Red Hat Enterprise Linux 7
開発者ガイド

Red Hat Enterprise Linux 7 のアプリケーション開発ツールのご紹介

Red Hat Subject Matter ExpertsDave Brolley
William Cohen
Roland Grunberg
Aldy Hernandez
Karsten Hopp
Jakub Jelinek
Jeff Johnston
Benjamin Kosnik
Chris Moller
Phil Muldoon
Alex Kurtakov
Charley Wang
Development Community
Kent Sebastian

Red Hat Enterprise Linux 7 開発者ガイド

Red Hat Enterprise Linux 7 のアプリケーション開発ツールのご紹介

Dave Brolley
Red Hat Engineering ツール開発
brolley@redhat.com
プロファイリング

William Cohen
Red Hat Engineering ツール開発
wcohen@redhat.com
プロファイリング

Roland Grunberg
Red Hat Engineering ツール開発
rgrunber@redhat.com
Eclipse および Eclipse プラグイン

Aldy Hernandez
Red Hat Engineering ツール開発
aldyh@redhat.com
コンパイルおよびビルド

Karsten Hopp
Base Operating System コアサービス - BRNO
karsten@redhat.com
コンパイル

Jakub Jelinek
Red Hat Engineering ツール開発
jakub@redhat.com
プロファイリング

Jeff Johnston
Red Hat Engineering ツール開発
jjohnstn@redhat.com
Eclipse および Eclipse プラグイン

Benjamin Kosnik
Red Hat Engineering ツール開発
bkoz@redhat.com
ライブラリおよびランタイムのサポート

Chris Moller
Red Hat Engineering ツール開発
cmoller@redhat.com
デバッグ

Phil Muldoon
Red Hat Engineering ツール開発
pmuldoon@redhat.com
デバッグ

Alex Kurtakov
Red Hat Engineering ツール開発
akurtako@redhat.com
Eclipse および Eclipse プラグイン

Charley Wang
Red Hat Engineering ツール開発
cwang@redhat.com
Eclipse および Eclipse プラグイン

Kent Sebastian
kent.k.sebastian@gmail.com
プロファイリング

Red Hat Subject Matter Experts

Development Community

編集者

Jacquelynn East
Engineering コンテンツサービス
jeast@redhat.com

Don Domingo
Red Hat Engineering コンテンツサービス
ddomingo@redhat.com

法律上の通知

概要

本書では、Red Hat Enterprise Linux 7 をアプリケーション開発の理想的なエンタープライズプラットフォームとする様々な機能やユーティリティーを説明します。ここでは、エンドツーエンドの総合開発環境 (IDE) として 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も説明されています。

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イ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を特徴としています。

Eclipse は幅広い異なるツールを統一された環境に統合して豊かな開発体験を作り出します。例えば、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イスで (通常はコマンドラインで実行される) メモリプロファイリングを実行できます。この機能は、Eclipse のみというわけではありません。

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イ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に幅広いドキュメンテーションを提供します。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イ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、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。

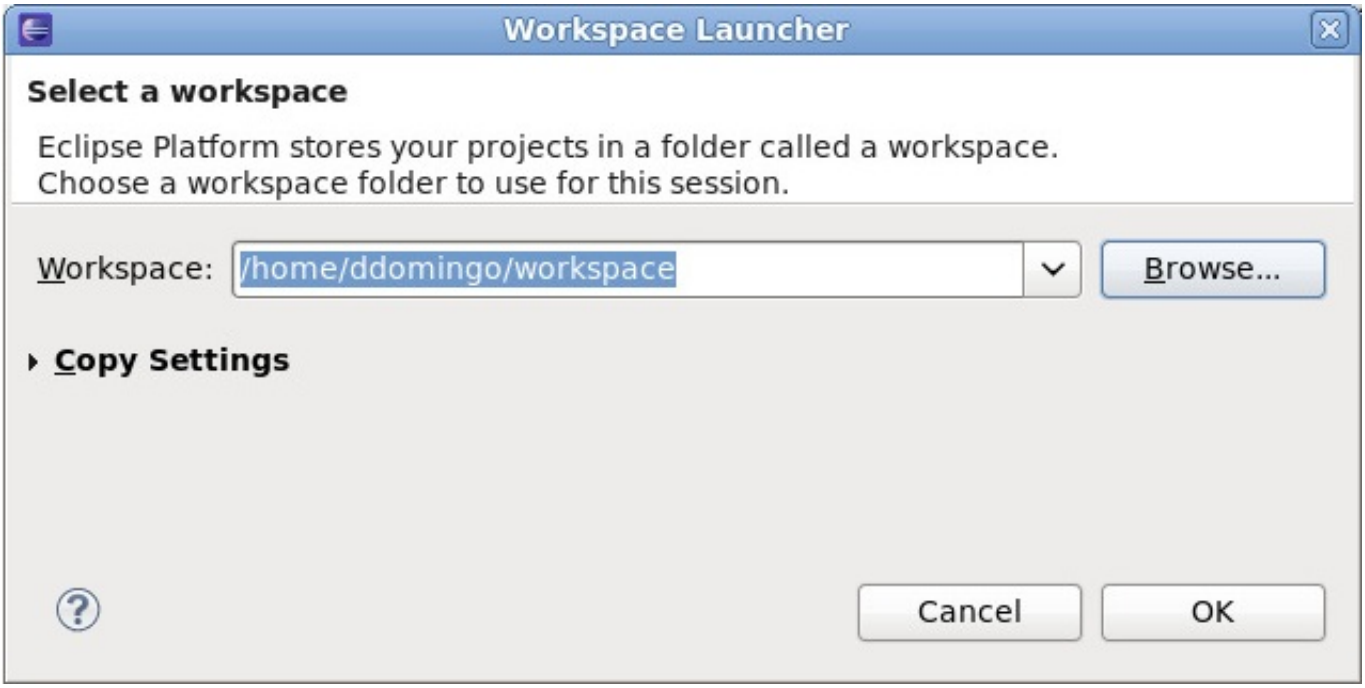


図1.1 Workspace Launcher

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イ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。

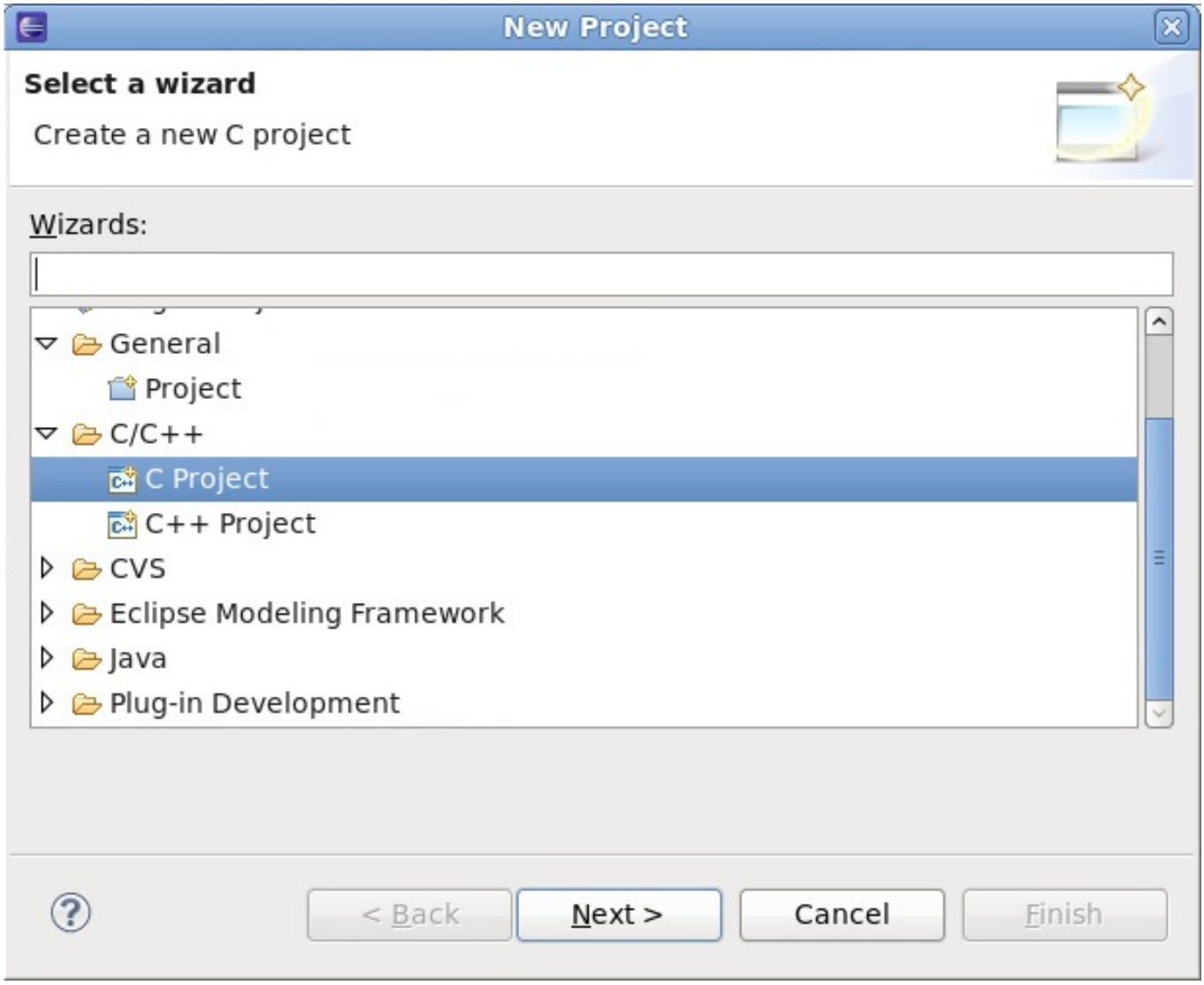


図1.2 New Project ウィザード

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。

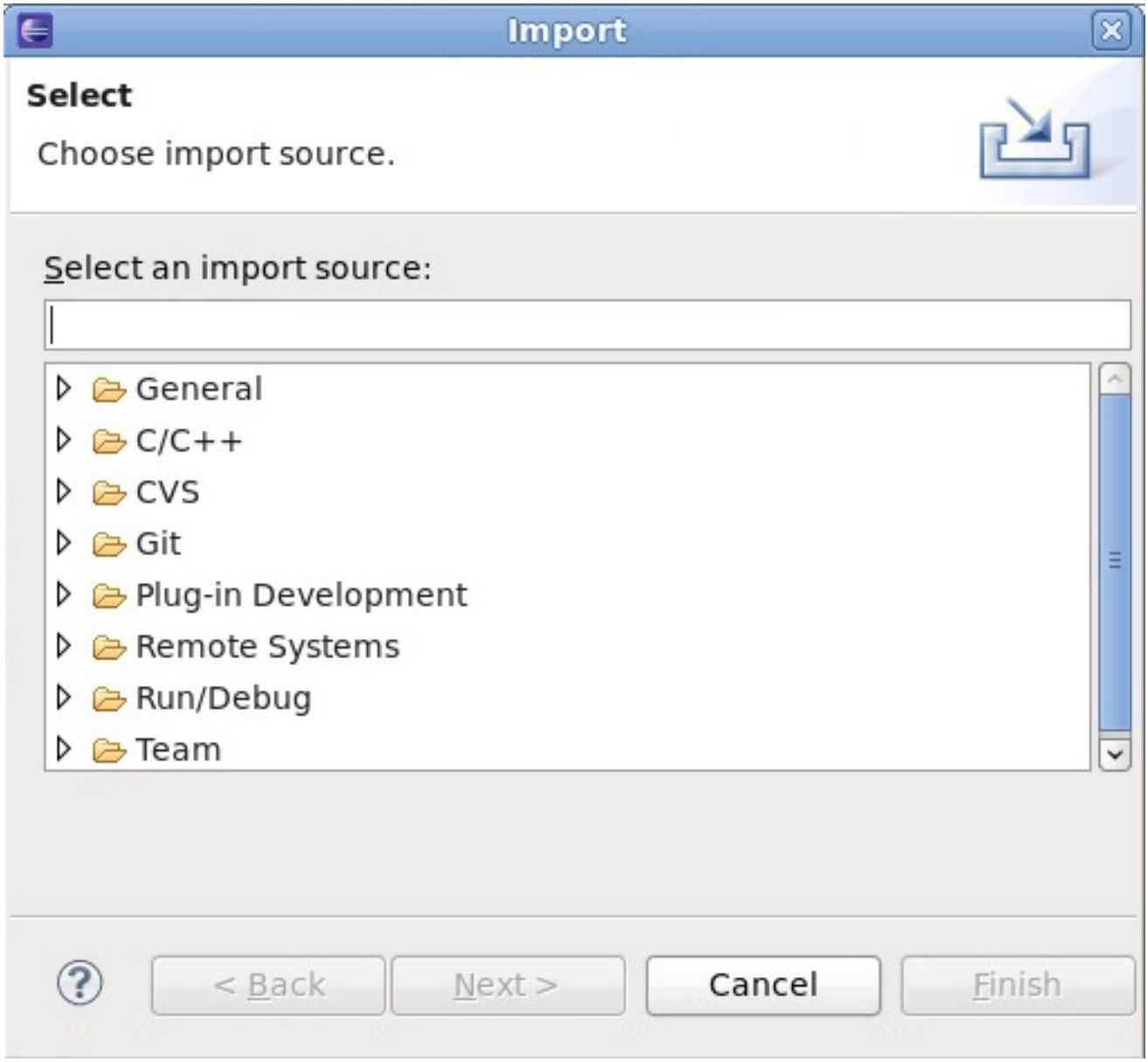


図1.3 Import ウィザード

1.2. Eclipse ユーザーインターフェイス

図1.4「Eclipse ユーザーインターフェイス (デフォルト)」 内のユーザーインターフェイス全体は、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イ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イス (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。例えば、**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イス要素を作成/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イ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イスを提供します。このセクションでは、Eclipse の主要ユーザーインターフェイスの概要を説明します。

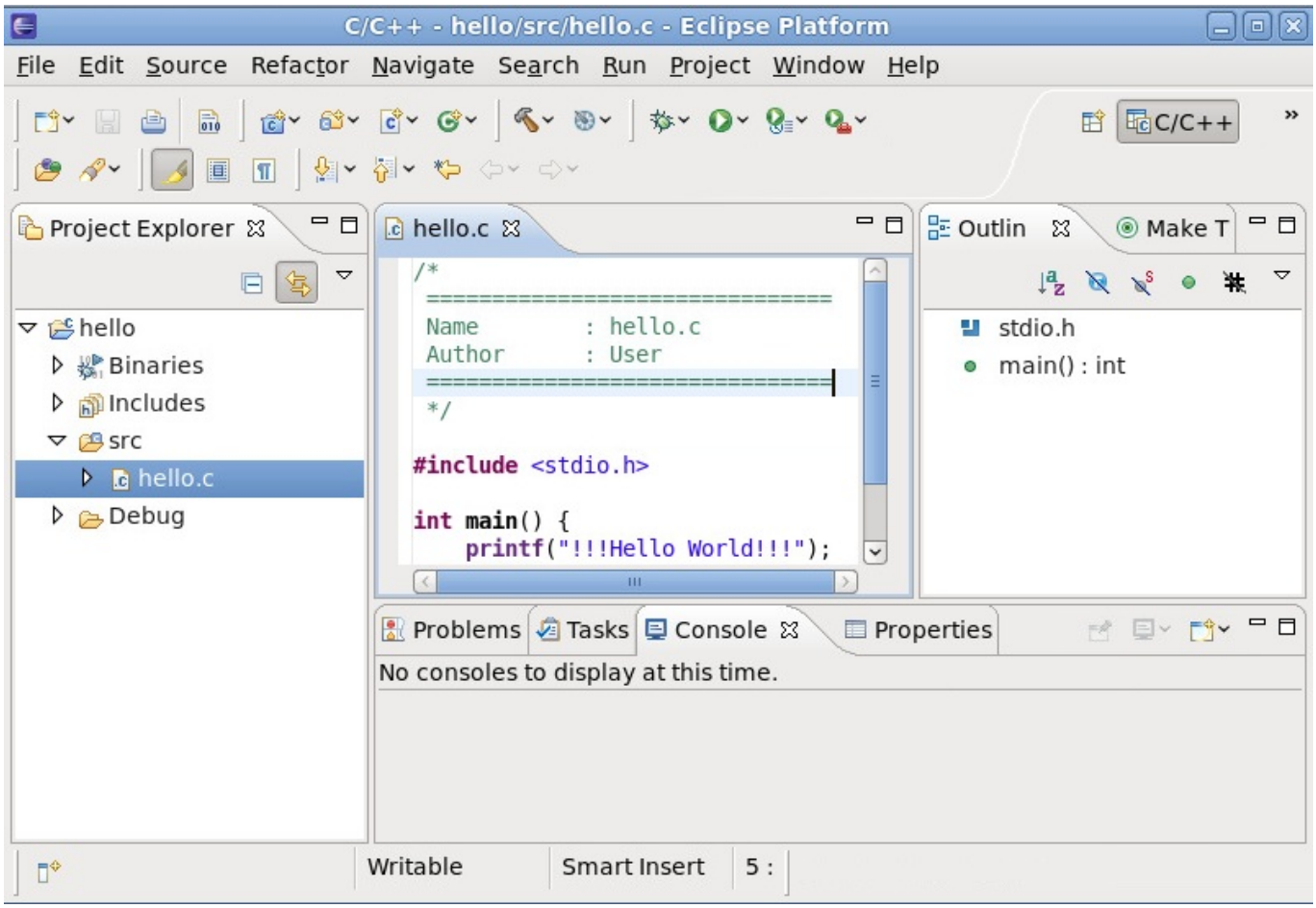


図1.4 Eclipse ユーザーインターフェイス (デフォルト)

図1.4「Eclipse ユーザーインターフェイス (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。

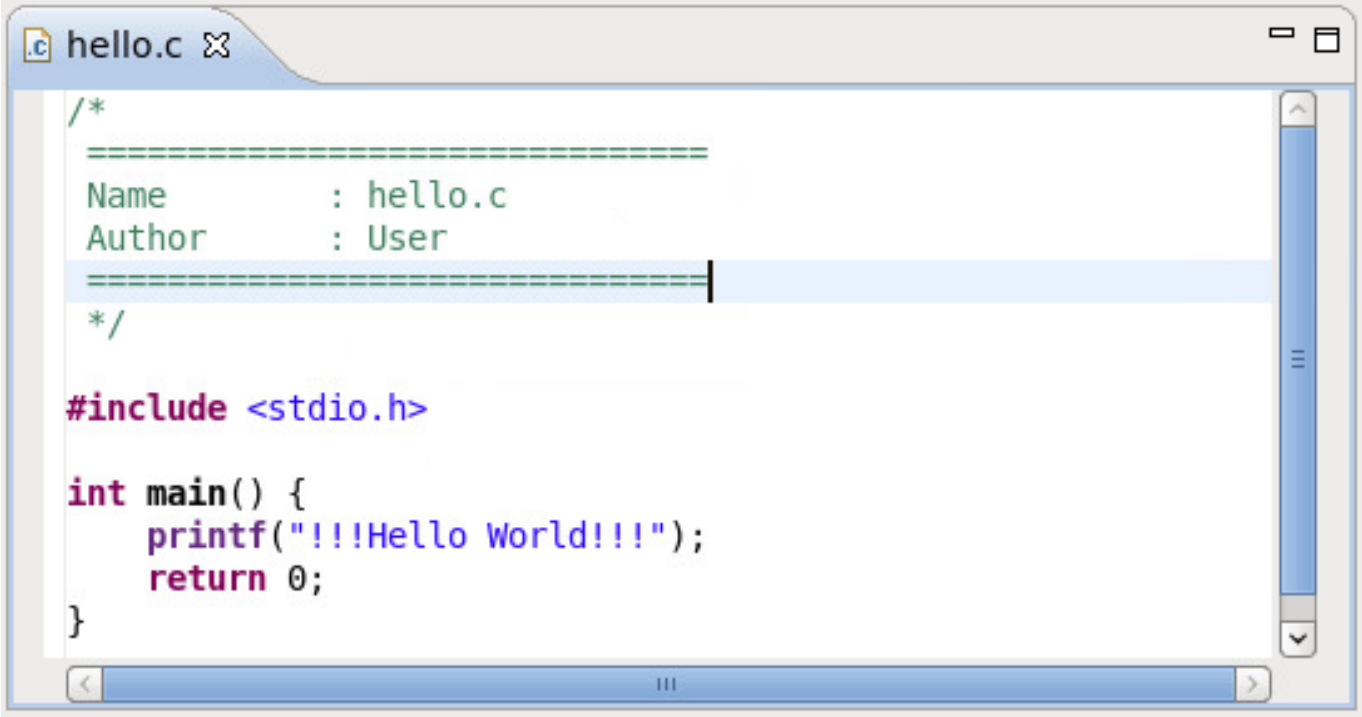


図1.5 Eclipse Editor

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 (例えば、**.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 (例えば、Java、C++)*** > **Code Style** に移動します。

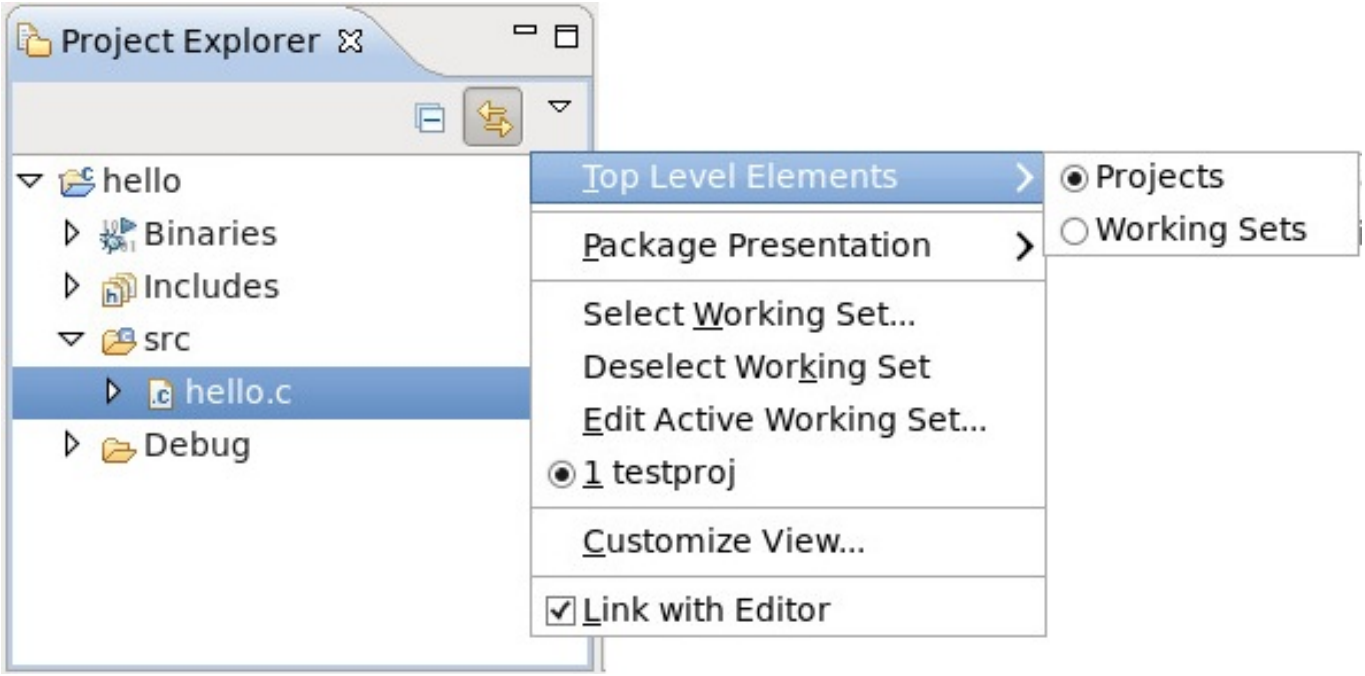


図1.6 Project Explorer

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、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、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。

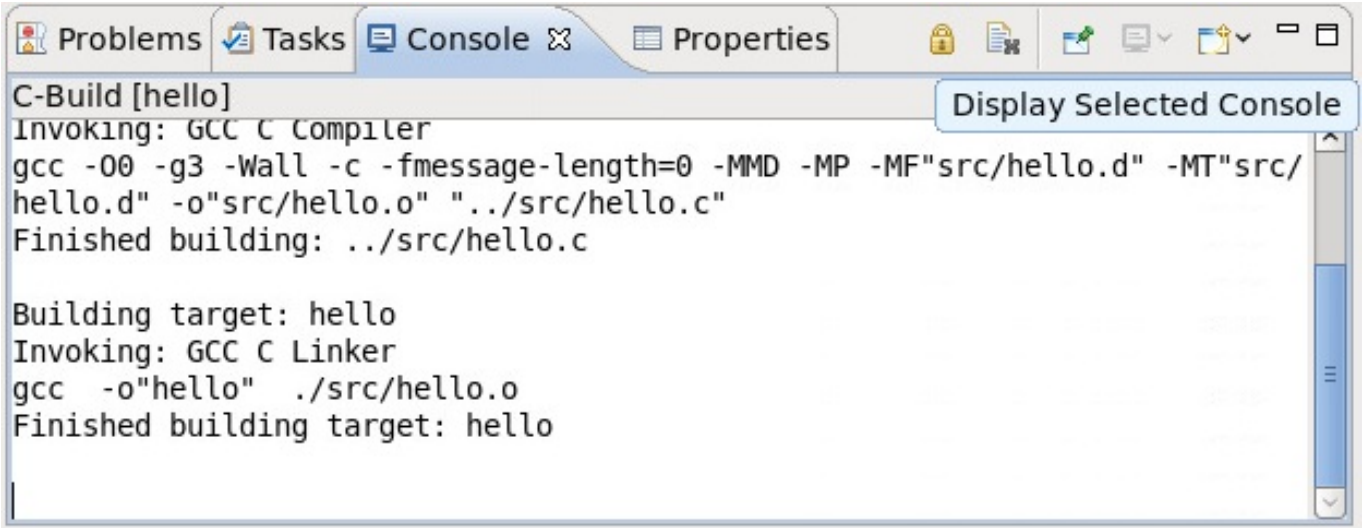


図1.8 Console View

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。

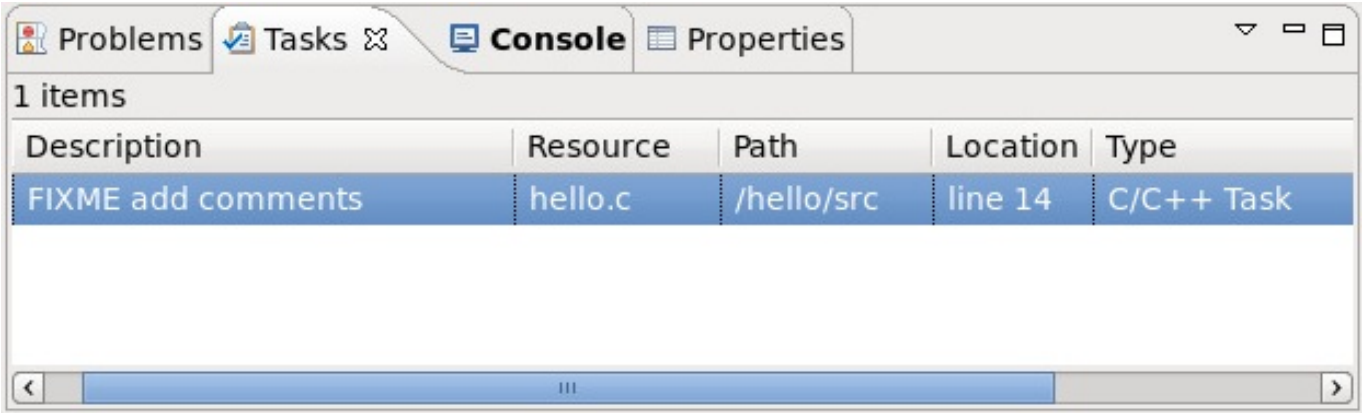


図1.9 Tasks View

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。

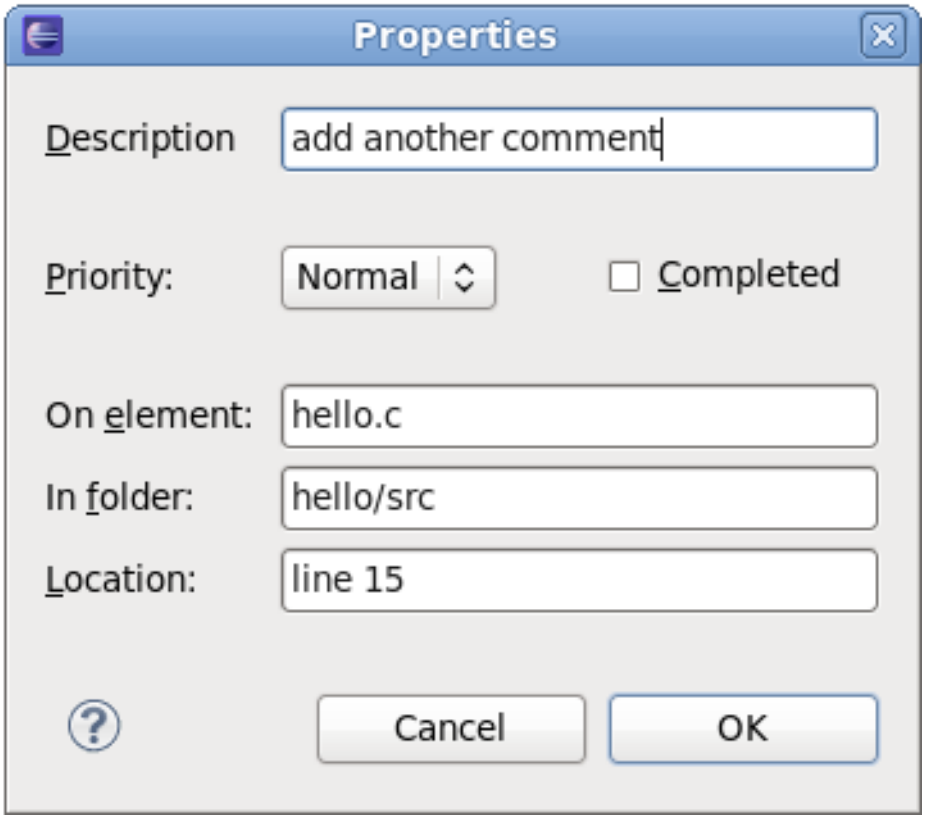


図1.11 Task Properties

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。

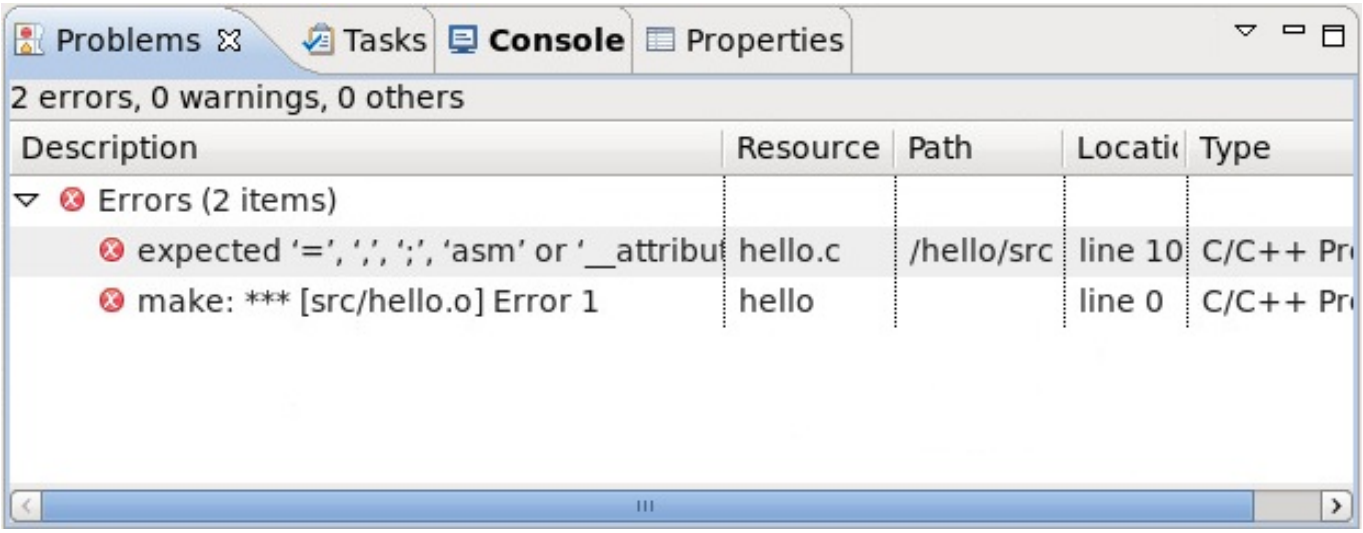


図1.12 Problems View

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。

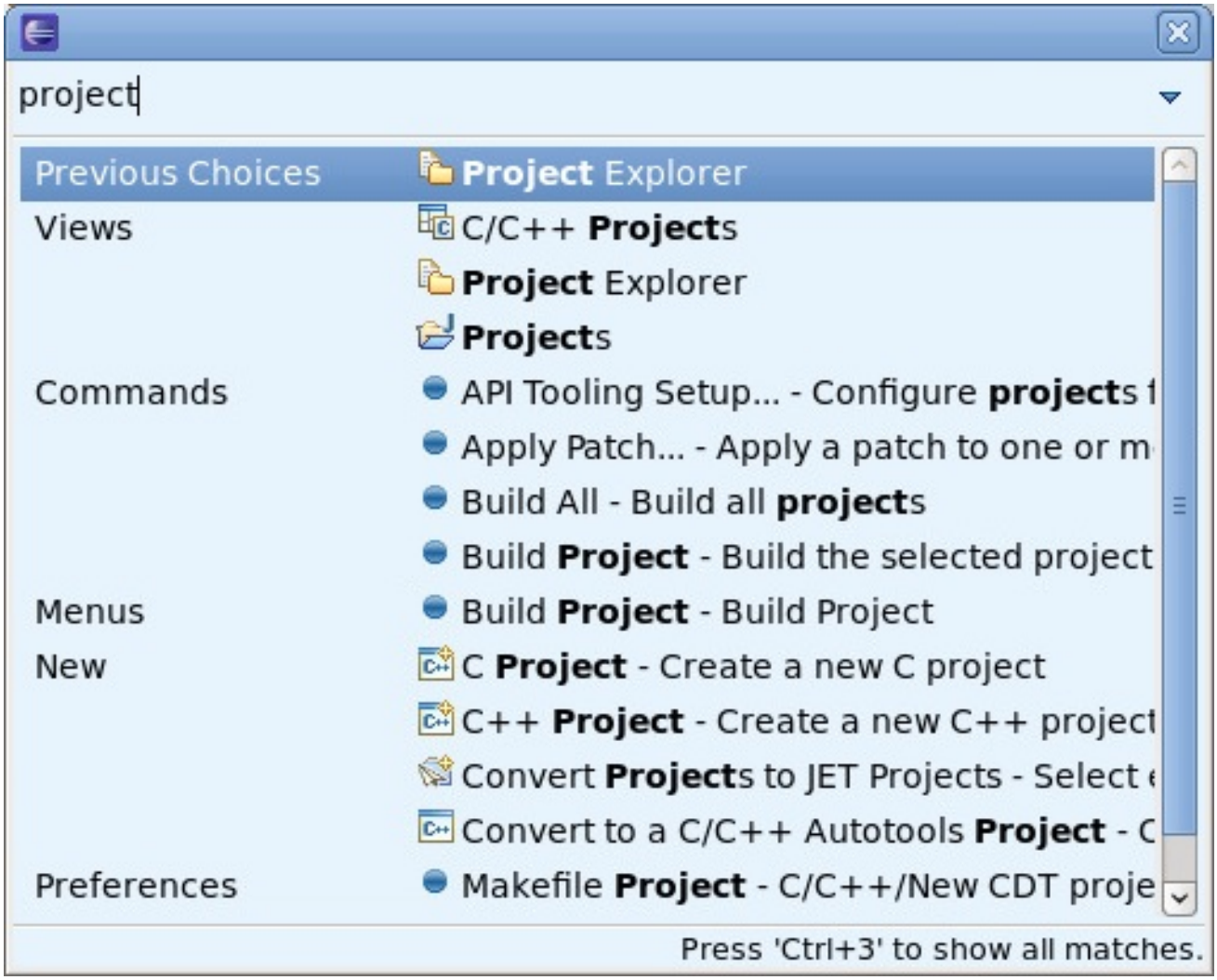


図1.13 クイックアクセスメニュー

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。

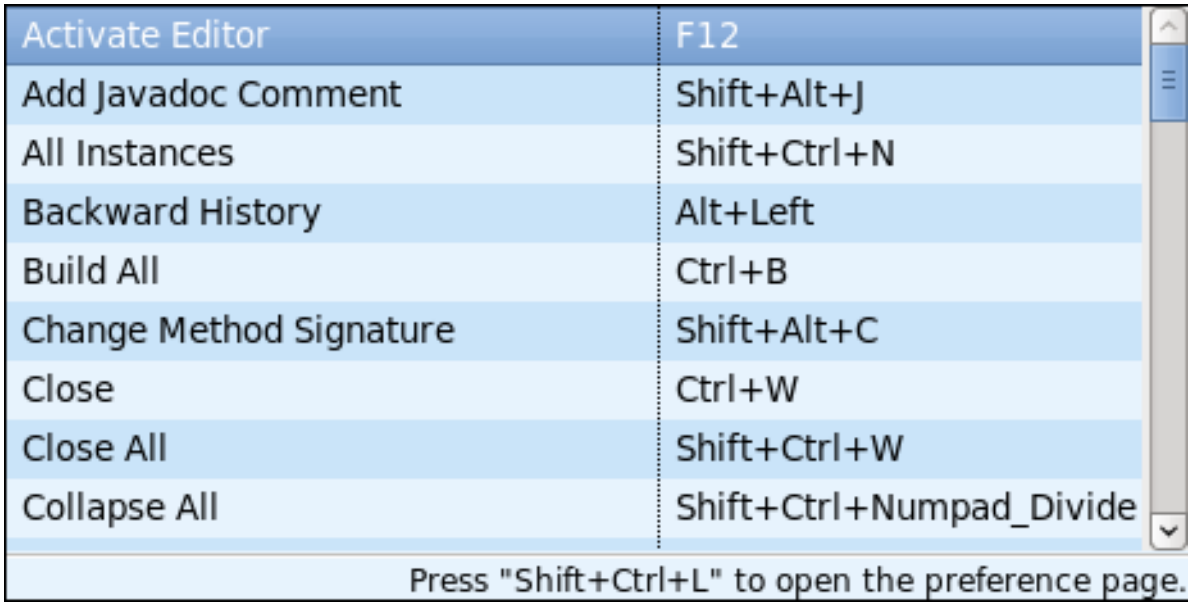


図1.14 キーボードショートカット

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。

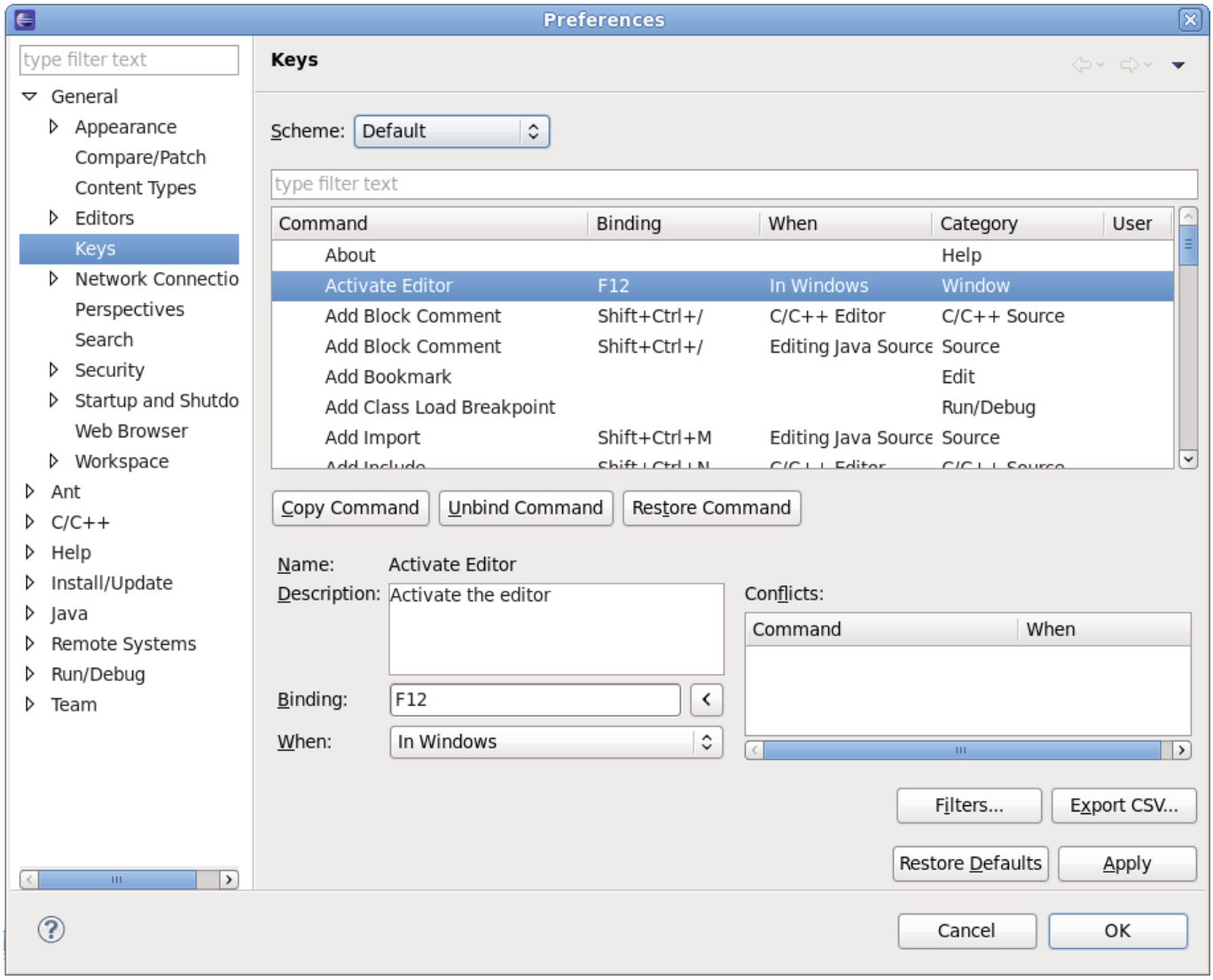


図1.15 キーボードショートカットの設定

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。

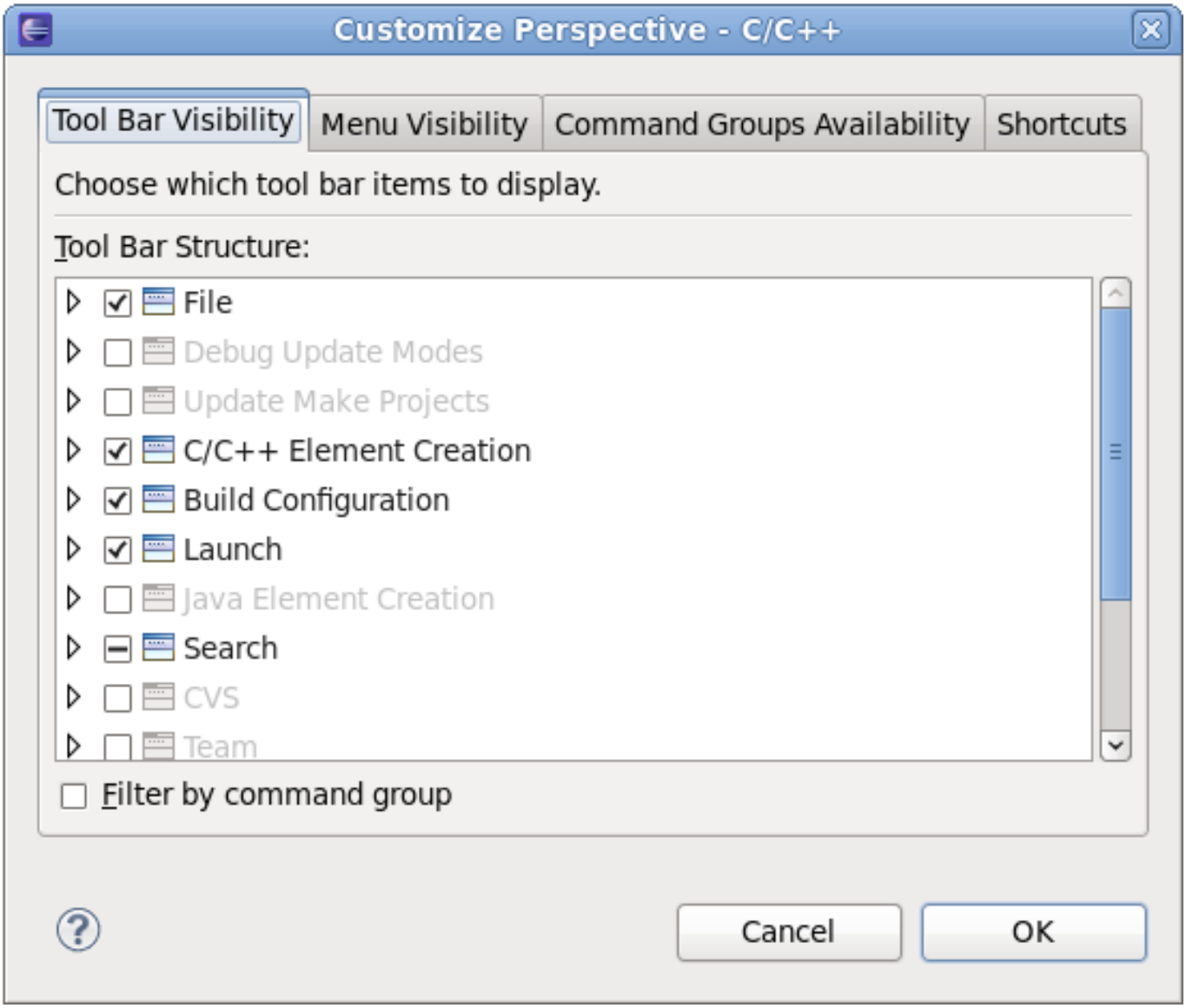


図1.16 パースペクティブメニューのカスタマイズ

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。

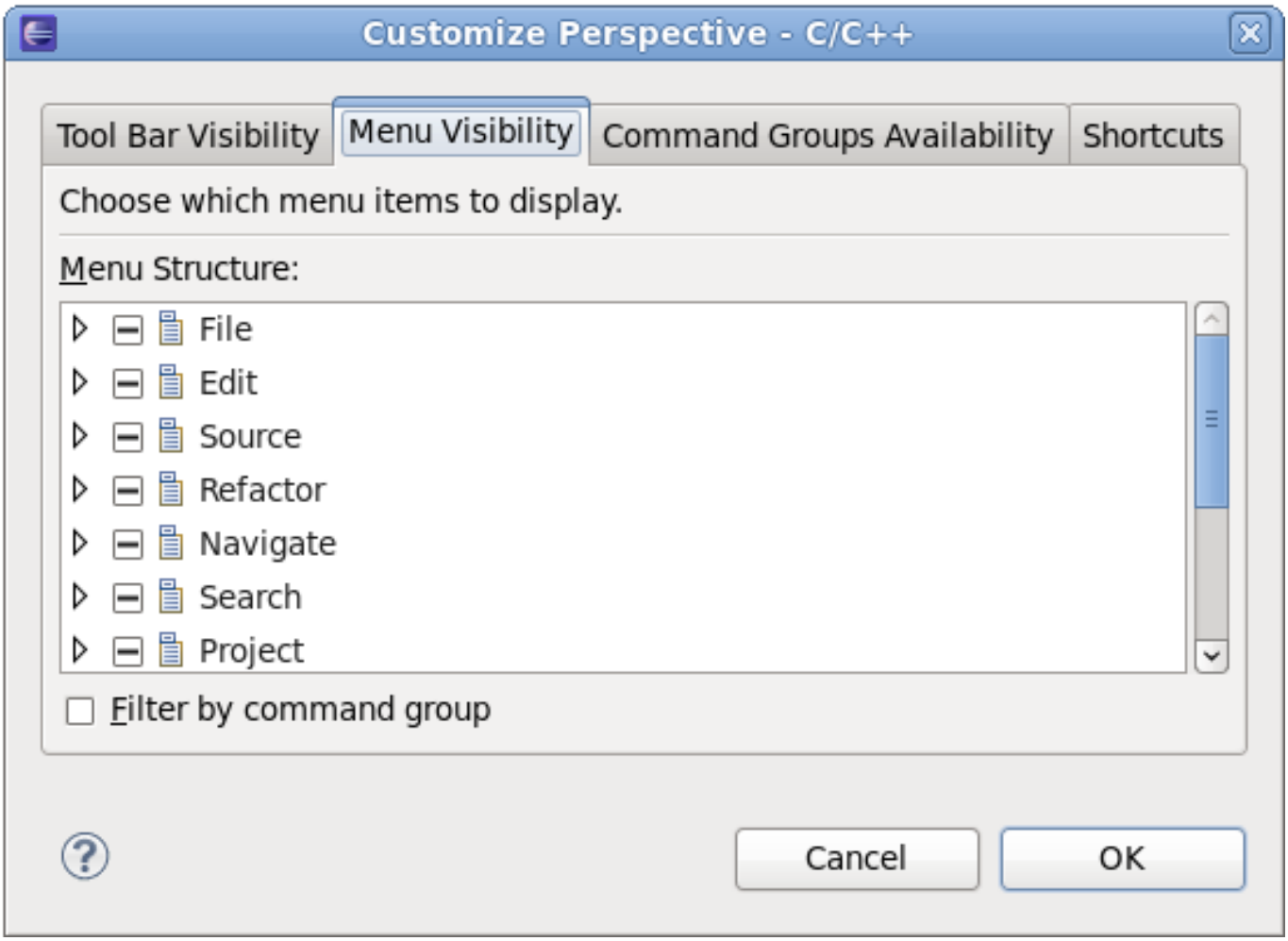


図1.18 Menu Visibility タブ

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。

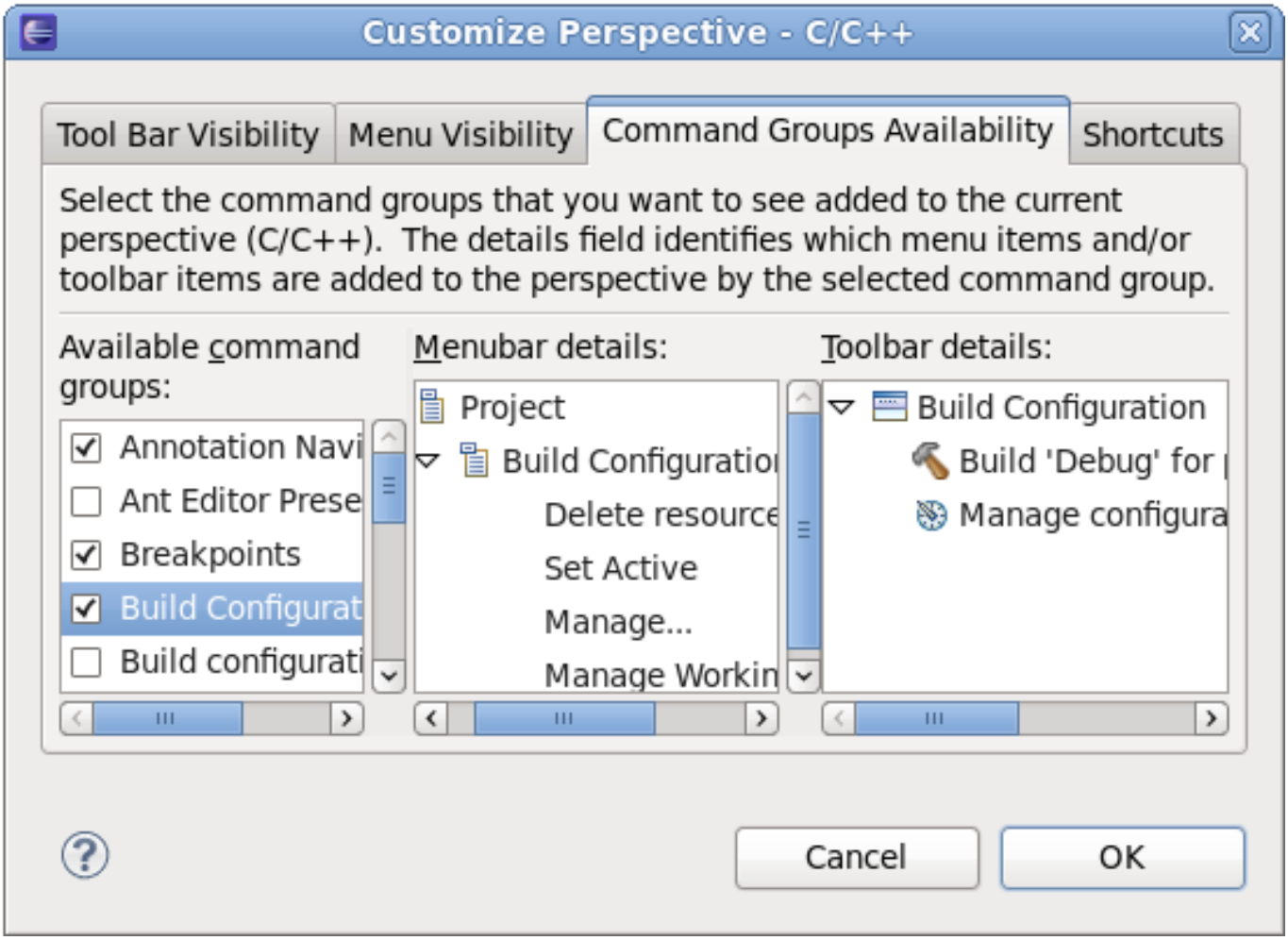


図1.19 Command Group Availability タブ

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。

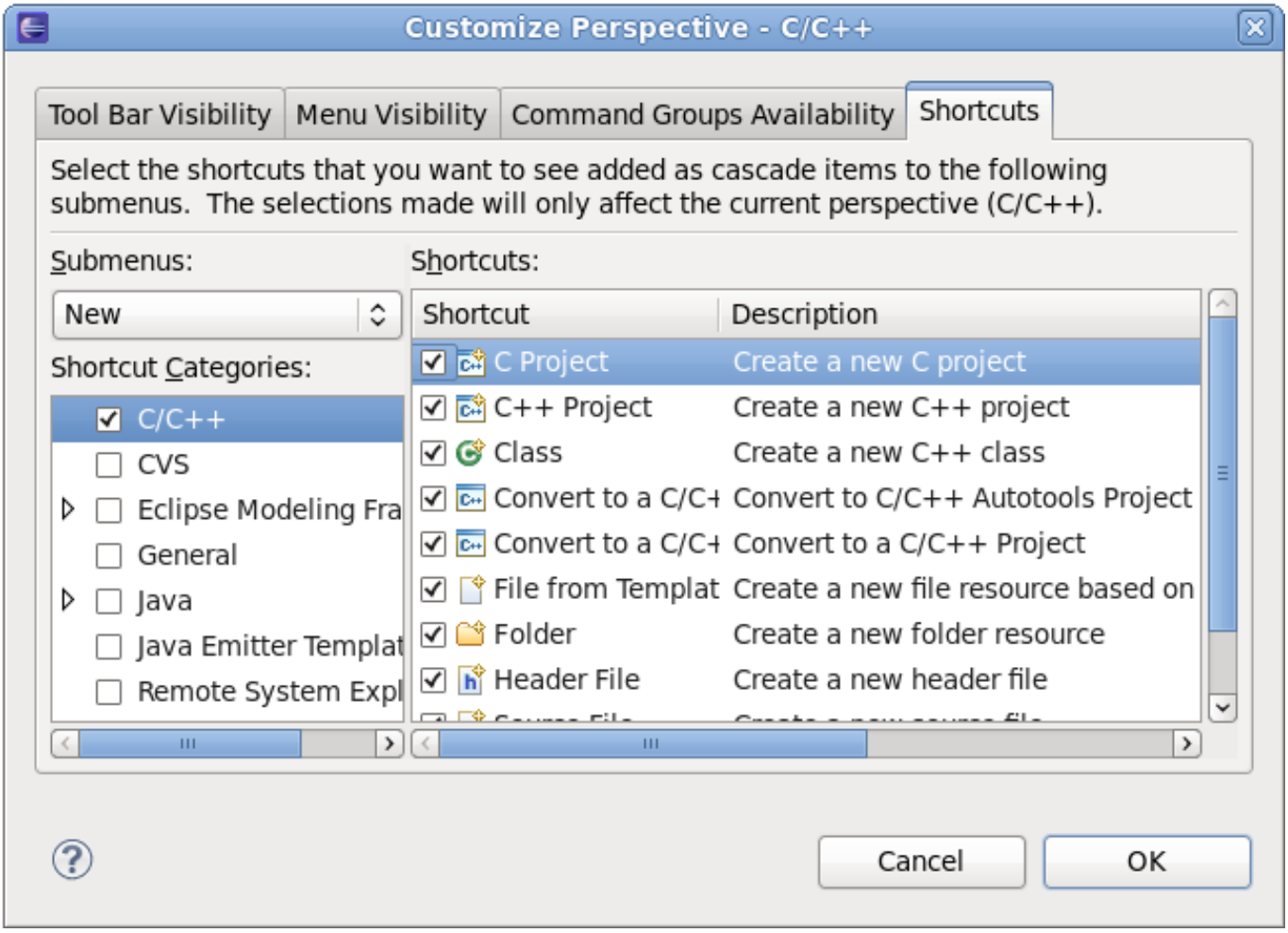


図1.20 Shortcuts タブ

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**) といった比較的ユニークな機能もあります。

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テーションを参照できます。

C++ 言語

メソッド入力候補のドキュメンテーションは 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++ メソッドでは、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テーション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シグネチャ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、入力候補 (**`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 (例えば、prin と入力して **`Ctrl`**+**`Space`** を押すと completion の一つとして**`printf`** が一覧表示されます)、新たなウィンドウでドキュメンテーションが表示され、厳密にどの C 関数が必要かを判断します。

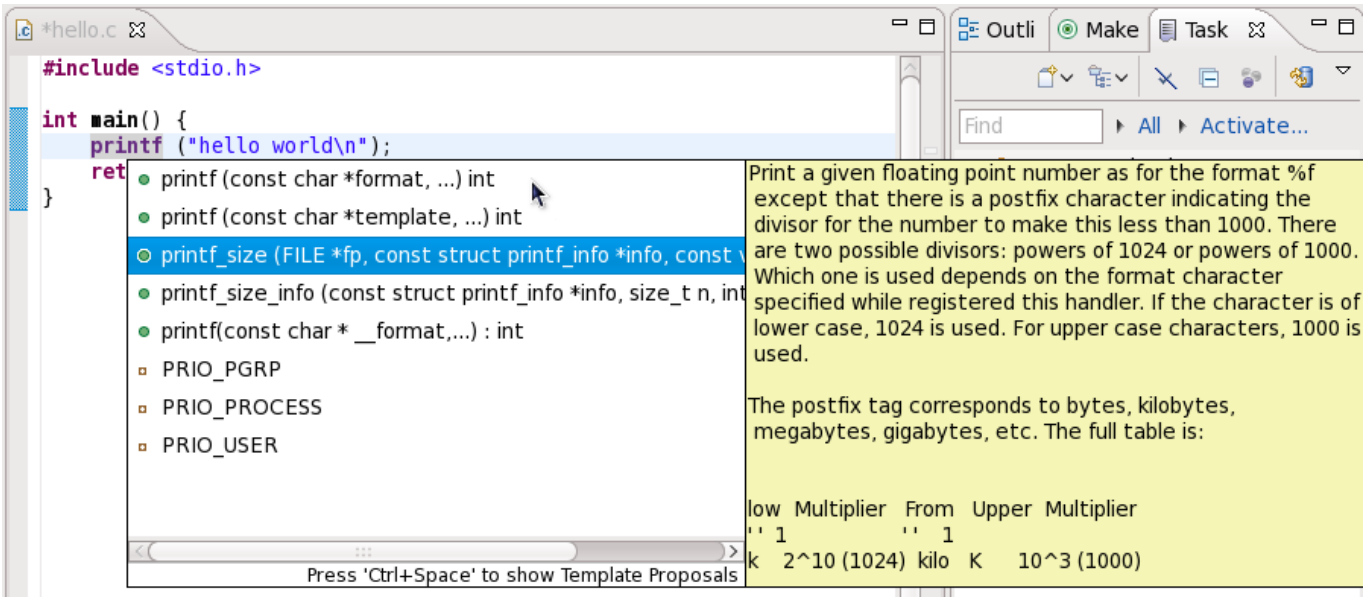


図1.21 Code Completion (入力候補) の使用

C 関数は名前だけでドキュメンテーション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にチェックを入れるか外すかして、そのライブラリのホバーヘルプを有効/無効にします。

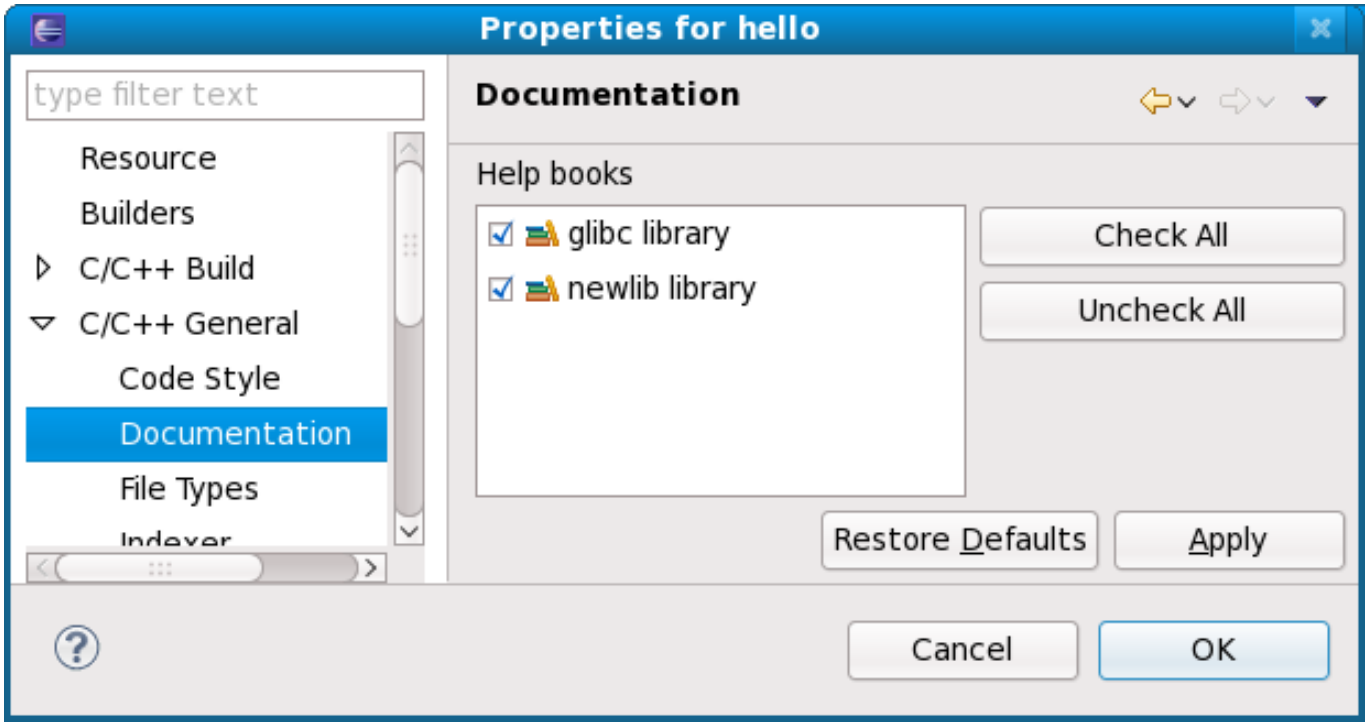


図1.22 ホバーヘルプの有効/無効化

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** ライブラリが有効となっていてライブラリ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ドキュメンテーションが表示されます。

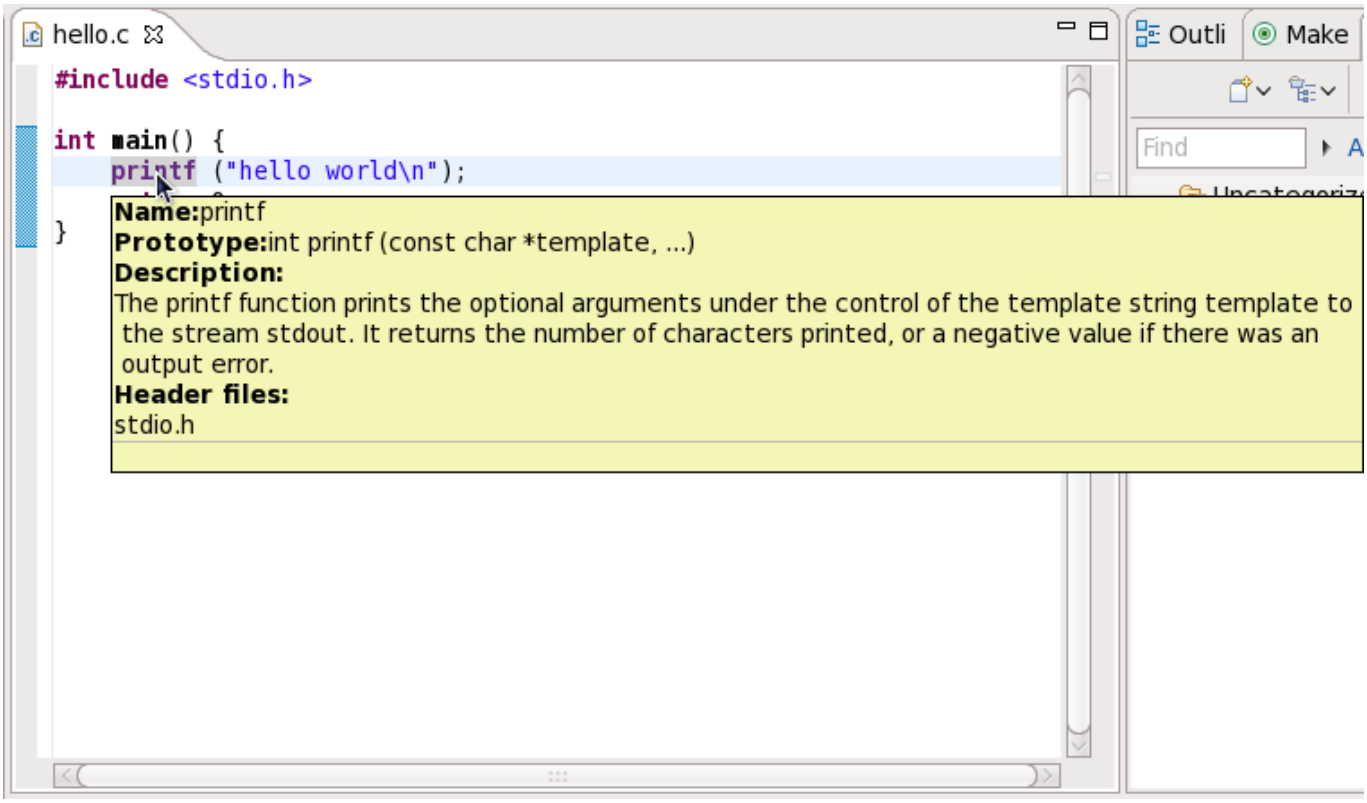


図1.23 ホバーヘルプの使用

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。例えば、**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テーション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 インターフェイスに統合します。**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 インターフェイ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 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 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 のドキュメンテーション

Eclipse には包括的な内部ヘルプライブラリがあり、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 のドキュメンテーションプラグインはすべて、そのコンテンツをこのライブラリにインストールし、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。

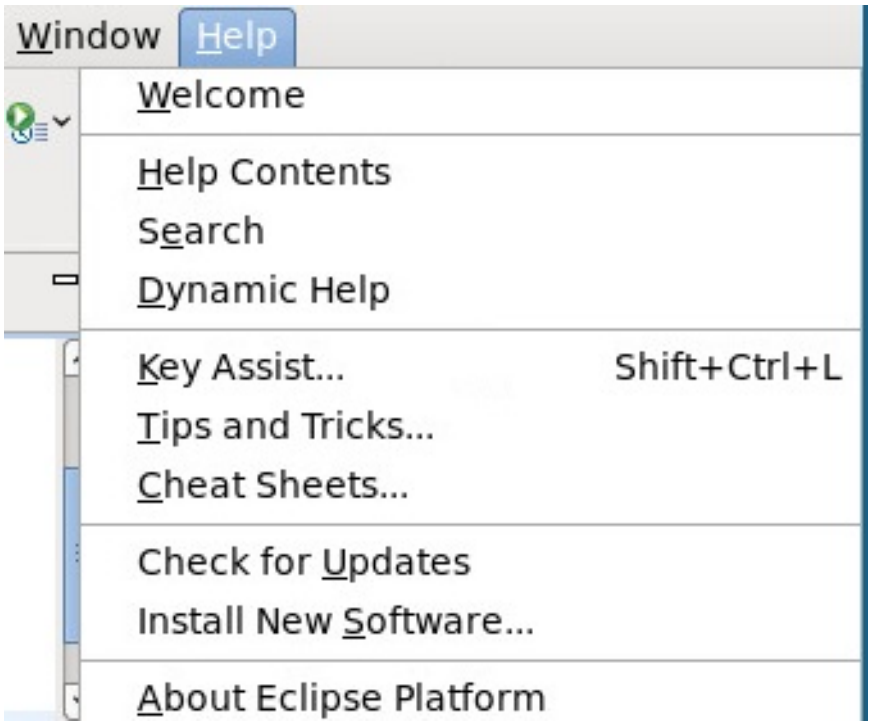


図1.24 Help

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テーション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。

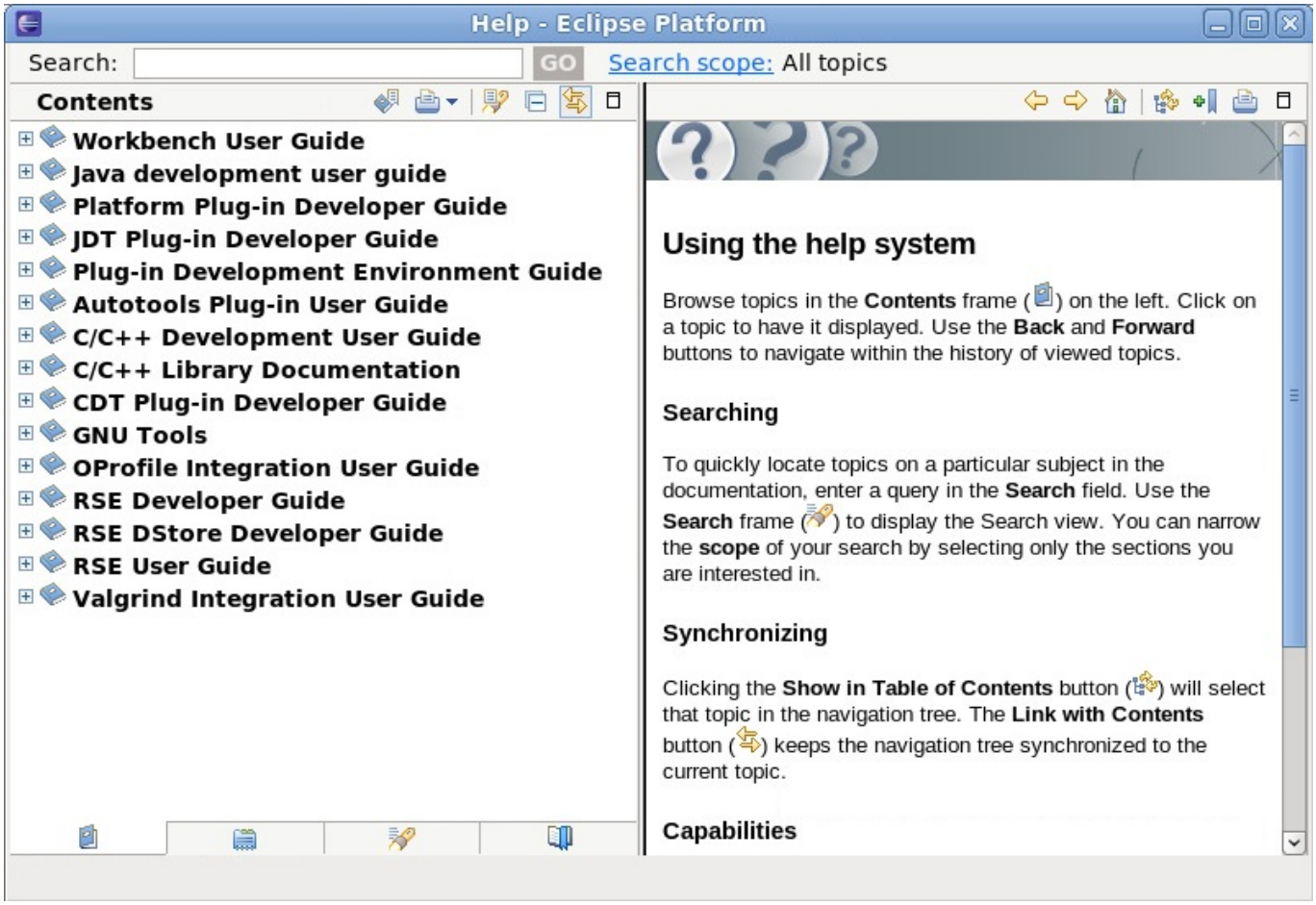


図1.25 Help Menu

Contents フィールド下部のタブは、Eclipse ドキュメンテーション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イ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。このドキュメンテーション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イス要素に関連のあるドキュメンテーションを表示します。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。

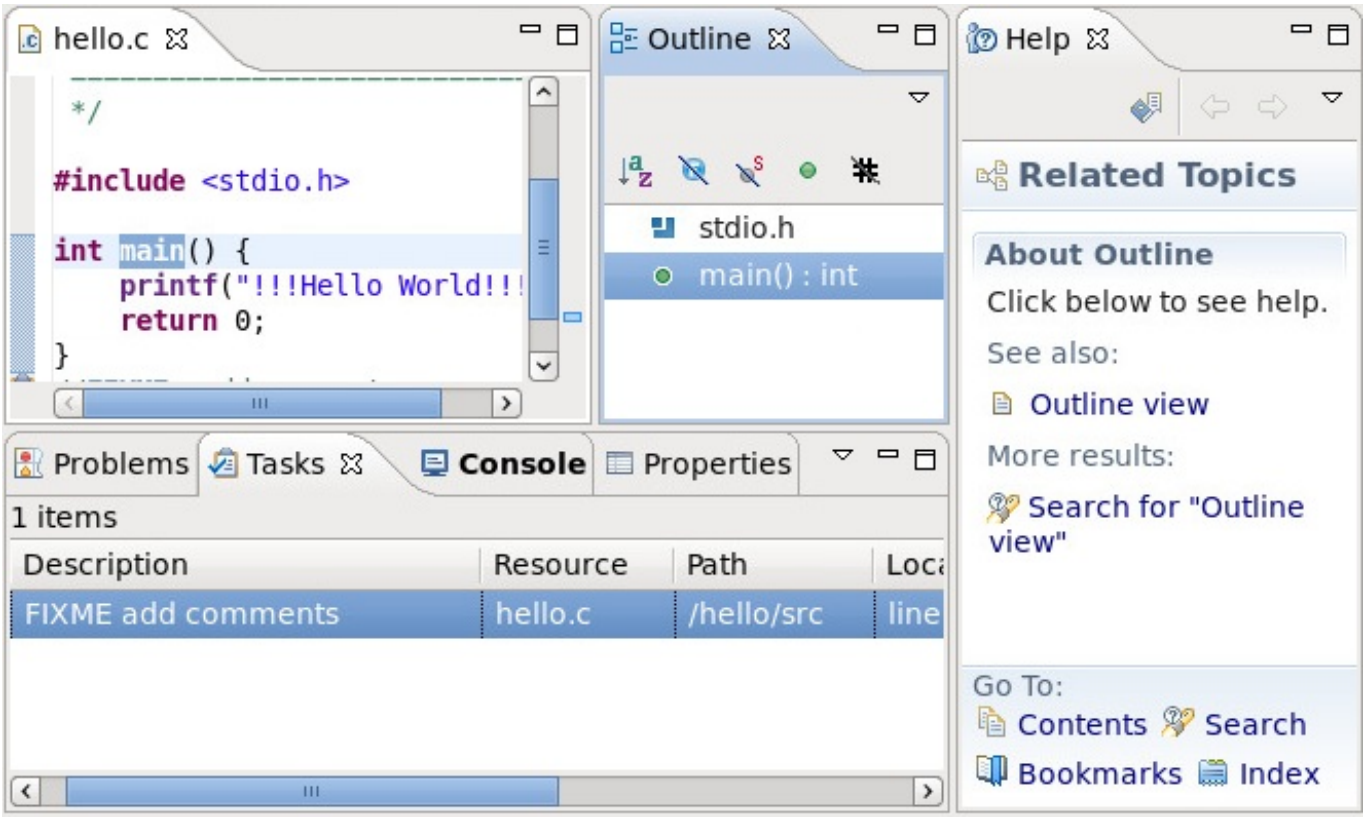


図1.26 動的ヘルプ

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イ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プロジェクトでは、効果的な改訂管理が必須になります。これにより、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 7 は、CVS、SVN、Git というオープンソースの改訂管理システムのなかで 3 つの最も人気のあるものをサポートしています。これらの改訂管理システム向けのツールは、個別の内部コードリポジトリを設定する機能に加え、公開で入手可能な幅広いオープンソースコードへのアクセスを提供します。

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)

Concurrent Versions System (CVS) は、クライアント/サーバーアーキテクチャーの改訂管理システム (RCS) フォーマットをベースとした中央管理のバージョン管理システムです。これは最初のバージョン管理システムで、Subversion (SVN) 以前のものです。

2.1.1. CVS の概要

このセクションでは、CVS の長短両方の様々な要素を見ていきます。

CVS は、ネットワーク接続の信頼性が低く、頻繁に切断していた時期に開発されました。つまり、ファイルをいくつか同時にコミットした際にネットワークが切断されると、コミットは失敗に終わりました。ネットワークの信頼性が低いと、この現象は現在でも起こり得ますが、現状のネットワークインフラストラクチャーではその可能性は低いものとなっています。ネットワーク切断時には、CVS 管理者は 2 つのオプションで問題を解決できます。1 つめは **admin** コマンドを使用してストールがロックされたファイルを取り除き、変更ファイルを取り消す方法です。2 つ目は、コミットコマンドを再発行する方法です。

CVS は バックアップの作成に 1 つの集中した場所を使います。これは不安定なネットワークでは便利なものです。中央化されたアーキテクチャーのため、手動で準備されたトリガー (自動化されたテスト、ビルド、アクセス制御リスト (ACL) バグ追跡システムとの統合) によるコミットポリシーの強制が可能になります。

バックアップへのより詳細なコミットを作成するには、CVS は アットマーク (@) の付いたキーワードを拡張して、コミットの詳細 (例: コミッター名、コミットメッセージ、コミット時間など) をコミットされたファイルに記録することができます。

これらのコミットを継続して追跡するために、CVS はサーバーを使って各ファイルの変更を別個に、最新のものから順に追跡します。こうすることで、最新バージョンが直接保存され、すばやく取り出すことができます。古いバージョンはサーバーが再計算する必要があります。変更され、コミットされたファイルはそれぞれ個別の改訂識別子を付けて追跡されます。こうすると、複数の変更ファイルがコミットされた際にどのファイルが変更されたかを見つけ出すのが困難になる場合があります。これに対処するには、ユーザーはリポジトリ状態にタグ付けを実行し、必要に応じて変更に戻って参照し、この変更を表示すべきです。

CVS リポジトリは 2 つの方法でアクセスできます。リポジトリがクライアント (**:local:** アクセス方法) と同じマシン上にある場合は、クライアントがサーバーを起動します。リポジトリがリモートのマシン上にある場合は、サーバーは rsh/SSH (**CVS_RHS** 環境変数) を使ってクライアントもしくは inet デーモン (**/etc/xinetd.d/cvs**) が起動することができ、異なる認証方法 (例えば、**:gserver:** アクセス方法が Kerberos 認証を統合) が使用可能です。

最後に、CVS ではセキュリティ目的でクライアント/サーバーアプローチが使用されます。つまり、クライアントはサーバーへの接続性に依存し、サーバーアクセスのパーミッションなしにはいかなる操作 (コミットやコミットログの読み取り) も実行できません。

2.1.2. 典型的なシナリオ

以下のコマンドの手順では、**$CVSROOT** ディレクトリに (シグナル **:local:** アクセス方法への絶対パスを使用して) CVSリポジトリを作成する方法を示しています。**$SOURCES** からソースをインポートし、リポジトリから **$WORKDIR** にソースをチェックアウトし、ファイルを修正し、変更をコミットします。

手順2.1 CVS の使用方法

1. CVS ストレージを初期化します。

```
$ mkdir "$CVSROOT"
$ cvs -d "$CVSROOT" init
```

   リポジトリ設定で **$CVSROOT** 下に **CVSROOT** サブディレクトリが作成されます。

2. **$SOURCES** ディレクトリから CVS にコードを**$REPOSITORY** としてインポートし、コミット **$MESSAGE** とともに **$VENDOR_TAG** および **$RELEASE_TAG** をタグ付けします。

```
$ cd "$SOURCES"
$ cvs -d "$CVSROOT" import -m "$MESSAGE" "$REPOSITORY" \
  "$VENDOR_TAG" "$RELEASE_TAG"
```

   $SOURCES コンテンツは**$CVSROOT/$REPOSITORY** 下で CVS にインポートされます。単一 CVS ストレージ内に複数のリポジトリを設置することも可能です。ただし、この例では 1 つにしています。**$VENDOR_TAG** および **$RELEASE_TAG** は、暗黙的なリポジトリブランチ用のタグです。

3. これで異なる開発者がコードを **$WORKDIR** にチェックアウトできます。

```
$ cd "$WORKDIR"
$ cvs -d "$CVSROOT" checkout "$REPOSITORY"
```

警告

元の **$SOURCES** にチェックアウトしないでください。これを行った場合、クライアント側でデータ破損が発生する可能性があり、CVS は様々な CVS 呼び出しでエラーを出力することになります。

4. 最新バージョンの CVS リポジトリが **$REPOSITORY** サブディレクトリに転送されました。開発者は一つのサーバーから複数のリポジトリをチェックアウトすることもできます。

```
$ cd $REPOSITORY
```

5. 新規 **$FILE** の追加をスケジュールするには、以下を使用します。

```
$ vi "$FILE"
$ cvs add "$FILE"
```

6. 開発者は **$EXISTING_FILE** を修正することができます。

```
$ vi "$EXISTING_FILE"
```

7. 最後に、開発者は **$COMMIT_MESSAGE** を付けてこれらの変更をコミットすることができます。

```
$ cvs commit -m "$COMMIT_MESSAGE"
```

$CVSROOT 値を **CVSROOT** 環境変数としてエクスポートすることは可能で、**cvs** ツールはこれを優先します。これにより開発者は **-d "$CVSROOT"** オプションを繰り返し提供する必要がなくなります。この値は初回のチェックアウトで CVS ヘルパーサブディレクトリに保存され、CVS ツールはそこから自動的にこの値を取り出します。

2.1.3. CVS ドキュメンテーション

CVS の man ページは、**man cvs** でアクセスできます。

また、ローカルの FAQ ページは **/usr/share/doc/cvs-*version*/FAQ** にあります。

CVS の情報ページは http://ximbiot.com/cvs/manual/ で入手可能です。

CVS の ホームページは http://www.nongnu.org/cvs/ になります。

2.2. Apache Subversion (SVN)

Subversion は、ファイルやディレクトリ、それらへの変更を管理し、障害時にはそれらをリカバリして確認できるバージョン管理システムです。これは CVS の機能に適合して同一の開発モデルを保存できるように作成されており、CVS で頻繁に発生する問題に対処できるようになっています。このため、CVS ユーザーは最小限の労力で SVN に切り替えができます。

本セクションでは、SVN のインストールと詳細な使用方法を説明します。

2.2.1. インストール

SVN はバイナリパッケージでソースコードから直接またはコンソールからインストールできます。

SVN をインストールする最も簡単な方法は、**yum install subversion** コマンドでコンソールから行います。この方法を選択すると、Red Hat 認定パッケージのみが使用されることを確認でき、手動更新の要件がなくなります。

最後に、SVN はソースコードからもインストールできますが、これは複雑なものとなる可能性があります。SVN Webサイトからソースコードの最新リリースをダウンロードし、**install** の指示にしたがいます。

2.2.2. SVN リポジトリ

SVN を使い始めるためには、まず新たなリポジトリを作成する必要があります。SVN はプロジェクト間の違いを判断できません。ファイルツリーを管理し、プロジェクトを好みに合わせて別のディレクトリに置くのはユーザーの判断になります。以下のコマンドを使用してリポジトリを作成します。

```
# mkdir /var/svn
# svnadmin create /var/svn/repos
# ls /var/svn/repos/
conf db format hooks locks README.txt
```

このコマンドで必要なデータベースファイルのある新規ディレクトリ **/var/svn/repos** が作成されます。

SVN リポジトリには URL でアクセスします。通常、http:// の標準構文が使用されますが、これに限定されるものではありません。以下の URL 形式も使用できます。

file:///

直接リポジトリアクセス (ローカルディスク上)

http://

WebDAV プロトコルで Subversion 対応の Apache サーバーにアクセスします。

https://

http:// と同じですが SSL 暗号化を使用

svn://

カスタムプロトコルを使って **svnserver** サーバーを使用

svn+ssh://

svn:// と同じですが SSH トンネルを使用

重要

URL に空白が含まれている場合は、その前後を引用符で囲んでシェルが単一引数として処理するようにします。引用符を用いないと、URL は無効となります。

2.2.3. データのインポート

複数ファイルで構成されるプロジェクトがすでに作成されたと仮定し、これらが一つのディレクトリに置かれるようにまとめます。**branches**、**tags**、**trunk** という名前で 3 つのトップレベルディレクトリを使用することが推奨されます。これは SVN が必要とするものではありませんが、一般的な慣習です。**trunk** ディレクトリにはプロジェクトファイルを格納し、**branches** および **tags** ディレクトリは空にしておきます。例えば、以下のようになります。

```
myproject/branches/
myproject/tags/
myproject/trunk
  foo.c
  bar.c
  Makefile
```

情報を適切にまとめた後は、これを SVN リポジトリにインポートします。以下のように、**svn import** コマンドで実行します。

```
$ svn import /path/to/mytree \
  http://host.example.com/svn/repo/myproject  \
  -m "Initial import"
Adding  myproject/foo.c
Adding myproject/bar.c
Adding myproject/subdir
Adding myproject/subdir/quux.h

Committed revision 1.
$
```

SVN は、ファイルツリーの設定方法に基づいて必要なディレクトリを作成します。これは作成した URL が

コマンドで表示できます。

```
$ svn list http://host.example.com/svn/repo/myproject
```

2.2.4. 作業コピー

リポジトリにプロジェクトの最初のリビジョンがチェックされた後は、これを修正/作業することができます。これを行うには、作業コピーを作成します。以下のように、**svn checkout** コマンドで実行します。

```
$ svn checkout http://host.example.com/svn/repo/trunk
A  trunk/README
A  trunk/INSTALL
A  trunk/src/main.c
A  trunk/src/header.h
...
Checked out revision 8810.
$
```

これでプロジェクトの作業コピーのあるディレクトリがローカルマシン上に作成されました。以下のコマンドで、プロジェクトのチェックアウト先となるローカルディレクトリを指定することもできます。

```
$ svn checkout http://host.example.com/svn/repo/trunk my-working-copy
```

指定されたローカルディレクトリが存在しない場合は、SVN が作成します。

警告

作業コピー内のディレクトリにはすべて、**.svn** と呼ばれるサブディレクトリが含まれています。これは管理ディレクトリなので、通常は list コマンドでは表示されません。これは重要なファイルなので、削除や変更はしないでください。Subversion はこのディレクトリを使って作業コピーを管理するので、これに手を加えるとエラーが発生したり不安定になります。このディレクトリを誤って削除してしまった場合は、全体を含むディレクトリを削除し (**svn delete** ではなく、通常のシステム削除)、親ディレクトリから **svn update** を実行するのが最善の修正方法です。削除されたディレクトリは、なくなったり変更された **.svn** ディレクトリを含めて再作成されます。これは、データ損失につながる可能性があります。

これで作業コピーは編集可能となりましたが、ファイルツリーが変更される際は常にこの変更をリポジトリにも送信する必要があることに注意してください。これは様々なコマンドで実行できます。

svn add *filename*

新たに作成されたファイルもしくはディレクトリおよびそれに含まれるファイルは、リポジトリに追加されるようフラグが付けられます。次回のコミットでこれらはリポジトリに追加され、これは全員がアクセス/表示することができます。

svn delete *filename*

ファイルもしくはディレクトリおよびそれに含まれるファイルは、リポジトリから削除されるようフラグが付けられます。次回のコミットで削除されます。しかし、削除されたファイルは、以前のリビジョンが SVN でアクセス可能です。

svn copy *filename1 filename2*

filename1 の同一コピーである新規ファイル *filename2* を作成します。そして、次回のコミットで

filename2 を追加するスケジュールを行います。**svn copy** は **--parents** オプションがないと、中間ディレクトリを作成しないことに注意してください。

svn move *filename1* *filename2*

これは、**svn copy *filename1* *filename2*** の次に **svn delete *filename1*** を実行するのと同様の動作になります。コピーが作成され、次回のコミットで *filename1* の削除をスケジュールします。**svn move** は **--parents** オプションがないと、中間ディレクトリを作成しないことに注意してください。

svn mkdir *directory*

このコマンドは、指定されたディレクトリを作成し、次回のコミットでのリポジトリへの追加をスケジュールします。

簡単な変更を加えるために作業コピー全体をチェックアウトするのは現実的でない場合もあります。こういう場合は、リポジトリ URL 上で直接 **svn mkdir**、**svn copy**、**svn move**、**svn delete** を実行することができます。

重要

これらのこのコマンドを使用する際は、その結果を最初に作業コピーで確認する方法がないので、注意してください。

2.2.5. 変更のコミット

変更作業が完了し、間違いがないことを確認したら、他の関係者も変更を閲覧できるように発行します。

作業コピー内の各ファイルで、SVN は 2 つの情報を記録します。

- 現在の作業ファイルの基となっているファイルの作業リビジョン。
- ローカルコピーをリポジトリが最後に更新したことを記録するタイムスタンプ。

SVN はこの情報を使って作業コピーをローカルシステム上で 4 つのカテゴリに分類します。

Unchanged; current (変更なし; 現行)

作業ディレクトリ内のファイルは変更されておらず、リポジトリ内のコピーと一致します。つまり、初回のチェックアウト以降、コミットされた変更はありません。**svn commit** または **svn update** を実行しても何も起こりません。

Locally changed; current (ローカルで変更; 現行)

作業ディレクトリ内のファイルは編集されていますが、リポジトリにはコミットされていません。また、リポジトリのバージョンは、初回のチェックアウト以降、変更されていません。**svn commit** を実行すると、作業ディレクトリ内の変更でリポジトリが更新されます。**svn update** を実行しても何も起こりません。

Unchanged; out of date (変更なし; 期限切れ)

作業ディレクトリ内のファイルは編集されていませんが、リポジトリ内のバージョンは編集されているので、作業コピーは期限切れになってしまっています。**svn commit** を実行しても何も起こりません。**svn update** を実行すると、リポジトリ内の変更とローカルの作業コピーをマージします。

Locally changed; out of date (ローカルで変更; 期限切れ)

作業ディレクトリ内のファイルとリポジトリの両方が変更されています。**svn commit** を先に実行すると、「out-of-date (期限切れ)」エラーが発生します。最初にファイルを更新してください。**svn update** を実行すると、リポジトリ内の変更と作業コピーの変更のマージを試みます。競合があると、競合を解決する最善のアクションをユーザーが選択できるように SVN はオプションを提供します。

svn status を実行すると、リポジトリ内の現行バージョンに一致しない作業ツリー内の全ファイルが文字ごとにコード化されて表示されます。

? アイテム

ファイルが SVN で認識されていません。つまり、作業コピー内にあるものの、リポジトリに追加されていないか、アクションがスケジュールされていません。

A アイテム

ファイルがリポジトリに追加するようスケジュールされており、次回のコミットで追加されます。

C アイテム

ファイルはリポジトリでなされた変更と競合しています。つまり、作業コピーで現在変更されたファイルの同一セクションを別のユーザーが編集してコミットしていて、SVN はどちらが「正しい」か分からない状態です。変更がコミットされる前にこの競合を解決する必要があります。

D アイテム

ファイルは次回のコミットで削除されるようにスケジュールされています。

M アイテム

ファイルが修正され、次回のコミットで変更が更新されます。

--verbose (**-v**) が **svn status** で実行されると、変更されていないものも含め、作業コピー内の全アイテムのステータスが表示されます。例えば、以下のようになります。

```
$ svn status -v
M 44 23 sally README
 44 30 sally INSTALL
M 44 20 harry bar.c
 44 18 ira stuff
 44 35 harry stuff/trout.c
D 44 19 ira stuff/fish.c
 44 21 sally stuff/things
A  0  ?  ? stuff/things/bloo.h
 44 36 harry stuff/things/gloo.c
```

表示されているのは順に、文字コード、作業リビジョン、アイテムが最後に変更されたリビジョン、変更者名、変更されたアイテム、になります。

チェックアウトが最後に実行されてからリポジトリ内で修正されたアイテムを確認することも役に立ちます。これは、**svn status** で **--show-updates** (**-u**) の実行で可能です。アスタリスク (*) が文字ステータスと **svn commit** 実行時に更新されるファイルの作業リビジョン番号の間に表示されます。

変更を表示するもう一つの方法は、**svn diff** コマンドです。これにより、変更部分が統一された差分フォーマットで表示され、ファイルのコンテンツの「スニペット」と説明されます。各行の先頭には以下の文字が付けられます。空白の場合は変更なし、マイナス記号 (-) の場合は削除された行、プラス記号 (+) の場合は追加された行を示します。

競合が発生する場合もあります。SVN は最も一般的な 3 種類の応答を返し (postpone、diff-full、edit)、4 つ目のオプションは全オプションとその内容を一覧表示します。利用可能な各オプションは以下の通りです。

(p) postpone

競合を後で解消するようにマークします。

(df) diff-full

ベースリビジョンと競合ファイルの違いを統一された差分フォーマットで表示します。

(e) edit

マージされたファイルをエディターで変更します。

(r) resolved

マージされたバージョンのファイルを受け付けます。

(mf) mine-full

ユーザーのファイル全体を受け入れ、リポジトリの最新の変更を無視します。

(tf) theirs-full

ユーザーと競合している側のファイル全体を受け入れ、ローカルの作業コピーの最新の変更を無視します。

(l) launch

競合解消のために外部ツールを起動します (これには外部ツールを前もって設定しておく必要があります)。

(h) help

ここで説明されたオプションを一覧表示します。

最後に、プロジェクトがローカルで変更され、競合が解消された場合、この変更は **svn commit** コマンドに **-m** オプションをつけてコミットすることができます。

```
$ svn commit filename -m"Fixed a typo in filename"
Sending filename
Transmitting file data .
Committed revision 57.
$
```

上記の例のように、**-m** オプションを使うとコミットメッセージを記録することができます。メッセージが分かりやすいと役に立ち、コミットに戻って参照する場合も分かりやすくなります。

これでリポジトリにアクセス可能なユーザーはだれでも最新の更新バージョンを使ってローカルで最新コピーに更新できるようになります。

2.2.6. SVN ドキュメンテーション

svn --help コマンドは、SVN と一緒に使用する利用可能なコマンドに関する情報を提供し、**svn *subcommand* --help** は特定のサブコマンドに関するより詳細な情報を提供します。

SVN の公式ガイドは http://svnbook.red-bean.com/ で入手可能です。

SVN の公式 Web サイトは http://subversion.apache.org/ になります。

2.3. Git

Git は CVS や SVN の改善というよりも、それらに対抗するために開発されたバージョン管理システムです。Git は以下の 4 つの点を念頭に作成されました。

1. CVS や SVN とは異なる。Git の作成者である Torvalds 氏は、これらのプログラムを好まなかったので、これらと違うものを作成したいと思いました。
2. BitKeeper のような作業フローをサポート。理想的なプロジェクトの管理方法は BitKeeper と同様のプロセスとする一方で、独自の設計を維持して BitKeeper のクローンとはならないようにする。
3. アクシデントもしくは悪意による破損に対する強固な安全対策。
4. 非常に高いパフォーマンス。

これらを達成するために、Git はこれまでとは違う方法でデータ処理方法にアプローチしています。

このセクションでは、通常の Git の使用方法で最も一般的なプロセスについて説明します。

(CVS や SVN などの) 以前のバージョン管理は、データを各ファイルの基本バージョンに対する変更として処理していました。Git ではその代わりにデータ変更をファイルがどのように見えるかという別個のスナップショットとして扱い、該当ファイルへのリファレンスを保存します (ただし、ファイルが変更されていなければ、Git は別のファイルをコピーするのではなく、以前の同一バージョンへのリンクを保存します)。これにより、新たなミニファイルシステムが作成されます。下図は、これらの概念を視覚的に比較したものです。

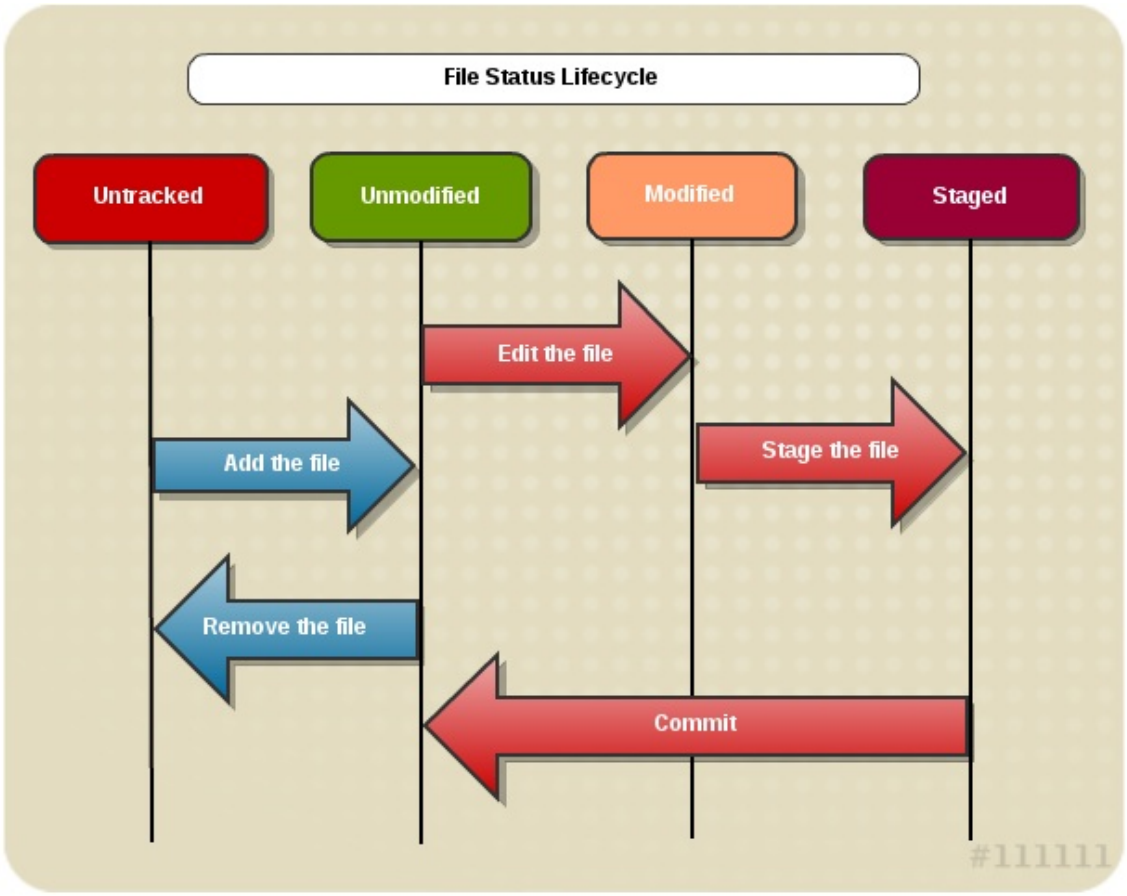


図2.1 Git バージョン管理

Git は継続的にリモートのリポジトリに接続する必要がないので、非常に高速なものになっています。Git のスナップショットの性質と全バージョンがローカルのファイルシステムに保存される方法が意味するのは、ほとんどすべてのことがネットワークに接続することなくでき、プロジェクトの履歴がローカルで入手可能ということです。

Torvalds 氏の完全な要件を満たすために、Git ではすべてのものが保存前にチェックサムを行い、その後にそのチェックサムで照合されます。つまり、Git が知らない間にコンテンツが変更されるということがなく、情報が転送中に失われたり、破損したりすることがありません。SHA-1 ハッシュメカニズム (40 文字の 16 進法の文字列) がこのために使われています。

また、Git では元に戻すことができないものはほとんどありません。これは、ファイルが以下の 3 つの状態のいずれかに属することが役に立っています。

Committed (コミット済み)

データは安全にローカルのデータベースに保存され、変更されていません。

Modified (変更済)

ファイルは変更されているが、データベースにはコミットされていない。

Staged

修正されたファイルが現行バージョンでコミットされるようにマークされている。

2.3.1. インストール

Git はソースまたはコンソールのいずれかからインストールできます。ユーザーに自信がある場合は、ソースからのインストールが推奨されます。これは、バイナリインストーラーには常に最新のバージョンがあるとは限らないためです。

ソースコードから Git をインストールする場合は、以下の手順にしたがいます。

手順2.2 ソースコードからの Git インストール

1. Git が依存する以下のライブラリをインストールします。**curl**、**zlib**、**openssl**、**expat**、**libiconv**。

```
# yum install curl-devel expat-devel gettext-devel \
openssl-devel zlib-devel gcc
```

2. 以下のアドレスにある Git Web サイトから最新のスナップショットをダウンロードします。http://git-scm.com/download

3. コンパイルして、インストールします。

```
# tar -zxf git-1.7.6.1.tar.gz
# cd git-1.7.6.1
# make prefix=/usr/local
# make prefix=/usr/local install
```

4. これで Git から Git のアップデートを入手することが可能になりました。

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
```

コンソールからバイナリインストーラーで Git をインストールするには、単に以下のコマンドを実行します。

```
# yum install git
```

2.3.2. 初期設定

インストール後に Git をパーソナル化するステップを実行して使用する準備を整えます。これらのセッティングが設定されると、以降の Git セッションで維持されます。ただし、これ以降に変更する場合は、コマンドを再度実行します。

これらの変更は、3 つの別の場所の保存されている変数を変更することで実行されます。

1. **/etc/gitconfig** ファイルにはシステム上の全ユーザーとその全リポジトリ用の変数が含まれています。ここには基本設定が格納されており、**--system** オプションを **git config** に指定すると、このファイルから読み書きをするように設定します。
2. **~/.gitconfig** ファイルはユーザー固有のものです。**--global** を指定すると、Git にこのファイルへの読み書きを指示し、上記 1 番の設定に優先します。
3. 現在使用しているリポジトリの Git ディレクトリ内の設定ファイルです (**.git/config**)。これは、このリポジトリのみに固有のもので、1 および 2 番の設定に優先します。

初回のコミット前に、変更の際に表示される名前と E メールアドレスを提供して、Git に詳細を記入します。

例えば、ユーザーの名前が John Q. Smith であれば、以下のコマンドを使用します。

```
$ git config --global user.name "John Smith"
$ git config --global user.email "jsmith@example.com"
```

上記の説明にあるように、**--global** オプションを指定すると、これらの設定は維持されますが、特定のリポジトリで無効にすることもできます。

エディターが必要な場合は、Git はデフォルトで Vi もしくは Vim を起動します。ただし、これらが好みに合わない場合は、別のエディターに変更することもできます。これを行うには、以下のコマンドを使用します。

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

差分ツールは様々なファイルで変更を表示させる際に使用し、コミット前に確認する際に役立ちます。

以下のコマンドを使って好みの差分ツールを設定します。

```
$ git config --global merge.tool DiffTool
```

最後に、これらの設定が正しく行われているかを確認します。以下のコマンドを実行します。

```
$ git config --list
user.name=John Smith
user.email=jsmith@example.com
```

異なるファイルで異なる設定になっている場合、Git はすべてをリッスンし、最後の値をアクティブなものとします。また、**git config *{key}*** コマンドを使用して、Git が変数への特定の応答を確認するようにすることもできます。例えば、以下のようになります。

```
$ git config user.name
John Smith
```

2.3.3. Git リポジトリ

Git リポジトリには、プロジェクトのメタデータやオブジェクトデータベースが保存されます。ローカルシステム上でリポジトリのローカルのクローンを作成する際は、ここからプロジェクトがプルされます。

Git リポジトリを取得するには 2 つの方法があります。1 つ目は、ディレクトリが既に存在し、Git リポジトリを初期化する必要がある場合に使われます。2 つ目は、既に存在するリポジトリをクローンする方法です。

既存のリポジトリ (例えば投稿するための) をクローンするには、以下のコマンドを実行します。

```
$ git clone git://location/of/git/repository.git
```

このコマンドは **`git clone`** であって、CVS や SVN などのバージョン管理システムで使われる**`git checkout`** ではないことに注意してください。これは、Git ではプロジェクトの全履歴にあるすべてのファイルのコピーが渡されるのに対し、他のバージョン管理システムでは最新のファイルのみが渡されることが理由です。

上記のコマンドでディレクトリが作成され、その名前は **`.git`** 接尾辞のない URL の最後の部分になります。しかし、**`clone`** コマンドは最後に好みのディレクトリ名を追加することで、どんな名前でも使用することができます。

```
$ git clone git://location/of/git/repository.git my_git_repo
```

最後に、このコマンドでは **`git://`** プロトコルを使用しますが、必要に応じて**`http://`** や **`https://`** を使うこともできます。

データ作成が可能な新規 Git リポジトリを作成するには、まずプロジェクトのディレクトリに移動し、以下を実行します。

```
$ git init
```

これで Git リポジトリのスケルトンが作成され、ここにはコンテンツの作成および追加に備えた必要なファイルすべてが含まれます。

これで スケルトンの Git リポジトリが設定されたか、ローカルクローンがコピーされローカルシステム上で準備が整ったので、Git サイクルの残りの部分を見ていきましょう。

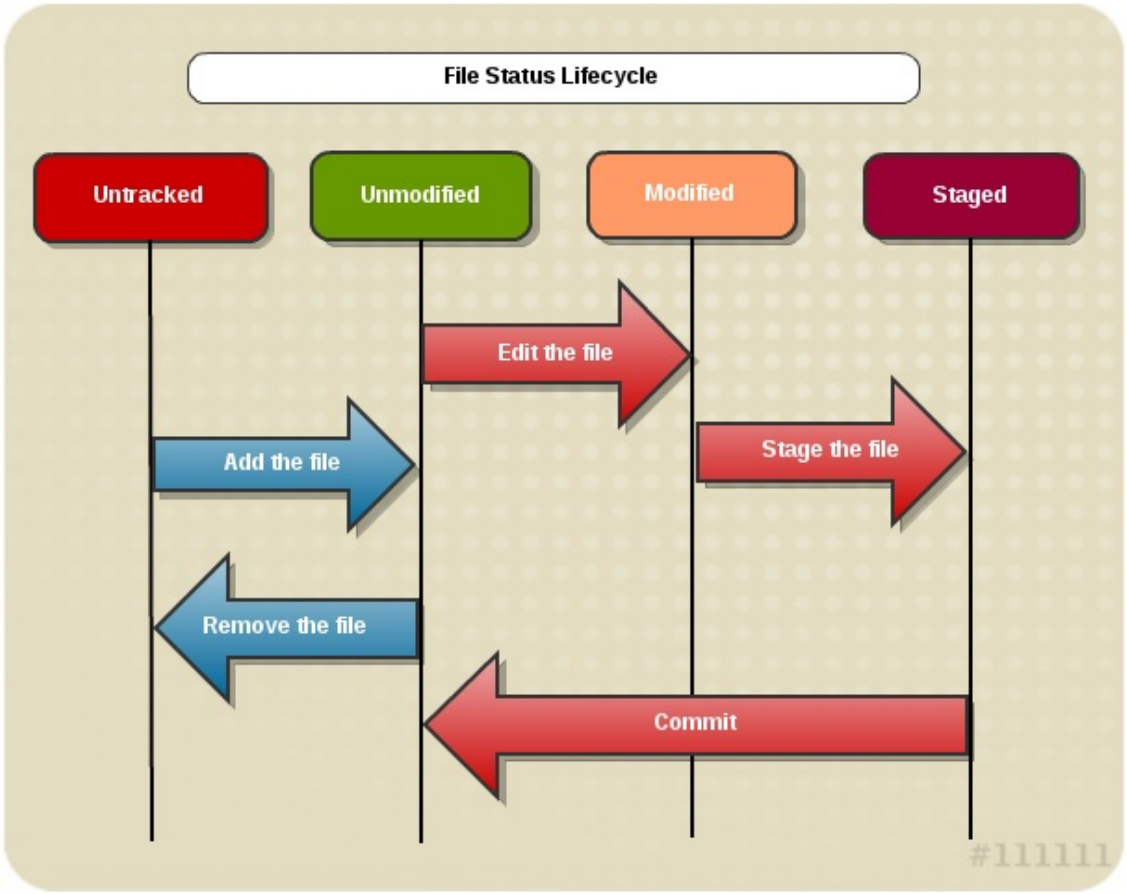


図2.2 Git ライフサイクル

上記のイメージは Git サイクルが機能する様子を示しています。以下のセクションで詳細を説明していきます。

2.3.4. 追跡されていない (Untracked) ファイル

追跡されていないファイルは、Git が認識しないものです。これは、ファイルが新たに作成された際か、新規プロジェクトの全ファイルで発生します。ファイルのステータスは、**`git status`** で表示されます。新規プロジェクトの場合は、ファイルは untracked ステータスになります。

```
$ git status
# On branch master
# Untracked files:
# (use "git add <file>..." to include in what will be committed)
#  filename
nothing added to commit but untracked files present (use "git add" to
track)
```

ステータスが分かりやすく示しているように、**`git add`** コマンドで Git にファイルを含めるように指示しない限り、ファイルは含まれません。

```
$ git add filename
```

`git add filename` コマンドは特定のファイルをまず未変更セクションに追加します。**`git add .`** を使って現行ディレクトリ (すべてのサブディレクトリを含む) 内の全ファイルを追加します。また、例として **`git add *.[ch]`** を使うと、現行ディレクトリ内のすべての **`.c`** および **`.h`** ファイルを追加します。

2.3.5. 未変更 (Unmodified) ファイル

未変更ステータスは、変更されていないファイルが保存されている状態です。Git はこれを認識しており、追跡することで、編集がなされると変更済みステータスに移動されます。また、コミット後にはファイルはこのステータスに戻されます。

また、この状態からファイルを削除して、Git による追跡を中止することも可能です。これは、ローカルでも削除されます。以下を実行します。

```
$ git rm filename
rm 'filename'
$ git status
# On branch master
#
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
#  deleted: filename
#
```

重要

ファイルが未変更の場合、**git rm *filename*** はファイル全体を削除します。**git rm *filename*** が診断してファイルを削除しないのは、ファイルが変更をコミットしていない場合のみです。変更がコミットされていないのに削除する場合は、 **--force** または **-f** オプションを使います。

ファイルをローカルで削除せずに Git による追跡を中止するには、 **--cached** オプションを使用し、その後に削除をコミットします。

```
$ git rm --cached filename
$ git commit -m'remove file message'
```

2.3.6. 変更済みステータス

ローカルシステム上のコピーが編集可能であるとします。変更が加えられると、Git はこのファイルを変更済みと認識し、変更済みステータスに移動します。**git status** の実行でこれが表示されます。

```
$ git status
# On branch master
# Changed but not updated:
# (use "git add <file>..." to update what will be committed)
#
#  modified: filename
#
```

ファイルは変更されましたが、まだコミットされていません (いずれにしても、さらに変更が加えられる可能性があり、コミットする準備が整っていない可能性があります)。出力が分かりやすく示しているように、**git add *filename*** コマンドを再度実行すると、変更済みファイルがステージング完了 (staged) ステータスにプッシュされ、コミットの準備が整います。

```
$ git add filename
$ git status
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
#  new file: filename
#
```

ステージング完了ファイルがコミット前に最後の修正を必要とすると、このファイルはステージング完了ステータスと変更済みステータスの両方に表示されるので、このプロセスはやや複雑になる場合があります。これが発生した場合は、ステータスは以下のようになります。

```
$ git status
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
#  modified: filename1
#
# Changed but not updated:
# (use "git add <file>..." to unstage)
#
#  modified: filename1
#
```

Git のスナップショットが目立つのはこういう場面です。コミットの準備ができたファイルのスナップショットがある一方で、変更済みステータスにも別のスナップショットがあります。コミットが実行されれると、ステージング完了ステータスのスナップショットのみがコミットされ、修正されたバージョンはコミットされません。**`git add`** を再度実行するとこれが解消され、ファイルの変更済みのスナップショットがステージング完了ステータスのスナップショットとマージされ、新たな変更をコミットする準備が整います。

2.3.7. ステージング完了 (staged) ファイル

ステージング完了ステータスは、コミットの準備ができたすべてのファイルのスナップショットがある状態です。このステータスにあるファイルはすべて、コミットコマンドでコミットされます。

2.3.7.1. 変更の表示

ステージング完了ステータスのスナップショットをコミットする前に、加えられた変更をチェックしてこれらが受け入れ可能なものかを確認するとよいでしょう。**`git diff`** コマンドはこういう場合に使います。

```
$ git diff
diff --git a/filename b/filename
index 3cb747f..da65585 100644
--- a/filename
+++ b/filename
@@ -36,6 +36,10 @@ def main
  @commit.parents[0].parents[0].parents[0]
 end

+ some code
```

```
+  some more code
+ a comment
+ another change
- a mistake
```

上記のように **git diff** コマンドをパラメーターなしで実行すると、作業ディレクトリをステージング完了ステータスにあるものと比較し、まだコミットされていない変更を表示します。

また、**--cached** オプションを使うと、ステージング完了ステータスの変更と最後にコミットされた変更を比較することもできます。

```
$ git diff --cached
diff --git a/filename b/filename
new file mode 100644
index 0000000..03902a1
-- /dev/null
+++ b/filename
@@ -0,0 +1,5 @@
+file
+ by name1, name2
+ http://path/to/file
+
+ added information to file
```

注記

Git のバージョン 1.6.1 およびそれ以降では、**--cached** の代わりに **--staged** オプションを使うことができます。

2.3.7.2. 変更のコミット

ステージング完了ファイルが正しいことをチェックして準備が整えば、変更をコミットします。

```
$ git commit
```

上記のコマンドで初期設定の際に選択されたエディターが起動するか、選択されていなければデフォルトの Vim が起動します。

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
#  new file: filename2
#  modified: filename1
~
~
~
".git/COMMIT_EDITMSG" 10L, 283C
```

上記のように、エディターには最新のステータスレポートも含まれますが、ハッシュ (#) のために実際のコミットログでは見えません。コミットされる内容に関する情報を取得するには、**git commit** で **-v** オプションを指定します。コミットメッセージはここに記入するか、**-m** オプションでコマンドラインに記入することができます。

```
$ git commit -m "commit message"
[master]: created 4156dc4f: "commit message"
2 files changed, 3 insertions(+), 1 deletions (-)
create mode 100644 filename
```

コミットメッセージは以下の情報を提供します。コミット先のブランチ (このケースではマスター)、コミットにある SHA-1 チェックサム (4156dc4f)、変更されたファイル数、ファイル内の変更内容についての統計情報。

注記

ステージング完了エリアは便利ですが、これをスキップすることは可能です。ただし、コミットの準備ができていないものをコミットしてしまうリスクもあります。これは、**git commit** で **-a** オプションを指定するとできます。

```
$ git status
# On branch master
#
# Changed but not updated:
#
# modified: filename
#
$ git commit -a -m 'commit message'
[master 16e15c7] commit message
1 files changed, 5 insertions(+), 2 deletions(-)
```

2.3.8. リモートのリポジトリ

プロジェクトを外部と共有するには、ネットワーク上もしくはインターネット上にあるリモートのリポジトリにプロジェクトをプッシュします。リモートのディレクトリにプッシュできるようにするには、まずディレクトリを作成する必要があります。これには、**git add [*shortname*] [*URL*]** を実行します。

```
$ git remote add shortname git://path/to/new/repo
```

shortname は、リモートリポジトリ照会の際に URL の代わりに使うことができます。

これでリポジトリが追加されたので、リモートリポジトリのリストを印刷することができます。**git remote** コマンドで **-v** オプションを指定すると関連する URL と希望する場合は shortname も表示されます。新規リモートリポジトリが追加されていない場合でも、プロジェクトがクローンされていれば、少なくともクローン元のリポジトリが一覧表示されます。

```
$ git remote -v
repo-name git://path/to/new/remote-repository
origin git://path/to/original/remote-repository
```

これで十分な情報が得られない場合は、**git show *[remote-repository]*** を実行すると、URL と Gi がこの特定のリポジトリで追跡しているブランチが一覧表示されます。

リモートリポジトリからデータを取得するには (例えば、プロジェクトで作業している誰かが新たな情報をプッシュした場合)、以下のコマンドを使用します。

```
$ git fetch [remote-name]
```

これで、ローカルコピーと異なるすべての情報がこの特定のリモートホストからプルされます。別の方法としては、**git pull** を実行すると、元のコピーのクローン元のリポジトリから同様の動作が実行されます。

より幅広い対象者とプロジェクトを共有するには、リモートリポジトリにプッシュすることが必要です。

```
$ git push remote-repository branch-name
```

このコマンドでは指定されたブランチが指定されたリポジトリにプッシュされますが、これはユーザーに書き込みアクセスがある場合のみです。競合が発生する場合は、リモートリポジトリにプッシュする前に、新規作業をプルしてからローカルバージョンに組み込みます。

最後に、リポジトリの名前を変更するには、**git remote rename *original-name new-name*** を実行します。この場合、リモートのブランチ名も変更されることに注意してください。リポジトリの削除も同様に、**git remote rm *remote-name*** を実行します。

2.3.9. ログのコミット

リポジトリとある程度作業を行い、変更やコミットを何回かした後に、これらの変更のログを確認する必要があるかもしれません。これは **git log** コマンドで行います。このコマンドが引数なしで実行されると、各コミットが最新のものから順に一覧表示され、コミットの日時、SHA-1 チェックサム、作成者の名前および E メール、コミットメッセージが示されます。

これらのログには差分レポートを含めることが可能で、また特定のエントリ数で出力を制限することもできます。例えば、**git log -p -*2*** を実行すると、直近のエントリ 2 つの差分レポートと通常のログ情報が一覧表示されます。

各コミットエントリの最後に統計情報を含めるには、**git log --stat** を使用します。このコマンドでコミットメッセージの後に、変更済みファイルのリスト、変更されたファイル数、追加および削除された行数、この情報のサマリーが含まれるようになります。

これらの便利なオプションの他に、このコマンドで指定できる多くのオプションがあります。

--shortstat

これは **--stat** オプションに似ていますが、表示されるのはコミットにおける変更、挿入、削除のみです。

--name-only

コミット情報の後に、変更済みのファイルのみを一覧表示します。

--name-status

変更、挿入、削除とともに変更済みファイルが一覧表示されます。

--abbrev-commit

SHA-1 チェックサムの全 40 文字ではなく、最初の数文字のみが表示されます。

--relative-date

現在の日付に対して相対的な日付を表示します。例えば、**Tue July 10:53:11 2011 -0700** ではなく、**2 weeks ago** (2 週間前) という表示をします。

--graph Display

ログ出力のそばにブランチの ASCII グラフとマージ履歴を表示します。

--since=*[date]*

指定された日付以降のログを表示します。つまり、その日付以降、すべてのものです。日付は特定の日付 (例: 2010-08-23) や相対的日付 (例: 1 年と 3 日 4 分前) のように異なる形式で入力可能です。

--until=*[date]*

--since オプションと似ており、特定の日付までのログを表示します。つまり、その日付前までのすべてのものです。

--author *name*

作成者フィールドで指定された名前のコミットのみを一覧表示します。

--committer *name*

コミッターフィールドで指定された名前のコミットのみを一覧表示します。

2.3.10. 問題の修正

何かに失敗して削除や取り消しが必要になる場合もあります。このセクションでは、この様なエラーを修正する方法を説明します。

警告

ここは、Git でデータが永久に失われてしまう可能性のある数少いエリアの一つです。取り消しが実行され、それをすべきでなかったことがわかった場合、失われたデータを回復できない可能性が非常に高くなります。注意して実行してください。

エラーは、コミットを早くプッシュし過ぎたり、準備ができていないものをコミットしたり、コミットメッセージでミスを犯したりすることで発生する場合が多くあります。このような場合は、**--amend** オプションを指定して直近のコミットの上にコミットすることで修復可能です。

```
$ git commit --amend
```

ステージング完了ステータスのファイルが最後のコミットと異なる場合、コミットは元のファイルを無効にする以外は通常通りに実行されます。ファイルが同じ場合は、再度コミットメッセージを変更して以前のコミットを無効にするオプションが提示されます。

また、ステージング完了ファイルをステージング前に戻すこともできます。これは、2 つ以上の別個のコミットを用意し、**git add *** が使われた場合に必要になることもあります。**git status** コマンドは、これを行う方法についてのヒントを提示します。

```
# (use "git reset HEAD <file>..." to unstage)
```

このアドバイスにしたがうには、**`git reset HEAD` *`filename`*** コマンドを使用します。すると、ファイルはステージング完了ステータスではなく変更済みステータスに戻ります。

ファイルを直近のコミット状態に戻すには、ステージング完了ステータス前で **`git status`** コマンドが役に立ちます。

```
# (use "git checkout -- <file>..." to discard changes in working
directory)
```

`git checkout --` *`filename`* コマンドを使ってこれらの指示に従うと、ファイルが元の状態に戻ります。ただし、前述の警告の繰り返しになりますが、この手順を行うと **データが失われます**。これを行う際には、対象のファイルのバージョンが必要ないことを必ず確認してください。

2.3.11. Git のドキュメンテーション

Git のメインの man ページは、**`man git`** で表示できます。ここでは、**`gittutorial(7)`**、**`Everyday Git[1]`**、**`gitglossary(7)`** などの他の man ページへのコマンドも紹介されています。

Git の公式ホームページは http://git-scm.com/ で、多くのドキュメンテーションをダウンロードできます。

以下の Web サイトでは、Git に関する詳細情報が提供されています。

- 基本的および高度な Git の指示に関する便利なブックがあります: http://progit.org/book/
- Red Hat Magazine の次の記事です、『Shipping quality code with Git』: http://magazine.redhat.com/2008/05/02/shipping-quality-code-with-git/
- GNU Smalltalk Web サイトの次の記事です、『Using Git without feeling stupid - part 1』: http://smalltalk.gnu.org/blog/bonzinip/using-git-without-feeling-stupid-part-1

  GNU Smalltalk Web サイトの次の記事です、『Using Git without feeling stupid - part 2』: http://smalltalk.gnu.org/blog/bonzinip/using-git-without-feeling-stupid-part-2
- パッチの準備および E メール送信方法についての説明: http://logcheck.org/git.html
- Git のコマンド早見表: http://cheat.errtheblog.com/s/git.
- Tim Waugh 氏によるブログ 『How I use Git』: http://cyberelk.net/tim/2009/02/04/how-i-use-git/
- ブログ記事 『Handling and avoiding conflicts in Git』 (Git での競合の処理および回避法) : http://weblog.masukomi.org/2008/07/12/handling-and-avoiding-conflicts-in-git
- 『Branching and merging with Git』 (Git でのブランチングおよびマージ) : http://lwn.net/Articles/210045/
- 『Getting the most out of Git in GNOME』 (GNOME で Git を最大限活用する方法) : http://live.gnome.org/Git/Developers
- 『Integrating vim with Git』: http://vim.runpaint.org/extending/integrating-vim-with-git/
- 『Git Reference』: http://gitref.org/
- 『A successful Git branching model』: http://nvie.com/posts/a-successful-git-branching-model/
- 概説ガイド、『Git for the lazy』: http://www.spheredev.org/wiki/Git_for_the_lazy
- 『Git tricks, tips, and workflows』: http://nuclearsquid.com/writings/git-tricks-tips-workflows/

第3章 ライブラリおよびランタイムのサポート

Red Hat Enterprise Linux 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 7 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 7、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 7 で開発されたソフトウェアの主要依存関係になります。

表3.1 言語およびランタイムライブラリのバージョン

パッケージ名	Red Hat Enterprise 7	Red Hat Enterprise 6	Red Hat Enterprise 5	Red Hat Enterprise 4
glibc	2.12	2.12	2.5	2.3
libstdc++	4.8	4.4	4.1	3.4
boost	1.53	1.41	1.33	1.32
java	1.7	1.5 (IBM)、1.6 (IBM、OpenJDK、Oracle Java)	1.4、1.5、1.6	1.4
python	2.7	2.6	2.4	2.3
php	5.4	5.3	5.1	4.3
ruby	2.0	1.8	1.8	1.8
httpd	2.4	2.2	2.2	2.0
postgresql	9.2	8.4	8.1	7.4
mysql	5.4	5.1	5.0	4.1
nss	3.15	3.12	3.12	3.12
openssl	1.0.1e	1.0.0	0.9.8e	0.9.7a
libX11	1.6	1.3	1.0	
firefox	24.4	3.6	3.6	3.6
kdebase	4.10	4.3	3.5	3.3
gtk2	2.24	2.18	2.10	2.04

注記

compat-glibc RPM は Red Hat Enterprise Linux 7 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

互換性は、コンピューターの稼働環境の異なるインスタンスにおけるバイナリオブジェクトとソースコードの移植性を特定します。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。

バイナリの互換性

バイナリの互換性は、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、Red Hat Enterprise Linux のサポート対象リリースを参照してください。こちらになります、https://access.redhat.com/support/policy/updates/errata/。また、Red Hat Enterprise Linux の互換性ポリシー全般は、https://access.redhat.com/site/articles/119073 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**

このライブラリセットにより、アプリケーションが上記のライブラリのみを使用するという条件で、開発者は Red Hat Enterprise Linux 7 で Red Hat Enterprise Linux 6 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
```

重要

ソースの重複または **memcpy** および他の関数の対象メモリーの場所に関して ISO 違反となるアプリケーションは、失敗する可能性が高くなります。

3.3.2. GNU C++ 標準ライブラリ

libstdc++ パッケージには GNU C++ 標準ライブラリが含まれています。これは ISO 14882 標準 C++ ライブラリ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、共有ライブラリファイルのみ)。C++ 開発で使用可能なライブラリ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 (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おり、このため **libstdc++** に互換性ライブラリは必要ありません。しかし、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++ 標準ライブラリ更新

Red Hat Enterprise Linux 6 バージョンの GNU C++ 標準ライブラリでは、Red Hat Enterprise Linux 5 バージョンと比較して以下の改善がなされています。

- 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 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++ 標準ライブラリドキュメンテーション

ライブラリコンポーネントの **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。例えば、**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++ ソースライブラリがあります。これらのライブラリは、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、共有ライブラリファイルのみ)。C++ 開発で使用可能なライブラリ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 機能はパッケージ化で多くの改善があり、また新機能もあります。

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パッケージが可能になっています。

さらに、全ライブラリ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 ドキュメンテーション

boost-doc パッケージは、以下のディレクトリで 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 一般公衆利用許諾契約書 (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

Qt の Red Hat Enterprise Linux 6 バージョンでの改善点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 ライブラリドキュメンテーション

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

Red Hat Enterprise Linux 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、その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 を使用するユーザーインターフェイス設計用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、http://techbase.kde.org/Development/Architecture/KDE4/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 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 ドキュメンテーション

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 ドキュメンテーションを提供します。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** に代わって、**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 のどのバージョンと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、Red 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

	Red Hat Enterprise Linux バージョン		
GNOME の電源管理のデスクトップコンポーネント	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

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
glib2	2.4.7	2.12.3	2.22.5
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 (もしくは Red Hat Enterprise Linux 5 と Red Hat Enterprise Linux 6) での *glib* の違いは以下の通りです。

- 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のサポート
- GIO (gnome-vfs の代わりとなる VFS ライブラリ)
- 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* のドキュメンテーションは、*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+* の違いは以下の通りです。

- 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+* の違いは以下の通りです。

- 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+* のドキュメンテーションは、*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 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 (例えば、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生成された バイナリ互換性の保証は、Red Hat Enterprise Linux 6 では NSS で

保持されます。この保証は、NSS 3.12 はすべての古い NSS 3.x の共有ライブラリ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 共有ライブラリの今後のバージョンと互換性を保ちます。

Red Hat Enterprise Linux 5 および 4 は、Red Hat Enterprise Linux 6 と同様のバージョンの NSS で稼働します。このため、ABI や API の変更はありません。しかし、Red Hat Enterprise Linux 6 では内部の暗号化モジュールは NSS 3.12 からのものであるのに対し、Red Hat Enterprise Linux 5 と 4 では未だに NSS 3.15 からの古いものを使用しているため、違いは存在します。つまり、共有データベースなどの NSS 3.12 で導入された新たな機能は、Red Hat Enterprise Linux 6 のバージョンの NSS で利用可能となります。

3.3.7.2. NSS 共有データベース のドキュメンテーション

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います。『こちら』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** ドキュメンテーション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** files) の両方が Python 2.4 (Red Hat Enterprise Linux 5 で使用)、Python 2.6 (Red Hat Enterprise Linux 6 で使用)、および Python 2.7 (Red Hat Enterprise Linux 7 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ユーザーが再構築する必要があります。

3.3.8.1. Python 更新

Red Hat Enterprise Linux 7 は Python 2.7 と出荷されています。この変更に関する情報は、以下のプロジェクトリソースを参照してください。

- What's New in Python 2.7: http://docs.python.org/dev/whatsnew/2.7.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。

重要

Python は、C 拡張モジュールで使用するさまざまな API を提供します。この 1 つであった PyCObject は、Python 2.7 では非推奨となっています。デフォルトでは非推奨の警告が無視されるため、通常は問題が発生しません。

ただし、標準の警告設定が上書きされると、PyCObject を使用するモジュールでインポートが成功するとみなされて、問題が発生する場合があります。特に、警告が「エラー」に設定されている場合は、そのようなモジュールをインポートする際に NULL ポインターで読み込むため、Python インタプリターが中止したり、さらにはセグメンテーション違反となる可能性もあります。

errors-for-warnings を有効にしてこれらのモジュールを使用するには、上書きを追加して、例外を発生する代わりに **PendiingDeprecationWarning** をログ記録するようにします。

```
>>> import warnings
>>> warnings.simplefilter('error')
>>> warnings.simplefilter('default', PendingDeprecationWarning)
```

3.3.8.2. Python デバッグビルド

Red Hat Enterprise Linux 7 では、*python-debug* パッケージに Python インタプリターのデバッグビルドが含まれています。

デバッグインタプリター (**/usr/bin/python-debug** にある) は 最適化インタプリター (**/usr/bin/python** にある) のおよそ半分の速度で実行し、再構築する拡張モデルを必要としますが、Python C 拡張モジュールの書き込みおよびデバッグ時にはまだ使用されます。デバッグビルド内では、最適化レベルは下げられており、デバッガー内でのコードのステップが容易になっています。

デバッグビルドは、以下の新たなデバッグ設定で構成されています。

--with-pydebug

sys.gettotalrefcount() や **sys.getobjects()** などのように、sys に各種の便利なメソッドを追加します。

--with-count-allocs

COUNT_ALLOCS 設定を有効にして **sys.getcounts()** メソッドを追加し、すべての型の情報を提供します。プロセスが存在するときは、デフォルトのアップストリーム動作は常にこの情報を **stdout** にダンプします。これはダウンストリームに更新されるので、情報は **PYTHONDUMPCOUNTS** が環境で設定されていれば、終了時にのみダンプされます。

--with-call-profile

CALL_PROFILE 設定を有効にします。これは実行される関数呼び出し数と、インタプリターによるこれらの呼び出しの処理方法をカウントします。

--with-tsc (x86_64 および ppc64 のみ)

sys.settscdump() メソッドを追加し、インタプリターの非常に低レベルのプロファイリンクを追加します。

デバッグビルドは通常の最適化ビルドと同様のバイトコードファイルを使用しますが、拡張モジュール (**.so** files) は互換性がありません。これは、Python オブジェクトのメモリー内レイアウトが外部インストルメントを理由に異なるためです。最適化拡張モデルの ***foo*.so** が与えられると、デバッグビルドは ***foo*_d.so** を探すように更新されます。

デバッグビルドおよびその設定についての詳細は、アップストリームの注意書き http://svn.python.org/projects/python/trunk/Misc/SpecialBuilds.txt を参照してください。

3.3.8.3. Python ドキュメンテーション

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。例えば、**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

Red Hat Enterprise Linux 7 は最新バージョンの JDK と出荷されるように常に更新されています。

java-*version_number*-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-*version_number*-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 の機能

Red Hat Enterprise Linux 7 では、Java に以下のような多くの新機能が備わっています。

動的型付け言語のサポート (InvokeDynamic)

Hotspot、Open JDK の Java Virtual Machine (JVM) に機能強化がなされました。これは、静的型付け言語および Java そのものと比較して最小限のパフォーマンスコストで動的型付け言語をサポートするように設計されています。具体的には、invokedynamic 命令が Java バイトコード仕様に追加され、JVM で実装されています。

小規模の言語機能強化 (Project Coin)

プログラマーに利便性とより洗練されたコードを提供し、いくつかの一般的なプログラミングエラーを削減する、多くの Java 言語レベルの改善。

switch の文字列

以前のバージョンの Java では、switch ステートメントは byte、short、char、および int のプリミティブ型とそれらに対応するオブジェクト型、さらには列挙型を許可していました。Java 7 の switch ステートメントでは、文字列値も使用できます。

バイナリ整数リテラルおよび数値リテラルにおけるアンダースコア

プログラマーはバイナリ形式で整数リテラルを表示するか、アンダースコアで数値リテラルの桁のグループを分けることで、コードの読みやすさが改善されています。

マルチ catch

Java の catch 構文が改善され、単一の catch 節に 1 つ以上の例外タイプをキャッチして、重複コードを減らすことができます。

より正確な再スロー

Java 7 コンパイラーが改善され、例外をキャッチ、再スローするメソッドがいくつかの状況ではメソッド宣言のスロー節でより正確にすることができます。

ジェネリックインスタンス作成の型の推論における改善 (diamond)

この構文上の改善により、プログラマーはジェネリック型の変数をインストルメント化する際に、完全なジェネリック型 (たとえば、<ClassName>) ではなく、ダイヤモンド演算子 (つまり、<>) を使用することができます。型は、代わりにその変数の宣言から推定されます。

Try-with-resources ステートメント

これは、ストリームやファイルなどの closeable リソースと使用するための try ステートメントの新しい形式です。この機能を使うことで、プログラマーはこれらのリソースを明示的に閉じる必要がなくなりました。

簡素化された varags メソッドの起動

これまでは、non-reifiable 引数で vararg メソッドを呼び出す際に、コンパイラー警告が発行さ

れていました。これが削除され、vararg メソッドの宣言時に non-reifiable 引数を受け入れる警告で置き換えられます。注釈を使ってこの警告を止めることもできますが、その場合は引数の正確性については開発者が責任を持ちます。これは主に、ジェネリック引数を受け付ける vararg メソッドに適用されます。

Concurrency およびコレクションの更新

除算用の fork/join フレームワークや conquer 型アルゴリズムを含む数多くの新しいクラスやインターフェースが **java.util.concurrency** パッケージに追加されました。これらは、マルチスレッドプログラムのパフォーマンスや正確性の改善に役立ちます。

Java プラットフォーム用の新しい I/O API

これには、開発者による正常な失敗処理を容易にする一方でプラットフォームにまたがる互換性を改善する新たなファイルシステム API が含まれます。これは、**java.nio.channels** パッケージのソケット/チャンネル API を改善し、**java.net** パッケージへの非直感的な依存関係を取り除きます。また、新たな非同期 I/O API も提供します。

swing 用の Nimbus のルックアンドフィール

com.sun.java.swing パッケージの名前空間の下で Java 6 で非公式に導入された Nimbus には、swing 用にベクトルグラフィックスベースのルックアンドフィールがありました。Java 7 では、公式 API となっており、**javax.swing** パッケージに移動しました。

3.3.9.2. Java ドキュメンテーション

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 ユーティリティについての詳細情報のために他の Java ドキュメンテーション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 (例えば、**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、ライブラリの拡張機能か既存ライブラリへの 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-irb
- ruby-libguestfs
- ruby-libs
- ruby-qpid
- ruby-rdoc
- ruby-ri

- ruby-tcltk
- rubygems
- rubygem-bigdecimal
- rubygem-devel
- rubygem-io-console
- rubygem-json
- rubygem-minitest
- rubygem-rake

注記

アプリケーションの依存関係の管理に Bundler を使用する場合は、**rubygem-bundler** パッケージが提供するBundler を常に使用してください。アップストリームパッケージは、Red Hat Enterprise Linux 7 で使用する RubyGems レイアウトと互換性がありません。

3.3.10.1. Ruby の更新

Red Hat Enterprise Linux 7 の Ruby 言語の更新についての情報は、以下の資料を参照してください。

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

Ruby ではファイルシステムレイアウトにおいて大幅は変更がされており、FHS との適合性が高まっています。バイナリライブラリーと Gems の拡張は **/usr/lib** (64 ビットシステムの場合は **/usr/lib64**) に置かれており、純粋な Ruby ライブラリーおよび Gems は **/usr/share** に置かれています。Gems は選択されたインストール方法によって、以下の 3 つの場所に置かれます。

- **/usr**
- **/usr/local**
- **~/.gem**

3.3.10.2. Ruby ドキュメンテーション

Ruby についての詳細情報は、**man ruby** を参照してください。また、**ri** コマンドの使用も可能です。これは、Ruby API 参照のフロントエンドです。gem ドキュメンテーションについては、**gem server** コマンドを使用してください。これは、システム上にインストールされている gems についての HTML マニュアルとリファレンスをブラウザで利用可能にします。

注記

ri および **gem server** コマンドを使って *-doc* サブパッケージをインストールする必要がある場合があります。

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 ドキュメンテーションが含まれています。gems のオンラインドキュメンテーションは、http://rdoc.info でご覧になれます。

ri コマンドのドキュメンテーションは、**/usr/share/ri/system** にあります。

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、いくつかの Perl コアモジュール、Perl Language Interpreter (Perl 言語インタープリター)、および **perldoc** ツールを提供します。Red Hat Enterprise Linux 7 では、*perl-5.16* が同梱されています。コアモジュールのすべてをインストールするには、**yum install perl-core** コマンドを使用してください。

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、外部ライブラリバインディングを提供します。

RPM パッケージに追加の perl モジュールが含まれる場合があります。公開使用の各モジュールはパッケージが提供し、*perl(The::Module)* という形式になっています。この式は**yum** に渡され、適切なパッケージのインストールが可能になります。

例3.1 Perl モジュールのインストール

```
# yum install 'perl(LWP::UserAgent)'
```

これで RPM パッケージ *perl-libwww-perl* がインストールされます。これには**LWP::UserAgent** モジュールが含まれており、プログラマーは **use LWP::UserAgent;** コマンドが使えるようになります。

3.3.11.1. Perl 更新

Red Hat Enterprise Linux 7 では perl 5.16 が同梱されており、Red Hat Enterprise Linux 6 で同梱の 5.10 バージョンから以下のような多くの変更点があります。

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 5.16 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* が廃止予定で、代わりに *CPAN::Meta::Requirements* が使用されます。
- 多くの perl4 ライブラリを削除:
  - *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.pl*
- *lib/finddepth.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。

Official 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。また、**vendor_perl** サブディレクトリーにも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** ディレクトリーが存在し、現行ユーザーによる書き込みが可能な場合にこれらのディレクトリーにインストールされます。

これらのディレクトリーが存在しない場合、**cpan** ツールは別の解決法を提案します。

```
Warning: You do not have write permission for Perl library
directories.

To install modules, you need to configure a local Perl library
directory or escalate your privileges. CPAN can help you by
bootstrapping the local::lib module or by configuring itself to
```

```
use 'sudo' (if available). You may also resolve this problem
manually if you need to customize your setup.

What approach do you want? (Choose 'local::lib', 'sudo' or
'manual')
[local::lib]
```

たとえば、「manual」を選択した場合、CPAN からモジュールをインストールする前に、ディレクトリーが存在し、書き込み可能であることをユーザーが確認しているとみなされます。

サードパーティーおよびカスタムのモジュールパッケージ

これらのモジュールは **/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。ファイル名が Red Hat Enterprise Linux のパッケージと競合する場合は、ファイル名を変更するか、Red Hat Enterprise Linux のパッケージを提供されるパッケージで適正に置き換えてください。

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。

新たな Perl モジュール検索パスが必要な場合、**/usr/local/share/perl5/sitecustomize.pl** スクリプトをシステム全体の修正に使うか (**perlrun(1)** man を参照)、または *perl-homedir* パッケージをユーザー固有の修正に使うことができます (*perl-homedir* パッケージの説明を参照)。

3.3.11.3. Perl のドキュメンテーション

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テーションを提供します。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。例えば、**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 関数の詳細については、**perldoc -f *function_name*** を使用してください。たとえば、perldoc -f split は split 関数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 ドキュメンテーションは perldoc.perl.org で提供されています。

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。

3.3.12. libStorageMgmt プラグイン

Red Hat Enterprise Linux 7 は **libStorageMgmt** と呼ばれる新たなライブラリーと出荷されています。これは、安定しかつ一貫性のある API を提供する個別の Application Programming Interface (API) のストレージアレイです。これにより開発者はプログラムで異なるストレージアレイを管理でき、提供されているハードウェア高速機能を活用できます。

libStorageMgmt ライブラリーの詳細情報については、Red Hat の『Storage Administration Guide』を参照してください。本セクションでは、ライブラリー用のプラグインの作成方法を詳述します。

libStorageMgmt ライブラリーでは、プラグインはやや異なる機能の仕方をします。このプラグインは、クライアントとプラグインの間のプロセス間通信 (IPC) でスタンドアロンの実行可能ファイルとして自身のアドレス領域内で実行します。クライアントアプリケーションもしくは **libStorageMgmt** コマンドライン (**lsmcli**) がライブラリーを使用すると、以下が発生します。

1. ライブラリーは uniform resource identifier (URI) を使ってどのプラグインが指定されているかを解析します。たとえば、**LSMCLI_URI=sim://** シミュレータープラグインを参照します。
2. ライブラリーはプラグイン名 **sim** を使って、ソケットディレクトリー内の unix ドメインソケットを探します。デフォルトのディレクトリーは **/var/run/lsm/ipc** ですが、これはランタイム時に **LSM_UDS_PATH** 環境変数を指定することで変更できます。

3. クライアントライブラリーが unix ドメインソケットを開き、**lsmd** デーモンがクライアントからの接続を受け入れるようにします。デーモンがプラグインを分岐、実行し、コマンドラインのソケット記述子をプラグインに渡します。これでクライアントプロセスがプラグインに直接接続されたことになります。

4. **lsmd** はすでにパスには存在せず、スリープに戻ってソケットを開く別のプロセスを待機します。

この異なる設計には、以下のような多くの利点があります。

- デーモンが停止したり、強制終了した場合、既存のクライアントプラグインセッションはアクティブのままになります。
- プラグインがクラッシュした場合、クライアントプロセスは操作可能なままです。
- デーモンが IPC プロトコルを知る必要がなく、シンプルさを保てます。
- ベンダーが必要とする場合、プラグインはクローズソースとすることができます。

3.3.12.1. libStorageMgmt ライブラリー用のプラグイン作成

libStorageMgmt ライブラリーには C および Python 用のプラグイン API があります。ソケットおよびテキストをサポートする言語はすべてプラグインの作成に使用できますが、ライブラリーはこの複雑性を隠す抽象化を提供します。

以下では、プログラム言語に関係なくプラグインの設計に関する一般的なガイドラインを示しています。

スレッドまたはマルチプロセス

ライブラリーはロックを提供しません。また、グローバル状態を保持することもしません。このため、クライアントが各スレッドやプロセス用に使用中の個別のプラグインインスタンスを持つことは有効です。プラグインは、プラグインの複数インスタンスが異なるアレイに同時に稼働可能であること、また場合によっては稼働するであろうことを予測することができます。ライブラリーが長時間オペレーションのメカニズムを提供するので、同一アレイ用の複数プラグインインスタンスは不要です。

プラグインは root 以外の権限で実行

ローカルの不正使用の可能性を減らすため、プラグインには低い権限が付けられています。プラグインの作成と設計の際には、この点を考慮する必要があります。

プラグインのライフタイム

クライアント API は、各プラグインインスタンスで開閉されるハンドルを提供します。この間、プラグインは、正常な操作を提供するために必要なデータを自由にキャッシュすることができます。**lsmcli** ツールを使用の際は、ライフタイムは 1 つのコマンドのみになります。

ログ機能

プラグインは、エラーを **syslog** にログ記録します。ライブラリーにはこの動作を促進するためのヘルパー機能があります。

エラー

ライブラリーは、言語に依存しない状態を継続するために、明確に定義されたエラーコードを使用します。エラー発生時にプラグインもしくはアレイそのものからテキストのエラーメッセージおよびオプションでデバッグデータを提供するために、追加のエラーデータは取得できます。エラー発生後および別のコマンド発行前にこの新たな情報を取得するのは、ライブラリー呼び出し元の責任です。新たなエラーデータが存在し

てかつ他の関数が呼び出された場合、新たなエラーの情報は失われます。C は例外をサポートしません。例外をサポートする言語については、エラーコードおよび追加情報を含むカスタムの例外クラスが提供されます。

場所および命名

プラグインは **/usr/bin** ディレクトリー内に配置されます。名前は **_lsmplugin** にする必要があります。これは、デーモン開始時にディレクトリー内で列挙を繰り返すためです。

ジョブコントロール

タイムアウトを取得して設定する方法は、プラグインがアレイからの反応を待機する時間を指定するために使われています。操作がタイムアウト内に安全に完了できない場合は、コールがジョブ ID を返すことでクライアントが操作の状況を確認できます。ジョブ ID は自由形式の文字列で、プラグイン定義です。プラグインの実装は、プラグインの発動の間にこの文字列からの非同期操作についてのすべてを決定する必要があります。

プラグインを作成するには、以下のベース機能もしくは方法が、使用する言語に関わらずすべて必要になります。

- タイムアウトの取得および設定
- スタートアップおよびシャットダウン
- ジョブステータス
- job free
- ケイパビリティ
- プラグイン情報
- プール
- システム

メインの実行可能ファイルにフォーム ***name*_lsmplugin** があるようにするために、一意の名前を選択する必要もあります。

以下のセクションでは、python および C 用のプラグインの作成方法を詳述します。

3.3.12.1.1. Python でのプラグインの作成方法

最初に提供する機能レベルをサポートするインターフェースを実装します (**iplugin.py** を参照)。ほとんどのインターフェースは、IStorageAreaNetwork または INfs から継承するか、プラグインがブロックおよびネットワークファイルシステムをサポートする場合は、両方を継承します。

次に、プラグイン runner を呼び出し、run メソッドを処理、実行するためのクラス名およびコマンドライン引数を渡します。

```
#!/usr/bin/env python
import sys

from lsm.pluginrunner import PluginRunner
from lsm.simulator import StorageSimulator

if __name__ == '__main__':
    PluginRunner(StorageSimulator, sys.argv).run()
```

注記

開発中にデバッグを容易にするため、プラグインをコマンドライン上で直接呼び出すことが可能です。

3.3.12.1.2. C でのプラグインの作成方法

最初に必要なヘッダーファイル **#include <libstoragemgmt/libstoragemgmt_plug_interface.h>** をインクルードします。

次に、必要なものと合わせてサポートされるコールバック関数を実装します。

最後に、ロードおよびアンロード関数でコマンドラインカウントおよび引数をライブラリーに渡します。

```
#include <libstoragemgmt/libstoragemgmt_plug_interface.h>
#include <stdlib.h>
#include <stdint.h>

static char name[] = "Simple limited plug-in example";
static char version [] = "0.01";

struct plugin_data {
    uint32_t tmo;
    /* All your other variables as needed */
};

/* Create the functions you plan on implementing that
    match the callback signatures */
static int tmoSet(lsm_plugin_ptr c, uint32_t timeout, lsm_flag flags )
{
    int rc = LSM_ERR_OK;
    struct plugin_data *pd = (struct
plugin_data*)lsm_private_data_get(c);
    /* Do something with state to set timeout */
    pd->tmo = timeout;
    return rc;
}

static int tmoGet(lsm_plugin_ptr c, uint32_t *timeout, lsm_flag flags )
{
    int rc = LSM_ERR_OK;
    struct plugin_data *pd = (struct
plugin_data*)lsm_private_data_get(c);
    /* Do something with state to get timeout */
    *timeout = pd->tmo;
    return rc;
}

/* Setup the function addresses in the appropriate
    required callback structure */
static struct lsm_mgmt_ops_v1 mgmOps = {
    tmoSet,
    tmoGet,
    NULL,
```

```
    NULL,
    NULL,
    NULL,
    NULL
};

int load( lsm_plugin_ptr c, const char *uri, const char *password,
                        uint32_t timeout, lsm_flag flags )
{
    /* Do plug-in specific init. and setup callback structures */
    struct plugin_data *data = (struct plugin_data *)
                                malloc(sizeof(struct plugin_data));

    if (!data) {
        return LSM_ERR_NO_MEMORY;
    }

    /* Call back into the framework */
    int rc = lsm_register_plugin_v1( c, data, &mgmOps, NULL, NULL, NULL);
    return rc;
}

int unload( lsm_plugin_ptr c, lsm_flag flags)
{
    /* Get a handle to your private data and do clean-up */
    struct plugin_data *pd = (struct
plugin_data*)lsm_private_data_get(c);
    free(pd);
    return LSM_ERR_OK;
}

int main(int argc, char *argv[] )
{
    return lsm_plugin_init_v1(argc, argv, load, unload, name, version);
}
```

3.3.12.2. プラグイン作成方法に関する参照資料

- The libStorageMgmt Writing Plug-ins wiki
  https://sourceforge.net/p/libstoragemgmt/wiki/WritingPlugins/

[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 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を含む、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 (**libgcc**、**libstdc++**、**libgfortran**、**libgomp** など)、その他のユーティリティが含まれます。

Red Hat Enterprise Linux 7 は PPC64 アーキテクチャー上にあります。つまり、GCC はデフォルトで POWER7 用にチューニングされた POWER7 プラットフォーム用のコードを生成します。以下の表では、各プラットフォーム用のデフォルトのコードを示しています。

表4.1 i?86 のデフォルトコード

オプション	デフォルトコード
mtune	generic
march	x86-64

パッケージの構築に使用される **redhat-rpm-config** には、**-mfpmath=sse** オプションも含まれています。

表4.2 x86_64 のデフォルトコード

オプション	デフォルトコード
mtune	generic
march	x86-64

表4.3 s390 および s390x のデフォルトコード

オプション	デフォルトコード
mtune	zEC12
march	z196

表4.4 ppc、ppc64、および ppc64p7 のデフォルトコード

オプション	デフォルトコード
mtune	power7
mcpu	power7

上記の表は、32 ビットと 64 ビットの両方のコンパイル用です。

-march=CPU オプションは、他のターゲット用のコード生成に使用することができます。特定のチップ用にコードをチューニングするには、**-mtune=CPU** オプションを使用します。**-march** オプションは、**-mtune** オプションを含みます。**-march=native** オプションは、コンパイラーにコンパイルマシンのプロセッサータイプ用にコードを生成するよう指示します。

注記

-march オプションは、PPC64 アーキテクチャー上には存在しません。代わりに**-mcpu** オプションを使用してください。

4.1.1. GCC の変更

Red Hat Developer Toolset 2.0 は **GCC 4.8** とともに配布されています。これは、Red Hat Enterprise Linux のシステムバージョンおよび Red Hat Developer Toolset 1.1 に含まれているバージョンにおける多くのバグ修正と機能強化を提供するものです。以下では、今回のリリースにおける新機能および互換性の変更の完全なリストを示しています。

4.1.1.1. Red Hat Developer Toolset 1.1 以降の変更

以下の機能は、Red Hat Developer Toolset 1.1 に含まれる GCC のリリース以降に追加されたものです。

4.1.1.1.1. 警告

アグレッシブなループの最適化

GCC のループオプティマイザーは、ループの反復数の範囲をもたらすために言語の制約を使用するよう改善されました。この範囲は、ループ展開、ループピーリング、およびループ終了テストの最適化の目安として使用されています。

最適化では、たとえば符号付き整数のオーバーフローを引き起こしたり範囲外のアレイアクセスを発生させたりすることで未定義の動作をループコードが引き起こさないことを想定しています。たとえば、以下のコードの一部を考えてみます。

```
unsigned int foo()
{
    unsigned int data_data[128];

    for (int fd = 0; fd < 128; ++fd)
        data_data[fd] = fd * (0x02000001); // error

    return data_data[0];
}
```

fd 変数の値が 64 以上の場合、**fd * 0x02000001** オペレーションがオーバーフローします。これは、C と C++ の両方の符号付き整数で無効なものです。上記の例では、GCC が間違ったコードを生成するか、無限ループに入る可能性があります。

このエラーを修正するには、符号付きと符号なしのタイプ間でオーバーフローを避けるために変換する際に適切なキャストを使用します。例を示します。

```
data_data[fd] = (uint32_t) fd * (0x02000001U); // ok
```

必要であれば、この最適化は新しいコマンドオプションの **-fno-aggressive-loop-optimizations** を使ってオフにすることができます。

4.1.1.1.2. 全般的な改善点と変更点

新たな Local Register Allocator

GCC 4.8 には、新たな *Local Register Allocator* (LRA) 機能があります。これは、26 年前からあるリロードパスに代わるもので、生成コードの質を高めます。この新たな local register allocator ではデバッグがシンプルかつ容易になり、レジスタ割り付けが改善されます

AddressSanitizer

AddressSanitizer という高速メモリーエラー検出機能が追加されており、**-fsanitize=address** コマンドラインオプションを使用することでこれを有効にできます。これは、ヒープ上のオブジェクトへの use-after-free (メモリー解放後の使用) アクセスおよび範囲外のアクセスを検出するためにメモリーアクセス命令を増やします。

ThreadSanitizer

GCC 4.8 では、*ThreadSanitizer* という高速データレース検出機能が追加されています。この機能は、**-fsanitize=thread** オプションで有効になります。

超大型関数のコンパイル

GCC 最適化パスからは多くのスケーラビリティーに関する障害が取り除かれました。その結果、より少ないメモリー消費量で短時間に非常に大型の関数をコンパイルすることが可能になっています。

新たな -Og 最適化レベル

新たな全般的最適化レベルである **-Og** が導入されました。この最適化レベルは、高速コンパイルとすぐれたデバッグエクスペリエンスのニーズに対応し、妥当なレベルのランタイムパフォーマンスを提供します。全体としては、開発体験はデフォルトの最適化レベルである **-O0** よりも改善されるはずです。

キャレット診断メッセージ

GCC の診断メッセージはソースコードの行を表示しますが、問題が検出されてコラムを指すキャレットも表示するようになりました。例を示します。

```
fred.cc:4:15: fatal error: foo: No such file or directory
 #include <foo>
               ^
compilation terminated.
```

新たな -fira-hoist-pressure オプション

コマンドラインオプション **-fira-hoist-pressure** が新たに追加されました。このオプションは、register allocator を使っていつ式をループ外に移動したらよいかを判断する手助けをします。これはコンパイラーコードのサイズを縮小できますが、コンパイラーを遅らせることになります。このオプションはデフォルトで **-Os** で有効になります。

新たな -fopt-info オプション

コマンドラインオプション **-fopt-info** が新たに追加されました。このオプションは、特定の最適化パスの効果についての情報の印刷を制御します。以下の形式になります。

```
-fopt-info[-info][=file_name]
```

オプションの *info* の部分が印刷されるものを制御します。これを**optimized** に置き換えると最適化の際に情報が印刷され、**missed** に置き換えると最適化が行われない時に情報が印刷されます。**note** に置き換えるとより詳細な情報が印刷され、**optall** にするとすべてが印刷されます。

file_name は、情報の書き込み先とするファイル名に置き換えます。オプションのこの部分を省略すると、GCC は情報を標準のエラー出力ストリームに書き込みます。

たとえば、**-O2** オプションで有効にしたものの、**foo.c** という名前のファイルをコンパイルする際に効果のなかった最適化のリストを表示するには、以下を入力します。

```
gcc -O2 -fopt-info-missed foo.c
```

新たな -floop-nest-optimize オプション

コマンドラインオプション **-floop-nest-optimize** が新たに追加されました。このオプションは、実験段階にある ISL ベースのループネストオプティマイザーを有効にします。これは一般的なループネストオプティマイザーで、Pluto 最適化アルゴリズムに基づき、データローカリティおよび並列処理用に最適化されたループ構造を計算します。このオプティマイザーについての詳細は、http://pluto-compiler.sourceforge.net を参照してください。

ラベルのホットおよびコールド属性

ホットおよびコールドの関数属性がラベルにも適用できるようになりました。ホットラベルは、ラベルに続く実行パスが他の実行パスよりも可能性が高いことをコンパイラーに通知します。コールドラベルは、逆のことを意味します。これらの属性は、**__builtin_expect** がたとえば計算された **goto** や **asm goto** と使用できない場合に使用可能です。

4.1.1.1.3. デバッグ機能の強化

DWARF4

デバッグ情報生成時に、**DWARF4** がデフォルトのデバッグデータ形式として使用されるようになりました。この新たなデバッグ表示法を最大限活用するには、本リリースに含まれている最新バージョンの **Valgrind**、**elfutils**、および **GDB** を使用します。

新たな -gsplit-dwarf オプション

コマンドラインオプション **-gsplit-dwarf** が新たに追加されました。このオプションは、できるだけ多くの DWARF デバッグ情報を **.dwo** ファイル拡張子の付いた別の出力ファイルに分けるようコンパイラードライバーに指示します。そうすることで、ビルドシステムがデバッグ情報のあるファイルのリンクを回避することが可能になります。

このオプションを活用にするには、Red Hat Developer Toolset 2.0 に含まれている **GDB** のバージョンのように **.dwo** ファイルを読み取ることのできるデバッガーが必要になります。

注記

elfutils、**SystemTap**、および **Valgrind** は、**.dwo** ファイルをサポート**しません**。

4.1.1.1.4. C++ の変更点

リリース予定の標準からの実験段階の C++ 機能

g++ は、新たなコマンドラインオプション**-std=c++1y** をサポートするようになりました。このオプションは、2014年頃にリリース予定の次期改訂版の標準に計画されている機能と使用することができます。現時点での **-std=c++11** との唯一の違いは、通常機能における戻り値の型の控除で、これはN3386で提案されています。

新たな thread_local キーワード

g++ は、C++11 **thread_local** キーワードを実装するようになりました。GNU__**thread** キーワードと比較すると、**thread_local** では動的な初期化と破壊セマンティクスが可能になっています。

thread_local キーワードの使用に関しては、現在重要な制限が 1 つあります。**dlclose()** 関数を使って **thread_local** オブジェクトの定義を含む動的に読み込まれた DSO をアンロードする際に、**thread_local** オブジェクトが破壊され、そのデストラクターが呼び出され、DSO がプロセスのアドレス領域からマッピング解除されます。プロセスのスレッドがこの後に **thread_local** オブジェクトにアクセスを試みると、プログラムが予期せず終了する場合があります。その結果、プログラマーは DSO 内の **thread_local** オブジェクトがアンロード後に確実に参照されないように非常な注意を払う必要がある可能性があります。

動的な初期化については、次の項目も参照してください。

Thread-local 変数の動的な初期化

C++11 および OpenMP 標準は、thread-local および thread-private の変数が動的な (つまり、ランタイムの) 初期化を行えるようにします。これをサポートするために、このような変数の使用は必要な初期化を実行するラッパー関数を通過します。

変数の使用および定義が同一の変換ユニット内にある場合は、このオーバーヘッドは最適化することが可能ですが、使用が異なる変換ユニットにある場合は、変数が実際には動的な初期化を必要としない場合でも、かなりのオーバーヘッドがあります。定義していない変換ユニット内の変数使用が不要であることをプログラマーが確実に分かっている場合 (変数が静的に初期化されるか、定義している変換ユニット内の変数使用が別の変換ユニット内での使用前に実行されるため)、新たな **-fno-extern-tls-init** オプションを使用することでこのオーバーヘッドを回避することができます。

デフォルトでは、**g++** は **-fextern-tls-init** オプションを使用します。

C++11 属性構文

g++ は、C++11 属性構文を実装するようになりました。例を示します。

```
[[noreturn]] void f();
```

C++11 アライメント指定子

g++ は、C++11 アライメント指定子を実装するようになりました。例を示します。

```
alignas(double) int i;
```

4.1.1.1.5. Fortran の変更点

4.1.1.1.5.1. 警告

モジュールファイルのバージョン (**.mod** ファイル) が上げられました。以前のバージョンの GCC でコンパイルされた Fortran モジュールは、GCC 4.8 でコンパイルされたファイルで使用する場合は再コンパイルする必要があります。これは、このバージョンの GCC は、以前のバージョンで作成された **.mod** ファイルを読み込めないためです。これを試みると、エラーメッセージが出て失敗します。

注記

生成されたアセンブラーデータの ABI 自体は変更されていません。オブジェクトファイルおよびライブラリーは、「ABI の互換性」に記載の分以外は完全に以前のバージョンと互換性があります。

4.1.1.1.5.2. ABI の互換性

アセンブラーもしくはオブジェクトファイル内で使用される内部名のいくつかは、モジュールの仕様部分で宣言される記号用に変更されました。影響のあるモジュールもしくはそのモジュールを使用関連経由で使用しているファイルが再コンパイルされると、そのモジュールおよびそのような記号を直接使用しているすべてのファイルも再コンパイルする必要があります。この変更が影響するのは、以下のモジュール記号のみです。

- **プロシージャーポインター。** C-interoperable 関数ポインター (**type(c_funptr)**) およびプロシージャーポインターコンポーネントには影響ないことに注意してください。
- **Deferred-length 文字列。**

4.1.1.1.5.3. その他の変更点

BACKTRACE 組み込み

新たな組み込みサブルーチンである **BACKTRACE** が追加されました。このサブルーチンはユーザーコードの任意の場所でバックトレースを表示し、その後はプログラムの実行が正常に継続されます。

指数関数として「q」を使用する浮動少数点数

指数関数に (**4.0q0** などの)**q** を使用する浮動少数点数の読み込みは、以前のファイルとの互換性を高めるためにベンダー拡張としてサポートされるようになりました。I/O 用の (**4.0e0** などの) **e** に適合する標準を除いて、同等のものを使用することが強く推奨されます。

Fortran ソースコードに関しては、浮動少数点のリテラルの **q** を (適合する **qp** などをともなう **4.0e0_qp** といった) パラメーターで置き換えます。Fortran ソースコードでは、**q** を単なる **e** で置き換えることは同等ではないことに注意してください。

GFORTRAN_TMPDIR 環境変数

STATUS="SCRATCH" で開かれたファイル用のデフォルト以外のディレクトリーを特定する **GFORTRAN_TMPDIR** 環境変数は、使用されなくなりました。その代わりに**gfortran** が POSIX/GNU 標準の **TMPDIR** 環境変数をチェックし、**TMPDIR** が定義されていない場合は、**gfortran** が他の方法にフォールバックしてユーザーマニュアルに記載されているように一時ファイル用のディレクトリーを決定します。

Fortran 2003

無制限のポリモーフィック変数 (**CLASS(*)**) のサポートが追加されました。非定型の文字数はまだサポートされていません。

TS 29113

仮定タイプ (**TYPE(*)**) がサポートされるようになりました。

仮定ランクアレイ (**dimension(..)**) に対する実験的なサポートが追加されました。現時点では **gfortran** アレイ記述子が使用されており、これは『TS 29113』で定義されているアレイ記述子とは異なることに注意してください。詳細情報は、**gfortran** のヘッダーファイルを参照するか、**Chasm** 言語相互運用性ツールを使用してください。

4.1.1.1.6. x86 特有の改善点

新たな命令

GCC 4.8 では、Intel **FXSR**、**XSAVE**、および **XSAVEOPT** 命令のサポートが追加されました。対応する組み込みおよびビルトイン機能は、それぞれ **-mfxsr**、**-mxsave**、および **-mxsaveopt** のコマンドラインオプションを使用することで有効にできます。

また、**RDSEED**、**ADCX**、**ADOX**、および **PREFETCHW** 命令のサポートも追加されており、**-mrdseed**、**-madx**、および **-mprfchw** コマンドラインオプションで有効にできます。

ランタイム CPU タイプおよび ISA を検出する新たなビルトイン機能

ビルトイン関数 **__builtin_cpu_is()** が新たに追加され、ランタイム CPU が特定のタイプかどうかを検出します。この機能は CPU 名をともなう 1 つの文字列リテラル引数を受け取り、マッチする場合は正の整数を、そうでない場合はゼロを返します。たとえば、ランタイム CPU が Intel Core i7 Westmere プロセッサーの場合、**__builtin_cpu_is("westmere")** は正の整数を返します。有効な CPU 名の完全な一覧は、ユーザーマニュアルを参照してください。

ビルトイン関数である **__builtin_cpu_supports()** が新たに追加され、ランタイム CPU が特定の ISA 機能をサポートするかどうかを検出します。この機能は、ISA 機能をともなう 1 つの文字列リテラル引数を受け取り、マッチする場合は正の整数を、そうでない場合はゼロを返します。たとえば、ランタイム CPU が **SSSE3** 命令をサポートする場合、**__builtin_cpu_supports("ssse3")** は正の整数を返します。有効な ISA 名の完全なリストは、ユーザーマニュアルを参照してください。

重要

これらのビルトイン関数は、**IFUNC** 初期化などの静的コンストラクターが起動する前に呼び出され、その後に CPU 検出初期化が新たに提供されたビルトイン関数である **__builtin_cpu_init()** を使って明示的に実行される必要があります。初期化が必要なのは 1 回のみです。たとえば、以下は **IFUNC** 初期化子内部の呼び出し例になります。

```
static void (*some_ifunc_resolver(void))(void)
{
    __builtin_cpu_init();
    if (__builtin_cpu_is("amdfam10h") ...
    if (__builtin_cpu_supports("popcnt") ...
}
```

関数のマルチバージョン

関数のマルチバージョンでは、プログラマーが同一関数の複数バージョンを指定できるようになり、これらの各バージョンは特定のターゲットのあるバリアントに特化したものになります。ランタイム時には、実行される場所のターゲットしだいで適切なバージョンが自動的に実行されます。たとえば、以下のコードの一部を考えてみます。

```
__attribute__ ((target ("default"))) int foo () { return 0; }
__attribute__ ((target ("sse4.2"))) int foo () { return 1; }
__attribute__ ((target ("arch=atom"))) int foo () { return 2; }
```

関数 **foo()** が実行されると、返される結果は、プログラムがコンパイルされたアーキテクチャーではなく、プログラムが実行されるアーキテクチャーに左右されます。詳細は、GCC Wiki を参照してください。

新たな RTM および HLE 組み込み

Intel RTM および HLE 組み込み、ビルトイン関数、およびコード生成のサポートが追加され、**-mrtm** および **-mhle** のコマンドラインオプションで有効にできます。これは、*Restricted Transactional Memory* (RTM) および *Hardware Lock Elision* (HLE) 用のメモリーモデル拡張の組み込み経由で実行されます。

HLE では、2 つの新たなフラグを使ってロックを使用するハードウェア省略としてマークできます。

__ATOMIC_HLE_ACQUIRE

ロック変数上のロック省略を開始します。使用するメモリーモデルは、**__ATOMIC_ACQUIRE** またはそれ以上とします。

__ATOMIC_HLE_RELEASE

ロック変数上のロック省略を終了します。メモリーモデルは、**__ATOMIC_RELEASE** またはそれ以上とします。

たとえば、以下のコードの一部を考えてみます。

```
while (__atomic_exchange_n (& lockvar, 1, __ATOMIC_ACQUIRE
                                        | __ATOMIC_HLE_ACQUIRE))
    _mm_pause ();

// work with the acquired lock

__atomic_clear (& lockvar, __ATOMIC_RELEASE | __ATOMIC_HLE_RELEASE);
```

Restricted Transactional Memory をサポートする新たな組み込みは以下の通りです。

unsigned _xbegin (void)

トランザクションの開始を試みます。成功するとこの関数は **_XBEGIN_STARTED** を返します。そうでない場合は、トランザクションが開始できなかった理由を示すステータス値を返します。

void _xend (void)

現在のトランザクションをコミットします。アクティブなトランザクションがない場合、この関数は障害を引き起こします。このトランザクションのメモリーに関するすべての副作用は、アトミックな方法で他のスレッドに見えるようになります。

int _xtest (void)

トランザクションが現在アクティブである場合はゼロ以外の値を、そうでない場合はゼロを返します。

void _xabort (unsigned char status)

現在のトランザクションを中止します。アクティブなトランザクションがない場合、これはノーオペレーションになります。この関数により中止された **_xbegin()** コールの戻り値に パラメーター **status** が含まれます。

これらの組み立ての使用例を以下で示します。

```
if ((status = _xbegin ()) == _XBEGIN_STARTED)
{
    // some code
    _xend ();
}
```

```
else
{
    // examine the status to see why the transaction failed and possibly
retry
}
```

Transactional Synchronization Extensions (TSX) を使用したトランザクション

GCC のトランザクションメモリー機能 (**-fgnu-tm** オプション) 内にあるトランザクションは、x86 ハードウェア上に *Transactional Synchronization Extensions* (TSX) がある場合は、これを使って実行可能となりました。

AMD Family 15h プロセッサーのサポート

GCC の x86 バックエンドは、64-bit x86 命令セットのサポートを使って AMD Family 15h コアをベースとした CPU をサポートするようになりました。これは、**-march=bdver3** オプションを使って有効にすることができます。

AMD Family 16h プロセッサーのサポート

GCC の x86 バックエンドは、64-bit x86 命令セットのサポートを使って AMD Family 16h コアをベースとした CPU をサポートするようになりました。これは、**-march=btver2** オプションを使って有効にすることができます。

4.1.1.2. Red Hat Enterprise Linux 6.4 および 5.9 以降の変更点

以下の機能は、 Red Hat Enterprise Linux 6.4 および 5.9 に含まれる GCC のリリース以降に追加されたものです。

4.1.1.2.1. ステータスおよび機能

4.1.1.2.1.1. C++11

GCC 4.7 およびそれ以降のバージョンでは、**-std=c++11** または **-std=gnu++11** を使って C++11 に準拠したビルディングアプリケーションに実験的なサポートを提供しています。しかし、異なるバージョンのコンパイラーでコンパイルされた C++11 コード間での互換性は保証されていません。詳細は、「C++ ABI」 を参照してください。

C++ ランタイムライブラリーである **libstdc++** は、C++11 機能のほとんどをサポートしています。しかし、タイプの特性に関する特定のプロパティや正規表現などのいくつかの機能については、サポートされていないか、サポートが限定されています。詳細は、**libstdc++** documentation を参照してください。ここでは、実装定義された動作も一覧表示されています。

C++11 **exception_ptr** および **future** のサポートは、システム *libstdc++* パッケージでの例外処理ランタイムの変更を必要とします。この変更は、通常の Z-stream チャンネルで配布されます。これらの機能の使用時には、すべての Red Hat Enterprise Linux エラータの適用が正常なランタイム機能を確認する必要がある可能性があります。

4.1.1.2.1.2. C11

GCC 4.7 およびそれ以降のバージョンでは、ISO C 標準の C11 改訂からのいくつかの機能に対する実験的なサポートを提供しています。また、以前の (現在は非推奨) **-std=c1x** および **-std=gnu1x** コマンドラインオプションに加えて、gcc は **-std=c11** および **-std=gnu11** を受け取るようになっています。このサポートは実験段階であるため、今後のリリースでは互換性がなくなる可能性があることに注意してください。

サポート対象となる機能としては、Unicode 文字列 (事前定義のマクロ **__STDC_UTF_16__** および **__STDC_UTF_32__** を含む)、戻り値のない関数 (**_Noreturn** および <stdnoreturn.h>)、アライメントサポート (**_Alignas**、**_Alignof**、**max_align_t**、および <stdalign.h>) が挙げられます。

4.1.1.2.1.3. 並列処理および同時実行

GCC 4.7 およびそれ以降のバージョンでは、並列アプリケーションのプログラミングのサポートが改善されています。

1. GCC コンパイラーは、並列プログラミングの OpenMP API 仕様バージョン 3.1 をサポートします。この仕様に関する詳細情報については、OpenMP ウェブサイトを参照してください。

2. C++11 および C11 標準は、マルチスレッドプログラム用のプログラミング抽象化を提供します。それぞれの標準ライブラリーには、スレッドおよびロックや条件変数、future などのスレッド関連の機能用のプログラミング抽象化が含まれます。新たなバージョンの標準は、コンパイラーが提供する保証やプログラマーがマルチスレッドプログラムを作成する際に注意を必要とする制限などのマルチスレッドプログラムのランタイム動作を正確に特定するメモリーモデルも定義します。

   メモリーモデルのサポートは、まだ実験段階であることに注意してください (詳細は下記を参照)。C++11 および C11 のサポートステータスについての詳細情報は、それぞれ「C++11」と「C11」を参照してください。

このセクションの残りの部分では、2 つの GCC 新機能について詳述しています。これらの機能では、プログラマーによる同時実行の処理 (マルチスレッドが本当に並列処理されておらず、共有状態への同時アクセスを同期させる必要がある場合など) が容易になります。また、どちらもメモリーへのアクセスの原子性を提供しますが、ランタイムサポートのスコープや適用性、複雑性は異なります。

アトミックメモリーアクセス用の C++11 タイプおよび GCC ビルトイン機能

C++11 には、*アトミックタイプ*のサポートがあります。このタイプのメモリーロケーションへのアクセスはアトミックで、他のスレッドが同一メモリーのロケーションに同時にアクセスする場合でも、1 つの不可分なアクセスに見えます。この原子性は単一の読み取りもしくは書き込みアクセス、またはそれらのタイプにサポートされた他のアトミック操作の 1 つ (たとえば、種々のアトミックタイプ上で実行された 2 つの連続する操作はそれぞれ別個にアトミックになりますが、1 つの結合アトミック操作を形成するわけではありません) に限られています。

アトミックタイプは **atomic<*T*>** として宣言されます。ここでの*T* は、非アトミックベースタイプで、普通にコピー可能である必要があります。(たとえば、**atomic<int>** はアトミック整数)。GCC は、ターゲットのアーキテクチャーが提供するアトミック命令でアトミックにアクセスできるもの以外は、いかなるベースタイプ *T* をもサポートしていません。アトミックは一義的にはアーキテクチャーに依存しない形でハードウェアのプリミティブを公開するように設計されていることから、これは実際には重大な制限ではありません。ターゲット上でマシンワードよりも大きくないポインターおよび整数は、ベースタイプとしてサポートされています。まだサポートされていないベースタイプを使用すると、リンク時のエラーになります。

メモリーオーダーを含むアトミックタイプ上での操作用に生成されたコードは、C++11 標準で指定されたセマンティクスを実装します。しかし、C++11 メモリーモデルのサポートはまだ実験段階であり、たとえばコードの最適化を行う際に、GCC が常にデータ競合の自由度を保持するとは限りません。

GCC はまた、アトミックメモリーアクセス用の新たなビルトイン機能をサポートします。これは、メモリーモデルと新たなアトミック操作の設計にしたがうものです。以前の同期ビルトイン機能のセット (つまり、**__sync** の接頭辞が付いたもの) もまだサポートされています。

トランザクションメモリー

トランザクションメモリー(TM) は、コードの要素が仮想的にアトミックにかつ他のトランザクションから孤立して実行されることを仮定しているとプログラムが宣言できるようにします。すると、GCC のトランザクションメモリーのランタイムライブラリーである **libitm** が、コンパイルされたプログラムの実行時にこの原子性を確実に保証します。アトミックメモリーアクセスと比較すると、高レベルのプログラミング

抽出になります。これは、単一メモリーロケーションに限定されておらず、修正するデータ用の特別なデータタイプを必要としないためです。また、トランザクションに任意のコードを含み、他のトランザクション内にネスト化することが可能なためです (制限のいくつかは後で説明)。

GCC は、Draft Specification for Transactional Language Constructs for C++, version 1.1 で指定されたとおりにトランザクションを実行します。このドラフトでは C の言語コンストラクトが指定されていませんが、C ソースコードをコンパイルする際に GCC は既に C 互換のコンストラクトのサブセットをサポートしています。

主な言語コンストラクトはトランザクションステートメントと式で、これらに **__transaction_atomic** もしくは **__transaction_relaxed** キーワードと複合ステートメントもしくは式がそれぞれ続くことで宣言されます。以下の例では、変数 **x** が 10 未満の場合に グローバル変数**y** を増加させる方法を示してします。

```
__transaction_atomic { if (x < 10) y++; }
```

これは、プログラムのマルチスレッド実行においてもアトミックに発生します。特に、トランザクションが **x** と **y** を読み込み、**y** に保存できても、これらすべてのメモリーアクセスは仮想的に 1 つの不可分なステップとして実行されます。

C++11 メモリーモデルと一致するよう、トランザクションを使用するプログラムにはデータ競合を含んではいけないことに注意してください。トランザクションは仮想的にグローバルの総合順序で実行されるよう保証されています。この順序はトランザクションのメモリー実装で決定されており、プログラムの残りの部分が強制する前に発生した順序と一致し、これに投稿するものです (つまり、トランザクションセマンティクスは C++11 メモリーモデルに基づいて指定されます。上記のドラフト仕様のリンクを参照してください)。しかしながら、プログラムにデータ競合がある場合、未定義の動作があることになります。たとえば、スレッドが最初にあるデータを初期化し、その後に以下のようなコードで公的にアクセス可能とすることができます。

```
init(data);
__transaction_atomic { data_public = true;}  // data_public is initially
false
```

すると、以下のように別のスレッドが安全にこのデータを使用できるようになります。

```
__transaction_atomic { if (data_public) use(data); }
```

しかし、下記のコードにはデータ競合があり、このため未定義の動作に終わります。

```
__transaction_atomic { temp = copy(data); if (data_public) use(temp); }
```

ここでは、**copy(data)** は初期化スレッドの **init(data)** と競合します。これは、**data_public** が正しくない場合でも実行可能なためです。データ競合の別な例は、あるスレッドがトランザクションで変数 **x** にアクセスしており、別のスレッドが潜在的に同時にトランザクション以外でこれにアクセスしている場合です。データは以下のようにコードを使って安全に回収できることに注意してください (1 つのスレッドがこれを行なっていることが前提)。

```
__transaction_atomic { data_public = false; }
destruct(data);
```

ここでは、**destruct()** はデータの潜在的な同時使用とは競合しません。トランザクション完了後に **data_public** が false でデータがプライベートであることが保証されるためです。バックグラウンド情報については、仕様および C++11 メモリーモデルを参照してください。

トランザクションが仮想的に総合順序で実行する必要がある場合でも、時間において相互排他的に実行するわけではありません。トランザクションメモリー実装はできる限り並列でトランザクション実行を試みてスケーラブルなパフォーマンスを提供します。

トランザクションには、*atomic transactions* (**__transaction_atomic**) と *relaxed transactions* (**__transaction_relaxed**) の 2 つのバリアントがあります。1 つ目は、他のすべてのコードに関する原子性を保証しますが、アトミックや揮発性メモリーアクセスなどのトランザクションではない同期を含まないことが分かっているコードのみを許可します。対称的に 2 つ目はすべてのコード (たとえば、I/O 関数へのコール) を許可しますが、他のトランザクションに関する原子性のみを提供します。このため、アトミックトランザクションは他のアトミックトランザクションと relaxed トランザクション内でネスト化が可能ですが、relaxed トランザクションは他の relaxed トランザクション内でしかネスト化ができません。更に、relaxed トランザクションはより低いパフォーマンスで実行される可能性が高いものの、これは、実装および利用可能なハードウェアによります。

GCC はコンパイル時にこれらの制限を静的に検証します (たとえば、アトミックトランザクション内から呼び出しが許可されるコードに関する要件)。これは、他のコンパイルユニット内 (ソースファイル) もしくはライブラリー内で定義される関数をトランザクションがいつ呼び出すかに影響してきます。トランザクションコードに関するこのようなコンパイルユニットにまたがる呼び出しを有効にするには、アトミックトランザクション内からの仕様が安全なコードを含むようにそれぞれの式をマークする必要があります。プログラマーは、**transaction_safe** 関数属性をこれらの関数の宣言に追加し、この宣言を関数の定義時に含めることで、これが可能になります。すると、GCC はこれらの関数内のコードがアトミックトランザクションに安全であることを確認し、それに応じてコードを生成します。プログラマーがこれらの制限やステップにしたがわないと、コンパイル時またはリンク時のエラーが発生します。コンパイルユニット内では、GCC は関数がトランザクション内での使用で安全かどうかを自動的に検出します。このため、通常は属性を追加する必要はありません。詳細は、上記のドラフト使用のリンクを参照してください。

GCC のトランザクションメモリーサポートは、該当トランザクションを使用しないプログラムのパフォーマンスとトランザクション以外のコードのパフォーマンスを低下させない設計になっています。ただし、CPU といった同一リソースを使用する同時スレッドによる通常の妨害を除きます。

GCC および **libitm** におけるトランザクションメモリーサポートはまだ実験段階です。言語コンストラクトの仕様が進化したり要件の実装が理由で ABI と API の両方が必要になった場合は、今後これらが変更される可能性があります。**-fgnu-tm** コマンドラインオプションで構築されたアプリケーションを実行する際には、適切なバージョンの **libitm.so.1** 共有ライブラリーもインストールされていることが現時点での必要条件となっていることに注意してください。

4.1.1.2.1.4. アーキテクチャー固有のオプション

Red Hat Developer Toolset 2.0 は、Red Hat Enterprise Linux 5 および 6 のみで利用可能で、32 ビットと 64 ビットの Intel および AMD アーキテクチャーの両方に対応しています。このため、以下のオプションはこれらのアーキテクチャーにのみ関連するものです。

プロセッサーのいくつかに対する最適化は、表4.5「プロセッサー最適化オプション」 で記載されているコマンドラインオプションで利用可能となっています。

表4.5 プロセッサー最適化オプション

オプション	説明
-march=core2 および **-mtune=core2**	Intel Core 2 プロセッサー用の最適化。
-march=corei7 および **-mtune=corei7**	Intel Core i3、i5、および i7 プロセッサー用の最適化。
-march=corei7-avx および **-mtune=corei7-avx**	AVX をともなう Intel Core i3、i5、および i7 プロセッサー用の最適化。
-march=core-avx-i	RDRND、FSGSBASE、および F16C のあるコード名 Ivy Bridge の Intel プロセッサー用の最適化。

オプション	説明
-march=core-avx2	AVX2、FMA、BMI、BMI2、および LZCNT のある Intel からの次世代プロセッサー用の最適化。
-march=bdver2 および **-mtune=bdver2**	コード名 Piledriver の AMD Opteron プロセッサー用 の最適化。
-march=btver1 および **-mtune=btver1**	コード名 Bobcat の AMD family 14h プロセッサー用 の最適化。
-march=bdver1 および **-mtune=bdver1**	コード名 Bulldozer の AMD family 15h プロセッサー用の最適化。

プロセッサー固有の各組み込みおよび命令は、表4.6「プロセッサー固有の組み込みおよび命令のサポート」で記載されているコマンドラインオプションで利用可能となっています。

表4.6 プロセッサー固有の組み込みおよび命令のサポート

オプション	説明
-mavx2	Intel AVX2 組み込み、ビルトイン関数、およびコード生成のサポート。
-mbmi2	Intel BMI2 組み込み、ビルトイン関数、およびコード生成のサポート。
-mlzcnt	**lzcnt** 命令を使った **__builtin_clz*** の実装および自動生成。
-mfma	Intel FMA3 組み込みおよびコード生成のサポート。
-mfsgsbase	専用のビルトイン機能で新たなセグメントレジスタ読み込みおよび書き込み命令の生成を有効にします。
-mrdrnd	Intel **rdrnd** 命令のサポート。
-mf16c	新たな 2 つの AVX ベクトル変換命令に対するサポート。
-mtbm	TBM (Trailing Bit Manipulation) ビルトイン関数およびコード生成に対するサポート。
-mbmi	AMD の BMI (Bit Manipulation) ビルトイン関数およびコード生成に対するサポート。
-mcrc32	**crc32** 組み込みのサポート。
-mmovbe	**__builtin_bswap32** および **__builtin_bswap64** を実装するための **movbe** 命令の使用を有効にします。
-mxop、**-mfma4**、および **-mlwp**	AMD Orochi プロセッサー用の XOP、FMA4、および LWP 命令セットのサポート。
-mabm	AMD プロセッサーでの **popcnt** および **lzcnt** 命令の使用を有効にします。
-mpopcnt	AMD および Intel プロセッサーでの **popcnt** 命令の使用を有効にします。

x87 浮動少数点ユニットを使用する際には、GCC は浮動少数点の過剰な精度の処理において ISO C99 に準ずるコードを生成するようになっています。これは **-fexcess-precision=standard** で有効になり、**-fexcess-precision=fast** で無効にできます。**-std=c99** などの標準に準ずるオプション使用時には、この機能はデフォルトで有効となります。

ベクトルタイプの **vector long long** もしくは **vector long** は、他の **VSX** 命令セットのベクトルと同じ方法で渡され、返されます。これまで GCC は、64-bit 整数ベースタイプの 128-bit ベクトル用の ABI に準拠していませんでした。

-mrecip コマンドラインオプションが追加されました。これは、逆数および逆数平方根命令を使うべきかどうかを示します。

-mveclibabi=mass コマンドラインオプションが追加されました。これを使うと、Mathematical Acceleration Subsystem ライブラリーを使ってコンパイルが数学関数を自動的にベクトル化できるようになります。

-msingle-pic-base コマンドラインオプションが追加されました。これは、コンパイラーに関数プロローグで **PIC** ベースレジスタを読み込まないように命令します。PIC ベースレジスタは、ランタイムシステムが初期化する必要があります。

-mblock-move-inline-limit コマンドラインオプションが追加されました。これは、ユーザーがインライン inlined **memcpy** 呼び出しおよび同様の最大サイズを制御できるようにするものです。

4.1.1.2.1.5. リンク時の最適化

リンク時の最適化(LTO) はコンパイル技術で、GCC がネイティブコードに加えてコンパイル済みの各入力ファイルの内部表現を生成し、この両方を出力オブジェクトファイルに書き込むものです。その後、いくつかのオブジェクトファイルが一緒にリンクされると、GCC はこのコンパイル済みコードの内部表現を使ってすべてのコンパイルユニットにまたがって手順間を最適化します。これは生成済みコードのパフォーマンスを改善する可能性があります (たとえば、あるファイル内で定義された関数が別のファイルで呼び出された場合にインライン化できる可能性があります)。

LTO を有効にするには、コンパイル時とリンク時の両方で **-flto** オプションを指定する必要があります。リンカーとの相互運用性および LTO の並列実行を含む詳細については、GCC 4.7.0 Manual 内の **-flto** ドキュメンテーションを参照してください。また、この内部表現は安定したインターフェースではないため、LTO は同一バージョンの GCC が生成したコードにのみ適用されることに注意してください。

注記

デバッグ生成におけるリンク時の最適化は、gcc 4.7 および 4.8 ではまだサポートされていません。このため、**-flto** と **-g** オプションを合わせた使用は、Red Hat Developer Toolset ではサポート対象外となります。

4.1.1.2.1.6. その他

-Ofast が全般的な最適化レベルとしてサポートされるようになりました。これは**-O3** と同様に作動し、より最適化されたコードを生成することが可能なオプションを追加しますが、一方で標準コンプライアンスを無効にする可能性があります (たとえば、**-ffast-math** は **-Ofast** で有効になります)。

GCC は、**const**、**pure**、および **noreturn** といった属性をヘッダーファイルで宣言された関数に追加することで改善する可能性のあるコードについてケースをユーザーに知らせることができます。これを有効にするには、**-Wsuggest-attribute=[const|pure|noreturn]** コマンドラインオプションを使用します。

アセンブラーコードは、C コードでのラベルへのジャンプを可能にする goto 機能を利用できます。

4.1.1.2.2. 言語の互換性

本セクションでは、Red Hat Developer Toolset コンパイラーと Red Hat Enterprise Linux システムコンパイラーのプログラム言語レベルでの互換性について説明します (たとえば、C99 などの言語標準の実装における違いや **-Wall** が生成する警告の変更点など)。

バグ修正による変更点もあれば、新標準をサポートするために意図的にこれまでの動作を変更したものもあります。また、コンパイルやランタイムパフォーマンスを促進するために標準に適合する方法で緩和したものもあります。これらの変更点には、一見しただけではわからず、以前のバージョンの更新時に問題を引き起こさないものもあります。しかし、いくつかものは目に見えるもので、Red Hat Developer Toolset バージョンの GCC に移ったユーザーを悩ませる可能性もあります。以下では、主な問題点の特定を試み、解決法を示します。

4.1.1.2.2.1. C

GCC は、定数式を C90 および C99 に準拠する方法で処理するようになっています。コンパイラーで定数に変換可能なコード式で、実際には ISO C が定義する定数式でないものについては、警告またはエラーが出る可能性があります。

ライブラリー関数の不適格な再定義は、コンパイラーが受け取らないようになりました。特に、ライブラリー関数のビルトイン宣言に類似したシグネチャーがある関数 (たとえば、**abort()** や **memcpy()**) が再定義とみなされるには、**extern "C"** で宣言される必要があります。そうでないと、不適格なものとなります。

重複メンバー

以下の **struct** 宣言を見てみましょう。

```
struct A { int *a; union { struct { int *a; }; }; };
```

これまでは、この宣言は C++ コンパイラーのみが診断していましたが、C コンパイラーも診断するようになっています。共用体と構造体が匿名なため、**.a** が実際に参照するものがあいまいで、フィールドの 1 つは名前を変更する必要があります。

4.1.1.2.2.2. C++

ヘッダー依存関係の変更点

これまでは **<unistd.h>** に実装詳細として (**gthr*.h** に機能マクロを獲得する目的で) 含まれていた **<iostream>**、**<string>**、およびその他の STL ヘッダーは、含まれていません。これは、C++ 標準違反であるためです。診断結果の出力は、以下のようになります。

```
error: ‘truncate’ was not declared in this scope
error: ‘sleep’ was not declared in this scope
error: ‘pipe’ was not declared in this scope
error: there are no arguments to 'offsetof' that depend on a template
parameter, so a declaration of 'offsetof' must be available
```

これを修正するには、該当するソースまたはヘッダーファイルの最初の方に以下の行を追加します。

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

標準 C++ ライブラリーインクルードファイルの多くは、**size_t** および **ptrdiff_t** の namespace-**std**-scoped バージョンを獲得する **<cstddef>** を含めないように編集されました。このため、**<cstddef>** 含まずにマクロ **NULL** または **offsetof** を使用する C++ プログラムは、コンパイルを行わなくなります。診断結果の出力は、以下のようになります。

```
error: 'ptrdiff_t' does not name a type
error: 'size_t' has not been declared
error: 'NULL' was not declared in this scope
error: there are no arguments to 'offsetof' that depend on a template
parameter, so a declaration of 'offsetof' must be available
```

この問題を修正するには、以下の行を追加します。

```
#include <cstddef>
```

名前検索の変更点

G++ はこれまで間違って実行してきた余分な無条件検索を行わなくなりました。その代わり、2 段階の検索ルールを正常に実装し、テンプレートに使用されている非修飾名は以下のどちらかを満たす適切な宣言がある必要があります。

1. テンプレートの定義時にスコープ内であること。
2. インスタンス化の際に変数に依存する検索で見つけられること。

インスタンス化の際に 2 番目の無条件検索に不適当に依存するコード (テンプレート後もしくは依存ベース内で宣言される関数の発見など) は、コンパイル時のエラーにつながります。

ケースによっては、G++ が提供する診断結果にバグ修正のヒントが含まれている場合もあります。以下のコードを見てみましょう。

```
template<typename T>
int t(T i)
{
    return f(i);
}

int f(int i)
{
    return i;
}

int main()
{
    return t(1);
}
```

以下の診断結果が出力されます。

```
In instantiation of ‘int t(T) [with T = int]’
required from here
error: ‘f’ was not declared in this scope, and no declarations were found
by argument-dependent lookup at the point of instantiation [-fpermissive]
note: ‘int f(int)’ declared here, later in the translation unit
```

この例でエラーを修正するには、テンプレート関数 **t()** の定義前に関数**f()** の宣言を移動します。**-fpermissive** コンパイラーフラグがコンパイル時のエラーを警告に変え、一時的な回避策として使用することができます。

初期化していない const

以下の宣言を見てみましょう。

```
struct A { int a; A (); };
struct B : public A { };
const B b;
```

このコードをコンパイルする試みは、以下のエラーが出て失敗します。

```
error: uninitialized const ‘b’ [-fpermissive]
note: ‘const struct B’ has no user-provided default constructor
```

これは **B** にユーザーが提供するデフォルトのコンストラクターがないために発生します。イニシャライザーを提供するか、デフォルトのコンストラクターを追加する必要があります。

テンプレートのインスタンス化の視認性

テンプレートのインスタンス化における ELF 記号の視認性は、テンプレート引数の視認性で適切に制限されるようになりました。たとえば、**struct my_hidden_struct** といった非表示のユーザー定義タイプで **std::vector** のような標準ライブラリーコンポーネントをインスタンス化するユーザーは、**std::vector<my_hidden_struct>** 記号で非表示の視認性を求めることが可能です。その結果、**-fvisibility=hidden** コマンドラインオプションでコンパイルを行うユーザーは、使用するライブラリーヘッダーから含まれるタイプの視認性を認識することになります。ヘッダーが明示的に記号の視認性を制御しない場合は、これらのヘッダーからのタイプは、これらのタイプを使用するインスタンス化とともに非表示となります。たとえば、以下のコードを見てみましょう。

```
#include <vector>                // template std::vector has default
visibility
#include <ctime>                 // struct tm has hidden visibility
template class std::vector<tm>; // instantiation has hidden visibility
```

ライブラリーヘッダー **<foo.h>** の視認性を調整する方法の 1 つは、以下のもので構成される転送ヘッダーを **-I** インクルードパス上に作成することです。

```
#pragma GCC visibility push(default)
#include_next <foo.h>
#pragma GCC visibility push
```

ユーザー定義のリテラルサポート

C++ を **-std={c++11,c++0x,gnu++11,gnu++0x}** コマンドラインオプションでコンパイルする際には、GCC 4.7.0 およびそれ以降ではそれ以前のバージョンとは違い、有効な ISO C++03 コードのいくつかと互換性のないユーザー定義のリテラルがサポートされています。特に、文字列の後に有効なユーザー定義のリテラルとなるものの前に空白が必要となります。以下のコードを見てみましょう。

```
const char *p = "foobar"__TIME__;
```

C++03 では、**__TIME__** マクロは文字列リテラルまで拡大し、別のものと連結されます。C++11 では、**__TIME__** 拡大されず、代わりに演算子 **"" __TIME__** が検索され、その結果、以下のような警告が出されます。

```
error: unable to find string literal operator ‘operator"" __TIME__’
```

これは、空白のないマクロが続く文字列リテラルに適用されます。これを修正するには、文字列リテラルとマクロ名の間に空白を加えます。

Temporary アドレスの除去

以下のコードを見てみましょう。

```
struct S { S (); int i; };
void bar (S *);
void foo () { bar (&S ()); }
```

これまでは、このコードをコンパイルしようとすると警告メッセージが出ましたが、今ではエラーが出て失敗するようになりました。これは、変数を追加し、temporary ではなくこの変数のアドレスを渡すことで修正可能です。**-fpermissive** コンパイラーフラグが、コンパイル時のエラーを警告に変え、一時的な回避策として使用することができます。

その他

G++ は事前定義されたマクロの **__cplusplus** を C++98/03 には**199711L** に、C++11 には **201103L** という正しい値に設定するようになっています。

G++ は、一時オブジェクトのライフタイムが終了する際に、これらのオブジェクトに割り当てられていたスタック領域を適切に再利用するようになりました。これは、いくつかの C++ 関数ではスタック消費を大幅に削減可能とします。その結果、未定義の動作のあるコードの中には中断するものもあります。

関数内の外部宣言が括弧内のコンテキストの宣言と適合しない場合、G++ は関数定義のすぐ前で開かれている名前領域ではなく、関数の名前領域内の名前を適切に宣言します。

G++ には、C++ 標準のコア問題についての解決案が実装されています。すべてのサブオブジェクトを初期化する場合はデフォルトの初期化が許可され、コンパイルに失敗したコードは以下のようなイニシャライザーを提供することで修正できます。

```
struct A { A(); };
struct B : A { int i; };
const B b = B();
```

テンプレートで使用される **typedef** 名にアクセス制御が適用されるようになりました。これにより、以前のリリースでは受け付けられた間違ったコードのいくつかが G++ に拒否される場合があります。このコードが訂正されるまでは、**-fno-access-control** オプションを一時的な回避策として使用できます。

G++ では、C++ 標準のコア問題 176 が実装されています。これまでは、G++ ではテンプレートベースクラスの injected-class-name の使用はサポートされていませんでした。このため、この名前の検索は、囲まれたスコープ内のテンプレートの宣言を見つけていましたが、injected-class-name を見つけるようになりました。これは、名前の後にテンプレート引数リストが続くかどうかにより、タイプまたはテンプレートとして使用できるものです。この変更の結果、以前は受け付けられていたコードが間違ったものになる場合があります。その理由は、

1. injected-class-name がプライベートベースからのものであることからアクセスできないため。
2. または、injected-class-name をテンプレートのパラメーター用に引数として使用できないため。

どちらの場合でも、nested-name-specifier を明示的にテンプレート命名に加えることで修正可能です。前者の場合は **-fno-access-control** を追加し、後者は **-pedantic** でのみ拒否されます。

4.1.1.2.2.3. C/C++ 警告

GCC 4.7.0 およびそれ以前では、デフォルトで有効になっているもしくは **-Wall** オプションを使うことで有効になる多くの新たな警告が追加されています。これらの警告だけではコンパイル失敗につながりませんが、多くの場合 **-Wall** は **-Werror** と合わせて使われ、これらの警告がエラーのように作動することを引き起こします。本セクションでは、これらの新たな警告および新たに有効となった警告の一覧を紹介します。特に記載がない限り、これらの警告は C と C++ の両方に適用されます。

-Wall コマンドラインオプションの動作が変更され、新たな警告フラグ**-Wunused-but-set-variable** および **-Wall -Wextra** の場合は **-Wunused-but-set-parameter** が含まれます。これにより、以前のバージョンの GCC で正常にコンパイルされたコード内に新たな警告が出る場合があります。たとえば、以下のコードを見てみましょう。

```
void fn (void)
{
```

```
    int foo;
    foo = bar ();  /* foo is never used.  */
}
```

以下の診断結果が出力されます。

```
warning: variable "foo" set but not used [-Wunused-but-set-variable]
```

この問題を修正するには、まず未使用の変数もしくはパラメーターを結果や周りのコードの論理を変更せずに削除できるかどうかを検討します。これが可能でなければ、**__attribute__((__unused__))** の注釈を付けます。回避策として、**-Wno-error=unused-but-set-variable** または **-Wno-error=unused-but-set-parameter** コマンドラインオプションを使用することができます。

-Wenum-compare オプションを使うと、異なる enum タイプの値を比較している際に、GCC が警告を出すようになります。これまでは、このオプションは C++ プログラムのみで作動しましたが、C でも作動するようになりました。この警告は **-Wall** で有効になり、タイプキャストを使用することで回避可能となります。

整数をより大きいポインタータイプにキャストすると、GCC はデフォルトで警告を表示します。この警告を表示させないようにするには、**-Wno-int-to-pointer-cast** オプションを使用します。これは、C と C++ の両方で使用できます。

NULL と非ポインタータイプ間での変換を行うと、GCC はデフォルトで警告をレポートするようになっています。これまでは、**-Wconversion** を明示的に使用した場合にのみ、これらの警告が表示されていました。この警告を表示させないようにするには、新しい **-Wno-conversion-null** コマンドラインオプションを使用します。

GCC は、仮想関数および非仮想デストラクターを持つクラスを **delete** を使って破棄する際に警告を発することができます。ポインターは仮想デストラクターのないベースクラスを参照する可能性があるので、これは安全ではありません。警告は **-Wall** と新しいコマンドラインオプションである**-Wdelete-non-virtual-dtor** で有効にします。

-Wc++11-compat と **-Wc++0x-compat** オプションが新たに利用可能になりました。これらのオプションを使うと、GCC は ISO C++ 1998 と ISO C++ 2011 の間で意味が異なる C++ コンストラクト (ISO C++ 2011 ではキーワードだが ISO C++ 1998 では識別子、など) についての警告を表示するようになります。この警告は **-Wall** で有効になり、**-Wnarrowing** オプションを有効にします。

4.1.1.2.2.4. Fortran

4.1.1.2.2.4.1. 新しい機能

- 新たなコンパイルフラグ **-fstack-arrays** が追加されました。このフラグは、すべてのローカルアレイをスタックメモリー上に置くことで、いくつかのプログラムのパフォーマンスを飛躍的に改善します。非常に大型のローカルアレイを使用するプログラムの場合、ユーザーがスタックメモリー用のランタイム制限を拡大する必要がある場合があることに注意してください。
- コンパイル時間が大幅に改善されました。たとえば、大型のアレイコンストラクターを使用するプログラムで作業すると、この改善点を認識できるはずです。
- コード生成と診断を改善するために、**-fwhole-file** コンパイルフラグがデフォルトで有効になっており、新たにサポートされている **-fwhole-program** フラグと合わせて使うことができます。これを無効にするには、非推奨の **-fno-whole-file** フラグを使用します。
- 新たなコマンドラインオプションの **-M** がサポートされるようになりました。**gcc** と同様に、このオプションを使うと、Makefile 依存関係が生成できます。**-cpp** オプションも必要となる場合があることに注意してください。

- **-finit-real=** コマンドラインオプションが**snan** を有効な値としてサポートするようになりました。これにより、signaling NaN (*非数*) をともなう REAL および COMPLEX 変数の初期化が可能になり、トラッピングを有効にすることが必要になります (たとえば、**-ffpe-trap=** コマンドラインオプションの使用で)。コンパイル時の最適化は、NaN のシグナル化を quiet NaN に変える場合があることに注意してください。
- 新たなコマンドラインオプション **-fcheck=** が追加されました。これは、以下の引数を受け取ります。
  - **-fcheck=bounds** オプションは、**-fbounds-check** コマンドラインオプションと同等のものです。
  - **-fcheck=array-temps** オプションは、**-fcheck-array-temporaries** コマンドラインオプションと同等のものです。
  - **-fcheck=do** オプションは、ループ反復変数の無効な修正をチェックします。
  - **-fcheck=recursive** オプションは、再帰的とマークされていないサブルーチンや関数への再帰呼び出しをチェックします。
  - **-fcheck=pointer** オプションは呼び出しのポインター関連チェックを実行しますが、未定義のポインターや式内のポインターは処理しません。
  - **-fcheck=all** オプションが上記のオプションすべてを有効にします。
- 新たなコマンドラインオプション **-fno-protect-parens** が追加されました。このオプションは、コンパイラーが括弧に関係なく REAL および COMPLEX の式を並び替えることを可能にします。
- OpenMP の **WORKSHARE** を使用すると、アレイの割り当ておよび**WHERE** が並行して実行されます。
- より多くの Fortran 2003 および Fortran 2008 数学関数が初期化式として使用可能となっています。
- **GCC$** コンパイラー指示文が**STDCALL** などの拡張属性をサポートできるようになりました。

4.1.1.2.2.4.2. 互換性の変更点

- **-Ofast** コマンドラインオプションが自動的に**-fno-protect-parens** および **-fstack-arrays** フラグを有効にします。
- フロントエンドの最適化が **-fno-frontend-optimize** オプションで無効にでき、**-ffrontend-optimize** オプションで選択可能になっています。前者の使用は、無効な Fortran ソースコードをコンパイルする必要がある場合 (たとえば、サブルーチンと比較した場合に関数に副作用がある場合) やコンパイラーバグを回避する場合に限定することをお勧めします。
- **GFORTRAN_USE_STDERR** 環境変数が削除され、GNU Fortran が常に標準エラーにエラーメッセージを表示します。
- **-fdump-core** コマンドラインオプションと**GFORTRAN_ERROR_DUMPCORE** 環境変数が削除されました。重大なエラーが発生した場合は、GNU Fortran は常にプログラムの実行を中止します。
- **-fbacktrace** コマンドラインオプションはデフォルトで有効になっています。致命的なエラーが発生すると、GNU Fortran はプログラムの実行を中止する前にバックトレースを標準エラーに表示しようとします。この動作を無効にするには、**-fno-backtrace** オプションを使用します。
- GNU Fortran は、モジュールパス用に Makefile 依存関係を生成する **-M** コマンドラインオプションをサポートしなくなりました。この操作を実行するには、代わりに **-J** オプションを使用してください。

- 警告の数を大幅に減らすために、**-Wconversion** コマンドラインオプションは変換で情報が失われる場合にのみ警告を表示するようになっており、他の変換についての警告を表示するために新たなコマンドラインオプション **-Wconversion-extra** が追加されました。**-Wconversion** オプションは **-Wall** に有効にします。

- 新たなコマンドラインオプション **-Wunused-dummy-argument** が追加されました。このオプションを使うと未使用ダミー引数についての警告が表示でき、**-Wall** で有効にできるようになっています。**-Wunused-variable** オプションは以前に未使用ダミー引数についても警告をしていたことに注意してください。

- **COMMON** のデフォルトのパディングが変更されました。これまでは、変数の前にパディングが追加されていました。変更後は変数の後にパディングを追加して他のベンダーとの互換性を高めています。また、ケースによっては、正確な出力の獲得向上にもつながります。この動作は **-falign-commons** オプションと対照的であることに注意してください。

- GNU Fortran は **libgfortranbegin** ライブラリーに対してリンクしないようになりました。**MAIN__** アセンブラー記号が実際の Fortran メインプログラムで、**main** 機能により起動します。これは **MAIN__** と同じオブジェクトファイルで生成され、置かれます。**libgfortranbegin** ライブラリーは、後方互換性のためにまだ存在していることに注意してください。

4.1.1.2.2.4.3. Fortran 2003 の機能

- ライブラリーおよびプログラム間のポリモーフィズムのサポートと、複雑な継承パターンのサポートは改善されましたが、まだ実験段階です。

- 派生型と同じ名前のジェネリックインターフェース名がサポートされるようになり、これによりコンストラクター関数の作成が可能になりました。Fortran は静的コンストラクター関数をサポートしないことに注意してください。利用可能なのは、デフォルトの初期化か明示的な構造-コンストラクター初期化のみです。

- 自動 (再) 割り当て。割り当て可能な変数への組み込み割り当てでは、左側が自動的に割り当て (割り当てられていない場合) られるか、再割り当て (シェイプもしくは型パラメーターが異なる場合) されます。小型のパフォーマンスペナルティーを避けるには、アレイおよび文字列に **a = ...** の代わりに **a(:) = ...** を使うことができます。もしくは、**-std=f95** または **-fno-realloc-lhs** を使ってこの機能を無効にできます。

- **ASSOCIATE** コンストラクトの実験段階のサポートが追加されました。

- ポインターの割り当てでは、ポインターの下限を指定することが可能になり、ランク 1 または単純な連続データ-ターゲットでは範囲を再マッピングすることが可能になっています。

- 遅延の型パラメーター。スカラー割り当て可能な変数およびポインター変数では、文字の長さを保留できるようになっています。

- 割り当て可能な属性、ポインター属性、および非定型の長さ型パラメーターのある名前リスト変数がサポートされるようになりました。

- プロシージャーポインター関数結果およびプロシージャーポインターコンポーネントのサポートが追加されました (PASS を含む)。

- 割り当て可能なスカラー (実験段階)、 **DEFERRED** 型にバインドされたプロシージャー、**ALLOCATE** および **DEALLOCATE** ステートメントの**ERRMSG=** 引数が新たにサポートされるようになりました。

- **ALLOCATE** ステートメントが型の定義、および**SOURCE=** 引数をサポートするようになりました。

- 出力の丸め (**ROUND=**, **RZ**, ...) がサポートされるようになりました。

- **ISO_C_BINDING** 組み込みモジュール型パラメーターの**INT_FAST{8,16,32,64,128}_T** フォーマットがサポートされるようになりました。

- **OPERATOR(*)** と **ASSIGNMENT(=)** が、**GENERIC** 型にバインドされたプロシージャーとして (つまり、型にバインドされた演算子として) 許可されています。

4.1.1.2.2.4.4. Fortran 2003 の互換性

型にバインドされたプロシージャーまたはプロシージャーポインターをともなう拡張可能な派生型で **PASS** 属性のあるものは、Fortran 2003 標準にしたがって **CLASS** を使用する必要があります。**TYPE** を使用する回避策は、サポート対象外となりました。

4.1.1.2.2.4.5. Fortran 2008 の機能

- 新たなコマンドラインオプション **-std=f2008ts** が追加されました。このオプションは、Fortran 2008 標準および Fortran の C とのさらなる互換性に関する技術的仕様 (TS) 29113 に準拠するプログラムのサポートを有効にします。詳細は、Chart of Fortran TS 29113 Features supported by GNU Fortran を参照してください。
- **DO CONCURRENT** コンストラクトがサポートされています。このコンストラクトを使用して、個別のループ反復が相互依存性を持たないように指定することが可能です。
- ポリモーフィック coarray を除く完全な単一イメージのサポートが追加され、**-fcoarray=single** コマンドラインオプションを使うことで有効にできます。また、GNU Fortran は MPI ベースの coarray コミュニケーションライブラリー経由で複数イメージの予備サポートも提供します。リモートの coarray アクセスがまだ可能でないため、ライブラリーバージョンが不安定であることに注意してください。
- **STOP** および **ERROR STOP** ステートメントが更新され、すべての定数式をサポートするようになりました。
- **CONTIGUOUS** 属性がサポートされています。
- **MOLD** 引数をともなう **ALLOCATE** がサポートされています。
- **STORAGE_SIZE** 組み込み問い合わせ関数がサポートされています。
- **NORM2** および **PARITY** 組み込み関数がサポートされています。
- 以下のビット組み込みが追加されました。
  - 1 ビットの数をカウントし、パリティを返す **POPCNT** および **POPPAR** ビット組み込み。
  - ビット単位の比較用に **BGE**、**BGT**、**BLE**、および **BLT** ビット組み込み。
  - 左および右シフトの組み合わせ用に **DSHIFTL** および **DSHIFTR** ビット組み込み。
  - 単純な左および右揃えマスク用の **MASKL** および **MASKR** ビット組み込み。
  - マスクを使ったビット単位のマージ用に **MERGE_BITS** ビット組み込み。
  - シフト演算用の **SHIFTA**、**SHIFTL**、および **SHIFTR** ビット組み込み。
  - 変換ビット組み込みの **IALL**、**IANY**、および **IPARITY**。
- **EXECUTE_COMMAND_LINE** 組み込みサブルーチンがサポートされています。
- プロシージャー用の **IMPURE** 属性がサポートされています。これにより、**PURE** の制限なしで **ELEMENTAL** プロシージャーの使用が可能になっています。
- Null ポインター (**NULL()** を含む) および未割り当て変数を使って、オプションの非ポインター、割り当て可能でない仮引数、不完全引数の表示を実際の引数として 使用できます。

- **TARGET** 属性をともなう非ポインター変数を、**INTENT(IN)** をともなう **POINTER** ダミーへの実際の引数として使用できるようになっています。
- プロシージャーポインターおよび派生型タイプのプロシージャーポインター (ポインターコンポーネント) を含むポインターが **NULL** のみではなく、ターゲットでも初期化できるようになりました。
- (コンストラクト名のある) **EXIT** ステートメントを使って **DO** に加え、**ASSOCIATE**、**BLOCK**、**IF**、**SELECT CASE**、および **SELECT TYPE** コンストラクトを解除できるようになっています。
- 内部プロシージャーを実際の引数として使用できます。
- 組み込みモジュール **ISO_FORTRAN_ENV** の名前付き定数値 **INTEGER_KINDS**、**LOGICAL_KINDS**、**REAL_KINDS**、および **CHARACTER_KINDS** が追加されました。これらのアレイには、各タイプのサポートされた「有益な」値が含まれています。
- **ISO_C_BINDINGS** 組み込みモジュールの **C_SIZEOF** モジュールプロシージャーおよび **ISO_FORTRAN_ENV** 組み込みモジュールの **COMPILER_VERSION** および **COMPILER_OPTIONS** モジュールプロシージャーが実装されました。
- **OPEN** ステートメントが **NEWUNIT=** オプションをサポートしています。このオプションは、一意のファイルユニットを返すので、プログラムの異なる部分で同一ユニットが間違って使用されることを防止します。
- 無制限のフォーマットアイテムがサポートされています。
- **ISO_FORTRAN_ENV** 組み込みモジュール型パラメーターの **INT{8,16,32}** および **REAL{32,64,128}** フォーマットがサポートされています。
- **TAN**、**SINH**、**COSH**、**TANH**、**ASIN**、**ACOS**、および **ATAN** 関数での複素引数の使用が可能になりました。また、新たな関数 **ASINH**、**ACOSH**、および **ATANH** が実引数および複素引数用に追加され、**ATAN(Y,X)** が **ATAN2(Y,X)** のエイリアスとして機能します。
- **BLOCK** コンストラクトが実装されました。

4.1.1.2.2.4.6. Fortran 2008 の互換性

GCC での **ASYNCHRONOUS** 属性の実装は、『TS 29113: Technical Specification on Further Interoperability with C』 のドラフト候補と互換性があります。

4.1.1.2.2.4.7. Fortran 77 の互換性

GNU Fortran コンパイラーが **-fno-sign-zero** オプションと発行されると、ゼロが常にプラスであるかのように **SIGN** 組み込みが動作します。

4.1.1.2.3. ABI の互換性

本セクションでは、*アプリケーションバイナリインターフェース*(ABI) レベルでの Red Hat Developer Toolset コンパイラーとシステムコンパイラーの互換性について説明します。

4.1.1.2.3.1. C++ ABI

アップストリームの GCC コミュニティーの開発では、GCC のメジャーバージョンにまたがる C++11 ABI の互換性を保証していないため、Red Hat Developer Toolset での C++11 の使用においても同じ状況となっています。このため、Red Hat Developer Toolset 2.0 における **-std=c++11** オプションの使用は、そのフラグでコンパイルされたすべての C++ オブジェクトが同じメジャーバージョンの Red Hat Developer Toolset を使って構築された場合にのみサポートされます。**-std=c++0x** または **-**

std=gnu++0x フラグを使って Red Hat Enterprise Linux 5 もしくは 6 のシステムツールチェーン GCC で構築されたオブジェクト、バイナリ、およびライブラリーを **-std=c++11** または **-std=gnu++11** フラグを使って Red Hat Developer Toolset の GCC で構築されたものと混ぜても、明示的にはサポートされません。

Red Hat Developer Toolset の最近のメジャーバージョンでは、より新しい GCC のメジャーリリースを使用している可能性が高いため、**-std=c++11** または **-std=gnu++11** オプションで構築されたオブジェクト、バイナリ、およびライブラリーの前方互換性は保証、サポートがされていません。

Red Hat Developer Toolset の言語標準のデフォルト設定は C++98 です。このデフォルトモードで構築された (または **-std=c++98** で明示的に) C++98 準拠のバイナリもしくはライブラリーはいかなるものでも、Red Hat Enterprise Linux 5 または 6 システムツールチェーン GCC で構築されたバイナリおよび共有ライブラリーと自由に混ぜ合わせることができます。Red Hat では、本番ソフトウェア開発にこのデフォルトの **-std=c++98** モードを使用することを推奨しています。

重要

アプリケーションで C++11 機能を使用するには、上記の ABI 互換性に関する情報を慎重に考慮してください。

上記の C++11 ABI の説明のほかは、Red Hat Developer Toolset では the Red Hat Enterprise Linux Application Compatibility Specification に変更はありません。Red Hat Developer Toolset で構築されたオブジェクトと Red Hat Enterprise Linux v5.x/v6.x ツールチェーン (特に .o/.a ファイル) で構築されたオブジェクトを混ぜると、Red Hat Developer Toolset ツールチェーンがリンケージに使用されます。これにより、Red Hat Developer Toolset のみが提供するより新しいライブラリー機能がリンク時に確実に解決するようになります。

さまざまな長さのベクトルとのシステム上での名前競合を避けるために、SIMD ベクトル型の新たな標準マングル化が追加されました。デフォルトではコンパイラーは旧型のマングル化を使用しますが、新型のマングル化では強力なエイリアスをサポートするターゲットにエイリアスを発信します。**-Wabi** が、旧型マングル化を使用するコードについての警告を表示するようになります。

4.1.1.2.3.2. その他

GCC は、**strlen()**、**strchr()**、**strcpy()**、**strcat()** および **stpcpy()** (これらの **_FORTIFY_SOURCE** バリアントも) などのさまざまな標準 C 文字列関数への呼び出しを、カスタマイズされた、より速いコードにこれらを変換することで最適化します。つまり、これらの関数への呼び出しは、元のソースコードのものよりも少ないもしくは別のものになる可能性があります。この最適化はデフォルトで、**-O2** もしくはそれ以上の最適化レベルで有効になります。**-fno-optimize-strlen** を使用するか、サイズの最適化を実行すると無効になります。

32-bit GNU/Linux でコンパイルを実行し、サイズの最適化を行なっていない場合、**-fomit-frame-pointer** がデフォルトで有効になります。デフォルト設定を事前に選択するには、**-fno-omit-frame-pointer** コマンドラインオプションを使用します。

x86 ターゲット上で厳密な C99 モードでの浮動少数点の計算は、より厳密な標準準拠の GCC でコンパイルされます。このため計算の実行が非常に遅くなる場合があります。これは、**-fexcess-precision=fast** を使って無効にできます。

4.1.1.2.4. デバッグ機能の互換性

GCC は、以前よりも多くのまたはより新しい DWARF 機能を使用する DWARF デバッグ情報を生成します。Red Hat Developer Toolset に含まれる GDB はこれらの機能の取り扱いが可能ですが、7.0 よりも古いバージョンの GDB は処理できません。GCC は、**-gdwarf-2 -gstrict-dwarf** または **-gdwarf-3 -gstrict-dwarf** オプション (後者は GDB 7.0 よりも古いバージョンが一部処理) を使用することで古い

DWARF 機能でのデバッグ情報のみを生成するように制限することができます。

Valgrind、**SystemTap**、またはサードパーティーのデバッガーといった多くのツールは、デバッグ情報を使用します。これらのツールでは、**-gdwarf-2 -gstrict-dwarf** オプションの使用が推奨されます。

注記

デバッグ生成におけるリンク時の最適化は、gcc 4.7 および 4.8 ではまだサポートされていません。このため、**-flto** と **-g** オプションを合わせた使用は、Red Hat Developer Toolset ではサポート対象外となります。

4.1.1.2.5. 他の互換性

GCC はコマンドラインオプションを解析する際により厳密になっており、無効なコマンドラインオプションが使用されると、**gcc** と **g++** がエラーを報告します。特に、リンクするだけでコードをコンパイルしない場合、GCC の以前のバージョンは **--** で始まるすべてのオプションを無視します。たとえば、**--as-needed** や **--export-dynamic** といったリンカーが受け付けるオプションを**gcc** と **g++** はもう受け付けず、**-Wl,--as-needed** もしくは **-Wl,--export-dynamic** が意図されている場合はこれを使ったリンカーに向けられることになります。

新たなリンク時最適化機能 (「リンク時の最適化」 を参照) が理由で、旧型のモジュール間最適化フレームワークのサポートが廃止され、**-combine** コマンドラインオプションは受け付けられなくなりました。

4.2. 分散コンパイル

Red Hat Enterprise Linux は *分散コンパイル* もサポートします。これは、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することです。これらのジョブがマシン群に分配されるので、ビルド時間が早まります (大型のコードベースのプログラムでは特に)。**distcc** パッケージがこの機能を提供します。

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イスを提供し、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イス (例えば、ウィザード) が含まれます。

Autotools プラグインの Red Hat Enterprise Linux 6 バージョンは、**git** や **mercurial** を Eclipse に統合することもしません。このため、**git** リポジトリを使用する 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、ツールや入力ファイル、プロジェクトのビルドのために使用可能なその他の機能についてシステムのテストを行います [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。例えば、**Makefile** は特定のライブラリ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 のドキュメンテーション

Red Hat Enterprise Linux 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テーションを以下の 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テーションは 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て入門書となります。

- 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 インターフェイスに統合します。**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 インターフェイ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、静的ライブラリ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 (例えば、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 (例えば、**.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、バイナリの build-id を表示します。

```
$ eu-readelf -n usr/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以下を入力して *scl-utils* および *scl-utils-build* をインストールします。

```
# yum install -y scl-utils scl-utils-build
```

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 アプリケーションを直接実行する

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 Developer Toolset』 の『Software Collections Guide』 の部分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。例えば、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

Red Hat Enterprise Linux は、実行可能バイナリ、共有ライブラリまたは debuginfo ファイルに ELF を使用します。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** と同等です)。DWARF debuginfo には以下が含まれます。

- バイナリ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

バイナリの状態	コマンド	注意事項
Stripped (削除)	**strip *file*** または **gcc -s -o *file***	共有ライブラリ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。 使用される ELF セクション: **.dynsym**

バイナリの状態	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。 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。 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。 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に依存するすべての 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

デバッガー機能のほとんどは、debuginfo と同等の情報がインストールされていないと、パッケージをデバッグしようとする場合に失敗します。例えば、エクスポートされた共有ライブラリの関数の名前は使用できますが、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 の debuginfo パッケージは Red Hat Network の新規チャネルで利用できるようになりました。パッケージの **-debuginfo** パッケージ (通常は ***packagename*-debuginfo**) をインストールするには、まずマシンを対応する Debuginfo チャネルにサブスクライブさせる必要があります。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**) を使ってコンパイルされています。これは、いくつかの変数が **<optimized out>** として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 /usr/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イメージの表現です。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 が目的に合わない場合のために、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
- クラッシュ時にバイナリでロードされたすべての共有ライブラリ
- 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
usr/bin/sleep /usr/lib/debug/bin/sleep.debug [exe]
0x7fff26fff000+0x1000
1e2a683b7d877576970e4275d41a6aaec280795e@0x7fff26fff340 . - linux-vdso.so.1
0x35e7e00000+0x3b6000
374add1ead31ccb449779bc7ee7877de3377e5ad@0x35e7e00280 /usr/lib64/libc-
2.14.90.so /usr/lib/debug/lib64/libc-2.14.90.so.debug libc.so.6
0x35e7a00000+0x224000
3ed9e61c2b7e707ce244816335776afa2ad0307d@0x35e7a001d8 /usr/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。

- 特定のバイナリがマップされるメモリー内アドレス (例えば、最初の行の **0x400000**)。
- バイナリのサイズ (例えば、最初の行の **+0x207000**)。
- バイナリの 160-bit SHA-1 build-id (例えば、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 (例えば、最初の行の **@0x400284**)。
- ディスク上のバイナリファイル (ある場合)(例えば、最初の行の **usr/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)(例えば、**/usr/lib/debug/bin/sleep.debug**)。ただし、バイナリ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リストに保存される共有ライブラリ名 (例えば、3 行目の **libc.so.6**)。

それぞれの build-id (例えば、 **ab/cdef0123456789012345678901234567890123**) について、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 **/usr/bin/sleep** 実行可能ファイルを例とすると、**coreutils-debuginfo** RPM には、他のファイルのほかに以下が含まれます。

```
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 の一致する情報を表示する必要があります。Red Hat Enterprise Linux RPM パッケージ用に 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
```

例えば、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/')
```

バイナリが RPM にパッケージされておらず、yum リポジトリ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 (例えば、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。例えば、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。例えば、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例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 コマンドは、大規模な内部ドキュメンテーションへのアクセスを提供します。このコマンドは引数を取ります。例えば、**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 **hello** が **hello.c** と同じディレクトリにあることを確認します。

2. **hello** バイナリで **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。例えば、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。

例えば、**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。

例えば、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 ドキュメンテーション

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 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** 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テーションが多数あります。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。この白書には、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から
- 関連するバイナリが含まれるプロジェクトから

複数のバイナリを起動できる場合、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 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。

これまで、**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。例えば、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利用できるドキュメンテーションと共に、「Valgrind のドキュメンテーション」 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。

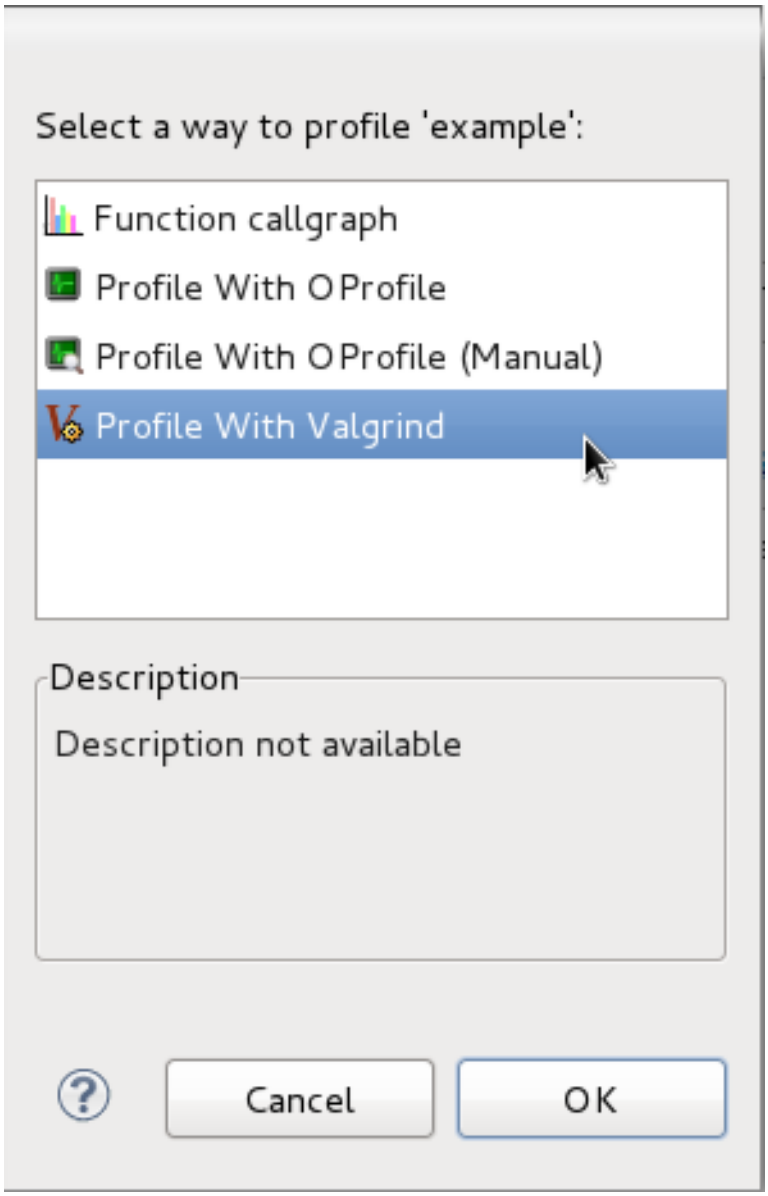


図6.1 Profile As

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。

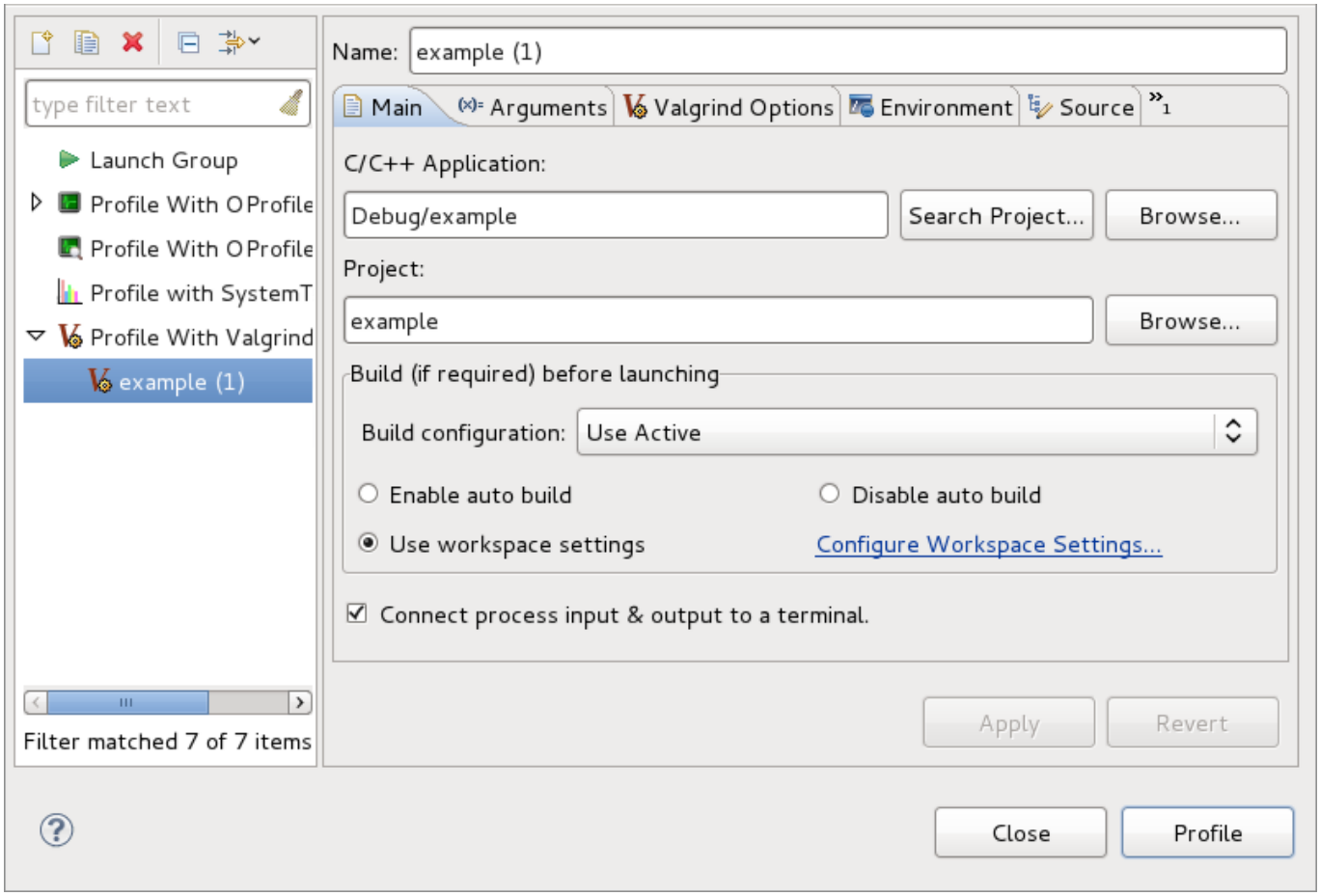


図6.2 Profile Configuration

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 のドキュメンテーション

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、PDF および HTML 形式で利用可能な総合的な 『Valgrind Documentation』 を提供していますが、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 (例えば、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 (例えば、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。

operf

Red Hat Enterprise Linux 7 では初めてとなる **operf** は、Linux Performance Events サブシステムを使用するので、**opcontrol** デーモンを完全に置き換えられます。詳細は、『Red Hat Enterprise Linux 7 System Administrator's Guide』 を参照してください。

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を作成します。このディレクトリは (**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テーションは、「OProfile のドキュメンテーション」 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、ライブラリを使用しているアプリケーションで共有ライブラリ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. Red Hat Enterprise Linux 7 での OProfile

Red Hat Enterprise Linux 7 用には、OProfile 0.9.8 がリリースされています。これはアルファバージョンですが、多くのユーザーが安定性を確認しています。OProfile 0.9.8 リリースでは、特定の個別プロセスのプロファイリングにも使用できるようになっています。

6.2.3.1. 新しい機能

新たな**operf** プログラムが利用可能となっており、これは root 以外のユーザーによる単一プロセスのプロファイリングを可能にします。これを使ってシステム全体のプロファイリングもできますが、その場合は root 権限が必要になります。この機能は、kernel バージョン 2.6.31 またはそれ以降が必要になります。

OProfile は、以下の多くの新プロセッサーもサポートします。

- Tilera tile64
- Tilera tilepro
- Tilera tile-gx
- IBM System z10
- IBM System z196
- Intel Ivy Bridge
- ARMv7 Cortex-A5

- ARMv7 Cortex-A15
- ARMv7 Cortex-A7

6.2.3.2. 以前のリリースとの非互換性

OProfile 0.9.8 は、以前のリリースといくつかの非互換性があります。

- 2.6 kernel より前のサポートがなくなりました。
- 2.6 より前のサポートがなくなったことで、**--with-kernel-support** 設定オプションが不要および無効になっています。
- サンプルヘッダー mtime フィールドが u64 に変更されました。
- **configure.in** ファイルが **configure.ac** に名前変更されています。これは透過的変更です。

6.2.3.3. 既知の問題と制限

OProfile 0.9.8 には以下のような既知の問題と制限があります。

- AMD Instruction Based Sampling (IBS) は現在、新 operf プログラムに対応していません。IBS プロファイリングには **legacy** opcontrol プロファイリングを使用してください。
- **operf** を使って複数イベントをプロファイリングする際は、記録されるイベントの絶対数は通常予測されるものより大幅に少なくなります。これは、Linux kernel の Performance Events Subsystem のバグが原因で、このバグは Linux kernel バージョンの 3.1 と 3.5 の間で修正されました。.
- NMI ウォッチドッグが x86_64 システムで無効になっていないと、**opcontrol** は必要なハードウェアパフォーマンスカウンターの割り当てに失敗する可能性があります。この問題の進行状況は、https://bugzilla.redhat.com/show_bug.cgi?id=683176 で確認できます。
- 多くのアルファ ev67 イベントが機能しません。この問題の進行状況は https://bugzilla.redhat.com/show_bug.cgi?id=931875 で確認できます。

6.2.4. OProfile のドキュメンテーション

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 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テーションで、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 (例えば、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 (例えば、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***

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。

6.3.1. SystemTap 2.0 での DynInst

SystemTap 2.0 では、*DynInst* システムを使用したインストルメンテーション実行の実験段階のサポートが導入されています。DynInst は純粋なユーザー領域バイナリ操作ライブラリーで、プログラムが他の実行中のプログラムを修正できるようになります。高度に効率的なインストルメンテーションまたは他の修正を挿入することで、これを行います。SystemTap 2.0 またはそれ以降では、これをバックエンドとして使用し、スクリプトの制限付きクラスを実行できます。制限を付けることで、インストルメンテーションが速くなり、root アクセスや kernel モジュール操作なしにユーザー領域全体で実行されます。制限は変化していますが、暗号作成や kernel モジュール、特殊グループのメンバーシップに依存する権限なしのユーザープローブのものよりも厳密なものです。

この実験的なバックエンドを使用するには、**--runtime** オプションを stap コマンドラインに追加します

```
$ stap --runtime=stapdyn script.stp -c command
```

スクリプトが DynInst で利用可能な機能以上のものを必要とする場合、SystemTap はアドバイスします。その場合、標準の kernel モジュールベースのバックエンドは、**--runtime** オプションを省略して使用する必要があります。

6.3.2. SystemTap コンパイルサーバー

Red Hat Enterprise Linux 7 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、集中化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。

6.3.3. 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 7 の SystemTap には、SystemTap クライアントに **--unprivileged** という新オプションがあります。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。

6.3.4. SSL および認証管理

Red Hat Enterprise Linux 7 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。

6.3.4.1. 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。

6.3.4.2. 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。

6.3.4.3. 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。

6.3.5. SystemTap のドキュメンテーション

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 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。例えば、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。例えば、**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).
/usr/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。例えば、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 のインターフェイスよりも非常にシンプルなインターフェイ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 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イスを提供します。**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。例えば、**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。

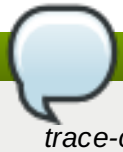
/sys/kernel/debug/tracing/trace_pipe

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 のドキュメンテーション

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**



注記

trace-cmd パッケージは、**ftrace** の有用な代替となるものと同一名のツールを提供します。詳細は、**trace-cmd** man ページを参照してください。

第7章 Red Hat Developer Toolset

Red Hat Developer Toolset は、Red Hat Enterprise Linux プラットフォーム上の開発者用の Red Hat 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、ツールの好きな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 2.1 User Guide』 を参照してください。

7.1. Red Hat Developer Toolset の機能

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、アトミックタイプ、Transactional Memory、最新のコンパイラー最適化、OpenMP 3.1 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	4.3.1	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	4.8.2	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	2.23.52	オブジェクトファイルおよびバイナリを検査、操作するバイナリツールおよび他のユーティリティーのコレクション。
elfutils	0.157	ELF ファイルを検査、操作するバイナリ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。
Git	1.8.4	ピアツーピアアーキテクチャーを備えた分散型バージョン管理システム。
GDB	7.6.1	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。
strace	4.7	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	0.1.4	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	2.1	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。

名前	バージョン	説明
Valgrind	3.8.1	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	0.9.8	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	8.0	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。

7.2. binutils の変更点

Red Hat Developer Toolset 2.0 は **binutils 2.23.52** とともに配布されています。これは、Red Hat Enterprise Linux のシステムバージョンおよび Red Hat Developer Toolset 1.1 に含まれているバージョンにおける多くのバグ修正と機能強化を提供するものです。以下では、今回のリリースにおける新機能の完全なリストを示しています。

The GNU アセンブラー (**as**)、GNU リンカー (**ld**)、および binutils の一部である他のバイナリツールは、GNU 一般公衆利用許諾契約書バージョン 3 の元でリリースされています。

7.2.1. GNU リンカー

既存の GNU リンカーである **ld** に加え、別の ELF リンカー**gold** が利用可能になっています。**gold** は、一時的に **ld** の代わりとなるものなので、**ld** のドキュメンテーションは参照用ドキュメンテーションとなります。**gold** は **ld** のほとんどの機能をサポートしますが、目立つ例外は MRI 互換のリンカースクリプト、相互参照レポート (**--cref**)、および他の種々ののマイナーオプションになります。また、非常に大型の C++ アプリケーションとのリンク時間を大幅に改善します。

Red Hat Developer Toolset 2.0 では、**gold** リンカーはデフォルトでは有効になっていません。ユーザーは **alternatives** メカニズムを使って明示的に**ld** と **gold** を切り替えることができます。

7.2.1.1. 新しい機能

Red Hat Enterprise Linux 6.4 以降の変更点

以下の機能は、Red Hat Enterprise Linux 6.4 に含まれる binutils のリリース以降に追加されたものです。

- **INPUT_SECTION_FLAGS** キーワードがリンカースクリプト言語に新たに追加されました。このキーワードを使うと、セクションのヘッダーフラグで入力セクションを選択することができます。
- **SORT_BY_INIT_PRIORITY** キーワードがリンカースクリプト言語に新たに追加されました。このキーワードを使うと、セクション名にエンコードされている GCC ***init_priority*** 属性の数値でセクションを分類できます。
- **SORT_NONE** キーワードがリンカースクリプト言語に新たに追加されました。このキーワードを使うと、セクションの分類を無効にできます。
- 新たなリンカー提供の記号 **__ehdr_start** が追加されました。ELF 出力を生成する際に、この記号がプログラムのメモリーイメージ内にある ELF ファイルヘッダー (および近くのプログラムヘッダー) を指します。

Red Hat Enterprise Linux 5.9 以降の変更点

以下の機能は、Red Hat Enterprise Linux 5.9 に含まれる binutils のリリース以降に追加されたものです。

- GNU/Linux ターゲットが、ELF 記号バインディングの標準セットへの GNU 拡張である **STB_GNU_UNIQUE** 記号バインディングをサポートするようになりました。このバインディングがダイナミックリンカーに渡されることで、プロセス全体で特定の名前と型を持つ記号が 1 つしか使われていないことを保証します。

注記

この機能の実装は、**glibc** ライブラリーの新しいバージョンにのみある機能に依存しています。そのため、この機能は現在、Red Hat Enterprise Linux 6 の Red Hat Developer Toolset で利用可能です。

- コマンドラインオプション **--no-export-dynamic** が新たに追加されました。このオプションを使うと、**-E** および **--export-dynamic** オプションの効果を無効にできます。
- コマンドラインオプション **--warn-alternate-em** が新たに追加されました。このオプションを使うと、ELF フォーマットオブジェクトファイルが別のマシンコードを使っている場合に警告が表示されます。
- リンカースクリプト機能 **REGION_ALIAS** が新たに追加されました。この機能を使うと、メモリー領域のエイリアス名を作成することができます。
- コマンドラインオプション **-Ttext-segment *address*** が ELF ベースのターゲットに新たに追加されました。このオプションを使うと、テキスト領域の最初のバイトのアドレスを設定することができます。
- リンカースクリプトコマンド **INSERT** が新たに追加されました。このコマンドを使うと、デフォルトスクリプトを拡大することができます。
- リンカースクリプト入力セクションで、***archive*:*file*** 構文を使ってアーカイブ内でファイルを特定することができます。
- **--sort-common** コマンドラインオプションが**ascending** および **descending** をオプション引数として使用することができます。これを使うと、どの分類順序を使用するかを指定することができます。
- コマンドラインオプション **--build-id** が ELF ベースのターゲットに新たに追加されました。このオプションを使うと、ノートセクションに埋め込まれたバイナリあたりの一意の識別子を生成することができます。
- コマンドラインオプション **--default-script=*file_name*** (または **-dT *file_name***) が新たに追加されました。このオプションを使うと、ビルトインリンカースクリプトの代替を指定することができます。
- コマンドラインオプション **-Bsymbolic-functions** が新たに追加されました。このオプションは共有ライブラリー作成時に、共有ライブラリーとの定義がある場合に、グローバル機能記号への参照が定義にバインドされるようにします。
- コマンドラインオプション **--dynamic-list-cpp-new** および **--dynamic-list-data** が新たに追加されました。これらを使うと、動的リストを修正することができます。

7.2.1.2. 互換性の変更点

Red Hat Enterprise Linux 6.4 以降の変更点

以下の互換性の変更点は、Red Hat Enterprise Linux 6.4 に含まれる binutils のリリース以降になされたものです。

- **--copy-dt-needed-entries** コマンドラインオプションは、デフォルトで有効になりません。代わりに **--no-copy-dt-needed-entries** がデフォルトのオプションとなっています。
- リンカースクリプト式の評価は、大幅に改善されました。これは、古い式の評価のドキュメント化されていない動作に依存するスクリプトにマイナスの影響を与える可能性があることに注意してください。

Red Hat Enterprise Linux 5.9 以降の変更点

以下の互換性の変更点は、Red Hat Enterprise Linux 5.9 に含まれる binutils のリリース以降になされたものです。

- **--add-needed** コマンドラインオプションは**--copy-dt-needed-entries** に名前を変更され、**--as-needed** オプションとの混同を防いでいます。
- GNU/Linux システムでは、**STT_GNU_IFUNC** タイプの記号に対してなされた再配置をリンカーが処理しなくなりました。代わりに、ローダーが処理するようリンカーは結果のバイナリに生成します。

注記

この機能の実装は、**glibc** ライブラリーの新しいバージョンにのみある機能に依存しています。そのため、この機能は現在、Red Hat Enterprise Linux 6 の Red Hat Developer Toolset で利用可能です。

- **--as-needed** コマンドラインオプションは、以下の 2 つの場合に動的ライブラリー内でリンクするようになりました。
  - 動的ライブラリーが正規オブジェクトの未定義記号にしたがっている場合。
  - ライブラリーが既にリンクされているライブラリーの **DT_NEEDED** エントリーで見つけられている場合を除き、動的ライブラリーが他の動的ライブラリーの未定義記号にしたがっている場合。
- **-l:*file_name*** コマンドラインオプションが、**.a** または **.so** ファイル拡張子を追加せずに *file_name* という名前のファイル名のライブラリーパスを検索するようになりました。

7.2.2. GNU アセンブラー

7.2.2.1. 新しい機能

Red Hat Enterprise Linux 6.4 以降の変更点

以下の機能は、Red Hat Enterprise Linux 6.4 に含まれる binutils のリリース以降に追加されたものです。

- GNU アセンブラーは、マクロでアンパサンド文字 2 つを必要としなくなりました。
- コマンドラインオプション **--compress-debug-sections** が新たに追加され、再配置可能な出力ファイルの DWARF デバッグ情報の圧縮を可能にしています。圧縮デバッグセクションは現在、**readelf**、**objdump**、および **gold** ツールがサポートしていますが、**ld** はサポートしていません。
- x86 ターゲットの**.bundle_align_mode**、**.bundle_lock**、および **.bundle_unlock** 指示文のサポートが追加されました。

- x86 アーキテクチャーでは、GNU アセンブラーは **rep bsf**、**rep bsr**、および **rep ret** 構文を使うことができます。

Red Hat Enterprise Linux 5.9 以降の変更点

以下の機能は、Red Hat Enterprise Linux 5.9 に含まれる binutils のリリース以降に追加されたものです。

- GNU/Linux ターゲットは、**gnu_unique_object** を **.type** 擬似オペレーションの値としてサポートします。この値を使って、プロセス全体で記号をグローバルで一意のものとしてマークすることができます。
- **.loc** 指示文用の識別子オペランドをともなう DWARF 行テーブル内の新たな識別子のコラムに対するサポートが追加されました。
- **.type** 擬似オペレーションが **STT_GNU_IFUNC** のタイプを受け付けるようになりました。これを使って、記号が再配置のターゲットである場合、この値は使うべきでないことを示すことができます。代わりに、関数が起動され、その結果を値として使います。
- 擬似オペレーション **.cfi_val_encoded_addr** が新たに追加されました。これを使って、ランタイム再配置なしにアンワインドテーブルの一定アドレスを記録することができます。
- コマンドラインオプション **-msse-check=[none|error|warning]** が x86 ターゲット用に新たに追加されました。
- **-a** コマンドラインオプションが**g** を有効なサブオプションとして使用できます。この組み合わせを使って、アセンブリーについての追加情報をともなうアセンブリーリスティングを有効にできます。これには、提供されるコマンドラインオプションやアセンブラーバージョンが含まれます。
- コマンドラインオプション **-msse2avx** が x86 ターゲット用に新たに追加されました。このオプションを使って、VEX 接頭辞のある SSE 命令をエンコードできます。
- x86 ターゲットは、Intel XSAVE、EPT、MOVBE、AES、PCLMUL、および AVX/FMA 命令をサポートします。
- コマンドラインオプション **-march=*cpu*[,+*extension*…]**、**-mtune=*cpu***、**-mmnemonic=[att|intel]**、**-msyntax=[att|intel]**、**-mindex-reg**、**-mnaked-reg**、および **-mold-gcc** が x86 ターゲット用に新たに追加されました。
- 擬似オペレーション **.string16**、**.string32**、および **.string64** が新たに追加されました。これらの擬似オペレーションを使って、ワイド文字を生成することができます。
- i386 は SSE5 命令セットをサポートします。
- 擬似オペレーション **.reloc** が新たに追加されました。この擬似オペレーションは、再配置を作成する低レベルのインターフェースとして機能します。

7.2.3. 他のバイナリツール

7.2.3.1. 新しい機能

Red Hat Developer Toolset 1.1 以降の変更点

以下の機能は、Red Hat Developer Toolset 1.1 に含まれる binutils のリリース以降に追加されたものです。

- **dwp** ユーティリティーのマニュアルページが追加されました。
- バイナリツールは AMD Family 15h プロセッサーのモデル 02h および 10-1fh をサポートします。

Red Hat Enterprise Linux 6.4 以降の変更点

以下の機能は、Red Hat Enterprise Linux 6.4 に含まれる binutils のリリース以降に追加されたものです。

- **readelf** および **objdump** ツールは **.debug.macro** セクションのコンテンツを表示できます。
- コマンドラインオプション **--dwarf-start** および **--dwarf-end** が **readelf** および **objdump** ツールに新たに追加されました。これらのオプションは、新しい Emacs モード (**dwarf-mode.el** ファイルを参照) で使用されます。
- コマンドラインオプション **--interleave-width** が **objcopy** ツールに新たに追加され、入力から出力に幅広いバイトをコピーする **--interleave** の使用が可能になりました。
- コマンドラインオプション **--dyn-syms** が **readelf** ツールに新たに追加されました。このオプションを使って動的記号テーブルをダンプすることができます。
- 新ツール **elfedit** が binutils に追加されました。このツールを使って直接 ELF フォーマットのバイナリを操作することができます。
- 新コマンドラインオプション **--addresses** (または短縮で **-a**) が **addr2line** ツールに追加されました。このオプションを使うと、関数またはソースファイル名の前にアドレスを表示することができます。
- 新コマンドラインオプション **--pretty-print** (または短縮で **-p**) が **addr2line** ツールに追加されました。このオプションを使うとヒューマンリーダブルな出力を作成することができます。
- **dwz -m** 最適化デバッグ情報が追加されました。
- *devtoolset-2-binutils-devel* パッケージは **demangle.h** ヘッダーファイルを提供します。

Red Hat Enterprise Linux 5.9 以降の変更点

以下の機能は、Red Hat Enterprise Linux 5.9 に含まれる binutils のリリース以降に追加されたものです。

- 新コマンドラインオプション **--insn-width=*width*** が **objdump** ツールに追加されました。このオプションを使うと、命令の逆アセンブル時に一行に表示するバイト数を指定できます。
- 新コマンドラインオプション **--relocated-dump=*name*|*number*** が **readelf** ツールに追加されました。このオプションを使うと、再配置されたセクションのコンテンツをバイトのシーケンスとして表示できます。
- 新コマンドラインオプション **--external-symbols-table=*filename*** が **gprof** ツールに追加されました。このオプションを使うと、特定のファイルから記号テーブルを読み込むことができます。
- **bfd** はプラグインターゲットをサポートしており、**gold** が使用する同一の共有オブジェクトをプラグインターゲットがロードすることで新たなファイルフォーマット用に基本的なサポートを得ることができます。
- **objdump** ツールの **--dwarf** (または短縮で **-W** コマンドラインオプションは、**readelf** の **--debug-dump** (または **-w**) オプションと同様の柔軟性を持つようになっています。
- 新コマンドラインオプション **--prefix=*prefix*** および **--prefix-strip=*level*** が **objdump** ツールに追加されました。これらのオプションを使うと、**--source** (または短縮で **-S**) オプションに絶対パスを追加することができます。
- 新コマンドラインオプション **-wL** が **readelf** ツールに追加されました。このオプションを使うと、**.debug_line** セクションのデコード済みのコンテンツをダンプすることができます。
- 「Thin」 アーカイブがサポートされています。このアーカイブはオブジェクトファイルではなく、それらのファイルを指すパス名のみを含みます。

- 新コマンドラインオプション **-F** が **objdump** ツールに追加されました。このオプションを使うと、逆アセンブリーにファイルオフセットを含めることができます。
- 新コマンドラインオプション **-c** が **readelf** ツールに追加されました。このオプションを使うと、アーカイブ記号インデックスの文字列ダンプが可能になります。
- i386 は SSE5 命令セットをサポートします。
- 新コマンドラインオプション **-p** が **readelf** ツールに追加されました。このオプションを使うと、セクションの文字列ダンプが可能になります。

7.2.3.2. 互換性の変更点

Red Hat Enterprise Linux 5.9 以降の変更点

以下の互換性の変更点は、Red Hat Enterprise Linux 5.9 に含まれる binutils のリリース以降になされたものです。

- **--as-needed** コマンドラインオプションは、以下の 2 つの場合に動的ライブラリー内でリンクするようになりました。
  - 動的ライブラリーが正規オブジェクトの未定義記号にしたがっている場合。
  - ライブラリーが既にリンクされているライブラリーの **DT_NEEDED** エントリーで見つけられている場合を除き、動的ライブラリーが他の動的ライブラリーの未定義記号にしたがっている場合。

7.3. プラットフォームの互換性

Red Hat Developer Toolset 2.1 は、Red Hat Enterprise Linux 5 および 6 のみで利用可能で、32 ビットと 64 ビットの Intel および AMD アーキテクチャーの両方に対応しています。

図7.1「Red Hat Developer Toolset 2.1 の互換性に関する表」 は、Red Hat Enterprise Linux の特定バージョン上で Red Hat Developer Toolset を使ってビルドされたバイナリが本システムの他のバージョンで実行される際のサポートを示しています。y 軸では、バイナリがビルドされる Red Hat Enterprise Linux のバージョンが示され、アプリケーションを稼働する Red Hat Enterprise Linux のバージョンと相互参照されています。

図7.1 Red Hat Developer Toolset 2.1 の互換性に関する表

7.4. Red Hat Developer Toolset の参考資料

Red Hat Developer Toolset 2.1 およびそのオファリングについての詳細情報は、以下のリソースを参照してください。

- Red Hat Developer Toolset 2.1 Release Notes — Red Hat Developer Toolset 2.1 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、もしくは製品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 Developer Toolset 2.1 User Guide — Red Hat Developer Toolset 2.1 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

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**、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、実行したい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、本番使用を目的としています。

8.1. Red Hat Software Collections の機能

Red Hat Software Collections 1.0 は、表8.1「Red Hat Software Collections 1.0 コンポーネント」に記載のツールの最新バージョンを提供します。

表8.1 Red Hat Software Collections 1.0 コンポーネント

コンポーネント	Software Collection	説明
Perl 5.16.3	*perl516*	最新の安定した Perl のリリースで、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。
PHP 5.4.14	*php54*	**PEAR 1.9.4** および多くの追加ユーティリティーを含む最新の安定した PHP リリースです。このバージョンでは、新たな言語構文、コマンドライン用のビルトインウェブサーバー、パフォーマンス改善などが含まれます。
Python 2.7	*python27*	**MySQL と PostgreSQL 用のデータベースコネクター**および多くの追加ユーティリティーがある最新の安定した Python リリースです。このバージョンでは、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。
Python 3.3	*python33*	**PostgreSQL 用のデータベースコネクター**および多くの追加ユーティリティーがある最新の安定した Python 3 リリースです。この Software Collection は、Red Hat Enterprise Linux 6 での開発者に Python 3 へのアクセスを提供し、このバージョンとのアプリケーションの互換性テストを可能にします。

コンポーネント	Software Collection	説明
Ruby 1.9.3	*ruby193*	**Rails 3.2.8** および **大型の Ruby gems コレクション** のある安定した Ruby の最新リリースです。この Software Collection は Red Hat Enterprise Linux 6 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などを含む多くの新機能および機能強化を提供します。
MariaDB 5.5	*mariadb55*	安定した最新リリースの MariaDB です。この Software Collection は、Red Hat Enterprise Linux 6 のユーザーに MySQL の代替を提供します。これは MariaDB とバイナリ互換で、データ変換なしにこれを置き換えることができます。
MySQL 5.5	*mysql55*	最新の安定した MySQL のリリースです。このバージョンでは、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	*postgresql92*	最新の安定した PostgreSQL リリースです。このバージョンでは、ネイティブ JSON サポートや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 [a]	*nodejs010*	安定した最新リリースの Node.js です。この Software Collection は、Red Hat Enterprise Linux 6 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。

[a] Red Hat Software Collections 1.0、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。

8.2. プラットフォームの互換性

Red Hat Software Collections 1.0 は、以下のバージョンの AMD64 および Intel 64 アーキテクチャーの Red Hat Enterprise Linux システムで利用できます。

- Red Hat Enterprise Linux 6.2 Extended Update Support
- Red Hat Enterprise Linux 6.3 Extended Update Support
- Red Hat Enterprise Linux 6.4

8.3. Red Hat Software Collections の使用

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 を使用する Software Collections を空白で区切った一覧で置き換え、*command* を実行するコマンドで置き換えます。たとえば、*perl516* Software Collection から **hello.pl** というファイル名に保存されている 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 の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シェル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 を使用する Software Collections を空白で区切った一覧で置き換えます。たとえば、デフォルトとして新たなシェルセッションを *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するようにします。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 の完全なリストについては、表 8.1「Red Hat Software Collections 1.0 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 1.0 Release Notes』 を参照してください。

8.4. Red Hat Software Collections を使用するアプリケーションの導入

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、一般的に以下のどちら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 Collection 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 Ha Software Collections 1.0 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。

8.5. その他の情報

- Red Hat Software Collections 1.0 Release Notes — Red Hat Software Collections 1.0 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知りたい場合、もしくは製品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。

第9章 ドキュメンテーションツール

Red Hat Enterprise Linux 6 には、プロジェクトにドキュメンテーションを含めるためのツールが 2 つあります。Publican と Doxygen です。

9.1. Publican

Publican は DocBook XML でドキュメンテーションを発行、処理するプログラムです。ブックを発行するプロセスで、XML が有効かつ発行可能な基準にあることを確認します。新規作成されたアプリケーションについてのドキュメンテーションの発行には、特に便利です。

9.1.1. コマンド

Publican には利用可能な多くのコマンドとアクションがあり、これらすべては **--help** もしくは **--man** ページで見られます。一般的なものを以下に挙げます。

build

XML ファイルをドキュメンテーションに適合するフォーマット (PDF、HTML、HTML-single など) に変換します。

create

「ファイル」 で説明されている必要なファイルすべてを含むあらたなブックを作成します。

create_brand

「ブランド」 で説明されているように、新たなブランドを作成し、すべてのブックの外観を揃えます。

package

ブックのファイルを RPM 配布可能にパッケージ化します。

9.1.2. 新規ドキュメントの作成

すべての必要なファイルを含む新規ドキュメントを作成するには、**publican create** コマンドを使います。

publican create には、以下のような多くのオプションがあります。

--help

publican create コマンドで使用できるオプション一覧を表示します。

--name *Doc_Name*

ブックの名前を設定します。タイトルには空白を入れないように注意してください。

--lang *Language_Code*

このオプションを設定しない場合は、デフォルトで en-US となります。**--lang** オプションは、**publican.cfg** ファイル内の **xml_lang** を設定し、ドキュメントディレクトリ内でこの名前を使ってディレクトリを作成します。

--version *version*

ブックについての製品バージョン番号を設定します。

--product *Product_Name*

ブックについての製品名を設定します。この名前には空白を入れないよう注意してください。

--brand *brand*

ドキュメントの見た目の一貫性を保つために使うブランド名を設定します。

他のオプションについては **--help** を参照してください。

publican create を実行する前に、ブックを作成するディレクトリに切り替えることを忘れないでください。こうすることで、ファイルやディレクトリがユーザーのホームディレクトリに追加されなくなります。

9.1.3. ファイル

ブックが作成されると、多くのファイルがブックのディレクトリ内に作成されます。これらのファイルは、ブックが正常にビルドされるために必要なもので、削除してはいけません。しかし、(各章などの) リンクが機能するにはこれらを編集すると共に、著者やタイトルに関する正確な情報を含める必要もあります。

publican.cfg

このファイルはビルドオプションを設定し、常に ***xml_lang*** (ブックの言語、例えば en-US)、***type*** (ドキュメントのタイプ、例えば book または set)、***brand*** (「ブランド」にある、ドキュメントが使用するブランディング、例えば Red Hat) の各パラメーターが含まれます。オプションのパラメーターは多くありますが、これらは後で翻訳の分野などで問題を引き起こす可能性もあるので、使用する際には注意が必要です。これらの高度なパラメーター一覧は、Publican User Guide にあります。**publican.cfg** ファイルは、初期作成時のものから編集されることはほとんどありません。

book_info.xml

このファイルはブックのテンプレートで、タイトルやサブタイトル、著者、発行番号、ブックの ID 番号などの情報が含まれています。また、注意や警告、基本的なスタイルガイドに関する情報と発行物の最初に印刷される基本的な Publican 情報も含まれます。ブックが更新される際には発行番号を増やす必要があるので、このファイルは頻繁に編集されます。

Author_Group.xml

このファイルは、著者および投稿者に関する情報を保存するために使われます。一旦設定されると、著者に関する変更がない限り、編集されることは滅多にありません。

Chapter.xml

このファイルは、実際のコンテンツがどのようなものかを示す例です。プレースホルダーとして作成されますが、(下記の) ***Doc_Name.xml*** でリンクされていなければ、実際のブックには表示されません。発行するコンテンツを書く際に、新規 XML ファイルが作成され、適切に名前が付けられ (例えば、ch-publican.xml)、**Doc_Name.xml** 内にリンクされます。ブックがビルドされると、このファイルのコンテンツがブックのコンテンツを形成します。この特定ファイルが編集される可能性は低いですが、同様のファイルは継続的に変更、更新、追加、削除などがされるので、編集されることになります。

***Doc_Name*.xml**

このファイルは発行物のコンテンツページで、ブックに含まれる各章へのリンク一覧が含まれます。実際には「Doc_Name」とは呼ばれず、発行物のタイトル (例えば、Developer_Guide.xml) になります。このファイルは、新しい章が追加されるか、削除または整

理し直した場合にのみ、編集されます。このファイルは、***Doc_Name*.ent** と同じである必要があり、そうでない場合は、ブックがビルドされません。

***Doc_Name*.ent**

このファイルには、ローカルエンティティの一覧が含まれます。デフォルトでは、***YEAR*** は現在の年に設定され、***HOLDER*** には著作権所有者の名前をそこに表示するというメモが含まれます。***Doc_Name*.xml** と同様に、このファイルは「Doc_Name」とは呼ばれず、ドキュメントのタイトル (例えば、Developer_Guide.ent) になります。発行の最初の 1 回か、著作権所有者が変更された場合にのみ編集される可能性があります。***Doc_Name*.xml** と同じになっている必要があり、そうでない場合はブックがビルドされません。

Revision_History.xml

publican package の実行時に、**<revhistory>** タグを含む最初の **XML** ファイルを使って RPM 改訂履歴がビルドされます。

9.1.3.1. ドキュメンテーションにメディアを追加する

説明されている内容を例証するために、様々なメディアを追加する必要がある場合もあります。

イメージ

images フォルダは、publican がドキュメントのディレクトリ内に作成します。ドキュメントで使われるイメージはすべてここに保存します。そして、このドキュメントにイメージを追加する際は、**images** ディレクトリ内にイメージにリンクさせます (例えば、**./images/image1.png**)。

コードサンプル

時間が経過し技術が変化するにつれ、プロジェクトのドキュメンテーションはコード内の差異を反映するために更新が必要になります。これを容易にするには、好みのエディターで各コードサンプルに個別のファイルを作成し、ドキュメントのディレクトリ内の **extras** と呼ばれるフォルダにこれを保存します。ドキュメントにコードサンプルを挿入する際に、ファイルとファイルが入っているフォルダにリンク付けします。この方法だと、数カ所で使われている例が一度で更新でき、変更する特定のアイテムをドキュメント全体で探すのではなく、コードサンプルが一箇所で見つかるので、時間と労力が節約できます。

任意のファイル

ドキュメンテーションに含まれていないファイルを RPM (例えば、ビデオによる説明) でバンドルする必要がある場合もあります。これらのファイルを発行物のディレクトリ内にある **files** と呼ばれるディレクトリに加えることで、ブックがコンパイルされる際にファイルが RPM に追加されます。

これらのファイルにリンクを貼るには、以下の XML を使用します。

```
<xi:include parse="text" href="extras/fork/fork1.c"
xmlns:xi="http://www.w3.org/2001/XInclude" />
```

9.1.4. ドキュメントのビルド

root ディレクトリでまず最初に **publican build --formats=*chosen_format* --langs=*chosen_language*** を入力してテストビルドを実行し、すべての XML が正しく受け入れられるかを確かめます。例えば、US English のドキュメントを single HTML ページでビルドするには、**publican build --formats=html-single --langs=en-US** を実行します。エラーがなければ、ブックは root ディレクトリにビルドされ、そこで必要な仕上がりとなっているかを確認できます。トラブルシューティングをできるだけ簡単なものとするため、この確認を定期的に行うことが推奨されます。

注記

XML コード内のバグをテストするためにビルドを行う際は、**--novalid** オプションが便利な時もあります。これは、まだ存在しないファイルやドキュメントのセクションを指している相互参照やリンクをスキップします。その代わりに、クエスチョンマーク 3 つ (???) で示されます。

ドキュメントは以下に挙げる多くの異なる形式で発行できます。

html

通常の HTML ページで、次の章やセクションへのリンクがあります。

html-single

1 ページの長い HTML ページで、異なる章やセクションへのリンクはページのトップにあり、新たなページではなくページ下部に移動します。

html-desktop

1 ページの長い HTML ページで、異なる章やセクションへのリンクはドキュメントの左側のパネルに表示され、新たなページではなくページ下部に移動します。

man

Linux、UNIX、その他の同様のオペレーティングシステムシステムの man ページです。

pdf

PDF ファイルです。

test

実際に表示されるファイルを作成することなく XML を検証します。

txt

単一のテキストファイルです。

epub

EPUB 形式の電子書籍です。

eclipse

Eclipse ヘルプのプラグインです。

9.1.5. 発行物のパッケージ

ドキュメンテーションが完成し、エラーなしでビルドができたら、**publican package --lang=*chosen_language*** を実行します。これで SRPM パッケージがドキュメントのディレクトリの **tmp/rpm** に出力され、バイナリ RPM パッケージがドキュメントのディレクトリの**tmp/rpm/noarch** に行きます。デフォルトでは、これらのパッケージの名前は ***productname-title-productnumber-[web]-language-edition-pubsnumber.[build_target].noarch.file_extension*** となり、各セクションの情報は **publican.cfg** からのものとなります。

9.1.6. ブランド

ブランドは、ロゴやイメージ、配色の一致という点で多くのドキュメントの種類にわたって外観で一貫性のレベルを作成するという点で、テンプレートと同様の使われ方をします。同一アプリケーションやアプリケーションの同一バンドルのブックを複数作成する場合は、これが特に有用となります。

新規ブランドの作成には、名前と言語が必要になります。**publican create_brand --name=*brand* --lang=*language_code*** を実行します。これで**publican-brand** というフォルダが作成され、発行物のディレクトリ内に置かれます。このフォルダには、以下のファイルが含まれます。

COPYING

SRPM パッケージの一部で、著作権ライセンスおよび詳細が含まれます。

defaults.cfg

publican.cfg で設定可能なパラメーターのデフォルト値を提供します。このファイルからの仕様は、**publican.cfg** ファイル内の仕様が適用される前に最初に適用されます。つまり、**publican.cfg** 内の値は **defaults.cfg** ファイルの値を上書きします。これはドキュメント内で定期的に使用される側面に使用することが最善ですが、ライターは設定を変更することもできます。

overrides.cfg

これも **publican-brand.spec** で設定可能なパラメーターの値を提供します。このファイルからの仕様は最後に適用されるので、**defaults.cfg** と **publican.cfg** の両方を上書きします。ライターによる変更が許可されていない側面に使用されることが最善です。

publican.cfg

バージョンやリリース番号、ブランド名といった基本的な情報を設定する点で、**publican.cfg** ファイルと同様のファイルです。

publican-brand.spec

このファイルは、RPM Package Manager が発行物を RPM にパッケージ化する際に使用します。

README

SRPM パッケージの一部で、パッケージの概説を提供します。

言語コードの名前がつけられたサブディレクトリもこのディレクトリ内に置かれ、以下のファイルが含まれます。

Feedback.xml

これはデフォルトで生成され、閲覧者がフィードバックを残せるようにします。カスタマイズして、関連の連絡先詳細やバグ報告プロセスを含めることができます。

Legal_Notice.xml:

著作権情報が含まれます。選択された著作権ライセンスの詳細を編集・変更できます。

このディレクトリには、あと 2 つのサブディレクトリがあります。**images** サブディレクトリには、ラスター (PNG) とベクター (SVG) 形式の両方のイメージ数が含まれており、イメージの置換で変更可能な様々なナビゲーションアイコンのプレースホルダとして機能します。**css** フォルダには **overrides.css** が含まれ、これはブランドの視覚スタイルを設定し、**common.css** のものを上書きします。

配布可能な新規ブランドをパッケージ化するには、**publican package** コマンドを使用します。デフォルトでは、このコマンドは *ソース RPM パッケージ*(SRPM パッケージ) を作成しますが、**--binary** オプションを使用して *バイナリ RPM パッケージ* を作成することもできます。パッケージ名は、***publican-brand-version-release.[build_target].*[noarch]*.file_extension*** とな

り、**publican.cfg** ファイルから取られた必須パラメーターが付けられます。

注記

SRPM パッケージには .src.rpm 拡張子が付き、バイナリ RPM パッケージには .rpm 拡張子が付きます。

バイナリ RPM パッケージには、**[build_target].noarch** に拡張子の付いたファイルが含まれ、この **[build_target]** は **publican.cfg** ファイルの **os_ver** パラメーターで設定されているパッケージがビルドされる対象のオペレーティングシステムとバージョンを示しています。noarch 要素は、このパッケージがシステムアーキテクチャに関係なく、どのシステム上にでもインストールできることを明記しています。

9.1.7. Web サイトのビルド

Publican は、ドキュメンテーションを管理するための Web サイトをビルドすることもできます。これは、1 人でドキュメンテーションを維持しているものの、Web サーバー上にインストールする Publican が生成可能な PRM パッケージのドキュメンテーションをチームで作業している場合に特に便利なものです。作成される Web サイトは、ホームページ、製品およびバージョン情報のページ、ドキュメンテーションのページで構成されます。発行物の root ディレクトリに Publican は、設定ファイル、SQLite データベースファイル、サブディレクトリ 2 つを作成します。言語ごとに新規サブディレクトリが作成されるので、ドキュメンテーションが発行される言語数によっては多くの設定ファイルが存在することになります。

詳細情報は 「ドキュメンテーション」 を参照してください。

9.1.8. ドキュメンテーション

Publican には端末からアクセス可能な **--man**、**--help**、**--help_actions** の各種の包括的ページがあります。

利用可能な異なるタグを含む XML についての情報は、Norman Walsh および Leonard Muellner 著の DocBook ガイドである 『DocBook: the definitive guide』 を参照してください。http://www.docbook.org/tdg/en/html/docbook にあります。タグの全一覧とそれらの簡単な使用方法については、 『Part II: Reference』 を参照してください。

また、包括的な Publican ユーザーガイドもオンラインで http://jfearn.fedorapeople.org/en-US/index.html にアクセスするか、**yum install publican-doc** でローカルにインストールすることができます。

9.2. 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テーションツールです。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テーション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。

9.2.1. 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

9.2.2. 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。ただし、編集の必要がある場合、すべての利用可能なタグの詳細な説明について、Doxygen ドキュメンテーション Web サイト 『configuration page』 を参照してください。また、**doxywizard** と呼ばれる GUI インターフェイスもあります。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テーションは Doxygen ドキュメンテーション 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やツリーで構成されている場合は、root ディレクトリ (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 (例えば、***.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。例えば、ソースツリーからすべての **test** ディレクトリを省略するには、**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テーション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テーション内に含めることもできます。

9.2.3. Doxygen の実行

doxygen *config-file* を実行すると、**html**、**rtf**、**latex**、**xml**、および/または **man** ディレクトリが doxygen が開始されたディレクトリ内に作成され、対応するファイルタイプのドキュメンテーションが含まれます。

HTML OUTPUT

このドキュメンテーションは、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 ブラウザで閲覧することができます。ブラウザ (例えば、Mozilla、Safari、Konqueror、Internet Explorer 6) を **html** の **index.html** に向けます。

LaTeX OUTPUT

Doxygen は、**Makefile** を **latex** ディレクトリに書き込み、Latex ドキュメンテーション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

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 ディレクトリからのドキュメンテーションは、**manpath** の man path に正確な man ディレクトリ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。

9.2.4. 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、ドキュメンテーション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テーションを作成できます。
2. このドキュメンテーションを作成するには 2 つの方法があります。

   ***特別な* ドキュメンテーションブロック**

   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テーション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 ...
      */
     ```

- ドキュメンテーション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。

ドキュメンテーション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テーション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テーション

文書化しているコードの前にドキュメンテーション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テーション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テーション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' のドキュメンテーション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**: タイプ定義を文書化します。
- **\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 インターフェイスを文書化します。

3. 次に、特別なドキュメンテーションブロックを HTML および / Latex 出力ディレクトリに書き込む前に、解析します。以下の手順が含まれます。
    a. 特別なコマンドを実行します。
    b. 空白およびアスタリスク (*) をすべて削除します。
    c. 空白の行を新たな段落として取り扱います。
    d. 単語が対応するドキュメンテーション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。
    e. 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 ドキュメンテーション Web サイトの 『automatic link generation page』 にあります。
    f. ドキュメンテーションが 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 ドキュメンテーション Web サイトの 『HTML commands page』 にあります。

9.2.5. 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** インターフェイ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 1-2.3 **Fri Aug 21 2015** **Chester Cheng**
ソート番号を更新し、ドキュメントを再ビルドします。

改訂 1-2 **Wed Jun 11 2014** **Jacquelynn East**
リンク切れを修正

改訂 1-0 **Mon Jun 2 2014** **Jacquelynn East**
7.0 GA リリース向けバージョン

改訂 0-33 **Wed May 21 2014** **Jacquelynn East**
oprofile セクションのバグに URL を追加 (BZ#795987)

改訂 0-32 **Mon Apr 12 2014** **Jacquelynn East**
スタイル変更のための再ビルド

改訂 0-21 **Tue Apr 8 2014** **Jacquelynn East**
java セクションに新機能を追加 (BZ#795590)
テクノロジープレビューのステータスを反映するよう btrfs の重要な注記を編集 (BZ#1025572)

改訂 0-20 **Mon Apr 7 2014** **Jacquelynn East**
最新の JDK バージョンで RHEL7 が常に更新されている文を追加 (BZ#911433)
SystemTap の章を更新 (BZ#795980)
GCC および DTS についてのセクションを更新 (BZ#1032615)
GCC パワーデフォルトを追加 (BZ#1009827)

改訂 0-18 **Fri Apr 4 2014** **Laura Bailey**
operf コマンドを OProfile セクションに追加およびマイナー修正 (BZ#1048636)

改訂 0-17 **Thu Apr 3 2014** **Jacquelynn East**
libStorageMgmt ライブラリー用のプラグインの作成に関するセクションを追加 (BZ#795222)

改訂 0-16 **Fri Mar 14 2014** **Jacquelynn East**
DTS の章を更新 (BZ#1026417)

改訂 0-15 **Mon Feb 3 2014** **Jacquelynn East**
perl セクションで若干の編集 (BZ#795583)

改訂 0-13 **Fri Jan 10 2014** **Jacquelynn East**
oProfile セクションを更新 (BZ#795987)

改訂 0-12 **Wed Nov 27 2013** **Jacquelynn East**
python セクションを Python 2.7 に更新 (BZ#795585)
perl セクションを Perl 5.16 に更新 (BZ#795583)
ruby セクションを Ruby 1.9 に更新 (BZ#795629 および BZ#795631)

改訂 0-9 **Wed Nov 27 2013** **Jacquelynn East**
ベータ前のビルド

改訂 0-7 **Wed Oct 2 2013** **Jacquelynn East**

/bin、/sbin、/lib、/lib64 への参照を /usr 下にネスト化するよう変更 (BZ#952479)

改訂 0-6 **Mon Sep 23 2013** **Jacquelynn East**

memcpy 失敗の可能性に関する重要な注記を追加 (BZ#836490)

改訂 0-5 **Wed Sep 11 2013** **Jacquelynn East**

Red Hat Enterprise Linux 7.0 用にブランチ

索引

シンボル

ドキュメンテーション

- Autotools
  - コンパイルおよびビルド,Autotools のドキュメンテーション
- Boost
  - ライブラリおよびランタイムサポート,Boost ドキュメンテーション
- Doxygen, Doxygen
  - Doxygen の実行, Doxygen の実行
  - ソースの文書化, ソースの文書化
  - リソース, リソース
  - 使用開始, 使用開始
  - 対応の出力および言語, Doxygen 対応の出力および言語
- GNU C++ 標準ライブラリ
  - ライブラリおよびランタイムのサポート,GNU C++ 標準ライブラリドキュメンテーション
- GNU デバッガー, GDB ドキュメンテーション
- Java
  - ライブラリおよびランタイムサポート,Java ドキュメンテーション
- KDE 開発フレームワーク
  - ライブラリおよびランタイムサポート,kdelibs ドキュメンテーション
- OProfile
  - プロファイル, OProfile のドキュメンテーション
- Perl
  - ライブラリおよびランタイムサポート,Perl のドキュメンテーション
- Python
  - ライブラリおよびランタイムサポート,Python ドキュメンテーション
- Python pretty-printer
  - デバッグ, Python Pretty-Printer
- Qt
  - ライブラリおよびランタイムサポート,Qt ライブラリドキュメンテーション
- Ruby
  - ライブラリおよびランタイムサポート,Ruby ドキュメンテーション
- SystemTap
  - プロファイル, SystemTap のドキュメンテーション
- Valgrind
  - プロファイル, Valgrind のドキュメンテーション
- プロファイル
  - ftrace, ftrace のドキュメンテーション

ドキュメンテーションツール, ドキュメンテーションツール

- Publican, Publican
  - Publican のドキュメンテーション, ドキュメンテーション
  - Web サイトのビルド,Web サイトのビルド
  - コマンド, コマンド
  - ドキュメンテーションにメディアを追加する, ドキュメンテーションにメディアを追加する