

CodeWarrior®

IDE User Guide



CodeWarrior に関する最新のテクニカル情報については CodeWarriorCD の Release Notes フォルダをご覧ください。

Revised: 3/23/00 mds-JP000323

Metrowerks CodeWarrior © Copyright 1993-2000 by Metrowerks Inc. and its Licensors. All rights reserved.

お客様は、本 CD に記録されている文書を個人使用目的に限り、プリントすることができます。この場合を除いて、Metrowerks Inc. からの書面による承諾なしに、本 CD に記録されている文書の全部、または、一部をいかなる形態、方法（電子的、物理的な複製、または、写真複写、録音録画、その他すべての情報記録、再生システムを含む）により、複製または伝達することを禁じます。

Metrowerks の名称、ロゴ、CodeWarrior、PowerPlant、Metrowerks University は、Metrowerks Inc. の登録商標です。

CodeWarrior Constructor、Geekware、PowerParts、Discover Programming は、Metrowerks Inc. の商標です。

記載の商標および登録商標は、各社が保有します。

CD に記録されているすべてのソフトウェアおよび文書は、CodeWarrior QuickStart の巻末に記述されているライセンス契約が適用されます。

連絡先：

日本	メトロワークス株式会社 150-0042 東京都渋谷区宇田川町 36-6 ワールド宇田川ビル 8F TEL : (03) 3780-6091 FAX : (03) 3780-6092
U.S.A.	Metrowerks Corporation 9801 Metric Boulevard, Suite 100 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
WWW サーバ	http://www.metrowerks.co.jp/ http://www.metrowerks.com/
ユーザー登録	j-register@metrowerks.com
テクニカルサポート	j-support@metrowerks.com
購入 / 契約更新	j-sales@metrowerks.com
インフォメーション	j-info@metrowerks.com

目次

第 1 章 紹介	15
CodeWarrior IDE について	15
IDE User Guide の概要	16
リリースノートについて	17
マニュアルの表記規則	17
表記について	17
ホストについて	18
新機能	18
RAD ツール (Rapid Application Development Tool)	18
新規コマンド	19
IDE のカスタマイズ	19
キーバインディングの改良	19
XML プロジェクトのインポートとエクスポート	19
デバッガの自動変数表示	19
ユーザー定義アクセスパス	19
プロジェクトウィンドウの改良	20
さらに知識を深めるために	20
QuickStart とチュートリアル	20
ターゲットマニュアル	20

第 2 章 はじめに	23
システムの必要条件	23
Windows の必要条件	23
CodeWarrior IDE のインストール	24
プログラミングの概念	24
入力ファイルの作成	24
ソフトウェアの生成	26
デバッグと修正	26
CodeWarrior IDE の紹介	27
プロジェクトとターゲット	27
ソースコードの編集とブラウズ	27
コンパイルとリンク	28
プロジェクトのデバッグ	28
RAD ツール	28
バージョン管理	29
スクリプト	29
IDE のカスタマイズ	29
サードパーティツールの使用	29

第 3 章 プロジェクト	31
プロジェクトウィンドウの解説	32
プロジェクトウィンドウのナビゲート	33
プロジェクトウィンドウのツールバー	33
ファイルビュー	33
デザインビュー	38
リンク順ビュー	39
ターゲットビュー	39
新規プロジェクトを作成	40
プロジェクトファイルの種類	40
プロジェクトステーションリを選択	40
新規プロジェクトの名前	42
新規プロジェクトダイアログを使う	43
新規プロジェクトの修正	44
新規プロジェクトのビルド	44
プロジェクトステーションリを使う	44
プロジェクトステーションリについて	44
プロジェクトステーションリのフォルダについて	45
独自のプロジェクトステーションリの作成	45
プロジェクトヘメイクファイルをインポート	47
Makefile インポータウィザードを使う	47
Makefile インポータのオプション	48
既存のプロジェクトを開く	49
開くコマンドを使う	50
過去のファイルを開くコマンドを使う	50
プロジェクトウィンドウでサブプロジェクトを開く	50
他のプラットフォームで作成したプロジェクトファイルを開く	50
古いバージョンのプロジェクトファイルを開く	51
プロジェクトを保存	52
プロジェクトと共に保存される項目	52
プロジェクトのコピーを保存	52
プロジェクトを閉じる	52
デフォルトのプロジェクトを選択	53
プロジェクト内のファイル管理	53
グループの拡張と縮小	54
ファイルとグループを選択	55
ファイルを追加	56
ファイルやグループを移動	61
グループの作成	61
ファイルやグループの削除	61
グループ名の変更	62
ファイルのタッチ / アンタッチ	63

複合プロジェクトを作成	64
ビルドターゲットとは	64
サブプロジェクトとは	65
デザインとは	66
複合プロジェクトの作成方法	66
新しいビルドターゲットを作成	66
ビルドターゲット名の変更	67
ビルドターゲット設定の変更	68
現在のビルドターゲットを設定	68
ターゲットの依存関係を作成	68
ターゲットにファイルを割り当てる	69
プロジェクト内にサブプロジェクトを作成	70
プロジェクト情報を検証	71
プロジェクトを移動	72
プロジェクトのインポートとエクスポート	73
プロジェクトでデバッグを制御	73
プロジェクトのデバッグをアクティブにする	74
ファイルのデバッグをアクティブにする	74
プリプロセッサシンボルをプロジェクトに追加	75

第 4 章 ファイル	77
新規ファイルを作成	77
既存ファイルを開く	77
ファイルメニューからファイルを開く	78
プロジェクトウィンドウからファイルを開く	79
エディタウィンドウからファイルを開く	80
関連するファイルを開く	82
ファイルを保存	82
ファイルを閉じる	85
1 つのファイルを閉じる	85
すべてのファイルを閉じる	85
ファイルを印刷	86
印刷オプションを設定	86
ウィンドウを印刷	86
直前に保存したファイルに戻す	87
ファイル/フォルダの比較とマージ	87
ファイル比較とマージについて	88
比較ファイルを選択	90
差分の検査と適用	91
比較フォルダを選択	92

第 5 章 ソースコードの編集	95
エディタウィンドウの解説	95
テキスト編集域	97
ヘッダポップアップメニュー	97
関数ポップアップメニュー	98
マーカポップアップメニュー	98
オプションポップアップメニュー	99
VCS ポップアップメニュー	99
ファイルパス表示欄	100
ファイル変更マーカ	100
欄分割コントロール	100
行番号	100
ツールバー拡張ボタン	101
エディタウィンドウの設定	101
文字サイズとフォントを指定	101
ウィンドウコントロールを表示	101
ウィンドウを欄に分割	102
エディタウィンドウの設定を保存	104
基本的なテキスト編集	104
ウィンドウナビゲーションの基本	104
テキストを追加	105
テキストを削除	105
テキストを選択	106
テキストを移動 (ドラッグ & ドロップ)	107
カット、コピー、ペースト、消去	107
括弧のバランスをとる	107
テキストを左右に移動	108
変更を取り消す	108
カラーをコントロール	109
テキストをナビゲート	109
関数を検索	110
マーカを追加、削除、選択	110
関連するファイルを開く	112
特定の行にジャンプ	112
戻ると次へ	113
エディタのコマンドを設定	113
オンラインリファレンス	113
シンボル定義の検索	113
WinHelp	114

第 6 章 テキストの検索と置換	115
検索ダイアログの解説	115
検索と置換	115
複数ファイル検索機能	120
選択したテキストを検索	123
1 つのファイルでのテキスト検索と置換	125
テキストを検索	125
検索範囲をコントロール	126
検索パラメータをコントロール	127
特殊文字を検索	127
検索したテキストを置換	127
バッチ検索	129
複数のファイルでのテキスト検索と置換	130
複数ファイルの検索を行う	131
検索するファイルを選択	131
ファイルセットを保存	134
ファイルセットを削除	134
複数ファイルでの検索範囲をコントロール	135
正規表現 (grep) を使う	136
特殊なオペレータ	137
正規表現を使う	138
 第 7 章 ソースコードのブラウズ	 143
ブラウザを利用	143
ブラウザの概要	144
コンテンツビュー	144
ブラウザビュー	145
階層ビュー	146
ブラウザの解説	147
ブラウザのコンテキストメニュー	148
ブラウザコンテンツ	149
ブラウザウィンドウ	150
クラス階層ウィンドウ	156
単一クラス階層ウィンドウ	158
シンボルウィンドウ	159
ブラウザメニュー	161
ブラウザウィザード	162
新規クラスウィザード	162
新規メンバー関数ウィザード	167
新規データメンバーウィザード	171
ブラウザを使う	173

ブラウザのオプションを設定	174
ブラウザデータベースでシンボルを識別	174
ブラウザ内のコードをナビゲート	175
サブプロジェクトをブラウズ	176
シンボルを補完	176
ソースファイルを開く	177
宣言を表示	177
関数定義を表示	177
ブラウザ内でコードを編集	177
継承を分析	178
オーバーライドされた関数を検索	178
MFC、PowerPlant クラスを表示	178
デフォルトのブラウザを保存	179
第 8 章 IDE のオプション設定	181
IDE 設定パネルの解説	181
IDE 設定パネル	181
ダイアログのボタン	182
IDE 設定パネルを選択	183
一般 IDE 設定	183
エディタの IDE 設定	193
デバッガの IDE 設定	201
IDE をカスタマイズ	207
ダイアログのボタン	208
コマンドをカスタマイズ	209
キーバインディングをカスタマイズ	213
ツールバーをカスタマイズ	219
第 9 章 ターゲットのオプション設定	225
ターゲット設定の解説	225
設定パネル	225
ダイアログのボタン	226
設定パネルを選択	228
ターゲットの設定	229
コード生成の設定	247
エディタの設定	251
デバッガの設定	251
第 10 章 コンパイルとリンク	255
コンパイラを選択	255
プラグインコンパイラの仕組み	256

ファイル拡張子を設定	256
プロジェクトのコンパイル、リンク	256
ファイルをコンパイル	257
リンク順を設定	258
プロジェクトを更新	258
プロジェクトをメイク	259
デバッグを有効にする	259
プロジェクトを実行	260
プロジェクトをデバッグ	260
リンクマップを生成	261
修正日時を同期	261
オブジェクトを削除	261
高度なコンパイルオプション	262
プリコンパイルまたはプリプロセスヘッダを使う	262
プリコンパイルヘッダを作成	263
C/C++ 用のシンボルを定義	265
Pascal 用のシンボルを定義	267
ソースコードをプリプロセス	268
ソースコードを逆アセンブル	268
メッセージウィンドウの解説	269
メッセージウィンドウを使う	271
エラーと警告を見る	271
メッセージを 1 つずつ見る	272
コンパイラのエラーと警告を修正	273
リンクのエラーを修正	274
Pascal の循環参照を修正	274
メッセージウィンドウの保存と印刷	275
変更したファイルでエラーを見つける	276

第 11 章 ソースコードのデバッグ	277
シンボルファイル	277
デバッグの準備	278
デバッグのためにビルドターゲットを設定	279
デバッグのためにファイルを設定	279
シンボル情報を生成	280
デバッグを使う	281
デバッグのウィンドウ	281
プログラムウィンドウ	282
ブラウザウィンドウ	289
シンボルのヒント	293
デバッグのコンテキストメニュー	294
プロセスウィンドウ	294

Expression ウィンドウ	297
大域変数ウィンドウ	297
ブレークポイントウィンドウ	298
Watchpoint ウィンドウ	299
レジスタ詳細ウィンドウ	300
レジスタウィンドウ	304
Log ウィンドウ	306
変数ウィンドウ	307
配列ウィンドウ	307
メモリウィンドウ	309
基本的なデバッグ	311
起動方法	311
コードの実行、ステップ実行、停止	313
コードのナビゲーション	320
ブレークポイント	324
Watchpoint	330
データの表示と変更	332
ソースコードの編集	343
評価式	343
評価式の翻訳	344
評価式を使う	346
評価式の例	348
評価式のシンタックス	349
トラブルシューティング	353
一般的な問題	353
デバッガ起動時の問題	353
デバッガ実行時の問題 / クラッシュ	354
ブレークポイントの問題	355
変数の問題	356
ソースファイルの問題	359
デバッガのエラーメッセージ	360

第 12 章 RAD デザインとレイアウト	365
RAD とは	365
CodeWarrior RAD ツール	365
クラスオーサリング	366
コンポーネントモデル	366
RAD プロジェクトを作成	367
RAD ウィザード	369
Java アプレットウィザード	369
Java アプリケーションウィザード	375
Java Bean ウィザード	379

デザインを作成	383
プロジェクトへデザインを追加	383
プロジェクトウィンドウのデザインビュー	385
ターゲットビューのデザイン	385
レイアウトを作成	386
第 13 章 RAD レイアウトの編集	389
レイアウトエディタ	389
レイアウトを作成	389
レイアウトを修正	390
レイアウトウィザード	396
Java フレームウィザード	396
コンポーネントパレット	398
コンポーネントパレットのツールバー	398
カタログのポップアップメニュー	398
コンポーネントツール	399
コンポーネントカタログウィンドウ	400
コンポーネントカタログのツールバー	401
カタログ欄	402
コンポーネント欄	402
コンポーネントカタログのコンテキストメニュー	405
コンポーネントカタログを作成	406
オブジェクトインスペクタ	409
オブジェクトポップアップメニュー	410
プロパティタブ	411
イベントタブ	411
オブジェクトインスペクタのコンテキストメニュー	412
第 14 章 RAD コンポーネント	415
Java AWT コンポーネント	415
Java Swing コンポーネント	416
コンポーネントエディタ	418
メニューエディタ	418
ポップアップメニューエディタ	421
第 15 章 RAD ブラウザ	423
ブラウザウィンドウの RAD 機能	423
コントロールタブ	423
プロパティビュー	424
メソッドビュー	425
イベントビュー	426

RAD ダイアログ	427
新規プロパティダイアログ	427
新規メソッドダイアログ	430
新規イベントセットダイアログ	433
新規 Java イベントセットウィザード	434
新規イベントダイアログ	439
第 16 章 バージョン管理システム	443
バージョン管理システムを使う	443
入手可能な VCS プラグイン	443
VCS を有効にする	444
VCS ソフトウェアをインストール	444
VCS ソフトウェアを利用	445
VCS 機能を実行	447
VCS メニュー	447
VCS ログインダイアログ	449
VCS ポップアップメニュー	449
プロジェクトウィンドウ	451
VCS メッセージウィンドウ	452
一般的な VCS の操作	452
ファイル状況を取得	453
チェックインしたファイルを修正	453
その他の操作	453
第 17 章 CodeWarrior リファレンス	455
IDE メニューリファレンス	455
ファイルメニュー	456
編集メニュー	459
検索メニュー	461
プロジェクトメニュー	464
デバッグメニュー	468
データメニュー	471
ブラウザメニュー	473
ウィンドウメニュー	474
レイアウトメニュー	477
バージョン管理システム (VCS) メニュー	477
ヘルプメニュー	478
Java 例外サブメニュー	479
ツールバーサブメニュー	480
デフォルトのキーバインディング	481
ファイルメニュー	481

編集メニュー	482
検索メニュー	483
プロジェクトメニュー	484
デバッグメニュー	484
データメニュー	485
ブラウザメニュー	486
その他機能	486
エディタ	486
コンポーネントカタログ	488
レイアウトメニュー	488
ウィンドウメニュー	488

付録 A Perl スクリプト	491
Perl ソフトウェアプラグインをインストール	491
Perl 設定パネルの設定	492
Perl スクリプトの例	493
注意点	494
StdIn の用途	494
リンカエラーを避ける	494

索引	495
----	-----

第 1 章 紹介

このマニュアルでは CodeWarrior IDE (Integrated Development Environment : IDE) について詳しく説明します。IDE の機能を使って、さまざまなプログラミング言語 (C/C++、Java など) で各種プラットフォーム上で実行可能なソフトウェアを開発することができます。

注意 : IDE 4.0 よりも古いバージョンの IDE が製品に入っている場合、ここで述べる新機能は利用できません。「[新機能](#)」(p18) で IDE 4.0 の新機能についてご確認ください。ツールをアップデートするパッチの情報は、メトロワークスの Web サイト (<http://www.metrowerks.co.jp/>) をご覧ください。

この章では CodeWarrior IDE の概要を示します。ここでは以下の項目について説明します。

[CodeWarrior IDE について](#)

[IDE User Guide の概要](#)

[リリースノートについて](#)

[マニュアルの表記規則](#)

[新機能](#)

[さらに知識を深めるために](#)

CodeWarrior IDE について

CodeWarrior IDE はシンプルで強力なグラフィカルユーザーインターフェースとコンピュータソフトウェア開発ツールを備えたアプリケーションです。IDE を使って、各種コンピュータシステム上で動作する実行可能コード (プログラム、プラグイン、ライブラリなど) をさまざまなプログラム言語で開発できます。

IDE は、ソースファイル (例えば C++ 言語で書かれたファイル)、リソースファイル、ライブラリ、その他のファイルや設定ファイルをプロジェクトにまとめます。複雑なビルドスクリプトやメイクファイルを書く必要はありません。マウスやキーボード操作で簡単にソースファイルをプロジェクトへ追加、削除することができます。ビルドスクリプトの編集は不要です。

1 つのプロジェクトで複数のプラットフォーム用の設定を作成し、管理することができます。CodeWarrior が動作するプラットフォームを「ホスト」と呼びます。このホスト上で IDE を使って、さまざまなプラットフォームやオペレーティングシステムをターゲットにしたコードを開発します。「ターゲット」には 2 つの意味があります。

プラットフォームターゲット：作成するコードが動作するオペレーティングシステム、プロセッサ、マイクロコントローラを意味します。例えばあるプロセッサを持つデスクトップオペレーティングシステム上で動作するコードを書く場合、プラットフォームターゲット用のコードを作成していることになります。

ビルドターゲット：コードの内容を決定する設定やファイルをまとめたものです。コードを最終出力にコンパイル、リンクする方法なども含まれます。

1つのプラットフォームターゲットに対し、複数のビルドターゲットを設定することもできます。例えば Windows (プラットフォームターゲット) 用プログラムのデバッグバージョン (ビルドターゲット) と最適化バージョン (ビルドターゲット) をコンパイルすることができます。ビルドターゲットは同じプロジェクトに含まれるファイルを共有することができます。プログラムのデバッグを終えたら、プロジェクトのビルドターゲットを変更して [[メイク](#)] コマンドを選択するだけで簡単に最終バージョンを生成できます。

CodeWarrior IDE にはデバッガ、コンパイラ、リンカ、コードブラウザ、ソースコードエディタや Java 用 RAD (Rapid Application Development) ツールが含まれています。これらのツールを使って、実行可能アプリケーションを作成するまでに必要な作業、つまり、コードの編集、ナビゲート、ブラウズ、コンパイル、リンク、デバッグができます。プロジェクトのコード生成、デバッグ、ナビゲーションに関するオプションはすべて IDE 内で設定できます。

CodeWarrior IDE には、コード開発に必要なものがすべて含まれています。

IDE User Guide の概要

本マニュアルにはいくつかの章があります。各章の先頭には、その章の内容をまとめた概要があります。各章の概要を以下に示します。

[紹介](#) : (この章) : CodeWarrior IDE および関連マニュアルの紹介

[はじめに](#) : システムの条件、インストール方法の説明

[プロジェクト](#) : プロジェクトの作成、設定、管理方法の説明

[ファイル](#) : ファイルの保存、バックアップ、比較、印刷の説明

[ソースコードの編集](#) : テキスト、ソースコードの編集とナビゲートの説明

[テキストの検索と置換](#) : テキストをファイル (複数も可) から検索、置換する方法の説明

[ソースコードのブラウズ](#) : プロジェクトをさまざまな角度から検証するブラウザの説明

[IDE のオプション設定](#) : IDE 設定パネルとツールバーをカスタマイズする方法の説明

[ターゲットのオプション設定](#) : プロジェクトとそのビルドターゲットの設定、カスタマイズの説明

[コンパイルとリンク](#) : プロジェクトのビルドターゲットとファイルをコンパイル、リンク、実行、更新、プリプロセス、プリコンパイルする方法の説明

[ソースコードのデバッグ](#) : CodeWarrior デバッガでコードを検証する方法の説明

[RAD デザインとレイアウト](#) : CodeWarrior における RAD ツールの実装

[RAD レイアウトの編集](#) : CodeWarrior の RAD ツールを使ってプロジェクトデザインとレイアウトを作成、管理する方法の説明

[RAD コンポーネント](#) : RAD ツールに含まれるコンポーネントの説明

[RAD ブラウザ](#) : クラスブラウザとオブジェクトインスペクタを使って RAD デザインを検証する

[バージョン管理システム](#) : CodeWarrior IDE とバージョン管理システムを使用する方法の説明

[CodeWarrior リファレンス](#) : メニューコマンドやデフォルトのキーバインディングの説明

リリースノートについて

リリースノートを必ずお読みください。リリースノートには、重要な情報（新しい機能、バグ修正、マニュアルに書かれていない最新の情報など）が含まれています。リリースノートは CodeWarrior CD の Release Notes フォルダにあります。

マニュアルの表記規則

CodeWarrior IDE マニュアルの表記規則を説明します。

[表記について](#)

[ホストについて](#)

表記について

特定の情報を表すスタイルについて説明します。

注意、警告、ヒント、および初心者用のヒント

「注意」は、重要な事実を言い換えたり、自明ではない事実に注意を向けます。

「**警告**」は、実行すると取り返しのつかないものを注意したり、発生する可能性のあるエラーを知らせます。

「ヒント」は、CodeWarrior IDE をより活用するためのヒントです。

「初心者」は、プログラミングの初心者が用語や概念をよりよく理解できるようにします。

書体の規則

異なる書体 (Courier という書体です) のテキストは、ファイル名やフォルダ名、コード、またはコンピュータのハードディスク上で見られるその他の項目を示します。

CodeWarrior のメニューコマンドやオプション名などは括弧付き([ポストリンカ] オプションなど) で示します。ターゲットによって名前が変更されるコマンドの変更部分は、斜体で表示します ([*Target* の設定])。ユーザーご本人が入力するテキストを、異なる書体 (Courier という書体です) で表示します (「 *Target* 」)。

下線付きの青色のテキストは (例 : 「 [IDE User Guide の概要](#) 」) オンラインドキュメント (Adobe Acrobat など) でのハイパーテキストを示します。これをクリックすると該当ページへジャンプします。

ホストについて

CodeWarrior は以下のホストプラットフォームおよびオペレーティングシステム上で動作します。このマニュアルでは、オペレーティングシステムに関係なく、プラットフォームの名前をホスト名として使用します。

CodeWarrior は以下のオペレーティングシステム上で動作します。

Windows : Win32 に準拠するデスクトップバージョンの Windows オペレーティングシステム (Windows 95/98、Windows NT4.0 など)

新機能

CodeWarrior IDE 4.0 で採用された新しい機能を以下にあげます。

[RAD ツール \(Rapid Application Development Tool \)](#)

[新規コマンド](#)

[IDE のカスタマイズ](#)

[キーバインディングの改良](#)

[XML プロジェクトのインポートとエクスポート](#)

[デバッグの自動変数表示](#)

[ユーザー定義アクセスパス](#)

[プロジェクトウィンドウの改良](#)

RAD ツール (Rapid Application Development Tool)

今回のバージョンの CodeWarrior には、Java 用の RAD (Rapid Application Development) ツールが含まれています。この RAD ツールを使って、Java のアプレット、アプリケーション、フレームワークを素早く作成することができます。このツールにはレイアウトエディタ、RAD コンポーネント、カタログウィンドウ、オブジェクトインスペクタが含まれます。CodeWarrior ウィザードは RAD デザインを作成する過程を支援するツールです。プロジェクトウィンドウの [デザイン] タブでプロジェクトに対するさまざまなデザインを管理できます。

詳細は「[RAD デザインとレイアウト](#)」(p365)、[「RAD レイアウトの編集」](#)(p389)、[「RAD コンポーネント」](#)(p415)、[「RAD ブラウザ」](#)(p423) を参照してください。

新規コマンド

ファイルメニューの [新規] コマンドが新しくなりました。このコマンドでファイル、プロジェクト、RAD デザイン、オブジェクトを作成します。また CodeWarrior の Makefile インポートウィザード、RAD ウィザードへもアクセスできます。

詳細は「[新規プロジェクトを作成](#)」(p40)、「[プロジェクトへメイクファイルをインポート](#)」(p47)、「[新規ファイルを作成](#)」(p77)、「[RAD プロジェクトを作成](#)」(p367) を参照してください。

IDE のカスタマイズ

プログラマの個人的なニーズに合わせて CodeWarrior をカスタマイズできます。メニューコマンド、ツールバーのボタン、キーバインディング、コンテキストメニューを作成することができます。新しい項目とコマンドラインを関連づけることもできます。カスタマイズしたメニューコマンドから IDE の情報（現在のエディタの選択、アクティブなウィンドウ、現在のプロジェクト、出力ファイル）へアクセスできます。

詳細は「[IDE をカスタマイズ](#)」(p207) を参照してください。

キーバインディングの改良

新旧のキーバインディングに互換性があります。クロスプラットフォームでキーバインディングファイルをインポート、エクスポートすることができます。キーバインディングを変更するウィンドウも改良されました。

詳細は「[キーバインディングをカスタマイズ](#)」(p213) を参照してください。

XML プロジェクトのインポートとエクスポート

CodeWarrior プロジェクトマネージャにより、プロジェクトの詳細を XML (eXtensible Markup Language) フォーマットでインポート、エクスポートすることができます。

詳細は「[プロジェクトのインポートとエクスポート](#)」(p73) を参照してください。

デバッガの自動変数表示

デバッガに自動変数表示機能が追加されました。ソースコードが参照している変数を自動的に表示します。

詳細は「[シンボルのヒント](#)」(p293) を参照してください。

ユーザー定義アクセスパス

現在のアクセスパス設定の他に、独自のソースツリーを設定することができます。グローバルまたはターゲットレベルのソースツリーを指定できます。絶対パス、環境パス、レジストリキーのいずれかを使います。ソースツリーを使用してアクセスパスを設定することもできます。

独自のソースツリーとアクセスパスの作成方法は「[ソースツリー](#)」(p188) を参照してください。

プロジェクトウィンドウの改良

プロジェクトウィンドウが改良され、ファイルリストを各列の名前でソートすることができます。

詳細は「[ソートボタン](#)」(p36) を参照してください。

さらに知識を深めるために

新しいプラットフォームで作業を始める場合、または初めて CodeWarrior を使う方は、「[QuickStart とチュートリアル](#)」(p20) を参照してください。

CodeWarrior IDE の熟練したユーザーの方は、「[新機能](#)」(p18) を参照してください。

新しいオペレーティングシステムをターゲットにして作業を始める方は、「[ターゲットマニユアル](#)」(p20) を参照してください。

すぐにデバッグができる場合は、「[ソースコードのデバッグ](#)」(p277) を参照してください。

CodeWarrior を効率的に使うための情報を以下に示します。

[QuickStart とチュートリアル](#)

[ターゲットマニユアル](#)

QuickStart とチュートリアル

ここで紹介するドキュメントはすべて CodeWarrior CD (製品によっては CodeWarrior Reference CD) の CodeWarrior Documentation フォルダにあります。

初めて CodeWarrior IDE を使う方は、次のマニュアルをお読みください。日本語に翻訳されているマニュアルは日本語版 CD にあります。すべてのマニュアルが翻訳されているわけではありません。日本語に翻訳されているマニュアルについては、このマニュアルの最後に記載されている「CodeWarrior 文書のガイド」を参照してください。

『CodeWarrior QuickStart』に CodeWarrior の概要、関連ドキュメントの所在が書かれています。印刷されたマニュアルとは別に『CodeWarrior QuickStart』が CodeWarrior CD または日本語版 CD に入っています。

「[CodeWarrior IDE の紹介](#)」(p27) に、CodeWarrior IDE ユーザーインターフェースの簡単な説明があります。

CodeWarrior CD の Core Tutorials フォルダに、CodeWarrior IDE に慣れるためのサンプルプロジェクトがいくつか含まれています。

メトロワークスが提供するオンラインドキュメント用のビューアの使い方は、CodeWarrior CD の CodeWarrior Documentation フォルダ内にあります。

ターゲットマニユアル

このマニュアルはクロスプラットフォームな CodeWarrior IDE の機能の使い方を説明するもので、特定のターゲット用ソフトウェアの開発方法については説明しません。

特定のプラットフォームターゲット用のソフトウェアを開発するには、[表 1.1](#) の各マニュアルを参考にしてください。

注意： CodeWarrior 製品によっては、[表 1.1](#) のターゲット用ソフトウェアを生成できないものもあります

表 1.1 プラットフォームターゲットのマニュアル

プラットフォームターゲット	『Targeting』マニュアル
Java 仮想マシン	Targeting the Java VM
Mac OS	Targeting Mac OS
MIPS	Targeting MIPS
Nucleus	Targeting Nucleus
Palm OS	Targeting Palm OS
PlayStation	Targeting PlayStation OS
PowerPC エンベデッドシステム	Targeting PowerPC for Embedded Systems
Win32/x86	Targeting Windows

第 2 章 はじめに

この章では CodeWarrior IDE (Integrated Development Environment : 統合開発環境) を使い始めるにあたって必要となる情報を提供します。システムの条件、ソフトウェアをインストールするための情報、さらに CodeWarrior IDE のユーザーインターフェースや機能について説明します。

[システムの必要条件](#)

[CodeWarrior IDE のインストール](#)

[プログラミングの概念](#)

[CodeWarrior IDE の紹介](#)

注意： このマニュアルはクロスプラットフォームな IDE の機能の使い方を説明するものです。特定のプラットフォーム用ソフトウェアを開発する方法については各『Targeting』マニュアルを参照してください。詳細は「[ターゲットマニュアル](#)」(p20) を参照してください。

ヒント： CodeWarrior IDE の機能を「[CodeWarrior IDE の紹介](#)」(p27) で簡単に説明しています。

システムの必要条件

CodeWarrior IDE に必要なシステムを各プラットフォームごとにまとめます。

[Windows の必要条件](#)

Windows の必要条件

Windows をホストにする CodeWarrior は以下のシステムが必要です。

Pentium クラスのプロセッサ

最低 32MB の RAM (64MB を推奨)

約 120MB のハードディスクスペース (最小インストール)

約 450MB のハードディスクスペース (フルインストール)

Microsoft Windows オペレーティングシステム (Windows 95、Windows 98、Windows NT 4.0 SP3)

ソフトウェアをインストールするための CD-ROM ドライブ

CodeWarrior IDE のインストール

CodeWarrior IDE をインストールする方法は、CodeWarrior CD に含まれる『CodeWarrior QuickStart』を参照してください。CodeWarrior インストーラは CodeWarrior IDE、コンパイラ、リンカ、各ツール、およびデバッグもインストールします。

プログラミングの概念

コンピュータプログラミングの初心者の方にとって、この節はこのマニュアルで使用される用語の参考になります。

コンピュータプログラムを作成する手順を次に示します。

[入力ファイルの作成](#)

[ソフトウェアの生成](#)

[デバッグと修正](#)

まず、C、C++、Java、Pascal、アセンブリ言語（機械コード）などのコンピュータ言語を使用して、ステートメントを含むファイルを作成します。ウィンドウ、ダイアログ、メニュー、その他のユーザーインターフェース項目などの記述を含むリソースファイルも作成します。

次にソースファイルを、ターゲットコンピュータ上で実行できるファイルに変えるためのツールを使います。ツールには、コンパイラ、リンカ、アセンブリ言語のアセンブラなどがあります。ツールおよびターゲットコンピュータの違いによって、いろいろな種類のソフトウェアを数多く作成できます。作成できるソフトウェアの種類は、アプリケーション（または実行可能ファイル）、ダイナミックリンク（共有）ライブラリ、および静的ライブラリです。

ソフトウェアを作成したら、デバッグを使ってソフトウェアを検証します。デバッグでソフトウェアが意図した通りに動作するかどうかを調べます。ソフトウェアが正しく動作しない場合、ソースを変更して動作を修正します。正しく動作するまでソースのデバッグを繰り返します。

入力ファイルの作成

プログラムを作成するために必要なファイルには、ソースファイル、リソースファイル、ヘッダファイル、ライブラリファイル、およびプロジェクトファイルがあります。

ソースファイル

ソースファイルは、プログラムステートメントを含むテキスト形式のコンピュータ言語ファイルで、C、C++、Java、Pascal、アセンブリ言語などの言語で書かれます。

リソースファイル

リソースファイルには、ウィンドウ定義、ダイアログのレイアウト、テキスト文字列などのユーザーインターフェース項目の記述が含まれます。リソースファイルは、ソフトウェアにリンクされるバイナリファイルか、またはリンクされる前に特別なリソースコンパイラでコンパイルされるテキストファイルです。リソースを別のファイルに入れておくと、プログラムの他の部分を再コンパイルしなくともリソースをカスタマイズすることができ、便利です。

ヘッダファイル (インターフェースファイル)

ヘッダファイルは、インターフェースファイルやインクルードファイルとも呼ばれます。ヘッダファイルはソースファイルが参照するテキストファイルです。一般に、これらのファイルにはソースファイルによって使用されるオブジェクト、変数、データ構造体、関数、ライブラリや他のソースファイルにある項目が含まれます。

ライブラリファイル

ライブラリファイルには、既にコンパイルされたオブジェクト、変数、関数が含まれます。静的ライブラリとダイナミックリンクライブラリの 2 種類のライブラリがあります。

IDE はプログラムの中に 静的ライブラリをビルドします。プログラムをリビルドしない限り、静的ライブラリは変化しません。静的ライブラリを他のプログラムと共有することはできません。

プログラムは ダイナミックリンクライブラリも参照しますが、こちらはプログラムの中にはビルドされません。その代わり、プログラムはその実行中にライブラリを動的に参照します。一般的に複数のプログラムがダイナミックリンクライブラリを共有します。また、ダイナミックリンクライブラリを新しいライブラリと取り替えても (新しいライブラリに同じ項目が含まれていれば) 影響はありません。

注意： ほとんどのプロセッサやオペレーティングシステムは静的ライブラリをサポートしていますが、ダイナミックリンクライブラリをサポートするものは多くありません。各プラットフォームターゲット用のライブラリの作成、使用方法是「[ターゲットマニユアル](#)」(p20) を参照してください。

プロジェクトファイル

プロジェクトファイルは、少なくとも 1 つのビルドターゲットを持ちます。各ビルドターゲットにはソースファイルのリスト、リソース、ヘッダ、ライブラリが含まれます。他のプロジェクトを含めることもできます。各ビルドターゲットにはどのようにソフトウェアを作成するのかを IDE に指示する独自の設定も含まれます。詳細は「[プロジェクトとターゲット](#)」(p27) を参照してください。

ソフトウェアの生成

ソフトウェアをビルドするための IDE のツールには、コンパイラ、アセンブラ、リンカがあります。IDE はプロジェクトのビルドターゲット情報や設定から、使用するツールを自動的に選択します。

コンパイラ

コンパイラは、C、C++、Pascal、または Java ファイルなどのソースファイルを、マシンコード（オブジェクトコードとも呼ばれます）にコンパイルするツールです。このマシンコードは、最終的なビルド段階で使用されます。コンパイラは、プログラムをビルドするために起動する最初のツールです。

アセンブラ

アセンブラは、アセンブリ言語のソースファイルをオブジェクトコードのファイルへ翻訳するツールです。アセンブラはコンパイラ的一种ですが、アセンブリ言語のソースコードだけを翻訳します。高レベルのソースコード（C、C++、Pascal、Java）は翻訳しません。

リンカ

リンカはプロジェクトのビルドターゲットにあるオブジェクトコードと、コンパイラとアセンブラが生成したオブジェクトコードを一緒にリンクし、ソフトウェアを作成するツールです。

出力ファイル

出力ファイルはリンカが生成するソフトウェアです。ソフトウェアにはアプリケーションやライブラリなど、多くの種類があります。プラットフォームターゲットに対して開発できるソフトウェアの種類についての詳細は、各プラットフォームの『Targeting』マニュアルを参照してください。各『Targeting』マニュアルの内容は「[ターゲットマニュアル](#)」(p20)を参照してください。

デバッグと修正

デバッグは、プログラムの実行をコントロールし、コードのエラーを見つけるためのツールです。デバッグを使ってプログラムの実行をある位置で停止したり、変数の内容を見ることができます。コードを1行ずつ実行することもできます。

デバッグを使用するためには、コードの実行時にデバッグが必要とする特殊な情報を作成します。この情報はデバッグ情報またはシンボルファイルと呼ばれ、デバッグがソースのステートメント、変数、オブジェクト、データ型の表示やコントロールするために必要な情報を含んでいます。

IDE はプログラミング環境にデバッグを統合しています。プラットフォームによっては IDE はサードパーティのデバッグもサポートします。詳細は各『Targeting』マニュアルを参照してください。

CodeWarrior デバッグの使い方は「[ソースコードのデバッグ](#)」(p277)を参照してください。

CodeWarrior IDE の紹介

ここでは CodeWarrior IDE で行うタスクや操作について説明します。

[プロジェクトとターゲット](#)

[ソースコードの編集とブラウズ](#)

[コンパイルとリンク](#)

[プロジェクトのデバッグ](#)

[RAD ツール](#)

[バージョン管理](#)

[スクリプト](#)

[IDE のカスタマイズ](#)

[サードパーティツールの使用](#)

ここで使われる用語の説明は「[CodeWarrior IDE について](#)」(p15)、「[プログラミングの概念](#)」(p24)を参照してください。

プロジェクトとターゲット

IDE はプロジェクトとビルドターゲットを使って、プログラムを作成するために使うファイルや設定を管理します。プロジェクトファイルには最低 1 つのビルドターゲットが含まれます。ビルドターゲットはソースファイル、リソースファイル、ライブラリ、他のプロジェクト、特定のプロセッサまたはオペレーティングシステム用ソフトウェアを作成するための設定や記述をまとめたものです。プロジェクトの中のビルドターゲットはファイルを共有できますが、それぞれ独自の設定を使います。

CodeWarrior IDE にはあらかじめ設定されたステーションリプロジェクトが含まれています。対象となるプラットフォームターゲットと使用言語を決め、それに一致するステーションリプロジェクトを選択するだけで、簡単に新規プロジェクトを作成できます。

ビルドターゲットの設定パネルで、ターゲットプロセッサやオペレーティングシステムのオプションの設定、コンパイラによる最適化やオブジェクトコードのカスタマイズ、ソースコードの翻訳方法やビルドターゲットに追加するファイルの種類の指定、その他のオプションの設定が可能です。プラットフォームターゲットによって設定可能なオプションは異なります。

プロジェクトとビルドターゲットについての詳細は「[ターゲットマニュアル](#)」(p20)、「[プロジェクト](#)」(p31)、「[ターゲットのオプション設定](#)」(p225)を参照してください。

ソースコードの編集とブラウズ

IDE は強力かつ柔軟な、ソースコードとテキストファイル編集用テキストエディタを備えています。一般的なテキスト編集機能に加え、IDE はドラッグ&ドロップ、いろいろなフォーマットのテキストファイルの編集、保存ができます。またテキストファイルにマーカーを設定してマーカーの位置へ移動することも可能です。

エディタには多くの実用的な機能があります。自動インデント、シンタックスカラーリング、関数およびヘッダのポップアップメニュー、括弧の自動バランス機能などでプログラミングを支援します。編集、表示、関数、データ構造体、変数、オブジェクトのナビゲートについてエディタはクラスブラウザと密接な関係があります。

「[検索](#)」ダイアログと[検索メニュー](#)には、ファイル（複数も可）内のテキスト検索および置換に関するコマンドがあります。通常のテキストか、評価式の検索と置換ができます。

ファイルの比較とマージのコマンドは、2 つのテキストファイルを横に並べて表示するので、それらの差異を簡単に見つけることができます。また、2 つのフォルダの内容を比較することもできます。

テキスト、ソースファイルについての詳細は「[ファイル](#)」(p77)、[「ソースコードの編集」](#)(p95)、[「テキストの検索と置換」](#)(p115)、[「ソースコードのブラウズ」](#)(p143) を参照してください。

コンパイルとリンク

IDE には、[プリプロセス] [プリコンパイル] [コンパイル] [実行] [逆アセンブル] [シンタックスをチェック] のコマンドがあります。IDE は [コンパイル] または [メイク] コマンドが選択されたときに、自動的に適切なコンパイラとリンカを選択します。IDE はプロジェクトのビルドターゲット設定を使ってコンパイラとリンカがどのようにファイルとデータを処理するのかを指定します。ユーザーの皆さんがすべきことは、[プロジェクトメニュー](#) からコマンドを選択することだけです。

オプションの設定、コンパイル、リンク、その他のソフトウェア生成についての詳細は、[「ターゲットのオプション設定」](#)(p225)、[「コンパイルとリンク」](#)(p255) を参照してください。

プロジェクトのデバッグ

CodeWarrior デバッガはソースコードのプログラミングとデバッグをシームレスにサポートします。デバッガは x86、PowerPC、68K、Java のデバッグをフルサポートします。

一旦デバッガを有効にしたら、プロジェクトを実行するだけでデバッガ機能が利用可能になります。いつでもプログラムを一時停止してブレークポイントや Watchpoint を設定したり、変数やメモリを見たり、関数へのステップインやステップアウトが可能です。その他のデバッグ機能も使用できます。

注意： デバッガを含まない CodeWarrior IDE も販売されています。この場合、独立したデバッガアプリケーションか、サードパーティのデバッガを使用してください。詳細は各『Targeting』マニュアルを参照してください。

RAD ツール

CodeWarrior には、迅速なアプリケーション開発を支援する RAD ツールが含まれています。RAD ツールを使ってグラフィカルユーザーインターフェースやそれに関連する動作を作成できます。CodeWarrior は自動的にソフトウェアのフレームワークを管理して基本的なソー

スコードを生成します。また生成されたソースコードを修正したり、より効率の良いコードを自分で書くこともできます。

CodeWarrior の RAD ツールについては「[RAD デザインとレイアウト](#)」(p365) を参照してください。

バージョン管理

IDE をバージョン管理システム (Version Control Systems : VCS) と一緒に使うこともできます。ファイルサーバにログインしてファイルの修正や保存、VCS ソフトウェアによるバージョン管理などが可能です。

IDE と VCS ソフトウェアと一緒に使うための情報は、「[VCS ポップアップメニュー](#)」(p99)、[「バージョン管理システム」](#) (p443) を参照してください。

スクリプト

IDE は Perl スクリプトやバッチファイルによるスクリプトをサポートします。プリエンブタイプで時間のかかる複雑なタスクをスクリプトで自動化できます。

IDE とスクリプトについては「[Perl スクリプト](#)」(p491)

IDE のカスタマイズ

IDE にはユーザーが設定するオプションが多数あります。IDE の「[IDE 設定](#)」ダイアログでカスタマイズ(ソースコードを表示するカラーやフォントの指定、キーボードショートカットの割り当てなど)が可能です。IDE には簡単にカスタマイズできるツールバーがあります。このツールバーのボタンをクリックするだけで簡単にコマンドを実行したり情報を見ることができます。

IDE のカスタマイズの詳細は「[IDE のオプション設定](#)」(p181) を参照してください。

サードパーティツールの使用

IDE はサードパーティのテキストエディタ、デバッガなどの開発ツールに対応します。

CodeWarrior とサードパーティツールの使用については「[IDE その他設定](#)」(p185)、[「プロジェクトをデバッグ」](#) (p260)、[「バージョン管理システム」](#) (p443) を参照してください。

第 3 章 プロジェクト

この章では、CodeWarrior IDE のプロジェクトウィンドウの使い方、プロジェクトの作成および設定方法について説明します。

プロジェクトには、最低 1 つのビルドターゲットが含まれます。プロジェクト内の各ビルドターゲットには、IDE が出力ファイルをビルドするために使うファイルがまとめられています。プロジェクト内のビルドターゲットはファイルを共有することができます。出力ファイルにはアプリケーション、静的ライブラリ、ダイナミックリンクライブラリを含むものもあります。

プロジェクト内の各ビルドターゲットは、IDE がどのように出力ファイルをビルドするかを指定したオプション設定をそれぞれ持っています。コードの最適化、ブラウザ、デバッグ、コンパイラの警告などをコントロールするオプションが多数あります。

プロジェクトのビルドターゲットを、他のビルドターゲットに基づいて設定することも可能です。これにより、別々のプラットフォームターゲット用の出力ファイルを 1 つの出力ファイルにまとめることができます。

この章では、プロジェクトを作成する、開く、ファイルを追加する、保存するなどのプロジェクトに関する基本的な作業を説明します。また、プロジェクトウィンドウ内でファイルを移動する、デバッグ用にファイルをマークする、入れ子（ネスト）されたプロジェクトおよびターゲットを作成する、さらにプロジェクトウィンドウをファイルのグループに分けるなどの高度な操作についても説明します。

この章では、以下の項目について説明します。

[プロジェクトウィンドウの解説](#)

[新規プロジェクトを作成](#)

[プロジェクトステーションナリを使う](#)

[プロジェクトへメイクファイルをインポート](#)

[既存のプロジェクトを開く](#)

[プロジェクトを保存](#)

[プロジェクトを閉じる](#)

[デフォルトのプロジェクトを選択](#)

[プロジェクト内のファイル管理](#)

[複合プロジェクトを作成](#)

[プロジェクト情報を検証](#)

[プロジェクトを移動](#)

[プロジェクトのインポートとエクスポート](#)

[プロジェクトでデバッグを制御](#)

[プリプロセッサシンボルをプロジェクトに追加](#)

プロジェクトウィンドウの解説

プロジェクトウィンドウは、プロジェクトファイル内のファイルやターゲットを 4 つのビューで表示します。ビューには、[ファイル]、[デザイン]、[リンク順] (セグメント、オーバーレイとも呼ばれます) 「ターゲット」があります。各ビューを表示するには、プロジェクトウィンドウで該当するタブをクリックします ([図 3.1](#))。

図 3.1 プロジェクトウィンドウのビュータブ



ファイルビューはプロジェクトにあるファイルの一覧を表示します。このビューは各自の作成した階層グループを表示します。

デザインビューはプロジェクトの RAD (Rapid Application Development) デザインを表示します。

リンク順ビュー (セグメントビューまたはオーバーレイビューとも呼ばれます) は、IDE がどのように現在のターゲットの最終ファイルをコンパイル、リンクするのかを表示します。

ターゲットビューはアクティブなビルドターゲット、RAD デザインに関するビルドターゲット、ビルドターゲットへの依存性、リンク可能なビルドターゲットの情報を表示します。

以下の節でプロジェクトウィンドウについて詳しく説明します。

[プロジェクトウィンドウのナビゲート](#)

[プロジェクトウィンドウのツールバー](#)

[ファイルビュー](#)

[デザインビュー](#)

[リンク順ビュー](#)

[ターゲットビュー](#)

デバッグ情報の詳細は「[プロジェクトでデバッグを制御](#)」(p73) を参照してください。

プロジェクトウィンドウのナビゲート

プロジェクトウィンドウ内をナビゲートするには、ウィンドウの右側の縦スクロールバー、またはキーボードの上下矢印キーを使います。プロジェクトウィンドウに多くのファイルがある場合、Home キーで最初のセグメントまたはグループの最初のヘジャンプできます。End キーは最後のセグメントの最後のファイルヘジャンプします。

Page Up キーおよび Page Down キーはプロジェクトウィンドウを上下に 1 ページずつスクロールします。

タイプ入力でファイルを選択する方法については「[キーボードによる選択](#)」(p56) を参照してください。

プロジェクトウィンドウのツールバー

プロジェクトウィンドウ中のツールバーには、コマンドを実行するボタンやプロジェクト情報を表示する項目があります。ツールバー上にどのアイコンを置くか、およびその表示順番を設定できます。ツールバーそのものを表示するかどうかも設定できます。CodeWarrior IDE のツールバーの詳細は「[IDE をカスタマイズ](#)」(p207) を参照してください。

ファイルビュー

プロジェクトウィンドウのファイルビューでは、プロジェクトのすべてのビルドターゲットについてのファイルを表示します。ファイルをグループ分けして階層的に並べることができます。これは IDE のビルドターゲットには影響しません。ファイルビューは修正状態、アクセスパス、データサイズ、コードサイズ、現在のビルドターゲット、デバッグ状況などの情報も表示します。

ファイルビューについて詳しく説明します。

[ファイル列](#)

[コード列](#)

[データ列](#)

[デバッグ列](#)

[ターゲット列](#)

[タッチ列](#)

[ソートボタン](#)

[ヘッダポップアップ](#)

[ファイルコントロールポップアップメニュー](#)

[チェックアウト状況列](#)

[プロジェクトチェックアウト状況アイコン](#)

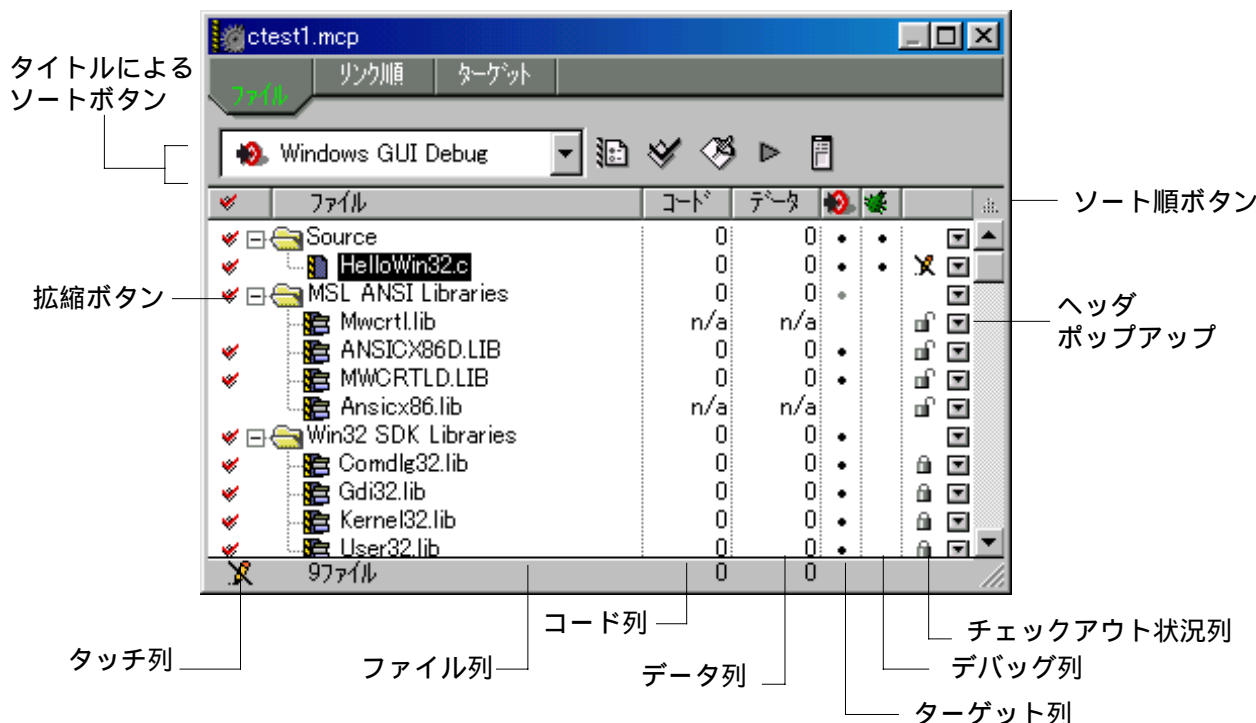
ファイル列

ファイル列は、プロジェクトに含まれるファイルとグループをユーザーの設定した階層順に表示します。グループはファイルや他のグループを含むことがあります。

ファイル列にあるファイル名をダブルクリックすると、ファイルが開きます。ファイル列からファイルを開く方法は、[「プロジェクトウィンドウからファイルを開く」\(p79\)](#)を参照してください。

ファイル列左の拡張ボタンをクリックすると、グループの内容を表示したり、隠すことができます。

図 3.2 プロジェクトウィンドウ



以下はIDEがアクセスするファイルパスを見る方法です。

プロジェクトウィンドウでファイル名を右クリックしてから、コンテキストメニューで[エクスプローラで表示する]を選択してください。

ファイル列からファイルを開く方法は、[「プロジェクトウィンドウからファイルを開く」\(p79\)](#)を参照してください。

コード列

コード列は、コンパイルされた実行可能なオブジェクトコードのサイズをファイルやグループごとに、バイトまたはキロバイト単位で示します。コード列に「0」(ゼロ)が表示された場合、ファイルがまだコンパイルされていないことを意味します。「n/a」が表示され

た場合、ファイルには IDE がコンパイルした実行可能コードがない、または現在のビルドターゲットにそのファイルが含まれていないことを意味します。

コード列の値は、最終的な出力ファイルに追加されるオブジェクトコードの量を反映しているわけではありません。リンカは、最終的な出力ファイルを作成するときにすべてのオブジェクトコードを使うわけではありません。リンカはプロジェクト内の他のファイルによって参照されるデータとコードだけを使用し、それ以外のものは無視するからです（これを「デッドストリップ」と呼びます）。

リンカの機能の詳細は、[「プロジェクトのコンパイル、リンク」\(p256\)](#) を参照してください。

データ列

データ列は、ファイルのオブジェクトコードにある非実行可能なデータ域のサイズをバイトまたはキロバイト単位で示します。「0」(ゼロ)が表示された場合、ファイルがまだコンパイルされていない、またはファイルのオブジェクトコードにデータセクションがないことを意味します。「n/a」が表示された場合、ファイルにはオブジェクトコードデータがないことを意味します。

コード列と同様に、データ列の値は最終的な出力ファイルに追加されるデータの量を反映していません。

現在のビルドターゲットにある項目に対してのみ数値が示されます。現在のビルドターゲットにない項目に対する値は、灰色で示されます。

リンカの機能の詳細は、[「プロジェクトのコンパイル、リンク」\(p256\)](#) を参照してください。

デバッグ列

デバッグ列は、ファイルのデバッグ情報を生成するか否かを示します。デバッグ列に黒いマーカがある場合、その項目のデバッグ情報を生成することを意味します。灰色のマーカは、グループ内の一部のファイルのデバッグ情報を生成できます。

デバッグ情報を生成するには：

ファイル：ファイルの横のデバッグ列をクリックする

グループ：グループの横のデバッグ列をクリックする

プロジェクト：デバッグ列を Alt + クリックする

ファイルに対するデバッグ情報の詳細は、[「ファイルのデバッグをアクティブにする」\(p74\)](#) を参照してください。

ターゲット列

ターゲット列は、ある項目がプロジェクトの現在のビルドターゲットに含まれるか否かを示します。この列は、プロジェクトに複数のビルドターゲットがあるときに表示されます。この列に黒いマーカがある場合、その項目が現在のビルドターゲットに含まれることを意味します。灰色のマーカは、グループ内の一部のファイルが現在のビルドターゲットに含まれることを示します。

現在のビルドターゲットを割り当てる、または取り除くには：

ファイル：ファイルの横のターゲット列をクリックする

グループ：グループの横のターゲット列をクリックする

プロジェクト：ターゲット列を Alt + クリックする

ターゲット列でファイルを追加または除去する方法については「[ターゲットにファイルを割り当てる](#)」(p69) を参照してください。

タッチ列

タッチ列は、ターゲットを次にビルドする際に、ファイルをコンパイルする必要があるかどうかを示します。タッチ列のマーカがある項目は、次に「[最新状態に更新する](#)」、「[メイク](#)」、「[実行](#)」、「[デバッグ](#)」コマンドが実行されるときに再度コンパイルされることを意味します。灰色のマーカはグループ内の一部のファイルが再コンパイルされることを示します。

タッチ列のマーカを付けるまたは消すには：

ファイル：ファイルの横のタッチ列をクリックする


グループ：グループの横のタッチ列をクリックする

プロジェクト：タッチ列を Alt + クリックする

この機能の詳細は「[修正日時同期](#)」(p63) を参照してください。

ソートボタン

ソートボタンはプロジェクトウィンドウのファイルビュー上で項目をソートします。各列のタイトルがソートボタンの働きをします。ある列のタイトルをクリックすると、その列に従って項目がソートされます。現在選択されているソートボタンは濃い灰色で表示されます。再度クリックするとそのタイトルによるソートが解除されます。

 3.3 のようにボタンをクリックすると、ファイル名がアルファベット順にソートされます。再度ボタンをクリックするとファイル名によるソートが解除されます。複数のタイトルによるソートはできません。

ソート順を変えるには、ソート順ボタンをクリックします。



項目を昇順でソートする



項目を降順でソートする

図 3.3 ファイルビューで項目をソート



この場合、項目はファイル名に従ってソートされる。再度このボタンをクリックするとファイル名によるソートが解除される

ヘッダポップアップ

ヘッダポップアップで、プロジェクトのソースファイル用のヘッダファイルを見たり、開いたりできます。ヘッダポップアップで、ファイルのタッチ情報をタッチとアンタッチの間で切り替えたり、その他のオプション(現在のビルドターゲットによって異なります)を設定することもできます。

グループ列のヘッダポップアップは、そのグループに含まれるファイルの一覧を表示します。ここからファイルを選択して開くこともできます。

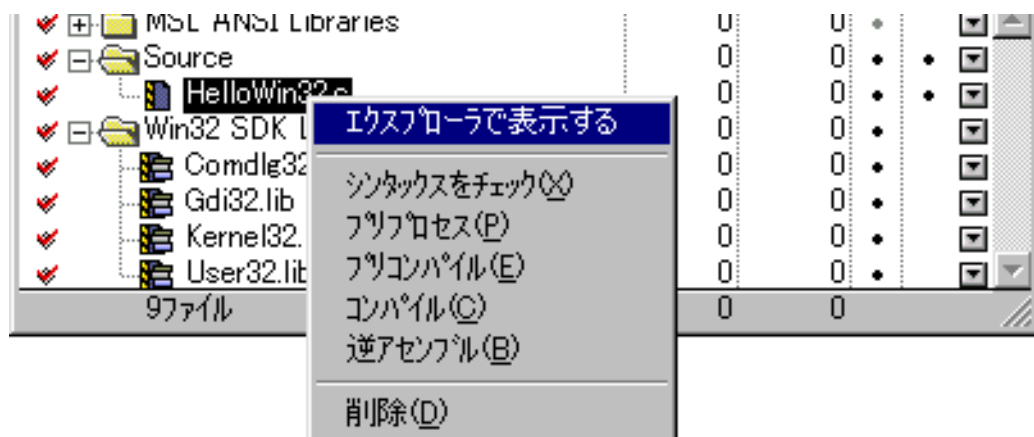
ヘッダファイルを開く詳細は「[ヘッダポップアップメニュー](#)」(p79)を参照してください。

ファイルコントロールポップアップメニュー

以下の方法でファイルコントロールポップアップメニュー(図 3.4)を表示します。

プロジェクトウィンドウでアイコンを右クリックする

図 3.4 プロジェクトウィンドウのファイルコントロールポップアップメニュー



ファイルコントロールポップアップメニューのコマンドで項目を操作できます。コマンドは選択した項目によって変わります。以下のコマンドがあります。

ファイルへのアクセスパスを見るには、ポップアップメニューから [エクスプローラで表示する] を選択してください。

チェックアウト状況列

チェックアウト状況列で、バージョン管理システム (VCS) にファイルをチェックインしたり、チェックアウトします。これによってコードに対する変更を簡単に追跡できます。特に複数の人間がソフトウェアプロジェクトに携わっている場合に便利です。チェックアウト状況列 ([図 3.3 \(p37\)](#)) はバージョン管理システムを使うように CodeWarrior プロジェクトを設定している場合にのみ表示されます。その他のバージョン管理システムについての詳細は、バージョン管理システムに付属のマニュアルを参照してください。

プロジェクトチェックアウト状況アイコン

これはプロジェクトが書き込み可能か否か、プロジェクトのファイルへのアクセス権を示します。バージョン管理システムが、プロジェクトファイルをチェックイン、チェックアウトしたときにアクセス権を割り当てます。

プロジェクトチェックアウト状況アイコンの詳細は「[ファイル状況を取得](#)」(p453) を参照してください。その他のバージョン管理システムの使用についての詳細は、バージョン管理システムに付属のマニュアルを参照してください。

デザインビュー

デザインビューはプロジェクトの RAD デザインを表示します。特定のデザインに属するオブジェクトとその名前を表示します。デザインビューはオブジェクト名、属性、デザイン階層を調べるのに便利です。

プロジェクトに RAD デザインが含まれている場合、ターゲットビューはデザインの階層、プロジェクト全体のビルドターゲットを表示します。1つのプロジェクトで複数のデザインを管理できます。各デザインに複数のビルドターゲットを割り当てることができます。

RAD デザインの詳細は「[RAD デザインとレイアウト](#)」(p365) を参照してください。

リンク順ビュー

CodeWarrior はリンク順ビューに表示されている順番でファイルをコンパイルします。ビルド順を明確に指定するのに便利です。例えばあるファイルが、他のまだコンパイルされていないファイルに依存している場合、コンパイルはできません。このような問題を避けるため、リンク順ビューでファイルを正しい順番に並べます。

リンク順ビューでファイル順を変えると、プロジェクトファイルが生成する最終的なバイナリコードに影響を与えます。リンク順ビューの項目は 1 レベルだけ入れ子（ネスト）できます。

リンク順については「[リンク順を設定](#)」(p258)、グループについては「[プロジェクト内のファイル管理](#)」(p53) を参照してください。

エンベデッドシステム：ビルドターゲットによってはリンク順ビューの名前はオーバーレイビューになります。「[ターゲットマニュアル](#)」(p20) を参照してください。

ターゲットビュー

ターゲットビュー（[図 3.5](#)）は、プロジェクトのビルドターゲットの一覧を表示します。また最終的な出力ファイルを生成するときにビルドターゲットが依存するオブジェクトも表示します。

プロジェクトが RAD デザインを含む場合、ビューは各デザインの中のビルドターゲットの階層も表示します。ビルドターゲットについては「[複合プロジェクトを作成](#)」(p64)、デザインについては「[RAD デザインとレイアウト](#)」(p365) を参照してください。

図 3.5 プロジェクトウィンドウのターゲットビュー



新規プロジェクトを作成

新規プロジェクトの作成方法について説明します。

[プロジェクトファイルの種類](#)

[プロジェクトステーションリを選択](#)

[新規プロジェクトの名前](#)

[新規プロジェクトダイアログを使う](#)

[新規プロジェクトの修正](#)

[新規プロジェクトのビルド](#)

プロジェクトファイルの種類

CodeWarrior で新規プロジェクトを作成するときに、以下のファイルを利用できます。

プロジェクトステーションリ：プロジェクトステーションリは、あらかじめ設定されたライブラリやソースコード、リソースファイルなどの仮ファイルを含んでいます。新規プロジェクトの作成に非常に便利です。

RAD ウィザード：RAD ウィザードはダイアログを使って RAD プロジェクトを作成します。ウィザードの使い方は「[RAD プロジェクトを作成](#)」(p367) を参照してください。

空のプロジェクト：空のプロジェクトはファイルを含みません。このプロジェクトはコンパイラやリンクの設定をカスタマイズするのに便利です。熟練したユーザーの方ならば、空のプロジェクトからカスタマイズしたアプリケーションを作成することもできます。

Makefile インポートウィザード：Makefile インポートウィザードは Visual C nmake ファイルや GNU make ファイルをパースして CodeWarrior プロジェクトを作成します。Makefile インポートについては「[プロジェクトヘメイクファイルをインポート](#)」(p47) を参照してください。

次にプロジェクトステーションリを利用する新規プロジェクトの作り方を説明します。詳細は「[プロジェクトステーションリを使う](#)」(p44) を参照してください。

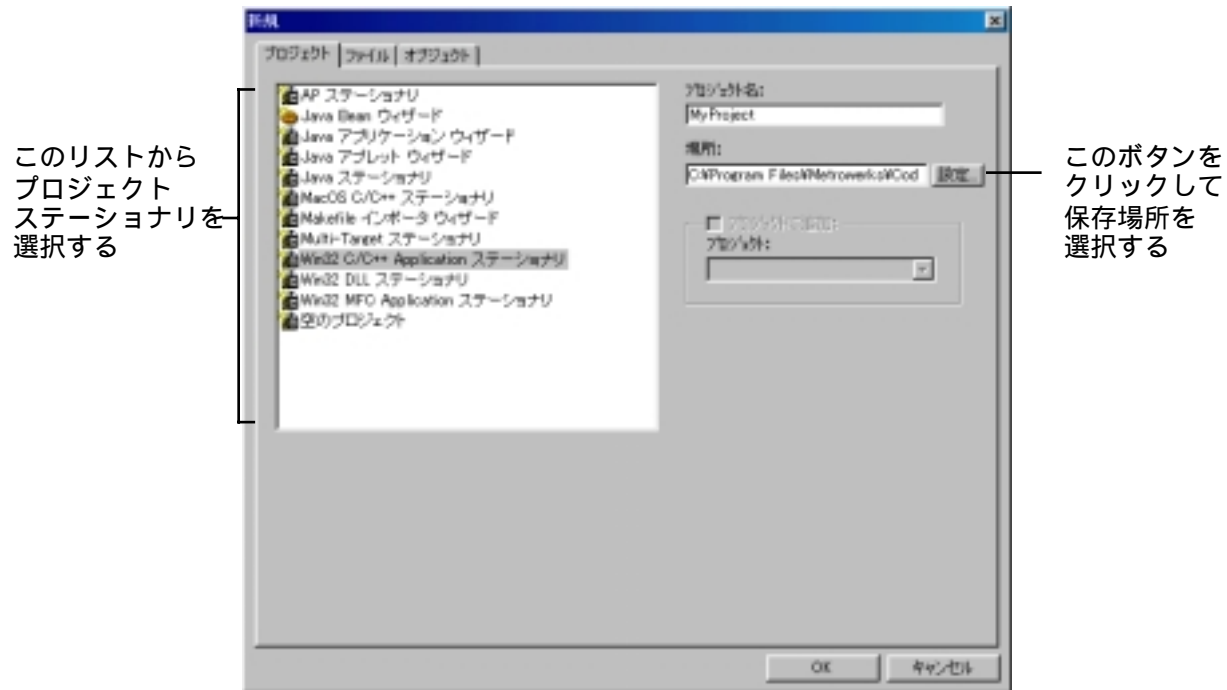
注意： 空のプロジェクトを作成する場合、「[ターゲットのオプション設定](#)」(p225) を参照してください。コンパイラ、リンクの設定について説明しています。

プロジェクトステーションリを選択

プロジェクトステーションリを利用して新規プロジェクトを作成するには、[ファイルメニュー](#)の[[新規](#)]を選択します。「新規」ダイアログが表示されます。このウィンドウからプロジェクトステーションリを選択してください。

「新規」ダイアログの[プロジェクト]タブをクリックしてプロジェクトパネルを表示します。プロジェクトパネルにはさまざまな種類のプロジェクトファイルが表示されます。ステーションナリから新規プロジェクトを作る場合、階層を開いて選択します。まずプラットフォームターゲット（Windows や Mac OS など）用のステーションナリを選択します。次にオペレーティングシステム、言語、プログラミングフレームワークを指定します。これらのオプションを指定すると、あらかじめ設定されたプロジェクトステーションナリが選択可能になります。

図 3.6 「新規」ダイアログ



プロジェクトパネルのリストから項目を選択すると、パネルの右側にオプションが表示されます。

ステーションナリから新規プロジェクトを作成するには、名前に「ステーションナリ」が付いているファイルをプロジェクトパネルから選択します。Mac OS プラットフォーム用のプロジェクトを作る場合、パネルから [MacOS ステーションナリ] という項目を選びます。

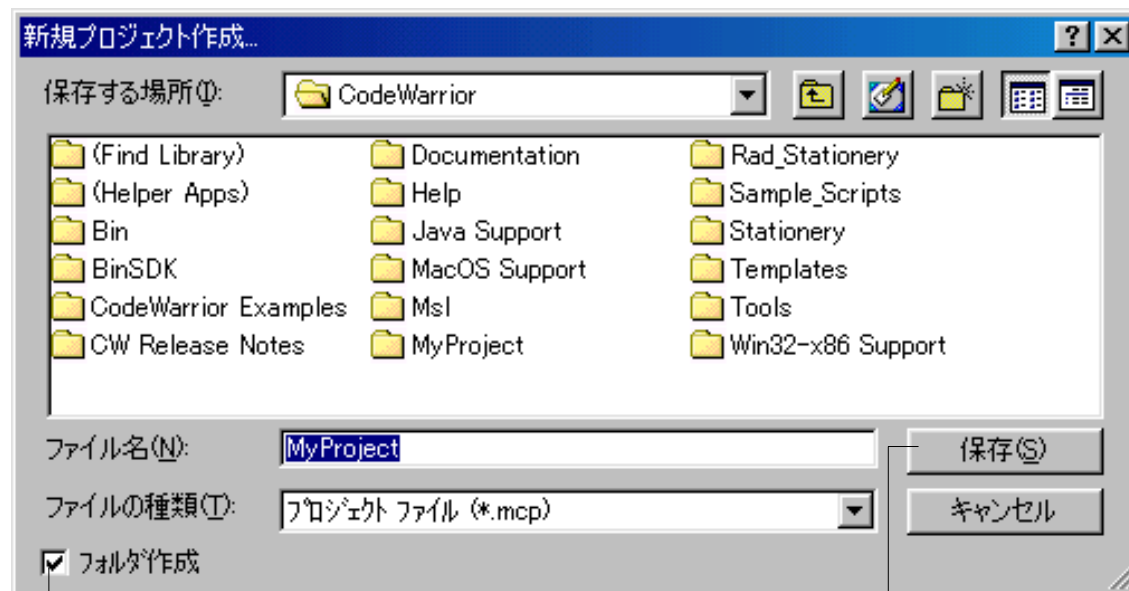
ライブラリやサポートファイルを含まない空のプロジェクトを作成するには、プロジェクトパネルで [空プロジェクト] を選択します。

初心者： 空のプロジェクトの作成はお奨めできません。正しいライブラリ、ファイルの追加や、ターゲットの設定を行うのは複雑であるためです。新規プロジェクトにはプロジェクトステーションナリを使用してください。

新規プロジェクトの名前

[ファイルメニュー](#)の[[新規](#)]を選択してプロジェクトパネルから項目を選択した後、[プロジェクト名] フィールドにプロジェクトの名前を入力します。

図 3.7 「新規プロジェクト作成」ダイアログ



プロジェクトと関連ファイルを 1 つのフォルダに保存するにはこれをオンにする

位置を決定し、「新規」ダイアログに戻るにはボタンをクリックする

ヒント： プロジェクトには拡張子 `.mcp` を付けてください（例：`MyProject.mcp`）。これによりプロジェクトファイルの見分けが簡単になります。Windows 版の CodeWarrior はこの拡張子でプロジェクトファイルを識別しています。

[場所] フィールドはプロジェクトを保存するフォルダへの完全なパスを表示します。現在のパスを変更するには、新しいディレクトリをフィールドへ入力します。[設定] ボタンをクリックしてダイアログ（[図 3.7](#)）を表示し、保存場所を指定することもできます。プロジェクトファイルと関連ファイルを保存するフォルダを新たに作成するには、[フォルダを作成] チェックボックスをオンにします。[保存] ボタンをクリックすると位置が決定し、「新規」ダイアログへ戻ります。

新規プロジェクトの名前と保存場所を指定した後、「新規」ダイアログで [OK] ボタンをクリックします。IDE は「新規プロジェクト」ダイアログを表示します。

警告！ 新規プロジェクトを保存するときに、ハードディスクに同じ名前のプロジェクトが既に存在すると、IDE はエラーメッセージを表示します。新規プロジェクトにはユニークな名前を付ける必要があります。

新規プロジェクトダイアログを使う

「新規プロジェクト」ダイアログ ([図 3.8](#)) は「新規」ダイアログでの選択肢に従ってプロジェクトステーションナリの階層を表示します。例えば「新規」ダイアログで [Win32 C/C++ Application ステーションナリ] を選択した場合、「新規プロジェクト」ダイアログは Windows プラットフォーム用のプロジェクトステーションナリの階層を表します。

図 3.8 「新規プロジェクト」ダイアログ



ステーションナリを選択したら [OK] ボタンをクリックしてください。IDE は自動的に以下の項目を設定します。

プロジェクトと同じ名前 (拡張子なし) のフォルダが作成されます ([「新規プロジェクトの名前」\(p42\)](#) の [フォルダを作成] チェックボックスがオンの場合のみ)。新しいフォルダに選択したステーションナリの新規プロジェクトファイルを保存します。

[IDE 設定](#)、[Target の設定](#) パネルを選択したステーションナリと同じに設定します。

プロジェクトウィンドウを開きます。新規プロジェクトにはライブラリ、ソースとリソースの仮ファイルが含まれます。空プロジェクトを選択した場合、プロジェクトウィンドウには何も表示されません。

例えば C++ コンソールアプリケーションを作成する場合、「新規プロジェクト」ダイアログで [C++ Console App] を選択して [OK] ボタンをクリックすると、CodeWarrior IDE は必要なライブラリとサポートファイルを含む新規プロジェクトを作成します。

新規プロジェクトの修正

ステーションから作成したプロジェクトは基本的な仮ファイルを含んでいます。仮ファイルを削除して自分のソースファイルを追加します。プロジェクト内のファイルの操作については「[プロジェクト内のファイル管理](#)」(p53) を参照してください。

プロジェクトにライブラリも追加します。必要なライブラリについては各『Targeting』マニュアルを参照してください。「[ターゲットマニュアル](#)」(p20) を参照してください。

新規プロジェクトのビルド

プロジェクトを作成してファイルを追加した後、ビルドしてソフトウェアを作ります。プロジェクトのビルド方法は「[コンパイルとリンク](#)」(p255) を参照してください。

プロジェクトステーションを使う

ここでは、プロジェクトステーションの作り方と使い方について説明します。「[新規プロジェクトを作成](#)」(p40) で説明したプロジェクトステーションのカスタマイズについても説明します。

CodeWarrior プロジェクトは、複数のビルドターゲットを持つように設定でき、サブプロジェクトを含むことができます。詳細は、この節を読んだ後で「[複合プロジェクトを作成](#)」(p64) を参照してください。

以下の項目を説明します。

[プロジェクトステーションについて](#)

[プロジェクトステーションのフォルダについて](#)

[独自のプロジェクトステーションの作成](#)

プロジェクトステーションについて

プロジェクトステーションファイルは、最小限のスタートプロジェクトファイルです。新規プロジェクトを簡単に作成するためのテンプレート、または空の予定表と考えることもできます。新規プロジェクトを作成するとき、またはプロジェクトステーションファイルを開くときに、CodeWarrior IDE は新規プロジェクトを作成し、オプションとして新規プロジェクトフォルダを作成します。そして新しいフォルダにステーション情報ファイルをすべてコピーします。

ステーションプロジェクトには、以下が含まれます。

プロジェクトのすべてのオプション設定

プロジェクトステーションに含まれるすべてのファイル（ライブラリ、ソースファイル、およびリソースファイル）

ステーションナリから新規プロジェクトを作成した後、これを開いて CodeWarrior IDE で開発を行います。

プロジェクトステーションナリのフォルダについて

CodeWarrior はさまざまな種類のプロジェクトステーションナリを用意しています。プロジェクトステーションナリは専用のフォルダにあります。

以下のファイルは Stationery フォルダ、または (Project Stationery) フォルダに置くことができ、プロジェクトステーションナリとして認識されます。

通常に保存されたプロジェクト

プロジェクトステーションナリファイル

プロジェクトステーションナリは以下のフォルダにあります。

CodeWarrior フォルダの中の Stationery フォルダ

独自のプロジェクトステーションナリの作成

スタータプロジェクトとして使用するファイル、オプションを含む独自のステーションナリまたは「テンプレート」プロジェクトファイルを作成できます。このプロジェクトステーションナリは、新規プロジェクトを作るたびに再利用できるので、常に独自にカスタマイズした設定から開始できます。

どのような CodeWarrior プロジェクトもステーションナリプロジェクトになり得ます。以下はステーションナリとなるための条件です。

プロジェクトは project stationery フォルダにあること

プロジェクトに関連するソースファイルを一緒に保存すること

「新規プロジェクト」ダイアログでプロジェクトステーションナリを選択すると、CodeWarrior はステーションナリのプロジェクトとソースファイルをコピーして、新規プロジェクト名を付けます。

初心者： 独自のプロジェクトステーションナリを作成する前に、CodeWarrior に含まれるプロジェクトステーションナリを見て、ファイルの重要性を理解してください。

以下は独自のプロジェクトステーションナリファイルを作成する手順です。

1. 既存のプロジェクトステーションナリから新規プロジェクトを作成する、または空のプロジェクトを作る
2. ファイルメニューの [コピーを保存] を選択する


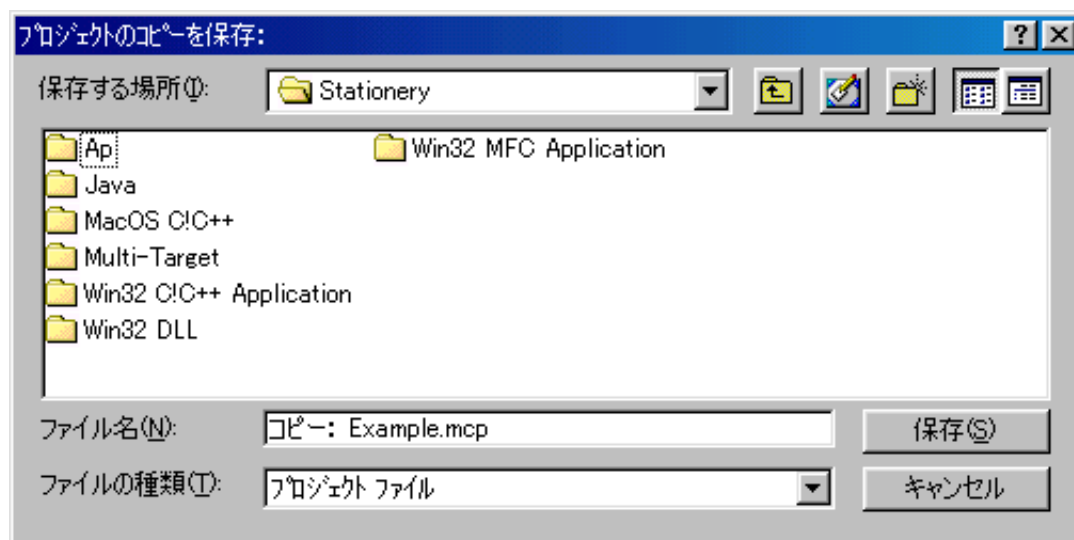
ダイアログ ( 3.9) が現れます。

図 3.9 「プロジェクトのコピーを保存」ダイアログ



3. ダイアログで新規プロジェクトの保存場所をプロジェクトステーションフォルダに指定する

新規プロジェクトを保存するフォルダを必ず作成してください。CodeWarrior IDE はプロジェクトステーションをそのフォルダで識別します。Windows 版の IDE を使用している場合、プロジェクトファイルの名前に拡張子 `.mcp` を忘れずに付けてください。

4. 必要に応じてプロジェクトの設定を修正する

希望通りのプロジェクトを作るために、必要に応じてファイルを追加、または削除してください。

5. プロジェクトのソースファイルのコピーがそのフォルダにあることを確認する

フォルダにソースファイルがなければ、プロジェクトステーションから作ったプロジェクトにファイルはコピーされません。

6. プロジェクトの修正後、保存する

すべての変更を保存した後、プロジェクトは新しいプロジェクトステーションファイルとして利用可能になります。

独自のステーションファイルをプロジェクトステーションフォルダに保存した後、次に [ファイルメニュー](#) の [[新規](#)] を選択したときからそのステーションを選択できます。

独自のステーションプロジェクトから新規プロジェクトを作成すると、プロジェクトステーションに関連するファイルがすべて新規プロジェクトにコピーされ保存されます。

[[IDE 設定](#)] の詳細は「[IDE 設定パネルを選択](#)」(p183) を参照してください。同様に、独自のステーション用に [[Target の設定](#)] もカスタマイズできます。詳しくは「[設定パネルを選択](#)」(p228) を参照してください。

プロジェクトにファイルを追加または変更する方法は「[プロジェクト内のファイル管理](#)」(p53)を参照してください。

プロジェクトのコピーを別の名前で保存する方法は「[ファイルのコピー保存](#)」(p83)を参照してください。

作業を進める前にステーションナリプロジェクトを保存する理由は、ハードディスクに「スタート」または「テンプレート」プロジェクトファイルを作ることです。「スタート」プロジェクトのコピーを作成して、好みの設定で新規プロジェクトを迅速に開始できます。

ステーションナリを使って作成した新規プロジェクトは、プロジェクトステーションナリと同じに設定されています。

新規プロジェクトに対して別の設定を使いたい場合、新しいプロジェクトステーションナリを作成します。プロジェクトウィンドウを開き、オプションを設定した後、適切なステーションナリフォルダに新しいステーションナリを保存してください。

プロジェクトへメイクファイルをインポート

CodeWarrior IDE は Visual C `nm`make ファイルまたは GNU `make` ファイルを CodeWarrior プロジェクトファイルへインポートできます。IDE は Makefile インポートウィザードを使って以下の操作を行います。

- メイクファイルをパースする

- CodeWarrior プロジェクトを作成する

- ビルドターゲットを作成する

- メイクファイルで指定されているソースファイルを追加する

- メイクファイルで指定されている情報と出力名、出力ディレクトリ、ビルドターゲットへのアクセスパスを一致させる

- プロジェクト用のリンクを選択する

以下の項目について説明します。

- [Makefile インポートウィザードを使う](#)

- [Makefile インポートのオプション](#)

Makefile インポートウィザードを使う

メイクファイルから新規プロジェクトを作るには、プロジェクトメニューの[[新規](#)]を選択します。「新規」ダイアログが現れます([図 3.6 \(p41\)](#))。[プロジェクト] タブをクリックして、プロジェクトパネルのリストから[Makefile インポートウィザード]を選択してください。

次に命名規約に従って新規プロジェクトに名前を付け、保存します。命名規約については「[新規プロジェクトの名前](#)」(p42)を参照してください。

プロジェクトの命名と保存場所の指定を終えたら、「新規」ダイアログの[OK]ボタンをクリックしてください。Makefile インポータウィザードが現れます。このオプションの設定については「[Makefile インポータのオプション](#)」(p48)を参照してください。

Makefile インポータのオプション

Makefile インポータウィザード(図 3.10)には変換方法をカスタマイズするためのオプションがあります。

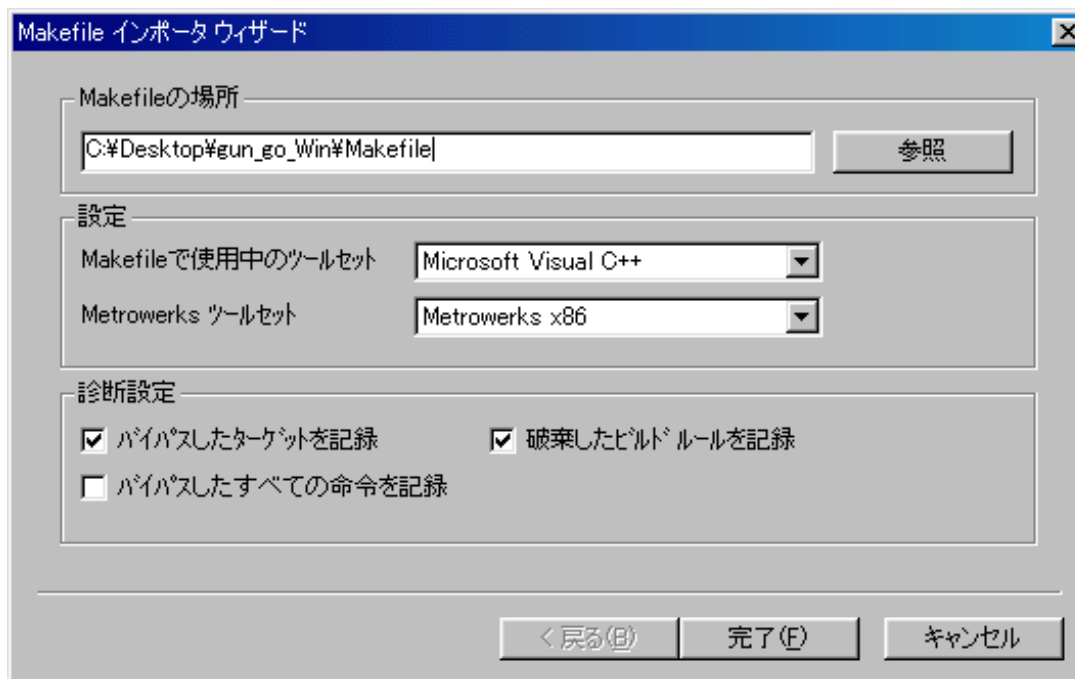
[Makefile の場所](#)

[設定](#)

[診断設定](#)

これらのオプション設定を終えたら、[完了]ボタンをクリックします。Makefile インポータウィザードがサマリウィンドウを表示します。このウィンドウは現在の変換方法の設定を表示します。この設定でメイクファイルをインポートするならば[完了]ボタンをクリックします。設定を変更する、またはウィンドウへ戻るならば[キャンセル]ボタンをクリックします。

図 3.10 Makefile インポータウィザード



Makefile の場所

[Makefile の場所] フィールドに、CodeWarrior プロジェクトへインポートするメイクファイルの位置を入力します。または [参照] ボタンをクリックしてダイアログを開き、ここでメイクファイルを選ぶこともできます。メイクファイルを選択したら [開く] ボタンを

クリックしてください。[ファイルの場所] フィールドにメイクファイルへのパスが表示されます。

設定

[設定] 欄でメイクファイルの変換に使用するツールと新規プロジェクト用のリンクを指定します。2つのポップアップメニューがあります。

[Makefile で使用中のツールセット] ポップアップメニュー：メイクファイルで使われているツールを選択する

[Metrowerks ツールセット] ポップアップメニュー：Makefile インポートが作成した CodeWarrior プロジェクトに使用するリンクを選択する

診断設定

[診断設定] 欄のチェックボックスで、変換過程で生成される各種の情報をログするか否かを指定します。

[バイパスしたターゲットを記録] チェックボックス：オンの場合、CodeWarrior ビルドターゲットに変換されなかったビルドターゲット（メイクファイルでパースされたもの）に関する情報をログします。

[廃棄したビルドルールを記録] チェックボックス：オンの場合、CodeWarrior プロジェクトへの変換中に削除されたメイクファイルのビルドルールに関する情報をログします。

[バイパスしたすべての命令を記録] チェックボックス：オンの場合、[バイパスしたターゲットを記録] と [廃棄したビルドルールを記録] と同じ情報をログしますが、パース中に解析できなかったメイクファイルの他の項目についての情報もログします。

メッセージは変換作業が終了したときにプロジェクトメッセージウィンドウに表示されます。メッセージウィンドウと似ています。詳細は [「メッセージウィンドウの解説」\(p269\)](#) を参照してください。

既存のプロジェクトを開く

CodeWarrior IDE からプロジェクトファイルを開く方法は複数あります。ここではその方法を説明します。これで作業を始めることができます。

一度に複数のプロジェクトを開くことができます。[「デフォルトのプロジェクトを選択」\(p53\)](#) を参照してください。

開いているプロジェクトウィンドウは、[ウィンドウメニュー](#)を使用してアクティブなウィンドウにすることができます。他の開いているプロジェクトに切り替えるには、このメニューからプロジェクトを選択します。

ここでは、以下の項目を説明します。

[開くコマンドを使う](#)

[過去のファイルを開くコマンドを使う](#)

[プロジェクトウィンドウでサブプロジェクトを開く](#)

[他のプラットフォームで作成したプロジェクトファイルを開く](#)

[古いバージョンのプロジェクトファイルを開く](#)

開くコマンドを使う

プロジェクトファイルを開くには[ファイルメニュー](#)の[[開く](#)]を選択します。「開く」ダイアログ ([図 4.1 \(p78\)](#)) が現れます。

[ファイルの種類] ポップアップメニューで [プロジェクトファイル] を選択します。ここには開くことのできるプロジェクトファイルだけが表示されます。

開きたいプロジェクトファイルを選択し、[開く] ボタンをクリックします。IDE がプロジェクトを開き、プロジェクトウィンドウが表示されます。

古いバージョンの CodeWarrior で作成したプロジェクトは、更新が必要であるというメッセージが表示されます。IDE はプロジェクトのバックアップコピーを作成してから更新を行います。「[古いバージョンのプロジェクトファイルを開く](#)」(p51) を参照してください。

CodeWarrior プロジェクトファイルの詳細は「[プロジェクト](#)」(p31) を参照してください。

過去のファイルを開くコマンドを使う

CodeWarrior IDE は、[ファイルメニュー](#)の下に最近使ったプロジェクトを表示します。[[過去のファイルを開く](#)]を使用してこれらのプロジェクトの 1 つを再び開くことができます。

このメニューに記憶するファイル数の設定方法は「[IDE その他設定](#)」(p185) を参照してください。

プロジェクトウィンドウでサブプロジェクトを開く

プロジェクトに含まれるサブプロジェクトの 1 つを開くには、プロジェクトウィンドウでプロジェクトファイルアイコンをダブルクリックします。IDE はサブプロジェクトをプロジェクトウィンドウに表示します。

サブプロジェクトの詳細とプロジェクトウィンドウへの追加方法は「[複合プロジェクトを作成](#)」(p64) を参照してください。

他のプラットフォームで作成したプロジェクトファイルを開く

プロジェクトファイルはクロスプラットフォーム互換です。つまり、Mac OS コンピュータ上で作成したプロジェクトファイルを、Windows コンピュータ上で開くことができます。

他のプラットフォームで作成されたプロジェクトを使うには、自分のコンピュータ上へそのプロジェクトファイルだけをコピーします (データフォルダはコピーしません)。プロジェクトをコピーした後、それを IDE で開き、ファイルを再コンパイルします。プロジェ

クトのフォーマットはクロスプラットフォーム互換ですが、コンパイルされたオブジェクトコードには互換性はありません。

注意： プロジェクトをコピーする前に、プロジェクト名に拡張子 `.mcp` が付いていることを確認してください。プロジェクトの名前が「MyProject」ならば、「MyProject.mcp」へ変更してください。Windows 版の CodeWarrior IDE はファイル拡張子でプロジェクトファイルを識別します。3 文字の拡張子がなければ Windows 版 IDE はプロジェクトファイルを識別できません。

ヒント： プロジェクトファイルの名前に必ず拡張子 `.mcp` を付けてください。他のプラットフォームへの移植が容易になります。

他のプラットフォームで作成したソースファイルの編集については「[オプションポップアップメニュー](#)」(p99) を、ホストプラットフォームへのアクセスパスの設定については「[ホストフラグ](#)」(p234) を参照してください。

古いバージョンのプロジェクトファイルを開く

バージョン 3.0 以降の CodeWarrior IDE では、1.7 以前のバージョンのプロジェクトファイルは使用できません。新しい IDE でプロジェクトファイルを作り直すか、新しい IDE 用にプロジェクトファイルを変換しなくてはなりません。プロジェクトの変換方法を説明します。

1.7 のプロジェクト（単数）を変換する

CodeWarrior IDE には、1.7 バージョンのプロジェクトファイルを最新のフォーマットに変換するユーティリティがあります。

最新バージョンの IDE で 1.7 プロジェクトを開きます。IDE はプロジェクトを最新のフォーマットに変換するというメッセージを表示します。変換されたプロジェクトファイルは元のファイルに基づいた名前が付けられます。

最新バージョンの IDE で 1.7 プロジェクトを開きます。IDE 4.0 は自動的に Project Converter を実行し、最新のフォーマットに変換します。

1.7 バージョンのプロジェクトファイルを Project Converter ヘドラッグ＆ドロップします。

注意： 1.7.x プロジェクトは新しいフォーマットにコンバートしなければ、最新バージョンの IDE で使うことはできません。最新フォーマットへの変換は永続的なもので、最新バージョンの IDE は、1.7.x フォーマットでプロジェクトを保存することはできません。

プロジェクトを保存

CodeWarrior IDE は、ある操作を実行するとプロジェクトを自動的に更新し保存します。ここでは、プロジェクトファイルを保存するための操作を説明します。

設定が保存されるのは、以下の場合です。

- プロジェクトを閉じる

- プロジェクトの [IDE 設定](#) または [Target の設定](#) を変更する

- プロジェクトへファイルを追加または削除する

- プロジェクト内のファイルをコンパイルする

- プロジェクト内のグループを編集する

- プロジェクトからオブジェクトコードを削除する

- CodeWarrior IDE を終了する

プロジェクトは上記の操作が実行されるたびに自動的に保存されるので、プロジェクトのコピーを作成するなどして手作業でプロジェクトを保存する必要はありません。

プロジェクトと共に保存される項目

CodeWarrior IDE はプロジェクトを保存するときに以下の情報も保存します。

- プロジェクトに追加されたファイルの名前とその位置

- 設定されたすべてのオプション

- 従属情報（タッチ状態およびヘッダファイルのリストなど）

- ブラウザ情報

- コンパイルしたソースファイルのオブジェクトコード

プロジェクトのコピーを保存

プロジェクトファイルを変更する前にバックアップコピーを取るには、[ファイルメニュー](#)の[[コピーを保存](#)]([「ファイルのコピー保存」](#))を使います。新しい名前でプロジェクトのコピーを作成できます。オリジナルのファイルは変更されません。

警告！ プロジェクトを開いたままでコピーしないでください。ファイルの破損を避けるために必ずプロジェクトを閉じてください。

プロジェクトを閉じる

プロジェクトでの作業を終えたら、他のプロジェクトに移るため、またはCodeWarrior IDEを終了して他の作業を行うために、プロジェクトを閉じることになります。

プロジェクトを閉じるには、プロジェクトウィンドウをアクティブにしてから [ファイルメニュー](#) の [[閉じる](#)] を選択します。

プロジェクト設定は自動的に保存されるので、CodeWarrior IDE を終了する前にプロジェクトを閉じる必要はありません。保存されたプロジェクトの詳細は「[プロジェクトを保存](#)」(p52) を参照してください。

CodeWarrior IDE では一度に複数のプロジェクトを開くことができますが、プロジェクトを切り替えるときに他のプロジェクトを閉じる必要はありません。新規プロジェクトを開いて作業を始めるだけです。

ヒント： 一度に複数のプロジェクトを開くとコンピュータのメモリ使用量が多くなり、プロジェクトを開くのに少し時間がかかります。複数のプロジェクトを扱うときにはこの点に注意してください。

デフォルトのプロジェクトを選択

CodeWarrior IDE では複数のプロジェクトを開けるので、[[メイク](#)] [[実行](#)] またはその他の操作を実行する際に、どのプロジェクトを使用しているのかが分かりにくくなります。IDE は現在アクティブなプロジェクトウィンドウのプロジェクトファイルに対して操作を実行します。

1 つのソースファイルが複数のプロジェクトに含まれていることもあります。[プロジェクトメニュー](#) の [[デフォルトプロジェクトを設定](#)] を使ってビルド用のデフォルトのプロジェクトを指定することができます。ソースファイルが複数の開いているプロジェクトに含まれている場合、どのプロジェクトの設定を使ってビルドするのが曖昧になります。このような場合、CodeWarrior IDE は [[デフォルトプロジェクトを設定](#)] コマンドで選択されているデフォルトプロジェクトを使ってビルドを行います。

最初を開いたプロジェクトがデフォルトプロジェクトになります。デフォルトプロジェクトを閉じると、次に最前面のプロジェクトウィンドウのプロジェクトがデフォルトプロジェクトになります。

プロジェクト内のファイル管理

ここでは、プロジェクトのファイルの追加、移動、名前付け、編成、表示、コンパイルのためのマーク付け、および削除について説明します。ここでは、以下の項目を説明します。

[グループの拡張と縮小](#)

[ファイルとグループを選択](#)

[ファイルを追加](#)

[ファイルやグループを移動](#)

[グループの作成](#)

[ファイルやグループの削除](#)

[グループ名の変更](#)

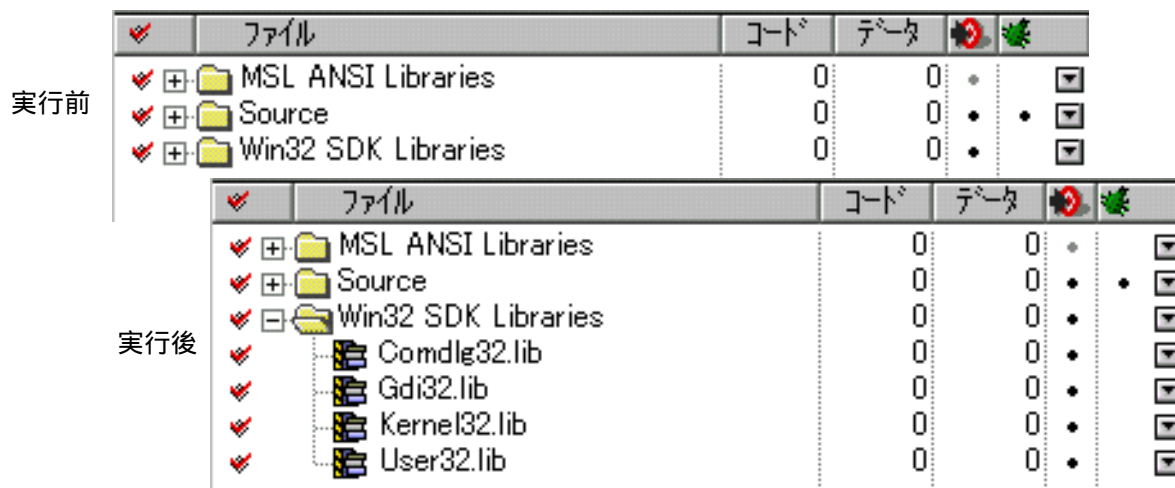
[ファイルのタッチ / アンタッチ](#)

グループの拡張と縮小

グループを拡張または縮小して一覧表示することができます。グループを拡張してファイルを表示するには、グループ名の左にある拡張ボタンをクリックします。グループを閉じて名前だけを表示するには拡張ボタンを再度クリックします。

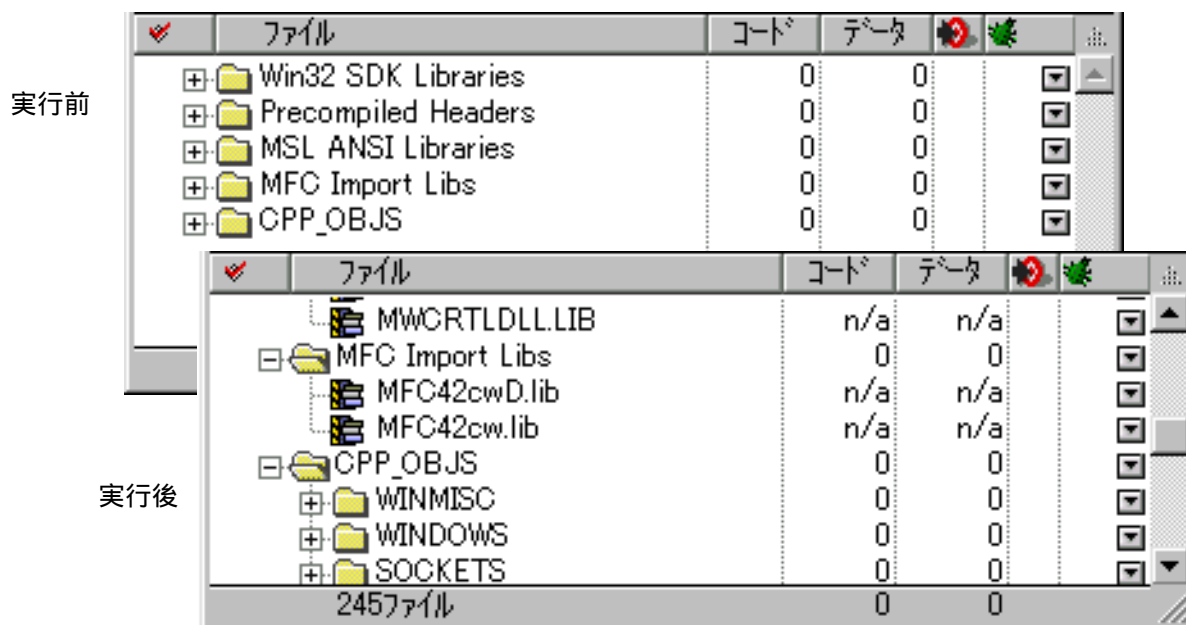
プロジェクトウィンドウでグループ名を Alt + クリックすると、グループとそのサブグループを拡張表示できます ([図 3.11](#))。選択したグループと同じ階層にあるグループは拡張しません。再度 Alt + クリックするとグループを縮小します。

図 3.11 グループとサブグループを拡張



Ctrl + クリックで、プロジェクトウィンドウの同じ階層にある全グループを拡張表示できます ([図 3.12](#))。再度 Ctrl + クリックするとグループを縮小します。

図 3.12 同じ階層のグループを拡張



ファイルとグループを選択

プロジェクトウィンドウでファイルやグループ（複数も可）を選択して、コンパイル、文法の検査、削除、別のグループへ移動することができます。

グループを選択すると、グループ内の個々のファイルがプロジェクトウィンドウ上で選択されているかどうかにかかわらず、グループ内のすべてのファイルが選択されます。

マウスのクリックによる選択

プロジェクトウィンドウ内で 1 つのファイルまたはグループを選択するには、その名前をクリックします。

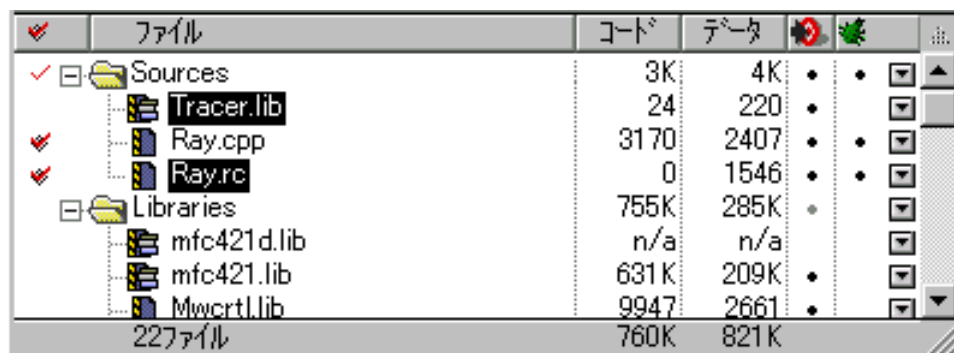
連続したファイルまたはグループを選択するには、最初のファイルまたはグループをクリックした後、最後のファイルまたはグループを Shift + クリックします。

最初にクリックしたファイルまたはグループと、最後に Shift + クリックしたファイルまたはグループの間にあるものが、グループ、ファイルに関係なくすべて選択されます。

デスクトップ上の操作同様に、クリック + ドラッグで連続した項目を選択することもできます。

Ctrl + クリックで複数の連続しないファイルを選択、または選択解除することができます（[図 3.13](#)）。

図 3.13 連続しないファイルを選択



キーボードによる選択

キーボードを使って選択するには、選択したいファイル名の最初の数文字を入力します。入力した文字に最も近いファイルが識別され、プロジェクトウィンドウでファイルが選択されます。

入力ミスしたときは、Backspace キーを押します。

ファイルを開くには、Enter キーを押します。

注意： キーボードで選択できるのは、プロジェクトウィンドウで現在拡張されているグループにあるファイルだけです。縮小されているグループ内のファイルは文字入力では選択できません。

ファイルを追加

ここでは、CodeWarrior プロジェクトにファイルを追加する方法を説明します。

ここで説明する項目を以下に示します。

[ファイルの保存場所](#)：プロジェクトに追加したファイルが保存される場所を説明します。

[ファイルを追加コマンドを使う](#)：1つまたは複数のファイルの追加方法を示します。

[ドラッグ&ドロップ](#)：1つまたは複数のファイルの追加方法を示します。

[ウィンドウを追加コマンドを使う](#)：1つのファイルの追加方法を示します。

プロジェクトへファイルを追加すると、そのファイルへの[アクセスパス](#)が自動的に設定されます。アクセスパスが設定されるときにメッセージウィンドウが現れます。

注意： 空のプロジェクトからプロジェクトを作成した場合、[「リンク」\(p230\)](#) および [「ファイルマッピング」\(p239\)](#) を参照してください。ファイルの追加方法を説明しています。

通常 IDE はプロジェクト内のファイルがユニークな名前を持つように強制します。この制限をはずすには、[ターゲット設定](#)の [\[相対パスを使ってプロジェクトエントリを保存 \]](#) オプションをオンにしてください。

アクセスパスの詳細は [「アクセスパス」\(p231\)](#) を参照してください。

メッセージウィンドウの詳細は [「メッセージウィンドウの解説」\(p269\)](#) を参照してください。

ファイルの保存場所

ファイルは、プロジェクトウィンドウ上で現在選択されている項目の後に追加されます。何も選択されていない場合、プロジェクトウィンドウの最後に追加されます。新しいファイルを特定の位置に置きたい場合は、[\[ファイルを追加 \]](#) または [\[ウィンドウを追加 \]](#) コマンドを実行する前に、望みの位置のすぐ上のファイルまたはグループを選択してください。

グループを選択すると、グループが拡張されているか縮小されているかにかかわらず、ファイルは選択したグループの最後に置かれます。ファイルまたはグループの選択方法については、[「ファイルとグループを選択」\(p55\)](#) を参照してください。

注意： ファイルを追加する前にグループ内のグループやファイルが選択されていなければ、新しいグループが作成されてプロジェクトに追加されます。追加されたファイルは、この新しいグループに置かれます。

プロジェクトに追加した後でも、ファイルやグループを望みの位置に移動することができます。プロジェクトウィンドウ内でファイルやグループを移動する方法は [「ファイルやグループを移動」\(p61\)](#) を参照してください。

ファイルを追加コマンドを使う

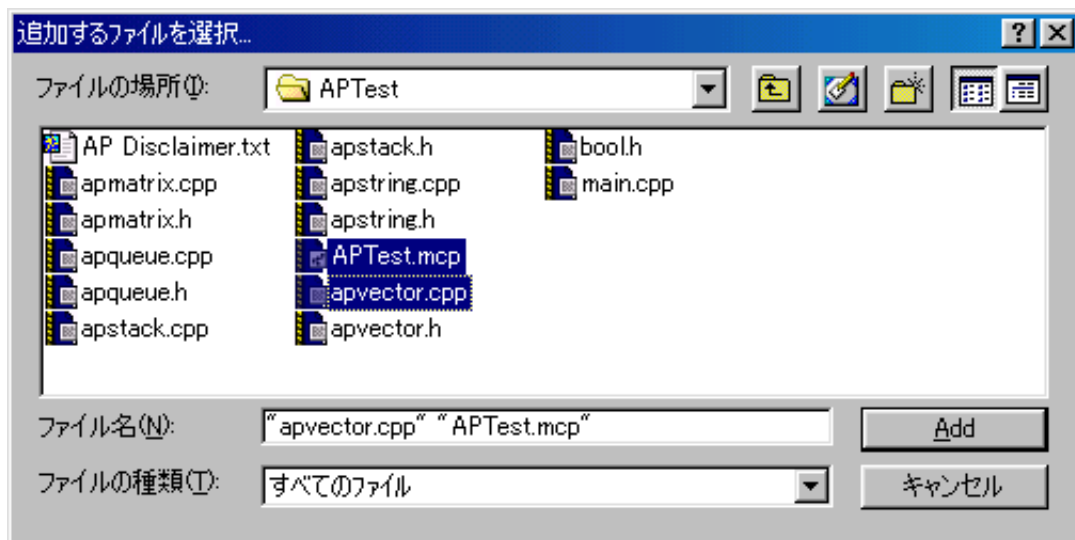
[プロジェクトメニュー](#)の [\[ファイルを追加 \]](#) は、プロジェクトにファイルを追加するときにダイアログを表示します。このコマンドを使って、ソースファイル、ライブラリ、リソースファイル、または共有ライブラリを追加できます。

この「ファイルを追加」ダイアログにファイルを表示するには、ファイル名にファイルマッピング設定パネルで指定した拡張子が付いていなくてはなりません。ファイルのマッピングについては、[「ファイルマッピング」\(p239\)](#) を参照してください。

「追加するファイルを選択」ダイアログ

「追加するファイルを選択」ダイアログ ([図 3.14](#)) の [\[ファイルの種類 \]](#) ポップアップメニューはすべてのファイルタイプを表示するように設定されています。

図 3.14 ファイルをプロジェクトへ追加



このダイアログでフォルダやドライブを検索して追加したいファイルを指定します。

プロジェクトへ追加したいファイルを選択して [Add] ボタンをクリックします。ファイルをダブルクリックしても追加できます。

複数のファイルを選択するには、ファイル名を Ctrl+ クリックします ([図 3.14](#))。

連続する複数のファイルを選択するには、最初のファイルをクリックし、次に最後のファイルを Shift + クリックします。

プロジェクトへ追加したいファイルをすべて選択したら [Add] ボタンをクリックします。プロジェクトの複数のビルドターゲットがある場合、どのビルドターゲットへファイルを追加するのかを尋ねられます ([図 3.15 \(p60\)](#))。選択したファイルを検索し、プロジェクトへ追加するまでに時間がかかることもあります。

[キャンセル] ボタンをクリックすると、ファイルは追加されず、ダイアログが閉じます。

ドラッグ&ドロップ

ファイルまたはフォルダをプロジェクトウィンドウにドラッグ & ドロップすると、プロジェクトに追加されます。

この方法でファイルを追加するには、まずプロジェクトに追加したいファイルまたはフォルダを選択します。

ファイル選択は、デスクトップ、CodeWarrior IDE の「検索」ダイアログの複数ファイル検索リストなど、いろいろな場所で可能です。

追加操作を完了するには、選択したファイルをプロジェクトウィンドウにドラッグします。プロジェクトに複数のビルドターゲットがある場合、ファイルを追加すべきターゲットを指定するよう要求されることに注意してください。

選択したファイルをプロジェクトウィンドウにドラッグしたとき、CodeWarrior IDE はそのファイルがプロジェクトに追加できるかどうかを検査します。フォルダをドラッグしたときは、CodeWarrior IDE はそのフォルダまたはサブフォルダが最低 1 つのソースファイル、ライブラリ、リソースファイルを含んでおり、そのファイルがまだプロジェクト内にないことを検査します。

選択したものに CodeWarrior IDE が認識できるファイルが 1 つも含まれていない場合、これらをプロジェクトに追加することはできません。

注意： IDE が認識できるファイルについては「[リンカ](#)」(p230)、「[ファイルマッピング](#)」(p239) を参照してください。

フォーカスバー（アンダーライン）を使ってプロジェクトウィンドウのどこにファイルを追加するのかを指定します。

マウスボタンを離す（ファイルをドロップする）と、ドラッグした項目がプロジェクトに追加されます。フォーカスバーで指定した場所に挿入します。プロジェクトファイルに複数のビルドターゲットがある場合、ダイアログ（[図 3.15](#)）が開きます。ここでどのターゲットにファイルを追加するのかを指定します。

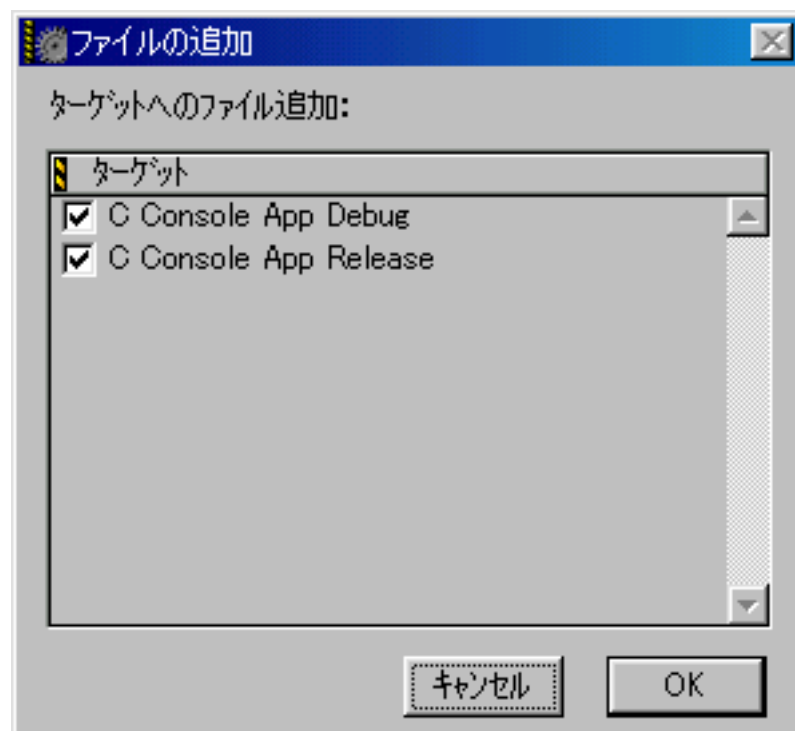
新しいグループを作成してファイルを追加するには、最後のグループの後の空白のスペースにカーソルがあるときにマウスを離します。

CodeWarrior IDE では、ボリューム（ハードディスクなど）全体をプロジェクトウィンドウにドラッグすることはできません。

CodeWarrior IDE は、プロジェクトウィンドウの外へのドラッグ & ドロップもサポートしています。ファイルを他のアプリケーションにドラッグして、そのアプリケーションで開くこともできます。

CodeWarrior IDE はプロジェクトウィンドウへのドラッグ & ドロップはサポートしていますが、プラットフォームによってはファイルをプロジェクトウィンドウの外へドラッグすることはできません。プロジェクトウィンドウからのファイルの削除については、「[ファイルやグループの削除](#)」(p61) を参照してください。

図 3.15 ファイルの追加ダイアログ



ウィンドウを追加コマンドを使う

[[ウィンドウを追加](#)] コマンドは、アクティブなエディタウィンドウで開いているファイルをプロジェクトに追加します。新しいファイルをアクティブなプロジェクトに追加したい場合に便利です。

[[ウィンドウを追加](#)] コマンドを使うには、プロジェクトウィンドウ内の位置を選択します。その後、ソースまたはテキストファイルを開き、そのウィンドウがアクティブであることを確認してから [プロジェクトメニュー](#) の [[ウィンドウを追加](#)] を選択します。ウィンドウが名称未設定の場合、「[名前を付けて保存](#)」ダイアログが現れ、ファイルの位置と名前の指定を要求します。ファイルを保存すると、ファイルは開いているプロジェクトに追加されます。

プロジェクトに複数のビルドターゲットがある場合、ファイルを追加すべきターゲットの指定を要求されます ([図 3.15](#))。

注意： [[ウィンドウを追加](#)] コマンドは、アクティブなウィンドウがテキストファイルで、そのファイルがまだプロジェクトになく、認識可能なファイル拡張子が付いているときのみ有効になります (ファイル拡張子の設定については、「[ファイルマッピング](#)」(p239)を参照してください)。そうでない場合、[[ウィンドウを追加](#)] コマンドは薄く表示されません。

ファイルやグループを移動

ファイルビューでファイルまたはグループ（複数も可）を移動する、またはターゲットビューでビルドターゲットを並べ換えるには、まずファイルまたはグループを選択します。グループ内のファイルは、プロジェクトウィンドウで選択されているかどうかにかかわらず、すべて選択されます。ファイルやグループの選択操作の詳細は「[ファイルとグループを選択](#)」(p55) を参照してください。

次に選択したファイルまたはグループを、プロジェクト内の新しい位置にドラッグします。

フォーカスバー（アンダーライン）のある位置に、マウスボタンを離したときに選択したファイルが移動されます。

ファイルまたはグループのどちらが移動されるかは、選択によります。例えばファイルを選択した場合、グループおよびファイルの両方のリストにフォーカスバーが表示されます。

少なくとも1つのグループを選択している場合、フォーカスバーはグループのリストにのみ表示されます。これにより、グループを再設定できます。

最後に、フォーカスバーが希望の位置に表示されているときにマウスボタンを離します。選択したファイルまたはグループがその位置に移動します。

ヒント：フォーカスバーの左の小さな矢印が階層への挿入位置を示します。矢印がグループアイコンの左側を指している場合、そのグループと同じ階層に挿入されます。矢印がアイコンの右側を指している場合、グループの中に挿入されます。

グループの作成

プロジェクトウィンドウを最前面にして、ファイルビューを表示します。プロジェクトメニューの「[新規グループを作成](#)」を選択します。「グループ作成」ダイアログでグループの名前を入力し、[OK] ボタンをクリックします。グループ名の変更は「[グループ名の変更](#)」(p62) を参照してください。

エンベデッドシステム：新しいグループは「グループ」ではなく「オーバーレイ」になります。

ファイルやグループの削除

プロジェクトウィンドウからファイルを削除するには、メニューコマンドまたはドラッグ & ドロップを使います。

メニューコマンドを使う

プロジェクトウィンドウのファイルビュー、またはリンク順ビューでファイルを削除することができます。その際に注意すべき点があります。ファイルビューでファイルを削除すると、全ビルドターゲットを含むプロジェクト全体から削除されます。リンク順ビュー（ターゲットによってはセグメントビューまたはオーバーレイビュー）からファイルを削除すると、ファイルは現在のビルドターゲットから削除されるだけです。

ビルドターゲットの詳細は「[ビルドターゲットとは](#)」(p64) を参照してください。

複数のファイルまたはグループを削除するには、まずファイルビューで削除するファイルまたはグループを選択します。グループを選択すると、プロジェクトウィンドウ内の表示にかかわらず、グループ内のファイルがすべて選択されることに注意してください。

ファイルの選択方法は「[ファイルとグループを選択](#)」(p55) を参照してください。

ファイルまたはグループを選択した後、[編集メニュー](#)の[[削除](#)]を選択するか、Delete キーを押します。選択されたファイルおよびグループがすべてプロジェクトから削除されます。

警告! ファイルまたはグループの削除は取り消すことができません。間違ってグループやファイルを削除すると[プロジェクトメニュー](#)の[[ウィンドウを追加](#)]または[[ファイルを追加](#)] コマンドを使って、ファイルを再度追加しなければなりません。

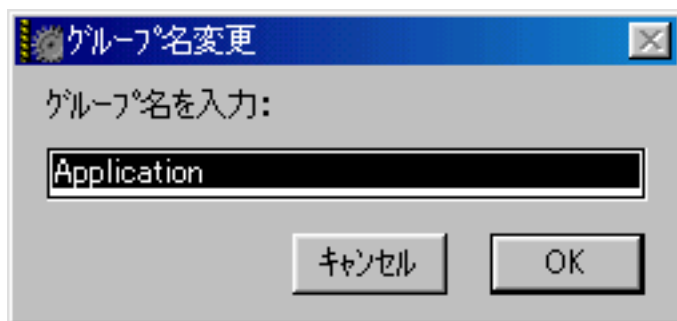
ドラッグ&ドロップを使う

CodeWarrior IDE はプロジェクトウィンドウへのドラッグ&ドロップをサポートしますが、プラットフォームによってはプロジェクトウィンドウの外へファイルをドラッグすることはできません。

グループ名の変更

グループ名を変えるには、グループをクリックしてから Enter キーを押します。矢印キーでもグループを選択することができます。「グループ名を変更」ダイアログ ([図 3.16](#)) が現れるので [グループ名] フィールドに新しい名前を入力します。

図 3.16 グループ名を変更



[グループ名] フィールドに新しい名前を入力して [OK] ボタンをクリックします。これで、プロジェクトウィンドウ内のグループの名前が変わります。

複数のグループを選択した場合、「グループ名を変更」ダイアログが次々に開きます。それぞれの [グループ名] フィールドに新しい名前を入力します。

ファイルのタッチ / アンタッチ

コンパイルが必要なファイルのフラグをオンにするには、[タッチ列](#)を使います（[図 3.2 \(p34\)](#)）。CodeWarrior IDE は常にファイルの変更を認識できるわけではなく、すべてのファイルを自動的に再コンパイルできないことがあります。このような場合はタッチ列の機能が役立ちます。

ファイルを確実にコンパイルするための方法が3つあります。

プロジェクトウィンドウのファイルビューでファイル名の横の[タッチ列](#)をクリックします。ファイル名の横のタッチ列にチェックマークが表示されます。次にプロジェクトをビルドするときに、チェックマーク付きのファイルが再コンパイルされます。

[ヘッダポップアップ](#)メニューから [タッチする] を選択します。

タッチ列の一番上のチェックアイコンをクリックします。これにより、最後に修正された日時によって、プロジェクト中のファイルの状態は再び同期を取ります。これはサードパーティのエディタなどによって、CodeWarrior IDE の外でファイルが修正された場合に便利です。

ヒント：最後にコンパイルされた時点からファイルが変更されていなければ、これらのポップアップの最初のコマンドは [タッチする] です。[タッチする] を選択すると、CodeWarrior IDE はファイルを変更したとマークし、次にプロジェクトをビルドする際にファイルをコンパイルします。最後のコンパイルの後ファイルが変更されている場合は、このコマンドは [アンタッチ] になります。

コンパイルしないようにするには、ファイル名の横の[タッチ列](#)を再度クリックしてチェックマークを消すか、[ヘッダポップアップ](#)から [アンタッチ] を選択します。

タッチ列の一番上のチェックアイコンで、プロジェクト内のファイルすべてをタッチすることができます。

修正日時の同期

プロジェクトファイルに保存された修正日時を更新するには、[タッチ列](#)の上のチェックマークアイコンをクリックする、または[プロジェクトメニュー](#)の [[ファイル更新日時を同期させる](#)] を選択します。

[[ファイル更新日時を同期させる](#)] を選択した後、CodeWarrior IDE はプロジェクト内の各ファイルの修正日時を検査します。ファイルが前回のコンパイル以降に修正されていれば、再コンパイルの必要ありと判断します。最後に修正された日付によって、プロジェクト中のファイルの状態は再び同期します。サードパーティのエディタなどはファイルの修正をCodeWarrior IDE に通知しないため、CodeWarrior IDE の外部でファイルが修正された場合に便利です。

複合プロジェクトを作成

CodeWarrior IDE は、洗練されたビルド規則を含むプロジェクトファイルを作成するための柔軟な機能を提供します。ここでは、複数の異なる種類のターゲットコード、または他のプロジェクトを含む複合プロジェクトファイルを作成する方法を説明します。この機能によって、ソフトウェアプロジェクト全体に対して強力なビルド階層を作成できます。

例えば、1 つのプロジェクトファイルに出荷版とデバッグ版の両方のコード用ビルドターゲットを含む複合プロジェクトを作成できます。ビルドターゲットを切り替えることによって、開発プロセスの間に異なるバージョンのソフトウェアが生成できます。これらのビルドターゲットのそれぞれが、独自の設定を持つことができます。例えばデバッグ用のビルドターゲットでは、最適化せずにデバッグ情報を使用可能にし、出荷用のビルドターゲットでは最適化したコードを生成することができます。

ここでは、以下の項目を説明します。

[ビルドターゲットとは](#)

[サブプロジェクトとは](#)

[デザインとは](#)

[複合プロジェクトの作成方法](#)

[新しいビルドターゲットを作成](#)

[ビルドターゲット名の変更](#)

[ビルドターゲット設定の変更](#)

[現在のビルドターゲットを設定](#)

[ターゲットの依存関係を作成](#)

[ターゲットにファイルを割り当てる](#)

[プロジェクト内にサブプロジェクトを作成](#)

ビルドターゲットとは

ビルドターゲットとは、出力ファイル（アプリケーションやライブラリ）を作成するための一連の規則および設定です。

CodeWarrior IDE は、1 つのプロジェクトファイルから異なる種類の出力ファイル、またはビルドターゲットを多数ビルドすることができます（[図 3.5 \(p39\)](#)）。例えば 1 つのプロジェクトに 2 つのビルドターゲットを設定し、出荷コード用とデバッグ用のビルドターゲットを管理できます。

プロジェクト内の複数ビルドターゲットにそれぞれビルド順を設定することができます。IDE はこのビルド順を守ります。これはビルドターゲットでリソースファイルを共有するのに便利です。依存するターゲットを強制的に最初にビルドすることによって、他のターゲットに依存するターゲットを作成できます。

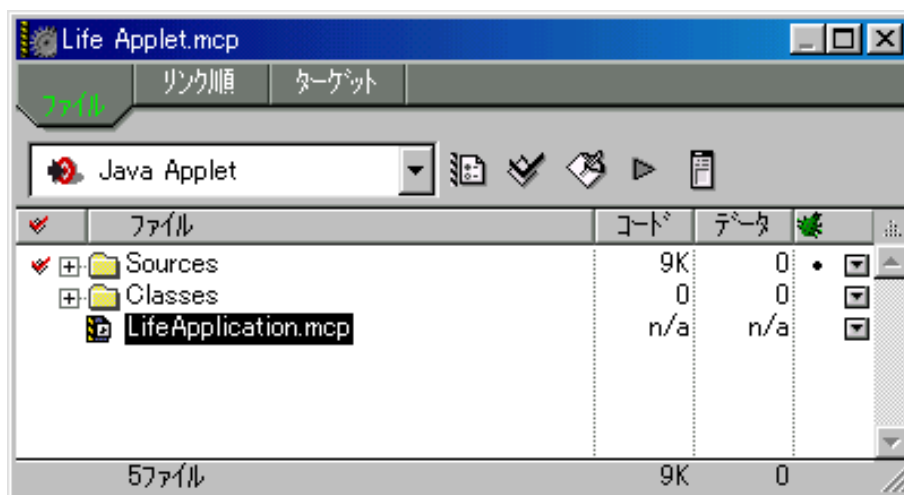
プロジェクトのターゲットはそれぞれ独自のビルド設定を持つことができます。

ビルドターゲットについての注意は「[ターゲットマニュアル](#)」(p20) および「[複合プロジェクトの作成方法](#)」(p66) を参照してください。

サブプロジェクトとは

サブプロジェクトは、プロジェクト中に入れ子（ネスト）された通常の独立型のプロジェクトファイルです。[図 3.17](#) の Life Application.mcp がサブプロジェクトです。メインのプロジェクトと別にしておきたいプロジェクトがある場合、サブプロジェクトは便利です。これにより別々のプロジェクトファイルのビルドプロセスを管理することができます。

図 3.17 サブプロジェクトのあるプロジェクト



プラグインアーキテクチャを利用するアプリケーションを開発する場合にサブプロジェクトは便利です。プログラムが多くのプラグインコードモジュールを使用し、各々が他のプラグインと共通のソースコードをいくつか共有していると考えてください。各プラグイン用に別々のターゲットを作成することによって、全プラグインをビルドする1つのプロジェクトファイルを作成できます。この場合、プラグイン用のプロジェクトファイルがサブプロジェクトです。メインアプリケーションのプロジェクトファイルにサブプロジェクトを追加することによって、メインアプリケーションをビルドする前に、すべてのプラグインをビルドできます。

プロジェクトファイルは、プロジェクト中の任意のビルドターゲットに割り当てられます。詳細は「[ターゲットにファイルを割り当てる](#)」(p69) を参照してください。

プロジェクトにファイルを追加する方法は「[ファイルを追加](#)」(p56) を参照してください。

メインプロジェクトにあるターゲットをビルドするときに、サブプロジェクト内でもビルドターゲットを複数選択できます。メインプロジェクトのターゲットがビルドされる際、CodeWarrior IDE はサブプロジェクトで選択されたターゲットをまずビルドします。

オプションで、メインプロジェクトの出力コードをサブプロジェクトのターゲットの出力とリンクできます。これはメインプロジェクトのターゲットビューのリンク列で設定します。詳細は「[ターゲットビュー](#)」(p39) を参照してください。

親のプロジェクトに追加されたサブプロジェクトのターゲットは、自動的にビルドされません。サブプロジェクト内で選択されたビルドターゲットのみがビルドされます。

サブプロジェクトは、ターゲット固有にできます。つまり、追加したサブプロジェクトがどのターゲットに属するのかが選択できます。選択したターゲットにサブプロジェクトファイルが追加されなければ、メインプロジェクト中の他のターゲットはサブプロジェクトをビルドしません。

デザインとは

デザインとは RAD 用のコンポーネント、レイアウト、オブジェクトをまとめたものです。

プロジェクト内のデザインは別個のビルドターゲットを持つことができます。デザインにソースファイルを追加するとき、ファイルは自動的にデザインのビルドターゲットに追加されます。

CodeWarrior の RAD 機能については「[RAD デザインとレイアウト](#)」(p365) を参照してください。

複合プロジェクトの作成方法

1 つのプロジェクトファイル内で複数のビルドターゲットと、サブプロジェクトのどちらを使うかの選択は、何を優先するかによります。1 つのプロジェクト中のソースコードすべてにアクセスしたければ、複数のターゲットを使用することをお勧めします。独立型のプロジェクトファイルを別にしておきたければ、サブプロジェクトの方がよいでしょう。

例えば、アプリケーションに伴う多くのプラグインライブラリをビルドする必要がある場合は、単一の「[メイク](#)」コマンドでサブプロジェクトをビルドできるプロジェクトを作成します。その後、このプロジェクトファイルをサブプロジェクトとして、メインアプリケーションのプロジェクトファイルに追加します。メインアプリケーションがビルドされる時、サブプロジェクトのプラグインがまず最初にビルドされます。

プロジェクトに設定できるビルドターゲットの数は 255 個です。この制限に達する前に、メモリやプロジェクトのロード時間を考慮すべきです。たくさんのビルドターゲットを持つプロジェクトは、多くのディスクスペースとメモリを使い、ロード時間も長くなります。

ビルドターゲットの数が 10 や 20 を超えたら、いくつかをサブプロジェクトへ移動した方がよいでしょう。あまり頻繁にビルドされないものや、別のソースファイルを使用するものが、サブプロジェクトに移動する候補になります。

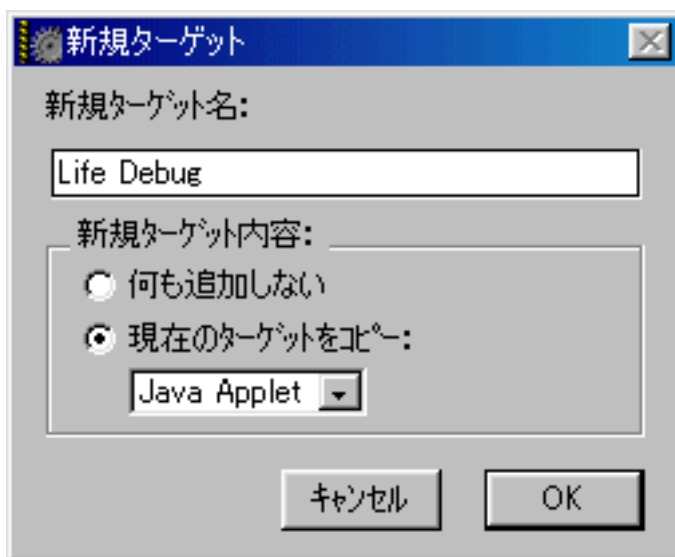
新しいビルドターゲットを作成

プロジェクトに新しいビルドターゲットを作成するには、[プロジェクトメニュー](#)の「[新規ターゲットを作成](#)」を選択します。プロジェクトウィンドウでターゲットビューを選択していると、このコマンドが表示されます ([図 3.5 \(p39 \)](#))。

「新規ターゲットを作成」コマンドを選択すると、[図 3.18](#) のダイアログが現れます。このダイアログの「ターゲット名」フィールドで新しいターゲットの名前を指定します。

新しいビルドターゲットを空にするか、または以前のターゲットのコピーにするかを選択します。[何も追加しない]を選択すると、新規プロジェクトウィンドウを開いたときのよう、ターゲットの全オプションを設定する必要があります。[現在のターゲットをコピー]を選択すると、新しいビルドターゲットに対する設定は、ポップアップメニューから選択するターゲットからコピーされます。元のターゲットに含まれているファイルのコピーもすべて取得します。

図 3.18 「新規ターゲット」ダイアログ



ターゲットを作成した後、ビルドの従属関係を作るためにこのビルドターゲットを別のターゲットに関連付けます。[「ターゲットの依存関係を作成」\(p68\)](#)を参照してください。

ビルドターゲットの設定方法は[「設定パネルを選択」\(p228\)](#)を参照してください。

他のビルドターゲットの出力ファイルに依存するターゲットを新しく作成する場合は、その出力ファイルを作成するターゲットのリンク列をクリックする必要があります。

ビルドターゲット名の変更

プロジェクトウィンドウのターゲットビューでビルドターゲット名を変更する場合、名前を変更したいファイルをダブルクリックします([図 9.4 \(p230\)](#))。表示される設定ダイアログの左側のリストの[ターゲット]の下にある[[ターゲット設定](#)]を選択し、ターゲット設定パネルの[ターゲット名]フィールドでターゲットの名前を変更します。

編集メニューの[*Target* の設定](*Target* には現在のターゲットの名前が入ります)を選択すると表示されるダイアログの、左側のリストの[ターゲット]の下にある[[Target の設定](#)]を選択して設定パネルを開くこともできます。

[ターゲット設定](#)の詳細は、[「ターゲット設定」\(p229\)](#)を参照してください。

ビルドターゲット設定の変更

プロジェクト中の各ビルドターゲットには独自の設定がなされています。これらはターゲット設定ダイアログで変更できます。

ターゲット設定ダイアログを開くには、ターゲットビューで、ターゲット名をダブルクリックします。または編集メニューの [*Target* の設定] (*Target* には現在のターゲットの名前が入ります) を選択します。

ターゲット設定ダイアログの左側には、現在のビルドターゲットで利用できる設定パネルのリストがあります。クリックでパネルを選択してください。

設定パネルの変更方法は、[「ターゲットのオプション設定」\(p225\)](#) を参照してください。このマニュアルで扱う設定パネルについて説明しています。特定のオペレーティングシステムに固有の設定パネルについては、各『*Targeting*』マニュアルを参照してください。

現在のビルドターゲットを設定

[プロジェクトメニュー](#) の [[デフォルトターゲットを設定](#)] を使って、現在のアクティブなビルドターゲットを変更できます。プロジェクトの複数のターゲットの間を切り替えて、それぞれをビルドするときには便利です。

プロジェクトウィンドウのターゲットビューでも現在のビルドターゲットを変更できます ([図 3.5 \(p39\)](#))。ビルドされる現在のターゲットは、矢の付いた円のアイコン (アーチェリーの「的」) で印付けられます。ビルドターゲットを変更するには、現在のターゲットとして選択したいターゲットの名前をクリックします。

ターゲットの依存関係を作成

あるビルドターゲットと他のビルドターゲットの依存関係を設定できます。これはあるターゲットを先にビルドしたい場合に便利です。例えば、第2ビルドターゲットの情報を参照する第1ターゲットをビルドする場合、IDE は依存される第2ターゲットを、それに依存する第1ターゲットよりも先にビルドしなくてはなりません。

第2ビルドターゲットに依存するビルドターゲットを作成する手順を以下に紹介します。

1. プロジェクトウィンドウのターゲットビューを開く

プロジェクトウィンドウの [ターゲット] タブをクリックするとターゲットビューが表示されます。

2. 第2ビルドターゲットを第1ビルドターゲットの下にドラッグする

第1ビルドターゲットグループの中に、第2ビルドターゲットの名前が斜体でインデントされて表示されます。このエントリによって、第1ビルドターゲットは第2ビルドターゲットに依存します。

これで IDE は先に第2ターゲットをビルドし、次に第1ターゲットをビルドします。

第1ビルドターゲットと、第2ビルドターゲットのオブジェクトコードをリンクする手順を紹介します。

1. 第2ビルドターゲット名（斜体）のリンク列をクリックする

第2ターゲットの斜体の名前は、第1ビルドターゲットグループの中にあります。

2. 第2ビルドターゲットのオブジェクトコードが、第1ビルドターゲットのリンク順ビューに表示される

第1ビルドターゲットがアクティブにして、プロジェクトウィンドウの[リンク順]タブをクリックするとリンク順ビューが表示されます。第2ターゲットのオブジェクトコードが第1ターゲットのグループに表示されます。

注意： Windows 上では、第2ビルドターゲットグループのオブジェクトコードはリンク順の最初に表示されます。

新しいターゲットの作成方法は「[新しいビルドターゲットを作成](#)」(p66)を参照してください。

現在のターゲットを設定する方法は「[現在のビルドターゲットを設定](#)」(p68)を参照してください。

ターゲットおよびサブプロジェクトを使う複合プロジェクトの設定方法は、「[複合プロジェクトの作成方法](#)」(p66)を参照してください。

ターゲットにファイルを割り当てる

プロジェクトのターゲットにファイルを割り当てる方法は2つあります。ファイルビューのターゲット列を使う、またはプロジェクトインスペクタウィンドウを使う方法です。

ターゲット列を使う

プロジェクトウィンドウのファイルビューのターゲット列は、ファイルが現在のビルドターゲットに属するか否かを示します。プロジェクトが複数のターゲットを持つ場合にこの列が表示されます。プロジェクトのファイルが現在のターゲットに含まれる場合、そのファイルのターゲット列にマーカが付きます。

ファイルをアクティブなターゲットに割り当てるには、そのターゲット列をクリックしてマーカを付けます。ファイルをアクティブなターゲットから除去するには、そのターゲット列をクリックしてマーカを消します。

ファイルビューにあるすべてのファイルを現在のターゲットに割り当てる、または除去するには、ターゲット列をAlt+クリックします。ターゲット列のすべての項目にマーカが付く、または消えます。

プロジェクトインスペクタを使う

[プロジェクトインスペクタ](#)ウィンドウでファイルをビルドターゲットに割り当てることができます。最初に、プロジェクトウィンドウでファイルを選択します。詳細は「[ファイル](#)

[「とグループを選択」\(p55\)](#)を参照してください。次に[ウィンドウメニュー](#)の[[プロジェクトインスペクタ](#)]を選択します。[図 3.20](#)のようなウィンドウが現れます。

ターゲットビュー ([図 3.21 \(p72\)](#)) に切り替えるには、ターゲットタブをクリックします。このウィンドウは、選択したファイルがどのビルドターゲットに属しているのかを示します。ファイルが別のターゲットに属するように変更したければ、ウィンドウの左端のチェックボックスをクリックします。チェックボックスがオンの場合、ファイルが指定したターゲットに割り当てられます。オフの場合はターゲットから除外されます。

変更を取り消す場合は、[復帰] ボタンをクリックします。プロジェクトインスペクタウィンドウを開いたままで変更を適用する場合は、[保存] ボタンをクリックします。

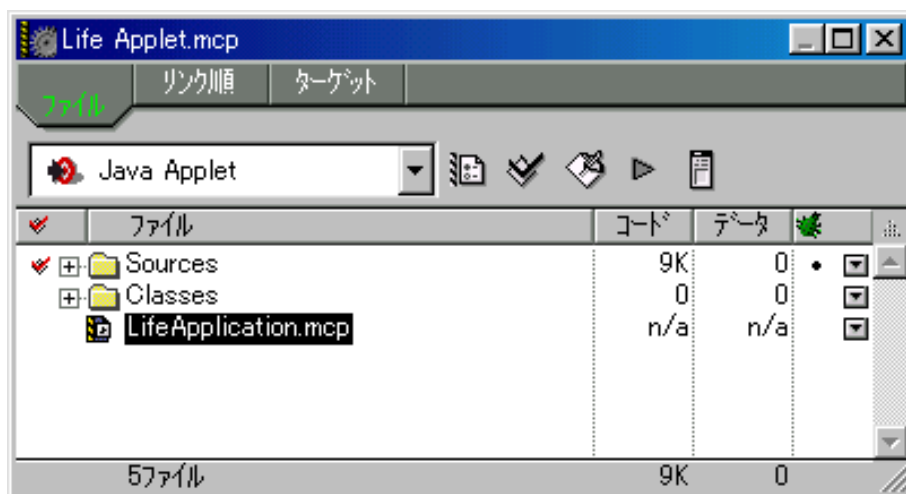
プロジェクト内にサブプロジェクトを作成

サブプロジェクトを作成するには、開いているプロジェクトウィンドウにプロジェクトファイルをドラッグ & ドロップするだけです。プロジェクトファイルに複数のターゲットが含まれている場合、サブプロジェクトを追加するターゲットを選択するよう要求されます ([図 3.15 \(p60\)](#))。これを行うと、プロジェクトウィンドウは[図 3.19](#) と同じようになり、プロジェクトウィンドウのファイルリストにはプロジェクトファイルが追加されます。[「ファイルを追加」\(p56\)](#) で説明した方法を使用しても、ファイルを追加できます。

これによりサブプロジェクトファイルは、メインプロジェクトファイルの一部になります。メインプロジェクト上で [「メイク」](#) を実行すると、最初にサブプロジェクトがビルドされます。

プロジェクトファイルにサブプロジェクトを追加したとき、サブプロジェクトを割り当てるビルドターゲットを指定できます。[「ターゲットにファイルを割り当てる」\(p69\)](#) を参照してください。

図 3.19 プロジェクト内のサブプロジェクト



プロジェクト情報を検証

プロジェクトインスペクタウィンドウ (図 3.20) で、プロジェクト、ソースファイルの情報を検証、設定することができます。このウィンドウを開くには、[ウィンドウメニュー](#)の [[プロジェクトインスペクタ](#)] を選択します。

このウィンドウには2つのタブ ([アトリビュート] と [ターゲット]) があります。

コンパイルするファイルをビルドターゲットへ割り当てる方法は、[「ターゲットにファイルを割り当てる」](#) (p69) を参照してください。

プロジェクト内のファイルの詳細をアトリビュートビューで見ることができます。アトリビュートビューは、ファイルの名前、ハードディスク上の位置へのパス、属するプロジェクトの名前、コンパイルされていればコードとデータのサイズ、およびコンパイル時にデバッグ情報を生成するかなどの属性を表示します。

図 3.20 アトリビュートを表示する「プロジェクトインスペクタ」ウィンドウ



変更を取り消すには、[復帰] ボタンをクリックします。ウィンドウを開いたままで変更を適用するには、[保存] ボタンをクリックします。

デバッグ情報の設定については「[プロジェクトでデバッグを制御](#)」(p73)を参照してください。

図 3.21 ビルドターゲットを表示するプロジェクトインスペクタウィンドウ



プロジェクトを移動

CodeWarrior IDE はプロジェクトに関する情報をプロジェクトファイルに保存します。それ以外の、重要ではない情報（ウィンドウの位置、オブジェクトコード、デバッグ情報、ブラウザデータ、その他の設定など）を含むファイルは、プロジェクトと似た名前のフォルダに保存されます。これらは CodeWarrior IDE がプロジェクトを再作成するときに使用しないファイルです。

警告！ Windows 版の旧バージョンの CodeWarrior IDE は、Resource.frk フォルダにプロジェクトファイルの状態の情報を保存していました。Resource.frk は現在使用されておらず、CodeWarrior IDE によって生成されたり利用されたりはしていません。

ハードディスク上でプロジェクトを移動するには、プロジェクトファイル（拡張子 .mcp）をハードディスク上の新しい場所にコピーするだけです。プロジェクトファイル以外のファイルの情報も一緒に移動したければ、これらのファイルを含むフォルダをコピーします。しかし、これらのファイルは必要ありません。CodeWarrior IDE は、[\[最新状態に更新する \]](#) または [\[メイク \]](#) コマンドが実行されると、この状態を再設定できます。一般に、バージョン管理システムにチェックインするのはメインプロジェクトファイルだけで、他のファイルはしません。

絶対アクセスパスを設定している場合、プロジェクトファイルを移動する際にこれを修正する必要があります。詳しくは [「アクセスパス」\(p231\)](#) を参照してください。

[「他のプラットフォームで作成したプロジェクトファイルを開く」\(p50\)](#) も参照してください。

プロジェクトのインポートとエクスポート

CodeWarrior は、プロジェクトファイルを XML（eXtensible Markup Language）フォーマットへインポート、およびエクスポートすることができます。

開いているプロジェクトファイルをエクスポートするには、ファイルメニューの [\[プロジェクトの書き出し \]](#) を選択します。エクスポートファイルの名前を指定するダイアログが現れます。

XML フォーマットのファイルをインポートするには、ファイルメニューの [\[プロジェクトの読み込み \]](#) を選択します。インポートするファイルを選択するダイアログが現れます。IDE は選択されたファイルを CodeWarrior プロジェクトファイルへインポートし、保存するか否かを尋ねます。

ヒント： プロジェクトファイルを保存するフォルダを新たに作ってください。これにより IDE はプロジェクトに関連するファイルをプロジェクトファイルと同じフォルダに保存することができます。

プロジェクトでデバッグを制御

最初のビルドでは、おそらくプログラムは正しく動作しないでしょう。これをデバッグするには、プロジェクトとそのファイルのデバッグ情報を使用可能にします。ここではデバッグ情報の設定について説明します。

ここでは、以下の項目を説明します。

[プロジェクトのデバッグをアクティブにする](#)

[ファイルのデバッグをアクティブにする](#)

プロジェクトのデバッグをアクティブにする

プロジェクトのデバッグを有効にするには、[プロジェクトメニュー](#)の[[デバッグを有効にする](#)]を選択します。IDE は自動的にプロジェクト設定を行い、デバッグ情報を生成します。

手動でプロジェクトを設定してデバッグ情報を生成する方法は「[設定パネルを選択](#)」(p228)を参照してください。

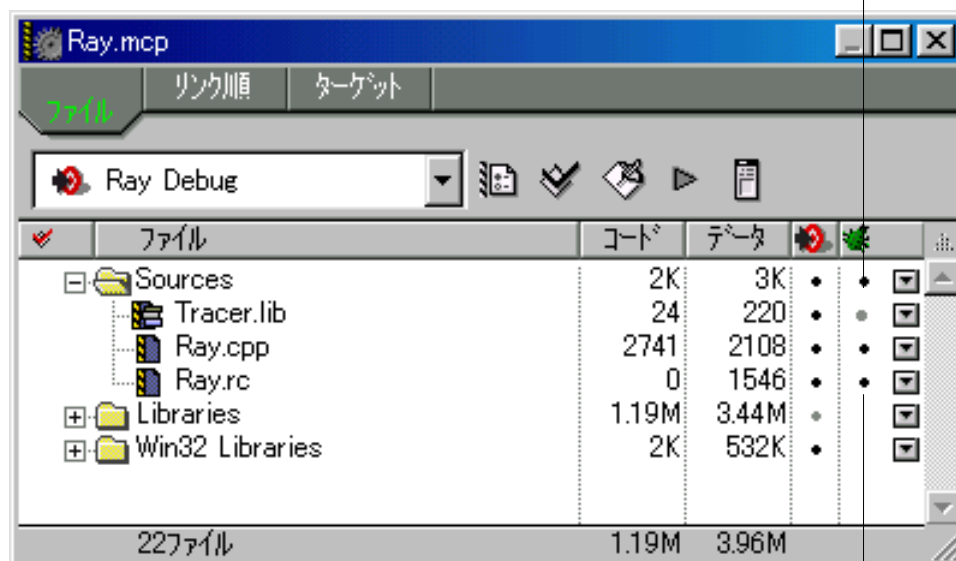
ファイルのデバッグをアクティブにする

1 つのソースファイルに対してデバッグ情報を生成するには、[デバッグ列](#)をクリックします。その列にデバッグ情報マーカが表示されます ([図 3.22 \(p74\)](#))。「*Target* の設定」ダイアログ (*Target* には現在のターゲットの名前が入ります) のリンク設定パネルで適切なオプションがオンになっている場合、プロジェクトにファイルを追加すると、CodeWarrior IDE は自動的にこのマーカを設定します。

「[プロジェクトインスペクタ](#)」ウィンドウでもデバッグ情報の生成を設定できます。「[プロジェクトウィンドウの解説](#)」(p32)を参照してください。プロジェクトにファイルが追加されると、そのデバッグ状態は[プロジェクトメニュー](#)の[[デバッグを有効にする](#)]コマンドの状態によって設定されます。[デバッグを有効にする]が表示されているとき、デバッグは無効なのでデバッグ情報も生成されません。[デバッグを無効にする]が表示されているとき、デバッグは有効で自動的にプロジェクトに追加されたファイルのデバッグ情報を生成します。

図 3.22 デバッグ情報マーカ

灰色のマーカはグループ内の一部のファイルのデバッグ情報を生成することを示す



黒いマーカはこのファイルのデバッグ情報を生成することを示す

デバッグ情報マーカ ([図 3.22](#)) は、プロジェクトがビルドされるときにこのファイルに対してデバッグ情報が生成されることを示します。マーカはクリックすると消え、デバッグ情報は生成されません。

デバッグ情報マーカが変更されるたびに、IDE は次回のビルド時のコンパイルの設定を行います。

現在のビルドターゲットに含まれる全ファイルのデバッグ情報を生成するには、任意のデバッグ列を Shift + クリックします。再度 Shift + クリックすることで解除されます。

注意： ソースファイルにマーカを設定することと、リンク中にデバッグファイルを作成することとは違います。リンクの設定パネルには、デバッグファイルの作成を指定するオプションがあります。

グループに対するデバッグ情報マーカ

グループのデバッグ列にもデバッグ情報マーカは表示され ([図 3.22](#))、クリックによってオンとオフを切り替えられます。グループに対する [デバッグ列](#) は、以下の 3 つのいずれかを表示します。

黒マーカ：グループ内の全ファイルがデバッグ情報を生成する

灰色マーカ：グループ内の一部のファイルだけがデバッグ情報を生成する

マーカなし：グループ内のどのファイルもデバッグ情報を生成しない

プリプロセッサシンボルをプロジェクトに追加

独自のシンボル定義をプロジェクトに追加しておく、プロジェクトをビルドするとき、各ソースファイルの先頭に自動的にシンボルを含めることができます。

以下は C/C++ 言語の例です。

```
#define GLOBAL_DEBUG
```

開発バージョンをビルドするときにはこのシンボルを定義して、最終版のビルド前に定義を削除することもできます。

これを行うには、プリコンパイルヘッダを作成し、そのヘッダにこのシンボル定義を挿入します。詳細は「[プリコンパイルまたはプリプロセスヘッダを使う](#)」(p262) を参照してください。

『C Compilers Reference』または『Pascal Compilers Reference』も参照してください。

第 4 章 ファイル

CodeWarrior IDE でファイルを扱う作業の背景となる概念を説明します。

この章では、CodeWarrior 環境でファイルを開く、作成する、保存する、閉じる、および印刷するための操作について説明します。

ファイルの編集については「[ソースコードの編集](#)」(p95) を参照してください。

バージョン管理システム (VCS : Version Control System) を使用するファイル作業については、「[バージョン管理システム](#)」(p443) を参照してください。

以下の項目について説明します。

[新規ファイルを作成](#)

[既存ファイルを開く](#)

[ファイルを保存](#)

[ファイルを閉じる](#)

[ファイルを印刷](#)

[直前に保存したファイルに戻す](#)

[ファイル / フォルダの比較とマージ](#)

新規ファイルを作成

ソースコードを入力するための新しい名称未設定のウィンドウを作成するには、[ファイルメニュー](#)の「[新規テキストファイル](#)」を選択します。

新しいウィンドウが開き、最初の行にテキストの挿入位置が置かれます。入力を始めると、テキストはこの挿入位置から表示されます。

作成したばかりのウィンドウでのテキスト編集についての詳細は、「[ソースコードの編集](#)」(p95) を参照してください。

既存ファイルを開く

CodeWarrior IDE でファイルを開くには、複数の方法があります。以下の方法を説明します。

[ファイルメニューからファイルを開く](#)

[プロジェクトウィンドウからファイルを開く](#)

[エディタウィンドウからファイルを開く](#)

[関連するファイルを開く](#)

注意： ライブラリはバイナリフォーマットであるため、CodeWarrior エディタで開くことはできません。

ファイルメニューからファイルを開く

CodeWarrior エディタで 2 種類のファイルを開くことができます。

[プロジェクトファイル](#)：CodeWarrior プロジェクトをビルドするための情報を含むファイル

[テキストファイル](#)：ソースコード、ヘッダ、またはその他のテキストファイル

プロジェクトファイル

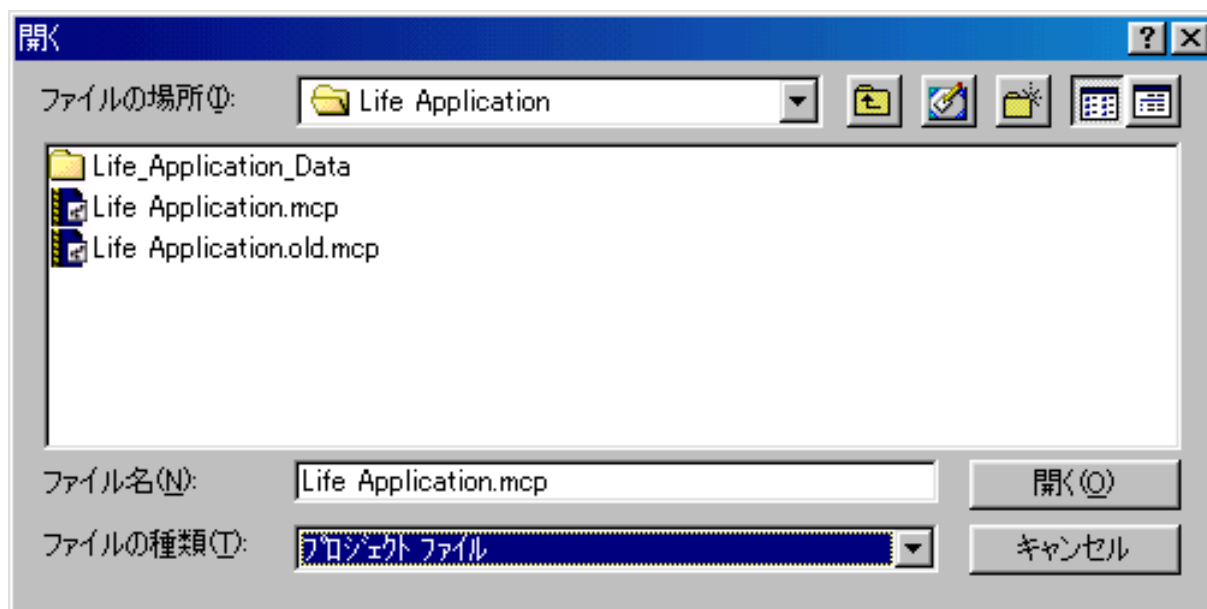
プロジェクトファイルを開くには、[ファイルメニュー](#)の[[開く](#)]を選択します。CodeWarrior プロジェクトファイルの詳細は、「[既存のプロジェクトを開く](#)」(p49) を参照してください。

テキストファイル

テキストファイルまたはソースファイルを開くには、[ファイルメニュー](#)の[[開く](#)]を選択します。「開く」ダイアログが表示されます（[図 4.1](#)）。

[ファイルの種類] ポップアップメニューから [すべてのファイル] を選択します。ファイルのリストには現在のフォルダにある全ファイルが表示されます。

図 4.1 「開く」ダイアログ



開きたいファイルをクリックして選択し [開く] ボタンをクリックします。ファイルがエディタウィンドウに開かれます。

ソースコードの編集の詳細は、[「ソースコードの編集」\(p95\)](#) を参照してください。

プロジェクトウィンドウからファイルを開く

プロジェクトウィンドウからファイルを開くには、ファイルの種類により 3 つの方法があります。以下がその方法です。

[ファイル列](#)：プロジェクト内にあるファイルを開く

[グループポップアップメニュー](#)：縮小されたグループ中からテキストソースファイルを開く

[ヘッダポップアップメニュー](#)：プロジェクトのソースファイルにインクルードされているヘッダファイルを開く

ファイル列

プロジェクトウィンドウの[ファイル列](#)、またはファイルビューからリンク順ビューのファイル列に表示されているファイルを見たい場合は、そのファイル名をダブルクリックして開きます。テキストファイルはエディタウィンドウに表示されます。

ファイルを選択してから Enter キーを押してもファイルは開きます。プロジェクトウィンドウで複数のファイルを選択して Enter キーを押すと、選択したすべてのファイルを開くことができます。プロジェクト中で複数のファイルを選択する方法は、[「ファイルとグループを選択」\(p55\)](#) を参照してください。

ファイル列の詳細は [「ファイル列」\(p34\)](#) を参照してください。

ヒント： ファイル列から一度に複数のファイルを開くには、ファイルを Ctrl + クリックします。その後、選択したファイルのいずれかの上でダブルクリックします。

グループポップアップメニュー

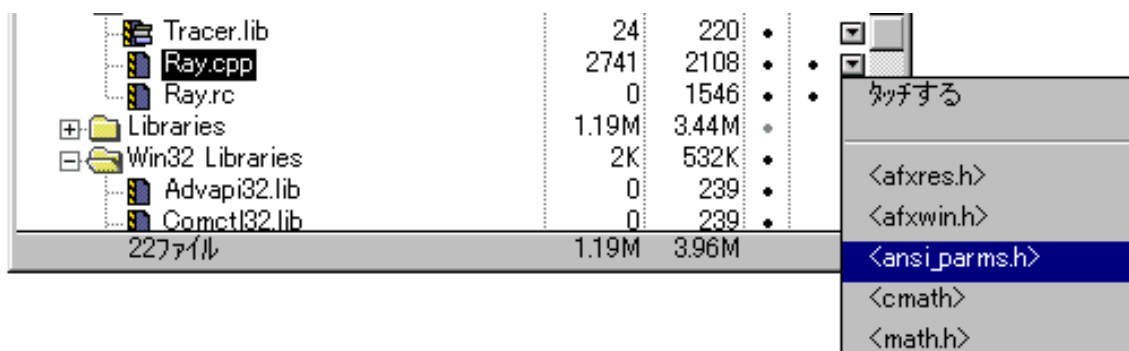
ソースファイルを開くには [[ヘッダポップアップ](#)] メニューをクリックします。ファイルのリストが表示されます。このリストから開きたいファイルを選択します ([図 4.2](#))。

ファイルを含むグループの[ヘッダポップアップ](#)メニューからソースファイルを選択して開くことができます。これは、グループが縮小されていてプロジェクトウィンドウに表示されていないときでも有効です。

ヘッダポップアップメニュー

ヘッダファイルを開くには [ヘッダポップアップ](#) をクリックします。ファイルのリストが表示されます。このリストから開きたいファイルを選択します ([図 4.2](#))。

図 4.2 プロジェクトウィンドウのヘッダポップアップメニュー



<...> で括られたヘッダファイルは、Metrowerks CodeWarrior フォルダ内に置かれたシステムヘッダファイルです。<...> のないファイルは、ユーザーが作成して、プロジェクトと同じフォルダか、指定した他のアクセスパスに格納したヘッダファイルです。アクセスパスの詳細と構成方法は、「[アクセスパス](#)」(p231) を参照してください。

ヒント： ソースファイルとヘッダファイルを切り替えるには、両者に拡張子以外は同じ名前を付けます。例えば、foo.cpp と foo.h という名前を付けます。そこで Ctrl + ` キーを押すと、2つのファイルの間を即座に切り替えられます。

プロジェクトの一部であるライブラリのヘッダファイルポップアップをクリックしたときは、ライブラリに対するタッチとアンタッチのオプションのみ有効です。ライブラリはヘッダファイルを含んでいないため、ライブラリのポップアップメニューからこれらのファイルを開くことはできません。

ファイルのタッチとアンタッチについては「[ファイルのタッチ / アンタッチ](#)」(p63) を参照してください。

エディタウィンドウからファイルを開く

編集中のソースファイルからヘッダファイルを開くには、エディタウィンドウの右上の[ヘッダポップアップメニュー](#)をクリックします([図 5.2\(p97\)](#))。このポップアップメニューは、ソースファイルで使われるすべてのヘッダファイルをリストします。このメニューからファイルを選択すると、新しいエディタウィンドウでそのファイルが開かれます。

注意： このメニューにファイルが表示されない場合、テキストファイルがソースコードを含んでいない、またはソースファイルがまだコンパイルされていないことを意味します。

ソースファイルを編集しているとき、そのファイルのどこかに記述されているヘッダファイルを[ファイルメニュー](#)の[[ファイル名を検索して開く](#)]コマンドで開くこともできます。

最初に、開きたいヘッダファイルの名前を含むテキストをエディタウィンドウで選択します。例えば、C ソースファイル中で `stdio.h` を見たいとします。`stdio` というテキストをダブルクリックして選択し、その後、[ファイルメニュー](#)の[[ファイル名を検索して開く](#)]を選択します。

CodeWarrior IDE はファイルを検索して、エディタウィンドウにそのファイルを開きます。ファイルが見つからなかった場合は、ピープ音が鳴ります。

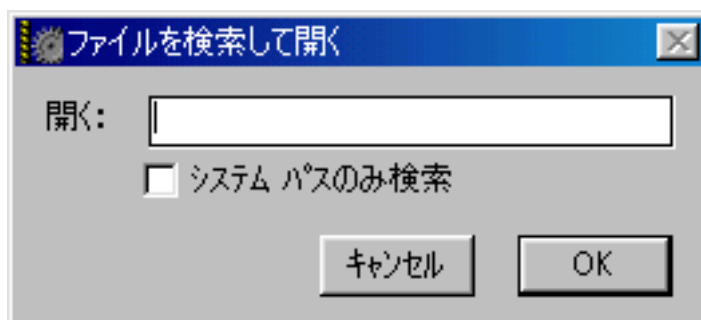
他にも便利な方法があります。C++ 言語の `.cpp` ファイルを編集していると仮定します。Ctrl + ` キーを押すと、新しいエディタウィンドウが開き、`.cpp` ファイルに対応する `.h` ファイルを表示します。同じキーボードショートカットで `.cpp` ファイルが再度表示されます。

このキーボードショートカットを使うためには、ソースファイルとヘッダファイルの名前が、拡張子を除いて同じでなくてはなりません。`myFile.cpp` を編集中にそのヘッダファイルを見たい場合、Ctrl + ` キーを押すと `myFile.h` がエディタウィンドウに表示されます。IDE はアクセスパス設定パネルのオプションに従って関連するファイルを検索します。[「アクセスパス」\(p231\)](#) を参照してください。

ソースファイルの編集中に、テキストを選択せずにファイルを開くには、[ファイルメニュー](#)の[[ファイルを検索して開く](#)]を選択します。このコマンドはプロジェクトの[アクセスパス](#)の設定を使って開きたいファイルを検索します。

[[ファイルを検索して開く](#)]コマンドを選択した後、CodeWarrior IDE はダイアログ ([図 4.3](#)) を表示します。この [開く] フィールドに検索するファイル名を入力してください。

図 4.3 「ファイルを検索して開く」ダイアログ



[システムパス欄](#)と[ユーザパス欄](#)の両方のディレクトリ ([アクセスパス](#)設定パネルで指定したすべてのパス)を検索するには、[システムパスだけで検索] をオフにします。

CodeWarrior ディレクトリ構造 ([アクセスパス](#)設定パネルの[システムパス欄](#)で指定したパス) だけを検索するには、[システムパスだけで検索] をオンにします。

アクセスパスの詳細と設定方法は、[「アクセスパス」\(p231\)](#) を参照してください。

関連するファイルを開く

ソースファイルを開いているときに対応するヘッダファイルを開く、またはヘッダファイルを開いているときに対応するソースファイルを開くには、Ctrl + ` キーを押します。このキーボードショートカットで 2 つのファイルを簡単に切り替えられます。

ファイルを保存

ここでは、CodeWarrior IDE のファイル保存方法を説明します。次の内容を説明します。

[1 つのファイルを保存](#)

[ファイルを自動的に保存](#)

[ファイルを新規保存](#)

[ファイルのコピー保存](#)

[MS-DOS、Mac OS、UNIX のテキストファイルとして保存](#)

1 つのファイルを保存

現在エディタで開いているファイルの変更を保存するには、[ファイルメニュー](#)の[[保存](#)]を選択します。CodeWarrior IDE はファイルをハードディスクに保存します。

新しいウィンドウでデータを何も入力していない状態、アクティブなウィンドウの内容が保存された直後である場合、またはアクティブなウィンドウがプロジェクトウィンドウであるとき、[[保存](#)] コマンドは薄く表示されます。

注意： ファイルが新しくて名称未設定の場合、「名前を付けて保存」ダイアログが表示されます（「[ファイルを新規保存](#)」(p83)）。このダイアログで新しいファイルの名前と保存位置を指定します。

プロジェクトは、閉じたとき、CodeWarriorを終了したとき、または[[コピーを保存](#)] コマンドを選択したときに保存されます。プロジェクトを明示的に保存する必要はありません。

すべてのファイルを保存

現在開いているすべてのファイルの変更を保存するには、Shift + Ctrl + S キーを押します。

ファイルを自動的に保存

[プロジェクトメニュー](#)の[[プリプロセス](#)]、[[プリコンパイル](#)]、[[コンパイル](#)]、[[逆アセンブル](#)]、[[最新状態に更新する](#)]、[[メイク](#)]、[[実行](#)] を選択したときに、修正したファイルはすべて自動的に保存されます。

[[ビルド前に開いているファイルを保存](#)] オプションは、プログラムが実行中にクラッシュしても、それまでの作業を無駄にしません。しかし、テストなどで変更を保存したくない場合はこのオプションをオフにします。

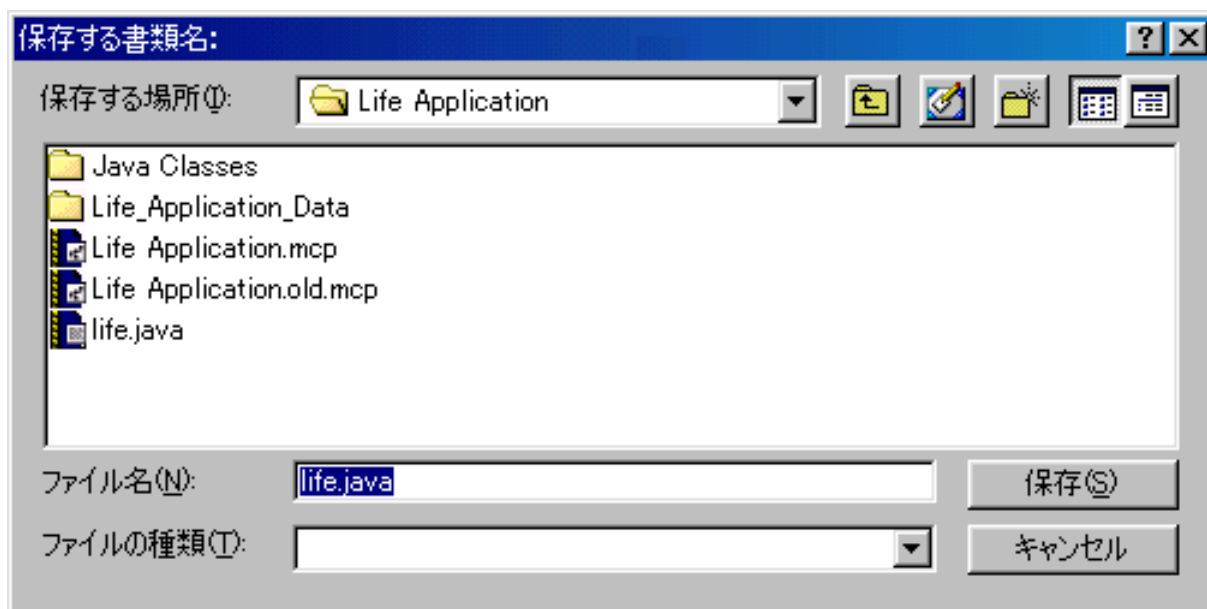
この機能の使い方は「[エディタ設定](#)」(p194) の [[ビルド前に開いているファイルを保存](#)] オプションを参照してください。

ファイルを新規保存

新しい名称未設定のファイルを保存する、またはファイルを新しい名前で保存するには、[ファイルメニュー](#) の [[名前を付けて保存](#)] を選択します。ファイルが現在のプロジェクトに含まれていれば、CodeWarrior IDE はプロジェクトを更新して新しい名前を使用します。

[ファイルメニュー](#) の [[名前を付けて保存](#)] を選択すると、「名前を付けて保存」ダイアログ ([図 4.4](#)) が表示されます。

図 4.4 「保存する書類名」ダイアログ



ファイルを保存する位置と名前を選択し、[保存] ボタンをクリックします。

CodeWarrior IDE はファイルを保存して、エディタウィンドウの名前を変更します。

ファイルが現在のプロジェクトに含まれていれば、CodeWarriorIDE はプロジェクト内でそのファイル名を新しい名前に変更します。プロジェクトを変更せずにファイルを保存する方法は「[ファイルのコピー保存](#)」を参照してください。

ファイルのコピー保存

元のファイルに変更を加える前に、テキストファイルのバックアップコピーを保存するには、[ファイルメニュー](#) の [[コピーを保存](#)] を使います。CodeWarrior IDE は、指定された新

しい名前でファイルのコピーを作成します。元のファイルも現在開かれているプロジェクトも変更されません。

[ファイルメニュー](#)の[[コピーを保存](#)]を選択すると、CodeWarrior IDE は「コピーを保存」ダイアログ ([図 4.4](#)) を表示します。ファイルの保存先の位置を指定し、ファイルにユニークな名前を付けてください。

[保存] ボタンをクリックすると、CodeWarrior IDE は新しい名前を付けたファイルを保存します。この操作はエディタウィンドウ内のファイルも、現在のプロジェクト内のファイルも変更しません。

プロジェクトウィンドウがアクティブな場合、[[コピーを保存](#)] コマンドでプロジェクトを新しい名前で保存できます。

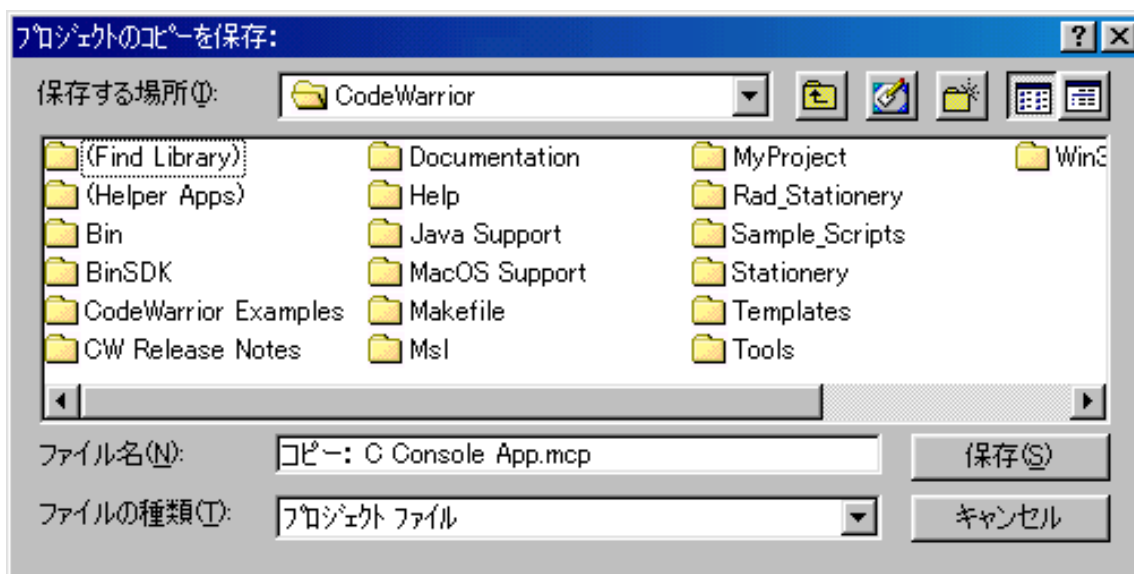
「プロジェクトのコピーを保存」ダイアログ ([図 4.5](#)) の [ファイル名] フィールドに、保存するファイルの名前を入力し、[ファイルの種類] ポップアップメニューで保存するファイルの種類を選択して、[保存] ボタンをクリックします。

MS-DOS、Mac OS、UNIX のテキストファイルとして保存

Windows、Mac OS、UNIX のエディタで作成されたテキストファイルを開くと、CodeWarrior は自動的にプラットフォーム互換のテキストファイルに変換します。このファイルは元のフォーマットで保存されます。

テキストファイルを異なるフォーマットで保存する方法は、[「オプションポップアップメニュー」\(p99\)](#) を参照してください。

図 4.5 プロジェクトのコピーを保存



ファイルを閉じる

CodeWarrior IDE で開かれているエディタまたはプロジェクトウィンドウはすべて、ハードディスク上のファイルと関連しています。ウィンドウを閉じると、ファイルも閉じられます。ウィンドウをすべて、または1つだけ閉じることもできます。

以下の項目について説明します。

[1つのファイルを閉じる](#)

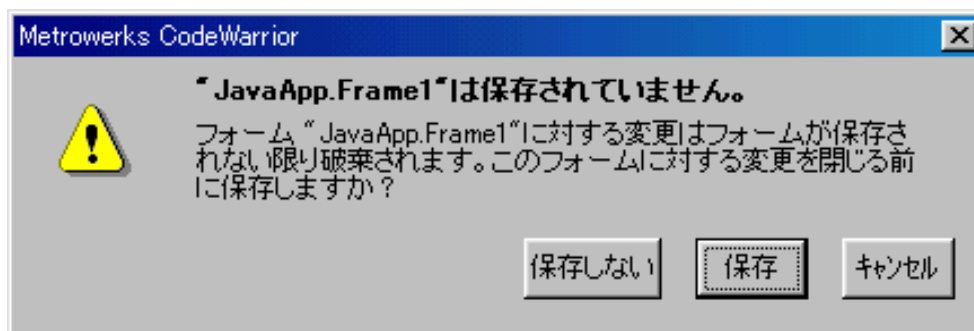
[すべてのファイルを閉じる](#)

1つのファイルを閉じる

ウィンドウを1つ閉じるには、[ファイルメニュー](#)の[[閉じる](#)]を選択します。

[ファイルメニュー](#)を使ってテキストファイルを閉じるとき、まだ変更を保存していなかった場合、ウィンドウを閉じる前に変更を保存するかどうかを確認する警告ダイアログが現れます([図 4.6](#))。保存しないでファイルを閉じると、すべての変更が失われます。

図 4.6 変更が保存されていないときの警告ダイアログ



アクティブなウィンドウのクローズボックスをクリックしてもウィンドウが閉じます。これは[ファイルメニュー](#)の[[閉じる](#)]を選択することと同じです。

プロジェクトウィンドウがアクティブである場合、ウィンドウを閉じると自動的にプロジェクトが保存されるので [図 4.6](#) のダイアログは表示されません。プロジェクトの保存については「[プロジェクトを保存](#)」(p52)を参照してください。

[[閉じる](#)] コマンドは、サイズ、位置、アクティブウィンドウで選択されたテキストなどのウィンドウの属性も保存します。これらのオプションについては「[エディタ設定](#)」(p194)を参照してください。適切なオプションをオンにすると、次回からソースファイルは画面上的同じ位置に同じテキストが選択された状態で表示されます。

すべてのファイルを閉じる

すべてのウィンドウを一度に閉じるには、[ファイルメニュー](#)の[[すべて閉じる](#)]コマンドを使います。ファイルに変更を加えた場合、ウィンドウを閉じる前に変更を保存するか否かを尋ねるダイアログが表示されます。

[[すべて閉じる](#)]は、すべての CodeWarrior IDE のウィンドウを閉じるわけではなく、エディタウィンドウとデバッガウィンドウのみを閉じます。「検索」ダイアログおよびプロジェクトウィンドウは開いたままです。

このコマンドのキーボードショートカットは、Ctrl + Shift + W です。

ヒント： すべてのエディタウィンドウを一度に閉じるには、任意のエディタウィンドウのクローズボックスを Alt + クリックします。

ファイルを印刷

CodeWarrior IDE の印刷オプションを使用して、開いているファイル、プロジェクトファイル、またはウィンドウの内容を印刷できます。

ここでは、以下の項目を説明します。

[印刷オプションを設定](#)

[ウィンドウを印刷](#)

印刷オプションを設定

印刷オプションを設定するには、各プラットフォームの[プリンタ設定]を選択します(プリンタの設定の名称は各プラットフォームによって異なります)。印刷用ダイアログが表示されます。ダイアログを表示するには、[ファイルメニュー](#)の[[ページ設定](#)]またはを選択してください。

このダイアログで用紙サイズ、方向、およびその他の設定を行います。設定とオプションは選択しているプリンタによって変わります。印刷については、お使いのコンピュータおよびプリンタのマニュアルを参照してください。

[OK] ボタンをクリックすると、次回ファイルを印刷するときのためにオプションが保存されます。

ウィンドウを印刷

ウィンドウを印刷するには、ウィンドウをアクティブにして[ファイルメニュー](#)の[[印刷](#)]を選択します。このコマンドにより、アクティブなウィンドウをすべてまたは一部を印刷できます。

このコマンドを選択すると、「印刷」ダイアログが表示されます。エディタウィンドウを印刷する場合、このダイアログには、CodeWarrior 固有の追加のオプションが 2 つあります。ご使用のプリンタおよびプリンタソフトウェアにより、このオプションの表示方法は変わります。

CodeWarrior 独自のオプションを以下に示します。

[選択部分のみ印刷](#)

[カラーシンタックスを利用して印刷](#)

選択部分のみ印刷

印刷するエディタウィンドウでテキストを選択している場合、[選択部分のみ印刷] オプションを設定できます。このオプションがオンの場合、ファイル全体ではなくウィンドウ内の選択されたテキストだけが印刷されます。オフの場合、ファイル全体が印刷されます。

カラーシンタックスを利用して印刷

[カラーシンタックスを利用して印刷] オプションがオンの場合、ファイルはシンタックスカラーで印刷されます。モノクロプリンタでは灰色の階調で印刷されます。オフの場合、シンタックスカラーを使わずファイルは単色で印刷されます。

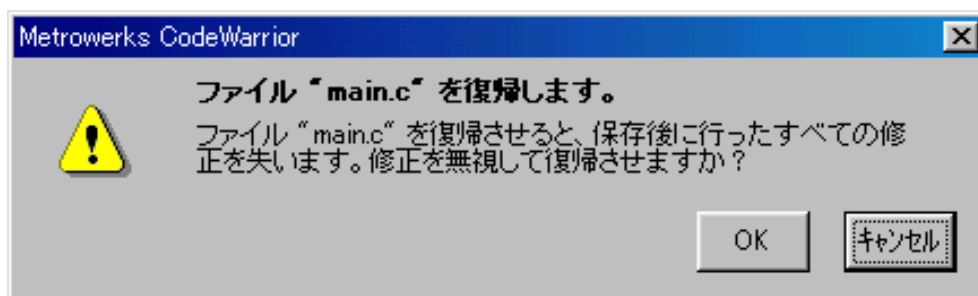
印刷用の設定が終わったら、「印刷」ダイアログで [印刷] ボタンをクリックします。ファイルは印刷ソフトウェアにスプールされます。

「印刷」ダイアログのその他のオプションについては、プリンタまたはプリンタソフトウェアに付属のマニュアルを参照してください。

直前に保存したファイルに戻す

テキストファイルを編集した後で元の内容に戻したくなった場合、[ファイルメニュー](#)の[[復帰](#)] を使います。このコマンドを選択すると警告ダイアログ ([図 4.7](#)) が表示されます。

図 4.7 ファイルを復帰する警告ダイアログ



[OK] ボタンをクリックするとファイルは直前に保存した状態に戻り、保存後に行った変更はすべて失われます。[キャンセル] ボタンをクリックすると作業中のファイルは復帰せず、ディスクへも保存されず、編集を続けることができます。

ファイル / フォルダの比較とマージ

CodeWarrior IDE には 2 つのファイルを比較し、その差分を適用する機能があります。2 つのフォルダの中身を比較する機能もあります。

ここでは IDE のファイル比較機能を説明します。

[ファイル比較とマージについて](#)

[比較ファイルを選択](#)

[差分の検査と適用](#)

[比較フォルダを選択](#)

ファイル比較とマージについて

IDE の「ファイル比較結果」ウィンドウ ([図 4.8](#)) は 2 つのファイルとその差分 (挿入と削除) を表示します。ウィンドウには差分を検査、追加、削除するためのコントロールボタンがあります。選択した差分は濃い色で表示されます。

「ファイル比較結果」ウィンドウには以下の欄があります。

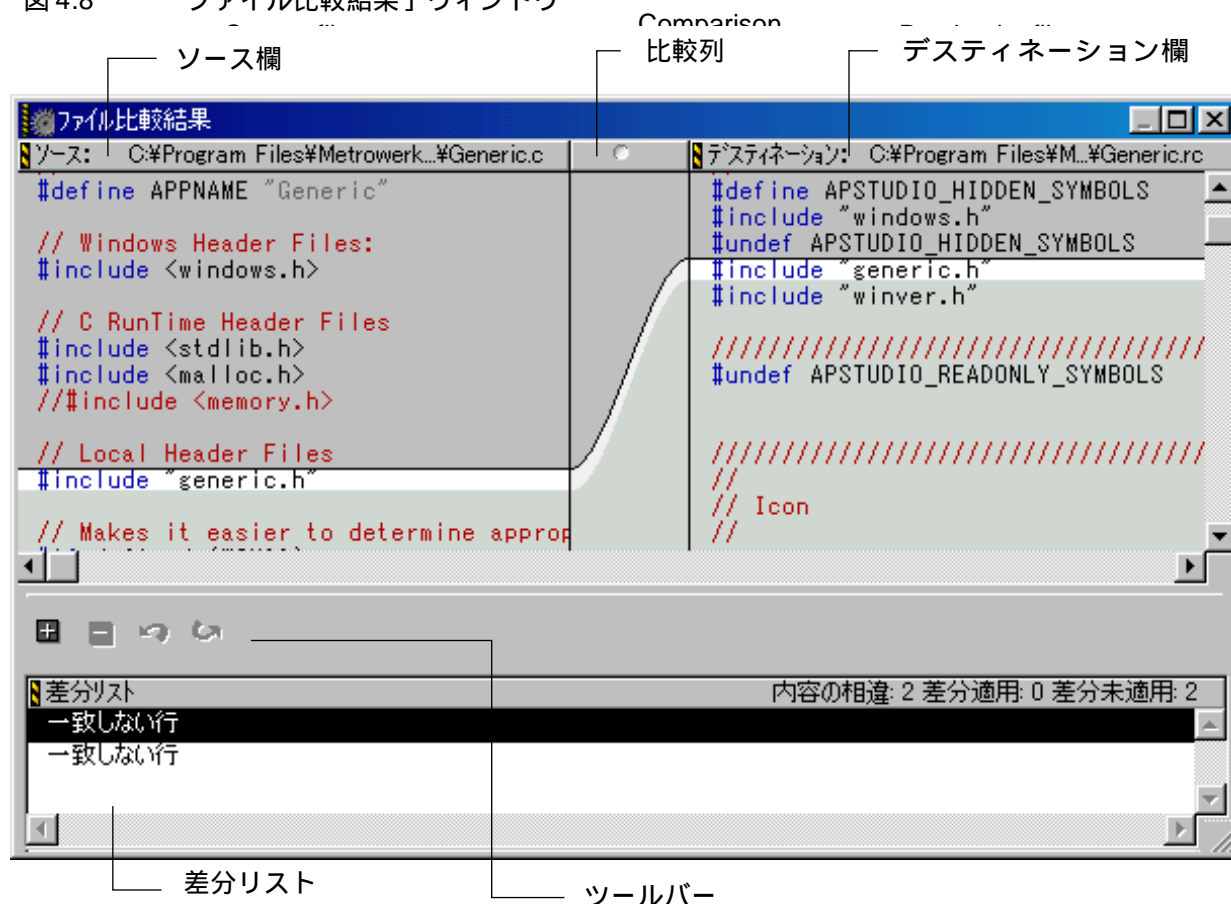
ソース欄

ここには比較の基本となるソースファイルを表示します。この欄は「ファイル比較結果」ウィンドウの左側にあります。

デスティネーション欄

元ファイルと比較するデスティネーションファイルを表示します。この欄は「ファイル比較結果」ウィンドウの右側にあります。ソースファイルとの差分を先ファイルに追加したり、削除することができます。

図 4.8 「ファイル比較結果」ウィンドウ



比較列

ファイルの差分を視覚的に表示します。ソース欄とデスティネーション欄の幅を左右に変更します。この欄は「ファイル比較結果」ウィンドウのソース欄とデスティネーション欄の間にあります。

差分リスト

2つのファイルで違いのある行や挿入、削除を表示します。このリストで項目を選択すると、2つのファイルの差分を表示します。比較列にはファイルがどのように違うのか、またどこにあるのかを表示します。

ツールバー

2つのファイルの差分をソースファイルに適用したり、削除するボタンがあります。[表 4.1](#)はツールバーのボタンの説明です。ソースファイルとデスティネーションファイルへの変更を取り消す、またはやり直すボタンもあります。ツールバーの設定については[「IDE をカスタマイズ」\(p207\)](#)を参照してください。

表 4.1 ツールバーのボタンとコマンド

ボタン	説明
	差分を適用
	差分の適用を取り消す
	取り消し
	やり直し

比較ファイルを選択

「ファイル比較設定」ダイアログを開くには、[検索メニュー](#)の[[ファイル比較](#)]を選択します。比較元となるソースとデスティネーションを選択するためのダイアログが表示されます。テキストファイルを該当するフィールドにドラッグ&ドロップすることもできます。

テキスト比較オプション

[テキスト比較オプション] 欄には 2 つのオプションがあります。

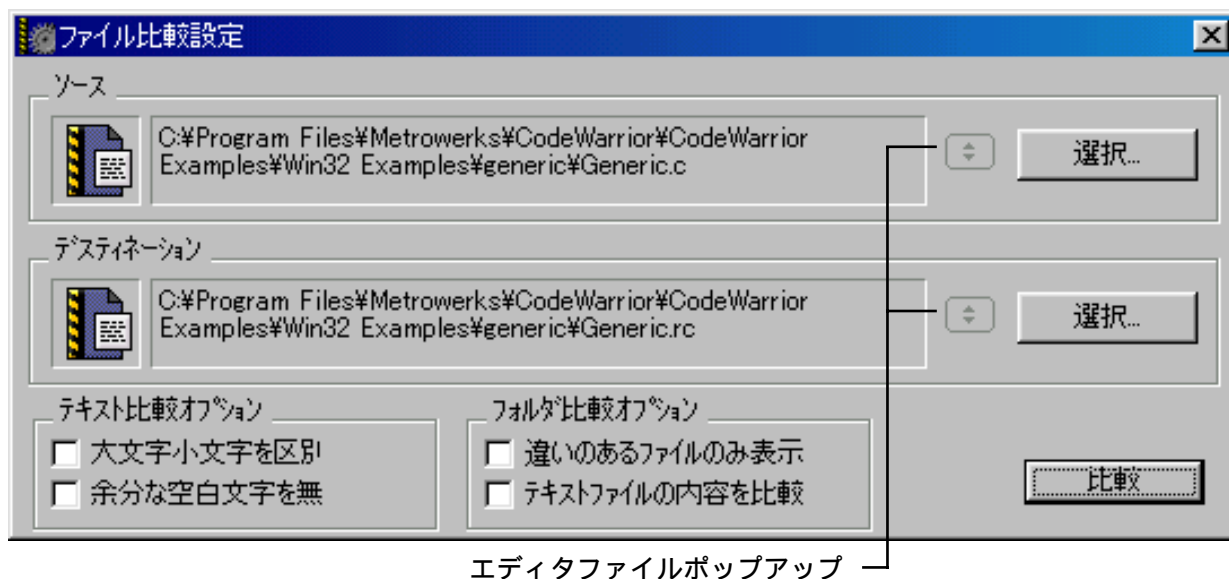
大文字、小文字の区別をする場合は、[大文字小文字を区別] チェックボックスをオンにします。区別しない場合はオフにします。

余分な空白文字 (余分なスペース、タブ、改行文字) を無視する場合、[余分な空白文字を無視] チェックボックスをオンにします。比較する場合、オフにします。

フォルダ比較オプションについては[「フォルダ比較オプション」\(p92\)](#)を参照してください。

比較するファイルを選択してオプションを設定した後、[比較] ボタンをクリックすると「ファイル比較結果」ウィンドウが表示されます。

図 4.9 「比較ファイル設定」ダイアログ



差分の検査と適用

元ファイルと先ファイルの差分を、先ファイルに適用するには比較ウィンドウのツールバーと差分リストを使います。

2 つのファイルの差分を見るには、差分リストの項目をクリックします。差分を適用するには、ツールバーの[差分を適用] アイコンボタンをクリックするか、[検索メニュー](#)の[[差分を適用する](#)] を選択します。すでに適用した変更を取り消すには [差分の適用を取り消す] アイコンボタンをクリックするか、[検索メニュー](#)の[[差分の適用を取り消す](#)] を選択します。

警告！ [+] または [-] ボタンを実行するとスタックからすべての操作が消えてしまいます。つまり差分を適用、または適用を取り消した後では編集操作の取り消し、やり直しはできません。

エディタファイルを比較

エディタウィンドウで開いているファイルを比較するには、デスティネーションパスの横の [エディタファイル] ポップアップメニュー ([図 4.9](#)) を使います。開いているエディタウィンドウのリストが表示されるのでここからファイルを選択します。ソースとデスティネーションを選択したら、[比較] ボタンをクリックします。

比較フォルダを選択

「ファイル比較設定」ダイアログを開くには、[検索メニュー](#)の[[ファイル比較](#)]を選択します。比較するソースフォルダとデスティネーションフォルダを選択するダイアログが現れます。各欄にフォルダをドラッグ&ドロップして比較フォルダを設定します ([図 4.10](#))。

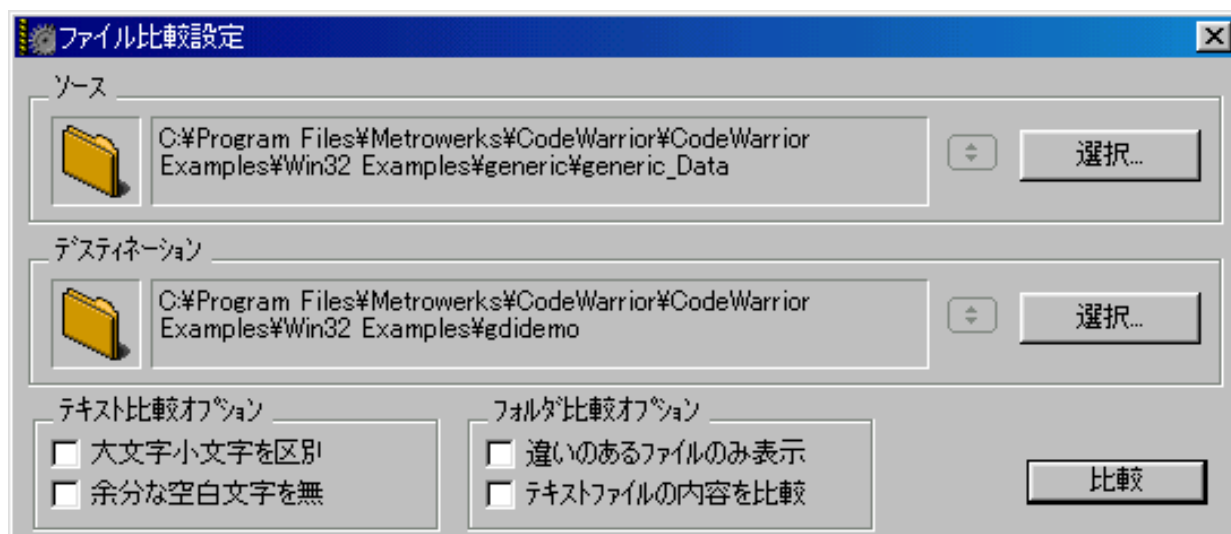
フォルダ比較オプション

[フォルダ比較オプション] 欄には2つのオプションがあります。

[違いのあるファイルのみ表示] オプションをオンにすると、「フォルダ比較結果」ウィンドウに両方のフォルダの異なるファイルだけが表示されます ([図 4.11](#))。デフォルトではこのオプションはオフなので、ソースとデスティネーションフォルダにあるすべてのファイルがリストに表示されます。

ソースフォルダとデスティネーションフォルダのファイルの比較は修正日時とサイズに基づいて行われます。通常でこれはファイルの違いを調べるのに十分です。不可視の項目は比較されません。

図 4.10 「ファイル比較設定」ダイアログ



[テキストファイルの内容を比較] をオンにするとより正確な比較が行われます。これは修正日時やサイズではなく、ソースとデスティネーションフォルダにあるファイルをそれぞれ比較します。このため比較には時間がかかりますが情報は非常に正確です。

ファイル比較のオプションについては「[テキスト比較オプション](#)」(p90) を参照してください。

[比較] ボタンをクリックすると、IDE は「フォルダ比較結果」ウィンドウを開きます ([図 4.11](#))。ソースファイル、ヘッダファイル、テキストファイル、フォルダは標準のスタイルで表示されます。その他のファイルは斜体で表示されます。

[違いのあるファイルのみ表示] オプションがオフの場合、「両方のフォルダにあるファイル」リストはソースとデスティネーションフォルダにあるファイルをすべて表示します。両フォルダで異なるファイルには名前の右側に黒い点が表示されます。

[両方のフォルダにあるファイル] [ソースにだけあるファイル] [デスティネーションにだけあるファイル] リストでファイルをクリックすると、そのファイルが比較ウィンドウの [選択項目] 欄に表示されます。

図 4.11 「フォルダ比較結果」ウィンドウ



[両方のフォルダにあるファイル] リストでファイルをダブルクリックすると、ファイル比較ウィンドウが開きます。ここで 2 つのファイルの違いを見ることができます。

[ソースにだけあるファイル] リストはソースフォルダだけにあるファイルを示します。逆に、「デスティネーションにだけあるファイル」リストはデスティネーションフォルダだけにあるファイルを示します。

ズームボックスでリストをウィンドウの大きさに広げることができます。再度クリックすると元の大きさに戻ります。

第 5 章 ソースコードの編集

この章では、ソースコードを編集するための CodeWarrior IDE テキストエディタの使い方を説明します。

CodeWarrior エディタは、特にプログラマ向けに設計された高機能なテキストエディタです。以下の機能を備えています。

各エディタウィンドウにヘッダファイル用のポップアップメニューがあり、関連するヘッダファイルや関数へ簡単に移動できます。

シンタックスカラーリングにより、ソースコード中のコメントやキーワードが簡単に識別できます。

マウスでポイントしてクリックするだけで、各ルーチンの便利なオンラインリファレンスへアクセスできます。

この章では、以下の項目について説明します。

[エディタウィンドウの解説](#)

[エディタウィンドウの設定](#)

[基本的なテキスト編集](#)

[テキストをナビゲート](#)

[オンラインリファレンス](#)

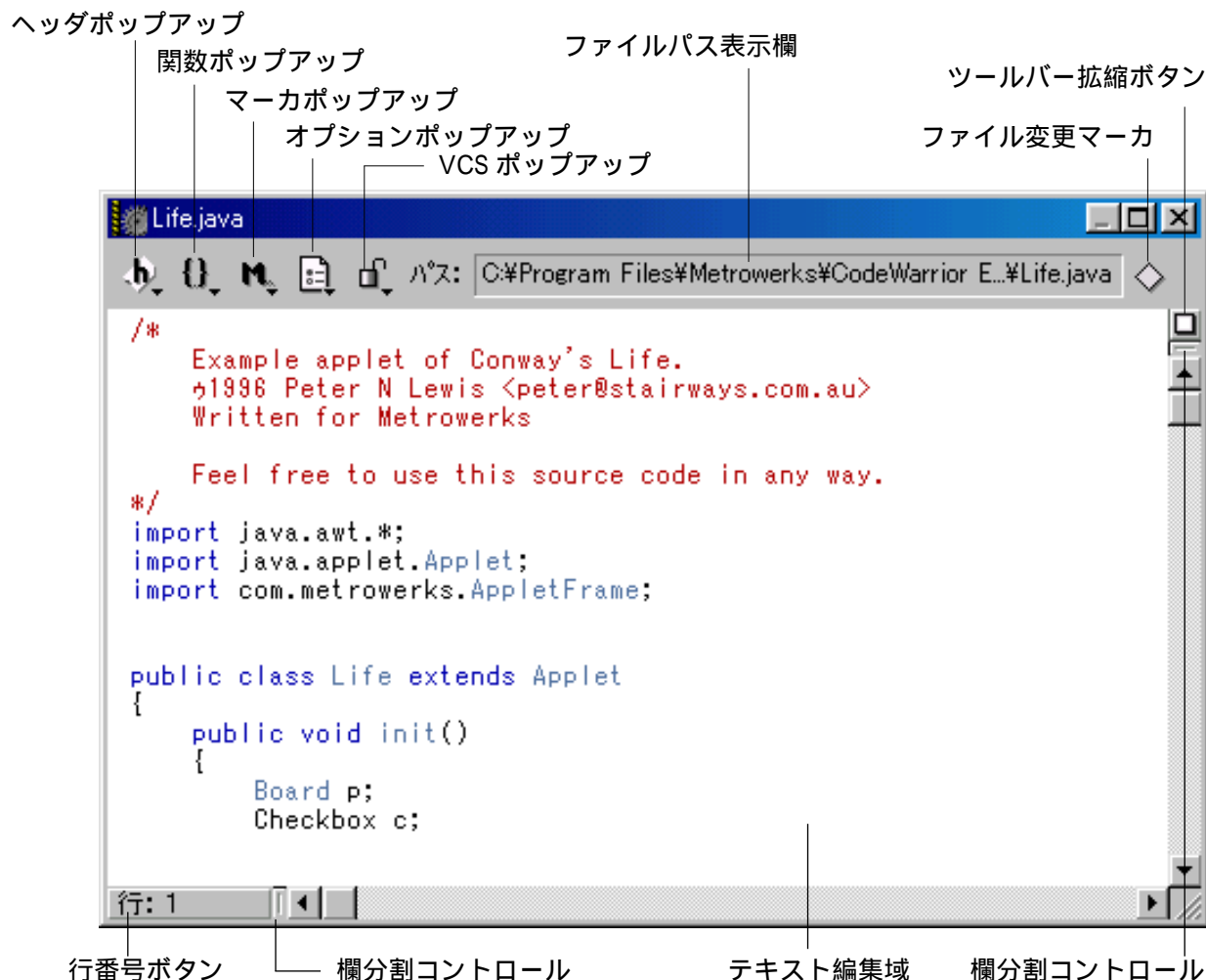
CodeWarrior エディタの動作に関するオプションをカスタマイズすることもできます。詳細は「[エディタ設定](#)」(p194) を参照してください。

エディタウィンドウの解説

CodeWarrior エディタウィンドウ ([図 5.1](#)) には、ソースファイルの表示および編集に便利な要素が含まれています。

エディタウィンドウを開くには、[ファイルメニュー](#)の [[新規テキストファイル](#)] を選択して、新しいテキストファイルを作成します。

図 5.1 エディタウィンドウ



ここでは、エディタウィンドウ（[図 5.1](#)）の各部分を説明します。

[テキスト編集域](#)

[ヘッダポップアップメニュー](#)

[関数ポップアップメニュー](#)

[マーカポップアップメニュー](#)

[オプションポップアップメニュー](#)

[VCS ポップアップメニュー](#)

[ファイルパス表示欄](#)

[ファイル変更マーカ](#)

[欄分割コントロール](#)

[行番号](#)[ツールバー拡張ボタン](#)

テキスト編集域

エディタウィンドウのテキスト編集域は、テキストを入力、編集する場所です。

エディタウィンドウ内のテキストを選択して、別に開いたエディタウィンドウなどのドロップ対応の場所へドラッグすることができます。逆に、ドラッグ & ドロップ対応のアプリケーションで選択したテキストを、エディタウィンドウにドラッグすることもできます。

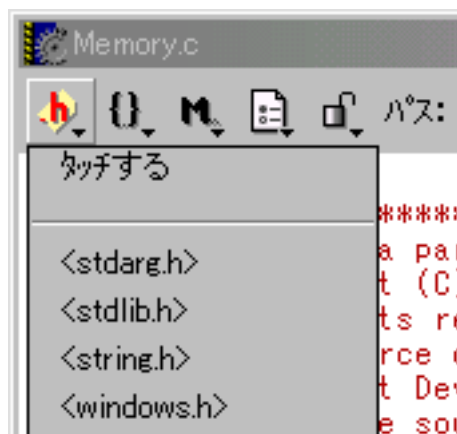
テキストのドラッグ & ドロップ操作の詳細は、[「テキストを移動（ドラッグ & ドロップ）」\(p107\)](#) を参照してください。

ヘッダポップアップメニュー



ヘッダポップアップメニュー（[図 5.2](#)）から、現在のファイルが参照しているヘッダファイルを開くことができます。このメニューから [タッチする] および [アンタッチ] コマンドも選択できます。

図 5.2 ヘッダポップアップメニュー



リスト内のファイルを開くには、表示したいファイル名を選択します。プロジェクトファイルが開いていなければ、メニューにファイルのリストは表示されません。また、プリコンパイルヘッダやライブラリなど、開くことができないファイルもあります。

ファイルを開く方法については、[「既存ファイルを開く」\(p77\)](#) を参照してください。

プロジェクトを次にリビルドするとき、ファイルを再コンパイルするには、[タッチする] コマンドを選択します。[アンタッチ] コマンドを選択すると、ファイルはコンパイルされません。

ファイルのタッチについては[「ファイルのタッチ / アンタッチ」\(p63\)](#) を参照してください。

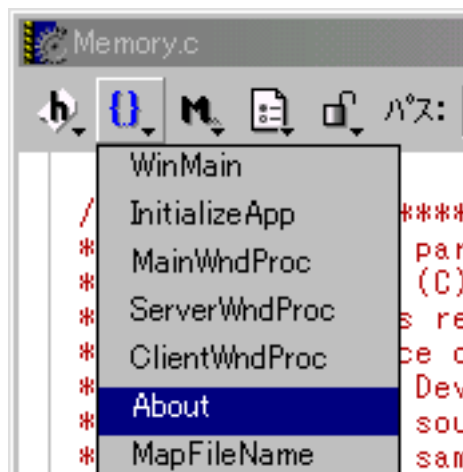
関数ポップアップメニュー



関数ポップアップメニュー（[図 5.3](#)）を利用すると、ソースファイル内の関数を簡単にナビゲートすることができます。

関数ポップアップメニューが空の場合、表示されているファイルはソースファイルではありません。メニュー内でハイライト表示されている関数は、現在テキスト挿入位置が置かれている関数を示しています。

図 5.3 関数ポップアップメニュー



注意： デフォルトでは、ポップアップメニューはソースファイル内に書かれている順番で関数を表示します。関数をアルファベット順にソートするには、関数ポップアップのアイコンを Ctrl + クリックします。

デフォルトで関数をアルファベット順に表示するには、[[関数ポップアップメニューをソート](#)] オプションをオンにします。詳細は [「エディタ設定」\(p194\)](#) を参照してください。

ヒント： Pascal ファイルを編集しているとき、関数ポップアップメニューはプロシージャ名を斜体、関数名を普通の書体、メインプログラムを太字で表示します。

マーカポップアップメニュー



マーカポップアップメニュー（[図 5.4](#)）でテキストファイルにマーカを追加、削除できます。このマーカは、特定のテキスト行へアクセスしたり、テキスト行を覚えておくための付箋として使うことができ便利です。

マーカの使い方の詳細は [「マーカを追加、削除、選択」\(p110\)](#) を参照してください。

図 5.4 マーカポップアップメニュー



オプションポップアップメニュー



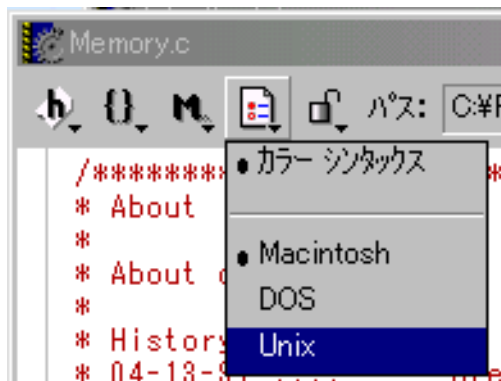
オプションポップアップメニュー(図 5.5)で現在のファイルのシンタックスカラーリングを設定したり、テキストファイルを保存するフォーマットを設定します。

オプションポップアップメニューの [カラーシンタックス] でテキストのカラー表示のオン/オフを設定します。

また、このポップアップメニューで、テキストファイルを次に保存するフォーマット ([Macintosh], [DOS], [UNIX]) を選択できます。現在開いているファイルのフォーマットの横にはマーカが付いています。別のフォーマットでファイルを保存するには、ポップアップメニューから希望のフォーマットを選択します。ここで選択したフォーマットでテキストファイルは保存されます。

[カラーシンタックス] オプションの詳細は「[カラーシンタックス](#)」(p198) を参照してください。

図 5.5 オプションポップアップメニュー



VCS ポップアップメニュー

VCS ポップアップメニューはバージョン管理システムに登録されているファイルに対して、読み取り / 書き込みデータベースの状態を示します。VCS ポップアップメニューのア

アイコンはファイルが変更可能か否かを示します。アイコンとその意味は「[一般的な VCS の操作](#)」(p452) を参照してください。

このポップアップメニューからファイルのコピーを作成したり、変更用にファイルをチェックアウトしたり、チェックアウトせずに変更できるように書き込み可能にしたり、その他の操作ができます。

バージョン管理システム (VCS : Version Control System) の詳細は「[バージョン管理システムを使う](#)」(p443) を参照してください。



ファイルパス表示欄

CodeWarrior IDEは自動的にファイルパス表示欄に現在のファイルへのディレクトリパスを示します。ファイルパス表示欄はウィンドウの右上にあります ([図 5.1 \(p96 \)](#))。

ファイル変更マーカ

ファイル変更マーカは、現在表示しているファイルが直前の保存以降に変更されているか否かを示します。[表 5.1](#) にファイル変更マーカの意味を示します。

表 5.1 ファイル変更マーカの意味

マーカ	状態
	ファイルは変更されていない
	ファイルは変更されている (変更が保存されていない)

欄分割コントロール

欄分割コントロールは、エディタウィンドウを欄に分割します。1 つのウィンドウ内にファイルの別の部分を表示することができます。

このコントロールを使って各欄のサイズを調整できます。[図 5.7 \(p103 \)](#) に複数の欄を使ったエディタウィンドウの例を示します。

詳細は「[ウィンドウを欄に分割](#)」(p102) を参照してください。

行番号

行番号ボタン ([図 5.1](#)) は、テキスト挿入位置がある行の番号を表示します。このボタンを使ってファイル内の他の行にジャンプすることができます。

テキスト挿入位置を他の行に設定する方法は、「[特定の行にジャンプ](#)」(p112) を参照してください。

ツールバー拡張ボタン

ツールバー拡張ボタン ([図 5.6 \(p102\)](#)) で、エディタウィンドウのツールバーを表示したり隠したりできます。ツールバーが隠れている場合、ウィンドウの下に小さなコントロールボタンが表示されます。

ポップアップメニュー拡張ボタンの詳細は、[「ウィンドウコントロールを表示」 \(p101\)](#) を参照してください。

エディタウィンドウの設定

ソースコードを表示するエディタウィンドウをカスタマイズすることができます。以下の項目について説明します。

[文字サイズとフォントを指定](#)

[ウィンドウコントロールを表示](#)

[ウィンドウを欄に分割](#)

[エディタウィンドウの設定を保存](#)

エディタウィンドウのツールバーについては [「IDE をカスタマイズ」 \(p207\)](#) を参照してください。

文字サイズとフォントを指定

フォント & タブ設定パネルでエディタウィンドウに表示するテキストのサイズ、フォントを指定します。詳細は [「フォント & タブ」 \(p196\)](#) を参照してください。

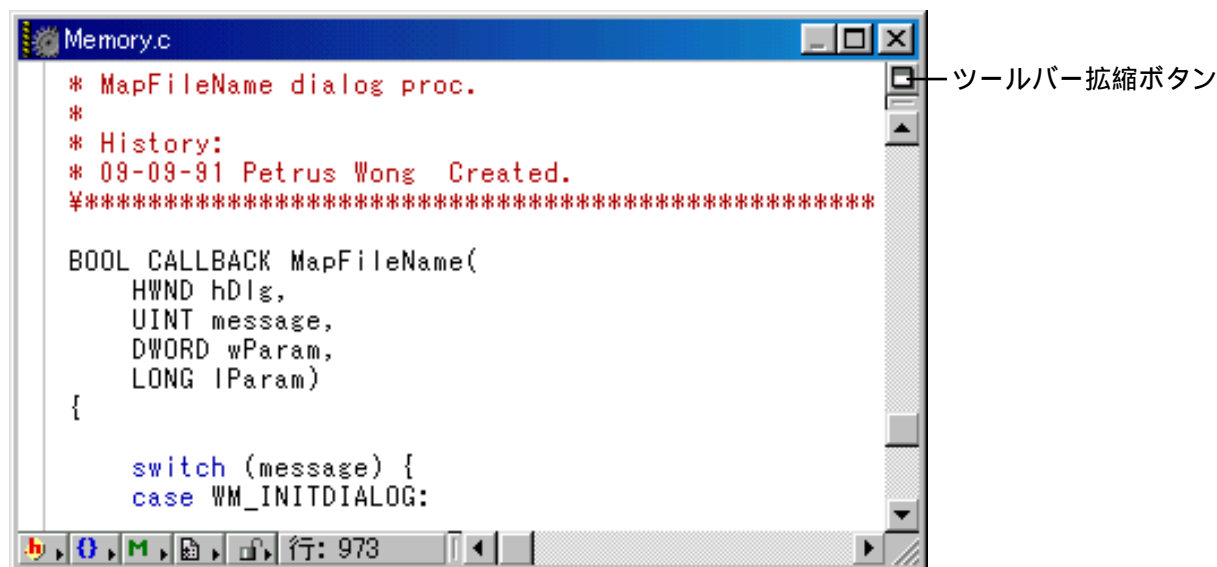
ウィンドウコントロールを表示

エディタウィンドウの上部に並んでいるポップアップメニューとコントロールは、ツールバーと呼ばれます。[ツールバー拡張ボタン](#) ([図 5.1 \(p96\)](#)) をクリックすると、ツールバーを表示したり、隠したりします。

ツールバーを隠すと、デフォルトのポップアップメニューコントロールがエディタウィンドウの下部に表示されます ([図 5.6](#))。 [ファイルパス表示欄](#) は表示されません。

注意： エディタウィンドウのツールバーをカスタマイズした場合、独自に設定した項目はウィンドウ下部には表示されません。ツールバーを隠したときに表示されるのはデフォルトの項目だけです。ツールバーを再度表示すると、カスタム設定のまま表示されます。ツールバーのカスタマイズについては [「IDE をカスタマイズ」 \(p207\)](#) を参照してください。

図 5.6 エディタウィンドウ下部のポップアップメニューコントロール



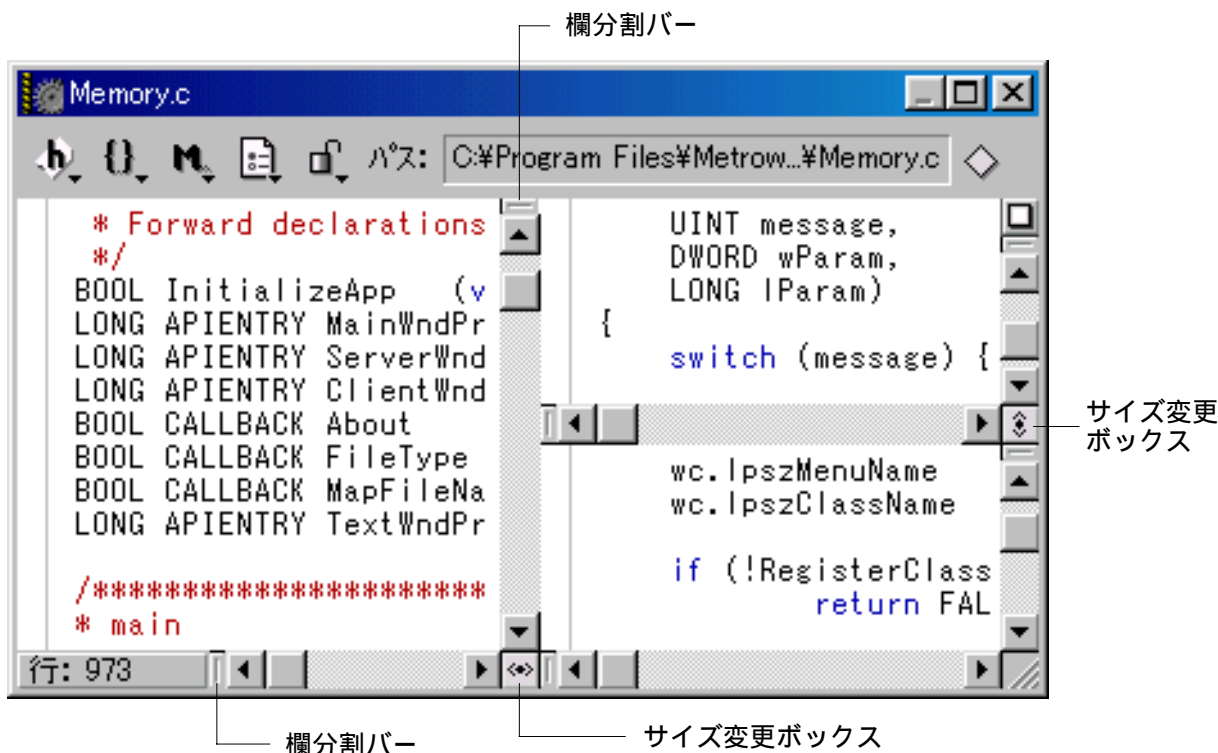
ツールバーを再度エディタウィンドウ上部に表示するには、ツールバー拡張ボタンを再度クリックします。

デフォルトのツールバーを表示するか隠すかを選択することもできます。[「ツールバーを表示する / 隠す」\(p221\)](#)を参照してください。


ウィンドウを欄に分割

エディタウィンドウを欄に分割することもできます。これにより、同一のウィンドウでファイルの別の部分を表示できます（[図 5.7](#)）。ここでは複数の欄の作成、調整、削除について説明します。

図 5.7 ウィンドウ内の複数の欄




新しい欄を作成

 エディタウィンドウ内に新しい欄を作成するには、欄分割バーをクリックしてドラッグします。欄分割バーは、エディタウィンドウの各スクロールバーの上端または左端についています。

欄分割バーをドラッグすると灰色の線が欄分割バーとともに動き、新しい欄のサイズを示します。マウスボタンを離すと新しい欄が作成されます。


欄分割バーをダブルクリックすると、同じ大きさの 2 つの欄に分かれます。

欄のサイズを変更

 エディタウィンドウ中の欄のサイズを変更するには、サイズ変更ボックスをクリックしてドラッグします。

サイズ変更ボックスをドラッグすると灰色の線がサイズを示します。マウスボタンを離すと新しい位置で欄が再表示されます。

欄を削除

 エディタウィンドウから欄を削除するには、サイズ変更ボックスをウィンドウの端までドラッグします。

サイズ変更ボックスをドラッグすると、灰色の線がサイズの変更具合を示します。ウィンドウの端近くまでドラッグすると、灰色の線が消えます。このときマウスボタンを離すと、ウィンドウから欄が 1 つ削除されます。

サイズ変更ボックスをダブルクリックすると、分割が削除されます。

エディタウィンドウの設定を保存

ウィンドウを閉じたとき、またはツールバーの表示を切り替えたときに現在のエディタウィンドウの設定が自動的に保存されます。

保存されるのは、ウィンドウのサイズと位置、ツールバーの表示の設定です。次にこのエディタウィンドウを開くとき、同じ設定が使われます。

エディタウィンドウの設定の詳細は [「フォント & タブ」\(p196\)](#) を参照してください。

CodeWarrior ブラウザでこのコマンドを使う方法は、[「デフォルトのブラウザを保存」\(p179\)](#) を参照してください。

基本的なテキスト編集

CodeWarrior IDE にはソースファイルの編集に便利な多くの機能があります。

ここでは、以下の項目を説明します。

[ウィンドウナビゲーションの基本](#)

[テキストを追加](#)

[テキストを削除](#)

[テキストを選択](#)

[テキストを移動 \(ドラッグ & ドロップ\)](#)

[カット、コピー、ペースト、消去](#)

[括弧のバランスをとる](#)

[テキストを左右に移動](#)

[変更を取り消す](#)

[カラーをコントロール](#)

ウィンドウナビゲーションの基本

CodeWarrior IDE には、ファイル内でテキスト挿入位置を移動する方法が複数あります。スクロールバーとキーボードによるナビゲーションが一般的な方法です。

スクロールバーによるナビゲーション

他のテキストエディタ同様、CodeWarrior エディタでもスクロールバーでエディタウィンドウ内を移動してテキストを見ることができます。

キーボードによるナビゲーション

[表 5.2](#) に、ファイル内で挿入位置を移動するためのデフォルトのキーバインディングを示します。キーバインディングとは、キーの組み合わせを意味する CodeWarrior 用語です。

表 5.2 キーボードによるテキストナビゲーション

挿入位置の移動先	Windows
直前の語	Ctrl +
次の語	Ctrl +
行の先頭	Home
行末	End
ファイルの先頭	Ctrl + Home
ファイルの終わり	Ctrl + End

[表 5.3](#) に、挿入位置を移動せずにファイルの表示位置をスクロールする方法をまとめます。

表 5.3 キーボードによるスクロール

スクロールする先	Windows
前のページ	Page Up
次のページ	Page Down
ファイルの先頭	Ctrl+ Home
ファイルの終わり	Ctrl-End
前の行	Ctrl+
次の行	Ctrl+

テキストを追加

開いているファイルにテキストを追加するには、ウィンドウの[テキスト編集域](#)を1度クリックします。挿入位置が表示されたら、キーボードからテキストを入力できます。

エディタウィンドウ内で挿入位置を移動する方法は、「[ウィンドウナビゲーションの基本](#)」([p104](#))を参照してください。

テキストを削除

テキストの削除方法は複数あります。

入力したばかりのテキストを削除するには、Backspace キーを押します。

テキスト挿入位置の前にあるテキストを削除するには、Delete キーを押します。

連続した複数の文字を一度に削除するには、テキストを選択して Backspace キーを押します。

テキストの選択方法は「[テキストを選択](#)」を参照してください。

テキストを選択

エディタウィンドウでテキストを選択する方法は複数あります。

単語、行、任意の範囲のテキストを選択できます。Shift キーを押しながら [表 5.2](#) のキーバインディングを使うと、そこまでのテキストを選択することができます。

単語を選択するには：

単語をダブルクリックします。

行を選択するには：

行の上でトリプルクリックします。

エディタウィンドウの左端までマウスポインタを移動し、マウスポインタが右向きになったときにクリックします。

この方法は [エディタ設定](#) パネルで [[左端をクリックして行を選択する](#)] オプションがオンの場合のみ使えます。

任意の範囲のテキストを選択するには：

マウスボタンを押したまま、マウスポインタを選択したい範囲までドラッグしてからボタンを離します。

挿入位置を選択範囲の一方の端に設定します。その後、Shift キーを押しながら、反対側の端をクリックします。挿入位置と Shift + クリックした位置の間のテキストが選択されます。

エディタウィンドウの左端までマウスポインタを動かしてクリックし、選択したい範囲のテキストまでドラッグします。この方法は [エディタ設定](#) で [[左端をクリックして行を選択する](#)] がオンの場合のみ使えます。

エディタウィンドウ内で 1 つの関数全体を選択するには、Shift キーを押しながら [関数ポップアップメニュー](#) で関数を選択します。これは、関数をコピーするとき、またはドラッグ & ドロップでファイル内を移動するときに便利な方法です。

[表 5.4](#) にキーボードでテキストを選択する方法をまとめます。現在の挿入位置から始めます。

表 5.4 キーボードによるテキスト選択

～までのテキストを選択	Windows
前の語	Shift + Ctrl+
次の語	Shift + Ctrl+
行の先頭	Shift + Home

～までのテキストを選択	Windows
行末	Shift + End
ページの先頭	Shift + Page Up
ページの終わり	Shift + Page Down
ファイルの先頭	Shift + Ctrl+ Home
ファイルの終わり	Shift + Ctrl+ End

テキストのドラッグ & ドロップの詳細は、「[テキストを移動 \(ドラッグ & ドロップ\)](#)」を参照してください。

[バランス] コマンドでコードのブロックを簡単に選択できます。「[括弧のバランスをとる \(p107\)](#)」を参照してください。

テキストを移動 (ドラッグ & ドロップ)

ドラッグ & ドロップ機能を使って、選択したテキストを新しい場所へ移動することができます。ドラッグ & ドロップ機能の設定は「[エディタ設定](#)」(p194) を参照してください。

CodeWarrior エディタは、他のドラッグ & ドロップ対応アプリケーションからテキストを受け取ることもできます。アプリケーションがドラッグ & ドロップをサポートしているかどうかは付属マニュアルを参照してください。

カット、コピー、ペースト、消去

標準的な編集コマンドにカット、コピー、ペーストがあります。これらのコマンドは CodeWarrior IDE の[編集メニュー](#)にあります。

これらのコマンドを使ってウィンドウ内、ウィンドウ間、およびアプリケーション間でテキストのコピーとペーストができます。

これらのコマンドの詳細は「[編集メニュー](#)」(p459) を参照してください。

括弧のバランスをとる

CodeWarrior IDE は括弧 ()、大括弧 []、中括弧 { } が対応しているか否かをチェックすることができます。

括弧、大括弧、中括弧の対応を検査するには、検査したいテキストに挿入位置を置いて[編集メニュー](#)の [[バランス](#)] を選択します。または、検査したい括弧、大括弧、中括弧をダブルクリックします。

CodeWarrior エディタはテキスト挿入位置から括弧、大括弧、または中括弧が見つかるまでファイルを検索します。次に、それに対応する括弧が見つかるまで反対方向に検索します。対応する括弧が見つかったら、その括弧の間のテキストを選択します。括弧のバランスがとれていない場合、ピープ音を鳴らします。例えば、挿入位置のあるテキストが括弧で括られていない場合にピープ音を鳴らします。

ヒント：[[バランス](#)] コマンドを使うとコードブロックを簡単に選択できます。

自動的に括弧のバランスをとる

CodeWarrior エディタは括弧のバランスを自動的に検査できます。詳細は「[入力時に括弧の対応を確認する](#)」(p195) を参照してください。

テキストを左右に移動

[編集メニュー](#)の[[左に移動](#)]、[[右に移動](#)] コマンドでテキストブロックを左右に移動することができます。

テキストブロックを選択した後、[編集メニュー](#)の[[左に移動](#)] または [[右に移動](#)] を選択します。テキスト選択については「[テキストを選択](#)」(p106) を参照してください。

CodeWarrior エディタは、選択した行の先頭にタブを挿入または削除することにより、選択したテキストを 1 タブ分だけ右または左に移動します。

字下げで使うスペースの数を変える方法は「[フォント & タブ](#)」(p196) を参照してください。

変更を取り消す

CodeWarrior エディタは、編集集中に間違いを取り消す方法があります。

直前の編集を取り消す

[[取り消し](#)] コマンドは直前の操作を取り消します。[編集メニュー](#)の[[取り消し](#)] コマンドは最後に行った操作により変化します。例えば、テキストを入力した場合は[入力取り消し] に変わります。[入力取り消し] を選択すると入力したテキストが削除されます。

複数の操作を取り消す、またはやり直す

[[複数アンドゥを利用](#)] オプションがオンの場合、複数の操作の取り消し、または再実行ができます。

例えば、単語をカットしてペーストし、その後テキストを入力したとします。[[取り消し](#)] コマンドを 3 回使うと、これらの操作すべてを元に戻すことができます。最初の [取り消し] は入力したテキストを削除し、2 回目の [取り消し] はペーストしたテキストを削除し、3 回目の [取り消し] はカットしたテキストを元に戻します。これでテキストは元の状態に戻ります。[[再実行](#)] コマンドを 3 回使うと、同じ操作を同じ順序で再度行うことができます。

[[複数アンドゥを利用](#)] オプションをオンにする方法は、「[複数アンドゥを利用](#)」(p196) を参照してください。

[[複数アンドゥを利用](#)] オプションがオフの場合、[再実行] のキーボードショートカットが [取り消し] のキーボードショートカットと同じになります。

警告！ [取り消し] 操作はスタックに保存されます。そのため、複数回の [取り消し] [再実行] 操作は失われる可能性があります。1 回の [取り消し] はスタックに 1 項目を増やし、[再実行] は次の [取り消し] 操作へのポインタとなります。例えば、スタックに 5 回の [取り消し] 操作 (ABCDE) があり、2 回を [再実行] した場合、スタックのポインタは ABC となり、次に別の操作をすると ABCF となります。DE の [取り消し] はできなくなります。

ファイルを直前に保存した状態に戻す

[ファイルメニュー](#)の [復帰] コマンドは、ファイルを最後に保存した状態に戻します。詳細は「[直前に保存したファイルに戻す](#)」(p87) を参照してください。

カラーをコントロール

ソースコード中のコメント、キーワード、および引用符で囲んだ文字列などをカラーでハイライト表示することができます。ハイライト表示した文字は見分けやすく、またカラーパターンを識別することによりスペルや文法を検査できます。カラーシンタックスオプションの設定方法は、「[カラーシンタックス](#)」(p198) を参照してください。

単語を指定して、独自のキーワードをハイライトすることもできます。この設定方法は「[カラーシンタックス](#)」(p198) を参照してください。

テキストをナビゲート

CodeWarrior エディタには、ソースファイル内をナビゲート (移動) する方法がいくつかあります。

ここでは、以下の方法を紹介します。

[関数を検索](#)

[マーカを追加、削除、選択](#)

[関連するファイルを開く](#)

[特定の行にジャンプ](#)

[戻ると次へ](#)

[エディタのコマンドを設定](#)

また、ソースコードブラウザにはコード内をナビゲートするための強力な機能があります。CodeWarrior ブラウザの使用法は「[ソースコードのブラウズ](#)」(p143) を参照してください。

ファイル内でテキスト挿入位置を移動するためのキーバインディングは変更可能です。詳細は「[エディタ](#)」(p486) を参照してください。

関数を検索



関数アイコンをクリックすると、関数ポップアップメニューが表示されます（「[関数ポップアップメニュー](#)」(p98) を参照）。ここで選択した関数へジャンプします。

注意： ポップアップが空の場合、ファイルはソースファイルではありません。

マーカを追加、削除、選択

CodeWarrior エディタの機能を使って、あらゆるテキストファイルにマーカを追加または削除できます。マーカはしおりのようなものです。ファイルの特定の位置にジャンプしたり、作業中のコードに印を付けることのできる便利な機能です。

マーカを追加

マーカを追加するには、マークしたいテキスト位置に挿入位置を置いて[マーカポップアップメニュー](#)の「マーカを追加」を選択します。「マーカを追加」ダイアログ ([図5.8](#)) が現れます。

図 5.8 「マーカを追加」ダイアログ



[新規マーカの名称] フィールドには、メモ、コメント、関数名、または検索の助けになる情報など好きなものを入力してください。

入力した後 [追加] ボタンをクリックすると、次回[マーカポップアップメニュー](#)を表示したときに追加したマーカが表示されるようになります（[図 5.9](#)）。

ヒント： ソースファイルでテキストを選択してから [マーカを追加] を選択すると、そのテキストが [新規マーカの名称] フィールドに表示されます。簡単に関数や行をマーカとして追加することができます。

図 5.9 マーカを追加したテキストファイル



#pragma でマーカを追加

ファイルにマークを付けるにはもう 1 つ、より長期的な方法があります。以下は C/C++ 言語でマーカを作成する例です。

```
#pragma mark myMarker
```

Pascal でマーカを作成する例です。

```
{ $PRAGMA MAC MARK myMarker }
```

上記の例ではエディタでファイルを開いたとき、[関数ポップアップメニュー](#)に `myMarker` が自動的に追加されます。

[マーカポップアップメニュー](#)で作成したマーカと異なり、この `#pragma` で定義したマーカは関数ポップアップメニューに表示されます。

マーカを削除

マーカを削除するには、[マーカポップアップメニュー](#)をクリックして、[マーカーを削除] コマンドを選択します。「マーカーを削除」ダイアログ ([図 5.10](#)) が現れます。ここで削除したいマーカを選択した後、[削除] ボタンをクリックするとマーカがリストから削除されます。作業を終えたら [終了] ボタンをクリックして「マーカーを削除」ダイアログを閉じてください。

図 5.10 「マーカを削除」ダイアログ



マーカヘジャンプ

ファイル内の特定のマーカヘジャンプするには、[マーカポップアップメニュー](#)からそのマーカを選択します。テキスト挿入位置がそのマーカヘジャンプします。

関連するファイルを開く

アクティブなエディタウィンドウに関連するファイルを開く方法はいくつかあります。例えばソースファイルで使われているヘッダファイルを見ることができます。

現在のファイルが参照しているヘッダファイルを開くには、[ヘッダポップアップメニュー](#) ([図 5.2 \(p97\)](#)) を使います。ここで選択したファイルが開きます。このポップアップメニューで現在のファイルを [タッチする] または [アンタッチ] することができます。

ソースファイルが使っているヘッダファイルを開くには、もう 1 つ別の方法があります。関連ファイルを開くには、アクティブなウィンドウ内でファイル名を選択した後、Ctrl + D キーを押します。ファイルを開くコマンドの詳細は、[「既存ファイルを開く」\(p77\)](#) を参照してください。

特定の行にジャンプ

行番号が分かっている場合は、エディタウィンドウ中のその行ヘジャンプできます。行番号は最初の行を 1 として順に付けられます。

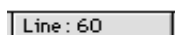
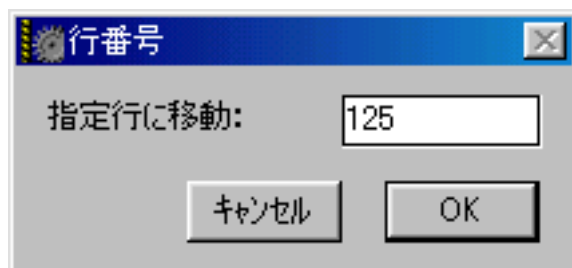
 エディタウィンドウの [[行番号](#)] ボタンをクリックして、「行番号」ダイアログを開きます ([図 5.11](#))。ここで、ジャンプ先の行番号を入力して [OK] ボタンをクリックします。

図 5.11 「行番号」ダイアログ



戻ると次へ

[[戻る](#)] および [[次へ](#)] コマンドは、CodeWarrior ブラウザを使用しているときのみ使用可能です。詳細は「[戻ると次へ](#)」(p175) を参照してください。

CodeWarrior ブラウザについては「[ソースコードのブラウズ](#)」(p143) を参照してください。

エディタのコマンドを設定

CodeWarrior IDE では、作業スタイルに合わせてエディタのキーバインディングをカスタマイズできます。詳細は「[エディタ](#)」(p486) を参照してください。

オンラインリファレンス

プログラムの記述中に、特定のルーチン、変数、型などのドキュメントや定義を参照することができます。また CodeWarrior IDE の使い方のオンラインヘルプもあります。

CodeWarrior IDE のオンラインヘルプを利用するには、ヘルプファイルとビューアアプリケーションをインストールする必要があります。これらは CodeWarrior Tools CD に含まれています。ホストごとのインストール方法を紹介します。

CodeWarrior Reference CD を CD-ROM ドライブに入れてください。ダイアログが現れます。[Launch CodeWarrior Setup] ボタンをクリックして、画面上の指示に従ってください。

ここでは、オンラインドキュメントを見る方法とオンラインリファレンスデータベースのセットアップ方法を説明します。

ここでは、以下の項目を説明します。

[シンボル定義の検索](#)

[WinHelp](#)

シンボル定義の検索

プロジェクトのソースファイルにあるシンボルの定義を検索できます。シンボルが定義されていない場合、IDE その他設定パネルで選択したオンラインドキュメントのビューアを使って、シンボル定義を検索することができます。IDE は [WinHelp](#) のビューアをサポートしています。

ヒント： CodeWarrior ブラウザはシンボル定義を参照することもできます。[「ブラウザを使う」\(p173\)](#) を参照してください。

選択したシンボルの定義を検索するには、[検索メニュー](#)の [[定義を検索](#)] を選択します。CodeWarrior はすべてのプロジェクトファイルからシンボル定義を検索します。

CodeWarrior はプロジェクト内でシンボル定義（複数も可）を見つけると、自動的にエディタウィンドウを開いて各シンボルをハイライト表示します。定義が見つからなければ、システムビープ音を鳴らします。

ヒント： シンボル定義を見た後で、元のテキスト位置に戻るには、Shift + Ctrl + B を押します。このキーバインディングは [戻る] コマンドと等価です。詳細は [「戻ると次へ」\(p175\)](#) を参照してください。

WinHelp

CodeWarrior IDE は WinHelp ビューア用のヘルプファイルを提供しています。WinHelp は Windows オペレーティングシステムに含まれています。

WinHelp を使って Win32、MFC、ANSI ライブラリ、CodeWarrior IDE ドキュメントを検索できます。また IDE のツールを右ボタンクリックするとポップアップヘルプが表示されます。

WinHelp を使うには、ヘルプメニューの [CodeWarrior ヘルプ] コマンドを選択してください。IDE のヘルプファイルが表示されます。

IDE に関する情報は、WinHelp ドキュメントのハイパーテキストリンクをクリックしてください。タスクの実行については WinHelp ウィンドウの [How to] ボタンを選択してください。[Glossary] ボタンをクリックするとドキュメントで使われている用語の説明が表示されます。

第 6 章 テキストの検索と置換

この章では、ファイル中のテキストを検索、置換するための CodeWarrior IDE の使い方を説明します。

CodeWarrior IDE は、「検索」ダイアログを使って検索と置換の機能を提供します。プロジェクト内の 1 つのファイル、またはすべてのファイルで、またはどのような組み合わせのファイルでも、テキストを検索および置換できます。UNIX の `grep` コマンドで使われているような正規表現で検索することもできます。

以下の項目について説明します。

[検索ダイアログの解説](#)

[1 つのファイルでのテキスト検索と置換](#)

[複数のファイルでのテキスト検索と置換](#)

[選択したテキストを検索](#)

[正規表現 \(grep\) を使う](#)

検索ダイアログの解説

「[検索](#)」ダイアログ ([図 6.1 \(p116\)](#)) は、CodeWarrior IDE の高機能な検索ダイアログです。「検索」ダイアログを表示するには、[検索メニュー](#)の [[検索](#)] を選択します。このダイアログで、プロジェクト内の 1 つまたは複数のファイルでテキストを検索できます。テキスト文字列、テキストサブ文字列、パターン照合による検索、置換が可能です。

「検索」ダイアログには 2 つの欄があります。

[検索と置換](#)

[複数ファイル検索機能](#)

検索と置換

「[検索](#)」ダイアログ ([図 6.1 \(p116\)](#)) の検索および置換欄を簡単に説明します。ダイアログ内の項目を以下に示します。

[検索文字列](#)

[置換文字列](#)

[最近指定した文字列ポップアップメニュー](#)

[検索](#)

[置換](#)

[置換、次を検索](#)

[すべてを置換](#)

[バッチ](#)

[ラップ](#)

[大小文字の違いを無視](#)

[単語全体](#)

[正規表現](#)

[複数ファイル検索](#)

[複数ファイル検索](#)

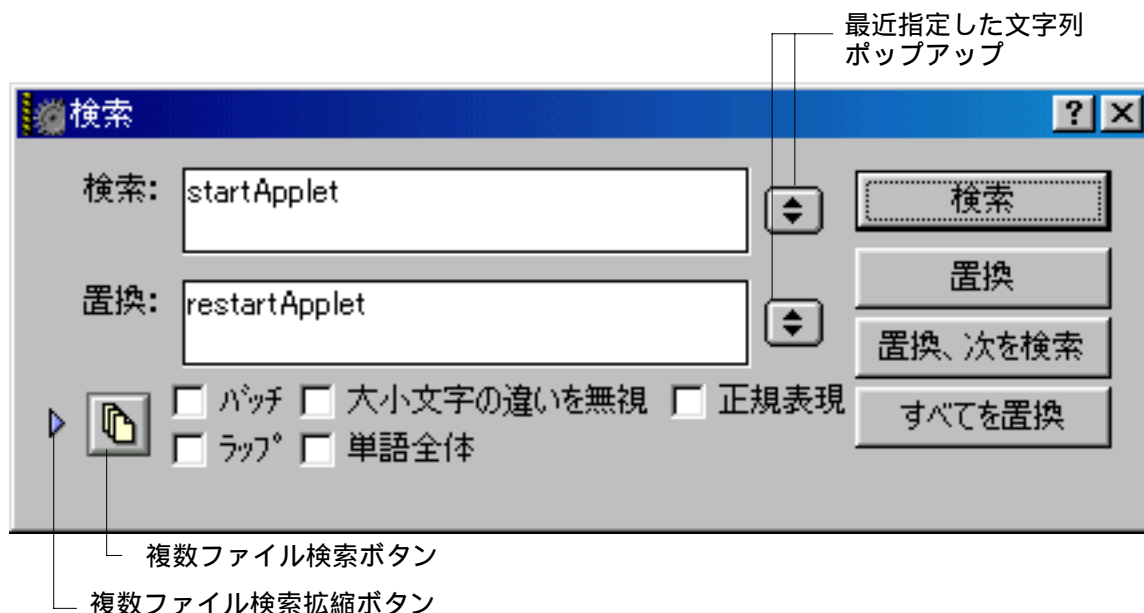
検索文字列

[検索] フィールド ([図 6.1](#)) に検索したいテキストを入力します。

[検索] フィールドで [[カット](#)] [[コピー](#)] [[貼り付け](#)] [[削除](#)] コマンドを使うことができます。これらのコマンドについては「[編集メニュー](#)」(p459) を参照してください。

「[選択部分を検索文字列へ](#)」(p462) (選択部分を検索文字列へ) では、「検索」ダイアログを使わずに [検索] フィールドにテキストを入力する方法を説明します。

図 6.1 「検索」ダイアログの検索と置換の欄



置換文字列

[置換] フィールド ([図 6.2](#)) に入力したテキストで、検索したテキストを置換します。

[置換] フィールドで [[カット](#)] [[コピー](#)] [[貼り付け](#)] [[削除](#)] コマンドを使うことができます。これらのコマンドについては「[編集メニュー](#)」(p459) を参照してください。

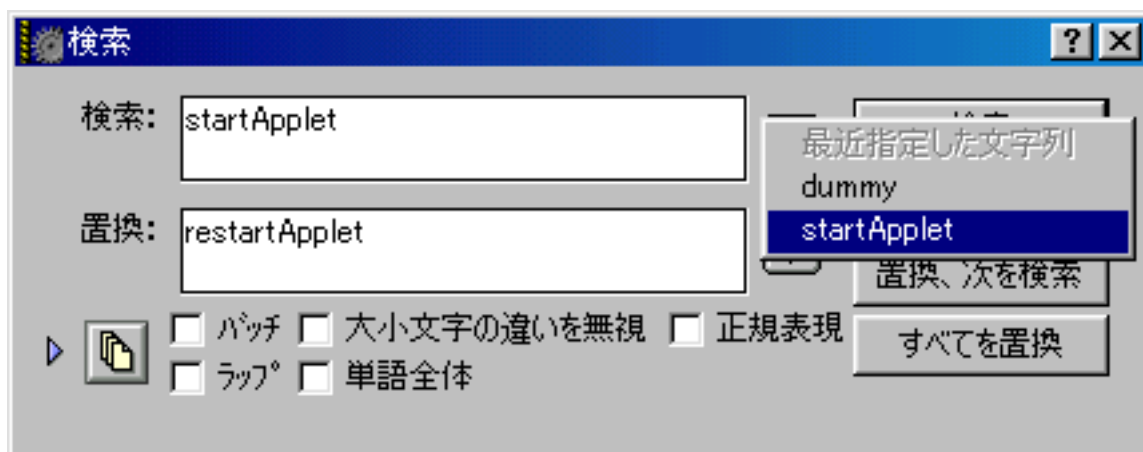
「[選択部分を置換文字列へ](#)」(p462) では、「検索」ダイアログを使わずに [置換] フィールドにテキストを入力する方法を示します。

最近指定した文字列ポップアップメニュー

[最近指定した文字列] ポップアップメニュー ([図 6.2](#)) には、最近検索に使われたテキストが表示されます。

このポップアップは、[置換] フィールドと [検索] フィールドの右に 1 つずつあります。ポップアップからテキストを選択すると、対応する編集可能テキストフィールドのテキストを置換します。

図 6.2 [最近指定した文字列] ポップアップメニュー



検索

[検索] ボタン ([図 6.1 \(p116\)](#)) は検索を開始するボタンです。「検索」ダイアログの他のコントロールを設定し、必要なフィールドへの入力を完了した後でこのボタンを押すと、テキスト検索が始まります。

テキスト検索についての詳細は、「[選択したテキストを検索](#)」(p123) を参照してください。

置換

[置換] ボタン ([図 6.1 \(p116\)](#)) は置換を実行するボタンです。

[検索] フィールドにテキストを入力して、[検索] ボタンをクリックすると、CodeWarrior IDE は他のコントロールの設定に従って一致するテキストを検索します。[検索] フィールドのテキストと一致するテキストが見つかったら、[置換] ボタンがクリック可能になり [置換] フィールドのテキストに置換することができます。

テキストの検索および置換についての詳細は、「[検索したテキストを置換](#)」(p127) を参照してください。

置換、次を検索

[置換、次を検索] ボタン ([図 6.1 \(p116\)](#)) は [置換] フィールドと同じようにテキスト置換を実行した後、さらに次の[検索](#)操作を行います。

テキストの検索および置換についての詳細は、[「1 つのファイルでのテキスト検索と置換」\(p125\)](#) および [「複数のファイルでのテキスト検索と置換」\(p130\)](#) を参照してください。

すべてを置換

[すべてを置換] ボタン ([図 6.1 \(p116\)](#)) は [置換] フィールドと同じ機能を実行しますが、適切なウィンドウで発見された [検索] フィールドと一致する「すべての」テキストを [置換] フィールドのテキストで置換します。

テキストの検索と置換についての詳細は [「検索したテキストを置換」\(p127\)](#) を参照してください。

バッチ

[バッチ] チェックボックス ([図 6.2 \(p117\)](#)) をオンにすると、[検索] コマンドの結果を「検索結果」ウィンドウ ([図 6.7 \(p130\)](#)) に表示します。

検索における [バッチ] チェックボックスの役割の詳細は [「バッチ検索」\(p129\)](#) を参照してください。

ラップ

[ラップ] チェックボックス ([図 6.2 \(p117\)](#)) をオンにすると、検索がファイルやグループの終わりに到達したときに反対方向から検索を続けます。

この機能の詳細は [「検索範囲をコントロール」\(p126\)](#) を参照してください。

大小文字の違いを無視

[大小文字の違いを無視] チェックボックス ([図 6.2 \(p117\)](#)) をオンにすると、検索時に [検索] フィールドに入力されたテキストの大文字と小文字の区別をしません。

この機能の詳細は [「検索パラメータをコントロール」\(p127\)](#) を参照してください。

単語全体

[単語全体] チェックボックス ([図 6.2 \(p117\)](#)) をオンにすると、テキストを検索するときに、単語の中に含まれる一致するテキスト部分を無視します。単語全体が一致するテキストだけを検索します。

この機能の詳細は [「検索パラメータをコントロール」\(p127\)](#) を参照してください。

正規表現

[正規表現] チェックボックス ([図 6.2 \(p117\)](#)) をオンにすると [検索] フィールドのテキストを正規表現として解釈します。

CodeWarrior の正規表現は、UNIX の grep 正規表現に似ています。この機能の詳細は、「[正規表現 \(grep\) を使う](#)」(p136) を参照してください。

複数ファイル検索

▶ [複数ファイル検索] 拡張ボタン ([図 6.2 \(p117\)](#)) をクリックすると[検索](#)ダイアログの [複数ファイル検索機能] 欄が表示されます ([図 6.3 \(p119\)](#))。

[検索](#)ダイアログを使った複数ファイルの検索の詳細は、「[複数のファイルでのテキスト検索と置換](#)」(p130) を参照してください。

複数ファイル検索

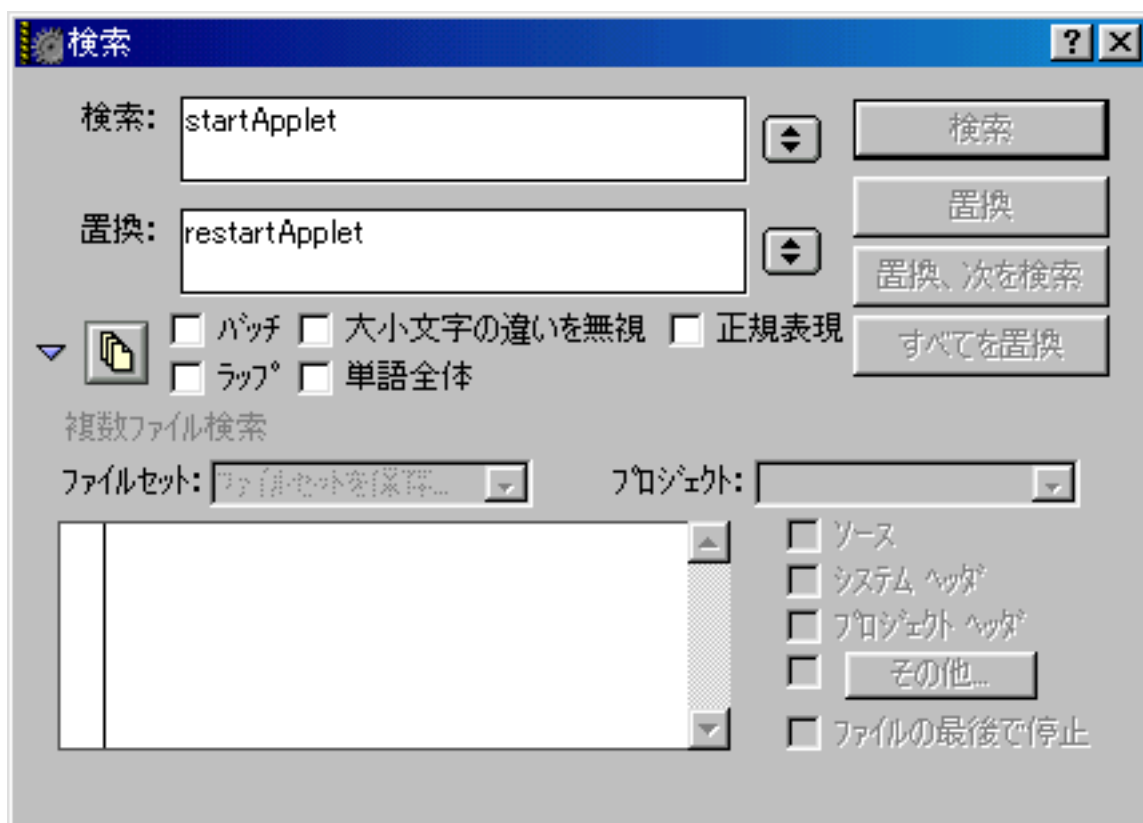


[複数ファイル検索] ボタン ([図 6.3](#)) をクリックすると、「[検索](#)」ダイアログの下の部分が検索のために使えるようになります ([図 6.4 \(p121\)](#))。

複数ファイル検索ボタンをクリックしない場合、[複数ファイル検索機能] 欄は[図 6.5](#) のように薄く表示されます。

複数ファイル検索ボタンの詳細は「[複数ファイルの検索を行う](#)」(p131) を参照してください。

図 6.3 複数ファイル検索ボタンをクリックしていないとき



複数ファイル検索機能

ここでは、「[検索](#)」ダイアログの[複数ファイル検索機能] 欄のユーザーインターフェース項目について説明します ([図 6.4](#))。このインターフェースには、以下が含まれます。

[ファイルリスト](#)

[ファイルセット](#)

[プロジェクト](#)

[ファイルの最後で停止](#)

[ソース](#)

[システムヘッダ](#)

[プロジェクトヘッダ](#)

[その他](#)

ファイルリスト

ファイルリスト ([図 6.4](#)) は複数の検索すべきファイルを表示します。[ソース] [システムヘッダ] [プロジェクトヘッダ] [その他] のオプションでファイルを追加できます。プロジェクトウィンドウからファイルをドラッグ&ドロップしても追加できます。

ファイルセットへのファイルの追加と削除については「[検索するファイルを選択](#)」(p131)を参照してください。

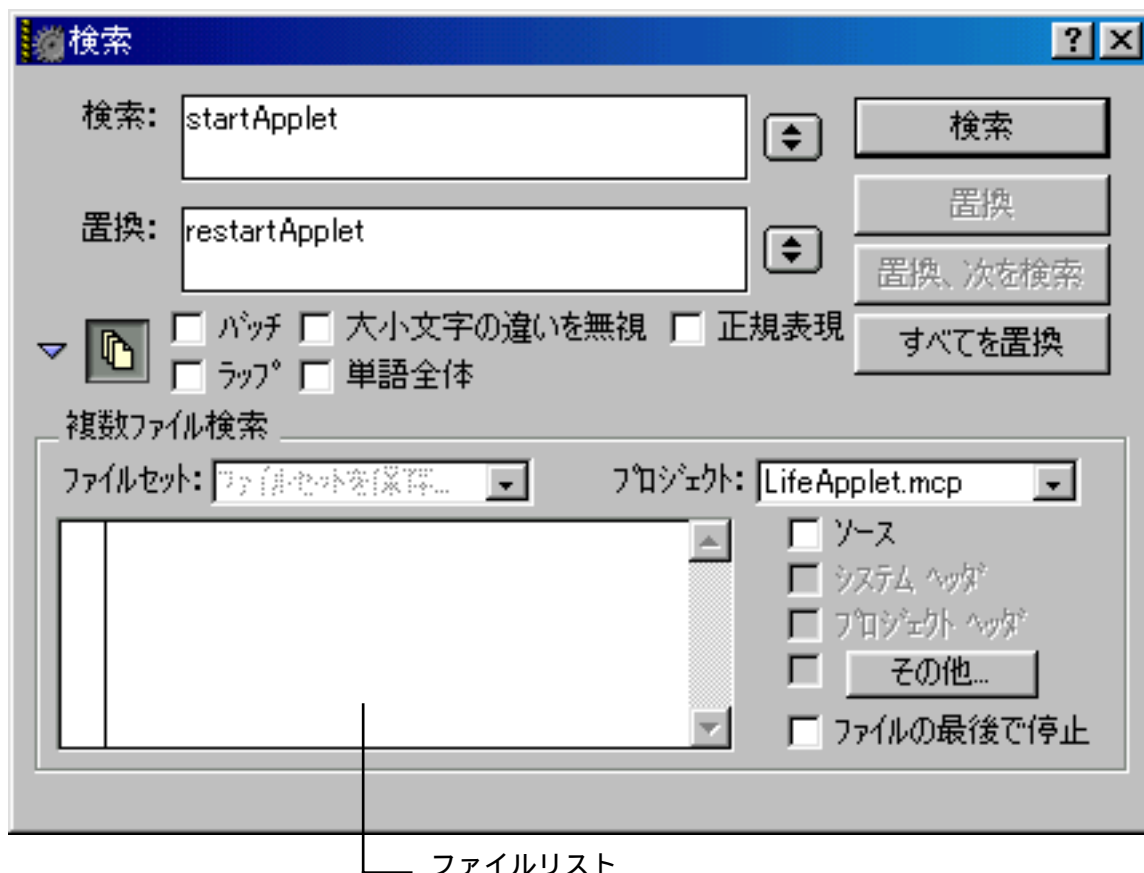
ファイルセット

[ファイルセット] ポップアップメニュー ([図 6.5](#)) は、複数ファイルの検索に使われます。このポップアップを選択すると、検索を行うファイルの選択や削除、またはセットの保存ができます。

ヘッダファイルの集合など、ファイルをまとめてテキスト検索するための関連するファイルのセットを作成できます。

複数ファイル検索のファイルのセットについては、「[検索するファイルを選択](#)」(p131)を参照してください。

図 6.4 「検索」ダイアログの複数ファイル検索欄



プロジェクト

[プロジェクト] ポップアップメニュー (図 6.6) で、検索を実行したいプロジェクトファイルを選択できます。CodeWarrior IDE は一度に複数のプロジェクトを開くことができるので、このメニューによって異なるプロジェクトで同一の検索を実行できます。

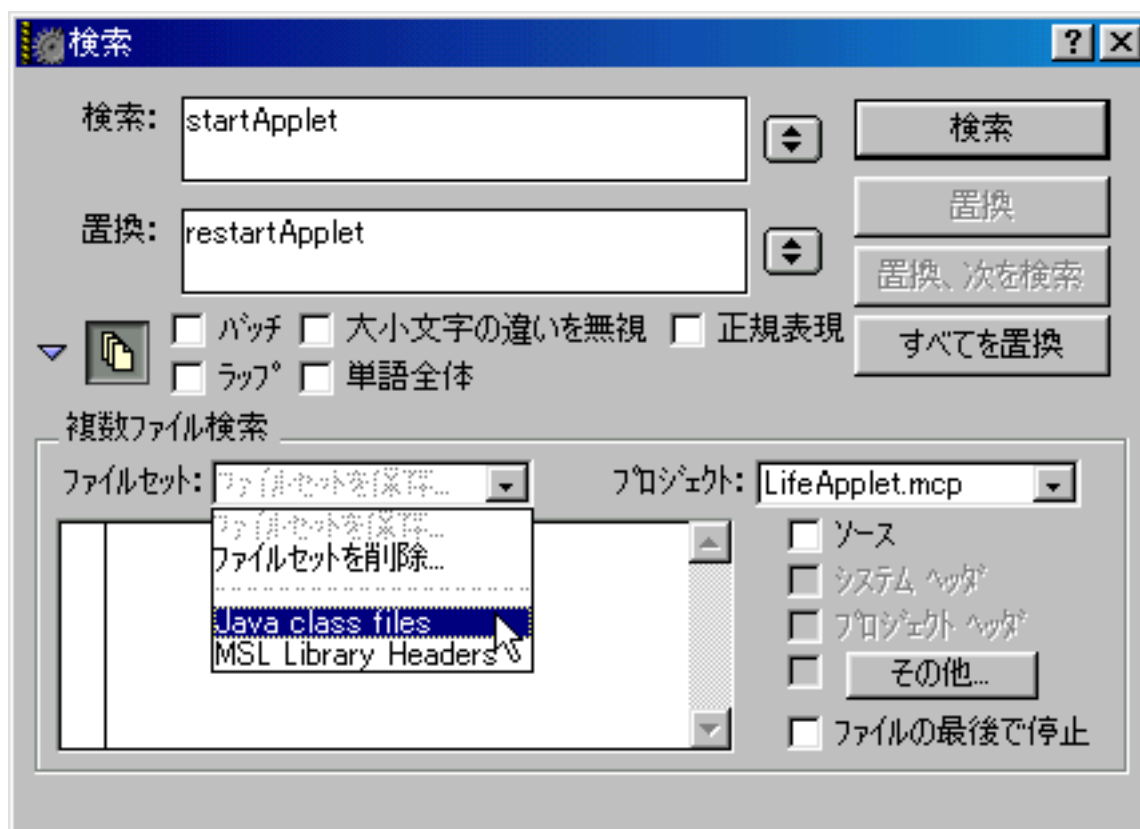
ファイルの最後で停止

[ファイルの最後で停止] チェックボックスをオフにすると、[ファイルリスト](#)にあるすべてのファイルが 1 つの大きいファイルであるかのように続けて検索されます。1 つのファイルの終りに達すると、次のファイルの検索が開始されます。リストの最後のファイルの終わりに達すると、ピープ音が鳴ります。

各ファイルを個別に検索するには、[ファイルの最後で停止] チェックボックスをオンにします。1 つのファイルの終りに達すると、ピープ音が鳴ります。検索を続けるには[検索メニュー](#)の [次のファイル内を検索] を選択します。

[ファイルの最後で停止] チェックボックスの詳細は「[検索範囲をコントロール](#)」(p126) を参照してください。

図 6.5 [ファイルセット] ポップアップメニュー



ソース

[ソース] チェックボックス ([図 6.4 \(p121\)](#)) は、現在のプロジェクトのすべてのソースファイルを [ファイルリスト](#) に追加します。

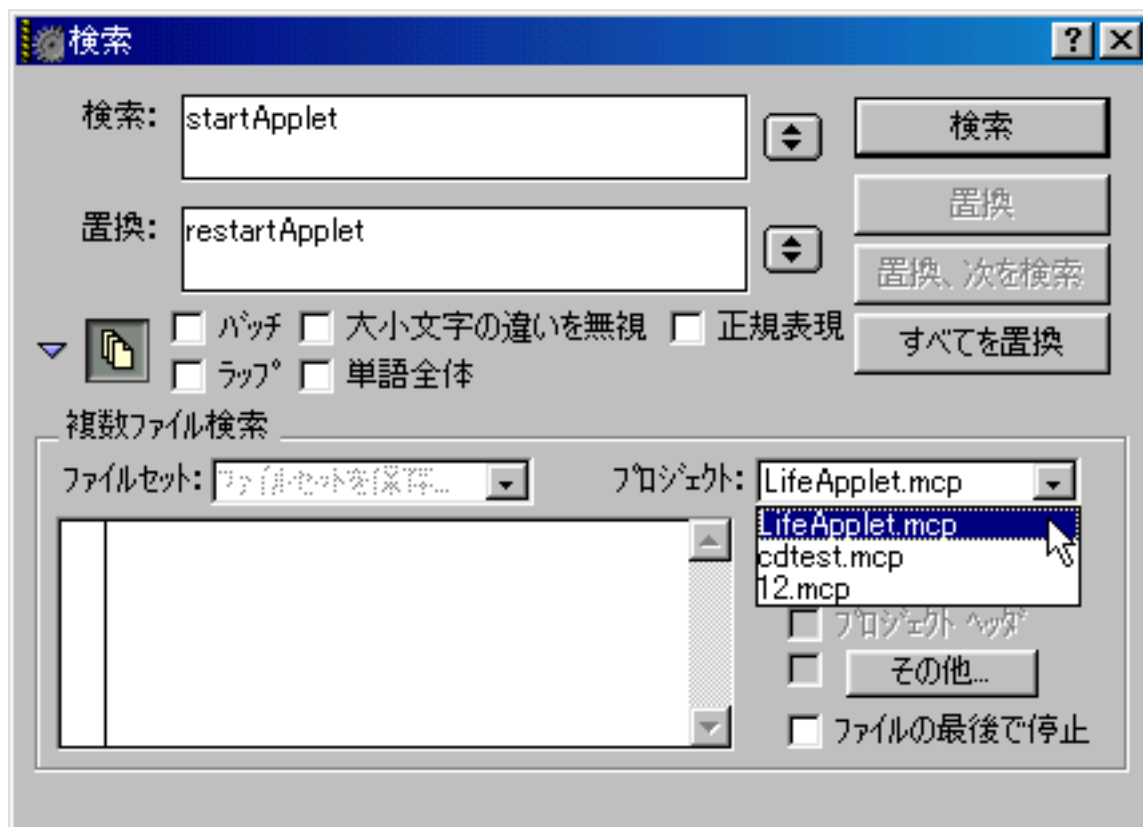
ファイルセット内のソースファイル、および複数ファイル検索でのその役割の詳細は [「プロジェクトソースファイルを追加」 \(p132\)](#) を参照してください。

システムヘッダ

[システムヘッダ] チェックボックス ([図 6.4 \(p121\)](#)) は、現在のプロジェクトのすべてのシステムヘッダファイルを [ファイルリスト](#) に追加します。

ファイルセット内のシステムヘッダファイルおよび複数ファイル検索でのその役割の詳細は、[「システムヘッダファイルを追加」 \(p132\)](#) を参照してください。

図 6.6 [プロジェクト] ポップアップメニュー



プロジェクトヘッダ

[プロジェクトヘッダ] チェックボックス ([図 6.4 \(p121\)](#)) は、現在のプロジェクトのすべてのヘッダファイルを [ファイルリスト](#) に追加します。

ファイルセット内のプロジェクトヘッダ、および複数ファイル検索でのその役割の詳細は、[「プロジェクトヘッダファイルを追加」 \(p132\)](#) を参照してください。

その他

[その他] ボタン ([図 6.4 \(p121\)](#)) とチェックボックスにより、[ファイルリスト](#) にファイルを追加できます。

ファイルセットへのファイルの追加の詳細は、[「任意のファイルを追加、削除」 \(p132\)](#) を参照してください。

選択したテキストを検索

CodeWarrior エディタには、「検索」ダイアログを使わずにテキストを検索する方法が 2 つあります。これらは両方とも、ウィンドウ内でテキストを選択することにより、「検索」ダイアログを表示させずに検索を行います。

テキストを検索するとき、「[検索](#)」ダイアログで最後に選択したオプション設定が使われます。これらのオプション設定を変更するには、「[検索](#)」ダイアログを使います。

この節をお読みになる前に、エディタウィンドウのテキストの選択方法を理解しておく必要があります。テキストの選択方法は「[テキストを選択](#)」(p106)を参照してください。

アクティブなエディタウィンドウでテキストを検索

作業中のエディタウィンドウでテキストの別の出現箇所を検索するには、この方法が有効です。

最初に検索したいテキストを選択します。その後、[検索メニュー](#)の[[選択部分を検索](#)]を選択します。

CodeWarrior IDE は、現在のファイル内でテキストの次の出現箇所を探します。

順方向（ファイルの終わりへ向かう）へ次のテキストを検索するには、[[検索](#)]ボタンをクリックするか、または[検索メニュー](#)の[[次を検索](#)]を選択します。またメニューコマンドの横にあるキーボードショートカットも使うことができます。

逆方向（ファイルの先頭へ向かう）へ前のテキストを検索するには[[前を検索](#)]を選択します。

このコマンドのキーボードショートカットは Shift + F3 キーです。

CodeWarrior IDE は、1 つ前のテキストの出現箇所を検索して、そのテキストを選択表示します。テキストが見つからないと、ビープ音が鳴ります。

さらに検索を行いたい場合は、続けて[検索メニュー](#)の[[検索](#)] [[次を検索](#)] [[前を検索](#)] のいずれかのコマンドを使います。

他のウィンドウのテキストを検索

この方法は、複数のファイルで同じテキストを検索するときに有効です。

最初に検索したいテキストを選択します。次に[検索メニュー](#)の[[選択部分を検索文字列へ](#)]を選択します。エディタは、そのテキストを「検索」ダイアログの[検索]フィールドに入力します。

ここで検索したいウィンドウをアクティブにします。次に[検索メニュー](#)の[[次を検索](#)]を選択して順方向へ検索を始めます。

テキストを逆方向に検索したい場合、[[前を検索](#)] コマンドを選択してください。

このコマンドのキーボードショートカットは Shift + F3 です。

CodeWarrior IDEはアクティブなエディタウィンドウ上のテキスト挿入位置から検索を始めます。

順方向へテキストを検索するには[[検索](#)]ボタンをクリックするか、[検索メニュー](#)の[[次を検索](#)]を選択してください。メニューコマンドの横のキーボードショートカットも使用できます。

逆方向へテキストを検索するには [\[前を検索 \]](#) を選択してください。

[検索] フィールドのテキストの次の出現箇所を検索するには、続けて[検索メニュー](#)の [\[検索 \]](#) [\[次を検索 \]](#) [\[前を検索 \]](#) コマンドを使います。

1 つのファイルでのテキスト検索と置換

「[検索](#)」ダイアログを使って、アクティブなエディタウィンドウ内でテキストを検索できます。テキストが見つかったとき、これを変更するか、または次の出現箇所を続けて検索できます。

ここでは「検索」ダイアログを使って、アクティブなエディタウィンドウで特定のテキストを検索して置換する方法を説明します。

ウィンドウを開いたことがない方は [「既存ファイルを開く」\(p77\)](#) を参照してください。

ファイルを作成したことがない方は [「新規ファイルを作成」\(p77\)](#) を参照してください。

ここでは、以下の項目を説明します。

[テキストを検索](#)

[検索範囲をコントロール](#)

[検索パラメータをコントロール](#)

[特殊文字を検索](#)

[検索したテキストを置換](#)

[バッチ検索](#)

テキストを検索

[検索] フィールドにテキストを入力するには、検索メニューの [検索] を使って「[検索](#)」ダイアログを表示します。このダイアログの [検索] フィールドにテキストを入力するか、または [最近指定した文字列] ポップアップメニューでテキストを選択します ([図 6.2 \(p117\)](#))。

注意： 改行またはタブ文字を含む検索テキストを入力する方法は、[「特殊文字を検索」\(p127\)](#) を参照してください。

検索する前に、検索範囲をコントロールするオプションを設定します。

検索範囲は、ファイル全体を検索するのか、テキスト挿入位置から一方向のみを検索するのかを決めます。検索の範囲を設定する方法は [「検索範囲をコントロール」\(p126\)](#) を参照してください。

検索パラメータは、大文字小文字を無視して検索するか、単語の一部分のテキストも検索するかなどを設定します。詳細は「[検索パラメータをコントロール](#)」(p127) を参照してください。

検索を始める前に、複数ファイルの検索はオフになっていることを確認してください。ここでは、アクティブなエディタウィンドウ内のファイルのみの検索を考えています。複数ファイルの検索を行う方法は「[複数ファイルの検索を行う](#)」(p131) を参照してください。

「[検索](#)」ダイアログの「[検索](#)」ボタンをクリックする、または[検索メニュー](#)の「[検索](#)」または「[次を検索](#)」を選択すると、テキスト挿入位置からファイルの終わりに向かって検索を開始します。

テキスト挿入位置からファイルの先頭に向かって検索したい場合は、[検索メニュー](#)の「[前を検索](#)」を選択します。

テキスト挿入位置からファイルの先頭に向かって検索したい場合は、Shift + F3 キーを押します。これは「[前を検索](#)」コマンドと同じ働きをします。

ファイルの終わりに向かって「[検索](#)」フィールドのテキストの次の出現箇所を検索するには、「[検索](#)」ボタンをクリックするか、[検索メニュー](#)の「[次を検索](#)」を選択します。

ファイルの先頭に向かって「[検索](#)」フィールドのテキストの出現箇所を検索するには、「[前を検索](#)」を選択します

エディタは「[検索](#)」フィールドのテキストを発見した場合、それを選択します。テキストが見つからないとピープ音が鳴ります。

「[検索](#)」フィールドのテキストの次の出現箇所を検索するには、続けて「[検索](#)」、「[次を検索](#)」、「[前を検索](#)」コマンドを使います。

ここで検索したテキストのすべて、または一部を新しいテキストで置換できます。

テキストを置換するには「[検索したテキストを置換](#)」(p127) を参照してください。

検索範囲をコントロール

「[検索](#)」ダイアログの「[ラップ](#)」チェックボックスは、検索がファイルの終わりまたは先頭に達した後の動作をコントロールします。

例えば、アクティブなエディタウィンドウのテキストファイルの中ほどに、テキスト挿入位置があり、「[ラップ](#)」チェックボックスがオンであると仮定します。[検索メニュー](#)の「[次を検索](#)」を選択すると、CodeWarrior IDE はファイルの終わりに向けて検索を始めます。ファイルの終点に達すると、ファイルの先頭から挿入位置まで検索を続けます。

「[前を検索](#)」コマンドも、検索がファイルの先頭に達したときに同様の動作をします。検索がファイルの先頭で折り返します。

「[ラップ](#)」チェックボックスがオフのときに[検索メニュー](#)の「[次を検索](#)」を選択すると、検索はファイルの終点で止まります。

「[ラップ](#)」チェックボックスがオンのときに複数のファイルを検索すると、最後のファイルの終点まで検索を終えた後、最初のファイルから検索を行います。

検索パラメータをコントロール

検索するテキストの一致に関して、2 つのオプション([単語] チェックボックスと [単語] チェックボックス) があります。

大小文字無視

[大小文字無視] チェックボックス ([図 6.2 \(p117\)](#)) をオンにすると、検索時に [検索] フィールドに入力されたテキストの大文字と小文字の区別をしません。オフの場合、大文字と小文字を区別します。

例えば、オンにして [検索] フィールドに「Foobar」と入力して検索すると、foobar または FOOBAR など、どんな大文字小文字の組み合わせでも検索されます。

単語

[単語] チェックボックス ([図 6.2 \(p117\)](#)) をオンにすると、CodeWarrior IDE は [検索] フィールドのテキストに完全に一致する単語(句読点かスペースで区切られている単語) だけを検索します。オフの場合、単語に含まれるテキストも検索します。例えば [検索] フィールドに「Word」と入力したとします。オンの場合、ファイル内の Word が検索されます。オフの場合、Word や、Words、WordCount、BigWordCount などの単語も検索されます。

特殊文字を検索

以下の方法でタブ文字、または改行文字を [検索] [置換] フィールドに入力できます。

[検索] [置換] フィールドにタブ文字、または改行文字を含むテキストをコピー & ペーストする

ドラッグ&ドロップがサポートされている場合、エディタウィンドウからテキストを直接フィールドヘドラッグする

[正規表現] オプションをオンにして、フィールドに \t (タブ) \r (改行) を入力する

警告！ [正規表現] については「[正規表現 \(grep \) を使う \(p136\)](#)」を参照してください。

検索したテキストを置換

テキストの出現箇所を検索したとき、一度に 1 つずつ置換するか、または一度にファイル全体の出現箇所を置換することができます。

選択して置換

テキストを選んで置換するには、最初に検索するテキストを入力して、検索するために、[検索メニュー](#)の [検索] を選択するか、または「検索」ダイアログの [検索] ボタンをクリックします。テキストの検索については、「[テキストを検索 \(p125\)](#)」を参照してください。

次に、置換するテキストを「検索」ダイアログの [置換] フィールドに入力します。

または[置換]フィールドの右の矢印アイコンをクリックして[最近指定した文字列]ポップアップメニューからテキストを選択します。このメニュー ([図 6.2 \(p117\)](#)) には、最近使ったテキストが保存されています。

次は見つけたテキストを置換するかどうかの選択です。このための「検索」ダイアログのボタンは 3 つあります。[置換] ボタン、[置換後 再検索] ボタン、および [すべて置換] ボタンです。各ボタンは異なる操作を実行します。

テキストを置換して結果を見るには、「検索」ダイアログの [置換] ボタンをクリックするか、または[検索メニュー](#)の [[置換](#)] を選択します。エディタは、検索されたテキストを [置換] フィールドのテキストで置換します。

置換せずに検索を続けるには、[検索メニュー](#)の [[次を検索](#)] を選択するか、または「検索」ダイアログの [検索] ボタンをクリックします。

ファイルの先頭に向かって検索を続けるならば [[前を検索](#)] を選択するか、または Shift キーを押しながら「検索」ダイアログの [検索] ボタンをクリックします。

テキストを置換してから次の出現箇所を検索するには、[検索メニュー](#)の [[置換後次を検索](#)] および [[次を検索](#)] を選択するか、「検索」ダイアログの [置換後 再検索] ボタンをクリックします。エディタは選択されたテキストを [置換] フィールドのテキストで置換し [検索] フィールドのテキストの次の出現箇所を検索します。次の出現箇所が見つからないと、ピープ音が鳴ります。

[検索文字列](#) テキストを置換して、前の出現箇所を検索するには、[[置換後逆方向検索](#)] コマンドを使います。

キーボードショートカット、Ctrl + Shift + L を使います。または [[置換、次を検索](#)] をクリックします。

選択されたテキストが[置換](#)テキストで置換され、[検索文字列](#)テキストの前の出現箇所が検索されます。次の出現箇所が見つからないと、ピープ音が鳴ります。

すべて置換

テキストを置換するには、最初に検索するテキストを [検索] フィールドに入力して[検索メニュー](#)の [[検索](#)] を選択するか、または「検索」ダイアログの [検索] ボタンをクリックします。テキストの検索については「[テキストを検索](#)」(p125) を参照してください。

次に、置換するテキストを「検索」ダイアログの [置換] フィールドに入力します。

[検索] フィールドのテキストの出現箇所をすべて置換するには、「検索」ダイアログの [すべて置換] ボタンをクリックするか、または[検索メニュー](#)の [[すべて置換](#)] を選択します。

警告! [[すべて置換](#)] コマンドを使うときは注意してください。この操作は取り消しできません。

ヒント： 1 つのソースファイルで [すべて置換] コマンドを使う場合、実行する前に必ずソースファイルを保存してください。取り消しはできませんが、[復帰] コマンドで最後に保存した状態に復帰することが可能です。これは複数ファイルではできません。

バッチ検索

CodeWarrior IDE は、検索テキストに一致したすべてのテキストを 1 つのウィンドウに集めて、簡単に参照できる方法を提供しています。

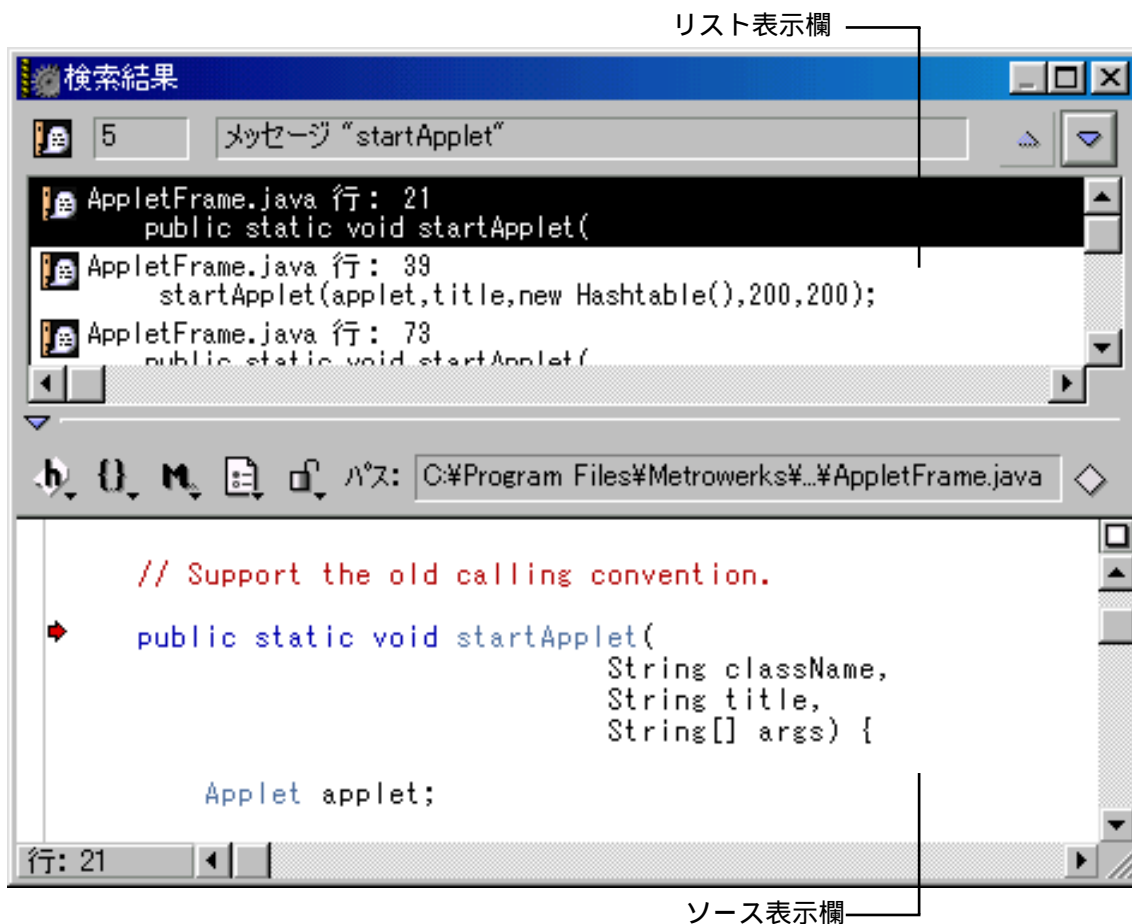
「検索」ダイアログで [バッチ] チェックボックスをオンにして、[検索] ボタンをクリックすると、[検索] フィールドのテキストのすべての出現箇所が検索され「検索結果」ウィンドウにその結果がリストされます ([図 6.7](#))。

「検索結果」ウィンドウ ([図 6.7](#)) には、リスト表示欄とソース表示欄があります。

[検索] フィールドのテキストの出現箇所がリスト表示欄に表示されます。特定の箇所を表示するには、その該当項目をダブルクリックすると、ソース表示欄にソースが表示されます。

このウィンドウの機能についての詳細は「[メッセージウィンドウの解説](#)」(p269) を参照してください。

図 6.7 バッチ探索の結果



複数のファイルでのテキスト検索と置換

CodeWarrior IDE は、テキストの出現箇所を複数ファイルで検索できます。

ここでは、複数ファイルの検索方法を説明します。

複数ファイルの検索にはもう 1 つ簡単な方法があります。それは、ブラウザの検索メニューの[戻る]および[次へ]コマンドです。これらのコマンドの使い方は「[戻ると次へ](#)」(p175)を参照してください。

ここでは、以下の項目を説明します。

[複数ファイルの検索を行う](#)

[検索するファイルを選択](#)

[ファイルセットを保存](#)

[ファイルセットを削除](#)

[複数ファイルでの検索範囲をコントロール](#)

複数ファイルの検索を行う

複数のファイルを検索するには、「検索」ダイアログで複数ファイル検索機能をアクティブにします。



[複数ファイル検索](#)をオンにすると、ボタンが押されたように表示されます。

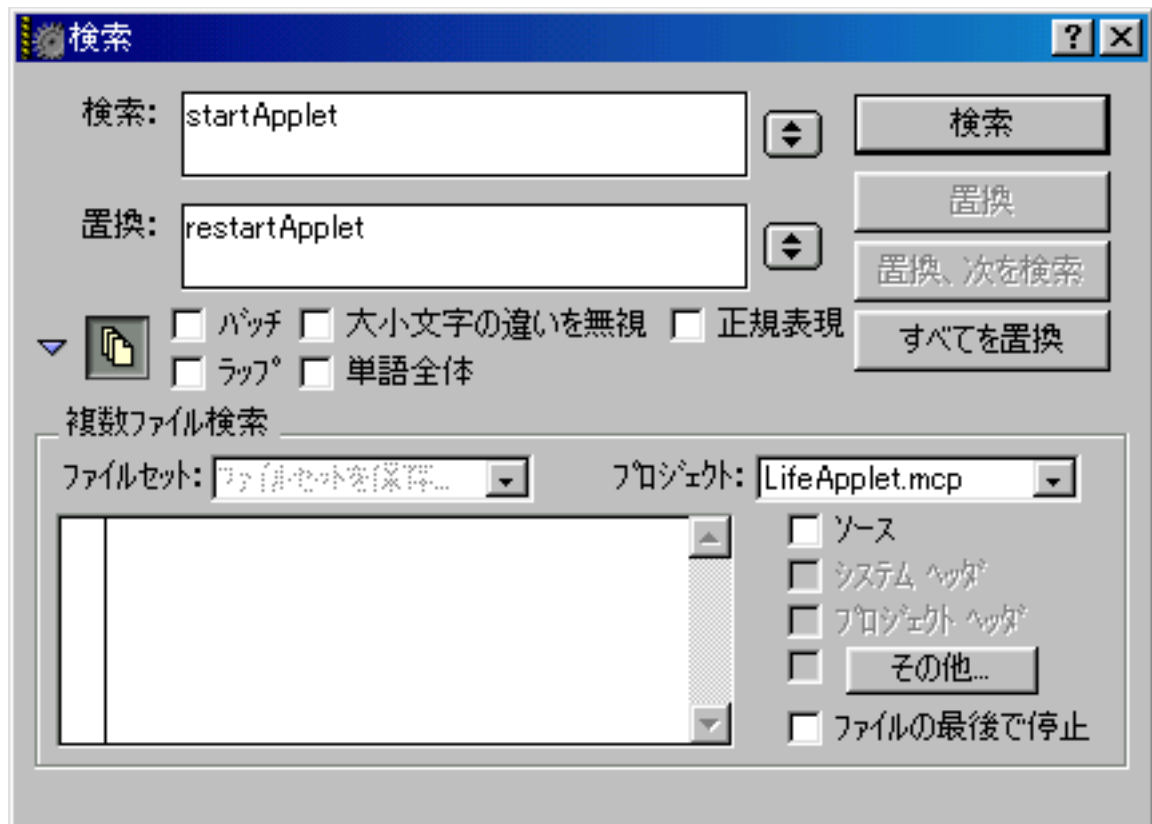


[複数ファイル検索](#)をオフにすると、ボタンは押されていないように表示されます。

[複数ファイル検索](#)の左の[複数ファイル検索](#)をクリックすると、ボタンのアイコンが変わります ([図 6.1 \(p116\)](#))。

「検索」ダイアログに有効な [複数ファイル検索機能] 欄が表示されます ([図 6.8 \(p131\)](#))。

図 6.8 [複数ファイル検索機能] 欄が有効な「検索」ダイアログ



検索するファイルを選択

検索ファイルを選択するにはいくつかの方法があります。

プロジェクトソースファイルを追加

現在のプロジェクトからすべてのソースファイルを追加するには、[ソース] チェックボックスをオンにします。オフにすると、関連するファイルがすべてファイルリストから削除されます。

ファイルの一部だけを含めるには、[ソース] チェックボックスをオンにして、不要なファイルを選択してファイル名をクリックした後 Backspace キーを押すと削除されます。

このオプションをオンにしてもファイルが追加されない場合は、プロジェクトのヘッダファイルの内部リストを[メイク] コマンドを使って更新します。[「プロジェクトをメイク」\(p259\)](#) を参照してください。

プロジェクトヘッダファイルを追加

現在のプロジェクトのプロジェクトヘッダファイルをすべて追加するには [プロジェクトヘッダ] チェックボックスをオンにします。オフにすると、関連するファイルがすべてファイルリストから削除されます。

ファイルの一部だけを含めるには、[プロジェクトヘッダ] チェックボックスをオンにして、不要なファイルを選択して Backspace キーを押すと削除されます。

このオプションをオンにしてもファイルが追加されない場合は、プロジェクトのヘッダファイルの内部リストを[メイク] コマンドを使って更新します。[「プロジェクトをメイク」\(p259\)](#) を参照してください。

システムヘッダファイルを追加

現在のプロジェクトのシステムヘッダファイルをすべて追加するには [システムヘッダ] チェックボックスをオンにします。[システムヘッダ] チェックボックスをオフにすると、関連するファイルがすべてファイルリストから削除されます。

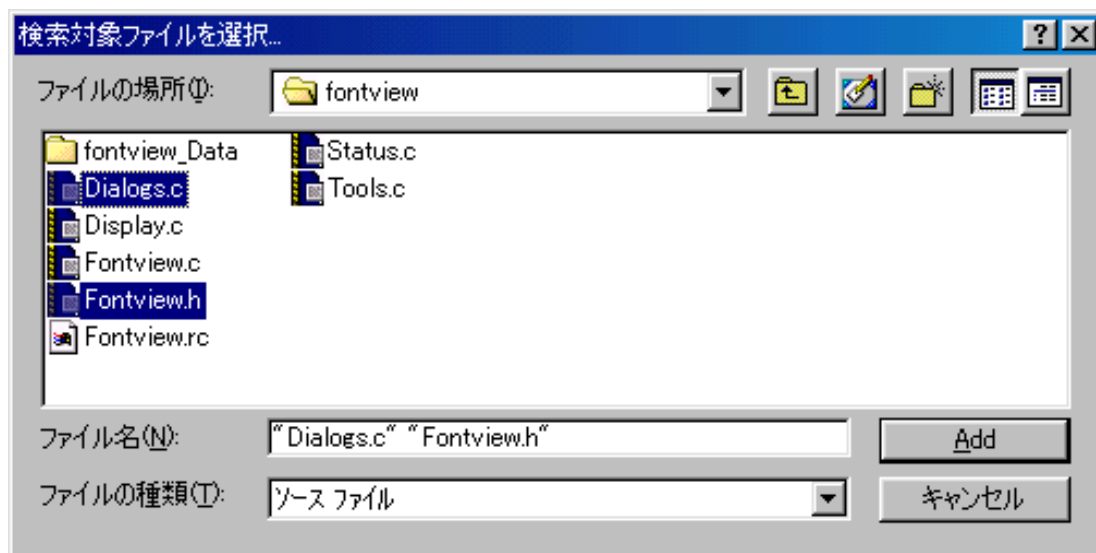
ファイルの一部だけを含めるには、[システムヘッダ] チェックボックスをオンにして、不要なファイルを選択して Backspace キーを押すと削除されます。

このオプションをオンにしてもファイルが追加されない場合は、プロジェクトのヘッダファイルの内部リストを[メイク] コマンドを使って更新します。[「プロジェクトをメイク」\(p259\)](#) を参照してください。

任意のファイルを追加、削除

複数ファイルの検索では、手動でファイルをファイルリストに追加、または削除できます。この方法は、現在のプロジェクトに含まれていないファイルの検索に有効です。

図 6.9 「検索対象ファイルを選択」ダイアログ



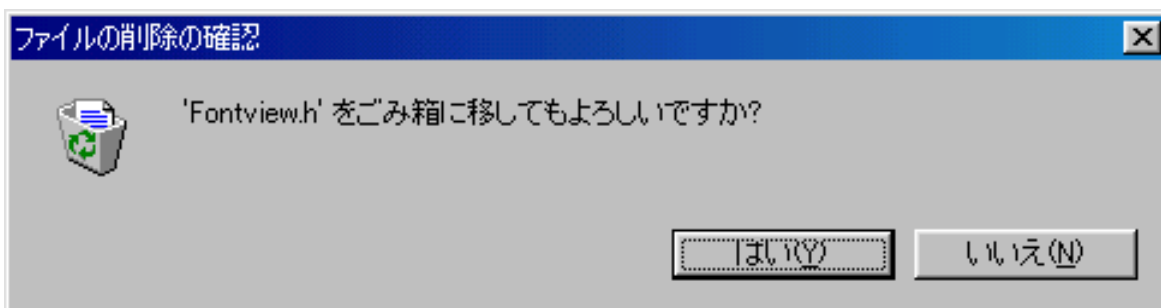
最初に、「検索」ダイアログの[[複数ファイル検索機能](#)]の[その他]ボタン をクリックします。[検索するファイルの選択]ダイアログ ([図 6.9](#)) が現れます。

「検索するファイルの選択」ダイアログ ([図 6.9](#)) は、ファイルセットに追加されるカレントディレクトリにあるファイルを表示します。ファイルセットにファイルを追加するには、選択してから [Add] ボタンをクリックします。[キャンセル] ボタンをクリックすると変更は行われません。

Ctrl+ クリックでこのリストから複数のファイルを選択することができます。ファイルの選択が終わったら、[Add]ボタンをクリックします。検索するファイルがすべてファイルセットに表示されます。セットからファイルを削除するには、「検索」ダイアログ([図 6.8\(p131\)](#))でそれらを選択してから Backspace キーを押します。

警告！ 「検索対象ファイルを選択」ダイアログ ([図 6.9](#)) でファイルのアイコンを選択して Delete キーを押すと、「ファイルの削除の確認」ダイアログが現れます ([図 6.10](#))。これはハードディスクからファイルを削除しようとしていることを警告するダイアログです。ファイルアイコンではなく、[ファイル名] フィールドでファイル名 (クォートマークを含む) を選択して Backspace キーを押してください。これが「検索対象ファイルを選択」ダイアログからファイルを消去する正しい方法です。

図 6.10 「ファイルの削除の確認」ダイアログ



この後で更にファイルを追加するには、「検索するファイルの選択」ダイアログの[その他] ボタンをクリックして上記の手順を繰り返します。

ファイルセットを選択

以前に保存されたファイルセットを選択して検索に含めるには、[ファイルセット](#)をクリックしてメニューからファイルセットを選択します ([図 6.5 \(p122\)](#))。ファイルセットリストにファイルが表示されます。

ファイルセットを保存

後で複数ファイルの検索に使うために、ファイルセットを保存するには [ファイルセット] ポップアップメニューの [ファイルセットを保存] を選択します。 [図 6.11 \(p135\)](#) に示す「ファイルセットを保存」ダイアログが表示されます。

[ファイルセットを保存] フィールドにファイルセットの名前を入力してください。

このファイルセットを使うことのできるプロジェクトを選択するために、ダイアログの[すべてのプロジェクトから利用可能] または [現在のプロジェクトに限定] ラジオボタンのどちらかを選択します。

このファイルセットを現在のプロジェクト内だけで使う予定ならば、[現在のプロジェクトに限定] を選択します。CodeWarrior IDEは、ファイルセットをプロジェクト内に保存します。

このファイルセットを他のプロジェクトでも使う予定ならば、すべてのプロジェクト用として [すべてのプロジェクトから利用可能] を選択します。CodeWarrior IDE は、ファイルセットを環境設定ファイル内に保存して、すべてのプロジェクトがこれを使えるようにします (現在のプロジェクトももちろん使えます)。

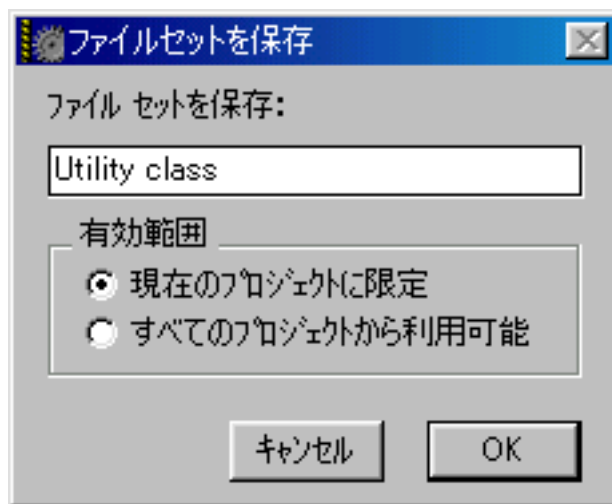
選択した後、ファイルセットに名前を付けて [OK] ボタンをクリックします。ファイルセットの保存を取り止めたい場合は [キャンセル] をクリックします。

ファイルセットを削除

以前に保存したファイルセットを削除するには、「検索」ダイアログの [[ファイルセット](#)] ポップアップメニューから [ファイルセットを削除] コマンドを選択します。 [図 6.12](#) に示すダイアログが表示されます。

削除したいファイルセットを選択して、[削除] ボタンをクリックします。選択したファイルセットが削除され、以後の検索では使えなくなります。ファイルセットの削除を終えたら、[終了] ボタンをクリックして「検索」ダイアログに戻ります。削除を中止する場合は、[キャンセル] をクリックします。

図 6.11 「ファイルセットを保存」ダイアログ



複数ファイルでの検索範囲をコントロール

CodeWarriorエディタでは、テキストの検索に任意の数のファイルを含めることができます。ファイルは現在のプロジェクト内のファイルでも、ディスク上のどのテキストファイルでもかまいません。特定のファイルの集合で頻繁に作業を行うならば、この集合をセットとして保存して後で復元して使うことができます。

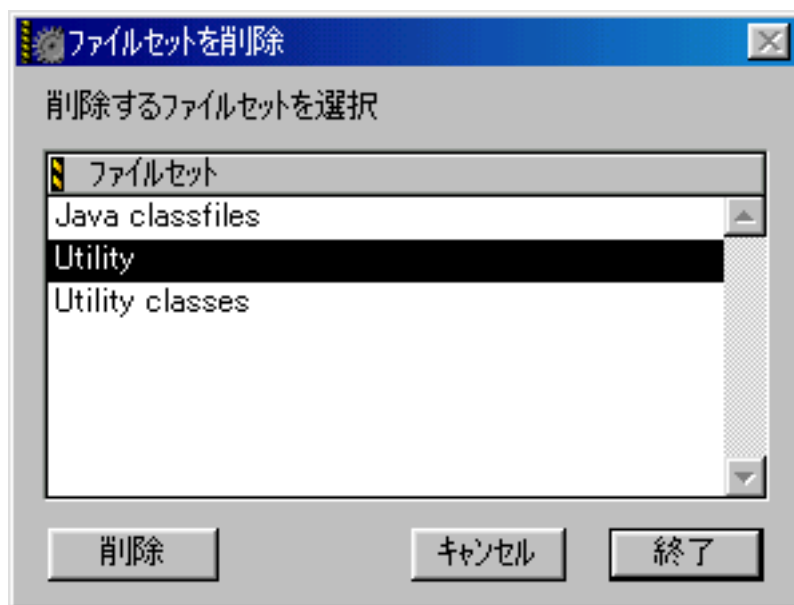
各ファイルの終わりで検索を停止するか、停止せずにすべてのファイルを検索するのかを指定します。

ファイルセット内のすべてのファイルを 1 つの大きなファイルとして扱うには、[ファイルの最後で停止] チェックボックス をオフにします。1 つのファイルの終わりに達すると、選択したテキストが見つかるまで、次のファイルの検索が開始されます。リストの最後のファイルの終わりに達すると、ピープ音が鳴ります。テキストが見つかると、[[検索](#)] [[次を検索](#)] [[前を検索](#)] コマンドを使って、次の出現箇所の検索を続行できます。

各ファイルを個別に検索するには [ファイルの最後で停止] チェックボックス をオンにします。

1 つのファイルの終りに達すると、ピープ音が鳴ります。ファイルセットの左の矢印が、現在検索中のファイルを示します。

図 6.12 「ファイルセットを削除」ダイアログ



検索を続けるには、[検索メニュー](#)の[[次のファイル内を検索](#)]または[[前のファイル内を検索](#)]を選択します。特定のファイルから検索を始めるには、ファイルを選択してその左の欄をクリックします。

[前のファイル内を検索] コマンドのキーボードショートカットは Ctrl + Shift + T です。
オプションを選択した後、1 つのファイルの検索と同じように、検索を始められます。

ヒント：[前のファイル内を検索] コマンドで、ファイルリストにある直前に検索したファイルに戻ることができます。

テキスト検索の詳細は、[「選択したテキストを検索」\(p123\)](#) または [「複数のファイルでのテキスト検索と置換」\(p130\)](#) を参照してください。

正規表現 (grep) を使う

正規表現とは、ファイル内のテキストを比較する際にマスクとして使うテキストの部分テキストです。1 文字、または複雑な文字列を正規表現として使用できます。正規表現をファイル内のテキストと比較するとき、入力した正規表現にファイル内のテキストが一致するかどうか比較します。

ここでは、CodeWarrior IDE が認識する正規表現と、テキストの検索と置換でどのように使うかを説明します。CodeWarrior の正規表現は、UNIX の grep コマンドが使う正規表現に似ています。

注意：「検索」ダイアログで [正規表現] チェックボックスをオンにします。

以下の内容について説明します。

[特殊なオペレータ](#)

[正規表現を使う](#)

特殊なオペレータ

[表 6.1](#) のキャラクタは正規表現のどこに置かれるかによって特別な意味を持ちます。詳細は「[正規表現を使う](#)」(p138) を参照してください。

表 6.1 メタキャラクタの内容

メタキャラクタ	内容
"."	あらゆる文字と一致するオペレータ：印刷可、または印刷不可の文字と一致する。newline と null は除く。
"*"	ゼロ以上と一致するオペレータ：パターンに一致するまで最小の通常表現式を無制限に繰り返す（ゼロを含む）。
"+"	1 以上と一致するオペレータ：パターンに一致するまで最小の通常表現式を繰り返す。
"?"	ゼロか 1 に一致するオペレータ：最小の通常表現式を最低 1 回行う、または全く行わない。
"\n"	後方参照オペレータ：検索文字列にある指定したグループを参照するために置換文字列で使用される。各グループは括弧で括られる。 <i>n</i> は 1 から 9 まで。数は正規表現の左から特定のグループを指定する。
" "	代替オペレータ：選択した通常表現式に一致する。2 つの通常表現式の間にこのオペレータを置くと、一致する文字列の最大共有体と一致する。
"^"	行頭と一致するオペレータ：文字列の最初か、改行文字の後の文字列に一致する。"^" の後ろにある場合、動作しない。
"\$"	行末と一致するオペレータ：文字列か、文字列の終わり、または文字列の改行文字と一致する。
"["	リストオペレータ：アイテムセットを一致するものとして定義できる。リストのアイテムは <code>[]</code> で囲む。空のリストは定義できない。
"("	グループオペレータ：通常表現式の中で単一ユニットとして使用可能なサブ式を定義する。
"-"	範囲オペレータ：リストの始めから終わりの文字を定義する。

正規表現を使う

正規表現を使ってテキストの検索、置換を実行することができます。

[単純な正規表現](#)

[任意の文字と一致](#)

[表現を繰り返す](#)

[表現のグループ化](#)

[多くの文字から 1 文字を選択](#)

[行の先頭または終わりとの一致](#)

[検索テキストを置換テキストとして使う](#)

[サブ表現の一部を記憶](#)

[リファレンス](#)

正規表現によって検索置換機能が向上します。サンプルを[例 6.1](#) に示します。

例 6.1 サンプルコード

```
#include <iostream>

#define var1 10;
#define var2 20;

using namespace std;

int result = 0;

int main(void)
{
    cout << "This example provides information about" << endl;
    cout << "the use of regular expressions in the" << endl;
    cout << "CodeWarrior IDE." << endl << endl;
    cout << "Refer to this code when reading the" << endl;
    cout << "sections in the manual dealing with" << endl;
    cout << "the use of regular expressions." << endl << endl;
    cout << "The value of var1 is " << var1;
    cout << " out of 50, and" << endl;
    cout << "the value of var2 is " << var2;
    cout << " out of 50." << endl;
    cout << "$" << var1;
    cout << " + $" << var2;
    cout << " = $" << endl;
    result += var1;
    result += var2;
    cout << result;
    cout << endl << endl;
```

```
    return 0;  
}
```

単純な正規表現

ほとんどの文字は、そのままの文字を表します。例えば `a` という文字はコード内のすべての `a` に一致します。例外は、特殊文字と呼ばれ、アスタリスク (`*`)、プラス記号 (`+`)、バックスラッシュ (`\`、日本語環境では円記号 `¥`)、ピリオド (`.`)、キャレット (`^`)、大括弧 (`[` と `]`)、ドル記号 (`$`) およびアンパサンド (`&`) が含まれます。特殊文字を普通の文字のように検索したい場合は、`*` のように、前にバックスラッシュを付けます。

[例 6.1 \(p138\)](#) の例で、コード内の `m` という文字を検索するには、[検索] フィールドに「`m`」と入力します。CodeWarrior IDE は `m` を持つ正規表現、`iostream` を探します。他にも `namespace`、`main(void)`、`example`、`information`、`manual` を探します。

コード内のドル記号を検索したい場合、[検索] フィールドに「`\$`」と入力します。バックスラッシュによってドル記号は普通の文字として解釈されます。IDE は最初のドル記号を探します ([例 6.1](#))。

```
cout << "$" << var1;
```

検索を続けるとこの後の 2 行にあるドル記号を探し出します。

任意の文字と一致

ピリオド (`.`) は、改行文字を除く任意の 1 文字と一致します。より柔軟な条件で検索ができます。

[例 6.1 \(p138\)](#) の例で、`var` で始まる 4 文字の式を検索するには、[検索] フィールドに「`var.`」(`var` とピリオド) と入力します。ピリオドは `var` の後に続く任意の文字に一致します。IDE は `var1`、`var2` を探し出します。

表現を繰り返す

アスタリスクまたはプラス記号は、検索文字列で表現を繰り返す特殊オペレータです。

正規表現の後にアスタリスク (`*`) を付けると、ゼロ回以上出現する正規表現と一致します。一致するものが複数ある場合は、1 行の中で一番長い、一番左にある一致したテキストを選択します。

正規表現の後にプラス記号 (`+`) を付けると、1 回以上出現する正規表現と一致します。一致するものが複数ある場合は、1 行の中で一番長い、一番左にある一致したテキストを選択します

正規表現の後にクエスチョンマーク (`?`) を付けると、ゼロ回以上出現する正規表現と一致します。一致するものが複数ある場合は、1 行の中で一番長い、一番左にある一致したテキストを選択します

[例 6.1 \(p138\)](#) の例で [検索] フィールドに「s*ion」と入力した場合、IDE は s の後に ion が続く単語を探します。[例 6.1](#) では、以下の 2 行に ion を含む表現式、information と sections があります。

```
cout << "This example provides information about" << endl;  
cout << "sections in the manual dealing with" << endl;
```

以下の行には ssion を含む正規表現、expressions があります。

```
cout << "the use of regular expressions in the" << endl;  
cout << "the use of regular expressions." << endl << endl;
```

ion の前に少なくとも 1 つの s がある単語を探すには、[検索] フィールドに「s+ion」と入力します。IDE は ion の直前に少なくとも 1 つの s がある単語を検出します。この正規表現は expressions の ssion と一致します ([例 6.1](#))。s+ion の場合、information や sections とは一致しません。ion の前に s がないからです。

数字のゼロの後にピリオド (0.) を含む、あるいはゼロだけを含む正規表現を検索するには、[検索] フィールドに「0\.?」と入力します。バックスラッシュにより、IDE はピリオドを普通の文字として扱います。特殊オペレータ ? は普通の文字として動作します。この表現は[例 6.1](#) での 0、および以下の行の 0. と一致します。

```
cout << " out of 50." << endl;
```

アスタリスクとプラス記号は通常 1 文字として参照されます。表現をグループ化することによって複数の文字を参照することも可能です。[表現のグループ化](#)を参照してください。

表現のグループ化

括弧 () で囲まれている表現は、1 つの表現として扱われます。アスタリスク (*) またはプラス記号 (+) は表現のグループに適用されます。

[例 6.1 \(p138\)](#) の例では、is に一致する表現式を検索するには、[検索] フィールドに「is」と入力します。正規表現として (i) s も使えます。(i) s により、IDE はスペースの後にある s、i の後ろにある s を探し出します。is に一致するのは、This、this、is です。(i) s は以下の 2 行の is にのみ一致します ([例 6.1](#))。

```
cout << "The value of var1 is " << var1;  
cout << "The value of var2 is " << var2;
```

多くの文字から 1 文字を選択

大括弧 [] で囲まれた文字列の表現は、[] 内の任意の 1 文字と一致するテキストを検出します。[] で囲まれていない文字を検出するには、[^abc] のようにキャレット (^) を使います。

[例 6.1 \(p138\)](#) の例で、`x`、`y`、`z` を含む表現式を検索するには、[検索] フィールドに「`[xyz]`」と入力します。IDE は `x` に一致するものとして `example`、`expressions` を検出します。`x`、`y`、`z` を含まない表現式を検索するには、[検索] フィールドに「`[^xyz]`」と入力します。`example`、`expressions` の `x` 以外の文字を表現を一致するものとして検出します。

[] 内にあるマイナス記号 `-` は ASCII 文字の範囲を示します。例えば、`[0-9]` は `[0123456789]` を意味します。マイナス記号は、[] で囲まれたテキストの最初または最後の文字である場合には特殊な意味を失い、普通の文字として扱われます。`[-bc]` は範囲ではなく、マイナス記号と `b` と `c` を表します。

[] が [の直後に置かれると、テキストを終了する特殊な意味を失い、検索対象となる普通の文字として扱われます。例えば `[]0-9` の場合、IDE は [と数字を検索します。バックスラッシュ (`\`)、アスタリスク (`*`)、またはプラス記号 (`+`) が [の直後に置かれると、特殊な意味を失い、検索対象となる普通の文字として扱われます。`[.]` の場合、IDE は `.` を検索します。

[] の使い方は () とほぼ同じです。IDE は [] 内の文字を単一ユニットとして扱います。例えば `[bsl]ag` の場合、IDE は `bag`、`sag`、`lag` を検出します。`[aeiou][0-9]` の場合、IDE は `a1` などの母音に続く数字を検出します。

行の先頭または終わりとの一致

行の先頭または終わりと一致する正規表現を指定できます。

キャレット (`^`) が正規表現全体の一番前にある場合、これは行の先頭と一致します。

ドル記号 (`$`) が正規表現全体の一番後ろにある場合、これは行末と一致します。

正規表現全体がキャレットとドル記号で囲まれる場合 (`^like this$`)、行全体と一致します。

[例 6.1 \(p138\)](#) の例で行の先頭にある `cout` を検索するには [検索] フィールドに「`^[\t]*cout`」と入力します。`[\t]*` は、複数のスペースやタブが `cout` の前にあってもそれらを検出します。

検索テキストを置換テキストとして使う

検索テキストを置換テキスト内で使うには、アンパサンド (`&`) で指定します。[例 6.1 \(p138\)](#) の例で、検索テキストが `var[0-9]`、置換テキストが `my_&` だとします。`var1` と `var2` が検出されます。「検索」ダイアログの [置換] ボタンをクリックすると `var1` が `my_var1`、`var2` が `my_var2` に置換されます。

置換テキスト中で特殊な意味ではないアンパサンドを使いたい場合は、`\&` と入力します。アンパサンドは検索テキスト中では特殊な意味を持ちません。

サブ表現の一部を記憶

検索文字列である正規表現の一部を記憶して、その一部を置換テキストとして使うことができます。検索文字列で最高 9 個のサブ式を作ることができます。格サブ式を () で括ります。これを置換文字列で使うには、「`\n`」と入力します。「`n`」はサブ式を指定する番号です。検索文字列の左から数えた番号です。

[例 6.1 \(p138\)](#) の例では、`#define` 宣言を `const` 宣言に変えるには、`#define` を検索してから手動で `const` 宣言に変えます。またはサブ式の記憶を利用して自動的に変更することもできます。[検索] フィールドに以下の正規表現を入力します。

```
\#define[ \t]+(.+)[ \t]+([0-9]+);
```

これにより、IDE は `#define` と複数のスペースやタブ、文字、桁、セミコロンを検索します。正規表現の左から、第 1 のサブ式は `(.+)`、第 2 は `([0-9]+)` です。2 つのサブ式は以下の置換文字列で利用されます。

```
const int \1 = \2;
```

`\1` は検索文字列の第 1 のサブ式、`\2` は第 2 サブ式を表します。これら 2 つのサブ式は元の `#define` 宣言の変数名と変数を使います。置換文字列は 2 つのサブ式を参照して `#define` 宣言を `const` 宣言に代えます。[例 6.1](#) では、`#define var1 10;` を検出し、これを `var1 = 10;` へ変えます。次の `#define` ステートメントも同様に変更します。

リファレンス

『MasteringRegular Expressions』(Jeffrey E.F. Friedl 著、O'Reilly & Associates, Inc. 発行) は正規表現の使い方の優れた参考書です。

第 7 章 ソースコードのブラウズ

この章では、プロジェクトソースコードをいろいろな観点で調べるためのツールである、CodeWarrior のクラスブラウザを説明します。

CodeWarrior IDE ブラウザは、コード内のすべてのシンボルのデータベースを作成し、言語に関係なくデータに素早く簡単にアクセスできるユーザーインターフェースを提供します。

プログラマは主にオブジェクト指向のコードと共にブラウザを使用していましたが、CodeWarrior IDE ブラウザは、プロシージャ型のコードでもオブジェクト指向のコードでも使えます。C、C++、Pascal、および Java を含む大部分のコンパイラと共に使えます。

ブラウザを理解しやすいように、3 つの段階（高レベルのアーキテクチャ、ユーザーインターフェース、機能）に分けて説明していきます。

ここでは次の項目について説明します。

[ブラウザを利用](#)

[ブラウザの概要](#)

[ブラウザの解説](#)

[ブラウザウィザード](#)

[ブラウザを使う](#)

ブラウザを利用

ブラウザを利用する方法については以下を参照してください。

[「IDE のオプション設定」\(p181\)](#)：ブラウザを利用するためのオプションを含むダイアログの説明

[「ブラウザを利用する」\(p236\)](#)：ブラウザを利用するためのターゲット固有の環境設定

ブラウザが利用可能になると、コンパイラがブラウザデータベース情報を生成します。

ブラウザの設定およびオプションの詳細は [「ブラウザのオプションを設定」\(p174\)](#) を参照してください。

コンテキストメニューの詳細は [「ブラウザのコンテキストメニュー」\(p148\)](#) を参照してください。

ブラウザの概要

ブラウザを利用すると、CodeWarrior IDE コンパイラがコードに関する情報のデータベースを生成します。このデータベースは、コードに関するデータに加え、継承の階層などコード部分の間に存在する関係も含んでいます。

ブラウザは、この情報を要求に応じて並べたり選択したりするためのユーザーインターフェースです。

よくできたデータベースアクセスプログラムと同様に、ブラウザの情報表示方法に制限はありません。ユーザーの作業スタイルに合わせてさまざまなツールが提供されています。

ブラウザで使用可能な情報を表示するには、基本的に 3 つの方法があります。

[コンテンツビュー](#)：すべてのデータの包括的な表示

[ブラウザビュー](#)：クラスベースの表示

[階層ビュー](#)：継承ベースの表示

ここでは、各オプションを簡単に説明します。「[ブラウザの解説](#)」(p147) で、ユーザーインターフェースの詳細を説明します。

ブラウザには、情報に簡単にアクセスする機能もあります。データベースに情報があるシンボルをマウスで右ボタンクリックすると、関連するソースコードに簡単にアクセスできます。「[ブラウザのコンテキストメニュー](#)」(p148) でこの機能を説明します。

ブラウザでは、データを表示する欄の大きさを決めることができます。クラスすべてのデータを表示することも、1 つのクラスに焦点を絞ることもできます。

ブラウザおよび階層ビューで、複数のクラスまたは単一のクラスを表示できます。[表 7.1](#) に、ブラウザを使うときの主な選択項目をまとめます。

表 7.1 ブラウザビューのオプション

表示スタイル	広いフォーカス	狭いフォーカス
包括的なビュー	コンテンツ	なし
継承ベースのビュー	複数クラス階層ウィンドウ	単一クラス階層ウィンドウ

[ウィンドウメニュー](#)にあるブラウザ関連のコマンド ([[ブラウザコンテンツ](#)] [[クラス階層ウィンドウ](#)] [[新規クラスブラウザ](#)]) は広いフォーカスのビューを表示します。広いビューを表示すると、特定のクラスにもフォーカスを当てることができます。

ビュースタイルやフォーカスによらず、他の種類のビューに切り替えるための簡単で直感的な機能があります。

コンテンツビュー

ブラウザコンテンツウィンドウは、カテゴリごとに保存されたすべてのデータをアルファベット順に表示します。[図 7.1](#) にブラウザコンテンツを示します。

調べたいカテゴリをポップアップメニューから選択できます。クラス、定数、enum 型（列挙型）、関数、大域変数、マクロ、関数テンプレート、および型定義にフォーカスを当てることができます。

カタログウィンドウの詳細は、「[ブラウザコンテンツ](#)」(p149) を参照してください。

図 7.1 ブラウザコンテンツウィンドウ



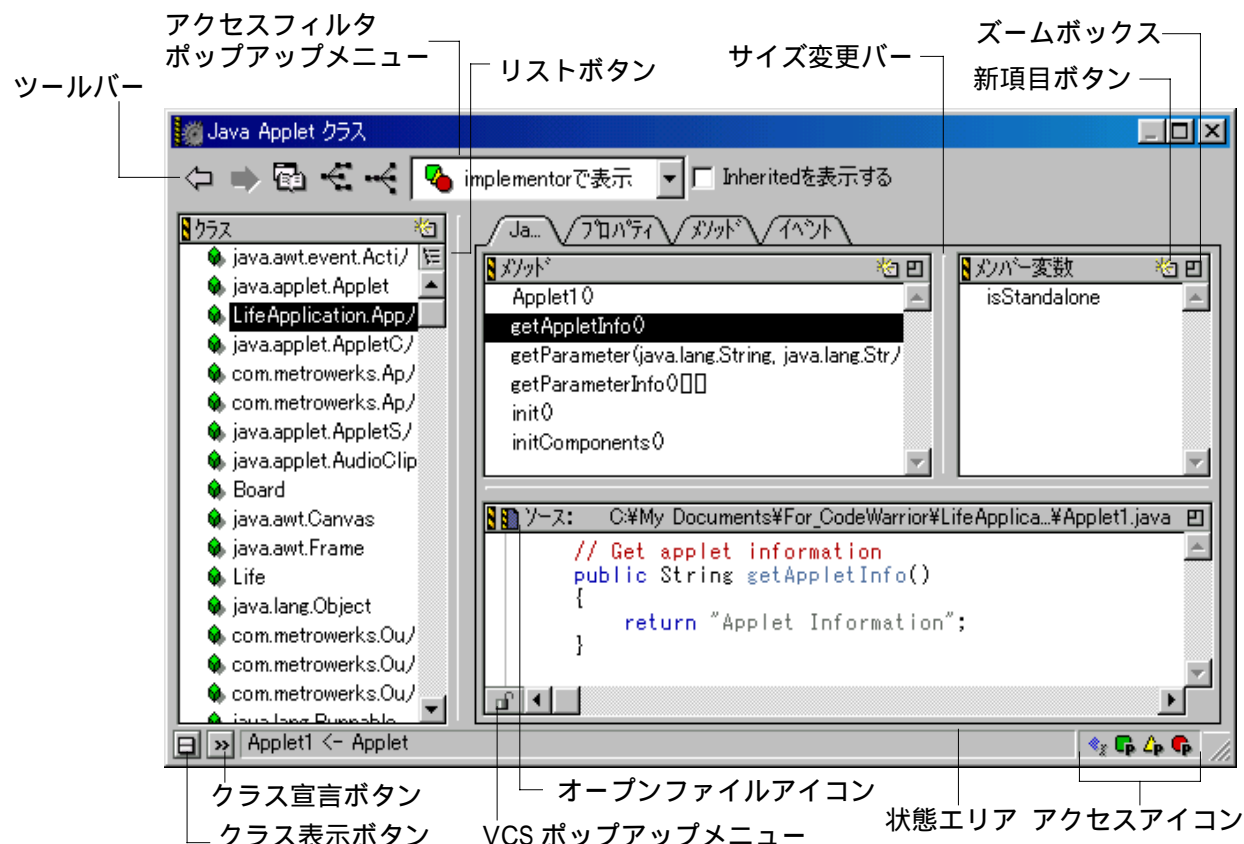
ブラウザビュー

ブラウザビューは従来のクラスブラウザと似ています。このビューを使って、クラス指向の観点からデータを見ることができます。[図 7.2](#) にブラウザのビューを示します。

ブラウザビューにはクラスが表示されます。リスト内で選択したクラスの、すべてのメンバ関数およびデータメンバを見ることができます。項目を選択すると、それに関連したソースコードがソース欄に表示されます。

ブラウザビューインターフェースの詳細は「[ブラウザウィンドウ](#)」(p150) を参照してください。

図 7.2 ブラウザビュー



注意： クラスブラウザビューにはツールバーがあります。使い方、カスタマイズの仕方は「[IDE をカスタマイズ](#)」(p207)を参照してください。RAD アプリケーションのブラウザビューにはタブが現れます。詳細は「[RAD ブラウザ](#)」(p423)を参照してください。

階層ビュー

階層ビューは、クラス階層を視覚的に表示します。このビューでクラスの間係を一目で理解できます。図 7.3 にいくつかのクラスの複数クラス階層ビューを示します。



ここでは、ブラウザで使われる各ウィンドウとそのコントロールを説明します。

[ブラウザのコンテキストメニュー](#)

[ブラウザコンテンツ](#)

[ブラウザウィンドウ](#)

[クラス階層ウィンドウ](#)

[単一クラス階層ウィンドウ](#)

[シンボルウィンドウ](#)

[ブラウザメニュー](#)

ブラウザのコンテキストメニュー

ブラウザがアクティブなときに、ブラウザデータベース内にデータがあるシンボルを右ボタンクリックすると、さまざまな項目を含むコンテキストメニューが表示されます。

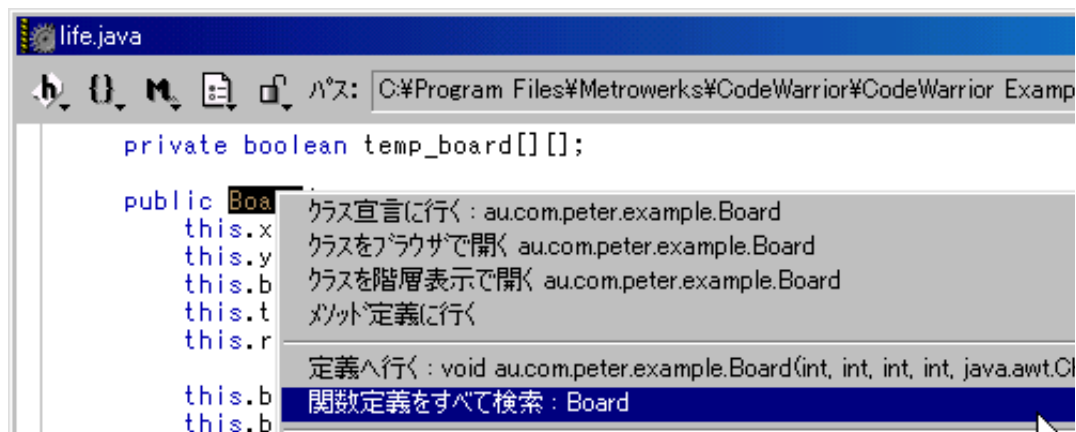
コンテキストメニューのコマンドは、調べたいシンボルの属性によって変わります。ブラウザウィンドウを開くコマンドや、コード内の位置へ直接ジャンプするコマンドがあります。

コンテキストメニューでデバッグに関する作業も行うことができます。例えばアクティブなウィンドウ上にブレークポイントを設定したり、消去することができます。

コードにあるシンボル（関数名、クラス名、データメンバ名、定数、enum 型、テンプレート、マクロ、型定義など）は、ソースコード内のある位置へのハイパーテキストリンクになります。

例えばクラス名を選択してブラウザのコンテキストメニューを使うと、クラスの宣言、[ブラウザウィンドウ](#)、[単一クラス階層ウィンドウ](#)を開くことができます。関数名のコンテキストメニューには別のコマンドが表示されます（[図 7.4](#)）。

図 7.4 関数のコンテキストメニュー



希望の位置に関数のテンプレートを挿入できます。「[ブラウザ表示](#)」(p193)を参照してください。

他の種類のシンボルのメニューにも同様のコマンドがあります。

ヒント： 選択または入力したテキストを見つけるにはそのテキストをクリックして、コンテキストメニューが表示されるのを待ちます。ここに一致するテキストが表示されます。ここでそのテキストを選択するとブラウザに入力されます。「[シンボルを補完](#)」(p176)を参照してください。

ヒント： コンテキストメニュー機能はブラウザウィンドウ内だけでなく、ソースコードエディタでも利用できます。これが、ブラウザウィンドウを使わないときでもブラウザを常に使用可能にしておく大きな理由です。

関数名のコンテキストメニューで注目すべきコマンドは[関数定義をすべて検索]です。複数の定義を持つ関数でこのコマンドを選択すると[シンボルウィンドウ](#)が開きます。

ブラウザコンテンツ

ブラウザコンテンツ (図 7.5) は、カテゴリ別に分類されたブラウザデータをアルファベット順に表示します。ブラウザコンテンツを表示するには、[ウィンドウメニュー](#)の[[ブラウザコンテンツ](#)]を選択します。

図 7.5 ブラウザコンテンツ



ウィンドウ内には以下のアイテムがあります。

[カテゴリポップアップメニュー](#)

[シンボル欄](#)

カテゴリポップアップメニュー

カテゴリポップアップメニューで[シンボル欄](#)に表示する情報の種類を選択します。現在選択されている項目は、ハイライトで表示されます。

シンボル欄

シンボル欄は、現在選択されているカテゴリのメンバの、ブラウザデータベース内のすべての項目を表示します。項目はアルファベット順にリストされます。

注意： 関数は、名前によってアルファベット順にリストされますが、クラス名が最初に表示されます。このため、関数がアルファベット順にはリストされていないように見えます。

ブラウザウィンドウ

ブラウザウィンドウ ([図 7.2 \(p146\)](#)) はブラウザデータベースの情報をクラスごとに表示します。ウィンドウに複数の欄があります。

以下の方法でブラウザウィンドウを開きます。

[ウィンドウメニュー](#)の [[新規クラスブラウザ](#)] を選択する

関数を表示しているときに、[コンテンツビュー](#)の[ブラウザのコンテキストメニュー](#)を使う

[クラス階層ウィンドウ](#)でクラス名をダブルクリックする

[単一クラス階層ウィンドウ](#)でクラス名をダブルクリックする

ヒント： クラスブラウザビューのツールバーの使い方、カスタマイズの方法については「[IDE をカスタマイズ](#)」([p207](#)) を参照してください。Java 用 RAD (Rapid Application Development) ツールのブラウザウィンドウにはタブがあります。「[RAD ブラウザ](#)」([p423](#)) を参照してください。

[クラス欄](#)、[メンバー関数欄](#)、および[データメンバー欄](#)は、それぞれのデータをリストします。欄をクリックすることによりアクティブにできます。Tab キーを使って欄を順にアクティブにすることもできます (ソース欄を除きます)。

ヒント： Tab キーの使用は、ソースコードにとって若干危険があります。ソース欄がアクティブであるときに Tab キーを押すと、タブがソースコード内に入力されてしまいます。ソース欄では Tab キーを使ってコードをナビゲートすることはできません。

別の欄へ移動するにはマウスを使ってください。

リスト内の項目を選択するには、項目をクリックするか、矢印キーを使います。名前を入力することでも選択できます。入力したテキストにもっとも近い項目が選択されます。

このウィンドウ内には以下があります。

[ツールバー](#)

[アクセスフィルタポップアップメニュー](#)

[ズームボックス](#)

[サイズ変更バー](#)

[クラス欄](#)

[リストボタン](#)

[クラス欄ボタン](#)

[クラス宣言ボタン](#)

[メンバー関数欄](#)

[データメンバー欄](#)

[識別アイコン](#)

[ソース欄](#)

[ファイルボタン](#)






[VCS ポップアップメニュー](#)

[状態エリア](#)

ツールバー

ブラウザのツールバーのボタンをクリックすると CodeWarrior コマンドが実行されます。ブラウザウィンドウのツールバーのボタンを以下にまとめます ([表 7.2](#))。

表 7.2 ブラウザのツールバーのボタン

ボタン	動作
	戻る
	進む
	ブラウザコンテンツを表示
	複数クラス階層ウィンドウを表示
	単数クラス階層ウィンドウを表示







アクセスフィルタポップアップメニュー

ブラウザのアクセスフィルタポップアップメニューは、ブラウザウィンドウにおけるメンバ関数とデータメンバーの表示をコントロールします。ポップアップメニューのコマンドは、public、private、protected アクセス型に従ってフィルタを実行します。利用可能なフィルタにマーカがつきます。ポップアップメニューのコマンドを[表 7.3](#)にまとめます。

最初の 3 つのコマンドは複数フィルタへのショートカットです。例えば public、private、protected フィルタを実行するためにブラウザアクセスフィルタポップアップメニューを 3 回使うよりも、[Implementor で表示]を選択すると、一度に 3 つのフィルタを実行できます。


フィルタを実行した後、ブラウザの右下にあるアイコンが変化します。使用中のアイコンは暗く、使用不可のアイコンは灰色で表示されます（[表 7.3](#)）。


表 7.3 アクセスフィルタポップアップメニューのコマンド

コマンド	アイコン	表示する項目
Implementor で表示		public、private、protected access
サブクラスで表示		public and protected access
ユーザで表示		public access
public を表示		public access
protected を表示		protected access
private を表示		private access

ズームボックス

ズームボックスはブラウザの各欄の大きさをコントロールします。

 これをクリックすると欄がブラウザウィンドウいっぱいになります。

 これをクリックすると元の大きさに戻ります。

サイズ変更バー

サイズ変更バーは欄の間にあります ([図 7.2 \(p146\)](#))。これをドラッグすると欄の大きさが変わります。

ヒント： ソースファイルが[ソース欄](#)に表示されているとき、Control + Tab キーを押すと関連するヘッダファイルを開きます。[ソース欄](#)にヘッダファイルが表示されているとき、関連するソースファイルを開きます。

クラス欄

クラス欄は、ブラウザデータベース内のすべてのクラスを表示します。

リストをアルファベット順またはクラス階層で表示できます。表示の切り替えは、欄の右上の[リストボタン](#) ([図 7.2 \(p146\)](#)) をクリックします。

階層ビューでは、サブクラスを持つクラス名の横に拡張ボタンが表示されます ([図 7.3 \(p147\)](#))。拡張ボタンをクリックすると、隠されていたサブクラスが表示されます。


ヒント： 拡張ボタンを Alt + クリックすると、すべてのレベルのすべてのサブクラスを開きます。これを「深さの拡張」といいます。Ctrl + クリックすると、クラスのサブクラスと同じレベルにある兄弟クラスを表示します。これを「広さの拡張」といいます。

クラス欄でクラスを選択すると、[クラス階層ウィンドウ](#)も変化します。必要に応じて階層ビューは新たに選択されたクラスへスクロールします。

注意： メンバ関数、基底クラス、サブクラスを持たないクラスや構造体の情報はクラス欄に表示されません。つまりフィールドやデータメンバしか持たない構造体やクラスは表示されません。

リストボタン

リストボタンは[クラス欄](#)の右上にあり、クラスの表示方法をコントロールします。アルファベット順か階層順で表示できます。

 これをクリックするとアルファベット順でクラスを表示します。


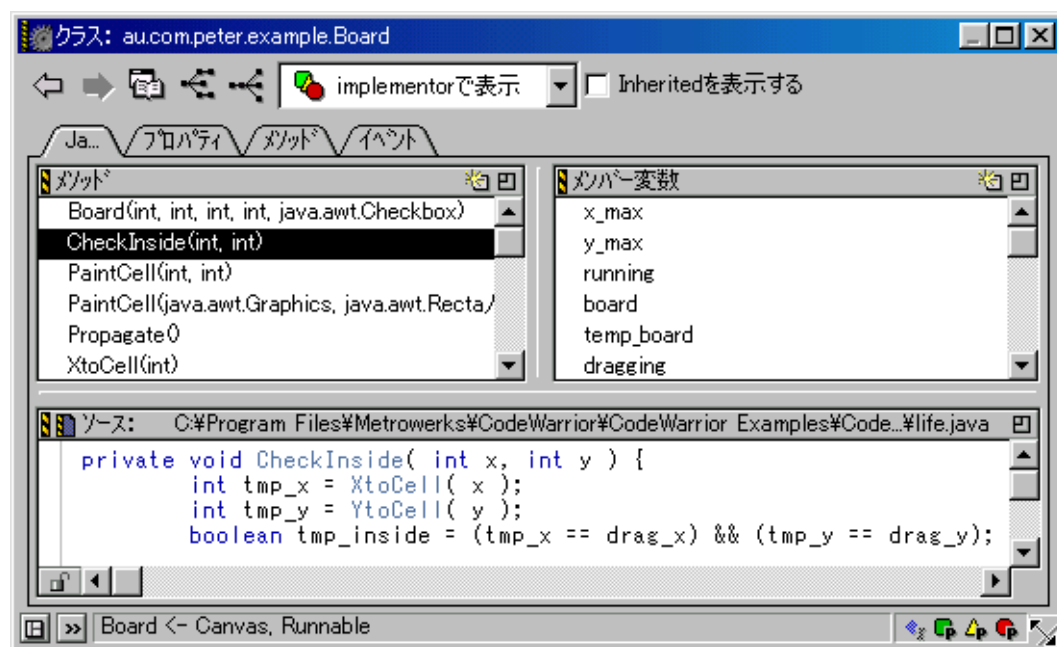

 これをクリックすると階層順でクラスを表示します。


図 7.6 クラス欄を隠したブラウザウィンドウ



クラス欄ボタン


クラス欄ボタンはブラウザウィンドウの左下にある、[クラス欄](#)の表示をコントロールします。

 これをクリックするとクラス欄を隠します ([図 7.6](#))。

 これをクリックするとクラス欄を表示します ([図 7.2](#))。


クラス宣言ボタン

クラス宣言ボタンはブラウザウィンドウの左下にある、現在のクラスへアクセスします。

 クラス宣言ボタンをクリックすると、現在のクラスの宣言が[ソース欄](#)に表示されます。現在のクラスはブラウザウィンドウの[状態エリア](#)に表示されます。


メンバー関数欄

メンバー関数欄は、現在選択しているクラスで定義されているすべてのメンバー関数をリストします。コンストラクタ関数とデストラクタ関数がリストの先頭に、その後他の項目がアルファベット順に表示されます。

ツールバーの [Inherited を表示] チェックボックスをオンにすると、継承されたメンバ関数を表示できます。継承されたメンバ関数を表示しているとき、ブラウザウィンドウの Inherited アクセスアイコン () が暗くなります。

ヒント： [ブラウザウィンドウのメンバー関数欄](#) でメンバ関数を選択した場合、Enter キーでエディタウィンドウを開いてその関数の定義を見ることができます。

データメンバー欄

データメンバー欄は、現在選択しているクラスで定義されているすべてのデータメンバーをリストします。ツールバーの [Inherited を表示] チェックボックスをオンにすると、継承されたデータメンバーを表示できます。継承されたデータメンバーを表示しているとき、ブラウザウィンドウの [Inherited] アクセスアイコン () が暗くなります。




データメンバーはアルファベット順に並べられます。継承されたメンバーが表示されている場合は、データメンバーがスーパークラスによりリストされますが、各クラス内ではアルファベット順に表示されます。

ヒント： [ブラウザウィンドウのデータメンバー欄](#) でデータメンバーを選択した場合、Enter キーでエディタウィンドウを開いてそのデータメンバーの定義を見ることができます。

識別アイコン

関数やデータメンバーの名前の横に識別アイコンが表示されます。次の表は、このアイコンの意味を説明します。

表 7.4 ブラウザの識別アイコン

アイコン	意味	メンバー
	static	静的メンバー
	virtual	オーバーライド可能な仮想関数、または継承された関数のオーバーライド
	pure virtual または abstract	サブクラスでインスタンスを作成する場合は、サブクラス内でオーバーライドしなければならないメンバ関数

ソース欄

ソース欄は、現在選択されている項目のソースコードを表示します。項目がクラスの場合、この欄はクラス宣言を表示します。項目が関数の場合、関数定義を表示します。項目がデータメンバーの場合、ヘッダファイルからデータメンバー宣言が表示されます。

注意： [メンバー関数欄](#)または[データメンバー欄](#)の項目を Alt+ クリックすると、項目が [ソース欄](#)のテキストの現在の挿入位置に入力されます。これは、[ソース欄](#)にルーチンコールや変数名を入力する簡単で確実な方法です。

ソース欄のテキストは編集可能です。表示されているコードを含むファイルへのパスは、欄の一番上に示されます。

ファイルボタン



ファイルボタンはソース欄の上にあります。これをクリックすると、ソース欄に表示されているコードを含むファイルが開きます。ソースコードは新しいエディタウィンドウに表示されます。

VCS ポップアップメニュー



このポップアップで表示しているソースファイルのバージョン管理を行います。この機能の詳細は「[一般的な VCS の操作](#)」(p452) を参照してください。

状態エリア

状態エリアは状況メッセージなどを表示します。例えばクラス欄でクラスを選択したとき、状態エリアはそのクラスの基底クラスを表示します。

クラス階層ウィンドウ

複数クラス階層ウィンドウ ([図 7.7](#)) はブラウザデータベースにあるクラスの完全なマップを視覚的に表示します。四角形の中にクラス名があり、それに関連するクラスは線で結ばれます。複数クラス階層ウィンドウを表示するには[ウィンドウメニュー](#)の[[クラス階層ウィンドウ](#)]を選択します。

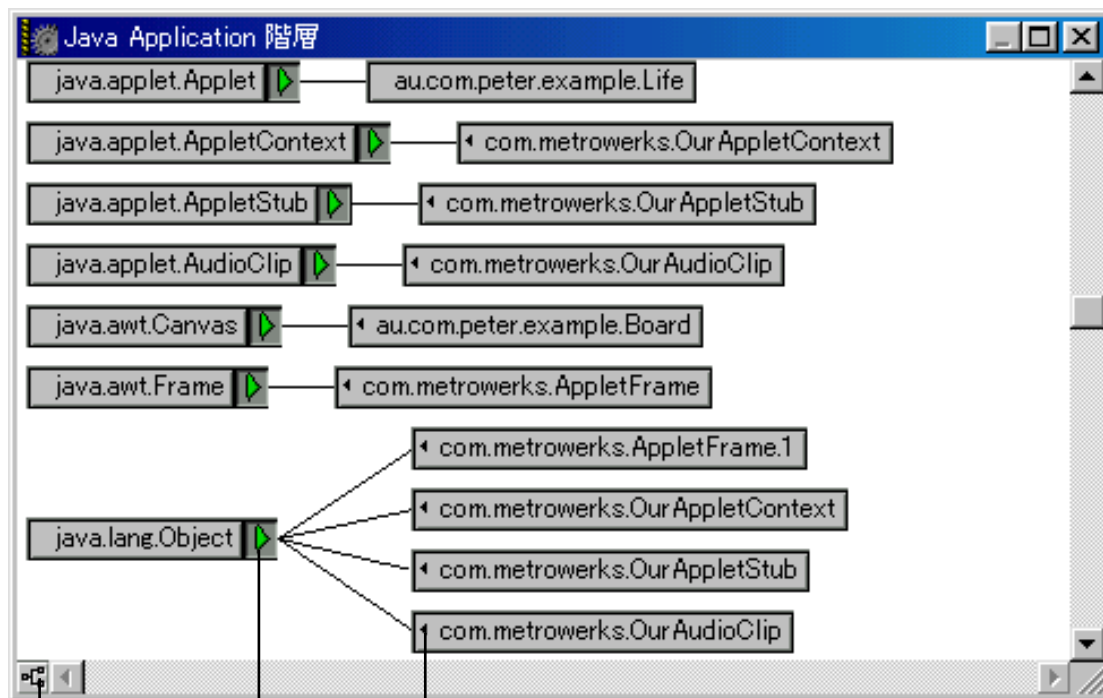
矢印キーで選択しているクラスを移動できます。上下の矢印キーは同じ階層を、左右の矢印キーは先祖、子孫を移動します。

タイプ入力でも選択クラスを移動できます。入力したテキストにもっとも近い名前のクラスを選択します。Tab キーはクラスをアルファベット順に移動します。

ヒント： [クラス欄](#)で選択したクラスに伴って、[クラス階層ウィンドウ](#)での選択クラスも変わります。

クラスをダブルクリック、または選択してからキーを押すと、そのクラスの[ブラウザウィンドウ](#)が開きます。

図 7.7 クラス階層ウィンドウ（複数）



線ボタン 階層拡張ボタン 先祖クラスポップアップメニュー

複数のクラス階層ウィンドウには以下の項目があります。

[線ボタン](#)

[階層拡張ボタン](#)

[先祖クラスポップアップメニュー](#)

線ボタン

線ボタンは関連するクラスをつなぐ線の外観をコントロールします。直線と水平垂直線を切り替えることができます。これは外観を変えるだけです。

階層拡張ボタン

階層拡張ボタンはサブクラスの表示をコントロールします。

この拡張ボタンをクリックすると、次のレベルのサブクラスが表示または隠されます。拡張した状態は、最後にこのクラスが開かれたときの状態が保存されています。

注意： 拡張ボタンを Alt + クリックすると、すべてのレベルのすべてのサブクラスを開きます。これを「深さの拡張」といいます。Ctrl + クリックすると、クラスのサブクラスと同じレベルで並んだ兄弟クラスをすべて表示します。これを「広さの拡張」といいます。「深さ」と「広さ」の両方の情報を表示するには、Ctrl + Alt + クリックします。この

キーの組み合わせは、先祖を持たない一番上の基本クラスに対して拡張ボタンをクリックしたときと同様に、マップ全体を拡張または縮小します。

先祖クラスポップアップメニュー

先祖クラスの拡張ボタンをクリックすると、ポップアップメニューが開きます([図 7.7](#))。このメニューは直接の先祖を表示します。項目を選択すると、マップ内のそのクラスにジャンプします。項目が現在表示されていない場合は、ピープ音が鳴ります。

このコントロールは、複数の基底クラスを持つクラスにのみ表示されます。

単一クラス階層ウィンドウ

単一クラス階層ウィンドウは、ブラウザデータベースにある 1 つのクラスの完全なマップを視覚的に表示します。このマップは、クラスの直接の先祖を「すべて」、またすべての子孫を表示します。([クラス階層ウィンドウ](#) は、基底クラスを 1 つしか表示できません。)

[図 7.8](#) に、複数の基底クラスとサブクラスを表示しているクラスウィンドウの例を示します。下線の付いたクラス名が、フォーカスの当たっているクラスです。

単一クラス階層ウィンドウは、以下の方法で開くことができます。

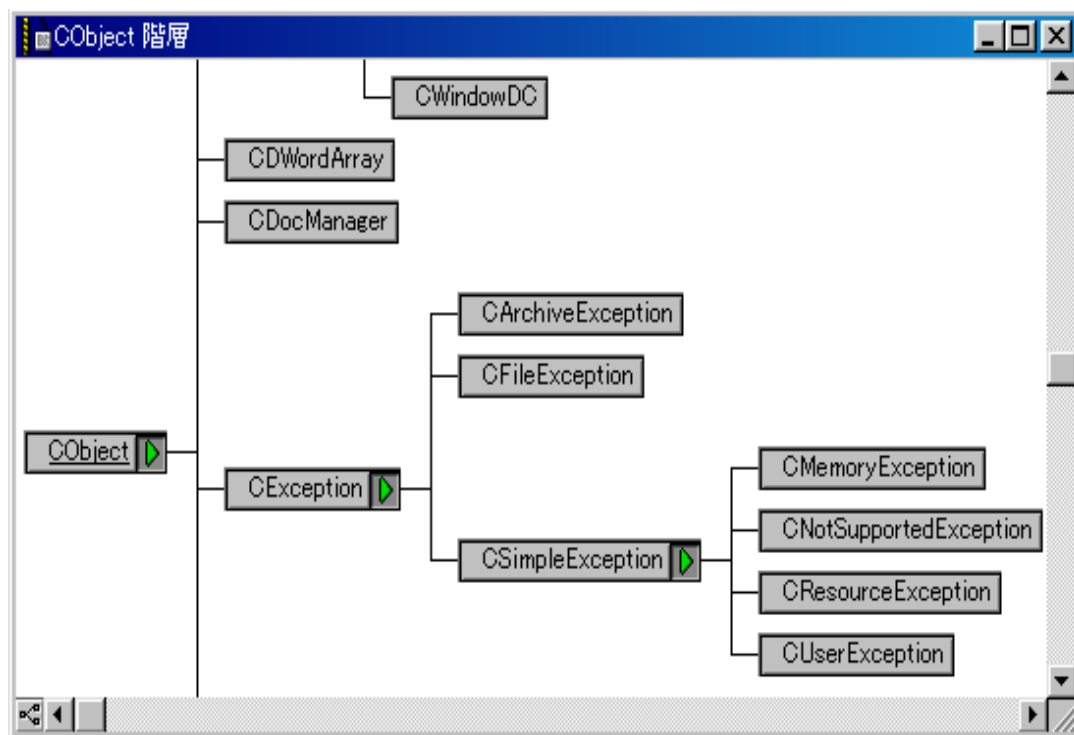
[ブラウザコンテンツのブラウザのコンテキストメニュー](#)を使う

[クラス階層ウィンドウのブラウザのコンテキストメニュー](#)を使う

[ブラウザウィンドウのブラウザのコンテキストメニュー](#)を使う

[ツールバー](#)の [[複数クラス階層ウィンドウを表示](#)] ボタンを使う

図 7.8 単一クラス階層ウィンドウ

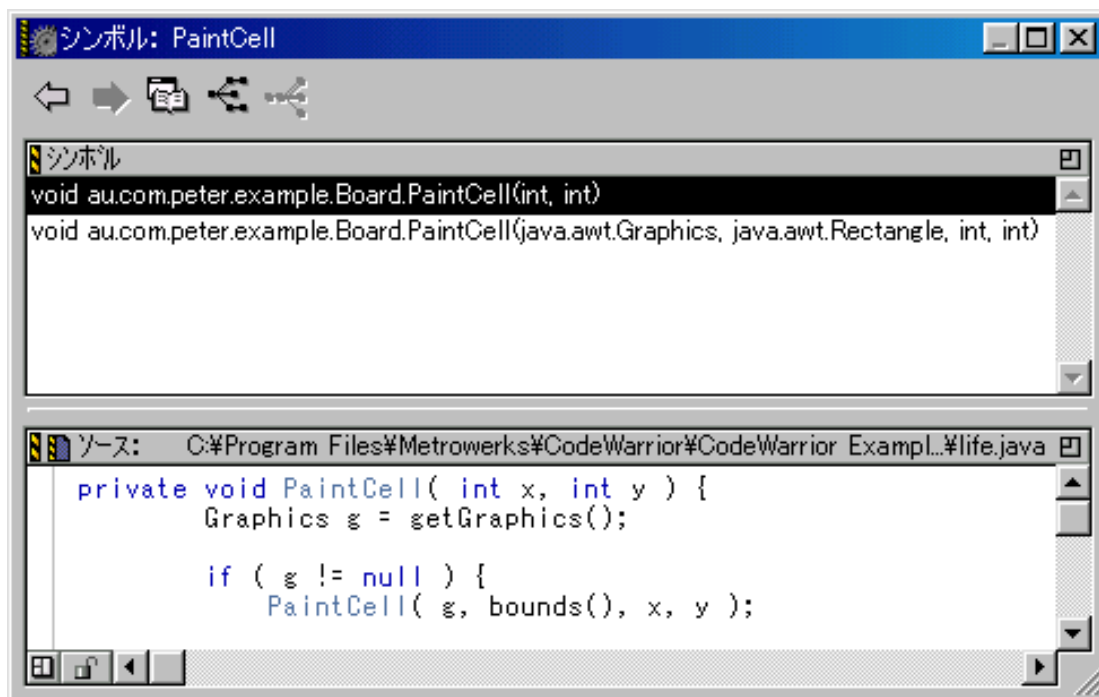


[単一クラス階層ウィンドウ](#)は表示するマップに制限があるだけで、その他は[ブラウザウィンドウ](#)と同じです。このウィンドウの動作については「[ブラウザウィンドウ](#)」(p150)を参照してください。

シンボルウィンドウ

シンボルウィンドウ ([図 7.9 \(p160\)](#)) は、多重定義を持つシンボルの実装をすべてリストします。一般にオブジェクト指向コードでは関数のオーバーライドにより、複数のバージョンが存在します。しかしシンボルウィンドウは、データベース内で多重定義されている「どんな」シンボルでも表示します。

図 7.9 シンボルウィンドウ



シンボルウィンドウでリストで実装を選択することにより、その定義をソース欄で見ることができます。

シンボルウィンドウを開くには、ブラウザがエディタウィンドウ上でシンボル名を右ボタンクリックすると、[ブラウザのコンテキストメニュー](#)が表示されます。関数に複数の実装がある場合、このポップアップメニューのコマンドの 1 つは [関数定義をすべて検索] になります。これを選択すると、シンボルウィンドウが表示されます。

ヒント： [ソース欄](#)またはエディタウィンドウで、ルーチンまたはその他のシンボル名を、Alt + ダブルクリック、または Ctrl + クリックすると、すべての実装が検索され、ポップアップメニューを使わずにシンボルウィンドウが開きます。

このウィンドウ内のほとんどの項目は、[ブラウザウィンドウ](#)のものと同じです。ウィンドウの概要は「[ブラウザウィンドウ](#)」(p150) を参照してください。下記のシンボルウィンドウの項目の説明があります。

[ツールバー](#)

[サイズ変更バー](#)

[ソース欄](#)

[ファイルボタン](#)

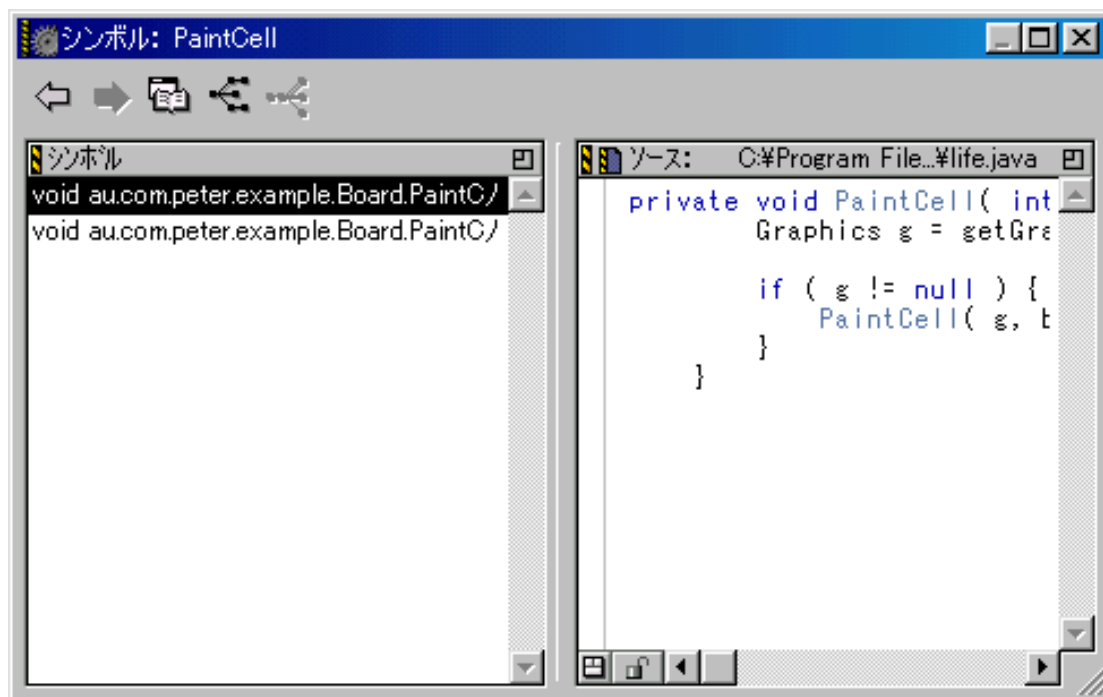
[VCS ポップアップメニュー](#)

このウィンドウ固有の項目に、シンボル欄と方向ボタンがあります。

シンボル欄([図 7.9](#))は、データベースにあるシンボルのすべてのバージョンを表示します。シンボル欄で項目を選択すると、その項目の定義が[ソース欄](#)に表示されます。

方向ボタンはシンボル欄とソース欄の位置を上下と左右に切り替えます([図 7.9](#)、[図 7.10](#))。

図 7.10 シンボルウィンドウ：左右に表示



ブラウザメニュー

[コンテンツビュー](#)、[ブラウザビュー](#)、[階層ビュー](#)を表示しているときはブラウザメニューがメニューバーに表示されます。新しい項目を作成するコマンドがあります。選択したコマンドによって異なるウィザードが開きます。ウィザードには新しいクラス、メンバー関数、データメンバーを作成するためのオプションがあります。

ブラウザメニューには以下のコマンドがあります。

新規クラス：新しいクラスの作成を支援する[新規クラスウィザード](#)を開きます。ウィザードを使って新しいクラスの名前、位置、ファイルタイプ、演算子を指定できます。

新規メソッド：選択したクラスに対する新しいメンバー関数の作成を支援する[新規メンバー関数ウィザード](#)を開きます。ウィザードを使って新しいメンバー関数の名前、戻り値の型、パラメータ、演算子などを指定できます。

新規データメンバー：新しいデータメンバーの作成を支援する[新規データメンバーウィザード](#)を開きます。ウィザードを使って新しいデータメンバーの名前、型、初期設定子、演算子などを指定できます。

ブラウザウィザード

新しいクラス、メンバー関数、データメンバーを作成するときにブラウザウィザードを選択します。ウィザードは各項目の作成を支援します。以下の項目について説明します。

[新規クラスウィザード](#)

[新規メンバー関数ウィザード](#)

[新規データメンバーウィザード](#)

初心者：各ウィザードはプログラミング言語の知識をお持ちの方を対象に設計されています。ウィザードのオプションについて分からない方はプログラミング言語のリファレンスを参照してください。

ブラウザウィザードには以下のナビゲーションボタンがあります。

戻る：前のページに戻る

次へ：次のページに進む

完了：現在の情報を表示する

キャンセル：変更を破棄する

各ウィザードは一連のページを表示します。順番に従って各ページで情報を設定し、終了したら [次へ] ボタンをクリックします。

各ページのデフォルト設定を変更することもできます。現在のウィザードの設定を適用するには [完了] ボタンをクリックします。ウィザードは新規プロジェクトの現在の設定を表示します。現在の設定を適用して新しい RAD アイテムを作るには、「一覧」ウィンドウの [作成] ボタンをクリックします。ウィザードに戻って設定の変更を続けるには [キャンセル] ボタンをクリックします。

新規クラスウィザード

以下は新規クラスウィザードを使う手順です。

1. [新しいクラスの名前と位置を指定する](#)
2. [新クラスの基底クラスとメソッドを指定する](#)
3. [新クラス用の #include ファイルをリストする](#)
4. [新クラスをプロジェクトのビルドターゲットに割り当てる](#)

1. 新しいクラスの名前と位置を指定する

ウィザードは新しいクラスの名前、宣言、位置を指定するパネル ([図 7.11](#)) を表示します。その他のオプションもあります。

以下のオプションについて説明します。

[クラス名](#)

[宣言ファイル](#)[名前空間](#)[メンバー定義に別ファイルを使用する](#)

クラス名

[クラス名] フィールドにクラスの名前を入力します。

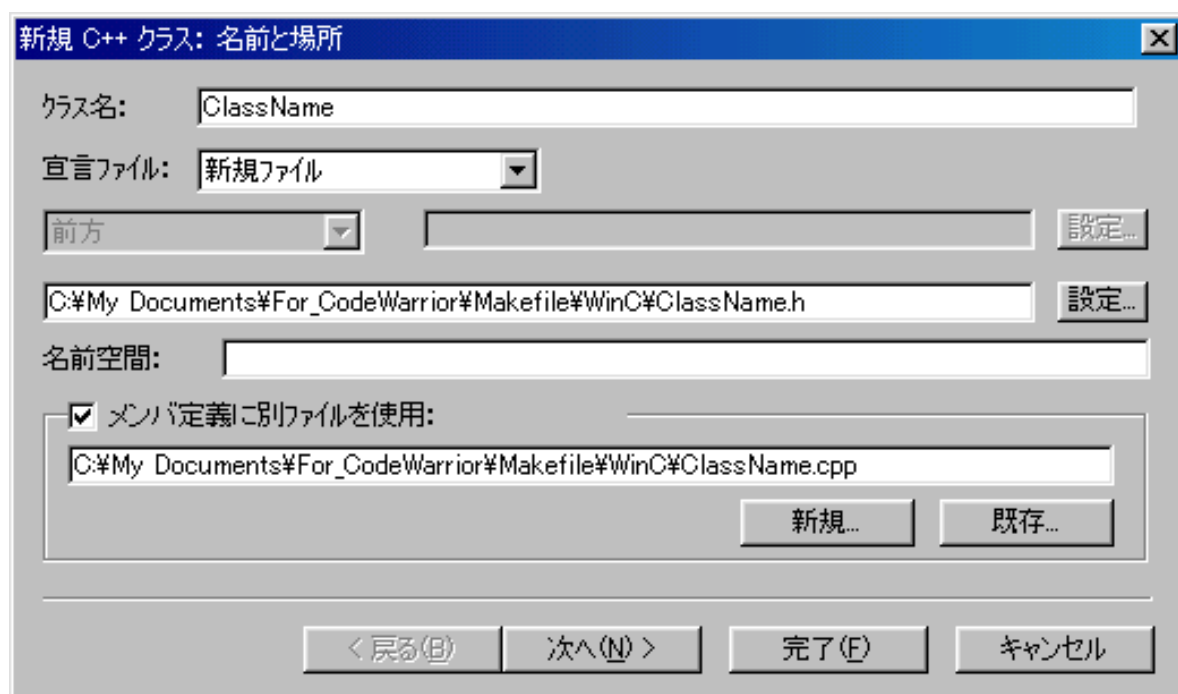
宣言ファイル

[宣言ファイル] ポップアップメニューで宣言ファイルの種類を指定します。

新規ファイル：宣言ファイル

クラス相対：プロジェクトの既存ファイルに依存する宣言ファイル

図 7.11 新規クラスウィザード：名前と場所



[宣言ファイル] ポップアップメニューで選択した項目によって、異なるオプションが表示されます。[図 7.11](#)は [新規ファイル] を、[図 7.12](#)は [クラス相対] を選択した場合のパネルです。

[新規ファイル] を選択した場合、空白のフィールドにファイルの保存先へのパスを入力します。代わりに [設定] ボタンをクリックして標準の「保存」ダイアログで保存先を選択することもできます。

[クラス相対] を選択した場合、ポップアップメニューと [クラス] フィールドが表示されます ([図 7.12](#))。[クラス] フィールドに新しいクラスの前になるクラスの名前を入力してください。代わりに [設定] ボタンをクリックしてウィンドウを開き、そこでクラスを選

択して[選択]ボタンで選択することもできます。次にポップアップメニューで新しいクラスを置く位置を指定します。[クラス]フィールドのクラスの[前方]か[後方]のいずれかを選択してください。

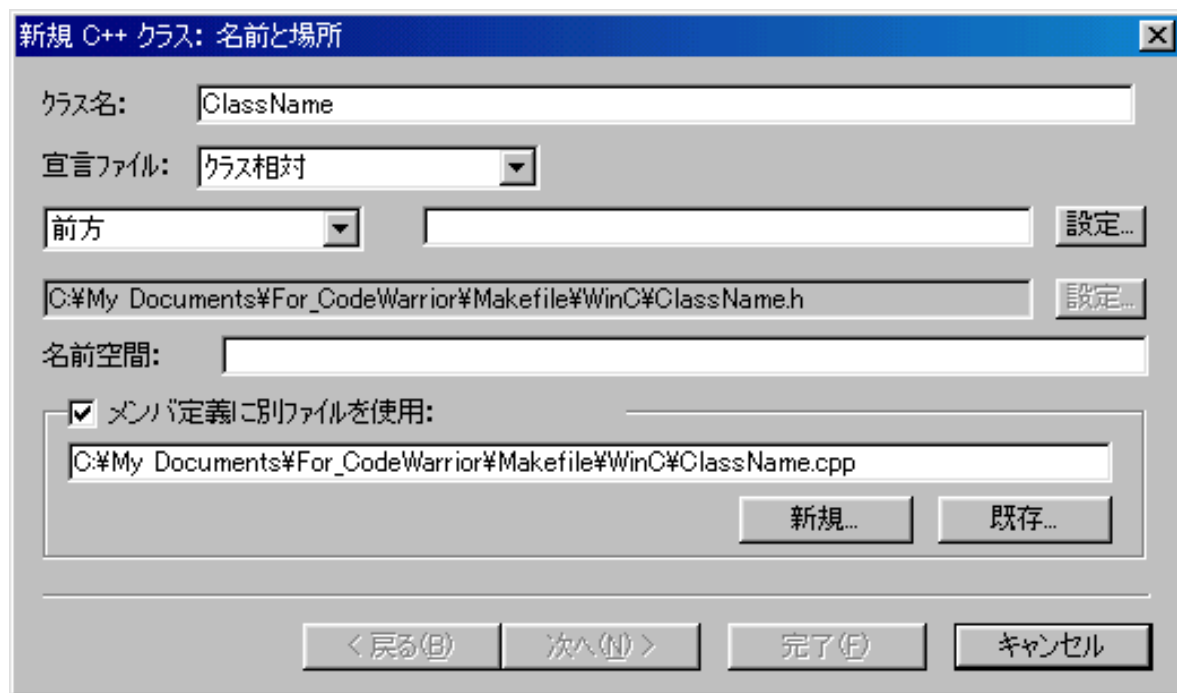
名前空間

[名前空間]フィールドに新クラスの名前空間を入力します。ユニークな名前空間を指定してください。

メンバー定義に別ファイルを使用する

[メンバー定義に別ファイルを使用する]チェックボックスがオンの場合、別のファイルを使って新クラスのメンバを定義できます。チェックボックスの下フィールドに別のファイルへのパスを入力します。[既存ファイル選択]ボタンをクリックして標準ダイアログからファイルを選択することもできます。新たに別のファイルを作るには、[新規ファイル選択]ボタンをクリックしてファイルを保存します。

図 7.12 [クラスに対応]を選択



2. 新クラスの基底クラスとメソッドを指定する

ウィザードのパネル(図 7.13)で新しいクラスの基底クラス、メソッドなどの情報を指定します。

以下のオプションについて説明します。

[基底クラス](#)

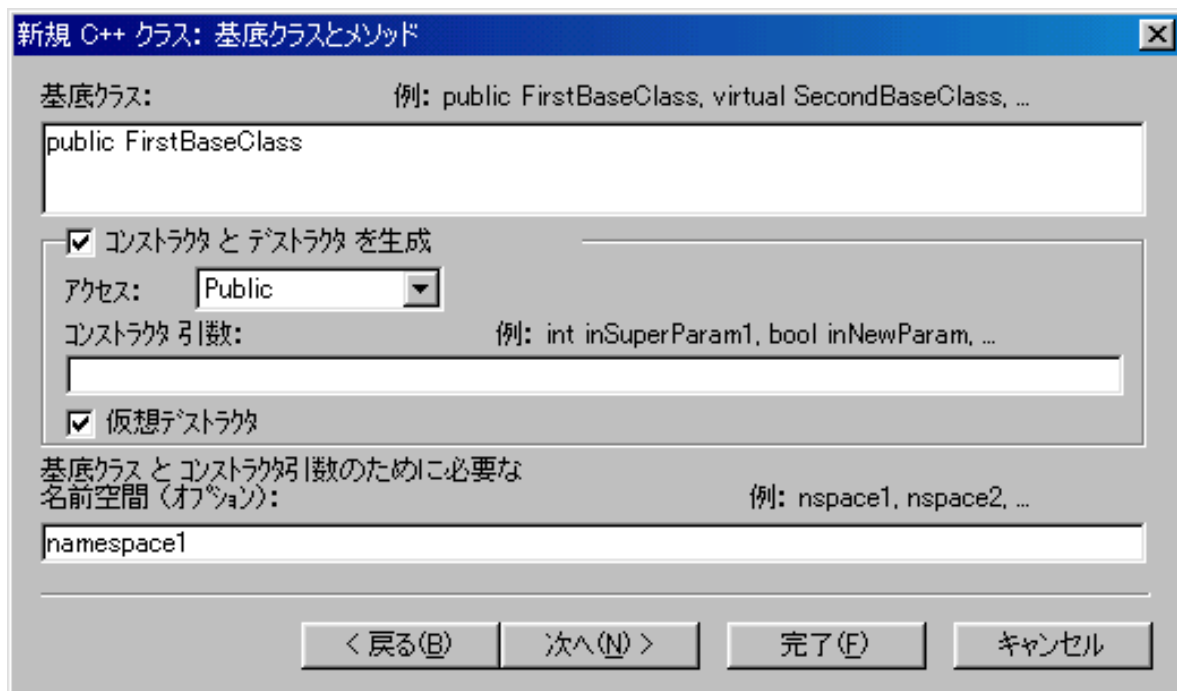
[コンストラクタとデストラクタを生成](#)

基底クラスとコンストラクタ引数に必要な名前クラス (オプション)

基底クラス

[基底クラス] フィールドに新クラスの基底クラスのリストを入力します。基底クラスのサンプルが表示されます。

図 7.13 新規クラスウィザード：基底クラスとメソッド



コンストラクタとデストラクタを生成

[コンストラクタをデストラクタを生成] チェックボックスがオンの場合、新クラスのコンストラクタ、デストラクタ関数を生成します。オンにすると以下のオプションが利用可能になります。

アクセス：ポップアップメニューから新クラスのアクセスタイプを選びます。アクセスタイプは [Public] [Protected] [Private] のいずれかです。

コンストラクタ引数：必要があればこのフィールドにコンストラクタの引数リストを入力します。引数のサンプルが表示されます。

Virtual デストラクタ：オンの場合、新クラスの仮想デストラクタ関数を作成できます。

基底クラスとコンストラクタ引数に必要な名前クラス (オプション)

[基底クラスとコンストラクタ引数に必要な名前クラス] フィールドに、[[基底クラス](#)] フィールドから基底クラスに必要なネームスペースを入力することができます。[[コンストラクタとデストラクタを生成](#)] 欄のコンストラクタのパラメータを入力することもできます。

3. 新クラス用の #include ファイルをリストする

[図 7.14](#) のパネルで新しいファイルの追加ヘッダ #include ファイルを指定します。

以下のコンテンツについて説明します。

[基底クラスのために自動的に追加されるインクルードファイル](#)

[追加ヘッダインクルードファイル](#)

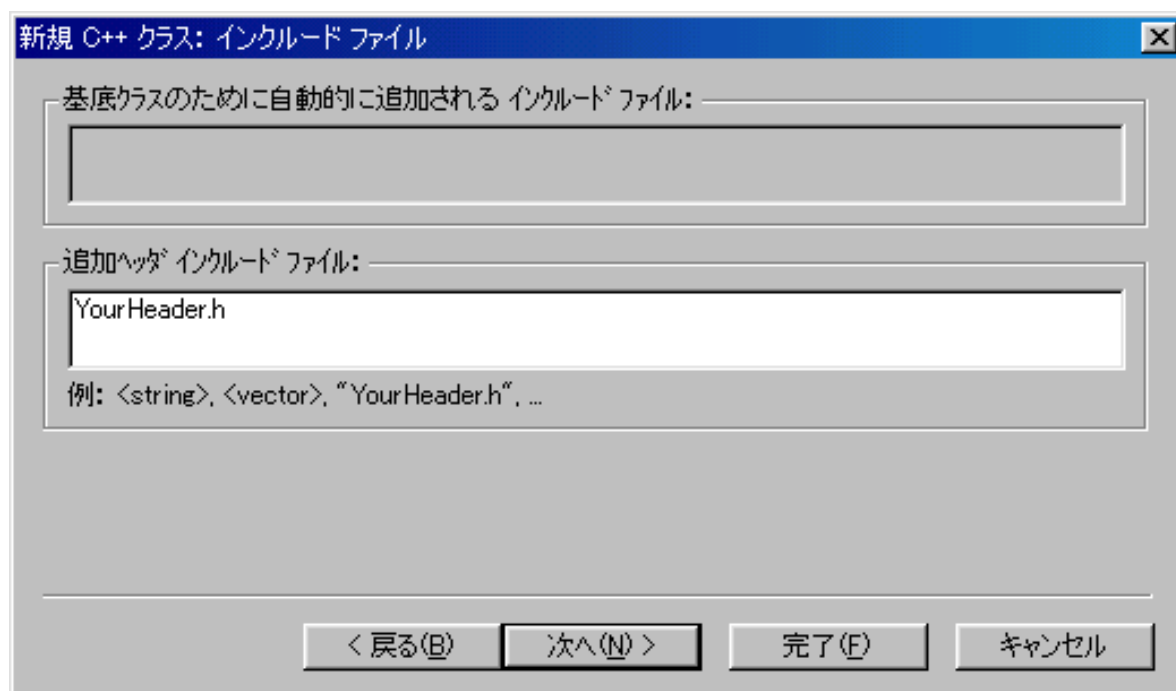
基底クラスのために自動的に追加されるインクルードファイル

[基底クラスのために自動的に追加されるインクルードファイル] フィールドは自動的に基底クラスへ追加される #include ファイルのリストを表示します。

追加ヘッダインクルードファイル

[追加ヘッダインクルードファイル] フィールドに新クラスのための #include ファイル ([基底クラスのために自動的に追加されるインクルードファイル] フィールド以外のファイル) のリストを表示します。リストはコンマで区切ります。サンプルのファイルが表示されます。

図 7.14 新規クラスウィザード：インクルードファイル



4. 新クラスをプロジェクトのビルドターゲットに割り当てる

[図 7.15](#) のパネルで新しいクラスをプロジェクトのビルドターゲットに割り当てます。

以下のオプションについて説明します。

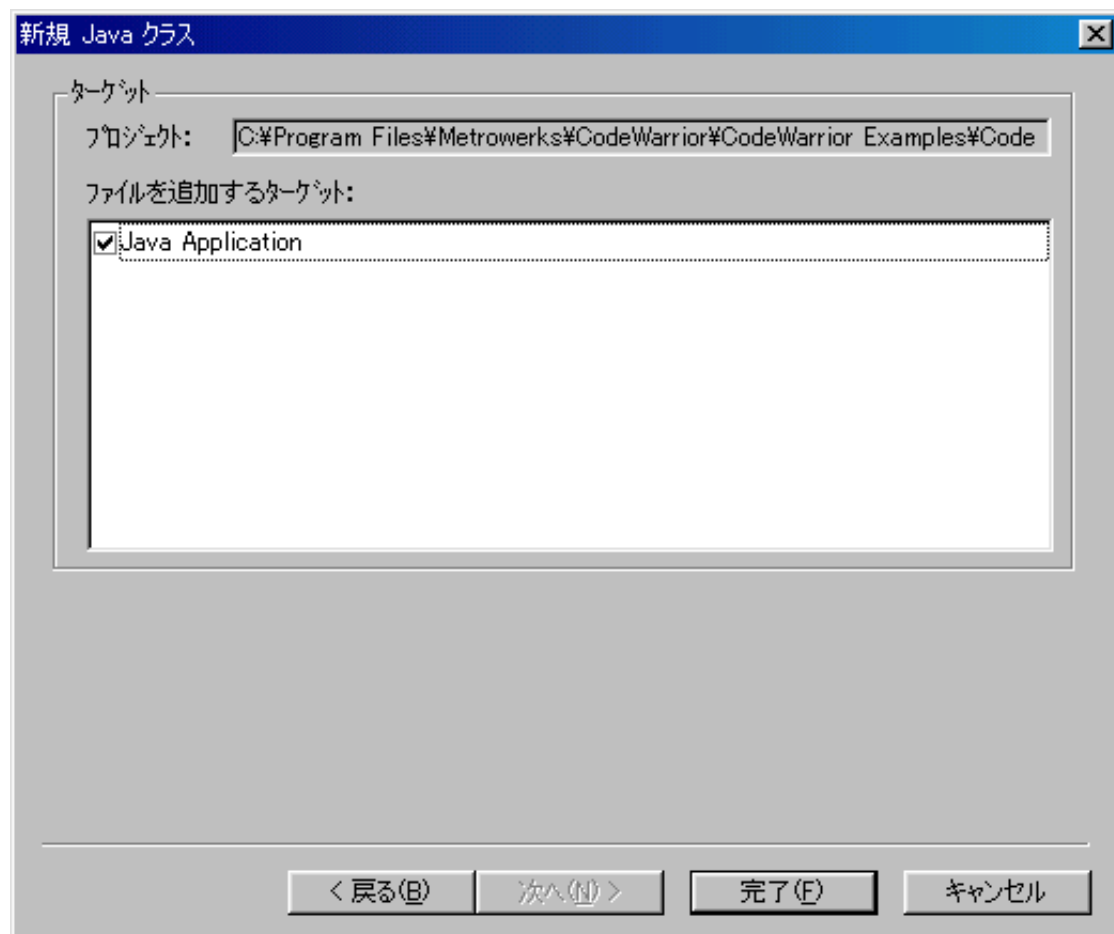
[プロジェクト](#)

ファイルを追加するターゲット

プロジェクト

[プロジェクト] フィールドは、[ファイルを追加するターゲット] フィールドに表示されているビルドターゲットを含むプロジェクトへのパスを表示します。

図 7.15 新規クラスウィザード：ターゲット



ファイルを追加するターゲット

[ファイルを追加するターゲット] 欄でクラスを割り当てるビルドターゲットの横のチェックボックスをオンにします。[図 7.15](#) では新クラスは [Java Application] ビルドターゲットに割り当てられます。

新規メンバー関数ウィザード

以下は新規メンバー関数ウィザードを使う手順です。

1. [新メンバー関数の宣言を記述する](#)
2. [新メンバー関数の位置を指定する](#)

図 7.16 新規メンバー関数ウィザード：メンバー関数宣言

新規 C++ メンバー関数: メンバー関数宣言

名前:
MemberFunctionName

戻り値型:
void

引数: 例: int inParam1, bool, char& outParam3, ...
int inParam1

引数のために必要な名前空間 (オプション): 例: namespace1, n2, ...
namespace1

修飾子:
アクセス: Public 指定子: None
☐ Inline ☐ Const

< 戻る(B) 次へ(N) > 完了(F) キャンセル

次の手順に従ってください。

1. 新メンバー関数の宣言を記述する

[図 7.16](#) のパネルで新しいメンバー関数の名前、戻り値の型、パラメータを指定します。その他のオプションもあります。

以下のオプションについて説明します。

[名前](#)

[戻り値型](#)

[引数](#)

[引数のために必要な名前空間 \(オプション\)](#)

[修飾子](#)

名前

[名前] フィールドにメンバー関数の名前を入力します。

戻り値型

[戻り値型] フィールドに関数の戻り値の型を入力します。

引数

必要があれば [引数] フィールドに関数のパラメータリストを入力します。サンプルのパラメータが表示されます。

引数のために必要な名前空間 (オプション)

[引数のために必要な名前空間] フィールドに [[引数](#)] フィールドのパラメータに必要な名前空間を入力します。サンプルのパラメータが表示されます。

修飾子

[修飾子] 欄の [アクセス] と [指定子] ポップアップメニューで新しいメソッドのアクセスレベルとメソッド指定子を選択します。アクセスレベルは [Public] [Protected] [Private] のいずれかです。指定子は [なし]、[Virtual]、[Pure Virtual]、[Static] のいずれかです。[Inline] [Const] チェックボックスも利用できます。

2. 新メンバ関数の位置を指定する

[図 7.17](#) のパネルで新しいメンバ関数のファイルの位置を指定します。

以下のオプションについて説明します。

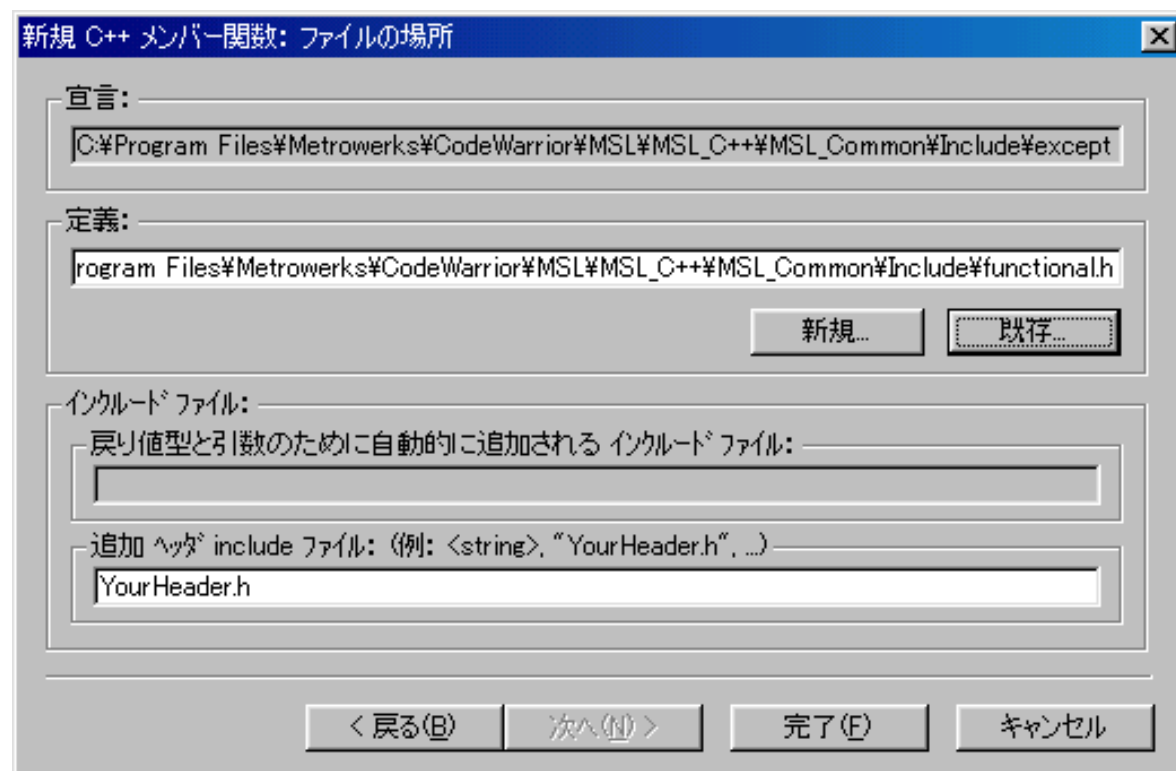
[宣言](#)

[定義](#)

[戻り値型と引数のために自動的に追加されるインクルードファイル](#)

[追加ヘッダ include ファイル](#)

図 7.17 新規メンバー関数ウィザード：メンバー関数ファイルの場所



宣言

[宣言] フィールドはメンバー関数の宣言が追加されるファイルの位置を表示します。

定義

[定義] フィールドにメンバー関数の定義に使われているファイルへのパスを入力します。
[既存] ボタンをクリックして標準ダイアログからファイルを選ぶこともできます。メンバー関数のために新しいファイルを作るには、[新規] ボタンをクリックしてファイルを保存してください。

戻り値型と引数のために自動的に追加されるインクルードファイル

[戻り値型と引数のために自動的に追加されるインクルードファイル] フィールドは自動的にメンバー関数に追加される `#include` ファイルのリストを表示します。前のステップで指定した戻り値の方とパラメータに基づいて、これらのファイルは自動的に追加されます。

追加ヘッダ include ファイル

[追加ヘッダ include ファイル] フィールドはメンバー関数に追加される `#include` ファイル ([戻り値型と引数のために自動的に追加されるインクルードファイル] フィールド以外のファイル) のリストを表示します。サンプルが表示されます。

新規データメンバーウィザード

以下は新規データメンバーウィザードを使うステップです

1. [新データメンバの宣言を指定する](#)
2. [新データメンバーのファイル位置を指定する](#)

次の手順に従ってください。

1. 新データメンバの宣言を指定する

[図 7.16](#) のパネルで新しいデータメンバの名前、型、初期設定子を指定します。その他のオプションもあります。

以下のオプションについて説明します。

[名前](#)

[型](#)

[型のために必要な namespace](#)

[初期設定子](#)

[修飾子](#)

図 7.18 新規データメンバーウィザード：データメンバー宣言

新規 C++ データメンバー: データメンバー宣言

名前:
DataMembername

型:
void

型のために必要な namespace (オプション):
std 例: std

初期設定子:
100 例: 100 または inConstructorParameterName

修飾子:
アクセス: Protected 指定子: None
☐ Const ☐ Volatile

< 戻る(B) 次へ(N) > 完了(F) キャンセル

名前

[名前] フィールドにデータメンバの名前を入力します。

型

[型] フィールドにデータメンバの型を入力します。

型のために必要な namespace

[型のために必要な namespace] フィールドに、[型] フィールドのデータメンバ型に必要な名前空間のリストを入力します。サンプルのリストが表示されます。

初期設定子

[初期設定子] フィールドにデータメンバーの初期値を入力します。サンプルの初期設定子が表示されます。

修飾子

[修飾子] 欄の [アクセス] と [指定子] ポップアップメニューで新しいデータメンバーのアクセスレベルとメソッド指定子を選択します。アクセスレベルは [Public] [Protected] [Private] のいずれかです。指定子は [なし] [Static] [Mutable] のいずれかです。[Const] [Volatile] チェックボックスも利用できます。

2. 新データメンバーのファイル位置を指定する

[図 7.19](#) のパネルで新しいメンバー関数に関連するファイルの位置を指定します。

以下のオプションについて説明します。

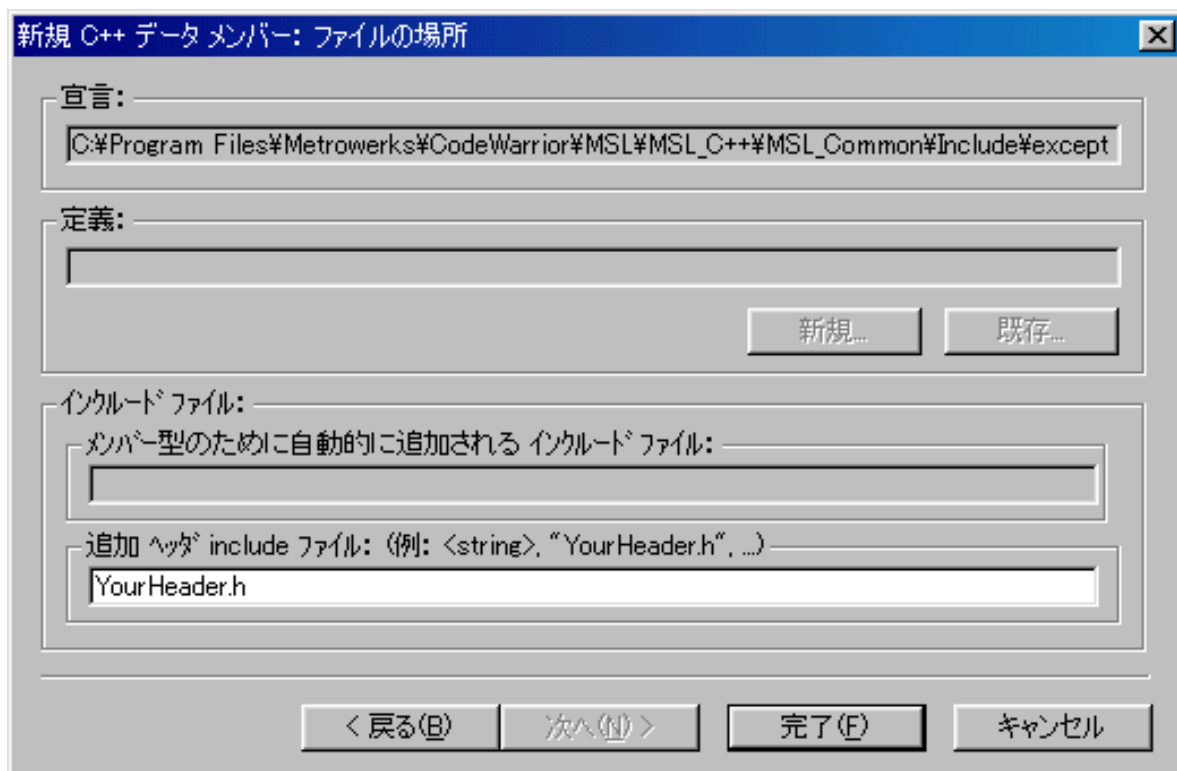
[宣言](#)

[定義](#)

[メンバー型のために自動的に追加されるインクルードファイル](#)

[追加ヘッダ include ファイル](#)

図 7.19 新規データメンバーウィザード：ファイルの場所



宣言

[宣言] フィールドはデータメンバーの宣言が追加されるファイルの場所を示します。

定義

[定義] フィールドは利用できません。

メンバー型のために自動的に追加されるインクルードファイル

[メンバー型のために自動的に追加されるインクルードファイル] フィールドはデータメンバー型に自動的に追加される `#include` ファイルのリストを示します。

追加ヘッダ include ファイル

[追加ヘッダ include ファイル] フィールドはデータメンバー型に追加される `#include` ファイル([宣言] フィールド以外のファイル)のリストを示します。サンプルが表示されます。

ブラウザを使う

ブラウザは、コードに関連するデータを表示する方法をいくつか提供します。ここではブラウザの基本的な動作と一般的な作業を行うための基本的な技法を説明します。

ここでは、以下の項目を説明します。

[ブラウザのオプションを設定](#)

[ブラウザデータベースでシンボルを識別](#)

[ブラウザ内のコードをナビゲート](#)

[サブプロジェクトをブラウズ](#)

[シンボルを補完](#)

[ソースファイルを開く](#)

[宣言を表示](#)

[関数定義を表示](#)

[ブラウザ内でコードを編集](#)

[継承を分析](#)

[オーバーライドされた関数を検索](#)

[MFC、PowerPlant クラスを表示](#)

[デフォルトのブラウザを保存](#)

ブラウザのオプションを設定

ブラウザ関連のメニュー項目およびブラウザ固有のオプションは、ブラウザを利用すると使用可能になります。ブラウザを利用する方法は、[「ブラウザを利用」\(p143\)](#) を参照してください。

ブラウザが利用できるとそれに関連するコマンドが選択可能になります。ブラウザ関連のコマンドは、[ウィンドウメニュー](#)の[[ブラウザコンテンツ](#)] [[クラス階層ウィンドウ](#)] [[新規クラスブラウザ](#)] です。

ヒント：ブラウザが利用できることを確かめる簡単な方法は、[ウィンドウメニュー](#)のコマンドを見ることです。ブラウザ関連のコマンドが使用可能であれば、ブラウザは利用可能になっています。

また、ブラウザに関連するグローバルな IDE オプションもあります。ブラウザウィンドウで使う項目の色などをコントロールできます。

ブラウザウィンドウでサブプロジェクトから項目を追加する方法は [「サブプロジェクトをキャッシュする」\(p236\)](#) を参照してください。

各種の設定の詳細は [「ブラウザ表示」\(p193\)](#) を参照してください。

ブラウザデータベースでシンボルを識別

[ブラウザのコンテキストメニュー](#)よりも簡単に、ブラウザデータベースにシンボルがあるか否かを確認する方法があります。ブラウザカラーリングを使う方法です。ブラウザが利用可能ならば、ブラウザデータベースにあるシンボルは指定した色で表示されます。詳細は [「ブラウザ表示」\(p193\)](#) を参照してください。

ヒント： デフォルトの設定では、ブラウザデータベースの 8 種類のシンボルの色はすべて同じです。シンボルの種類ごとに違う色を設定できます。シンタックスカラーを使っている場合、シンボルの色は 1 ~ 2 色だけにすると方がブラウザデータベースで識別しやすくなります。

ブラウザ内のコードをナビゲート

ブラウザでコード内を移動するには、多くの方法があります。

コンテキストメニューを使う

もっとも強力で柔軟な方法は[ブラウザのコンテキストメニュー](#)です。ブラウザデータベースに情報があるシンボルを右ボタンクリックすると、このメニューが現れます。ここには、クラス名、関数名、大域変数、クラスデータメンバーなどが表示されます。詳細は、「[ブラウザのコンテキストメニュー](#)」(p148)を参照してください。

[ブラウザウィンドウ](#)でクラス、関数、データメンバーを選択するだけで、ウィンドウの[ソース欄](#)に関連するコードが表示されます。

戻ると次へ

ブラウザは、[検索メニュー](#)の[[戻る](#)]と[[次へ](#)]をサポートします。見ているビュー、ウィンドウ、またはコードにかかわらず、前に見ていたビューに戻ることができます。

これまでに行った一連の変更をさかのぼったり、進んだりできます。例えば、ブラウザでプロジェクトを見てファイルに変更を加えたとします。その後、ファイルを切り替えてさらに変更したとします。これを数回行ったと考えてください。実行した操作を複数回戻るには[[戻る](#)]コマンドを使います。ファイルに変更を行わなかったとしても、ファイルまたはそのクラスやメソッドを見たという操作に戻ることができます。同様に、いったん戻ると、[[次へ](#)]コマンドを使って始めたところまで戻ることができます。

[[戻る](#)]と[[次へ](#)]のコマンドを CodeWarrior のツールバーに追加すると、関連するポップアップメニューを使うことができます([図 7.20](#))。[[戻る](#)]と[[次へ](#)]アイコンをクリックし続けるとポップアップメニューが現れます。

図 7.20 ツールバーの[戻る]と[次へ]ボタン



メニュー内の項目を選択すると、その操作にジャンプします。シーケンス外の操作を選ぶと、CodeWarrior は以前のアクションを通過しないで、その操作にジャンプします。

注意：[戻る] および [次へ] は、実行した操作を取り消すものではありません。ブラウザウィンドウで作業した特定の位置へ移動するための、より柔軟な方法です。

サブプロジェクトをブラウズ

通常 IDE で表示するのはカレントターゲットのブラウザだけです。カレントターゲットのサブプロジェクトの情報もブラウザビューに追加するには、サブプロジェクトのキャッシングをオンにします。[「サブプロジェクトをキャッシュする」\(p236\)](#) を参照してください。

シンボルを補完

IDE には、ブラウザが情報を持つアイテム名が入力されたとき、その名をユーザーの代わりに補完するコマンドがあります。[指定語で始まるシンボルを検索] [指定語を含むシンボルを検索] [次のシンボル] [前のシンボル] などのキーボードコマンドを使って、ソースファイルで入力または選択したテキストに一致するブラウザアイテムを検索して選択することができます。これらのコマンドは IDE のメニューにはなく、キーボードからのみ使用できます。[「エディタ」\(p486\)](#) を参照してください。

選択または入力したテキストと一致するブラウザアイテムの名前を入力するには、[指定語を含むシンボルを検索] キーボードコマンドを使います。最初の数文字が同じブラウザアイテムの名前を入力するには、[指定語で始まるシンボルを検索] キーボードコマンドを使います。

[指定語を含むシンボルを検索] [指定語で始まるシンボルを検索] キーボードコマンドを使った後に [次のシンボル]、[前のシンボル] を使うと、選択または入力したテキストに一致するブラウザシンボルを検索します。

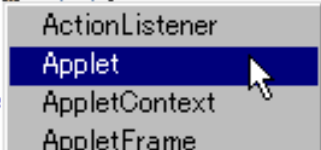
入力したいブラウザアイテムを見つけたら、キーで挿入位置を指定して入力が続けます。

ブラウザアイテムを検索して入力するには、テキストの最初の数文字を選択して右ボタンクリックする方法があります。これで一致する項目を表示するコンテキストメニューが現れます ([図 7.21](#))。このメニューで選択した項目が、選択していたテキストを置換します。詳細は [「ブラウザのコンテキストメニュー」\(p148\)](#) を参照してください。

図 7.21 選択したテキストを自動的に置換する

実行前

```
java.awt.*;  
java.applet.*;  
  
class Life extends Applet {  
    ...  
}
```



実行後

```
import java.awt.*;  
import java.applet.Applet;  
  
public class Life extends Applet {  
    ...  
}
```


ソースファイルを開く

ソースファイルを開く方法はいくつかあります。

[ブラウザウィンドウ](#)の[ソース欄](#)に開きたいファイルが表示されているときに、[ファイルボタン](#)をクリックします。詳細は「[ファイルボタン](#)」(p156)を参照してください。

ファイル内で使われているシンボルをクリックして選択します。次に[ブラウザのコンテキストメニュー](#)から望みのファイルを開くか、または関数を表示します。

ソースファイルを見ているときにヘッダファイルを見たい場合、またはその逆の場合は、Ctrl + ` キーを押すと関連ファイルを見ることができます。

宣言を表示

宣言を見る方法もいくつかあります。この方法は、調べているシンボルの種類によって変わります。

[ブラウザウィンドウ](#)でクラス名またはデータメンバー名を選択すると、宣言が[ソース欄](#)に表示されます。宣言を含むファイルを開くには、その名前をダブルクリックします。関数名を選択またはダブルクリックすると、その定義を見ることができます。

ブラウザウィンドウの[クラス宣言ボタン](#)をクリックすると、[ソース欄](#)でクラス宣言を見ることができます。

ウィンドウ内の名前を右ボタンクリックすると、[ブラウザのコンテキストメニュー](#)を使って宣言を開くことができます。この方法で関数宣言も表示できます。

関数定義を表示

関数定義を見る方法もいくつかあります。

[ブラウザウィンドウ](#)の[メンバー関数欄](#)で関数を選択します。定義が[ソース欄](#)に表示されます。宣言を含むファイルを開くには、[メンバー関数欄](#)でその名前をダブルクリックします。

エディタまたはブラウザウィンドウで関数名を右ボタンクリックして、表示される[ブラウザのコンテキストメニュー](#)を使って、特定の関数定義にジャンプします。

[検索メニュー](#)の[戻る]を選択します。

任意のソースビューでルーチン名を Alt + ダブルクリックまたは Ctrl + ダブルクリックすると、ルーチンのすべての実装をリストした[シンボルウィンドウ](#)を開きます。

[クラス階層ウィンドウ](#)または[単一クラス階層ウィンドウ](#)で関数名を右ボタンクリックして、表示される[ブラウザのコンテキストメニュー](#)を使って、特定の関数定義にジャンプします。

ブラウザ内でコードを編集

[ソース欄](#)に表示されたコードはどれでも編集することができます。編集したい定義を探します。コードが[ソース欄](#)に表示されたら、CodeWarrior エディタと同じ操作で編集できます。

CodeWarrior エディタの詳細は「[ソースコードの編集](#)」(p95) を参照してください。

継承を分析

コードの継承関係を分析するには、[クラス階層ウィンドウ](#)または[単一クラス階層ウィンドウ](#)を使います。クラス名の左の小さな拡張ボタンはクラスに複数の先祖があることを示します。[先祖クラスポップアップメニュー](#)を使って先祖クラスにジャンプし、そこで先祖や子孫を調べます。[階層拡張ボタン](#)でサブクラスの拡張 / 縮小表示ができます。

詳細は「[クラス階層ウィンドウ](#)」(p156) を参照してください。

オーバーライドされた関数を検索

オーバーライドされた関数を検索するには、[ブラウザウィンドウ](#)のツールバーの[Inherited を表示] チェックボックスをオフにします。これで変更していない継承された関数を隠すことができます。オーバーライドした関数は隠すことができません。

その後、[識別アイコン](#)を使って、「virtual」とマークされた関数を探します。これらの関数のほとんどは継承する関数をオーバーライドして使います。ここには、このクラスで宣言され、先祖から継承していない関数も表示されています。

多重定義を持つ関数の[シンボルウィンドウ](#)を開くには、関数名を右ボタンクリックして、コンテキストメニューから [関数定義をすべて検索] を選択します。階層ビューと組み合わせ、関数をオーバーライドするものを見つけることができ、それがクラス階層のどこにあるかを知ることができます。

MFC、PowerPlant クラスを表示

ブラウザでフレームワーククラス (MFC クラス) を表示できます。ブラウザでフレームワーククラスを表示するには、コンパイラが認識できるようにフレームワークヘッダをインクルードする必要があります。

プリコンパイルヘッダを使うと、コンパイル時間を短縮できます。プリコンパイルヘッダを直接使うだけでは、コンパイラはフレームワーククラスを認識できず、シンボルデータベースの情報を生成できません。

この問題を解決するには、プリコンパイルヘッダを作成するソースファイルをコピーして名前を変更して、それをプロジェクトに追加します。ソースファイルにはプリコンパイルヘッダの拡張子 (.pch や .pch++) を付けます。他に、プリコンパイルヘッダを作るプロジェクトをサブプロジェクトとして追加する方法もあります。

これで、プロジェクト固有のプリコンパイルされたヘッダを作成できます。CodeWarrior はプロジェクト内でプリコンパイルされたヘッダをビルドして、フレームワークシンボルをコンパイラに渡します。これによりブラウザデータベース用の情報を生成できます。

名前を変更したコピーを追加する理由は、同じプリコンパイルヘッダを複数のプロジェクトで使うときの問題を回避するためです。1 つのプロジェクトがヘッダを更新すると、このファイルは他のすべてのプロジェクトでも変更されたとマークされ、プリコンパイルヘッ

ダの再ビルドが必要になります。これでは、プリコンパイルヘッダを使うメリットがなくなります。

デフォルトのブラウザを保存

ブラウザウィンドウの構成を変更できます。変更した設定をデフォルトとして保存することができます。

ブラウザウィンドウを好みに合わせて設定します。例えばブラウザウィンドウの各欄のサイズやウィンドウ全体のサイズと位置を設定します。その後、[ウィンドウメニュー](#)の[[ウィンドウの設定を保存](#)]を選択します。次回、複数クラスブラウザウィンドウを開くと、設定したとおりの構成のウィンドウが開きます。

他のブラウザウィンドウでも同様に設定を保存できます。各ウィンドウの設定は、個別に、それぞれのウィンドウをアクティブにして保存する必要があります。

エディタウィンドウの保存については「[エディタウィンドウの設定を保存](#)」(p104) を参照してください。

第 8 章 IDE のオプション設定

この章では、CodeWarrior IDE (Integrated Development Environment : 統合開発環境) の「 IDE 設定」ダイアログ、およびツールバーについて説明します。

「 IDE 設定」ダイアログで CodeWarrior IDE の機能をカスタマイズします。IDE 全体の環境を設定するダイアログです。

「 IDE 設定」ダイアログを開くには、[編集メニュー](#)の [[IDE 設定](#)] を選択します。エディタのフォントやタブの設定など、機能ごとに分類された IDE 設定パネルがあります。

IDE のツールバーもカスタマイズ可能です。独自のツールバーを設定することにより作業を効率化できます。

以下の内容について説明します。

[IDE 設定パネルの解説](#)

[IDE 設定パネルを選択](#)

[IDE をカスタマイズ](#)

IDE 設定パネルの解説

CodeWarrior IDE の「 IDE 設定」ダイアログは編集メニューの [[IDE 設定](#)] で表示されます。

以下の内容について説明します。

[IDE 設定パネル](#)

[ダイアログのボタン](#)

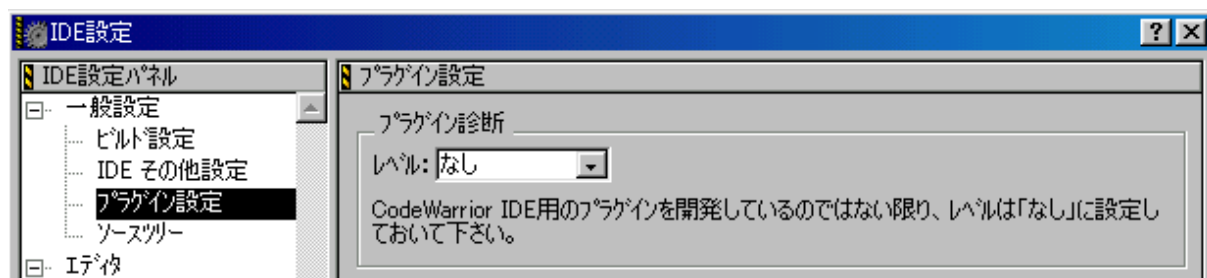
IDE 設定パネル

「 IDE 設定」ダイアログの左側に IDE 設定パネルの階層リストがあります。ここで選択した IDE 設定パネルがダイアログの右側に表示されます。実際に使用可能な設定パネルは CodeWarrior 製品により異なります。

IDE 設定パネルのオプションは、IDE 全体、およびすべてのプロジェクトに適用されます。表示したい IDE 設定パネルをリストで選択します。矢印キーまたはクリックで選択できます。[図 8.1](#) に「 IDE 設定」ダイアログを示します。

各パネルにはユーザーが設定するオプションがあります。オプション設定を変更して保存、変更を破棄、または元の状態に復帰することもできます。詳細は「[ダイアログのボタン](#)」([p182](#)) を参照してください。

図 8.1 「IDE 設定」ダイアログでパネルを選択



ダイアログのボタン

「IDE 設定」ダイアログには、オプションの使用方法与適用方法をコントロールするボタンがあります。

以下の内容について説明します。

[保存しない](#)

[出荷時設定](#)

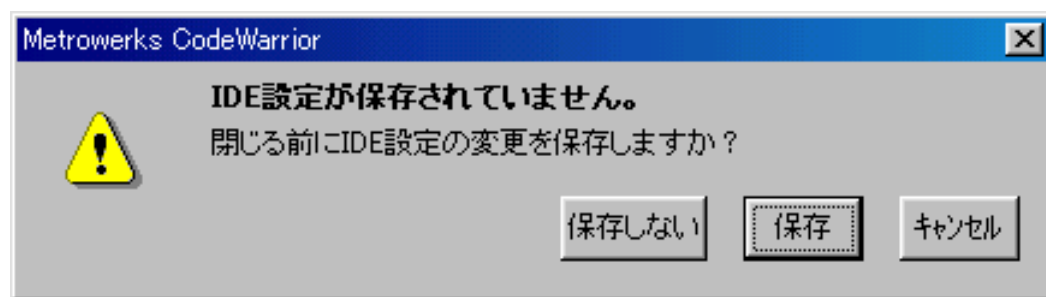
[復帰](#)

[保存](#)

保存しない

「IDE 設定」ダイアログでオプション設定を変更した後にダイアログを閉じようとする、警告ダイアログ(図 8.2)が表示されます。変更を破棄するには [保存しない] ボタンをクリックします。変更を保存するには [保存] ボタンをクリックします。変更を保存せずに「IDE 設定」ダイアログへ戻るには [キャンセル] ボタンをクリックします。

図 8.2 IDE 設定の確認ダイアログ



出荷時設定

[出荷時設定] ボタンをクリックすると、IDE 設定パネルのオプションがデフォルトの設定に戻ります。他の IDE 設定パネルのオプションには影響しません。現在表示中のパネルのオプションの設定だけがリセットされます。

復帰

[復帰] ボタンは、現在表示中の IDE 設定パネルの状態を、そのパネルを最後に保存したときの設定に戻します。これは、パネルに変更を加えた後で変更を適用したくない場合に便利です。

保存

[保存] ボタンは、任意の設定パネル内で加えた変更を保存します。「IDE 設定」ダイアログを閉じた後、CodeWarrior IDE の挙動が変わります。

IDE 設定パネルを選択

ここでは、IDE 全体の環境の設定方法を説明します。デバッガ、エディタなどのオプションを設定する方法を学習します。

IDE 設定パネルの選択方法は「[IDE 設定パネルの解説](#)」(p181) を参照してください。

ここでは IDE 設定でコントロールする IDE の機能について説明します。以下の内容について説明します。

[一般 IDE 設定](#)

[エディタの IDE 設定](#)

[デバッガの IDE 設定](#)

一般 IDE 設定

ここでは IDE の一般的な機能をコントロールする IDE 設定パネルについて説明します。

[ビルド設定](#)

[IDE その他設定](#)

[プラグイン設定](#)

[ソースツリー](#)

ビルド設定

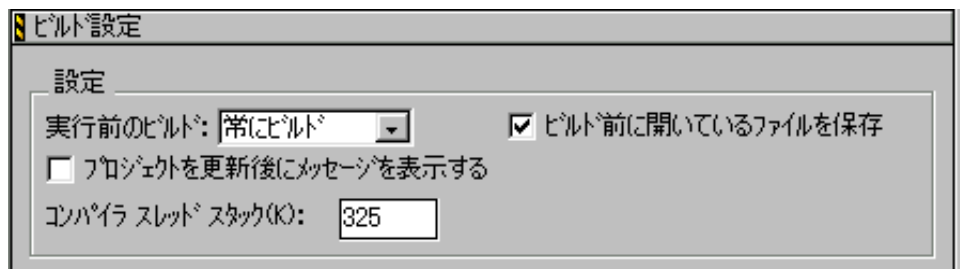
ビルド設定パネル ([図 8.3](#)) でプロジェクトのビルド方法を設定します。

IDE 設定とビルド設定パネルについては「[IDE 設定パネルの解説](#)」(p181) を参照してください。

実行前のビルド

[実行前のビルド] ポップアップメニューでプロジェクトを実行する前にビルドを行うか否かを選択します。[常にビルド] [確認する] [ビルドしない] の選択肢があります。

図 8.3 ビルド設定パネル



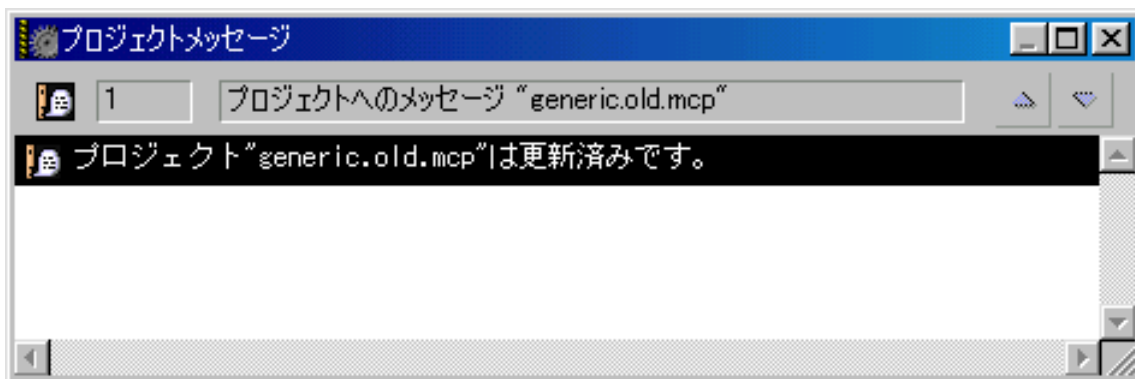
ビルド前に開いているファイルを保存

[ビルド前に開いているファイルを保存]チェックボックスがオンの場合、[[プリプロセス](#)] [[プリコンパイル](#)] [[コンパイル](#)] [[逆アセンブル](#)] [[最新状態に更新する](#)] [[メイク](#)] [[実行](#)] コマンドを実行する前に、開いているファイルを自動的に保存します。

プロジェクトを更新後にメッセージを表示する

[プロジェクトを更新後にメッセージを表示する]チェックボックスがオンの場合、IDE は最新のプロジェクトをビルドした後にメッセージを表示します。[図 8.4](#) のように、メッセージウィンドウに表示されます。最新プロジェクトについては「[プロジェクトを更新](#)」(p258)を参照してください。

図 8.4 最新プロジェクトをビルドした後のメッセージ



コンパイラスレッドスタック

[コンパイラスレッドスタック] フィールド ([図 8.3](#)) でコンパイル、リンクのスレッドサポートに割り当てられるスタックサイズの上限を指定します。

CodeWarrior でのビルドはスレッド化されており、コンパイル、リンクはメインアプリケーションのスレッドとは別のスレッドで行われます。このオプションでスレッドのスタックサイズを指定します。

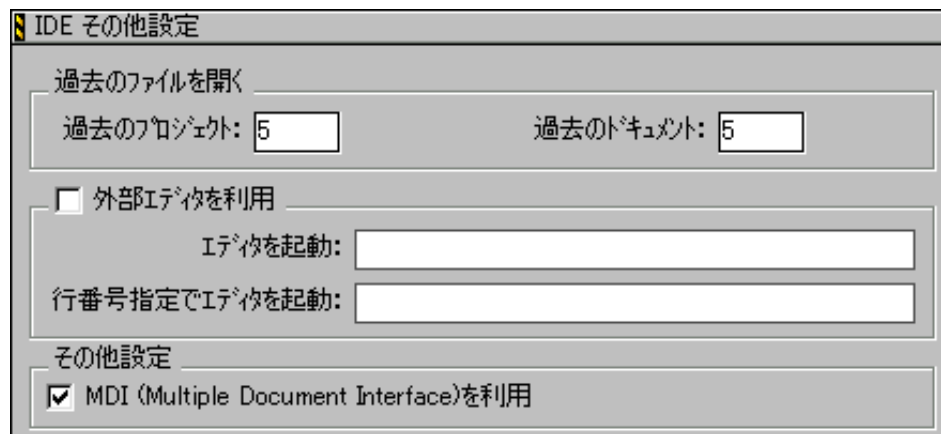
通常このオプションは変更しませんが、非常に複雑なプロジェクトではコンパイラのクラッシュを避けるために値を増やしてください。

IDE その他設定

IDE その他設定パネル([図 8.5](#))で、最近開いたプロジェクトやテキストファイルの記憶や、サードパーティエディタを設定します。

「IDE 設定」ダイアログと IDE その他設定パネルを開く方法は「[IDE 設定パネルの解説](#)」(p181)を参照してください。

図 8.5 IDE その他設定パネル



過去のプロジェクト

[過去のプロジェクト] フィールドに、[ファイルメニュー](#)の[[過去のファイルを開く](#)]に表示する最近開いたプロジェクトの最大数を入力します。

過去のドキュメント

[過去のドキュメント] フィールドに、[ファイルメニュー](#)の[[過去のファイルを開く](#)]に表示する最近開いたドキュメントの最大数を入力します。

外部エディタを利用

[外部エディタを利用] オプションで、テキストファイルを CodeWarrior エディタで開くか否かを指定します。オフの場合、CodeWarrior エディタがデフォルトのエディタになります。オンの場合、サードパーティのテキストエディタ([エディタを起動] フィールドで指定する)を使ってファイルを表示します。

サードパーティエディタを起動するためのコマンドラインが2つあります。第1は[エディタを起動]で、ここで IDE が起動するテキストエディタを指定します。第2の[行番号指定でエディタを起動]では、テキストエディタを起動したときにジャンプするテキストラインを指定します。例えばエラーメッセージの原因となったテキストを表示するためにエラーメッセージをダブルクリックしたときに、IDE はこの第2のコマンドラインを呼び出します。

コマンドラインでは2つの変数(%file と %line)を使うことができます。IDE はこの変数を発見したとき、%file を完全なファイルパスへ拡張し、%line をファイルの第1行へ拡張します。例えば、Emacs テキストエディタでファイルを編集する場合、以下のコマンドラインを[エディタを起動]フィールドに入力します。

```
runemacs %file
```

テキストファイルの特定の行へジャンプするには、以下のコマンドラインを [行番号指定でエディタを起動] フィールドに入力します。

```
runemacs +%line %file
```

外部テキストエディタとコマンドラインについてはエディタに付属のマニュアルを参照してください。

MDI 使用

[MDI(Multiple Document Interface) を利用] チェックボックスで CodeWarrior IDE で使用する Windows インターフェースを指定します。MDI (Multiple Document Interface) または FDI (Floating Document Interface) を使用できます。

プラグイン設定

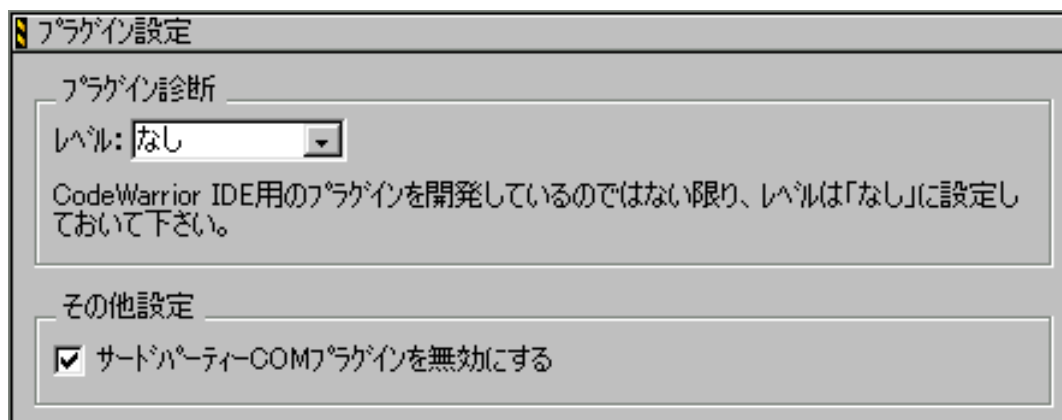
プラグイン設定パネル([図 8.6](#))にサードパーティの IDE プラグインを調べるオプションがあります。

「IDE 設定」ダイアログからプラグイン設定パネルを開く方法は「[IDE 設定パネルの解説](#)」(p181)を参照してください。

プラグイン診断

CodeWarrior 用プラグインを開発している場合、[プラグイン診断] オプションで詳しい情報を表示することができます。プラグインが正確に機能しない、またはインストールしたプラグインの属性を知りたい場合、このオプションを利用します。

図 8.6 プラグイン設定パネル



[プラグイン診断] には 3 つのレベルがあります。

なし：デフォルト設定。プラグイン診断を行わず、出力もしない。

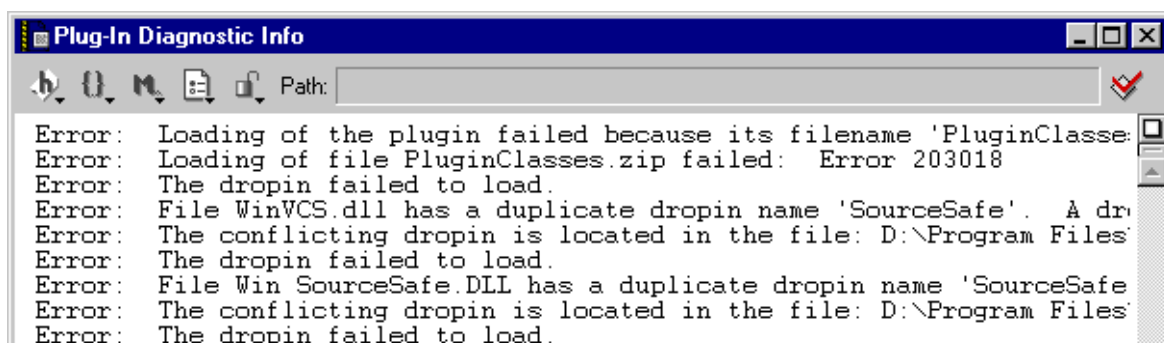
エラーのみ：プラグインのロード中に発生したエラーを報告する。IDE の起動後にテキストで表示される ([図 8.7](#))。

すべての情報：各プラグインの詳細情報を報告する。ロード中のエラー、追加的プラグイン情報、プラグインの属性等が表示される。IDE の起動後にテキストで表示される（図 8.8）。このテキストにはインストールされたプラグインとその IDE 設定パネル、コンパイラ、リンカの情報も含まれる。

[レベル] ポップアップメニューからプラグイン診断のレベルを選択します。

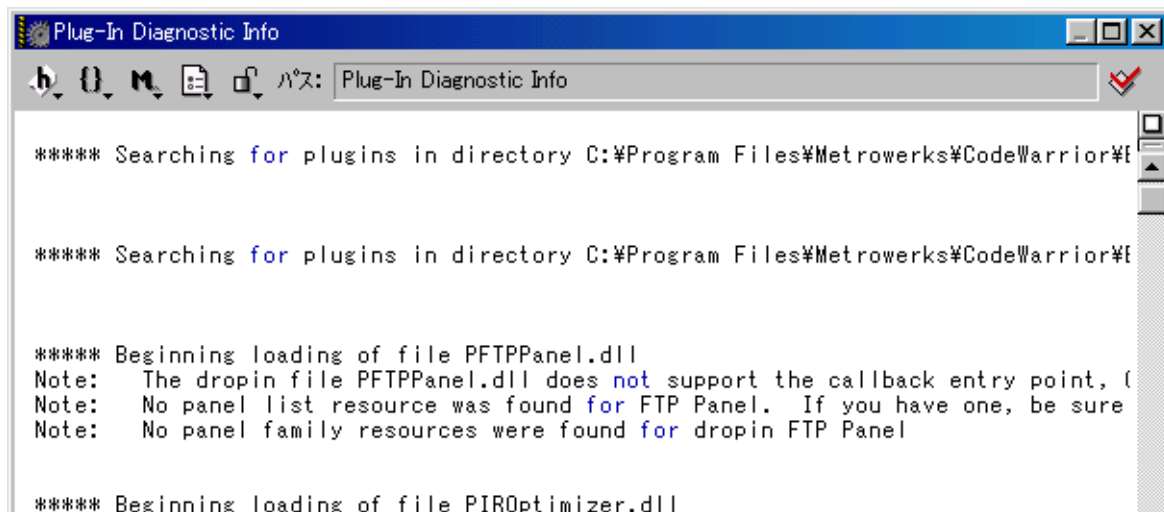
注意： プラグインの診断レベルを変更した場合、CodeWarrior の再起動を要求するダイアログが表示されます。必ず CodeWarrior を再起動してください。

図 8.7 プラグイン診断（エラーのみ）



生成されたテキストファイルを保存、および印刷することができます。これらをプラグインのエラーリファレンスとしても利用できます。CodeWarrior は一般的なプラグインエラーの修正方法をテキストファイルに提供します。

図 8.8 プラグイン診断（すべての情報）



サードパーティ COM プラグインを無効にする

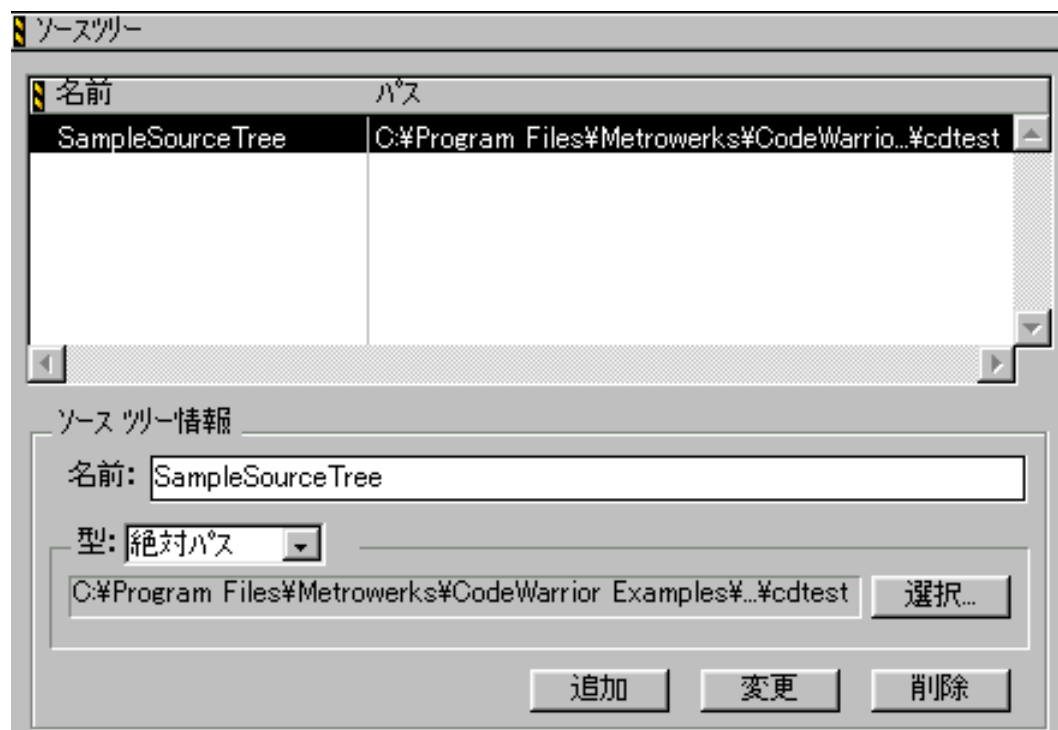
[サードパーティ COM プラグインを無効にする] チェックボックスがオンの場合、サードパーティの COM (Common Object Model) プラグインが無効になります。IDE で問題を解決するときに便利です。

ヒント： IDE の使用中に問題が起きた場合、サードパーティのプラグインを無効にしてください。プラグインを無効にした後に問題が起きなければ、CodeWarrior のプラグインとサードパーティのプラグインの衝突が原因かもしれません。

ソースツリー

ソースツリー設定パネル([図 8.9](#))でプロジェクトのグローバルなソースツリー(ルートパス)を定義します。プロジェクトのアクセスパスとビルドターゲットの出力を設定することができます。これによりさまざまなホストでプロジェクトを共有できます。ソースツリーのパスを少し変更するだけでプロジェクトの機能を保持できます。

図 8.9 ソースツリー設定パネル



「IDE 設定」ダイアログとソースツリー設定パネルを開く方法は [「IDE 設定パネルの解説」\(p181\)](#) を参照してください。

ターゲット特有のソースツリー設定パネルもあります。詳細は [「ソースツリー」\(p242\)](#) を参照してください。「IDE 設定」ダイアログで指定したソースツリーがすべてのプロジェクトへ適用されます。ターゲット設定ダイアログで指定したソースツリーはアクティブなブ

プロジェクトの現在のビルドターゲットに適用されます。両方で同じソースツリーを指定すると、ターゲット固有のソースツリーが優先されます。

ソースツリーリスト

ソースツリーリストは定義されたソースツリーをすべて表示します。2 つの列があります。

名前：この列は各ソースツリーの名前を表示します。ソースツリーとしてアクセスパスを定義すると、アクセスパスの定義にこの名前を使用できます。詳細は「[追加](#)」(p234) を参照してください。

パス：この列は各ソースツリーへのパスを表示します。プロジェクトを他のホストへ移植するときはソースツリーのパスを修正しなくてはなりません。パスの修正方法は「[変更](#)」(p192) を参照してください。

名前

[ソースツリー情報] 欄の [名前] フィールド ([図 8.9](#)) で新しいソースツリーの名前を入力するか、またはソースツリーリストの行を選択して名前を変更します。

型

[ソースツリー情報] 欄の [型] ポップアップメニュー ([図 8.9](#)) でソースツリーの型を選択します。

絶対パス：ファイルパスに基づくソースツリー

環境変数：既存の環境変数の定義に基づくソースツリー (Mac OS をホストにする IDE では使用できない)

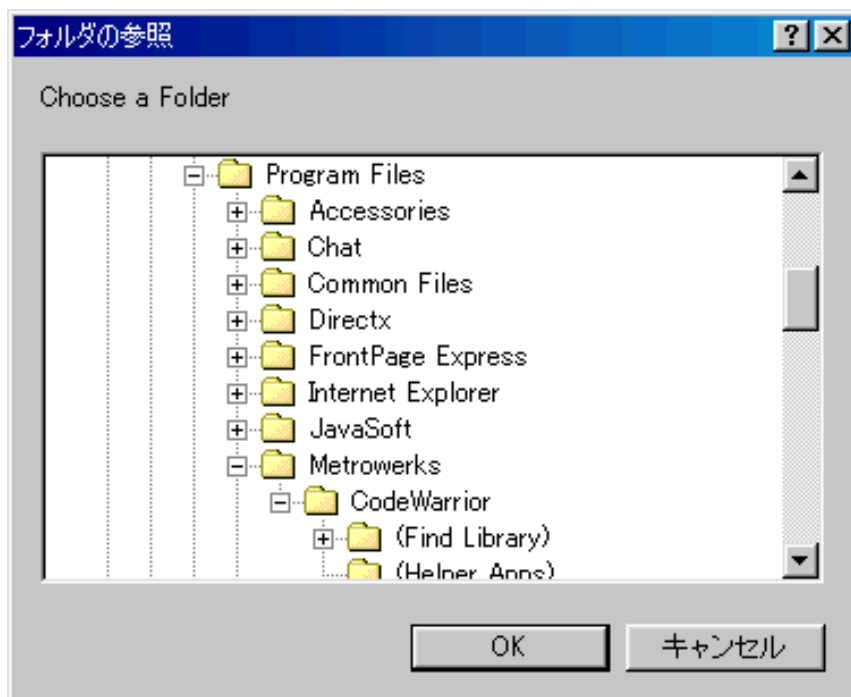
レジストリキー：レジストリにあるキーエントリに基づくソースツリー

追加

新しいソースツリーを追加するには、[型] ポップアップメニューからソースツリーの種類を選択します。次に [名前] フィールドに新しいソースツリーの名前を入力します。

[絶対パス] のソースツリーを作るときは [選択] ボタンが使用可能になります。このボタンをクリックすると標準ダイアログでパスを選択することができます ([図 8.10](#))。

図 8.10 「フォルダの参照」ダイアログ



Windows をホストとする IDE では、保存するパスの種類を選択できます。

絶対パス：IDE は起動ハードディスクのルートレベルから、追加するフォルダへのパスを、その間のフォルダも含めて保存します。プロジェクトを他のシステムへ移動したとき、ハードディスクの名前を変更したとき、その間のフォルダの名前を変更したときに絶対パスを更新する必要があります。

コンパイラ相対：CodeWarrior IDE のあるフォルダから、追加するフォルダへのパスを保存します。相対パスの階層が変化していなければ、プロジェクトを移動したときでも相対パスを更新する必要はありません。CodeWarrior がインストールされているドライブ以外への相対パスは作成できません。

システム相対：オペレーティングシステムシステムのベースフォルダから、追加するフォルダへのパスを保存します。システム相対パスの階層が変化していなければ、プロジェクトを移動したときでもシステム相対パスを更新する必要はありません。オペレーティングシステムシステムのベースフォルダがあるドライブ以外への相対パスは作成できません。

注意： 相対パスを使うと、複数の同じ名前のファイルをプロジェクトに追加することができます。しかし大きなプロジェクトで相対パスを使うと、若干パフォーマンスが低下します。

[環境変数] または [レジストリキー] のソースツリーを作成したとき、[型] フィールドが図 8.11 のように変化します。ここに環境変数かレジストリキーへのパスを入力してください。

図 8.11 レジストリキーを作成

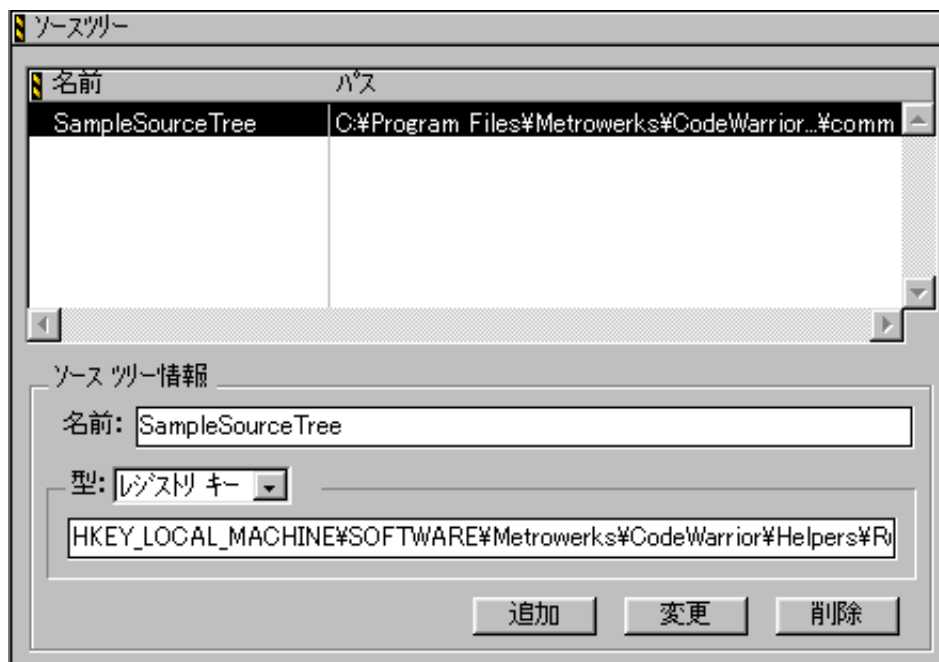
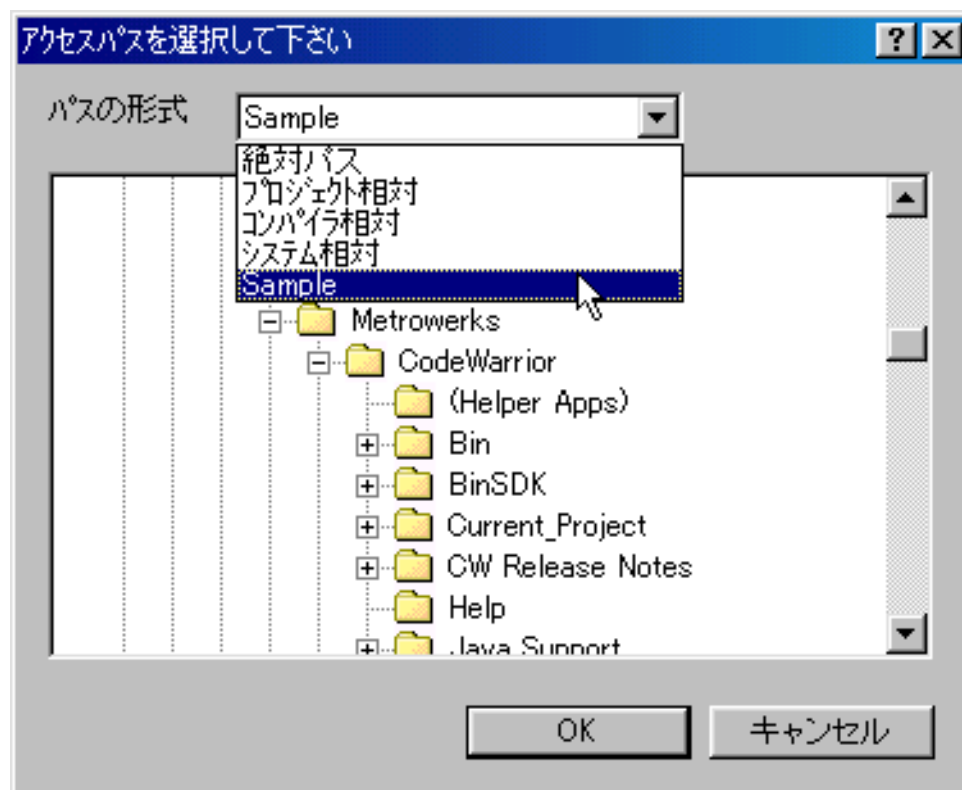


図 8.12 [相対] ポップアップメニューからソースツリーを選択



ソースツリーの追加を終えたら、「IDE 設定」ダイアログのボタンをクリックして変更を保存します。ソースツリーがプロジェクトの[相対]ポップアップメニューに表示されます。例えば「Sample」という名前のソースツリーを作った場合、これを利用してアクセスパスを作成できます(図 8.12)。また[ターゲット設定](#)パネルの[出力ディレクトリ]フィールドを使ってプロジェクトのターゲット出力を定義することもできます。詳細は「[ターゲット設定](#)」(p229)を参照してください。

変更

ソースツリーを変更するには、[ソースツリーリスト](#)で選択します。次に[名前]と[型]フィールドを変更し、[変更] ボタンをクリックします。変更を終えたら「IDE 設定」ダイアログの[保存] ボタンをクリックして保存します。

注意： ソースツリーを変更した後、元のソースツリーを参照しないようにプロジェクトを修正する必要があります。IDE はプロジェクトの更新を要求するメッセージを表示します。

プロジェクトのファイルまたはライブラリがソースツリーになれば、IDE はプロジェクトをコンパイル、リンク、または実行するときにそれらを見つけることができません。このため、既存のソースツリーを変更したときにIDEはプロジェクトの更新を求めるメッセージを表示します。

削除

ソースツリーを削除するには、ソースツリーを設定パネルで選択します。[削除] ボタンをクリックして削除します。ソースツリーの削除を終えたら、「IDE 設定」ダイアログの [保存] ボタンをクリックして変更を保存します。

注意： ソースツリーを削除した後、削除したソースツリーを参照しないようにプロジェクトを修正する必要があります。IDE はプロジェクトの更新を要求するメッセージを表示します。

エディタの IDE 設定

ここではエディタの IDE 設定パネルとその機能を説明します。

[ブラウザ表示](#)

[エディタ設定](#)

[フォント & タブ](#)

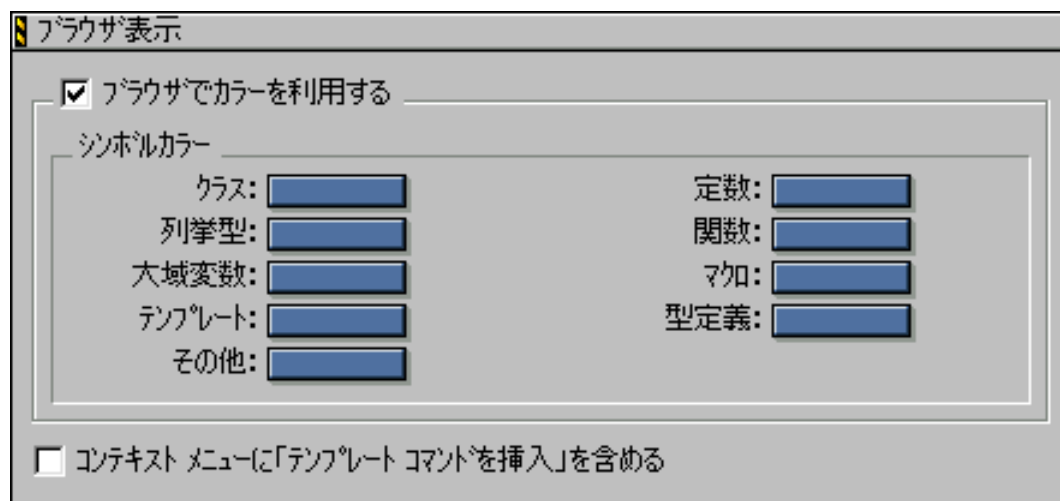
[カラーシンタックス](#)

ブラウザ表示

ブラウザ表示設定パネルを[図 8.13](#)に示します。

「IDE 設定」ダイアログとブラウザ表示設定パネルの開き方は[「IDE 設定パネルの解説」\(p181\)](#)を参照してください。

図 8.13 ブラウザ表示設定パネル



ブラウザはシンボルのリストとタイプを CodeWarrior エディタへエクスポートできます。これによりエディタでいろいろなタイプのシンボルを違う色で表示することができます。

この機能を使うには、[ブラウザでカラーを利用する] をオンにしてください。オンの場合、各シンボルタイプの色がエディタウィンドウとブラウザウィンドウに表示されます。色を選択するには、カラーエリアをクリックしてください。

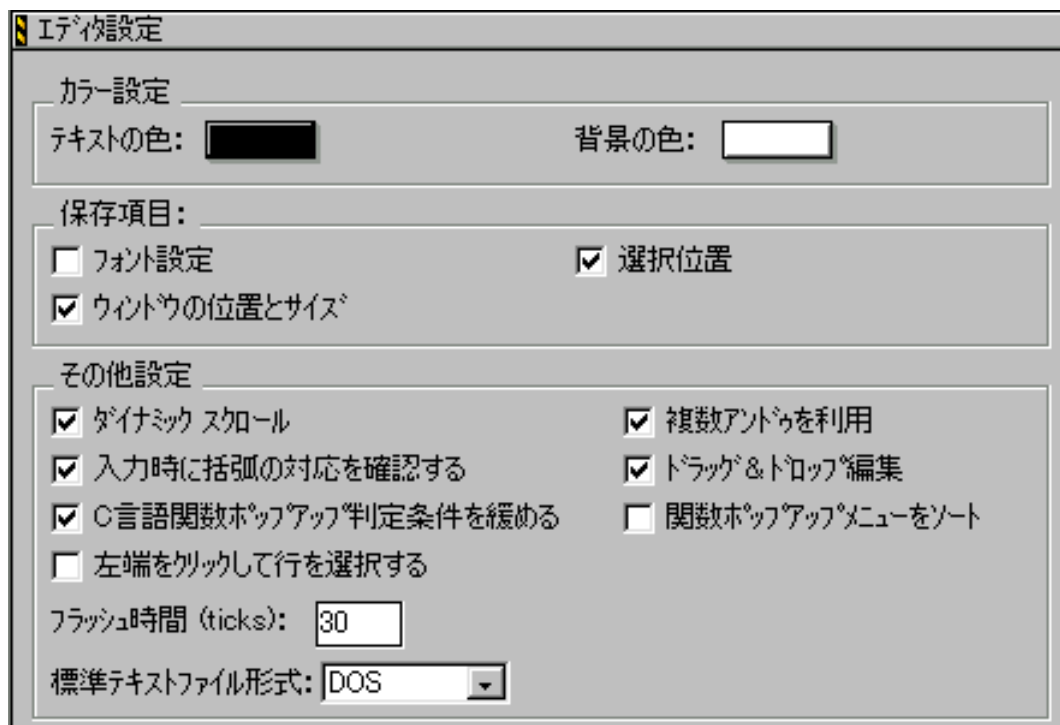
コンテキストポップアップメニューを使ってソースコードにテンプレートを挿入するには、[コンテキストメニューに「テンプレートコマンドを挿入」を含める] オプションをオンにします。デフォルトではこのオプションはオフです。オンの場合、コンテキストポップアップメニューに [テンプレートを挿入] コマンドが表示されます ([図 7.4 \(p148\)](#))。

エディタ設定

エディタ設定パネル ([図 8.14](#)) で、CodeWarrior エディタの動作を設定します。

「IDE 設定」ダイアログとエディタ設定パネルの開き方は [「IDE 設定パネルの解説」\(p181\)](#) を参照してください。

図 8.14 エディタ設定パネル



このパネルのオプションは3つのグループに分かれています。[[カラー設定](#)] 欄はメインテキスト (シンタックス以外) のカラーとエディタとブラウザウィンドウの背景色を指定します。[[保存項目](#)] 欄はエディタウィンドウのどの情報を保存するかを指定します。[[その他設定](#)] 欄 ([[入力時に括弧の対応を確認する](#)]) はエディタの作業環境を指定します。

カラー設定

[カラー設定] 欄のオプションでエディタのカラーを設定します。

テキストの色 : このオプションは [[ブラウザ表示](#)] [[カラーシンタックス](#)] または [[カスタムキーワード](#)] のカラーセットで色が付けられないテキストの色を設定します。カラーエリアをクリックすると色を変更できます。

背景の色 : カラーエリアをクリックすると、エディタおよびブラウザのウィンドウの背景色を変更できます。

保存項目

[保存項目] 欄のオプションでエディタが保存する情報を設定します。

フォント設定 : このオプションがオンの場合、フォント情報を個別のファイルに設定できます。オフの場合、すべてのファイルが CodeWarrior IDE のデフォルトのフォント設定を使用します。

ウィンドウの位置とサイズ : ウィンドウの画面上の位置、サイズを記憶し、ファイルを毎回同じ場所に開きます。この機能は、ファイルが書き込み可能である必要があります。書き込み可能ファイルの詳細は、[「一般的な VCS の操作」\(p452\)](#) を参照してください。

選択位置 : ビュー内でスクロールされているテキストの位置および挿入位置、または選択の位置を記憶します。オフの場合、ウィンドウを開くとき常にファイルの先頭で開きます。この機能は、ファイルが書き込み可能である必要があります。書き込み可能ファイルの詳細は、[「一般的な VCS の操作」\(p452\)](#) を参照してください。

その他設定

[その他設定] 欄でその他のエディタの設定を行います。

入力時に括弧の対応を確認する : [入力時に括弧の対応を確認する] オプションをオンにすると、入力時に括弧、大括弧、中括弧の対応がチェックされます。右の括弧、大括弧、または中括弧を入力したとき、エディタが対応する左側の括弧を探します。見つけると [[フラッシュ時間](#)] に指定した時間だけ強調表示し (必要ならスクロールされます)、その後元に戻ります (必要ならスクロールして戻ります)。対応する左側が見つからないときは、ピープ音が鳴ります。デフォルトではこのオプションはオンです。フラッシュ時間については [「フラッシュ時間」\(p196\)](#) を参照してください。

ヒント : 強調表示をせずに対応する括弧をチェックするには、[フラッシュ時間] を 0 に設定します。

C 言語関数ポップアップ判定条件を緩める : K&R (The C Programming Language, Second Edition (プレンティスホール出版)、Kernighan and Ritchie 著) スタイルのコードを書く場合、このオプションをオンにします。CodeWarrior IDE は K&R スタイルの関数を認識し、関数ポップアップメニューに関数名を正しく表示します。K&R スタイルのコードに干渉する標準以外のマクロを使っている場合、オフにします。

注意 : [C 言語関数ポップアップ判定条件を緩める] がオンの場合、マクロ関数が認識されないことがあります。関数名が表示されないという問題が発生したら、このオプションをオフにしてください。

左端をクリックして行を選択する : オンの場合、マウスポインタをエディタウィンドウの左端に移動すると、マウスポインタが右向きの矢印に変わります。そのときに左端をクリックすると、マウスポインタのある行が選択できます。クリックしてからドラッグすると、複数行を選択できます。オフの場合、マウスポインタは通常の動作をします。

複数アンドゥを利用 : 複数回の取り消し機能を使いたい場合は、このオプションをオンにします。オフにすると、直前の操作の取り消しまたはやり直ししかできません。詳細は「[再実行](#)」(p459) を参照してください。

ドラッグ & ドロップ編集 : このオプションがオンの場合、ドラッグ & ドロップをサポートします。詳細は、「[テキストを移動 \(ドラッグ & ドロップ\)](#)」(p107) を参照してください。

関数ポップアップメニューをソート : このオプションをオンにすると、エディタウィンドウの[関数ポップアップメニュー](#)がデフォルトでアルファベット順にソートされます。この機能の詳細は、「[関数ポップアップメニュー](#)」(p98) を参照してください。

フラッシュ時間 [フラッシュ時間] はエディタで項目を強調表示する時間です。1/60 秒単位で指定します。このオプションは括弧のバランスをとるときに使われます。括弧のバランスについては「[括弧のバランスをとる](#)」(p107)、「[入力時に括弧の対応を確認する](#)」(p195) を参照してください。

警告! [ポップアップ遅延時間] フィールドに 0 (ゼロ) を入力すると、強調表示はされません。

警告! 「ポップアップ遅延時間」フィールドに 0 (ゼロ) を入力すると、ポップアップメニューは表示されません。

標準テキストファイル形式 : このポップアップメニューで、新規ファイルを作成する際の行末文字の規約を指定します。[DOS] [Macintosh] [UNIX] のいずれかを選択してください。テキストファイルを異なるフォーマットで保存する方法は「[オプションポップアップメニュー](#)」(p99) を参照してください。

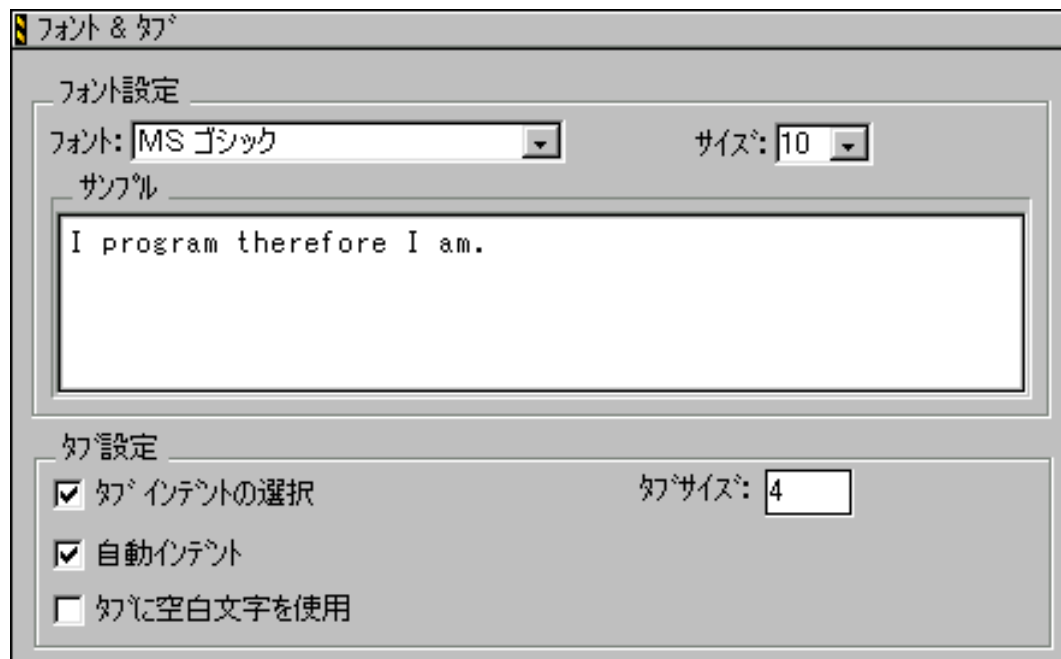
フォント & タブ

フォント & タブ設定パネル ([図 8.15](#)) でアクティブなエディタウィンドウ用のフォントとタブ情報を設定します。開いているエディタウィンドウがない場合、このパネルの設定が CodeWarrior IDE のデフォルトになります。

「IDE 設定」ダイアログとフォント & タブ設定パネルの開き方は「[IDE 設定パネルの解説](#)」(p181) を参照してください。

ファイルのフォントおよびタブ設定を変更するには、ファイルをエディタウィンドウに開きます。その後、「IDE 設定」ダイアログからフォント & タブ設定パネルを開いて変更を加えます。ファイルを開くたびに記憶しているフォントを使うことができます。

図 8.15 フォント & タブ設定パネル



ファイルに対して書き込み許可がある（ファイルがリードオンリーではない）限り、加えた変更は記憶されます。書き込み可能ファイルの詳細は、[「一般的な VCS の操作」\(p452\)](#) を参照してください。

フォント

[フォント] ポップアップメニューでテキストのフォントを選択します。

サイズ

[サイズ] ポップアップメニューでフォントのサイズを選択します。

サンプル

[サンプル] 欄は現在選択している [フォント](#) と [サイズ](#) でテキスト見本を表示します。

ドキュメント・タブ設定

[ドキュメント・タブ設定] 欄で CodeWarrior IDE のタブを設定します。

タブインデントの選択 : オンの場合、テキスト行を選択して Tab キーを押すと、その行の前にタブが入ります。タブはテキストを置換しません。オフの場合、選択した表がタブで置換されます。デフォルトはオンです。

自動インデント : 新しい行に、前の行と同じ字下げ位置を自動的に適用したい場合は、[自動インデント] をオンにします。

タブに空白文字を使用 : Tab キーを押してスペースを入力するにはこのオプションをオンにします。スペースの数は [[タブサイズ](#)] フィールドで決めます。

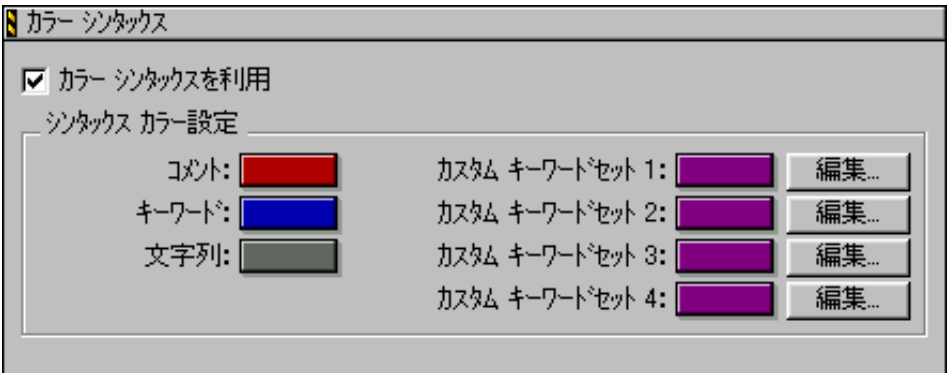
タブサイズ : Tab キーを押したときに入力するときのスペースの数を入力します。

カラーシンタックス

カラーシンタックス設定パネル (図 8.16) には、コメント、キーワード、文字列、カスタムキーワードセットに色を設定するオプションがあります。[カラーシンタックスを利用] チェックボックスをオンにすると、このパネルで指定した色でそのテキストファイルを表示します。

「IDE 設定」ダイアログとカラーシンタックス設定パネルの開き方は「[IDE 設定パネルの解説](#)」(p181) を参照してください。

図 8.16 カラーシンタックス設定パネル



4 種類のワードにそれぞれカラーを指定できます。各カスタムキーワードセットには関数名、型名、またはその他何でもエディタウィンドウ内で強調表示したいテキストを指定できます。表 8.1 にカラーを設定できるテキストを示します。

表 8.1 カラーシンタックスの要素

要素	説明
コメント	コードのコメント。Java、C/C++ では、コメントは /* と */ で囲まれたテキストか、// から行の最後までのテキストです。Pascal ではコメントは { }, または (* *) で囲まれたテキストです。
キーワード	プログラミング言語のキーワード。システムヘッダファイルで定義したマクロ、型、変数、またはユーザーが定義した変数は含まれません。
カスタム キーワード	カスタムキーワードリストにあるキーワード。このリストには、マクロ、型、その他の強調表示したい名前を指定できます。
文字列	コメント、キーワード、またはカスタムキーワード以外のすべて (例: リテラル値、変数名、関数名、型名など)。

シンタックス強調表示の色を変える

CodeWarrior IDE は、テキストの種類によって違う色を使います。色を変えるには、名前の横にある色のサンプルをクリックします。色を選択するためのダイアログが表示されます。次回テキストを表示するときから、この色が使われます。

ウィンドウ内でシンタックスカラーリングをコントロール

[オプションポップアップメニュー](#)の[カラーシンタックス]でシンタックスカラーをオン、オフを切り替えます。詳細は「[オプションポップアップメニュー](#)」(p99)を参照してください。

独自のキーワード用に色を使う

「カスタムキーワード」ダイアログ ([図 8.17](#)) を使って色で表示する単語を選択、追加できます。単語は、マクロ、型、またはその他の強調したい単語を何でも指定できます。このキーワードは CodeWarrior IDE 全体に有効で、すべてのプロジェクトに適用されます。

修正したい [カスタムキーワードセット] の右の [編集] ボタンをクリックします。「カスタムキーワード」ダイアログ ([図 8.17](#)) が表示されます。

図 8.17 「カスタムキーワード」ダイアログ



「カスタムキーワード」ダイアログの [カスタムキーワード] フィールドにキーワードを入力して、[追加] ボタンをクリックします。キーワードがカスタムキーワードリストに追加されます。カスタムキーワードリストに追加するキーワードが多すぎると、追加ができなくなります。この場合キーワードの追加が失敗したというエラーメッセージが現れます。

キーワードを削除するには、選択して Backspace キーを押します。カスタムキーワードリストからそのキーワードが削除されます。

終わったら [終了] ボタンを押します。ダイアログが消えます。次回ソースファイルを表示したとき、入力したすべてのキーワードが選択したカラーで表示されます。

ヒント： カスタムキーワードにターゲット固有の色を付けることもできます。[「カスタムキーワード」\(p251\)](#) を参照してください。

カスタムキーワードをインポートまたはエクスポート

既存のカスタムキーワードセットをIDEへインポートしたり、その他のホストへエクスポートすることができます。

以下にカスタムキーワードをインポートする手順を示します。

1. [カラーシンタックス](#)設定パネルで [カスタムキーワードセット] の横の [編集] ボタンをクリックする

「カスタムキーワード」ダイアログが現れます。

2. [ファイルから読み込む] ボタンをクリックする

標準の「開く」ダイアログが現れます。

3. ダイアログでインポートするカスタムキーワードセットを選択する

4. [開く] ボタンをクリックする

IDE は選択したセットからカスタムキーワードをカスタムキーワードリストへ追加します。

5. [終了] ボタンをクリックする

インポートしたカスタムキーワードが、カスタムキーワードセットに表示されます。

以下にカスタムキーワードをエクスポートする手順を示します。

1. [カラーシンタックス](#)設定パネルで [カスタムキーワードセット] の横の [編集] ボタンをクリックする

「カスタムキーワード」ダイアログが現れます。

2. [ファイルに書き出す] ボタンをクリックする

「ファイルへ書き出し」ダイアログが現れます。

3. ダイアログでエクスポートするカスタムキーワードセットの保存位置を選択する

エクスポートしたカスタムキーワードセットを別のホストで使う場合、セットの名前に拡張子 `.txt` を付けてください。Windows 版の IDE はカスタムキーワードセットを認識するために拡張子が必要です。

4. [保存] ボタンをクリックする

IDE はカスタムキーワードセットをエクスポートし、それを保存します。

5. [終了] ボタンをクリックする

カラーシンタックス設定パネルへ戻ります。

デバッガの IDE 設定

以下のデバッグ機能の IDE 設定パネルについて説明します。

[表示設定](#)

[ウィンドウ指定](#)

[一般設定](#)

[Java デバッグング](#)

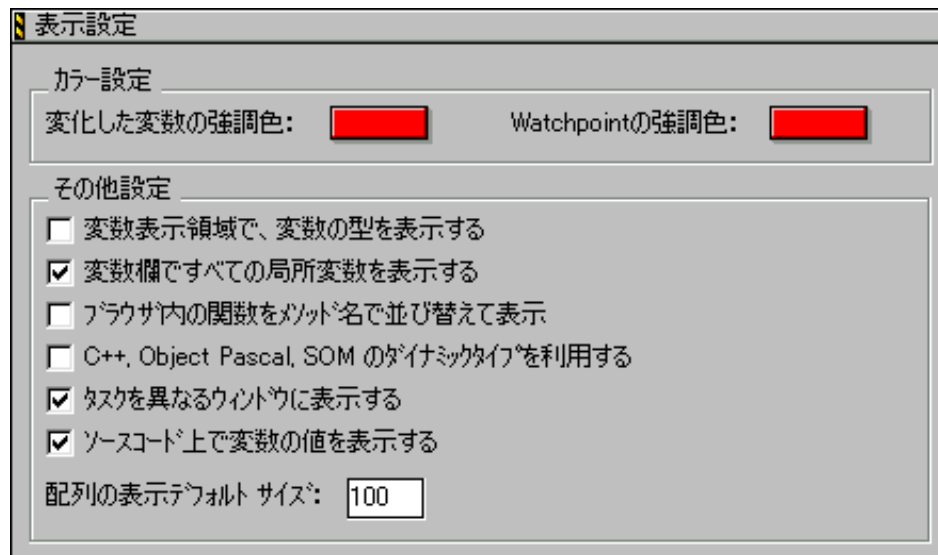
[Java 設定](#)

表示設定

表示設定パネル ([図 8.18](#)) で CodeWarrior デバッガの作業環境を設定します。

「IDE 設定」ダイアログと表示設定パネルの開き方は [「IDE 設定パネルの解説」\(p181\)](#) を参照してください。

図 8.18 デバッガの表示設定パネル



変化した変数の強調色

[変化した色の強調色] オプションで変化した変数を示すカラーを選択します。カラーを選択するには、名前の横のサンプルをクリックします。CodeWarrior IDE は標準のカラー選択ダイアログを表示します。次にデバッグウィンドウを開くと、指定したカラーが使われます。

Watchpoint 強調色

[Watchpoint 強調色] オプションは Watchpoint を示すカラーを選択します。カラーを選択するには、名前の横のサンプルをクリックします。CodeWarrior IDE は標準のカラー選択ダイアログを表示します。次にデバッガウィンドウを開くと、指定したカラーが使われます。

変数表示領域で変数の型を表示する

[変数表示領域で変数の型を表示する] チェックボックスがオンの場合、新しい変数ウィンドウを開いたときにデフォルトで変数型を表示します。

変数欄ですべての局所変数を表示する

[変数欄ですべての局所変数を表示する] チェックボックスがオンの場合、新しい変数ウィンドウを開いたときに局所変数を表示します。オンの場合、変数欄は常に [変数 : すべて] を表示します。オフの場合、変数欄には、状態によって [変数 : すべて] か、[変数 : 自動] のどちらかが表示されます。詳細は「[変数欄](#)」(p284) を参照してください。

ブラウザ内の関数をメソッド名で並べ替えて表示

ブラウザウィンドウで C++、Object Pascal、Java 関数のソート順を変えます。[ブラウザ内の関数をメソッド名で並び替えて表示] チェックボックスがオフの場合、関数名は `className: : methodName` 形式でアルファベット順にソートされ、最初にクラス名、次にメソッド名が表示されます。オンの場合、関数はメソッド名でアルファベット順にソートされます。C++、Object Pascal、Java のソースコードには同じクラスのメソッドを含むことが多いため、このオプションによりキーボード入力でメソッドを選択することが簡単になります。

C++、Object Pascal、SOM のダイナミックタイプを利用する

[C++、Object Pascal、SOM のダイナミックタイプを利用する] チェックボックスがオンの場合、C++、Object Pascal のオブジェクトのランタイムタイプを表示します。オフの場合、オブジェクトの静的タイプだけを表示します。

タスクを異なるウィンドウに表示する

タスクの表示方法を切り替えます。[タスクを異なるウィンドウに表示する] チェックボックスがオンの場合、[プロセスウィンドウ](#)でタスクをダブルクリックすると、別の[プログラムウィンドウ](#)を開いてコードを表示します。オフの場合、プログラムウィンドウ上部に[タスク] ポップアップメニューが表示されます。このメニューでプログラムウィンドウに表示するタスクを選択します。

注意： このオプションはすぐには適用されません。これはデバッグを開始したときに有効になります。デバッグ中にこのオプションを変更した場合、デバッグを中止してやり直しをすると変更が適用されます。

ソースコード上で変数の値を表示する

[ソースコード上で変数の値を表示する] チェックボックスがオンの場合、自動的にソースコードの変数の値を表示し、オフの場合、変数の値は表示しません。

配列の表示デフォルトサイズ

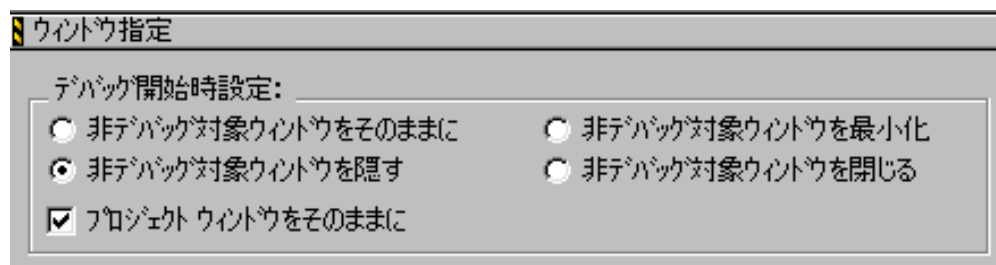
[配列の表示デフォルトサイズ] フィールドで配列のデフォルトサイズを指定します。

ウィンドウ指定

ウィンドウ指定設定パネル ([図 8.19](#)) で、デバッグ中のウィンドウの挙動を設定します。

「IDE 設定」ダイアログとウィンドウ指定設定パネルの開き方は [「IDE 設定パネルの解説」\(p181\)](#) を参照してください。

図 8.19 ウィンドウ指定設定パネル



デバッグ開始時設定

[デバッグ開始時設定] 欄のオプションでデバッグ以外のウィンドウの、デバッグ中の挙動を設定します。

非デバッグ対象ウィンドウをそのままに：オンの場合、デバッグが始まってでもデバッグ以外のウィンドウを閉じたり隠したりしません。複数のビルドターゲットや複数プロジェクトをデバッグするときに便利です。

非デバッグ対象ウィンドウを隠す：オンの場合、デバッグ以外のウィンドウを隠しますが、閉じません。隠されたデバッグ以外のウィンドウへは[ウィンドウメニュー](#)からアクセスします。プロジェクトウィンドウでファイルをダブルクリックするか、シンボルを参照すると隠されたウィンドウが表示されます。デバッグが終わると隠されていたウィンドウが表示されます。

非デバッグ対象ウィンドウを最小化：オンの場合、デバッグ以外のウィンドウを最小サイズで表示します。デバッグ以外のウィンドウには[ウィンドウメニュー](#)からアクセスします。プロジェクトウィンドウでファイルをダブルクリックするか、シンボルを参照すると最小化されたウィンドウが拡張されます。これは MDI モードでのみ利用可能です。[「IDE その他設定」\(p185\)](#)、[「MDI 使用」\(p186\)](#) を参照してください。

非デバッグ対象ウィンドウを閉じる：オンの場合、デバッグ中のプロジェクトウィンドウを除く、デバッグ以外のウィンドウを閉じます。デバッグが終わると閉じられていたウィンドウが再度開かれます。

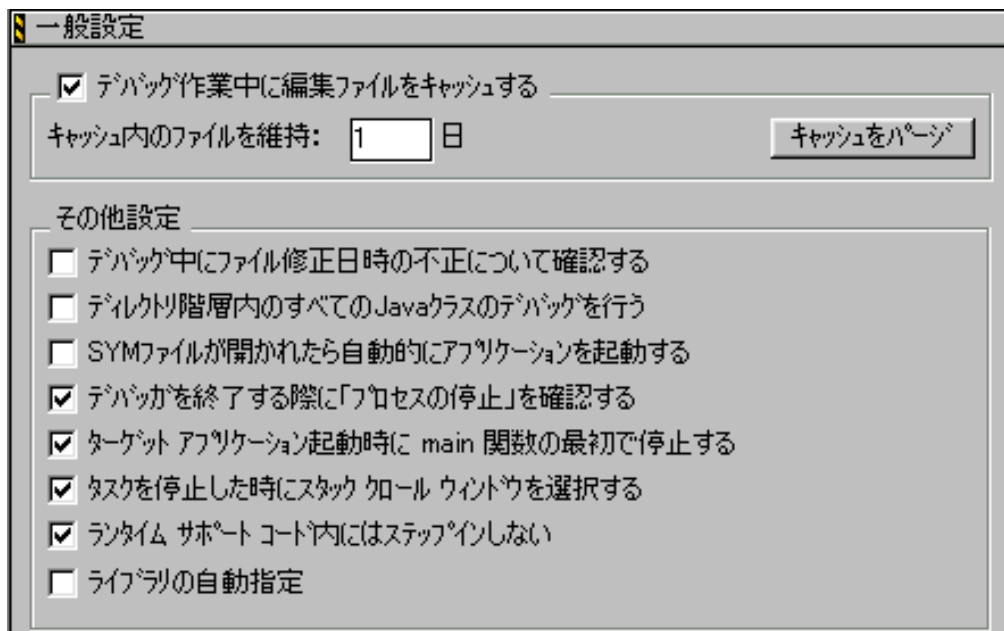
プロジェクトウィンドウを変更しない：オンの場合、デバッグ中のプロジェクトウィンドウを除くデバッグ以外のウィンドウを閉じます。デバッグが終わると閉じられていたウィンドウが開かれます。

一般設定

一般設定パネル ([図 8.20](#)) でデバグの挙動を設定します。

「IDE 設定」ダイアログと一般設定パネルの開き方は「[「IDE 設定パネルの解説」\(p181\)](#)」を参照してください。

図 8.20 一般設定パネル



デバグ作業中に編集ファイルをキャッシュする

[デバグ作業中に編集ファイルをキャッシュする] チェックボックスがオンの場合、編集しているファイルのキャッシュをデバグ中にも維持することができます。オフの場合、キャッシュを保存する日数を [キャッシュ内のファイルを維持] フィールドで指定します。[キャッシュをリセット] ボタンをクリックするとキャッシュをリセットできます。オフの場合、ファイルキャッシュは維持されません。

デバグ中にファイル修正日時の不正について確認する

デバグはシンボルファイルを生成するソースファイルの更新日付を追跡します。更新日付が一致しないとき、デバグはオブジェクトコードとソースコードに差があるかもしれないことを警告します。[デバグ中にファイルの修正日時の不正について確認する] チェックボックスオンの場合、デバグは警告を表示します。

ディレクトリ階層内のすべての Java クラスのデバグを行う

[ディレクトリ階層内のすべての Java クラスのデバグを行う] チェックボックスはプロジェクトファイルを開かずに Java プログラムをデバグするときに役立ちます。

[ディレクトリ階層内のすべての Java クラスのデバグを行う] チェックボックスがオンの場合、Java のデバグを始めると、デバグはシンボルファイルを検索して開きます。開いているクラスファイルのあるフォルダとそのサブフォルダを検索します。

SYM ファイルが開かれたら自動的にアプリケーションを起動する

[SYM ファイルが開かれたら自動的にアプリケーションを起動する] チェックボックスがオンの場合、シンボルファイルが開かれ、暗黙のブレークポイントがプログラムのメインエントリポイントに設定されたときに自動的にターゲットプログラムを起動します。オフの場合、シンボルファイルを開いてもプログラムは起動せず、メインルーチンの前に実行されるオブジェクトコード (C++ 静的コンストラクタ関数など) を調べることができます。Alt キーを押しながらシンボルファイルを開くと、ターゲットプログラムは起動しません。

デバッガを終了する際にプロセスの停止を確認する

[デバッガを終了する際に「プロセスの停止」を確認する] チェックボックスがオンの場合、ターゲットプログラムを終了する際にプロセスの終了を問い合わせます。

タスクを停止したときにプログラムウィンドウを選択する

[タスクを停止した時にプログラムウィンドウを選択する] チェックボックスがオンの場合、タスクが停止したときに、自動的にプログラムウィンドウを最前面に表示します。オフの場合、プログラムウィンドウの位置は変わりません。複数の変数ウィンドウを開いていて、ステップ実行したときの変化を見たいときに便利です。

ランタイムサポートコード内にはステップインしない

[ランタイムサポートコード内にはステップインしない] チェックボックスがオンの場合、C++ の静的オブジェクトのコンストラクタコードを、プログラムウィンドウに表示せずに実行します。

ライブラリの自動指定

[ライブラリの自動指定] チェックボックスは、実行中のプロセスをアタッチしたときなど、プロジェクトファイル以外のファイルをデバッグするときに適用されます。プロセスのアタッチについては「[プロセスウィンドウのツールバー](#)」(p295) を参照してください。

オンの場合、ターゲットアプリケーションにロードされたダイナミックリンクライブラリ (DLL) をデバッグしようとしています。IDE は自動的にシンボル情報を持つ DLL をデバッグしようとしています。

注意： [自動ターゲット・ライブラリ] チェックボックスがオンの場合、実行速度が遅くなることがあります。これを避けるにはチェックボックスをオフにしてください。

Java デバッグング

Java デバッグング設定パネル ([図 8.21](#)) で、Java デバッグのためのコンピュータと仮想マシンの接続を設定します。Java デバッグの詳細は『Targeting Java』マニュアルを参照してください。

「IDE 設定」ダイアログと Java デバッグング設定パネルを開く方法は「[IDE 設定パネルの解説](#)」(p181) を参照してください。

図 8.21 Java デバッグ設定パネル

Java デバッグ

タイムアウト(秒): 5

☐ リモート デバッグ

リモート IP アドレス: 127.0.0.1

ポート ID: 8000

プロトコル: 1.1.X Wire プロトコル

JDK バージョン: 1.1

タイムアウト (秒)

[タイムアウト (秒)] フィールドへ、CodeWarrior デバッガが仮想マシンの接続を待つ時間 (秒単位) を入力します。

リモートデバッグ

自分のコンピュータからコンピュータから Java プロジェクトをリモートデバッグするには、[リモートデバッグ] チェックボックスをオンにします。

リモート IP

デバッグに使用する仮想マシンへネットワーク接続している場合に使います。仮想マシンが動作しているコンピュータのインターネットプロトコル (IP) アドレスを、[リモート IP] フィールドへ入力します。このフィールドは [[リモートデバッグ](#)] チェックボックスがオンの場合のみ利用可能です。

ポート ID

デバッグに使用する仮想マシンへネットワーク接続している場合に使います。仮想マシンと CodeWarrior デバッガを接続するポート ID 番号を、[ポート ID] フィールドへ入力します。このフィールドは [[リモートデバッグ](#)] チェックボックスがオンの場合のみ利用可能です。

プロトコル

[プロトコル] ポップアップメニューからコンピュータとリモート仮想マシンの間の転送プロトコルを選択します。このポップアップメニューは [[リモートデバッグ](#)] チェックボックスがオンの場合のみ利用可能です。

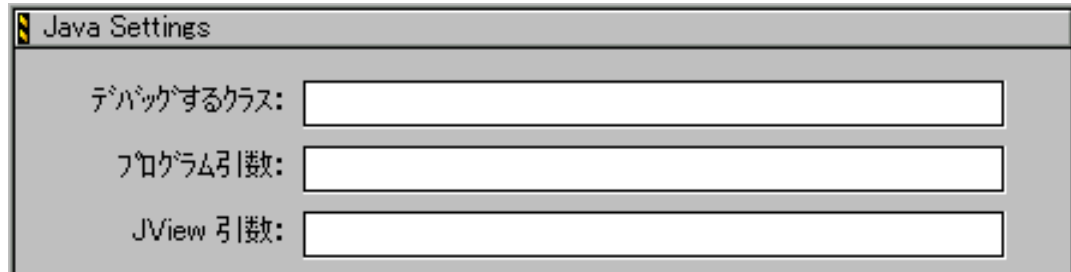
JDK バージョン

[JDK バージョン] ポップアップメニューから、リモートデバッグに使用する JDK (Java Development Kit) のバージョンを選択します。このポップアップメニューは [[リモートデバッグ](#)] チェックボックスがオン、かつ [[プロトコル](#)] ポップアップメニューで [1.1X Wire Protocol] を選択した場合のみ利用可能です。

Java 設定

Java 設定パネル ([図 8.22](#)) で Java 用デバuggの挙動を設定します。Java プログラムとアプレットのデバッグについては『Targeting Java』マニュアルを参照してください。

図 8.22 Java 設定パネル



「IDE 設定」ダイアログと Java 設定パネルの開き方は「[IDE 設定パネルの解説](#)」(p181) を参照してください。

デバッグするクラス

[デバッグするクラス] フィールドにデバッグするクラスを入力します。

プログラム引数

[プログラム引数] フィールドに Java アプリケーションをデバッグするときに使うコマンドライン引数を入力します。

JView 引数

[JView 引数] フィールドに、プロジェクトをデバッグするときに JView が必要とする引数を入力します。

IDE をカスタマイズ

IDE のメニュー、キーバインディング、ツールバーを好みに合わせてカスタマイズできます。「IDE コマンドのカスタマイズ」ダイアログを使います。このウィンドウを表示するには、編集メニューの [コマンドとキーバインディング] を選択します。

以下の項目について説明します。

[ダイアログのボタン](#)

[コマンドをカスタマイズ](#)

[キーバインディングをカスタマイズ](#)

ツールバーをカスタマイズ

ダイアログのボタン

「IDE コマンドのカスタマイズ」ダイアログには、設定の適用を制御するボタンがあります。

以下の内容について説明します。

[保存しない](#)

[出荷時設定](#)

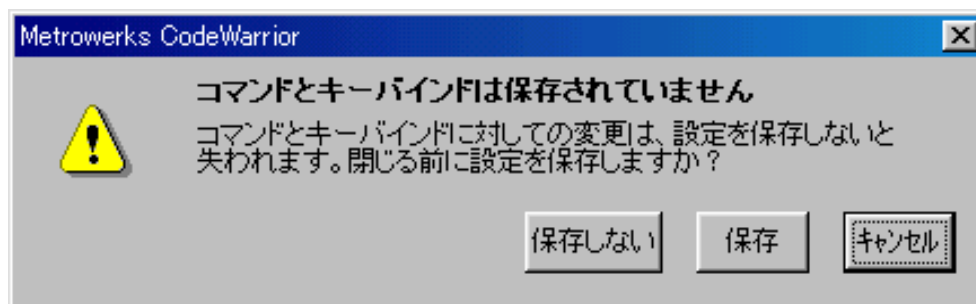
[復帰](#)

[保存](#)

変更を破棄

「IDE コマンドのカスタマイズ」ダイアログでオプション設定を変更した後にダイアログを閉じようとする、警告ダイアログ ([図 8.23](#)) が表示されます。変更を破棄するには [保存しない] ボタンをクリックします。変更を保存するには [保存] ボタンをクリックします。変更を保存せずに「IDE 設定」ダイアログへ戻るには [キャンセル] ボタンをクリックします。

図 8.23 コマンドとキーバインディングの警告ダイアログ



出荷時設定

[出荷時設定] ボタンをクリックすると、「IDE コマンドのカスタマイズ」ダイアログのオプションがデフォルトの設定に戻ります。

警告！ [工場出荷時] ボタンをクリックするとすべての設定がデフォルトに戻ります。デフォルトの設定に戻す前にカスタマイズしたコマンドグループやメニューコマンドをエクスポートしなければ、カスタマイズした項目は失われます。コマンドのエクスポートについては [「キーバインディングとコマンドをエクスポート」\(p217\)](#) を参照してください。

復帰

[復帰] ボタンは、現在表示中の「IDE コマンドのカスタマイズ」ダイアログの状態を、最後に保存したときの設定に戻します。これは、変更を加えた後で変更を適用したくない場合に便利です。

保存

[保存] ボタンは、「IDE コマンドのカスタマイズ」ダイアログで加えた変更を保存します。ダイアログを閉じた後、CodeWarrior IDE の挙動が変わります。

コマンドをカスタマイズ

IDE のメニューバーに表示するコマンドをカスタマイズできます。メニューコマンドの表示方法をコントロールしたり、新しいコマンドグループを作ってメニューに入れたり、コマンドラインに関するコマンドを作ることもできます。カスタマイズしたコマンドは IDE 情報(現在のエディタ、最前面のウィンドウ、現在のプロジェクトやその出力ファイル)へアクセスできます。

例えば「MyMenu」というコマンドグループを作って、そこに外部アプリケーションを起動する [RunApp] というコマンドを入れることもできます。「IDE コマンドのカスタマイズ」ダイアログを使って新しいコマンドやグループを定義します。

「IDE コマンドのカスタマイズ」ダイアログの [コマンド] タブをクリックすると [コマンド] ビューが現れます ([図 8.24 \(p210\)](#))。このビューで新しいコマンドグループやコマンドを作成します。

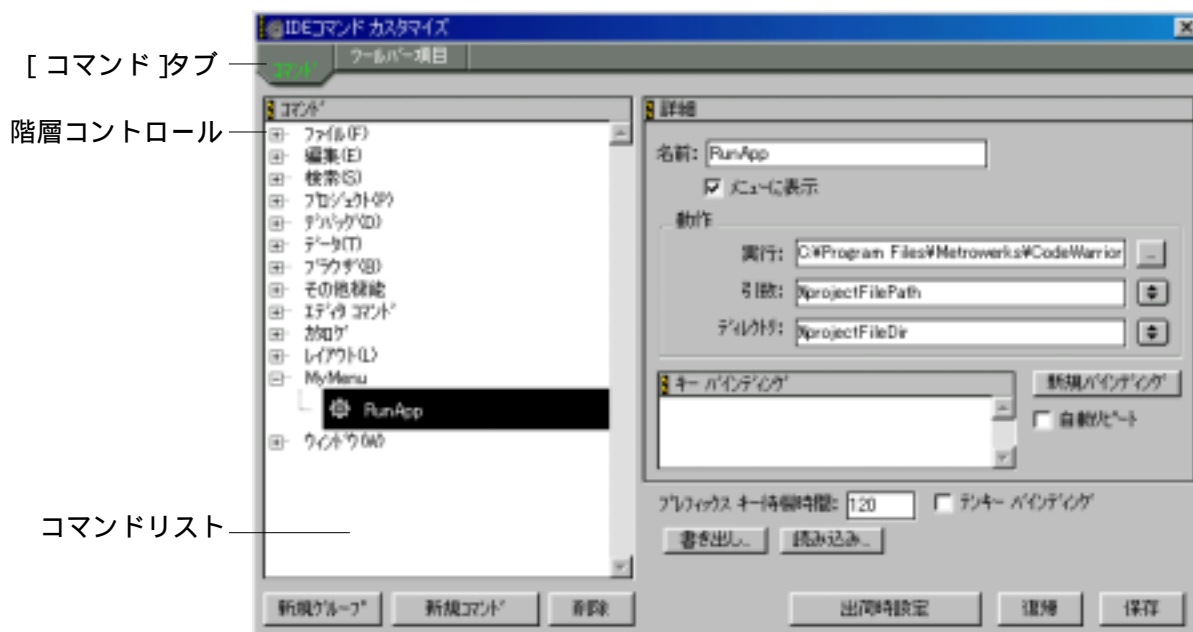
以下の内容について説明します。

[既存のコマンドグループとメニューコマンドを変更](#)

[独自のコマンドグループとメニューコマンドを追加](#)

[独自のコマンドグループやメニューコマンドを削除](#)

図 8.24 IDE コマンドのカスタマイズ：独自のコマンド



既存のコマンドグループとメニューコマンドを変更

「IDE コマンドのカスタマイズ」ダイアログのコマンドビューで既存のコマンドグループとメニューコマンドを調べることができます。コマンドビューにはコマンドリスト(図 8.24)があります。このリストの階層を使って個々のメニューコマンドをグループにまとめることができます。コマンドグループの横の階層コントロールをクリックするとグループの中身が拡張表示されます。

調べたい項目をコマンドリストで選択します。その項目についての情報が、「IDE コマンドのカスタマイズ」ダイアログの右側に表示されます。以下の情報が表示されます。

名前:[名前] フィールドは選択した項目の名前を表示します。フィールドが灰色で表示されている場合、その名前は変更できません。


メニューに表示:[メニューに表示] チェックボックスがオンの場合、選択した項目を CodeWarrior のメニューバーの特定に位置に表示します。例えば [RunApp] コマンドに対してこのチェックボックスをオンにすると(図 8.24)、CodeWarrior メニューバーの [MyMenu] コマンドグループの下にこのコマンドが現れます。オフの場合、[MyMenu] コマンドグループから [RunApp] コマンドが消えます。

動作:[動作] 欄はコマンドを選択したときに実行する動作についての情報を表示します。デフォルトのコマンドに対してはコマンドの種類(コマンドか階層メニューか)を示します。この欄で、カスタマイズしたコマンドを選択したときに実行するコマンドラインを指定します。

キーバインディング:[キーバインディング] リスト、[新規バインディング] ボタン、[自動リPEAT] チェックボックスの詳細は「[キーバインディングをカスタマイズ](#)」(p213) を参照してください。

独自のコマンドグループとメニューコマンドを追加

「IDE コマンドのカスタマイズ」ダイアログで新しいメニューコマンドを CodeWarrior のメニューバーに追加したり、コマンドグループに入れることができます。独自に作成したメニューコマンドでコマンドラインやアプリケーション、を実行することができます。

[コマンド] タブをクリックしてコマンドビュー ( 8.25) を開きます。ここで新しいメニューコマンドの作成や、コマンドグループへの追加を行います。コマンドリストの一番上の階層には既存のコマンドグループが表示されます。このグループの中身は既存のメニューコマンドです。

以下の内容について説明します。

[コマンドグループを作成](#)

[メニューコマンドを作成](#)

[メニューコマンドの動作を定義](#)

コマンドグループを作成

メニューコマンドを入れるコマンドグループを作成する手順を説明します。

1. [新規グループ] ボタンをクリックする

IDE は「New Group」という名前のコマンドグループを作ります。これをコマンドリストへ追加し、「IDE コマンドのカスタマイズ」ダイアログに情報を表示します。新規コマンドグループは既存のコマンドグループと一緒に、アルファベット順でコマンドリストに表示されます。

2. 新規コマンドグループの名前を入力する

「New Group」というデフォルトの名前を変更します。「IDE コマンドのカスタマイズ」ダイアログの [名前] フィールドに新しい名前を入力してください。

3. 新規コマンドグループの表示 / 非表示を設定する

[メニューに表示] チェックボックスは新規コマンドグループを CodeWarrior のメニューバーに表示するか否かをコントロールします。オンの場合、コマンドグループはメニューバーに表示されます。オフの場合、表示されません。

メニューコマンドを作成

以下は新規メニューコマンドを作成する手順です。

1. 新しいメニューコマンドを入れるグループを選択する

コマンドリストにあるコマンドグループを選択してください。メニューコマンドを入れるグループを作る方法は [「コマンドグループを作成」\(p211\)](#) を参照してください。

2. [新規コマンド] ボタンをクリックする

IDE は「New Command」という名前のメニューコマンドを作って選択したコマンドグループの中に入れ、アルファベット順で表示します。「IDE コマンドのカスタマイズ」ダイアログに新規メニューコマンドの情報を表示します。

3. 新規メニューコマンドの名前を入力する

新規コマンドのデフォルトの名前を変更します。「IDE コマンドのカスタマイズ」ダイアログの [名前] フィールドに新しい名前を入力してください。

4. 新規メニューコマンドの表示 / 非表示を設定する

[メニューに表示] チェックボックスは新規メニューコマンドを選択したコマンドグループの中に表示するか否かをコントロールします。オンの場合、メニューコマンドはコマンドグループの下に表示されます。オフの場合、表示されません。

5. 新規メニューコマンドの動作を定義する

詳細は[メニューコマンドの動作を定義](#)を参照してください。

メニューコマンドの動作を定義

新規メニューコマンドを作成 ([「メニューコマンドを作成」\(p211\)](#) を参照) すると、「IDE コマンドのカスタマイズ」ダイアログは [動作] 欄に 3 つのフィールドを表示します。ここで新しいコマンドの動作を定義します。

実行 : [実行] フィールドにアプリケーションを実行するコマンドラインを入力します。横のボタンをクリックしてダイアログからアプリケーションを選択することもできます ([図 8.24 \(p210\)](#))。

引数 : CodeWarrior から [実行] フィールドで指定したアプリケーションへ渡す引数を [引数] フィールドに入力します。横のポップアップメニューから引数を選択することもできます ([図 8.24 \(p210\)](#))。

ディレクトリ : CodeWarrior から [実行] フィールドで指定したアプリケーションへ渡すディレクトリを [ディレクトリ] フィールドに入力します。横のポップアップメニューからディレクトリを選択することもできます ([図 8.24 \(p210\)](#))。

独自のコマンドグループやメニューコマンドを削除

独自に作成したコマンドグループやメニューコマンドを削除することができます。削除したコマンドグループやメニューコマンドはメニューから消え、コマンドラインやアプリケーションを実行することはできません。

ヒント : 作成したコマンドグループやメニューコマンドを一時的に削除する方法に、設定のエクスポートがあります。これならば再度コマンドを使うときに設定を再構築する必要はありません。IDE に設定をインポートして再利用することができます。[「キーバインディングとコマンドをエクスポート」\(p217\)](#) を参照してください。

以下はコマンドグループやメニューコマンドを削除する手順です。

1. 削除するコマンドグループかメニューコマンドを選択する
必要に応じて階層コントロールをクリックしてグループを拡張表示します。
2. [削除] ボタンをクリックする
[削除] ボタンをクリックすると、選択したコマンドグループか、メニューコマンドがコマンドリストから消えます ([図 8.25](#))。

キーバインディングをカスタマイズ

CodeWarrior IDE のメニュー、キーボード、エディタコマンド用のキーボードショートカットをカスタマイズすることができます。ほぼすべてのコマンドをキーボードショートカットに割り当てることができます。コマンドを実行するとき、そのキーバインディングを入力します。「IDE コマンドのカスタマイズ」ダイアログでデフォルトのキーバインディングを変更します。

例えば「[上へ 1 行移動](#)」のキーバインディングを キーから Shift + キーに変えるとしします。「IDE コマンドのカスタマイズ」ダイアログでそのコマンドのキーバインディングを変更すると、このキーボードショートカットを変更できます。

「IDE コマンドのカスタマイズ」ダイアログの[コマンド]タブをクリックして[コマンド]ビューを表示します ([図 8.25](#))。ここでメニューコマンド、エディタやその他の動作のキーバインディングを変更します。特殊なプレフィックスキーを指定することもできます。デフォルトのキーバインディングリストは「[デフォルトのキーバインディング](#)」(p481) を参照してください。

以下の内容について説明します。

[キーバインディングの制限](#)

[キーバインディングを変更](#)

[キーバインディングを追加](#)

[キーバインディングを削除](#)

[キーバインディングに自動リピートを設定](#)

[キーバインディングとコマンドをエクスポート](#)

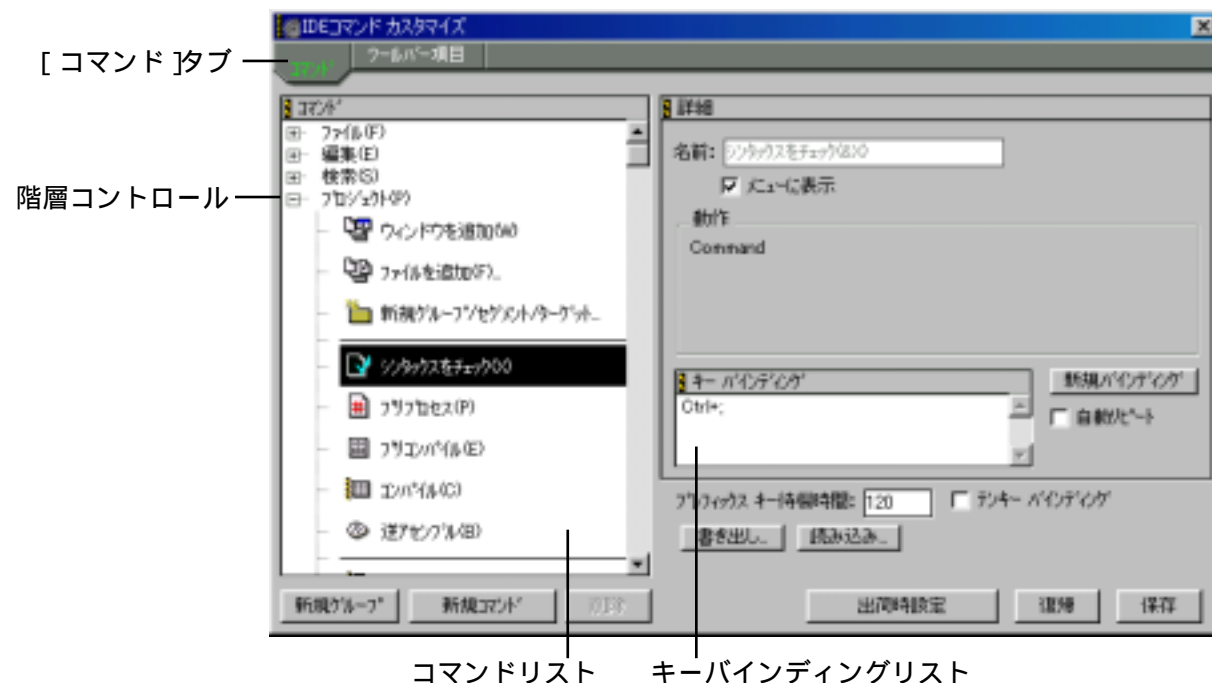
[コマンドとキーバインディングをインポート](#)

[プレフィックスキー](#)

[クオートキー](#)

[プレフィックスキーの待機時間を設定](#)

図 8.25 IDE コマンドのカスタマイズ：キーバインディング



キーバインディングの制限

キーバインディングをカスタマイズする際に利用できるキーとできないキーがあることに注意してください。キーバインディングには以下の制限があります。

Escape および Space キーは、キーバインディングでは常に無効です。

Function および Clear キーはキーバインディングの作成に利用できます。

Return および Tab キーは少なくとも Ctrl または Shift キーと一緒に使わなくてはなりません。キーバインディングの 2 番目のキーにはこの制限はありません。

キーバインディングを変更

[コマンド] ビューのコマンドリストは、IDE のコマンドをカテゴリごとに表示します ([図 8.25 \(p214\)](#))。階層コントロールをクリックするとカテゴリの中身が拡張表示されます。

以下はキーバインディングを変更する手順です。

1. 変更するコマンドをコマンドリストから選択する

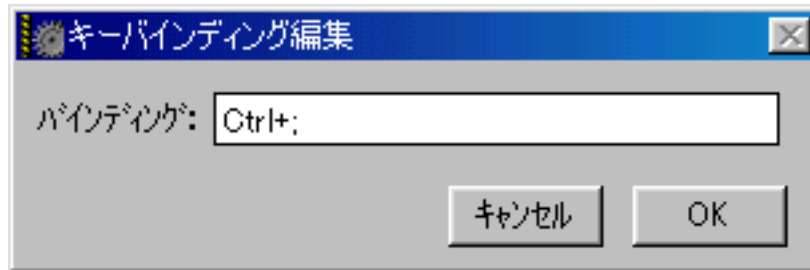
カテゴリを拡張してコマンドを見るには階層コントロールをクリックしてください。

注意： キーバインディングにテンキーパッドのキーを使うには、「IDE コマンドのカスタマイズ」ダイアログの [テンキーバインディング] チェックボックスをオンにしてください。

2. 変更するコマンドをキーバインディングリストでダブルクリックする

キーバインディングリスト ([図 8.25 \(p214 \)](#)) でコマンドをダブルクリックすると、IDE は「キーバインディング編集」ダイアログ ([図 8.26](#)) を表示します。

図 8.26 「キーバインディング編集」ダイアログ



3. 選択したコマンドに対するキーバインディングを入力する

例えばキーバインディングを Ctrl + 8 に変更する場合、Ctrl キーと 8 キーを同時に押します。

入力したばかりのキーバインディングを変更する、または入力を間違った場合、「キーバインディング編集」ダイアログの [キャンセル] ボタンをクリックします。「IDE コマンドのカスタマイズ」ダイアログへ戻ります。

4. 「キーバインディング編集」ダイアログの [OK] ボタンをクリックする

「IDE コマンドのカスタマイズ」ダイアログのキーバインディングリストに新しいキーバインディングが表示されます。

5. 「IDE コマンドのカスタマイズ」ダイアログの [保存] ボタンをクリックして変更を保存する

IDE で新しいキーバインディングが利用できます。

キーバインディングを追加

「IDE コマンドのカスタマイズ」ダイアログではキーバインディングを変更するだけでなく、コマンドに対してさらにキーバインディングを追加することもできます。

以下はキーバインディングを追加する手順です。

1. 新しいキーバインディングを追加するコマンドをコマンドリストから選択する

カテゴリを拡張してコマンドを見るには階層コントロールをクリックしてください。

注意： キーバインディングにテンキーパッドのキーを使うには、「IDE コマンドのカスタマイズ」ダイアログの [テンキーバインディング] チェックボックスをオンにしてください。

2. [新規バインディング] ボタンをクリックする

[新規バインディング] ボタン ([図 8.25 \(p214 \)](#)) をクリックすると IDE は「キーバインディング編集」ダイアログ ([図 8.26 \(p215 \)](#)) を表示します。

3. 選択したコマンドに対するキーバインディングを入力する

例えばキーバインディングを Ctrl+ 8 に変更する場合、Ctrl キーと 8 キーを同時に押します。

入力したばかりのキーバインディングを変更する、または入力を間違った場合、「キーバインディング編集」ダイアログの [キャンセル] ボタンをクリックします。「IDE コマンドのカスタマイズ」ダイアログへ戻ります。

4. 「キーバインディング編集」ダイアログの [OK] ボタンをクリックする

「IDE コマンドのカスタマイズ」ダイアログのキーバインディングリストに新しいキーバインディングが表示されます。

5. 「IDE コマンドのカスタマイズ」ダイアログの [保存] ボタンをクリックして変更を保存する

IDE で新しいキーバインディングが利用できます。

キーバインディングを削除

CodeWarrior IDE で使いたくないキーバインディングを削除できます。削除したキーバインディングは無効になります。

以下はキーバインディングを削除する手順です。

1. 削除したいキーバインディングを含むコマンドをコマンドリストから選択する

カテゴリを拡張してコマンドを見るには階層コントロールをクリックしてください。

2. キーバインディングリストから削除するキーバインディングを選択する

キーバインディングリストを [図 8.25 \(p214 \)](#) に示します。

3. Backspace キーを押す

選択したキーバインディングがキーバインディングリストから消えます。

4. 「IDE コマンドのカスタマイズ」ダイアログの [保存] ボタンをクリックして変更を保存する

削除したキーバインディングは無効になります。

キーバインディングに自動リピートを設定

[自動リピート] チェックボックス ([図 8.25 \(p214 \)](#)) はキーバインディングリストの項目の自動的な挙動を変えます。

[自動リピート] チェックボックスがオンの場合、キーバインディングを押している間、そのコマンドが自動的に繰り返されます。オフの場合、コマンドを実行するたびにキーバインディングを押す必要があります。

例えば [[次を検索](#)] コマンドのキーバインディングに対して [自動リピート] をオンにすると、キーを押し続けているだけで開いているファイルのテキストをすべて検索できます。テキストを高速で検索できます。オフにすると、毎回キーを押して離さなくてはなりません。

キーバインディングとコマンドをエクスポート

コマンドとキーバインディングをファイルに保存しておく、それをいつか IDE へインポートして使うことができます。「IDE コマンドのカスタマイズ」ダイアログの [書き出し] ボタンをクリックすると「Export」ダイアログが現れます。ここでコマンドとキーバインディングを含むファイルの保存先を指定して [保存] ボタンをクリックします。

ヒント： コマンドとキーバインディングを含むファイルには拡張子 `.mkb` を付けてください（例：MyCmsKbs.mkb）。命名規約に従うことにより、ファイルを素早く識別できます。Windows 版の CodeWarrior IDE は拡張子でファイルを識別します。

コマンドとキーバインディングをインポート

事前にエクスポートしたファイルからコマンドとキーバインディングをインポートするには、「IDE コマンドのカスタマイズ」ダイアログの [読み込み] ボタンをクリックします。「開く」ダイアログでコマンドとキーバインディングのファイルを指定して開いてください。

プレフィックスキー

プレフィックスキーは複数のキーバインディングを使います。Emacs テキストエディタでファイルを保存するには `Ctrl + X` キーの次に `Ctrl + S` キーを押します。

CodeWarrior IDE で Emacs のキーバインディングをエミュレートするには、プレフィックスキーに `Ctrl + X` を割り当て、[[保存](#)] コマンドのキーに `Ctrl + X Ctrl + S` を割り当てます。

プレフィックスキーの後のキーを待機する時間を調整できます。詳細は「[プレフィックスキーの待機時間を設定](#)」(p219) を参照してください。

クオートキー

クオートキーとは、単純な機能を持つ特別なプレフィックスキーです。普通の文字キーに（修飾キーなしで）コマンドを割り当てることができ、またその文字キーを入力に使うことも可能になります。

通常、コマンドと等価なキーは 2 つのキーで構成されます。修飾キー（`Ctrl` キーなど）とその他の文字キーです。CodeWarrior では修飾キーは必要ありません。数字の 1 のキーを（修飾キーなしで）コマンドに割り当てることができます。

しかしこうすると、1 をエディタに入力することはできません。1 はコマンドとみなされるためです。この問題をクオートキーが解決します。

どんなキーでもクオートキーとすることができます。「[クオートキーを割り当てる](#)」(p218)を参照してください。クオートキーはテキストのシンボルを引用するキーではありません。

CodeWarrior はクオートキーの直後に入力されたキーをコマンドではなく通常の入力と解釈します。

1 のキーにコマンドを割り当て、波(~)キーをクオートキーにしたと仮定します。コマンドを実行するには、1 キーを押します。エディタに 1 と入力するには、~ キー、次に 1 キーを押します。~ キーを入力するには、~ キーを 2 回押します。

警告! クオートキーが効果を持つのは、その直後のキーまたはキーバインディングの入力だけです。文字として入力したいキーの直前に一度ずつクオートキーを押してください。

クオートキーを割り当てる

以下はクオートキーを割り当てる手順です。

1. [その他機能] の横の階層コントロールをクリックする
[その他機能] 項目 ([図 8.27](#)) は「IDE コマンドのカスタマイズ」ダイアログのコマンドリストにあります。
2. [クオートキー] 項目を選択する
[クオートキー] 項目は [その他機能] コマンドグループにあります。

注意： キーバインディングにテンキーバッドのキーを使うには、「IDE コマンドのカスタマイズ」ダイアログの [テンキーバインディング] チェックボックスをオンにしてください。

3. キーバインディングリストの [新規バインディング] ボタンをクリックする
キーバインディングリスト ([図 8.27](#)) の横の [新規バインディング] ボタンをクリックすると、「キーバインディング」ダイアログが現れます。
4. 「キーバインディング編集」ダイアログでクオートキーを入力する
入力したキーはダイアログに表示されます。入力したばかりのキーバインディングを変更する、または入力を間違った場合、「キーバインディング」ダイアログの [キャンセル] ボタンをクリックします。「IDE コマンドのカスタマイズ」ダイアログへ戻ります。

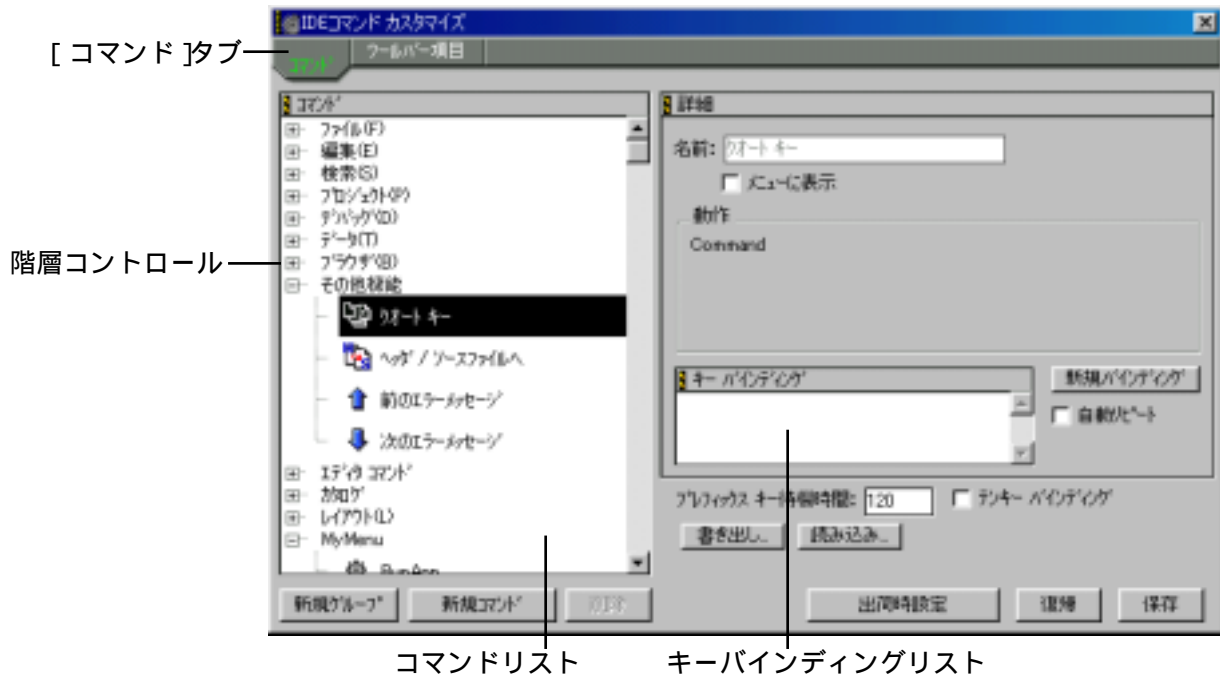
5. 「キーバインディング」ダイアログの [OK] ボタンをクリックする

キーバインディングリストに新しいクオートキーのプレフィックスが表示されます。

プレフィックスキーの待機時間を設定

[プレフィックスキー待機時間] フィールドで、プレフィックスキーを押してから 2 番目のキーを押すまで IDE が待機する時間を設定します。大きい値を指定すると、2 番目のキーが押すまで IDE がより長い時間待機することを意味します。

図 8.27 「IDE コマンドのカスタマイズ」ダイアログ：クオートキープレフィックス



この値の単位は「チック」で、1 チックは 1/60 秒 (16.67 ミリ秒) です。有効値は 1 から 999 までです。デフォルトは 120 です。

ツールバーをカスタマイズ

ツールバーにはいくつかの項目 (アイコンボタン) があります。各項目は対応するメニューコマンドを表します。ボタンをクリックするとそのコマンドが実行されます。ツールバーにメニューコマンド以外の動作を実行するボタンを置くことができます。以下に CodeWarrior プロジェクトウィンドウのツールバー (図 8.28) を示します。

図 8.28 プロジェクトウィンドウのツールバー



ここではツールバーを詳しく説明します。

[ツールバーの種類](#)

[ツールバーの項目](#)

[ツールバーを表示する / 隠す](#)

[ツールバーを変更](#)

[ツールバーをデフォルトの設定に戻す](#)

ツールバーの種類

CodeWarrior には 2 種類のツールバーがあります。

フロートツールバー（グローバルツールバー、メインツールバーとも呼ばれます）。常に使用できます。

ウィンドウ内部に表示されるツールバー（プロジェクトウィンドウやブラウザウィンドウにあります）。

これは重要な違いです。CodeWarrior IDE のウィンドウメニューのツールバーサブメニューにはツールバーを表示する、隠す、消去またはリセットするメニューコマンドがあります。これでフロートツールバーとウィンドウツールバーを区別できます。

あるウィンドウのツールバーへの変更はそのウィンドウすべてに適用されます。例えば、あるエディタウィンドウのツールバーはすべてのエディタウィンドウに共通します。あるエディタウィンドウのツールバーを変更すると、すべてのエディタウィンドウのツールバーに適用されます。[「ツールバーを変更」\(p222\)](#) を参照してください。

各ツールバーの設定はいつでもデフォルトに戻すことができます。[「ツールバーをデフォルトの設定に戻す」\(p223\)](#) を参照してください。

ウィンドウツールバーに関係のあるメニューコマンドを選択すると、アクティブなウィンドウにあるツールバーに反映されます。詳細は [「ツールバーサブメニュー」\(p480\)](#) を参照してください。

ツールバーの項目

ツールバーには 4 種類の項目があります。

コマンド：IDE メニューのコマンドを実行するボタン

コントロール：IDE ポップアップメニューのボタン（ドキュメントの設定、関数、ヘッダ、マーカ、バージョン管理）およびカレントターゲットの変更ボタン

その他機能：その他のボタン（ファイルパス表示欄など）

ウィンドウで [ツールバー項目] タブをクリックするとツールバービュー（[図 8.29](#)）が表示されます。このビューでツールバーに新しい項目を追加することができます。詳細は [「ツールバー項目を追加」\(p222\)](#) を参照してください。

ツールバーを表示する / 隠す

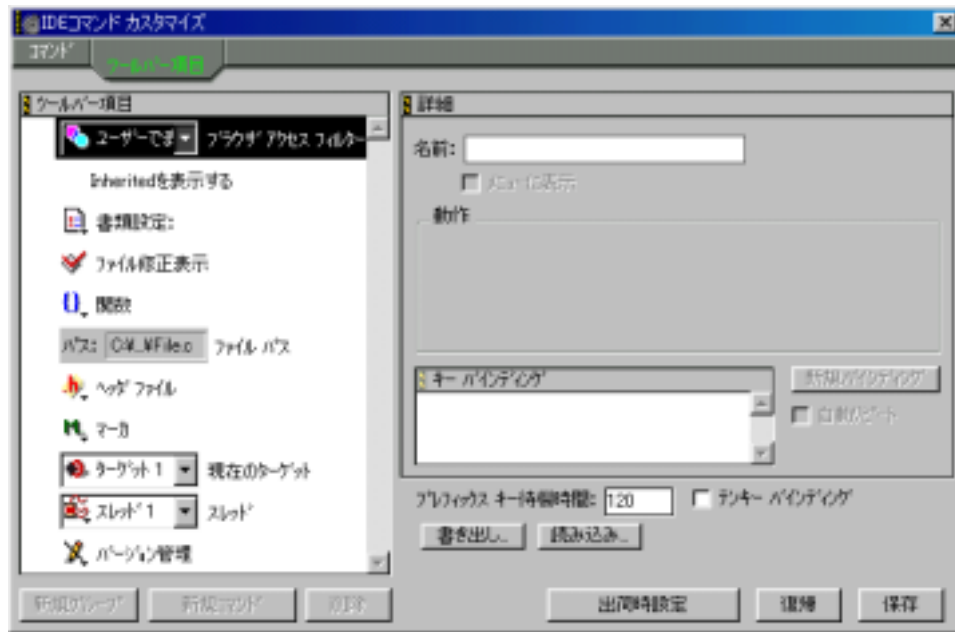
[ツールバーサブメニュー](#)でツールバーを表示したり、隠したりできます。隠してもツールバーの構成は変わりません。

ツールバーの種類について説明します。

[メインツールバー](#)

[ウィンドウのツールバー](#)

図 8.29 「IDE コマンドのカスタマイズ」ダイアログ：ツールバー項目ビュー



メインツールバー

メインツールバー（フロートツールバーとも呼ばれます）を表示するには、ウィンドウメニューのツールバーサブメニューから [メインツールバーを表示] を選択します。メインツールバーを隠すには、ウィンドウメニューのツールバーサブメニューから [メインツールバーを隠す] を選択します。ツールバーをデフォルト設定に戻す方法は「[ツールバーをデフォルトの設定に戻す](#)」(p223) を参照してください。

ウィンドウのツールバー

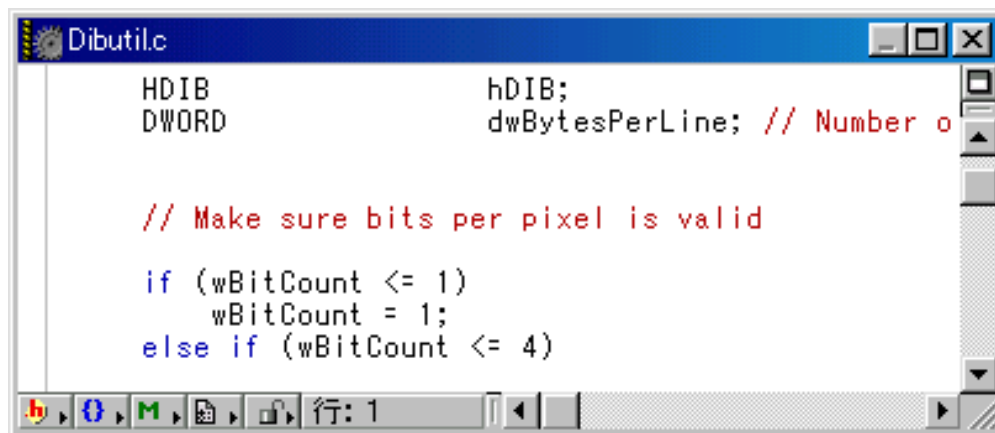
アクティブなウィンドウのツールバーを隠すには、ウィンドウメニューのツールバーサブメニューから [ウィンドウツールバーを隠す] を選択します。ウィンドウツールバーのあった場所にはその他のコンポーネントが移動します。同じタイプのウィンドウからツールバーが隠れます。

アクティブなウィンドウで隠したツールバーを表示するには、ウィンドウメニューのツールバーサブメニューから [ウィンドウツールバーを表示] を選択します。コンポーネントの位置がずれて、ツールバーが表示されます。同じタイプのウィンドウでツールバーが表示されます。

エディタウィンドウでツールバーを隠すと、デフォルトのツールがウィンドウの下部に表示されます ([図 8.30](#))。エディタウィンドウのツールバーを表示すると、エディタウィンドウの上部に表示されます ([図 5.1 \(p96\)](#))。

ツールバー拡張ボタンでもエディタウィンドウのツールバーを隠すことができます。「[ウィンドウコントロールを表示](#)」(p101) を参照してください。

図 8.30 ツールバーを隠したエディタウィンドウ



ツールバーを変更

以下の方法でツールバーを変更できます。

[ツールバー項目を追加](#)

[ツールバー項目を削除](#)

[すべてのツールバー項目を除去](#)

ツールバーに追加または削除できる項目には制限がある場合もあります。詳細は「[ツールバー項目を追加](#)」(p222)、[「ツールバー項目を削除」](#)(p223) を参照してください。

ツールバーに加えた変更は、そのウィンドウのツールバーに適用されます。例えばプロジェクトウィンドウのツールバーをカスタマイズするとアクティブなプロジェクトウィンドウだけでなく、既に開かれたものなど、すべてのプロジェクトウィンドウのツールバーが変わります。

ツールバー項目を追加

ツールバー項目リストから項目をドラッグしてツールバーに追加することができます。ツールバー項目リストは「IDE コマンドのカスタマイズ」ダイアログのツールバー項目ビューにあります。

以下はツールバー項目リストからツールバーへ項目を追加する手順です。

1. ツールバーへ追加する項目を選択する

追加先のツールバーが表示されていることを確認してください。

2. ツールバー項目リストからツールバーへ項目をドラッグする

追加先のツールバーが項目を受け入れ可能ならば、ツールバーに点滅する四角形が現れます。これは項目の挿入位置を示します。項目を追加できない場合、四角形は表示されません。

3. 希望の位置で項目をドロップする

ドロップした後、点滅していた位置に項目が現れます。

ツールバーに項目を追加できない理由を以下に示します。

ツールバーがいっぱいです。

項目が既にツールバーにあります。

ウィンドウツールバーに追加できるコマンドは、現在アクティブなウィンドウで利用できるメニューコマンドだけです。

[ドキュメント設定] [関数] [ヘッダファイル] [マーカー] [バージョン管理] [ファイル修正表示] [ファイルパス] を追加できるのはエディタウィンドウのツールバーだけです。

[現在のターゲット] を追加できるのはプロジェクトウィンドウのツールバーだけです。

ツールバー項目を削除

ツールバーから不要な項目を除くことができます。ツールバー上の項目を取り除くには、その項目の上で Ctrl + 右クリックします。ポップアップメニューが現れるので、そこから [ツールバー項目を削除] を選択してください。ツールバー上からその項目が消えます。

すべてのツールバー項目を除去

ウィンドウメニューのコマンドでツールバーを消去できます。ツールバーを消去してから好みに合わせて作成することもできます。

メインのツールバーを消去するには、ウィンドウメニューのツールバーサブメニューから [メインツールバーを消去] を選択します。

現在のウィンドウからツールバーを消去するにはウィンドウメニューのツールバーサブメニューから [ウィンドウツールバーを消去] を選択します。

ウィンドウの機能に不可欠な項目は [ウィンドウツールバーを消去] でも除去できません。

注意： フロートツールバーが隠れているときは消去できません。まずフロートツールバーを表示しなくてはなりません。[「ツールバーを表示する / 隠す」\(p221\)](#) を参照してください。

ツールバーをデフォルトの設定に戻す

ウィンドウメニューのコマンドでツールバーを元（デフォルト）の設定に戻すことができます。

メインのツールバーに戻すには、ウィンドウメニューのツールバーサブメニューから [メインツールバーをリセット] を選択します。

現在のウィンドウのツールバーをデフォルトに戻すには、ウィンドウメニューのツールバーサブメニューから [ウィンドウツールバーをリセット] を選択します。

注意： フロートツールバーが隠れているときはそれらをリセットできません。まずフロートツールバーを表示しなくてはなりません。[「ツールバーを表示する / 隠す」\(p221\)](#)を参照してください。

第 9 章 ターゲットのオプション設定

この章ではターゲット設定ダイアログについて説明します。ターゲット設定で IDE がどのようにプロジェクトのビルドターゲットを扱うのかを指定します。

ターゲット設定ダイアログでは、トピックごとにオプションが設定パネルにまとめられています。例えばファイルを検索するパスを設定するためのパネルなどがあります。

ビルドターゲットの設定を終えて、好みの設定をしたプロジェクトをステーションナリとして保存することができます。詳細は「[独自のプロジェクトステーションナリの作成](#)」(p45)を参照してください。

ここでは以下の内容について説明します。

[ターゲット設定の解説](#)

[設定パネルを選択](#)

ターゲット設定の解説

ターゲット設定ダイアログを開くには、編集メニューの[[Target の設定](#)]を選択します。プロジェクトファイルが開いているとき、このコマンドの名前にビルドターゲット名が含まれます。例えばカレントターゲットが ANSI Console Win32 の場合、[[Target の設定](#)] は、[ANSI Console Win32 の設定] となります。

[設定パネル](#)

[ダイアログのボタン](#)

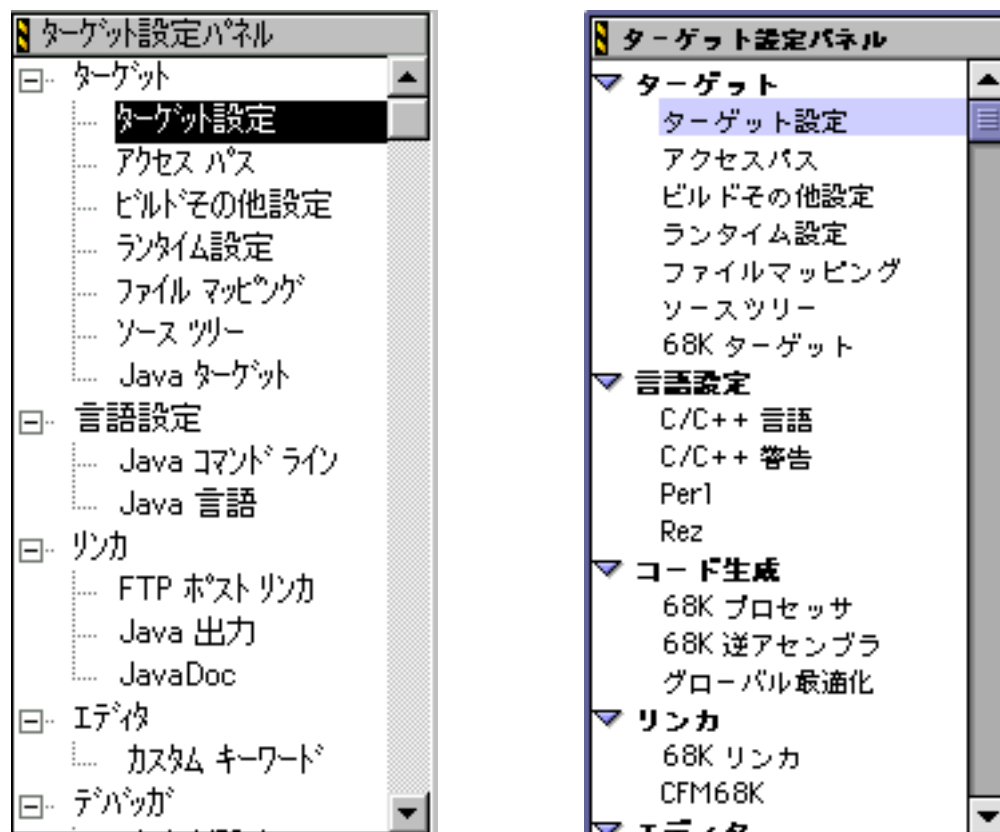
設定パネル

ターゲット設定ダイアログの左側には設定パネルの階層リストがあります。このリストで選択した設定パネルが右側に表示されます。利用可能な設定パネルは CodeWarrior 製品の種類、カレントビルドターゲットによって異なります。

設定パネルを見るには、ターゲット設定ダイアログのリストで名前をクリックするか、矢印キーで選択します ([図 9.1](#))。

各設定パネルにはユーザーが設定するオプションがあります。設定したオプションは現在アクティブなプロジェクトのカレントビルドターゲットに適用されます。設定パネルの変更を適用する、または破棄する方法は「[ダイアログのボタン](#)」を参照してください。

図 9.1 設定パネルを選択



ビルドターゲットを好みの設定に変更して、プロジェクトをステーションリとして保存することができます。このステーションリから新規プロジェクトを作成できます。詳細は「[独自のプロジェクトステーションリの作成](#)」(p45)を参照してください。

ダイアログのボタン

ターゲット設定ダイアログには、パネルのオプションの適用方法をコントロールするボタンがあります。

ここでは、以下の項目を説明します。

[変更を破棄](#)

[出荷時設定](#)

[復帰](#)

[保存](#)

変更を破棄

ターゲット設定ダイアログでオプション設定を変更した後にダイアログを閉じようとする
と、警告ダイアログ (図 9.2) が表示されます。変更を保存してダイアログを閉じるには

[保存] ボタンをクリックします。変更を破棄してダイアログを閉じるには [保存しない] ボタンをクリックします。変更を保存せずにターゲット設定ダイアログへ戻るには [キャンセル] ボタンをクリックします。

図 9.2 設定パネルの確認ダイアログ



出荷時設定

[出荷時設定] ボタンをクリックすると、設定パネルのオプションがデフォルトの設定に戻ります。他の設定パネルのオプションには影響しません。現在表示中のパネルのオプションの設定だけがリセットされます。

復帰

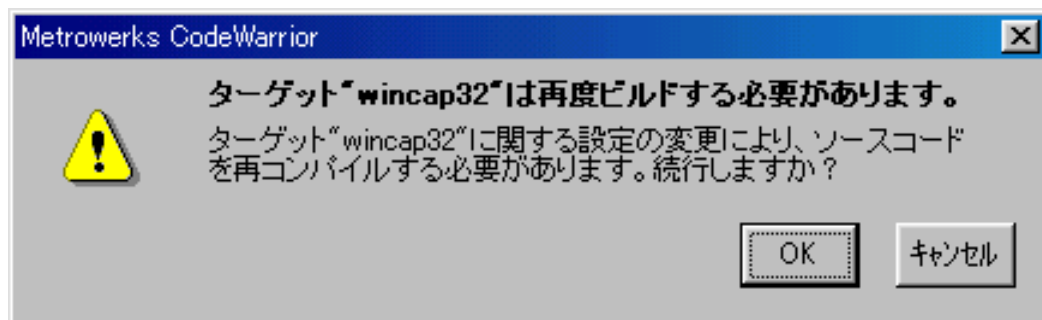
[復帰] ボタンは、現在表示中の設定パネルの状態を、そのパネルを最後に保存したときの設定に戻します。これは、パネルに変更を加えた後で変更を適用したくない場合に便利です。

保存

[保存] ボタンは、任意の設定パネル内で加えた変更を保存します。プロジェクトの再コンパイルが必要となるオプションを変更した場合、[図 9.3](#) のようなダイアログが表示されます。変更を保持するか否かを [OK] または [キャンセル] ボタンで指定します。

ターゲット設定ダイアログを閉じた後、CodeWarrior IDE は保存した設定を使ってターゲットをビルドします。

図 9.3 変更を確認するダイアログ



設定パネルを選択

ここでは、ビルドターゲットの設定について説明します。CodeWarrior IDE の設定をさまざまに変更して、プロジェクトおよびターゲットを好きな方法でビルドすることができます。

ターゲット設定ダイアログと設定パネルを開く方法は「[ターゲット設定の解説](#)」(p225) を参照してください。

CodeWarrior には多数の設定パネルがあります。利用できる設定パネルはオペレーティングシステムやターゲットのプロセッサ、プログラム言語によって異なります。

例えば x86 プロジェクトの場合、Motorola 68K 用の設定パネルは表示されません。Pascal 言語を使用できないターゲットでは、Pascal に関する設定パネルは表示されません。

CodeWarrior IDE はオプションの変化による再コンパイルを最小限に抑えます。例えば、リソースコンパイラの環境設定を変更しても、再コンパイルが必要になるのはリソースコンパイラのソースだけです。再コンパイルの回数を減らし、リンクスピードを上げます。

このマニュアルではオペレーティングシステム、マイクロプロセッサ、言語についての説明はしません。[表 9.1](#) に特定の OS、言語、設定パネルについて参照すべきマニュアルを示します。この表は CodeWarrior 製品のすべてのターゲットや設定パネルを網羅するものではなく、CodeWarrior で利用可能な共通の設定パネルだけを記しています。

設定パネルには、特定のビルドターゲットだけに影響するものや、すべてのビルドターゲットに適用できるものがあります。ここでは以下の設定パネルについて説明します。

[ターゲットの設定](#)

[コード生成の設定](#)

[エディタの設定](#)

[デバッグの設定](#)

表 9.1 各設定パネルの参照マニュアル

設定パネル	参照マニュアル
x86 ターゲット Windows RC x86 プロセッサ x86 リンカ x86 例外	Targeting Windows
Java コマンドライン Java 言語 Java リンカ	Targeting Java

設定パネル	参照マニュアル
68K ターゲット Rez 68K プロセッサ 68K 逆アセンブラ 68K リンカ CFM68K	Targeting Mac OS
PPC ターゲット PPCAsm Rez PPC プロセッサ PPC 逆アセンブラ PPC リンカ PPC PEF	Targeting Mac OS
C/C++ 言語 C/C++ 警告	C Compilers Reference
Pascal 言語 Pascal 警告	Pascal Compilers Reference

ターゲットの設定

以下の設定パネルについて説明します。

[ターゲット設定](#)

[アクセスパス](#)


[ビルドその他設定](#)

[ランタイム設定](#)

[ファイルマッピング](#)

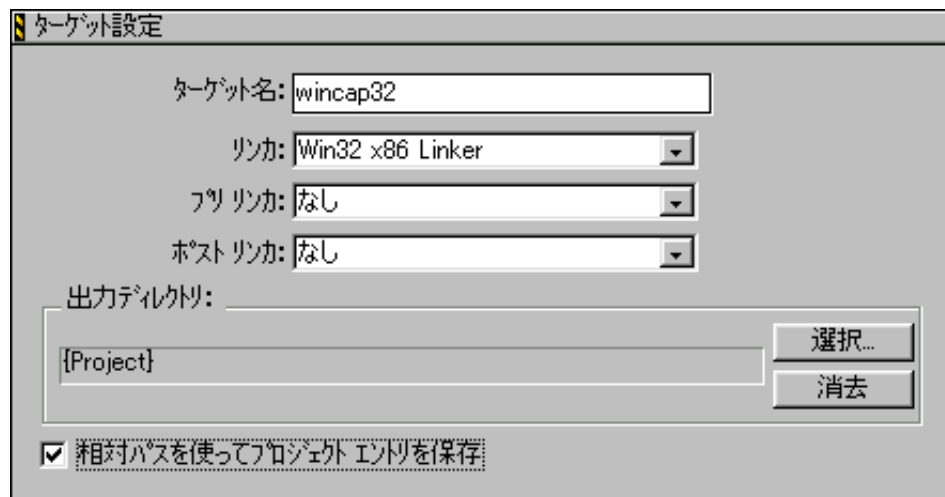
[ソースツリー](#)

ターゲット設定

ターゲット設定パネル ( 9.4) は CodeWarrior で最も重要なパネルです。ここでターゲットとするオペレーティングシステムやマイクロプロセッサを指定します。

ターゲット設定ダイアログを開いて設定パネルを選択する方法は「[ターゲット設定の解説](#)」(p225) を参照してください。

図 9.4 ターゲット設定パネル



ターゲット設定パネルで現在のビルドターゲットの名前、リンカプラグインを設定します。リンカを選択することにより、ターゲットオペレーティングシステムやプロセッサが指定されます。これによって利用可能な設定パネルが変わります。

選択されたリンカによって表示できる設定パネルが異なるため、コンパイラやリンカなどの設定よりも先にリンカを指定します。

このパネルでビルドターゲットを完全に変更することも可能です。ターゲットを変更すると、そのプロジェクトファイルに含めるライブラリも変えなくてはなりません。新しいターゲットを選択しても、ライブラリは自動的に変更されません。このためターゲットを変更するには不要なライブラリを削除し、必要なものを追加するように注意が必要です。

ステーションナリから新規プロジェクトを作成する場合、必要なライブラリやファイルは自動的に含まれます。古いプロジェクトからソースファイルを追加するだけです。ステーションナリの作成については「[新規プロジェクトを作成](#)」(p40) を参照してください。

ターゲット名

[ターゲット名] フィールドにカレントビルドターゲットの名前を入力、または変更します。プロジェクトウインドウのターゲットビューで設定した名前が表示されます。

これは最終的な出力ファイルの名前ではありません。ターゲットに対して個人的に付ける名前です。最終的な出力ファイルの名前は通常リンカ設定パネルで指定します。

リンカ

[リンカ] ポップアップメニューでリンカを選択します。

利用可能なリンカ、ポストリンカについては各『Targeting』マニュアルを参照してください。選択できるリンカは CodeWarrior 用のプラグインリンカの種類によっても異なります。

例えば CodeWarrior Pro には 68K および PowerPC の Mac OS 用リンカ、x86 の Win32 用リンカ、Java 仮想マシンのリンカが含まれます。CodeWarrior for PlayStation OS には MIPS 上で動作する PlayStation OS 用のリンカ、ポストリンカが含まれます。

注意： 選択したリンクのファイルマッピングについては「[ファイルマッピング](#)」(p239)を参照してください。ファイルマッピングで IDE が認識できるファイルを指定します。

プリリンク

一部のビルドターゲットには、最終の実行可能ファイルにオブジェクトコードをリンクする前に、追加作業を行うプリリンクがあります。プリリンクについては各『Targeting』マニュアルを参照してください。

ポストリンク

一部のビルドターゲットには、最終の実行可能ファイルを作成するための追加作業（データ形式の変換など）を行うポストリンクがあります。各『Targeting』マニュアルを参照してください。

出力ディレクトリ

[出力ディレクトリ] フィールドでリンク済みの最終出力ファイルを書き込むディレクトリを選択します。デフォルトはプロジェクトファイルと同じ位置です。[選択] ボタンをクリックしてディレクトリを変更します。現在の位置を消去するには[消去] ボタンをクリックします。

相対パスを使ってプロジェクトエントリを保存

[相対パスを使ってプロジェクトエントリを保存] オプションがオンの場合、IDE はプロジェクトの位置と相対アクセスパス情報を記憶します。これにより同名のファイルを区別します。ファイルを検索するとき、IDE は「アクセスパス」と各プロジェクトのパスを使用します。

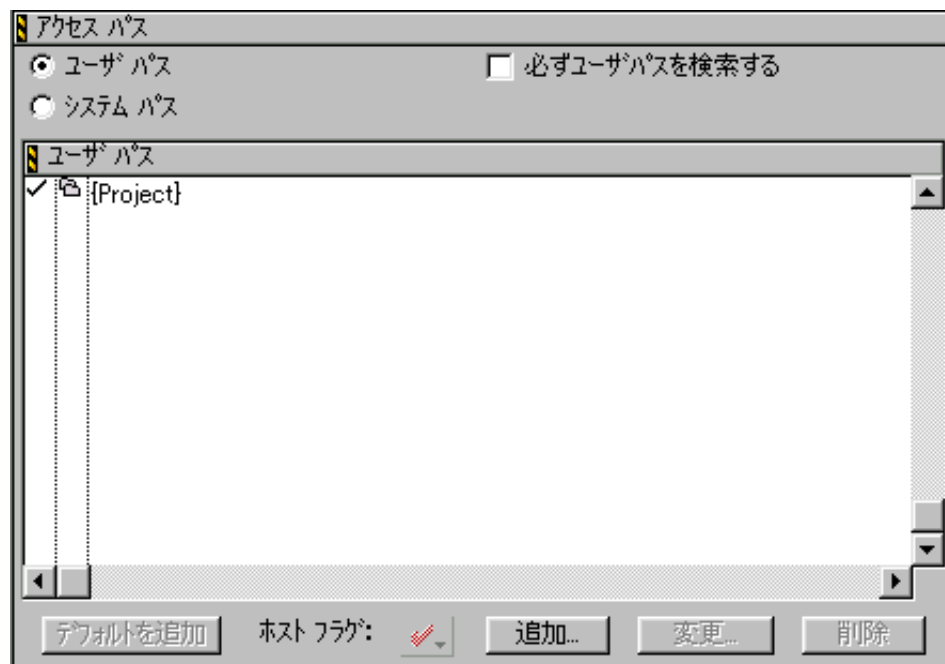
オフの場合、IDE はプロジェクトにプロジェクトの名前だけを記憶します。ファイルを検索するとき、IDE は「アクセスパス」だけを使用します。

詳細は「[ファイルパスを再検索](#)」(p467)、「[プロジェクト内のパスをリセット](#)」(p467) を参照してください。

アクセスパス

CodeWarrior IDE がプロジェクトのコンパイルおよびリンク中に検索する追加のアクセスパスを定義しておく必要がある場合は、アクセスパス設定パネル（[図 9.5](#)）を使います。

図 9.5 アクセスパス設定パネル



ターゲット設定ダイアログを開いてアクセスパス設定パネルを選択する方法は「[ターゲット設定の解説](#)」(p225)を参照してください。

[[ユーザパス欄](#)] または [[システムパス欄](#)] でフォルダ名の左横にフォルダアイコンが表示されているとき、CodeWarrior IDE はこのパスのリカーシブ検索を行います。つまり、そのフォルダとすべてのサブフォルダを検索します。

[[ユーザパス欄](#)] または [[システムパス欄](#)] で、パスの左のフォルダアイコンをクリックすると、全サブディレクトリのリカーシブ検索をオフにできます。フォルダが表示されているれば、リカーシブ検索はオンになっています。フォルダが表示されていないければ、サブディレクトリは検索されません。

ヒント： パスのリカーシブ検索をオフにして、ファイルを含む各ディレクトリへのパスを、それぞれ [システムパス] 欄または [ユーザパス] 欄に追加すれば、プロジェクトのコンパイル時間を短縮できます。

[ユーザパス] 欄または [システムパス] 欄のアイコンの左のチェックマークをクリックすると、現在の CodeWarrior ホストコンピュータからのパス検索を無効にできます。カレントホストコンピュータはチェックマークのあるパスを検索します。チェックマークがない場合、カレントホストコンピュータはそのパスを無視します。

プロジェクトのファイルまたはライブラリがどちらのデフォルトアクセスパスにもなければ、プロジェクトのコンパイル、リンク、実行の際に CodeWarrior IDE はこれらのファイル

を見つけることができません。検索先を設定するために、アクセスパスを追加する必要があります。

Resource.frk ファイルは検索の対象にはなりません。

ヒント： フォルダ名を括弧でくくると、そのフォルダとサブフォルダは検索から除外されます。例えば、Gamelmages を (Gamelmages) と変えると検索の対象になりません。(Gamelmages) を検索するためには明示的にアクセスパスに追加してください。

アクセスパスの詳細は「[ヘッダポップアップメニュー](#)」(p79) を参照してください。

ユーザパス

[ユーザパス] ラジオボタンをクリックするとアクセスパス設定パネルの[ユーザパス欄](#)にユーザパスを表示します。

システムパス

[システムパス] ラジオボタンをクリックするとアクセスパス設定パネルの[システムパス欄](#)にシステムパスを表示します。

必ずユーザパスを検索する

ユーザヘッダファイルと同じようにシステムヘッダファイルを検索するには、[必ずユーザパスを検索する] チェックボックスをオンにします。オフの場合、IDE は `#include <...>` 疑似命令と `#include "..."` 疑似命令を別のものとして扱います。

ユーザパス欄

[ユーザパス] 欄はプロジェクト特有のアクセスパスを示します。Pascal では、これらのアクセスパスが最初に検索されます。C では、`#include "..."` 疑似命令によりこれらのアクセスパスが検索されます。デフォルトでは、開いているプロジェクトを含むフォルダである {Project} になります。

システムパス欄

[システムパス] 欄はシステムヘッダなどのファイルへのアクセスパスを示します。Pascal では、このアクセスパスは [ユーザパス] 欄の後に検索されます。C では、`#include <...>` 疑似命令によりこれらのアクセスパスが検索されます。デフォルトでは、CodeWarrior IDE を含むフォルダである {Compiler} になります。

デフォルトを追加

CodeWarrior IDE ではデフォルトのアクセスパスを削除した後にも、[ユーザパス欄](#)または[システムパス欄](#)のデフォルトパスを復元できます。アクティブな欄へデフォルトパスを追加するには、[デフォルトを追加] ボタンをクリックします。デフォルトのアクセスパスが表示されているときはこのボタンは使用不可になります。

ホストフラグ

[ホストフラグ] ポップアップメニューでアクセスパスを使うホストプラットフォームを指定します。[ユーザパス欄](#)または[システムパス欄](#)でアクセスパスを選択し、[ホストフラグ] ポップアップメニューからホストプラットフォームを選択します。

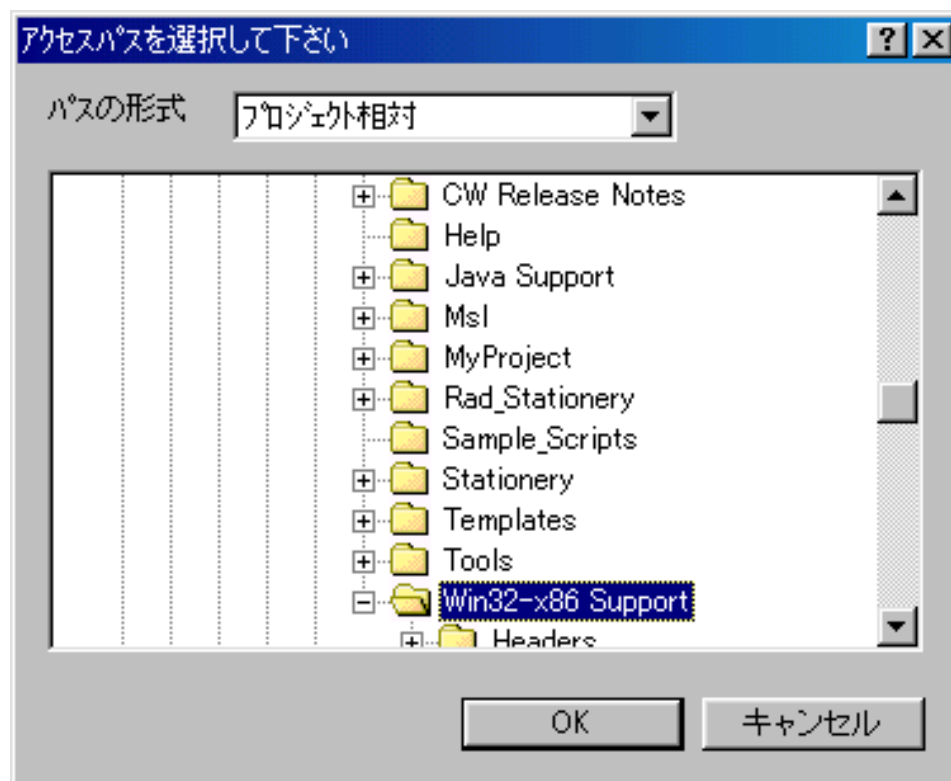
すべてのホストプラットフォームでアクセスパスを使う場合、ポップアップメニューから [すべて] を選択します。ホストにアクセスパスを使わせない場合 [なし] を選択します。

パスを選択して [ホストフラグ] ポップアップメニューから [Mac OS] を選択した場合、IDE は Mac OS コンピュータ上で動作しているときにアクセスパスを検索し、他のプラットフォーム上ではアクセスパスを無視します。

追加

新しいアクセスパスを追加するには、まず[システムパス欄](#)または[ユーザパス欄](#)を選択します。次に [追加] ボタンをクリックします。ダイアログ ([図 9.6](#)) が表示されます。このダイアログでアクセスパスを追加する項目を選択します。またパス欄へフォルダをドラッグ & ドロップすることもできます。

図 9.6 アクセスパス選択ダイアログ



アクセスパスを保存する方法を、以下のオプションで選択します。

絶対パス：起動ディスクのルートレベルから追加するフォルダまでの完全なパスを保存します。プロジェクトを他のシステムへ移動したり、またはハードディスクやパスに含まれるフォルダの名前を変更したときには絶対パスを更新する必要があります。

プロジェクト：プロジェクトを含むフォルダから追加するフォルダまでのパスを保存します。プロジェクトを移動しても、相対パスの階層が同じであれば、相対アクセスパスを更新する必要はありません。しかし、プロジェクトファイルのあるフォルダとは別のハードディスクにあるフォルダへの相対パスを作ることはできません。

CodeWarrior：CodeWarrior IDE を含むフォルダから追加するフォルダまでのパスを保存します。プロジェクトを移動しても相対パスの階層が変わらなければ、コンパイラ相対パスを更新する必要はありません。CodeWarrior のあるフォルダとは別のハードディスクにあるフォルダへの相対パスを作ることはできません。

システム：オペレーティングシステムのフォルダから追加するフォルダまでのパスを保存します。プロジェクトを移動しても相対パスの階層が変わらなければ、システム相対パスを更新する必要はありません。オペレーティングシステムのベースフォルダとは別のハードディスクにあるフォルダへの相対パスを作ることはできません。

ソースツリー：ソースツリーで指定されたフォルダからのパスを保存します。実際のオプション名にはソースツリーの名前が入ります。例えば「Sample」というソースツリーがあれば、[Sample] というオプションが表示されます。グローバルなソースツリーの定義については「[ソースツリー](#)」(p188) を参照してください。プロジェクト特有のソースツリーについては「[ソースツリー](#)」(p242) を参照してください。

注意： 相対パスを使うと、プロジェクトに同名のファイルを複数含めることができます。しかし大きなプロジェクトで相対パスを使うと、パフォーマンスが低下します。

上記のアクセスパスからいずれかを選択し、追加先のフォルダを選択します。[OK] ボタンをクリックすると CodeWarrior IDE はアクセスパスを、[ユーザパス欄](#)か[システムパス欄](#)のいずれかに追加します。変更しない場合は [キャンセル] ボタンをクリックします。

変更

アクセスパスを変更するには、最初に [システムパス] 欄または [ユーザパス] 欄でパスを選択します。次に [変更] ボタンをクリックします。ダイアログ ([図 9.6 \(p234\)](#)) が表示されます。このダイアログを使って古いパスを置換する新しいアクセスパスを選択します。ダイアログのオプションの詳細は、「[追加](#)」(p234) を参照してください。

削除

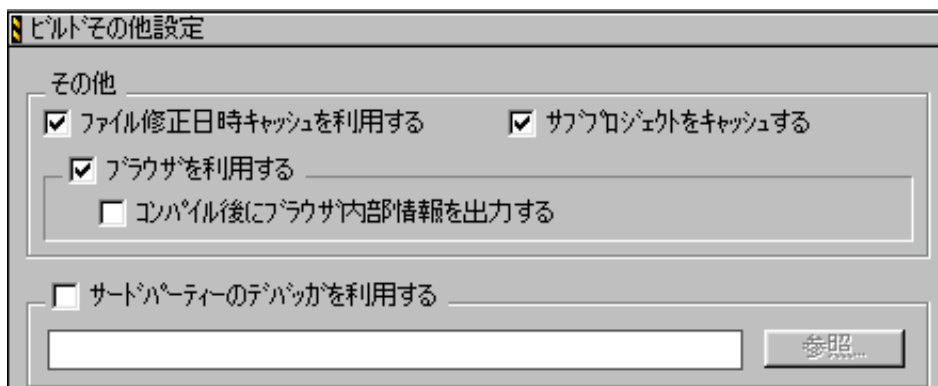
アクセスパスを削除するには、最初に [システムパス] 欄または [ユーザパス] 欄でパスを選択します。次に、[削除] ボタンをクリックします。パスが削除されます。デスクトップのごみ箱へドラッグしても削除できます。

ビルドその他設定

ビルドその他設定パネル ([図 9.7](#)) にはプロジェクトのビルドに関するオプションがあります。

ターゲット設定ダイアログを開いてビルドその他設定パネルを選択する方法は「[ターゲット設定の解説](#)」(p225) を参照してください。

図 9.7 ビルドその他設定パネル



ファイル修正日時キャッシュを利用する

[ファイル修正日時キャッシュを利用する] チェックボックスで、プロジェクトをメイクする前にファイルの修正日時を調べるか否かを指定します。オンの場合、IDE はプロジェクトのファイルの修正日時をキャッシュします。IDE はコンパイル時にこのキャッシュを参照して再コンパイルすべきファイルを決定します。オフの場合、IDE はコンパイルするたびにプロジェクト内の全ファイルを調べます。

CodeWarrior エディタだけを使ってファイルを編集するのであれば、このチェックボックスをオンにすると、コンパイル時間を短縮できます。CodeWarrior 以外のエディタを使っている場合、オフにします。IDE はコンパイルするたびにすべてのファイルを調べます。

サブプロジェクトをキャッシュする

[サブプロジェクトをキャッシュする] チェックボックスがオンの場合、複数プロジェクトの更新とリンクが改善されます。またブラウザでターゲットのサブプロジェクト両方のシンボル情報を生成できます。オフの場合、CodeWarrior IDE が使用するメモリが少なくなります。

ブラウザを利用する

[ブラウザを利用する] チェックボックスがオンの場合、次にプロジェクトをビルドするときに CodeWarrior ブラウザが必要とする情報を生成します。この情報がなければプロジェクトのブラウザウィンドウを開くことはできません。

プロジェクトの再ビルドについては「[プロジェクトをメイク](#)」(p259)、ブラウザの設定とオプションの詳細は「[ソースコードのブラウズ](#)」(p143) を参照してください。

コンパイル後にブラウザ内部情報を出力する

[コンパイル後にブラウザ内部情報を出力する] チェックボックスがオンの場合、プラグインコンパイラまたはリンクが IDE 用に生成したブラウザの生情報を表示します。このオプションは IDE 用のプラグインを作成するときに便利です。

注意： [コンパイル後にブラウザ内部情報を出力する] チェックボックスがオンの場合、1つのファイルまたは小さいファイルしかコンパイルできません。IDE の表示する情報が多すぎてプロジェクト全体のコンパイルはできません。

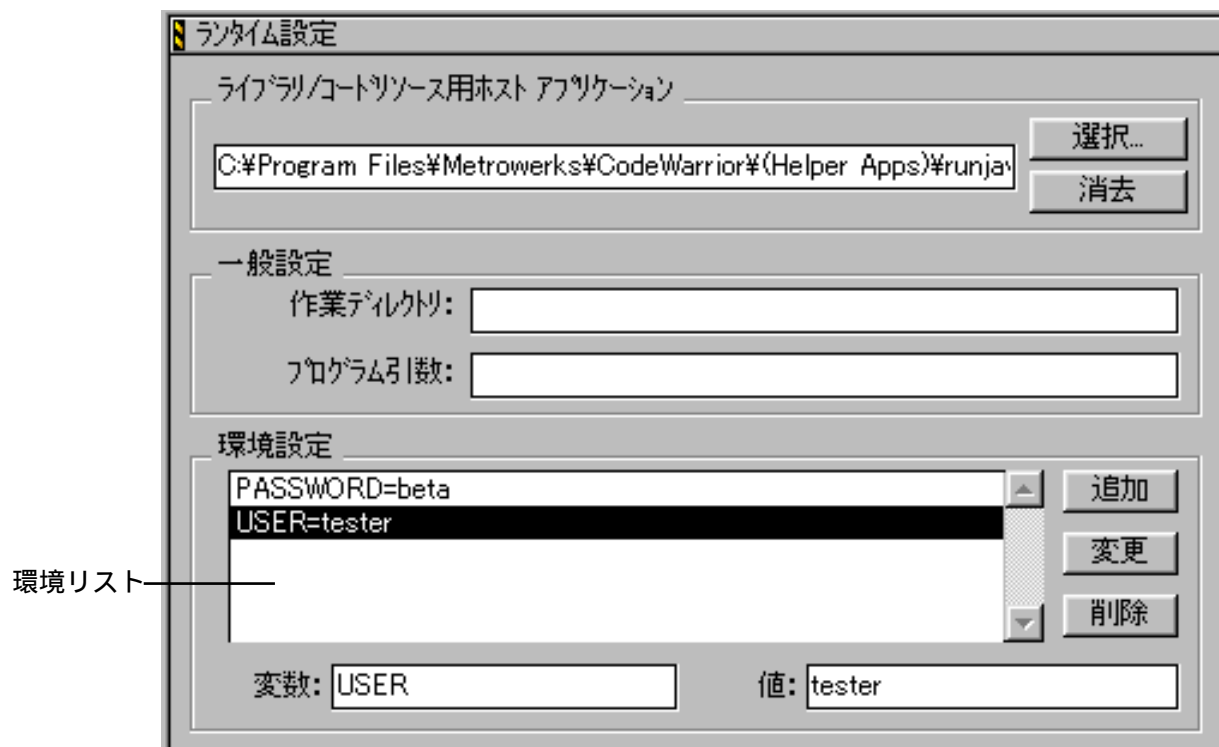
サードパーティのデバッガを利用する

[サードパーティのデバッガを利用する] チェックボックスがオンの場合、CodeWarrior のデバッガではなくサードパーティのデバッガを使うことができます。フィールドにデバッガアプリケーションへのパスを入力するか、[参照] ボタンをクリックしてダイアログで位置を指定してください。

ランタイム設定

ランタイム設定パネル([図 9.8](#))で単体実行不可ファイルをデバッグするアプリケーション、出力ディレクトリ、プログラム引数、環境変数を設定します。

図 9.8 ランタイム設定パネル



ライブラリ / コードリソース用ホストアプリケーション

[ライブラリ / コードリソース用ホストアプリケーション] フィールドで単体実行不可ファイル (共有ライブラリ、ダイナミックリンクライブラリ (DLL)、コードリソースなど) をデバッグするためのアプリケーションを指定します。ここで指定するのはデバッガアプリケーションではなく、単体実行不可ファイルとインタラクトするアプリケーションです。

例えば Photoshop プラグインを作成している場合、Photoshop をホストアプリケーションにします。[デバッグ] コマンドを実行すると IDE はプラグインをビルドし、そのシンボルファイルをロードします。次に Photoshop を起動してプラグインのデバッグの準備を整えます。

ホストアプリケーションは、そこへのパスをフィールドに入力することで指定できます。[選択] ボタンをクリックしてダイアログを開き、そこでアプリケーションを選択することもできます。[消去] ボタンはフィールドからパスを削除します。

一般設定

[一般設定] 欄には以下のフィールドがあります。

作業ディレクトリ：このフィールドでカレントプロジェクトがアクセスするデフォルトのディレクトリを指定します。ここでデバッグを行います。ディレクトリの指定がなければ、実行可能ファイルのあるディレクトリでデバッグを行います。

プログラム引数：このフィールドでデバッグ開始時にプロジェクトへ渡すコマンドライン引数を指定します。プロジェクトメニューの [実行] コマンドを選択したとき、プロジェクトはこの引数を受け取ります。

環境設定

[環境設定] 欄で環境変数を指定します。環境変数は、プログラムの `main()` 関数の環境パラメータの一部として、または環境呼び出しの一部としてプログラムに渡されます。この変数はターゲットプログラムでのみ利用可能です。プログラムが終了すると、利用できません。例えばサーバへログするプログラムを作っている場合、`PASSWORD` と `USER` に対して変数を利用できます ([図 9.8](#))。

新しい環境変数を作成する手順です。

1. [変数] フィールドに環境変数の名前を入力する
2. [値] フィールドに環境変数の値を入力する
3. [追加] ボタンをクリックする

新しい環境変数が環境リストに現れます ([図 9.8](#))。

既存の環境変数を変更する手順です。

1. 変更する環境変数を環境リストから選択する
2. [変数] と [値] フィールドを変更する
3. [変更] ボタンをクリックする

変更した環境変数が環境リストに現れます ([図 9.8](#))。

既存の環境変数を削除する手順です。

1. 削除する環境変数を環境リストから選択する

2. [削除] ボタンをクリックする

選択した環境変数が環境リストから消えます ([図 9.8](#))。

ファイルマッピング

ファイルマッピング設定パネル ([図 9.9](#)) は、.c または .p などのファイル拡張子を、プラグインコンパイラに関連付けるために使います。これにより、CodeWarrior が使うコンパイラをファイル名から選択できるようになります。

注意： ファイルマッピングにより IDE はファイルを認識できるようになります。プロジェクトにファイルを追加できない、またはプロジェクトウィンドウにファイルをドラッグ&ドロップできない場合など、ファイルマッピング設定パネルを調べてください。ファイルマッピングは現在のリンクと関係しているため、使用するリンクが変わるとファイルマッピングも変わります。「[リンク](#)」(p230) を参照してください。

図 9.9 ファイルマッピング設定パネル



ターゲット設定ダイアログとファイルマッピング設定パネルの開き方は「[ターゲット設定の解説](#)」(p225) を参照してください。

ファイルマッピングリスト

ファイルマッピングリストには、ファイルタイプ、拡張子、リスト内の各ファイル拡張子のコンパイラの情報が含まれています。このリストにより、CodeWarrior IDE はファイル名に対応したコンパイラを起動することができます。

このリストに拡張子を追加するには、リスト内の既存の項目を選択して、[マッピング情報] 欄の情報を編集して [追加] ボタンをクリックします。

ドキュメントファイルマッピングをプロジェクトに追加するには、以下の手順で行います。

1. ファイルマッピングリストで既存の項目を選択する
2. [[ファイルタイプ](#)] フィールドのテキストを削除する
3. [[拡張子](#)] フィールドのテキストを .doc に変える
4. [[コンパイラ](#)] ポップアップメニューで [なし] を選択する
5. [追加] ボタンをクリックする

ファイルタイプ

[ファイルタイプ] フィールドに [ファイルマッピングリスト](#) で選択したファイルの種類を入力します。

拡張子

[拡張子] フィールドに、[ファイルマッピングリスト](#) で選択したファイルの種類に対応するファイル名拡張子 (.c、.h など) を入力します。 [表 9.2](#) に CodeWarrior IDE のデフォルトのファイル名拡張子をまとめます。

表 9.2 デフォルトのファイル名拡張子

タイプ	拡張子	説明
CodeWarrior の最小インストール	.iSYM	CodeWarrior Intel シンボル
	.mch	CodeWarrior プリコンパイルヘッダ
	.mcp	CodeWarrior プロジェクトファイル
	.SYM	CodeWarrior Mac OS 68K デバッグシンボル
	.xSYM	CodeWarrior Mac OS PPC デバッグシンボル
CodeWarrior のデフォルトインストール	.dbg	CodeWarrior Debug デバッグプリファレンス
	.exp	エクスポートシンボルファイル
	.iMAP	CodeWarrior リンクマップ
	.MAP	CodeWarrior リンクマップ
	.xMAP	CodeWarrior リンクマップ

タイプ	拡張子	説明
ライブラリ	.lib	ライブラリファイル
	.o	オブジェクトファイル
	.obj	オブジェクトファイル
	.pch	プリコンパイルヘッダソースファイル
	.pch++	プリコンパイルヘッダソースファイル
デフォルトの C/C++	.c	C ソースファイル
	.cp	C++ ソースファイル
	.cpp	C++ ソースファイル
	.h	C/C++ ヘッダファイル
デフォルトの Java	.class	Java クラスファイル
	.jar	Java アーカイブファイル
	.jav	Java ソースファイル
	.java	Java ソースファイル
デフォルトの Pascal	.p	Pascal ソースファイル
	.pas	Pascal ソースファイル
アセンブリ言語	.a	アセンブリソースファイル
	.asm	アセンブリソースファイル
	.dump	CodeWarrior 逆アセンブラファイル
C/C++	.c++	C++ ソースファイル
	.cc	C++ ソースファイル
	.hh	C++ ヘッダファイル
	.hpp	C++ ヘッダファイル
	.i	C インラインソースファイル
	.icc	C++ インラインソースファイル
	.m	Object C ソースファイル
	.mm	Object C++ ソースファイル
Java	.JMAP	Java Import Mapping Dump
	.jpob	Java Constructor ファイル
	.mf	Java Manifest ファイル
Pascal	.ppu	Pascal プリコンパイルユニット

コンパイラ

[コンパイラ] ポップアップメニューは、[ファイルマッピングリスト](#)で選択したファイルの種類に対応するコンパイラを指定します。

フラグ

[フラグ] ポップアップメニューで以下の 4 種類のフラグをオン、オフを設定します。

リソース : このフラグをオンにすると、選択したファイルマッピングを持つファイルからリソースを完成製品に組み込みます。

起動可能 : このフラグをオンにすると、プロジェクトウィンドウでソースコードファイルをクリックしたときに、そのファイルを作成したアプリケーションを使ってファイルを開きます。

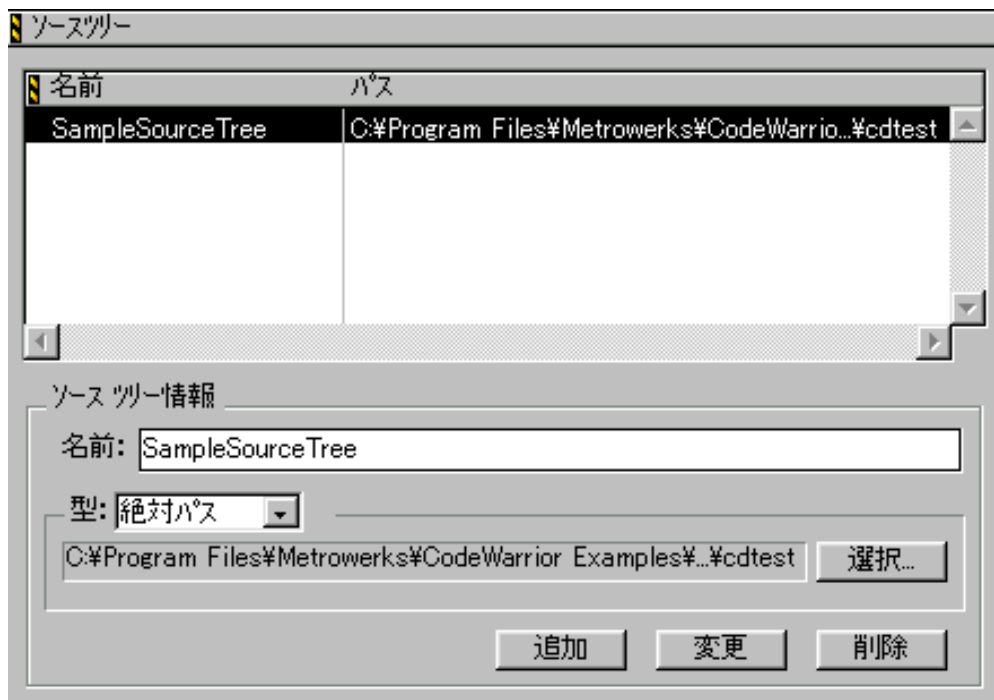
プリコンパイル : このフラグをオンにすると、他のファイルをコンパイルする前にこれらのファイルをコンパイルします。これらのファイルで作成されたドキュメントを他のソースファイルまたはコンパイラが使う場合に効果的です。例えばこのオプションにより、ファイルを C ソースコードファイルに変換するコンパイラを作成しておいて、その後 C ファイルをコンパイルすることができます。YACC (Yet Another Compiler Compiler) は、C コンパイラがコンパイルする C ソースコードを生成するので、YACC ファイルはプリコンパイルファイルとみなされます。

メイク不可 : このフラグをオンにすると、IDE はプロジェクトのコンパイルおよびリンク時に、これらのファイルを無視します。これはプロジェクトに含めたいコメントやドキュメントを含むファイルの場合に便利です。

ソースツリー

ソースツリー IDE 設定パネル ([図 9.10](#)) でプロジェクト特有のソースツリー (ルートパス) を定義します。プロジェクトのアクセスパスをビルドターゲットの出力を設定することができます。これによりさまざまなホストでプロジェクトを共有できます。ソースツリーのパスを少し変更するだけでプロジェクトの機能を保持できます。

図 9.10 ソースツリー設定パネル



IDE のプロジェクトすべてに影響するグローバルなソースツリー IDE 設定パネルもあります。詳細は「[ソースツリー](#)」(p188) を参照してください。ターゲット設定ダイアログで指定したソースツリーはアクティブなプロジェクトの現在のビルドターゲットに適用されます。IDE の IDE 設定ダイアログで指定したソースツリーはすべてのプロジェクトへ適用されます。両方で同じソースツリーを指定すると、ターゲット固有のソースツリーが優先されます。

ソースツリーリスト

ソースツリーリストはプロジェクトのソースツリーをすべて表示します。2 つの列があります。

名前：この列は各ソースツリーの名前を表示します。ソースツリーとしてアクセスパスを定義すると、アクセスパスの定義にこの名前を使用できます。詳細は「[追加](#)」(p234) を参照してください。

パス：この列は各ソースツリーへのパスを表示します。プロジェクトを他のホストへ移植するときはソースツリーのパスを修正しなくてはなりません。パスの修正方法は「[変更](#)」(p246) を参照してください。

名前

[ソースツリー情報] 欄の [名前] フィールド (図 9.10) で新しいソースツリーの名前を入力するか、またはソースツリーリストの行を選択して名前を変更します。

注意： このフィールドが空欄の場合、[追加] ボタンをクリックしてもソースツリーリストには何も表示されません。

型

[ソースツリー情報] 欄の [型] ポップアップメニュー ([図 9.10](#)) でソースツリーの型を選択します。

絶対パス：ファイルパスに基づくソースツリー

環境変数：環境変数の定義に基づくソースツリー

レジストリキー：レジストリにあるキーエントリに基づくソースツリー

追加

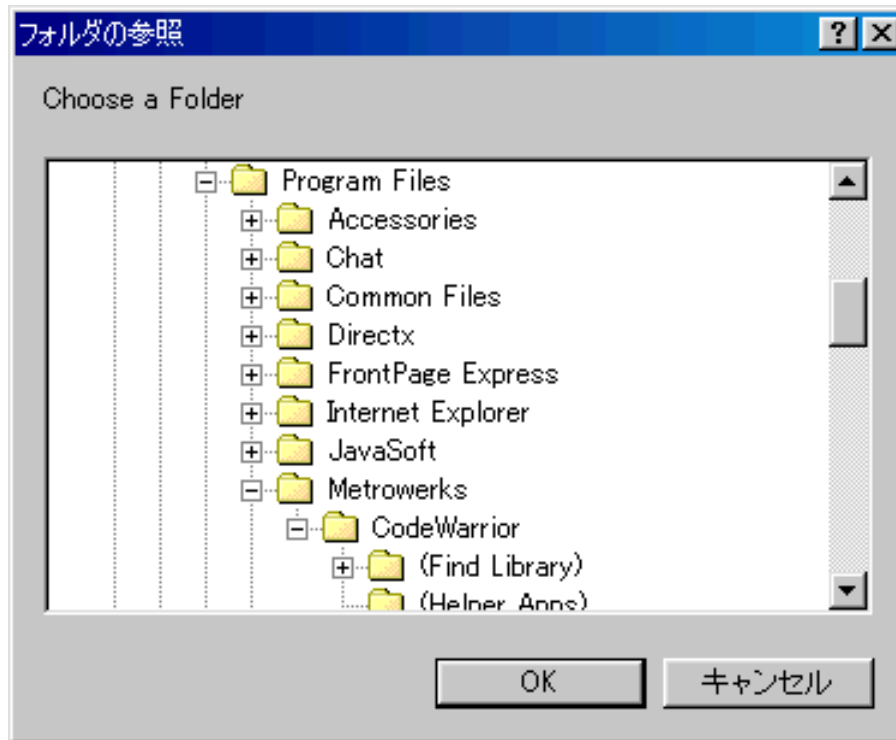
新しいソースツリーを追加するには、[型] ポップアップメニューからソースツリーの種類を選択します。次に [名前] フィールドに新しいソースツリーの名前を入力します。[追加] ボタンをクリックしてソースツリーの追加を終えたら、[保存] ボタンをクリックして変更を保存します。

「絶対パス」のソースツリーを作るときは [選択] ボタンが使用可能になります。このボタンをクリックすると標準ダイアログでパスを選択することができます ([図 9.11](#))。

Windows をホストとする IDE では、パスの保存方法を指定することができます。

絶対パス：起動ディスクのルートレベルから追加するフォルダまでの完全なパスを保存します。プロジェクトを他のシステムへ移動したり、またはハードディスクやパスに含まれるフォルダの名前を変更したときには絶対パスを更新する必要があります。

図 9.11 フォルダの参照ダイアログ



コンパイラ：CodeWarrior IDE を含むフォルダから追加するフォルダまでのパスを保存します。プロジェクトを移動しても相対パスの階層が変わらなければ、コンパイラ相対パスを更新する必要はありません。CodeWarrior のあるフォルダとは別のハードディスクにあるフォルダへの相対パスを作ることはできません。

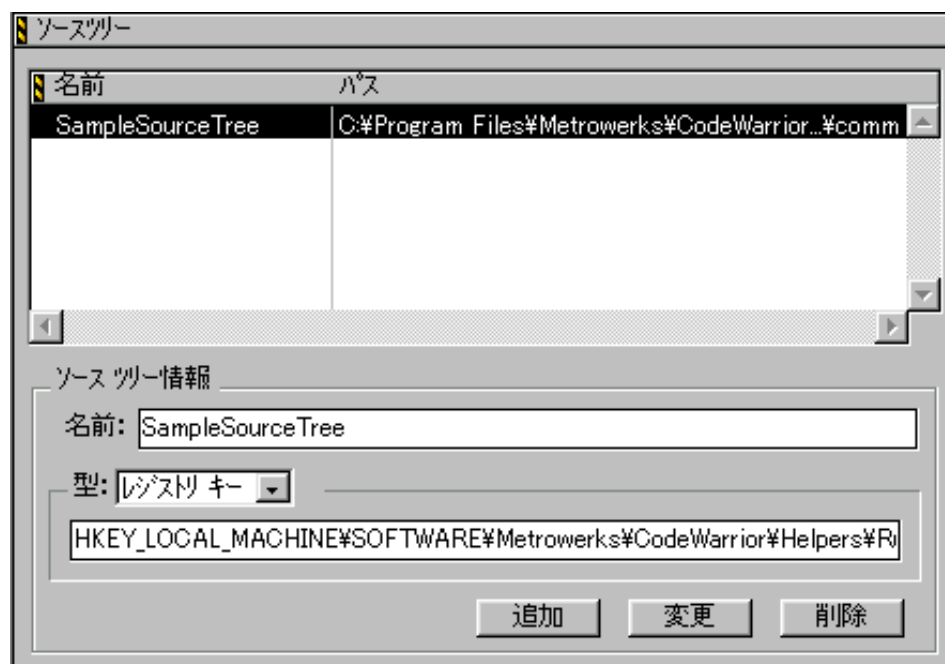
システム：オペレーティングシステムのフォルダから追加するフォルダまでのパスを保存します。プロジェクトを移動しても相対パスの階層が変わらなければ、システム相対パスを更新する必要はありません。オペレーティングシステムのベースフォルダとは別のハードディスクにあるフォルダへの相対パスを作ることはできません。

注意： 相対パスを使うと、プロジェクトに同名のファイルを複数含めることができます。しかし大きなプロジェクトで相対パスを使うと、パフォーマンスが低下します。

上記のアクセスパスからいずれかを選択し、追加先のフォルダを選択します。[OK] ボタンをクリックすると CodeWarrior IDE はアクセスパスを、[ユーザパス欄](#)か[システムパス欄](#)のいずれかに追加します。変更しない場合は [キャンセル] ボタンをクリックします。

[環境変数] または [レジストリキー] ソースツリーを作成するとき、[型] フィールドが [図 9.12](#) のようになります。このフィールドへ環境変数、またはレジストリキーへのパスを入力します。

図 9.12 レジストリキーの作成



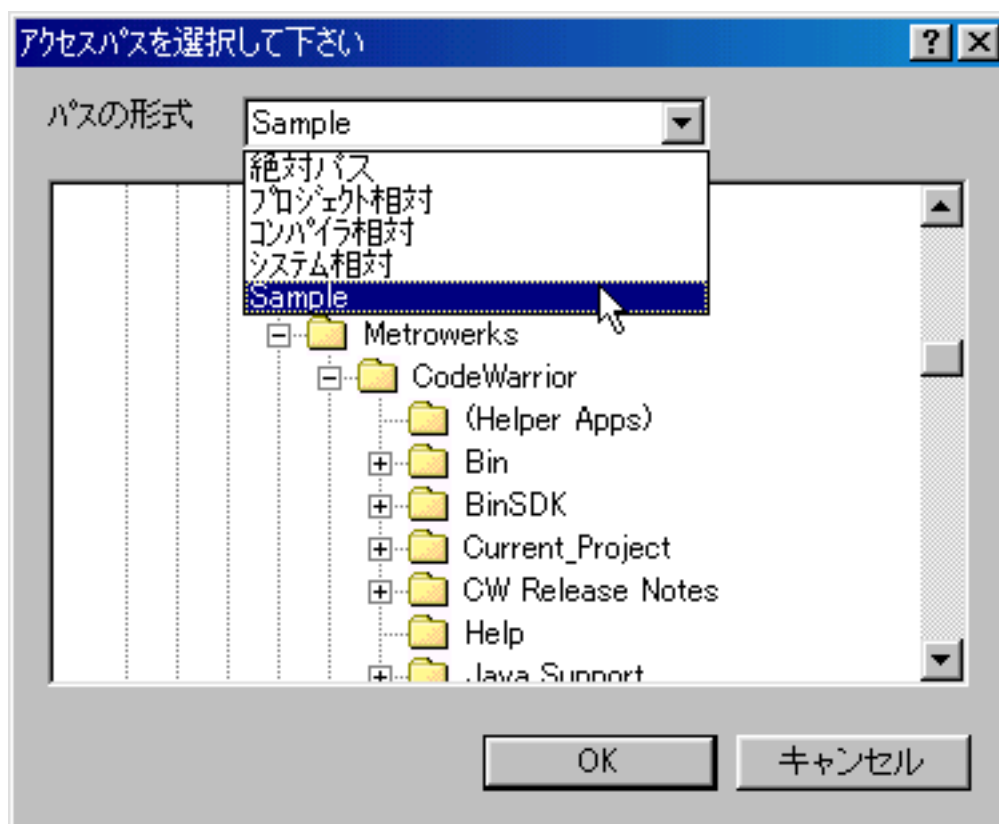
ソースツリーがプロジェクトのパスポップアップメニューに表示されます。例えば「Sample」という名前のソースツリーを作った場合、これを利用してアクセスパスを作成できます（[図 9.13](#)）。また[ターゲット設定](#)パネルの[出力ディレクトリ]フィールドを使ってプロジェクトのターゲット出力を定義することもできます。詳細は「[ターゲット設定](#)」(p229)を参照してください。

変更

ソースツリーを変更するには、[ソースツリーリスト](#)で選択します。次に[名前]と[型]フィールドを変更し、[変更]ボタンをクリックします。変更を終えたら IDE 設定ダイアログの[保存]ボタンをクリックして保存します。

注意： ソースツリーを変更した後、元のソースツリーを参照しないようにプロジェクトを修正する必要があります。IDE はプロジェクトの更新を要求するメッセージを表示します。

図 9.13 アクセスパスポップアップメニューでソースツリーを選択



プロジェクトのファイル、またはライブラリがソースツリーに含まれていなければ、IDE はそれらのファイルをコンパイル、リンク、実行するときに見つけることができません。このため、既存のソースツリーを変更したとき、IDE はプロジェクトの更新を要求するメッセージを表示します。

削除

ソースツリーを削除するには、ツリーを設定パネルで選択します。[削除] ボタンをクリックして削除します。ソースツリーの削除を終えたら、ターゲット設定ダイアログの[保存] ボタンをクリックして変更を保存します。

注意： ソースツリーを削除した後、削除したソースツリーを参照しないようにプロジェクトを修正する必要があります。IDE はプロジェクトの更新を要求するメッセージを表示します。

コード生成の設定

以下の設定パネルで IDE がコードを生成する方法をカスタマイズします。

[グローバル最適化](#)

グローバル最適化

グローバル最適化設定パネル (図 9.14) で、より速い実行コードを生成するためのコンパイラの命令の並べ替えなどを指定します。プログラムから冗長な操作を取り除いたり、プログラムの分析をする最適化があります。プログラムパフォーマンスを改善するためのパネルです。

ターゲット設定ダイアログを開いてグローバル最適化設定パネルを選択する方法は「[ターゲット設定の解説](#)」(p225) を参照してください。

最適化は、オブジェクトコードの実行時の論理的シーケンスを乱すことなく、オブジェクトコードの並べ替えを行います。最適化されていないプログラムと最適化されたプログラムは同じ結果を出します。

注意： コンパイラの最適化は、ソフトウェアのデバッグを終えてから行ってください。最適化したプログラムでデバッガを使うと、デバッガのソースコードの表示に影響を与えます。

これらの設定をビルドターゲットに適用する方法については「[ターゲットマニュアル](#)」(p20) を参照してください。

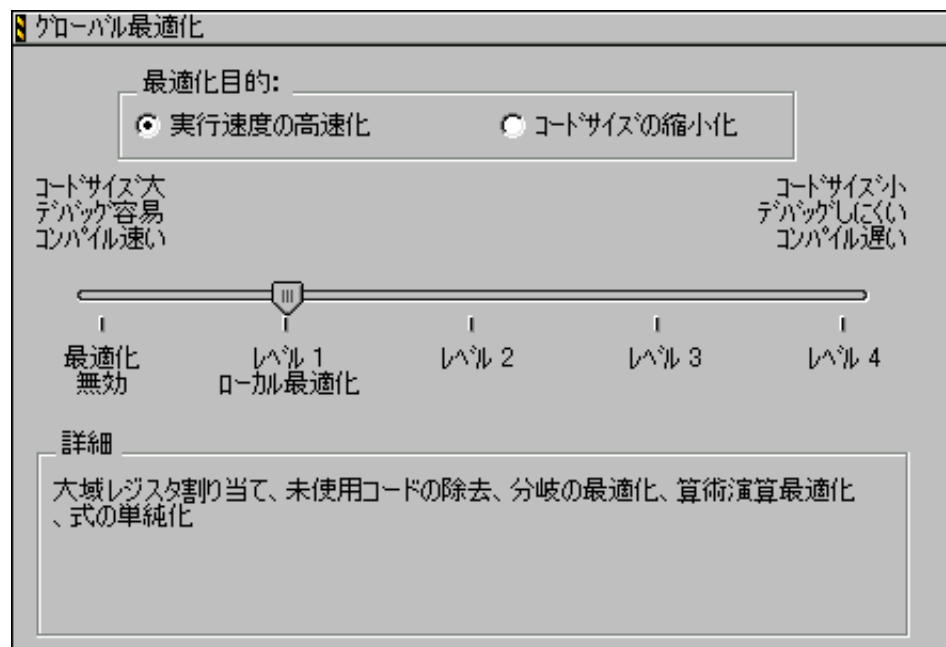
グローバル最適化設定パネル (図 9.14) には以下のオプションがあります。

[最適化目的](#) (Optimize For)

[最適化レベルスライダー](#) (Optimization Level Slider)

[詳細](#) (Details)

図 9.14 グローバル最適化設定パネル



最適化目的

[最適化目的] 欄でコードを最適化する基準を決めます。

実行速度の高速化 (Faster Execution Speed): オブジェクトコードの実行速度を上げますが、サイズが大きくなります。

コードサイズの縮小化 (Smaller Code Size): オブジェクトコードのサイズを小さくしますが、速度は下がります。

最適化レベルスライダー

以下のレベルを考慮してコードを最適化してください。最適化をしない、またはいずれかのレベルの最適化を行うことができます。レベルが高いほど高度な最適化がなされます。

詳細

[最適化レベルスライダー](#) の下の [詳細] 欄はプロジェクトに適用する最適化を詳しく表示します。ターゲットプロセッサによって、使用できる最適化は異なります。

大域レジスタ割り当て (Global Register Allocation): 頻繁に使う変数の値をメモリではなくレジスタに保存します。[最適化レベルスライダー](#) が [最適化無効] のとき [テンポラリ値にのみ大域レジスタを割り当てる] とも表示されます。

未使用コードの除去 (Dead Code Elimination): 論理的に一度も実行も参照もされないステートメントを削除します。このオプションがオンの場合、コードは小さくなります。

分岐の最適化 (Branch Optimizations): 中間言語のコードをマージして再構成します。分岐命令を削除して最適化します。このオプションがオンの場合、オブジェクトコードは小さくなり、実行速度が上がります。

算術演算最適化 (Arithmetic Optimizations): 複雑な演算インストラクションを、より高速で同じ結果を出す、等価なインストラクションで置換します。例えば乗算のインストラクションを加算に置き換えます。また、乗算インストラクションをライブラリ呼び出しで置換することもあります。このオプションがオンの場合、オブジェクトコードは速くなります。

式の単純化 (Expression Simplification): 演算式を等価または単純な式で置換します。例えば $1 * a$, $a + b$ を $a - (-b)$ で置換します。 $x \& (x \& y)$ を $x \& y$ に、そして $x \& (x | y)$ を x で置換します。

共通部分式の削除 (Common Subexpression Elimination): 類似する余分な評価式を統一します。例えば 2 つの連続するステートメント両方で $a * b * c + 10$ という評価式が使われている場合、コンパイラはその評価式を一度だけ演算するオブジェクトコードを生成して、両方のステートメントで結果と評価式を置換します。このオプションがオンの場合、オブジェクトコードはより小さく、速くなります。

コピーによる伝播 (Copy Propagation): 1 つの変数が複数箇所で出現しているとき、それを一箇所にまとめます。オンの場合、オブジェクトコードは小さく、高速になります。

ピープホール最適化 (Peephole Optimization): コードの小さなセクションに局所的な最適化を行います。最適化されたセクションは高速になります。

デッドストア削除 (Dead Store Elimination): 変数が再度代入されるまでに使用されていない場合、代入を削除します。このオプションがオンの場合、オブジェクトコードは小さく、速くなります。

範囲分割最適化 (Live Range Splitting): 全く別の用途で、2ヶ所で使われている局所変数を最適化します。例えば[例 9.1](#) の変数 `i` で 2 つの分離が使われています。変数 `i` の 2 度目の使用箇所は別のコンパイラテンポラリ変数で置換できます。[範囲分割最適化] は変数の寿命を縮めて最適な割り当てを行います。Long live ranges はレジスタを不必要に分割してしまいます。

例 9.1 範囲分割最適化

```
int a[100], b[100], c[100], d[100], e[100], f[100];
foo(int n)
{
    int i;

    for(i=0; i<n; i++)
    {
        a[i] = b[i] + c[i];
    }

    for(i=0; i<n; i++)
    {
        d[i] = e[i] + f[i]
    }
}
```

ループ不変の移動 (Loop-Invariant Code Motion): ループ内で変化しない演算をループの外へ移動し、ループの速度を上げます。このオプションがオンの場合、オブジェクトコードは速くなります。

強度の削減 (Strength Reduction): ループ内にある複数のインストラクションを置換してループの速度を上げます。このオプションがオンの場合、オブジェクトコードは大きくなりますが、速度は速くなります。

ループの変換 (Loop Transformations): ループ設定やループのコンパイルテストによるコードのオーバーヘッドを減らすために、ループを翻訳したコードを並べ替えます。オブジェクトコードの実行速度は上がります。

ループアンローリング (Loop Unrolling): ループ内のコードを複数回コピーする最適化です。ループコンパイルのテストや、ループ開始点への分岐戻りによるオーバーヘッドを生じる操作をコピーします。オブジェクトコードの実行速度は上がりますが、サイズは大きくなります。

ベクトル演算化 (Vectorization): ベクタ最適化をサポートするプロセッサに対して、コードループを使う配列計算を適切なベクトルインストラクションへ翻訳します。オブジェクトコードの実行速度は上がります。翻訳できないコードループもあります。

生存分析レジスタ割り当て (Lifetime Based Register Allocation) : 同じ関数にある異なる変数が同一のステートメントで使われていない場合、変数に対して同じプロセッサレジスタを使います。このオプションがオンの場合、オブジェクトコードの実行速度は速くなります。

命令スケジューリング (Instruction Scheduling) : プログラム命令を並べ替え、レジスタの使用とプロセッサリソースの衝突を減少します。オブジェクトコードの実行速度が上がります。

次の繰り返し (Repeated) : [[詳細](#)] 欄で () の中にある最適化を繰り返します。

エディタの設定

CodeWarrior エディタの設定パネルについて説明します。

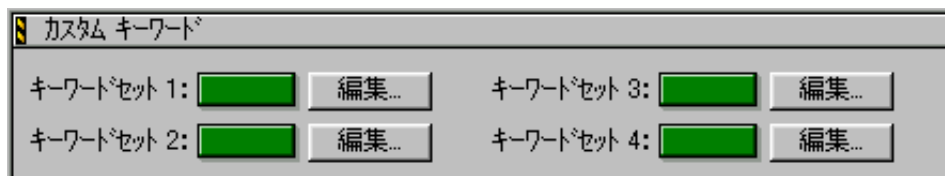
[カスタムキーワード](#)

カスタムキーワード

カスタムキーワード設定パネル ([図 9.15](#)) で、エディタでファイルが表示されたときに特定の色で表示される独自のキーワードセットを定義できます。キーワードはプロジェクト固有であり、CodeWarriorグローバルではありません。

ターゲット設定ダイアログを開いてカスタムキーワード設定パネルを選択する方法は「[ターゲット設定の解説](#)」(p225) を参照してください。

図 9.15 カスタムキーワード設定パネル



キーワードセットの定義については「[カラーシンタックス](#)」(p198) を参照してください。色の設定、キーワードの指定、インポート、エクスポートについても説明しています。

キーワードの色を変更するには、カラーサンプルをクリックします。キーワードセットを変更するには [編集] ボタンをクリックして、表示されたダイアログに必要な事項を入力します。

デバッガの設定

以下は CodeWarrior デバッガ用の設定パネルです。

[デバッガ設定](#)

デバッガ設定

デバッガ設定パネル ([図 9.16](#)) にログ状況やデータ更新のタイミングなどを設定するオプションがあります。

ターゲット設定ダイアログを開いてデバグ設定パネルを選択する方法は「[ターゲット設定の解説](#)」(p225) を参照してください。

リロケートライブラリ、コードリソースまたはリソースモードデバグフォルダの場所
[リロケートライブラリ、コードリソースまたはリモートデバグフォルダの場所] フィールドで再配置されたライブラリ、コードリソース、現在のプロジェクトに必要な単体実行不可ファイルの位置を指定します。またプロジェクトが使うリモートデバグフォルダも指定できます。フィールドにはパスを入力します。[選択] ボタンをクリックしてダイアログを開き、そこでファイルを選択することもできます。

図 9.16 デバグ設定パネル



アプリケーション起動時に一時的ブレークポイントで停止する

[アプリケーション起動時に一時的ブレークポイントで停止する] チェックボックスがオンの場合、デバグ開始時にデバグが指定された一時的ブレークポイントを見つけると、プログラムの実行を停止します。[初期値] ラジオボタンをオンにすると、常に `main()` 関数で停止します。[ユーザー指定] ラジオボタンをオンにすると、右側のフィールドに停止する関数を指定することができます。

自動的にライブラリをターゲットにする

[自動的にライブラリをターゲットにする] チェックボックスは、デバグでプロジェクトファイル以外のファイルをデバグするとき利用します。例えば実行中のプロセスのアタッチなどです。プロセスのアタッチについては「[プロセスウィンドウのツールバー](#)」(p295) を参照してください。

[自動的にライブラリをターゲットにする] チェックボックスがオンの場合、IDE はターゲットアプリケーションにロードされたダイナミックリンクライブラリ (DLL) をデバグしようとしています。IDE は自動的にロードされた DLL (シンボルファイルがあるもの) をデバグします。

注意： [自動的にライブラリをターゲットにする] チェックボックスがオンの場合、パフォーマンスが低下する可能性があります。これを避けるには、チェックボックスをオフにしてください。

システムメッセージを記録する

[システムメッセージを記録する] チェックボックスがオンの場合、すべてのシステムメッセージをファイルにログします。オフの場合、はログファイルを生成しません。ログファイルについては「[Log ウィンドウ](#)」(p306) を参照してください。

注意： Windows ではログ情報は DLL のロードやアンロード、`debug printf()` に関するメッセージも含みます。

実行毎にシンボルをキャッシュする

[実行毎にシンボルをキャッシュする] チェックボックスがオンの場合、デバッガはプロジェクトのシンボル情報をキャッシュして、次のデバッグ時にそのキャッシュ情報を参照します。オフの場合、デバッグが終了するとデバッガはプロジェクトのシンボル情報を破棄します。連続してデバッグを行うときに便利な機能です。

Watchpoint で停止する

[Watchpoint で停止する] チェックボックスがオンの場合、デバッガが Watchpoint を見つけたときに (Watchpoint の値が変わっていても) プログラムの実行が一時停止します。オフの場合、Watchpoint の値が変化したときだけ実行が一時停止します。

データ更新頻度

[データ更新頻度] チェックボックスがオンの場合、指定した間隔でデバッグウィンドウの情報を更新します。間隔を [データ更新頻度] フィールドに入力します。[データ更新頻度] 欄には次の更新までの時間を秒単位で入力します。オフの場合、デバッガ情報を更新しません。デバッグの間中デバッガウィンドウの情報は変わりません。

第 10 章 コンパイルとリンク

この章ではプロジェクトをコンパイル、リンクして最終の実行可能ファイルを生成する方法、またメッセージウィンドウを利用してコンパイル、リンクのエラーを修正する方法について説明します。読者が既にプロジェクトを作成し、必要なファイルを追加、グループ化し、プロジェクトのオプションを設定していることを前提としています。これらの作業方法は「[プロジェクト](#)」(p31)、「[ファイル](#)」(p77)、「[ソースコードの編集](#)」(p95)、「[IDE のオプション設定](#)」(p181)を参照してください。

プロジェクトウィンドウの列やポップアップメニューについては「[プロジェクト](#)」(p31)を参照してください。

CodeWarrior で作成できるプログラムの詳細については説明していません。そのような情報は、該当するプラットフォームの『Targeting』マニュアルを参照してください。各マニュアルの内容は、「[プラットフォームターゲットのマニュアル](#)」(p21)を参照してください。

CodeWarrior IDE では、開いているプロジェクトに含まれているファイルだけをコンパイルおよびリンクできます。つまりファイルをコンパイルする前に、プロジェクトを開いておく必要があります。

この章では、以下の項目について説明します。

[コンパイラを選択](#)

[プロジェクトのコンパイル、リンク](#)

[プリコンパイルまたはプリプロセスヘッダを使う](#)

[ソースコードをプリプロセス](#)

[ソースコードを逆アセンブル](#)

[メッセージウィンドウの解説](#)

[メッセージウィンドウを使う](#)

コンパイラを選択

ソースファイルを作成するときには、特定のプログラム言語（C、C++、Pascal、Java などの言語）を使います。これらの言語はそれぞれ、ファイルの名前に規約があります。例えば C 言語では、ソースファイルは .c、ヘッダファイルは .h の拡張子で終わります。

ここでは、CodeWarrior IDE で各言語のコンパイラにファイル拡張子を関連付ける方法を説明します。

プラグインコンパイラの仕組み

CodeWarrior IDE は、多くのプログラム言語でコンパイルができます。多くの異なる言語を受け入れられるようにするため、プラグインコンパイラが使われています。

プラグインはそれぞれ独自のコンパイル作業を行わせるための、基本的に小さなロード可能なコードモジュールです。例えば、C、C++、Pascal、Java、およびアセンブリ言語のプラグインコンパイラがあります。

CodeWarrior IDE でプラグインコンパイラが扱うプロジェクトファイルを判断できるように、通常プラグインコンパイラにはデフォルトのビルドターゲット設定があります。IDE は通常のコンパイルおよびリンク操作中に、ファイルを正しいプラグインに自動的に割り当てます。

ファイル拡張子を設定

プラグインコンパイラにファイルを関連付けるには、[[ファイルマッピング](#)] オプションを使います。このオプションの設定方法は「[ファイルマッピング](#)」(p239) を参照してください。

プロジェクトのコンパイル、リンク

CodeWarrior IDE は、プロジェクトをビルドする多くの方法を提供しています。プロジェクトをビルドするということは、プロジェクトをオブジェクトコードへコンパイルして、オブジェクトコードを最終の実行可能ファイルへリンクするということです。

コンパイルおよびリンク用のコマンドは、すべて[プロジェクトメニュー](#)から利用できます。プロジェクトの種類によって、これらのコマンドの一部が選択不可になったり、名前が変わったりします。例えば共有ライブラリを実行することはできませんが、メイクすることはできます。また、他のコマンドの実行中やプロジェクトがデバッグ中の場合は、コンパイルまたはリンクのコマンドは薄く表示され使用できません。

CodeWarrior IDE は、開いているプロジェクトに含まれるファイルだけをコンパイル、リンクします。つまりファイルをコンパイルする前に、プロジェクトを開いておく必要があるということです。

同時に複数のプロジェクトを開いている場合、コンパイル、リンクを実行する前に、デフォルトプロジェクトの選択方法を知っておくと便利です。詳細は「[デフォルトのプロジェクトを選択](#)」(p53) を参照してください。

ここでは、以下の項目を説明します。

[ファイルをコンパイル](#)

[リンク順を設定](#)

[プロジェクトを更新](#)

[プロジェクトをメイク](#)

[デバッグを有効にする](#)

[プロジェクトを実行](#)

[プロジェクトをデバッグ](#)[リンクマップを生成](#)[修正日時を同期](#)[オブジェクトを削除](#)[高度なコンパイルオプション](#)

ファイルをコンパイル

プロジェクト内の1つのファイルまたは複数のファイルをコンパイルすることもできます。もちろん、プロジェクトのすべてのファイルをコンパイルすることもできます。

ファイルをコンパイルするとき、プロジェクト内の複数ターゲット間でターゲットを切り替えることができます。詳細は「[現在のビルドターゲットを設定](#)」(p68)を参照してください。

[プロジェクトメニュー](#)の[[コンパイル](#)] コマンドは、以下のとき薄く表示され、選択不可になります。

開いているプロジェクトがない

アクティブなエディタウィンドウのファイルに、ソースコードファイル名拡張子が付いていない

アクティブなウィンドウのソースファイルがプロジェクトに含まれていない

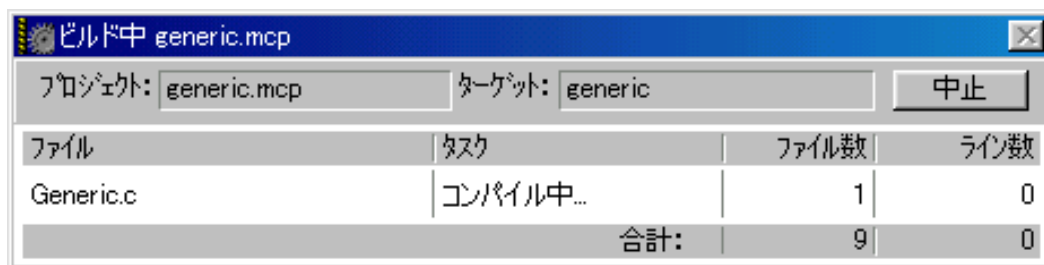
プロジェクトから作成されたアプリケーションなどのバイナリファイルが、バグの下で実行中である

プロジェクトから作成されたアプリケーションが実行中である

CodeWarrior はコンパイルしているソースファイルとライブラリを、プロジェクトウィンドウで強調表示します。「ビルド中」ウィンドウ([図 10.1](#)) にコンパイル中のファイルの名前と行数が表示されます。

プロジェクトウィンドウの下部の状況表示行に、コンパイルする全ファイルの数と、コンパイル済みのファイルの数也表示されます。

図 10.1 「ビルド中」ウィンドウ



1 つのファイルをコンパイル

プロジェクト内の 1 つのファイルをコンパイルするには、プロジェクトウィンドウ内でのファイルを選択して [プロジェクトメニュー](#) の [[コンパイル](#)] を選択します。

他の方法としては、CodeWarrior エディタでファイルを開き、ウィンドウをアクティブにして [プロジェクトメニュー](#) の [[コンパイル](#)] を選択します。

選択したファイルをコンパイル

プロジェクト内の選択したファイルだけをコンパイルすることができます。プロジェクトウィンドウでファイルを選択し、次に [プロジェクトメニュー](#) の [[コンパイル](#)] を選択します。プロジェクト内の複数のファイルを選択する方法は「[ファイルとグループを選択](#)」(p55) を参照してください。

ファイルを再コンパイル

CodeWarrior が変更を認識していないファイルを最初にタッチすることにより、強制的に再コンパイルすることもできます。「[ファイルのタッチ / アンタッチ](#)」(p63) を参照してください。

再コンパイルしたいファイルをタッチした後、[プロジェクトメニュー](#) の [[最新状態に更新する](#)] または [[メイク](#)] を選択します。

リンク順を設定

プロジェクトウィンドウの [リンク順ビュー](#) で、ファイルをコンパイルする順番を指定できます。ファイルの順番を変更すると、ファイルの依存性が原因のリンクエラーを防ぐことができます。つまり、Aleph がコンパイル済みの Zeta に依存している場合、Zeta を Aleph よりも先にコンパイルするとエラーを防げます。

以下はリンク順の設定方法です。

1. プロジェクトウィンドウの [[リンク順](#)] タブをクリックする

プロジェクトウィンドウにリンク順ビューが表示されます。

2. ファイルを並べ替えてリンク順を設定する

ファイルを正しい順にドラッグ&ドロップしてください。

次にプロジェクトを更新、メイク、実行、デバッグするときに、新しい順番でファイルがコンパイルされます。

詳しくは「[リンク順ビュー](#)」(p39) を参照してください。

プロジェクトを更新

新しく追加、修正、またはタッチされたファイルが多数ある場合、[プロジェクトメニュー](#) の [[最新状態に更新する](#)] コマンドですべてのファイルをコンパイルできます。

このコマンドはリンクを起動しないため、プロジェクトのバイナリは出力されません。[[最新状態に更新する](#)] コマンドはコンパイラだけを実行します。

プロジェクトを更新するとき、プロジェクト内の複数ビルドターゲット間でターゲットを切り替えることができます。詳細は「[現在のビルドターゲットを設定](#)」(p68) を参照してください。

プロジェクトをメイク

アプリケーション、ライブラリ、共有ライブラリなどのバイナリファイルを作成する準備が整ったら、[プロジェクトメニュー](#)の[[メイク](#)] コマンドを使います。このコマンドは、新しく追加、修正、および「タッチ」されたファイルを更新してリンクし、実行可能ファイルを生成します。

選択したプロジェクトの種類により、その生成物は異なります。例えばプロジェクトがアプリケーションであれば、[[メイク](#)] コマンドはアプリケーションをビルドして、プロジェクトのフォルダに保存します。[表 10.1](#) にプロジェクトの種類と[[メイク](#)] コマンドが生成するファイルを示します。生成可能なソフトウェアの種類の一覧は、ビルドターゲットの『Targeting』マニュアルを参照してください。どのマニュアルを参照するかは、[表 1.1 \(p21\)](#) を参照してください。

表 10.1 プロジェクトの種類と生成ファイルの例

プロジェクトの種類	ビルドターゲット	生成ファイル
アプリケーション	Win32	Win32 アプリケーション
ライブラリ	68K/PowerPC	Mac OSA4 または A5 ライブラリ
ダイナミックリンクライブラリ	Solaris	Solaris 共有ライブラリ (.o)

修正されたファイルとタッチされたファイルがすべて正常にコンパイルされると、CodeWarrior はプロジェクト内の全ファイルをリンクして、出力バイナリを作成します。プロジェクトが既に[[最新状態に更新する](#)] または他のコマンドを使ってコンパイルされている場合は、[[メイク](#)] コマンドはコンパイル済みのファイルをリンクするだけです。

プロジェクトを作成するとき、プロジェクト内の複数ターゲット間でターゲットを切り替えることができます。詳細は「[現在のビルドターゲットを設定](#)」(p68) を参照してください。

デバッグを有効にする

[プロジェクトメニュー](#)の[[デバッグを有効にする](#)] を選択した後、[デバッグ] コマンドで CodeWarrior デバッグを起動して、プロジェクトをデバッグすることができます。[プロジェクトメニュー](#)の[[デバッグを無効にする](#)] を選択すると、[[実行](#)] コマンドはプログラムを通常に実行します。

デバッグ情報を生成する方法は「[プロジェクトでデバッグを制御](#)」(p73) を参照してください。

プロジェクトの実行方法は「[プロジェクトを実行](#)」を参照してください。

プロジェクトを実行

[プロジェクトメニュー](#)の[[実行](#)]コマンドを選択すると、CodeWarrior は必要な場合、プロジェクトのコンパイル、リンクを行い、スタンドアロンのアプリケーションを作成して起動します。

カレントプロジェクトが実行可能ファイル以外の場合、[実行] コマンドは使えません。実行不可能なファイルとはライブラリ、共有ライブラリ、ダイナミックリンクライブラリ、コードリソース、ツールなどです。

コンパイルとリンクが成功すると、新しいアプリケーションはハードディスクに保存されます。アプリケーションの名前はターゲット設定ダイアログで指定します。名前の変更は「[設定パネルを選択](#)」(p228) を参照してください。

カレントプロジェクトが他のプラットフォーム用アプリケーションであれば、[実行] コマンドを選択する前に、ホストコンピュータをターゲットコンピュータかデバイスに接続する必要があります。詳細は各『Targeting』マニュアルを参照してください。

プロジェクトをデバッグ

プロジェクトをデバッグするには、必要な手順が 2 つあります。まず、プロジェクトをコンパイルおよびリンクしてデバッグ情報を生成しておかなければなりません。プロジェクトのデバッグ方法は「[デバッグを有効にする](#)」(p259) を参照してください。

2 番目の手順は、コンパイルしたアプリケーションをデバッグターゲットとしてデバッグを起動することです。[プロジェクトメニュー](#)から[[デバッグ](#)] コマンドを選択します。[[デバッグ](#)] コマンドがメニューにない場合は、プロジェクトでデバッグを有効にしていないためです。「[デバッグを有効にする](#)」(p259) を参照してください。

[デバッグ] コマンドを選択すると、プロジェクトがコンパイル、リンクされ、デバッグ情報ファイルが作成されます。次いでデバッグがそのデバッグ情報ファイルを開きます。

[[デバッグ](#)] コマンドが灰色で表示されるときは、デバッグに関連するオプションが正しく設定されているかどうかを確かめてください(「[デバッグを有効にする](#)」(p259) を参照)。また、Metrowerks デバッグがハードディスクにあることも確認してください。それでも[[デバッグ](#)] が灰色で表示されていれば、実行不可能なプロジェクト(共有ライブラリまたはライブラリなど)を実行しようとしているか、またはアプリケーションが既に実行されている可能性があります。

注意：[[デバッグ](#)] はプロジェクトをデバッグするためのアプリケーションを開くコマンドではありません。Adobe Photoshop のプラグインなど、他のアプリケーションを必要とするプロジェクトをデバッグするときは、[[デバッグ](#)] を選択する前に必要なアプリケーションを起動しておきます。

カレントプロジェクトが他のプラットフォーム用アプリケーションであれば、[実行] を選択する前にホストコンピュータをターゲットコンピュータかデバイスに接続する必要があります。詳細は各『Targeting』マニュアルを参照してください。

リンクマップを生成

CodeWarrior C/C++ コンパイラは、生成されたオブジェクトコードの関数およびクラスセクション情報を含むリンクマップファイルを作成します。

CodeWarrior Pascal コンパイラは、依存関係とコンパイル順序のリストを含むメイクマップファイルを作成します。

これらのオプションはリンカ設定パネルにあります。プロジェクトの設定が終わったら、プロジェクトをメイクする必要があります。コンパイルとリンクが成功すると、「プロジェクト .MAP」という名前のリンクマップファイルがプロジェクトフォルダ内に作成されます。

リンカ設定パネルの詳細は各『Targeting』マニュアルを参照してください。

修正日時を同期

プロジェクト内のすべてのファイルの修正日時を更新するには、[プロジェクトメニュー](#)の[[ファイル更新日時を同期させる](#)] を選択します。

詳細は「[修正日時の同期](#)」(p63) を参照してください。

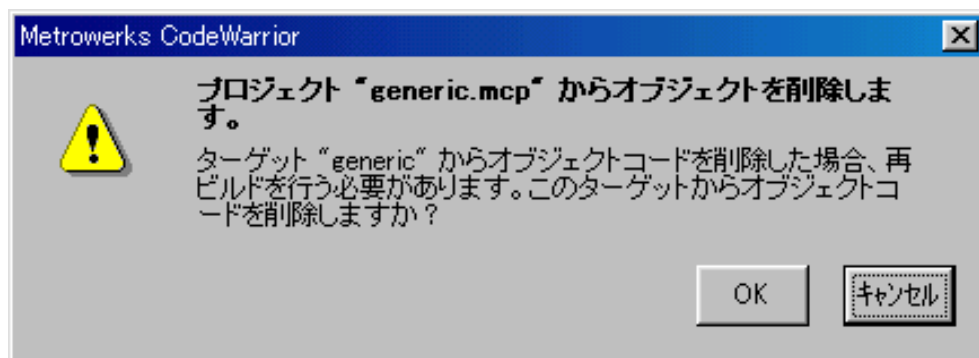
オブジェクトを削除

プロジェクトをコンパイルすると、オブジェクトコードが各ソースファイルからプロジェクトに追加されます。このバイナリオブジェクトコードはプロジェクトファイルのサイズを増加します。ハードディスクの使用量を減らすために、またはすべてのオブジェクトコードを削除して再コンパイルを行うためのコマンドがあります。

オブジェクトコードを削除

ハードディスクの使用量を減らすために、またはすべてのオブジェクトコードを削除して再コンパイルを行うには、[プロジェクトメニュー](#)の[[オブジェクトを削除](#)] コマンドを選択します。オブジェクトを削除するダイアログ ([図 10.2](#)) が現れます。

図 10.2 オブジェクトを削除するダイアログ



[全ターゲット] ボタンをクリックすると、プロジェクト内の全ビルドターゲットのオブジェクトコードがすべて削除され、プロジェクトウィンドウ内の各ファイルのコードおよびデータサイズがゼロにリセットされます。[現在のターゲット] は、現在のビルドターゲットのオブジェクトだけを削除し、その他のビルドターゲットのオブジェクトはそのまま残します。[キャンセル] は操作を中止するので、オブジェクトコードは削除されません。

プロジェクトの現在のビルドターゲットを変更する方法については、[「現在のビルドターゲットを設定」](#) (p68) を参照してください。

オブジェクトコードの削除と圧縮

[オブジェクトコードの削除と圧縮] コマンドは、プロジェクトからすべてのバイナリを削除して圧縮します。ハードディスクの使用量を最小限にします。[「オブジェクトコードを削除」](#) (p261) と似ていますが、ここではプロジェクトファイルが圧縮されます。

プロジェクトを圧縮することにより、プロジェクトファイル内のすべてのオブジェクトコードとデバッグ情報を削除し、プロジェクト固有のオプション設定とファイルだけを残します。

このコマンドの詳細は [「オブジェクトコードの削除と圧縮」](#) (p467) を参照してください。

高度なコンパイルオプション

ここでは、プロジェクトのビルド時間を短縮するオプション、ビルドが完了したときにアラートを出すオプションについて説明します。

日付の検査を省略してビルドを短縮

ビルド速度の最適化については、[「ファイル修正日時キャッシュを利用する」](#) (p236) を参照してください。

プリコンパイルまたはプリプロセスヘッダを使う

プロジェクトのソースファイルでは、一般に多数のヘッダファイル (.h または .hpp ファイル) が使われます。同じヘッダファイルが多くの異なるソースファイルに含まれること

もあり、コンパイル中に同じヘッダファイルを何回も読むことは非効率的です。C/C++ も含め、プログラミング言語のほとんどはプリコンパイルヘッダをサポートしています。

ヘッダファイルのコンパイルおよび再コンパイルの時間を短縮するには、[プロジェクトメニュー](#)の[[プリコンパイル](#)] コマンドを使います。プリコンパイルされたヘッダファイルは、コンパイルされていないヘッダファイルと比較すると、コンパイル時間がいちじるしく短くなります。

例えば、プロジェクト内で頻繁に使われるヘッダを含むヘッダファイルは、プリコンパイルすべきです。プロジェクトの各ソースファイルのヘッダファイルを何回もコンパイルする代わりに、プリコンパイルされたヘッダファイルを一度ロードするだけで済むからです。

注意： 1つのソースファイルには、1つのプリコンパイルヘッダしかインクルードできません。複数のプリコンパイルヘッダをインクルードするとエラーになります。

ヒント： CodeWarrior の新しい機能を実装するために、プリコンパイルヘッダのフォーマットは頻繁に変わります。このため、CodeWarrior をアップデートした際にプリコンパイルヘッダのフォーマットの互換性がなくなることがあります。新しいバージョンの CodeWarrior をインストールした後は、新しいフォーマットを使うために、ヘッダを再度プリコンパイルしてください。プリコンパイルヘッダの更新の詳細は「[自動更新](#)」(p264)を参照してください。

ここでは、以下の項目を説明します。

[プリコンパイルヘッダを作成](#)

[C/C++ 用のシンボルを定義](#)

[Pascal 用のシンボルを定義](#)

プリコンパイルヘッダを作成

プリコンパイルヘッダファイルを作成するには、最初にプロジェクトを開きます。このプロジェクトのオプション設定がプリコンパイルに使われます。プリコンパイルするファイルはヘッダファイル（.h または .hpp）である必要はありませんが、下記の条件は満たす必要があります。

ソースファイルはテキストファイルでなければなりません。ライブラリまたはその他のファイルはプリコンパイルできません。

ファイルは認識可能な拡張子（.pch や .pch++）付きで保存されてなくてはなりません。

ファイルはプロジェクトに含まれている必要はありませんが、プリコンパイルするにはプロジェクトを開く必要があります。

データまたはコードを生成するステートメントを含んではいけません。しかし、C++ ソースコードはインライン関数および定数変数宣言 (`const`) を含むことができます。

異なるビルドターゲットのプリコンパイルヘッダファイルとは交換できません。例えば、Windows 95/98/NT 4.0 コンパイラ用のプリコンパイルヘッダを生成するには、Windows 95/98/NT 4.0 コンパイラが必要です。

ソースファイルは `#include` 疑似命令を使って、1 つのプリコンパイルヘッダファイルをインクルードできます。

プリコンパイルヘッダを作成するには、[ファイルメニュー](#) の [[新規テキストファイル](#)] を選択してテキストファイルを作成します。このファイルに `#include` 疑似命令を置きます。例えば、`string.h` と `stdio.h` のプリコンパイルヘッダファイルを作成する場合、テキストファイルは次のようになります。

```
#include <stdio.h>
#include <string.h>
```

テキストファイルが保存されると、IDE はそれを使ってプリコンパイルヘッダを作成します。プリコンパイルするときにプリコンパイルヘッダの名前を指定します。テキストファイルの最初に以下の行を入れても指定できます。 *name* にプリコンパイルヘッダの名前を入れます。

```
#pragma precompile_target "name"
```

name の後にプリコンパイルヘッダ用の拡張子 `.pch` を付けてください。

プリコンパイルコマンド

保存したテキストファイルをプリコンパイルするには、[プロジェクトメニュー](#) の [[プリコンパイル](#)] を選択します。このコマンドは、アクティブウィンドウのテキストファイルをプリコンパイルしてプリコンパイルヘッダファイルを作成します。コンパイラがエラーを検出した場合は、メッセージウィンドウが現れます。

メッセージウィンドウとコンパイルエラーの修正については、[「コンパイラのエラーと警告を修正」 \(p273\)](#) を参照してください。

プリコンパイルヘッダの自動更新については次の [「自動更新」](#) を参照してください。

自動更新

ソースが修正されている場合、CodeWarrior IDE は [[メイク](#)] または [[最新状態に更新する](#)] の実行中にプリコンパイルヘッダを自動的に更新します。

CodeWarrior IDE はプロジェクトで、最後のプリコンパイル以後に修正されたファイル (`.pch` または `.pch++`) を見つけると、再度プリコンパイルを行ってヘッダを更新します。

自動更新するプリコンパイルヘッダを作るには、それを使っているプロジェクトを開いておきます。次にプリコンパイルヘッダ用のテキストファイルを作成します。

プリコンパイルヘッダファイルの条件は、[「プリコンパイルヘッダを作成」 \(p263\)](#) を参照してください。

テキストファイルの先頭行に次の行を置きます。

```
#pragma precompile_target "name"
```

このプラグマは、コンパイラにプリコンパイルヘッダを *name* という名前で作成するように指示します。

name の後にプリコンパイルヘッダ用の拡張子 *.mch* を付けてください。

テキストファイルの作業を終えたら、適切な拡張子を付けてプロジェクトフォルダに保存します。

C/C++ でプリコンパイルヘッダを作成するために使用するテキストファイルには拡張子 *.pch* を付けて保存してください。

ここで[プロジェクトメニュー](#)の[[プリコンパイル](#)]を選択してください。

[プロジェクトメニュー](#)の[[ウィンドウを追加](#)] コマンドを使って、開いているプロジェクトにテキストファイルを追加します。

プリコンパイルファイルが修正されると、CodeWarrior プロジェクトマネージャがこれを自動的にプリコンパイルして更新します。

プリコンパイルヘッダをプロジェクトのソースファイルにインクルードするには、ソースファイルの先頭に以下の `#include` 疑似命令を追加します。

```
#include "name"
```

または、プリコンパイルヘッダファイルをプリフィクスファイルとして使うこともできます。詳細は『C Compilers Reference』マニュアルの C/C++ 言語設定パネルについての説明を参照してください。

注意： プリコンパイルヘッダ用テキストファイル (*.pch*、*.pch++*) を `#include` 疑似命令に使わないでください。生成したプリコンパイルヘッダファイルの名前を指定してください。プリコンパイルヘッダ用テキストファイルの名前を使用しても、最終のバイナリに何も効果がなく、プリコンパイルヘッダによるコンパイル時間短縮にはなりません。

C/C++ 用のシンボルを定義

定義済みのシンボルおよびその他のプリプロセッサ疑似命令を自動的に更新して追加するには、プリコンパイルヘッダファイルを作成してプロジェクトに追加し、C/C++ 言語設定パネルでそれをプリフィクスファイルにインクルードします。

1. 新しいテキストファイルを作成する

プロジェクトを開いて[ファイルメニュー](#)の[[新規テキストファイル](#)] コマンドで、新しいテキストファイルを作成します。

新しいテキストファイルに、プリプロセッサ疑似命令を入れます。このファイルをプリコンパイルヘッダファイルとして使い、プロジェクトに追加します。

2. C/C++ 言語設定パネルを開く

[編集メニュー](#)の [[Target の設定](#)] を選択し、C/C++ 言語設定パネルを選択します。実際のコマンド名にはカレントターゲットの名前が入ります。ターゲット設定ダイアログでC/C++言語設定パネルを選択してください。

3. プリフィックスファイルの名前を取得する

[プリフィックスファイル] フィールドにファイル名があればクリップボードにコピーしてから、「ターゲット設定」ダイアログを閉じます。

新しいテキストファイルで、[プリフィックスファイル] フィールドからコピーした名前を `include` 疑似命令にペーストします。ファイルで「第 1 の」疑似命令であることを確認してください。

例えば、プリフィックスファイルが `MyHeaders` であれば、編集集中のテキストファイルの最初の疑似命令は以下のようになります。

```
#include <MyHeaders>
```

4. #pragma ステートメントを追加する

`#pragma precompile_target` ステートメントをテキストファイルへ追加します。このステートメントはプリコンパイルファイルの名前を指定します。例えば、`MyPrecomp` という名前のファイルを作成するには、以下のステートメントを使います。

```
#pragma precompile_target "MyPrecomp"
```

名前の後にプリコンパイルヘッダ用の拡張子 `.mch` を付けてください。上記の例では、ファイル名は `MyPrecomp.mch` としてください。

5. プリプロセッサ疑似命令を入力する

独自の `#define`、`#include`、およびソースファイルに必要なプリプロセッサ疑似命令を入力してください。実行可能コードを生成するソースコードをテキストファイルに含めることはできません。しかし C++ ソースコードにはインライン関数と定数変数宣言 (`const`) を含めることができます。

6. テキストファイルを保存する

[プロジェクトメニュー](#)の [[ウィンドウを追加](#)] を選択して、このテキストファイルをプロジェクトに追加します。テキストに適切な拡張子を付けてプロジェクトフォルダへ保存してください。

C でプリコンパイルヘッダを作成するために使用するテキストファイルには拡張子 `.pch` を付けて保存してください。C++ でプリコンパイルヘッダを作成するために使用するテキストファイルには拡張子 `.pch++` を付けて保存してください。

7. テキストファイルをプリコンパイルする

[プロジェクトメニュー](#)の [[プリコンパイル](#)] を選択してください。IDE はプリコンパイルヘッダファイルを作成します。

8. C/C++ 言語設定パネルを開く

[編集メニュー](#)の[[Target の設定](#)](*Target*には現在のターゲットの名前が入ります) を選択して C/C++ 言語設定パネルを開いてください。

9. 新しいプリフィックスファイルを指定する

[プリフィックスファイル] フィールドに、プリコンパイルヘッダの名前(この例では MyPrecomp) を入力します。[OK] ボタンをクリックして変更を保存します。

この例では [プリフィックスファイル] フィールドに MyPrecomp.mch と入力します

プロジェクトをビルドすると、CodeWarrior プロジェクトマネージャがプリコンパイルヘッダを自動更新し、各ソースファイルに自動的にインクルードします。

Pascal 用のシンボルを定義

Pascal のプリプロセッサはC/C++ プリプロセッサほど強力ではありませんが、自分で書いたプリプロセッサシンボルやコンパイラ疑似命令をプロジェクトに自動的に挿入することはできます。Pascal コンパイラ疑似命令の詳細は、CodeWarrior CD の『Pascal Compilers Reference』マニュアルを参照してください。

1. 新しいテキストファイルを作成する

プロジェクトを開いて、[[新規テキストファイル](#)] コマンドで新しいテキストファイルを作成します。

この新しいテキストファイルに、コンパイラ疑似命令を入れます。

2. Pascal 言語設定パネルを開きます

[編集メニュー](#)の[[Target の設定](#)] を選択し、ターゲット設定ダイアログから Pascal 言語設定パネルを開いてください。

3. プリフィックスファイルの名前を取得する

[プリフィックスファイル] フィールドにファイル名があればクリップボードにコピーしてから、「ターゲット設定」ダイアログを開いてください。

新しいテキストファイルで、[プリフィックスファイル] フィールドからコピーした名前を include 疑似命令 { \$I } にペーストします。ファイルで「第 1 の」疑似命令であることを確認してください。

例えば、プリフィックスファイルが OtherDefs.p であれば、テキストファイルの最初の疑似命令は以下ようになります。

```
{ $I OtherDefs.p }
```

4. { \$SETC }、{ \$I }、その他のプリプロセッサ疑似命令を入力する

テキストファイルには、実行可能コードを生成するソースコードは入れられません。

5. テキストファイルを保存する

このファイルを通常の Pascal ソースファイルとして、プロジェクトと同じフォルダに保存します。例えば、MyPrecomp.pas というファイル名で保存します。

6. テキストファイルをプリコンパイルする

[プロジェクトメニュー](#)の [[プリコンパイル](#)] を選択してください。IDE はテキストファイルからプリコンパイルヘッダファイルを作成します。

7. Pascal 言語設定パネルを開く

[編集メニュー](#)の [[Target の設定](#)] (*Target* には現在のターゲット名が入ります) を選択して Pascal 言語設定パネルを開いてください。

8. 新しいプリフィックスファイルを指定する

[プリフィックスファイル] フィールドにファイルの名前 (この例では MyPrecomp) を入力し、[保存] ボタンをクリックして変更を保存します。

プロジェクトをビルドするたびに、CodeWarrior プロジェクトマネージャが各ソースファイルに自動的にインクルードします。

ソースコードをプリプロセス

プリプロセッサは、コンパイラで使えるようにソースコードを準備します。# と \$ シンボルで始まる疑似命令 (#define、#pragma、#ifdef など) を解釈して、余計なスペースや空白行、コメント (/*.....*/、//) を除きます。コンパイルを実行する前に、ファイルをプリプロセスして、コードがどのようになるのかを見ることができます。

プリプロセスするファイルを開くか、または現在開いているプロジェクトウィンドウでファイルを選択します。ファイルをプリプロセスするには、[プロジェクトメニュー](#)の [[プリプロセス](#)] コマンドを使います。プリプロセス後のファイルは、プリプロセス前のソースファイル名に接頭子 # を付けた名前で保存されます。

新しいウィンドウの内容を保存するには[ファイルメニュー](#)の保存コマンドの 1 つを選択します。

ソースコードを逆アセンブル

生成済みのファイルのコードを見たい場合は、逆アセンブルします。逆アセンブルは、ソースコードを実行するときに実行されるマシンレベルのコードを調べたいときに役立ちます。さらに、逆アセンブルしたコードはアセンブラ関数を書くときの参考にもなります。ライブラリファイルもこのコマンドを使って調べられます。

[プロジェクトメニュー](#)の [[逆アセンブル](#)] コマンドは、プロジェクトウィンドウで選択されたコンパイル済みのソースファイルを逆アセンブルして、アセンブリ言語コードを新しいウィンドウに表示します。新しいウィンドウのタイトルは、ソースファイルの名前に拡張子 .dump が付いたものになります。

.dump ウィンドウの内容を保存するには、[ファイルメニュー](#)の保存コマンドの1つを使います。

逆アセンブルしようとするファイルがまだコンパイルされていない場合は、逆アセンブルコマンドはファイルをコンパイルしてから逆アセンブルします。

メッセージウィンドウの解説

メッセージウィンドウは([図 10.3](#)) コンパイル、リンク、またはファイル検索のときに起きたイベントのメッセージを表示します。エラーの位置を探したり、プロジェクトのすべてのメッセージを見るためにスクロールしたり、ウィンドウ内での作業を有効に行うための多くの要素があります。

メッセージウィンドウのユーザーインターフェース項目の中で、ここでは説明されていないものがいくつかあります。[マーカーポップアップメニュー](#)、[オプションポップアップメニュー](#)、[ファイルパス表示欄](#)、[行番号](#)などです。詳細は、「[エディタウィンドウの解説](#)」(p95)を参照してください。

ここでは、以下の項目を説明します。

[エラーボタン](#)

[警告ボタン](#)

[プロジェクト情報欄](#)

[エクストラ情報ボタン](#)

[ステップボタン](#)

[メッセージリスト欄](#)

[ソースコード拡張ボタン](#)

[ソース欄](#)

[欄のサイズ変更バー](#)

エラーボタン



メッセージウィンドウのエラーボタンは、エラーメッセージの表示のオン、オフを切り替えます。これは、ウィンドウの表示を他の目的に切り替えた後でエラーメッセージ表示に戻すときなどに便利です。

メッセージウィンドウ内で警告メッセージを見る方法は「[エラーと警告を見る](#)」(p271)を参照してください。

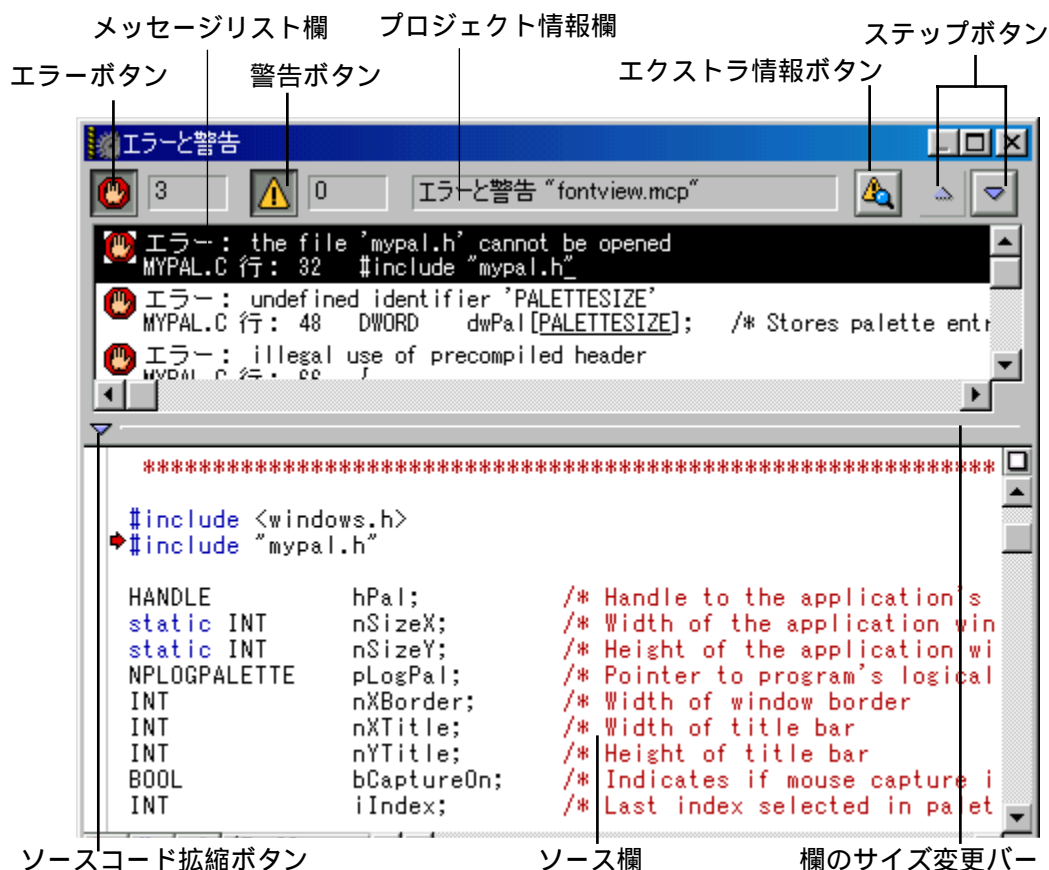
警告ボタン



メッセージウィンドウの警告ボタンは、警告メッセージの表示のオン、オフを切り替えます。

メッセージウィンドウ内で警告メッセージを見る方法は「[エラーと警告を見る](#)」(p271) を参照してください。

図 10.3 メッセージウィンドウ



プロジェクト情報欄

プロジェクト情報欄は、メッセージウィンドウに表示中のビューの短い説明を表示します。プロジェクトの名前はここに表示されます。

エクストラ情報ボタン

エクストラ情報ボタンはメッセージを拡張してプロジェクト、ターゲット、ファイルに関する情報を表示します。

ステップボタン

ステップボタンは、ウィンドウ内の上または下のメッセージに移動するために使います。

メッセージの移動については、「[メッセージを 1 つずつ見る](#)」(p272) を参照してください。

メッセージリスト欄

メッセージリスト欄は、メッセージを表示します。

メッセージウィンドウにメッセージを表示する方法は、[「エラーと警告を見る」\(p271\)](#)を参照してください。

ソースコード拡張ボタン

ソースコード拡張ボタンを使って、メッセージウィンドウのソースコード列を表示したり、隠すことができます。

ソース欄

メッセージウィンドウのソース欄には、メッセージが参照しているソースコードの位置を表示できます。このウィンドウの詳細は、[「エラーと警告を見る」\(p271\)](#)を参照してください。

欄のサイズ変更バー

欄のサイズ変更バーを使うと、メッセージウィンドウ内のソースコード欄とメッセージリスト欄の割り振りを変えることができます。このバーをクリックしてドラッグすることにより、欄のサイズを変えられます。

メッセージウィンドウを使う

プロジェクトのコンパイル中に、CodeWarrior IDE がシンタックスエラーやその他の種類のコンパイラエラーを検出することがあります。エラーを検出すると、エラーおよび警告の総数と、それぞれに関する情報をメッセージウィンドウに表示します。ウィンドウのアイテムについては[「メッセージウィンドウの解説」\(p269\)](#)を参照してください。

ここでは、メッセージウィンドウに表示される情報の解釈、ナビゲート、および使い方を説明します。以下の項目を説明します。

[エラーと警告を見る](#)

[メッセージを1つずつ見る](#)

[コンパイラのエラーと警告を修正](#)

[リンクのエラーを修正](#)




[Pascal の循環参照を修正](#)

[メッセージウィンドウの保存と印刷](#)

[変更したファイルでエラーを見つける](#)

エラーと警告を見る

メッセージウィンドウは、下記の種類のメッセージを表示します。

アイコン		アイコンを選択したときに表示される内容
	エラー	コンパイラまたはリンカのどちらかのエラー。どちらのエラーも、最終バイナリの作成ができません。
	警告	コンパイラまたはリンカのどちらかの警告。バイナリの作成は妨げません。ただし、これらは実行中に問題が発生する可能性を示します。警告メッセージにつながる条件を指定するか、またはすべての警告をエラーの扱いにすることができます。
	注意	メッセージウィンドウで発行される、他のすべての種類のメッセージ。例えば、バッチ検索の結果は通知メッセージです。

メッセージウィンドウを閉じるには、メッセージウィンドウをアクティブにしてクローズボックスをクリックするか、または[ファイルメニュー](#)の[[閉じる](#)]を選択します。メッセージウィンドウを閉じた後で、再度表示したい場合は、[ウィンドウメニュー](#)の「[エラーと警告ウィンドウ](#)」を選択します。

[メッセージリスト欄](#)にエラーメッセージを表示するには、[エラーボタン](#)をオン、[警告ボタン](#)をオフにします。

[メッセージリスト欄](#)に警告を表示するには、[警告ボタン](#)をオン、[エラーボタン](#)をオフにします。

[メッセージリスト欄](#)にエラーと警告の両方を表示するには、両方のボタンをオンにします。注意は、「エラーと警告」ウィンドウには表示されません。

他の種類のメッセージも必要に応じてメッセージウィンドウに表示されます。

[[ウィンドウを追加](#)] または [[ファイルを追加](#)] で、既存のアクセスパスにないファイルが追加されたとき

リンク中にプロジェクトに矛盾するリソースが見つかったとき

「検索」ダイアログの [バッチ] チェックボックスをオンにして検索を行ったとき

メッセージを1つずつ見る

コンパイラがビルド中にエラーを見つけたとき、または [[バッチ検索](#)] で CodeWarrior IDE の検索コマンドが検索しているテキストを見つけたときにメッセージウィンドウが表示されます。このウィンドウは2つの欄に分かれています。

[メッセージリスト欄](#)：メッセージをリストします。

[ソース欄](#)：選択したメッセージのソースコードを表示します。

ウィンドウの詳細は「[メッセージウィンドウの解説](#)」(p269) を参照してください。

メッセージのリストを上または下に移動するには、上向きまたは下向きの[ステップボタン](#)をクリックするか、または直接エラーメッセージをクリックします。

特定のメッセージに対して [ソース欄](#) に表示されたソースコード内をナビゲートするには、[ヘッダポップアップメニュー](#)、[関数ポップアップメニュー](#)、または[行番号](#)を使います。ナビゲーション機能の使い方については、「[エディタウィンドウの解説](#)」(p95) を参照してください。

コンパイラのエラーと警告を修正

コンパイル中にエラーが発生したとき、メッセージウィンドウはこのエラーのメッセージを [メッセージリスト欄](#) に表示します。メッセージが参照するソースコードの位置を [ソース欄](#) に表示します。エラーの起きたソースコード位置に簡単にナビゲートできるため、ここでコードの検査および修正ができます。

コンパイラのエラーのリストと原因については、ヘルプメニューの『Error Reference』マニュアルを参照してください。

ソース欄でエラーを修正

コンパイラエラーまたは警告を修正するにはまず原因を見つけます。最初に、メッセージウィンドウの [ソース欄](#) が表示されていることを確認してください。表示されていなければ「[ソースコード拡張ボタン](#)」(p271) を参照してください。

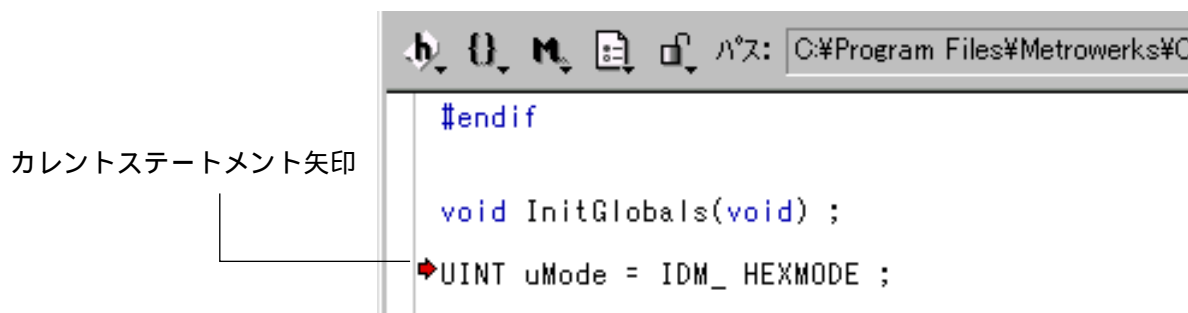
エラーまたは警告の原因と判断されたステートメントを表示するには、メッセージウィンドウの [メッセージリスト欄](#) でメッセージを選択します。 [ソース欄](#) にそのメッセージに対応するソースコードが表示されます。ステートメント矢印 ([図 10.4](#)) が原因のコードを指し、ここで直接修正できます。

[ソース欄](#) 内で [ヘッダポップアップメニュー](#)、[関数ポップアップメニュー](#)、または [行番号](#) を使って、コードをナビゲートしたり、ヘッダファイルを開いたりできます。ナビゲーション機能の使い方については、「[エディタウィンドウの解説](#)」(p95) を参照してください。

メッセージに対応するファイルを開く

特定のメッセージに対応するソースファイルを開くには、[メッセージリスト欄](#) でメッセージを選択し、Enter キーを押します。関連するファイルを開くには、[メッセージリスト欄](#) 内のメッセージをダブルクリックします。

図 10.4 エラーを示すステートメント矢印



リンクのエラーを修正

プロジェクトのリンク時に発生したエラーを表示して修正することができます。

リンクエラーを表示

プロジェクトをリンク中にリンクでエラーが検出されると、メッセージウィンドウにエラーが表示されます。このウィンドウは、スクロールバーまたは[ステップボタン](#)を使ってスクロールすることができます。

メッセージウィンドウ内でメッセージをスクロールする方法は、「[ステップボタン](#)」(p270)を参照してください。ウィンドウ内のメッセージの表示を変更する方法は、「[エラーと警告を見る](#)」(p271)を参照してください。

リンクエラーはオブジェクトコードの問題が原因のため、プロジェクトのソースファイル中のエラーに対応する箇所を示すことはできません。

リンクエラーの原因

リンクエラーは通常、下記のような原因で発生します。

ライブラリ関数の名前の綴りの間違い：つまり、リンクが検索している関数が見つからなかったことを意味します。関数の名前を検査して、綴りが正しいことを確認してください。

コンパイルしたコードが、選択したコードモデルでは処理できない参照を生成しています。

プロジェクトに必要なライブラリの欠落：プロジェクトが必要とするライブラリが見つからない場合に発生します。使用するライブラリについては、各『Targeting』マニュアル（[表 1.1 \(p21\)](#)）を参照してください。

Pascal の循環参照を修正

CodeWarrior Pascal コンパイラの [[メイク](#)] および [[実行](#)] コマンドは、プロジェクトファイル内のすべての Pascal ファイルを検査して、プロジェクトをビルドします。この検査が実行されると、ユニットのインターフェースおよびその実装の依存関係のツリーが構築されます。

ユニットが別のユニットを宣言し、後者の中で元のユニットが宣言されている場合、循環参照が発生します。このループを切るために、Pascal コンパイラはユニットのインターフェース部品でこのような参照を許していませんが、実装では許可されています。

[例 10.1](#) の例では、A と B のインターフェースは C に依存していますが、互いに独立なため、有効です。すべてのインターフェースが宣言されているので、A の実装はすべてのインターフェースに依存し、B と C についても同様のことが言えます。この例では、メイクユーティリティはコンパイラに[例 10.1](#) を次の順序でコンパイルするように指示します。

例 10.1 有効な循環参照

UNIT A; INTERFACE USES C;	UNIT B; INTERFACE USES C;	UNIT C; INTERFACE
TYPE A_type = ...	TYPE B_type =	TYPE C_type = ...
IMPLEMENTATION USES B;	IMPLEMENTATION USES A; ...	IMPLEMENTATION USES A, B; ...

1. C のインターフェースをコンパイルする
2. B のインターフェースをコンパイルする
3. ユニット A のすべて (ユニットと実装) をコンパイルする
4. B の実装をコンパイルする
5. C の実装をコンパイルする

インターフェースのコンパイルの後、コンパイラはインターフェースのすべての宣言を含むバイナリシンボルテーブルをプロジェクトファイルの `sbmf` リソースに出力します。この情報は、他のファイルのコンパイルで、ユニット名が `USES` 文で使われたときに読み出されます。ユニットの再コンパイルは、次の条件のいずれかが起きたときにのみ行われま

- ソースが修正されたとき
- ソースが現在開かれていて編集されている
- ソースが依存するユニットが再コンパイルされたとき

メッセージウィンドウの保存と印刷

メッセージウィンドウの内容を保存または印刷する手順を説明します。

メッセージウィンドウを印刷する

1. メッセージウィンドウをアクティブにする

アクティブになっていないメッセージウィンドウをクリックするか、または[ウィンドウメニュー](#)の [[エラーと警告ウィンドウ](#)] コマンドを選択します。
2. ファイルメニューの [印刷] コマンドを選択する

[ファイルメニュー](#)の[[印刷](#)] コマンドを選択すると、メッセージウィンドウを印刷するためのダイアログが表示されます。印刷オプションを指定して[OK] ボタンをクリックします。すべてのエラー、警告、およびメッセージが印刷されます。

印刷の詳細は、プリンタのマニュアルを参照してください。

メッセージウィンドウを保存する

1. メッセージウィンドウをアクティブにする

これには、アクティブになっていないメッセージウィンドウをクリックするか、または [ウィンドウメニュー](#)の[[エラーと警告ウィンドウ](#)] コマンドを選択します。

2. ファイルメニューの[[コピーを保存](#)] コマンドを選択します。

[[コピーを保存](#)] コマンドは、次のダイアログを表示します。

3. 名前と位置をダイアログに指定します。

メッセージウィンドウにリストされているすべてのエラー、警告、およびメッセージを含むテキストファイルが保存されます。

変更したファイルでエラーを見つける

エラーを修正したり、ソースコードを変更すると、コンパイラがソースファイル中の別のエラーを見つけられなくなります。エラーの位置を見つけられないことを示す警告が表示されるのでこれを知ることができます。この状態になったら、プロジェクトを再コンパイルしてメッセージウィンドウ内のエラーのリストを更新します。

第 11 章 ソースコードのデバッグ

この章では CodeWarrior のソースレベルデバッガについて説明します。CodeWarrior デバッガはさまざまなプロセッサ、オペレーティングシステム、言語（C/C++、Pascal、Java、アセンブラ）に対応しています。

デバッガはプログラムの実行をコントロールします。デバッガを使うと、実行中のプログラムの問題点を見つけることができます。デバッガは、プログラムを 1 ステップずつ実行したり、希望する箇所で実行を停止したり、指定したメモリ位置の値を変更したときにプログラムに割り込むことができます。デバッガがプログラムを停止した後、関数のコールチェーンを見たり、変数の値を調べたり、プロセッサのレジスタ内容を検証することができます。

注意： この章ではすべてのプラットフォームに共通するデバッガの機能について説明します。ビルドターゲットによってデバッガの機能は若干異なります。ビルドターゲットによる差違については各『Targeting』マニュアルを参照してください。『Targeting』マニュアルの内容は「[ターゲットマニュアル](#)」(p20) で解説しています。

ここでは次の項目について説明します。

[シンボルファイル](#)

[デバッグの準備](#)

[デバッガを使う](#)

[デバッガのウィンドウ](#)

[基本的なデバッグ](#)

[評価式](#)

[トラブルシューティング](#)

シンボルファイル

プロジェクトのシンボルファイルには、プロジェクトをデバッグするためにデバッガが必要とする情報が含まれています。ルーチン名や変数名（シンボル）、ソースコードのどの位置にそれらのシンボルがあるか、オブジェクトコードのどこにそれらがあるか、という情報も含まれています。

デバッガはシンボルファイルを使ってオブジェクトコードに対応するソースコードを表示します。プログラムを停止したときにソースコードを表示します。

対応するアセンブリ言語のインストラクションとメモリアドレスを見ることができます。
[「ソースコードをアセンブラで見る」\(p286\)](#) を参照してください。

CodeWarrior はその他のビルドターゲットのシンボルフォーマットもサポートします（[表 11.1](#)）。

注意：（エンベデッドシステム）ビルドターゲットによっては、シンボル情報がなくても実行可能ファイルをデバッグすることができます。詳細はビルドターゲットの『Targeting』マニュアルを参照してください。

表 11.1 サポートするシンボルファイルフォーマット

フォーマット	ビルドターゲット
CodeView	Win32
DWARF	エンベデッドシステム
SYM	Mac OS

シンボルファイルを生成するためのプロジェクトとソースファイルの設定については [「デバッグの準備」\(p278\)](#) を参照してください。

コンパイラやリンカの設定についての詳細は [「ターゲットのオプション設定」\(p225\)](#) を参照してください。

ビルドターゲットに特有のシンボル情報については各『Targeting』マニュアルを参照してください。

デバッグの準備

あるターゲット用に生成された CodeWarrior プロジェクトのファイルをデバッグするには、ビルドターゲットとそれに含まれる各ファイルの両方に、デバッグ用の設定をしなければなりません。ビルドターゲットとソースファイルを正しく設定すると、CodeWarrior はデバッガに必要なシンボル情報を生成します。

注意：（エンベデッドシステム）ビルドターゲットによっては、プロジェクトファイルを作成しなくともデバッグすることができます。この場合、デバッガは自動的にデフォルトのプロジェクトファイルを作成して、デフォルトの設定を行います。詳細はビルドターゲットの『Targeting』マニュアルを参照してください。

ここでは以下の項目について説明します。

[デバッグのためにビルドターゲットを設定](#)

[デバッグのためにファイルを設定](#)

シンボル情報を生成

デバッグのためにビルドターゲットを設定

デバッグ用の設定は最初にビルドターゲットをアクティブなターゲットにします。次に CodeWarrior IDE のプロジェクトメニューの [[デバッグを有効にする](#)] を選択します。デバッグが可能になるとプロジェクトメニューの [デバッグを有効にする] コマンドは [デバッグを無効にする] へ変わります。デバッグ機能をオフにするには、[デバッグを無効にする] を選択してください。メニューは [デバッグを有効にする] に戻ります。

[デバッグを有効にする] コマンドはデバッグを有効にします。プロジェクトのターゲットが正しく設定された後、コンパイラとリンカはソースレベルのデバッグに必要な情報を含むシンボルファイルを生成します。デフォルトではシンボルファイルは出力ディレクトリと同じ場所に保存されます。

ビルドターゲットの設定の詳細は「[ターゲットのオプション設定](#)」(p225) を参照してください。シンボルファイルの役割とデバッグとの関係については、「[シンボルファイル](#)」(p277) を参照してください。

[デバッグを有効にする] コマンドを選択すると警告が表示されることがあります([図11.1](#))。これはプロジェクトをデバッグするためにビルドターゲットの設定を変更すること確認するものです。[はい] ボタンをクリックしてデバッグ用の設定を適用してください。

図 11.1 デバッグ用の設定を適用する

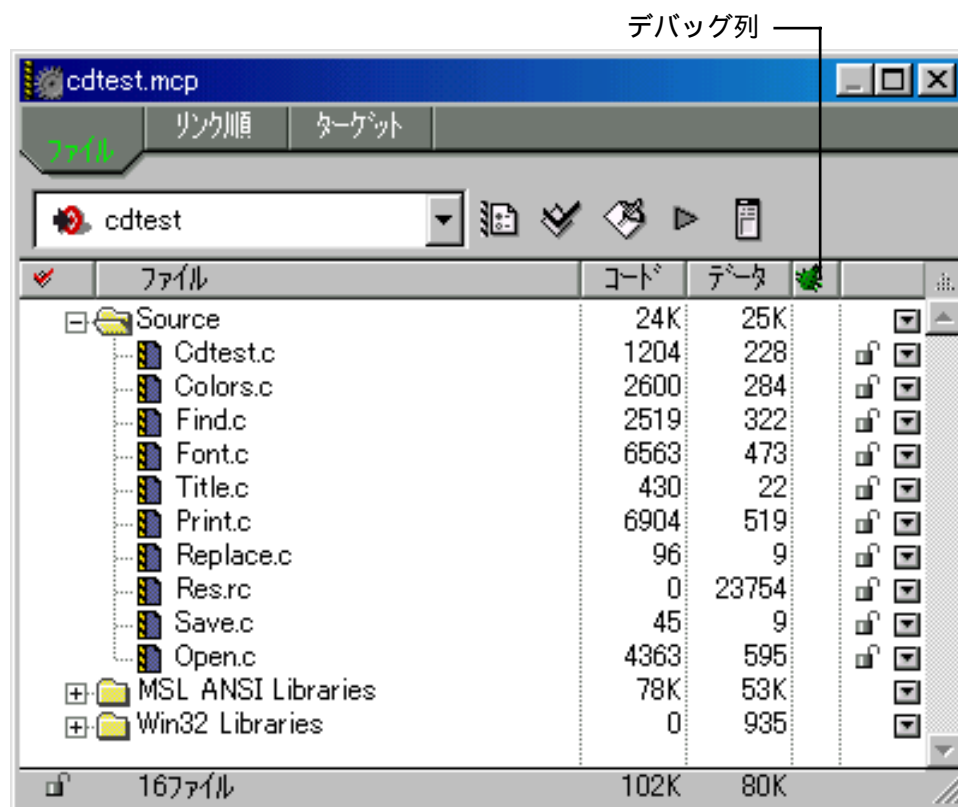


デバッグのためにファイルを設定

現在のビルドターゲットに対してデバッグ用の設定を行った後、個々のファイルのデバッグ用設定を行います。プログラムをデバッグする際は、通常すべてのソースファイルのデバッグマーカをオンにします。

CodeWarrior IDE のプロジェクトウィンドウには、デバッグ列があります([図 11.2](#))。デバッグ列にマーカ (黒い点) のあるファイルに対してシンボル情報が生成されます。マーカがなければシンボル情報は生成されません。グループ名のデバッグ列にマーカがあるとき、そのグループに属する全ファイルがデバッグ可能です。マーカがないとき、そのグループに属する一部のファイルがデバッグできません。

図 11.2 プロジェクトウィンドウのデバッグ設定



ファイルに対するデバッグ機能をオン / オフにするには、ファイル名の横のデバッグ列をクリックします。グループ名の横のデバッグ列をクリックすると、グループに属する全ファイルのデバッグ機能が、オン / オフに切り替わります。ファイルがデバッグ不可能な場合（ファイルがソースファイルではないなど）、そのファイルについてはデバッグをオンにできません。

シンボル情報を生成

シンボル情報を生成するためには、現在のビルドターゲットとソースファイルの両方を正しくデバッグ用に設定しなくてはなりません。詳細は「[デバッグのためにファイルを設定](#)」(p279) を参照してください。

注意：（エンベデッドシステム）ビルドターゲットによっては、シンボル情報がなくても実行可能ファイルをデバッグすることができます。詳細はビルドターゲットの『Targeting』マニュアルを参照してください。

現在のターゲットとソースファイルの設定を終えたら、プロジェクトメニューの[メイク]を選択します。最終的なコードがコンパイル、リンクされます。

コンパイルとリンクについての詳細は「[ターゲットのオプション設定](#)」(p225)、およびそれぞれの『Targeting』マニュアルを参照してください。

デバッガを使う

プロジェクトをデバッグするには、まずデバッガを有効にします。プロジェクトメニューの「[デバッガを有効にする](#)」コマンドを選択してください。デバッガが有効になった後、「[デバッガを有効にする](#)」コマンドは「[デバッガを無効にする](#)」に変わります。

デバッガを有効にしたとき、プロジェクトメニューの「実行」コマンドは「デバッグ」に変わります。「デバッグ」コマンドを選択すると、IDE はプロジェクトをコンパイル、リンクしてシンボルファイルを生成し、デバッガを立ち上げます。デバッガは自動的にシンボルファイルをロードしてプロジェクトのデバッグを始めます。詳細は「[デバッグの準備](#)」(p278)を参照してください。

IDE では起動できないビルドターゲットやコードがあります。例えば、アプリケーションのプラグインはそれのみでは実行できません。また CodeWarrior IDE にはそのプラグインを使うアプリケーションも不明です。このような場合、プラグインを呼び出すために起動するアプリケーションを指定します。

通常、プラグインをデバッグするためにはシンボルファイルを開きます。以下の方法もあります。

1. プロジェクトのターゲット設定ダイアログを開く

プロジェクトメニューの「*Target* の設定」を選択してください。*Target* には現在のビルドターゲットの名前が入ります。例えば現在のビルドターゲット名が MyAppRelease であれば、コマンドは「MyAppRelease の設定」になります。

2. ランタイム設定パネルを表示する

ターゲット設定ダイアログ左側のリストをスクロールして「ランタイム設定」をクリックしてください。右側にパネルが表示されます。

3. 「選択」ボタンをクリックする

ランタイム設定パネルの「ライブラリとコードリソースのためのホストアプリケーション」フィールドに共有ライブラリ、ダイナミックリンクライブラリ (DLL)、コードリソースをデバッグするときに起動するアプリケーションを指定します。プラグインをデバッグするときに指定したアプリケーションが起動されます。

デバッガのウィンドウ

ここではデバッガの各種ウィンドウ、表示欄、および表示される項目について説明します。次の項目を説明します。

[プログラムウィンドウ](#)

[ブラウザウィンドウ](#)

[シンボルのヒント](#)

[デバッガのコンテキストメニュー](#)

[プロセスウィンドウ](#)

[Expression ウィンドウ](#)

[大域変数ウィンドウ](#)

[ブレークポイントウィンドウ](#)

[Watchpoint ウィンドウ](#)

[レジスタ詳細ウィンドウ](#)

[レジスタウィンドウ](#)

[Log ウィンドウ](#)

[変数ウィンドウ](#)

[配列ウィンドウ](#)

[メモリウィンドウ](#)

プログラムウィンドウ

デバッガがシンボルファイルを開くと、プログラムウィンドウ（スタッククロールウィンドウとも呼ばれます）が現れます（[図 11.3](#)）。一度に複数のシンボルファイルを開くことができます。つまり、一度に複数のプログラムのデバッグが可能です。例えばアプリケーションと、そのアプリケーションのためのプラグインを同時にデバッグできるのです。各シンボルファイルとスレッドごとにプログラムウィンドウが表示されます。

プログラムウィンドウは、現在停止しているルーチンに関するデバッグ情報を表示します。このウィンドウには 4 つの欄があります。

[デバッガツールバー](#)

[スタッククロール欄](#)

[変数欄](#)

[ソース欄](#)

欄の境界線の間をクリック + ドラッグして、各欄の大きさを自由に変更することができます。欄を移動するには、Tab キーを使います。

タイプahead選択は、スタッククロール欄および変数欄で有効です。アイテム名の最初の数文字を入力してください。アクティブな欄の中を移動するには、矢印キーを使います。

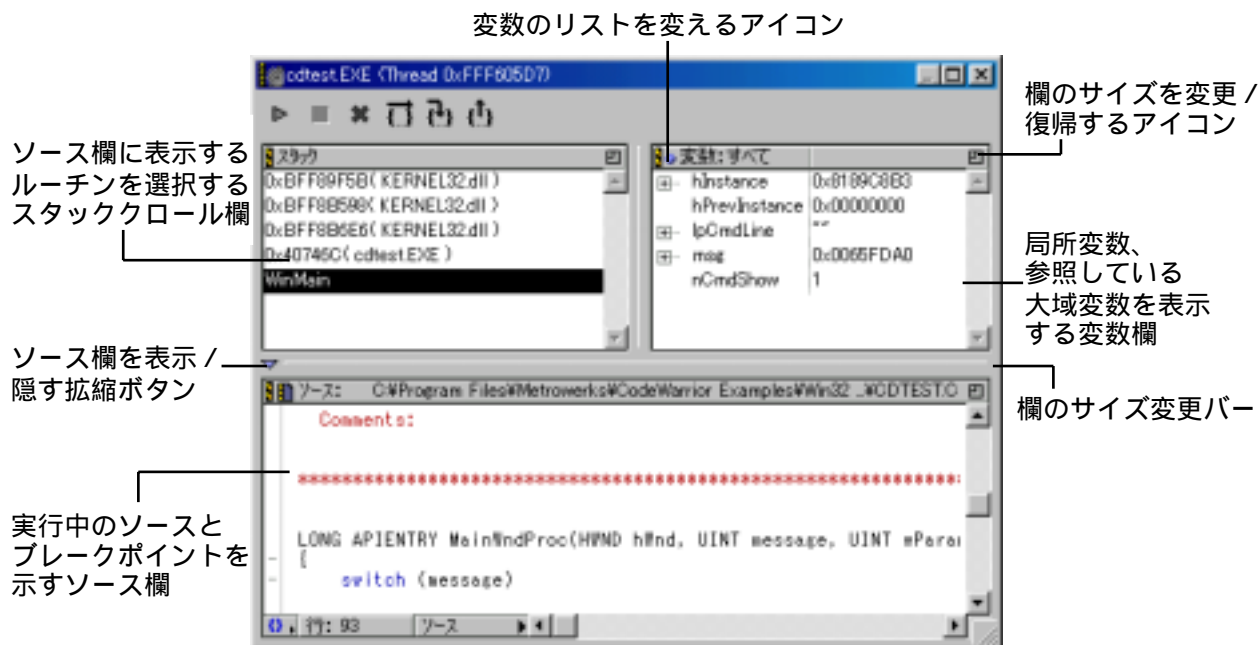
ソース欄の一番下の水平スクロールバーの左側に、その他のコントロール項目があります。

[関数ポップアップメニュー](#)

[行番号](#)

[ソース欄](#) ポップアップメニュー

図 11.3 プログラムウィンドウの各部



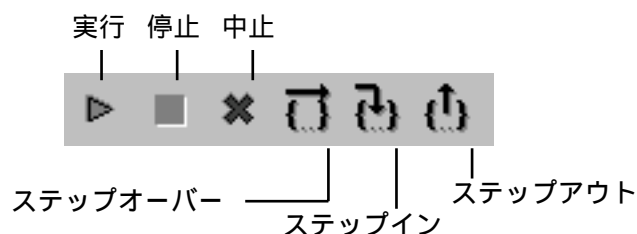
デバッガツールバー

ツールバー (図 11.4) には、コントロールメニューの [実行] [停止] [中止] [ステップオーバー] [ステップイン] [ステップアウト] を実行するコマンドのボタンがあります。

ウィンドウメニューの [ツールバー] の [[ウィンドウツールバーを表示](#)] を選択するとツールバーが現れます。ツールバーを隠すには、ウィンドウメニューの [ツールバー] の [[ウィンドウツールバーを閉じる](#)] を選択します。

詳細は「[基本的なデバッグ](#)」(p311) を参照してください。

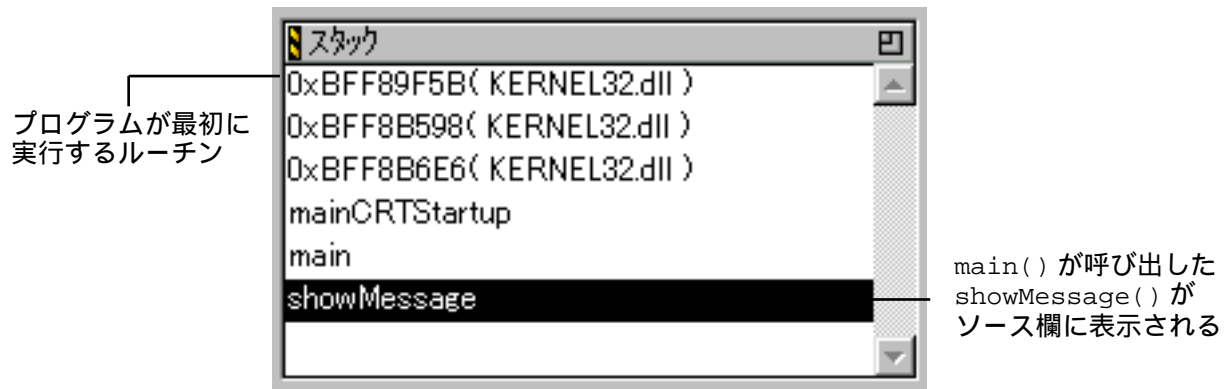
図 11.4 デバッガのツールバー



スタックロール欄

プログラムウィンドウのスタックロール欄は、現在コールチェーンで呼び出されているサブルーチンを表示します (図 11.5)。それぞれの関数は、呼び出し元の関数の下に表示されます。スタックロール欄で選択した関数のコードがソース欄に表示されます。

図 11.5 スタッククロール欄



変数欄


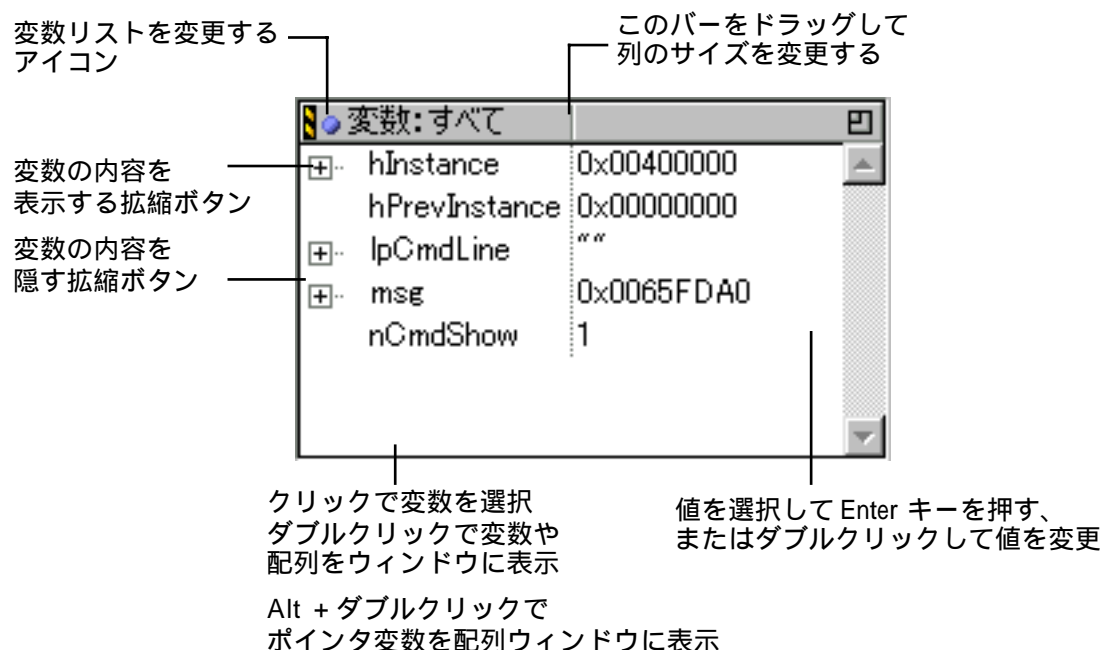
変数欄はプログラムで使われている局所変数をリスト表示します。変数リストは変数欄のアイコン () でコントロールできます (図 11.6)。このアイコンをクリックするとリストは [変数: すべて] と [変数: 自動] に切り替わります。表 11.2 でリストのタイプを解説します。

表 11.2 変数欄のリストタイプ

タイプ	表示される変数
変数: すべて	コード内のすべての変数
変数: 自動	カレントステートメント矢印 が指しているルーチンの局所変数

変数欄では、変数をアウトライン形式で表示します。アイテムの横にある拡張ボタンをクリックすると、変数の中のエントリが表示されたり、隠されたりします。

図 11.6 変数欄



例えば図 11.6 では、変数 `WP` の左横のボタンをクリックすると、メンバ変数が隠れます。再びそのボタンをクリックすると、`C` の構造体のメンバ変数が表示されます。Ctrl キー + クリックすると、複数の階層のポインタのデータを見ることができます。ハンドル構造体型へ拡張して、構造体メンバを見ることもできます。

詳細は「[グループの拡張と縮小](#)」(p54) を参照してください。

注意：汎用レジスタや浮動小数点レジスタの内容を見るには、汎用レジスタウィンドウと FPU レジスタウィンドウを使います。FPU を持たないビルドターゲットでは、FPU レジスタウィンドウを利用できません。詳細は「[レジスタウィンドウ](#)」(p304) を参照してください。

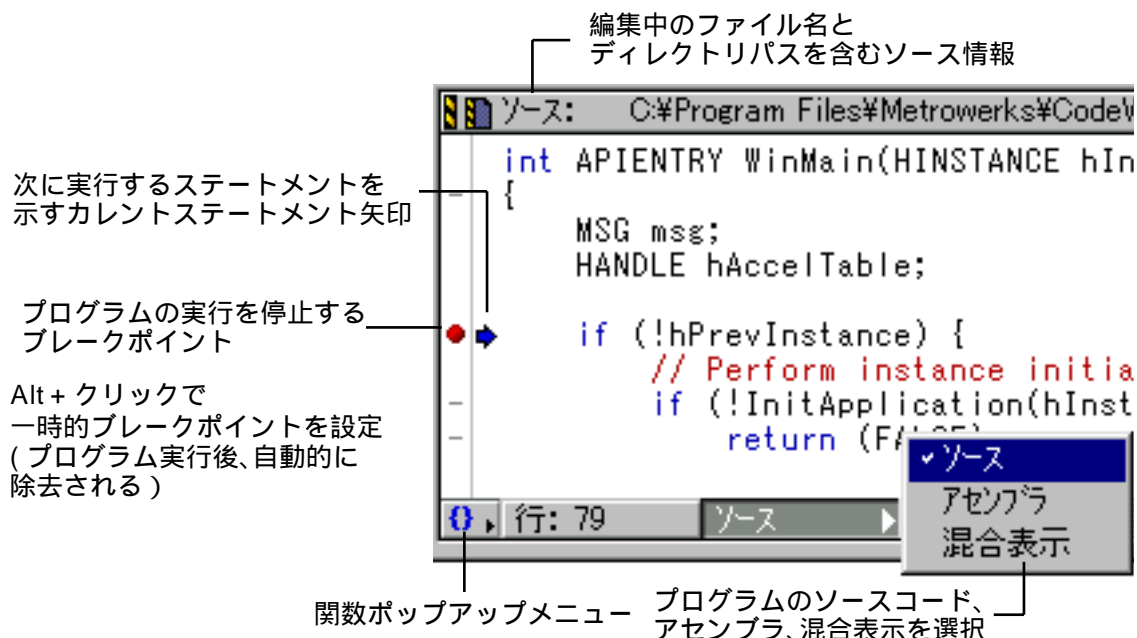
ソース欄

プログラムウィンドウのソース欄は、ソースコードを表示します。デバッガは、現在のビルドターゲットのソースファイルからコメントやスペースを含むソースコードをそのまま表示します。[フォント & タブ](#) 設定パネルで指定したフォントや色で、C/C++、Pascal、Java、およびインラインアセンブラコードを表示します (図 11.7)。

ソース欄の拡張ボタンをクリックすると (図 11.3 (p283))、ソース欄の表示 / 非表示が切り替わります。ソース欄を隠した場合、コードをステップ実行したときにプログラムウィンドウとエディタウィンドウの両方が開かれます。エディタウィンドウはソース欄と同じくソースコードとブレークポイントを表示します。

ソース欄では、プログラムのソースコードを 1 行ずつ実行できます。カレントステートメント矢印は常に、次に実行されるステートメントを指し、実行の過程を示します。コンピュータの CPU 内部のプログラムカウンタは、カレントステートメント矢印が指す命令のアドレスへセットされます。このため、カレントステートメント矢印はプログラムカウンタとも呼ばれます。

図 11.7 プログラムウィンドウのソース欄



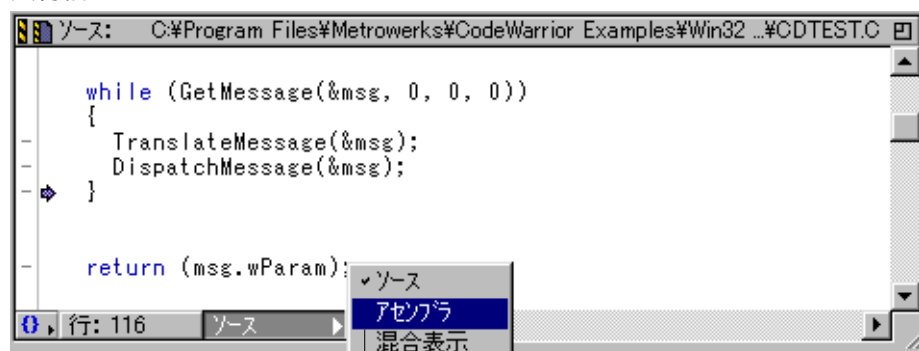
1 行の中で複数の関数が呼び出されているとき、各関数は別々に 1 ステップで実行されます。このような場合、すべての関数が実行されてからステートメント矢印が次の行に移動します。矢印は、プログラムカウンタがソースコードの途中にあるときには薄く表示されます。

ソースコードをアセンブラで見る

ソースコードをアセンブラで表示するには、プログラムウィンドウの下にあるソースポップアップメニューをクリックしてください。[アセンブラ] を選択すると、ソース欄にアセンブラコードが表示されます (図 11.8)。アセンブラコードを表示しているときも、ステップ実行や、ブレークポイントの設定ができます。

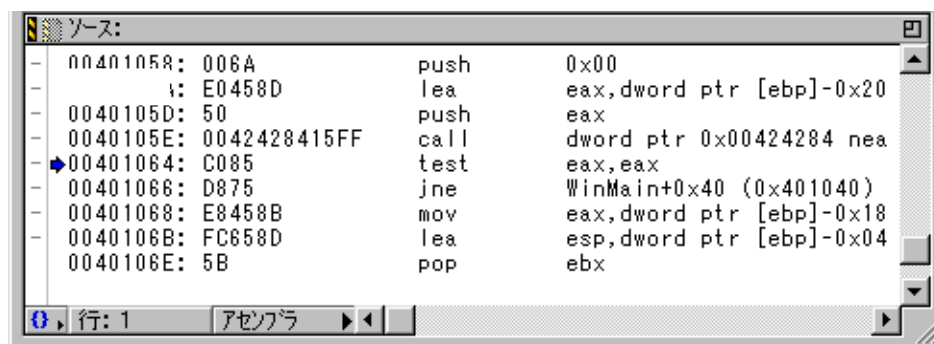
図 11.8 ソースとアセンブラの表示

実行前



アセンブラを見るには、**アセンブラ** を選択
アセンブラビューではブレークポイントと
ステップ実行が可能

実行後

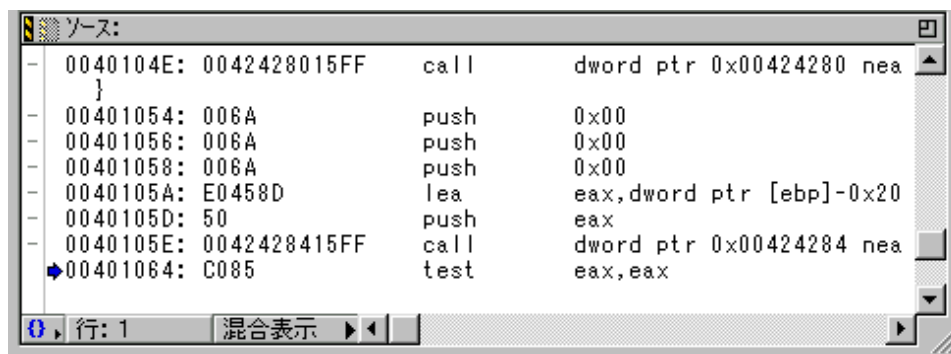


注意：汎用レジスタや浮動小数点レジスタの内容を見るには、汎用レジスタウィンドウと FPU レジスタウィンドウを使います。FPU を持たないビルドターゲットでは、FPU レジスタウィンドウを利用できません。詳細は「[レジスタウィンドウ](#)」(p304)を参照してください。

アセンブラとソースを混合表示

ソースコードとアセンブラコードを同時に表示するには、プログラムウィンドウ下部のソースポップアップメニューをクリックしてください。[混合表示]を選択すると現在のルーチンのソースコードとアセンブラコードが表示されます(図 11.9)。アセンブラ命令を生成したソースは、そのアセンブラの前に表示されます。コードを混合表示しているときもアセンブラコードにブレークポイントの設定や、ステップ実行ができます。しかしソースコードの行にはブレークポイントは設定できません(図 11.9)。

図 11.9 コードの混合表示



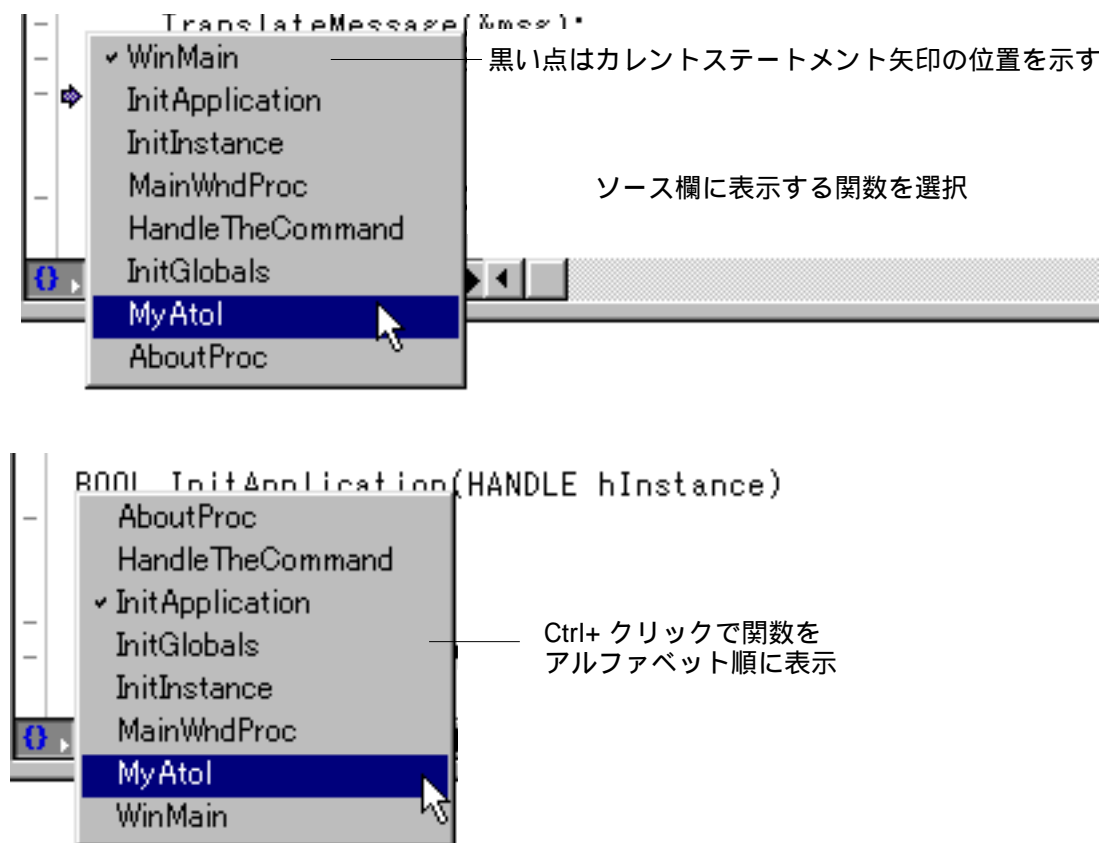
オブジェクトコードのシンボル情報がない場合、アセンブラを表示します。コードを混合表示しているとき、シンタックスカラーリングは使えません。テキストはすべて標準で表示されます。

関数ポップアップメニュー

ソース欄の左下にある関数ポップアップメニュー（[図 11.10](#)）には、ソース欄で選択したソースファイルで定義されている関数がリスト表示されます。関数ポップアップメニュー選択した関数がソース欄に表示されます。

関数ポップアップメニューを Alt+n + クリックすると関数名がアルファベット順に表示されます。

図 11.10 関数ポップアップメニュー



ブラウザウィンドウ

シンボルファイルをダブルクリックすると、デバッガはブラウザウィンドウを開きます([図 11.11](#))。

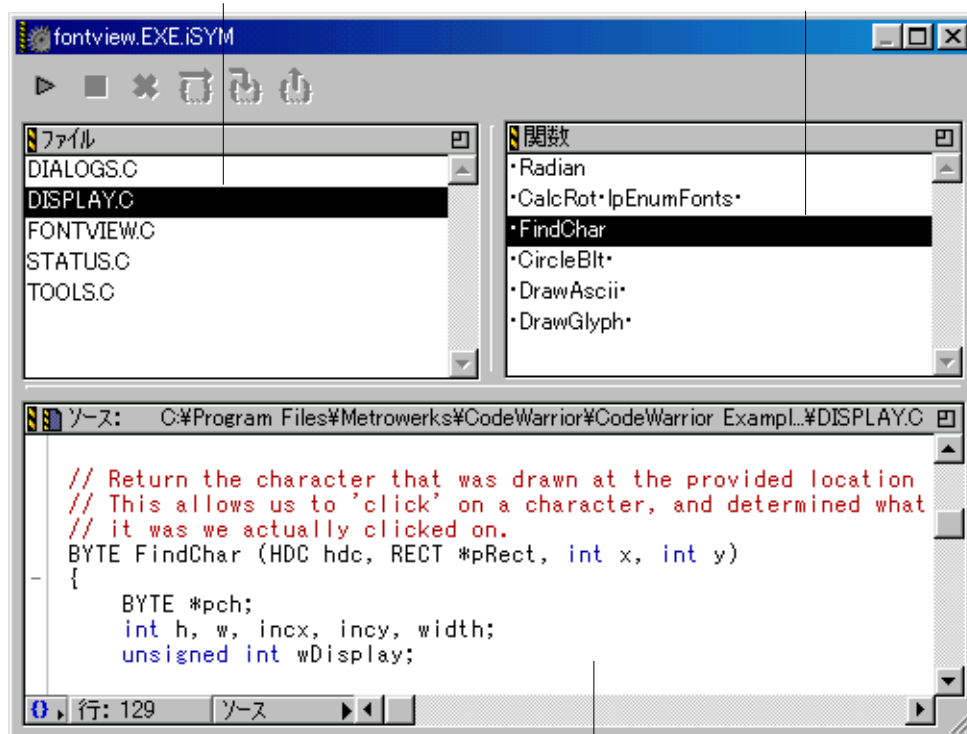
ブラウザウィンドウは、プログラムウィンドウに外観も機能もよく似ています。しかし、表示する情報が異なります。ブラウザウィンドウは、現在のビルドターゲットにある、シンボル情報を持つファイルをすべて表示することができます。一方プログラムウィンドウで表示できるのは、スタックロール欄で選択した関数を含むファイルだけです。また、ブラウザウィンドウでプログラムのすべての大域変数の値を編集できます。プログラムウィンドウでは、コールチェーンにある現在アクティブな関数が参照している大域変数しか変更できません。大域変数の表示については「[大域変数ウィンドウ](#)」(p297) を参照してください。

注意： ブラウザウィンドウとクラスブラウザを混同しないでください。この2つの外観はよく似ていますが、デバッガのブラウザウィンドウはソースコードのブラウザであり、シンボルファイルをダブルクリックしたときのみ開きます。

図 11.11 ブラウザウィンドウ

ファイル欄: 関数欄に表示する
ルーチンを含むファイルを選択

関数欄: ソース欄に表示する
ルーチンを選択



ソース欄: 関数欄で選択したルーチンのソースコードを表示

ヒント: ブラウザウィンドウの代わりに、大域変数ウィンドウを使うことができます。
[「大域変数ウィンドウ」\(p297\)](#) を参照してください。

ブラウザウィンドウには 3 つの表示欄があります。

[ファイル欄](#) (左上)

[関数欄](#) (右上)

[ソース欄](#) (下)

プログラムウィンドウと同様に、ブラウザウィンドウにもデバッガツールバー、関数ポップアップメニュー、行番号表示、ソースポップアップメニューがあります。またブラウザウィンドウの各欄の境界線をクリックやドラッグすることで、サイズを変更することもできます。Tab キーで各欄を移動できます。

ファイル欄や関数欄でアイテムをタイプahead選択ができます。アイテムの最初の数文字を入力してください。また矢印キーか Tab キーでアクティブな欄の中を移動できます。

同時に複数のシンボルファイルを開いてデバッグできます。つまり複数のプログラムを一度にデバッグできるのです。アプリケーションとアプリケーションのプラグインを同時に

デバッグするときなどに便利な機能です。シンボルファイルごとにブラウザウィンドウが開きます。

プログラムウィンドウの内容について詳しくは、「[プログラムウィンドウ](#)」(p282) も参照してください。

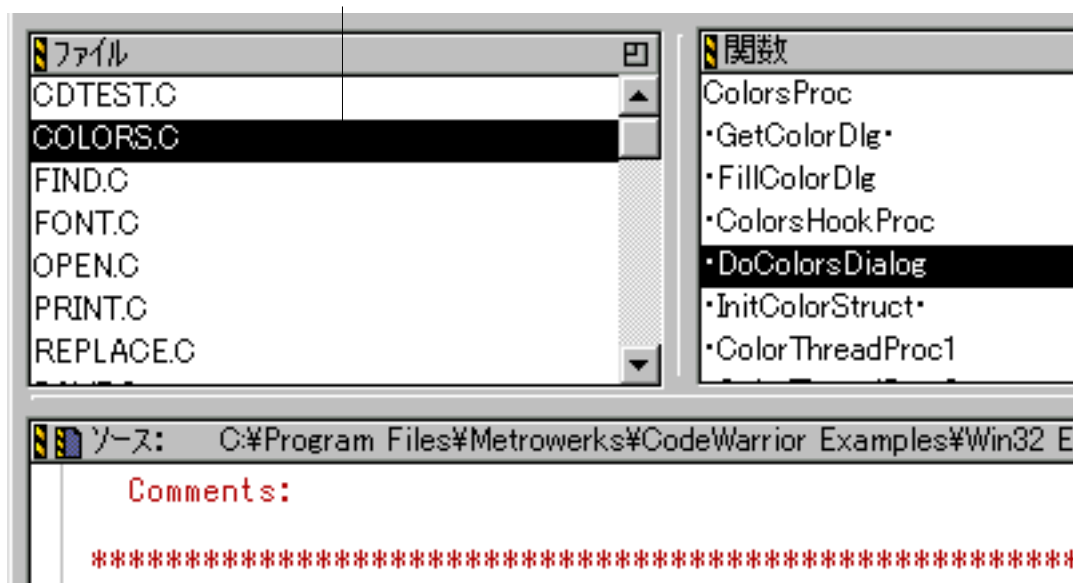
ファイル欄

ブラウザウィンドウのファイル欄には、現在デバッグしているビルドターゲットに関連するソースファイルのリストが表示されます（[図 11.12](#)）。ファイル欄で選択したファイルの関数が関数欄にリスト表示されます。ファイル欄は、関数欄、ソース欄とともに、プログラムにブレークポイントを設定するために使います。

「[大域変数ウィンドウ](#)」(p297) と「[ブレークポイント](#)」(p324) も参照してください。

図 11.12 ファイル欄（ブラウザウィンドウ）

ファイル欄:シンボル情報を持つソースファイルをリストする
強調表示されているファイルがソース欄に表示される



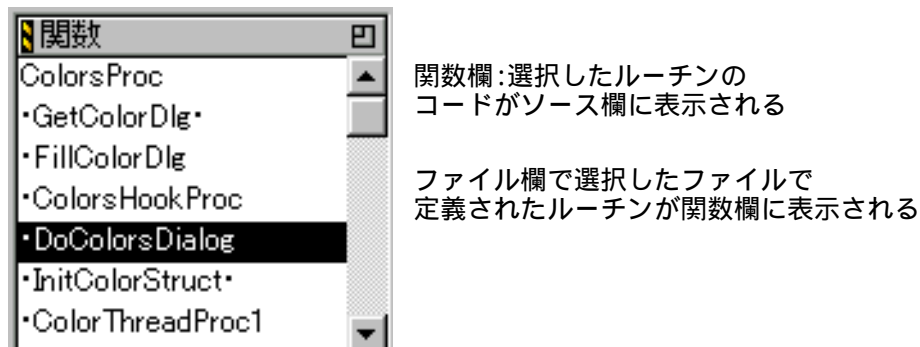
関数欄

ブラウザウィンドウのファイル欄でソースファイルを選択すると、関数欄にはそのファイルで定義されているすべての関数がリスト表示されます（[図 11.13](#)）。関数名をクリックすると、ウィンドウの下部のソース欄にその関数が表示されます。

注意： C++ または Object Pascal でコードを書いている場合、IDE 設定の表示設定パネルの「ブラウザ内の関数をメソッド名で並べ替えて表示」オプションをオンにすると（「[表示設定](#)」(p201)）関数欄の関数名をアルファベット順に並べて表示します。

className::methodName のようなメンバ関数 (メソッド) についても className ではなく、methodName 名でソートします。

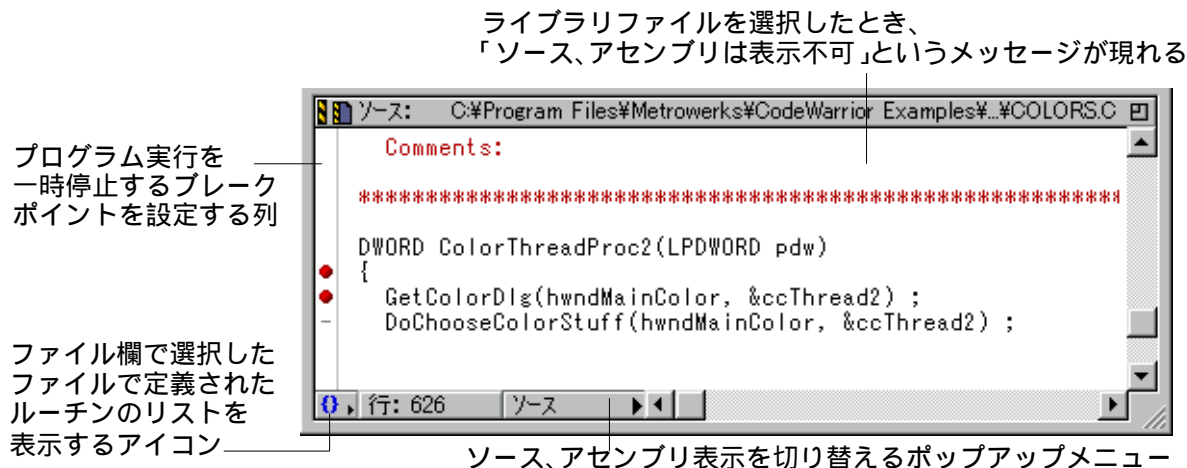
図 11.13 関数欄 (ブラウザウィンドウ)



ソース欄

ブラウザウィンドウのソース欄には、[ファイル欄](#)で選択したファイルのソースコードが表示されます (図 11.14)。このソース欄で、ファイル欄あるソースファイルに対してブレークポイントを設定できます。ブレークポイントについては「[ブレークポイント](#)」(p324) を参照してください。

図 11.14 ソース欄 (ブラウザウィンドウ)



しかし、ブラウザウィンドウのソース欄には現在実行中のステートメントは表示されないことに注意してください。実行中のステートメントまたは局所変数を表示するには、プログラムウィンドウを使います。「[プログラムウィンドウ](#)」(p282) を参照してください。

ソース欄のコードは、DE 設定パネルで指定したフォントとカラーで表示されます。「[エディタ設定](#)」(p194)、「[表示設定](#)」(p201) を参照してください。

ファイル欄で選択した項目がソースコードを含まない場合、ソース欄に「ソーステキストは表示できません」という内容のメッセージが表示されます。

プログラムウィンドウと同様に、ブラウザウィンドウにも一番下にソースポップアップメニューがあります（「[ソースコードをアセンブラで見る](#)」(p286)）。[アセンブラ]を選択すると、ソース欄の内容がアセンブラコードで表示されます（[図 11.8 \(p287\)](#)）。アセンブラコードにもソースコードと同様にブレークポイントを設定できます。[混合表示]を選択すると、ソースコードとアセンブラコードの両方が表示されます（[図 11.9 \(p288\)](#)）。

関数ポップアップメニュー

ソース欄の左下にある関数ポップアップメニューは、ファイル欄で選択したソースファイルで定義されている関数をリスト表示します。関数欄で関数をクリックしたときと同じように、関数ポップアップメニューで関数を選択すると、それがソース欄に表示されます。

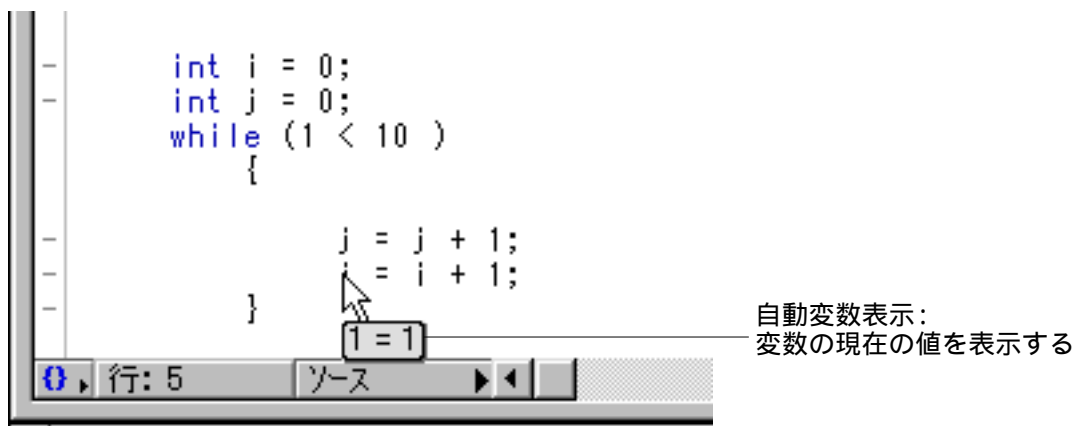
関数ポップアップメニューを Alt + クリックすると、関数をアルファベット順にソートして表示します。（[図 11.10 \(p289\)](#)）

注意： ソース欄にソースコードが表示されていない場合、関数ポップアップメニューは何も表示しません。

シンボルのヒント

ソースコードをデバッグするとき、デバッガはプログラムの変数について有益な情報を表示します。この情報は、カーソルがソース欄の特定の変数を通過したときに自動的に表示されます。デバッグ中にプログラムウィンドウのソース欄でコードを検証していると仮定します。カーソルの位置が変数 `i` の上に来たとき（[図 11.15](#)）、デバッガはその変数の現在の値を表示します。

図 11.15 シンボルのヒント



注意： シンボルのヒントはデバッガがアクティブなときに使用できます。デバッガがアクティブでないときは変数の情報は表示されません。「[デバッグの準備](#)」(p278) を参照してください。

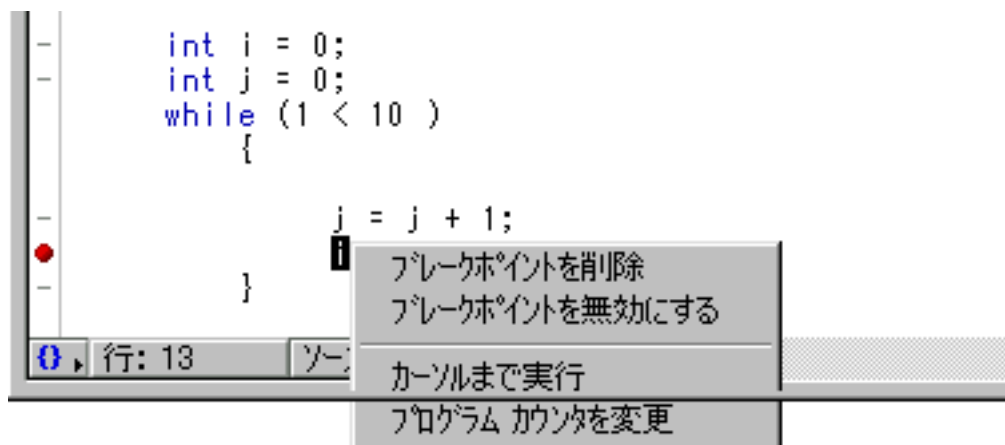
デバッガのコンテキストメニュー

デバッガのコマンドを選択するための便利なコンテキストメニュー([図 11.16](#))があります。コンテキストメニューは以下の方法で表示できます。

アイテムを右クリックする

デバッガは選択されているアイテムから、コンテキストメニューに表示するコマンドを判断します。例えば変数を選択している場合、コンテキストメニューには変数の内容を表示するためのコマンドが含まれます。

図 11.16 デバッガのコンテキストメニュー



ヒント： デバッガのコンテキストメニューは非常に便利です。例えば、[変数欄](#)と[汎用レジスタウィンドウ](#)でコンテキストメニューを使うと、異なる形式で変数を見ることができます。[ソース欄](#)でコンテキストメニューを使うと、ブレークポイントを操作してプログラムカウンタを移動することができます。コンテキストメニューのさまざまな機能を試してみてください。

プロセスウィンドウ

プロセスウィンドウ([図 11.17](#))には現在実行中のプロセスが、隠しプロセスも含めて表示されます。プロセスウィンドウには、選択したプロセスのタスク也表示されます。プロセスウィンドウを開くには、ウィンドウメニューの [プロセスウィンドウ] を選択します。

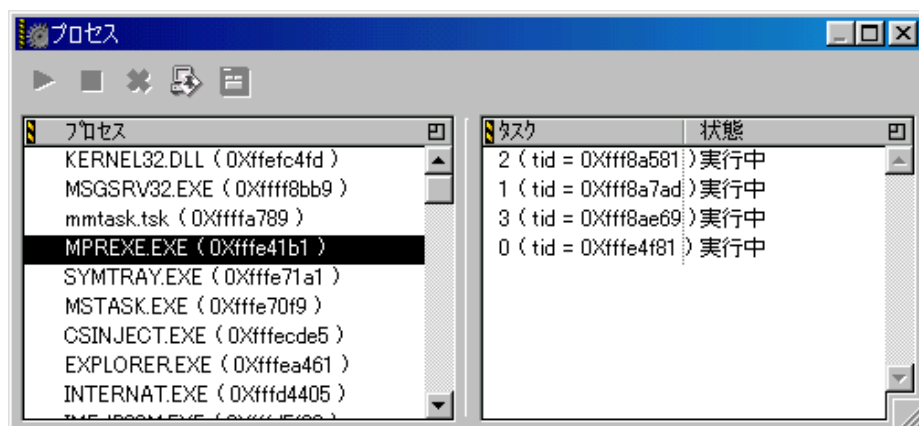
プロセスウィンドウには、次の 2 つの欄とツールバーがあります。

プロセスウィンドウのツールバー：デバッガのコントロール下でプロセスやタスクを実行、終了、停止します。

プロセス欄：現在実行中の全プロセス名を表示します。

タスク欄：選択したプロセスの全タスクを表示します。

図 11.17 プロセスウィンドウ



プロセスウィンドウのツールバー

プロセスウィンドウのツールバー（図 11.18）には、デバッガの制御下にあるプロセスを [実行] [停止] [終了] させるボタンがあります。これらはそれ以外の、現在実行中のプロセスやタスクには影響しません。[プロセスアタッチ] ボタンは選択したプロセスをデバッグにアタッチし、[プログラムウィンドウ] ボタンはプログラムウィンドウを開きます。

図 11.18 プロセスウィンドウのツールバー

[プロセスアタッチ] ボタン



[プログラムウィンドウ] ボタン

[プロセスアタッチ] ボタンは選択したプロセスをデバッグします。このボタンは、デバッガがターゲットにできないプロセスをデバッグするときに便利です。例えばダイナミックリンクライブラリや共有ライブラリを使うプログラムをデバッグするとき、[プロセスアタッチ] ボタンを使うとライブラリをデバッガでコントロールできます。プロセス欄にはアタッチされたプロセスの横にチェックマークが現れます。

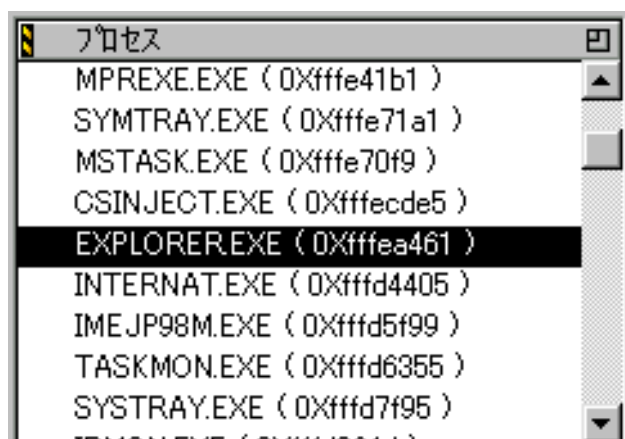
[プログラムウィンドウ] ボタンは、選択されているプロセスまたはタスクのプログラムウィンドウを表示します。選択されたプロセスのプログラムウィンドウが前に出ます。タ

スクが選択されると、そのタスクのプログラムウィンドウが前に出ます。同時に複数のプログラムウィンドウを表示できます。

プロセス欄

プロセス欄はアクティブなプロセスをすべて表示します。デバッガのコントロール下にあるプロセスにはチェックマークが付きます。チェックマーク列をクリックすると、デバッガがそのプロセスをコントロールします。プロセスの名前をダブルクリックするとそのプロセスがアクティブになります。

図 11.19 プロセス欄



タスク欄

タスク欄は指定したプロセスに所属するアクティブなタスクをすべて表示します。デバッガのコントロール下にあるプログラムのタスクのみが表示されます。タスク名をダブルクリックすると、そのタスクのコードを含むプログラムウィンドウがアクティブになります。タスクを選択して [プログラムウィンドウ] ボタンをクリックしても同じです ([図 11.18](#))。

図 11.20 タスク欄



タスク欄には 2 つの列があります。最初の列はタスク ID を示し、次の列はタスク状態を示します。タスク状態は実行、停止、クラッシュのいずれかです。

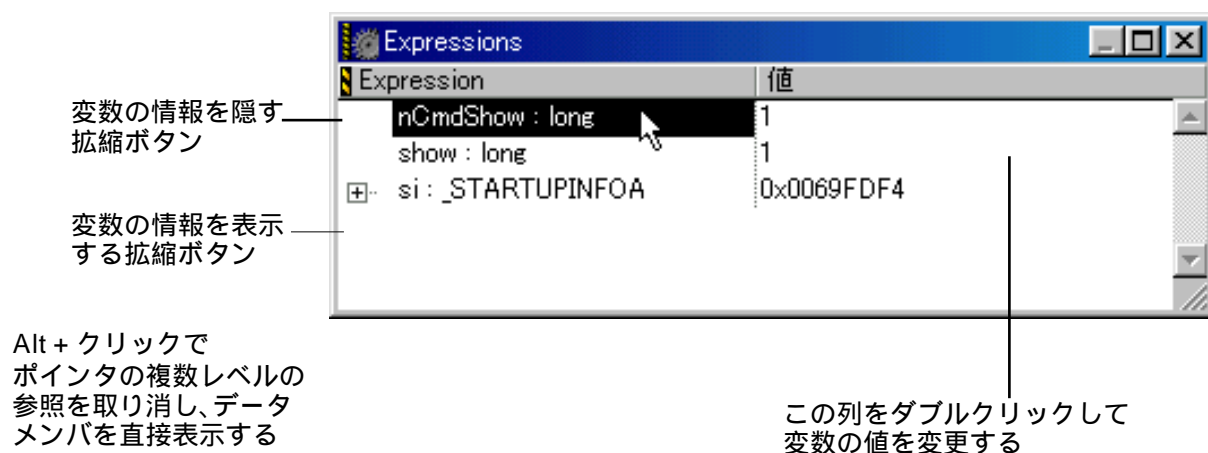
Expression ウィンドウ

「Expressions」ウィンドウ ([図 11.21](#)) は、頻繁に使用する局所変数や大域変数、構造体のメンバ変数や配列の要素を置いておくために使います。

「Expressions」ウィンドウを開くには、ウィンドウメニューの [Expression ウィンドウ] コマンドを選択します。

選択した項目を「Expressions」ウィンドウへ追加するには、データメニューの [Expression へコピー] を使います。デバッガのコンテキストメニューからも同じコマンドを選択できます ([「デバッガのコンテキストメニュー」\(p294\)](#))。マウスを使って他の変数欄やウィンドウから「Expressions」ウィンドウへ、項目をドラッグすることもできます。また「Expressions」ウィンドウのリスト内でドラッグして項目の順序を変えることもできます。

図 11.21 「Expressions」ウィンドウ



「Expressions」ウィンドウから項目を削除するには、項目を選択してから、Backspace キーを押すか、編集メニューの [削除] を選択してください。

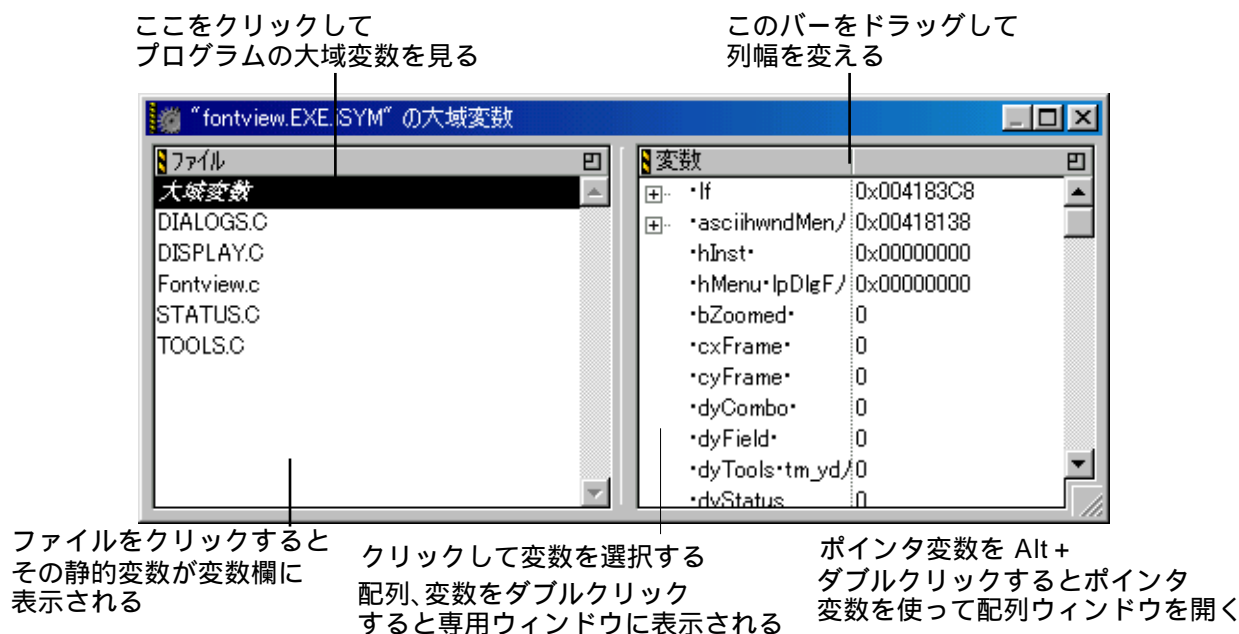
普通の変数ウィンドウに表示される局所変数と違って、「Expressions」ウィンドウにある局所変数は、定義されたルーチンが戻るときに削除されません。

「Expressions」ウィンドウについては [「Expression ウィンドウを使う」\(p339\)](#) を参照してください。

大域変数ウィンドウ

大域変数ウィンドウ ([図 11.22](#)) はプログラムの大域変数を表示します。ファイル欄でファイルを選択することにより静的変数を表示することもできます。静的変数は変数欄に表示されます。

図 11.22 大域変数ウィンドウ



大域変数を専用ウィンドウに表示

大域変数を専用のウィンドウに表示するには、変数欄で大域変数の名前をダブルクリックします。新しい変数ウィンドウが開き、大域変数の名前と値を表示します。変数欄で変数を選択して、デバッガのデータメニューの[変数を表示]、またはコンテキストメニュー([図 11.16 \(p294\)](#))の[変数を表示]を選択することでも変数ウィンドウが開きます。大域変数は専用のウィンドウに表示され、変数欄と同様に表示と編集が可能です。大域変数を「Expressions」ウィンドウに追加することもできます。

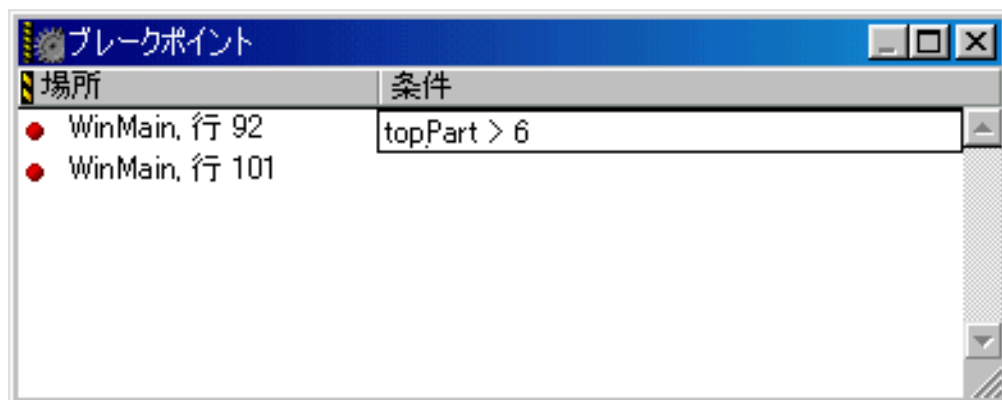
大域変数を表示するウィンドウはデバッグの間は開いたままです。変数ウィンドウを閉じるには、それを最前面にしてからファイルメニューの[すべての変数ウィンドウを閉じる]を選択してください。

詳細は「[Expression ウィンドウ](#)」(p297)、「[変数ウィンドウ](#)」(p307)、「[配列ウィンドウ](#)」(p307)を参照してください。

ブレークポイントウィンドウ

ブレークポイントウィンドウ([図 11.23](#))は現在のビルドターゲットに含まれるブレークポイントの一覧を、ソースファイルと行番号で表示します。ウィンドウメニューの[ブレークポイントウィンドウ]を選択するとブレークポイントウィンドウが開きます。

図 11.23 ブレークポイントウィンドウ



リストの左側にブレークポイントマーカがあります。マーカをクリックするとブレークポイントの状態が変わります。赤い点はブレークポイントが有効、白い丸はブレークポイントが無効を示します。どちらの場合も IDE はプログラム内のブレークポイントの位置を記憶します。ブレークポイントウィンドウのブレークポイントマーカを Alt + クリックするとすべてのブレークポイントが有効または無効になります。ブレークポイント列をダブルクリックするとそれに対応するソースコードがエディタウィンドウに表示されます。

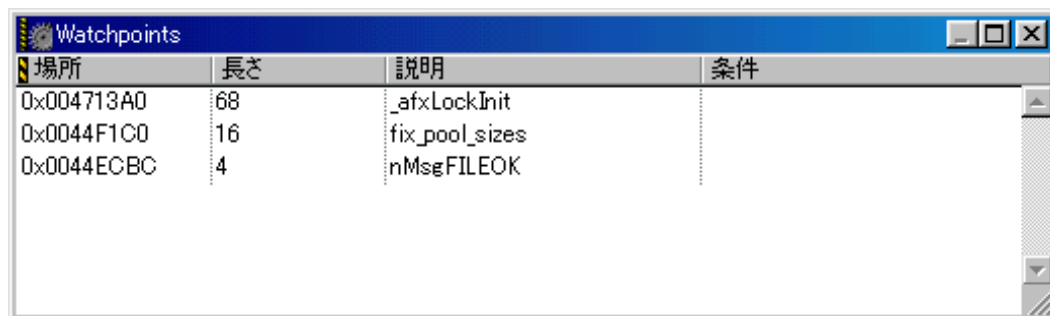
ブレークポイントに条件を付けることができます。条件が TRUE でブレークポイントが有効の場合、そのブレークポイントでプログラムの実行が停止します。ブレークポイントが無効、または条件が FALSE の場合、ブレークポイントは働かず、実行は停止しません。

詳細は「[ブレークポイント](#)」(p324)、「[ブレークポイントを表示](#)」(p470)、「[ブレークポイントを隠す](#)」(p470)、「[条件ブレークポイント](#)」(p327)を参照してください。

Watchpoint ウィンドウ

Watchpoint ウィンドウ (図 11.24) はプロジェクトで設定されたすべての Watchpoint をメモリアドレスで表示します。Watchpoint ウィンドウを開くには、ウィンドウメニューから [Watchpoint ウィンドウ] を選択します。

図 11.24 Watchpoint ウィンドウ



Watchpoint を消去するには、それを選択してから次のいずれかを行います。

デバッグメニューの [Watchpoint の削除] を選択する

編集メニューの [削除] を選択する

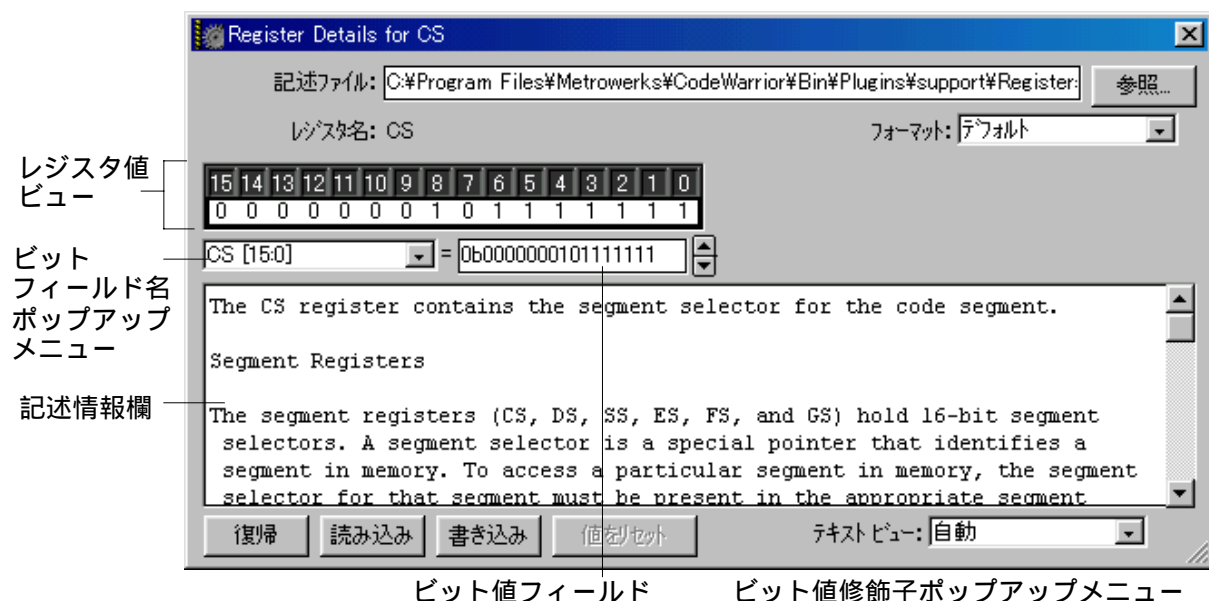
Backspace キーを押す

詳細は「[Watchpoint](#)」(p330) を参照してください。

レジスタ詳細ウィンドウ

レジスタ詳細ウィンドウ ([図 11.25](#)) は 32 ビットレジスタにある個別のレジスタビットについて詳細な情報を表示します。システムレジスタとメモリマップレジスタの両方の情報を表示します。レジスタ詳細ウィンドウを開くには、ウィンドウメニューの [レジスタ詳細ウィンドウ] を選択します。

図 11.25 レジスタ詳細ウィンドウ



レジスタ詳細ウィンドウには、レジスタ、ビットフィールドとその値を表示するフィールドがあります。これらのフィールドは Registers フォルダにある XML (Extensible Markup Language) ファイルから派生したものです。Registers フォルダは Plugins フォルダの中の Support フォルダにあります。

注意： レジスタ詳細ウィンドウで使うための XML ファイルの作成については『Register Details Window XML File Format』という CodeWarrior ホワイトペーパーを参照してください。

レジスタ詳細ウィンドウには以下のオプションがあります。

[記述ファイル](#)

[参照](#)[アドレス](#)[フォーマット](#)[レジスタ値](#)[ビットフィールド名](#)[ビット値](#)[ビット値修飾子](#)[記述情報](#)[復帰](#)[読み込み](#)[書き込み](#)[値をリセット](#)[テキストビュー](#)

記述ファイル

特定のレジスタの情報を見るには、[記述ファイル] フィールドへそのレジスタの名前を入力して Enter キーを押します。または、ハードディスク上にあるレジスタの記述ファイルへの完全なパスを入力して Enter キーを押します。デバッガは大小文字を無視します。

レジスタの名前かパスを入力して Enter キーを押した後、デバッガは `Registers` フォルダとプロジェクトアクセスパスを検索して、入力された名前と一致する XML ファイルを探します。デバッガが一致するファイルを見つけると、それに対応する記述ファイルがレジスタ詳細ウィンドウに表示されます。見つけれなかったとき、エラーメッセージが表示されます。

例えば、Opcode レジスタの中身を見るには、[記述ファイル] フィールドへ「`Opcode`」と入力して Enter キーを押します。または `opcode.xml` ファイルへの完全なパスを[記述ファイル] フィールドへ入力して Enter キーを押します。どちらの場合も、デバッガは `opcode.xml` ファイルを探し、Opcode レジスタの情報を表示します。

レジスタ情報を表示した後、デバッガは現在のレジスタ値を読み取ってレジスタ詳細ウィンドウに表示します。読み取りに失敗するとエラーメッセージが表示されます。

参照

[参照] ボタンをクリックするとファイルを開くダイアログが現れます。ここで XML レジスタ記述ファイルを指定します。記述ファイルがレジスタ詳細ウィンドウに表示されます。

アドレス

[レジスタ値](#)に表示したいレジスタの開始アドレスを [アドレス] フィールドへ入力します。無効、または不正確なアドレスを入力するとエラーメッセージが表示されます。

フォーマット

[フォーマット] ポップアップメニュー から、レジスタ詳細ウィンドウに表示するデータフォーマットを選択します。[デフォルト] [バイナリ] [文字] [10 進数] [16 進数] [符号なし 10 進数] の選択肢があります。[デフォルト] を選択すると、デバッガが自動的にレジスタのデータに適したフォーマットを選びます。

レジスタ値

[レジスタ値] ビューは 32 ビットレジスタデータの現在の中身を、[アドレス] フィールドで指定されたアドレスから表示します。データは [フォーマット] ポップアップメニューで選択したフォーマットで表示されます。

[レジスタ値] ビューは 32 ビットのデータを適切なレジスタビットフィールドへまとめます。あるビットをクリックすると、そのビットフィールドが選択されます。名前やアクセス権などのビットフィールドの詳細は、レジスタ詳細ウィンドウの [記述情報] 欄に表示されます。

ビットフィールド名

[ビットフィールド名] ポップアップメニューで [レジスタ値] ビューにあるビットフィールドを選択します。ポップアップメニューはビットフィールドの名前と対応するビット位置を表示します。選択したビットフィールドをに示します (図 11.25 (p300))。

別のビットフィールドを選択するには、[ビットフィールド名] ポップアップメニューで名前を選択します。または [レジスタ値] ビューで希望のビットフィールドをクリックして選択します。ビットフィールドの選択を解除するには、[ビットフィールド名] ポップアップメニューから [なし] を選択します。

ビット値

[ビット値] フィールドは、[レジスタ値] ビューで選択されたビットフィールドの現在の値を表示します。[フォーマット] ポップアップメニューでフォーマットを決定します。

現在の値を変更するには、[ビット値] フィールドへ新しい値を入力して Enter キーを押します。[レジスタ値] ビューが更新されます。



[ビット値] フィールド横のボタンで現在の値をインクリメント、またはデクリメントすることもできます。

注意： [ビット値] フィールドのデータを変更は、[レジスタ値] ビューに表示されているデータにのみ反映されます。レジスタそのものの値を変更するには、[書き込み] ボタンをクリックします。

ビット値修飾子

[ビット値修飾子] ポップアップメニューには、[[レジスタ値](#)] ビューで選択したビットフィールドを変更するためのオプションがあります。このポップアップメニューにはビット値の意味の解説が表示されることもあります。

注意： [ビット値修飾子] ポップアップメニューで変更したデータは [[レジスタ値](#)] ビューに表示されているデータにのみ反映されます。レジスタそのものの値を変更するには、[[書き込み](#)] ボタンをクリックします。

記述情報

[記述情報] 欄は、レジスタ詳細ウィンドウにある項目についてのレジスタ記述、ビットフィールド記述、レジスタ詳細などの追加情報を表示します。[[テキストビュー](#)] ポップアップメニューで別の項目を選択すると、[記述情報] 欄が更新されます。

復帰

変更した値を元に戻すには、[復帰] ボタンをクリックします。

警告！ 値を変更した後で [[書き込み](#)] ボタンをクリックすると、祖の値を元に戻すことはできません。

読み込み

[[レジスタ値](#)] ビューにあるレジスタの現在のビット値を表示するには、[読み込み] ボタンをクリックします。

書き込み

通常、[[レジスタ値](#)] ビューの値を変更することは、実レジスタにある対応する値を変更することではありません。実レジスタへ変更を適用するには、[書き込み] ボタンをクリックします。これによって [[レジスタ値](#)] ビューの値が実レジスタへ書き込まれます。

警告！ [書き込み] ボタンを押した後は、[[復帰](#)] ボタンを押してもレジスタを元の値に戻すことはできません。

値をリセット

デフォルトのビット値を持つレジスタもあります。これらのデフォルト値をリセットするには、[値をリセット] ボタンをクリックします。選択したレジスタにデフォルト値がない場合、[値をリセット] ボタンは薄く表示されます。

テキストビュー

[テキストビュー] ポップアップメニューで [[記述情報](#)] 欄に表示される情報を変更します。以下の選択肢があります。

自動：自動的に、選択したレジスタやビットフィールドに最も適した情報を表示します。

レジスタ記述：レジスタ全体の情報を表示します。一般的にレジスタの名前とその中身の意味を表示します。

ビットフィールド記述： [[レジスタ値](#)] ビューで選択したビットフィールドに関する情報を表示します。一般的にビットフィールドの名前とアクセス権を表示します。

レジスタ詳細：現在のレジスタに関する詳細情報を表示します。一般的にレジスタの名前と値、ビット値の解説を表示します。

レジスタウィンドウ

レジスタウィンドウはレジスタの内容を表示し、それを編集することができます。ビルドターゲットによっていろいろな種類のレジスタウィンドウが利用できます。以下のレジスタに付いて説明します。

[汎用レジスタウィンドウ](#)

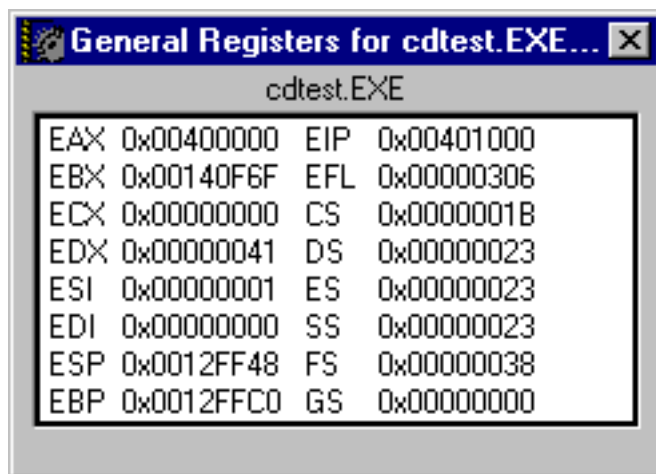
[FPU レジスタウィンドウ](#)

汎用レジスタウィンドウ

汎用レジスタウィンドウ ([図 11.26](#)) には CPU レジスタが表示され、このレジスタの内容を編集できます。汎用レジスタを開くには、ウィンドウメニューの [レジスタウィンドウ] のサブメニューにある [General Registers] を選択します。

注意： ビルドターゲットによって汎用レジスタウィンドウの外観、およびレジスタウィンドウサブメニューが変わります。サブメニューがない場合、ウィンドウメニューの [レジスタウィンドウ] を選択します ([図 11.26](#)) 。

図 11.26 汎用レジスタウィンドウ



レジスタの値を変更するには、
その値をダブルクリックする

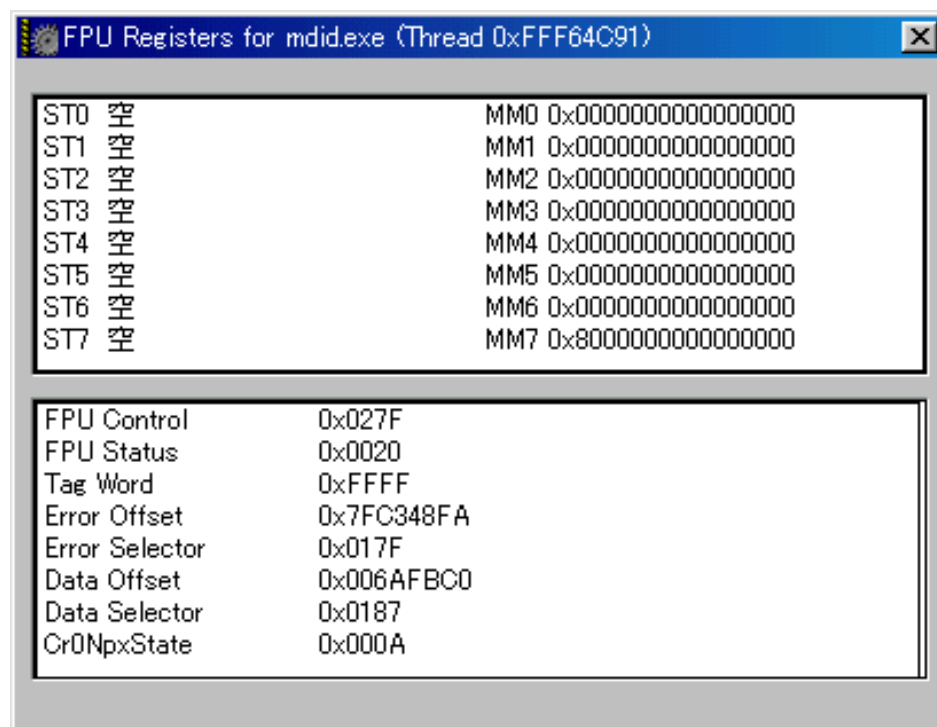
レジスタの値を変更するには、値をダブルクリックするか、またはレジスタを選択してから Enter キーを押します。この後、新しい値を入力します。

FPU レジスタウィンドウ

FPU レジスタウィンドウ (図 11.17) には FPU レジスタが表示され、このレジスタの内容を編集できます。FPU レジスタウィンドウを開くには、ウィンドウメニューの [レジスタウィンドウ] のサブメニューにある [FPU Registers] を選択します。

注意： ビルドターゲットによって FPU レジスタウィンドウの外観、およびレジスタウィンドウサブメニューが変わります。サブメニューがない場合、ウィンドウメニューの [FPU レジスタウィンドウ] を選択します (図 11.17)。

図 11.27 FPU レジスタウィンドウ



レジスタの値を変更するには、値をダブルクリックするか、またはレジスタを選択してから Enter キーを押します。その後、新しい値を入力します。

警告！ レジスタの値を変えることは非常に危険です。データやメモリを壊して、クラッシュを引き起こすこともあります。

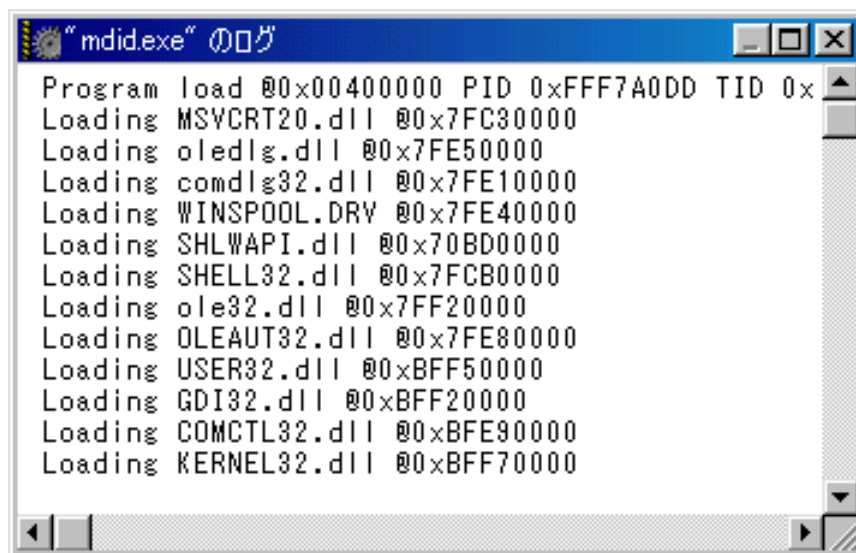
Log ウィンドウ

Log ウィンドウ (図 11.28) は、プログラムがシステム DLL の呼び出し、または新しいタスクの開始をしたときにメッセージを表示します。

Log ウィンドウを使うためにはログ情報が必要です。デバッグ設定パネルの[システムメッセージの記録する]オプションをオンにします。[「デバッグ設定」\(p251\)](#)を参照してください。

注意： ログ情報に DLL のロード / アンロードに関するメッセージ、およびデバッグ `printf()` メッセージも含まれます。

図 11.28 Log ウィンドウ



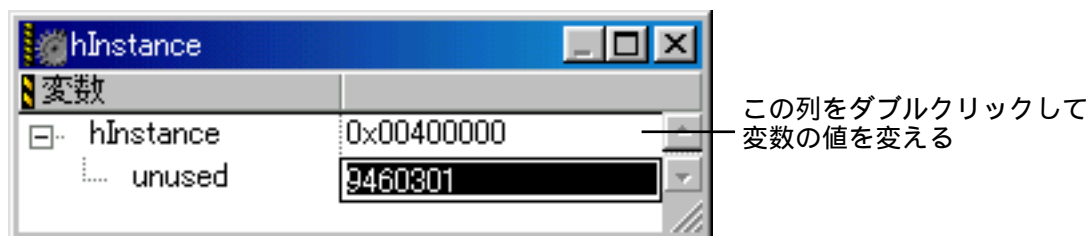
編集メニューの [コピー] コマンドで Log ウィンドウのテキストをコピーできます。ファイルメニューの [名前をつけて保存] コマンドでも Log ウィンドウの内容をテキストファイルに保存できます。

テキストファイルとして保存する場合、ファイル名に適切な拡張子 (log.txt など) を付けてください。

変数ウィンドウ

変数ウィンドウ (図 11.29) には、1 つの変数が表示されます。このウィンドウで変数の値を編集できます。局所変数を表示している変数ウィンドウは、変数を定義している関数が呼び出し元に戻るときに閉じます。

図 11.29 変数ウィンドウ



配列ウィンドウ

配列ウィンドウ (図 11.30) は、隣接しているメモリのブロックを、要素の配列として表示します。また、その要素の内容を編集することもできます。配列ウィンドウを開くには、変数欄 (局所または大域) で配列変数を選択して、データメニューの [配列を表示] を選んでください。

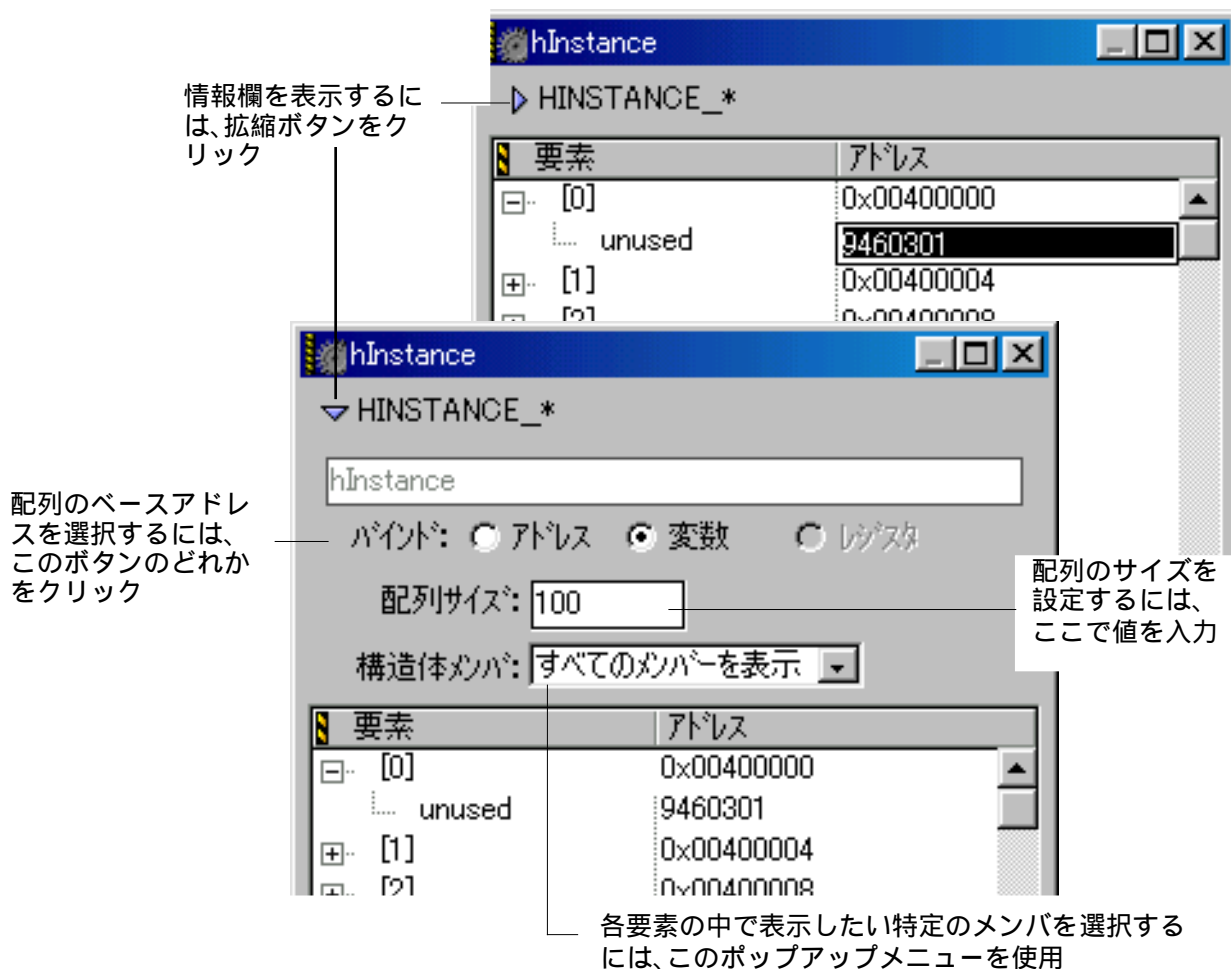
データメニューの [メモリの表示を変更] でも配列ウィンドウが開きます。このコマンドはデータ型を選択するためのダイアログを開き、そこで選択された型の配列としてメモリを解釈して配列ウィンドウを開きます。[「メモリの表示を変更」\(p472\)](#) を参照してください。

配列ウィンドウのタイトルバーは、配列の基底アドレスを示します。配列の基底アドレスは、アドレス、変数、またはレジスタに結び付けることができます。変数欄またはレジスタ欄からレジスタ名または変数名を配列ウィンドウにドラッグすることで、配列のアドレスを設定します。局所変数に結び付けられた配列は、変数を定義している関数が呼び出し元に戻るときに閉じられます。

情報欄には、配列の要素のデータ型と配列の基底アドレスが表示されます。情報欄の矢印をクリックすると、配列の詳細が表示されます。拡張された情報欄には、配列の基底アドレス、サイズ、および配列要素が構造型の一部である場合は、そのメンバが表示されます。

配列の内容は、要素 0 から順にリストされます。配列の要素が構造体の形をとっている場合、各配列の要素の左側に拡張ボタンが表示されます。このボタンをクリックして、配列要素のメンバ変数を表示したり、隠したりすることができます。

図 11.30 配列ウィンドウ

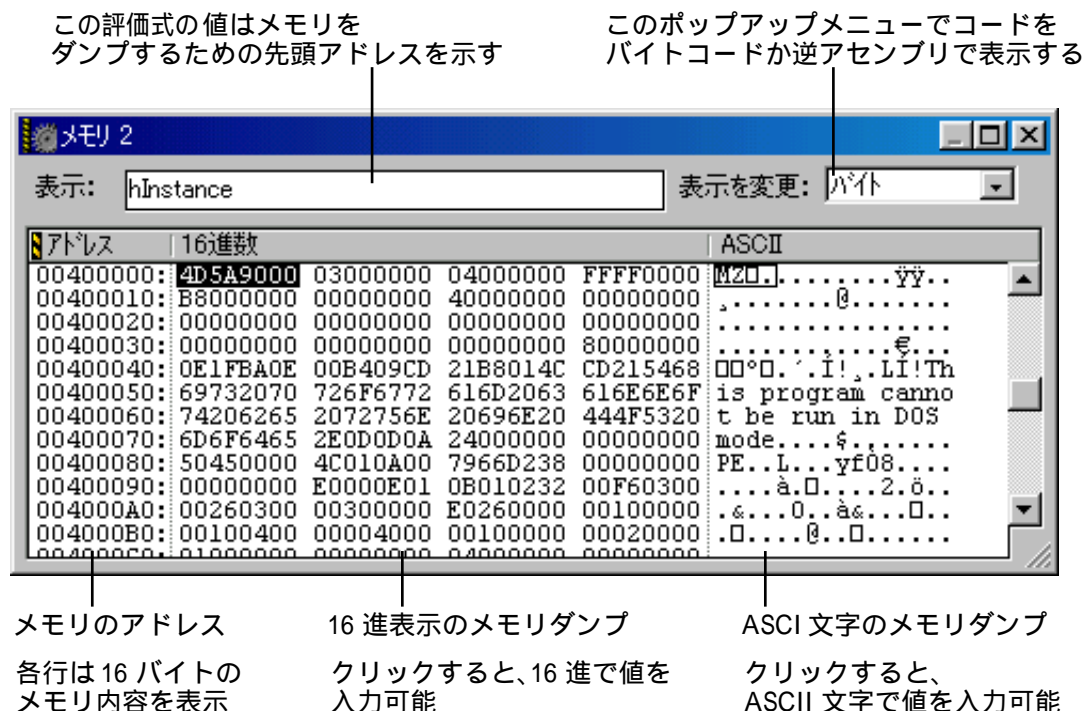


メモリウィンドウ

メモリウィンドウ (図 11.31) は、メモリの内容を 16 進数と ASCII 文字で表示します。メモリウィンドウを開くには、プログラムウィンドウ、ブラウザウィンドウ、または「Expressions」ウィンドウで、変数、関数名、または表示したい基底アドレスを表す評価式を選択してからデータメニューの [メモリを表示] を選択してください。

注意： メモリを特定のデータ型の配列として表示する配列ウィンドウ (「[配列ウィンドウ](#)」(p307)) を開くには、[メモリの表示を変更] を選択します。

図 11.31 メモリウィンドウ



ソースの基底アドレス（変数、関数、評価式、0xCFAF64C のようなローアドレスになり得ます）はウィンドウの一番上に表示されます。IDE が基底アドレスにアクセスできないとき、メモリウィンドウは空白になります。

基底アドレスを変えるには [表示] フィールドに新しい評価式を入力するか、ドラッグします。評価式がメモリのオブジェクトに lvalue（左辺式）を与えないとき、評価式の値が基底アドレスとなります。例えば、メモリウィンドウの評価式

PlayerRecord

は PlayerRecord のアドレスから始まるメモリを表示します。

評価式の結果が lvalue のとき、そのオブジェクトのアドレスが基底アドレスとして使われます。例えば、以下の評価式

*myArrayPtr

は myArrayPtr が指しているオブジェクトのアドレスから始まるメモリを表示します。

メモリウィンドウで、メモリ内の値を変更することもできます。表示されているデータ内で変更したいメモリの開始点をクリックしてから入力します。16 進表示のデータを選択したときは、16 進数で入力します。ASCII 表示のデータを選択したときは、英数字で入力します。入力するとき、Backspace、Tab、Enter などのキーは使用できません。データを入力するとメモリ内の値は上書きされます。

警告！ メモリの内容を勝手に変更することは、非常に危険です。クラッシュを起こす恐れがあります。完全に把握していない場合、データを絶対に変更しないでください。

基本的なデバッグ

ここではデバッグの使い方について説明します。デバッグを使ってソースコードの問題点を見つけて解決する方法を説明します。

[起動方法](#)：デバッグを起動する際の注意

[コードの実行、ステップ実行、停止](#)：プログラムを 1 行ずつ実行する

[コードのナビゲーション](#)：プログラムの実行をコントロールする

[ブレークポイント](#)：指定した箇所で実行を停止する

[Watchpoint](#)：メモリ位置の内容が変更されたとき実行を停止する

[データの表示と変更](#)：変数を見たり、変更する

[ソースコードの編集](#)：デバッグ中にソースコードを編集する

この節は既にデバッグのユーザーインターフェースについての知識がある方を対象としています。「[デバッグのウィンドウ](#)」(p281) を参照してください。デバッグを使うためにビルドターゲットを準備する方法については「[デバッグの準備](#)」(p278) を参照してください。デバッグの IDE 設定については「[デバッグの IDE 設定](#)」(p201) を参照してください。

起動方法

デバッグするビルドターゲットによってデバッグの起動方法が異なります。通常、プロジェクトをデバッグするためには、プロジェクトファイルとシンボルファイルが必要です。一部のエンベデッドビルドターゲットにはこれらのファイルは必要ありません。

プロジェクトをデバッグするときは「[一般的な起動方法](#)」(p311) を参照してください。エンベデッドビルドターゲットをデバッグするときは「[エンベデッド向けの起動方法](#)」(p313)、および適切な『Targeting』マニュアルを参照してください。

一般的な起動方法

デバッグを使うためには、まずプロジェクトを開き、次にプロジェクトメニューの[デバッグを有効にする]を選択します。次にプロジェクトメニューの[デバッグ]を選択するとプロジェクトのデバッグが始まります。

デバッグを使う場合、以下の点に注意してください。

プロジェクトをデバッグするためにプロジェクトメニューの[デバッグ]を選択したとき、[プログラムウィンドウ](#)が現れます。通常、デバッグはプログラムの第 1 行目の main 関数で実行を停止します。これは正常な状態です。プログラムのコードは停止しており、実行の準備が整っています。

シンボルファイルを開いた場合、[ブラウザウィンドウ](#)が現れます。この場合はプロジェクトメニューの [デバッグ] を選択するか、[デバッガツールバー](#)の [実行] ボタンをクリックしてください。デバッガはビルドターゲットを起動してコントロール下に置き、プログラムウィンドウを最前面に表示します。プログラムは第 1 行目で停止しています。

デバッガが該当ファイルのある位置を尋ねるダイアログを表示することがあります。

このダイアログが表示されるのはデバッガを起動したとき（デバッガがメインエントリポイントを持つファイルを検索するとき）、または[ブラウザウィンドウ](#)にあるファイルをクリックしたときです。

以下の状況でもファイル検索ダイアログが表示されます。

ファイルが他のディレクトリに移動されているとき

他人から受け取ったプロジェクトファイルのパスが異なるとき

コンパイルされたライブラリに属するファイルを選択し、かつそのソースファイルを持っていないとき

ビルドターゲットが、シンボルファイルの生成をオンにしてコンパイルされたライブラリを持っている場合、最後のケースがもっともありがちなものです。CodeWarrior にはコンパイルされたライブラリが含まれていますので特にそうなることがあります。

一度ファイルを指定すると、デバッガはその位置を（デバッグ中であっても）記憶します。

[「シンボル情報を生成」\(p280\)](#) を参照してください。

図 11.32 ファイルを検索



エンベデッド向けの起動方法

ほとんどのエンベデッドプロジェクトをデバッグする手順は「[一般的な起動方法](#)」(p311)と同じです。しかし、プロジェクトファイルやシンボル情報を生成しなくともデバッグが可能なエンベデッドビルドターゲットもあります。このような場合、IDE はデフォルトの設定でデフォルトプロジェクトを作成します。デバッガはこのデフォルト設定でデバッグを行います。

さらに、デバッガはシンボル情報がなくとも実行可能ファイルをダウンロードして実行することができます。詳細は各ビルドターゲットの『Targeting』マニュアルを参照してください。

コードの実行、ステップ実行、停止

ここでは、プログラムの実行、1 行ずつのステップ実行、デバッグを終えるときのビルドターゲットの停止または終了の方法について説明します。

1 行ずつコードを実行することを「ステップ実行」と呼びます。これは、プログラムの最初から始めて、コードを順にたどっていく地道な方法であり、コードの流れを理解するには重要です。デバッガを使うと、任意の位置へ直接移動したり、特定の位置で特定の条件を満たしたときにコードを停止したり、さらにデータの表示や変更することもできます。

ツールバーのボタン、キーボード、またはデバッガの Control メニューのコマンドにより、コードのステップ実行ができます。[図 11.3](#) に、ツールバーのボタンと等価なデバッグメニューのコマンド、およびキー操作を示します。

表 11.3 ボタンと等価なデフォルトのキー操作

ボタン	メニューコマンド	キーボード操作
	実行	F5
	停止	
	中止	Shift + F5
	ステップオーバー	F10
	ステップイン	F11
	ステップアウト	Shift + F11

ここでは以下の内容について説明します。

[カレントステートメント矢印](#)

[コードを実行](#)

[1 行ずつステップ実行する](#)

[ルーチンの中へ入る](#)

[ルーチンの中から呼び出し元に戻る](#)

[ステートメントをスキップする](#)

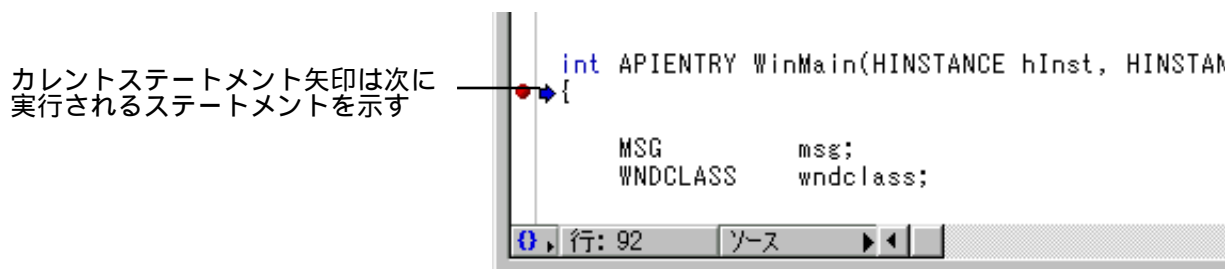
[実行を停止する](#)

[プログラムを終了する](#)

カレントステートメント矢印

プログラムウィンドウのカレントステートメント矢印 (図 11.33) は、次に実行されるステートメントを指します。これは、プロセッサのプログラムカウンタレジスタが指している位置と同じです。デバッグを始めた直後、カレントステートメント矢印はプログラムの実行コードの最初の行を指しています。

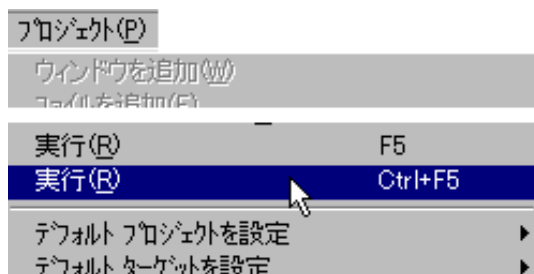
図 11.33 カレントステートメント矢印



コードを実行

プログラムが起動しているのに実行が停止している場合、プロジェクトメニューの[実行]を選択するか、ツールバーの[実行]ボタンをクリックしてください (図 11.34)。カレントステートメント矢印から実行が再開されます。

図 11.34 [実行] コマンド



ターゲットプログラムを実行するにはプロジェクトメニューの[実行]を選択するか、ツールバーの[実行]ボタンをクリックする



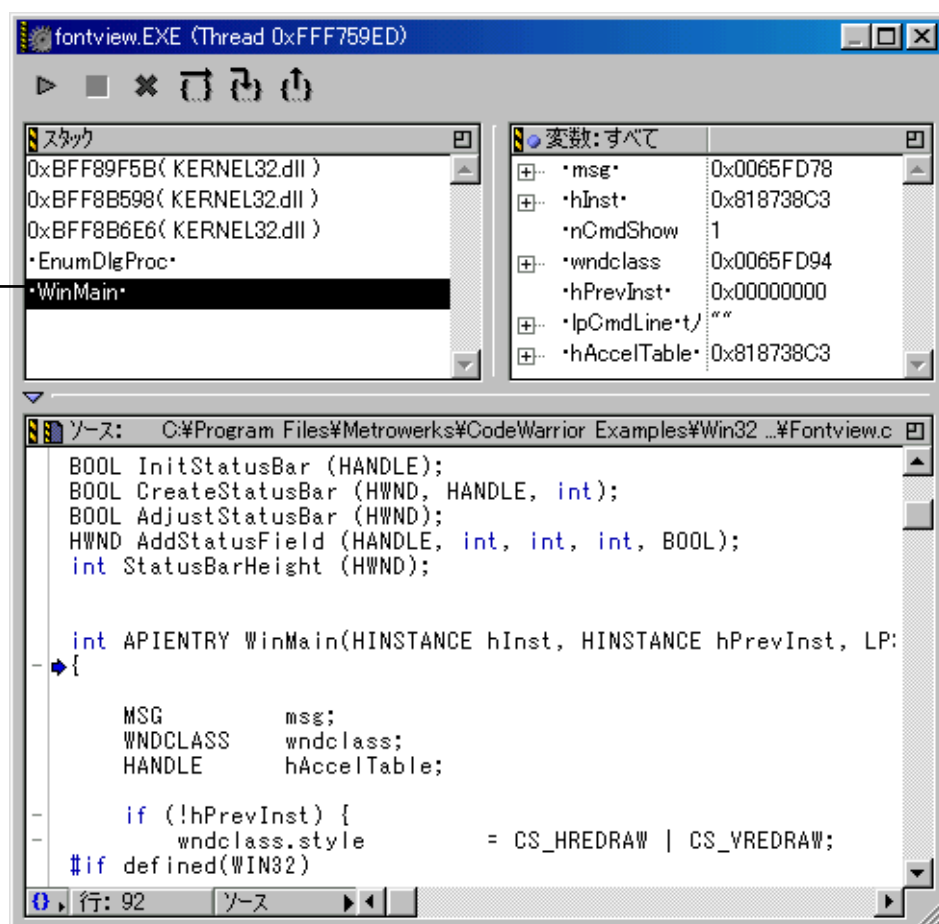
ブレークポイントまたは[停止]コマンドで停止すると、コントロールはデバッガに戻り、プログラムウィンドウにカレントステートメント矢印と変数の現在の値が表示されます。

デバッガは、暗黙のブレークポイントをプログラムのメインエントリーポイントに設定して、そこで停止します (図 11.35)。ここで [実行] コマンドを使うと、割込みの起きた場所からプログラム実行が再開されます。[中止] コマンドの後に [実行] コマンドを選択すると、プログラムを最初から再実行します。

ヒント：プログラムの自動起動を禁止するには、Alt キーを押しながらシンボルファイルを開きます。[SYM ファイルが開かれたら自動的にアプリケーションを起動する] オプションを変えておくこともできます (「一般設定」(p204) を参照)。この機能は、プログラムのメインルーチンの前に実行される、C++ の static コンストラクタ関数のデバッグなどに有効です。

図 11.35 プログラムの実行を開始

プログラムがデバッガから起動されたとき、プログラムの実行は自動的に main のエントリーポイントで停止する。

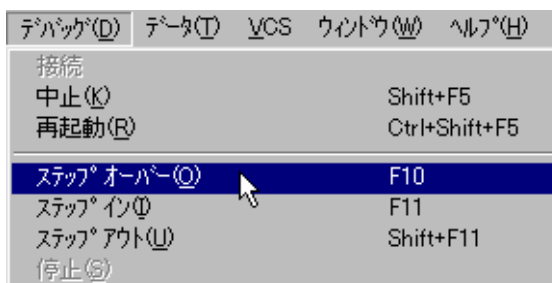


1 行ずつステップ実行する

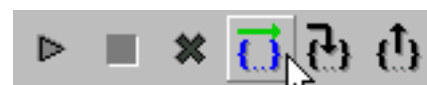
ステートメントを 1 行ずつ実行するには [ステップオーバー] コマンドを選択します (図 11.36)。ステートメントがルーチンコールの場合、ルーチン全体が実行され、カレントス

ステートメント矢印は次の行に進みます。つまり、[ステップオーバー] コマンドは、呼び出されたルーチンのコード内容にはふれずにルーチンコールを実行します。ルーチンにブレークポイントがない場合、また他のルーチンを呼び出さないときも同様です。コードを 1 行ずつ実行してルーチンの終わりに達すると、カレントステートメント矢印はルーチンの呼び出し元に戻ります。

図 11.36 [ステップオーバー] コマンド



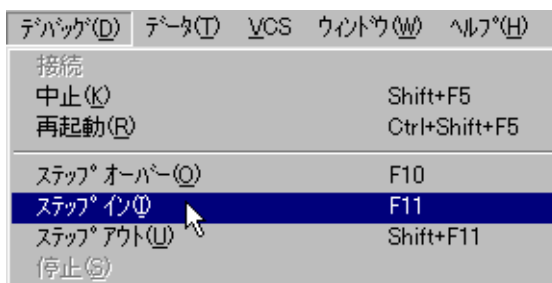
次のステートメントを実行するには、デバッグメニューの[ステップオーバー] を選択するか、ツールバーの[ステップオーバー] ボタンをクリックする。これはルーチンコールの中には入らない



ルーチンの中へ入る

呼び出したルーチンの実行を追跡することもできます。ステートメントを 1 行ずつ実行してルーチンコールにステップインするには [ステップイン] コマンドを選択します ([図 11.37](#))。

図 11.37 [ステップイン] コマンド



ターゲットプログラムの中に入るには、デバッグメニューの[ステップイン] を選択するか、ツールバーの[ステップイン] ボタンをクリックする

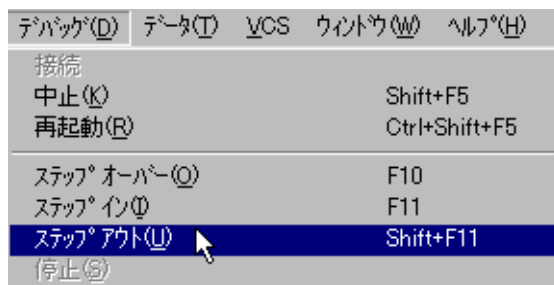


[ステップイン] は、現在のステートメントがルーチンコールでなければ、カレントステートメントを 1 ステートメント分下に移動します。[ステップイン] がルーチンコールにあたると、呼び出されたルーチンの中へ実行を移します。

ルーチンの中から呼び出し元に戻る

現在のルーチンが呼び出し元に戻るまでステートメントを実行するには [ステップアウト] コマンドを選択します ([図 11.38](#))。[ステップアウト] コマンドは、ステートメント矢印が指しているステートメントからプログラムを実行し、ルーチンがその呼び出し元に戻ると停止します。コールチェーンを 1 レベル「上」に戻ります。「[コールチェーンによるナビゲーション](#)」(p320) を参照してください。

図 11.38 [ステップアウト] コマンド



実行中のルーチンが呼び出し元へ戻るまでを実行するには、デバッグメニューの[ステップアウト]を選択するか、ツールバーの[ステップアウト]ボタンをクリックする



ステートメントをスキップする

ステートメントを実行せずにコードの別の位置へ移動することもできます。現在実行中のコードとは異なる部分へカレントステートメント矢印を移動する方法を以下の節で紹介します。

[カレントステートメント矢印を操作](#)

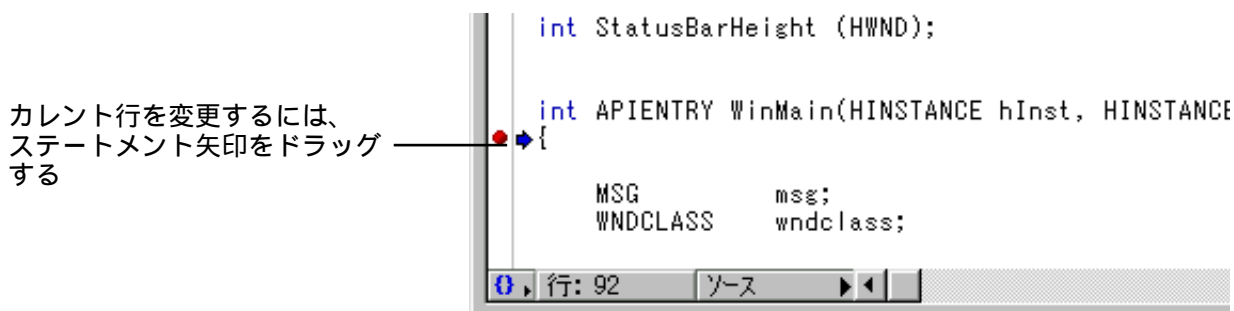
[「プログラムカウンタを変更」ダイアログを使う](#)

警告！ ステートメント矢印をドラッグすると、レジスタウィンドウのプログラムカウンタを変更することになります。これはとても危険な操作です。ルーチンコールやルーチンからの戻りをスキップすることは、スタックを破壊することにつながるからです。デバッガはランタイム環境の破壊を防止することはできないので、この操作を行うときは十分な配慮が必要です。

カレントステートメント矢印を操作

ステートメントをスキップする方法の 1 つは、カレントステートメント矢印を新しい場所へドラッグ移動することです (図 11.39)。カレントステートメント矢印を上方、または下方へとドラッグしても、元の位置と新しい位置の間にあるステートメントは実行されません。

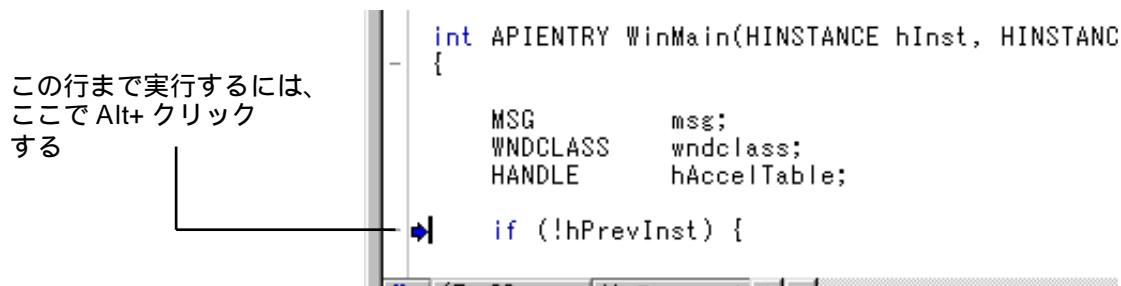
図 11.39 カレントステートメント矢印をドラッグ



ランタイム環境を壊さずにステートメント矢印を移動するには、ブレークポイント列のステートメントを Alt + クリックしてください (図 11.40)。その位置に一時的なブレークポイ

ントが設定されます。コードはカレントステートメント矢印が一時的なブレークポイントに達するまで正常に実行され、停止します。実行が停止した後、一時的ブレークポイントは自動的に除去されます。詳しくは「[ブレークポイント](#)」(p324)を参照してください。

図 11.40 一時的ブレークポイントを設定



一度に数行のステートメントをスキップするには、カレントステートメント矢印のドラッグは便利な方法です。しかしステートメント少しずつ移動するには、ドラッグでは不便です。直接カレントステートメント矢印を移動する代わりに、[プログラムカウンタを変更] コマンドのキーバインディングを利用できます。デフォルトのキーバインディングを以下に示します。

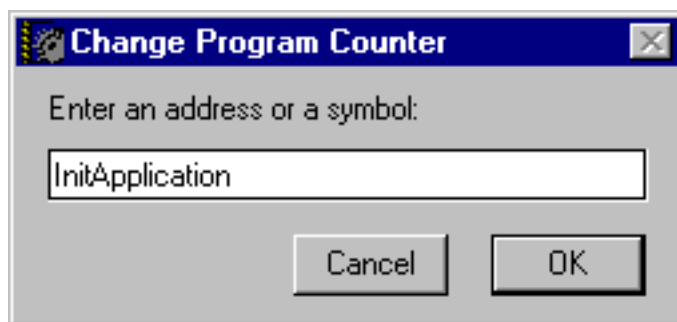
Ctrl + Shift + F10

このコマンドを使うには、スキップしたいソースコードをクリックします。次に [プログラムカウンタを変更] キーバインディングを押します。デバッガはクリックされた行へカレントステートメント矢印を移動します。矢印を移動できない場合、エラーメッセージが現れます。

「プログラムカウンタを変更」ダイアログを使う

新しいアドレスやシンボルを指定してカレントステートメント矢印を移動するには、「プログラムカウンタを変更」ダイアログ (図 11.41) を利用します。ダイアログを開くには、デバッグメニューの [プログラムカウンタを変更] を選択しています。[アドレスまたはシンボルを入力] フィールドへスキップしたいアドレスやシンボル名を入力して [OK] ボタンをクリックします。デバッガは指定されたアドレスやシンボルへカレントステートメント矢印を移動します。矢印を移動できない場合、また指定したアドレスやシンボルがシンボルファイルに含まれていない場合、エラーメッセージが現れます。

図 11.41 「プログラムカウンタを変更」ダイアログ

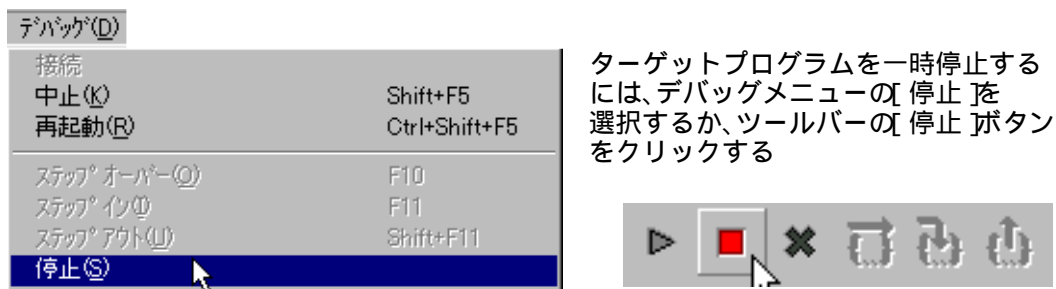


実行を停止する

プログラムの実行中でも、デバッグメニューの [停止] コマンド ([図 11.42](#)) で実行を停止してデバッガでコードを見ることができます。この後、1 行ずつ実行したり、[実行] コマンドで実行を再開することもできます。

[停止] コマンドでは、停止する場所を指定できません。コードの実行は非常に高速であるため、[停止] コマンドを使ったときにどこで止まるかは正確に指定できません。停止する場所を正確に指定したいときは、ブレークポイントを使います。[「ブレークポイント」\(p324\)](#) を参照してください。

図 11.42 [停止] コマンド

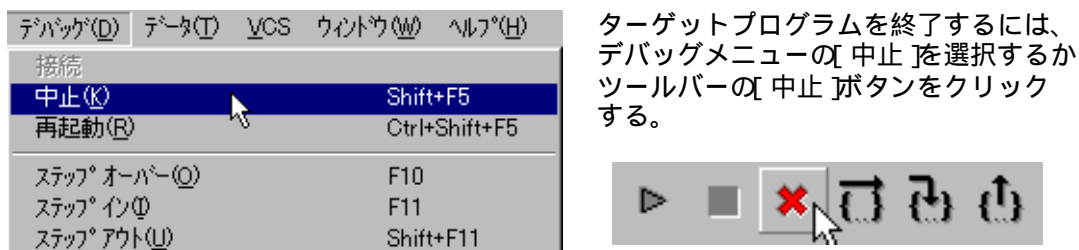


注意： [停止] コマンドはオペレーティングシステムに依存するため、ビルドターゲットによっては使用できません。詳細は各『Targeting』マニュアルを参照してください。各『Targeting』マニュアルについては、[「ターゲットマニュアル」\(p20\)](#) を参照してください。

プログラムを終了する

プログラムとデバッグを完全に終了するには、デバッグメニューの [中止] コマンドを選択するか、ツールバーの [中止] ボタンをクリックしてください ([図 11.43](#))。プログラムが終了します。プログラムが実行されている場合、プログラムウィンドウにはデバッグメニューの [中止] コマンドを選択するようにメッセージが表示されます。

図 11.43 [中止] コマンド



プログラムの終了は停止と異なります。[停止] コマンドはプログラムを一時停止するだけで、停止した位置から再開できますが、[中止] コマンドはプログラムを終了します。

実行を再開する

[デバッグ] コマンドの後に [再起動] コマンドを選択することは、[中止] コマンドと等価です。現在のデバッグを終了して新たなデバッグを始めるには、[再起動] コマンドを選択します。

コードのナビゲーション

プログラムをナビゲーション（移動）する方法を説明します。これはブレークポイントに必要な箇所に設定するために重要です。

[一般的なナビゲーション](#)：コードをステップ実行

[コールチェーンによるナビゲーション](#)：アクティブなルーチンへ移動

[ブラウザウィンドウによるナビゲーション](#)：ソースファイルのコードへ移動

[フォントとカラーの変更](#)：テキストのフォントとカラーの変更

一般的なナビゲーション

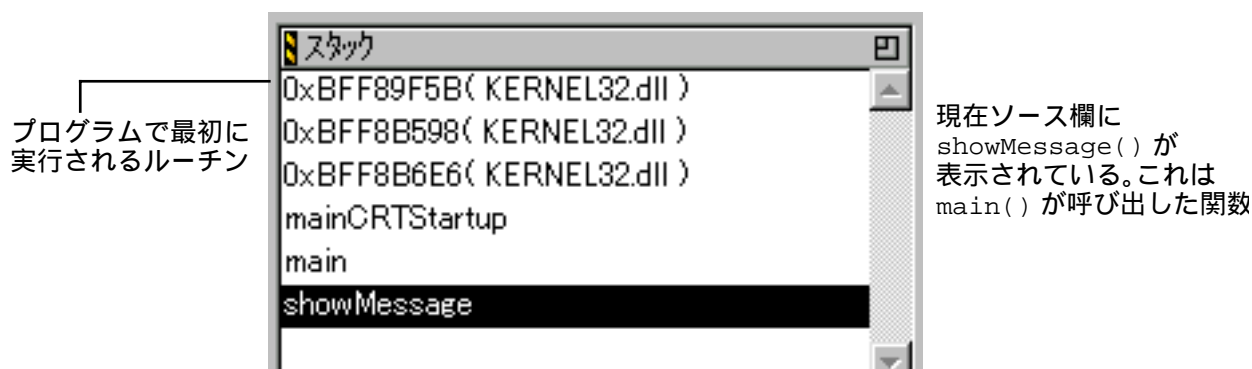
[ステップオーバー] [ステップイン] [ステップアウト] コマンドを選択してプログラムを実行しながら、必要な箇所に移動できます。少しのコードを追いたいときには有効な方法ですが、離れたところにあるコードまで実行するにはあまり有効ではありません。

[「1 行ずつステップ実行する」\(p315\)](#)、[「ルーチンの中へ入る」\(p316\)](#)、[「ルーチンの中から呼び出し元に戻る」\(p316\)](#) も参照してください。

コールチェーンによるナビゲーション

プログラムウィンドウのスタッククロール欄には、ルーチンのコールチェーンが表示されます（[図 11.44](#)）。チェーン内のそれぞれのルーチンは、呼び出し元のルーチンの下に表示されます。現在実行中のルーチンは、チェーンの一番下にあり、プログラムで最初に行われたルーチンは一番上にあります。

図 11.44 スタッククロール欄

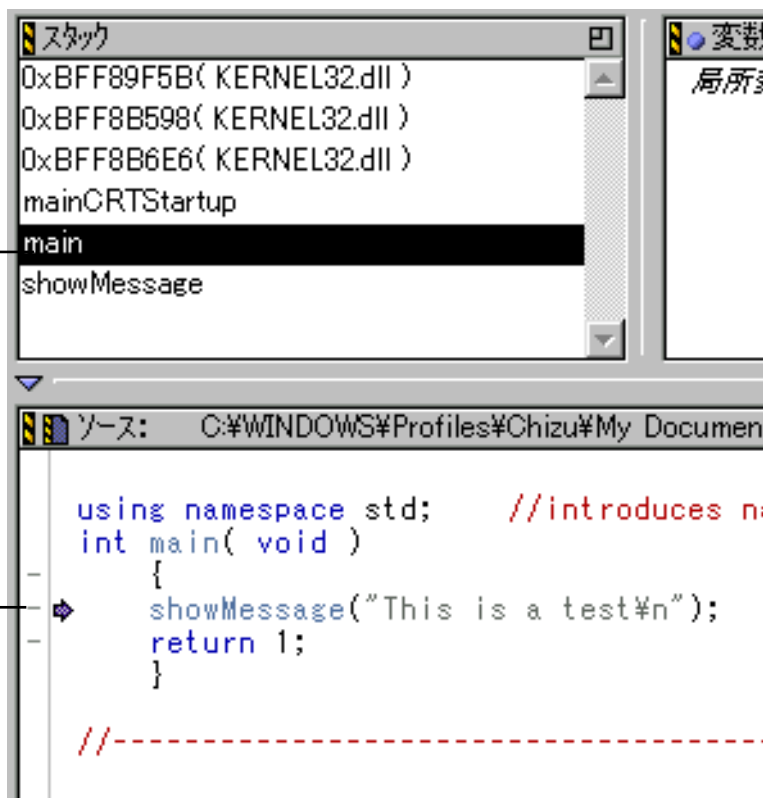


スタッククロール欄を使って、現在実行中のルーチンを呼び出したルーチンにナビゲートできます。スタッククロール欄のルーチンを呼び出したルーチンを見つけるには、呼び出

し元の名前をクリックします。ソース欄に呼び出し元のソースコードが表示されます ([図 11.45](#))。

図 11.45 ルーチンを呼び出した位置を見つける

showMessage() の呼び出し元、
main をクリックすると
main が showMessage() を
呼び出した箇所を表示する



ブラウザウィンドウによるナビゲーション

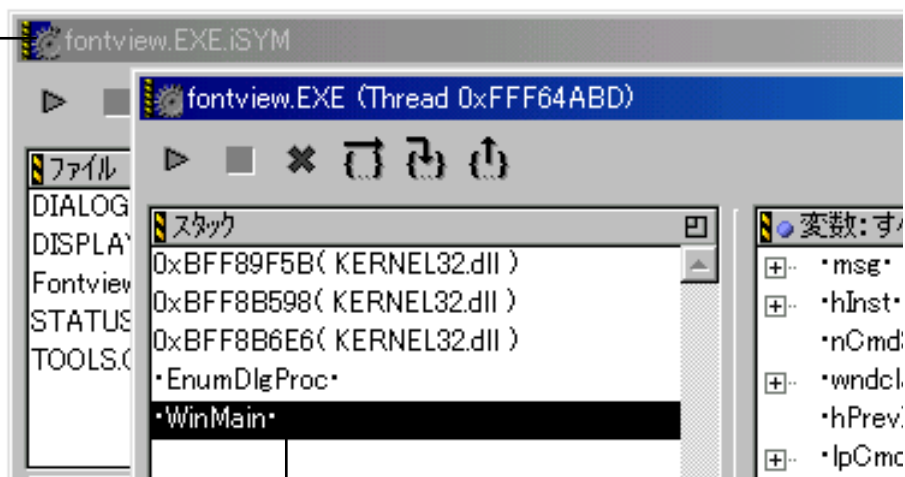
ブラウザウィンドウを使って、ソースコード内の任意の位置にジャンプできます。ブラウザウィンドウはシンボルファイルを開いたときのみ現れます。特定のルーチンを表示するには、以下のようにします。

1. ブラウザウィンドウをアクティブにする ([図 11.46](#))

図 11.46 ブラウザウィンドウをアクティブにする

アクティブにするには、
ブラウザウィンドウを
クリックする

ブラウザウィンドウは
プロジェクトに含まれる
すべてのファイルを表示
する

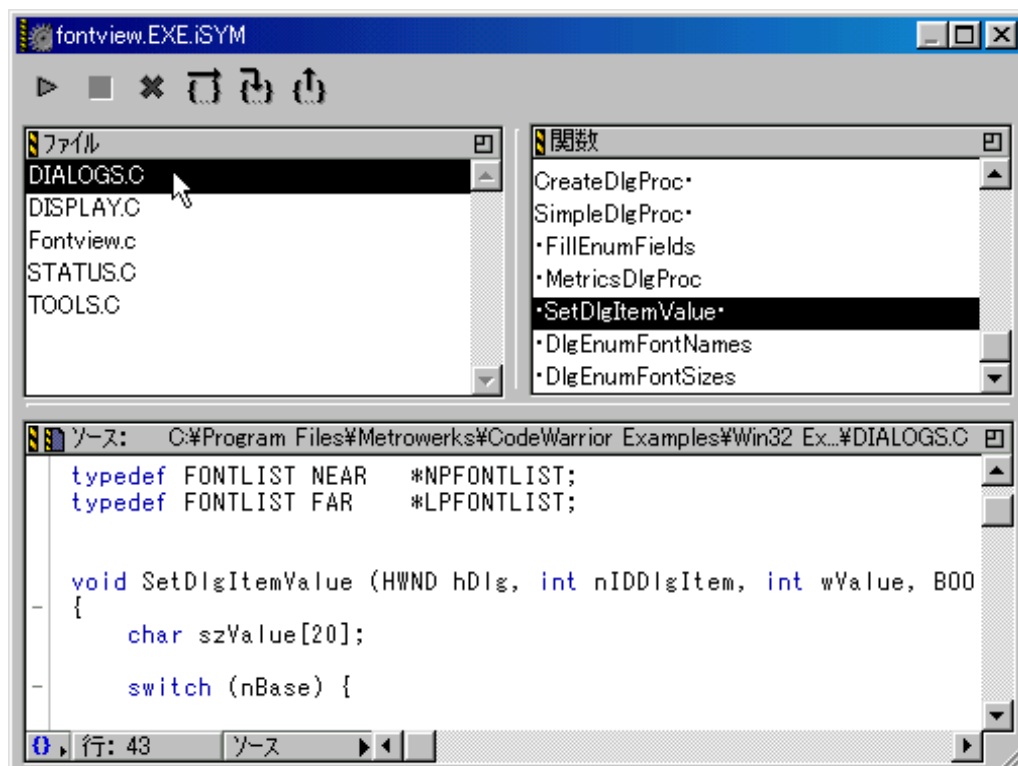


プログラムウィンドウ

2. ブラウザウィンドウのファイル欄で、ルーチンを定義しているファイルを選択する ([図 11.47](#))。

見たいファイルをクリックするか、矢印キーを使ってリストをスクロールしてください。そのファイルのソースコードがソース欄に表示されます。ファイル名を入力してもよいです。

図 11.47 内容を見たいファイルを選択する

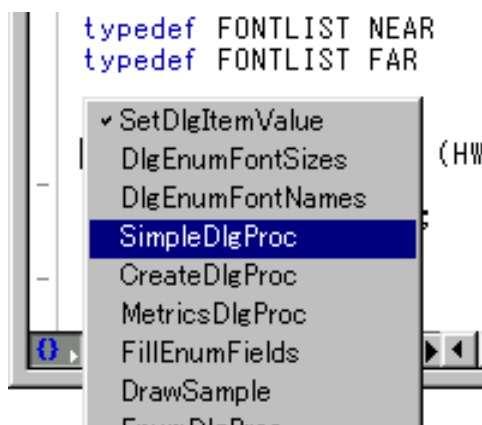


3. ソースファイル内で見たいコードを探す

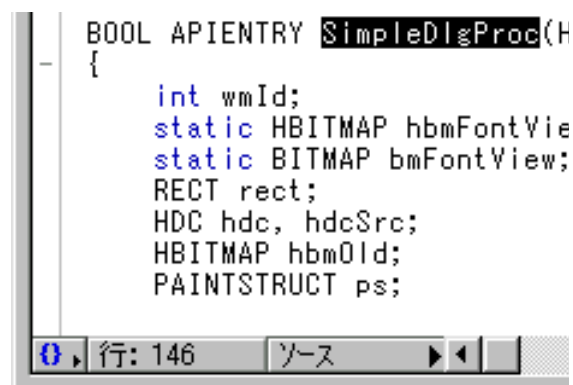
ソース欄をスクロールしてルーチンを見つけます。ブラウザウィンドウの関数欄や関数ポップアップメニュー（図 11.48）を使ってルーチンを選択する方が効率的です。ブラウザウィンドウのソース欄にルーチンが表示されます。ブレークポイントの設定もできます。詳細は「[ブレークポイント](#)」(p324)を参照してください。

図 11.48 表示するルーチンを選択する

実行前



実行後



フォントとカラーの変更

デバッガは IDE 設定パネルで指定されたフォントとカラーを使ってソースコードを表示します。

詳細は「[エディタの IDE 設定](#)」(p193) を参照してください。

ブレークポイント

ブレークポイントはプログラムの実行を一時停止し、コントロールをデバッガに戻します。デバッガは、ブレークポイントが設定されているステートメントを実行する直前にプログラムを停止します。停止すると、プログラムウィンドウにブレークポイントを含むルーチンを表示します。ブレークポイントにカレントステートメント矢印が表示され、そのステートメントから実行できることを示します。

ここでは、以下の内容について説明します。

[ブレークポイントの設定と削除](#)

[一時的なブレークポイント](#)

[ブレークポイントを表示](#)

[条件ブレークポイント](#)

[テンプレート化、再定義した関数にブレークポイントを設定](#)

[ブレークポイントでコードを最適化](#)

ブレークポイントの設定と削除


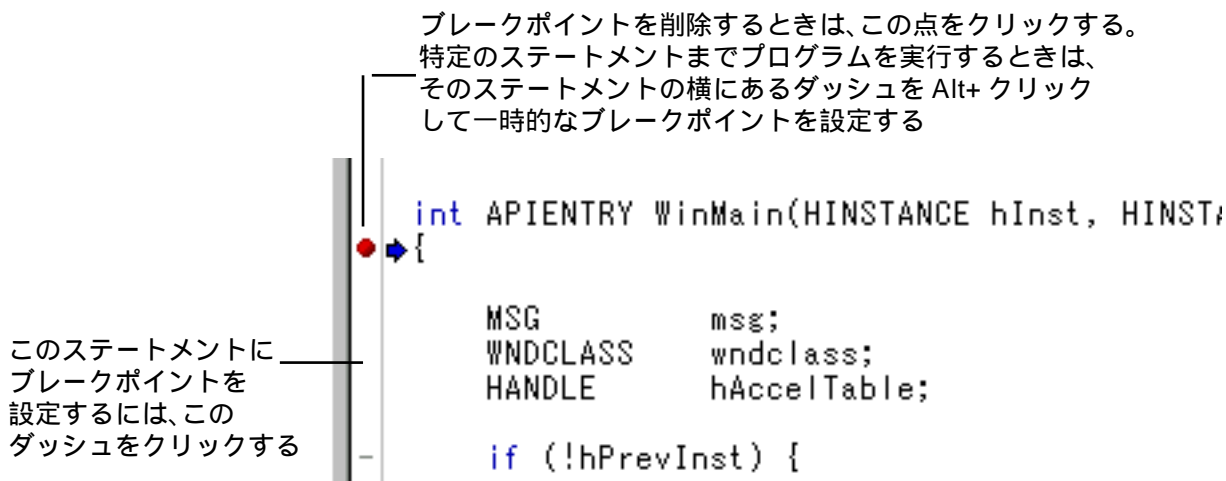
エディタウィンドウ、プログラムウィンドウおよびブラウザウィンドウのソース欄でブレークポイントを設定します。ブレークポイントは、マーカのあるところに設定できます。ブレークポイントマーカは、ステートメントの左のブレークポイント列にあるダッシュ記号です ( 11.49)。これをクリックすると丸 (カラーモニタでは赤) に変わります。これは、このステートメントにブレークポイントが設定されたことを示します。このステートメントの直前で実行が停止します。

図 11.49 ブレークポイントを設定

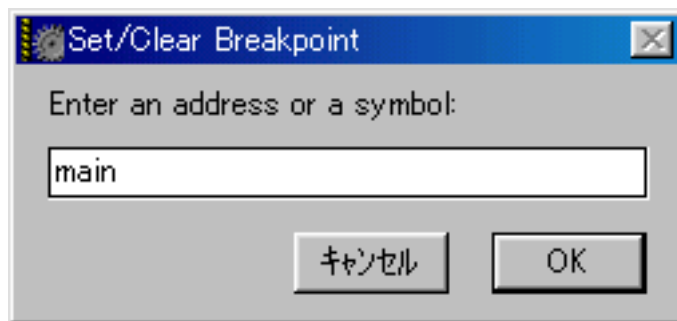


プログラムウィンドウで特定のソースコードをクリックしてから、デバッグメニューの[ブレークポイントを設定]を選択すると、ブレークポイントを設定できます。

デバッグメニューの[ブレークポイントを設定/削除する場所]コマンドを選択することでも、ブレークポイントを設定できます。「ブレークポイントを設定/削除」ダイアログ(図 11.50)が現れます。ブレークポイントを設定アドレスやシンボルをこのダイアログへ入力します。[OK] ボタンをクリックすると、デバッガは指定されたアドレスやシンボルへブレークポイントを設定します。デバッガが設定できない場合、または指定されたアドレスやシンボルがシンボルファイルに含まれていない場合、エラーメッセージが現れます。

ヒント: 「ブレークポイントを設定/削除」ダイアログでブレークポイントを削除することもできます。ブレークポイントが設定されているアドレスやシンボルを入力して[OK] ボタンをクリックします。デバッガはブレークポイントを削除します。

図 11.50 「ブレークポイントを設定/削除」ダイアログ



1 つのブレークポイントを削除するには、ソース欄のブレークポイントマーカをクリックします。丸がダッシュに変わり、ブレークポイントが削除されます。

デバッグメニューのコマンドでもブレークポイントを操作できます。

[ブレークポイントを削除]: 現在の行からブレークポイントを削除する

[ブレークポイントを有効にする]: 現在の行のブレークポイントを有効にする

[ブレークポイントを無効にする]: 現在の行のブレークポイントを無効にする

[ブレークポイントをすべて削除]: プログラムのすべてのブレークポイントを削除する

[ブレークポイントを表示]: 隠されているブレークポイント列を表示する

[ブレークポイントを隠す]: 表示されているブレークポイント列を隠す

またデバッグのソースコードビューやブレークポイントウィンドウのコンテキストメニューでも同じコマンドを使うことができます。詳細は「[デバッグのコンテキストメニュー](#)」(p294)を参照してください。

ヒント： コードの各行に 1 つのステートメントを書くようにしてください。ソースコードが読みやすいだけでなく、デバッグが簡単になります。ソースコードの 1 行に含まれているステートメントの数にかかわらず、ブレークポイントは 1 つしか設定できません。

一時的なブレークポイント


通常のブレークポイントは、デバッグ中にプログラムの実行を各行で停止します。一時的なブレークポイントはプログラムの実行を一度だけ停止します。一時的なブレークポイントに達するとデバッグはプログラムを停止して一時的なブレークポイントを削除します。

一時的なブレークポイントを設定するには、停止したいステートメントの左のダッシュを Alt + クリックします。

注意： Alt + クリックしたところに既に普通のブレークポイントが設定されていると、そのブレークポイントは削除されますが、一時的なブレークポイントは有効です。

一時的なブレークポイントに着く前に、他のブレークポイントがある場合、プログラムは最初のブレークポイントで一時停止します。一時的なブレークポイントはまだ有効です。一時的なブレークポイントに実行が達した時に停止し、消去されます。

ブレークポイントを表示

現在設定されているブレークポイントの一覧を見るには、デバッグのウィンドウメニューの[ブレークポイントウィンドウ]を選択してください。各ブレークポイントのソースファイルと行番号を示すブレークポイントウィンドウが表示されます ( 11.51)。ブレークポイントウィンドウ上でブレークポイントマーカをクリックすることで、状態の切り替えができます。丸はアクティブなブレークポイントを示します。デバッグはアクティブなブレークポイントでプログラムの実行を停止します。円は無効のブレークポイントを示します。無

効のブレークポイントではプログラムを停止せず、実行を続けます。デバッガは無効のブレークポイントの位置も記憶しています。

注意： ブレークポイントウィンドウ上でブレークポイントをダブルクリックすると、プログラムウィンドウかブラウザウィンドウ上に該当するソースコードを表示します。

詳細は「[ブレークポイントウィンドウ](#)」(p298) を参照してください。

図 11.51 ブレークポイントウィンドウを表示



条件ブレークポイント

特定の位置で、指定した条件に合致したときに停止させる「条件ブレークポイント」を指定できます。条件ブレークポイントは、普通のブレークポイントに停止する条件式を付けたものです。コントロールがブレークポイントに達したとき、評価式の結果が TRUE (0 以外の値) であれば、そのブレークポイントでプログラムが停止します。このとき、評価式の結果が FALSE (結果の値が 0) であれば、プログラムはブレークポイントを無視して実行を続けます。

条件ブレークポイントは、ブレークポイントウィンドウで作成します。以下の手順に従ってください。

1. ソース欄でブレークポイントを設定する
2. ウィンドウメニューの [ブレークポイントウィンドウ] を選択して、ブレークポイントウィンドウを表示する

3. ブレークポイントウィンドウで、ブレークポイントの [条件] フィールドをダブルクリックして、評価式を入力する。または、ソースコードビューか「Expressions」ウィンドウから評価式をドラッグする

[図 11.52](#) では、デバッガは 13 行目の `main()` ルーチンで変数 `counter` が 6 より大きいときだけ、実行を停止します。

図 11.52 条件ブレークポイントを作成



テンプレート化、再定義した関数にブレークポイントを設定

テンプレート化した関数や再定義した関数にブレークポイントを設定することができます。

`file.h` ファイルで関数 `void FOO(args)` (`FOO` はマクロ) を定義していると仮定します。[例 11.1](#) のようなコードを書くと、デバッグ中にブレークポイント列にポップアップメニューが表示されます。このポップアップメニューで `function1`、`function2` のどちらにブレークポイントを設定するのかが選択できます。

例 11.1 テンプレート化、および再定義した関数

```
#define FOO function1
#include file.h
#define FOO function2
#include file.h
```

注意： 条件ブレークポイントは、ループ中で数回実行した後に停止したいときに有効な手段です。条件ブレークポイントをループの中に置いて、ループのインデックスが望みの値に達したら停止するように設定できます。

ブレークポイントでコードを最適化

正確にブレークポイントを設定するために、デバッガはソースコードとオブジェクトコードの間の直接の対応関係に依存しています。コードの最適化はこの関係を壊すため、ブレークポイントに問題を発生させるかもしれません。

ソースコードの左側にブレークポイントのバー (-) がない場合、その行にブレークポイントを設定することはできません。その理由を以下に示します。

その行のシンボル情報が利用できない。

その行のルーチンは未使用であるため、リンカによって削除される。

コードは既に最適化されていて、最終的なオブジェクトコードと元のソースコードが対応していない。

例えば PowerPC コンパイラでは、「ステートメントブロック」の開始点に対応するソースの開始点にブレークポイントを設定できますが、そこには最低でも 1 つのインストラクションが含まれていなくてはなりません。

通常、ソースファイルのデバッグマーカがオンの場合、コンパイラは実際にコードを生成する、各ソースの新しいステートメントブロックから開始させようとしています ([例 11.2](#))。

例 11.2 ブレークポイントのサンプルコード

```
- int i = 1;
- if (i)
-     {                                     //This is the third line.
-         int k;
-         int j = 1;
-         i = j;
-     }
```

3 行目の { のような行にはブレークポイントを設定できません。これはインストラクションを生成しないので、このソースのユニークなオブジェクトコードアドレスがないためです。

一度でも最適化の設定をオンにすると、すべてが破壊されます。例えば [グローバル最適化](#) 設定パネルの [インストラクションスケジューリング] オプションがオンの場合、コンパイラは各ソースステートメントの新しい基本ブロックを開始することはできなくなります。これはブロック内の命令の並べ替えにおいてスケジューラーに最大限の柔軟性を与えます。ソースに対応する異なるインストラクションは不連続になるため、他のステートメントからのインストラクションと混同されてしまいます。最適化を行うとデバッガはブレークポイントのダッシュを表示テキストなくなります ([例 11.2](#))。

さらに最適化を行うと、ソースステートメントが生成されたコードから消えます。以下のようになります。

例 11.3 ループのサンプルコード

```
i = 0;
j = 0;
while (i < 10)
{
    j = j + 1;
    i = i + 1;
}
```

最小限の最適化では、コンパイラは[例 11.3](#) のコードを次のように翻訳します。

```
j = j + 1;          // duplicate 10 times
j = j + 1;
.
.
.
j = j + 1;
```

最適化のレベルを上げると、コンパイラは `while` ループに対応するインストラクションを完全に消去して、次のような等価のコードを使います。

```
j = 10;
```

コードをデバッグするときには、最適化をオフにするか、デバッグに危険のない程度の最適化だけを行ってください。ビルドターゲットによって可能な最適化は異なります。詳細は各『Targeting』マニュアルを参照してください（「[ターゲットマニュアル](#)」(p20)）。

プロジェクトの最適化レベルをターゲット設定ダイアログの[グローバル最適化](#)設定パネルで見ることができます。「[ターゲットのオプション設定](#)」(p225) を参照してください。

ブレークポイントを設定した後もプログラムのデバッグを続けることができます。詳細は「[コードの実行、ステップ実行、停止](#)」(p313) を参照してください。

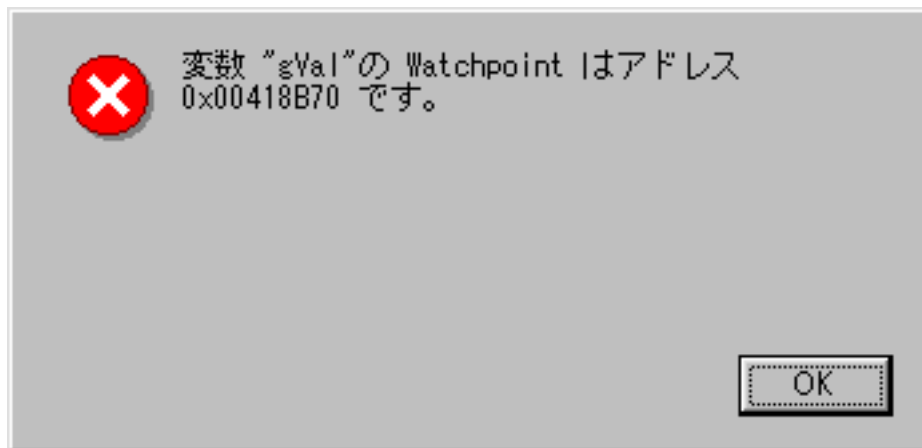
Watchpoint

Watchpoint とは、デバッガで注視したいメモリ上の位置または範囲です。この領域に新しい値が書き込まれるとデバッガは実行を停止し、アラートを表示します（[図 11.53](#)）。その後はデバッガにコントロールが戻るので、普通にデバッガのコマンドを使ってコールチェーンをたどったり、変数の値を表示、変更したり、ステップ実行したりできます（特にデバッガレベルでは、Watchpoint をトリガした位置の内容を、再度 Watchpoint をトリガすることなく変更できます）。[実行] コマンド（またはツールバーの [実行] ボタン）を使って実行を再開できます。

[Watchpoint の設定と消去](#)

[Watchpoint を表示](#)

図 11.53 Watchpoint の警告



Watchpoint の設定と消去

以下のいずれかの操作で、Watchpoint を設定できます。

- 変数ウィンドウ、またはブラウザウィンドウで変数を選択し、デバッグメニューの [Watchpoint を設定] を選択する

- 他のウィンドウから変数を Watchpoint ウィンドウへドラッグする

- メモリウィンドウのある範囲を選択し、デバッグメニューの [Watchpoint を設定] を選択する

- 変数やメモリ範囲からデバッガのコンテキストメニュー ([図 11.16 \(p294 \)](#)) を開いて [Watchpoint を設定] を選択する

Watchpoint が設定された変数またはメモリ範囲は、変数ウィンドウ、メモリウィンドウ上で下線付きの赤字で示されます。デフォルトの色の変更方法は [「カラーシンタックス」\(p198\)](#) を参照してください。

警告！ Watchpoint を指定できるメモリ領域には若干の制限があります。Watchpoint を設定できるのは大域変数、およびアプリケーションヒープにとられるオブジェクトのメモリ領域上だけです。スタックベースの局所変数やレジスタ変数には Watchpoint を設定できません。

以下のいずれかの操作で Watchpoint を消去できます。

- Watchpoint で停止した後、デバッグメニューの [Watchpoint をすべて削除] を選択する

- 変数ウィンドウ、またはブラウザウィンドウで変数を選択し、デバッグメニューの [Watchpoint を削除] を選択する

- メモリウィンドウから範囲またはバイトを選択し、デバッグメニューの [Watchpoint を削除] を選択する

Watchpoint ウィンドウで既存の Watchpoint を選択した後、以下のいずれかの操作を行う

- デバッグメニューの [Watchpoint を削除] を選ぶ
- 編集メニューの [消去] を選ぶ
- Backspace キーを押す

Watchpoint を選択してから、デバッガのコンテキストメニューで [Watchpoint を削除] を選択する。

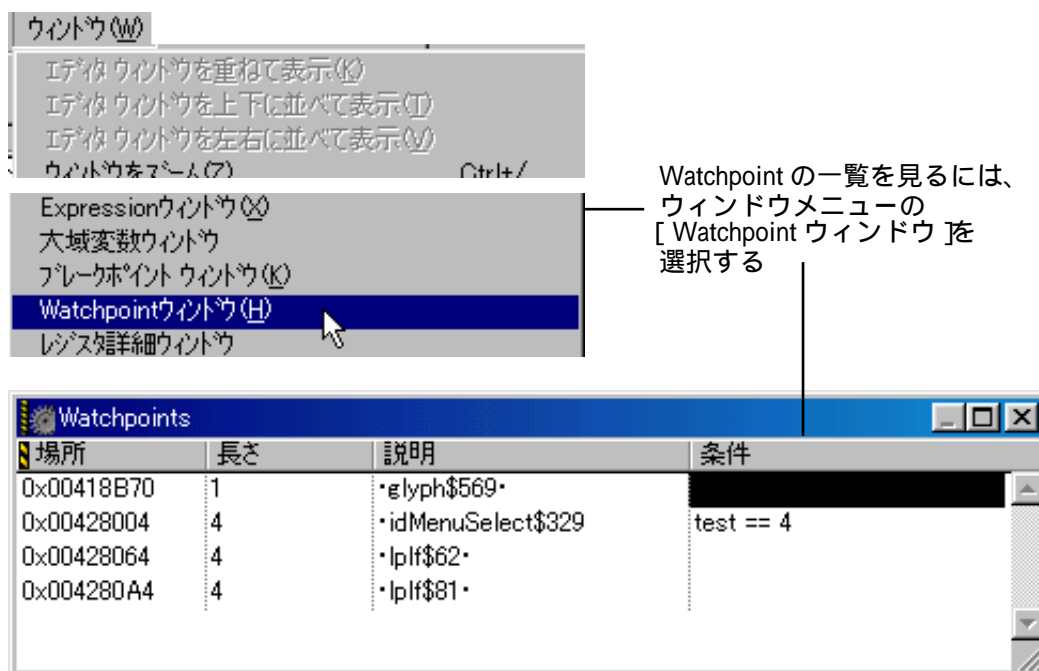
ターゲットプログラムが終了または中止されたとき、すべての Watchpoint は自動的に消去されます。

Watchpoint を表示

現在設定されている Watchpoint の一覧を表示するには、ウィンドウメニューの [Watchpoint ウィンドウ] を選びます。Watchpoint ウィンドウが表示されます ([図 11.54](#))。

詳細は「[Watchpoint ウィンドウ](#)」(p299) を参照してください。

図 11.54 Watchpoint ウィンドウを表示



データの表示と変更

デバッガの重要な機能に、変数の現在値を表示したり、必要に応じてその値を変更できる機能があります。ここでは変数を見たり、変更する方法について説明します。以下の内容について説明します。

[局所変数を表示](#)

[大域変数を表示](#)

[データを専用ウィンドウに表示](#)

[データ型を表示](#)

[デフォルトフォーマットでデータを表示](#)

[異なる型でデータを表示](#)

[変数の値を変更](#)

[Expression ウィンドウを使う](#)

[ローメモリを表示](#)

[アドレスを指定してメモリを表示](#)

[プロセッサレジスタを表示](#)

特定のターゲットのデータを表示、変更するための方法は、各『Targeting』マニュアルを参照してください。

局所変数を表示

局所変数は、プログラムウィンドウの変数欄に表示されます ([図 11.55](#))。変数がハンドル、ポインタ、または構造体の場合、名前の左にある拡張ボタンをクリックすると、表示を拡張してさらに詳しい情報 (構造体のメンバ、ポインタやハンドルによって参照されているデータ) を見ることができます。

拡張ボタンの便利な使い方は「[グループの拡張と縮小](#)」(p54) を参照してください。プログラムウィンドウの変数欄については「[変数欄](#)」(p284) を参照してください。

図 11.55 局所変数を表示する

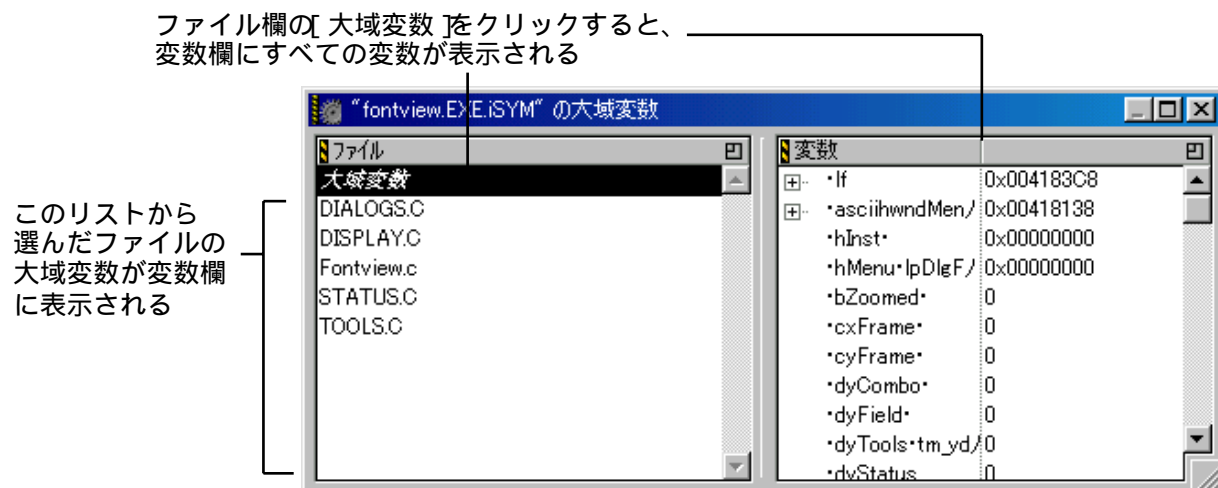


大域変数を表示

大域変数を表示するには、ウィンドウメニューの [大域変数ウィンドウ] を選択してください。次にファイル欄の [大域変数] 項目 ([図 11.56](#)) をクリックするとプログラムの大域変数に変数欄に表示されます。

詳細は「[大域変数ウィンドウ](#)」(p297) を参照してください。

図 11.56 大域変数を表示する



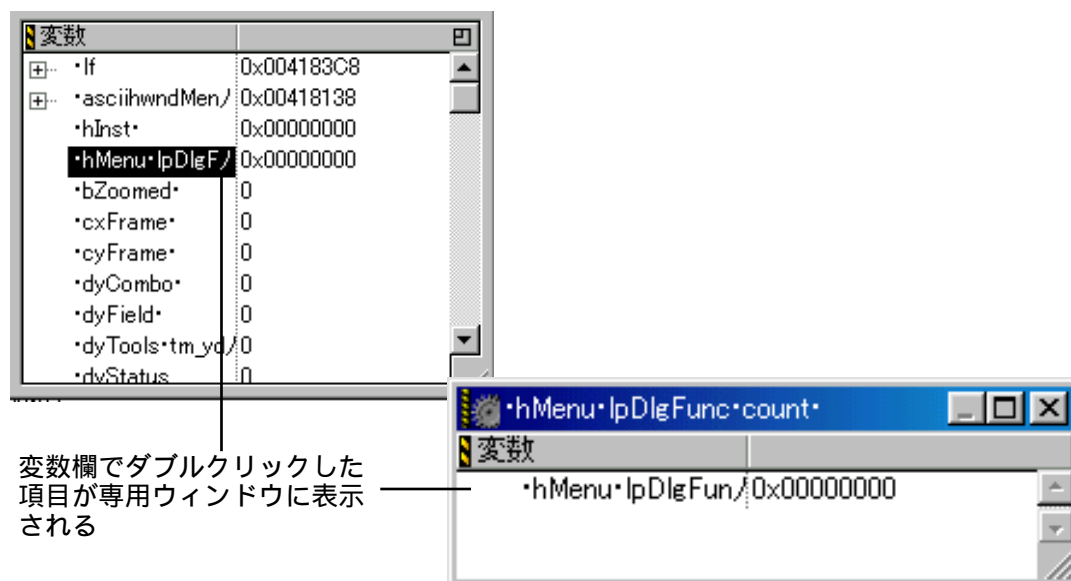
データを専用ウィンドウに表示

変数欄や大域変数ウィンドウではデータを見づらいことがあります。別のウィンドウに変数グループを表示することができます。

変数やメモリ位置を専用のウィンドウに表示するには、名前をダブルクリックします (図 11.57)。または名前を選択してからデータメニュー、デバッガコンテキストメニューの「変数を表示」を選択します (図 11.16 (p294))。変数が配列の場合、[配列を表示] コマンドを選択してください。変数が占有しているメモリをメモリダンプとして表示するには、[メモリを表示] か [メモリの表示を変更] コマンドを選択してください。

詳細は「[変数ウィンドウ](#)」(p307)、「[配列ウィンドウ](#)」(p307)、「[メモリウィンドウ](#)」(p309)を参照してください。

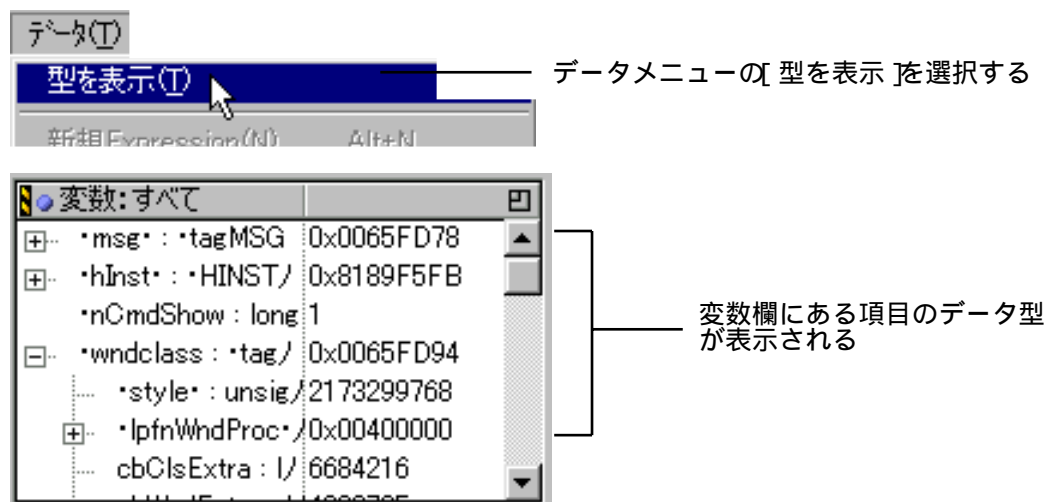
図 11.57 変数を専用のウィンドウに表示



データ型を表示

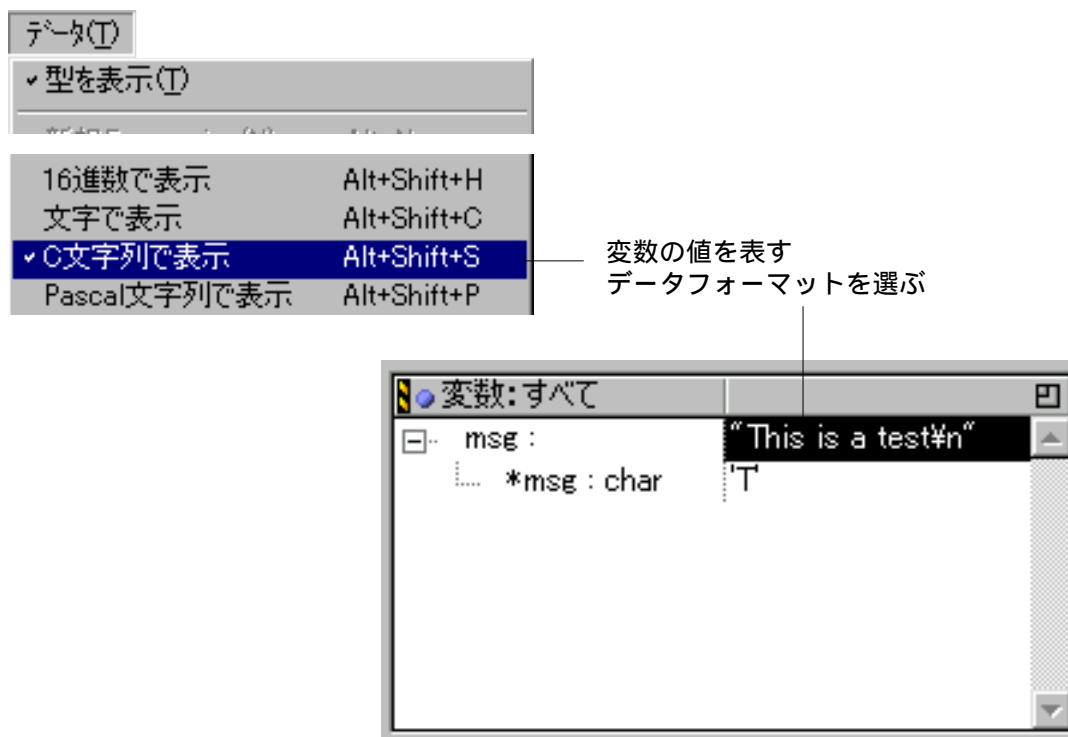
デバッガは 変数のデータ型をウィンドウごとに表示することができます。データ型を表示したいウィンドウ欄を選択して、データメニューかコンテキストメニューの[型を表示]を選択します(図 11.16 (p294))。ウィンドウ、または欄に変数の名前とメモリ位置、データ型が表示されます(図 11.58)。

図 11.58 データ型を表示



ヒント： 自動的に変数欄にデータ型を表示するには、IDE 設定の表示設定パネルの [変数表示域内で、変数の型を表示する] オプションをオンにしてください。
[「表示設定」\(p201\)](#) を参照してください。

図 11.59 データフォーマットを選択



デフォルトフォーマットでデータを表示

変数の値を以下のフォーマットで表示することができます。

- binary
- 符号付き 10 進数
- 符号なし 10 進数
- 16 進数
- 文字
- C 文字列
- Pascal 文字列
- unicode string
- 浮動小数点型
- 列挙型 (enum)

Fixed

Fract

希望のフォーマットでデータを表示するには、ウィンドウで変数の名前か値を選択してからデータメニュー（[図 11.59](#)）か、コンテキストメニュー（[図 11.16 \(p294\)](#)）からフォーマットを選択してください。

データ型によっては利用できないフォーマットがあります。例えば変数が整数値（`short` や `long` 型）の場合、符号付き 10 進数、符号なし 10 進数、16 進数、文字、文字列のフォーマットで表示できますが、浮動小数点型、Fixed 型、Fract 型では表示できません。

異なる型でデータを表示

データメニューの [表示を変更] コマンドで変数、レジスタ、メモリ値を表示するデータ型を変更できます。

1. ウィンドウまたは欄で項目を選択する
2. データメニューの [表示を変更] を選択する

デバッガのコンテキストメニュー（[図 11.16 \(p294\)](#)）でも同じコマンドを選択できます。[表示を変更] を選択した後、デバッガは「表示を変更」ダイアログを表示します（[図 11.60](#)）。

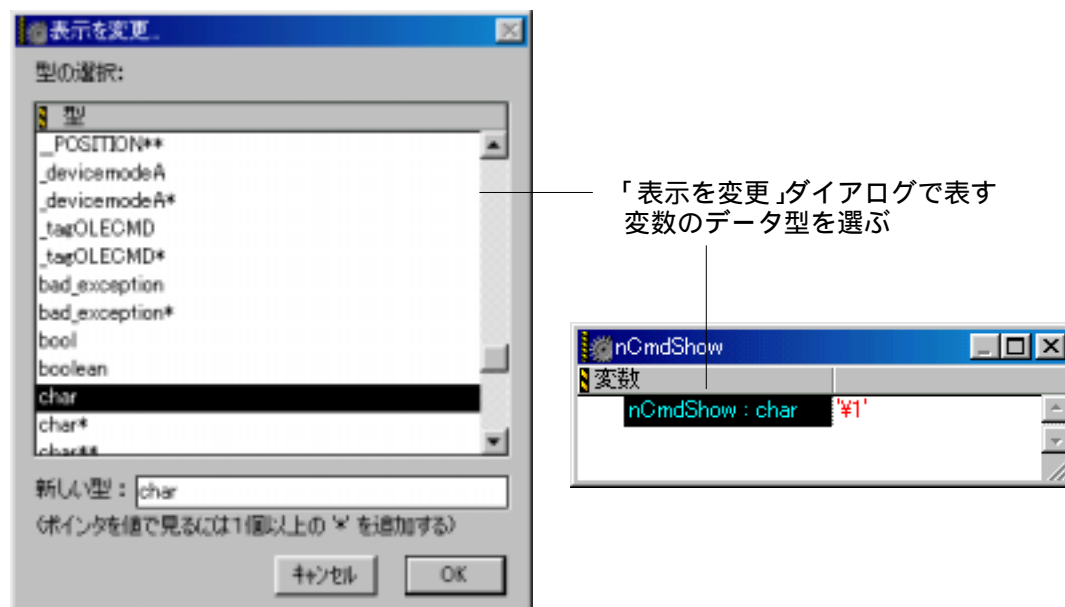
3. 項目を表示するデータ型を選択する

ダイアログ下部の [新しい型] フィールドに選択したデータ型が表示されます。項目をポインタとして扱うには、型の名前にアスタリスク（*）を付けます。

4. [OK] ボタンをクリックする

指定した型で項目の値が表示されます。

図 11.60 データ型を選択



変数の値を変更

任意の変数欄で変数の値を変更することができます。変数はさまざまなデバッガのウィンドウで表示されます。プログラムウィンドウの変数欄、ブラウザウィンドウ、変数ウィンドウ、配列ウィンドウ、Expression ウィンドウなどがあります。値を変更するには、変数をダブルクリックして（または選択してキーを押して）新しい値を入力します（[図 11.61](#)）。

以下のフォーマットで値を入力できます。

10 進数

16 進数

浮動小数点型

C 文字列

Pascal 文字列

文字定数

図 11.61 変数の値を変更



文字列または文字定数を入力するには、C 形式のクォート記号を使います（文字定数にはシングルクォート '、文字列にはダブルクォート "）。Pascal 文字列には、文字列の第一文字としてエスケープシーケンス \p を使います。

警告！ 変数の値の変更は危険です。変数の型や値を変更することができますが、ポインタを nil にするとマシンがクラッシュします。

Expression ウィンドウを使う

「Expressions」ウィンドウは、頻繁に使用する局所変数や大域変数、構造体のメンバ変数や配列要素、複雑な評価式を 1 つのウィンドウにまとめて表示します。別々のウィンドウを開く必要はありません。「Expressions」ウィンドウを開くには、ウィンドウメニューの [Expression ウィンドウ] コマンドを選択します。選択した項目を「Expressions」ウィンドウへ追加するには、データメニューかコンテキストメニューの [Expression へコピー] を使います。他の変数欄やウィンドウから「Expressions」ウィンドウへドラッグすることもできます（[図 11.16 \(p294\)](#)）。

「Expressions」ウィンドウの内容は、デバッガが実行を停止する度に更新されます。範囲外の変数は空白のままです。「Expressions」ウィンドウで以下のような便利な作業ができます。

ルーチンの局所変数を、その内容を表示するために拡張する前に「Expressions」ウィンドウへ入れておきます。そのルーチンが終了すると変数は「Expressions」ウィンドウに残り、ルーチンに実行が戻ると拡張されたままになります。「Expressions」ウィンドウは、ルーチンが終了してその変数が使用範囲の外に出てしまっても、拡張した変数表示が自動的に縮小されることはありません（局所変数欄などは変数が使用範囲の外に出ると変数表示は自動的に縮小されます）。

データメニューの [Expression へコピー] と [表示を変更] コマンドで、同じデータ項目のコピーを複数作成し、異なるデータ型で表示しておくことができます。

リストを並べ変えることができます。「Expressions」ウィンドウ内で項目をドラッグすることで順序を変えることができます。

局所変数を呼出しルーチンから表示できます。呼び出し元の局所変数を表示するためにコールチェーンをさかのぼる必要はありません（これを行うと、現在実行中のルーチンの局所変数が隠されてしまいます）。呼び出し元の局所変数を「Expressions」ウィンドウに追加することにより、スタッククロール欄のレベルを変更せずに表示することができます。

詳細は「[Expression ウィンドウ](#)」(p297) を参照してください。

ローメモリを表示

ローメモリの値を見たり、変更したりするときは以下の手順にしたがってください。

1. 表示したいメモリのベースアドレスを示す項目または評価式を選択する
2. データメニューの [メモリを表示] を選択する

新しいメモリウィンドウが開き、16 進数および ASCII 文字でメモリの内容を表示します。このメモリウィンドウ上で、16 進数または ASCII 文字を入力してメモリの内容を直接変更できます。さらに、ウィンドウ上部の [表示] フィールドで式を変更することによって、表示されているメモリの先頭アドレスを変更することもできます。

アドレスを指定してメモリを表示

データメニューの [メモリを表示] および [メモリの表示を変更] コマンドで、任意のポインタ（レジスタに格納されたアドレスも可）を追跡し、それが指しているメモリを表示できます。ポインタが参照しているメモリを表示するには、以下の手順に従ってください。

1. 表示したいウィンドウで変数の値またはレジスタを選択する
2. データメニューで [メモリを表示] または [メモリの表示を変更] コマンドを選択する

[メモリを表示] を選択するとメモリウィンドウが開き、ポインタが参照しているアドレスから始まるローメモリのダンプを表示します。[メモリの表示を変更] を選択すると、データ型を選択するダイアログが表示されるので（[図 11.62](#)）ステップ 3 へ進んでください。

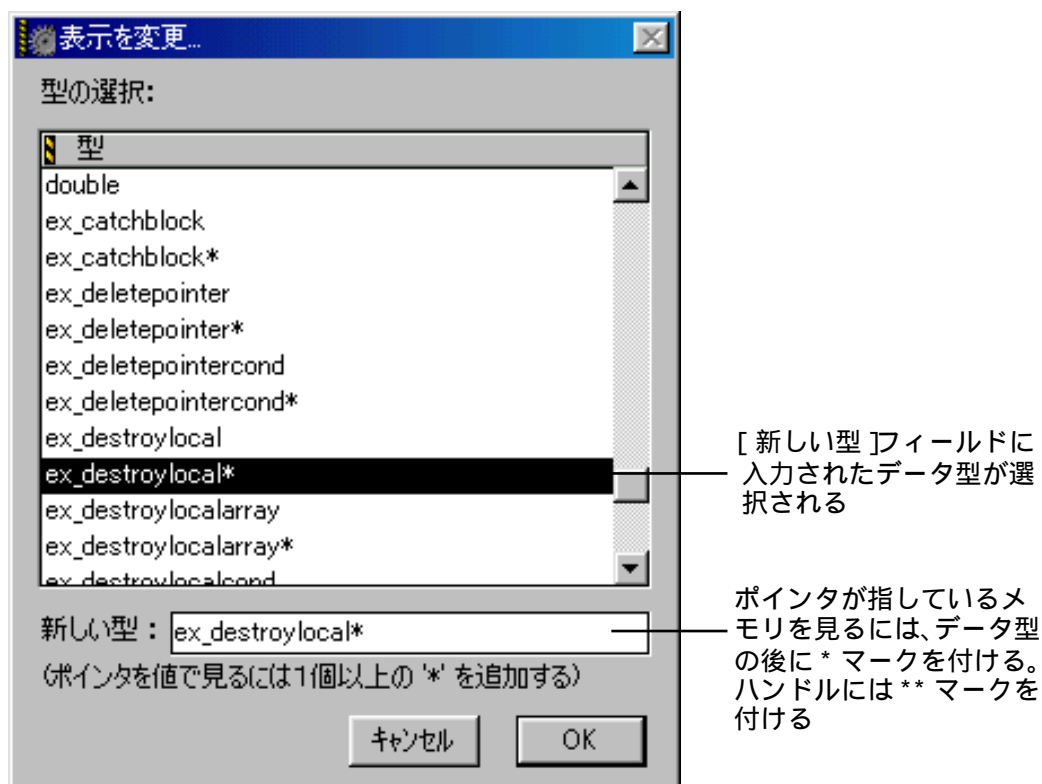
3. [メモリの表示を変更] を選択した場合はダイアログでデータ型を選択する

選択したデータ型がダイアログの下部の [新しい型] フィールドに表示されます。レジスタが指しているメモリを表示するには、型名にアスタリスク (*) を付けます。

4. [OK] ボタンをクリックする

新しいウィンドウが開き、ポインタが参照するアドレスから始まるメモリの内容を表示します (図 11.62)。

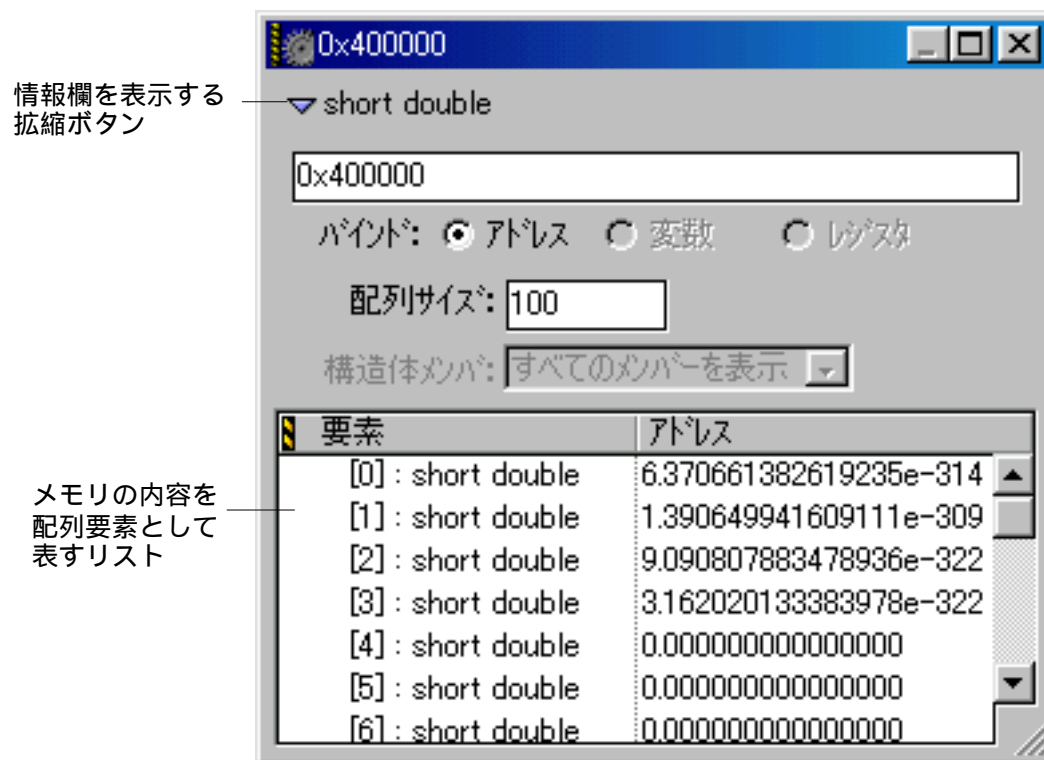
図 11.62 表示するメモリのデータ型を選択



注意： 同じ方法を使って、スタックの内容を表示することができます。ターゲットプロセッサがスタックポインタをレジスタに格納しているのならば、そのレジスタの値を選択します。その後は上記の手順と同じです。

[「メモリウィンドウ」\(p309\)](#) も参照してください。

図 11.63 特定のデータ型でメモリを表示

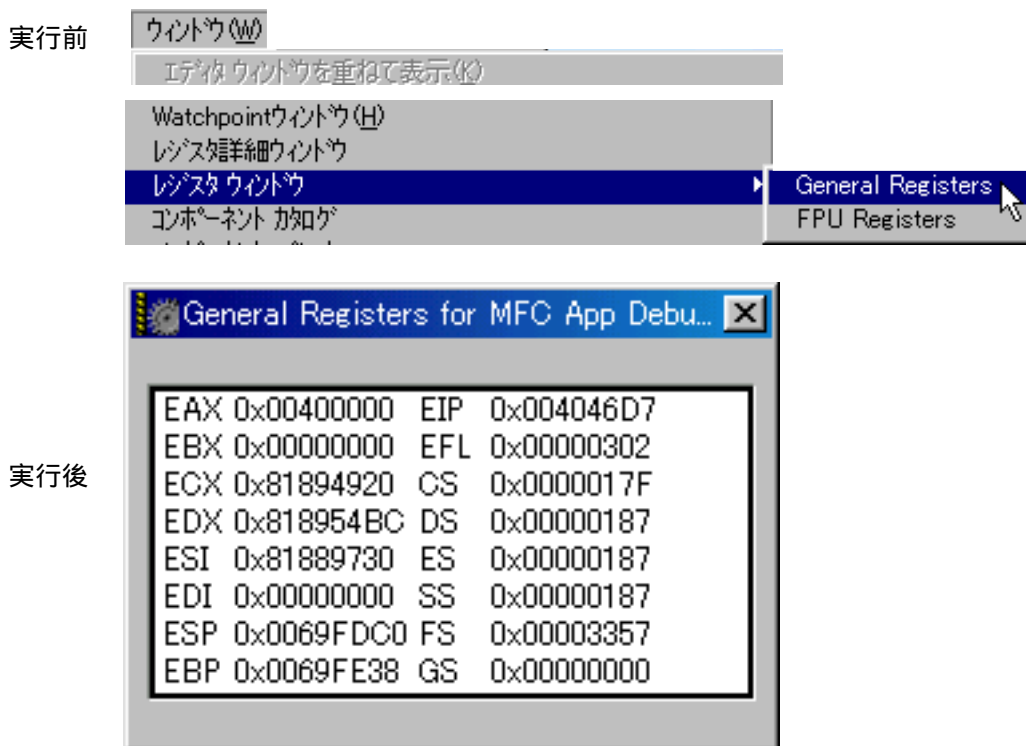


プロセッサレジスタを表示

プロセッサレジスタの内容を表示するにはウィンドウメニューの[レジスタウィンドウ]から [General Registers] を選択します (図 11.64)。

[「レジスタウィンドウ」\(p304\)](#) も参照してください。

図 11.64 汎用レジスタを表示



ソースコードの編集

デバッガから直接コードを編集することはできません。しかし、デバッガからファイルを開いてコードを変更することは可能です。ブラウザウィンドウのファイル欄で、ファイル名をダブルクリックするとエディタウィンドウが開きます。

サードパーティのエディタを使うこともできます。詳細は「[IDE その他設定](#)」(p185) を参照してください。

評価式

評価式は値を生成する計算式を表します。デバッガは、「Expressions」ウィンドウに値を表示します。ブレークポイントや Watchpoint に評価式の値を付けることもできます。デバッガは1つのステートメントを実行するたびに、すべての評価式を評価します。

評価式は、文字変数（数字と文字列）、変数、レジスタ、ポインタ、および C++ オブジェクトメンバ、さらに加減算、論理 and、等号などのオペレータにより構成されます。

評価式は、「Expressions」ウィンドウ、ブレークポイント、またはメモリウィンドウで使われます。デバッガは評価式の結果を各ウィンドウで別の用途に使用します。

この章ではデバッガがどのように評価式を扱うのかを説明します。

[評価式の翻訳](#)

[評価式を使う](#)

[評価式の例](#)

[評価式のシンタックス](#)

評価式の翻訳

ここではデバッガがどのように各ウィンドウの評価式を翻訳するのかを説明します。

[Expression ウィンドウの評価式](#)

[ブレークポイントウィンドウの評価式](#)

[メモリウィンドウの評価式](#)

Expression ウィンドウの評価式

「Expressions」ウィンドウは、評価式とその結果の値を表示します。評価式の値を見るには、「Expressions」ウィンドウに評価式を入力します。新しい評価式を入力するには、以下の手順に従ってください。

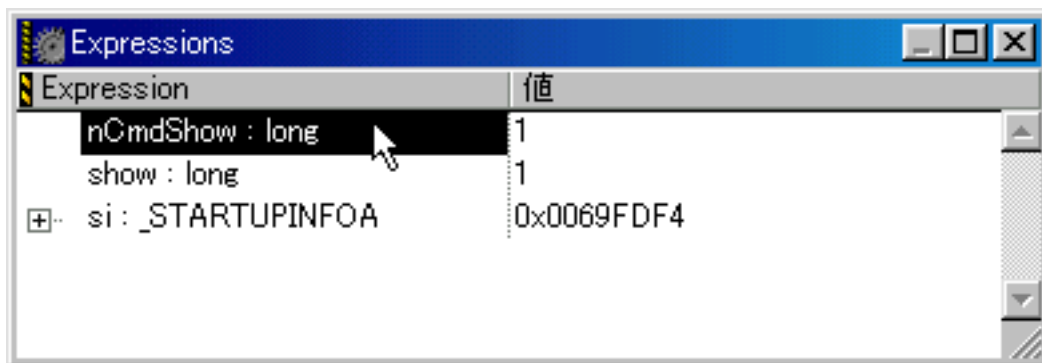
1. ウィンドウメニューの [Expression ウィンドウ] を選択する

または開いている「Expressions」ウィンドウをクリックしてアクティブにします。

2. データメニューの [新規 Expression] を選択する
3. 新しい評価式を入力して Enter キーを押す

評価式 (Expression 列) の値がその横 (値列) に表示されます (図 11.65)。他のウィンドウにある評価式を「Expressions」ウィンドウへドラッグして、新しい評価式を作成することもできます。

図 11.65 「Expressions」ウィンドウの評価式



「Expressions」ウィンドウでは、評価式はすべて数式として扱われます。デバッガは評価式の結果を論理値として扱いません (ただしブレークポイントウィンドウでは論理式として扱います)。

[「Expression ウィンドウ」\(p297\)](#) も参照してください。

ブレークポイントウィンドウの評価式

ブレークポイントウィンドウで、ブレークポイントへ評価式を付加することができます。ブレークポイントウィンドウでは、評価式は（数式ではなく）論理式として扱われます。

評価式の結果が FALSE（ゼロ）の場合、デバッガはそのブレークポイントを無視して実行を続けます。評価式の結果が TRUE（ゼロ以外の値）の場合、アクティブなブレークポイントで実行を一時停止します。ブレークポイントの設定方法については、[「ブレークポイントの設定と削除」\(p324\)](#) を参照してください。

ブレークポイントを設定した後、ブレークポイントに評価式を付けて、その評価式を条件にすることができます。

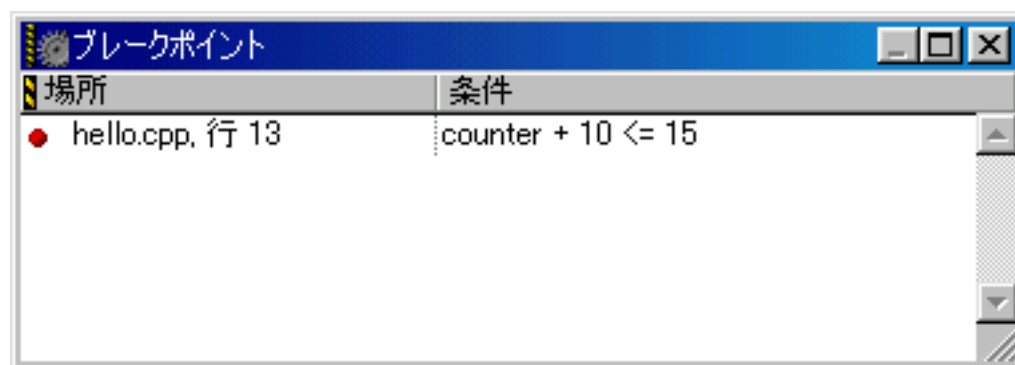
1. ウィンドウメニューの [ブレークポイントウィンドウ] を選択する

または開いているブレークポイントウィンドウをクリックしてアクティブにします。

2. 条件を設定する

ブレークポイントの条件列をダブルクリックした後、そこへ評価式を入力します（[図 11.66](#)）。他のウィンドウから評価式をドラッグ & ドロップして、評価式を追加または変更することもできます。

図 11.66 ブレークポイントウィンドウの条件評価式



条件ブレークポイントは、実行がブレークポイントに達したときに評価式の結果が TRUE（ゼロ以外）であれば、プログラムを停止します。評価式の結果が FALSE（ゼロ）であれば実行は停止せずに継続します。

[「ブレークポイントウィンドウ」\(p298\)](#)、および [「条件ブレークポイント」\(p327\)](#) も参照してください。

メモリウィンドウの評価式

メモリウィンドウでは、評価式はアドレスとして扱われます。ウィンドウ上部の [表示] フィールド内の評価式が、ウィンドウに表示するメモリの先頭アドレスを定義します。メモリウィンドウの基底アドレスを変更するには、以下の手順に従ってください。

1. データメニューの [メモリを表示] を選択する

または、開いているメモリウィンドウをクリックしてアクティブにします。

2. 新しい評価式を入力する

[表示] フィールドをダブルクリックし、新しい評価式を入力します。または、他のウィンドウから評価式をメモリウィンドウの [表示] フィールドへドラッグして、新しい評価式を作成することもできます。

新しい評価式の結果が示すアドレスを先頭アドレスとして、メモリの内容が表示されます。

[「メモリウィンドウ」\(p309\)](#) も参照してください。

評価式を使う

CodeWarrior デバッガの評価式シンタックスは、C/C++ と似ていますが、C/C++ の評価式シンタックスを少し拡張してあり若干の制限もあります。Pascal 形式の評価式もサポートしています。

[評価式の特殊機能](#)

[評価式の制限](#)

評価式の特殊機能

評価式は特別な項目を参照できます。

デバッガは `int` 型を 4 バイト長とみなします。2 バイト整数値としたい場合は `short` 型を使います。

デバッガは `double` 型を 8 バイト (64 ビット) ではなく 10 バイト (80 ビット) とみなします。

文字列を比較するには、`==` (等号) または `!=` (不等号) オペレータを使います。デバッガは、Pascal 文字列と C 文字列を区別しています。Pascal 文字列を比較するときは、文字列の先頭に `\p` を付けてください。次の評価式は、C 文字列と Pascal 文字列を比較しているので、結果は `FALSE` になります。

```
"Nov shmoz ka pop" == "\pNov shmoz ka pop"
```

評価式の制限

デバッガの評価式に適用される制限事項を以下に示します。

C/C++ プリプロセッサの定義およびマクロ（`#define` 疑似命令で定義）は使用できません。たとえそれらがソースコードの中で定義されていたとしても、評価式の中では使用できません。

副作用があるオペレータは使えません。増分オペレータ（`i++`）、減分オペレータ（`i--`）、代入（`i = j`）は使えません。

関数を呼び出しできません。

関数名や関数へのポインタは使用できません。

評価式のリストは使用できません。

C++ クラスメンバに対してのポインタは使用できません。

デバッガは、ネストされたブロックで使われている同じ名前の変数を区別できません（[例 11.4](#) を参照してください）。

例 11.4 同じ名前の変数がネストしたブロックにある（C++）

```
// The debugger cannot distinguish between x the
// int variable and x the double variable. If x is
// used in an expression, the debugger will not
// know which one to use.

void f(void)
{
    int x = 0;
    .
    .
    .
    {
        double x = 1.0;
        .
        .
        .
    }
}
```

デバッガで利用できない型の定義は、使用できません（[例 11.5](#)）。

例 11.5 評価式の型の定義（C/C++）

```
// Use long in expressions; Int32 not available
typedef long Int32;

// Use Rect* in expressions; RectPtr not available
typedef Rect* RectPtr;
```

ネストした型情報は使用できません。[例 11.6](#) ではデバッガ評価式で、`Outer::Inner` ではなく `Inner` を使います。

例 11.6 ネストした型情報 (C/C++)

```
// To refer to the i member, use Inner.i,  
// not Outer::Inner.i  
  
struct Outer  
{  
    struct Inner  
    {  
        int i;  
    };  
};
```

評価式の例

以下に入力可能な評価式の例を示します。

10 進数の定数を直接入れます。

`160`

16 進値の値は先頭に `0x` を付けて入れます。

`0xA0`

変数の値を求めます。

`myVariable`

変数を 4 ビット左へシフトした値を求めます。

`myVariable << 4`

2 つの変数の差を求めます。

`myRect.bottom - myRect.top`

2 つの数値変数のうち大きい方の値を返します。

`(foo > bar) ? foo : bar`

ポインタ変数で指されている変数の値を求めます。

`*MyTablePtr`

構造体のサイズ (コンパイル時に決定) を求めます。

`sizeof(myRect)`

ポインタ変数 `->` で指されている構造体のメンバの値を求めます。

`myRectPtr->bottom`

または

`(*myRectPtr).bottom`

オブジェクトのクラスメンバの値を求めます。

`myDrawing::theRect`

以下に論理式の例を示します。結果は、ゼロ以外を TRUE、ゼロを FALSE と解釈します。

変数が 0 であることを調べます。

```
!isDone
```

または

```
isDone == 0
```

変数がゼロでないことを調べます。

```
isReady
```

または

```
isReady != 0
```

数値変数が他の数値変数の値以上であることを調べます。

```
foo >= bar
```

数値変数が他の 2 つの数値変数より小さいことを調べます。

```
(foo < bar) && (foo < car)
```

変数の 4 ビット目が 1 であることを調べます。

```
((char)foo >> 3) & 0x01
```

C 文字列変数を C 文字列定数と比較します。

```
cstr == "Nov shmoz ka pop"
```

Pascal 文字列変数を Pascal 文字列定数と比較します。

```
pstr == "\pScram gravy ain't wavy"
```

常に TRUE です。

```
1
```

常に FALSE です。

```
0
```

評価式のシンタックス

この節では、デバッガの評価式 (Expression) のシンタックスについて説明します。

定義の最初の行は、これから定義する項目です。

字下げされている行は、その項目の定義を示します。

複数の定義を持つ項目は、それぞれの定義が 1 行ずつ表示されます。

山括弧 (<>) で括られた項目は、他で定義されています。

斜体の項目は、値またはシンボルによって置き換える項目です。

その他の項目は、すべてこのまま書かなければなりません。

例えば、

```
<name>  
    identifier  
    <qualified-name>
```

この例では、「name」を定義します。「name」は「*identifier*」または「*qualified-name*」となり得ます。後者は、ここでリストされている他の定義でシンタックスを定義されています。

以下に評価式のシンタックスをまとめます。

```

<name>
    identifier
    <qualified-name>

<typedef-name>
    identifier

<class-name>
    identifier

<qualified-name>
    <qualified-class-name>::<name>

<qualified-class-name>
    <class-name>
    <class-name>::<qualified-class-name>

<complete-class-name>
    <qualified-class-name>
    :: <qualified-class-name>

<qualified-type-name>
    <typedef-name>
    <class-name>::<qualified-type-name>

<simple-type-name>
    <complete-class-name>
    <qualified-type-name>
    char
    short
    int
    long
    signed
    unsigned
    float
    double
    void

<ptr-operator>
    *
    &

<type-specifier>
    <simple-type-name>

<type-specifier-list>
    <type-specifier> <type-specifier-list>(opt)

<abstract-declarator>
    <ptr-operator> <abstract-declarator>(opt)
    (<abstract-declarator>)

```

```
<type-name>
    <type-specifier-list> <abstract-declarator>(opt)

<literal>
    integer-constant
    character-constant
    floating-constant
    string-literal

<register-name>
    @PC
    @SP
    @Dnumber
    @Anumber

<register-name>
    @Rnumber
    @FPRnumber
    @RTOC

<register-name>
    $PC
    $SP
    $RTOC
    $Anumber
```

注意： プロセッサがターゲットにしていないレジスタは、未知のレジスタ評価式の値を表示できません。

注意： レジスタを指定する場合、*number* の範囲はターゲットプロセッサで利用可能なレジスタの数によって異なります。

```
<primary-expression>
    <literal>
    this
    ::identifier
    ::<qualified-name>
    (<expression>)
    <name>
    <register-name>

<postfix-expression>
    <primary-expression>
    <postfix-expression>[<expression>]
    <postfix-expression>.<name>
    <postfix-expression>-><name>

<unary-operator>
    *
```

```
&
+
-
!
~

<unary-expression>
    <postfix-expression>
    <unary-operator> <cast-expression>
    sizeof <unary-expression>
    sizeof(<type-name>)

<cast-expression>
    <unary-expression>
    (<type-name>)<cast-expression>

<multiplicative-expression>
    <cast-expression>
    <multiplicative-expression> * <cast-expression>
    <multiplicative-expression> / <cast-expression>
    <multiplicative-expression> % <cast-expression>

<additive-expression>
    <multiplicative-expression>
    <additive-expression> + <multiplicative-expression>
    <additive-expression> - <multiplicative-expression>

<shift-expression>
    <additive-expression>
    <shift-expression> << <additive-expression>
    <shift-expression> >> <additive-expression>

<relational-expression>
    <shift-expression>
    <relational-expression> < <shift-expression>
    <relational-expression> > <shift-expression>
    <relational-expression> <= <shift-expression>
    <relational-expression> >= <shift-expression>

<equality-expression>
    <relational-expression>
    <equality-expression> == <relational-expression>
    <equality-expression> != <relational-expression>

<and-expression>
    <equality-expression>
    <and-expression> & <equality-expression>

<exclusive-or-expression>
    <and-expression>
    <exclusive-or-expression> ^ <and-expression>

<inclusive-or-expression>
    <exclusive-or-expression>
    <inclusive-or-expression> | <exclusive-or-expression>
```



```
<logical-and-expression>
  <inclusive-or-expression>
  <logical-and-expression> && <inclusive-or-expression>

<logical-or-expression>
  <logical-and-expression>
  <logical-or-expression> || <logical-and-expression>

<conditional-expression>
  <logical-or-expression>
  <logical-or-expression> ? <expression> : <conditional-
expression>

<expression>
  <conditional-expression>
```

トラブルシューティング

ここでは、CodeWarrior デバッガについてよく質問される内容およびその解決方法について説明します。

[一般的な問題](#)

[デバッガ起動時の問題](#)

[デバッガ実行時の問題 / クラッシュ](#)

[ブレークポイントの問題](#)

[変数の問題](#)

[ソースファイルの問題](#)

一般的な問題

ここで紹介する解決法方が役に立たない場合もあるかもしれません。その場合は以下の操作を試してみてください。

簡単に生成できるファイルとデータ (`.(x)sym` ファイル、`.dbg` ファイル、プリファレンス、バイナリを含む) をプロジェクトから除去してください (プロジェクトメニューの [オブジェクトを削除] を選択してください。詳細は [「オブジェクトを削除」\(p261\)](#) を参照してください)。

ハードディスクに新しいバージョンの IDE をインストールしてください。

デバッガ起動時の問題

ここではデバッガの起動時に発生する問題について説明します。

デバッガが起動しない

問題

プロジェクトメニューで [デバッガを有効にする] を選択しているのに、アプリケーションを実行してもデバッガが起動しない。

CodeWarrior のプロジェクトメニューの [実行] または [デバッグ] コマンドが薄く表示され、選択できない。

背景

CodeWarrior IDE からデバッガを自動的に起動できるのは、プロジェクトがアプリケーションである場合のみです。デバッガが有効であれば、デバッグ情報を生成するようにプロジェクトが準備されます。

解決方法

プロジェクトがアプリケーションを生成するかどうかを確認してください。[実行] コマンドはアプリケーションを作成している場合のみ使用可能です。

デバッガが有効であるかどうかを確認してください。プロジェクトメニューの [デバッガを有効にする] を選択してデバッガを有効にしてください。メニューコマンドが [デバッガを無効にする] と表示されている場合、デバッガは既に有効です。

デバッグ情報の生成プロジェクトに設定してください。[「デバッグの準備」\(p278\)](#) を参照してください。

[デバッグ] コマンドが働かない

問題

[デバッグ] コマンドを選択しても何も起こらない。

背景

デバッガを実行するためには設定を正しく行わなくてはなりません。またコードが実際に何かを行っているかを確認してください。

解決方法

デバッガが有効になっているかどうかを確認してください。[「デバッグの準備」\(p278\)](#) を参照してください。

IDE のリリースノートを読んで、サードパーティのソフトウェアとの互換性を確認してください。

デバッガ実行時の問題 / クラッシュ

ここではデバッガの実行中に起こる問題について説明します。

プロジェクトをデバッグなしで実行するとクラッシュする

問題

デバッグのコントロール下ではプロジェクトは正常に動作するが、デバッグなしでプログラムを実行するとクラッシュする。

背景

デバッグ上での実行は、プログラムの動作環境であるオペレーティング環境を変更してしまいます。正確に動作する、バグのあるプログラムを作成してしまう可能性もあります。何が問題なのかを特定することは困難ですが、以下に奇妙な挙動の原因となりうる事柄を記します。

デバッグは実行速度を低下させます。時間に敏感なコードがある場合、デバッグなしでは実行速度が速すぎるかもしれません。これを頭に入れて考えてください。

プロジェクトのメモリ管理もデバッグによって影響されます。デバッグの動作中にメモリブロックを移動することはできません。デバッグが動作していない場合、ブロックは移動可能になり、メモリの再配置によるバグが発生します。

解決方法

レース状況などの時間に敏感なコードに注目してください。

メモリに関係する問題に注意してください。ヌルポインタやハンドルへのアクセス、リソースハンドルの不正な処理、複数回のハンドルの処理などです

ブレイクポイントの問題

ここではブレイクポイントの設定、消去に関する問題について説明します。

ステートメントにブレイクポイントを設定できない

問題

ステートメントのブレイクポイント列にダッシュ (-) がなく、ブレイクポイントを設定できない。

背景

CodeWarrior リンカは、最終的な製品にリンクされないソースコードのシンボル情報は生成しません。実際に使用されないステートメントは最終的なオブジェクトコードにインクルードされません。このようなステートメントにはオブジェクトコードがないため、ブレイクポイントを設定することはできません。

コードの最適化はオブジェクトコードを並べ換えるため、オブジェクトコードとソースコードの対応関係を乱します。このためブレイクポイントを正確に設定することは非常に困難になります。

解決方法

コードがすべて使用されているかを確認してください。必要ならばソースコードを変更してください。

コンパイラ疑似命令のせいでコンパイラがステートメントを無視していないか、ソースコードを見て確認してください。

すべてのコンパイラ最適化をオフにしてプロジェクトを再度ビルドしてください。最適化によってオブジェクトコードが変更され、ソースコードと対応しなくなることがあります。最適化をオフにすると、ステートメントにブレークポイントマーカが表示されます。[「ブレークポイントでコードを最適化」\(p329\)](#) を参照してください。

1 行のソースには 1 つのステートメントを入れるようにコードを書いてください。コンパイラは 1 行に複数のステートメントがあってもブレークポイント情報を出力できます。しかしデバッガはソース行を表示するだけなので、同じ行を複数回ステップ実行してしまいます。

ブレークポイントが反応しない

問題

設定したブレークポイントが働かない。

背景

ブレークポイントがアクティブであれば、条件(もしあれば)が TRUE である場合、ブレークポイントに達すると実行を停止します。

解決方法

コードをステップ実行して、ブレークポイントを設定したステートメントに達しているかを確認してください。[「ブレークポイントの設定と削除」\(p324\)](#) を参照してください。

ブレークポイントウィンドウでブレークポイントがアクティブであるか、確認してください。

ブレークポイントが条件付きである場合、それが TRUE であることを確認してください。デバッガは条件が FALSE のブレークポイントを無視します。[「条件ブレークポイント」\(p327\)](#) を参照してください。

変数の問題

ここでは変数に関係のある問題について説明します。

変数が変化しない

問題

変数に値を割り当てたが、デバッガで値が変化しない。

背景

コンパイラはコード内で使用されない変数を削除します。

解決方法

未使用の変数をコードから除去してください。

変数を使用するようにコードを修正してください。

変数に謝った値が割り当てられる

問題

変数の変化が不正確である。以下のいずれかの問題に遭遇している。

2 つ以上の変数に対して、同時に同じ値が設定される。

ある変数が、他の変数に割り当てられるべき値を受け取っている。

背景

コンパイラは変数が同時に使用されていないことを識別し、同じ場所に変数を格納しています。これはコンパイラが自動的に行う最適化で、「レジスタカラーリング」と呼ばれます。レジスタカラーリングは関数内で変数がどのように使われているかを調べます。2 つ以上の変数が同じ有効範囲内にあり、かつ同時に使われない場合、コンパイラはそれらの変数に対して同じプロセッサレジスタを使います。複数の変数に、メモリではなくレジスタを使うことによってプログラムのパフォーマンスが改善されます。

[例 11.7](#) はレジスタカラーリングを誘発するコードの例です。異なる 4 つの変数が設定されていますが、同時には使用されません。コンパイラは 4 つの変数に同じレジスタを使います。4 つの変数が同じレジスタにあることはデバッガにはわからないため、4 つの変数すべてが変化していることを表示します。

例 11.7 レジスタカラーリングで変化する変数

```
void main(void)
{
    long a = 0, b = 0, c = 0, d = 0;

    a = 1; /* a is set to 1 */
    a = 2; /* a is set to 2, b remains unchanged */
    a = 3; /* a is set to 3, c remains unchanged */
    a = 4; /* a is set to 4, d remains unchanged */
}
```

解決方法

そのままにしておきます。レジスタカラーリングは問題ではありません。

C/C++ でレジスタカラーリングを防ぐには、`volatile` キーワードで変数を宣言します。これはプリプロセッサ疑似命令で行ってください。そうするとデバッグの後に `volatile` 格納クラスの指定子を簡単に除去することができます。『C Compilers Reference』、『Assembler Reference』を参照してください。

奇妙な変数の名前

問題

大域変数欄、変数欄に、ソースコードで宣言していない変数が表示される。

背景

コンパイラはソースコードを翻訳するときに、オブジェクトコードの中に独自のテンポラリ変数を生成することがあります。このようなテンポラリ変数はドルマーク (\$) 付きの名前で表示されます。また C++ の仮想基底クラス型も \$ の接頭子付きで表示されます。

コンパイラとリンカはプログラムの初期化と終了を支援するため、ライブラリおよびランタイム関数から変数を追加することがよくあります。

解決方法

ありません。これは解決が必要な問題ではありません。

奇妙なデータ型

問題

データメニューの[型を表示]を選択したとき、enum 型の値が ?anonx 型 (x は任意の数) のように表示される。

背景

enum 型の名前がソースコードで定義されていなければ、デバッガはその名前を表示できません。コンパイル時に、コンパイラは enum 型に generic 型の名前を割り当てます。これはデバッガが表示可能な一般的な名前です。

例えば[例 11.8](#)で、[型を表示]を選択すると、変数 myMarx は enum 型 ?anonx を持つかのように表示されます。これは enum 型に名前がないためです。一方変数 myBeatle は Beatle 型として表示されます。enum 型がその名前で定義されているためです。

例 11.8 名前のない enum 型 (C/C++)

```
// Debugger displays as anonymous type
enum {Groucho,
      Harpo,
      Chico,
      Zeppo } myMarx = Harpo;

// Debugger displays as type Beatle
typedef enum Beatle {John,
                    Paul,
                    George,
                    Ringo} myBeatle = John;
```

解決方法

ありません。これは解決が必要な問題ではありません。

認識されないデータ型

問題

独自のデータ型を宣言したが、その型で変数を見ることができない。

背景

シンボルファイルにはプログラムで使用される型の情報だけが含まれます。typedef で定義された型はシンボルファイルには格納されません。このため、派生した型のように変数を表示する必要があります。例えば、long データ型に基づいて MyLong 型を宣言した場合、(MyLong ではなく) long 型として表示されます。[「異なる型でデータを表示」\(p337\)](#)を参照してください。

解決方法

基本のデータ型を使ってください。

データメニューの [型を表示] を選択し、デバッガがどのように型を識別しているかを調べてください

Expression ウィンドウの未定義の識別子

問題

「Expressions」ウィンドウの評価式にあるユーザー定義型の値が

- undefined identifier • になる。

背景

デバッガは、単純に他の型のエイリアスであるデータ型を認識できません。エイリアス型はシンボルファイルに含まれないためです。

パスカル形式の宣言を例にしてみます。

```
TYPE
    MYBIGINT = LONGINT;
```

評価式は

```
MYBIGINT(thePtr)
```

デバッガの「Expressions」ウィンドウには値が • undefined identifier • として表示されます。結果を修正するためには次の評価式を使ってください。

```
LONGINT(thePtr)
```

[「評価式の制限」\(p346\)](#)を参照してください。

解決方法

定義したデータ型ではなく、オリジナルのデータ型を使ってください。

ソースファイルの問題

ここではソースファイルに関係のある問題について説明します。

ソースコードが表示されない

問題

ソース欄にアセンブリ言語のコードしか表示されない。ソースポップアップメニューからもソースコードを表示できない。

背景

そのコードのシンボル情報がありません。ファイルに対してデバッグ機能をオンにしている、またはリンカが追加したソースコードと対応しない補助的なコード（グルーコードなど）を実行している可能性があります。シンボル情報がなければ、デバッガはアセンブラコードしか表示できません。

解決方法

コードが独自のファイルにある場合、そのファイルのシンボル情報が生成されているかを確認してください。[「デバッグのためにファイルを設定」\(p279\)](#) を参照してください。

他のソース（コンパイルされたライブラリなど）のコードでは、関数からステップアウトして呼び出し元へ戻ってください。表示するソースコードはありません。

ソースファイルの修正日時

問題

プロジェクトを実行すると、「修正日時が一致しない」という警告メッセージが表示される。

背景

シンボルファイルはソースファイルが最後にいつ変更されたかを追跡しています。シンボルファイルに保存された修正日時がオリジナルのファイルのものと一致しなければ、デバッガはシンボル情報が古いという警告を発します。

解決方法

ソースファイルをタッチ（または変更を加えずにメイクして保存）してからプロジェクトを再度ビルドするか、ファイルをアップデートしてください。

Pascal プロジェクトの ANSI C コード

問題

Pascal で書いたコードをステップ実行すると ANSI C 関数が見つかった。ANSI C ライブラリもインクルードしていない。

背景

Pascal のランタイムライブラリは C で記述されており、ANSI C 関数を使っています。Pascal プログラムをデバッグするときにこれらの関数が見つかります。

解決方法

ありません。これは解決が必要な問題ではありません。

デバッガのエラーメッセージ

以下にデバッガのエラーメッセージの一覧およびエラーの発生原因のヒントを示します。

An unknown error occurred while trying to target an existing process.

既存のプロセスをターゲットにしようとしたときに未知のエラーが起きました。

Bad type code

内部エラーです。

Bus Error

不当なアドレスを読み書きしようとしています。

can't display value -- type information not supported

シンボルファイルに CodeWarrior デバッガがサポートしていないデータ型が含まれています。

Can't use this source file, it was not saved before running, or was edited after linking.

コンパイラが扱っている最中のファイルをデバッガがアクセスすることはできません。デバッグ情報が存在しないテキストを参照しているのでなければ、デバッガは警告を発するだけです。参照していれば、このエラーメッセージが表示されます。

class name expected

評価式を評価中に、クラス名が想定される部分にそれ以外のものが見つかりました。

Could not complete your request because the process is not suspended.

プログラムを実行しているときに、該当コマンドを実行することはできません。

Could not set a watch point because the page containing that memory location overlaps low memory or the system heap.

Watchpoint は、ローメモリやシステムヒープに設定できません。

Could not set a watch point because the page containing that memory location overlaps the stack.

Watchpoint はメモリのライトプロテクションメカニズム（ページレベルの操作）に基づいて実装されています。スタックを含むページメモリへの書き込みをプロテクトすることはできません。

Couldn't locate the program entry point, program will not stop on launch.

プログラムを起動するとき、デバッガは暗黙のブレークポイントを `main()` (C/C++) 関数、または `main` プログラム (Pascal) の前に設定します。そのような部分がないときは、停止せずにただ実行するだけです。

identified or qualified name expected

評価式を評価中に、識別子または定義済みの名前以外のものが見つかりました。

illegal character constant

評価式を評価中に、不当な文字定数が見つかりました。

illegal string constant

評価式を評価中に、不当な文字列定数が見つかりました。

illegal token

評価式を評価中に、不当なトークンが見つかりました。

Invalid C or Pascal string.

評価式を評価中に、評価できない文字列が見つかりました。

Invalid character constant.

評価式を評価中に、評価できない文字定数が見つかりました。

Invalid escape sequence inside string or character constant.

文字列中の C/C++ のエスケープシーケンスは無効なシンタックスです。

invalid pointer or reference expression

評価式を評価中に、評価できないポインタまたは参照式が見つかりました。

invalid type declaration

評価式を評価中に、評価できない型が見つかりました。

invalid type information in SYM file

シンボルファイルに含まれるデータが不当なので、CodeWarrior デバッガは変数を表示できません。

New variable value is too large for the destination variable.

例えば、10 バイトの文字列変数に 20 バイトの文字列を割り当てようとした。

No type with that name exists.

CodeWarrior デバッガは「表示を変更」ダイアログで指定した型を認識しません。

Register not available

デバッガはレジスタ変数を表示するために有効なレジスタを取得できません。例えば、カレントルーチンからスタック上にあるルーチンを探すとき、(スタックの下にあるルーチンがデバッグ情報を持たない限り) デバッガは保存しているレジスタの値を見ることができません。

string too long

文字列長が長すぎ、最大長を越えています。

The new variable value is the wrong type for the destination variable.

文字列に整数値など、変数に間違った型の値を割り当てようとしています。

typedef name expected

評価式を評価中に、`typedef` 名を想定しているのに別のものが見つかりました。

Unable to step from here.

デバッガはこのポイントからステップ実行できません。

Unable to step out from here.

デバッガはこのポイントからステップアウトできません。

undefined identifier

評価式を評価中に、定義されていない識別子が見つかりました。

unexpected token

評価式を評価中に、予想しないトークンが見つかりました。

unknown error "^0"

内部エラーです。

unterminated comment

コメントが終了していません。

Variable or expression cannot be used as an address.

例えば、`r` が `short` 型るとき、`*(char*)r` は無効です。

Warning - this SYM file has some invalid or inconsistent data. The debugger may show incorrect information.

シンボルファイルが壊れている可能性があります。

'*' or '&' expected

評価式を評価中に、予想しないポインタまたは参照オペレータが見つかりました。

第 12 章 RAD デザインとレイアウト

この章では CodeWarrior の Java 用 RAD (Rapid Application Development) ツールについて説明します。RAD 開発の利点、CodeWarrior プロジェクトにおけるデザインやレイアウトの概念についても説明します。

以下の項目について説明します。

[RAD とは](#)

[CodeWarrior RAD ツール](#)

[RAD プロジェクトを作成](#)

[RAD ウィザード](#)

[デザインを作成](#)

[レイアウトを作成](#)

RAD とは

従来のアプリケーションプログラミングには膨大な手作業が必要です。新しいアプリケーションを作るとき、プログラムのあらゆる箇所のコードを書かなくてはなりません。初期化ルーチン、プログラムの相互関係やユーザーインターフェースなどを指定しなくてはなりません。

RAD ツールはこれらの作業を自動化し、アプリケーション開発に集中できるように支援するツールです。RAD ツールは初期化ルーチン进行处理し、ユーザーインターフェースの要素を画面の上に書き、自動的にコードを生成します。またアプリケーションに合わせて生成したコードを修正することもできます。

RAD ツールは開発環境の一部として CodeWarrior に含まれています。アプリケーション、アプレット、プラグインアーキテクチャのコンポーネント開発などをサポートします。IDE と既存の RAD プラグインを組み合わせて使うことも、または独自のツールを作ることもできます。複数の言語、プラットフォーム、アプリケーションフレームワークをサポートします。

CodeWarrior RAD ツール

RAD ツールはアプリケーションを視覚的に構築します。IDE に含まれる RAD ツールは Java 用です。他のアプリケーションプログラミングフレームワークも今後サポートする予定です。

以下の項目について説明します。

[クラスオーサリング](#)

[コンポーネントモデル](#)

クラスオーサリング

RAD ツールは高レベルなビジュアルインターフェースを用いてクラスを作成します。このインターフェースを利用してさまざまなクラス、メンバ関数、アプリケーションのデータメンバを作成、命名、削除することができます。アプリケーションが修正されるたびに、RAD ツールは自動的に宣言と定義の同期を取ります。

CodeWarrior は RAD にプラグインアーキテクチャを使用しているため、IDE の機能を簡単に拡張することができます。RAD ツールは自動的にスクリプトからソースコードを生成します。既存のスクリプトを修正して、アプリケーションのために生成されたコードをカスタマイズすることもできます。

コンポーネントモデル

CodeWarrior の RAD ツールはアプリケーションを作成するためにコンポーネントモデルを使います。RAD におけるコンポーネントとは、自己記述オブジェクトです。コンポーネントは、アプリケーションを作成するためのプログラミングフレームワークにおけるオブジェクトに相当します。属性、メソッド、オブジェクトを記述するイベントはコンポーネントに含まれます。

コンポーネントは、各種プラットフォームにパワフルで使いやすいアプリケーションユーザーインターフェースを提供します。RAD ツールはこれらのコンポーネントを使い、持続的な有効性と自動化を実現しています。

CodeWarrior は以下のツールにコンポーネントモデルを実装しています。

[レイアウトエディタ](#)

[カタログ](#)

レイアウトエディタ

レイアウトエディタは RAD ツールのもっとも重要なパートです。レイアウトエディタは、ビジュアル開発環境を使うアプリケーションビルダーです。ユーザーインターフェースの要素はコンポーネントとして視覚的に表示されます。コンポーネントをドラッグ&ドロップして、アプリケーションを作成することができます。変更を加えるたびに IDE はレイアウトエディタの画面を更新し、生成したソースコードも修正します。

カタログ

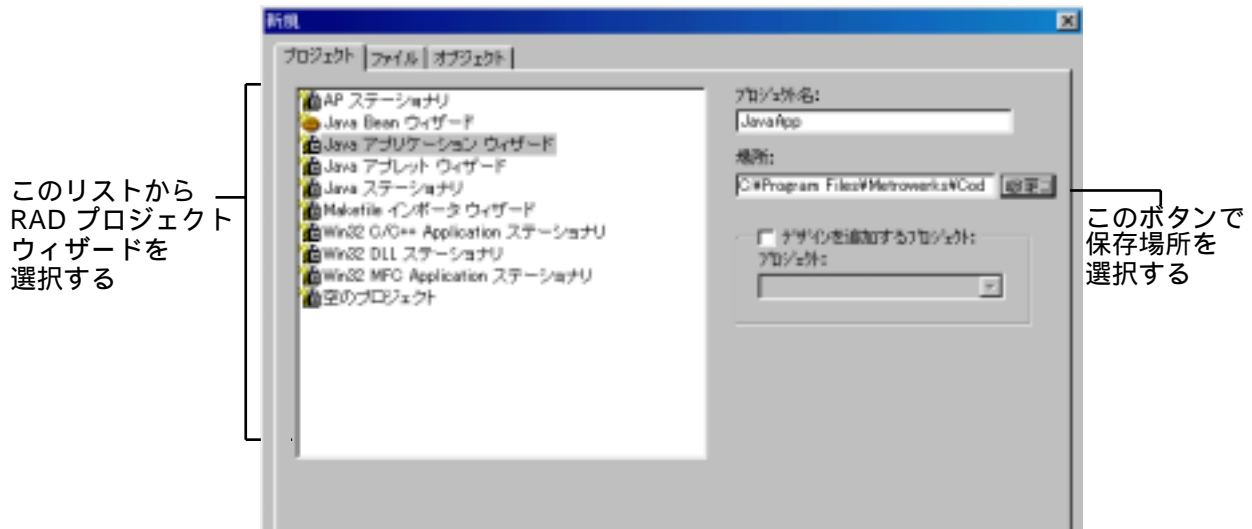
カタログは RAD ツールの 2 番目に重要なパートです。カタログはアプリケーションに利用できるコンポーネントの集合です。IDE は使用中のプログラミングフレームワークに基づいてカタログを表示します。Java アプリケーションを作成している場合、Java 用カタログからコンポーネントを選択します。カタログにあるコンポーネントの属性とイベントをカスタマイズすることができます。

RAD プロジェクトを作成

CodeWarrior の RAD プロジェクトを作成するには、ファイルメニューの [新規] を選択します。「新規」ダイアログ (図 12.1) が現れます。

「新規」ダイアログの [プロジェクト] タブをクリックするとプロジェクトパネルが表示されます。プロジェクトパネルは新規プロジェクトを作成するときに利用可能なステーションナリとウィザードをリスト表示します。リストから項目を選択すると、それに関するオプションがパネル右側に表示されます。

図 12.1 「新規」ダイアログ



新規 RAD プロジェクトを作るには、リストから以下のいずれかを選択してください。

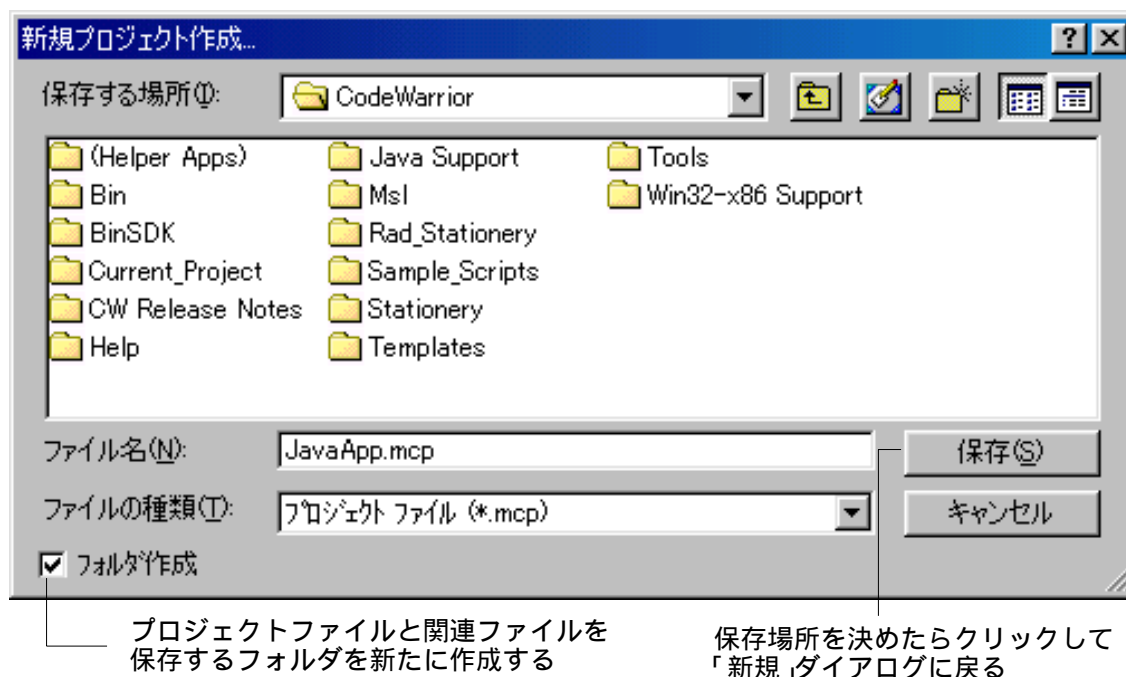
[Java アプレットウィザード](#) : Java アプレットを作成します。アプレットのクラス、HTML ページ、パラメータ、記述についての情報を設定します。

[Java アプリケーションウィザード](#) : Java アプリケーションを作成します。アプリケーションのクラス、フレームクラス、記述についての情報を設定します。

[Java Bean ウィザード](#) : Java Bean を作成します。Java Bean のクラス、位置、パッケージ、修飾子についての情報を設定します。

プロジェクトのウィザードを選択した後、新規プロジェクトの名前を [プロジェクト名] フィールドに入力します。

図 12.2 名前を付けて保存ダイアログ



ヒント： プロジェクトには拡張子 `.mcp` を付けてください（例：MyProject.mcp）。これによりプロジェクトファイルの見分けが簡単になります。Windows 版の CodeWarrior はこの拡張子でプロジェクトファイルを識別しています。

[場所] フィールドはプロジェクトを保存するフォルダへの完全なパスを表示します。パスを変更するには、フィールドへ新しいパスを入力します。または [保存] ボタンをクリックしてダイアログ (図 12.2) を表示し、ここで保存するフォルダを指定することもできます。プロジェクトファイルとそれに関連するファイルをすべて保存するフォルダを作成するには、[フォルダを作成] チェックボックスをオンにします。次に [保存] ボタンをクリックしてウィンドウへ戻ります。

警告！ 新規プロジェクトを保存するとき、ハードディスクに同じ名前のプロジェクトが既にあると、IDE はエラーメッセージを表示します。新規プロジェクトにはユニークな名前を付けてください。

RAD プロジェクトの名前の入力と、プロジェクトを保存する場所の選択を終えたら、「新規」ダイアログの [OK] ボタンをクリックします。プロジェクトパネルで選択したウィザードが起動します。このウィザードがプロジェクトの作成手順を指示します。各ウィザードの詳細は「[RAD ウィザード](#)」(p369) を参照してください。

RAD ウィザード

新しいRADプロジェクトやデザインを作成するとき、RAD ウィザードを選択します(「[RAD プロジェクトを作成](#)」(p367)、「[プロジェクトへデザインを追加](#)」(p383) 参照)。ウィザードは作成作業を支援します。ここでは以下の RAD ウィザードについて説明します。

[Java アプレットウィザード](#)

[Java アプリケーションウィザード](#)

[Java Bean ウィザード](#)

初心者： 各ウィザードは基本的なプログラミング言語を理解している方を対象に作られています。例えば Java アプリケーションウィザードを使う場合、AWT (Abstract Window Toolkit) フレームと Swing フレームの違いを理解しておく必要があります。ウィザードが表示する項目がよくわからないときは、プログラミングの参考書などをお読みください。

RAD ウィザードには以下のナビゲーションボタンがあります。

戻る：前のページへ戻る

次へ：次のページへ進む

完了：現在の情報を表示する

キャンセル：すべての変更を破棄する

各ウィザードは一連のページを表示します。順番に従って各ページで情報を設定し、終了したら [次へ] ボタンをクリックします。

各ページのデフォルト設定を変更することもできます。現在のウィザードの設定を適用するには [完了] ボタンをクリックします。ウィザードは新規プロジェクトの現在の設定を表示します。現在の設定を適用して新しい RAD アイテムを作るには [作成] ボタンをクリックします。ウィザードに戻って設定の変更を続けるには [キャンセル] ボタンをクリックします。

Java アプレットウィザード

Java アプレットのウィザードでは以下のステップを実行します。

1. [Java アプレット用のアプレットクラスを記述する](#)
2. [Java アプレット用の HTML ページを指定する \(オプションル\)](#)
3. [Java アプレットのパラメータを作る](#)
4. [Java アプレットに情報を追加する](#)

以下は Java アプレットウィザードを使う手順です。

1. Java アプレット用のアプレットクラスを記述する

Java アプレットウィザードのページ 1 ([図 12.3](#)) では、クラス名、パッケージ名、新規アプレットの保存場所を指定します。アプレット記述のためのオプションもあります。

図 12.3 Java アプレットウィザード：ページ 1



このページには以下のオプションがあります。

[クラス名](#)

[パッケージ名](#)

[場所](#)

[スタンドアローンアプリケーションとして実行可能](#)

[標準メソッドを生成](#)

[HTML ページを生成](#)

[AWT アプレット](#)

[Swing アプレット](#)

クラス名

[クラス名] フィールドにアプレットのクラス名を入力します。サンプル名が表示されます。

パッケージ名

必要であれば [パッケージ名] フィールドにアプレットクラスのパッケージ名を入力します。サンプル名が表示されます。

場所

[場所] フィールドに新規アプレットのアプレットクラスを保存する場所を入力します。横のボタンをクリックしてダイアログを開き、そこから場所を選択することもできます。有効な場所は厳密なファイル名とパッケージルートによって変わります。

スタンドアローンアプリケーションとして実行可能

[スタンドアローンアプリケーションとして実行可能] チェックボックスがオンの場合、他のアプリケーションから呼び出さなくてもアプレットを操作できます。

標準メソッドを生成

[標準メソッドを生成] チェックボックスがオンの場合、新規アプレットはインターフェースの標準メソッド(`show()` や `dispose()` など)を生成します。オフの場合、生成しません。

HTML ページを生成

[HTML ページを生成] チェックボックスがオンの場合、新規アプレットをテストするための HTML ページを生成します。[HTML ページを生成] チェックボックスがオンの場合のみ、Java アプレットウィザードは HTML ページを表示します。チェックボックスのデフォルトはオンです。

AWT アプレット

[AWT アプレット] ラジオボタンをクリックすると新規アプレットを AWT アプレットとして生成します。デフォルトはオンです。

Swing アプレット

[Swing アプレット] ラジオボタンをクリックすると新規アプレットを Swing アプレットとして生成します。

2. Java アプレット用の HTML ページを指定する (オプションル)

ページ 2 ([図 12.4](#)) は [HTML ページを生成] チェックボックスをオンにしたときのみ表示されます。[HTML ページ] 欄で新しいアプレットをテストする HTML (Hypertext Markup Language) ページを指定します。

図 12.4 Java アプレットウィザード : ページ 2



以下のオプションがあります。

[タイトル](#)

[アプレット名](#)

[コードベース / アーカイブ](#)

[アーカイブを使用 \(JAR\)](#)

[横幅](#)

[高さ](#)

[水平余白](#)

[垂直余白](#)

[配置](#)

タイトル

[タイトル] フィールドに生成する HTML ページのタイトルを入力します。サンプル名が表示されます。

アプレット名

[アプレット名] フィールドに入力したアプレット名についての情報が HTML ブラウザに表示されます。サンプル名が表示されます。

コードベース / アーカイブ

必要であれば [コードベース / アーカイブ] フィールドに新規アプレットで使用するコードベースかアーカイブを指定します。サンプル名が表示されます。

アーカイブを使用 (JAR)

[アーカイブを使用 (JAR)] チェックボックスがオンの場合、新規アプレットで Java アーキテクチャ (JAR) を使用できます。[コードベース / アーカイブ] フィールドの名前に拡張子 `.jar` が追加されます。

横幅

[横幅] フィールドで HTML ページ内のアプレットの横幅をピクセル単位で指定します。

高さ

[高さ] フィールドで HTML ページ内のアプレットの高さをピクセル単位で指定します。

水平余白

[H 余白] フィールドで HTML ページ内のアプレットの左右の余白をピクセル単位で指定します。

垂直余白

[V 余白] フィールドで HTML ページ内のアプレットの上下の余白をピクセル単位で指定します。

配置

[配置] ポップアップメニューで HTML ページ内におけるアプレットの配置方法を決めます。

absbottom : ページの完全な下に揃える

absmiddle : ページの完全な中心に揃える

baseline : ページのベースラインに揃える

bottom : ページの下に揃える

left : ページの左側に揃える

middle : ページの中心に揃える (デフォルト)

right : ページの右側に揃える

texttop : ページのテキストの上に揃える

top : ページの上に揃える

3. Java アプレットのパラメータを作る

ページ 3 ([図 12.5](#)) では新規アプレットのための引数を作成します。パラメータには以下のフィールドがあります。

名前 : パラメータの名前

型 : パラメータのデータ型

初期値 : パラメータの初期値

変数 : パラメータを表す変数

説明 : パラメータの簡単な説明

図 12.5 Java アプレットのウィザード：ページ 3



アプレットに別のパラメータを追加するには、[追加] ボタンをクリックします。ウィザードがサンプル用のパラメータを作成します。

パラメータを削除するには、選択して [削除] ボタンをクリックします。

サンプルの <New> パラメータを変更することができます。フィールドの値を変えるには、ダブルクリックしてから新しい値を入力し、Enter キーを押します。

[型] フィールドのポップアップメニューからデータ型を選択します。

4. Java アプレットに情報を追加する

ページ 4 ([図 12.6](#)) ではアプレットの追加情報を指定します。

図 12.6 Java アプレットウィザード：ページ 4



以下のオプションがあります。

[アプレットタイトル](#)

[説明](#)

[すべてのファイルに情報を含める](#)

アプレットタイトル

[アプレットタイトル] フィールドに新規アプレットのタイトルを入力します。

説明

必要であれば [説明] フィールドに新規アプレットの内容を入力します。

すべてのファイルに情報を含める

[アプレットタイトル] [説明] フィールドの情報を各ファイルの最初に追加するには、[すべてのファイルに情報を含める] チェックボックスをオンにします。デフォルトはオンです。

5. [完了] ボタンをクリック

現在の設定を適用して新しい RAD アイテムを作るには [生成] ボタンをクリックします。ウィザードに戻って設定の変更を続けるには [キャンセル] ボタンをクリックします。

Java アプリケーションウィザード

Java アプリケーションウィザードでは以下のステップを実行します。

1. [Java アプリケーション用のアプリケーションクラスを記述する](#)
2. [Java アプリケーションのフレームクラスを指定する](#)

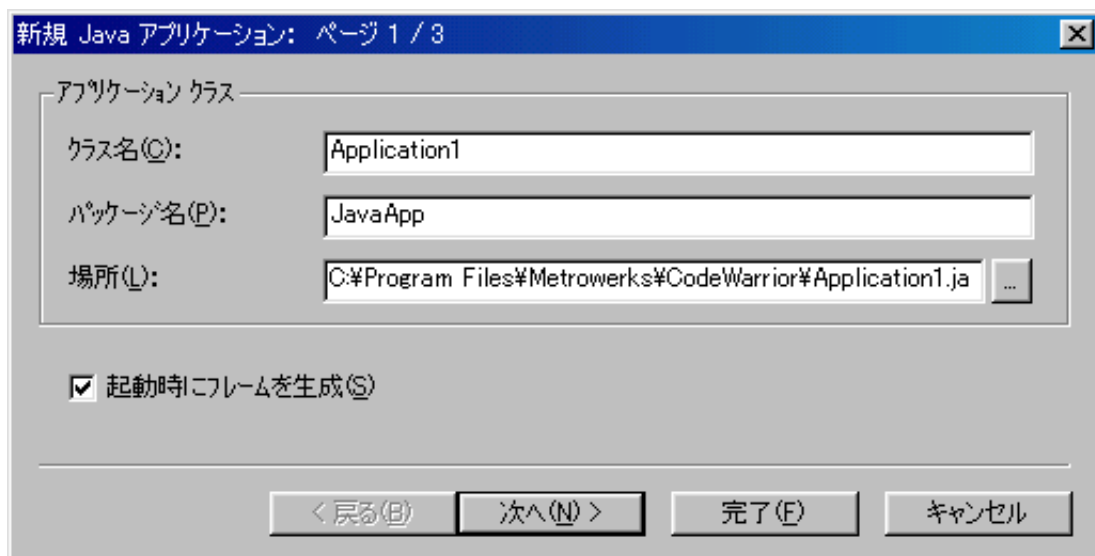
3. [Java アプリケーションの情報を追加する](#)

以下は Java アプリケーションウィザードを使う手順です。

1. Java アプリケーション用のアプリケーションクラスを記述する

Java アプリケーションウィザードのページ 1 ([図 12.7](#)) では、Java アプリケーションのアプリケーションクラスの情報を指定します。

図 12.7 Java アプリケーションウィザード：ページ 1



以下のオプションがあります。

[クラス名](#)

[パッケージ名](#)

[場所](#)

[起動時にフレームを作成](#)

クラス名

[クラス名] フィールドにアプリケーションのクラス名を入力します。サンプル名が表示されます。

パッケージ名

[パッケージ名] フィールドに (必要であれば) アプリケーションクラスのパッケージ名を入力します。サンプル名が表示されます。

場所

[場所] フィールドに新規アプリケーションのアプリケーションクラスを保存する場所を入力します。横のボタンをクリックしてダイアログを開き、そこから保存場所選択することもできます。

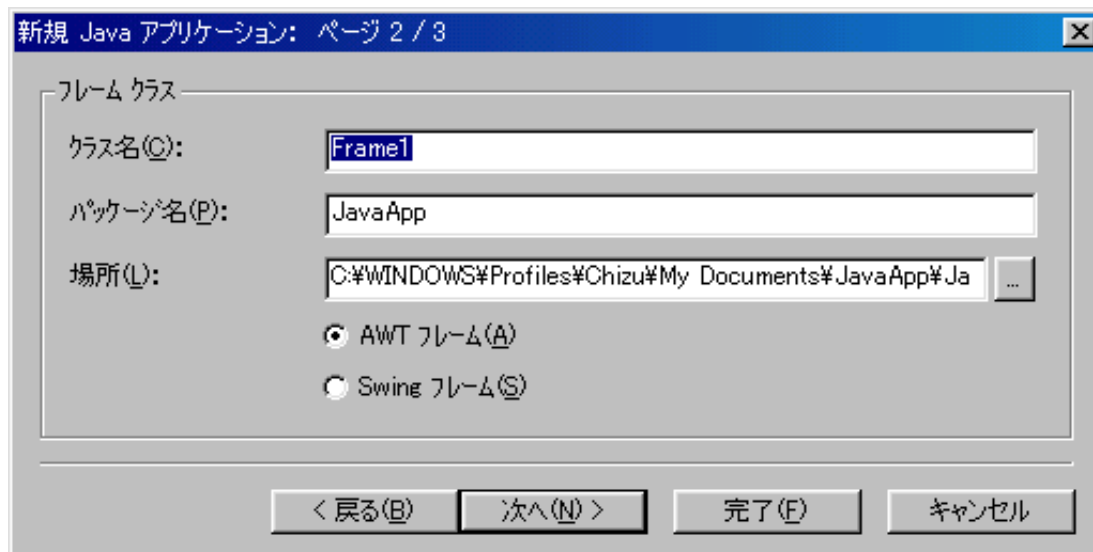
起動時にフレームを作成

[起動時にフレームを作成] チェックボックスがオンの場合、IDE は自動的に新規レイアウトフレームを生成します。また自動的に[レイアウトエディタ](#)、[コンポーネントパレット](#)、[オブジェクトインスペクタ](#)を表示します。オフの場合、これらのツールは表示されません。

2. Java アプリケーションのフレームクラスを指定する

ページ 2 ([図 12.8](#)) で Java アプリケーションのフレームクラス情報を指定します。

図 12.8 Java アプリケーションウィザード：ページ 2



以下のオプションがあります。

[クラス名](#)

[パッケージ名](#)

[場所](#)

[AWT フレーム](#)

[Swing フレーム](#)

クラス名

[クラス名] フィールドにフレームのクラス名を入力します。サンプル名が表示されます。

パッケージ名

[パッケージ名] フィールドに (必要であれば) フレームクラスのパッケージ名を入力します。サンプル名が表示されます。

場所

[場所] フィールドに新規アプリケーションのフレームクラスを保存する場所を入力します。横のボタンをクリックしてダイアログを開き、そこから保存場所を選択することもできます。有効な場所は厳密なファイル名とパッケージルーツによって変わります。

AWT フレーム

[AWT フレーム] ラジオボタンをクリックすると新規アプリケーションを AWT フレームとして生成します。デフォルトはオンです。

Swing フレーム

[Swing フレーム] ラジオボタンをクリックすると新規アプリケーションを Swing フレームとして生成します。

3. Java アプリケーションの情報を追加する

ページ 3 ([図 12.9](#)) ではアプレットの追加情報を指定します。

図 12.9 Java アプリケーションウィザード：ページ 3



以下のオプションがあります。

[アプリケーションタイトル](#)

[説明](#)

[すべてのファイルに情報を含める](#)

アプリケーションタイトル

[アプリケーションタイトル] フィールドに新規アプリケーションのタイトルを入力します。

説明

必要であれば [説明] フィールドに新規アプリケーションの内容を入力します。

すべてのファイルに情報を含める

[アプリケーションタイトル] [説明] フィールドの情報を各アプリケーションファイルの最初に追加するには、[すべてのファイルに情報を含める] チェックボックスをオンにします。デフォルトはオンです。

4. [完了] ボタンをクリック

現在の設定を適用して新しい RAD アイテムを作るには [作成] ボタンをクリックします。ウィザードに戻って設定の変更を続けるには [キャンセル] ボタンをクリックします。

Java Bean ウィザード

Java Bean ウィザードでは以下のステップを実行します。

1. [Java Bean の名前と位置を選択する](#)
2. [基底クラスと実装リストを指定する](#)

以下は Java Bean ウィザードを使う手順です。

1. Java Bean の名前と位置を選択する

Java Bean ウィザードの名前と場所のページ ([図 12.10](#)) で新規の Bean の名前と場所、修飾子を指定します。

図 12.10 Java Bean ウィザード：名前と場所

The screenshot shows the '新規 Java Bean' (New Java Bean) dialog box with the '名前と場所' (Name and Location) tab selected. The 'クラス名' (Class Name) field contains 'JavaBean'. The 'Bean のクラス' (Bean Class) checkbox is checked. The 'ファイル' (File) dropdown is set to '新規ファイル' (New File). The '内部' (Internal) dropdown is selected under the 'クラス' (Class) section, with a '設定...' (Settings...) button to its right. The 'パッケージ' (Package) field contains the path 'C:\Program Files\Metrowerks\CodeWarrior\MyProject\JavaBean\JavaBean.j', also with a '設定...' (Settings...) button. The 'アクセス' (Access) dropdown is set to 'Public', and the '指定子' (Qualifier) dropdown is set to 'なし' (None). At the bottom, there are four buttons: '< 戻る(B)' (Back), '次へ(N) >' (Next), '完了(F)' (Finish), and 'キャンセル' (Cancel).

以下のオプションがあります。

[クラス名](#)

[ファイル](#)

[パッケージ](#)

[修飾子](#)

クラス名

[クラス名] フィールドに Bean のクラス名を入力します。サンプル名が表示されます。[クラスは Bean] チェックボックスがオンになります。これは新規 Java Bean を作成していることを示しています。

注意： 他の開発ツールと互換性のあるアプレットを作るには、クラス名とアプレットのファイル名を同じにします。

ファイル

[ファイル] ポップアップメニューは [新規ファイル] を表示します。他の選択肢はありません。

[パッケージ] フィールドの上のフィールドにファイルを保存する場所を入力してください ([図 12.10](#))。横のボタンをクリックしてダイアログを開き、そこで保存場所を選択することもできます。

パッケージ

[パッケージ] フィールドに (必要であれば) Java Bean のパッケージ名を入力します。

注意： 他の開発ツールと互換性のあるアプレットを作るには、ファイルパスとパッケージパスを同じにします。

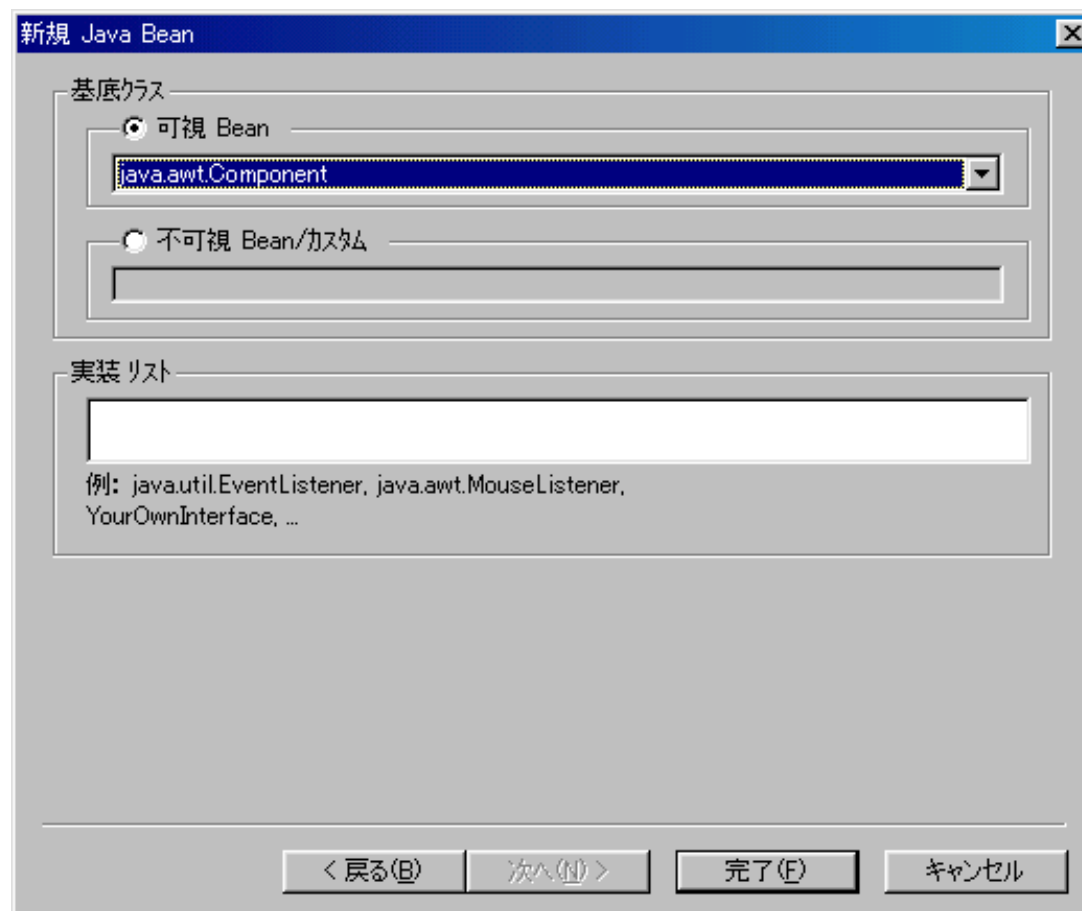
修飾子

[アクセス] ポップアップメニューは [Public] を表示します。これは Public アクセスを持つ Java Bean を作成することを意味します。他の選択肢はありません。[指定子] ポップアップメニューで新規 Java Bean のメソッド修飾子を選択します。選択肢には [なし] [Abstract] [Final] です。

2. 基底クラスと実装リストを指定する

基底クラスとインターフェースのページ ([図 12.11](#)) で新規 Bean の基底クラスと実装リストを指定します。

図 12.11 Java Bean ウィザード：基底クラスとインターフェース



以下のオプションがあります。

[基底クラス](#)

[実装リスト](#)

基底クラス

[基底クラス] オプションでは新規 Bean の基底クラス型を指定します。

可視 Bean：基底クラスが可視 Bean のときにオンにします。この下のポップアップメニューで [common AWT] [Swing base classes] いずれかの基底クラスを選択します。

不可視 Bean/ カスタム：イベントセットの基底クラスが不可視 Bean、またはカスタムクラスのときにオンにします。この下のフィールドに基底クラスの名前を入力してください。

実装リスト

[実装リスト] 欄に基底クラスに実装するインターフェースリストを入力します。サンプルのインターフェースが表示されます。複数のインターフェースはコンマで区切ります。

3. [完了] ボタンをクリック

現在の設定を適用して新しい RAD アイテムを作るには [作成] ボタンをクリックします。ウィザードに戻って設定の変更を続けるには [キャンセル] ボタンをクリックします。

デザインを作成

IDE はデザインを利用して RAD プロジェクトを管理します。デザインにはユーザーインターフェースのレイアウトセットやアプリケーションの一部であるコンポーネントが含まれます。CodeWarrior プロジェクトは複数のデザインを持つことができます。各デザインは複数のビルドターゲットを持つことができます。

以下の内容について説明します。

[プロジェクトへデザインを追加](#)

[プロジェクトウィンドウのデザインビュー](#)

[ターゲットビューのデザイン](#)

プロジェクトへデザインを追加

プロジェクトにデザインを追加するには、既存の Java プロジェクトを開いた後、ファイルメニューの [新規] を選択します。IDE は「新規」ダイアログ ([図 12.12](#)) を表示します。[プロジェクト] タブをクリックして利用可能なウィザードのリストを表示します。

デザインを作成するウィザードは、プロジェクトを作成するウィザードと同じです。

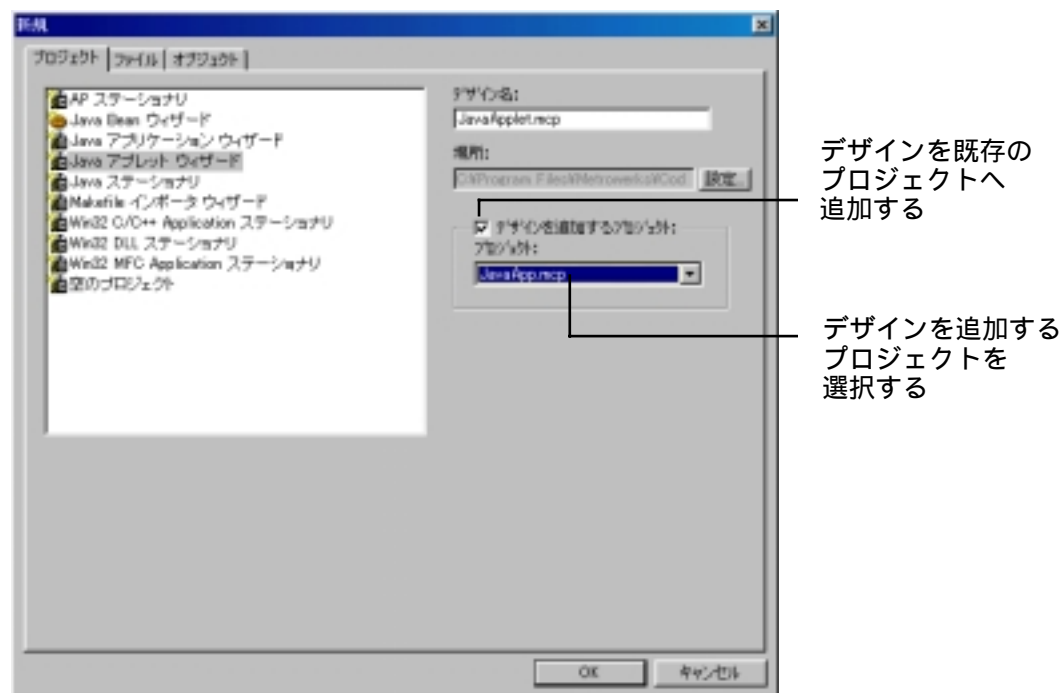
[Java アプレットウィザード](#) : Java アプレットの作成を支援します。アプレットのクラス、HTML ページ、パラメータ、内容などの情報を要求します。

[Java アプリケーションウィザード](#) : Java アプリケーションの作成を支援します。アプリケーションのクラス、フレームクラス、内容などの情報を要求します。

[Java Bean ウィザード](#) : Java Bean の作成を支援します。Bean のクラス、位置、パッケージ、修飾子などの情報を要求します。

例えば Java アプレットデザインを作る場合、[Java アプレットウィザード] を選択します。

図 12.12 プロジェクトにデザインを追加



プロジェクトへデザインを追加するには、[デザインをプロジェクトに追加]チェックボックスをオンにします。このチェックボックスはプロジェクトが1つも開かれていない場合、選択不可または選択無効です。既存のプロジェクトへデザインを追加するとき、[場所]フィールドは灰色で表示されます(図 12.12)。[デザイン名]フィールドに新規デザインの名前を入力してください。

開いているプロジェクトの名前が[プロジェクト]ポップアップメニューに表示されます。[プロジェクト]ポップアップメニューからデザインを追加するプロジェクト名を選びます(図 12.12)。

[OK]ボタンをクリックすると選択したウィザードが現れ、デザイン作成を支援します。各ウィザードの詳細は「[RAD ウィザード](#)」(p369)を参照してください。ウィザードでの作業を終了すると、IDE はプロジェクトへデザインを追加します。

RAD プロジェクトは複数のデザインを持つことができます。しかし 1 つのプロジェクトファイルに設定できるビルドターゲットの数は最大 255 個です。各デザインは複数のビルドターゲットを持つことができますが、このことに注意してください。RAD プロジェクトに複数のデザインを作成する方法は、「[複合プロジェクトの作成方法](#)」(p66)と同じです。

図 12.13 プロジェクトウィンドウのデザインビュー



プロジェクトウィンドウのデザインビュー

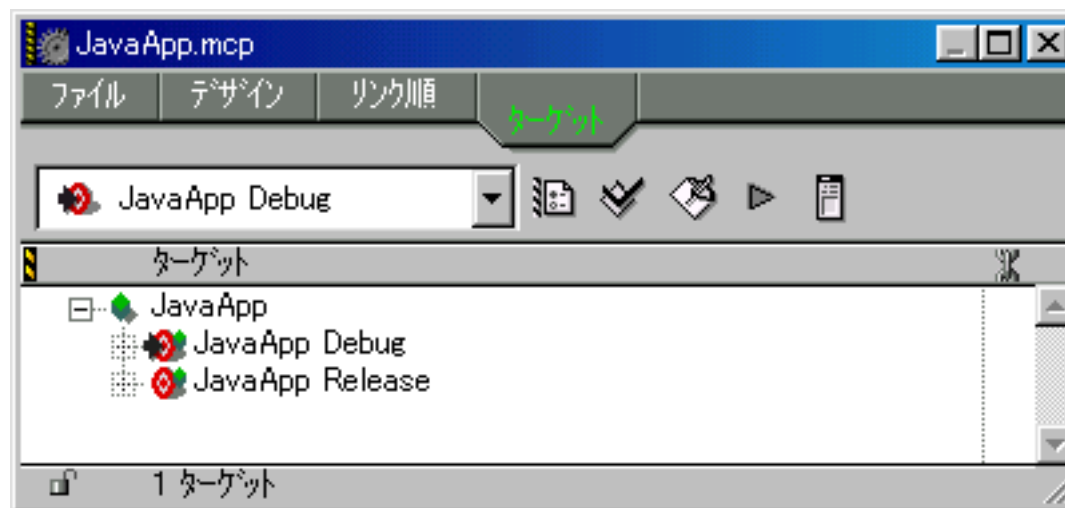
RAD プロジェクトを作成したとき、プロジェクトウィンドウの[デザイン]タブをクリックするとデザインビューが表示されます。デザインビュー(図 12.13)はアクティブなビルドターゲットのデザインに関する情報を表示します。図 12.13 は「Java App Debug」ビルドターゲットのデザインを示しています。他のビルドターゲットのデザインを見るには、デザインビューの[ターゲット]ポップアップメニューでビルドターゲットを選択します。

デザインビューの項目は階層コントロールをクリックすることで拡張/縮小表示できます。

ターゲットビューのデザイン

プロジェクトウィンドウの[ターゲット]タブをクリックすると各デザイン内のビルドターゲットの階層を見ることができます(図 12.14)。カレントデザインに含まれるビルドターゲットのアイコンにはデザインバッジ()が表示されます(図 12.14)。[JavaApp Release]ビルドターゲットはカレントデザインに含まれないため、このアイコンにはデザインバッジがありません。

図 12.14 ターゲットビューのデザイン

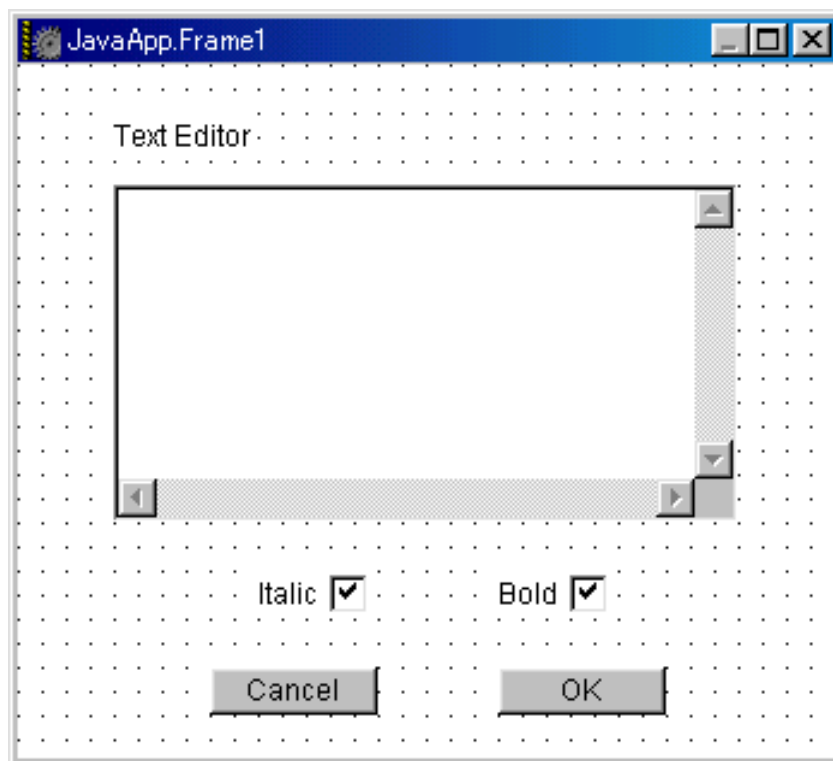


レイアウトを作成

RAD プロジェクトは複数のレイアウトを含むことができます。レイアウトとはアプリケーションで使うユーザーインターフェース要素をまとめたものです。レイアウトはウィンドウやダイアログの内容や警告を記述します。レイアウトはあるウィンドウで表示可能なビューの内容をすべて記述することができます。

レイアウトエディタは各プロジェクトに含まれるレイアウトを表示します。レイアウトエディタは IDE のプラグインアーキテクチャを利用してユーザーインターフェースを表示します。例えば Java RAD プロジェクトを作成するとき、IDE は Java RAD プラグインを利用してレイアウトエディタのユーザーインターフェースを表示します。[図 12.15](#) はレイアウトエディタの例です。

図 12.15 レイアウトエディタ上のレイアウト



レイアウトエディタとカタログを一緒に使ってアプリケーションをビルドします。RAD プロジェクトで利用可能なコンポーネントはカタログが管理しています。コンポーネントパレットとコンポーネントカタログでカタログの中身を見たり、使うことができます。レイアウトに新しいインターフェース用を追加するには、コンポーネントカタログからコンポーネントをドラッグしてレイアウト上にドロップします。コンポーネントパレットでサンプルコンポーネントを選択することができます ([図 12.16](#))。

図 12.16 コンポーネントパレット

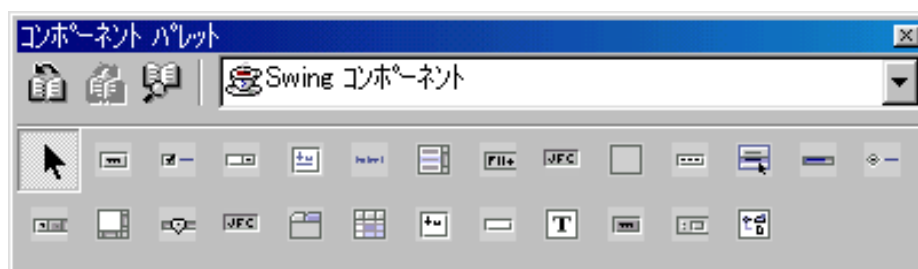


図 12.17 コンポーネントパレット



レイアウトの作り方、レイアウトエディタの使い方は[「RAD レイアウトの編集」\(p389\)](#)を参照してください。

レイアウトエディタとカタログツールについては[「コンポーネントパレット」\(p398\)](#)、[「コンポーネントカタログウィンドウ」\(p400\)](#)を参照してください。

[「RAD コンポーネント」\(p415\)](#) はカタログに含まれる各コンポーネントについて説明します。

第 13 章 RAD レイアウトの編集

CodeWarrior の Java 用 RAD (Rapid Application Development) ツールには、コンポーネントパレット、コンポーネントカタログ、オブジェクトインスペクタなどがあります。これらのツールを利用してアプリケーションのユーザーインターフェースを視覚的に作成します。この章で 5 つの RAD ツールについて説明します。

以下の項目について説明します。

[レイアウトエディタ](#)

[レイアウトウィザード](#)

[コンポーネントパレット](#)

[コンポーネントカタログウィンドウ](#)

[オブジェクトインスペクタ](#)

レイアウトエディタ

レイアウトエディタは RAD プロジェクトのグラフィカルユーザーインターフェースを表示します。IDE は RAD プラグインアーキテクチャを用いてレイアウトエディタ環境を表示します。例えば RAD ツールを使って Java アプリケーションを開発する場合、IDE は Java プラグインを利用して画面上にレイアウトを描画します。

[レイアウトを作成](#)

[レイアウトを修正](#)

レイアウトを作成

RAD プロジェクトで新規レイアウトを作成するとき、デフォルト状態のレイアウトはユーザーインターフェースコンポーネントを含んでいません。コンポーネントカタログウィンドウか、コンポーネントパレットを使ってレイアウト上にユーザーインターフェースコンポーネントを配置します。アプリケーションのインターフェースをビルドするとき、RAD ツールはランタイムで自動的にインターフェースを作るソースコードを生成します。

RAD デザインは複数のレイアウトを持つことができます。しかしパフォーマンスの低下を防ぐために、1 つのプロジェクトで使うレイアウトの数は制限したほうが良いでしょう。プロジェクトに 30 以上のレイアウトが含まれている場合、レイアウトを整理すべきです。RAD プロジェクトで複数レイアウトを作成する方法は、[「複合プロジェクトの作成方法」\(p66\)](#) と同じです。

開いているプロジェクトへ新規レイアウトファイルを追加するには、ファイルメニューの [新規] を選択します。「新規」ダイアログで [オブジェクト] タブをクリックしてオブジェクトウィザードの一覧を表示します ([図 13.1](#))。

作成するレイアウトに適したウィザードを選択してください。

[Java フレームウィザード](#) : プロジェクトのレイアウトフレームの作成を支援します。フレームのクラス、パッケージ、場所を指定します。

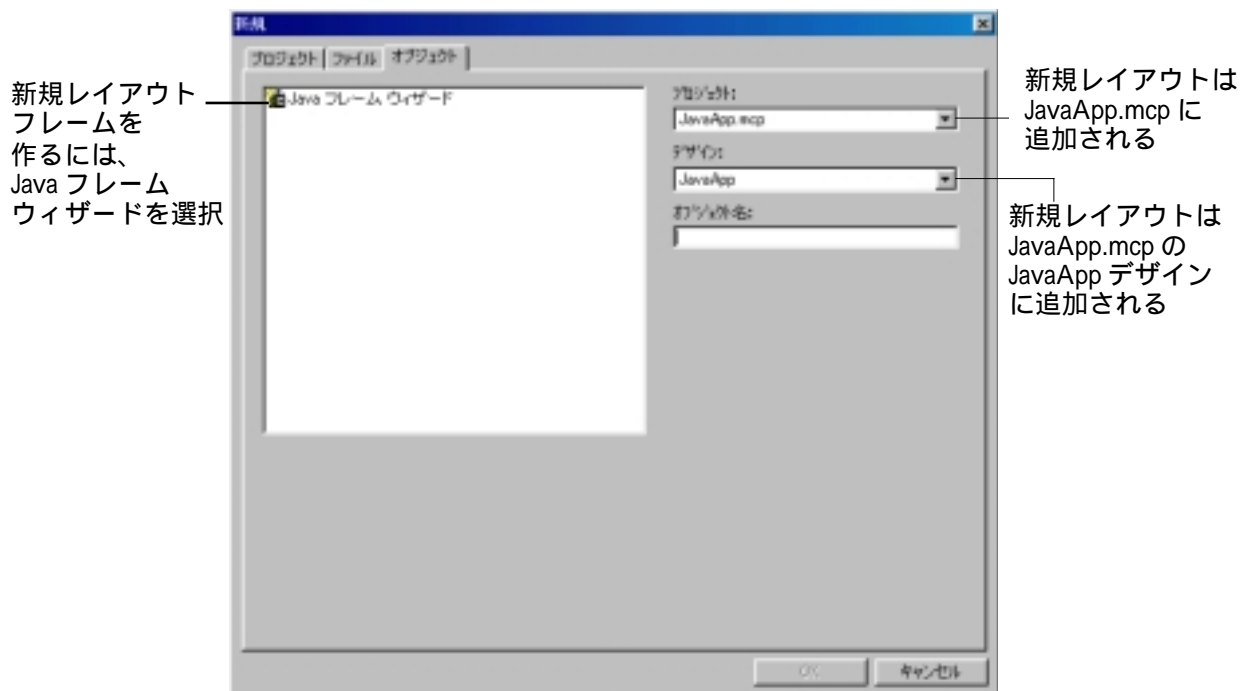
[プロジェクト] ポップアップメニューには開いているプロジェクトの一覧が表示されます。[プロジェクト] ポップアップメニューで新規レイアウトを追加するプロジェクトを選択します。[デザイン] ポップアップメニューには開いているプロジェクトに含まれるデザインの一覧が表示されます。[デザイン] ポップアップメニューで新規レイアウトを追加するデザインを選択します。

[OK] ボタンをクリックするとウィザードのページが現れます。新規レイアウトの設定を終えたら、[完了] ボタンをクリックしてレイアウトをプロジェクトへ追加します。新規レイアウトウィンドウが現れます。レイアウトウィザードについては「[レイアウトウィザード](#)」(p396) を参照してください。

レイアウトを修正

既存のレイアウトを変更するには、レイアウトエディタと一緒にコンポーネントパレット、コンポーネントカタログ、オブジェクトインスペクタのいずれかを使います。コンポーネントカタログウィンドウかコンポーネントパレットで選択したコンポーネントをレイアウトウィンドウに配置します。ドローアプリケーションでグラフィックオブジェクトを操作するように、レイアウトウィンドウでコンポーネントを操作します。

図 13.1 新規レイアウトを作成



レイアウトを変更する方法について説明します。

[オブジェクトを作成](#)

[オブジェクトを操作](#)

[オブジェクトを調べる](#)

[オブジェクトを削除](#)

[レイアウトエディタのコンテキストメニュー](#)

オブジェクトを作成

RAD ツールは現在のカatalogのプロジェクトコンポーネントから利用できます。コンポーネントパレットとコンポーネントCatalogウィンドウに、コンポーネントにアクセスする便利な方法が 2 つあります。

コンポーネントCatalogウィンドウでは、レイアウトウィンドウへコンポーネントをドラッグして新しいインターフェースオブジェクトを作成することができます。コンポーネントパレットでは、追加したいコンポーネントを表すボタンをクリックして、つぎにレイアウトウィンドウへカーソルをドラッグするとコンポーネントをレイアウト上へ配置できます。

コンポーネントパレットについては「[コンポーネントパレット](#)」(p398)、コンポーネントCatalogウィンドウについては「[コンポーネントCatalogウィンドウ](#)」(p400) を参照してください。

オブジェクトを操作

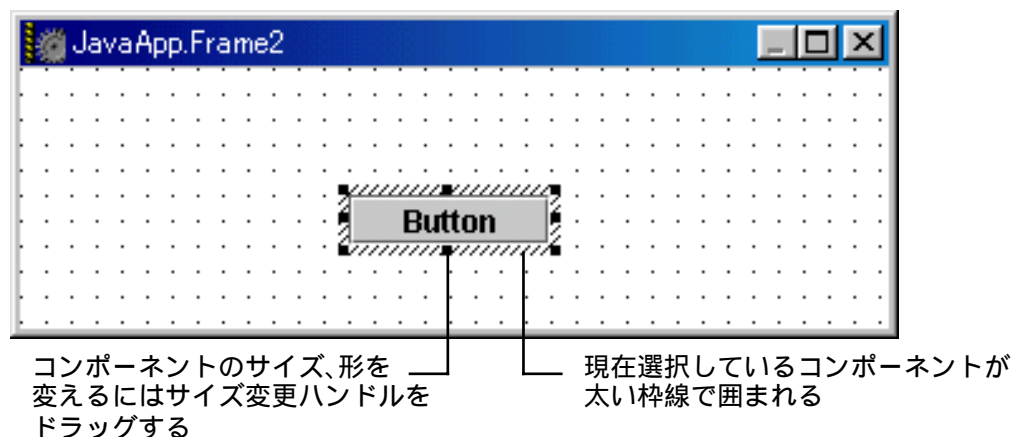
レイアウト内でオブジェクトを移動するには、そのオブジェクトを選択します ([図 13.2](#))。選択しているオブジェクトの周りには太い枠線が現れます。選択したオブジェクトをドラッグして新しい位置へ移動します。または矢印キーで移動することもできます。オブジェクトのサイズや形を変更するには、サイズ変更ハンドルを希望の位置へドラッグします。ドラッグ時に表示される太い枠線は、変更したオブジェクトのサイズや形を表わします。

以下の方法で複数のオブジェクトを選択できます。

追加選択したいオブジェクトを Shift + クリックします。太い枠線が現れます。オブジェクトの選択を解除するには、そのオブジェクトを再度 Shift + クリックします。

選択したい複数のオブジェクトを矩形選択します ([図 13.3 \(p394\)](#))。選択したいオブジェクトグループのそばにカーソルを置き、ドラッグすると四角い枠線が現れます。選択したいオブジェクトを枠線で囲んだときにカーソルを離します。さらにオブジェクトを追加選択するには、Shift キーを押しながらカーソルをドラッグしてオブジェクトを囲みます。カーソルを離れたときにオブジェクトが追加されます。

図 13.2 選択したオブジェクト



レイアウトエディタのコンテキストメニューでもオブジェクトを操作できます。「[レイアウトエディタのコンテキストメニュー](#)」(p395) を参照してください。

ヒント： Tab キーもオブジェクトのレイアウトに便利です。レイアウトウィンドウでオブジェクトを選択した後、Tab キーで他のオブジェクトを選択できます。Tab キーがオブジェクトを選択する順番はレイアウトウィンドウの配置順によって変わります。

オブジェクトを調べる

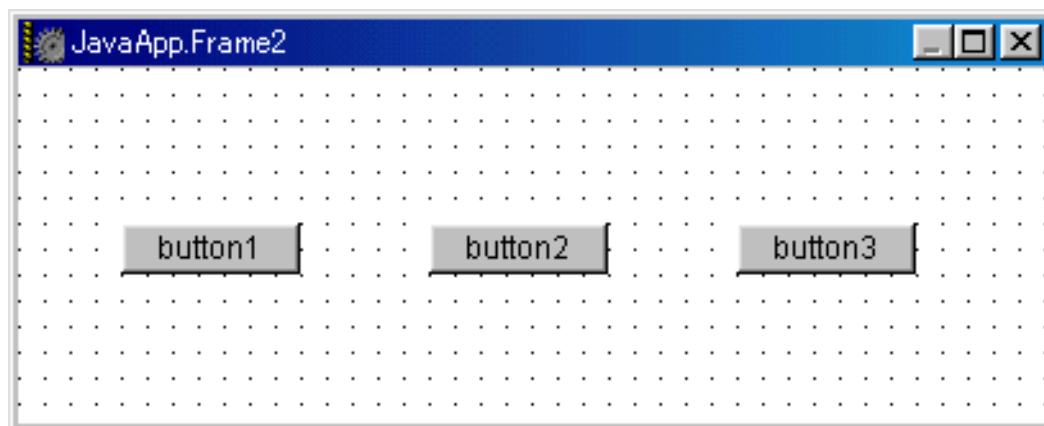
レイアウト上にオブジェクトを配置した後、各オブジェクトの属性、それが処理するイベントを編集することができます。オブジェクトインスペクタは選択したオブジェクトの属性とイベントを表示します。「[オブジェクトインスペクタ](#)」(p409) を参照してください。

オブジェクトを削除

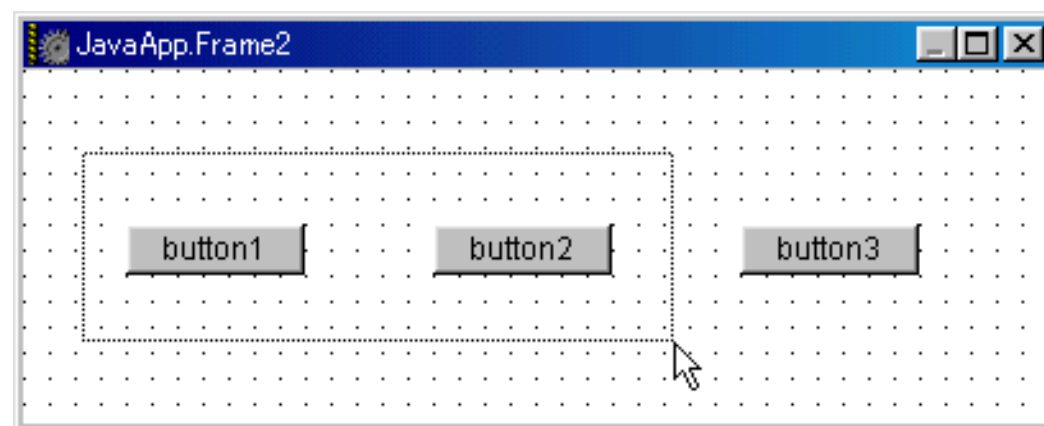
レイアウト上でオブジェクトを削除するには、オブジェクトを選択して Delete キーを押します。オブジェクトを選択して編集メニューの [削除] でも削除できます。レイアウトエディタがレイアウトからオブジェクトを削除します。

図 13.3 2つのオブジェクトを矩形選択

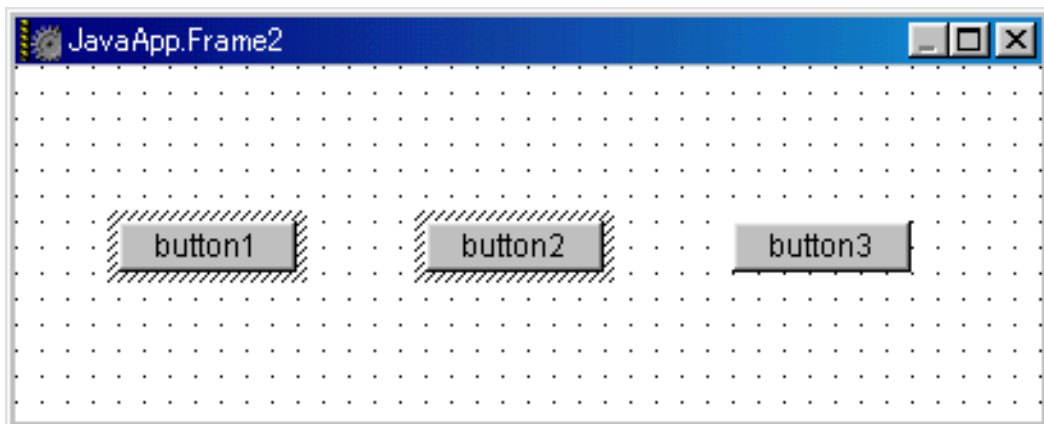
選択前



選択中



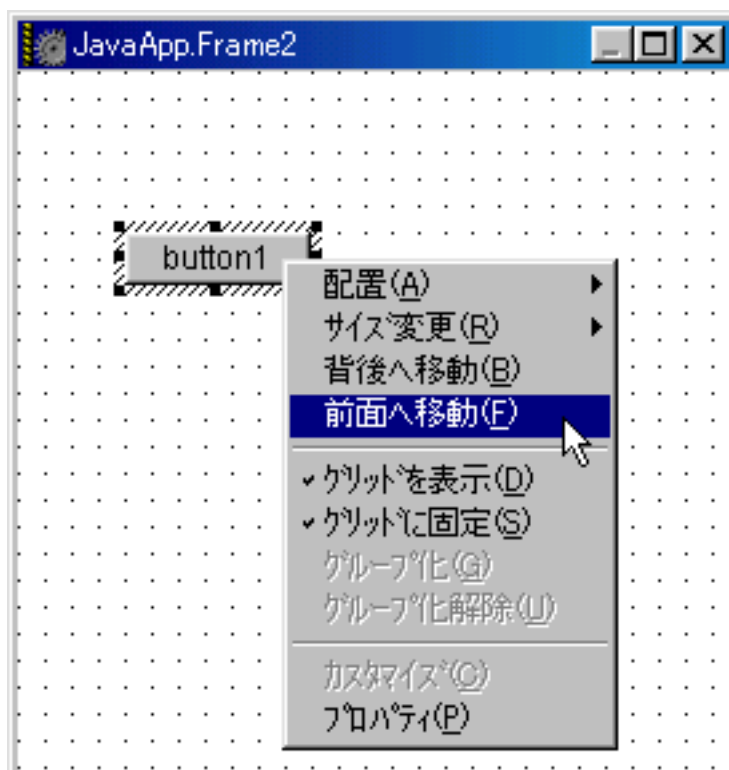
選択後



レイアウトエディタのコンテキストメニュー

レイアウトエディタに、レイアウトメニューのコマンドを表示するコンテキストメニューがあります(図 13.4)。利用可能なコンテキストメニューのコマンドは、選択した項目によって異なります。

図 13.4 レイアウトエディタのコンテキストメニュー



以下の方法でコンテキストメニューを表示します。

オブジェクトを右クリックする

コンテキストメニューのグラフィック編集コマンドをまとめます。

配置：選択したオブジェクトの配置方法を選ぶサブメニューを表示する

サイズ変更：選択したオブジェクトのサイズを変更する方法を選ぶサブメニューを表示する

背面へ移動：選択したオブジェクトを他のオブジェクトの背面に送る

前面へ移動：選択したオブジェクトを他のオブジェクトの前面に出す

グリッド表示：レイアウトウィンドウにグリッドを表示する

グリッドに固定：[レイアウトエディタ](#)は自動的にグリッドに沿ってオブジェクトを整列する

グループ化：選択したオブジェクトをグループ化する

グループ化解除：グループのグループ化を解除する

カスタマイズ：選択したオブジェクト（ポップアップメニューなど）をカスタマイズする

プロパティ：選択したオブジェクトの属性を調べる [オブジェクトインスペクタ](#) を表示する

レイアウトウィザード

RAD プロジェクトのレイアウトを作るとき（「[レイアウトを作成](#)」(p389) を参照）レイアウトウィザードを選択します。レイアウトウィザードはレイアウトの作成を支援します。

[Java フレームウィザード](#)

初心者：各ウィザードは基本的なプログラミング言語を理解している方を対象に作られています。例えば Java アプリケーションウィザードを使う場合、AWT（Abstract Window Toolkit）フレームと Swing フレームの違いを理解しておく必要があります。ウィザードが表示する項目がよくわからないときは、プログラミングの参考書などを参照してください。

レイアウトウィザードには以下のボタンがあります。

戻る：前のページへ戻る

次へ：次のページへ進む

完了：現在の情報を表示する

キャンセル：すべての変更を破棄する

各ウィザードは一連のページを表示します。順番に従ってページで情報を設定し、終了したら [次へ] ボタンをクリックします。

各ページのデフォルト設定を変更することもできます。現在のウィザードの設定を受け入れるには [完了] ボタンをクリックします。ウィザードは新規プロジェクトの現在の設定を表示します。現在の設定を適用して新しいレイアウトを作るには [作成] ボタンをクリックします。ウィザードに戻って設定の変更を続けるには [キャンセル] ボタンをクリックします。

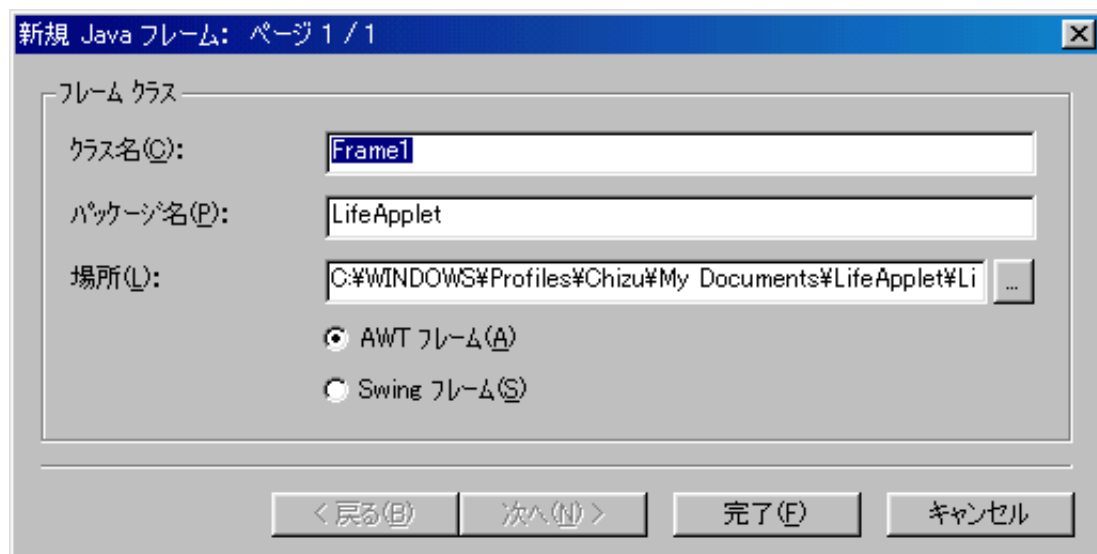
Java フレームウィザード

以下は Java フレームウィザードを使う手順です。

1. Java フレーム用のフレームクラスを記述する

Java フレームウィザードのページ 1/1（[図 13.5](#)）では、クラス名、パッケージ名、新規フレームの保存場所を指定します。フレーム記述のためのオプションもあります。

図 13.5 Java フレームウィザード：ページ 1/1



以下のオプションがあります。

[クラス名](#)

[パッケージ名](#)

[場所](#)

[AWT フレーム](#)

[Swing フレーム](#)

クラス名

[クラス名] フィールドにフレームのクラス名を入力します。サンプル名が表示されます。

パッケージ名

必要であれば [フィールドにフレームクラスのパッケージ名を入力します。サンプル名が表示されます。

場所

[場所] フィールドに新規レイアウトのフレームクラスを保存する場所を入力します。横のボタンをクリックしてダイアログを開き、そこから保存場所を選択することもできます。

AWT フレーム

[AWT フレーム] ラジオボタンをクリックすると新規フレームを AWT フレームとして生成します。デフォルトはオンです。

Swing フレーム

[Swing フレーム] ラジオボタンをクリックすると新規フレームを Swing フレームとして生成します。

コンポーネントパレット

コンポーネントパレットは、現在のカタログにあるコンポーネントツールを表示します。このツールにより、アプリケーションのユーザーインターフェースを簡単に作成できます。コンポーネントパレットを表示するには、ウィンドウメニューの[コンポーネントパレット]を選択してください ([図 13.6](#))。

図 13.6 コンポーネントパレット



コンポーネントパレットには 3 つの欄があります。

[コンポーネントパレットのツールバー](#)

[カタログのポップアップメニュー](#)

[コンポーネントツール](#)


コンポーネントパレットのツールバー



コンポーネントパレットツールバーからコンポーネントへアクセスできます。 [表 13.1](#) でコンポーネントパレットのツールバーのボタンを詳しく説明します。

カタログのポップアップメニュー

コンポーネントパレットはカタログにあるコンポーネントツールを表示します。他のカタログにあるコンポーネントツールを使うには、カタログポップアップメニューでカタログ名を選択します ([図 13.6](#))。

表 13.1 コンポーネントパレットのツールバーのボタン

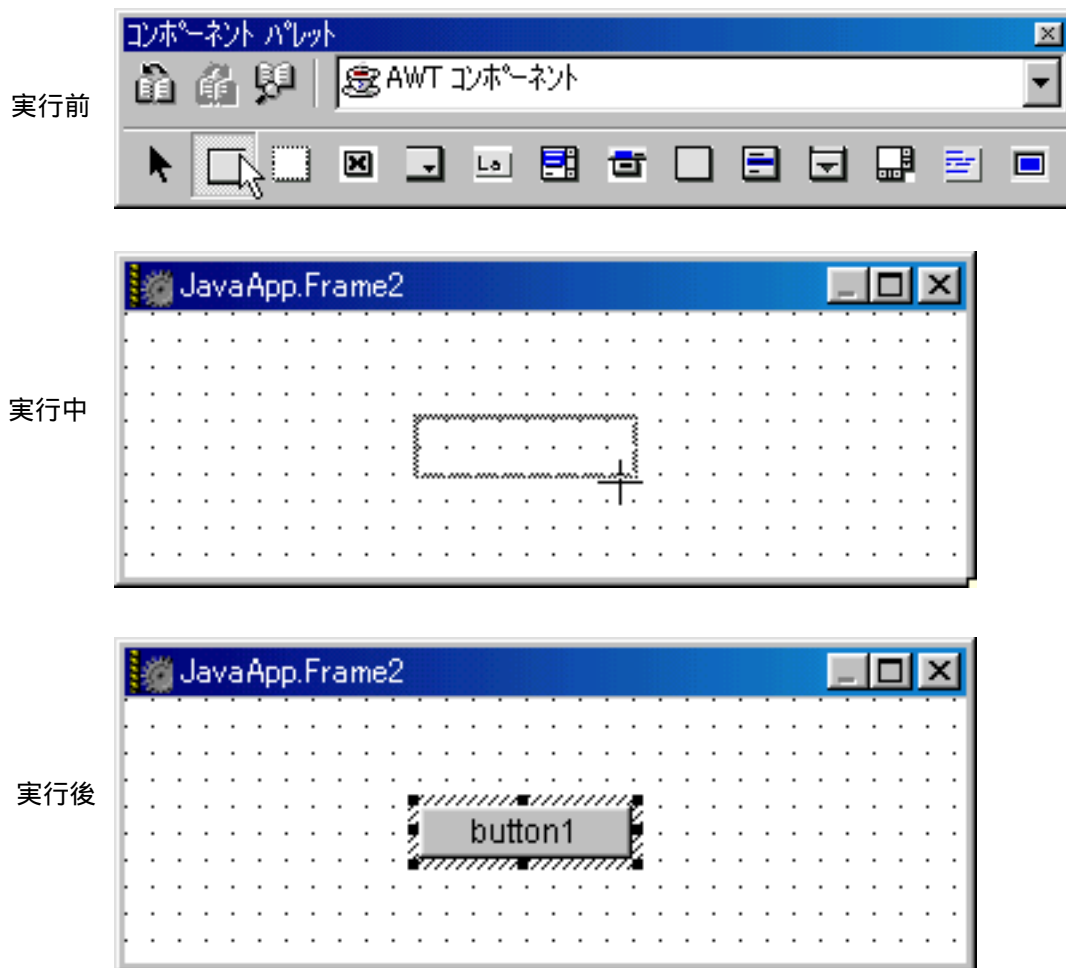
ボタン	名前	説明
	カタログを開く	カタログファイルを開き、それをコンポーネントパレットへ追加する

ボタン	名前	説明
	カタログを閉じる	追加したカタログファイルを閉じる（閉じたカタログはコンポーネントパレットとコンポーネントカタログウィンドウから削除できる）
	コンポーネントカタログ	コンポーネントカタログウィンドウを開く

コンポーネントツール

コンポーネントパレットの下部に現在のカタログにあるツールが表示されます。ツールをクリックした後、レイアウトウィンドウにオブジェクトを描くことが可能になります。オブジェクトの描画を終えたら、レイアウトエディタがレイアウトウィンドウを更新します（[図 13.7](#)）。

図 13.7 レイアウトウィンドウでオブジェクトを描く



以下はコンポーネントパレットを使う手順です。

1. レイアウトウィンドウに配置したいコンポーネントを表すツールをクリックする



間違ったツールを選択したときは [Unselect tool] をクリックしてください。他のツールを選択できます。

2. レイアウトウィンドウ上でカーソルをドラッグする

カーソルをドラッグすると、コンポーネントの形と位置を示す太い枠線が表示されます。

この操作中、カーソルは十字マークになります。十字マークをドラッグしているとき、レイアウトウィンドウに座標を示すポップアップが現れます。ドラッグ中にコンポーネントの使用を中止するには、Esc キーを押します。コンポーネントツールが消え、レイアウトウィンドウから枠線が消えます。

3. 枠線を離す

コンポーネントがレイアウトウィンドウに置かれます。[オブジェクトインスペクタ](#)でコンポーネントの属性とイベントを変更できます。

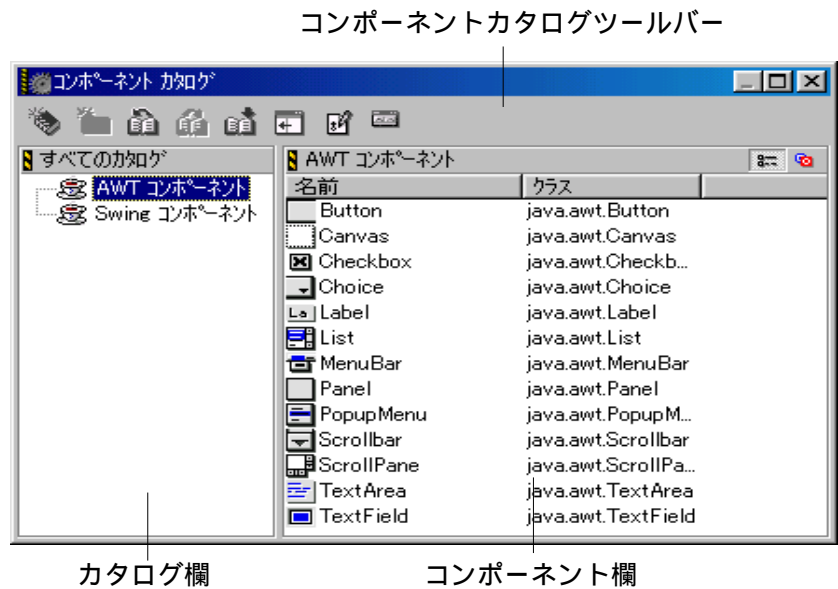
コンポーネントパレットにあるコンポーネントツールについては「[RAD コンポーネント](#)」([p415](#))を参照してください。

ヒント： コンポーネントツール上にカーソルを置いて少し待つと、そのツールの名前が表示されます。これでツールを簡単に区別できます。

コンポーネントカタログウィンドウ

コンポーネントカタログウィンドウを利用して、現在のカタログからレイアウトウィンドウへコンポーネントをドラッグ&ドロップできます。既存のカタログを見たり、カタログをインポートしたり、新しいカタログを作ることができます。コンポーネントカタログウィンドウを表示するには、ウィンドウメニューの [コンポーネントカタログ] を選択します ([図 13.8](#))。

図 13.8 コンポーネントカタログウィンドウ



以下の項目について説明します。

[コンポーネントカタログのツールバー](#)

[カタログ欄](#)

[コンポーネント欄](#)

[コンポーネントカタログのコンテキストメニュー](#)






[コンポーネントカタログを作成](#)

コンポーネントカタログのツールバー

コンポーネントカタログのツールバーでカタログとその内容进行操作できます。[表13.2](#)でツールバーのボタンを説明します。

表 13.2 コンポーネントカタログのツールバー

ボタン	名前	動作
	新規カタログ	カスタマイズしたコンポーネントを保存するための新しいカタログファイルを作成する
	新規フォルダ	独自のカタログの中に、コンポーネントを管理するための新しいフォルダを作成する
	カタログを開く	カタログファイルを開き、それをカタログ欄に追加する

ボタン	名前	動作
	カタログを閉じる	カタログ欄に追加されたカタログファイルを閉じる
	コンポーネントの読み込み	アクティブなカタログにコンポーネントをインポートする
	インデックスビューの切替え	コンポーネントカタログウィンドウのカタログ欄の表示、非表示を切り替える
	項目のプロパティを編集	選択した項目の属性を編集するオブジェクトインスペクタを開く
	コンポーネントパレット	コンポーネントパレットを開き、最前面に表示する

カタログ欄

カタログ欄 ([図 13.8 \(p401\)](#)) は利用可能なカタログや、カスタマイズしたカタログの情報を表示します。カタログの内容を拡張表示するには、横の階層コントロールをクリックします。再度クリックすると拡張表示します。また、カタログ欄で項目をダブルクリックしても同じです。

頻繁に使うコンポーネントを保存するために、カタログをカスタマイズすることができます。カタログをカスタマイズすることにより、アプリケーションのレイアウトのビルドが容易になります。独自のカタログの作り方は「[コンポーネントカタログを作成](#)」(p406) を参照してください。

ヒント： コンポーネントカタログのツールバーの「インデックスビューの切替え」ボタンをクリックするとカタログ欄を隠すことができます。

コンポーネント欄

コンポーネント欄 ([図 13.8 \(p401\)](#)) は各カタログにある項目の情報を表示します。カタログの名前は欄の左側に表示されます。コンポーネント欄からレイアウトウィンドウへ直接コンポーネントをドラッグして、ユーザーインターフェースオブジェクトを作ることができます。

コンポーネント欄について説明します。

[Content ビューのアイコンボタン](#)

[コンポーネント情報バー](#)

[コンポーネントリスト](#)

[コンポーネントリストのソートとサイズ変更](#)

Content ビューのアイコンボタン

[Content View] アイコンボタンは現在のカタログにあるコンポーネントの表示を切り替えます。



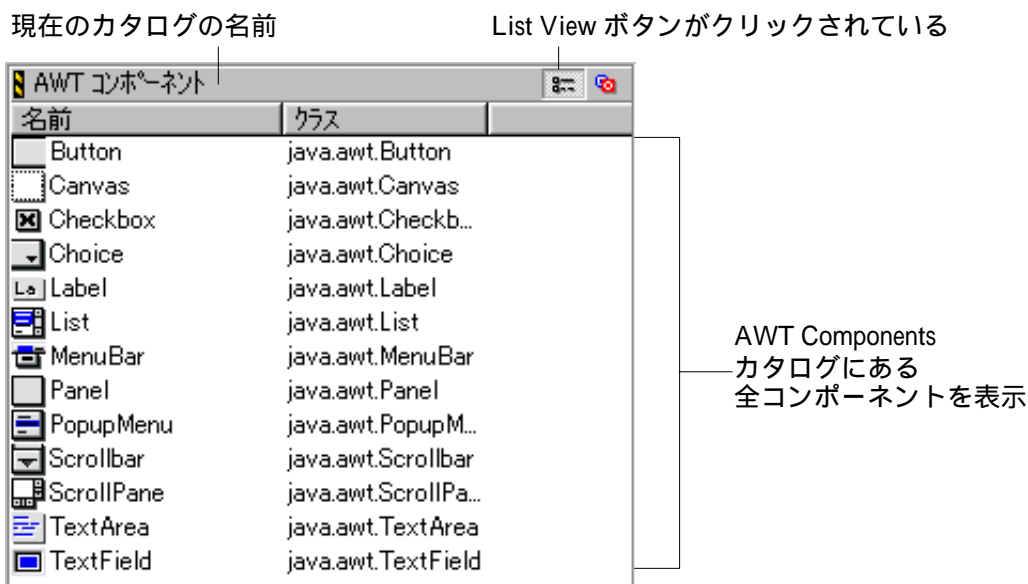
[List View] アイコンボタン ([図 13.9](#)) はコンポーネント欄にあるコンポーネントの一覧を表示します。List ビューはすべてのコンポーネントを表示します。



[Live View] アイコンボタン ([図 13.10](#)) はコンポーネント欄にあるコンポーネントをライブオブジェクトとして表示します。Live ビューは独自に作成したコンポーネントカタログでのみ利用可能です。詳細は「[コンポーネントカタログを作成](#)」(p406)を参照してください。

注意： 独自のカatalog以外でLive ビューを利用しようとすると、IDE は「[カタログに表示できるコンテンツはありません](#)」というエラーメッセージを表示します。カタログには[カタログ欄](#)で選択したダイアログの名前が入ります。

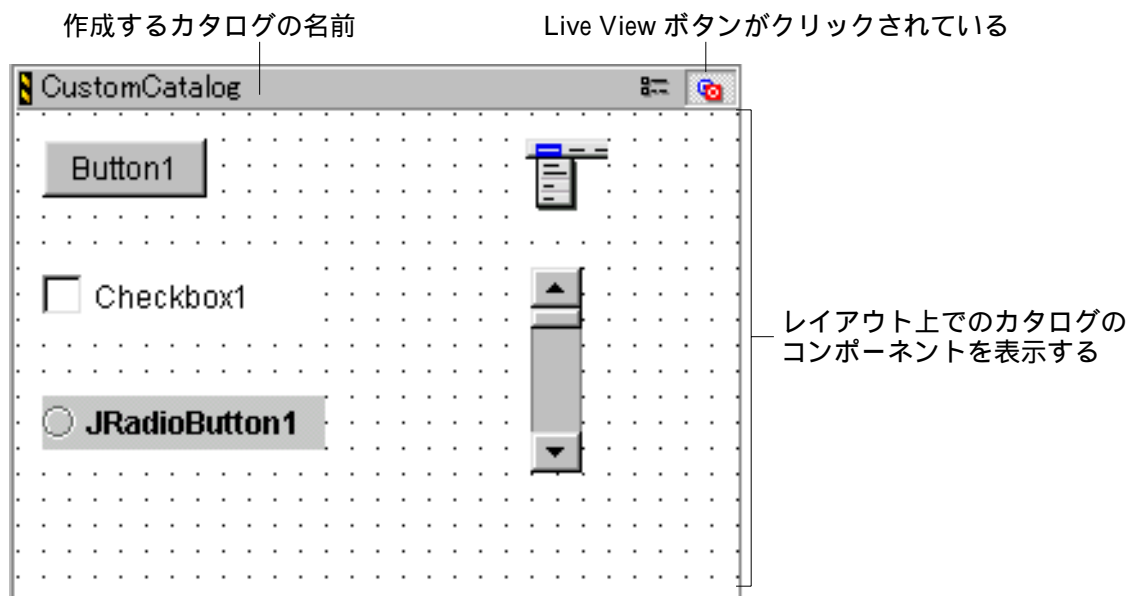
図 13.9 コンポーネント欄：List ビュー



コンポーネント欄の 2 つのビューはそれぞれ別のタスクを行います。List ビューはカタログの内容を素早く見るのに便利です。Live ビューは独自のカatalogの内容を視覚的に表示します。

また Live ビューはコンポーネントのカスタマイズセットの作成に役立ちます。例えば Live ビューでコンポーネントのデフォルトのサイズや形を変更できます。コンポーネントを使ってレイアウトウィンドウに新しいインターフェースオブジェクトを作る場合、新オブジェクトのサイズと形はそのコンポーネントと同じです。

図 13.10 コンポーネント欄：Live ビュー

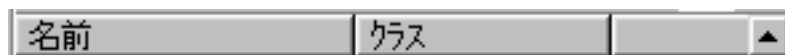


コンポーネント情報バー

コンポーネント欄の List ビューにコンポーネント情報バーがあります([図 13.11](#))。コンポーネント情報バーは現在のカタログにあるコンポーネントを管理します。

コンポーネント情報バーの列を基準にしてコンポーネントリストをソートできます。また各列のサイズも変更できます。詳細は「[コンポーネントリストのソートとサイズ変更](#)」(p405)を参照してください。

図 13.11 コンポーネント情報バー




[表 13.3](#) でカタログの種類ごとにコンポーネント情報バーの列を説明します。

コンポーネントリスト

コンポーネントリスト([図 13.9 \(p403\)](#))は現在のカタログとそれに関連するコンポーネントパレットにあるコンポーネントをすべて表示します。レイアウトにコンポーネントを配置するには、コンポーネントカタログウィンドウのコンポーネントリストからドラッグしてレイアウトウィンドウ上へドロップします。レイアウトウィンドウ上へドロップしたとき、太い枠線が最終位置を示します。枠線を離すと、レイアウトウィンドウ上にコンポーネントが配置されます。

表 13.3 コンポーネント情報バーの項目

カタログの種類	列	説明
Java カタログ (AWT コンポーネント、 Swing コンポーネント)	名前	コンポーネントの名前
	クラス	コンポーネントをメンバとして持つクラス
カスタマイズカタログ (作成またはインポート したもの)	Name	コンポーネントの名前
		コンポーネントのロック状態
	Modified	コンポーネントの最終修正日付
	Class	コンポーネントをメンバとして持つクラス
	Comments	コンポーネントのコメントを表示

レイアウトへのコンポーネントのドラッグを中止するには、コンポーネントをレイアウトウィンドウの外へドラッグして離します。コンポーネントを離した後、コンポーネントリストへ戻って他のコンポーネントをドラッグすることができます。

レイアウトからオブジェクトを削除する方法は「[オブジェクトを削除](#)」(p393) を参照してください。コンテキストメニューのコマンドについては「[レイアウトエディタのコンテキストメニュー](#)」(p395)、コンポーネントリストについては「[RAD コンポーネント](#)」(p415) を参照してください。

コンポーネントリストのソートとサイズ変更

コンポーネント情報バーの列を基準にして List ビューのコンポーネントをソートできます ([図 13.11](#) (p404))。

ソートの基準にする列をクリックしてください。列をクリックするたびにソート順の昇順と降順が切り替わります。例えばコンポーネントを名前でソートする場合、コンポーネント情報バーの [名前] 列をクリックします。再度列をクリックするとコンポーネントは降順に並びます。



コンポーネント情報バーの列サイズを変更するには、カーソルを列の間に置きます。カーソルのアイコンが左図のように変わります。これをドラッグしてサイズを変更して離します。

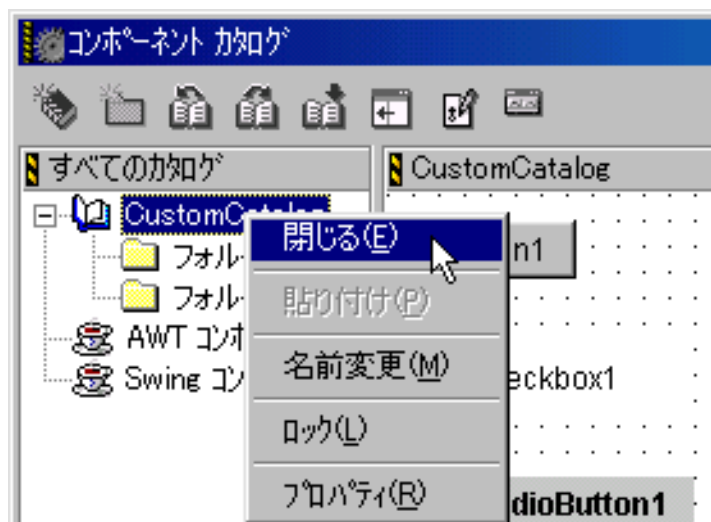
コンポーネントカタログのコンテキストメニュー

コンポーネントカタログウィンドウにはメニューコマンドを実行するコンテキストメニューがあります。選択した項目によって表示されるコマンドは異なります。コンテキストメニューの例を [図 13.12](#) に示します。

以下の操作でコンポーネントカタログのコンテキストメニューが現れます。

項目を右クリックする

図 13.12 コンポーネントカタログのウィンドウのコンテキストメニュー



コンテキストメニューの主なコマンドを説明します。

閉じる：カタログ欄から項目を削除します。

ペースト：コピーしたオブジェクトを、選択しているカタログへペーストします。

名前変更：選択した項目の名前を変更します。このコマンドを選択した後、新しい名前を入力して Enter キーを押してください。

ロック：選択した項目をロックします。ロックした項目の属性（ロック状態以外）を変更することはできません。しかし、ロックしたコンテナ（フォルダやカタログなど）へサブ項目を追加 / 削除、また変更することはできます（サブ項目をロックしていない場合）。ロックした項目に鍵のアイコンが表示されます。

ロック解除）：選択した項目のロックを解除します。

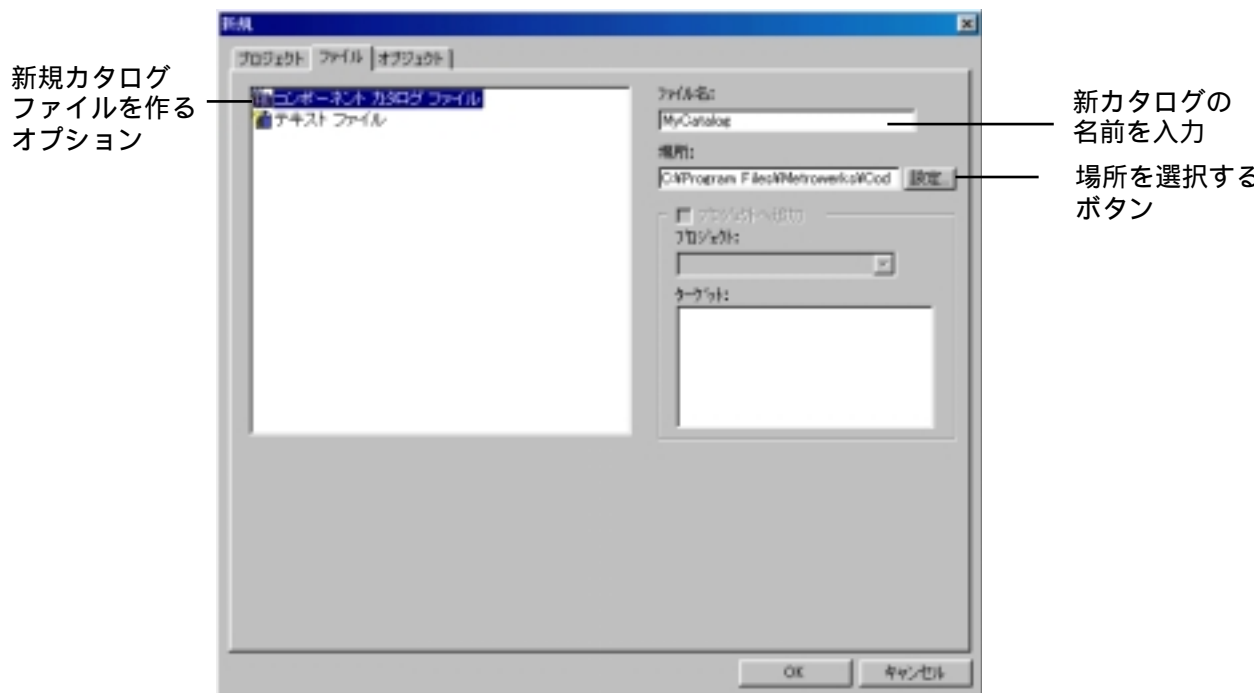
プロパティ：オブジェクトインスペクタで選択した項目の属性を表示します。

コンポーネントカタログを作成

独自のカスタマイズしたコンポーネントカタログを、頻繁に使う RAD コンポーネントへ保存することができます。複数のカタログからコンポーネントを集めて 1 つのカタログファイルにまとめることができます。新規レイアウトを作成するときに、カスタマイズしたカタログファイルを使うことができ便利です。

カタログを作るには、[コンポーネントカタログウィンドウ](#)のツールバーから [新規カタログ] を選択するか、またはファイルメニューの [新規] を選択します。ウィンドウの上の [ファイル] タブをクリックします。次にファイルビューで [コンポーネントカタログファイル] を選択します。[図 13.13](#) のようなウィンドウが現れます。

図 13.13 新規カタログファイルを作成



[名前] フィールドに新しいカタログファイルの名前を入力してください。

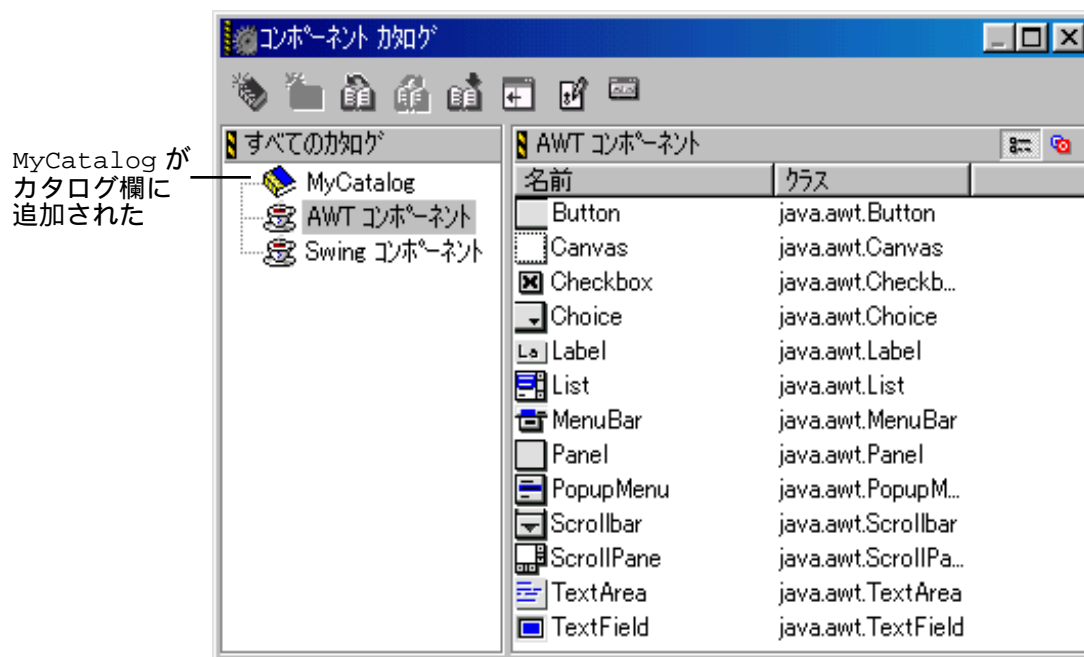
ヒント： カタログの名前には拡張子 `.ctlg` を付けてください

(例：MyCatalog.ctlg) この命名規約により高速にカタログファイルを検索できます。
Windows 版の CodeWarrior IDE は拡張子でカタログファイルを識別します。

[場所] フィールドはカタログファイルを保存するフォルダへの完全なパスを表示します。
現在のパスを変更するには、新しいパスディレクトリを入力してください。フィールド横
の [設定] ボタンをクリックしてダイアログを開き、そこで保存場所を指定することもで
きます。

カタログの名前の入力と保存場所の指定を終えたら、「新規」ダイアログの [OK] ボタン
をクリックしてください。IDE は [コンポーネントカタログウィンドウ](#) を表示し、[カタログ
欄](#) に新しいカタログを追加します ([図 13.14](#))。

図 13.14 カタログ欄へ追加した新規カタログ



以下はコンポーネントをカタログに追加する手順です。

1. 追加したいコンポーネントを含むカタログを選択する

例えば Java AWT の Button コンポーネントをカタログに追加するには、カタログ欄で [AWT コンポーネント] を選択してください。

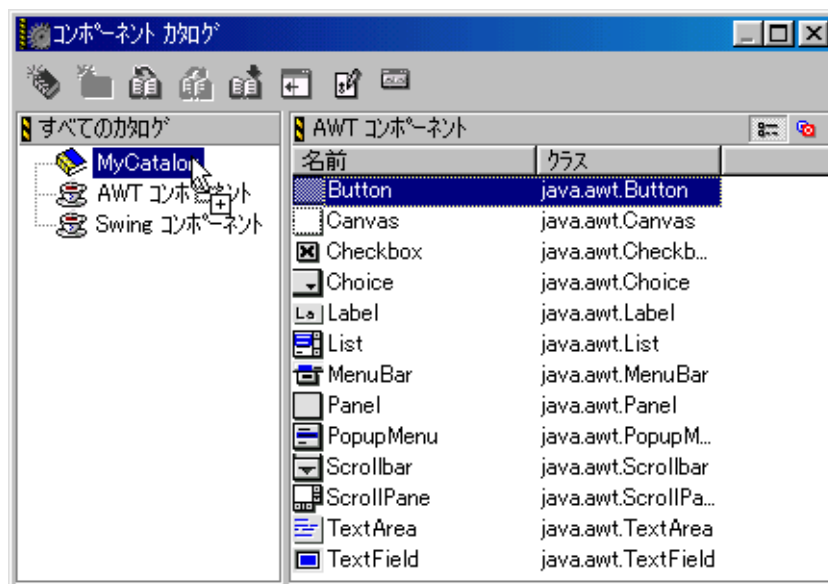
2. コンポーネント欄から追加するコンポーネントを選択する

この例では [Button] コンポーネントを選択します。

3. 選択したコンポーネントをドラッグしてカタログ上にドロップする

この例では、[Button] コンポーネントをカタログ欄の [MyCatalog] へ追加します ([図 13.15](#))。 [Button] コンポーネントを離すと、[MyCatalog] へ追加されます。

図 13.15 独自のカタログにコンポーネントを追加



ヒント： 複数のコンポーネントを同時に追加するには、ドラッグする前にコンポーネントを Ctrl + クリックします。階層カタログのビューを自動的に拡張表示するには、コンポーネントをカタログ上にドラッグして少し待ちます。

編集メニュー、および[レイアウトエディタのコンテキストメニュー](#)の[カット][コピー][ペースト]コマンドを使ってコンポーネントを追加することもできます。手順を以下に示します。

1. 追加したいコンポーネントを持つカタログを選択する
2. コンポーネント欄から追加したいコンポーネントを選択する
3. [カット]か[コピー]コマンドを選択する
4. コンポーネントの追加先カタログを選択する
5. [ペースト]コマンドを選択する

オブジェクトインスペクタ

オブジェクトインスペクタは選択したオブジェクトの属性やイベントを表示します。ポップアップメニュー、ダイアログ、値の入力などでオブジェクトの属性を編集できます。オブジェクトインスペクタを表示するにはウィンドウメニューの[オブジェクトインスペクタ]を選択します(図 13.16)。

図 13.16 オブジェクトインスペクタ



以下のオプションがあります。

[オブジェクトポップアップメニュー](#)

[プロパティタブ](#)

[イベントタブ](#)

[オブジェクトインスペクタのコンテキストメニュー](#)

オブジェクトポップアップメニュー

オブジェクトインスペクタは[オブジェクト]ポップアップメニューに名前があるオブジェクトの情報を表示します。選択したオブジェクトの階層にある他の項目を調べるには、[オブジェクト] ポップアップメニューでその名前を選択します。

矢印キーでレイアウト上のオブジェクトを選択できます。[オブジェクト]ポップアップメニューをクリックした後、 キーと キーで選択項目を上下に移動できます。Enter キーで選択を決定します。

プロパティタブ

選択したオブジェクトの属性を見るには [プロパティ] タブをクリックします。オブジェクトインスペクタが属性のリストを表示します。属性の名前と現在の値を表示します。

属性の内容を拡張表示するには、階層コントロールをクリックします。再度クリックすると縮小します。リストの列のサイズを変更するには、列の間にカーソルを置いてドラッグし希望の位置でカーソルを離します。

属性の値を変更するには、それをクリックします。選択した属性が枠線で囲まれて強調表示されます。ここに新しい値を入力します。

現在の値の横にあるボタンをクリックして変更することもできます。



このボタンをクリックするとポップアップメニューが現れます。現在の属性の値を変更するには、このポップアップメニューから項目を選択します。



このボタンをクリックするとダイアログが現れます。ダイアログのオプションを利用して現在の属性の値を変更します。現在の RAD プラグインがダイアログの内容を決定します。

属性の値の変更を中止するには、Esc キーを押します。オブジェクトインスペクタは変更を破棄します。

矢印キーでコンポーネントの属性を移動できます。オブジェクトインスペクタで属性を選択しているとき、`↑` キーと `↓` キーで選択項目を上下に移動できます。

ヒント： `↑` キーと `↓` キーはあらかじめ設定された値をトグルします。例えばブール値を選択した場合、`↑` キーと `↓` キーで現在の値を [True] または [False] に切り替えます。

イベントタブ

[イベント] タブは選択したコンポーネントのイベントについての情報を表示します。デフォルトではすべてのイベントはオフです。イベントを変更した後、RAD ツールは自動的にソースコードを生成します。このコードはイベント処理用の構造化フレームワークを提供します。オブジェクトの挙動とその他のユーザーインターフェースオブジェクトとの相互作用を決定するコードは自分で記述します。

イベントを変更するには、空のフィールドをクリックします。イベントのデフォルト名が強調表示されます。ここで新しい名前を入力します。

ヒント： 新しいイベントを作成するには、空のフィールドをダブルクリックします。

変更した後、Enter キーを押すと RAD ツールは自動的にソースコードを生成します。エディタウィンドウが開き、生成したコードを表示します。選択したイベントの名前が強調表示

されます。[例 13.1](#) は Java AWT の checkbox コンポーネントの `componentResized` イベントが生成したコードです。

例 13.1 生成されたコード

```
public
void checkbox1 componentResized( java.awt.event.ComponentEvent e)
{
}
```

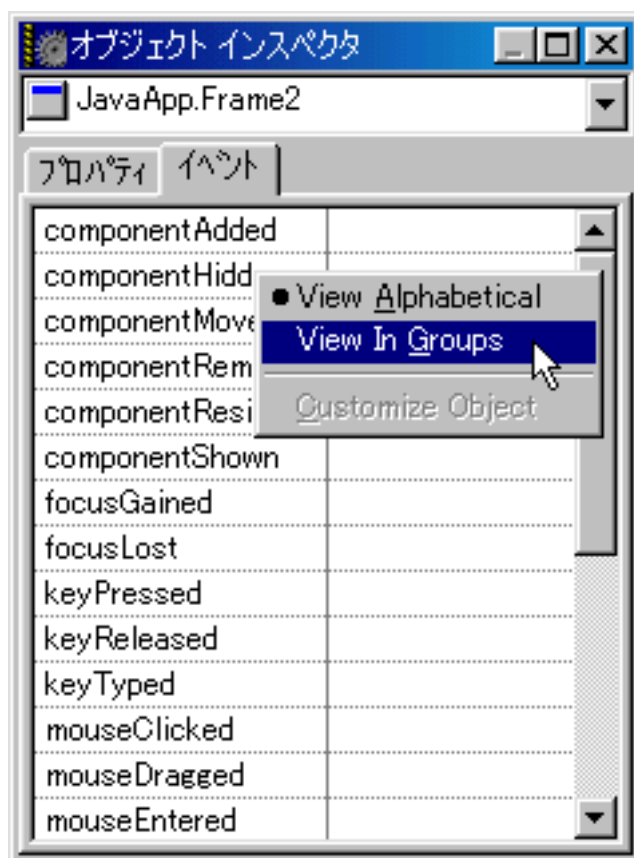
コンポーネントの挙動を指定するコードを書かなくてはなりません。例えば[例 13.1](#) の生成されたコードの括弧の中にはコードはありません。アプリケーション内でのコンポーネントの挙動をコントロールするためにはこの括弧内にコードを書く必要があります。ファイルメニューの [保存] または Ctrl + S キーで変更を保存します。

イベントの変更を中止するには、Esc キーを押します。オブジェクトインスペクタは変更を破棄します。

オブジェクトインスペクタのコンテキストメニュー

オブジェクトインスペクタのコンテキストメニュー ([図 13.17](#)) にオブジェクトを操作するためのコマンドがあります。選択したオブジェクトによってコマンドは異なります。

図 13.17 オブジェクトインスペクタのコンテキストメニュー



以下の操作でオブジェクトインスペクタのコンテキストメニューを表示します。

項目を右クリックする

オブジェクトインスペクタのコンテキストメニューのコマンドについて説明します。

Select All : 現在のフィールドの内容を選択する

Delete : 現在のフィールドの内容を削除する

Customize Object : オブジェクトの属性を変更する

View Alphabetical : イベントビューのオブジェクトイベントを、属するグループに関係なくアルファベット順に表示する

View In Groups : イベントビューのグループ階層にあるオブジェクトイベントを表示する

第 14 章 RAD コンポーネント

CodeWarrior の Java 用 RAD (Rapid Application Development) ツールは、コンポーネントを用いてアプリケーションのユーザーインターフェースを作成します。コンポーネントはインターフェースのそれぞれの要素 (ボタン、スクロールバー、チェックボックスなど) を指定します。ここでは RAD ツールに含まれるコンポーネントを説明します。

[Java AWT コンポーネント](#)








[Java Swing コンポーネント](#)







[コンポーネントエディタ](#)

Java AWT コンポーネント

[表 14.1](#) は AWT コンポーネントカタログに含まれるコンポーネントを示します。アイコンは[コンポーネントパレット](#)と[コンポーネントカタログウィンドウ](#)に表示されます。

表 14.1 Java AWT コンポーネント







アイコン	説明
	Button : ユーザーインターフェースに表示されるクリック可能なボタン
	Canvas : その他のユーザーインターフェース要素のコンテナ
	Checkbox : オン、オフを設定するチェックボックス
	Choice : 選択可能なオプションを示すポップアップメニュー
	Label : 他のユーザーインターフェース要素を識別または記述するためのテキスト領域
	List : オプションを選択するスクロールリスト
	MenuBar : 実際のアプリケーションとは表示が異なり、レイアウトウィンドウにアイコンを表示する (アイコンをダブルクリックすると メニューエディタ が開く)

アイコン	説明
	Panel : さまざまなユーザーインターフェース要素を配置するパネル
	PopupMenu : 実際のアプリケーションとは表示が異なり、レイアウトウィンドウにアイコンを表示する (アイコンをダブルクリックすると ポップアップメニューエディタ が開く)
	Scrollbar : ユーザーインターフェース内でオプションを調整するためのスクロールバー
	ScrollPane : 入力されたテキストが欄の境界線を越えると、スクロールバーを作る欄
	TextArea : ユーザーインターフェース内で複数行のテキストを入力できる領域
	TextField : ユーザーインターフェース内で 1 行のテキストを入力できる領域





Java Swing コンポーネント

[表 14.2](#) は Swing コンポーネントカタログに含まれるコンポーネントを示します。大部分のコンポーネントは AWT コンポーネントと同じですが、より拡張性が高いです。アイコンは [コンポーネントパレット](#) と [コンポーネントカタログウィンドウ](#) に表示されます。

表 14.2 Java Swing コンポーネント

アイコン	説明
	JButton : ユーザーインターフェース内でクリック可能なボタン
	JCheckBox : オン、オフを設定するチェックボックス
	JComboBox : 名前の入力またはポップアップリストでオプションを選択可能なコンビネーションボックス
	JEditorPane : 必要な編集作業へその特性を適用する編集欄
	JLabel : 他のユーザーインターフェース要素を識別または記述するためのテキスト領域
	JList : オプションを選択できるスクロールリスト

アイコン	説明
	JMenuBar：実際のアプリケーションとは表示が異なり、レイアウトウィンドウにアイコンを表示する（アイコンをダブルクリックすると メニューエディタ が開く）
	JOptionPane：警告ダイアログやユーザーに入力を要求するダイアログ
	JPanel：さまざまなユーザーインターフェース要素を配置するパネル
	JPasswordField：パスワードを入力するための 1 行のテキスト領域（画面上には入力した文字は表示されない）
	JPopupMenu：実際のアプリケーションとは表示が異なり、レイアウトウィンドウにアイコンを表示する（アイコンをダブルクリックすると ポップアップメニューエディタ が開く）
	JProgressBar：実行中のタスクの進行状況をパーセンテージで表示するバー
	JRadioButton：片方だけがオンになるラジオボタン
	JScrollBar：ユーザーインターフェース内でオプションを調整するためのスクロールバー
	JScrollPane：入力されたテキストが欄の境界線を越えると、スクロールバーを作る欄
	JSlider：スライダーコントロールでオプションを選択する画面上のメカニズム
	JSplitPane：2 つのユーザーインターフェース要素のサイズと表示をコントロールする方法
	JTabbedPane：コントロールタブでアクセス可能な複数に分割された欄
	JTable：列と行を持つ表
	JTextArea：ユーザーインターフェース内で複数行のテキストを入力できる領域
	JTextField：ユーザーインターフェース内で 1 行のテキストを入力できる領域

アイコン	説明
	JTextPane : グラフィック属性を持つテキスト領域
	JToggleButton : 2 種類の状態を切り替えるトグルボタン
	JToolBar : レイアウトウィンドウ内のツールバー
	JTree : ユーザーインターフェース内の階層コントロール

コンポーネントエディタ

あるコンポーネント ([表 14.1](#) の Menu Bar、Popup Menu、[表 14.2](#) の JMenu Bar、JPopup Menu など) は、実際のインターフェースではなくユーザーインターフェースを表すアイコンをレイアウトウィンドウに表示します。このようなコンポーネントのアプリケーションでの表示方法を変更するには別のエディタが必要です。

例えば、Java AWT MenuBar コンポーネントはレイアウトウィンドウに四角いアイコンを表示します。このアイコンはレイアウトウィンドウ上ではメニューバーを表示しません。代わりにこのアイコンをダブルクリックするとメニューエディタが開きます。メニューエディタはメニューバーの本当の形を表示します。

ここでは CodeWarrior IDE のカタログにあるコンポーネントエディタについて説明します。

[メニューエディタ](#)

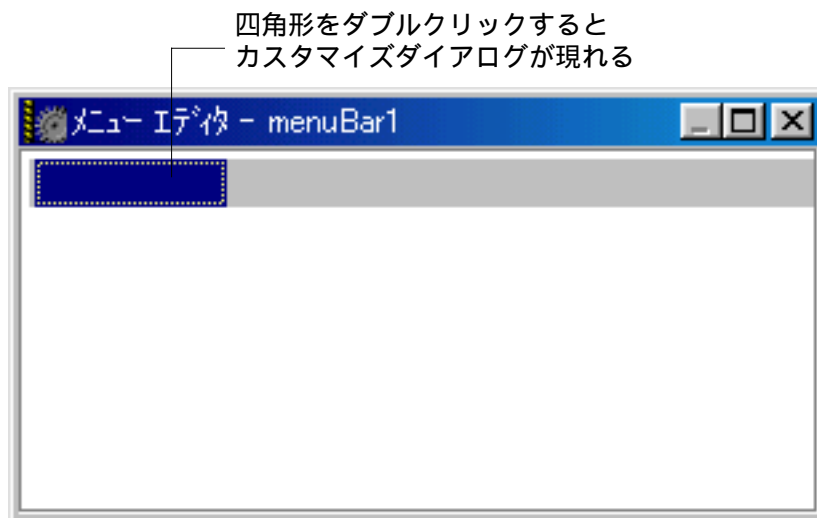
[ポップアップメニューエディタ](#)

メニューエディタ

メニューエディタを使って、アプリケーションのメニューバーとメニューコマンド階層の作成と変更を行います。メニューエディタは実際のアプリケーションと同様に、メニューバーを画面上に表示します。

レイアウトウィンドウでメニューエディタを示すアイコンをダブルクリックしてください。最初メニューエディタは空白です ([図 14.1](#))。メニュー階層をビルドすると、メニューエディタはそれを反映して画面を更新します。

図 14.1 メニューエディタ



以下の項目について説明します。

[メニューアイテムを作成](#)

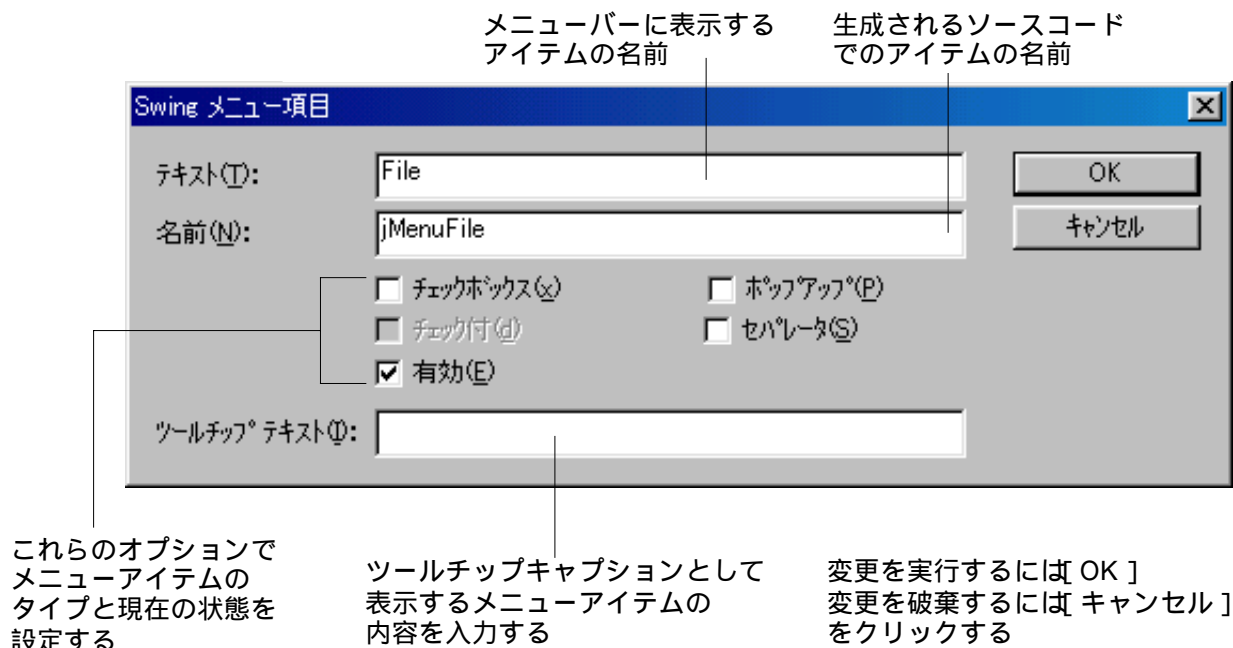
[メニューアイテムの変更](#)

[メニューアイテムを削除](#)

メニューアイテムを作成

アプリケーションメニューバーのメニューアイテムを作成するには、メニューエディタの四角い枠線をダブルクリックします。メニューアイテム用のダイアログが現れます。ダイアログのオプションは選択しているカタログによって異なります。例えばメニューエディタで Java Swing メニューバーを作成する場合、[図 14.2](#) のようなダイアログが現れます。

図 14.2 Java Swing メニューアイテムのメニューエディタダイアログ

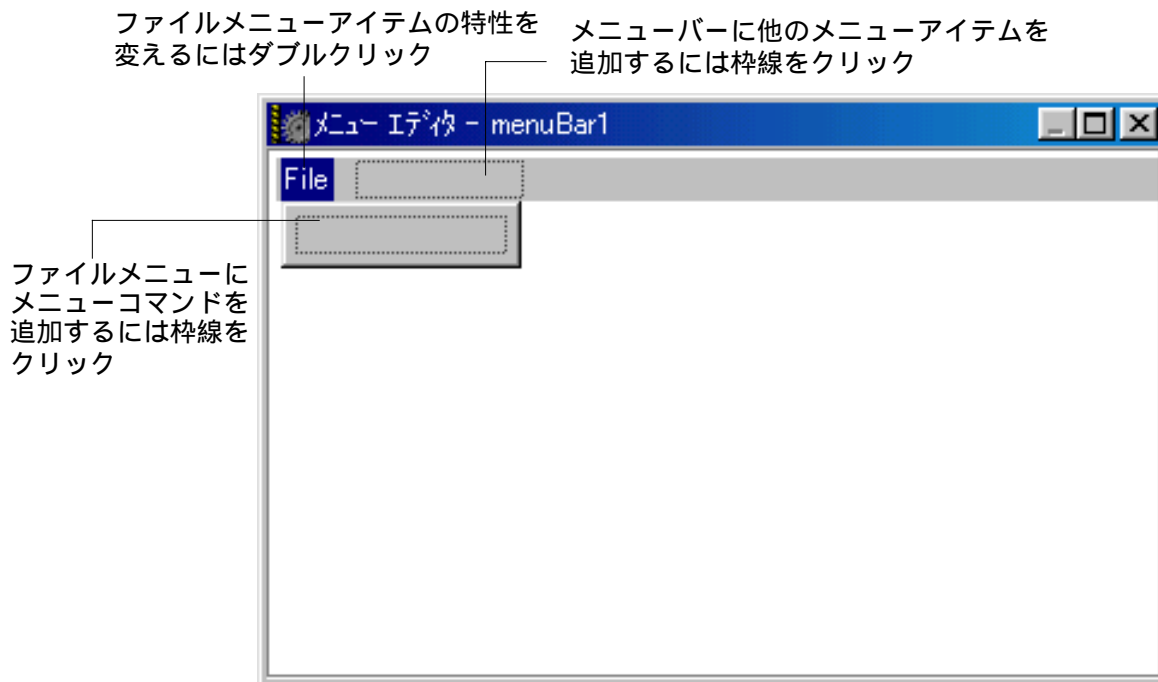


ダイアログに情報を入力したら、[OK] ボタンをクリックします。メニューエディタはメニューバーを更新します。[テキスト] フィールドに「File」と入力して[OK] ボタンをクリックした場合、メニューバーにファイルメニューというアイテムが表示されます。枠線はファイルメニューの右に移動します。

メニューアイテムの変更

アプリケーションのメニューバーを変更するには、既存のメニューアイテムをダブルクリックするか、または枠線をクリックします。ファイルメニューアイテム（[図 14.3](#)）をクリックすると、ファイルメニューの下に枠線が現れます。このようにして各メニューバーのアイテムを編集します。

図 14.3 メニューエディタでアイテムを変更



メニューバーの変更を終えたら、メニューエディタウィンドウを閉じます。レイアウトウィンドウに表示されるのは元のアイコンですが、メニューバーへの変更は残っています。メニューバーへの変更をすぐに保存するには、メニューエディタウィンドウを閉じてからファイルメニューの [保存] を選択してください。

メニューアイテムを削除

メニューアイテムを削除するには、メニューエディタウィンドウ上でその名前をクリックして Delete キーを押します。メニューエディタはメニューアイテムを削除し、画面を更新します。メニューバーからメニューアイテムを削除すると、それに関連するコマンドも削除されます。

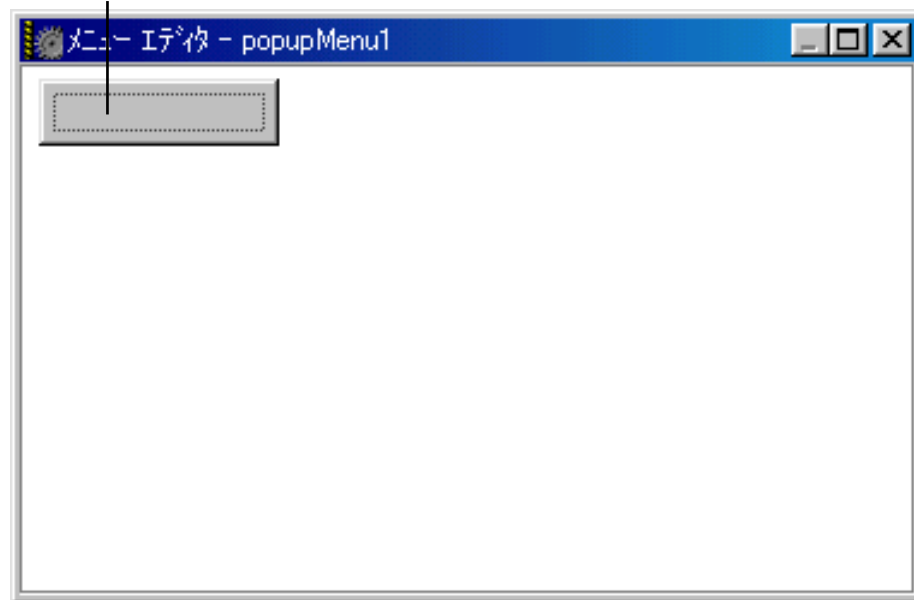
注意： メニューバーからメニューアイテムを削除するとき、その変更を反映させるためにメニューエディタを一度閉じてから再度開いてください。

ポップアップメニューエディタ

ポップアップメニューエディタの挙動はメニューエディタとよく似ています。違いはポップアップメニューエディタは一度に 1 つのポップアップメニューしか処理できないということだけです。図 14.4 はポップアップメニューエディタの初期状態を示します。

図 14.4 ポップアップメニューエディタ

ポップアップメニューにメニューコマンドを
追加するには枠線をダブルクリックする



ポップアップメニューエディタのメニューアイテムの作成、変更、削除については「[メニューエディタ](#)」(p418) を参照してください。ポップアップメニューエディタとメニューエディタの操作方法は同じです。

第 15 章 RAD ブラウザ

CodeWarrior ブラウザで Java 用 RAD (Rapid Application Development) プロジェクトをブラウズすることができます。ブラウザデータベースを生成するようにプロジェクトを設定すると、IDE は自動的に RAD 特有の情報をデータベースに追加します。ブラウザウィンドウのコントロールタブで RAD 特有の情報を見ることができます。

この章は一般的なブラウザ機能についての知識をお持ちの方を対象に書かれています。プロジェクトのブラウザ設定については「[ソースコードのブラウズ](#)」(p143) を参照してください。ここでは RAD 特有のブラウザ機能について説明します。

[ブラウザウィンドウの RAD 機能](#)

[RAD ダイアログ](#)

ブラウザウィンドウの RAD 機能

CodeWarrior ブラウザで RAD プロジェクトのブラウザデータベースを見ているとき、以下の機能を利用することができます。

[コントロールタブ](#)

[プロパティビュー](#)

[メソッドビュー](#)

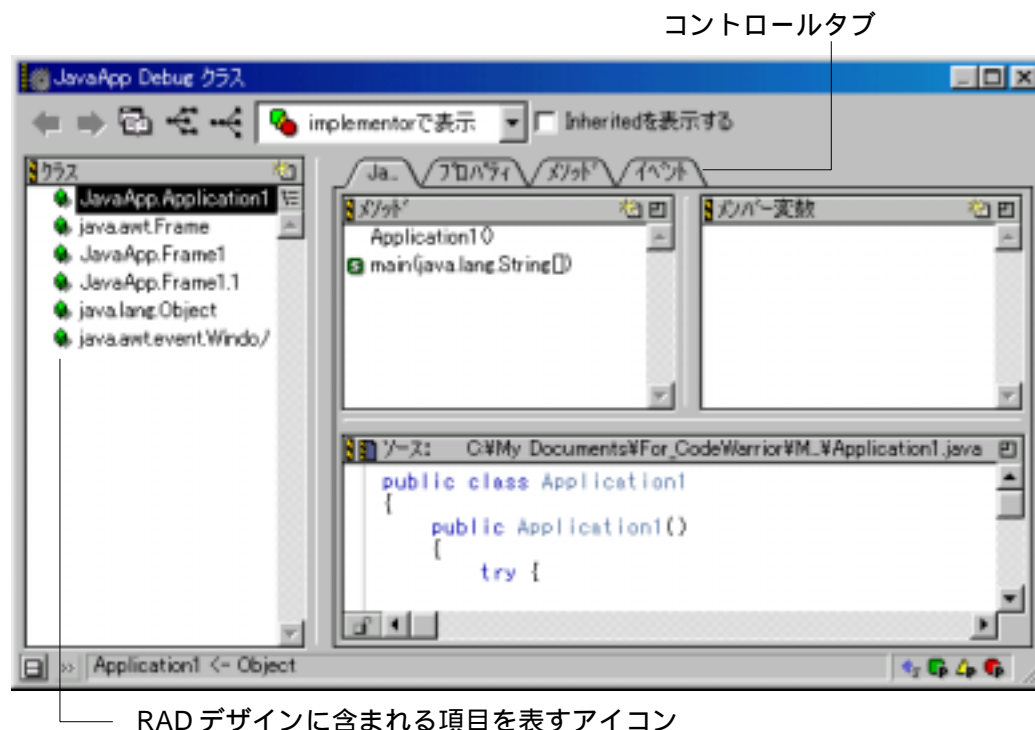
[イベントビュー](#)

コントロールタブ

コントロールタブは RAD コンポーネントがサポートする項目 (属性、メソッド、イベントなど) をブラウズするのに役立ちます。

コントロールタブでビューを選択します。各タブをクリックするとそのビューが表示されます。例えば [Java] タブをクリックすると Java ビューが表示されます ([図 15.1](#))。それぞれのビューは [クラス] 欄で選択したコンポーネントに関する Java RAD 情報を表示します。

図 15.1 Java RAD プロジェクトのブラウザウィンドウ

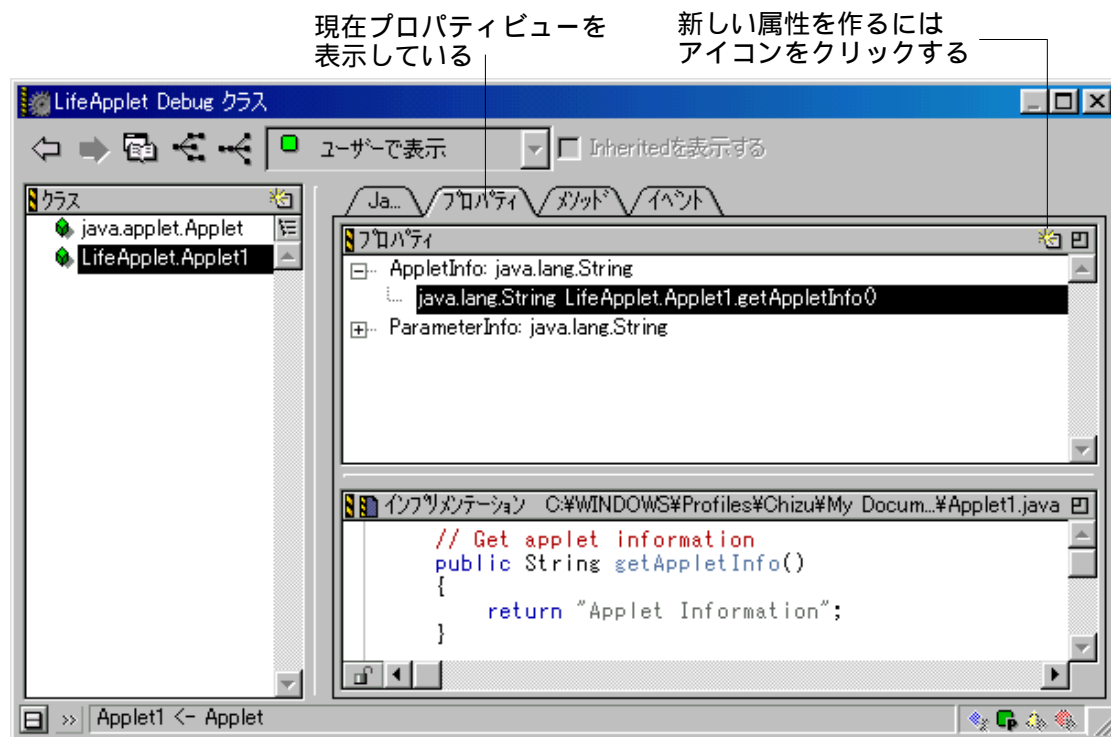


[プロパティ] [メソッド] [イベント] タブのいずれかをクリックすると、他のメンバ関数から継承していないパブリックデータの情報を対応するビューに表示します。この場合 [アクセスフィルタポップアップメニュー](#) と [Inherited を表示] チェックボックスは選択不可になり、設定を変更できません ([図 15.2](#)、[図 15.3](#)、[図 15.4](#))。

プロパティビュー

プロパティビューは [クラス] 欄で選択したコンポーネントに関する属性情報を表示します ([図 15.2](#))。このビューには [プロパティ] と [インプリメンテーション] 欄があります。

図 15.2 プロパティビュー



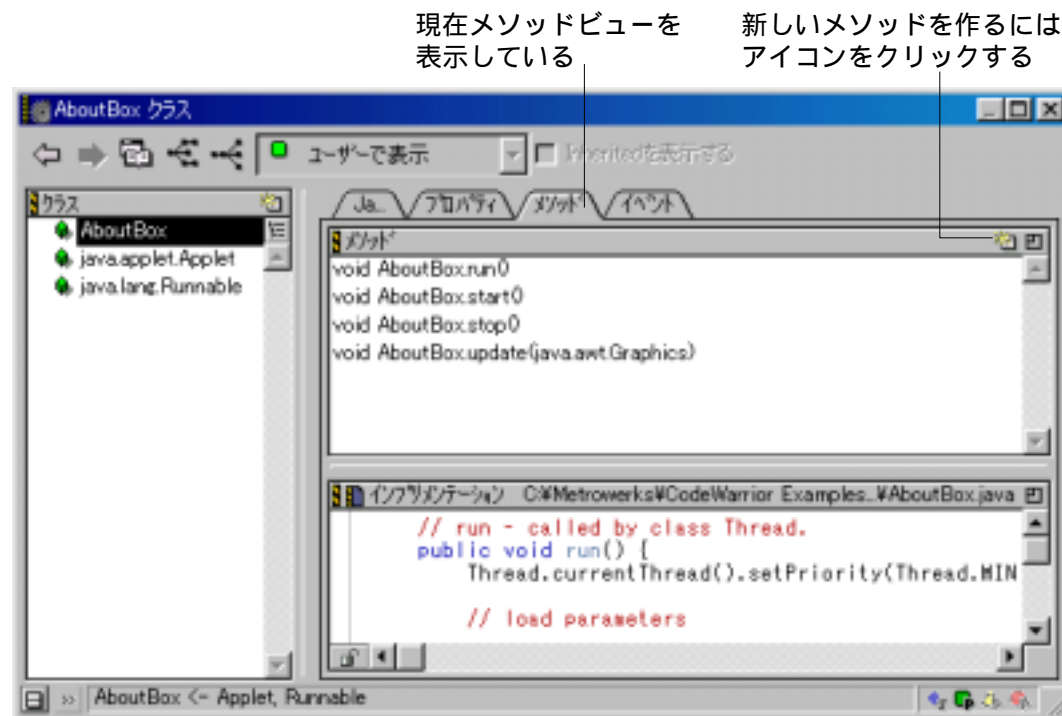
[プロパティ] 欄は選択したコンポーネントの属性をリスト表示します。このリストで選択した属性の実装が [インプリメンテーション] 欄に表示されます。新しい属性の作り方は「[新規プロパティダイアログ](#)」(p427) を参照してください。

[インプリメンテーション] 欄は[ソース欄](#)と似ています。選択した属性を実装するソースコードが表示されます。この欄でソースコードを編集できます。欄の上に実装コードを含むファイルへのパスが表示されます。

メソッドビュー

メソッドビューは [クラス] 欄で選択したコンポーネントのメソッドに関する情報を表示します([図 15.3](#))。このビューには [メソッド] と [インプリメンテーション] 欄があります。

図 15.3 メソッドビュー



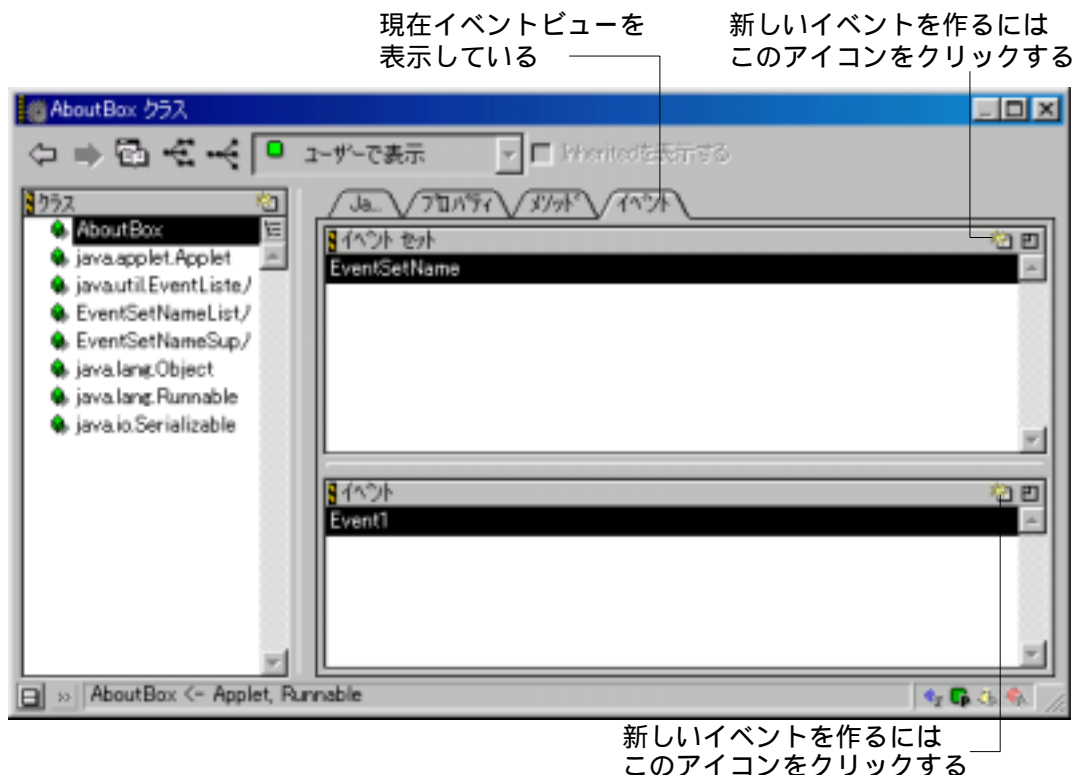
[メソッド] 欄は選択したコンポーネントのメソッドをリスト表示します。このリストで選択したメソッドの実装が [インプリメンテーション] 欄に表示されます。新しい属性の作り方は「[新規メソッドダイアログ](#)」(p430) を参照してください。

[インプリメンテーション] 欄は[ソース欄](#)と似ています。選択したメソッドを実装するソースコードが表示されます。この欄でソースコードを編集できます。欄の上に実装コードを含むファイルへのパスが表示されます。

イベントビュー

イベントビューは [クラス] 欄で選択したコンポーネントのイベントに関する情報を表示します ([図 15.4](#))。このビューには [イベントセット] と [イベント] 欄があります。

図 15.4 イベントビュー



[イベントセット] 欄は選択したコンポーネントのイベントグループを表示します。[イベント] 欄は選択したイベントセットの個々のイベントを表示します。新しいイベントセットの作り方は「[新規イベントセットダイアログ](#)」(p433)、イベントの作り方は「[新規イベントダイアログ](#)」(p439)を参照してください。

RAD ダイアログ

RAD プロジェクトのコンポーネントに対して新しい属性、メソッド、イベントセット、イベントを作成するとき、IDE はダイアログを表示します。ダイアログによって異なるオプションがあります。以下の項目について説明します。

[新規プロパティダイアログ](#)


[新規メソッドダイアログ](#)

[新規イベントセットダイアログ](#)

[新規イベントダイアログ](#)

新規プロパティダイアログ

「新規プロパティ」ダイアログを利用して、ブラウザウィンドウで選択したコンポーネントに対して新しい属性を作成します。このダイアログを表示するには、ブラウザウィンドウ

を最前面にしてからブラウザメニューの[[新規プロパティ](#)]を選択してください。または、新アイテムアイコン()をクリックしても表示されます([図 15.2 \(p425\)](#))。[新規プロパティ] コマンドは使用しているフレームワークによって名前が変わります。例えば Java RAD プロジェクトを作成しているとき、コマンド名は[新規 Bean プロパティ]になります。

「新規プロパティ」ダイアログ([図 15.5](#))には以下のオプションがあります。

[名前](#)

[型](#)

[型に必要なパッケージ](#)

[アクセッサ](#)

[データメンバあり](#)

[一覧](#)

[追加とキャンセル](#)

名前

[名前] フィールドに新しい属性の名前を入力します。フレームワークプログラミング言語の命名規約に従った名前を付けてください。名前にスペースを使用できない言語もあります。

型

[型] フィールドに属性のタイプを入力します。フレームワークプログラミング言語で有効なタイプを入力してください。

型に必要なパッケージ

必要であれば [型に必要なパッケージ] フィールドに属性のタイプのパッケージを入力します。

図 15.5 「新規プロパティ」ダイアログ

新規 Java プロパティ

プロパティを追加するクラス:
AboutBox

名前:
Property1

型:
void

型のために必要なパッケージ(オプション):

アクセッサ:
☒ ゲッター ☒ セッター

☒ データメンバーあり:
名前: DataMbr
初期化設定子: "text"
☐ Transient ☐ Volatile

一覧:
ゲッター: getProperty1
セッター: setProperty1
データメンバー: public void DataMbr = "text";

追加 キャンセル

アクセッサ

[アクセッサ] チェックボックスは属性の先祖の特性を指定します。

データメンバーあり

[データメンバーあり] チェックボックスがオンの場合、属性がデータメンバーを持つことができます。オンの場合、[名前]と[初期設定子]フィールド、[Transient]と[Volatile]

チェックボックスが利用可能になります。[名前] フィールドにデータメンバ名を入力します。その他の情報も設定します。


一覧

[一覧] 欄はダイアログに設定された情報を一覧表示します。値やチェックボックスを変更すると、[一覧] 欄の内容もそれに合わせて変化します。この欄に情報が表示されるのは、[名前] と [型] フィールドの入力を終え、[アクセッサ] を設定した後です。

追加とキャンセル

現在の設定で新しい属性を追加するには [追加] ボタンをクリックします。変更を破棄するには [キャンセル] ボタンをクリックします。

新規メソッドダイアログ

「新規メソッド」ダイアログを利用して、ブラウザウィンドウで選択したコンポーネントに対して新しいメソッドを作成します。このダイアログを表示するには、ブラウザウィンドウを最前面にしてからブラウザメニューの [[新規メソッド](#)] を選択してください。新アイテムアイコン () をクリックしても表示されます ([図 15.3 \(p426\)](#))。[新規メソッド] コマンドは使用しているフレームワークによって名前が変わります。例えば Java RAD プロジェクトを作成しているとき、コマンド名は [新規 Bean メソッド] になります。

「新規メソッド」ダイアログ ([図 15.6](#)) には以下のオプションがあります。

[名前](#)

[戻り値型](#)

[引数](#)

[追加 throw](#)

[引数に必要なパッケージ](#)

[修飾子](#)

[宣言](#)

[追加とキャンセル](#)

名前

[名前] フィールドに新しいメソッドの名前を入力します。フレームワークプログラミング言語の命名規約に従った名前を付けてください。名前にスペースを使用できない言語もあります。

戻り値型

[戻り値型] フィールドにメソッドの戻り値の型を入力します。フレームワークプログラミング言語で有効な型を入力してください。

引数

必要であれば [引数] フィールドにメソッドの引数を入力します。サンプルの引数が表示されます。

追加 throw

必要であれば [追加 throw] フィールドに例外処理情報 (*throws*) を入力します。サンプルの *throws* が表示されます。

引数に必要なパッケージ

必要であれば [引数に必要なパッケージ] フィールドにパラメータのパッケージを入力します。サンプルのパッケージが表示されます。

図 15.6 「新規メソッド」ダイアログ

新規 Java Bean メソッド

メソッドを追加するクラス:

名前:

戻り値型:

引数: 例: int a, java.util.Vector b, ...

Throw (オプション): 例: java.io.IOException, YourPkg.YourExcpt, ...

引数のために必要なパッケージ(オプション): 例: java.util.*, ...

修飾子

アクセス: 指定子:

☐ Native ☐ Synchronized

宣言:

修飾子

[アクセス] と [指定子] ポップアップメニューで新しいメソッドのアクセスレベルとメソッド修飾子を指定します。選択可能なアクセスレベルは [Public] [Protected]、[Private] です。選択可能な修飾子は [なし]、[Abstrac]、[Final]、[Static] です。さらに詳しくメソッドの修飾子を指定するには、[Native] または [Synchronized] チェックボックスをオンにします。

宣言

[宣言] 欄はダイアログで現在指定された形式の宣言を表示します。値やチェックボックスを変更すると、[宣言] 欄の内容もそれに合わせて変化します。この欄に情報が表示されるのは、[名前] と [型] フィールドの入力を終え、[アクセス] レベルを設定した後です。

追加とキャンセル

現在の設定で新しいメソッドを追加するには [追加] ボタンをクリックします。変更を破棄するには [キャンセル] ボタンをクリックします。

新規イベントセットダイアログ


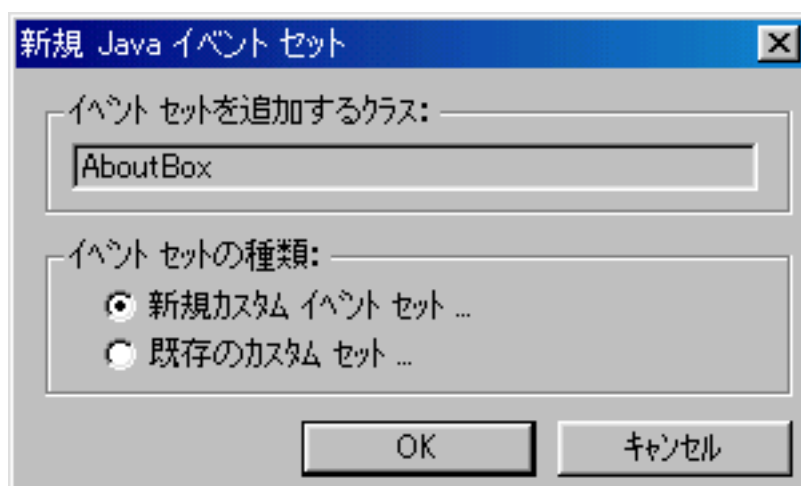
「新規イベントセット」ダイアログを利用して、ブラウザウィンドウで選択したコンポーネントに対して新しいイベントセットを作成します。このダイアログを表示するには、ブラウザウィンドウを最前面にしてからブラウザメニューの [[新規イベントセット](#)] を選択してください。新アイテムアイコン () をクリックしても表示されます ([図 15.4 \(p427\)](#))。[新規イベントセット] コマンドは使用しているフレームワークによって名前が変わります。例えば Java RAD プロジェクトを作成しているとき、コマンド名は [新規 Bean イベントセット] になります。

図 15.7 「新規イベントセット」ダイアログ



「新規イベントセット」ダイアログ ([図 15.7](#)) には以下のオプションがあります。

[新規カスタムイベントセット](#)

[既存のイベントセット](#)

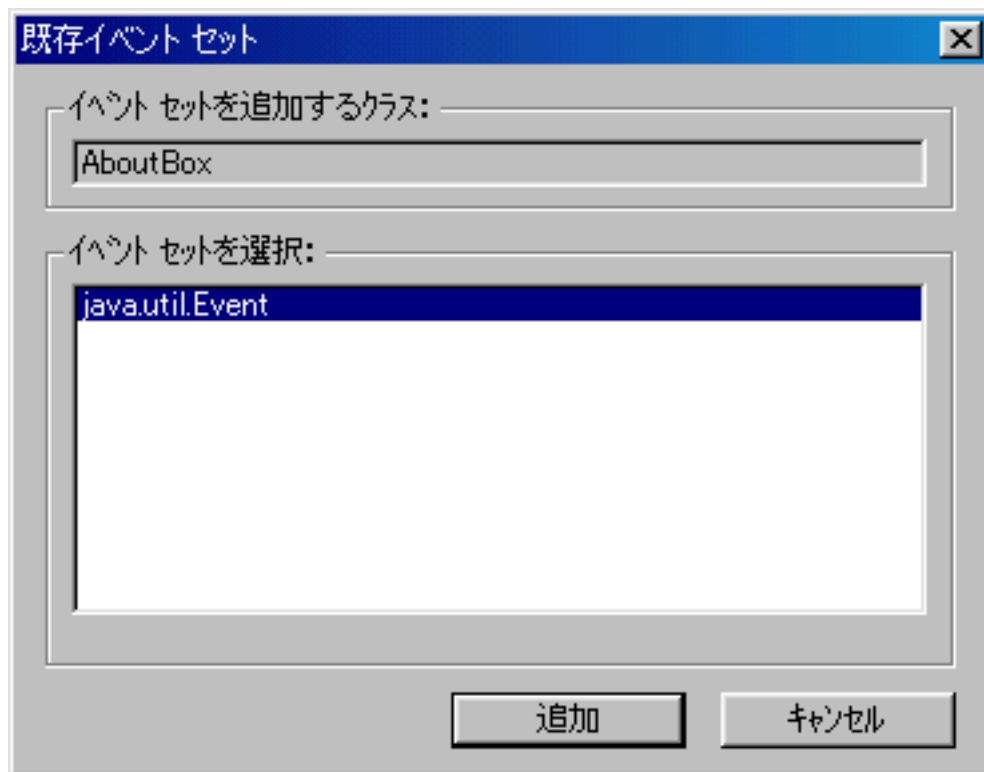
新規カスタムイベントセット

[新規カスタムイベントセット] オプションをオンにして [OK] ボタンをクリックすると、カスタムイベントセットを作成するためのウィザードが現れます。「[新規 Java イベントセットウィザード](#)」 ([p434](#)) を参照してください。

既存のイベントセット

[既存のイベントセット] オプションをオンにして [OK] ボタンをクリックすると、「新規イベントセット」ダイアログ ([図 15.8](#)) が現れます。追加したいイベントセットをリストから選択して [追加] ボタンをクリックしてください。ブラウザウィンドウで選択したコンポーネントへ、既存のイベントセットが追加されます。

図 15.8 既存のイベントセットを追加



新規 Java イベントセットウィザード

[新規イベントセットダイアログ](#)には[新規カスタムイベントセット](#)を作るためのオプションがあります。オプションを設定して [OK] ボタンをクリックすると、新規 Java イベントセットウィザードが現れます。

ウィザードには以下のナビゲーションボタンがあります。

- 戻る：前のページへ戻る
- 次へ：次のページへ進む
- 完了：現在の情報を表示する
- キャンセル：すべての変更を破棄する

新規 Java イベントセットウィザードでは以下のステップを実行します。

1. [新しいイベントセットの名前と場所を指定する](#)

2. [基底クラスと実装リストを指定する](#)
3. [新しいイベントセットをビルドターゲットに割り当てる](#)

ウィザードは一連のページを表示します。順番に従ってページで情報を設定し、終了したら [次へ] ボタンをクリックします。

現在のウィザードの設定を適用して新しいイベントセットを作るには [完了] ボタンをクリックします。ウィザードはイベントセットの現在の設定情報を表示します。ウィザードに戻って設定の変更を続けるには [キャンセル] ボタンをクリックします。

以下は新規 Java イベントセットウィザードを使う手順です。

1. 新しいイベントセットの名前と場所を指定する

ウィザードのこのページ([図 15.9](#))では、新しいイベントの名前、場所、修飾子を指定します。

このパネルには以下のオプションがあります。

[クラス名](#)

[ファイル](#)

[パッケージ](#)

[修飾子](#)

クラス名

[クラス名] フィールドにイベントセットのクラス名を入力します。サンプル名が表示されます。下にある [Bean のクラス] チェックボックスは Java ビーン専用のオプションであるため使用できません。

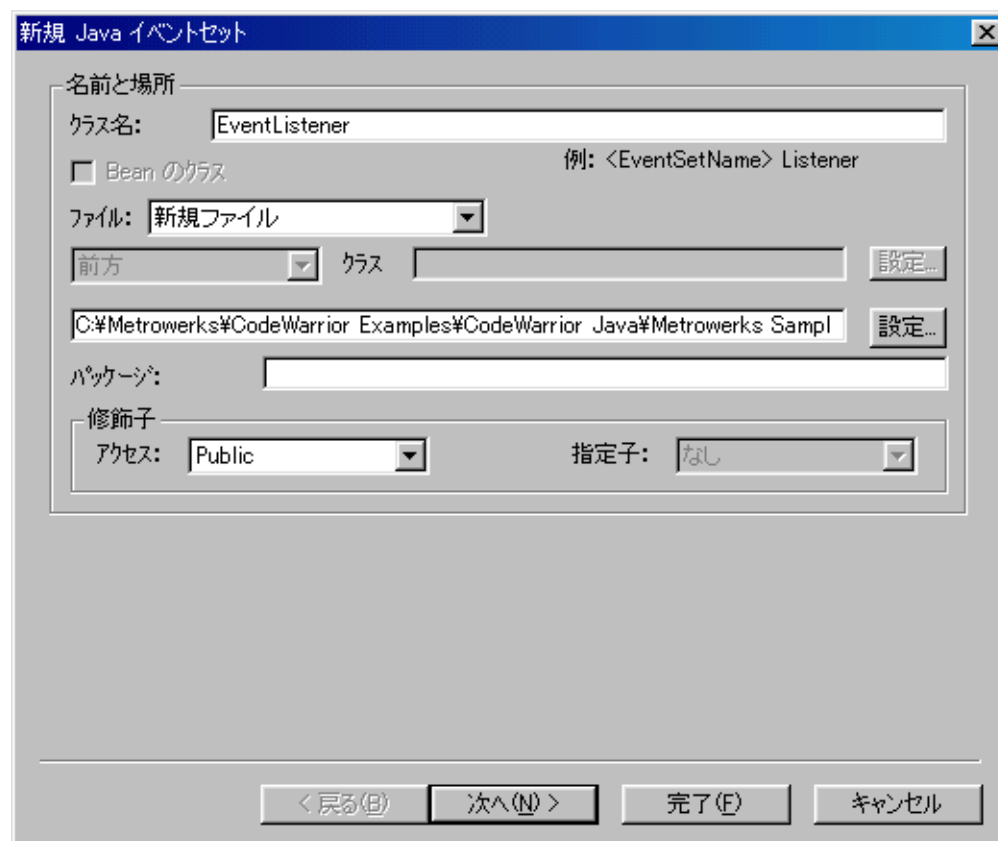
ファイル

[ファイル] ポップアップメニューでイベントセットの種類を選択します。

新規ファイル：新規イベントセットファイル

クラスに対応：プロジェクトの既存のクラスに対応するイベントセット

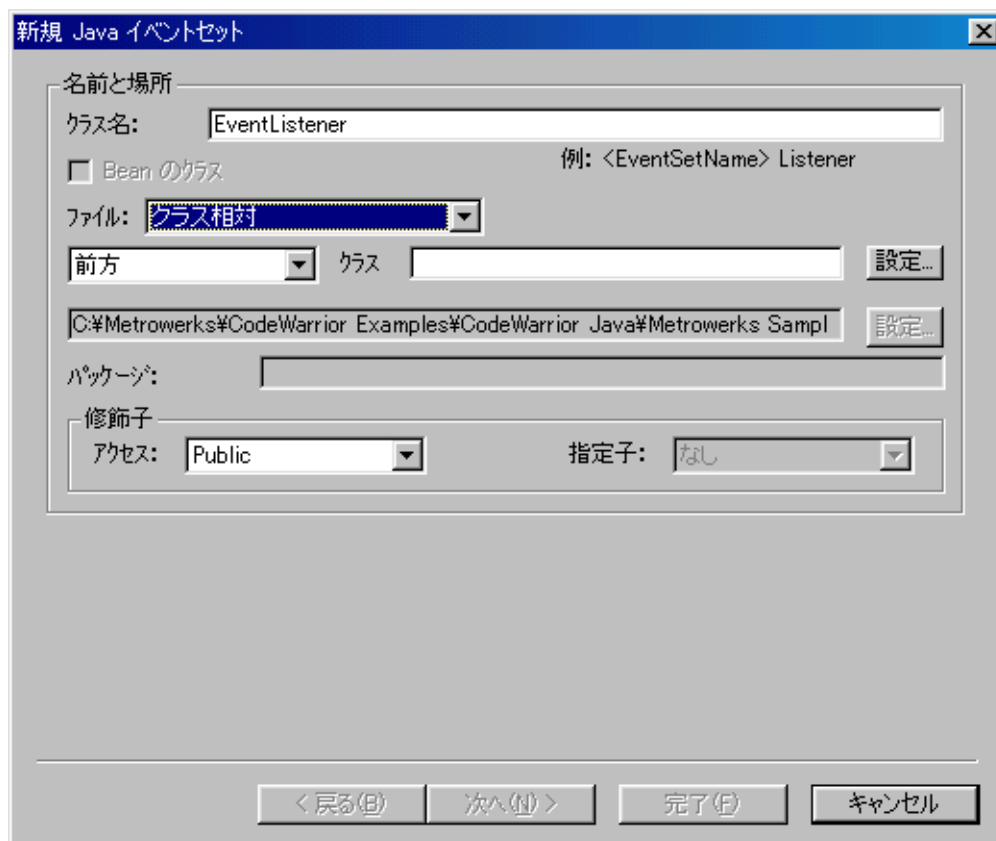
図 15.9 新規 Java イベントセットウィザード：名前と場所



選択したオプションによって、[ファイル] ポップアップメニューの下に表示されるフィールドが変わります。[新規ファイル] を選択した場合は図 15.9、[クラスに対応] の場合は図 15.10 になります。

[新規ファイル] を選択した場合、フィールドにそのファイルの保存場所を入力します。代わりに [設定] ボタンをクリックして「保存」ダイアログを開き、そこで保存場所を指定することもできます。

図 15.10 [クラスに対応] を選択した場合



[クラスに対応] を選択した場合、[クラス] フィールドとポップアップメニューが利用可能になります (図 15.10)。[クラス] フィールドにイベントセットと関連づけるクラス名を入力します。代わりに [設定] ボタンをクリックして「保存」ダイアログを開き、そこでクラスを選択することもできます。次にポップアップメニューでイベントセットをどこに置くのかを指定します。クラス [前方] [後方] [内部] のいずれかを選択します。

パッケージ

必要であれば [パッケージ] フィールドにイベントセットのパッケージを入力します。[ファイル] ポップアップメニューで [クラスに対応] オプションを選択した場合、このオプションは使用不可になります。」

修飾子

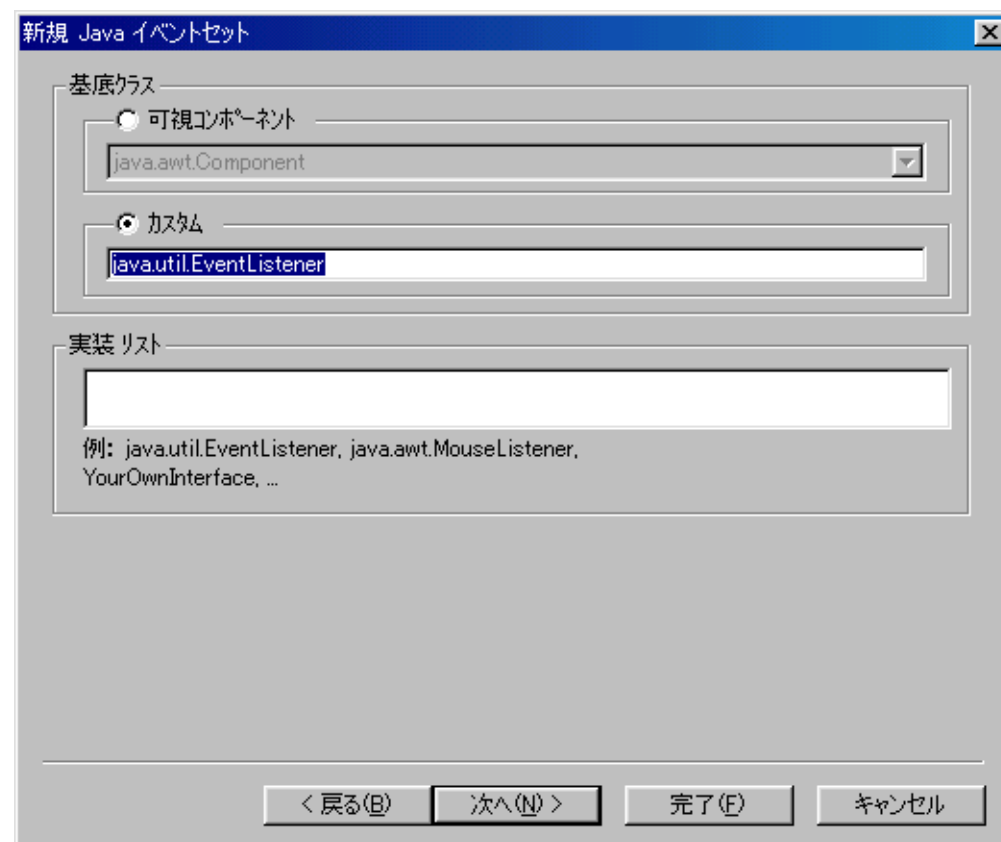
[アクセス] と [指定子] ポップアップメニューで新しいイベントセットのアクセスレベルとメソッド修飾子を指定します。選択可能なアクセスレベルは [Public]、[Protected]、[Private] です。選択可能な修飾子は [なし] [Abstract] [Final] [Static] です。さらに詳しくメソッドの修飾子を指定するには、[ネイティブ] または [同期化する] チェックボックスをオンにします。

2. 基底クラスと実装リストを指定する

[図 15.11](#) のウィザードのページで新規イベントセットの基底クラスと実装を指定します。

[図 15.11](#) のパネルでは新しいイベントセットのための基底クラスと実装リストを指定します。

図 15.11 新規 Java イベントセットウィザード：基底クラスと実装



以下のオプションがあります。

[基底クラス](#)

[実装リスト](#)

基底クラス

[基底クラス] 欄で新しいイベントセットの基底クラス型を指定します。

可視コンポーネント：基底クラスが可視コンポーネントの場合オンにします。この下のポップアップメニューはRAD プラグインが認識できるクラスをリストします。

カスタム：基底クラスが不可視コンポーネントの場合オンにします。この下のフィールドに基底クラスの名前を入力してください。デフォルトの名前があらかじめ入力されています。

実装リスト

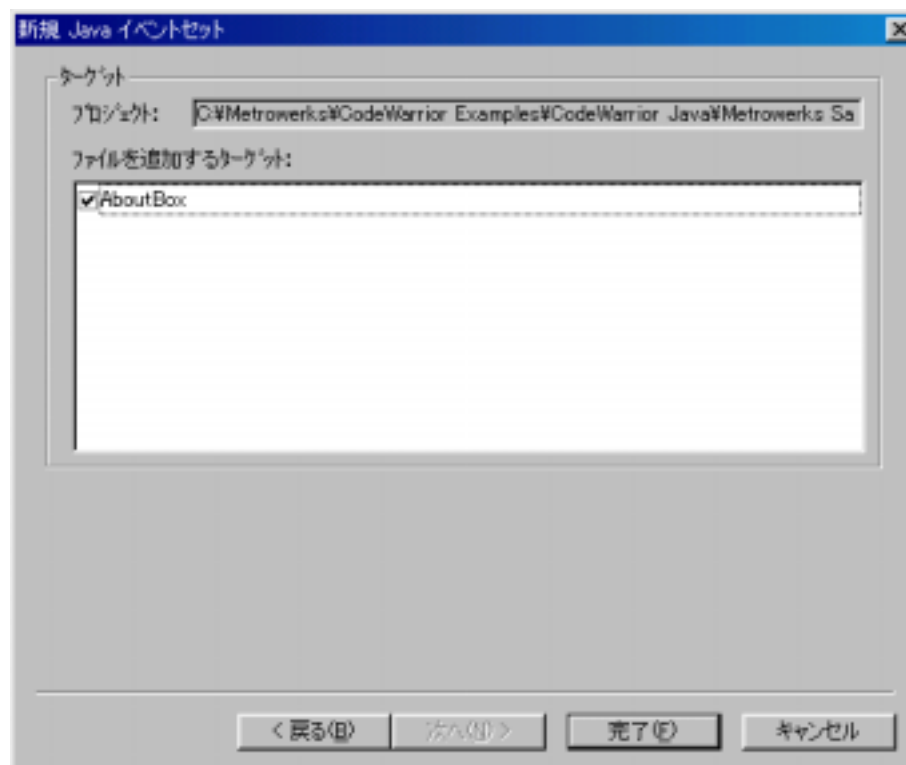
[実装リスト] フィールドに基底クラスが実装するインターフェースリストを入力してください。サンプルインターフェースが表示されます。インターフェースはコンマで区切ります。

3. 新しいイベントセットをビルドターゲットに割り当てる


[図 15.12](#) のパネルでは、ビルドターゲットにイベントセットを割り当てます。

イベントセットをビルドターゲットへ割り当てるには、[ファイルを追加するターゲット] 欄のリストの横のチェックボックスをクリックします。[図 15.12](#) では、新イベントセットを JavaApp Debug ビルドターゲットへ割り当てています。JavaApp Release へは割り当てていません。

図 15.12 新規 Java イベントセットウィザード：ターゲット



新規イベントダイアログ

「新規イベント」ダイアログ ([図 15.13](#)) を利用して、ブラウザウィンドウで選択したコンポーネントに対して新しいイベントを作成します。このダイアログを表示するには、ブラウザウィンドウを最前面にしてからブラウザメニューの [[新規イベント](#)] を選択してください。新アイテムアイコン () をクリックしても表示されます ([図 15.4 \(p427\)](#))。[新規イベント] コマンドは使用しているフレームワークによって名前が変わります。例えば

Java RAD プロジェクトを作成しているとき、コマンド名は [新規 Bean イベント] になります。

図 15.13 「新規イベント」ダイアログ

新規 Java イベント

イベントを追加するクラス:

名前:

引数: usually:<EventObjectType> evt

Throw (オプション):

引数のために必要なパッケージ(オプション):

修飾子
☐ Synchronized

宣言:

追加 キャンセル

「新規イベント」ダイアログには以下のオプションがあります。

[名前](#)

[引数](#)

[throw](#)

[引数のために必要なパッケージ](#)

[修飾子](#)

[宣言](#)

追加とキャンセル

名前

[名前] フィールドに新しいメソッドの名前を入力します。フレームワークプログラミング言語の命名規約に従った名前を付けてください。名前にスペースを使用できない言語もあります。

引数

必要であれば [引数] フィールドにイベントの引数を入力します。このフィールドの上に、サンプルの引数コンストラクタ関数が表示されます。

throw

必要であれば [throw] フィールドに例外処理情報 (*throws*) を入力します。

引数のために必要なパッケージ

必要であれば [引数のために必要なパッケージ] フィールドにイベントのパッケージを入力します。

修飾子

イベントを修正するには [Synchronized] チェックボックスをオンにします。

宣言

[宣言] 欄はダイアログで現在指定された形式の宣言を表示します。値やチェックボックスを変更すると、[宣言] 欄の内容もそれに合わせて変化します。この欄に情報が表示されるのは、[名前] と [型] フィールドの入力を終えた後です。

追加とキャンセル

ダイアログの現在の設定で新しいメソッドを追加するには [追加] ボタンをクリックします。変更を破棄するには [キャンセル] ボタンをクリックします。

第 16 章 バージョン管理システム

この章では、CodeWarrior IDE でソースコードのバージョン管理機能を使う方法を説明します。

[バージョン管理システムを使う](#)

[VCS を有効にする](#)

[VCS 機能を実行](#)

[一般的な VCS の操作](#)

バージョン管理システムを使う

バージョン管理システム（VCS：version control system）は、ソースコードのデータベースを維持し、そこでファイルをチェックアウト、チェックインします。これによりコードの変更を容易に管理することができます。

入手可能な VCS プラグイン

CodeWarrior のプラグインアーキテクチャはさまざまなバージョン管理システムをサポートします。以下にメトロワークスやサードパーティが提供する主なバージョン管理システムのプラグインをまとめます。

表 16.1 バージョン管理システムプラグイン

製品名	開発元	Windows
ClearCase	Rational	1
	Metrowerks	3
CVS	Electric Fish	
	Metrowerks	4
Perforce	Perforce	1
Projector	Apple Computer	
	Electric Fish	
	Metrowerks	
PVCS	Intersolv	1
	Synergex	

製品名	開発元	Windows
SourceSafe	Metrowerks	3
	Microsoft	4
	Mainsoft	
VOODOO	UNI Software	

解説：	4	IDE のサポート
	1	プラットフォームサポートのみ
	2	開発中
	3	各 VCS のツールが事前にインストールされている必要がある
空欄		サポートしない

プラグインツールに関する最新情報は、メトロワークスの Web サイトを参照してください。

<http://www.metrowerks.com/>

「Goto Product Specs for:」ポップアップの右横のフィールドで「Plugin tool」や「Version Control」を入力して検索してください。

VCS を有効にする

CodeWarrior とバージョン管理システムを使うために、バージョン管理システムをインストールして正しくセットアップします。

VCS ソフトウェアをインストール

バージョン管理システムソフトウェアをインストールする手順は、VCS プラグインを適切なフォルダにコピーして CodeWarrior IDE を再起動するだけです。ほとんどの VCS ソフトウェアはこれで使用できます。

注意： インストール手順については VCS ソフトウェアに付属のマニュアルを参照してください。問題に遭遇したときは VCS ソフトウェアの開発元へご連絡ください。

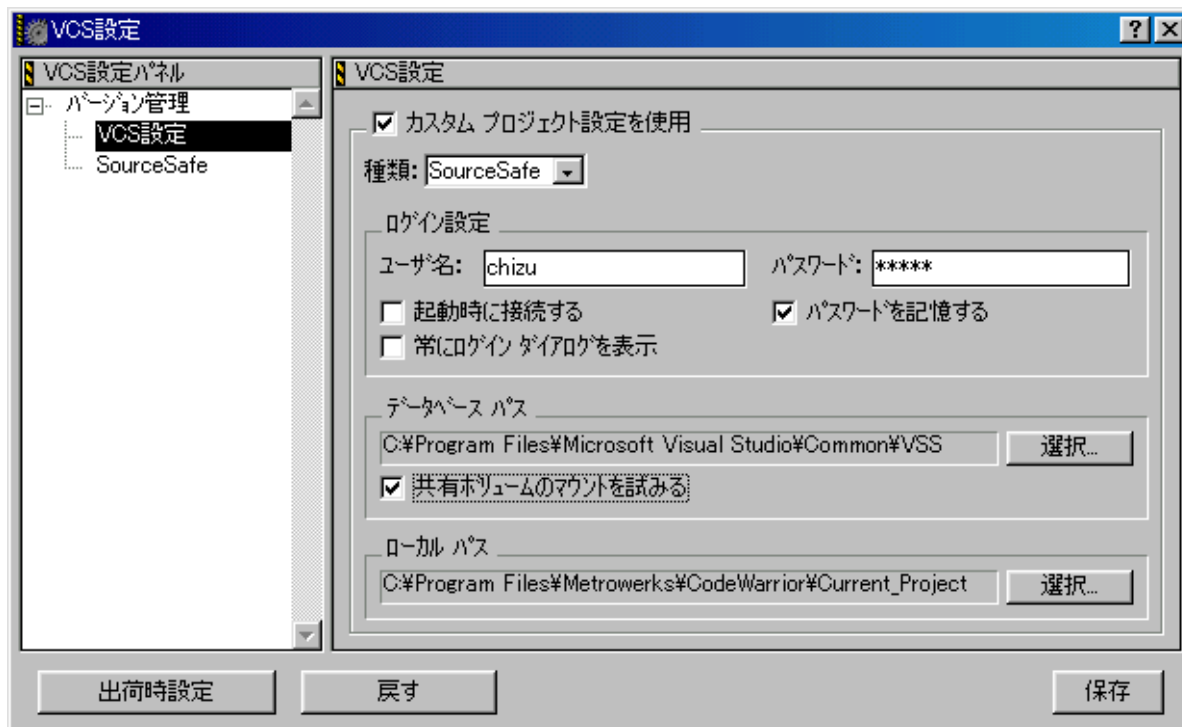
VCS プラグインは通常 Version Control フォルダへ配置します。以下のパスに VCS プラグインをコピーするフォルダを作ります。

Windows	{Compiler}\Plugins\Version Control
---------	------------------------------------

VCS ソフトウェアを利用

VCS 設定パネル (図 16.1) は VCS ソフトウェアをインストールすると表示されます。VCS 設定パネルでバージョン管理システムへ接続するためのオプションを設定します。

図 16.1 VCS 設定パネル



CodeWarrior プロジェクトでバージョン管理システムを使うための設定は簡単です。プロジェクトを開いて VCS 設定パネルに必要な情報を入力します。プロジェクトごとに別の情報を入力します。

注意： VCS ソフトウェアに付属のマニュアルを参照してください。問題に遭遇したときは VCS ソフトウェアの開発元へご連絡ください。

VCS の設定方法

1. 編集メニューの [バージョン管理設定]

VCS 設定パネル (図 16.1) が現れます。

2. [カスタムプロジェクト設定を使用]

[カスタムプロジェクト設定を使用] オプションをクリックして、バージョン管理を有効にします。

3. [種類] メニューでバージョン管理システムを選択する

CodeWarrior は数種類のバージョン管理システムをサポートします。VCS ソフトウェアが正しくインストールされていれば、その名前が [種類] ポップアップメニューに現れます (図 16.2)。

図 16.2 [種類] ポップアップメニュー



4. ユーザ名を入力する

[ユーザー名] フィールドにユーザー名を入力してください。

5. 必要ならばパスワードを入力する

[パスワードを記憶する] オプションをオンにして、[パスワード] フィールドにパスワードを入力してください。

6. その他のオプションを設定する

以下のオプションで VCS ソフトウェアの設定をカスタマイズします。

[起動時に接続する]

[常にログインダイアログを表示]

[パスワードを記憶する]

7. [データベースパス] を指定する

[選択] ボタンをクリックしてアクセスする VCS データベースを選択します。実際の [データベースパス] フィールドの設定は使用している VCS ソフトウェアによって異なります。詳細は VCS に付属のマニュアルを参照してください。

8. [ローカルパス] を指定する

[選択] ボタンをクリックしてファイルを保存するフォルダを選択します。実際の [ローカルパス] フィールドの設定は使用している VCS ソフトウェアによって異なります。詳細は VCS に付属のマニュアルを参照してください。

9. [保存] ボタンをクリックする

[保存] ボタンで変更を保存します。

選択が正しければ、[データベースパス] フィールドで指定した VCS データベースへ接続できます。接続できなければ、設定に間違いがないか上記のステップを見直してください。

注意： ソフトウェアによって設定すべき項目が異なります。問題があるときは VCS ソフトウェアの開発元にご連絡ください。

VCS 機能を実行

CodeWarrior IDE から VCS ソフトウェアの操作を実行したり、ログ情報を見ることができます。

[VCS メニュー](#)

[VCS ログインダイアログ](#)

[VCS ポップアップメニュー](#)

[プロジェクトウィンドウ](#)

[VCS メッセージウィンドウ](#)

VCS メニュー

プロジェクトに対して VCS ソフトウェアを有効にすると、VCS メニューが現れます ([図 16.3](#))。VCS メニューから [取り出す] [チェックアウト] [チェックイン] などの操作を実行します。

注意： VCS メニューの外観は使用している VCS により異なります。VCS ソフトウェアに付属のマニュアルを参照してください。

図 16.3 VCS メニュー

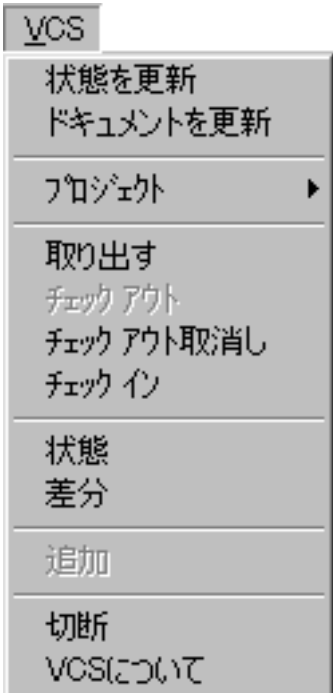


表 16.2 VCS のコマンド

コマンド	動作
状態を更新	プロジェクトファイルの状況を調べて情報を更新することにより、プロジェクトウィンドウの VCS 状況列を更新します。
プロジェクト	プロジェクトファイルに対して [取得] [チェックアウト] [チェックアウト取消し] [チェックイン] [状態] [追加] を実行するコマンドを含むサブメニューを表示します。
取り出す	プロジェクトデータベースにチェックインせずに、ファイルのコピーを取得します。
チェックアウト	ファイルを変更するためにチェックアウトします。
チェックアウト取消し	チェックアウトをキャンセルし、変更を破棄します。
チェックイン	変更したファイルをデータベースに戻し、チェックアウトを解除します。
状態	ファイルのチェックアウト状況を表示します。
追加	データベースにファイルを追加します。

コマンド	動作
接続 または切断	現在の接続状況により、プロジェクトデータベースへ接続、または接続を解除します。
VCS について	VCS プラグインの著作権情報やバージョン情報を表示します。

注意： VCS プラグインの種類によりメニューコマンドの名前が異なります。実行できる操作については VCS ソフトウェアに付属のマニュアルを参照してください。

図 16.4 「VCS ログイン」ダイアログ



VCS ログインダイアログ

「VCS ログイン」ダイアログ (図 16.4) は、バージョン管理システムを利用しているプロジェクトを開くとき、また [常にログインダイアログを表示] オプションがオンのときに表示されます。バージョン管理システムを利用するには、ユーザー名とパスワードを入力して [OK] ボタンをクリックしてください。利用しない場合は [キャンセル] ボタンをクリックしてください。

VCS ポップアップメニュー


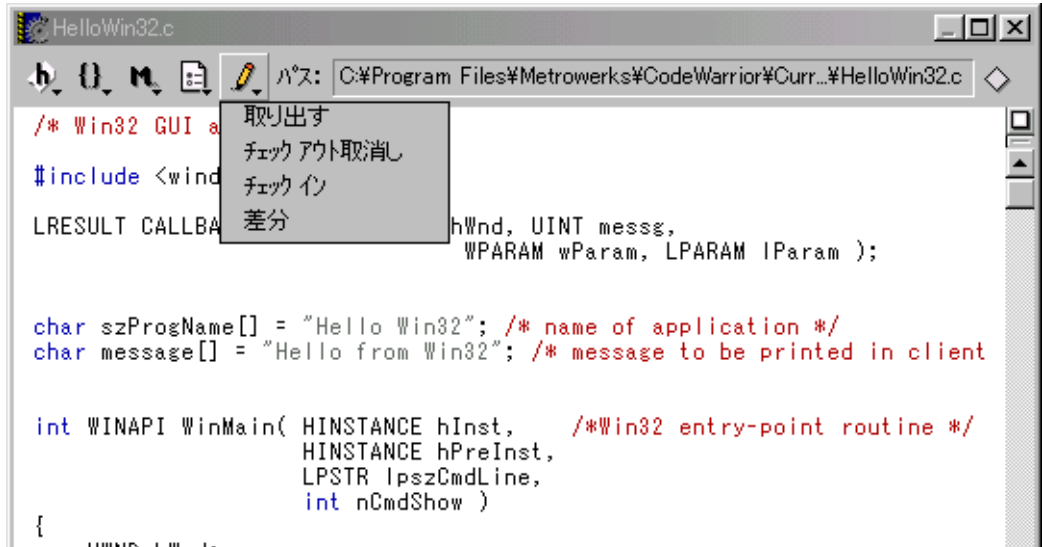
VCS ポップアップメニュー () はバージョン管理システムが有効なときにエディタウィンドウに現れます (図 16.5)。VCS ポップアップメニューのアイコンはファイルに対するアクセス権を示します。アクセス権の設定については「[ファイルのアクセス権](#)」(p450)を参照してください。






図 16.5 VCS ポップアップメニュー



ファイルのアクセス権

バージョン管理システムを利用して、プロジェクトにある各ファイルに異なるアクセス権を設定することができます。ファイルのアクセス権状態はプロジェクトウィンドウのチェックアウト状況列 (図 16.6) で見るすることができます。エディタウィンドウの VCS ポップアップメニュー (図 16.6) のアイコンの意味は表 16.3 を参照してください。

表 16.3 ファイルアクセス権アイコン

アイコン		意味
	チェックアウト	ファイルを編集でき、変更をバージョン管理データベースに追加できます。
	チェックイン	ファイルを編集できません。このファイルはバージョン管理データベースの管理下にあります。
	ライタブル	ファイルを編集できますが、変更をバージョン管理データベースに追加できません。変更するためにファイルをチェックアウトしていないためです。
	ロック解除	ファイルは編集できますが、バージョン管理データベースにチェックインしていません。
	ロック	ファイルは編集できません。バージョン管理データベースの管理下にもありません。ファイルにアクセスするためのアクセス権がないか、他のユーザーがファイルをロックしています。

VCS ポップアップメニューをクリックしたときの状況により、編集中のファイルに対して実行できるコマンドが変わります。コマンドの詳細は表 16.4 を参照してください。

VCS ポップアップメニューから実行したいコマンドを選択します。メニューコマンドの末尾に [...] があれば、高度なオプションを設定するためのダイアログが現れます。ここでコマンドをカスタマイズすることができます。

実行可能な操作についてはバージョン管理システムに付属のマニュアルを参照してください。

表 16.4 VCS ポップアップメニューのコマンド

コマンド	動作
ロック解除	可能であればファイルのロックを解除し、書き込み可能な状態にします。
追加	バージョン管理データベースにファイルを追加します。
取得	バージョン管理データベースからファイルのコピーを取得します。
チェックアウト	変更を加えるために、バージョン管理データベースからファイルをチェックアウトします。
チェックアウト取り消し	ファイルに加えた変更を破棄し、バージョン管理データベースからのチェックアウトをキャンセルします。
チェックイン	変更を加えたファイルをバージョン管理データベースにチェックインします。
ロック解除	ファイルを書き込み可能な状態にしますが、これをバージョン管理データベースへチェックインすることはできません。

プロジェクトウィンドウ


プロジェクトでバージョン管理システムを利用しているとき、プロジェクトウィンドウにチェックアウト状況列 ( 16.6) が表示されます。この列はプロジェクトファイルの現在のチェックアウト状況、ファイルアクセス権を示します。ファイルのチェックアウト状況を変更すると、この列もそれを反映して変化します。

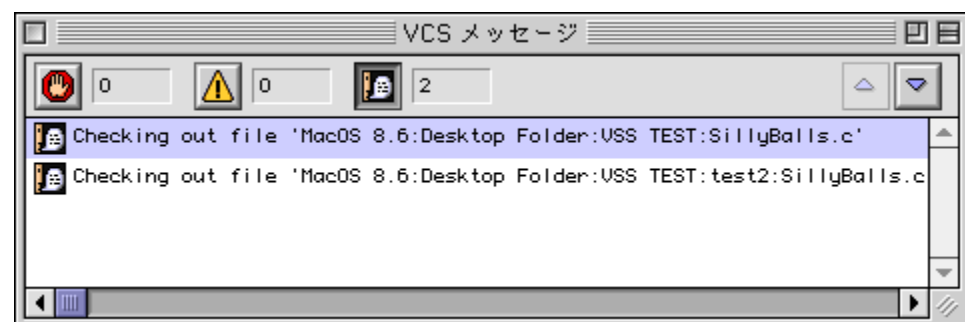
図 16.6 チェックアウト状況列



VCS メッセージウィンドウ

CodeWarrior IDEはメッセージウィンドウにバージョン管理システムのメッセージを表示します (図 16.7)。ウィンドウ内のコントロールの使い方は「[メッセージウィンドウを使う](#)」(p271) を参照してください。

図 16.7 VCS メッセージウィンドウ



一般的な VCS の操作

[追加] [取得] [チェックアウト] [チェックイン] 以外にも以下の操作が可能です。

[ファイル状況を取得](#)

[チェックインしたファイルを修正](#)

[その他の操作](#)

ファイル状況を取得

プロジェクトファイルでバージョン管理システムを使うように設定してファイルをチェックインすると、バージョン管理システムに管理されている各ファイルに対して、プロジェクトウィンドウのチェックアウト状況列に[表 16.3](#)のアイコンが表示されます。プロジェクトウィンドウ内のチェックアウト状況列の位置は、[図 16.6](#)を参照してください。

列の一番上のチェックアウト状況列アイコンをクリックすると、バージョン管理データベースに対して[状態を更新]コマンドが実行されます。ハードディスク内のすべてのファイルの状況が、バージョン管理データベース内のファイルの状況と比較され、同期を取られます。

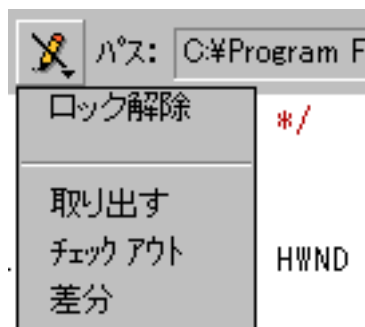
エディタウィンドウの VCS ポップアップメニューは常にファイルのチェックアウト状況を示します。

チェックインしたファイルを修正

リードオンリーのソースコード管理ファイルの状況を変更して、修正することもできますが、修正の後でソースコード管理データベースにチェックインして戻すことはできません。この機能は、ファイルを一時的に書き込めるようにしてテストなどを行うために提供されています。バージョン管理データベースが重要なファイルへの上書きのアクセス権を許せば、必要とされる管理サービスを維持することができません。

エディタウィンドウの VCS ポップアップメニューから[ロック解除]を選択してください([図 16.8](#))。これでファイルを修正できますが、バージョン管理データベースにチェックインして戻すことはできません。

図 16.8 アクセス権をリードオンリーからモディファイリードオンリーに変更



その他の操作

使用しているバージョン管理システムによっては、これ以外の操作を実行できます。詳細はバージョン管理システムに付属のマニュアルを参照してください。

第 17 章 CodeWarrior リファレンス

CodeWarrior IDE のメニューコマンドとそのキーバインディングについて説明します。この章を CodeWarrior のリファレンスとしてもご利用ください。

以下の内容について説明します。

[IDE メニューリファレンス](#)

[デフォルトのキーバインディング](#)

IDE メニューリファレンス

ここでは CodeWarrior IDE メニューの各コマンドを説明します。

CodeWarrior IDE メニューバーには以下のメニューがあります。

[ファイルメニュー](#)

[編集メニュー](#)

[検索メニュー](#)

[プロジェクトメニュー](#)

[デバッグメニュー](#)

[データメニュー](#)

[ブラウザメニュー](#)

[ウィンドウメニュー](#)

[レイアウトメニュー](#)

[バージョン管理システム \(VCS\) メニュー](#)

[ヘルプメニュー](#)

[Java 例外サブメニュー](#)

[ツールバーサブメニュー](#)

[ファイルメニュー](#)、[編集メニュー](#)、[検索メニュー](#)、[プロジェクトメニュー](#)、[デバッグメニュー](#)、[ウィンドウメニュー](#)、[ヘルプメニュー](#)は常に表示されます。

[バージョン管理システム \(VCS\) メニュー](#) は CodeWarrior と互換性のあるバージョン管理システム (別売) をインストールしている場合のみ表示されます。バージョン管理システムの詳細はバージョン管理システムに付属しているマニュアルを参照してください。

[データメニュー](#)、[ブラウザメニュー](#)、[レイアウトメニュー](#)はこのコマンドを利用できるウィンドウがアクティブなときに表示されます。

ファイルメニュー

ファイルメニューには既存または新規のソースファイルおよびプロジェクトを開く、作成する、保存する、閉じる、印刷するためのコマンドがあります。ファイルメニューには編集したファイルを保存する方法も複数用意されています。

新規テキストファイル

[新規テキストファイル] を選択して、編集可能なテキストファイルを新しく作成します。

このコマンドの詳細は [「新規ファイルを作成」\(p77\)](#) を参照してください。

新規

[新規] を選択して、「新規」ダイアログを開きます。このダイアログはさまざまな CodeWarrior IDE アイテムの作成を支援します。

開く

[開く] を選択して、既存のファイルを開きます。

このコマンドの詳細は [「ファイルメニューからファイルを開く」\(p78\)](#) を参照してください。

過去のファイルを開く

[過去のファイルを開く] を選択して、最近開いたプロジェクトおよびファイルのサブメニューを表示します。このサブメニューからファイルを選択すれば、即座にプロジェクトやファイルを開くことができます。

同名のファイルが複数ある場合、そのファイルへの完全なパスが表示されます。

このコマンドの詳細は [「ファイルメニューからファイルを開く」\(p78\)](#) を参照してください。

ファイルを検索して開く

[ファイルを検索して開く] を選択して、[アクセスパス](#) 設定パネルのアクセスパスを使って、既存のテキストファイルを開きます。

詳細は [「エディタウィンドウからファイルを開く」\(p80\)](#) を参照してください。

ファイル名を検索して開く

[ファイル名を検索して開く] を選択して、エディタウィンドウで現在選択しているテキストをターゲットファイル名として使って、既存のテキストファイルを開きます。

詳細は [「エディタウィンドウからファイルを開く」\(p80\)](#) を参照してください。

閉じる

[閉じる] を選択して、アクティブなウィンドウを閉じます。

詳細は「[ファイルを閉じる](#)」(p85) を参照してください。

すべてのエディタウィンドウを閉じる方法は「[すべてのファイルを閉じる](#)」(p85) を参照してください。

すべて閉じる

[すべて閉じる] を選択して、開いているウィンドウを種類ごとにすべて閉じます。このメニューコマンドの名前はウィンドウの種類によって変わります。例えば複数のエディタウィンドウを開いているとき、このメニューコマンドは[エディタ書類をすべて閉じる] になります。

詳細は「[すべてのファイルを閉じる](#)」(p85) を参照してください。

保存

[保存] を選択して、アクティブなエディタウィンドウの内容をディスクに保存します。

詳細は「[1 つのファイルを保存](#)」(p82) を参照してください。

すべて保存

[すべて保存] を選択して、開いているエディタファイルをすべて保存します。

詳細は「[すべてのファイルを保存](#)」(p82) を参照してください。

名前を付けて保存

[名前を付けて保存] を選択して、アクティブなウィンドウの内容を、別の名前を指定してディスクに保存します。

詳細は「[ファイルを新規保存](#)」(p83) を参照してください。

コピーを保存

[コピーを保存] を選択して、アクティブなウィンドウを別ファイルとして保存します。このコマンドは、アクティブなウィンドウの種類によって動作が異なります。

詳細は「[ファイルのコピー保存](#)」(p83) を参照してください。

復帰

[復帰] を選択して、アクティブなエディタウィンドウを最後に保存した状態に戻すときに使います。

ファイルを前の状態に戻す方法の詳細は「[直前に保存したファイルに戻す](#)」(p87) を参照してください。

コンポーネントの読み込み

[コンポーネントの読み込み] を選択して、他のカタログからコンポーネントをインポートします。

詳細は「[コンポーネントカタログのツールバー](#)」(p401) を参照してください。

カタログを閉じる

[カタログを閉じる] を選択して、現在のカタログを閉じ、それを[コンポーネントカタログウィンドウ](#)と[コンポーネントパレット](#)から削除します。

詳細は「[コンポーネントカタログのツールバー](#)」(p401) を参照してください。

プロジェクトの読み込み

[プロジェクトの読み込み] を選択して、XML (eXtensible Markup Language) ファイルを IDE ヘインポートします。これで XML ファイルを CodeWarrior プロジェクトとして保存することができます。IDE は新規プロジェクトファイルの名前と保存場所の指定を要求します。

詳細は「[プロジェクトのインポートとエクスポート](#)」(p73) を参照してください。

プロジェクトの書き出し

[プロジェクトの書き出し] を選択して、CodeWarrior プロジェクトを XML (eXtensible Markup Language) ファイル形式へエクスポートします。IDE は新規 XML ファイルの名前と保存場所の指定を要求します。

詳細は「[プロジェクトのインポートとエクスポート](#)」(p73) を参照してください。

ページ設定

[ページ設定] を選択して、CodeWarrior IDE からファイルを印刷するときのオプションを設定します。

詳細は「[印刷オプションを設定](#)」(p86) を参照してください。

印刷

[印刷] を選択して、CodeWarrior IDE からプリンタにファイルを印刷します。

ファイルの印刷の詳細は「[ウィンドウを印刷](#)」(p86) またはプリンタに付属しているマニュアルを参照してください。

終了

[終了] を選択して、以下の条件のいずれかに合えば、すぐに CodeWarrior IDE を終了します。

開いているエディタファイルの変更がすべて保存されている

開いているエディタファイルが変更されていない

プロジェクトウィンドウが開いている場合、プロジェクトファイルへの変更をすべて保存してから環境を終了します。エディタウィンドウが開いていて変更内容がまだ保存されていない場合、CodeWarrior IDE は終了する前に変更内容を保存するかどうかを確認します。

編集メニュー

編集メニューは通常の編集コマンドの他、「IDE 設定」ダイアログやターゲット設定ダイアログを開くためのコマンドなど、CodeWarrior 特有のコマンドもあります。

取り消し

このコマンドは、直前の操作とエディタオプションの設定により変わります。

[取り消し] は直前の操作の効果を取り消します。[取り消し] コマンドの名前は、直前に実行した操作により変わります。例えば、開いているエディタウィンドウでタイプ入力した直後は、[取り消し] から [入力取り消し] に変わります。[入力取り消し] コマンドを選択すると、入力したテキストが削除されます。

詳細は「[直前の編集を取り消す](#)」(p108)、[複数の操作を取り消す、またはやり直す](#)」(p108)を参照してください。

エディタ設定パネルで「[複数アンドゥを利用](#)」をオンにしていない場合、[取り消し] と [再実行] が切り替わります。このオプションの設定方法の詳細は、「[エディタ設定](#)」(p194)を参照してください。

再実行

取り消した操作は [再実行] で再実行できます。例えば [入力取り消し] コマンドを選択した後、コマンドは [入力再実行] に変わります。このコマンドを選択すると、再び [取り消し] へ戻ります。

エディタ設定パネルで「[複数アンドゥを利用](#)」をオンにすると、[取り消し] と [再実行] の操作にさらに柔軟性が加わります。[取り消し] を複数回選択して、複数の操作を元の状態まで戻すことができます。[再実行] を複数回選択すれば、複数の操作を再実行することができます。

取り消し操作の詳細は、「[直前の編集を取り消す](#)」(p108) と「[複数の操作を取り消す、またはやり直す](#)」(p108)を参照してください。

「複数アンドゥ」を設定する方法は「[エディタ設定](#)」(p194)を参照してください。

カット

[切り取り] を選択して、選択したテキストを削除してシステムのクリップボードに移します。クリップボードの元の内容は置換されます。

コピー

[コピー] を選択して、アクティブなエディタウィンドウで選択したテキストをシステムのクリップボードにコピーします。メッセージウィンドウがアクティブであれば、メッセージウィンドウ内のすべてのテキストをクリップボードにコピーします。

貼り付け

[貼り付け] を選択して、クリップボードの内容をアクティブなエディタウィンドウにペーストします。

[ペースト] コマンドは、選択したテキストをクリップボードの内容で置換します。テキストを選択していない場合は、クリップボードの内容がテキスト挿入位置の直後に置かれます。

アクティブなウィンドウがメッセージウィンドウの場合、[ペースト] コマンドは薄く表示され、使用できません。

削除

[削除] を選択して、選択したテキストを消去します。クリップボードにはコピーされません。[削除] コマンドは、Delete キーまたは Backspace キーを押した場合と同じ動作をします。

すべてを選択

[すべてを選択] を選択して、アクティブなウィンドウ内のすべてのテキストを選択します。このコマンドは通常、[カット]、[コピー]、および [削除] などの編集メニューの他のコマンドと一緒に使います。

テキスト選択の詳細は [「テキストを選択」\(p106\)](#) を参照してください。

バランス

[バランス] を選択して、括弧 ()、大括弧 []、中括弧 { } のいずれかの対で囲まれたテキストを選択します。このコマンドの使い方とタイプ入力時の対の確認の方法は、[「括弧のバランスをとる」\(p107\)](#) を参照してください。

左に移動

[左に移動] を選択して、選択したソースコードをタブ 1 つ分だけ左に移動します。タブサイズは、「IDE 設定」ダイアログで指定します。

この機能の詳細は [「テキストを左右に移動」\(p108\)](#) を参照してください。

右に移動

[右に移動] を選択して、選択したソースコードをタブ 1 つ分だけ右に移動します。

この機能の詳細は [「テキストを左右に移動」\(p108\)](#) を参照してください。

IDE 設定

[環境設定] を選択して、CodeWarrior IDE の「IDE 設定」ダイアログを表示します。

環境の設定方法の詳細は [「IDE 設定パネルを選択」\(p183\)](#) を参照してください。

Target の設定

[Target の設定] を選択して (Target には現在のターゲットの名前が入ります) ターゲット設定ダイアログを表示します。このコマンドの名前は、カレントターゲットの名前によって変わるので注意してください。

[ターゲット設定] の設定の詳細は、[「設定パネルを選択」\(p228\)](#) を参照してください。カレントビルドターゲットの変更方法は、[「デフォルトターゲットを設定」\(p468\)](#) を参照してください。

バージョン管理設定

[バージョン管理設定] を選択して、バージョン管理システムの設定パネルを表示します。このパネルの詳細は [「バージョン管理システム」\(p443\)](#) を参照してください。

CodeWarrior IDE にバージョン管理システム用の設定がされていない場合、このコマンドが選択できません。

コマンドとキーバインディング

[コマンドとキーバインディング] を選択して、「IDE コマンドのカスタマイズ」ダイアログを表示します。

検索メニュー

検索メニューにテキストの検索と置換に使うすべてのコマンドがあります。コードナビゲーションのためのコマンドも含まれています。

検索

[検索] を選択して、1 つまたは複数のファイル内で、指定した文字列の出現箇所を検索または置換するための「検索」ダイアログを開きます。

「検索」ダイアログとその機能の詳細は、[「検索ダイアログの解説」\(p115\)](#) を参照してください。

次を検索

[次を検索] を選択して、アクティブなウィンドウで[検索文字列](#)の次の出現箇所を見つけます。これは「検索」ダイアログの [検索] ボタンをクリックすることと同じです。

この機能の詳細は [「テキストを検索」\(p125\)](#) を参照してください。

前を検索

[前を検索] を選択して、アクティブなウィンドウで[検索文字列](#)の前の出現箇所を検索します。

この機能の詳細は「[テキストを検索](#)」(p125) を参照してください。

次のファイル内を検索

[次のファイル内を検索] を選択して、[検索文字列](#)を、複数ファイル検索欄（検索ウィンドウの[複数ファイル検索](#)で表示されます）にリストされた次のファイルで検索します。これは、検索ウィンドウに代わる方法の 1 つです。[複数ファイル検索](#)をクリックしていないとき（[図 6.3 \(p119\)](#)）このコマンドは薄く表示されます。

この機能の詳細は「[複数のファイルでのテキスト検索と置換](#)」(p130) を参照してください。

前のファイル内を検索

[前のファイル内を検索] を選択すると、[次のファイル内を検索] とほとんど同じ働きをします。ファイルリスト内の 1 つ前のファイルの終わりから[検索文字列](#)の次の出現箇所を探します。

この機能の詳細は「[複数のファイルでのテキスト検索と置換](#)」(p130) を参照してください。

選択部分を検索文字列へ

[選択部分を検索文字列へ] を選択して、アクティブなウィンドウで選択したテキストを [検索] ダイアログの [検索] フィールドにコピーして、[検索文字列](#)とします。これはテキストをコピーして [検索] ダイアログにペーストすることと同じです。

テキストの選択方法は「[テキストを選択](#)」(p106) を参照してください。

選択部分を置換文字列へ

[選択文字列を置換文字列へ] を選択して、アクティブなウィンドウで選択したテキストを [検索] ダイアログの『[Replace](#)』フィールドにコピーして、[置換文字列](#)とします。これはテキストを選択して [検索] ダイアログにコピーすることと同じです。

テキストの置換については「[検索したテキストを置換](#)」(p127) を参照してください。

選択部分を検索

[選択部分を検索] を選択して、アクティブなテキストエディタウィンドウで選択したテキストの次の出現箇所を検索します。

この機能の詳細は「[テキストを検索](#)」(p125) を参照してください。

選択部分を逆方向検索

[選択部分を逆方向検索] を選択して、アクティブなテキストエディタウィンドウで選択したテキストの前の出現箇所を検索します。

この機能の詳細は「[テキストを検索](#)」(p125) を参照してください。

テキストの選択方法は「[テキストを選択](#)」(p106) を参照してください。

置換

[置換] を選択して、アクティブなウィンドウで選択したテキストを、[検索] ダイアログの[置換文字列](#)で置換します。アクティブなエディタウィンドウでテキストが選択されていない場合は、このコマンドは薄く表示されます。

このコマンドは、[検索] ダイアログを開かずに文字列を置換するときに便利です。例えば、変数 `icount` をすべて `jcount` に置換したと仮定します。ソースコードをスクロールしていて変数 `icount` の綴りを間違えて `icont` になって残っているのを見つけたとします。この変数 `icont` を `jcount` で置換するには、`icont` を選択して[検索メニュー](#)の [置換] コマンドを選択します。

テキストの置換方法は「[検索したテキストを置換](#)」(p127) を参照してください。

テキストの選択方法は「[テキストを選択](#)」(p106) を参照してください。

置換後次を検索

[置換後次を検索] を選択して、選択したテキストを「検索」ダイアログの[置換文字列](#)で置換し、[[次を検索](#)] を実行します。アクティブなエディタウィンドウでテキストが選択されておらず、「検索」ダイアログの [検索] フィールドに[検索文字列](#)がない場合は、このコマンドは薄く表示されます。

テキストの置換方法は「[検索したテキストを置換](#)」(p127) を参照してください。

テキストの選択方法は「[テキストを選択](#)」(p106) を参照してください。

置換後逆方向検索

[置換後逆方向検索] を選択して、テキストを置換した後に [[前を検索](#)] を実行すること以外は、[[置換後次を検索](#)] と同じ働きをします。

すべて置換

[すべて置換] を選択して、検索文字列のすべての出現箇所を検索し、置換文字列で置換します。アクティブなエディタウィンドウでテキストが選択されておらず、[検索] ダイアログの [検索] フィールドにテキストがない場合は、このコマンドは薄く表示されます。

定義を検索

[定義を検索] を選択して、アクティブなウィンドウで選択されたルーチン名の定義を検索します。開いているプロジェクトに含まれるソースファイルを検索します。定義が見つかったら、CodeWarrior IDE はそのルーチンが定義されているソースファイルを開きます。ルーチンの名前が強調表示されます。

複数の定義が見つかった場合、複数定義を警告するメッセージウィンドウが表示されます。メッセージウィンドウの詳細は、[「メッセージウィンドウを使う」\(p271\)](#) を参照してください。

定義が見つからない場合は、システムビープ音が鳴ります。

戻る

[戻る] を選択すると、ブラウザの前のビューに戻ります。

この機能の詳細は [「戻ると次へ」\(p175\)](#) を参照してください。

次へ

[次へ] を選択すると、ブラウザの次のビューに移ります ([[戻る](#)] コマンドの後で使うと元のビューに戻ります)。

詳細は [「戻ると次へ」\(p175\)](#) を参照してください。

指定の行へ移動

[指定の行へ移動] を選択して、行番号を入力するダイアログを開き、そこで指定した行にテキスト挿入位置を移動します。

この機能の詳細は [「特定の行にジャンプ」\(p112\)](#) を参照してください。

ファイル比較

[ファイル比較] を選択して、比較する 2 つのファイルを選択するダイアログを開きます。ファイルを選択した後ファイル比較ウィンドウが現れます。詳細は [「ファイル/フォルダの比較とマージ」\(p87\)](#) を参照してください。

差分を適用する

[差分を適用する] を選択して、ファイル比較ウィンドウで、デスティネーションファイルに差異を追加、削除、変更します。

差分の適用を取り消す

[差分の適用を取り消す] を選択して、ファイル比較ウィンドウで、[差分を適用する] と反対の動作をします。

プロジェクトメニュー

プロジェクトメニューにはプロジェクトにファイルやライブラリを追加したり、削除するコマンドがあります。また、プロジェクトのコンパイル、ビルド、およびリンクを実行できます。

ウィンドウを追加

[ウィンドウを追加] を選択して、アクティブなエディタウィンドウ内のファイルを、開いているプロジェクトに追加します。

この機能の詳細は [「ウィンドウを追加コマンドを使う」\(p60\)](#) を参照してください。

ファイルを追加

[ファイルを追加] を選択して、プロジェクトウィンドウにファイルを追加します。

この機能の詳細は [「ファイルを追加コマンドを使う」\(p57\)](#) を参照してください。

新規グループを作成

[新規グループを作成] を選択して、カレントプロジェクト内に新しいグループを作成します。このコマンドは、カレントプロジェクトのウィンドウで [ファイル] タブを選択しているときにプロジェクトメニューに表示されます。

グループ作成の詳細は [「グループの作成」\(p61\)](#) を参照してください。

新規ターゲットを作成

[新規ターゲットを作成] を選択して、カレントプロジェクト内に新しいビルドターゲットを作成します。このコマンドは、カレントプロジェクトのウィンドウで [ターゲット] タブを選択しているときにプロジェクトメニューに表示されます。

ターゲット作成の詳細は [「複合プロジェクトを作成」\(p64\)](#) を参照してください。

シンタックスをチェック

[シンタックスをチェック] を選択して、アクティブなエディタウィンドウのソースファイル、または開いているプロジェクトウィンドウで選択したファイルの文法を検査します。アクティブなエディタウィンドウが空の場合、またはプロジェクトを開いていない場合は、薄く表示され、選択できません。

[シンタックスをチェック] はオブジェクトコードを生成しません。このコマンドは、ソースコードのシンタックスエラーを検査するだけです。この操作の進行状況は、ツールバーのメッセージ欄に示されます。

このコマンドは、Esc キーを押すことで、いつでも打ち切ることができます。

複数のエラーが検出された場合、メッセージウィンドウが現れます。コンパイルエラーの修正方法は、[「コンパイラのエラーと警告を修正」\(p273\)](#) を参照してください。

プリプロセス

[プリプロセス] を選択して、C、C++、Pascal などプリプロセッサを持つ言語で書かれたソースファイルをプリプロセスします。

このコマンドの詳細は [「ソースコードをプリプロセス」\(p268\)](#) を参照してください。

プリコンパイル

[プリコンパイル] を選択して、アクティブなエディタウィンドウのテキストファイルをプリコンパイルし、プリコンパイルヘッダファイルを作成します。

詳細は「[プリコンパイルまたはプリプロセスヘッダを使う](#)」(p262) を参照してください。

コンパイル

[コンパイル] を選択して、選択したファイルをコンパイルします。プロジェクトウィンドウがアクティブな場合は、選択したファイル、セグメントまたはグループがコンパイルされます。エディタウィンドウ内のソースファイルがアクティブな場合、そのソースファイルがコンパイルされます。ソースファイルは開いているプロジェクトに含まれていなければなりません。

詳細は「[プロジェクトのコンパイル、リンク](#)」(p256) を参照してください。

逆アセンブル

[逆アセンブル] を選択して、プロジェクトウィンドウで選択したソースファイルを逆アセンブルし、オブジェクトコードを新しいウィンドウに表示します。新しいウィンドウのタイトルはソースファイル名に拡張子 .dump を付けたものです。

この機能の詳細は「[ソースコードを逆アセンブル](#)」(p268) を参照してください。

最新状態に更新する

[最新状態に更新する] を選択して、修正、およびタッチされたファイルをすべてコンパイルすることにより、開いているプロジェクトを更新します。

詳細は「[プロジェクトを更新](#)」(p258) を参照してください。

メイク

[メイク] を選択して、修正、およびタッチされたファイルをコンパイル、リンクして、選択したプロジェクトをビルドします。ビルドが成功したときの生成物は、選択したプロジェクトの種類により異なります。

詳細は「[プロジェクトをメイク](#)」(p259) を参照してください。

ビルドを中止

[ビルドを中止] を選択して、ビルドを中止します。

詳細は「[プロジェクトのコンパイル、リンク](#)」(p256) を参照してください。

オブジェクトを削除

[オブジェクトを削除] を選択して、開いているプロジェクトに含まれるすべてのコンパイル済みバイナリを削除します。各ファイルの[コード列](#)および[データ列](#)の数はゼロにリセットされます。

詳細は「[オブジェクトを削除](#)」(p261) を参照してください。

オブジェクトコードの削除と圧縮

[オブジェクトコードの削除と圧縮] を選択して、プロジェクトからすべてのバイナリを削除してプロジェクトを圧縮します。プロジェクトファイルに格納されているすべてのバイナリとデバッグ情報が削除され、プロジェクトに含まれるファイルとプロジェクトの設定値に関する情報だけが保持されます。

詳細は「[オブジェクトを削除](#)」(p261) を参照してください。

ファイルパスを再検索

[ファイルパスを再検索] を選択して、ビルドや他のプロジェクト操作を高速化するため、IDE はアクセスパスを検索した後、プロジェクトファイルの位置をキャッシュします。[ファイルパスを再検索] コマンドはキャッシュしたファイル位置を消去するため、IDE はアクセスパスを再度検索します。このコマンドはファイルの位置を変更した場合に便利です。

ターゲット設定パネルの [相対パスを使ってプロジェクトエントリを保存] オプションがオンの場合、IDE はプロジェクトの相対パス情報をリセットしません。ファイルを再検索すると同じ位置にあるソースファイルを見つけます (ファイルが古い位置に存在しない場合は別です)。この場合 IDE はヘッダファイルを再検索します。ソースファイルも再検索するには、まず [プロジェクト内のパスをリセット] を選択します。

[相対パスを使ってプロジェクトエントリを保存] がオフの場合、IDE はヘッダファイルとソースファイルの両方を再検索します。

プロジェクト内のパスをリセット

[プロジェクト内のパスをリセット] を選択すると、[相対パスを使ってプロジェクトエントリを保存] オプションがオンの場合、プロジェクトの位置の情報をリセットします。次にプロジェクトにアクセスするときに IDE はアクセスパス内でプロジェクトを再検索します。[相対パスを使ってプロジェクトエントリを保存] がオフの場合、このコマンドは何もしません。

ファイル更新日時を同期させる

[ファイル更新日時を同期させる] を選択して、プロジェクトファイルに保管された修正日時を更新します。プロジェクト内の各ファイルの修正日時を検査して、ファイルが最後のコンパイルの後修正されていれば、再コンパイルの必要ありとマークします。

詳細は「[修正日時の同期](#)」(p63) を参照してください。

デバッグを有効にする

[デバッグを有効にする] を選択して、プロジェクトをデバッグできるように IDE 設定を設定します。[デバッグを有効にする] でデバッグを有効にし、[デバッグを無効にする] でデバッグを無効にします。

デバッグを無効にする

[デバッグを無効にする] を選択して、デバッグを無効にします。[デバッグを有効にする] でデバッグを有効にし、[デバッグを無効にする] でデバッグを無効にします。

詳細は「[プロジェクトでデバッグを制御](#)」(p73) を参照してください。

実行

[実行] を選択して、スタンドアロンアプリケーションをコンパイル、リンクして作成し、起動します。

プロジェクトの種類が、ライブラリまたは共有ライブラリに設定されている場合は、[実行] コマンドは薄く表示されます。

詳細は「[プロジェクトを実行](#)」(p260) を参照してください。

デバッグ

[デバッグ] を選択して、プロジェクトをコンパイル、リンクしてから、CodeWarrior デバッグを使ってプロジェクトのデバッグファイルを開きます。デバッグ対応のプロジェクトに対して CodeWarrior デバッグを起動します。

デフォルトプロジェクトを設定

[デフォルトプロジェクトを設定] を選択して、デフォルトのプロジェクトを選択します。デフォルトのプロジェクトの詳細は、「[デフォルトのプロジェクトを選択](#)」(p53) を参照してください。

デフォルトターゲットを設定

[デフォルトターゲットを設定] を選択して、カレントプロジェクト内で操作する別のターゲットを選択することができます。このメニューコマンドは、プロジェクト内の複数のターゲットを切り替えてそれぞれにビルドを行う場合に便利です。

デバッグメニュー

デバッグメニューにはプログラムの実行を制御するコマンドがあります。CodeWarrior、または外部デバッグでこれらのコマンドをすべて使用できます。Java 例外のブレイクをカスタマイズするサブメニューもあります。

中止

[中止] を選択して、ターゲットプログラムの実行を終了し、IDEにコントロールを戻します。

再起動

デバッグを再度始めます。

リセット (エンベデッドシステム)

プログラムをリセットし、IDE にコントロールを戻します。

ハードリセット (エンベデッドシステム)

プログラムとハードウェアをリセットし、IDE にコントロールを戻します。

ステップオーバー

[ステップオーバー] を選択して、1 つのステートメントを実行し、関数呼び出しを飛び越します。

ステップイン

[ステップイン] を選択して、1 つのステートメントを実行し、関数呼び出しの内部へ入ります。

ステップアウト

[ステップアウト] を選択して、呼び出し元から出るまで現在の関数を実行します。

停止

[停止] を選択して、ターゲットプログラムの実行を一時的に停止し、CodeWarrior デバッガにコントロールを戻します。

ブレークポイントの設定

[ブレークポイントの設定] を選択して、選択した行へブレークポイントを設定します。ブレークポイントの状態によって [ブレークポイントの設定] と [ブレークポイントの削除] は切り替わります。

エディタウィンドウでブレークポイントを表示 (デバッグメニューの [ブレークポイントを表示] コマンドを選択) している場合、左側にブレークポイント列が表示されます。ブレークポイントを設定したコードには、ブレークポイント列にマーカがあります。ブレークポイントを消去すると、このマーカはブレークポイント列から消えます。

ブレークポイントを削除

[ブレークポイントを削除] を選択して、選択した行のブレークポイントを削除します。ブレークポイントの状態によって [ブレークポイントの設定] と [ブレークポイントの削除] は切り替わります。

ブレークポイントを有効に

[ブレークポイントを有効に] を選択して、選択した行のブレークポイントを有効にします。ブレークポイントの状態によって [ブレークポイントを有効に] と [ブレークポイントを無効に] は切り替わります。

エディタウィンドウでブレークポイントを表示（デバッグメニューの [ブレークポイントを表示] コマンドを選択）している場合、左側にブレークポイント列が表示されます。ブレークポイントを設定したコードには、ブレークポイント列にマーカがあります。濃い色のマーカは有効なブレークポイントを示します。薄い色は無効のブレークポイントを示します。

ブレークポイントを無効に

[ブレークポイントを無効に] を選択して、選択した行のブレークポイントを無効にします。ブレークポイントの状態によって [ブレークポイントを有効に] と [ブレークポイントを無効に] は切り替わります。

ブレークポイントをすべて削除

[ブレークポイントをすべて削除] を選択して、ターゲットプログラムに属するすべてのソースファイルからブレークポイントを消去します。

ブレークポイントを表示

[ブレークポイントを表示] を選択して、エディタウィンドウ左側のブレークポイント列を表示します。ブレークポイント列の状態によって [ブレークポイントを表示] と [ブレークポイントを隠す] は切り替わります。

ブレークポイントを隠す

[ブレークポイントを隠す] を選択して、エディタウィンドウ左側のブレークポイント列を隠します。ブレークポイント列の状態によって [ブレークポイントを表示] と [ブレークポイントを隠す] は切り替わります。

Watchpoint を設定

[Watchpoint を設定] を選択して、選択した変数、またはメモリ範囲の Watchpoint を設定します。Watchpoint の状態によって [Watchpoint を設定] と [Watchpoint を削除] は切り替わります。

アンダーラインが Watchpoint を示します。表示ネルでこのアンダーラインのカラーを設定できます。詳細は「[表示設定](#)」(p201) を参照してください。

Watchpoint を削除

[Watchpoint を削除] を選択して、選択した変数、またはメモリ範囲の Watchpoint を消去します。Watchpoint の状態によって [Watchpoint を設定] と [Watchpoint を削除] は切り替わります。

Watchpoint を有効にする

[Watchpoint を有効にする] を選択して、選択した変数、またはメモリ範囲のウォッチポイントを有効にします。ブレークポイントの状態によって [Watchpoint を有効にする] と [Watchpoint を無効にする] は切り替わります。

選択した変数、またはメモリ範囲の有効なウォッチポイントにはアンダーラインが付きま
す。無効のウォッチポイントにはアンダーラインはありません。表示設定パネルでこのア
ンダーラインのカラーを設定できます。詳細は「[表示設定](#)」(p201)を参照してください。

Watchpoint を無効にする

[Watchpoint を無効にする] を選択して、選択した変数、またはメモリ範囲のウォッチポイ
ントを無効にします。ブレークポイントの状態によって [Watchpoint を有効にする] と
[Watchpoint を無効にする] は切り替わります。

Watchpoint をすべて削除

[Watchpoint をすべて削除] を選択して、現在のプログラムからすべてのウォッチポイント
を消去します。

C++ 例外でブレーク

[C++ 例外でブレーク] を選択して、C++ の例外が発生するたびに、デバッガは `__throw()`
でブレークします。

Java 例外でブレーク

[Java 例外でブレーク] を選択して、Java 例外でブレークサブメニューを表示します。詳細
は「[Java 例外サブメニュー](#)」(p479)を参照してください。

低レベルモニタへ切り替える

[低レベルモニタへ切り替える] を選択して、外部デバッガにコントロールを戻します。

データメニュー

データメニューでデータ値の表示方法を指定します。CodeWarrior デバッガおよび外部デ
バッガでこれらのコマンドをすべて使用できます。データメニューはデータメニューコマ
ンドを使用できるウィンドウがアクティブなときのみ表示されます。

型を表示

[型を表示] を選択して、アクティブな変数欄、変数ウィンドウに表示されたローカル、グ
ローバル変数のデータ型を表示します。

新規 Expression

[新規 Expression] を選択して、Expression ウィンドウに新しい項目を作成します。

Expression へコピー

[Expression へコピー] を選択して、アクティブな欄で選択した変数を評価式ウィンドウへ
コピーします。ソースコードやウィンドウ、欄から評価式をドラッグすることもできます。

表示を変更

[表示を変更] を選択して、選択した変数を指定したデータ型の値で表示します。

変数を表示

[変数を表示] を選択して、新しいウィンドウに選択した変数を表示します。

配列を表示

[配列を表示] を選択して、新しいウィンドウに選択した配列を表示します。

メモリを表示

[メモリを表示] を選択して、メモリの内容を 16 進法、ASCII 文字のダンプで表示します。

メモリの表示を変更

[メモリの表示を変更] を選択して、選択した変数またはレジスタポイントが指すメモリを指定した型で表示します。

デフォルト型で表示

[デフォルト型で表示] を選択して、選択した変数をそのデフォルトの型で表示します。

符号付き 10 進数で表示

[符号付き 10 進数で表示] を選択して、選択した変数を符号付き 10 進法で表示します。

符号なし 10 進数で表示

[符号なし 10 進数で表示] を選択して、選択した変数を符号なし 10 進法で表示します。

16 進数で表示

[16 進数で表示] を選択して、選択した変数を符号なし 16 進法で表示します。

文字で表示

[文字で表示] を選択して、選択した変数を文字で表示します。

C 文字列で表示

[C 文字列で表示] を選択して、選択した変数を C 文字列で表示します。

Pascal 文字列で表示

[Pascal 文字列で表示] を選択して、選択した変数を Pascal 文字列で表示します。

Unicode 文字列で表示

[Unicode 文字列で表示] を選択して、選択した変数を Unicode 文字列で表示します。

浮動小数点型で表示

[浮動小数点型で表示] を選択して、選択した変数を浮動小数点の値で表示します。

列挙型 (enum) で表示

[列挙型 (enum) で表示] を選択して、選択した変数を enum 型 (列挙型) で表示します。

ブラウザメニュー

ブラウザメニューでアクティブなプロジェクトに対して新しいクラス、メンバ関数、データメンバを作成します。ブラウザメニューはブラウザコマンドを使用できるウィンドウがアクティブなときのみ表示されます。

新規クラス

[新規クラス] を選択して、新しいクラスを作成するためのダイアログを表示します。

新規メンバー関数

[新規メンバー関数] を選択して、新しいメンバー関数を作成するためのダイアログを表示します。

新規データメンバー

[新規データメンバー] を選択して、新しいデータメンバーを作成するためのダイアログを表示します。

新規プロパティ

[新規プロパティ] を選択して、選択したクラスの新しい属性を作成するためのダイアログを表示します。

新規メソッド

[新規メソッド] を選択して、選択したクラスの新しいメソッドを作成するためのダイアログを表示します。

新規イベントセット

[新規イベントセット] を選択して、選択したクラスの新しいイベントセットを作成するためのダイアログを表示します。

新規イベント

[新規イベント] を選択して、選択したクラスの新しいイベントを作成するためのダイアログを表示します。

ウィンドウメニュー

ウィンドウメニューには、エディタウィンドウやデバッガウィンドウを開く、ウィンドウを切り替えるコマンドがあります。ツールバーをカスタマイズするサブメニューもあります。

エディタウィンドウを重ねて表示

[エディタウィンドウを重ねて表示] を選択して、すべてのエディタウィンドウを画面全体の大きさに開いて、ウィンドウタイトルだけが見えるように重ねて表示します。このコマンドは、アクティブなウィンドウがプロジェクトウィンドウまたはメッセージウィンドウのときは薄く表示されます。

エディタウィンドウを上下に並べて表示

[エディタウィンドウを上下に並べて表示] を選択して、すべてのエディタウィンドウを重ねないように並べます。このコマンドは、アクティブなウィンドウがプロジェクトウィンドウまたはメッセージウィンドウのときは薄く表示されます。

エディタウィンドウを左右に並べて表示

[エディタウィンドウを左右に並べて表示] を選択して、すべてのエディタウィンドウを重ねないように左右に並べます。

ウィンドウをズーム

[ウィンドウをズーム] を選択して、アクティブなウィンドウを可能な限り最大のサイズに拡張します。再度このコマンドを選択すると、ウィンドウは元のサイズに戻ります。

ウィンドウを最小化

[ウィンドウを最小化] を選択して、アクティブなウィンドウを縮小します。

ウィンドウを元に戻す

[ウィンドウを元に戻す] を選択して、アクティブなウィンドウを元のサイズに戻します。

ウィンドウの設定を保存

[ウィンドウの設定を保存] を選択して、アクティブなウィンドウの設定を保存し、同じ種類のウィンドウを次回開くときに保存した設定を使います。このコマンドは、ブラウザウィンドウ、メッセージウィンドウ、およびエディタウィンドウで使用できます。

このコマンドの詳細は、[「エディタウィンドウの設定を保存」\(p104\)](#) または [「デフォルトのブラウザを保存」\(p179\)](#) を参照してください。

ツールバー

[ツールバー] を選択すると、「ツールバー」サブメニューが表示されます。このサブメニューの詳細は [「ツールバーサブメニュー」\(p480\)](#) を参照してください。

ブラウザコンテンツ

[ブラウザコンテンツ] を選択して、ブラウザの [ブラウザコンテンツ](#) を表示します。ブラウザがアクティブになっていないときは、このコマンドは薄く表示されます。

この機能の詳細は、[「ブラウザコンテンツ」\(p149\)](#) を参照してください。ブラウザをアクティブにする方法は、[「ブラウザを利用」\(p143\)](#) を参照してください。

クラス階層ウィンドウ

[クラス階層ウィンドウ] を選択して、ブラウザの [クラス階層ウィンドウ](#) を表示します。ブラウザがアクティブになっていないときは、このコマンドは薄く表示されます。

この機能の詳細は [「クラス階層ウィンドウ」\(p156\)](#) を参照してください。

ブラウザをアクティブにする方法は、[「ブラウザを利用」\(p143\)](#) を参照してください。

新規クラスブラウザ

[新規クラスブラウザ] を選択して、ブラウザの [ブラウザウィンドウ](#) を表示します。ブラウザがアクティブになっていないときは、このコマンドは薄く表示されます。

この機能の詳細は [「ブラウザウィンドウ」\(p150\)](#) を参照してください。

ブラウザをアクティブにする方法は、[「ブラウザを利用」\(p143\)](#) を参照してください。

ビルド状況ウィンドウ

[ビルド状況ウィンドウ] を選択して、[図 10.1 \(p257\)](#) のビルド進行状況ウィンドウを表示します。

エラーと警告ウィンドウ

[エラーと警告ウィンドウ] を選択して、「エラーと警告」ウィンドウを表示します。

このウィンドウの詳細は、[「メッセージウィンドウの解説」\(p269\)](#) を参照してください。また、[「バッチ検索」\(p129\)](#) を参照してください。

プロジェクトインスペクタ

[プロジェクトインスペクタ] を選択して、プロジェクトに関する情報を表示するほか、デバッグ情報を生成することもできます。

詳細は [「プロジェクトウィンドウの解説」\(p32\)](#) を参照してください。

プロセスウィンドウ

[プロセスウィンドウ] を選択して、プロセスウィンドウを表示します。

Expression ウィンドウ

[Expression ウィンドウ] を選択して、Expressions ウィンドウを表示します。

大域変数ウィンドウ

[大域変数ウィンドウ] を選択して、大域変数ウィンドウを表示します。このウィンドウでプロジェクトやファイルに含まれるグローバル変数を見ることができます。Files list でファイル名をクリックすると Variables リストにグローバル変数が表示されます。

ブレークポイントウィンドウ

[ブレークポイントウィンドウ] を選択して、ブレークポイントウィンドウを表示します。

Watchpoint ウィンドウ

[Watchpoint ウィンドウ] を選択して、Watchpoints ウィンドウを表示します。

レジスタウィンドウ

[レジスタウィンドウ] を選択して、サブメニューを表示します。汎用レジスタか FPU レジスタを見ることができます。

コンポーネントカタログ

[コンポーネント カタログ] を選択して、RAD プロジェクトの「[コンポーネントカタログウィンドウ](#)」を表示します。

コンポーネント パレット

[コンポーネント パレット] を選択して、RAD プロジェクトの「[コンポーネントパレット](#)」を表示します。

オブジェクト インспекタ

[オブジェクト インспекタ] を選択して、RAD プロジェクトの「[オブジェクトインспекタ](#)」を表示します。

その他のウィンドウメニューコマンド

その他に表示されるウィンドウメニュー項目は、開いているプロジェクト、ソースファイル、ヘッダファイル、およびその他のウィンドウによって変わります。

開いているファイルはすべてこのメニューに表示され、最初の 9 ファイルはキーでも選択できます (1 ~ 9)。カレントプロジェクトは、いつも 0 で選択できます。エディタウィンドウを開くには Ctrl/Command + 数字を押します。チェックマークはアクティブなウィンドウを示します。

開いている CodeWarrior ファイルの 1 つをアクティブにしてそのウィンドウを最前面に表示するには、以下のいずれかを行います。

アクティブにしたいウィンドウをクリックする

[ウィンドウメニュー](#) から選択する

[ウィンドウメニュー](#) に示される同等キーを使う

レイアウトメニュー

レイアウトメニューには RAD (Rapid Application Development) レイアウトウィンドウのオブジェクトを操作するコマンドがあります。このメニューはコマンドを使用できるウィンドウが開いているときにのみ表示されます。

配置

レイアウトウィンドウで選択したオブジェクトをグリッドに沿って配置します。[配置] を選択すると、サブメニューが表示されます。複数のレイアウトオブジェクトを選択したのち、配置サブメニューで各オブジェクトを配置できます。

サイズ変更

「レイアウト」ウィンドウで選択したオブジェクトの大きさを変更します。[サイズ変更] を選択すると、サブメニューが表示されます。レイアウトオブジェクトを選択したのち、サイズ変更サブメニューで各オブジェクトの大きさを合わせることができます。

グリッドを表示

[グリッドを表示] を選択して、レイアウトウィンドウにグリッドを表示します。

グリッドに固定

[グリッドに固定] を選択すると、エディタは自動的にオブジェクトをグリッドに合わせて配置します。

グループ化

[グループ化] を選択して、選択したオブジェクトをグループ化します。

グループ化解除

[グループ化解除] を選択して、選択したグループのグループ化を解除します。

カスタマイズ

[カスタマイズ] を選択して、カスタマイズ可能なオブジェクトをカスタマイズします。

プロパティ

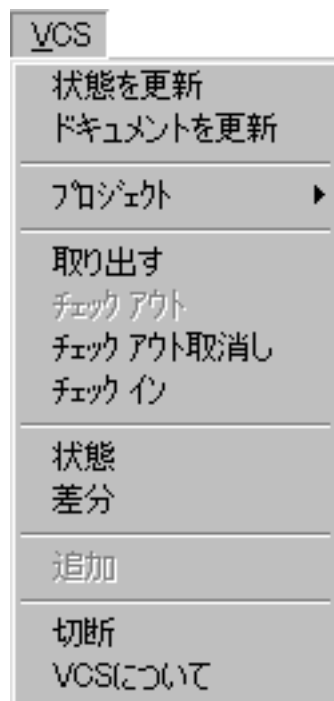
[プロパティ] を選択して、選択したオブジェクトのプロパティを表示します。

バージョン管理システム (VCS) メニュー

バージョン管理システム (VCS) メニュー ([図 17.1](#)) は MW Visual SourceSafe for Macintosh (別売) などのバージョン管理システムをインストールした場合に CodeWarrior IDE のメニューバーに表示されます。

各バージョン管理システムの設定方法はバージョン管理システムに付属のマニュアルをお読みください。

図 17.1 VCS メニュー



ヘルプメニュー

ヘルプメニューから CodeWarrior IDE のオンラインヘルプを利用できます。

CodeWarrior ヘルプ

[CodeWarrior ヘルプ] を選択して、CodeWarrior を使うためのヘルプを表示します。

How to

[How to] を選択して、ステップごとに項目を表示します。

用語集

[用語集] を選択して、用語のリストを表示します。

IDE

[IDE] を選択して、IDE を使うためのヘルプを表示します。

デバッガ

[デバッガ] を選択して、デバッガを使うためのヘルプを表示します。

エラー・リファレンス

IDE エラーを説明するヘルプを表示します。

C/C++ コンパイラ・リファレンス

C/C++ コンパイラを使うためのヘルプを表示します。

MSL C リファレンス

[MSL C リファレンス] を選択して、C 用の MSL (Metrowerks Standard Library) を使うためのヘルプを表示します。

MSL C++ リファレンス

[MSL C++ リファレンス] を選択して、C++ 用の MSL (Metrowerks Standard Library) を使うためのヘルプを表示します。

その他

[その他] を選択して、CodeWarrior 製品を使うためのヘルプを表示します。

About Metrowerks

[Metrowerks について] を選択して、Metrowerks About Box を表示します。

Java 例外サブメニュー

[デバッグメニュー](#)には [Java 例外でブレーク] コマンド用のサブメニューがあります。ここには Java 例外が発生したときの CodeWarrior デバッガの動作を設定するコマンドがあります。これは Java コードを扱っているときのみ表示されます。

すべての例外

[すべての例外] を選択すると、例外が発生するたびにデバッガにブレークします。仮想マシン、独自のクラス、デバッガ、classes.zip のクラスによる例外も含みます。Java は通常の実行で多くの例外を発生するため、頻繁にデバッガにブレークします。

ターゲットクラス内の例外

[ターゲットクラス内の例外] を選択すると、独自のクラスによる例外だけがデバッガにブレークします。通常の実行で発生した例外ではブレークしません。

例外なし

[例外なし] を選択すると、例外が発生してもデバッガにブレークしません。

ツールバーサブメニュー

[ウィンドウメニュー](#)に [ツールバー] コマンドのサブメニューがあります。ツールバーサブメニューには、CodeWarrior IDE ウィンドウに表示されるツールバーをカスタマイズするためのコマンドがすべて含まれています。

ツールバーのカスタマイズ方法の詳細は、[「IDE をカスタマイズ」\(p207\)](#) を参照してください。

ウィンドウツールバーを表示

[ウィンドウツールバーを表示] を選択して、アクティブウィンドウ内のツールバーを表示します。

実際にメニューに表示されるコマンドは、アクティブウィンドウのツールバーの状態によって、[ウィンドウツールバーを閉じる] と [ウィンドウツールバーを表示] の間で切り替わります。

ウィンドウツールバーを閉じる

[ウィンドウツールバーを閉じる] を選択して、アクティブウィンドウ内のツールバーを隠します。

ウィンドウツールバーをリセット

[ウィンドウツールバーをリセット] を選択して、アクティブウィンドウ内のツールバーをデフォルトの状態にリセットします。エディタウィンドウのツールバーを元のデフォルトの状態に戻す場合にこのコマンドを使用します。

ウィンドウツールバーを消去

[ウィンドウツールバーを消去] を選択して、アクティブなエディタ、プロジェクト、ブラウザの各ウィンドウのツールバーのアイコンをすべて削除します。アイコンをすべて削除すると、「IDE コマンドのカスタマイズ」ダイアログ上のアイコンを追加する事ができます。「IDE コマンドのカスタマイズ」ダイアログは、編集メニューの [コマンドとキーバインディング] を選択すると現れます。

デフォルトのアイコンをすべて再設定したい場合は [ウィンドウツールバーをリセット] コマンドを使います。

フロートツールバーを表示

[フロートツールバーを表示] を選択して、このコマンドはフロートツールバーを表示します。

実際にメニューに表示されるコマンドは、フロートツールバーの状態によって [フロートツールバーを表示] と [フロートツールバーを隠す] の間で切り替わります。

フロートツールバーを隠す

[フロートツールバーを隠す] を選択して、このコマンドはフロートツールバーを隠します。

フロートツールバーをリセット

[フロートツールバーをリセット]を選択して、フロートツールバーをデフォルトの状態に戻します。フロートツールバーの元のデフォルトの状態に戻す場合にこのコマンドを使用します。

フロートツールバーを消去

[フロートツールバーを消去]を選択して、フロートツールバーのアイコンをすべて削除します。アイコンをすべて削除すると、「IDE コマンドのカスタマイズ」ダイアログ上のアイコンを追加する事ができます。「IDE コマンドのカスタマイズ」ダイアログは、編集メニューの [コマンドとキーバインディング] を選択すると現れます。

デフォルトのキーバインディング

ここでは CodeWarrior IDE コマンドのデフォルトのキーバインディングを説明します。

キーバインディングのないコマンドの欄は空白になっています。これらのコマンドに対して自分でキーバインディングを割り当てることができます。詳細は「[キーバインディングをカスタマイズ](#)」(p213)を参照してください。

[ファイルメニュー](#)

[編集メニュー](#)

[検索メニュー](#)

[プロジェクトメニュー](#)

[デバッグメニュー](#)

[データメニュー](#)

[ブラウザメニュー](#)

[その他機能](#)

[エディタ](#)

[コンポーネントカタログ](#)

[レイアウトメニュー](#)

[ウィンドウメニュー](#)

ファイルメニュー

[表 17.1](#) に CodeWarrior IDE のファイルメニューのコマンドを実行するデフォルトのキーバインディングを示します。

表 17.1 ファイルメニューのキーバインディング

コマンド	Windows
新規テキストファイル	Ctrl + N
新規	Ctrl + Shift + N
新規プロジェクト	
新規 '空' プロジェクト	
開く	Ctrl + O
過去のファイルを開く	
選択箇所を検索して開く	Ctrl + D
ファイルを検索して開く	Ctrl + Shift + D
閉じる	Ctrl + W
すべて閉じる	Ctrl + Shift + W
上書き保存	Ctrl + S
すべて保存	Ctrl + Shift + S
名前を付けて保存	
コピーを保存	
復帰	
コンポーネントの読み込み	
カタログを閉じる	
プロジェクトの読み込み	
プロジェクトの書き出し	
ページ設定	
印刷	Ctrl + P
終了	

編集メニュー

[表 17.2](#) に CodeWarrior IDE の Edit メニューのコマンドを実行するデフォルトのキーバインディングを示します。

表 17.2 編集メニューのキーバインディング

コマンド	Windows
元に戻す	Ctrl + Z
やり直し	Ctrl + Shift + Z
切り取り	Ctrl + X
コピー	Ctrl + C
貼り付け	Ctrl + V
削除	Del
すべてを選択	Ctrl + A

コマンド	Windows
バランス	Ctrl + B
左に移動	Ctrl + [
右に移動	Ctrl +]
リファレンステンプレートを挿入	
環境設定	
ターゲット設定	
バージョン管理設定	
コマンドとキーバインディング	

検索メニュー

[表 17.3](#) に CodeWarrior IDE の検索メニューのコマンドを実行するデフォルトのキーバインディングを示します。

表 17.3 検索メニューのキーバインディング

コマンド	Windows
検索	Ctrl + F
次を検索	F3
前を検索	
次のファイル内を検索	Ctrl + T
前のファイル内を検索	
選択部分を検索文字列へ	Ctrl + E
選択部分を置換文字列へ	
選択部分を検索	Ctrl + F3
選択部分を逆方向検索	
置換	Ctrl + =
置換後次を検索	Ctrl + L
置換後逆方向検索	
すべて置換	
定義を検索	Ctrl + '
定義とリファレンスを検索	
リファレンスを検索	
戻る	Ctrl + Shift + B
次へ	Ctrl + Shift + F
指定の行へ移動	Ctrl + ,
ファイル比較	
差分を適用する	
差分の適用を取り消す	

プロジェクトメニュー

[表 17.4](#) に CodeWarrior IDE のプロジェクトメニューのコマンドを実行するデフォルトのキーバインディングを示します。

表 17.4 プロジェクトメニューのキーバインディング

コマンド	Windows
ウィンドウを追加	
ファイルを追加	
新規グループ/ セグメント/ ターゲット作成	
シンタックスをチェック	Ctrl + ;
プリプロセス	
プリコンパイル	
コンパイル	Ctrl + F7
逆アセンブル	
最新状態に更新する	Ctrl + U
メイク	F7
ビルドを中止	Ctrl + Break
オブジェクトを削除	Ctrl + -
オブジェクトコードの削除と圧縮	
ファイルパスを再検索	
プロジェクト内のパスをリセット	
ファイル更新日時を同期させる	
デバグガを有効にする	
実行/ デバグ	F5
デバグ/ 実行	Ctrl + F5
デフォルトプロジェクトを設定	
デフォルトターゲットを設定	

デバグメニュー

[表 17.5](#) に CodeWarrior IDE のデバグメニューのコマンドを実行するデフォルトのキーバインディングを示します。

表 17.5 デバグメニューコマンドのキーバインディング

コマンド	Windows
中止	Shift + F5
再起動	Ctrl + Shift + F5
リセット	
ハードリセット	

コマンド	Windows
ステップオーバー	F10
ステップイン	F11
ステップアウト	Shift + F11
停止	
ブレークポイントの設定 / 削除	F9
ブレークポイントを有効 / 無効に	Ctrl + F9
ブレークポイントをすべて削除	
ブレークポイントを表示 / 隠す	
Watchpoint を設定 / 削除	
Watchpoint を有効 / 無効にする	
Watchpoint をすべて削除	
C++ 例外でブレーク	
Java 例外でブレーク	
低レベルモニタへ切り替える	
プロセスヘアタッチ	
スタックロールウィンドウ	

データメニュー

[表 17.6](#) に CodeWarrior IDE のデータメニューのコマンドを実行するデフォルトのキーバインディングを示します。

表 17.6 データメニューのキーバインディング

コマンド	Windows
型を表示	
新規 Expression	Alt + N
Expression へコピー	
表示を変更	
変数を表示	
配列を表示	
メモリを表示	
メモリの表示を変更	
デフォルト型で表示	
符号付き 10 進数で表示	Alt + Shift + D
符号なし 10 進数で表示	Alt + Shift + U
16 進数で表示	Alt + Shift + H
文字で表示	Alt + Shift + C
C 文字列で表示	Alt + Shift + S
Pascal 文字列で表示	Alt + Shift + P

コマンド	Windows
Unicode 文字列で表示	Alt + Shift + O
浮動小数点型で表示	Alt + Shift + F
列挙型 (enum) で表示	Alt + Shift + E
Fixed で表示	
Fract で表示	

ブラウザメニュー

[表 17.7](#) に CodeWarrior IDE のブラウザメニューのコマンドを実行するデフォルトのキーバインディングを示します。

表 17.7 ブラウザメニューのキーバインディング

コマンド	Windows
新規クラス	
新規メンバー関数	
新規データメンバー	
新規プロパティ	
新規メソッド	
新規イベントセット	
新規イベント	

その他機能

[表 17.8](#) に CodeWarrior IDE のその他のコマンドを実行するデフォルトのキーバインディングを示します。

表 17.8 その他のキーバインディング

コマンド	Windows
クオートキー	
ヘッダー / ソースファイルへ	Ctrl + `
前のエラーメッセージ	Shift + F4
次のエラーメッセージ	F4
Run Script	
Stop Script	

エディタ

[表 17.9](#) に CodeWarrior IDE のエディタウィンドウを操作するデフォルトのキーバインディングを示します。

表 17.9 エディタウィンドウ用のキーバインディング

コマンド	Windows
左へ 1 文字移動	
右へ 1 文字移動	
左へ 1 語移動	Ctrl +
右へ 1 語移動	Ctrl +
左へ複合語内の 1 語分移動	
右へ複合語内の 1 語分移動	
現在行の先頭に移動	Home
現在行の行末に移動	End
上へ 1 行移動	
下へ 1 行移動	
現在ページの先頭行に移動	Page Up
現在ページの最終行に移動	Page Down
ファイルの先頭に移動	Ctrl + Home
ファイルの末尾に移動	Ctrl + End
左の 1 文字を削除	Backspace
右の 1 文字を削除	Del
行末まで削除	
左の 1 文字を選択	Shift +
右の 1 文字を選択	Shift +
左の 1 語を選択	Ctrl + Shift +
右の 1 語を選択	Ctrl + Shift +
左の複合語内の 1 語を選択	
右の複合語内の 1 語を選択	
上へ 1 行分選択	Shift +
下へ 1 行分選択	Shift +
現在行の先頭まで選択	Shift + Home
現在行の行末まで選択	Shift + End
現在ページの先頭行まで選択	Shift + Page Up
現在ページの最終行まで選択	Shift + Page Down
ファイルの先頭まで選択	Ctrl + Shift + Home
ファイルの末尾まで選択	Ctrl + Shift + End
上へ 1 行スクロール	Ctrl +
下へ 1 行スクロール	Ctrl +
前へ 1 ページ分スクロール	
後へ 1 ページ分スクロール	
ファイルの先頭行までスクロール	
ファイルの最終行までスクロール	

コマンド	Windows
選択位置までスクロール	
指定語で始まるシンボルを検索	Ctrl + ¥
指定語を含むシンボルを検索	Ctrl + Shift + ¥
次のシンボル	Ctrl + .
前のシンボル	

コンポーネントカタログ

[表 17.10](#) に CodeWarrior IDE の「コンポーネントカタログ」ウィンドウを操作するデフォルトのキーバインディングを示します。

表 17.10 「コンポーネントカタログ」ウィンドウ用のキーバインディング

コマンド	Windows
新規カタログ	
新規フォルダ	
カタログを開く	
インデックスの表示の切り替え	
項目のプロパティを編集	

レイアウトメニュー

[表 17.11](#) CodeWarrior IDE の RAD レイアウトエディタを操作するデフォルトのキーバインディングを示します。

表 17.11 レイアウトメニューのキーバインディング

コマンド	Windows
配置	
サイズ変更	
グリッドを表示	
グリッドに固定	
グループ化	
グループ化解除	
カスタマイズ	
プロパティ	

ウィンドウメニュー

[表 17.12](#) に CodeWarrior IDE のウィンドウを操作するデフォルトのキーバインディングを示します。

表 17.12 ウィンドウメニューのキーバインディング

コマンド	Windows
エディタウィンドウを重ねて表示	
エディタウィンドウを上下に並べて表示	
エディタウィンドウを左右に並べて表示	
ウィンドウをズーム	Ctrl + /
ウィンドウを縮小 / 拡大	
ウィンドウの設定を保存	
ツールバー	
ブラウザコンテンツ	
クラス階層ウィンドウ	
単一クラス階層ウィンドウ	
新規クラスブラウザ	
ビルド状況ウィンドウ	
エラーと警告ウィンドウ	
プロジェクトインスペクタ	
ToolServer ワークシート	
プロセスウィンドウ	
Expression ウィンドウ	
大域変数ウィンドウ	
ブレークポイントウィンドウ	
Watchpoint ウィンドウ	
Register Windows	
コンポーネント カタログ	
コンポーネント パレット	
オブジェクトインスペクタ	
デフォルトプロジェクトを選択	Ctrl + 0
ドキュメント 1 を選択	Ctrl + 1
ドキュメント 2 を選択	Ctrl + 2
ドキュメント 3 を選択	Ctrl + 3
ドキュメント 4 を選択	Ctrl + 4
ドキュメント 5 を選択	Ctrl + 5
ドキュメント 6 を選択	Ctrl + 6
ドキュメント 7 を選択	Ctrl + 7
ドキュメント 8 を選択	Ctrl + 8
ドキュメント 9 を選択	Ctrl + 9

付録 A Perl スクリプト

CodeWarrior プロジェクトで、プリフィックスファイルとして Perl スクリプトを実行することができます。このためにはソフトウェアプラグインのインストールが必要です。次に CodeWarrior に Perl スクリプトを認識させるための設定を行います。

以下の項目について説明します。

[Perl ソフトウェアプラグインをインストール](#)

[Perl 設定パネルの設定](#)

[Perl スクリプトの例](#)

[注意点](#)

Perl ソフトウェアプラグインをインストール

CodeWarrior はさまざまなソフトウェアプラグインを利用してその機能を拡張しています。IDE に Perl スクリプトを認識、実装させるためには Perl プラグインが必要です。プラグインの種類はホストプラットフォームによって異なります。Perl プラグインのインストール方法を説明します。

CodeWarrior に Perl スクリプト機能を実装するためには以下のプラグインが必要です。MWPerl.dll プラグイン、PerlDLL.dll プラグイン、MWPerl Panel.dll プラグイン、Perl 拡張ファイル (Perl5 フォルダ)。

Perl ツールは「CW_PRO5」CD の Thrill Seekers\Perl Tools フォルダに圧縮ファイルがあります。解凍してご利用ください。Perl ツールは Pro5 ではプレリリース版です。

MWPerl.dll および PerlDLL.dll プラグインを Compiler フォルダに置いてください。以下にフォルダへのパスを示します。

```
\Program Files\Metrowerks\CodeWarrior\Bin\Plugins\Compiler\
```

MWPerl Panel.dll プラグインを Preference Panel フォルダに置いてください。以下にフォルダへのパスを示します。

```
\Program Files\Metrowerks\CodeWarrior\Bin\Plugins\Preference Panel
```

PerlDLL.dll プラグインと Perl 拡張ファイル (Perl5 フォルダ) を support フォルダに置いてください。以下にフォルダへのパスを示します。

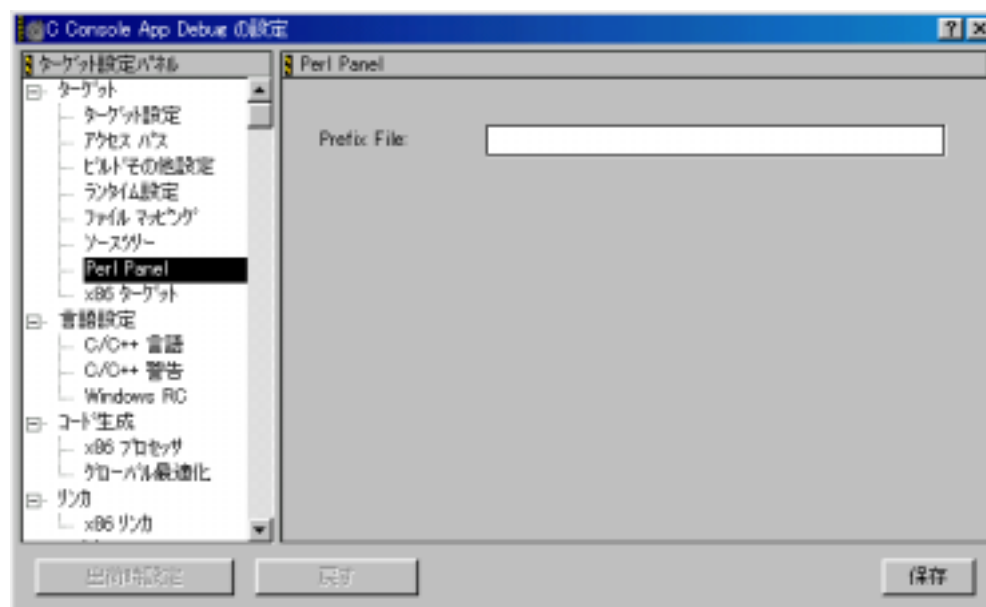
\Program Files\Metrowerks\CodeWarrior\Bin\Plugins\support

Perl 設定パネルの設定

ソフトウェアプラグインをインストールした後、CodeWarrior プロジェクトで Perl スクリプトをプリフィックスファイルとして扱うことができます。Perl Panel 設定パネルで Perl スクリプトを指定します。CodeWarrior はこのスクリプトを暗黙のファイルとして扱います。Perl スクリプトの `require` 疑似命令は、C/C++ の `#include` 疑似命令と等価です。

Perl Panel 設定パネルを表示するには、編集メニューの [*Target* の設定] (*Target* には現在のターゲット名が入ります) を選択してください。ターゲット設定ダイアログ ([図 A.1](#)) が現れます。

図 A.1 Perl 設定パネル



ターゲット設定ダイアログ左側のリストから [Perl Panel] を選択してください。右側に Perl Panel 設定パネルが表示されます ([図 A.1](#))。

プリフィックスファイルとして Perl スクリプトをプロジェクトへ追加するためには、[Implicit require file] フィールドに Perl スクリプトの名前を入力します。CodeWarrior はプロジェクトをビルドするときに、このファイルを暗黙の Perl `require` ファイルとして扱います。

注意： CodeWarrior 用の Perl プラグインは IDE の検索 / ロード機能を利用します。この機能は IDE が絶対パスを使って参照ファイルを検索する機能に影響されます。絶対パスで参照されていない Perl スクリプトや暗黙の `require` ファイルへのアクセスパスを必ず指定

してください。パス情報はアクセスパス設定パネルで設定します。[「アクセスパス」\(p231\)](#)を参照してください。

Perl スクリプトの例

IDE のプラグイン API (Application Programming Interface) を利用して Perl スクリプトを実装してます。

[リスト A.1](#) は Perl スクリプトのサンプルです。

リスト A.1 Perl スクリプトのサンプル

```
# Simple Perl Example

# Print a line of text
print "Hello World!\n";

$scale0 = 0;
$scale1 = 1;
$scale2 = 2.5;

# Create and open a file for output
open (theFile, ">output.txt");

# Dump some text into the file
print theFile "The file should now be open\n";
print theFile "Let's try a few things:\n\n";

# Arithmetic
print theFile "**Arithmetic \n";
print theFile $scale1 + $scale2 . "\n";
print theFile $scale1 * $scale2 . "\n";
print theFile $scale1 % $scale2 . "\n\n";

# Boolean logic
print theFile "**Boolean \n";
print theFile ($scale0 && $scale0) . "\n";
print theFile ($scale0 && $scale1) . "\n";
print theFile ($scale1 && $scale1) . "\n";

print theFile ($scale0 || $scale0) . "\n";
print theFile ($scale0 || $scale1) . "\n";

print theFile (!$scale0) . "\n\n";

# Comparison
print theFile "**Comparisons \n";
print theFile ($scale2 == $scale2) . "\n";
```

```
print "That's it, closing file\n";

# Close the file
close theFile;
```

注意点

Perl スクリプトを認識して CodeWarrior プロジェクトへ実装する際に、注意すべき点があります。

[StdIn の用途](#)

[リンカエラーを避ける](#)

StdIn の用途

プリフィックスファイルは StdIn をサポートしていません。つまり、Perl スクリプトはキーボード入力を受け付けません。

リンカエラーを避ける

CodeWarrior リンカは Perl スクリプトをプリフィックスファイルとして扱います。ゆえにリンカはそのプリフィックスファイルをプロジェクト内のソースコードに適用しようとしめます。プロジェクトにソースコードが含まれない場合、プロジェクトをコンパイルしたときにリンカは以下のエラーメッセージを表示します。

```
Link Error: 'main' is undefined.
```

プロジェクトのコンパイルが成功して、スクリプトが実行できても、リンカエラーが発生します。このエラーメッセージを避けるには、プロジェクトにソースコードが含まれていること、またはターゲット設定パネルの [リンカ] オプションが [None] であることを確認してください。詳しくは [「ターゲット設定」\(p229\)](#) を参照してください。

索引

記号

#include ファイル 166
\$ (変数名の) 358
(VCS) 448
.ctlg 拡張子 407
.dump 268
.MAP 261
.mcp 拡張子 42, 368
.mkb 拡張子 217
? (変数名の) 358
__throw() 471

数字

10 進数
 データ入力 338
16 進数
 データを表示 336
 データ入力 338
16 進数で表示 472
1 行ずつステップ実行 315

A

About Metrowerks 479
absbottom
 Java アプレットウィザード 373
absmiddle
 Java アプレットウィザード 373
Abstract
 指定子ポップアップメニュー 432, 381, 437
ANSI
 Pascal プロジェクトの C コード 360
Arithmetic Optimizations
 Details
 Global Optimizations 設定パネル 249
AWT
 RAD コンポーネント 415 ~ 416
AWT アプレット
 Java アプレットウィザード 371
AWT フレーム
 Java アプリケーションウィザード 378
 Java フレームウィザード 397

B

baseline
 Java アプレットウィザード 373
Bean
 基底クラスと実装 381
 名前と位置を指定 379
Bean のクラス
 新規 Java イベントセットウィザード 435
binary
 viewing data as 336
bottom
 Java アプレットウィザード 373
Branch Optimizations
 Details
 Global Optimizations 設定パネル 249
Build Settings preference panel
 Show Message After Building Up-To-Date Project
 checkbox 184
Button コンポーネント 415

C

C++
 デバッグ 205
 メソッド (アルファベット順) 202, 291
C++, Object Pascal、SOM のダイナミックタイプ
 を利用する
 表示設定パネル 202
C++ 例外でブレイク 471
C/C++ コンパイラ・リファレンス 479
Cache Symbolics Between Runs checkbox
 of Debugger Settings panel 253
Cancel ボタン
 Makefile Importer ウィザード 48
Canvas コンポーネント 415
Checkbox コンポーネント 415
Choice コンポーネント 415
Class 項目
 コンポーネント情報バー 405
ClearCase VCS プラグイン 443
CodeWarrior
 IDE 1.7 プロジェクトを最新版へ変換 51
 IDE の定義 15
 RAD コンポーネント 415

- RAD ツールの定義 28
- RAD デザインとレイアウト 365
- RAD レイアウトエディタ 389
- VCS 29
 - インストール 24
 - カスタマイズ 29
 - システムの必要条件 23
- 紹介 27 ~ 29
- スクリプト 29
- 相対パス 235, 245
- ソースコードのデバッグ 277
- ソフトウェア生成 26
- ターゲットオプションの設定 225
- ツール 16
 - はじめに 23
 - ブラウザ 143
 - ブラウザ (RAD) 423
 - プログラミングの概念 24 ~ 26
 - プロジェクト 31
 - メイクファイルをプロジェクトへ変換 47 ~ 49
 - メニューリファレンス 455
 - リファレンス情報 455
- 紹介 15
- CodeWarrior IDE
 - 概要 15
- CodeWarrior IDE について 15 ~ 16
- CodeWarrior QuickStart 24
- CodeWarrior ヘルプ 478
- COM 188
- Comments 項目
 - コンポーネント情報バー 405
- Common Object Model 188
- Common Subexpression Elimination
 - Details
 - Global Optimizations 設定パネル 249
- Const チェックボックス
 - 新規メンバー関数ウィザード 169
 - 新規データメンバーウィザード 172
- Content ビューのボタン
 - コンポーネントカタログウィンドウ 403
- Copy Propagation
 - Details
 - Global Optimizations 設定パネル 249
- Core Tutorials フォルダ 20
- .ctlg 拡張子 407
- Customize Object コマンド 413

- CVS
 - VCS プラグイン 443
- C 言語関数ポップアップ判定条件を緩める 195
- C ブレークポイントをすべて削除 470
- C 文字列
 - データを表示 336
 - データ入力 338
- C 文字列で表示 472

D

- Dead Code Elimination
 - Details
 - Global Optimizations 設定パネル 249
- Dead Store Elimination
 - Details
 - Global Optimizations 設定パネル 250
- Debugger Settings panel
 - Cache Symbolics Between Runs checkbox 253
- Delete コマンド 413
- Details
 - Global Optimizations 設定パネル 248
- Diagnostic Settings オプション 49
- DLL 205, 252
 - 自動的にデバッグ 205, 252
- DOS テキストファイル 84
- .dump 268

E

- Electric Fish
 - CVS プラグイン 443
 - Projector プラグイン 443
- Emacs テキストエディタ 185
- End キー 33
- Expression Simplification
 - Details
 - Global Optimizations 設定パネル 249
- Expressions ウィンドウ 297, 339, 475
 - 項目を追加 339
 - 順序の変更 340
 - 定義 297, 339
 - 呼び出し元の変数を追加 340
- Expression ヘコピー 297, 339, 471

F

- Faster Execution Speed

Optimize For
 Global Optimizations 設定パネル 249
 FDI (Floating Document Interface) 186
 Final
 指定子ポップアップメニュー 381, 432, 437
 Finish ボタン
 Makefile Importer ウィザード 48
 Fixed
 データを表示 337
 FPU レジスタ 285, 287
 FPU レジスタウィンドウ 305
 Fract
 データを表示 337

G

General Registers 304, 342
 Generate ボタン
 Makefile Importer ウィザード 48
 Global Optimizations 設定パネル 248
 Details 248
 Arithmetic Optimizations 249
 Branch Optimizations 249
 Common Subexpression Elimination 249
 Copy Propagation 249
 Dead Code Elimination 249
 Dead Store Elimination 250
 Expression Simplification 249
 Global Register Allocation 249
 Instruction Scheduling 251
 Lifetime Based Register Allocation 251
 Live Range Splitting 250
 Loop Transformations 250
 Peephole Optimization 249
 Repeated 251
 Strength Reduction 250
 Vectorization 250
 Optimization Level Slider 248
 Optimize For 248
 Faster Execution Speed 249
 Smaller Code Size 249
 Global Register Allocation
 Details
 Global Optimizations 設定パネル 249
 GNU make ファイルをプロジェクトへ変換 40
 grep 136

H

Home キー 33
 How To 478
 HTML
 Java アプレットウィザードでページを指定 371
 HTML ページを指定
 Java アプレットウィザード 371
 HTML ページを生成
 Java アプレットウィザード 371
 Hypertext Markup Language HTML を参照 371

I

IDE 478
 IDE 1.7 プロジェクトを最新版へ変換 51
 RAD コンポーネント 415
 RAD ツールの定義 28
 RAD デザインとレイアウト 365
 RAD レイアウトエディタ 389
 VCS 29
 インストール 24
 オンラインドキュメント 113
 カスタマイズ 29, 207 ~ 224
 設定パネル 181
 概要 15
 システムの必要条件 23
 紹介 15, 27 ~ 29
 スクリプト 29
 設定ダイアログのボタン 182
 設定パネルを選択 183 ~ 207
 ソースコードのデバッグ 277
 ソフトウェア生成 26
 ターゲットオプションの設定 225
 ターゲット設定ダイアログのボタン 226
 ツール 16
 定義 15
 デフォルトのキーバインディング 481 ~ 489
 はじめに 23
 ブラウザ 143, 423
 プログラミングの概念 24 ~ 26
 プロジェクト 31
 メニューコマンドを追加 211
 メニューリファレンス 455
 リファレンス情報 455
 IDE User Guide
 各章の概要 16
 IDE その他設定パネル 185

- エディタを起動フィールド 185
- 過去のプロジェクトフィールド 185
- IDE 環境設定
 - 解説 181 ~ 183
- IDE コマンドのカスタマイズダイアログ
 - キーバインディング欄 210
 - コマンドタブ 209
 - 実行フィールド 212
 - 自動リピートチェックボックス 216
 - テンキーバインディングチェックボックス 214, 215
 - ディレクトリフィールド 212
 - 動作欄 210
 - 名前 210
 - 名前フィールド 210, 211, 212
 - 引数フィールド 212
 - 変更を破棄 208
 - ボタン 208
 - メニューに表示 210
 - メニューに表示チェックボックス 211, 212
- キャンセルボタン
 - IDE コマンドのカスタマイズダイアログ 208
- 自動リピートチェックボックス
 - IDE コマンドのカスタマイズダイアログ 216
- 設定パネル 181
- IDE 設定パネル
 - Java 設定 207
 - 一般設定 204
 - ウィンドウ指定 203
 - カラーシンタックス 198
 - 選択 183 ~ 207
 - 表示設定 201
 - ビルド設定 183
 - フォント & タブ 196
 - 変数表示域内で変数の型を表示する 336
- IDE その他設定パネル
 - Multiple Document Interface を利用チェックボックス 186
- IDE その他設定パネル
 - MDI 使用 186
 - 過去のドキュメントフィールド 185
 - 外部エディタを利用チェックボックス 185
 - 行番号指定でエディタを起動フィールド 185
 - サードパーティエディタを使う 185
- IDE 設定ダイアログ 181
 - 一般設定 183
 - エディタの環境設定 193
 - カラーシンタックス設定パネル 198
- ソースツリー設定パネル 188
- デバッガの IDE 設定 201
- ビルド設定パネル 183
- フォント & タブ設定パネル 196
- プラグイン設定パネル 186
- 変更を破棄 182
- ボタン 182
- Implementor で表示 152
- Inherited を表示チェックボックス 155
- ブラウザ 178
- Inherited アクセスアイコン 155
- Inline チェックボックス
 - 新規メンバー関数ウィザード 169
- Instruction Scheduling
 - Details
 - Global Optimizations 設定パネル 251
- Integrated Development Environment 15
- Internet Protocol. See IP.
- Intersolv
 - PVCS プラグイン 443
- IP 206
- J**
- Java 21
 - AWT RAD コンポーネント 415 ~ 416
 - Swing RAD コンポーネント 416 ~ 418
- Java Bean ウィザード 379
 - 可視 Bean オプション 382
 - 基底クラス 382
 - 基底クラスと実装 381
 - クラスは Bean 380
 - クラス名 380
 - 指定子欄 381
 - 修飾子 381
 - 新規ダイアログ 367
 - 新規ファイル 381
 - 実装リスト 382
 - 名前と位置を指定 379
 - パッケージ名 381
 - ファイル 381
 - 不可視 Bean/Custom オプション 382
- Java デバッグ設定パネル
 - タイムアウトフィールド 206
 - プロトコルポップアップメニュー 206
 - リモート IP フィールド 206
- Java アプリケーション
 - クラスを指定 376

- Java アプリケーションウィザード 375, 378
 - AWT フレーム 378
 - Swing フレーム 378
 - アプリケーションクラスを指定 376
 - アプリケーションタイトル 378
 - 起動時にフレームを作成 377
 - クラス名 376, 377
 - 新規ダイアログ 367
 - すべてのファイルに情報を挿入 379
 - 説明 379
 - 追加情報 378
 - 場所 376, 378
 - パッケージ名 376, 377
 - フレームクラスを指定 377
- Java アプレット
 - クラスの指定 369
- Java アプレットウィザード 369
 - absbottom 373
 - absmiddle 373
 - AWT アプレット 371
 - baseline 373
 - bottom 373
 - HTML ページを指定 371
 - HTML ページを生成 371
 - left 373
 - middle 373
 - right 373
 - Swing アプレット 371
 - texttop 373
 - top 373
 - アーカイブ使用 (JAR) 372
 - アプレットクラスの指定 369
 - アプレットタイトル 375
 - アプレット名 372
 - 型 374
 - クラス名 370
 - コードベース / アーカイブ 372
 - 削除ボタン 374
 - 新規ダイアログ 367
 - 垂直余白 372
 - 水平余白 372
 - スタンドアローンアプリケーションとして実行 371
 - すべてのファイルに情報を挿入 375
 - 説明 375
 - タイトル 372
 - 高さ 372
 - 追加情報 374
 - 追加ボタン 374
- 配置ポップアップメニュー 373
 - 場所 370
 - パッケージ名 370
 - パラメータの型 373
 - パラメータの初期値 373
 - パラメータの説明 373
 - パラメータの変数 373
 - パラメータの名前 373
 - パラメータを作成 373
 - 標準メソッドを生成 371
 - 横幅 372
- 追加ボタン
 - Java アプレットウィザード 374
- Java デバッグ設定パネル 205
 - JDK バージョンポップアップメニュー 207
 - ポート ID フィールド 206
 - リモートデバッグ 206
- Java フレーム
 - クラスを指定 396
- Java フレームウィザード 396
 - AWT フレーム 397
 - Swing フレーム 397
 - クラス名 397
 - 場所 397
 - パッケージ名 397
 - フレームのクラスを指定 396
- Java 設定パネル 207
 - JView 引数 207
 - デバッグクラス 207
 - プログラム引数 207
- Java 例外でブレーク 471
- Java 例外サブメニュー 479
 - すべての例外 479
 - ターゲットクラス内の例外 479
 - 例外なし 479
- JButton コンポーネント 416
- JCheckBox コンポーネント 416
- JComboBox コンポーネント 416
- JDK バージョンポップアップメニュー
 - java デバッグ設定パネル 207
- JEditorPane コンポーネント 416
- JLabel コンポーネント 416
- JList コンポーネント 416
- JMenuBar コンポーネント 417
- JOptionPane コンポーネント 417
- JPanel コンポーネント 417
- JPasswordField コンポーネント 417

JPopupMenu コンポーネント 417
JProgressBar コンポーネント 417
JRadioButton コンポーネント 417
JScrollBar コンポーネント 417
JScrollPane コンポーネント 417
JSlider コンポーネント 417
JSplitPane コンポーネント 417
JTabbedPane コンポーネント 417
JTable コンポーネント 417
JTextArea コンポーネント 417
JTextField コンポーネント 417
JTextPane コンポーネント 418
JToggleButton コンポーネント 418
JToolBar コンポーネント 418
JTree コンポーネント 418
JView 引数
 Java 設定パネル 207

L

Label コンポーネント 415
left
 Java アプレットウィザード 373
Lifetime Based Register Allocation
 Details
 Global Optimizations 設定パネル 251
List View ボタン 403
List コンポーネント 415
List ビュー
 と Live ビュー 403
Live Range Splitting
 Details
 Global Optimizations 設定パネル 250
Live View ボタン 403
Live ビュー
 RAD レイアウト 403
 と List ビュー 403
Lock 項目
 コンポーネント情報バー 405
Log All Statements Bypassed チェックボックス 49
Log Build Rules Discarded チェックボックス 49
Log Targets Bypassed チェックボックス 49
Log ウィンドウ 306
Loop Transformations
 Details

 Global Optimizations 設定パネル 250
Loop Unrolling
 Details
 Global Optimizations 設定パネル 250
Loop-Invariant Code Motion
 Details
 Global Optimizations 設定パネル 250
lvalue (左辺式)
 定義 310

M

Mac OS 21
Mainsoft
 SourceSafe プラグイン 444
Makefile Importer ウィザード
 Cancel ボタン 48
 Finish ボタン 48
 Generate ボタン 48
Makefile Location オプション
 Makefile Importer ウィザード 48
Makefile インポータ 40
Makefile インポーター 47 ~ 49
Makefile インポータウィザード
 オプション 48
.mcp 拡張子 368
MDI 203
MDI (Multiple Document Interface)186
MDI 使用
 IDE その他設定パネル 186
MenuBar コンポーネント 415
Metrowerks
 CVS プラグイン 443
 Projector プラグイン 443
 SourceSafe プラグイン 444
Metrowerks Tool Set ポップアップメニュー 49
MFC
 ブラウザでクラスを表示 178
Microsoft
 SourceSafe プラグイン 444
Microsoft Windows 23
middle
 Java アプレットウィザード 373
MIPS 21
.mkb 拡張子 217
Modified 項目

コンポーネント情報バー 405
 MSL C++ リファレンス 479
 MSL C リファレンス 479
 Multiple Document Interface を利用チェックボックス
 IDE その他設定パネル 186
 Mutable
 指定子ポップアップメニュー 172
 MW Perl.dll 491
 MWPerl Panel.dll 491

N

Name 項目
 コンポーネント情報バー 405
 Native
 新規メソッドダイアログ 432
 Nucleus 21

O

Object Pascal
 メソッド, アルファベット順 291
 OK ボタン 235, 245
 Optimization Level Slider
 Global Optimizations 設定パネル 248
 Optimize For
 Global Optimizations 設定パネル 248
 options
 Caching symbolics between runs 253

P

Page Down キー 33, 105
 Page Up キー 33, 105
 Palm OS 21
 Panel コンポーネント 416
 Pascal
 循環参照を修正 274
 Pascal、C コード 360
 Pascal 文字列
 データを表示 336
 データ入力 338
 Pascal 文字列で表示 472
 Peephole Optimization
 Details
 Global Optimizations 設定パネル 249
 Perforce

Perforce プラグイン 443
 Perforce VCS プラグイン 443
 PerlDLL.dll 491
 Perl スクリプト 491
 IDE プラグイン API 493 ~ 494
 注意点 494
 Perl スクリプトのサンプル 493
 Perl ソフトウェアプラグイン
 インストール 491
 Perl プラグイン
 Windows ヘインストール 491
 Perl 設定パネル 492
 設定 492 ~ 493
 PlayStation 21
 PopupMenu コンポーネント 416
 PowerPC エンベデッドシステム 21
 PowerPlant
 ブラウザでクラスを表示 178
 precompile_target 264
 preferences
 Show Message After Building Up-To-Date Project
 184
 Private
 アクセスポップアップメニュー 169, 172, 432,
 437
 Private を表示 152
 Projector VCS プラグイン 443
 Protected
 アクセスポップアップメニュー 169, 172, 432,
 437
 Protected を表示 152
 コマンド
 Protected を表示 152
 Public
 アクセスポップアップメニュー 169, 172, 381,
 432, 437
 Public を表示 152
 Pure Virtual
 指定子ポップアップメニュー 169
 PVCS VCS プラグイン 443

Q

QuickStart 20, 24
 QuickStart とチュートリアル 20

R

RAD 398

- Bean の基底クラスと実装 381
- CodeWarrior アーキテクチャ 365
- Content ビューのボタン 403
- Java アプリケーションのクラスを指定 376
- Java アプレットクラスの指定 369
- Java フレームのクラスを指定 396
- RAD ブラウザのダイアログ 427 ~ 441
- アプリケーションのフレームクラスを指定 377
- アプレットのパラメータを作成 373
- アプレットの追加情報 374, 378
- アプレット用の HTML ページを指定 371
- イベントセットの基底クラスと実装を指定 438
- イベントセットの名前と場所を指定 435
- イベントセットをビルドターゲットへ割り当て 439
- イベントビュー 426
- ウィザード 367
 - Java Bean ウィザード 383
 - Java アプリケーションウィザード 383
 - Java アプレットウィザード 383
- オブジェクトのイベント 411
- オブジェクトの属性 411
- オブジェクトインスペクタ 409 ~ 413
- カタログ 366
- カタログポップアップメニュー 398
- カタログ欄 402
- 概要 365
- クラスオーサリング 366
- コンポーネント 366, 415
- コンポーネントのソート順 405
- コンポーネントをソート 405
- コンポーネントエディタ 418 ~ 422
- コンポーネントカタログを作成 406
- コンポーネントカタログウィンドウ 400 ~ 409
- コンポーネントカタログツールバー 401
- コンポーネントツール 399
- コンポーネントパレット 398 ~ 400
- コンポーネントモデル 366
- コンポーネント情報バー 404
- コンポーネント欄 402
- デザインの定義 66
- デザインをプロジェクトへ追加 383
- デザインを作成 383 ~ 385
- デザインビュー 385

- ビーンの名前と位置を指定 379
- ブラウザの機能 423 ~ 427
- ブラウザコントロールタブ 423
- プロパティビュー 424
- メソッドビュー 425
- 利点 365
- レイアウトを作成 386 ~ 388, 389
- レイアウトを修正 390
- レイアウトウィザード 396 ~ 397
- レイアウトエディタ 366, 389, 389 ~ 396
- レイアウトオブジェクトを作成 391
- レイアウトオブジェクトを削除 393
- レイアウトオブジェクトを操作 392

RAD ウィザード

- 完了ボタン 434
- キャンセルボタン 434
- 次へボタン 434
- 戻るボタン 434

40, 369 ~ 382

- Java Bean ウィザード 367
- Java アプリケーションウィザード 367
- Java アプレットウィザード 367
- Java フレームウィザード 390
- 完了ボタン 369, 396
- キャンセルボタン 369, 396
- 作成ボタン 369, 396
- 生成ボタン 369, 375, 379, 383
- 次へボタン 369, 396
- ナビゲーション 369
- 戻るボタン 369, 396

RAD オブジェクト

- 検査 392
- 複数選択 392

RAD コンポーネント

- Java AWT 415 ~ 416
- Java Swing 416 ~ 418

RAD ツール

- CodeWarrior 365 ~ 366
- 定義 28

RAD プロジェクト

- 作成 367 ~ 368

Rational

- ClearCase プラグイン 443

Registers フォルダ 300

Repeated

- Details

- Global Optimizations 設定パネル 251

require 疑似命令 492

right
 Java アプレットウィザード 373

S

Scrollbar コンポーネント 416
 ScrollPane コンポーネント 416
 Select All コマンド 413
 Settings オプション 49
 Show Message After Building Up-To-Date Project
 checkbox
 of Build Settings preference panel 184
 Smaller Code Size
 Optimize For
 Global Optimizations 設定パネル 249
 SourceSafe VCS プラグイン 444
 Static
 指定子ポップアップメニュー 169, 172, 432, 437
 指定子ポップアップメニュー
 Static 437
 Strength Reduction
 Details
 Global Optimizations 設定パネル 250
 Swing
 RAD コンポーネント 416 ~ 418
 Swing アプレット
 Java アプレットウィザード 371
 Swing フレーム
 Java アプリケーションウィザード 378
 Java フレームウィザード 397
 SYM ファイルが開かれたら自動的にアプリケーションを起動する
 一般設定パネル 205
 Synchronized
 新規メソッドダイアログ 432
 Synergex
 PVCS プラグイン 443

T

Target の設定 266, 267, 268, 461
 TextArea コンポーネント 416
 TextField コンポーネント 416
 texttop
 Java アプレットウィザード 373
 __throw() 471

Tool Set Used In Makefile ポップアップメニュー 49
 top

 Java アプレットウィザード 373

Transient
 新規プロパティダイアログ 429

U

UNI Software
 Voodoo プラグイン 444
 unicode string
 viewing data as 336
 Unicode 文字列で表示 472
 UNIX テキストファイル 84
 Unselect ツール
 コンポーネントパレット 400

V

VCS 29
 VCS について (VCS) 449
 VCS ポップアップメニュー 156, 449, 451, 453
 エディタ 99
 VCS メッセージウィンドウ 452
 VCS メニュー 447
 VCS ログインダイアログ 449
 VCS 設定パネル 445
 カスタムプロジェクト設定を使用 445
 Vectorization
 Details
 Global Optimizations 設定パネル 250
 View Alphabetical コマンド 413
 View In Groups コマンド 413
 Virtual デストラクタチェックボックス
 新規クラスウィザード 165
 Visual C 変換 nmake ファイルをプロジェクトへ 40
 Visual Sourcesafe SourceSafe を参照 444
 Volatile
 新規プロパティダイアログ 429
 Volatile チェックボックス
 新規データメンバーウィザード 172
 VOODOO
 VCS プラグイン 444

W

Watchpoint
 ウィンドウ 299

- 消去 299
- 設定と削除 331
- 表示 332
- 制限 331
- 設定 331
- 定義 330
- Watchpoint で停止する
 - デバッグ設定パネル 252, 253
- Watchpoint をすべて削除 331, 471
- Watchpoint を削除 300, 470
- Watchpoint を設定 331, 470
- Watchpoint を無効にする 471
- Watchpoint を有効にする 470
- Watchpoint ウィンドウ 299, 332, 476
- Watchpoint 強調色
 - 表示設定パネル 202
- Win32/x86 21
- X
- XML ファイル
 - レジスタ詳細ウィンドウ 300
- ア行
- アーカイブ使用 (JAR)
 - Java アプレットウィザード 372
- アクセスポップアップメニュー
 - 新規 Java イベントセットウィザード 437
 - 新規クラスウィザード 165
- アクセスパス 231
 - リカーシブ検索 232
- アクセスパス設定パネル 231
 - 必ずユーザーパスを検索する 233
 - 削除 235
 - システムパス 233
 - システムパス欄 233
 - 追加 234
 - 追加ボタン 234
 - デフォルトを追加 233
 - 変更ボタン 235
 - ホストフラグ 234
 - ユーザパスボタン 233
 - ユーザパス欄 233
- アクセスフィルタポップアップメニュー 152
- アクセスポップアップメニュー
 - Private 169, 172, 432, 437
 - Protected 169, 172, 432, 437
 - Public 169, 172, 381, 432, 437
- アクセッサ
 - 新規プロパティダイアログ 429
- アクティブ 74
- アセンブラ 26
 - 表示 286, 293
- アセンブリ言語
 - 表示 278
 - メモリ表示 285, 287
 - レジスタ表示 285, 287
- 値をリセットボタン
 - レジスタ詳細ウィンドウ 303
- アドレスフィールド
 - レジスタ詳細ウィンドウ 301
- アプリケーション
 - フレームクラスを指定 377
- アプリケーションクラスを指定
 - Java アプリケーションウィザード 376
- アプリケーションタイトル 378
- アプレット
 - HTML ページを指定 371
 - 追加情報 374, 378
 - パラメータを作成 373
- アプレットのパラメータを作成
 - Java アプレットウィザード 373
- アプレットクラスの
 - Java アプレットウィザード 369
- アプレットタイトル
 - Java アプレットウィザード 375
- アプレット名
 - Java アプレットウィザード 372
- アルファベット順
 - C++ メソッド 202
- アンタッチ
 - ファイル 63
- 一時的なブレークポイント 326, 317
 - 普通のブレークポイントへの影響 326
- 一覧欄
 - 新規プロパティダイアログ 430
- 一般設定
 - ランタイム設定パネル 238
- 一般設定パネル 204
 - SYM ファイルが開かれたら自動的にアプリケーションを起動する 205
 - キャッシュ内のファイルを維持フィールド 204

- キャッシュをパージボタン 204
- 自動ターゲットライブラリ 205
- タスクを停止したときにプログラムウィンドウを選択する 205
- ディレクトリ階層内のすべての Java クラスのデバッグを行う 204
- デバッグを終了する際にプロセスの停止を確認する 205
- デバッグ作業中に編集ファイルをキャッシュする 204
- デバッグ中にファイル修正日時の不正について確認する 204
- ランタイムサポートコード内にはステップインしない 205
- 一般設定
 - IDE 設定ダイアログ 183
- 一般的なナビゲーション 320
- 移動
 - ファイルとグループ 61
 - プロジェクト 72 ~ 73
- イベントセット
 - 基底クラスと実装を指定 438
 - ビルドターゲットへ割り当て 439
 - 名前と場所を指定 435
- イベントセットダイアログ
 - 新規イベントセット 433
- イベントタブ
 - オブジェクトインスペクタ 411
- イベントビュー
 - ブラウザ (RAD) 426
- インクルードファイル 25
 - 新規クラスウィザード 166
- 印刷 458
 - ウィンドウ 86
 - ウィンドウを印刷 86
 - カラーシンタックスを利用して印刷 87
 - シンタックスカラー 87
 - 選択部分のみ印刷 87
- インストール
 - CodeWarrior IDE 24
- インストラクションスケジューリング 251
- インターフェースファイル 25
- インデックスビューの切替えボタン 402
- インポート
 - カスタムキーワード 200
 - キーバインディング 217
 - コマンド 217
 - プロジェクト 73
- ウィザード
 - Java Bean 379
 - Java Bean ウィザード 367, 383
 - Java アプリケーション 375
 - Java アプリケーションウィザード 367, 383
 - Java アプレット 369
 - Java アプレットウィザード 367, 383
 - Java フレーム 396
 - Java フレームウィザード 390
 - RAD 369 ~ 382
 - RAD プロジェクト 367
 - RAD レイアウト 396 ~ 397
 - 新規クラス 162
 - 新規データメンバー 171
 - 新規メンバー関数 167
 - ブラウザ 162 ~ 173
 - 新規 Java イベントセット 434
- ウィザード (RAD プロジェクト) 40
- ウィンドウ
 - Expressions ウィンドウ 297
 - クラスブラウザ 150 ~ 156
 - 大域変数ウィンドウ 297
 - ブラウザ 289
 - プロジェクトウィンドウのオーバーレイビュー 32
 - プロジェクトウィンドウのセグメントビュー 32
 - プロジェクトウィンドウのターゲットビュー 32, 39
 - プロジェクトウィンドウのツールバー 33
 - プロジェクトウィンドウのデザインビュー 32, 38
 - プロジェクトウィンドウのナビゲート 33
 - プロジェクトウィンドウのファイルビュー 32, 33
 - プロジェクトウィンドウのリンク順ビュー 32, 39
- ウィンドウメニュー
 - 大域変数ウィンドウ 476
- ウィンドウの位置とサイズ
 - エディタ設定パネル 195
- ウィンドウの設定を保存 474
- ウィンドウをズーム 474
- ウィンドウを元に戻す 474
- ウィンドウを最小化 474
- ウィンドウを追加 60, 265, 266
- ウィンドウ指定設定パネル

- プロジェクトウィンドウを変更しない 203
- ウィンドウツールバーをリセット 224, 480
- ウィンドウツールバーを隠す 221
- ウィンドウツールバーを消去 223, 480
- ウィンドウツールバーを表示 221, 283, 480
- ウィンドウツールバーを閉じる 283, 480
- ウィンドウメニュー 474 ~ 476
 - Expressions ウィンドウ 475
 - Watchpoint ウィンドウ 476
 - ウィンドウの設定を保存 474
 - ウィンドウをズーム 474
 - ウィンドウを元に戻す 474
 - ウィンドウを最小化 474
 - エディタウィンドウを左右に並べて表示 474
 - エディタウィンドウを重ねて表示 474
 - エディタウィンドウを上下に並べて表示 474
 - エラーと警告ウィンドウ 475
 - オブジェクトインスペクタ 476
 - クラス階層ウィンドウ 475
 - コンポーネントカタログ 476
 - コンポーネントパレット 476
 - 新規クラスブラウザ 475
 - ツールバーサブメニュー 474
 - ビルド状況ウィンドウ 475
 - ブラウザコンテンツ 475
 - ブレークポイントウィンドウ 476
 - プロジェクトインスペクタ 475
 - プロセスウィンドウ 475
 - レジスタウィンドウ 476
- ウィンドウを追加 465
- ウィンドウを追加コマンド 60
- ウィンドウ指定設定パネル 203
 - デバッグ開始時設定 203
- エクストラ情報ボタン 270
- エクスポート
 - カスタムキーワード 200
 - キーバインディング 217
 - コマンド 217
 - プロジェクト 73
- エディタ 95 ~ 113
 - RAD コンポーネント 418 ~ 422
 - RAD ポップアップメニュー 421
 - RAD メニューバー 418
 - VCS ポップアップ 99
 - ウィンドウ設定を保存 104
 - オプションポップアップメニュー 99
 - 括弧のバランス 107
 - カラーシンタックス 109
- 関数ポップアップメニュー 98, 110
- 関連するファイルを開く、112
- ガイドツアー 95 ~ 101
- 行番号ボタン 100
- 行番号で移動 112
- 構成 101 ~ 104
- サードパーティエディタ 185
- ツールバー拡張ボタン 101
- 定義を検索 114
- テキスト編集 104 ~ 109
- テキスト編集域 97
- テキストを移動 107
- テキストを削除 105
- テキストを選択 106
- テキストを追加 105
- 取り消し、変更を 108 ~ 109
- ドラッグ & ドロップ 196
- ナビゲート、テキストを 109 ~ 113
- ファイル表示欄 100
- フォント 101, 196
- フォント設定 195
- ヘッダポップアップメニュー 97
- マーカ 110 ~ 112
- マーカポップアップメニュー 98
- メニューを作成 419
- メニューアイテム 420
- メニューアイテムを削除 421
- 文字サイズ 101, 196
- 戻ると次へ 113
- ユーザーインターフェース 95
- 欄 102 ~ 104
- 欄分割コントロール 100
- エディタの環境設定
 - IDE 設定ダイアログ 193
- エディタの設定
 - ターゲット設定ダイアログ 251
- エディタを起動フィールド
 - IDE その他設定パネル 185
- エディタウィンドウを左右に並べて表示 474
- エディタウィンドウを重ねて表示 474
- エディタウィンドウを上下に並べて表示 474
- エディタ設定パネル 194
 - ウィンドウの位置とサイズ 195
 - カラー設定 194
 - 選択部分 195
 - その他 195
 - 左端をクリックして行を選択するチェックボックス 196

- 標準テキストファイル形式ポップアップメニュー 196
- フォント設定 195
- フラッシュ時間フィールド 196
- 保存項目 195
- エラー・リファレンス 479
- エラーと警告ウィンドウ 475
- エラーのみ
 - プラグイン診断 186
- エラーメッセージ 271
 - コンパイラの 272
 - デバッグ 360
- オートインデント
 - フォント & タブ設定パネル 197
- オーバーレイビュー 32
- オブジェクト
 - RAD レイアウト 391
 - RAD レイアウトから削除 393
 - RAD レイアウトで操作 392
- オブジェクトを削除 466
- オブジェクトインスペクタ 409 ~ 413, 476
 - イベントタブ 411
 - オブジェクトポップアップメニュー 410
 - コンテキストメニュー 412
 - ナビゲーションキー 411
 - プロパティタブ 411
- オブジェクトインスペクタコマンド 409
- オブジェクトコード 26
- オブジェクトコードの削除と圧縮 467
- オブジェクトタブ
 - 新規ダイアログ 390
- オブジェクトポップアップメニュー
 - オブジェクトインスペクタ 410
 - キーボードナビゲーション 410
- オブジェクトを削除 261
- オプション 181
 - IDE その他設定 185
 - Makefile インポートウィザード 48
 - エディタ設定 194
 - 解説 181
 - 上級者向けコンパイルオプション 262
 - 設定パネル 183
 - ターゲット設定 228 ~ 253
 - ビルド設定 183
 - ブラウザ表示 193
 - プラグイン設定 186
- オプションポップアップメニュー

- エディタ 99
- オンラインドキュメント用ビューア 20
- オンラインヘルプとリファレンス 113

力行

- 拡張
 - 変数を 284
- 解説
 - IDE 環境設定 181 ~ 183
 - ビルドターゲットの設定 225 ~ 227
- 階層ビュー
 - ブラウザ 146
- 書き込みボタン
 - レジスタ詳細ウィンドウ 303
- 書き出しボタン
 - IDE コマンドのカスタマイズダイアログ 217
- 拡張
 - グループ 54
 - 変数を 308, 333
- 拡張子
 - mcp 42
 - ファイルマッピング設定パネル 239, 240
- 過去のドキュメントフィールド
 - IDE その他設定パネル 185
- 過去のファイルを開く 50
- コマンド
 - 過去のファイルを開く 185
- 過去のファイルを開く 185, 456
- 過去のプロジェクトフィールド
 - IDE その他設定パネル 185
- 可視 Bean オプション
 - Java Bean ウィザード 382
- 可視コンポーネント
 - 新規 Java イベントセットウィザード 438
- カスタマイズ 396, 477
 - IDE 29, 207 ~ 224
 - キーバインディング 213
 - コマンド 209
 - ツールバー 219
 - メニューコマンド 211, 212
- カスタム
 - 新規 Java イベントセットウィザード 438
- カスタムキーワード
 - 削除 199
- カスタムキーワードダイアログ
 - 終了ボタン 199

- 追加ボタン 199
- 編集ボタン 199
- カスタムキーワード
 - インポート 200
 - エクスポート 200
 - カラー 198, 199
- カスタムキーワード設定パネル 251
 - 編集ボタン 251
- カスタムコマンド
 - 動作の定義 212
- カスタムプロジェクト設定を使用 (VCS) 445
- 型
 - Java アプレットウィザード 374
 - 新規プロパティダイアログ 430, 428
 - ソースツリー設定パネル 189, 244
- 型に必要なパッケージ
 - 新規プロパティダイアログ 428
- 型に必要な名前空間
 - 新規データメンバーウィザード 172
- 型フィールド
 - 新規データメンバーウィザード 172
- 型ポップアップメニュー
 - 環境変数 244
 - 絶対パス 244
 - レジストリキー 244
- カタログ
 - 定義 366
- カタログを開くボタン 398
- カタログを閉じる 458
- カタログを閉じるボタン 399
- カタログポップアップメニュー 398
- カタログを閉じるボタン 402
- カタログを開くボタン 401
- カタログ欄
 - コンポーネントカタログウィンドウ 402
- 型を表示 335, 358, 471
- 括弧のバランス
 - エディタ 107
- カット 107
- カテゴリポップアップメニュー
 - ブラウザコンテンツ 150
- 必ずユーザーパスを検索する
 - アクセスパス設定パネル 233
- カラー
 - カスタムキーワード 198, 199
 - キーワード 198
 - コメント 198
 - ストリング 198
- カラーシンタックス 198, 199
 - エディタ 109
- カラーシンタックスを利用して印刷 87
- カラーシンタックスオプション
 - 印刷 87
- カラーシンタックス設定パネル 198
- カラー設定
 - エディタ設定パネル 194
- 空のプロジェクト 40
- 空プロジェクトオプション
 - 新規ダイアログ 41
- カレントステートメント矢印 286
 - ステートメントをスキップ 317
 - 定義 314
 - ドラッグ 317
 - ブラウザウィンドウ 292
 - ブレークポイントの 324
 - ステートメントをスキップ 317
 - ドラッグ 317
- 環境設定 181, 461
 - Java デバッグ 205
 - ランタイム設定パネル 238
- 環境変数
 - 型ポップアップメニュー 244
 - 削除 238
 - 作成 238
 - ソースツリー設定パネル 189
 - 変更 238
- 関数
 - 名前と位置の指定 168
 - ファイル位置 172
 - ファイル位置を指定 169
- 関数ポップアップメニュー
 - エディタ 110
- 288, 290, 293
 - アルファベット順に表示 288, 293
 - エディタ 98
- 関数ポップアップメニューをソート 196
- 関数名
 - 自動補完 176
- 関数欄 293
- 関数をすべて検索 159
- 管理
 - プロジェクトのファイル 53 ~ 63
- 完了ボタン

- RAD ウィザード 434, 369, 396
- ブラウザウィザード 162
- 関連するファイルを開く 82
- 概念
 - プログラミング 24 ~ 26
- 外部エディタ 185
- 外部エディタを利用 185
- 外部エディタを利用チェックボックス
 - IDE その他設定パネル 185
- 概要
 - RAD 365
- キー
 - End キー 33
 - Home キー 33
 - Page Down 105
 - Page Down キー 33
 - Page Up 105
 - Page Up キー 33
 - クオートキー 217
 - プレフィックスキー 217
 - 矢印キー 33
- キーバインディング
 - インポート 217
 - エクスポート 217
 - カスタマイズ 213
 - 削除 216
 - 制限 214
 - 追加 215
 - デフォルト 481 ~ 489
 - 変更 214
- キーバインディング欄
 - IDE コマンドのカスタマイズダイアログ 210
- キーワード
 - カラー 198
- 記述情報欄
 - レジスタ詳細ウィンドウ 303
- 記述ファイルフィールド
 - レジスタ詳細ウィンドウ 301
- 既存のイベントセット
 - 新規イベントセットダイアログ 434
- 既存ファイル選択ボタン
 - 新規クラスウィザード 164
- 既存ボタン
 - 新規メンバー関数ウィザード 170
- 基底クラス
 - Java Bean ウィザード 382
- 基底クラスフィールド
 - 新規クラスウィザード 165
- 基底クラス欄
 - 新規 Java イベントセットウィザード 438
- 基底クラスとコンストラクタ引数に必要な名前クラス
 - 新規クラスウィザード 165
- 基底クラスとメソッド
 - 新規クラスウィザード 164
- 基底クラスと実装
 - Java Bean ウィザード 381
- 基底クラスと実装を指定
 - 新規 Java イベントセットウィザード 438
- 基底クラスのために自動的に追加される #include ファイル
 - 新規クラスウィザード 166
- 起動
 - デバッグ 311
- 起動可能
 - フラグポップアップメニュー 242
- 起動時にフレームを作成
 - Java アプリケーションウィザード 377
- 起動時に接続する (VCS) 446
- 基本的なデバッグ 311 ~ 343
- キャッシュをパージボタン
 - 一般設定パネル 204
- キャッシュ内のファイルを維持フィールド
 - 一般設定パネル 204
- キャンセルボタン 182
 - IDE コマンドのカスタマイズダイアログ 208
- RAD ウィザード 434
- 新規プロパティダイアログ 430
- 新規メソッドダイアログ 433
- RAD ウィザード 369, 396
- アクセスパス 235, 245
- 新規イベントダイアログ 441
- ターゲット設定
 - 警告ダイアログ 227
- ブラウザウィザード 162
- 式共通部分式の削除 249
- 強度の低減 250
- 局所変数
 - デバッグで表示 333
- 切り取り 459
- 逆アセンブル 184, 268, 466
 - ソースコードを 268
- 行番号指定でエディタを起動 フィールド

- IDE その他設定パネル 185
- 行番号で移動
 - エディタ 112
- 行番号ボタン
 - エディタ 100
- クオートキー 217
 - 割り当て 218
- クラス
 - #include ファイルを指定 166
 - 基底クラスとメソッドを指定 164
 - 新規 Java イベントセットウィザード 437
 - 名前と位置を指定 162
 - ビルドターゲットへ割り当て 166
- クラスに対応
 - 新規 Java イベントセットウィザード 437
- クラスは Bean
 - Java Bean ウィザード 380
- クラスオーサリング
 - RAD 366
- クラス項目
 - コンポーネント情報バー 405
- クラス相対
 - 新規クラスウィザード 163
- クラスフィールド
 - 新規クラスウィザード 163
- クラス名
 - Java Bean ウィザード 380
 - Java アプリケーションウィザード 376, 377
 - Java アプレットウィザード 370
 - Java フレームウィザード 397
 - クラス名ウィザード 163
 - 新規クラスウィザード 163
 - 自動補完 176
 - 新規 Java イベントセットウィザード 435
- クラス階層ウィンドウ 475
- クラス宣言ボタン
 - ブラウザ 154
- クラス欄
 - ブラウザ 153
 - 表示されない項目 153
- クラス欄ボタン (ブラウザ) 154
- クラッシュ
 - デバッガ 354
- グリッドに固定 395, 477
- グリッドを表示 477
- グリッド表示 395
- グループコード 360
- グループ 55
 - 移動 61
 - 拡張と縮小 54
 - 削除 61
 - 作成 61
 - 選択 55
 - デバッグ情報マーカ 75
 - ドラッグ&ドロップで削除 62
 - 名前 62
- グループポップアップメニュー 79
- グループ化 395, 477
- グループ化解除 396, 477
- グローバル最適化設定パネル 248
 - コードサイズの最小化ボタン 249
 - 最適化レベルスライダー 248
 - 最適化目的 248, 249
 - 実行速度の高速化 249
 - 最適化レベルスライダー 249
 - ショウサイ 248
 - 詳細 249
 - 実行速度の高速化ボタン 249
- 警告メッセージ 271
- 検査
 - RAD オブジェクト 392
- 検索 461
 - 関数のすべての実装を 159
 - 最近指定した文字列ポップアップメニュー 125
 - 選択したテキスト 123 ~ 125
 - 選択部分を検索文字列へ 124
 - 他のウィンドウのテキストを 124
 - 単語 127
 - 大小文字無視 127
 - 置換
 - 最近指定した文字列 128
 - すべて置換 128
 - 次を検索 128
 - 次を検索 124
 - テキストを検索 125
 - 複数のファイル 130 ~ 136
 - 複数ファイル 120
 - 複数ファイルをアクティブにする 131
 - 前を検索 124, 126
- 検索メニュー
 - ファイル比較 464
- 検索ダイアログ
 - 紹介 115

- 選択部分を逆方向検索 462
- 検索メニュー 461 ~ 464
 - 次を検索 461
 - 検索 461
 - 差分の適用を取り消す 464
 - 差分を適用する 464
 - 指定の行へ移動 464
 - すべて置換 463
 - 選択部分を逆方向検索 462
 - 選択部分を検索 462
 - 選択部分を検索文字列へ 462
 - 選択部分を置換文字列へ 462
 - 置換 463
 - 置換後逆方向検索 463
 - 置換後次を検索 463
 - 次のファイル内を検索 462
 - 次へ 464
 - 次を検索 461
 - 定義を検索 114, 463
 - 前のファイル内を検索 462
 - 前を検索 462
 - 戻る 464
- 検証
 - プロジェクト情報 71 ~ 72
- 更新
 - プロジェクト 258
- 更新日付
 - 同期 63
- 後方
 - 新規 Java イベントセットウィザード 437
 - 新規クラスウィザード 164
- 項目
 - ツールバー 220
- 項目のプロパティを編集ボタン 402
- コード
 - 1 行ずつ実行 315
 - アセンブリ言語で表示 278
 - 混合で表示 287
 - コンパイルとリンク 28
 - 実行 313
 - 実行を終了 319
 - 実行を停止 319
 - ステップ実行 313
 - ソースファイル 24
 - 停止 313
 - デッドストリップ 35
 - デバッグで実行 314
 - デバッグと修正 26
 - ナビゲーション 320
 - フォントとカラーを選択 324
 - ブラウザでナビゲート 175
 - ブラウザウィンドウでナビゲーション 321
 - ルーチンヘステップイン 316
 - ルーチンをステップアウト 316
 - 編集とブラウザ 27
- コードのナビゲート
 - コンテキストメニュー 175
 - 戻る、次へ 175
- コードの最適化
 - ブレークポイントへの影響 329
- コードの修正 26
- コードをステップ実行 313
- コードを実行 313
- コードを停止 313
- コードサイズの最小化ボタン
 - グローバル最適化設定パネル 249
- コードナビゲーション
 - 一般的 320
 - コールチェーン 320
 - ファイル欄 322
 - ブラウザウィンドウ 321
- コードナビゲートコンテキストメニュー
 - ブラウザ 148
- コードベース / アーカイブ
 - Java アプレットウィザード 372
- コード生成の設定
 - ターゲット設定ダイアログ 247
- コード列 34
- コールチェーンによるナビゲーション 320
- コピー 107, 460
 - Log ウィンドウ 307
- コピーによる伝播 249
- コピーを保存 83
- コピーを保存コマンド 52
- コピーを保存 457
- コマンド 458
 - 16 進数で表示 472
 - C++ 例外でブレーク 471
 - C/C++ コンパイラ・リファレンス 479
 - CodeWarrior ヘルプ 478
 - Customize Object 413
 - C 文字列で表示 472
 - Delete 413
 - Expressions ウィンドウ 297, 339, 475
 - Expression へコピー 297, 471

- FPU レジスタ 305
- FPU レジスタウィンドウ 305
- General Registers 304, 342
- How To 478
- IDE 478
- Implementor で表示 152
- Java 例外でブレーク 471
- MSL C++ リファレンス 479
- MSL C リファレンス 479
- Pascal 文字列で表示 472
- Private を表示 152
- Public を表示 152
- Select All 413
- Target の設定 266, 267, 268, 461
- Unicode 文字列で表示 472
- View Alphabetical 413
- View In Groups 413
- Watchpoint をすべて削除 471
- Watchpoint を削除 300, 470
- Watchpoint を設定 331, 470
- Watchpoint を無効にする 471
- Watchpoint を有効にする 470
- Watchpoint ウィンドウ 299, 332, 476
- インサツ 458
- インポート 217
- ウィンドウの設定を保存 474
- ウィンドウをズーム 474
- ウィンドウを元に戻す 474
- ウィンドウを最小化 474
- ウィンドウを追加 265, 266, 465
- ウィンドウツールバーをリセット 224, 480
- ウィンドウツールバーを隠す 221
- ウィンドウツールバーを消去 223, 480
- ウィンドウツールバーを表示 221, 283, 480
- ウィンドウツールバーを閉じる 283, 480
- ウィンドウを追加 60
- エクスポート 217
- エディタウィンドウを左右に並べて表示 474
- エディタウィンドウを重ねて表示 474
- エディタウィンドウを上下に並べて表示 474
- エラー・リファレンス 479
- エラーと警告ウィンドウ 475
- オブジェクトを削除 466
- オブジェクトインスペクタ 409, 476
- オブジェクトコードの削除と圧縮 467
- オブジェクトを削除 261
- 過去のファイルを開く 50, 185, 456
- カスタマイズ 209, 396, 477
- カタログを閉じる 458
- 型を表示 335, 471
- カラーシンタックス 199
- 環境設定 181, 461
- 切り取り 459
- 逆アセンブル 184, 268, 466
- クラス階層ウィンドウ 475
- グリッドに固定 395, 477
- グリッドを表示 477
- グリッド表示 395
- グループ化 395, 477
- グループ化解除 396, 477
- 検索 461
- コピー 307, 460
- コピーを保存 52, 83, 457
- コマンドとキーバインディング 207, 461
- コンパイル 184, 258, 466
- コンポーネントの読み込み 458
- コンポーネントカタログ 400, 476
- コンポーネントパレット 398, 476
- 再起動 468
- 最新状態に更新する 258, 259, 264, 466
- 再実行 459
- サイズ変更 395, 477
- 削除 393, 460
- 作成 211
- サブクラスで表示 152
- 差分の適用を取り消す 464
- 差分を適用する 464
- 指定語で始まるシンボルを検索 176
- 指定語を含むシンボルを検索 176
- 指定の行へ移動 464
- 終了 458
- 消去 62
- 調べる 210
- 新規 40, 367, 390, 406, 456
- 新規 Expression 471
- 新規イベント 439, 473
- 新規イベントセット 433, 473
- 新規クラス 161, 473
- 新規クラスブラウザ 475
- 新規グループを作成 465
- 新規ターゲットを作成 465
- 新規テキストファイル 264, 265, 267, 456
- 新規データメンバー 161, 473
- 新規プロパティ 428, 473
- 新規メソッド 161, 430
- 新規メンバー関数 473
- シンタックスをチェック 465
- 実行 184, 259, 260, 281, 283, 313, 314, 315, 319, 468
- ステップアウト 283, 313, 316, 469

- ステップイン 283, 313, 316, 469
- ステップオーバー 283, 313, 315, 469
- すべて閉じる 457
- すべての変数ウィンドウを閉じる 298
- すべての例外 479
- すべてを保存 457
- すべてを選択 460
- 選択部分を逆方向検索 462
- 選択部分を検索 462
- 選択部分を検索文字列へ 462
- 選択部分を置換文字列へ 462
- その他 479
- ターゲットクラス内の例外 479
- 大域変数ウィンドウ 476
- タッチする 63
- 大域変数ウィンドウ 333
- 置換 463
- 置換後逆方向検索 463
- 置換後次を検索 463
- チュウシ 319
- 中止 283, 313, 315, 468
- 次のシンボル 176
- 次のファイル内を検索 462
- 次へ 464
- 次を検索 461
- 定義を検索 114, 463
- 停止 283, 313, 314, 319, 469
- 低レベルモニタへ切り替える 471
- テンプレートを挿入 (コンテキストメニュー)
) 194
- デバッグ 478
- デバッグを無効にする 259, 279, 468
- デバッグを有効にする 259, 279, 467
- デバッグ 238, 281
- デフォルトターゲットを設定 468
- デフォルトプロジェクトを設定 468
- デフォルト型で表示 472
- 閉じる 406, 457
- 取り消し 459
- 動作の定義 212
- 名前変更 406
- 名前を付けて保存 83, 307, 457
- 入力再実行 459
- 入力取り消し 459
- ハードリセット 469
- 配置 395, 477
- 背面へ移動 395
- 配列を表示 307, 472
- 張り付け 460
- バージョン管理設定 461
- バランス 460
- エディタ 107
- 左に移動 460
- エディタ 108
- 表示を変更 472
- 開く 50, 456
- ビルドを中止 466
- ビルド状況ウィンドウ 475
- ファイルを追加 465
- ファイル更新日時を同期させる 261
- ファイルのコピー保存 52
- ファイルパスを再検索 467
- ファイル比較 464
- ファイルを検索して開く 456
- ファイルを追加 57
- ファイル更新日時を同期させる 467
- ファイル名を検索して開く 456
- 符号付き 10 進数で表示 472
- 符号なし 10 進数で表示 472
- 復帰 457
- 浮動小数点型で表示 473
- フロートツールバーをリセット 481
- フロートツールバーを隠す 480
- フロートツールバーを消去 481
- フロートツールバーを表示 480
- ブラウザコンテンツ 475
- ブレークポイントの設定 469
- ブレークポイントをすべて削除 326, 470
- ブレークポイントを隠す 326, 470
- ブレークポイントを削除 326, 469
- ブレークポイントを設定 / 削除する場所 325
- ブレークポイントを表示 326, 470
- ブレークポイントを無効にする 326
- ブレークポイントを有効に 469
- ブレークポイントを有効にする 326
- ブレークポイントウィンドウ 298, 326, 476
- プリコンパイル 184, 263, 264, 265, 266, 268, 466
- プリプロセス 184, 465
- プリプロセッサ 268
- プログラムカウンタを変更 318
- プロジェクトの書き出し 73, 458
- プロジェクトの読み込み 458
- プロジェクトインスペクタ 475
- プロジェクトの読み込み 73
- プロジェクト内のパスをリセット 467
- プロセスウィンドウ 294, 475
- プロパティ 396, 406, 477
- 変更 210
- 変数を表示 298, 472

- ペースト 406
- 保存 457
- 前のシンボル 176
- 前のファイル内を検索 462
- 前を検索 462
- 右に移動 460
 - エディタ 108
- メイク 184, 259, 264, 466
- メインツールバーをリセット 224
- メインツールバーを隠す 221
- メインツールバーを消去 223
- メインツールバーを表示 221
- メニューバーから削除 212
- メニューバーに追加 211
- メモリの表示を変更 308, 309, 340, 472
- メモリを表示 340, 472
- 文字で表示 472
- 戻る 464
- 戻る、次へ 175
- 戻ると次へ
 - エディタ 113
- ユーザで表示 152
- 用語集 478
- リセット 469
- 例外なし 479
- レジスタウィンドウ 304
- レジスタ詳細ウィンドウ 300
- 列挙型で表示 473
- ロック 406
- ロック解除 406
- 新規 383
- データメニュー
 - 符号付き 10 進数で表示 472
- バージョン管理システム (VCS)
 - コマンド
 - 状態 448
 - 状態を更新 448
 - 切断 449
 - 追加 448
- コマンドとキーバインディング 207, 461
- コマンドグループ
 - 削除 213
 - 作成 211
- コマンドタブ
 - IDE コマンドのカスタマイズダイアログ 209
- コメント
 - カラー 198
- コメント (VCS) 449
- 混合コード
 - 表示 287, 293
- コンストラクタとデストラクタを生成
 - 新規クラスウィザード 165
- コンストラクタ引数フィールド
 - 新規クラスウィザード 165
- コンテキストメニュー
 - オブジェクトインスペクタ 412
 - コンポーネントカタログ 405
 - デバッグ 294
 - レイアウトエディタ 395
- コンテキストメニューにテンプレートコマンドを挿入を含めるチェックボックス
 - ブラウザ表示環境設定パネル 194
- コンテンツウィンドウ
 - ブラウザ 149
- コンテンツビュー 144
- コントロール
 - プロジェクトのデバッグ 73 ~ 75
- コントロールタブ
 - ビューを表示 423
 - ブラウザ (RAD) 423
- コンパイラ 26
 - ファイルマッピング設定パネル 242
- コンパイラスレッドスタック
 - ビルド設定パネル 184
- コンパイラ相対パス 190
- コンパイル 28, 184, 255 ~ 276, 466
 - 選択したファイルを 258
 - ソースファイル 258
 - 1つのファイルを 258
 - プロジェクト 256
- コンパイルとリンク
 - オブジェクトコードの削除 261
 - オプション 262
 - 解説 269
 - 逆アセンブル 268
 - 高速化 262
 - コンパイラを選択 255
 - 修正日時を同期 261
 - 実行 260
 - デバッグ 260
 - バイナリの削除 261, 262
 - ファイルをコンパイル 257 ~ 258
 - ファイル拡張子を設定 256
 - プラグインコンパイラ 256
 - プリコンパイルヘッダ 262 ~ 268

- プリプロセス 268
- メイク
 - プロジェクト 259
 - リンクマップ 261
- コンパイル後にブラウザ内部情報を出力する
 - ビルドその他設定パネル 236
- コンポーネント 415
 - Button 415
 - Canvas 415
 - Checkbox 415
 - Choice 415
 - Jbutton 416
 - JCheckBox 416
 - JComboBox 416
 - JEditorPane 416
 - JLabel 416
 - JList 416
 - JMenuBar 417
 - JOptionPane 417
 - JPanel 417
 - JPasswordField 417
 - JPopupMenu 417
 - JProgressBar 417
 - JRadioButton 417
 - JScrollBar 417
 - JScrollPane 417
 - JSlider 417
 - JSplitPane 417
 - JTabbedPane 417
 - JTable 417
 - JTextArea 417
 - JTextField 417
 - JTextPane 418
 - JToggleButton 418
 - JToolBar 418
 - JTree 418
 - Label 415
 - List 415
 - MenuBar 415
 - Panel 416
 - PopupMenu 416
 - Scrollbar 416
 - ScrollPane 416
 - TextArea 416
 - TextField 416
 - 定義 366
- コンポーネントの読み込み 458
- コンポーネントエディタ 418 ~ 422
- コンポーネントカタログ 400, 476
 - インデックスビューの切替えボタン 402
 - カタログを閉じるボタン 402
 - カタログを開くボタン 401
 - 項目のプロパティを編集ボタン 402
 - コンテキストメニュー 405
 - コンポーネントの読み込みボタン 402
 - コンポーネントパレットボタン 402
 - 新規フォルダボタン 401
- コンポーネントカタログウィンドウ 400 ~ 409
 - Content ビューのボタン 403
 - カタログを作成 406
 - カタログ欄 402
 - コンポーネントリスト 404
 - コンポーネント情報バー 404
 - コンポーネント欄 402
- コンポーネントカタログツールバー 401
- コンポーネントカタログファイルオプション
 - 新規ダイアログ 406
- コンポーネントカタログボタン 399
- コンポーネント情報バー
 - クラス項目 405
 - 名前項目 405
- コンポーネントツール 399
- コンポーネントの読み込みボタン 402
- コンポーネントパレット 398 ~ 400, 476
 - カタログを開く 398
 - カタログを閉じる 399
 - カタログポップアップメニュー 398
 - コンポーネントカタログ 399
 - コンポーネントツール 399
 - ツールバー 398
- コンポーネントパレットボタン 402
- コンポーネントモデル
 - RAD 366
- コンポーネントリスト
 - コンポーネントカタログウィンドウ 404
- コンポーネント情報バー 404, 405
 - Class 項目 405
 - Comments 項目 405
 - Lock 項目 405
 - Modified 項目 405
 - Name 項目 405
 - ソート 405
 - ソート順 405
 - 列サイズを変更 405
- コンポーネント欄
 - コンポーネントカタログウィンドウ 402
 - コンポーネント情報バー 404

コンポーネント欄のビュー 403

サ行

サードパーティ COM プラグインを無効にする
プラグイン設定パネル 188

サードパーティのデバッグを利用する
ビルドその他設定パネル 237

サードパーティエディタ

Emacs 185

サポート 185

サードパーティエディタを使う

IDE その他設定パネル 185

サードパーティツール 29

再起動 468

最近指定した文字列ポップアップメニュー 128

テキスト検索 125

再コンパイル 258

最新状態に更新する 258, 259, 264, 466

再実行 459

サイズ

フォント & タブ設定パネル 197

サイズ変更

欄を 282, 290

サイズポップアップメニュー

フォント & タブ設定パネル 197

サイズ変更 395, 477

サイズ変更バー (ブラウザウィンドウ) 153

再定義した関数

ブレークポイントを設定 328

最適化

インストラクションスケジューリング 251

共通部分式の削除 249

強度の低減 250

コピーによる伝播 249

算術演算最適化 249

式の単純化 249

生存分析レジスタ割り当て 251

大域レジスタ割り当て 249

次の繰り返し 251

範囲分割最適化 250

デッドストア削除 250

ピーブホール最適化 249

ループアンローリング 250

分岐の最適化 249

ベクトル演算化 250

未使用コードの除去 249

ループの変換 250

ループ不変の移動 250

最適化目的

グローバル最適化設定パネル 248, 249

最適化レベルスライダー

グローバル最適化設定パネル 249, 248

作業ディレクトリ

ランタイム設定パネル 238

削除 393, 460

Watchpoint 331

アクセスパス設定パネル 235

カスタムキーワード 199

環境変数 238

キーバインディング 216

コマンドグループ 213

評価式を 297

ファイルとグループ 61

ブレークポイント 324

メニューアイテム 421

メニューコマンド 212

レイアウトの RAD オブジェクト 393

削除ボタン

Java アプレットウィザード 374

ソースツリー設定パネル 193

IDE コマンドのカスタマイズダイアログ 213

ソースツリー設定パネル 247

作成

RAD オブジェクト 391

RAD コンポーネントカタログ 406

RAD プロジェクト 367 ~ 368

環境変数 238

サブプロジェクト 70

条件ブレークポイント 328

ステーションナリからプロジェクトを 41

入力ファイル 24

ビルドターゲット 66

ビルドターゲットの依存関係 68

複合プロジェクト 64 ~ 70

プロジェクト 40 ~ 44

プロジェクトステーションナリ 45

メニューアイテム 419

メニューコマンド 211

メニューコマンドグループ 211

レイアウト 389

作成ボタン

RAD ウィザード 369, 396

ブラウザウィザード 162

サブクラスで表示 152

- サブプロジェクト
 - 定義 65
 - ブラウザ 176
 - プロジェクト内に作成 70
 - 開く 50
- サブプロジェクトをキャッシュする
 - ビルドその他設定パネル 236
- 差分の適用を取り消す 464
- 差分を適用する 464
- 参照ボタン
 - ビルドその他設定パネル 237
 - レジスタ詳細ウィンドウ 301
- 算術演算最適化 249
- サンプル
 - フォント & タブ設定パネル 197
- サンプルフィールド
 - フォント & タブ設定パネル 197
- サンプルプロジェクト
 - Core Tutorials フォルダ 20
- 指定子ポップアップメニュー
 - Static 169
- 式の単純化 249
- 識別
 - ブラウザデータベースのシンボル 174
- 識別アイコン
 - ブラウザ 155
- システム
 - 相対パス 235, 245
- システムの必要条件 23
- システムパス
 - アクセスパス設定パネル 233
- システムパス欄
 - アクセスパス設定パネル 233
- システムヘッダチェックボックス
 - 複数ファイル検索 122
- システムメッセージのログを残す
 - デバッグ設定パネル 253
- システム相対パス 190
- 指定語で始まるシンボルを検索 176
- 指定語を含むシンボルを検索 176
- 指定子
 - 新規データメンバーウィザード 172
 - 新規メンバー関数ウィザード 169
- 指定子ポップアップメニュー
 - Pure Virtual 169
 - 新規 Java イベントセットウィザード 437
- Mutable 172
- Static 172
- なし 169, 172
- Abstract
 - 指定子ポップアップメニュー 432
- 指定子ポップアップメニュー
 - Static 437
 - Abstract 381, 432, 437
 - Final 381, 432, 437
 - Static 432
 - なし 432, 381, 437
- 指定子欄
 - Java Bean ウィザード 381
- 指定の行へ移動 464
- 修飾子
 - Java Bean ウィザード 381
- 修飾子欄
 - 新規 Java イベントセットウィザード 437
 - 新規イベントダイアログ 441
 - 新規データメンバーウィザード 172
 - 新規メソッドダイアログ 432
 - 新規メンバー関数ウィザード 169
- 新規イベントダイアログ
 - 修飾子欄 441
- 修正
 - プロジェクト 44
 - レイアウト 390
- 終了 458
 - コードの実行 319
- 終了ボタン
 - カスタムキーワードダイアログ 199
- 終了、実行を
 - 停止の違い 319
- 縮小
 - グループ 54
- 出荷時設定ボタン 182
 - IDE コマンドのカスタマイズダイアログ 208
 - ターゲット設定パネル 227
- 出力ディレクトリ
 - ターゲット設定パネル 231
- 出力ファイル 26, 31
- 種類
 - プロジェクト 40
- 種類ポップアップメニュー 446
- 消去 62, 107
 - Expression ウィンドウ 297
 - Watchpoint 299, 332

消去ボタン

ターゲット設定パネル 231

ランタイム設定パネル 238

詳細

グローバル最適化設定パネル 248, 249

初期設定子

新規データメンバーウィザード 172

新規プロパティダイアログ 429

調べる

コマンド 210

新規 367, 383, 390, 406, 456

新規ダイアログ

空プロジェクトオプション 41

設定ボタン 42

プロジェクトタブ 41

プロジェクトステーションナリのオプション 40

新規 Expression 471

新規 Java イベントセット 434

新規 Java イベントセットウィザード

Bean のクラス 435

内部 437

アクセスポップアップメニュー 437

可視コンポーネント 438

カスタム 438

基底クラスと実装を指定 438

基底クラス欄 438

クラス 437

クラスに対応 437

クラス名 435

後方 437

指定子ポップアップメニュー 437

修飾子欄 437

新規ファイル 436

実装リスト 439

設定ボタン 436, 437

前方 437

ターゲット 439

名前と場所を指定 435

ファイル 435

ファイルを追加するターゲット欄 439

新規イベント 439, 473

新規イベントセット 473

ブラウザメニュー 433

新規イベントダイアログ 439

修飾子欄 441

追加 throw 441

キャンセルボタン 441

宣言欄 441

追加ボタン 441

名前 441

引数 441

引数に必要なパッケージ 441

新規イベントセットダイアログ 433

既存のイベントセット 434

新規カスタムイベントセット 433

新規ダイアログ

プロジェクトタブ 367

新規カスタムイベントセット

新規イベントセットダイアログ 433

コンポーネントカタログ

新規カタログボタン 401

新規カタログボタン 401

新規ウィザード

基底クラスとコンストラクタ引数に必要な名

前クラス 165

新規クラス 473

ブラウザメニュー 161

新規クラスウィザード 162

Virtual デストラクタチェックボックス 165

アクセスポップアップメニュー 165

インクルードファイル 166

既存ファイル選択ボタン 164

基底クラスとメソッド 164

基底クラスフィールド 165

基底クラスのために自動的に追加される

#include ファイル 166

クラス相対 163

クラスフィールド 163

クラス名 163

後方 164

コンストラクタ引数フィールド 165

コンストラクタとデストラクタを生成 165

新規ファイル 163

新規ファイル選択ボタン 164

設定ボタン 163

宣言ファイル 163

宣言ファイルフィールド 163

前方 164

ターゲット 166

追加ヘッダ #include ファイル 166

名前空間フィールド 164

ファイルを追加するターゲット 167

プロジェクト 167

プロジェクトフィールド 167

メンバー定義に別ファイルを使用するチェッ

クボックス 164

- 名前と位置を指定 162
- 新規クラスブラウザ 475
- 新規グループボタン
 - IDE コマンドのカスタマイズダイアログ 211
- 新規グループを作成 465
- 新規コマンド 40
- 新規コマンドボタン 212
- 新規ターゲットを作成 465
- 新規ダイアログ 384, 407
 - Java Bean ウィザード 367
 - Java アプリケーションウィザード 367
 - Java アプレットウィザード 367
 - オブジェクトタブ 390
 - コンポーネントカタログファイルオプション 406
 - 設定ボタン 368
 - デザインポップアップメニュー 390
 - 場所 368
 - ファイルタブ 406
 - プロジェクトタブ 383
 - プロジェクトポップアップメニュー 390
 - プロジェクト名フィールド 367
- 新規テキストファイル 264, 265, 267, 456
- 新規データメンバー 161, 473
- 新規データメンバーウィザード 171
 - Const チェックボックス 172
 - Volatile チェックボックス 172
 - 型に必要な名前空間 172
 - 型フィールド 172
 - 指定子 172
 - 修飾子欄 172
 - 初期設定子 172
 - 宣言フィールド 173
 - 追加ヘッダ #include ファイル 173
 - 定義 173
 - データメンバーの宣言 171
 - 名前フィールド 172
 - ファイル位置を指定 172
 - メンバ型のために自動的に追加される #include ファイル 173
- 新規ファイル
 - Java Bean ウィザード 381
 - 新規 Java イベントセットウィザード 436
 - 新規クラスウィザード 163
- 新規ファイル選択ボタン
 - 新規クラスウィザード 164
- 新規フォルダボタン 401
- 新規プロジェクトダイアログ 43
- 新規プロパティ 428, 473
- 新規プロパティダイアログ
 - Transient 429
 - Volatile 429
 - アクセッサ 429
 - 一覧欄 430
 - 型 430
 - 型に必要なパッケージ 428
 - キャンセルボタン 430
 - 初期設定子 429
 - 追加ボタン 430
 - データメンバーあり 429
 - 名前 429, 430
 - 型 428
 - 名前 428
- 新規ボタン
 - 新規メンバー関数ウィザード 170
- コマンド
 - 新規メソッド 473
- 新規メソッド 430, 473
 - ブラウザメニュー 161
- 新規メソッドダイアログ 430
 - Native 432
 - Synchronized 432
 - キャンセルボタン 433
 - 修飾子欄 432
 - 宣言欄 433
 - 追加 throw 431
 - 追加ボタン 433
 - 同期化する 437
 - 名前 430
 - ネイティブ 437
 - 引数 431
 - 引数に必要なパッケージ 431
 - 戻り値型 430
- 新規メンバー関数 473
- 新規メンバー関数ウィザード 167
 - Const チェックボックス 169
 - Inline チェックボックス 169
 - 既存ボタン 170
 - 指定子 169
 - 修飾子欄 169
 - 新規ボタン 170
 - 宣言フィールド 170
 - 追加ヘッダ #include ファイル 170
 - 定義 170
 - 名前 168

- 引数 169
- 引数のために必要な名前空間 169
- ファイル位置を指定 169
- メンバ関数の指定 168
- 戻り値型 168
- 戻り値型と引数のために自動的に追加される
 #include ファイル 170
- シンタックスをチェック 465
- シンタックスカラー
 - 印刷 87
 - 変更 198
- 新規ダイアログ 384
- 診断設定
 - Makefile Importer ウィザード 49
- シンボル
 - ブラウザデータベース 174
 - 補完 176
- シンボルの自動入力 176
- シンボルを自動補完 176
- シンボルウィンドウ
 - ブラウザ 159
- シンボル情報
 - 生成 280
- シンボルファイル 26
 - 定義 277
 - デバッグ 279
 - 内容 277
 - 複数のファイルを開く 290
 - 複数を開く 282
 - 保存 279
- シンボル欄 (コンテントウィンドウ) 150
- 実行 184, 259, 260, 281, 283, 313, 314, 315, 319, 354, 468
 - 終了 319
 - 停止 319
 - デバッグ上のコード 314
- 実行フィールド
 - IDE コマンドのカスタマイズダイアログ 212
- 実行毎にシンボルをキャッシュする
 - デバッグ設定パネル 253
- 実行速度の高速化ボタン
 - グローバル最適化設定パネル 249
- 実行速度の高速化
 - 最適化目的
 - グローバル最適化設定パネル 249
- 実行前のビルドポップアップメニュー 183
- ビルド設定パネル 183
- 実装リスト
 - Java Bean ウィザード 382
 - 新規 Java イベントセットウィザード 439
- 自動的にライブラリをターゲットにする
 - デバッグ設定パネル 252
- 自動オプション
 - テキストビューポップアップメニュー 304
- 自動化
 - RAD ツール 365
- 自動更新 264
- 自動ターゲットライブラリ
 - 一般設定パネル 205
- 自動変数表示 293
 - デバッグ 19
- 自動補完
 - シンボル名 176
- 自動リピートチェックボックス
 - IDE コマンドのカスタマイズダイアログ 216
- 順序を変える
 - 評価式の 297
- 準備
 - デバッグ 278 ~ 281
 - ビルドターゲット 279
- 条件
 - ステーションナリの作成 45
- 条件付きブレークポイント 327
 - ループと 328
- 条件ブレークポイント 299
 - 作成 328
 - 評価式 345
- 状態 (VCS) 448
- 状態エリア
 - ブラウザ 156
- 状態を更新 448
- 状態を更新 (VCS) 453
- 情報
 - プロジェクトを検証 71 ~ 72
- 除去
 - ツールバー項目 223
- 垂直余白
 - Java アプレットウィザード 372
- 水平余白
 - Java アプレットウィザード 372
- スキップ
 - ステートメント 317
- スクリプト

- IDE スクリプト 29
 - Perl 491
- スタッククロール欄 283, 320
- スタックの内容
 - 表示 341
- スタンドアローンアプリケーションとして実行
 - Java アプレットウィザード 371
- ステーションナリ
 - 作成 45
 - 自動設定 43
 - 条件 45
 - フォルダ 45
 - 複合プロジェクトを作成 66
 - プロジェクト 40, 44 ~ 47
 - プロジェクトを作成 41
 - プロジェクトステーションナリ 44
- ステーションナリプロジェクト 27
- ステートメント
 - スキップ 317
- ステップ
 - コード実行 315
- ステップアウト 283, 313, 316, 469
 - ルーチン 316
- ステップイン 283, 313, 316, 469
 - ルーチンへ 316
- ステップオーバー 283, 313, 315, 469
- ステップボタン 270
- ステップ実行
 - 1 行ずつ実行 315
- ストリング
 - カラー 198
- すべて
 - ホストフラグポップアップメニュー 234
- すべて置換 128, 463
- すべて閉じる 457
- すべてのファイルに情報を挿入
 - Java アプレットウィザード 375
- すべての情報
 - プラグイン診断 187
- すべてのファイルに情報を挿入
 - Java アプリケーションウィザード 379
- すべてのファイルを閉じる 85
- すべてのファイルを保存 82
- すべての変数ウィンドウを閉じる 298
- すべての例外 479
- すべて保存 457

- すべてを選択 460
- スレッド
 - 表示 294
- ズームボックス (ブラウザウィンドウ) 153
- 正規表現 136
- 制限
 - キーバインディング 214
- 生成
 - シンボル情報 280
- 生成ボタン
 - RAD ウィザード 369, 375, 379, 383
- 生存分析レジスタ割り当て 251
- 静的コンストラクタ
 - デバッグ 205
- 静的ライブラリ 25
- セグメントビュー 32
- 設定
 - Makefile Importer ウィザード 49
 - Watchpoint 331
 - カレントビルドターゲット 68
 - 再定義した関数にブレークポイントを 328
 - ブレークポイント 324
- 設定パネル
 - Java デバッグ 205
 - アクセスパス 231
 - カスタムキーワード 251
 - グローバル最適化 248
 - ソースツリー 188, 242
 - ターゲット設定 229
 - ビルドその他 235
 - ファイルマッピング 239
 - プラグイン設定 186
 - ランタイム設定 237
- 設定パネルを選択 183
- 設定ボタン
 - 新規ダイアログ 42
 - 新規 Java イベントセットウィザード 436, 437
 - 新規クラスウィザード 163
 - 新規ダイアログ 368
- セットアップ
 - プロジェクトのデバッグ 73 ~ 75
- 切断 (VCS) 449
- 説明
 - Java アプリケーションウィザード 379
 - Java アプレットウィザード 375
- 宣言フィールド
 - 新規データメンバーウィザード 173

- 新規メンバー関数ウィザード 170
- 宣言欄
 - 新規イベントダイアログ 441
 - 新規メソッドダイアログ 433
- 宣言ファイル
 - 新規クラスウィザード 163
- 宣言ファイルフィールド
 - 新規クラスウィザード 163
- 選択
 - キーボード 56
 - 検索 124
 - デフォルトプロジェクト 53
 - 複数の RAD オブジェクト 392
 - マウスクリック 55
- 選択、欄での 282
- 選択したテキストを検索 123
- 選択部分
 - エディタ設定パネル 195
- 選択部分のみ印刷 87
- 選択部分を検索 462
- 選択部分を検索文字列へ 124, 462
- 選択部分を置換文字列へ 462
- 選択ボタン
 - ソースツリー設定パネル 189, 244
 - ターゲット設定パネル 231
 - デバッグ設定パネル 252
 - ランタイム設定パネル 238
- 線ボタン
 - クラス階層ウィンドウ 157
- 絶対パス
 - ソースツリー設定パネル 189
- 190, 235
 - 型ポップアップメニュー 244
 - ソースツリー設定パネル 244
- 前方
 - 新規 Java イベントセットウィザード 437
 - 新規クラスウィザード 164
- 操作
 - RAD レイアウトでオブジェクトを 392
- 相対パスを使ってプロジェクトエントリを保存
 - ターゲット設定パネル 231
- ソースコード
 - デバッグで編集 343
 - デバッグ 277
 - デバッグをアクティブに 74
 - 表示 359
 - フォントとカラー 324
- 編集とブラウズ 27
- ソースコード上で変数の値を表示する
 - 表示設定パネル 202
- ソースコード拡張ボタン 271
- ソースコード欄 271
- ソースチェックボックス
 - 複数ファイル検索 122
- ソースツリー 188, 242
 - IDE 設定ダイアログとターゲット設定ダイアログの違い 188, 243
 - 相対パス 235
- ソースツリーリスト
 - ソースツリー設定パネル 189, 243
- ソースツリー設定パネル
 - 選択ボタン 189
- 242
 - IDE 設定ダイアログ 188
 - 型 189, 244
 - 環境変数 189, 244
 - 削除ボタン 193, 247
 - 選択ボタン 244
 - 絶対パス 189, 244
 - ソースツリーリスト 189, 243
 - 追加ボタン 244, 189
 - 名前 189, 243, 244
 - 名前フィールド 189
 - 名前列 243
 - パス 189
 - パス列 243
 - 変更 246
 - 変更ボタン 192
 - レジストリキー 189, 244
- ソースファイル 24
 - コンパイル 258
 - プリコンパイル 262 ~ 268
- ソースポップアップメニュー 290, 293
- ソース欄 285
 - 拡張ボタン 285
 - ブラウザ 155
 - ブラウザウィンドウ 292
- ソート
 - RAD コンポーネント 405
 - ファイルビュー 36
- ソート順
 - RAD コンポーネント 405
- ソート順ボタン 36
- その他 479

- エディタ設定パネル 195
- その他ボタン
 - 複数ファイル検索 123
- ソフトウェア
 - 生成 26
- タ行
- ターゲット
 - IDE 21
 - 新規クラスウィザード 166
 - 設定 228
 - ビルドターゲットにファイルを割り当てる 69
 - ビルドターゲットの依存関係 68
 - ビルドターゲットの作成 66
 - ビルドターゲットの定義 64
 - ビルドターゲット設定 68
 - ビルドターゲット設定の変更 68
 - ビルドターゲット名 67
 - プラットフォームターゲットのマニュアル 20
 - 新規 Java イベントセットウィザード 439
 - 定義 15
- ターゲットクラス内の例外 479
- ターゲットタブ
 - プロジェクトウィンドウ 385
- ターゲットビュー 32
 - デザイン 385
 - プロジェクトウィンドウ 39
- ターゲットポップアップメニュー 385
- ターゲット設定ダイアログ 229
 - エディタの設定 251
 - コード生成の設定 247
 - ターゲット設定 229
 - デバッグの設定 251
 - 変更を破棄 226
 - ボタン 226
 - 設定パネル 225
 - 設定パネルを選択 228 ~ 253
- ターゲット設定パネル 229
 - 出力ディレクトリ 231
 - 消去ボタン 231
 - 選択ボタン 231
 - 相対パスを使ってプロジェクトエントリを保存 231
 - ターゲット名 230
 - ターゲット名フィールド 230
 - プリリンカ 231
 - プリリンカポップアップメニュー 231
 - ポストリンカ 231
 - ポストリンカポップアップメニュー 231
 - リンカ 230
 - リンカポップアップメニュー 230
- ターゲット名
 - ターゲット設定パネル 230
- ターゲット名フィールド
 - ターゲット設定パネル 230
- ターゲット列 35, 69
- 大域変数 297
 - 専用ウィンドウ 298
 - デバッグで表示 333
- 大域変数ウィンドウ 297, 476
- タイトル
 - Java アプレットウィザード 372
- タイムアウトフィールド
 - Java デバッグ設定パネル 206
- 高さ
 - Java アプレットウィザード 372
- タスク
 - 表示 294
- タスク欄 296
- タスクを異なるウィンドウに表示する
 - 表示設定パネル 202
- タスクを停止したときにプログラムウィンドウを選択する
 - 一般設定パネル 205
- タッチ
 - ファイル 63
- タッチするコマンド 63
- タッチ列 36, 63
- タブに空白文字を使用
 - フォント & タブ設定パネル 197
- タブインデント選択
 - フォント & タブ設定パネル 197
- タブサイズ
 - フォント & タブ設定パネル 197
- タブビュー
 - 制限 424
 - 表示 423
- タブ設定
 - フォント & タブ設定パネル 197
- 単一クラス階層ウィンドウ 158
- 単語チェックボックス 127
- ダイアログ
 - RAD ブラウザ 427 ~ 441
 - 新規イベント 439

- 新規イベントセット 433
- 新規メソッド 430
- ダイアログのボタン
 - IDE 設定ダイアログ 182
- 大域変数ウィンドウ 333
- 大域変数名
 - 自動補完 176
- 大域レジスタ割り当て 249
- 大小文字無視チェックボックス 127
- ダイナミックリンクライブラリ 25
- ダンプメモリ 309, 472
- 値
 - 変数の値を変更 338
- チェックアウト (VCS) 447, 448
- チェックアウト取り消し (VCS) 448
- チェックアウト状況列 38, 450, 451, 453
- チェックイン (VCS) 447, 448
- 置換 463
 - 複数のファイル 130 ~ 136
- 置換後逆方向検索 463
- 置換後次を検索 463
- 中止 283, 313, 315, 319, 468
- チュートリアルリソース 20
- 追加
 - アクセスパス設定パネル 234
 - キーバインディング 215
 - ツールバー項目 222
 - ドラッグ & ドロップ 58
 - ファイルをプロジェクトへ 57
 - ファイル名拡張子 240
 - プリプロセッサシンボルをプロジェクトへ 75
 - プロジェクトヘデザインを 383
 - プロジェクトへファイルを 56
 - メニューコマンド 211
- 追加 (VCS) 448
- 追加ボタン
 - カスタムキーワードダイアログ 199
 - 新規プロパティダイアログ 430
 - ソースツリー設定設定パネル 244
 - ファイルマッピング設定パネル 240
- 新規イベントダイアログ
 - 追加 throw 441
- 追加 throw
 - 新規イベントダイアログ 441
 - 新規メソッドダイアログ 431
- 追加情報
- Java アプリケーションウィザード 378
- Java アプレットウィザード 374
- 追加ヘッダ #include ファイル
 - 新規クラスウィザード 166
 - 新規データメンバーウィザード 173
 - 新規メンバー関数ウィザード 170
- 追加ボタン
 - Java アプレットウィザード 374
 - アクセスパス設定パネル 234
 - 新規イベントダイアログ 441
 - 新規メソッドダイアログ 433
 - ソースツリー設定パネル 189
- ツール
 - サードパーティ 29
- ツールバー
 - ウィンドウツールバー 221
 - カスタマイズ 219
 - 項目 220
 - 項目を除去 223
 - 項目を追加 222
 - コンポーネントカタログウィンドウ 401
 - コンポーネントパレット 398
 - 種類 220
 - 全項目を除去 223
 - デバッグ 283
 - デフォルトに戻す 223
 - 表示 / 隠す 221
 - ブラウザ 151
 - プロジェクトウィンドウのツールバー 33
 - 変更 222
 - メインツールバー 221
- ツールバー拡張ボタン
 - エディタ 101
- ツールバーサブメニュー 474, 480
 - ウィンドウツールバーをリセット 480
 - ウィンドウツールバーを消去 480
 - ウィンドウツールバーを表示 480
 - ウィンドウツールバーを閉じる 480
 - フロートツールバーをリセット 481
 - フロートツールバーを隠す 480
 - フロートツールバーを消去 481
 - フロートツールバーを表示 480
- 使う
 - デバッグ 281
 - 評価式 346
 - ブラウザメニュー 161
- 次のシンボル 176
- 次のファイル内を検索 462

- 次へ 464
- 次へボタン
 - RAD ウィザード 434, 369, 396
 - ブラウザウィザード 162
- 次を検索 124, 128, 461
- 常にログインダイアログを表示 (VCS) 446, 449
- 定義
 - サブプロジェクト 65
 - 新規データメンバーウィザード 173
 - 新規メンバー関数ウィザード 170
 - デザイン 66
 - ビルドターゲット 64
- 定義、C/C++ 用のシンボルを 265
- 定義、Pascal 用のシンボルを 267
- 定義を検索 463
 - エディタ 114
- 停止 283, 313, 314, 319, 469
 - コードの実行 319
- 停止、実行を
 - 終了との違い 319
- 定数名
 - 自動補完 176
- 低レベルモニタへ切り替える 471
- テキスト
 - ドラッグ & ドロップ 107
- テキストの色 195
- テキストの色オプション 195
- テキスト検索 124
 - 1 つのファイル 125 ~ 129
 - 概要 115
 - 検索範囲をコントロール 135
 - 検索パラメータをコントロール 127
 - 正規表現 136
 - 選択したテキストを検索 123
 - テキストを検索 125
 - 範囲をコントロール 126
 - バッチ検索 129
 - ファイルセットを削除 136
 - ファイルセットを選択 134
 - ファイルセットを保存 134
 - ファイルを選択 131
 - 複数のファイル 120, 130 ~ 136
 - 複数ファイルをアクティブにする 131
- テキスト置換
 - 1 つのファイル 125 ~ 129
 - すべて置換 128
 - 選択して置換 127
 - 発見した ~ 127
 - 複数のファイル 130 ~ 136
- テキストビューポップアップメニュー
 - 自動オプション 304
 - ビットフィールド記述オプション 304
 - レジスタ記述 304
 - レジスタ詳細ウィンドウ 304
 - レジスタ詳細オプション 304
- テキスト編集域
 - エディタ 97
- テンキーバインディングチェックボックス
 - IDE コマンドのカスタマイズダイアログ 214, 215
- テンプレートを挿入 194
- テンプレート名
 - 自動補完 176
- テンプレート関数
 - ブレークポイントを設定 328
- テンポラリ変数 358
- ディレクトリ階層内のすべての Java クラスのデバッグを行う
 - 一般設定パネル 204
- ディレクトリフィールド
 - IDE コマンドのカスタマイズダイアログ 212
- データ
 - 専用ウィンドウ 334
 - デフォルトフォーマットで表示 336
 - 変更と表示 (デバッグ) 332
 - 別の型で表示 337
- データ型
 - anon 358
 - enum 型 358
 - 構造体を表示 285
 - 表示 335
 - 複数の 340
- データ入力
 - フォーマット 338
- データフォーマット
 - 変数 338
- データベースパス (VCS) 446
- データメニュー 471 ~ 473
 - 16 進数で表示 472
 - C 文字列で表示 472
 - Expression ヘコピー 471
 - Pascal 文字列で表示 472
 - Unicode 文字列で表示 472
 - 型を表示 471

- 新規 Expression 471
- デフォルト型で表示 472
- 配列を表示 472
- 表示を変更 472
- 符号なし 10 進数で表示 472
- 浮動小数点型で表示 473
- 変数を表示 472
- メモリの表示を変更 472
- メモリを表示 472
- 文字で表示 472
- 列挙型で表示 473
- データメンバー
 - ブラウザに表示 152
 - 宣言 171
- データメンバーの宣言
 - 新規データメンバーウィザード 171
- データメンバーあり
 - 新規プロパティダイアログ 429
- データメンバー欄 (ブラウザ) 155
- データ更新頻度
 - デバッグ設定パネル 253
- データ列 35
- デザイン
 - ターゲットビュー 385
 - 定義 66
 - プロジェクトへ追加 383
 - 作成 383 ~ 385
 - 制限 384
- デザインタブ
 - プロジェクトウィンドウ 385
- デザインビュー 32, 385
 - プロジェクトウィンドウ 38
- デザインポップアップメニュー
 - 新規ダイアログ 390
- デザインをプロジェクトに追加
 - プロジェクトパネル 384
- デザイン名
 - プロジェクトパネル 384
- デッドストア削除 250
- デッドストリップ
 - コード 35
- デバッグ 26, 478
 - 1 行ずつ実行 315
 - Java デバッグ設定パネル 205
 - Java 設定パネル 207
 - 一般的な問題 353
 - ウィンドウ指定設定パネル 203
 - エラーメッセージ 360
 - 起動 311
 - 起動時の問題 353
 - 基本的なデバッグ 311 ~ 343
 - クラッシュ 354
 - コードを実行 313, 314
 - コードを停止 313
 - コンテキストメニュー 294
 - 実行時の問題 354
 - 自動変数表示 19, 293
 - ステップ実行 313
 - ソースコードを編集 343
 - ツールバー 283
 - 使う 281
 - 定義 277
 - データの変更と表示 332
 - トラブルシューティング 353 ~ 363
 - 評価式 343 ~ 353
 - 表示設定パネル 201
 - フォントの選択 324
 - ルーチンヘステップイン 316
 - ルーチンをステップアウト 316
 - 一般設定パネル 204
- デバッグの IDE 設定
 - IDE 設定ダイアログ 201
- デバッグの設定
 - ターゲット設定ダイアログ 251
- デバッグを無効にする 259, 279, 468
- デバッグを有効にする 259, 279, 354, 467
- デバッグ実行
 - 問題 354
- デバッグを終了する際にプロセスの停止を確認する
 - 一般設定パネル 205
- デバッグ設定パネル
 - Watchpoint で停止する 252, 253
 - システムメッセージのログを残す 253
 - 自動的にライブラリをターゲットにする 252
 - 選択ボタン 252
 - データ更新頻度 253
 - リロケートライブラリ、コードリソースまたはリモートデバッグフォルダの場所 252
- デバッグ 281
 - DLL を自動的に 205, 252
 - コードの修正 26
 - シンボル情報の生成 280
 - 準備 278 ~ 281
 - 静的コンストラクタ 205

- ソースコードをアクティブに 74
- デバッグ情報の生成 73 ~ 75
- デバッグ情報 74
- ビルドターゲットの準備 279
- ファイルを設定 279
- プロジェクト 28
- ランタイム設定パネル 238
- デバッグクラス
 - Java 設定パネル 207
- デバッグコマンド 354
- デバッグ作業中に編集ファイルをキャッシュする
 - 一般設定パネル 204
- デバッグ情報マーカ 74
- デバッグ中にファイル修正日時 of 不正について確認する
 - 一般設定パネル 204
- デバッグメニュー 468 ~ 471
 - C++ 例外でブレーク 471
 - Java 例外でブレーク 471
 - Watchpoint をすべて削除 471
 - Watchpoint を削除 470
 - Watchpoint を設定 470
 - Watchpoint を無効にする 471
 - Watchpoint を有効にする 470
 - 再起動 468
 - ステップアウト 469
 - ステップイン 469
 - ステップオーバー 469
 - 中止 468
 - 停止 469
 - 低レベルモニタへ切り替える 471
 - ハードリセット 469
 - ブレークポイントの設定 469
 - ブレークポイントをすべて削除 470
 - ブレークポイントを隠す 470
 - ブレークポイントを削除 469
 - ブレークポイントを表示 470
 - ブレークポイントを無効に 470
 - ブレークポイントを有効に 469
 - リセット 469
- デバッグ列、プロジェクトウィンドウ 280
- デバッグ開始時設定
 - ウィンドウ指定設定パネル 203
- デバッグ情報ファイル 26
- デバッグ情報マーカ
 - グループ 75
- デバッグ列 35, 74
- デフォルト
 - IDE のキーバインディング 481 ~ 489
 - デフォルト of ファイル名拡張子 240 ~ 241
 - デフォルトを追加
 - アクセスパス設定パネル 233
 - デフォルトターゲットを設定 468
 - デフォルトプロジェクトを設定 468
 - デフォルト型で表示 472
- 特殊機能
 - 評価式 346
- 閉じる 457
 - すべてのファイルを閉じる 85
 - ファイルを閉じる 85
 - プロジェクト 52 ~ 53
- 閉じるコマンド 406
- トラブルシューティング
 - 奇妙なデータ型 358
 - 奇妙な変数の名前 357
 - ソースコードがない 359
 - ソースファイル of 修正日時 360
 - デバッガ 353 ~ 363
 - デバッガ of 起動 354
 - デバッグ 354
 - 認識されないデータ型 358
 - バスエラー 356
 - ブレークポイント 355, 356
 - 変数 356
 - 変数の値 357
 - 変数が変わらない 356
 - 未定義 of 識別子 359
- 取り消し 459
- 取り出す (VCS) 448
- 同期
 - 更新日付 63
- 同期化する
 - 新規メソッドダイアログ 437
- 動作 欄
 - IDE コマンド of カスタマイズダイアログ 210
- 動的リンクライブラリ 205, 252
- ドキュメント
 - ビューア 20
- ドキュメントビューア 20
- ドラッグ & ドロップ
 - テキストを 107
 - ファイル & グループを削除 62
 - ファイルを追加 58
- ドラッグ & ドロップ編集サポート 196

ドラッグ&ドロップ編集 196

ナ行

内部

新規 Java イベントセットウィザード 437

なし

指定子ポップアップメニュー 169, 172, 381, 432, 437

ビットフィールド名ポップアップメニュー 302

プラグイン診断 186

ホストフラグポップアップメニュー 234

ナビゲーション

RAD ウィザード 369

コード 320

ナビゲート

ブラウザのコード 175

プロジェクトウィンドウ 33

名前

IDE コマンドのカスタマイズダイアログ 210

新規イベントダイアログ 441

新規プロパティダイアログ 429, 430, 428

新規メソッドダイアログ 430

新規メンバー関数ウィザード 168

ソースツリー設定パネル 189, 243, 244

名前フィールド

ソースツリー設定パネル 189

名前空間フィールド

新規クラスウィザード 164

名前項目

コンポーネント情報バー 405

名前と場所を指定

新規 Java イベントセットウィザード 435

名前と位置を指定

Java Bean ウィザード 379

新規クラスウィザード 162

名前フィールド

IDE コマンドのカスタマイズダイアログ 210, 211, 212

新規データメンバーウィザード 172

ファイルパネル 407

名前変更コマンド 406

名前列

ソースツリー設定パネル 243

名前を付けて保存 83, 457

Log ウィンドウ 307

入力再実行 459

入力時に括弧の対応を確認する 195

入力取り消し 459

入力ファイル

作成 24

ネイティブ

新規メソッドダイアログ 437

ハ行

ハードリセット 469

背景の色

ブラウザ表示設定パネル 195

配置 395, 477

配置ポップアップメニュー

Java アプレットウィザード 373

ハイパーテキストの表記 18

背面へ移動 395

配列

専用ウィンドウ 334

配列ウィンドウ 307

基底アドレスを設定 308

配列の表示デフォルトサイズ

表示設定パネル 203

配列を表示 472

配列ウィンドウ 307

張り付け 460

範囲分割最適化 250

汎用レジスタ 285, 287

汎用レジスタウィンドウ 304

バージョン管理システムメニュー 477

バージョン管理設定 461

バージョン管理システム (VCS) 443

IDE プラグイン

ClearCase 443

CVS 443

Perforce 443

Projector 443

PVCS 443

SourceSafe 444

VOODOO 444

インストーラ 444

ウィンドウ

VCS メッセージウィンドウ 452

プロジェクトウィンドウ 451

コマンド

VCS について 449

コメント 449

- 取得 451
- チェックアウト 448, 451
- チェックアウト取り消し 448, 451
- チェックイン 448, 451
- 追加 451
- プロジェクト 448
- ロック解除 451
- ダイアログ
 - VCS ログイン 449
- ファイルアクセス権
 - チェックアウト 450
 - ライタブル 450
 - リードオンリー 450
 - ロック 450
 - ロック解除 450
- メニュー
 - VCS ポップアップメニュー 449
 - VCS メニュー 447
- 利用 445
- コマンド
 - 取り出す 448
- バージョン管理設定 (VCS) 445
- 場所 384
 - Java アプリケーションウィザード 376, 378
 - Java フレームウィザード 397
 - ファイルパネル 407
 - プロジェクトパネル 368, 384
- 新規ダイアログ 384
- 場所
 - Java アプレットウィザード 370
- バックアップファイル 52, 83
- バッチ検索 129
- バランス 460
 - エディタ 107
- パス
 - ソースツリー設定パネル 189
- パスワード
 - VCS 設定 446
- パスワードを記憶する (VCS) 446
- パス列
 - ソースツリー設定パネル 243
- パッケージ名
 - Java Bean ウィザード 381
 - Java アプリケーションウィザード 376, 377
 - Java アプレットウィザード 370
 - Java フレームウィザード 397
- パラメータの型
 - Java アプレットウィザード 373
- パラメータの初期値
 - Java アプレットウィザード 373
- パラメータの説明
 - Java アプレットウィザード 373
- パラメータの変数
 - Java アプレットウィザード 373
- パラメータの名前
 - Java アプレットウィザード 373
- 引数
 - 新規イベントダイアログ 441
 - 新規メソッドダイアログ 431
 - 新規メンバー関数ウィザード 169
- 引数フィールド
 - IDE コマンドのカスタマイズダイアログ 212
- 引数に必要なパッケージ
 - 新規メソッドダイアログ 431
 - 新規イベントダイアログ 441
- 引数のために必要な名前空間
 - 新規メンバー関数ウィザード 169
- 左に移動 460
 - エディタ 108
- 左端をクリックして行を選択するチェックボックス
 - エディタ設定パネル 196
- 左端をクリックして行を選択 196
- 日付のキャッシング 262
- 非デバッグ対象ウィンドウを最小化
 - ウィンドウ指定設定パネル 203
- 非デバッグ対象ウィンドウをそのままに
 - ウィンドウ指定設定パネル 203
- 評価式 343 ~ 353
 - Expressions ウィンドウの 344
 - 公式のシンタックス 349
 - 作成 344
 - 制限 346
 - ソースアドレスとしての 346
 - 使う 346
 - 定義 343
 - 定数 348
 - 特殊機能 346
 - と構造体メンバ 348
 - と変数 348
 - ドラッグ 344
 - ドラッグで条件を変更 345
 - 内のポインタ 348
 - ブレークポイントウィンドウ 345
 - ブレークポイントに付ける 345

- 翻訳 344
- メモリウィンドウの 346
- 例 348, 349
- 論理式 349
- 評価式の順序を変える 297
- 評価式を削除 297
- 表記規則
 - 警告 17
 - 初心者 17
 - 書体 17
 - 注意 17
 - ハイパーテキスト 18
 - ヒント 17
 - ホスト 18
- 表示
 - Watchpoint 332
 - アセンブラコードを 286, 293
 - アセンブリ言語のコード 278
 - アドレスを指定してメモリを 340
 - 局所変数 284, 333
 - コールチェーン 283
 - 混合コード 287, 293
 - スタックの内容 341
 - スレッドとタスク 294
 - 大域変数 297, 333
 - 追加ファイル 57
 - データ (デバッグ) 332
 - データ型 335
 - データをデフォルトフォーマットで 336
 - データを複数の型で 340
 - データを別の型で 337
 - デバッグ中の変数 293
 - 評価式 297
 - ブラウザでタブビュー 423
 - ブレークポイント 326
 - プロセッサレジスタ 342
 - ポインタ型 337
 - メモリ 334
 - メンバー関数とデータメンバー 152
 - レジスタ 285, 287
 - レジスタを 305, 306
 - ローメモリ 340
- 表示設定パネル
 - C++、Object Pascal、SOM のダイナミックタイプを利用する 202
 - Watchpoint 強調色 202
- 表示パネル 201
- 標準テキストファイル形式ポップアップメニュー
 - エディタ設定パネル 196
 - 標準テキストファイル形式 196
 - 標準メソッドを生成
 - Java アプレットウィザード 371
- 表示を変更 472
- 表示設定パネル
 - ソースコード上で変数の値を表示する 202
 - タスクを異なるウィンドウに表示する 202
 - 配列の表示デフォルトサイズ 203
 - ブラウザ内の関数をメソッド名で並べ替えて表示 202
 - 変化した変数の強調色 201
 - 変数表示領域で変数の型を表示する 202
 - 変数欄ですべての局所変数を表示する 202
- 開く 50, 456
 - 関連するファイルを開く 82
 - 既存のプロジェクト 49 ~ 51
 - 既存ファイルを開く 77 ~ 82
 - ファイルメニューからファイルを開く 78
 - 古いバージョンのプロジェクトを開く 51
 - プロジェクトウィンドウからファイルを開く 79
 - ヘッダファイル 82
 - 別のホストのプロジェクトファイルを開く 50
- 開く、ブラウザでファイルを 177
- ビットフィールド記述オプション
 - テキストビューポップアップメニュー 304
- ビットフィールド名ポップアップメニュー
 - なし 302
 - レジスタ詳細ウィンドウ 302
- ビット値
 - レジスタビット値を変更 302
- ビット値フィールド
 - レジスタ詳細ウィンドウ 302
- ビット値修飾子ポップアップメニュー
 - レジスタ詳細ウィンドウ 303
- ビューア 20
- ビルド
 - プロジェクト 44
- ビルドその他設定パネル 235
 - コンパイル後にブラウザ内部情報を出力する 236
 - サードパーティのデバッグを利用する 237
 - サブプロジェクトをキャッシュする 236
 - 参照ボタン 237
 - ファイル修正日時をキャッシュする 236
 - ブラウザを利用する 236
- ビルドを中止 466

- ビルド設定パネル
 - コンパイルスレッドスタック 184
 - 実行前のビルドポップアップメニュー 183
 - ビルド前に開いているファイルを保存 184
- ビルドターゲット 27, 31
 - 設定 68
 - 設定の変更 68
 - 定義 64
 - デバッグの準備 279
 - 名前の変更 67
 - ファイルを割り当てる 69
 - 依存関係を作成 68
 - 作成 66
 - 選択 228 ~ 253
 - 定義 16
- ビルドターゲットの設定
 - 解説 225 ~ 227
- ビルド前に開いているファイルを保存
 - ビルド設定パネル 184
- ビルド状況ウィンドウ 475
- ビルド設定パネル 183
 - ビルド前に開いているファイルを保存 184
- ビルド前に開いているファイルを保存
 - ビルド設定パネル 184
- ピーブホール最適化 249
- ファイル
 - 古いバージョンのプロジェクトを開く 51
 - IDE 1.7 フォーマットの変換 51
 - Java Bean ウィザード 381
 - アクセス権
 - チェックアウト 450
 - ライタブル 450
 - リードオンリー 450
 - ロック 450
 - ロック解除 450
 - 移動 61
 - インクルードファイル 25
 - 印刷 86 ~ 87
 - 印刷オプション 86
 - インターフェースファイル 25
 - 空のプロジェクト 40
 - 既存ファイルを開く 77 ~ 81
 - 再コンパイル 258
 - 削除 61
 - 作成 77
 - 出力 31
 - 出力ファイル 26
 - シンボルファイル 26
 - ステーションナリの自動設定 43
 - 静的ライブラリ 25
 - 選択 55
 - ソースファイル 24, 25
 - ターゲット列で割り当て 69
 - タッチとアンタッチ 63
 - ダイナミックリンクライブラリ 25
 - 追加ファイルの表示 57
 - デバッグを準備 279
 - デバッグをアクティブに 74
 - デバッグ情報 26
 - デフォルトプロジェクトを選択 53
 - 閉じる 85 ~ 86
 - ドラッグ & ドロップ 58
 - ドラッグ & ドロップで削除 62
 - 入力ファイルの作成 24
 - バックアップ 52
 - 開く 112
 - ビルドターゲットに割り当てる 69
 - プロジェクトに命名 42
 - プロジェクトのコピーを保存 52
 - プロジェクトの種類 40
 - プロジェクトへ追加 56, 57
 - プロジェクトをビルド 44
 - プロジェクトを修正 44
 - プロジェクトインスペクタで割り当て 69
 - プロジェクトステーションナリ 40
 - プロジェクトファイル 25
 - プロジェクトファイルを管理 53 ~ 63
 - ヘッダファイル 25
 - 別のホストのプロジェクトファイルを開く 50
 - 保存 82 ~ 84
 - 保存した状態に戻す 87
 - ライブラリファイル 25
 - 新規 Java イベントセットウィザード 435
- ファイル / フォルダの比較とマージ 87
- ファイルのデバッグ 74
- ファイルの割り当て 69
 - ターゲット列 69
 - プロジェクトインスペクタ 69
- ファイルを自動的に保存 82
- ファイルを新規保存 83
- ファイルを追加 465
- ファイルを追加コマンド 57
- ファイルを閉じる 85
- ファイルを保存 82
- ファイル位置を指定

- 新規メンバー関数ウィザード 169
- 新規データメンバーウィザード 172
- ファイル更新日時を同期させるコマンド 63
- ファイル更新日時を同期させる 261
- ファイルコントロールポップアップメニュー 37
- ファイルセット
 - 保存 134
- ファイルセットポップアップメニュー
 - 複数ファイル検索 120
- ファイルセットを削除 134
- ファイルタイプ
 - ファイルマッピング設定パネル 239, 240
- ファイルタブ
 - 新規ダイアログ 406
- ファイルのコピー保存コマンド 52
- ファイルの最後で停止 135
 - 複数ファイル検索 121
- ファイルパスを再検索 467
- ファイルパス表示欄
 - エディタ 100
- ファイルパネル
 - 名前フィールド 407
 - 場所 407
- ファイル比較 464
- ファイルビュー 32
 - 項目をソート 36
 - コード列 34
 - ターゲット列 35
 - タッチ列 36
 - チェックアウト状況列 38
 - データ列 35
 - デバッグ列 35
 - ファイルコントロールポップアップメニュー 37
 - ファイル列 34
 - プロジェクトウィンドウ 33
 - プロジェクトチェックアウト状況アイコン 38
 - ヘッダポップアップメニュー 37
- ファイルボタン
 - ブラウザ 156
- ファイルマッピングリスト 239
 - ファイルマッピング設定パネル 239
- ファイルマッピング設定パネル 239
 - 拡張子 239, 240
 - コンパイラ 242
 - 追加ボタン 240
 - ファイルタイプ 239, 240
 - ファイルマッピングリスト 239
 - フラグ 242
 - マッピング情報欄 240
- ファイルメニュー 456 ~ 459
 - 印刷 458
 - 過去のファイルを開く 456
 - カタログを閉じる 458
 - コピーを保存 457
 - コンポーネントの読み込み 458
 - 終了 458
 - 新規 456
 - 新規テキストファイル 456
 - すべて閉じる 457
 - すべて保存 457
 - 閉じる 457
 - 名前を付けて保存 457
 - 開く 456
 - ファイルを検索して開く 456
 - ファイル名を検索して開く 456
 - 復帰 457
 - プロジェクトの書き出し 458
 - プロジェクトの読み込み 458
 - ページ設定 458
 - 保存 457
- ファイルメニューからファイルを開く 78
- ファイル欄 291
 - と大域変数 291
- ファイルリスト
 - 複数ファイルを検索 120
- ファイルを検索して開く 456
- ファイルを追加コマンド 57
- ファイルを追加するターゲット
 - 新規クラスウィザード 167
- ファイルを追加するターゲット欄
 - 新規 Java イベントセットウィザード 439
- ファイル更新日時を同期させる 467
- ファイル修正日時をキャッシュする
 - ビルドその他定パネル 236
- ファイル名を検索して開く 456
- ファイル名拡張子
 - デフォルト設定 240 ~ 241
 - 追加 240
- ファイル欄 297
 - コードナビゲーション 322
- ファイル列 34, 79
- フォーマット
 - データ表示 336

- データ入力 338
- フォーマットポップアップメニュー
 - レジスタ詳細ウィンドウ 302
- フォルダ
 - Registers 300
- フォルダを作成チェックボックス 368
- フォント
 - フォント & タブ設定パネル 197
- フォント & タブ設定パネル 196
 - オートインデント 197
 - サイズ 197
 - サイズポップアップメニュー 197
 - サンプル 197
 - サンプルフィールド 197
 - タブに空白文字を使用 197
 - タブインデント選択 197
 - タブサイズ 197
 - タブ設定 197
 - フォント 197
 - フォントポップアップメニュー 197
- フォントの選択 324
- フォントポップアップメニュー
 - フォント & タブ設定パネル 197
- フォント設定 195
 - エディタ設定パネル 195
- 不可視 Bean/Custom オプション
 - Java Bean ウィザード 382
- 複合プロジェクト
 - 作成 64 ~ 70
 - ステーションナリの作成 66
- 複数アンドゥを利用 196
- 複数クラス階層ウィンドウ 156 ~ 158
- 複数のデータ型 340
- 複数ファイル検索
 - システムヘッダチェックボックス 122
 - ソースチェックボックス 122
 - その他ボタン 123
 - ファイルセットポップアップメニュー 120
 - ファイルの最後で停止 121
 - ファイルリスト 120
 - プロジェクトヘッダ 123
- 複数ファイルを検索 120
- 符号付き 10 進数
 - データを表示 336
- 符号付き 10 進数で表示 472
- 符号なし 10 進数
 - データを表示 336
- 符号なし 10 進数で表示 472
- 復帰 457
- 復帰ボタン 183
 - IDE コマンドのカスタマイズダイアログ 209
 - ターゲット設定パネル 227
 - レジスタ詳細ウィンドウ 303
- 浮動小数点型
 - データ入力 338
 - データを表示 336
- 浮動小数点型で表示 473
- フラグ
 - ファイルマッピング設定パネル 242
- フラグポップアップメニュー
 - 起動可能 242
 - プリコンパイル 242
 - メイク不可 242
 - リソースファイル 242
- フラッシュ時間 196
- フラッシュ時間フィールド
 - エディタ設定パネル 196
- フレームのクラスを指定
 - Java フレームウィザード 396
- フレームクラスを指定
 - Java アプリケーションウィザード 377
- フロートツールバーをリセット 481
- フロートツールバーを消去 481
- フロートツールバーを表示 480
- ブラウザ
 - #include ファイルを指定 166
 - Inherited を表示チェックボックス 178
 - MFC クラスを表示 178
 - PowerPlant クラスを表示 178
 - RAD ダイアログ 427 ~ 441
 - RAD 機能 423 ~ 427
 - VCS ポップアップメニュー 156
 - アクティブにする 143
 - イベントビュー (RAD) 426
 - インターフェース 147 ~ 161
 - ウィンドウをカスタマイズ 179
 - ウィンドウを保存 179
 - オプションを表示 144
 - 階層にサブクラスを表示 157
 - 階層の基本クラス 158
 - 階層の線 157
 - 階層ビュー 146
 - 関数定義を見る 177
 - 関数のオーバーライドを検索 178

- 概要 144 ~ 147
- クラスの基底クラスとメソッドを指定 164
- クラスの名前と位置を指定 162
- クラス欄 153
- クラスをビルドターゲットへ割り当て 166
- クラス宣言ボタン 154
- クラス欄ボタン 154
- 継承を解析 178
- コードをナビゲート 148, 175
- コードを編集 177
- コンテンツウィンドウ 149
- コンテンツビュー 144
- コントロールタブ (RAD) 423
- サイズ変更バー 153
- 識別アイコン 155
- 使用 173 ~ 179
- シンボルの自動補完 176
- シンボルウィンドウ 159
- 状態エリア 156
- 宣言を見る 177
- 選択クラスの同期 153, 156
- 選択したメンバ関数の同期 155
- ソースファイルを開く 177
- ソース欄 155
- タブビューの制限 424
- 単一クラス階層 158
- ツールバー 151
- データベースのシンボル 174
- データメンバ欄 155
- ビューを表示 (RAD) 423
- ファイルボタン 156
- 複数クラス階層 156 ~ 158
- プロパティビュー (RAD) 424
- メソッドビュー (RAD) 425
- メンバー関数とデータメンバーを表示 152
- メンバ関数のファイル位置を指定 169, 172
- メンバ関数欄 154
- リストボタン 153
- ブラウザでカラーを利用するチェックボックス
 - ブラウザ表示設定パネル 194
- ブラウザを利用する
 - ビルドその他設定パネル 236
- ブラウザウィザード 162 ~ 173
 - 完了ボタン 162
 - キャンセルボタン 162
 - 作成ボタン 162
 - 次へボタン 162
 - 戻るボタン 162
- ブラウザウィンドウ 150 ~ 156, 289, 321
- コードのナビゲーション 321
- ズームボックス 153
- ソース欄 292
- ファイル欄 291
- ブレークポイントの設定 292
- プログラムウィンドウとの比較 289
- ブラウザコンテンツ 475
- ブラウザ内の関数をメソッド名で並べ替えて表示
 - 表示設定パネル 202
- ブラウザビュー 145
- 473
- ブラウザメニュー 161, 473
 - 新規イベント 473
 - 新規イベントセット 473
 - 新規クラス 161, 473
 - 新規データメンバー 161
 - 新規プロパティ 473
 - 新規メソッド 161, 473
 - 新規メンバー関数 473
 - 新規データメンバー 473
- ブラウザ表示環境設定パネル
 - コンテキストメニューにテンプレートコマンドを挿入を含めるチェックボックス 194
- ブラウザ表示設定パネル 193, 195
 - ブラウザでカラーを利用するチェックボックス 194
- ブラウズ
 - サブプロジェクト 176
 - ソースコード 27
- ブレークポイント
 - 一時的な 317, 326
 - 一時的なブレークポイントの影響 326
 - コードの最適化の影響 329
 - 再定義した関数に設定 328
 - 条件 299
 - 条件付き 327
 - 条件付き ~ とループ 328
 - 条件評価式 345
 - 条件を作成 328
 - 設定 356
 - 設定できない 355
 - 設定と削除 324
 - 定義 324
 - テンプレート関数に設定 328
 - トラブルシューティング 355
 - 表示 326
 - ブラウザウィンドウでの設定 292
 - 無効 299

- 無反応 356
- 有効 299
- ブレークポイントの設定 469
- ブレークポイントをすべて削除 326
- ブレークポイントを隠す 326, 470
- ブレークポイントを削除 326, 469
- ブレークポイントを設定 / 削除する場所 325
- ブレークポイントを表示 326, 470
- ブレークポイントを無効に 470
- ブレークポイントを無効にする 326
- ブレークポイントを有効に 469
- ブレークポイントを有効にする 326
- ブレークポイントウィンドウ 298, 326, 476
- 分岐の最適化 249
- プラグイン診断
 - すべての情報 187
 - エラーのみ 186
 - なし 186
- プラグイン診断オプション
 - プラグイン設定パネル 186
- プラグイン設定パネル 186
 - サードパーティ COM プラグインを無効にする 188
 - プラグイン診断 186
- プラットフォームターゲット 27
 - 定義 16
- プリコンパイル 184, 262, 263, 264, 265, 266, 268, 466
 - フラグポップアップメニュー 242
- プリコンパイルヘッダ
 - 作成 263
 - 自動更新 264
- プリプロセス 184, 465
- プリプロセス、コードを 268
- プリプロセッサ 268
- プリプロセッサシンボル
 - プロジェクトへ追加 75
- プリリンク
 - ターゲット設定パネル 231
- プリリンクポップアップメニュー
 - ターゲット設定パネル 231
- プレフィックスキー 217
 - クオートキー 217
 - 待機時間 219
- プログラムウィンドウ
 - 定義 282
 - ブラウザウィンドウとの比較 289
- プログラミングの概念 24 ~ 26
- プログラムウィンドウ
 - 変数欄 284
- プログラムカウンタを変更ダイアログ 318
 - ステートメントをスキップ 318
- プログラム引数
 - Java 設定パネル 207
 - ランタイム設定パネル 238
- プロジェクト 31, 235
 - 1.7 IDE を変換 51
 - IDE 1.7 フォーマットの変換 51
 - RAD ウィザード 40
 - 移動 72 ~ 73
 - インポートとエクスポート 73
 - 空のプロジェクト 40
 - 既存のプロジェクトを開く 49 ~ 51
 - 切り替え 49
 - グループの拡張と縮小 54
 - グループの名前 62
 - グループを作成 61
 - 更新 258
 - 更新日付の同期 63
 - コピーを保存 52
 - コンパイル 256
 - 削除、ファイルとグループを 61
 - 作成 40 ~ 44
 - サブプロジェクトの作成 70
 - サブプロジェクトの定義 65
 - サブプロジェクトを開く 50
 - 修正 44
 - 種類 40
 - 新規 40
 - 新規クラスウィザード 167
 - 新規プロジェクトダイアログ 43
 - 実行 260
 - 情報を検証 71 ~ 72
 - ステーションナリ 27, 40, 44 ~ 47
 - ステーションナリから作成 41
 - ステーションナリを作成 45
 - ステーションナリを選択 40
 - ステーションナリフォルダ 45
 - 設定 228
 - 選択、グループを 55
 - 選択、ファイルとグループを 55
 - 選択、ファイルを 55
 - ターゲット 15, 27
 - タッチとアンタッチ、ファイルを 63

- デザインを追加 383
- デバッグのセットアップ 73 ~ 75
- デバッグをアクティブに 74
- デバッグを制御 73 ~ 75
- デバッグを有効にする 259
- デバッグ 28, 260
- デフォルトプロジェクトを選択 53
- 閉じる 52 ~ 53
- ドラッグ & ドロップで追加 58
- ナビゲーション 33
- バックアップ 52
- ビルド 44, 256
- ビルドターゲット 16
- ファイルとグループの削除 61
- ファイルとグループを移動 61
- ファイルのタッチとアンタッチ 63
- ファイルを追加 56, 57
- ファイル管理 53 ~ 63
- 複合プロジェクト 64 ~ 70
- 複合プロジェクトのステーションナリ 66
- 古いバージョンを開く 51
- プラットフォームターゲット 16
- プリプロセッサシンボルを追加 75
- プロジェクトステーションナリ 44
- プロジェクトファイル 25
- 別のホストのプロジェクトファイルを開く 50
- 保存 52
- 保存される項目 52
- メイク 259
- メイクファイルを変換 47 ~ 49
- 命名 42
- リンク順を設定 258
- 設定 228 ~ 253
- 定義 15
- プロジェクトタブ
 - 新規ダイアログ 367
- プロジェクトフィールド
 - 新規クラスウィザード 167
- プロジェクトメニュー
 - ファイル更新日時を同期させる 63
- プロジェクト (VCS) 448
- プロジェクトでファイルを共有 360
- プロジェクトと保存される項目 52
- プロジェクトの使い方 31
- プロジェクトの書き出し 458
- プロジェクトの書き出しコマンド 73
- プロジェクトの読み込み 458
- プロジェクトの読み込みコマンド 73
- プロジェクトを更新 258
- プロジェクトインスペクタ 69, 475
- プロジェクトウィンドウ 451
 - オーバーレイビュー 32
 - 解説 32 ~ 39
 - コード列 34
 - セグメントビュー 32
 - ターゲットタブ 385
 - ターゲットビュー 32, 39
 - ターゲット列 35
 - タッチ列 36
 - チェックアウト状況列 38
 - ツールバー 33
 - データ列 35
 - デザインタブ 385
 - デザインビュー 32, 38
 - デバッグ列 35
 - ナビゲート 33
 - ファイルコントロールポップアップメニュー 37
 - ファイルビュー 32, 33
 - ファイルビューで項目をソート 36
 - ファイル列 34, 79
 - プロジェクトチェックアウト状況アイコン 38
 - ヘッダポップアップメニュー 37
 - リンク順ビュー 32, 39
- プロジェクトウィンドウからファイルを開く 79
- プロジェクトウィンドウのデバッグ列 280
- プロジェクトウィンドウを変更しない
 - ウィンドウ指定設定パネル 203
- プロジェクトステーションナリ 40, 45
 - 作成 45
 - 自動設定 43
 - 条件 45
 - フォルダ 45
- プロジェクトステーションナリのオプション
 - 新規ダイアログ 40
- プロジェクトタブ
 - 新規ダイアログ 41, 383
- プロジェクトチェックアウト状況アイコン 38
- プロジェクトパネル
 - デザインをプロジェクトに追加 384
 - デザイン名 384
 - 場所 368
 - プロジェクト名フィールド 367
- プロジェクトファイル 25
- プロジェクトヘッダチェックボックス
 - 複数ファイル検索 123

- プロジェクトポップアップメニュー
 - 新規ダイアログ 390
- プロジェクトメニュー
 - 逆アセンブル 466
- プロジェクトメニュー 464 ~ 468
 - ウィンドウを追加 465
 - オブジェクトを削除 466
 - オブジェクトコードの削除と圧縮 467
 - コンパイル 466
 - 最新状態に更新する 466
 - 新規グループを作成 465
 - 新規ターゲットを作成 465
 - シンタックスをチェック 465
 - 実行 468
 - デバッグを無効にする 468
 - デバッグを有効にする 467
 - デフォルトターゲットを設定 468
 - デフォルトプロジェクトを設定 468
 - ビルドを中止 466
 - ファイルを追加 465
 - ファイルパスを再検索 467
 - ファイル更新日時を同期させる 467
 - プリコンパイル 466
 - プリプロセス 465
 - プロジェクト内のパスをリセット 467
 - メイク 466
- プロジェクト情報欄 270
- プロジェクト切り替えリストサブメニュー 49
- プロジェクト内のパスをリセット 467
- プロジェクト名フィールド
 - プロジェクトパネル 367
- プロセスを表示 294
- プロセスウィンドウ 294, 475
 - タスク欄 296
 - プロセス欄 296
- プロセスウィンドウのツールバー 295
- プロセス欄 296
- プロセッサレジスタ
 - 表示 342
- プロトコルポップアップメニュー
 - Java デバッグ設定パネル 206
- プロパティ 396, 477
- プロパティコマンド 406
- プロパティタブ
 - オブジェクトインスペクタ 411
- プロパティビュー
 - ブラウザ (RAD) 424
- ヘッダファイル 25
 - プリコンパイル 262, 268
 - 開く 82
- ヘッダポップアップメニュー 37, 63, 79
 - エディタ 97
- ヘルプメニュー 478 ~ 479
 - About Metrowerks 479
 - CodeWarrior ヘルプ 478
 - How To 478
 - IDE 478
 - MSL C++ リファレンス 479
 - MSL C リファレンス 479
 - エラー・リファレンス 479
 - その他 479
 - デバッグ 478
 - 用語集 478
- 変化した変数の強調色
 - 表示設定パネル 201
- 変換
 - IDE 1.7 フォーマット 51
- 変更
 - カレントビルドターゲット 68
 - 環境変数 238
 - キーバインディング 214
 - 既存のコマンド 210
 - シンタックスカラー 198
 - ソースツリー設定パネル 246
 - データ (デバッグ) 332
 - ビルドターゲット設定 68
 - ビルドターゲット名 67
 - フォントとカラー 324
 - 変数の値 338
 - メニューアイテム 420
 - メモリ変更の危険性 311
 - メモリを 310
 - レジスタを 305, 306
- 変更ボタン
 - ソースツリー設定パネル 192
 - アクセスパス設定パネル 235
- 変更を破棄
 - IDE コマンドのカスタマイズダイアログ 208
 - IDE 設定ダイアログ 182
 - ターゲット設定ダイアログ 226
- 編集
 - ソースコード 27
 - デバッグでソースコードを 343
 - ブラウザで 177
- 編集の取り消し

- エディタ 108
- 編集のやり直し
 - エディタ 108
- 編集ボタン
 - カスタムキーワードダイアログ 199
 - カスタムキーワード設定パネル 251
- 編集メニュー 459 ~ 461
 - Target の設定 461
 - 環境設定 461
 - コピー 460
 - コマンドとキーバインディング 461
 - 再実行 459
 - 削除 460
 - すべてを選択 460
 - 取り消し 459
 - 入力再実行 459
 - 入力取り消し 459
 - 張り付け 460
 - バージョン管理設定 461
 - バランス 460
 - 左に移動 460
 - 右に移動 460
 - 切り取り 459
- 変数
 - 環境変数を削除 238
 - 環境変数を作成 238
 - 環境変数を変更 238
 - 奇妙な名前 357, 358
 - 局所 333
 - 自動 202, 284
 - 自動的にウィンドウを閉じる 307
 - すべて 202, 284
 - 専用ウィンドウ 334
 - 大域 297, 333
 - 値を変更 338
 - テンポラリ 358
 - データフォーマット 338
 - トラブルシューティング 356
- 変数ウィンドウ 298, 307
- 変数表示領域で変数の型を表示する
 - 表示設定パネル 202
- 変数表示域内で変数の型を表示する 336
- 変数欄 284, 333
 - 変数
 - 自動 284
 - すべて 284
- 変数欄ですべての局所変数を表示する
 - 表示設定パネル 202

- 変数を拡張 284, 308, 333
- 変数を表示 298, 472
- ベクトル演算化 250
- ページ設定 458
- ペースト 107
- ペーストコマンド 406
- 補完
 - シンボル 176
- ホスト
 - 定義 15
- ホストフラグ
 - アクセスパス設定パネル 234
- ホストフラグポップアップメニュー
 - すべて 234
 - なし 234
- 保存 457
 - コピーを保存 83
 - すべてのファイルを保存 82
 - 名前を付けて保存 83
 - ファイル 52
 - ファイルを保存 82
 - ファイルを新規保存 83
 - ファイルを自動的に保存 82
 - プロジェクト 52
 - プロジェクトのコピー 52
- 保存ボタン 182, 183
 - IDE コマンドのカスタマイズダイアログ 208, 209
- 保存項目
 - エディタ設定パネル 195
- 保存しないボタン 182
 - IDE コマンドのカスタマイズダイアログ 208
 - ターゲット設定
 - 警告ダイアログ 227
- 保存ボタン
 - ターゲット設定
 - 警告ダイアログ 227
 - ターゲット設定パネル 227
- ボタン
 - IDE コマンドのカスタマイズダイアログ 208
 - List View 403
 - Live View 403
 - OK 235, 245
 - インデックスビューの切替え 402
 - 書き出し 217
 - カタログを開く 398
 - カタログを閉じる 399, 402

- カタログを開く 401
- キャンセル 182, 208, 227, 235, 245
- 項目のプロパティを編集 402
- コンポーネントカタログ 399
- コンポーネントの読み込み 402
- コンポーネントパレット 402
- 削除 213
- 出荷時設定 182, 208, 227
- 新規カタログ 401
- 新規グループ 211
- 新規コマンド 212
- 新規フォルダ 401
- ソート順 36
- ターゲット設定ダイアログ 226
- 追加 234
- デフォルトを追加 233
- 復帰 183, 209, 227
- 変更 235
- 保存 182, 183, 208, 209, 227
- 保存しない 182, 208, 227
- 読み込み 217
- ポイント型
 - 表示 337
- ポート ID フィールド
 - Java デバッグ設定パネル 206
- ポストリンク
 - ターゲット設定パネル 231
- ポストリンクポップアップメニュー
 - ターゲット設定パネル 231
- ポップアップメニューエディタ 421
- マ行
- マーカ, デバッグ情報 75
- マーカポップアップメニュー
 - エディタ 98
- 前のシンボル 176
- 前のファイル内を検索 462
- 前を検索 124, 462
 - テキスト検索 126
- マクロ名
 - 自動補完 176
- マッピング情報欄
 - ファイルマッピング設定パネル 240
- マニュアル
 - QuickStart とチュートリアル 20
 - プラットフォームターゲット 20
- 右に移動 460

- エディタ 108
- 未使用コードの除去 249
- 未定義の識別子 359
- 無効
 - ブレークポイント 299
- メイク 184, 259, 264, 466
- メイクファイル
 - 新規プロジェクトを作成 47
 - プロジェクトへ変換 47 ~ 49
- メイクファイルの場所
 - Makefile Importer ウィザード 48
- メイク不可
 - フラグポップアップメニュー 242
- 命名
 - プロジェクト 42
- メインツールバーをリセット 224
- メインツールバーを隠す 221
- メインツールバーを消去 223
- メインツールバーを表示 221
- メソッド
 - アルファベット順 C++/Object Pascal メソッド 291
- メソッド (C++)
 - アルファベット順 202
- メソッドビュー
 - ブラウザ (RAD) 425
- メッセージウィンドウ
 - エラーと警告メッセージ 271
 - 使用 271
- メッセージリスト欄 271
- メッセージウィンドウ
 - 修正、コンパイラエラーを 273
 - メッセージを 1 つずつ見る 272
- メニュー
 - アイテムを削除 421
 - メニューエディタで作成 419
 - メニューエディタで変更 420
- メニューに表示
 - IDE コマンドのカスタマイズダイアログ 210
- メニューに表示チェックボックス
 - IDE コマンドのカスタマイズダイアログ 211
- メニューエディタ 418
 - メニューアイテムを作成 419
 - メニューアイテムを削除 421
 - メニューアイテムを変更 420
- メニューコマンド

- カスタマイズ 211, 212
 - グループを作成 211
 - 作成 211
- メニューに表示チェックボックス
 - IDE コマンドのカスタマイズダイアログ 212
- メニューリファレンス
 - IDE 455
- メモリ
 - ローメモリを表示 340
- メモリの表示を変更 340, 472
- メモリを表示 340, 472
 - メモリウィンドウ 309
- メモリを表示コマンド 340
- メモリウィンドウ 309
 - アドレスを変更 310
 - 内容を変更 310
 - メモリの表示を変更 309
 - メモリを表示 309
 - メモリダンプ 334
 - メモリ変更の危険性 311
- メモリダンプ 309, 334, 472
- メモリダンプを表示 334
- メモリの表示を変更
 - 配列ウィンドウ 308
 - メモリウィンドウ 309
- メモリを変更 310
- メンバー定義に別ファイルを使用するチェックボックス
 - 新規クラスウィザード 164
- メンバー関数
 - ブラウザに表示 152
- メンバ関数の指定
 - 新規メンバー関数ウィザード 168
- メンバ型のためのに自動的に追加される #include ファイル
 - 新規データメンバーウィザード 173
- メンバ関数
 - ファイル位置を指定 169, 172
- メンバ関数欄
 - ブラウザ 154
- 文字
 - データを表示 336
- 文字定数
 - データ入力 338
- 文字で表示 472
- 戻り値型

- 新規メソッドダイアログ 430
- 新規メンバー関数ウィザード 168
- 戻り値型と引数のために自動的に追加される #include ファイル
 - 新規メンバー関数ウィザード 170
- 戻る 464
- 戻るボタン
 - RAD ウィザード 434, 369, 396
 - ブラウザウィザード 162

ヤ行

- 矢印キー 33
- 有効
 - デバッグ 74
 - ブレークポイント 299
- ユーザで表示 152
- ユーザパスボタン
 - アクセスパス設定パネル 233
- ユーザパス欄
 - アクセスパス設定パネル 233
- ユーザ名
 - VCS 設定 446
- 用語集 478
- 横幅
 - Java アプレットウィザード 372
- 読み込みボタン
 - IDE コマンドのカスタマイズダイアログ 217
 - レジスタ詳細ウィンドウ 303

ラ行

- ファイル
 - 静的ライブラリ 25
- ライブラリ
 - ダイナミックリンクライブラリ 25
- ライブラリ / コードリソース用ホストアプリケーションフィールド
 - ランタイム設定パネル 237
- ライブラリファイル 25
- 欄
 - エディタウィンドウ 102 ~ 104
 - 関数 291
 - 項目を選択 290
 - 選択 282
- ランタイムサポートコード内にはステップインしない
 - 一般設定パネル 205

- ランタイム設定パネル
 - 消去ボタン 238
- 237
 - 一般設定 238
 - 環境設定 238
 - 作業ディレクトリ 238
 - 選択ボタン 238
 - プログラム引数 238
 - ライブラリ / コードリソース用ホストアプリ
セッションフィールド 237
- 欄のサイズ変更バー 271
- 欄分割コントロール
 - エディタ 100
- 欄をサイズ変更 282, 290
- リカーシブ検索
 - アクセスパス 232
- リストボタン
 - ブラウザの 153
- リセット 469
- リソース
 - QuickStart とチュートリアル 20
- リソースファイル 25
 - フラグポップアップメニュー 242
- リファレンス情報
 - DE メニュー 455
 - IDE 455
- リモート IP フィールド
 - Java デバッグ設定パネル 206
- リモートデバッグチェックボックス
 - Java デバッグ設定パネル 206
- リロケートライブラリ、コードリソースまたはリ
モートデバッグフォルダの場所
 - デバッグ設定パネル 252
- リンカ 26
 - ターゲット設定パネル 230
- リンカポップアップメニュー
 - ターゲット設定パネル 230
- リンク 28, 255 ~ 276
- リンク順の設定 258
- リンク順ビュー 32
 - プロジェクトウィンドウ 39
- ルーチン
 - ステップアウト 316
 - ステップイン 316
 - ローカルな 340
- ルートバス 188, 242
- ループの変換 250
- ループアンローリング 250
- ループと条件付きブレークポイント 328
- ループ不変の移動 250
- 例
 - 評価式 348
- レイアウトエディタ
 - カスタマイズ 396
 - グリッドに固定 395
 - グリッド表示 395
 - グループ化 395
 - グループ化解除 396
 - サイズ変更 395
 - 背面へ移動 395
 - プロパティ 396
- レイアウトエディタ
 - 配置 395
- レイアウト
 - Live ビュー 403
 - RAD オブジェクトを操作 392
 - オブジェクトを作成 391
 - オブジェクトを削除 393
 - 作成 389
 - 修正 390
 - 作成 386 ~ 388
 - 制限 389
- レイアウトウィザード 396 ~ 397
 - Java フレームウィザード 390
 - 完了ボタン 396
 - キャンセルボタン 396
 - 次へボタン 396
 - 戻るボタン 396
- レイアウトエディタ 366, 389, 389 ~ 396
 - コンテキストメニュー 395
- レイアウトメニュー 477
 - カスタマイズ 477
 - グリッドに固定 477
 - グリッドを表示 477
 - グループ化 477
 - グループ化解除 477
 - サイズ変更 477
 - 配置 477
 - プロパティ 477
- 例外なし 479
- レジスタ
 - 値を変更 305, 306
 - で指されたメモリを表示 340
 - 表示 285, 287, 305, 306

- ビット値を変更 302
- レジスタ詳細ウィンドウ 300
- 値を変更 304
- 表示 304
- レジスタウィンドウ 285, 287, 304, 476
 - FPU レジスタ 305
 - General Registers 304
 - コマンド 476
- レジスタカラーリング 357
- レジスタ記述
 - テキストビューポップアップメニュー 304
- レジスタ詳細ウィンドウ 300
 - XML ファイル 300
 - 値をリセットボタン 303
 - アドレスフィールド 301
 - 書き込みボタン 303
 - 記述情報欄 303
 - 記述ファイルフィールド 301
 - 参照ボタン 301
 - 自動テキストビューオプション 304
 - テキストビューポップアップメニュー 304
 - ビット値を変更 302
 - ビットフィールド記述 テキストビューオプション 304
 - ビットフィールド値 302
 - ビットフィールド名ポップアップメニュー 302
 - ビット値修飾子ポップアップメニュー 303
 - フォーマットポップアップメニュー 302
 - 復帰ボタン 303
 - 読み込みボタン 303
 - レジスタ記述テキストビューオプション 304
 - レジスタ詳細テキストビューオプション 304
 - レジスタ値ビュー 302
- レジスタ詳細オプション
 - テキストビューポップアップメニュー 304
- レジスタ値ビュー
 - レジスタ詳細ウィンドウ 302
- レジストリキー
 - 型ポップアップメニュー 244
 - ソースツリー設定パネル 189
- 列挙型
 - 自動補完 176
- 列挙型 (enum)
 - データを表示 336
- 列挙型で表示 473
- ローカルパス (VCS) 446
- ローメモリ

- 表示 340
- ロックコマンド 406
- ロック解除コマンド 406
- 論理評価式 349

ワ行

- ワイルドカード検索 136
- 割り当て
 - クオートキー 218
 - ビルドターゲットにファイルを 69
- 再起動
 - コードを実行 320
- 取り出す (VCS) 447

CodeWarrior IDE User Guide

Credits

writing lead:	Derek Saldaña
other writers:	Caresse Bennett, David Blache, Alisa Dean, Chris Magnuson, Roopa Malavally, Marc Paquette, Jim Trudeau, L. Frank Turovich
engineering:	Lubo Antonov, Kevin Bell, Greg Bolsinga, Kenny Drycksback, David Dubrow, Matt Henderson, Michael Marks, Glenn Meter, Mathavi Paramasivam, Dan Podwall, Max Requenes, Rob Vaterlaus
quality assurance:	Joe Hayden, Patrick Sullivan, Isaac Wankerl, Josef Wankerl
frontline warriors:	Richard Atwell, John Cortell, Ron Liechty, Todd McDaniel, Jim Trudeau, Roger Wong, Ben Kenobi, CodeWarrior users everywhere
translation:	Naomi Owashi
proofreader:	Eiko Kambara, Chizu Kanbara

CodeWarrior 文書のガイド

CodeWarrior の文書はツールと同様にモジュールのように構成されています。ツール、言語、ライブラリ、ターゲットごとにマニュアルがあります。各 CodeWarrior 製品によって含まれるマニュアルが異なります。この表に記載されていないマニュアルが含まれることもあります。

コアマニュアル	
IDE User Guide	CodeWarrior IDE と CodeWarrior デバッガの使い方
言語 / コンパイラのマニュアル	
C Compilers Reference	C/C++ フロントエンドコンパイラの情報
Error Reference	コンパイラ / リンカのエラーメッセージのリストおよび解説
Assembler Reference	スタンドアローンアセンブラのシンタックス
Command-Line Tools Reference	Mac OS MPW コンパイラのコマンドラインのオプション
Plugin API Manual	CodeWarrior のプラグインコンパイラ / リンカの API
ライブラリのマニュアル	
MSL C Reference	Metrowerks ANSI 標準 C ライブラリの関数のリファレンス
MSL C++ Reference	Metrowerks ANSI 標準 C++ ライブラリの関数のリファレンス
MFC Reference	Win32 用の Microsoft Foundation Classes リファレンス
Win32 SDK Reference	Win32 API の Microsoft リファレンス
The PowerPlant Book	Mac OS 用アプリケーションフレームワークのガイド
PowerPlant Advanced Topics	PowerPlant での Mac OS プログラミングの高度なテクニック
ターゲットマニュアル	
Targeting Java VM	Java 仮想マシンプログラミングでの CodeWarrior の使い方
Targeting Mac OS	Mac OS プログラミングでの CodeWarrior の使い方
Targeting MIPS	MIPS 組み込みプロセッサプログラミングでの CodeWarrior の使い方
Targeting NEC V800/V850 series	NEC V810/830 プロセッサプログラミングでの CodeWarrior の使い方
Targeting Net Yaroze	Net Yaroze プログラミングでの CodeWarrior の使い方
Targeting PlayStation	PlayStation プログラミングでの CodeWarrior の使い方
Targeting PlayStation2	PlayStation2 プログラミングでの CodeWarrior の使い方
Targeting PowerPC	PPC 組み込みプロセッサプログラミングでの CodeWarrior の使い方

印の付いているマニュアルは日本語訳が用意されています。