成します。これは、システム設定に基づいて**make** ツールがプロジェクトをビルドできるようにします。

**configure** スクリプトを作成するには、ます入力ファイルを作成します。そして、それを Autotools ユーティリティーにフィードして **configure** スクリプトを作成します。この入力ファイルは通常 **configure.ac** または **Makefile.am** です。前者は通常 **autoconf** が処理を行い、後者は **automake** にフィードされます。

**Makefile.am** 入力ファイルが利用可能な場合は、**automake** ユーティリティーが **Makefile** テンプレート (つまり、**Makefile. in**) を作成します。これは、設定時に収集された情報を参照する場合があります。たとえば、**Makefile** は特定のライブラリーにリンクする必要がある場合がありますが、これはそのライブラリーがすでにインストールされている **場合のみ** です。**configure** スクリプトが実行されると、**automake** が **Makefile. in** テンプレートを使用して **Makefile** を作成します。

代わりに **configure.ac** ファイルが利用可能な場合は、**configure.ac** が起動したマクロに基づいて **autoconf** が自動的に **configure** スクリプトを作成します。予備の**configure.ac** を作成するには、**autoscan** ユーティリティーを使用してファイルを適時編集します。

### 4.3.3. Autotools のドキュメント

Red Hat Enterprise Linux 6 には、**autoconf**、**automake**、**autoscan** および Autotools スイートに含まれるほとんどのツールの **man** ページが含まれています。また、Autotools コミュニティーは **autoconf** および **automake** に関する幅広いドキュメントを以下の Web サイトで提供しています。

- http://www.gnu.org/software/autoconf/manual/autoconf.html
- http://www.gnu.org/software/autoconf/manual/automake.html

以下のサイトは Autotools の使用方法を説明しているオンラインブックです。上記のオンラインドキュメントは Autotools に関する最新情報で推奨されますが、このオンラインブックもすぐれた代替手段で入門書となります。

- http://sourceware.org/autobook/

Autotools 入力ファイルの作成方法に関する情報は、以下のサイトを参照してください。

- http://www.gnu.org/software/autoconf/manual/autoconf.html#Making-configure-Scripts
- http://www.gnu.org/software/autoconf/manual/automake.html#Invoking-Automake

以下のアップストリームの例も簡単な **hello** プログラムでの Autotools の使用を説明しています。

- http://www.gnu.org/software/hello/manual/hello.html

## 4.4. ECLIPSE BUILT-IN SPECFILE EDITOR

Eclipse 用の Specfile Editor プラグインは、開発者が **.spec** ファイルを管理する際に役立つ機能を提供します。このプラグインを使うと、ユーザーは **.spec** ファイルの編集の際に、オートコンプリート機能やハイライト、ファイルのハイパーリンク、折り曲げなどのいくつかの Eclipse GUI 機能を活用することができます。

また、Specfile Editor プラグインは **rpmlint** ツールを Eclipse インターフェースに統合します。**rpmlint** はコマンドラインツールで、開発者が一般的な RPM パッケージエラーを検出する際に役立ちます。Eclipse インターフェースが提供する豊富な仮想化は、**rpmlint** がレポートするミスを開発者が迅速に検出、表示、訂正する際に役立ちます。

Eclipse 用の Specfile Editor は、**eclipse-rpm-editor** パッケージで提供されます。このプラグインについての詳細は、Eclipse **ヘルプコンテンツ**の 『Specfile Editor User Guide』 を参照してください。

## 4.5. ECLIPSE の CDT

*CDT* (C/C++ 開発ツール) は、Eclipse で C および C++ プロジェクトを開発する際のサポートを追加する Eclipse プロジェクトです。ユーザーは以下の 3 つの形式のプロジェクトを作成できます。

1. Managed Make プロジェクト
2. Standard Make プロジェクト
3. Autotools プロジェクト

### 4.5.1. Managed Make プロジェクト

managed make CDT プロジェクトは管理プロジェクトとも呼ばれるもので、プロジェクトのビルド方法に関する詳細がエンドユーザーのために自動化されているものです。これは、別のタイプの CDT C/C++ プロジェクトである standard make プロジェクトとは異なるものです。standard make プロ

ジェクトでは、ビルドの詳細が指定されている Makefile をユーザーが提供します。

管理プロジェクトでは、管理プロジェクトのタイプと必要なツールチェーンを選択して開始します。プロジェクトのタイプは、実行可能ファイルや共有ライブラリー、静的ライブラリーといったプロジェクトの最終的なターゲットを基にカテゴリ分けされます。これらのカテゴリ内には、ベースソースファイルが既に提供されているより特定されたプロジェクト (たとえば、hello world サンプルの実行可能プロジェクト) 用のテンプレートがある場合があります。これらはさらにカスタマイズが可能です。

ツールチェーンは、ターゲット生成に使用されるツールセットです。通常、Red Hat Enterprise Linux の C/C++ 開発者は、コンパイルやリンク、アセンブリーに GCC を使用する Linux GCC ツールチェーンを選択します。ツールチェーンの各ツールは、通常ファイル接尾辞 (例えば、**.c** や **.h**、**.S**) やファイル名で指定されている 1 つ以上の入力タイプに関連付けられています。このツールには開発者がカスタマイズできるパラメーター設定があり、各ツールには作成される出力タイプがあります。また、関連付けられるコマンドもしくはバイナリ実行可能ファイルがあり、これは複数のツールで重複する場合があります。例えば、C コンパイラーとリンカーは両方とも GCC を使用できますが、コンパイラーとリンカーのツールではそれぞれ出入力タイプが異なり、開発者に提示される設定も異なります。ツール設定をカスタマイズするには、**Properties > C/C++ Build > Settings** に移動します。ツールチェーン自体のカスタマイズは、**Properties > C/C++ Build > Toolchain Editor** で使用ツールを追加、削除、置き換えることで可能です。

ソースファイルやヘッダーファイルなどの新規ファイルは、作成後にプロジェクトに追加できます。新規ファイルは、入力タイプとツール設定に基づいて自動的にビルドに追加されます。管理プロジェクトがプロジェクトと共に配布可能な Makefile を生成するには、**Builder Settings** タブにある **Project > C/C++ Build** に移動します。これで、Eclipse 外での Makefile の使用が容易になります。

managed make C/C++ プロジェクトの詳細情報については、『C/C++ Development User Guide』 を参照してください。**Concepts > Project Types, Tasks > Creating a Project** か **Reference > C/C++ Properties > C/C++ Project Properties > C/C++ Build > Settings Page** に移動します。

### 4.5.2. Standard Make プロジェクト

standard make CDT プロジェクトは、開発者が手動で管理する Makefile による従来の C プロジェクトです。managed make プロジェクトとは違い、Makefile における規則決定で使用するツール設定はありません。ビルドの一部として処理される新規ソースファイルがプロジェクトに追加される際は、Makefile を手動で追加する必要があります。新規ファイル名が一致するパターン規則が存在する場合 (たとえば、**.c:.o**。これは、接尾辞 **.c** の付いたファイルを接尾辞 **.o** の付いたファイルに処理する方法を提示)、これは必要ありません。

プロジェクトのビルドにおけるデフォルトの make ターゲットは **all** で、プロジェクト消去用のデフォルトの make ターゲットは **clean** です。また、ユーザーが Makefile で見つかった他のターゲットをビルドすることもできます。これを行うには、Makefile Target ダイアログを使用して既存のものを実行もしくはビルドするためのターゲットを作成します。Makefile Target ダイアログでは、makefile で見つかった複数のターゲットをグループ化する仮のターゲットを特定の順番で作成することもできます。特定の作成およびビルドダイアログにアクセスするにはそれぞれ、**Project > Make Target > Create...** もしくは **Project > Make Target > Build...** に移動します。別の方法では、プロジェクトの **resources** で 右クリック をして **Make Targets** オプションを選択し、**Create...** または **Build...** にアクセスします。

standard make C/C++ プロジェクトに関する詳細情報は、C/C++ Development ユーザーガイドを参照してください。**Concepts > Project Types, Tasks > Creating a project** か **Reference > C/C++ Properties > C/C++ Project Properties > C/C++Build > Settings Page** に移動します。

### 4.5.3. Autotools プロジェクト

autotools プロジェクトは standard make プロジェクトに非常に似ていますが、Makefile は通常、ビルド前に発生する設定手順の一部として生成されます。このタイプのプロジェクトに対するサポートを追

加する Autotools および Autotools プラグインの詳細に関しては、「Autotools」 を参照してください。standard make プロジェクトのように、make ターゲットは Make Target ダイアログで実行できます。

## 4.6. BUILD-ID バイナリーの一意の ID

Red Hat Enterprise Linux Server 6 およびそれ以降でビルドされた実行可能ファイルや共有ライブラリーにはそれぞれ、160 ビットの SHA-1 文字列である一意の ID が割り当てられます。これは、バイナリーの選択部分のチェックサムとして生成されたものです。これにより、同一ホスト上の同一プログラムの 2 つのビルドが常に一貫性のある build-id とバイナリーコンテンツを作成できるようになります。

以下のコマンドで、バイナリーの build-id を表示します。

```
$ eu-readelf -n /bin/bash
[...]
Note section [ 3] '.note.gnu.build-id' of 36 bytes at offset 0x274:
  Owner  Data size    Type
  GNU         20    GNU_BUILD_ID
    Build ID: efdd0b5e69b0742fa5e5bad0771df4d1df2459d1
```

バイナリーの一意の ID は、「コアファイル解析用の Debuginfo パッケージのインストール」にあるように、コアファイルの分析などの場合に便利です。

## 4.7. SOFTWARE COLLECTIONS および SCL-UTILS

Software Collections を使うと、システム上で複数バージョンの同一 RPM パッケージをビルドして、同時にインストールすることが可能になります。Software Collections は、従来の RPM パッケージマネジャーがインストールするパッケージのシステムバージョンに影響を与えません。

システム上で Software Collections のサポートを有効にするには、シェルプロンプトで **root** として以下を入力して scl-utils パッケージをインストールします。

```
~]# yum install scl-utils
```

scl-utils パッケージは **scl** ツールを提供します。これは Software Collection を有効にし、Software Collection 環境でアプリケーションを実行します。

**scl** ツールの一般的な使用方法は、以下の構文の使用で説明できます。

```
scl action software_collection_1 software_collection_2 command
```

**例4.1 アプリケーションの直接実行する**

**software_collection_1** と呼ばれる Software Collection **--version** オプションで **Perl** を直接実行するには、以下のコマンドを実行します。

```
scl enable software_collection_1 'perl --version'
```

**例4.2 複数の Software Collection を有効にして、シェルを実行する**

複数の Software Collection を有効にした環境で **Bash** シェルを実行するには、以下のコマンドを実行します。

```
scl enable software_collection_1 software_collection_2 bash
```

上記のコマンドは、**software_collection_1** および **software_collection_2** と呼ばれる 2 つの Software Collection を有効にします。

**例4.3 ファイルに保存されたコマンドを実行する**

ファイルに保存された多くのコマンドを Software Collection 環境で実行するには、以下のコマンドを実行します。

```
cat cmd | scl enable software_collection_1 -
```

上記のコマンドは、**software_collection_1** と呼ばれる Software Collection 環境の **cmd** ファイルに保存されているコマンドを実行します。

Software Collections および **scl-utils** に関する詳細情報は、Red Hat Software Collections 1.2 Beta Packaging Guide を参照してください。

---

[2] **configure** が実行可能なテストについての詳細情報は以下のリンクを参照してください

http://www.gnu.org/software/autoconf/manual/autoconf.html#Existing-Tests

# 第5章 デバッグ

通常、有用で質の高いソフトウェアは、アプリケーション開発の複数フェーズを経て作成されるため、その間にミスについての十分な機会が設けられます。一部のフェーズには、エラーを検出する一連のメカニズムがそれぞれ設定されています。たとえば、変数や関数などのオブジェクトが適切に記述されていることを確認するために、コンパイル時に基本的なセマンティクス解析が行われます。

アプリケーション開発の各フェーズで実行されるエラーチェックのメカニズムは、コード内の単純で明らかな誤りを見つけることを目的としています。一方、デバッグフェーズは、所定のコード検査で見過ごされるより見つけにくいエラーを明らかにする際に役に立ちます。

## 5.1. ELF の実行可能バイナリー

Red Hat Enterprise Linux は、実行可能バイナリー、共有ライブラリーまたは debuginfo ファイルに ELF を使用し、これらの debuginfo ELF ファイル内では DWARF フォーマットが使用されます。DWARF のバージョン 3 が ELF ファイルで使用されます (**gcc -g** は **gcc -gdwarf-3** と同等)。DWARF debuginfo には以下が含まれます。

- バイナリーのターゲットアドレスを含む、コンパイルされた関数および変数すべての名前
- ソース行番号を含む、コンパイルに使用されるソースファイル
- ローカル変数の場所

**重要**

STABS はたまに UNIX で使用されますが、より古く、機能性に乏しいフォーマットです。Red Hat ではこの使用を奨励していません。GCC および GDB での STABS の作成と使用のサポートは可能な範囲でのみ行なわれます。

これらの ELF ファイルでは、GCC debuginfo のレベルも使用されます。デフォルトはレベル 2 であり、ここではマクロ情報が表示されません。レベル 3 には C/C++ マクロ定義が含まれますが、この設定ではデバッグ情報が非常に大きくなる可能性があります。デフォルト **gcc -g** のコマンドは **gcc -g2** と同一です。マクロ情報をレベル 3 に変更するには、**gcc -g3** を使用します。

利用可能な debuginfo のレベルは複数あります。ファイルでどのセクションが使用されているかを確認するには、コマンド **readelf -WS *file*** を使用します。

**表5.1 debuginfo のレベル**

| バイナリーの状態 | コマンド | 注意事項 |
|---|---|---|
| Stripped (削除) | **strip *file***<br>または<br>**gcc -s -o *file*** | 共有ライブラリーとのランタイムリンクに必要なシンボルのみが表示されます。<br>使用される ELF セクション: **.dynsym** |

| バイナリーの状態 | コマンド | 注意事項 |
|---|---|---|
| ELF シンボル | **gcc -o *file*** | 関数および変数の名前のみが表示され、ソースファイルのバインディングおよびタイプはありません。<br><br>使用される ELF セクション: **.symtab** |
| DWARF debuginfo (マクロあり) | **gcc -g -o *file*** | タイプも含め、ソースファイル名および行番号は認識されます。<br><br>使用される ELF セクション: **.debug_*** |
| DWARF debuginfo (マクロあり) | **gcc -g3 -o *file*** | **gcc -g** と似ていますが、マクロが GDB に認識されます。<br><br>使用される ELF セクション: **.debug_macro** |

### 注記

GDB はソースファイルを解釈することはなく、テキストとしてそれらを表示するのみです。情報を DWARF に保存するには **gcc -g** とその変数を使用します。

**gcc -rdynamic** を使用してプログラムやライブラリーをコンパイルすることは奨励されません。特定のシンボルについては、**gcc -Wl, --dynamic-list=...** を代わりに使用してください。**gcc -rdynamic** が使用される場合、**strip** コマンドや **-s** gcc オプションの効果は一切ありません。それは、共有ライブラリーとのランタイムリンケージについてのすべての ELF シンボルがバイナリーで保持されるためです。

ELF シンボルは、**readelf -s *file*** コマンドによって読み取られます。

DWARF シンボルは、**readelf -w *file*** コマンドによって読み取られます。

コマンド **readelf -wi *file*** は、プログラム内でコンパイルされた debuginfo の優れた検証コマンドです。コマンド **strip *file*** または **gcc -s** は、プログラムのコンパイルのさまざまなステージの出力で誤って実行されます。

**readelf -w *file*** コマンドは、フォーマットを持つ **.eh_frame** と呼ばれる特殊セクションを表示するために使用でき、その目的は DWARF セクションの **.debug_frame** と類似しています。**.eh_frame** セクションは、ランタイム C++ 例外の解決に使用され、**-g** gcc オプションが使用されない場合であっても存在します。これはプライマリ RPM に保存され、debuginfo RPM には存在しません。

Debuginfo RPM には、**.symtab** および **.debug_*** セクションが含まれます。**.eh_frame**、**.eh_frame_hdr**、または **.dynsym** のいずれのセクションもプログラムのランタイム時に必要になるため、debuginfo RPM には移行されず、またこの中には存在しません。

## 5.2. DEBUGINFO パッケージのインストール

Red Hat Enterprise Linux は、オペレーティングシステムに含まれるアーキテクチャー依存の RPM 用の **-debuginfo** パッケージも提供します。***packagename*-debuginfo-*version*-*release*.*architecture*.rpm** パッケージには、パッケージソースファイルと最終的なインストール済みバイナリの関係についての詳細情報が含まれます。debuginfo パッケージには両方の **.debug** ファイルが含まれ、これらには DWARF debuginfo とバイナリパッケージのコンパイルに使用されるソースファイルが含まれます。

### 注記

debuginfo と同等の情報がインストールされていない場合にパッケージのデバッグを試行しても、デバッガー機能のほとんどは機能しません。例えば、エクスポートされた共有ライブラリー関数の名前は利用可能ですがが、これらに一致するソースファイル行は debuginfo パッケージがインストールされていないと使用できません。

お使いのプログラムに **gcc** コンパイルオプション **-g** を使用してください。デバッグのエクスペリエンスは、最適化 (**-O2** などの gcc オプション **-O**) が **-g** と一緒に適用される場合に向上します。

Red Hat Enterprise Linux 6 では、debuginfo パッケージが Red Hat Network の新規チャンネルで利用できるようになりました。パッケージ (通常は ***packagename*-debuginfo**) の **-debuginfo** パッケージをインストールするには、まずマシンが対応する Debuginfo チャンネルをサブスクライブする必要があります。例えば、Red Hat Enterprise Server 6 の場合、対応するチャンネルは **Red Hat Enterprise Linux Server Debuginfo (v. 6)** になります。

Red Hat Enterprise Linux システムパッケージは、最適化 (gcc オプション **-O2**) でコンパイルされています。これは、いくつかの変数が **<optimized out>** と表示されることを意味します。コードのステップスルーによる多少の「ジャンプ」がありますが、クラッシュを分析することができます。最適化のために一部の情報が欠落する場合、コードを逆アセンブルし、ソースに手動で一致させることにより、正しい変数情報を得ることができます。ただし、これは例外ケースにのみ適用され、通常のデバッグには適していません。

システムパッケージの場合、GDB はその機能を制限する debuginfo パッケージの欠落があるかどうかをユーザーに通知します。

```
gdb ls
[...]
Reading symbols from /bin/ls...(no debugging symbols found)...done.
Missing separate debuginfos, use: debuginfo-install coreutils-8.4-
16.el6.x86_64
(gdb) q
```

デバッグされるシステムパッケージが既知である場合、上記の GDB で提案されるコマンドを使用します。これにより、*packagename* が依存するすべてのデバッグパッケージが自動的にインストールされます。

```
# debuginfo-install packagename
```

### 5.2.1. コアファイル解析用の Debuginfo パッケージのインストール

コアファイルは、プロセスのクラッシュ時のメモリーイメージを表すものです。システムプログラムのクラッシュをバグ報告する場合、Red Hat では 『Red Hat 導入ガイド』の『自動バグ報告ツール』の章で説明されている ABRT ツールの使用を推奨しています。ABRT が目的に適さない場合のために、

ABRT で自動化されるステップを以下に説明します。

プロセスのクラッシュ時に **ulimit -c unlimited** 設定が使用される場合、コアファイルは現在のディレクトリーにダンプされます。コアファイルには、プロセスによってディスクファイルの元の状態から修正されたメモリ領域のみが含まれます。クラッシュの完全な解析を実行するには、コアファイルに以下が含まれている必要があります。

- コアファイル自体
- **/usr/sbin/sendmail** などのクラッシュした実行可能バイナリー
- クラッシュ時にバイナリーでロードされたすべての共有ライブラリー
- 実行可能ファイルおよびそのロードされたすべてのライブラリー用の .debug ファイルおよびソースファイル (どちらも debuginfo RPM に保存されている)

適切な解析を行うには、関係するすべての RPM 用の正確な ***version-release.architecture*** か、または独自にコンパイルしたバイナリーの同じビルドが必要になります。クラッシュ時に、アプリケーションがすでにディスク上の **yum** によって再コンパイルされているか、または更新されている場合、コアファイルの解析に適してないファイルのレンダリングが行われます。

コアファイルには、関連するすべてのバイナリーの build-id が含まれます。build-id についての詳細は、「build-id バイナリーの一意の ID」 を参照してください。コアファイルのコンテンツは以下のように表示されます。

```
$ eu-unstrip -n --core=./core.9814
0x400000+0x207000 2818b2009547f780a5639c904cded443e564973e@0x400284
/bin/sleep /usr/lib/debug/bin/sleep.debug [exe]
0x7fff26fff000+0x1000
1e2a683b7d877576970e4275d41a6aaec280795e@0x7fff26fff340 . - linux-
vdso.so.1
0x35e7e00000+0x3b6000
374add1ead31ccb449779bc7ee7877de3377e5ad@0x35e7e00280 /lib64/libc-
2.14.90.so /usr/lib/debug/lib64/libc-2.14.90.so.debug libc.so.6
0x35e7a00000+0x224000
3ed9e61c2b7e707ce244816335776afa2ad0307d@0x35e7a001d8 /lib64/ld-2.14.90.so
/usr/lib/debug/lib64/ld-2.14.90.so.debug ld-linux-x86-64.so.2
```

各行に含まれるそれぞれの列の意味は以下のようになります。

- 特定のバイナリーがマップされるメモリー内アドレス (たとえば、最初の行の **0x400000**)。
- バイナリーのサイズ (たとえば、最初の行の **+0x207000**)。
- バイナリーの 160-bit SHA-1 build-id (たとえば、最初の行の **2818b2009547f780a5639c904cded443e564973e**)。
- build-id バイトが保存されていたメモリ内アドレス (たとえば、最初の行の **@0x400284**)。
- ディスク上のバイナリーファイル (ある場合)(たとえば、最初の行の **/bin/sleep**)。これは、このモジュールの **eu-unstrip** で検索されました。
- ディスク上の debuginfo ファイル (ある場合)(たとえば、**/usr/lib/debug/bin/sleep.debug**)。ただし、バイナリーファイル参照を代わりに使用するのがベストプラクティスになります。

- コアファイルの共有ライブラリーリストに保存される共有ライブラリー名 (たとえば、3 行目の **libc.so.6**)。

それぞれの build-id (例えば、**ab/cdef0123456789012345678901234567890123**) では、シンボリックリンクがその debuginfo RPM に組み込まれます。上記の **/bin/sleep** 実行可能ファイルを例にすると、**coreutils-debuginfo** RPM には、他のファイルの中でもとりわけ以下が含まれます。

```
lrwxrwxrwx 1 root root 24 Nov 29 17:07 /usr/lib/debug/.build-
id/28/18b2009547f780a5639c904cded443e564973e -> ../../../../../bin/sleep*
lrwxrwxrwx 1 root root 21 Nov 29 17:07 /usr/lib/debug/.build-
id/28/18b2009547f780a5639c904cded443e564973e.debug ->
../../bin/sleep.debug
```

ケースによっては (コアファイルのロードなど)、GDB は、***name*-debuginfo-*version*-*release*.rpm** パッケージ の名前、バージョンまたはリリースを認識しません。このような場合に、GDB は異なるコマンドを提案します。

```
gdb -c ./core
[...]
Missing separate debuginfo for the main executable filename
Try: yum --disablerepo='*' --enablerepo='*debug*' install
/usr/lib/debug/.build-id/ef/dd0b5e69b0742fa5e5bad0771df4d1df2459d1
```

バイナリーパッケージ *packagename*-debuginfo-*version-release.architecture*.rpm の *version-release.architecture* は完全一致である必要があります。それが異なる場合、GDB は debuginfo パッケージを使用することができません。異なるビルドの同じ *version-release.architecture* であっても、debuginfo パッケージの互換性がなくなる可能性があります。GDB が debuginfo が不足していることを報告する場合は、以下を必ず再チェックしてください。

**rpm -q *packagename packagename*-debuginfo**

*version-release.architecture* 定義が一致する必要があります。

**rpm -V *packagename packagename*-debuginfo**

このコマンドは、*packagename* などの修正された可能性のある設定ファイルを除いて一切の出力を行いません。

**rpm -qi *packagename packagename*-debuginfo**

*version-release.architecture* は、Vendor、Build Date、および Build Host が一致する情報を表示する必要があります。Red Hat Enterprise Linux RPM パッケージ用の CentOS debuginfo RPM を使用しても機能しません。

必要な build-id が既知である場合、以下のコマンドはどの RPM にそれが含まれるかを照会します。

```
$ repoquery --disablerepo='*' --enablerepo='*-debug*' -qf
/usr/lib/debug/.build-id/ef/dd0b5e69b0742fa5e5bad0771df4d1df2459d1
```

たとえば、コアファイルに一致する実行可能ファイルのバージョンは、以下でインストールできます。

```
# yum --enablerepo='*-debug*' install $(eu-unstrip -n --core=./core.9814 |
sed -e 's#^[^ ]* \(..\)\([^@ ]*\).*$#/usr/lib/debug/.build-id/\1/\2#p' -e
's/$/.debug/')
```

バイナリーが RPM にパッケージされておらず、yum リポジトリーに保存されていない場合は同様のメソッドを使用することができます。**/usr/bin/createrepo** を使用し、カスタムアプリケーションビルドでローカルリポジトリーを作成することができます。

## 5.3. GDB

基本的に、大半のデバッガーのように、GDB は非常に厳密に制御された環境でコンパイルされたコードの実行を管理します。この環境は、GDB の操作に必要な以下の基本的なメカニズムを可能にします。

- デバッグされるコード内のメモリの検査および修正 (たとえば、変数の読み取りおよび設定)。
- 主に実行中かまたは停止しているかなどの、デバッグされるコードの実行状態の制御。
- コードの特定セクションの実行の検出 (プログラマーの関心のある特定エリアに達した際にコードの実行を停止するなど)。
- メモリの特定領域へのアクセスの検出 (指定された変数にアクセスした際にコードの実行を停止するなど)。
- 制御された方法による (それ以外の停止されたプログラムから) のコードの複数部分の実行。
- シグナルなどのプログラミングによる非同期イベントの検出。

これらのメカニズムの操作は、コンパイラーが生成する情報にほぼ依存します。たとえば、変数の値を表示するには、GBD は以下を認識する必要があります。

- メモリ内の変数の場所
- 変数の性質

つまり、倍精度の浮動小数点値を表示するには、文字ストリングを表示する場合とは非常に異なるプロセスが必要であることを意味しています。構造などの複雑なケースでは、GDB は構造内の個々のエレメントの特徴だけでなく、構造の形態も認識する必要があります。

GDB が完全に機能するには以下のアイテムが必要です。

**デバッグ情報**

GDB の操作の多くは、プログラムの*デバッグ情報*に依存します。この情報は通常コンパイラーから発生するものですが、これらの多くは、プログラムのデバッグ中にのみ必要になります。つまり、プログラムの通常の実行時には使用されません。この理由により、コンパイラーは常にデフォルトでこの情報を利用可能にする訳ではありません。たとえば、GCC は **-g** フラグを使ってこのデバッグ情報を提供するために明示的に指示される必要があります。

GDB の機能を完全に利用するには、GDB でデバッグ情報を利用できるようにすることを**強くお勧め**します。GDB は、利用可能なデバッグ情報がない状態でコードに対して実行される場合にその使用が**非常に限られます**。

**ソースコード**

GDB (またはそれ以外のデバッガー) の最も役に立つ機能の 1 つに、プログラム実行内のイベントと状況を、ソースコードのこれらに対応するロケーションに関連付ける機能があります。このロケーションは、通常はソースファイルの特定の行または一連の行を参照します。当然これを行うには、プログラムのソースコードがデバッグ時に GDB で利用可能であることが必要です。

### 5.3.1. 単純な GDB

GDB には文字通り数十のコマンドが含まれます。このセクションでは、最も基本的なコマンドを説明します。

**br (breakpoint)**

breakpoint コマンドは、GDB に対し実行内の指定ポイントに達すると実行を一時停止するように指定します。このポイントは数多くの方法で指定することができますが、最も一般的な方法は、単にソースファイルの行番号か、または関数の名前になります。任意の数のブレークポイントを同時に有効にすることができます。これは、しばしば GDB の開始後に発行される最初のコマンドになります。

**r (run)**

**run** コマンドはプログラムの実行を開始します。**run** を任意の引数を使って実行すると、それらの引数は、プログラムが正常に開始されているかのように実行可能ファイルに渡されます。通常、ユーザーはブレークポイントを設定した後にこのコマンドを発行します。

実行可能ファイルを開始する前か、または実行可能ファイルがたとえばブレークポイントで停止する場合、プログラムの状態を多くの面から検査することができます。以下のコマンドは、検査に使用できるより一般的な方法です。

**p (print)**

**print** コマンドは、指定された引数の値を表示し、この引数にはプログラムに関連するほぼすべてのものを使用できます。通常引数は、単純な単一値から構造に至る、様々な複雑度を持つ変数の名前です。また引数は、プログラム変数やライブラリー関数、またはテスト中のプログラムで定義される関数の使用を含む、現在の言語で有効な式として使用することもできます。

**bt (backtrace)**

**backtrace** は、実行が停止されるまで使用される関数呼び出しのチェーンを表示します。これは、定義することが難しい原因を持つ重大なバグ (セグメント化障害など) を調査する際に役に立ちます。

**l (list)**

実行が停止すると、**list** コマンドは、プログラムが停止した場所に対応するソースコードの行を表示します。

停止したプログラムの実行は、数多くの方法で再開することができます。以下は最も一般的なものです。

**c (continue)**

**continue** コマンドは、プログラムの実行を再開し、プログラムがブレークポイントが検出するか、特定または緊急の条件 (エラーなど) を検出するか、または停止するまで継続して実行されます。

**n (next)**

**continue** のように、**next** コマンドも実行を再開しますが、**continue** コマンドの暗黙的な停止条件のほかに、**next** は、現在のソースファイル内のコードの次のシーケンシャル行でも実行を停止します。

**s (step)**

**next** のように、**step** コマンドも、現在のソースファイル内のコードのそれぞれのシーケンシャル行で実行を停止します。ただし、実行が **関数呼び出し** を含むソース行で現在停止している場合、GDB は関数呼び出しを入力した後に (それを実行するのではなく) 実行を停止します。

**fini (finish)**

前述のコマンドのように、**finish** コマンドは実行を再開しますが、実行が関数から返されると停止します。

最後に、以下は 2 つの基本的なコマンドになります。

**q (quit)**

これは実行を終了させます。

**h (help)**

**help** コマンドにより、数多くの内部ドキュメントにアクセスできます。このコマンドでは引数を指定でき、**help breakpoint** (または **h br**) は、 **breakpoint** コマンドの詳細な説明を表示します。さらに詳しい情報は、各コマンドの **help** 出力を参照してください。

### 5.3.2. GDB の実行

このセクションでは、以下の単純なプログラムを使った GDB の基本的な実行について説明します。

**hello.c**

```
#include <stdio.h>

char hello[] = { "Hello, World!" };

int
main()
{
  fprintf (stdout, "%s\n", hello);
  return (0);
}
```

以下の手順は、最も基本的な形式でデバッグプロセスを説明しています。

**手順5.1 'Hello World' プログラムのデバッグ**

1. 以下のように、デバッグフラグを使った実行可能ファイルに hello.c をコンパイルします。

   ```
   gcc -g -o hello hello.c
   ```

   生成されるバイナリー **hello** が **hello.c** と同じディレクトリーにあることを確認します。

2. **hello** バイナリーで **gdb** を実行します (つまり **gdb hello**)。

3. いくつかの導入部のコメントの後に、**gdb** はデフォルトの GDB プロンプトを表示します。

   ```
   (gdb)
   ```

4. 変数 **hello** はグローバルであるため、**メインの**手順が開始する前に表示することができます。

```
gdb) p hello
$1 = "Hello, World!"
(gdb) p hello[0]
$2 = 72 'H'
(gdb) p *hello
$3 = 72 'H'
(gdb)
```

**print** は **hello[0]** をターゲットとしており、 ***hello** には ***(hello + 1)** などのような式の評価が必要になることに注意してください。

```
(gdb) p *(hello + 1)
$4 = 101 'e'
```

5. 次に、ソースをリストします。

```
(gdb) l
1       #include <stdio.h>
2
3       char hello[] = { "Hello, World!" };
4
5       int
6       main()
7       {
8         fprintf (stdout, "%s\n", hello);
9         return (0);
10      }
```

**list** は、**fprintf** 呼び出しが 8 行目にあることを示しています。その行にブレークポイントを適用して、コードを再開します。

```
(gdb) br 8
Breakpoint 1 at 0x80483ed: file hello.c, line 8.
(gdb) r
Starting program: /home/moller/tinkering/gdb-manual/hello

Breakpoint 1, main () at hello.c:8
8         fprintf (stdout, "%s\n", hello);
```

6. 最後に、**next** コマンドを使用して、**fprintf** 呼び出しの後に進んでこれを実行します。

```
(gdb) n
Hello, World!
9         return (0);
```

以下のセクションでは、GDB のより複雑なアプリケーションについて説明します。

### 5.3.3. 条件付きブレークポイント

現実の多くの場面で、プログラムは最初の数千回はタスクを適切に実行するものの、タスクの反復回数が 8,000 回に達するとクラッシュし始めたり、エラーが検出される場合があります。プログラマーが

クラッシュした反復に達するためだけに **continue** コマンドを数千回も辛抱強く実行するとは想像し難いため、このようなデバッグプログラムは困難なものです。

このような状況は現実ではよくあります。そのため、GDB はプログラマーが条件をブレークポイントに付加できるようにします。たとえば、以下のプログラムを見てみましょう。

**simple.c**

```
#include <stdio.h>

main()
{
  int i;

  for (i = 0;; i++) {
fprintf (stdout, "i = %d\n", i);
  }
}
```

GDB プロンプトで条件付きブレークポイントを設定するには、以下のようにします。

```
(gdb) br 8 if i == 8936
Breakpoint 1 at 0x80483f5: file iterations.c, line 8.
(gdb) r
```

この条件により、プログラムは次の出力を表示して停止します。

```
i = 8931
i = 8932
i = 8933
i = 8934
i = 8935

Breakpoint 1, main () at iterations.c:8
8           fprintf (stdout, "i = %d\n", i);
```

ブレークポイントの状況を確認するためにブレークポイントの情報を検査します **(info br** を使用する)。

```
(gdb) info br
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x080483f5 in main at iterations.c:8
        stop only if i == 8936
        breakpoint already hit 1 time
```

### 5.3.4. フォークされる実行

プログラマーが直面するバグの中でも、1 つのプログラム **(親)** がそれ自体の独立したコピー **(フォーク)** を作成するケースに関連するバグがとりわけ困難になります。そのフォークは子プロセスを作成し、その結果失敗してしまいます。親プロセスのデバッグは役に立つ場合とそうでない場合があります。バグに辿り着く唯一の方法は子プロセスのデバッグである場合がよくありますが、これは常に可能とは限りません。

**set follow-fork-mode** 機能は、この障害を克服するために使用され、プログラマーが親プロセスの

代わりに子プロセスに従うことを可能にします。

**set follow-fork-mode parent**

元のプロセスがフォークの後にデバッグされます。子プロセスは問題なく実行されます。これはデフォルトです。

**set follow-fork-mode child**

新規プロセスがフォークの後にデバッグされます。親プロセスは問題なく実行されます。

**show follow-fork-mode**

フォーク呼び出しに対する現在のデバッガー応答を表示します。

**set detach-on-fork** コマンドを使用して、フォーク後に親プロセスと子プロセスの両方をデバッグするか、またはそれらの両方に対してデバッガー制御を保持します。

**set detach-on-fork on**

子プロセス (または **follow-fork-mode** の値によっては親プロセス) が切り離され、別個に実行できるようになります。これはデフォルトです。

**set detach-on-fork off**

両方のプロセスが GDB の制御下に保持されます。1 つのプロセス (**follow-fork-mode** の値に応じて子または親のどちらか) のデバッグが、他方が一時停止している間に通常どおり実行されます。

**show detach-on-fork**

**detach-on-fork** モードがオンであるか、またはオフであるかどうかを表示します。

以下のプログラムを見てみましょう。

**fork.c**

```
#include <unistd.h>

int main()
{
  pid_t  pid;
  const char *name;

  pid = fork();
  if (pid == 0)
    {
      name = "I am the child";
    }
  else
    {
      name = "I am the parent";
    }
  return 0;
}
```

このプログラムは、コマンド **gcc -g fork.c -o fork -lpthread** でコンパイルされ、GDB で検査された後に、以下を表示します。

```
gdb ./fork
[...]
(gdb) break main
Breakpoint 1 at 0x4005dc: file fork.c, line 8.
(gdb) run
[...]
Breakpoint 1, main () at fork.c:8
8   pid = fork();
(gdb) next
Detaching after fork from child process 3840.
9   if (pid == 0)
(gdb) next
15      name = "I am the parent";
(gdb) next
17   return 0;
(gdb) print name
$1 = 0x400717 "I am the parent"
```

GDB は親プロセスに従い、子プロセス (プロセス 3840) の実行継続を許可します。

以下は、**set follow-fork-mode child** を使用した同じテストです。

```
(gdb) set follow-fork-mode child
(gdb) break main
Breakpoint 1 at 0x4005dc: file fork.c, line 8.
(gdb) run
[...]
Breakpoint 1, main () at fork.c:8
8   pid = fork();
(gdb) next
[New process 3875]
[Thread debugging using libthread_db enabled]
[Switching to Thread 0x7ffff7fd5720 (LWP 3875)]
9   if (pid == 0)
(gdb) next
11      name = "I am the child";
(gdb) next
17   return 0;
(gdb) print name
$2 = 0x400708 "I am the child"
(gdb)
```

GDB は、ここで子プロセスに切り替えました。

この設定を適切な **.gdbinit** に追加すると、永続的な設定にすることができます。

たとえば、**set follow-fork-mode ask** が **~/.gdbinit** に追加されると、 ask モードがデフォルトモードになります。

### 5.3.5. 個別スレッドのデバッグ

GDB には、個別のスレッドをデバッグし、それらを個別に操作および検査する機能があります。この

機能はデフォルトでは有効ではありません。これを実行するには、**set non-stop on** および **set target-async on** を使用します。これらは **.gdbinit** に追加できます。この機能がオンになると、GDB はスレッドデバッグを実行する準備ができます。

たとえば、以下のプログラムは 2 つのスレッドを作成します。これらの 2 つのスレッドとメインを実行する元のスレッドと合わせると、合計 3 つのスレッドになります。

**three-threads.c**

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

pthread_t thread;

void* thread3 (void* d)
{
  int count3 = 0;

  while(count3 < 1000){
    sleep(10);
    printf("Thread 3: %d\n", count3++);
  }
  return NULL;
}

void* thread2 (void* d)
{
  int count2 = 0;

  while(count2 < 1000){
    printf("Thread 2: %d\n", count2++);
  }
  return NULL;
}

int main (){

  pthread_create (&thread, NULL, thread2, NULL);
  pthread_create (&thread, NULL, thread3, NULL);

  //Thread 1
  int count1 = 0;

  while(count1 < 1000){
    printf("Thread 1: %d\n", count1++);
  }

  pthread_join(thread,NULL);
  return 0;
}
```

これを GDB で検査するためにこのプログラムをコンパイルします。

```
gcc -g three-threads.c -o three-threads  -lpthread
gdb ./three-threads
```

まず、すべてのスレッド関数 **thread1**、**thread2**、およびメインにブレークポイントを設定します。

```
(gdb) break thread3
Breakpoint 1 at 0x4006c0: file three-threads.c, line 9.
(gdb) break thread2
Breakpoint 2 at 0x40070c: file three-threads.c, line 20.
(gdb) break main
Breakpoint 3 at 0x40074a: file three-threads.c, line 30.
```

次に、プログラムを実行します。

```
(gdb) run
[...]
Breakpoint 3, main () at three-threads.c:30
30   pthread_create (&thread, NULL, thread2, NULL);
[...]
(gdb) info threads
* 1 Thread 0x7ffff7fd5720 (LWP 4620)  main () at three-threads.c:30
(gdb)
```

コマンド **info threads** がプログラムのスレッド要約と現在の状態についての詳細情報の一部を提供することに注意してください。この場合、作成されているのは 1 つのスレッドのみになります。

さらに実行を継続します。

```
(gdb) next
[New Thread 0x7ffff7fd3710 (LWP 4687)]
31   pthread_create (&thread, NULL, thread3, NULL);
(gdb)
Breakpoint 2, thread2 (d=0x0) at three-threads.c:20
20   int count2 = 0;
next
[New Thread 0x7ffff75d2710 (LWP 4688)]
34   int count1 = 0;
(gdb)
Breakpoint 1, thread3 (d=0x0) at three-threads.c:9
9   int count3 = 0;
info threads
  3 Thread 0x7ffff75d2710 (LWP 4688)  thread3 (d=0x0) at three-threads.c:9
  2 Thread 0x7ffff7fd3710 (LWP 4687)  thread2 (d=0x0) at three-
threads.c:20
* 1 Thread 0x7ffff7fd5720 (LWP 4620)  main () at three-threads.c:34
```

ここで、2 つのスレッドがさらに作成されました。スターマークの付いたスレッドは、現在フォーカスされているスレッドであることを示しています。また、新たに作成されたスレッドの **thread2()** と **thread3()** は、初期化関数でそれらに対して設定されたブレークポイントにヒットしました。

実際のスレッドデバッグを開始するには、**thread <thread number>** コマンドを使用してフォーカスを別のスレッドに切り替えます。

```
(gdb) thread 2
[Switching to thread 2 (Thread 0x7ffff7fd3710 (LWP 4687))]#0  thread2
(d=0x0)
    at three-threads.c:20
20   int count2 = 0;
(gdb) list
15   return NULL;
16 }
17
18 void* thread2 (void* d)
19 {
20   int count2 = 0;
21
22   while(count2 < 1000){
23     printf("Thread 2: %d\n", count2++);
24   }
```

**Thread 2** は、その関数 **thread2()** の **20** 行目で停止しました。

```
(gdb) next
22   while(count2 < 1000){
(gdb) print count2
$1 = 0
(gdb) next
23     printf("Thread 2: %d\n", count2++);
(gdb) next
Thread 2: 0
22   while(count2 < 1000){
(gdb) next
23     printf("Thread 2: %d\n", count2++);
(gdb) print count2
$2 = 1
(gdb) info threads
  3 Thread 0x7ffff75d2710 (LWP 4688)  thread3 (d=0x0) at three-threads.c:9
* 2 Thread 0x7ffff7fd3710 (LWP 4687)  thread2 (d=0x0) at three-
threads.c:23
  1 Thread 0x7ffff7fd5720 (LWP 4620)  main () at three-threads.c:34
(gdb)
```

上記では、**thread2** の数行がカウンターの **count2** を出力しており、**'info threads'** の出力に表示されるように **thread 2** が **23** 行目に置かれています。

次は **thread3** です。

```
(gdb) thread 3
[Switching to thread 3 (Thread 0x7ffff75d2710 (LWP 4688))]#0  thread3
(d=0x0)
    at three-threads.c:9
9    int count3 = 0;
(gdb) list
4
5 pthread_t thread;
6
7 void* thread3 (void* d)
8 {
```

```
9    int count3 = 0;
10
11    while(count3 < 1000){
12      sleep(10);
13      printf("Thread 3: %d\n", count3++);
(gdb)
```

Thread 3 は、Sleep ステートメントがあり、実行スピードが遅くなるという点で少し異なっています。これは、無視される IO スレッドの表現として考えてください。このスレッドは無視されるため、その実行は、**continue** を使用して中断せずに続行されます。

```
(gdb) continue &
(gdb) Thread 3: 0
Thread 3: 1
Thread 3: 2
Thread 3: 3
```

**continue** の末尾にある **&** に注意してください。これによって、GDB のプロンプトが戻るため、他のコマンドを実行することができます。**interrupt** を使用して thread 3 が再び関連性を持つ場合は、実行が停止される可能性があります。

```
(gdb) interrupt
[Thread 0x7ffff75d2710 (LWP 4688)] #3 stopped.
0x000000343f4a6a6d in nanosleep () at
../sysdeps/unix/syscall-template.S:82
82 T_PSEUDO (SYSCALL_SYMBOL, SYSCALL_NAME, SYSCALL_NARGS)
```

また、元のメインスレッドに戻り、これをさらに検査することも可能です。

```
(gdb) thread 1
[Switching to thread 1 (Thread 0x7ffff7fd5720 (LWP 4620))]#0  main ()
    at three-threads.c:34
34    int count1 = 0;
(gdb) next
36    while(count1 < 1000){
(gdb) next
37      printf("Thread 1: %d\n", count1++);
(gdb) next
Thread 1: 0
36    while(count1 < 1000){
(gdb) next
37      printf("Thread 1: %d\n", count1++);
(gdb) next
Thread 1: 1
36    while(count1 < 1000){
(gdb) next
37      printf("Thread 1: %d\n", count1++);
(gdb) next
Thread 1: 2
36    while(count1 < 1000){
(gdb) print count1
$3 = 3
(gdb) info threads
  3 Thread 0x7ffff75d2710 (LWP 4688)  0x000000343f4a6a6d in nanosleep ()
    at ../sysdeps/unix/syscall-template.S:82
```

```
  2 Thread 0x7ffff7fd3710 (LWP 4687)  thread2 (d=0x0) at three-
threads.c:23
* 1 Thread 0x7ffff7fd5720 (LWP 4620)  main () at three-threads.c:36
(gdb)
```

情報スレッドの出力からも分かるように、他のスレッドはそれらが置かれた場所にあるため、**thread 1** のデバッグの影響は受けません。

### 5.3.6. GDB の代替ユーザーインターフェース

GDB はデフォルトのインターフェースとしてコマンドラインを使用します。ただし、これには *machine interface* (MI) と呼ばれる API もあります。MI を使用すると、IDE 開発者は GDB への他のユーザーインターフェースを作成することができます。

これらのインターフェースのいくつかの例は以下の通りです。

**Eclipse (CDT)**

Eclipse 開発環境に統合されたグラフィカルデバッガーインターフェース。さらに詳しい情報は、『Eclipse web サイト』 にあります。

**Nemiver**

GNOME デスクトップ環境に適したグラフィカルなデバッガーインターフェース。さらに詳しい情報は、『Nemiver web サイト』にあります。

**Emacs**

emacs に統合された GDB インターフェース。さらに詳しい情報は、『Emacs web サイト』にあります。

### 5.3.7. GDB ドキュメント

GDB についての詳細は、GDB マニュアルを参照してください。

http://sources.redhat.com/gdb/current/onlinedocs/gdb.html

さらに、コマンドの **`info gdb`** および **`man gdb`** は、gdb のインストール済みバージョンに関する最新でより詳しい情報を提供しています。

## 5.4. VARIABLE TRACKING AT ASSIGNMENTS

*Variable Tracking at Assignments* (VTA) は、最適化時の変数追跡を改善するために使用される GCC に組み込まれた新規のインフラストラクチャーです。これにより、GCC は GDB、SystemTap および他のデバッグツール用の詳細で、意味のある役に立つデバッグ情報を生成できます。

GCC が最適化を有効にしてコードをコンパイルする場合、変数の名前が変更されるか、変数が移動するか、またはすべて一緒に削除されます。そのため、最適化されたコンパイルにより、デバッガーは一部の変数が **<optimized out>** されていることを報告します。 VTA を有効にすると、最適化されたコードには内部で注釈が付けられ、変数が移動されているか、または削除されるかどうかにかかわらず、それぞれの変数の値を透過的に追跡するために最適化が渡されることを確認できます。この結果、最適化された (**`gcc -O2 -g`** でビルドされた) コードの場合でもより多くのパラメーターと変数値を利用できるようになります。さらに、表示される **<optimized out>** メッセージがより少なくなります。

VTA の利点は、アプリケーションをインライン関数でデバッグする場合にさらに明らかになります。VTA を使用しないと、最適化によりインライン関数の一部の引数が完全に削除され、デバッガーがその

値を検査できなくなる可能性があります。VTA を使用すると最適化が実行され、さらに欠落した引数についての適切なデバッグ情報が生成されます。

VTA は、コードを最適化でコンパイルし、デバッグ情報を有効にする (つまり、**gcc -O -g** またはより一般的には **gcc -O2 -g**) 場合にデフォルトで有効になります。このビルド時に VTA を無効にするには、**-fno-var-tracking-assignments** を追加します。さらに、VTA インフラストラクチャーには、新規の **gcc** オプション **-fcompare-debug** が含まれます。このオプションは、デバッグ情報を使った場合とデバッグ情報を使わなかった場合の GCC でコンパイルされたコードをテストします。テストはこれら 2 つのバイナリーが同一である場合にパスします。このテストにより、実行可能コードがいずれのデバッグオプションによっても影響されず、かつデバッグコードには隠されたバグがないことを確認できます。**-fcompare-debug** はコンパイル時に多くのコストを追加することに注意してください。このオプションについての詳細は、**man gcc** を参照してください。

VTA のインフラストラクチャーおよび開発についての詳細は、以下のリンクから閲覧可能な 『A Plan to Fix Local Variable Debug Information in GCC』 を参照してください。

http://gcc.gnu.org/wiki/Var_Tracking_Assignments

このホワイトペーパーのスライド資料のバージョンは、http://people.redhat.com/aoliva/papers/vta/slides.pdf にあります。

## 5.5. PYTHON PRETTY-PRINTER

GDB コマンド **print** は、ターゲットアプリケーションのついての総合的なデバッグ情報を出力します。GDB は、可能な限り多くのデバッグデータをユーザーに提供することを目的としています。ただし、非常に複雑なプログラムの場合には、データ量は非常に把握しにくくなる可能性があります。

また、GDB は GDB **print** 出力の解読に役立つツールを提供しません。GDB は、プログラムデータの解読に役立つツールを簡単に作成する権限をユーザーに与えません。そのため、デバッグデータを読み取り、把握する方法が、とりわけ大規模で複雑なプロジェクトの場合にかなり難しくなります。

ほとんどの開発者にとって、GDB **print** 出力をカスタマイズし、(これにより意味をもたせる) 唯一の方法は、GDB を変更し、再コンパイルする方法です。ただし、これを実際に行える開発者はほとんどいません。さらにこの方法には拡張性がなく、開発者が異種のプログラムで、同じ様に複雑なデバッグデータを含む複数プログラムもデバッグする必要がある場合には特に対応が難しくなります。

これに対処するため、Red Hat Enterprise Linux 6 バージョンの GDB は Python *pretty-printer* との互換性を持つようになりました。これにより、イントロスペクション、出力、およびフォーマットロジックを **サードパーティー** の Python スクリプトに残すことにより、より意味を持つデバッグデータの取得が可能になります。

Python pretty-printer との互換性により、GDB 出力のカスタマイズを目的に適合するものにできます。これにより、GDB 出力を必要に応じて**変化**させる柔軟性とその容易性が大幅に向上するため、GDB はより幅広いプロジェクトに対するより実行可能なデバッグソリューションになります。また、プロジェクトや特定のプログラミング言語に精通した開発者こそが意味を持つ出力の種類を決定する上で最適であり、その有用性を高めることができます。

Python pretty-printer の実装により、ユーザーは仕様に従ってプログラムデータの検査、フォーマット、および出力を自動的に実行できます。これらの仕様は Python スクリプトによって実装されるルールとして作成されます。これには以下のメリットがあります。

**安全性**

プログラムデータを登録済みの一連の Python pretty-printer に渡すために、GDB 開発チームは GBD 印刷コードに *フック*を追加しました。これらのフックは、安全性に注意を払って実装されました。組み込まれた GBD 印刷コードは依然として元のままの状態で、デフォルトのフォールバック印刷ロジックと

して機能できます。そのため、専門のプリンターが使用できない場合でも、GDB はこれまでと同様の方法でデバッグデータを出力します。これにより、GDB に後方互換性を持たせ、pretty-printer を必要としないユーザーが GDB を引き続き使用することができます。

**高いカスタマイズ性**

この新規の「Python スクリプト作成」アプローチにより、ユーザーは必要に応じた量の知識を特定のプリンターに抽出することができます。そのため、プロジェクトには、ユーザー要件に特化した特定の方法でプログラムデータを解析するプリンタースクリプトのライブラリー全体を組み込むことができます。ユーザーが特定プロジェクト用にビルドできるプリンター数には制限がありません。さらに、スクリプトによってデバッグデータのスクリプトがカスタマイズできることで、ユーザーによるプリンタースクリプト – またはそれらのライブラリー全体の再利用と目的の再設定がより容易になります。

**習得が容易**

このアプローチの最もすぐれている点は、初めてでも実行しやすいことです。Python スクリプト作成は学習が比較的に容易で、オンラインで利用できる無料のドキュメントが多数あります。さらに、大半のプログラマーは Python スクリプト作成や一般的なスクリプト作成における基本から中級レベルの経験をすでに有しています。

以下は、pretty printer の小さな例です。以下の C++ プログラムを見てみましょう。

**fruit.cc**

```
enum Fruits {Orange, Apple, Banana};

class Fruit
{
  int fruit;

 public:
  Fruit (int f)
    {
      fruit = f;
    }
};

int main()
{
  Fruit myFruit(Apple);
  return 0;             // line 17
}
```

これは、コマンド **g++ -g fruit.cc -o fruit** でコンパイルされています。これから、GDB を使ってこのプログラムを検査します。

```
gdb ./fruit
[...]
(gdb) break 17
Breakpoint 1 at 0x40056d: file fruit.cc, line 17.
(gdb) run

Breakpoint 1, main () at fruit.cc:17
17   return 0;             // line 17
(gdb) print myFruit
$1 = {fruit = 1}
```

**{fruit = 1}** の出力は、これがデータ構造 'Fruit' 内の 'fruit' の内部表現であるために正確なものです。ただし、ここでは整数 1 がどの fruit を表すかを判別することが難しいため、人間による読み取りは容易ではありません。

この問題を解決するために、以下の pretty printer を作成します。

```
fruit.py

class FruitPrinter:
    def __init__(self, val):
        self.val = val

    def to_string (self):
        fruit = self.val['fruit']

        if (fruit == 0):
            name = "Orange"
        elif (fruit == 1):
            name = "Apple"
        elif (fruit == 2):
            name = "Banana"
        else:
            name = "unknown"
        return "Our fruit is " + name

def lookup_type (val):
    if str(val.type) == 'Fruit':
        return FruitPrinter(val)
    return None

gdb.pretty_printers.append (lookup_type)
```

ボトムアップの視点でこのプリンターを検査します。

**gdb.pretty_printers.append (lookup_type)** 行は、関数 **lookup_type** を、printer lookup 関数の GDB のリストに追加します。

関数 **lookup_type** は、出力するオブジェクトのタイプを検査し、適切な pretty printer を返します。オブジェクトは、パラメーター ***val*** で GDB によって渡されます。 **val.type** は pretty printer のタイプを表す属性です。

**FruitPrinter** は、実際の作業が行われる場所です。さらに具体的には、その場所はそのクラスの **to_string** 関数になります。この関数では、整数 **fruit** は、python ディクショナリ構文 **self.val['fruit']** を使って取得されます。次に、名前がその値を使って決定されます。この関数によって戻される文字列は、ユーザーに出力される文字列になります。

**fruit.py** の作成後、これは次のコマンドを使って GDB にロードされる必要があります。

```
(gdb) python execfile("fruit.py")
```

『GDB および Python Pretty-Printer』 ホワイトペーパーは、この機能についてのより詳しい情報を提供します。このホワイトペーパーには、独自の Python pretty-printer の作成方法や、これを GDB にインポートする方法についての詳細情報といくつかの例も記載されています。詳細は以下のリンクを参照してください。

http://sourceware.org/gdb/onlinedocs/gdb/Pretty-Printing.html

## 5.6. ECLIPSE による C/C++ アプリケーションのデバッグ

Eclipse C/C++ 開発ツールには GNU Debugger (GDB) との優れた統合性があります。これらの Eclipse プラグインは、GDB で利用できる最新機能を利用します。

アプリケーションのデバッグセッション開始は、コンテキストメニューの **Debug As (次をデバッグ)** → **C/C++ Application (C/C++ アプリケーション)** から、または **Run (実行)** メニューの使用のいずれでアプリケーションを起動する作業に似ています。コンテキストメニューは以下の 3 つの方法のいずれかによってアクセスできます。

- エディター内のカーソルを使って右マウスボタンをクリック
- アプリケーションバイナリーから
- 関連するバイナリーが含まれるプロジェクトから

複数のバイナリーを起動できる場合、どちらかを選択できるようにダイアログが表示されます。

セッションが開始されると、デバッグに関連する以下のビューのコレクションが含まれるデバッグパースペクティブに切り替えるためのプロンプトが表示されます。

**コントロールビュー**

コントロールビューはデバッグビューとして知られており、これには、コード選択のステップオーバーとステップインを行うためのボタンが含まれます。また、ここからスレッドプロセスを一時停止できます。

**ソースコードエディタービュー**

ソースコードエディタービューは、どのソースコード行が実行内のデバッガーの位置に対応するかを示します。デバッグビューツールバー内の **Instruction Stepping Mode (命令のステップ実行モード)** ボタンを押すことにより、ソースコード行ではなくアセンブリ命令によってアプリケーションの実行を制御することができます。

**コンソールビュー**

コンソールビューは、利用可能な入力および出力を表示します。

最後に、変数データおよび他の情報は、デバッグパースペクティブの対応するビューで確認できます。

詳細は、ヘルプコンテンツの 『C/C++ Development User Guide』 の **Concepts** → **Debug**、**Getting Started** → **Debugging Projects** および **Tasks** → **Running and Debugging Projects** セクションを参照してください。

# 第6章 プロファイル

開発者は、パフォーマンスに最も大きな影響を与えるプログラムの部分に注目するためにプログラムのプロファイルを作成します。収集されるデータのタイプには、プロセッサーの時間を最も多く消費するプログラムのセクションや、メモリが割り振られる場所などがあります。プロファイルでは、実際のプログラム実行からデータを収集します。そのため、収集されるデータの質は、プログラムが実施する実際のタスクに影響されます。プロファイル時に実施されるタスクは、実際の使用を表すものでなければなりません。これにより、プログラムの実際の使用に起因する問題への対応を開発時に確実に行うことができるようになります。

Red Hat Enterprise Linux 6 には、プロファイルデータを収集するための数多くの異なるツール (Valgrind、OProfile、 **perf**、および SystemTap) が含まれます。それぞれのツールは、以下のセクションで説明されているように特定タイプのプロファイルの実行に適しています。

## 6.1. VALGRIND

**Valgrind** は、アプリケーションの詳細なプロファイルを作成するために使用する動的な分析ツールを構築するための計測フレームワークです。**Valgrind** ツールは、通常多くのメモリー管理およびスレッド化の問題を自動検出するために使用されます。また、**Valgrind** スイートには、新規のプロファイルツールを随時作成できるツールも含まれます。

**Valgrind** は、初期化されていないメモリーの使用、メモリーの不適切な割り振り/解放、およびシステム呼び出しの不適切な引数などのエラーをチェックするためのユーザースペースバイナリーの計測を提供します。そのプロファイルツールは、標準的なユーザーがほとんどのバイナリーに対して使用できますが、他のプロファイラーと比較すると、**Valgrind** プロファイルの実行速度は大幅に遅くなります。バイナリーのプロファイルを作成するために、**Valgrind** はその実行可能ファイルを再作成し、再作成されたバイナリーを計測します。**Valgrind** のツールは、ユーザースペースプログラムにおけるメモリ関連の問題を見つける際に最も役立ちます。ただし、これはある時間に特定の問題や、カーネルスペースの計測/デバッグには適していません。

過去のリリースでは、**Valgrind** は IBM System z アーキテクチャーをサポートしていませんでした。しかし、6.1 の時点でこのサポートが追加されました。つまり、**Valgrind** は、Red Hat Enterprise Linux 6.x でサポートされるすべてのハードウェアアーキテクチャーをサポートするようになりました。

### 6.1.1. Valgrind ツール

**Valgrind** スイートは以下のツールで構成されています。

**memcheck**

このツールは、メモリーからの読み取りおよびメモリーへの書き込みのすべてをチェックし、**malloc**、**new**、**free**、および **delete** へのすべてのシステム呼び出しをインターセプトすることにより、プログラム内のメモリー管理の問題を検出します。他の手段を使ってメモリー管理の問題を検出することは困難であるため、**memcheck** は最もよく使用される **Valgrind** ツールと言えるかもしれません。このような問題は長期にわたって検出されないままになることが多く、最終的には診断しにくいクラッシュを生じさせます。

**cachegrind**

**cachegrind** は、CPU 内の I1、D1 および L2 キャッシュの詳細なシミュレーションを実行することにより、コード内のキャッシュミスの原因を正確に指摘するキャッシュプロファイラーです。これは、キャッシュミス数、メモリ参照、およびソースコードの各行になる命令を示します。また、**cachegrind** は関数別、モジュール別、およびプログラム全体の要約を提供し、個々のマシン命令のカウントを表示することもできます。

**callgrind**

**cachegrind** のように、**callgrind** はキャッシュ動作をモデリングできます。ただし、**callgrind** の主な目的は、実行済みコードについての callgraphs データを記録することにあります。

**massif**

**massif** はヒーププロファイラーです。これは、プログラムが使用するヒープメモリーの量を測定し、ヒープブロック、ヒープ管理オーバーヘッド、およびスタックサイズについての情報を提供します。ヒーププロファイラーは、ヒープメモリーの使用量を減らす方法を探す際に役立ちます。仮想メモリを使用するシステムでは、最適なヒープメモリー使用量が設定されたプログラムがメモリー不足になる可能性は少なく、必要なページングが少ない分、スピードが速くなることがあります。

**helgrind**

POSIX pthreads スレッド化プレミティブを使用するプログラムでは、 **helgrind** は同期エラーを検出します。このようなエラーには以下が含まれます。

- POSIX pthreads API の誤用
- ロックの順序付けの問題から生じる潜在的なデッドロック
- データレース (ロックが適切でない状態でのメモリーへのアクセス)

**Valgrind** により、独自のプロファイルツールを開発することもできます。これに関連して、**Valgrind** には、独自のツールを生成するためのテンプレートとして使用できるサンプルの **lackey** ツールが含まれます。

### 6.1.2. Valgrind の使用

**valgrind** パッケージとその依存関係は、**Valgrind** プロファイル実行を実施するために必要なすべてのツールをインストールします。**Valgrind** を使ってプログラムのプロファイルを作成するには、以下を使用します。

```
valgrind --tool=toolname program
```

***toolname*** の引数のリストについては、「Valgrind ツール」 を参照してください。**Valgrind** ツールのスィートに加えて、**none** も ***toolname*** の有効な引数です。この引数を使用することにより、プロファイルを実行せずにプログラムを **Valgrind** の下で実行できます。これは、**Valgrind** 自体のデバッグまたはベンチマークの際に役立ちます。

さらに、**Valgrind** に対し、その情報のすべてを特定ファイルに送信するように指示することもできます。これを実行するには、オプションの **--log-file=*filename*** を使用します。たとえば、実行可能ファイルの **hello** のメモリー使用量をチェックしたり、プロファイル情報を **output** に送信するには、以下を使用します。

```
valgrind --tool=memcheck --log-file=output hello
```

詳細は、**Valgrind** スイートのツールに関する他のドキュメントと合わせて、「Valgrind のドキュメント」を参照してください。

### 6.1.3. Eclipse 用の Valgrind プラグイン

Eclipse 用の **Valgrind** プラグインは、複数の **Valgrind** ツールを Eclipse に統合します。これにより、

Eclipse ユーザーは、各種のプロファイル機能をそれぞれのワークフローにシームレスに組み込むことができます。現在、Eclipse 用の **Valgrind** プラグインは、以下の 3 つの **Valgrind** ツールをサポートしています。

- **Memcheck**
- **Massif**
- **Cachegrind**

**Valgrind** プロファイル実行を起動するには、**Run > Profile** に移動します。これにより、**Profile As** ダイアログが開かれ、ここからプロファイル実行用のツールを選択できます。

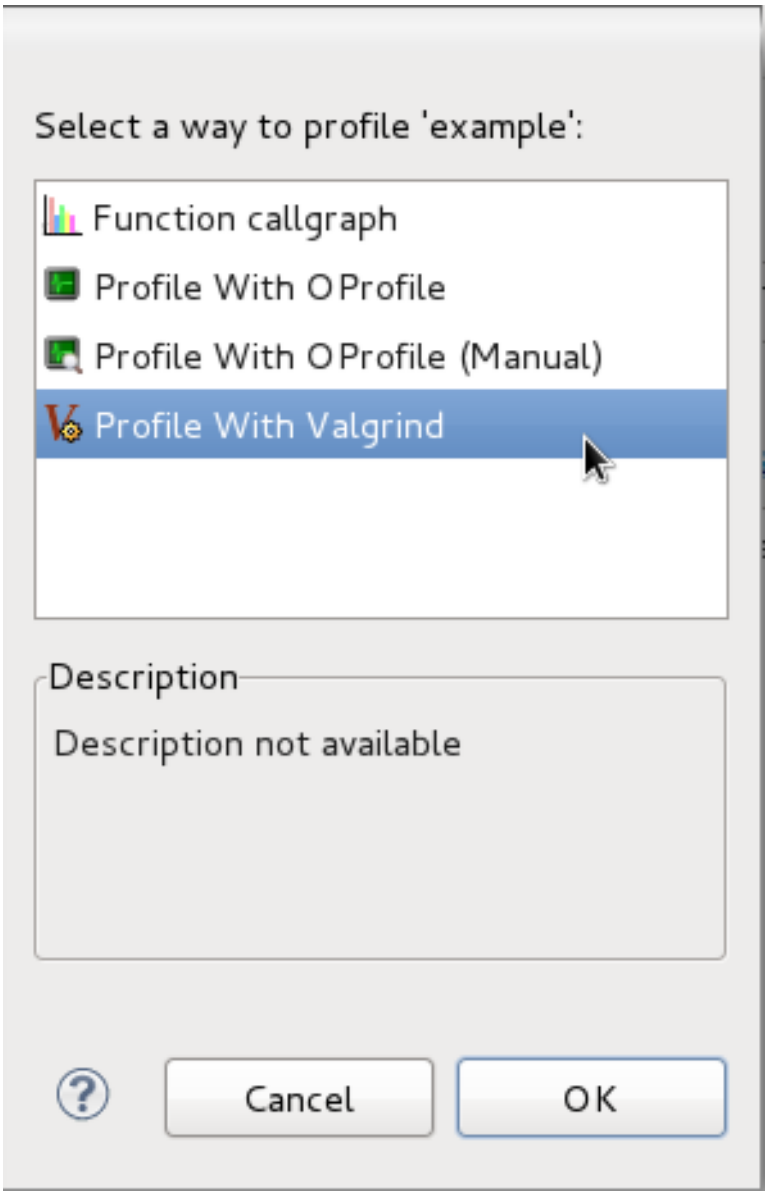



**図6.1 Profile As**

プロファイル実行用に各ツールを設定するには、**Run > Profile Configuration** に移動します。これにより、**Profile Configuration** メニューが開かれます。

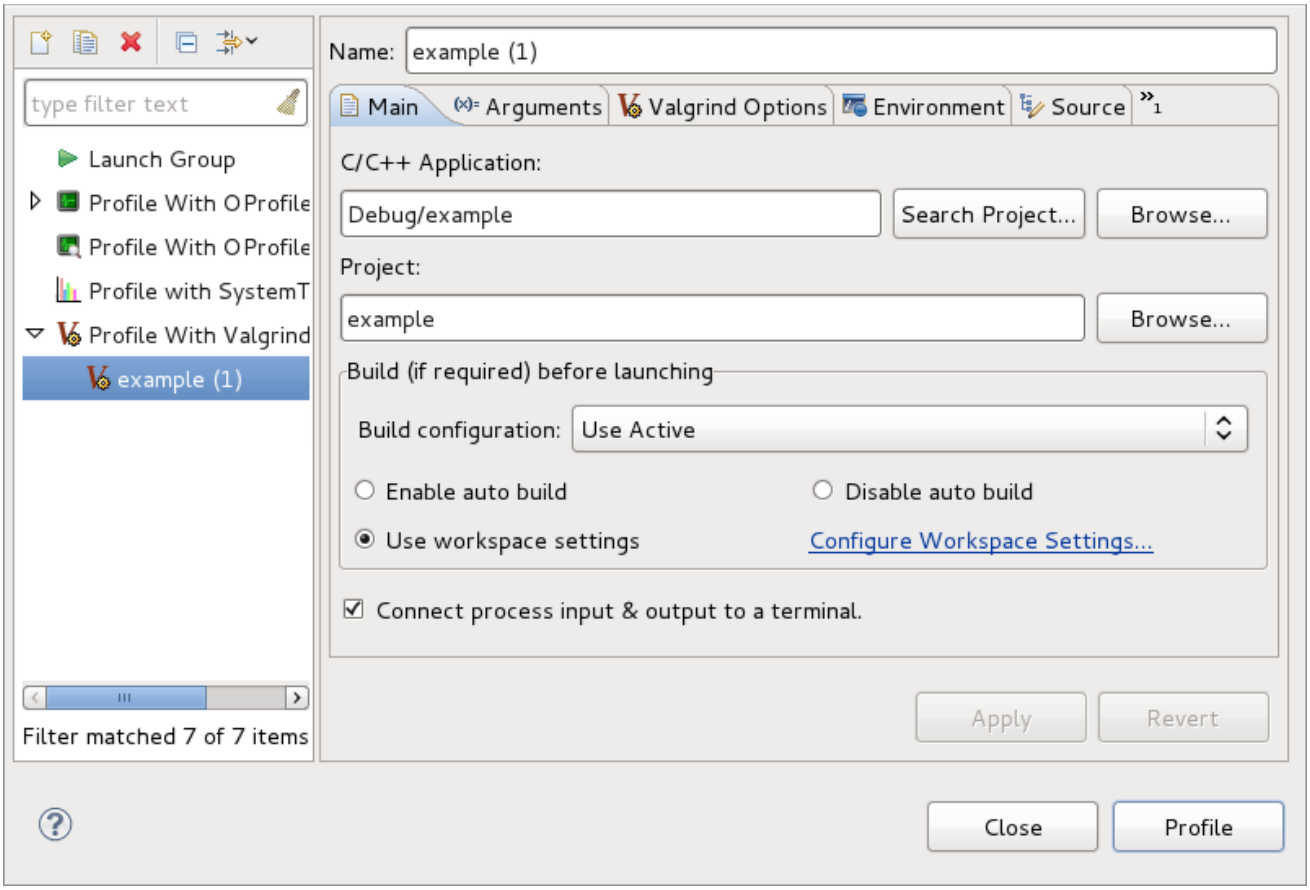



**図6.2 Profile Configuration**

Eclipse 用の **Valgrind** プラグインは、**eclipse-valgrind** パッケージで提供されます。このプラグインについての詳細は、Eclipse **Help Contents** の『Valgrind Integration User Guide』を参照してください。

### 6.1.4. Valgrind のドキュメント

**Valgrind** についての詳細は、**man valgrind** を参照してください。また Red Hat Enterprise Linux 6 は、包括的なドキュメント『Valgrind Documentation』が PDF および HTML 形式で提供されており、これらは以下にあります。

- **file:///usr/share/doc/valgrind-*version*/valgrind_manual.pdf**
- **file:///usr/share/doc/valgrind-*version*/html/index.html**

Eclipse **Help Contents** にある『Valgrind Integration User Guide』も、Eclipse 用の **Valgrind** プラグインのセットアップと使用についての詳細情報を提供しています。このガイドは、**eclipse-valgrind** パッケージで提供されています。

## 6.2. OPROFILE

OProfile はシステム全体にわたる Linux プロファイラーで、低いオーバーヘッドでの稼働が可能です。これは、raw サンプルデータ収集のためのカーネルドライバーとデーモン、さらにそのデータを有意な情報に解析するためのツールのスイートで構成されています。OProfile は通常、コードのどのセクションが CPU 時間を最も多く消費しているか、またその理由を判断するために開発者が使用します。

プロファイルの実行中に OProfile はプロセッサーのパフォーマンス監視ハードウェアを使用します。**Valgrind** はアプリケーションのバイナリーを書き直し、次にインストルメント化します。一方 OProfile は、実行中のアプリケーションをそのままプロファイルします。パフォーマンス監視ハード

ウェアをセットアップし、イベント *x* 回ごとにサンプルを採取します (たとえば、キャッシュミスやブランチ指示など)。各サンプルには、プログラムのどこでそれが発生したかという情報が含まれています。

OProfile のプロファイル方法は **Valgrind** よりも少ないリソースを消費します。しかし、OProfile は root 権限を必要とします。OProfile はコード内での「ホットスポット」を見つけ出し、その原因を探す際に便利です (たとえば、キャッシュパフォーマンスが低い、分岐予想が誤っている、など)。

OProfile を使用する際には、OProfile デーモン (**oprofiled**) を開始し、プロファイル対象のプログラムを実行し、システムプロファイルデータを収集し、このデータをより理解しやすい形式に解析することになります。OProfile はこのプロセスのすべてのステップにツールを提供します。

### 6.2.1. OProfile のツール

便利な OProfile コマンドを以下に挙げます。

**opcontrol**

このツールは OProfile デーモンを開始/終了し、プロファイルセッションの設定に使われます。

**opreport**

**opreport** コマンドは、OProfile プロファイルセッションからバイナリーイメージのサマリーまたは記号ごとのデータを出力します。

**opannotate**

**opannotate** は OProfile セッションのプロファイルデータから注釈付きソースおよび/またはアセンブリを出力します。

**oparchive**

**oparchive** コマンドは、実行可能ファイルやデバッグ、OProfile サンプルファイルの入ったディレクトリーを作成します。このディレクトリーは (**tar** で) 別のマシンに移動でき、そこでオフラインで分析ができます。

**opgprof**

**opreport** のように、**opgprof** コマンドは OProfile セッションからあるバイナリーイメージのプロファイルデータを出力します。**opgprof** の出力は、**gprof** 形式です。

OProfile コマンドの全一覧は、**man oprofile** を参照してください。各 OProfile コマンドの詳細情報については、それぞれの **man** ページを参照してください。OProfile に関するその他のドキュメントは、「OProfile のドキュメント」 を参照してください。

### 6.2.2. OProfile の使用

**oprofile** パッケージおよびその依存関係は、OProfile 実行に必要なすべてのユーティリティーをインストールします。OProfile にシステム上で実行中の全アプリケーションのプロファイルを行い、ライブラリーを使用しているアプリケーションで共有ライブラリーのサンプルをグループ化するように指示するには、以下のコマンドを実行します。

```
# opcontrol --no-vmlinux --separate=library --start
```

OProfile デーモンはシステムデータを収集せずに開始することもできます。これには、オプション **--start-daemon** を使用します。**--stop** オプションはデータ収集を中止し、**--shutdown** は OProfile デーモンを終了します。

収集されたプロファイルデータを表示するには、**opreport** か **opannotate**、**opgprof** を使用します。デフォルトでは、OProfile デーモンが収集したデータは **/var/lib/oprofile/samples/** に保存されます。

#### OProfile と Performance Counters for Linux (PCL) ツールとの競合

OProfile と Performance Counters for Linux (PCL) は同じハードウェアの Performance Monitoring Unit (PMU) を使用します。PCL もしくは NMI ウォッチドッグタイマーがハードウェア PMU を使用している場合、OProfile 開始時に以下のようなメッセージが表示されます。

```
# opcontrol --start
Using default event: CPU_CLK_UNHALTED:100000:0:1:1
Error: counter 0 not available nmi_watchdog using this resource ? Try:
opcontrol --deinit
echo 0 > /proc/sys/kernel/nmi_watchdog
```

システム上で稼働しているすべての **perf** コマンドを停止し、NMI ウォッチドッグをオフにして以下のコマンドで OProfile カーネルドライバーをリロードします。

```
# opcontrol --deinit
```

```
# echo 0 > /proc/sys/kernel/nmi_watchdog
```

### 6.2.3. Eclipse 用の OProfile プラグイン

**OProfile** スイートのツールは、強力な呼び出しプロファイル機能を提供します。プラグインとして、このらの機能は Eclipse ユーザーインターフェースにうまくポートされます。**OProfile** プラグインは以下の利点をもたらします。

#### 対象を絞ったプロファイル

**OProfile** プラグインを使うと Eclipse ユーザーは特定のバイナリーのプロファイルを行い、関連の共有ライブラリー/カーネルモジュールを含め、さらにはバイナリーを除外することができます。これにより、ソースコードの個別の行番号にいたるまで、各バイナリー、機能、記号に関する非常に対象を絞った詳細な使用量の結果が作成されます。

#### CDT に完全統合されたユーザーインターフェース

このプラグインは他のプラグインと同様に、Eclipse で豊富な **OProfile** 結果を表示します。結果のソース行の上でダブルクリックすると、Eclipse エディター上の対応する行に移動します。ここではユーザーは単一インターフェースでコードをビルド、プロファイル、編集することが可能で、Eclipse ユーザーにとってプロファイルが便利になります。さらには、プロファイルの実行が Eclipse 内で C/C++ アプリケーションと同じ方法で開始、設定されます。

#### 完全にカスタマイズ可能なプロファイルオプション

Eclipse インターフェースを使うとユーザーは **OProfile** コマンドラインユーティリティーで利用可能な全オプションを使用しているプロファイルの実行を設定できます。プラグインは、プロセッサーデバッグレジスタ (つまりカウンター) やハードウェアカウンターをサポートしていないカーネルもしくはプロセッサー用の割り込みベースのプロファイルに基づいてイベント設定をサポートします。

#### 使いやすさ

**OProfile** プラグインは通常、ほとんどのプロファイル実行に使用可能な、全オプション向けの便利なデフォルトを提供します。さらに、これらのデフォルトを使用してプロファイル実行を行う「1 クリックプロファイル」の機能もあります。ユーザーは、最初から最後までアプリケーションのプロファイル化を行うことも可能ですし、手動のコントロールダイアログでコードの特定分野を選択することもできます。

**Valgrind** プロファイルの実行を開始するには、**Run > Profile** に移動します。これで **Profile As** ダイアログが開き、プロファイル実行用のツールを選択できます。

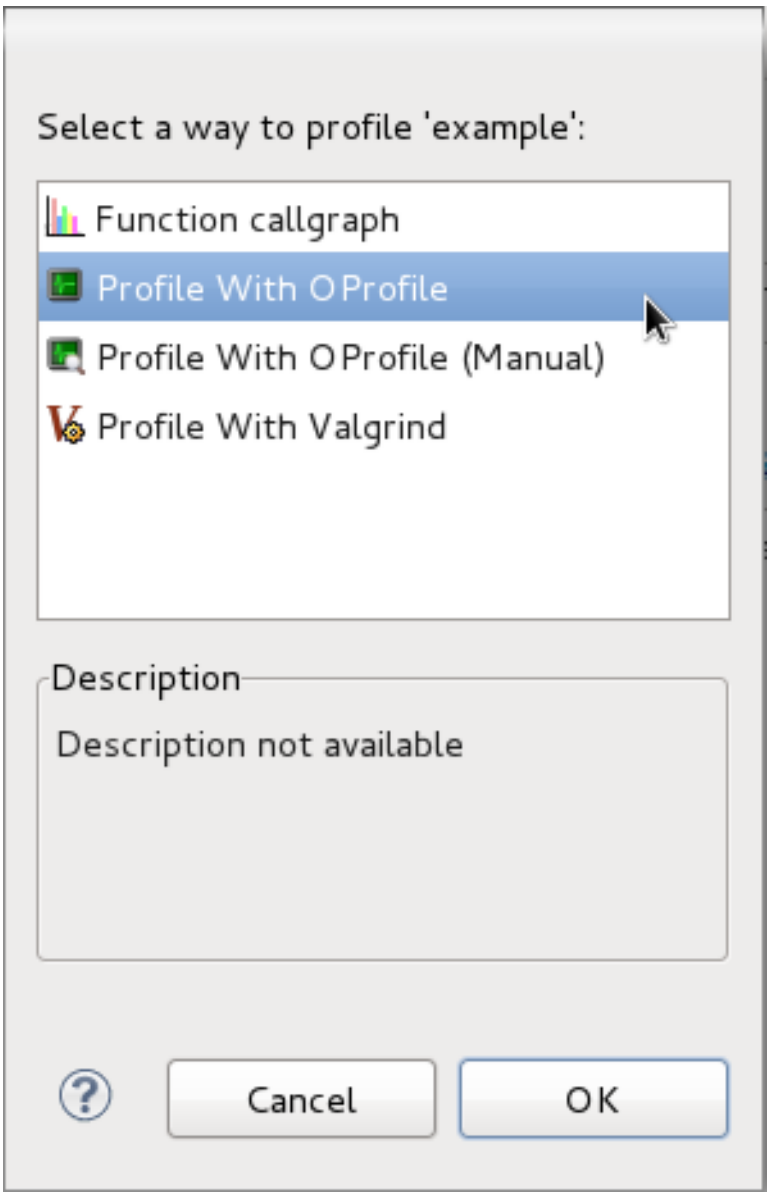



**図6.3 Profile As**

プロファイル実行用の各ツールを設定するには、**Run > Profile Configuration** に移動します。これで **Profile Configuration** メニューが開きます。

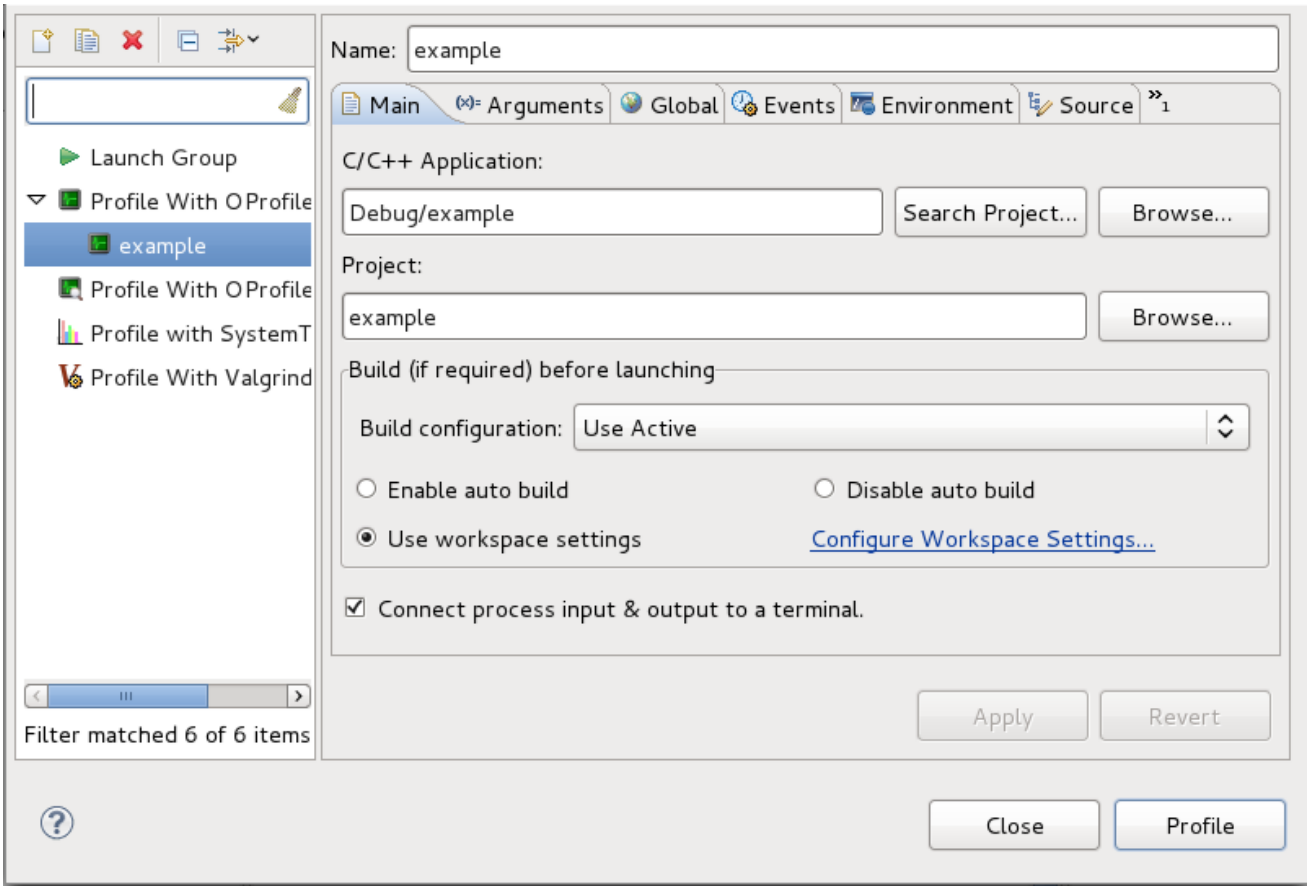



**図6.4 Profile Configuration**

Eclipse 用の OProfile プラグインは、**eclipse-oprofile** パッケージで提供されます。このプラグインについての詳細情報は、Eclipse の**ヘルプコンテンツ**にある『OProfile Integration User Guide』を参照してください (**eclipse-profile** でも提供)。

### 6.2.4. OProfile のドキュメント

OProfile に関する詳細情報は、**man oprofile** を参照してください。Red Hat Enterprise Linux 6 は **file:///usr/share/doc/oprofile-*version*/** でも OProfile の 2 種類の総合ガイドを提供しています。

**OProfile Manual**

OProfile の設定および使用方法に関する詳細な指示が記載されている総合マニュアルで、**file:///usr/share/doc/oprofile-*version*/oprofile.html** にあります。

**OProfile Internals**

OProfile の構成に関するドキュメントで、OProfile アップストリームへの投稿に興味のあるプログラマーに有用なものです。**file:///usr/share/doc/oprofile-*version*/internals.html** にあります。

Eclipse の**ヘルプコンテンツ**にある『OProfile Integration User Guide』 も Eclipse 用 OProfile プラグインの設定および使用方法に関する詳細情報を提供しています。このガイドは **eclipse-oprofile** パッケージで提供されています。

## 6.3. SYSTEMTAP

SystemTap は、Linux システム上で実行中のプロセスおよびカーネルアクティビティをプローブするための有用なインストルメンテーションプラットフォームです。プローブを実行するには、以下の手順にしたがいます。

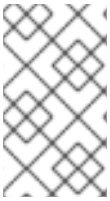
1. どのシステムイベント (たとえば、仮想ファイルシステムの読み込み、パケット送信) が特定のアクション (たとえば、印刷、解析、またはデータ操作) を開始するかを指定する *SystemTap スクリプト* を書き込みます。

2. SystemTap がスクリプトを C プログラミングに翻訳し、さらにカーネルモジュールにコンパイルします。

3. SystemTap がこのカーネルモジュールを読み込んで、実際のプローブを実行します。

SystemTap スクリプトは、通常のシステム運用に最小限の割り込みでシステム運用を監視しシステム問題を診断する際に便利なものです。インストルメント化されたコードを再コンパイルしたり再インストールすることなく、実行中のシステムテスト仮説をすばやくインストルメント化できます。**kernel-space** をプローブする SystemTap スクリプトをコンパイルするために、SystemTap は以下の 3 つの異なる *カーネル情報パッケージ*からの情報を使用します。

- **kernel-*variant*-devel-*version***
- **kernel-*variant*-debuginfo-*version***
- **kernel-debuginfo-common-*arch*-*version***



注記

Red Hat Enterprise Linux 6 のカーネル情報パッケージの名前は **kernel-debuginfo-common-*arch*-*version*** となっています。Red Hat Enterprise Linux 5 では **kernel-debuginfo-common-*version*** でした。

これらのカーネル情報パッケージは、プローブ対象のカーネルと一致する必要があります。さらに、複数のカーネル用に SystemTap スクリプトをコンパイルするには、各カーネルのカーネル情報パッケージがインストールされている必要もあります。

Red Hat Enterprise Linux 6.1 からは、**--remote** オプションという重要な新機能が追加されました。これにより、ユーザーは SystemTap モジュールをローカルでビルドし、SSH 経由でリモートでの実行が可能になります。これを使用する際の構文は、**--remote [USER@]HOSTNAME** となります。実行ターゲットを特定の SSH ホストに設定し、オプションで異なるユーザー名を使用します。複数の実行ターゲットを対象とするために、このオプションは繰り返すことができます。パス 1-4 はスクリプトを通常通りビルドするようにローカルで完了し、パス 5 がモジュールをターゲットにコピーして実行します。

以下のセクションでは、Red Hat Enterprise Linux 6 リリースで利用可能な SystemTap の他の新機能を説明します。

### 6.3.1. SystemTap コンパイルサーバー

Red Hat Enterprise Linux 6 の SystemTap は、*コンパイルサーバーとクライアント*の導入をサポートします。この設定では、ネットワークの **すべての** クライアントシステムのカーネル情報パッケージが 1 つ (もしくは 2、3) のコンパイルサーバーホストにインストールされます。クライアントシステムが SystemTap スクリプトからカーネルモジュールをコンパイルしようとすると、集約されたコンパイルサーバーホストから必要となるカーネル情報にリモートでアクセスします。

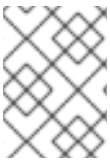
正常に設定/保守されている SystemTap コンパイルサーバーホストは、以下の利点をもたらします。

- システム管理者は、パッケージをユーザーに利用可能とする前にカーネル情報パッケージの整合性を検証できます。
- コンパイルサーバーの ID が *Secure Socket Layer* (SSL) を使って認証できます。SSL は、送信中の盗聴や改ざんを防ぐ暗号化されたネットワーク接続を提供します。
- 個別のユーザーが独自のサーバーを運用し、それらを信頼できるサーバーとして権限を与えることができます。
- システム管理者が、ネットワーク上の 1 つ以上のサーバーを全ユーザーの使用において信頼できるサーバーとして権限を付与できます。
- 明示的に権限が付与されていないサーバーは無視され、サーバーの偽装や同様の攻撃を防ぎます。

### 6.3.2. 特権のないユーザーに対する SystemTap のサポート

セキュリティのために、エンタープライズ環境でユーザーが自身のマシンへの特権アクセス (つまり、root や **sudo**) を与えられることは滅多にありません。さらに、SystemTap の完全な機能があるとシステムを完全にコントロールできるため、これも特権のあるユーザーに制限されています。

Red Hat Enterprise Linux 6 の SystemTap では、SystemTap クライアントに **--unprivileged** という新しいオプションが追加されました。このオプションを使うと、特権のないユーザーが **stap** を実行できるようになります。もちろん、**stap** を実行しようとする特権のないユーザーにはいくつかの制限が課せられます。



**注記**

特権のないユーザーとは、**stapusr** グループのメンバーですが**stapdev** グループのメンバーではありません (また、root でもありません)。

特権のないユーザーが作成したカーネルモジュールを読み込む前に、SystemTap が標準のデジタル (暗号法) 署名技術を使ってモジュールの整合性を検証します。**--unprivileged** オプションを使用する際は毎回、サーバーは特権のないユーザーに対して課せられら制限についてスクリプトをチェックします。チェックが成功した場合は、サーバーはスクリプトをコンパイルして自己生成した証明書を使って作成されたモジュールに署名します。クライアントがモジュールを読み込もうとする際には、root が保守/権限を付与している信頼できる署名証明書のデータベースに対してモジュールの署名をチェックすることで、**staprun** が最初にモジュールの署名を検証します。

署名されたカーネルモジュールが正常に検証されると、**staprun** は以下の点が保証されます。

- モジュールが信頼できる systemtap サーバー実装を使用して作成された。
- モジュールが **--unprivileged** オプションを使用してコンパイルされた。
- モジュールが、特権のないユーザーによる使用で必要とされる制限を満たしている。
- モジュールは作成以降、改ざんされていない。

### 6.3.3. SSL および認証管理

Red Hat Enterprise Linux 6 の SystemTap は、証明書および公開/プライベートキーのペアで認証およびセキュリティーを実装します。コンパイルサーバーの認証情報 (つまり証明書) を信頼できるサーバーのデータベースに追加するのはシステム管理者の責任です。SystemTap はこのデータベースを

使って、クライアントがアクセスしようとするコンパイルサーバーの ID を確認します。同様に、SystemTap はこの方法で、**--unprivileged** オプションを使ってコンパイルサーバーが作成したカーネルモジュールの確認も行います。

#### 6.3.3.1. コンパイルサーバー接続の承認

コンパイルサーバーはサーバーホスト上での初回開始時に、自動的に証明書を生成します。この証明書は、SSL 認証およびモジュール署名中にコンパイルサーバーの ID を検証します。

クライアントがコンパイルサーバーにアクセスするには (同一サーバーホスト上でもクライアントマシンからでも)、システム管理者はコンパイルサーバーの証明書を信頼できるサーバーのデータベースに追加する必要があります。コンパイルサーバーの使用を意図する各クライアントホストは、そのようなデータベースを維持しています。これにより個別のユーザーは、信頼できるサーバーのデータベースをカスタマイズできます。これには、ユーザー自身の使用のみに承認されているコンパイルサーバーの一覧を含めることができます。

#### 6.3.3.2. コンパイルサーバーのモジュール署名の承認 (特権のないユーザー)

特権のないユーザーができるのは、署名済み、承認済みの SystemTap カーネルモジュールの読み込みのみです。モジュールがそのように認識されるには、証明書が信頼できる署名者のデータベースに表示されるコンパイルサーバーが作成したものでなくてはなりません。このデータベースは、モジュールの読み込み先となる各ホストで維持される必要があります。

#### 6.3.3.3. 自動承認

**stap-server** initscript を使用して開始したサーバーは、同一ホスト上のすべてのクライアントからの接続を受け付けるよう自動的に承認されます。

他の方法で開始されたサーバーは、サーバーを開始したユーザーが稼働する同一ホスト上のクライアントからの接続を受け付けるよう自動的に承認されます。これは便利さを考慮して実装されています。クライアントとサーバーが同一ホスト上で稼働していれば、ユーザーは自分で開始したサーバーに接続するよう自動的に承認されます。

root がコンパイルサーバーを開始する時は常に、同一ホスト上で稼働している **すべての** クライアントがサーバーを承認済みと認識します。しかし、Red Hat はこれを推奨しません。

同様に、**stap-server** で開始したコンパイルサーバーは、稼働するホスト上で信頼できる署名者として自動的に承認されます。コンパイルサーバーが他の方法で開始された場合は、このように自動的には承認されません。

### 6.3.4. SystemTap のドキュメント

SystemTap についての詳細情報は (Red Hat が提供する) 以下のブックを参照してください。

- 『SystemTap Beginner's Guide』
- 『SystemTap Tapset Reference』
- 『SystemTap Language Reference』 (IBM 提供)

『SystemTap Beginner's Guide』 および 『SystemTap Tapset Reference』 は、**systemtap** パッケージをインストールするとローカルでも見ることができます。

- **file:///usr/share/doc/systemtap-*version*/SystemTap_Beginners_Guide/index.html**

- **file:///usr/share/doc/systemtap-*version*/SystemTap_Beginners_Guide.pdf**
- **file:///usr/share/doc/systemtap-*version*/tapsets/index.html**
- **file:///usr/share/doc/systemtap-*version*/tapsets.pdf**

「SystemTap コンパイルサーバー」、「特権のないユーザーに対する SystemTap のサポート」、「SSL および認証管理」 はすべて、『SystemTap Support for Unprivileged Users and Server Client Deployment』 白書からの抜粋です。この白書には各機能の詳細情報のほか、実働環境での適用を説明するケーススタディも含まれています。

## 6.4. PERFORMANCE COUNTERS FOR LINUX (PCL) ツールおよび PERF

*Performance Counters for Linux* (PCL) は、パフォーマンスデータ収集および分析用のフレームワークを提供する新しいカーネルベースのサブシステムです。これらのイベントは、パフォーマンス監視ハードウェアおよびシステムのソフトウェア設定によって異なります。Red Hat Enterprise Linux 6 には、データ収集のためのこのカーネルサブシステムと、収集されたパフォーマンスデータを分析するためのユーザースペースツールである **perf** が含まれています。

PCL サブシステムは、削除した指示やプロセッサークロックサイクルを含むハードウェアイベントを測定するために使用できます。また、主なページフォルトやコンテキストスイッチなどのソフトウェアイベントも測定できます。たとえば、PCL カウンターは、*Instructions Per Clock* (IPC) を削除した指示のプロセスカウントとプロセッサーのクロックサイクルから計算できます。IPC 率が低いと、コードが CPU をうまく活用していないことを示します。CPU の低パフォーマンスを診断するために、他のハードウェアイベントも使用することができます。

パフォーマンスカウンターは、サンプルを記録するように設定することもできます。サンプルの相対的頻度を使用して、コードのどの領域がパフォーマンスに最も影響があるかを特定することが可能です。

### 6.4.1. Perf ツールコマンド

以下に便利な **perf** コマンドを挙げます。

**perf stat**

この **perf** コマンドは、実行された指示やクロックサイクルを含む一般的なパフォーマンスイベントの全体的な統計情報を提供します。オプションでは、デフォルトの測定イベント以外のものが選べます。

**perf record**

この **perf** はパフォーマンスデータをファイルに記録し、これは後で**perf report** を使って分析できます。

**perf report**

この **perf** コマンドはファイルからパフォーマンスデータを読み取り、分析します。

**perf list**

この **perf** コマンドは、特定のマシン上で利用可能なイベントを一覧表示します。これらのイベントは、パフォーマンス監視ハードウェアおよびシステムのソフトウェア設定によって異なります。

**perf** コマンドの完全な一覧表は、**perf help** を使って取得できます。各 **perf** コマンドの **man** ページ情報を取得するには、**perf help *command*** を使います。

### 6.4.2. Perf の使用方法

プログラム実行に関する統計情報やサンプルの収集のために基本的な PCL インフラストラクチャーを使用する方法は簡単なものです。このセクションでは、全体の統計情報およびサンプリングの簡単な例を説明します。

**make** およびその子についての統計情報を収集するには、以下のコマンドを使用します。

```
# perf stat -- make all
```

**perf** コマンドは、多くの異なるハードウェアおよびソフトウェアカウンターを収集します。そして、以下の情報を表示します。

```
Performance counter stats for 'make all':

  244011.782059  task-clock-msecs         #      0.925 CPUs
          53328  context-switches         #      0.000 M/sec
            515  CPU-migrations           #      0.000 M/sec
        1843121  page-faults              #      0.008 M/sec
   789702529782  cycles                   #   3236.330 M/sec
  1050912611378  instructions             #      1.331 IPC
   275538938708  branches                 #   1129.203 M/sec
     2888756216  branch-misses            #      1.048 %
     4343060367  cache-references         #     17.799 M/sec
      428257037  cache-misses             #      1.755 M/sec

  263.779192511  seconds time elapsed
```

**perf** ツールはサンプルを記録することもできます。たとえば、**make** コマンドおよびその子に関するデータを記録するには、以下のコマンドを実行します。

```
# perf record -- make all
```

これで収集されたサンプル数とサンプルが保存されているファイルが表示されます。

```
[ perf record: Woken up 42 times to write data ]
[ perf record: Captured and wrote 9.753 MB perf.data (~426109 samples) ]
```

Red Hat Enterprise Linux 6.4 では、**{}** グループ構文に新機能が追加され、これによりコマンドライン上でイベントグループが指定されている方法に基づいてイベントグループが作成できるようになりました。

最新の **--group** および **-g** オプションには変更はありません。記録、統計情報、トップコマンドが指定されていれば、指定イベントすべてが単一グループのメンバーになり、最初のイベントがグループリーダーとなります。

新たな **{}** グループ構文は、以下のようなグループの作成を可能にします。

```
# perf record -e '{cycles, faults}' ls
```

上記のコマンドを実行すると、*cycles* および *faults* イベントを含む単一のイベントグループが作成され、*cycles* イベントがグループリーダーになります。

すべてのグループはスレッドおよび CPU に関して作成されます。このため、CPU が 4 つあるサーバー上で 2 つのスレッド内のイベントは、8 つの異なるグループを作成することになります。

グループに標準イベント修飾子を使うことができます。これは、グループ内の全イベントにまたがるもので、各イベント修飾子設定を更新します。

```
# perf record -r '{faults:k,cache-references}:p'
```

上記のコマンドでは、**:kp** 修飾子が *faults* に使われ、**:p** 修飾子が *cache-references* イベントに使われることになります。

#### Performance Counters for Linux (PCL) ツールと OProfile の競合

OProfile と Performance Counters for Linux (PCL) は同じハードウェアの Performance Monitoring Unit (PMU) を使用します。PCL **perf** コマンドを使用する際に OProfile が実行中である場合は、OProfile 開始時に以下のようなエラーメッセージが表示されます。

```
Error: open_counter returned with 16 (Device or resource busy). /bin/dmesg
may provide additional information.

Fatal: Not all events could be opened.
```

**perf** コマンドを使用するには、まず OProfile をシャットダウンします。

```
# opcontrol --deinit
```

その後に **perf.data** を分析してサンプルの相対頻度を測定することができます。レポート出力には、コマンド、オブジェクト、サンプルの機能などが含まれます。**perf report** を使って **perf.data** の分析を出力します。たとえば、以下のコマンドは最も多くの時間を消費する実行可能ファイルのレポートを作成します。

```
# perf report --sort=comm
```

出力は以下のようになります。

```
# Samples: 1083783860000
#
# Overhead          Command
# ........  ...............
#
    48.19%         xsltproc
    44.48%        pdfxmltex
     6.01%             make
     0.95%             perl
     0.17%       kernel-doc
     0.05%          xmllint
     0.05%              cc1
     0.03%               cp
     0.01%            xmlto
     0.01%               sh
     0.01%          docproc
     0.01%               ld
     0.01%              gcc
     0.00%               rm
     0.00%              sed
```

```
      0.00%   git-diff-files
      0.00%             bash
      0.00%   git-diff-index
```

左のコラムはサンプルの相対頻度を示しています。この出力では、**make** が **xsltproc** および **pdfxmltex** で最も多くの時間を消費していることを示しています。**make** が完了する時間を短縮するには、**xsltproc** と **pdfxmltex** にフォーカスします。**xsltproc** が実行する機能を一覧表示するには、以下のコマンドを実行します。

```
# perf report -n --comm=xsltproc
```

以下が生成されます。

```
comm: xsltproc
# Samples: 472520675377
#
# Overhead  Samples                     Shared Object  Symbol
# ........ ..........  ..............................  ......
#
    45.54%215179861044  libxml2.so.2.7.6               [.]
xmlXPathCmpNodesExt
    11.63%54959620202  libxml2.so.2.7.6               [.]
xmlXPathNodeSetAdd__internal_alias
     8.60%40634845107  libxml2.so.2.7.6               [.]
xmlXPathCompOpEval
     4.63%21864091080  libxml2.so.2.7.6               [.]
xmlXPathReleaseObject
     2.73%12919672281  libxml2.so.2.7.6               [.]
xmlXPathNodeSetSort__internal_alias
     2.60%12271959697  libxml2.so.2.7.6               [.] valuePop
     2.41%11379910918  libxml2.so.2.7.6               [.]
xmlXPathIsNaN__internal_alias
     2.19%10340901937  libxml2.so.2.7.6               [.]
valuePush__internal_alias
```

## 6.5. FTRACE

**ftrace** フレームワークはユーザーにいくつかのトレース機能を提供します。これは SystemTap のインターフェースよりも非常にシンプルなインターフェースでアクセス可能です。このフレームワークは、**debugfs** ファイルシステムにある仮想ファイルのセットを使用します。これらのファイルは、特定のトレーサーを有効にします。**ftrace** 機能トレーサーは、カーネルで呼び出された各機能をリアルタイムで出力します。**ftrace** フレームワーク内の他のトレーサーは、ウェイクアップ待ち時間やタスクスイッチ、カーネルイベントなどの分析にも使用できます。

また、**ftrace** に新たなトレーサーを追加し、カーネルイベントの分析用に柔軟性のあるソリューションとすることもできます。**ftrace** フレームワークは、ユーザースペースの外で発生する待ち時間やパフォーマンスの問題のデバッグや分析に便利です。本ガイド内で説明している他のプロファイラーと違い、**ftrace** カーネルのビルドイン機能です。

### 6.5.1. ftrace の使用方法

Red Hat Enterprise Linux 6 のカーネルには、**CONFIG_FTRACE=y** オプションが設定されています。このオプションは、**ftrace** が必要とするインターフェースを提供します。**ftrace** を使用するには、以下の方法で **debugfs** ファイルシステムをマウントします。

```
mount -t debugfs nodev /sys/kernel/debug
```

**ftrace** ユーティリティーはすべて、**/sys/kernel/debug/tracing/** にあります。**/sys/kernel/debug/tracing/available_tracers** ファイルで自分のカーネルでどのトレーサーが利用可能か確認してください。

```
cat /sys/kernel/debug/tracing/available_tracers
```

```
power wakeup irqsoff function sysprof sched_switch initcall nop
```

特定のトレーサーを使用するには、そのトレーサーを **/sys/kernel/debug/tracing/current_tracer** に書き込みます。たとえば、**wakeup** は、優先順位の最も高いタスクをタスクのウェイクアップ後にスケジュールするのにかかる最大時間を追跡・記録します。これを使用するには、以下のコマンドを実行します。

```
echo wakeup > /sys/kernel/debug/tracing/current_tracer
```

トレースを開始または停止するには、**/sys/kernel/debug/tracing/tracing_on** に以下のように書き込みます。

**echo 1 > /sys/kernel/debug/tracing/tracing_on** (追跡を有効化)

**echo 0 > /sys/kernel/debug/tracing/tracing_on** (追跡を無効化)

追跡結果は以下のファイルで確認できます。

**/sys/kernel/debug/tracing/trace**

このファイルには、ヒューマンリーダブルな追跡出力が含まれています。

**/sys/kernel/debug/tracing/trace_pipe**

このファイルには **/sys/kernel/debug/tracing/trace** と同じ出力が含まれていますが、これはコマンドにパイプ処理されることになっています。**/sys/kernel/debug/tracing/trace** とは異なり、このファイルからの読み取りは出力を消費します。

### 6.5.2. ftrace のドキュメント

**ftrace** フレームワークは、以下のファルで完全に文書化されています。

- 『ftrace - Function Tracer (機能トレーサー)』: **file:///usr/share/doc/kernel-doc-*version*/Documentation/trace/ftrace.txt**
- 『function tracer guts』: **file:///usr/share/doc/kernel-doc-*version*/Documentation/trace/ftrace-design.txt**

# 第7章 RED HAT DEVELOPER TOOLSET

## 7.1. RED HAT DEVELOPER TOOLSET とは?

Red Hat Developer Toolset は、開発者向けの Red Hat Enterprise Linux プラットフォームの Red Hat のオファリングで、Red Hat Enterprise Linux の複数バージョンでインストール/使用できる開発およびパフォーマンス分析ツールの包括的セットを提供します。Red Hat Developer Toolset ツールチェーンに組み込まれた実行可能ファイルも、複数バージョンの Red Hat Enterprise Linux 上で導入/実行可能です。

Red Hat Developer Toolset は、Red Hat Enterprise Linux 6 プラットフォームにインストールされても、このプラットフォームでデフォルトで提供されるシステムツールやライブラリーに代わるものではありません。開発者ツールの類似セットは、これらのツールの代わりとなる新しいバージョンを開発者が最適に使用できるように提供されるものです。Red Hat Developer Toolset が提供するアプリケーションおよびライブラリーは、Red Hat Enterprise Linux システムのバージョンに代わるものでははく、またこれらに優先して使われるものでもありません。たとえば、デフォルトのコンパイラーやデバッガーは、ベースの Red Hat Enterprise Linuxシステムが提供するものにとどまります。

開発者は、**scl** ユーティリティーを使用することで、任意のバージョンをいつでも選択できます。この製品をインストールして実行可能ファイルを起動する詳細な方法については、Red Hat Developer Toolset 3.0 User Guide を参照してください。

## 7.2. RED HAT DEVELOPER TOOLSET が提供するもの

Red Hat Enterprise Linux 6 と比較すると、Red Hat Developer Toolset は最新バージョンの **Eclipse** 開発環境、**GNU Compiler Collection** (GCC)、**GNU Debugger** (GDB)、および 表7.1「Red Hat Developer Toolset のコンポーネント」 にリスト表示されている他の開発およびパフォーマンス分析ツールを提供します。これらの更新ツールにより開発者は、最新のコンパイラー最適化、OpenMP 4.0 との並列プログラミング、およびデバッグサポートの改善などを含む実験段階の C++11 言語機能を使用しながらアプリケーションを開発することが可能になっています。

**表7.1 Red Hat Developer Toolset のコンポーネント**

| 名前 | バージョン | 説明 |
|---|---|---|
| Eclipse | 4.4 | グラフィカルユーザーインターフェースの統合開発環境。[a] |
| GCC | 4.9.1 | C、C++、および Fortran をサポートするポータブルコンパイラースイート。 |
| binutils | 2.24 | オブジェクトファイルおよびバイナリーを検査、操作するバイナリーツールおよび他のユーティリティーのコレクション。 |
| elfutils | 0.159 | ELF ファイルを検査、操作するバイナリーツールおよび他のユーティリティーのコレクション。 |
| dwz | 0.11 | ELF 共有ライブラリーおよびサイズの ELF 実行可能ファイルに含まれる DWARF デバッグ情報を最適化するツール。 |
| GDB | 7.8 | C、C++、および Fortran で書かれたプログラム用のコマンドラインデバッガー。 |

| 名前 | バージョン | 説明 |
| --- | --- | --- |
| ltrace | 0.7.91 | プログラムが作成するダイナミックなライブラリーへの呼び出しを表示するデバッグツール。プログラムが実行するシステム呼び出しを監視することもできます。 |
| strace | 4.8 | プログラムが使用するシステムコールおよび受け取るシグナルを監視するデバッグツール。 |
| memstomp | 0.1.5 | さまざまな標準では許可されていないメモリー領域の重複をともなうライブラリー関数へのコールを特定するデバッグツール。 |
| SystemTap | 2.5 | インストルメント化、再コンパイル、インストール、および再起動する必要なしにシステム全体のアクティビティーを監視する追跡およびプローブツール。 |
| Valgrind | 3.9.0 | メモリーエラーの検出、メモリー管理問題の特定、さらにシステムコールでの不適切な引数使用の報告を行うためのインストルメント化フレームワークおよびアプリケーションをプロファイル化する多くのツール。 |
| OProfile | 0.9.9 | プロセッサー上にあるパフォーマンス監視ハードウェアを使用して、システム上のカーネルと実行可能ファイルに関する情報を取得するシステム全体のプロファイラー。 |
| Dyninst | 8.2 | 実行中にインストルメント化を行い、ユーザー領域の実行可能ファイルとの作業を行うためのライブラリー。 |

[a] Red Hat JBoss Middleware 用にアプリケーションを開発する場合、もしくは OpenShift Tools 用にサポートが必要な場合は、Red Hat JBoss Developer Studio の使用が推奨されます。

## 7.3. プラットフォームの互換性

Red Hat Developer Toolset 3.0 は、 64-bit Intel および AMD アーキテクチャーの Red Hat Enterprise Linux 6 と 7 で利用可能です。図7.1「Red Hat Developer Toolset 3.0 互換性に関する表」 では、バイナリーが当システムの各種システムで実行する場合に Red Hat Developer Toolset を使用して構築したバイナリーが Red Hat Enterprise Linux の特定のバージョンでサポートされているかなどについて説明しています。

ABI の互換性に関する情報は『Red Hat Developer Toolset 3.0 User Guide 』の ABI Compatibility のセクションを参照してください。

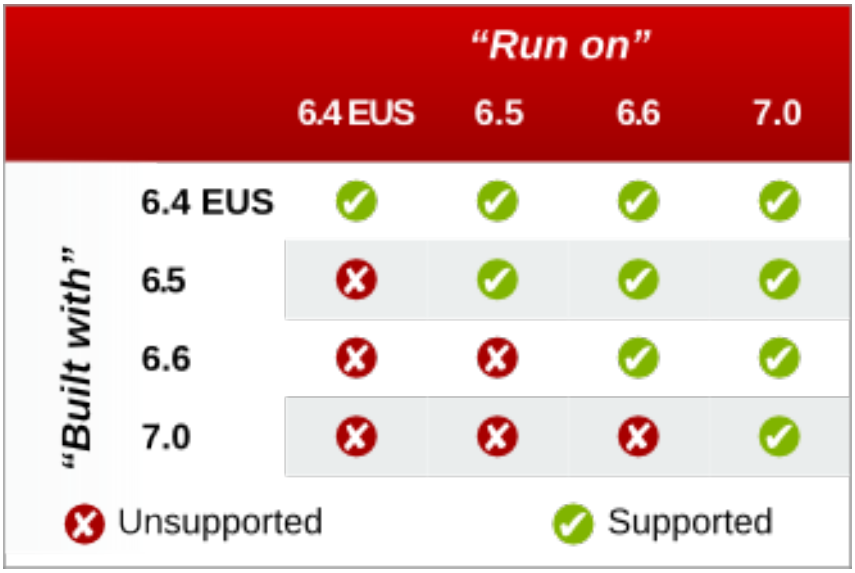



**図7.1 Red Hat Developer Toolset 3.0 互換性に関する表**

## 7.4. その他のリソース

Red Hat Developer Toolset に関する詳細情報は、以下に記載のリソースを参照してください。

- Red ">Red Hat Developer Toolset 3.0 Release Notes – Red Hat Developer Toolset 3.0 の『Release Notes』は、リリース時に入手可能な重要情報を提供しています。システム要件について、もしくは製品の既知の問題については、こちらを参照してください。
- Red Hat Developer Toolset 3.0 User Guide – Red Hat Developer Toolset 3.0 の『User Guide』は、製品の概要を提供し、Red Hat Developer Toolset のバージョンの始め方および使用方法を説明しています。Red Hat Developer Toolset をご使用のシステムで取得、インストール、使用する方法について、または本製品の詳細な変更点の一覧については、このガイドを参照してください。

# 第8章 RED HAT SOFTWARE COLLECTIONS

## 8.1. RED HAT SOFTWARE COLLECTIONS とは

アプリケーションによっては、最新機能を使うためにソフトウェアコンポーネントの最新バージョンが必要になることがよくあります。Red Hat Software Collections は Red Hat のオファリングで、動的なプログラム言語のセットやデータベースサーバーを提供します。また、Red Hat Enterprise Linux のベースシステムに含まれているものと同等の関連パッケージバージョンよりも最新のもの、またはこのシステムで初めて利用可能となるパッケージを提供します。

Red Hat Software Collections で配布される動的言語やデータベースサーバー、および他のツールは、Red Hat Enterprise Linux 6 で提供されるデフォルトのシステムツールを置き換えるもの**ではなく**、またこれらのツールに優先して使用されるものでもありません。たとえば、デフォルトバージョンの **Perl** および **PostgreSQL** は Red Hat Enterprise Linux のベースシステムが提供するもので変わりありません。ユーザーは **scl** ユーティリティーを使うことで、実行するツールのバージョンをいつでも選択することができます。

**Node.js** は明らかな例外ですが、他のすべての Red Hat Software Collections コンポーネントは、Red Hat Enterprise Linux サブスクリプションレベルアグリーメントで完全にサポートされ、機能的に完成しており、実稼動環境での使用を目的としています。

Red Hat Developer Toolset は、Red Hat Software Collections の一部ですが、別個の Software Collection となっています。Red Hat Developer Toolset に関する詳しい情報は、Red Hat Developer Toolset Release Notes および Red Hat Developer Toolset User Guide を参照してください。

## 8.2. RED HAT SOFTWARE COLLECTIONS が提供するもの

Red Hat Software Collections 1.2 は、表8.1「6 コンポーネント」に記載の最新バージョンのツールが含まれています。

**表8.1 6 コンポーネント**

| コンポーネント | Software Collection | 説明 |
|---|---|---|
| **Red Hat Developer Toolset 3.0** | devtoolset-3 | Red Hat Developer Toolset は、Red Hat_Enterprise Linux プラットフォームを使用する開発者向けに設計されています。現行バージョンの **GNU Compiler Collection**、**GNU Debugger**、**Eclipse** の開発プラットフォーム、その他の開発、デバッグ、パフォーマンス監視ツールなどが提供されます。コンポーネントの完全な一覧は、『Red Hat Developer Toolset User Guide』の Red Hat Developer Toolset Components の表を参照してください。 |
| **Perl 5.16.3** | perl516 | 最新の安定版の Perl には、多くの追加ユーティリティーやスクリプト、および **MySQL と PostgreSQL 用のデータベースコネクター** があります。このバージョンでは多くの新機能および機能強化が提供されており、新たなデバッグオプション、Unicode サポートの改善、すぐれたパフォーマンスなどが含まれます。また、**httpd24** Software Collection パッケージでのみサポートされる **perl-DateTime** と **mod_perl** が追加されています。 |

| コンポーネント | Software Collection | 説明 |
|---|---|---|
| **PHP 5.4.16** | php54 | **PEAR 1.9.4** や多くの追加拡張機能が含まれる PHP リリース。PHP 5.4 では、**言語やインターフェースが数多く向上されています**。**APC**、**memcache**、**Zend OPcache** の拡張機能も含まれます。 |
| **PHP 5.5.6** | php55 | 強化された例外処理、ジェネレーター、**Zend OPcache** などを含む **言語機能が向上された** PHP のリリース。**memcache** と **mongodb** の拡張機能も含まれています。 |
| **Python 2.7.5** | python27 | 多くのユーティリティーが追加された Python 2.7 のリリース。このバージョンの Python では、さまざまな新機能と機能強化が提供され、新たな順序付き辞書型、より速い I/O 操作、Python 3 との前方互換性の改善などが含まれます。python27 Software Collections には **Python 2.7.5 インタープリター**、ウェブアプリケーションのプログラミングと **mod_wsgi** (httpd24 Software Collection でのみ対応) に便利な拡張ライブラリー、MySQL および PostgreSQL データベースコネクター、および **numpy** と **scipy** が含まれています。 |
| **Python 3.3.2** | python33 | 多くのユーティリティーが追加された Python 3 のリリース。この Software Collection は Red Hat_Enterprise Linux の開発者に Python 3 へのアクセスを提供し、このバージョンの新機能や各種の利点を活用できるようにします。python33 Software Collections には **Python 3.3.2 インタープリター**、ウェブアプリケーションのプログラミングと **mod_wsgi** (httpd24 Software Collection でのみ対応) に便利な拡張ライブラリー、PostgreSQL データベースコネクター、および **numpy** と **scipy** が含まれています。 |
| **Ruby 1.9.3**[a] | ruby193 | Ruby 1.9.3 および**大型の Ruby gems コレクション**のある **Ruby on Rails 3.2.8** のリリースです。この Software Collection は Red Hat_Enterprise Linux の開発者に Ruby 1.9 へのアクセスを提供します。Ruby 1.9 は、改善された Unicode サポート、スレッドの機能強化、迅速な読み込み時間、httpd24 Software Collection パッケージでのみサポートされる **mod_passenger** などを含む多くの新機能および機能強化を提供します。 |
| **Ruby 2.0.0** | ruby200 | Ruby 2.0.0 のリリースです。このバージョンは大幅なパフォーマンスと信頼性の改善がなされており、Ruby 1.9.3 とのソースレベルでの後方互換性が保たれる一方で、多くの新機能とデバッグ機能の改善が追加されています。 |

| コンポーネント | Software Collection | 説明 |
| --- | --- | --- |
| **Ruby on Rails 4.0.2**[a] | ror40 | **Ruby on Rails 4.0** のリリースで、これは Ruby 言語で作成されたウェブアプリケーション開発フレームワークです。このバージョンでは多くの新機能と機能改善が提供されており、固定接続向けのライブストリーミングが追加されています。この Software Collection は、ruby200 コレクションと合わせてサポートされています。 |
| **MariaDB 5.5.37** | mariadb55 | MariaDB のリリースで、Red Hat_Enterprise Linux. のユーザーには **MySQL　の代わりとなるものです**。MySQL は MariaDB との互換性があるバイナリーで、データ変換せずに置き換えることが可能です。このバージョンは、PAM 認証プラグインを MariaDB に追加します。 |
| **MongoDB 2.4.9**[b] | mongodb24 | MongoDB のリリースで、クロスプラットフォームの**NoSQL データベースとして分類されるドキュメント指向のデータベースシステム**です。この Software Collection には、mongo-java-driver パッケージが含まれています。 |
| **MySQL 5.5.37** | mysql55 | MySQL のリリースで、パフォーマンスの改善など、多くの新機能および機能強化が提供されます。 |
| **PostgreSQL 9.2.8** | postgresql92 | PostgreSQL リリースで、**カスケードレプリケーションやネイティブ JSON サポート**、スケーラビリティーの改善、パフォーマンスの改善などの多くの新機能や機能強化が提供されます。 |
| **Node.js 0.10** [b][c] | nodejs010 | **npm 1.3.24** の Node.js リリースです。この Software Collection は、Red Hat_Enterprise Linux のユーザーにこのプログラミングプラットフォームへのアクセスを提供します。 |
| **nginx 1.6.1** | nginx16 | nginx のリリースで、高い同時実行性とパフォーマンス、低いメモリー使用量にフォーカスしたウェブおよびプロキシサーバーです。このバージョンでは、各種の **SSL 機能改善**、**SPDY 3.1**、**条件リクエストのキャッシュ再検証**のサポート、および**認証リクエストモジュール**などの多くの新機能が導入されます。 |
| **Apache httpd 2.4.6** | httpd24 | Apache HTTP サーバー (httpd) のリリースで、高パフォーマンスの **イベントベースの処理モデル、強化 SSL モジュールおよび FastCGI サポート** が含まれます。**mod_auth_kerb** モジュールも含まれています。 |
| **Thermostat 1.0.4** | thermostat1 | Thermostat のリリースで、**OpenJDK HotSpot JVM** 用の監視およびインストルメンテーションツールです。**複数の JVM インスタンス**の監視をサポートします。この Software Collection は、mongodb24 コンポーネントに依存します。 |

| コンポーネント | Software Collection | 説明 |
|---|---|---|
| **Git 1.9.4** | git19 | Git のリリースで、ピアツーピアのアーキテクチャーによる**分散リビジョン管理**システムです。クライアントサーバーモデルの集中型バージョン管理システムとは異なり、Git では Git リポジトリーの作業コピーは完全なリビジョン履歴を持つ完全なコピーになります。 |
| **DevAssistant 0.9.1** | devassist09 | DevAssistant のリリースで、各プログラム言語で開発者が**基本的なプロジェクトを作成し、設定する**際に役立つよう設計されています。また依存関係のインストール、開発環境のセットアップ、ソース管理との作業にも役立ちます。DevAssistant は、C、C++、Java、および Python の各プログラミング言語をサポートしますが、モジュラーアーキテクチャーであることから、他の言語、フレームワーク、ツールとの作業もサポートします。 |
| **Maven 3.0.5** | maven30 | Maven のリリースで、主に Java プロジェクトに使用される**ソフトウェアプロジェクト管理および包括ツール**です。プロジェクトオブジェクトモデル (POM) の概念をベースに、Maven はプロジェクトのビルド、レポート、ドキュメンテーションを集中化された情報から管理できます。 |

[a] この Software Collection の一部では、JavaScript エンジンが必要になります。Red Hat_Enterprise Linux に含まれている v8314 Software Collection は V8 JavaScript エンジンを提供し、これは Software Collection の依存関係としてのみサポートされています。

[b] この Software Collection は、**v8314** も必要になります。Red Hat_Enterprise Linux に含まれているv8314 Software Collection は V8 JavaScript エンジンを提供し、これは Software Collection の依存関係としてのみサポートされています。

[c] Red Hat_Enterprise Linux では、**Node.js** はテクノロジープレビューとして含まれています。Red Hat テクノロジープレビューについての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

## 8.3. サポート対象のプラットフォーム

Red Hat Software Collections 1.2 は、AMD64 および Intel 64 アーキテクチャーの Red Hat Enterprise Linux 6 リリースすべてで利用可能です。

## 8.4. RED HAT SOFTWARE COLLECTIONS の使い方

特定の Software Collection から実行可能ファイルを実行するには、シェルプロンプトで以下のコマンドを入力します。

```
scl enable software_collection... 'command...'
```

*software_collection* を空白で区切った Software Collections の一覧で、*command* を実行するコマンドで置き換えます。例えば、perl516 Software Collection から **hello.pl** というファイルに保存されている Perl プログラムを Perl インタープリターで実行するには、以下のように入力します。

```
~]$ scl enable perl516 'perl hello.pl'
Hello, World!
```

Red Hat Enterprise Linux と同等のものに優先して、選択した Software Collection から実行可能ファイルで新たな shell セッションを開始するには、シェルプロンプトで以下を入力します。

```
scl enable software_collection... bash
```

*software_collection* を空白で区切った Software Collections の一覧で、*command* を実行するコマンドで置き換えます。例えば、デフォルトとして新たな shell セッションを python27 および postgresql92 Software Collections で開始するには、以下のように入力します。

```
~]$ scl enable python27 postgresql92 bash
```

現行セッションで有効となっている Software Collections の一覧は、**$X_SCLS** 環境変数に保存されます。例を示します。

```
~]$ echo $X_SCLS
python27 postgresql92
```

**scl** ユーティリティーを使うことで、どのコマンドも実行できます。これは、Red Hat Enterprise Linux の同等のものに優先して、選択された Software Collection からの実行ファイルで実行されるようにします。システムにこのコマンドラインユーティリティーをインストールして使用する方法については、「Software Collections および scl-utils」 を参照してください。

Red Hat Software Collections で配布される Software Collections の完全な一覧については、表8.1「6 コンポーネント」 を参照してください。これらの Software Collections の詳細な使用方法については、『Red Hat Software Collections 1.1 Release Notes』 を参照してください。

## 8.5. RED HAT SOFTWARE COLLECTIONS を使用するアプリケーションのデプロイ方法

稼働中の Red Hat Software Collections からのコンポーネントに依存するアプリケーションを導入するには、一般的に以下のいずれかの方法を使うことができます。

- すべての必要な Software Collections とパッケージを手動でインストールし、その後にアプリケーションを導入する。
- アプリケーション用に新たな Software Collections を作成し、必要なすべての Software Collections およびその他のパッケージを依存関係として指定する。

個別の Red Hat Software Collections コンポーネントを手動でインストールする詳細な情報は、Red Hat Software Collections 1.2 Release Notes を参照してください。カスタマイズされた Software Collection を作成する詳細な説明は、『Red Hat Developer Toolset Software Collections Guide』 を参照してください。

## 8.6. その他のリソース

- Red Hat Software Collections 1.2 Release Notes – Red Hat Software Collections 1.2 の『Release Notes』 は、リリース時に入手可能な重要情報を提供しています。システム要件について知りたい場合、もしくは Red Hat Software Collections の既知の問題について知りたい場合は、こちらを参照してください。

- Red Hat Software Collections 1.2 Packaging Guide — Red Hat Software Collections 1.2 の『Packaging Guide』は、カスタマイズされた Software Collections のビルド方法について説明しています。

# 第9章 ドキュメントツール

Red Hat Enterprise Linux 6 は、ソースコードからドキュメントを生成し、スタンドアローンのドキュメントを記述する **Doxygen** ツールを提供しています。

## 9.1. DOXYGEN

Doxygen は、オンラインの HTML とオフラインの Latex の両方で参照資料を作成するドキュメントツールです。これは、文書化されたソースファイルのセットから作成を行い、これによりドキュメントの一貫性を保ちソースコードとの正確性を保つことが容易になります。

### 9.1.1. Doxygen 対応の出力および言語

Doxygen は以下の主力をサポートします。

- RTF (MS Word)
- PostScript
- ハイパーリンク付き PDF
- 圧縮 HTML
- Unix man ページ

Doxygen は以下のプログラム言語をサポートします。

- C
- C++
- C#
- Objective -C
- IDL
- Java
- VHDL
- PHP
- Python
- Fortran
- D

### 9.1.2. 使用開始

Doxygen は設定ファイルを使ってセッティングを決定するので、これが正確に作成されることが非常に重要になります。各プロジェクトには、それぞれの設定ファイルが必要になります。最も簡単に設定ファイルを作成する方法は、**doxygen -g *config-file*** コマンドを使用する方法です。この方法だと、編集が容易なテンプレート設定ファイルが作成されます。変数 *config-file* は、設定ファイル名です。コマンドからコミットされた場合は、デフォルトで Doxyfile と呼ばれます。設定ファイル作成で

のもうひとつの便利なオプションは、ファイル名にマイナス記号 (**-**) を使うことです。これは、標準出力 (**stdin**) から設定ファイルを Doxygen に読み取らせるようにするため、スクリプティングに便利なものです。

設定ファイルは、シンプルな Makefile と同様に、多くの変数およびタグから構成されます。

```
TAGNAME = VALUE1 VALUE2...
```

ほとんどの場合はこのままにしておいて構いませんが、編集の必要がある場合には、利用可能な全タグに関する詳しい説明は、Doxygen ドキュメント Web サイト 『configuration page』 を参照してください。また、**doxywizard** と呼ばれる GUI インターフェースもあります。この編集方法が望ましい場合は、機能についてのドキュメントは Doxygen ドキュメント Web サイトの 『Doxywizard usage page』 にあります。

以下の 8 つのタグは、慣れておくと便利なものです。

**INPUT**

C もしくは C++ ソースとヘッダーファイルで主に構成されている小規模プロジェクトの場合、なにも変更する必要はありません。ただし、プロジェクトが大型でソースディレクトリーやツリーで構成されている場合は、root ディレクトリー (単数または複数) を INPUT タグに割り当てます。

**FILE_PATTERNS**

ファイルパターン (たとえば、***.cpp** や ***.h**) をこのタグに追加し、パターンのいずれかに適合するファイルのみを解析させることができます。

**RECURSIVE**

この設定を **yes** にすると、ソースツリーの再帰分析が可能になります。

**EXCLUDE および EXCLUDE_PATTERNS**

これらは、回避するファイルパターンを追加することで解析されるファイルをさらに細かくチューニングするために使用されます。たとえば、ソースツリーからすべての **test** ディレクトリーを省略するには、**EXCLUDE_PATTERNS = */test/*** を使用します。

**EXTRACT_ALL**

これを **yes** に設定すると、doxygen はプロジェクトが完全に文書化されるとどのように見えるかが分かるように、ソースファイル内のすべてが文書化されているかのように振る舞います。しかし、このモードでは、文書化されていないメンバーに関する警告は生成されません。終了時これを修正するには、**no** に設定を戻します。

**SOURCE_BROWSER および INLINE_SOURCES**

**SOURCE_BROWSER** タグを **yes** に設定すると、doxygen は相互参照を生成し、ソースファイル内で存在するドキュメントでソフトウェアの定義について分析します。これらのソースは、**INLINE_SOURCES** を **yes** に設定することでドキュメント内に含めることもできます。

### 9.1.3. Doxygen の実行

**doxygen *config-file*** を実行すると、**html**、**rtf**、**latex**、**xml**、および/または **man** ディレクトリーが doxygen が開始されたディレクトリー内に作成され、対応するファイルタイプのドキュメントが含まれます。

**HTML OUTPUT**

このドキュメントは、カスケードスタイルシート (CSS) や一部では DHTML や Javascript をサポートする HTML ブラウザーで閲覧することができます。ブラウザー (たとえば、Mozilla、Safari、Konqueror、Internet Explorer 6) を **html** の **index.html** に向けます。

**LaTeX OUTPUT**

Doxygen は、**Makefile** を **latex** ディレクトリーに書き込み、Latex ドキュメントの最初のコンパイルを容易にします。これを行うには、最新の teTeX ディストリビューションを使用します。このディレクトリーに含まれるものは、**USE_PDFLATEX** が **no** に設定されているかどうかに依存します。設定されている場合は、**latex** ディレクトリーで **make** と入力すると、**refman.dvi** が生成されます。これは、**xdvi** で閲覧するか、**refman.ps** と入力して **make ps** に変換することができます。これには **dvips** が必要であることに注意してください。

便利なコマンドは多くあります。**make ps_2on1** は 2 ページを 1 ページに印刷できます。また、ghostscript インタプリターがインストールされていれば、**make pdf** コマンドで PDF に変換することもできます。ほかには **make pdf_2on1** も有効なコマンドです。これを実行する際は、**PDF_HYPERLINKS** および **USE_PDFLATEX** タグを **yes** に設定します。こうすることで、**Makefile** には **refman.pdf** を直接ビルドするターゲットのみが含まれることになります。

**RTF OUTPUT**

このファイルは、RTF 出力を単一ファイルである **refman.rtf** と組み合わせることで Microsoft Word にインポートするように設計されています。情報の中にはフィールドを使ってエンコードされるものもありますが、これはすべてを選択し (**CTRL+A** または 編集 **->** すべて選択)、右クリックをして **toggle fields** オプションをドロップダウンメニューから選ぶと表示することができます。

**XML OUTPUT**

**xml** ディレクトリーへの出力は多くのファイルで構成され、各複合ファイルは **index.xml** に加えて doxygen が収集したものです。XSLT スクリプトである **combine.xslt** も、すべての XML ファイルを単一ファイルに組み合わせるために作成されます。また、インデックスファイル用の **index.xsd** と複合ファイル用の **compound.xsd** という 2 つのXML スキーマファイルが作成され、これらは可能性のある要素、それらの属性、それらの構成方法を説明します。

**MAN PAGE OUTPUT**

**man** ディレクトリーからのドキュメントは、**manpath** の man path に正確な man ディレクトリーがあることを確認した後で **man** プログラムを使って閲覧できます。man ページ形式の制限により、図や相互参照、式といった情報は失われることに留意してください。

### 9.1.4. ソースの文書化

ソースを文書化するには 3 つのステップがあります。

1. まず、**EXTRACT_ALL** が **no** に設定され、警告が正確に生成され、ドキュメントが適切にビルドされることを確認します。これにより、doxygen は文書化されたメンバー、ファイル、クラス、ネームスペースのドキュメントを作成できます。

2. このドキュメントを作成するには 2 つの方法があります。

   ***特別な* ドキュメントブロック**

   このコメントブロックには追加のマーキングが含まれていることで Doxygen はこれがドキュメントの一部であることが分かり、C もしくは C++ で書かれています。簡単な説明もしくは詳細な説明で構成されています。これらはどちらもオプションです。しかし、*本文* の説明はオプションではありません。これがメソッドもしくは関数の本文で見つかるすべてのコメントブロックを結びつけます。

### 注記

簡単もしくは詳細な説明は 1 つ以上が認められますが、順番が指定されないのでこれは推奨されません。

以下で、コメントブロックを詳細な説明としてマークする方法を詳述します。

- JavaDoc スタイルで 2 つのアスタリスク (*) で始まる C スタイルコメントブロック

```
/**
 * ... documentation ...
 */
```

- Qt スタイルを使用した C スタイルコメントブロック。もうひとつのアスタリスクの代わりに感嘆符 (!) で構成されています。

```
/*!
 * ... documentation ...
 */
```

- ドキュメント行の最初のアスタリスクは、なくても構いません。

- C++ では最初と最後の行は空白でもよく、スラッシュ 3 つか、スラッシュ 2 つと感嘆符を付けます。

```
///
/// ... documentation
///
```

または

```
//!
//! ... documentation ...
//!
```

- 別の方法としては、コメントブロックをより見やすくするために、アスタリスクもしくはスラッシュの行を使うことができます。

```
//////////////////////////////////////////////////
/// ... documentation ...
//////////////////////////////////////////////////
```

または

```
/**********************************************//**
 * ... documentation ...
 **********************************************/
```

通常のコメントブロックの最後にあるスラッシュ 2 つが特別なコメントブロックの開始となっていることに注意してください。

ドキュメントに簡単な説明を加えるには 3 つの方法があります。

- 簡単な説明を追加するには、コメントブロックの上に **\brief** を使います。この簡単なセクションは段落の最後で終わり、それ以降の段落は詳細な説明になります。

```
/*! \brief brief documentation.
 *         brief documentation.
 *
 *  detailed documentation.
 */
```

- **JAVADOC_AUTOBRIEF** を **yes** に設定することで、簡単な説明は最初のドットとその後の空白もしくは新たな行までしか続きません。このため、簡単な説明は単一行に制限されています。

```
/** Brief documentation. Detailed documentation continues *
from here.
 */
```

これは、上記の 3 つのスラッシュ (**///**) のコメントブロックでも使用できます。

- 3 つ目のオプションは、特別な **C++** スタイルコメントを使用して、1 行以上に及ばないようにします。

```
/// Brief documentation.
/** Detailed documentation. */
```

または

```
//! Brief documentation.

//! Detailed documentation //! starts here
```

上記の例の空白の行は、簡単な説明と詳細な説明を分けるために必要で、**JAVADOC_AUTOBRIEF** を **no** に設定する必要があります。

**Qt** スタイルを使って文書化された **C++** コードの例は、『Doxygen documentation website』 にあります。

ファイル、構造体、ユニオン、クラス、**enum** のメンバーの後にドキュメントを持ってくることも可能です。 **<** マーカーをコメントブロック **** の後に追加します。

```
int var; /*!< detailed description after the member */
```

**Qt** スタイルでは以下のようになります。

```
int var; /**< detailed description after the member */
```

または

```
int var; //!< detailed description after the member
         //!<
```

または

```
int var; ///< detailed description after the member
         ///<
```

メンバーユースの後の簡単な説明は、以下のようになります。

```
int var; //!< brief description after the member
```

または

```
int var; ///< brief description after the member
```

これらの例と HTML の作成方法は 『Doxygen documentation website』 で確認できます。

### 他の場合でのドキュメント

文書化しているコードの前にドキュメントを置くのが望まれますが、特にファイルを文書化する場合などはファイルの前にドキュメントを置くことは不可能なので、異なる場所にしか置けない場合もあります。異なる場所に置くと情報の重複が発生する可能性があるので、絶対に必要な場合以外はこれを避けることが最善策となります。

これを行うには、構造コマンドをドキュメントブロック内に持つことが重要になります。構造コマンドは JavaDoc では、バックスラッシュ (\) もしくはアットマーク (@) で始まり、その後に 1 つ以上のパラメーターが続きます。

```
/*! \class Test
    \brief A test class.

    A more detailed description of class.
 */
```

上記の例では、**\class** コマンドが使われています。これは、コメントブロックにクラス 'Test' のドキュメントが含まれていることを示しています。他の例では、

- **\struct**: C 構造体を文書化します。
- **\union**: union を文書化します。
- **\enum**: 列挙タイプを文書化します。
- **\fn**: 関数を文書化します。
- **\var**: 変数、typedef、enum 値を文書化します。
- **\def**: #define を文書化します。
- **\typedef**: document a type definition
- **\file**: ファイルを文書化します。
- **\namespace**: ネームスペースを文書化します。
- **\package**: Java パッケージを文書化します。
- **\interface**: IDL インターフェースを文書化します。

3. 次に、特別なドキュメントブロックを HTML および / Latex 出力ディレクトリーに書き込む前に、解析します。以下の手順が含まれます。

    1. 特別なコマンドを実行します。

    2. 空白およびアスタリスク (*) をすべて削除します。

    3. 空白の行を新たな段落として取り扱います。

    4. 単語が対応するドキュメントにリンク付されます。単語の前にパーセンテージ記号 (%) がある場合は、その記号を削除して単語を残します。

    5. テキストに特定のパターンが見つかった場合は、メンバーへのリンクが作成されます。この例は、Doxygen ドキュメント Web サイトの 『automatic link generation page』 にあります。

    6. ドキュメントが Latex についての場合は、HTML タグが Latex のタグに相当するものに変換されます。サポート対象の HTML タグは Doxygen ドキュメント Web サイトの 『HTML commands page』 にあります。

### 9.1.5. リソース

詳細情報は以下の Doxygen web サイトで見られます。

- 『Doxygen homepage』
- 『Doxygen introduction』
- 『Doxygen documentation』
- 『Output formats』

# 付録A 付録

## A.1. MALLOPT

**mallopt** は、プログラムによる malloc メモリアロケータの動作の変更を可能にするライブラリーコールです。

> **例A.1 アロケータヒューリスティック**
>
> アロケータには、昔からあるオブジェクトと最近できたオブジェクトを見分けるヒューリスティックがあります。前者に対しては **mmap** を割り当てようとし、後者には **sbrk** を割り当てようとします。
>
> これらのヒューリスティックを上書きするには、**M_MMAP_THRESHOLD** を設定します。

マルチスレッドのアプリケーションでは、アロケータは *arenas* にある既存のロック競合に対応して複数の arenas を作成します。これより、いくつかのマルチスレッドのアプリケーションでは、メモリ使用量が増えるという代わりに、パフォーマンスが大幅に改善する場合があります。これをコントロールしておくには、**mallopt** インターフェースを使って作成可能な arenas 数を制限します。

アロケータには、作成できる arenas の数に限りがあります。32 ビットのターゲットの場合、2 * # のコア arenas を作成します。64 ビットターゲットの場合、8 * # コア arenas を作成します。**mallopt** を使うと、開発者はこれらの制限を無効に出来ます。

> **例A.2 mallopt**
>
> 9 以上の arenas が作成されないようにするには、以下のライブラリーコールを発行します。
>
> ```
> mallopt (M_ARENA_MAX, 8);
> ```

**mallopt** の最初の引数は、以下のものが可能です。

- M_MXFAST
- M_TRIM_THRESHOLD
- M_TOP_PAD
- M_MMAP_THRESHOLD
- M_MMAP_MAX
- M_CHECK_ACTION
- M_PERTURB
- M_ARENA_TEST
- M_ARENA_MAX

上記の引数の特定の定義は、http://www.makelinux.net/man/3/M/mallopt で確認できます。

**malloc_trim**

**malloc_trim** は、アロケータに未使用のメモリをオペレーティングシステムに戻すように要求するライブラリーコールです。これは、オブジェクトが解放済みの場合、通常は自動で行われます。しかし、小さいオブジェクトを解放している場合、 **glibc** がメモリを即座にオペレーティングシステムに戻さないこともあります。この理由は、メモリをオペレーティングシステムからリリースさせて割り当てるのは高くつき、空きメモリは来たるべきメモリ割り当て要求を満たすために使用できるからです。

**malloc_stats**

**malloc_stats** は、アロケータの内部状態を **stderr** にダンプするために使用されます。**mallinfo** はこれに似ていますが、代わりに状態を構造体に置きます。

## 追加情報

**mallopt** に関する追加情報は http://www.makelinux.net/man/3/M/mallopt および http://www.gnu.org/software/libc/manual/html_node/Malloc-Tunable-Parameters.html にあります。

# 付録B 改訂履歴

**改訂 2-56.2** **Tue Jul 5 2016** **Kenzo Moriguchi**
翻訳完成

**改訂 2-56.1** **Tue Jul 5 2016** **Kenzo Moriguchi**
翻訳ファイルを XML ソースバージョン 2-56 と同期

**改訂 2-56** **Tue Jul 6 2015** **Robert Krátký**
Red Hat_Enterprise Linux 6.7 開発者ガイドのリリース

**改訂 2-55** **Wed Apr 15 2015** **Robert Krátký**
Red Hat_Enterprise Linux 6.7 ベータ版開発者ガイドのリリース

**改訂 2-54** **Tue Dec 16 2014** **Robert Krátký**
Red Hat カスタマーポータルでの並び替え順序の更新

**改訂 2-52** **Wed Nov 11 2014** **Robert Krátký**
RHSCL 1.2 および DTS 3.0 の再リリース

**改訂 2-51** **Fri Oct 10 2014** **Robert Krátký**
Red Hat_Enterprise Linux 6.6 開発者ガイドのリリース

# 索引

## シンボル

## A



## B

## F



## G

## H



## I

## L



## M

## P

## R



## S

## W