

IAR Embedded Workbench®

IDE プロジェクト管理およびビルドガイド

Advanced RISC Machines Ltds

ARM コア



UIDEARM-2-J

 **IAR**
SYSTEMS

版權事項

Copyright © 1999–2011 IAR Systems AB.

IAR Systems AB が事前に書面で同意した場合を除き、このドキュメントを複製することはできません。このドキュメントに記載するソフトウェアは、正当な権限の範囲内でインストール、使用、およびコピーすることができます。

免責事項

このドキュメントの内容は、予告なく変更されることがあります。また、IAR Systems 社では、このドキュメントの内容に関して一切責任を負いません。記載内容には万全を期していますが、万一、誤りや不備がある場合でも IAR Systems 社はその責任を負いません。

IAR Systems 社、その従業員、その下請企業、またはこのドキュメントの作成者は、特殊な状況で、直接的、間接的、または結果的に発生した損害、損失、費用、課金、権利、請求、逸失利益、料金、またはその他の経費に対して一切責任を負いません。

商標

IAR Systems、IAR Embedded Workbench、C-SPY、visualSTATE、From Idea to Target、IAR KickStart Kit、IAR PowerPac、IAR YellowSuite、IAR Advanced Development Kit、IAR、および IAR Systems のロゴタイプは、IAR Systems AB が所有権を有する商標または登録商標です。J-Link は IAR Systems AB がライセンスを受けた商標です。

Microsoft および Windows は、Microsoft Corporation の登録商標です。

ARM および Thumb は、Advanced RISC Machines Ltd. の登録商標です。EmbeddedICE は、Advanced RISC Machines Ltd. の商標です。OCDemon は Macraigor Systems LLC の商標です。μC/OS-II は Micrium, Inc. の商標です。CMX-RTX は CMX Systems, Inc. の商標です。ThreadX は Express Logic の商標です。RTXC は、Quadros Systems の商標です。Fusion は、Unicoi Systems の商標です。

Adobe および Acrobat Reader は、Adobe Systems Incorporated の登録商標です。

その他のすべての製品名は、その所有者の商標または登録商標です。

改版情報

第 2 版 : 2011 年 4 月

部品番号 : UIDEARM-2-J

本ガイドは、Advanced RISC Machines Ltd の ARM コアファミリ用 IAR Embedded Workbench® のバージョン 6.2x に適用する。『ARM® 用 C-SPY® デバッグガイド』と併せて、『ARM® 用 IDE プロジェクト管理およびビルドガイド』は『ARM® 用 IAR Embedded Workbench IDE ユーザガイド』を置き換えます。

内部参照 : M10、Too6.3、ISUD。

目次（章）

表	9
図	11
はじめに	15
パート 1. プロジェクト管理とビルド	21
開発環境	23
プロジェクト管理	29
ビルド	61
編集	69
パート 2. リファレンス情報	81
インストールファイル	83
IAR Embedded Workbench IDE リファレンス	89
一般オプション	159
コンパイラオプション	167
アセンブラオプション	183
出力コンバータオプション	191
カスタムビルドオプション	193
ビルドアクションオプション	195
リンカのオプション	197
ライブラリビルダオプション	209
用語集	211
索引	227

目次

表	9
図	11
はじめに	15
本ガイドの対象者	15
このガイドの使用方法	15
このガイドの概要	16
その他のドキュメント	17
ユーザガイドおよびリファレンスガイド	17
オンラインヘルプシステムを参照	18
Web サイト	18
表記規則	18
表記規則	19
命名規約	19
 パート I. プロジェクト管理とビルド	21
開発環境	23
IAR Embedded Workbench IDE 概要	23
ツールチェーン	23
連続したワークフロー	23
拡張可能なモジュール化構造の環境	23
ウィンドウ管理	23
IDE の実行	24
終了	25
環境のカスタマイズ	25
画面上のウィンドウの編成	25
IDE のカスタマイズ	26
外部ツールの呼出し	27

プロジェクト管理	29
プロジェクト管理の概要	29
プロジェクト管理の手順	35
プロジェクト管理のリファレンス情報	41
ビルド	61
プロジェクトのビルド	61
オプションの設定	61
プロジェクトのビルド	63
バッチによる複数構成のビルド	64
ビルド前およびビルド後のアクションの使用	64
ビルド中に検出されたエラーの修正	65
コマンドラインからのビルド	66
ツールチェーンの拡張	66
ツールチェーンに追加可能なツール	67
外部ツールの追加	67
編集	69
IAR Embedded Workbench エディタの使用	69
ファイルの編集	69
コードテンプレートの使用と追加	74
ファイル間のナビゲート	76
検索	77
エディタ環境のカスタマイズ	78
外部エディタの連携	78
パート 2. リファレンス情報	81
インストールファイル	83
ディレクトリ構成	83
ファイルタイプ	85
デフォルトでないファイル名拡張子のファイル	88

IAR Embedded Workbench IDE リファレンス	89
ウィンドウ	89
メニュー	103
一般オプション	159
一般オプションの説明	159
コンパイラオプション	167
コンパイラオプションの説明	167
アセンブラオプション	183
アセンブラオプションの概要	183
出力コンバータオプション	191
出力コンバータオプションの説明	191
カスタムビルドオプション	193
カスタムビルドオプションの説明	193
ビルドアクションオプション	195
ビルドアクションのオプションの説明	195
リンカのオプション	197
リンカオプションの説明	197
ライブラリビルダオプション	209
ライブラリビルダオプションの説明	209
用語集	211
索引	227

表

1: このガイドの表記規則	19
2: このガイドで使用されている命名規約	19
3: iarbuild.exe コマンドラインオプション	66
4: ARM ディレクトリ	83
5: common ディレクトリ	85
6: ファイルタイプ	85
7: エディタで挿入ポイントを移動するキーボードコマンド	98
8: エディタでのスクロール用キーボードコマンド	98
9: エディタでのテキスト選択用キーボードコマンド	99
10: 引数変数	124



1: [IAR Embedded Workbench IDE] ウィンドウ	24
2: [ツールの設定] ダイアログボックス	27
3: カスタマイズした [ツール] メニュー	28
4: ワークスペースとプロジェクトの例	32
5: [ワークスペース] ウィンドウでプロジェクトを表示	37
6: [ワークスペース] ウィンドウの概要	38
7: [ワークスペース] ウィンドウ	42
8: [ワークスペース] ウィンドウのコンテキストメニュー	45
9: [新規プロジェクトの作成] ダイアログボックス	47
10: [プロジェクトの構成] ダイアログボックス	48
11: [新規プロジェクトの作成] ダイアログボックス	49
12: [ソースブラウザ] ウィンドウ	50
13: [ソースブラウザ] ウィンドウのコンテキストメニュー	52
14: SCC のバージョン管理システムメニュー	53
15: [ソースコード管理プロバイダの選択] ダイアログボックス	55
16: ファイルのチェックインダイアログボックス	56
17: ファイルのチェックアウトダイアログボックス	57
18: サブバージョンのバージョン管理システムメニュー	58
19: 一般オプション	62
20: エディタウィンドウ	70
21: エディタウィンドウの括弧の対応	73
22: エディタウィンドウのステータスバー	74
23: コードテンプレートの挿入	75
24: 外部コマンドラインエディタの指定	79
25: 外部エディタの DDE 設定	80
26: [IAR Embedded Workbench IDE] ウィンドウ	90
27: IDE ツールバー	91
28: [IAR Embedded Workbench IDE] ウィンドウのステータスバー	92
29: エディタウィンドウ	92
30: エディタウィンドウタブのコンテキストメニュー	93
31: [関数に移動] ウィンドウ	94

32: エディタウィンドウのコンテキストメニュー	95
33: [ビルド] ウィンドウ (メッセージウィンドウ)	99
34: [ビルド] ウィンドウのコンテキストメニュー	100
35: [ファイルで検索] ウィンドウ (メッセージウィンドウ)	100
36: [ファイルで検索] ウィンドウのコンテキストメニュー	101
37: [ツール出力] ウィンドウ (メッセージウィンドウ)	101
38: [ツール出力] ウィンドウのコンテキストメニュー	102
39: [デバッグログ] ウィンドウ (メッセージウィンドウ)	102
40: [デバッグログ] ウィンドウのコンテキストメニュー	103
41: [ファイル] メニュー	104
42: [編集] メニュー	106
43: [検索] ダイアログボックス	110
44: [置換] ダイアログボックス	111
45: [ファイルで検索] ダイアログボックス	112
46: [インクリメンタル検索] ダイアログボックス	114
47: [テンプレート] ダイアログボックス	115
48: [表示] メニュー	116
49: [プロジェクト] メニュー	118
50: [メモリ消去] ダイアログボックス	122
51: オプションダイアログボックス	123
52: [バッチビルド] ダイアログボックス	126
53: [バッチビルドの編集] ダイアログボックス	127
54: [ツール] メニュー	128
55: [共通フォント] オプション	129
56: [キーバインディング] オプション	130
57: [言語] オプション	131
58: [エディタ] オプション	132
59: [自動インデントの設定] ダイアログボックス	134
60: [外部エディタ] のオプション	136
61: [エディタセットアップファイル] オプション	137
62: [エディタ色とフォント] オプション	138
63: [メッセージ] オプション	139
64: [このダイアログは表示しない] オプションを含む [メッセージ] ダイアログボックス	140

65: プロジェクトオプション	141
66: [ソースコード管理] オプション	143
67: [デバッガ] オプション	144
68: [スタック] オプション	145
69: [レジスタフィルタ] のオプション	147
70: [ターミナル I/O] オプション	149
71: [ツールの設定] ダイアログボックス	150
72: カスタマイズした [ツール] メニュー	150
73: [ファイル名の拡張子] ダイアログボックス	152
74: [ファイル名拡張子のオーバーライド] ダイアログボックス	153
75: [ファイル名拡張子の編集] ダイアログボックス	154
76: [ビューアの設定] ダイアログボックス	155
77: [ビューア拡張子の編集] ダイアログボックス	156
78: [ウィンドウ] メニュー	157
79: 一般ターゲットオプション	159
80: [出力] オプション	161
81: [ライブラリ構成] オプション	162
82: ライブラリオプション	165
83: 複数ファイルのコンパイル	167
84: コンパイラの言語オプション	168
85: コンパイラの言語オプション	171
86: コードオプション	172
87: 最適化オプション	173
88: コンパイラの出力オプション	175
89: コンパイラのリストファイルオプション	176
90: コンパイラのプリプロセッサオプション	177
91: コンパイラの診断オプション	179
92: コンパイラの追加オプション	181
93: アセンブラの言語オプション	183
94: マクロの引用符の選択	184
95: アセンブラの出力オプション	185
96: アセンブラリストファイルオプション	185
97: アセンブラのプリプロセッサオプション	187
98: アセンブラ診断オプション	188

99: アセンブラ用の [追加オプション]	189
100: 出力コンバータオプション	191
101: カスタムツール構成オプション	193
102: ビルドアクションオプション	195
103: リンカ設定オプション	197
104: リンカライブラリオプション	198
105: リンカ入力オプション	199
106: リンカの最適化オプション	201
107: リンカ出力オプション	202
108: リンカリストオプション	203
109: リンカ #define オプション	204
110: リンカ診断オプション	205
111: リンカのチェックサムおよびフィルオプション	206
112: リンカ追加オプション	208
113: ライブラリビルダの出力オプション	210

はじめに

『ARM 用 IDE プロジェクト管理およびビルドガイド』へようこそ。
本ガイドは、内蔵の Windows 用開発ツールを使用して ARM コア用 IAR Embedded Workbench の機能を十分に活用していただけるよう支援することを目的としています。IAR Embedded Workbench IDE は、完全な組込みアプリケーションプロジェクトの開発および管理が可能な、非常に強力な統合開発環境です。

本ガイドでは、編集プロセスやプロジェクト管理、ビルドについて説明するほか、関連のリファレンス情報も提供しています。

本ガイドの対象者

本ガイドは、IDE で利用可能なすべての機能およびツールを活用する場合に利用してください。また、以下について十分な知識があるユーザを対象としています。

- C/C++ プログラミング言語
- 組込みシステム用アプリケーションの開発
- ARM コア（チップメーカーのドキュメントを参照）
- ホストコンピュータのオペレーティングシステム

IDE に統合されている他の開発ツールの詳細は、それぞれのドキュメントを参照（17 ページの *その他のドキュメント* を参照）してください。

このガイドの使用方法

この製品を初めて使用する場合は、まずガイド「*IAR Embedded Workbench® の使用開始の手順*」で、IDE で利用可能なツールおよび機能の概要を確認することをお勧めします。IAR インフォメーションセンタにあるチュートリアルでは、使用について説明しています。IAR Embedded Workbench の使用を開始するにあたって、役に立ちます。

プロジェクト管理、ビルド、編集手順については本書で説明しています。C-SPY を使用したデバッグの方法については、『*ARM® 用 C-SPY® デバッグガイド*』で説明しています。

十分な開発経験があり、本ガイドのリファレンス情報だけが必要な場合は、「パート2. リファレンス情報」のリファレンス情報および IAR Embedded Workbench IDE の「ヘルプ」メニューから使用できるオンラインヘルプを参照してください。

最後に、IAR システムズのユーザドキュメントでわからない用語がある場合は、用語集を参照してください。

このガイドの概要

本ガイドの構成および各章の概要を以下に示します。

パート1. プロジェクト管理とビルド

アプリケーションの編集、ビルドの手順について説明します。

- 「開発環境」では、IAR Embedded Workbench 開発環境の概要を説明します。また、必要に応じて環境をカスタマイズする機能についても説明します。
- 「プロジェクト管理」では、ワークスペースを作成し、複数のオブジェクト、ビルド構成、グループ、ソースファイル、オプションを指定して、バージョンの異なるアプリケーションを管理する方法を説明します。
- 「ビルド」では、アプリケーションのビルド手順について説明します。
- 「編集」では、IAR Embedded Workbench エディタの詳細、使用方法、関連機能について説明します。また、任意の外部エディタとの連携方法についても説明します。

パート2. リファレンス情報

- インストールファイルでは、ディレクトリ構成および各ディレクトリに含まれるファイルの種類について説明します。
- 「IAR Embedded Workbench IDE リファレンス」は、グラフィカルユーザインタフェースの詳細など、開発環境の詳細なリファレンス情報を収録しています。
- 「一般オプション」は、ターゲット、出力、ライブラリ、MISRA-C オプションについて説明します。
- 「コンパイラオプション」では、言語、最適化、コード、出力、リストファイル、プリプロセッサ、診断、MISRA-C のコンパイラオプションを指定します。
- 「アセンブラオプション」では、言語、出力、リスト、プリプロセッサ、診断用のアセンブラオプションについて説明します。
- 「出力コンバータオプション」では、ELF 形式からリンカ出力ファイルの変換に使用できるオプションについて説明します。

- 「カスタムビルドオプション」では、ツールのカスタム設定用オプションについて説明します。
- 「ビルドアクションオプション」では、ビルド前とビルド後のアクション用オプションについて説明します。
- 「リンクのオプション」では、リンクを設定するオプションについて説明します。
- 「ライブラリビルダオプション」では、ライブラリをビルドするためのオプションについて説明します。

その他のドキュメント

ユーザドキュメンテーションは、ハイパーテキスト PDF 形式、およびコンテキスト依存のオンラインヘルプシステム (HTML フォーマット) があります。ドキュメンテーションには、インフォメーションセンタあるいは IAR Embedded Workbench IDE の [ヘルプ] メニューからアクセスできます。オンラインヘルプシステムは、F1 キーを押しても使用できます。

ユーザガイドおよびリファレンスガイド

IAR システムズの各開発ツールについては、一連のガイドで説明しています。知りたい情報に対応するドキュメントを以下に示します。

- IAR システムズの製品のインストールおよび登録の要件と詳細については、同梱されているクイックレファレンスのブックレットおよび『インストールとライセンス登録ガイド』をご覧ください。
- IAR Embedded Workbench および提供されるツールの利用にあたっては、『IAR Embedded Workbench® の使用開始の手順』を参照してください。
- IAR C-SPY® デバッガの使用については、『ARM® 用 C-SPY® デバッガガイド』を参照してください。
- ARM 用 IAR C/C++ コンパイラのプログラミングおよび IAR ILINK リンカを使用したリンクについては、『ARM 用 IAR C/C++ 開発ガイド』を参照してください。
- ARM 用 IAR アセンブラを使用したプログラミングについては、『ARM® IAR アセンブラリファレンスガイド』を参照してください。
- IAR DLIB ライブラリの使用については、オンラインヘルプで利用できる DLIB ライブラリリファレンス情報を参照してください。
- ARM 用 IAR Embedded Workbench の旧バージョンで開発したアプリケーションコードやプロジェクトの移植については、『ARM® 用 IAR Embedded Workbench® 移行ガイド』を参照してください。

- MISRA-C ガイドラインを使用して、安全性を最重要視したアプリケーションを開発する方法については、『*IAR Embedded Workbench® MISRA-C:2004 リファレンスガイド*』または『*IAR Embedded Workbench® MISRA-C:1998 リファレンスガイド*』を参照してください。

注：製品のインストール内容によっては、他のドキュメントも提供される場合があります。

オンラインヘルプシステムを参照

コンテキスト依存のオンラインヘルプの内容は以下のとおりです。

- IAR C-SPY® デバッガを使用したデバッグについての包括的な情報
- IDE のメニューやウィンドウ、ダイアログボックスに関するリファレンス情報
- コンパイラのリファレンス情報
- DLIB ライブラリ関数のキーワードリファレンス情報 関数のリファレンス情報を確認するには、エディタウィンドウで関数名を選択し、F1 キーを押します

WEB サイト

推奨 Web サイト：

- Advanced RISC Machines Ltd の Web サイト (www.arm.com) には、the ARM コアに関する情報とニュースが記載されています。
- IAR システムズの Web サイト (www.iarsys.co.jp) では、アプリケーションノートおよびその他の製品情報を公開しています。
- C 標準化作業グループの Web サイト、www.open-std.org/jtc1/sc22/wg14。
- C++ Standards Committee の Web サイト、www.open-std.org/jtc1/sc22/wg21。
- Embedded C++ Technical Committee の Web サイト (www.caravan.net/ec2plus) には、Embedded C++ 規格についての情報が公開されています。

表記規則

本ガイドでプログラミング言語 C と記述されている場合、特に記述がない限り C++ も含まれます。

製品インストール先のディレクトリ (arm¥doc) の記述がある場合、その場所までのフルパス（例：c:¥Program Files¥IAR Systems¥Embedded Workbench 6.n¥arm¥doc）を意味します。

表記規則

このガイドでは、次の表記規則を使用します。





スタイル	用途
コンピュータ	<ul style="list-style-type: none">• ソースコードの例、ファイルパス。• コマンドライン上のテキスト。• 2 進数、16 進数、8 進数。
パラメータ	パラメータとして使用される実際の値を表すブレースホルダ。たとえば、 <code>filename.h</code> の場合、 <code>filename</code> はファイルの名前を表します。
[オプション]	コマンドのオプション部分。
[a b c]	代替の選択肢を持つコマンドのオプション部分。
{a b c}	コマンドの必須部分に選択肢があることを示します。
太字	画面に表示されるメニュー名、メニューコマンド、ボタン、およびダイアログボックス。
斜体	<ul style="list-style-type: none">• 本ガイドや他のガイドへのクロスリファレンスを示します。• 強調。 3 点リーダーは、その前の項目を任意の回数繰り返せることを示します。
	IAR Embedded Workbench IDE 固有の内容を示します。
	コマンドラインインタフェース固有の内容を示します。
	開発やプログラミングについてのヒントを示します。
	ワーニングを示します。

表 1: このガイドの表記規則

命名規約

以下の命名規約は、このガイドに記述されている IAR システムズの製品およびツールで使用されています。

ブランド名	一般名称
ARM 用 IAR Embedded Workbench®	IAR Embedded Workbench®
ARM 用 IAR Embedded Workbench® IDE	IDE
ARM 用 IAR C-SPY® デバッガ	C-SPY、デバッガ
IAR C-SPY® シミュレータ	シミュレータ

表 2: このガイドで使用されている命名規約

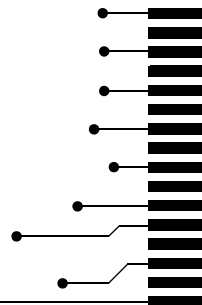
ブランド名	一般名称
ARM 用 IAR C/C++ コンパイラ	コンパイラ
ARM 用 IAR アセンブラ	アセンブラ
IAR ILINK リンカ	ILINK、リンカ
IAR DLIB ライブラリ	DLIB ライブラリ

表 2: このガイドで使用されている命名規約 (続き)

パート I. プロジェクト管理とビルド

『IDE プロジェクト管理およびビルドガイド』のこのパートは、以下の章で構成されています。

- 開発環境
- プロジェクト管理
- ビルド
- 編集





開発環境

この章では、IAR Embedded Workbench® 統合開発環境 (IDE) について説明します。また、要件に適合するように環境をカスタマイズする方法について説明します。

IAR Embedded Workbench IDE 概要

ツールチェーン

IDE は、すべての必要なツール（ツールチェーン）が統合された環境です。C/C++ コンパイラ、アセンブラ、リンカ、エディタ、作成ユーティリティ付属のプロジェクトマネージャ、IAR C-SPY® デバッガが含まれます。ソースコードのビルド専用を使用されるツールは、**ビルドツール**と呼ばれます。

ビルド済みプロジェクト環境で外部ツールとして利用したい場合は、コンパイラ、アセンブラ、リンカを、コマンドライン環境で実行することもできます。

連続したワークフロー

使用するマイクロコントローラに関わらず、ユーザインタフェースは同じで、各デバイスに対する一般およびターゲット固有のサポートが提供されています。

拡張可能なモジュール化構造の環境

IDE にはプロジェクトに必要なあらゆる機能が備わっていますが、他のツールを統合することも可能です。たとえば、IAR visualSTATE をツールチェーンに追加できます。すなわち、有限オートマトン図を IDE のプロジェクトに直接追加することができます。バージョン管理システムを使用すると、異なるバージョンのソースコードをトレースするときに便利です。IDE では、Microsoft が公開している SCC インタフェースに準拠する任意のサードパーティ製バージョン管理システムを利用できます。また、IDE ではサブバージョンの作業用コピーのファイルにアタッチできます。カスタムビルドというメカニズムを使用して、他のツールをツールチェーンに組み込むこともできます（66 ページの **ツールチェーンの拡張**を参照）。

ウィンドウ管理

ウィンドウの配置を自在かつ簡単に設定できるように、各ウィンドウは **ドッキング可能**になっています。また、ウィンドウを **タブグループ**で整理することもできます。

以下の図は、[IAR Embedded Workbench IDE] ウィンドウのさまざまなコンポーネントを示します。

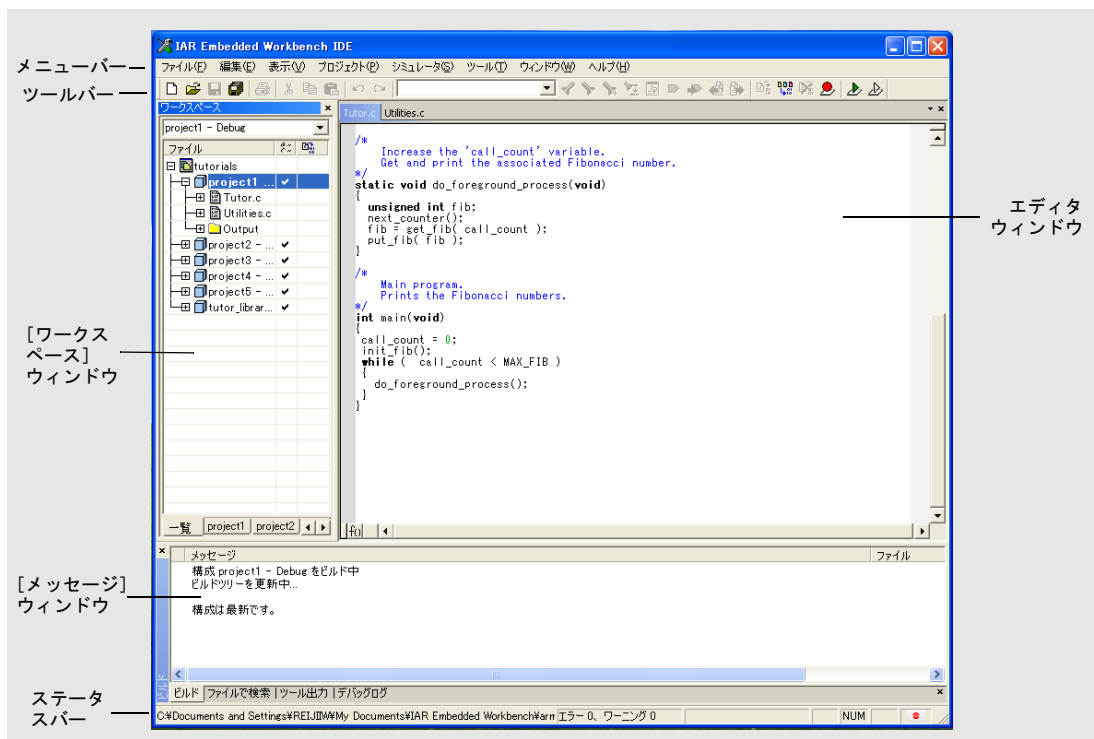


図 1: [IAR Embedded Workbench IDE] ウィンドウ

使用するツールによっては、ウィンドウの外観が異なる場合があります。

IDE の実行

Windows のタスクバーで [スタート] ボタンをクリックして、[全てのプログラム] > [IAR システムズ] > IAR Embedded Workbench for ARM > IAR Embedded Workbench を選択します。

コマンドラインまたは Windows エクスプローラからプログラムを起動するには、IAR システムズのインストール先の common¥bin ディレクトリにある IarIdePm.exe ファイルを実行します。

ワークスペースファイル名のダブルクリック

ワークスペースファイル名には、拡張子 `eww` が付いています。ワークスペースのファイル名をダブルクリックすると、IDE が起動します。複数バージョンの IAR Embedded Workbench がインストールされている場合、ワークスペースファイルは、そのファイルタイプを使用する最新バージョンの IAR Embedded Workbench によって開かれます。

終了

IDE を終了するには、[ファイル] > [終了] を選択します。終了する前に、エディタウィンドウ、プロジェクト、ワークスペースに対する変更を保存するかどうかを確認するメッセージが表示されます。

環境のカスタマイズ

IDE は、高度にカスタマイズ可能な環境です。このセクションでは、画面上のウィンドウを操作し編成する方法、IDE で実行できるカスタマイズ、外部ツールと通信するための環境設定方法について説明します。

画面上のウィンドウの編成

IDE では、ウィンドウの位置やレイアウトの調整を任意に設定できます。ウィンドウは特定の位置に **ドッキング**して、タブグループとして編成できます。また、ウィンドウを **フローティング化**することができます。フローティングウィンドウは、常に他のウィンドウよりも前に表示されます。フローティングウィンドウのサイズや位置を変更しても、現在開かれている他のウィンドウは影響を受けません。

一度保存したワークスペースを開くと、保存したときと同じウィンドウが同じサイズで同じ位置に開きます。

C-SPY 環境で実行されるプロジェクトのレイアウトはすべて個別に保存されます。ワークスペースに関する情報の他に、開いているすべてのデバッガ固有のウィンドウに関する情報も保存されます。

ドッキングウィンドウとフローティングウィンドウの使用

開くウィンドウにはそれぞれデフォルトの位置があり、それは現在開かれている他のウィンドウによって変わります。ウィンドウの位置を簡単に、そして完全に制御できるように、各ウィンドウをドッキングしたりフローティング化したりできます。

ドッキングされたウィンドウは、ユーザが決めた Embedded Workbench メインウィンドウの特定の領域にロックされます。同時に複数のウィンドウを開いた状態を維持するには、ウィンドウをタブグループとして編成します。これ

は、画面の特定の領域を、同時に開いている複数のウィンドウが使用することを意味します。この場合、ウィンドウのサイズ変更も簡単に実行できます。ドッキングされたウィンドウの1つをサイズ変更すると、ドッキングされた他のウィンドウのサイズがそれに従って変更されます。

フローティングウィンドウは、常に他のウィンドウよりも前面に表示されます。その位置とサイズは、現在開かれている他のウィンドウには影響を与えません。フローティングウィンドウは画面上の任意の位置に移動することができ、IAR Embedded Workbench IDE メインウィンドウの外部にも配置できます。

注：エディタウィンドウは常にドッキングされています。エディタウィンドウを開くと、その位置は現在開いている他のウィンドウに応じて自動的に決まります。エディタウィンドウの操作方法の詳細については、69 ページの *IAR Embedded Workbench エディタの使用* を参照してください。

ウィンドウの編成

ウィンドウを *個別* のウィンドウとして配置するには、開いている別のウィンドウの横にウィンドウをドラッグします。

ウィンドウを開いている別のウィンドウと同じタブグループに配置するには、ドラッグして、他のウィンドウの中央にドロップします。

ウィンドウをフローティング化するには、ウィンドウのタイトルバーをダブルクリックします。



IAR Embedded Workbench IDE メインウィンドウの下端にあるステータスバーには、ウィンドウのサイズを変更するためのヘルプが用意されています。

IDE のカスタマイズ

[ツール] > [オプション] を選択すると、以下に示すような IDE をカスタマイズするためのさまざまなコマンドを使用できます。

- エディタの設定
- エディタの色とフォントの設定
- プロジェクトビルドコマンドの設定
- C-SPY のウィンドウの設定
- 外部エディタの連携
- 共通フォントの変更
- キーバインディングの変更
- [メッセージ] ウィンドウへの出力数の設定

この他に、認識するファイル名拡張子の数を増やすことができます。デフォルトでは、ビルドツールチェーンの各ツールは、標準的なファイル名拡張子に対応します。それ以外のファイル名拡張子を持つソースファイルを使用する場合は、使用可能なファイル名拡張子を変更できます。[ツール] > [ファイル名の拡張子] を選択して、必要なコマンドを実行してください。

IDE をカスタマイズするコマンドのリファレンス情報については、128 ページの [ツール] メニューを参照してください。また、エディタのカスタマイズの詳細については、78 ページの *エディタ環境のカスタマイズ* を参照してください。C-SPY 関連のカスタマイズの詳細については、「ARM® 用 C-SPY® デバッガガイド」を参照してください。

外部ツールの呼出し

[ツール] メニューは設定可能なメニューであり、外部ツールを追加することによって、IDE 内から簡単にそれらのツールを使用できます。そのため、メニューコマンドとしてメニューに表示されるように事前に設定したツールに応じて、表示されるメニューが異なる場合があります。

メニューに外部ツールを追加するには、[ツール] > [ツールの設定] を選択して、[ツールの設定] ダイアログボックスを開きます。

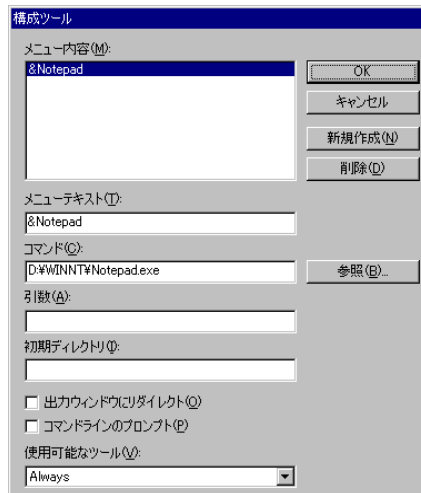


図 2: [ツールの設定] ダイアログボックス

このダイアログボックスのリファレンス情報については、150 ページの [ツールの設定] ダイアログボックスを参照してください。

注：IDE のツールチェーンの拡張に [ツールの設定] ダイアログボックスを使用することはできません (23 ページの ツールチェーンを参照)。

適切な情報を入力して **[OK]** をクリックすると、指定したメニューコマンドが **[ツール]** メニューに表示されます。

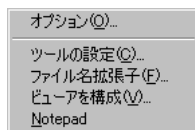


図3: カスタマイズした **[ツール]** メニュー

注: 標準ビルドツールチェーンに外部ツールを追加する場合は、「66 ページの **ツールチェーンの拡張**」を参照してください。

コマンドラインコマンドの追加

コマンドラインコマンドとバッチファイル呼出しは、コマンドシェルから実行する必要があります。コマンドラインコマンドを **[ツール]** メニューに追加すると、そのメニューからコマンドラインコマンドを実行できます。

- 1 **[ツール]** メニューにコマンドを追加するには、適切なコマンドシェルを指定する必要があります。**[コマンド]** テキストボックスに、たとえば `cmd.exe` というようにコマンドシェルを入力します。
- 2 **[引数]** テキストボックスで、コマンドラインコマンドかバッチファイル名を指定します。

[引数] のテキストは、以下に示すように指定する必要があります。

```
/C name
```

ここで、`name` は、実行するコマンドかバッチファイルの名前です。

`/c` オプションは、実行後にシェルを終了するように指定し、ツールの終了を IDE が検出できるようにします。

例

ネットワークドライブに `project` ディレクトリ全体のコピーを作成するために、コマンド **Backup** を **[ツール]** メニューに追加する場合は、**[コマンド]** をホスト環境に応じて `cmd.exe` か `command.cmd` を指定し、**[引数]** は下記のように指定します。

```
/C copy c:\project\%*. * F:
```

別の方法として、引数に変数を使用して、再配置可能パスを使用することもできます。

```
/C copy $PROJ_DIR\%*. * F:
```

プロジェクト管理

この章では、プロジェクトの編成方法、バージョンの異なるアプリケーションの管理を可能にする複数のプロジェクト、ビルド構成、グループ、ソースファイル、オプションを含むワークスペースの指定方法について説明します。また、サードパーティ製の外部バージョン管理システムを対話的に操作する手順についても説明します。

具体的には以下の項目を解説します。

- プロジェクト管理の概要
- プロジェクト管理の手順
- プロジェクト管理のリファレンス情報

プロジェクト管理の概要

ここでは、IDE のプロジェクト管理の概要を説明します。

以下のトピックを解説します。

- プロジェクト管理の概要について
- プロジェクトの作成方法
- バージョン管理システムの操作

プロジェクト管理の概要について

数百ものファイルを扱う大規模な開発プロジェクトでは、簡単にアクセスでき、数人のエンジニアによる保守が可能な構造に、ファイルを編成する必要があります。

IDE は、C/C++ ソースコードファイル、アセンブラファイル、インクルードファイル、その他の関連モジュールなど、すべてのプロジェクトモジュールを管理するための機能を装備しています。ワークスペースを作成し、1 つまたは複数のプロジェクトを追加できます。ファイルはグループ化が可能で、プロジェクト、グループ、ファイルのすべてのレベルでオプションを設定できます。リビルド実行時に必要なモジュールが再変換されるように、変更が記録されます。そのため、古いモジュールが含まれる実行可能ファイルが作成されることがありません。

他にも、以下のような特長があります。

- スムーズに開発が開始できるように、ビルドおよび実行がすぐに可能なプロジェクトテンプレートが付属
- プロジェクトを階層構造で表示
- 階層構造でシンボルを表示できるソースブラウザ
- オプションを全体、ソースファイルのグループ単位、個々のソースファイル単位に設定可能
- [作成] コマンドでは自動的に変更を検出して、必要な操作のみを実行します
- テキストベースのプロジェクトファイル
- Custom Build ユーティリティにより、標準ツールチェーンを簡単に拡張可能
- プロジェクトファイルを入力としてコマンドラインビルド可能

プロジェクトファイルのナビゲート

プロジェクトファイルをナビゲートするには、[ワークスペース] ウィンドウまたは[ソースブラウザ] ウィンドウを使用する、主に 2 種類の方法があります。[ワークスペース] ウィンドウには、論理的にグループ化されたソースファイル、依存ファイル、出力ファイルが、階層構造で表示されます。一方、[ソースブラウザ] ウィンドウには、現在[ワークスペース] ウィンドウでアクティブなビルド構成の情報が表示されます。ビルド構成については、変数、関数、型定義など、グローバルに定義されているすべてのシンボルが階層的に表示されます。クラスについては、基底クラスの情報も表示されます。

プロジェクトの作成方法

IDE は、ソフトウェア開発プロジェクトで通常行われる作成方法に合わせて設計されています。たとえば、バージョンの異なるターゲットハードウェアに対応して関連するバージョンのアプリケーションを開発する、初期のバージョンにはデバッグルーチンを組み込み最終アプリケーションには組み込まないようにする、などの作成が考えられます。

異なるターゲットハードウェアに応じ、複数バージョンのアプリケーションを開発する場合でも、ソースファイルは共通であることが多いので、それらのファイルのコピーを 1 つだけ保持するようにして、修正が自動的にアプリケーションの各バージョンに反映されるように編成することができます。また、ハードウェア依存部分を処理アプリケーションのように、複数のバージョンでソースファイルが異なる場合もあります。

IDE を使用すると、論理構造が一目でわかるような階層ツリー構造にプロジェクトを編成できます。以降のセクションでは、階層のさまざまなレベルについて説明します。

プロジェクトとワークスペース

通常は、1 つまたは複数のプロジェクトを作成し、それぞれが次のいずれかを含むようにします。

- ソースコードファイル。組込みアプリケーションまたはライブラリを生成するときに使用できます。ライブラリプロジェクトとアプリケーションプロジェクトを組み合わせた例については、チュートリアル「ライブラリの作成と利用」の例を参照してください。
- C-SPY でロードする外部でビルドされた実行可能ファイル。IDE の外でビルドされた実行可能ファイルをロードする方法については、『ARM® 用 C-SPY® デバッグガイド』を参照してください。

関連プロジェクトが複数存在する場合は、それらに同時にアクセスして操作できます。そのために、関連するプロジェクトをワークスペースに編成する機能があります。

1 つのワークスペースには、1 つ以上のプロジェクトを追加できます。どのプロジェクトも、少なくとも 1 つのワークスペースに属している必要があります。

1 つ例を示します。2 つの関連するアプリケーション、たとえば A と B を開発します。開発チーム A はアプリ A を、開発チーム B はアプリ B を開発します。2 つのアプリケーションは関連性があるので、ソースコードの一部は両アプリケーション間で共有できます。この場合、以下のプロジェクトモデルを適用できます。

- 3 つのプロジェクト。各アプリケーション用に 1 つずつのプロジェクト、共通ソースコード用にもう 1 つのプロジェクト
- 2 つのワークスペースチーム A のワークスペースとチーム B のワークスペース

共通のソースをライブラリプロジェクト（コンパイル済みだがリンクはされていないオブジェクトコード）にまとめる方法は、不要なコンパイルを避けることができるので、使いやすさと効率の両面で優れています。

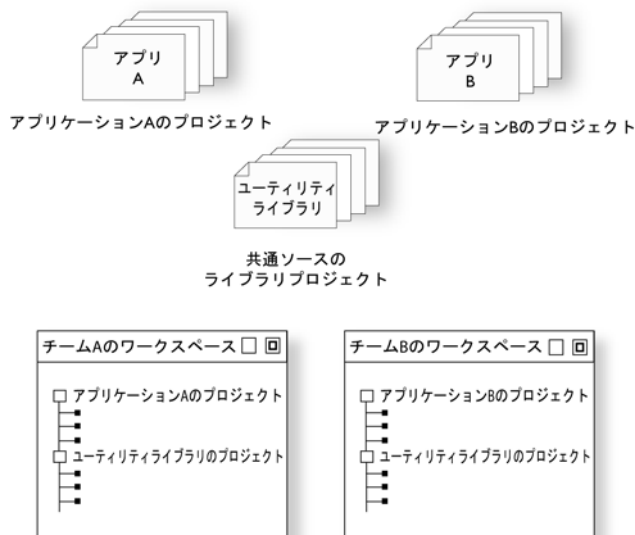


図4: ワークスペースとプロジェクトの例

プロジェクトとビルド構成

多くの場合、複数バージョンのプロジェクトをビルドする必要があります。Embedded Workbench を使用すると、プロジェクトごとに複数のビルド構成を定義できます。たとえば、**デバッグ**と**リリース**の2つだけを必要とする単純なケースがあります。この2つのビルド構成は、最適化、デバッグ情報、出力形式に使用するオプションだけが異なります。リリース構成では、プリプロセッサシンボル NDEBUG が定義され、アプリケーションにはアサートが含まれません。

ビルド構成を追加すると、複数のターゲットデバイス上でアプリケーションを使用する場合などに便利です。つまり、アプリケーションは同じで、コードのハードウェア関連の部分が異なる場合です。したがって、ビルドするターゲットデバイスに応じて、ビルド構成からいくつかのソースファイルを除外できます。プロジェクト A では、以下のビルド構成によって要件が満たされます。

- プロジェクト A - デバイス 1: リリース
- プロジェクト A - デバイス 1: デバッグ

- プロジェクト A - デバイス 2: リリース
- プロジェクト A - デバイス 2: デバッグ

グループ

通常は、プロジェクトには論理的に関連する数百のファイルが含まれます。そこで、関連するソースファイルをまとめてグループを定義して、そのような複数のグループを各プロジェクトに定義することができます。また、複数レベルのサブグループを定義して、論理階層を表現することもできます。デフォルトでは、グループはすべてのビルド構成に存在しますが、グループを特定のビルド構成から排除するように指定することもできます。

ソースファイルとそのパス

ソースファイルは、プロジェクトノードかグループ階層の直下に置くことができます。プロジェクトのファイルの数が多く、扱いにくい場合は、グループ階層を使用すると便利です。デフォルトでは、各ファイルはプロジェクトのすべてのビルド構成に存在しますが、ファイルを特定のビルド構成から排除するように指定することもできます。

実際にビルドされ、出力コードにリンクされるのは、ビルド構成に含まれるファイルだけです。

プロジェクトを正常にビルドすると、ソースファイルの下に、そこにインクルードされているファイルとそこから生成された出力ファイルが構造化されて表示されます。

注：ビルド構成の設定によって、ソースファイルのコンパイル時に使用するインクルードファイルを選択できます。これは、コンパイル後にソースファイルに関連付けられているインクルードファイルのセットは、ビルド構成によって異なる場合があることを意味します。

IDE は、ある程度までのソースファイルの相対パスをサポートします。

● プロジェクトファイル

プロジェクトファイルのファイル部分のパスは、同じドライブにある場合は相対パスです。パスは、`$PROJ_DIR$` または `EW_DIR` のいずれかに対して相対的です。引数変数 `EW_DIR` が使用されるのは、パスが `EW_DIR` へのサブディレクトリにあるファイルを参照し、`EW_DIR` からの距離が `$PROJ_DIR$` からの距離より短い場合だけです。

プロジェクトファイルの一部であるファイルのパスは、ファイルが異なるドライブにある場合は絶対パスです。

- ワークスペースファイル

ワークスペースファイルと同じドライブにあるファイルについては、パスは `$PROJ_DIR$` に対して相対的です。

ワークスペースファイルと異なるドライブにあるファイルについては、パスは絶対パスです。

- デバッグファイル

使用する ELF ファイルにデバッグ情報が含まれる場合、ソースファイルを参照するそのファイルのすべてのパスは絶対パスです。

ドラッグアンドドロップ

エクスプローラから、個々のソースファイルやプロジェクトファイルを [ワークスペース] ウィンドウにドラッグすることができます。ソースファイルをグループにドロップすると、そのグループに追加されます。プロジェクトツリーの外 ([ワークスペース] ウィンドウの背景) にドロップされたソースファイルは、アクティブプロジェクトに追加されます。

バージョン管理システムの操作

IAR Embedded Workbench IDE は、以下のすべてを識別してアクセスできます。

- Microsoft が発行する SCC (ソースコード管理) インタフェースに準拠したインストール済のサードパーティ製バージョン管理システム。
- サブバージョン (SVN) の作業用コピーにあるファイル。

IDE 内から IAR Embedded Workbench プロジェクトを外部 SCC プロジェクトまたは SVN プロジェクトに接続すると、通常使用する操作を一部実行できます。

IAR Embedded Workbench プロジェクトをバージョン管理システムに接続する場合、使用しているバージョン管理のクライアントアプリケーションの操作に慣れている必要があります。IDE からバージョン管理システムを操作する際に表示されるウィンドウやダイアログボックスの一部は、そのバージョン管理システムによって表示されたものであり、IAR システムズが提供するドキュメントでは説明されていないため注意してください。SCC システムのクライアントアプリケーションの詳細については、そのアプリケーションに付属するドキュメントを参照してください。

注：異なるバージョン管理システムでは、最も基本的な概念に関する部分であっても、使用されている用語が大きく異なります。IDE とバージョン管理システム間で操作を行う場合の説明を読む際は、この点に注意する必要があります。

プロジェクト管理の手順

ここでは、プロジェクト管理に関連する特定の機能の使用方法を手順ごとに説明します。

具体的には、以下の項目について説明します。

- ワークスペースの作成と管理
- ワークスペースの表示
- ブラウズ情報の表示
- SCC 互換のシステムの操作
- サブバージョンの操作

ワークスペースの作成と管理

ここでは、ワークスペースやプロジェクト、グループ、ファイル、ビルド構成を作成する全体的な手順を説明しますが、それに対応する各手順の説明は省略します。[ファイル] メニューには、ワークスペースを作成するコマンドがあります。[プロジェクト] メニューには、プロジェクトの作成、プロジェクトへのファイルの追加、グループの作成、プロジェクトオプションの指定、現在のプロジェクトに対する IAR システムズ製開発ツールの実行を行うコマンドがあります。

ワークスペースとその内容の作成と管理に関連する手順を以下に示します。

- ワークスペースの作成
空の [ワークスペース] ウィンドウが表示されます。ここで、プロジェクト、グループ、ファイルを表示できます。
- ワークスペースへの新規 / 既存プロジェクトの追加
新しいプロジェクトを作成する場合、プロジェクト設定が事前に設定されているテンプレートプロジェクトを使用できます。C アプリケーション、C++ アプリケーション、アセンブラアプリケーション、ライブラリプロジェクトのそれぞれで利用できるテンプレートプロジェクトがあります。
- グループの作成
グループは、プロジェクトの最上位ノードかプロジェクト内の別のグループのどちらかに追加できます。
- プロジェクトへのファイルの追加
ファイルは、プロジェクトの最上位ノードかプロジェクト内のグループのどちらかに追加できます。

- 新しいビルド構成の作成

デフォルトでは、ワークスペースに追加する各プロジェクトには、**Debug** と **Release** の 2 つのビルド構成があります。

新しいビルド構成を作成する場合、既存のビルド構成を使用できます。別の方法として、デフォルトのビルド構成を選択することもできます。

新しいビルド構成に使用するツールチェーンは、同じプロジェクト内の他のビルド構成と同一である必要はありません。

- ビルド構成からのグループとファイルの排除

[ワークスペース] ウィンドウで、排除されたグループやファイルを表すアイコンの色が白に変わることにご注意ください。

- プロジェクトから項目を削除

注：これらの手順の一部は実行する必要がない場合があります。

詳しい例については、チュートリアル「*アプリケーションプロジェクトの作成*」を参照してください。

ワークスペースの表示

「ワークスペース」ウィンドウは、アプリケーションの開発中にプロジェクトやファイルにアクセスするインターフェースです。

- 「ワークスペース」ウィンドウの下端にあるタブをクリックすると、そのプロジェクトが表示されます。

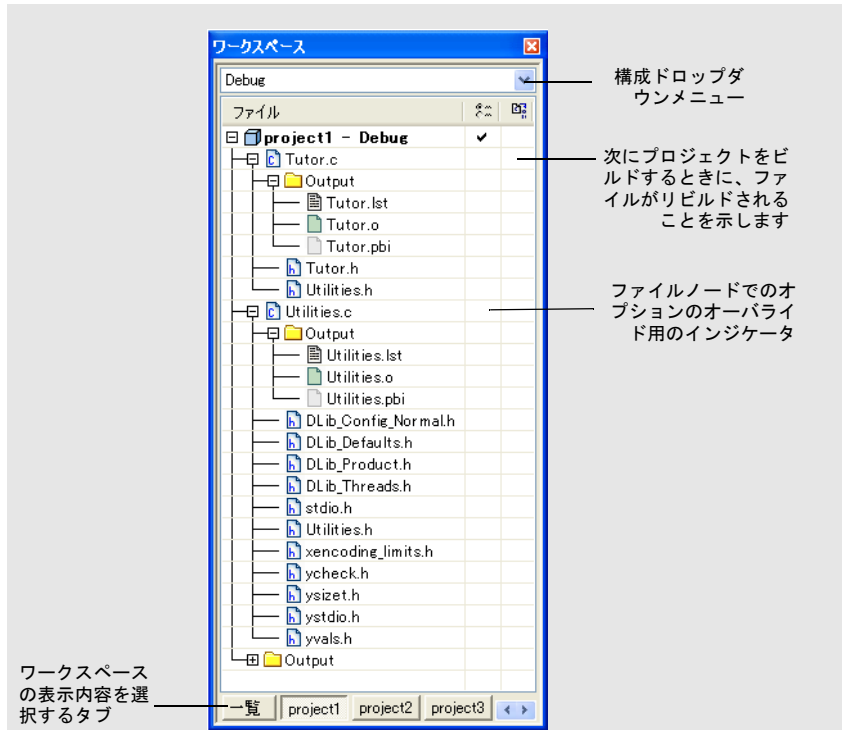


図5: 「ワークスペース」ウィンドウでプロジェクトを表示

ビルドされたファイルごとに出カフォルダアイコンが表示されます。このフォルダには、オブジェクトファイルやリストファイルなどの生成されたファイルがあります。リストファイルは、リストファイルオプションが有効な場合にのみ生成されます。プロジェクトノードにも出カフォルダが関連付けられています。このフォルダには、実行可能ファイルやリンカマップファイル（リストファイルオプションが有効な場合）など、プロジェクト全体に関連して生成されたファイルが含まれます。

また、インクルードされているヘッダファイルも表示され、依存関係をわかりやすく示しています。

- 別のビルド構成のプロジェクトを表示するには、[ワークスペース] ウィンドウ上端のドロップダウンリストからそのビルド構成を選択します。

選択したプロジェクトとビルド構成は、[ワークスペース] ウィンドウで強調表示されます。アプリケーションをビルドすると、ドロップダウンリストで選択したこのプロジェクトとビルド構成がビルドされます。

- ワークスペースですべてのプロジェクトの概要を表示するには、[ワークスペース] ウィンドウの下端にある **[概要]** タブをクリックします。

すべてのプロジェクトメンバの概要が表示されます。

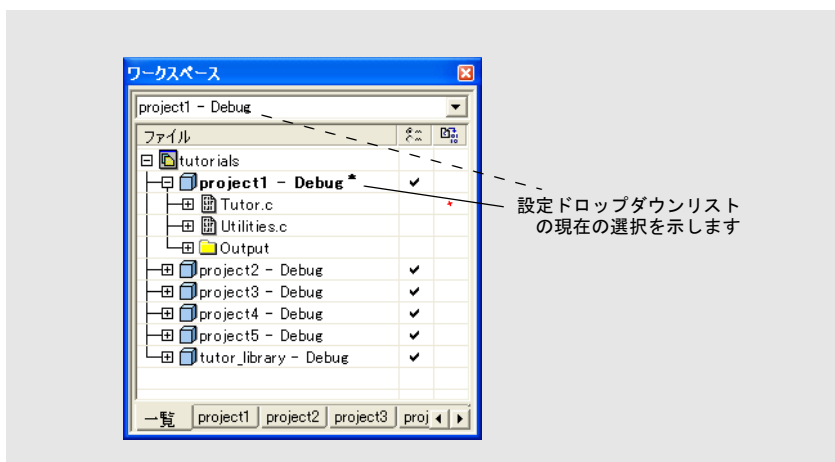


図 6: [ワークスペース] ウィンドウの概要

[ビルド構成] ドロップダウンリストで現在選択されている項目は、ワークスペースの概要が表示されるときも強調表示されます。

ブラウズ情報の表示

- [ソースブラウザ] ウィンドウを開くには、**[表示] > [ソースブラウザ]** を選択します。

[ソースブラウザ] ウィンドウは、デフォルトでは [ワークスペース] ウィンドウにドッキングされています。表示されるのは、アクティブなビルド構成のソースブラウズ情報です。

ウィンドウの上部ペインを右クリックして表示されるコンテキストメニューで、ファイルフィルタとタイプフィルタを選択できます。

- 2 [ソースブラウザ] ウィンドウでブラウズ情報を表示するには、[ツール] > [オプション] > [プロジェクト] を選択し、[ブラウズ情報を生成] オプションを選択します。
- 3 グローバルシンボルや関数の定義を表示するには、以下に示す 3 つの方法があります。
 - [ソースブラウザ] ウィンドウでシンボルか関数を右クリックして、表示されるコンテキストメニューで **[定義に移動]** コマンドを選択
 - [ソースブラウザ] ウィンドウで行をダブルクリック
 - エディタウィンドウでシンボルか関数を右クリックして、表示されるコンテキストメニューで **[定義に移動]** コマンドを選択

シンボルや関数の定義は、エディタウィンドウに表示されます。

ソースブラウズ情報は、バックグラウンドで継続的に更新されます。ソースファイルの編集や、新しいプロジェクトを開いているときは、最新の情報が表示されるまで少し時間がかかります。

SCC 互換のシステムの操作

SCC 互換のシステムでは、クライアントアプリケーションを使用して、主アーカイブを管理します。このアーカイブに、プロジェクトで使用するファイルの作業用コピーが保存されます。IAR Embedded Workbench にバージョン管理を統合することによって、通常使用するバージョン管理操作の一部を、IDE 内から直接、簡単な操作で実行できます。ただし、それ以外にもさまざまなタスクをクライアントアプリケーションで実行する必要があります。

IAR Embedded Workbench プロジェクトを SCC システムに接続するには、以下を行います。

- 1 Microsoft SCC 互換のクライアントアプリケーションで、SCC プロジェクトを設定します。
- 2 IDE で、アプリケーションプロジェクトを SCC プロジェクトに接続します。

SCC クライアントアプリケーションでの SCC プロジェクトの設定

SCC クライアントツールを使用して、SCC システムで管理する IAR Embedded Workbench プロジェクトのファイルの作業用ディレクトリを設定します。ファイルは、共通ルートの下の 1 つ以上のネストしたサブディレクトリに置かれます。具体的には、すべてのソースファイルは ewp プロジェクトファイルと同じディレクトリか、そのディレクトリのサブディレクトリに置かれます。

関連する手順については、SCC クライアントアプリケーションに付属するドキュメントを参照してください。

アプリケーションプロジェクトの SCC プロジェクトへの接続

- 1 [ワークスペース] ウィンドウで、SCC プロジェクトを作成したプロジェクトを選択します。
- 2 [プロジェクト] メニューから、[バージョン管理システム] > [プロジェクトを SCC プロジェクトに接続] を選択します。このコマンドは、[ワークスペース] ウィンドウを右クリックして表示されるコンテキストメニューでも選択できます。

注：[ソースコード管理] サブメニューのコマンドは、アプリケーションプロジェクトを問題なく SCC プロジェクトに接続したときに使用可能になります。

- 3 異なるベンダの SCC 互換システムをインストールしている場合、接続するシステムを選択するプロンプトが表示されます
- 4 SCC 固有のダイアログボックスが表示され、設定した SCC プロジェクトに移動します。

SCC システムにアクセスするコマンドのリファレンス情報については、53 ページの *SCC のバージョン管理システムメニュー* を参照してください。

SCC 状態の表示

IAR Embedded Workbench プロジェクトが SCC プロジェクトに接続されると、バージョン管理のステータス情報を表す列が [ワークスペース] ウィンドウに表示されます。状態に応じて異なるアイコンが表示されます。

これらの状態の組合せを表すアイコンが表示される場合もあります。表示される状態の解釈は、使用している SCC クライアントアプリケーションによって異なることに注意してください。アイコンとそれが表す状態のリファレンス情報については、58 ページの *ソースコード管理状態* を参照してください。

IDE と SCC 間の相互作用の設定

IDE と SCC 間の相互作用を設定するには、[ツール] > [オプション] を選択して、[ソースコード管理] タブをクリックします。使用できるコマンドのリファレンス情報については、143 ページの *[ソースコード管理] オプション* を参照してください。

サブバージョンの操作

IAR Embedded Workbench のバージョン管理統合では、クライアントアプリケーション `svn.exe` と `TortoiseProc.exe` を使用して、最も一般的なサブバージョン操作のいくつかを IDE から直接実行できます。

IAR Embedded Workbench のプロジェクトを SVN 管理システムに接続するには、以下の手順を実行する必要があります。

- 1 SVN クライアントアプリケーションで、SVN の作業用コピーを設定します。
- 2 IDE で、アプリケーションプロジェクトを SVN の作業用コピーに接続します。

サブバージョンの作業用コピーの設定

- 1 IDE でサブバージョンの統合を使用するには、svn.exe と TortoiseProc.exe がパスにあることを確認します。
- 2 サブバージョンリポジトリから作業用コピーをチェックアウトします。

プロジェクトを構成するファイルは、同じ作業用コピーからでなくても問題ありません。プロジェクトのすべてのファイルは個別に処理されます。ただし、TortoiseProc.exe では、異なるリポジトリからのファイルを同時にチェックインすることはできません。

サブバージョンの作業用コピーへのアプリケーションプロジェクトの接続

- 1 [ワークスペース] ウィンドウで、サブバージョンの作業用コピーを作成したプロジェクトを選択します。
- 2 [プロジェクト] メニューから、[バージョン管理システム] > [プロジェクトをサブバージョンに接続] を選択します。このコマンドは、[ワークスペース] ウィンドウを右クリックして表示されるコンテキストメニューでも選択できます。

サブバージョンの作業用コピーにアクセスするコマンドのリファレンス情報については、58 ページの *サブバージョンのバージョン管理システムメニュー* を参照してください。

サブバージョンの状態の表示

IAR Embedded Workbench プロジェクトがサブバージョンの作業用コピーに接続されると、バージョン管理のステータス情報を表す列が [ワークスペース] ウィンドウに表示されます。さまざまなアイコンが表示され、それぞれの svn のステータスを表します (60 ページの *サブバージョンの状態* を参照)。

プロジェクト管理のリファレンス情報

このセクションでは、以下のウィンドウおよびダイアログボックスのリファレンス情報を提供します。

- 42 ページの [ワークスペース] ウィンドウ

- 47 ページの [新規プロジェクトの作成] ダイアログボックス
- 48 ページの [プロジェクトの構成] ダイアログボックス
- 49 ページの [新規ビルド構成] ダイアログボックス
- 50 ページの [ソースブラウザ] ウィンドウ
- 53 ページの SCC のバージョン管理システムメニュー
- 55 ページの [ソースコード管理プロバイダの選択] ダイアログボックス
- 56 ページの [ファイルのチェックイン] ダイアログボックス
- 57 ページの [ファイルのチェックアウト] ダイアログボックス
- 58 ページの ソースコード管理状態
- 58 ページの サブバージョンのバージョン管理システムメニュー
- 60 ページの サブバージョンの状態

関連項目：

143 ページの [ソースコード管理] オプション

「ワークスペース」ウィンドウ

「ワークスペース」ウィンドウは「表示」メニューから利用できます。

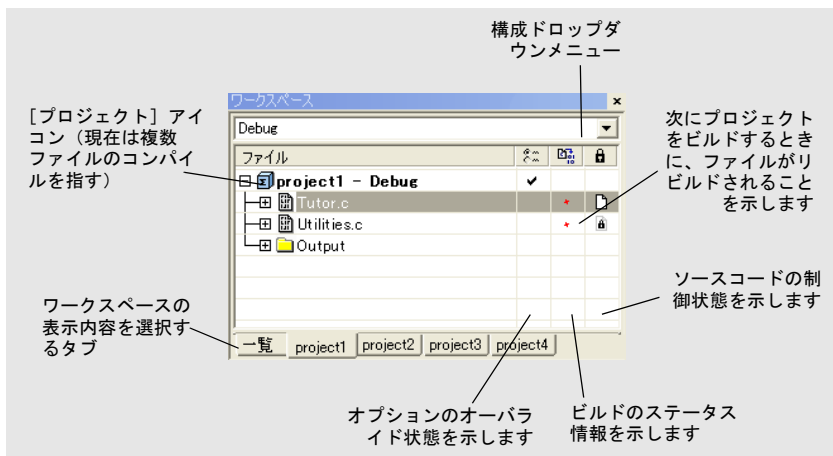


図7: 「ワークスペース」ウィンドウ

「ワークスペース」ウィンドウを使用して、アプリケーション開発中にプロジェクトやファイルにアクセスします。



















ドロップダウンリスト

ウィンドウ上部にあるドロップダウンリストでは、特定プロジェクト用のウィンドウで表示するビルド構成を選択できます。

表示エリア

このエリアには4つの列が含まれます。

【ファイル】列には、現在のワークスペースの名前、ワークスペースに含まれるプロジェクト、グループ、ファイルのツリー表現が表示されます。以下から1つまたは複数のアイコンが表示されます。

	ワークスペース
	プロジェクト
	複数ファイルのコンパイルを伴うプロジェクト
	ファイルグループ
	ビルドから除外されたグループ
	ファイルグループ、複数ファイルコンパイルの一部
	ファイルグループ、複数ファイルコンパイルの一部だがビルドから除外される
	オブジェクトファイルまたはライブラリ
	アセンブラソースファイル
	C ソースファイル
	C++ ソースファイル
	ビルドから除外されたソースファイル
	ヘッダファイル
	テキストファイル
	HTML テキストファイル
	制御ファイル、たとえばリンカ構成ファイルなど
	IDE 内部ファイル
	その他のファイル



オプションのオーバーライドに関するステータス情報を示す列には、3つのアイコンのいずれかが表示されます。

空白	このファイル/グループの設定/オーバーライドはありません。
黒のチェックマーク	このファイル/グループのローカル設定/オーバーライドがあります。
赤のチェックマーク	このファイル/グループのローカル設定/オーバーライドがありますが、これらは継承された設定と同じか、複数ファイルのコンパイルが使用されるため無視されます。すなわち、オーバーライドは不要です。



ビルドステータス情報を示す列には、プロジェクトのファイルごとに3つのアイコンのいずれかが表示されます。

空白	次回のプロジェクトのビルド時、このファイルはリビルドされません。
赤い星印	次回のプロジェクトのビルド時、このファイルはリビルドされます。
歯車	このファイルはリビルド中です。



この列にはバージョン管理のステータス情報が含まれます。さまざまなアイコンについては、以下を参照してください。

- 58 ページの *ソースコード管理状態*
- 60 ページの *サブバージョンの状態*

ウィンドウ下部のタブを使用して、表示するプロジェクトを選択できます。また、ワークスペース全体の概要を表示することもできます。

プロジェクト管理および「ワークスペース」ウィンドウの使用について詳しくは、29 ページの *プロジェクト管理の概要* を参照してください。

コンテキストメニュー

以下のコンテキストメニューがあります。

オプション(O)...
メイク(M) コンパイル(C) すべてを再ビルド(B) クリーン(L)
ビルドを停止(S)
追加(A) ▶
削除(V) 名前の変更...
バージョン管理システム(V) ▶
含むフォルダを開く... ファイルのプロパティ(P)...
アクティブに設定(E)

図 8: [ワークスペース] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

オプション	[ワークスペース] ウィンドウで選択した項目に対して、各ビルドツールのオプションを設定できるダイアログボックスを表示します。プロジェクト全体、ファイルのグループ、個々のファイルのオプションを設定できます。61 ページの オプションの設定を参照してください。
メイク	最後のビルド以降に変更されたファイルだけをコンパイル、アセンブル、リンクして、現在のターゲットを最新状態に更新します。
コンパイル	選択されているファイルを必要に応じてコンパイル / アセンブルします。[ワークスペース] ウィンドウで選択するか、コンパイル対象ファイルが開かれたエディタウィンドウを選択することで、ファイルを選択できます。
すべてを再ビルド	選択したビルド構成のすべてのファイルを再コンパイルし、再リンクします。
クリーン	中間ファイルを削除します。
ビルドを停止	現在のビルド処理を停止します。

追加 > ファイルの追加	プロジェクトにファイルを追加するためのダイアログボックスを表示します。
追加 > ファイル名の追加	指定したファイルをプロジェクトに追加します。このコマンドは、エディタで開かれているファイルがある場合にだけ使用できます。
追加 > グループの追加	[グループの追加] ダイアログボックスを表示して、新規グループをプロジェクトに追加できます。グループの詳細は、33 ページの <i>グループを参照</i> してください。
削除	選択した項目を [ワークスペース] ウィンドウから削除します。
名称変更	[グループの名称変更] ダイアログボックスが表示され、グループの名前を変更できます。グループの詳細は、33 ページの <i>グループを参照</i> してください。
バージョン管理システム	ソースコード管理用コマンドのサブメニューを表示します (53 ページの <i>SCC のバージョン管理システムメニュー</i> を参照)。
ファイルのあるフォルダを開く	選択したファイルが存在するディレクトリを表示するファイルエクスプローラを開きます。
ファイルのプロパティ	選択したファイルについて、標準の [ファイルプロパティ] ダイアログボックスを表示します。
アクティブに設定	概要ウィンドウで選択したプロジェクトをアクティブプロジェクトに設定します。[作成] コマンド実行時にビルドされるのがアクティブプロジェクトです。

【新規プロジェクトの作成】ダイアログボックス

【新規プロジェクトの作成】ダイアログボックスは、【プロジェクト】メニューから使用できます。



図9: 【新規プロジェクトの作成】ダイアログボックス

このダイアログボックスを使用して、テンプレートプロジェクトに基づいて新しいプロジェクトを作成します。テンプレートプロジェクトは、C/C++ アプリケーション、アセンブラアプリケーション、ライブラリプロジェクトに使用できます。また、自分でテンプレートプロジェクトを作成することもできます。

ツールチェーン

ビルド対象ターゲットを選択します。異なるターゲット用に複数のバージョンの IAR Embedded Workbench がホストコンピュータにインストールされている場合は、ドロップダウンリストからターゲットのすべてまたは一部を選択できることがあります。

プロジェクトテンプレート

この使用可能なテンプレートプロジェクトのリストから、新規プロジェクトの基となるテンプレートを選択します。

説明

現在選択されているテンプレートの説明。

【プロジェクトの構成】ダイアログボックス

【プロジェクトの構成】ダイアログボックスは、【プロジェクト】>【設定の編集】を選択するとアクセスできます。

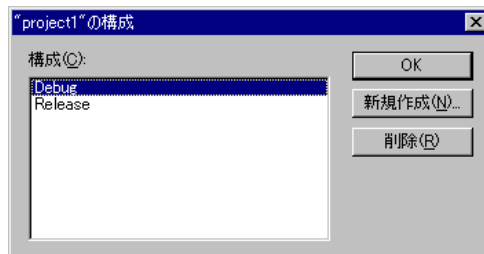


図10: 【プロジェクトの構成】ダイアログボックス

このダイアログボックスを使用して、選択したプロジェクトに新しいビルド構成を定義します。完全に新規か、前のプロジェクトに基づくものかどちらかです。

構成

既存の構成を表示します。ここに表示された構成を、新しい構成のテンプレートとして使用できます。

新規作成

新しいビルド構成を定義するためのダイアログボックスを表示します (49 ページの [【新規ビルド構成】ダイアログボックス](#)を参照)。

削除

【構成】リストで選択した構成を削除します。

「新規ビルド構成」ダイアログボックス

「新規構成」ダイアログボックスは、「プロジェクトの構成」ダイアログボックスで「新規」をクリックすると使用できます。

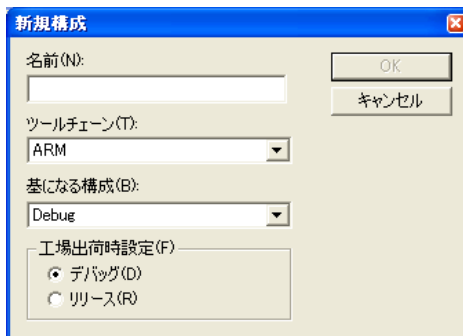


図 11: 「新規プロジェクトの作成」ダイアログボックス

このダイアログボックスを使用して、新しいビルド構成を定義します。全く新しい構成か、現在定義された構成に基づくかどうかです。

名前

ビルド構成名を入力します。

ツールチェーン

ビルド対象ターゲットを指定します。異なるターゲット用に複数のバージョンの IAR Embedded Workbench がホストコンピュータにインストールされている場合は、ドロップダウンリストからターゲットのすべてまたは一部を選択することができます。

基になる構成

新しい構成の基になる、現在定義済みのビルド構成を選択します。新しい構成は、プロジェクトの設定と出荷時設定情報を既存の構成から継承します。「なし」を選択すると、新しい構成は工場出荷時の設定だけに基づくようになります。

工場出荷時設定

新規のビルド構成に適用するデフォルトの工場出荷時設定を選択します。
【オプション】 ダイアログボックスの「工場出荷時設定」ボタンをクリックすると、ここで指定した出荷時設定がプロジェクトで使用されます。

以下から選択します。

Debug	デバッグのビルド構成に適した工場出荷時設定。
Release	リリースのビルド構成に適した工場出荷時設定。

「ソースブラウザ」ウィンドウ

「ソースブラウザ」ウィンドウは「表示」メニューから利用できます。

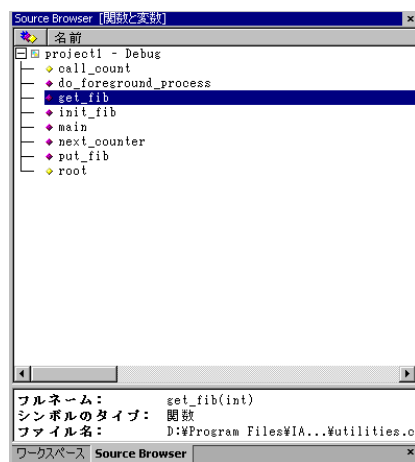


図 12: 「ソースブラウザ」ウィンドウ

「ソースブラウザ」ウィンドウには、アクティブなビルド構成で定義されているすべてのシンボルが、アルファベット順で階層表示されます。すなわち、ソースファイル内のシンボルについてソースブラウザの情報が利用でき、その構成のファイル部分が情報に含まれます。ソースブラウザの情報は、リンクされたライブラリ内のシンボルについては利用できません。このウィンドウは、2つの別々の表示エリアで構成されています。

「ソースブラウザ」ウィンドウの使用方法について詳しくは、38 ページの *ブラウザ情報の表示* を参照してください。

上側の表示エリア

上側の表示エリアには、2つの列があります。



シンボルの種類に対応するアイコンについては、51 ページの *シンボルの種類* に使用されるアイコンを参照してください。

名前 プロジェクトで定義されているグローバルシンボルや関数の名前が表示されます。名前のない `struct` や `union` のような未指定の型の場合、定義されたファイル名および行番号に基づいて名前が決められます。これらの擬似名は角括弧で囲まれています。

ウィンドウのヘッダをクリックすると、名前またはシンボルの種類を基準にシンボルをソートできます。

上側の表示エリアでは、コンテキストメニューも使用できます (52 ページのコンテキストメニューを参照)。












下の表示エリア




上側の表示エリアで選択したシンボルに応じて、下側のエリアにそのプロパティが表示されます。

フルネーム	各エレメントの一意の名前 (<code>classname::membername</code> など) が表示されます。
シンボルのタイプ	各エレメントのシンボルの種類が表示されます (51 ページのシンボルの種類に使用されるアイコンを参照)。
ファイル名	エレメントが定義されているファイルのパスを示します。

シンボルの種類に使用されるアイコン

使用されるアイコンは次のとおりです。

	基底クラス
	クラス
	設定
	列挙型
	列挙定数
 (黄色のひし形)	構造体のフィールド
 (紫のひし形)	関数
	マクロ
	名前空間
	テンプレートクラス
	テンプレート関数

	タイプの定義
	共用体
	変数

コンテキストメニュー

以下のコンテキストメニューが表示エリアの上部分で使用できます。

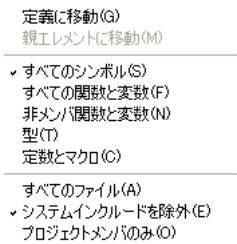


図 13: [ソースブラウザ] ウィンドウのコンテキストメニュー

コンテキストメニューから実行できるコマンドは以下のとおりです。

定義に移動	エディタウィンドウで、選択した項目の定義を表示します。
親に移動	選択したエレメントがクラス、構造体、共用体、列挙型、名前空間のメンバーである場合に、このメニューコマンドを使用して、そのエレメントの親エレメントに移動することができます。
すべてのシンボル	タイプフィルタ。プロジェクト内で定義されたすべてのグローバルシンボルや関数が表示されます。
すべての関数と変数	タイプフィルタ。プロジェクト内で定義されたすべての関数や変数が表示されます。
非メンバ関数と変数	タイプフィルタ。クラスのメンバではない関数および変数がすべて表示されます。
タイプ	タイプフィルタ。プロジェクト内で定義されたすべての型（構造体、クラスなど）が表示されます。
定数とマクロ	タイプフィルタ。プロジェクト内で定義されたすべての定数やマクロが表示されます。

すべてのファイル	ファイルフィルタ。プロジェクトに明示的に追加したすべてのファイルと、それらのファイルでインクルードされるすべてのファイル内のシンボルが表示されます。
システムインクルードを除外	ファイルフィルタ。プロジェクトに明示的に追加したすべてのファイルと、それらのファイルでインクルードされるすべてのファイル内（IAR Embedded Workbenchのインストールディレクトリ内のインクルードファイルを除く）のシンボルが表示されます。
プロジェクトメンバのみ	ファイルフィルタ。プロジェクトに明示的に追加したすべてのファイル（それらのファイルでインクルードされるファイルを除く）内のシンボルが表示されます。

SCC のバージョン管理システムメニュー

[バージョン管理システム] サブメニューは、[プロジェクト] メニューか、[ワークスペース] ウィンドウのコンテキストメニューから選択できます。

以下は SCC 互換システムのメニューです。

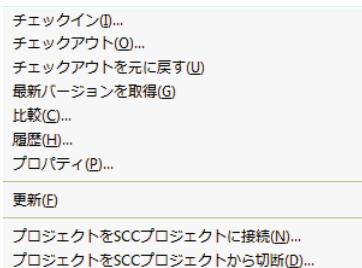


図 14: SCC のバージョン管理システムメニュー

注：バージョン管理システムのサブメニューの内容は、使用中のバージョン管理システムを反映し、SCC 互換システムまたはサブバージョンのどちらかです。

外部バージョン管理システムの操作の詳細は、34 ページの *バージョン管理システムの操作* を参照してください。

以下のコマンドが SCC で使用できます。

チェックイン	[ファイルのチェックイン] ダイアログボックスを表示します。このダイアログボックスで、選択したファイルをチェックインできます (56 ページの [ファイルのチェックイン] ダイアログボックスを参照)。ファイルで行ったすべての変更は、アーカイブに保存されます。このコマンドは、チェックアウトしているファイルを [ワークスペース] ウィンドウで選択すると使用可能になります。
チェックアウト	選択したファイルをチェックアウトします。使用する SCC (ソースコード管理) システムによっては、ダイアログボックスが表示されます (57 ページの [ファイルのチェックアウト] ダイアログボックスを参照)。チェックアウトとは、ファイルをローカルにコピーして編集できるようにすることです。このコマンドは、チェックインしているファイルを [ワークスペース] ウィンドウで選択すると使用可能になります。
チェックアウトを元に戻す	選択したファイルを最新のアーカイブバージョンに戻します。ファイルはチェックアウト状態ではなくなります。ファイルで行ったすべての変更は破棄されます。このコマンドは、チェックアウトしているファイルを [ワークスペース] ウィンドウで選択すると使用可能になります。
最新バージョンを取得	選択したファイルを最新のアーカイブバージョンに置き換えます。
比較	SCC 専用ウィンドウで、ローカルバージョンと最新アーカイブバージョンの違いを表示します。
履歴	選択したファイルのレビジョン履歴に関する SCC 固有情報を表示します。
プロパティ	選択したファイルについてバージョン管理システムで使用可能な情報を表示します。
更新	プロジェクトに含まれるすべてのファイルのバージョン管理システムの表示ステータスを更新します。このコマンドは、バージョン管理システムで管理するすべてのプロジェクトで常に使用できます。

**プロジェクトを
SCC プロジェク
トに接続**

SCC クライアントアプリケーションでダイアログを表示し、選択した IAR Embedded Workbench プロジェクトと SCC プロジェクトを関連付けます。IAR Embedded Workbench プロジェクトが SCC 管理化のプロジェクトになります。関連付けを行うと、ステータス情報を示す特別な列が [ワークスペース] ウィンドウに表示されます。

**プロジェクトを
SCC プロジェク
トから切断**

選択した IAR Embedded Workbench プロジェクトと SCC プロジェクトの関連付けを削除します。プロジェクトが SCC 管理の対象外になります。[ワークスペース] ウィンドウで SCC ステータス情報を示す列が、そのプロジェクトについては表示されなくなります。

【ソースコード管理プロバイダの選択】 ダイアログボックス

さまざまなベンダ製の複数の SCC システムが使用可能な場合は、[ソースコード管理プロバイダの選択] ダイアログボックスが表示されます。



図 15: 【ソースコード管理プロバイダの選択】 ダイアログボックス

このダイアログボックスを使用して、使用する SCC システムを選択します。

【ファイルのチェックイン】ダイアログボックス

【ファイルのチェックイン】ダイアログボックスは、[プロジェクト] > [ソースコード管理] > [チェックイン] コマンドを選択する（[ワークスペース] ウィンドウのコンテキストメニューからも選択可能）と表示されます。

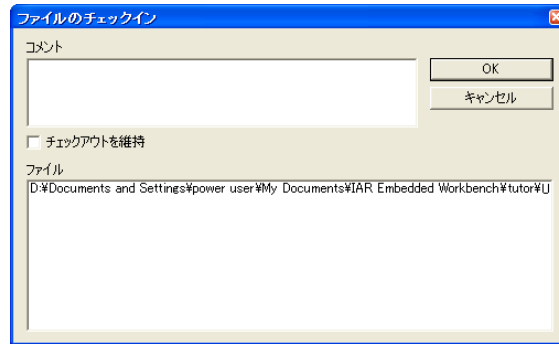


図 16: ファイルのチェックインダイアログボックス

コメント

ファイルのレビジョンとともにアーカイブに格納されるコメント（通常は変更の説明）を指定します。このテキストボックスは、SCC システムがチェックイン時のコメント追加をサポートしている場合にだけ使用可能になります。

チェックアウトを維持

チェックイン後も、ファイルがチェックアウト状態のままになるよう指示します。通常は、ファイルに対する作業を停止せずに、プロジェクトチームの他のメンバが修正を確認できるようにする場合に使用します。

詳細設定

SCC クライアントアプリケーションの詳細オプション設定用ダイアログボックスを表示します。このボタンは、SCC システムがチェックイン時の詳細オプション設定をサポートしている場合にだけ使用可能になります。

ファイル

チェックインされるファイルのリストを表示します。リストには、【ファイルのチェックイン】ダイアログボックス表示時に [ワークスペース] ウィンドウで選択されていたすべてのファイルが表示されます。

【ファイルのチェックアウト】ダイアログボックス

【ファイルのチェックアウト】ダイアログボックスは、[プロジェクト] > [ソースコード管理] > [チェックアウト] コマンド（[ワークスペース] ウィンドウのコンテキストメニューからも選択可能）を選択すると表示されます。ただし、このボタンは、SCC システムがチェックアウト時のコメント追加や詳細オプション設定をサポートしている場合にだけ使用可能になります。

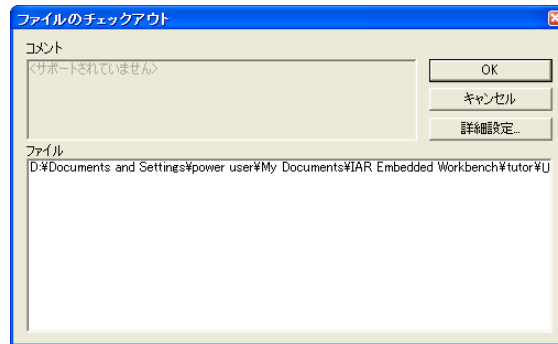


図 17: ファイルのチェックアウトダイアログボックス

コメント

ファイルのレビジョンとともにアーカイブに配置されるコメント（通常はファイルのチェックアウト理由）を指定します。このテキストボックスは、SCC システムがチェックアウト時のコメント追加をサポートしている場合にだけ使用可能になります。

詳細設定







SCC クライアントアプリケーションの詳細オプション設定用ダイアログボックスを表示します。このボタンは、SCC システムがチェックアウト時の詳細オプション設定をサポートしている場合にだけ使用可能になります。

ファイル

チェックアウトされるファイルのリストを表示します。リストには、【ファイルのチェックアウト】ダイアログボックス表示時に [ワークスペース] ウィンドウで選択されていたすべてのファイルが表示されます。

ソースコード管理状態

ソースコード管理対象ファイルは、複数の状態のいずれかになります。

-  (ブランク) チェックアウトされています。ファイルの編集が可能です。
-  (チェックマーク) チェックアウトされています。ファイルの編集が可能で、ファイルが修正されています。
-  (灰色の錠) チェックインされています。多くの SCC システムでは、ファイルが書き込み保護されていることを示します。
-  (灰色の錠) チェックインされています。アーカイブに新しいバージョンがあります。
-  (赤の錠) 他のユーザ専用 to チェックアウトされています。多くの SCC システムでは、ファイルをチェックアウトできないことを示します。
-  (赤の錠) 他のユーザ専用 to チェックアウトされています。アーカイブに新しいバージョンがあります。多くの SCC システムでは、ファイルをチェックアウトできないことを示します。

注：IAR Embedded Workbench IDE でのソースコード管理では、SCC システムが提供する情報を使用します。SCC システムがステータスについて誤った、あるいは不完全な情報を提供すると、IDE で誤ったアイコンが表示されることがあります。

サブバージョンのバージョン管理システムメニュー

[バージョン管理システム] サブメニューは、[プロジェクト] メニューか、[ワークスペース] ウィンドウのコンテキストメニューから選択できます。

以下はサブバージョンのメニューです。

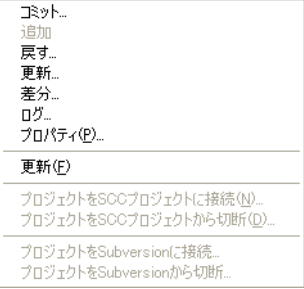


図 18: サブバージョンのバージョン管理システムメニュー

注：バージョン管理システムのサブメニューの内容は、使用中のバージョン管理システムを反映し、**SCC 互換システム**または**サブバージョン**のどちらかです。












外部バージョン管理システムの操作の詳細は、34 ページの **バージョン管理システムの操作**を参照してください。

以下のコマンドがサブバージョンで使用できます。

コミット	選択したファイルについて Tortoise の ［コミット］ ダイアログボックスを表示します。
追加	選択したファイルについて Tortoise の ［追加］ ダイアログボックスを表示します。
戻す	選択したファイルについて Tortoise の ［戻す］ ダイアログボックスを表示します。
更新	選択したファイルについて Tortoise の ［更新］ ウィンドウを開きます。
差分	選択したファイルについて Tortoise の ［差分］ ウィンドウを開きます。
ログ	選択したファイルについて Tortoise の ［ログ］ ウィンドウを開きます。
プロパティ	選択したファイルについてバージョン管理システムで使用可能な情報を表示します。
更新	プロジェクトに含まれるすべてのファイルのバージョン管理システムの表示ステータスを更新します。このコマンドは、バージョン管理システムで管理するすべてのプロジェクトで常に使用できます。
プロジェクトを SVN プロジェクトに接続	svn.exe と TortoiseProc.exe がパスにあるかどうかを確認し、IAR Embedded Workbench プロジェクトと既存のチェックアウトされた作業用コピーとの接続を有効にします。この接続を作成すると、ステータス情報を示す特別な列が ［ワークスペース］ ウィンドウに表示されます。ソースファイルは IDE の外部からチェックアウトしなくてはならない点に注意してください。
プロジェクトを SVN プロジェクトから切断	選択した IAR Embedded Workbench プロジェクトとサブバージョンの接続を削除します。 ［ワークスペース］ ウィンドウで SVN ステータス情報を示す列が、そのプロジェクトについては表示されなくなります。

サブバージョンの状態

サブバージョンにより管理される各ファイルは、複数の状態のいずれかになります。

	(青の A)	追加済。
	(赤の C)	衝突あり。
	(赤の D)	削除済。
	(赤の I)	無視されました。
	(ブランク)	変更なし。
	(赤の M)	変更済。
	(赤の R)	置換済。
	(灰色の X)	外部定義によって作成されたバージョンがつけられていないディレクトリ。
	(灰色の疑問符 question mark)	アイテムがバージョン管理の下にありません。
	(黒の感嘆符)	アイテムが存在しないか (svn 以外のコマンドにより削除済)、不完全です。
	(赤の波形符号)	アイテムが別の型のアイテムによって妨害されました。

注：IAR Embedded Workbench IDE のバージョン管理システムでは、サブバージョンが提供する情報を使用します。サブバージョンが状態について誤った、あるいは不完全な情報を提供すると、IDE で誤ったアイコンが表示されることがあります。

ビルド

この章では、プロジェクトをビルドする処理について簡単に説明し、続いてサードパーティ製ツールをビルドツールチェーンに追加する方法について説明します。

プロジェクトのビルド

ビルド処理は、以下の手順で構成されます。

- プロジェクトオプションの設定
- アプリケーションプロジェクトまたはライブラリプロジェクトのビルド
- ビルドで検出されたエラーの修正

バッチビルドコマンドを使用すると、ビルド処理の効率を上げることができます。このコマンドを使用すると、1回の操作で複数のビルドを実行できます。必要に応じて、ビルド前とビルド後のアクションを指定することも可能です。

プロジェクトをビルドするには、**IAR Embedded Workbench IDE** を使用する以外に、コマンドラインユーティリティ `iarbuild.exe` を使用方法もあります。

アプリケーションおよびライブラリオブジェクトのビルドの例については、インフォメーションセンタのチュートリアルを参照してください。ライブラリプロジェクトのビルドの詳細については、『**ARM 用 IAR C/C++ 開発ガイド**』を参照してください。

オプションの設定

プロジェクトのビルド方法を指定するには、1つ以上のビルド構成を定義する必要があります。ビルド構成はそれぞれ独自の設定があり、他の設定には依存しません。設定はすべて [ワークスペース] ウィンドウで個別の列に表示されます。

たとえば、デバッグに使用する設定は、最適化の程度は低く、デバッグに適した出力を生成します。逆に、最終アプリケーションのビルド構成は、高度に最適化され、フラッシュ / PROM プログラマに適した出力を生成します。

各ビルド構成に対して、プロジェクトレベル、グループレベル、ファイルレベルでオプションを設定できます。プロジェクトレベルでのみ設定できるオプションが数多く存在しますが、これはそれらのオプションがビルド構成全体に影響を与えるためです。このようなオプションの例としては一般オプション（派生プロセッサやライブラリオブジェクトファイルなど）、リンク設定、デバッグ設定があります。それ以外の、コンパイラオプションやアセンブラオプションなどのオプションをプロジェクトレベルで設定すると、それがビルド構成全体のデフォルト設定になります。

プロジェクトレベルの設定をオーバーライドするには、必要な項目（たとえば特定のファイルグループ）を選択して、オプション **「継承した設定をオーバーライド」** を選択します。新しい設定は、選択されたグループのすべてのメンバ、すなわちファイルとファイルグループに影響を与えます。すべての設定をデフォルトの出荷時設定に戻すには、**「工場出荷時設定」** ボタンをクリックします。

注： オプションの設定について、1 つ重要な制限があります。グループまたはファイルレベルでオプションを設定（グループまたはファイルレベルでオーバーライド）すると、ファイルを適用対象とする上位レベルのオプションは、一切このグループまたはファイルに適用されなくなります。

【オプション】 ダイアログボックスの使用

【オプション】 ダイアログボックス（**【プロジェクト】** > **【オプション】**）を選択して表示）では、ビルドツールのオプションを設定できます。設定したオプションは、**「ワークスペース」** ウィンドウで選択されている項目に適用されます。**【一般オプション】**、**【リンカ】**、**【デバッグ】** の各カテゴリのオプションは、ビルド構成全体に対してのみ設定できます。個別のグループやファイルに対しては設定できません。ただし、他のカテゴリのオプションは、ビルド構成全体だけでなく、ファイルグループや個別のファイルに対しても設定できます。

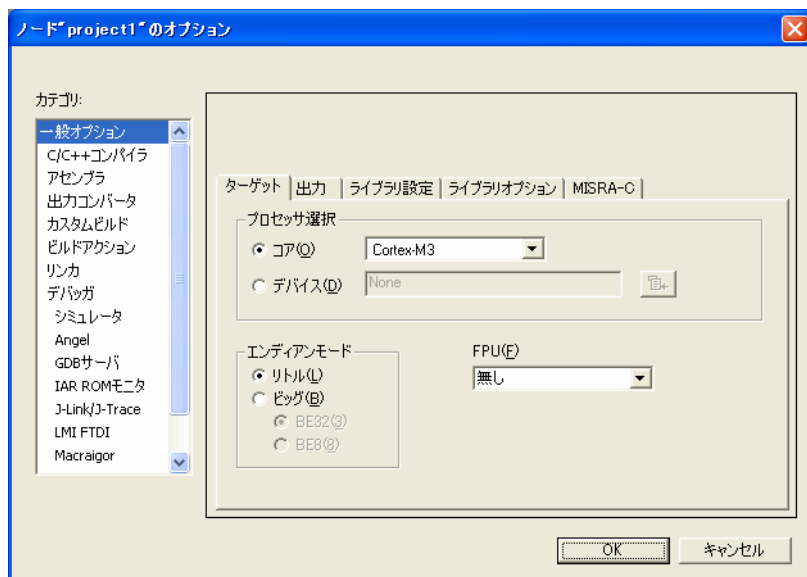


図 19: 一般オプション

[カテゴリ] リストで、オプションを設定するビルドツールを選択できます。
[カテゴリ] リストで利用できるツールは、製品に含まれるツールによって異なります。カテゴリを選択すると、そのコンポーネントのオプションを含むページが表示されます。このページは複数ページにわたる場合もあります。

表示 / 変更するオプションのタイプに対応するタブをクリックします。すべての設定をデフォルトの出荷時設定に戻すには、[工場出荷時設定] ボタンをクリックします。このボタンは、[一般オプション] と [カスタムビルド] を除くすべてのカテゴリで使用できます。使用可能な出荷時設定は 2 つあります。Debug と Release です。どちらを使用するかは、ビルド構成に応じて異なります (49 ページの [新規ビルド構成] ダイアログボックス参照)。

各オプションの説明とオプションの設定方法については、以下の章を参照してください。

- 一般オプション
- コンパイラオプション
- アセンブラオプション
- 出力コンバータオプション
- リンカのオプション
- ライブラリビルダオプション
- カスタムビルドオプション
- C-SPY ドライバオプション (『ARM® 用 C-SPY® デバグガガイド』を参照)

注：認識されないファイル名拡張子を持つソースファイルをプロジェクトに追加した場合、そのソースファイルに対してオプションを設定することはできません。ただし、そのファイル名拡張子に対するサポートを追加できます。リファレンス情報については、152 ページの [ファイル名拡張子] ダイアログボックスを参照してください。

プロジェクトのビルド

プロジェクトは、アプリケーションプロジェクトまたはライブラリプロジェクトとしてビルドできます。

注：ライブラリプロジェクトとしてプロジェクトをビルドするには、プロジェクトをビルドする前に [プロジェクト] > [オプション] > [一般オプション] > [出力] > [出力ファイル] > [ライブラリ] を選択します。続いて、オプションのダイアログボックスの [カテゴリ] リストで [リンカ] が [ライブラリビルダ] に置き換わり、ビルドの結果がライブラリになります。例については、チュートリアルを参照してください。

ビルドコマンドにアクセスするには、[プロジェクト] メニューか [ワークスペース] ウィンドウで項目を右クリックして表示されるコンテキストメニューを使用します。

3 つのビルドコマンド、**Make**、**Compile**、**Rebuild All** はバックグラウンドで動作するので、プロジェクトをビルドしている間も IDE で編集作業を続行できます。

詳細なリファレンス情報については、118 ページの [プロジェクト] メニューを参照してください。

バッチによる複数構成のビルド

バッチビルド機能を使用すると、複数の構成を同時にビルドできます。バッチは、ビルド構成が順序付けて記述されているリストです。[バッチビルド] ダイアログボックスは、[プロジェクト] メニューからアクセスでき、複数の構成バッチを作成、変更、ビルドできます。

複数の構成を含むワークスペースの場合は、複数のバッチを定義すると便利です。ワークスペース全体をビルドするのではなく、リリース設定とデバッグ設定のように特定のビルド構成だけをビルドできます。

[バッチビルド] ダイアログボックスの詳細については、126 ページの [バッチビルド] ダイアログボックスを参照してください。

ビルド前およびビルド後のアクションの使用

必要に応じて、ビルド前とビルド後に実行するアクションを指定することが可能です。[ビルドアクション] ダイアログボックス ([プロジェクト] メニューからアクセス) を使用して必要なアクションを指定できます。

[ビルドアクション] ダイアログボックスについて詳しくは、『195 ページのビルドアクションオプション』を参照してください。



ビルド前アクションの使用によるタイムスタンプ

ビルド前アクションを使用して、ビルドに関するタイムスタンプを結果のバイナリファイルに埋め込むことができます。以下の手順を実行します。

- 1 専用のタイムスタンプファイル (timestamp.c など) を作成し、プロジェクトに追加します。
- 2 このソースファイルで、プリプロセッサマクロ `__TIME__` と `__DATE__` を使用して文字列変数を初期化します。
- 3 [プロジェクト] > [オプション] > [ビルドアクション] を選択して、[ビルドアクション] ダイアログボックスを開きます。

- 4 [プリビルドコマンドライン] テキストフィールドにビルド前アクションを指定します。たとえば、以下のように指定します。

```
"touch $PROJ_DIR$%timestamp.c"
```

オープンソースコマンドラインユーティリティ touch をこの目的に使用できます。また、他にもソースファイルの変更時刻を更新する適当なユーティリティを使用できます。

- 5 プロジェクトが完全には最新状態でない場合は、次の **Make** コマンドの使用時に、通常のビルドプロセスの前にビルド前アクションが呼び出されます。そして、通常のビルドプロセスで常に timestamp.c が再コンパイルされ、最終的には正しいタイムスタンプがバイナリファイルに埋め込まれます。

すでにプロジェクトが最新状態の場合には、ビルド前アクションは呼び出されません。すなわち、ビルドは実行されず、バイナリファイルには最後にビルドされたときのタイムスタンプが引き続き使用されます。

ビルド中に検出されたエラーの修正

コンパイラ、アセンブラ、デバッガは、完全に開発環境に統合されています。ソースコードにエラーが含まれる場合、[ビルド] メッセージウィンドウでエラーリストのエラーメッセージをダブルクリックするか、エラーを選択して **Enter** キーを押すことによって、該当するソースファイルの選択したエラーの位置に直接移動できます。

ビルド中に検出された問題をすべて解決して、プロジェクトをリビルドしたら、生成されたコードをソースレベルで直接デバッグできます。

[ビルド] メッセージウィンドウへの出力レベルを指定するには、[ツール] > [オプション] を選択して [IDE オプション] ダイアログボックスを開きます。[メッセージ] タブをクリックして、[ビルドメッセージの表示] ドロップダウンリストで出力レベルを選択します。または、[ビルドメッセージ] ウィンドウでダブルクリックし、コンテキストメニューから [オプション] を選択します。

[ビルド] メッセージウィンドウのリファレンス情報については、99 ページの [ビルド] ウィンドウを参照してください。

コマンドラインからのビルド

コマンドラインからプロジェクトをビルドするには、common¥bin ディレクトリにある IAR コマンドラインビルドユーティリティ (iarbuild.exe) を使用します。入力としてプロジェクトファイルを使用して、以下の構文で呼び出します。

```
iarbuild project.ewp [-clean|-build|-make] <configuration>
[-log errors|warnings|info|all]
```

パラメータ	説明
project.ewp	IAR Embedded Workbench プロジェクトファイル。
-clean	すべての中間ファイルおよび出力ファイルを削除します。
-build	現在のビルド構成のすべてのファイルをリビルド / 再リンク。
-make	最後のビルド以降に変更されたファイルだけをコンパイル、アセンブル、リンクして、現在のビルド構成を最新状態に更新。
configuration	ビルドする構成の名前（定義済の構成 [デバッグ] または [リリース] か、ユーザが独自に定義した名前）を指定（ビルド構成の詳細については、32 ページの <i>プロジェクトとビルド構成</i> を参照）。
-log errors	ビルドのエラーメッセージを表示。
-log warnings	ビルドのワーニング、エラーメッセージを表示。
-log info	ビルドのワーニングメッセージ、エラーメッセージ、および #pragma message プリプロセッサディレクティブによって出力されるメッセージを表示します。
-log all	ビルドで出力されるすべてのメッセージを表示（コンパイラのサインオン情報やフルコマンドラインなど）。

表 3: iarbuild.exe コマンドラインオプション

プロジェクトファイルを指定しないでコマンドシェルからアプリケーションを実行すると、使用できるパラメータとその構文を示すサインオンメッセージが表示されます。

ツールチェーンの拡張

IAR Embedded Workbench では、標準のツールチェーンを拡張するためのツール (Custom Build) が提供されています。この機能を使用して、外部ツール (IAR システムズ以外のベンダが提供するツール) を実行します。プロジェクト内の特定のファイルが変更されるたびに、外部ツールを実行させることができます。

[カスタムツール構成] ページでカスタムビルドオプションを指定すると、ビルドコマンドは、IAR Embedded Workbench IDE とその関連ファイルを処理す

のと同じ方法で、外部ツールとその関連ファイルを処理します。外部ツールとその入力ファイルと生成される出力ファイルの関係は、C/C++ コンパイラ、c ファイル、h ファイル、o ファイルの間の関係に似ています。使用可能なカスタムビルドオプションについて詳しくは、193 ページの *カスタムビルドオプション* を参照してください。

外部ツールの入力として使用するファイルのファイル名拡張子を指定します。プロジェクトを最後にビルドした後で入力ファイルが変更された場合は、c ファイルが変更されたときにコンパイラが実行されるのと同じように、外部ツールが実行されます。同様に、他の入力ファイル（インクルードファイルなど）への変更も検出されます。

外部ツールの名前を指定する必要があります。同時に、外部ツールが必要とするコマンドラインオプションや、外部ツールが生成する出力ファイルの名前も指定できます。また、ファイルパスを置換する引数変数も使用できます。

ファイル情報の一部を表すのに、引数変数を使用できます。

カスタムビルドオプションは、プロジェクトツリーの任意のレベルに対して指定できます。指定したオプションは、プロジェクトツリーの下位レベルに継承されます。

ツールチェーンに追加可能なツール

IAR Embedded Workbench ツールチェーンに追加できる外部ツールか、ツールの種類の例を以下に示します。

- 言語仕様に基づいてファイルを生成するツール（Lex、YACC など）
- バイナリファイル、たとえばビットマップイメージやオーディオデータを含むファイルを、アセンブラか C ソースファイルのデータテーブルに変換するツールです。（このデータは、コンパイルして、アプリケーションの他のファイルとリンク可能）

外部ツールの追加

ツール *Flex* をツールチェーンに追加する例を以下に示します。他のツールも同じ手順で追加できます。

この例では、Flex はファイル `myFile.lex` を入力として受け取ります。2 つのファイル、`myFile.c` と `myFile.h` が出力として生成されます。

- 1 `myFile.lex` など、使用するファイルをプロジェクトに追加します。
- 2 [ワークスペース] ウィンドウでこのファイルを選択して、[プロジェクト] > [オプション] を選びます。カテゴリリストで [カスタムビルド] を選択します。

- 3** **【ファイル名の拡張子】** フィールドにファイル名拡張子「.lex」を入力します。先頭にピリオド(.)を指定するのを忘れないでください。

- 4** **【コマンドライン】** フィールドに、外部ツールを実行するコマンドラインを入力します。以下に例を示します。

```
flex $FILE_PATH$ -o$FILE_BNAME$.c
```

ビルド処理中に、このコマンドラインは以下のように展開されます。

```
flex myFile.lex -omyFile.c
```

引数変数の使用方法に注意してください。特に \$FILE_BNAME\$ の使用方法には注意してください。これは入力ファイルのベース名を出力します。この例では c 拡張子が追加されて、入力ファイル foo.lex と同じディレクトリに C ソースファイルが提供されます。これらの変数について詳しくは、124 ページの *引数変数* を参照してください。

- 5** **【出力ファイル】** フィールドに、ビルドに関連して生成される出力ファイルを記述します。この例では、ツール Flex がソースファイルとヘッダファイルを 1 つずつ生成します。**【出力ファイル】** テキストボックスでこれら 2 つのファイルを表すテキストは以下のようになります。

```
$FILE_BPATH$.c  
$FILE_BPATH$.h
```

- 6** 外部ツールがビルド中に使用するファイルが他にもある場合は、それらのファイルをたとえば次のように **【追加入力ファイル】** フィールドに追加する必要があります。

```
$TOOLKIT_DIR$%inc%stdio.h
```

使用するファイルを追加する必要があるのは、依存ファイルが変更された場合、条件が変わるのでリビルドする必要があるためです。

- 7** **【OK】** をクリックします。
- 8** アプリケーションをビルドするには、**【プロジェクト】** > **【作成】** を選択します。

編集

この章では、IAR Embedded Workbench エディタの使用方法について説明します。最後のセクションでは、エディタのカスタマイズ方法とユーザが選択した外部エディタの使用方法について説明します。

IAR Embedded Workbench エディタの使用

統合されているテキストエディタには、複数ファイルを同時に編集する機能の他、最新エディタが通常提供している基本的な編集機能がすべて用意されています。また、キーワードのカラー表示（C/C++、アセンブラ、ユーザ定義）、ブロックインデント、ソースファイル内での関数間の移動などのソフトウェア開発用機能も装備しています。さらに、括弧のマッチングなど、C 言語のエレメントの認識にも対応しています。他にも、以下のような特長があります。

- DLIB ライブラリ関数のリファレンス情報を表示できる文脈依存ヘルプシステム
- テキストスタイルおよびカラーで C/C++ プログラム、アセンブラディレクティブの構文を区別して表示
- 複数ファイル検索などの強力な検索 / 置換コマンド
- エラーリストからコンテキストを直接表示
- マルチバイト文字のサポート
- 括弧のマッチング
- 自動完了およびインデント
- ブックマーク
- 各ウィンドウで無制限にアンドウ / リドゥ可能

ファイルの編集

エディタウィンドウでは、ソースコードの記述、表示、変更を行います。**[ファイル]** メニューからファイルを選択するか、**[ワークスペース]** ウィンドウでファイルをダブルクリックすることによって、1 つ以上のテキストファイルを開くことができます。複数のファイルを開いている場合、それらのファイルはタブグループとして編成されます。複数のエディタウィンドウを同時に開いておくことができます。

表示するファイルに対応するタブをクリックします。開いているファイルはすべて、エディタウィンドウの右上にあるドロップダウンメニューで選択することもできます。

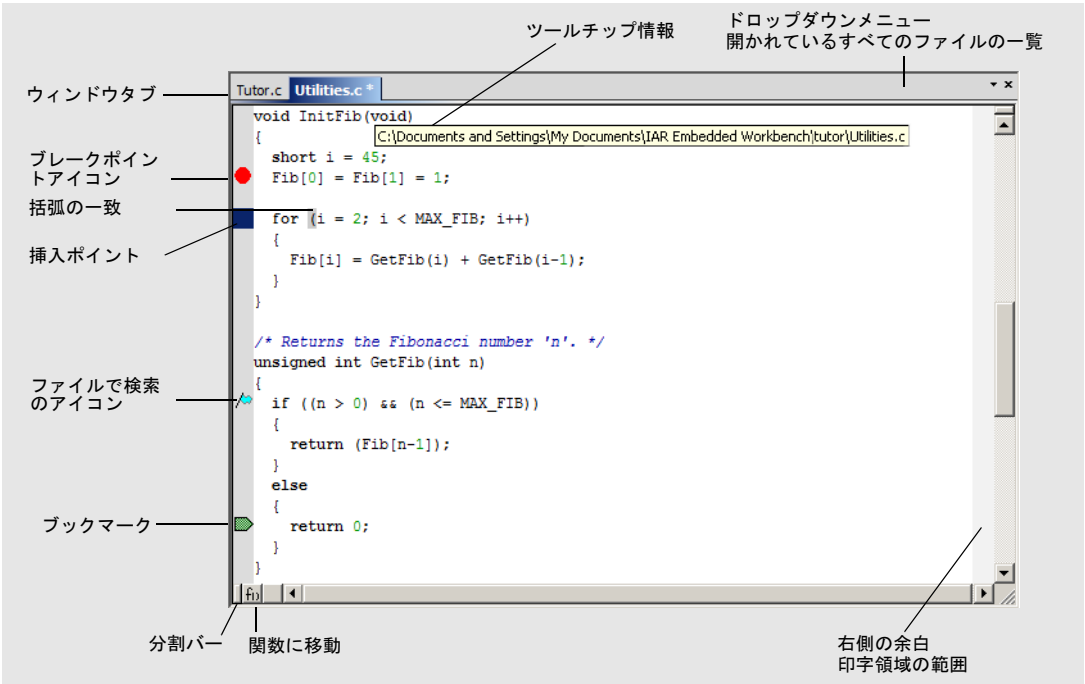


図 20: エディタウィンドウ

開いているソースファイルの名前がタブに表示されます。ファイルがリードオンリーの場合は、エディタウィンドウの左下隅に錠アイコンが表示されます。ファイルが最後に保存された後に変更された場合、たとえば「Utilities.c *」のように、タブのファイル名の末尾にアスタリスクが表示されます。

[ウィンドウ] メニューのコマンドを使用すると、エディタウィンドウをペインに分割できます。**[ウィンドウ]** メニューには、複数のエディタウィンドウを開くコマンドや、エディタウィンドウ間でファイルを移動するためのコマンドもあります。メニューの各コマンドのリファレンス情報については、157 ページの **[ウィンドウ]** メニューを参照してください。エディタウィンドウのリファレンス情報については、92 ページのエディタウィンドウを参照してください。



注： ソースファイルを出力する場合には、**[ツール] > [オプション] > [エディタ]** を選択し、**[行番号の表示]** オプションを有効にしておくと便利です。

DLIB ライブラリ関数のリファレンス情報へのアクセス

ライブラリ関数の構文を知る必要がある場合は、エディタウィンドウで関数名を選択して、F1 を押します。選択された関数のライブラリドキュメントがヘルプウィンドウに表示されます。

エディタコマンドとショートカットキーの使用とカスタマイズ

[編集] メニューには、エディタウィンドウで編集や検索を行うためのコマンドが用意されています。たとえば、無制限のアンドゥ/リドゥ（それぞれ、**[編集]** > **[アンドゥ]** コマンド、**[編集]** > **[リドゥ]** コマンド）などがあります。これらのコマンドの一部は、エディタウィンドウを右クリックして表示されるコンテキストメニューからも選択できます。各コマンドのリファレンス情報については、106 ページの **[編集]** メニューを参照してください。

以下の操作を行うエディタショートカットキーもあります。

- 挿入ポイントの移動
- テキストのスクロール
- テキストの選択

ショートカットキーの詳細については、98 ページの *エディタのショートカットキー操作のまとめ* を参照してください。

デフォルトのショートカットキーバインディングを変更するには、**[ツール]** > **[オプション]** を選択して、**[キーカスタマイズ]** タブをクリックします。詳細については、130 ページの **[キーカスタマイズ]** オプションを参照してください。

エディタウィンドウをペインに分割

エディタウィンドウを水平または垂直に複数のペインに分割して、同一ソースファイルの異なる部分を同時に表示したり、2 つの異なるペイン間でテキストを移動することができます。

ウィンドウを分割するには、適切な分割バーをダブルクリックするか、ウィンドウの中央までドラッグします。別の方法として、**[ウィンドウ]** > **[分割]** コマンドを使用してウィンドウをペインに分割することもできます。

1 つのペインに戻すには、分割バーをダブルクリックするか、スクロールバーの端までドラッグします。

テキストのドラッグアンドドロップ

エディタウィンドウ内またはエディタウィンドウ間でテキストを簡単に移動できます。テキストを選択して、移動先にドラッグします。

構文カラー表示

[ツール] > [オプション] > [エディタ] > [構文の強調表示] オプションを有効にすると、IAR Embedded Workbench エディタは、自動的に以下の構文を認識します。

- C と C++ のキーワード
- C と C++ のコメント
- アセンブラディレクティブとコメント
- プリプロセッサのディレクティブ
- 文字列

ソースコードは、部分ごとに異なるテキストスタイルで表示されます。

これらのスタイルを変更するには、[ツール] > [オプション] を選択して、[エディタ] > [色とフォント] オプションを使用します。その他の情報については、138 ページの [色とフォント] オプションを参照してください。

独自にキーワードセットを定義して、それらを自動的に構文カラー表示の対象にすることができます。

- 1 テキストファイルに、自動的に構文カラー表示の対象にするすべてのキーワードを記述します。各キーワードは、スペースや改行で区切ります。
- 2 [ツール] > [オプション] を選択して、[エディタ > セットアップファイル] を選択します。
- 3 [カスタムキーワードファイルの使用] オプションを選択して、新しく作成したテキストファイルを指定します。参照ボタンを使用して選択することもできます。
- 4 [エディタ] > [色とフォント] を選択して、[構文の色] リストから [ユーザキーワード] を選択します。フォント、色、タイプスタイルを指定します。その他の情報については、138 ページの [色とフォント] オプションを参照してください。
- 5 エディタウィンドウで、キーワードファイルに記述したキーワードを入力して、指定どおりにそのキーワードがカラー表示されていることを確認します。

自動テキストインデント

テキストエディタには、さまざまな種類のインデントがあります。アセンブラソースファイルと通常のテキストファイルは、エディタによって、行の先頭が前の行の先頭に一致するように自動インデントされます。複数の行をインデントする場合は、該当する行を選択して、Tab キーを押します。Shift+Tab キーを押して、選択した行をまとめて左に移動します。

C/C++ ソースファイルの場合、エディタは C/C++ ソースコードの構文に従って行をインデントします。インデントは以下のタイミングで実行されます。

- Enter キーを押したとき
 - {、}、:、# のいずれかの特殊文字が入力されたとき
 - 1 行または複数行を選択して **【編集】** > **【自動インデント】** を選択したとき
- インデントを有効/無効にするには、以下の手順を実行します。

- 1 **【ツール】** > **【オプション】** を選択して、**【エディタ】** を選択します。
- 2 **【自動インデント】** オプションを選択/選択解除します。

C/C++ の自動インデントをカスタマイズするには、**【設定】** ボタンをクリックします。

その他の情報については、134 ページの **【自動インデントの設定】** ダイアログボックスを参照してください。

中括弧と括弧の対応

括弧の近くに挿入ポイントがある場合、対応する括弧が淡い灰色で強調表示されます。

```
for( int i = 0; i < 10; i++)
{
}
```

図 21: エディタウィンドウの括弧の対応

挿入ポイントが括弧の近くにある間は、対応する括弧は強調表示されたままです。

挿入ポイントを含む中括弧で囲まれたテキストをすべて選択するには、**【編集】** > **【括弧のマッチング】** を選択します。その後は、**【括弧のマッチング】** を選択するたびに、選択される範囲が次の階層の中括弧で囲まれた範囲まで広がります。

注：括弧の対応の機能（自動検出、括弧で囲まれたテキストの選択）はどちらも、()、[]、{} に適用されます。

ステータス情報の表示

編集中は、ステータスバー（[表示] > [ステータスバー] を選択して表示）に、現在挿入ポイントがある行番号と列番号、Caps Lock、Num Lock、上書きステータスが表示されます。

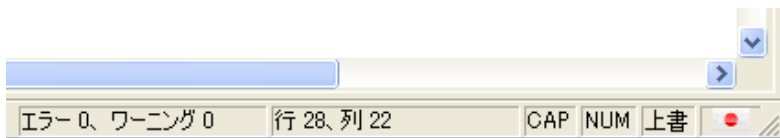


図 22: エディタウィンドウのステータスバー

コードテンプレートの使用と追加

コードテンプレートは、たとえば for ループや if 文のように、頻繁に使用されるソースコードシーケンスを簡単に挿入するための方法です。コードテンプレートは、通常のテキストファイルで定義します。デフォルトで、いくつかのサンプルテンプレートが提供されています。それ以外に、簡単に独自のコードテンプレートを追加できます。

コードテンプレートの有効化

デフォルトで、コードテンプレートは有効になっています。コードテンプレートの使用を有効/無効にするには、以下の手順を実行します。

- 1 [ツール] > [オプション] を選択します。
- 2 [エディタセットアップファイル] ページに移動します。
- 3 [コードテンプレートの使用] オプションを選択/選択解除します。
- 4 テキストフィールドで、使用するテンプレートとして、デフォルトファイルか独自に作成したテンプレートファイルを指定します。参照ボタンを使用して選択することもできます。

ソースコードへのコードテンプレートの挿入

ソースコードにコードテンプレートを挿入するには、テンプレートを挿入する場所に挿入ポイントを配置し、右クリックして表示されるメニューから**【テンプレートの挿入】** および適切なコードテンプレートを選択します。

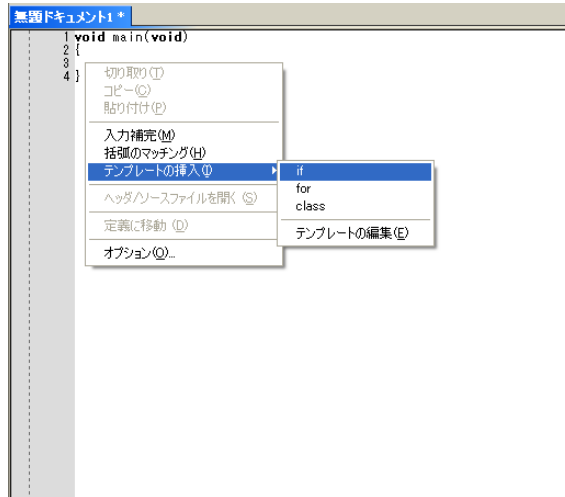


図 23: コードテンプレートの挿入

終値とカウンタ変数を必要とする `for` ループのように、選択したコードテンプレートがフィールドへの入力が必要な場合は、入力ダイアログボックスが表示されます。

独自のコードテンプレートの追加

ソースコードテンプレートは、通常のテキストファイルで定義します。オリジナルテンプレートファイル `CodeTemplates.txt` は、`common\config` インストールディレクトリにあります。初めて IAR Embedded Workbench を使用するときに、オリジナルテンプレートファイルがローカル設定用ディレクトリにコピーされます。コードテンプレートが有効な場合は、このファイルがデフォルトで使用されます。独自のテンプレートファイルを使用する手順については、74 ページの *コードテンプレートの有効化* を参照してください。

テンプレートファイルを開いて、独自のコードテンプレートを定義するには、**【編集】 > 【コードテンプレート】 > 【テンプレートの編集】** を選択します。

テンプレートを定義するための構文は、デフォルトテンプレートファイルに記述されています。

コードテンプレートファイルの正しい言語バージョンの選択

IAR Embedded Workbench IDE を初めて起動したとき、言語バージョンの選択が求められます。これが該当するのは、英語以外の言語が利用可能な IDE を使用している場合のみです。

言語を選択すると、デフォルトのコードテンプレートファイルの対応言語バージョンが、現在の Windows ユーザの Application Data¥IAR Embedded Workbench サブディレクトリに作成されます（たとえば、英語の場合は `CodeTemplates.ENU.txt`、日本語の場合は `CodeTemplates.JPN.txt`）。後から IDE の言語バージョンを変更した場合、デフォルトのコードテンプレートファイルは自動的に変更されません。

コードテンプレートを変更するには、以下のように操作してください。

- 1 [ツール] > [オプション] > [IDE オプション] > [エディタ] > [セットアップファイル] を選択します。
- 2 [コードテンプレートの使用] オプションの参照ボタンをクリックし、異なるテンプレートファイルを選択します。

参照したディレクトリに目的のコードテンプレートファイルがない場合には、必ず以下のように操作してください。
- 3 そのファイル名を [コードテンプレートの使用] テキストボックスから削除します。
- 4 [コードテンプレートの使用] オプションの選択を解除し、[OK] をクリックします。
- 5 IAR Embedded Workbench IDE を再起動します。
- 6 再度、[ツール] > [オプション] > [IDE オプション] > [エディタ] > [セットアップファイル] を選択します。

今度は、選択した言語のバージョンの IDE に対応したデフォルトのコードテンプレートファイルが [コードテンプレートの使用] テキストボックスに表示されるはずです。チェックボックスをオンにすると、テンプレートが有効になります。

ファイル間のナビゲート

エディタには、ファイル内やファイル間を簡単にナビゲートするための機能が用意されています。

- ソースファイルとヘッダファイル間の切替え
挿入ポイントが `#include` 行にある場合、コンテキストメニューで [header.h 開く] コマンドを選択して、ヘッダファイルをエディタウィンドウで開くことができます。また、コマンド [ヘッダ/ソースファイルを

開く]を選択すると、現在のファイルに対応するヘッダファイルやソースファイルを開いたり、すでに開いている場合はアクティブにしたりできます。このコマンドは、挿入ポイントが `#include` 行の近くにあるときに選択できます。

- 関数ナビゲーション



エディタウィンドウの左下隅にある**【関数に移動】** ボタンをクリックすると、ウィンドウに表示されているソース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。リストで関数をダブルクリックすると、その関数の位置に直接移動できます。

- ブックマークの追加

【編集】 > 【移動】 > 【ブックマークの切替え】 コマンドを使用すると、ブックマークを追加 / 削除できます。ブックマークされた位置間を移動するには、**【編集】 > 【移動】 > 【ブックマークへ移動】** を選択します。

検索

エディタには、以下に示すさまざまな標準的検索機能が用意されています。

- **【クイックサーチ】** テキストボックス
- **【検索】** ダイアログボックス
- **【置換】** ダイアログボックス
- **【ファイルから選択】** ダイアログボックス
- **【インクリメンタル検索】** ダイアログボックス

ツールバーの**【クイックサーチ】** テキストボックスを使用するには、次の手順に従います。

- 1 検索する文字を入力して **Enter** キーを押します。
- 2 **Esc** キーを押すと検索をキャンセルします。アクティブなエディタウィンドウでテキストを検索する場合は、この方法が最も簡単です。

【検索】、**【置換】**、**【ファイルで検索】**、**【インクリメンタル検索】** の各機能を使用するには、次の手順に従います。

- 1 検索コマンドを使用する前に、**【ツール】 > 【オプション】 > 【オプション】** を選択して、**【ブックマークの表示】** オプションが選択されているか確認します。
- 2 **【編集】** メニューから適切な検索を選択します。各検索機能のリファレンス情報については、106 ページの **【編集】** メニューを参照してください。
- 3 左端に表示される青い旗のアイコンを削除するには、**【ファイルで検索】** ウィンドウで右クリックしてコンテキストメニューから **【すべてをクリア】** を選択します。

エディタ環境のカスタマイズ

IDE エディタは、[IDE オプション] の [エディタ] ページと [カラーとフォントを編集する] ページで構成できます。これらのページにアクセスするには、[ツール] > [オプション] を選択します。

これらのページの詳細については、128 ページの [ツール] メニューを参照してください。

外部エディタの連携

[外部エディタ] オプション ([ツール] > [オプション] > [エディタ] を選択して表示) では、任意の外部エディタを指定できます。

注：C-SPY を使用したデバッグ中には、現在のデバッグ状態の表示に外部エディタは使用されません。内蔵のエディタが使用されます。

任意の外部エディタを指定するには、以下の手順を実行してください。

- 1 [外部エディタを使用する] オプションを選択します。
- 2 外部エディタを呼び出すには、[種類] ドロップダウンメニューで以下の2つの方法のどちらかを選択します。

[コマンドライン] は、外部エディタを呼び出して、コマンドラインパラメータを渡します。

[DDE] は、DDE (Windows Dynamic Data Exchange: Windows 動的データ交換) を使用して、外部エディタを呼び出します。

- 3 コマンドラインを使用する場合は、エディタに渡すコマンドライン、すなわちエディタの名前とそのパスを指定します。以下に例を示します。

C:\Windows\NOTEPAD.EXE.

引数を外部エディタに送信するには、[引数] フィールドに引数を入力します。たとえば、「\$FILE_PATH\$」と入力すると、エディタが起動されて、アクティブファイルが開きます（エディタ、プロジェクト、[メッセージ] ウィンドウ）。

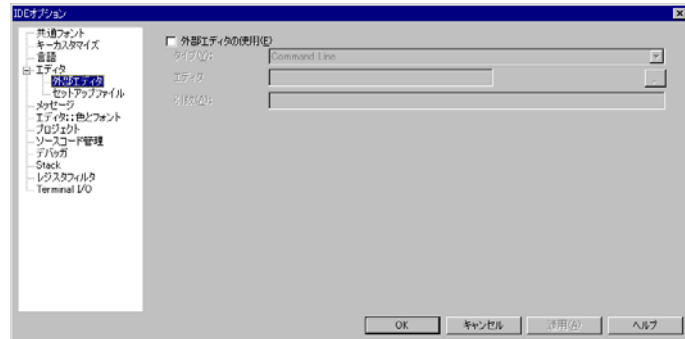


図 24: 外部コマンドラインエディタの指定

注：[レジスタフィルタ] と [ターミナル I/O] のオプションは、C-SPY デバッガの実行中にのみ使用できます。

- 4 DDE を使用する場合、[サービス] フィールドでエディタの DDE サービス名を指定します。[コマンド] フィールドで、エディタに送信するコマンドシーケンスを表す文字列を指定します。

サービス名とコマンド文字列は、使用する外部エディタに応じて指定します。外部エディタのドキュメントを参照して、適切に設定してください。

コマンド文字列は、以下の形式で入力する必要があります。

```
DDE-Topic CommandString1
DDE-Topic CommandString2
```

以下に例を示します。この例は、Codewright® に適用されます。

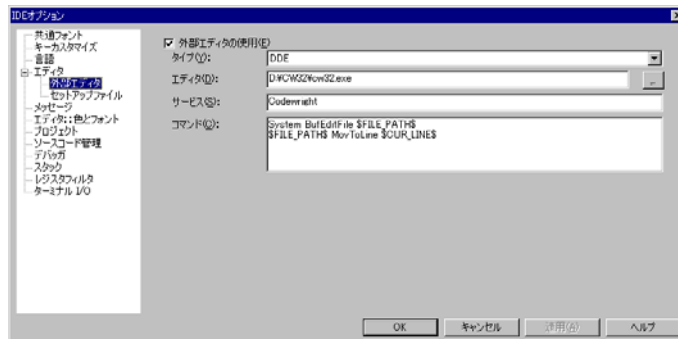


図 25: 外部エディタの DDE 設定

この例で指定したコマンド文字列で、外部エディタが開いて、専用ファイルがアクティブになります。カーソルは、たとえばファイル内の文字列を検索している場合や [メッセージ] ウィンドウでエラーメッセージをダブルクリックした場合のように、ファイルを開いたコンテキストの定義に従って、現在の行に置かれます。

5 [OK] をクリックします。

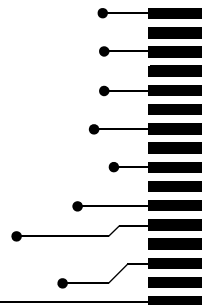
[ワークスペース] ウィンドウでファイルをダブルクリックすると、そのファイルは外部エディタで開かれます。

引数に変数を使用できます。引数変数の詳細については、124 ページの *引数変数* を参照してください。

パート 2. リファレンス情報

『IDE プロジェクト管理およびビルドガイド』のこのパートは以下の章で構成されています。

- インストールファイル
- IAR Embedded Workbench IDE リファレンス
- 一般オプション
- コンパイラオプション
- アセンブラオプション
- 出力コンバータオプション
- カスタムビルドオプション
- ビルドアクションオプション
- リンカのオプション
- ライブラリビルダオプション





インストールファイル

この章では、インストールの際に作成されたディレクトリと、使用されるファイルタイプについて説明します。

ディレクトリ構成

インストール手順を実行すると、IAR システムズの開発ツールで使用される各種ファイルを含む複数のディレクトリが作成されます。以下では、各ディレクトリにデフォルトで含まれるファイルについて説明します。

ルートディレクトリ

デフォルトのインストール手順で作成されるルートディレクトリは、`x:\Program Files\IAR Systems\Embedded Workbench 6.n` です。ここで、`x` は Microsoft Windows がインストールされているドライブ、`6.n` は IDE のバージョンを示します。

ARM ディレクトリ

arm ディレクトリには、製品固有のサブディレクトリがすべて含まれています。

ディレクトリ	説明
arm\bin	arm\bin サブディレクトリには、コンパイラ、アセンブラ、ARM 固有のコンポーネント。
arm\CMSIS	arm\CMSIS サブディレクトリには、CMSIS および CMSIS DSP ライブラリファイルとドキュメントが含まれます。
arm\config	arm\config サブディレクトリには、開発環境やプロジェクトの設定に使用する以下のようなファイルが含まれています。 <ul style="list-style-type: none">• リンカ構成ファイル (*.icf)• C-SPY デバイス記述ファイル (*.ddf)• デバイス選択ファイル (*.i79, *.menu)• さまざまなデバイス用のフラッシュローダアプリケーション (*.out)• 構文カラー表示用設定ファイル (*.cfg)• アプリケーション/ライブラリプロジェクト用のプロジェクトテンプレート (*.ewp)、ライブラリプロジェクト用のライブラリ設定ファイル

表 4: ARM ディレクトリ

ディレクトリ	説明
arm¥doc	arm¥doc サブディレクトリには、ARM ツールの最新更新情報を記載したリリースノートが含まれています。これらのファイルすべての内容を確認することをお勧めします。また、本ユーザガイド、ARM リファレンスガイドのオンライン版（ハイパーテキスト PDF フォーマット）、オンラインヘルプファイル (*.chm) も含まれています。
arm¥drivers	arm¥drivers サブディレクトリには、C-SPY ドライバで必要な低レベルのデバイスドライバ（特に USB ドライブ）が格納されています。
arm¥examples	arm¥examples サブディレクトリには、サンプルプロジェクトに関連するファイルが含まれており、[インフォメーションセンタ] ダイアログボックスから開くことができます。
arm¥inc	arm¥inc サブディレクトリには、標準 C/C++ ライブラリのヘッダファイルなどのファイルが含まれています。また、特殊機能レジスタ (SFR) を定義するヘッダファイルも含まれています。これらのファイルは、コンパイラとアセンブラの両方で使用されます。
arm¥lib	arm¥lib サブディレクトリには、コンパイラが使用するビルド済みライブラリおよび対応するライブラリ設定ファイルが含まれています。
arm¥plugins	arm¥plugins サブディレクトリには、プラグインモジュールとしてロード可能なコンポーネント用の実行可能ファイルおよび説明ファイルが含まれています。
arm¥src	arm¥src サブディレクトリには、設定可能なライブラリ関数のソースファイルが含まれています。また、このディレクトリには、ライブラリソースコードおよび ELF ユーティリティのソースコードも含まれています。
arm¥tutor	arm¥tutor サブディレクトリには、インフォメーションセンタのチュートリアルで使用されるファイルが含まれています。

表 4: ARM ディレクトリ (続き)

COMMON ディレクトリ

common ディレクトリには、すべての IAR Embedded Workbench 製品で共有するコンポーネント用のサブディレクトリが含まれています。

ディレクトリ	説明
common¥bin	common¥bin サブディレクトリには、エディタ、グラフィカルユーザインタフェースコンポーネントなど、すべての IAR Embedded Workbench 製品に共通のコンポーネント用実行可能ファイルが含まれています。IDE 用の実行可能ファイルもここに含まれています。
common¥config	common¥config サブディレクトリには、IDE で開発環境の設定に使用されるファイルが含まれています。
common¥doc	common¥doc サブディレクトリには、すべての IAR Embedded Workbench 製品に共通のコンポーネントに関する最近の追加情報とリリースノートが含まれます。これらのファイルの内容を確認することをお勧めします。このディレクトリには、インストールおよびライセンスに関するドキュメント、および IAR Embedded Workbench を使用した利用ガイドも含まれます。
common¥plugins	common¥plugins サブディレクトリには、プラグインモジュールとしてロード可能なコンポーネント用の実行可能ファイルや記述ファイル（コードカバレッジやプロファイリング用のサンプルモジュール）が格納されています。

表 5: common ディレクトリ

INSTALL-INFO ディレクトリ

install-info ディレクトリには、インストールされている製品コンポーネントのメタデータ（バージョン番号、名前など）が含まれています。これらのファイルは変更しないでください。

ファイルタイプ

ARM バージョンの IAR システムズの開発ツールは、以下のデフォルトのファイル名拡張子を使用して、製品ファイルおよびその他の対応ファイルタイプを識別します。

拡張子	ファイルタイプ	出力元	入力先
a	ライブラリ	iarchive	ILINK
asm	アセンブラソースコード	テキストエディタ	アセンブラ
bat	Windows コマンドバッチファイル	C-SPY	ウィンドウ

表 6: ファイルタイプ

拡張子	ファイルタイプ	出力元	入力先
board	フラッシュローダの設定	テキストエディタ	C-SPY
c	C ソースコード	テキストエディタ	コンパイラ
cfg	構文カラー表示設定	テキストエディタ	IDE
chm	オンラインヘルプシステムファイル	--	IDE
cpp	C++ ソースコード	テキストエディタ	コンパイラ
dat	STL コンテナのフォーマット用マクロ	IDE	IDE
dbgd	デバッガのデスクトップ設定	C-SPY	C-SPY
ddf	デバイス記述ファイル	テキストエディタ	C-SPY
dep	依存関係情報	IDE	IDE
dni	デバッガ初期化ファイル	C-SPY	C-SPY
ewd	C-SPY のプロジェクト設定	IDE	IDE
ewp	IAR Embedded Workbench プロジェクト (現行バージョン)	IDE	IDE
ewplugin	プラグインモジュール用 IDE 記述ファイル	--	IDE
eww	ワークスペースファイル	IDE	IDE
flash	フラッシュローダの設定	テキストエディタ	C-SPY
fmt	[ローカル] ウィンドウ、 [ウォッチ] ウィンドウでの 表示フォーマット設定	IDE	IDE
h	C/C++、アセンブラのヘッダソース	テキストエディタ	コンパイラ、アセンブラの #include
helpfiles	[ヘルプ] メニュー構成ファイル	テキストエディタ	IDE
html、htm	HTML ドキュメント	テキストエディタ	IDE
i	プリプロセス済みソース	コンパイラ	コンパイラ
i79	デバイス選択ファイル	テキストエディタ	IDE
icf	リンカ設定ファイル	テキストエディタ	ILINK リンカ
inc	アセンブラのヘッダソース	テキストエディタ	アセンブラの #include
ini	プロジェクト設定	IDE	-
ログ	ログ情報	IDE	-

表 6: ファイルタイプ (続き)

拡張子	ファイルタイプ	出力元	入力先
lst	リスト出力	コンパイラ、アセンブラ	-
mac	C-SPY マクロ定義	テキストエディタ	C-SPY
メニュー	デバイス選択ファイル	テキストエディタ	IDE
o	オブジェクトモジュール	コンパイラ、アセンブラ	ILINK
out	ターゲットアプリケーション	ILINK	EPROM、C-SPY など
out	ターゲットアプリケーション (デバッグ情報を含む)	ILINK	C-SPY、その他のシンボリックデバッガ
pbd	ソースブラウザ情報	IDE	IDE
pbi	ソースブラウザ情報	IDE	IDE
pew	IAR Embedded Workbench プロジェクト (旧プロジェクトフォーマット)	IDE	IDE
prj	IAR Embedded Workbench プロジェクト (旧プロジェクトフォーマット)	IDE	IDE
s	ARM アセンブラソースコード	テキストエディタ	アセンブラ
svd	CMSIS システムビューの説明	--	C-SPY
vsp	visualSTATE プロジェクトファイル	IAR visualSTATE Designer	IAR visualSTATE Designer および IAR Embedded Workbench IDE
wsdt	ワークスペースのデスクトップ設定	IDE	IDE
xcl	拡張コマンドライン	テキストエディタ	アセンブラ、コンパイラ、リンカ

表 6: ファイルタイプ (続き)

IDE を実行すると、いくつかのファイルが作成され、プロジェクトディレクトリの専用ディレクトリに格納されます。デフォルトでは \$PROJ_DIR\$\Debug、\$PROJ_DIR\$\Release、\$PROJ_DIR\$\settings、ファイル *.dep がインストールディレクトリに格納されます。これらのディレクトリやファイルはどれも IDE の実行には影響を与えないため、必要に応じて問題なくこれらのファイルを削除できます。

デフォルトでないファイル名拡張子のファイル



IDE では、**「ファイル名の拡張子」** ダイアログボックス（**「ツール」** メニューから利用可能）を使用して、認識ファイル名の拡張子の数を増やすことができます。ツールチェーンの特定のツールにファイル名の拡張子を関連付けることもできます。152 ページの **「ファイル名拡張子」** ダイアログボックスを参照してください。



デフォルトのファイル名拡張子をコマンドラインからオーバーライドするには、ファイル名の指定時に拡張子を明示的に指定します。

IAR Embedded Workbench IDE リファレンス

この章では、IDE のウィンドウ、メニュー、メニューコマンド、対応するコンポーネントに関するリファレンス情報を提供します。この章は、次の項目で構成されています。

- (ウィンドウ 89 ページ)
- (メニュー 103 ページ)

IDE は、モジュール化構造のアプリケーションです。使用可能なメニューは、インストールされているコンポーネントによって異なります。

ウィンドウ

使用可能なウィンドウは、以下のとおりです。

- [IAR Embedded Workbench IDE] ウィンドウ
- [ワークスペース] ウィンドウ
- エディタウィンドウ
- [ソースブラウザ] ウィンドウ
- メッセージウィンドウ

また、デバッガを起動すると、C-SPY 専用のウィンドウが使用可能になります。これらのウィンドウのリファレンス情報については、「ARM® 用 C-SPY® デバッガガイド」を参照してください。

[IAR Embedded Workbench IDE] ウィンドウ

IDE のメインウィンドウは、IDE を起動すると表示されます。

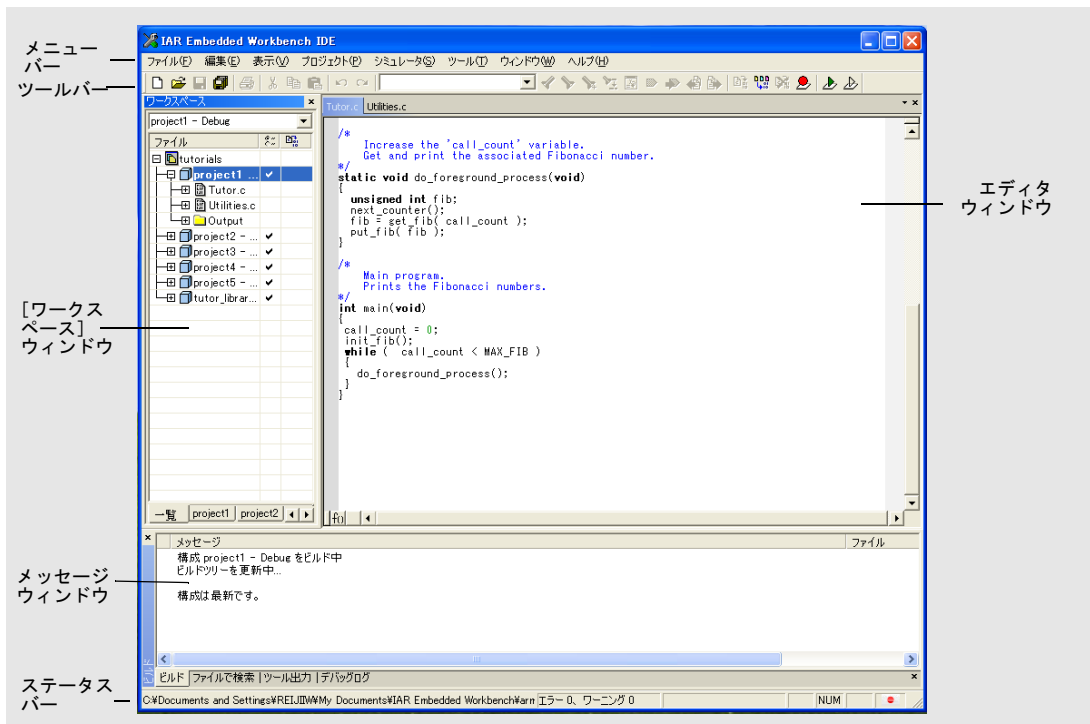


図 26: [IAR Embedded Workbench IDE] ウィンドウ

この図は、ウィンドウとそのさまざまなコンポーネントを示します。使用するプラグインモジュールによって、ウィンドウの表示が異なる場合があります。

メニューバー

メニューバーには以下が含まれます。

- ファイル** ソースファイルおよびプロジェクトファイルのオープン、保存、出力、IDE の終了を実行するためのコマンド。
- 編集** エディタウィンドウでの編集 / 検索用コマンドと、C-SPY でのブレークポイントの設定 / 解除用コマンド。
- 表示** ウィンドウを開いたり、表示するツールバーを制御するためのコマンド。

プロジェクト プロジェクトへのファイルの追加、グループの作成、現在のプロジェクトでの IAR システムズツールの実行のためのコマンド。

ツール ユーザーが設定可能なメニューで、IDE と使用するツールをこれに追加できます。

ウィンドウ IDE ウィンドウの操作や画面上での配置変更のコマンド。

ヘルプ IDE に関するヘルプを提供するコマンド。

各メニューのリファレンス情報については、103 ページのメニューを参照してください。

ツールバー

IDE ツールバー（[表示] メニューから表示）には、IDE のメニューで最も便利なコマンドを実行するためのボタンと、文字列を入力してすばやく検索するためのテキストボックスがあります。

マウスでボタンをポイントすると、そのボタンの説明が表示されます。コマンドが使用できない場合は、対応するツールバーボタンは灰色表示され、クリックできないようになっています。

下図に、各ツールバーボタンに対応するメニューコマンドを示します。

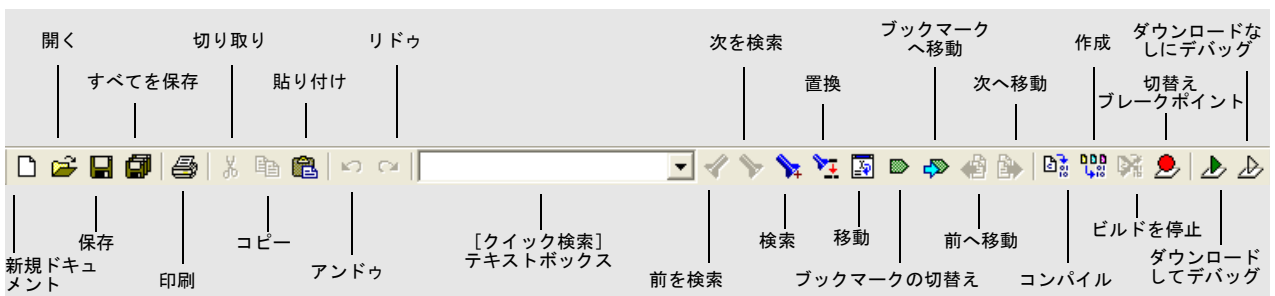


図 27: IDE ツールバー



注: C-SPY を起動すると、[ダウンロードしてデバッグ] ボタンは [作成してデバッグ] ボタンになり、[ダウンロードなしにデバッグ] は [デバッグを再起動] ボタンになります。



ステータスバー

ウィンドウ下部のステータスバーには、ビルド中に発生したエラーおよびワーニングの数、編集ウィンドウでの挿入位置、修飾キーの状態が表示されます。ステータスバーは、[表示] メニューから有効にできます。

編集中は、ステータスバーには挿入ポイントがある現在の行 / 列番号と、Caps Lock、Num Lock、上書きの各状態が表示されます。隅に表示されるフラグが、使用中の言語バージョンを示します。次回 IDE を起動するときに言語を変更するには、このフラグをクリックします。



図 28: [IAR Embedded Workbench IDE] ウィンドウのステータスバー

エディタウィンドウ

エディタウィンドウは、IDE でテキストファイルを開いたり作成すると表示されます。

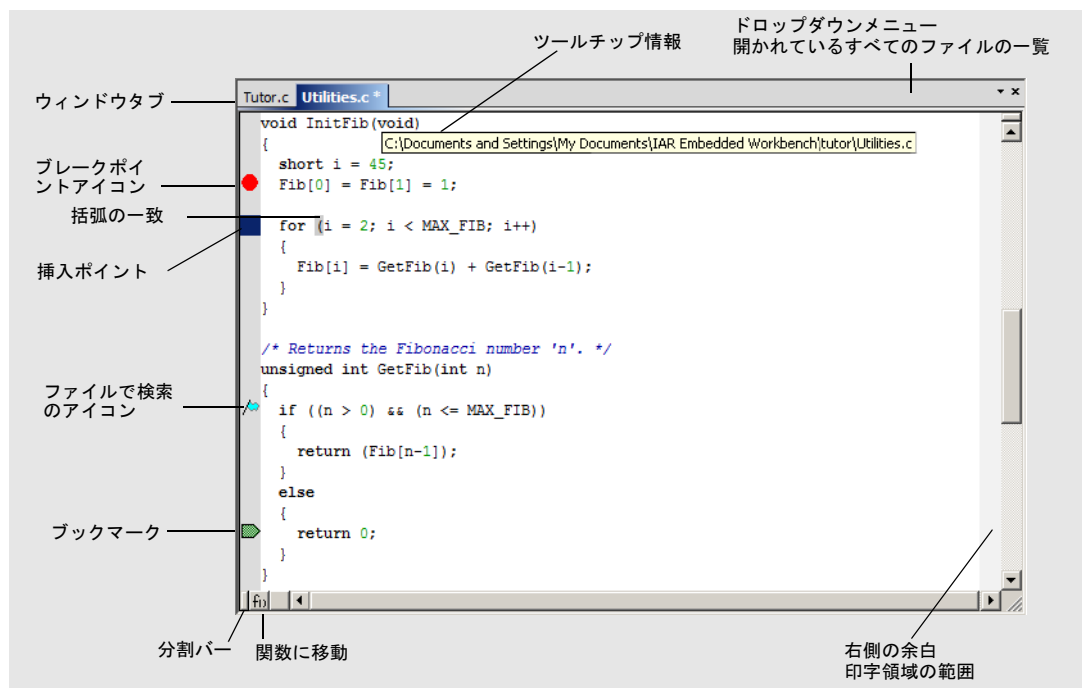


図 29: エディタウィンドウ

ソースコードファイルと HTML ファイルがエディタウィンドウに表示されます。開いている HTML 文書では、HTML ファイルへのハイパーリンクは通常のウェブブラウジングと同じように機能します。eww ワークスペースファイルへのリンクは、IDE でワークスペースを開くことや、現在開いているワークスペースおよび HTML ドキュメントを閉じることができます。

同時に複数のエディタウィンドウを表示できます。[ウィンドウ] メニューには、複数のエディタウィンドウを開くコマンドや、エディタウィンドウ間でファイルを移動するためのコマンドがあります。

エディタウィンドウは常にドッキングされていて、サイズと位置は他の開いているウィンドウに応じて変化します。ファイルがリードオンリーの場合は、エディタウィンドウの左下隅に錠アイコンが表示されます。

エディタの使用について詳しくは、106 ページの [編集] メニューとを参照してください。

ソースファイルパス

IDE は、ソースファイルの相対パスを一部サポートします。

ソースファイルがプロジェクトファイルディレクトリかプロジェクトファイルディレクトリ内のサブディレクトリにある場合、IDE はプロジェクトファイルとの相対パスを使用してソースファイルにアクセスします。

ウィンドウタブ

開いているファイルの名前がタブに表示されます。最後に保存した後にファイルが修正されている場合は、Utilities.c * のように、タブ上のファイル名の後にアスタリスクが表示されます。

エディタウィンドウのタブを右クリックするとコンテキストメニューが表示されます。

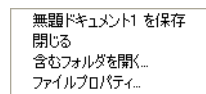


図 30: エディタウィンドウタブのコンテキストメニュー

以下のコマンドがあります。

ファイルの保存	ファイルを保存します。
閉じる	ファイルを閉じます。
含むフォルダを開く	選択したファイルが存在するディレクトリを表示するファイルエクスプローラを開きます。

ファイルプロパティ 標準のファイルプロパティダイアログボックスを表示します。

エディタウィンドウの右上にあるドロップダウンメニューから、開いているすべてのファイルを選択できます。

分割バー

エディタウィンドウを縦／横に分割するには、[ウィンドウ] > [分割] コマンドを選択するか、分割バーを使用します。

関数に移動



エディタウィンドウの左下隅にある **関数に移動** ボタンをクリックして、[C/C++] エディタウィンドウで使用されているすべての関数を一覧表示することができます。



図 31: [関数に移動] ウィンドウ

[エディタ] ウィンドウに表示する関数をダブルクリックします。

コンテキストメニュー

以下のコンテキストメニューがあります。

切り取り (⌘) コピー (⌘) 貼り付け (⌘)
入力補完 (M) 括弧のマッチング (H) テンプレートの挿入 (Q) ▶
ヘッダ/ソースファイルを開く (S)
callCount の定義に移動 (D)
トレースを検索
ブレークポイントの切り替え (A) (Code) ブレークポイントの切り替え (A) (Log) ブレークポイントの切り替え (A) (Trace Start) ブレークポイントの切り替え (A) (Trace Stop) ブレークポイントの有効化/無効化 (N) 'callCount'のDataブレークポイントを設定
次の文の設定
クイックウォッチ (Q) ウォッチへ追加 (W)
PCへ移動 (V) カーソルまで実行 (R)
オプション (O)...

図 32: エディタウィンドウのコンテキストメニュー

このメニューの内容は、デバッガが起動中かどうか、および使用している C-SPY ドライバによって異なります。通常は、このメニューで他のブレークポイントタイプが使用できることがあります。使用可能なブレークポイントについては、『ARM® 用 C-SPY® デバッガガイド』を参照してください。

以下のコマンドがあります。

切り取り、コピー、 Windows 標準のコマンド。
貼り付け

入力補完 入力内容に応じて、エディタドキュメントの他の部分の内容から入力語を推測して補完します。

括弧のマッチング 挿入ポイントの直近の括弧内のテキストをすべて選択します。すでに選択されている場合は、その外側の次の括弧まで選択範囲を拡大します。外側に括弧がない場合は、ビープ音を再生します。

テンプレートの挿入	挿入ポイントの位置に挿入するコードテンプレートを選択できるリストを、エディタウィンドウで表示します。選択したコードテンプレートでフィールドへの入力が必要な場合は、 [テンプレート] ダイアログボックスが表示されます。このダイアログボックスについては、115 ページの [テンプレート] ダイアログボックスを参照してください。コードテンプレートの使用方法については、74 ページの コードテンプレートの使用と追加 を参照してください。
"header.h" を開く	"header.h" という名前のヘッダファイルをエディタウィンドウで開きます。このメニューコマンドは、コンテキストメニューを表示したときに挿入ポイントが #include 行にある場合にだけ使用できます。
ヘッダ / ソースファイル ファイル	現在のファイルから、対応するヘッダファイルかソースファイルに移動します。コマンド実行時に対象ファイルが開かれていない場合は、そのファイルを開きます。このメニューコマンドは、コンテキストメニューを表示したときに挿入ポイントが #include 行を除く任意の行にある場合に使用できます。このコマンドは、 [ファイル] > [開く] メニューから選択することもできます。
シンボル定義に移動	挿入ポイントのある箇所のシンボルの定義を表示します。
チェックイン	ソースコード管理用コマンド (53 ページの SCC のバージョン管理システムメニュー を参照)。これらのコマンドは、エディタウィンドウで表示中のソースファイルが SCC 管理対象である場合にだけ使用できます。また、ファイルが現在のプロジェクトに含まれている必要があります。
チェックアウト	
チェックアウトを元に戻す	
ブレークポイントの切替え (コード)	ソースウィンドウで、カーソルを含む、または直近の文か命令で、コードブレークポイントを設定 / 解除します。コードブレークポイントについては、『 ARM® 用 C-SPY® デバッグガイド 』を参照してください。
ブレークポイントの切替え (ログ)	ソースウィンドウで、カーソルを含む、または直近の文か命令で、ログブレークポイントを設定 / 解除します。ログブレークポイントについては、『 ARM® 用 C-SPY® デバッグガイド 』を参照してください。

ブレークポイントの切替え (トレース開始)	トレース開始ブレークポイントを切替えます。ブレークポイントがトリガされると、トレースデータの収集が始まります。トレース開始ブレークポイントについては、『 <i>ARM® 用 C-SPY® デバッグガイド</i> 』を参照してください。このメニューコマンドは、使用している C-SPY ドライバでトレースがサポートされている場合にのみ使用できます。
ブレークポイントの切替え (トレース停止)	トレース停止ブレークポイントを切替えます。ブレークポイントがトリガされると、トレースデータの収集が停止します。トレース停止ブレークポイントについては、『 <i>ARM® 用 C-SPY® デバッグガイド</i> 』を参照してください。このメニューコマンドは、使用している C-SPY ドライバでトレースがサポートされている場合にのみ使用できます。
ブレークポイントの有効化 / 無効化	ブレークポイントの有効 (実際には削除せず、後で再度使用できる状態にする) と有効を切り替えます。
データブレークポイントの設定 (変数用)	静的記憶寿命変数のデータブレークポイントを切り替えます。使用している C-SPY ドライバでサポートされていることが必要です。
トレースを検索	指定の場所 (ソースコードの挿入ポイントの位置) に該当する箇所があるか [トレース] ウィンドウの内容を検索し、結果を [トレースを検索] ウィンドウに表示します。このメニューコマンドでは、使用する C-SPY ドライバでトレースがサポートされている必要があります (『 <i>ARM® 用 C-SPY® デバッグガイド</i> 』を参照)。
ブレークポイントの編集	[ブレークポイントの編集] ダイアログボックスが表示され、ソースコード行の挿入ポイントがある場所で使用可能なブレークポイントを編集できます。複数のブレークポイントが行にある場合、使用可能なすべてのブレークポイントの一覧を示すサブメニューがその行に表示されます。
次の実行文の設定	コードを実行せずに、選択した文か命令の位置に PC を設定します。このコマンドは慎重に使用してください。このコマンドは、デバッグ使用時にだけ使用できます。
クイックウォッチ	[クイックウォッチ] ウィンドウを表示します (『 <i>ARM® 用 C-SPY® デバッグガイド</i> 』を参照)。このコマンドは、デバッグ使用時にだけ使用できます。

ウォッチへ追加	選択したシンボルを「ウォッチ」ウィンドウに追加します。このコマンドは、デバッグ使用時にだけ使用できます。
PC へ移動	挿入ポイントを、エディタウィンドウで現在の PC 位置に移動します。このコマンドは、デバッグ使用時にだけ使用できます。
カーソルまで実行	現在の文 / 命令から、選択した文 / 命令までコードを実行します。このコマンドは、デバッグ使用時にだけ使用できます。
オプション	「IDE オプション」ダイアログボックスを表示します (128 ページの 「ツール」メニューを参照)。

エディタのショートカットキー操作のまとめ

下表に、エディタのショートカットキーをまとめています。

挿入ポイントの移動

挿入ポイントの移動操作	キー
左に 1 文字分移動	左矢印
右に 1 文字分移動	右矢印
左に 1 語分移動	Ctrl+ 左矢印
右に 1 語分移動	Ctrl+ 右矢印
上に 1 行分移動	上向きの矢印
下に 1 行分移動	下向きの矢印
行の先頭まで	Home
行の最後まで	End
ファイルの先頭行に移動	Ctrl+Home
ファイルの最終行に移動	Ctrl+End

表 7: エディタで挿入ポイントを移動するキーボードコマンド

テキストのスクロール

スクロールの操作	キー
上に 1 行	Ctrl+ 上矢印
下に 1 行	Ctrl+ 下矢印
上に 1 ページ	Page Up
下に 1 ページ	Page Down

表 8: エディタでのスクロール用キーボードコマンド

テキストの選択

選択の操作	キー
左側の文字	Shift+ 左矢印
右側の文字	Shift+ 右矢印
左側の 語	Shift+Ctrl+ 左矢印
右側の 語	Shift+Ctrl+ 右矢印
前の行の同一位置まで	Shift+ 上矢印
次の行の同一位置まで	Shift+ 下矢印
行の先頭まで	Shift+Home
行の最後まで	Shift+End
上に 画面分選択	Shift+Page Up
下に 画面分選択	Shift+Page Down
ファイルの先頭まで	Shift+Ctrl+Home
ファイルの最後まで	Shift+Ctrl+End

表 9: エディタでのテキスト選択用キーボードコマンド

[ビルド] ウィンドウ

[ビルド] ウィンドウは、[表示] > [メッセージ] を選択すれば使用できます。

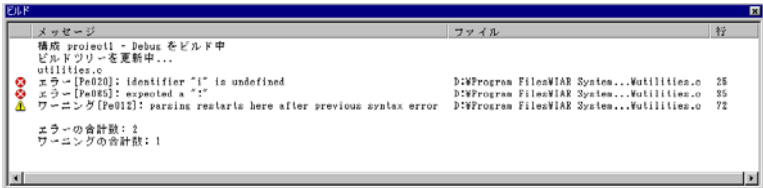


図 33: [ビルド] ウィンドウ (メッセージウィンドウ)

[ビルド] ウィンドウには、ビルド構成をビルドする際に生成されたメッセージが表示されます。デフォルトでは、このウィンドウは他のメッセージウィンドウとグループ化されて表示されます (89 ページの ウィンドウを参照)。
[ビルド] ウィンドウでメッセージをダブルクリックすると、該当ファイルが編集用に開かれ、挿入ポイントが正しい箇所に表示されます。

コンテキストメニュー

以下のコンテキストメニューがあります。

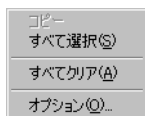


図 34: [ビルド] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

コピー	ウィンドウの内容をコピーします。
すべて選択	ウィンドウの内容を選択します。
すべてクリア	ウィンドウの内容を削除します。
オプション	[IDE オプション] ダイアログボックスの [メッセージ] ページが開きます。このページで、メッセージ関連オプションを設定することができます (139 ページの [メッセージ] オプションを参照)。

[ファイルで検索] ウィンドウ

[ファイルで検索] ウィンドウは、[表示] > [メッセージ] を選択すれば使用できます。

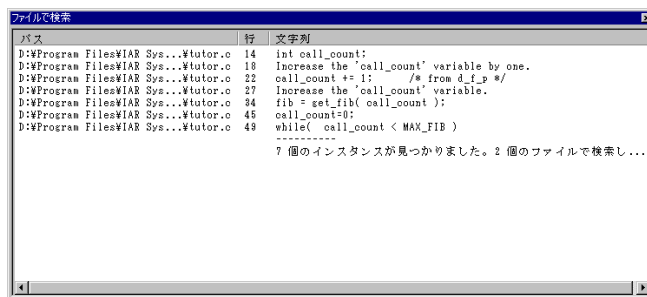


図 35: [ファイルで検索] ウィンドウ (メッセージウィンドウ)

[ファイルで検索] ウィンドウには、[編集] > [検索と置換] > [ファイルで検索] コマンドの出力が表示されます。デフォルトでは、このウィンドウは他のメッセージウィンドウとグループ化されて表示されます (89 ページのウィンドウを参照)。

このウィンドウでメッセージをダブルクリックすると、該当ファイルが編集用に開かれ、挿入ポイントが正しい箇所に表示されます。ソースの位置は、青い旗のアイコンで強調表示されます。

コンテキストメニュー

以下のコンテキストメニューがあります。

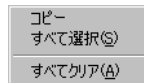


図36: [ファイルで検索] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

コピー	ウィンドウの内容をコピーします。
すべて選択	ウィンドウの内容を選択します。
すべてクリア	ウィンドウの内容とエディタウィンドウの左端にある青い旗のアイコンをすべてを削除します。

[ツール出力] ウィンドウ

[ツール出力] ウィンドウは、[表示] > [メッセージ] > [ツール出力] を選択すれば使用できます。

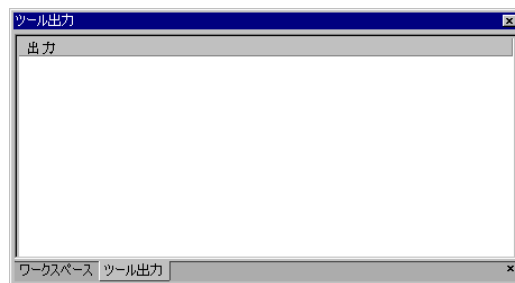


図37: [ツール出力] ウィンドウ (メッセージウィンドウ)

[ツール出力] ウィンドウには、[ツール] メニューのユーザ定義ツールによるすべてのメッセージ出力が表示されます。ただし、[ツールの設定] ダイアログボックスで [出力ウィンドウにリダイレクト] オプションを選択している必要があります (150 ページの [ツールの設定] ダイアログボックスを参照)。デフォルトでは、このウィンドウは他のメッセージウィンドウとグループ化されて表示されます (89 ページの ウィンドウを参照)。

コンテキストメニュー

以下のコンテキストメニューがあります。

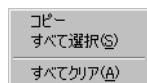


図 38: [ツール出力] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

コピー	ウィンドウの内容をコピーします。
すべて選択	ウィンドウの内容を選択します。
すべてクリア	ウィンドウの内容を削除します。

[デバッグログ] ウィンドウ

[デバッグログ] ウィンドウは、[表示] > [メッセージ] > [デバッグログ] を選択すれば使用できます。



図 39: [デバッグログ] ウィンドウ (メッセージウィンドウ)

[デバッグログ] ウィンドウには、診断メッセージやトレース情報のようなデバッガの出力が表示されます。デフォルトでは、このウィンドウは他のメッセージウィンドウとグループ化されて表示されます (89 ページの ウィンドウ を参照)。

以下のフォーマットのいずれかの行をダブルクリックすると、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。

```
<path> (<row>):<message>
<path> (<row>,<column>):<message>
```

コンテキストメニュー

以下のコンテキストメニューがあります。

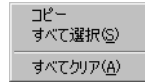


図 40: [デバッグログ] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

コピー	ウィンドウの内容をコピーします。
すべて選択	ウィンドウの内容を選択します。
すべてクリア	ウィンドウの内容を削除します。

メニュー

使用可能なメニューは、以下のとおりです。

- [ファイル] メニュー
- [編集] メニュー
- [表示] メニュー
- [プロジェクト] メニュー
- [ツール] メニュー
- [ウィンドウ] メニュー
- [ヘルプ] メニュー

また、デバッガを起動すると、C-SPY 専用のメニューが使用可能になります。これらのメニューのリファレンス情報については、『*ARM® 用 C-SPY® デバッグガイド*』を参照してください。

[ファイル] メニュー

[ファイル] メニューには、ワークスペースおよびソースファイルのオープン、保存、出力、IDE の終了を実行するためのコマンドが表示されます。

また、最近開いたファイルやワークスペースの番号付きリストも表示されます。メニューから選択することにより、これらを開くことができます。



図 41: [ファイル] メニュー

以下のコマンドがあります。



新規作成
CTRL+N

新しいワークスペースやテキストファイルを作成するコマンドを含むサブメニューを表示します。



開く > ファイル
CTRL+O

テキストファイルや HTML ドキュメントを選択して開くことができるサブメニューを表示します。
92 ページの *エディタウィンドウ* を参照してください。



開く > ワークスペース

開くワークスペースファイルを選択するためのサブメニューを表示します。新しいワークスペースを開く前に、現在開かれているワークスペースを保存して閉じるかどうかを確認するメッセージが表示されます。





**開く > ヘッダ /
ソースファイル**
Ctrl+Shift+H

現在のファイルに対応するヘッダファイルやソースファイルを開き、現在のファイルから新しく開かれたファイルに移動します。このコマンドは、エディタウィンドウのコンテキストメニューからも実行できます。

閉じる

アクティブなウィンドウを閉じます。修正されているファイルを、閉じる前に保存するかどうかを確認するメッセージが表示されます。

ワークスペースを開く	ワークスペースファイルを開くためのダイアログボックスを表示します。 新しいワークスペースを開く前に、現在開かれていて修正されているワークスペースファイルを保存して閉じるかどうかを確認するメッセージが表示されます。
名前を付けてワークスペースを保存	現在のワークスペースファイルを保存します。
ワークスペースを閉じる	現在のワークスペースファイルを閉じます。
 保存 CTRL+S	現在のテキストファイルやワークスペースファイルを保存します。
名前を付けて保存	現在のファイルを別名で保存するためのダイアログボックスを表示します。
すべて保存	開かれているすべてのテキストドキュメントとワークスペースファイルを保存します。
ページ設定	印刷オプションを設定するためのダイアログボックスを表示します。
 印刷 CTRL+P	テキストドキュメントを印刷するためのダイアログボックスを表示します。
最近使用したファイル	最近開いたテキストドキュメントをすばやく開くためのサブメニューを表示します。
最近使用したワークスペース	最近開いたワークスペースファイルをすばやく開くためのサブメニューを表示します。
終了	IDEを終了します。テキストファイルを閉じる前に、変更内容を保存するかどうかを確認するメッセージが表示されます。プロジェクトの変更は自動的に保存されます。

【編集】メニュー

【編集】メニューから、編集 / 検索用のコマンドを実行できます。

元に戻す(U)	Ctrl+Z
繰り返す(R)	Ctrl+Y
切り取り(T)	Ctrl+X
コピー(C)	Ctrl+C
貼り付け(P)	Ctrl+V
形式を選択して貼り付け(S)...	
すべて選択(L)	Ctrl+A
検索と置換(F)	▶
移動(G)	▶
コードテンプレート(O)	▶
次のエラー/タグ(E)	F4
前のエラー/タグ(U)	Shift+F4
入力補完(M)	Ctrl+Space
括弧のマッチング(H)	Ctrl+B
自動インデント(I)	Ctrl+T
ブロックコメント(C)	Ctrl+K
ブロックコメントの解除(B)	Ctrl+Shift+K
ブレークポイントの切り替え(A)	F9
ブレークポイントの有効化/無効化(N)	Ctrl+F9

図 42: 【編集】メニュー

以下のコマンドがあります。



元に戻す
CTRL+Z

現在のエディタウィンドウで最後に行った変更を取り消します。



やり直し
CTRL+Y

現在のエディタウィンドウで【元に戻す】により取り消した変更を再実行します。

エディタウィンドウごとに、編集のアンドウ / リドゥを無制限に実行することができます。



切り取り
CTRL+X

エディタウィンドウとテキストボックスでテキストを切り取るための Windows 標準のコマンド。



コピー
CTRL+C

エディタウィンドウとテキストボックスでテキストをコピーするための Windows 標準のコマンド



貼り付け
CTRL+V

エディタウィンドウとテキストボックスでテキストを貼り付けるための Windows 標準のコマンド

形式を選択して貼り付け

クリップボードに最近コピーした内容から、エディタドキュメントに貼り付けるものを選択できます。

すべて選択
CTRL+A

アクティブなエディタウィンドウで、すべてのテキストを選択します。



検索と置換 > 検索
CTRL+F

現在のエディタウィンドウでテキストを検索するための **〔検索〕** ダイアログボックスを表示します (110 ページの **〔検索〕** ダイアログボックスを参照)。
〔検索〕 コマンドの選択時に **〔メモリ〕** ウィンドウに挿入ポイントがある場合は、ダイアログボックスに表示されるオプションが変化します。**〔検索〕** コマンドを選択したときに挿入ポイントが **〔トレース〕** ウィンドウにある場合、**〔トレースを検索〕** ダイアログボックスが開きます。このダイアログボックスの内容は、使用する C-SPY ドライバによって異なります (詳しくは *ARM® 用 C-SPY® デバッグガイド* を参照)。



検索と置換 > 次を検索
F3

指定した文字列に一致する次の箇所を検索します。



検索と置換 > 前を検索
Shift+F3

指定した文字列に一致する前の箇所を検索します。

検索と置換 > 次を検索 (指定文字列)
CTRL+F3

現在選択されている文字列または現在挿入ポイントを囲んでいる単語に一致する次の箇所を検索します。

検索と置換 > 前を検索 (指定文字列)
Ctrl+Shift+F3

現在選択されている文字列または現在挿入ポイントを囲んでいる単語に一致する前の箇所を検索します。



検索と置換 > 置換
CTRL+H

指定した文字列を検索し、一致箇所を別の文字列に置換するためのダイアログボックスを表示します (111 ページの **〔置換〕** ダイアログボックスを参照)。
〔置換〕 コマンドの選択時に **〔メモリ〕** ウィンドウに挿入ポイントがある場合は、ダイアログボックスに表示されるオプションが変化します。

検索と置換 > ファイルから検索

指定した文字列を複数のテキストファイルで検索するためのダイアログボックスを表示します (112 ページの **〔ファイルから検索〕** ダイアログボックスを参照)。

検索と置換 > インクリメンタル検索
CTRL+I

検索文字列を少しずつ変更し、検索の絞り込みや拡大を行うことができるダイアログボックスを表示します (114 ページの **「インクリメンタル検索」** ダイアログボックスを参照)。



移動 > 行へ移動
CTRL+G

「行へ移動」 ダイアログボックスを表示します。このダイアログボックスを使用して、現在のエディタウィンドウで指定されている行や列に挿入ポイントを移動できます。

移動 > ブックマークの切替え
CTRL+F2

アクティブなエディタウィンドウの挿入ポイントのある行で、ブックマークを設定 / 解除します。

移動 > ブックマークへ移動
F2

挿入ポイントを、**「ブックマークの切替え」** コマンドで定義した次のブックマークに移動します。

移動 > 前へ移動
ALT+ ←

挿入ポイント履歴で前の項目に移動します。挿入ポイントの現在の位置は、**「定義に移動」** コマンドの実行時や、**「ファイルで検索」** コマンドの結果をクリックしたときに、履歴に追加されます。

移動 > 次へ移動
ALT+ →

挿入ポイント履歴で次の項目に移動します。挿入ポイントの現在の位置は、**「定義に移動」** コマンドの実行時や、**「ファイルで検索」** コマンドの結果をクリックしたときに、履歴に追加されます。

移動 > 定義に移動
F12

選択されたシンボルや挿入ポイントが置かれているシンボルの定義を表示します。ブラウズ情報が有効な場合に、このメニューコマンドを使用できます (141 ページの **「プロジェクト」** オプションを参照)。

コードテンプレート > テンプレートの挿入
Ctrl+Shift+ スペース

挿入ポイントの位置に挿入するコードテンプレートを選択できるリストを、エディタウィンドウで表示します。選択したコードテンプレートでフィールドへの入力が必要な場合は、**「テンプレート」** ダイアログボックスが表示されます (115 ページの **「テンプレート」** ダイアログボックスを参照)。コードテンプレートの使用方法については、74 ページの **コードテンプレートの使用と追加を参照** してください。

コードテンプレート > テンプレートの編集	現在のコードテンプレートファイルを開き、既存のコードテンプレートの修正やユーザ定義コードテンプレートの追加を行います。コードテンプレートの使用方法については、74 ページの <i>コードテンプレートの使用と追加を参照してください。</i>
次のエラー / タグ F4	メッセージウィンドウにエラーメッセージのリストや [ファイルで検索] による検索の結果が含まれる場合、このコマンドによってそのリストの次の項目がエディタウィンドウに表示されます。
前のエラー / タグ Shift+F4	メッセージウィンドウにエラーメッセージのリストや [ファイルで検索] による検索の結果が含まれる場合、このコマンドによってそのリストの前の項目がエディタウィンドウに表示されます。
入力補完 CTRL+ スペース	入力内容に応じて、エディタドキュメントの他の部分の内容から入力語を推測して補完します。
括弧のマッチング	挿入ポイントの直近の括弧内のテキストをすべて選択します。すでに選択されている場合は、その外側の次の括弧まで選択範囲を拡大します。外側に括弧がない場合は、ビープ音を再生します。
自動インデント CTRL+T	C/C++ ソースファイルで選択した行にインデントを設定します。インデントの設定については、134 ページの [自動インデントの設定] ダイアログボックスを参照してください。
ブロックコメント CTRL+K	C++ のコメント文字列 <code>//</code> を、選択した行の最初に追加します。
ブロックコメントの解除 CTRL+K	C++ のコメント文字列 <code>//</code> を、選択した行の最初から削除します。
ブレイクポイントの切替え F9	ソースウィンドウで、カーソルを含むかまたはカーソルの近くの文か命令で、ブレイクポイントを設定 / 解除します。 このコマンドは、デバッグバーのアイコンボタンから実行できます。
ブレイクポイントの有効化 / 無効化 CTRL+F9	ブレイクポイントの無効（実際には削除せず、後で再度使用できる状態にする）と有効を切り替えます。

【検索】 ダイアログボックス

【検索】 ダイアログボックスは【編集】メニューからアクセスできます。

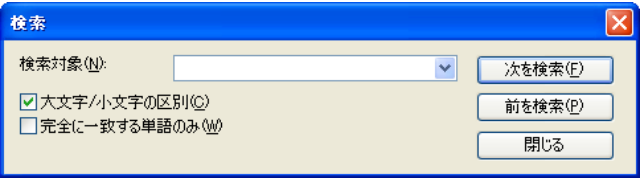



図 43: 【検索】 ダイアログボックス

エディタウィンドウで検索した場合は、【メモリ】ウィンドウで検索した場合に比べて内容が異なります。

検索対象	検索するテキストを指定します。
大文字 / 小文字 の区別	指定されたテキストの大文字と小文字が完全に一致するものだけを検索します。このオプションを指定しない場合は、int を検索すると、INT、Int も検索されます。このオプションは、エディタウィンドウでの検索時にだけ使用できます。
完全に一致する 単語のみ	単語として一致する箇所だけを検索します。このオプションを指定しない場合は、int を検索すると、print、sprintf も検索されます。このオプションは、エディタウィンドウでの検索時にだけ使用できます。
16 進数値を検索	指定した 16 進数値を検索します。このオプションは、【メモリ】ウィンドウでの検索時にだけ使用できます。
 次を検索	選択したテキストの次の一致箇所を検索します。
前を検索	選択したテキストに一致する前の箇所を検索します。
停止	実行中の検索を停止します。このボタンは、【メモリ】ウィンドウでの検索時にだけ使用できます。

「置換」ダイアログボックス

「置換」ダイアログボックスは「編集」メニューからアクセスできます。

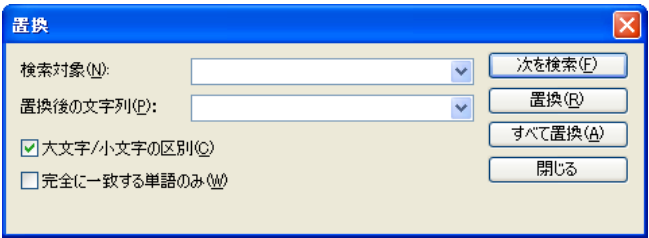


図 44: 「置換」ダイアログボックス

エディタウィンドウで検索した場合は、「メモリ」ウィンドウで検索した場合に比べて内容が異なります。

検索対象	検索するテキストを指定します。
置換後の文字列	一致する箇所と置換するテキストを指定します。
大文字 / 小文字 の区別	指定されたテキストの大文字と小文字が完全に一致するものだけを検索します。このオプションを指定しない場合は、int を検索すると、INT、Int も検索されます。このオプションは、エディタウィンドウでの検索時にだけ使用できます。
完全に一致する 単語のみ	単語として一致する箇所だけを検索します。このオプションを指定しない場合は、int を検索すると、print、sprintf も検索されます。このオプションは、エディタウィンドウでの検索時にだけ使用できます。
16 進数値を検索	指定した 16 進数値を検索します。このオプションは、「メモリ」ウィンドウでの検索時にだけ使用できます。
次を検索	指定したテキストに一致する次の箇所を検索します。
置換	一致箇所のテキストを指定テキストに置換します。
すべて置換	現在のエディタウィンドウで検索テキストに一致する箇所をすべて置換します。

【ファイルから検索】ダイアログボックス

【ファイルで検索】ダイアログボックスは、【編集】メニューからアクセスできます。

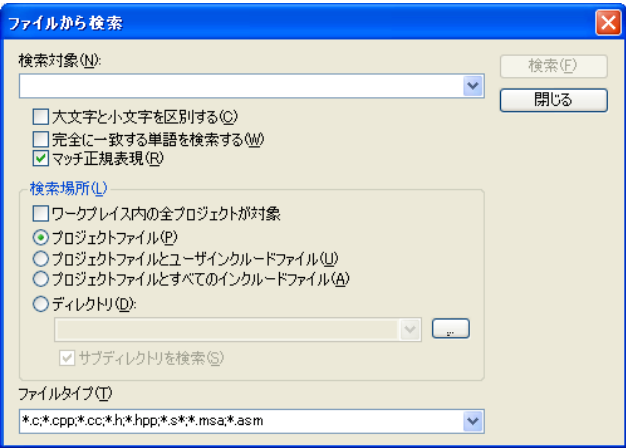


図 45: 【ファイルで検索】ダイアログボックス

このダイアログボックスを使用して、ファイル内で文字列を検索します。

検索結果は、【ファイルから検索】メッセージウィンドウ（【表示】メニューからアクセス）で表示されます。【編集】>【次のエラー/タグ】コマンドを選択するか、【ファイルから検索】メッセージウィンドウでメッセージをダブルクリックして、一致箇所に移動することができます。対応するファイルがエディタウィンドウで表示され、一致箇所の最初の位置に挿入ポイントが設定されます。左端の余白にある青色のフラグにより、該当行が示されます。

検索テキスト

検索する文字列または正規表現を指定します。以下の条件を必要なだけ使用して、検索を絞り込むことができます。

- 大文字 / 小文字の
区別

指定されたテキストの大文字と小文字が完全に一致するものだけを検索します。このオプションを指定しない場合は、int を検索すると、INT、Int も検索されます。
- 完全に一致する
単語のみ

単語として独立した文字列のみを検索します（ショートカット &w）。このオプションを指定しなかった場合、int を検索すると、print、sprintf など検索されます。
- マッチ正規表現

正規表現のみを検索します。つまり、Perl プログラミング言語の標準に従う必要があります。

検索場所

検索するファイルを以下から選択します。

ワークスペース内の全プロジェクトが対象	アクティブなプロジェクトだけでなく、ワークスペースの全プロジェクトが検索されます。
プロジェクトファイル	明示的にプロジェクトに追加した全ファイルが検索されます。
プロジェクトファイルとユーザインクルードファイル	プロジェクトに明示的に追加したすべてのファイルと、それらのファイルに含まれるすべてのファイル（IAR Embedded Workbench のインストールディレクトリ内のインクルードファイルを除く）に対して検索が実行されます。
プロジェクトファイルとすべてのインクルードファイル	プロジェクトに明示的に追加したすべてのファイルと、それらに含まれるすべてのファイルが検索されます。
ディレクトリ	指定したディレクトリが検索されます。最近検索した場所が、ドロップダウンリストに保存されます。参照ボタンを使用して、ディレクトリを指定します。
サブディレクトリを検索	指定したディレクトリと、そのサブディレクトリがすべて検索されます。

ファイルタイプ

検索するファイルのタイプを選択するフィルタ。フィルタはすべての**【検索場所】**設定に適用されます。ドロップダウンリストからフィルタを選択します。テキストフィールドは編集可能で、自分のフィルタを追加することができます。フィルタのゼロ文字以上の任意の文字を示すには * を、任意の 1 文字を示すには ? を使用します。

停止

実行中の検索を停止します。このボタンは、検索中にだけ使用できます。

【インクリメンタル検索】ダイアログボックス

【インクリメンタル検索】ダイアログボックスは【編集】メニューからアクセスできます。



図 46: 【インクリメンタル検索】ダイアログボックス

このダイアログボックスを使用して、検索文字列を徐々に調整または拡張します。

検索対象

検索する文字列を入力します。検索は、挿入ポイントの位置（開始位置）から実行されます。検索文字列に文字を追加したり削除するたびに、検索結果がすぐ変わります。文字を1つ削除すると、開始地点から検索がもう一度スタートします。

【インクリメンタル検索】ダイアログボックスを表示したときにエディタウィンドウで文字列を選択している場合は、その文字列が【検索テキスト】テキストボックスに表示されます。

大文字 / 小文字の区別

指定されたテキストの大文字と小文字が完全に一致するものを検索します。このオプションを指定しない場合は、int を検索すると、INT、Int も検索されます。

次を検索

現在の検索文字列に一致する次の箇所を検索します。【次を検索】ボタンをクリックしたときに【検索テキスト】テキストボックスが空白の場合は、検索文字列がドロップダウンリストから自動的に選択されます。この文字列を検索するには、【次を検索】をクリックします。

閉じる

ダイアログボックスを閉じます。

【テンプレート】 ダイアログボックス

【テンプレート】 ダイアログボックスは、フィールドへの入力が必要なコードテンプレートを挿入すると表示されます。

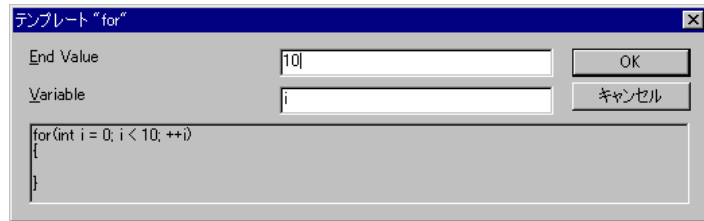


図 47: 【テンプレート】 ダイアログボックス

挿入したソースコードテンプレートに必要なフィールド入力を指定するには、このダイアログボックスを使用します。

注: この図は、for ループ用コードを自動挿入するためのデフォルトコードテンプレートの表示です。

テキストフィールド

テキストフィールドの必要な入力内容を指定します。コードテンプレートの定義に応じたフィールドが表示されます。

表示エリア

表示エリアには、入力した値を使用してコードテンプレートから生成されるコードが表示されます。

コードテンプレートの使用方法については、74 ページの *コードテンプレートの使用と追加* を参照してください。

【表示】メニュー

【表示】メニューは、IDE でウィンドウを開いてツールバーを表示するためのいくつかのコマンドを提供します。デバッガの実行中は、このメニューからデバッガ固有のウィンドウも開くことができます。以下については、『ARM® 用 C-SPY® デバッガガイド』を参照してください。



図 48: 【表示】メニュー

以下のコマンドがあります。

メッセージ	IAR Embedded Workbench コマンドからのメッセージやテキスト出力を表示するメッセージウィンドウ（[ビルド]、[ファイルで検索]、[ツール出力]、[デバッグログ]）を選択するためのサブメニューを表示します。メニューから選択したウィンドウがすでに開いている場合は、そのウィンドウがアクティブになります。
ワークスペース	現在の [ワークスペース] ウィンドウを開きます（42 ページの [ワークスペース] ウィンドウを参照）。
ソースブラウザ	[ソースブラウザ] ウィンドウを表示します（50 ページの [ソースブラウザ] ウィンドウを参照）。
ブレークポイント	[ブレークポイント] ウィンドウを表示します（ARM® 用 C-SPY® デバッガガイドを参照）。
逆アセンブリウィンドウ	[逆アセンブリ] ウィンドウを開きます。デバッガの実行中にのみ使用できます。
メモリ	[メモリ] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
シンボルメモリ	[シンボルメモリ] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
レジスタ	[レジスタ] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
ウォッチ	[ウォッチ] ウィンドウを表示します。デバッガの実行中にのみ使用できます。

ローカル	[ローカル] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
静的変数	[静的] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
自動	[自動] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
ライブウォッチ	[ライブウォッチ] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
クイックウォッチ	[クイックウォッチ] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
呼出しスタック	[呼出しスタック] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
ターミナル I/O	[ターミナル I/O] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
コードカバレッジ	[コードカバレッジ] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
プロファイリング	[プロファイリング] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
スタック	[スタック] ウィンドウを表示します。デバッガの実行中にのみ使用できます。
ツールバー	[メイン] / [デバッグ] オプションは、2 つのツールバーの表示 / 非表示を切り替えます。
ステータスバー	ステータスバーの表示 / 非表示を切り替えます。

【プロジェクト】メニュー

【プロジェクト】メニューには、ワークスペース、プロジェクト、グループ、ファイルの操作用コマンド、ビルドツールのオプションの指定用コマンド、現在のプロジェクトでツールを実行するためのコマンドが表示されます。

ファイルの追加(F)...	
グループの追加(G)...	
ファイルリストのインポート(O)...	
ビルド構成の編集(C)...	
削除(V)	
新規プロジェクトの作成(N)...	
既存プロジェクトの追加(E)...	
オプション(O)...	Alt+F7
バージョン管理システム(V)	▶
メイク(M)	F7
コンパイル(C)	Ctrl+F7
すべてを再ビルド(B)	
クリーン(L)	
バッチビルド(A)...	F8
ビルドを停止(S)	Ctrl+Break
ダウンロードしてデバッグ(D)	Ctrl+D
ダウンロードせずにデバッグ(H)	
メイク後デバッグを再起動(R)	Ctrl+R
デバッグを再起動(R)	Ctrl+Shift+R
ダウンロード(V)	▶
デバイスファイルを開く	▶

図 49: 【プロジェクト】メニュー

以下のコマンドがあります。

- ファイルの追加

現在のプロジェクトに追加するファイルを選択するためのダイアログボックスを表示します。
- グループの追加

新しいグループを作成するためのダイアログボックスを表示します。【グループ名】テキストボックスには、新しいグループ名を入力します。グループの詳細は、33 ページの [グループ](#)を参照してください。

ファイルリスト のインポート

通常の **【開く】** ダイアログボックスを表示します。
このダイアログボックスを使用して、IAR システムズの別のツールチェーンで作成したプロジェクトからファイルやグループに関する情報をインポートできます。

ファイル拡張子が古い形式の `pew`、`prj` のいずれかであるプロジェクトファイルから情報をインポートするには、現在の IAR Embedded Workbench のコンテキストメニューにある **【ファイルリストのエクスポート】** を使用して、先に情報をエクスポートする必要があります。

ビルド構成の編集

新しいビルド構成の定義や既存のビルド構成の削除を行うための **【プロジェクトの構成】** ダイアログボックスを表示します。48 ページの **【プロジェクトの構成】** ダイアログボックスを参照してください。

削除

【ワークスペース】 ウィンドウで、選択した項目をワークスペースから削除します。

新規プロジェクト の作成

【新規プロジェクトの作成】 ダイアログボックスを表示します。ここでは、新規プロジェクトを作成してそれをワークスペースに追加できます (47 ページの **【新規プロジェクトの作成】** ダイアログボックスを参照)。

既存プロジェクト の追加

既存のプロジェクトをワークスペースに追加するための標準の **【開く】** ダイアログボックスを表示します。

オプション Alt+F7

【ワークスペース】 ウィンドウで選択した項目に対して、各ビルドツールのオプションを設定できる **【オプション】** ダイアログボックスを表示します (123 ページの **【オプションダイアログボックス】** を参照)。プロジェクト全体、ファイルのグループ、個々のファイルのオプションを設定できます。

バージョン管理 システム

バージョン管理用コマンドのサブメニューを表示します (53 ページの **SCC のバージョン管理システムメニュー** を参照)。



メイク F7

最後のビルド以降に変更されたファイルだけをコンパイル、アセンブル、リンクして、現在のビルド構成を最新状態に更新。



コンパイル
CTRL+F7

選択されているファイルやグループをコンパイル/アセンブルします。

[ワークスペース] ウィンドウで、1 つまたは複数のファイルを選択できます。グループが異なる場合も含め、同一プロジェクト内のすべてのファイルを選択できます。コンパイルするファイルが表示されたエディタウィンドウを選択することもできます。**[コンパイル]** コマンドは、選択したすべてのファイルがコンパイルまたはアセンブル可能な場合にのみ有効です。

グループを選択することもできます。その場合、コンパイルできないファイル（ヘッダファイルなど）がグループに含まれている場合でも、そのグループ内（ネストされたグループを含む）のコンパイル可能なファイルごとにコマンドが実行されます。

選択したファイルが複数ファイルコンパイルグループの一部である場合にも、コマンドは選択したファイルのみに作用します。

すべてを再ビルド

現在のターゲットのすべてのファイルをリビルドし再リンクします。

クリーン

すべての中間ファイルを削除します。

バッチビルド
F8

[バッチビルド] ダイアログボックスを表示します。ここでは、指定したバッチビルド構成を作成し、指定したバッチをビルドできます。126 ページの **[バッチビルド]** ダイアログボックスを参照してください。



ビルドを停止
Ctrl+Break

現在のビルド処理を停止します。



ダウンロードして
デバッグ
CTRL+D

プロジェクトのオブジェクトファイルのデバッグができるように、アプリケーションをダウンロードし、**C-SPY** を起動します。必要であれば、**C-SPY** の実行前に **make** が実行され、プロジェクトが更新されます。このコマンドは、デバッグ中は使用できません。



ダウンロードせずに
デバッグ

プロジェクトのオブジェクトファイルのデバッグができるように、**C-SPY** を起動します。このメニューコマンドは、**[ダウンロード]** ページの **[ダウンロードを中止する]** オプションのショートカットです。**[ダウンロードせずにデバッグ]** コマンドは、デバッグ中は使用できません。



メイク後デバッグを再起動

C-SPY の停止、アクティブなビルド構成の作成、デバッグ再開を実行します。すべてを 1 つのコマンドで実行します。このコマンドは、デバッグ中のみ使用できます。



デバッグを再起動

C-SPY の停止とデバッグ再開を実行します。すべてを 1 つのコマンドで実行します。このコマンドは、デバッグ中のみ使用できます。

ダウンロード

フラッシュダウンロードおよび消去のためのコマンド。以下のコマンドから選択します。

[アクティブなアプリケーションのダウンロード] は、すべてのデバッグセッションを開始せずに、アクティブなアプリケーションをターゲットにダウンロードします。この結果は、デバッグセッションを開始して実行が発生する前に終了したときとほぼ同じになります。

[ファイルのダウンロード] は、標準の **【開く】** ダイアログボックスを開きます。ここでは、完全なデバッグセッションを開始せずにターゲットシステムにダウンロードするファイルを指定できます。

[メモリ消去] フラッシュメモリのすべてまたは一部を消去します。

.board ファイルでフラッシュメモリが 1 つだけ指定されていれば、消去を確認する場面で単純な確認用のダイアログボックスが表示されます。ただし、.board ファイルで 2 つ以上のフラッシュメモリが指定されている場合、**[メモリ消去]** ダイアログボックスが表示されます。122 ページの **[メモリ消去]** ダイアログボックスを参照してください。

デバイスファイルを開く

デバイス記述や SFR 定義を含むアクティブなファイルを開くためのコマンドがあるサブメニューを開きます。

[メモリ消去] ダイアログボックス

[メモリ消去] ダイアログボックスは、[プロジェクト] > [ダウンロード] > [メモリ消去] を選択し、フラッシュメモリのシステム構成ファイル（ファイル名拡張子 .board）で複数のフラッシュメモリが指定されているときに表示されます。

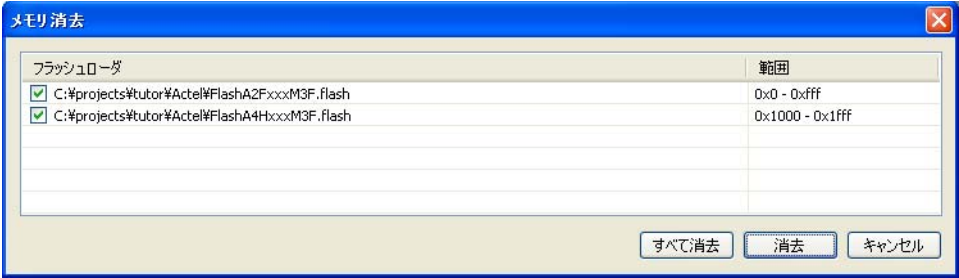


図 50: [メモリ消去] ダイアログボックス

このダイアログボックスを使用して、必要な数のフラッシュメモリを消去します。

表示エリア

各行にフラッシュメモリのデバイス設定ファイル（ファイル名の拡張子 .flash）のパスと関連のメモリ範囲が一覧表示されます。消去するメモリを選択してください。

ボタン

以下のボタンを選択できます。

- すべて消去

個別に選択した行に関係なく、このダイアログボックスに表示されたすべてのメモリが消去されます。
- 消去

選択したメモリが消去されます。
- キャンセル

ダイアログボックスを閉じます。

オプションダイアログボックス

[オプション] ダイアログボックスは、[プロジェクト] メニューから使用できます。

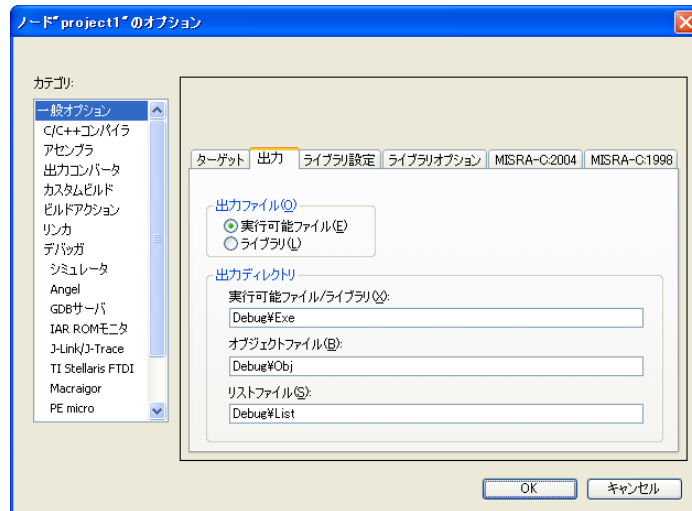


図 51: オプションダイアログボックス

このダイアログボックスを使用して、プロジェクト設定を指定します。

カテゴリ

オプションを設定する対象のビルドツールを選択します。使用可能なカテゴリは、IAR Embedded Workbench IDE にインストールされているツールによって異なり、通常は以下が含まれます。

一般オプション	一般オプション (159 ページの <i>一般オプション</i> を参照)。
C/C++ コンパイラ	IAR C/C++ コンパイラオプション (167 ページの <i>コンパイラオプション</i> を参照)。
アセンブラ	IAR アセンブラオプション (183 ページの <i>アセンブラオプション</i> を参照)。
出力コンバータ	ELF 出力を Motorola、Intel 標準、その他の簡易フォーマットに変換するためのオプション (191 ページの <i>出力コンバータオプション</i> を参照)。

カスタムビルド	ツールチェーンを拡張するオプション (193 ページの <i>カスタムビルドオプション</i> を参照)。
ビルドアクション	ビルド前 / ビルド後のアクション用オプション (195 ページの <i>ビルドアクションオプション</i> を参照)。
リンカ	リンカオプション (197 ページの <i>リンカのオプション</i> を参照)。このカテゴリは、アプリケーションプロジェクトの場合に表示されます。
ライブラリビルダ	ライブラリビルダオプション (209 ページの <i>ライブラリビルダオプション</i> を参照)。このカテゴリは、ライブラリプロジェクトの場合に表示されます。
デバッガ	IAR C-SPY デバッガオプション (『ARM® 用 C-SPY® デバッガガイド』を参照)。
シミュレータ	シミュレータ固有のオプション (『ARM® 用 C-SPY® デバッガガイド』を参照)。
C-SPY ハードウェアドライバ	インストールされたドライバによっては、追加のハードウェアデバッガに固有のオプション (『ARM® 用 C-SPY® デバッガガイド』を参照)。

カテゴリを選択すると、IDE のコンポーネントに対するオプションのページが表示されます。

工場出荷時設定

すべての設定をデフォルトの工場出荷時設定に戻します。

引数変数

[オプション] ダイアログボックスのほとんどのページで、パスや引数に引数変数を使用できます。

変数	説明
\$CONFIG_NAME\$	現在のビルド構成の名前 (Debug、Release など)
\$CUR_DIR\$	現在のディレクトリ
\$CUR_LINE\$	現在の行
\$DATE\$	今日の日付
\$EW_DIR\$	IAR Embedded Workbench のトップディレクトリ (例: c:\program files\iar systems\embedded workbench 6.n)

表 10: 引数変数

変数	説明
\$EXE_DIR\$	実行可能ファイル出力用ディレクトリ
\$FILE_BNAME\$	ファイル名（拡張子を除く）
\$FILE_BPATH\$	フルパス（拡張子を除く）
\$FILE_DIR\$	アクティブなファイルのディレクトリ（ファイル名を除く）
\$FILE_FNAME\$	アクティブなファイルのファイル名（パスを除く）
\$FILE_PATH\$	エディタ、プロジェクト、メッセージウィンドウで、アクティブなファイルのフルパス
\$LIST_DIR\$	リスト出力用ディレクトリ
\$OBJ_DIR\$	オブジェクト出力用ディレクトリ
\$PROJ_DIR\$	プロジェクトディレクトリ
\$PROJ_FNAME\$	プロジェクトファイル名（パスなし）
\$PROJ_PATH\$	プロジェクトファイルのフルパス
\$TARGET_DIR\$	主要出力ファイル用ディレクトリ
\$TARGET_BNAME\$	主要出力ファイルのファイル名（パス、拡張子を除く）
\$TARGET_BPATH\$	主要出力ファイルのフルパス（拡張子を除く）
\$TARGET_FNAME\$	主要出力ファイルのファイル名（パスを除く）
\$TARGET_PATH\$	主要出力ファイルのフルパス
\$TOOLKIT_DIR\$	アクティブな製品のディレクトリ（例:c:\program files\iar systems\embedded workbench 6.n\arm）
\$USER_NAME\$	ホストログイン名
\$_ENVVAR_\$	環境変数 ENVVAR。\$_ と \$_\$ に囲まれた名前はすべて、そのシステム環境変数に展開されます

表 10: 引数変数 (続き)

引数変数は、[IDE オプション] ダイアログボックスの一部のページでも使用できます (128 ページの [ツール] メニューを参照)。

「バッチビルド」ダイアログボックス

「バッチビルド」ダイアログボックスは、「プロジェクト」>「バッチビルド」を選択すると使用できます。



図 52: 「バッチビルド」ダイアログボックス

このダイアログボックスには、ビルド構成の定義済みバッチがすべて一覧表示されます。詳細は 64 ページの [バッチによる複数構成のビルド](#) を参照してください。

バッチ

現在定義されたビルド構成のバッチのこのリストから、ビルドするバッチを選択します。

ビルド

実行するビルドコマンドを指定します。

- 作成
- クリーン
- すべてをビルド

新規作成

新しいビルド構成のバッチを定義するための「[バッチビルドの編集](#)」ダイアログボックスを表示します（127 ページの「[バッチビルドの編集](#)」ダイアログボックスを参照）。

削除

選択したバッチを削除します。

編集

既存のビルド構成のバッチを編集するための**「バッチビルドの編集」**ダイアログボックスを表示します。

「バッチビルドの編集」ダイアログボックス

「バッチビルドの編集」ダイアログボックスは、**「バッチビルド」**ダイアログボックスから使用できます。



図 53: 「バッチビルドの編集」ダイアログボックス

このダイアログボックスを使用して、ビルド構成の新しいバッチを作成し、既存のバッチを編集します。

名前

作成しているバッチ名を入力するか、編集中のバッチの既存の名前を変更します（希望する場合）。

使用可能な構成

ワークスペースに属するすべてのビルド構成のこのリストから、作成または編集しているバッチに含める構成を選択します。

ビルド構成を**「使用可能な構成」**リストから**「ビルドする構成」**リストに移動するには、矢印ボタンを使用します。

ビルドする構成

作成または編集中のバッチに含めるビルド構成の一覧を表示します。ビルド構成を上下にドラッグして、構成の順序を設定します。

[ツール] メニュー

[ツール] メニューには、共通フォントの変更、ショートカットキーの変更などの、環境のカスタマイズ用コマンドが表示されます。

ユーザが定義可能なメニューで、IAR Embedded Workbench で使用するツールを追加することができます。したがって、メニュー項目として表示されるように設定したツールに応じて、表示が異なる場合があります。

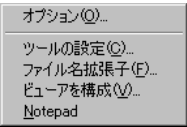


図 54: [ツール] メニュー

以下のコマンドがあります。

- オプション** **[IDE オプション]** ダイアログボックスを表示します。このダイアログボックスを使用して、IDE のカスタマイズができます。
- ツールの設定** 外部ツールを利用するためのインタフェースを設定できる **[構成ツール]** ダイアログボックスを表示します (150 ページの **[ツールの設定]** ダイアログボックスを参照)。
- ファイル名拡張子** ビルドツールで指定可能なファイル名の拡張子を定義するための **[ファイル名の拡張子]** ダイアログボックスを表示します (152 ページの **[ファイル名拡張子]** ダイアログボックスを参照)。
- ビューアの設定** ドキュメント表示用のビューアアプリケーションを設定するための **[ビューアの設定]** ダイアログボックスを表示します (155 ページの **[ビューアの設定]** ダイアログボックスを参照)。
- メモ帳** ユーザ設定項目。ユーザが **[ツール] メニュー** に追加した項目が表示されます。

【共通フォント】オプション

【共通フォント】オプションは、[ツール] > オプションを選択すると使用できます。



図 55: 【共通フォント】オプション

このページを使用して、エディタウィンドウを除くすべてのプロジェクトウィンドウで使用されるフォントを設定します。

エディタウィンドウのフォントの変更方法については、138 ページの [色とフォント] オプションを参照してください。

固定幅フォント

[逆アセンブリ]、[レジスタ]、[メモリ] の各ウィンドウで使用するフォントを選択します。

プロポーショナルフォント

[逆アセンブリ]、[レジスタ]、[メモリ] およびエディタウィンドウを除く、すべてのウィンドウで使用するフォントを選択します。

「キーカスタマイズ」 オプション

「キーカスタマイズ」 オプションは、「ツール」 > オプションを選択すると使用できます。

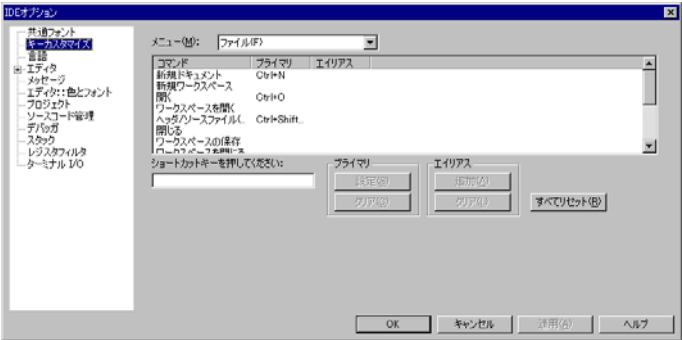


図 56: 「キーバインディング」 オプション

このページを使用して、IDE のメニューコマンドで使用されるショートカットキーをカスタマイズします。

メニュー

編集するメニューを選択します。選択したメニューについて現在定義されているすべてのショートカットキーが、「メニュー」 ドロップダウンリストの下に一覧表示されます。

コマンドのリスト

独自のショートカットキーを設定するメニューコマンドを、選択したメニューで使用可能なすべてのコマンドのリストから選択します。

ショートカットキーを押してください

選択したコマンドのショートカットキーとして使用するキーの組合せを入力します。他のコマンドで使用されているショートカットの設定や追加はできません。

プライマリ

以下から選択します。

セット キーの組合せを、リストで選択したコマンドのショートカットとして「ショートカットキーを押してください」 フィールドに保存します。

クリア リストで選択したコマンドのショートカットとして表示されたプライマリキーの組合せを削除します。

メニューコマンド名の横に新しいショートカットキーが表示されます。

エイリアス

以下から選択します。

追加 キーの組合せを、リストで選択したコマンドのエイリアス（表示されないショートカット）として **［ショートカットキーを押してください］** フィールドに保存します。

クリア リストで選択したコマンドのショートカットとして表示されたエイリアスのキーの組合せを削除します。

メニューコマンド名の横に新しいショートカットキーは表示されません。

すべてリセット

すべてのコマンドショートカットキーを出荷時設定に戻します。

［言語］ オプション

［言語］ オプションは、**［ツール］ > オプション**を選択すると使用できます。

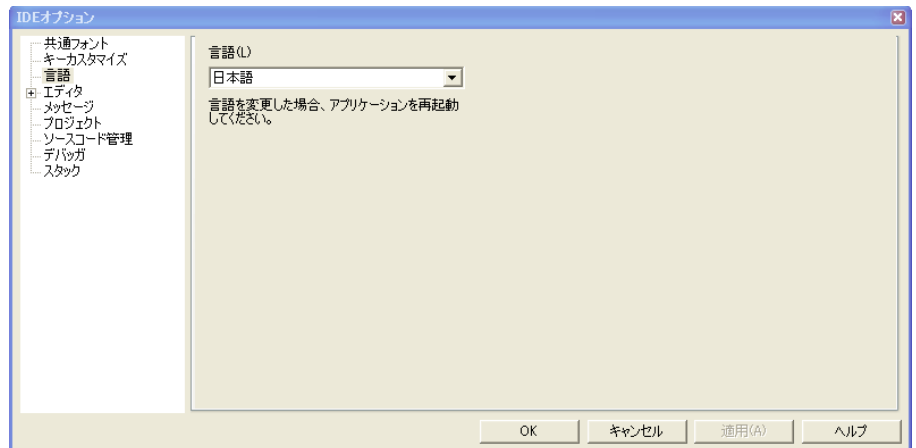


図 57: 言語 オプション

このページを使用して、ウィンドウやメニュー、ダイアログボックスなどを使用する言語を指定します。

言語

使用する言語を指定します。IDE では **【英語（米国）】** および **【日本語】** が使
用できます。

注：同一ディレクトリの複数の異なるツールチェーンに対して IAR Embedded Workbench がインストールされ、これらのツールチェーンで異なる言語が使用可能である場合、IDE で言語が混在することがあります。

[エディタ] オプション

[エディタ] オプションは、[ツール] > オプションを選択すると使用できます。

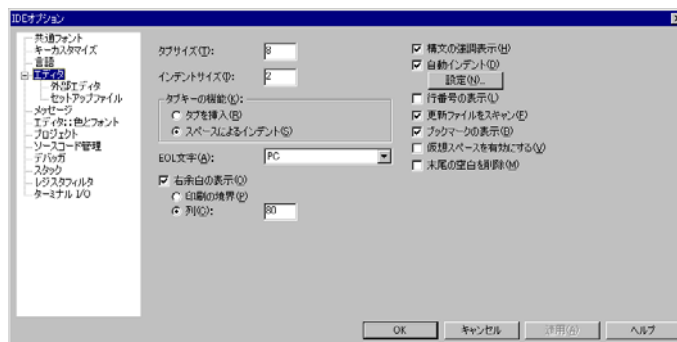


図58: [エディタ] オプション

このページを使用して、エディタを設定します。

エディタについて詳しくは、69 ページの **編集** を参照してください。

タブサイズ

タブ文字の幅を文字間隔で指定します。

インデントサイズ

インデント付きで表を作成するときに使用するスペースの数を指定します。

タブキーの機能

[タブ] キーを押したときの動作を制御します。以下から選択します。

タブを挿入 タブキーを押すと、タブ文字を1つ挿入します。

スペースによるインデント タブキーを押したときに、インデント（スペース文字）を1回分挿入します。

EOL 文字

エディタの文書を保存したときに、使用する改行文字を選択します。以下から選択します。

PC (デフォルト) Windows と DOS 形式の改行文字。

Unix UNIX 形式の改行文字。

元ファイルに従う 開かれたときファイルに設定されているのと同じ、PC または UNIX のどちらかの形式の改行文字。開かれたファイルに両方の形式が存在する場合や、どちらも存在しない場合には、PC 形式の改行文字が使用されます。

右余白の表示

エディタウィンドウの右側の余白部分の外側領域が薄い灰色で表示されます。このオプションを選択すると、左右の余白の間にあるテキストエリアの幅を設定できます。以下を基準に幅を選択して設定します。

印刷の境界 印刷可能な領域（プリンタの一般設定から読み込まれます）を基準に幅を決定します。

列 列数を基準に幅を決定します。

構文の強調表示

C/C++ アプリケーションの構文をさまざまなテキスト形式でエディタに表示します。

構文強調表示の詳細については、「138 ページの [色とフォント] オプション」、「72 ページの 構文カラー表示」。

自動インデント

Return キーを押すと、新しい行が自動的にインデントされます。C/C++ ソースファイルの場合、[設定] ボタンをクリックして自動インデント機能を設定します（134 ページの [自動インデントの設定] ダイアログボックスを参照）。他のテキストファイルの場合は、新しい行のインデントは前の行と同一に設定されます。

行番号の表示

エディタウィンドウに行番号を表示します。

更新ファイルをスキャン

他のツールで修正されたファイルをエディタで再ロードします。

ファイルが IDE で開かれていて、同じファイルが同時に別のツールで修正されている場合、そのファイルが自動的に IDE で再ロードされます。ただし、ファイルの編集をすでに開始している場合、ファイルを再ロードする前にプロンプトが表示されます。

ブックマークの表示

エディタウィンドウの左側に列を表示します。この列には、コンパイラのエラーとワーニング、[ファイルで検索] の結果、ユーザのブックマーク、ブレークポイントのアイコンが表示されます。

仮想スペースを有効にする

挿入ポイントをテキストエリアの外側に動かせるようにします。

末尾の空白を削除

ファイルをディスクに保存するときに、末尾の空白を削除します。最後の空白とは、空白以外の最後の文字と行末文字の間の空白文字です。

[自動インデントの設定] ダイアログボックス

[自動インデントの設定] ダイアログボックスは、[IDE オプション] ダイアログボックスから使用できます。

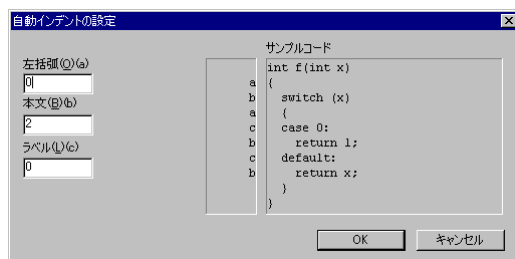


図 59: [自動インデントの設定] ダイアログボックス

このダイアログボックスを使用して、エディタによる C/C++ ソースコードの自動インデントを設定します。

インデントの詳細については、72 ページの *自動テキストインデント* を参照してください。

〔自動インデントの設定〕ダイアログボックスを開くには、次の手順に従います。

- 1 [ツール] > [オプション] を選択します。
- 2 [エディタ] ページを開きます。
- 3 [自動インデント] オプションを選択して、[設定] ボタンをクリックします。

左括弧 (a)

左括弧をインデントするときの空白文字数を指定します。

本文 (b)

左括弧の後または次の行にまたがる文の後のコードのインデントに使用する追加空白文字数を指定します。

ラベル (c)

ケースラベルを含むラベルをインデントする際に使用する追加空白文字数を指定します。

サンプルコード

このエリアに、テキストボックスで設定したインデント用設定が反映されます。すべてのインデントは、前の行、文、その他の文法構造と相対的に設定されます。

[外部エディタ] のオプション

[外部エディタ] オプションは、[ツール] > [オプション] を選択すると使用できます。

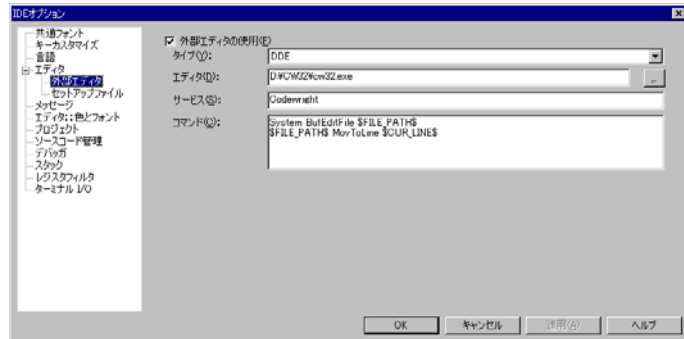


図 60: [外部エディタ] のオプション

このページを使用して、外部エディタを指定します。

注: このダイアログボックスの内容は、**[タイプ]** オプションの設定によって異なります。

78 ページの 外部エディタの連携も参照してください。

外部エディタの使用

外部エディタを使用を有効にします。

タイプ

インタフェースのタイプを選択します。以下から選択します。

- コマンドライン
- **DDE** (Windows Dynamic Data Exchange).

エディタ

外部エディタのファイル名とパスを指定します。参照ボタンを使用して選択することもできます。

引数

エディタに引き渡す引数を指定します。インタフェースのタイプとして【コマンドライン】を選択した場合に限り、適用できます（136 ページのタイプを参照）。

サービス

エディタで使用する DDE サービス名を指定します。インタフェースのタイプとして **[DDE]** を選択した場合に限り、適用できます (136 ページの タイプを参照)。

サービス名は、使用する外部エディタに応じて指定します。外部エディタのドキュメントを参照して、適切に設定してください。

コマンド

エディタに引き渡すコマンド文字列のシーケンスを指定します。コマンド文字列は、以下の形式で入力する必要があります。

```
DDE-Topic CommandString1
DDE-Topic CommandString2
```

インタフェースのタイプとして **[DDE]** を選択した場合に限り、適用できます (136 ページの タイプを参照)。

コマンド文字列は、使用する外部エディタに応じて指定します。外部エディタのドキュメントを参照して、適切に設定してください。



注： 引数に変数を使用できます。使用可能な引数変数については、124 ページの 引数変数を参照してください。

[セットアップファイル] のオプション

[セットアップファイル] オプションは、[ツール] > オプションを選択すると使用できます。

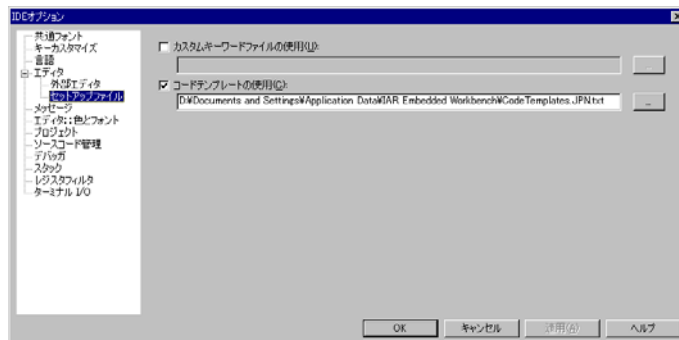


図 61: [エディタセットアップファイル] オプション

このページを使用して、エディタのセットアップファイルを指定します。

カスタムキーワードファイルの使用

エディタで強調表示するキーワードを含むテキストファイルを指定します。構文カラー表示については、「72 ページの [構文カラー表示](#)」を参照してください。

コードテンプレートの使用

頻繁に使用するコードをソースファイルに挿入するためのコードテンプレートを記述したテキストファイルを指定します。コードテンプレートの使用方法については、74 ページの [コードテンプレートの使用と追加](#)を参照してください。

【色とフォント】 オプション

【色とフォント】 オプションは、[ツール] > オプションを選択すると使用できます。

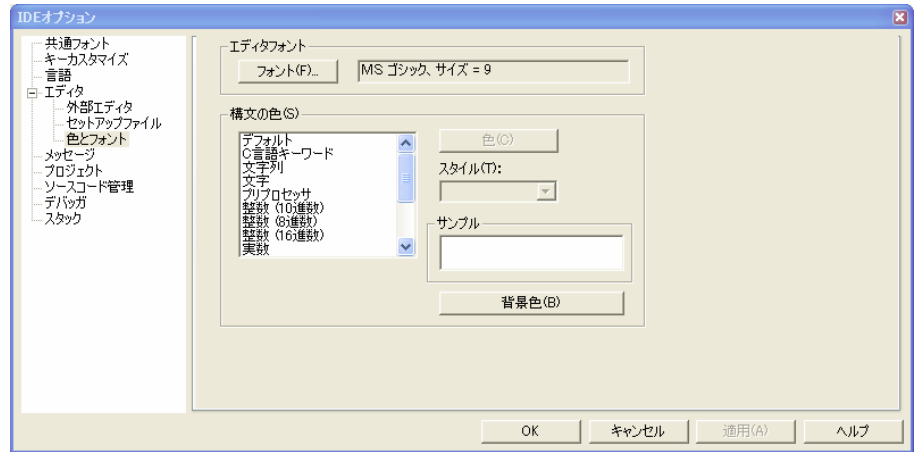


図 62: 【エディタ色とフォント】 オプション

このページを使用して、エディタウィンドウのテキストに使用する色とフォントを指定します。アセンブラおよび C/C++ ソースコードの構文強調表示を制御するキーワードは、それぞれ `syntax_icc.cfg` と `syntax_asm.cfg` のファイルに記述されています。これらのファイルは、`arm%config` ディレクトリにあります。

エディタフォント

【フォント】 ボタンをクリックして、標準【フォント】ダイアログを開き、エディタウィンドウで使用するフォントとそのサイズを選択できます。

構文の色

構文の要素をリストで選択して、色とスタイルを設定します。

- | | |
|-------------|--|
| 色 | 選択可能な色が一覧表示されます。リストから [カスタム] を選択して、自分の色を定義します。 |
| スタイル | 選択した要素について、 [ノーマル] 、 [太字] 、 [斜体] のスタイルを選択します。 |
| サンプル | 選択した要素の現在の外観を表示します。 |
| 背景色 | エディタウィンドウの背景色をクリックして設定します。 |

注：**[ユーザキーワード]** 構文要素は、カスタムキーワードファイルにリストしたキーワードを参照します 138 ページの *カスタムキーワードファイルの使用を参照*。

[メッセージ] オプション

[メッセージ] オプションは、**[ツール] > オプション**を選択すると使用できます。

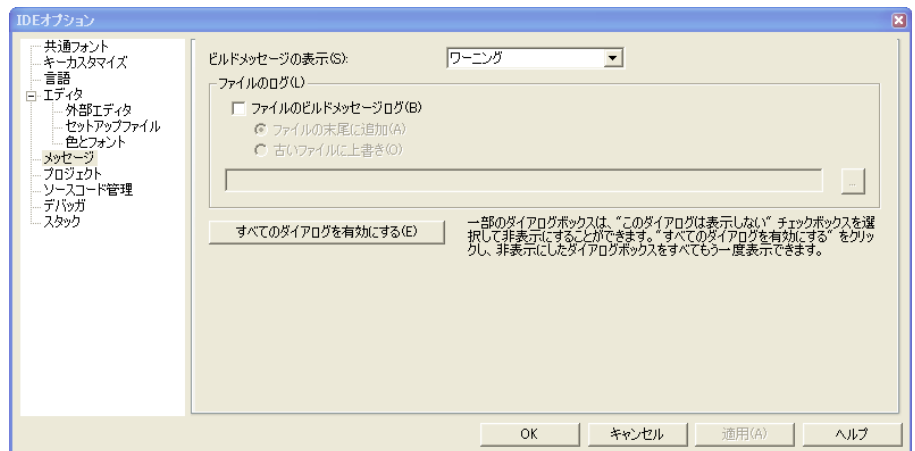


図 63: [メッセージ] オプション

このページを使用して、**[ビルド]** メッセージウィンドウでの出力内容を選択します。

ビルドメッセージの表示

[ビルド] メッセージウィンドウに表示する出力量を選択します。以下から選択します。

すべて	コンパイラとリンカの情報を含むすべてのメッセージを表示します。
メッセージ	メッセージ、ワーニング、エラーを表示します。
ワーニング	ワーニングやエラーを表示します。
エラー	エラーのみ表示します。

ファイルにログ

[ビルドメッセージをファイルにログ] オプションを選択すると、ビルドメッセージがログファイルに書き込まれます。以下から選択します。

- ファイルの末尾に追加 指定したファイルの最後にメッセージを追加します。
- 古いファイルに上書き 指定したファイルの内容を置換します。

使用するファイル名をテキストボックスに入力します。参照ボタンを使用して選択することもできます。

すべてのダイアログを有効にする

たとえば、[次回からこのダイアログを表示しない] チェックボックスを選択して非表示にしたすべてのダイアログボックスを有効にできます。

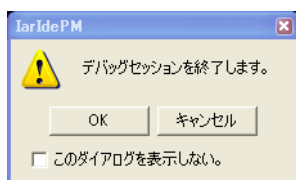


図 64: [このダイアログは表示しない] オプションを含む [メッセージ] ダイアログボックス

【プロジェクト】 オプション

【プロジェクト】 オプションは、【ツール】 > オプションを選択すると使用できます。

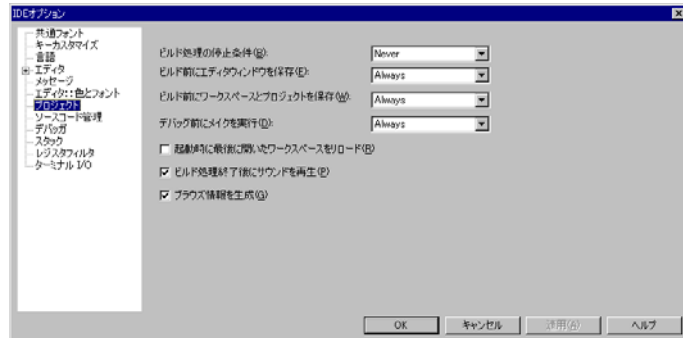


図 65: プロジェクトオプション

このページを使用して、【作成】と【ビルド】コマンドにオプションを設定します。

ビルド処理の停止条件

ビルド処理の停止条件を選択します。以下から選択します。

停止しない	停止しません。
ワーニング	ワーニングやエラーで停止します。
エラー	エラーで停止します。

ビルド前にエディタウィンドウを保存

ビルド処理の前にエディタウィンドウを保存するタイミングを選択します。以下から選択します。

保存しない	保存しません。
保存前に確認する	保存前に確認します。
常に保存する	【作成】 / 【ビルド】の実行前に常に保存します。

ビルド前にワークスペースとプロジェクトを保存

ビルド処理の前に、ワークスペースとインクルードされたプロジェクトをいつ保存するかを選択します。以下から選択します。

- 保存しない 保存しません。
- 保存前に確認する 保存前に確認します。
- 常に保存する [作成] / [ビルド] の実行前に常に保存します。

デバッグ前にメイクを実行

デバッガセッションを開始するにあたってメイク処理を実行するタイミングを選択します。以下から選択します。

- 保存しない デバッグの前にメイク処理を実行しません。
- 保存前に確認する メイク処理を実行する前に確認します。
- 常に保存する デバッグの前にメイク処理を常に実行します。

起動時に最後に開いたワークスペースをリロード

次に IAR Embedded Workbench IDE を起動するときに、前回アクティブだったワークスペースを自動的にロードします。

ビルド処理終了後にサウンドを再生

ビルド処理の完了時に音を再生します。

ブラウザ情報を生成

[ソースブラウザ] ウィンドウの使用を有効にします (50 ページの [ソースブラウザ] ウィンドウを参照)。

【ソースコード管理】 オプション

【ソースコード管理】 オプションは、【ツール】 > オプションを選択すると使用できます。

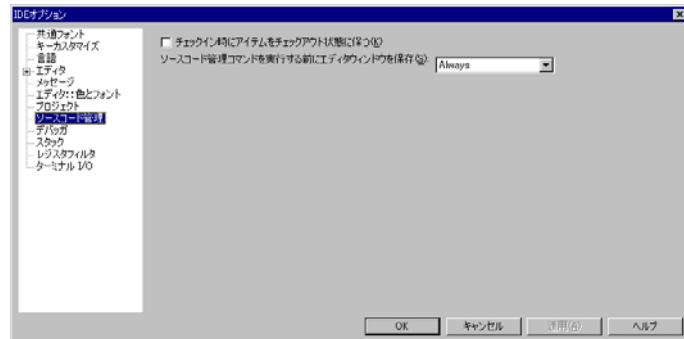


図 66: 【ソースコード管理】 オプション

このページを使用して、IAR Embedded Workbench プロジェクトと SCC プロジェクト間の相互作用を設定します。

チェックイン時にアイテムをチェックアウト状態に保つ

【ファイルのチェックイン】 ダイアログボックスの【チェックアウトを維持】オプションのデフォルト設定を指定します (56 ページの【ファイルのチェックイン】ダイアログボックスを参照)。

ソースコード管理コマンドを実行する前にエディタウィンドウを保存

ソースコード管理コマンドの実行前にエディタウィンドウを保存するかどうかを決定します。以下から選択します。

保存しない	ソースコード管理コマンドを実行する前にエディタウィンドウを保存しません。
保存前に確認する	ソースコード管理コマンドを実行する前に確認します。
常に保存する	ソースコード管理コマンドを実行する前に常にエディタウィンドウを保存します。

「デバッガ」オプション

「デバッガ」オプションは、「ツール」>オプションを選択すると使用できます。

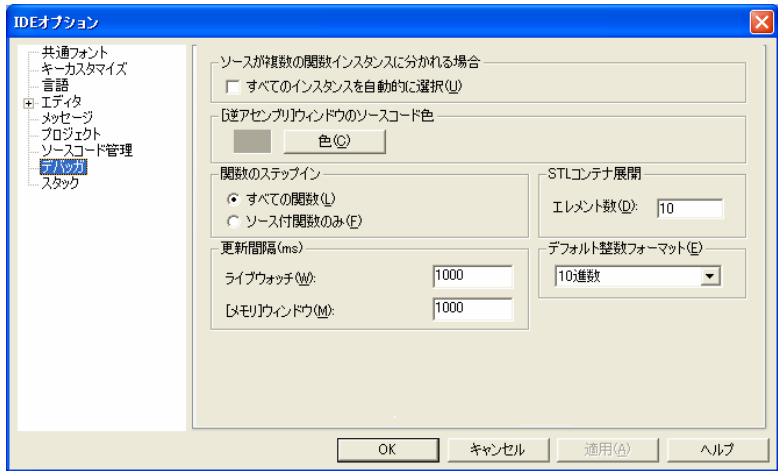


図 67: 「デバッガ」オプション

このページを使用して、デバッガ環境を設定します。

複数の関数インスタンスにソースを分解する場合

一部のソースコード（テンプレートコードなど）は、複数のコードインスタンスに対応しています。このようなコードでソース位置を指定する場合（ソースのブレークポイント設定時など）は、C-SPY ですべてのインスタンスを選択するか、その一部だけを選択するかを指定できます。「すべてのインスタンス」オプションを使用して、C-SPY で最初に確認しないですべてのインスタンスを処理します。

「逆アセンブリ」ウィンドウのソースコード色」ウィンドウ

「色」ボタンをクリックして、「逆アセンブリ」ウィンドウのソースコードの色を選択します。独自の色を定義するには、リストから「カスタム」を選択します。

関数のステップイン

「ステップイン」コマンドの動作を制御します。以下から選択します。

すべての関数

デバッガですべての関数にステップインします。

ソース付関数のみ

デバッガは、ソースコードを認識する関数だけにステップインします。これにより、ライブラリ関数内のコードのステップ実行や、逆アセンブリモードのデバッグを回避することができます。

STL コンテナ展開

コンテナの値を [ウォッチ] ウィンドウなどで展開したときに、最初に表示されるエレメント数を指定します。

更新間隔

[ライブウォッチ] および [メモリ] の各ウィンドウの内容を更新する頻度を指定します。

これらのテキストボックスは、使用している C-SPY ドライバがアプリケーションの実行中にターゲットのシステムメモリにアクセス可能な場合にのみ使用できます。

デフォルト整数フォーマット

[ウォッチ]、[ローカル] ウィンドウおよび関連するウィンドウでのデフォルトの整数フォーマットを選択します。

[スタック] オプション

[スタック] オプションは、[ツール] > オプションまたは [メモリ] ウィンドウのコンテキストメニューから選択すると使用できます。

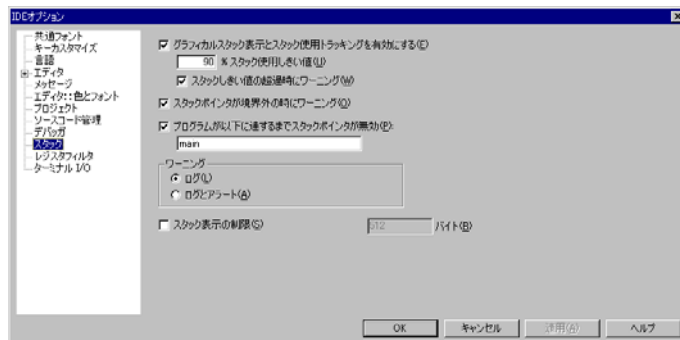


図 68: [スタック] オプション

このページを使用して、[スタック] ウィンドウに固有のオプションを設定します。

グラフィカルスタック表示とスタック使用トラッキングを有効にする

[スタック] ウィンドウ上部のグラフィカルスタックバーを有効にします。スタックオーバーフローの検出も有効になります。スタックバーの詳細やスタックバーが示す内容については、『*ARM® 用 C-SPY® デバッグガイド*』を参照してください。

% スタック使用しきい値

C-SPY がスタックオーバーフローについてワーニングを表示するスタック使用率を指定します。

スタックしきい値の超過時にワーニング

スタック使用量が [% スタック使用しきい値] オプションに指定したしきい値を超えた場合に、C-SPY からワーニングを発生します。

スタックポインタが境界外の時にワーニング

スタックポインタがスタックメモリ範囲外を指したときに、C-SPY からワーニングを発生します。

プログラムが以下に達するまでスタックポインタが無効

アプリケーションコード中でスタックの表示と検証を行う *位置* を指定します。[スタック] ウィンドウでは、実行がこの位置に到達するまでは、スタック使用率情報が表示されません。

デフォルトでは、main 関数に到達するまではスタック使用率が表示されません。main 関数がないアプリケーション（たとえば、アセンブラのみのプロジェクト）の場合、独自の開始ラベルを指定する必要があります。このオプションを選択すると、C-SPY は、毎回のリセット後、指定された位置のブレイクポイントに到達するまでこのブレイクポイントを保持します。

通常は、スタックポインタはシステム初期化コード `cstartup` に設定しますが、最初の命令からスタック使用率をトレースする必要はありません。このオプションを使用することで、アプリケーションのこの部分について誤ったワーニングや誤解を招くスタック表示を回避できます。

ワーニング

ワーニングを出力する場所を選択します。以下から選択します。

- | | |
|-----------|--|
| ログ | [デバッグログ] ウィンドウにワーニングを表示します。 |
| [ログとアラート] | [デバッグログ] ウィンドウ、ワーニングダイアログボックスにワーニングを表示します。 |

スタック表示の制限

〔スタック〕ウィンドウで表示されるメモリ容量を、スタックポインタからのバイト数で指定します。このオプションは、スタックが大きい場合や、スタックの最初の部分の表示だけが必要な場合に便利です。このオプションを使用すると、特にターゲットシステムからのメモリリード速度が遅い場合に、〔スタック〕ウィンドウの表示速度を向上することができます。デフォルトでは、〔スタック〕ウィンドウにはスタック全体、つまりスタックポインタからスタックの最後までが表示されます。デバッガがスタックのメモリ範囲を特定できない場合は、このオプションを選択していない場合でも、表示されるバイト範囲が制限されます。

注：〔スタック〕ウィンドウは、アプリケーションの実行速度には影響ませんが、実行停止時に表示される情報を更新するために、大量のデータをリードする場合があります。

〔レジスタフィルタ〕オプション

〔レジスタフィルタ〕オプションは、デバッガの実行中に〔ツール〕>〔オプション〕を選択すると使用できます。

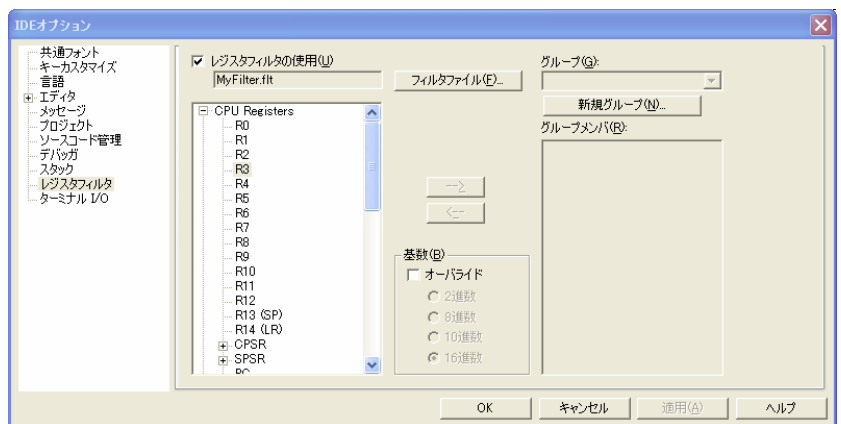


図 69: 〔レジスタフィルタ〕のオプション

このページを使用して、作成済みのグループ別に〔レジスタ〕ウィンドウでレジスタを表示します

レジスタグループについて詳しくは、『ARM® 用 C-SPY® デバッガガイド』を参照してください。

アプリケーション固有のレジスタグループを定義するには、次の手順に従います。

- 1 [ツール] > [オプション] > [レジスタフィルタ] を選択します。
- 2 新しいグループのファイル名を指定します。
- 3 [新規グループ] をクリックして、グループ名を指定します。
- 4 矢印ボタンを使用して、含めるレジスタを選択します。
- 5 オプションで、デフォルトの整数の基本型をオーバーライドできます。
- 6 新しいグループが [レジスタ] ウィンドウで使用できるようになりました。

レジスタフィルタの使用

レジスタフィルタの使用を有効にします。

フィルタファイル

フィルタファイルの選択や新規作成を行うダイアログボックスを表示します。

グループ

フィルタファイルに使用可能なレジスタグループをすべて表示します。また、新しいレジスタグループも表示されます。

新規グループ

クリックすると、新しいレジスタグループを作成します。

グループメンバ

[グループ] ドロップダウンリストで現在選択されているグループのレジスタを表示します。

グループにレジスタを追加するには、使用可能なレジスタのリストからレジスタを選択して左に追加し、矢印ボタンを使用してそれらを移動します。

グループからレジスタを削除するには、削除するレジスタを選択し、矢印ボタンを使用してそれらを移動します。

基数

デフォルトの整数基数をオーバーライドします。

【ターミナル I/O】 オプション

【ターミナル I/O】 オプションは、デバッガの実行中に [ツール] > 【オプション】 を選択すると使用できます。

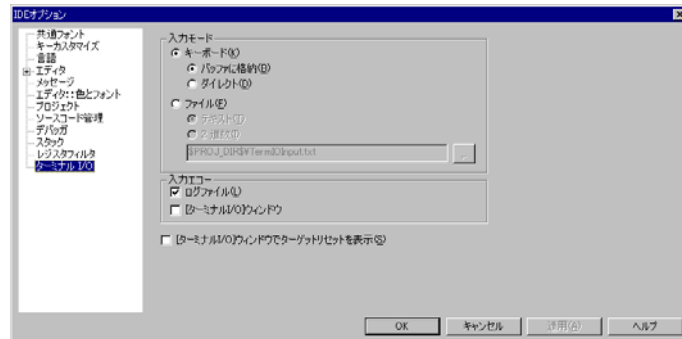


図 70: 【ターミナル I/O】 オプション

このページを使用して、C-SPY のターミナル I/O 機能を設定します。

入力モード

ターミナル I/O の入力を読み取る方法を制御します。

キーボード キーボードからの入力文字を読み込みます。以下から選択します。

バッファに格納: 入力された文字をバッファに格納します。

ダイレクト: 入力された文字をバッファに格納しません。

ファイル ファイルからの入力文字を読み込みます。以下から選択します。

テキスト: テキストファイルから入力文字を読み込みます。

バイナリ: バイナリファイルから入力文字を読み込みます。

参照ボタンを使用して入力ファイルを選択することもできます。

入力エコー

入力文字をエコーするかどうか、およびどこにエコーするかを指定します。以下から選択します。

- **ログファイル**。オプション [デバッグ] > [ログ] > [ログの有効化] を有効にしておく必要があります
- **【ターミナル I/O】 ウィンドウ**

[ターミナル I/O] ウィンドウでターゲットを表示

ターゲットのリセット時に、C-SPY の [ターミナル I/O] ウィンドウでメッセージを表示します。

[ツールの設定] ダイアログボックス

[ツールの設定] ダイアログボックスは、[ツール] メニューから表示します。

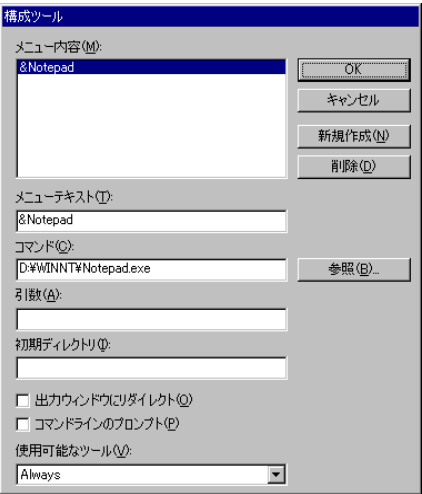


図 71: [ツールの設定] ダイアログボックス

このダイアログボックスを使用して、以下のように [ツール] メニューに追加するツールを指定します。

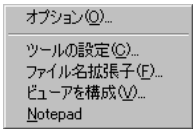


図 72: カスタマイズした [ツール] メニュー

注：標準ビルドツールチェーンに外部ツールを追加する場合は、66 ページのツールチェーンの拡張を参照してください。



引数で変数を使用することによって、コマンドラインのレビジョン管理システムとのインタフェースや、選択したファイルに対して外部ツールを実行するなどの便利なツールを設定できます。

コマンドラインのコマンドやバッチファイルを [ツール] メニューに追加する：

- 1 **[コマンド]** テキストボックスで、cmd.exe コマンドシェルを指定または検索します。
- 2 **[引数]** テキストボックスで、コマンドラインコマンドかバッチファイル名を指定します。

[引数] のテキストは、以下に示すように指定する必要があります。

/C name

ここで、name は、実行するコマンドかバッチファイルの名前です。

/c オプションは、実行後にシェルを終了するように指定し、ツールの終了を IDE が検出できるようにします。

例については、28 ページの *コマンドラインコマンドの追加* を参照してください。

新規作成

このダイアログボックスを使用して設定する新規メニューコマンドにスタブを作成します。

削除

[メニュー内容] リストで選択されたコマンドを削除します。

メニュー内容

定義したすべてのメニューコマンドを一覧表示します。

メニューテキスト

メニューコマンドの名前を指定します。& という記号を名前のどこかに追加すると、その後の文字（この例では n）がこのコマンドのニーモニックキーとして表示されます。指定したテキストが、**[メニュー内容]** リストに反映されます。

コマンド

メニューからコマンドを選択したときに実行されるツールとそのパスを指定します。参照ボタンを使用して選択することもできます。

引数

オプション：コマンドの引数を指定します。

初期ディレクトリ

ツールの初期作業ディレクトリを指定します。

出力ウィンドウにリダイレクト

ツールから [メッセージ] ウィンドウの [ツール出力] ページにコンソール出力をすべて送信するよう指定します。このオプションを指定して起動したツールは、キーボードなどによるユーザ入力を受け付けることはできません。

入力が必要なツールや、実行するコンソールに特別な条件があるツールは、このオプションを指定すると動作しません。

コマンドラインのプロンプト

コマンドを [ツール] メニューから選択したときに、コマンドライン引数の指定を指示するプロンプトを表示します。

ツール使用可能時

ツールが使用可能なコンテキストを指定します。以下から選択します。

- 常時
- デバッグ時
- デバッグ時以外

[ファイル名拡張子] ダイアログボックス

[ファイル名拡張子] ダイアログボックスは、[ツール] メニューから表示します。

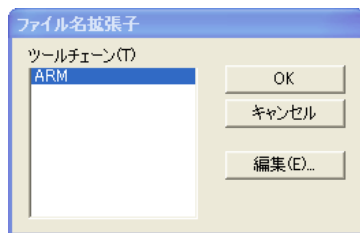


図 73: [ファイル名の拡張子] ダイアログボックス

このダイアログボックスを使用して、ビルドツールで認識されるファイル名拡張子をカスタマイズします。これは、ファイル名の拡張子が異なるソースファイルが多数ある場合に便利です。

ツールチェーン

ホストコンピュータ上に IAR Embedded Workbench がインストールされたツールチェーンを一覧表示します。ファイル名拡張子をカスタマイズするツールチェーンを選択します。

* 文字は、ユーザ定義のオーバーライドを示します。* 文字がない場合は、出荷時設定が使用されます。

編集

[ファイル名拡張子のオーバーライド] ダイアログボックスを表示します (153 ページの [ファイル名拡張子のオーバーライド] ダイアログボックスを参照)。

[ファイル名拡張子のオーバーライド] ダイアログボックス

[ファイル名拡張子のオーバーライド] ダイアログボックスは、[ファイル名拡張子] ダイアログボックスから表示します。

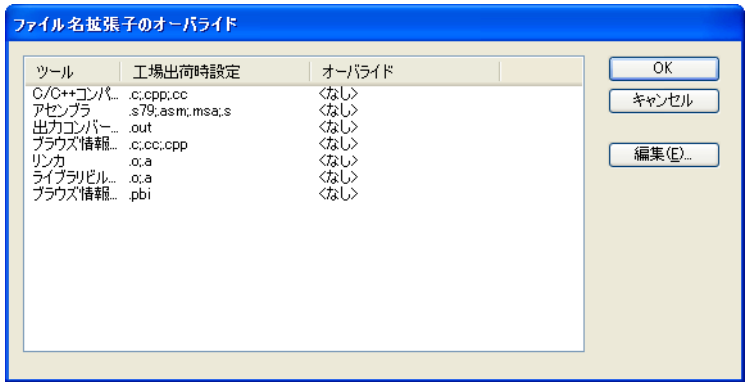


図 74: [ファイル名拡張子のオーバーライド] ダイアログボックス

このダイアログボックスには、ビルドツールで認識されるファイル名拡張子が一覧表示されます。

表示エリア

このエリアには以下の列が含まれます。

- | | |
|---------|---|
| ツール | ビルドチェーンで使用可能なツール。 |
| 工場出荷時設定 | ビルドツールによりデフォルトで認識されるファイル名拡張子。 |
| オーバーライド | デフォルトへのオーバーライドがあった場合に、ビルドツールにより認識されるファイル名拡張子。 |

編集

選択したツールについて **「ファイル名拡張子の編集」** ダイアログボックスが表示されます。

「ファイル名拡張子の編集」ダイアログボックス

「ファイル名拡張子の編集」 ダイアログボックスは、**「ファイル名拡張子のオーバーライド」** ダイアログボックスから表示します。



図 75: 「ファイル名拡張子の編集」ダイアログボックス

このダイアログボックスには、IDE で認識されるファイル名拡張子が一覧表示され、新しいファイル名拡張子を追加することができます。

工場出荷時設定

デフォルトで認識されるファイル名拡張子を一覧表示します。

オーバーライド

認識させたいファイル名拡張子を指定します。拡張子が複数の場合はコンマかセミコロンで区切ります。また、最初のピリオドも含めて入力する必要があります。

「ビューアの設定」ダイアログボックス

「ビューアの設定」ダイアログボックスは、「ツール」メニューから表示します。

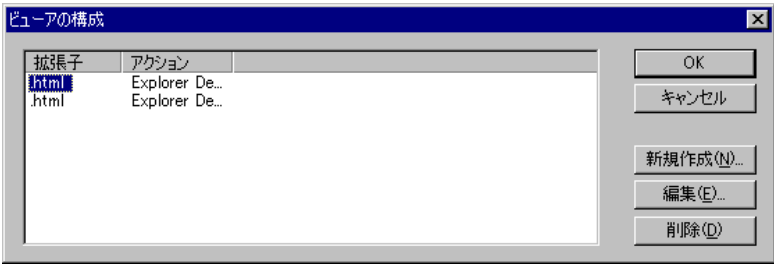


図 76: 「ビューアの設定」ダイアログボックス

このダイアログボックスには、IAR Embedded Workbench が処理できる文書フォーマットとビューアアプリケーション間のデフォルトの関連に対するオーバーライドが一覧表示されます。

表示エリア

このエリアには以下の列が含まれます。

拡張子	IAR Embedded Workbench が処理できる、明示的に定義された文書フォーマットのファイル名拡張子。
アクション	文書タイプを開くときに使用されるビューアアプリケーション。[デフォルトエクスプローラ] は、Windows Explorer の指定タイプに関連するデフォルトのアプリケーションが使用されるということです。

新規作成

「ビューア拡張子の編集」ダイアログボックスを参照。

編集

「ビューア拡張子の編集」ダイアログボックスを参照。

削除

選択したファイル名拡張子とビューアアプリケーション間の関連を削除します。

【ビューア拡張子の編集】 ダイアログボックス

【ファイル名拡張子の編集】 ダイアログボックスは、【ビューアの設定】 ダイアログボックスから表示します。



図 77: 【ビューア拡張子の編集】 ダイアログボックス

このダイアログボックスを使用して、新規文書タイプを開いたり、既存の文書タイプの設定を編集する方法を指定します。

ファイル名拡張子

区切り文字のピリオド (.) も含めて、文書タイプのファイル名拡張子を指定します。

アクション

【ファイル名の拡張子】 テキストボックスで指定したファイル名拡張子を持つ文書を開く方法を選択します。以下から選択します。

- | | |
|-----------------|--|
| 内蔵テキストエディタ | 指定したタイプのすべての文書を IAR Embedded Workbench のテキストエディタで開きます。 |
| エクスプローラの関連付けを使用 | 指定したタイプのすべての文書を、Windows Explorer の指定タイプに関連付けられたデフォルトのアプリケーションで開きます。 |
| コマンドライン | 指定したタイプのすべての文書を、入力または指定したビューアアプリケーションで開きます。希望する任意のコマンドラインオプションをツールに追加できます。 |

[ウィンドウ] メニュー

[ウィンドウ] メニューでは、IDE ウィンドウの操作や画面上での配置変更のコマンドを提供します。

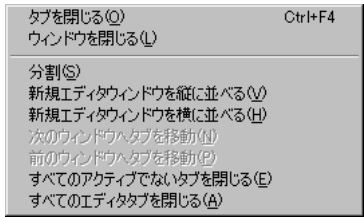


図 78: [ウィンドウ] メニュー

[ウィンドウ] メニューの最後のセクションには、画面で開かれているウィンドウのリストが一覧表示されます。リストからウィンドウを選択すると、そのウィンドウに切り替ります。

以下のコマンドがあります。

タブを閉じる	アクティブなタブを閉じます。
ウィンドウを閉じる CTRL+F4	アクティブなエディタウィンドウを閉じます。
分割	ウィンドウを縦または横方向に 2 つか 4 つに分割し、同一ファイルの異なる部分を同時に表示します。
新規エディタウィンドウを縦に並べる	新しい空白ウィンドウを、現在のエディタウィンドウの横に表示します。
新規エディタウィンドウを横に並べる	新しい空白ウィンドウを、現在のエディタウィンドウの下に表示します。
次のウィンドウヘタブを移動	現在のウィンドウのすべてのタブを次のウィンドウに移動します。
前のウィンドウヘタブを移動	現在のウィンドウのすべてのタブを前のウィンドウに移動します。
すべてのアクティブでないタブを閉じる	アクティブなタブ以外のすべてのタブを閉じます。
すべてのエディタタブを閉じる	エディタウィンドウで表示されているすべてのタブを閉じます。

【ヘルプ】メニュー

【ヘルプ】メニューには、IAR Embedded Workbench に関するヘルプと、IDE のユーザインタフェースのバージョン番号が表示されます。

インフォメーションセンタには【ヘルプ】メニューからもアクセスできます。インフォメーションセンタは、チュートリアルやプロジェクトのサンプル、ユーザガイド、サポート情報、リリースノートなど、プロジェクト開発の開始時や作業中に必要な情報リソースに簡単にアクセス可能にするナビゲーションシステムです。また、IAR Systems の Web サイトで役に立つセクションへのショートカットも提供します。

一般オプション

この章では、IAR Embedded Workbench® IDE の一般オプションについて説明します。

オプションの設定方法の詳細については、61 ページの *オプションの設定* を参照してください。

一般オプションの説明

このセクションでは、[一般オプション] カテゴリのオプションについて説明します。

IDE の一般オプションを設定するには、以下の手順に従います。

- 1 [プロジェクト] > [オプション] を選択して、[オプション] ダイアログボックスを表示します。
- 2 [カテゴリ] リストで [一般オプション] 選択します。
- 3 すべての設定をデフォルトの出荷時の設定に戻すには、[出荷時設定] ボタンをクリックします。

ターゲットオプション

[ターゲット] オプションでは、IAR C/C++ コンパイラおよびアセンブラのターゲット固有の機能を指定します。



図 79: 一般ターゲットオプション

プロセッサ選択

派生プロセッサを選択します。

コア	使用しているプロセッサコアです。派生品の詳細については、『 <i>ARM 用 IAR C/C++ 開発ガイド</i> 』を参照してください。
デバイス	使用しているデバイスです。デバイスを選択すると、デフォルトの C-SPY® デバイス記述ファイルが自動的に決定されます。デフォルトファイルのオーバーライド方法については、『 <i>ARM® 用 C-SPY® デバッグガイド</i> 』を参照してください。

エンディアンモード

プロジェクトのバイトオーダーを選択します。

リトル	最も低いバイトはメモリの最も低いアドレスに格納されます。最も高いバイトは一番重要です。最も高いアドレスに格納されます。
ビッグ	一番低いアドレスに一番重要なバイトが保持されます。一方、最も高いアドレスには重要度の低いバイトが保持されます。ビッグエンディアンモードには2つのうちどちらかを選択します。 BE8 を選択すると、データはビッグエンディアン、コードはリトルエンディアンになります。 BE32 を選択すると、データとコードの両方がビッグエンディアンコードになります。

FPU

浮動小数点ユニットを選択します。

なし (デフォルト)	ソフトウェア浮動小数点ライブラリが使用されます。
VFPv2	アーキテクチャ VFPv2 に準拠した VFP ユニット。
VFPv3	アーキテクチャ VFPv3 に準拠した VFP ユニット。
VFPv3 + NEON	アーキテクチャ VFPv3 と NEON ユニットに準拠する VFP ユニット。
VFPv4	アーキテクチャ VFPv4 に準拠した VFP ユニット。

- VFPv4 + NEON** アーキテクチャ VFPv4 と NEON ユニットに準拠する VFP ユニット。
- VFPv9-S** CPU コアの ARM9E ファミリで使用可能な VFPv2 アーキテクチャ。そのため、このコプロセッサを選択することは、VFPv2 アーキテクチャを選択することと同じです。

VFP コプロセッサを選択することで、ソフトウェア浮動小数点ライブラリの使用を、サポートされたすべての浮動小数点演算にオーバーライドします。

出力

【出力】 オプションによって、出力ファイルのタイプが決まります。また、実行可能ファイル、オブジェクトファイル、リストファイルの保存先も指定できます。

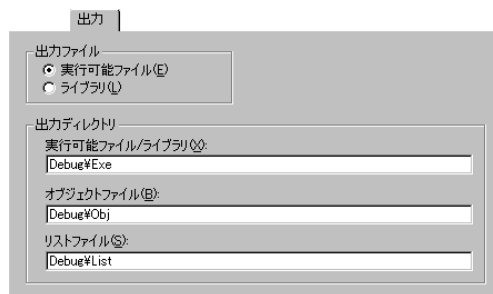


図 80: 【出力】 オプション

出力ファイル

出力ファイルのタイプを選択します。

- 実行可能ファイル** (デフォルト) ビルドプロセスの結果、リンカがアプリケーション（実行可能出力ファイル）を作成します。この設定を使用した場合は、リンカのオプションを【オプション】ダイアログボックスで設定できます。出力を作成する前に、該当するリンカオプションを設定する必要があります。
- ライブラリ** ビルドプロセスの結果、ライブラリビルダがライブラリ出力ファイルを作成します。この設定を使用した場合は、ライブラリビルダオプションを【オプション】ダイアログボックスで設定でき、【リンカ】は、カテゴリリストから削除されます。ライブラリを作成する前に、オプションを設定する必要があります。

出力ディレクトリ

目的のディレクトリのパスを指定します。プロジェクトディレクトリとの相対パスを指定します。以下を指定できます。

実行可能ファイル/ ライブラリ	実行可能ファイルやライブラリファイルのデフォルトディレクトリをオーバーライドします。プロジェクトの実行可能ファイルを保存するディレクトリの名前を入力します。
オブジェクトファイル	オブジェクトファイルのデフォルトのディレクトリをオーバーライドします。プロジェクトのオブジェクトファイルを保存するディレクトリの名前を入力します。
リストファイル	リストファイルのデフォルトのディレクトリをオーバーライドします。プロジェクトのリストファイルを保存するディレクトリの名前を入力します。

ライブラリ設定

〔ライブラリ設定〕 オブジェクトによって、使用するライブラリが決まります。

ライブラリ設定

ライブラリ(L):
Normal

説明
C/C++ランタイムライブラリの通常の設定を使用します。ロケールインタフェースなし、Cロケール、ファイル記述子サポートなし、printf/scanfでのマルチバイト文字なし、strtodでの16進数対手動小数点数なし。

設定ファイル(C):
\$TOOLKIT_DIR\$%INC%c%DLib_Config_Normal.h

低レベルインタフェースのライブラリ実装(B)
☐ なし(N)
☒ セミホスティング(S)
☐ IARブレークポイント(I)

stdout/stderr
☒ セミホスティング経由
☐ SWO経由

CMSIS
☒ CMSISを使用する
☐ DSPライブラリ

図81: 〔ライブラリ構成〕 オプション

ランタイムライブラリ、ライブラリ設定、これらのライブラリ設定が提供するランタイム環境、可能なカスタマイズについては、『ARM 用 IAR C/C++ 開発ガイド』を参照してください。

ライブラリ

使用するランタイムライブラリを選択します。使用可能なライブラリについては、『*ARM 用 IAR C/C++ 開発ガイド*』を参照してください。

実際に使用されるライブラリオブジェクトファイルとライブラリ設定ファイルの名前は、それぞれ **【ライブラリ】** テキストボックスと **【設定ファイル】** テキストボックスに表示されます。

設定ファイル

使用されるライブラリ設定を表示します。ライブラリ設定ファイルは、プロジェクトの設定に応じて自動的に選択されます。**【カスタム】** を **【ライブラリ】** ドロップダウンリストで選択した場合は、ライブラリ設定ファイルを指定する必要があります。

低レベルインタフェースのライブラリ実装

Cortex-M の場合、以下から選択します。

- | | |
|---|--|
| なし | ライブラリで利用可能な I/O の低レベルサポートはなし。独自の <code>__write</code> 関数を用意して、ライブラリの一部の I/O 関数を使用する必要があります。 |
| セミホスティング
およびセミホス
ティングを経由し
た <code>stdout/stderr</code> | BKPT 命令を使用するセミホスティング I/O。 |
| セミホスティング
および SWO を経由
した <code>stdout/stderr</code> | SWO インタフェース（一部の J-Link デバッグプローブで使用可能）が使用されている <code>stdout</code> および <code>stderr</code> 出力以外のすべての関数に BKPT 命令を使用する <code>Semihosted I/O</code> 。これは、アプリケーションがデータ転送の実行を停止する必要のない、きわめて高速のメカニズムを意味します。 |
| IAR ブレークポイント | 使用できません。 |

他のコアの場合、以下から選択してください。

なし	ライブラリで利用可能な I/O の低レベルサポートはなし。独自の <code>__write</code> 関数を用意して、ライブラリの一部の I/O 関数を使用する必要があります。
セミホスティング	svc 命令（以前の SWI）を使用するセミホスティング I/O。
IAR ブレークポイント	IAR 独自の派生セミホスティング。svc 命令を使用しないため svc ベクタ上にブレークポイントを設定する必要がありません。RTOS など、自身のために svc ベクタを必要とするアプリケーションをデバッグするときに便利です。この方法は、パフォーマンスの向上にも有効です。ただし、他のベンダ製ツールを使用してビルドされたアプリケーション、ライブラリ、オブジェクトファイルでは動作しません。

CMSIS

CMSIS サポートを有効にするには、以下のオプションを使用します。

CMSIS を使用	<p>CMSIS ヘッダファイルをコンパイラのインクルードパスに追加します</p> <p>アプリケーションのソースコードに CMSIS ヘッダファイルが明示的に含まれる場合、このオプションは使用しないでください。</p> <p>このオプションは、Cortex-M デバイスでのみ使用できます。</p>
DSP ライブラリ	<p>アプリケーションを CMSIS DSP ライブラリにリンクします。このオプションは、Cortex-M デバイスでのみ使用できます。</p>

ライブラリオプション

【ライブラリオプション】では、printf と scanf のフォーマットを選択します。

図 82: ライブラリオプション

フォーマットの機能の詳細については、『ARM 用 IAR C/C++ 開発ガイド』を参照してください。

printf フォーマット

【自動】が選択されている場合、リンカはコンパイラからの情報に基づいて、printf 関連の機能に適切なフォーマットを自動的に選択します。

すべての printf 関連の機能に対するデフォルトのフォーマットをオーバーライドする（wprintf の派生型を除く）には、以下から選択します。

フル、フル（マルチバイト）、大、大（マルチバイト）、小、小（マルチバイト）、極小

アプリケーションの要件に合ったフォーマットを選択してください。

scanf のフォーマット

【自動】が選択されている場合、リンカはコンパイラからの情報に基づいて、scanf 関連の機能に適切なフォーマットを自動的に選択します。

すべての scanf 関連の機能に対するデフォルトのフォーマットをオーバーライドする（wscanf の派生型を除く）には、以下から選択します。

フル、フル（マルチバイト）、大、大（マルチバイト）、小、小（マルチバイト）

アプリケーションの要件に合ったフォーマットを選択してください。

バッファターミナル出力

プログラムの実行中に、それぞれの新しい文字をすぐに **C-SPY** の [ターミナル I/O] ウィンドウに出力するのではなく、ターミナル出力をバッファに格納します。このオプションは、通信速度が遅いデバッグシステムを使用する場合に便利です。

MISRA-C

この **[MISRA-C:1998]** および **[MISRA-C:2004]** オプションは、ソースコードの MISRA-C 規則からの逸脱を IDE が確認する方法を制御します。この設定は、コンパイラとリンカの両方に使用されます。

特定のオプションの詳細は、『*IAR Embedded Workbench® MISRA-C:2004 リファレンスガイド*』、または [ヘルプ] メニューから 『*IAR Embedded Workbench® MISRA-C:1998 リファレンスガイド*』を参照してください。

コンパイラオプション

この章では、IAR Embedded Workbench® IDE のコンパイラオプションについて説明します。

オプションの設定方法の詳細については、61 ページの *オプションの設定* を参照してください。

コンパイラオプションの説明

このセクションでは、[C/C++ コンパイラ] カテゴリのオプションについて説明します。

IDE のコンパイラオプションを設定するには、以下の手順に従います。

- 1 [プロジェクト] > [オプション] を選択して、[オプション] ダイアログボックスを表示します。
- 2 [カテゴリ] リストで [C/C++ コンパイラ] を選択します。
- 3 すべての設定をデフォルトの出荷時の設定に戻すには、[出荷時設定] ボタンをクリックします。

複数ファイルのコンパイル

特定のコンパイラオプションを設定する前に、複数ファイルのコンパイルを使用するかどうかの指定ができます。これは、最適化のテクニックの 1 つです。[複数ファイルのコンパイル] では、[ワークスペース] ウィンドウで選択したプロジェクトファイルのグループに対して複数ファイルのコンパイルを有効にします。

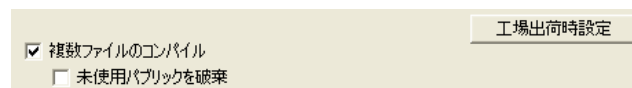


図 83: 複数ファイルのコンパイル

このオプションは、プロジェクト全体で使用するほか、ファイルのグループごとに使用することもできます。このようなグループ内のすべての C/C++ ソースファイルが、1 回のコンパイラの呼出しで一緒にコンパイルされます。

未使用パブリックを破棄

コンパイルユニットからの未使用のパブリック関数および変数をすべて破棄します。

つまり、選択したグループに含まれるファイルはすべて、そのグループまたは任意のオプションが設定された直近の親ノードに設定されているコンパイラオプションを使用してコンパイルされます。1 つまたは複数のファイルに対してオーバーライドするコンパイラオプションはすべて、ビルド時に無視されます。これは、グループコンパイルでオプションのセットを 1 つだけ使用する必要があるためです。

複数ファイルのコンパイルがどのように [ワークスペース] ウィンドウに表示されるかについては、42 ページの [ワークスペース] ウィンドウを参照してください。

複数ファイルのコンパイルおよび未使用パブリック関数の破棄の詳細については、『ARM 用 IAR C/C++ 開発ガイド』を参照してください。

言語 1

[言語 1] オプションによって、使用するプログラミング言語と有効にする拡張を決定します。

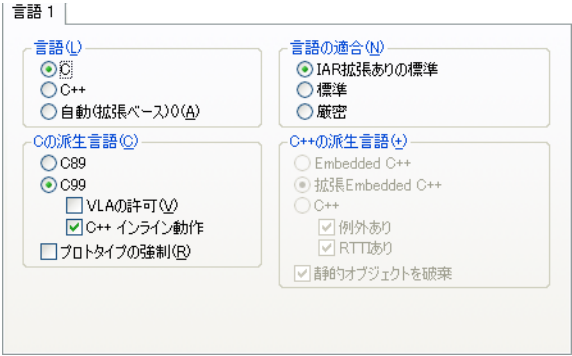


図 84: コンパイラの言語オプション

サポートされている言語、派生言語、言語拡張の詳細については、『ARM 用 IAR C/C++ 開発ガイド』を参照してください。

言語

C または C++ のコンパイラサポートを決定します。

C (デフォルト) コンパイラでソースコードを C として扱います。つまり、C++ 固有の機能は使用できません。

C++ コンパイラでソースコードを C++ として扱います。

自動 コンパイルするファイルのファイル名拡張子に応じて、言語サポートが自動的に決定されます。

c: このファイル名拡張子を持つファイルは C ソースファイルとして扱われます。

cpp: このファイル名拡張子を持つファイルは C++ ソースファイルとして扱われます。

C の派生言語

C がサポートされている言語の場合に、派生言語を選択します。

C89 C 規格ではなく C89 規格を有効にします。この設定は、MISRA C チェックが有効になっている場合は必須です。

C99 C99 規格 (C 規格) を有効にします。これはコンパイラで使用される標準規格で、C89 よりも厳密です。C89 に固有の機能は使用できません。このほかに、以下を選択してください。

VLA の許可: C99 可変長配列の使用を許可します。

C++ インライン動作: C 規格のソースコードファイルをコンパイルする際に C++ インライン動作を有効にします。

プロトタイプの強制

コンパイラが、すべての関数に正しいプロトタイプがあるかどうかを強制的に検証するようにします。すなわち、ソースコードに以下のいずれかが含まれているとエラーが出力されます。

- 宣言を持たない関数の関数呼出し、または Kernighan & Ritchie C 形式の宣言を持つ関数の呼び出し
- 先にプロトタイプが宣言されていない public 関数の関数定義
- プロトタイプを含まない型の関数ポインタによる間接的な関数呼出し

C++ の派生言語

C++ がサポート言語の場合に、派生言語を選択します。

Embedded C++	コンパイラでソースコードを Embedded C++ として扱います。つまり、クラスやオーバーロードといった C++ 固有の機能を使用できます。
拡張 Embedded C++	ソースコードで名前空間や標準テンプレートライブラリを有効にします。
C++	コンパイラでソースコードを標準の C++ として扱います。以下から選択します。 例外あり : C++ 言語で例外サポートを有効にします。 RTTI あり : C++ 言語でランタイム型情報 (RTTI) のサポートを有効にします。
静的オブジェクトを破棄	コンパイラはコードを出力して、プログラム終了時に破棄が必要な C++ 静的変数を破壊します。

言語の適合

標準の C/C++ 言語にどれくらい厳密に準拠するかを制御します。

標準 (IAR 拡張あり)	ARM 固有のキーワードを標準の C/C++ 言語に対する拡張として受け入れます。IDE では、この設定はデフォルトで有効です。
標準	IAR システムズの拡張を無効にしますが、選択した C/C++ の派生言語に厳密に準拠するわけではありません。非常に役立つ C/C++ への緩和対応もそのまま利用できます。
厳密	選択した C/C++ の派生言語に厳密に準拠します。この設定は C/C++ に役立つ拡張や緩和措置の数多くを無効にします。

言語 2

【言語 2】 オプションは、一部の言語拡張の使用を制御します。

言語 2

‘CHAR’ の型(P)

☐ 符号付
☒ 符号なし

浮動小数点数動作(F)

☒ 厳密な適合
☐ 緩和(サイズ縮小/高速化)

☐ マルチバイト文字サポートを有効にする(E)

図 85: コンパイラの言語オプション

‘CHAR’ の型

コンパイラは通常、char の型を unsigned char として解釈します。【‘CHAR’ の型】を選択すると、コンパイラは別のコンパイラとの互換目的などで char 型を signed char として認識します。

注：ランタイムライブラリは、符号なしの単純な文字型を使用してコンパイルされています。【符号付】オプションを選択すると、単純な符号なしの文字型を使用するライブラリ機能への参照は機能しなくなります。

浮動小数点動作

浮動小数点動作を制御します。以下から選択します。

厳密な適合

コンパイラで、浮動小数点式について C および浮動小数点の標準に厳密に準拠します。

緩和

コンパイラで、言語規則を緩和して、浮動小数点式をより積極的に最適化します。このオプションは、以下の条件を満たす浮動小数点式のパフォーマンスを向上させます。

- 式に単精度および倍精度の値が両方含まれている。
- 倍精度の値が精度を失わずに単精度に変換できる。
- 式の結果は単精度に変換されます。

倍精度の代わりに単精度で計算を実行すると、精度が失われることがあります。

マルチバイトサポートを有効にする

デフォルトでは、マルチバイト文字を C や Embedded C++ のソースコードで使用することはできません。[マルチバイト文字サポートを有効にする]を使用すると、ソースコード内のマルチバイト文字が、ホストコンピュータのデフォルトのマルチバイト文字サポート設定に従ってアセンブラにより解釈されます。

マルチバイト文字は、C/C++ 形式のコメント、文字列定数、文字定数で使用できます。これらはそのまま生成コードに移動します。

コード

[コード] オプションは、コンパイラのコード生成を制御します。

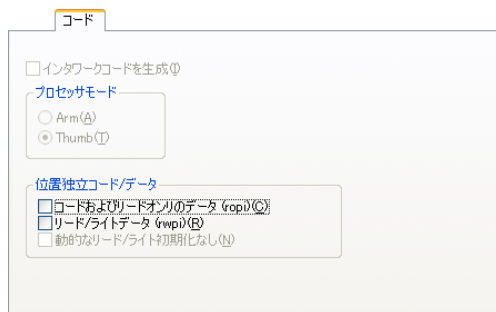


図 86: コードオプション

これらのコンパイラオプションについての詳しいリファレンス情報は、『ARM 用 IAR C/C++ 開発ガイド』を参照してください。

インターワークコードを生成

コンパイラが ARM と Thumb コードを混在させられるようにします。このオプションはデフォルトで有効になっています。

プロセッサモード

プロジェクトのプロセッサモードを選択します。

Arm	完全な 32 ビット命令セットを使用するコードを生成します。
Thumb	縮小 16 ビット命令セットを使用するコードを生成します。Thumb コードではメモリの使用量を最小限に抑え、8/16 ビットパス環境でのパフォーマンスを向上させます。

Position-independence

コンパイラで position-independent コードとデータをどう扱うかを決定します。

コードおよびリードオンリーのデータ (ropi)	アドレスコードおよびリードオンリーのデータへの PC 関連の参照を使用するコードを生成。
リード/ライトデータ (rwpi)	静的ベースレジスタからアドレス書き込み可能なデータへのオフセットを使用するコードを生成。
動的なリード/ライト初期化なし	静的 C 変数のランタイムの初期化を無効化。

最適化

[最適化] オプションは、オブジェクトコード生成の最適化の種類とレベルを設定します。

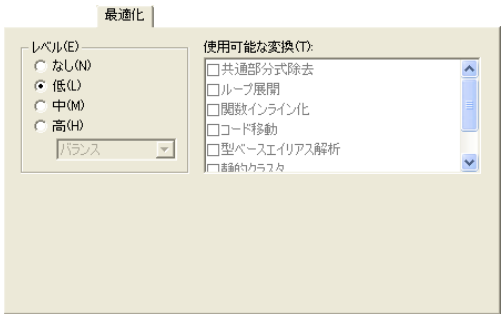


図 87: 最適化オプション

レベル

最適化レベルを選択します。

なし	最適化なし。最も充実したデバッグサポートを提供します。
低	最も低い最適化レベルです。
中	中くらいの最適化レベルです。
高 (バランス)	最も高い最適化レベルで、速度とサイズのバランスをとります。

高 (速度) 最も高い最適化レベルで、速度を重視します。

高 (サイズ) 最も高い最適化レベルで、サイズを重視します。

デフォルトでは、デバッグプロジェクトのサイズの最適化は、完全にデバッグが可能なレベルに設定されます。一方リリースプロジェクトでは、速度を損なうことなく小さいコードを生成する、バランスの取れた最適化レベルに設定されます。

各最適化レベルで実行される最適化リストについては、『*ARM 用 IAR C/C++ 開発ガイド*』を参照してください。

使用可能な変換

異なる最適化レベルでどの変換が使用可能かを選択します。変換が使用可能な場合、チェックボックスを選択すれば有効 / 無効にできます。以下から選択します。

- 共通部分式除去
- ループ展開
- 関数のインライン化
- コード移動
- 型ベースエイリアス解析
- 静的変数クラスタ
- 命令スケジューリング

デバッグプロジェクトでは、デフォルトで変換が無効になっています。リリースプロジェクトでは、デフォルトで変換が有効になっています。

個別に無効にできる変換の説明は、『*ARM 用 IAR C/C++ 開発ガイド*』を参照してください。

出力

【出力】 オプションは、生成されるコンパイラ出力を決定します。

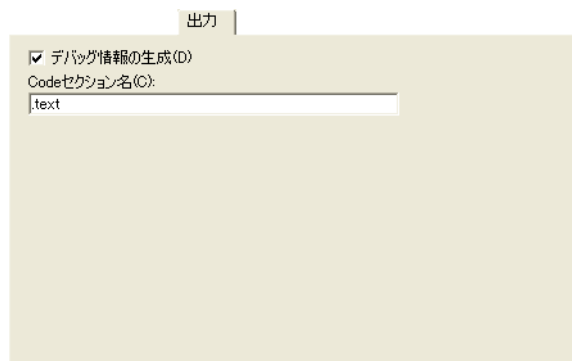


図 88: コンパイラの出力オプション

デバッグ情報の生成

C-SPY® や他のシンボリックデバッガで必要なオブジェクトモジュールに追加情報を含めるようにコンパイラを設定します。

【デバッグ情報の生成】は、デフォルトでは選択されています。コンパイラでデバッグ情報を生成しないように設定する場合は、このオプションの選択を解除します。

注：デバッグ情報を含めると、オブジェクトファイルのサイズが増加します。

コードセクション名

コンパイラは、IAR ILINK リンカが参照する指定セクションに関数を配置します。【コードセクション名】を使用してデフォルト名とは異なる名前を指定し、アプリケーションソースコードの任意の部分を、デフォルトでない別のセクションに配置します。異なるアドレス範囲のコードの配置を管理し、@ 表記または #pragma location ディレクティブでは不十分な場合に、このオプションが有益です。

注：デフォルトで使用するセクション以外の定義済みセクションに関数を明示的に配置する場合は注意してください。状況によっては有益なオプションですが、配置を間違えると、コンパイル時やリンク時のエラーメッセージからアプリケーションの誤動作までを発生することがあります。状況を慎重に考慮し、宣言および関数や変数の使用に関する要件に、厳密に従ってください。

セクション名の変更時には、対応するリンカ設定ファイルも変更する必要があります。ことに、注意してください。

セグメントの詳細およびコードの配置を制御するための各種方法の詳細については、『*ARM 用 IAR C/C++ 開発ガイド*』を参照してください。

リスト

【リスト】 オプションによって、コンパイラでリストファイルを生成し、その内容を決定します。

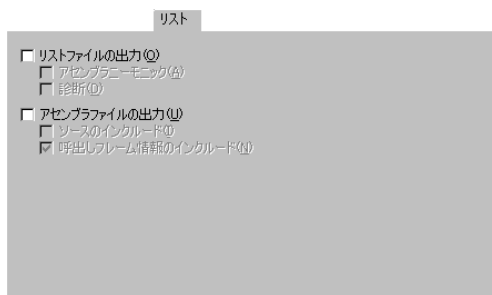


図 89: コンパイラのリストファイルオプション

デフォルトでは、コンパイラはリストファイルを生成しません。リストファイルかアセンブラファイルを生成する場合は、以下のオプションを選択します。リストファイルは `List` ディレクトリに保存され、ファイル名はソースファイル名とファイル名の拡張子 `lst` から構成されます。

このリストファイルをデフォルトのリストファイル用ディレクトリ以外のディレクトリに保存する場合、【一般オプション】カテゴリの【出力ディレクトリ】オプションを使用します。詳細については、161 ページの *出力* を参照してください。

【ワークスペース】ウィンドウの【出力】フォルダから出力ファイルを直接開くことができます。

リストファイルの出力

コンパイラにリストファイルを生成するよう指示します。出力ファイルは、【出力】フォルダ（【ワークスペース】ウィンドウ）から直接開くことができます。デフォルトでは、コンパイラはリストファイルを生成しません。リストファイルの内容を、以下から選択します。

アセンブラニーモニック	アセンブラニーモニックをリストファイルに含めます。
診断	診断情報をリストファイルに含めます。

アセンブラファイルの出力

コンパイラにアセンブラリストファイルを生成するよう指示します。リストファイルの内容を、以下から選択します。

ソースのインクルード	ソースコードをアセンブラファイルに含めます。
呼出しフレーム情報のインクルード	コンパイラが生成したランタイムモデル属性情報、呼出しフレーム情報、フレームサイズ情報を含めます。

プリプロセッサ

[プリプロセッサ] オプションを使用して、コンパイラで使用するシンボルおよびインクルードパスを定義できます。

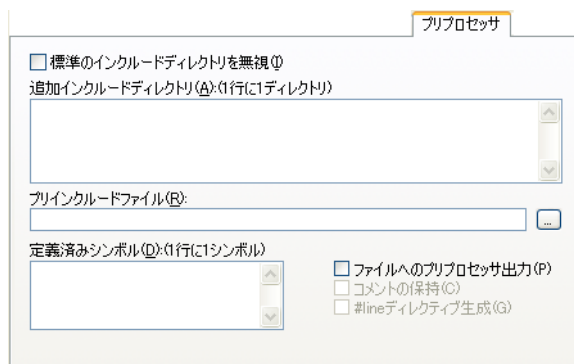


図 90: コンパイラのプリプロセッサオプション

標準のインクルードディレクトリを無視

標準のインクルードディレクトリを無視します。これは、プロジェクトがビルドされる際には使用されません。

追加インクルードディレクトリ

`#include` ファイルパスのリストのフルパスを指定します。製品で必要なパスは、ランタイムライブラリの選択に基づいて自動的に指定されます。

注： 指定するすべてのディレクトリが、標準のインクルードディレクトリより先に検索されます。

プロジェクトの移植性を向上するには、アクティブな製品のサブディレクトリに引数変数 `$TOOLKIT_DIR$`、現在のプロジェクトのディレクトリに `$PROJ_DIR$` を使用します。引数変数の概要については、*124 ページの 引数変数を参照してください。*

プリインクルードファイル

コンパイラがソースファイルを読み込む前に、指定したインクルードファイルを含めるようにコンパイラに指示します。これは、アプリケーション全体のソースコードで変更を行う場合（新しいシンボルを定義する場合など）に便利です。

シンボル定義

シンボルを定義します。通常はソースファイルでこれを定義しなければなりません。定義するシンボルを 1 行ごとに 1 つ入力して、その値を指定します。

次に例を示します。

```
TESTVER=1
```

等号の前後に空白文字を挿入しないでください。

シンボル `TESTVER` が定義されているかどうかに応じて、アプリケーションのテストバージョンと製品バージョンのいずれかを生成するように、ソースコードを記述するとします。この場合、以下のようなセクションを記述します。

```
#ifdef TESTVER
... ; テストバージョンのみの追加コード行
#endif
```

このとき、シンボル `TESTVER` は、デバッグターゲットでは定義し、リリースターゲットでは定義しません。

[シンボル定義] オプションは、ソースファイルの最初に `#define` 文を記述した場合と同様に機能します。

ファイルへのプリプロセッサ出力

プリプロセッサ出力を、1st ディレクトリにあるファイル名拡張子 `i` を持つファイルに生成します。デフォルトでは、コンパイラはプリプロセッサ出力を生成しません。以下から選択します。

コメントの保持

コメントを保持します。

#line ディレクティブ生成

#line ディレクティブを生成します。

診断

[診断] オプションは、診断メッセージの分類 / 表示方法を設定します。デフォルト以外の分類を指定する場合に使用します。

注：診断メッセージでは致命的なエラーの表示を無効にすることはできず、致命的なエラーの分類を変更することはできません。

診断

☐ リマークを有効化 (N)

☐ 診断を無効化 (S)

リマークとして処理 (R):

ワーニングとして処理 (W):

エラーとして処理 (E):

☐ すべてのワーニングをエラーとして処理 (T)

図 91: コンパイラの診断オプション

リマークを有効化

コンパイラにリマークを生成するよう指示します。デフォルトでは、リマークは出力されません。

最も軽度の診断メッセージを、リマークと呼びます。リマークは、ソースコード中で、生成したコードで異常な動作の原因となる可能性がある部分を示します。

診断を無効化

指定するタグについて、診断メッセージの出力を無効にします。

たとえば、Pe117 および Pe177 のワーニングを無効にするには、次のように入力します。

```
Pe117, Pe177
```

リマークとして処理

診断メッセージをリマークとして分類します。リマークは、最も軽度の診断メッセージです。リマークは、ソースコード中で、生成したコードで異常な動作の原因となる可能性がある部分を示します。

たとえば、Pe117 のワーニングをリマークとして分類するには、次のように入力します。

```
Pe117
```

ワーニングとして処理

診断メッセージをワーニングとして分類します。ワーニングは、問題はあるが、コンパイルの途中終了の原因にはならないエラーや脱落を示します。

たとえば、Pe826 のリマークをワーニングとして分類するには、次のように入力します。

```
Pe826
```

エラーとして処理

診断メッセージをエラーとして分類します。エラーは、C/C++ 言語の規則違反のうち、オブジェクトコードが生成されず、終了コードがゼロ以外になるものを示します。

たとえば、Pe117 のワーニングをエラーとして分類するには、次のように入力します。

```
Pe117
```

すべてのワーニングをエラーとして処理

すべてのワーニングをエラーとして分類します。コンパイラがエラーを検出した場合は、オブジェクトコードは生成されません。

MISRA-C

[MISRA-C:1998] と [MISRA-C:2004] オプションは、[一般オプション] カテゴリの対応するオプションをオーバーライドします。

特定のオプションの詳細については、『*IAR Embedded Workbench® MISRA-C:2004 リファレンスガイド*』または『*IAR Embedded Workbench® MISRA-C:1998 リファレンスガイド*』を参照してください。これらは、[ヘルプ] メニューから利用できます。

追加オプション

[追加オプション] ページは、コンパイラへのコマンドラインインタフェースを提供します。

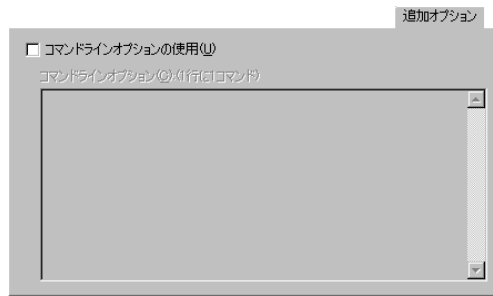


図 92: コンパイラの追加オプション

コマンドラインオプションの使用

コンパイラに引き渡される追加のコマンドライン引数を指定します（GUI でサポートされていません）。

アセンブラオプション

この章では、IAR Embedded Workbench® IDE で使用できるアセンブラオプションについて説明します。

オプションの設定方法の詳細については、61 ページの *オプションの設定* を参照してください。

アセンブラオプションの概要

このセクションでは、[アセンブラ] カテゴリのオプションについて説明します。

IDE のアセンブラオプションを設定するには、以下の手順に従います。

- 1 [プロジェクト] > [オプション] を選択して、[オプション] ダイアログボックスを表示します。
- 2 [カテゴリ] リストで [アセンブラ] を選択します。
- 3 すべての設定をデフォルトの出荷時の設定に戻すには、[出荷時設定] ボタンをクリックします。

言語

[言語] オプションは、アセンブラ言語の特定の動作を制御します。

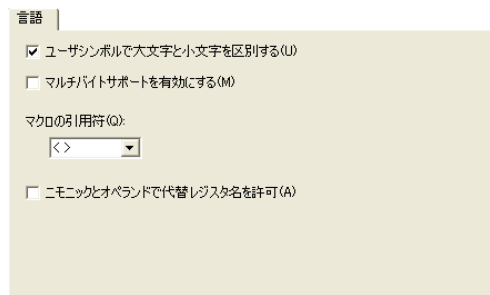


図 93: アセンブラの言語オプション

ユーザシンボルで大文字 / 小文字を区別する

大文字 / 小文字の区別を切り替えます。デフォルトでは大文字 / 小文字の区別が行われます。つまり、`LABEL` と `label` は異なるシンボルを示します。大文字 / 小文字の区別をオフにする場合、`LABEL` と `label` は同じシンボルを指します。

マルチバイトサポートを有効にする

ソースコード内のマルチバイト文字は、ホストコンピュータのデフォルトのマルチバイト文字サポート設定に従ってアセンブラにより解釈されます。デフォルトでは、マルチバイト文字をアセンブラのソースコードで 사용할ことはできません。

マルチバイト文字は、コメント、文字列定数、文字定数で使用できます。これらはそのまま生成コードに移動します。

マクロの引用符

各マクロ引数の左右の引用符に使用する文字を選択します。デフォルトでは、これらの引用符は `<` と `>` です。

マクロ引用符の文字によって、他の表記法に合わせて引用符を変更したり、あるいはマクロ引数に `<` や `>` を含めることができるようにします。



図 94: マクロの引用符の選択

代替ニモニック、オペランド、レジスタ名を許可

既存のアプリケーションから IAR Assembler for ARM へ移行するために、代替レジスタ名、ニモニック、およびオペランドを使用可能にすることができます。この操作には、アセンブラコマンドラインの `-j` オプションを使用します。このオプションを、ARM ADS/RVCT アセンブラ用に書かれたアセンブラソースコードに使用します。詳細については、『*ARM® IAR アセンブラリファレンスガイド*』を参照してください。

出力

[出力] オプションによって、生成されるアセンブラ出力が決まります。

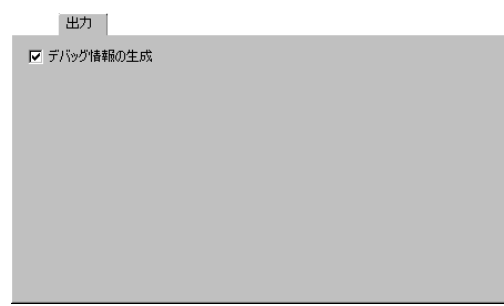


図 95: アセンブラの出力オプション

デバッグ情報の生成

アセンブラでデバッグ情報を生成します。アプリケーションとともにデバッガを使用する場合に、このオプションを使用します。デフォルトでは、このオプションは、デバッグプロジェクトでは選択されていて、リリースプロジェクトでは選択が解除されています。

リスト

[リスト] オプションによって、アセンブラでリストファイルを生成し、その内容を決定します。

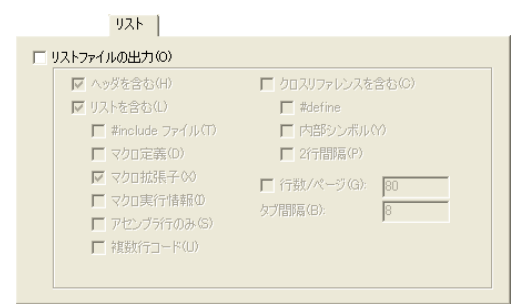


図 96: アセンブラリストファイルオプション

リストファイルの出力

アセンブラでリストファイルを生成して、それをファイル `sourcename.lst` に送信します。デフォルトでは、アセンブラはリストファイルを生成しません。

このリストファイルをデフォルトのリストファイル用ディレクトリ以外のディレクトリに保存する場合、**[一般オプション]** カテゴリの **[出力ディレクトリ]** オプションを使用します。詳細については、161 ページの *出力* を参照してください。**[ワークスペース]** ウィンドウの **[出力]** フォルダから出力ファイルを直接開くことができます。

ヘッダを含む

ヘッダを含めます。アセンブラリストファイルのヘッダには、製品バージョン、アセンブリの日付と時刻の情報、および使用されたアセンブラオプションと同等のコマンドラインを含みます。

リストを含む

リストファイルにインクルードする情報のタイプを選択します。

#include されたテキスト	リストファイルに #include ファイルを含みます。
マクロ定義	マクロ定義をリストファイルに含みます。
マクロ拡張子	マクロ拡張をリストファイルから除外します。
マクロ実行情報	マクロ実行情報をすべてのマクロ呼出しに印刷します。
アセンブラ行のみ	リストファイルから偽の条件付きアセンブラセクションの行を除外します。
複数行コード	必要に応じて、ディレクティブで生成したコードを複数行にリストにします。

クロスリファレンスを含む

リストファイルの末尾にクロスリファレンステーブルをインクルードします。

#define	プリプロセッサ #defines をインクルードします。
内部シンボル	ユーザ定義およびアセンブラ内部のすべてのシンボルをインクルードします。
2 行間隔	2 行間隔を許可します。

行数 / ページ

ページあたりの行数を 10 から 150 までで指定します。デフォルトのページあたりの行数は、アセンブラのリストファイルの場合 80 です。

タブ間隔

タブストップあたりの文字位置数を、2 ～ 9 の範囲で変更します。デフォルトでは、アセンブラでタブストップが 8 文字ごとに設定されます。

プリプロセッサ

[プリプロセッサ] オプションを使用して、アセンブラで使用するシンボルおよびインクルードパスを定義できます。

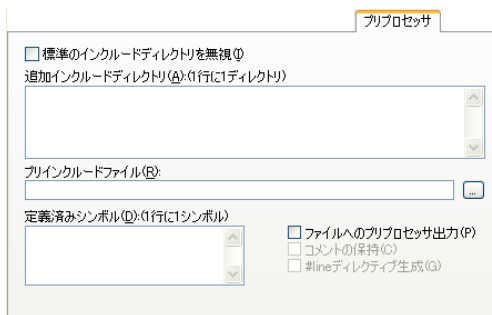


図 97: アセンブラのプリプロセッサオプション

標準のインクルードディレクトリを無視

標準のインクルードファイルを無視します。これは、プロジェクトがビルドされる際には使用されません。

追加インクルードディレクトリ

#include ファイルパスのリストのフルパスを指定します。製品で必要なパスは、自動的に指定されます。

プロジェクトの移植性を向上するには、アクティブな製品のサブディレクトリに引数変数 \$TOOLKIT_DIR\$、現在のプロジェクトのディレクトリに \$PROJ_DIR\$ を使用します。引数変数の概要については、124 ページの *引数変数* を参照してください。

#include ディレクティブについては、『ARM® IAR アセンブラリファレンスガイド』を参照してください。

注: デフォルトでは、アセンブラは `#include` ファイル (`IASMARM_INC` の環境変数で指定したパスにあります) も検索します。ただし、IDE で環境変数を使用することはお勧めしません。

シンボル定義

通常はソースファイルで定義しなければならないシンボルを定義します。定義するシンボルを行ごとに 1 つ、値も含めて指定します。

たとえば、シンボル `TESTVER` が定義されているかどうかに応じてアプリケーションのテストバージョンと製品バージョンのいずれかを生成するように、ソースコードを記述するとします。この場合、以下のようなセクションを記述します。

```
#ifdef TESTVER
... ; テストバージョンのみの追加コード行
#endif
```

このとき、シンボル `TESTVER` は、デバッグターゲットでは定義し、リリースターゲットでは定義しません。

また、頻繁に変更する必要がある変数 `FRAMERATE` をソースで使用するとします。この場合、ソースではこの変数を定義せず、このオプションを使用してプロジェクトでの値を `FRAMERATE=3` のように指定することができます。

診断

【診断】 オプションでは、個々のワーニングやワーニングの範囲を制御します。

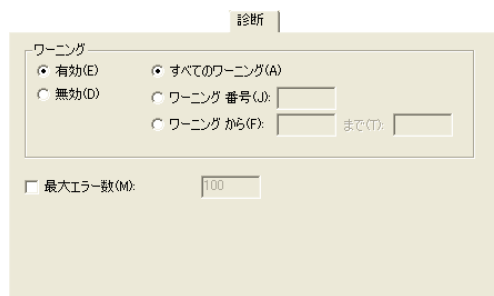


図 98: アセンブラ診断オプション

ワーニング

アセンブラのワーニングを制御します。プログラミングエラーに起因するなどの理由で正当なソースコードの要素を検出した場合、アセンブラはワーニングメッセージを表示します。デフォルトでは、すべてのワーニング

が有効になっています。ワーニングの生成を制御するには、以下のいずれかを選択します。

有効	ワーニングを有効にします。
無効	ワーニングを無効にします。
すべてのワーニング	すべての警告を有効 / 無効にします。
特定ワーニング	指定するワーニングを有効 / 無効にします。
ワーニング範囲指定	指定した範囲のすべてのワーニングを有効 / 無効にします。

アセンブラワーニングの詳細については、『ARM® IAR アセンブラリファレンスガイド』を参照してください。

最大エラー数

エラーの最大数を指定します。つまり、たとえば 1 つのアセンブリでより多くのエラーを確認するために、報告されるエラーの数を増やしたり減らすことができます。デフォルトでは、アセンブラで報告されるエラーの最大数は 100 です。

追加オプション

[追加オプション] ページは、アセンブラへのコマンドラインインタフェースを提供します。

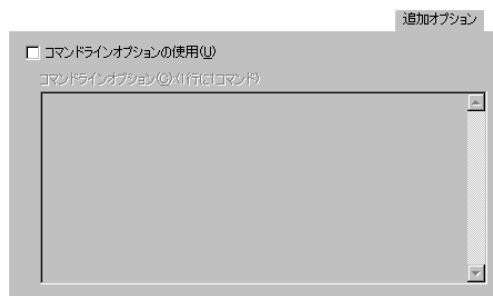


図 99: アセンブラ用の [追加オプション]

コマンドラインオプションの使用

アセンブラに引き渡される追加のコマンドライン引数を指定します（GUI でサポートされていません）。

出力コンバータオプション

この章では、ELF フォーマットからの出力ファイルを変換する際に IAR Embedded Workbench® IDE で使用可能なオプションについて説明します。

オプションの設定方法の詳細については、61 ページの *オプションの設定* を参照してください。

出力コンバータオプションの説明

このセクションでは、[出力コンバータ] カテゴリのオプションについて説明します。

IDE のコンバータのオプションを設定するには、以下の手順に従います。

- 1 [プロジェクト] > [オプション] を選択して、[オプション] ダイアログボックスを開きます。
- 2 [カテゴリ] リストで [出力コンバータ] を選択します。

出力

[出力] オプションによって、プログラム可能な出力フォーマットの詳細を決定します。

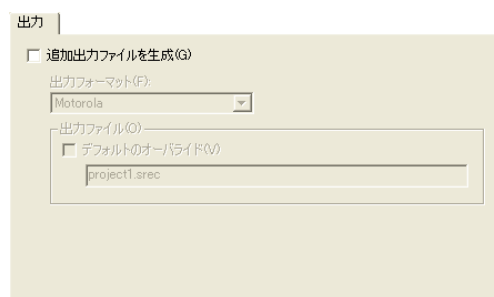


図 100: 出力コンバータオプション

追加出力ファイルの生成

ILINK リンカでは、出力として ELF を生成します（オプションとしてデバッグ情報用の DWARF を含む）。[追加出力の生成]を使用すると、コンバータ ielftool で ELF 出力を Motorola や Intel-extended など指定した形式に変換できます。コンバータの詳細については、『*ARM 用 IAR C/C++ 開発ガイド*』を参照してください。

出力フォーマット

ielftool からの出力形式を選択します。以下から選択します。Motorola、Intel-extended、バイナリ、simple-code。コンバータの詳細については、『*ARM 用 IAR C/C++ 開発ガイド*』を参照してください。

出力ファイルのオーバーライド

iarelf により変換された出力ファイルの名前を指定します。デフォルトでは、リンカはファイル名の拡張子を持つプロジェクト名を使用します。ファイル名拡張子は、選択した出力フォーマットによって異なります。たとえば、srec や hex です。デフォルト名をオーバーライドするには、[デフォルト]を選択して、代替のファイル名またはファイル名拡張子を指定します。

カスタムビルドオプション

この章では、IAR Embedded Workbench® IDE のカスタムビルドオプションについて説明します。

オプションの設定方法の詳細については、61 ページの *オプションの設定* を参照してください。

カスタムビルドオプションの説明

このセクションでは、[カスタムビルド] カテゴリのオプションについて説明します。IDE のカスタムビルドオプションを設定するには、以下の手順に従います。

- 1 [プロジェクト] > [オプション] を選択して、[オプション] ダイアログボックスを表示します。
- 2 [カテゴリ] リストで [カスタムビルド] を選択します。

カスタムツール設定

[カスタムツール設定] オプションは、ツールチェーンに追加するツールの呼出しを制御します。

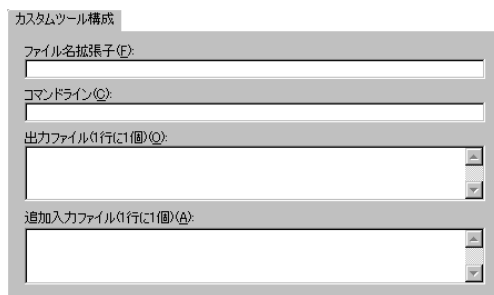


図 101: カスタムツール構成オプション

例については、66 ページの *ツールチェーンの拡張* を参照してください。

ファイル名拡張子

カスタムツールで処理するファイルタイプのファイル名拡張子を指定します。複数のファイル名拡張子を入力できます。区切り文字には、コンマ、セミicolon、空白文字を使用します。次に例を示します。

```
.htm; .html
```

コマンドライン

外部ツールを実行するためのコマンドラインを指定します。

出力ファイル

外部ツールからの出力ファイルの名前を指定します。

追加入力ファイル

ビルド処理中に外部ツールが使用する追加ファイルがあれば指定します。これらの追加入力ファイル（依存ファイル）を修正した場合は、リビルドの必要性が検出されます。

ビルドアクションオプション

この章では、IAR Embedded Workbench® IDE のビルド前とビルド後のアクション用オプションについて説明します。

オプションの設定方法の詳細については、61 ページの *オプションの設定* を参照してください。

ビルドアクションのオプションの説明

このセクションでは、[ビルドアクション] カテゴリのオプションについて説明します。

IDE のビルドアクションのオプションを設定するには、以下の手順に従います。

- 1 [プロジェクト] > [オプション] を選択して、[オプション] ダイアログボックスを表示します。
- 2 [カテゴリ] リストで [ビルドアクション] を選択します。

[ビルドアクション]

[ビルドアクション] オプションでは、IDE でのビルド前およびビルド後のアクションを指定します。これらのオプションは、ビルド構成全体に適用されます。グループやファイル単位で設定することはできません。

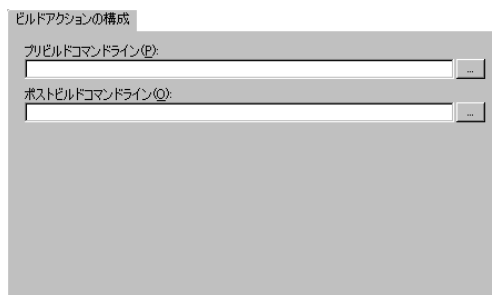


図 102: ビルドアクションオプション

プリビルドコマンドライン

ビルドの前に直接実行されるコマンドラインを指定します。参照ボタンを使用して、実行するツールを検索します。構成が更新済みの場合は、コマンドは実行されません。

ポストビルドコマンドライン

ビルドが成功した後に直接実行されるコマンドラインを指定します。参照ボタンを使用して、実行するツールを検索します。構成がすでに更新されていた場合は、コマンドは実行されません。このオプションは、出力ファイルのコピーや後処理に便利です。

リンカのオプション

この章では、IAR Embedded Workbench® IDE で使用できるリンカのオプションについて説明します。

オプションの設定方法の詳細については、61 ページの *オプションの設定* を参照してください。

リンカオプションの説明

このセクションでは、[リンカ] カテゴリのオプションについて詳しく説明します。

IDE のリンカオプションを設定するには、以下の手順に従います。

- 1 [プロジェクト] > [オプション] を選択して、[オプション] ダイアログボックスを表示します。
- 2 [カテゴリ] リストで [リンカ] を選択します。
- 3 すべての設定をデフォルトの出荷時の設定に戻すには、[出荷時設定] ボタンをクリックします。

設定

[設定] オプションを使用すると、リンカ設定ファイルのパスと名前を指定して、設定ファイルにシンボルを定義できます。

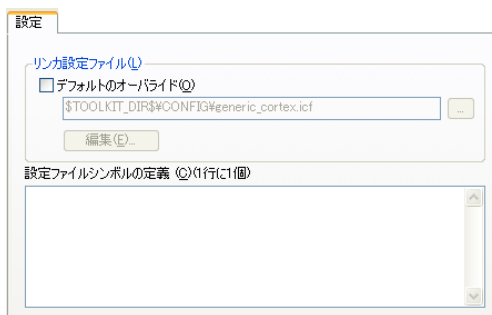


図 103: リンカ設定オプション

リンカ設定ファイル

デフォルトのリンカ設定ファイルは、使用するプロジェクト設定に応じて自動的に選択されます。デフォルトのファイルをオーバーライド・するには、**[デフォルトのオーバーライド]** を選択し、他のファイルを指定します。

引数変数 \$TOOLKIT_DIR\$ か \$PROJ_DIR\$ を使用して、プロジェクト固有の設定ファイルか定義済みの設定ファイルを指定することもできます。

設定ファイルシンボルの定義

設定ファイルで使用する常時設定シンボルを定義します。このようなシンボルは、リンカ設定ファイルの `define symbol` ディレクティブを使用して定義したシンボルと同じ効果があります。

ライブラリ

[ライブラリ] オプションでは、使用済みライブラリのセットを選択します。

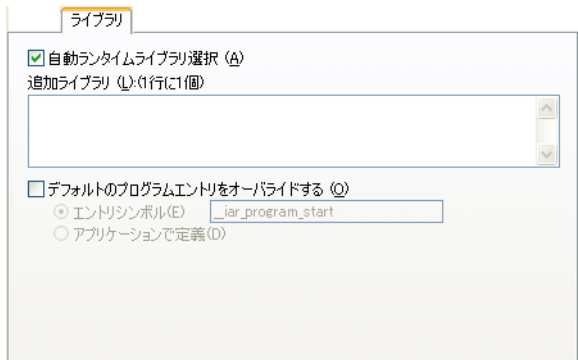


図104: リンカライブラリオプション

使用可能なライブラリの詳細については、『*ARM 用 IAR C/C++ 開発ガイド*』を参照してください。

自動ランタイムライブラリ選択

プロジェクト設定に基づいて、リンカで適切なライブラリを自動的に選択します。

追加ライブラリ

リンク処理中にリンカが含める追加ライブラリを指定します。ライブラリは1行に1つしか指定できず、ライブラリへのフルパスを指定する必要があります。引数変数 \$PROJ_DIR\$ と \$TOOLKIT_DIR\$ が使用できます。

または、[ワークスペース] ウィンドウで補足のライブラリをプロジェクトに直接追加することができます。この例は、ライブラリの作成および使用のチュートリアルにあります。

デフォルトのプログラムエントリをオーバーライドする

デフォルトでは、プログラムエントリには `__iar_program_start` というラベルが設定されています。リンカは、プログラムエントリラベルを含むモジュールが含まれていて、そのラベルを含むセクションが破棄されていないことを確認します。

[デフォルトプログラムエントリのオーバーライド] は、デフォルトのエントリラベルをオーバーライドします。以下から選択してください。

エントリシンボル デフォルト以外のエントリシンボルを指定します。

アプリケーションで定義 リンクされたオブジェクトコードに定義されたエントリシンボルを使用します。リンカは、通常の場合と同様に、すべてのプログラムモジュールと、すべてのシンボル参照に必要なライブラリモジュールを含め、`root` 属性が設定されたすべてのセクション、またはそのようなセクションから直接的 / 間接的に参照されるすべてのセクションを保持します。

入力

[入力] オプションは、リンカへの入力を処理する方法を指定します。

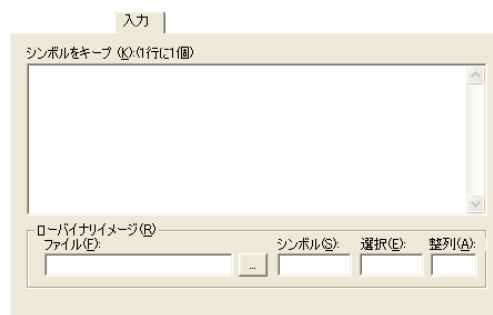


図 105: リンカ入力オプション

シンボルをキープ

最終のアプリケーションに常に含まれるべきシンボルを行ごとに定義します。

デフォルトでは、リンカはアプリケーションに必要なシンボルのみを保存します。

ロウバイナリイメージ

通常の入力ファイルに加えて、ピュアバイナリファイルをリンクします。以下のパラメータを指定できます。

ファイル	リンクするピュアバイナリファイルを入力します。
シンボル	バイナリデータが配置されるセクションにより定義されるシンボルを入力します。
セクション	バイナリデータを配置するセクションを入力します。
アライン	バイナリデータが配置されるセクションのアラインメントを入力します。

ファイルの内容全体が、指定したセクションに配置されます。つまり、このセクションはロウバイナリ出力フォーマットなどのピュアバイナリデータだけを含むことができます。指定したファイルの内容が配置されるセクションは、指定したシンボルがアプリケーションで要求される場合にだけ含まれます。シンボルを強制的に参照するには、`--keep` リンカオプションを使用します。単一出力ファイルおよび `--keep` オプションについては、『*ARM 用 IAR C/C++ 開発ガイド*』を参照してください。

最適化

【最適化】 オプションは、リンカの最適化を制御します。

最適化

- ☐ 小さいルーチンのインライン化
- ☐ 重複セクションのマージ(M)
- ☒ C++仮想関数除去を実行(P)
 - ☐ VFE情報を持たないモジュールがある場合(V)
- ☒ C++例外を許可(A)
 - ☐ 常に含める

図 106: リンカの最適化オプション

これらのオプションについては、『ARM 用 IAR C/C++ 開発ガイド』を参照してください。

小さいルーチンのインライン化

可能な場合にはリンカがルーチンの呼出しをルーチン本体と置き換えるようにします。

重複セクションのマージ

リンカで、リードオンリーのセクションのコピーを 1 つだけ保持します。これによって異なる関数や定数が同じアドレスを持つことがあるため、このオプションを選択すると、異なるアドレスに依存するアプリケーションが正しく機能しなくなるため注意してください。

C++ 仮想関数除去を実行

仮想関数除去の最適化を有効にします。

仮想関数の除去を強制的に使用するには、**「VFE 情報を持たないモジュールがある場合」** オプションを有効にします。これは、必要な情報を持たない一部のモジュールが仮想関数の呼出しを実行したり、動的ランタイム型情報を使用すると安全でなくなる可能性があります。

C++ 例外を許可

このオプションを使用しない場合、インクルードされたコードに `throw` がある場合にリンカがエラーを生成します。

アプリケーションで例外が間違って使用されないようにリンカでチェックする場合は、このオプションを使用しないでください。

C++ 例外を常に含める

不要と思われる場合でも、リンカは例外処理コードとテーブルをインクルードします。

インクルードされるコードに `rethrow` ではない `throw` 式がある場合、リンカは使用する例外を考慮します。コードのその他の部分にそうした `throw` 式がなければ、リンカは `operator new`、`dynamic_cast`、`typeid` を用意して、失敗したときに例外をスローするのではなく `abort` を呼び出します。コードに他のスローが含まれておらず、これらのコンストラクトからの例外を検出しなければならない場合、このオプションを使用しなければならないことがあります。

出力

[出力] オプションによって、生成されるリンカ出力が決まります。

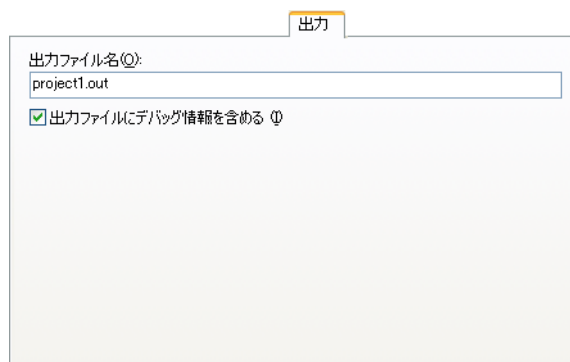


図 107: リンカ出力オプション

出力ファイル名

ILINK 出力ファイル名を設定します。デフォルトでは、リンカはファイル名の拡張子を持つプロジェクト名を使用します。out デフォルト名をオーバーライドするには、出力ファイルの別名を指定します。

出力ファイルにデバッグ情報を含める

リンカでデバッグ情報の DWARF も含めた ELF 出力ファイルを生成します。

リスト

[リスト] オプションは、リンカリストの生成を制御します。

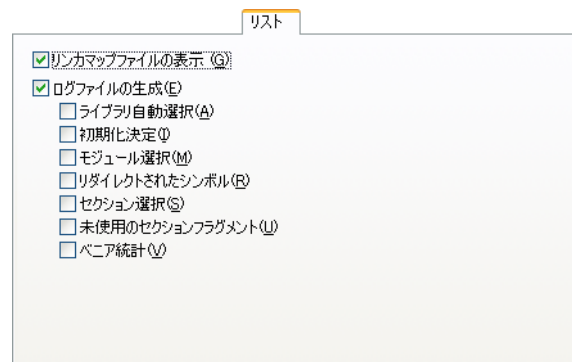


図 108: リンカリストオプション

リンカマップファイルの表示

リンカでリンカメモリマップファイルを生成して、`projectname.map` ファイルに送ります。マップファイルとその内容について詳しくは、『*ARM 用 IAR C/C++ 開発ガイド*』を参照してください。

ログファイルの生成

リンカで、ログ情報を `projectname.log` ファイルに保存します。ログ情報は、実行可能なイメージが現在の状態になった原因を把握するために利用できる場合があります。以下を保存できます。

- ライブラリ自動選択
- 初期化決定
- モジュール選択
- リダイレクトされたシンボル
- セクション選択
- 未使用のセクションフラグメント
- ペニア統計

#define

[#define] は、リンク時に絶対シンボルを定義する場合に使用します。

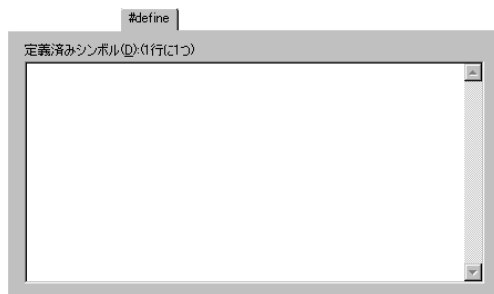


図 109: リンカ #define オプション

シンボル定義

リンク時に使用する絶対シンボルを定義します。これは、設定目的の場合に特に便利です。プロジェクトに定義するシンボルを 1 行ごとに 1 つ入力して、その値を指定します。次に例を示します。

```
TESTVER=1
```

等号の前後に空白文字を挿入しないでください。

リンカ設定ファイルでは、任意の個数のシンボルを定義できます。この方法で定義したシンボルは、リンカが生成する ?ABS_ENTRY_MOD という特別なモジュールに含まれます。

既存のシンボルを再定義しようとする、エラーメッセージが表示されます。

診断

[診断] オプションは、診断の分類 / 表示方法を設定します。デフォルト以外の分類を指定する場合に使用します。

注：致命的なエラーの表示を無効にする、あるいは致命的なエラーの分類を変更することはできません。

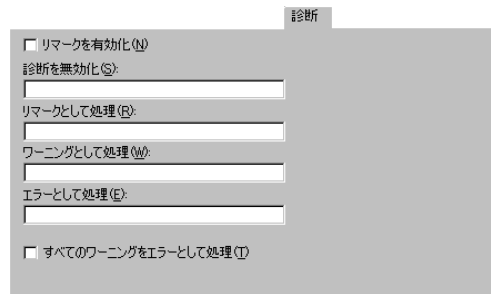


図 110: リンカ診断オプション

リマークを有効化

リンカでリマークを生成します。デフォルトでは、リマークは出力されません。

最も軽度の診断メッセージを、*リマーク*と呼びます。リマークは、ソースコード中で、生成したコードで異常な動作の原因となる可能性がある部分を示します。

指定した診断を無効化

指定したタグの診断メッセージの出力を無効にします。

たとえば、Pe117 および Pe177 のワーニングを無効にするには、次のように入力します。

Pe117,Pe177

リマークとして処理

診断メッセージをリマークとして分類します。

たとえば、Pe117 のワーニングをリマークとして分類するには、次のように入力します。

Pe177

ワーニングとして処理

診断メッセージをワーニングとして分類します。ワーニングは、問題はあるが、リンク処理の終了前にリンカが終了する原因にはならないエラーや脱落を示します。

たとえば、Pe826 のリマークをワーニングとして分類するには、次のように入力します。

```
Pe826
```

エラーとして処理

診断メッセージをエラーとして分類します。エラーは、リンクの規則違反のうち、実行可能なイメージが生成されず、終了コードがゼロ以外になるものを示します。

たとえば、Pe117 のワーニングをエラーとして分類するには、次のように入力します。

```
Pe117
```

すべてのワーニングをエラーとして処理

すべてのワーニングをエラーとして分類します。リンカがエラーを検出した場合は、実行可能イメージは生成されません。

チェックサム

[チェックサム] オプションは、フィルとチェックサムを制御します。

チェックサム

☒ 未使用コードメモリをフィルする(F)
フィルパターン: 0xFF
開始アドレス(T): 0x0 終了アドレス(E): 0x0

☒ チェックサム生成(G)
サイズ(Z): 2/バイト 整列(A): 1

☐ 算術合計(S) ☐ フルサイズでの結果(U)
☒ CRC16 (0x11021)(1)
☐ CRC32 (0x4C11DB7)(3)
☐ CRC多項式(R): 0x11021

初期値(D): 0x0

補数(C): そのまま使用 ☒ 入力として使用(N)
ビット順(B): MSBが先頭

☐ 語句内でバイトオーダーを逆順にする

図 111: リンカのチェックサムおよびフィルオプション

フィリングおよびチェックサムに関する詳細については、『ARM 用 IAR C/C++ 開発ガイド』を参照してください。

未使用コードメモリをフィルする

指定範囲の未使用メモリのフィル:

フィルパターン	セグメントパート間のギャップに設定するフィルのサイズを、16 進数表記で指定します。
開始アドレス	フィルする範囲の開始アドレスを指定します。
終了アドレス	フィルする範囲の終了アドレスを指定します。

チェックサム生成

指定範囲にチェックサムを生成します。

以下から選択します。

サイズ	チェックサムのサイズ (1、2、4 バイト) を選択します。
アラインメント	チェックサムのオプションのアラインメントを指定します。アラインメントを明示的に指定しない場合は、2 のアラインメントが使用されます。
算術合計	単純な算術合計のアルゴリズムを選択します。結果は 8 ビットに切り詰められます。
フルサイズでの結果	算術合計アルゴリズムの結果を、1 バイトに切り詰めるのではなく、指定したサイズで生成します。
CRC16 (デフォルト)	CRC16 アルゴリズムを選択します (生成多項式 0x11021)。
CRC32	CRC32 アルゴリズムを選択します (生成多項式 0x104C11DB7)。
CRC 多項式	CRC 多項式アルゴリズムの選択し、指定した値の多項式を生成します。
補数	派生した補数 (1 の補数または 2 の補数) を選択します。
ビット順	出力する結果のビット順を選択します。以下から選択します。 MSB 優先: 各バイトで最重要のビットを最初に出力します。 LSB 優先: 各バイトのビット順を逆にして、最も重要でないビットを先に出力します。

語句内でバイトオーダーを逆順にする	[サイズ] で指定したサイズの各語句内で、入力データのビット順を逆にします。
初期値	チェックサムの初期値を指定します。これは、使用するコアに専用のチェックサム計算があり、その計算をリンカが実行する計算に一致させる場合に使用します。
入力として使用	入力データの先頭に、[初期値] で指定した値を含む [サイズ] の語句を 1 字付けます。

追加オプション

[追加オプション] ページは、リンカへのコマンドラインインタフェースを提供します。

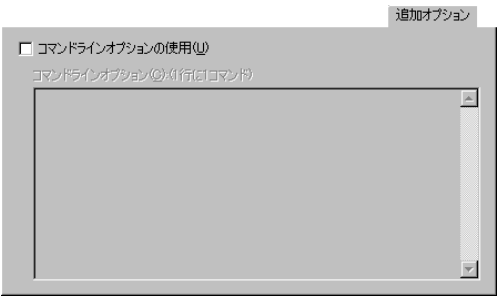


図 112: リンカ追加オプション

コマンドラインオプションの使用

リンカに引き渡される追加のコマンドライン引数を指定します（GUI でサポートされていません）。

ライブラリビルダオプション

この章では、IAR Embedded Workbench® IDE のライブラリビルダオプションについて説明します。

オプションの設定方法の詳細については、61 ページの *オプションの設定* を参照してください。

ライブラリビルダオプションの説明

このセクションでは、[ライブラリビルダ] カテゴリのオプションについて説明します。

ライブラリビルダのオプションはデフォルトでは使用できません。これらのオプションを IDE で設定するには、先にライブラリビルダツールをカテゴリリストに追加する必要があります。

- 1 [プロジェクト] > [オプション] > [一般オプション] > [出力] を選択します。
- 2 [ライブラリ] オプションを選択します。[ライブラリビルダ] が [オプション] ダイアログボックスに表示されます。
- 3 [カテゴリ] リストで [ライブラリビルダ] を選択します。

出力

[出力] オプションはライブラリビルダを制御し、ビルド処理の結果として、ライブラリビルダはライブラリ出力ファイルを作成します。

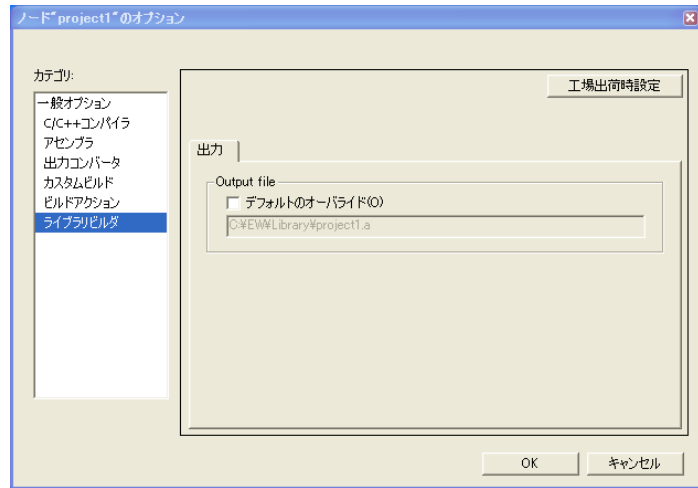


図 113: ライブラリビルダの出力オプション

工場出荷時設定

すべての設定をデフォルトの工場出荷時設定に戻します。

出力ファイル

ライブラリビルダからの出力ファイルの名前を指定します。デフォルトでは、リンクはファイル名の拡張子を持つプロジェクト名を使用します。デフォルト名をオーバーライドするには、[デフォルトのオーバーライド] を選択して出力ファイルの別名を指定します。

用語集

この用語集は、組込みシステムのプログラミングに関連した一般的な用語を対象しています。用語によっては、ご使用の IAR Embedded Workbench® のバージョンに該当しないこともあります。

A

絶対アドレス

リンクが割り当てるアドレスではなく、ソースコードで指定したオブジェクトの特定メモリアドレス。

アドレス式

値がアドレスになっている式。

AEABI

ARM Limited が提唱する ARM Embedded Application Binary Interface。

アプリケーション

IAR システムズのツールキットのユーザが開発し、ターゲットプロセッサで組込みアプリケーションとして実行されるプログラム。

Ar

アーカイブ、つまりライブラリから作成、修正、抽出するための GNU バイナリユーティリティ。関連項目 [larchive](#)。

アーキテクチャ

コンピュータ設計者の間で使用される、複雑な情報処理システムの構造を示す用語。使用される命令やデータの種類、メモリ構成やアドレッシング方法、システムの実装方法などを示します。プロセッサ設計で使用される主流アーキテクチャとして、**ハーバードアーキテクチャ**と**ノイマンアーキテクチャ**の2つがあります。

アーカイブ

ライブラリを参照。

アセンブラディレクティブ

アセンブラの処理を制御するコマンドセット。

アセンブラ言語

ターゲットプロセッサと、入出力レジスタやデータエリアに対する処理を指定するために使用する、個々のマシン固有のニーモニックセット。メモリ使用量の節約や、アプリケーションの実行速度の向上には、C/C++ よりもアセンブラ言語の方が適している場合があります。

アセンブラオプション

アセンブラのデフォルトの動作を変更するためのパラメータ。

属性

[セクション属性](#)を参照。

自動変数

変数が宣言されている関数が呼び出されるごとに、変数の新しいインスタンスが自動的に作成されることを指します。静的オーバーレイを使用するシステム（関数が再帰的に呼び出された場合でも、ローカル変数が1つのインスタンスにだけ存在）でのローカル変数の処理と比較できます。ローカル変数と呼ばれることもあります。[レジスタ変数](#)と比較してください。

B

バックトレース

IAR C-SPY® デバッガが関数から正常に戻るように、呼出しフレーム情報を最新に保つための情報。[呼出しフレーム情報](#)も参照してください。

バンク

[メモリバンク](#)を参照。

バンク切替え

異なるメモリバンクの切替え。このソフトウェア技術を使用することで、メモリの異なる部分が同一のアドレス空間を占有できるため、コンピュータの使用可能メモリが増加します。

バンクコード

複数のメモリバンクに分散したコード。各関数はそれぞれ1つのバンクだけにしか常駐できません。

バンクデータ

複数のメモリバンクに分散したデータ。各データオブジェクトがそれぞれ1つのメモリバンク内に収まる必要があります。

バンクメモリ

同一アドレス用に複数の格納場所があるメモリ。
メモリバンクも参照してください。

バンク切替えルーチン

メモリバンクを選択するコード。

パッチファイル

コマンドラインインタプリタで実行されるオペレーティングシステムコマンドを記述したテキストファイル。UNIXでは、コマンドラインインタプリタがUNIXシェルスに含まれているため、「シェルスクリプト」と呼びます。パッチファイルを使用して、既存のコマンドを組み合わせ、新しいコマンドとして実行することができます。

ビットフィールド

1単位として見なされるビットのグループ。

リンカ設定ファイル内のブロック

連続するコードまたはデータ。ブロック、オーバーレイ、セクションのいずれかで構成され、空の場合もあります。ブロックには名前があり、ブロックの開始および終了アドレスはアプリケーションから参照できます。また、ブロックには、最大サイズ、特定のサイズ、または最小アラインメントなどの属性を指定できます。内容の順序は固定または任意です。

ブレークポイント

1. コードブレークポイント: プログラム中、そこに到達するとデバッグ用の特殊な処理が実行される地点。通常は、プログラムの実行の停止や、プログラムの変数の一部または全部のダンプを行う箇所に、ブレークポイントを使用します。ブレークポイントは、プログラムの実行を詳細に検証する場合にプログラムそのものの一部として、またはプログラマがデバッグツールでの対話セッションの一部として設定します。

2. データブレークポイント: メモリ中で、そこにアクセスするとデバッグ用の特殊な処理が実行される地点。通常は、リード/ライト処理のいずれかでアドレス位置がアクセスされてプログラムの実行を停止する場合に、データブレークポイントを使用します。

3. イミディエイトブレークポイント: メモリ中で、そこにアクセスするとデバッグ用の特殊な処理が実行される地点。通常は、メモリアクセス命令の実行中（アクセスの種類に応じて、実際のメモリアクセスの前後）に、プログラム実行を一時停止してユーザが指定したアクションを実行する場合に、イミディエイトブレークポイントを使用します。実行はその後再開されます。この機能は、C-SPYのシミュレータバージョンでのみ使用できます。

C

呼出しフレーム情報

C関数をコンパイルしたコードで、完全な関数の呼出しスタック（呼出しスタック）を、プログラムカウンタの位置に関わらず、また実行に影響を及ぼすことなく、IAR C-SPY® デバッガで表示できるようにするための情報。**バックトレース**も参照してください。

呼出し規約

プログラム内の関数が別の関数を呼び出す方法を規定したもの。レジスタパラメータの処理方法、値を返す方法、呼出し先関数が保持するレジスタなどが規定されています。C/C++ 関数では、すべてコンパイラが自動的に処理します。アセンブラ言語で記述したコードの場合は、C/C++ 関数からの呼出しや、C/C++ 関数の呼出しを実行できるように、呼出し規約のルールに従う必要があります。Cの呼出し規約およびC++の呼出し規約は、同一でない場合があります。

安価

安価なメモリアccessのように使用します。安価なメモリアccessでは、実行にかかるサイクル数や、実装に必要なコードバイト数が少なくなります。メモリアccessが安価なことを、低コストと言います。**メモリアccessコスト**を参照。

チェックサム

アプリケーションの全体または一部の ROM の内容に基づいて計算され、データの破壊を検出するためにアプリケーションに格納される値。チェックサムは、アプリケーションで検証するためにリンクで生成されます。複数のアルゴリズムがサポートされています。**CRC (巡回冗長検査)**と比較してください。

コードバンキング

バンクコードを参照。

コードモデル

コードモデルは、アプリケーション用コードの生成方法を制御します。通常は、コードモデルは、関数の呼出し方法や関数が配置されるコードセクションなどの挙動を制御します。アプリケーションのすべてのオブジェクトファイルは、同一のコードモデルを使用してコンパイルする必要があります。

コードポインタ

コードポインタとは、関数ポインタを意味します。多くのコアでは複数の異なる方法で関数を呼び出せるため、組み込みシステム用のコンパイラでは通常はこれらの方法をすべて使用できます。

コードポインタとデータポインタを混同しないでください。

コードセクション /

コードを含むリードオンリーセクション。**セクション**も参照してください。

コンパイル単位

翻訳単位を参照。

コンパイラオプション

コンパイラのデフォルトの動作を変更するためのパラメータ。

コスト

メモリアクセスコストを参照。

CRC (巡回冗長検査)

データ損傷を検出するため、データブロックから計算してデータブロックと共に保存する値。**CRC**は、多項式を使用して計算される値で、単純な演算によるチェックサムよりも高度なエラー検出方法です。**チェックサム**と比較してください。

C-SPY オプション

IAR C-SPY デバッガのデフォルトの動作を変更するためのパラメータ。

Cstartup

アプリケーションの実行開始前にシステムを設定するコード。

C 形式のプリプロセッサ

プリプロセッサは、実際のコンパイル前に入力ストリームを前処理するスタンドアロンアプリケーションかコンパイラ内蔵機能です。C 形式のプリプロセッサは、標準の C で設定された規則に従い、**#define**、**#if**、**#include**などのテキストマクロ置換、条件付きコンパイル、他のファイルのインクルードなどを処理するためのコマンドを実装します。

D

データバンキング

バンクデータを参照。

データモデル

データモデルは、デフォルトのメモリタイプを指定します。言い換えれば、このデータモデルで通常、下記の 1 つまたは複数を制御します。静的 / グローバル変数、動的に割り当てられたデータ、ランタイムスタックに、アクセスするために使用される方法および生成されるコードを、制御します。また、デフォルトのポインタタイプと、静的 / グローバル変数が配置されるデータセクションも制御します。1 つのプロジェクトで同時に使用できるデータモデルは 1 つだけです。また、プロジェクト内のすべてのユーザモジュールとライブラリモジュールで同一のモデルを使用する必要があります。

データポインタ

多くのコアでは、異なるメモリタイプやアドレス空間にアクセスするため、複数のアドレッシングモードがあります。通常は、組み込みシステム用コンパイラでは、空きメモリに効率的にアクセスできるように、複数のデータポインタタイプセットに対応しています。

データ表現

データタイプのメモリでの配置方法、データタイプが表現する値の範囲。

宣言

オブジェクト（変数、関数）が存在することをコンパイラに対して明示することを指します。オブジェクトそのものは、1つの翻訳単位（ソースファイル）だけで定義する必要があります。オブジェクトは、使用前に宣言し、定義しておく必要があります。通常は、多くのファイルで使用するオブジェクトを1つのソースファイルで定義します。オブジェクトの宣言はヘッダファイルに記述し、そのオブジェクトを使用するファイルでそのヘッダファイルをインクルードします。

次に例を示します。

```
/* 変数 "a" がどこかに存在。関数  
"b" は2つの int パラメータを取得して1つの  
int. を返します。*/
```

```
extern int a;  
int b(int, int);
```

定義

変数か関数そのものを指します。プリケーションの各変数 / 関数につき1つだけ、定義を記述できます。

仮定義も参照してください。

次に例を示します。

```
int a;  
int b(int x, int y)  
{  
    return x + y;  
}
```

デマングル

マングル化された名前をより一般的な C/C++ 名に復元すること。 **マングル化**も参照してください。

デバイス記述ファイル

入出力レジスタ (SFR) 定義、割込みベクタ、制御レジスタ定義などのデバイス固有の情報を含む、C-SPY で使用されるファイル。

デバイスドライバ

高水準のプログラミングインタフェースを周辺デバイスに提供するソフトウェア。

デジタル信号プロセッサ (DSP)

マイクロプロセッサに類似するデバイスで、内部 CPU が離散時間信号処理用に最適化されています。デジタル信号プロセッサは、マイクロプロセッサの標準命令に加えて、一般的な信号処理計算を高速に実行するための複雑な命令セットもサポートしています。

逆アセンブリウィンドウ

メモリの内容を逆アセンブルしてマシン命令に変換し、可能であれば、対応する C ソースコードを挿入して表示する C-SPY ウィンドウ。

DWARF

ソースレベルデバッグをサポートする業界標準デバッグフォーマット。これは、オブジェクトでデバッグ情報を表すときに IAR ILINK リンカで使用されるフォーマットです。

動的初期化

C で記述されたプログラム内の変数は、実行の初期段階で（main 関数が呼び出される前に）初期化されます。これらの変数は、コンパイル時やリンク時に決定される静的な値で初期化されます。これを静的初期化と呼びます。C++ では、グローバルオブジェクトのコンストラクタや、動的メモリ割当てなどのコードを実行することで、変数の初期化が必要な場合があります。

動的メモリ割当て

変数の保存には、リンク時に静的に行う方法と、実行時に動的に行う方法の2つがあります。動的メモリ割当ては、多くの場合はヒープから実行されます。ヒープのサイズにより、動的オブジェクトや変数に使用可能

なメモリ量が決定されます。動的メモリ割当てには、同時に使用されない複数の変数やオブジェクトを同一メモリに格納することで、アプリケーションに必要なメモリ量を削減できるという利点があります。**ヒープメモリ**も参照してください。

動的オブジェクト

実行時に割当て、作成、破棄、解放が行われるオブジェクト。動的オブジェクトは、ほとんどの場合、動的に割り当てられたメモリに格納されます。**静的オブジェクト**と比較してください。

E

EEPROM

Electrically Erasable, Programmable Read-Only Memory（電氣的消去可能プログラマブルリードオンリーメモリ）の略。電子的に消去して書き換えることが可能な ROM。

ELF

Executable and Linking Format、業界標準オブジェクトファイルフォーマット。これは、IAR ILINK リンカにより使用されるフォーマットです。デバッグ情報は DWARF を使用してフォーマット化されます。

Embedded C++

組込みシステムのプログラミング用に設計された、C++ プログラミング言語のサブセット。言語の設計時に、組込みシステム開発で性能と移植性が特に重要であることが考慮されています。

組込みシステム

特定用途向けに設計されたハードウェアとソフトウェアの組合せ。組込みシステムがより大規模なシステムや製品の一部分となっている場合も多数あります。

エミュレータ

プロセッサファミリの派生品のエミュレーションを実行するハードウェアデバイス。エミュレータは、しばしば実際のコアの代わりに使用し、プリント基板（実際の用途ではコアを接続）に接続デバイス経由で接続

します。エミュレータは、常にターゲットプロセッサと完全に同様に動作し、デバッグですべてのシステムアクチュータが必要な場合や、デバイスドライバをデバッグする場合に使用します。

Enea OSE Load モジュールフォーマット

OSE オペレーティングシステムでロード可能な特別な ELF フォーマット。**ELF** も参照してください。

列挙型

その型の変数で可能なすべての値のリストを定義に含む型。一般的な例として、[true, false] のリスト中のいずれかの値を取るブール値や、[Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday] のいずれかの値を取る曜日などがあります。列挙型は、C や Ada などの型付き言語の機能です。

文字、整数（サイズ固定）、浮動小数点数などの型も、大きな意味では列挙型と見なされる場合があります（通常は列挙型には属さない）。

EPROM

Erasable, Programmable Read-Only Memory（消去可能プログラマブルリードオンリーメモリ）の略。紫外線の照射により消去した後に、書き換えることが可能な ROM。

実行可能イメージ

実行可能なイメージが含まれます。いくつかの再配置可能オブジェクトファイルのリンク結果とライブラリで構成されます。オブジェクトファイルに使用されるファイルフォーマットは、ILINK ではデバッグ情報用の組込み DWARF を含む ELF です。

例外

プロセッサハードウェア、メモリ管理ユニット (MMU) などの、プロセッサと緻密に結合されたハードウェアが開始する割込み。アーキテクチャルールの違反（保護メモリへのアクセス）や、極端なエラー状態（ゼロによる除算）を示します。

この用語と、C++ 言語（Embedded C++ を除く）で使用される **例外** という用語とを混同しないでください。

高価

高価なメモリアクセスのように使用します。高価なメモリアクセスでは、実行にかかるサイクル数か、実装に必要なコードバイト数が多くなります。メモリアクセスが高価なことを、高コストであると言います。**メモリアクセスコスト**を参照。

拡張キーワード

C/C++ での非標準キーワード。通常は、オブジェクト、定義、宣言（データ、関数）を制御します。**キーワード**も参照してください。

F

フィル

実行可能イメージのセクション間に存在するバイト（特定のフィルパターンを使用）を埋めること。これらのバイトは、セクションのアラインメント要求のために存在します。

フォーマット指定子

printf などのライブラリ関数で文字列のフォーマットを指定するために使用します。次の例では、関数呼出しでフォーマット指定子 %c を 1 つ含むフォーマット文字列を 1 つ指定しており、a の値を 1 つの ASCII 文字として出力します。

```
printf("a = %c", a);
```

G

一般オプション

IDE に含まれる全ツールのデフォルトの動作を変更するためのパラメータ。

汎用ポインタ

ハーバードアーキテクチャベースのコアなどで、すべてのメモリタイプを示すことができるポインタ。

H

ハーバードアーキテクチャ

ハーバードアーキテクチャベースのコアは、独立したデータベースと命令バスを備えています。これにより、並列実行が可能となっています。命令のフェッチ中に、現在の命令がデータベースで実行されます。現在の命令が完了すると、次の命令をすぐに実行できます。これにより、理論上はノイマンアーキテクチャよりも大幅に高速な実行が可能です。ただし、回路は複雑になります。**ノイマンアーキテクチャ**と比較してください。

ヒープメモリ

ヒープとは、システムで動的メモリ割当て用に確保されたメモリプールです。ヒープの一部をアプリケーション専用に使用することができます。ヒープから割り当てたメモリは、アプリケーションが明示的に解放してヒープに戻すまで有効です。このタイプのメモリは、アプリケーションを実行するまで必要なオブジェクト量がわからない場合に便利です。このタイプのメモリは、メモリ容量が限られているシステムや、長期間実行するシステムで使用すると問題が生じることがあります。

ヒープサイズ

動的に割当て可能なメモリの合計サイズ。

ホスト

ターゲットプロセッサと通信するコンピュータ。この用語は、デバッガを実行するコンピュータと、開発した組込みアプリケーションを実行するコアとを区別するために使用します。

I

Iarchive

アーカイブ、つまりライブラリを作成する IAR システムズのユーティリティ。Iarchive は、IAR Embedded Workbench に付属しています。

IDE（統合開発環境）

必要なすべてのツールを 1 つのアプリケーションに統合したプログラミング環境。

lelfdumparm

IAR システムズのユーティリティ。ELF 再配置可能イメージまたは実行可能イメージの内容のテキスト表示を作成するために使用します。

lelftool

IAR システムズのユーティリティ。ELF 実行可能イメージ上でさまざまな変換（フィル、チェックサム、フォーマット変換など）を実行するために使用します。

ILINK

ELF/DWARF フォーマットで絶対出力を生成する IAR ILINK リンカ。

ILINK 設定

使用可能な物理メモリの定義およびこれらのメモリに対するセクション（コードやデータ）の配置。ILINK では、実行可能イメージを構築する設定が必要です。

イメージ

実行可能イメージを参照。

インクルードファイル

ソースファイルにインクルードされるテキストファイル。この処理は、多くの場合はプリプロセッサが実行します。

リンカ設定ファイル内の初期化設定

イニシャライザによる RAM セクションの初期化方法の定義。通常、定数でなく、`noinit` 以外の変数のみが初期化されます。たとえば、コードの一部も初期化できます。

初期化済みセクション

起動時に特定の値で初期化されるリード/ライトセクション。**セクション**も参照してください。

インラインアセンブラ

C 言語の文の間に直接挿入するアセンブラ言語。

インライン化

呼出し先関数の本体を関数呼出しに置き換える最適化処理。この最適化により実行速度が向上し、場合によっては生成コードのサイズも削減できます。

命令ニーモニック

アセンブラ言語で、マシン命令を表現するために使用される語や頭辞語。プロセッサによって命令セットが異なるため、ADD、BR（分岐）、BLT（値が小さい場合に分岐）、MOVE、LDR（レジスタのロード）などの命令を表現するニーモニックセットも異なります。

割込みベクタ

割込み発生時に実行されるコードの一部か、そのコードを示すポインタ。

割込みベクタテーブル

割込みベクタをタイプ別にインデックス化して格納したテーブル。このテーブルには、プロセッサでの割込みと割込みサービスルーチンのマッピングが格納され、プログラマが初期化する必要があります。

割込み

組込みシステムでは、割込みを使用して、タイマオーバフローやボタンが押されたときなどの外部イベントを即座に検出します。

割込みは、通常処理を一時停止し、制御フローを「割込みハンドラ」ルーチンに一時的に渡す非同期イベントです。割込みは、ハードウェア（入出力、タイマ、マシンチェック）とソフトウェア（モニタプログラム、システム呼出し、トラップ命令）の両方により発生します。**トラップ**と比較してください。

組込み

ネイティブのコンパイラオブジェクト、プロパティ、イベント、メソッドを意味する形容詞。

組込み関数

1. 特定のマシンコードシーケンスに直接展開される関数呼出し。2. コンパイラが内部的用途（浮動小数点演算など）で呼び出す関数。

lobjmanip

ELF オブジェクトファイルの低レベルの操作に使用する IAR システムズのユーティリティ。

K

キーバインディング

IDE で使用するメニューコマンド用キーショートカット。

キーワード

プログラミング言語の構文で定義されているシンボルセット。言語で使用されるすべてのキーワードは予約済みで、識別子（変数やプロシージャなどのユーザ定義オブジェクト）として使用することはできません。

拡張キーワードも参照してください。

L

L 値

代入文の左辺の変更可能な値。単純な変数、逆参照されたポインタがこれに該当します。(x + 10) のような式には新しい値を代入できないため、L 値にはなりません。

言語拡張

ターゲット固有の C 言語拡張。

ライブラリ

ランタイムライブラリを参照。

ライブラリ設定ファイル

ランタイムライブラリの設定が記述されたファイル。このファイルでは、ランタイムライブラリに含まれる機能が定義されています。ランタイムライブラリのビルドを調整するために使用されます。**ランタイムライブラリ**も参照してください。

リンカ設定ファイル

実行可能イメージを構築するときに IAR ILINK リンカにより使用される設定を含むファイル。**ILINK 設定**も参照してください。

ローカル変数

自動変数を参照。

ロケーションカウンタ

プログラムロケーションカウンタ (PLC) を参照。

論理アドレス

仮想アドレス (論理アドレス) を参照。

M

MAC (積和演算)

乗算を加算と共に実行する特殊な命令、オンチップデバイス。次の形式のフィルタや変換を多数使って信号処理を実行する場合に多用されます。

$$y_j = \sum_{i=0}^N c_i \cdot x_{i+j}$$

MAC のアキュムレータは、通常のレジスタより高精度（ビット数が多い）です。**デジタル信号プロセッサ (DSP)** も参照してください。

マクロ

1. アセンブラマクロは、ユーザ定義のアセンブラ行セットであり、後で指定のマクロ名を参照することにより、ソースファイルに展開されます。参照時には、パラメータの置換が行われます。

2. C マクロは、ソースファイルの前処理中に使用されるテキスト置換の仕組みです。マクロは、`#define` プリプロセッサディレクティブを使用して定義します。それ以降の翻訳単位でマクロ名が記述された箇所が、各マクロに対応する置換用テキストに置換されます。

3. C-SPY マクロは、C-SPY の機能を拡張するためにユーザが記述できるプログラムです。C-SPY マクロは、典型的な例として、ブレークポイントに対応付けて使用します。ブレークポイントに到達したときにそのマクロを実行し、周辺デバイスのシミュレーション、複雑な条件の評価、トレースの出力などを行うことができます。

C-SPY マクロ言語は、C の簡易版ですが、C ほど厳密なデータ型がありません。

メールボックス

RTOS でのメールボックスとは、複数のタスク間の通信拠点です。タスクは、別のタスクのメールボックスにメッセージを保存することで、そのタスクにメッセージを送信できます。メールボックスは、メッセージキュー、メッセージポートとも呼びます。

マングル化

マングル化とは、複雑な C/C++ 名を簡単な名前にマッピングするときに使用される技術です。ILINK メッセージの C/C++ シンボルに対して、マングル化した名前とデマングル化した名前の両方を生成できます。

メモリ、リンカ設定ファイル内

物理メモリ。物理メモリに含まれるユニット数および 1 つのユニットを構成するビット数。リンカ設定ファイルで定義されます。メモリは、常に 0x0 ~ (サイズ-1) からアドレスできます。

メモリアクセスコスト

メモリアクセスコストは、アクセス実行に必要なクロックサイクル数かコードのバイト数で示されます。サイズの大きな命令や多数の命令が必要なメモリは、よりサイズが小さい命令や少ない命令でアクセスできるメモリよりもアクセスコストが高い、というように使用します。

メモリエリア

メモリの領域を意味します。

メモリバンク

バンクメモリ内のシーケンシャルメモリの最小単位。コアの物理アドレス空間で一度に認識できるメモリバンクは 1 つです。

メモリマップ

コアで使用可能なさまざまなメモリエリアのマップ。

メモリモデル

メモリ階層やシステムが処理できるメモリ容量を示します。アプリケーションで同時に使用できるメモリモデルは 1 つだけです。また、すべてのユーザモジュールやライブラリモジュールで同一のモデルを使用する必要があります。

マイクロコントローラ

組込みシステムとして動作する 1 つの集積回路上のマイクロプロセッサ。マイクロコントローラは、CPU に加え、小容量の RAM、PROM、タイマ、入出力ポートを内蔵しています。

マイクロプロセッサ

1 つ（または少数の）集積回路に内蔵された CPU。シングルチップマイクロプロセッサには、メモリ、メモリ管理、キャッシュ、浮動小数点演算ユニット、入出力ポート、タイマなどのコンポーネントを内蔵できます。このようなデバイスを、マイクロコントローラとも呼びます。

モジュール

オブジェクト。オブジェクトファイルはモジュールを含み、ライブラリは 1 つ以上のオブジェクトを含みます。リンクの基本単位。モジュールには、シンボル定義（エクスポート）や外部シンボルへの参照（インポート）が含まれます。C/C++ のコンパイル時には、翻訳単位ごとに 1 つモジュールが生成されます。

複数ファイルのコンパイル

コンパイラで複数のソースファイルを 1 つのコンパイルユニットとしてコンパイルするテクニック。これにより、コンパイルユニット内の複数のソースファイルでのインライン化、クロスコール、クロスジャンプなど、プロシージャ間の最適化が可能になります。

N

ネスト割込み

割込みに対して別の割込みを実行できるシステムを、ネスト割込み機能を持つと言います。

非バンクメモリ

コアの物理アドレス空間で、各メモリアドレスにつき 1 つ格納場所があること。

非初期化メモリ

リセット時に任意の値を持つことができる、またはソフトリセット時にリセット前の値を保持できるメモリ。

No-init セクション s

起動時に初期化されないリード/ライトセクション。
*セクション*も参照してください。

不揮発性ストレージ

バッテリーバックアップ RAM、ROM、磁気テープ、磁気ディスクなどの、電源を切断してもデータを保持できるメモリデバイス。*揮発性ストレージ*と比較してください。

NOP

No operation（無動作命令）の略。何の処理も実行せず、遅延を発生させるために使用する命令。パイプラインアーキテクチャでは、NOP 命令を使用して、パイプラインを同期させることができます。*パイプライン*も参照してください。

O

Objcopy

ELF フォーマットの絶対オブジェクトファイルを、たとえば、フォーマット Motorola-std や Intel-std などの絶対オブジェクトファイルに変換する GNU バイナリユーティリティ。*elftool* も参照してください。

オブジェクト

ライブラリメンバのオブジェクトファイル。

オブジェクトファイル、絶対

*実行可能イメージ*を参照。

オブジェクトファイル、再配置可能

ソースファイルをコンパイルまたはアセンブルした結果。オブジェクトファイルに使用されるファイルフォーマットは、*ILINK* ではデバッグ情報用の組込み DWARF を含む ELF です。

演算子

関数として使用されるシンボルで、引数が 2 つある場合は中置記法（+ など）、引数が 1 つだけの場合は前置記法（ビット単位の否定を示す ~ など）で使われます。多くの言語では、算術演算や論理演算などの組込み関数に演算子を使用します。

演算子の優先順位

それぞれの演算子には優先順位が設定され、演算子とオペランドが評価される順番はそれによって決定されます。優先順位が一番高い演算子が最初に評価されます。演算子およびオペランドをグループ化し、式の評価順序を変更するには、括弧を使用します。

オプション

コンパイラやリンカなどツールの動作を制御するコマンドのセット。オプションは、コマンドラインや IDE によって指定できます。

出カイメージ

リンク後の結果のアプリケーション。この用語は、IAR システムズのユーザドキュメントで使用される用語である、*実行可能イメージ*と同じです。

オーバレイ、リンカ設定ファイル内

ブロックと似ているが、それぞれがブロック、オーバレイ、セクションで構成されるいくつかの重複エンティティを含む。オーバレイのサイズは、その最大要素で決定されます。

P

パラメータの受渡し

*呼出し規約*を参照。

周辺ユニット

メモリや入出力デバイスなど、プロセッサ以外のハードウェアコンポーネント。

パイプライン

計算が流れる一連のステージで構成される構造。他の処理がパイプライン経由で実行中でも、パイプラインの開始地点で新しい処理を開始できます。

配置、リンカ設定ファイル内

ブロック、オーバレイおよびセクションを領域に配置する方法。コードおよびデータが、使用可能な物理メモリに実際にどのように配置されるかを決定します。

ポインタ

指定した型の他のオブジェクトのアドレスを格納するオブジェクト。

#pragma

C/C++ プログラムのコンパイル中に、`#pragma` プリプロセッサディレクティブが検出されると、コンパイラを処理系定義に従って動作させます。これには、コンソールでの出力生成、それ以降のオブジェクトの宣言の変更、最適化レベルの変更、言語拡張の有効/無効の切替えなどがあります。

プリエンティブマルチタスク

RTOS のタスクは、より優先順位の高いプロセスが有効になるまでの間、実行を許可されます。割込みの結果、優先順位の高いタスクが有効になる場合があります。プリエンティブとは、タスクが一定の実行時間（タイムスライス）を割り当てられている場合でも、プロセッサの使用権を失うことがあることを意味します。割込みが発生するごとに、タスクスケジューラはそのときに有効なタスクの中で優先順位が最高のものを特定し、そのタスクに処理を切り替えます。特定されたタスクが割込み前に実行されていたタスクと異なる場合は、前のタスクは割込み時点の状態で一時的に停止します。

ラウンドロビンと比較してください。

プリプロセッサディレクティブ

実際のコードの解析を開始する前に実行されるディレクティブ。

プリプロセッサ

C 形式のプリプロセッサを参照。

派生プロセッサ

コンパイラがサポートする別のチップ構成。

プログラムカウンタ (PC)

命令のアドレッシングに使用する特殊なプロセッサレジスタ。プログラムロケーションカウンタ (PLC) と比較してください。

プログラムロケーションカウンタ (PLC)

IAR アセンブラで、現在の命令のコードアドレスを指定する際に使用します。PLC は、算術式で使用できる特別なシンボル（通常は \$）で表現されます。単にロケーションカウンタ (LC) と呼びます。

プロジェクト

ユーザアプリケーション開発プロジェクト。

プロジェクトオプション

アプリケーションを実行するターゲットプロセッサなどの、プロジェクト全体に適用される一般オプション。

PROM

Programmable Read-Only Memory（プログラマブルリードオンリーメモリ）の略。1 回だけライト可能な ROM。

Q

修飾子

型修飾子を参照。

R

範囲、リンカ設定ファイル

メモリ内での連続するアドレスの範囲。領域は、範囲で構成されます。

リードオンリーセクション

コードや定数を含むセクション。セクションも参照してください。

リアルタイムオペレーティングシステム (RTOS)

割込みが発生してから割込みハンドラが開始されるまでの遅延と、タスクのスケジューリング方法を保証するオペレーティングシステム。一般的に RTOS は、通常のデスクトップ用オペレーティングシステムよりも大幅に小さなサイズとなっています。リアルタイムシステムと比較してください。

リアルタイムシステム

プロセスが時間に依存するコンピュータシステム。リアルタイムオペレーティングシステム (RTOS) と比較してください。

領域、リンカ設定ファイル

重複しない範囲のセット。範囲は、1 つ以上のメモリに存在できます。ILINK の場合はブロック、オーバーレイ、セクションは、リンカ設定ファイルの領域に配置されます。

領域式、リンカ設定ファイル内

領域リテラル、領域、リンカ設定ファイルで使用できる共通セット操作で構成される領域。

領域リテラル、リンカ設定ファイル内

メモリ内で重複しない 1 つ以上の範囲セットを定義するリテラル。

レジスタ

特にアクセス速度が高速で、プログラム実行時の一時記憶エリアとして確保されている小型オンチップメモリユニット。通常の容量は数バイトです。

レジスタ定数

システム初期化の際に、プロセッサの専用レジスタにロードされる値。コンパイラは、定数が専用レジスタに格納されていることを前提に、コードを生成することができます。

レジスタロック

通常のコード生成時に、コンパイラで一部のプロセッサレジスタの使用を禁止することを指します。多くの状況で使用します。たとえば、高速化のため、システムの一部をアセンブラ言語で記述する場合があります。この部分に、専用のプロセッサレジスタを割り当てる場合もあります。また、オペレーティングシステムやサードパーティ製ソフトウェアでレジスタが使用される場合もあります。

レジスタ変数

通常、レジスタ変数は、関数の（スタック）フレームの代わりにレジスタに格納されるローカル変数を指します。レジスタ変数は、メモリアccessが不要で、コンパイラでレジスタ変数を使用することで命令の実行時間を短縮できるため、他の変数よりも大幅に高速です。

自動変数も参照してください。

リレー

ベニアの同義語。ベニアを参照。

再配置可能セクション

セクション。

リセット

システムの初期状態から再起動することを指します。リセットは、ハードウェア（ハードリセット）またはソフトウェア（ソフトリセット）から実行できます。ハードリセットは通常は電源投入と区別できませんが、ソフトリセットは区別できます。

ROM モニタ

デバッグツールでの使用に特化した組込みソフトウェア。評価ボードチップの ROM に格納されていて、シリアルポートかネットワーク接続経由でデバッガと通信します。ROM モニタは、メモリアドレス（ロケーション）やレジスタの表示と修正、ブレークポイントの作成と削除、アプリケーションの実行などの基本コマンドセットを提供します。デバッガは、これらの基本コマンドを組み合わせて、プログラムのダウンロードやステップ実行など、より高度な機能を実現できます。

ラウンドロビン

オペレーティングシステムでのタスクスケジュール。ここでは、すべてのタスクの優先順位レベルが同じであり、1 つずつ順番に実行されます。プリエンプティブマルチタスクと比較してください。

RTOS

リアルタイムオペレーティングシステム (RTOS) を参照。

ランタイムライブラリ

オブジェクトファイルから参照される場合のみ、つまり条件付きでリンクされる場合のみ、実行可能イメージに含まれる再配置可能オブジェクトファイルの集合。

ランタイムモデル属性

相互に互換性のない複数のモジュールがアプリケーションにリンクされないようにする仕組み。ランタイム属性は、名前付きのキーと対応する値のペアで構成されます。

ILINK は、ライブラリを自動的に選択するときに、ランタイムモデル属性を使用して、正しいライブラリが使用されているか確認します。

R 値

代入文の右辺に指定可能な値。単純に値だけがこれに該当します。L 値も参照してください。

S

飽和演算

ほとんどの C/C++ の実装では、 $\text{mod-}2^N$ の補数ベースの演算を使用します。オーバフロー時には、定義域で値がラップされます。つまり、 $(127 + 1) = -128$ となります。一方、飽和演算では、定義域でのラップが許可されていません。たとえば、定義域の上限値が 127 の場合、 $(127 + 1) = 127$ となります。飽和演算は、ラップが許可されているとオーバフロー状態が致命的な問題になる信号処理で、よく使用されます。

スケジューラ

RTOS でタスク切替を担当する部分。また、実行を許可するタスクの選択も担当します。スケジューリングアルゴリズムには多種ありますが、ほとんどは静的スケジューリング（コンパイル時に実行）または動的スケジューリング（次に実行するタスクを、タスク切替時のシステムの状態に応じて実行時に選択）のいずれかです。ほとんどのリアルタイムシステムでは、システムのリアルタイム要件違反を排除できるため、静的スケジューリングが使用されています。

スコープ

アプリケーションコード内で、関数や変数を名前で参照できる部分。ある項目のスコープは、ファイル、関数、ブロックのいずれかに制限されることがあります。

セクション

データまたはテキストのいずれかを含むエンティティ。通常は 1 つ以上の変数または関数です。セクションは、最小のリンク可能ユニットです。

セクション属性

各セクションは名前と属性を持つ。属性は、セクションの内容、つまり、セクションの内容がリードオンリー、リード/ライト、コード、データなどを定義します。

セクションフラグメント

セクションの一部。通常は変数または関数です。

セクション選択

リンカ設定ファイルにおいて、セクションセレクトアを使用してセクションのセットを定義します。セクションは、複数の選択の一部となる可能性がある場合、最も制限の厳しいセクションセレクトアに属します。セレクトアには、セクション属性（セクションの内容で選択）、セクション名（セクション名で選択）、オブジェクト名（特定のオブジェクトから選択）の 3 種類があり、これらは個別に使用したり、組み合わせて使用してセクションのセットを選択したりできます。

セマフォ

リソースへの排他的アクセスを保証するために使用するフラグの一種。リソースとしては、ハードウェアポート、構成メモリ、変数などがあります。複数のタスクが同一リソースにアクセスする必要がある場合は、リソースにアクセスする部分のコード（クリティカルセクション）をすべてのタスクに対して排他的にする必要があります。これには、そのリソースを保護するセマフォを取得し、他のタスクからそのリソースを遮断します。他のタスクがそのリソースを使用する場合は、そのタスクもセマフォを取得する必要があります。セマフォが使用中の場合は、セマフォが解放されるまで待機する必要があります。セマフォが解放された後は、2 番目のタスクが実行を許可され、セマフォを取得してリソースへの排他的アクセスを実行できます。

重要度

何らかの問題を検出したときにアセンブラ、コンパイラ、デバッガから返される診断応答の重要度。通常、重要度は、リマーク、ワーニング、エラー、致命的なエラーの 4 段階です。リマークは問題の可能性を示すだけです。致命的なエラーの場合はプログラミングツールが処理の完了前に終了したことを示します。

共有

いくつかの方法でアドレスが可能な物理メモリ。ILINK の場合は、リンカ設定ファイルで定義します。

ショートアドレッシング

多くのコアでは、内部 RAM、メモリマップド I/O へのアクセスを効率的に行うため、特別なアドレッシングモードがあります。データポインタも参照してください。

副作用

C/C++ の式がシステムの状態を変更することを、副作用があると言います。例として、変数への代入や、変数に後置インクリメント演算子を使用する場合などがあります。C/C++ の規格では、副作用のある変数を式で複数回使用しないように規定されています。たとえば、次の文はこのルールに違反します。

```
*d++ = *d;
```

シグナル

シグナルは、イベントベースのタスク間通信を提供します。1つのタスクは、他の1つ以上のタスクからのシグナルを待つことがあります。待っているシグナルをタスクが受信すると、実行が続行されます。RTOS では、シグナルを待つタスクは処理時間を費やさないため、他のタスクを実行できます。

シミュレータ

ホスト上で実行し、ターゲットプロセッサと可能な限り同一に動作するデバッグツール。シミュレータは、ハードウェアが使用できないときか、ハードウェアをデバッグに使わないときに、アプリケーションのデバッグのために使用します。物理的な周辺デバイスには通常接続しません。シミュレーションされたプロセッサは、多くの場合は実際のハードウェアよりも（場合によっては大幅に）低速になります。

ステップ実行

デバッガで一度に1つずつ命令やC言語の文を実行することを指します。

スケルトンコード

ユーザがコードを特定用途化できる、未完成のコードフレームワーク。

特殊機能レジスタ (SFR)

コアのハードウェアコンポーネントに対するリード/ライトに使用するレジスタ。

スタックフレーム

データオブジェクト（保持レジスタ、ローカル変数、特定のスコープ用に一時的に保持する必要のある他のデータオブジェクト）を含むデータ構造（通常は関数）。

以前のコンパイラでは、関数全体でスタックフレームのサイズとレイアウトが固定されていましたが、最近のコンパイラでは、関数内の任意の箇所/時間で、非常に動的にレイアウトとサイズを変更できる場合があります。

スタックセクション

スタックのエリアを予約するセクションまたはセクション。ほとんどのプロセッサは呼出しとパラメータで同一のスタックを使用しますが、一部のプロセッサでは個別のスタックを使用します。

標準ライブラリ

C/C++ 標準で定義されている C/C++ ライブラリ関数、および浮動小数点ルーチンなどのコンパイラのサポートルーチン。

静的オブジェクト

リンク時にメモリが割り当てられ、システム起動時（または最初の使用時）に作成されるオブジェクト。**動的オブジェクト**と比較してください。

静的オーバーレイ

パラメータや自動変数に動的配置方式を使用する代わりに、リンク時にパラメータや自動変数にエリアを割り当てます。この方法ではスタックの使用効率は最悪になりますが、スタックアクセスが高価な、またはスタックアクセスがまったくない小型チップには、適している場合があります。

静的割当てメモリ

この種のメモリは、リンク時に1度だけ割り当てられ、アプリケーションの実行終了まで有効です。global または static として宣言された変数が、この方法で割り当てられます。

構造体値

構造体および共用体の集合名。構造体は、メモリに連続的に配置されたデータオブジェクトの集合です（データオブジェクト間にパッドバイトが挿入されていることもある）。共用体は、同一メモリアドレス（ロケーション）を共有するデータの集合です。

シンボル位置

正確なアドレスがわからないためにシンボル名を使用している位置。

T

ターゲット

1. アーキテクチャ。2. ハードウェア。アプリケーション開発対象の組込みシステムを指します。この用語は、通常はシステムとホストシステムの区別に使用します。

タスク (スレッド)

タスクは、システムでの実行スレッドです。多くの並列で実行されるタスクを含むシステムを、マルチタスクシステムと呼びます。プロセッサは一度に1つの命令ストリームだけを実行するため、ほとんどのシステムは何らかのタスク切替えメカニズム（多くの場合コンテキスト切替えと呼ぶ）を実装し、処理時間をすべてのタスクに配分します。次に実行を許可するタスクの決定プロセスを、スケジューリングと呼びます。一般的なスケジューリング方法として、**プリエンプティブマルチタスク**と**ラウンドロビン**があります。

仮定義

定義が同一で、絶対アドレスである場合に、複数のファイルで定義可能な変数。

ターミナル I/O

C-SPY の端末シミュレーションウィンドウ。

タイマ

プログラム実行とは無関係にカウントを実行する周辺デバイス。

タイムスライス

RTOS で、タスクスケジューリングアルゴリズムを実行せずに1つのタスクを実行可能な（最長）時間。タスク切替えまでに、複数の連続したタイムスライスにわたって1つのタスクが実行されることがあります。また、プリエンプティブシステムで、より優先順位の高いタスクが割り込みにより実行された場合のように、タスクが自身に割り当てられたタイムスライス全体を使用できないこともあります。

翻訳単位

ソースファイルと、プリプロセッサディレクティブ `#include` でインクルードされるすべてのヘッダファイルやソースファイル（`#if` や `#ifdef` などの条件プリプロセッサディレクティブで省略された行を除く）を合わせたもの。

トラップ

命令ストリームに特殊な命令を挿入することで実行される割り込み。多くのシステムでは、トラップを使用して、オペレーティングシステム関数を呼び出します。ソフトウェア割り込みとも呼びます。

型修飾子

標準 C/C++ では `const`、`volatile`。IAR システムズのコンパイラは、通常はメモリや他の型属性用にターゲット固有の型修飾子を追加します。

U

UBROF (Universal Binary Relocatable Object Format)

使用する製品パッケージに XLINK リンカが含まれる場合に、一部の IAR システムズのプログラミングツールにより生成されるファイルフォーマット。

V

値式、リンカ設定ファイル

C 式と同様の構文を使用する式で構成できる定数値。

ベニア

以下のときに呼出し側と呼出し先の間のスプリングボードとして挿入される小さなコード。

- モードに不一致がある（たとえば、ARM と Thumb など）。
- 呼出し命令がその宛先に達しない場合。

仮想アドレス（論理アドレス）

コンパイラ、リンカ、ランタイムシステムによって、使用前に物理メモリアドレスに変換する必要があるアドレス。仮想アドレスはアプリケーションで認識されるアドレスであり、システムの他の部分で認識されるアドレスとは異なる場合があります。

仮想空間

IAR Embedded Workbench のエディタの機能で、実際の文字のある領域外に挿入ポイントを移動できます。

揮発性ストレージ

揮発性記憶デバイスに保存したデータは、そのデバイスの電源を切った場合は保持されません。電源を切った後もデータを保持するには、不揮発性ストレージに保存する必要があります。C 言語のキーワードである `volatile` と混同しないでください。不揮発性ストレージと比較してください。

ノイマンアーキテクチャ

命令とデータの両方が共通のデータチャネルで転送されるコンピュータアーキテクチャ。ハーバードアーキテクチャと比較してください。

W

ウォッチポイント

C 言語の変数や式の値を、アプリケーション実行中に C-SPY の「ウォッチ」ウィンドウでトレースします。

X

XLINK

UBROF 出力フォーマットを使用する IAR XLINK リンカ。

Z

ゼロ初期化済みセクション

起動時にゼロに初期化されるべきセクション。セクションも参照してください。

ゼロオーバーヘッドループ

ループ条件（ループ開始地点へ戻る分岐を含む）の処理に時間がまったくかからないループ。通常はプロセッサの特別なハードウェア機能として実装されるため、利用できないアーキテクチャがあります。

ゾーン

プロセッサによって、メモリアーキテクチャは大幅に異なります。ゾーンとは、C-SPY で名前付きメモリエリアを示す用語です。たとえば、個別にアドレッシング可能なコードおよびデータメモリを持つプロセッサでは、少なくとも 2 つゾーンがあります。複雑なバンクメモリ方式を採用したプロセッサの場合は、ゾーンが複数存在する場合があります。

A

a (ファイル名の拡張子)	85
AEABI、定義	211
arm (ディレクトリ)	83
Arm (プロセッサモード設定)	172
ARM コード、Thumb コードとの混在	172
ar、定義	211
asm (ファイル名の拡張子)	85

B

bat (ファイル名の拡張子)	85
bin、arm (サブディレクトリ)	83
bin、common (サブディレクトリ)	85
board (ファイル名の拡張子)	86

C

c (ファイル名の拡張子)	86
C (言語設定)	169
cfg (ファイル名の拡張子)	86
'CHAR' の型 (コンパイラオプション)	171
chm (ファイル名の拡張子)	86
CMSIS (一般オプション)	164
CMSIS、arm (サブディレクトリ)	83
common (ディレクトリ)	85
\$CONFIG_NAME\$ (引数変数)	124
config、arm (サブディレクトリ)	83
config、common (サブディレクトリ)	85
c++ (ファイル名の拡張子)	86
CRC 多項式 (チェックサム生成の設定)	207
CRC、定義	213
CRC16 (チェックサム生成の設定)	207
CRC32 (チェックサム生成の設定)	207
cstartup (システム起動コード)	
プログラムが以下に達するまでスタックポインタが 無効	146
定義	213

\$CUR_DIR\$ (引数変数)	124
\$CUR_LINE\$ (引数変数)	124
C ソースファイル ([ワークスペース] ウィンドウアイコン)	43
C のキーワード、エディタのテキストスタイル	72
C のコメント、エディタのテキストスタイル	72
C の派生言語 (コンパイラオプション)	169
C 形式プリプロセッサ、定義	213
C-SPY オプション	
定義	213
設定	61
[オプション] ダイアログボックス内	124
C/C++ 構文	
コンパイラで有効化	169
スタイルのオプション	138
C++ 例外を許可 (リンカオプション)	202
C++ (言語設定)	169
C++ インライン動作 (C 派生言語の設定)	169
C++ ソースファイル ([ワークスペース] ウィンドウアイコン)	43
C++ のキーワード、エディタのテキストスタイル	72
C++ のコメント、エディタのテキストスタイル	72
C++ の派生言語 (コンパイラオプション)	170
C++ 仮想関数除去を実行 (リンカオプション)	201
C++ 用語	18
C89 (C 派生言語の設定)	169
C99 (C 派生言語の設定)	169

D

dat (ファイル名の拡張子)	86
\$DATE\$ (引数変数)	124
dbgt (ファイル名の拡張子)	86
ddf (ファイル名の拡張子)	86
define (リンカオプション)	204
dep (ファイル名の拡張子)	86
Diff (サブバージョン管理メニュー)	59
DLIB ライブラリ関数、リファレンス情報	71
DLIB、ドキュメント	17

dni (ファイル名の拡張子)	86
doc、arm (サブディレクトリ)	84
doc、common (サブディレクトリ)	85
drivers、arm (サブディレクトリ)	84
DSP デジタル信号プロセッサを参照。	
DWARF、定義	214
Dynamic Data Exchange (DDE)	78
外部エディタの呼び出し	136

E

EEC++ の構文 (C++ 派生言語の設定)	170
EEPROM、定義	215
ELF、変換	192
Embedded C++	
定義	215
構文、コンパイラで有効化	170
Embedded C++ Technical Committee	18
Embedded C++ (C++ 派生言語の設定)	170
Embedded Workbench	
エディタ	69
バージョン番号、表示	158
メインウィンドウ	24, 90
リファレンス情報	89
レイアウト	25
実行	24
終了	25
Enea OSE Load モジュールフォーマット、定義	215
EOL 文字 (エディタのオプション)	133
EPROM、定義	215
ewd (ファイル名の拡張子)	86
ewp (ファイル名の拡張子)	86
ewplugin (ファイル名の拡張子)	86
eww (ファイル名の拡張子)	86
ワークスペースファイル	25
\$EW_DIR\$ (引数変数)	124
examples、arm (サブディレクトリ)	84
\$EXE_DIR\$ (引数変数)	125

F

\$FILE_DIR\$ (引数変数)	125
\$FILE_FNAME\$ (引数変数)	125
\$FILE_PATH\$ (引数変数)	125
flash (ファイル名の拡張子)	86
fnt (ファイル名の拡張子)	86
FPU (一般オプション)	160

H

h (ファイル名の拡張子)	86
helpfiles (ファイル名の拡張子)	86
htm (ファイル名の拡張子)	86
html (ファイル名の拡張子)	86
HTML テキストファイル ([ワークスペース] ウィンドウアイコン)	43

I

i (ファイル名の拡張子)	86
iarbuild、コマンドラインからのビルド	66
iarchive、定義	216
IarIdePm.exe	24
icf (ファイル名の拡張子)	86
IDE	
定義	216
概要	23
IDE 内部ファイル ([ワークスペース] ウィンドウアイコン)	43
ILINK	
オプション	197
定義	217
ILINK 設定ファイルでの初期化、定義	217
inc (ファイル名の拡張子)	86
inc、arm (サブディレクトリ)	84
ini (ファイル名の拡張子)	86
iobjmanip、定義	217
i79 (ファイル名の拡張子)	86

- L**
- lib、arm (サブディレクトリ) 84
 - lightbulb アイコン、本ガイドの 19
 - #line ディレクティブ、生成
 - コンパイラ 179
 - \$LIST_DIR\$ (引数変数) 125
 - lst (ファイル名の拡張子) 87
 - L 値、定義 218
- M**
- mac (ファイル名の拡張子) 87
 - MAC、定義 218
 - metadata (サブディレクトリ) 85
 - MISRA-C
 - コンパイラオプション 180
 - ドキュメント 18
 - 一般オプション 166
- N**
- NDEBUG、プリプロセッサシンボル 32
 - NOP (アセンブラ命令)、定義 220
 - no-init セクション、定義 220
- O**
- o (ファイル名の拡張子) 87
 - objcopy、定義 220
 - objdump、定義 217
 - \$OBJ_DIR\$ (引数変数) 125
 - out (ファイル名の拡張子) 87
 - output
 - ELF からの変換 192
 - アセンブラ 185
 - デバッグ情報を含める 185
 - コンパイラ 175
 - デバッグ情報を含める 175
 - デバッグ情報を含める 203
 - プリプロセッサ 179
 - リンク、ファイル名の指定 202
- P**
- pbd (ファイル名の拡張子) 87
 - pbi (ファイル名の拡張子) 87
 - pew (ファイル名の拡張子) 87
 - Position-independence (コードオプション) 173
 - PRINTF フォーマッタ (一般オプション) 165
 - ptj (ファイル名の拡張子) 87
 - \$PROJ_DIR\$ (引数変数) 125
 - \$PROJ_FNAME\$ (引数変数) 125
 - \$PROJ_PATH\$ (引数変数) 125
 - PROM、定義 221
- R**
- readme ファイル 84
 - readme ファイル、リリースノートを参照
 - ROM モニタ、定義 222
 - ropi、位置に依存しない 173
 - RTOS、定義 221
 - rwpi、位置に依存しない 173
 - R 値、定義 223
- S**
- s (ファイル名の拡張子) 87
 - SCANF のフォーマッタ (一般オプション) 165
 - SCC。ソースコード管理システムを参照してください。
 - SFR
 - ヘッダファイル 84
 - 定義 224
 - src、arm (サブディレクトリ) 84
 - STL コンテナ拡張 (IDE オプション) 145
 - svd (ファイル名の拡張子) 87

T

\$TARGET_BNAME\$ (引数変数)	125
\$TARGET_BPATH\$ (引数変数)	125
\$TARGET_DIR\$ (引数変数)	125
\$TARGET_FNAME\$ (引数変数)	125
\$TARGET_PATH\$ (引数変数)	125
Thumb (プロセッサモード設定)	173
Thumb コード、ARM コードとの混在	173
\$TOOLKIT_DIR\$ (引数変数)	125
touch、オープンソースコマンドラインユーティリティ	65
tutor、arm (サブディレクトリ)	84

U

UBROF、定義	225
\$USER_NAME\$ (引数変数)	125

V

VFPv2 (FPU 設定)	160
VFPv3 (FPU 設定)	160
VFPv3 + NEON (FPU 設定)	160
VFPv4 (FPU 設定)	160
VFPv4 + NEON (FPU 設定)	161
VFP9-S (FPU 設定)	161
visualSTATE	
ツールチェーンの一部	23
プロジェクトファイル	87
VLA の許可 (C 派生言語の設定)	169
vsp (ファイル名の拡張子)	87

W

Web サイト、推奨	18
wsdt (ファイル名の拡張子)	87

X

xcl (ファイル名の拡張子)	87
XLINK、定義	226

あ

アイコン	
SVN の状態	60
[ワークスペース] ウィンドウ	43
アクティブに設定 ([ワークスペース] ウィンドウのコンテキストメニュー)	46
アサーション、ビルドアプリケーション	32
アセンブラオプション	183
定義	211
プリプロセッサ	187
リスト	185
出力	185
言語	183
診断	188
アセンブラシンボル、定義	188
アセンブラソースファイル ([ワークスペース] ウィンドウアイコン)	43
アセンブラディレクティブ	
エディタのテキストスタイル	72
定義	211
アセンブラニーマニックス (出力リストファイルの設定)	177
アセンブラのコメント、エディタのテキストスタイル	72
アセンブラの出力、デバッグ情報を含める	185
アセンブラプリプロセッサ	187
アセンブラリストファイル	
クロスリファレンス、生成	186
コンパイラの呼出しフレーム情報	
タブによる移動量、指定	187
ヘッダ、含む	186
ページあたりの行数、指定	187

条件付き情報、指定	186
生成	185
アセンブラ言語、定義	211
アセンブラ、コマンドラインバージョン	23
アセンブルされた行のみ（インクルードリスト化 の設定）	186
アドレス式、定義	211
アプリケーション	
テスト	65
定義	211
アプリケーションで定義（デフォルトプログラムエン トリのオーバーライド設定）	199
アラインメント（チェックサム生成の設定）	207
アラインメント（未処理バイナリイメージの設定） ..	200
アンドゥ（ボタン）	91
アーカイブ、定義	211
アーキテクチャ、定義	211

い

インクルードファイル	84
アセンブラ、パスを指定	187
コンパイラ、パスを指定	177-178
定義	217
インストールされるファイル	83
インクルード	84
ドキュメント	84
ライブラリ	84
実行可能	85
インストールパス、デフォルト	83
インストール先ディレクトリ	18
インデントサイズ（エディタのオプション）	132
インデント、エディタ	72
インラインアセンブラ、定義	217
インライン化、定義	217

う

ウインドウ	89
画面上の編成	25
ウォッチポイント、定義	226
エイリアス（[キーカスタマイズ] オプション） ...	131
エディタ	
インデント	72
オプション	132
キーボードコマンド	98
コマンド	71
コードテンプレート	74
ステータスバー、使用	74
使用	69
分割バー	94
外部	78
括弧と中括弧の対応	73
環境のカスタマイズ	78
関数のショートカット	77, 94
エディタ（[外部エディタ] オプション）	136
エディタウインドウ	92
エディタウインドウでの検索	77
エディタセットアップファイル（[IDE オプション] ダイアログボックス）	137
エディタセットアップファイル、オプション	137
エディタフォント（[エディタ色とフォント] オプション）	138
エディタ色とフォント（[IDE オプション] ダイアログボックス）	138
エミュレータ（C-SPY ドライバ）、定義	215
エラーとして処理（コンパイラオプション）	180
エラーとして処理（リンカオプション）	206
エラーメッセージ	
コンパイラ	180
リンカ	206
エンディアンモード（一般オプション）	160
エントリシンボル（デフォルトプログラムエントリの オーバーライド設定）	199

お

オブジェクトファイル（出力ディレクトリの設定） ..	162
オブジェクトファイルまたはライブラリ （[ワークスペース] ウィンドウアイコン）	43
オブジェクトファイル（再配置可能）、定義	220
オブジェクトファイル（絶対）、定義	220
オブジェクト、定義	220
オプション	
アセンブラ	183
エディタ	132
エディタセットアップファイル	137
カスタムビルド	193
コンパイラ	167
コンバータ	191
ビルドアクション	195
ライブラリビルダ	209
リンカ	197
一般	159
オプション（[ワークスペース] ウィンドウの コンテキストメニュー）	45
オプション名（カテゴリオプション）	49
オプション、定義	220
オンラインドキュメント	
ターゲット固有、ディレクトリ	84
[ヘルプ] メニューから使用可能	158
オーバーレイ、定義	220

か

ガイドラインの確認	15
カスタムキーワードファイルの使用（エディタの オプション）	138
カスタムツール構成（カスタムビルドオプション） ..	193
カスタムビルド	66
使用	67
カスタムビルド構成	66
カテゴリ、[オプション] ダイアログボックス	63, 123

き

キーバインディング、定義	218
キーバインディング（[IDE オプション] ダイアログ ボックス）	130
キーワード	
エディタでの構文カラーの指定	72
定義	218
言語拡張の有効化	170
キー操作のまとめ、エディタ	98

く

グラフィカルスタック表示とスタック使用トラッキン グを有効にする（[スタック] オプション）	146
-clean（iarbuild コマンドラインオプション）	66
クリーン（[ワークスペース] ウィンドウの コンテキストメニュー）	45
グループ（[レジスタフィルタ] オプション）	148
グループの追加（[ワークスペース] ウィンドウの コンテキストメニュー）	46
グループメンバ（[レジスタフィルタ] オプション） ..	148
グループ、定義	33
クロスリファレンス（アセンブラオプション）	186

こ

コア（派生プロセッサの設定）	160
コスト。メモリアクセスコストを参照。	
このガイドで使用されている規則	18
コピー（ボタン）	91
コマンド（[外部エディタ] オプション）	137
コマンドプロンプトアイコン、本ガイド	19
コマンドライン（カスタムビルドオプション）	194
コマンドラインオプション	
表記規則	19
[ツール] メニューから指定	28
コマンドラインオプションの使用（アセンブラの オプション）	189

コマンドラインオプションの使用 (コンパイラのオプション)	181
コマンドラインオプションの使用 (リンカのオプション)	208
コミット (サブバージョン管理メニュー)	59
コメントの保持 (ファイルへのプリプロセッサ 出力の設定)	179
コンパイラオプション	167
定義	213
MISRA-C	180
コード	172
コードおよびリードオンリーのデータ	173
プリプロセッサ	177
リスト	176
リード/ライトデータ	173
出力	175
最適化	173
言語 1	168
言語 2	171
診断	179
コンパイラのシンボル、定義	178
コンパイラのプリプロセッサ	177
コンパイラのリストファイル	
アセンブラニモニック、含める	177
ソースコード、含める	177
生成	176
コンパイラの呼出しフレーム情報のインクルード (アセンブラ出力ファイル設定)	177
コンパイラの診断	177
無効化	179
コンパイラ出力、デバッグ情報を含める	175
コンパイラ、コマンドラインバージョン	23
コンパイル ([ワークスペース] ウィンドウの コンテキストメニュー)	45
コンバータオプション	191
コンピュータスタイル、表記規則	19
コード	
スケルトン、定義	224
テスト	65
バンク、定義	211

コードおよびリードオンリーのデータ (コンパイラオプション)	173
コードセクション名 (コンパイラオプション)	175
コードセクション、定義	213
コードテンプレートの使用 (エディタのオプション)	138
コードテンプレート、エディタで使用	74
コードのテンプレート、使用	74
コードの整合性	34
コードの相互作用、生成	172
コードボインタ、定義	213
コードメモリ、使用部分のフィル	207
コードモデル、定義	213

さ

サイズ (チェックサム生成の設定)	207
サイズの最適化	174
サブバージョンの状態と対応するアイコン	60
サービス ([外部エディタ] のオプション)	137

し

シグナル、定義	224
シミュレータ、定義	224
ショートアドレッシング、定義	223
ショートカットキー	71
ショートカットキーを押してください ([キーバイン ディング] オプション)	130
シンボル	
ユーザシンボルも参照	
アセンブラでの定義	188
コンパイラでの定義	178
リンカでの定義	204
定義	224
シンボル (未処理バイナリイメージの設定)	200
シンボルをキープ (リンカオプション)	200
シンボル位置、定義	224
シンボル定義 (アセンブラオプション)	188

シンボル定義 (コンパイラオプション)	178
シンボル定義 (リンカオプション)	204

す

スクロール、～のためのショートカットキー.....	71
スケジューラ (RTOS)、定義	223
スケルトンコード、定義	224
スコープ、定義.....	223
スタック ([IDE オプション] ダイアログボックス) ..	145
スタックしきい値の超過時にワーニング ([スタック] オプション)	146
スタックセグメント、定義	224
スタックフレーム、定義	224
スタックポインタが境界外の時にワーニング ([スタック] オプション)	146
ステップ実行、定義	224
ステータスバー	91
すべてのワーニング (ワーニング設定)	189
すべてのワーニングをエラーとして処理 (コンパイラオプション)	180
すべてのワーニングをエラーとして処理 (リンカオプション)	206
すべてリセット ([キーバインディング] オプション)	131
すべてをビルド ([ワークスペース] ウィンドウの コンテキストメニュー)	45
すべてを保存 (ボタン)	91
すべてを保存 ([ファイル] メニュー)	105
スペースによるインデント (タブキーの機能設定) ..	132
スレッド、定義	225

せ

セクション	
バイナリデータ	200
定義	223
セクション (未処理バイナリイメージの設定).....	200
セクションの属性、定義	223

セクションフラグメント、定義	223
セクション選択、定義	223
セマフォ、定義	223
ゼロオーバーヘッドループ、定義	226
ゼロ初期化されたセクション、定義	226
ゼロ、定義	226

そ

その他のファイル ([ワークスペース] ウィンドウアイコン)	43
ソースコード	
コンパイラリストファイルに含める	177
テンプレート	74
ソースコード管理 ([IDE オプション] ダイアログボックス)	143
ソースコード管理システム	34
ソースのインクルード (アセンブラ出力ファイル 設定)	177
ソースファイル	
パス	33, 93
プロジェクト管理	33
編集	69
ソースファイルの編集	69

た

タイプ ([外部エディタ] のオプション)	136
タイマ、定義	225
タイムスライス、定義	225
タスク、定義	225
タブキーの機能 (エディタのオプション)	132
タブサイズ (エディタのオプション)	132
ダブルスペースによる改行 (インクルードクロスリ ファレンスの設定)	186
タブを挿入 (タブキーの機能設定)	132
タブ間隔 (アセンブラオプション)	187
ターゲット、定義	225
ターゲット (一般オプション)	159

ターミナル I/O ([IDE オプション] ダイアログボックス).....	149
ターミナル I/O ウィンドウ、定義.....	225

ち

チェックアウト (ソースコード管理メニュー).....	54
チェックアウトを元に戻す (ソースコード管理メ ニュー)	54
チェックイン (ソースコード管理メニュー)	54
チェックサム 定義.....	213
チェックサム (リンカオプション)	206
チェックサム生成 (リンカオプション)	207
チェックサム、生成.....	207
チェックマーク (ソースコード管理アイコン).....	58

つ

ツールアイコン、本ガイド.....	19
ツールチェーン 拡張.....	66
概要.....	23
ツールの設定 ([ツール] メニュー)	150
ツールバー、IDE	91
ツール、ユーザ設定.....	150

て

ディレクトリ arm.....	83
common.....	85
アセンブラ、標準のインクルードを無視.....	187
コンパイラ、標準のインクルードを無視.....	177
ルート.....	83
ディレクトリ構成.....	83
テキストの選択、～のためのショートカットキー...	71
テキストファイル ([ワークスペース] ウィンドウアイコン).....	43

デジタル信号プロセッサ、定義.....	214
テスト、コード	65
デバイス (派生プロセッサの設定)	160
デバイスドライバ、定義.....	214
デバイス記述ファイル.....	83
定義.....	214
デバイス選択ファイル.....	83
デバグ ([IDE オプション] ダイアログボックス) ..	144
デバグ (構成の工場出荷時設定)	50
デバグ前にメイクを実行 ([IDE プロジェクト] オプション)	142
デバグ情報 アセンブラで生成.....	185
コンパイラ、生成.....	175
デバグ情報の生成 (アセンブラオプション)	185
デバグ情報の生成 (コンパイラオプション)	175
デフォルトのインストールパス.....	83
デフォルトの出荷時設定の復元.....	63
デフォルトプログラムエントリのオーバーライド (リンカオプション).....	199
デフォルト整数フォーマット (IDE オプション)...	145
デマングル、定義	214
データポインタ、定義.....	214
データモデル、定義.....	213
データ表現、定義	214

と

ドキュメント	83
オンライン.....	84
ガイドの概要.....	17
本ガイド.....	15
本ガイドの概要	16
ドッキング可能なウィンドウ.....	25
ドラッグアンドドロップ エディタウィンドウのテキスト	71
[ワークスペース] ウィンドウのファイル.....	34
トラップ、定義	225

な

なし (レベル設定)..... 174

ね

ネスト割込み、定義..... 219

の

ノイマンアーキテクチャ、定義..... 226

は

バイトオーダー、設定..... 160

パイプライン、定義..... 220

パス

アセンブラのインクルードファイル..... 187

コンパイラのインクルードファイル..... 177-178

ソースファイル..... 93

相対、Embedded Workbench..... 33, 93

バックトレース情報、定義..... 211

バッチビルド..... 64

バッチファイル

定義..... 212

[ツール] メニューから指定..... 28

バッファした書込み (リンカオプション)..... 166

パラメータ

コマンドラインからのビルド時..... 66

表記規則..... 19

バンクコード、定義..... 211

バンクデータ、定義..... 212

バンクメモリ、定義..... 212

バンク切替ルーチン、定義..... 212

バンク切替え、定義..... 211

バージョン番号

Embedded Workbench..... 158

本ガイド..... 2

バージョン管理システム..... 34

バージョン管理システム ([ワークスペース]

ウィンドウ/コンテキストメニュー)..... 46

バージョン管理システムメニュー..... 53, 58

ハーバードアーキテクチャ、定義..... 216

ひ

ビッグ (エンディアンモードの設定)..... 160

ビットフィールド、定義..... 212

ビット順 (チェックサム生成の設定)..... 207

ビューア拡張子の編集 ([ツール] メニュー)..... 156

ビルド

オプション..... 141

コマンド..... 63

コマンドラインから..... 66

処理..... 61

前後のアクション..... 64

-build (iarbuild コマンドラインオプション)..... 66

ビルドアクション..... 64

ビルドアクションの構成 (ビルドアクションオプション)..... 195

ビルドウィンドウ ([表示] メニュー)..... 99

ビルドから除外されたグループ ([ワークスペース]

ウィンドウアイコン)..... 43

ビルドから除外されたソースファイル

([ワークスペース] ウィンドウアイコン)..... 43

ビルドを停止 ([ワークスペース] ウィンドウの
コンテキストメニュー)..... 45

ビルド処理終了後にサウンドを再生

([IDE プロジェクト] オプション)..... 142

ビルド前にエディタウィンドウを保存

([IDE プロジェクト] オプション)..... 141

ビルド前にワークスペースとプロジェクトを保存

([IDE プロジェクト] オプション)..... 142

ビルド後コマンドライン (ビルドアクション

オプション)..... 196

ビルド構成

作成..... 36

定義..... 32

ヒープサイズ、定義.....	216
ヒープメモリ、定義.....	216

ふ

ファイル	
編集.....	69
間のナビゲート.....	30
ファイル (未処理バイナリイメージの設定).....	200
ファイルグループ ([ワークスペース]	
ウィンドウアイコン).....	43
ファイルタイプ	
readme.....	84
インクルード.....	84
コマンドライン拡張.....	87
デバイスの選択.....	83
デバイス記述.....	83
ドキュメント.....	84
ドライバ.....	84
フラッシュローダアプリケーション.....	83
プロジェクトテンプレート.....	83
ヘッダ.....	84
ライブラリ.....	84
リンカ設定ファイル.....	83
構文カラー表示設定.....	83
ファイルのチェックアウトダイアログボックス.....	57
ファイルのチェックインダイアログボックス.....	56
ファイルの追加 ([ワークスペース] ウィンドウの	
コンテキストメニュー).....	46
ファイルプロパティ ([ワークスペース]	
ウィンドウのコンテキストメニュー).....	46
ファイルへのプリプロセッサ出力	
(コンパイラオプション).....	179
ファイル名の拡張子.....	85
cfg、構文強調表示.....	138
eww、ワークスペースファイル.....	25
デフォルト以外.....	88
ファイル名の拡張子 (カスタムビルドオプション)...	194
ファイル拡張子。ファイル名の拡張子を参照	

フィルタファイル ([レジスタフィルタ]	
オプション).....	148
フィルパターン (フィルの設定).....	207
フィル、定義.....	216
フォント	
エディタ.....	138
プロポーショナル.....	129
固定幅.....	129
フォーマット指定子、定義.....	216
ブックマーク	
エディタで表示.....	134
追加.....	77
ブックマークの切替え (ボタン).....	91
ブックマークの表示 (エディタのオプション)....	134
ブックマークへ移動 (ボタン).....	91
プライマリ ([キーバインディング] オプション)...	130
ブラウズ情報を生成 ([IDE プロジェクト]	
オプション).....	142
プラグイン	
arm (サブディレクトリ).....	84
common (サブディレクトリ).....	85
#pragma ディレクティブ、定義.....	221
フラッシュローダアプリケーション.....	83
ブランク (ソースコード管理アイコン).....	58
プリインクルードファイル	
(コンパイラオプション).....	178
プリエンプティブマルチタスク、定義.....	221
プリビルドコマンドライン (ビルドアクションオブ	
ション).....	196, 210
プリプロセッサ	
NDEBUG シンボル.....	32
定義。C 形式プリプロセッサを参照	
文字列変数を初期化するマクロ.....	64
プリプロセッサ (コンパイラオプション).....	177
プリプロセッサディレクティブ	
エディタのテキストスタイル.....	72
定義.....	221
プリプロセッサ (アセンブラオプション).....	187
フルサイズでの結果 (チェックサム生成の設定) ..	207
ブレークポイントの切替え (ボタン).....	91

ブレークポイント、定義	212
プログラミング経験	15
プログラムカウンタ、定義	221
プログラムロケーションカウンタ、定義	221
プログラム可能な出力フォーマット (コンバータオプション)	192
プロジェクト	
オプションの設定	61
グループとファイルの排除	36
グループ、作成	35
ソースコード管理	34
テスト	65
バージョン管理システム	34
ビルド	63
バッチで	64
ビルド構成、作成	36
ファイルの追加	35, 118
ファイル、移動	36
ワークスペース、作成	35
作成	35
定義	31, 221
管理	29
編成	30
項目の削除	36
プロジェクト IDE ([IDE プロジェクト] オプション)	141
プロジェクト ([ワークスペース] ウィンドウアイコン)	43
プロジェクトオプション、定義	221
プロジェクトメイク、オプション	141
プロジェクトモデル	29
プロジェクトを SCC プロジェクトから切断 (ソースコード管理メニュー)	55
プロジェクトを SCC プロジェクトに接続 (ソースコード管理メニュー)	55
プロジェクトを SVN プロジェクトから切断 (サブバージョン管理メニュー)	59
プロジェクトを SVN プロジェクトに接続 (サブバージョン管理メニュー)	59

プロセッサのモード (コードオプション)	172
プロセッサのモード (コードオプション)	172
ブロック、定義	212
プロトタイプの強制 (C 派生言語の設定)	169
プロトタイプ、存在の検証	169
プロパティ (サブバージョン管理メニュー)	59
プロパティ (ソースコード管理メニュー)	54
プロポーショナルフォント (IDE オプション)	129
フローティングウィンドウ	25

へ

ヘッダファイル	84
迅速なアクセス	76
ヘッダファイル ([ワークスペース] ウィンドウアイコン)	43
ヘッダを含む (アセンブラオプション)	186

ほ

ポインタ	
スタックポインタが範囲外の時にワーニング ...	146
定義	221
ホスト、定義	216

ま

マイクロコントローラ、定義	219
マイクロプロセッサ、定義	219
マクロテキスト (インクルードリスト化の設定) ..	186
マクロの引用符 (アセンブラオプション)	184
マクロ実行情報 (インクルードリスト化の設定) ..	186
マクロ拡張子 (インクルードリスト化の設定)	186
マクロ、定義	218
マップファイル、リンカからの生成	203
マルチタスク、定義	221
マルチバイト文字サポートを有効にする (アセンブラオプション)	184

マルチバイト文字サポートを有効にする (コンパイラオプション)	172
マングル化、定義	219

め

メッセージ ([IDE オプション] ダイアログボックス)	139
メニュー	103
メニュー (ファイル名の拡張子)	87
メニューバー	90
メニュー ([キーバインディング] オプション)	130
メモリ	
定義	219
未使用部分のフィル	207
メモリアクセスコスト、定義	219
メモリエリア、定義	219
メモリバンク、定義	219
メモリマップ、定義	219
メモリモデル、定義	219
メールボックス (RTOS)、定義	219

も

モジュール、定義	219
----------------	-----

ゆ

ユーザシンボルの大文字 / 小文字を区別する (アセンブラオプション)	184
--	-----

ら

ライブラリ (出力ファイル設定)	161
ライブラリオプション (一般オプション)	165
ライブラリビルダオプション、出荷時設定	210
ライブラリビルダ、出力オプション	210
ライブラリファイル	84

ライブラリ低レベルインタフェースの実装 (一般オプション)	163
ライブラリ構成 (一般オプション)	162
ライブラリ設定ファイル	
IDE からの指定	163
定義	218
ライブラリ関数	
リファレンス情報	71
設定可能	84
ライブラリ、定義	222
ライブラリ (リンカオプション)	198
ライブラリ (一般オプション)	163
ラウンドロビン、定義	222
ラベル (c) ([自動インデントの設定] オプション)	135
ランタイムモデル属性、定義	222
ランタイムライブラリ	
定義	222
指定	163

り

リアルタイムオペレーティングシステム、定義 ...	221
リアルタイムシステム、定義	221
リスト (リンカオプション)	203
リストファイル	
アセンブラ	
クロスリファレンス、生成	186
コンパイラのランタイム情報	
タブによる移動量、指定	187
ヘッダ、含む	186
ページあたりの行数、指定	187
条件付き情報、指定	186
コンパイラ	
アセンブラニーモニック、含める	177
ソースコード、含める	177
生成	176
リストファイル (出力ディレクトリの設定)	162
リストファイルの出力 (アセンブラオプション) ..	186
リストファイルの出力 (コンパイラオプション) ..	176

リストを含む (アセンブラオプション)	186
リスト (アセンブラオプション)	185
リスト (コンパイラオプション)	176
リセット、定義.....	222
リドウ (ボタン).....	91
リトルエンディアン (エンディアンモードの設定) ...	160
リマーク	
コンパイラの診断	180
リンカ診断	205
リマークとして処理 (コンパイラオプション).....	180
リマークとして処理 (リンカオプション)	205
リマークを有効化 (コンパイラオプション)	179
リマークを有効化 (リンカオプション)	205
リリース (構成の工場出荷時設定)	50
リレー、定義.....	222
リンカ	
コマンドラインバージョン.....	23
リンカオプション.....	197
チェックサム	206
ライブラリ	198
リスト	203
入力	199
出力	202
最適化.....	201
設定	197
診断	204
追加オプション	208
#define	204
リンカコマンドファイル。リンカ設定ファイルを参照	
リンカシンボル、定義.....	204
リンカの設定ファイル、定義.....	217
リンカマップファイルの表示 (リンカオプション) ...	203
リンカ設定ファイル	
ディレクトリ	83
リンカで指定	198
定義.....	218
リンカ設定ファイル (リンカオプション)	198
リンカ診断、無効化.....	205

リードオンリーセクション、定義.....	221
リード/ライトデータ (コンパイラオプション)...	173

る

ルートディレクトリ.....	83
----------------	----

れ

レイアウト、Embedded Workbench の	25
レジスタ	
inc ディレクトリのヘッダファイル.....	84
定義.....	222
レジスタフィルタ ([IDE オプション]	
ダイアログボックス).....	147
レジスタフィルタの使用 ([レジスタフィルタ]	
オプション)	148
レジスタロック、定義.....	222
レジスタ変数、定義.....	222
レジスタ定数、定義.....	222
レベル (コンパイラオプション).....	174

ろ

-log (iarbuild コマンドラインオプション)	66
ログ (サブバージョン管理メニュー)	59
ログ (ファイル名の拡張子).....	86
ログファイルの生成 (リンカオプション)	203
ログファイル、リンカからの生成.....	203
ロケーションカウンタ、定義.....	221

わ

ワークスペース	
作成.....	35
使用.....	35
ワークスペース ([ワークスペース]	
ウィンドウアイコン).....	43
ワークスペースを閉じる ([ファイル] メニュー)...	105

ワークスペースを開く ([ファイル] メニュー)...	105
ワーニング	
コンパイラ	180
リンカ	206
ワーニング (アセンブラオプション)	188
ワーニングとして処理 (コンパイラオプション)...	180
ワーニングとして処理 (リンカオプション)	206
ワーニングのみ (ワーニング設定)	189
ワーニング範囲指定 (ワーニング設定)	189

記号

[IDE オプション] ダイアログボックス.....	132
[インクリメンタル検索] ダイアログボックス ([編集] メニュー).....	114
[ウィンドウ] メニュー.....	157
[オプション] ダイアログボックス ([プロジェクト] メニュー)	123
使用	62
[クイック検索] テキストボックス	91
[グループの名称変更] ダイアログボックス	46
[コード] ページ (コンパイラオプション)	172
[ソースコード管理プロバイダの選択] ダイアログボッ クス ([プロジェクト] メニュー)	55
[ソースブラウザ] ウィンドウ	50
使用	38
[ダウンロードなしにデバッグ] テキストボックス ...	91
[ツール出力] ウィンドウ	101
[ツール] メニュー.....	128
[デバッグログ] ウィンドウ ([表示] メニュー)...	102
[テンプレート] ダイアログボックス ([編集] メニュー)	115
[パッチビルドの編集] ダイアログボックス ([プロジェクト] メニュー)	127
[パッチビルド] ダイアログボックス ([プロジェクト] メニュー)	126
[ビューアの設定] ダイアログボックス ([ツール] メニュー)	155
[ファイルで検索] ウィンドウ ([表示] メニュー) ...	100

[ファイルで検索] ダイアログボックス ([編集] メニュー)	112
[ファイル名の拡張子] ダイアログボックス ([ツール] メニュー)	152
[ファイル名拡張子のオーバーライド] ダイアログボッ クス ([ツール] メニュー).....	153
[ファイル名拡張子の編集] ダイアログボックス ([ツール] メニュー).....	154
[ファイル] メニュー.....	103
[プロジェクトの構成] ダイアログボックス ([プロジェクト] メニュー).....	48
[プロジェクト] ページ ([IDE オプション] ダイアログボックス).....	141
[プロジェクト] メニュー.....	118
[ヘルプ] メニュー	158
[メッセージ] ウィンドウ、出力内容	139
[メモリ消去] ダイアログボックス	122
[ワークスペース] ウィンドウ	42
ファイルのドラッグアンドドロップ.....	34
[ワークスペース] ウィンドウアイコン	43
[新規プロジェクトの作成] ダイアログボックス ([プロジェクト] メニュー).....	47
[新規構成] ダイアログボックス ([プロジェクト] メニュー)	49
[最適化] ページ (コンパイラオプション)	173
[検索] ダイアログボックス ([編集] メニュー) ..	110
[編集] メニュー	106
[置換] ダイアログボックス ([編集] メニュー) ..	111
[行へ移動] ダイアログボックス	108
[表示] メニュー	116
[言語] ([IDE オプション] ダイアログボックス) ...	131
[逆アセンブリ] ウィンドウ、定義	214
[[逆アセンブリ] ウィンドウのソースコード色] ウィンドウ (IDE オプション)	144
#define オプション (リンカオプション).....	204
#define 文、コンパイラ	178
#define (インクルードクロスリファレンスの設定) ...	186
#define オプション (リンカオプション).....	204
#define 文、コンパイラ	178
#included テキスト (インクルードリスト化の設定) ...	186

#line ディレクティブを生成 (ファイルへの プリプロセッサ出力の設定)	179
#pragma ディレクティブ、定義.....	221
% スタック使用しきい値 ([スタック] オプション) ...	146
\$CONFIG_NAME\$ (引数変数)	124
\$CUR_DIR\$ (引数変数)	124
\$CUR_LINE\$ (引数変数)	124
\$DATE\$ (引数変数)	124
\$EW_DIR\$ (引数変数)	124
\$EXE_DIR\$ (引数変数)	125
\$FILE_DIR\$ (引数変数)	125
\$FILE_FNAME\$ (引数変数)	125
\$FILE_PATH\$ (引数変数)	125
\$LIST_DIR\$ (引数変数)	125
\$OBJ_DIR\$ (引数変数)	125
\$PROJ_DIR\$ (引数変数)	125
\$PROJ_FNAME\$ (引数変数)	125
\$PROJ_PATH\$ (引数変数)	125
\$TARGET_BNAME\$ (引数変数)	125
\$TARGET_BPATH\$ (引数変数)	125
\$TARGET_DIR\$ (引数変数)	125
\$TARGET_FNAME\$ (引数変数)	125
\$TARGET_PATH\$ (引数変数)	125
\$TOOLKIT_DIR\$ (引数変数)	125
\$USER_NAME\$ (引数変数)	125