

CodeWarrior®

Targeting Dreamcast



CodeWarrior は出荷直前でも改良されることがあるので、このマニュアルに記載されている内容の一部が本物のソフトウェアの動きと異なることがあるかもしれません。最新の情報については CodeWarrior の「Release Notes」フォルダをご覧ください。

Revised: 990715 rw-JP000113

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

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

Metrowerks の名称、ロゴ、CodeWarrior、Software at Work は、Metrowerks Inc. の登録商標です。

PowerPlant、PowerPlant Constructor は、Metrowerks Inc. の商標です。

SEGA,Dreamcast は（株）セガ・エンタープライゼスの商標です。

記載の商標および登録商標は、各社が保有します。

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

連絡先：

日本	メトロワークス株式会社 150-0042 東京都渋谷区宇田川町 36-6 ワールド宇田川ビル 8F TEL : (03) 3780-6091 FAX : (03) 3780-6092
U.S.A.	Metrowerks Corporation 9801 Metric Boulevard, Suite 100 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
WWW サーバ	http://www.metrowerks.co.jp/ http://www.metrowerks.com/

目次

第 1 章 紹介	7
リリースノートについて	7
新機能	7
オーバーレイのサポート	7
関数の並べ替えをサポート	8
リンカコマンドファイルの自動生成	8
コマンドラインツール	8
CodeWarrior のマニュアルについて	8
このマニュアルについて	9
さらに知識を深めるために	10
Metrowerks の西暦 2000 年問題対応	10
マニュアルの表記規則	10
表記について	10

第 2 章 はじめに	13
システムの必要条件	13
CodeWarrior for Dreamcast をインストール	13
CodeWarrior for Dreamcast ツールをインストール	14
Dreamcast ランタイムライブラリをインストール	14
Dreamcast 用開発システムの動作確認	14

第 3 章 Dreamcast 用ツール	17
Dreamcast 用ツールの紹介	17
CodeWarrior IDE	17
Dreamcast 用 CodeWarrior コンパイラ	18
Dreamcast 用 CodeWarrior アセンブラ	18
Dreamcast 用 CodeWarrior リンカ	18
Dreamcast 用 CodeWarrior デバッガ	18
Dreamcast 用 Codescape デバッガ	18
CodeWarrior での開発過程	19

第 4 章 アプリケーションの作成	21
アプリケーションを作成する	21

第 5 章 静的ライブラリの作成	29
静的ライブラリについて	29
静的ライブラリを作成する	29

第 6 章 SHC/C++ プロジェクトを変換	31
SHC/C++ プロジェクトを変換する	31
第 7 章 CodeWarrior でのデバッグ	37
CodeWarrior デバッガを使う	37
mw_pr() を使う	38
静的ライブラリをデバッグ	38
第 8 章 Codescape でのデバッグ	39
Codescape デバッガを使う	39
debug-printf() を使う	41
第 9 章 ターゲット設定	43
ターゲット設定の概要	43
Dreamcast 用の設定パネル	44
Target Settings	45
SH Target	47
SH Assembler	48
SH Processor	50
Global Optimizations	50
BatchRunner PostLinker	52
LCF Prelinker	53
Section Mappings	54
SH Linker	54
Debugger Settings	56
第 10 章 Dreamcast の C/C++	57
Dreamcast の数値フォーマット	58
Dreamcast の整数フォーマット	58
Dreamcast の浮動小数点フォーマット	58
Dreamcast の呼び出し規約	59
Dreamcast コードの最適化	59
Dreamcast のプラグマ	62
Dreamcast の C/C++ の注意	62
例外処理	62
ストリームと IO クラス	63
その他の制限	63

第 11 章 Dreamcast 用リンカについて	65
未使用コードの削除 (デッドストリップ)	65
リンク順	65
リンカコマンドファイル	66
リンカコマンドファイルを作成	66
リンカコマンドファイルのフレームワーク	68
リンカコマンドファイルの文法	69
関数順の変更	75
第 12 章 インラインアセンブラと組込み関数	79
インラインアセンブラを使う	79
インラインアセンブラの文法	79
ラベルの使用	81
コメントの使用	81
レジスタの使用	81
アセンブラ疑似命令	82
組込み関数	84
組込み関数のリスト	84
Hitachi SH C コンパイラ互換の組込み関数	86
インラインアセンブラのニーモニック	89
インラインアセンブラの特殊インストラクション	89
インラインアセンブラニーモニックのリスト	91
第 13 章 オーバーレイ	99
オーバーレイプロジェクトを作成	99
オーバーレイの注意点	105
オーバーレイと例外	105
オーバーレイヘッダ	105
GDWorkshop	106
第 14 章 ライブラリとランタイムコード	107
Metrowerks のユーティリティライブラリ	107
MWBlod()	107
MWNotifyOverlayLoaded()	108
MWInitOverlay()	108
MWLoadOverlay()	108
ランタイムライブラリ	109
メモリとヒープの割り当て	109

第 15 章 コマンドラインツール	111
コマンドラインツールと IDE の違い	111
オーバーレイのサポート	111
リンカコマンドファイルの生成	111
コマンドラインツールの場所	111
コマンドラインのスイッチ	112
mwasmshx : アセンブラのスイッチ	112
mwccshx : コンパイラのスイッチ	112
環境変数の設定	112
C/C++ コンパイラ変数	113
リンカ変数	113
コンパイルとリンク	113

第 16 章 トラブルシューティング	115
ハードウェアとの通信	115
コンパイラの問題	115
デバッガの問題	116

索引	117
----	-----

第 1 章 紹介

このマニュアルでは CodeWarrior を使って Dreamcast プラットフォーム用のコードを開発する方法を説明します。スタンドアロンアプリケーションや静的ライブラリの作成方法についても説明します。

Dreamcast 用のプロジェクト特有のオプション設定やランタイムライブラリについても解説します。

以下の内容について説明します。

[リリースノートについて](#)

[新機能](#)

[CodeWarrior のマニュアルについて](#)

[このマニュアルについて](#)

[さらに知識を深めるために](#)

[Metrowerks の西暦 2000 年問題対応](#)

[マニュアルの表記規則](#)

リリースノートについて

CodeWarrior IDE やツールを使う前に、リリースノートを必ずお読みください。マニュアルには書かれていないバグの修正、互換性などの最新情報が含まれています。

CodeWarrior を標準インストールするとリリースノートフォルダもインストールされます。リリースノートフォルダは CodeWarrior CD のトップレベルにあります。

新機能

Dreamcast 用 CodeWarrior 開発環境の新機能を紹介します。

オーバーレイのサポート

オーバーレイについて、またオーバーレイを使って通常のプログラムで利用可能な RAM よりも大きな Dreamcast アプリケーションを作成、実行する方法を説明する章があります。

関数の並べ替えをサポート

Codescape Profiler から CodeWarrior へ出力を渡すことによって、インストラクションキャッシュヒットの頻度と実行速度を向上するようにアプリケーション内の関数の順番を変更することができます。

リンカコマンドファイルの自動生成

新しいプリリンカ、SH LCF Generator はプロジェクトを検査して、プロジェクトにぴったりのリンカコマンドファイルを自動的に生成します。

コマンドラインツール

今バージョンにはコマンドラインのコンパイラ、リンカ、アセンブラが含まれています。コマンドラインスクリプトやメイクファイルを利用できます。

CodeWarrior のマニュアルについて

CodeWarrior はマルチホスト、マルチ言語、マルチターゲットの開発環境です。この意味について説明します。

マルチホスト :CodeWarrior は Windows や Mac OS などの複数のオペレーティングシステム上で動作します。CodeWarrior の機能、ヒューマンインターフェース、操作方法はすべてのホストにおいてほぼ同じです。

マルチ言語 :CodeWarrior を使って複数の言語 (C/C++, Pascal, Java など) でプログラムができます。ターゲットによっては、サードパーティのコンパイラがサポートするその他の言語 (Fortran など) も利用することができます。

マルチターゲット :CodeWarrior を使って、さまざまなチップ、オペレーティングシステム用のソフトウェアを開発することができます。CodeWarrior は組み込みプロセッサ、リアルタイムオペレーティングシステム、Java 仮想マシン、デスクトップオペレーティングシステム (Mac OS、Windows、Be OS) のプログラミングをサポートします。

CodeWarrior のほとんどの機能は、ホスト、言語、ターゲットに関係なく利用できます。CodeWarrior の共通機能については『IDE User Guide』で詳しく説明しています。

各ターゲットに特有の機能もあります。このマニュアルではそれらの独特の機能について説明します。

CodeWarrior を完璧に理解するためには、『IDE User Guide』、および各ターゲットの『Targeting』マニュアルをお読みください。

このマニュアルの各章は、他のマニュアルの内容を補足するものです (表 1.1)。ある項目について詳しく知りたい場合は、他のマニュアルとこの『Targeting』マニュアルの両方をお読みください。

表 1.1 CodeWarrior マニュアルの構成

このマニュアルの章名	参照マニュアル
アプリケーションの作成 静的ライブラリの作成	Targeting Dreamcast
Dreamcast 用ツール ターゲット設定	IDE User Guide
『CodeWarrior でのデバッグ』	IDE User Guide
Dreamcast の C/C++	C Compilers Reference

例えば、C/C++ コンパイラについて完全に理解するためには、『C Compilers Reference』(C/C++ フロントエンドコンパイラについて説明しています)と、このマニュアルの『[Dreamcast の C/C++](#)』(ターゲット用コードを生成するバックエンドコンパイラについて説明しています)の両方をお読みください。

このマニュアルについて

[表 1.2](#) にこのマニュアルの各章の概要を示します。このマニュアルでは Dreamcast 用ソフトウェア開発についての情報を説明します。このマニュアルと他の CodeWarrior マニュアルの関係については『[CodeWarrior のマニュアルについて](#)』(p8)を参照してください。

表 1.2 各章の内容

章名	内容
紹介	この章
CodeWarrior for Dreamcast をインストール	CodeWarrior for Dreamcast のインストール
Dreamcast 用ツール	Dreamcast 用ツールの説明
アプリケーションの作成	Dreamcast アプリケーションの作成
静的ライブラリの作成	Dreamcast ライブラリの作成
SHC/C++ プロジェクトを変換	既存のプロジェクトを CodeWarrior プロジェクトへ変換する
CodeWarrior でのデバッグ	CodeWarrior で Dreamcast 用アプリケーションをデバッグ
Codescape でのデバッグ	CodeWarrior と Codescape デバッガを使う
ターゲット設定	Dreamcast 用コンパイラ、リンカの設定
Dreamcast の C/C++	Dreamcast 用 C/C++ バックエンドコンパイラについて
インラインアセンブラと組み込み関数	インラインアセンブラと組み込み関数の詳細

章名	内容
ライブラリとランタイムコード	CodeWarrior が提供する Dreamcast 用ライブラリについて
トラブルシューティング	トラブルシューティングのヒント

さらに知識を深めるために

ここで紹介するマニュアルはすべて CodeWarrior CD に含まれています。

すべての方：

CodeWarrior IDE、デバッガの共通の機能については『IDE User Guide』、『Debugger User Guide』を参照してください。

C/C++ フロントエンドコンパイラの詳細は、『C Compilers Reference』を参照してください。

Dreamcast 用プログラミングの情報

Dreamcast および SH プロセッサ用のプログラミングマニュアルについては、ご使用の Dreamcast 開発ハードウェアの開発元にお尋ねください。

Metrowerks の西暦 2000 年問題対応

ライセンス契約に基づいて Metrowerks が提供する製品は、ホストまたはターゲットオペレーティングシステムによって提供される日付データを利用して内部プロセスの日付データ（ファイル修正日など）を処理しています。ゆえに、西暦 2000 年問題から生じる製品のオペレーションは、ホストまたはターゲットオペレーティングシステムの西暦 2000 年問題対応によるものです。Microsoft 社、Sun Microsystems 社、Apple Computer 社などの西暦 2000 年問題についての文書をお読みください。Metrowerks 製品自体は日付データを処理しないため、西暦 2000 年問題とは無関係です。

詳細は、<http://www.metrowerks.co.jp/y2k.htm> をご覧ください。

マニュアルの表記規則

ここではマニュアルの表記規則を説明します。

表記について

特定の情報を表すスタイルについて説明します。

注意、警告、ヒント、および初心者用のヒント

「注意」は、重要な事実を言い換えたり、自明ではない事実に注意を向けます。

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

「ヒント」は、CodeWarrior をより活用するためのヒントです。

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

書体の規則

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

CodeWarrior のメニューにある項目は括弧付き (『Open』など) で示します。

下線付きの青色のテキストは (例: [『IDE User Guide の概要』\(p17\)](#)) オンラインドキュメント (Adobe Acrobat など) でのハイパーテキストを示します。これをクリックすると該当ページへジャンプします。

第 2 章 はじめに

この章では CodeWarrior のインストール方法、および Dreamcast 用プログラミングについて説明します。

以下の内容について説明します。

[システムの必要条件](#)

[CodeWarrior for Dreamcast をインストール](#)

システムの必要条件

Pentium クラス以上のプロセッサ。最良のパフォーマンスを得るためには、Pentium II クラスのプロセッサの使用をお奨めします。

Windows 95/98、または Windows NT 4.0 オペレーティングシステム。

500MB のハードディスクスペース。

最低 32MB の RAM。64MB 以上の RAM の使用をお奨めします。

CD-ROM ドライブ。CodeWarrior ソフトウェア、ドキュメント、例題をインストールするために必要です。

HKT-01 開発ハードウェア（リビジョン 5-24）。シリアルナンバーは HKT-01 に含まれるリビジョンコードの末尾に記載されています。シリアルナンバーが S524... で始まらない場合、セガ社にお問い合わせください。

Dreamcast SDK(SEGA ライブラリ)Ver.1.55J

CodeWarrior for Dreamcast をインストール

Dreamcast 用プログラミングには CodeWarrior 開発ツールと Dreamcast 用開発ハードウェアをインストールおよび設定する必要があります。

インストーラを使ってインストールします。現時点では Dreamcast 開発用ハードウェアを PC へ接続します。

CodeWarrior ツールを使う前に、以下の手順を実行してください。

1. CodeWarrior をインストールする

[『CodeWarrior for Dreamcast ツールをインストール』\(p14\)](#) をご覧ください。

2. Dreamcast ライブラリをインストールする

[『Dreamcast ランタイムライブラリをインストール』\(p14\)](#) をご覧ください。

3. システムをテストする

プログラミングを始める前に『[Dreamcast 用開発システムの動作確認](#)』(p14)をご覧ください。

CodeWarrior for Dreamcast ツールをインストール

ソフトウェア開発の第一歩は、CodeWarrior ツールをインストールすることです。

CodeWarrior CD から `setup.exe` ファイルをダブルクリックし、インストールウィザードの指示に従ってください。インストーラについて疑問があれば、CodeWarrior Installer に含まれる情報をご覧ください。

注意： Windows 95/98 および Windows NT をインストールしたデュアルブートシステムを使用している場合、最初に Windows 95/98 ヘルプをインストールしてください。インストールを終えたら Windows 95/98 をシャットダウンします。この後で Windows NT を再起動して、Windows 95/98 で選択したのと同じディレクトリへ CodeWarrior ツールをインストールしてください。

これで CodeWarrior for Dreamcast のツールのインストールが完了しました。

Dreamcast ランタイムライブラリをインストール

SEGA ライブラリはほとんどの Dreamcast プロジェクトに必要です。

このバージョンには、CodeWarrior と互換性のある SEGA ライブラリが Dreamcast Support フォルダに含まれています。これらはインストールしたときに同時にコピーされます。

Dreamcast 用開発システムの動作確認

ソフトウェアをインストールした後、動作確認が必要です。ここでは CodeWarrior の例題ファイルの 1 つ、teapot デモをコンパイル、実行します。

1. CodeWarrior IDE を起動する

CodeWarrior IDE のアイコンを探して起動してください。

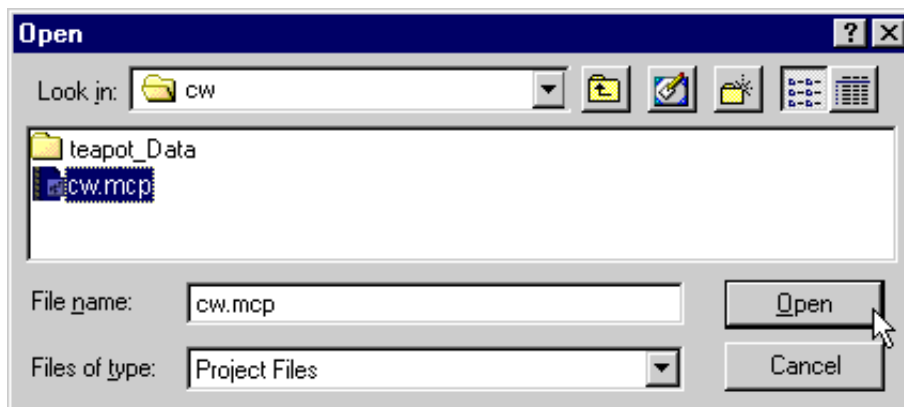
2. プロジェクトを開く

File メニューの『Open』を選択してください。ダイアログ ([図 2.1](#)) が現れます。

プロジェクトは以下のディレクトリにあります。

`sample/sample3d/teapot/cw/cw.mcp`

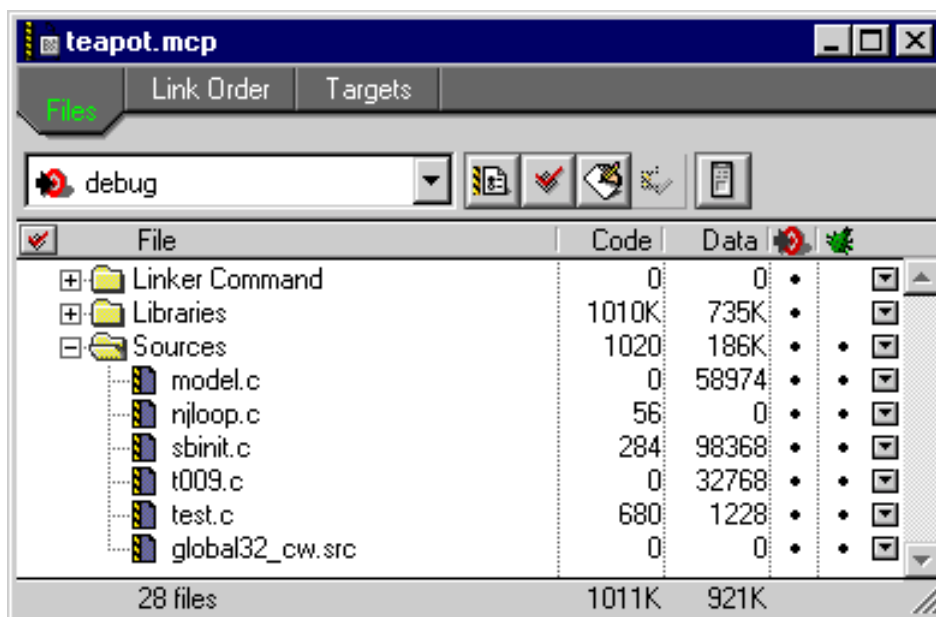
図 2.1 open ダイアログ



プロジェクトファイルを選択して開いてください。CodeWarrior のプロジェクトウィンドウ (図 2.2) が現れます。

プロジェクトウィンドウは開発作業の中心地です。ここでソースファイルの追加や削除、ライブラリの追加、コードのコンパイルやデバッグ情報の生成などをコントロールします。CodeWarrior IDE のプロジェクトマネージャの詳細は『IDE User Guide』を参照してください。

図 2.2 プロジェクトウィンドウ



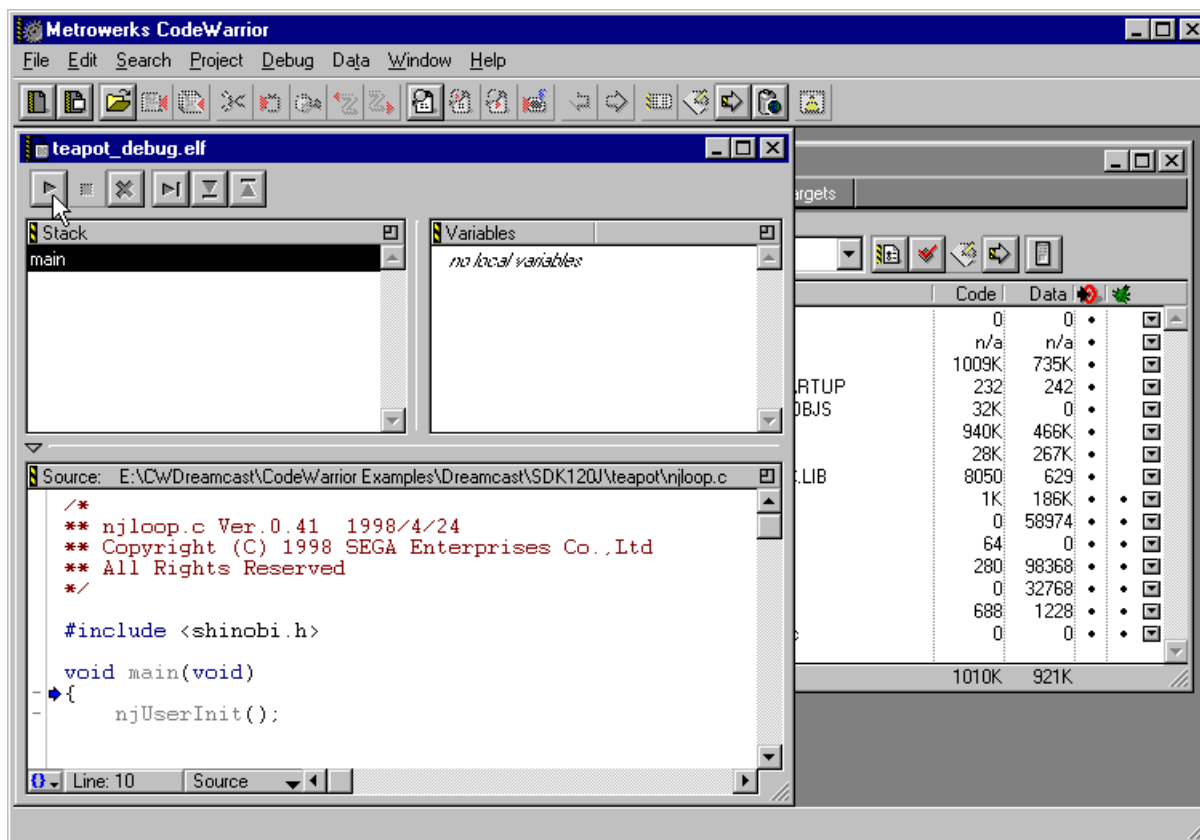
3. プロジェクトをビルドする

Project メニューの『Make』を選択してプロジェクトをビルドしてください。CodeWarrior はプロジェクトをコンパイル、リンクして `teapot_debug.elf` というプログラムファイルを作成します。

4. プロジェクトをデバッグする

Project メニューの『Debug』を選択してください。CodeWarrior がコンパイル済みのプログラムを HKT-01 ハードウェアにアップロードすると、プログラムウィンドウ (図 2.3) が現れます。

図 2.3 プログラムウィンドウ



5. プロジェクトを実行する

Project メニューの『Run』を選択してください。ソフトウェアおよびハードウェアが正しくセットアップされていれば、teapot デモが実行されます (図 2.4)。

図 2.4 teapot デモ



第 3 章 Dreamcast 用ツール

CodeWarrior for Dreamcast 開発環境のツールを紹介します。

CodeWarrior 開発環境とコマンドライン環境との開発プロセスの違いについても簡単に説明します。

以下の内容について説明します。

[Dreamcast 用ツールの紹介](#)

[CodeWarrior での開発過程](#)

Dreamcast 用ツールの紹介

Dreamcast 用プログラミングは、他の CodeWarrior ターゲットでのプログラミングとほぼ同じです。CodeWarrior for Dreamcast には以下のツールが含まれています。

[CodeWarrior IDE](#)

[Dreamcast 用 CodeWarrior コンパイラ](#)

[Dreamcast 用 CodeWarrior アセンブラ](#)

[Dreamcast 用 CodeWarrior リンカ](#)

[Dreamcast 用 Codescape デバッガ](#)

IDE やデバッガは他の CodeWarrior 製品と同じです。

CodeWarrior IDE

CodeWarrior IDE はソフトウェアを書くためのアプリケーションです。CodeWarrior IDE でプロジェクトマネージャ、ソースコードエディタ、クラスブラウザ、コンパイラ、リンカをコントロールします。

CodeWarrior プロジェクトマネージャは、コマンドラインの開発ツールにはないものです。プロジェクトマネージャはプロジェクトに関係のあるすべてのファイルを管理します。一目でプロジェクトが把握でき、またソースファイルの管理が容易にできます。

CodeWarrior IDE とコマンドライン環境の比較については、『[CodeWarrior での開発過程](#)』(p19) を参照してください。ここでは IDE が実装している従来のコマンドライン開発環境の機能を説明しています。

CodeWarrior IDE は、プラグインコンパイラ、リンカを使用する拡張可能なアーキテクチャを備えています。これによりさまざまなオペレーティングシステムやマイクロプロセッサをターゲットにすることができます。CodeWarrior for Dreamcast には、Hitachi SH-4 プロセッ

サ用の C/C++ コンパイラが含まれています。他の CodeWarrior 製品には x86 や 68000 プロセッサ、またはその他のプラットフォーム用のコンパイラが含まれます。

CodeWarrior IDE についての詳細は『IDE User Guide』をお読みください。

Dreamcast 用 CodeWarrior コンパイラ

Dreamcast 用 CodeWarrior コンパイラは ANSI 準拠の C/C++ コンパイラです。このコンパイラは、すべての CodeWarrior C/C++ コンパイラに共通のアーキテクチャを使用しています。Dreamcast 用 CodeWarrior リンカと一緒に使うと、Dreamcast 用アプリケーション、ライブラリを生成します。

コンパイラの設定については『[ターゲット設定](#)』(p43) をご覧ください。CodeWarrior の C/C++ 言語の実装については『C Compilers Reference』をお読みください。

Dreamcast 用 CodeWarrior アセンブラ

Dreamcast 用 CodeWarrior アセンブラにより、プロジェクトにアセンブラのソースコードをインクルードすることができます。

Dreamcast アセンブラプログラミングについては「Supe H RISC engine アセンブラユーザーズマニュアル」(日立制作所)をお読みください。

Dreamcast 用 CodeWarrior リンカ

Dreamcast 用 CodeWarrior リンカは、オブジェクトコードを ELF フォーマットの実行可能ファイルへリンクします。DWARF フォーマットのデバッグ情報も生成できます。このリンカは絶対アドレスを使うコードを生成します。

リンカの設定については『[ターゲット設定](#)』(p43) をご覧ください。

Dreamcast 用 CodeWarrior デバッガ

CodeWarrior デバッガはプログラムの実行をコントロールし、その内部で何が起こっているのかを見ることができます。

デバッガを使ってプログラム実行の問題点を発見することができます。デバッガはプログラムを 1 行ずつ実行したり、特定のポイントで実行を一時停止することができます。デバッガがプログラムを停止したときに、関数のコールチェーンや変数の値の変化を見たり、プロセッサレジスタの内容を調べる事ができます。

デバッガの機能やユーザーインターフェースなどについての詳細は、『Debugger User Guide』をお読みください。Dreamcast 用デバッガの詳細は『[CodeWarrior でのデバッグ](#)』(p37) を参照してください。

Dreamcast 用 Codescape デバッガ

Cross Products 社の Codescape デバッガは、CodeWarrior IDE とは別個の、スタンドアロンアプリケーションです。

CodeScape デバッガの機能やユーザーインターフェースなどについての詳細は、『CodeScape マニュアル』をご覧ください。

CodeWarrior での開発過程

CodeWarrior を使った開発過程が変わったところはありません。コードを書き、コンパイル、リンクし、デバッグを行います。編集、コンパイル、リンクなどのソフトウェア開発の方法については『IDE User Guide』をお読みください。CodeScape を使うデバッグについては『CodeScape マニュアル』をご覧ください。

CodeWarrior と従来のコマンドライン環境との違いは、ソフトウェア（この場合 IDE）が、効率的に作業の管理を支援するということです。統合開発環境、特に CodeWarrior を初めて使う方には、この節はとても役立ちます。各項目でコマンドライン環境に相当する CodeWarrior ツールを説明します。

CodeWarrior IDE とコマンドラインプログラミングの違いを説明します。

[メイクファイル](#)：ソースファイルの依存関係をコントロールし、コンパイラやリンカを設定する IDE のプロジェクト

[編集](#)：IDE でのソースコード編集の概要

[コンパイル](#)：IDE でのコンパイル

[リンク](#)：IDE でのリンク

[デバッグ](#)：IDE でのデバッグ

メイクファイル

CodeWarrior IDE における『プロジェクト』とは、メイクファイルに相当します。1 つのプロジェクトで複数のソフトウェアをビルドすることが可能なので、実際にはプロジェクトがメイクファイルの集合に相当します。例えば、1 つのプロジェクトでデバッグバージョンとリリースバージョンのコードを作成することができます。片方だけ、または両方をビルドできます。CodeWarrior では 1 つのプロジェクト内の異なるビルドを『ターゲット』と呼びます。

IDE はプロジェクトマネージャウィンドウにすべてのプロジェクトファイルを表示します。プロジェクトにはソースファイルやライブラリが含まれます。

ファイルの削除や追加は簡単です。プロジェクト内で、1 つまたは複数の異なるターゲットにファイルを割り当てることができます。これにより複数のターゲットに共通するファイルを簡単に管理できます。

IDE は自動的にファイルの依存関係を管理します。前回のビルド以降に変更されたファイルも追跡します。再度ビルドするときには変更されたファイルだけを再コンパイルします。

IDE はターゲットごとのコンパイラやリンカの設定を保存します。IDE で、またはコードの `#pragma` ステートメントを使って設定を変更できます。

編集

CodeWarrior IDE には統合テキストエディタがあります。Windows、UNIX、Mac OS フォーマットのテキストを扱うことができます。

ソースファイル、またはその他の編集可能ファイルを編集するには、プロジェクトウィンドウでそのファイル名をダブルクリックするだけです。

エディタウィンドウには優れたナビゲーション機能があり、関係のあるファイルの切り替えや、ファイル内の特定の関数や任意の位置に移動することができます。

コンパイル

コンパイルするソースファイルは、カレントターゲットに含まれていなくてはなりません。プロジェクトに含まれていれば、プロジェクトウィンドウでファイルを選択してから Project メニューの『Compile』を選択します。

カレントターゲットに含まれる、前回のコンパイル以降に変更されたファイルをすべてコンパイルするには、Project メニューの『Bring Up To Date』を選択します。

UNIX や他のコマンドライン環境では、オブジェクトファイルはソースコードから生成されるバイナリファイル(.o または .obj)として保存されます。CodeWarrior IDE はオブジェクトファイルを内部的に保存し、管理します。

リンク

オブジェクトコードを最終的なバイナリファイルへリンクするのは簡単です。Project メニューの『Make』を選択するだけです。『Make』コマンドはアクティブなプロジェクトを更新して、オブジェクトコードを最終出力ファイルへリンクします。

リンクは IDE からコントロールします。オブジェクトファイルのリストを指定する必要はありません。プロジェクトマネージャが自動的にすべてのファイルを管理します。

プロジェクトマネージャでリンクの順番の指定も可能です。

デバッグ

プロジェクトをデバッグするには、Project メニューの『Enable Debugging』を選択します。

第 4 章 アプリケーションの作成

Dreamcast 用アプリケーションはスタンドアロンの実行可能プログラムです。システムの動作確認をしたときには、デモプロジェクトをコンパイルして実行しました。

この章では自分のアプリケーションを作る方法を説明します。

アプリケーションを作成する

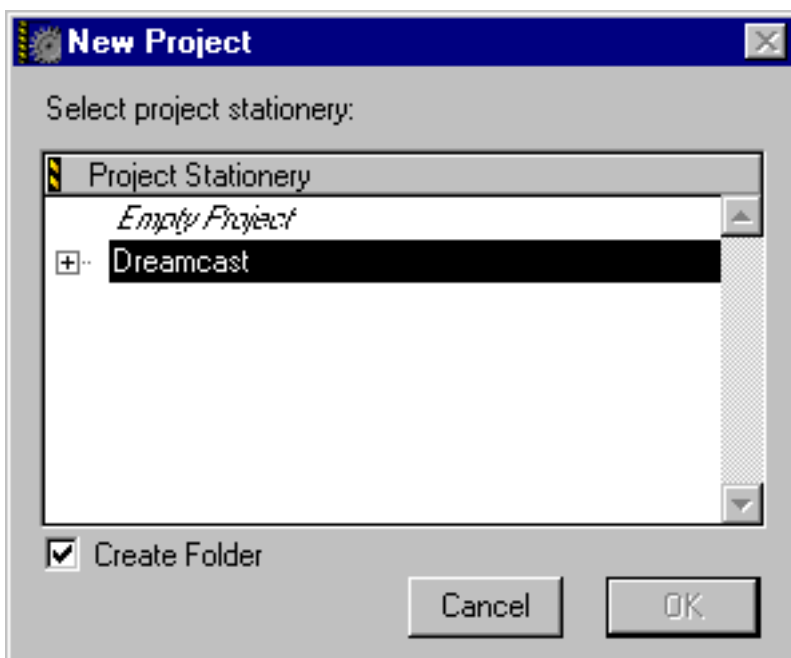
アプリケーションを作成する

Dreamcast 用アプリケーションを作る手順を説明します。

1. New Project ダイアログを表示する

CodeWarrior の File メニューから『New Project』を選択してください。New Project ダイアログ (図 4.1) が現れ、プロジェクトステーションの選択を要求します。

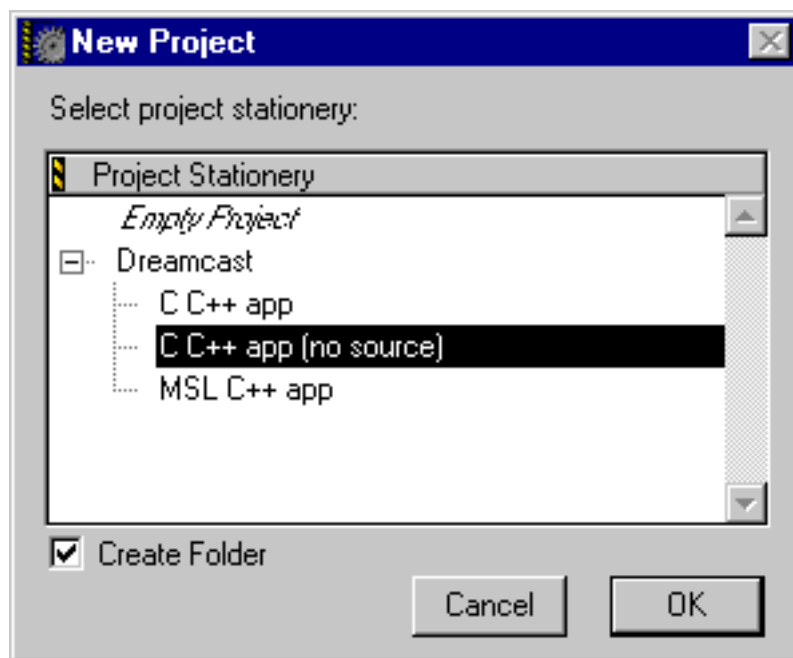
図 4.1 New Project ダイアログ



2. 利用可能な Dreamcast 用プロジェクトステーションを見る

『Dreamcast』の横の拡張ボタンをクリックすると、利用可能なプロジェクトステーションが表示されます (図 4.2)。

図 4.2 プロジェクトステーションナリを選択

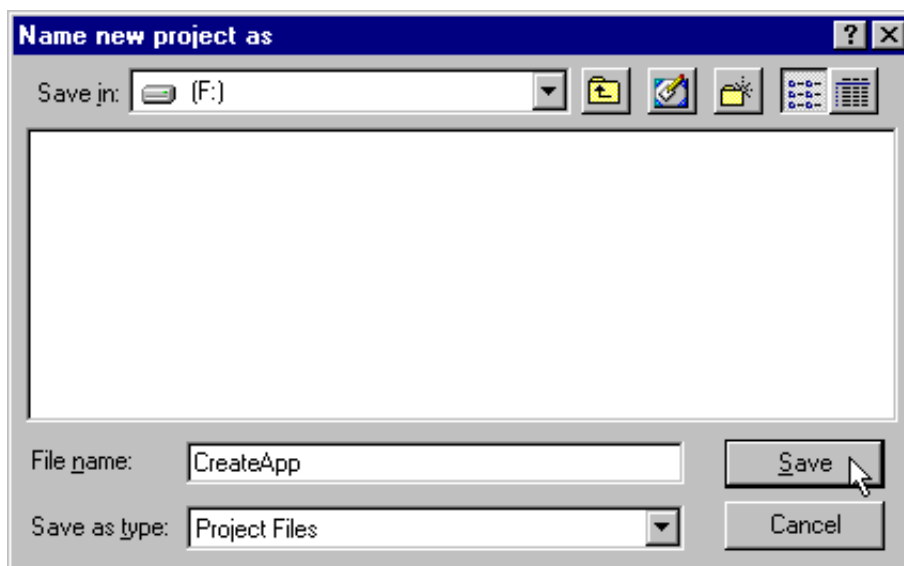


3. プロジェクトステーションナリを選択する

使用するステーションナリをクリックして、[OK] ボタンをクリックしてください。Name new project ダイアログ ([図 4.3](#)) が現れます。

注意： プロジェクトステーションナリを使わずに新規プロジェクトを作るには、New Project ダイアログで『Empty Project』を選択します。これでプロジェクトを最初の段階から作成できますが、正しいライブラリの選択やオプションの設定は複雑であるため、お奨めできません。

図 4.3 Name new project ダイアログ



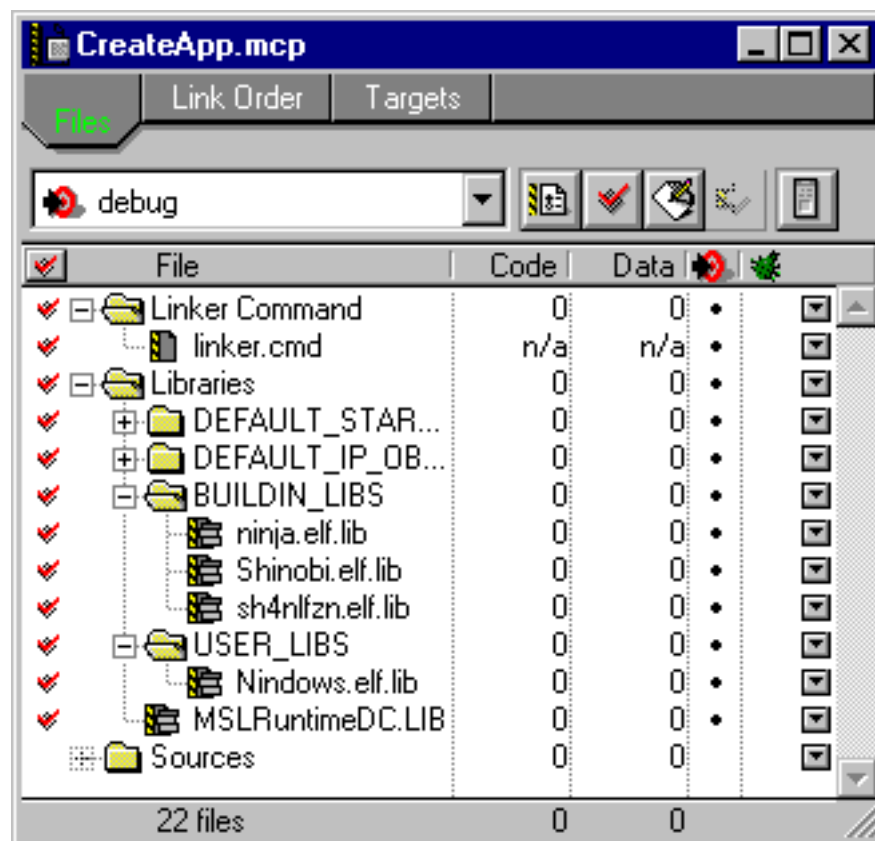
ヒント： プロジェクト名に拡張子 .mcp が付いていなければ、CodeWarrior が自動的に付加します。New Project ダイアログの『Create Folder』チェックボックスをオンにした場合は、プロジェクト名に .mcp を付けないでください。フォルダ名に .mcp が付加されてしまいます。

4. Name new project ダイアログに入力する

『保存する場所』フィールドでプロジェクトを保存するディレクトリを指定してください。『ファイル名』フィールドにプロジェクトの名前を入力してください。[保存] ボタンをクリックすると CodeWarrior は指定したディレクトリに拡張子 .mcp を付けて新規プロジェクトファイルを作成します。

ライブラリやソースフォルダを含むプロジェクトウィンドウが現れます ([図 4.4](#))。

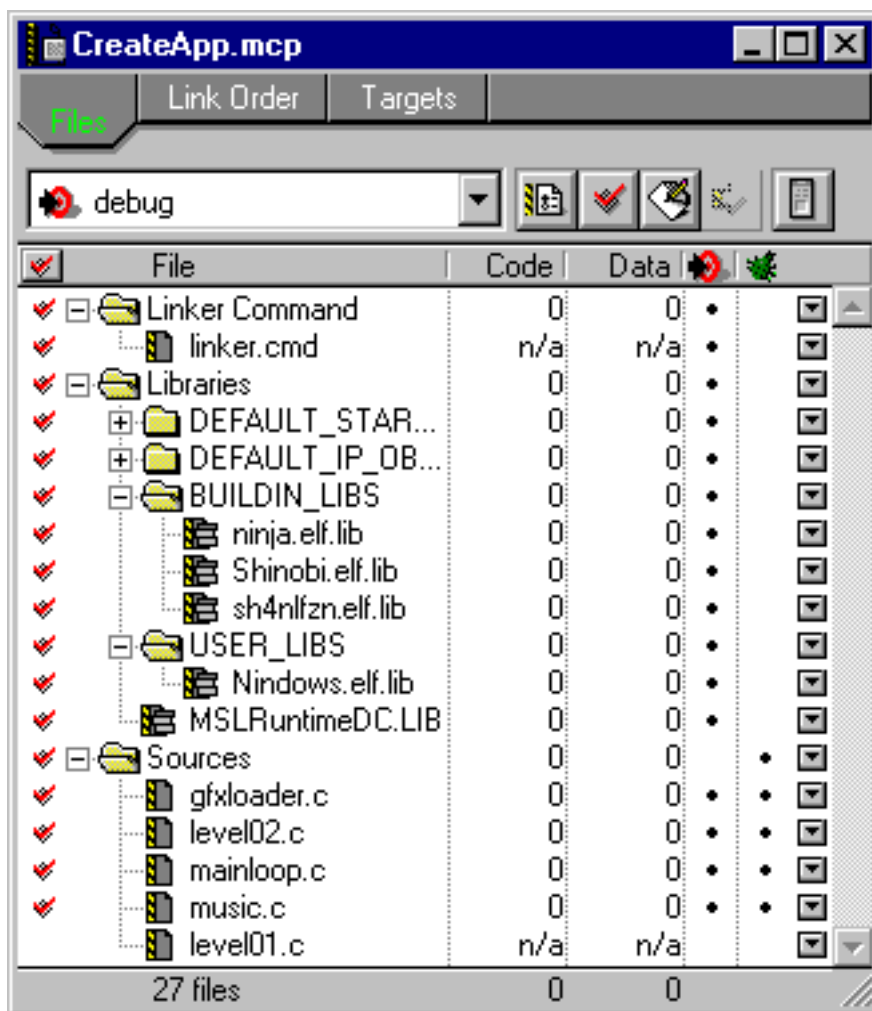
図 4.4 プロジェクトウィンドウ



5. 新規プロジェクトの内容を変更する

新規プロジェクトに自分のソースファイルを追加します。[図 4.5](#) はソースファイルが追加されたプロジェクトウィンドウです。

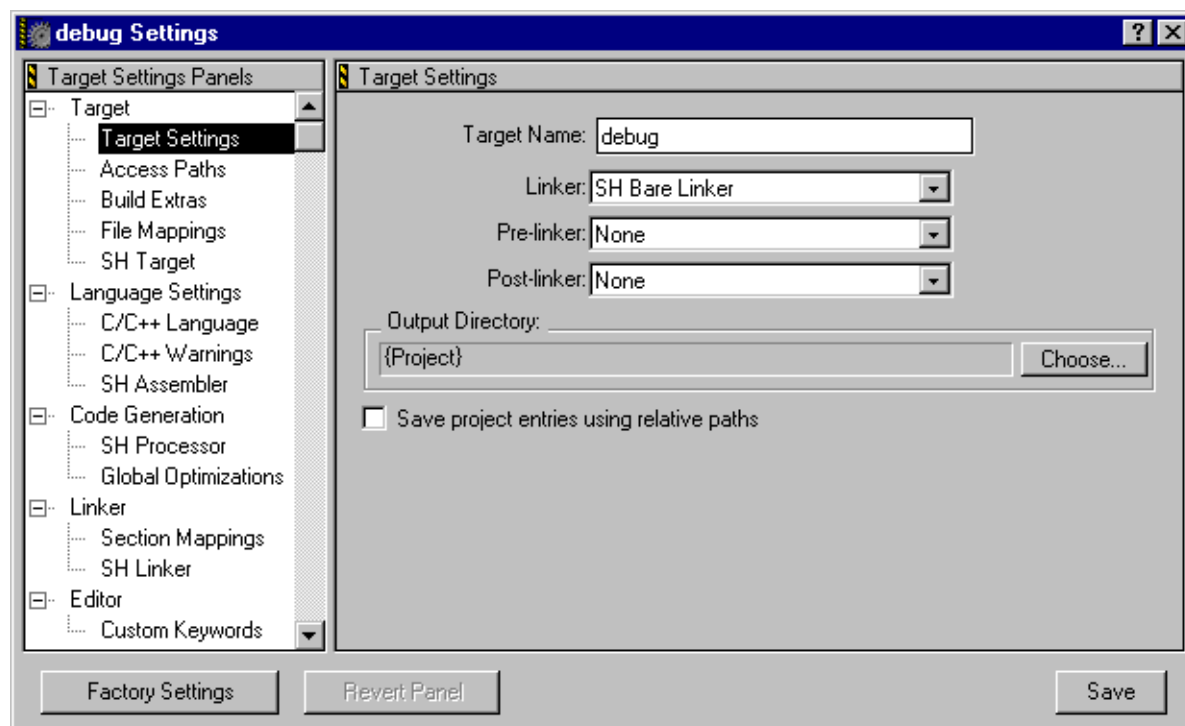
図 4.5 変更後のプロジェクトウィンドウ



6. ターゲット設定ダイアログを開く

プロジェクトウィンドウをアクティブ（最前面）にしてから、Edit メニューの『Target Settings』を選択してください。（実際には『Target』の部分にプロジェクトのカレントターゲットの名前が入ります）。図 4.5 の場合、メニューコマンドは『debug Settings』となります。

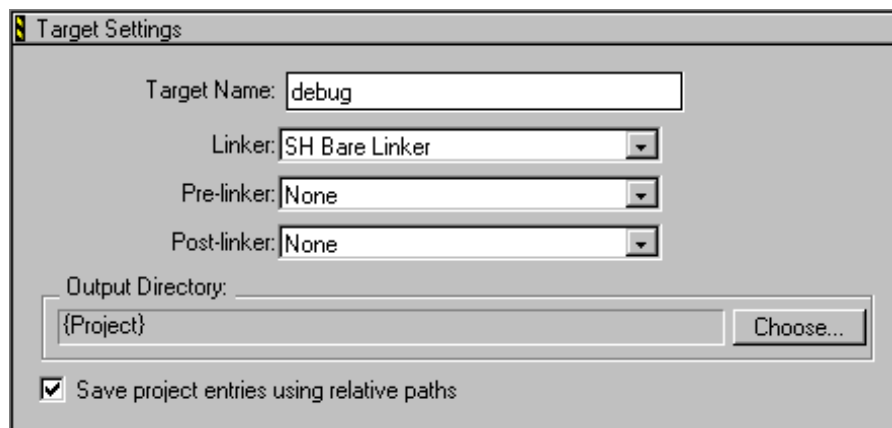
図 4.6 ターゲット設定ダイアログ



ターゲット設定ダイアログ (図 4.6) でプロジェクトに適したオプション設定を行います。

Dreamcast 用プロジェクトを作成するには、ターゲットプラットフォーム、プロジェクトの種類、コンパイラ、リンカを指定します。その他のオプションもあります。

図 4.7 Target Settings 設定パネル



7. ターゲットを設定する

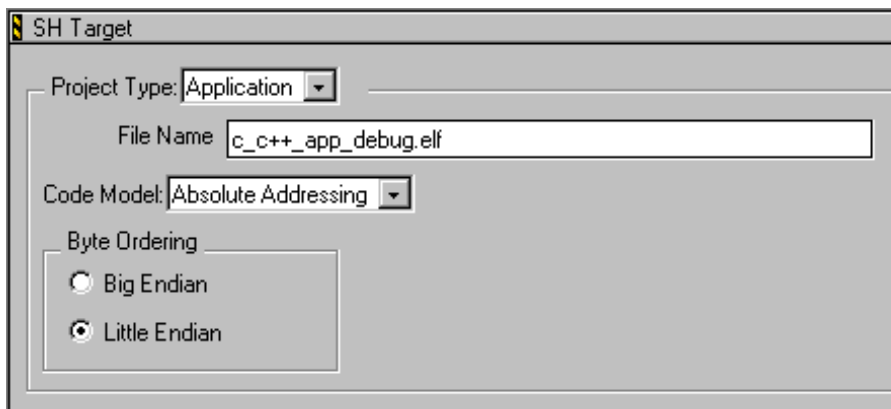
ターゲット設定ダイアログの左側に設定パネルのリストが表示されます。『Target Settings』を選択してください。プロジェクトで現在選択しているターゲットを設定するための

Target Settings 設定パネル (図 4.7) が右側に表示されます。『Linker』 オプションはデフォルトで『SH Linker』に設定されています。ターゲットの名前やオプションは変更することができます。

8. プロジェクトの種類を指定する

リストで『SH Target』をクリックすると設定パネル (図 4.8) が表示されます。プロジェクトステーションナリによってデフォルトの設定がされています。アプリケーションを作成する場合、『Project Type』は『Application』のままにしておきますが、その他のオプションは変更することができます。

図 4.8 SH Target 設定パネル (アプリケーションプロジェクト)



9. その他のオプションを設定する

その他の設定パネルの詳細は『[ターゲット設定](#)』(p43) をご覧ください。また『IDE User Guide』、『C Compilers Reference』も参照してください。

プロジェクトの設定を終えたら、ターゲット設定ダイアログを閉じてください。

10. プロジェクトをビルドする

プロジェクトを作成し、必要なファイルとオプションを設定したら、コンパイルとデバッグを行います。Project メニューの『Make』はプロジェクトをコンパイル、リンクします。成功すると、出力ファイルがプロジェクトフォルダに保存されます。出力ファイルの名前は SH Target 設定パネルで指定します。

コンパイル、リンクの詳細は『IDE User Guide』を参照してください。

11. アプリケーションをデバッグする

プロジェクトのビルドが成功したら、デバッガを起動してコードを実行し、デバッグします。

第 5 章 静的ライブラリの作成

この章では Dreamcast プロジェクトにおける静的ライブラリの役割とその作成方法を説明します。

[静的ライブラリについて](#)

[静的ライブラリを作成する](#)

実行可能アプリケーションの作成方法は『[アプリケーションの作成](#)』(p21) をご覧ください。プロジェクト全般については『IDE User Guide』を参照してください。

静的ライブラリについて

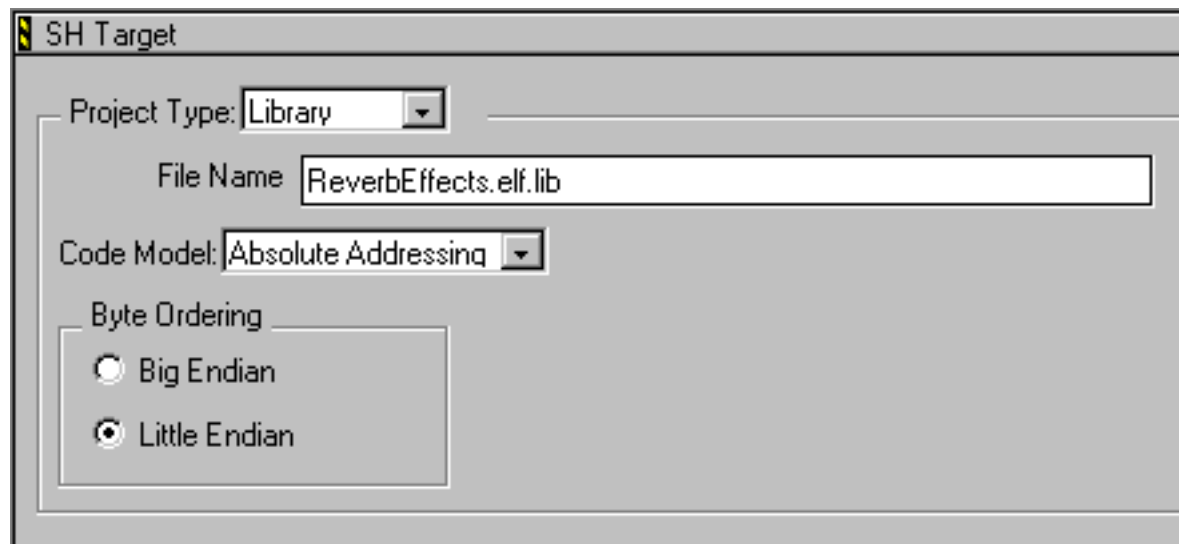
静的ライブラリは、アプリケーション(または他のライブラリ)に組み込まれる関数やデータをまとめたものです。CodeWarrior は定義済みのライブラリを提供していますが、自分のプロジェクトで使用するために、独自に設計したライブラリを作ることができます。

静的ライブラリを作成する

静的ライブラリを作る手順はスタンドアロンアプリケーションを作る手順とほとんど同じですが、例外がいくつかあります。

SH Target 設定パネル ([図 5.1](#)) の『Project Type』ポップアップメニューで『Library』を選択してください。

図 5.1 SH Target 設定パネル



独自の拡張子を使うことができます。CodeWarrior の命名規約ではライブラリの拡張子は `.lib`、実行可能ファイルは `.elf` です。

ビルドが成功した静的ライブラリを他のアプリケーションへ組み込むことができます。この場合、アプリケーションをビルドする前に、静的ライブラリをプロジェクトウィンドウへ追加してください。

静的ライブラリ自体をデバッグすることはできません。しかしそれが組み込まれているアプリケーションの一部としてデバッグできます。

アプリケーションプロジェクトの作成手順は『[アプリケーションの作成](#)』(p21) を参照してください。プロジェクトの設定パネルについては『[ターゲット設定](#)』(p43)、また『IDE User Guide』、『C Compilers Reference』をお読みください。

第 6 章 SHC/C++ プロジェクトを変換

この章では既存のメイクファイルベースの SH プロジェクトを CodeWarrior プロジェクトへ変換する方法を説明します。

[SHC/C++ プロジェクトを変換する](#)

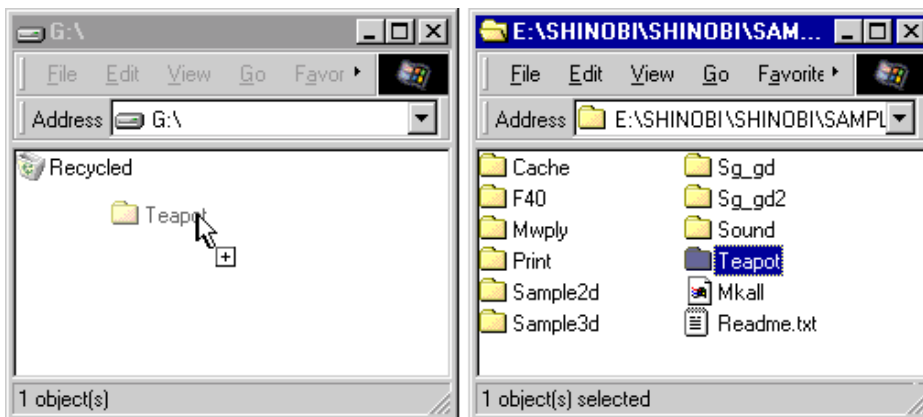
SHC/C++ プロジェクトを変換する

以下の手順で SDK Teapot デモを CodeWarrior プロジェクトへ変換します。

1. teapot サンプルを自分のフォルダにコピーする

すべての teapot ファイルを新しいフォルダにコピーしてください。図 6.1 では、新しい teapot フォルダは G:\ にあります。

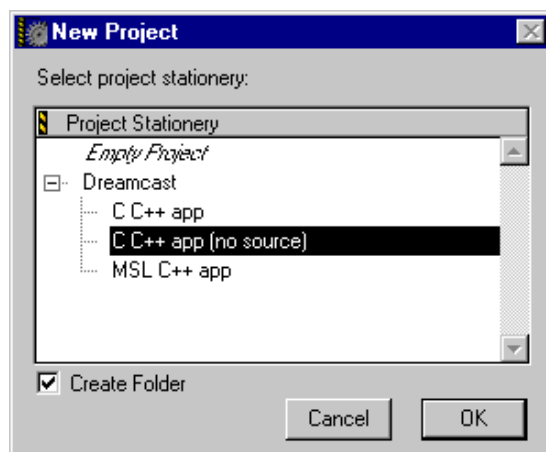
図 6.1 teapot ファイルを新しいフォルダへコピー



2. 新規プロジェクトを作成する

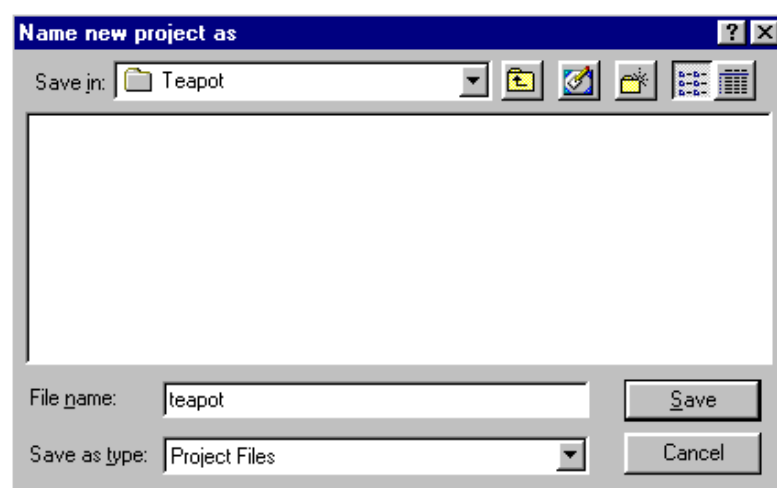
CodeWarrior の File メニューから『New Project』を選択してください。New Project ダイアログで Dreamcast プロジェクトステーションナリから『C app (no source)』を選択し (図 6.2) [OK] ボタンをクリックしてください。

図 6.2 ステーションナリを選択 : Dreamcast C app (no source)



『Create Folder』チェックボックスはオフにしておきます。新規プロジェクト用のフォルダ (teapot フォルダ) が既にあるからです。図 6.3 のように、新規プロジェクトを teapot フォルダに保存し、『File name』フィールドに『teapot』と入力してください。

図 6.3 新規プロジェクトを teapot フォルダへ保存



3. メイクファイルからソースファイルを追加する

Makeuser ファイルは、プロジェクトへ追加すべきソースファイルの名前を含んでいます。teapot フォルダにある Makeuser ファイルを開いてください。

図 6.4 Makeuser からソースファイルを探す

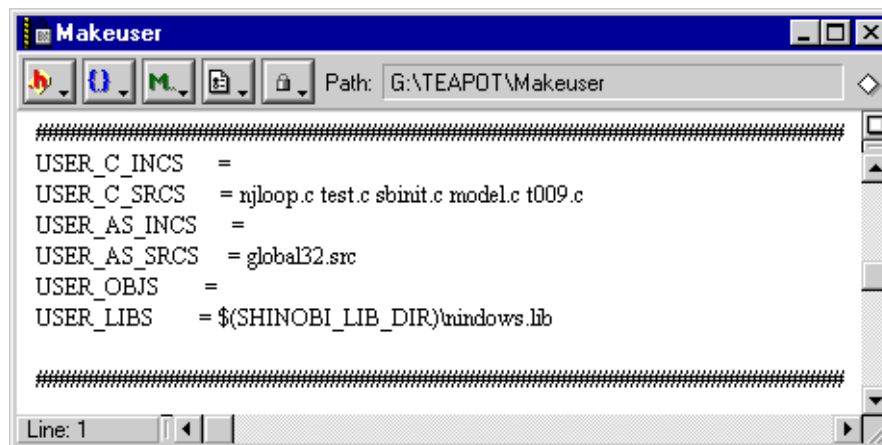
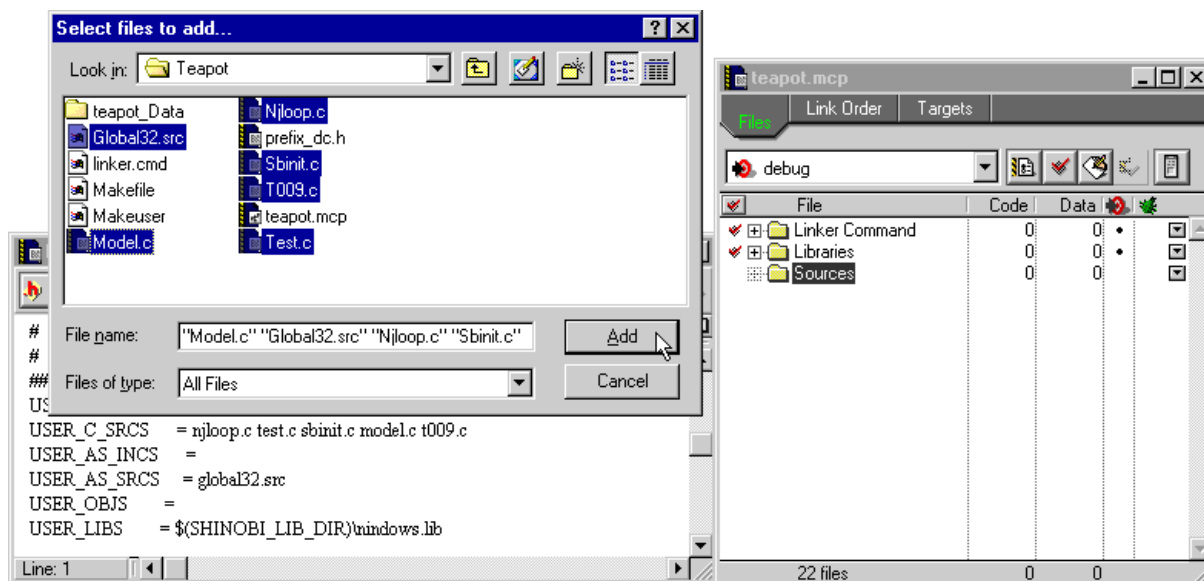


図 6.4 にリストされているファイルを CodeWarrior プロジェクトへ追加します。これらをプロジェクトの『Sources』グループへ入れます。

プロジェクトウィンドウで『Sources』グループを強調表示してください。Project メニューの『Add Files...』を選択してください。Select Files ダイアログ (図 6.5) が現れます。ここで teapot フォルダからソースファイルを選択し、プロジェクトへ追加します。追加したファイルは自動的にリンク順の下に置かれます。

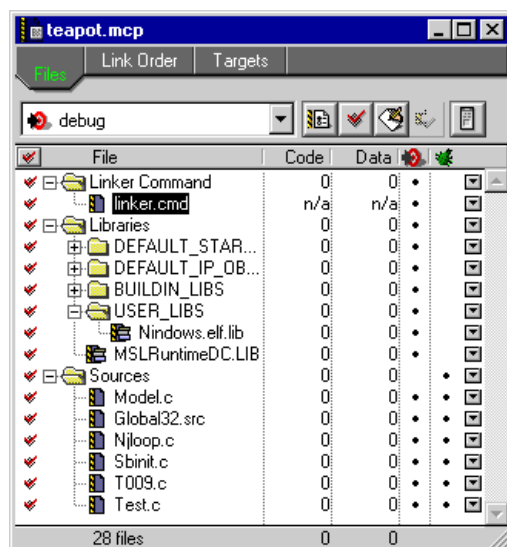
図 6.5 プロジェクトウィンドウへソースファイルを追加



nindows.lib を追加する必要はありません。CodeWarrior バージョンの nindows.elf.lib がステーションリに含まれているためです。これは『Libraries』グループの中の『USER_LIBS』にあります。

ソースファイルを追加したプロジェクトウィンドウは図 6.6 のようになります。

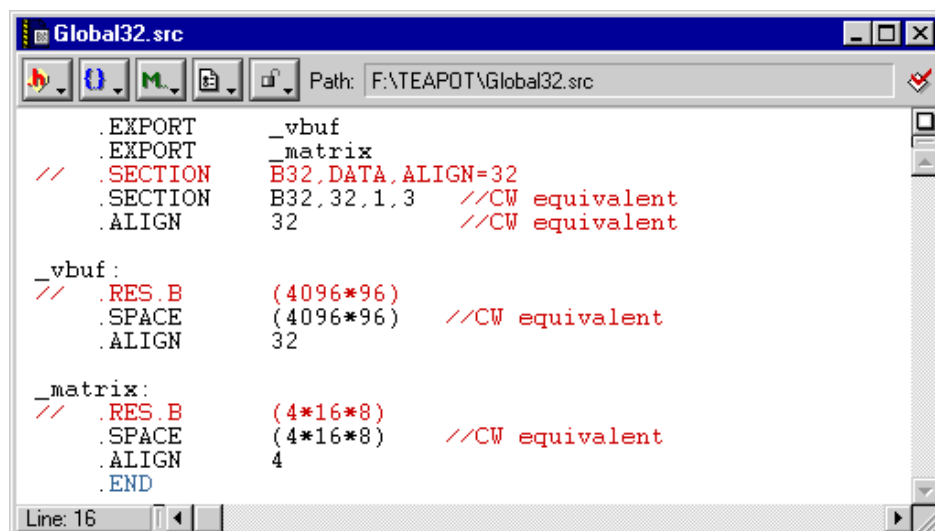
図 6.6 ファイルを追加した後のウィンドウ



4. アセンブラファイルを変換する

CodeWarrior で teapot をコンパイルする前に、アセンブラソースファイル、global32.src を変更しなくてはなりません(図 6.7)。global32.src ファイル内のアセンブラ疑似命令は Hitachi アセンブラの挙動をコントロールします。CodeWarrior アセンブラが使う疑似命令とは若干異なるので、Hitachi アセンブラ疑似命令を CodeWarrior のそれに置換します。

図 6.7 Hitachi アセンブラを CodeWarrior アセンブラに変換



Hitachi アセンブラの .SECTION 疑似命令は B32 セクションを bss セクションとして、32 バイトでアラインしています。CodeWarrior では以下のようになります。

```
SECTION B32, 32, 8, 3  
.ALIGN 32
```

Hitachi アセンブラの `.SECTION` 疑似命令を Metrowerks の SH アセンブラにおける疑似命令に置換します。

注意： ELF セクションフラグの完全なリストは、「SH-4 Assembler Reference」の「疑似命令の使い方」の章を参照してください。

CodeWarrior では `.RES.B` 疑似命令ではなく、`.SPACE` 疑似命令を使います。`.RES.B` を `.SPACE` で置換してください。

5. 変換作業を終了する

teapot サンプルを CodeWarrior プロジェクトへ変換できました。これで他のプロジェクトと同様にコンパイルやデバッグができます。

第 7 章 CodeWarrior でのデバッグ

この章では CodeWarrior デバッガを使って Dreamcast コードをデバッグする方法を説明します。Dreamcast プラットフォーム特有の操作方法などを解説します。

[CodeWarrior デバッガを使う](#)

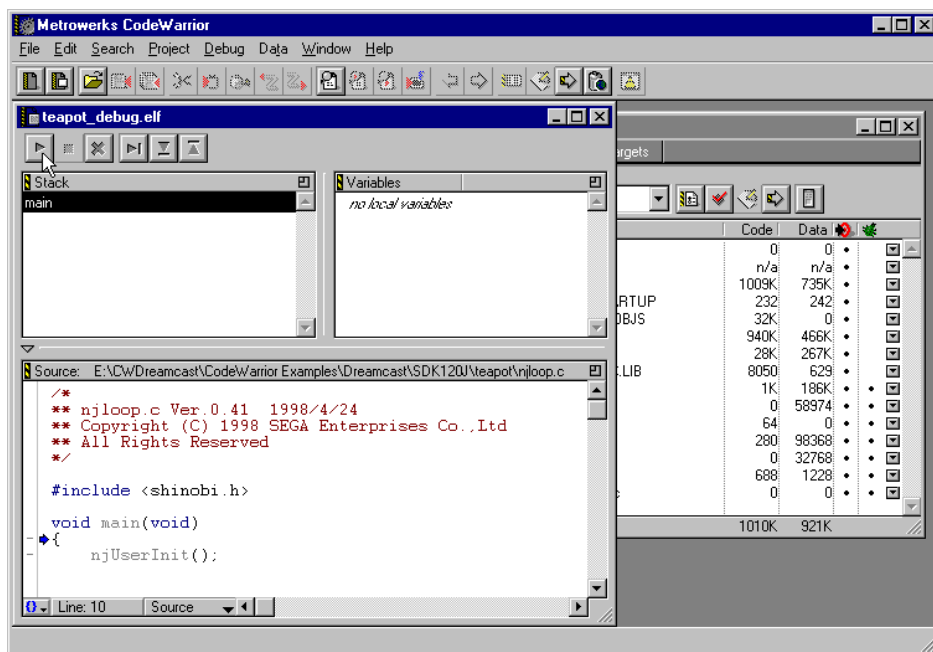
[mw_pr\(\) を使う](#)

[静的ライブラリをデバッグ](#)

CodeWarrior デバッガを使う

Project メニューの『Debug』を選択してください。デバッガのプログラムウィンドウ ([図 7.1](#)) が現れます。

図 7.1 プログラムウィンドウ



プログラムウィンドウはスタックロール欄、変数欄、ソース欄に分かれています。デバッガのコントロールバーはウィンドウの上部にあります。ここからプログラムの実行、停止、ステップ実行が可能です。

CodeWarrior デバッガの詳細は『Debugger User Guide』を参照してください。

mw_pr() を使う

`printf()` 関数は Dreamcast コードのデバッグでは動作しません。代わりに `mw_pr(const char *)` 関数を提供しています。これはコンソールウィンドウへ文字列を送信します。

`mw_pr()` を使うには、`mw_output.lib` というライブラリをプロジェクトへ追加します。このライブラリは `Dreamcast Support` フォルダにあります。

`mw_pr()` は入力としてポインタを使います。`\n` を改行キャラクタとして認識し、最大 1024 バイトの文字列を受け付けます。以下は使用例です。

```
char *p = "Hello World!\n";  
mw_pr(p);
```

静的ライブラリをデバッグ

大きなアプリケーションの一部として静的ライブラリをデバッグすることができますが、静的ライブラリだけをデバッグすることはできません。

第 8 章 Codescape でのデバッグ

この章では CodeWarrior と Codescape を使って Dreamcast コードをデバッグする方法について説明します。

[Codescape デバッガを使う](#)

[debug-printf\(\) を使う](#)

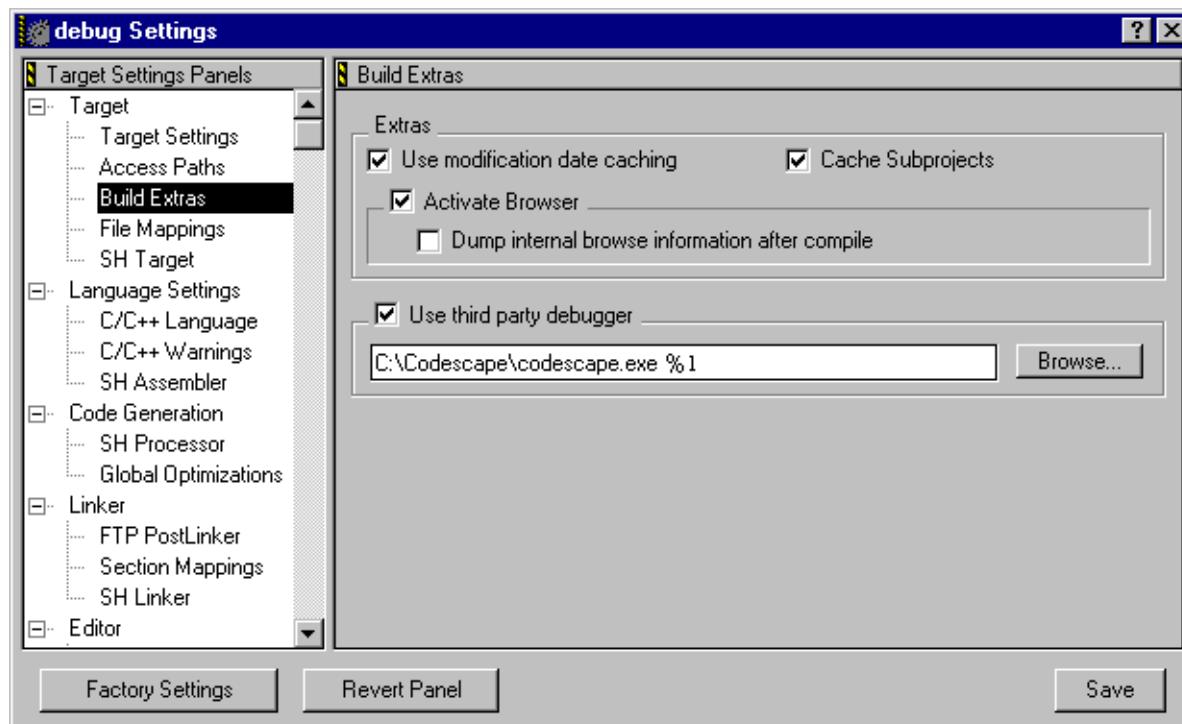
注意： Codescape デバッガの最新情報はリリースノートをご覧ください。

Codescape デバッガを使う

Project メニューの『Debug』を選択したときに、(CodeWarrior デバッガではなく)Codescape デバッガを起動させるためには、Codescape をサードパーティデバッガとして設定する必要があります。

Build Extras 設定パネル ([図 8.1](#)) の『Use third party debugger』チェックボックスをオンにして、Codescape の実行可能ファイルへのパスを指定してください。

図 8.1 サードパーティデバッガとして CodeScape を設定

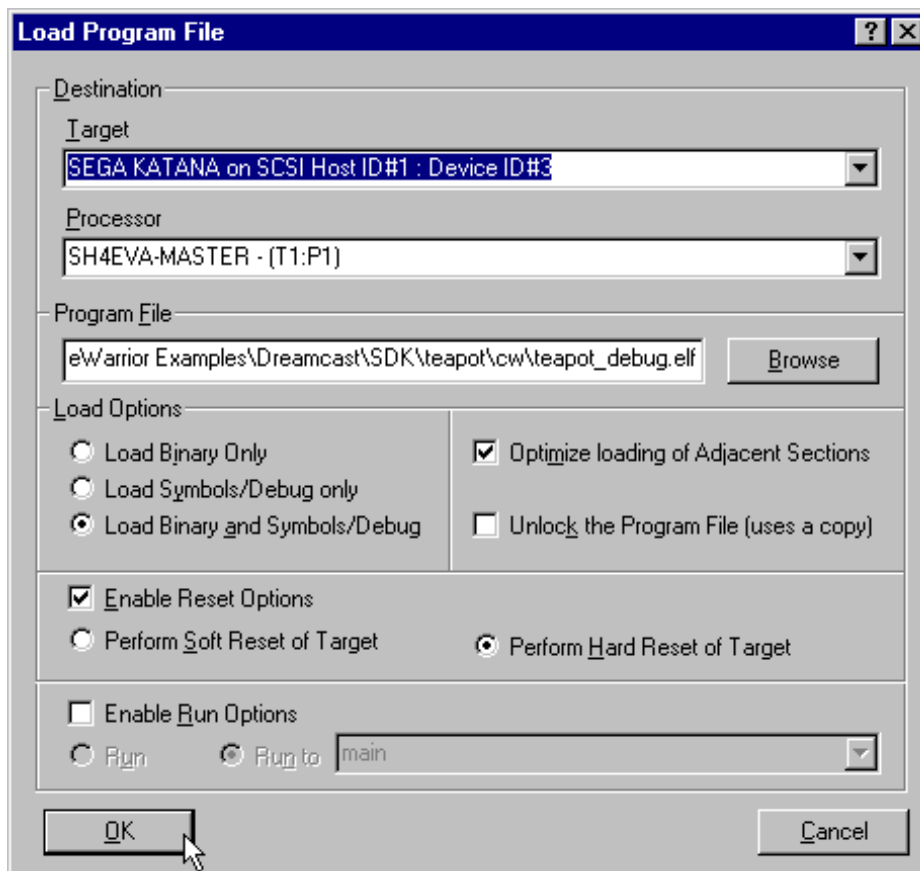


これにより、Project メニューの『Debug』を選択したときに CodeWarrior は自動的に Codescape デバッガを起動します。

Codescape デバッガを使用している場合、File メニューの『Load Program File』を選択すると CodeWarrior でビルドした実行可能ファイルがデバッガにロードされます。図 8.2 は、実行可能ファイル、teapot_debug.elf をデバッグする際の設定です。

注意： Build Extras の Use third party debugger で \Codescape\codescape.exe %1 と設定すると、Codescape デバッガに対してターゲットのプログラムのファイル名が渡されます。

図 8.2 Codescape デバッガの Load Program File メニュー



Codescape デバッガの使い方は『Codescape マニュアル』をご覧ください。

debug_printf() を使う

debug_printf()関数はCodescapeデバッガでは動作しません。デバッガコンソールウィンドウに文字列を表示するには、LIBCRS ライブラリ（Cross Products 社）が必要です。詳細は Codescape のドキュメントをご覧ください。

第 9 章 ターゲット設定

Dreamcast用CodeWarrior開発環境でコード生成に関係する設定パネルについて説明します。各設定パネルにあるオプションの設定によってコンパイラ、リンカなどをコントロールします。

コンパイラとリンカの詳細（プラグマ、シンボルなど）は『[Dreamcast の C/C++](#)』をご覧ください。

[ターゲット設定の概要](#)

[Dreamcast 用の設定パネル](#)

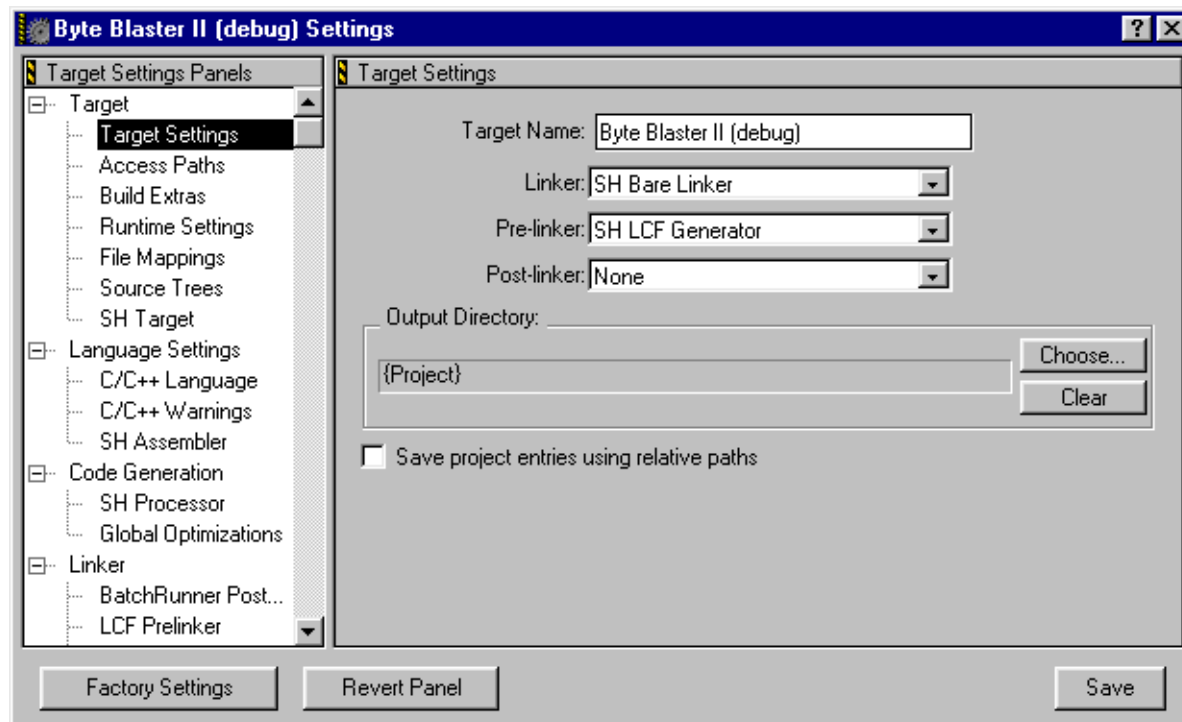
ターゲット設定の概要

CodeWarrior プロジェクトのターゲットは、それぞれ個別に設定します。この設定によってコンパイラオプション、リンカ出力、エラー / 警告メッセージなどをコントロールします。ターゲット設定ダイアログでこれらを変更できます。詳細は『IDE User Guide』をお読みください。

ターゲット設定ダイアログに表示される設定パネルのオプションによってコンパイラとリンカの挙動をコントロールします。ターゲット設定ダイアログを表示するには、Edit メニューの『Target Settings』を選択します（Target には CodeWarrior プロジェクトのカレントターゲットの名前が入ります）。または、プロジェクトウィンドウの Target ビューで該当するターゲットをダブルクリックしてください。

ターゲット設定ダイアログ（[図 9.1](#)）が現れます。

図 9.1 ターゲット設定ダイアログ



設定ダイアログの左側にあるリストから、表示したい設定パネルを選択してください。パネルで必要に合わせてオプションを変更します。

既存のプロジェクトの設定を変更した場合、各パネルの [Revert Panel] ボタンをクリックすると元の状態に戻すことができます。出荷時の設定状態に戻すには、[Factory Settings] ボタンをクリックします。

ヒント： 新規プロジェクトを作成する場合、プロジェクトステーションリを使ってください。このステーションリではすべての設定パネルが適切な値に設定されています。新規プロジェクトの設定を変更してステーションリフォルダに保存するだけで、独自のステーションリファイルを作成することもできます。詳細は『IDE User Guide』を参照してください。

Dreamcast 用の設定パネル

Dreamcast 用開発環境に特有な設定パネルについて説明します。

[Target Settings](#)

[SH Target](#)

[SH Assembler](#)

- [SH Processor](#)
- [Global Optimizations](#)
- [BatchRunner PostLinker](#)
- [LCF Prelinker](#)
- [Section Mappings](#)
- [SH Linker](#)
- [Debugger Settings](#)

共通の設定パネルはこれ以外の CodeWarrior マニュアルで説明しています。[表 9.1](#) に参照すべきマニュアルを示します。

表 9.1 設定パネルの参照マニュアル

設定パネル	マニュアル
Access Paths	IDE User Guide
Build Extras	IDE User Guide
File Mappings	IDE User Guide
Custom Keywords	IDE User Guide
C/C++ Language	C Compilers Reference
C/C++ Warnings	C Compilers Reference

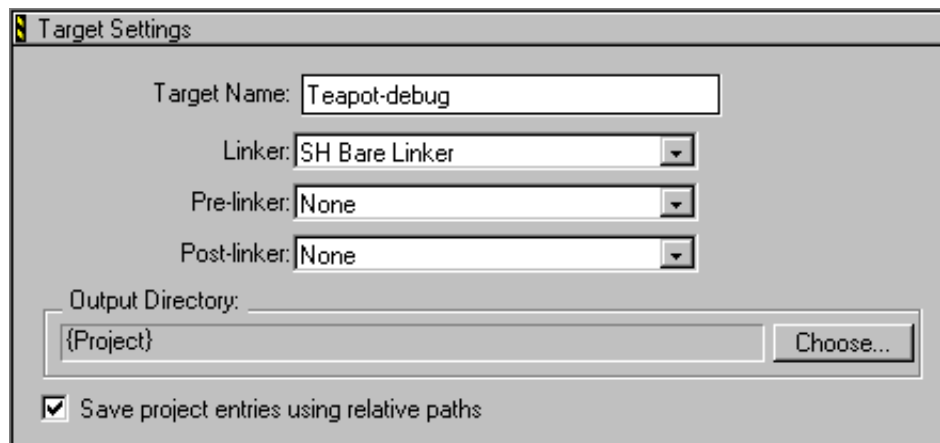
Target Settings

各設定パネルはターゲット設定ダイアログに表示されます。この 2 つは違うものです。ダイアログはパネルを 1 度に 1 つずつ表示します。Target Settings 設定パネルは設定パネルの 1 つです。

Target Settings 設定パネル ([図 9.2](#)) は非常に重要です。このパネルでターゲットを指定します。リンカを選択することによって、ターゲットとなるオペレーティングシステム / プロセッサが決定されます。この選択によってターゲット設定ダイアログに表示される設定パネルが変化します。

選択したリンカによって表示されるパネルが異なります。最初にターゲットを指定し、その後ターゲット特有のコンパイラやリンカの設定を行います。

図 9.2 Target Settings 設定パネル



注意： Target Settings 設定パネルと [SH Target](#) 設定パネルは違うものです。ターゲットは Target Settings 設定パネルで指定します。プロジェクトに関するその他のオプションを [SH Target](#) 設定パネルで指定します。

以下のオプションがあります。

[Target Name](#)

[Linker](#)

[Pre-linker](#)

[Post-Linker](#)

[Output Directory](#)

[Save Project Entries Using Relative Paths](#)

Target Name

『Target Name』フィールドでターゲットの名前を入力、または変更します。ここで指定した名前がプロジェクトウィンドウの Targets ビューで表示されます。

ここで指定するターゲットの名前は、最終出力ファイルの名前ではありません。ターゲット名はそのターゲットに個人的に付ける名前です。最終出力ファイルの名前は [SH Target](#) 設定パネルで指定します。

Linker

『Linker』ポップアップメニューでリンカを選択します。Dreamcast 用リンカは『SH Bare Linker』です。

Pre-linker

『Pre-Linker』フィールドに、リンク前にオブジェクトコードを処理するプリリンカ（LCF Generator）を指定します。

Post-Linker

『Post-Linker』フィールドでオブジェクトコードのフォーマットの変換など、追加作業を行うポストリンカを指定します（例：Batch Runner Post Linker）。

Output Directory

『Output Directory』フィールドに最終的にリンクした出力ファイルを保存するディレクトリを指定します。。デフォルトはプロジェクトファイルのあるディレクトリです。ディレクトリを変更するには、[Choose] ボタンをクリックします。

Save Project Entries Using Relative Paths

1 つのプロジェクトに同名のファイルを複数追加するには、このオプションをオンにします。オフの場合、各ファイルの名前はユニークでなくてはなりません。

オンの場合、IDE はファイルの名前とアクセスパス情報を記憶します。ファイルを検索するとき、IDE は『Access Path』の設定と各ファイルのアクセスパス情報を使います。

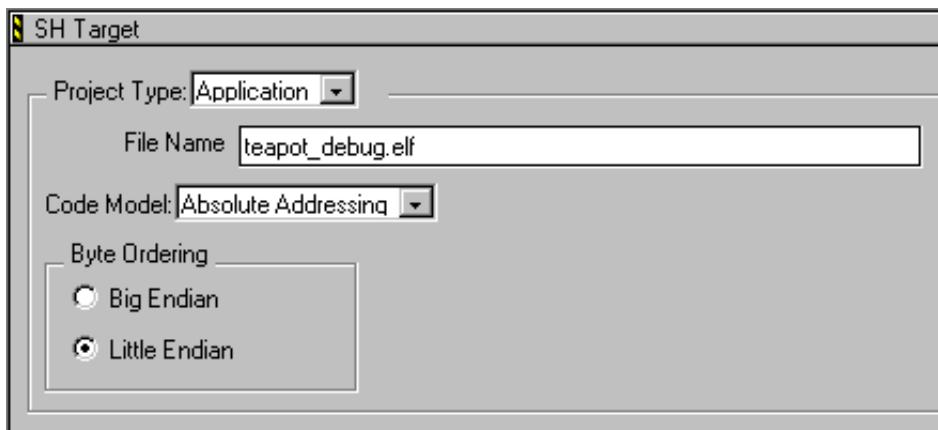
オフの場合、IDE はプロジェクトに追加されたファイルの名前だけを記憶します。ファイルを検索するとき、IDE は『Access Path』の設定だけを使います。

SH Target

SH Target 設定パネル（[図 9.3](#)）で最終出力ファイルの名前を指定します。

このパネルに表示されるオプションは、作成するプロジェクトの種類によって異なります。

図 9.3 SH Target 設定パネル



以下のオプションがあります。

[Project Type](#)
[File Name](#)

[Code Model](#)
[Byte Ordering](#)

Project Type

『Project Type』ポップアップメニューで作成するプロジェクトの種類を選択します。[図 9.4](#)にプロジェクトの種類を示します。

図 9.4 SH Target のプロジェクトの種類



通常は『Application』を選択します。

File Name

『File Name』フィールドに、実行可能ファイルまたはライブラリを入力します。Metrowerks の命名規約では、実行可能ファイルに `.elf`、ライブラリに `.elf.lib` という拡張子を付けます。

Byte Ordering

『Byte Ordering』ボタンで生成したコードを保存するフォーマットを指定します。ビッグエンディアンでは最も大きいバイトが先になります (B3 B2 B1 B0)。リトルエンディアンでは最も小さいバイトが先になります (B0 B1 B2 B3)。

Dreamcast 用アプリケーションには、『Little Endian』を選択してください。

Code Model

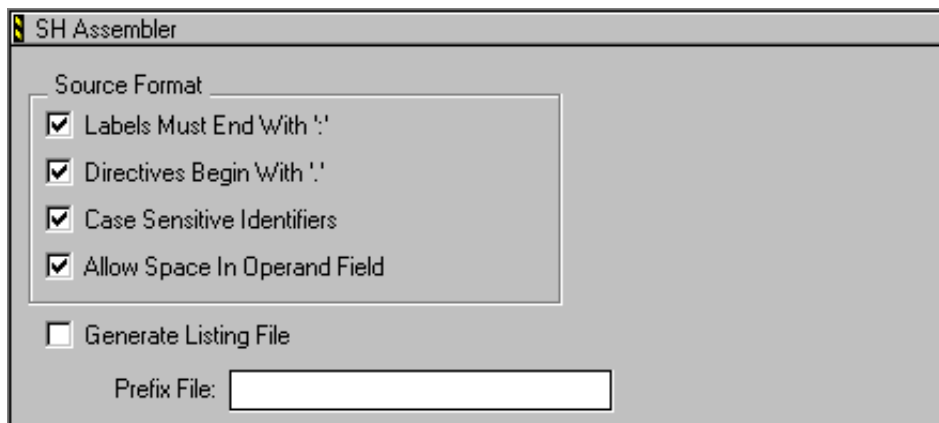
『Code Model』ポップアップメニューで実行可能ファイルのアドレッシングモードを選択します。

Dreamcast 用アプリケーションには、『Absolute Addressing』を選択してください。

SH Assembler

SH Assembler 設定パネル ([図 9.5](#)) で SH アセンブラによるアセンブリ言語の処理方法をコントロールします。

図 9.5 SH Assembler 設定パネル



以下のオプションがあります。

[Labels Must End With ':'](#)

[Case Sensitive Identifiers](#)

[Generate Listing File](#)

[Directives Begin With ':'](#)

[Allow Space In Operand Field](#)

[Prefix File](#)

Labels Must End With ':'

ラベルがコロン (:) で終わるか否かを指定します。

Directives Begin With ':'

アセンブラ疑似命令がピリオド (.) で始まるか否かを指定します。

Case Sensitive Identifiers

ソースコードで使用している通りの大文字と小文字を使って識別子を表示します。オフの場合、識別子は大文字でのみ表示されます。

Allow Space In Operand Field

オペランドの区切りに空白スペースを使用するか否かを指定します。

Generate Listing File

プロジェクトのソースファイルをアセンブルしたときにリストファイルを生成するか否かを指定します。

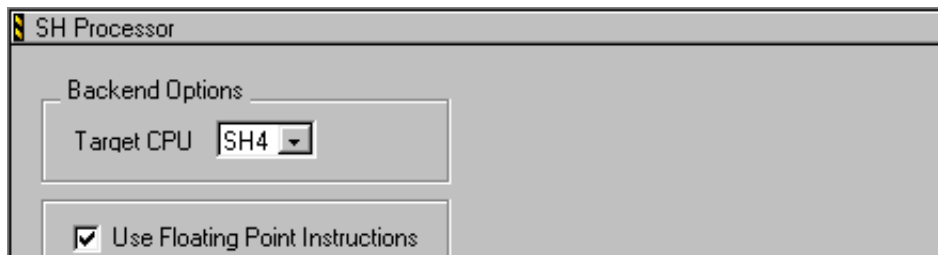
Prefix File

プロジェクトのアセンブラファイルに自動的にインクルードされるファイルを指定します。このフィールドで共通の定義をソースファイルに一度にインクルードすることもできます。

SH Processor

SH Processor 設定パネル ([図 9.6](#)) で Dreamcast 用コードの生成をコントロールします。

図 9.6 SH Processor 設定パネル



以下のオプションがあります。

[Target CPU](#)

[Use Floating Point Instructions](#)

Target CPU

このポップアップメニューでどのプロセッサ用のコードを生成するかを指定します。Dreamcast 用には、『SH4』を選択してください。

Use Floating Point Instructions

オンの場合、コンパイラはプロセッサの浮動小数点インストラクションを利用します。

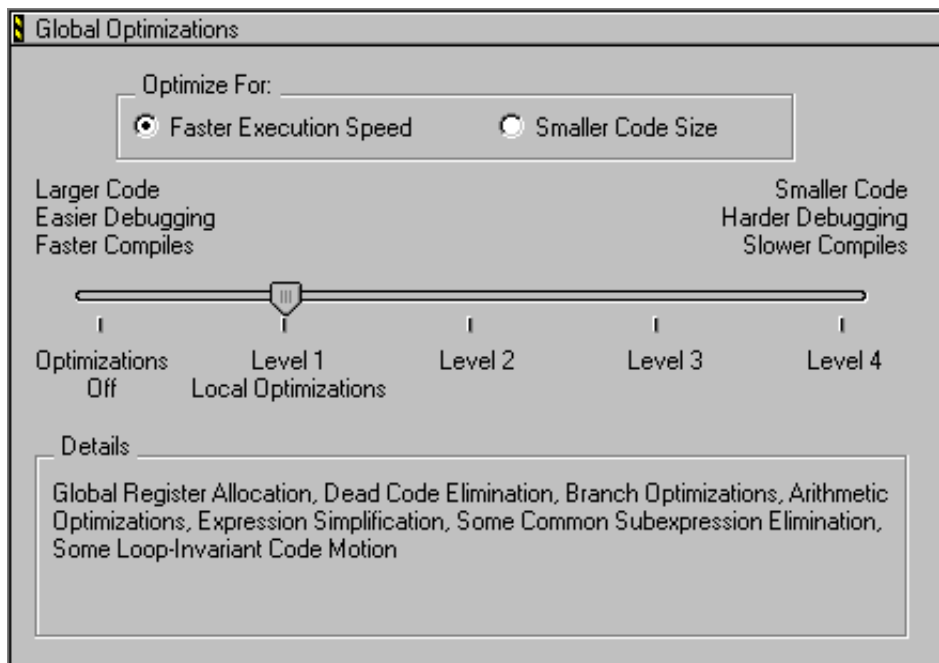
オフの場合、プロセッサの浮動小数点レジスタは使われません。コンパイラはランタイムルーチンを呼び出して浮動小数点演算を処理します。

注意： 今回のバージョンではこのオプションは無視されます。浮動小数点レジスタは常に使用されます。

Global Optimizations

Global Optimization 設定パネル ([図 9.7](#)) でコンパイラによる最適化の方法やレベルをコントロールします。

図 9.7 Global Optimizations 設定パネル



以下のオプションがあります。

[Optimize For](#)

[Optimization Level Slider](#)

Optimize For

CodeWarrior IDE によるコードの最適化の基準を指定します。

Faster Execution Speed

このオプションがオンの場合、オブジェクトコードの実行速度が上昇しますが、コードのサイズは大きくなります。

Smaller Code Size

このオプションがオンの場合、オブジェクトコードのサイズが小さくなりますが、実行速度は遅くなります。

Optimization Level Slider

コードに適用する最適化のレベルを指定します。コードの最適化をしない (Optimization off) または 4 つの最適化レベルのいずれかを選択できます。レベルが高くなるほど、多くの最適化が適用されます。

『Details』欄には適用される最適化の種類が表示されます。[表 9.2](#) にその内容を示します。最適化についての詳細は『[Dreamcast コードの最適化](#)』(p59) をご覧ください。

表 9.2 SH 最適化レベル

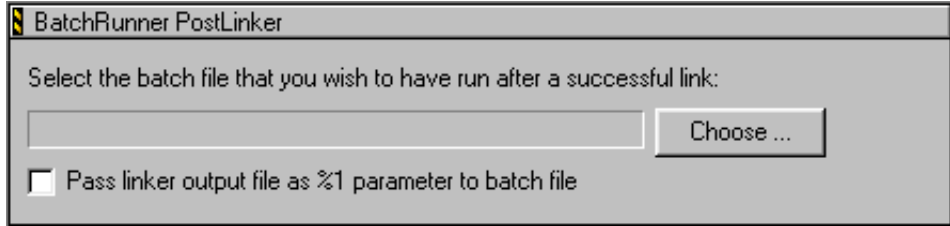
レベル	効果	デバッグ
0	最適化しない	安全
1	大域レジスタ割り当て (Global Register Allocation) 未使用コードの除去 (Dead Code Elimination) ループ不変コードの移動 (Loop Invariant Code Motion) 分岐の最適化 (Branch Optimization) 演算の最適化 (Arithmetic Optimizations) 式の簡略化 (Expression Simplification)	危険
2	共通部分式の削除 (Common Sub-Expression Elimination) インストラクションスケジュール (Instruction Scheduling) ディレイスロットの充てん (Delay-slot Filling) コピーによる伝搬 (Copy and Expression Propagation) ピープホール最適化 (Peephole Optimization)	危険
3	デッドストアの除去 (Dead Store Elimination) 強度の削減 (Strength Reduction) ライフタイムレジスタ割り当て (Lifetime Based Register Allocation) ループアンローリング (Loop Unrolling) ループの変換 (Loop Transformations) ライフレンジの分割 (Life Range Splitting) ベクトル化 (Vectorization)	危険
4	最適化の繰り返し (Optimizations are repeated)	n/a

注意：最適化レベルが 0 の場合、『Smart inlining』は使えません。

BatchRunner PostLinker

BatchRunner Postlinker 設定パネル ([図 9.8](#)) について説明します。

図 9.8 BatchRunner Postlinker 設定パネル



この設定パネルでプロジェクトのビルド後に実行するバッチファイルを指定します。
[Choose] ボタンをクリックして .bat ファイルを選択してください。

リンカ出力ファイルの名前をパラメータとしてバッチファイルへ渡すには、『Pass linker output file as %1 parameter to batch file』チェックボックスをオンにしてください。

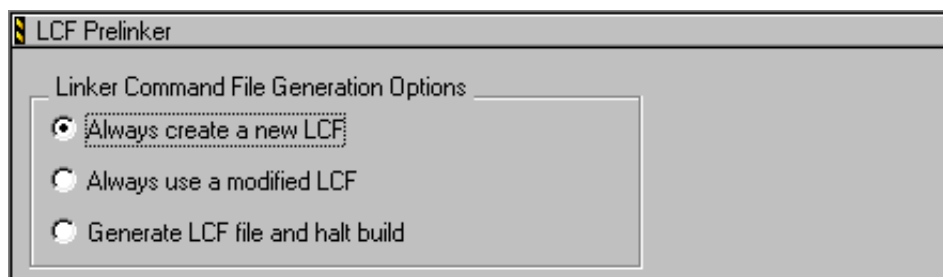
注意： Batchrunner PostLinker がターゲットのポストリンカとして選択されていなければ、指定したバッチファイルは実行されません。ポストリンカのオプションは [Target Settings](#) 設定パネルで修正できます。

LCF Prelinker

LCF Prelinker 設定パネル ([図 9.9](#)) で LCF (Linker Command File) Generator の挙動を設定します。プリリンカは、コードとデータと最終実行可能ファイルへ正確にマップするために必要なリンカ命令を含む LCF を生成します。

SH LCF Generator、および LCF についての詳細は『[リンカコマンドファイルを作成](#)』(p66) をご覧ください。

図 9.9 LCF Prelinker 設定パネル



Linker Command File Generation Options

オプションはラジオボタンで設定します。選択できるオプションは一度に 1 つだけです。

Always create a new LCF

プロジェクトをビルドするたびに新しい LCF を生成します。新しい LCF は既存の LCF を上書きします。

Always use a modified LCF

新しい LCF を生成しません。このオプションを選択することは、プリリンカを選択しないことと等価です。

Generate LCF file and halt build

新しい LCF を生成しますが、リンカがファイルを読み取る前にビルドを停止します。これにより、生成された LCF を自分の手で編集して変更することができます。

警告！ LCF を編集した後、Target Settings 設定パネルで LCF プリリンカをオフにするか、または上記の『Always use a modified LCF』オプションを選択してください。これを行わないと、次回プロジェクトをビルドしたときに SH LCF Generator がファイルを上書きします。

Section Mappings

Section Mappings 設定パネル ([図 9.10](#)) でリンクのためにセクションのレイアウトをマップします。リンクコマンドファイルがない場合、リンクはコードをリンクするための情報を Section Mappings 設定パネルから取得します。

Dreamcast SDK ライブラリに Section Mappings 設定パネルを使うことはできません。マップの定義と特殊なリンクの設定が複雑であるためです。代わりにリンクコマンドファイルを使ってください。しかし、Section Mappings 設定パネルは Dreamcast SDK ライブラリを使わないプログラムのデバッグに役立ちます。

図 9.10 Section Mappings 設定パネル

Section Mappings			
Segment	Address	Max. Size	Contains Sections
CODE	10000000		.text
DATA	20000000		.data .sdata .sbss .bss

以下のオプションがあります。

[Segment](#)
[Max Size](#)

[Address](#)
[Contains Sections](#)

Segment

セクションを含むセグメントのユーザーが定義した名前です。

Address

セグメントの開始アドレスです。

Max Size

セグメントの最大サイズです。このサイズを超過したとき、リンクはエラーを発します。

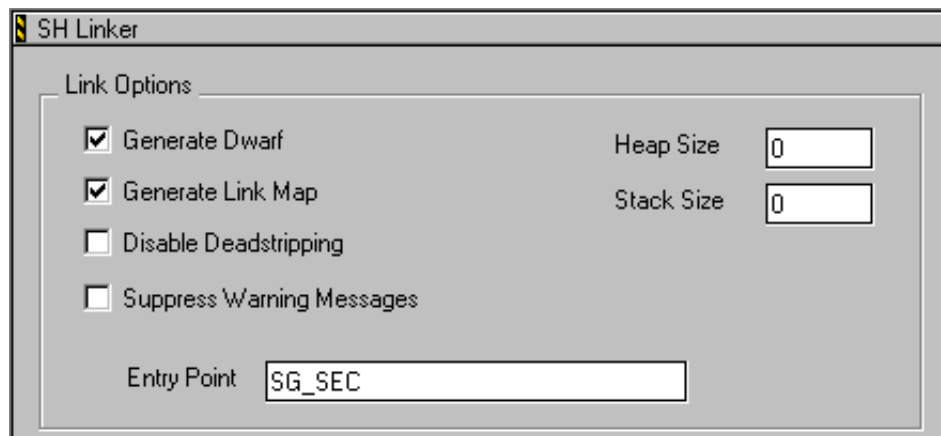
Contains Sections

セグメントに含まれるセクションの名前です。

SH Linker

SH Linker 設定パネル ([図 9.11](#)) でオブジェクトコードを実行可能ファイルやライブラリなどの最終形式へリンクする方法をコントロールします。

図 9.11 SH Linker 設定パネル



以下のオプションがあります。

[Generate Dwarf Info](#)

[Heap Size](#)

[Generate Link Map](#)

[Stack Size](#)

[Disable Deadstripping](#)

[Suppress Warning Messages](#)

[Entry Point](#)

Generate Dwarf Info

このオプションがオンの場合、リンカはデバッグ情報を生成します。リンカは、コンパイラが生成したデバッグ情報を組み込みます。デバッグ情報がなければプログラムをデバッグすることはできません。

Generate Link Map

このオプションがオンの場合、リンカはリンクマップを生成します。オフの場合、生成しません。

リンクマップは、出力ファイルのオブジェクトや関数の定義がどのファイルに含まれているのかを表示します。各オブジェクトや関数のアドレス、メモリ上にあるセクションのメモリマップ、リンカが生成したシンボルも表示します。

Disable Deadstripping

『Disable Deadstripping』チェックボックスオンの場合、リンカは未使用コードを除去(デッドストリップ)しません。

Suppress Warning Messages

『Suppress Warning Messages』チェックボックスで警告を表示するか否かを指定します。

Heap Size

使用しません。ヒープサイズは Dreamcast SDK ライブラリで規定されています。

Stack Size

使用しません。スタックサイズは Dreamcast SDK ライブラリで規定されています。

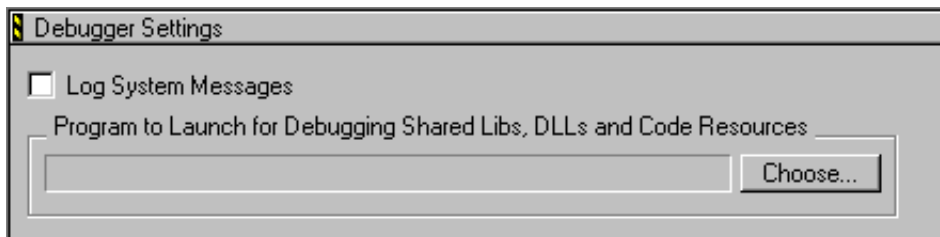
Entry Point

最初に呼び出される関数の名前です。この値はプログラムの開始アドレスです。

Debugger Settings

Debugger Settings 設定パネル ([図 9.11](#)) はこのバージョンでは使用しません。

図 9.12 Debugger Settings 設定パネル



第 10 章 Dreamcast の C/C++

この章では Dreamcast 用 CodeWarrior コンパイラについて説明します。

[Dreamcast の数値フォーマット](#)

[Dreamcast の呼び出し規約](#)

[Dreamcast コードの最適化](#)

[Dreamcast の C/C++ の注意](#)

この章ではコード生成に関する項目（フロントエンドコンパイラ、コンパイラ / リンカのエラー、C++ コードのサイズなど）は説明しません。これらについては他の CodeWarrior マニュアルで解説しています。[表 10.1](#) をご覧ください。

表 10.1 その他のコンパイラ / リンカのマニュアル

項目	参照マニュアル
CodeWarrior の C/C++ 言語の実装	C Compilers Reference
C/C++ Language 設定パネル、 C/C++ Warnings 設定パネル	C Compilers Reference、『C/C++ コンパイラオプションの設定』の章
C++ コードのサイズの調整	C Compilers Reference、『C++ とエンベデッドシステム』の章
コンパイラプラグマの使用	C Compilers Reference、『プラグマとシンボル』の章
ビルド、コンパイルするファイル、 エラーレポートの処理	IDE User Guide、『コンパイルとリンク』の章
特定のエラーに関する情報	Error Reference
インラインアセンブラ	インラインアセンブラと組み込み関数
Dreamcast 用アセンブラ	SH プロセッサのマニュアル

注意：ここで説明する機能のいくつかは、実際にはフロントエンドコンパイラに実装されているものです。しかし、フロントエンド、バックエンドのどちらのコンパイラに実装されているかは問題ではありません。結局は 1 つのコンパイラです。

Dreamcast の数値フォーマット

ここでは CodeWarrior C/C++ コンパイラが Dreamcast プロセッサの整数、浮動小数をどのように実装しているのかについて説明します。整数型についての詳細は `limits.h`、浮動小数型については `float.h` を参照してください

以下の内容について説明します。

[Dreamcast の整数フォーマット](#)

[Dreamcast の浮動小数点フォーマット](#)

Dreamcast の整数フォーマット

Dreamcast 用バックエンドコンパイラは整数のサイズを変更することはできません。short int のサイズは常に 2 バイトです。int または long int は常に 4 バイトです。

[表 10.2](#) に Dreamcast 用コンパイラの整数型のサイズと範囲をまとめます。

表 10.2 Dreamcast の整数型

型	サイズ	範囲
bool	8 ビット	true または false
char	8 ビット	-128 ~ 127
unsigned char	8 ビット	0 ~ 255
short	16 ビット	-32,768 ~ 32,767
unsigned short	16 ビット	0 ~ 65,535
int	32 ビット	-2,147,483,648 ~ 2,147,483,647
unsigned int	32 ビット	0 ~ 4,294,967,295
long	32 ビット	-2,147,483,648 ~ 2,147,483,647
unsigned long	16 ビット	0 ~ 4,294,967,295
long long	サポートされていない	サポートされていない

Dreamcast の浮動小数点フォーマット

[表 10.3](#) に Dreamcast 用コンパイラの浮動小数点型のサイズと範囲をまとめます。

注意： double は現在 float として実装されています。

表 10.3 Dreamcast の浮動小数点型

型	サイズ	範囲
float	32 ビット	1.17549e-38 ~ 3.40282e+38

Dreamcast の呼び出し規約

CodeWarrior は Hitachi の ABI (Application Binary Interface) 仕様に完全に準拠します。CodeWarrior が生成するコードは、Hitachi SH コンパイラが生成するコードと互換性があります。この 2 つのコンパイラでビルドしたコードを一緒にリンクすることができます。

Hitachi の ABI については「SH Series C/C++ Compiler User's Manual」(日立制作所発行) で詳しく説明されています。

Dreamcast コードの最適化

Dreamcast 用 CodeWarrior に特有の最適化について説明します。Global Optimization 設定パネルで最適化の設定を行います。詳細は『[Global Optimizations](#)』(p50) を参照してください。

[大域レジスタ割り当て](#)

[ループ不変コードの移動](#)

[未使用コードの除去](#)

[デッドストアの除去](#)

[共通部分式の削除](#)

[インストラクションスケジュール](#)

[ディレイスロットの充てん](#)

[コピーによる伝搬](#)

[ピープホール最適化](#)

[強度の削減](#)

[ライフタイムレジスタ割り当て](#)

[ループアンローリング](#)

大域レジスタ割り当て

同時に使われることのない複数の変数を、1 つのレジスタに割り当てる最適化です。以下の例では、コンパイラは `i` と `j` を同じレジスタに割り当てます。

```
short i;
int j;
```

```
for (i=0; i<100; i++) { MyFunc(i); }  
for (j=0; j<100; j++) { MyFunc(j); }
```

しかし関数内に以下のような行があれば、コンパイラは `i` と `j` が同時に使われると判断し、別のレジスタに割り当てます。

```
MyFunc (i + j);
```

レジスタ割り当てによってコードのサイズは小さくなります。実行速度は変わりません。

大域レジスタ割り当てをオンにしてデバッグを行うと、同じレジスタを共有する変数がおかしく見えることがあります。上記の例では、`i` と `j` は常に同じ値になります。`i` が変わると、同様に `j` も変わります（逆も同様です）。

大域レジスタ割り当ては [SH Processor](#) 設定パネルのレベル 1 を選択するとオンになります。デバッグに影響を与えるため、デバッグ時にはレベル 0 を選択してください。

ループ不変コードの移動

ループ内で変化しない演算をループの外へ移動します。これによりループの速度が上がります。この最適化を行うとオブジェクトコードの速度が上がります。

未使用コードの除去

論理的に実行されない、または参照されないとコンパイラが判断したステートメントを除去します。オブジェクトコードのサイズが小さくなります。

デッドストアの除去

変数が再度代入される前に使われていなければ、その変数への代入を除去します。オブジェクトコードのサイズが小さくかつ高速になります。

共通部分式の削除

似たような複数の冗長式を 1 つの式で置き換えます。例えば `a * b * c + 10` という式を使うステートメントが 2 つ連続する場合、コンパイラはこの式を 1 度だけ計算するオブジェクトコードを生成し、その結果を 2 つの式に適用します。

オブジェクトコードのサイズが小さくかつ高速になります。

インストラクションスケジュール

実行速度を上げる最適化です。各インストラクションがお互いの実行を遅れさせないように、プロセッサのインストラクションを並べ換えます。

ディレイスロットの充てん

コンパイラはディレイスロットインストラクションのディレイスロットを充てんします。例えば以下は：

```
JSR  
NOP
```

JSR はディレイスロットインストラクションですが、この場合ディレイスロットは非アクティブです。JSR の後にインストラクションを追加することによって、ディレイスロット機能を利用できます。

```
JSR  
instruction
```

ディレイスロットの充てんがアクティブの場合、JSR インストラクションのディレイスロットに `instruction` が置かれます。ディレイスロットにあるインストラクションは JSR よりも先に実行されます。

コピーによる伝搬

同一の変数が複数に出現する場合、その出現箇所を 1 つにします。オブジェクトコードのサイズが小さくなり、速度も上がります。

ピープホール最適化

コードの小さいセクションに適用される局所的な最適化です。最適化コードのセクションの実行速度が速くなります。

強度の削減

ループの中にある複数のインストラクションを別のインストラクションで置き換えます。オブジェクトコードが大きくなりますが実行速度が速くなります。

ライフタイムレジスタ割り当て

1 つのルーチン内の異なる変数がある場合、それらの変数が同一のステートメントで使用されていなければ同一のレジスタに割り当てます。オブジェクトコードの実行速度が速くなります。

ループアンローリング

最適化レベルが 3 または 4 のとき、コンパイラはループアンローリングを実行します。アンローリングファクタは 2 に設定されています。ループに 20 以上のインストラクションがない場合、ループはアンローリングされません。

ループアンローリングを無効にするには、ソースコードに以下のプラグマを追加してください。

```
#pragma opt_unroll_loops off
```

Dreamcast のプラグマ

Dreamcast 用 CodeWarrior コンパイラがサポートするプラグマについては『C Compilers Reference』で説明しています。オンラインのマニュアルは CodeWarrior Documentation フォルダにあります。

[表 10.4](#) に Dreamcast 用開発環境ではサポートされていないプラグマを示します。このリスト以外の、『C Compilers Reference』で定義されているプラグマを使用できます。

表 10.4 Dreamcast ではサポートしていないプラグマ

code_seg	define_section	disable_registers
interrupt	longlong	longlong_enums
no_register_coloring	peephole	register_coloring
scheduling	section	stack_cleanup
use_fp_instructions		

プラグマ `opt_unroll_count` と `opt_unroll_instr_count` に対する Dreamcast のデフォルト値は 2 と 40 です。プラグマについては『C Compilers Reference』を参照してください。

Dreamcast の C/C++ の注意

標準 C++ ライブラリへアクセスするためには、`MSLCppDC.lib` ライブラリをプロジェクトへ追加します。これは Metrowerks の標準 C++ ライブラリです。

今回のバージョンは C++ をサポートしていますが、以下の制限があります。

[例外処理](#)

[ストリームと IO クラス](#)

[その他の制限](#)

例外処理

プログラムコード内で例外のキャッチ/スローを行うためには、[例 10.1](#) のような行をリンカコマンドファイルに追加して、例外テーブルを定義してください。

```
.exception
```

必ず上記の行を `$INCLUDE` セクションのリストへ追加してください。また例外テーブルそのもののために新しいデータセグメントを作る必要があります。

例 10.1 リンカコマンドファイルで例外テーブルを作成

```
$INCLUDE
{
    IP
    DSGLH
```

```

    DSGLE
    .exception    # Needed for C++
}

$SEGMENT DATA 0x8C040000 R
{
    # Include the exception table index.

    ALIGN(0x4)
    *(.exception) # Needed for C++

    ALIGN(0x4)
    *(.exceptlist) # Needed for C++
}

```

例外処理を使い、かつ例外ハンドラがある場合、フレームポインタとして R14 レジスタが使われます。例えば組込み関数 `__alloca` を使ったとき、R14 レジスタがフレームポインタとして使用されます。上記条件の場合、ソースコード（アセンブラ、インラインアセンブラ）で、R14 を利用しないようにする必要があります。組込み関数の詳細は『[組込み関数のリスト](#)』（p84）をご覧ください。

ストリームと IO クラス

今回のバージョンはストリームも I/O クラスもサポートしません。しかし C++ ストリーム関数、`cout` を真似して、出力をデバッガのコンソールウィンドウへ送ることができます。

プログラム中の次のような行を

```
cout << "Hello"
```

`mw_pr()` 関数を使って以下のようにします。

```
mw_pr("Hello");
```

`mw_pr()` 関数を使ってライブラリをプロジェクトへ追加します。このライブラリは Dreamcast Support フォルダにあります。

その他の制限

今回のバージョンでは、以下の C++ の機能をサポートしていません。

テンプレート定義の外にあるメンバテンプレート / 入れ子クラスのテンプレートメンバはサポートしない

メンバテンプレート変換関数はサポートしない

メンバテンプレートフレンドはサポートしない

テンプレート引数はサポートしない

「エクスポートされた」テンプレートはサポートしない

第 11 章 Dreamcast 用リンカについて

Dreamcast 用リンカの仕組みについて説明します。

[未使用コードの削除 \(デッドストリップ\)](#)

[リンク順](#)

[リンカコマンドファイル](#)

[関数順の変更](#)

未使用コードの削除 (デッドストリップ)

Dreamcast ライブラリおよび CodeWarrior C/C++ コンパイラで作成したライブラリでは、使用されるオブジェクトだけがリンクされるプログラムに含まれます。アセンブラを含むライブラリやその他の C/C++ コンパイラで作成したライブラリでは、最低でも 1 度参照されるオブジェクトがリンクされるプログラムに含まれます。1 度も参照されないオブジェクトファイルは常に見捨てられます。

参照されないコードやデータのセクションを最終的なアプリケーションに残したい場合、[\\$INCLUDE](#) を使います。このリンカコマンドファイルの疑似命令は、リンカによるデッドストリップを防ぎます。`$include` 疑似命令の詳細は『[リンカコマンドファイルのフレームワーク](#)』(p68) をご覧ください。SH Linker 設定パネルの『Disable Deadstripping』オプションでデッドストリップを無効にできます。『[SH Linker](#)』(p54) をご覧ください。

Dreamcast 用リンカは、CodeWarrior C/C++ コンパイラで作成したライブラリから未使用コードやデータを削除 (デッドストリップ) します。その他のアセンブラ再配置可能ファイルや他のコンパイラで作成した C/C++ オブジェクトファイルからは削除しません。

リンク順

リンク順はプロジェクトウィンドウの Overlays ビューで指定します。リンク順の設定については『IDE User Guide』を参照してください。

ライブラリのリンク順はとても重要です。ステーションナリでは Dreamcast SDK ライブラリの正しいリンク順がデフォルトで指定されています。ステーションナリを使わない場合、以下の順番でライブラリをリンクしてください。

```
strt1.obj.elf
strt2.obj.elf
systemid.obj.elf
toc.obj.elf
sg_sec.obj.elf
sg_arejp.obj.elf
sg_areus.obj.elf
```

```
sg_areec.obj.elf  
sg_are00.obj.elf  
sg_are01.obj.elf  
sg_are02.obj.elf  
sg_are03.obj.elf  
sg_are04.obj.elf  
sg_ini.obj.elf  
aip.obj.elf  
zero.obj.elf
```

このライブラリの後に、他のライブラリやソースファイルを置いてください。

ヒント： Dreamcast 用リンカはプロジェクトに含まれる実行可能ファイルを無視します。実行可能ファイルを残しておくことで逆アセンブルも可能になり、便利です。ビルドが成功したら、プロジェクトウィンドウ左側のタッチ列にあるチェックマークが消えます。新しいファイルができたからです。ビルドが失敗したとき、IDE は実行可能ファイルを見つけないことができず、メッセージを表示して停止します。

リンカコマンドファイル

Section Mappings 設定パネルだけではリンクプロセスのコントロールを十分にできない場合、リンカコマンドファイル (Linker Command File) を使います。リンカコマンドファイルはプログラムとデータセクションの配置を定義します。

[リンカコマンドファイルを作成](#)

[リンカコマンドファイルのフレームワーク](#)

[リンカコマンドファイルの文法](#)

リンカコマンドファイルを作成

リンカコマンドファイルの作成は困難であるため、ユーザーフレンドリーなツールを開発しました。LCF Generator はプロジェクトを管理して LCF を作成します。LCF Generator はプリリンカとして IDE に統合されており、Target Settings 設定パネルで選択します。

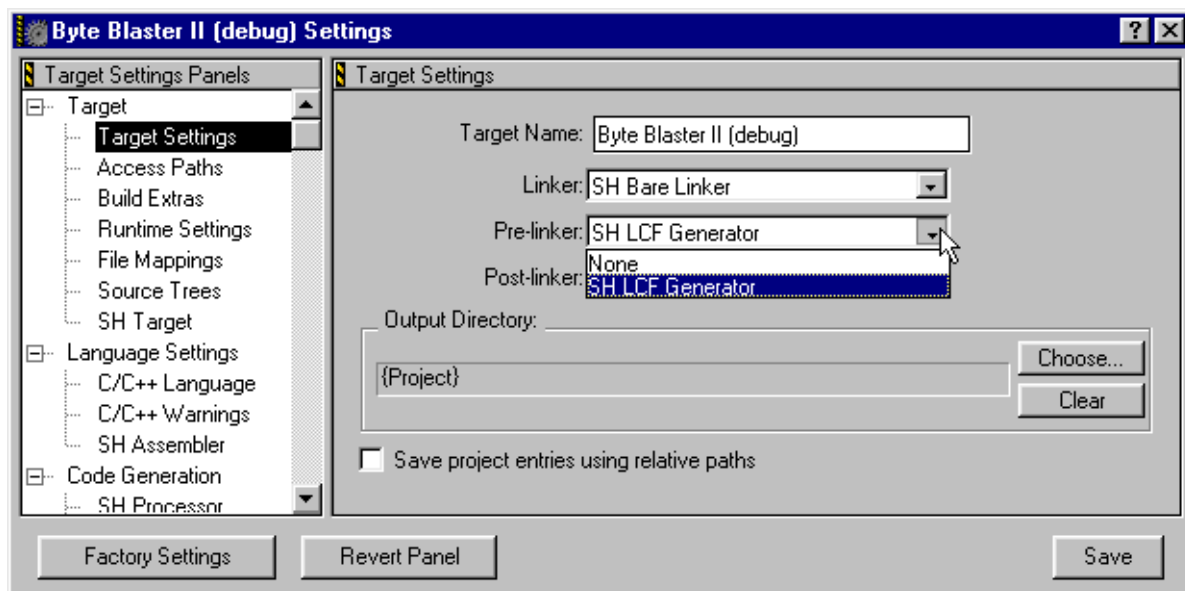
LCF Generator の設定については『[LCF Prelinker](#)』(p53) をご覧ください。LCF Generator がどのようにオーバーレイのオーバーレイを管理するかについては『[オーバーレイ](#)』(p99) をご覧ください。

以前のバージョンからアプレットグレードした場合、またはプロジェクトステーションリを使用していない場合、以下の手順に従って LCF Generator を設定してください。

1. プリリンカとして LCF Generator を選択する

プロジェクトの Target Settings 設定パネルを開き、『Pre-linker』ポップアップメニューで『LCF Generator』を選択してください ([図 11.1](#))。

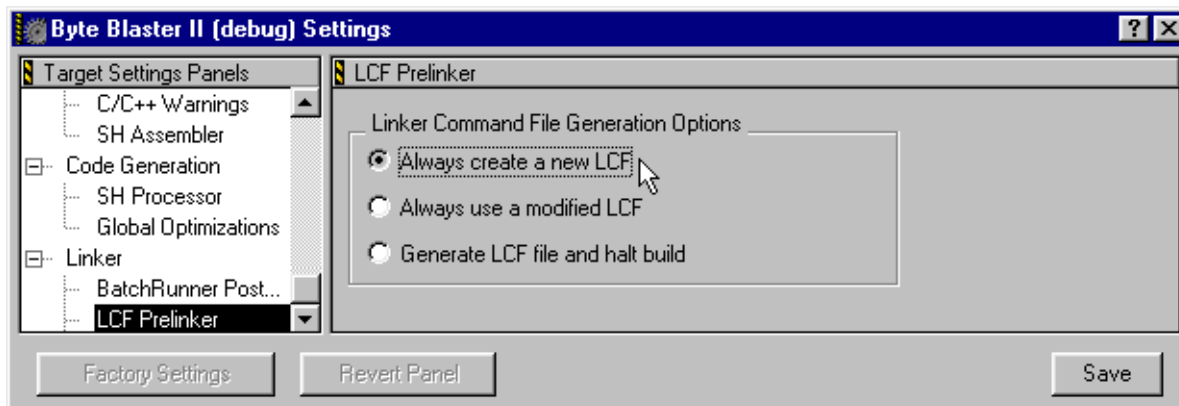
図 11.1 Pre-linker ポップアップメニューで LCF Generator を選択



2. 新しい LCF を作成するように LCF Generator を設定する

LCF Prelinker 設定パネルで 『Always create a new LCF』 を選択してください (図 11.2)

図 11.2 Always Create a New LCF を選択



3. ターゲット設定を保存する
4. 空のファイルに linker.lcf という名前を付けてプロジェクトに追加する

LCF Generator はリンカコマンド命令を既存の linker.lcf ファイルへ書き込みます。プリリンカはこのファイルをプロジェクトへ追加しないため、自分で追加します。

- a. 空のテキストファイルを作成する

File メニューの 『New Text File』 を選択してください。

b. 空のファイルを linker.lcf として保存する

File メニューの『Save As...』を選択して、linker.lcf と名前を付けてプロジェクトのあるディレクトリへ保存してください。

c. プロジェクトへ linker.lcf ファイルを追加する

Project メニューの『Add Files...』を選択して、linker.lcf ファイルの名前をダブルクリックしてください。作成した linker.lcf ファイルが見つからない場合、プロジェクトディレクトリを検索しているか、また『Files of type』ポップアップメニューで『All Files』を選択しているかを確認してください。

注意： CodeWarrior for Dreamcast R2 ステーションナリを使わずにプロジェクトを作成した場合、.lcf ファイルをプロジェクトへ追加する前に .lcf リンカコマンドファイル用のファイルマッピングを作成しなくてはなりません。ファイルマッピングを以下に示します。

ファイルタイプ： [blank]
拡張子： .lcf
フラグ： [blank]
コンパイラ： None

リンカコマンドファイルのフレームワーク

リンカコマンドファイルは、プログラムのコンポーネントをリンクする順番と、リンカに必要なシンボルとオーバーレイヘッダの両方を定義するリンカ命令を含みます。

3 つの構造体 (includes、segments、overlays) が Dreamcast 環境の LCF フレームワークを構成しています。

1. Includes

`$INCLUDE` 疑似命令でデッドストリップに関係なくコードセクションを定義します。以下の例では、一度も直接参照されないコードを含めて IP セクションの全コードをインクルードします。

```
$INCLUDE
{
    IP
}
```

2. Segments

`$SEGMENT` 疑似命令でセグメントの境界と内容を定義します。リンカやランタイムライブラリが参照するシンボルも定義します。

以下は BSS_USER セグメントを定義する例です。

```
$SEGMENT BSS_USER
{
    __START_B = . ;
```

```
*(.bss)
GROUP(ROOT)(.bss)
GROUP(ROOT)(B)
__END_B = .;
}
```

- a. `__START_B` と `__END_B` にメモリ位置を割り当てる

Sega ランタイムライブラリはこれらのシンボルを `B` セクションの位置へのインデックスとして使用します。

- b. 残りの `B` と `.bss` セクションを追加する

`B` と `.bss` は同じデータ型でそれぞれ未初期化データとゼロ初期化データを含んでいます。`.bss` と `B` の両方をインクルードすることにより、Main Applications グループ内の全ファイルの未初期化データセクションがインクルードされます。ファイルをコンパイルしたのが Sega、CodeWarrior コンパイラのどちらでも関係ありません。

3. Overlays

`$OVERLAY` 疑似命令でアプリケーション内のオーバーレイの内容、境界、ヘッダを定義します。

以下はオーバーレイを定義する例です。

```
$OVERLAY
```

リンカコマンドファイルの文法

CodeWarrior リンカは以下の疑似命令をサポートします。[] 中のパラメータはオプションです。定義すべき変数は斜体で表示されています。

[\\$segment](#)

[\\$INCLUDE](#)

[\\$REF_INCLUDE](#)

[\\$OUTPUT_NAME](#)

[アラインメント](#)

[演算オペレータ](#)

[代入](#)

[コメント](#)

[ロケーションカウンタ](#)

[セクションマッピング](#)

[シンボル](#)

[データの書き込み](#)

\$segment

説明 セグメントの境界と内容を定義します。

プロトタイプ `$segment name [baseaddress] [LENGTH length] [R]`
 {
 symbols and sections go here
 }

注意 未指定のベースアドレスは、最後のセグメントの直後にセグメントを置くことを暗示します。

未指定の `length` は、無制限のセグメント長として解釈されます。

`R` オプションは ROM 用セクションを処理します。

セクションマッピングについては『[セクションマッピング](#)』(p73) をご覧ください。

例題 1 以下のセグメントのベースアドレスは `0x2F001050`、最大長は `0xC000` です。実際のセグメントのサイズが指定した長さよりも大きい場合、リンカはエラーを発生します。

すべてのファイルの `.text` セクションはこのセグメントにマップされます。

```
$segment CODE 0x2F001050 LENGTH 0xC000
{
    *(.text)
}
```

例題 2 以下のセグメントには最大長はなく、ROM セグメントとして処理されます。

すべてのファイルの `.data` セクションはこのセグメントにマップされます。

```
$segment DATA 0x30000000 R
{
    *(.data)
}
```

例題 3 以下のセグメントは最後のセグメントの直後のアドレスにマップされます。長さに制限はありません。

`foo.c` ファイルの `.data` セクションはこのセグメントにマップされます。

```
$segment BSS
{
    foo.c(.data)
}
```

\$OVERLAY

説明 オーバーレイを定義します。

```

プロトタイプ      $OVERLAY name [base address]
                   [LENGTH length] > filename
{
overlay header, symbols, and sections go here
}

```

注意 未指定のベースアドレスは、最後のセグメントの直後にセグメントを置くことを暗示します。

未指定の length は、無制限のセグメント長として解釈されます。

オーバーレイそのものは `filename` ファイルへ書き込まれ、後にアプリケーションにロードされます。

初期化プロシージャを含め、オーバーレイについては『[オーバーレイ](#)』（p99）をご覧ください。

例題 以下のオーバーレイはアドレス 0x8C1000000 にマップされます。長さは無制限です。オーバーレイの内容は `overlay1.bin` という名前のファイルに書き込まれます。

```
$overlay Overlay1 0x8C100000 > overlay1.bin
{
    # declare variables for overlay
    _Overlay1SegmentStart = .;
    _Overlay1SegmentTextStart = .;
    #other variables omitted

    # write overlay header at beginning
    WRITEB ...
    # actual overlay header omitted

    # begin section mapping
    GROUP(Overlay1)(.text)
    _Overlay1SegmentTextEnd = .;
    _Overlay1SegmentDataStart = .;
    GROUP(Overlay1)(.data)
    GROUP(Overlay1)(.rodata)
    _Overlay1SegmentDataEnd = .;
    # other sections omitted.
}
```

\$INCLUDE

説明 シンボルまたはセクションを出力にインクルードします

```
プロトタイプ    $INCLUDE
                {
                  _my_unreferenced_function_name
                  .myunreferencedsection
                }
```

注意 `$include` を使った場合、リンカは直接参照されないシンボルやセクションを削除（デッドストリップ）しません。名前のあるシンボルやセクションは強制的に出力にリンクされます。

`$REF_INCLUDE`

説明 シンボルまたはセクションを含むファイルが参照されている場合のみ、出力内のシンボルまたはセクションをインクルードします。

プロトタイプ

```
$REF_INCLUDE
{
    .rodata
}
```

注意 `$REF_INCLUDE` 疑似命令を使うと、直接参照されていないシンボルやセクションの削除（デッドストリップ）を防ぐことができます。名前のあるシンボルやセクションは出力へ含まれます。

`$OUTPUT_NAME`

説明 ファイル名の拡張子を変えずに、出力ファイルの名前を変更します。

プロトタイプ

```
$OUTPUT_NAME
{
    my_output_name
}
```

アラインメント

`$SEGMENT` 定義の内部で使用可能な 2 つの異なるアラインメント指定子を使うことにより、リンカコマンドファイルでアラインメントを強制できます。

例題 1 `ALIGN` はセクショングループの最初のセクションに影響します。以下のコードでは、第 1 の `.text` セクションだけが 8 バイト境界にアラインされます。

```
ALIGN(0x8)
*(.text)
```

例題 2 `ALIGNALL` はセクショングループに関係するファイルの最初のセクションに影響します。以下のコードでは、それぞれの第 1 の `.text` セクションだけが 8 バイト境界にアラインされます。

```
ALIGNALL(0x8)
*(.text)
```

演算オペレータ

C の標準演算オペレータを使用できます。オペレータは常に左側に置きます。詳細は『C Compilers Reference』をお読みください。


```
_sizeof_data = _end_data - (_start_data);
```

代入

グローバルシンボルを作成して、標準代入オペレータを使ってそれらにアドレスを代入することができます。

```
_symbolicname = expression;
```

代入ステートメントはセミコロンで終わります。代入は式の最初でのみ使用できます。以下は不当です。

```
_sym1 + _sym2 = _sym3;           #illegal
```

例題 シンボル `_my_stack_symbol` は `__stack_begin` の値に代入されます。シンボルは CodeWarrior リンカによって内部で生成されます。これは、CodeWarrior が生成したものと違うシンボルを使用している場合にコードの移植をするときに便利です。

```
_my_stack_symbol = __stack_begin;
```

コメント

コメントにはポンド記号 (#) を使います。パーサーはポンド記号の右にある文字を無視します。

例題 以下は有効なコメントです。

```
# This is a one-line comments
*(.data)    # This is a partial-line comment
```

ロケーションカウンタ

ピリオド (.) は、常に出力ロケーションの現在の位置を保持します。ピリオドは常にセグメント内のロケーションを参照するため、常に `$SEGMENT` ステートメントの中にあります。

ピリオドはシンボルが許可されているところの、式の右辺で使用できます。

例題 `_start_data` は `.data` セクションの開始アドレスを含みます。`_end_data` は `.data` の終点のアドレスを含みます。

```
$segment DATA 0x30000000 LENGTH 0 R
{
    _start_data = .;
    *(.data)
    _end_data = .;
}
```

セクションマッピング

マッピング指定子は、マッピングのセクションに関係のあるファイルをリンカに指定します。

例題 1 filename(section) は 1 つのファイルのセクションをマップします。次の例は、foo.c ファイルの .text セクションをマップします。

```
foo.c(.text)
```

例題 2 *(section) は一般的なマッピングを指定します。アスタリスクをワイルドカードシンボルとして考えると、プロジェクトの全ファイルからのセクションを取得することができます。次の例は、あらゆるファイルの .data セクションをマップします。

```
*(.data)
```

例題 3 GROUP (overlayname)(section) は、プロジェクトウィンドウで定義された特定のオーバーレイグループのファイルを指定します。次の例は OverlayOne という名前のオーバーレイに含まれる全ファイルの .data および .rodata セクションをマップします。

```
GROUP (OverlayOne)(.data)
GROUP (OverlayOne)(.rodata)
```

注意：『Main Application』オーバーレイ用のリンカコマンドファイル名は ROOT です。

例題 4 *(.bss) などの一般的なマッピングの後に、GROUP(ROOT)(.bss) または foo.c(.bss) などの特定のマッピングを続けることができます。次の例は「ROOT 以外のすべての .bss をここに入れて、ROOT の .bss をその後に入れる」と指定しているようなものです。

```
*(.bss)
GROUP (ROOT)(.bss)
```

例題 5 FUNCTION(functionname) は、ある関数を含むセクションからセクショングループを作成します。頻繁に相互参照を行う関数をメモリ内で近くにマップして、キャッシュヒットを減らすことができます。これによりコードのパフォーマンスが上昇します。

次の例は指定された関数を含むセクションを、8k 境界にアラインされたセクショングループにまとめます。

```
SEGMENT OPTIMIZED 0x80001000
{
    ALIGN(0x2000)
    FUNCTION(_function_foo)
    FUNCTION(_function_bar)
    FUNCTION(_function_foo2)
}
```

シンボル

リンカコマンドファイルで定義するシンボルは、アンダースコアで始まり、かつ \$SEGMENT の中で定義されなくてはなりません。

```
_start_data = .;
_address = 0x2544000;
```

データの書き込み

WRITEn ステートメントを使うと、セグメント内のどこにでもデータを書き込むことができます。WRITEn ステートメントの後にある WRITEn ステートメントは、ALIGN ステートメントでアライン可能な、単一の結合データセクションです。

例題 1 3 種類の WRITEn ステートメントがあります。WRITEB は 1 バイトを書き込みます。WRITEH は 2 バイトハーフワードを書き込みます。WRITEW は 4 バイトワードを書き込みます。

```
ALIGN(0x8)
WRITEB 0xFF;
WRITEH 0xDDFF;
WRITEW 0xBBCCDDFF;
```

例題 2 WRITEn ステートメント内で演算と表現式を使うことができます。

```
WRITEW _address_of_data + _sizeof_data;
WRITEW SIZEOF(DATA);
WRITEW ADDR(DATA);
WRITEW ADDR(DATA) + SIZEOF(DATA);
```

関数順の変更

メモリ内でどのようにコードを配置するかによって、実行速度に若干の違いが生じます。メモリの配置を整理して、頻繁に相互参照を行う関数を互いに近い位置に置くことができます。キャッシュのヒット率を向上することによってメモリアクセスがより高速になり、プログラムの速度もあがります。

コードのロードおよびランタイムで実行される方法を最適化するために、Codescape Profiler を使用してリンクオーダー (.lor) ファイルを作成します。.lor ファイルは深度優先走査によりアプリケーション内の関数の順番を替えます。つまり関数 main() はルーチン A() と B() を呼び出し、ルーチン A() は C() と D() を呼び出すとします。深度優先順は main()、A()、C()、D()、B() となります。

以下は .lor ファイルの作成、および使用する手順です。

1. Codescape でコードをプロファイルする

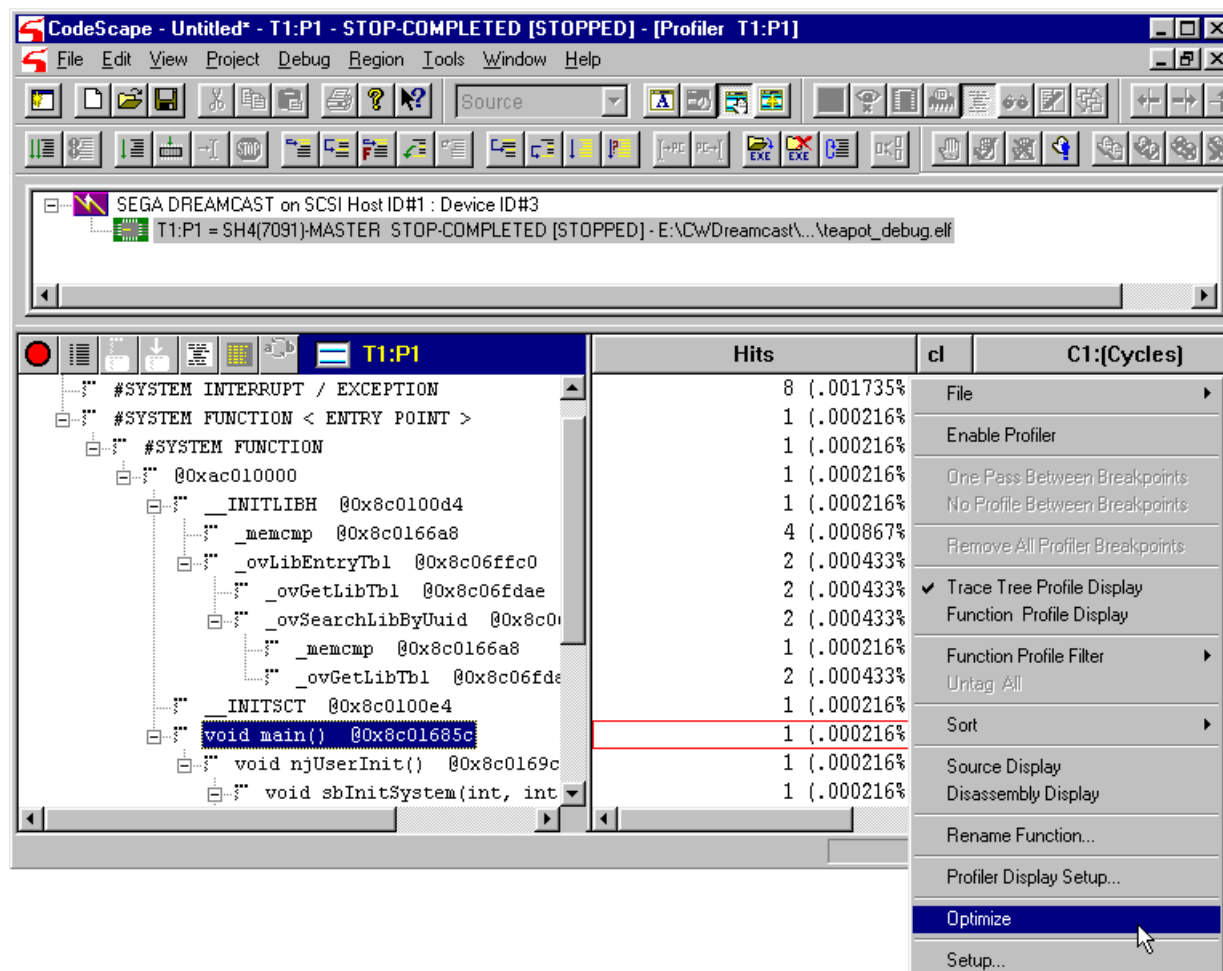
Codescape については付属のユーザーガイドをお読みください。以下は操作の概要です。

- Codescape デバッガにプログラムをロードする
- プロファイラを起動してプログラムを実行する
- プログラムの十分なプロファイルデータが取れたらプロファイラを停止する

2. Codescape を使って .lor ファイルを作成する

コードのプロファイルを終わったら、プロファイラウィンドウ内でマウスを右クリックしてください。表示されるメニューから『Optimize』を選択してください ([図 11.3](#))。これで Profiler Instruction Cache Optimizer ダイアログが現れます。

図 11.3 プロファイル後に『Optimize』を選択



Profiler Instruction Cache Optimizer ダイアログで [Select All] ボタンをクリックしてください。次に [Optimize...] ボタンをクリックしてファイルダイアログを開きます。 .lor ファイルに名前を付けてプロジェクトフォルダへ保存してください。

3. .lor ファイルをプロジェクトへ追加する

CodeWarrior へ戻り、プロジェクトへ .lor ファイルを追加してください。

注意： CodeWarrior for Dreamcast R2 ステーションナリを使わずにプロジェクトを作成した場合、 .lor ファイルをプロジェクトへ追加する前に .lor リンクオーダーファイル用のファイルマッピングを作成しなくてはなりません。ファイルマッピングを以下に示します。

ファイルタイプ： [blank]
拡張子： .lor

フラグ : [blank]
コンパイラ : None

4. SH LCF Generator プリリンカを起動する

プリリンカはプロジェクトに `.lor` ファイルがあることを自動的に検知します。リンカコマンドファイルは `.lor` ファイルと同じ順番に関数をマップします。

注意 : CodeWarrior for Dreamcast R2 ステーションナリを使わずにプロジェクトを作成した場合、`.lcf` ファイルをプロジェクトへ追加する前に `.lcf` リンカコマンドファイル用のファイルマッピングを作成しなくてはなりません。ファイルマッピングを以下に示します。

ファイルタイプ : [blank]
拡張子 : .lcf
フラグ : [blank]
コンパイラ : None

5. プロジェクトをコンパイルする

Codescape Profiler のインストラクションキャッシュオプティマイザーに従ってプログラムの関数の順番を変えることができました。

第 12 章 インラインアセンブラと組み込み関数

この章では CodeWarrior がサポートするインラインアセンブリ言語プログラミングについて説明します。アセンブラインストラクションの詳細は SH プロセッサ付属のマニュアルをご覧ください。

[インラインアセンブラを使う](#)

[アセンブラ疑似命令](#)

[組み込み関数](#)

[インラインアセンブラのニーモニック](#)

インラインアセンブラを使う

コンパイラがサポートするアセンブリ言語プログラミングについて説明します。

[インラインアセンブラの文法](#)

[ラベルの使用](#)

[コメントの使用](#)

[レジスタの使用](#)

インラインアセンブラの文法

C/C++ ソースファイルへアセンブリ言語ステートメントを追加する方法は 2 つあります。

第 1 の方法を例 12.1 に示します。asm 修飾子で関数内のすべてのステートメントをアセンブリ言語に指定しています。関数内の局所変数を asm 修飾子で関数内の局所変数を定義することができます。

例 12.1 asm で関数を定義

```
asm int MyAsmFunction (void)
{
    /* Local variable definitions */
    /* Assembly language instructions */
}
```

第 2 の方法を例 12.2 に示します。『インライン』アセンブリ言語インストラクションを提供するために、asm 修飾子をステートメントとして使用しています。

言い換えると、同一の関数定義でアセンブリ言語ステートメントと標準の C/C++ ステートメントを組み合わせて使うことができます。しかし、インライン `asm` ステートメントはその関数の局所変数を参照することはできません。

例 12.2 asm を使うインラインアセンブラ

```
int MyInlineAsmFunction(void)
{
    /* Local variable definitions and C/C++ statements */
    asm { /* Assembly language instructions */ }
    /* Local variable definitions and C/C++ and asm {} statements */
}
```

C/C++ コンパイラに `asm` キーワードを認識させるためには、C/C++ language 設定パネルの『ANSI Keywords Only』オプションをオフにしてください。この設定パネルについては『C Compilers Reference』をご覧ください。

ビルトインアセンブラはすべての標準 SH アセンブラインストラクションをサポートします。

ヒント： インラインアセンブラの代わりに、コンパイラがサポートする組み関数を使って、1つの関数に数行のアセンブラコードを追加することもできます。[『組み関数』\(p84\)](#) をご覧ください。

アセンブラ関数を書くときは、以下のことに注意してください。

アセンブリ言語の関数、および `asm` ブロックを含む関数に対して最適化が行われることがあります。これはコンパイラによる最適化の設定によって異なります。最適化レベルについては[『Global Optimizations』\(p50\)](#) をお読みください。

`.set noreorder` 疑似命令を使って、アセンブラの最適化を無効にすることができます。詳細は[『.set』\(p82\)](#) をお読みください。

すべてのステートメントは以下のようなラベルか、

[*LocalLabel* :]

以下のようなインストラクションになる。

((*instruction* ≠ *directive*) [*operands*])

各ステートメントは改行で終わる。

コンパイラは、インラインアセンブラのブロック内で初期化された変数を認識しない。

アセンブラ疑似命令、組み関数、レジスタはケースセンシティブではない (大文字小文字を区別しない)。以下の 2 つは同じである。

```
ADD      R2, R4          // OK
add      r2, r4          // OK
```


16 進定数は C 言語の形式を使う。

```
0x123ABC    // OK
$123ABC     // ERROR
H'123ABC    // ERROR
```

ラベルの使用

局所変数として宣言していない識別子ならば、ラベルにすることができます。ラベルはコロンで終わってください。同じ行の中で、ラベルの後にインストラクションを続けることはできません。以下に例を示します。

```
x1:  ADD    R2,R3    // ERROR
x2:           // OK
      ADD    R2,R3    // OK
```

例 12.3 ラベルを使う

```
extern void foo(void);

int foo() {
    asm
    {
        MOVA    foo_addr, R0;
foo_addr:
        .data.w 0;
        .data.l foo;
    }
}
```

コメントの使用

C/C++ 言語形式のコメントを使用できます。しかしセミコロン (;) は使えません。

```
ADD    R2,R4    // OK
ADD    R2,R4    /* OK */
ADD    R2,R4    ; ERROR
```

レジスタの使用

[例 12.4](#) では、アセンブラステートメントが関数内に組込まれています。インラインアセンブラステートメントから `i` を直接参照するには、変数を `register` として入力します。

例 12.4 レジスタを使う

```
int foo3(int register i){
    asm{
        MOV i,R1;
        ADD 1, R1;
```

```
        MOV R1, R4;
    }
    return i;
}
```

ステータスレジスタ

インラインアセンブラを通じてステータスレジスタを読み取り、またセットすることができます ([例 12.5](#))。

例 12.5 ステータスレジスタを使う

```
/* Get status register */
static inline unsigned int get_sr(void)
{
    register unsigned int sr = 0;

    asm
    {
        STC SR, sr
    };
    return sr;
}
/* Set status register */
static inline void set_sr(unsigned int sr)
{
    register int value = sr;

    asm
    {
        LDC value, SR
    };
}
```

アセンブラ疑似命令

現時点では、Dreamcast 用アセンブラ特有の疑似命令は 2 つあります。

.set

プロトタイプ .set [reorder | noreorder]

これを使うと、アセンブラはインストラクションスケジュール最適化を実行してコードのパフォーマンスを上昇させます。これは、インストラクション同士が互いの実行を妨げないようにプロセッサインストラクションを並べ換える最適化です。

.set のデフォルト設定は最適化レベルによって決定されます。最適化レベルが 0 または 1 のとき、デフォルトは .set noreorder です。それ以上のレベルでは、デフォルトは .set reorder です。最適化レベルの設定については『[Global Optimizations](#)』(p50) をご覧ください。

[例 12.6](#) では、foo() 呼び出しのためのディレイスロットで $x + y$ を計算しています。JSR インストラクションの後に故意に ADD インストラクションを置いているため、.set noreorder を使ってインストラクションシーケンスを並べ替えないようにしています。

例 12.6 .set を使う

```
asm int ADD (int x, int y)
{
    .set    noreorder
    // y = x + y
    // call foo
    MOV.L   foo, R0;
    JSR     @R0;
    // return x + y;
    ADD     R4, R5;
    MOV     R5, R0;
}
```

.frame

プロトタイプ .frame

.frame 疑似命令はスタックフレームを生成するためのエピローグとプロローグを生成します。インラインアセンブラインストラクションでもスタックフレームを作成できますが、.frame を使う方が簡単です。関数が以下に該当する場合、スタックフレームが必要です。

他の関数を呼び出す。

局所変数を宣言している。

[例 12.7](#) は .frame の文法です。RTS インストラクションをコメントにしていることに注意してください。.frame を使うと、コンパイラは自動的に RTS を生成します。

例 12.7 .frame を使う

```
asm int foo()
{
    .frame
    MOV 12, R0;
    // RTS;
    ADD 1, R0;
}
```

組み込み関数

コンパイラはインラインアセンブラインストラクションを生成することのできる組み込み関数を提供しています。コンパイラは組み込み関数をインラインアセンブラインストラクションへ翻訳するため、組み込み関数の実行速度は普通の関数よりも高速です。インラインアセンブラ文法を使って `asm` ブロックでオペコードを指定するよりも、組み込み関数を使う方が便利です。

注意： 組み込み関数のサポートは ANSI C/C++ の規格には含まれません。これは CodeWarrior コンパイラの拡張機能です。

コンパイラはソースコードで組み込み関数を見つけると、即座にその関数呼び出しをアセンブラインストラクションに置き換えます。結果として、最終的なオブジェクトコードでは関数呼び出しが発生しなくなります。最終的なコードには、組み込み関数に対応するアセンブラインストラクションが含まれます。

[組み込み関数のリスト](#)

[Hitachi SH C コンパイラ互換の組み込み関数](#)

組み込み関数のリスト

[表 12.1](#) に CodeWarrior プロジェクトで利用可能な組み込み関数を示します。

表 12.1 組み込み関数

__abs	__labs
fabs	fsqrt
alloca	memcpy
__abs	

説明 絶対値を示します。

例題

```
int Intrinsic_abs (int i)
{
    int j;
    j = __abs(i);
    return j;
}
```

[__labs](#)

説明 long の絶対値を示します。

例題

```
long Intrinsic_labs (long i)
{
```

```
    long j;  
    j = __labs(i);  
    return j;  
}
```

__fabs

説明 浮動小数点の絶対値を示します。

例題

```
double Intrinsic_fabs (double i)  
{  
    double j;  
    j = __fabs(i);  
    return j;  
}
```

__fsqrt

説明 平方根を示します。

例題

```
float Intrinsic_fsqrt (float i)  
{  
    float j;  
    j = __fsqrt(i);  
    return j;  
}
```

__alloca

説明 動的なスタック割り当てを示します。

例題

```
void Intrinsic_alloca(void)  
{  
    int i;  
    short *x = (short *)__alloca(1024*sizeof(short));  
    for (i = 0; i < 1024; i++) x[i] = i;  
}
```

__memcpy

説明 メモリコピーを示します。

例題

```
typedef struct s  
{  
    int i1;  
    int i2;  
    int i3;  
}
```

```
        s i;

s s1;
s s2;

void Intrinsic_memcpy(s si)
{
    s2 = si;
    __memcpy(&s1, &si, sizeof(s));
}
```

Hitachi SH C コンパイラ互換の組み込み関数

[表 12.2](#) の組み込み関数は Hitachi SH C コンパイラの組み込み関数関数との互換性を提供します。

表 12.2 Hitachi SH C コンパイラ互換の組み込み関数

set_fpscr	sub4
get_fpscr	mtrx4mul
fipr	mtrx4muladd
ftrv	mtrx4mulsub
ftrvadd	ld_ext
ftrvsub	st_ext
add4	

set_fpscr

説明 システム / コントロールレジスタの浮動小数点ユニットへ 32 ビット値を書き込みます。

プロトタイプ void set_fpscr(int cr);

注意 なし。

get_fpscr

説明 システム / コントロールレジスタの浮動小数点ユニットの値を読みます。

プロトタイプ int get_fpscr();

注意 なし。

戻り値 get_fpscr() は FPSCR の値を返します。

fipr

説明 単精度の浮動小数点ベクトル vect1 と vect2 の内積を計算します。

プロトタイプ `float fipr(float vect1[4], float vect2[4]);`

注意 なし。

戻り値 `fipr()` は `vect1` と `vect2` の内積を `float` 型で返します。

`ftrv`

説明 単精度浮動小数点ベクトル (`vect1`) を拡張レジスタ (MTX) に保存されている 4x4 マトリクスで乗算します。結果は `vect2` に保存されます。

`vect2 = vect1 x MTX`

プロトタイプ `void ftrv(float vect1[4], float vect2[4]);`

注意 この関数を使う前に、[ld_ext](#) 組込み関数を使って拡張レジスタに MTX のデータをロードしておく必要があります。

`ftrvadd`

説明 単精度浮動小数点ベクトル (`vect1`) を拡張レジスタ (MTX) に保存されている 4x4 マトリクスで乗算します。次にその結果に `vect2` を加算します。結果は `vect3` に保存されます。

`vect3 = (vect1 x MTX) + vect2`

プロトタイプ `void ftrvad(float vect1[4], float vect2[4],
float vect3[4]);`

注意 この関数を使う前に、[ld_ext](#) 組込み関数を使って拡張レジスタに MTX のデータをロードしておく必要があります。

`ftrvsub`

説明 単精度浮動小数点ベクトル (`vect1`) を拡張レジスタ (MTX) に保存されている 4x4 マトリクスで乗算します。次にその結果から `vect2` を減算します。結果は `vect3` に保存されます。

`vect3 = (vect1 x MTX) - vect2`

プロトタイプ `void ftrvsub(float vect1[4], float vect2[4],
float vect3[4]);`

注意 この関数を使う前に、[ld_ext](#) 組込み関数を使って拡張レジスタに MTX のデータをロードしておく必要があります。

`add4`

説明 単精度浮動小数点ベクトル、`vect1` と `vect2` を加算します。結果は `vect3` に保存されます。

```
vect3 = vect1 + vect2
```

プロトタイプ `void add4(float vect1[4], float vect2[4],
float vect3[4]);`

注意 なし。

sub4

説明 `vect1` から単精度浮動小数点ベクトル、`vect2` を減算します。結果は `vect3` に保存されます。

```
vect3 = vect1 - vect2
```

プロトタイプ `void sub4(float vect1[4], float vect2[4],
float vect3[4]);`

注意 なし。

mtrx4mul

説明 単精度浮動小数点 4x4 マトリクス (`mtrx1`) を拡張レジスタ (MTX) に保存されている 4x4 マトリクスで乗算します。結果は `mtrx2` に保存されます。

```
mtrx2 = mtrx1 x MTX
```

プロトタイプ `void mtrx4mul(float mtrx1[4][4],
float mtrx2[4][4]);`

注意 この関数を使う前に、[Id_ext](#) 組み込み関数を使って拡張レジスタに MTX のデータをロードしておく必要があります。

mtrx4muladd

説明 単精度浮動小数点 4x4 マトリクス (`mtrx1`) を拡張レジスタ (MTX) に保存されているマトリクスで乗算します。次に別のマトリクス `mtrx2` が加算され、結果は `mtrx3` に保存されます。

```
mtrx3 = (mtrx1 x MTX) + mtrx2
```

プロトタイプ `void mtrx4muladd(float mtrx1[4][4],
float mtrx2[4][4],
float mtrx3[4][4]);`

注意 この関数を使う前に、[Id_ext](#) 組み込み関数を使って拡張レジスタに MTX のデータをロードしておく必要があります。

mtrx4mulsub

説明 単精度浮動小数点 4x4 マトリクス (mtrx1) を拡張レジスタ (MTX) に保存されているマトリクスで乗算します。次に別のマトリクス mtrx2 が減算され、結果は mtrx3 に保存されます。

mtrx3 = (mtrx1 x MTX) - mtrx2

プロトタイプ void mtrx4mulsub(float mtrx1[4][4],
float mtrx2[4][4],
float mtrx3[4][4]);

注意 この関数を使う前に、[ld_ext](#) 組込み関数を使って拡張レジスタに MTX のデータをロードしておく必要があります。

ld_ext

説明 4x4 マトリクスのデータを浮動小数点拡張レジスタへロードします。

プロトタイプ void ld_ext(float mtrx[4][4]);

注意 なし。

st_ext

説明 浮動小数点拡張レジスタを読み取り、マトリクスデータを 4x4 マトリクスへ保存します。

プロトタイプ void st_ext(float mtrx[4][4]);

注意 なし。

インラインアセンブラのニーモニック

インラインアセンブラのインストラクションは通常のアセンブラと若干異なります。

[インラインアセンブラの特殊インストラクション](#)

[インラインアセンブラニーモニックのリスト](#)

インラインアセンブラの特殊インストラクション

インラインアセンブラには特殊なインストラクションがあります。以下に示すインストラクションは、コンパイラによってマシン命令のシーケンスへ展開されます。以下のフォーマットで記述します。

"mnemonic", "format"

Rn へ定数を移動する

"MOV.L", "w,Rn"

ラベルの有効なアドレスをロードする

```
"MOVA",    "l,=R0"
```

定数プールからロードする

```
"MOV.L",   "l,Rn"
```

インラインアセンブラ疑似命令

```
"_set",    ""
```

```
"_unset",  ""
```

コードストリーム内ヘデータを埋め込む

以下のインラインインストラクションを使って、コードストリーム内ヘデータを埋め込むことができます。

```
".data.b"   "u"
```

```
".data.w"   "v"
```

```
".data.l"   "w"
```

特殊インストラクションの例

[例 12.8](#) は特殊インストラクションの使用例です。ここでは `MOV.L` インストラクションで定数 `12345678` を `R1` ヘロードしています。

例 12.8 特殊インストラクションを使う

```
asm int foo1() {  
    MOV.L    12345678,R1;  
    RTS;  
    NOP;  
}
```

[例 12.9](#) では、コンパイラが実際に特殊インストラクションをマシン命令へ展開しています。

例 12.9 コンパイラが特殊インストラクションを展開

```
        _foo4:  
0xD101          mov.l    @(4,pc),r1  
0x000B          rts  
0x0009          nop  
0x0000          .data.w  0x0000  
0x614E          .data.w  0x614E  
0x00BC          .data.w  0x00BC
```

この特殊インストラクションを使わない場合、コードに埋め込まれた定数へアクセスするためのディスプレースメントとアラインメントの計算を自分で行わなくてはなりません。特殊インストラクションを使わなければ、[例 12.10](#) のようなコードを書く必要があります。

以下の `MOV.L` インストラクションでは、コンパイラはアクセスしているデータと同じサイズにディスプレースメントを増幅しています（この場合、`long` の 4）。

例 12.10 特殊インストラクションを使わない

```
asm int foo2() {
    MOV.L @(1,PC), R0;
    RTS;
    NOP;
    .data.w 0;
    .data.l 12345678;
}
```

インラインアセンブラニーモニックのリスト

[表 12.3](#) は CodeWarrior コンパイラがサポートするインラインアセンブラインストラクションのリストです。通常のアセンブラインストラクションと似ていますが、スラッシュ（/）をアンダースコア（_）に変えなくてはなりません。サポートしていないインストラクションは灰色で表示し、『非サポート』と示します。

表 12.3 インラインアセンブラのニーモニック

ニーモニック	フォーマット	サポート
"ADD "	"i ,Rn "	
"ADD "	"Rm ,Rn "	
"ADDC "	"Rm ,Rn "	
"ADDV "	"Rm ,Rn "	
"AND "	"i ,R0 "	
"AND "	"Rm ,Rn "	
"AND.B"	"i,@(R0,GBR)"	非サポート
"BF "	"1 "	
"BF_S "	"1 "	
"BRA "	"m "	
"BRAF "	"Rn "	
"BSR"	"m "	非サポート
"BSRF "	"Rn "	
"BT "	"1 "	
"BT_S "	"1 "	
"CLRMAC "	" "	

ニーモニック	フォーマット	サポート
"CLRS "	" "	
"CLRT "	" "	
"CMP_EQ "	"i ,R0 "	
"CMP_EQ "	"Rm ,Rn "	
"CMP_GE "	"Rm ,Rn "	
"CMP_GT "	"Rm ,Rn "	
"CMP_HI "	"Rm ,Rn "	
"CMP_HS "	"Rm ,Rn "	
"CMP_PL "	"Rn "	
"CMP_PZ "	"Rn "	
"CMP_STR "	"Rm ,Rn "	
"DIV0S "	"Rm ,Rn "	
"DIV0U "	" "	
"DIV1 "	"Rm ,Rn "	
"DMULS.L "	"Rm ,Rn "	
"DMULU.L "	"Rm ,Rn "	
"DT "	"Rn "	
"EXTS.B "	"Rm ,Rn "	
"EXTS.W "	"Rm ,Rn "	
"EXTU.B "	"Rm ,Rn "	
"EXTU.W "	"Rm ,Rn "	
"FABS "	"Fn "	
"FADD "	"Fm ,Fn "	
"FCMP_EQ "	"Fm ,Fn "	
"FCMP_GT "	"Fm ,Fn "	
"FCNVDS "	"Fn "	
"FCNVSD "	"Fn "	
"FDIV "	"Fm ,Fn "	
"FIPR "	"FVm ,FVn "	非サポート
"FLDI0 "	"Fn "	
"FLDI1 "	"Fn "	
"FLDS "	"Fn "	

ニーモニック	フォーマット	サポート
"FLOAD"	"Fn"	
"FMAC"	"F0, Fm, Fn"	
"FMOV"	"Fm, Fn"	
"FMOV.S"	"Fm, @Rn"	
"FMOV.S"	"@Rm, Fn"	
"FMOV.S"	"@Rm+, Fn"	
"FMOV.S"	"Fm, @-Rn"	
"FMOV.S"	"@(R0, Rm), Fn"	
"FMOV.S"	"Fm, @(R0, Rn)"	
"FMOV"	"Xm, @Rn"	非サポート
"FMOV"	"@Rm, Xn"	非サポート
"FMOV"	"@Rm+, Xn"	非サポート
"FMOV"	"Xm, @-Rn"	非サポート
"FMOV"	"@(R0, Rm), Xn"	非サポート
"FMOV"	"Xm, @(R0, Rn)"	非サポート
"FMOV"	"Xm, Xn"	非サポート
"FMOV"	"Xm, Dn"	非サポート
"FMOV"	"Dm, Xn"	非サポート
"FMUL"	"Fm, Fn"	
"FNEG"	"Fn"	
"FRCHG"	" "	
"FSCHG"	" "	
"FSQRT"	"Fn"	
"FSTS"	"Fn"	
"FSUB"	"Fm, Fn"	
"FTRC"	"Fn"	
"FTRV"	"XM, FVn"	非サポート
"JMP"	"@Rn"	
"JSR"	"@Rn"	非サポート
"LDC"	"Rn, GBR"	非サポート
"LDC"	"Rn, SR"	
"LDC"	"Rn, VBR"	

ニーモニック	フォーマット	サポート
"LDC "	"Rn , SSR "	
"LDC "	"Rn , SPC "	
"LDC "	"Rn , DBR "	
"LDC "	"Rn,Rb"	非サポート
"LDC.L "	"@Rn+,GBR"	非サポート
"LDC.L "	"@Rn+ , SR "	
"LDC.L "	"@Rn+ , VBR "	
"LDC.L "	"@Rn+ , SSR "	
"LDC.L "	"@Rn+ , SPC "	
"LDC.L "	"@Rn+ , DBR "	
"LDC.L "	"@Rn+,Rb"	非サポート
"LDS "	"Rn , FPSCR "	
"LDS "	"Rn , MACH "	
"LDS "	"Rn , MACL "	
"LDS "	"Rn , PR "	
"LDS "	"Rn , FPUL "	
"LDS.L "	"@Rn+ , FPSCR "	
"LDS.L "	"@Rn+ , MACH "	
"LDS.L "	"@Rn+ , MACL "	
"LDS.L "	"@Rn+ , PR "	
"LDS.L "	"@Rn+ , FPUL "	
"LDTLB "	" "	
"MAC.L "	"@Rm+ , @Rn+ "	
"MAC.W "	"@Rm+ , @Rn+ "	
"MOV "	"i , Rn "	
"MOV "	"Rm , Rn "	
"MOV.B "	"@(d8,GBR),R0"	非サポート
"MOV.B "	"@(d4,Rm) , R0 "	
"MOV.B "	"@(R0 , Rm) , Rn "	
"MOV.B "	"@Rm+ , Rn "	
"MOV.B "	"@Rm , Rn "	
"MOV.B "	"R0,@(d8,GBR)"	非サポート

ニーモニック	フォーマット	サポート
"MOV.B"	"R0, @(d4, Rm) "	
"MOV.B"	"Rm, @(R0, Rn) "	
"MOV.B"	"Rm, @-Rn "	
"MOV.B"	"Rm, @Rn "	
"MOV.W"	"@(d8, GBR), R0 "	非サポート
"MOV.W"	"@(d8, PC), Rn "	非サポート
"MOV.W"	"@(d4, Rm), R0 "	
"MOV.W"	"@(R0, Rm), Rn "	
"MOV.W"	"@Rm+, Rn "	
"MOV.W"	"@Rm, Rn "	
"MOV.W"	"R0, @(d8, GBR)"	非サポート
"MOV.W"	"R0, @(d4, Rm) "	
"MOV.W"	"Rm, @(R0, Rn) "	
"MOV.W"	"Rm, @-Rn "	
"MOV.W"	"Rm, @Rn "	
"MOV.L"	"@(d8, GBR), R0"	非サポート
"MOV.L"	"@(d8, PC), Rn "	
"MOV.L"	"@(d4, Rm), Rn "	
"MOV.L"	"@(R0, Rm), Rn "	
"MOV.L"	"@Rm+, Rn "	
"MOV.L"	"@Rm, Rn "	
"MOV.L"	"R0, @(d8, GBR)"	非サポート
"MOV.L"	"Rm, @(d4, Rn) "	
"MOV.L"	"Rm, @(R0, Rn) "	
"MOV.L"	"Rm, @-Rn "	
"MOV.L"	"Rm, @Rn "	
"MOVA"	"@(d8, PC), R0 "	
"MOVA"	<label>, R0	
"MOVCA.L"	"@R0, @Rn "	
"MOVT"	"Rn "	
"MUL.L"	"Rm, Rn "	
"MULS.W"	"Rm, Rn "	

ニーモニック	フォーマット	サポート
"MULU.W"	"Rm,Rn"	
"NEG"	"Rm,Rn"	
"NEGC"	"Rm,Rn"	
"NOP"	" "	
"NOT"	"Rm,Rn"	
"OCBI"	"@Rn"	
"OCBP"	"@Rn"	
"OCBWB"	"@Rn"	
"OR"	"i,R0"	
"OR"	"Rm,Rn"	
"OR.B"	"i,@(R0,GBR)"	非サポート
"PREF"	"@Rn"	
"ROTCL"	"Rn"	
"ROTCR"	"Rn"	
"ROTL"	"Rn"	
"ROTR"	"Rn"	
"RTE"	" "	
"RTS"	" "	
"SETS"	" "	
"SETT"	" "	
"SHAD"	"Rm,Rn"	
"SHAL"	"Rn"	
"SHAR"	"Rn"	
"SHLD"	"Rm,Rn"	
"SHLL"	"Rn"	
"SHLL2"	"Rn"	
"SHLL8"	"Rn"	
"SHLL16"	"Rn"	
"SHLR"	"Rn"	
"SHLR2"	"Rn"	
"SHLR8"	"Rn"	
"SHLR16"	"Rn"	

ニーモニック	フォーマット	サポート
"SLEEP"	" "	
"STC"	"GBR,=Rn"	非サポート
"STC"	"SR,=Rn"	
"STC"	"VBR,=Rn"	
"STC"	"SSR,=Rn"	
"STC"	"SPC,=Rn"	
"STC"	"DBR,=Rn"	
"STC"	"Rb,=Rn"	非サポート
"STC.L"	"G,@Rn+"	非サポート
"STC.L"	"SR,@Rn+"	
"STC.L"	"VBR,@Rn+"	
"STC.L"	"SSR,@Rn+"	
"STC.L"	"SPC,@Rn+"	
"STC.L"	"DBR,@Rn+"	
"STC.L"	"Rb,@Rn+"	非サポート
"STS"	"FPSCR,Rn"	
"STS"	"MACH,Rn"	
"STS"	"MACL,Rn"	
"STS"	"PR,Rn"	
"STS"	"FPUL,Rn"	
"STS.L"	"FPSCR,@-Rn"	
"STS.L"	"MACH,@-Rn"	
"STS.L"	"MACL,@-Rn"	
"STS.L"	"PR,@-Rn"	
"STS.L"	"FPUL,@-Rn"	
"SUB"	"Rm,Rn"	
"SUBC"	"Rm,Rn"	
"SUBV"	"Rm,Rn"	
"SWAP.B"	"Rm,Rn"	
"SWAP.W"	"Rm,Rn"	
"TAS.B"	"@Rn"	
"TRAPA"	"i"	

ニーモニック	フォーマット	サポート
"TST "	"i ,R0 "	
"TST "	"Rm ,Rn "	
"TST.B"	"i,@(R0,GBR)"	非サポート
"XOR "	"i ,R0 "	
"XOR "	"Rm ,Rn "	
"XOR.B"	"i,@(R0,GBR)"	非サポート
"XTRCT "	"Rm ,Rn "	

第 13 章 オーバーレイ

オーバーレイとは、プログラムそのものよりも小さいメモリ（たとえ仮想メモリがなくても）にプログラムをフィットさせるためのプログラミングテクニックです。

オーバーレイを使うためには、同時にロードする必要のないコードの塊にプログラムを分割します。コードの塊はコンパイルされて互いにリンクするオーバーレイになります。メインプログラムは個別のオーバーレイのロードを管理します。

例えば、ムービーを再生するオーバーレイ、メインメニューを表示するオーバーレイ、ゲームをプレイするためのオーバーレイがあるとします。これらのオーバーレイは別個に実行されるため、必要に応じてメモリへの出し入れが可能です。

以下の項目について説明します。

[オーバーレイプロジェクトを作成](#)

[オーバーレイの注意点](#)

オーバーレイプロジェクトを作成

CodeWarrior のプロジェクトウィンドウに『Overlays』タブがあります。これによりゲーム用のオーバーレイを簡単に作成することができます。

このチュートリアルでは 2 つのオーバーレイを含むプログラムをビルドし、テストします。

以下はオーバーレイを利用するプロジェクトを作成する手順です。

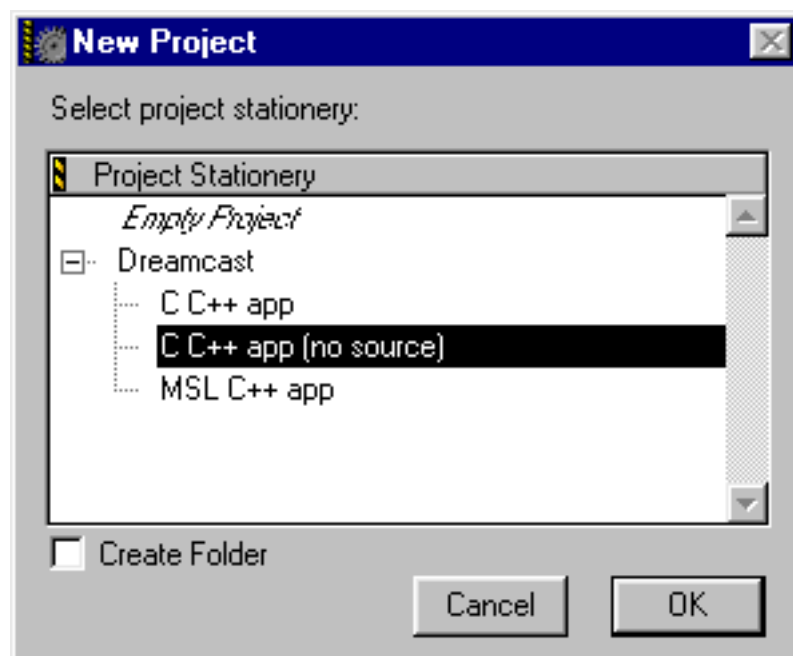
1. プロジェクトファイルをビルドする

a. 新規プロジェクトを作成する

CodeWarrior を起動して File メニューの『New Project』を選択してください。

『Dreamcast』の横の拡張ボタンをクリックすると表示される『C C++ app (no source)』ステーションナリを選択してください ([図 13.1](#))。新しいフォルダを作成する必要はありません。

図 13.1 オーバーレイチュートリアルのステーショナリを選択



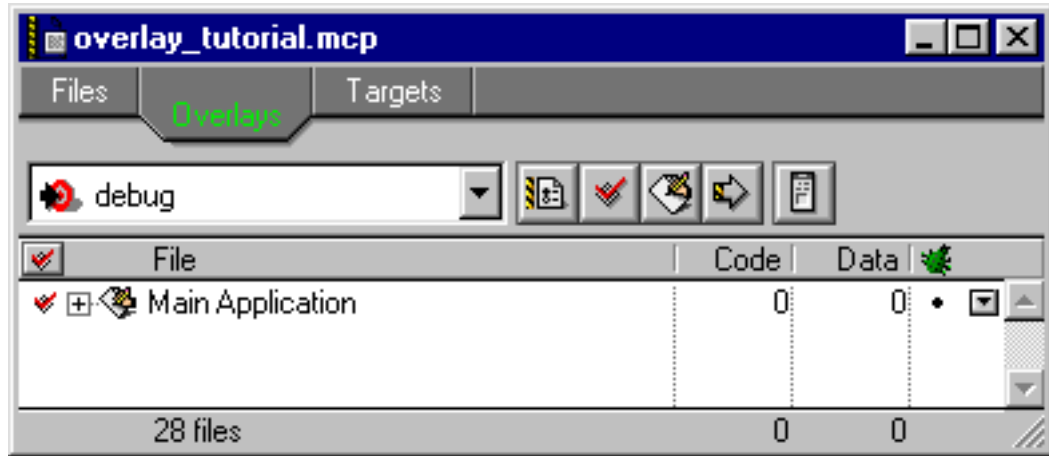
ファイルダイアログで以下の場所に Tutorial フォルダを指定します。

Examples\Overlay\

ファイル名に『overlay』と入力して [OK] ボタンをクリックしてください。
overlay.mcp というプロジェクトウィンドウが表示されます。

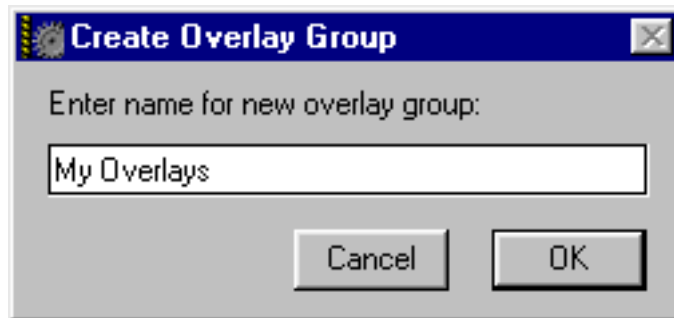
- b. プロジェクトへファイルを追加する
プロジェクトウィンドウの『Sources』グループへソースファイル (test.c、sbinit.c、mw_utils.c、njloop.c、overlay1.c、overlay2.c) を追加してください。ファイル名がリスト表示されます。
- c. オーバーレイを作成する
プロジェクトウィンドウの『Overlays』タブをクリックしてください ([図 13.2](#))。
『Main Application』を拡張してソースファイルを表示してください。

図 13.2 プロジェクトウィンドウの『Overlay』タブ



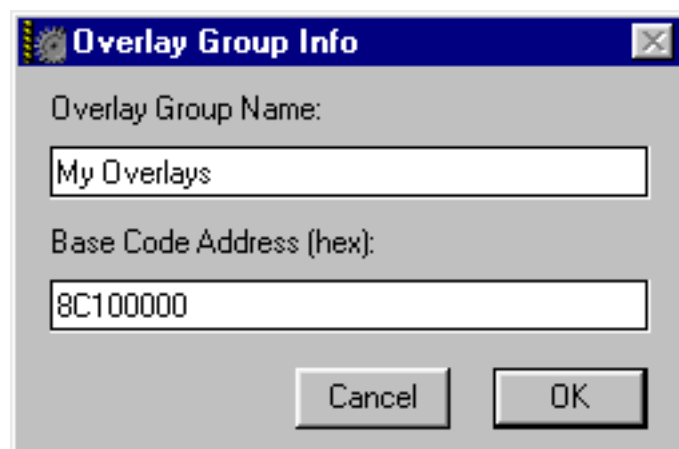
Project メニューの『Create Overlay Group』を選択してください。図 13.3 のダイアログが現れます。『My Overlays』と入力して [OK] ボタンをクリックしてください。これは新しいオーバーレイグループの名前です。

図 13.3 Create Overlay Group ダイアログ



オーバーレイグループの基底コードアドレスを指定します。『My Overlays』グループをダブルクリックすると図 13.4 のダイアログが現れます。

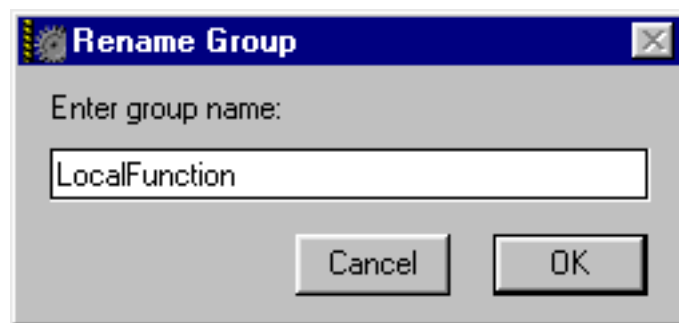
図 13.4 オーバーレイグループの基底コードアドレスを指定



『My Overlays』グループのオーバーレイコードは『Base Code Address』フィールドに入力した 16 進アドレス（例：8C100000）へロードされます。このアドレスを選んだ理由は、アプリケーションの他の部分が使用するメモリスペースから十分上の位置にあることがわかっているためです。

プロジェクトウィンドウで『My Overlays』グループを拡張表示してください。New Overlay という名前のオーバーレイがあります。その名前、またはアイコンをダブルクリックして Rename Group ダイアログを開きます（[図 13.5](#)）。テキストフィールドへ『LocalFunction』と入力して [OK] ボタンをクリックしてください。

図 13.5 Rename Group（オーバーレイ）ダイアログ



注意： オーバーレイの名前は、ハードディスクへ書き込むオーバーレイのファイル名でもあります。このソースコードでは、ファイル名によってオーバーレイをメモリへロードします。

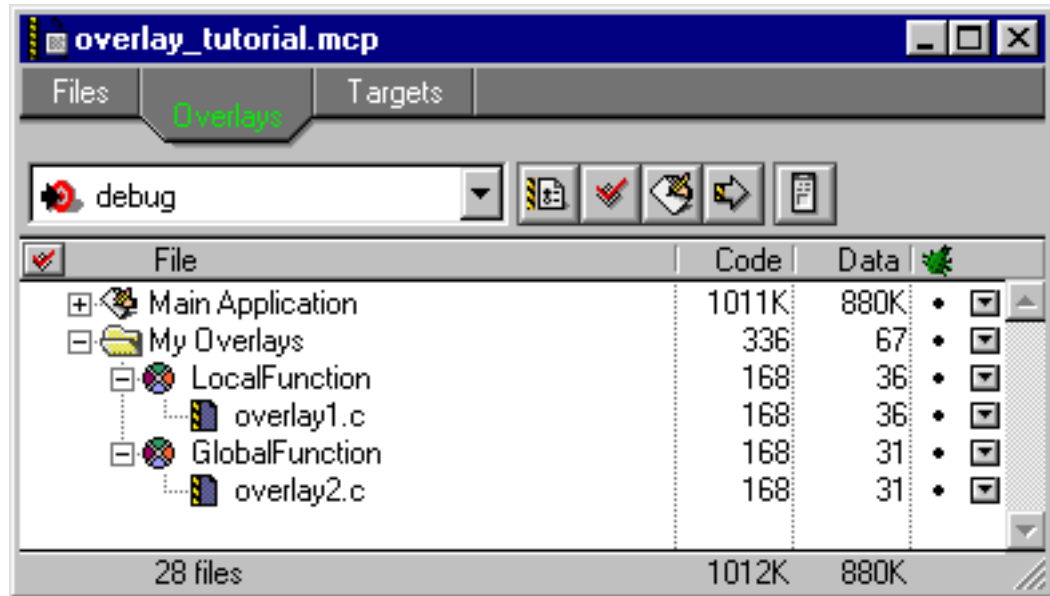
プロジェクトウィンドウのファイルリストで Overlay1.c ソースファイルを GlobalFunction オーバーレイの中へ置きます。Overlay1.c ソースファイルのアイコンを LocalFunction アイコンの下へドラッグしてください。

『My Overlays』グループの第 2 のオーバーレイを作成します。プロジェクトウィンドウで『My Overlays』をクリックしてから、Project メニューの『Create New Overlay』

を選択してください。テキストフィールドに『GlobalFunction』と入力して [OK] ボタンをクリックしてください。『GlobalFunction』 オーバーレイを拡張して『GlobalFunction』 アイコンの下に『overlay2.c』をドラッグしてください。

- d. オーバーレイの設定が完了
プロジェクトウィンドウは図 13.6 のようになります。

図 13.6 オーバーレイのあるプロジェクトウィンドウ



2. 他のプロジェクト設定を行う

アプリケーションをビルドするための設定を行います。

Edit メニューの『debug Settings』を選択して SH Target 設定パネルを開いてください。『File Name』フィールドを『overlay.elf』に変更してください。『Target Settings』を選択して Target Settings 設定パネルを開いてください。『Pre-linker』オプションを『SH LCF Generator』に変更してください。ステーションナリを使ってオーバーレイプロジェクトを作成している場合はこれらの修正は不要です。

変更を保存してターゲット設定ダイアログを閉じてください。

3. オーバーレイのソースコードを修正する

コードをコンパイルするためにソースコードを修正します。

a. オーバーレイのソースファイルを検査する

プロジェクトウィンドウの『Files』タブをクリックしてプロジェクトに含まれる全ソースファイルを表示します。Overlay1.c ファイルをダブルクリックして開きます。こ

のファイルには `func_ol1` という名前の単純な算術関数が含まれています。ファイルを閉じてください。

`Overlay2.c` ファイルを開いてください。このファイルは `gVar` という名前の大域変数が含まれています。`func_ol2` 関数はこの大域変数 (`test.c` で定義されている) を修正します。

b. オーバーレイに合わせて `test.c` を編集する

`test.c` のコードはコンパイルできません。プログラムをコンパイルして実行するためにいくつかの定数を変更しなくてはなりません。

コードはプロジェクトのオーバーレイリストの位置を利用してオーバーレイを参照しなくてはなりません。プロジェクトウィンドウの『Overlay』タブをクリックして、オーバーレイの位置を決定することができます。オーバーレイは `LocalFunction` と `GlobalFunction` で、この順番で現れます。`LocalFunction` のインデックスは 0、`GlobalFunction` のインデックスは 1 です。

`test.c` ファイルで定数 `OVERLAY1_NUMBER` を探してください。***Index of first overlay*** のプレースキーバーを 0 に変更してください。第 2 のオーバーレイのプレースキーバーを 1 に変更してください。

オーバーレイのファイル名をプロジェクトにエントリした名前に一致するようにオーバーレイのファイル名を変更します。`OVERLAY1_FILENAME` の定義を探して以下のように変更してください。

```
#define OVERLAY1_FILENAME "LOCALFUNCTION."
```

`OVERLAY2_FILENAME` の定義を以下のように変更してください。

```
#define OVERLAY2_FILENAME "GLOBALFUNCTION."
```

注意： ファイル名はすべて大文字で入力してください。また拡張子のないファイル名はピリオド (.) で終えてください。

4. オーバーレイのロード方法を検査する

`MWLoadOverlay()` の戻り値を見てコードのロードが成功したかどうかを確かめます。成功した場合、オーバーレイコード定義した関数を安全に呼び出すことができます。オーバーレイをロードした後、関数 `MWNotifyOverlayLoaded()` を呼び出すこともできます。この偽関数はオーバーレイがロードされたことをデバッガに通知します。これによりデバッガはこのオーバーレイをデバッグするために必要な調整を行うことができます。これらの関数の詳細は『[Metrowerks のユーティリティライブラリ](#)』(p107) をご覧ください。

5. コードをコンパイルする

Project メニューの『Make』を選択してオーバーレイチュートリアルプログラムをコンパイルします。

6. オーバーレイをテストする

a. GD-ROM エミュレータにオーバーレイファイルを書き込む

オーバーレイファイルをメモリに読み込む前に、まず GD-ROM へ書き込まなくてはなりません。GDWorkshop を起動して、ダミーファイルと2つのオーバーレイファイル (LocalFunction と GlobalFunction) で構成される GD-ROM イメージを作成します。次に [Open/Close CD] ボタンをクリックしてエミュレータをアクティブにし、シュミレートした GD-ROM ドアを閉じます。

注意： GD-ROM の最小の長さの必要条件を満たすために、使用するダミーファイルのサイズは最低でも 800kb はなくてはなりません。第1の作業では2つのダミーファイル、データセクションには1つのダミーファイルが必要です。

b. デバッガを起動する

Project メニューの『Enable Debugger』を選択してください。次に Project メニューの『Debug』を選択してください。これでプログラムが Dreamcast にダウンロードされ、実行されます。開発用コンピュータ上で、CodeWarrior デバッガが起動して main() の開始点で停止するのを見ることができます。

c. プログラムをステップ実行する

result=func_ol1(gCount) の箇所までプログラムをシングルステップ実行してください。この時点で LocalFunction オーバーレイファイルがロードされます。この関数へステップインして、デバッガが表示するオーバーレイコードを見ます。njUserMain() へ戻ってください。このとき変数 inVar の値が1であることに注意してください。

func_ol2() 呼び出しまでステップ実行してください。この関数をステップ実行すると、大域変数 gVar の値が10になります。

7. オーバーレイを使ったプログラムのビルドとテストの完了

オーバーレイの注意点

オーバーレイの作成および使用方法に関する注意点を説明します。

[オーバーレイと例外](#)

[オーバーレイヘッダ](#)

[GDWorkshop](#)

オーバーレイと例外

オーバーレイを使用しているとき、C++ 例外はサポートされません。

オーバーレイヘッダ

[例 13.1](#) は、各オーバーレイファイルの先頭に表示されるオーバーレイヘッダレコードのフォーマットです。デバッガは64バイト長のヘッダを読み取って、オーバーレイを識別します。SH LCF generator がアクティブなとき、オーバーレイヘッダは自動的に作成されます。

例 13.1 CodeWarrior のオーバーレイレコードヘッダ

```
typedef struct overlayHeader
{
    char          flag[3];          /* 'MWo' */
    char          version;
    unsigned long overlayID;        /* Same ID found in DWARF */
    unsigned long loadAddress;      /* Address where to load the overlay*/
    unsigned long TextSize;        /* Size of the executable part */
    unsigned long DataSize;        /* Size of the data part */
    unsigned long StaticInit;      /* Static init pointer */
    unsigned long bssSize;         /* unused */
    unsigned long entryPoint;      /* unused */
    char          overlayName[32];
} OverlayHeader;
```

GDWorkshop

オーバーレイを処理するコードを再コンパイルした後、GD-ROM エミュレータの古いファイルを新しいファイルで置換しなくてはなりません。

古いファイルを置換しない場合、プログラムや CodeWarrior IDE に問題が発生します。

第 14 章 ライブラリとランタイムコード

Metrowerks は ANSI 基準の C/C++ ライブラリやランタイムライブラリなど、CodeWarrior 開発環境で利用できるライブラリを多数提供しています。ここでは CodeWarrior for Dreamcast 環境でのライブラリの使い方を説明します。

[Metrowerks のユーティリティライブラリ](#)

[ランタイムライブラリ](#)

[メモリとヒープの割り当て](#)

Metrowerks のユーティリティライブラリ

プログラミングをできるだけシンプルにするため、Metrowerks では `mw_utils.c` を用意しました。このライブラリはオーバーレイをロードおよび初期化する関数を含んでいます。

`mw_utils.c` を使うためには以下の手順に従ってください。

1. `mw_utils.c` ライブラリソースをプロジェクトへ追加する
2. ライブラリの関数を呼び出すファイルすべてに `mw_utils.h` ヘッダファイルをインクルードする

以下の関数がこのライブラリに含まれます。

[MWLoad\(\)](#)

[MWNotifyOverlayLoaded\(\)](#)

[MWInitOverlay\(\)](#)

[MWLoadOverlay\(\)](#)

`MWLoad()`

説明 名前のある指定したファイルの内容すべてを選択したメモリアドレスへロードします。

プロトタイプ `long MWLoad(char *pfileName, void *address)`

引数 `pfileName` : ロードするファイルの名前

`address` : ロードデータを配置するメモリアドレス

戻り値 成功した場合読みとったバイト数を返します。それ以外は -1 を返します。

MWNotifyOverlayLoaded()

説明 オーバーレイをロードしたことをデバッガに通知します。

プロトタイプ `void MWNotifyOverlayLoaded
(void *overlayLoadAddress)`

引数 `overlayLoadAddress` : オーバーレイのロードアドレス

注意 `MWNotifyOverlayLoaded()` は、実際にはコードを含まない偽関数です。しかしデバッガがこの関数を見つけたとき、デバッガはオーバーレイがロードされたことを認識します。次にデバッガはオーバーレイの先頭にあるヘッダ情報を読みとり、オーバーレイのためのブレイクポイントを復元します。

MWInitOverlay()

説明 `overlay` セクションのためにメモリを初期化します。

プロトタイプ `MWInitOverlay(void* address, signed long sizeByte)`

引数 `address` : ロードしたオーバーレイのメモリアドレス
`sizeByte` : オーバーレイのバイト長

注意 オーバーレイと `MWBload()` をロードした後にこの関数を呼び出します。これは CPU キャッシュを無効にし、セクションをクリアし、オーバーレイのための静的イニシャライザを呼び出します。

MWLoadOverlay()

説明 `MWBload()`、`MWInitOverlay()`、`MWNotifyOverlayLoaded()` を 1 つの関数ヘラップします。

プロトタイプ `MWLoadOverlay(char* pFileName, void* address)`

引数 `pFileName` : ロードするオーバーレイファイルの名前
`address` : ロードしたオーバーレイを配置するメモリアドレス

戻り値 成功した場合 `true` を返します。

注意 この関数はオーバーレイをロードするときに関数をまとめます。指定したオーバーレイファイルをメモリへロードし、オーバーレイを初期化し、オーバーレイをロードしたことをデバッガへ通知します。

ランタイムライブラリ

以下のライブラリをプロジェクトに組み込んでください。Dreamcast Support フォルダにあります。

以下のランタイムライブラリは Dreamcast SDK に含まれているものと同じですが、CodeWarrior で使用できるように変換されています。

```
nindows.elf.lib  
ninja.elf.lib  
shinobi.elf.lib  
sh4nlfzn.elf.lib
```

CodeWarrior には以下のランタイムライブラリが必要です。

```
MSLRuntimeDC.lib
```

C++ 標準ライブラリを使うためには以下のライブラリが必要です。

```
MSLCppDC.lib
```

`mw_pr()` 関数 (文字列を表示する関数) を使うためには以下のライブラリが必要です。

```
mw_output.lib
```

メモリとヒープの割り当て

ヒープとスタックのサイズは Dreamcast SDK ライブラリで規定されています。[SH Linker](#) 設定パネルでヒープやスタックを指定することはできません。

第 15 章 コマンドラインツール

CodeWarrior for Dreamcast にはコマンドラインコンパイラ、アセンブラ、リンカが含まれています。この章では CodeWarrior for Dreamcast のコマンドラインコンパイラとリンカを使ってアプリケーションをビルドする方法を説明します。

以下の項目について説明します。

[コマンドラインツールと IDE の違い](#)

[コマンドラインツールの場所](#)

[コマンドラインのスイッチ](#)

[環境変数の設定](#)

[コンパイルとリンク](#)

注意： ご使用前に Command Line Tools Release Notes をお読みください。

コマンドラインツールと IDE の違い

IDE のツールとコマンドラインツールの機能は異なります。

[オーバーレイのサポート](#)

[リンカコマンドファイルの生成](#)

オーバーレイのサポート

コマンドラインツールではオーバーレイを含むプロジェクトは作成できません。

リンカコマンドファイルの生成

コマンドラインツールはリンカコマンドファイル (.lcf) を自動生成しません。自分で .lcf ファイルを記述してプロジェクトへリンクしなくてはなりません。

コマンドラインツールの場所

コマンドラインツールは 3 つの実行可能ファイルのセットです。

mwccshx.exe : Dreamcast コンパイラ

mwldshx.exe : Dreamcast リンカ

`mwasmshx.exe` : Dreamcast アセンブラ

ツールは以下の場所にあります。

`CodeWarrior\Tools\Command Line Tools`.

コマンドラインのスイッチ

IDE 上では、リンカとプロジェクトの設定は設定パネルで行います。コマンドラインツールの設定は、コマンドラインのスイッチとオプションで行います。

ツールコンポーネントのコマンドラインスイッチの完全なリストを得るには、`-help` オプションを使います。以下はコマンドラインツールコンパイラのスイッチの完全なリストを取得する例です。

```
mwccshx -help
```

コマンドラインツールを使う場合、コンパイラとリンカを手動で設定します。一般に Dreamcast アプリケーションをコンパイルするためには、以下のスイッチとオプションを使う必要があります。

`mwasmshx` : アセンブラのスイッチ

```
-little
```

`mwccshx` : コンパイラのスイッチ

```
-prefix prefix_dc.h
-inline off
-g
-v
-little
-ansi off
-ARM off
-bool off
-strict off
-wchar_t off
-proc SH4
-heapsize 32768
-stacksize 32768
-fp hard
-main SG_SEC
-map
```

環境変数の設定

ランタイムにシステムパスとライブラリを検索するために使用される環境変数がいくつかあります。これらの変数は多くのタスクのコマンドラインを短縮することができます。変数のリストはセミコロン (;) で区切ります。

注意： 空白文字を含む環境変数を定義するときに、クオートを含める必要はありません。Windows はクオートを削除しません：そのままにしておくとコマンドラインツールは「未知のディレクトリ」警告を発します。バッチファイル、またはコマンドラインで変数を定義するときは以下の文法を使ってください。

```
set Folders=C:\First Path\Foo;D:\Second Path\Bar
```

C/C++ コンパイラ変数

MWCIncludes：指定したパスがシステムパスへ追加されます。この変数は以下のように定義します。

```
set MWCIncludes=  
CodeWarrior\Dreamcast Support\INCLUDE\;  
CodeWarrior\Dreamcast Support\Shinobi\Lib\  
CodeWarrior\Dreamcast Support\Shc\INCLUDE\  
CodeWarrior\Dreamcast Support\Shinobi\INCLUDE\  
CodeWarrior\Dreamcast Support\Runtime\Runtime DC
```

リンカ変数

MWLibraries：指定したパスがシステムパスへ追加されます。この変数は以下のように定義します。

```
set MWLibraries=  
CodeWarrior\Dreamcast Support\Shinobi\Lib\  
CodeWarrior\Dreamcast Support\Runtime\Runtime DC
```

MWLibraryFiles：指定したファイルがリンク順の最後へ追加されます。この変数は以下のように定義します。

```
set MWLibraryFiles=ninja.elf.lib;  
Shinobi.elf.lib;sh4nlfzn.elf.lib;  
Nindows.elf.lib;MSLRuntimeDC.LIB
```

コンパイルとリンク

コンパイラは自動的にリンカを起動します。リンク順はコマンドラインにリストされたファイルの順番によって決定します。コードをリンクするために有効なリンカコマンドファイルが必要であることに注意してください。

以下のフォルダにコマンドラインツールの使い方を紹介する 2 つの例題があります。

```
Examples\Command Line Tools\
```

teapot サンプルで使ったライブラリとソースファイルが、コマンドラインツールと同じフォルダにあることを確認してください。プロジェクトをコンパイル、リンクして **teamake1.elf** という実行可能ファイルを作成するバッチファイルを、以下のコマンドで作成することができます。

```
mwasmshx -little global32_cw.src

mwccshx -prefix prefix_dc.h -O4,p -inline off
-g -little -ansi off -ARM off -bool off
-strict off -wchar_t off -proc SH4
-heapsize 32768 -stacksize 32768 -fp hard
-main SG_SEC -v -o teamakel.elf -map strt1.obj.elf strt2.obj.elf
systemid.obj.elf toc.obj.elf sg_sec.obj.elf sg_arejp.obj.elf
sg_areus.obj.elf sg_areec.obj.elf sg_are00.obj.elf
sg_are01.obj.elf sg_are02.obj.elf sg_are03.obj.elf
sg_are04.obj.elf sg_ini.obj.elf aip.obj.elf zero.obj.elf
ninja.elf.lib Shinobi.elf.lib sh4nlfzn.elf.lib Nindows.elf.lib
MSLRuntimeDC.LIB model.c njloop.c sbinit.c t009.c test.c
global32_cw.o linker.lcf
```

第 16 章 トラブルシューティング

この章ではCodeWarrior for Dreamcastの使用中に遭遇するかもしれない問題とその解決方法を説明します。問題が起きたときはまずこの章をお読みください。

[ハードウェアとの通信](#)

[コンパイラの問題](#)

[デバッガの問題](#)

ハードウェアとの通信

ホストコンピュータと HKT-01 との通信に関する問題について説明します。

CodeWarrior が HKT-01 を認識しない

問題 CodeWarrior が HKT-01 と通信できない。

背景 HKT-01 は SCSI デバイスです。オペレーティングシステムが起動したときに HKT-01 の電源が入っていないければ認識されません。

解決方法 HKT-01 の電源を入れてからコンピュータを再起動してください。

Codescape が SCSI ドライバの更新を要求する

問題 SCSI ドライバが古すぎます。

背景 Codescape を使うには、最新版の Adaptec SCSI ドライバが必要です。

解決方法 Adaptec 社の Web サイト (<http://www.adaptec.com>) から最新版の SCSI ドライバをダウンロードしてください。

コンパイラの問題

コンパイラに関する問題について説明します。

Error '@5' がレジスタに割り当てられない

問題『Global Optimization』が 0 に設定されている場合、コンパイラはインラインアセンブラステートメントをコンパイルできません。

背景 最適化レベルが 0 のとき、コンパイラは仮想レジスタ割り当てをしません。このため、インラインアセンブラルーチンをコンパイルするとき、利用可能な実レジスタがないことがあります。

解決方法 C/C++ language 設定パネルで『Don't Inline』をオンにするか、最適化レベルを 1 以上にしてください。

デバッガの問題

デバッガに関する問題について説明します。

GDROM のデータファイルを持つプログラムを実行できない

問題 デバッガがデータファイルを見つけることができません。

背景 データファイルは GDROM からスプールされ、CodeWarrior デバッガではなく GD Workshop を通じてロードされます。

解決方法 Workshop を使って GDROM デバイスをエミュレートしてください。

索引

記号

* 74

A

Address オプション

Section Mappings 設定パネル 54

ALIGN 72

ALIGNALL 72

Allow Space In Operand Field オプション

SH Assembler 設定パネル 49

B

Byte Ordering オプション

SH Target 設定パネル 48

C

C++

サポート 62

制限 62

標準ライブラリ 62

Case Sensitive Identifiers オプション

SH Assembler 設定パネル 49

Code Model オプション

SH Target 設定パネル 48

Codescape デバッガ 18

CodeWarrior と使う 39

printf() 41

起動 40

CodeWarrior

概要 8

マニュアルの構成 8

Contains Sections オプション

Section Mappings 設定パネル 54

cout()

代わりに mw_pr() を使う 63

D

debug-printf() の代わり

デバッグ用 41

Directives Begin With '!' オプション

SH Assembler 設定パネル 49

Disable Deadstripping オプション

SH Linker 設定パネル 55

DWARF の生成 55

E

Entry Point オプション

SH Linker 設定パネル 56

F

File Name フィールド

SH Target 設定パネル 48

FUNCTION 74

G

GDROM

トラブルシューティング 116

GDWorkshop 105, 106

Generate Dwarf Info オプション

SH Linker 設定パネル 55

Generate Link Map オプション

SH Linker 設定パネル 55

Generate Listing File オプション

SH Assembler 設定パネル 49

GROUP 74

H

Heap Size オプション

SH Linker 設定パネル 55

I

IDE

解説 17

\$INCLUDE 参照 LCF

\$REF_INCLUDE 参照 LCF

L

Labels Must End With ':' オプション

SH Assembler 設定パネル 49

LCF

\$INCLUDE 71

\$output_name 72

\$REF_INCLUDE 72

FUNCTION 74

GROUP 74

- アラインメント 72
- 演算オペレータ 72
- 疑似命令 69
- 文法 68
- マッピング 73
- LCF 参照 リンカコマンドファイル
- Linker ポップアップメニュー
 - Target Settings 設定パネル 46

M

- Mac Size オプション
 - Section Mappings 設定パネル 54
- MSLCppDC.lib 62, 109
 - 参照 ライブラリ
- mw output.lib 109
- mw_pr()
 - printf() の代わり 38

O

- Optimize For オプション
 - Global Optimizations 設定パネル 51
- Output Directory オプション
 - Target Settings 設定パネル 47
- \$output_name 参照 LCF

P

- Prefix File オプション
 - SH Assembler 設定パネル 49
- printf() の代わり
 - デバッグ 38
- Project Type オプション
 - SH Target 設定パネル 48

S

- Save Project Entries オプション
 - Target Settings 設定パネル 47
- SCSI ドライバの問題 115
- \$segment 参照 リンカコマンドファイル
- Segment オプション
 - Section Mappings 設定パネル 54
- Stack Size オプション
 - SH Linker 設定パネル 56
- Suppress Warning Messages オプション
 - SH Linker 設定パネル 55

T

- Target CPU オプション
 - SH Processor 設定パネル 50
- Target Name フィールド
 - Target Settings 設定パネル 46

U

- Use Floating Point Instructions オプション
 - SH Processor 設定パネル 50

ア行

- アセンブラ
 - 解説 18
 - 疑似命令 34, 82
 - プリフィクスファイル 49
- アセンブラコード
 - Hitachi 疑似命令を変換 34
 - 埋め込まれた C ソースコード 79
- アプリケーション
 - 作成 21
- インストール
 - CodeWarrior 13
 - SEGA ライブラリ 14
 - システムの必要条件 13
 - 動作確認 14
- インラインアセンブラ
 - インストラクション 80
 - 局所変数を参照 80
 - ケースセンシティブ 80
 - コメント 81
 - 最適化 80
 - サポートするニーモニック 91 ~ 98
 - ニーモニック 89
 - 文法 79
 - 変数を初期化 80
 - ラベル 81
 - レジスタ 81
- エンディアンフォーマット 48
- オーバーレイ 99
 - コマンドラインツール 111
 - チュートリアル 99
 - デバッグ 104
 - ヘッダフォーマット 105
- オーバーレイと C++ 例外 105
- オプション
 - コマンドラインツール 112

カ行

環境変数

コマンドラインツール 112

関数の並べ替え 74

組込み関数 84

__abs 84

__fabs 85

__fsqrt 85

__labs 84

__memcpy 85

Hitachi との互換性 86 ~ 89

コマンドライン

リンク順 113

コマンドラインツール

オーバーレイ 111

オプション 112

環境変数 112

スイッチ 112

場所 111

コンパイラ

解説 18

サ行

最適化

Dreamcast 59

インストラクションスケジュール 60, 82

インラインアセンブラ 80

キャッシュヒット 74

共通部分式の削除 60

強度の削減 61

コピーによる伝搬 61

大域レジスタ割り当て 59

ディレイスロットの充てん 60

デッドストアの除去 60

デバッグの安全性 51

ピーブホール 61

未使用コードの除去 60

ライフタイムレジスタ割り当て 61

ループ不変コード 60

ループアンローリング 61

レベルを指定 51

ローカル 51

実行可能ファイル

命名規約 30, 48

スイッチ

コマンドラインツール 112

数値フォーマット 58

整数 58

浮動小数点 58

スタックサイズ 109

スタックフレーム 83

ポインタ 63

ステーションナリ 21

デフォルトの内容 23, 44

静的ライブラリ

作成 29

デバッグ 38

設定パネル

Debugger Settings 56

Section Mappings 54

SH Assembler 48

SH Linker 54

SH Processor 50

SH Target 47

Target Settings 45

タ行

ターゲット設定 43

解説 43

ダイアログ 43

デフォルト値 44

デッドストリップ 72

無効 55

デバッガ

Codescape 18

printf() の代わり 38

開始 37

解説 18

起動 37

デバッグ

最適化したコード 60

静的ライブラリ 38

トラブルシューティング 115 ~ 116

GDROM 116

SCSI ドライバ 115

通信 115

レジスタ 115

ハ行

ヒープサイズ 109

浮動小数点

フォーマット 58

プラグマ 62

変数

局所 , インラインアセンブラ 80
初期化 , インラインアセンブラ 80

マ行

マッピング 73
マニュアルの構成 8
未使用コードの削除 (デッドストリップ) 65
メイクファイル
 CodeWarrior プロジェクトへ変換 31
 解説 19
命名規約
 実行可能ファイル 30, 48
 ライブラリ 30, 48

ヤ行

呼び出し規約
 Dreamcast 59

ラ行

ライブラリ
 C++ 標準 62, 109
 SEGA 65
 SEGA ライブラリをインストール 14
 デバッグ 109
 デバッグ用 38
 命名規約 30, 48
 ランタイム , CodeWarrior 109
 ランタイム , SDK 109
 リンク順 65
リリースノート 7
リンカ
 解説 18
 セクションマップ 66
 設定 46
 実行可能ファイル 66
リンカコマンドファイル 66
 #include 65
 \$segment 70
 アラインメント 72
 コメント 73
 シンボル 74
 ロケーションカウンタ 73
 演算オペレータ 72
 代入 73
リンク順 65
 コマンドライン 113
レジスタ

インラインアセンブラ 81
スタックフレームポインタ 63
ステータスレジスタ 82
トラブルシューティング 115
浮動小数点 50

CodeWarrior Targeting Dreamcast

Credits

writing lead: Roger Wong

engineering: Aaron Smith, Guohua Cao,
Laurent Visconti, Nick Havens,
Shoji Ueda, Takashi Kashima

frontline warriors: David Wilson, Takashi Kashima,
CodeWarrior users everywhere

translation: Naomi Owashi

proofreader: Chizu Kanbara

CodeWarrior 文書のガイド

CodeWarrior の文書はツールと同様にモジュールのように構成されています。ツール、言語、ライブラリ、ターゲットごとにマニュアルがあります。各 CodeWarrior 製品によって含まれるマニュアルが異なります。この表に記載されていないマニュアルが含まれることもあります。

コアマニュアル	
IDE User Guide	CodeWarrior IDE と CodeWarrior デバッガの使い方
言語 / コンパイラのマニュアル	
C Compilers Reference	C/C++ フロントエンドコンパイラの情報
Pascal Compilers Reference	Pascal フロントエンドコンパイラの情報
Pascal Language Reference	Metrowerks における ANS Pascal の実装
Assembler Guide	スタンドアローンアセンブラの文法
Command-Line Tools Reference	Mac OS MPW コンパイラのコマンドラインのオプション
Plugin API Manual	CodeWarrior のプラグインコンパイラ / リンカの API
ライブラリのマニュアル	
MSL C Reference	Metrowerks ANSI 標準 C ライブラリの関数のリファレンス
MSL C++ Reference	Metrowerks ANSI 標準 C++ ライブラリの関数のリファレンス
Pascal Library Reference	Metrowerks ANS Pascal ライブラリの関数のリファレンス
MFC Reference	Win32 用の Microsoft Foundation Classes リファレンス
Win32 SDK Reference	Win32 API の Microsoft リファレンス
The PowerPlant Book	Mac OS 用アプリケーションフレームワークのガイド
PowerPlant Advanced Topics	PowerPlant での Mac OS プログラミングの高度なテクニック
ターゲットマニュアル	
Targeting Java VM	Java 仮想マシンプログラミングでの CodeWarrior の使い方
Targeting Mac OS	Mac OS プログラミングでの CodeWarrior の使い方
Targeting MIPS	MIPS 組み込みプロセッサプログラミングでの CodeWarrior の使い方
Targeting NEC V800 series	NEC V810/830 プロセッサプログラミングでの CodeWarrior の使い方
Targeting Net Yaroze	Net Yaroze プログラミングでの CodeWarrior の使い方
Targeting Palm OS	Palm OS プログラミングでの CodeWarrior の使い方
Targeting PlayStation	PlayStation プログラミングでの CodeWarrior の使い方
Targeting PowerPC	PPC 組み込みプロセッサプログラミングでの CodeWarrior の使い方
Targeting Windows	Windows プログラミングでの CodeWarrior の使い方

印の付いているマニュアルは日本語訳が用意されています。