Visual Studio 2005 スマートデバイス開発

このドキュメントのコンテンツは廃止され、今後は更新もサポートもされません。一部のリンクは機能しない可能性があります。
廃止されたコンテンツは、このコンテンツの最後に更新されたバージョンです。

スマート デバイス開発

Visual Studio 2005 では、Pocket PC や Smartphone など、Windows CE ベースのスマート デバイスで実行するソフトウェアを開発するために、統合的で充実したサポートを実現しています。.NET Compact Framework で実行するマネージ アプリケーションを記述するには、Visual C++ または Visual Basic を使用します。また、ネイティブ アプリケーションを記述するには Visual C++ も使用できます。どの言語を選択しても、デスクトップ コンピュータの開発時と同じコード エディタ、デザイナ、およびデバッガのインターフェイスを使用できます。選択した言語で使用できるスマート デバイス プロジェクトのいずれかを選択し、コーディングを始めるだけです。

また、Visual Studio には、記述したコードを開発用コンピュータで実行およびデバッグできるエミュレータや、アプリケーションとそのリソースをエンド ユーザーのデバイスに配置するために CAB ファイルにパッケージ化する処理を簡略化するツールが用意されています。

スマート デバイス プロジェクトの最新情報については、「Mobile Developer Center」を参照してください。

このセクションの内容

スマート デバイス プロジェクトの概要

デバイス アプリケーション開発に固有の問題に関する概要を説明します。ここでは、Visual Studio 2005 の新機能、Visual Studio for Devices と他の Windows Mobile の SDK およびツールとの関係、およびデバイス用のソフトウェア開発を行うためのコンピュータの設定方法などについて説明します。

スマート デバイス開発のデザイン

プロジェクトの種類の選択、開発言語の選択、およびエミュレータ用のスキンのカスタマイズについて説明します。

スマート デバイスを開発コンピュータに接続する

接続方法と接続オプションについて説明します。

.NET Compact Framework を使用したデバイスのプログラミング

Visual C# または Visual Basic のいずれかと .NET Compact Framework を使用してスマート デバイス ソフトウェアを開発するときの一般的な手順について説明します。

Visual C++ を使用したデバイスのプログラミング

Visual C++ を使用してネイティブ デバイス アプリケーションを開発するときの一般的な手順について説明します。

デバイス プロジェクトのデバッグ

デスクトップのデバッグとの違い、およびネイティブ コードとマネージ コードの両方で構成されたソリューションのデバッグ手順について説明します。

配置用のデバイス ソリューションのパッケージ化

開発したデバイス アプリケーションをパッケージ化する手順およびそのアプリケーションを 1 つ以上の対象デバイスに転送する手順について説明します。

デバイス プロジェクトにおけるセキュリティ

作成したファイルにセキュリティ証明書を使用して署名し、デバイスを用意する方法について説明します。

サンプルとチュートリアル (スマート デバイス プロジェクト)

完全なプロジェクトを例に使って、デバイスのプログラミングに使用できる構文、構造、およびテクニックを紹介します。

リファレンス (デバイス)

ATL for Devices と MFC for Devices のリファレンス トピック、デバイス プロジェクトのユーザー インターフェイス リファレンス、およびエラー メッセージなどが含まれます。

関連するセクション

.NET Compact Framework

デバイス アプリケーションのプログラミング方法について説明します。.NET Compact Framework により、.NET Framework の機能をデバイスで利用できます。.NET Compact Framework を .NET Framework と比較し、重要なコンポーネントについて説明し、共通のプログラミング タスクを示し、サポートされているクラスの一覧を示します。

Visual Studio の紹介

Visual Studio の新機能について説明します。

Visual Studio の統合開発環境

Visual Studio で作成するアプリケーションのデザイン、開発、デバッグ、テスト、および管理について説明します。

スマート デバイス プロジェクトの概要

Visual Studio 2005 には、Pocket PC、Smartphone など Windows CE .NET ベースのプラットフォーム向けにアプリケーションを開発するときに必要なツールとフレームワークが用意されています。スマート デバイスを持っていなくても、Visual Studio の統合開発環境を残したまま、エミュレーション技術を使用して、スマート デバイス アプリケーションの作成とテストを行うことができます。

Visual Studio .NET 2005 では、スマート デバイス アプリケーション開発用の言語として Visual Basic .NET、Visual C#、および Visual C++ がサポートされています。

このセクションの内容

スマート デバイス プロジェクトの新機能

スマート デバイス プロジェクトに関する Visual Studio 2005 の新機能について説明します。

アプリケーション開発の概要 (デバイス)

Windows CE ベースのソリューション、小さなデバイスの設計問題、既存ソリューションの移植、プログラム言語の選択、デスクトップ開発との比較、および用語の問題について説明します。

デバイス機能と必要な開発ツール

さまざまなバージョンのスマート デバイスと、それぞれを開発するときに必要なツールについて説明します。

スマート デバイス プロジェクトのハードウェア要件とソフトウェア要件

デバイス アプリケーションをビルドするとき、デバイスと接続するとき、および開発段階でデバイスでアプリケーションを実行するときの要件を示します。

デバイス プロジェクトのリモート ツール

スマート デバイス プロジェクトに使用できる特別なツールについて説明します。

方法 : スマート デバイス開発用のヘルプを最適化する

デバイス プロジェクトに関するヘルプの上手な使い方を示します。フィルタの使用方法についても説明します。

方法 : Visual Studio でデバイス エミュレータを起動する

物理デバイスの代わりにデバイス エミュレータを開く方法について説明します。

以前のツールで作成されたプロジェクトの更新

以前のバージョンやツールで開発したプロジェクトの移植方法について説明します。

スマート デバイス開発での操作方法

スマート デバイスに関する一般的な問題についてのヘルプを提供するリンクを示します。

参照

その他の技術情報

スマート デバイス開発
Mobile Developer Center
スマート デバイスのプログラミング

スマート デバイス プロジェクトの新機能

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

Visual Studio 2005 では、次の新機能または拡張機能を使用できます。

SP1 の新機能

このセクションは、Visual Studio 2005 SP1 に合わせて更新されています。

eMbedded Visual C++ から Visual Studio 2005 へのアップグレード ウィザード

eMbedded Visual C++ プロジェクトのアップグレード ウィザードが改良されました。

デバイスでサポートされているデスクトップ MFC クラスの一覧

デバイス MFC ライブラリに 15 個の MFC クラスが追加されました。次のクラスは、eMbedded Visual C++ には存在しましたが、Visual Studio 2005 には存在しませんでした。

- CBitmapButton クラス
- CDialogBar クラス
- CEditView クラス
- CFindReplaceDialog クラス
- CHttpConnection クラス
- CHttpFile クラス
- CInternetConnection クラス
- CInternetException クラス
- CInternetFile クラス
- CInternetSession クラス
- COleSafeArray クラス
- CReBar クラス
- CReBarCtrl クラス
- CRecentFileList クラス
- CSplitterWnd クラス

Prepare for SQL Server Compact Edition

SQL Server 2005 Mobile Edition は Microsoft SQL Server 2005 Compact Edition に置き換えられています。この変更は、Visual Studio IDE の各種ダイアログ ボックスで確認できます。

データ アクセスの概要 (マネージ デバイス プロジェクト)

SQL Server Compact Edition データベース、スキーマの変更、および他のデータベース管理タスクの実行を、すべて Visual Studio の IDE から行います。ビジネス オブジェクト、SQL Server データベース、SQL Server Compact Edition データベース、または Web サービスをデータ ソースとして使用します。

Windows CE 6.0 リリース (2006 年 11 月 1 日)

Windows CE 6.0 アプリケーションの開発をサポートします。

Visual Studio 2005 の新機能

Visual C++ デバイス アプリケーション開発の新機能

MFC (Microsoft Foundation Classes)、ATL (Active Template Library)、および Windows CE ネイティブ API を使用して、スマート デバイス アプリケーションを開発します。eMbedded Visual C++ 4.0 プロジェクトを移行します。

マネージ デバイス プロジェクトの新機能

マネージ開発の新機能には、統合 Smartphone のサポート、新しいコントロール、アンカーとドッキングなどがあります。

方法 : Visual Studio でデバイス エミュレータを起動する

ARM プロセッサ向けにコンパイルされたコードを実行するために一から構築された、完全に新しいデバイス エミュレータを使用して、ランタイム イメージを実行、テスト、およびデバッグできます。

データ アクセスの概要 (マネージ デバイス プロジェクト)

SQL Mobile データベース、スキーマの変更、および他のデータベース管理タスクの実行を、すべて Visual Studio の IDE から行います。ビジネス オブジェクト、SQL Server データベース、SQL Mobile データベース、または Web サービスをデータ ソースとして使用します。

デバイス ソリューションをパッケージ化する方法の概要

ファイル システム エディタおよびレジストリ エディタと共にスマート デバイス Cab プロジェクトを使用して、開発用のアプリケーションをパッケージ化します。

デバイス プロジェクトのプラットフォーム切り替え

1 つのソースを使用して、Pocket PC と Smartphone などの複数のプラットフォームをターゲットとします。

デバイス プロジェクトのリモート ツール

Windows CE ベースのデバイスで、プログラミングおよびデバッグ タスクをリモート実行します。

セキュリティの概要 (デバイス)

適切な証明書でファイルを署名し、デバイスを用意します。

接続方法の選択

USB、イーサネット、802.11b/g、Bluetooth、シリアル ポート、または赤外線をとおして、開発コンピュータをデバイスに接続します。

Mobile Developer Center Web Site

初めての使用方法、コード サンプル、よく寄せられる質問、練習コーナー、その他のリソースなど、スマート デバイス プロジェクト情報の中心的なサイトにアクセスします。

関連項目

Visual Studio 2005 の新機能

.NET Compact Framework 2.0 の新機能

新機能 (SQL Server Books Online)

アプリケーション開発の概要 (デバイス)

Visual Studio は、デバイス用アプリケーションの開発に 2 つの方法をサポートしています。

- Web サーバー上で実行され、ブラウザ機能が付いた各種モバイル デバイスで、さまざまな形式の表示ができるモバイル Web アプリケーションを開発できます。
- デバイス上で実行される Windows CE ベースおよび Windows Mobile ベースのリッチ クライアント アプリケーションを開発することもできます。このドキュメントで "スマート デバイスのアプリケーション開発" と表現する場合は、こちらの方法を指します。

スマート デバイス ソリューションと Windows CE

Windows CE、Pocket PC、Smartphone、Windows Mobile™ の各ソフトウェアの関係については、Microsoft Mobile Developer Center の「Mobile Developer Center Editor's Note」を参照してください。

バージョンの互換性

デバイス アプリケーションを開発するときに連携するツールのバージョンやテクノロジを特定するには、「Windows Mobile ベースのデバイス向け開発ツールの紹介」を参照してください。Visual Studio 2005 のマネージ プロジェクトでは、特に記述されていない限り、すべてのプラットフォームが .NET Compact Framework のバージョン 2 を対象にしています。たとえば [新しいプロジェクト] ダイアログ ボックスでは、バージョン 1.0 の .NET Compact Framework を対象にしているスマート デバイス テンプレートは "(1.0)" とマークされています。

デザイン ガイドライン

デバイス アプリケーションのデザインによって、ユーザーがタスクをどれだけ簡単に、すばやく、効率的に完了できるかが決まります。さまざまなデバイスの機能を利用できるようにアプリケーションを最適化することにより、より使いやすく、統一性が取れ、応答性に優れ、アクセスしやすいアプリケーションをユーザーに提供できます。特定のインターフェイス機能についての細かいデザイン ガイドラインについては、各デバイスのソフトウェア開発キット (SDK : Software Development Kit) を参照してください。

デバイス エミュレータ

デバイス エミュレータは、Visual Studio 2005 デバイス プロジェクトに特化して設計されています。ARM 命令セット向けにコンパイルされたアプリケーションが実行されます。また、ユーザー モード プロセスとして実行されます。Visual Studio には、エミュレータと通信できる DMA のトランスポート機能ができました。DMA のトランスポートは、従来の TCP/IP のトランスポートよりも高速で、ネットワークの接続性など外部の要因に左右されません。また、確定的な接続と接続解除の機能があります。

Visual Studio 2005 には、Pocket PC 2003 SE、Pocket PC 2003 SE Square、Pocket PC 2003 SE Square VGA、Pocket PC 2003 VGA、Smartphone 2003 SE、および Smartphone 2003 SE QVGA 用のエミュレータ イメージが用意されています。

エミュレータのメニュー バーにある [ヘルプ] をクリックすると、エミュレータについて説明したヘルプ トピック一覧が表示されます。

エミュレータを開くには、[ツール] クリックし、[デバイスへの接続] をクリックして、開くエミュレータを選択します。次に、[接続] をクリックします。

メモ :

デバイス エミュレータは、Direct3D ライブラリおよび DirectPlay ライブラリをサポートしています。ただし、ハードウェア アクセラレータについては一切サポートされません。エミュレータ上では、実際のハードウェア上で動作する Direct3D アプリケーションおよび DirectPlay アプリケーションのパフォーマンスは必ずしも反映されません。また、実際のハードウェアでは、ハードウェア アクセラレータがサポートされる場合とサポートされない場合とがあります。Direct3D アプリケーションおよび DirectPlay アプリケーションを、実際に出荷されるデバイス上でテストすることを強くお勧めします。

セキュリティ

デバイス アプリケーションのリモート接続には、新たなセキュリティ問題があります。詳細については、「デバイス プロジェクトにおけるセキュリティ」、「.NET Compact Framework のセキュリティ」、および「ネイティブ コードと .NET Framework コードのセキュリティ」を参照してください。

既存ソリューションの移植

移植と移行に関するヒントについては次を参照してください。

- マネージ デバイス プロジェクトの作成および開発
- eMbedded Visual C++ から Visual Studio 2005 へのアップグレード ウィザード

デバイスとデスクトップ

デバイス アプリケーションの開発には、デスクトップ アプリケーションの開発に使用する Visual Studio 環境と同じ環境を使用しますが、いくつかの違いもあります。次に例を示します。

- Visual Studio 環境では、デバイスに接続したり、デバイス上でデバッグを行ったりするためのツールが用意されています。
- プロジェクト作成時にプロジェクトの種類とテンプレートを選択することに加え、アプリケーションの実行およびデバッグに使用するデバイスを選択する必要があります。選択できるデバイスは、開発用コンピュータに接続された物理デバイス、ネットワーク接続されたデバイス、開発用コンピュータ上で実行されるデバイス エミュレータのいずれかです。
- クラスの数とメンバは、デスクトップ アプリケーションの開発に使用できるものとは異なります。.NET Compact Framework を使用したマネージ プロジェクトの場合、デバイスに使用できるクラス数は少なくなります。また、クラスの補数は一般にプラットフォームによって異なります。ネイティブ プロジェクトの場合も同様であり、一部の Windows API、MFC クラス、ATL コンポーネントのみを使用できます。クラスが使用できるかどうかを調べる方法としては、ドキュメントを調べる、IntelliSense を使用する、プロジェクトがアクティブなときに Visual Studio オブジェクト ブラウザを使用するなどの方法があります。
- ネイティブ コードには、デスクトップ アプリケーションと同様、プラットフォーム呼び出しによりアクセスできます。.NET Compact Framework では、COM 相互運用は制限付きでサポートされています。マネージ コード内に COM オブジェクトを作成すること、および ActiveX コントロールとの相互運用はサポートされていません。
- 言語項目に関しても相違点があります。たとえば、デスクトップ開発で使用されている Visual Basic キーワードでも、デバイス開発ではサポートされていないものがあります。
- Visual Studio ドキュメントでデスクトップ プロジェクト用に用意されているコード スニペットをデバイス プロジェクトで使用すると、ビルド エラーが発生する場合があります。
- デバイス開発では、デバイスのフォーム ファクタ、使用電力、メモリ制約など、デスクトップ開発では考慮する必要のないデザイン上の考慮事項があります。

その他のリソース

詳細については、「Mobile Developer Center」を参照してください。

参照

その他の技術情報

スマート デバイス プロジェクトの概要

デバイス機能と必要な開発ツール

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

Visual Studio 2005 は、Windows Mobile Version 5.0、Windows Mobile 2003 および 2003 Second Edition を実行するデバイスや、Windows CE 5.0 を実行する Windows CE ベースのハードウェアのためのアプリケーション開発に対応しています。

しかし、レガシ デバイスも数多く存在します。そのため、開発ツール、.NET Compact Framework のバージョン、基になる Windows CE オペレーティング システムという点において、何が必要なのか混乱することがあります。

ツール比較表

さまざまなスマート デバイス ハードウェア、ハードウェア機能、および開発ツールを表にまとめたものを次に示します。この一覧は、時間の経過につれて変更されることがあります。最新の完全な情報を取得するには、MSDN ライブラリの技術文書「Introduction to Development Tools for Windows Mobile-based Devices」を参照してください。

IDE 機能の概要

この表は、各種 IDE の機能の概要を示しています。列見出しの略語は、次のことを意味しています。

- eVT3C = eMbedded Visual C++ 3.0
- eVT3V = eMbedded Visual Basic 3.0
- eVC4 = eMbedded Visual C++ 4.0 and service pack 4.0
- VS2003 = Visual Studio .NET 2003
- VS2005 = Visual Studio 2005

		eVT3C	eVT3V	eVC4	VS2003	VS2005
コードの種類	ネイティブ コード	X		X		○
	インタープリタ コード		X			
	マネージ コード				X	X
	サーバー側コード				X	○
デバイス SDK	Pocket PC 2000 および Pocket PC 2002	X	X		X	
	Smartphone 2002	X				
	Windows Mobile 2003			X	X	X
	Windows Mobile 2003 Second Edition			X	X	X
	Windows Mobile 5.0					○

.NET Compact Framework のツールと OS サポート

この表は、どのバージョンのツールと Windows Mobile ソフトウェアが .NET Compact Framework 1.0 および 2.0 をサポートしているかを示しています。

		Version 1.0	Version 2.0
ツール	Visual Studio .NET 2003	X	
	Visual Studio 2005	X	○

Windows Mobile ソフトウェアのバージョン	Windows Mobile 5.0	ROM 内 (1.0 SP3)	ユーザー インストール可能
	Windows Mobile 2003 Second Edition	ROM 内 (1.0 SP2)	ユーザー インストール可能 (Pocket PC のみ)
	Windows Mobile 2003	ROM 内 (1.0 SP1)	ユーザー インストール可能 (Pocket PC のみ)
	Smartphone 2002		
	Pocket PC 2002	ユーザー インストール可能	
	Pocket PC 2000	ユーザー インストール可能	

データベース テクノロジのサポート

次の表は、Visual Studio 2005 SP1 に合わせて更新されています。

この表は、さまざまなバージョンの Windows Mobile でどのデータベース テクノロジがサポートされているかを示しています。

	SQL Server 2005 Compact Edition または SQL Server 2005 Mobile Edition	SQL CE 2.0	EDB	CEDB	ADOCE
Windows Mobile 5.0	ユーザー インストール可能	ユーザー インストール可能 (Pocket PC のみ)	ROM 内	ROM 内*	ユーザー インストールはサポート外
Windows Mobile 2003 Second Edition	ユーザー インストール可能 (Pocket PC のみ)	ユーザー インストール可能 (Pocket PC のみ)	なし	ROM 内	ROM 内
Windows Mobile 2003	ユーザー インストール可能 (Pocket PC のみ)	ユーザー インストール可能 (Pocket PC のみ)	なし	ROM 内	ROM 内
Smartphone 2002	なし	なし	なし	ROM 内	なし
Pocket PC 2002	なし	ユーザー インストール可能 (Pocket PC のみ)	なし	ROM 内	ROM 内
Pocket PC 2000	なし	ユーザー インストール可能 (Pocket PC のみ)	なし	ROM 内	ROM 内 (ほとんどのデバイス)

* Windows Mobile 5.0 では、CEDB は ROM に入っていますが推奨されていません。代わりに EDB を使用してください。

メモ

- 新しいバージョンの Windows CE または Windows Mobile に合わせてデバイスをアップグレードすることについては、デバイス メーカーに問い合わせてください。Microsoft では、エンド ユーザーに対して、特定のデバイスのアップグレードは提供していません。
- Visual Studio 2005 Express Editions には、スマート デバイス プロジェクトのサポートは含まれません。
- eMbedded Visual Basic のツールはサポートされていません。eMbedded Visual Basic ランタイムは、デバイスの ROM に含まれません。
- eMbedded Visual C++ 4.0 および eMbedded Visual Basic 4.0 は、Mobile Developer Center からダウンロードできます。
- バージョン 1.0 の .NET Compact Framework が ROM に含まれていない場合は、Pocket PC 2000、2002、2003 および 2003 SE の各デバイスの RAM にインストールできます。バージョン 2.0 が ROM に含まれていない場合は、Pocket PC 2003、Windows CE 5.0、

および Windows Mobile 5.0 の RAM または永続的な記憶領域にインストールできます。

- Compact Framework の最新バージョンは 2.0 であり、Mobile Developer Center から RAM インストールの形式で入手できます。

参照

概念

以前のツールで作成されたプロジェクトの更新

その他の技術情報

スマート デバイス プロジェクトの概要

スマート デバイス プロジェクトのハードウェア要件とソフトウェア要件

開発用コンピュータ、対象デバイス、およびその接続に必要な要件を次に示します。

開発用コンピュータ

Visual Studio 2005 をインストールしたときに [スマート デバイス プログラマビリティ] (既定) を選択すると、約 900 MB のハード ディスク容量が使用されます。スマート デバイス アプリケーションを開発しない場合、Smart Device Programmability をアンインストールすると、この容量を使用できるようになります。コントロール パネルの [プログラムの追加と削除] で Visual Studio のインストールをクリックし、[変更と削除] をクリックして、手順に従います。

デバイス プロジェクトでエミュレータを使用する場合、少なくとも追加で 64 MB の RAM が必要です。

デバイス

対象のデバイスで、開発先のプラットフォームをサポートしている必要があります。Visual Studio 2005 では、Pocket PC 2003、Smartphone 2003、Windows CE 5.0 以降がサポートされています。

.NET Compact Framework が ROM にインストールされていない場合、Framework 対応のデバイスには約 2 MB の RAM が必要です。

接続

Visual Studio 2005 の要件は次のとおりです。

ハードウェア

物理デバイスと開発コンピュータとの間で、無線接続が有効ではない場合、デバイスと開発コンピュータを接続するには、シリアル ケーブルまたは USB ケーブルが必要です。これらのケーブルはデバイス メーカーから提供されます。この接続を使用する前に、デバイス メーカーによって提供された指示に従って開発用コンピュータとデバイスを設定する必要があります。

デバイスにエミュレータを使用する場合、追加のハードウェアは必要ありません。

ソフトウェア

Microsoft ActiveSync Version 4.0 以降

参照

概念

インストールとセットアップの基本事項

その他の技術情報

スマート デバイス プロジェクトの概要

Visual Studio のエディション

デバイス プロジェクトのリモート ツール

embedded Visual C++ 4.0 では、以下のリモート ツールが使用できました。このリモート ツールは Visual Studio 2005 と同梱され、デバイス アプリケーションの開発とデバッグに役立っていました。

このようなスタンドアロン ツールを起動するには、Windows デスクトップの [スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2005] の順にポイントして、[Visual Studio Remote Tools] をクリックします。次に、メニューからリモート ツールのうち 1 つを選択します。

目的	使用するツール
対象デバイスでファイル システムの表示と管理を行います (エクスポートとインポートを含みます)。	リモート ファイル ビューア
対象デバイスで実行されている各プロセスについて、ヒープ ID とフラグに関する情報を表示します。	リモート ヒープ ウォーカ
対象デバイスで実行されている各プロセスに関する情報を表示します。	リモート プロセス ビューア
対象デバイスのレジストリの表示と管理を行います。	リモート レジストリ エディタ
対象デバイスで実行されているアプリケーションに関連付けられた、Windows の受信メッセージを表示します。	リモート スパイ
対象デバイスからビットマップ (.bmp) ファイル形式の画面イメージをキャプチャします。	リモート ズームイン

参照

その他の技術情報

スマート デバイス プロジェクトの概要

方法：スマート デバイス開発用のヘルプを最適化する

Visual Studio の MSDN ライブラリをインストールした際に [最小] または [カスタム] を選択した場合、スマート デバイス開発者が使用可能な Visual Studio 2005 ヘルプ ファイルの一部がインストールされていない可能性があります。特に、[最小] セットアップには、ライブラリの Mobile および Embedded 開発セクションが含まれていません。

Mobile および Embedded 開発セクションには、Pocket PC および Smartphone SDK ドキュメント、Windows CE 情報などの資料が含まれています。このセクションのトピックは、Visual Studio のスマート デバイス開発セクションのトピックから参照されています。このセクションは必ずインストールされ、Visual Studio 環境を使用したスマート デバイス アプリケーションの開発について重点的に説明しています。次の手順は、インストールしたヘルプを調べる方法、Mobile および Embedded 開発のヘルプを追加する方法、およびヘルプのフィルタ処理をする方法について示しています。

Mobile および Embedded 開発のヘルプ セクションがインストールされているかどうかを確認するには

- 「Pocket PC 開発者ガイド」をクリックします。

  トピックが表示される場合、Mobile および Embedded 開発セクションはインストールされています。

  トピックが見つからない場合、次の手順に従って Mobile および Embedded 開発セクションを追加できます。

Mobile および Embedded 開発トピックを追加するには

1. Windows の [コントロール パネル] の [プログラムの追加と削除] をダブルクリックします。
2. [Visual Studio 2005 MSDN ライブラリ] を選択し、[変更] をクリックします。
3. セットアップ ウィザード の [ようこそ] ページで、[次へ] をクリックします。
4. [プログラムのメンテナンス] ページで、[変更] をクリックします。
5. [カスタム セットアップ] ページで、[モバイルおよび Embedded 開発] を選択します。
6. ドロップダウン ボタンから使用可能なショートカット メニューで、[この機能は、ローカル ハード ドライブにすべてインストールされます。] を選択します。
7. [次へ] をクリックします。
8. [プログラム変更の準備完了] ページで、[インストール] をクリックします。

ヘルプ トピックのフィルタ処理

Visual Studio には、目次および索引のスマート デバイス開発フィルタが用意されています。このフィルタを適用すると、表示されるトピックが減り、スマート デバイス開発者にとって基本と思われるドキュメントだけを参照できるようになります。このフィルタの適用は、.NET Compact Framework や MFC ライブラリおよび ATL ライブラリなど、実行時の参照トピックのスマート デバイスに関する部分だけを表示するのに最適な方法です。

言語フィルタ (Visual C# など) を適用した場合、表示されるトピックからスマート デバイス開発のトピックが除外されます。このため、スマート デバイス開発フィルタを使用するか、フィルタをまったく使用しないかのどちらかにすることをお勧めします。Visual C# 開発設定などの言語開発設定を使用して Visual Studio にログオンした場合は、既定で言語フィルタが割り当てられる点にも注意してください。

スマート デバイス開発フィルタを設定するには

1. Visual Studio の [ヘルプ] メニューで、[目次] または [キーワード] をクリックします。
2. [フィルタ条件] ボックスで、[スマート デバイス開発] を選択します。

開発設定を変更するには

1. Visual Studio の [ツール] メニューで、[設定のインポートとエクスポート] をクリックします。
2. ウィザードの [ようこそ] ページで、[すべての設定をリセット] をクリックし、[次へ] をクリックします。

   選択しているオプションがわからない場合、F1 キーを押してウィザードのヘルプ トピックを開きます。
3. [現在の設定の保存] ページで、現在の設定を保存するかどうかを選択し、[次へ] をクリックします。
4. [設定の既定のコレクションの選択] ページで、[全般的な開発設定] を選択し、[完了] をクリックします。

参照

処理手順

方法 : .NET Compact Framework のクラス ライブラリを使用する

概念

Visual Studio のヘルプ フィルタ

その他の技術情報

スマート デバイス プロジェクトの概要

方法：Visual Studio でデバイス エミュレータを起動する

デバイス エミュレータは、スマート デバイス プロジェクトの開発サイクルの大部分で、物理デバイスの代わりとして機能します。デバイス エミュレータでは、RAM サイズの指定などの多くのオプションがサポートされますが、すべての SDK ですべてのオプションがサポートされるわけではありません。詳細については、SDK のドキュメントを確認してください。

詳細については、デバイス エミュレータを起動し、[ヘルプ] をクリックしてください。

[デバイスへの接続] ダイアログ ボックスを使用してデバイス エミュレータを起動するには

1. Visual Studio で、[ツール] メニューの [デバイスへの接続] をクリックします。
2. [デバイスへの接続] ダイアログ ボックスで、[デバイス] ボックスからエミュレータを選択し、[接続] をクリックします。

デバイス エミュレータ マネージャを使用してデバイス エミュレータを起動するには

1. Visual Studio の [ツール] メニューの [デバイス エミュレータ マネージャ] をクリックします。
2. [デバイス エミュレータ マネージャ] ウィンドウで、起動するエミュレータを右クリックします。
3. ショートカット メニューの [接続] をクリックします。

参照

その他の技術情報

スマート デバイス プロジェクトの概要

以前のツールで作成されたプロジェクトの更新

Microsoft Visual Studio 2005 には、スマート デバイス開発者用の強力な開発ツールが用意されています。C# および Visual Basic を使用するプログラマは、Visual Studio 2005 に用意されている改良されたツールを活用できるようになりました。さらに、デスクトップ アプリケーションの作成に使用される環境と同じ開発環境を使用して、Pocket PC、Smartphone、または Windows CE デバイス用の C++ プロジェクトを作成できるようになりました。

Visual Studio 2005 開発環境での改良点は次のとおりです。

- C# ソース コードのリファクタリング ツール。詳細については、「リファクタリング」を参照してください。
- 改良されたデバッグ。詳細については、「デバイス プロジェクトのデバッグ」を参照してください。
- C# および Visual Basic 用のコード スニペット。詳細については、「IntelliSense コード スニペットの作成と使用」を参照してください。
- 改良された配置ツール。詳細については、「デバイス ソリューションをパッケージ化する方法の概要」を参照してください。
- 改良されたスマート デバイス エミュレータ。詳細については、「方法 : Visual Studio でデバイス エミュレータを起動する」を参照してください。

eMbedded Visual C++ から Visual Studio 2005 へのプロジェクトの移行

移行ウィザードを使用して、eMbedded Visual C++ プロジェクトを移行できます。詳細については、「eMbedded Visual C++ から Visual Studio 2005 へのアップグレード ウィザード」を参照してください。

eMbedded Visual Basic から Visual Studio 2005 へのプロジェクトの移行

eMbedded Visual Basic (eVB) で作成されたプロジェクトは、自動的には Visual Studio 2005 プロジェクトに変換されません。代わりに、Visual Studio 2005 で作成された新しい Visual Basic スマート デバイス プロジェクトに既存のソース ファイルとリソース ファイルを追加する必要があります。

メモ :
Windows Mobile 2003 デバイスの ROM には eMbedded Visual Basic ランタイムが含まれておらず、ランタイムは RAM インストールとしてデバイスにダウンロードできますが、この構成はサポートされません。

Visual Studio 2005 を使用して eVB プロジェクトを変換する方法の詳細については、次のトピックを参照してください。

- Visual Basic .NET への eVB ファイル コントロールの移行
- Visual Basic .NET への eVB フォームの移行

Visual Studio .NET 2003 から Visual Studio 2005 へのマネージ プロジェクトの移行

Visual Studio .NET 2003 で開発された Visual C# および Visual Basic スマート デバイス プロジェクトは、Visual Studio 2005 にインポートできます。Visual Studio 変換ウィザードでは、必要な変更をプロジェクトに加えることができます。

2002 デバイスから 2003 デバイスへのプロジェクトの更新

Windows Mobile 2000 から Windows Mobile 2003 にプロジェクトを更新する方法の詳細については、「Windows Mobile Platform Migration FAQ for Developers」を参照してください。

参照

概念

デバイス機能と必要な開発ツール

その他の技術情報

スマート デバイス プロジェクトの概要

Migrating to the eVC 4.0 Environment

スマート デバイス開発での操作方法

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

Visual Studio 2005 では、Pocket PC や Smartphone など、Windows CE ベースや Windows Mobile ベースのスマート デバイスを実行するソフトウェアを開発するために、統合的で充実したサポートを実現しています。.NET Compact Framework で実行するマネージ アプリケーションを記述するには、Visual C++ または Visual Basic を使用します。また、ネイティブ アプリケーションを記述するには Visual C++ も使用できます。どちらに言語を選択しても、デスクトップ コンピュータの開発と同じコード エディタ、デザイナ、デバッガのインターフェイスを使用できます。選択した言語で使用できるスマート デバイス プロジェクトのいずれかを選択し、コーディングを始めるだけです。

また、Visual Studio にはエミュレータ機能があります。そのため、物理デバイスがなくても開発コンピュータでコードの実行やデバッグを行うことができます。

概要 (スマート デバイスでの操作方法)

Windows フォーム アプリケーション、デバイス エミュレータの起動、各ツールでサポートされているバージョン、開発言語の選択、ヘルプ フィルタの使用などについて説明します。

デバイス接続 (スマート デバイスでの操作方法)

Virtual PC セッション、DMA、Bluetooth、ActiveSync が使用できない場合、トラブルシューティングなどについて説明します。

Visual Basic と Visual C# (スマート デバイスでの操作方法)

プロジェクトの作成、ソースの共有、プラットフォームの変更、コード スニペット、既定のターゲットの変更などについて説明します。

Visual C++ (スマート デバイスでの操作方法)

C++ デバイス プロジェクトの作成、eMbedded Visual C++ の移行、SQL Mobile Edition/SQL Server Compact Edition データベースの追加、マルチプラットフォーム ソリューションの開発、MFC ActiveX ホストの作成などについて説明します。

デバッグ (スマート デバイスでの操作方法)

プロセスへのアタッチ、併用型ソリューションのデバッグ、デバイス レジストリの変更などについて説明します。

データ (スマート デバイスでの操作方法)

データベースの作成と管理、プロジェクトへのデータ ソースの追加、クエリの作成、マスター/詳細リレーションシップの処理、ResultSet と DataSet の作成などについて説明します。

パッケージ化 (スマート デバイスでの操作方法)

CAB プロジェクトの作成、ショートカットの作成、デバイス レジストリの編集などについて説明します。

セキュリティ (スマート デバイス開発での操作方法)

証明書のインポート、セキュリティ モデルの照会、ファイルへの署名、デバイスの用意などについて説明します。

参照

その他の技術情報

スマート デバイス開発

概要 (スマート デバイスでの操作方法)

ここでは、スマート デバイス アプリケーション開発の概要に関するヘルプへのリンクを示します。

スマート デバイス プロジェクトの新機能

Visual Studio 2005 でのデバイス アプリケーション開発で使用できる新機能と拡張機能について説明します。

デバイス機能と必要な開発ツール

スマート デバイスのさまざまなバージョンおよびリリースと、それぞれをサポートするために必要なツールについて説明します。

デバイス プロジェクトのリモート ツール

デバイス アプリケーション開発で使用する特殊なツールの一覧と、それぞれについての詳しいヘルプ トピックへのリンクを示します。

方法 : スマート デバイス開発用のヘルプを最適化する

デバイス プロジェクトに関するヘルプの上手な使い方を示します。フィルタの使用方法についても説明します。

方法 : Visual Studio でデバイス エミュレータを起動する

物理デバイスをエミュレートするときに使用する標準ツールの起動方法について説明します。

以前のツールで作成されたプロジェクトの更新

以前のバージョンやツールで開発したプロジェクトの移植方法について説明します。

開発言語の選択

デバイス プロジェクトの開発言語という観点から、Visual Basic、Visual C#、Visual C++ を比較します。

チュートリアル : デバイス対応の Windows フォーム アプリケーションの作成

Windows フォーム アプリケーションを開発し、それをデバイス エミュレータ上で実行するための詳細な手順を示します。

参照

概念

スマート デバイス開発での操作方法

Visual Studio の概要

その他の技術情報

Visual Studio の統合開発環境

デバイス接続 (スマート デバイスでの操作方法)

このページには、デバイス接続の一般的なタスクに関するヘルプへのリンクが含まれています。

方法 : Virtual PC セッションからデバイス エミュレータに接続する

TCP/IP を使用せずに接続する方法について説明します。

方法 : Smartphone エミュレータ ファイル システムにアクセスする

独自のファイル ビューアを持たない、Smartphone エミュレータのファイル システムにアクセスする方法について説明します。

方法 : Bluetooth を使用して接続する

Bluetooth を使用して接続する方法について説明します。

方法 : IR を使用して接続する

赤外線を使用して接続する方法について説明します。

方法 : ActiveSync を使用せずに Windows CE デバイスに接続する

ActiveSync サービスを使用できない場合にデバイスに接続する方法について説明します。

方法 : エミュレータから開発用コンピュータのファイルにアクセスする

共有フォルダを使用して、エミュレータから開発用コンピュータのファイルにアクセスする方法について説明します。

方法 : 接続オプションの設定 (デバイス)

接続オプションを設定するためのコモン ダイアログ ボックスの場所について説明します。

参照

処理手順

接続のトラブルシューティング (デバイス)

概念

スマート デバイス開発での操作方法

接続方法の選択

Visual Basic と Visual C# (スマート デバイスでの操作方法)

ここでは、.NET Compact Framework を使用した一般的な開発タスクについてのヘルプへのリンクを示します。

方法 : Visual Basic または Visual C# を使用してデバイス アプリケーションを作成する

デバイス アプリケーションの作成方法と、このプロセスとデスクトップ アプリケーション開発との相違点について説明します。

方法 : プラットフォーム間でソース コードを共有する (デバイス)

コンパイラ定数を使用してさまざまなプラットフォーム間で同じソース コードを共有する方法について説明します。

方法 : デバイス プロジェクトのプラットフォームを変更する

同じプロジェクト内でプラットフォームを切り替える方法について説明します。

方法 : プロジェクトを新しいバージョンの .NET Compact Framework にアップグレードする

新しいバージョンのプラットフォームがインストールされた場合に、既存のプロジェクトのプラットフォームをアップグレードする方法について説明します。

デバイス プロジェクト内のコード スニペットの管理

デバイス プロジェクトだけに関係するコード スニペットの使用方法について説明します。

方法 : デバイス プロジェクトのコードがプラットフォームでサポートされているか検査する

コードが対象プラットフォームでサポートされているかどうかを確認する方法について説明します。

方法 : HardwareButton イベントを処理する (デバイス)

Pocket PC 上のアプリケーション キーをオーバーライドする方法について説明します。

方法 : フォームの向きおよび解像度を変更する (デバイス)

既定値が不正または欠けている場合に、フォームの向きと解像度を変更する方法について説明します。

方法 : 既定のデバイスを変更する (マネージ プロジェクト)

プロジェクトの開発中に対象デバイスを変更する方法について説明します。

方法 : スマート デバイス開発用のヘルプを最適化する

スマート デバイス ヘルプのフィルタを使用して、デバイス アプリケーション開発でサポートされている .NET Framework 要素のみを表示する方法について説明します。

参照

関連項目

データ (スマート デバイスでの操作方法)

概念

スマート デバイス開発での操作方法

.NET Compact Framework の使用方法のトピック

その他の技術情報

.NET Compact Framework を使用したデバイスのプログラミング

.NET Compact Framework

マネージ デバイス プロジェクトのデータ

Visual C++ (スマート デバイスでの操作方法)

このページには、スマート デバイスの C++ タスクに関するヘルプへのリンクがあります。

全般

デスクトップ プロジェクトのデバイス サポート

デスクトップなどの複数プラットフォームをターゲットとするのに役立つ、関連トピックの一覧を示します。

ネイティブ デバイス プロジェクトでのリソース編集

デバイスおよびデスクトップのリソース エディタ間の類似点と相違点について説明します。

方法 : ネイティブ デバイス プロジェクトにデータベースを追加する

SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースを含む C++ スマート デバイス アプリケーションを開発する方法について説明します。

方法 : プライマリ プロジェクト出力のリモート パスを指定する

リモート パスを設定する方法について説明します。

方法 : 既定のデバイスを変更する (ネイティブ プロジェクト)

C++ デバイス プロジェクトの既定のターゲットを変更する方法について説明します。

作成と移行

方法 : 新しい Visual C++ デバイス プロジェクトを作成する

デバイスの C++ アプリケーション、およびデスクトップで実行されるアプリケーションの作成について説明します。

方法 : マルチプラットフォーム対応のデバイス プロジェクトを作成する (Visual C++)

同じ C++ 開発プロジェクトから複数のデバイスをターゲットとする方法について説明します。

eVC からの移植に関する既知の問題

eMbedded Visual C++ 4.0 プロジェクトを Visual Studio に移行する方法について説明します。

MFC

チュートリアル : スマート デバイス用マルチプラットフォーム MFC アプリケーションの作成

複数プラットフォームを対象とする場合について重点的に説明します。

チュートリアル : スマート デバイス用 MFC マルチプラットフォーム ActiveX コントロールの作成

デバイスの MFC を使用した ActiveX コントロールの作成について重点的に説明します。

方法 : デバイスでサポートされる MFC クラスおよびメソッドのヘルプを検索する

スマート デバイス ヘルプのフィルタを使用して、デバイス プロジェクトでサポートされている MFC 要素のみを表示する方法について説明します。

ATL

チュートリアル : スマート デバイス用 ATL マルチプラットフォーム ActiveX コントロールの作成

デバイスのATL を使用した ActiveX コントロールの作成について重点的に説明します。

方法 : デバイスでサポートされる ATL クラスおよびメソッドのヘルプを検索する

スマート デバイス ヘルプのフィルタを使用して、デバイス プロジェクトでサポートされている ATL 要素のみを表示する方法について説明します。

参照

概念

スマート デバイス開発での操作方法

その他の技術情報

Visual C++

デバッグ (スマート デバイスでの操作方法)

ここでは、スマート デバイスのデバッグ タスクの概要と、デスクトップ プロジェクトで同様のタスクを行うときとの違いについて説明するリンクを示します。

デバイスのデバッグとデスクトップのデバッグとの違い

サポート外の機能、またはサポートが一部制限されている機能の一覧を示します。さらに、デバイス デバッグ用の cordbg.exe の追加機能について説明します。

方法 : マネージ デバイスのプロセスに接続する

デバッガを使用せずに実行中のプロセスにアタッチする方法について説明します。

方法 : デバイスのレジストリ設定を変更する

リモート レジストリ エディタを使用してデバイスのレジストリ設定を編集する方法について説明します。

チュートリアル : マネージ コードとネイティブ コードの両方を含むソリューションのデバッグ

このような併用型ソリューションのデバッグ方法について説明します。

参照
概念
スマート デバイス開発での操作方法
その他の技術情報
Visual Studio でのデバッグ

データ (スマート デバイスでの操作方法)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

ここでは、スマート デバイスのデータ処理タスクに関するヘルプへのリンクを示します。

全般

方法 : SqlCeResultSet コードを生成する (デバイス)

データセットの代わりに結果セットを生成する方法について説明します。

方法 : デザイン時の接続文字列を変更する (デバイス)

Visual Studio がデザイン時に SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースへの接続に使用する文字列を変更する方法について説明します。

方法 : 実行時の接続文字列を変更する (デバイス)

アプリケーションが実行時に SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースへの接続に使用する文字列を変更する方法について説明します。

方法 : ナビゲーション ボタンを追加する (デバイス)

.NET Compact Framework ではサポートされていない **DataNavigator** クラスの代替手法について説明します。

方法 : データベースのデータの変更を永続化する (デバイス)

データセットの変更内容をデータベースに適用する方法について説明します。

データ ソースの追加

方法 : データベースを作成する (デバイス)

Visual Studio 環境を使用してプロジェクトの内部または外部に SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースを作成する方法について説明します。

方法 : デバイス プロジェクトにデータベースを追加する

サーバー エクスプローラで使用できる SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースを Visual Basic または Visual C# プロジェクトのデータ ソースとして追加する方法について説明します。

方法 : SQL Server データベースをデータ ソースとして追加する (デバイス)

SQL Server データベースを Visual Basic または Visual C# プロジェクトのデータ ソースとして追加する方法について説明します。

方法 : ビジネス オブジェクトをデータ ソースとして追加する (デバイス)

ビジネス オブジェクトを Visual Basic または Visual C# プロジェクトのデータ ソースとして追加する方法について説明します。

方法 : Web サービスをデータ ソースとして追加する (デバイス)

Web サービスを Visual Basic または Visual C# プロジェクトのデータ ソースとして追加する方法について説明します。

データ ソースの管理

方法 : データベースのテーブルを管理する (デバイス)

テーブルを追加および削除する方法と、既存のテーブル スキーマを編集する方法について説明します。

方法 : データベースの列を管理する (デバイス)

列を追加および削除する方法と、列のプロパティを編集する方法について説明します。

方法 : データベースのインデックスを管理する (デバイス)

インデックスを追加および削除する方法と、インデックスの並べ替え順序のプロパティを変更する方法について説明します。

方法 : データベースのパスワードを管理する (デバイス)

新しい SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースのパスワードを設定する方法と、既存のデータベースのパスワードを変更する方法について説明します。

方法 : データベースを縮小および修復する (デバイス)

SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースを縮小および修復する方法について説明しま

す。

クエリの作成

方法 : パラメータ付きクエリを作成する (デバイス)

パラメータ クエリの作成方法について説明します。

チュートリアル : パラメータ クエリ アプリケーション

パラメータ クエリの作成を含む End-to-end プロジェクトの詳細な手順を示します。

マスター/詳細リレーションシップの処理

方法 : マスター/詳細アプリケーションを作成する (デバイス)

マスター/詳細リレーションシップの実装方法について説明します。

チュートリアル : データベース マスター/詳細アプリケーション

マスター/詳細アプリケーションの作成と実行を含む End-to-end プロジェクトの詳細な手順を示します。

データの表示と編集

方法 : データベース内のデータをプレビューする (デバイス)

データベース内のデータを表示するためのいくつかのオプションについて説明します。

方法 : データ アプリケーション用の概要ビューと編集ビューを生成する (デバイス)

データ フォームを使用して、データ グリッドの単一データ行を表示および編集する方法について説明します。

参照

概念

スマート デバイス開発での操作方法

その他の技術情報

マネージ デバイス プロジェクトのデータ

Overview of SQL Server 2005 Mobile Edition

データへのアクセス (Visual Studio)

パッケージ化 (スマート デバイスでの操作方法)

このページには、デバイス プロジェクトのタスクをパッケージ化し、セット アップするために役立つページのリンクが含まれています。

チュートリアル : 配置用のスマート デバイス ソリューションのパッケージ化

アプリケーションとそのリソースをパッケージ化する詳細な手順を示します。

参照

概念

スマート デバイス開発での操作方法
デバイス アプリケーションのパッケージ化をサポートする IDE 機能
デバイス ソリューションをパッケージ化する方法の概要

セキュリティ (スマート デバイス開発での操作方法)

このページには、スマート デバイスの共通のセキュリティ タスクに関するヘルプへのリンクが含まれています。

全般

方法 : デバイス プロジェクトで証明書をインポートおよび適用する

デバイス プロジェクトの署名に、[証明書の選択] ダイアログ ボックスを効率的に使用する方法について説明します。

方法 : Signtool.exe をビルド後のイベントとして起動する (デバイス)

ビルド後のイベントにより元のバイナリが変更される際に、プロジェクトに署名する方法について説明します。

方法 : デバイスのセキュリティ モデルを照会する

デバイス証明書ストアに既にインストールされている証明書を調べる方法について説明します。

署名

方法 : Visual Basic アプリケーションまたは Visual C# アプリケーションに署名する (デバイス)

.NET Compact Framework に対して記述されたアプリケーションに署名する手順を示します。

方法 : Visual Basic アセンブリまたは Visual C# アセンブリに署名する (デバイス)

プロジェクト アセンブリに署名する手順を示します。

方法 : Visual C++ プロジェクトでプロジェクトの出力に署名する (デバイス)

Visual C++ プロジェクト出力に署名する手順を示します。

方法 : CAB ファイルに署名する (デバイス)

デバイス CAB プロジェクトに署名する手順を示します。

用意

方法 : Visual Basic プロジェクトまたは Visual C# プロジェクトでデバイスを用意する

マネージ プロジェクトのデバイス ストアにデジタル証明書を追加する手順を示します。

方法 : Visual C++ プロジェクトでデバイスを用意する

ネイティブ プロジェクトのデバイス ストアにデジタル証明書を追加する手順を示します。

方法 : セキュリティ モデルを使用したデバイスを用意する

RapiConfig.exe を使用して、セキュリティ モデルによってデバイスを用意する方法について説明します。

参照

概念

スマート デバイス開発での操作方法

その他の技術情報

ネイティブ コードと .NET Framework コードのセキュリティ

スマート デバイス開発のデザイン

Visual Studio 2005 では、スマート デバイス開発用に 3 つのプログラム言語とさまざまなプロジェクトの種類を使用できるようになりました。ここでは、さまざまなプロジェクトの種類の概要と、プロジェクトに適した言語を選択するためのアドバイスを示します。

このセクションの内容

スマート デバイス プロジェクトの種類の選択

Visual Studio 2005 で使用できるスマート デバイス プロジェクトの多様な種類と、サポートされるさまざまなハードウェア プラットフォームの概要について説明します。

開発言語の選択

プロジェクトに最適な開発言語を選択するためのアドバイスを示します。

スキンのカスタマイズ (デバイス)

スマート デバイス プロジェクトのスキンの概要と、そのカスタマイズ方法について説明します。

参照

その他の技術情報

スマート デバイス開発
スマート デバイスのプログラミング
Mobile Developer Center

スマート デバイス プロジェクトの種類の選択

新しいスマート デバイス プロジェクトを作成する場合、Visual Studio 2005 には次のプロジェクト テンプレートが用意されています。"(1.0)" としてマークされたテンプレートは、.NET Compact Framework 1.0 に基づくプロジェクトであることを示します。残りのテンプレートは、バージョン 2.0 に基づくプロジェクトです。

Visual C# と Visual Basic

テンプレート名	サポートされるデバイス	説明
デバイス アプリケーション	Pocket PC 2003、Windows CE 5.0	.NET Compact Framework 2.0 Windows フォーム アプリケーションを作成するプロジェクト。
コントロール ライブラリ	Pocket PC 2003、Windows CE 5.0	.NET Compact Framework 2.0 コントロールを作成するプロジェクト。
クラス ライブラリ	Pocket PC 2003、Windows CE 5.0	.NET Compact Framework 2.0 クラス ライブラリ (DLL) を作成するプロジェクト。
コンソール アプリケーション	Pocket PC 2003、Windows CE 5.0	.NET Compact Framework 2.0 の GUI がないアプリケーションを作成するプロジェクト。
空のプロジェクト	Pocket PC 2003、Windows CE 5.0	.NET Compact Framework 2.0 アプリケーションを作成する空のプロジェクト。
デバイス アプリケーション (1.0)	Pocket PC 2003、Smartphone 2003	.NET Compact Framework 1.0 Windows フォーム アプリケーションを作成するプロジェクト。
クラス ライブラリ (1.0)	Pocket PC 2003、Smartphone 2003	.NET Compact Framework 1.0 クラス ライブラリ (DLL) を作成するプロジェクト。
コンソール アプリケーション (1.0)	Pocket PC 2003、Smartphone 2003	.NET Compact Framework 1.0 の GUI がないアプリケーションを作成するプロジェクト。
空のプロジェクト (1.0)	Pocket PC 2003、Smartphone 2003	.NET Compact Framework 1.0 アプリケーションを作成する空のプロジェクト。

Visual C++

テンプレート名	サポートされるデバイス	コメント
ATL スマート デバイス プロジェクト	Pocket PC 2003、Smartphone 2003、Windows CE 5.0	Active Template Library を使用したアプリケーションを作成するプロジェクト。
MFC スマート デバイス プロジェクト	Pocket PC 2003、Smartphone 2003、Windows CE 5.0	MFC (Microsoft Foundation Class) ライブラリを使用したアプリケーションを作成するプロジェクト。
MFC スマート デバイス ActiveX コントロール	Pocket PC 2003、Smartphone 2003、Windows CE 5.0	Microsoft Foundation Class ライブラリを使用した ActiveX コントロールを作成するプロジェクト。
MFC スマート デバイス DLL	Pocket PC 2003、Smartphone 2003、Windows CE 5.0	Microsoft Foundation Class ライブラリを使用したダイナミック リンク ライブラリを作成するプロジェクト。
Win32 プロジェクト	Pocket PC 2003、Smartphone 2003、Windows CE 5.0	Win32 アプリケーションを作成するプロジェクト。

その他のプロジェクトの種類

テンプレート名	サポートされるデバイス	コメント
スマート デバイス CAB プロジェクト	Pocket PC 2003、Smartphone 2003、Windows CE 5.0	スマート デバイス アプリケーションを配置するための CAB ファイルを作成するプロジェクト。

参照

概念

開発言語の選択

開発言語の選択

スマート デバイスで配置するアプリケーション、コントロール、またはライブラリを開発するときに、選択できるプログラム言語は、Visual C#、Visual Basic、および Visual C++ の 3 つです。

Visual C#

C# は最新のオブジェクト指向言語です。ガベージ コレクション機能があり、.NET Compact Framework クラスがサポートされているため、信頼性の高い安全なモバイル アプリケーションを開発するときに最適な言語です。Visual C# for Smart Devices には、グラフィカル ユーザー インターフェイス (GUI) を簡単に作成できる数多くのコントロールが含まれ、Compact Framework クラスでは、GDI+、XML、および Web サービスなどの機能がサポートされます。Visual C# では、.NET Compact Framework でサポートされていない状況でも、ネイティブの Windows CE 関数を呼び出すことができます。

Visual C# を使用して開発する方法とネイティブ Windows CE 関数にアクセスする方法の詳細については、以下を参照してください。

- Getting Started with Visual Studio .NET and the Microsoft .NET Compact Framework
- C# リファレンス
- An Introduction to P/Invoke and Marshaling on the Microsoft .NET Compact Framework
- Creating a P/Invoke Library
- Advanced P/Invoke on the Microsoft .NET Compact Framework

Visual Basic

Visual Basic for Smart Devices には、Visual Basic が完全実装され、以前の開発ツールである eMbedded Visual Basic よりも大幅に機能強化されています。Visual Basic では、デスクトップ アプリケーションのモバイル デバイスへの移植タスクが単純化され、リッチ クライアント アプリケーションを簡単に作成できるようになりました。Visual C# と同様に、Visual Basic では .NET Compact Framework を利用しています。Visual Basic の開発経験があれば、既存アプリケーションの移植や新しいアプリケーションの作成を簡単に行うことができます。また C# と同様に、Visual Basic ではネイティブ Windows CE 関数にアクセスできます。

Visual Basic で開発する方法の詳細については、以下を参照してください。

- Getting Started with Visual Studio .NET and the Microsoft .NET Compact Framework
- Visual Basic リファレンス

Visual C++

スマート デバイスで Visual C++ を使用した開発が優先されるのは、パフォーマンスが重要な場合や、システム レベルのアプリケーション、デバイス ドライバ、[今日] 画面のプラグイン、または [ホーム] 画面のプラグインを開発する場合です。Visual C++ では .NET Compact Framework はサポートされませんが、Win32 API セットのサブセットとして機能します。マネージ C# コードまたは Visual Basic コードで記述されたアプリケーションから、相互運用機能を利用して、DLL に含まれる C++ コードにアクセスできます。

Visual C++ で開発する方法の詳細については、以下を参照してください。

- C/C++ Languages

参照

その他の技術情報

スマート デバイス プロジェクトの概要

スキンのカスタマイズ (デバイス)

スキンは、デバイス エミュレータまたは Visual Studio デザイナでアプリケーションの四角形のフォーム、つまりビューポートを囲むグラフィックです。アプリケーションの開発中にスキンを使用すると、そのアプリケーションが実際のデバイスでどのように表示されるかをうまく視覚化できます。Visual Studio デザイナとデバイス エミュレータでは、同じスキンを使用できます。

スキンでは、見た目をよくするだけではなく、ハードウェアのボタンやソフトキーに対するマウス イベントの処理などの機能も用意しています。

メモ :
Visual Studio と共にインストールされたスキン ファイルが壊れた場合は、Visual Studio 2005 のインストールを修復することによって、これらのスキン ファイルを再インストールできます。詳細については、「方法 : Visual Studio を登録する」を参照してください。

このセクションの内容

スキンのテクノロジ (デバイス)

XML スキン定義ファイル、およびスキンのグラフィック ファイルの要件について説明します。

方法 : スキン ファイルを作成する (デバイス)

簡単なスキンを作成する手順について説明します。

方法 : スキンの視覚的な特性を変更する (デバイス)

エミュレータのタイトル バー、ビューポートの場所と大きさ、スキンを枠内に表示するかどうか、または透過を設定するために、スキン定義ファイルに対して行うことができる変更について説明します。

方法 : マウス イベントを処理する (デバイス)

カラー マップを使用して、マウス イベントに応答できるスキンの領域を指定する方法について説明します。この領域はホットスポットとも呼ばれます。

方法 : ツールヒントを表示する (デバイス)

スキンのホットスポットにツールヒントを指定する方法について説明します。

方法 : デバイス エミュレータでスキンを使用する

デバイス エミュレータのユーザー インターフェイスで使用できるオプションについて説明します。このオプションには、スキンの選択、スキンを OS イメージと共に保存する方法、およびツールヒントの表示/非表示を切り替える方法などがあります。

方法 : Visual Studio 2005 でスキンを使用する (デバイス)

Visual Studio のユーザー インターフェイスで使用できるオプションについて説明します。このオプションには、既定値の設定、解像度の指定、回転の有効化、その他のプロパティなどがあります。

スキン定義ファイルの詳細 (デバイス)

スキン定義ファイルの個別の要素 (必須要素と省略可能な要素) の値について説明し、サンプル値を示します。

スキン定義ファイルの例 (デバイス)

完全なスキン定義ファイルを示します。

参照

その他の技術情報

スマート デバイス開発のデザイン

Mobile Developer Center

スキンのテクノロジ (デバイス)

スキンは、最大 3 つのビットマップ (BMP: Bitmap) ファイルまたは PNG (Portable Network Graphics) ファイルと、拡張マークアップ言語 (XML: Extensible Markup Language) を含む 1 つのスキン定義ファイルで構成されます。スキン定義ファイルとそれに関連する BMP ファイルや PNG ファイルは、同じディレクトリに置く必要があります。

- XML ファイルには、スキンの動作と BMP ファイルまたは PNG ファイルの場所が記述されます。また、XML ファイルでは、マッピング ファイルに示されたカラー コードに基づいてボタンの動作も定義されます。詳細については、「スキン定義ファイルの例 (デバイス)」を参照してください。
- 1 つ目の BMP ファイルまたは PNG ファイルは標準アート ファイルと呼ばれ、エミュレータ スキンの既定の外観を示します。この標準アート ファイルは、スキンで唯一の必須ファイルです。
- 2 つ目の省略できる BMP ファイルまたは PNG ファイルはダウン アート ファイルと呼ばれ、すべてのボタンが押された状態のスキンの外観を示します。

3 つ目の省略できる BMP ファイルまたは PNG ファイルはマッピング ファイルと呼ばれ、スキンの各ボタンのカラー コードを示します。各ボタンのカラー コードは、マッピング ファイルでボタン全体を覆う 1 色の領域として表されます。スキンのユーザーには、マッピング ファイルのコードに使用する色が表示されません。

参照

その他の技術情報

スキンのカスタマイズ (デバイス)

方法：スキン ファイルを作成する (デバイス)

同じ手法を使用して、新しいスキンの作成または既存のスキンのカスタマイズを実行できます。OEM は、パフォーマンスの高いグラフィック ツールを使用し、スキンに必要なグラフィック ファイルをデザインおよび作成します。一方で、Microsoft ペイントのようなより簡単なツールを使用しても、満足のいく結果を得ることができます。スキン定義ファイル (XML ファイル) は、Microsoft メモ帳のように簡単なツールを使用して作成できます。

既存のスキンをカスタマイズするには、新しい名前で既存のスキン ファイルを保存して編集します。新しいスキンを使用する場合は、スキンのすべてのファイル、つまり、スキン定義ファイルおよびそれに付随するグラフィック ファイルで同じディレクトリを使用するようにします。

次の手順では、新しいスキンの作成方法について説明します。詳細については、「スキンのテクノロジ (デバイス)」を参照してください。

新しいスキンを作成するには

1. スキンの既定の外観を示すビットマップ (BMP: Bitmap) ファイルまたは PNG (Portable Network Graphics) ファイルを作成します。
2. すべてのボタンが押された状態のスキンの外観を示す BMP ファイルまたは PNG ファイルを作成します。
3. 各ボタンの領域を一意の 1 色で塗りつぶした BMP ファイルまたは PNG ファイルを作成します。

   これらのボタンの色は、ホットスポットを表し、マウス イベントを処理する際に使用されます。各ボタンの外観をスキンの他のボタンとは関係なく変更する場合は、別の色を使用して、各ボタンの領域を塗りつぶします。詳細については、「方法 : マウス イベントを処理する (デバイス)」を参照してください。

メモ：
見やすくするために、これらの領域の塗りつぶしには、白または黒を使用しないでください。

4. スキン定義ファイルを作成します。

   詳細については、「スキン定義ファイルの詳細 (デバイス)」および「スキン定義ファイルの例 (デバイス)」を参照してください。

5. 3 つの BMP ファイルまたは PNG ファイルと XML ファイルを 1 つのディレクトリに保存します。

参照

その他の技術情報

スキンのカスタマイズ (デバイス)

方法：スキンの視覚的な特性を変更する (デバイス)

スキンの視覚的な多くの特性を変更できます。

メモ：

ビューポートとは、アプリケーションのユーザー インターフェイスが表示される、スキン内のウィンドウ領域を指します。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

デバイス エミュレータのタイトル バーの内容を変更するには

- スキン定義ファイルで、`titleBar` 要素の内容を置き換えます。次に例を示します。

```
titlebar = "My Device"
```

スキン内のビューポートの位置を変更するには

- スキン定義ファイルで、x 軸と y 軸に割り当てられた値を変更します。

  たとえば、Pocket PC 2003 Second Edition の既定のスキーマでは、displayPosX="51" および displayPosY="47" が指定されています。`displayPosX` 値を大きくすると、ビューポートは右に移動します。`displayPoxY` 値を大きくすると、ビューポートは下に移動します。

ビューポートの高さと幅を変更するには

- スキン定義ファイルで、`displayWidth` 要素および `displayHeight` 要素に割り当てられた値を変更します。

  たとえば、Pocket PC 2003 Second Edition の既定のスキーマでは、`displayWidth="240"` および `displayHeight="320"` が指定されています。`displayWidth` 値を大きくすると、ビューポートは右方向に大きくなります。`displayHeight` 値を大きくすると、ビューポートは下方向に大きくなります。

ビューポートの色深度を変更するには

- スキン定義ファイルで、`displayDepth` 要素に割り当てられた値を変更します。

  たとえば、Pocket PC 2003 Second Edition の既定のスキーマでは、`displayDepth="16"` が指定されています。

  メモ：

  色深度 (ビットの深さとも呼ばれます) は、各ピクセルの色を定義するために使用できるビット数を表します。通常、スマート デバイスのスキンの色深度は 16 です。

背景の透明度を実装するには

- Up グラフィック ファイルまたは Normal グラフィック ファイルの左下のピクセルを透明色に設定します。このため、現在 Smartphone と Pocket PC 用に付属しているスキンのカラー値は FEFFFF に設定されています。

  メモ：

  この方法を使用すると、スキンの枠がなくなります。つまり、デバイス エミュレータと Visual Studio デザイナ内のグラフィックは、スキンの視覚要素によって関連付けられるため、スキンが表示されるウィンドウとは背景色が異なる四角形には表示されません。

参照

その他の技術情報

スキンのカスタマイズ (デバイス)

方法：マウス イベントを処理する (デバイス)

スキンは、実際のデバイスの視覚的なレプリカを示すためだけでなく、マウス イベントの処理にも使用できます。これにより、実際のデバイスのエミュレーションは、より実際の環境に近い状態で行うことができます。

スキン定義ファイルの各ボタン領域に一意な色 (*mappingColor*) を割り当てることにより、スキンのボタン上にカーソルを移動した場合の動作、ボタンをクリックした場合の動作、またはボタンをクリックしたままの場合の動作を指定できます。この色は、ユーザー インターフェイスには表示されません。デバイス エミュレータおよび Visual Studio デザイナでのイベント処理を示す一意なインジケータを指定する場合のみ機能します。

たとえば、グラフィック ツールを使用して PocketPC_2003_Mask.PNG ファイル (既定では、\Program Files\Microsoft Visual Studio 8\SmartDevices\Skins\PocketPC_2003\PocketPC_2003\1033 にインストールされています) を表示すると、各ボタンが異なる色で表示されていることがわかります。

onClick イベントを処理するには

1. スキン定義ファイルの button タグで、`mappingColor` にカラー値を代入します。

   Pocket PC 2003 のスキン定義ファイルから抜粋した例を次に示します。

   ```
   <button
       toolTip="Soft Key 1"
       onClick="DOWN:0x5b 0x70 UP:0x5b"
       mappingColor="0xF26C4F"
   />
   ```

2. `onClick` イベントにキーストロークを割り当てます。

   詳細については、ボタンをキーストロークに関連付ける手順を参照してください。

   カラー値が 0xF26C4F のボタンをクリックすると、この button セクションで指定された `onClick` イベントが処理されます。スキン定義ファイルで指定したキーストロークは、エンジンに渡されます。

onPressAndHold イベントを処理するには

1. スキン定義ファイルの button タグで、`mappingColor` にカラー値を代入します。

   Pocket PC 2003 のスキン定義ファイルから抜粋した例を次に示します。

   ```
   <button
       toolTip="Power"
       onPressAndHold="0x75"
       mappingColor="0xED145B"
   />
   ```

2. `onPressAndHold` イベントにキーストロークを割り当てます。

   詳細については、ボタンをキーストロークに関連付ける手順を参照してください。

   カラー値が 0xED145B のボタンをクリックすると、この button セクションで指定された `onPressAndHold` イベントが処理されます。

ボタンをキーストロークに関連付けるには

- 前の例のようにキーボードのスキャン コードを使用するか、Key_Down などの一連の定義済み定数を使用します。

  詳細については、MSDN オンライン ライブラリの「Emulator Skin XML Schema」を参照してください。

参照

その他の技術情報

スキンのカスタマイズ (デバイス)

方法：ツールヒントを表示する (デバイス)

スキン定義ファイルの各 button タグには、ツールヒントを表す行を含めることができます。スキンのホットスポット上にマウス ポインタを移動すると、ツールヒントが割り当てられている場合、そのツールヒントがポップアップ表示されます。ツールヒントでは、ホットスポットが表す機能に関する情報を示します。通常、ホットスポットは [ソフト キー 1] などのボタンです。ツールヒントには、スキン定義ファイルの ToolTip 要素に指定したボタン名が表示されます。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

スキンでツールヒントを表示するには

1. スキン定義ファイルの button タグで、`mappingColor` にカラー値を代入します。

   Pocket PC 2003 のスキン定義ファイルから抜粋した例を次に示します。

   ```
   <button
       toolTip="Soft Key 1"
       onClick="DOWN:0x5b 0x70 UP:0x5b"
       mappingColor="0xF26C4F"
   />
   ```

2. `toolTip` にテキスト文字列を割り当てます。

   カラー値が 0xF26C4F のボタン上にマウス ポインタを移動すると、ツールヒントとして Soft Key 1 というテキストが表示されます。

参照

その他の技術情報

スキンのカスタマイズ (デバイス)

方法：デバイス エミュレータでスキンを使用する

Microsoft Device Emulator には、スキン定義ファイルを選択するためのダイアログ ボックスに加え、コマンド ラインからエミュレータを起動する場合に使用するスキン オプションが用意されています。

エミュレータのスキン定義ファイルを選択するためのグラフィカル インターフェイスは、[エミュレータのプロパティ] ダイアログ ボックスの [表示] タブで使用できます。

一部のオプションは、コマンド ラインからデバイス エミュレータを起動する場合しか使用できません。詳細については、デバイス エミュレータの [ヘルプ] をクリックしてください。

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

デバイス エミュレータを開くには

1. Visual Studio で、[ツール] メニューの [デバイスへの接続] をクリックします。
2. [デバイスへの接続] ダイアログ ボックスで、エミュレータを選択し、[接続] をクリックします。

デバイス エミュレータから [エミュレータのプロパティ] ダイアログ ボックスを開くには

- デバイス エミュレータで、[ファイル] メニューの [構成] クリックし、[表示] タブをクリックします。

Visual Studio 2005 から [エミュレータのプロパティ] ダイアログ ボックスを開くには

1. Visual Studio で、[ツール] メニューの [オプション] をクリックします。
2. [デバイス ツール] ノードを展開し、[デバイス] をクリックします。
3. [デバイス] ボックスからエミュレータを選択し、[プロパティ] をクリックします。
4. [<EmulatorName> エミュレータ のプロパティ] ダイアログ ボックスで、[エミュレータ オプション] をクリックし、[表示] タブをクリックします。

デバイス エミュレータにスキンを適用するには

- [エミュレータのプロパティ] ダイアログ ボックスの [表示] タブで、[スキン] ボックスを使用します。

  または

- コマンド ラインからエミュレータを起動する場合は、**/skin** スイッチを使用します。

  または

- コマンド ラインから **/s** スイッチを使用して起動して、以前エミュレータの OS イメージと共に保存したスキンを適用します。

エミュレータの OS イメージと共にスキン ファイルを保存するには

- デバイス エミュレータで、[ファイル] メニューの [状態を保存して終了] をクリックします。

  スキンを含む保存状態ファイルが生成および保存されます。詳細については、エミュレータのヘルプの SavedState ファイルに関するトピックを参照してください。

ツールヒントの表示/非表示を切り替えるには

- [エミュレータのプロパティ] ダイアログ ボックスの [表示] タブで、[ツール ヒントの有効化] チェック ボックスをオンまたはオフにします。

メモ：
ツールヒントを表示できるのは、スキン定義ファイルでツールヒントが指定されている場合だけです。

参照

その他の技術情報

スキンのカスタマイズ (デバイス)

方法：Visual Studio 2005 でスキンを使用する (デバイス)

Visual Studio 2005 には、Express Edition を除いて、デザイナでのスキンの管理に役立つ多くのユーザー インターフェイス要素が用意されています。通常、これらの機能は、[フォーム ファクタ オプション] ダイアログ ボックスおよび [フォーム ファクタのプロパティ] ダイアログ ボックスにあります。

メモ：
必要に応じて、Visual Studio の環境を使用し、エミュレータおよび Visual Studio デザイナのスキン プロパティにアクセスできます。デバイス エミュレータを開く必要はありません。詳細については、「デバイス エミュレータのスキン プロパティにアクセスするには」を参照してください。

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

[フォーム ファクタ オプション] ダイアログ ボックスを開くには

1. Visual Studio で、[ツール] メニューの [オプション] をクリックします。
2. [デバイス ツール] を展開し、[フォーム ファクタ] をクリックします。

[フォーム ファクタのプロパティ] ダイアログ ボックスを開くには

- [フォーム ファクタ オプション] ダイアログ ボックスで、[フォーム ファクタ] を選択し、[プロパティ] をクリックします。

デザイナのスキンを選択するには

1. [フォーム ファクタ オプション] ダイアログ ボックスの [スキン] ボックスで、デザイナで使用するスキン定義ファイルを表す XML ファイルを選択します。

   使用するスキンが、[スキン] ボックスの左側にあるパネルにグラフィカルに表示されます。

2. [OK] をクリックして [フォーム ファクタ オプション] ダイアログ ボックスに戻ります。
3. [名前を付けて保存] をクリックし、この新しい配置状態を新しい名前で格納します。

   新しい名前のフォーム ファクタには、カスタマイズされたスキンが含まれます。

メモ：
プロジェクトを開いている場合は、変更を有効にするために、フォームを閉じてもう一度開く必要があります。

特定のスキンを既定のスキンとして設定するには

- [フォーム ファクタ オプション] ダイアログ ボックスの [既定のフォーム ファクタ] ボックスで、既定値として設定するフォーム ファクタを選択します。

  選択した既定のフォーム ファクタが新しいプロジェクトの既定のフォーム ファクタになります。

メモ：
この既定値は、プロジェクトを開いている場合のみ設定できます。

回転サポートを有効にするには

- [フォーム ファクタのプロパティ] ダイアログ ボックスで、[回転サポートを有効にする] を選択します。

水平解像度および垂直解像度を設定するには

- [フォーム ファクタのプロパティ] ダイアログ ボックスで、水平解像度と垂直解像度の両方に対して、1 インチあたりのピクセルの値を入力します。

スキンの表示/非表示を切り替えるには

- [フォーム ファクタのプロパティ] ダイアログ ボックスで、[スキンの表示] チェック ボックスをオンまたはオフにします。

  [スキンの表示] チェック ボックスをオンにした場合、スキンによって高さ、幅、および色深度が決まるため、これらのプロパティは淡色表示され、変更できなくなります。

ビューポートの高さと幅を設定するには

- [フォーム ファクタのプロパティ] ダイアログ ボックスの [画面の幅] ボックスおよび [画面の高さ] ボックスで、ピクセル数の値を入力します。

  [スキンの表示] チェック ボックスをオンにした場合、これらのプロパティは、スキンによって設定されるため、淡色表示され、変更できなくなります。

色深度を設定するには

- [フォーム ファクタのプロパティ] ダイアログ ボックスの [色の深度] ボックスで、1 ピクセルあたりのビット数を選択します。

  [スキンの表示] チェック ボックスをオンにした場合、このプロパティは、スキンによって設定されるため、淡色表示され、変更できなくなります。

ツールヒントを非表示にするには

- スキン定義ファイルから ToolTip 要素を削除します。

  メモ：

  Visual Studio には、ツールヒントが既にスキン定義ファイルで定義されている場合に、デザイナでそのツールヒントを非表示にするユーザー インターフェイス オプションがありません。このツールヒントの動作は、デバイス エミュレータに適用されたスキンの動作と区別してください。デバイス エミュレータでは、ツールヒントを有効または無効にできます。詳細については、「方法 : デバイス エミュレータでスキンを使用する」を参照してください。

デバイス エミュレータのスキン プロパティにアクセスするには

1. Visual Studio で、[ツール] メニューの [オプション] をクリックします。
2. [デバイス ツール] ノードを展開し、[デバイス] をクリックします。
3. [デバイス] ボックスからエミュレータを選択し、[プロパティ] をクリックします。
4. [<EmulatorName> エミュレータ のプロパティ] ダイアログ ボックスで、[エミュレータ オプション] をクリックし、[表示] タブを選択します。

   このダイアログ ボックスは、デバイス エミュレータで、[ファイル] メニューの [構成] をクリックしたときに表示されるダイアログ ボックスと同じです。詳細については、「方法 : デバイス エミュレータでスキンを使用する」を参照してください。

参照

その他の技術情報

スキンのカスタマイズ (デバイス)

スキン定義ファイルの詳細 (デバイス)

次の表では、デバイス用のスキン定義ファイルの要素および値のサンプルについて説明します。詳細については、「スキン定義ファイルの例 (デバイス)」を参照してください。

要素

XML タグと XML 要素	説明
<skin> タグ	エミュレータ スキンのスキーマをカプセル化します。各 XML ファイルで使用できる <skin> タグは 1 つだけです。
<view> タグ	エミュレータ スキンのスキーマを含みます。各 <skin> タグで使用できる <view> タグは 1 つだけです。
titleBar ="My Emulator skin" タイトル バー要素	エミュレータのウィンドウ タイトルを指定します。
displayPosX="10" 要素および displayPosY="149" 要素	エミュレータ スキンのウィンドウ内で、エミュレータの表示が含まれるウィンドウの位置を指定します。表示を不可視にする場合、画面に表示しない座標を選択します。
displayWidth="272" 要素および displayHeight="224" 要素	エミュレータを表示する幅と高さを指定します。幅には、8 で割り切れる 80 ～ 1024 の整数を選択します。高さには、64 ～ 768 の整数を選択します。
displayDepth="8" 要素	エミュレータを表示する色深度を指定します。色深度では 8、16、または 32 を選択します。
normalImage="up.bmp" 要素	必要に応じて、エミュレータ スキンに標準アート ファイルを指定します。標準アート ファイルは、エミュレータのウィンドウ サイズとエミュレータ スキンの外観を指定するファイルです。
mappingImage="map.bmp" 要素	エミュレータ スキンのマッピング ファイルを指定します。マッピング ファイルは、エミュレータ スキンに占めるボタンの領域を定義するオプションのファイルです。
downImage="down.bmp" 要素	エミュレータ スキンのダウン アート ファイルを指定します。ダウン アート ファイルは、エミュレータ スキンでボタンを押したときの外観を指定するオプションのファイルです。
<button> タグ	エミュレータ スキンのボタンの説明を含みます。
mappingColor="0x00FF00" 要素	ボタンに使用する、マッピング ファイルの RGB カラーを指定します。マッピング イメージでこの色が指定されたピクセルは、エミュレータ スキンでこのボタンをクリックできる領域を表します。この領域は、ボタンを押したときに表示するダウン アート ファイルのマスクとして機能します。
toolTip="This is my ToolTip." 要素	省略可能です。ボタン上にポインタを移動したときに表示するテキストを指定します。
onClick=" DOWN:Key_LeftShift Key_Z 0x00000015 UP: Key_LeftShift Key_A"	省略可能です。ボタンをクリックしたときにエンジンへ渡すキーボード入力を指定します。ロー キーボードのスキャン コードに対応する 16 進数または整数を指定します。

<button toolTip="Soft Key 1 " onPressAndHold="0x3 B" mappingColor="0xF26 C4F" KeyEvent="F1" Comment="Not handle d when menu is present."	エミュレータ スキンのボタンが押されている間に繰り返すキーボード イベントを指定します。KeyEvent を指定すると、デザイナで、ボタン用のコードが生成されます。Comment を指定すると、そのタグ内のテキストが生成されたコードにコメントとして追加されます。既定では、ツールヒントがコメントとして使用されます。 この機能では、SHUTDOWN キー コード以外のすべてのキー コードがサポートされています。

参照

その他の技術情報

スキンのカスタマイズ (デバイス)

スキン定義ファイルの例 (デバイス)

縦長形式の Pocket PC 2003 のスキンを表す PocketPC_2003_Skin.xml というスキン定義ファイルのコードを次に示します。Visual Studio 2005 (Express Edition を除く) では、既定で、このスキン ファイルおよびその他のスキン ファイルが \Program Files\Microsoft Visual Studio 8\SmartDevices\Skins にインストールされます。

デバイス エミュレータおよび Visual Studio デザイナでは、同じスキン ファイルを使用できます。詳細については、「スキン定義ファイルの詳細 (デバイス)」を参照してください。

コード

```
<?xml version="1.0" encoding="UTF-8" ?>
<skin>
    <view
        titleBar ="Pocket PC 2003 Second Edition"
        displayPosX="51"
        displayPosY="47"
        displayWidth="240"
        displayHeight="320"
        displayDepth="16"
        mappingImage="PocketPC_2003_Mask.png"
        normalImage="PocketPC_2003_Up.png"
        downImage= "PocketPC_2003_Down.png">
        <button
            toolTip="Power"
            onPressAndHold="0x75"
            mappingColor="0xED145B"
        />
        <button
            toolTip="Record"
            onPressAndHold="0x44"
            mappingColor="0xF5989D"
        />
        <button
            toolTip="Rocker Up"
            onPressAndHold="0x48"
            mappingColor="0x0072BC"
            KeyEvent="Up"
        />
        <button
            toolTip="Rocker Down"
            onPressAndHold="0x50"
            mappingColor="0x605CA8"
            KeyEvent="Down"
        />
        <button
            toolTip="Soft Key 1"
            onClick="DOWN:0x5b 0x70 UP:0x5b"
            mappingColor="0xF26C4F"
        />
        <button
            toolTip="Soft Key 2"
            onClick="DOWN:0x5b 0x71 UP:0x5b"
            mappingColor="0xF68E56"
        />
        <button
            toolTip="Soft Key 3"
            onClick="DOWN:0x5b 0x72 UP:0x5b"
            mappingColor="0xFBAF5D"
        />
        <button
            toolTip="Soft Key 4"
            onClick="DOWN:0x5b 0x73 UP:0x5b"
            mappingColor="0xF7941D"
        />
        <button
```

```
            toolTip="Up"
            onPressAndHold="0x48"
            mappingColor="0x39B54A"
            KeyEvent="Up"
        />
        <button
            toolTip="Down"
            onPressAndHold="0x50"
            mappingColor="0x009900"
            KeyEvent="Down"
        />
        <button
            toolTip="Left"
            onPressAndHold="0x4B"
            mappingColor="0x66CC66"
            KeyEvent="Left"
        />
        <button
            toolTip="Right"
            onPressAndHold="0x4D"
            mappingColor="0x00CC00"
            KeyEvent="Right"
        />
        <button
            toolTip="Enter"
            onClick="0x1C"
            mappingColor="0x006600"
            KeyEvent="Return"
        />
    </view>
</skin>
```

参照

その他の技術情報

スキンのカスタマイズ (デバイス)

スマート デバイスを開発コンピュータに接続する

このセクションでは、開発コンピュータと対象デバイス (実機またはエミュレータ) との間で、安全で信頼できる接続を確立する方法について説明します。

このセクションの内容

接続方法の選択

さまざまな接続方法について説明します。

方法 : 接続オプションの設定 (デバイス)

既定の接続設定を変更する方法について説明します。

方法 : ActiveSync を使用せずに Windows CE デバイスに接続する

接続をサポートする ActiveSync が利用できないときの接続手順について説明します。

方法 : Bluetooth を使用して接続する

Bluetooth 接続を確立するための手順について説明します。

方法 : IR を使用して接続する

IR 接続を確立するための手順について説明します。

方法 : Virtual PC セッションからデバイス エミュレータに接続する

開発コンピュータを、VPC セッションを実行しているエミュレータに接続する方法について説明しています。

方法 : Smartphone エミュレータ ファイル システムにアクセスする

ファイル エクスプローラのない Smartphone エミュレータのファイル システムにアクセスする手順について説明します。

方法 : エミュレータから開発用コンピュータのファイルにアクセスする

共有フォルダを使用してエミュレータと開発コンピュータの間でファイルを移動する方法について説明します。

接続のトラブルシューティング (デバイス)

適切な接続の妨げになる問題と、その解決方法について説明します。

参照

その他の技術情報

スマート デバイス開発

Mobile Developer Center

接続方法の選択

開発時に、スマート デバイスと開発コンピュータとの間に、高速で信頼できる接続を確立することは重要です。開発のほぼすべての段階でスマート デバイス エミュレータを使用できますが、実環境のハードウェアでアプリケーションをテストすることは、開発サイクルで重要な部分です。

後述するように、さまざまな接続オプションを使用できます。最も一般的な構成は次のとおりです。

- DMA トランスポートを使用してデバイス エミュレータに接続します。このトランスポートはネットワーク関連の接続問題を排除し、通常はこれが既定のトランスポートになっています。他のトランスポートを使用する特別な理由がない場合は、デバイス エミュレータでは必ず DMA トランスポートを使用してください。
- ActiveSync 4.x と USB ポートを使用して物理デバイスに接続します。

これらをはじめとする接続オプションは、Visual Studio の [ツール] メニューから選択できます。詳細については、「方法 : 接続オプションの設定 (デバイス)」を参照してください。

ActiveSync 4.x

ActiveSync 4.x では、ケーブル接続、クレードル接続、Bluetooth 接続、または赤外線接続を使用して、開発コンピュータとデバイスとを安全に接続できます。これは、必要な接続ファイルとセキュリティ ファイルを自動的にデバイスにダウンロードするための伝達手段にもなります。デバイスをクレードルに接続すると、ActiveSync が他のすべてのネットワーク カードを無効にするので、このデバイスが開発コンピュータとのみ通信していることを保証できます。ActiveSync は、デバイス アプリケーションを開発している間の標準接続メカニズムです。

目的のデバイスで ActiveSync がサポートされていない場合は、「方法 : ActiveSync を使用せずに Windows CE デバイスに接続する」を参照してください。

接続オプション

Pocket PC、Smartphone、および他の Windows CE ベースのハードウェアには、デバイスとコンピュータをリンクする方法が数多くあります。このセクションでは、各種の接続オプション、およびその利点と欠点について説明します。

関連するハードウェア デバイスによって、次の接続方法の 1 つ以上を採用できます。

USB 接続

最も簡単な形式の接続。Pocket PC デバイスと Smartphone デバイスは、すべて USB 接続をサポートしています。イーサネット接続またはワイヤレス 802.11b/g 接続ほど速くありませんが、USB 接続の簡易さは魅力的です。また、多くのデバイスは、USB ポートを使用して強化することもできます。これもまた別の利点です。

ワイヤード (有線) イーサネット ネットワーク

Pocket PC デバイスと Smartphone デバイスは、既定ではイーサネット接続をサポートしていません。追加のハードウェアが必要です。ただし、この接続規格は高速なため、デバッグなど、データが集中する操作を実行するときに推奨されます。

ワイヤレス 802.11b/g ネットワーク

ワイヤレス ネットワーク カードは、Pocket PC で使用できます。また、いくつかのモデルが、内蔵機能としてワイヤレス ネットワーク接続を備えるようになりました。ワイヤレス ネットワークは、ワイヤード イーサネット接続と同じくらいの速度です。

Bluetooth

Pocket PC デバイスと Smartphone デバイスの多くは、Bluetooth ワイヤレス ネットワーク機能を備えています。スマート デバイスと Bluetooth を組み合わせると、デスクトップ コンピュータの範囲にあれば、ActiveSync を介して接続できます。Bluetooth は 802.11b/g ワイヤレスほど速くないため、デバッグ目的では推奨されません。

シリアル接続

USB、ワイヤード ネットワーク、ワイヤレス ネットワークの各オプションを使用できない場合は、シリアル ポートを使用できます。低速でも、スマート デバイスを開発コンピュータに接続する方法として利用できます。

赤外線接続

赤外線接続には、追加の配線が必要ありません。また、Pocket PC デバイスおよび Smartphone デバイスには、標準として IrDA ポートが内蔵されています。ただし、赤外線接続の場合、確実に操作するには赤外線の通り道を確保する必要があります。また、そのように配慮した場合でも、デバッグ目的には適さない処理速度です。ただし、ファイルをデバイスにコピーする最後の手段として、IrDA は有効です。

参照

処理手順

接続のトラブルシューティング (デバイス)

その他の技術情報

スマート デバイスを開発コンピュータに接続する

方法：接続オプションの設定 (デバイス)

Visual Studio 2005 には、開発コンピュータをデバイスに接続するための多くのオプションが用意されています。これらの接続を管理するには、[デバイスのプロパティ] ダイアログ ボックスを使用します。

Visual Studio には、既定では DMA 転送を使用するエミュレータ接続と、既定では TCP/IP を使用する物理デバイスが付属しています。

接続オプションを設定するには

1. Visual Studio で、[ツール] メニューの [オプション] をクリックします。
2. [デバイス ツール] を展開して [デバイス] をクリックし、デバイスまたはエミュレータを選択して [プロパティ] をクリックします。

   関連するダイアログ ボックスを使用して、接続を選択して設定します。

参照

関連項目

[デバイスのプロパティ] ダイアログ ボックス

概念

接続方法の選択

その他の技術情報

スマート デバイスを開発コンピュータに接続する

方法：ActiveSync を使用せずに Windows CE デバイスに接続する

ActiveSync が使用できない場合、Visual Studio 2005 では、必要な接続ファイルが自動的にはデバイスにコピーされません。これらのファイルをデバイスにインストールし、Visual Studio の接続構成を変更して、デバイスのセキュリティを確立するには、次の手順を使用します。

最初の 2 つの手順は、デバイスおよび Visual Studio を準備するために 1 回だけ実行する必要があります。最後の手順は、セキュリティを設定して接続を確立するため、Visual Studio の新しいインスタンスから接続するたびに繰り返し実行する必要があります。

接続用にデバイスを準備するには

1. 所有するデバイスで何か接続を使用している場合は、そのデバイスの \Windows\ フォルダに次のファイルをコピーします。既定では、これらのファイルは、開発用コンピュータの \Program Files\Common Files\Microsoft Shared\CoreCon\1.0\Target\wce400\<CPU> にあります。
   - Clientshutdown.exe
   - ConmanClient2.exe
   - CMaccept.exe
   - eDbgTL.dll
   - TcpConnectionA.dll
2. デバイスのコマンド プロンプトから、conmanclient2.exe を実行します。
3. デバイスの IP アドレスを決定します。

接続用に Visual Studio を準備するには

1. Visual Studio で、[ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックします。次に、[デバイス] をクリックします。
2. [Windows CE 5.0 デバイス] を選択し、[プロパティ] をクリックします。
3. [トランスポート] ボックスの右側で、[構成] をクリックします。
4. [TCP/IP トランスポートの構成] ダイアログ ボックスで、[特定の IP アドレスを使用] を選択し、デバイスの IP アドレスを入力します。
5. ダイアログ ボックスを閉じます。

   デバイスをリセットするように求めるメッセージ ボックスが表示されます。この場合は、ソフト リセットでもかまいません。

セキュリティを設定して接続を確立するには

1. デバイスのコマンド プロンプトで、cMaccept.exe を実行します。
2. 3 分以内にデバイスに接続します。

   3 分以内に最初の接続を確立すると、同じ Visual Studio インスタンスを使用している限り、無期限で配置およびデバッグを続行できます。Visual Studio の別のインスタンスから接続する必要がある場合は、このセキュリティに関する手順をもう一度実行してください。

セキュリティに関するメモ：
この cMaccept を使用した手順は、デバイスのセキュリティを無効にすることで省略できます。そのためには、リモート レジストリ エディタを使用して `HLKM\System\CoreConOverrideSecurity = 1` DWORD 値を設定します。ただし、セキュリティを無効にすると、デバイスを悪意のある攻撃にさらすことになるため、適切な対策を用意している場合を除いて、この操作を行わないことをお勧めします。

参照

概念

デバイス プロジェクトのリモート ツール

その他の技術情報

スマート デバイスを開発コンピュータに接続する

方法：Bluetooth を使用して接続する

次の手順では、Bluetooth を使用して、開発用コンピュータにデバイスを接続する方法について説明します。このプロセスを完了するには、ActiveSync 4.0 以降をインストールしておく必要があります。

Bluetooth を使用して接続するには

1. 開発用コンピュータで、Bluetooth アンテナが正しく差し込まれていることを確認します。
2. 開発用コンピュータの Bluetooth プロパティ ダイアログ ボックスで、発見可能性をオンに変更します。
3. デバイスの Bluetooth アイコンをタップし、[Bluetooth を有効にする] をタップします。
4. [Bluetooth マネージャ] をタップします。
5. Bluetooth ウィンドウ メニューの [New] をクリックして、Bluetooth 対応デバイスの検索を開始します。
6. ActiveSync を使用して接続する開発用コンピュータを選択します。
7. プロンプトが表示されたら、一時パスキーを入力し、すぐに開発用コンピュータで同じパスキーを入力します。

   これで、デバイスとの Bluetooth 接続が確立されます。ActiveSync をセットアップするには、この Bluetooth 接続の仮想 COM ポートを作成します。この操作を行うには、次の手順に従います。

ActiveSync をセットアップするには

1. デスクトップ コンピュータで、Bluetooth の構成を調査し、Bluetooth 接続の受信用 COM ポートを追加します。
2. ActiveSync で、[接続の設定] ダイアログ ボックスを開き、[次のうちの 1 つへの接続を許可する] を選択します。
3. Bluetooth デバイスの COM ポートを選択し、[OK] をクリックします。
4. デバイスで、ActiveSync を起動します。
5. ActiveSync メニューの [Bluetooth に接続] をタップします。

   これで、ActiveSync 接続が確立されます。

参照

その他の技術情報

スマート デバイスを開発コンピュータに接続する

方法：IR を使用して接続する

次の手順では、赤外線ポート (IR: Infrared Port) を使用して、デバイスを開発用コンピュータに接続する方法について説明します。このプロセスを完了するには、ActiveSync 4.0 以降をインストールしておく必要があります。

開発用コンピュータをセットアップするには

1. 開発用コンピュータで、Microsoft ActiveSync を開きます。
2. ActiveSync の [ファイル] メニューの [接続の設定] をクリックします。
3. [次のうちの 1 つへの接続を許可する] を選択します。
4. [赤外線ポート (IR)] を選択し、[OK] をクリックします。

デバイスをセットアップするには

1. デバイスの IR ポートが開発用コンピュータの IR ポートを指していて、通信可能範囲内に配置されていることを確認します。
2. デバイスで、ActiveSync を開きます。
3. ActiveSync で、[ツール] メニューの [IR 接続を行う] をタップします。

   接続が確立されます。

参照

処理手順

接続のトラブルシューティング (デバイス)

その他の技術情報

スマート デバイスを開発コンピュータに接続する

方法：Virtual PC セッションからデバイス エミュレータに接続する

Virtual PC (VPC) セッション中は、TCP/IP を使用してデバイス エミュレータに接続できません。VPC は仮想ネットワーク スイッチ ドライバをサポートしていないからです。エミュレータが TCP/IP 接続を行うためには、このドライバが必要です。代わりに、次の手順に従います。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Virtual PC セッション中にエミュレータに接続するには

1. Virtual PC イメージに Microsoft ActiveSync 4.x をインストールします。
2. ActiveSync の [ファイル] メニューで、[接続の設定] をクリックします。
3. [接続の設定] ダイアログ ボックスで、接続トランスポートを [DMA] に変更します。
4. エミュレータを起動します。
5. Visual Studio の [ツール] メニューで、[デバイス エミュレータ マネージャ] をクリックします。
6. [利用可能なエミュレータ] ボックスのエミュレータを右クリックし、ショートカット メニューの [クレードルに接続] をクリックします。
7. ActiveSync ウィザードの指示に従って、連携を確立します。

参照

処理手順

方法 : Visual Studio でデバイス エミュレータを起動する

その他の技術情報

スマート デバイス プロジェクトの概要

スマート デバイスを開発コンピュータに接続する

方法：Smartphone エミュレータ ファイル システムにアクセスする

Smartphone エミュレータには、ファイル エクスプローラがありません。Smartphone ファイル システムにアクセスするには、次の方法を使用してください。このプロセスを完了するには、ActiveSync 4.0 以降をインストールしておく必要があります。

エミュレータで ActiveSync を使用する前に、デバイスがデスクトップ コンピュータに接続されていないこと、および USB 接続が ActiveSync で無効になっていることを確認します。

メモ：
次の手順を実行した後、デバイス エミュレータ マネージャを閉じたり、エミュレータを閉じたりすると、ActiveSync 接続も閉じます。

Smartphone エミュレータ ファイル システムにアクセスするには

1. Visual Studio の [ツール] メニューの [デバイス エミュレータ マネージャ] をクリックします。
2. [利用可能なエミュレータ] ボックスで、アクセスするファイル システムが存在するエミュレータを選択します。
3. デバイス エミュレータ マネージャの [操作] メニューで、[接続] をクリックします。

   選択したエミュレータの横に、接続が確立されたことを示すアイコンが表示されます。
4. 選択したエミュレータを右クリックし、[クレードルに接続] をクリックします。

   エミュレータがクレードルに接続されたことを示すアイコンに変わります。
5. ActiveSync を開きます。
6. ActiveSync の [ファイル] メニューの [接続の設定] をクリックします。
7. [次のうちの 1 つへの接続を許可する] チェック ボックスをオンにします。
8. ポートの一覧から [DMA] を選択し、[OK] をクリックします。

   ActiveSync が、エミュレータとの連携を開始します。新しいパートナーシップ ウィザードの指示に従います。

メモ：
連携が自動的に開始しない場合、[接続の設定] ダイアログ ボックスの [接続] をクリックし、接続ウィザードの指示に従ってください。

9. ActiveSync の連携手順が完了したら、ActiveSync ツール バーの [探索] をクリックし、Smartphone エミュレータ ファイル システムにアクセスします。

メモ：
ActiveSync を使用して Visual Studio に接続されたエミュレータを使用する場合は、対応するプラットフォームの対象である、エミュレータではなくデバイスを使用してください。

参照

その他の技術情報

スマート デバイスを開発コンピュータに接続する

方法：エミュレータから開発用コンピュータのファイルにアクセスする

次の手順では、フォルダ共有を使用し、デバイス エミュレータから開発用コンピュータのファイルにアクセスする方法について説明します。

メモ：
一般的に、Smartphone エミュレータにはファイル ビューアがないため、次に示す Smartphone のテスト手順ではリモート ファイル ビューアを使用します。詳細については、「リモート ファイル ビューア」を参照してください。

共有フォルダをセットアップするには

1. 開発用コンピュータで、その開発用コンピュータとデバイス エミュレータの間で共有するフォルダを作成します。
2. デバイス エミュレータで、[ファイル] メニューの [構成] をクリックします。
3. [全般] タブの [共有フォルダ] ボックスで、開発用コンピュータの共有フォルダのパスを入力するか、その共有フォルダに移動します。
4. [OK] をクリックします。

作成した共有フォルダを Pocket PC エミュレータでテストするには

1. Pocket PC エミュレータで、ファイル エクスプローラを開きます。
2. [マイ デバイス] をタップして選択します。

   [Storage Card] エントリは共有フォルダになっています。

作成した共有フォルダを Smartphone エミュレータでテストするには

1. Windows の [スタート] ボタンをクリックし、[すべてのプログラム] をポイントします。次に、[Microsoft Visual Studio 2005]、[Visual Studio Remote Tools] の順をポイントし、[リモート ファイル ビューア] をクリックします。
2. [Windows CE デバイスの選択] で、Smartphone エミュレータを選択し、[OK] をクリックして [Windows CE リモート ファイル ビューア] ウィンドウを開きます。

   [Storage Card] エントリは共有フォルダになっています。

参照

その他の技術情報

スマート デバイスを開発コンピュータに接続する

接続のトラブルシューティング (デバイス)

開発用コンピュータとデバイス間の接続に関する問題の多くは、セキュリティまたはネットワークに関する問題が原因で発生します。ここでは、接続に関する一般的な問題を特定して解決するのに役立つ情報を示し、セキュリティで保護された信頼性の高い接続を確立する手順について説明します。

デバイス エミュレータへの接続

デバイス エミュレータに接続するには、Visual Studio 2005 に用意されている DMA トランスポートを使用します。このトランスポートを使用すると、開発用コンピュータとエミュレータ間の接続に関するほぼすべての問題が解決されます。

重要
TCP/IP トランスポートは、特別な理由がない限り使用しないでください。エミュレータで TCP/IP を使用したことが原因で発生した問題を解決するには、次の手順に従います。詳細については、「Mobile Developer Center」を参照してください。

仮想スイッチ ドライバを開くことができない

エミュレートされた NE2000 カードまたは CS8900 カードを使用してデバイス エミュレータをネットワークに接続する場合は、仮想スイッチ ドライバが必要です。ドライバは、「Mobile Developer Center」からダウンロードできます。

ドライバを開く際にエラーが発生する原因としては、次のような理由があります。

- ドライバが不足している。
- 開発用コンピュータ上のネットワーク カードのドライバがインストールされていない。
- ドライバのインストール中に問題があった。
- ドライバが無効な状態である。
- 開発用コンピュータにネットワーク カードがない。

次の手順を使用して、正確な原因を診断します。

エラーの正確な原因を診断するには

1. [エミュレータのプロパティ] ダイアログ ボックスの [ネットワーク] タブを調べます。

   NE2000 カードおよび CS8900 カードのどちらか、または両方が有効な場合、それらが関連付けられているネットワーク カードが存在し、接続されていることを確認します。[エミュレータのプロパティ] ダイアログ ボックスを開くには、エミュレータで、[ファイル] メニューの [構成] をクリックします。

2. アダプタのネットワーク プロパティを調べ、[仮想マシン ネットワーク サービス] という項目が存在して有効になっており、バージョンが正しい (2.6.465.224 以降) ことを確認します。

3. これらの手順により問題が解決されない場合、ドライバを再インストールします。

エミュレータへの配置を行う際のエラー

開発用コンピュータでワイヤレス ネットワーク接続が設定されている場合に TCP トランスポートを使用すると、Microsoft Loopback Adapter のインストールなど、追加の手順が必要になる可能性があります。詳細については、「Mobile Developer Center」を参照してください。

メモ :
TCP トランスポートを使用する特別な理由がない限り、ネットワークに関する問題を回避するには、DMA トランスポートを使用してください。

トランスポートを切り替えた後にデバッグできない

エミュレータのトランスポートは変更できますが、デバイスをソフト リセットするまでは、エミュレータは新しいトランスポートにバインドされません。

メモ :
DMA トランスポートは、デバイス エミュレータに適したトランスポートです。TCP/IP トランスポートは、特別な理由がない限り使用しないでください。

トランスポートを切り替えるには

1. Visual Studio で、[ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックします。次に、[デバイス] をクリックします。
2. エミュレータを選択し、[プロパティ] をクリックします。
3. [トランスポート] ボックスで、別のトランスポートを選択します。

   TCP/IP に切り替える場合は、[構成] をクリックし、追加オプションを設定します。
4. [OK] をクリックしてダイアログ ボックスを閉じます。

Virtual PC セッションでの実行中にエミュレータに接続できない

この接続に関する問題は、エミュレータに DMA トランスポートを使用することで回避できます。詳細については、「方法 : Virtual PC セッションからデバイス エミュレータに接続する」を参照してください。

デバイス エミュレータのインストールの修復

一般に、デバイス エミュレータへの接続が失敗したことを示すエラーは、インストールのエラーではありません。ただし、次の手順を実行して、デバイス エミュレータのインストールを修復できます。この手順を実行するには、元のインストール メディアが必要です。Visual Studio 2005 のインストールを修復しても、デバイス エミュレータのインストールは修復されません。

デバイス エミュレータのインストールを修復するには

1. Visual Studio 2005 の元のインストール メディアで、wcu\ARM に移動します。

   このフォルダの場所は、Disk 1、Disk 2 など、Visual Studio のエディションによって異なります。
2. vs_emulator.exe をダブルクリックしてデバイス エミュレータ セットアップ ウィザードを開き、ウィザードの指示に従います。

その他のヒント

デバイス エミュレータ独自のヘルプ システムには、その他のヒントが用意されています。詳細については、デバイス エミュレータの [ヘルプ] をクリックし、[目次] タブまたは [キーワード] タブで「接続のトラブルシューティング」を検索してください。

物理デバイスへの接続

デバイスに適切な証明書がない

Smartphone 2003 以降を含む、一部のデバイスでは、セキュリティ上の理由から適切な証明書をインストールする必要があります。日常の開発作業に必要な証明書は、その証明書をインストールするためのツールと共に、Visual Studio 2005 に用意されています。

必要な証明書をインストールするには

1. 使用できる任意の接続機構を使用してデバイスに接続します。
2. SDKCERTS.cab を開発用コンピュータからデバイスにコピーします。

   既定では、SDKCERTS.cab は \Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools にあります。
3. デバイスで、sdkcerts.cab を展開して証明書をインストールします。

Windows CE 5.0 デバイスが準備されていない

ActiveSync をサポートしていない Windows CE 5.0 デバイスでは、Visual Studio インスタンスで接続を確立する前に、準備手順を実行する必要があります。詳細については、「方法 : ActiveSync を使用せずに Windows CE デバイスに接続する」を参照してください。

配置時に予期しない動作が起きる

開発用コンピュータが ActiveSync を通じてあるデバイスに接続されているときに、たとえば Windows CE デバイスとの TCP/IP 接続を確立しようとして接続エラーが発生した場合は、開発用コンピュータは ActiveSync 接続デバイスに接続されており、TCP/IP 接続の失敗について警告を発しません。

ワイヤレス接続

Visual Studio 2005 では、デバイスへの接続にワイヤレス テクノロジを使用できますが、ワイヤレス テクノロジを使用すると、保守性のある正常な接続に悪影響を与える可能性のある別の要因が発生します。このような要因には、IR ポートの不整合、RF 接続におけるシグナルの障害や劣化などがあります。

参照

その他の技術情報

スマート デバイスを開発コンピュータに接続する

.NET Compact Framework を使用したデバイスのプログラミング

このセクションでは、Visual Basic 言語または Visual C# 言語と .NET Compact Framework を使用したスマート デバイス アプリケーションの開発について説明します。

このセクションの内容

マネージ デバイス プロジェクトの新機能

Visual Basic .NET 2003 からの重要な変更点を示します。

.NET Compact Framework 開発デスクトップとの違い

デスクトップ アプリケーションの開発との違いについて説明します。

.NET Compact Framework の各種のバージョン

.NET Compact Framework の複数のリリース (Service Pack を含む) について説明します。

Pocket PC 2003 コントロールと Smartphone 2003 コントロールの違い

Pocket PC 2003 の開発でサポートされているコントロールおよび Smartphone 2003 の開発でサポートされているコントロールの表を示します。

マネージ デバイス プロジェクトのコントロールおよびコンポーネント

デバイス開発に使用できる追加のコントロールおよびコンポーネントについて説明します。

デバイス プロジェクトのカスタム コントロールおよびユーザー コントロール

デバイス プロジェクトに使用できる機能について説明します。

デバイスの COM 相互運用性

マネージ デバイス プロジェクトで COM 相互運用機能アセンブリを作成する方法について説明します。

マネージ デバイス プロジェクトの作成および開発

一般的なデバイス開発タスクの詳細な手順を説明します。

マネージ デバイス プロジェクトのデータ

Visual Studio 2005 でデバイス用のデータを管理する方法について説明します。

チュートリアル : 簡単なアプリケーションの作成

簡単な Visual Basic デバイス プロジェクトまたは Visual C# デバイス プロジェクトを作成する方法について、詳細な手順を説明します。

参照

処理手順

チュートリアル : デバイス対応の Windows フォーム アプリケーションの作成

その他の技術情報

.NET Compact Framework

スマート デバイス開発

.NET Compact Framework

よく寄せられる質問

Mobile Developer Center

マネージ デバイス プロジェクトの新機能

次の一覧では、Visual Studio .NET 2003 で追加された、マネージ デバイス アプリケーションの開発のための拡張機能について説明します。

メモ :
Visual Studio 2005 を使用すると、.NET Compact Framework Version 1.0 および Version 2.0 の両方に対してプログラムを作成できますが、次の一覧でアスタリスク (*) が付いている機能は、Version 1.0 ではサポートされていません。

新機能

Visual Studio 2005 では、次の機能を使用できます。

機能	詳細
*Windows フォームでのコントロールの固定。	方法 : Windows フォームにコントロールを固定する
Visual Studio のセットアップ プロジェクトと配置プロジェクトを使用した CAB ファイルの生成。この処理によって、.inf ファイルを手動で削除する必要がなくなります。	デバイス ソリューションをパッケージ化する方法の概要
サポートされていないプロパティ/メソッド/イベントのフィルタ処理。	この機能には、フィルタ処理される IntelliSense、ビルド検証手順、およびサポートされていないコントロールの変換機能が含まれます。
カラー エディタ。	デバイスに表示される色を指定します。 [色の作成] ダイアログ ボックス(デバイス)
カスタム コントロール *ユーザー コントロール。	完全にサポートされています。クラスと型のデザインおよび表示を使用してカスタムのデザイナ属性を追加します。
*Windows フォームのコントロールのドッキング。	方法 : Windows フォーム上のコントロールをドッキングする
*データ デザインのサポート。	マネージ デバイス プロジェクトのデータ
フォント エディタ。	現在のプラットフォームでサポートされるフォントを対象として機能します。[フォント] ダイアログ ボックス (デバイス)
フォーム ファクタのサポート (向きと解像度を含む)。 *Autoscale。	[フォーム ファクタのプロパティ] ダイアログ ボックス (デバイス)
コード作成時に使用するフォームのスキン。	[フォーム ファクタのプロパティ] ダイアログ ボックス (デバイス)
*ツールボックスの新しいコントロールとコンポーネント。	WebBrowser, Notification, DocumentList, DateTimePicker, MonthCalendar, LinkLabel, Splitter, BindingSource
プラットフォーム固有の WYSIWYG。	デザイナが改良され、各プラットフォームでの外観が正確になります。
対象として .NET Compact Framework Version 1.0 をサポート。	Smartphone 2003 と Pocket PC 2003 がサポートされます。
同じソース コードを使用したプラットフォームの変更。	方法 : プラットフォーム間でソース コードを共有する (デバイス)

参照

その他の技術情報

.NET Compact Framework 2.0 **の新機能**

開発環境の新機能

.NET Compact Framework **を使用したデバイスのプログラミング**

.NET Compact Framework 開発デスクトップとの違い

デバイス プロジェクトを起動する前に、.NET Framework を使用したデスクトップ開発と .NET Compact Framework を使用したデバイス開発の主な違いについて理解することが重要です。

Visual Basic のプログラミング要素

Visual Basic を使用して .NET Compact Framework に対してプログラムを作成する場合は、完全な .NET Framework に対してプログラムを作成する場合に使用できる関数やキーワードなどのプログラミング要素をすべて使用できるわけではありません。その違いは、「デバイス対応の Visual Basic の言語リファレンス」に要約されています。また、「Visual Basic リファレンス」にはそれらの要素に関する個別のトピックがあります。

My による開発

Visual Studio 2005 では、My.Resources、My.Forms、および My.WebServices がサポートされます。My.Application、My.Computer、My.User、または My.Settings はサポートされません。詳細については、「My の参照」を参照してください。

ファイルの入出力

Visual Basic では、ファイルの入出力 (I/O) に関して次の 2 つのオプションがあります。

- 標準の .NET Framework System.IO 名前空間。共通言語ランタイム (CLR: Common Language Runtime) のすべての言語はこれらのライブラリをサポートします。
- 以前のバージョンの Visual Basic に類似した開発環境を提供する、Visual Basic 固有のライブラリー式。

デバイス プロジェクトでサポートされるのは、.NET Framework **System.IO** 名前空間だけです。FileSystem 名前空間によるファイル入出力は、次の理由のためにサポートされていません。

- **FileSystem** 名前空間で共通に使用されるいくつかの機能が、デバイスに存在しません。たとえば、デバイスには、現在のディレクトリや現在のドライブという概念がありません。このため、ChDir 関数や ChDrive 関数は使用できません。
- .NET Framework **System.IO** 名前空間だけをサポートすることにより、Visual Basic ヘルパー ライブラリのサイズを抑えることができます。このため、デバイス上の利用できる領域が多くなります。

暗黙の遅延バインディング

Visual Basic では、オブジェクト型 (Object) 型として宣言された変数にオブジェクトを代入する場合、そのオブジェクトが "遅延バインディング" されます。この型のオブジェクトは実行時にバインドされます。これらのオブジェクトに対しては、値を代入したり取得したりできます。ドット表記を使用してオブジェクト変数のメソッドまたはプロパティを指定することはできません。次のコードでは、オブジェクトのプロパティを取得しようとしているため、コンパイラ エラーが発生します。

```
dim a as object = "automobile"

dim i as integer = a.horsepower
```

COM 相互運用

デスクトップ アプリケーションの開発者は、COM の相互運用性を使用することにより、既存の COM オブジェクトを利用しながら、自分のペースで .NET Framework に移行していくことができます。デバイス プロジェクトでサポートされるのは、COM 相互運用の特定のシナリオのみです。詳細については、「デバイスの COM 相互運用性」を参照してください。

デバッグ

実行中のプロセスへのアタッチは、デスクトップの場合とは多少異なります。詳細については、「方法 : マネージ デバイスのプロセスに接続する」を参照してください。

参照

関連項目

デバイス対応の Visual Basic の言語リファレンス
System.IO Namespace
使用する技術とツールの決定
My の参照

概念

事前バインディングと遅延バインディング
Visual Basic における Me、My、MyBase、MyClass
デバイスの COM 相互運用性

その他の技術情報

Visual Basic におけるファイル アクセス
Visual Studio での .NET Framework プログラミング

.NET Compact Framework の各種のバージョン

スマート デバイス向けのアプリケーションを開発する場合、インストールされている Compact Framework のバージョン間の差異を考慮することが重要です。

.NET Compact Framework Version 1.0

- Compact Framework の最初のリリース。

.NET Compact Framework Version 1.0 SP1

- バージョン 1.0 からのバグ修正。
- Smartphone のサポート。

.NET Compact Framework Version 1.0 SP2

- バージョン 1.0 SP1 からのバグ修正。
- パフォーマンスの向上。
- 横向き表示モードを持つデバイスのサポート。
- 横向きモードのダイアログを自動的にサポートする自動スクロールのサポート。

.NET Compact Framework Version 1.0 SP3

- バージョン 1.0 SP2 からのバグ修正。

.NET Compact Framework Version 2.0

.NET Compact Framework Version 2.0 にアップグレードされたスマート デバイスは、次の機能を持ちます。

- パフォーマンスの向上。
- ジェネリック クラスのサポート。
- COM の相互運用機能のサポート。
- コントロールのサポートの改良。
- モバイル デバイス用 Direct3D と DirectDraw のサポート。

参照

その他の技術情報

.NET Compact Framework を使用したデバイスのプログラミング
.NET Compact Framework 2.0 の新機能
.NET Compact Framework FAQ - .NET Compact Framework 2.0

Pocket PC 2003 コントロールと Smartphone 2003 コントロールの違い

次の表は、Windows Mobile Pocket PC 2003 **と** Windows Mobile Smartphone 2003 **でサポートされているコントロールの相違点をまとめたものです。一部のコントロールは、たとえばタッチ スクリーンがないなどの理由で、**Smartphone **のコンテキストでは使用できません。**

コントロール	**Pocket PC 2003**	**Smartphone 2003**
Label	○	○
LinkLabel	○	×
Button	○	×
TextBox	○	○
MainMenu	○	○
CheckBox	○	○
RadioButton	○	×
PictureBox	○	○
Panel	○	○
DataGrid	○	×
BindingSource	○	×
ListBox	○	×
ComboBox	○	○
ListView	○	○
TreeView	○	○
TabControl	○	×
DateTimePicker	○	×
MonthCalendar	○	×
HScrollBar	○	○
VScrollBar	○	○
Timer	○	○
Splitter	○	×
DomainUpDown	○	×

NumericUpDown	○	×
TrackBar	○	×
ProgressBar	○	○
ImageList	○	○
ContextMenu	○	×
ToolBar	○	×
StatusBar	○	×
OpenFileDialog	○	×
SaveFileDialog	○	×
WebBrowser	○	×
InputPanel	○	×
Notification	○	×
DocumentList	○	×

ソース コードはプラットフォーム間で共有できます。詳細については、「方法 : プラットフォーム間でソース コードを共有する (デバイス)」を参照してください。

参照

処理手順

方法 : プロジェクトを新しいバージョンの .NET Compact Framework にアップグレードする

その他の技術情報

.NET Compact Framework を使用したデバイスのプログラミング

マネージ デバイス プロジェクトのコントロールおよびコンポーネント

マネージ デバイス プロジェクトには、独自のコントロールとコンポーネントのセットがあります。また、Visual Studio のツールボックスにはマネージ デバイス プロジェクト独自のタブがあります。デバイス プロジェクトがアクティブな場合、デスクトップ用のプロジェクトと同様に、この [デバイス] タブをドラッグ アンド ドロップ操作で使用できます。

詳細については、「.NET Compact Framework の Windows フォーム コントロール」を参照してください。

参照

その他の技術情報

マネージ デバイス プロジェクトの作成および開発

デバイス プロジェクトのカスタム コントロールおよびユーザー コントロール

.NET Compact Framework では、Windows フォーム プロジェクト向けのコントロールの作成およびカスタマイズが適切にサポートされます。

ユーザー コントロール

デスクトップ プロジェクトと同じ方法で、ユーザー コントロールを作成してマネージ デバイス プロジェクトに追加できます。詳細については、「方法 : 複合コントロールを作成する」を参照してください。

カスタム コントロール

.NET Compact Framework には、コントロールをカスタマイズする方法が数多く用意されています。詳細については、「カスタム コントロールの開発」を参照してください。

参照

処理手順

チュートリアル : デバイスのユーザー コントロールの作成
チュートリアル : 簡単な属性をユーザー コントロールに追加する

その他の技術情報

デザイン時の Windows フォーム コントロールの開発
クラスと型のデザインおよび表示
.NET Compact Framework を使用したデバイスのプログラミング

デバイスの COM 相互運用性

.NET Compact Framework では、COM オブジェクトのランタイム呼び出し可能ラッパー ("相互運用機能アセンブリ" とも呼ばれる) がサポートされています。この機能には、複合型のマーシャリングが含まれています。デバイス用の COM 相互運用性は、デスクトップの実装に基づいています。したがって、コンポーネントはデスクトップで登録する必要があります。

サポートされているシナリオ

次のシナリオは、Visual Studio 2005 のデバイス プロジェクトでサポートされています。

- 既存の COM コンポーネントをマネージ プロジェクトに参照として追加できます。この操作を行うと、相互運用機能アセンブリが作成され、そのアセンブリが参照として自動的に追加されます。追加した相互運用機能アセンブリはマネージ アセンブリと同様に使用でき、オブジェクトのプロパティ、メソッド、およびイベントを IntelliSense およびオブジェクト ブラウザで使用できます。追加できるファイルの種類は DLL、EXE、および TLB です。
- ネイティブ プロジェクトを作成して COM コンポーネントを生成した後、その COM コンポーネントを使用するマネージ プロジェクトを同じソリューション内に作成できます。これを行うプロセスは次のとおりで、デスクトップの場合と同じです。
    - ネイティブ プロジェクトを設定して TLB 出力を生成します。
    - ネイティブ プロジェクトをコンパイルして DLL を生成します。
    - マネージ プロジェクトで、DLL への参照を追加します。この操作により、相互運用機能アセンブリが生成されます。

サポートされていないシナリオ

次のシナリオは、Visual Studio 2005 ではサポートされていません。

- マネージ プロジェクト内部から既存の ActiveX COM コンポーネントへの参照
- システム以外の子コンポーネントを持つ COM オブジェクト
- データ ソース ウィザード内部からビジネス オブジェクトとして参照される COM オブジェクト

参照

処理手順

チュートリアル : Hello World: スマート デバイスの COM 相互運用の例
チュートリアル : マネージ コードとネイティブ コードの両方を含むソリューションのデバッグ

概念

COM 相互運用の概要
ランタイム呼び出し可能ラッパー
選択したインターフェイスのマーシャリング

その他の技術情報

.NET Framework アプリケーションにおける COM 相互運用性
.NET Compact Framework を使用したデバイスのプログラミング
.NET Compact Framework の相互運用性

マネージ デバイス プロジェクトの作成および開発

マネージ デバイス プロジェクトの開発は、マネージ デスクトップ プロジェクトの開発によく似ています。主な違いとして、.NET Compact Framework は .NET Framework のサブセットなので、一部の領域がサポートされていません。詳細については、「.NET Compact Framework」を参照してください。

このセクションの内容

方法 : Visual Basic または Visual C# を使用してデバイス アプリケーションを作成する

方法 : Visual Basic プロジェクトで [デバイス] ツール バーを表示する

方法 : Visual Basic プロジェクトで [構成マネージャ] を表示する (デバイス)

方法 : プラットフォーム間でソース コードを共有する (デバイス)

方法 : デバイス プロジェクトのプラットフォームを変更する

方法 : プロジェクトを新しいバージョンの .NET Compact Framework にアップグレードする

方法 : デバイス プロジェクトのコードがプラットフォームでサポートされているか検査する

方法 : HardwareButton イベントを処理する (デバイス)

方法 : フォームの向きおよび解像度を変更する (デバイス)

方法 : 既定のデバイスを変更する (マネージ プロジェクト)

方法 : プラットフォーム呼び出しを使用して Wave ファイルを再生する (デバイス)

デバイス プロジェクト内のコード スニペットの管理

参照

その他の技術情報

.NET Compact Framework を使用したデバイスのプログラミング
Visual Studio での .NET Framework プログラミング

方法：Visual Basic または Visual C# を使用してデバイス アプリケーションを作成する

デバイス用の Visual Basic マネージ プロジェクトおよび Visual C# マネージ プロジェクトを作成する一般的なプロセスは、デスクトップ用のプロジェクトを作成する場合と同じです。ただし、プロジェクトを実行する対象プラットフォーム (Pocket PC 2003 など) と .NET Compact Framework のバージョン (バージョン 2.0 など) を選択する必要があります。

Visual Studio 2005 の強化された [新しいプロジェクト] ダイアログ ボックスは、Visual Studio .NET 2003 のスマート デバイス アプリケーション ウィザードに代わるものです。Visual Studio 2005 では、プロジェクトの種類やプロジェクト テンプレートに関するすべての項目を [新しいプロジェクト] ダイアログ ボックスで選択できます。

メモ：
ダイアログ ボックスとメニュー コマンドで使用できるオプションは、アクティブな設定つまりプロファイルによって異なります。次に示す手順では、Visual Basic 開発設定と Visual C# 開発設定を使用しています。設定を表示または変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。

デバイス プロジェクトを作成するには

1. (Visual Basic) Visual Studio 2005 の [ファイル] メニューで、[新しいプロジェクト] をクリックします。

   または

   (Visual C#) Visual Studio 2005 で、[ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。

2. [新しいプロジェクト] ダイアログ ボックスの [プロジェクトの種類] で、[Visual Basic プロジェクト] または [Visual C#] を展開し、[スマート デバイス] を展開します。次に、目的のプラットフォーム ([Pocket PC 2003] など) をクリックします。

   目的の言語が最初に表示されない場合は、[他の言語] を展開します。この表示は、ユーザーの開発設定で管理されます。設定を表示または変更するには、[ツール] メニューの [設定のインポートとエクスポート] を選択します。

3. [テンプレート] で、[Pocket PC 2003 クラス ライブラリ] など、目的に合ったテンプレートをクリックします。

メモ：
名前に (1.0) が付加されているテンプレートは、バージョン 1.0 の .NET Compact Framework 用にデザインされたものです。その他のテンプレートはバージョン 2.0 用にデザインされています。

4. [プロジェクト名] ボックスにプロジェクト名を入力します。

   [場所] ボックスが表示される場合は、プロジェクト ファイルを格納する場所を確認し、[OK] をクリックします。

既存のソリューションにプロジェクトを追加するには

- [ファイル] メニューの [追加] をポイントし、[新しいプロジェクト] または [既存のプロジェクト] をクリックします。

  [新しいプロジェクト] コマンドでは [新しいプロジェクト] ダイアログ ボックスが開かれるので、新しいプロジェクトを作成できます。[既存のプロジェクト] コマンドでは [既存プロジェクトの追加] ダイアログ ボックスが開かれるので、現在のソリューションに含める既存のプロジェクトを選択できます。

既存のプロジェクトを移植するには

- 以前のバージョンの Visual Studio で作成したプロジェクトを移植するための手順は、Visual Studio のプロジェクト管理向けの一般的なドキュメントで説明されています。詳細については、「プロジェクトと下位互換性」および「複数のバージョンの .NET Framework の使用」を参照してください。

参照

処理手順

チュートリアル : デバイス対応の Windows フォーム アプリケーションの作成

関連項目

使用する技術とツールの決定

その他の技術情報

.NET Compact Framework 2.0 の新機能
ソリューション、プロジェクト、およびファイルの管理
マネージ デバイス プロジェクトの作成および開発

方法：Visual Basic プロジェクトで [デバイス] ツール バーを表示する

開発設定を [Visual Basic **開発設定**] **に設定した場合、既定では、**[デバイス] **ツール バーが表示されません。**

[デバイス] ツール バーを表示するには

- Visual Studio **で、**[**表示**] **メニューの** [**ツール バー**] **をポイントし、**[**デバイス**] **をクリックします。**

参照

その他の技術情報

マネージ デバイス プロジェクトの作成および開発

方法：Visual Basic プロジェクトで [構成マネージャ] を表示する (デバイス)

[Visual Basic 開発設定] を選択した場合、既定では、[構成マネージャ] が表示されません。[構成マネージャ] は、デバッグ構成からリリース構成に切り替える場合などに使用します。[構成マネージャ] を表示する手順を次に示します。

Visual Basic デバイス プロジェクトで [構成マネージャ] を表示するには

1. Visual Studio で、[ツール] メニューの [オプション] をクリックし、[プロジェクトおよびソリューション] を展開します。次に、[全般] をクリックします。
2. [ビルド構成の詳細を表示] をクリックします。

参照

その他の技術情報

マネージ デバイス プロジェクトの作成および開発

方法：プラットフォーム間でソース コードを共有する (デバイス)

対象のプラットフォームに依存しているコードのセクションをコンパイラ定数で区別することにより、プラットフォーム間でソース コードを共有できます。有効な定数は、*PocketPC*、*Smartphone*、および *WindowsCE* です。これらのプラットフォームは、同一バージョンの .NET Compact Framework を対象にする必要があります。

以下の手順は、この方法を使用した単純な例です。まず、Visual Basic Pocket PC アプリケーションを作成し、コンパイラ ディレクティブを追加して、アプリケーションを実行します。次に、アプリケーションを終了し、Smartphone アプリケーションに変更します。その後 Smartphone アプリケーションを実行して、タイトル バーのテキストが変更されたことを確認します。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Pocket PC バージョンを作成して実行するには

1. Visual Studio で、[ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [プロジェクトの種類] ペインの [Visual Basic] を展開し、[スマート デバイス] を展開して、[Pocket PC 2003] をクリックします。
3. [テンプレート] ペインの [デバイス アプリケーション (1.0)] をクリックし、[OK] をクリックします。

   付加されている (1.0) は、これが .NET Compact Framework 1.0 プロジェクトであることを示しています。
4. デザイナで、フォームを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
5. フォームの [Text] プロパティの値を消去 (空白に) します。
6. ソリューション エクスプローラで Form1.vb を右クリックし、ショートカット メニューの [コードの表示] をクリックします。
7. [Windows フォーム デザイナで生成されたコード] 領域を展開します。
8. Public Sub New() の InitializeComponent() の後に、次のコードを挿入します。

```
#If PocketPC Then
   Me.Text = "PPC2003"
#Else
   Me.Text = "Smartphone"
#Endif
```

9. [デバッグ] メニューの [デバッグ開始] をクリックします。
10. [<プロジェクト名> の配置] ダイアログ ボックスで、[Pocket PC 2003 SE エミュレータ] をクリックし、[配置] をクリックします。

    この Pocket PC アプリケーションをエミュレータ上で実行すると、フォームのタイトル バーに "PPC2003" と表示されます。

Smartphone バージョンを作成して実行するには

1. 状態を保存せずにエミュレータを閉じます。

   接続が切断されたことを示すメッセージが表示された場合は、[OK] をクリックします。
2. [プロジェクト] メニューの [ターゲット プラットフォームの変更] をクリックします。
3. [ターゲット プラットフォームの変更] ダイアログ ボックスの [変更] ボックスで、[Smartphone 2003] を選択し、[OK] をクリックします。
4. プロジェクトを終了して再起動することを確認するメッセージ ボックスで、[はい] をクリックします。

   ツール バーの [ターゲット デバイス] ボックスに [Smartphone 2003 SE エミュレータ] が表示されます。
5. [デバッグ] メニューの [デバッグ開始] をクリックします。
6. [<プロジェクト名> の配置] ダイアログ ボックスで、[Smartphone 2003 SE エミュレータ] をクリックし、[配置] をクリックします。

この Smartphone アプリケーションをエミュレータ上で実行すると、フォームのタイトル バーに Smartphone と表示されます。

参照

処理手順

方法 : デバイス プロジェクトのプラットフォームを変更する

その他の技術情報

マネージ デバイス プロジェクトの作成および開発

方法：デバイス プロジェクトのプラットフォームを変更する

同じプロジェクト内でプラットフォームを切り替えることができます。たとえば、Pocket PC プラットフォームを対象としている場合、その対象を Windows CE プラットフォームに切り替えることができます。ただし、切り替え先のプラットフォームと元のプラットフォームが同じバージョンの .NET Compact Framework を対象にしている必要があります。

デバイス プロジェクトで対象のプラットフォームを変更するには

1. [プロジェクト] メニューの [ターゲット プラットフォームの変更] をクリックします。
2. [変更] ボックスで、別のプラットフォームを選択し、[OK] をクリックします。

   選択できるのは、アクティブなプラットフォームと同じバージョンの .NET Compact Framework を対象とするプラットフォームのみです。

参照

関連項目

[ターゲット プラットフォームの変更] ダイアログ ボックス (デバイス)

その他の技術情報

マネージ デバイス プロジェクトの作成および開発

方法：プロジェクトを新しいバージョンの .NET Compact Framework にアップグレードする

この手順では、開発用コンピュータに新しいバージョンの .NET Compact Framework がインストールされている場合に、既存のプロジェクトの .NET Compact Framework のバージョンをアップグレードします。

プロジェクトを新しいバージョンの .NET Compact Framework にアップグレードするには

1. [プロジェクト] メニューの [プロジェクトのアップグレード] をクリックします。

メモ：
メニューに [プロジェクトのアップグレード] コマンドが表示されない場合は、新しいバージョンのプラットフォームが開発用コンピュータにインストールされていないので、現在のプロジェクトではアップグレード プロセスを使用できません。

2. 表示されるダイアログの指示に従います。

参照

その他の技術情報

マネージ デバイス プロジェクトの作成および開発

方法：デバイス プロジェクトのコードがプラットフォームでサポートされているか検査する

Visual Studio では、コードが対象のプラットフォームでサポートされているかどうかが常に検証されます。サポートされていない場合は警告が生成されます。

すべての警告がエラーとして処理されるように指定して、ビルドが失敗するようにすると、サポートされていないコードが配置されるのを防ぐことができます。

たとえば、次の Visual Basic コードは、コンパイルしても Smartphone アプリケーションで警告またはエラーが生成されます。Smartphone でボタンがサポートされていないためです。

```
Dim btn as System.Windows.Forms.Button

btn = new System.Windows.Forms.Button()

…

btn.Caption = "MyButton"
```

Visual Basic ですべての警告をエラーとして処理するには

1. ソリューション エクスプローラで、[<プロジェクト名>] を右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [コンパイル] タブの [すべての警告をエラーとして扱う] を選択します。

Visual C# ですべての警告をエラーとして処理するには

1. ソリューション エクスプローラで、[<プロジェクト名>] を右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [ビルド] タブの [警告をエラーとして扱う] セクションで、[すべて] を選択します。

参照

その他の技術情報

マネージ デバイス プロジェクトの作成および開発

方法：HardwareButton イベントを処理する (デバイス)

HardwareButton コントロールを使用して、Pocket PC 上のアプリケーション キーをオーバーライドします。

特定のアプリケーション キーに HardwareButton コンポーネントを割り当てるには

1. ツールボックスの [デバイス コンポーネント] タブから、HardwareButton コンポーネントを Windows フォームまたはデザイナのコンポーネント トレイにドラッグします。
2. コンポーネント トレイで、[HardwareButton] コントロールを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
3. [AssociatedControl] プロパティをフォーム (たとえば、[Form1]) に設定します。
4. [HardwareKey] プロパティを、オーバーライドするキー (たとえば、[ApplicationKey1]) に設定します。
5. デザイナのスキン上のボタン (たとえば、[ソフト キー 1]) をクリックします。

   コード エディタが開き、`Form_KeyDown` イベント ハンドラが表示されます。
6. 次のコードを挿入します。

   ```
   if ((int) e.KeyCode == (int) Microsoft.WindowsCE.Forms.HardwareKeys.ApplicationKey1)
   {
   //TODO
   }
   ```

   通常は、`//TODO` セクションを使用してアプリケーションを起動します。

参照

処理手順

方法 : HardwareButton コンポーネントを使用する

関連項目

HardwareButton

その他の技術情報

マネージ デバイス プロジェクトの作成および開発

方法：フォームの向きおよび解像度を変更する (デバイス)

既定の向き (回転) のプロパティは、Visual Studio と共にインストールされたプラットフォーム用に設定済みです。プロパティを変更する必要がある場合、またはこれらのプロパティが正しくないか不足している SDK をインストールした場合は、次の手順を使用します。

方向を変更するには

1. [ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックして、[フォーム ファクタ] をクリックします。
2. 使用しているプロジェクトのフォーム用にフォーム ファクタ (たとえば、[Pocket PC 2003 縦長]) を選択し、[プロパティ] をクリックします。
3. [Windows フォーム デザイナでスキンを表示する] および [回転サポートを有効にする] を選択します。
4. [OK] をクリックして [フォーム ファクタのプロパティ] ダイアログ ボックスを閉じます。もう一度 [OK] をクリックして [オプション] ダイアログ ボックスを閉じます。

メモ：
デザイナを開いたままオプションを変更した場合は、変更を有効にするためにデザイナを閉じて再度開きます。

5. デザイナでフォームまたはスキンを右クリックし、[左に回転] または [右に回転] を選択してスキンを回転させます。

   デバイスを回転させるときにフォームが回転するかどうかは、以下のように、現在のプラットフォーム、およびフォーム プロパティの設定によって異なります。

   - Smartphone の場合 : デバイスが回転すると、フォームも必ず回転します。
   - Pocket PC の場合 : 既定では、デバイスが回転すると、フォームも回転します。デバイスと一緒にフォームが回転するのを防ぐには、フォームの WindowState プロパティを *Normal* に設定し、FormBorderStyle プロパティを *None* に設定します。
   - Windows CE の場合 : 既定では、デバイスが回転しても、フォームは回転しません。デバイスと一緒にフォームを回転させるには、WindowState プロパティを *Maximized* に設定します。

解像度を変更するには

1. [ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックして、[フォーム ファクタ] をクリックします。
2. 使用しているプロジェクトのフォーム用にフォーム ファクタ (たとえば、[Pocket PC 2003 縦長]) を選択し、[プロパティ] をクリックします。
3. [フォーム ファクタのプロパティ] ダイアログ ボックスで、垂直解像度および水平解像度を設定します。
4. [OK] をクリックして [フォーム ファクタのプロパティ] ダイアログ ボックスを閉じます。その後、もう一度 [OK] をクリックして [オプション] ダイアログ ボックスを閉じます。

メモ：
デザイナを開いたままオプションを変更した場合は、変更を有効にするためにデザイナを閉じて再度開きます。

参照

関連項目

[フォーム ファクタのプロパティ] ダイアログ ボックス (デバイス)
[フォーム ファクタ] ([オプション] ダイアログ ボックス - [デバイス ツール])

方法 : 既定のデバイスを変更する (マネージ プロジェクト)

マネージ プロジェクトの既定のターゲット デバイスを変更するには、以下の手順を使用します。

マネージ プロジェクト用に新しく既定のターゲット デバイスを選択するには

1. ソリューション エクスプローラで [<プロジェクト名>] を右クリックします。
2. ショートカット メニューの [プロパティ] をクリックし、[デバイス] ペインを選択します。
3. [ターゲット デバイス] ボックスのドロップダウン リストを使用して、新しい既定のターゲットを選択します。

参照

処理手順

方法 : 既定のデバイスを変更する (ネイティブ プロジェクト)

関連項目

[配置] ダイアログ ボックス (デバイス)

その他の技術情報

マネージ デバイス プロジェクトの作成および開発

方法：プラットフォーム呼び出しを使用して Wave ファイルを再生する (デバイス)

次のサンプル コードは、モバイル デバイスでプラットフォーム呼び出し (PInvoke) を使用して wave サウンド ファイルを再生する方法を示しています。

使用例

このサンプル コードでは、モバイル デバイスで `PlaySound` を使用してサウンド ファイルを再生します。このコードでは、System.Runtime.InteropServices を使用して、Compact FrameWork の CoreDll.DLL の `PlaySound` メソッドを呼び出します。

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.Runtime.InteropServices;

namespace MobileSoundPInvoke
{
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.MainMenu mainMenu1;

        public Form1()
        {
            InitializeComponent();
            PlaySound(".\\sound.wav");
        }

        #region Windows Form Designer generated code
        private void InitializeComponent()
        {
            this.mainMenu1 = new System.Windows.Forms.MainMenu();
            this.Menu = this.mainMenu1;

            this.Text = "Form1";
        }

        #endregion
        protected override void Dispose(bool disposing)
        {
            base.Dispose(disposing);
        }
        static void Main()
        {
            Application.Run(new Form1());
        }
        private enum Flags
        {
            SND_SYNC = 0x0000,
            SND_ASYNC = 0x0001,
            SND_NODEFAULT = 0x0002,
            SND_MEMORY = 0x0004,
            SND_LOOP = 0x0008,
            SND_NOSTOP = 0x0010,
            SND_NOWAIT = 0x00002000,
            SND_ALIAS = 0x00010000,
            SND_ALIAS_ID = 0x00110000,
            SND_FILENAME = 0x00020000,
            SND_RESOURCE = 0x00040004
        }

        [DllImport("CoreDll.DLL", EntryPoint = "PlaySound", SetLastError = true)]
        private extern static int MobilePlaySound(string szSound, IntPtr hMod, int flags);
```

```
        public void PlaySound(string fileName)
        {
            MobilePlaySound(fileName, IntPtr.Zero, (int)(Flags.SND_ASYNC | Flags.SND_FILENA
ME));
        }

    }
}
```

コードのコンパイル方法

- 新しい C# Smartphone アプリケーション プロジェクトを Visual Studio で作成し、**MobileSoundPInvoke** という名前を付けます。
- 上記のサンプルのコードをコピーし、コンソール アプリケーションの **MobileSoundPInvoke** プロジェクトの Form1.cs ファイルに貼り付けます。

堅牢性の高いプログラム

- 適切なパラメータを C の `MobilePlaySound (string szSound, IntPtr hMod, int flags)` 関数に渡します。このとき、.wav ファイルのパスとファイル名を含めます。

セキュリティ

セキュリティの詳細については、「.NET Framework Security」を参照してください。

参照

処理手順

プラットフォーム呼び出しの技術サンプル

その他の技術情報

スマート デバイスのサンプル

スマート デバイスのチュートリアル

C++ での明示的な PInvoke (DllImport 属性) の使用方法

Creating a P/Invoke Library

http://pinvoke.net/

デバイス プロジェクト内のコード スニペットの管理

Visual Basic には、マウスを数回クリックするだけでアプリケーションに挿入できる IntelliSense コード スニペットのコード ライブラリが付属します。各スニペットは、それぞれ完結したプログラミング タスクを実行します。また、独自のスニペットを作成してライブラリに追加し、後で必要に応じてそれらを使用することもできます。

Visual Studio には、デバイス プロジェクトだけに関係する追加のスニペットがあります。これらのスニペットのショートカットは、どれも sd という文字で始まります。詳細については、「Visual Basic の IntelliSense コード スニペット」を参照してください。

参照

その他の技術情報

マネージ デバイス プロジェクトの作成および開発

マネージ デバイス プロジェクトのデータ

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

Visual Studio 2005 には、デバイス ソリューション向けの新しいデータベース管理ツールが用意されています。新しいデータベースの作成や、データベースのスキーマの変更に加えて、パスワードの設定やデータベースの圧縮など、その他の管理タスクを処理するための機能を使用できます。

次のセクションは、Visual Studio 2005 SP1 に合わせて更新されています。

このセクションの内容

データ アクセスの概要 (マネージ デバイス プロジェクト)

デバイス プロジェクトでデータ アプリケーションを処理するために、Visual Studio で使用できる機能について説明します。

結果セットおよびデータセット (デバイス)

この 2 つの手法の違いと、それぞれどのような場合に使用するのかを説明します。

型指定された ResultSet の生成

ResultSet の機能について詳しく説明します。

デバイス プロジェクトにおけるデータ ソースの管理

SQL Server Mobile データベースまたは SQL Server Compact Edition データベースの作成、テーブル スキーマの変更、パスワードの管理など、共通タスクの手順を一覧表示します。

参照

処理手順

チュートリアル : データベース マスター/詳細アプリケーション

概念

Visual Studio でのデータへの接続の概要

その他の技術情報

クライアント データ アプリケーションの作成

データに関するチュートリアル

.NET Compact Framework を使用したデバイスのプログラミング

SQL Server Mobile Edition Books Online Home Page

.NET Compact Framework を使用したデバイスのプログラミング

データ アクセスの概要 (マネージ デバイス プロジェクト)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

データを操作するデバイス プロジェクトを開発するための Visual Studio 環境は、デスクトップのデータ アプリケーションを開発する環境とほぼ同じです。デバイス用のマネージ データ アプリケーションは、.NET Compact Framework でサポートされている ADO.NET 名前空間に依存しています。この組み合わせは、デバイスのデータ ストアが通常はサーバー上のデータから切断されているため定期的にしかデータ ストアが同期されないアプリケーションに役立ちます。

デバイスのデータ : Visual Studio 2005 の新機能

次のセクションは、Visual Studio 2005 SP1 に合わせて更新されています。

- SQL Server Mobile データベースまたは Microsoft SQL Server 2005 Compact Edition データベースの作成、パスワードの割り当て、スキーマの変更、およびその他の基本的なデータベース管理タスクを Visual Studio IDE を使用して行うことができます。
- 型指定されたデータセット、結果セット、ビジネス オブジェクト、SQL Server データベース、SQL Mobile データベース、SQL Server Compact Edition データベース、および Web サービスを、データ ソースとして使用できます。
- これらのデータ ソースを [データ ソース] ウィンドウから Windows フォームへドラッグ アンド ドロップすることで、目的のデータ バインド コントロールを生成できます。

メモ :
.NET Compact Framework 1.0 を対象にするプロジェクトでは、データ ソース構成ウィザードは使用できません。

SQL Server Mobile Edition および SQL Server Compact Edition のアーキテクチャ

次のセクションは、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

SQL Server Mobile および SQL Server Compact Edition は、ごくたまに接続されるデバイスにとっての最適なローカル データ エンジンです。マネージ アプリケーションの場合は .NET Compact Framework を使用し、ネイティブ アプリケーションの場合は Microsoft Visual C++ for Devices を使用してプログラムを作成できます。詳細については、「SQL Server Mobile Architecture」を参照してください。

SQL Server Mobile Edition および SQL Server Compact Edition の一般的な用途

次のセクションは、Visual Studio 2005 SP1 に合わせて更新されています。

SQL Server Mobile および SQL Server Compact Edition は、それほど頻繁には接続されないモバイル デバイスのデータ アクセス形態に対するソリューションです。企業のモバイル アプリケーションでは、接続を使用できないときでもデータ操作が要求される場面が多くあります。SQL Server Mobile および SQL Server Compact Edition では、接続を使用できるときに SQL Server と同期できる、機能豊富なリレーショナル ストアを提供することで、そのような利用形態に対応します。詳細については、「Typical Uses of SQL Server Mobile」を参照してください。

メモ :
SQL Server Mobile および SQL Server Compact Edition は、Tablet PC アプリケーションのデータ ストアとしても、デバイス アプリケーションのデータ ストアとしても使用できます。Visual Studio 2005 または SQL Server 2005 がインストールされていれば、ラップトップ コンピュータやデスクトップ上でも実行できます。詳細については、「Building a SQL Server Mobile Application for Tablet PCs」を参照してください。

SQL Server Mobile および SQL Server Compact Edition の機能

次のセクションは、Visual Studio 2005 SP1 に合わせて更新されています。

SQL Server Mobile および SQL Server Compact Edition には豊富な機能があり、.NET Compact Framework アプリケーションの一部として、または、スマート デバイス上の独立したインストールとしてそれらの機能を使用できます。データは、オフラインの状態で操作し、後でサーバーと同期できます。詳細については、「SQL Server Mobile Features」を参照してください。

SQL Server データベースの設計と管理

デバイス向けの効果的なデータ アプリケーションを開発するには、適切なデータベース設計と SQL Server データベース エンジンについての理解が必要です。データベースの保守方法やセキュリティで保護する方法、データベース内のデータにアクセスしたりデータベース内のデータを変更したりする方法、データベースを効果的に照会する方法、および、データベースのパフォーマンスを最大限に高める方法を熟知している必要があります。詳細については、「Working with Databases (SQL Server Mobile)」および「Enhancing Performance (SQL Server Mobile)」を参照してください。

サーバーとの接続

次のセクションは、Visual Studio 2005 SP1 に合わせて更新されています。

SQL Server Mobile および SQL Server Compact Edition は、マージ レプリケーション、リモート データ アクセス (RDA: Remote Data Access)、および、サーバーでのセキュリティ計画と実装をサポートします。詳細については、「Managing Connectivity (SQL Server Mobile)」を参照してください。

プログラムによる共通タスクの実装

共通のタスクをプログラムで実装するための手順については、「How To (SQL Server Mobile)」を参照してください。

ローカル セキュリティ

SQL Server Mobile および SQL Server Compact Edition のデータベース エンジンには、デバイスのローカル データベースをセキュリティで保護するための、パスワードによる保護と暗号化の機能があります。また、接続のセキュリティに関するオプションも用意されています。詳細については、「Securing Databases (SQL Server Mobile)」を参照してください。

参照

処理手順

チュートリアル : データベース マスター/詳細アプリケーション
方法 : サンプル データベースをインストールする

概念

データの新機能
接続文字列のセキュリティ保護
Visual Studio でのデータへの接続の概要

その他の技術情報

SQL Server Mobile Edition Books Online Home Page
クライアント データ アプリケーションの作成
データに関するチュートリアル
.NET Compact Framework を使用したデバイスのプログラミング
マネージ デバイス プロジェクトのデータ
データへのアクセス (Visual Studio)

結果セットおよびデータセット (デバイス)

Visual Studio 2005 では、データ ソース ウィザードを使用して新しいデータ ソースを作成すると、データセットと結果セットの両方が生成されます。結果セットのコードはデータセットのコードに比べ、パフォーマンスが高く、冗長性は大幅に低くなっています。一方、データセットは、複数のテーブルを保持できる点、およびテーブル間のリレーションシップに関する情報を保持できる点で優れています。この 2 つのテクノロジのプログラミング モデルは大きく異なります。

目的に合わせて、これらの構造体のどちらか一方または両方を生成することを選択できます。この機能は、上級開発者のために用意されています。詳細については、「方法 : SqlCeResultSet コードを生成する (デバイス)」を参照してください。

参照

処理手順

方法 : SqlCeResultSet コードを生成する (デバイス)

概念

データ アクセスに関する推奨事項

その他の技術情報

SQL Server Compact Edition ポータル

マネージ デバイス プロジェクトのデータ

型指定された ResultSet の生成

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

XSD スキーマ ファイルの **CustomTool** プロパティを **MSResultSetGenerator** に設定すると、通常の型指定された **DataSet** データ ソース オブジェクトではなく、型指定された ResultSet データ ソース オブジェクトが生成されます。ResultSet は、高速のデータベース カーソルで、UI のデータ バインディング、データの後方スクロールと前方スクロール、およびデータベース内のデータの更新をサポートします。ResultSet は、常時接続されたモデルとしてデータベースへの接続を維持します。

次のセクションは、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

型指定された ResultSet の特徴

生成した型指定された ResultSet オブジェクトは、型指定された **DataSets** によく似たデータベース テーブルへのタイプ セーフなアクセスを提供します。したがって、生成されたコードは、アプリケーションとデータベース間で受け渡しされたデータがコンパイル時にデータベース スキーマに正確に一致することを確認します。型指定された ResultSet で生成されたコードは、SQL Server Mobile Edition または SQL Server Compact Edition の ResultSet 基本クラスを拡張し、対象となるデータベース テーブル固有のプロパティおよびメソッドを提供します。

型指定された ResultSet 用に生成されたコードの特徴を次に示します。

- **コンストラクタ**　生成した型指定された ResultSet には、2 つのコンストラクタが含まれています。既定のコンストラクタは、デザイン時のシナリオと実行時の基本的なシナリオの両方に使用されます。Visual Studio のデザイン時には、既定のコンストラクタで、ローカルの開発コンピュータに存在するデータベースへの接続を確立する必要があります。したがって、生成されたコードには、ローカルの SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベース ファイルの接続文字列が含まれています。既定のコンストラクタは、実行時に、実行中のアプリケーションと同じディレクトリからローカルのデータベース ファイルに切り替えます。これは、実行時の基本的なシナリオです。
- **接続文字列**　実行時のシナリオがさらに複雑な場合、たとえば、実行時のディレクトリ以外のデバイスの場所に SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベース ファイルが存在する場合や、接続文字列がその他の方法 (パスワードなど) で変更される場合、オーバーロードされたコンストラクタを使用できます。オーバーロードされたコンストラクタは、カスタム接続文字列およびカスタム **ResultSetOptions** フラグという 2 つの引数を使用します。
- **接続プロパティ**　接続プロパティは、データベースのデータにアクセスするために、型指定された ResultSet で使用される有効な **SqlCeConnection** オブジェクトです。接続プロパティはパブリック プロパティであり、型指定された ResultSet に対する接続の状態を管理するためにアクセスを有効にします。たとえば、トランザクションによっては、接続を閉じる必要がある場合があります。
- **テーブル列へのタイプ セーフ アクセス**　生成されたコードは、データ テーブル内の各列にプロパティ アクセサを作成します。プロパティの型は、基になるデータベースの型を .NET Framework 型に割り当てることによって決まります。たとえば、`nvarchar` は、.NET Framework の文字列型に割り当てられます。生成されたプロパティでは、`Get` アクセサと `Set` アクセサの両方がサポートされているため、基になるデータベースの `nvarchar` ではなく .NET Framework の文字列型を使用して、データをプルおよびプッシュできます。各列には、**IsxxxDBNull** メソッドおよび **SetxxxDBNull** メソッドも含まれています。この場合の xxx は列名で、データベースへの DBNull の取得および設定を容易にします。
- **AddxxxRecord メソッド** (xxx はテーブル名)　このメソッドを使用すると、新しい行をデータベース テーブルに追加できます。**AddxxxRecord** メソッドは、データ テーブルの列ごとに 1 つのパラメータを使用します。ただし、データベース エンジンによって値が割り当てられている自動インクリメント列は除きます。各パラメータは .NET Framework 型として再度宣言されます。その結果、アプリケーションでの使用が容易になり、基になるデータベース型を抽象化できます。このメソッドを呼び出した後、更新を呼び出す必要はありません。一度呼び出すと、新しい行がデータベースにコミットされます。
- **DeletexxxRecord メソッド** (xxx はテーブル名)　このメソッドは、データベースから現在の行を削除します。DataSet とは異なり、この遷移はキャッシュされません。そのため、一度呼び出すと、行がデータベースから削除されます。このメソッドを呼び出した後、更新を呼び出す必要はありません。
- **Bind メソッド**　Bind メソッドは BindingSource という 1 つのパラメータを受け取ります。このパラメータは、フォーム上の 1 つ以上の UI コントロールにデータバインドされます。その後、このメソッドが、型指定された ResultSet のインスタンスに **BindingSource** をデータバインドすると、データバインディング チェーンが完成し、コントロールがデータベース内のデータを直接表示できるようになります。

メモ：

デザイン時のシナリオでは、SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースのハードコーディングされた接続文字列が必要です。この接続文字列は、たとえば、開発者間でプロジェクトが共有されている場合に最新のものでなくなる場合があります。その結果、[データソース] ウィンドウ、Windows フォーム デザイナ、またはその両方でプロジェクトを開くことができない場合があります。この問題を解決するには、型指定された ResultSet のコードを再生成します。ソリューション エクスプローラで、XSD スキーマ ファイルを右クリックし、[カスタム ツールの実行] をクリックします。

参照

関連項目

SqlCeResultSet

概念

結果セットおよびデータセット (デバイス)

その他の技術情報

マネージ デバイス プロジェクトのデータ

デバイス プロジェクトにおけるデータ ソースの管理

以下のセクションでは、デバイス プロジェクトに一般的なデータ タスクを実装する方法について説明します。

メモ :
.NET Compact Framework 1.0 を対象にするプロジェクトでは、データ ソース構成ウィザードは使用できません。

このセクションの内容

参照

概念

Visual Studio でのデータへの接続の概要

その他の技術情報

SQL Server Mobile Edition Books Online Home Page

マネージ デバイス プロジェクトのデータ

SQL Server Mobile Edition および SQL Server Compact Edition データベースの使用

このトピックは、Visual Studio 2005 SP1 に合わせて新たに追加されました。

Visual Studio 2005 には、スマート デバイスの開発者がデータベース アプリケーションを作成する際に使用できる SQL Server Mobile Edition が付属しています。Visual Studio 2005 Service Pack 1 では、デスクトップ開発者向けの機能が追加された新バージョン、Microsoft SQL Server 2005 Compact Edition を使用できるようになりました。SQL Server Compact Edition と SQL Server Mobile Edition は、スマート デバイスの開発者向けに同じ機能セットを装備しています。

ただし、Visual Studio Service Pack 1 が備える豊富なデザイン時データ機能を存分に活用するためには、SQL Server Compact Edition 3.1 と .NET Compact Framework 2.0 を使用してデータ アプリケーションを作成する必要があります。Visual Studio SP1 と SQL Server Compact Edition Tools for Visual Studio 2005 SP1 の両方をインストールする必要があります。詳細については、「方法 : サンプル データベースをインストールする」を参照してください。

次の表には、Visual Studio 統合開発環境 (IDE) を使用して効率的にデータベースを操作するための、有用な情報へのリンクが記載されています。

共通のデータベース タスク

方法 : データベースを作成する (デバイス)	方法 : データ アプリケーション用の概要ビューと編集ビューを生成する (デバイス)
方法 : マスター/詳細アプリケーションを作成する (デバイス)	方法 : データベースの列を管理する (デバイス)
方法 : パラメータ付きクエリを作成する (デバイス)	方法 : データベースのパスワードを管理する (デバイス)
方法 : SqlCeResultSet コードを生成する (デバイス)	方法 : データベース内のデータをプレビューする (デバイス)
方法 : データベースを縮小および修復する (デバイス)	方法 : デバイス プロジェクトにデータベースを追加する
方法 : データベースのテーブルを管理する (デバイス)	方法 : データベースのインデックスを管理する (デバイス)
方法 : ナビゲーション ボタンを追加する (デバイス)	方法 : データベースのデータの変更を永続化する (デバイス)

参照

概念

SQL Server Compact Edition と .NET Compact Framework

方法：データベースを作成する (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

SQL Server Mobile データベースまたは SQL Server Compact Edition データベースは、プロジェクトを開いているかどうかに関係なく作成できます。データベースをプロジェクトに含める場合は、データベースをプロジェクトのデータ ソースとしてプロジェクト内で作成することを検討してください。データベースをプロジェクトの外で作成した場合でも、後でプロジェクトに追加できます。詳細については、「方法 : デバイス プロジェクトにデータベースを追加する」を参照してください。

メモ：

同じ SQL Server Mobile データベースまたは SQL Server Compact Edition データベースに対し、ネットワーク共有を介して複数のプロセスが同時にアクセスすることはできません。同時にアクセスした場合、2 つ目のプロセスでファイル共有違反エラーが発生します。ただし、1 つのプロセスが、データベースに対して複数の接続を確立することは可能です。たとえば、特定のプロセスが、ある接続では挿入クエリを、もう 1 つの接続では選択クエリを同時に実行しても、エラーは発生しません。ネットワーク共有を介してデータベース ファイルを開く場合は、DB_MODE_SHARE_EXCLUSIVE ファイル モードを使用します。詳細については、「How to: Set the File Mode When Opening a Database with OLE DB (Programmatically)」を参照してください。

プロジェクトの外部にデータベースを作成するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [データ接続] を右クリックし、[接続の追加] をクリックして [接続の追加] ダイアログ ボックスを開きます。
3. [変更] をクリックして、[データ ソースの変更] ダイアログ ボックスを開きます。
4. [データ ソース] ボックスで、[Microsoft SQL Server Mobile Edition] を選択し、[OK] をクリックします。

   または

   [データ ソース] ボックスで、[Microsoft SQL Server Compact Edition] を選択し、[OK] をクリックします。
5. [Microsoft SQL Server Mobile Edition] または [Microsoft SQL Server Compact Edition] を選択し、[OK] をクリックします。
6. 「処理を続行して接続を構成するには」に進みます。

プロジェクトの内部にデータベースを作成するには

1. プロジェクトを開いた状態で、[データ] メニューの [新しいデータ ソースの追加] をクリックします。

   データ ソース構成ウィザードが開きます。
2. [データソースの種類を選択] ページで、[データベース] を選択し、[次へ] をクリックします。
3. [データ接続の選択] ページで、[新しい接続] をクリックして [接続の追加] ダイアログ ボックスを開きます。
4. [変更] をクリックして、[データ ソースの変更] ダイアログ ボックスを開きます。
5. [Microsoft SQL Server Mobile Edition] または [Microsoft SQL Server Compact Edition] を選択し、[OK] をクリックします。
6. 「処理を続行して接続を構成するには」に進みます。

処理を続行して接続を構成するには

1. [接続の追加] ダイアログ ボックスで、[マイ コンピュータ] を選択します。
2. [作成] をクリックします。
3. [SQL Server 2005 Mobile Edition データベースの新規作成] ダイアログ ボックスで、新しいデータベースの絶対パス (c:\MyDB など) を

入力します。

または

[SQL Server 2005 Compact Edition データベースの新規作成] ダイアログ ボックスで、新しいデータベースの絶対パス (c:\MyDB など) を入力します。

4. [新しいパスワード] ボックスと [パスワードの確認入力] ボックスに、新しいデータベースのパスワード (MyPassword など) を入力し、[OK] をクリックします。

セキュリティに関するメモ :
実際のアプリケーションで使用するプロジェクトの場合は、強力なパスワードを選択してください。

5. [OK] をクリックすると、[接続の追加] ダイアログ ボックスに戻ります。
6. [接続の追加] ダイアログ ボックスで、[接続の確認] をクリックして、接続が作成されたことを確認します。

   接続のテストが成功したことを示すメッセージが表示されます。
7. [OK] をクリックして [接続の追加] ダイアログ ボックスに戻り、[OK] をクリックしてそのダイアログ ボックスを閉じます。

   プロジェクト内でデータベースおよびデータセットを作成する場合にのみ、次の手順に従います。
8. [データ接続の選択] ページで [はい、重要情報を接続文字列に含めます。] を選択します。

セキュリティに関するメモ :
実際のアプリケーションで使用するプロジェクトの場合は、重要情報を接続文字列に含めない (*exclude*) オプションを選択します。少なくとも、重要情報を暗号化する必要があります。

9. [次へ] をクリックします。

   データ ファイルを現在のプロジェクトに含めるかどうかを確認する [ローカル データベース ファイル] メッセージ ボックスが表示されます。

   [はい] をクリックします。
10. [データベース オブジェクトの選択] ページで、プロジェクトに含めるテーブルまたはその他のオブジェクトを選択します。
11. [完了] をクリックします。

    [データ] メニューの [データ ソースの表示] をクリックすると、新しいデータベースが [データ ソース] ウィンドウにデータセットとして表示されます。データベースにテーブル、列、制約などを追加するには、「デバイス プロジェクトにおけるデータ ソースの管理」を参照してください。

参照

処理手順

方法 : サンプル データベースをインストールする

その他の技術情報

Create Database Dialog (SQL Server Mobile)

デバイス プロジェクトにおけるデータ ソースの管理

方法：マスター/詳細アプリケーションを作成する (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

以下の手順では、SQL Server Mobile データベースまたは SQL Server Compact Edition データベースが存在し、テーブルのリレーションシップが [データ ソース] ウィンドウで利用可能になっているものと想定しています。詳細については、「方法 : データベースを作成する (デバイス)」を参照してください。

詳細テーブルをドラッグする際は、グリッド全体ではなく、目的のために必要な列だけをドラッグすることをお勧めします。列を選択するには、テーブル名の右側にある矢印をクリックします。

次の手順は、デバイス プロジェクトが開いていること、および、データ ソースの構成が済んでいることを前提に記述されています。

マスター/詳細アプリケーションを作成するには

1. デザイナで、[データ ソース] ウィンドウからフォーム上にマスター テーブルをドラッグします。
2. [データ ソース] ウィンドウで、マスター テーブルを展開して、詳細テーブルを表示します。
3. マスター テーブル ノードで詳細テーブルを見つけ、フォーム上にドラッグします。

メモ：
これは、マスター テーブル内に表示された状態の詳細テーブルであり、マスター テーブルと同じツリー レベルにある詳細テーブルではありません。

   外部キー制約から、マスター/詳細リレーションシップがデザイナによって自動的に検出されます。詳細については、「チュートリアル : データベース マスター/詳細アプリケーション」を参照してください。
4. アプリケーションに合うようにフォーム上のコントロールを調整します。

参照

処理手順

チュートリアル : データベース マスター/詳細アプリケーション

方法 : サンプル データベースをインストールする

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法：パラメータ付きクエリを作成する (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

以下の手順では、[データ ソース] ウィンドウで SQL Server Mobile データベースまたは SQL Server Compact Edition データベースが利用可能になっているものと想定しています。詳細については、「方法 : データベースを作成する (デバイス)」または「方法 : デバイス プロジェクトにデータベースを追加する」を参照してください。

ユーザーがさまざまな値をパラメータとして入力できるようにするには、クエリの作成時に疑問符 ("?") をパラメータとして使用します。Windows フォーム デザイナでスマート タグを使用してクエリを作成する場合は、次の手順に示すように、Windows フォーム内にユーザー インターフェイスが自動生成されます。データセット デザイナの [TableAdapter] からクエリを作成する場合は、最後の手順に示すように、ユーザー インターフェイスは自動生成されません。

Windows フォーム デザイナを使用してパラメータの指定をセットアップするには

1. デザイナで、[データ ソース] ウィンドウからフォーム上に DataGrid 形式または Details 形式のテーブルをドラッグします。

   形式を選択するには、テーブル名の右側にある矢印をクリックします。

2. ドラッグしたコンポーネントのスマート タグをクリックし、ショートカット メニューの [クエリの追加] をクリックします。

   マウスを使用していない場合は、ショートカット キー (Shift + Alt + F10 キー) を使用して、[タスク] ダイアログ ボックスを開きます。

3. [検索条件ビルダ] ダイアログ ボックスで、[新しいクエリ名] を選択します。

   既定の名前を使用するか、独自の名前を作成します。

4. これで、[クエリ テキスト] ボックスの SQL ステートメントを変更するか、[クエリ ビルダ] をクリックすることにより、使用するパラメータを指定できます。

[クエリ テキスト] ボックスを使用してパラメータを指定するには

1. `WHERE` 句を `SELECT` ステートメントの末尾に追加します。
2. [OK] をクリックして [検索条件ビルダ] ダイアログ ボックスを閉じます。

   クエリにバインドされたボタンが、デザイナのフォーム上に表示されます。

クエリ ビルダを使用してパラメータを指定するには

1. [クエリ ビルダ] ダイアログ ボックスで、次の操作のいずれかを実行します。

   - SQL ステートメント ペインで `WHERE` 句を追加します。

     または

   - 適切な [列] リストの [フィルタ] にパラメータを入力します。

     この方法では、SQL ステートメント ペインに自動的に `WHERE` 句が書き込まれます。

2. [OK] をクリックして [クエリ ビルダ] ダイアログ ボックスを閉じます。
3. [OK] をクリックして [検索条件ビルダ] ダイアログ ボックスを閉じます。

   クエリにバインドされたボタンが、デザイナのフォーム上に表示されます。

データセット デザイナを使用してパラメータを指定するには

1. ソリューション エクスプローラで、.xsd ファイルを右クリックし、[開く] をクリックします。

2. データセット デザイナで、[TableAdapter] を右クリックし、[追加] をポイントして、ショートカット メニューの [クエリ] をクリックします。
3. TableAdapter クエリの構成ウィザードで、[SQL ステートメントを使用する] を選択し、[次へ] をクリックします。
4. [クエリの種類の選択] ページで、[単一の値を返す SELECT] を選択し、[次へ] をクリックします。
5. [SQL SELECT ステートメントの指定] ページで、[クエリ ビルダ] をクリックします。

   必要な場合は、ここで `WHERE` 句を追加できます。
6. このトピックで前述したように、クエリ ビルダを使用します。

メモ：
TableAdapter クエリの構成ウィザードを使用してクエリを作成したときは、ユーザー インターフェイス要素は自動生成されません。

参照

処理手順

方法 : パラメータ クエリを Windows アプリケーションのフォームに追加する
チュートリアル : パラメータ クエリ アプリケーション
方法 : サンプル データベースをインストールする

関連項目

[検索条件ビルダ] ダイアログ ボックス

概念

クエリおよびビュー デザイナ ツール

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法：SqlCeResultSet コードを生成する (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

次のセクションは、Visual Studio 2005 SP1 に合わせて更新されています。

以下の手順では、[データ ソース] ウィンドウで SQL Server Mobile データベースまたは Microsoft SQL Server 2005 Compact Edition データベースが利用可能になっているものと想定しています。詳細については、「方法 : データベースを作成する (デバイス)」または「方法 : デバイス プロジェクトにデータベースを追加する」を参照してください。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

デバイス プロジェクトの場合、Visual Studio は、既定ではデータセットのみのコードを生成します。この既定値を変更して、結果セットのコードを生成することや、両方のコードを生成することもできます。2 種類の違いの詳細については、「結果セットおよびデータセット (デバイス)」を参照してください。

結果セット オプションおよび結果セット/データセット オプションは、SQL Server Mobile 接続または SQL Server Compact Edition 接続に対して作成された .xsd ファイルでのみサポートされています。データセット オプションは、すべての接続でサポートされているため、既定でコードが生成されます。

メモ：
既存のデータセット アプリケーションを結果セット アプリケーションに変換する場合は、[カスタム ツール] プロパティを [MSDataSetResultSetGenerator] に設定します。このように設定すると、両方の種類のデータ アクセス クラスが生成されます。これにより、ビルド エラーを起こさずにコードを一方から他方へ移行できます。コードの移行が終わったら、[カスタム ツール] プロパティを [MSResultSetGenerator] に設定します。ビルドし直して、すべてのデータセット使用部分がコードから削除されていることを確認します。

結果セットのみのコードを生成するには

1. ソリューション エクスプローラで、データベースの .xsd ファイルを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [カスタム ツール] の値を [MSResultSetGenerator] に設定します。
3. ソリューションをビルドし直します。

データセットのみのコードを生成するには

1. ソリューション エクスプローラで、データベースの .xsd ファイルを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [カスタム ツール] の値を [MSDataSetGenerator] に設定します。

   この値は既定値です。
3. ソリューションをビルドし直します。

結果セットとデータセットの両方のコードを生成するには

1. ソリューション エクスプローラで、データベースの .xsd ファイルを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [カスタム ツール] の値を [MSDataSetResultSetGenerator] に設定します。
3. ソリューションをビルドし直します。

参照

処理手順

方法 : サンプル データベースをインストールする

概念

結果セットおよびデータセット (デバイス)

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法：データ アプリケーション用の概要ビューと編集ビューを生成する (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

データ フォームを使用して、データ グリッドの単一データ行を表示および編集します。

データ フォームのユーザー インターフェイスは、データ グリッドで選択した行の概要ビューを表示する [表示] ダイアログ ボックスと、その行を編集できる [編集] ダイアログ ボックスの 2 つのダイアログ ボックスで構成されています。

- 実行中のアプリケーションで [表示] ダイアログ ボックスを開くには、デバイス エミュレータでデータ グリッドの特定の行をダブルクリックするか、デバイスで行をタップします。
- [編集] ダイアログ ボックスを開くには、データグリッドが表示されているときに [新規作成] をクリック (タップ) します。この操作により、データグリッドに新しい行が作成されます。また、[表示] ダイアログ ボックスが表示されているときに [編集] をクリック (タップ) しても、このダイアログ ボックスを開くことができます。

データ フォームは、カスタマイズ用のテンプレートとしてデザインされています。このカスタマイズの一部として、データベースへの変更を検証およびコミットするために必要なコードを追加してください。

データ フォームを使用して行われた変更は、データベースで保持されません。変更を保持する方法の詳細については、「方法 : データベースのデータの変更を永続化する (デバイス)」を参照してください。

メモ：
この機能は、SQL Server Mobile、SQL Server Compact Edition、およびサーバー データベースのデータセットに対して作成されたデータ ソースでのみサポートされます。結果セット、ビジネス オブジェクト、および Web サービスではサポートされません。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

以下の手順では、[データ ソース] ウィンドウで SQL Server Mobile データベースまたは SQL Server Compact Edition データベースが利用可能になっているものと想定しています。詳細については、「方法 : データベースを作成する (デバイス)」を参照してください。このトピックでは、プロジェクトのデータ ソースとして、既存の SQL Server データベースを追加する方法についても説明しています。

データ フォームを生成するには

1. デザイナで、[データ ソース] ウィンドウからフォーム上にデータグリッド形式のテーブルをドラッグします。

   テーブル名の右側にある矢印をクリックすることにより、形式を選択できます。

2. ドラッグしたコンポーネントのスマート タグをクリックし、ショートカット メニューの [データフォームを生成します] をクリックします。マウスを使用していない場合、ショートカット キー (Shift+Alt+F10 キー) を使用して、[タスク] ダイアログ ボックスを開きます。

メモ：
ダイアログ ボックスは、デザイナで生成されたときに一時的に表示されます。これらのダイアログ ボックスがプロジェクトの一部であることを確認するには、ソリューション エクスプローラでこれらのダイアログ ボックスを検索します。

実行中のアプリケーションのデータを変更するには

1. データ アプリケーションを起動します。

   データが設定されたデータ グリッドが表示されます。

2. データ行をダブルクリックします。

   その行の概要ビューが [表示] ダイアログ ボックスに表示されます。このビューは、コンテンツが含まれた列ごとのラベルとデータで構成されて

います。つまり、[表示] ダイアログ ボックスでは、値が DBNULL の列が表示されません。

3. メイン メニューの [編集] をクリックして [編集] ダイアログ ボックスを開きます。

   すべての列が表示される [編集] ダイアログ ボックスを使用してデータを変更してから、[OK] をクリックします。

   変更されたデータがデータグリッドに表示されます。

実行中のアプリケーションで新しい行をデータ グリッドに作成するには

1. データ グリッドが開いている状態で、メイン メニューの [新規作成] をクリックします。

   [編集] ダイアログ ボックスが表示されます。このダイアログ ボックスを使用して新しいデータ行を追加します。

2. [OK] をクリックします。

   新しい行がデータ グリッドに追加されます。

参照

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法：データベースの列を管理する (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

以下の手順では、[サーバー エクスプローラ] ウィンドウで SQL Mobile データベースまたは Microsoft SQL Server 2005 Compact Edition データベースが利用可能になっているものと想定しています。詳細については、「方法 : データベースを作成する (デバイス)」を参照してください。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

列を追加するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、テーブルを表示するデータ接続を展開します。
3. 列を追加するテーブルを右クリックし、ショートカット メニューの [テーブル スキーマの編集] をクリックします。
4. [テーブルの編集] ウィンドウで、列およびそのプロパティを追加し、[OK] をクリックします。

列のプロパティを編集するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、テーブルを表示するデータ接続を展開します。
3. プロパティを編集する列を含んだテーブルを右クリックし、ショートカット メニューの [テーブル スキーマの編集] をクリックします。
4. 編集を行って、[OK] をクリックします。

メモ：
参照整合性に違反する編集 (たとえば、他の制約から参照されている列の Primary Key プロパティを No に変更するなど) は実行できません。

テーブルから列を削除するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、テーブルを表示するデータ接続を展開します。
3. 列を削除するテーブルを右クリックし、ショートカット メニューの [テーブル スキーマの編集] をクリックします。
4. [テーブルの編集] ウィンドウで、削除する列を選択し、[削除] をクリックし、[OK] をクリックします。

メモ：
列に対する参照がすべて削除されるまでは、その列を削除できません。

参照

処理手順

方法 : サンプル データベースをインストールする

その他の技術情報

Column Properties

デバイス プロジェクトにおけるデータ ソースの管理

方法：データベースのパスワードを管理する (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

データベースの作成時にパスワードを設定したり、既存のデータベースでパスワードを変更したりできます。

セキュリティに関するメモ：
接続文字列にパスワードを含めることには、セキュリティ上のリスクが伴います。詳細については、「Securing Databases (SQL Server Mobile)」を参照してください。

次のサンプルの手順は、Pocket PC Windows フォーム アプリケーションを開いていることを前提にしています。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

データベースの作成時にパスワードを設定するには

1. [データ] メニューの [新しいデータ ソースの追加] をクリックします。
2. [データソースの種類を選択] ページで、[データベース] を選択し、[次へ] をクリックします。
3. [データ接続の選択] ページで、[新しい接続] をクリックして [接続の追加] ダイアログ ボックスを開きます。
4. [変更] をクリックし、[Microsoft SQL Server Mobile Edition] をクリックします。次に、[OK] をクリックします。

   または

   [変更] をクリックし、[Microsoft SQL Server Compact Edition] をクリックします。次に、[OK] をクリックします。
5. [接続の追加] ダイアログ ボックスで、[マイ コンピュータ] をクリックします。
6. [作成] をクリックします。
7. [SQL Server 2005 Mobile Edition データベースの新規作成] ダイアログ ボックスで、新しいデータベースの絶対パス (c:\MyDatabase.sdf など) を入力します。

   または

   [SQL Server 2005 Compact Edition データベースの新規作成] ダイアログ ボックスで、新しいデータベースの絶対パス (c:\MyDatabase.sdf など) を入力します。
8. [新しいパスワード] ボックスと [パスワードの確認入力] ボックスに、新しいデータベースのパスワードを入力し、[OK] をクリックします。

既存のデータベースのパスワードを変更するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. サーバー エクスプローラで、パスワードを変更するデータ ソースを右クリックします。
3. ショートカット メニューの [データベース プロパティ] をクリックします。
4. [データベース プロパティ] ダイアログ ボックスの [パスワードの設定] をクリックします。
5. 古いパスワードと新しいパスワードを指示に従って入力して、[OK] をクリックします。

参照

処理手順

方法 : データベースを作成する (デバイス)

その他の技術情報

Securing Databases (SQL Server Mobile)

デバイス プロジェクトにおけるデータ ソースの管理

方法：データベース内のデータをプレビューする (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

以下の手順では、[サーバー エクスプローラ] ウィンドウで SQL Server Mobile データベースまたは SQL Server Compact Edition データベースが利用可能になっているものと想定しています。詳細については、「方法 : データベースを作成する (デバイス)」を参照してください。

サーバー エクスプローラを使用してプロジェクト外のデータをプレビューしたり、[データ ソース] ウィンドウを使用してプロジェクト内からデータをプレビューしたりできます。ここでは、ビューを作成するクエリをパラメータ化することもできます。詳細については、「方法 : パラメータ付きクエリを作成する (デバイス)」を参照してください。

サーバー エクスプローラを使用してデータを表示するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、データ接続を展開してテーブルの一覧を表示します。
3. データを表示するテーブルを右クリックし、ショートカット メニューの [テーブル データの表示] をクリックします。

プロジェクト内の [データ ソース] ウィンドウを使用してデータを表示するには

1. [データ] メニューの [データ ソースの表示] をクリックします。
2. [データ ソース] ウィンドウで、データ (テーブルなど) を表示するデータ ソースを右クリックします。
3. ショートカット メニューの [データのプレビュー] をクリックします。
4. [データのプレビュー] ダイアログ ボックスの [プレビュー] をクリックします。

DataGrid コントロールでスマート タグを使用してデータを表示するには

1. デザイナ内で、DataGrid コントロールのスマート タグをクリックします。
2. [DataGrid タスク] ショートカット メニューの [データのプレビュー] をクリックします。
3. [データのプレビュー] ダイアログ ボックスで、プレビューするオブジェクトを選択し、[プレビュー] をクリックします。

   データが [結果] ボックスに表示されます。[データのプレビュー] ダイアログ ボックスの下部には、グリッド内の列数と行数も表示されます。

参照

処理手順

方法 : サンプル データベースをインストールする

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法：デバイス プロジェクトにデータベースを追加する

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

以下の手順では、[サーバー エクスプローラ] ウィンドウで SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースが利用可能になっているものと想定しています。詳細については、「方法 : データベースを作成する (デバイス)」を参照してください。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

SQL Server Mobile Edition または SQL Server Compact Edition のデータベースをデータ ソースとして追加するには

1. スマート デバイス プロジェクトを開いた状態で、[データ] メニューの [新しいデータ ソースの追加] をクリックします。
2. [データソースの種類を選択] ページで、[データベース] を選択し、[次へ] をクリックします。
3. [データ接続の選択] ページで、データベース名を含むデータ接続文字列を選択し、[次へ] をクリックします。

   データベースが既にサーバー エクスプローラで使用できる場合、この接続はドロップダウン ボックスに表示されます。データベース接続が一覧に表示されない場合は、[新しい接続] をクリックして [接続の追加] ダイアログ ボックスを開きます。その後、[参照] をクリックして [データベース ファイルを選択してください。] ダイアログ ボックスを開き、データベースに移動して [開く] をクリックします。[OK] をクリックして、[接続の追加] ダイアログ ボックスを閉じます。
4. [データ接続の選択] ページの [次へ] をクリックします。

   データ ファイルを現在のプロジェクトに含めるかどうかを確認する [ローカル データベース ファイル] メッセージ ボックスが表示されます。

   [はい] をクリックします。
5. [データベース オブジェクトの選択] ページで、プロジェクトのデータ ソースとして使用するオブジェクトを選択します。
6. [完了] をクリックします。

   SQL Server Mobile データベースまたは SQL Server Compact Edition データベースから選択したオブジェクトがプロジェクトのデータ ソースとして [データ ソース] ウィンドウに表示されます。このウィンドウを開くには、[データ] メニューの [データ ソースの表示] をクリックします。

参照

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法：データベースを縮小および修復する (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

以下の手順では、[サーバー エクスプローラ] ウィンドウで SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースが利用可能になっているものと想定しています。詳細については、「方法 : データベースを作成する (デバイス)」を参照してください。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

データベースを縮小または修復するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、縮小または修復するデータ接続を右クリックし、ショートカット メニューの [データベース プロパティ] をクリックします。
3. ページの選択ペインで、[圧縮と修復] をクリックします。
4. [圧縮と修復] ページのオプションの中から選択します。

   このページの下部に、選択したオプションの説明が表示されます。
5. 選択した縮小オプションまたは修復オプションを起動するには [OK] をクリックします。データベースを現在の状態のままにするには [キャンセル] をクリックします。

参照

その他の技術情報

Shrink & Repair

デバイス プロジェクトにおけるデータ ソースの管理

方法：データベースのテーブルを管理する (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

以下の手順では、[サーバー エクスプローラ] ウィンドウで SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースが利用可能になっているものと想定しています。詳細については、「方法 : データベースを作成する (デバイス)」を参照してください。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

テーブルを追加するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、テーブルを追加するデータ接続を展開します。
3. [テーブル] を右クリックし、[テーブルの作成] をクリックします。
4. [新しいテーブル] ダイアログ ボックスで、[名前] ボックスにテーブル名を入力します。
5. 最初の列では、[列名]、[データ型]、[長さ]、[Null を許容]、[一意]、および [主キー] の値を入力します。追加の列についても続行します。
6. [OK] をクリックします。

   データ接続用のテーブルの一覧に、新しいテーブルが表示されます。

既存のテーブル スキーマを編集するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、編集するテーブル スキーマが存在するデータ接続を展開します。
3. 編集するテーブルを右クリックし、ショートカット メニューの [テーブル スキーマの編集] をクリックします。
4. [テーブル <テーブル名> の編集] ダイアログ ボックスで、変更を行い、[OK] をクリックします。

テーブルを削除するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、テーブルを削除するデータ接続を展開します。
3. [テーブル] で、削除するテーブルを右クリックし、ショートカット メニューの [テーブルの削除] をクリックします。
4. [オブジェクトの削除] ダイアログ ボックスで、[削除] をクリックし、[OK] をクリックします。

メモ：
テーブルに対する参照がすべて削除されるまでは、そのテーブルを削除できません。

参照

その他の技術情報

New Table
Edit Table (SQL Server Mobile)
Table Properties
デバイス プロジェクトにおけるデータ ソースの管理

方法：データベースのインデックスを管理する (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

以下の手順では、[サーバー エクスプローラ] ウィンドウで SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースが利用可能になっているものと想定しています。詳細については、「方法 : データベースを作成する (デバイス)」を参照してください。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

SQL Server Mobile Edition または SQL Server Compact Edition のデータベースにインデックスを作成するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、新しいインデックスを作成するデータ ソースを展開します。
3. 新しいインデックスを作成するテーブルを展開し、[インデックス] フォルダを右クリックします。
4. ショートカット メニューの [インデックスの作成] をクリックします。
5. [新しいインデックス] ダイアログ ボックスで、インデックスの名前を入力し、[追加] をクリックします。
6. [<テーブル> から列を選択] ダイアログ ボックスで、インデックス キーに追加する列を選択し、[OK] をクリックします。
7. [OK] をクリックして、[新しいインデックス] ダイアログ ボックスを閉じます。

インデックス プロパティの並べ替え順序のみを編集するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、プロパティを編集するインデックスが含まれるデータ接続とテーブルを展開します。
3. インデックスを右クリックし、ショートカット メニューの [インデックス: プロパティ] をクリックします。
4. このダイアログ ボックスでは並べ替え順序を変更できます。

   その他のインデックス プロパティを変更するには、既存のインデックスを新しいインデックスで置き換える必要があります。

SQL Server Mobile Edition または SQL Server Compact Edition のデータベースからインデックスを削除するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、インデックスを削除するデータ接続を展開します。
3. インデックスが含まれているテーブルを展開して [インデックス] を展開し、削除するインデックスを右クリックします。
4. ショートカット メニューの [インデックスの削除] をクリックします。
5. [オブジェクトの削除] ダイアログ ボックスで、[削除] をクリックし、[OK] をクリックします。

参照

その他の技術情報

New Index
Index Properties (SQL Server Mobile)
デバイス プロジェクトにおけるデータ ソースの管理

方法 : ナビゲーション ボタンを追加する (デバイス)

次の手順を使用して、データ ソース内のさまざまな行を参照するための移動ボタンを作成します。この手法により、.NET Compact Framework でサポートされていない .NET Framework の **DataNavigator** クラスと同等の機能を使用できるようになります。

以下の手順は C# で記述されており、Northwind データベースの Customers テーブルに依存しています。[データ ソース] ウィンドウにはデータセットまたは結果セットが表示されているものとします。詳細については、「方法 : デバイス プロジェクトにデータベースを追加する」を参照してください。実際のプロジェクトでは、このコード サンプルでは示していない範囲チェックも行います。

移動ボタンを追加できるように設定するには

1. テーブルを [データ ソース] ウィンドウからフォーム上にドラッグ アンド ドロップします。
2. ボタンを Windows フォーム上にドラッグ アンド ドロップします。
3. ボタンの Text プロパティを適切に設定します (たとえば "Next")。
4. フォームのボタンをダブルクリックしてコード エディタを開きます。コード エディタには、ボタンの Click イベント ハンドラ部分が表示されます。
5. 次のコード例を使用して、[First] ボタン、[次へ] ボタン、[Previous] ボタン、および [Last] ボタンの各イベント ハンドラをコーディングします。

[First] ボタンをコーディングするには

- 「`this.customersBindingSource.MoveFirst();`」と入力します。

[次へ] ボタンをコーディングするには

- 「`this.customersBindingSource.MoveNext();`」と入力します。

[Previous] ボタンをコーディングするには

- 「`this.customersBindingSource.MovePrevious();`」と入力します。

[Last] ボタンをコーディングするには

- 「`this.customersBindingSource.MoveLast();`」と入力します。

参照

処理手順

方法 : SqlCeResultSet コードを生成する (デバイス)

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法：データベースのデータの変更を永続化する (デバイス)

デバイス プロジェクトでのデータの変更を永続化するには以下の手順を使用します。

以下の手順は C# で記述されており、Northwind データベースの Customers テーブルに依存しています。[データ ソース] ウィンドウにはデータセットが 1 つ表示されている (他の種類のデータ ソースがない) ものと想定しています。詳細については、「方法 : デバイス プロジェクトにデータベースを追加する」を参照してください。

データ変更をデータベースに永続化するには

1. テーブルを [データ ソース] ウィンドウから Windows フォーム上にドラッグします。
2. ボタンを Windows フォーム上にドラッグします。
3. ボタンの [テキスト] プロパティを [保存] に変更します。
4. フォームのボタンをダブルクリックしてコード エディタを開きます。コード エディタには、ボタンの Click イベント ハンドラ部分が表示されます。
5. 次のコードを入力します。

```
this.customersBindingSource.EndEdit();
this.customersTableAdapter.Update(this.northwindDataSet.
   Customers);
```

参照

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法：デザイン時の接続文字列を変更する (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

デザイン時の接続文字列は、既定で、プロジェクトの SQL Server Mobile Edition データ ソースまたは SQL Server Compact Edition データ ソースを作成するときに作成されます。Visual Studio では、この文字列を使用してデザイン時にデータベースに接続し、スキーマ情報を取得します。プロジェクト開発の過程で、この文字列を変更することもできます。

セキュリティに関するメモ：
接続文字列にパスワードを含めることには、セキュリティ上のリスクが伴います。詳細については、「Securing Databases (SQL Server Mobile)」を参照してください。

メモ：
デザイン時の接続文字列は、実行時の接続文字列と区別します。詳細については、「接続文字列プロパティ、ファイル プロパティのダイアログ ボックス (デバイス)」を参照してください。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

デザイン時の接続文字列を変更するには

1. ソリューション エクスプローラで、.xsd ファイルをダブルクリックし、データセット デザイナを開きます。
2. [TableAdapter] を右クリックし、ショートカット メニューの [プロパティ] をクリックします。
3. [接続] プロパティを展開します。
4. [ConnectionString] に新しい値を入力します。

参照

処理手順

方法 : 接続文字列を編集する

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法：実行時の接続文字列を変更する (デバイス)

データ ソースをプロジェクトに追加すると、接続文字列が .xsd ファイルに生成されます。この文字列は、デザイン時に接続文字列と見なされるため、デザイン時に Visual Studio をデータ ソースへ接続するときに適しています。この接続文字列は、既定の実行時の接続文字列としても機能します。

デバイスでアプリケーションを実行するときに異なる接続文字列が必要な場合、次の手順で実行時の文字列を指定できます。詳細については、「接続文字列プロパティ、ファイル プロパティのダイアログ ボックス (デバイス)」を参照してください。

セキュリティに関するメモ：
接続文字列にパスワードを含めることには、セキュリティ上のリスクが伴います。詳細については、「Securing Databases (SQL Server Mobile)」を参照してください。

実行時の接続文字列を変更するには

1. ソリューション エクスプローラで、.xsd ファイルを選択します。
2. [表示] メニューの [プロパティ ウィンドウ] をクリックします。
3. [接続文字列] プロパティに実行時の接続文字列を入力します。

参照

処理手順

方法 : デザイン時の接続文字列を変更する (デバイス)

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法 : ビジネス オブジェクトをデータ ソースとして追加する (デバイス)

ビジネス オブジェクトをスマート デバイス プロジェクトのデータ ソースとして追加できます。詳細については、「Visual Database Tools」を参照してください。

オブジェクトから新しいデータ ソースを作成するには

1. [データ] メニューの [新しいデータ ソースの追加] をクリックします。
2. [データソースの種類を選択] ページで、[オブジェクト] を選択します。
3. [バインド先のオブジェクトを選択します。] ページで、既にアプリケーションに含まれているオブジェクトを選択します。

   メモ :

   ウィザードにオブジェクトが表示される前に、そのオブジェクトを含むプロジェクトをビルドする必要がある場合があります。現在のアプリケーションにないオブジェクトへの参照を追加することもできます。そのためには、[参照の追加] をクリックし、[参照の追加] ダイアログ ボックスで目的のアセンブリを探します。アセンブリがツリー ビューに追加されます。

4. バインドするオブジェクトを含むアセンブリを展開し、ツリー ビューでオブジェクトを選択します。
5. [完了] をクリックします。

   データ ソースが [データ ソース] ウィンドウに追加されます。

データ ソースをフォームに追加するには

1. [データ] メニューの [データ ソースの表示] をクリックして [データ ソース] ウィンドウを開きます。
2. [データ ソース] ウィンドウでアイテムを選択し、選択したアイテムを Windows フォーム上にドラッグして、オブジェクト内のプロパティにバインドされるコントロールを作成します。詳細については、「データの表示の概要」を参照してください。

参照

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法：Web サービスをデータ ソースとして追加する (デバイス)

スマート デバイス プロジェクトのデータ ソースとして Web サービスを追加できます。

アプリケーションを Web サービスに接続するには

1. [データ] メニューの [新しいデータ ソースの追加] をクリックします。
2. [データ ソースの種類を選択] ページで、[Web サービス] を選択し、[次へ] をクリックします。
3. [Web 参照の追加] ダイアログ ボックスを使用して、必要な Web サービスへの参照を追加します。
4. [完了] をクリックします。

   データ ソースが [データ ソース] ウィンドウに追加されます。

データ ソースをフォームに追加するには

- [データ ソース] ウィンドウでアイテムを選択して Windows フォーム上にドラッグし、バインド コントロールを作成します。詳細については、「データの表示の概要」を参照してください。

参照

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

方法：SQL Server データベースをデータ ソースとして追加する (デバイス)

マネージ デバイス プロジェクトで、SQL Server データベースをデータ ソースとして使用できます。

SQL Server のデータ接続をサーバー エクスプローラに追加するには

1. [表示] メニューの [サーバー エクスプローラ] をクリックします。
2. [サーバー エクスプローラ] ウィンドウで、[データ接続] を右クリックし、ショートカット メニューの [接続の追加] をクリックします。
3. [接続の追加] ダイアログ ボックスで、[変更] をクリックします。
4. [データ ソースの変更] ダイアログ ボックスで、[Microsoft SQL Server] を選択し、[OK] をクリックして [接続の追加] ダイアログ ボックスを再び開きます。
5. [サーバー名] ボックスで、データ ソースが格納されているサーバーの名前を指定します。
6. サーバーにログオンします。

   サーバーが SQL Server 認証を使用している場合は、ユーザー名とパスワードを指定する必要があります。
7. [データベースの選択または入力] ボックスで、データベース名を選択または入力し、[OK] をクリックします。

   新しいデータ接続がサーバー エクスプローラに表示されます。

SQL Server データ接続をプロジェクト内のデータ ソースとして使用するには

1. 必要条件 : Visual Studio IDE 内で .NET Compact Framework スマート デバイス プロジェクトを既に開いている必要があります。
2. [データ] メニューで、[新しいデータ ソースの追加] をクリックし、データ ソース構成ウィザードを起動します。
3. [データソースの種類を選択] ページで、[データベース] を選択し、[次へ] をクリックします。
4. [データ接続の選択] ページの [新しい接続] をクリックします。
5. [接続の追加] ダイアログ ボックスで、[変更] をクリックします。
6. [データ ソースの変更] ダイアログ ボックスで、[Microsoft SQL Server] を選択し、[OK] をクリックして [接続の追加] ダイアログ ボックスを再び開きます。
7. [サーバー名] ボックスで、データ ソースが格納されているサーバーの名前を入力または選択します。
8. サーバーにログオンします。

   サーバーが SQL Server 認証を使用している場合は、ユーザー名とパスワードを指定する必要があります。
9. [データベースの選択または入力] ボックスで、データベース名を選択または入力し、[OK] をクリックします。
10. [データ接続の選択] ページで、重要情報を接続文字列から除外するオプションを選択し、[次へ] をクリックします。

セキュリティに関するメモ：
重要情報を接続文字列に含めると、セキュリティ上のリスクが伴います。

11. [データベース オブジェクトの選択] ページで、データ ソースとして使用するオブジェクトを選択し、[完了] をクリックします。

    [データ ソース] ウィンドウに、データ接続がデータ ソースとして表示されます。

参照

その他の技術情報

デバイス プロジェクトにおけるデータ ソースの管理

チュートリアル : 簡単なアプリケーションの作成

このチュートリアルでは、Visual Studio 2005 を使用して簡単なスマート デバイス プロジェクトを作成する方法を示します。その際、Visual C# または Visual Basic を使用します。また、アプリケーションを PC のエミュレータで、または、PC と接続した実際のデバイス上で実行する方法も示します。

メモ :
完成したアプリケーションを実際のデバイスに永続的にインストールするには、まず、そのアプリケーションをパッケージ化して CAB ファイルにする必要があります。詳細については、「チュートリアル : 配置用のスマート デバイス ソリューションのパッケージ化」を参照してください。

メモ :
ダイアログ ボックスで使用できるオプションとメニュー コマンドは、開発時の設定によって異なります。このチュートリアルでは、Visual Basic 開発設定と Visual C# 開発設定を使用しています。設定を参照したり変更したりするには、[ツール] メニューの [設定のインポートとエクスポート] を選択し、[IDE 設定のリセット] をクリックします。次に、目的の設定を選択して [設定のリセット] をクリックします。

このチュートリアルは、次の 4 つの部分で構成されます。

- 対象デバイスの選択。
- デバイス プロジェクトの作成。
- ボタンとイベント ハンドラのフォームへの追加。
- 対象デバイス上でのアプリケーションの実行。

対象デバイスの選択

このチュートリアルは、エミュレータと物理デバイスのどちらか 1 つを対象に選択して実行できます。チュートリアルの途中で切り替えることもできます。ソリューションを配置するときに毎回デバイス選択のダイアログが表示されるようにするには、次の手順を実行します。

配置時にデバイス選択のダイアログを表示するには

1. [ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックします。次に、[全般] をクリックします。[デバイス ツール] が表示されない場合は、[オプション] ダイアログ ボックスの下部にある [すべての設定を表示] を選択します。
2. [デバイス プロジェクトの配置前に選択できるデバイスを表示] チェック ボックスをオンにします。

デバイス プロジェクトの作成

このセクションでは、簡単な Windows フォーム アプリケーションをビルドします。以下の手順では、Windows フォーム プロジェクトを作成して、フォームにコントロールを追加し、そのコントロールにイベント処理を追加します。最後にアプリケーションのビルドとテストを行います。C++ プロジェクトの作成手順は示しません。C++ で開発する場合は、基本的な Hello World アプリケーションを作成し、リリース バージョンをビルドしてから、「アプリケーションをビルド、テスト、および保存するには」セクションに進んでください。

デバイス プロジェクトを作成するには

1. (Visual Basic) Visual Studio の [ファイル] メニューの [新しいプロジェクト] をクリックします。

   または

   (Visual C#) Visual Studio の [ファイル] メニューの [新規作成] をポイントして、[プロジェクト] をクリックします。

2. [新しいプロジェクト] ダイアログ ボックスの [プロジェクトの種類] で、[Visual Basic] または [Visual C#] を展開し、[スマート デバイス] を展開します。次に、[Pocket PC 2003] をクリックします。

   目的の言語が最初に表示されない場合は、[他の言語] を展開します。この表示は、ユーザーのプロファイル設定で管理されます。

3. [テンプレート] の [デバイス アプリケーション] をクリックします。
4. [プロジェクト名] ボックスに「**AppForDeployment**」と入力します。
5. (Visual C# のみ) [場所] ボックスで、プロジェクト ファイルを格納する場所を確認します。
6. [OK] をクリックします。

Windows フォーム デザイナに Pocket PC デバイスが表示されます。

コントロールをフォームに追加するには

1. ツールボックスから、フォームに Button コントロールをドラッグします。

   ツールボックスが表示されていない場合は、[表示] メニューの [ツールボックス] をクリックします。

   ツールボックスに [デバイス コントロール] タブが表示されていない場合は、ツールボックスを右クリックして [すべて表示] を選択するか、[ツールボックスのリセット] を選択します。

2. Button コントロールを右クリックし、[プロパティ] をクリックします。

3. [プロパティ] ウィンドウに「**Say Hello**」と入力し、Enter キーを押して [Text] プロパティを設定します。

Button コントロールのイベント処理を追加するには

1. フォーム上のボタンをダブルクリックします。

   コード エディタが開き、イベント ハンドラにカーソルが表示されます。

2. 次の Visual Basic コードを入力します。

   ```
   MessageBox.Show("Hello, World!")
   ```

   または

   次の C# コードを入力します。

   ```
   MessageBox.Show("Hello, World!");
   ```

アプリケーションをビルド、テスト、および保存するには

1. [デバッグ] メニューの [開始] をクリックします。

2. [配置] ダイアログ ボックスで、[Pocket PC 2003 SE エミュレータ] を選択し、[配置] をクリックします。

   進行状況メッセージが、[出力] ウィンドウ、ステータス バー、およびエミュレータ画面に表示されます。

3. アプリケーションをエミュレータで実行しているときに、[Say Hello] ボタンをタップして、"Hello, World!" と表示されることを確認します。

4. このチュートリアルの次のセクションに進む前に、エミュレータで、[ファイル] メニューの [終了] をクリックして、エミュレータを閉じます。

5. [デバイス エミュレータ] ダイアログ ボックスで、[終了する前にエミュレータの状態を保存しますか?] プロンプトに対して [いいえ] をクリックします。

6. Visual Studio で、[ファイル] メニューの [すべてを保存] をクリックします。

   エミュレータがまだ完全にシャットダウンしていない場合は、"デバッグを中止しますか?" というメッセージが表示されます。[はい] をクリックします。

7. [プロジェクトの保存] ダイアログ ボックスで、任意の場所を選択して AppForDeployment プロジェクトを保存します。

参照

その他の技術情報

.NET Compact Framework を使用したデバイスのプログラミング

Visual C++ を使用したデバイスのプログラミング

Visual Studio 2005 では、Visual C++ を使用したデバイス アプリケーションの開発をサポートしています。

このセクションの内容

Visual C++ デバイス アプリケーション開発の新機能

Visual C++ for Devices の新機能について説明します。

Visual C++ デバイス プロジェクトの作成と移植

新しい Visual C++ デバイス プロジェクトを作成する方法、および既存の Visual C++ デスクトップ プロジェクトと eMbedded Visual C++ 4.0 プロジェクトを Visual C++ デバイス プロジェクトに移植する方法について説明します。

Visual C++ デバイス プロジェクトの開発

Visual Studio 2005 の機能を使用して、Visual C++ でデバイス アプリケーションを開発する方法について説明します。

Visual C++ デバイス プロジェクトのビルドとデバッグ

Visual C++ デバイス プロジェクトをビルドおよびデバッグする方法について説明します。

参照

その他の技術情報

スマート デバイス開発

Mobile Developer Center

Visual C++ デバイス アプリケーション開発の新機能

Visual Studio 2005 では、Visual C++ を使用してデバイスを開発できます。以前は、Visual C++ を使用してデバイス プロジェクトを開発する場合に、eMbedded Visual C++ のバージョンが必要でした。次のセクションでは、eMbedded Visual C++ から強化された新機能について詳しく説明します。

統合開発環境 (IDE: Integrated Development Environment)

対象となる複数のオペレーティング システム

Visual Studio 2005 ツールセットにより、Pocket PC 2003、Smartphone 2003、Windows CE 5.0 ベースのカスタム SDK、および今後リリースされる SDK を対象にすることができます。

プロジェクト システム

プロジェクト システムでは、プラットフォームがそのプラットフォームでサポートされている CPU アーキテクチャに関連付けられています。以前のバージョンの eMbedded Visual C++ では、現在アクティブなプロジェクトでサポートされていない CPU アーキテクチャを選択できました。

IntelliSense

Visual Studio 2005 では、IntelliSense がソフトウェア開発キット (SDK: Software Development Kit) のヘッダー ファイルに反映されており、対象となるプラットフォームに関する正確な情報が提供されます。

混合モードのソリューション

Visual Studio 2005 では、デバイス開発者は、マネージ コード (Visual C# または Visual Basic) とアンマネージ コード (Visual C++) の両方を同じソリューション内で使用できます。また、同じプロジェクト内にデスクトップとデバイス両方の Visual C++ コードを含めることもできます。

アプリケーションのインストール

開発者は、デバイス アプリケーションのパッケージ化およびデスクトップ インストーラの作成を行い、そのアプリケーションをデバイスに配布できます。

カスタム アプリケーションとカスタム クラス ウィザード

Visual Studio 2005 を使用すると、デバイス プロジェクト用のアプリケーションおよびクラス ウィザードを容易に作成できます。

リソース エディタ

リソース エディタには、入力パネルの状態コントロールや CAPEdit コントロールなど、デバイス固有のリソースが含まれています。

デバッグ

Visual Studio 2005 C++ のデバイス デバッガの新機能または強化された機能を次に示します。

- 以前のデバイス アプリケーション デバッガより強化された保全性およびパフォーマンス。
- プロセスにデタッチおよび再アタッチする機能。
- 実行可能ファイルを使用せずにプロセスにアタッチする機能。
- 共有 DLL 内のブレークポイントのサポート。
- 複数のプロセスのデバッグ。
- 改善された式の評価 (多様性のあるデバッガ ウィンドウなど)。

コンパイラ

Visual Studio 2005 C++ のデバイス コンパイラの新機能または強化された機能を次に示します。

- /LTCG (リンク時のコード生成) のサポート。
- コンパイラの制限の強化。
- /GS スイッチによるバッファ オーバーラン検出のサポート。
- __restrict キーワードのサポート。
- より多くの C++ 標準に準拠するための更新。

ライブラリ

デバイス用の Visual C++ ライブラリの新機能または強化された機能を次に示します。

- Microsoft Active Template Library (ATL) for Devices と Microsoft Foundation Classes (MFC) for Devices が 8.0 コード ベースに更新されました。
- MFC for Devices、ATL for Devices、デバイス用の C ランタイム (CRT) ライブラリ、およびデバイス用の標準 C++ ライブラリでは、セキュリティ、パフォーマンス、安定性、および標準への準拠が強化されています。
- ATL と MFC の共通機能が統合されました。
- デバイス用の標準 C++ ライブラリに、ストリームのサポートが追加されました。

ATL

ATL for Devices に追加された機能を次に示します。

- ActiveX コントロールのホスト。
- Web サービスを利用する機能。
- **CImage** クラスによるビットマップのサポート。
- 配列、リスト、およびツリーを管理するための新しいクラス。
- 拡張された文字列の操作と変換。
- IPv6 に対する拡張されたソケット サポート。

MFC

MFC for Devices に追加された機能を次に示します。

- Smartphone の利用。
- ActiveX コントロールのホストおよび作成。

参照

その他の技術情報

Visual C++ を使用したデバイスのプログラミング

Visual C++ デバイス プロジェクトの作成と移植

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

Visual Studio 2005 には、C++ デバイス プロジェクトを開始する多くの方法が用意されています。このセクションには、新しい Visual C++ デバイス プロジェクトを作成し、既存のプロジェクトが Visual Studio 2005 環境内のデバイスをターゲットとできるようにする方法についてのトピックが含まれています。

次の表は、Visual Studio 2005 SP1 に合わせて更新されています。

このセクションの内容

方法 : 新しい Visual C++ デバイス プロジェクトを作成する

新しい ATL、MFC、およびネイティブ Windows CE アプリケーション プロジェクトを作成する方法について説明します。

eMbedded Visual C++ から Visual Studio 2005 へのアップグレード ウィザード

eMbedded Visual C++ 3.0 および 4.0 で記述されたプロジェクトを、アップグレード ウィザードを使用して Visual Studio 2005 にインポートする方法について説明します。

eVC からの移植に関する既知の問題

eMbedded Visual C++ 3.0 および 4.0 で記述されたプロジェクトを Visual Studio 2005 に移行するときの注意点について説明します。

デスクトップ プロジェクトのデバイス サポート

デスクトップの C++ プロジェクトをセットアップして、デバイスをターゲットとする方法について説明します。

ネイティブ デバイス プロジェクトで対象となる複数のプラットフォーム

1 つのプロジェクトから複数のプラットフォームをターゲットとする 2 つの方法について説明します。

ネイティブ デバイス プロジェクトでのリソース編集

デバイスのリソース エディタと、デスクトップの対応する部分の間にある共通点について説明します。

ネイティブ デバイス プロジェクト作成後の制限事項

プロジェクトの作成後に別のプラットフォームを追加した場合の制限事項について説明します。

方法 : ネイティブ デバイス プロジェクトにデータベースを追加する

Visual C++ を使用して SQL Server Mobile または SQL Server Compact Edition デバイス アプリケーションを開発する方法について説明します。

関連するセクション

Visual C++ を使用したデバイスのプログラミング

方法 : 新しい Visual C++ デバイス プロジェクトを作成する

eVC からの移植に関する既知の問題

Visual C++ デバイス プロジェクトの開発

Visual C++ デバイス プロジェクトのビルドとデバッグ

方法：新しい Visual C++ デバイス プロジェクトを作成する

デバイス プロジェクトの作成とデスクトップ プロジェクトの作成との間には、いくつかの大きな違いがあります。

- デスクトップ プロジェクト テンプレートのサブセットが、デバイスでサポートされます。次の手順を実行すると、これらのテンプレートを参照できます。
- 作成を選択したプロジェクトのアプリケーション ウィザードでは、ウィザードの [プラットフォーム] ページに示される一覧から、ターゲットとするプラットフォームを 1 つ以上選択する必要があります。このプラットフォーム一覧は、現在インストールされている SDK に基づいて生成されます。

> **メモ：**
>
> `CE_ALLOW_SINGLE_THREADED_OBJECTS_IN`、define `_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA` の定義に関する警告がコンパイラにより発行される場合、stdafx.h でフラグを定義します。これを行うには、stdafx.h ヘッダー ファイルを開き、ファイルの上部付近に `#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA` を追加します。

Visual C++ デバイス プロジェクトを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [プロジェクトの種類] ペインで [Visual C++] を展開し、[スマート デバイス] をクリックします。[テンプレート] ペインには、デバイスでサポートされるプロジェクト テンプレートがすべて表示されます。
3. 使用するプロジェクト テンプレートをクリックします。
4. [プロジェクト名] ボックスで、プロジェクトの名前を入力し、[OK] をクリックします。
5. アプリケーション ウィザードを使用して、プロジェクトの作成を完了します。

参照

処理手順

アプリケーション ウィザードを使用したプロジェクトの作成

その他の技術情報

Visual C++ デバイス プロジェクトの作成と移植

eMbedded Visual C++ から Visual Studio 2005 へのアップグレード ウィザード

Visual Studio 2005 には、eMbedded Visual C++ 3.0 および eMbedded Visual C++ 4.0 のプロジェクトを Visual Studio 2005 に移行するためのアップグレード ウィザードがあります。

このアップグレード ウィザードの機能は次のとおりです。

- eMbedded VC++ から移行したソース コード、ヘッダー、リソースを含む Visual Studio 2005 ソリューションとプロジェクトの作成
- 移行する MFC プロジェクト用の MFC dll の配置リストへの追加
- プロジェクト設定 (コンパイラのスイッチなど) の移行
- eVC ではサポートされるが Visual Studio 2005 ではサポートされないアーキテクチャを、Visual Studio 2005 でサポートされるアーキテクチャに変換

eVC から Visual Studio 2005 へのアップグレード ウィザードの使用方法

アップグレード ウィザードを使用して eVC プロジェクトを Visual Studio 2005 に移行するには

1. [ファイル] メニューの [開く] をクリックし、[プロジェクト/ソリューション] をクリックします。
2. eVC プロジェクトが保存されているディレクトリを表示します。eVC のワークスペースに 1 つのプロジェクトしかない場合は、.vcw ファイルまたは .vcp ファイルのどちらかを選択できます。eVC ワークスペースに複数のプロジェクトがあり、そのすべてを移行する場合は、.vcw ファイルを選択します。
3. [OK] をクリックします。

メモ :
移行ウィザードは、その場での移行プロセスを実行します。たとえば、ソース コードのコピーを作成するのではなく、Visual Studio 2005 プロジェクトのみを作成します。移行によって作成された Visual Studio 2005 プロジェクトには、元の eVC プロジェクトに含まれていたものと同じソース ファイルが含まれます。

アーキテクチャの変換

eMbedded Visual C++ でサポートされていたデバイス アーキテクチャの一部は、Visual Studio 2005 ではサポートされなくなりました。これは、Visual Studio 2005 で対象とする新しいプラットフォームで、新しいアーキテクチャがサポートされているためです。ただし、古いアーキテクチャはすべて新しいデバイス アーキテクチャに変換できます。アップグレード ウィザードでは、この変換が自動的に実行されます。次の表に、eMbedded Visual C++ でサポートされるデバイス アーキテクチャと Visual Studio 2005 でサポートされるデバイス アーキテクチャを示します。

eVC アーキテクチャ	互換性のある Visual Studio 2005 のアーキテクチャ
ARM	ARMv4
ARMv4	ARMv4
ARMv4i	ARMv4i
ARMv4T	ARMv4i
MIPS	MIPSII
Mips16	MIPSII
MipsII	MipsII
MipsII_fp	MipsII_fp

MipsIV	MipsIV
MipsIV_fp	MipsIV_fp
SH3	SH4
SH4	SH4
エミュレータ	X86
X86	X86

このウィザードを使用して eVC プロジェクトをアップグレードしたときは、Visual Studio 2005 で作成される新しいプロジェクトは、新しいプロジェクトのアーキテクチャをサポートするすべてのインストール済み SDK を対象とします。移行されたアーキテクチャでは、eVC アーキテクチャのいずれかの設定が継承されます。次の表に、eMbedded Visual C++ でサポートされるデバイス アーキテクチャと Visual Studio 2005 でサポートされるデバイス アーキテクチャの対応付けを示します。

元のアーキテクチャ	変換先	メモ
ARM/ARMV4/ARMV4I 以外	「アーキテクチャの変換」の表を参照してください	
ARM で ARMV4i 以外	ARMV4 と ARMV4i	ARMV4i の基本設定は eVC の ARM の基本設定から継承されます。
ARMV4 で ARMV4i 以外	ARMV4 と ARMV4i	ARMV4i の基本設定は eVC の ARMV4 の基本設定から継承されます。
ARM/ARMV4 と ARMV4i	ARMV4 と ARMV4i	ARMV4i の基本設定は eVC の ARMV4i の基本設定から継承されます。

eMbedded Visual C++ バージョン 4.0 では、ダイアログ ボックスのスタイルが既定で MFC Pocket PC アプリケーションの `DS_MODALFRAME` に設定されます。MFC 8.0 では、このスタイルはサポートされません。

メモ：
"利用できるプラットフォームで、このプロジェクト ファイルの元のプラットフォームと一致するものはありません" というエラー メッセージが表示された場合は、元のプロジェクトの構成時に使用した SDK と互換性のあるバージョンの SDK をインストールする必要があります。

参照

関連項目

Windows Mobile Platform Migration FAQ for Developers
Migrating Microsoft eMbedded Visual C++ Projects to Visual Studio 2005
Step by Step: Migrating an eMbedded Visual C++ Application to Visual Studio 2005

概念

eVC からの移植に関する既知の問題

eVC からの移植に関する既知の問題

既存の eMbedded Visual C++ プロジェクトを Visual Studio 2005 に変換するのに役立つ、多くの C++ ツールおよびリソースが用意されています。詳細については、「eMbedded Visual C++ から Visual Studio 2005 へのアップグレード ウィザード」を参照してください。

ATL (Active Template Library)、MFC (Microsoft Foundation Classes)、および標準 C++ ライブラリは、eVC から更新および変更されました。いくつかのクラスはサポートされなくなりました。「MFC 8.0 でサポートされなくなった MFC 3.0 の eVC クラスの一覧」を参照してください。それらのクラスを呼び出すコードは、変更して Visual Studio 2005 でコンパイルする必要があります。eVC から移植を行う際に発生する一般的な問題を次に示します。

問題	説明/代替手段
CCeSocket::OnReceive() メソッドは、CE 3.0 より新しいデバイスの MFC では呼び出されません。	解決策の詳細については、http://support.microsoft.com/default.aspx?scid=kb;ja-jp;253945 で説明されています。
CArchive クラス クラスはサポートされていません。	多くの eVC プロジェクトには、**CArchive Class** クラスへの参照が含まれています。これを回避するには、**CArchive** への参照を削除する必要があります。

CObArray や CMapPtrToPtr など、特定のコレクション クラスは、対応するテンプレート クラス (CArray<>、CMap<> など) を使って CE 5.0 に実装されています。eMbedded Visual C++ バージョン 4.0 およびデスクトップ C++ ライブラリでは、これらの型が通常のクラス (非テンプレート クラス) として実装されています。そのため、これらのテンプレート クラスで IMPLEMENT_SERIAL を呼び出すと、次のようなコンパイル エラーが発生します。
 	
エラー C2039: 'classCObArray' : 'CArray<TYPE,ARG_TYPE>' のメンバではありません
 	
エラー C2065: 'classCObArray' : 定義されていない識別子です
	こうした実装の違いから生じる問題を回避するには、CObArray や CMapPtrToPtr などではなく、CObject を使用するように IMPLEMENT_SERIAL マクロを変更します。
	
	たとえば、次のように記述するのは誤りです。
	
	`IMPLEMENT_SERIAL(CYourClass, CObArray, 0)`
	
	これを次のように書き換えます。
	
	`IMPLEMENT_SERIAL(CYourClass, CObject, 0)`

eMbedded Visual C++ バージョン 4.0 では、ダイアログ スタイルが既定で MFC Pocket PC アプリケーションの **DS_MODALFRAME** に設定されます。MFC 8.0 では、このスタイルはサポートされません。

サンプル

このセクションでは、プロジェクトを eMbedded Visual C++ から Visual Studio 2005 に移行するときに起こり得る一般的なエラーの概要を示します。詳細については、「Migrating Microsoft eMbedded Visual C++ Projects to Visual Studio 2005」を参照してください。

- コンパイル エラー: インクルード ファイル 'wceres.rc' を開けません

  プロジェクト リソース (RC) ファイルを右クリックして [コードの表示] を選択し、次の行をコメント アウトします。

  ```
  //#include "wceres.rc"
  ```

- NUM_TOOL_TIP が定義されていません

  ヘッダー ファイルで、Pocket PC 構成に `#define _WIN32_WCE_PSPC` を定義し、Smartphone 構成に `_WIN32_WCE_WFSP` を定義します。

- ファイル 'OLDNAMES.lib' を開けません

  ソリューション エクスプローラで、プロジェクト ファイルを右クリックし、[プロパティ] をクリックします。

  [リンカ] をクリックします。**OLDNAMES.LIB** を追加することにより、[特定のライブラリの無視] プロパティを編集します。

- あいまいなオーバーロード

標準 C++ ライブラリ (SCL) および ATL に、デバイス SDK にも存在する API があります。:: など、名前空間を明確にしてください。

- モジュールのコンピュータの種類 'THUMB' は対象コンピュータの種類 'ARM' と競合しています。

  ソリューション エクスプローラで、プロジェクト ファイルを右クリックし、[プロパティ] を選択します。

  [構成プロパティ] の下で、[リンカ] を展開し、[コマンド ライン] プロパティをクリックします。[プロパティ] ページの各 Windows Mobile 5.0 構成のコマンド ラインから **/MACHINE:THUMB** スイッチを削除します。

- リソース文字列が正しく区切られていません

  移植されるアプリケーションからのリソース文字列が正しく区切られていない場合に、問題が発生することがあります。ソリューション エクスプローラで、プロジェクト ファイルを右クリックし、[プロパティ] をクリックします。[構成プロパティ] の下で、[リソース] を展開し、[コマンド ライン] プロパティを選択します。リソース コンパイラのコマンド ラインに **-n** スイッチを追加します。

- ダイアログ内に BEGIN が予期されるエラー

  このエラーは、通常 "ファイル '0x1' が見つかりません。" など、ファイルが見つからないエラーの後に続きます。エラーに示されている RC ファイルを開き、次のコード例に示すようにコードを編集して、**FONT** 宣言の前後で `#ifdef` ステートメントを使用します。

  元のコード

  ```
  IDD_COMPTEST DIALOGEX 0, 0, 186, 95
  STYLE DS_SETFONT | WS_CHILD
  EXSTYLE WS_EX_CONTROLPARENT
  FONT 8, "MS Sans Serif", 0, 0, 0x1
  BEGIN
  END
  ```

  変更後のコード

  ```
  IDD_COMPTEST DIALOGEX 0, 0, 186, 95
  STYLE DS_SETFONT | WS_CHILD
  EXSTYLE WS_EX_CONTROLPARENT
  #ifdef _WIN32_WCE
  FONT 8, "MS Sans Serif"
  #else
  FONT 8, "MS Sans Serif", 0, 0, 0x1
  #endif
  BEGIN
  END
  ```

参照

概念

eMbedded Visual C++ から Visual Studio 2005 へのアップグレード ウィザード

その他の技術情報

Windows Mobile Platform Migration FAQ for Developers

デスクトップ プロジェクトのデバイス サポート

Visual Studio 2005 では、デスクトップや、Pocket PC および Smartphone デバイスなどのスマート デバイス プラットフォームを含む、複数のプラットフォームをターゲットとするプロジェクトの使用がサポートされます。デバイス プロジェクトの C++ クラス ウィザードを使用すると、デスクトップ プロジェクトと同じソリューションでデバイス プロジェクトを作成できます。詳細については、「デバイス プロジェクトの C++ クラス ウィザード」を参照してください。その後、次のリソースを使用して、新しく作成されたデバイス プロジェクトにデスクトップ アプリケーションを移植できます。

- デバイス プロパティでコード ウィザードを使用する
- 方法 : 新しい Visual C++ デバイス プロジェクトを作成する
- ネイティブ デバイス プロジェクトで対象となる複数のプラットフォーム
- ネイティブ デバイス プロジェクトでのリソース編集
- Visual C++ デバイス プロジェクトのビルドとデバッグ
- デバイス プロジェクトのリソース エディタ
- デバイス プロジェクトのデバッグ
- 配置用のデバイス ソリューションのパッケージ化
- 複数のプラットフォームでのリソースの使用.

参照

関連項目

デバイス対応の Visual Basic の言語リファレンス

概念

デバイス プロジェクト用 .NET Compact Framework リファレンス
デバイス用の標準 C++ ライブラリ リファレンス
デバイスの C ランタイム ライブラリ リファレンス

その他の技術情報

デバイス用の ATL リファレンス
デバイス用の MFC リファレンス

ネイティブ デバイス プロジェクトで対象となる複数のプラットフォーム

Visual Studio 2005 を使用して、Pocket PC 2003 や Smartphone 2003 などの複数のデバイス プラットフォームを対象とする 1 つのデバイス プロジェクトを作成できます。

デバイス プロジェクトで複数のプラットフォームを対象にするには、2 つの方法があります。

- プロジェクト作成時にアプリケーション ウィザードを使用する方法。この方法が簡単です。詳細については、「方法 : ウィザードを使用してマルチプラットフォーム対応のデバイス プロジェクトを作成する」を参照してください。
- プロジェクトの作成後に追加する方法。詳細については、「方法 : 新しいプラットフォームをデバイス プロジェクトに追加する」を参照してください。

プロジェクトのアプリケーション ウィザードの [プラットフォーム] ページで複数のプラットフォームを選択すると、プラットフォームごとにリソース ファイルが生成および設定されます。一方、プロジェクトの作成後にプラットフォームを追加すると、手動でプラットフォームおよびリソース ファイルを追加する必要があります。詳細については、「複数のプラットフォームでのリソースの使用」を参照してください。

参照

処理手順

アプリケーション ウィザードを使用したプロジェクトの作成

概念

Windows CE の MFC ウィザードの C++

その他の技術情報

Visual C++ デバイス プロジェクトの作成と移植

ネイティブ デバイス プロジェクトでのリソース編集

Visual Studio 2005 の複数プラットフォームの性質のため、Pocket PC および Smartphone など、ターゲットとして選択したプラットフォームごとに別個のリソース ファイルが生成されます。デバイス プロジェクトのリソース エディタは、デスクトップ プロジェクトのリソース エディタとほとんど同じです。すべてのエディタがサポートされますが、ダイアログ エディタのみ大きく変更されました。詳細については、「デバイス プロジェクトのリソース エディタの使用」および「リソース エディタ」を参照してください。

プロジェクトの作成時に、すべてのプラットフォームをターゲットとすることができます。これにより、デバイス フォームの要因に合わせてアプリケーションの UI をカスタマイズするのが容易になり、複数プラットフォームのアプリケーションを 1 つのプロジェクトで保持することが可能になります。「チュートリアル : スマート デバイス用マルチプラットフォーム MFC アプリケーションの作成」で作成されたサンプル プロジェクトには、Pocket PC のリソース ファイルと Smartphone リソース ファイルの両方が含まれています。プロジェクトのアクティブな現在の構成は Pocket PC 2003 (Armv4) であるため、Smartphone のリソース ファイルには [ビルドなし] アイコンが表示されている点に注意してください。Smartphone 2003 (ArmV4) のアクティブな構成は、[構成マネージャ] (スマート デバイス プロジェクト ウィザード) を使用して変更できます。これにより、Smartphone 2003 のリソース ファイルから [ビルドなし] アイコンが削除され、代わりに Pocket PC 2003 (Armv4) のリソース ファイルにアイコンが表示されます。

参照

その他の技術情報

Visual C++ デバイス プロジェクトの作成と移植

ネイティブ デバイス プロジェクト作成後の制限事項

このセクションでは、スマート デバイス用のネイティブ C++ アプリケーションに適用される制限事項について、デスクトップ用のネイティブ アプリケーションの場合と比較しながら説明します。

デスクトップ プロジェクトとデバイス ネイティブ プロジェクトの違い

C++ デバイス プロジェクトの作成の詳細については、「デバイス プロジェクトの C++ クラス ウィザード」を参照してください。

- ヘルプ トピックおよびリファレンスのフィルタ処理については、「方法 : スマート デバイス開発用のヘルプを最適化する」を参照してください。
- プロジェクト間のユーザー インターフェイスの違いについては、「デバイス用のユーザー インターフェイス リファレンス」を参照してください。
- ATL デバイス プロジェクトの開発については、「ATL for Devices と標準 ATL の違い」を参照してください。
- MFC デバイス プロジェクトの開発については、「MFC C++ for Devices と標準 MFC の違い」および「デバイス クラスの一意な MFC」を参照してください。
- C++ デバイス開発については、「デバイス用の標準 C++ ライブラリ リファレンス」を参照してください。
- デバイス プロジェクトでのリソースの使用については、「複数のプラットフォームでのリソースの使用」を参照してください。
- コンパイラ スイッチについては、「スマート デバイスのコンパイラ」を参照してください。
- デバイス プロジェクトのデバッグについては、「デバイス プロジェクトのデバッグ」を参照してください。
- デバイス プロジェクトのパッケージ化については、「配置用のデバイス ソリューションのパッケージ化」を参照してください。

参照

処理手順

方法 : デバイスでサポートされる ATL クラスおよびメソッドのヘルプを検索する

方法 : デバイスでサポートされる MFC クラスおよびメソッドのヘルプを検索する

概念

eMbedded Visual C++ から Visual Studio 2005 へのアップグレード ウィザード

その他の技術情報

Visual C++ デバイス プロジェクトの作成と移植

方法：ネイティブ デバイス プロジェクトにデータベースを追加する

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

Visual Studio 開発環境では、SQL Server Mobile Edition または SQL Server Compact Edition を使用したスマート デバイス アプリケーションを開発できます。

Visual C++ for Devices を使用して SQL Server Mobile Edition アプリケーションまたは SQL Server Compact Edition アプリケーションを開発する方法については、「Installing a Development Environment」を参照してください。

SQL Server Mobile Edition または SQL Server Compact Edition をネイティブ プロジェクトで使用している場合、プロジェクトを配置しても、インストール用の CAB ファイルは自動的にはデバイスにダウンロードされません。ただし、CAB ファイルをプロジェクトに追加できます。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

SQL Server Mobile Edition または SQL Server Compact Edition の CAB ファイルをプロジェクトに追加するには

1. プロジェクトを右クリックし、[プロジェクト] ショートカット メニューの [既存項目の追加] をクリックします。
2. CAB ファイルに移動します。

   既定では、CAB ファイルが %Program Files%\Microsoft Visual Studio 8\SmartDevices\SDK\SQL Server\Mobile\v3.0\wce500\<device processor>\ に格納されます。

   または
3. CAB ファイルをスマート デバイスにコピーします。

   既定では、CAB ファイルが %Program Files%\Microsoft Visual Studio 8\SmartDevices\SDK\SQL Server\Mobile\v3.0\wce500\<device processor>\ に格納されます。
4. デバイス上で CAB ファイルを実行して、SQL Server Mobile Edition または SQL Server Compact Edition をインストールします。

参照

その他の技術情報

Visual C++ デバイス プロジェクトの作成と移植

Visual C++ デバイス プロジェクトの開発

ここでは、デバイス アプリケーションを開発するときの独自の側面について説明します。

このセクションの内容

デバイス プロジェクトのプラットフォーム切り替え

同じプロジェクトを使用するターゲットの切り替え方法について説明します。

デバイス プロジェクトの C++ クラス ウィザード

デバイス プロジェクトの C++ ウィザードを使用する方法について説明しています。

デバイス プロジェクトのリソース エディタ

デバイス プロジェクトのリソース エディタについて説明しています。

スマート デバイスのコンパイラ

スマート デバイスで使用されるマイクロプロセッサを対象としたコンパイラについて説明します。

コードの説明 : Visual C++ デバイス プロジェクトのウィザード生成コード

デバイス プロジェクトの Visual C++ ウィザードで生成されるコードについて説明します。

方法 : プライマリ プロジェクト出力のリモート パスを指定する

プロジェクトを対象デバイスのどこに配置するかを指定する方法について説明します。

方法 : 既定のデバイスを変更する (ネイティブ プロジェクト)

ネイティブ プロジェクトの既定のターゲットを変更する方法について説明します。

参照

関連項目

Windows Mobile へようこそ

その他の技術情報

Visual C++ を使用したデバイスのプログラミング

デバイス プロジェクトのプラットフォーム切り替え

対象のプラットフォームは簡単に切り替えることができます。たとえば、同じプロジェクトを使用して、Pocket PC 2003 プラットフォームと Smartphone プラットフォームを対象にできます。

Visual C++ で複数のプラットフォームを対象にしたプロジェクトを作成するには、以下を参照してください。

- ネイティブ デバイス プロジェクトで対象となる複数のプラットフォーム
- [構成マネージャ] (スマート デバイス プロジェクト ウィザード).

構成マネージャで対象プラットフォームを設定する方法については、以下を参照してください。

- 方法 : マルチプラットフォーム対応のデバイス プロジェクトを作成する (Visual C++)
- [構成マネージャ] (スマート デバイス プロジェクト ウィザード).

Visual C++ プロジェクトで複数のプラットフォームのリソースと連携する場合の詳細については、「複数のプラットフォームでのリソースの使用」を参照してください。

参照

概念

eMbedded Visual C++ から Visual Studio 2005 へのアップグレード ウィザード

eVC からの移植に関する既知の問題

その他の技術情報

スマート デバイス開発

デバイス プロジェクトの C++ クラス ウィザード

Visual C++ デバイス プロジェクトでは、デスクトップ Visual C++ プロジェクトでサポートされるクラス ウィザードのサブセットがサポートされます。Windows プラットフォームと Windows CE プラットフォームの違いのため、一部のウィザードはデバイス プロジェクトでサポートされません。詳細については、「コード ウィザードを使用した機能の追加」を参照してください。

このセクションの内容

デバイス プロパティでコード ウィザードを使用する

サポートされる C++ クラス ウィザードと、それらにアクセスする方法について説明します。

ネイティブ デバイス プロジェクトのウィザード オプション

特定の C++ クラス ウィザードでサポートされないウィザード オプションについて説明するトピックのリンクが含まれています。

プロジェクトのプロパティ ダイアログ ボックスでサポートされないオプション

デスクトップ プロジェクトとは異なる [プロジェクトのプロパティ] ダイアログ ボックスの動作について説明します。

すべてのスマート デバイスのネイティブ アプリケーション ウィザードで、静的リンクと動的リンクの両方を選択できるわけではありません。ランタイム リンクに関する スマート デバイス アプリケーション ウィザードの動作の概要を次の表に示します。

ウィザード	備考
Win32 スマート デバイス プロジェクト – Windows アプリケーション	静的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。
Win32 スマート デバイス プロジェクト – コンソール アプリケーション	静的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。
Win32 スマート デバイス プロジェクト – DLL	静的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。
Win32 スマート デバイス プロジェクト – スタティック ライブラリ	静的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。
ATL スマート デバイス プロジェクト – DLL	静的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。
ATL スマート デバイス プロジェクト – EXE	静的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。
MFC スマート デバイス アプリケーション – SDI	静的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。
MFC スマート デバイス アプリケーション – SDI w. **DocList**	静的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。
MFC スマート デバイス アプリケーション – ダイアログ ベース	静的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。
MFC スマート デバイス DLL – 標準 DLL	静的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。
MFC スマート デバイス ActiveX コントロール	静的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。
MFC スマート デバイス DLL – 拡張 DLL	動的リンク。プロジェクトの作成時には、動的および静的リンクのオプションはありません。

上の表では、F5 ショートカット キーを使用して配置を参照できます。アプリケーションのインストールは、このセクションで説明されているとおりです。

- C++ で記述されたアプリケーションでスマート デバイスの CAB プロジェクトを作成するときに、CAB プロジェクトと DLL (**atl80.dll**、**mfc80U.dll**、**msvcrt.dll** など) を動的にリンクするには、CAB プロジェクトに依存関係を手動で追加する必要があります。動的にリンクしていて、CAB で DLL を再配布する必要がある場合、この DLL をデバイスのシステム ディレクトリ (\Windows など) にインストールしないでください。このような DLL は、ローカルのアプリケーション ディレクトリにインストールします。ATL/MFC ランタイムに動的にリンクしている一連のアプリケーションを再配布する場合、アプリケーションとランタイム DLL のすべてを 1 つのアプリケーション ディレクトリにインストールし、そのフォルダに配置されるアプリケーションにショートカットを指定します。これにより、サイズが小さくなり、システム ディレクトリ内の DLL が後でアプリケーションの他のインストールと置き換えられて、DLL に動的にリンクするアプリケーションが破壊される危険が回避されます。
- MFC/ATL DLL に関する依存関係を減らすために、静的なリンクにすることを強くお勧めします。また、静的にリンクしている場合、DLL をアプリケーションに再配布しないことをお勧めします。

参照

その他の技術情報

Visual C++ デバイス プロジェクトの開発

デバイス プロパティでコード ウィザードを使用する

デバイス プロジェクトでは、C++ デスクトップ プロジェクトと同じコード ウィザードを利用できる場合が多くあります。サポートされていないウィザードや、ウィザードがサポートされていてもオプションがサポートされていないこともあります。特定のコード ウィザードでサポートされていないオプションの詳細については、「ネイティブ デバイス プロジェクトのウィザード オプション」を参照してください。

サポートされるコード ウィザード

Visual C++ デバイス プロジェクトでは、次のウィザードをサポートしています。

- ATL シンプル オブジェクト ウィザード。このウィザードでサポートされないオプションの詳細については、「ATL シンプル オブジェクト ウィザードの [オプション] のウィザード オプション」を参照してください。
- ATL コントロール ウィザード。このウィザードでサポートされないオプションの詳細については、「オプションのウィザード オプション (ATL コントロール ウィザード)」、「インターフェイスのウィザード オプション (ATL コントロール ウィザード)」、および「ATL コントロールウィザードの [ストック プロパティ] でサポートされないウィザード オプション」を参照してください。
- ATL ダイアログ ウィザード.
- MFC プロジェクトへの ATL サポートの追加
- ATL プロパティ ページ ウィザード。このウィザードでサポートされないオプションの詳細については、「ATL プロパティ ページ ウィザードの [オプション] でサポートされないウィザード オプション」を参照してください。
- 一般 C++ クラス ウィザード
- MFC クラス ウィザード。このウィザードでサポートされないオプションの詳細については、「MFC クラス ウィザードでサポートされないウィザード オプション」を参照してください。

Visual C++ スマート デバイスのコード ウィザードへのアクセス

Visual C++ スマート デバイスのコード ウィザードは、[クラスの追加] ダイアログ ボックスで使用できます。[クラスの追加] ダイアログ ボックスには、次の方法でアクセスできます。

- [プロジェクト] メニューの [クラスの追加] を選択します。
- ソリューション エクスプローラで、任意のフォルダを右クリックし、[追加] をクリックして、[クラス] をクリックします。
- クラス ビュー ウィンドウで、適切なノードを右クリックし、[追加] をクリックして、[クラス] をクリックします。

スマート デバイス プロジェクトで使用できるウィザードを表示するには、[クラスの追加] ダイアログ ボックスの [カテゴリ] ペインにあるディレクトリ構造を展開し、[スマート デバイス] をクリックします。使用できるテンプレートが [テンプレート] ペインに表示されます。

参照

その他の技術情報

デバイス プロジェクトの C++ クラス ウィザード

ネイティブ デバイス プロジェクトのウィザード オプション

デバイス プロジェクトとデスクトップ プロジェクトには違いがあるため、C++ クラス ウィザードの一部には、デバイスでサポートされないオプションや、デバイス プロジェクトでは動作が異なるオプションがあります。

現在のプロジェクトでデバイスを対象としているときに、サポートされないウィザード オプションについて説明します。

このセクションの内容

ATL シンプル オブジェクト ウィザードの [オプション] のウィザード オプション

ATL シンプル オブジェクトウィザードの [オプション] ページで、サポートされないウィザード オプションについて説明します。

オプションのウィザード オプション (ATL コントロール ウィザード)

ATL コントロールウィザードの [オプション] ページで、サポートされないウィザード オプションについて説明します。

インターフェイスのウィザード オプション (ATL コントロール ウィザード)

ATL コントロールウィザードの [インターフェイス] ページで、サポートされないウィザード オプションについて説明します。

ATL コントロール ウィザードの [表示] でサポートされないウィザード オプション (デバイス)

ATL コントロールウィザードの [表示] ページで、サポートされないウィザード オプションについて説明します。

ATL コントロールウィザードの [ストック プロパティ] でサポートされないウィザード オプション

ATL コントロールウィザードの [ストック プロパティ] ページで、サポートされないウィザード オプションについて説明します。

ATL プロパティ ページ ウィザードの [オプション] でサポートされないウィザード オプション

ATL プロパティ ページウィザードの [オプション] ページで、サポートされないウィザード オプションについて説明します。

MFC クラス ウィザードでサポートされないウィザード オプション

MFC クラスウィザードでサポートされないウィザード オプションについて説明します。

[アプリケーションの設定] のサポートされないウィザード オプション (ATL スマート デバイス プロジェクト ウィザード)

ATL スマート デバイス プロジェクトウィザードの [アプリケーションの設定] ページで、サポートされないウィザード オプションについて説明します。

参照

その他の技術情報

デバイス プロジェクトの C++ クラス ウィザード

ATL シンプル オブジェクト ウィザードの [オプション] のウィザード オプション

ATL シンプル オブジェクト ウィザードの [オプション] ページで、スマート デバイスの場合にサポートされないウィザード オプションについて説明します。このウィザード ページに示されているいくつかの要素は、デバイスでサポートされていないか、またはデバイス プロジェクトにおいて異なる動作をします。

サポートされないオプションとサポート方法が異なるオプション

デバイス プロジェクトで異なる動作を持つ要素を次の表に示します。

セクション	動作
[スレッド モデル]	スレッド モデルの設定は、対象プラットフォームで DCOM をサポートしているかどうかによって変わります。対象プラットフォームで DCOM をサポートしていない場合、生成されるコードで使用されるスレッド モデルは、選択したスレッド モデルにかかわらず、常に "フリー" です。 Windows Mobile 2003 ソフトウェアでは DCOM をサポートしていません。 Windows CE での COM、DCOM、スレッド モデルの詳細については、Windows CE 5.0 のドキュメントの「コンポーネント サービス (COM および DCOM)」と「COM スレッドとプロセス」を参照してください。
[インターフェイス]	[オートメーション互換] チェック ボックスは、デバイス プロジェクトではサポートされません。
[サポート]	[IObjectWithSite (IE オブジェクト サポート)] チェック ボックスは、デバイス プロジェクトではサポートされません。

参照

その他の技術情報

デバイス プロジェクトの C++ クラス ウィザード

オプションのウィザード オプション (ATL コントロール ウィザード)

このウィザード ページに示されているいくつかの要素は、デバイスでサポートされていないか、またはデバイス プロジェクトにおいて異なる動作をします。

サポートされないオプションとサポート方法が異なるオプション

デバイス プロジェクトで異なる動作を持つ要素を次の表に示します。

セクション	動作
スレッド モデル	スレッド モデルの設定は、対象プラットフォームで DCOM をサポートしているかどうかによって変わります。対象プラットフォームで DCOM をサポートしていない場合、生成されるコードで使用されるスレッド モデルは、選択したスレッド モデルにかかわらず、常に "フリー" です。 Windows Mobile 2003 ソフトウェアでは DCOM をサポートしていません。 Windows CE での COM、DCOM、スレッド モデルの詳細については、Windows CE 5.0 のドキュメントの「コンポーネント サービス (COM および DCOM)」と「COM スレッドとプロセス」を参照してください。
コントロールの型	[DHTML コントロール] オプションは、デバイス プロジェクトではサポートされません。
サポート	[ライセンス] チェック ボックスは、デバイス プロジェクトではサポートされません。

参照

その他の技術情報

デバイス プロジェクトの C++ クラス ウィザード

インターフェイスのウィザード オプション (ATL コントロール ウィザード)

ATL コントロール ウィザードの [インターフェイス] ページで、スマート デバイスの場合にサポートされないインターフェイスについて説明します。

このウィザード ページに表示されるいくつかのインターフェイスは、デバイスでサポートされていないか、またはデバイス プロジェクトにおいて異なる動作をします。

サポートされていないオプションと、サポートされる別のオプション

デバイス プロジェクトで異なる動作を持つインターフェイスを次の表に示します。

インターフェイス	動作
IDataObject	このインターフェイスは、Windows CE ではサポートされません。
IPersistStorage	このインターフェイスは、DCOM をサポートするプラットフォームでのみサポートされます。 Windows Mobile™2003 ソフトウェアでは、DCOM はサポートされません。 Windows CE での COM および DCOM の詳細については、Windows CE 5.0 ドキュメントの「コンポーネント サービス (COM および DCOM)」を参照してください。

参照

その他の技術情報

デバイス プロジェクトの C++ クラス ウィザード

ATL コントロール ウィザードの [表示] でサポートされないウィザード オプション (デバイス)

このウィザード ページの一部の要素は、デバイス プロジェクトで完全にはサポートされません。

サポートの異なるオプション

次の表は、デバイス プロジェクトでの要素の相違点を示しています。

セクション	相違点
[コントロールの追加]	デバイス プラットフォームでは、次のものだけがサポートされます。 • Button • ComboBox • Edit • ListBox • Scrollbar • Static • RichEdit (大半の Windows CE プラットフォームでサポートされていますが、すべてではありません)

参照

その他の技術情報

ネイティブ デバイス プロジェクトのウィザード オプション

ATL コントロールウィザードの [ストック プロパティ] でサポートされないウィザード オプション

GUID の CLSID_StockPicturePage、CLSID_StockColorPage、および CLSID_StockFontPage がないため、デバイスの場合、このウィザード ページのストック プロパティの一部はサポートされません。

サポートされないプロパティ

デバイスの場合にサポートされないストック プロパティを次に示します。

- BackColor
- BorderColor
- FillColor
- Font
- ForeColor
- MouseIcon
- Picture

参照

その他の技術情報

デバイス プロジェクトの C++ クラス ウィザード

ATL プロパティ ページ ウィザードの [オプション] でサポートされないウィザード オプション

このウィザード ページに示されているいくつかの要素は、デバイスでサポートされていないか、またはデバイス プロジェクトにおいて異なる動作をします。

デバイス プロジェクトで異なる動作を持つ要素を次の表に示します。

セクション	動作
[スレッド モデル]	スレッド モデルの設定は、対象プラットフォームで DCOM をサポートしているかどうかによって変わります。対象プラットフォームで DCOM をサポートしていない場合、生成されるコードで使用されるスレッド モデルは、選択したスレッド モデルにかかわらず、常に "フリー" です。 Windows Mobile 2003 ソフトウェアでは DCOM をサポートしていません。 Windows CE での COM、DCOM、スレッド モデルの詳細については、Windows CE 5.0 のドキュメントの「コンポーネント サービス (COM および DCOM)」と「COM スレッドとプロセス」を参照してください。
[インターフェイス]	[オートメーション互換] チェック ボックスは、デバイス プロジェクトではサポートされません。
[サポート]	[IObjectWithSite] (IE object support) チェック ボックスは、デバイス プロジェクトではサポートされません。

参照

その他の技術情報

デバイス プロジェクトの C++ クラス ウィザード

MFC クラス ウィザードでサポートされないウィザード オプション

このウィザード ページに示されているいくつかの要素は、デバイスでサポートされていないか、またはデバイス プロジェクトにおいて異なる動作をします。

サポートされないオプション

次の要素は、デバイス プロジェクトではサポートされません。

- [アクティブ アクセシビリティ]
- [DHTML リソース ID]
- .HTM ファイル
- オートメーション
- [ドキュメント テンプレート リソースの生成]
- [ドキュメント テンプレート文字列]

参照

その他の技術情報

デバイス プロジェクトの C++ クラス ウィザード

[アプリケーションの設定] のサポートされないウィザード オプション (ATL スマート デバイス プロジェクト ウィザード)

ATL スマート デバイス プロジェクト ウィザードの [アプリケーションの設定] ページで、スマート デバイスの場合にサポートされないウィザード オプションについて説明します。

このウィザード ページに示されているいくつかの要素は、デバイスでサポートされていないか、またはデバイス プロジェクトにおいて異なる動作をします。

サポートされないオプション

デバイス プロジェクトで異なる動作を持つ要素を次の表に示します。

セクション	動作
(なし)	[属性] チェック ボックスは、デバイス プロジェクトではサポートされません。

ATL スマート デバイス プロジェクト ウィザードによりタイプ ライブラリの登録解除が実装されない

Windows Mobile では、レジストリからタイプ ライブラリを削除する COM 機能が実装されないため、ATL スマート デバイス プロジェクト ウィザードにより `DllUnregisterServer` 関数を実装するコードが代わりに生成されます。

```
// DllUnregisterServer - Removes entries from the system registry
STDAPI DllUnregisterServer(void)
{
    HRESULT hr = _AtlModule.DllUnregisterServer(false);
    return hr;
}
```

`DllUnregisterServer` 関数に **false** を渡すと、COM にタイプ ライブラリを登録解除しないように指示されます。**true** へのパラメータを変更した場合、DllUnregisterServer へのすべての呼び出しが `E_NOTIMPL` を生成して失敗します。

参照

その他の技術情報

ネイティブ デバイス プロジェクトのウィザード オプション

プロジェクトのプロパティ ダイアログ ボックスでサポートされないオプション

[プロジェクトのプロパティ] ダイアログ ボックスのいくつかの要素は、デバイスでサポートされていないか、またはデバイス プロジェクトにおいて異なる動作をします。

サポートされていないオプションと、サポートされる別のオプション

デバイス プロジェクトでデスクトップ PC とは異なる動作を持つ要素を次の表に示します。

セクション	動作
[プロジェクトの既定値]	[MFC の使用法] を [スタティック ライブラリで MFC を使用する] に設定し、[ATL の使用] を [ATL に動的にリンク] に設定することも可能ですが、このシナリオはデバイス プロジェクトではサポートされません。
[プロジェクトの既定値]	**メモ：** このアイテムは、MFC スマート デバイス アプリケーション プロジェクトおよび MFC スマート デバイス DLL プロジェクトだけに適用されます。 [MFC の使用法] を [共有 DLL で MFC を使用する] (既定値) から [スタティック ライブラリで MFC を使用する] に変更する場合は、[配置] プロパティ ページの [全般] セクションの [追加ファイル] から **mfc80ud.dll** (デバッグ) と **mfc80u.dll** (リリース) を削除する必要があります。 同様に、[MFC の使用法] を [スタティック ライブラリで MFC を使用する] から [共有 DLL で MFC を使用する] に変更する場合は、[配置] プロパティ ページの [全般] セクションの [追加ファイル] に **mfc80ud.dll** (デバッグ) と **mfc80u.dll** (リリース) を追加する必要があります。 **mfc80u.dll** または **mfc80ud.dll** を追加するときは、**atl80.dll** および **msvcr80.dll** または **msvcr80d.dll** も追加する必要があります。

参照

関連項目

プロパティ ページ (C++)

その他の技術情報

ネイティブ デバイス プロジェクトのウィザード オプション

デバイス プロジェクトのリソース エディタ

デバイス プロジェクトでリソース エディタを使用するのは、デスクトップの Visual C++ プロジェクトでリソース エディタを使用するのと似ています。このセクションでは、デスクトップのリソース エディタとデバイスのリソース エディタの間の違いについていくつか詳細に説明します。詳細については、「リソース エディタ」を参照してください。

デバイス SDK では、独自のユーザー インターフェイス (UI) モデルを定義できます。このモデルは、ダイアログ エディタに表示されるコントロールのリストにフィルタ処理をして、そのプラットフォームでサポートされるコントロールのみを表示するために使用できます。Visual Studio 2005 には、Windows Mobile 2003 SDK を含む、Windows CE UI モデルが付属しています。

このセクションの内容

デバイス プロジェクトのリソース エディタの使用

デバイス オブジェクトでリソース エディタを使用する方法の概要について説明します。

複数のプラットフォームでのリソースの使用

複数のプラットフォームをターゲットとするプロジェクトでリソースを使用する方法について説明します。

デバイスのダイアログ ボックス コントロール

デバイス プロジェクトでサポートされるダイアログ コントロールについて説明します。

参照

その他の技術情報

Visual C++ デバイス プロジェクトの開発

デバイス プロジェクトのリソース エディタの使用

デバイス プロジェクトのリソース エディタは、デスクトップ プロジェクトのリソース エディタとほとんど同じです。すべてのエディタがサポートされますが、ダイアログ エディタのみ大きく変更されました。詳細については、「リソース エディタ」を参照してください。

Visual Studio 2005 のネイティブ スマート デバイス プロジェクトでは、次のリソースの種類がサポートされます。

- アクセラレータ
- ビットマップ
- カーソル
- ダイアログ
- アイコン
- メニュー
- レジストリ
- ストリング テーブル
- ツール バー
- バージョン

ダイアログ エディタ

デバイスのダイアログ エディタは、次の点でデスクトップのダイアログ エディタとは異なります。

- デスクトップ コントロールにあるいくつかのコントロールがなくなっており、デバイスでサポートされるコントロールの属性は、対応するデスクトップ コントロールとわずかに異なっています。詳細については、「デバイスのダイアログ ボックス コントロール」を参照してください。
- 一般的なデバイス フォームの要因の新しいダイアログ テンプレートがあります。
- ダイアログ ボックス コントロールの動作とプロパティは、インストールされた各ソフトウェア開発キット (SDK) に付属するユーザー インターフェイス (UI) モデルが基になります。この UI モデルにより、現在ターゲットとなっているプラットフォームのコントロールの正しいセットが提供されます。SDK により UI モデルが定義されない場合、ダイアログ エディタでは既定の Windows CE UI モデルが使用されます。
- デバイス プロジェクトには、固有の 2 つのコントロールである 入力パネルの状態コントロール と CAPEdit コントロール があります。

RC2 ファイル

一部のアプリケーション ウィザードでは、標準の (.RC) リソース ファイルに加えて .RC2 リソース ファイルが生成されます。この .RC2 ファイルは、リソース コンパイラによりコンパイルするためのものではなく、実際にはリソース コンパイラにより処理されないリソースが含まれています。例として、HI_RES_AWARE カスタム リソースおよびメニュー リソース データ (RCDATA) などがあります。.RC2 ファイルは、リソース コンパイラで編集しない他のカスタム リソースを配置する巨大な場所です。

Smartphone のメニュー リソースを作成する方法の詳細については、「How to: Create a Soft Key Bar」を参照してください。Smartphone メニューを作成するには、RCDATA セクションがあることを確認します。通常、これは .RC2 ファイルにあります。リソース ID は、100 以上の値にする必要があります。ID はリソース ヘッダー ファイル (Smartphone の場合は resourcesp.h) で設定されます。ボタンには、インデックスとして NOMENU が必要です (IDR_MENU RCDATA)。この点を次の例に示します。

```
BEGIN
    IDR_MENU,
    2,
    I_IMAGENONE, IDM_OK, TBSTATE_ENABLED, TBSTYLE_BUTTON | TBSTYLE_AUTOSIZE,
    IDS_OK, 0, NOMENU,
    I_IMAGENONE, IDM_HELP, TBSTATE_ENABLED, TBSTYLE_DROPDOWN | TBSTYLE_AUTOSIZE,
    IDS_HELP, 0, 0,
END
```

デバイスのリソース エディタで作業をしている場合、次の理由でエラーが発生する可能性があります。

- フォームまたはユーザー コントロールなどの他のプロジェクト アイテムに属する RESX アイテムを編集しているため。

- Windows フォーム デザイナでは、コントロールにリンクしないすべてのアイテムが自動的に破棄されるため。コメントもすべて削除され、リンクされたアイテムはサポートされません。さらに、フォームまたはユーザー コントロールがリソース エディタで RESX ファイルに追加された場合は、そのフォームまたはユーザー コントロールの読み込みに失敗します。
- Tiff ファイルなどのいくつかのリソースの種類は、Windows CE ではサポートされないため。
- リソース ファイルの形式がサポートされない、ファイルが空である、または形式が破損している場合も、エラーが生成されるため。

参照

その他の技術情報

デバイス プロジェクトのリソース エディタ

複数のプラットフォームでのリソースの使用

Visual Studio 2005 では、1 つのデバイス プロジェクトで、Pocket PC および Smartphone などの複数のプラットフォームをターゲットとすることができます。プラットフォーム間でのユーザー インターフェイス (UI) の違いのため、各プラットフォームではプロジェクト内に独自のリソース スクリプト (.rc) ファイルが必要です。

複数のリソース ファイル

複数のプラットフォームをターゲットとするようにデバイス プロジェクトを設定するには、次の 2 つの方法があります。

- プロジェクト作成時にアプリケーション ウィザードを使用
- プロジェクトの作成後

プロジェクトのアプリケーション ウィザードの [プラットフォーム] ページで複数のプラットフォームを選択した場合、リソース ファイルが生成され、使用している各プラットフォーム用に構成されます。たとえば、ターゲット プラットフォームとして Pocket PC および Smartphone を選択した場合、Pocket PC のリソース ファイルは Smartphone プラットフォームのビルドから除外され、Smartphone のリソース ファイルは Pocket PC プラットフォームのビルドから除外されます。

ただし、プロジェクトの作成後にプラットフォームを追加した場合、プラットフォームとリソース ファイルを手動で追加する必要があります。

新しいプラットフォームの追加

新しいプラットフォームを追加するには

1. [ビルド] メニューの [構成マネージャ] をクリックします。
2. [アクティブ ソリューション プラットフォーム] ボックスで、[[<新規作成...>] をクリックします。
3. プロジェクトに追加するプラットフォームを選択し、設定のコピー元のプラットフォームを選択します。次に、[OK] をクリックします。

   メモ :

   [<既定>] から設定をコピーする場合、プラットフォームのプロジェクト プロパティは空になります。類似したプラットフォームから設定をコピーし、必要に応じてプロジェクト プロパティを変更することをお勧めします。たとえば、プラットフォームとして Smartphone を追加する場合、Pocket PC プラットフォームから設定をコピーします。

4. [閉じる] をクリックします。

新しいリソース ファイルの追加

新しいプラットフォームを追加したら、そのプラットフォームのリソースを追加する必要があります。

新しいプラットフォームのリソース ファイルを追加するには

1. [プロジェクト] メニューの [新しい項目の追加] をクリックします。
2. [新しい項目の追加] ダイアログ ボックスの [リソース] をクリックし、[テンプレート] ペインの [リソース ファイル (.rc)] をクリックします。
3. [ファイル名] ボックスで、ファイルの名前を入力し、[追加] をクリックします。

新しいリソース スクリプト (.rc) ファイルに対応する新しいヘッダー (.h) ファイルがプロジェクトに追加されます。

ビルドからのリソース ファイルの除外

ターゲット プラットフォームのプロジェクトをビルドする際、他のプラットフォームからのリソース ファイルを含める必要のない場合があります。ターゲットとなるプラットフォームをベースとして、ビルドからファイルを除外できます。

ビルドからリソース ファイルを除外するには

1. リソース スクリプト (.rc) ファイルを右クリックし、[プロパティ] をクリックします。
2. [プラットフォーム] ボックスで、一覧から最初のプラットフォームを選択します。
3. [全般] プロパティ ページで、選択されたプラットフォーム用にプロジェクトがビルドされる際にこの .rc ファイルを含めない場合は、[ビルドから除外] ボックスの [はい] を選択します。

4. プラットフォーム構成ごとに前の手順を繰り返し、現在選択されているプラットフォームに所属しないリソース ファイルだけが除外されることを確認します。

5. 前のすべての手順 (1 ～ 4) を、プロジェクト内の .rc ファイルごとに繰り返します。

ソリューション エクスプローラでは、現在選択されているプラットフォームのビルドから除外されている各ファイルのアイコンに赤いマークが付いているのがわかります。

新しいプラットフォーム構成のプロジェクト プロパティの変更

リソース ファイルをプラットフォーム用にセットアップしたら、プロジェクト プロパティが新しいプラットフォーム構成にとって適切な内容となっていることを確認する必要があります。類似したプラットフォームから設定をコピーした場合、変更する設定は多くありませんが、[<既定>] を選択した場合はすべての設定を手動で追加する必要があります。この例では、新しい Smartphone 2003 (ARMV4) プラットフォームをプロジェクトに追加し、Pocket PC 2003 (ARMV4) プラットフォームから設定をコピーしたことを想定しています。

プロジェクト プロパティを変更するには

1. [プロジェクト] メニューの [プロパティ] をクリックします。

2. [C/C++] ノードを展開し、[プリプロセッサ] をクリックします。

3. [プリプロセッサの定義] ボックスで、「**POCKETPC2003_UI_MODEL**」を「**SMARTPHONE2003_UI_MODEL**」に変更し、[OK] をクリックします。

メモ :
別のプラットフォームを追加したか、別のプラットフォームから設定をコピーした場合は、さらに設定を変更する必要が生じることがあります。

ヘッダー ファイルへの #ifdef ディレクティブの追加

プロジェクトのメインのヘッダー ファイルでは、前の手順で設定した UI モデルのプリプロセッサ定義を確認し、対応するリソース ファイルだけが含まれていることを確認する必要があります。

ヘッダーファイルに #ifdef ディレクティブを追加するには

1. *ProjectName*.h を開きます。

2. 元のプラットフォームの UI モデルの **#ifdef** の後に、次のコードを追加します。

```
#ifdef SMARTPHONE2003_UI_MODEL
   #include "ResourceFileName.h"
#endif
```

3.

参照

その他の技術情報

デバイス プロジェクトのリソース エディタ

デバイスのダイアログ ボックス コントロール

ダイアログ エディタを使用すると、Visual C++ デスクトップ プロジェクトの場合とほぼ同じ方法で、Visual C++ デバイス プロジェクトのダイアログ ボックス リソースを作成または編集できます。このセクションでは、デバイス プロジェクトのダイアログ エディタでサポートされているコモン デスクトップ コントロールに加え、デバイス固有の 2 つのコントロールについて説明します。

このセクションの内容

デバイスでサポートされているダイアログ ボックス コントロール

デバイスでもサポートされているコモン デスクトップ コントロールについて説明します。

デバイス固有のダイアログ ボックス コントロール

デバイス固有の 2 つのコントロールについて説明します。

参照

その他の技術情報

デバイス プロジェクトのリソース エディタ

デバイスでサポートされているダイアログ ボックス コントロール

Visual Studio 2005 のデバイス ダイアログ ボックス エディタでは、デスクトップ プロジェクトでサポートされている多くのコモン コントロールに加え、デバイス固有の 2 つのコントロールもサポートしています。

サポートされているコントロール

デバイス用のダイアログ ボックス エディタでサポートされているコモン コントロールを次に示します。

ボタン	垂直スクロール バー
チェック ボックス	スライダ コントロール
エディット コントロール	プログレス コントロール
コンボ ボックス	リスト コントロール
リスト ボックス	ツリー コントロール
グループ ボックス	タブ コントロール
オプション ボタン	日時指定コントロール
静的テキスト	月間予定表
ピクチャ コントロール	カスタム コントロール
水平スクロール バー	

デバイスでサポートされているダイアログ ボックス コントロールの最新の一覧を参照するには、ネイティブな Win32.exe デバイス プロジェクトを作成し、RC ファイルをダブルクリックした後、[ダイアログ] ノードを展開して一覧を開きます。

デバイス固有のコントロール

Visual Studio 2005 のダイアログ ボックス エディタには、デバイスの開発用に 2 つの新しいコントロールも追加されています。

コントロール	説明
CAPEdit コントロール	CAPEdit コントロールは、コントロール内の各単語の先頭を大文字にするエディット コントロールです。
入力パネルの状態コントロール	入力パネルの状態コントロールは、このコントロールをサポートするプラットフォームの入力パネルを有効にします。

参照

その他の技術情報

デバイスのダイアログ ボックス コントロール

デバイス固有のダイアログ ボックス コントロール

入力パネルの状態と CAPEdit Control は、デバイス固有のダイアログ ボックスです。これらのダイアログ ボックスは、デバイス固有の特性を利用するようにデザインされています。

このセクションの内容

CAPEdit コントロール

CAPEdit コントロールは、コントロール内に入力された最初の文字を大文字にするエディット コントロールです。

入力パネルの状態コントロール

State of Input Panel Control は、任意のダイアログ ボックスに入力パネルを表示します。

参照

その他の技術情報

デバイスのダイアログ ボックス コントロール

CAPEdit コントロール

CAPEdit コントロールは、コントロールの最初の文字が自動的に大文字になる点を除いて、CEdit クラス コントロールと同じように動作します。このコントロールは、長い文字列を入力する処理を簡略化します。また、名前や場所など、常に先頭を大文字にする情報を入力する必要がある場合によく使用されます。

参照

関連項目

CDialog クラス

その他の技術情報

デバイス固有のダイアログ ボックス コントロール

Creating an Edit Control

入力パネルの状態コントロール

入力パネルの状態コントロールは、Pocket PC など、入力パネルをサポートするプラットフォームで入力パネルを表示する非可視のコントロールです。このコントロールがダイアログ ボックスに含まれている場合、入力コントロールがフォーカスを得ると必ず入力パネルが浮き出て表示されます。コントロールがフォーカスを失うと、入力パネルは元に戻ります。

メモ：
ダイアログ ボックスに複数の 入力パネルの状態コントロールを追加すると、アプリケーションがデバイス上でハングアップします。この問題は、今後のリリースで解決される可能性があります。

参照

概念

デバイスでサポートされているダイアログ ボックス コントロール

その他の技術情報

デバイス固有のダイアログ ボックス コントロール

スマート デバイスのコンパイラ

Visual Studio 2005 には、スマート デバイスで使用されるマイクロプロセッサをターゲットとする次のコンパイラが含まれています。

- 32 ビット ARM C、および C++ プログラムのコンパイルとリンクに使用される 32 ビット C/C++ コンパイラ
- 32 ビット Renesas SH-4 C、および C++ プログラムのコンパイルとリンクに使用される 32 ビット C/C++ コンパイラ
- MIPS16、MIPS32、MIPS64 C、および C++ プログラムのコンパイルとリンクに使用される C/C++ コンパイラ

コンパイラは、COFF (Common Object File Format) のオブジェクト ファイルを生成します。コンパイラ プログラムは、各ソース ファイルと、指定されていない限り各コンパイルのオブジェクト ファイルをコンパイルします。コンパイラには、コマンド ライン (CL)、CL 環境変数、および指定されたすべての応答ファイルに一覧表示されたオプションが含まれています。詳細については、「Compilers for Smart Devices」を参照してください。

デスクトップとデバイスの間での Visual Studio 2005 コンパイラの相違点

相違点	説明
[詳細] タブの [アーキテクチャ用コンパイル] ドロップダウン リスト	デバイス プロジェクトの [プロジェクトのプロパティ] の [詳細] タブの [アーキテクチャ用コンパイル] ドロップダウン リスト ([C/C++] ノードの下) では、一覧に [ARM4 (/QRarch4)]、[ARM5 (/QRarch5)]、[ARM4T (/QRarch4t)]、[ARM5T (/QRarch5t)] というオプションがあります。
[詳細] タブの [ARM と ARM Thumb 呼び出しの相互作用] ドロップダウン リスト	デバイス プロジェクトの [プロジェクトのプロパティ] の [詳細] タブの [ARM と ARM Thumb 呼び出しの相互作用] ドロップダウン リスト ([C/C++] ノードの下) では、一覧に [はい](/QRInterwork-return)、[いいえ] というオプションがあります。[はい] に設定すると、コンパイラは、ARM 16 および 32 ビット コードを相互にサンクするコードを生成します。
[詳細] タブの [浮動小数点のエミュレーションを有効にする] ドロップダウン リスト	デバイス プロジェクトの [プロジェクトのプロパティ] の [詳細] タブの [浮動小数点のエミュレーションを有効にする] ドロップダウン リスト ([C/C++] ノードの下) では、一覧に [はい] と [いいえ] というオプションがあります。[はい] に設定すると、コンパイラで浮動小数点演算のエミュレーションが有効になります。
[プリプロセッサ] タブの [プリプロセッサの定義] 入力ボックス	デバイス プロジェクトの [プロジェクトのプロパティ] の [プリプロセッサ] タブの [プリプロセッサの定義] 入力ボックス ([C/C++] ノードの下) には、[親またはプロジェクトの既定値から継承] チェック ボックスと、マクロを追加するための [マクロ] ボタンがあります。
[最適化] タブの [浮動小数点の整合性] ドロップダウン リスト	デバイス プロジェクトの [プロジェクトのプロパティ] の [最適化] タブの [浮動小数点の整合性] ドロップダウン リスト ([C/C++] ノードの下) には、[既定の整合性] または [整合性の向上 (/Op)] を選択するためのドロップダウン リストがあります。

詳細については、「アルファベット順のコンパイラ オプション」を参照してください。

Visual Studio 2003 と Visual Studio 2005 の間のコンパイラの変更

デバイス コンパイラは、デスクトップ コンピュータの Visual C++ コンパイラをベースとしているため、デスクトップ コンパイラのバージョン間の違いを調べると、eMbedded Visual C++ デバイス コンパイラと Visual Studio 2005 デバイス コンパイラの間の変更をよく理解できます。Visual Studio 6.0 と Visual Studio 2003 の間の変更の詳細については、「Standard Compliance Issues in Visual C++」を参照してください。

Visual Studio 2003 と Visual Studio 2005 の間のコンパイラの変更について、概要を次の表に示します。

問題	説明
メンバへのポインタでは、関数呼び出しに修飾名、アドレス演算子 (&)、およびかっこが必要になりました。	メソッド名のみを使用した以前のバージョンのコンパイラ用に記述されたコードでは、コンパイラ エラー C3867 またはコンパイラの警告 C4867 が生成されるようになりました。この診断は、ISO C++ 標準により必要とされます。メンバ関数へのポインタを作成するには、アドレス演算子 (&) とメソッドの完全修飾名を使用する必要があります。エラーは、& 演算子とメソッドの完全修飾名を要求しないか、関数呼び出しでかっこが見つからないことが原因で発生する可能性があります。引数リストなしで関数の名前を使用すると、複数の種類に変換可能な関数ポインタが発生します。そのため、実行時にコードにより予期しない動作が行われる可能性があります。

クラスは、**friend** 宣言にアクセスできる必要があります。	以前の Visual C++ コンパイラは、宣言を含むクラスのスコープ内でアクセスできないクラスへの friend 宣言を有効にしていました。Visual C++ 2005 では、このような状況の場合はコンパイラによりコンパイラ エラー C2248 が生成されます。このエラーを解決するには、friend 宣言で指定されたクラスのアクセシビリティを変更します。この変更は、ISO C++ 標準に準拠するために加えられました。
コピー コンストラクタおよびコピー代入演算子に明示的な特化を使用できません。	コピー コンストラクタまたはコピー代入演算子の明示的なテンプレート特化に依存するコードでは、コンパイラ エラー C2299 が生成されるようになりました。ISO C++ 標準では、この使用が禁止されています。この変更は、標準に準拠して、コードの移植性を向上するために加えられました。
特化されていないクラス テンプレートは、基本クラス リストのテンプレート引数として使用できません。	クラス定義用の基本クラス リスト内にある特化されていないテンプレート クラス名を使用すると、コンパイラ エラー C3203 が生成されます。特化されていないテンプレート クラス名を、基本クラス リストでテンプレート パラメータとして使用することはできません。基本クラス リストでテンプレート パラメータとしてテンプレート クラスを使用する際は、テンプレート型パラメータをテンプレート クラスに明示的に追加する必要があります。この変更は、標準に準拠して、コードの移植性を向上するために加えられました。
入れ子にされた型の宣言を使用することはできなくなりました。	入れ子にされた型の宣言を使用するコードでは、コンパイラ エラー C2885 が生成されるようになりました。このエラーを解決するには、入れ子にされた型への参照を完全に修飾するか、名前空間に型を配置するか、または `typedef` を作成する必要があります。この変更は、標準に準拠して、コードの移植性を向上するために加えられました。
/YX コンパイラ オプションが削除されました。	**/YX** コンパイラ オプションにより、自動プリコンパイル済みヘッダーをサポートしていました。このオプションは、開発環境で既定で使用されていました。ビルド構成から **/YX** コンパイラ オプションを削除した場合、ビルドが高速化する場合があります。パフォーマンスの問題に加えて、**/YX** コンパイラ オプションにより実行時に予期しない動作が発生する可能性があります。**/Yc** (プリコンパイル済みヘッダーを作成する)、および **/Yu** (プリコンパイル済みヘッダー ファイルを使用する) の方が、プリコンパイル済みヘッダーの用途をより細かく制御できるために適しています。
/Oa および **/Ow** コンパイラ オプションが削除されました。	**/Oa** および **/Ow** コンパイラ オプションは削除されたため、無視されます。コンパイラがエイリアスを使用する方法を指定するには、**noalias** 修飾子または **restrict declspec** 修飾子を使用してください。
/Op コンパイラ オプションが削除されました。	**/Op** コンパイラ オプションは削除されました。代わりに **/fp:precise** を使用できます。
/ML および **/MLd** コンパイラ オプションが削除されました。	Visual C++ 2005 では、シングルスレッドの、静的にリンクされた CRT ライブラリがサポートされなくなりました。代わりに **/MT** および **/MTd** を使用できます。
/G3、**/G4**、**/G5**、**/G6**、**/G7**、および **/GB** コンパイラ オプションが削除されました。	コンパイラでは、すべてのアーキテクチャに最適な出力ファイルの作成を試みる、融合されたモデルが使用されるようになりました。
/Gf コンパイラ オプションは削除されました。	代わりに **/GF** を使用できます。**/GF** は、プールされた文字列を読み取り専用セクションに配置します。これは、**/Gf** がそれらの文字列を追加する書き込み可能セクションより安全です。
/GS コンパイラ オプションは、既定で有効になりました。	バッファ オーバーフローのチェックが、既定で有効になりました。**/GS-** を使用すると、バッファ オーバーフローのチェックを無効にできます。

/Zc:wchar_t 変数は既定で有効になりました。	これは、ISO C++ 標準の動作です。wchar_t 変数は、短い符号なし整数の代わりに組み込み型が既定になります。この変更により、クライアント コードが、**/Zc:wchar_t** を使用せずにコンパイルされたライブラリとリンクしている場合は、バイナリ互換性がなくなります。**/Zc:wchar_t-** を使用すると、標準の動作ではない以前の動作に戻すことができます。この変更は、準拠したコードを既定で作成するために加えられました。
/Zc:forScope 変数は既定で有効になりました。	これは、ISO C++ 標準の動作です。**for** ループ スコープが終了した後に **for** ループで宣言された変数の使用に依存しているコードは、コンパイルに失敗するようになりました。**/Zc:forScope** を使用すると、標準の動作ではない以前の動作に戻すことができます。この変更は、準拠したコードを既定で作成するために加えられました。
Visual C++ 属性のパラメータ チェックを強制的に実行します。	型が文字列ではない場合は引用符で囲まれており、型が文字列の場合には引用符がない属性コンストラクタに、名前付き属性を渡すコードでは、コンパイラ エラー C2065 またはコンパイラの警告 (レベル 1) C4581 が生成されるようになりました。以前は、すべてのコンパイラ属性が文字列として解析され、必要な場合は不足している引用符をコンパイラが挿入していました。属性サポートは、パラメータ チェック検査を追加することにより強化されました。この変更により、属性コンストラクタに渡される間違った引数が原因で発生する、予期しない動作が回避されます。
コンパイラが、宣言内の既定の型として **int** 型を挿入しなくなりました。	宣言内に型がないコードの既定は、**int** 型ではなくなりました。コンパイラでは、コンパイラの警告 C4430 またはコンパイラの警告 (レベル 4) C4431 が生成されます。ISO C++ 標準では、既定の **int** がサポートされません。この変更は、明示的に指定した型を確実に取得するのに役立ちます。

詳細については、「Breaking Changes in the Visual C++ 2005 Compiler」を参照してください。

参照

その他の技術情報

Visual C++ デバイス プロジェクトの開発

コードの説明：Visual C++ デバイス プロジェクトのウィザード生成コード

Visual C++ の各種機能とウィザードにより、マルチプラットフォーム デバイス アプリケーション、構成ファイル、およびプロジェクト ファイルを生成するための日常的タスクの大部分は簡略化されています。このチュートリアルでは、Win32 スマート デバイス プロジェクト ウィザードによって自動生成されるコードについて説明します。このコードを理解していれば、Windows アプリケーションを必要に応じて拡張したり修正したりできます。

ウィザードで生成された Windows 32 デバイス アプリケーション用のコード

Windows (Win32) のデスクトップ プログラミングに習熟している開発者は、ウィザードで生成されるメイン ルーチンを簡単に見つけることができます。

通常の Windows デバイス アプリケーションでは、メインのエントリは <yourprojectname>.cpp ファイルに含まれます。このファイルのサンプル リストが続きます。

このアプリケーションには、主に次の機能があります。

- **WinMain 関数**
- **MyRegisterClass 関数**
- **InitInstance 関数**
- **About Dialog 関数**

各関数について、次に詳細を示します。

- **WinMain 関数**：`int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)`。詳細については、「WinMain Function」を参照してください。

```
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPTSTR    lpCmdLine,
                   int       nCmdShow)
{
      MSG msg;

      // Perform application initialization:
      if (!InitInstance(hInstance, nCmdShow))
      {
            return FALSE;
      }
```

- Win32 プラットフォーム。詳細については、「Win32 プラットフォーム」を参照してください。

```
#ifndef WIN32_PLATFORM_WFSP
        HACCEL hAccelTable;
        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_WIN32SMARTDEVICE);
#endif // !WIN32_PLATFORM_WFSP
```

- メッセージ ループ。詳細については、「メッセージ処理とコマンド ターゲット」を参照してください。

```
        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
#ifndef WIN32_PLATFORM_WFSP
            if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
#endif // !WIN32_PLATFORM_WFSP
            {
                  TranslateMessage(&msg);
                  DispatchMessage(&msg);
```

```
            }
        }

        return (int) msg.wParam;
}
```

- **MyRegisterClass 関数は、**Windows **アプリケーションを登録します。**

```
//  FUNCTION: MyRegisterClass()
ATOM MyRegisterClass(HINSTANCE hInstance, LPTSTR szWindowClass)
{
        WNDCLASS wc;

        wc.style         = CS_HREDRAW | CS_VREDRAW;
        wc.lpfnWndProc   = (WNDPROC)WndProc;
        wc.cbClsExtra    = 0;
        wc.cbWndExtra    = 0;
        wc.hInstance     = hInstance;
        wc.hIcon         = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_WIN32SMARTDEVICE));
        wc.hCursor       = 0;
        wc.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
        wc.lpszMenuName  = 0;
        wc.lpszClassName = szWindowClass;

        return RegisterClass(&wc);
}
```

- **InitInstance 関数** : `FUNCTION InitInstance(HANDLE, int)`**。詳細については、「**InitInstance **メンバ関数」を参照してください。**

  `WIN32_PLATFORM_WFSP` **は** Smartphone **の場合の条件として使用し、**`WIN32_PLATFORM_PSPC` **は** Pocket PC **の場合の条件として使用します。さらに区別するために、いつでも独自の条件を定義できます。**

```
// FUNCTION: InitInstance(HANDLE, int)
// PURPOSE: Saves instance handle and creates main window.
// COMMENTS:
// In this function, we save the instance handle in a global
// variable and create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;
    TCHAR szTitle[MAX_LOADSTRING];
// title bar text
    TCHAR szWindowClass[MAX_LOADSTRING];
// main window class name

    g_hInst = hInstance;
// Store instance handle in your global variable.

#ifdef WIN32_PLATFORM_PSPC
    // SHInitExtraControls should be called once during your application's initializat
ion to initialize any
    // of the Pocket PC special controls such as CAPEDIT and SIPPREF.
    SHInitExtraControls();
#endif // WIN32_PLATFORM_PSPC
```

```
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadString(hInstance, IDC_WIN32SMARTDEVICE, szWindowClass, MAX_LOADSTRING);

#if defined(WIN32_PLATFORM_PSPC) || defined(WIN32_PLATFORM_WFSP)
    //If it is already running, then focus on the window, and exit.
    hWnd = FindWindow(szWindowClass, szTitle);
    if (hWnd)
    {
        // Set the focus to the foremost child window.
        // The "| 0x00000001" is used to bring any owned windows
        //to the foreground and activate them.
        SetForegroundWindow((HWND)((ULONG) hWnd | 0x00000001));
        return 0;
    }
#endif // WIN32_PLATFORM_PSPC || WIN32_PLATFORM_WFSP

    if (!MyRegisterClass(hInstance, szWindowClass))
    {
      return FALSE;
    }

    hWnd = CreateWindow(szWindowClass, szTitle, WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, hInsta
nce, NULL);

    if (!hWnd)
    {
        return FALSE;
    }

#ifdef WIN32_PLATFORM_PSPC
    // When the main window is created using CW_USEDEFAULT,
    // the height of the menubar is not taken into account.
    // So the generated code resizes the window after creating it.
    if (g_hWndMenuBar)
    {
        RECT rc;
        RECT rcMenuBar;

        GetWindowRect(hWnd, &rc);
        GetWindowRect(g_hWndMenuBar, &rcMenuBar);
        rc.bottom -= (rcMenuBar.bottom - rcMenuBar.top);

        MoveWindow(hWnd, rc.left, rc.top, rc.right-rc.left, rc.bottom-rc.top, FALSE);
    }
#endif // WIN32_PLATFORM_PSPC

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);


    return TRUE;
}
```

- **また、ダイアログの作成方法の例として、アプリケーションの [バージョン情報] ダイアログ ボックスも生成されます。コードは次のとおりです。** `LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)`

```
// win32smartdevice.cpp : Defines the entry point for the application.
#include "stdafx.h"
#include "win32smartdevice.h"
#include <windows.h>
#include <commctrl.h>

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE               g_hInst;
// Current instance:
HWND                    g_hWndMenuBar;
// Menu bar handle

// Forward declarations of functions included in this code module:
ATOM              MyRegisterClass(HINSTANCE, LPTSTR);
BOOL              InitInstance(HINSTANCE, int);
LRESULT CALLBACK  WndProc(HWND, UINT, WPARAM, LPARAM);
#ifndef WIN32_PLATFORM_WFSP
LRESULT CALLBACK  About(HWND, UINT, WPARAM, LPARAM);
#endif
// !WIN32_PLATFORM_WFSP
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//  PURPOSE:  Processes messages for the main window.
//  WM_COMMAND    - process the application menu
//  WM_PAINT      - Paint the main window
//  WM_DESTROY    - post a quit message and return
```

- **機能拡張のために、いくつかの主要なメッセージ (`WM_COMMAND` など) が既に含まれています。また、処理システムとユーザー入力メッセージのために、`WinProc` が含まれています。コードは次のとおりです。`LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam`。`WinProc` の詳細については、「Windows Overview」を参照してください。**

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
        HGDIOBJ  hbrWhite, hbrGray;
        POINT aptStar[6] = {50,2, 2,98, 98,33, 2,33, 98,98, 50,2};

#if defined(SHELL_AYGSHELL) && !defined(WIN32_PLATFORM_WFSP)
    static SHACTIVATEINFO s_sai;
#endif // SHELL_AYGSHELL && !WIN32_PLATFORM_WFSP
    switch (message)
    {
        case WM_COMMAND:
            wmId    = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
#ifndef WIN32_PLATFORM_WFSP
                case IDM_HELP_ABOUT:
                    DialogBox(g_hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
                    break;
#endif // !WIN32_PLATFORM_WFSP
#ifdef WIN32_PLATFORM_WFSP
                case IDM_OK:
```

```
                    DestroyWindow(hWnd);
                    break;
#endif // WIN32_PLATFORM_WFSP
#ifndef WIN32_PLATFORM_WFSP
                case IDM_OK:
                    SendMessage (hWnd, WM_CLOSE, 0, 0);
                    break;
#endif // !WIN32_PLATFORM_WFSP
                case default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
            }
            break;
        case WM_CREATE:
#ifdef SHELL_AYGSHELL
            SHMENUBARINFO mbi;

            memset(&mbi, 0, sizeof(SHMENUBARINFO));
            mbi.cbSize     = sizeof(SHMENUBARINFO);
            mbi.hwndParent = hWnd;
            mbi.nToolBarId = IDR_MENU;
            mbi.hInstRes   = g_hInst;

            if (!SHCreateMenuBar(&mbi))
            {
                g_hWndMenuBar = NULL;
            }
            else
            {
                g_hWndMenuBar = mbi.hwndMB;
            }

#ifndef WIN32_PLATFORM_WFSP
            // Initialize the shell activate info structure
            memset(&s_sai, 0, sizeof (s_sai));
            s_sai.cbSize = sizeof (s_sai);
#endif // !WIN32_PLATFORM_WFSP
#endif // SHELL_AYGSHELL
      hbrWhite = GetStockObject(WHITE_BRUSH);
        hbrGray  = GetStockObject(GRAY_BRUSH);
        return 0L;
break;
        case WM_PAINT:
         RECT  rc;
       hdc = BeginPaint(hWnd, &ps);
       GetClientRect(hWnd, &rc);
       Polyline(hdc, aptStar, 6);
       EndPaint(hWnd, &ps);
       return 0L;

            break;
        case WM_DESTROY:
#ifdef SHELL_AYGSHELL
            CommandBar_Destroy(g_hWndMenuBar);
#endif // SHELL_AYGSHELL
            PostQuitMessage(0);
            break;

#if defined(SHELL_AYGSHELL) && !defined(WIN32_PLATFORM_WFSP)
        case WM_ACTIVATE:
```

```
            // Notify shell of your activate message.
            SHHandleWMActivate(hWnd, wParam, lParam, &s_sai, FALSE);
            break;
        case WM_SETTINGCHANGE:
            SHHandleWMSettingChange(hWnd, wParam, lParam, &s_sai);
            break;
#endif // SHELL_AYGSHELL && !WIN32_PLATFORM_WFSP

        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

#ifndef WIN32_PLATFORM_WFSP
// Message handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_INITDIALOG:
#ifdef SHELL_AYGSHELL
            {
                // Create a Done button and size it.
                SHINITDLGINFO shidi;
                shidi.dwMask = SHIDIM_FLAGS;
                shidi.dwFlags = SHIDIF_DONEBUTTON | SHIDIF_SIPDOWN | SHIDIF_SIZEDLGFUL
LSCREEN | SHIDIF_EMPTYMENU;
                shidi.hDlg = hDlg;
                SHInitDialog(&shidi);
            }
#endif // SHELL_AYGSHELL

            return TRUE;

        case WM_COMMAND:
#ifdef SHELL_AYGSHELL
            if (LOWORD(wParam) == IDOK)
#endif
            {
                EndDialog(hDlg, LOWORD(wParam));
                return TRUE;
            }
            break;

        case WM_CLOSE:
            EndDialog(hDlg, message);
            return TRUE;

#ifdef _DEVICE_RESOLUTION_AWARE
        case WM_SIZE:
            {
            DRA::RelayoutDialog(
                  g_hInst,
                  hDlg,
                  DRA::GetDisplayMode() != DRA::Portrait ? MAKEINTRESOURCE(IDD_ABOUTBO
X_WIDE) : MAKEINTRESOURCE(IDD_ABOUTBOX));
            }
            break;
```

```
#endif
    }
    return FALSE;
}
#endif // !WIN32_PLATFORM_WFSP
```

参照

処理手順

チュートリアル : 簡単な Windows フォームの作成

関連項目

[全般] ([オプション] ダイアログ ボックス - [デバイス ツール])

ツールボックス

その他の技術情報

スマート デバイスのチュートリアル

Visual C++ デバイス プロジェクトの開発

方法：プライマリ プロジェクト出力のリモート パスを指定する

デバイス プロジェクトのプロパティ ページを使用して、配置用のリモート ディレクトリを設定できます。リモート ディレクトリでは、アプリケーションの出力が配置されるデバイス ディレクトリが指定されます。

リモート ディレクトリは、既定では %CSIDL_PROGRAM_FILES%\<プロジェクト名> に設定されます。最初の部分は **CSIDL** 値で、2 つ目の部分はプロジェクト名です。**CSIDL** 値では、システムに依存しない方法でリモート ディレクトリ フォルダが指定されます。これらの値は、同じ目的の環境変数の使用に代わるものです。リモート ディレクトリ パラメータの有効な **CSIDL** 値を次の表に示します。値をカプセル化するには、% 記号を使用します。

CSIDL	値	説明
CSIDL_DESKTOP	0x0000	Smartphone ではサポートされません。
CSIDL_FAVORITES	0x0006	ユーザーのお気に入りの項目を格納する共通リポジトリの役割をするファイル システム ディレクトリ。
CSIDL_FONTS	0x0014	フォントが含まれる仮想フォルダ。
CSIDL_PERSONAL	0x0005	ドキュメントの共通リポジトリの役割をするファイル システム ディレクトリ。
CSIDL_PROGRAM_FILES	0x0026	プログラム ファイル フォルダ。
CSIDL_PROGRAMS	0x0002	ユーザーのプログラム グループが含まれるファイル システム ディレクトリ。プログラム グループもファイル システム ディレクトリです。
CSIDL_STARTUP	0x0007	ユーザーのスタートアップ プログラム グループに対応するファイル システム ディレクトリ。デバイスの電源がオンになると、システムによりこれらのプログラムが起動されます。
CSIDL_WINDOWS	0x0024	Windows フォルダ。

プライマリ プロジェクト出力のリモート パスを指定するには

1. ソリューション エクスプローラで、[<プロジェクト名>] を右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [構成プロパティ] ノードを展開し、[配置] をクリックします。
3. 一番右のペインで、プロジェクトのリモート ディレクトリ プロパティを設定します。

参照

その他の技術情報

Visual C++ デバイス プロジェクトの開発

方法 : 既定のデバイスを変更する (ネイティブ プロジェクト)

ネイティブ プロジェクトの既定のターゲット デバイスを変更するには、以下の手順を使用します。

ネイティブ プロジェクト用に新しく既定のターゲット デバイスを選択するには

1. ソリューション エクスプローラで、[<プロジェクト名>] を右クリックします。
2. ショートカット メニューの [プロパティ] をクリックします。
3. [プロパティ] ダイアログ ボックスで、[構成プロパティ] を展開し、[配置] をクリックします。
4. ドロップダウン リストで、配置デバイス プロパティの値を変更して、新しい既定のターゲット デバイスを選択します。

参照

処理手順

方法 : 既定のデバイスを変更する (マネージ プロジェクト)

関連項目

[配置] ダイアログ ボックス (デバイス)

その他の技術情報

Visual C++ デバイス プロジェクトの開発

Visual C++ デバイス プロジェクトのビルドとデバッグ

デバイス プロジェクトのビルドおよびデバッグは、デスクトップ プロジェクトのビルドおよびデバッグと少し異なります。ここでは、これらの相異点について説明します。

ビルドおよびデバッグの手法を次に示します。

- 既定では、デバイス プロジェクトのスレッド処理モデルはフリーです。ただし、Windows CE では、CE OS イメージの作成時に DCOM オプションを選択しない場合、COM マーシャリングおよび関連する定義が十分にサポートされません。そのため、特定の CE プラットフォームでは、DCOM のサポートおよびシングルスレッドとマルチスレッドの定義に関する警告がコンパイラで生成される場合があります。この警告は、独自のコードでスレッドおよび同期を処理するように通知します。たとえば、ATL デバイス プロジェクトをコンパイルする場合、コンパイラは、**_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA** の定義に関する警告メッセージを表示することがあります。また、これは、Windows Mobile プラットフォーム上の ATL COM オブジェクトだけでなく、COM オブジェクトの作成、Web サービスの使用などのシナリオにも当てはまります。シングルスレッド オブジェクトの場合は、メイン ヘッダー ファイルで、このフラグを `#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA` のように定義できます。コードでマルチスレッドを処理する場合は、この警告を問題なく無視できます。Windows CE 上の COM、DCOM、およびスレッド処理の各モデルの詳細については、Windows CE 5.0 ドキュメントの「COM Threads and Processes」および「Component Services (COM and DCOM)」を参照してください。
- 既定では、静的なリンクを含むネイティブ デバイス アプリケーションが作成されます。動的なリンクに切り替えると、[追加ファイル] プロジェクト プロパティに次のランタイム DLL を追加できます。
    - MFC アプリケーションのリリース ビルド構成 : [追加ファイル] プロジェクト プロパティに msvcr80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;atl80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;MFC80U.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0; というランタイム DLL を追加します。
    - MFC アプリケーションのデバッグ ビルド構成 : [追加ファイル] プロジェクト プロパティに msvcr80d.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;msvcr80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;atl80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;MFC80Ud.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0; というランタイム DLL を追加します。
    - ATL アプリケーションのリリース ビルド構成 : [追加ファイル] プロジェクト プロパティに msvcr80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;atl80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0; というランタイム DLL を追加します。
    - ATL アプリケーションのデバッグ ビルド構成 : [追加ファイル] プロジェクト プロパティに msvcr80d.dll|$(BINDIR)\$(INSTRUCTIONSET)\%CSIDL_PROGRAM_FILES%\<projectname>|0;msvcr80.dll|$(BINDIR)\$(INSTRUCTIONSET)\%CSIDL_PROGRAM_FILES%\<projectname>|0;atl80.dll|$(BINDIR)\$(INSTRUCTIONSET)\%CSIDL_PROGRAM_FILES%\<projectname>|0; というランタイム DLL を追加します。

      Windows CE DLL の読み込みの詳細については、「LoadLibrary」を参照してください。

メモ :
異なるディレクトリから同じ名前の複数の DLL (たとえば、\Windows\aDLL.dll と **\Program** Files\aDLL.dll) を同時に読み込むと、予測できない結果が生じる場合があります。一度に 1 つの DLL のコピーを読み込むか、最初に読み込まれた DLL が呼び出されるようにすることをお勧めします。

その他の考慮事項を次に示します。

- MFC 8.0 # に移植する場合は、_WIN32_WCE_PSPC を定義します。既定では、このフラグが MFC8.0 で定義されていません。
- [アーキテクチャ用コンパイル] ボックスの一覧で Pocket PC 2003 または Smartphone 2003 を対象とする場合、[ARM4T] オプションはサポートされません。この一覧を表示するには、[<プロジェクト名> プロパティ ページ] ダイアログ ボックスで、[構成プロパティ] をクリックし、[C/C++] をクリックします。その後、[詳細] をクリックし、[アーキテクチャ用コンパイル] を選択します。
- GAPI 関数は、C++ で使用できますが、C では使用できません。コードに gx.h を含めることができるのは、C++ を使用している場合だけ

です。コード内で GAPI を使用している場合は、**/TC** コンパイラ オプションでコンパイルしないでください。

- MFC 8.0 for Devices では、**CommandBar** の作成および挿入を制御します。**CDialog::m_pWndEmptyCB** メンバはサポートされなくなりました。このメンバは、ダイアログ ボックス用に作成された空の CCommandBar クラス (Pocket PC では、**MenuBar** とも呼ばれています) を指すために使用されていました。そのため、そのメンバを参照し、独自の **MenuBar** を挿入できました。
- **Checked::_strlwr_s**、**_strlwr_s_l**、**_mbslwr_s**, **_mbslwr_s_l**、**_wcslwr_s**、**_wcslwr_s_l**、**Checked::tcslwr_s**、および **Checked::gcvt_s** は Windows CE プラットフォーム用に用意されており、基になる CRT メソッドは、将来のリリースでセキュリティを最大限に高めるためにセキュリティで保護されています。
- **_WIN32_WCE_PSPC** フラグは定義されなくなったため、代替手段として **_WIN32_WCE_PSPC WIN32_PLATFORM_PSPC** フラグを使用できます。
- STL アプリケーションを移植する場合は、`#include <deque.h>` ではなく <deque> を含めます。
- 開発環境のマルチプラットフォームという性質により、デバイス プロジェクトで MFC または ATL の ActiveX オブジェクトをホストする場合、ダイアログ ボックスのリソースでは、同等のデスクトップ ActiveX プロジェクトで同等のコントロールを作成および登録し、リソース エディタでデバイスのダイアログ ボックス テンプレートに追加して正しく実行できるようにする必要があります。ActiveX コントロールのデスクトップ バージョンとデバイス バージョンには、背景色など、同じ GUID およびデザイン時プロパティを指定する必要があります。

このセクションの内容

Visual C++ デバイス プロジェクトのデバッグと配置の設定

Visual C++ デバイス プロジェクトに固有のデバッグ プロパティおよび配置プロパティについて説明します。

参照

関連項目

[デバッグ] ([<プロジェクト名> プロパティ ページ] ダイアログ ボックス - [構成プロパティ]) (デバイス)

概念

Visual C++ デバイス プロジェクトのデバッグと配置の設定

その他の技術情報

デバイス プロジェクトのデバッグ

Visual C++ デバイス プロジェクトの作成と移植

Visual C++ デバイス プロジェクトの開発

Visual C++ デバイス プロジェクトのデバッグと配置の設定

Visual C++ デバイス プロジェクトでは、Visual C++ デスクトップ プロジェクトでサポートされていないプロパティをサポートしています。これらのプロパティは、プロジェクトのプロパティ ページで設定します。

固有のプロパティ ページ

Visual C++ デバイス プロジェクト固有のプロパティ ページを次に示します。

- [デバッグ] ([<プロジェクト名> プロパティ ページ] ダイアログ ボックス - [構成プロパティ]) (デバイス)
- デバイス アプリケーションのパッケージ化をサポートする IDE 機能
- [デバッグ] ペイン (プロジェクト デザイナ)
- プロパティ ページの詳細については、「Visual C++ デバイス プロジェクトのビルドとデバッグ」および「プロパティ ページ (C++)」を参照してください。

Visual C++ デバイス プロジェクトのプロパティへのアクセス

デバイス プロジェクトのプロパティ ページにアクセスする方法を次に示します。

- ソリューション エクスプローラで、プロジェクトを右クリックし、[プロパティ] をクリックします。
- ソリューション エクスプローラで、プロジェクト名をクリックし、[プロジェクト] メニューの [プロパティ] をクリックします。

> **メモ :**
>
> 配置先が DCOM をサポートしている Windows CE 5.0 SDK の場合など、ATL .exe プロジェクトをデバッグする場合は、[プロジェクトのプロパティ] の [出力の登録] を [はい] に設定し、[コマンド] プロパティに [/RegServer] を追加します。これにより、デバッグの開始時に .exe が登録されます。詳細については、「[配置] ([<プロジェクト名> プロパティ ページ] ダイアログ ボックス - [構成プロパティ]) (デバイス)」を参照してください。

> **メモ :**
>
> ネイティブ デバイス プロジェクトを配置した場合、依存プロジェクトは、そのネイティブ デバイス プロジェクトと共に自動的に配置されず、別途配置が必要になる場合があります。

参照

その他の技術情報

Visual C++ デバイス プロジェクトのビルドとデバッグ

デバイス プロジェクトのデバッグ

スマート デバイスで実行するプロジェクトをデバッグする方法は、デスクトップで実行されているプロジェクトのデバッグ方法と似ています。明確な違いの 1 つは、デバイス プロジェクトのデバッグは、開発コンピュータとは別のプロセッサで実行される点です。そのため、デバイスのデバッグには接続の問題が生じます。

このセクションでは、デスクトップ プロジェクトのデバッグとの差異を概説し、デバイスでデバッグするときに役立つヒントを説明します。

セキュリティに関するメモ：
機密情報や機微な情報が保存されているデバイスまたはエミュレータ上でアプリケーションをデバッグする場合、セキュリティ上のリスクが伴う場合があります。

このセクションの内容

デバイスのデバッグとデスクトップのデバッグとの違い

デバイスのデバッグ方法とデスクトップのデバッグ方法の差異を説明します。

方法 : マネージ デバイスのプロセスに接続する

実行中のマネージ プロセスに接続する方法について説明します。

方法 : デバイスのレジストリ設定を変更する

リモート レジストリ エディタを使用してデバイスのレジストリ設定を変更する方法について説明します。

チュートリアル : マネージ コードとネイティブ コードの両方を含むソリューションのデバッグ

ネイティブ デバッガおよびマネージ デバッガを交互に起動する手順を説明します。

参照

その他の技術情報

スマート デバイス開発

アプリケーションのデバッグとプロファイリング

Mobile Developer Center

デバイスのデバッグとデスクトップのデバッグとの違い

デバイスのデバッガは、デスクトップのデバッガとほぼ同じ機能がサポートされます。ただし、次の例外があります。

エディット コンティニュはサポート外

デバイスのデバッガでは、中断モード時にソースの編集と継続を行う機能はサポートされません。デバッグ中にコードを変更するには、デバッグを停止し、コードを編集してから変更したソースで再起動する必要があります。中断モードでコードを変更しようとすると、警告が表示されます。

関数の評価はネイティブ デバッガでサポート外

ネイティブ デバイス デバッガでは、関数の評価はサポートされません。関数を含む式を入力しても、関数の評価結果は返されません。

マネージ デバイス デバッガでは、関数の評価がサポートされています。

相互運用デバッグの制約

1 インスタンスのデバッガ内で、ネイティブ コードとマネージ コードをデバッグすることはできません。

ネイティブ コードとマネージ コードの両方 (または pInvoke を使用しているマネージ コード) を含んでいるアプリケーションをデバッグするには、コードのステップ実行を開始する各セクション内にブレークポイントを設定します。その後、特定のセクション (たとえばマネージ セクション) に必要なデバッガをアタッチします。他のデバッガが必要になったときは、そのデバッガをデタッチして別のデバッガをアタッチします。プログラムをステップ実行するには、必要に応じて頻繁にデタッチとアタッチの手順を繰り返してください。詳細については、「チュートリアル : マネージ コードとネイティブ コードの両方を含むソリューションのデバッグ」を参照してください。

同じプロセスで同時にデバッグ インスタンスを 2 つ使用する操作は、現在サポートされていません。

属性ベースのデバッグはサポート外

.NET Compact Framework では、属性ベースのデバッグは現在サポートされていません。そのため、ビジュアライザの属性を定義する機能などは、デバイス デバッガで使用できません。

デスクトップのデバッグはサポート外

デバイス デバッガを使用して、デスクトップ用に記述されたアプリケーションをデバッグすることはできません。デスクトップ デバッガを使用してください。

カーネルのデバッグはサポート外

カーネルのデバッグにデバイス デバッガは使用できません。

"マイ コードのみ" デバッグはサポート外

"マイ コードのみ" デバッグは使用できません。

ランタイム デバッガ (Cordbg.exe) の追加項目

ランタイム デバッガを使用すると、ツール販売元およびアプリケーション開発者は、.NET Framework 共通言語ランタイム (CLR) を対象にしたプログラムのバグを発見および修正しやすくなります。デバイス プロジェクトでは、ランタイム デバッガに新しいコマンドと新しいモード引数を追加します。次の表は、新しいコマンドとモード引数の構文です (Cordbg.exe セッション内)。

詳細と構文については、「ランタイム デバッガ (Cordbg.exe)」を参照してください。

コマンド	説明
m[ode] EmbeddedCLR {0|1}	EmbeddedCLR は、デバッガをデバイス プロジェクト用に設定するモード引数です。この設定を制御にするとき、*on* にするには 1 を、*off* にするには 0 を指定します。

conn[ect] *machine_name port*	**リモートの埋め込み** CLR **デバイスへ接続します。** **パラメータ** : *Machine_name* **必ず指定します。リモート コンピュータの名前または** IP **アドレス。** *Port* **必ず指定します。リモート コンピュータへの接続に使用するポート。**

接続の問題

デバッガの実行中にデバイスの電源をオフにすると、デバッガは接続障害により閉じてしまいます。接続障害の原因は、デバイスのバックグラウンドでアプリケーションがまだ実行中であるためです。Pocket PC **の** [X] **ボタンは、アプリケーションを閉じずに最小化するスマート最小化機能です。アプリケーションはバックグラウンドで実行されます。**

Pocket PC **のバックグラウンドで実行されているアプリケーションを適切に閉じるには、**[Start] **ボタンをクリックし、**[Settings] **をタップして、**[System] **タブをタップします。次に、**[Memory] **をタップします。**[Running Programs] **タブで、閉じるアプリケーションをタップし、**[Stop] **をタップします。**

参照

その他の技術情報

デバッガのロードマップ
デバイス プロジェクトのデバッグ

方法：マネージ デバイスのプロセスに接続する

デバイス上のプロセスには、デスクトップの場合とほとんど同じ方法で接続できます。ただし、プロセスがデバッガなしの状態で既に実行されていて、マネージ デバッグを有効にするために、デバイスでレジストリ キーを設定する必要がある場合は除きます。このキーの設定は、手動で変更するまで変更されません。またはエミュレータの場合、エミュレータで設定を保存しない状態でエミュレータを終了するまで変更されません。

メモ：
デバイスのデバッグ キーを設定すると、パフォーマンスが低下します。デバッグしていないときは、キーをリセットします。

2 つのデバッガに接続しようとしたとき、またはデバイスのレジストリ キーが設定されていない状態でマネージ デバッガに接続しようとしたときは、エラー メッセージが表示されます。

プロセスの起動には、ファイル エクスプローラ、コマンド ラインなどいくつかの方法があります。次の手順では、[デバッグ] メニューからプロセスを起動します。また、マネージ デバッガがない状態でもプロセスを起動して、後から接続することもできます。

Platform Builder から生成した Windows CE プラットフォームを対象にしている場合、toolhelp.dll ライブラリを用意して、[選択可能なプロセス] ペインを設定する必要があります。このライブラリは、Windows Mobile SDK に含まれます。

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

マネージ プロセスのデバッグ

マネージ プロセスをデバッグするには

- [デバッグ] メニューの [開始] をクリックします。

メモ：
[デバッグ] メニューで起動したプロセスから接続を解除した場合、再接続するには、プロセスの実行中に接続する以下の手順を実行する必要があります。つまり、デバイスのレジストリ キーを設定する必要があります。

実行中のマネージ プロセスへの接続

実行中のプロセスに接続する場合 (たとえば、[デバッグなしで開始] をクリックして実行中のマネージ プロセスに接続する場合)、プロセスを開始する前に、さらに [プロセスにアタッチ] ダイアログ ボックスを使用して接続する前に、デバイスのレジストリ キーを設定する必要があります。次に詳細なプロセスの手順を示します。

デバイスのレジストリ キーを設定して実行中のプロセスに接続するには

1. Windows の [スタート] ボタンをクリックし、[すべてのプログラム] をポイントします。次に [Microsoft Visual Studio 2005] をポイントし、[Visual Studio Tools] をポイントして、[リモート レジストリ エディタ] をクリックします。
2. リモート レジストリ エディタを使用して、デバイスに接続します。
3. HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETCompactFramework\Managed Debugger キーを表示します。ない場合は作成します。
4. `AttachEnabled` という DWORD 値を設定または作成します。
5. この値のデータを 1 に設定します。

メモ：
デバイスのデバッグ キーを設定すると、パフォーマンスが大幅に低下します。デバッグ中ではない場合、マネージ接続は無効にします。データ値を 0 にリセットするか、`AttachEnabled` 値を削除します。

6. リモート レジストリ エディタを閉じます。

   これで、マネージ接続が有効になります。また、デバッガなしでプロセスを起動して、[プロセスにアタッチ] ダイアログ ボックスを使用してプロセスに接続できます。

プロセスが実行されているときにマネージ プロセスに接続するには

1. 前述の手順でレジストリ キーを設定した後に、デバッガなしでプロセスを開始します。
2. [ツール] メニューの [プロセスにアタッチ] をクリックします。
3. [トランスポート] ボックスで、[スマート デバイス] をクリックします。
4. [修飾子] ボックスの [参照] をクリックします。

メモ :
現在のセッションから最近使用されたデバイスについて、[修飾子] ボックスが表示されます。

5. [デバイスへの接続] ダイアログ ボックスで、プラットフォームとデバイスを選択し、[接続] をクリックします。
6. [選択可能なプロセス] ペインで、接続するプロセスを 1 つ以上選択し、[アタッチ] をクリックします。

メモ :
コードの種類は、使用できる場合は既定で自動的に [マネージ (.NET Compact Framework)] に設定されます。それ以外の場合は、[ネイティブ (スマート デバイス)] に設定されます。既定の設定をオーバーライドするには、[選択] をクリックし、[コードの種類の選択] ダイアログ ボックスを開きます。両方を選択できない点に注意してください。

メモ :
相互運用のデバッグはサポートされていません。つまり、マネージ コードとネイティブ コードの両方を同時にデバッグすることはできません。

プロセスの接続解除または終了

プロセスを接続解除または終了するには

1. [デバッグ] メニューの [ウィンドウ] をポイントし、[プロセス] をクリックします。
2. [プロセス] ウィンドウで、接続解除または終了するプロセスを右クリックします。
3. ショートカット メニューの [プロセスの終了] または [プロセスのデタッチ] をクリックします。

メモ :
このショートカット メニューから、[プロセスにアタッチ] ダイアログ ボックスを開き直すこともできます。

[選択可能なプロセス] ペインの設定

Windows CE プロジェクトの [選択可能なプロセス] ペインを設定するには

- Windows CE の OS イメージに toolhelp.dll を含めます。

  または

  手動で toolhelp.dll を対象デバイスにコピーします。

参照

処理手順

チュートリアル : マネージ コードとネイティブ コードの両方を含むソリューションのデバッグ

その他の技術情報

デバイス プロジェクトのデバッグ

Visual C++ デバイス プロジェクトのビルドとデバッグ

方法：デバイスのレジストリ設定を変更する

リモート レジストリ エディタを使用して、デバイスのレジストリ設定を変更します。

レジストリの設定を変更するには

1. Windows の [スタート] ボタンをクリックし、[すべてのプログラム] をポイントします。次に [Visual Studio 2005] をポイントし、[Visual Studio Remote Tools] をポイントして、[リモート レジストリ エディタ] をクリックします。
2. [Windows CE デバイスの選択] ウィンドウで、レジストリを編集する対象デバイスをクリックします。

   [レジストリ エディタ] の左側のペインに、対象デバイスのレジストリを表すノードが表示されます。
3. ノードを展開して変更します。

参照

その他の技術情報

デバイス プロジェクトのデバッグ

チュートリアル : マネージ コードとネイティブ コードの両方を含むソリューションのデバッグ

このチュートリアルでは、マネージ .NET Compact Framework コンポーネントとネイティブ コンポーネントの両方を含むソリューションをデバッグする手順を説明します。Visual Studio 2005 では、このようなデバイス アプリケーションの相互運用デバッグがサポートされていません。つまり、ネイティブ デバッガとマネージ デバッガを同時にアタッチしておくことはできません。

ネイティブ要素とマネージ要素の両方を組み込んだソリューションをデバッグする場合は、特定のセクション (たとえば、マネージ セクション) に必要なデバッガをアタッチし、その後、他のデバッガが必要になったときに、そのデバッガをデタッチして他のデバッガをアタッチします。プログラムをステップ実行するには、必要に応じて頻繁にデタッチとアタッチの手順を繰り返してください。

メモ :
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

このチュートリアルは、Visual C# 開発設定を使用して記述されています。このチュートリアルは、次のセクションで構成されています。

- マネージ デバッガのアタッチの有効化
- アプリケーションの起動
- ネイティブ コードでのブレークポイントの設定
- ネイティブ デバッガによるアタッチ
- ネイティブ ブレークポイントまでの実行
- マネージ デバッガによるアタッチ
- マネージ コードでのブレークポイントの設定
- マネージ ブレークポイントまでの実行
- まとめ

前提条件

このチュートリアルは、「チュートリアル : Hello World: スマート デバイスの COM 相互運用の例」という別のチュートリアルでビルドしたソリューションに基づいています。この Hello World チュートリアルが正常にビルドおよび実行されていることを確認してください。

マネージ デバッガのアタッチの有効化

デバイス (エミュレータを含む) では、既定で、マネージ デバッガは、既に実行中のプロセスにアタッチできません。既に実行中のプロセスにマネージ デバッガをアタッチするのは、マネージ コードとネイティブ コードの両方が含まれているデバイス ソリューションを操作する場合です。

最初の手順では、既に実行中のプロセスにマネージ デバッガがアタッチできるようにデバイスを設定します。この操作を行うには、デバイスのレジストリ キーを設定します。

メモ :
キーの設定は、既に実行中のマネージ プロセスへのアタッチのみに影響します。[デバッグ開始] (F5 キー) を使用してプロジェクトを起動する場合には影響しません。ただし、[デバッグ開始] の後にデタッチした場合に、再度アタッチしてデバッグを開始するには、キーの設定が必要になります。

実行中のプロセスにマネージ デバッガがアタッチできるようにするには

1. Windows の [スタート] ボタンをクリックし、[すべてのプログラム] をポイントします。次に [Visual Studio 2005] をポイントし、[Visual Studio Remote Tools] をポイントして、[リモート レジストリ エディタ] をクリックします。
2. [Windows CE デバイスの選択] ウィンドウで、[Pocket PC 2003] を展開し、[Pocket PC 2003 SE エミュレータ] をクリックします。このチュートリアルでは、これが対象のデバイスになります。

3. [OK] をクリックします。

   [デバイスに接続中] 進行状況ウィンドウが開き、デバイス エミュレータおよび Windows CE リモート レジストリ エディタが開きます。

4. レジストリ エディタで、[Pocket PC 2003 SE エミュレータ] を展開し、HKEY_LOCAL_MACHINE\SOFTWARE\Microsot\.NETCompactFramework\Managed Debugger キーを作成します。

   キーを作成するには、[.NETCompactFramework] を右クリックし、[新規作成] をポイントして、[キー] をクリックします。

   "Managed" と "Debugger" の間にはスペースがあることに注意してください。

5. `AttachEnabled` という名前の DWORD を作成します。

   DWORD を作成するには、[Managed Debugger] を右クリックし、[新規作成] をポイントして、[DWORD 値] をクリックします。

6. [名前] に *AttachEnabled* を、[値] に *1* を設定します。

メモ：
このデバイスのデバッグ キーを設定すると、パフォーマンスが大幅に低下します。デバッグをしていない場合は、データ値を 0 にリセットしてこの機能を無効にします。

7. 残りの手順でレジストリの設定を保持するために、デバイス エミュレータを開いたままにしておきます。レジストリ エディタは閉じても問題ありません。

アプリケーションの起動

次の手順では、InteropSolution アプリケーションを起動します。

アプリケーションを起動するには

1. 「チュートリアル : Hello World: スマート デバイスの COM 相互運用の例」で作成したソリューションを開きます。

   ツール バーの [ターゲット デバイス] ボックスに [Pocket PC 2003 SE エミュレータ] が表示されていることを確認します。

2. Visual Studio で、[デバッグ] メニューの [デバッグ開始] をクリックするか、F5 キーを押します。

   この手順により、ユーザーが操作する必要なく、HelloCOMObject というネイティブ プロジェクトがエミュレータに配置されます。

3. [SayHello の配置] ダイアログ ボックスが表示されたら、[Pocket PC 2003 SE エミュレータ] を選択し、[配置] をクリックします。

   この手順により、マネージ プロジェクトが配置されます。

   エミュレータでアプリケーションが開きます。まだボタンをクリックしないでください。

ネイティブ コードでのブレークポイントの設定

次の手順では、ネイティブ デバッガをアタッチできるようにネイティブ コードにブレークポイントを設定します。

ネイティブ コードにブレークポイントを設定するには

1. ソリューション エクスプローラで Hello.cpp を右クリックし、ショートカット メニューの [コードの表示] をクリックします。

2. `*text` で始まる行で、コード エディタの左の余白をクリックし、ブレークポイントを挿入します。

   このブレークポイント シンボルは、感嘆符付きの白丸として表示され、ブレークポイントが現在解決できないことを示します。これは、この時点で適切なシンボルおよびソースが欠けているためです。

3. Visual Studio で、[デバッグ] メニューの [ウィンドウ] をポイントし、[モジュール] をクリックします。

   [モジュール] ウィンドウでは、これまでに読み込まれたすべてのモジュール (SayHello.exe マネージ アプリケーションなど) が表示されます。アプリケーション内のボタンをクリックしていないため、ネイティブな HelloCOMObject.dll はまだ読み込まれていません。

ネイティブ デバッガによるアタッチ

次の手順では、ネイティブ デバッガでアタッチできるようにマネージ デバッガをデタッチします。デバイス プロジェクトでは、両方のデバッガを同時にアタッチできないことを思い出してください。マネージ デバッガからネイティブ デバッガへの切り替えが必要な場合に使用する手順を次に示します。

ネイティブ デバッガでアタッチするには

1. Visual Studio で、[デバッグ] メニューの [すべてデタッチ] をクリックします。

   この手順により、マネージ デバッガはデタッチされますが、アプリケーションは引き続き実行できます。

2. [デバッグ] メニューの [プロセスにアタッチ] をクリックします。

3. [トランスポート] ボックスで、[スマート デバイス] を選択します。

4. [修飾子] ボックスを設定するには、[参照] をクリックします。

5. [デバイスへの接続] ダイアログ ボックスで、[Pocket PC 2003 SE エミュレータ] を選択し、[接続] をクリックします。

6. [アタッチ先] ボックスを設定するには、[選択] をクリックします。

7. [コードの種類の選択] ダイアログ ボックスで、[次のコードの種類をデバッグする] を選択します。[マネージ] チェック ボックスをオフにし、[ネイティブ] チェック ボックスをオンにして、[OK] をクリックします。

8. [選択可能なプロセス] ボックスで、[SayHello.exe] を選択し、[アタッチ] をクリックします。

   これで、ネイティブ デバッガでアタッチされます。

ネイティブ ブレークポイントまでの実行

これで、ネイティブ コードに設定したブレークポイントまで進む準備ができました。[モジュール] ウィンドウを再度参照すると、ネイティブ モジュールが存在することがわかります。ただし、まだ `button1` をクリックしていないため、HelloCOMObject.dll は読み込まれていません。

メモ：
以前にこのチュートリアルを実行したことがある場合は、デバッグ シンボルが既に読み込まれている可能性があるため、この手順を省略できます。まだ実行していない場合は、次のセクションの手順を実行して、デバッグ シンボルを読み込みます。

ネイティブ ブレークポイントまで実行を進めるには

1. デバイス エミュレータのフォームで、`button1` をクリックします。

   "Hello World!" メッセージがフォーム上に表示され、hellocomobject.dll が [モジュール] ウィンドウに表示されます。

   hellocomobject.dll の [シンボルの状態] 列に [シンボルが読み込まれていません。] が表示されている場合は、次の手順を実行します。

   a. hellocomobject.dll を右クリックし、ショートカット メニューの [シンボルの読み込み] をクリックします。

   b. [シンボル検索] ダイアログ ボックスで、InteropSolution\HelloCOMObject\Pocket PC 2003 (ARMV4)\Debug\HelloCOMObject.pdb に移動します。

   c. [開く] をクリックします。

      [シンボルの状態] 列が [シンボルが読み込まれていません。] から [読み込まれたシンボル] に変更され、ブレークポイント インジケータはブレークポイントが解決されたことを示します。

2. デバイス エミュレータのフォーム上で、[Hello World!] ウィンドウの [OK] をクリックし、再度 `button1` をクリックします。

   ブレークポイント インジケータは、実行がブレークポイントで停止したことを示します。

3. [デバッグ] メニューの [ステップ イン] をクリックするか、F11 キーを押します。

   実行が次の行に移動することに注意してください。これは、ソリューションのネイティブな部分をステップ実行できることを示します。

マネージ デバッガによるアタッチ

次の手順では、マネージ デバッガでアタッチできるようにネイティブ デバッガをデタッチします。デバイス プロジェクトでは、両方のデバッガを同時にアタッチできないことを思い出してください。ネイティブ デバッガからマネージ デバッガへの切り替えが必要な場合に使用する手順を次に示します。

マネージ デバッガでアタッチするには

1. Visual Studio で、[デバッグ] メニューの [すべてデタッチ] をクリックします。

   この手順により、ネイティブ デバッガはデタッチされますが、アプリケーションは引き続き実行されます。

2. [デバッグ] メニューの [プロセスにアタッチ] をクリックし、[トランスポート] ボックスに [スマート デバイス] が表示されていることを確認します。

3. [修飾子] を設定します。この設定を行うには、[選択] をクリックし、[Pocket PC 2003 SE エミュレータ] を選択して、[接続] をクリックしま

す。

4. [アタッチ先] ボックスを設定するには、[選択] を選択し、[次のコードの種類をデバッグする] を選択します。[マネージ] チェック ボックスをオンにし、[ネイティブ] チェック ボックスをオフにして、[OK] をクリックします。

   マネージ デバッグとネイティブ デバッグに互換性のないことを示すメッセージが表示されたら、[OK] をクリックします。

5. [選択可能なプロセス] ボックスで、[SayHello.exe] を選択し、[アタッチ] をクリックします。

   ここで、マネージ デバッガがアタッチされます。

マネージ コードでのブレークポイントの設定

次の手順では、マネージ デバッガをアタッチできるようにマネージ コードにブレークポイントを設定します。

マネージ コードにブレークポイントを設定するには

1. ソリューション エクスプローラで Form1.cs を右クリックし、ショートカット メニューの [コードの表示] をクリックします。
2. `string text;` という行にブレークポイントを挿入します。

マネージ ブレークポイントまでの実行

これで、マネージ コードに設定したブレークポイントまで進む準備ができました。

マネージ ブレークポイントまで実行を進めるには

- デバイス エミュレータで、`button1` をクリックします。

  ブレークポイントで実行が中断されます。

まとめ

パフォーマンス上の理由により、既に実行中のプロセスにマネージ デバッガをアタッチする必要がなくなったら、デバイスのレジストリ キーを 0 にリセットすることを忘れないでください。

参照

処理手順

方法 : マネージ デバイスのプロセスに接続する

方法 : デバイスのレジストリ設定を変更する

その他の技術情報

デバイス プロジェクトのデバッグ

Visual Studio でのデバッグ

配置用のデバイス ソリューションのパッケージ化

デバイス アプリケーションをエンド ユーザーに配布するには、CAB ファイルを作成する必要があります。

このセクションの内容

デバイス ソリューションをパッケージ化する方法の概要

エンド ユーザーへの配布用にスマート デバイス アプリケーションをパッケージ化するために必要なプロセスについて説明します。

デバイス アプリケーションのパッケージ化をサポートする IDE 機能

パッケージ化に使用する Visual Studio 環境の要素に加え、対象のプラットフォームによって異なるパッケージ化の相異点について説明します。

チュートリアル : 配置用のスマート デバイス ソリューションのパッケージ化

アプリケーションとそのリソースをパッケージ化する詳細な手順を示します。

関連するセクション

スマート デバイス開発

Mobile Developer Center

デバイス ソリューションをパッケージ化する方法の概要

配置は、アプリケーションまたはコンポーネントを、対象デバイスまたは実行予定のデバイスに転送するプロセスです。ソリューションを配置する前に、CAB ファイルにパッケージ化する必要があります。CAB ファイルは、実行可能なアーカイブ ファイルの一種で、アプリケーション、DLL などの依存ファイル、リソース、ヘルプ ファイル、および任意のファイルを含めることができます。CAB プロジェクトをビルドすると、Microsoft Visual Studio 2005 で CAB の作成に使用する INF ファイルが生成されます。INF ファイルには、各ファイルのインストール先フォルダ、アプリケーションが動作する Windows CE のバージョン、アプリケーションのアンインストールが可能かどうか、などの情報が記述されています。また、CAB ファイルにカスタムのネイティブ DLL を含めて、カスタムのインストール手順を実行できます。たとえば、Windows CE または .NET Compact Framework のバージョン番号のチェック、他のコンポーネントが存在するかどうかの判断などです。CAB を対象デバイスにコピーし、その CAB をタップすると、インストール プロセスが開始されます。これは、"CAB の展開" と呼ばれます。

メモ :

Microsoft Visual Studio 2005 には、CAB ファイルをパッケージ化するツールがあります。ただし、CAB ファイルを対象デバイスに配置するツールは用意されていません。単純なシナリオの場合、ActiveSync 接続を使用して、デスクトップ コンピュータから CAB をデバイスにドラッグする方法があります。複雑なシナリオの場合、使用可能ないくつかのサード パーティ製の配置ソリューションがあります。詳細については、「Mobile and Embedded Application Developer Center」を参照してください。

Visual Studio 2005 を使用すると、ほとんどの場合、Visual Studio の統合開発環境 (IDE) で必要なあらゆるパッケージ化作業を直接実行できます。CAB ファイルを作成するには、まず既存のソリューションにスマート デバイス CAB プロジェクトを追加し、デスクトップのセットアップ プロジェクトと同じユーザー インターフェイスを使用して、ファイル、ショートカット、およびレジストリ エントリを追加します。セットアップ プロジェクトをビルドすると、CAB ファイルが作成されます。

Pocket PC アプリケーション用に作成した場合と Smartphone アプリケーション用に作成した場合とでは、CAB ファイルにいくつかの違いがあります。Windows Mobile Mobile 2003SE 以前を元にした Pocket PC では、圧縮された CAB ファイルまたは署名された CAB ファイルはサポートされません。Smartphone の CAB ファイルは必ず圧縮されます。また、EXE ファイルまたは DLL ファイルと、CAB ファイルのどちらも、デバイスにインストールする前に必ずデジタル署名が追加されます。

Visual Studio で CAB ファイルを作成した後の手順は、対象デバイスに転送することです。このとき、任意の一般的なファイル転送方法を使用します。たとえば、デバイスからの FTP または HTTP 要求を経由する方法、デスクトップの開発コンピュータから接続したデバイスのフォルダに Windows エクスプローラを使用して手動でコピーする方法、Smartphone に無線 (OTA) 転送する方法などが挙げられます。

参照

処理手順

チュートリアル : 配置用のスマート デバイス ソリューションのパッケージ化

概念

デバイス アプリケーションのパッケージ化をサポートする IDE 機能

その他の技術情報

スマート デバイス開発

配置でのファイルのインストール管理

デバイス アプリケーションのパッケージ化をサポートする IDE 機能

スマート デバイスに配置するソリューションをパッケージ化するには、デスクトップ ソリューションの場合と同様の Microsoft Visual Studio 2005 統合開発環境 (IDE) 機能を使用します。これらの機能の説明を次の表に示します。

機能	アクセス方法	解説
スマート デバイス CAB プロジェクト テンプレート	[ファイル] メニューの [追加] をポイントし、[新しいプロジェクト] をクリックして、[その他のプロジェクトの種類] をクリックします。次に、[セットアップとデプロイメント] をクリックします。	新しい CAB プロジェクトを既存のソリューションに追加するには、このアイコンをクリックします。このダイアログ ボックスの中で、スマート デバイス向けに有効なプロジェクトの種類はこれだけです。CAB プロジェクトの名前を選択し、[OK] をクリックすると、プロジェクトがソリューションに追加されて、ソリューション エクスプローラに表示されます。
ファイル システム エディタ	ソリューション エクスプローラで CAB プロジェクト名を右クリックし、[表示] をクリックして、[ファイル システム] をクリックします。	このエディタで、CAB に追加するファイルとインストール先のデバイス フォルダを指定します。
レジストリ エディタ	ソリューション エクスプローラで、CAB プロジェクト名を右クリックし、[表示] をクリックして、[レジストリ] をクリックします。	このエディタで、アプリケーションに必要な特別なレジストリ キーを指定します。
CAB プロジェクト用のプロパティ ウィンドウ	ソリューション エクスプローラで CAB プロジェクトを選択し、[表示] メニューの [プロパティ ウィンドウ] をクリックします。	このウィンドウでは、CE Setup DLL の名前 (名前がある場合)、アプリケーションのメーカー名、アプリケーションがサポートする Windows CE のバージョン (最低と最高) などのオプションを指定します。
<プロジェクト名> プロパティ ページ	ソリューション エクスプローラで、CAB プロジェクト名を右クリックし、[プロパティ] をクリックします。	このダイアログ ボックスで、構成 (*Debug* など)、出力ファイル名、およびセキュリティ証明書を指定します。

メモ :

このようにデスクトップのセットアップ プロジェクトと同じエディタが使用されるため、スマート デバイス CAB プロジェクトでは一部のオプションが無効にされることがあります。

場合によっては、Windows Mobile 2003 SE 以降など、特定のプラットフォームでのみ実行するように設計されたアプリケーションを開発することがあります。このとき、指定したサポート対象外のプラットフォームに CAB ファイルがインストールされないようにできます。ただし、INF ファイルを手動で編集し、コマンド ライン ツールで CAB を再パッケージ化する必要があります。Visual Studio を使用して CAB を再パッケージ化すると、手動の変更が上書きされます。

Pocket PC と Smartphone

Windows Mobile 2003 SE 以前の場合、Pocket PC 用 CAB ファイルと Smartphone 用 CAB ファイルの主な違いは、Pocket PC では圧縮された CAB ファイルや署名された CAB ファイルがサポートされていない点です。Smartphone の CAB ファイルは必ず圧縮されます。また .exe ファイルまたは .dll ファイルと、CAB ファイルのどちらも、デバイスにインストールする前に必ずデジタル署名が追加されます。詳細については、「デバイス プロジェクトにおけるセキュリティ」を参照してください。

ネイティブ アプリケーションとマネージ アプリケーション

アプリケーションのスマート デバイス CAB プロジェクトを C++ で記述する場合と、Visual C# または Visual Basic で記述する場合との唯一の違いは、ネイティブ アプリケーションでは、システムの依存関係 (**atl80.dll**、**mfc80U[d].dll**、または **msvcrt[d].dll**) を CAB プロジェクトに手動で追加する必要がある点です。マネージ アプリケーションの場合、.NET Compact Framework DLL を CAB ファイルに追加できません。バージョン 1.0 を対象にする場合、すべての DLL が Windows Mobile 2003 SE 以降のデバイスに配置されています。.NET Compact Framework のバージョン 2.0 を対象にする場合、デバイスにそのバージョンがインストール済みかどうかを確認する必要があります。この場合、対象デバイスで mscoree.dll のバージョンを確認する Windows CE Setup DLL を記述します。.NET Compact Framework のバージョン 2.0 がない場合、Microsoft Visual Studio 8\SmartDevices\SDK\CompactFramework\2.0\WindowsCE の下位に、Visual Studio と共に提供されている適切な CAB を配置できます。使用するアプリケーション CAB とは別にこの CAB を配置したり、アプリケーション CAB 内に埋め込んだりできます。

注意

MFC/ATL に動的にリンクされるネイティブ アプリケーションを再配布し、MFC/ATL ランタイム DLL をアプリケーション ディレクトリに配置したときは、アプリケーションがそのディレクトリ内の DLL にリンクしないことがあります。Windows CE では、同じファイル名を持つ 2 つの DLL が別々のパス上にある場合、そのファイル名を持つ最初の DLL だけが読み込まれます。以降の同一ファイル名の DLL は読み込まれません。その代わりに、アプリケーションは、以前に別のアプリケーションで読み込まれた同一ファイル名の DLL にリンクします。

アプリケーションをそのディレクトリ内の DLL に確実にリンクさせるためには、他のアプリケーションで同一ファイル名の DLL を使用しないでください。

スマート デバイスの配置とデスクトップの配置

デスクトップとデバイスの各セットアップ プロジェクトにアクセスするには、[新しいプロジェクト] ダイアログ ボックスで [その他のプロジェクトの種類] をクリックし、[セットアップとデプロイメント] をクリックします。デスクトップ アプリケーションを配置する場合、[セットアップ プロジェクト]、[マージ モジュール プロジェクト]、[CAB プロジェクト]、[Web セットアップ プロジェクト]、および [セットアップ ウィザード] を使用できます。これらのプロジェクトの種類のうち、デバイス アプリケーションに使用できるものはありません。ClickOnce 配置はスマート デバイスではサポートされません。Smartphone や Pocket PC など、任意の Windows CE ベースのデバイスに配置できる CAB ファイルを作成するには、スマート デバイス CAB プロジェクトを使用する必要があります。

参照

処理手順

チュートリアル : 配置用のスマート デバイス ソリューションのパッケージ化

概念

デバイス ソリューションをパッケージ化する方法の概要

チュートリアル : 配置用のスマート デバイス ソリューションのパッケージ化

このチュートリアルでは、エンド ユーザーのスマート デバイスに配置できるように、Visual Studio 2005 を使用してアプリケーションとリソースを CAB ファイルにパッケージ化する方法を説明します。

メモ :

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

このチュートリアルでは、Visual Basic 2005、Visual C# 2005、または Visual C++ 2005 で記述されたスマート デバイス ソリューションを使用することから始めます。詳細については、「チュートリアル : 簡単なアプリケーションの作成」を参照してください。

このチュートリアルでは、次の操作方法を説明します。

- CAB プロジェクトをソリューションに追加します。
- プロジェクト名を変更します。
- 出力パスを変更します。
- アプリケーションのプライマリ出力を使用して CAB ファイルを作成します。
- 必要に応じて依存関係を追加します。
- アプリケーションにショートカットを追加します。
- レジストリ エントリを編集します。

前提条件

既存のスマート デバイス ソリューション。このパッケージ化のチュートリアルを説明する目的から、「チュートリアル : デバイス対応の Windows フォーム アプリケーションの作成」に説明されているように単純なプロジェクトの作成と構築を想定しています。

メモ :

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

CAB プロジェクトの設定

ソリューションにスマート デバイス CAB プロジェクトを追加するには

1. 既存のスマート デバイス プロジェクトを開き、ソリューション エクスプローラが表示されることを確認します。
2. [ファイル] メニューの [追加] をポイントし、[新しいプロジェクト] をクリックします。

   [新しいプロジェクトの追加] ダイアログ ボックスが表示されます。
3. 左側の [プロジェクトの種類] ペインで、[その他のプロジェクトの種類] ノードを展開し、[セットアップとデプロイメント] をクリックします。
4. 右側の [テンプレート] ペインで、[スマート デバイス CAB プロジェクト] を選択します。

   スマート デバイスで有効なのは、この CAB プロジェクトの種類のみです。他のプロジェクトの種類は、デスクトップ ソリューションの場合にのみ有効です。
5. [プロジェクト名] ボックスに「**CABProject**」と入力し、[OK] をクリックします。

   CAB プロジェクトがソリューションに追加され、ソリューション エクスプローラに表示されます。2 つのペインのファイル システム エディタが表示されるようになります。

CAB プロジェクトのカスタマイズ

製品名や他のプロジェクト プロパティを変更するには

1. ソリューション エクスプローラで、[CABProject] を選択します (選択されていない場合)。
2. [表示] メニューの [プロパティ ウィンドウ] をクリックして、[プロパティ] ウィンドウを開きます。
3. プロパティ グリッドの [ProductName] フィールド値を「MyProduct」に変更します。

   [ProductName] プロパティ値では、フォルダに表示されるアプリケーション名および [アプリケーションの追加と削除] ダイアログ ボックスに表示されるアプリケーション名を指定します。

   - また、メーカーの名前を変更するとき、およびオペレーティング システムの許容バージョン (最低と最高) を指定するときにも、このウィンドウを使用できます。
   - 使用している Pocket PC アプリケーションで画面の方向に対応することを示すには、[OSVersionMin] プロパティを「4.21」に設定します。ただし、このプロパティを 4.21 に設定すると、Windows Mobile 2003 以前をベースにした Pocket PC にはアプリケーションをインストールできなくなります。このような古いデバイスにもインストールでき、新しいデバイスについては画面の方向に対応できるようにするには、.inf ファイルを手動で編集する必要があります。この場合、[BuildMax] プロパティを次に示す値のいずれかに設定します。

     0xA0000000 は、アプリケーションが四角形の画面 (240 × 240 ピクセル) をサポートすることを示します。

     0xC0000000 は、アプリケーションが画面の回転をサポートすることを示します。

     または

     0xE0000000 は、アプリケーションが四角形の画面と画面の回転をサポートすることを示します。

     詳細については、MSDN のホワイトペーパー「Developing Screen Orientation-Aware Applications」を参照してください。
   - Windows Mobile 2003SE 以前での Pocket PC ソリューションの場合、[Compress] プロパティと [NoUninstall デバイスの配置] プロパティは false にする必要があります。Compact Framework 2.0 が実装されたデバイスでは、このオプションを **true** に設定できます。詳細については、「[プロパティ] ウィンドウ (スマート デバイス CAB プロジェクト)」を参照してください。
   - Windows CE のセットアップ DLL を使用している場合、このプロパティ グリッドを使用して、ファイル名と位置を指定します。Windows CE のセットアップ DLL の詳細については、Pocket PC または Smartphone SDK のドキュメントを参照してください。

CAB ファイル名を変更して認証を追加するには

1. ソリューション エクスプローラで、CABProject を右クリックし、[プロパティ] をクリックします。

   CAB プロジェクトの [プロパティ ページ] ダイアログ ボックスが表示されます。[出力ファイル名] ボックスの CAB ファイルの名前とパスを「**Debug\MyApp.cab**」に変更し、[OK] をクリックします。
2. このプロパティ ページは、プロジェクトに認証を追加するときにも使用できます。認証は、Smartphone ソリューションでは必須です。また、Windows Mobile 2003 SE 以前をベースにした Pocket PC ソリューションではサポートされません。詳細については、「デバイス プロジェクトにおけるセキュリティ」を参照してください。

デバイス プロジェクト アプリケーションを CAB プロジェクトに追加するには

1. ファイル システム エディタの左ペインにある [アプリケーション フォルダ] ノードを選択します。ここで、以下の手順でファイルを選択し、対象デバイスのこのフォルダにインストールされるように指定します。

   ファイル システム エディタが表示されていない場合、ソリューション エクスプローラの CAB プロジェクト名を右クリックし、[表示] をポイントして、[ファイル システム] をクリックします。
2. Visual Studio で、[操作] メニューの [追加] をポイントし、[プロジェクト出力] をクリックします。
3. [プロジェクト出力グループの追加] ダイアログ ボックスで、[プロジェクト] ボックスから、使用するスマート デバイス プロジェクトをクリックします。
4. 出力の一覧から、[プライマリ出力] をクリックし、[OK] をクリックします。

メモ :

C++ で記述されたアプリケーションでスマート デバイス CAB プロジェクトを作成するときに、CAB プロジェクトと DLL (atl80.dll、mfc80U.dll、msvcr.dll など) を動的にリンクするには、CAB プロジェクトに依存関係を手動で追加する必要があります。ただし、MFC/ATL DLL に関する依存関係を減らすために、静的なリンクにすることを強くお勧めします。また、静的にリンクしている場合、DLL をアプリケーションに再配布しないことをお勧めします。動的にリンクしていて、CAB で DLL を再配布する必要がある場合、この DLL をデバイスのシステム ディレクトリ (\Windows など) にインストールしないでください。このような DLL は、ローカルのアプリケーション ディレクトリにインストールします。ATL/MFC ランタイムに動的にリンクしている一連のアプリケーションを再配布する場合、アプリケーションとランタイム DLL のすべてを 1 つのアプリケーション ディレクトリにインストールし、そのフォルダに配置されるアプリケーションにショートカットを指定します。こうすることでサイズがいくらか小さくなります。また、システム ディレクトリの DLL が置換され、アプリケーションとの動的なリンクが切断されるのを防ぐことができます。

CAB プロジェクトに依存関係を追加するには (C++ プロジェクトのみ)

1. ソリューション エクスプローラで、使用する CAB プロジェクトを右クリックします。[追加] をポイントし、[ファイル] をクリックします。
2. <Visual Studio インストール フォルダ>\VC\ce\dll\<プラットフォーム> に移動します。
3. 追加するファイルを選択します。
   - MFC プロジェクトの場合、Ctrl キーを押しながら MFC80U.DLL、atl80.dll、および msvcr80.dll をクリックします。使用しているアプリケーションで MFC 言語固有のリソースが必要な場合、言語固有の DLL を 1 つ以上クリックすることもできます。
   - ATL プロジェクトの場合、Ctrl キーを押しながら atl80.dll と msvcr80.dll をクリックします。ATL ソリューションで MFC をサポートしている場合、MFC80U.DLL もクリックします。
   - Win32 プロジェクトの場合、msvcr80.dll をクリックします。
4. [ファイルの追加] ダイアログ ボックスの [開く] をクリックし、ファイルを CAB プロジェクトに追加します。
5. ファイル システム エディタの左側のペインで、[対象コンピュータ上のファイル システム] を右クリックします。
6. [特別なフォルダの追加] をクリックし、[Windows フォルダ] をクリックします。
7. ファイル システム エディタの左側のペインで、プライマリ出力が保存されるフォルダをクリックします。既定では、DLL はプライマリ出力と同じフォルダに追加されます。Windows フォルダに移動するには、ファイル システム エディタの中央のペインでファイルを選択し、[Windows フォルダ] アイコンにドラッグします。
8. 同じ手順で、ソリューションに必要な他の依存関係を追加します。任意のフォルダに依存関係を追加できます。Windows フォルダに依存関係を追加する必要はありません。

デバイス プロジェクト アプリケーションのショートカットを作成するには

1. ファイル システム エディタの右側のペインで、[<アプリケーション プロジェクト名> の プライマリ出力] をクリックします。
2. [操作] メニューの [<アプリケーション プロジェクト名> の プライマリ出力 へのショートカットを作成] をクリックします。

   このコマンドで、[出力] 項目の下に [ショートカット] が追加されます。
3. [ショートカット] 項目を右クリックし、[名前の変更] をクリックして、ショートカットを適切な名前に変更します。

レジストリ エントリを追加するには

1. ソリューション エクスプローラで CAB プロジェクトを選択します。
2. [表示] メニューの [エディタ] をポイントし、[レジストリ] をクリックします。
3. レジストリ エディタで HKEY_CURRENT_USER を右クリックし、ショートカット メニューの [新しいキー] をクリックします。
4. レジストリ エディタに [新しいキー] エントリが表示されるので、名前を「SOFTWARE」に変更します。
5. この新しいキーを右クリックし、[新規作成] をポイントし、[キー] をクリックします。
6. レジストリ エディタに [新しいキー] エントリが表示されるので、名前を「MyCompany」に変更します。
7. [MyCompany] エントリを右クリックし、ショートカット メニューの [プロパティ ウィンドウ] をクリックします。

   [名前] の値が [MyCompany] に変更されます。

CAB ファイルのビルドと配置

CAB ファイルをビルドするには

1. [ビルド] メニューの [CABProject のビルド] をクリックします。

   または

   ソリューション エクスプローラの [CABProject] を右クリックし、[ビルド] をクリックします。

2. [ファイル] メニューの [すべてを保存] をクリックします。

   Smartphone ソリューションの CAB ファイルは、エンド ユーザーのデバイスに配置する前に、デジタル署名を追加する必要があります。デジタル署名は、Windows Mobile 2003SE 以前をベースとした Pocket PC ソリューションではサポートされません。詳細については、「方法 : CAB ファイルに署名する (デバイス)」を参照してください。

CAB ファイルをデバイスに配置するには

1. Windows エクスプローラで、このソリューションを保存したフォルダを開きます。CAB ファイルは、ソリューションの CABProject\Release フォルダにあります。

2. ActiveSync 4.0 以降で接続されたデバイスに CAB ファイルをコピーします。

   デバイスのファイル エクスプローラで CAB ファイル名をタップすると、Windows CE によって CAB が展開され、デバイスにアプリケーションがインストールされます。

   詳細については、Smartphone または Pocket PC SDK のドキュメントを参照してください。

参照

その他の技術情報

スマート デバイスのチュートリアル

配置用のデバイス ソリューションのパッケージ化

デバイス プロジェクトにおけるセキュリティ

デバイスでは、セキュリティ ポリシーの設定とセキュリティ ロールに基づいて、アプリケーションのインストールと実行およびシステム リソースへのアクセスが制限されます。このセクションでは、さまざまなセキュリティ モデルが適用されたデバイスで実行されるようにアプリケーションをセットアップする方法を説明するトピックを示しています。

このセクションの内容

セキュリティの概要 (デバイス)

証明書、セキュリティ モデル、プロビジョニング、およびその他の一般的な情報について説明します。

デバイスの署名プロセスのグラフィック フローチャート

デバイス プロジェクトでの署名のしくみを図で示します。

方法 : デバイス プロジェクトで証明書をインポートおよび適用する

Visual Studio 環境で証明書を配置および適用する方法について説明します。

方法 : Visual Basic アプリケーションまたは Visual C# アプリケーションに署名する (デバイス)

Visual Studio 環境で、.NET Compact Framework を使用するプロジェクトの EXE ファイルおよび DLL ファイルに証明書を適用する方法について説明します。

方法 : Visual Basic アセンブリまたは Visual C# アセンブリに署名する (デバイス)

Visual Studio 環境で、.NET Compact Framework に対して記述されたプロジェクトに証明書を適用する方法について説明します。

方法 : Visual C++ プロジェクトでプロジェクトの出力に署名する (デバイス)

Visual Studio 環境で、Visual C++ プロジェクトのプロジェクト出力に証明書を適用する方法について説明します。

方法 : CAB ファイルに署名する (デバイス)

Visual Studio 環境で、スマート デバイス CAB プロジェクトに証明書を適用する方法について説明します。

方法 : Visual Basic プロジェクトまたは Visual C# プロジェクトでデバイスを用意する

Visual Studio 環境で、.NET Compact Framework を使用するプロジェクトの証明書ストアに証明書を追加する方法について説明します。

方法 : Visual C++ プロジェクトでデバイスを用意する

Visual Studio 環境で、Visual C++ で記述されたプロジェクトの証明書ストアに証明書を追加する方法について説明します。

方法 : セキュリティ モデルを使用したデバイスを用意する

デバイスのセキュリティ モデルを設定する方法について説明します。

方法 : デバイスのセキュリティ モデルを照会する

デバイスにインストールされている証明書を確認する方法について説明します。

方法 : Signtool.exe をビルド後のイベントとして起動する (デバイス)

ビルド後のイベントによって元のバイナリが変更された後に証明書を適用する方法について説明します。

参照

セキュリティ (C# プログラミング ガイド)

セキュリティと Visual Basic 開発

C++ のセキュリティ推奨事項

.NET Compact Framework のセキュリティ

参照

その他の技術情報

スマート デバイス開発

Mobile Developer Center

セキュリティの概要 (デバイス)

デバイスでは、セキュリティ ポリシーの設定とセキュリティ ロールに基づいて、アプリケーションのインストールと実行、およびシステム リソースへのアクセスが制限されます。特定のデバイスでアプリケーションを正しく実行するには、そのアプリケーションが適切な証明書で署名され、その証明書がデバイスの証明書ストアに存在している必要があります。

署名が必要なファイル

推奨される手順では、EXE、DLL、CAB、MUI (Multilingual User Interface) の各ファイルが署名されている必要があります。また、マネージ プロジェクトでは、アセンブリ ファイルも署名されている必要があります。

セキュリティ モデル

デバイスは、いくつかのセキュリティ モデルで動作するように設定できます。セキュリティ モデルによっては、正しく承認されていないアプリケーションへのアクセスを制限できます。デバイス セキュリティ モデルの詳細については、『Windows Mobile ベースの Smartphone 開発者ガイド』および「方法 : セキュリティ モデルを使用したデバイスを用意する」を参照してください。

ビルド後にバイナリを変更した後の署名

バイナリを変更するビルド後の手順を実行した場合は、バイナリに再度署名する必要があります。つまり、プロジェクトのプロパティの [Authenticode 署名をする] を無効にし、代わりにビルド後の手順として署名する必要があります。この操作を実行する必要があるのは、バイナリへの署名後にバイナリを変更すると、署名が無効になるためです。したがって、バイナリに再度署名する必要があります。

用意

デバイスの用意とは、デバイスの証明書ストアにデジタル証明書を追加することです。デバイスでアプリケーションをインストールまたは実行しようとすると、デバイスのオペレーティング システムによって、アプリケーションの署名に使用された証明書がデバイスの証明書ストアにあるかどうかがチェックされます。証明書ストアの詳細については、「A Practical Guide to the Smartphone Application Security and Code Signing Model for Developers」を参照してください。

- Smartphone では、モバイル オペレータによって事前に用意が行われます。
- Windows Mobile 5.0 ベースの Pocket PC では、OEM によって事前に用意が行われます (それ以前の Pocket PC では、通常は行われません)。
- Visual Studio のデバイス エミュレータ用のイメージでは、事前に用意が行われます。

用意の詳細については、Smartphone または Pocket PC SDK のドキュメントを参照してください。

証明書ファイル

さまざまなセキュリティ モデル向けのアプリケーション開発を支援するために、Visual Studio 2005 には次の証明書ファイルが含まれています。既定では、これらのファイルは \Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools\TestCertificates にあります。

- TestCert_Privileged.pfx
- TestCert_UnPrivileged.pfx

これらの .pfx ファイルには、証明書とそれに対応する秘密キーの両方が含まれています (対応する秘密キーを持たない証明書でアプリケーションに署名しようとすると、署名は失敗します)。パスワードはありません。

これらの .pfx ファイルを、デスクトップ コンピュータの個人用証明書ストアにインポートします。別のストア (信頼されたルート証明機関のストアなど) にインポートしようとした場合、Visual Studio により詳細なセキュリティ警告が表示されます。詳細については、「方法 : デバイス プロジェクトで証明書をインポートおよび適用する」を参照してください。

参照

処理手順

方法 : Visual C++ プロジェクトでデバイスを用意する
方法 : Visual Basic プロジェクトまたは Visual C# プロジェクトでデバイスを用意する

概念

接続文字列のセキュリティ保護

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

デバイスの署名プロセスのグラフィック フローチャート

次のグラフィカル表示は、署名プロセスの各部分がどのように関連しているかを示しています。

プロジェクトに署名する場合の最初の手順は、証明書を選択することです。既に証明書ストアに格納されている証明書を使用することも、新しい証明書をインポートして選択することもできます。

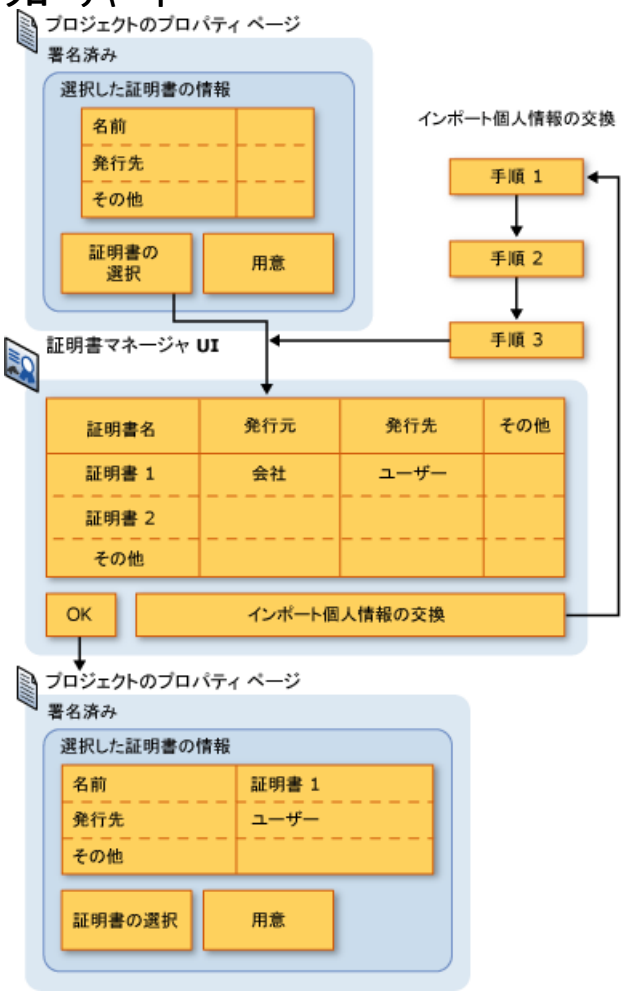
証明書を選択した後は、その証明書を使用してデバイスを用意するかどうかを決める必要があります。デバイスを用意する場合は、デバイス上のどの証明書ストア (特権のある証明書ストアまたは特権のない証明書ストア) が証明書を受け取るかを選択する必要があります。

ビルドを実行するたびに、プロジェクトは、選択した証明書で署名されます。配置する (たとえば、F5 キーを押す) たびに、選択した証明書を使用したデバイスが用意されます。この証明書は、デバイス上の選択した証明書ストアに格納されます。

Visual Studio IDE では、通常、プロジェクトのプロパティ ページのいずれかで署名オプションを設定します。

Authenticode 署名を行わないプロジェクトの場合、このフローチャートの内容は一切適用されません。

フローチャート



参照

処理手順

方法 : Visual Basic アプリケーションまたは Visual C# アプリケーションに署名する (デバイス)
方法 : Visual Basic アセンブリまたは Visual C# アセンブリに署名する (デバイス)
方法 : Visual C++ プロジェクトでプロジェクトの出力に署名する (デバイス)
方法 : CAB ファイルに署名する (デバイス)

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

方法：デバイス プロジェクトで証明書をインポートおよび適用する

[証明書の選択] ダイアログ ボックスは、デバイス プロジェクトに署名するための中心ポータルです。次の手順で説明するように、このダイアログ ボックスから、[証明書の管理] ダイアログ ボックスと証明書のインポート ウィザードを使用できます。

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

[証明書の選択] ダイアログ ボックスを表示する

[証明書の選択] ダイアログ ボックスへのアクセス方法は、署名対象のプロジェクトの種類によって異なります。

[証明書の選択] ダイアログ ボックスを表示するには

1. ソリューション エクスプローラで、[<プロジェクト名>] を右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. 続いて、次のいずれかの手順を使用します。
   - Visual Basic プロジェクトおよび Visual C# プロジェクトでは、プロジェクト デザイナで [デバイス] をクリックして [Authenticode で署名] を選択します。次に、[証明書の選択] をクリックします。
   - Visual C++ プロジェクトでは、[Authenticode 署名をする] を選択してから [証明書] プロパティ行にある [...] ボタンをクリックします。
   - スマート デバイス CAB プロジェクトでは、[Authenticode 署名をする] を選択してから [ストアから選択] をクリックします。

デバイス プロジェクトの証明書を選択する

上に示した手順に従って [証明書の選択] ダイアログ ボックスを表示した後は、目的の証明書を選択できます。

[証明書の選択] ダイアログ ボックスを使用してプロジェクトの証明書を選択するには

- プロジェクトに使用する証明書が [証明書の選択] ダイアログ ボックスに表示されている場合は、その証明書を選択して [OK] をクリックします。

  プロジェクトのビルド時に、その証明書を使用してプロジェクトが署名されます。
- プロジェクトに使用する証明書が [証明書の選択] ダイアログ ボックスに表示されていない場合は、証明書のインポート ウィザードを使用して証明書をインポートできます。

デバイス プロジェクトの証明書をインポートする

Visual Studio に用意されているテスト証明書をインポートし、プロジェクトにそれらの証明書を適用することで、[証明書の選択] ダイアログ ボックスを設定する方法を次の手順に示します。この手順は、他の証明書を適用する場合も同じです。

Visual Studio には、プロジェクトに適用する証明書をインポートするためのユーザー インターフェイス要素として次の 3 つがあります。

- [証明書の選択] ダイアログ ボックス。現在のプロジェクトに適用する証明書を指定します。
- [証明書の管理] ダイアログ ボックス。開発用コンピュータ上で使用できる証明書ファイルを一覧表示します。
- 証明書のインポート ウィザード。証明書ファイルを選択し、それを格納する場所を指定できます。

証明書のインポート ウィザードを使用してテスト証明書をインポートするには

1. [証明書の選択] ダイアログ ボックスで、[証明書の管理] をクリックします。

   [証明書の管理] ダイアログ ボックスに、開発用コンピュータに格納されている証明書の一覧が表示されます。
2. [インポート] をクリックして、証明書のインポート ウィザードを開きます。
3. [次へ] をクリックして、ウィザードの [インポートする証明書ファイル] ページを開きます。

4. [参照] をクリックし、Visual Studio の [TestCertificates] フォルダに移動します。

   既定では、このフォルダは \Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools の下にあります。

5. [ファイルの種類] の [すべてのファイル (*.*)] を選択します。次に、*TestCert_Privileged.pfx* または *TestCert_Unprivileged.pfx* を選択して [開く] をクリックします。

6. ウィザードの [インポートする証明書ファイル] ページで、[次へ] をクリックして [パスワード] ページを開きます。

   [Passwod] ボックスは空欄のままにしておきます。これらのテスト証明書にはパスワードは設定されていません。

7. [次へ] をクリックして [証明書ストア] ページを開きます。[証明書ストア] ボックスで [Personal] が選択されていることを確認します。

8. [次へ] をクリックして完了ページを表示します。次に、[完了] をクリックします。

   "正常に読み込まれました" というメッセージが表示されます。

9. [OK] をクリックしてこのメッセージ ボックスを閉じます。

   [証明書の管理] ダイアログ ボックスの一覧に、インポートした証明書が表示されます。[閉じる] をクリックし、[証明書の選択] ダイアログ ボックスに戻ります。

10. 必要な証明書をクリックし、[OK] をクリックします。

    最初のプロパティ ページに証明書が表示されます。

参照

処理手順

方法 : Visual Basic アプリケーションまたは Visual C# アプリケーションに署名する (デバイス)
方法 : Visual Basic アセンブリまたは Visual C# アセンブリに署名する (デバイス)
方法 : Visual C++ プロジェクトでプロジェクトの出力に署名する (デバイス)
方法 : CAB ファイルに署名する (デバイス)

概念

セキュリティの概要 (デバイス)

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

方法：Visual Basic アプリケーションまたは Visual C# アプリケーションに署名する (デバイス)

以下の手順では、ソリューションにスマート デバイス Visual Basic プロジェクトまたはスマート デバイス Visual C# プロジェクトが含まれているものとします。詳細については、「.NET Compact Framework を使用したデバイスのプログラミング」を参照してください。

これらの手順は、EXE プロジェクトでも DLL プロジェクトでも同じです。

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Visual Basic デバイス プロジェクトまたは Visual C# デバイス プロジェクトに署名するには

1. ソリューション エクスプローラで、Visual Basic プロジェクトまたは Visual C# プロジェクトを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [デバイス] ページで、[Authenticode で署名] をクリックします。
3. [証明書の選択] をクリックします。
4. [証明書の選択] ダイアログ ボックスで、次の操作を実行します。
   - 必要な証明書が一覧に表示されている場合は、その証明書を選択し、[OK] をクリックしてダイアログ ボックスを閉じます。
   - 必要な証明書が一覧に表示されていない場合は、[証明書の管理] をクリックして [証明書の管理] ダイアログ ボックスを開きます。詳細については、「方法 : デバイス プロジェクトで証明書をインポートおよび適用する」を参照してください。

     証明書の取得が完了したら、[証明書の選択] ダイアログ ボックスの [OK] をクリックします。証明書の詳細が [デバイス] ページの [アプリケーションの署名] ウィンドウに表示されます。

参照

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

方法 : Visual Basic アセンブリまたは Visual C# アセンブリに署名する (デバイス)

以下の手順では、ソリューションにスマート デバイス Visual Basic プロジェクトまたはスマート デバイス Visual C# プロジェクトが含まれているものと想定しています。これらのプロジェクトの作成の詳細については、「.NET Compact Framework を使用したデバイスのプログラミング」を参照してください。

これらの手順は、EXE プロジェクトでも DLL プロジェクトでも同じです。

メモ :
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Visual Basic デバイス プロジェクトまたは Visual C# デバイス プロジェクトのアセンブリに署名するには

1. ソリューション エクスプローラで、Visual Basic プロジェクトまたは Visual C# プロジェクトを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [署名] ページで、[アセンブリの署名] をクリックします。
3. [厳密な名前のキー ファイルを選択してください] ボックスで次の操作を行います。
   - 既存の厳密な名前のキー ファイルを使用する場合は、[<参照...>] をクリックして [ファイルの選択] ダイアログを開きます。
   - 新しい厳密な名前のキー ファイルを作成する場合は、[新規作成] をクリックして [厳密な名前キーの作成] ダイアログを開きます。

アセンブリへの署名を遅延するには

- 上の手順の完了後に、[遅延署名のみ] をクリックします。

  必要な秘密キーにアクセスできない場合にこの機能を使用します。遅延署名では公開キーが用意され、アセンブリが渡されるまで秘密キーの追加が延期されます。詳細については、「方法 : アセンブリに遅延署名する (Visual Studio)」を参照してください。

参照

概念

マネージ アプリケーションに対する厳密な名前による署名

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

方法：Visual C++ プロジェクトでプロジェクトの出力に署名する (デバイス)

以下の手順では、ソリューションに スマート デバイス Visual C++ プロジェクトが含まれているものと想定しています。詳細については、「Visual C++ を使用したデバイスのプログラミング」を参照してください。

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Visual C++ デバイス プロジェクトでプロジェクト出力に署名するには

1. ソリューション エクスプローラで、Visual C++ プロジェクトを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [Authenticode 署名をする] ページを展開します。
3. [Authenticode で署名] プロパティで、[はい] を選択します。
4. [証明書] プロパティで、省略記号ボタン ([...]) をクリックします。
5. [証明書の選択] ダイアログ ボックスで、次の操作を実行します。
   - 必要な証明書が一覧に表示されている場合は、その証明書を選択し、[OK] をクリックしてダイアログ ボックスを閉じます。
   - 必要な証明書が一覧に表示されていない場合は、[証明書の管理] をクリックして [証明書の管理] ダイアログ ボックスを開きます。詳細については、「方法 : デバイス プロジェクトで証明書をインポートおよび適用する」を参照してください。

     証明書の取得が完了したら、[証明書の選択] ダイアログ ボックスの [OK] をクリックします。証明書は [Authenticode 署名をする] ページの [証明書] 行に表示されます。
6. [Authenticode 署名をする] ページで、[OK] をクリックします。

参照

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

方法 : CAB ファイルに署名する (デバイス)

以下の手順では、ソリューションにスマート デバイス Cab プロジェクトが含まれているものと想定しています。詳細については、「デバイス ソリューションをパッケージ化する方法の概要」を参照してください。

メモ :
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

スマート デバイスの .cab ファイルに署名するには

1. ソリューション エクスプローラで、スマート デバイス Cab プロジェクトを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [ビルド] ページで、[Authenticode で署名] を選択します。
3. [ストアから選択] をクリックします。
4. [証明書の選択] ダイアログ ボックスで、次の操作を実行します。
   - 必要な証明書が一覧に表示されている場合は、その証明書を選択し、[OK] をクリックしてダイアログ ボックスを閉じます。
   - 必要な証明書が一覧に表示されていない場合は、[証明書の管理] をクリックして [証明書の管理] ダイアログ ボックスを開きます。詳細については、「方法 : デバイス プロジェクトで証明書をインポートおよび適用する」を参照してください。

     証明書の取得が完了したら、[証明書の選択] ダイアログ ボックスの [OK] をクリックします。証明書が [ビルド] ページの [証明書] ボックスに表示されます。
5. [ビルド] ページで、[OK] をクリックします。

参照

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

方法：Visual Basic プロジェクトまたは Visual C# プロジェクトでデバイスを用意する

デバイスの用意とは、デバイスの証明書ストアにデジタル証明書を追加することです。詳細については、「A Practical Guide to the Smartphone Security and Code Signing Model for Developers」を参照してください。

次の手順では、以下の点を想定しています。

- ソリューションにスマート デバイス Visual Basic プロジェクトまたはスマート デバイス Visual C# プロジェクトが含まれていること。詳細については、「.NET Compact Framework を使用したデバイスのプログラミング」を参照してください。
- アプリケーションに署名していること。詳細については、「方法 : Visual Basic アプリケーションまたは Visual C# アプリケーションに署名する (デバイス)」を参照してください。

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

マネージ デバイス プロジェクトでデバイスを用意するには

1. ソリューション エクスプローラで、Visual Basic プロジェクトまたは Visual C# プロジェクトを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [デバイス] ページを開きます。
3. [デバイスのプロビジョニング] ボックスで、次のいずれかのオプションを選択します。
   - [ターゲット デバイスをプロビジョニングしません]
   - [特権のあるストアに証明書を追加します]
   - [特権のないストアに証明書を追加します]

参照

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

方法：Visual C++ プロジェクトでデバイスを用意する

手順を実行するための要件は次のとおりです。

- ソリューションにスマート デバイス Visual C++ プロジェクトが含まれていること。詳細については、「Visual C++ を使用したデバイスのプログラミング」を参照してください。
- Visual C++ プロジェクトのプロジェクト出力に署名したこと。Visual C++ プロジェクトへの署名の詳細については、「方法 : Visual C++ プロジェクトでプロジェクトの出力に署名する (デバイス)」を参照してください。

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Visual C++ デバイス プロジェクトでデバイスを用意するには

1. ソリューション エクスプローラで、Visual C++ プロジェクトを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [Authenticode 署名をする] ページを展開します。
3. [デバイスのプロビジョニング] プロパティで、[特権のある証明書ストア] または [特権のない証明書ストア] を選択します。
4. [Authenticode 署名をする] ページで、[OK] をクリックします。

参照

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

方法：セキュリティ モデルを使用したデバイスを用意する

デバイスのセキュリティ モデルを明示的に設定することで、さまざまなセキュリティ モデルでアプリケーションをテストできます。デバイスが相手先ブランド供給 (OEM: Original Equipment Manufacturer) によってロックされている場合は、異なるセキュリティ モデルを使用できない可能性があります。デバイスがロックされていない場合は、任意のセキュリティ モデルを使用してそのデバイスを用意できます。

次のセキュリティ モデル XML ファイルは、Visual Studio 2005 に同梱されています。既定の場所は、\Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools\SecurityModels です。

- Locked.xml では、次に示す 2 階層のセキュリティ モデルが設定されます。
    - アプリケーションを実行する前に確認メッセージを表示します。
    - 署名されていないアプリケーションを実行しません。
- Prompt.xml では、次に示す 2 階層のセキュリティ モデルが設定されます。
    - アプリケーションを実行する前に確認メッセージを表示します。
    - 署名されていないアプリケーションを特権のないアプリケーションとして実行します。
- Open.xml では、次に示す 1 階層のセキュリティ モデルが設定されます。
    - 確認メッセージを表示しません。
    - 署名されていないアプリケーションを含むすべてのアプリケーションを特権のあるアプリケーションとして実行します。

詳細については、「Windows Mobile ベース デバイスのための準備」を参照してください。

セキュリティ モデルを使用してデバイスを用意するには、次のコンポーネントが必要です。

- ActiveSync。
- RapiConfig.exe。既定では、\Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools にあります。
- セキュリティ XML ファイル。

セキュリティ モデルを使用してデバイスを用意するには

1. デバイスへの ActiveSync 接続を確立します。
2. コマンド プロンプトで次のコマンドを入力します。*securityfile.xml* は、セキュリティ モデル XML ファイルです。

```
RapiConfig.exe /P /M <securityfile.xml>
```

参照

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

方法：デバイスのセキュリティ モデルを照会する

デバイスに照会することで、デバイスの証明書ストアに既にインストールされている証明書を確認できます。その情報を使用して、アプリケーションの署名に使用する証明書を決定できます。

照会を実行するには、RapiConfig.exe を実行して、証明書ストアのクエリを含んだ StoreQuery XML ファイルを渡します。これにより、RapiConfig.exe から、クエリの結果を含んだ XML ファイルが出力されます。

RapiConfig.exe、CertStoreQuery.xml、およびいくつかのサンプル xml クエリ ファイルは、既定では \Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools にあります。

デバイスの証明書ストアを照会するには、次のコンポーネントが必要です。

- ActiveSync
- RapiConfig.exe
- 証明書ストア クエリ XML ファイル (CertStoreQuery.xml)

デバイスに対してセキュリティ モデルを照会するには

1. デバイスへの ActiveSync 接続を確立します。
2. コマンド プロンプトで次のコマンドを入力します。*certstorequery.xml* は、証明書ストア クエリ XML ファイルです。

   ```
   Rapiconfig.exe /P /M <certstorequery.xml>
   ```

3. 生成される RapiConfigOut.xml ファイルを参照します。

参照

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

方法：Signtool.exe をビルド後のイベントとして起動する (デバイス)

signtool.exe は、他のビルド後のイベントが元のバイナリを変更したときにビルド後のイベントとして実行されます。署名されたバイナリに変更を加えると、元の署名は無効になります。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Visual Basic デバイス プロジェクトおよび Visual C# デバイス プロジェクトで signtool.exe をビルド後のイベントとして起動するには

1. ソリューション エクスプローラで、プロジェクトを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [ビルド イベント] ページ (Visual C# の場合) または [コンパイル] ページ (Visual Basic の場合) で、[ビルド後の編集] をクリックします。
3. [ビルド後に実行するコマンド ライン] ダイアログ ボックスで、signtool コマンド ラインに選択したオプションを含めて入力します。

   コマンド ラインから signtool を実行する方法の詳細については、「SignTool」を参照してください。

Visual C++ デバイス プロジェクトで signtool.exe をビルド後のイベントとして起動するには

1. ソリューション エクスプローラで、プロジェクトを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [構成プロパティ] で、[ビルド イベント] ノードを展開します。
3. [ビルド後のイベント] をクリックします。
4. [コマンド ライン] プロパティを選択し、省略記号ボタン ([...]]) をクリックして [コマンド ライン] ダイアログ ボックスを開きます。
5. signtool コマンド ラインに選択したオプションを含めて入力します。

   コマンド ラインから signtool を実行する方法の詳細については、「SignTool」を参照してください。

参照

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

サンプルとチュートリアル (スマート デバイス プロジェクト)

次のサンプル プロジェクトを使用して、デバイス プログラミングのさまざまな問題を解決するための構文、構造、および技法について説明します。

このセクションの内容

スマート デバイスのサンプル

調査用のビルド済みプロジェクトを紹介します。

スマート デバイスのチュートリアル

さまざまな種類のアプリケーションおよびコンポーネントを作成する方法について、詳細な手順を説明します。このトピックでは、マネージ プロジェクトとネイティブ プロジェクトの両方について説明しています。

関連するセクション

Visual Studio のサンプル

アプリケーションのデモンストレーションを行うサンプルが用意されています。

Visual Studio に関するチュートリアル

さまざまなプログラミング言語によって異なる種類のアプリケーションやコンポーネントを作成する方法について説明します。

参照

その他の技術情報

スマート デバイス開発

Mobile Developer Center

スマート デバイスのチュートリアル

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

これらのチュートリアルでは、さまざまなタスクをスマート デバイス プロジェクトに実装するための手順について詳しく説明します。各チュートリアルでは、特定の種類のデバイス アプリケーションを開発するための手順を示します。デスクトップ アプリケーション用にデザインされた Visual Studio チュートリアルをデバイス アプリケーションで動作させるには、通常、そのチュートリアルに変更を加える必要があります。.NET Framework で利用できる 1 つ以上のクラスが .NET Compact Framework に存在しないことが主な理由です。

この後のチュートリアルで使用する既定の対象デバイスはエミュレータです。物理デバイスがなくてもチュートリアルを実行できます。開発用コンピュータに接続して通信を行うことができる物理デバイスがある場合は、エミュレータの代わりにそのデバイスを対象にできます。詳細については、「スマート デバイス プロジェクトのハードウェア要件とソフトウェア要件」を参照してください。

メモ :
一部のインターフェイス要素 (たとえば、新しいプロジェクト ウィザード) によって、チュートリアルのテキストが見えなくなる場合があります。1 つの対処方法としては、Visual Studio 統合開発環境 (IDE: Integrated Development Environment) ブラウザのヘルプ ビューアを外部ウィンドウに切り替えてください。外部ウィンドウに切り替えるには、[ツール] メニューの [オプション] をクリックし、[環境]、[ヘルプ] の順にクリックします。[外部ヘルプ] を選択し、変更を有効にするために IDE を再起動します。

次の表は、Visual Studio 2005 SP1 に合わせて更新されています。

このセクションの内容

チュートリアル : デバイス対応の Windows フォーム アプリケーションの作成

簡単な Windows フォーム アプリケーションを開発する方法と、それをスマート デバイスにダウンロードする方法について説明します。

チュートリアル : デバイスのユーザー コントロールの作成

マネージ デバイス プロジェクトでユーザー コントロールを作成する方法について説明します。

チュートリアル : 簡単な属性をユーザー コントロールに追加する

デバイス プロジェクトでのユーザー コントロールのカスタマイズについて説明します。

チュートリアル : データベース マスター/詳細アプリケーション

SQL Server Mobile データベースまたは SQL Server Compact Edition データベースをデータ ソースとして設定する方法、コントロールにバインドされたデータ オブジェクトを Windows フォームにドラッグする方法、マスター/詳細リレーションシップを構成する方法などについて説明します。

チュートリアル : パラメータ クエリ アプリケーション

SQL Server Mobile データベースまたは SQL Server Compact Edition データベースをデータ ソースとして設定する方法、コントロールにバインドされたデータ オブジェクトを Windows フォームにドラッグする方法、およびパラメータ クエリをアプリケーションに含める方法について説明します。

チュートリアル : Hello World: スマート デバイスの COM 相互運用の例

マネージ プロジェクトで COM オブジェクトをホストする方法について説明します。

方法 : マルチプラットフォーム対応のデバイス プロジェクトを作成する (Visual C++)

複数のデバイスを対象にする方法について説明します。

チュートリアル : スマート デバイス用マルチプラットフォーム MFC アプリケーションの作成

複数のプラットフォームを対象にする MFC アプリケーションの作成方法について説明します。

チュートリアル : スマート デバイス用 MFC マルチプラットフォーム ActiveX コントロールの作成

MFC ActiveX コントロールの作成方法について説明します。

チュートリアル : スマート デバイス用 ATL マルチプラットフォーム ActiveX コントロールの作成

マルチプラットフォーム対応のスマート デバイス ATL プロジェクトと簡単な ATL コントロールを作成する方法について説明します。

関連するセクション

チュートリアル : マネージ コードとネイティブ コードの両方を含むソリューションのデバッグ

ネイティブ デバッガおよびマネージ デバッガを交互に起動する手順を説明します。

チュートリアル : 配置用のスマート デバイス ソリューションのパッケージ化

エンド ユーザーに配布する、デバイスにインストールするための CAB ファイルを生成する方法、および .inf ファイルを手動で構成せずに IDE を使用してさまざまな設定を変更する方法について説明します。

スマート デバイスのサンプル

.NET Compact Framework に基づくアプリケーションの例を示す完全なプロジェクトと、ネイティブ コードを使用するサンプルへのリンクを示します。

Visual Studio に関するチュートリアル

デスクトップ対応のさまざまなアプリケーションの手順を詳しく説明します。

参照

その他の技術情報

サンプルとチュートリアル (スマート デバイス プロジェクト)

チュートリアル : デバイス対応の Windows フォーム アプリケーションの作成

このチュートリアルでは、Visual Basic または Visual C# を使用して単純な Windows フォーム アプリケーションをビルドし、Pocket PC エミュレータでそのアプリケーションを実行します。このチュートリアルでは、デスクトップのプログラミングと、デバイスのプログラミング、つまりデバイスを対象にする場合のプログラミングでの主な違いを示します。このチュートリアルでは、デバイスとは Pocket PC 2003 の組み込みエミュレータです。

メモ :
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

このチュートリアルでは、Visual Basic 開発設定と Visual C# 開発設定を使用しています。

このチュートリアルは、主に次の 5 つの手順で構成されています。

- Windows フォームを使用するデバイス プロジェクトの作成。
- フォームへのコントロールの追加。
- コントロールへのイベント処理の追加。
- プロジェクトを実行するデバイスの選択。
- アプリケーションのビルドとデバイスへの配置。

対象デバイスの選択

ソリューションを配置するときにデバイス選択のダイアログが表示されるようにするには、次の手順を実行します。

配置時にデバイス選択のダイアログを表示するには

1. [ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックします。次に、[全般] をクリックします ([デバイス ツール] が表示されない場合は、[オプション] ダイアログ ボックスの下部にある [すべての設定を表示] を選択します)。
2. [デバイス プロジェクトの配置前に選択できるデバイスを表示] チェック ボックスをオンにします。

アプリケーションの作成

Windows フォーム プロジェクトの作成は、コントロールやイベント処理の追加と同様、デスクトップ プロジェクトでもデバイス プロジェクトでも同じプロセスです。大きな違いは、.NET Compact Framework では使用できるクラスが少ないことです。

Windows フォームを使用するデバイス プロジェクトを作成するには

1. (Visual Basic) Visual Studio 2005 の [ファイル] メニューの [新しいプロジェクト] をクリックします。

   または

   (Visual C#) Visual Studio 2005 の [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。

2. [新しいプロジェクト] ダイアログ ボックスの [プロジェクトの種類] で、[Visual Basic] または [Visual C#] を展開し、[スマート デバイス] を展開します。次に、[Pocket PC 2003] をクリックします。

   目的の言語が最初に表示されない場合は、[他の言語] を展開します。この表示は、ユーザーの開発設定で管理されます。

3. [テンプレート] の [デバイス アプリケーション] をクリックします。
4. [プロジェクト名] ボックスに「DeviceSample」と入力し、[OK] をクリックします。
5. (Visual C# のみ) [場所] ボックスで、プロジェクト ファイルを格納する場所を確認し、[OK] をクリックします。

   Windows フォーム デザイナに Pocket PC デバイスが表示されます。

コントロールをフォームに追加するには

1. ツールボックスから、フォームに Button コントロールをドラッグします。

ツールボックスが表示されていない場合は、[表示] メニューの [ツールボックス] をクリックします。

ツールボックスに [デバイス コントロール] タブが表示されていない場合は、ツールボックスを右クリックして [すべて表示] をクリックします。

2. Button コントロールを右クリックして、[プロパティ] をクリックします。
3. [プロパティ] ウィンドウに「**Say Hello**」と入力し、Enter キーを押して [Text] プロパティを設定します。

Button コントロールのイベント処理を追加するには

1. フォーム上のボタンをダブルクリックします。

   コード エディタが開き、イベント ハンドラにカーソルが表示されます。

2. 次の Visual Basic コードを入力します。

```
MessageBox.Show("Hello, World!")
```

   または

   次の C# コードを入力します。

```
MessageBox.Show("Hello, World!");
```

アプリケーションのビルドとテスト

ここでは、デスクトップ プロジェクトとの違いがあります。デバイス プロジェクトでは、通常はプロジェクトを実行する対象を複数から選択できます。このチュートリアルでは、Pocket PC エミュレータを選択します。サポートされている物理デバイスが開発用コンピュータと既に連携している場合は、物理デバイスを選択することもできます。

アプリケーションをビルドしてテストするには

1. [デバッグ] メニューの [開始] または [デバッグ開始] をクリックします。
2. [配置] ダイアログ ボックスで、[Pocket PC 2003 SE エミュレータ] を選択し、[配置] をクリックします。

   ステータス バーに進行状況が表示されます。

3. アプリケーションをエミュレータで実行しているときに、ボタンをタップして、"Hello, World!" と表示されることを確認します。

他のチュートリアルの準備

他のチュートリアルを実行する場合、または他のプロジェクトを開く場合、エミュレータを完全にシャットダウンし、このソリューションを終了します。

エミュレータとソリューションを終了するには

1. エミュレータの [ファイル] メニューで、[終了] をクリックします。
2. [デバイス エミュレータ] メッセージ ボックスで、[終了する前にエミュレータの状態を保存しますか?] に対して [いいえ] をクリックします。
3. Visual Studio で、[デバッグ] メニューの [デバッグの停止] をクリックします。
4. 接続が切断されたことを示すメッセージが表示された場合は、[OK] をクリックします。
5. [ファイル] メニューの [ソリューションを閉じる] をクリックします。

参照
処理手順
チュートリアル : 簡単な Windows フォームの作成
関連項目
[全般] ([オプション] ダイアログ ボックス - [デバイス ツール])
ツールボックス
その他の技術情報
スマート デバイスのチュートリアル

チュートリアル : デバイスのユーザー コントロールの作成

このチュートリアルでは、コントロール ライブラリを作成し、これに格納するユーザー コントロールを作成します。1 つの再利用可能単位としての複数の Windows フォーム コントロールの組み合わせであるユーザー コントロールと、標準のコントロールからは利用できない UI または機能を必要とするコントロールであるカスタム コントロールとの違いについて説明します。

このチュートリアルのユーザー コントロールは、デバイスの単純な時計表示であり、「チュートリアル : Visual Basic による複合コントロールの作成」および「チュートリアル : Visual C# による複合コントロールの作成」など、同様のデスクトップ チュートリアルをひな型としています。

このチュートリアルは、次の 4 つの部分で構成されます。

- コントロール ライブラリとコントロールの作成
- ライブラリとコントロールの名前変更
- コントロールへのコンポーネントの追加
- デバイス エミュレータでのコントロールのテスト

このチュートリアルでは、Visual Basic または Visual C# のどちらかを使用できます。ファイル名が同じでも、言語固有の拡張子が付いているファイルをどちらの言語でも呼び出す場合、使用する言語の拡張子を選択する必要があることが縦棒により示されます。たとえば、**filename.vb|cs** となります。

メモ :
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。 このチュートリアルでは、Visual Basic 開発設定と Visual C# 開発設定を使用しています。

前提条件

なし。

対象デバイスの選択

ソリューションを配置するときにデバイス選択のダイアログが表示されるようにするには、次の手順を実行します。

配置時にデバイス選択のダイアログを表示するには

1. [ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックします。次に、[全般] をクリックします

   ([デバイス ツール] が表示されない場合は、[オプション] ダイアログ ボックスの下部にある [すべての設定を表示] を選択します)。

2. [デバイス プロジェクトの配置前に選択できるデバイスを表示] チェック ボックスをオンにします。

プロジェクトの作成

新しいプロジェクトに付けた名前により、ルート名前空間とアセンブリ名も設定されます。

コントロール ライブラリとコントロールを作成するには

1. (Visual Basic) [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。

   または

   (Visual C#) [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。

2. [新しいプロジェクト] ダイアログ ボックスの [プロジェクトの種類] で、[Visual Basic] または [Visual C#] を展開し、[スマート デバイス] を展開します。次に、[Pocket PC 2003] をクリックします。

   目的の言語が最初に表示されない場合は、[他の言語] を展開します。この表示は、ユーザーの開発設定で管理されます。

3. [テンプレート] の [コントロール ライブラリ] をクリックします。
4. [プロジェクト名] ボックスに、「ctlDevClockLib」と入力します。
5. (Visual C# のみ) [場所] ボックスで、プロジェクト ファイルを格納する場所を確認します。

6. [OK] をクリックします。

   コンポーネント デザイナが表示されます。

プロジェクト名、ルート名前空間、およびアセンブリ名 (ctlDevClockLib) は既に指定しました。しかし、プロジェクト内のコンポーネントには Visual Studio により割り当てられた既定の名前があります (たとえば、UserControl1 など)。通常は、それらの名前をもっとわかりやすい用語に変更します。

ライブラリとコントロールの名前を変更するには

1. ソリューション エクスプローラで、[UserControl1.vb|cs] を右クリックし、コンテキスト メニューの [プロパティ] をクリックします。
2. [ファイル名] を「**ctlDevClock.vb|cs**」に変更します。
3. (Visual C# のみ) このコード要素に対するすべての参照の名前を変更するかどうかを確認するメッセージ ボックスに対して、[はい] をクリックします。

   [プロパティ] ウィンドウで行った名前の変更が、他の参照 (クラス名やコンストラクタなど) に反映されます。

次に、[ツールボックス] からコンポーネントを追加して、機能やユーザーとの対話をユーザー コントロールに設定します。このチュートリアルでは、システム時刻にアクセスする **タイマー** コントロールと、時刻を表示する **ラベル** コントロールを追加します。

コンポーネントを追加して、それらのプロパティを変更するには

1. [ツールボックス] の [ラベル] をダブルクリックします。

   ラベル コントロールが コンポーネント デザイナ のユーザー コントロールに追加されます。

2. ラベル コントロールの [プロパティ] ウィンドウで、次の手順を実行します。

プロパティ	変更後の値
(Name)	**lblDisplay**
テキスト	(空白)
TextAlign	TopCenter
Font.Size	14

1. [ツールボックス] の [タイマー] をダブルクリックします。

   コンポーネント トレイに **タイマー** コントロールが表示されます。

2. タイマ コントロールの [プロパティ] ウィンドウで、次の手順を実行します。

プロパティ	変更後の値
Interval	1000
Enabled	True

1.

次の手順では、**ラベル** コントロールにイベント ハンドラを追加して、時計が時を刻むようにします。

イベント ハンドラを追加するには

1. コンポーネント トレイで、[Timer1] をダブルクリックして、コード エディタで Tick イベントの場所を開きます。
2. (Visual Basic) 次のイベント処理コードを挿入します。

```
lblDisplay.Text = Format(Now, "hh:mm:ss")
```

   または

(Visual C#)

```
lblDisplay.Text = DateTime.Now.ToLongTimeString();
```

3. (Visual Basic) アクセス修飾子を **Private** から **Protected** に変更し、**Overridable** キーワードを追加して、ハンドラ コードを次のようにします。

```
Protected Overridable Sub Timer1_Tick(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles Timer1.Tick
   ' Causes the label to display the current time
   lblDisplay.Text = Format(Now, "hh:mm:ss")
End Sub
```

または

(Visual C#) アクセス修飾子を **private** から **protected** に変更し、**virtual** キーワードを追加して、ハンドラ コードを次のようにします。

```
protected virtual void timer1_Tick(object sender, System.EventArgs
   e)
{
   // Causes the label to display the current time.
   lblDisplay.Text = DateTime.Now.ToLongTimeString();
}
```

4. [ファイル] メニューの [すべてを保存] をクリックします。
5. (Visual Basic のみ) [プロジェクトの保存] ダイアログ ボックスで、プロジェクトを「**ctlDevClockLib**」として選択した場所に保存します。

コントロールのテスト

この単純なデバイス アプリケーションは、コントロールのテスト コンテナとして機能します。

コントロールをビルドし、テスト コンテナを作成するには

1. [ビルド] メニューの [ビルド] (または [ctlDevClockLib のビルド]) をクリックします。
2. [ファイル] メニューの [追加] をポイントし、[新しいプロジェクト] をクリックします。
3. [新しいプロジェクトの追加] ダイアログ ボックスの [プロジェクトの種類] ペインで [Pocket PC 2003] をクリックし、[テンプレート] ペインで [デバイス アプリケーション] をクリックします。
4. [プロジェクト名] ボックスに「Test」と入力し、[OK] をクリックします。
5. ソリューション エクスプローラで、Test プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
6. もう一度 Test プロジェクトを右クリックし、[参照の追加] をクリックします。
7. [参照の追加] ダイアログ ボックスの [プロジェクト] タブをクリックし、[ctlDevClockLib] をクリックします。
8. [ツールボックス] の [ctlDevClockLib コンポーネント] タブを探し、[ctlDevClock] コンポーネントをダブルクリックします。

   コントロールがデザイナに読み込まれます。
9. [デバッグ] メニューの [開始] または [デバッグ開始] をクリックします。
10. [配置] ダイアログ ボックスで、[Pocket PC 2003 SE エミュレータ] を選択し、[配置] をクリックします。

参照
概念
コントロールの種類に関するアドバイス
その他の技術情報
デザイン時の Windows フォーム コントロールの開発

チュートリアル : 簡単な属性をユーザー コントロールに追加する

このチュートリアルでは、デバイス プロジェクトのユーザー コントロールに属性を追加する方法について説明します。特に、デザイン時に、コントロールのプロパティを非表示にするカスタム属性を追加します。この機能は、プロパティ値の変更を防ぐときに便利です。

このプロセスはデスクトップの場合と似ていますが、デバイス プロジェクトではこの情報を別のメタデータ ファイル (.xmta) に格納します。

メモ :

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

このチュートリアルは、Visual C# 開発設定を使用して記述されています。

UserControl1 クラスを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [プロジェクトの種類] ペインの [Visual C#] を展開し、[スマート デバイス] を展開して、[Pocket PC 2003] をクリックします。
3. [テンプレート] ペインの [コントロール ライブラリ] をクリックします。
4. [プロジェクト名] ボックスに「**MyControlLibrary**」と入力し、[OK] をクリックします。

   デザイナが開き、新しいユーザー コントロール クラスを表す四角形が表示されます。

プロパティを追加するには

1. ソリューション エクスプローラで、[UserControl1.cs] を右クリックし、ショートカット メニューの [クラス ダイアグラムで表示] をクリックします。

   クラス ダイアグラムを表す角の丸い四角形が開きます。
2. クラス ダイアグラムを右クリックし、ショートカット メニューの [クラスの詳細情報] をクリックします。
3. [クラスの詳細] ウィンドウの [プロパティ] セクションに表示される [<プロパティの追加>] プロンプトで、「**MyProperty**」と入力します。
4. [型] 列の [int] を [文字列] で置き換えます。
5. [MyProperty] 列の始めにあるアイコンを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
6. [カスタム属性] プロパティの値を指定するには、省略記号ボタン (...) をクリックして [カスタム属性] ダイアログ ボックスを開きます。
7. 「**Browsable(false)**」と入力し、[OK] をクリックします。

   ソリューション エクスプローラに、カスタム属性を含む、デザイン時属性の .xmta ファイル (DesignTimeAttributes.xmta) が表示されます。

コントロール ライブラリをビルドするには

1. ソリューション エクスプローラで、[UserControl1.cs] を右クリックし、ショートカット メニューの [コードの表示] をクリックします。
2. `System.NotImplementedException` をスローする行をコメント アウトし、`get` アクションとして、代わりに `return "";` を挿入します。
3. [ビルド] メニューの [MyControlLibrary のビルド] をクリックします。

[MyProperty] がプロパティ ブラウザに表示されないことをテストするには

1. ソリューション エクスプローラで、[MyControlLibrary] を右クリックし、ショートカット メニューの [追加] をポイントして、[新しい項目] をクリックします。
2. [新しい項目の追加] ダイアログ ボックスの [Windows フォーム] を選択し、[追加] をクリックします。
3. ツールボックスから、フォームに [UserControl1] をドラッグします。
4. フォームのユーザー コントロール イメージを右クリックし、ショートカット メニューの [プロパティ] をクリックします。

   [MyProperty] は [プロパティ] ブラウザに表示されません。

5. ソリューション エクスプローラで .xmta ファイルをダブルクリックし、false を true で置き換えます。

6. 上記の手順を繰り返して、[プロパティ] グリッドを表示します。今度は [MyProperty] が表示されます。

参照

その他の技術情報

スマート デバイスのチュートリアル

チュートリアル : データベース マスター/詳細アプリケーション

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

このチュートリアルでは、Visual Studio 2005 環境を使用して、データベースに接続し、プロジェクトに含めるデータベース オブジェクトを選択し、データ連結コントロールを作成して、スマート デバイス アプリケーションにデータを表示する方法について説明します。

メモ :
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。 このチュートリアルでは、Visual Basic 開発設定と Visual C# 開発設定を使用しています。

前提条件

SQL Server Mobile Edition 用の Northwind データベース (Visual Studio 2005 に付属)。

Microsoft SQL Server 2005 Compact Edition 用の Northwind データベース (Visual Studio 2005 SP1 に同梱される SQL Server Compact Edition をインストールすると一緒にインストールされる)。

メモ :
開発コンピュータで管理者ではない場合、既定の位置 (\Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SQL Server\Mobile\v3.0) にある Northwind.sdf ファイルを開くことができません。指示に従って、ファイルをデスクトップまたは My Documents にコピーして開いてください。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

対象デバイスの選択

ソリューションを配置するときにデバイス選択のダイアログが表示されるようにするには、次の手順を実行します。

配置時にデバイス選択のダイアログを表示するには

1. [ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックします。次に、[全般] をクリックします。
2. [デバイス プロジェクトの配置前に選択できるデバイスを表示] チェック ボックスをオンにします。

アプリケーションの作成

これは、このチュートリアルのデータ機能をホストする Windows フォームのサンプル アプリケーションです。

Windows フォームのデバイス プロジェクトを作成するには

1. (Visual Basic) Visual Studio 2005 の [ファイル] メニューで、[新しいプロジェクト] をクリックします。

   または

   (Visual C#) Visual Studio 2005 の [ファイル] メニューで、[新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [プロジェクトの種類] で、[Visual Basic] または [Visual C#] を展開し、[スマート デバイス] を展開します。次に、[Pocket PC 2003] をクリックします。

   目的の言語が最初に表示されない場合は、[他の言語] を展開します。この表示は、開発設定で管理されます。
3. [テンプレート] の [デバイス アプリケーション] をクリックします。

メモ :
[デバイス アプリケーション (1.0)] はクリックしないでください。.NET Compact Framework 1 を基にしているため、このチュートリアルには適していません。

4. [プロジェクト名] ボックスに「MasterDetailSample」と入力します。

5. (Visual C# のみ) [場所] ボックスで、プロジェクト ファイルを格納する場所を確認します。
6. [OK] をクリックします。

   Windows フォーム デザイナに Pocket PC デバイスが表示されます。

データ機能の追加

ここでは、次の作業を行います。

- データ ソースの種類の選択
- データ接続の選択と構成
- データベース オブジェクトの選択
- フォームにデータ連結コントロールを追加

データ ソースの種類を選択するには

1. [データ] メニューで、[新しいデータ ソースの追加] をクリックし、データ ソース構成ウィザードを起動します。
2. [データ ソースの種類を選択] ページで、[データベース] を選択し、[次へ] をクリックします。

データ接続の選択と構成を行うには

1. [データ接続の選択] ページの [新しい接続] をクリックします。
2. [データ ソースの選択] ダイアログ ボックスで、[Microsoft SQL Server Mobile Edition] を選択し、[続行] をクリックします。

   または

   [データ ソースの選択] ダイアログ ボックスで、[Microsoft SQL Server Compact Edition] を選択し、[続行] をクリックします。

メモ :
設定や前のプロジェクトによっては、[データ ソースの選択] ダイアログ ボックスではなく、[接続の追加] ダイアログ ボックスが表示されることがあります。この場合、[接続の追加] ダイアログ ボックスの [変更] をクリックすると、[データ ソースの変更] ダイアログ ボックスが表示されます。次に、[Microsoft SQL Server Mobile Edition] または [Microsoft SQL Server Compact Edition] を選択し、[OK] をクリックします。

3. [接続の追加] ダイアログ ボックスで、[マイ コンピュータ] を選択します。
4. [接続の追加] ダイアログ ボックスで、[参照] をクリックします。
5. [SQL Server Mobile Edition のデータベース ファイルを選択してください。] ダイアログ ボックスで、[Northwind.sdf] を選択し、[開く] をクリックします。

   または

   [Microsoft SQL Server 2005 Compact Edition のデータベース ファイルを選択してください。] ダイアログ ボックスで、[Northwind.sdf] を選択し、[開く] をクリックします。
6. [接続の追加] ダイアログ ボックスの [パスワード] ボックスは空のままにします。

   このデータベースにはパスワードがありません。
7. [接続の確認] をクリックして接続を確認します。

メモ :
Northwind.sdf ファイルへのアクセスが拒否された場合、ファイルをデスクトップにコピーして、そのコピーを開きます。この問題は、開発コンピュータ上で既定の位置にあるファイルを開くのに十分なアクセス権がないときに発生します (既定の位置については、このチュートリアルの最初の説明を参照してください)。

8. 接続の成功を示すメッセージ ボックスの [OK] をクリックし、[接続の追加] ダイアログ ボックスの [OK] をクリックして、ダイアログ ボックスを閉じます。
9. [次へ] をクリックし、[データ接続の選択] ページを閉じます。

10. ファイルをプロジェクトにコピーするかどうかを確認するメッセージ ボックスに対して、[はい] をクリックします。

データベース オブジェクトを選択するには

1. [データベース オブジェクトの選択] ページの [Tables] ノードを展開し、[Customers] テーブルと [Orders] テーブルを選択します。
2. [完了] をクリックします。

   NorthwindDataset が作成されます。このデータ ソースは、[データ] メニューの [データ ソースの表示] をクリックすると表示されます。

フォームにデータ バインド コントロールを追加するには

1. [データ ソース] ウィンドウで、[Customers] テーブルを選択し、ドロップダウン矢印をクリックして、[DataGrid] オプションを選択します。
2. デザイナで、[データ ソース] ウィンドウからフォーム上に [Customers] テーブルをドラッグします。

   ウィンドウの上部にグリッド位置を指定します。
3. [データ ソース] ウィンドウで、[Customers] テーブルを展開して、[Orders] テーブルを表示します。

メモ：
これは、[Customers] テーブル内に表示される [Orders] テーブルです。[Customers] テーブルと同じツリー レベルにある [Orders] テーブルではありません。

4. この [Orders] テーブルのドロップダウン矢印をクリックし、[DataGrid] オプションを選択します。
5. デザイナで、[データ ソース] ウィンドウからフォーム上に [Orders] テーブルをドラッグします。

   ウィンドウの下部にグリッド位置を指定します。

アプリケーションのテスト

このセクションでは、アプリケーションを構築して Pocket PC 2003 SE エミュレータにダウンロードし、アプリケーションが正常に機能することを確認します。

アプリケーションをテストするには

1. [デバッグ] メニューの [デバッグなしで開始] または [デバッグ開始] をクリックします。
2. [配置] ダイアログ ボックスで、[Pocket PC 2003 SE エミュレータ] を選択し、[配置] をクリックします。

   配置の進行状況がステータス バーに表示されます。エミュレータへの配置には時間がかかることがあります。
3. アプリケーションをエミュレータで実行する場合、↑キーと↓キーを使用するか、エミュレータの NAVIGATION コントロールを使用して、[Customers] グリッドで選択したレコードを変更します。選択したレコードが [Orders] グリッドで変更されることを確認します。

他のチュートリアルの準備

他のチュートリアルを実行する場合、または他のプロジェクトを開く場合、エミュレータを完全にシャットダウンし、このソリューションを終了します。

エミュレータとソリューションを終了するには

1. エミュレータの [ファイル] メニューで、[終了] をクリックします。
2. [デバイス エミュレータ] メッセージ ボックスで、エミュレータの状態を保存する確認メッセージが表示された場合は、[いいえ] をクリックします。
3. 接続が切断されたことを示すメッセージ ボックスに対して、[OK] をクリックします。
4. (Visual Basic) [ファイル] メニューの [プロジェクトを閉じる] をクリックします。

   プロジェクトまたはソリューションの保存を確認するメッセージが表示された場合、後で再利用するには [保存] をクリックします。保存しない場合は [破棄] をクリックします。

   または

   (Visual C#) [ファイル] メニューの [ソリューションを閉じる] をクリックします。

参照

処理手順

方法 : サンプル データベースをインストールする

関連項目

データ ソース構成ウィザード

その他の技術情報

データに関するチュートリアル

データへのアクセス (Visual Studio)

チュートリアル : パラメータ クエリ アプリケーション

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

このチュートリアルでは、Visual Studio 2005 環境を使用して、簡単なパラメータ クエリ アプリケーションを開発する方法について説明します。データ連結とユーザー インターフェイスは自動的に生成されます。このアプリケーションは既存の Northwind データベースを元に作成され、スマート デバイス ユーザーが [Order Number] しかわからないときに [Shipping Country] を判断できる機能を備えています。ここで構築するアプリケーションには、[Order Number] のユーザー入力機能と、対応する [Shipping Country] の結果表示機能があります。

メモ :

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

このチュートリアルでは、Visual Basic 開発設定と Visual C# 開発設定を使用しています。

次の手順は、Visual Studio 2005 SP1 に合わせて更新されています。

インストールされるデータベースは、どのリリースの Visual Studio がインストールされているかによって異なります。

- Microsoft SQL Server Mobile Edition (Visual Studio 2005)
- Microsoft SQL Server 2005 Compact Edition (Visual Studio 2005 SP1)

前提条件

SQL Server Mobile または SQL Server Compact Edition 用の Northwind データベースは、Visual Studio 2005 に付属しています。

メモ :

開発コンピュータの管理者ではない場合、既定の位置 (\Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SQL Server\Mobile\v3.0) にある Northwind.sdf ファイルを開くことはできません。指示に従って、ファイルをデスクトップまたは [My Documents] にコピーしてから開いてください。

対象デバイスの選択

ソリューションを配置するときにデバイス選択のダイアログが表示されるようにするには、次の手順を実行します。

配置時にデバイス選択のダイアログを表示するには

1. [ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックします。次に、[全般] をクリックします。
2. [デバイス プロジェクトの配置前に選択できるデバイスを表示] チェック ボックスをオンにします。

アプリケーションの作成

これは、このチュートリアルのデータ機能をホストする Windows フォームのサンプル アプリケーションです。

Windows フォームのデバイス プロジェクトを作成するには

1. (Visual Basic) Visual Studio 2005 の [ファイル] メニューで、[新しいプロジェクト] をクリックします。

   または

   (Visual C#) Visual Studio 2005 の [ファイル] メニューで、[新規作成] をポイントし、[プロジェクト] をクリックします。

2. [新しいプロジェクト] ダイアログ ボックスの [プロジェクトの種類] で、[Visual Basic] または [Visual C#] を展開し、[スマート デバイス] を展開します。次に、[Pocket PC 2003] をクリックします。

   目的の言語が最初に表示されない場合は、[他の言語] を展開します。この表示は、ユーザーの開発設定で管理されます。

3. [テンプレート] の [デバイス アプリケーション] をクリックします。

   メモ :

[デバイス アプリケーション (1.0)] はクリックしないでください。.NET Compact Framework 1.0 を元にしているため、このチュートリアルには適していません。

4. [プロジェクト名] ボックスに「**ParamQuerySample**」と入力します。
5. (Visual C# のみ) [場所] ボックスで、プロジェクト ファイルを格納する場所を確認します。
6. [OK] をクリックします。

   Windows フォーム デザイナに Pocket PC デバイスが表示されます。

データ機能の追加

ここでは、次の作業を行います。

- データ ソースの種類の選択。
- データ接続の選択と構成。
- データベース オブジェクトの選択。
- フォームにデータ連結コントロールを追加。

データ ソースの種類を選択するには

1. [データ] メニューで、[新しいデータ ソースの追加] をクリックし、データ ソース構成ウィザードを起動します。
2. [データ ソースの種類を選択] ページで、[データベース] を選択し、[次へ] をクリックします。

データ接続の選択と構成を行うには

1. [データ接続の選択] ページの [新しい接続] をクリックします。
2. [データ ソースの選択] ダイアログ ボックスで、[Microsoft SQL Server Mobile Edition] を選択し、[続行] をクリックします。

   または

   [データ ソースの選択] ダイアログ ボックスで、[Microsoft SQL Server Compact Edition] を選択し、[続行] をクリックします。

メモ :

設定や前のプロジェクトによっては、[データ ソースの選択] ダイアログ ボックスではなく、[接続の追加] ダイアログ ボックスが表示されることがあります。この場合、[接続の追加] ダイアログ ボックスの [変更] をクリックすると、[データ ソースの変更] ダイアログ ボックスが表示されます。次に、[Microsoft SQL Server Mobile Edition] または [Microsoft SQL Server Compact Edition] を選択し、[OK] をクリックします。

3. [接続の追加] ダイアログ ボックスで、[マイ コンピュータ] を選択します。
4. [接続の追加] ダイアログ ボックスで、[参照] をクリックします。
5. [SQL Server Mobile Edition のデータベース ファイルを選択してください。] ダイアログ ボックスで、[Northwind.sdf] を選択し、[開く] をクリックします。

   または

   [Microsoft SQL Server 2005 Compact Edition のデータベース ファイルを選択してください。] ダイアログ ボックスで、[Northwind.sdf] を選択し、[開く] をクリックします。
6. [接続の追加] ダイアログ ボックスの [パスワード] ボックスは空のままにします。

   このデータベースにはパスワードがありません。
7. [接続の確認] をクリックして接続を確認します。

メモ :

Northwind.sdf ファイルへのアクセスが拒否された場合、ファイルをデスクトップにコピーして、そのコピーを開きます。この問題は、開発コンピュータ上で既定の位置にあるファイルを開くのに十分なアクセス権がないときに発生します (既定の位置については、このチュートリアルの最初の説明を参照してください)。

8. 接続の成功を示すメッセージ ボックスの [OK] をクリックし、[接続の追加] ダイアログ ボックスの [OK] をクリックして、ダイアログ ボックスを閉じます。
9. [次へ] をクリックし、[データ接続の選択] ページを閉じます。
10. ファイルをプロジェクトにコピーするかどうかを確認するメッセージ ボックスに対して、[はい] をクリックします。

データベース オブジェクトを選択するには

1. [データベース オブジェクトの選択] ページの [Tables] ノードを展開し、[Orders] テーブルを選択します。
2. [完了] をクリックします。

   NorthwindDataset が作成されます。このデータ ソースは、[データ] メニューの [データ ソースの表示] をクリックすると表示されます。

クエリを作成するには

1. [データ ソース] ウィンドウで、[Orders] テーブルを展開します。
2. [Ship Country] 列をクリックし、ドロップダウン矢印をクリックして、[ラベル] オプションを選択します。
3. [Ship Country] 列をデザイナのフォームにドラッグします。
4. デザイナのラベル コントロールで、スマート タグをクリックし、ショートカット メニューの [クエリの追加] をクリックします。
5. [検索条件ビルダ] ダイアログ ボックスで、[クエリ ビルダ] をクリックします。
6. [Order ID] 行の [フィルタ] 列に、疑問符 (**?**) を入力します。

   この記号は、アプリケーションのユーザーが [Order ID] の値を入力する必要があることを示します。
7. [OK] をクリックします。

   [クエリ テキスト] ボックスの `WHERE` 句は、`([Order ID]=@PARAM1)` になります。
8. [OK] をクリックして [検索条件ビルダ] ダイアログ ボックスを閉じます。

   パネルがデザイナのフォーム上に表示されます。

ユーザー インターフェイスを調整するには

1. デザイナの [PARAM1] ラベル コントロールを右クリックし、ショートカット メニューの [プロパティ] をクリックします。
2. [Text] プロパティを「**Order ID**」に変更します。
3. [FillBy] をクリックし、テキストのプロパティを [Show country] に変更します。
4. スクロール バーが表示されないようにパネルとコントロールを展開し、すべてのテキストを表示します。Ship_CountryLabel とそのテキスト ボックスが FillByPanel とそのコントロールに隠れないように注意してください。

アプリケーションのテスト

このセクションでは、アプリケーションを構築して Pocket PC 2003 SE エミュレータにダウンロードし、アプリケーションが正常に機能することを確認します。

アプリケーションをテストするには

1. [デバッグ] メニューの [開始] または [デバッグ開始] をクリックします。
2. [配置] ダイアログ ボックスで、[Pocket PC 2003 SE エミュレータ] を選択し、[配置] をクリックします。

   配置の進行状況がステータス バーに表示されます。エミュレータへの配置には時間がかかることがあります。
3. アプリケーションをエミュレータで実行するときに、Order Numberを入力します。Northwind データベースでこの番号は、10000 ～

11077 です。次に [Show country] をクリックします。

その注文の [Ship Country] がラベル コントロールに表示されます。

他のチュートリアルの準備

他のチュートリアルを実行する場合、または他のプロジェクトを開く場合、エミュレータを完全にシャットダウンし、このソリューションを終了できます。

エミュレータとソリューションを終了するには

1. エミュレータの [ファイル] メニューで、[終了] をクリックします。
2. [デバイス エミュレータ] メッセージ ボックスで、エミュレータの状態を保存する確認メッセージが表示された場合は、[いいえ] をクリックします。
3. (Visual Basic) [ファイル] メニューの [プロジェクトを閉じる] をクリックします。

   または

   (Visual C#) [ファイル] メニューの [ソリューションを閉じる] をクリックします。

   プロジェクトまたはソリューションの保存を確認するメッセージが表示された場合、後で再利用するには [保存] をクリックします。保存しない場合は [破棄] をクリックします。

参照

処理手順

方法 : パラメータ付きクエリを作成する (デバイス)

方法 : サンプル データベースをインストールする

関連項目

データ ソース構成ウィザード

その他の技術情報

スマート デバイスのチュートリアル

マネージ デバイス プロジェクトのデータ

チュートリアル : Hello World: スマート デバイスの COM 相互運用の例

このチュートリアルでは、簡単な COM オブジェクトとマネージ アプリケーションを 1 つのソリューションとして結合します。

メモ :
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。 このチュートリアルは、Visual C++ 開発設定を使用して記述されています。

COM オブジェクトの作成

スマート デバイスの ATL プロジェクトを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックして、[プロジェクトの種類] ペインの Visual C++ ノードを展開します。次に、[スマート デバイス] をクリックします。
2. [テンプレート] ペインの [ATL スマート デバイス プロジェクト] をクリックします。
3. [プロジェクト名] ボックスに「**HelloCOMObject**」と入力します。
4. [ソリューション名] ボックスに「**InteropSolution**」と入力します。
5. [OK] をクリックして、ATL スマート デバイス プロジェクト ウィザードを起動します。
6. [完了] をクリックして、ウィザードを閉じます。

   このチュートリアルでは、ウィザードで既定の設定を変更する必要はありません。

クラスを追加するには

1. ソリューション エクスプローラ で、[HelloCOMObject] プロジェクトを右クリックし、[追加] をポイントします。次に、[クラス] をクリックして [クラスの追加] ダイアログ ボックスを開きます。
2. [カテゴリ] ペインで、[スマート デバイス] をクリックします。
3. [テンプレート] ペインで、[ATL シンプル オブジェクト] をクリックし、[追加] をクリックして ATL シンプル オブジェクト ウィザードを開きます。
4. [短い名前] ボックスに「**Hello**」と入力します。
5. 左ペインで、[オプション] をクリックして [オプション] ページを開きます。
6. [スレッド モデル] の下で、[フリー] をクリックし、[完了] をクリックします。

クラスにメソッドを追加するには

1. デスクトップのタブから、または [表示] メニューから [クラス ビュー] ウィンドウを開きます。
2. [HelloCOMObject] を展開し、[IHello] インターフェイスを表示します。
3. [IHello] を右クリックし、[追加] をポイントします。次に、[メソッドの追加] をクリックしてメソッド追加ウィザードを開きます。
4. [メソッド名] ボックスに「**HelloWorld**」と入力します。
5. [パラメータの型] ボックスで、[BSTR*] を選択します。
6. [パラメータ名] ボックスに、「**text**」と入力します。
7. [パラメータの属性] の下で、[out] を選択します。
8. [追加] をクリックします。

   メソッド ボックスに `[out] BSTR* text` と表示されます。
9. [完了] をクリックして、メソッド追加ウィザードを閉じます。

   `STDMETHOD(HelloWorld)(BSTR* text)` メソッドが Hello.h ファイルに表示されます。

メソッドに実装を追加するには

1. ソリューション エクスプローラで、Hello.cpp をダブルクリックしてコード エディタでファイルを開きます。
2. `STDMETHODIMP` セクションで、return ステートメントの前に次の実装コードを挿入します。

```
*text = SysAllocString(L"Hello World!");
```

3. [ビルド] メニューの [HelloCOMObject のビルド] をクリックします。

   COM オブジェクトがソリューションの一部となり、チュートリアルの最初の部分が完了しました。

マネージ プロジェクトの作成

ソリューションにマネージ プロジェクトを追加するには

1. ソリューション エクスプローラの [InteropSolution] を右クリックし、[追加] をポイントして [新しいプロジェクト] をクリックします。
2. [プロジェクトの種類] ペインで、Visual C# ノードに移動して展開し、[スマート デバイス] ノードを展開します。次に、[Pocket PC 2003] をクリックします。
3. [テンプレート] ペインの [デバイス アプリケーション] をクリックします。
4. [プロジェクト名] ボックスに「**SayHello**」と入力し、[OK] をクリックします。

   SayHello マネージ プロジェクトがソリューションの一部として作成され、Pocket PC フォームがデザイナ ウィンドウに表示されます。

マネージ プロジェクトで、COM オブジェクトを参照として追加します。

マネージ プロジェクトで COM オブジェクトを参照として追加するには

1. ソリューション エクスプローラで、[SayHello] プロジェクトを右クリックし、ショートカット メニューの [参照の追加] をクリックします。
2. [参照の追加] ダイアログ ボックスで、[参照] をクリックします。

   SayHello フォルダが [ファイルの場所] ボックスに表示されます。
3. 親フォルダに移動します (このチュートリアルでは、InteropSolution です)。
4. フォルダの内容が示されているウィンドウで、[HelloCOMObject] をダブルクリックし、[Pocket PC 2003 (ARMV4)] をダブルクリックします。次に、[Debug] をダブルクリックし、HelloCOMObject.dll をクリックします。
5. [OK] をクリックして、[参照の追加] ダイアログ ボックスを閉じます。
6. ソリューション エクスプローラで Form1.cs を右クリックし、ショートカット メニューの [コードの表示] をクリックします。
7. ファイルの上部の `Using directives` 領域に、次のコードを追加します。

```
using HelloCOMObjectLib;
```

マネージ プロジェクトへのイベント処理の追加

マネージ プロジェクトにイベント処理を追加して、構築するには

1. Form1 デザイナを開きます。
2. ツールボックスから、フォームにボタンをドラッグします。
3. ボタンをダブルクリックして、クリック イベント時にコード エディタを開きます。
4. ボタンの次のイベント処理コードを挿入します。

```
string text;
HelloClass h = new HelloClass();
h.HelloWorld(out text);
MessageBox.Show(text);
```

5. [ビルド] メニューの [SayHello のビルド] をクリックします。

ソリューションの最終調整

ソリューションを配置用に構成するには

1. ソリューション エクスプローラで、[SayHello] プロジェクトを右クリックし、ショートカット メニューの [スタートアップ プロジェクトに設定] をクリックします。
2. ソリューション エクスプローラで、[InteropSolution] を右クリックし、ショートカット メニューの [プロジェクト依存関係] をクリックします。
3. [プロジェクト依存関係] ダイアログ ボックスの [プロジェクト] ボックスの [SayHello] を選択し、[依存先] ボックスの [HelloCOMObject] を選択します。
4. [OK] をクリックします。

   これで、ソリューションを配置できるようになります。

構成されたソリューションの配置

ソリューションを配置するには

1. [デバッグ] メニューの [デバッグなしで開始] をクリックします。
2. [配置] ダイアログ ボックスで、[Pocket PC 2003 SE エミュレータ] を選択し、[配置] をクリックします。

「チュートリアル : マネージ コードとネイティブ コードの両方を含むソリューションのデバッグ」で使用するために、このソリューションを保存します。

参照
概念
デバイスの COM 相互運用性
その他の技術情報
スマート デバイスのチュートリアル

方法：マルチプラットフォーム対応のデバイス プロジェクトを作成する (Visual C++)

デバイス用の C++ を使用して、同じ開発プロジェクトから複数のデバイスを対象にできます。Visual Studio 2005 を使用すると、1 つのデバイス プロジェクトで複数のデバイス プラットフォーム (Pocket PC と Smartphone など) を対象にでき、デバイス プロジェクトをデスクトップ プロジェクトに追加できます。詳細については、「デスクトップ プロジェクトのデバイス サポート」を参照してください。

デバイス プロジェクトで複数のプラットフォームを対象にするには、2 つの方法があります。

1. プロジェクトを作成するときに、スマート デバイス プロジェクト ウィザードを使用する方法。もう 1 つの方法より効率的な技術です。

   選択したデバイス プラットフォームすべてについて、スマート デバイス プロジェクト ウィザードでプロジェクト ファイルが生成されます (ソース、ヘッダー、リソースの各ファイルなど)。プラットフォームの SDK または デバイスの SDK をインストールすることで、追加で新しいプラットフォームをインストールすることもできます。

   Visual Studio 2005 を使用すると、Smartphone や Pocket PC 2003 以降だけでなく、任意の Windows CE デバイス バージョン 5.0 以降もサポートする任意のデバイス SDK をインストールできます。詳細については、「方法 : ウィザードを使用してマルチプラットフォーム対応のデバイス プロジェクトを作成する」を参照してください。

2. プロジェクトを作成した後に、[構成マネージャ] (スマート デバイス プロジェクト ウィザード) ダイアログ ボックスの [新しいプラットフォーム] ペインを使用する方法。詳細については、「方法 : 新しいプラットフォームをデバイス プロジェクトに追加する」を参照してください。

参照

処理手順

方法 : 既定のデバイスを変更する (ネイティブ プロジェクト)

複数のプラットフォームでのリソースの使用

その他の技術情報

How to Maintain a Single Binary for Pocket PC and Smartphone

方法：ウィザードを使用してマルチプラットフォーム対応のデバイス プロジェクトを作成する

プロジェクトの作成後にプラットフォームを追加するのではなく、プロジェクトの作成時にスマート デバイス アプリケーション ウィザードを使用してマルチプラットフォーム対応のデバイス プロジェクトを作成することにより、さまざまな利点がもたらされます。これらの利点は次のとおりです。

- アプリケーション ウィザードの [プラットフォーム] ページで複数のプラットフォームを選択すると、プラットフォームごとにリソース ファイルが生成および設定されます。一方、プロジェクトの作成後にプラットフォームを追加する場合は、手動でプラットフォームおよびリソース ファイルを追加する必要があります。詳細については、「複数のプラットフォームでのリソースの使用」を参照してください。
- アプリケーション ウィザードの [プラットフォーム] ページで複数のプラットフォームを選択すると、マルチプラットフォーム対応のコーディングに適合するように、生成されるファイルのソースに `#if defined` ステートメントが埋め込まれます。一方、プロジェクトの作成後にプラットフォームを追加する場合は、コードの適切なセクションに手動で `#if defined` ステートメントを追加する必要があります。

次の手順で、各技術の使用方法について説明します。詳細については、「コードの説明 : Visual C++ デバイス プロジェクトのウィザード生成コード」を参照してください。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

新しいプロジェクト ウィザードを使用してマルチプラットフォーム対応のデバイス プロジェクトを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。

   [新しいプロジェクト] ダイアログ ボックスが表示されます。

2. [プロジェクトの種類] の [Visual C++] ノードを展開し、[スマート デバイス] をクリックします。次に、[MFC スマート デバイス アプリケーション] または使用できる別のプロジェクトの種類をクリックします。

3. [プロジェクト名] ボックスに「**MultiPlatformProject**」と入力します。

4. [場所] ボックスで、プロジェクト ファイルを格納する場所を確認し、[OK] をクリックします。

   <プロジェクトの種類> スマート デバイス アプリケーション ウィザードが表示されます。

5. [次へ] をクリックし、[プラットフォーム] ペインで、[Pocket PC 2003] や [Smartphone 2003] など、対象にするプラットフォームを選択します。

6. [完了] をクリックします。

   プロジェクトがソリューション エクスプローラに表示されます。

構成マネージャを使用して新しいデバイス構成を追加するには

1. Visual Studio で、[ビルド] メニューの [構成マネージャ] をクリックします。

   [構成マネージャ] ダイアログ ボックスが表示されます。

2. [アクティブ ソリューション構成] ボックスの一覧の [新規作成] を選択し、新しい構成を追加します。

   [新しいソリューション構成] ダイアログ ボックスが表示されます。

3. [名前] ボックスに、新しい構成の名前を入力します。

4. [設定のコピー元] ボックスに、既存の構成を指定します。

5. [OK] をクリックします。

参照

処理手順

方法 : 新しいプラットフォームをデバイス プロジェクトに追加する

方法 : **既定のデバイスを変更する (ネイティブ プロジェクト)**
複数のプラットフォームでのリソースの使用
その他の技術情報
How to Maintain a Single Binary for Pocket PC and Smartphone

方法：新しいプラットフォームをデバイス プロジェクトに追加する

次の手順では、既存のデバイス プロジェクトにプラットフォームを追加する方法について説明します。

これは、ウィザードを使用して最初からマルチプラットフォーム対応のプロジェクトを作成する方法に代わる手段です。ウィザードを使用する方法をお勧めします。詳細については、「方法 : ウィザードを使用してマルチプラットフォーム対応のデバイス プロジェクトを作成する」を参照してください。

メモ :

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

構成マネージャを使用して新しいプラットフォームを追加するには

1. デバイス プロジェクトで、[ビルド] メニューの [構成マネージャ] をクリックします。

   [構成マネージャ] ダイアログ ボックスが表示されます。

2. [アクティブ ソリューション プラットフォーム] ボックスの一覧の [新規作成] をクリックし、新しい構成を追加します。

   [新しいソリューション プラットフォーム] ダイアログ ボックスが表示されます。

3. [名前] で、[Smartphone 2003] など、追加するプラットフォームの名前を選択するか入力します。.

4. [設定のコピー元] ボックスの一覧で、既存のプラットフォームを選択してその設定を新しいプラットフォームにコピーするか、空の設定を使用して新しいプラットフォームを作成します。後者の場合は、プロジェクトの設定を手動で行う必要があります。

5. [新しいプロジェクト プラットフォームを作成する] チェック ボックスがオンになっていることを確認します。

   オフの場合は、構成ファイル、ウィザードで生成されるプラットフォームに依存するコード、および他のプロジェクト ファイル (リソースなど) を管理および追加する必要があります。

6. [OK] をクリックします。

   新しく追加されたプラットフォームについて、[新しいソリューション プラットフォーム] ダイアログ ボックスでプロジェクト ファイルが生成されます (ソース、ヘッダー、リソースの各ファイルなど)。

7. [OK] をクリックして [構成マネージャ] ダイアログ ボックスを閉じます。

   メモ :

   デバイス プロジェクトが作成されたため、これ以降はいつでもプラットフォームを追加できます。ただし、プロジェクトの作成後に新しいプラットフォームを追加しても、依存ランタイム DLL は [追加ファイル] 構成プロパティに追加されません。たとえば、アプリケーションで動的に MFC にリンクしている場合は、新しく追加したプラットフォームの [追加ファイル] プロパティに `Mfc80u.dll`、`Atl80.dll`、および `Msvcr80.dll` の各 DLL を追加する必要があります。この例では、リテール構成を前提としています。

参照

処理手順

方法 : 既定のデバイスを変更する (ネイティブ プロジェクト)

複数のプラットフォームでのリソースの使用

チュートリアル : スマート デバイス用マルチプラットフォーム MFC アプリケーションの作成

Visual C++ を使用して、複数デバイスを対象としたコードを記述できます。次のチュートリアルでは、マルチプラットフォーム対応の MFC アプリケーションを構築する方法について説明しています。詳細については、「MFC スマート デバイス アプリケーション ウィザード」を参照してください。

MFC マルチプラットフォーム プロジェクトの作成

このチュートリアルは、主に次の 3 つの手順で構成されています。

- マルチプラットフォーム対応のスマート デバイス MFC プロジェクトを作成します。
- マルチプラットフォーム **OnDraw()** メソッドにコードを追加します。
- マルチプラットフォーム ソリューションを配置します。

メモ :
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。 このチュートリアルは、Visual C++ 開発設定を使用して記述されています。

マルチプラットフォーム対応のスマート デバイス MFC プロジェクトを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックして、[プロジェクトの種類] ペインの Visual C++ ノードを展開します。次に、[スマート デバイス] をクリックします。
2. [テンプレート] ペインの [MFC スマート デバイス アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「HelloMFC」と入力します。
4. [OK] をクリックして、MFC スマート デバイス アプリケーションウィザードを起動します。
5. [次へ] をクリックし、現在のプロジェクトに追加するプラットフォームの SDK を選択します。
6. [インストール済み SDK] ペインから、Smartphone 2003 や Pocket PC 2003 など、現在のプロジェクトに追加するプラットフォームを選択します。
7. [次へ] をクリックして [アプリケーションの種類] ページを開きます。
8. [シングル ドキュメント] チェック ボックスと [スタティック ライブラリで MFC を使用する] チェック ボックスをオンにします。[ドキュメント ビュー アーキテクチャ サポート] のチェック ボックスはオンのままにします。
9. ウィザードを完了して閉じるには [完了] をクリックします。ウィザードの残りのオプションについて既定の設定を受け入れる場合は [次へ] をクリックします。

メモ :
デバイス プロジェクトは作成されているため、最初の作成後にいつでもプラットフォームを追加できます。ただし、最初の作成後に、新しいプラットフォームをプロジェクトに追加しても、追加されたプラットフォームの [追加ファイル] 構成プロパティに、依存するランタイム DLL は追加されません。たとえば、アプリケーションで動的に MFC にリンクしている場合、追加した新しいプラットフォームの設定の [追加ファイル] プロパティに、DLL を追加する必要があります (Mfc80u.dll、Atl80.dll、Msvcr80.dll)。このサンプルでは、リテール構成を前提としています。

マルチプラットフォーム の OnDraw() メソッドにコードを追加する

OnDraw() メソッドにコードを追加するには

1. ソリューション エクスプローラで、[ソース ファイル] ノードを展開します。HelloMFCView.cpp をダブルクリックし、エディタでソース ファイルを開きます。
2. `OnDraw` シグネチャを変更し、`OnDraw(CDC* pDC)` メソッドの `pDC` をコメントから外します。結果として、次のようなコードになります。

```
```

```
void CHelloMFCView::OnDraw(CDC* pDC)
```

3. OnDraw メソッドの //TODO コメントの後に、次のコードを挿入します。

```
// Define a rectangle to draw on the screen.
CRect rect;
// Use the client area of the MFC form for drawing.
    GetClientRect(&rect);
    // Draw the text on the screen.
    pDC->DrawTextW(_T("Hello World"),11, &rect,1);
```

4. [ビルド] メニューの [ソリューションのリビルド] をクリックします。

対象デバイスの選択

ソリューションを配置するときにデバイス選択のダイアログが表示されるようにするには、次の手順を実行します。

配置時にデバイス選択のダイアログを表示するには

1. [ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックします。次に、[全般] をクリックします。

   [デバイス ツール] が表示されない場合は、[オプション] ダイアログ ボックスの下部にある [すべての設定を表示] を選択します。

2. [デバイス プロジェクトの配置前に選択できるデバイスを表示] チェック ボックスをオンにします。

マルチプラットフォーム MFC ソリューションの配置

ソリューションを配置するには

1. Visual Studio ツール バーにある [ターゲット デバイス] ボックスの一覧で、[Pocket PC 2003 SE エミュレータ] または [Pocket PC 2003 デバイス] などの対象を選択します。

2. [ビルド] メニューの [配置] をクリックします。

このチュートリアルで生成したコードの詳細については、「コードの説明 : Hello World: スマート デバイス用マルチプラットフォーム MFC アプリケーション」を参照してください。

参照

その他の技術情報

スマート デバイスのチュートリアル

方法 : マルチプラットフォーム対応のデバイス プロジェクトを作成する (Visual C++)

コードの説明 : Hello World: スマート デバイス用マルチプラットフォーム MFC アプリケーション

Visual Studio C++ を使用して Windows CE (Mobile) やその他の一般的なモバイル デバイスを対象とする場合でも、eMbedded Visual C++ を使用してデバイス アプリケーションを開発する場合でも、C++ スマート デバイス用マルチプラットフォーム MFC アプリケーション ウィザードを使用することで、プロジェクト ファイルを生成するための日常的タスクの大部分を簡略化できます。このようなファイルの中には、リソース ファイルとマルチプラットフォーム プロジェクト構成ファイルが含まれます。このウィザードにより、すぐに使用できるコードが生成されるので、中核となるビジネス アプリケーション機能の開発に集中できます。

このチュートリアルでは、スマート デバイス用マルチプラットフォーム MFC アプリケーション ウィザードによって自動生成されるコードについて説明します。これにより、必要に応じてアプリケーションを簡単に拡張したり修正したりできるようになります。

詳細については、「チュートリアル : スマート デバイス用マルチプラットフォーム MFC アプリケーションの作成」、「デバイス用の MFC リファレンス」、および「MFC リファレンス」を参照してください。

スマート デバイスの C++ 向けマルチプラットフォーム MFC アプリケーションのコード リスト

このウィザード生成コードには、次の内容が含まれています。

- **include** セクション。

```
#include "stdafx.h"
```

- stdafx.h。詳細については、「プリコンパイル済みヘッダー ファイル」を参照してください。

```
#include "MFCHello1.h"
#include "MFCHello1Doc.h"
#include "MFCHello1View.h"
```

- デバッグ構成。

```
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

IMPLEMENT_DYNCREATE(CMFCHello1View, CView)
```

- MFC のメッセージ マップと具体的な BEGIN_MESSAGE_MAP。
- IMPLEMENT_DYNCREATE。詳細については「IMPLEMENT_DYNCREATE」を参照してください。

```
BEGIN_MESSAGE_MAP(CMFCHello1View, CView)
END_MESSAGE_MAP()
```

- 構築用コード。詳細については、「コンストラクタのデザイン」を参照してください。また例外処理の詳細については、「例外処理」を参照してください。

```
// CMFCHello1View construction/destruction
CMFCHello1View::CMFCHello1View()
{
        // TODO: add construction code here.
}
CMFCHello1View::~CMFCHello1View()
{
}
```

- MFC PreCreateWindow。

```
BOOL CMFCHello1View::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs.

        return CView::PreCreateWindow(cs);
}
```

- **画面の描画** (ペイント) **は、**CView::OnDraw **メソッドで発生します。このメソッドでは、パラメータとして渡されるグラフィカル デバイス インターフェイス** (GDI: Graphical Device Interface) **を制御できます** (**コード内の** `CDC* pDC`)**。忘れずにコメントから外してください。たとえば、アプリケーション内のアニメーション ゲームのテキストから描画グラフィックスにいたるまで、あらゆるものに** GDI **を活用できます。この** GDI **サンプルでは、**CDC::DrawText **メソッドを使用して、**CWnd::GetClientRect **で定義された四角形の中に** "Hellow World" **というテキストを描画します。**

- BEGIN_MESSAGE_MAP。
- CDC **クラス。**
- GetClientRect。

```
// CMFCHello1View drawing
void CMFCHello1View::OnDraw(CDC* pDC)
{
        CMFCHello1Doc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        // TODO: add draw code for native data here.
        CRect rect;
      GetClientRect(&rect);
     // Length and string to draw are hard coded for simplicity of
     // example.
```

- DrawText **メソッドとその他の関連メソッド。詳細については、「**CDC **クラス」を参照してください。**

```
        pDC->DrawTextW(_T("Hello World"),11, &rect,1);
        // nCount ( set to 11) can be a -1, then
        //lpszString is assumed to be
        // a long pointer to a null-terminated string
        // and DrawText method automatically
        // computes the character count.
}
// CMFCHello1View diagnostics
#ifdef _DEBUG
void CMFCHello1View::AssertValid() const
{
        CView::AssertValid();
}
#ifndef _WIN32_WCE
void CMFCHello1View::Dump(CDumpContext& dc) const
{
        CView::Dump(dc);
}
#endif // !_WIN32_WCE
CMFCHello1Doc* CMFCHello1View::GetDocument() const
// non-debug version is inline
{
```

```
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMFCHello1Doc)));
        return (CMFCHello1Doc*)m_pDocument;
}
#endif //_DEBUG
// CMFCHello1View message handlers
```

ウィザードで作成される readme ファイルで使用できる情報

アプリケーション ウィザードにより、この MFCHello1 アプリケーションが作成されます。このアプリケーションは、Microsoft Foundation Class の基本的な使用方法を示すだけでなく、独自のアプリケーションを作成するための開始点にもなります。

ここでは、ウィザードで生成されるファイルの一覧を示し、各ファイルの概要を説明します。これらのファイルを開始点として、MFC アプリケーションを簡単に開発できます。

ここでは HelloMFC という名前を例として使用します。実際のプロジェクト名に置き換えて読んでください。

HelloMFC.vcproj

アプリケーション ウィザードで生成される VC++ プロジェクトのメイン プロジェクト ファイルです。ファイルを生成した Visual C++ のバージョン情報と、アプリケーション ウィザードで選択したプラットフォーム、構成、およびプロジェクトの機能に関する情報が記述されています。

HelloMFC.h

アプリケーションのメイン ヘッダー ファイルです。その他のプロジェクト固有のヘッダーを含んでおり、CMFCHello1App アプリケーション クラスを宣言しています。

HelloMFC.cpp

アプリケーションのメイン ソース ファイルです。CMFCHello1App アプリケーション クラスのクラス定義を含みます。

HelloMFCppc.rc

プロジェクトのメイン リソース ファイルです。*Pocket PC platform*、または同じユーザー インターフェイス モデルをサポートするプラットフォームをコンパイルするときに、プロジェクトで使用するすべての Microsoft Windows リソースの一覧を含みます。RES サブディレクトリに格納されているアイコン、ビットマップ、カーソルを含みます。このファイルは、Microsoft Visual C++ 内で直接編集できます。プロジェクト リソースは 1033 に格納されます。.rc ファイルを保存すると、データ セクション内の定義が、定義の表示名ではなく、定義済みの 16 進数値のバージョンで保存されます。

res\HelloMFCppc.rc2

Microsoft Visual C++ で編集されないリソースを含んだファイルです。リソース エディタで編集できないリソースはすべてこのファイルに含めます。

HelloMFCsp.rc

プロジェクトのメイン リソース ファイルです。*Smartphone platform*、または同じユーザー インターフェイス モデルをサポートするプラットフォームをコンパイルするときに、プロジェクトで使用するすべての Microsoft Windows リソースの一覧を含みます。RES サブディレクトリに格納されているアイコン、ビットマップ、カーソルを含みます。このファイルは、Microsoft Visual C++ 内で直接編集できます。プロジェクト リソースは 1033 に格納されます。.rc ファイルを保存すると、データ セクション内の定義が、定義の表示名ではなく、定義済みの 16 進数値のバージョンで保存されます。

res\HelloMFCsp.rc2

Microsoft Visual C++ で編集されないリソースを含んだファイルです。リソース エディタで編集できないリソースはすべてこのファイルに含めます。

res\HelloMFC.ico

アプリケーションのアイコンとして使用されるアイコン ファイルです。このアイコンは、メイン リソース ファイルに含まれています。

MainFrm.h、MainFrm.cpp

CMainFrame フレーム クラスを含むファイルです。このクラスは CFrameWnd から派生したもので、SDI フレームのすべての機能を制御します。

さらにアプリケーション ウィザードは、次に示す MFC ドキュメント型と MFC ビューを作成します。

- **HelloMFCDoc.h、HelloMFCDoc.cpp**

  HelloMFCDoc クラスを含むファイルです。特殊なドキュメント データの追加や、ファイルの保存および読み込みの実装 (CMFCHello1Doc::Serialize を使用) を行うには、これらのファイルを編集します。

- **HelloMFCView.h、HelloMFCView.cpp**

  HelloMFCView クラスを含むファイルです。HelloMFCView オブジェクトは、HelloMFCDoc オブジェクトを表示するために使用されます。

StdAfx.h、StdAfx.cpp

HelloMFC.pch というプリコンパイル済みヘッダー (PCH) ファイルと、StdAfx.obj というプリコンパイル済み型ファイルをビルドするために使用されます。

Resourceppc.h と Resourcesp.h

標準ヘッダー ファイルです。新しいリソース ID を定義します。Microsoft Visual C++ はこのファイルを読み取って更新します。

アプリケーション ウィザードは、追加やカスタマイズが必要なソース コードの部分を `TODO:` で示します。

アプリケーションが共有 DLL で MFC を使用していて、オペレーティング システムの現在の言語とは異なる言語を使用している場合は、対応するローカライズされたリソースの MFC80XXX.DLL をアプリケーション ディレクトリにコピーしてください。この名前の "XXX" は、言語の省略形を表しています。たとえば MFC80DEU.DLL には、ドイツ語に翻訳されたリソースが含まれています。そうしないと、アプリケーションの UI 要素の一部が、オペレーティング システムの言語のまま残されることがあります。

参照

その他の技術情報

サンプルとチュートリアル (スマート デバイス プロジェクト)

チュートリアル : スマート デバイス用 MFC マルチプラットフォーム ActiveX コントロールの作成

Visual C++ を使用して、複数デバイスを対象とした MFC ActiveX コントロールのコードを記述できます。このチュートリアルでは、複数デバイスで使用できるように C++ マルチプラットフォームの MFC ActiveX コントロールをビルドする方法について説明します。

MFC ActiveX マルチプラットフォーム コントロール プロジェクトの作成

このチュートリアルは、主に次の 3 つの手順で構成されています。

- マルチプラットフォームのスマート デバイス MFC ActiveX コントロール プロジェクトの作成
- MFC ActiveX コントロールの `OnDraw()` メソッドへのコードの追加
- マルチプラットフォームの MFC ActiveX コントロール ソリューションのテスト配置

詳細については、「MFC スマート デバイス ActiveX コントロール ウィザード」を参照してください。

> **メモ :**
>
> 使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

このチュートリアルは、Visual C++ 開発設定を使用して記述されています。

マルチプラットフォームのスマート デバイス MFC ActiveX コントロール プロジェクトを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックして、[プロジェクトの種類] ペインの Visual C++ ノードを展開します。次に、[スマート デバイス] をクリックします。
2. [テンプレート] ペインの [MFC スマート デバイス ActiveX コントロール] をクリックします。
3. [プロジェクト名] ボックスに「**MFCAX**」と入力します。
4. [ソリューション名] ボックスでは、既定の [ソリューションのディレクトリを作成] オプションをそのまま使用します。
5. [OK] をクリックすると、MFC スマート デバイス ActiveX コントロール ウィザードが起動します。
6. [MFC スマート デバイス ActiveX コントロール ウィザード - 概要] ページで、[次へ] をクリックします。[プラットフォーム] (MFC スマート デバイス ActiveX コントロール ウィザード)が表示されるので、現在のプロジェクトに追加するプラットフォームを選択します。

   インストール済み SDK ペインから、Smartphone 2003 や Pocket PC 2003 など、現在のプロジェクトに追加する対象プラットフォームを選択します。プラットフォームを追加するには、左ペインで Smartphone 2003 などのプラットフォームを選択し、右矢印のボタン ([>]) をクリックします。プラットフォームを削除するには、右ペインで Pocket PC 2003 などのプラットフォームを選択し、左矢印のボタン ([<]) をクリックします。
7. ウィザードを完了して閉じるには [完了] をクリックします。ウィザードの残りのオプションについて既定の設定を受け入れる場合は [次へ] をクリックします。

   > **メモ :**
   >
   > デバイス プロジェクトが作成されたため、これ以降はいつでもプラットフォームを追加できます。ただし、新しいプラットフォームを既存のプロジェクトに追加しても、依存するランタイム DLL が [追加ファイル] 構成プロパティに新しく追加されるわけではありません。たとえば、アプリケーションが動的に MFC にリンクしている場合は、新しく追加したプラットフォームの 追加ファイル プロパティに Mfc80u.dll、Atl80.dll、Msvcr80.dll の各 DLL を追加する必要があります。このサンプルでは、リテール構成を前提としています。
8.

マルチプラットフォーム MFC コントロールの OnDraw() メソッドへのコード追加

MFC ActiveX コントロールの OnDraw メソッドにコードを追加するには

1. ソリューション エクスプローラの [ソース ファイル] ノードを展開し、エディタで MFCAXCtrl.cpp ソース ファイルを選択して開きます。
2. `OnDraw` メソッドのコードを次のコードで置き換えます。重要なのは最後の 3 行です。

```
void CMFCAXCtrl::OnDraw(
    CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
    if (!pdc)
        return;

    CRect rect;
    GetClientRect(&rect);
    pdc->DrawTextW(_T("Hello World"),11, &rect,1);
}
```

3. [ビルド] メニューの [ソリューションのリビルド] をクリックします。

マルチプラットフォーム ソリューションの配置

ソリューションを配置するには

1. 配置したソリューションを実行するには、まず対象デバイスに ActiveX コントロール プロジェクトを配置し、登録します。
2. Visual Studio ツール バーにある [ターゲット デバイス] ボックスの一覧で、[Pocket PC 2003 SE エミュレータ] または [Pocket PC 2003 デバイス] などの対象を選択します。
3. [ビルド] メニューの [配置] をクリックします。

対象デバイスの選択

ソリューションを配置するときにデバイス選択のダイアログが表示されるようにするには、次の手順を実行します。

配置時にデバイス選択のダイアログを表示するには

1. [ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックします。次に、[全般] をクリックします[デバイス ツール] が表示されない場合は、[オプション] ダイアログ ボックスの下部にある [すべての設定を表示] を選択します。
2. [デバイス プロジェクトの配置前に選択できるデバイスを表示] チェック ボックスをオンにします。

詳細については、「Mobile Developer Center」を参照してください。

参照

その他の技術情報

スマート デバイスのチュートリアル

方法 : マルチプラットフォーム対応のデバイス プロジェクトを作成する (Visual C++)

コードの説明 : Hello World : スマート デバイスのマルチプラットフォーム ActiveX コントロール

C++ スマート デバイス用 MFC ActiveX コントロール ウィザードは、スマート デバイス プロジェクト用のマルチプラットフォーム MFC ActiveX コントロール (リソース ファイルやマルチプラットフォーム プロジェクト構成ファイルなど) を生成するための日常的タスクの大部分を自動化します。このウィザードは一連の基本ファイルを生成します。

このチュートリアルでは、スマート デバイス用マルチプラットフォーム MFC ActiveX コントロール ウィザードを使用したときに自動生成されるコードについて説明します。これにより、必要に応じてアプリケーションを簡単に拡張したり変更したりできるようになります。

コードのチュートリアル

MFC ActiveX コントロールのプログラミングや ActiveX のプログラミングに慣れている開発者ならば、ウィザードで生成された main 関数を簡単に見つけることができます。このウィザード コードには次の内容が含まれています。

- **include** セクション。

```
#include "stdafx.h"
```

- stdafx.h。詳細については、「プリコンパイル済みヘッダー ファイル」を参照してください。

```
#include "stdafx.h"
#include "<PROJECTNAME>.h"
#include "MFCAXCtrl.h"
#include "MFCAX1PropPage.h"
```

- デバッグ構成。

```
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

- IMPLEMENT_DYNCREATE。詳細については、「IMPLEMENT_DYNCREATE」を参照してください。
- MFC メッセージ マップ。詳細については、「BEGIN_MESSAGE_MAP」を参照してください。

```
// Message map

BEGIN_MESSAGE_MAP(CMFCAXCtrl, COleControl)
    ON_OLEVERB(AFX_IDS_VERB_PROPERTIES, OnProperties)
END_MESSAGE_MAP()
```

- MFC ディスパッチ マップ。詳細については、「BEGIN_DISPATCH_MAP」を参照してください。

```
// Dispatch map
BEGIN_DISPATCH_MAP(CMFCAXCtrl, COleControl)
#ifndef WIN32_PLATFORM_WFSP
      DISP_FUNCTION_ID(CMFCAXCtrl, "AboutBox", DISPID_ABOUTBOX, AboutBox, VT_EMPTY, VT
S_NONE)
#endif // !WIN32_PLATFORM_WFSP
END_DISPATCH_MAP()
```

- MFC イベント マップ。詳細については、「BEGIN_EVENT_MAP」を参照してください。

```
    // Event map
```

```
BEGIN_EVENT_MAP(CMFCAXCtrl, COleControl)
END_EVENT_MAP()
```

```
STDMETHODIMP CMFCAXCtrl::XObjectSafety::SetInterfaceSafetyOptions(REFIID riid, DWORD dwOpti
onSetMask, DWORD dwEnabledOptions)
{
      // Return S_OK if modified, and S_FALSE otherwise.
      METHOD_PROLOGUE_EX_(CMFCAXCtrl, ObjectSafety)

        // Check if we support the interface and
        // return E_NOINTEFACE if we don't.
      IUnknown* pUnk;
        if (FAILED(this->QueryInterface(riid, (void**)&pUnk)))
            return E_NOINTERFACE;

        // If we are asked to set options we don't support then fail
        if (dwOptionSetMask & ~pThis->m_dwSupportedSafety)
            return E_FAIL;

        // Set the safety options we have been asked to.
        pThis->m_dwCurrentSafety = (pThis->m_dwCurrentSafety  & ~dwOptionSetMask)  |  (dwOp
tionSetMask & dwEnabledOptions);
      return S_OK;
}
#endif //defined(_WIN32_WCE) && (_WIN32_WCE < 0x500)
```

ウィザードで作成される readme ファイルで使用できる情報

アプリケーション ウィザードにより、このようなすぐに使用できるアプリケーションが作成されます。このアプリケーションは、Microsoft Foundation Class の基本的な使用方法を示すだけでなく、独自の ActiveX コントロール アプリケーションを作成するための開始点にもなります。

ここでは、ウィザードで生成されるファイルの一覧を示し、各ファイルの概要を説明します。これらのファイルを開始点として、MFC ActiveX を簡単に開発できます。

MFCAX1 という名前が例として使用されます。実際のプロジェクトの名前に置き換えて読んでください。

MFCAXDLL.vcproj

ウィザードによって作成される MFCAX1 ActiveX コントロール DLL 用のプロジェクトです。1 つのコントロールが含まれています。

MFCAX1.vcproj

アプリケーション ウィザードで生成される VC++ プロジェクトのメイン プロジェクト ファイルです。ファイルを生成した Visual C++ のバージョン情報と、アプリケーション ウィザードで選択したプラットフォーム、構成、およびプロジェクトの機能に関する情報が記述されています。

MFCAX1.h

ActiveX コントロール DLL のメイン インクルード ファイルです。その他のプロジェクト固有のインクルードが含まれています。

MFCAX1.cpp

DLL の初期化、中止などのブックキーピング関連のコードを含むメイン ソース ファイルです。

MFCAX1ppc.rc

Pocket PC プラットフォーム、または同じユーザー インターフェイス モデルをサポートするプラットフォームをコンパイルするときに、プロジェクトで使用するすべての Microsoft Windows リソースの一覧です。.rc ファイルを保存すると、データ セクション内の **#defines** のインスタンスが、定義の表示名ではなく、定義済みの 16 進数値のバージョンで保存されます。

MFCAX1sp.rc

Smartphone プラットフォーム、または同じユーザー インターフェイス モデルをサポートするプラットフォームをコンパイルするときに、プロジェクトで使用するすべての Microsoft Windows リソースの一覧です。.rc ファイルを保存すると、データ セクション内の **#defines** のインスタンスが、定義の表示名ではなく、定義済みの 16 進数値のバージョンで保存されます。

MFCAX1.rc2

Microsoft Visual C++ で編集されないリソースを含んだファイルです。リソース エディタで編集できないリソースはすべてこのファイルに含めます。

MFCAX1.def

ActiveX コントロール DLL を Microsoft Windows で実行するために必要な、ActiveX コントロール DLL に関する情報を含んだファイルです。

MFCAX1.idl

コントロールのタイプ ライブラリに関するオブジェクト記述言語のソース コードを含んだファイルです。

C MFCAX1Ctrl コントロールのファイル：

- **MFCAX1Ctrl.h**: C<PROJECTNAME>Ctrl C++ クラスの宣言を含んだファイルです。
- **MFCAX1Ctrl.cpp**: C **MFCAX1**Ctrl C++ クラスの実装を含んだファイルです。
- **MFCAX1PropPage.h**: C **MFCAX1**PropPage C++ クラスの宣言を含んだファイルです。
- **MFCAX1PropPage.cpp**: **C MFCAX1PropPage** C++ クラスの実装を含んだファイルです。
- **CMFCAX1Ctrl.bmp**: コンテナが **CMFCAX1Ctrl** コントロールをツール パレットに表示するときに、このコントロールを表すために使用するビットマップを含んだファイルです。このビットマップは、メイン リソース (.rc) ファイルに含まれています。

他の標準的なファイル：

- **stdafx.h、stdafx.cpp**: **MFCAX1.pch** というプリコンパイル済みヘッダー (PCH) ファイルと、**stdafx.obj** というプリコンパイル済み型 (PCT) ファイルをビルドするときに使用されるファイルです。
- **resourceppc.h**: 標準ヘッダー ファイルです。新しいリソース ID を定義します。Visual C++ のリソース エディタで、このファイルの読み取りと更新が行われます。
- **resourcesp.h**: 標準ヘッダー ファイルです。新しいリソース ID を定義します。Visual C++ のリソース エディタで、このファイルの読み取りと更新が行われます。コントロール ウィザードは、追加やカスタマイズが必要なソース コードの部分を **TODO** で示します。

参照

その他の技術情報

サンプルとチュートリアル (スマート デバイス プロジェクト)

チュートリアル : スマート デバイス用 ATL マルチプラットフォーム ActiveX コントロールの作成

デバイス用の Visual C++ を使用して、複数デバイスを対象とした ActiveX コントロールを記述できます。次のチュートリアルでは、マルチプラットフォーム対応の ATL ActiveX コントロールを構築する方法について説明しています。

このチュートリアルで行う主なタスク

- マルチプラットフォーム対応のスマート デバイス ATL プロジェクトを作成します。
- ウィザードを使用して、プロジェクトに ActiveX コントロールを追加します。基本的な構造とコードの大部分がウィザードによって生成されます。
- **stdafx.h** ファイルおよび **samplecontrol.h** ファイル内のコードを修正して、スレッド処理モデルを定義し、コンパイラの警告を回避します。
- マルチプラットフォーム ソリューションを配置します。コントロールのテストと実行を容易にするために、Internet Explorer ファイルも生成されることに注意してください。

このチュートリアルは、Visual C++ 開発設定を使用して記述されています。

メモ :
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

マルチプラットフォーム対応の ATL ActiveX コントロールを作成する

マルチプラットフォーム対応の ATL ActiveX コントロールを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックして、[プロジェクトの種類] ペインの Visual C++ ノードを展開します。次に、[スマート デバイス] をクリックします。
2. [テンプレート] ペインの [ATL スマート デバイス プロジェクト] をクリックします。
3. [プロジェクト名] ボックスに「**ATLAXControl**」と入力し、[OK] をクリックします。

   ATL スマート デバイス プロジェクトウィザードが起動します。
4. ATL スマート デバイス プロジェクトウィザードの [ようこそ] ページで、[次へ] をクリックします。

   [プラットフォーム] (ATL スマート デバイス プロジェクト ウィザード) が表示されるので、現在のプロジェクトに追加するプラットフォームの SDK を選択します。
5. [インストール済み SDK] リストから、Smartphone 2003 や Pocket PC 2003 など、現在のプロジェクトに追加するプラットフォームを選択します。プラットフォームを追加するには、左ペインで Smartphone 2003 などのプラットフォームを選択し、右矢印のボタン ([>]) をクリックします。プラットフォームを削除するには、右ペインで Pocket PC 2003 などのプラットフォームを選択し、左矢印のボタン ([<]) をクリックします。
6. [完了] をクリックしてウィザードを終了します。

プロジェクトに ActiveX コントロールを追加する

プロジェクトに ActiveX コントロールを追加するには

1. ソリューション エクスプローラで、[ATLAXControl] を右クリックし、[追加] をポイントします。次に、[クラス] をクリックします。
2. [カテゴリ] ペインで、[スマート デバイス] をクリックします。
3. [テンプレート] ペインの [ATL コントロール] をクリックし、[追加] をクリックします。

   [ATL コントロール ウィザード] ダイアログ ボックスが表示されます。

4. [短い名前] ボックスに、「**samplecontrol**」と入力します。
5. [完了] をクリックしてウィザードを終了します。

ヘッダー ファイルのコードを変更する

stdafx.h のコードを変更するには

1. ソリューション エクスプローラで、stdafx.h をダブルクリックしてエディタで開きます。
2. 次に示すように、`#pragma once` の後に定義 `#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA` を追加します。

```
    // Add this define after
    #pragma once
#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA
```

3. 次の手順に従って、ActiveX コントロールをプロジェクトに追加します。

プロジェクトに ActiveX コントロールを追加するには

1. ソリューション エクスプローラで、samplecontrol.h をダブルクリックしてエディタで開きます。
2. `Isamplecontrol` を定義するコード内で、文字列 `ATL 8.0 : samplecontrol` を `Hello World ActiveX Control` で置き換えます。

メモ :

DCOM プラットフォームの ActiveX コントロールは、ビルド時にアパートメント モデルのスレッドとしてマークする必要があります。これは、ATL コントロール ウィザードの既定の設定です。コンパイル時に警告が表示されても無視できます。また、ATL、GUI、および EXE プロジェクトでは (ATL EXE プロジェクトに atlwin.h、atlctl.h、atlhost.h を追加した場合など)、ATL ヘッダー ファイルを含める前に、stdafx.h で定義されている `_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA` を含める必要があります。この開発手順は、デスクトップの場合と同様です。詳細については、「Visual C++ デバイス プロジェクトのビルドとデバッグ」を参照してください。

マルチプラットフォーム ATL ソリューションの配置

ソリューションを配置するには

1. [ビルド] メニューの [ソリューションのリビルド] をクリックし、コントロールをビルドします。
2. [ビルド] メニューの [ソリューションの配置] をクリックします。
3. Visual Studio ツール バーにある [ターゲット デバイス] ボックスの一覧で、[Pocket PC 2003 SE エミュレータ] または [Pocket PC 2003 デバイス] などの対象を選択します。
4. [ビルド] メニューの [配置] をクリックします。

対象デバイスの選択

ソリューションを配置するときにデバイス選択のダイアログが表示されるようにするには、次の手順を実行します。

配置時にデバイス選択のダイアログを表示するには

1. [ツール] メニューの [オプション] をクリックし、[デバイス ツール] ノードを展開します。次に、[全般] をクリックします。
2. [デバイス ツール] が表示されない場合は、[オプション] ダイアログ ボックスの下部にある [すべての設定を表示] を選択します。
3. [デバイス プロジェクトの配置前に選択できるデバイスを表示] チェック ボックスをオンにし、[OK] をクリックします。

コントロールを実行するには、デバイスでファイル エクスプローラを使用し、Program Files\ATLAXControl に移動して、Internet Explorer ファイルの ATLAXControl をダブルクリックします。セキュリティ メッセージがいくつか表示されます。ページを実行するには [はい] をクリックします。

参照

その他の技術情報

スマート デバイスのチュートリアル
方法 : マルチプラットフォーム対応のデバイス プロジェクトを作成する (Visual C++)

方法：ActiveX コントロールをダイアログ リソースでホストする

Visual Studio 2005 を使用して、デバイス用の ActiveX コントロールを設計する場合、いくつかの追加の手順が発生します。リソース エディタは、デザイン時の編集対象となるコントロールが、それを操作するためのデスクトップ コンピュータに登録されることを前提として設計されています。しかし、デバイス用のコントロールをデスクトップ コンピュータに登録することはできないため、デザイン時には、それに代わる手段が必要です。次の手順は、その方法を示したものです。以下の手順は、ActiveX コントロール プロジェクトとホスト プロジェクトが既に存在すること、および、ActiveX コントロールがダイアログ ボックスでホストされていることを前提としています。

メモ：

使用している設定またはエディションによっては、ヘルプの記載と異なるダイアログ ボックスやメニュー コマンドが表示される場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ダイアログ エディタを使用して ActiveX コントロールを追加するには

1. ダイアログ エディタで、ホスト プロジェクトのダイアログ ボックスを開きます。
2. ツールボックスから、ダイアログ ボックスにカスタム コントロールをドラッグします。
3. ActiveX コントロールをどのように表示するかを考慮しながら、ダイアログ ボックスにカスタム コントロールを配置し、サイズを調整します。
4. カスタム コントロールを右クリックし、[プロパティ] をクリックします。
5. [クラス] プロパティに、ActiveX コントロールの GUID を貼り付けます。中かっこ ({...}) を含めて入力してください。
6. ソリューション エクスプローラで、Project Name.RC2 ファイルを右クリックし、[コードの表示] をクリックします。
7. [手動で編集されたりソースをここに追加します] セクションに、次のコードを追加します。カスタム コントロールを正しく表示するためには、DLGINIT セクションが必要です。実際の DLGINIT セクションの内容は使用されません。<project name> の部分は、該当するプロジェクトの名前に置き換えます。

```
IDD_<project name>_DIALOG DLGINIT BEGIN IDC_CUSTOM1, 0x376, 22, 0 0x0000, 0x0000, 0x08
00, 0x0000, 0x094d, 0x0000, 0x043d, 0x0000, 0x0013, 0xcdcd, 0xcdcd, 0
```

8. ホスト プロジェクトをビルドして実行します。ActiveX コントロールを対象のデバイスに配置して登録します。

別の方法で ActiveX コントロールをホストするには

1. アプリケーション内で **AtlAxWinInit** を呼び出して、AtlAxWin80 ウィンドウ クラスを登録します。ATL アプリケーションでは、この処理がモジュールの初期化コードで実行されます。Win32 アプリケーションの場合、この関数を **WinMain** 関数で呼び出す必要があります。MFC アプリケーションの場合は、次の手順に従ってください。
   a. ソリューション エクスプローラでプロジェクト ノードを右クリックし、[追加] をクリックして [クラス] をクリックします。
   b. [スマート デバイス] の [MFC に ATL サポートを追加] をクリックします。
   c. ホスト アプリケーション クラスの **InitInstance** メソッドの先頭に、**AtlAxWinInit** 呼び出しを追加します。
2. ダイアログ リソース (ATL ダイアログ、複合コントロール、MFC ダイアログなど) で、次の作業を行います。
   a. ツールボックスからカスタム コントロールをドラッグします。
   b. ウィンドウ クラス プロパティを AtlAxWin80 に設定します。
   c. キャプションには、中かっこで囲んだ GUID または progID を設定します。
3. MFC の場合は、リンク入力として別途 atl.lib を追加します。
4. MFC の場合は、[配置] の [追加ファイル] オプションに次の行を追加します。ダイナミック リンク ライブラリの場合は、これらの行があらかじめ存在しますが、MFC のスタティック リンク ライブラリの場合は、別途追加する必要があります。

```
msvcr80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\$(ProjectName)|0
atl80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\$(ProjectName)|0
```

```
msvcr80d.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\$(ProjectName)|0
```

参照

その他の技術情報

Visual C++ を使用したデバイスのプログラミング

スマート デバイスのチュートリアル

スマート デバイスのサンプル

オンライン上のデバイス サンプルを参照するには、「Mobile Developer Center」を参照してください。

次の Visual C# のサンプルは、.NET Compact Framework の機能を示しています。次のネイティブ サンプルは、Visual C++ を使用して実現した一般的なデバイス アプリケーションのシナリオを示しています。

このセクションの内容

カスタム コントロールの作成 (Visual C#)

2 行の ListView カスタム コントロールを示します。デザイン時に WYSIWYG を試すこともできます。

このサンプルは、.NET Compact Framework Version 2.0 に基づいています。

CompactNav (Visual C#)

Smartphone ファイル システムのブラウザを示し、アンマネージ API の呼び出しなどの技法を紹介します。

このサンプルは、.NET Compact Framework Version 1.0 に基づいています。

ネイティブ サンプル (デバイス)

Mobile Developer Center のサイト上にある 3 つのサンプル ネイティブ アプリケーションへのリンクを示します。

参照

その他の技術情報

.NET Compact Framework を使用したデバイスのプログラミング
サンプルとチュートリアル (スマート デバイス プロジェクト)

カスタム コントロールの作成 (Visual C#)

Download sample

この Visual C# のサンプルでは、2 行の ListView カスタム コントロールを作成する方法について説明します。デザイン時に WYSIWYG を試すこともできます。

このサンプルは、.NET Compact Framework のバージョン 1.0 または 2.0 を使用して、Pocket PC と Smartphone のどちら用にもビルドできます。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

このサンプルを実行するには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルの内容

このサンプルは、完全なデザイン時サポートを備えたカスタム描画を使用する TwoLineListBox カスタム コントロールの実装方法を示しています。次のテクニックを含んでいます。

- System.Drawing および System.Drawing.Imaging 名前空間のクラスを使用してカスタム描画コントロールの外観を作成する。
- カスタム コントロールを再描画するためのイベント処理を実現する。
- データ バインディングを使用してデータ ソースからデータを読み込み、コントロールに格納する。
- カスタム コントロールにイメージを表示する。
- クラス ダイアグラムと DesignTimeAttributes を使用して、説明、カテゴリ、その他のプロパティおよびイベントについてのデザイン時サポートを追加する。
- コントロールを既存のデータ ソースにバインドするためのデザイン時サポートを追加する。
- DesignTimeAttributes を通じて、プロパティ (DataSource、Data) に DataSourceListEditor と DataMemberFieldEditor を適用する。
- DesignTimeAttributes を通じて、プロパティ (Context) にカスタム エディタ (Contact Editor) を適用する。
- Line1DisplayMember、Line2DisplayMember など、一部のプロパティに既定値を設定する。
- コレクション プロパティをカスタム エディタ (Contact Editor) からコードにシリアル化する。
- コレクション プロパティをコードからカスタム エディタ (Contact Editor) に逆シリアル化する。
- イメージをカスタム エディタ (Contact Editor) から .resx ファイルにシリアル化する。
- イメージを .resx ファイルからカスタム エディタ (Contact Editor) に逆シリアル化する。

CompactNav (Visual C#)

Download sample

CompactNav サンプルは、Smartphone ファイル システムのブラウザです。.NET Compact Framework 1.0 を使用してマネージ コードで記述されています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

このサンプルを実行するには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

   [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

4. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルの内容

このサンプルでは、基本的なファイル システムのナビゲータを実装する方法を示します。次の技術が含まれます。

- プラットフォーム呼び出し (pInvoke) を使用して、**CreateProcess** などのアンマネージ API を呼び出す
- 例外処理
- ファイル システムからイメージを読み込む
- **CreateProcess** を使用して実行可能ファイルを選択し、実行する
- ディレクトリを選択してアクション キーを押すことでディレクトリを入力する
- メニュー項目 (UpDir) を使用するか、従来の ".." ディレクトリを選択して親ディレクトリにアクセスする
- テスト用の便宜を図るために Quit オプションを実装する (実際の Smartphone アプリケーションの場合は、一般的な事項でも推奨事項でもありません)

ネイティブ サンプル (デバイス)

Visual C++ を使用したデバイス アプリケーションのサンプルについては、Mobile Developer Center を参照してください。これらのネイティブ サンプルには、次のものがあります。

サンプル	説明
PhoneMenus	Smartphone アプリケーションでメニューを示します。
TextEditor	Microsoft Foundation Class の基本的な使用方法を示すと共に、MFC アプリケーションを作成するための開始点にもなります。
PoomViewer	予定、連絡先、およびタスクのデータベースのすべてのフィールドを表示したり、クエリを実行するプログラムで、デバイスに依存しないユーザー インターフェイスを示します。

これらのサンプルおよびサンプルが示す機能の詳細については、各サンプルに含まれる Readme.txt ファイルを参照してください。

参照

その他の技術情報

スマート デバイスのサンプル

リファレンス (デバイス)

このセクションには、スマート デバイス プロジェクトのリファレンス情報が用意されています。関連項目では、デバイス プロジェクト用の言語固有のリファレンス トピックへのリンクを示しています。

このセクションの内容

デバイス プロジェクト用 .NET Compact Framework リファレンス

.NET Compact Framework について説明します。

デバイス用の ATL リファレンス

デバイス プロジェクトでサポートされている ATL リファレンス トピックを表示する方法について説明します。

デバイス用の MFC リファレンス

MFC for Devices のリファレンス トピックを示します。

デバイスの C ランタイム ライブラリ リファレンス

デバイス用の C ランタイム ライブラリのリファレンス トピックを示します。

デバイス用の標準 C++ ライブラリ リファレンス

デバイス アプリケーション開発に使用できる標準 C++ ライブラリのサブセットのリファレンス トピックを示します。

Compilers for Smart Devices

ARM、Renesas、および MIPS の各ファミリのプロセッサについて説明し、デスクトップ コンパイラとデバイス コンパイラの違いを示します。

デバイス用のユーザー インターフェイス リファレンス

スマート デバイス アプリケーション開発だけで使用できるインターフェイス要素の一覧を表示します。

デバイス対応の Visual Basic の言語リファレンス

スマート デバイス プロジェクトで使用できる Visual Basic プログラミング要素の違いについて説明します。

エラー メッセージ (デバイス)

特定のエラーに関する問題の解決方法について説明します。

参照

その他の技術情報

スマート デバイス開発

デバイス プロジェクト用 .NET Compact Framework リファレンス

.NET Compact Framework は、.NET Framework クラス ライブラリのサブセットであり、専用にデザインされたクラスを含みます。共通言語ランタイムやマネージ コード実行に関する、.NET Framework のアーキテクチャを継承しています。

詳細については、「.NET Compact Framework」を参照してください。

参照

その他の技術情報

.NET Framework

デバイス用の ATL リファレンス

デバイスの制限事項により、ATL for Devices では、標準 ATL でサポートされていることがすべてサポートされるわけではありません。このセクションには、その違いを説明するトピックが含まれています。

このセクションの内容

方法 : デバイスでサポートされる ATL クラスおよびメソッドのヘルプを検索する

ヘルプのフィルタ処理機構を使用して、デバイス用の ATL リファレンス、ATL クラス、およびデバイスでサポートされるメソッドに関する最新情報を表示する方法について説明します。

ATL for Devices と標準 ATL の違い

標準 ATL と ATL for Devices の違いについて説明します。

関連するセクション

ATL リファレンス

ATL をよく知らない方のために、このトピックでは、デスクトップ コンピュータ用の Active Template Library について説明します。

参照

その他の技術情報

リファレンス (デバイス)

方法：デバイスでサポートされる ATL クラスおよびメソッドのヘルプを検索する

Visual Studio ヘルプには、目次および索引の [スマート デバイス開発] フィルタが用意されています。このフィルタを適用すると、表示されるトピックが減り、スマート デバイス開発者にとって重要と見なされるドキュメントだけを参照できるようになります。このフィルタの適用は、ATL ライブラリなど、参照トピックのスマート デバイスに関する部分だけを表示するのに最適な方法です。

C++ のトピックと参照ドキュメントを参照するには、[Visual C++ 開発設定] を使用します。

Visual C# などの言語フィルタを適用した場合、表示されるトピックからスマート デバイス開発のトピックが除外されます。このため、[スマート デバイス開発] フィルタを使用するか、フィルタをまったく使用しないかのどちらかにすることをお勧めします。Visual Studio を、[Visual C# 開発設定] などの言語開発設定を使用して起動します。この設定は、[Visual C++ 開発設定] に変更できます。

スマート デバイス開発フィルタを設定する方法

スマート デバイス開発フィルタを設定するには

1. Visual Studio の [ヘルプ] メニューの [目次] または [キーワード] をクリックします。
2. [フィルタ条件] ボックスで、[スマート デバイス開発] を選択します。

開発設定を変更するには

1. Visual Studio の [ツール] メニューの [設定のインポートとエクスポート] をクリックします。
2. ウィザードの [ようこそ] ページで、[すべての設定をリセット] をクリックし、[次へ] をクリックします。

   選択しているオプションがわからない場合、F1 キーを押してウィザードのヘルプ トピックを開きます。
3. [現在の設定の保存] ページで、現在の設定を保存するかどうかを選択し、[次へ] をクリックします。
4. [設定の既定のコレクションの選択] ページで、[Visual C++ 開発設定] を選択し、[完了] をクリックします。

参照

処理手順

方法 : デバイスでサポートされる MFC クラスおよびメソッドのヘルプを検索する

概念

Visual Studio のヘルプ フィルタ

その他の技術情報

スマート デバイス プロジェクトの概要

ATL for Devices と標準 ATL の違い

ATL は、これまで COM ベースのアプリケーションの開発に使用されてきました。ATL 8.0 には、COM プログラミングを簡略化できる機能が用意されています。たとえば、文字列の操作と変換、配列の管理、およびリストとツリーに使用するクラスなどがあります。ATL デバイスの開発者が認識している ATL 8.0 と eMbedded Visual C++ の違いとして、Web サービス クライアントのサポート、拡張されたソケット サポート (IPv6)、セキュリティの強化などがあります。ただし、ATL 8.0 for Devices には、デスクトップ ATL の機能がすべて含まれているわけではありません。ATL for Devices には、Web サービスを使用する機能が備わっていますが、セキュリティ、サービス、ATL データ、および ATL Server の機能は備わっていません。

このセクションには、ATL for Devices と標準 ATL の相異点および類似点を説明するトピックが含まれています。デバイス用の ATL リファレンスに関するヘルプ トピックのフィルタ処理の詳細については、「方法 : デバイスでサポートされる ATL クラスおよびメソッドのヘルプを検索する」を参照してください。

このセクションの内容

サポートされている ATL クラスの一覧

デバイス プロジェクトで完全にサポートされているすべての ATL クラスの一覧を示します。

ATL for Devices 固有のクラス

デバイスだけでサポートされている ATL クラスの一覧を示します。

参照

関連項目

サポートされている ATL クラスの一覧

その他の技術情報

デバイス用の ATL リファレンス

ATL for Devices 固有のクラス

サポートされている ATL クラスの一覧

デバイスでサポートされている ATL クラスを次に示します。デバイスの制限事項により、デスクトップで使用できるクラスの機能には、デバイスで使用できない機能もあります。デバイスでサポートされているクラスおよびメソッドの詳細については、開発環境のヘルプのフィルタ処理機構を使用して参照できます。

ATL のトピックおよびリファレンス (デバイス用の ATL メソッドを含む) を参照するために開発環境のヘルプをフィルタ処理する方法については、「方法 : デバイスでサポートされる ATL クラスおよびメソッドのヘルプを検索する」を参照してください。

デバイスでサポートされている ATL クラスを次に示します。

- allocator Class
- AtlHtmlAttrs クラス
- CAdapt クラス
- CAtlBaseAuthObject クラス
- CAtlBaseModule クラス
- CAtlComModule クラス
- CAtlException クラス
- CAtlFile クラス
- CAtlFileMappingBase クラス
- CAtlHttpClientT クラス
- CAtlNavigateData クラス
- CAtlTemporaryFile クラス
- CAtlWinModule クラス
- CAutoPtr クラス
- CAutoPtrArray クラス
- CAutoPtrElementTraits クラス
- CAutoPtrList クラス
- CAutoVectorPtr クラス
- CAutoVectorPtrElementTraits クラス
- CBasicAuthObject クラス
- CCacheDataBase クラス
- CComAggObject クラス
- CComAllocator クラス
- CComAutoCriticalSection クラス
- CComBSTR クラス
- CComCachedTearOffObject クラス
- CComClassFactory クラス
- CComClassFactory2 クラス
- CComClassFactorySingleton クラス
- CComContainedObject クラス
- CComCriticalSection クラス

- CComCritSecLock クラス
- CComDynamicUnkArray クラス
- CComFakeCriticalSection クラス
- CComHeap クラス
- CComHeapPtr クラス
- CComModule クラス
- CComMultiThreadModel クラス
- CComMultiThreadModelNoCS クラス
- CComObject クラス
- CComObjectGlobal クラス
- CComObjectNoLock クラス
- CComObjectRootEx クラス
- CComObjectStack クラス
- CComPolyObject クラス
- CComPtr クラス
- CComPtrBase クラス
- CComQIPtr クラス
- CComSafeArrayBound クラス
- CComTearOffObject クラス
- CComUnkArray クラス
- CComVariant クラス
- CContainedWindowT クラス
- CCriticalSection クラス
- CCRTAllocator クラス
- CCRTHeap クラス
- CCryptDerivedKey クラス
- CCryptHash クラス
- CCryptHMACHash クラス
- CCryptImportKey クラス
- CCryptKey クラス
- CCryptKeyedHash クラス
- CCryptMACHash クラス
- CCryptMD2Hash クラス
- CCryptMD4Hash クラス
- CCryptMD5Hash クラス
- CCryptProv クラス
- CCryptRandomKey クラス
- CCryptSHAHash クラス

- CCryptSSL3SHAMD5Hash クラス
- CCryptUserExKey クラス
- CCryptUserSigKey クラス
- CDefaultCharTraits クラス
- CDefaultCompareTraits クラス
- CDefaultElementTraits クラス
- CDefaultHashTraits クラス
- CDynamicChain クラス
- CElementTraits クラス
- CElementTraitsBase クラス
- CEvent クラス
- CExpireCuller クラス
- CFileTime クラス
- CFileTimeSpan クラス
- CFirePropNotifyEvent クラス
- CFixedLifetimeCuller クラス
- CGlobalHeap クラス
- CHandle クラス
- CHtmlGen クラス
- CHtmlGenBase クラス
- CHttpResponse クラス
- CImage クラス
- CLifetimeCuller クラス
- CLocalHeap クラス
- CLOUFlusher クラス
- CLRUFlusher クラス
- CMutex クラス
- CNoDllCachePeer クラス
- CNoExpireCuller クラス
- CNoFileCachePeer クラス
- CNoFlusher クラス
- CNonStatelessWorker クラス
- CNoStatClass クラス
- CNoWorkerThread クラス
- CNTLMAuthObject クラス
- COldFlusher クラス
- COleDateTime クラス
- COleDateTimeSpan クラス

- COrCullers クラス
- COrFlushers クラス
- CPoint クラス
- CPrimitiveElementTraits クラス
- CRect クラス
- CRegKey クラス
- CSemaphore クラス
- CSimpleArrayEqualHelper クラス
- CSimpleArrayEqualHelperFalse クラス
- CSimpleDialog クラス
- CSimpleMap クラス
- CSimpleMapEqualHelper クラス
- CSimpleMapEqualHelperFalse クラス
- CSocketAddr クラス
- CStrBufT クラス
- CStreamFormatter クラス
- CStreamOnWriteStream クラス
- CStringRefElementTraits クラス
- CTime クラス
- CTimeSpan クラス
- CUrl クラス
- CWin32Heap クラス
- CWindow クラス
- CWriteStreamHelper クラス
- Win32ThreadTraits クラス

参照

その他の技術情報

ATL for Devices と標準 ATL の違い
デバイス用の ATL リファレンス

ATL for Devices 固有のクラス

ATL for Devices には、一意のクラスが 1 つあります。このセクションでは、このクラスについて説明します。

このセクションの内容

CAtlCEValidateThreadIDDefault メソッド

デバイス プロジェクトに固有の、スレッドに関連するマクロおよびメソッドが用意されているクラスです。

参照

処理手順

方法 : デバイスでサポートされる ATL クラスおよびメソッドのヘルプを検索する

その他の技術情報

デバイス用の ATL リファレンス

CAtlCEValidateThreadIDDefault クラス

CAtlCEValidateThreadIDDefault クラスについて説明します。このクラスには、デバイス プロジェクトに固有の、スレッドに関連するマクロおよびメソッドが用意されています。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : atlcore.h で宣言されます。

参照

概念

CAtlCEValidateThreadIDDefault メソッド

その他の技術情報

ATL for Devices 固有のクラス

CAtlCEValidateThreadIDDefault メソッド

CAtlCEValidateThreadIDDefault メソッドを次の表に示します。

CAtlCEValidateThreadIDDefault::CAtlCEValidateThreadIDDefault	**CAtlCEValidateThreadIDDefault** オブジェクトのインスタンスを作成し、**m_ThreadId** プライベート データ メンバを設定します。
CAtlCEValidateThreadIDDefault::Validate	呼び出し元スレッドのハンドルとして使用されるスレッド識別子が元の呼び出し元スレッドと同じかどうか、または現在のスレッド識別子が異なるかどうかを示します。

CAtlCEValidateThreadIDDefault マクロを次の表に示します。

CE_VALIDATE_THREADID_ASSERT マクロ	デバッグ ビルドでは、CE_VALIDATE_THREADID_ASSERT は、CAtlCEValidateThreadIDDefault::Validate メソッドが false を返すときにデバッグ レポートを生成します。
CE_VALIDATE_THREADID_RETURN マクロ	スレッド識別子が確認された場合に、このマクロのパラメータを返します。
CE_VALIDATE_THREADID_THROW マクロ	スレッド識別子が確認された場合に、例外をスローします。

参照

概念

CAtlCEValidateThreadIDDefault クラス

その他の技術情報

ATL for Devices 固有のクラス

CAtlCEValidateThreadIDDefault::CAtlCEValidateThreadIDDefault

CAtlCEValidateThreadIDDefault オブジェクトのインスタンスを作成し、**m_ThreadId** プライベート データ メンバを現在のスレッド識別子に設定します。このスレッド識別子は呼び出し元スレッドのハンドルとして使用されます。

```
CAtlCEValidateThreadIDDefault();
```

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : atlcore.h で宣言されます。

参照
概念
CAtlCEValidateThreadIDDefault メソッド
その他の技術情報
ATL for Devices 固有のクラス

CAtlCEValidateThreadIDDefault::Validate

呼び出し元スレッドのハンドルとして使用されるスレッド識別子が元の呼び出し元スレッドと同じかどうか、または現在のスレッド識別子が異なるかどうかを示します。

```
bool Validate ();
```

戻り値

正常終了した場合は true を、それ以外の場合は false を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : atlcore.h **で宣言されます。**

参照
概念
CAtlCEValidateThreadIDDefault **メソッド**
その他の技術情報
ATL for Devices **固有のクラス**

CE_VALIDATE_THREADID_ASSERT マクロ

デバッグ ビルドでは、CE_VALIDATE_THREADID_ASSERT は、CAtlCEValidateThreadIDDefault::Validate **メソッドが** false **を返すときにデバッグ レポートを生成します。**

```
CE_VALIDATE_THREADID_ASSERT();
```

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : atlcore.h **で宣言されます。**

参照
概念
CAtlCEValidateThreadIDDefault **メソッド**
その他の技術情報
ATL for Devices **固有のクラス**

CE_VALIDATE_THREADID_RETURN マクロ

スレッド識別子が確認された場合に、このマクロのパラメータを返します。

```
CE_VALIDATE_THREADID_RETURN(r);
```

パラメータ

CAtlCEValidateThreadIDDefault クラスの **CE_VALIDATE_THREADID_RETURN** マクロのパラメータの説明を次の表に示します。

パラメータ	説明
r	スレッドが確認された場合に返すパラメータ。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : atlcore.h で宣言されます。

参照
関連項目
CAtlCEValidateThreadIDDefault::Validate
概念
CAtlCEValidateThreadIDDefault メソッド
その他の技術情報
ATL for Devices 固有のクラス

CE_VALIDATE_THREADID_THROW マクロ

スレッド識別子が確認された場合に、例外をスローします。

```
CE_VALIDATE_THREADID_THROW(e);
```

パラメータ

CAtlCEValidateThreadIDDefault クラスの CE_VALIDATE_THREADID_THROW マクロのパラメータの説明を次の表に示します。

パラメータ	説明
e	スローする例外。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : atlcore.h で宣言されます。

参照

関連項目

CAtlCEValidateThreadIDDefault::Validate

概念

CAtlCEValidateThreadIDDefault メソッド

その他の技術情報

ATL for Devices 固有のクラス

デバイス用の MFC リファレンス

Visual Studio 2005 のスマート デバイス プロジェクトでは、MFC for Devices のバージョン 8.0 が使用されています。MFC 8.0 for Devices は、標準デスクトップ MFC と eMbedded Visual C++ MFC 3.0 のどちらとも異なります。不明な点については、以下で紹介している各トピックを参照してください。

MFC for Devices のファイルは、%Program Files%\Microsoft Visual Studio 8\VC\ce フォルダに格納されています。

このセクションの内容

MFC クラス

MFC for Devices の大半のドキュメントが記載されています。

MFC for Devices と標準デスクトップ MFC とでは、同じドキュメントが共有されます。

標準デスクトップ MFC との動作の違いについては、「スマート デバイス開発者のためのメモ」というセクションのトピックに記載されています。

デバイス クラスの一意な MFC

スマート デバイスに固有の MFC クラスについて記載されています。固有のクラスはごく少数です。

方法 : デバイスでサポートされる MFC クラスおよびメソッドのヘルプを検索する

デバイス向け MFC/ATL リファレンスについて最新情報を表示するためのフィルタ処理機能の使用方法について説明します。

デバイスでサポートされているデスクトップ MFC クラスの一覧

デバイスでサポートされている標準デスクトップ MFC のクラスが一覧形式で記載されています。

デバイスでサポートされていないデスクトップ MFC クラスの一覧

デバイスでサポートされない標準デスクトップ MFC のクラスが一覧形式で記載されています。

MFC 8.0 でサポートされなくなった MFC 3.0 の eVC クラスの一覧

eVC MFC 3.0 と Visual Studio 2005 MFC 8.0 for Devices の違いについて説明します。

MFC C++ for Devices と標準 MFC の違い

デスクトップだけでなく、MFC C++ for Devices でもサポートされているクラスを図で紹介しています。

参照

概念

eMbedded Visual C++ から Visual Studio 2005 へのアップグレード ウィザード

デバイス用の標準 C++ ライブラリ リファレンス

その他の技術情報

リファレンス (デバイス)

Windows CE の MFC ウィザードの C++

Visual C++ のウィザードを使用すると、複数の Windows CE デバイスをターゲットとする MFC アプリケーションを記述できます。Visual C++ のウィザードには、MFC スマート デバイス アプリケーション、MFC スマート デバイス ActiveX コントロール、および MFC スマート デバイス DLL などがあります。

Windows CE の MFC ウィザードの C++

Windows CE の MFC ウィザードの C++ には次のウィザードが含まれます。

- MFC スマート デバイス アプリケーション ウィザード

  MFC スマート デバイス アプリケーション ウィザードでは、組み込み機能と、Windows CE 実行可能アプリケーション (.exe) の基本機能を持つアプリケーションが生成されます。詳細については、次のトピックを参照してください。

- [高度な機能] (MFC スマート デバイス アプリケーション ウィザード)
- [アプリケーションの種類] (MFC スマート デバイス アプリケーション ウィザード)
- [プラットフォーム] (MFC スマート デバイス アプリケーション ウィザード)
- [ユーザー インターフェイスの機能] (MFC スマート デバイス アプリケーション ウィザード)
- [ドキュメント テンプレート文字列] (MFC スマート デバイス アプリケーション ウィザード)
- [生成されたクラス] (MFC スマート デバイス アプリケーション ウィザード)
- MFC スマート デバイス ActiveX コントロール ウィザード

  MFC ウィザードにより生成されたプロジェクトには、C++ ソース (.cpp) ファイル、リソース (.rc) ファイル、およびプロジェクト (.vcproj) ファイルが含まれます。詳細については、次のトピックを参照してください。

- [コントロールの設定] (MFC スマート デバイス ActiveX コントロール ウィザード)
- [コントロール名] (MFC スマート デバイス ActiveX コントロール ウィザード)
- [プラットフォーム] (MFC スマート デバイス ActiveX コントロール ウィザード)
- [アプリケーションの設定] (MFC スマート デバイス ActiveX コントロール ウィザード)
- MFC スマート デバイス DLL ウィザード

  MFC スマート デバイス コントロール DLL プロジェクトを使用して、スマート デバイス DLL を作成できます。詳細については、次のトピックを参照してください。

- [アプリケーションの設定] (MFC スマート デバイス DLL ウィザード)
- [プラットフォーム] (MFC スマート デバイス DLL ウィザード).

参照

処理手順

チュートリアル : スマート デバイス用マルチプラットフォーム MFC アプリケーションの作成

方法：デバイスでサポートされる MFC クラスおよびメソッドのヘルプを検索する

Visual Studio ヘルプには、目次および索引の [スマート デバイス開発] フィルタが用意されています。このフィルタを適用すると、表示されるトピックが減り、スマート デバイス開発者にとって重要と見なされるドキュメントだけを参照できるようになります。このフィルタの適用は、MFC ライブラリなど、参照トピックのスマート デバイスに関する部分だけを表示するのに最適な方法です。

C++ のトピックと参照ドキュメントを参照するには、[Visual C++ 開発設定] を使用します。

Visual C# などの言語フィルタを適用した場合、表示されるトピックからスマート デバイス開発のトピックが除外されます。このため、[スマート デバイス開発] フィルタを使用するか、フィルタをまったく使用しないかのどちらかにすることをお勧めします。

Visual Studio を、[Visual C# 開発設定] などの言語開発設定を使用して起動します。この既定の設定は、[Visual C++ 開発設定] に変更できます。

スマート デバイス開発フィルタを設定する方法

スマート デバイス開発フィルタを設定するには

1. Visual Studio の [ヘルプ] メニューの [目次] または [キーワード] をクリックします。
2. [フィルタ条件] ボックスで、[スマート デバイス開発] を選択します。

開発設定を変更するには

1. Visual Studio の [ツール] メニューで、[設定のインポートとエクスポート] をクリックします。
2. ウィザードの [ようこそ] ページで、[すべての設定をリセット] をクリックし、[次へ] をクリックします。

   選択しているオプションがわからない場合、F1 キーを押してウィザードのヘルプ トピックを開きます。
3. [現在の設定の保存] ページで、現在の設定を保存するかどうかを選択し、[次へ] をクリックします。
4. [設定の既定のコレクションの選択] ページで、[Visual C++ 開発設定] を選択し、[完了] をクリックします。

参照

概念

デバイスでサポートされていないデスクトップ MFC クラスの一覧

Visual Studio のヘルプ フィルタ

その他の技術情報

スマート デバイス プロジェクトの概要

MFC 8.0 でサポートされなくなった MFC 3.0 の eVC クラスの一覧

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

eMbedded Visual C++

- CDaoFieldExchange クラス
- CDBVariant クラス
- CFieldExchange クラス
- CFontDialog クラス
- CLongBinary クラス
- COleCmdUI クラス
- COleCurrency クラス
- COleDataObject クラス
- COlePropertyPage クラス
- CPrintDialog クラス
- CPrintInfo のメンバ
- CSocketFile クラス

Visual Studio 2005 SP1 では、eMbedded Visual C++ プロジェクトの移行を支援する 15 個の eVC MFC クラスがデバイス MFC ライブラリに追加されています。次のクラスは、Visual Studio 2005 SP1 for Devices ではサポートされますが、Visual Studio 2005 MFC for Devices ではサポートされません。

- CBitmap クラス
- CDialogBar クラス
- CEditView クラス
- CFindReplaceDialog クラス
- CHttpConnection クラス
- CHttpFile クラス
- CInternetConnection クラス
- CInternetException クラス
- CInternetFile クラス
- CInternetSession クラス
- COleSafeArray クラス
- CReBar クラス
- CReBarCtrl クラス
- CRecentFileList クラス
- CSplitterWnd クラス

次のクラスは、テンプレート クラスを使用して同等の機能を提供する typedef です。

- CByteArray クラス
- CDWordArray クラス
- CMapPtrToPtr クラス

- CMapPtrToWord クラス
- CMapStringToOb クラス
- CMapStringToPtr クラス
- CMapStringToString クラス
- CMapWordToOb クラス
- CMapWordToPtr クラス
- CObArray クラス
- CObList クラス
- COleSafeArray クラス
- CPtrArray クラス
- CPtrList クラス
- CStringArray クラス
- CStringList クラス
- CUIntArray クラス
- CWordArray クラス

MFC 3.0 と MFC 8.0 の API の動作の違い

- CDocument::SaveModified ダイアログ クラスおよび関連するリソースは、どのプラットフォーム向けの MFC 8.0 からも削除されています。したがって、Pocket PC 2003 プラットフォームおよび Smartphone 2003 プラットフォームでは、**DoSave** メソッドおよび **SaveModified** メソッドは、使用時に既定のファイル名がありません。また、既定では、ファイル名 (自動生成されたファイル名など) の入力を求められません。ただし、Pocket PC 2003 プラットフォームでは、この動作をオーバーライドし、ファイル名の入力を求めるオプションが用意されています。Smartphone プラットフォームでは、ファイル名の入力を求める場合、**CDocManager::DoPromptFileName** を呼び出すことができます。**DoSave** メソッドおよび **SaveModified** メソッドの既定のファイル名に関する動作は Windows CE プラットフォームでサポートされているため、その機能はデスクトップの場合と同じです。
- MFC 8.0 for Devices では、ドッキングをサポートしていません。たとえば、**CCommandBar::m_pDockBar** メンバおよび **CCommandBar::m_pDockContext** メンバはサポートされていません。詳細については、「CCommandBar クラス」を参照してください。ドッキングのサポートの詳細については、「ドッキング ツール バーとフローティング ツール バー」を参照してください。
- MFC 8.0 for Devices では、CDC::FrameRect が CDC クラスのメンバではなくなりました。
- MFC 8.0 for Devices では、CCeDocList の名前が CDocList クラスに変更されました。
- MFC 8.0 for Devices では、CCeSocket の機能が CAsyncSocket クラスにカプセル化されています。
- MFC 8.0 for Devices では、CFont::CreateFont がサポートされていません。代わりに、CFont::CreatePointFont を使用できます。
- MFC 8.0 for Devices では、**CCommandBar::m_pDockBar** メンバおよび **CCommandBar::m_pDockContext** メンバがサポートされなくなりました。
- MFC 8.0 for Devices では、**LPINLINEIMAGEINFO** 構造体が INLINEIMAGEINFO に置き換わっています。
- Visual Studio 2005 のウィザードで生成されたリソースは、Windows Mobile 5.0 ユーザー インターフェイス (UI: User Interface) のガイドラインに準拠しています。つまり、すべてのアプリケーションの **MenuBar** クラスでは、必ず、左側に [新規作成] ボタン、右側に [メニュー] が表示されます。したがって、MFC 8.0 for Devices では、**m_bShowSharedNewButton** 変数をサポートしません。たとえば、アプリケーション コードで `wndCommandBar.m_bShowSharedNewButton = TRUE;` を使用している場合は、このコード行をコメント アウトし、アプリケーションを MFC 8.0 for Devices に移植できます。
- アプリケーション コードで `ON_NOTIFY(DLN_CE_CREATE、AFXCE_ID_DOCLIST` または `OnCreateDocList` を使用している場合は、次のコンパイル エラーが発生します。
- "MainFrm.cpp(42) : エラー C2065: 'DLN_CE_CREATE' : 定義されていない識別子です。"
- "MainFrm.cpp(42) : エラー C2065: 'AFXCE_ID_DOCLIST' : 定義されていない識別子です。"
- MFC 8.0 では、DLN_DOCLIST_CREATE、DLN_DOCLIST_DESTROY、および AFX_ID_DOCLIST を安全に使用できます。

- MFC 8.0 を使用する場合、標準の CRT ライブラリにリンクすることはできません。
- MFC 8.0 に移植する場合は、`# define _WIN32_WCE_PSPC` を追加してください。既定では、このフラグは MFC8.0 で定義されていません。
- 詳細については、「デバイスでサポートされていないデスクトップ MFC クラスの一覧」を参照してください。

参照

概念

MFC C++ for Devices と標準 MFC の違い

その他の技術情報

デバイス クラスの一意な MFC

MFC C++ for Devices と標準 MFC の違い

メモリなどのデバイスの制約により、MFC for Devices では、標準デスクトップ MFC でサポートされているすべてのクラスおよび機能をサポートしているわけではありません。次の表では、デバイス対応の Visual Studio の最新リリース (MFC Version 8.0) でサポートされているクラスおよびサポートされていないクラスについて説明します。

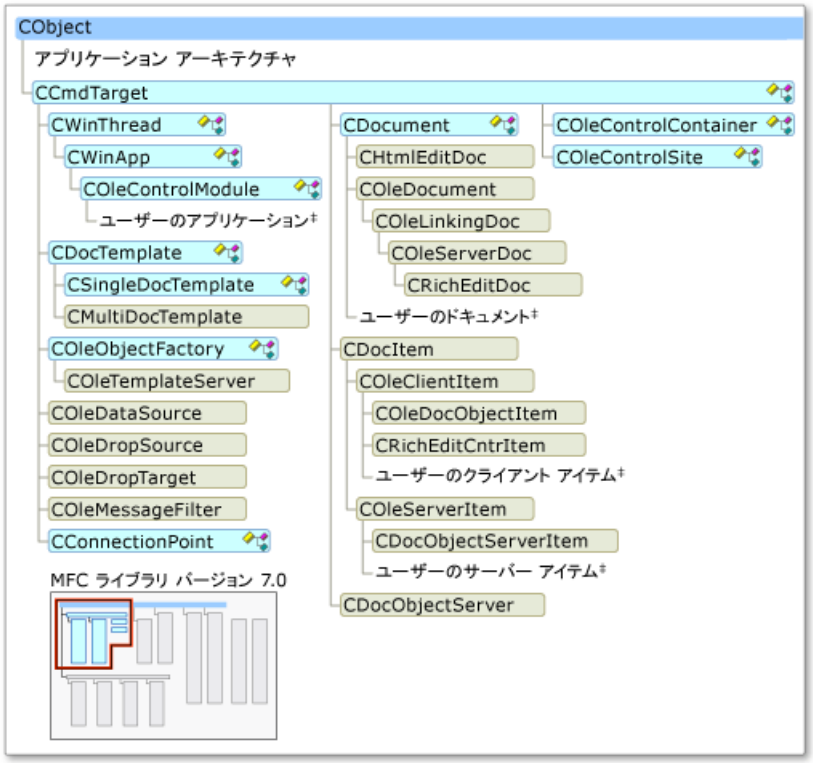
また、ヘルプ機能のフィルタ処理機構を使用して、デスクトップおよびデバイスの両方でサポートされている MFC クラスの一覧を表示することもできます。

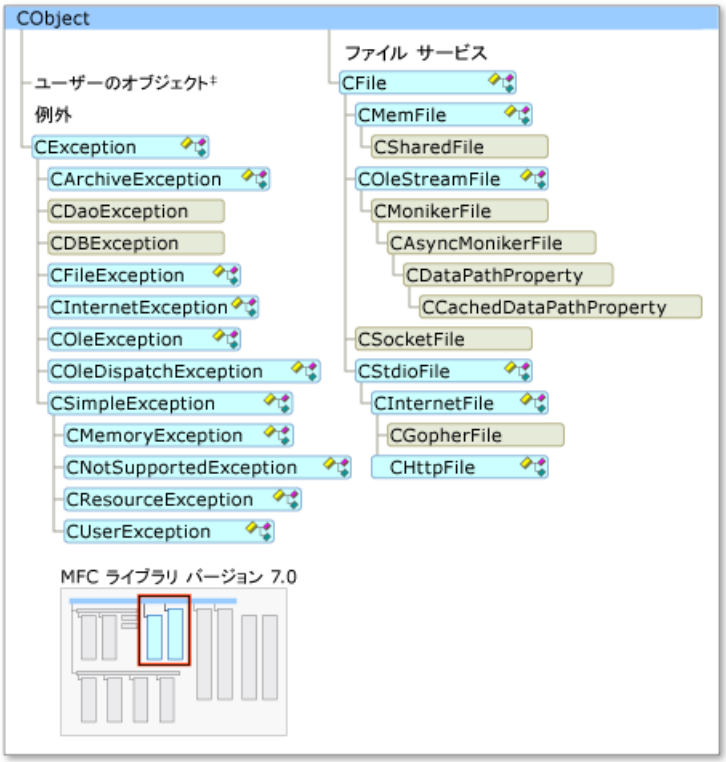
- サポートされるクラスやメソッドなど、デバイス用の MFC リファレンスに関するヘルプ トピックのフィルタ処理の詳細については、「方法 : デバイスでサポートされる MFC クラスおよびメソッドのヘルプを検索する」を参照してください。
- デバイスでサポートされている MFC クラスの詳細については、「デバイスでサポートされているデスクトップ MFC クラスの一覧」を参照してください。
- デバイスでサポートされていない MFC クラスの詳細については、「デバイスでサポートされていないデスクトップ MFC クラスの一覧」を参照してください。
- デスクトップだけでなく、MFC C++ for Devices でもサポートされているクラスを次に示します。

MFC 8.0 for Devices と標準デスクトップ MFC 7.0 の相異点

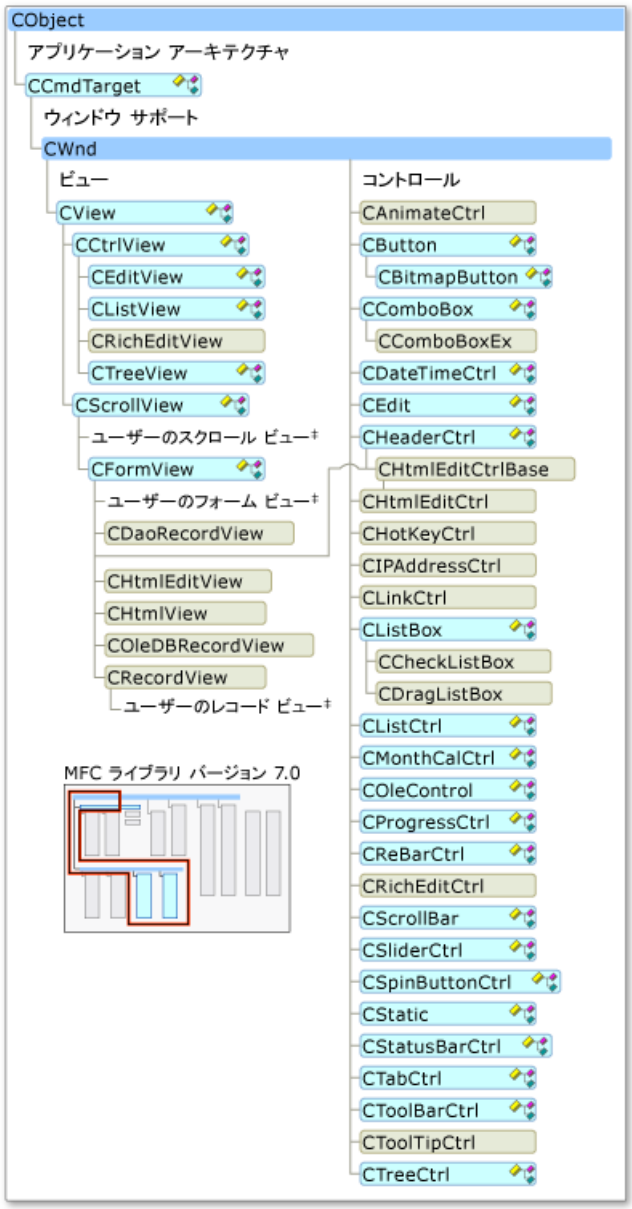
Legend

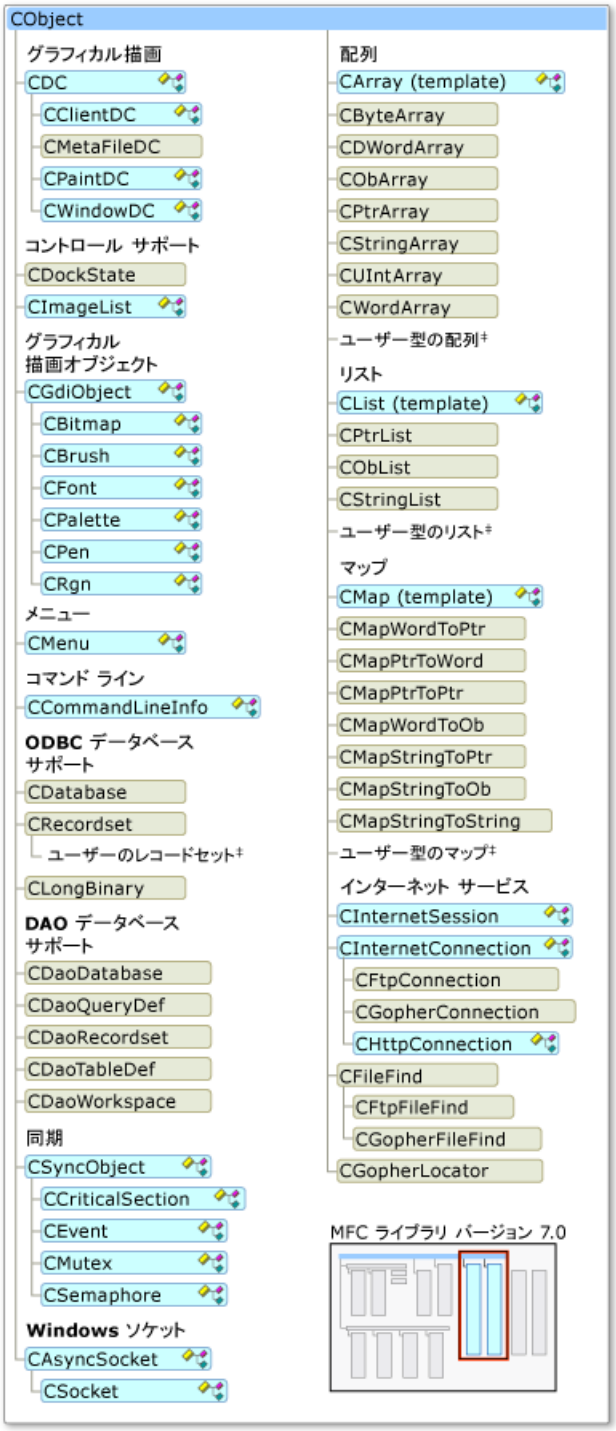
記号	説明
	MFC for Devices 8.0 でサポートされる標準デスクトップ MFC 7.0 クラス
	MFC for Devices 8.0 でサポートされない標準デスクトップ MFC 7.0 クラス
‡	クラスの分類

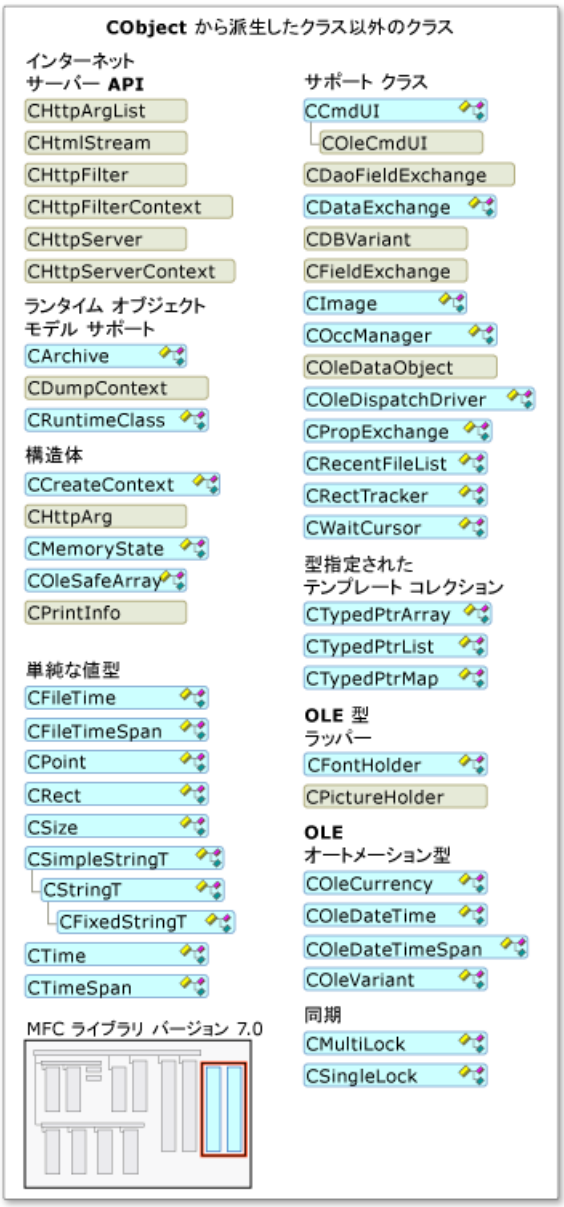



CObject
ユーザーのオブジェクト‡
例外
CException
CArchiveException
CDaoException
CDBException
CFileException
CInternetException
COleException
COleDispatchException
CSimpleException
CMemoryException
CNotSupportedException
CResourceException
CUserException
ファイル サービス
CFile
CMemFile
CSharedFile
COleStreamFile
CMonikerFile
CAsyncMonikerFile
CDataPathProperty
CCachedDataPathProperty
CSocketFile
CStdioFile
CInternetFile
CGopherFile
CHttpFile
MFC ライブラリ バージョン 7.0


CObject
アプリケーション アーキテクチャ
CCmdTarget
ウィンドウ サポート
CWnd
フレーム ウィンドウ
CFrameWnd
CMDIChildWnd
ユーザーの MDI ウィンドウ‡
CMDIFrameWnd
ユーザーの MDI ワークスペース‡
CMiniFrameWnd
ユーザーの SDI ウィンドウ‡
COleIPFrameWnd
CSplitterWnd
コントロール バー
CControlBar
CDialogBar
COleResizeBar
CReBar
CStatusBar
CToolBar
プロパティ シート
CPropertySheet
MFC ライブラリ バージョン 7.0
ダイアログ ボックス
CDialog
CCommonDialog
CColorDialog
CFileDialog
CFindReplaceDialog
CFontDialog
COleDialog
COleBusyDialog
COleChangeIconDialog
COleChangeSourceDialog
COleConvertDialog
COleInsertDialog
COleLinksDialog
COleUpdateDialog
COlePasteSpecialDialog
COlePropertiesDialog
CPageSetupDialog
CPrintDialog
CPrintDialogEx
COlePropertyPage
CPropertyPage
ユーザーのダイアログ ボックス‡
CDHtmlDialog
CMultiPageDHtmlDialog


CObject
アプリケーション アーキテクチャ
CCmdTarget
ウィンドウ サポート
CWnd
ビュー
CView
CCtrlView
CEditView
CListView
CRichEditView
CTreeView
CScrollView
ユーザーのスクロール ビュー‡
CFormView
ユーザーのフォーム ビュー‡
CDaoRecordView
CHtmlEditView
CHtmlView
COleDBRecordView
CRecordView
ユーザーのレコード ビュー‡
MFC ライブラリ バージョン 7.0
コントロール
CAnimateCtrl
CButton
CBitmapButton
CComboBox
CComboBoxEx
CDateTimeCtrl
CEdit
CHeaderCtrl
CHtmlEditCtrlBase
CHtmlEditCtrl
CHotKeyCtrl
CIPAddressCtrl
CLinkCtrl
CListBox
CCheckListBox
CDragListBox
CListCtrl
CMonthCalCtrl
COleControl
CProgressCtrl
CReBarCtrl
CRichEditCtrl
CScrollBar
CSliderCtrl
CSpinButtonCtrl
CStatic
CStatusBarCtrl
CTabCtrl
CToolBarCtrl
CToolTipCtrl
CTreeCtrl





参照

その他の技術情報

デバイス用の MFC リファレンス

デバイス クラスの一意な MFC

デバイスでサポートされているデスクトップ MFC クラスの一覧

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

デバイス対応の Visual Studio の最新リリースでサポートされている MFC クラスを次に示します。デバイスの制限事項により、デスクトップで使用できるクラスの機能には、デバイスで使用できない機能もあります。

次の一覧は、Visual Studio 2005 SP1 に合わせて更新されています。

デバイスでサポートされている MFC クラスを次に示します。

- CArchive クラス
- CArchiveException クラス
- CArray クラス
- CAsyncSocket クラス
- CBitmap クラス
- CBitmap クラス (Visual Studio 2005 SP1 でサポート)
- CBrush クラス
- CButton クラス
- CClientDC クラス
- CCmdTarget クラス
- CCmdUI クラス
- CColorDialog クラス
- CComboBox クラス
- CCommandBar クラス
- CCommandLineInfo クラス
- CCommonDialog クラス
- CConnectionPoint クラス
- CControlBar クラス
- CCriticalSection クラス
- CCtrlView クラス
- CDataExchange クラス
- CDateTimeCtrl クラス
- CDC クラス
- CDialog クラス
- CDialogBar クラス (Visual Studio 2005 SP1 でサポート)
- CDocList クラス
- CDocListDocTemplate クラス
- CDocTemplate クラス
- CDocument クラス
- CEdit クラス
- CEditView クラス (Visual Studio 2005 SP1 でサポート)

- CEvent クラス
- CException クラス
- CFile クラス
- CFileDialog クラス
- CFileException クラス
- CFindReplaceDialog クラス (Visual Studio 2005 SP1 でサポート)
- CFixedStringT クラス
- CFont クラス
- CFontHolder クラス
- CFormView クラス
- CFrameWnd クラス
- CGdiObject クラス
- CHeaderCtrl クラス
- CHttpConnection クラス (Visual Studio 2005 SP1 でサポート)
- CHttpFile クラス (Visual Studio 2005 SP1 でサポート)
- CImageList クラス
- CInternetConnection クラス (Visual Studio 2005 SP1 でサポート)
- CInternetException クラス (Visual Studio 2005 SP1 でサポート)
- CInternetFile クラス (Visual Studio 2005 SP1 でサポート)
- CInternetSession クラス (Visual Studio 2005 SP1 でサポート)
- CInvalidArgException クラス
- CList クラス
- CListBox クラス
- CListCtrl クラス
- CListView クラス
- CMap クラス
- CMemFile クラス
- CMemoryException クラス
- CMenu クラス
- CMonthCalCtrl クラス
- CMultiLock クラス
- CMutex クラス
- CNotSupportedException クラス
- CObject クラス
- COccManager クラス
- COleControl クラス
- COleControlContainer クラス
- COleControlModule クラス

- COleControlSite クラス
- COleDateTime クラス
- COleDispatchDriver クラス
- COleDispatchException クラス
- COleException クラス
- COleObjectFactory クラス
- COlePropertyPage クラス
- COleSafeArray クラス (Visual Studio 2005 SP1 でサポート)
- COleStreamFile クラス
- COleVariant クラス
- CPaintDC クラス
- CPalette クラス
- CPen クラス
- CProgressCtrl クラス
- CPropertyPage クラス
- CPropertySheet クラス
- CPropExchange クラス
- CReBar クラス (Visual Studio 2005 SP1 でサポート)
- CReBarCtrl クラス (Visual Studio 2005 SP1 でサポート)
- CRecentFileList クラス (Visual Studio 2005 SP1 でサポート)
- CRectTracker クラス
- CResourceException クラス
- CRgn クラス
- CScrollBar クラス
- CScrollView クラス
- CSemaphore クラス
- CSimpleException クラス
- CSimpleStringT クラス
- CSingleDocTemplate クラス
- CSingleLock クラス
- CSliderCtrl クラス
- CSocket クラス
- CSocketFile クラス
- CSpinButtonCtrl クラス
- CSplitterWnd クラス (Visual Studio 2005 SP1 でサポート)
- CStatic クラス
- CStatusBar クラス
- CStatusBarCtrl クラス

- CStdioFile クラス
- CStringT クラス
- CSyncObject クラス
- CTabCtrl クラス
- CTime クラス
- CTimeSpan クラス
- CToolBar クラス
- CToolBarCtrl クラス
- CTreeCtrl クラス
- CTreeView クラス
- CTypedPtrList クラス
- CUserException クラス
- CView クラス
- CWinApp クラス
- CWindowDC クラス
- CWinThread クラス
- CWnd クラス

上記のクラス間では、いくつかの動作が異なります。詳細については、「MFC 8.0 でサポートされなくなった MFC 3.0 の eVC クラスの一覧」を参照してください。

サポートされている MFC トピックと MFC リファレンスの詳細については、開発環境のヘルプのフィルタ処理機構を使用して参照できます。このフィルタ処理機構の詳細については、「方法 : デバイスでサポートされる MFC クラスおよびメソッドのヘルプを検索する」を参照してください。

参照

概念

MFC 8.0 でサポートされなくなった MFC 3.0 の eVC クラスの一覧

その他の技術情報

デバイス用の MFC リファレンス

デバイス クラスの一意な MFC

デバイスでサポートされていないデスクトップ MFC クラスの一覧

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

デバイスの制限事項により、デスクトップで使用できる MFC クラスの機能には、デバイスで使用できない機能もあります。デバイスでサポートされている MFC クラスの一覧については、「デバイスでサポートされているデスクトップ MFC クラスの一覧」を参照してください。MFC for Devices でサポートされない標準デスクトップ MFC クラスは次のとおりです。

- CAnimateCtrl クラス
- CAsyncMonikerFile クラス
- CByteArray クラス
- CCachedDataPathProperty クラス
- CCheckListBox クラス
- CComboBoxEx クラス
- CDaoDatabase クラス
- CDaoException クラス
- CDaoFieldExchange クラス
- CDaoQueryDef クラス
- CDaoRecordset クラス
- CDaoRecordView クラス
- CDaoTableDef クラス
- CDaoWorkspace クラス
- CDatabase クラス
- CDataPathProperty クラス
- CDBException クラス
- CDBVariant クラス
- CDHtmlDialog クラス
- CDocItem クラス
- CDockState クラス
- CDocObjectServer クラス
- CDocObjectServerItem クラス
- CDragListBox クラス
- CDumpContext クラス
- CDWordArray クラス
- CFieldExchange クラス
- CFileFind クラス
- CFileTime クラス
- CFileTimeSpan クラス
- CFontDialog クラス
- CFtpConnection クラス
- CFtpFileFind クラス

- CGopherConnection クラス
- CGopherFile クラス
- CGopherFileFind クラス
- CGopherLocator クラス
- CHotKeyCtrl クラス
- CHtmlEditCtrl クラス
- CHtmlEditCtrlBase クラス
- CHtmlEditDoc クラス
- CHtmlEditView クラス
- CHtmlStream クラス
- CHtmlView クラス
- CHttpArg 構造体
- CHttpArgList クラス
- CHttpFilter クラス
- CHttpFilterContext クラス
- CHttpServer クラス
- CHttpServerContext クラス
- CImage クラス
- CIPAddressCtrl クラス
- CLinkCtrl クラス
- CLongBinary クラス
- CMapPtrToPtr クラス
- CMapPtrToWord クラス
- CMapStringToOb クラス
- CMapStringToPtr クラス
- CMapStringToString クラス
- CMapWordToOb クラス
- CMapWordToPtr クラス
- CMDIChildWnd クラス
- CMDIFrameWnd クラス
- CMetaFileDC クラス
- CMiniFrameWnd クラス
- CMonikerFile クラス
- CMultiDocTemplate クラス
- CMultiPageDHtmlDialog クラス
- CObArray クラス
- CObList クラス
- COleBusyDialog クラス

- COleChangeIconDialog クラス
- COleChangeSourceDialog クラス
- COleClientItem クラス
- COleCmdUI クラス
- COleConvertDialog クラス
- COleCurrency クラス
- COleDataObject クラス
- COleDataSource クラス
- COleDateTimeSpan クラス
- COleDBRecordView クラス
- COleDialog クラス
- COleDocObjectItem クラス
- COleDocument クラス
- COleDropSource クラス
- COleDropTarget クラス
- COleInsertDialog クラス
- COleIPFrameWnd クラス
- COleLinkingDoc クラス
- COleLinksDialog クラス
- COleMessageFilter クラス
- COlePasteSpecialDialog クラス
- COlePropertiesDialog クラス
- COleResizeBar クラス
- COleServerDoc クラス
- COleServerItem クラス
- COleTemplateServer クラス
- COleUpdateDialog クラス
- CPageSetupDialog クラス
- CPictureHolder クラス
- CPrintDialog クラス
- CPrintDialogEx クラス
- CPrintInfo のメンバ
- CPoint クラス
- CPtrArray クラス
- CPtrList クラス
- CRecordset クラス
- CRecordView クラス
- CRect クラス

- CRichEditCntrItem クラス
- CRichEditCtrl クラス
- CRichEditDoc クラス
- CRichEditView クラス
- CSharedFile クラス
- CSize クラス
- CStrBufT クラス
- CStringArray クラス
- CStringData クラス
- CStringList クラス
- CToolTipCtrl クラス
- CTypedPtrArray クラス
- CTypedPtrMap クラス
- CUIntArray クラス
- CWaitCursor クラス
- CWinFormsControl クラス
- CWinFormsDialog クラス
- CWinFormsView クラス
- CWordArray クラス

Visual Studio 2005 SP1 では、デバイス MFC ライブラリに、15 個のデスクトップ MFC クラスが追加されています。次のクラスは、Visual Studio 2005 SP1 for Devices ではサポートされますが、Visual Studio 2005 MFC for Devices ではサポートされません。

- CBitmap クラス
- CDialogBar クラス
- CEditView クラス
- CFindReplaceDialog クラス
- CHttpConnection クラス
- CHttpFile クラス
- CInternetConnection クラス
- CInternetException クラス
- CInternetFile クラス
- CInternetSession クラス
- COleSafeArray クラス
- CReBar クラス
- CReBarCtrl クラス
- CRecentFileList クラス
- CSplitterWnd クラス

参照

関連項目

デバイスでサポートされているデスクトップ MFC クラスの一覧

概念

MFC 8.0 でサポートされなくなった MFC 3.0 の eVC クラスの一覧
その他の技術情報
デバイス用の MFC リファレンス
デバイス クラスの一意な MFC
MFC クラス

デバイス クラスの一意な MFC

デバイスの MFC には、いくつかの一意のクラスがあります。このセクションでは、それらのクラスについて説明します。

このセクションの内容

CCommandBar クラス

メニュー バーとツール バーの機能を組み合わせます。表示画面が小さいデバイス向けにデザインされています。

CDocList クラス

DocList ウィンドウ タイプのオペレーティング システム実装をラップします。MFC Doc/View アーキテクチャと統合する追加機能が含まれています。

CDocListDocTemplate クラス

SDI アプリケーション タイプの改良版で、CSingleDocTemplate により実現されます。

AfxEnableDRA

デバイス解像度認識を有効にする関数です。

関連するセクション

デバイス用の MFC リファレンス

方法 : デバイスでサポートされる MFC クラスおよびメソッドのヘルプを検索する

CCommandBar クラス

このクラスを使用すると、コマンド バーの作成と変更を行うことができます。コマンド バーは、[閉じる] ボタン、[ヘルプ] ボタン、および [OK] ボタンと同様に、メニュー バーに含めることのできるツール バーです。コマンド バーには、メニュー、コンボ ボックス、ボタン、および区切り記号を含めることができます。区切り記号は、他の要素をグループに分けたり、コマンド バーの領域を確保したりするために使用される空白のことです。**CCommandBar** オブジェクトを作成した後、InsertMenuBar、InsertComboBox、および InsertSeparator メソッドを使用して、それぞれメニュー バー、コンボ ボックス、および区切り記号を挿入します。他のすべての要素をコマンド バーに追加し終わったら、AddAdornments メソッドを使用して [キャンセル] ボタンを追加し、オプションで [ヘルプ] ボタンと [OK] ボタンを追加します。コマンド バー上のメニューを変更するたびにコマンド バーを再描画するには、DrawMenuBar メソッドを使用します。

Windows CE 5.0 では、**CDialog::m_pWndEmptyCB** メンバがサポートされなくなり、ユーザーが作成および挿入処理を制御するようになりました。以前は、このメンバ変数は Pocket PC 上の空の **CommandBar** または **MenuBar** をポイントするために使用されていました。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CCommandBar メソッド

その他の技術情報

デバイス クラスの一意な MFC

CCommandBar メソッド

CCommandBar メソッドを次の表に示します。

CCommandBar::AddAdornments	コマンド バーに [閉じる] ボタンを追加し、オプションで [ヘルプ] ボタンおよび [OK] ボタンを追加します。
CCommandBar::AddBitmap	このメソッドは、コマンド バーで使用可能なボタン イメージのリストに 1 つ以上のイメージを追加します。
CCommandBar::AddButtons	ツール バー コントロールに 1 つ以上のボタンを追加します。
CCommandBar::CCommandBar	**CCommandBar** オブジェクトを構築して、初期化します。
CCommandBar::Create	新しいコマンド バーをインスタンス化します。
CCommandBar::DrawMenuBar	コマンド バーのメニューが変更された後、コマンド バーの位置を変更して再描画します。
CCommandBar::GetMenu	コマンド バーのメニューへのハンドルを取得します。
CCommandBar::Height	コマンド バーの高さ (ピクセル単位) を取得します。
CCommandBar::InsertButton	このメソッドは、ツール バー コントロールに 1 つ以上のボタンを追加します。
CCommandBar::InsertComboBox	コマンド バーにコンボ ボックスを追加します。
CCommandBar::InsertMenuBar	コマンド バーにメニュー バーを追加します。
CCommandBar::InsertSeparator	コマンド バーに区分線を追加します。
CCommandBar::Show	コマンド バーを表示または非表示にします。

CCeDocList クラスは、MFC 8.0 では CDocList クラス になりました。MFC 3.0 アプリケーションからのコードを次に示します。

```
CCommandBar* pDocListCB = pDocList->GetCommandBar();
```

このコードをコンパイルすると、次のエラーが生成されます。

MainFrm.cpp(115) : **エラー** C2039: 'GetCommandBar' : 'CDocList' **のメンバではありません。**

このコードを使用する代わりに、MFC 8.0 で **CDocListCommandBar** オブジェクトを作成できます。

参照
関連項目
CCommandBar クラス
その他の技術情報
デバイス クラスの一意な MFC
CCeCommandBar

CCommandBar::AddAdornments

コマンド バーに [閉じる] ボタンを追加し、オプションで [ヘルプ] ボタンおよび [OK] ボタンを追加します。

```
BOOL AddAdornments(
Dword dwFlags = 0 );
```

パラメータ

CCommandBar クラス の **AddAdornments** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
dwFlags	コマンド バーに追加するオプション ボタンを指定します。[閉じる] ボタンのみが必要な場合は値は 0 で、他のボタンを作成する場合は次の値を組み合わせます。 **値** / **説明** CMDBAR_HELP / コマンド バーに [ヘルプ] ボタンを追加します。クリックすると、WM_HELP メッセージがこのボタンにより送信されます。 CMDBAR_OK / コマンド バーに [OK] ボタンを追加します。選択すると、メッセージ ID として IDOK が付いた WM_COMMAND メッセージがこのボタンにより送信されます。

戻り値

正常終了した場合は **true** を、それ以外の場合は **false** を返します。

解説

このメソッドは、最低限の 2 つのボタンを作成します。それらのボタンは、選択された表示要素をコマンド バーの右側に合わせる区切り記号と、選択された表示要素です。

ユーザーが [閉じる] ボタン、[OK] ボタン、または [ヘルプ] ボタンをクリックすると、そのボタンに関連付けられたメッセージがアプリケーションのメッセージ キューに配置されます。[閉じる] ボタンは、すべてのコマンド バーに必要です。[OK] ボタンと [ヘルプ] ボタンはオプションです。

メニュー、ボタン、およびコンボ ボックスなどの他のすべての要素がコマンド バーに追加された後にこのメソッドを呼び出します。

AddAdornments での依存関係を削除するように、既存のアプリケーションを再デザインする必要があります。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CCommandBar メソッド

その他の技術情報

デバイス クラスの一意な MFC

CCommandBar::AddBitmap

コマンド バーで使用可能なボタン イメージのリストに 1 つ以上のイメージを追加します。

```
int AddBitmap (
int nBitMapID,
int nNumImages );
```

パラメータ

CCommandBar クラス の **AddBitmap** メソッドのパラメータの説明を次の表に示します。

nBitMapID

追加するボタン イメージを持つビットマップのリソース ID を指定します。

nNumImages

ビットマップのボタン イメージの数を指定します。

戻り値

正常終了した場合は、最初の新しいイメージの 0 から始まるインデックスを返します。それ以外の場合は –1 を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CCommandBar メソッド

その他の技術情報

デバイス クラスの一意な MFC

CCommandBar::AddButtons

ツール バー コントロールに 1 つ以上のボタンを追加します。

```
BOOL AddButtons(
UINT uNumButtons,
LPTBBUTTON lpButtons );
```

パラメータ

CCommandBar クラス の **AddButtons** メソッドのパラメータの説明を次の表に示します。

uNumButtons

追加するボタンの数を指定します。

lpButtons

追加するボタンの情報を持つ TBBUTTON 構造体の配列へのアドレスを指定します。配列には、*nNumButtons* で指定されたボタンの数と同じ数の要素が必要です。

戻り値

正常終了した場合は **true** を、それ以外の場合は **false** を返します。

解説

lpButtons ポインタは、TBBUTTON 構造体の配列をポイントします。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照
関連項目
CCommandBar メソッド
その他の技術情報
デバイス クラスの一意な MFC
CCeCommandBar

CCommandBar::CCommandBar

CCommandBar オブジェクトのインスタンスを作成します。

```
CCommandBar ( );
```

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CCommandBar **メソッド**

その他の技術情報

デバイス クラスの一意な MFC

CCommandBar::Create

新しいコマンド バーのインスタンスを生成します。

```
BOOL CommandBar::Create(
        CWnd* pWndParent,
    DWORD dwStyle,
    UINT nBarID = AFX_IDW_TOOLBAR
);
```

パラメータ

CCommandBar クラス の **Create** メソッドのパラメータの説明を次の表に示します。

pWndParent

コマンド バー オブジェクトの親へのポインタを指定します。

dwStyle

コマンド バーのスタイルを指定します。「解説」を参照してください。

nBarID

省略可能なパラメータです。ツール バーの子ウィンドウ ID として AFX_IDW_TOOLBAR を指定します。

戻り値

正常終了した場合は **true** を、それ以外の場合は **false** を返します。

解説

Create 関数により、空のコマンド バーが作成されます。Pocket PC および Smartphone では、コマンド バーがウィンドウの一番下に作成されます。Windows CE 5.0 では、**CDialog::m_pWndEmptyCB** がサポートされなくなったため、開発者が作成処理および挿入処理を制御します。以前、このメンバ変数は、Pocket PC の空の **CommandBar** または **MenuBar** を指すために使用されていました。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CCommandBar メソッド

その他の技術情報

デバイス クラスの一意な MFC

CCommandBar::DrawMenuBar

コマンド バーのメニューが変更された後、コマンド バーの位置を変更して再描画します。

```
BOOL DrawMenuBar(
WORD wButton )
const;
```

パラメータ

CCommandBar クラス の **DrawMenuBar** メソッドのパラメータの説明を次の表に示します。

wButton

コマンド バー内のメニュー バーの 0 から始まるインデックスを指定します。

戻り値

正常終了した場合は **true** を、それ以外の場合は **false** を返します。

解説

コマンド バーのメニューを変更した後、必ずこのメソッドを呼び出します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照
関連項目
CCommandBar メソッド
その他の技術情報
デバイス クラスの一意な MFC
CCeCommandBar

CCommandBar::GetMenu

コマンド バーのメニューのハンドルを取得します。

```
HMENU CommandBar_GetMenu();
```

戻り値

成功した場合はメニューのハンドルを返します。失敗した場合は NULL を返します。

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CCommandBar **メソッド**

その他の技術情報

デバイス クラスの一意な MFC

CCeCommandBar

CCommandBar::Height

コマンド バーの高さ (ピクセル単位) を取得します。

```
int Height();
```

戻り値

コマンド バーの高さをピクセル単位で返します。

解説

コマンド バーで占有されるのは、アプリケーションのメイン ウィンドウのクライアント領域の一部です。そのため、クライアント領域の実際のサイズを決定する手段として、この関数を呼び出すことができます。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CCommandBar メソッド

その他の技術情報

デバイス クラスの一意な MFC

CCommandBar::InsertButton

ツール バー コントロールに 1 つ以上のボタンを追加します。

```
BOOL InsertButton(
int nButton,
LPTBBUTTON lpButton );
```

パラメータ

CCommandBar クラス の **InsertButton** メソッドのパラメータの説明を次の表に示します。

nButton

ボタンの 0 から始まるインデックスを指定します。このメソッドは、このボタンの左に新しいボタンを挿入します。

lpButton

挿入するボタンについての情報を含んでいる TBUTTON 構造体のアドレスを指定します。

戻り値

正常終了した場合は **true** を、それ以外の場合は **false** を返します。

解説

lpButton ポインタは、TBUTTON 構造体をポイントします。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CCommandBar メソッド

その他の技術情報

デバイス クラスの一意な MFC

CCeCommandBar

CCommandBar::InsertComboBox

コマンド バーにコンボ ボックスを挿入します。

```
CComboBox* InsertComboBox(
int nWidth,
DWORD dwStyle,
WORD wComboBoxID,
WORD wButton );
```

パラメータ

CCommandBar クラス の **InsertComboBox** メソッドのパラメータの説明を次の表に示します。

nWidth

コンボ ボックスの幅 (ピクセル数) を指定します。

dwStyle

コンボ ボックスに適用されるウィンドウ スタイルを指定します。WS_VISIBLE および WS_CHILD スタイルは自動的に適用されます。既定のスタイルは、CBS_DROPDOWNLIST および WS_SCROLL です。

wComboBoxID

コンボ ボックスの識別子を指定します。

wButton

コマンド バーのボタンの 0 から始まるインデックスを指定します。

戻り値

正常終了した場合は *wComboBoxID* により識別されるメニューへのポインタを、それ以外の場合は **null** を返します。

解説

このメソッドは、w*Button* により識別されるボタンの左側にコンボ ボックスを挿入します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CCommandBar メソッド

その他の技術情報

デバイス クラスの一意な MFC

CCeCommandBar

CCommandBar::InsertMenuBar

メニュー バーをコマンド バーに挿入します。

```
BOOL InsertMenuBar(
WORD wMenuID,
WORD wButton );
```

パラメータ

CCommandBar クラス の **InsertMenuBar** メソッドのパラメータの説明を次の表に示します。

wMenuID

MenuBar のリソース ID を指定します。

wButton

コマンド バーのボタンの 0 から始まるインデックスを指定します。

戻り値

正常終了した場合は **true** を、それ以外の場合は **false** を返します。

解説

このメソッドは、*wButton* により示されるボタンの位置に **MenuBar** を配置します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CCommandBar メソッド

その他の技術情報

デバイス クラスの一意な MFC

CCeCommandBar

CCommandBar::InsertSeparator

コマンド バーに区切りボタンを挿入します。

```
BOOL InsertSeparator(
int nWidth = 6,
WORD wButton );
```

パラメータ

CCommandBar クラス の **InsertSeparator** メソッドのパラメータの説明を次の表に示します。

nWidth

区切りボタンの幅 (ピクセル数) を指定します。

wButton

コマンド バーで、ボタンの 0 から始まるインデックスを指定します。このインデックスは、0 以上で、バー上のボタンの数より少ない数になります。CMDBAR_END は **CommandBar** の終わりを指定します。

戻り値

正常終了した場合は **true** を、それ以外の場合は **false** を返します。

解説

このメソッドは、*wButton* により示されるボタンの位置に区切り記号を配置します。区切り記号は、ユーザー入力を受け取りません。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : afxext.h で宣言されます。

参照

関連項目

CCommandBar メソッド

その他の技術情報

デバイス クラスの一意な MFC

CCeCommandBar

CCommandBar::Show

コマンド バーを表示または非表示にします。

```
BOOL Show(BOOL bShow)
```

パラメータ

CCommandBar クラス の **Show** メソッドのパラメータの説明を次の表に示します。

bShow

コマンド バーを表示するか非表示にするかを指定するブール値。このパラメータを **false** に設定するとコマンド バーが表示され、**true** に設定するとコマンド バーが非表示になります。

戻り値

CCommandBar の以前の表示状態を返します。コマンド バーが表示されていた場合は **true**、表示されていなかった場合は **false** を返します。

解説

コマンド バーは、表示状態を **true** に設定した状態で作成されます。コマンド バーを非表示にするには、`Show(false)` を呼び出します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CCommandBar メソッド

その他の技術情報

デバイス クラスの一意な MFC

CCeCommandBar

CDocList クラス

CDocList を使用して、一貫した方法でドキュメントの一覧を表示します。CDocList は、DocList ウィンドウ タイプのオペレーティング システム実装をラップします。MFC Doc/View **アーキテクチャと統合するための追加機能が含まれています。**

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

その他の技術情報

デバイス クラスの一意な MFC

CDocList **メソッド**

CCeDocList

CDocList メソッド

CDocList クラス のメソッドを次に示します。このクラスは、**DocList** ウィンドウ タイプのオペレーティング システム実装をラップします。

CDocList::CDocList	**CDocList** オブジェクトを構築します。
CDocList::Create	**CDocList** オブジェクトに関連付けられているリストを作成し、初期化します。
CDocList::DeleteSelection	選択されたアイテムを削除します。
CDocList::DisableUpdate	**CDocList** の更新を無効にします。
CDocList::EnableUpdate	**CDocList** の更新を有効にします。
CDocList::GetFilterIndex	たとえば、コンボ ボックスの現在のフィルタを設定するなどのために、現在のファイル フィルタの選択内容を取得します。
CDocList::GetItemCount	アイテムのカウントを取得します。
CDocList::GetNextItem	**CDocList** 内の次のアイテムを取得します。
CDocList::GetNextWaveFile	次の .wav ファイルを選択します。
CDocList::GetPrevWaveFile	前の .wav ファイルを選択します。
CDocList::GetSelectCount	選択されたアイテムの数を取得します。
CDocList::GetSelectedPathname	選択されたアイテムに関連付けられたパス名を取得します。
CDocList::ReceiveIR	赤外線信号を受信します。
CDocList::Refresh	**CDocList** を更新します。
CDocList::RenameMoveSelectedItems	選択されたアイテムの [名前の変更/移動] ダイアログ ボックスを表示します。
CDocList::SelectItem	パスに関連付けられたアイテムを選択します。
CDocList::SendEMail	電子メールを送信します。
CDocList::SendIR	赤外線信号を送信します。
CDocList::SetFilterIndex	ファイル フィルタが変更されたことを示します。たとえば、ユーザーが [オプション] ダイアログ ボックスの設定を変更した場合などです。
CDocList::SetFolder	フォルダを設定します。
CDocList::SetItemState	**CDocList** 内のアイテムの状態を設定します。
CDocList::SetSelectedPathname	パスに関連付けられたアイテムを、次の更新中に選択するように設定します。

CDocList::SetSelection	アイテムをインデックスにより選択します。
CDocList::SetSortOrder	**CDocList** を並べ替えます。
CDocList::Update	**CDocList** を更新しますが、選択内容は元に戻しません。

CCeDocList クラスは、MFC 8.0 では CDocList クラス になりました。MFC 3.0 アプリケーションからのコードを次に示します。

```
CCommandBar* pDocListCB = pDocList->GetCommandBar();
```

このコードをコンパイルすると、次のエラーが生成されます。

MainFrm.cpp(115) : エラー C2039: 'GetCommandBar' : 'CDocList' のメンバではありません。

MFC 8.0 では、代わりに **CDocListCommandBar** のインスタンスを作成します。

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::CDocList

CDocList オブジェクトのインスタンスを作成します。

```
CDocList();
```

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList **クラス**

その他の技術情報

デバイス クラスの一意な MFC

CDocList::Create

CDocList オブジェクトに関連付けられているリストを作成し、初期化します。

```
BOOL Create (
CWnd* pParentWnd,
LPCTSTR lpszFilterList,
LPCTSTR lpszFolder,
DWORD dwFlags = DLF_SHOWEXTENSION ) ;
```

パラメータ

CDocList クラス の **Create** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
pParentWnd	親ウィンドウへのポインタ (必須)。
lpszFilterList	フィルタのリストへのポインタ。リストは縦棒 (|) により区切られ、||\0 で終わります。
lpszFolder	フォルダへのポインタ。
dwFlags	現在のバージョンで使用できる値は、DLF_SHOWEXTENSION のみです。

戻り値

リストが正常に作成されて表示された場合は **true** を返します。それ以外の場合は、**false** を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照
関連項目
CDocList クラス
その他の技術情報
デバイス クラスの一意な MFC

CDocList::DeleteSelection

選択されたアイテムを削除します。

```
HRESULT DeleteSelection();
```

戻り値

正常終了した場合は true を、それ以外の場合は false を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CDocList::DisableUpdate

ドキュメント リストの更新を無効にします。

```
void DisableUpdate();
```

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList **クラス**

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::EnableUpdate

ドキュメント リストの更新を有効にします。

```
void EnableUpdate();
```

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList **クラス**

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::GetFilterIndex

現在のファイル フィルタの選択内容を取得します。

```
int GetFilterIndex();
```

戻り値

フィルタの現在の選択内容に関する、縦棒 (|) で区切られたフィルタ リストのインデックスです。インデックスでは、インデックス番号が 0 **からではなく、**1 **から始まります。**

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList **クラス**

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::GetItemCount

アイテムのカウントを取得します。

```
int GetItemCount();
```

戻り値

ドキュメント リストのドキュメント数を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::GetNextItem

ドキュメント リスト内の次のアイテムを取得します。

```
int GetNextItem(
int nStart,
UINT nFlags );
```

パラメータ

CDocList クラス の **GetNextItem** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
nStart	検索を開始するドキュメント リスト内のアイテム番号を指定します。
nFlags	要求されたアイテムと指定したアイテムの幾何学的関係と、要求されたアイテムの状態を指定します。幾何学的関係は、次の値の 1 つを指定できます。

値	説明
LVNI_ABOVE	指定したアイテムの上にあるアイテムを検索します。
LVNI_ALL	インデックス順で次のアイテムを検索します (既定値)。
LVNI_BELOW	指定したアイテムの下にあるアイテムを検索します。
LVNI_TOLEFT	指定したアイテムの左にあるアイテムを検索します。
LVNI_TORIGHT	指定したアイテムの右にあるアイテムを検索します。

さらに、アイテムの状態を指定することもできます。0 にするか、次の値のうち 1 つ以上を指定できます。

値	説明
LVNI_DROPHILITED	アイテムに LVIS_DROPHILITED 状態フラグが設定されています。
LVNI_FOCUSED	アイテムに LVIS_FOCUSED 状態フラグが設定されています。
LVNI_SELECTED	アイテムに LVIS_SELECTED 状態フラグが設定されています。

解説

指定した状態フラグがアイテムに設定されていない場合は、次のアイテムを続けて検索します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::GetNextWaveFile

次の .wav ファイルを取得します。

```
BOOL GetNextWaveFile(
int* pnItem );
```

パラメータ

CDocList クラス の **GetNextWaveFile** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
pnItem	リスト内の次の .wav ファイルのポインタです。

戻り値

正常終了した場合は **true** を、それ以外の場合は **false** を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CDocList::GetPrevWaveFile

前の .wav **ファイルを取得します。**

```
BOOL GetPrevWaveFile(
int* pnItem );
```

パラメータ

CDocList **クラス** の **GetPrevWaveFile** **メソッドのパラメータの説明を次の表に示します。**

パラメータ	**説明**
pnItem	**リスト内の前の** .wav **ファイルのポインタです。**

戻り値

正常終了した場合は **true** **を、それ以外の場合は** **false** **を返します。**

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList **クラス**

その他の技術情報

デバイス クラスの一意な MFC

CDocList::GetSelectCount

選択されたアイテムの数を取得します。

```
int GetSelectCount();
```

戻り値

ドキュメント リストで選択されたアイテムの数を返します。

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList **クラス**

その他の技術情報

デバイス クラスの一意な MFC

CDocList::GetSelectedPathname

選択されたアイテムに関連付けられたパスを取得します。

```
BOOL GetSelectedPathname(
LPTSTR pszPath,
WORD nSize );
```

パラメータ

CDocList クラス の **GetSelectedPathname** メソッドのパラメータの説明を次の表に示します。

pszPath

ドキュメント リスト内の選択されたアイテムのパスへのポインタです。

nSize

パスを受け取るバッファの長さを、文字数で指定します。

戻り値

正常終了した場合は true を、それ以外の場合は false を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照
関連項目
CDocList クラス
その他の技術情報
デバイス クラスの一意な MFC

CDocList::ReceiveIR

赤外線情報の受信に使用されます。

```
void ReceiveIR(
LPTSTR pszPath );
```

パラメータ

CDocList クラス の **ReceiveIR** メソッドのパラメータの説明を次の表に示します。

pszPath

パスを含む文字列へのポインタです。

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::Refresh

ドキュメント リストを更新します。

```
BOOL Refresh();
```

戻り値

正常終了した場合は true を、それ以外の場合は false を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::RenameMoveSelectedItems

選択されたアイテムの [名前の変更/移動] ダイアログ ボックスを表示します。

```
BOOL RenameMoveSelectedItems();
```

戻り値

正常終了した場合は true を、それ以外の場合は false を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CDocList::SelectItem

パスに関連付けられたアイテムを選択します。

```
HRESULT SelectItem(
LPTSTR pszPath,
BOOL fVisible );
```

パラメータ

CDocList クラス の **SelectItem** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
pszPath	パスを含む文字列へのポインタです。
fVisible	ドキュメントが表示されるのか、非表示にされるのかを示す値。

戻り値

正常終了した場合は **true** を、それ以外の場合は **false** を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::SendEMail

電子メールの送信に使用されます。

```
void SendEMail(
LPTSTR pszPath );
```

パラメータ

CDocList クラス の **SendEMail** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
pszPath	パスを含む文字列へのポインタです。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::SendIR

赤外線情報の送信に使用されます。

```
void SendIR(
LPTSTR pszPath );
```

パラメータ

CDocList クラス の **SendIR** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
pszPath	パスを含む文字列へのポインタです。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::SetFilterIndex

ファイル フィルタが変更されたことを示します。

```
HRESULT SetFilterIndex(
int nIndex );
```

パラメータ

CDocList クラス の **SetFilterIndex** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
nIndex	縦棒 (|) で区切られるフィルタ リストのインデックスです。インデックスでは、インデックス番号が 0 からではなく、1 から始まります。

戻り値

正常終了した場合は **true** を、それ以外の場合は **false** を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::SetFolder

フォルダを設定します。

```
void SetFolder(
LPTSTR pszFolder );
```

パラメータ

CDocList クラス の **SetFolder** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
pszFolder	フォルダ名を含む文字列へのポインタです。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::SetItemState

ドキュメント リスト内のアイテムの状態を変更します。

```
BOOL SetItemState(
int nItem,
UINT nState,
UINT nStateMask );
```

パラメータ

CDocList クラス の **SetItemState** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
nItem	アイテムのインデックス番号です。
nState	変更する状態の値を指定します。
nStateMask	どの状態が変更できるかを指定します。

戻り値

正常終了した場合は **true** を、それ以外の場合は **false** を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::SetSelectedPathname

パスに関連付けられたアイテムを、次の更新中に選択するように設定します。

```
void SetSelectedPathname(
LPTSTR pszPath );
```

パラメータ

CDocList クラス の **SetSelectedPathname** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
pszPath	パスを含む文字列へのポインタです。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::SetSelection

アイテムをインデックスにより選択します。

```
BOOL SetSelection(
int wItem );
```

パラメータ

CDocList クラス の **SetSelection** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
wItem	アイテムのインデックス番号です。

戻り値

正常終了した場合は true を、それ以外の場合は false を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CDocList::SetSortOrder

ドキュメント リストを並べ替えます。並べ替え順序は、常に昇順です。

```
BOOL SetSortOrder();
```

戻り値

正常終了した場合は true を、それ以外の場合は false を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocList::Update

ドキュメント リストを更新しますが、選択内容は元に戻しません。

```
HRESULT Update();
```

戻り値

正常終了した場合は true を、それ以外の場合は false を返します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h **で宣言されます。**

参照

関連項目

CDocList クラス

その他の技術情報

デバイス クラスの一意な MFC

CCeDocList

CDocListDocTemplate クラス

このクラスは、SDI アプリケーション タイプの改良版で、CSingleDocTemplate クラス により実現されます。このクラスを使用すると、ドキュメントが開いていないときにアプリケーションを **DocList** モードにすることができます。

このドキュメント タイプをアプリケーションに追加した後、**ShowDocList** メンバを呼び出して、最初に **CDocList** クラスのインスタンスを作成します。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

その他の技術情報

デバイス クラスの一意な MFC

CDocListDocTemplate メソッド

CDocListDocTemplate クラス は、SDI アプリケーション タイプの改良版であるデスクトップの **CDocListDocTemplate** と同様のクラスです。

このクラスを使用すると、ドキュメントが存在しないときに、アプリケーションがドキュメント モードで存在するようにします。

このドキュメント タイプをアプリケーションに追加した後、**ShowDocList** メンバを呼び出して、**CDocList** インスタンスを作成します。

このセクションの内容

CDocListDocTemplate::CDocListDocTemplate

CDocListDocTemplate オブジェクトを構築します。

CDocListDocTemplate::ShowDocList

ドキュメント リストを表示します。**CDocList** は、**CDocListDocTempate::ShowDocList** が呼び出されたときに自動的に作成されます。

参照

その他の技術情報

デバイス クラスの一意な MFC

CCeDocListDocTemplate

CDocListDocTemplate::CDocListDocTemplate

CDocListDocTemplate オブジェクトを構築します。このクラスを使用すると、**CDocListDocTemplate** オブジェクトを動的に割り当てて、それをアプリケーション クラスの **InitInstance** メソッドから **CWinApp::AddDocTemplate** に渡すことができます。

```
CDocListDocTemplate(
UINT nIDResource,
CRuntimeClass* pDocClass,
CRuntimeClass* pFrameClass,
CRuntimeClass* pViewClass,
LPCTSTR lpszFilterList,
LPCTSTR lpszFolder );
```

パラメータ

CDocListDocTemplate クラス の **CDocListDocTemplate** メソッドのパラメータの説明を次の表に示します。

パラメータ	説明
nIDResource	ドキュメント型で使用するリソースの ID を指定します。メニュー、アイコン、アクセラレータ テーブル、文字列リソースなどを指定できます。
pDocClass	ドキュメント クラスの **CRuntimeClass** オブジェクトへのポインタ。このクラスは、ドキュメントを表示するために定義した **CDocument** の派生クラスです。
pFrameClass	フレーム ウィンドウ クラスの **CRuntimeClass** オブジェクトへのポインタ。このクラスとして、**CFrameWnd** クラスの派生クラス、またはメイン フレーム ウィンドウの既定の動作でよければ、**CFrameWnd** クラスを使用できます。
pViewClass	ビュー クラスの **CRuntimeClass** オブジェクトへのポインタ。このクラスは、ドキュメントを表示するために定義した **CView** の派生クラスです。
lpszFilterList	フィルタのリストへのポインタ。リストは縦棒 (|) により区切られ、||\0 で終わります。
lpszFolder	フォルダへのポインタ。

解説

詳細については、「CDocListDocTemplateCDocListDocTemplate」を参照してください。

必要条件

Windows CE のバージョン 5.0 以降。

ヘッダー ファイル : Afxext.h で宣言されます。

参照

その他の技術情報

デバイス クラスの一意な MFC

CCeDocListDocTemplate

CDocListDocTemplate::ShowDocList

ドキュメント リストを最初に表示するために、CWinApp::InitInstance からの呼び出しとして使用されます。

```
virtual void ShowDocList();
```

必要条件

Windows CE **のバージョン** 5.0 **以降。**

ヘッダー ファイル : Afxwin.h **で宣言されます。**

参照

その他の技術情報

デバイス クラスの一意な MFC

AfxEnableDRA

この関数は、デバイス アプリケーション プロジェクトでデバイス解像度認識を有効にします。

```
void AfxEnableDRA(BOOL bEnable);
```

パラメータ
bEnable

TRUE を指定するとデバイス解像度認識が有効になり、**FALSE** を指定するかこの関数を呼び出さないとデバイス解像度認識が無効になります。

解説

デバイス解像度認識機能を使うと、アプリケーションでは、縦向きモードから横向きモードへの変更など、実行時に解像度の変更に応答できます。

CDialog を直接インスタンス化するときに **AfxEnableDRA()** 関数を使用します。この場合には、dlgcore.cpp に定義され、MFC DLL と LIB に実装された **OnSize** メソッドを使用することになります。これらのライブラリ バージョンでは、**AfxEnableDRA(TRUE)** が以前に呼び出されたことがある場合にのみ **DRA::RelayoutDialog(...)**. **AfxIsDRAEnabled()** 呼び出しが true を返すかどうかを実行時に確認するために **AfxIsDRAEnabled()** が使用されます。

メモ :

ウィザードを使ってデバイスに MFC プロジェクトを作成すると、生成されるコードには、**CDialog** 派生クラスのオーバーライド (**CDialog::OnSize(int, int)**) が実装されます。デバイス解像度認識がコンパイル時にチェックされ、**DRA::RelayoutDialog(...)** の呼び出しを実行するかどうかが決定されます。

使用例

```
AfxEnableDRA(TRUE); //Enable Device Resolution Awareness
...
void CDialog::OnSize(UINT nType, int cx, int cy)
{
    if (AfxIsDRAEnabled())
    {
        DRA::RelayoutDialog(
            AfxGetInstanceHandle(),
            this->m_hWnd,
            DRA::GetDisplayMode() != DRA::Portrait ?
                m_lpszWideTemplateName : m_lpszTemplateName);
    }
    else
    {
        CWnd::OnSize(nType, cx, cy);
    }
}
```

参照
その他の技術情報
デバイス クラスの一意な MFC

デバイスの C ランタイム ライブラリ リファレンス

デバイスの Microsoft C ランタイム ライブラリでは、デスクトップの完全な Microsoft C ランタイム ライブラリで使用可能な関数のサブセットがサポートされます。

完全なライブラリのサブセットだけをサポートすることにより、デスクトップ コンピュータよりもリソースが制限されているデバイスをアプリケーションが実行できるようになります。デバイスのランタイム ライブラリ リファレンスのヘルプ トピックをフィルタ処理する方法の詳細については、「方法 : デバイスでサポートされる MFC クラスおよびメソッドのヘルプを検索する」および「方法 : デバイスでサポートされる ATL クラスおよびメソッドのヘルプを検索する」を参照してください。

C ランタイム ライブラリ

デバイスの C ランタイム ライブラリの完全なドキュメントについては、「Windows CE 用の Microsoft C ランタイム ライブラリ」を参照してください。

セキュリティで保護された API

Visual Studio 2005 のデバイス プロジェクトには、次の API が追加されています。

wcsncpy_s	wcscpy_s	wcscat_s
strncpy_s	strcpy_s	strcat_s
memmove_s	memcpy_s	_wsplitpath_s
_wmakepath_s	_ultoa_s	_ui64tow_s
_ui64toa_s	_ltoa_s	_localtime64_s
_itow_s	_itoa_s	_i64tow_s
_i64toa_s	_gmtime64_s	

参照

その他の技術情報

リファレンス (デバイス)

デバイス用の標準 C++ ライブラリ リファレンス

標準 C++ ライブラリのデバイス バージョンは、完全なStandard C++ Library Referenceのサブセットです。

新機能

このバージョンの標準 C++ ライブラリには、ストリームのサポートが追加されました。

サポートされていない機能

- デバイス用の標準 C++ ライブラリでは、ロケールがサポートされていません。
- `uncaught_exception` は、Windows Mobile 2005 プラットフォームを含む、Windows CE 5.0 以降のバージョンでしかサポートされていません。

サポートされていないヘッダー

標準 C++ ライブラリのデバイス バージョンでは、次のヘッダーがサポートされていません。

- <cerrno>
- <csignal>
- <locale>

参照

その他の技術情報

リファレンス (デバイス)
Visual C++ ライブラリのリファレンス

スマート デバイスのコンパイラ

Visual Studio 2005 には、スマート デバイスで使用するマイクロプロセッサを対象にした次のコンパイラが用意されています。

- 32-bit C/C++ コンパイラ。32 ビット ARM C および C++ プログラムのコンパイルとリンクに使用します。
- 32-bit C/C++ コンパイラ。32 ビット Renesas SH-4 C および C++ プログラムのコンパイルとリンクに使用します。
- C/C++ コンパイラ。MIPS16、MIPS32、MIPS64 C および C++ プログラムのコンパイルとリンクに使用します。

上記コンパイラによって、共通オブジェクト ファイル形式 (COFF) オブジェクト ファイルが生成されます。

このセクションのトピック

デバイスでサポートされるマイクロプロセッサ

デバイス コンパイラがサポートするマイクロプロセッサ ファミリの一覧を記載しています。

デスクトップ コンパイラとデバイス コンパイラの違い

コンパイラ オプション、組み込み関数、および定義済みマクロに関する参照情報について記載しています。また、スマート デバイスをプログラムする際に重要となることのあるデータ配置と構造化例外処理の違いについても説明しています。

ARM ファミリ プロセッサ

ARM テクノロジのコンパイラ オプション、組み込み関数、および呼び出し仕様について説明しています。

Renesas ファミリ プロセッサ

Renesas テクノロジのコンパイラ オプション、組み込み関数、および呼び出し仕様について説明しています。

MIPS ファミリ プロセッサ

MIPS テクノロジのコンパイラ オプション、組み込み関数、および呼び出し仕様について説明しています。

関連セクション

スマート デバイスのプロジェクトのオンライン ヘルプ

Visual Studio 2005 でのスマート デバイス プログラミングのサポートについて説明しています。

デバイスでサポートされるマイクロプロセッサ

デバイス コンパイラは、デバイスで使用されるさまざまなマイクロプロセッサをサポートします。サポート対象には、コンパイラ、完全な最適化を可能にする組み込み関数、および他の方法ではサポートされないタスクを行うアセンブラが含まれます。

サポートされるマイクロプロセッサ

次の一覧は、サポートされるマイクロプロセッサ ファミリを示しています。

- アーキテクチャ v4、v4T、Thumb、v5TE、および Intel XScale についての ARM のライセンス付きテクノロジ
  ARM のライセンス付きマイクロプロセッサの詳細については、ARM Web サイトを参照してください。
  ARM のライセンス付き Intel マイクロプロセッサについては、Intel Web サイトを参照してください。
- Renesas SuperH SH4 マイクロプロセッサ
  Renesas マイクロプロセッサについては、Renesas Web サイトを参照してください。
- NEC、Toshiba、Philips Semiconductor、Integrated Device Technologies、LSI Logic、および Quantum Effect Design が開発した、MIPS のライセンス付きテクノロジ
  - MIPS のライセンス付きテクノロジについては、MIPS Web サイトを参照してください。
  - MIPS のライセンス付き NEC マイクロプロセッサについては、NEC Web サイトを参照してください。
  - MIPS のライセンス付き Toshiba マイクロプロセッサについては、Toshiba Web サイトを参照してください。
  - MIPS のライセンス付き Philips Semiconductor マイクロプロセッサについては、Philips Semiconductor Web サイトを参照してください。

関連項目

その他のリソース

デスクトップ コンパイラとデバイス コンパイラの違い

デスクトップ コンパイラとデバイス コンパイラの違い

以下のトピックでは、デスクトップ コンパイラとデバイス コンパイラの動作の違いについて説明するほか、組み込み関数とコンパイラ エラーに関する詳細な参照情報も示します。

このセクションのトピック

固有のビルド オプション

デバイス コンパイラに対して固有に定義されたビルド オプションについて説明します。

共有ビルド オプション

デスクトップ コンパイラで共有されるビルド オプションの一覧を示します。

デバイス コンパイラの組み込み関数

すべてのデバイス コンパイラに共通の組み込み関数に関する詳細な参照情報を提供します。

定義済みマクロ

デバイス コンパイラに備えられているマクロに関する参照情報を提供します。

デバイス コンパイラのデータ配置に関する問題

マイクロプロセッサ間で異なるデータ配置の問題に関する情報をプログラマに提供します。

デバイス コンパイラの例外処理

RISC ベースのマイクロプロセッサ コンパイラで例外処理を行う固有の方法について説明します。

デバイス コンパイラ エラー メッセージ

デバイス コンパイラの使用時に表示されることがあるコンパイラおよびリンカのエラー メッセージに関する情報を提供します。

関連セクション

ARM ファミリ プロセッサ

ARM テクノロジのコンパイラ オプション、組み込み関数、および呼び出し仕様について説明しています。

Renesas ファミリ プロセッサ

Renesas テクノロジのコンパイラ オプション、組み込み関数、および呼び出し仕様について説明しています。

MIPS ファミリ プロセッサ

MIPS テクノロジのコンパイラ オプション、組み込み関数、および呼び出し仕様について説明しています。

固有のビルド オプション

デスクトップ コンパイラに提供されているほぼすべてのビルド オプションは、デバイス コンパイラでも使用でき、実装方法も同じです。リンカ オプションの実装も、どちらの環境でも同じです。

デスクトップ コンパイラとデバイス コンパイラに共通のコンパイラ オプションの一覧については、「共有ビルド オプション」を参照してください。

次の表では、デスクトップの実装とは大きく異なる方法でサポートされているデバイス コンパイラのコンパイラ オプションを示します。

オプション	説明
/callcap – callcap プロファイルの有効化	各関数の最初と最後に callcap プロファイル フックを挿入します。
/GS – セキュリティ チェックの有効化	スタック チェックを有効にして、バッファ オーバーラン攻撃を検出します。

また、サポートされている各マイクロプロセッサ ファミリには、独自の機能を実装したプロセッサ固有のコンパイラ オプションがあります。

関連項目

参照情報

ARM コンパイラ オプション
Renesas コンパイラ オプション
MIPS コンパイラ オプション

/callcap – callcap プロファイルの有効化

/callcap オプションを使用すると、コンパイラによって各関数の最初と最後にプロファイル フックへの呼び出しが挿入されます。

プロファイル フックをコンパイルする場合は、**/callcap** スイッチを使用せずにコンパイルする必要があります。**/callcap** スイッチを使用してプロファイル フック関数をコンパイルすると、その関数自体への無限の再帰呼び出しを実行することになります。

次のコード callcaphooks.c では、callcap を使用しないコンパイルのプロファイル フック関数 **_CAP_Enter_Function** の例を示します。

/callcap コンパイラ スイッチを使用してコンパイルすると、**/callcap** スイッチによってすべての関数にプロファイル フックへの呼び出しが挿入されます。

```
int main()
{
    double d0 = 2.0, d1 = 4.0, result;
    // No profiling for printf, because the library
    // function is not compiled with /callcap.
    printf("\n");
//  callcap profiling hooks are called
//  after the function is entered and before
//  leaving it.
// The following example shows how to insert profiling hooks.
// File: callcaphooks.c

#include <stdio.h>
int main();
void _CAP_Enter_Function(void *p)
{
    if (p != main)
        printf("Enter function   (at address %p) at %d\n",
            p, GetTickCount());
        return;
}
void _CAP_Exit_Function(void *p)
{
    if (p != main)
        printf("Leaving function (at address %p) at %d\n",
            p, GetTickCount());
    return;
}
```

関連項目
概念
固有のビルド オプション

/GS – セキュリティ チェックの有効化

/GS ビルド オプションを使用してビルドすると、コンパイラはリターン アドレスへの直接バッファ オーバーランを検出しようとします。バッファ オーバーランによってリターン アドレスが上書きされると、バッファ サイズが制限されていないコードが悪用されるおそれがあります。

バッファ オーバーランの例を次に示します。**/GS** を使用してビルドすると、メッセージ ボックスが表示されて処理が終了します。

```
#include <cstring>

// Vulnerable function
void vulnerable(const char *str)
{
   char buffer[10];
   strcpy(buffer, str); // overrun buffer !!!
}

int main()
{
   // declare buffer that is bigger than expected
   char large_buffer[] = "This string is longer than 10 characters!!!";
   vulnerable(large_buffer);
}
```

備考

x86 などのコンピュータの呼び出し規則では、関数呼び出しのリターン アドレスがスタックで渡されるため、バッファ オーバーランの悪用がもっと簡単です。

/GS を使用して関数をコンパイルした場合、コンパイラは、バッファ オーバーランの悪用を避けるために、バッファ オーバーランの問題が発生する可能性のある関数を識別し、スタック内の関連するリターン アドレスの前にセキュリティ Cookie を挿入します。関数の終了時にセキュリティ Cookie が変更された場合は、コンパイラがエラーを報告し、処理を終了します。

ただし、スマート デバイスで作業する場合、セキュリティ Cookie が問題になることがあります。既定の CRT エントリ ポイントを使用せずに CRT 起動コードから **_cinit** を呼び出す EXE または DLL の場合、**_cinit** によってセキュリティ Cookie の予期される値がリセットされるため、コンパイラはエラーを報告します。**_cinit** によってセキュリティ Cookie の値が変更された場合、コンパイラはバッファ オーバーランであると誤認識し、エラーを報告して処理を終了します。

この問題を回避するには

- **_cinit** を呼び出す関数では、配列または **_alloca** を使用しないようにします。
- 通常どおり **WinMainCRTStartup** や **_DllMainCRTStartup** などの既定のエントリ ポイントを使用して CRT を初期化します。

/GS は、バッファ オーバーランを悪用したすべての攻撃からセキュリティを保護できるわけではありません。たとえば、バッファ オーバーラン攻撃は、パラメータ領域を上書きすることによっても可能です。

/GS を使用する場合でも、安全なコードを記述する必要があります。つまり、バッファ オーバーランが発生するようなコードを記述しないようにします。バッファ サイズを制限することによって、コンパイル済みコードにセキュリティ チェックを挿入することができます。

関連項目
その他のリソース
デスクトップ コンパイラとデバイス コンパイラの違い

共有ビルド オプション

次の表は、デスクトップ ワークステーションおよびスマート デバイスを対象とするコンパイラなど、サポートされるすべてのコンパイラでサポートされているオプションを示しています。これらのオプションの詳細については、Visual Studio .NET の統合ドキュメントにある Visual C++ リファレンスを参照してください。

/?	/c (リンクなしにコンパイルする)	/C (前処理時にコメントを保存)
/D (プリプロセッサの定義)	/E (stdout に前処理を行う)	/EH (例外処理モデル)
/EP (#line ディレクティブなしに stdout に前処理を行う)	/F (スタック サイズを設定する)	/FA、/Fa (ファイルの一覧)
/Fd (プログラム データベース ファイル名)	/Fe (EXE ファイル名の指定)	/FI (強制インクルード ファイル名の指定)
/Fm (MAP ファイル名の指定)	/Fo (オブジェクト ファイル名)	/Fp (.pch ファイル名の指定)
/FR、/Fr (.sbr ファイルの作成)	/GF (重複文字列の削除)	/GL (プログラム全体の最適化)
/GR (ランタイム型情報の有効化)	/GX (例外処理の有効化)	/Gy (関数レベルのリンクの有効化)
/GZ (スタック フレームのランタイム エラー チェックの有効化)	/H (外部名の長さの制限)	/HELP (コンパイラのコマンドライン ヘルプ)
/I (追加のインクルード ディレクトリ)	/J (char 型を既定で符号なしにする)	/link (リンカにオプションを渡す)
/MD、/MT、/LD (ランタイム ライブラリの使用)	/nologo (起動バナーの抑止) (C/C++)	/O1、/O2 (サイズの最小化、速度の最大化)
/Ob (インライン関数の展開)	/Od (無効化 (デバッグ))	/Og (グローバルな最適化)
/Oi (組み込み関数の生成)	/Os、/Ot (小さいサイズのコードを優先、高速のコードを優先)	/Ox (最大限の最適化)
/P (ファイルに前処理を行う)	/RTC (ランタイム エラー チェック)	/showIncludes (インクルード ファイルの一覧)
/Tc、/Tp、/TC、/TP (ソース ファイルの種類の指定)	/U、/u (シンボルを定義しない)	/V (バージョン番号)
/vd (構造のディスプレイスメントの無効化)	/vmb、/vmg (表記方法)	/vmm、/vms、/vmv (汎用の表記)
/w、/Wn、/WX、/Wall、/wln、/wdn、/wen、/won (警告レベル)	/WL (1 行診断の有効化)	/Wp64 (64 ビット移植性の問題の検出)
/X (標準インクルード パスを無視する)	/Y (プリコンパイル済みヘッダー)	/Yc (プリコンパイル済みヘッダー ファイルの作成)
/Yd (オブジェクト ファイルへのデバッグ情報の配置)	/Yu (プリコンパイル済みヘッダー ファイルの使用)	/YX (プリコンパイル済みヘッダーを自動的に使用)
/Z7、/Zd、/Zi、/ZI (デバッグ情報の形式)	/Za、/Ze (言語拡張の無効化)	/Zc (準拠)

/Zg (関数プロトタイプの生成)	/Zl (既定のライブラリ名の省略)	/Zm (プリコンパイル済みヘッダーのメモリ割り当て制限の指定)
/Zp (構造体メンバの配置)	/Zs (構文チェックのみ)	..

関連項目

参照情報

ARM コンパイラ オプション

Renesas コンパイラ オプション

MIPS コンパイラ オプション

デバイス コンパイラの組み込み関数

デバイス コンパイラはさまざまな組み込み関数をサポートします。組み込み関数の中には、すべてのデバイス コンパイラでサポートされているものがあります。また、マイクロプロセッサ ファミリごとに、その他の組み込み関数がサポートされます。

共通の組み込み関数を使用すると、C や C++ では簡潔に表現するのが難しい操作を簡単で便利に実行できます。これらの関数が共通と呼ばれるのは、すべてのデバイス コンパイラでサポートされるためです。

組み込み関数を使用して、インライン アセンブリではなくアセンブリ命令にアクセスしても、コンパイラによって完全に最適化されたコードを生成できます。すべての組み込み関数は永続的です。これらを **#pragma** 関数で使用すると、コンパイル時にエラー メッセージが出力されます。

これらの関数を有効にするには、Cmnintrin.h を含めます。

このセクションのトピック

デバイス コンパイラの共通組み込み関数

デバイス コンパイラでサポートされている組み込み関数の参照情報を提供します。

CRT 関数の組み込み形式

デバイス コンパイラによって組み込み関数としてサポートされている CRT 関数の参照情報を提供します。

サポートされていない組み込み関数

デスクトップ コンパイラでサポートされるが、デバイス コンパイラではサポートされない組み込み関数の一覧を記載しています。

共通組み込み関数のマクロ

プロセッサ固有の要件と共に、マクロの参照情報の詳細を提供します。

関連セクション

ARM マイクロプロセッサの組み込み関数

特に ARM マイクロプロセッサ コンパイラに定義されている組み込み関数の参照資料を提供します。

MIPS マイクロプロセッサの組み込み関数

特に MIPS マイクロプロセッサ コンパイラに定義されている組み込み関数の参照資料を提供します。

Renesas マイクロプロセッサの組み込み関数

特に SH マイクロプロセッサ コンパイラに定義されている組み込み関数の参照資料を提供します。

デバイス コンパイラの共通組み込み関数

次の表に、すべてのデバイス コンパイラでサポートされる組み込み関数を示します。

__assume	_debugbreak	__noop	_CacheRelease
_CacheWriteBack	_CopyFloatFromInt32	_CopyInt32FromFloat	_CopyInt64FromDouble
_CountLeadingOnes、_CountLeadingOnes64	_CountLeadingSigns、_CountLeadingSigns64	_CountLeadingZeros、_CountLeadingZeros64	_CountOneBits、_CountOneBits64
_ICacheRefresh	_isunordered、_isunorderedf	_MulHigh、_MulUnsignedHigh	_prefetch
_ReadWriteBarrier、_WriteBarrier	_ReturnAddress	__trap	MulDiv

関連項目

参照情報

サポートされない組み込み関数

CRT **関数の組み込み形式**

_debugbreak

この関数では、デバッグ ブレークポイント例外が挿入されます。

```
void __cdecl __debugbreak(void);
```

パラメータ
なし。

戻り値

なし。

備考

ブレークポイントは、例外ハンドラにコントロールを移します。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_debugbreak	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目
その他のリソース
デバイス コンパイラの組み込み関数

_prefetch

この関数は、メイン メモリからデータ キャッシュを読み込みます (可能な場合)。

```
void __cdecl __prefetch(
  void *
);
```

パラメータ

*

[in] **キャッシュ行へのポインタ。**

戻り値

なし。

備考

この関数は、要求された機能がターゲットのハードウェア プラットフォームで使用できない場合は何も実行しません。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_prefetch	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

__trap

この関数はトラップ命令を挿入します。

```
int __cdecl __trap(
  arg1,
);
```

パラメータ

arg1

[in] トラップ番号。

戻り値

トラップ ハンドラによって提供される整数値。トラップ ハンドラから戻り値が提供されない場合、戻り値は未定義です。

備考

コンパイラのバックエンドで、トラップ番号などの引数に制限を課すことができます。これによって、トラップに特殊な呼び出し規則を定義することができます。

トラップ番号の解釈とトラップ ハンドラによって実行されるアクションは定義されません。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
__trap	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_CacheRelease

この関数は、メイン メモリへのポインタが参照するアドレスを記述するキャッシュ行を書き込み、このキャッシュ行を空としてマークします。

```
void __cdecl __CacheRelease(
  void *
);
```

パラメータ

*

[in] キャッシュ行へのポインタ。

戻り値

なし。

備考

この関数は、要求された機能がターゲットのハードウェア プラットフォームで使用できない場合は何も実行しません。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_CacheRelease	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_CacheWriteBack

この関数は、ポインタが参照するアドレスが格納されるキャッシュ行をメイン メモリへ書き込みます。

```
void __cdecl __CacheWriteback(
  void *
);
```

パラメータ

*

[in] **キャッシュ行へのポインタ。**

戻り値

なし。

備考

この関数は、要求された機能がターゲットのハードウェア プラットフォームで使用できない場合は何も実行しません。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_CacheWriteBack	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_ICacheRefresh

この関数は、命令キャッシュからポインタで参照されるアドレスを含むキャッシュ行を解放します。

```
void __cdecl __ICacheRefresh(
  void *
);
```

パラメータ

*

[in] キャッシュ行へのポインタ。

戻り値

なし。

備考

リフレッシュされる命令キャッシュの大きさは実装に依存します。通常は 16 バイト以上ですが、命令キャッシュ全体の場合もあります。また、ポインタで参照する命令のプリフェッチも実装に依存します。

この関数は、SH および MIPS マイクロプロセッサではサポートされません。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_ICacheRefresh	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_ReadWriteBarrier、_WriteBarrier

_ReadWriteBarrier は、次に続くメモリ アクセスが開始される前に、前のすべてのメモリ アクセスを強制的に完了させます。

_WriteBarrier は、次に続く書き込み処理が開始される前に、前のすべてのメモリの書き込み処理を強制的に完了させます。

```
void __cdecl _ReadWriteBarrier(void);
void __cdecl _WriteBarrier(void);
```

パラメータ
なし。

戻り値

なし。

備考

_WriteBarrier は通常、デバイス ドライバの書き込みに使用し、さらにコマンドが発行される前に、一連のコマンドがデバイスに送られるようにします。コンパイラは、_WriteBarrier の呼び出し中は、明示的な同期化命令がない場合、ハードウェア プラットフォーム上であってもメモリの書き込みの再スケジュールを行いません。

_ReadWriteBarrier は通常、デバイス ドライバの書き込みに使用し、ステータスが読み取られる前に、一連のコマンドがデバイスに送られるようにします。

コンパイラは、_ReadWriteBarrier の呼び出し中は、明示的な同期化命令がない場合、ハードウェア プラットフォーム上であってもメモリの読み取りおよび書き込みの再スケジュールを行いません。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_WriteBarrier	<cmnintrin.h>	x86、ARM、SH-4、MIPS
_ReadWriteBarrier	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目
その他のリソース
デバイス コンパイラの組み込み関数

_ReturnAddress

この関数は、コントロールが呼び出し元に戻った後に実行される呼び出し元関数の命令のリターン アドレスを提供します。

```
void _ReturnAddress(void);
```

パラメータ
なし。

戻り値

なし。

備考

「例」で示しているプログラムを作成し、作成したプログラムをデバッガで実行します。

プログラムを実行する際は、**_ReturnAddress** が返すアドレスに留意してください。

_ReturnAddress が使用された関数から戻った直後に [逆アセンブリ] ウィンドウを開くと、実行される次の命令のアドレスと **_ReturnAddress** が返したアドレスが一致していることがわかります。

インライン化などの最適化がリターン アドレスに影響を与えることがあります。

たとえば、**/Ob (インライン関数の展開)**、n=1 を指定して次のプログラム例をコンパイルすると、inline_func は呼び出し元関数 main にインライン化されます。このため、inline_func および main からの **_ReturnAddress** への呼び出しでは、共に同じ値が生成されます。

例

```
// compiler_intrinsics__ReturnAddress.cpp
#include <stdio.h>
// _ReturnAddress should be prototyped before use
#ifdef __cplusplus
extern "C"
#endif
void * _ReturnAddress(void);

#pragma intrinsic(_ReturnAddress)

__declspec(noinline)
void noinline_func(void)
{
   printf("Return address from %s: %p\n", __FUNCTION__, _ReturnAddress());
}

__forceinline
void inline_func(void)
{
   printf("Return address from %s: %p\n", __FUNCTION__, _ReturnAddress());
}

int main(void)
{
   noinline_func();
   inline_func();
   printf("Return address from %s: %p\n", __FUNCTION__, _ReturnAddress());

   return 0;
}
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_ReturnAddress	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_CopyDoubleFromInt64

この関数は浮動小数点倍精度長整数を長整数レジスタにコピーします。

```
double _CopyInt64FromDouble(
  __int64 arg1
);
```

パラメータ

arg1

[in] **関数で操作する長整数の引数。**

戻り値

arg1 **を変換した浮動小数点倍精度の結果。**

備考

この関数は、一時メモリロケーションを使用してソース値をコピーすることによって実装できます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_CopyDoubleFromInt64	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_CopyFloatFromInt32

この関数は整数値を浮動小数点数レジスタにコピーします。

```
float _CopyFloatFromInt32(
  ___int32 arg1
);
```

パラメータ

arg1

[in] **関数で操作する整数値。**

戻り値

arg1 **の浮動小数点数変換。**

備考

この関数は、一時メモリロケーションを使用してソース値をコピーすることによって実装できます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_CopyFloatFromInt32	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_CopyInt32FromFloat

この関数は、浮動小数点数を整数レジスタにコピーします。

```
__int32 _CopyInt32FromFloat(
  float arg1
);
```

パラメータ

arg1

[in] **関数で操作する浮動小数点値。**

戻り値

arg1 **の整数変換。**

備考

この関数は、一時メモリロケーションを使用してソース値をコピーすることによって実装できます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_CopyInt32FromFloat	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_CopyInt64FromDouble

この関数は、浮動小数点倍精度を長整数レジスタにコピーします。

```
__int64 _CopyInt64FromDouble(
  double arg1
);
```

パラメータ

arg1

[in] **関数で操作する浮動小数点倍精度引数。**

戻り値

変換結果の長整数。

備考

この関数は、一時メモリロケーションを使用してソース値をコピーすることによって実装できます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_CopyInt64FromDouble	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_CountLeadingOnes、_CountLeadingOnes64

この関数は最上位ビットで始まる連続する 1 ビットの数を戻します。

```
unsigned __cdecl _CountLeadingOnes(
  long arg1
);

unsigned __cdecl _CountLeadingOnes64(
  __int64 arg1
);
```

パラメータ

arg1

[in] **関数で 1 ビットを検出する長整数の引数。**

戻り値

arg1 **の中の連続する 1 ビットの数。***arg1* **のすべてのビットが設定されている場合は、_CountLeadingOnes の戻り値が** 32 **になるか、または _CountLeadingOnes64 の戻り値が** 64 **になります。**

備考

この関数はランタイム関数を呼び出すことによって実装できます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_CountLeadingOnes	<cmnintrin.h>	x86、ARM、SH-4、MIPS
_CountLeadingOnes64	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_CountLeadingSigns、_CountLeadingSigns64

この関数は、符号ビットに一致する、上位 2 番目のビットで始まる連続したビットの数を戻します。

```
unsigned __cdecl _CountLeadingSigns(
  long arg1
);

unsigned __cdecl _CountLeadingSigns64(
  __int64 arg1
);
```

パラメータ

arg1

[in] 関数で操作する符号なし整数値。

戻り値

arg1 の中の、符号ビットに一致する連続したビットの数。

備考

この関数はランタイム関数を呼び出すことによって実装できます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_CountLeadingSigns	<cmnintrin.h>	x86、ARM、SH-4、MIPS
_CountLeadingSigns64	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_CountLeadingZeros、_CountLeadingZeros64

この関数は、引数の最上位ビットで始まる連続する 0 ビットの数を返します。

```
unsigned __cdecl _CountLeadingZeros(
  long arg1
);

unsigned __cdecl _CountLeadingZeros64(
  __int64 arg1
);
```

パラメータ

arg1

[in] **関数によって検証される符号なし整数。**

戻り値

arg1 **内の連続する** 0 **ビットの数。***arg1* **のビットがいずれも設定されていない場合は、_CountLeadingZeros の戻り値が** 32 **になるか、または _CountLeadingZeros64 の戻り値が** 64 **になります。**

備考

この関数はランタイム関数を呼び出すことによって実装できます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_CountLeadingZeros	<cmnintrin.h>	x86、ARM、SH-4、MIPS
_CountLeadingZeros	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_CountOneBits、_CountOneBits64

この関数は、引数の 1 ビットの数を返します。

```
unsigned __cdecl _CountOneBits(
  long arg1
);

unsigned __cdecl _CountOneBits64(
  __int64 arg1
);
```

パラメータ

arg1

[in] **関数が 1 ビットを検索する長整数の値。**

戻り値

1 **ビットの数。**

備考

この関数はランタイム関数を呼び出すことによって実装できます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_CountOneBits	<cmnintrin.h>	x86、ARM、SH-4、MIPS
_CountOneBits64	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_isunordered、_isunorderedf

_isunordered は、2 つの倍精度数値を比較し、順序付けされていないかどうかを判断します。

_isunorderedf は、2 つの浮動小数点数値を比較し、順序付けされていないかどうかを判断します。

```
int __cdecl _isunordered(
  double arg1,
  double arg2
);

int __cdecl _isunorderedf(
  float arg1,
  float arg2
);
```

パラメータ

arg1

[in] *arg2* と比較する値。

arg2

[in] *arg1* と比較する値。

戻り値

ブール値を返します。

TRUE は、*arg1* と *arg2* が順序付けされていないことを示します。

備考

IEEE-754 浮動小数点比較には、次の 4 種類の結果値があります。より小さい、等しい、より大きい、順序なしです。

最初の 3 つの条件は通常の C 演算子を使用して評価でき、この関数は、最後の条件の評価に使用します。

2 つの値のいずれかが NaN の場合、それらの値は順序付けされません。NaN はそれ自体の値も含めてどのような値とも等しくなりません。

C++ コンパイラは、*int* ではなく *bool* 値を返します。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_isunordered	<cmnintrin.h>	x86、ARM、SH-4、MIPS
_isunorderedf	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

MulDiv

MulDiv 関数は、2 つの 32 ビット値を乗算した後、64 ビットのその結果を 3 つ目の 32 ビット値で割ります。戻り値は最も近い整数に切り上げまたは切り捨てられます。

```
int MulDiv(
  int nNumber,
  int nNumerator,
  int nDenominator
);
```

パラメータ

nNumber

[in] **被乗数。**

nNumerator

[in] **乗数。**

nDenominator

[in] **乗算** (nNumber * nNumerator) **の結果を割る数。**

戻り値

関数が正常に終了した場合、戻り値は乗算と除算の結果になります。オーバーフローが発生した場合、または nDenominator **の値が** 0 **だった場合は、**–1 **が戻されます。**

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
MulDiv	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

_MulHigh、_MulUnsignedHigh

この関数は 2 **つの引数を乗算したことによる高位** 32 **ビットの結果を戻します。**

```
long __cdecl _MulHigh(
  long arg1,
  long arg2
);

unsigned long __cdecl _MulUnsignedHigh(
  unsigned long arg1,
  unsigned long arg2
);
```

パラメータ

arg1

[in] **積の第** 1 **引数。**

arg2

[in] **積の第** 2 **引数。**

戻り値

arg1 **と** *arg2* **の乗算結果である長整数。**

備考

この関数はオーバーフローの検出に役立ちます。_MulHigh は [-0.5..0.5) **を表すよう調整された整数の乗算に役立ち、_MulUnsignedHigh は** 0..1) **を表すよう調整された整数の乗算に役立ちます。**

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_MulHigh	<cmnintrin.h>	x86、ARM、SH-4、MIPS
_MulUnsignedHigh	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

__assume

__assume 組み込み関数は、オプティマイザにヒントを渡します。

```
__assume(expression)
```

パラメータ
expression

テストする条件。

戻り値

なし。

備考

オプティマイザでは、キーワードが表示された時点では式で表される条件は真であり、変数への割り当てなどによって式が変更されるまで真のままであることが前提となっています。__assume によってオプティマイザに渡されるヒントを選択して使用することで、最適化が向上します。

例

```
//
// A common use of __assume tests the default case of a switch statement.
//
#ifdef DEGUG
# define ASSERT(e)   ( ((e) || assert(__FILE__, __LINE__) )
#else
# define ASSERT(e)   ( __assume(e) )
#endif

void gloo(int p)
{
  switch(p){
    case 1:
      blah(1);
      break;
    case 2:
      blah(-1);
      break;
    default:
         __assume(0);
        // This tells the optimizer that the default
        // cannot be reached. Hence, no extra code
            // is generated to check that 'p' has a value
            // not represented by a case arm.  This makes the switch
            // run faster.
  }
}
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
__assume	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目
その他のリソース
デバイス コンパイラの組み込み関数

__noop

__noop を使用して、関数を無視し、引数リストを評価しない場合に使用する関数名を指定します。

```
__noop(functionname)
```

パラメータ

Functionname

無視する関数の名前。

戻り値

なし。

例

```
    // The following code shows how you could use __noop
    // compile with or without /DDEBUG
  #include <stdio.h>

  #if DEBUG
    #define PRINT  printf
  #else
    #define PRINT  __noop
  #endif

  void main() {
  PRINT("\nhello\n");
  }
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
__noop	<cmnintrin.h>	x86、ARM、SH-4、MIPS

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

CRT 関数の組み込み形式

次の表は、デバイス コンパイラでサポートされる組み込み形式の CRT **関数の一覧です。**

abs、_abs64	_alloca
_byteswap_uint64、_byteswap_ulong、_byteswap_ushort	labs
_lrotl、_lrotr	memcmp、wmemcmp
memcpy、wmemcpy	memset、wmemset
_rotl、_rotl64、_rotr_rotr64	strcat、wcscat、_mbscat
strcmp、wcscmp、_mbscmp	strcpy、wcscpy、_mbscpy
strlen、strlen_l、wcslen、wcslen_l、_mbslen、_mbslen_l、_mbstrlen、_mbstrlen_l	_strset、_wcsset、_mbsset、_mbsset_l

数学ライブラリ関数の組み込み形式

次の表は、デバイス コンパイラでサポートされる組み込み形式の数学ライブラリ関数の一覧です。

acos	asin	atan	atan2
ceil、ceilf	cos、cosf、cosh、coshf	fmod、fmodf	exp、log、**および** log10
floor、floorf	pow	sin、sinf、sinh、sinhf	sqrt
tan、tanf、tanh、tanhf	.	.	.

関連項目

参照情報

サポートされない組み込み関数

デバイス コンパイラの共通組み込み関数

サポートされていない組み込み関数

x86、AMD64、または Intel Itanium (IPF) アーキテクチャ向けの Visual Studio デスクトップ コンパイラでは、次の一覧の関数がサポートされています。デバイス コンパイラでは、これらの組み込み関数はサポートされていません。

__mul128	__shiftleft128
__shiftright128	_InterlockedCompare64Exchange
_InterlockedExchangePointer	_umul128
_umulh	__cpuid
__getcx86、AMD64、IA64、IPFerseflags	__int2c
_BitScanForward、_BitScanForward64	_BitScanReverse、_BitScanReverse64
_bittest、_bittest64	_bittestandcomplement、_bittestandcomplement64
_bittestandreset、_bittestandreset64	_bittestandset、_bittestandset64
_InterlockedAddLargeStatistic	_InterlockedAnd
_InterlockedAnd、_InterlockedAnd64	_interlockedbittestandreset、_interlockedbittestandreset64
_InterlockedCompareExchange16	_InterlockedCompareExchange64
_InterlockedDecrement16	_InterlockedIncrement16
_InterlockedOr	_InterlockedXor
_ReadBarrier	.
__faststorefence	__fc
__mulh	__writegsbyte、__writegsdword、__writegsqword、__writegsword
__segmentlimit	__load128、__load128_acq
__break	__dsrlz
__fci	__fclrf
__flushrs	__fsetc
__fwb	__getCFS
__getPSP	__getReg
__invalat	__isNat

__isrlz	__lfetch、__lfetch_excl、__lfetchfault、__lfetchfaultexcl
__ptcg、__ptcga	__ptcl
__ptrcl、__ptri	__rdteb
__rdtsc	__rsm
__rum	__setReg
__ssm	__store128、__store128_rel
__sum	__synci
__yield	_AcquireSpinLock
_InterlockedAdd	_ReleaseSpinLock
_thash	_ttag
__writefsbyte、__writefsdword、__writefsqword、__writefsword	_wbinvd
__inbyte	__inbytestring
__indword	__indwordstring
__invlpg	__Inword、__inwordstring
__movsb、__movsd、__movsq、__movsw	__outbyte
__outbytestring	__outword
__outwordstring	__readcr0、__readcr2、__readcr3、__readcr4、__readcr8
__readfsbyte、__readfsdword、__readfsqword、__readfsword	__readgsbyte、__readgsdword、__readgsqword、__readgsword
__readmsr	__readpmc
__stosb、__stosd、__stosq、__stosw	__writemsr
_AddressOfReturnAddress	_writecr0、_writecr2、_writecr3、_writecr4、_writecr8

次の表は、デバイス コンパイラでサポートされていない組み込み形式の数学ライブラリ関数の一覧です。

acosf	acosl	asinf
asinl	atanf	atanl
atan2f	atan2l	ceilf
ceill	coshf	coshl
cosf	cosl	expf

expl	floorf	floorl
fmodf	fmodl	logf
logl	log10f	log10l
powf	powl	sinf
sinl	sinhf	sinhl
sqrtf	sqrtl	tanf
tanl	tanhf	tanhl

関連項目

その他のリソース

デバイス コンパイラの組み込み関数

共通組み込み関数のマクロ

共通組み込み関数をサポートするすべてのマイクロプロセッサ ファミリでは、ヘッダー ファイル cmnintr.h に、サポートする記述がコンピュータごとに示されています。このコンピュータごとの記述によって実行される処理は次のとおりです。

- 正しいプロトタイプによる関数の宣言
- 組み込みバージョンの有効化
- 関数のサポート方法を分類するマクロの定義

組み込み関数を無条件で使用することも、いずれかのマクロを使用して各自の CPU 環境で組み込み関数のサポートを分類することもできます。

次の表は、共通組み込み関数のマクロに関する説明です。

マクロ	説明
_INTRINSIC_IS_HELPER	このマクロは、C ランタイム ライブラリ (CRT) への呼び出しによって組み込み関数がインスタンス化されるかどうかを決定します。
_INTRINSIC_IS_INLINE	このマクロは、指定した組み込み関数をコンパイラで 1 つ以上の行に展開できるかどうかを調べます。
_INTRINSIC_IS_SAFE	このマクロは、指定された組み込み関数が OS に依存せずにインスタンス化されるかどうかを決定します。
_INTRINSIC_IS_SUPPORTED	このマクロは、指定した組込み関数がサポートされているかどうかを判断します。

関連項目

参照情報

デバイス コンパイラの共通組み込み関数

_INTRINSIC_IS_HELPER

このマクロは、指定した組込み関数がサポートされているかどうかを判断します。

```
_INTRINSIC_IS_SUPPORTED(arg)
```

パラメータ
Arg

対象の組込み関数の名前。

戻り値

0 以外の戻り値は、指定した組み込み関数がコンパイラでサポートされていることを示します。

関連項目
その他のリソース
デバイス コンパイラの組み込み関数

_INTRINSIC_IS_INLINE

このマクロは、指定した組込み関数がサポートされているかどうかを判断します。

```
_INTRINSIC_IS_SUPPORTED(arg)
```

パラメータ
Arg

[in] **対象の組込み関数の名前。**

戻り値

0 **以外の戻り値は、指定した組み込み関数がコンパイラでサポートされていることを示します。**

例

```
   //  The followig example shows how to use the _INTRINSIC_IS_INLINE macro to determine if
the _rotl intrinsic will be expanded inline.
   //
#include <cmnintrin.h>
#if _INTRINSIC_IS_INLINE(_rotl)
     x = _rotl(y, 3);
#else
     x = MY_ROTL(y, 3);   // call my inline implementation, not the CRT helper
#endif
```

関連項目
その他のリソース
デバイス コンパイラの組み込み関数

_INTRINSIC_IS_SAFE

このマクロは、指定された組み込み関数が OS に依存せずにインスタンス化されるかどうかを決定します。

```
_INTRINSIC_IS_SAFE(arg)
```

パラメータ
Arg

[in] 対象の組込み関数の名前。

戻り値

0 以外の戻り値は、指定した組み込み関数で OS 機能の起動を必要とすることを示します。

備考

組み込み関数の展開では、コンパイラによる OS 機能の呼び出しが必要になることがあります。これは、コンパイラ自体ではその特定のタスクを実行できないためです。コンパイラは、必要な機能を OS が提供するかどうかを制御できないため、このような組み込み関数は確実ではないと見なされます。

たとえば、__trap 組み込み関数では、OS ハンドラによる対応が可能であることが前提となっています。コンパイラはこのようなハンドラの存在を保証できないため、**__trap** 組み込み関数は確実ではないと見なされます。

関連項目
その他のリソース
デバイス コンパイラの組み込み関数

_INTRINSIC_IS_SUPPORTED

このマクロは、組み込み関数が Microsoft C ランタイム ライブラリ (CRT) **の呼び出しによってインスタンス化されるかどうかを判断します。**

```
_INTRINSIC_IS_HELPER(arg)
```

パラメータ
Arg

[in] **対象の組込み関数の名前。**

戻り値

0 **以外の戻り値は、***arg* **で指定された組み込み関数が** CRT **の呼び出しによってインスタンス化されることを示します。**

例

```
    //
    // The following example shows how to use the
    // _INTRINSIC_IS_SUPPORTED macro to determine
    // support for the __trap intrinsic function.
    //
#include <cmnintrin.h>
#if INTRINSIC_IS_SUPPORTED(__trap)
  __trap(1);  // __trap IS SUPPORTED
#else
   //  __trap IS NOT SUPPORTED
#endif
  //
```

関連項目
参照情報
デバイス コンパイラの共通組み込み関数

定義済みマクロ

デバイス コンパイラでは 7 つの定義済み ANSI C マクロが認識されます。その他に、Microsoft C++ 実装によっていくつかのマクロが提供されます。

これらのマクロは引数を取らず、また再定義できません。**__LINE__** と **__FILE__** を除いて、これらのマクロの値はコンパイル中一定でなければなりません。

次の表に、定義済みマクロに関する追加情報を示します。これらのマクロの一部では、複数の値が定義されます。

マクロ	説明
__DATE__	現在のソース ファイルのコンパイル日。 この日付は Mmm dd yyyy の形式の文字列リテラルです。
__FILE__	現在のソース ファイルの名前。**__FILE__** は二重引用符で囲まれた文字列に展開されます。
__FUNCTION__	個々の関数内のみで有効であり、その関数に含まれる非装飾名 (文字列) を戻します。 /EP (#line ディレクティブなしの stdout への前処理) または /P (ファイルへの前処理) コンパイラ オプションを使用する場合、**__FUNCTION__** は展開されません。
__LINE__	現在のソース ファイル内の行番号。 この行番号は 10 進整定数です。これは **#line** ディレクティブを使用して変更することができます。
__STDC__	ANSI C 標準に完全に準拠していることを示します。 /Za、/Ze (言語拡張機能の無効化) コンパイラ オプションを指定していて、C++ コードはコンパイルしない場合にのみ、整定数 1 として定義されます。それ以外の場合は定義されません。
__TIME__	現在のソース ファイルの最新のコンパイル時刻。 この時刻は hh:mm:ss の形式の文字列リテラルです。
__TIMESTAMP__	現在のソース ファイルの最新の変更日時。Ddd Mmm Date hh:mm:ss yyyy (Ddd は曜日の省略形、Date は 1 ～ 31 の整数) の形式の文字列リテラルとして表されます。

Microsoft 固有のマクロ

次の表に、Microsoft 固有のその他の定義済みマクロを示します。

_CHAR_UNSIGNED

既定の char 型は符号なしです。

/J (既定の char 型は符号なし) の指定時に定義されます。

__cplusplus

C++ プログラムに対してのみ定義されます。

_CPPRTTI

/GR (ランタイム型情報の有効化) を指定してコンパイルしたコードにのみ定義されます。

_CPPUNWIND

/GX (例外処理の有効化) を指定してコンパイルしたコードにのみ定義されます。

_MFC_VER

MFC のバージョンを定義します。

Microsoft Foundation Class Library 6.0 以降に対して 0x0600 として定義されます。

常に定義されます。

_MSC_EXTENSIONS

このマクロは、/Za、/Ze (言語拡張機能の無効化) コンパイラ オプション (既定) を指定したコンパイル時に定義されます。

定義された場合、その値は 1 です。

_MSC_VER

コンパイラのバージョンを定義します。

Microsoft Visual C++ 6.0 以降に対して 1200 として定義されます。

常に定義されます。

_WIN32

Win32 用アプリケーションに対して定義されます。

常に定義されます。

関連項目

その他のリソース

デスクトップ コンパイラとデバイス コンパイラの違い

デバイス コンパイラのデータ配置に関する問題

データの配置は、特に x86 以外のターゲットのアーキテクチャにコードを移植したり、書き込んだりする場合に重要な問題になります。

アーキテクチャによっては、基本データ型を使用した整列されていない演算を行うと、アプリケーションにパフォーマンスの低下、エラー、異常終了などが起こる場合があります。

ここでは、このような問題を回避するために役立つ情報を提供します。

このセクションのトピック

データ配置について

データ配置に関する問題がデバイスのプログラミングにどのような影響を及ぼすかについて説明します。

配置エラーの回避

基本型の正しい配置を保持するためのガイドラインを提供します。

構造体のパッキングに関連する処理

構造体のパッキングと配置との相互作用について説明します。

__unaligned キーワード

__unaligned 修飾子を使用した配置に関する問題の処理についての参照情報を提供します。

データ配置について

Alpha、IA-64、MIPS、および SuperH などのアーキテクチャに基づく CPU の多くでは、整列されていないデータの読み取りが拒否されます。プログラムがこれらの CPU のいずれかに対して整列されていないデータへのアクセスを要求すると、CPU は例外状態になり、ソフトウェアに対して続行できないことを通知します。たとえば、ARM、MIPS および SH デバイス プラットフォーム上では、整列されていないデータへのアクセスが要求されると、既定でオペレーティング システムがアプリケーションに例外通知を出力します。

整列されていないメモリへのアクセスは、ハードウェアでこのようなアクセスがサポートされていない場合に莫大なパフォーマンスの損失を招く可能性があります。

配置

配置は、メモリ アドレスのプロパティの 1 つで、数値アドレス モジュロ 2 の累乗で表します。たとえば、アドレス 0x0001103F モジュロ 4 は 3 で、このアドレスは 4n+3 に整列されていることになります。ここで 4 は選択した 2 の累乗を示します。アドレスの配置は、選択した 2 の累乗に依存します。同じアドレス モジュロ 8 は 7 です。

配置が *Xn+0* の場合、アドレスは X に整列されていることになります。

CPU はメモリに保存されているデータを操作する命令を実行し、データはメモリ内におけるアドレスによって識別されます。各データにはアドレスに加え、サイズも設定されています。アドレスがサイズに合わせて整列されている場合、各データは自然に整列されて呼び出されますが、そうでない場合は正しく整列されません。たとえば、データの識別に使用されるアドレスが 8 に整列されている場合、8 バイトの浮動小数点データは自然に整列されます。

データ配置のコンパイラによる処理

デバイス コンパイラでは、データの不適切な配置を防ぐ方法でデータの割り当てが試行されます。

単純なデータ型の場合、コンパイラはデータ型のバイト サイズの倍数のアドレスを割り当てます。コンパイラはこのようにして、**long** 型の変数に 4 の倍数のアドレスを割り当て、アドレスの最下位の 2 ビットをゼロに設定します。

また、コンパイラは、構造の各要素が自然に整列されるように構造を埋め込みます。次に、構造 **struct x_** のコード例を示します。

```
struct x_
{
   char a;     // 1 byte
   int b;      // 4 bytes
   short c;    // 2 bytes
   char d;     // 1 byte
} MyStruct;
```

コンパイラは、自然に整列されるようにこの構造を埋め込みます。

例

次のコード例で、コンパイラが埋め込んだ構造をメモリに配置するしくみを示します。

```
// Shows the actual memory layout
struct x_
{
   char a;             // 1 byte
   char _pad0[3];      // padding to put 'b' on 4-byte boundary
   int b;             // 4 bytes
   short c;           // 2 bytes
   char d;            // 1 byte
   char _pad1[1];     // padding to make sizeof(x_) multiple of 4
}
```

いずれの宣言でも **sizeof(struct x_)** は 12 バイトとして返されます。

2 番目の宣言には次の 2 つの埋め込み要素が含まれています。

- **char _pad0[3]** は 4 バイト境界上に **int** b メンバを配置します。
- **char _pad1[1]** は、構造 **struct _x bar[3]** の配列要素を整列します。

埋め込みによって、自然なアクセスが可能になるように **bar[3]** の要素が整列されます。

次のコード例は、**bar[3]** 配列のレイアウトです。

```
adr
offset   element
------   -------
0x0000   char a;         // bar[0]
0x0001   char pad0[3];
0x0004   int b;
0x0008   short c;
0x000a   char d;
0x000b   char _pad1[1];

0x000c   char a;         // bar[1]
0x000d   char _pad0[3];
0x0010   int b;
0x0014   short c;
0x0016   char d;
0x0017   char _pad1[1];

0x0018   char a;         // bar[2]
0x0019   char _pad0[3];
0x001c   int b;
0x0020   short c;
0x0022   char d;
0x0023   char _pad1[1];
```

関連項目

参照情報

__unaligned キーワード

概念

構造体のパッキングに関連する処理

配置エラーの回避

すべてのデータ型は基本型あるいは、配列、構造体、または単純な型の集合体で構成される複合型です。

基本型の配置

通常、配置エラーは次の規則を守ることによって回避できます。

- 構造体のパッキングを有効にしません。
- 大規模な配置の再キャスト ポインタを使用して、小規模に配置されたアドレスにアクセスしません。

たとえば、**char** データ型のアドレスを **long** へのポインタとして使用すると、4 の倍数ではないアドレスからの 4 バイト移行の実行を意味する場合があるため、配置エラーが発生する可能性があります。

次の例は、この方法によるコーディングを示しています。

```
char a[10];
char *p = &a[1];
long l = *(long *)p;   // ERROR!; Attempt to move a long from
                       // the address of a char.
```

大規模に配置されたアドレスへのアクセス

小規模の配置の再キャスト ポインタを使用して、大規模に配置されたアドレスにアクセスすることは問題ありません。たとえば、**char *** キャストを使用して、long 型変数の先頭バイトや任意のバイトにアクセスできます。

構造体のパックまたは未整列アドレスへのデータ移行が必要な場合は、__unaligned キーワードを使用します。

このキーワードでは、継承されたコードのクラスや MFC (Microsoft Foundation Class) ライブラリのクラスなど、既存クラスの配置問題は解決できません。

構造体の大規模な配列を使用するプログラムでは、構造体のパッキングの使用、または既存のデータ形式の読み取りが必要な場合があります。

このような場合でも、プログラムのデータを使用する前に、パックされた構造体のメンバを慎重にアンパックすれば、パックされた構造体を引き続き使用できます。この技法では、メンバ間、要素間、またはフィールド間で、正しく配置されている一時ロケーションに構造体のデータをコピーする必要がある場合があります。

構文的には、**__unaligned** は **const** や **volatile** のような型修飾子であり、ポインタが指す内容を制御します。**__unaligned** が機能するのは、ポインタ宣言で使用する場合のみです。

次のコード例は、**__unaligned** ポインタ宣言の使用方法を示しています。

```
int __unaligned *p1;             // p1 is a pointer to unaligned int
struct {int i;} __unaligned *p2; // p2 is a pointer to unaligned struct
```

次のコード例は、整数演算と組み合わせた **__unaligned** と **#pragma pack** の正しい使用例と不適切な使用例を示しています。最初のセクションでは、**__unaligned** 修飾子が使用されていないためにエラーが発生しています。一方、2 番目のセクションでは、**__unaligned** 修飾子が正しく使用されています。

```
#pragma pack (1)
struct s {
   char c;            // offset 0
   int i;             // offset 1!
} ss;
#pragma pack ()

void f_improper(int *p)
{
   *p = 23;        // generates a fault
}
```

```
void g_proper(int __unaligned *p)   // OK
{
   *p = 42;
}

void main ()
{
   f_improper(&ss.i);
   g_proper(&ss.i);
}
```

この例の出力は、次のマシン コード例に表示されます。出力では、関数 **f_improper** は未整列データの不適切な処理によって生成されたコードを示し、関数 **g_proper** は **__unaligned** の使用時に生成された追加のコードを示しています。

関数 **g_proper** では、未整列データを処理するために倍以上の数の命令が生成されていますが、配置エラーは発生していません。

```
f_improper::
   mov r3, #0x17
   str r0, [r0]      // This instruction gets an alignment fault.
   mov pc, lr

g_proper::
   mov r3, #0x2A
   strb r3, [r0]     // Four individual bytes are stored,
   mov r3, #0        // avoiding an alignment fault.
   strb r3, [r0, #1]
   strb r3, [r0, #2]
   strb r3, [r0, #2]
   mov  pc, lr
```

__unaligned ポインタを介したデータ アクセスはパフォーマンス面で不利なため、**__unaligned** キーワードは必要な場合にのみ使用するようにします。

配置エラーが起きないようにするには、参照解除された、小さな一連の要素としてのデータにコンパイラからアクセスする必要があります。

データが既に整列されている場合、このデータ アクセス技法は不要です。

複合型の配置

次の一覧は、複合型の配置の規則を要約したものです。

- 配列の調整は、基本型の配置と同じです。
- 構造体またはオブジェクトの配置は、その構造体のメンバすべての配置要素の最大値です。また、構造体のパッキングがオフの場合、コンパイラはその構造体を埋め込んで、各メンバが次に利用可能な正しく配置されたアドレスにメンバを配置します。

集合体の配置もまた、すべてのメンバの配置要素の最大値です。各メンバのアドレスは集合体自体のアドレスであるため、埋め込みは行われません。

関連項目

参照情報

__unaligned キーワード

概念

構造体のパッキングに関連する処理

構造体のパッキングに関連する処理

プログラマが意図したバイト数より多くのバイトが構造体で必要になった場合、特に所要領域が重要な場合は、問題が発生することがあります。

構造体のパッキングと配置

構造体のパッキングは、コンパイラの配置動作と次のように相互作用します。

- パックサイズが既定の配置と同じかそれ以上に設定されている場合、パックサイズは無視されます。
- パックサイズが既定の配置より小さく設定されている場合、コンパイラはそのパックサイズの値に従って配置を行います。

したがって、パックサイズが 4 に設定されている場合、サイズが 4、8、または 16 バイトのデータ型は 4 の倍数のアドレスに配置されます。ただし、サイズが 8 バイト (64 ビット) のデータ型が必ずしも 8 の倍数のアドレスに配置されるわけでははありません。パックサイズは構造体の外部のデータ型には影響を及ぼしません。

また、パッキングはパッキングされた構造体全体の配置に影響します。たとえば、**#pragma pack(1)** で宣言された構造体では、パッキングなしでも自然に配置されるかどうかにかかわらず、すべてのメンバの配置が強制的に 1 に設定されます。

次の手法では、パックサイズがバイトで設定されます。

- コマンドライン オプション /Zp (構造体メンバの配置) では、パックサイズが *n* (1、2、4、8、または 16。既定は 8) に設定されます。
- コンパイラ ディレクティブ **#pragma pack([n])** では、パックサイズが n (1、2、4、8、または 16) に設定されます。n を指定しないと、パックサイズは **#pragma pack** によってコンパイル開始時の値にリセットされます。コンパイル開始時の値とは、/Zp (構造体メンバの配置) によって指定した値か、または既定値 (ほとんどのプラットフォームで 8) です。
  pragma はソース内で出現した時点からのみ適用されます。たとえば、**/Zp1** オプションではパックサイズが 1 に設定されますが、この場合、コンパイラは構造体内で埋め込みを使用しなくなります。この問題を回避するには、構造体のパッキングをオフにするか、またはポインタを使用してこのような構造体の非整列メンバにアクセスするときに __unaligned キーワードを使用します。

構造体のパッキングのガイドライン

次に、構造体の問題に関して考えられる解決策を示します。

- 構造体メンバの並べ替え
  所要領域が重要な場合は、構造体のメンバを並べ替えて同じサイズの要素を隣り合わせにし、緊密にパッキングします。通常、サイズの大きいメンバから小さいメンバの順に並べます。
  構造体の並べ替えではユーザーがそのデータ構造に対してフル コントロールを持つことが前提になりますが、ユーザーにメンバの並べ替えを実行する許可が与えられていない場合があります。たとえば、データ構造がディスク上のファイル内のフィールドのレイアウトを表すことがあります。
  次にコード例を示します。

```
struct x_
{
   char a;      // 1 byte
   int b;       // 4 bytes
   short c;     // 2 bytes
   char d;      // 1 byte
} MyStruct;
```

  この構造体のメンバを次のコード例のように並べ替えると、並べ替え後の構造体ではすべてのメンバが自然の境界に配置され、構造体のサイズは 12 バイトではなく 8 バイトになります。

```
struct x_
{
   int b;      // 4 bytes
   short c;    // 2 bytes
   char d;     // 1 byte
   char a;     // 1 byte
} MyStruct;
```

- **構造体への埋め込み**
  各配列要素の配置が同じになるように構造体のサイズに埋め込みが必要なときに、ユーザーの側で配列要素間に埋め込みが行われないようにする必要がある場合は、別の種類の問題が発生することがあります。たとえば、メモリの使用量を制限する必要がある場合や、固定形式のソースからデータを読み取る必要がある場合です。
  構造体で埋め込みが必要な場合は、コンパイラ ディレクティブ **#pragma pack** を使用することができます。ただし、**#pragma pack** を使用した場合、構造体の要素は整列されません。また、__unaligned キーワード修飾子を使用して、配置エラーを発生させずにこのデータにアクセスするためのコードを生成する必要があります。
  次のコード例では、**#pragma pack** の使用により、ポインタ px が自然に配置されないデータを指していることをコンパイラに示すと共に、読み込み、マージ、および保存操作の適切なシーケンスを生成して配置を効率よく行うようコンパイラに指示しています。

```
# pragma pack (1)
struct x_
{
   char a;    // 1 byte
   int b;     // 4 bytes
   short c;   // 2 bytes
} MyStruct;
# pragma pack ()

void bar()
{
  struct x_ __unaligned *px = &MyStruct;
   . . . .
   px->b = 5;
}
```

  __unaligned キーワードは最後の手段としてのみ使用してください。生成されたコードでは、自然に配置されたデータにアクセスする場合より効率が悪くなるためです。ただし、配置エラーを発生させるよりは、__unaligned キーワードを使用した方がいいのは確かです。
  できる限り、データ構造のメンバを並べ替えて配置を維持すると同時に、使用領域を最小限に抑えてください。

関連項目

参照情報

__unaligned キーワード

__unaligned キーワード

__unaligned キーワードはポインタ定義における型修飾子です。このキーワードでは、ポイント先のデータが正しいアドレス上に適切に配置されていない可能性があることが、コンパイラに示されます。

適切に配置するには、オブジェクトのアドレスがその型のサイズの倍数でなければなりません。たとえば、2 バイトのオブジェクトは偶数アドレスに配置する必要があります。

__unaligned で宣言されたポインタを使用してデータにアクセスする場合、配置エラーを発生させることなく、そのデータの読み込み、保存、読み取り、または書き込みを実行するために必要な追加コードがコンパイラによって生成されます。

非整列データの使用を避けることが第一ですが、場合によっては、共有ディスク構造や入出力ハードウェアなどのパッキングされた構造体へのアクセスの必要性から、非整列データの使用がやむをえないこともあります。

注: Win32 マクロの **UNALIGNED** は、**__unaligned** を必要とするハートウェア プラットフォーム上で **__unaligned** に展開されます。**UNALIGNED** は、**__unaligned** を必要としないプラットフォーム上では展開されません。

移植性を考慮して、**__unaligned** キーワードではなく **UNALIGNED** を使用してください。UNALIGNED マクロの使用時には、次の例にあるように、windef.h ヘッダーを挿入します。

```
#include <windef.h>
int UNALIGNED *p1;               // p1 is a pointer to unaligned int
struct {int i;} UNALIGNED *p2; // p2 is a pointer to unaligned struct
```

関連項目
概念
構造体のパッキングに関連する処理

デバイス コンパイラの例外処理

ARM、SH-4、MIPS などの RISC ベースのマイクロプロセッサの例外処理の機能は、x86 マイクロプロセッサの場合と異なります。多くの場合、この違いはコンパイラによって隠されますが、コードにアセンブリ言語のセグメントが含まれている場合はこの違いが重要になります。

簡単に言うと、x86 環境では、例外処理に関連付けられたデータ構造が実行時にスタックに書き込まれます。OS は、これらの構造を調べて適切な例外処理ルーチンを見つけます。

RISC 環境では、例外処理に関連する多くのデータ構造はコンパイル時に計算され、構築されるモジュールのデータ セクションに書き込まれます。RISC ベースのマイクロプロセッサは、コンテキスト、フレーム ポインタとスタック ポインタ、およびプログラム カウンタによって異なる例外処理に対処するために、PDATA 構造を使用したテーブルベースの機構をサポートします。コンパイラはアドレス限界が関数テーブルのエントリに関連付けられているスタック フレームを設定するコード セグメントを生成するため、コンパイラが処理するコードだけがこのテーブル エントリに自動的にアクセスできます。

したがって、RISC 環境では、使用するアセンブリ コード関数の例外関連処理に対応できるコードを記述する必要があります。

このセクションのトピック

x86 環境の SEH

x86 プロセッサ環境での例外処理の仕組みについて簡潔に説明します。

RISC 環境の SEH

RISC プロセッサ環境での例外処理のしくみについて簡潔に説明します。

PDATA 構造

RISC 例外処理で使用されるデータ構造の参照情報を提供します。

関連セクション

ARM プロローグおよびエピローグ

ARM マイクロプロセッサ コンパイラのプロローグおよびエピローグのコード シーケンスの作成におけるガイドラインと例を示します。

Renesas SH-4 プロローグおよびエピローグ

Renesas マイクロプロセッサ コンパイラのプロローグおよびエピローグのコード シーケンスの作成におけるガイドラインと例を示します。

MIPS プロローグおよびエピローグ

MIPS マイクロプロセッサ コンパイラのプロローグおよびエピローグのコード シーケンスの作成におけるガイドラインと例を示します。

x86 環境の SEH

簡単に言うと、x86 環境での例外処理に関連付けられたデータ構造は実行時にスタックに書き込まれます。OS はこれらの構造を調べて、適切な例外処理ルーチンを見つけ出します。

例外処理の構造

x86 環境において、**FS** レジスタはスレッド情報ブロック (TIB) 構造の現在の値をポイントします。TIB 構造の 1 つの要素は、**EXCEPTION_RECORD** 構造へのポインタで、この **EXCEPTION_RECORD** 構造には例外処理コールバック関数へのポインタが含まれています。このように、各スレッドは独自の例外コールバック関数を持っています。

x86 コンパイラは関数の処理時、スタック上に例外処理構造を構築します。**FS** レジスタは常に TIB をポイントし、TIB には **EXCEPTION_RECORD** 構造へのポインタが含まれています。この **EXCEPTION_RECORD** 構造は例外処理関数をポイントします。

EXCEPTION_RECORD 構造によって、リンクされたリストが形成されます。新しい **EXCEPTION_RECORD** 構造には直前の **EXCEPTION_RECORD** 構造へのポインタが含まれるというように形成されます。Intel ベースのコンピュータでは、常にスレッド情報ブロックの最初の DWORD である **FS:[0]** によって、リストの先頭がポイントされます。

アンワインディング

例外が発生すると、システムによって **EXCEPTION_RECORD** 構造のリストがスキャンされ、その例外のハンドラが検出されます。ハンドラが検出されると、再びリストがスキャンされ、その例外を処理するノードが検出されます。この 2 度目のスキャン時に、各ハンドラ関数がもう一度呼び出されます。その際、例外フラグが EH_UNWINDING に設定されます。

API によって、コンテキストと例外コールバック関数が格納されたダミー例外レコード構造が構築されます。各コールバック後には対応する例外フレームが削除され、API は次のフレームに進みます。API は最初のパラメータとしてアドレスが渡されたフレームに行き着くと、アンワインディングを停止します。

例外が処理され、先行するすべての例外フレームが呼び出されてアンワインドされると、処理コールバック関数が示す場所から実行が続行されます。

実行の再開位置のコードでは、スタック ポインタとフレーム ポインタ (Intel CPU では **ESP** レジスタと **EBP** レジスタ) が、例外を処理したスタック フレーム内のそれぞれの値に設定されることが必要になります。

このため、例外を受け入れるハンドラでは、スタック ポインタとフレーム ポインタを、例外を処理した SEH コードを含むスタック フレーム内のそれぞれの値に設定する必要があります。

関連項目

概念

RISC **環境の** SEH

RISC 環境の SEH

RISC 環境では、例外処理に関連するデータ構造はコンパイル時に計算され、構築されるモジュールのデータ セクションに書き込まれます。

x86 以外の Win32 環境で例外が発生したときに適切なハンドラを探すために、まず、コールスタック上にあるフレームと、コード内の関連付けられた関数が特定されます。関数にハンドラが関連付けられている場合があります。この場合、関数に関連付けられたハンドラによって例外が処理されます。

x86 と同様に、RISC システムは最後のハンドラから呼び出します。つまり、対応するフレームが最も新しくスタックにプッシュされたハンドラを最初に呼び出します。

スタック上のフレームを特定するために、各関数のコードの実行が最後の部分からシミュレーションされます。このシミュレーションによって、その関数の開始時に実際の CPU コンテキストが保持していた状態と同様の CPU コンテキストが作成されます。

スタックのアンワインドがシミュレーションされるだけで、実際には実行されないため、この逆の実行プロセスは仮想アンワインディングと呼ばれます。

アンワインドの対象となるコード要素

逆になったコードの部分は、関数のプロローグと呼ばれます。プロローグは、スタック ポインタを変更し、関数の実行が開始されるとすぐにスタック フレームを設定する命令で構成されます。

フレームが仮想的にアンワインドされると、仮想コンテキストには前のフレームのスタック ポインタと現在の関数のリターン アドレスが格納されます。リターン アドレスは、前の関数からコントロールが移動した場所のすぐ近くなので、前のフレームのプログラム カウンタに対応します。

連続する各プログラム カウンタおよびスタック ポインタによって、スタック上のフレームがなくなるまで、アンワインド プロセスが繰り返されます。

仮想的にアンワインドするために、システムでは各関数に関する多少の情報が必要です。この情報は、PDATA 構造と呼ばれるデータ構造に格納されます。

PDATA 構造によって、コード ストリーム内で関数が開始する場所と終了する場所、および関数のプロローグの場所が指定されます。

プログラム カウンタが特定のスタック フレームと関連付けられている場合、アンワインダによって PDATA のテーブルで格納されている関数に対応するエントリが検索されます。エントリが見つかると、関数フレームがアンワインドされます。

また、関数に関連付けられた例外処理ルーチンがある場合、PDATA 構造によってその場所が特定されます。

コンパイラでコンパイルされる関数については、正しいプロローグおよびエピローグのシーケンスと PDATA がコンパイラによって生成されますが、アセンブリ言語で記述する関数については、適切なコードと PDATA を自分で作成する必要があります。

仮想アンワインディングが機能するには、プロローグおよびエピローグ シーケンスが厳密なガイドラインに準拠している必要があります。

適切なプロローグおよびエピローグの詳細については、対象のプラットフォームのドキュメントを参照してください。

関連項目

参照情報

プロローグおよびエピローグの例

概念

プロローグ

仮想アンワインディング

その他のリソース

プロローグおよびエピローグ

プロローグ

プロローグは、間に命令が入らない連続する複数の部分で構成されます。

通常、プロローグ セグメントには、以下のタスクを実行する命令の個別シーケンスが格納されます。

- スタック フレームの割り当て。
- 入力引数レジスタの保存。
- フレーム ポインタの設定 (フレーム ポインタを確立する場合)。プロローグでは、初期レジスタを保存する前に、指定したレジスタにスタック ポインタがコピーされます。次に、この値を使用してフレーム ポインタの値が計算されます。
- リターン アドレスを格納したリンク レジスタの保存。
- コンパイラで生成される一時レジスタ、ローカル変数、および引数作成領域のための領域の割り当て。
- プロローグ コードの終了の指定。

簡潔なプロローグ コードを記述し、必要な演算だけを実行するようにします。

関連項目

参照情報

プロローグおよびエピローグの例

概念

RISC 環境の SEH

エピローグ

各プロシージャに含められるプロローグは 1 つだけですが、プロシージャで複数の終了ポイントを使用する場合は、そのプロシージャに多くのエピローグを含めることができます。

各エピローグを特定の部分で構成する必要があります。すべての部分を連続して記述し、間に命令を挿入しません。

通常、エピローグ セグメントには、以下のタスクを実行する命令の個別シーケンスが格納されます。

- フレーム ポインタ レジスタの復元 (プロローグにそのレジスタが保存されている場合)
- プログラム カウンタやスタックなどの不揮発性レジスタの復元
- リターン アドレスの復元
- ローカル フレームの割り当て解除
- 呼び出し元関数へのリターン

関連項目

参照情報

プロローグおよびエピローグの例

概念

RISC 環境の SEH

プロローグおよびエピローグの例

次のコード例では、SH-4 マイクロプロセッサ用に、64 KB のメモリを必要とするスタック フレームを割り当てています。ARM または MIPS のマイクロプロセッサ用の場合、コード セグメントは、使用するレジスタの名前を除いて同様です。

この例のプロローグ セグメントでは、引数レジスタが入力引数保存領域に保存されます。独立したフレーム ポインタは不要です。次に、リターン アドレスが保存され、永続レジスタが保存されて、スタック フレームが割り当てられます。また、例外ハンドラが宣言されます。

エピローグ セグメントでは、このスタック フレームは削除され、リターン アドレスが回復されます。

```
EXCEPTION_HANDLER RoutineHandler
NESTED_ENTRY Function
// Step 1.
//
mov.l R4, @R15   // Save argument to incoming argument save area.
mov.l R8, @-R15
sts.l PR, @-R15
mov.l @(0x0000001C,pc),r1 // Load constant -65528.
add     r1,r15      // Allocate stack frame.
mov.l R5, R8      // Save argument to register.

PROLOG_END

// Routine body

mov.l @(0x0000000C,pc),r1 // Load constant 65528.
add     r1,r15      // Remove stack frame.
lds.l @R15+, PR      // Recover return address.
rts
mov.l @R15+, R8     // Restore R8.

ENTRY_END Function
```

仮想アンワインディング

ルーチンのエントリに存在したコンテキストを再構築するために、RISC プロセッサの SEH では、仮想アンワインディングと呼ばれる処理を使用して、プロローグ コードおよびエピローグ コード内の命令の小さなサブセットがエミュレートされます。

仮想アンワインディングでは、構文的に効率がよい方法でカーネル例外ハンドラからユーザーモード コードにコントロールが移ります。

仮想アンワインディングでは、カーネルは呼び出しスタックをスキャンして適切な例外ハンドラを見つけます。

CPU コンテキスト レコードおよび命令アドレスを初めとして、アンワインディング処理ではプロローグまたはエピローグ内の命令が解釈され、関数呼び出しが行われる前の状態にコンテキストが再構築されます。

アンワインドの対象となるコード要素

仮想アンワインダは、PDATA **構造**を使用してプロシージャの開始と終了、およびプロローグの終了を判別します。**PDATA** 構造には、例外ハンドラへのポインタを格納することもできます。

仮想アンワインダがエミュレートするプロローグおよびエピローグのコードのサブセットの例を次に示します。

- レジスタの値の加算または減算
- スタック フレームのレジスタの読み込みと格納
- レジスタへの整数定数の読み込み
- レジスタ間の移動

仮想アンワインダは、プロローグまたはエピローグのシーケンス内で検出された他の命令を無視します。

仮想アンワインダによる処理

次の一覧は、**仮想アンワインダ**の実行手順を示しています。

1. フレーム ポインタ、スタック ポインタ、またはリンク レジスタを保存する命令をプロローグで検索します。
   命令が存在する場合、この命令は**仮想アンワインダ**で復元する必要がある永続レジスタをすべて保存します。
   命令が存在しない場合はリンク レジスタにリターン アドレスが格納され、**仮想アンワインダ**でプログラム カウンタだけが更新されます。
2. フレーム ポインタを書き込む命令をプロローグで検索します。アンワインディング処理では、番号の一番小さいレジスタから一番大きいレジスタまで、このアドレスから下のすべてのレジスタが復元されます。
3. スタックの書き込みを行う命令を検索します。命令が存在する場合、アンワインディング処理ではスタック リンクが逆実行されます。この減算の右オペランドは、定数の即値であるスタック サイズです。
4. プロローグ内で実行が停止すると、**仮想アンワインダ**は永続レジスタを保存する命令が実行済みかどうか、およびスタック リンクが実行済みかどうかを判別します。
   - この関数で永続レジスタが保存されていない場合、**仮想アンワインダ**はリンク レジスタの値をプログラム カウンタ レジスタにコピーします。
   - 関数でレジスタ値が保存されている場合にスタック リンクが実行されていないときは、**仮想アンワインダ**はスタック ポインタから永続レジスタを更新します。
   - スタックにリンクしているプロローグ内で実行が停止すると、**仮想アンワインダ**はそのプロローグを逆実行します。

注:
SEH を使用するには、スタック ポインタを移動するすべての関数に PDATA 構造を関連付けておく必要があります。スタック領域の割り当て、他の関数の呼び出し、永続レジスタの保存、または例外ハンドラの保持を行う関数はすべてこれに該当します。永続レジスタを変更しないリーフ関数 (他の関数を呼び出さない関数) では、PDATA を必要としません。この場合、**仮想アンワインダ**はリンク レジスタからプログラム カウンタを更新し、次のフレームに進みます。

関連項目

参照情報

プロローグおよびエピローグの例
PDATA **構造**

概念

RISC **環境の** SEH

PDATA 構造

ARM、MIPS、および SHx デバイス コンパイラでは、実行時のスタック スキャンに PDATA 構造が使用されます。この構造は、デバッグおよび例外処理に役立ちます。

コンパイラは、各プロシージャに PDATA 構造を 1 つ関連付けます。

このデータ構造は、COFF .pdata セクションに格納されるテーブルです。.pdata セクションは、例外処理のための関数テーブル エントリの配列を格納し、イメージ データ ディレクトリの例外テーブル エントリによってポイントされます。

MIPS 呼び出し標準では、非圧縮 PDATA 形式の _IMAGE_ALPHA_RUNTIME_FUNCTION_ENTRY がサポートされます。ほとんどの場合、現在 MIPSII では、_IMAGE_ALPHA_RUNTIME_FUNCTION_ENTRY エントリの各関数に非圧縮 20 バイトが使用されます。

例外処理ルーチンに関連付けられていないリーフ関数には、pdata エントリは関連付けられていません。

次の表に、MIPSII イメージの関数テーブル エントリの形式を示します。

オフセット	サイズ	フィールド	説明
0	4	Begin Address (開始アドレス)	対応する関数の仮想アドレス。
4	4	End Address (終了アドレス)	関数終了の仮想アドレス。
8	4	Exception Handler (例外ハンドラ)	実行される例外ハンドラへのポインタ。
12	4	Handler Data (ハンドラ データ)	ハンドラに渡される追加情報へのポインタ。
16	4	Prolog End Address (プロローグ終了アドレス)	関数プロローグ終了の仮想アドレス。

ARM および SH-4 デバイス コンパイラでは、圧縮 PDATA 構造 _IMAGE_CE_RUNTIME_FUNCTION_ENTRY がサポートされます。

次の表に、ARM および SH-4 ハードウェア プラットフォーム用 COFF 指定関数テーブル エントリの形式を示します。

オフセット	サイズ	フィールド	説明
0	4	Begin Address (開始アドレス)	対応する関数の仮想アドレス。
4	8 ビット	Prolog Length (プロローグの長さ)	関数のプロローグ内の命令の数。
4	22 ビット	Function Length (関数の長さ)	関数内の命令の数。
4	1 ビット	32-bit Flag (32 ビット フラグ)	関数が 32 ビット命令で構成されている場合に設定し、16 ビット関数の場合はクリアします。
4	1 ビット	Exception Flag (例外フラグ)	関数に例外ハンドラが存在する場合に設定します。

例外ハンドラが存在する場合、または関数長がゼロの場合、.text セクションの関数の前に別の PDATA_EH 構造があります。関数に例外ハンドラまたはハンドラ データが関連付けられている場合、関数は PDATA_EH を使用します。

ほとんどの場合、**PDATA** 構造では関数 1 つあたりに 8 バイトだけ占有されます。例外ハンドラがある関数の場合は PDATA_EH 構造用にさらに 8 バイトが必要です。

例外処理データ レコード、およびプロローグと関数長のレコードは、必ず 4 バイトに整列されます。これは、これらのレコードが 1 つ以上関連付けられている関数はすべて 4 バイト整列であることを意味します。

関連項目
概念
仮想アンワインディング

_IMAGE_CE_RUNTIME_FUNCTION_ENTRY

この構造には、実行時の例外処理に関する詳細情報が格納されます。

_IMAGE_CE_RUNTIME_FUNCTION_ENTRY は、ARM および Renesas マイクロプロセッサ ファミリのみで使用されます。MIPS マイクロプロセッサには適用されません。

```
typedef struct _IMAGE_CE_RUNTIME_FUNCTION_ENTRY {
  unsigned int FuncStart : 32;
  unsigned int PrologLen : 8;
  unsigned int FuncLen : 22;
  unsigned int ThirtyTwoBit : 1;
  unsigned int ExceptionFlag : 1;
} IMAGE_CE_RUNTIME_FUNCTION_ENTRY,
*PIMAGE_CE_RUNTIME_FUNCTION_ENTRY;
```

パラメータ

FuncStart

関数の最初の命令のアドレス。

これは、関数のエントリ アドレスです。

PrologLen

ThirtyTwoBit フラグで指定された命令のサイズに基づいた、命令のプロローグの長さ。

PrologLen は、ARM 関数の場合は 1 に、THUMB 関数および SH-4 関数の場合は 0 に設定されます。

FuncLen

命令の関数全体の長さ。上の *PrologLen* を参照してください。ARM 命令が 200 ある関数の *FuncLen* は 200、つまり 800 バイトになります。

ThirtyTwoBit

関数内の命令のサイズ。*ThirtyTwoBit* には、以下のいずれかの値を格納できます。

1 は ARM 関数を表します。各関数は 32 ビット命令、つまり 4 バイトで構成されます。

0 は THUMB 関数を表します。各関数は 16 ビット命令、つまり 2 バイトで構成されます。

ExceptionFlag

関連付けられた例外ハンドラまたはハンドラ データ。

値が 1 の場合、**.text** セクションに PDATA_EH が存在します。

ExceptionFlag が 0 の場合、PDATA_EH は存在しません。

備考

IMAGE_CE_RUNTIME_FUNCTION_ENTRY データ構造は **PDATA** とも呼ばれます。これらのレコードを格納するテーブルは、.pdata と呼ばれるセクションに保存されます。.pdata セクションは、デバッグおよび例外処理に使用されます。

ExceptionFlag が設定されている場合、または *FuncLen* が 0 に設定されている場合、**.text** セクションの関数の前に別の PDATA_EH 構造が存在します。

ExceptionFlag ビットが設定されている場合のみ、**.text** セクションの関数のすぐ前にこの情報を格納するデータ レコードが存在します。

このレコードは、関数に関連付けられた例外ハンドラまたはハンドラ データがある場合にのみ使用されます。

関連項目

参照情報

PDATA 構造

PDATA_EH

_IMAGE_ALPHA_RUNTIME_FUNCTION_ENTRY

この構造には、実行時の例外処理に関する詳細情報が格納されます。

この構造は非圧縮 20 バイト形式です。

```
typedef struct_IMAGE_ALPHA_RUNTIME_FUNCTION_ENTRY {  ULONG BeginAddress;  ULONG EndAddress;
PVOID ExceptionHandler;  PVOID HandlerData;  ULONG PrologEndAddress;} IMAGE_ALPHA_RUNTIME_F
UNCTION_ENTRY,*PIMAGE_ALPHA_RUNTIME_FUNCTIONG_ENTRY;
```

パラメータ

BeginAddress

関数の最初の命令のアドレス。

これは、関数のエントリ アドレスです。

EndAddress

関数の最後の命令のアドレス。

これは、関数の終了アドレスです。

ExceptionHandler

関数の例外ハンドラのアドレス。

HandlerData

関数の例外ハンドラのデータ レコードのアドレス。

PrologEndAddress

プロローグの最後の命令のアドレス。

関連項目

参照情報

PDATA **構造**

PDATA_EH

この構造には、関連付けられている例外ハンドラ関数の詳細な情報が格納されます。これは、OS 例外処理で使用される内部データ構造です。

```
struct PDATA_EH {  unsigned int* pHandler;  unsigned int* pHandlerData;};
```

パラメータ

pHandler

関数の例外ハンドラのアドレス。

pHandlerData

関数の例外ハンドラのデータ レコードのアドレス。

関連項目

参照情報

PDATA **構造**

_IMAGE_CE_RUNTIME_FUNCTION_ENTRY

デバイス コンパイラ エラー メッセージ

次の表は、デバイス コンパイラの固有のエラー メッセージの説明です。

エラー	説明
コンパイラ エラー C2729	組み込み関数が Thumb モードでサポートされないことを示します。
コンパイラ エラー C2759	インライン アセンブリのエラーを示します。
コンパイラ エラー C2822	保護されたセクションが途中で終了したことを示します。
コンパイラ エラー C2880	名前空間が既に存在しているために、名前空間エイリアスの作成に失敗したことを示します。
コンパイラ エラー C2887	__swi 組み込み関数の引数が多すぎることを示します。
コンパイラの警告 (レベル 2) C4720	SH 固有のさまざまな問題の 1 つを示します。
コンパイラの警告 (レベル 1) C4721	不明な組み込み関数であることを示します。
コンパイラの警告 (レベル 1) C4732	ターゲットのアーキテクチャで組み込み関数がサポートされないことを示します。
コンパイラの警告 (レベル 1) C4567	異なるバージョンのコンパイラによる互換性のないオブジェクト ファイルがリンカで検出されたことを示します。

関連項目

その他のリソース

スマート デバイスのコンパイラ

コンパイラ エラー C2729

```
intrinsic not allowed in Thumb mode
```

このエラーは、コンパイラが、_prefetch などの Thumb モードでサポートされていない組み込み関数をコンパイルしようとしたことを示します。

ソースが /GL (プログラム全体の最適化) スイッチを使用してコンパイルされる場合、参照したオブジェクトがリンクされるまでこのエラーは出力されません。

コンパイラ エラー C2759

```
in-line assembler reports: "various"
```

インライン アセンブリ コード内で発生するエラーで、コンパイルに失敗しました。

コンパイラ エラー C2822

```
local unwind is not supported on this platform
```

このプラットフォームでの構造化例外処理の実装では、try-finally ステートメントの保護されたセクションまたは終了ハンドラを途中で終了するときに必要なローカルのアンワインド処理はサポートされません。保護されたセクションを終了する必要がある場合は、__leave キーワードを使用します。終了ハンドラを途中で終了すると、未定義の動作が起こる可能性があるため、途中終了は避ける必要があります。

次のコードでは、このエラー メッセージが生成される例を 2 つ示しています。

```
int g;

int main(void)
{
    __try {
        if (g) return g; // requires local unwind
        g = 1;
    } __finally {
        if (g) return g; // undefined; requires local unwind
        g = 2;
    }

    return 0;
}
```

```

```

コンパイラ エラー C2880

```
__swi requires a valid constant as first argument (SWI number)
```

_swi 組み込み関数は、最初の引数に必要な整数定数を受け取りませんでした。整数定数は、ARM マイクロプロセッサの場合は [0 - 16777215]、Thumb マイクロプロセッサの場合は [0 - 255] の範囲内である必要があります。

例

```
int test_intrinsic(int x)
{

     return __swi(x, 12, 14, 13, 12); // error C2880
}
```

コンパイラ エラー C2887

```
__swi cannot have more than five arguments (SWI number r0 - r3)
```

__swi **組み込み関数には、**5 **個以下の引数を指定する必要があります。**

例

```
#pragma intrinsic(__swi)
int test_intrinsic()
{
     return __swi(10, 12, 14, 13, 15, 12);
 // error cannot have more than 5 args
}
```

コンパイラの警告 (レベル 2) C4720

```
in-line assembler reports: 'message'
```

この SH 固有の警告は、SH インライン アセンブリで生じる可能性のある複数の状況に適用されます。

この警告は、-GL (プログラム全体の最適化) オプションを指定してコンパイルした場合は表示されません。

次のコード例では、コンパイラは遅延スロットの分岐を示すためにこの警告を発行します。

例

```
/* C4720.c */
#ifdef __cplusplus
extern "C" void __asm(const char *, ...);
#else
extern void __asm(const char *,...);
#endif

int main()
{
   int ValA = 10;
   int ValB = 0;
   __asm(
      "mov.l @r4, r2\n"
      "mov #0, r6\n"
      "add r2, r6\n"
      "bf/s lala\n"
      "bf la\n"
      "lala: add r2, r6\n"
      "la: mov.l r6, @r5\n",
      &ValA,&ValB);     /* delay slot branch */
   return 0;
}
```

コンパイラの警告 (レベル 1) C4721

```
'function' : not available as an intrinsic
```

コンパイラで不明な組み込み関数が検出されました。#pragma intrinsic('function') **は無視されます。**

コンパイラの警告 (レベル 1) C4732

```
instrinsic ' %s' is not supported in this architecture
```

コンパイラで、対象のアーキテクチャでサポートされていない組み込み関数が検出されました。

__trap 組み込み関数は MIPS 16 ISA ではサポートされません。このため、次のコード例の場合、コンパイラの警告 C4732 が生成されます。

例

```
#include <cmnintrin.h>

int main()
{
  int returnCode = 1;
      #if (_INTRINSIC_IS_SUPPORTED(__trap))
              __trap(1);
      #else
              return 1;
      #endif
  return 0;
}
```

コンパイラの警告 (レベル 1) C4567

```
'function' : behavior change due to parameter 'parameter': calling convention incompatible
with previous compiler versions
```

この警告は、古いバージョンのコンパイラによってコンパイルされたコードとリンクされている場合、正しく実行されない可能性があるコードをコンパイラが検出したことを示します。

この警告は、ARM(R) アーキテクチャの C++ コンパイラに固有です。14.00 より古いバージョンのコンパイラでは、特定の関数パラメータを値で渡すときに、それ以降のコンパイラとは異なる呼び出し規則が使用されます。この 2 つの呼び出し規則には互換性がないため、古いコンパイラのオブジェクト ファイルを新しいオブジェクト ファイルとリンクした場合、古いオブジェクト ファイルと新しいオブジェクト ファイル間でこのようなパラメータが値で渡されると、予期しない動作が生じてクラッシュすることがあります。

ユーザー定義のコピー コンストラクタを持つクラス、構造体、または集合体の種類のオブジェクトが値で渡される場合、この呼び出し規則の変更の影響を受けます。参照で渡されるオブジェクトには影響はありません。

古いコンパイラのオブジェクト ファイルにリンクする場合は、この警告を使用して、呼び出し規則が変更されたコード内の場所を探してください。ユーザー定義のコピー コンストラクタを持つオブジェクトが古いオブジェクト ファイルと新しいオブジェクト ファイル間で値で渡される場合は、バージョン 14.00 以降のコンパイラで古いオブジェクト ファイルを再コンパイルする必要があります。

この警告は、既定ではオフになっています。

例

```
// The following sample generates C4567:

// C4567.cpp
// (optional) compile with: -w14567
#pragma warning(default : 4567)
#pragma inline_depth(0) // disable function inlining

#include <cstdio>

struct S {
   S () { self = this; }
   S (S& that) { self = this; }
   ~S() {  // older compilers will fail this test
      if ( self != this ) {
         printf ("s passed incorrectly\n");
      }
   }
   S* self;
};

void func ( S s ) // C4567 at definition
{
   // s destructor is called here
}

int main()
{
   S s;
   func (s); // C4567 at call site
   return 0;
}
```

関連項目

その他のリソース

デスクトップ コンパイラとデバイス コンパイラの違い

ARM ファミリ プロセッサ

ARM マイクロプロセッサ群は、次のようなアプリケーションのためのソリューションを提供します。

- ワイヤレス アプリケーション、コンシューマ アプリケーション、およびイメージング アプリケーションがインストールされている複雑なオペレーティング システムを実行するオープン ハードウェア プラットフォーム
- 大容量記憶装置、自動車用、産業用、およびネットワーク用の各アプリケーションのための埋め込み型リアルタイム システム
- スマート カードや SIMM (シングル インライン メモリ モジュール) などのセキュリティで保護されたアプリケーション

ARM ISA (命令セット アーキテクチャ) には、最適な機能とパフォーマンスを可能にする THUMB テクノロジなどの拡張テクノロジが含まれています。

このセクションのトピック

ARM マイクロプロセッサの組み込み関数

主な ARM マイクロプロセッサ ファミリでサポートされる組み込み関数の表および詳細な参照情報を提供します。

ARM コンパイラ オプション

ARM マイクロプロセッサとコンパイラに固有のコンパイラ オプションに関する参照情報を提供します。

ARM コーリング シーケンスの仕様

ARM レジスタとスタック フレーム レイアウト、SEH の ARM プロローグとエピローグ、および ARM アセンブラに関する参照情報を提供します。

関連セクション

デバイス コンパイラの組み込み関数

すべてのデバイス コンパイラでサポートされる組み込み関数に関する参照情報を提供します。

デバイス コンパイラのデータ配置に関する問題

RISC マイクロプロセッサのデータ配置のガイドラインを提供します。

RISC 環境の SEH

RISC 環境における構造化例外処理の主な違いについて説明しています。

ARM マイクロプロセッサの組み込み関数

ARM デバイス コンパイラでは、特定の ARM マイクロプロセッサに定義される一連の組み込み関数がサポートされます。

組み込み関数の各種セットは、ARM10、ARM DSP、ARM XSCALE、および Intel PXA270 アーキテクチャで使用できます。

ARM コンパイラでは、**/QR** コンパイラ オプションを使用して、特定のコンパイルに対して選択する組み込み関数のセットが判別されます。たとえば、**/QRxscale** を指定すると XScale MAC 組み込み関数を使用できます。**/QR** フラグの詳細については、「ARM コンパイラ オプション」を参照してください。

適切なターゲット アーキテクチャが定義されていない場合、XScale MAC 関数など組み込み関数によっては失敗するものがあります。

組み込み関数を保護する場合、または特定のターゲット アーキテクチャが定義済みのときにのみ組み込み関数を呼び出す場合は、システム管理関数 **IsProcessorFeaturePresent** を使用します。

このセクションのトピック

ARM10 組み込み関数

以下の ARM10 命令は、組み込み関数によってサポートされています。

命令	説明
CLZ	最初の 1 ビットの前にある先行ゼロをカウントします。 **CLZ** 命令には、共通組み込み関数 _CountLeadingZeros、_CountLeadingZeros64 がアクセスします。
BKPT	ソフト ブレークポイントを作成します。 BKPT 命令には、共通組み込み関数 __trap がアクセスします。
_swi	SWI ソフトウェア割り込み命令を使用して、OS への呼び出しを生成します。
__emit	命令ストリームに、指定された命令を挿入します。
_MoveFromCoProcessor、_MoveFromCoProcessor2	ARM コプロセッサからデータを読み取ります。
_MoveToCoProcessor、MoveToCoprocessor2	ARM コプロセッサにデータを書き込みます。

関連項目

参照情報

ARM コンパイラ オプション

CLZ

ARM10 命令は、レジスタ値の先頭バイナリ 1 ビットの前のバイナリ ゼロ ビットの数をカウントします。共通組み込み関数 _CountLeadingZeros、_CountLeadingZeros64 は **CLZ** をサポートしています。

```
unsigned cdecl _CountLeadingZeros(
  long Arg1
);
```

パラメータ

Arg1

[in] 先行ゼロ ビットを決定する値。

戻り値

バイナリ ゼロ ビットの数。

備考

_CountLeadingZeros 組み込み関数の **CLZ** 命令を生成するには、**-QRarch5** または **-QRarch5T** フラグを使用します。

ARM4 マイクロプロセッサでコンパイルを行うと、ライブラリ名の呼び出し、または ARM4 コードの他のシーケンスの呼び出しがコンパイラで生成されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
CLZ	<armintr.h>	ARM

関連項目

参照情報

ARM10 組み込み関数

/QRArch – ターゲットのアーキテクチャの指定

その他のリソース

共通組み込み関数の参照情報

BKPT

この ARM10 命令は、ソフトウェアのブレークポイントを設定します。__trap 共通組み込み関数は BKPT をサポートします。

```
int __trap(
  int Arg1
);
```

パラメータ

Arg1

[in] ブレークポイントが設定された命令のアドレス。

戻り値

なし。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
BKPT	<armintr.h>	ARM

関連項目

参照情報

ARM10 組み込み関数

その他のリソース

共通組み込み関数の参照情報

__emit

この組み込み関数は、コンパイラによって出力された命令のストリームに、指定された命令を挿入します

```
void __emit(
   const unsigned __int32 opcode
);
```

パラメータ

opcode

挿入される命令ワード。

戻り値

なし。

備考

opcode の値は、コンパイル時に既知の定数式にする必要があります。

コンパイラは *opcode* の内容を解釈しようとしません。また、挿入された命令の実行前に CPU またはメモリの状態を保証しません。

コンパイラは、挿入された命令の実行後も CPU とメモリの状態に変更がないものと見なします。このため、状態を変更する命令の場合は、コンパイラによって生成された通常のコードに有害な影響を及ぼす場合があります。

このような理由により、コプロセッサの状態など通常ではコンパイラが処理しない CPU の状態に影響を与える命令の挿入、または **__declspec(naked)** で宣言される関数を実装する場合にのみ **__emit** を使用します。

ARM 命令を生成するときの命令ワードのサイズは 32 ビットです。/QRthumb を指定した場合と同様に、Thumb 命令を生成するときの命令ワードのサイズは 16 ビットで、*opcode* の最上位の 16 ビットは無視されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
__emit	<armintr.h>	ARM

関連項目

参照情報

ARM10 組み込み関数

/QRthumb

_MoveFromCoProcessor、_MoveFromCoProcessor2

これらの組み込み関数は、コプロセッサ データ転送命令を介して ARM コプロセッサのデータを読み取ります。

```
int _MoveFromCoprocessor(
      unsigned int coproc,
      unsigned int opcode1,
      unsigned int crn,
      unsigned int crm,
      unsigned int opcode2
);

int _MoveFromCoprocessor2(
      unsigned int coproc,
      unsigned int opcode1,
      unsigned int crn,
      unsigned int crm,
      unsigned int opcode2
);
```

パラメータ

coproc

0 から 15 までの範囲のコプロセッサ番号。

opcode1

0 から 7 までの範囲のコプロセッサ固有の命令コード。

crn

0 から 15 までの範囲のコプロセッサ レジスタ番号。命令の第 1 オペランドを指定します。

crm

0 から 15 までの範囲のコプロセッサ レジスタ番号。ソースまたは出力先の追加オペランドを指定します。

opcode2

0 から 7 までの範囲のコプロセッサ固有の追加命令コード。

戻り値

コプロセッサから読み取られる値。

備考

この組み込み関数の 5 つのパラメータ値はすべて、コンパイル時に既知の定数式である必要があります。

_MoveFromCoprocessor では MRC 命令が、**_MoveFromCoprocessor2** では MRC2 命令が使用されます。パラメータは命令ワードに直接エンコードされた bitfield に対応します。パラメータの解釈は、コプロセッサに依存します。詳細については、各コプロセッサのマニュアルを参照してください。

/QRthumb を指定する場合など、THUMB 命令の生成時にはこの組み込み関数を使用できません。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_MoveFromCoprocessor	<armintr.h>	ARM
ルーチン	**必須のヘッダー**	**アーキテクチャ**
_MoveFromCoprocessor2	<armintr.h>	ARM

関連項目

参照情報

ARM10 **組み込み関数**

/QRthumb

_MoveToCoProcessor、MoveToCoprocessor2

これらの組み込み関数は、コプロセッサ データ転送命令を介して ARM コプロセッサにデータを書き込みます。

```
Void _MoveToCoprocessor(
      unsigned int value,
      unsigned int coproc,
      unsigned int opcode1,
      unsigned int crn,
      unsigned int crm,
      unsigned int opcode2
);

Void _MoveToCoprocessor2(
      unsigned int value,
      unsigned int coproc,
      unsigned int opcode1,
      unsigned int crn,
      unsigned int crm,
      unsigned int opcode2
);
```

パラメータ

value

コプロセッサに書き込まれる値。

coproc

0 から 15 までの範囲のコプロセッサ番号。

opcode1

0 から 7 までの範囲のコプロセッサ固有の命令コード。

crn

0 から 15 までの範囲のコプロセッサ レジスタ番号。命令の第 1 オペランドを指定します。

crm

0 から 15 までの範囲のコプロセッサ レジスタ番号。ソースまたは出力先の追加オペランドを指定します。

opcode2

0 から 7 までの範囲のコプロセッサ固有の追加命令コード。

戻り値

なし。

備考

coproc、*opcode1*、*crn*、*crm*、および *opcode2* は、コンパイル時に既知の定数式である必要があります。

_MoveToCoprocessor では **MCR** 命令が、**_MoveToCoprocessor2** では **MCR2** が使用されます。

パラメータは命令ワードに直接エンコードされた bitfield に対応します。パラメータの解釈は、コプロセッサに依存します。詳細については、各コプロセッサのマニュアルを参照してください。

/QRthumb を指定する場合など、THUMB 命令の生成時にはこの組み込み関数を使用できません。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_MoveToCoprocessor、 **_MoveToCoprocessor2**	<armintr.h>	ARM

関連項目

参照情報

ARM10 **組み込み関数**

/QRthumb

_swi

この組み込み関数は、ソフトウェア割り込み命令 **SWI** を使用して OS ルーチンへの呼び出しを生成します。

```
unsigned int __swi(
   unsigned swi_number,
   arg2,
   arg3,
   arg4
);
```

パラメータ

swi_number

ソフトウェア割り込み番号。

arg2-arg4

渡される追加の引数。

戻り値

_swi 組み込み関数は、SWI に続いて、コントロールが命令に戻ったときにレジスタ R0 に残っている値を返します。

備考

_swi 組み込み関数は、コンパイラが 32 ビット コードと 16 ビット コードのいずれを生成しているかに従って ARM または THUMB 命令セットに適用されます。

第 1 パラメータ *swi_number,* は、命令の即時フィールドに直接エンコードされます。ARM の場合は [0 - 16777215]、THUMB の場合は [0 - 255] の範囲の整数定数にする必要があります。

追加引数が指定されている場合、関数は標準の ARM 呼び出し規則に従って値を渡しますが、次のように例外が 1 つ適用されます。引数または引数の部分をメモリ、つまりスタックに渡すことはできません。

このため、レジスタ **R0**、**R1**、**R2**、および **R3** のみを使用して、すべての引数を渡せるようにする必要があります。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_swi	<armintr.h>	ARM

関連項目

参照情報

/QRArch – ターゲットのアーキテクチャの指定

/QRthumb

ARM DSP 拡張組み込み関数

すべての ARM DSP 命令は組み込み関数としてサポートされています。ARM DSP 組み込み関数を使用するには、armintr.h ヘッダーを含めます。

次の ARM DSP 命令には、組み込み関数が定義されています。

関数	対応する ARM DSP 命令	説明
_SmulAddLo_SW_SL _SmulAddHi_SW_SL _SmulAddHiLo_SW_SL _SmulAddLoHi_SW_SL	**SMLAxy**	符号付き整数の乗算および累算。16 × 16 ビットの乗算の後、32 ビットの加算を実行します。
_SmulAddWLo_SW_SL _SmulAddWHi_SW_SL	**SMLAWy**	32 × 16 ビットの乗算の後、48 ビットの積の上位 32 ビットの、32 ビットの加算を実行します。
_SmulAddHi_SW_SQ _SmulAddLo_SW_SQ _SmulAddHiLo_SW_SQ _SmulAddLoHi_SW_SQ	**SMLALxy**	16 × 16 ビットの乗算の後、その積に 64 ビット整数との 64 ビットの加算を実行します。
_SmulLo_SW_SL _SmulHi_SW_SL _SmulHiLo_SW_SL _SmulLoHi_SW_SL	**SMULxy**	符号付き整数の 16 × 16 ビットの乗算。
_SmulWLo_SW_SL _SmulWHi_SW_SL	**SMULWy**	符号付き整数の 32 × 16 ビットの乗算を実行し、上位 32 ビットを返します。
_AddSatInt	**QADD**	飽和加算命令。
_DSubSatInt	**QSUB**	飽和減算命令。
_DAddSatInt	**QDADD**	整数を 2 倍して飽和し、その結果を別の整数に加算して飽和します。
_DSubSatInt	**QDSUB**	整数を 2 倍して飽和し、その結果を別の整数から除算して飽和します。
_ReadCoProcessor _WriteCoProcessor	**MRRC、** **MCRR**	値をコプロセッサから 2 つの ARM レジスタに転送します。

関連項目

参照情報

ARM XSCALE 組み込み関数

その他のリソース

デバイス コンパイラの組み込み関数

_SmulAddLo_SW_SL

この ARM DSP **拡張、符合付き整数の乗算および累算では、レジスタ Rm の下半分とレジスタ Rs の下半分が乗算され、**32 **ビットの積が生成されます。次に、Rn との** 32 **ビットの累算が実行されます。**

```
int _SmulAddLo_SW_SL(
  int Arg1,
  int Arg2,
  int Arg3
);
```

パラメータ

Arg1

Rn の内容。*Arg2* **と** *Arg3* **の積に加算される値。**

Arg2

[in] **Rm の内容。乗算の第** 1 **項。**

Arg3

[in] **Rs の内容。乗算の第** 2 **項。**

戻り値

乗算と累算の結果。

備考

この命令は、コンパイラで smlabb アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddLo_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_SmulAddHi_SW_SL

_SmulAddHiLo_SW_SL

_SmulAddLoHi_SW_SL

_SmulAddHi_SW_SL

この ARM DSP **拡張、符合付き整数の乗算および累算では、レジスタ Rm の上半分とレジスタ Rs の上半分が乗算され、**32 **ビットの積が生成されます。次に、Rn との** 32 **ビットの累算が実行されます。**

```
int _SmulAddHi_SW_SL(
  int Arg1,
  int Arg2,
  int Arg3
);
```

パラメータ

Arg1

Rn の内容。*Arg2* **と** *Arg3* **の積に加算される値。**

Arg2

[in] **Rm の内容、乗算の第** 1 **項。**

Arg3

[in] **Rs の内容、乗算の第** 2 **項。**

戻り値

乗算と累算の結果。

備考

この命令は、コンパイラで smlatt アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddHi_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**
_SmulAddLo_SW_SL
_SmulAddHiLo_SW_SL
_SmulAddLoHi_SW_SL

_SmulAddHiLo_SW_SL

この ARM DSP **拡張、符号付き整数の乗算および累算では、レジスタ Rm の上半分とレジスタ Rs の下半分が乗算され、**32 **ビットの積が生成されます。次に、この演算では、Rn と** 32 **ビットの累算が実行されます。**

```
int _SmulAddHiLo_SW_SL(
  int Arg1,
  int Arg2,
  int Arg3
);
```

パラメータ

Arg1

Rn の内容。*Arg2* **と** *Arg3* **の積に加算される値。**

Arg2

[in] **Rm の内容、乗算の第** 1 **項。**

Arg3

[in] **Rs の内容、乗算の第** 2 **項。**

戻り値

乗算と累算の結果。

備考

この命令は、コンパイラで smlatb アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddHiLo_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**
_SmulAddHi_SW_SL
_SmulAddLo_SW_SL
_SmulAddLoHi_SW_SL

_SmulAddLoHi_SW_SL

この ARM DSP **拡張、符合付き整数の乗算および累算では、レジスタ Rm の下半分とレジスタ Rs の上半分が乗算され、**32 **ビットの積が生成されます。次に、Rn との** 32 **ビットの累算が実行されます。**

```
int _SmulAddLoHi_SW_SL(
  int Arg1,
  int Arg2,
  int Arg3
);
```

パラメータ

Arg1

Rn の内容。*Arg2* **と** *Arg3* **の積に加算される値。**

Arg2

[in] **Rm の内容、乗算の第** 1 **項。**

Arg3

[in] **Rs の内容、乗算の第** 2 **項。**

戻り値

乗算結果の整数。

備考

この命令は、コンパイラで smlabt アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddLoHi_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_SmulAddLo_SW_SL

_SmulAddHiLo_SW_SL

_SmulAddHi_SW_SL

_SmulAddWLo_SW_SL

この ARM DSP **拡張、符号付き整数の乗算および累算では、Rm と Rs の下位** 16 **ビットが乗算され、その結果が Rn に累算されます。この演算は、**48 **ビットの積の上位** 32 **ビットを** 32 **ビットの Rn に加算します。**

```
int _SmulAddWLo_SW_SL(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **Rm の内容、積の第** 1 **項。**

Arg2

[in] **Rs の内容、積の第** 2 **項。**

戻り値

乗算および累算結果の整数。

備考

この命令は、コンパイラで smlawb アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddW_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_SmulAddWHi_SW_SL

_SmulAddWHi_SW_SL

この ARM DSP **拡張、符号付き整数の乗算および累算では、Rm と Rs の上位** 16 **ビットを乗算し、Rn に累算します。この演算は、**48 **ビットの積の上位** 32 **ビットを** 32 **ビットの Rn に加算します。**

```
int _SmulAddWHi_SW_SL(
  int Arg1,
  int Arg2,
  int Arg3
);
```

パラメータ

Arg1

[in] **Rn の内容。***Arg2* **と** *Arg3* **の積に加算される値。**

Arg2

[in] **Rm の内容、積の第** 1 **項。**

Arg3

[in] **Rs の内容、積の第** 2 **項。**

戻り値

乗算および累算結果の整数。

備考

この命令は、コンパイラで smlawt アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddWHi_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_SmulAddWLo_SW_SL

_SmulAddHi_SW_SQ

この ARM DSP 拡張、符号付き整数の乗算および累算では、最初にレジスタ **Rm** の上半分とレジスタ **Rs** の上半分の 2 つの 16 ビット ソース オペランドに対して乗算が実行されます。次に、32 ビット レジスタの RdLo と RdHi の 64 ビットの累算が実行されます。

```
__int64 _SmulAddHi_SW_SQ(
  __int64 Arg1,
  int Arg2,
  int Arg3
);
```

パラメータ

Arg1

RdHi と RdLo を含む 64 ビットの累算へのポインタ。

Arg2

[in] **Rm** の内容、積の第 1 項。

Arg3

[in] **Rs** の内容、積の第 2 項。

戻り値

乗算および累算結果の長整数。

備考

この命令は、コンパイラで **smlaltt** アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddHi_SW_SQ	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**
_SmulAddLo_SW_SQ
_SmulAddHiLo_SW_SQ
_SmulAddLoHi_SW_SQ

_SmulAddLo_SW_SQ

この ARM DSP **拡張、符号付き整数の乗算および累算では、最初にレジスタ Rm の下半分とレジスタ Rs の下半分の** 2 **つの** 16 **ビット ソース オペランドに対して乗算が実行されます。次に、**32 **ビット レジスタの** RdLo **と** RdHi **の** 64 **ビットの累算が実行されます。**

```
__int64 _SmulAddLo_SW_SQ(
  __int64 Arg1,
  int Arg2,
  int Arg3
);
```

パラメータ

Arg1

RdHi **と** RdLo **の内容の累算に使用する** 64 **ビット変数へのポインタ。**

Arg2

[in] **Rm の内容、積の第** 1 **項。**

Arg3

[in] **Rs の内容、積の第** 2 **項。**

戻り値

乗算と累算の結果。

備考

この命令は、コンパイラで smlalbb アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddLo_SW_SQ	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_SmulAddHi_SW_SQ

_SmulAddHiLo_SW_SQ

_SmulAddLoHi_SW_SQ

_SmulAddHiLo_SW_SQ

この ARM DSP 拡張、符号付き整数の乗算および累算では、レジスタ **Rm** の上半分とレジスタ **Rs** の下半分が乗算されます。次に、32 ビット レジスタの RdLo と RdHi の 64 ビットの累算が実行されます。

```
__int64 _SmulAddHiLo_SW_SQ(
  __int64 Arg1,
  int Arg2,
  int Arg3
);
```

パラメータ

Arg1

RdHi と RdLo の内容の累算に使用する 64 ビット変数へのポインタ。

Arg2

[in] **Rm** の内容。積の第 1 項。

Arg3

[in] **Rs** の内容、積の第 2 項。

戻り値

乗算および累算結果の長整数。

備考

この命令は、コンパイラで **smlaltb** アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddHiLo_SW_SQ	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP 拡張組み込み関数
_SmulAddLo_SW_SQ
_SmulAddLoHi_SW_SQ
_SmulAddHi_SW_SQ

_SmulAddLoHi_SW_SQ

この ARM DSP 拡張、符号付き整数の乗算および累算では、レジスタ **Rm** の下半分とレジスタ **Rs** の上半分が乗算されます。次に、32 ビット レジスタの RdLo と RdHi の 64 ビットの累算が実行されます。

```
__int64 _SmulAddLoHi_SW_SQ(
  __int64 Arg1,
  int Arg2,
  int Arg3
);
```

パラメータ

Arg1

RdHi と RdLo の内容の累算に使用する 64 ビット変数へのポインタ。

Arg2

[in] **Rm** の内容、積の第 1 項。

Arg3

[in] **Rs** の内容、積の第 2 項。

戻り値

乗算および累算結果の長整数。

備考

この命令は、コンパイラで **smlalbt** アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddLoHi_SW_SQ	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP 拡張組み込み関数
_SmulAddLo_SW_SQ
_SmulAddHiLo_SW_SQ
_SmulAddHi_SW_SQ

_SmulHi_SW_SL

この ARM DSP **拡張、符号付き整数の乗算では、レジスタ Rm の上半分とレジスタ Rs の上半分が乗算され、**32 **ビットの結果が Rd に生成されます。**

```
int _SmulHi_SW_SL(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **Rm の内容、積の第** 1 **項。**

Arg2

[in] **Rs の内容、積の第** 2 **項。**

戻り値

乗算結果の整数。

備考

この命令は、コンパイラで smultt アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulHi_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_SmulLo_SW_SL

_SmulLoHi_SW_SL

_SmulHiLo_SW_SL

_SmulLo_SW_SL

この ARM DSP **拡張、符号付き整数の乗算では、レジスタ Rm の下半分とレジスタ Rs の下半分が乗算され、**32 **ビットの結果が Rd に生成されます。**

```
int _SmulLo_SW_SL(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **Rm の内容、積の第** 1 **項。**

Arg2

[in] **Rs の内容、積の第** 2 **項。**

戻り値

乗算結果の整数。

備考

この命令は、コンパイラで smulbb アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulLo_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_SmulLoHi_SW_SL

_SmulHiLo_SW_SL

_SmulHi_SW_SL

_SmulHiLo_SW_SL

この ARM DSP **拡張、符号付き整数の乗算では、レジスタ Rm の上半分とレジスタ Rs の下半分が乗算され、**32 **ビットの結果が Rd に生成されます。**

```
int _SmulHiLo_SW_SL(
  int Arg1,
  int Arg2
);
```

パラメータ

arg1

[in] **Rm の内容、積の第** 1 **項。**

Arg2

[in] **Rs の内容、積の第** 2 **項。**

戻り値

乗算結果の整数。

備考

この命令は、コンパイラで smultb アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulHiLo_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_SmulLo_SW_SL

_SmulLoHi_SW_SL

_SmulHi_SW_SL

_SmulLoHi_SW_SL

この ARM DSP **拡張、符号付き整数の乗算では、レジスタ Rm の下半分とレジスタ Rs の上半分が乗算され、**32 **ビットの結果が Rd に生成されます。**

```
int _SmulLoHi_SW_SL(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **Rm の内容、積の第** 1 **項。**

Arg2

[in] **Rs の内容、積の第** 2 **項。**

戻り値

乗算結果の整数。

備考

この命令は、コンパイラで smulbt アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulLoHi_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_SmulLo_SW_SL

_SmulHiLo_SW_SL

_SmulHi_SW_SL

_SmulWHi_SW_SL

この ARM DSP 拡張、符号付き整数の乗算では、**Rm** の 32 ビットのオペランドとレジスタ **Rs** の上半分の 16 ビットのソース オペランドに対して、32 × 16 ビットの乗算が実行されます。次に、48 ビットの積から上位 32 ビットが取得されます。

```
int _SmulWHi_SW_SL(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] 積の第 1 項、**Rm** の内容。

Arg2

[in] 積の第 2 項、**Rs** の内容。

戻り値

乗算結果の整数。

備考

この命令は、コンパイラで **smulwt** アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulWHi_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP 拡張組み込み関数

_SmulWLo_SW_SL

_SmulWLo_SW_SL

この ARM DSP 拡張、符号付き整数の乗算では、**Rm** の 32 ビット オペランドとレジスタ **Rs** の下半分の 16 ビット ソース オペランドに対して、32 x 16 ビットの乗算が実行されます。次に、48 ビットの積から高位 32 ビットが取得されます。

```
int _SmulWLo_SW_SL(
  int Arg1,
  int Arg2
);
```

パラメータ

arg1

[in] **Rm** の内容、積の第 1 項。

Arg2

[in] **Rs** の内容、積の第 2 項。

戻り値

乗算および累算結果の整数。

備考

この命令は、コンパイラで **smlawb** アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulWLo_SW_SL	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_SmulAddWHi_SW_SL

_AddSatInt

この ARM DSP **拡張演算では、飽和** add **命令が実行されます。これによって、レジスタ Rm と Rn が加算され、レジスタ Rd に結果が格納されます。加算でオーバーフローが発生した場合は、固定オーバーフロー ビット** 'Q' **に影響が及びます。**

```
int _AddSatInt(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **Rm の内容、和の第** 1 **項。**

Arg2

[in] **Rn の内容、和の第** 2 **項。**

戻り値

2 **進演算の結果。**

備考

この命令は、コンパイラで qadd アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_AddSatInt	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_SubSatInt

_DAddSatInt

この演算では、SAT(Rm + SAT(Rn*2)) が計算されます。つまり、最初に Rn の倍数が飽和され、結果が Rm に加算された後、その和が飽和されます。

```
int _DAddSatInt(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **この整数は、*Arg2* の倍数に加算されます。これは、Rm の値に相当します。**

Arg2

[in] **この整数は、2 倍される項です。これは、Rn の値に相当します。**

戻り値

2 進演算の結果。

備考

この命令は、コンパイラで qdadd アセンブリ命令に変換されます。

飽和は、2 倍演算または加算、あるいはその両方で行われます。飽和が 2 倍演算で行われ、加算では行われない場合、Q フラグが設定され、最終結果は飽和されません。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_DAddSatInt	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP 拡張組み込み関数

_DSubSatInt

_SubSatInt

この ARM DSP **拡張演算では、飽和減算命令が実行されます。Rn の値を Rm の値から減算し、その結果をレジスタ Rd に格納します。この演算は、減算でオーバーフローが発生した場合、固定オーバーフロー ビット** 'Q' **に影響します。**

```
int _SubSatInt(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **差の第** 1 **項、Rm の内容。**

Arg2

[in] **差の第** 2 **項、Rn の内容。**

戻り値

2 **進演算の結果。**

備考

この組み込み関数は、コンパイラで qsub アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SubSatInt	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_AddSatInt

_DSubSatInt

この演算は、Rn を 2 倍にして飽和した後、その結果を Rm から減算して飽和します。この演算は、減算でオーバーフローが発生した場合、固定オーバーフロー ビット 'Q' に影響します。

```
int _DSubSatInt(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] 2 **倍にした** *Arg2* **をこの整数から減算します。これは、Rm の値の内容です。**

Arg2

[in] **この整数を** 2 **倍にします。これは、Rn の値の内容です。**

戻り値

算術の実行結果の値。

備考

この命令は、コンパイラで qdsub アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_DSubSatInt	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP **拡張組み込み関数**

_DAddSatInt

_ReadCoProcessor

この命令では、指定したコプロセッサ レジスタから値が 2 つの ARM レジスタに転送されます。

```
__int64 _ReadCoProcessor(
  int Arg1
);
```

パラメータ

Arg1

[in] コプロセッサ番号、cp_num と同じ。

戻り値

コプロセッサ レジスタに保持される値。

備考

この命令は、コンパイラで ARM DSP 拡張プロセッサに対する **MRRC** アセンブリ命令と、ARM XScale プロセッサに対する **MRA** アセンブリ命令に変換されます。**MRA** は、**MRCC** 命令として逆アセンブルされます。

XScale マイクロプロセッサと DSP 拡張 ARM マイクロプロセッサがこの命令を実装する方法は次のように異なります。

- ARM XScale 実装の場合は、この命令によって次の処理が実行されます。
  - コプロセッサ レジスタから 64 ビットのデータを ARM レジスタに移動します。
  - 40 ビットのアキュムレータ値 (**acc0**) を 2 つのレジスタに移動します。
  - acc0 の値のビット [31:0] をレジスタ RdLo に移動します。
  - acc0 の値のビット [39:32] を 32 ビットに符号拡張し、これをレジスタ RdHi に移動します。
- ARM DSP 拡張実装の場合、この命令によって、コプロセッサから値が 2 つの汎用レジスタ **Rd** と **Rn** に転送されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_ReadCoProcessor	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM DSP 拡張組み込み関数
ARM XSCALE 組み込み関数
_WriteCoProcessor

_WriteCoProcessor

この命令では、指定した ARM レジスタの値がコプロセッサ レジスタに転送されます。

```
void _WriteCoProcessor(
  __int64 Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] コプロセッサに書き込まれる値。

Arg2

[in] コプロセッサ番号。この値は 0 です。

戻り値

なし。

備考

ARM XScale プロセッサの場合、この組み込み関数はコンパイラで **MAR** アセンブリ命令に変換されます。**MAR** は **MCRR** 命令として逆アセンブルされます。

ARM DSP 拡張プロセッサの場合、この組み込み関数はコンパイラで **MCRR** アセンブリ命令に変換されます。

ARM XScale マイクロプロセッサと ARM DSP 拡張マイクロプロセッサは、若干異なる 2 種類の方法でこの命令を実装します。

- ARM XScale の実装
  この命令はレジスタ RdLo の値を 40 ビット アキュムレータ (**acc0**) のビット [31:0] に移動し、レジスタ RdHi の値のビット [7:0] を acc0 のビット [39:32] に移動します。
- ARM DSP 拡張の実装
  この命令は、2 つの汎用レジスタ **Rd** および **Rn** の値をコプロセッサに転送します。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_WriteCoProcessor	<armintr.h>	ARM10、ARM-DSP

関連項目

参照情報

ARM XSCALE 組み込み関数
ARM DSP 拡張組み込み関数
_ReadCoProcessor

ARM XSCALE 組み込み関数

オーディオ処理アルゴリズムのパフォーマンスと精度を向上させるため、ARM を実装した Intel 80200(XScale) マイクロプロセッサには、デジタル シグナル処理 (DSP) コプロセッサが追加されています。

このコプロセッサには、40 ビットのアキュムレータと新しい命令が組み込まれています。

ARM XScale 命令セットの組み込み関数を実装するには、コードのコンパイル時に /QRxscale – XSCALE のターゲットの指定コンパイラ オプションを使用します。

組み込み関数によって実装されている XScale 命令には次のものがあります。

関数	ARM XScale 命令	説明
_SmulAdd_SL_ACC	**MIA**	レジスタ **Rs** の符号付き値とレジスタ **Rm** の符号付き値を乗算し、その結果を 40 ビット アキュムレータに加算します。
_SmulAddPack_2SW_ACC	**MIAPH**	パックされたハーフワード データに対して 16 × 16 の符号付き乗算を 2 回実行し、それらの結果を 1 つの 40 ビット アキュムレータに累算します。
_SmulAddLo_SW_ACC _SmulAddHi_SW_ACC _SmulAddLoHi_SW_ACC _SmulAddHiLo_SW_ACC	**MIAxy**	16 ビットの符号付き乗算を 1 回実行し、その結果を 1 つの 40 ビット アキュムレータに累算します。
_ReadCoProcessor	**MAR**	ARM レジスタの 64 ビット データをコプロセッサ レジスタに移動します。
_WriteCoProcessor	**MRA**	コプロセッサ レジスタから 64 ビット データを ARM レジスタに移動します。
_PreLoad	**PLD**	この命令はメモリ システムに対してこの直後に、指定したアドレスからのメモリ アクセスを行うことを示すヒントとして使用します。

関連項目

参照情報

ARM10 組み込み関数

ARM XSCALE 組み込み関数

_PreLoad

この命令は、ソフト プリロード命令です。つまり、この命令はメモリ システムに対してこの直後に、指定したアドレスからのメモリ アクセスを行うことを示します。

```
void _PreLoad(
  unsigned long* addr
);
```

パラメータ

addr

メモリ アクセスの位置。

戻り値

なし。

備考

この命令は、コンパイラで **PLD** アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_PreLoad	<armintr.h>	ARM10、ARM-DSP、ARM XSCALE

関連項目

参照情報

ARM XSCALE 組み込み関数

_SmulAdd_SL_ACC

この演算では、レジスタ Rs の符号付き値とレジスタ Rm の符号付き値が乗算され、その結果が 40 **ビットのアキュムレータ** acc0 **に追加されます。**

```
void _SmulAdd_SL_ACC(
  int Arg1,
  int Arg2
);
```

パラメータ

arg1

[in] **コプロセッサに書き込む Rs の値。**

Arg2

[in] **コプロセッサに書き込む Rm の値。**

戻り値

なし。

備考

この命令は、コンパイラで mia アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAdd_SL_ACC	<armintr.h>	ARM10、ARM-DSP、ARM XSCALE

関連項目

参照情報

ARM XSCALE **組み込み関数**

_SmulAddHi_SW_ACC

この命令では、Rm の上半分と Rs の上半分が乗算され、1 つの 40 ビット アキュムレータに結果が累積されます。

この命令では符号なし乗算はサポートされず、すべての引数が符号付きデータ値として解釈されます。

```
void _SmulAddHi_SW_ACC(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **Rm 内の値。**

Arg2

[in] **Rs 内の値。**

戻り値

なし。

備考

この命令は、コンパイラで miatt アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddHi_SW_ACC	<armintr.h>	ARM10、ARM-DSP、ARM XSCALE

関連項目

参照情報

ARM XSCALE **組み込み関数**

_SmulAddLo_SW_ACC

_SmulAddHiLo_SW_ACC

_SmulAddLoHi_SW_ACC

_SmulAddHiLo_SW_ACC

この ARM XScale 命令では、**Rm** の上半分と **Rs** の下半分が乗算され、その結果が 1 つの 40 ビット アキュムレータに累算されます。

この命令では符号なし乗算はサポートされず、すべての引数が符号付きデータ値として解釈されます。

```
void _SmulAddHiLo_SW_ACC(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **Rm** 内の値。

Arg2

[in] **Rs** 内の値。

戻り値

なし。

備考

この命令は、コンパイラで **miatb** アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddHiLo_SW_ACC	<armintr.h>	ARM10、ARM-DSP、ARM XSCALE

関連項目

参照情報

ARM XSCALE **組み込み関数**

_SmulAddLo_SW_ACC

_SmulAddHi_SW_ACC

_SmulAddLoHi_SW_ACC

_SmulAddLo_SW_ACC

この ARM XScale 命令では、**Rm** の下半分と **Rs** の下半分が乗算され、結果が 1 つの 40 ビット アキュムレータに累算されます。

この命令では符号なし乗算はサポートされず、すべての引数が符号付きデータ値として解釈されます。

```
void _SmulAddLo_SW_ACC(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **Rm** 内の値。

Arg2

[in] **Rs** 内の値。

戻り値

なし。

備考

この命令は、コンパイラで **miabb** アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddLo_SW_ACC	<armintr.h>	ARM10、ARM-DSP、ARM XSCALE

関連項目

参照情報

ARM XSCALE **組み込み関数**

_SmulAddHi_SW_ACC

_SmulAddHiLo_SW_ACC

_SmulAddLoHi_SW_ACC

_SmulAddLoHi_SW_ACC

この ARM XScale 命令では、**Rm** の下半分と **Rs** の上半分が乗算され、結果が 1 つの 40 ビット アキュムレータに累積されます。

この命令では符号なし乗算はサポートされず、すべての引数が符号付きデータ値として解釈されます。

```
void _SmulAddLoHi_SW_ACC(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **Rm** 内の値。

Arg2

[in] **Rs** 内の値。

戻り値

なし。

備考

この命令は、コンパイラで **miabt** アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddLoHi_SW_ACC	<armintr.h>	ARM10、ARM-DSP、ARM XSCALE

関連項目

参照情報

ARM DSP **拡張組み込み関数**

ARM XSCALE **組み込み関数**

_SmulAddLo_SW_ACC

_SmulAddHi_SW_ACC

_SmulAddHiLo_SW_ACC

_SmulAddPack_2SW_ACC

この ARM XScale 命令では、パックされたハーフワード データに対して 16 × 16 の符号付き乗算が 2 回実行され、それらの結果が 1 つの 40 ビット アキュムレータに累算されます。

```
void _SmulAddPack_2SW_ACC(
  int Arg1,
  int Arg2
);
```

パラメータ

Arg1

[in] **Rm** 内の値。

Arg2

[in] **Rs** 内の値。

戻り値

なし。

備考

最初に、この命令では **Rm** の値の下位 16 ビットと **Rs** の値の下位 16 ビットが乗算されます。

次に、**Rs** と **Rm** の上位 16 ビットの乗算が実行されます。

この命令は、両方の符号付き 32 ビット積を符号拡張して 40 ビット アキュムレータ (acc0) の値に加算します。

この命令は、コンパイラで **miaph** アセンブリ命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SmulAddPack_2SW_ACC	<armintr.h>	ARM10、ARM-DSP、ARM XSCALE

関連項目

参照情報

ARM XSCALE 組み込み関数

WMMX 組み込み関数

ワイヤレス MMX (WMMX) テクノロジを搭載した PXA270 プロセッサなど、ARM 準拠の Intel プロセッサは、最適化されたマルチメディア アプリケーションの開発を可能にする命令を備えています。これらの命令は、コプロセッサ命令として実装されます。

このテクノロジでは、単一命令複数データ (SIMD) 技法が採用されています。データ要素を並行処理することにより、SIMD 命令を使用して、メディアリッチなビット ストリームを持つアプリケーションのパフォーマンスを大きく向上させることができます。

ハードウェア命令、組み込み関数、データ型、およびレジスタの詳細については、『Intel Wireless MMX Technology Developer Guide』(番号 251793-001) (http://www.intel.com/design/pca/prodbref/251669_devguide.pdf) を参照してください。

このセクションのトピック

WMMX テクノロジの概要

WMMX テクノロジの概要を示します。

WMMX 算術組み込み関数

パックされた算術組み込み関数の参照テーブルを示します。

WMMX シフト組み込み関数

シフト組み込み関数の参照テーブルを示します。

WMMX 論理組み込み関数

論理組み込み関数の参照テーブルを示します。

WMMX 比較組み込み関数

比較組み込み関数の参照テーブルを示します。

WMMX パック/アンパック組み込み関数

パック/アンパック組み込み関数の参照テーブルを示します。

WMMX セット組み込み関数

セット組み込み関数の参照テーブルを示します。

WMMX の一般的なサポート組み込み関数

一般的なサポート組み込み関数の参照テーブルを示します。

外部リソース

Intel Wireless MMX Technology Developer Guide (番号 251793-001)

WMMX テクノロジの概要

Intel のワイヤレス MMX (WMMX) テクノロジは、Intel アーキテクチャ (IA) MMX 命令セットの拡張実装です。このテクノロジでは単一命令複数データ (SIMD) 技法を使用して、データ要素の並行処理によるマルチメディアおよび通信ソフトウェアの高速化を実現します。

WMMX 命令セットでは、57 個の命令コードと 64 ビットのデータ型が追加されています。さらに、16 個の 64 ビット MMX テクノロジ レジスタが搭載されており、各レジスタにはレジスタ名 wR0 ～ wR15 を使用して直接アドレス指定することができます。

MMX 命令、データ型、およびレジスタの追加情報と詳細については、『Intel Wireless MMX Technology Developer Guide』(番号 251793-001) を参照してください。このガイドはオンラインの Intel Hardware Design (英語) サイトから入手できます。

新規レジスタ

WMMX テクノロジの組み込み関数では、64 ビット長 (0 ～ 63) の 16 個のレジスタ (wR0 ～ wR15) を使用できます。

これらの新しいデータ レジスタでは、データ要素を並行して処理することができます。各レジスタは複数のデータ要素を格納できるため、プロセッサは複数のデータ要素を同時に処理できます。この処理機能は、SIMD 処理とも呼ばれます。C/C++ コンパイラで SIMD 処理を有効にするには、新規レジスタの拡張サイズを利用するよう新しいデータ型を定義します。

組み込み関数を使用すると、アセンブリ言語ではなく、C 関数呼び出しと変数の構文によるコーディングを行うことができます。新しい拡張セットの計算およびデータ操作の各命令に対応する C 組み込み関数があり、この関数によって各命令が直接実装されます。これにより、レジスタの管理とアセンブリのプログラミングが不要になります。さらに、コンパイラで命令のスケジュールが最適化されるため、実行ファイルの高速化が実現します。

__m64 データ型

__m64 データ型は、WMMX レジスタの内容を表すために使用されます。WMMX レジスタ は、WMMX テクノロジの組み込み関数で使用されるレジスタです。**__m64** データ型は、8 つの 8 ビット値、4 つの 16 ビット値、2 つの 32 ビット値、または 1 つの 64 ビット値を保持できます。

新しいデータ型の使用上のガイドライン

新しい **__m64** データ型は ANSI C の基本データ型ではないため、次の使用制限に従う必要があります。

- **__m64** は、戻り値またはパラメータとして割り当ての左側でのみ使用します。このデータ型は他の算術式 (" + "、" " など) と一緒に使用できません。
- 和集合などの集合体のオブジェクトとして **__m64** を使用して、バイト要素および構造にアクセスします。
- **__m64** は、WMMX 組み込み関数でのみ使用します。

データ配置

多くの WMMX 組み込み関数には、データ配置の必要条件があります。これらの組み込み関数の使用時にデータが適切に配置されていない場合、プログラムはそのプログラムによる処理を必要とする例外をスローします。さもなければ、プログラムは失敗します。WMMX 組み込み関数の使用をサポートするには、ユーザーはより積極的な役割を果たして配置の問題が必ず適切に解決されるようにする必要があります。

詳細については、「デバイス コンパイラのデータ配置に関する問題」を参照してください。

関連項目

その他のリソース

WMMX 組み込み関数

WMMX 算術組み込み関数

次の表に示した組み込み関数は、パックされた算術演算を実行します。

組み込み関数名	演算	WMMX 命令	引数および結果値/ビット
_mm_add_pi16	加算	WADDH	4/16, 4/16
_mm_add_pi32	加算	WADDW	2/32, 2/32
_mm_add_pi8	加算	WADDB	8/8, 8/8
_mm_adds_pi16	加算、符号付き	WADDHSS	4/16, 4/16
_mm_adds_pi32	加算、符号付き	WADDWSS	2/32, 2/32
_mm_adds_pi8	加算、符号付き	WADDBSS	8/8, 8/8
_mm_adds_pu16	加算	WADDHUS	4/16, 4/16
_mm_adds_pu32	加算	WADDWUS	2/32, 2/32
_mm_adds_pu8	加算	WADDBUS	8/8, 8/8
_mm_sub_pi16	減算	WSUBH	4/16, 4/16
_mm_sub_pi32	減算	WSUBW	2/32, 2/32
_mm_sub_pi8	減算	WSUBB	8/8, 8/8
_mm_subs_pi16	減算、符号付き	WSUBHSS	4/16, 4/16
_mm_subs_pi32	減算、符号付き	WSUBWSS	2/32, 2/32
_mm_subs_pi8	減算、符号付き	WSUBBSS	8/8, 8/8
_mm_subs_pu16	減算	WSUBHS	4/16, 4/16
_mm_subs_pu32	減算	WSUBWSS	2/32, 2/32
_mm_subs_pu8	減算	WSUBBUS	8/8, 8/8
_mm_madd_pi16	乗算	WMADDS	4/16, 2/32
_mm_madd_pu16	乗算	WMADDU	4/16, 2/32
_mm_mulhi_pi16	乗算、符号付き	WMULSH	4/16、4/16 (高)
_mm_mulhi_pu16	乗算、符号付き	WMULUH	4/16、4/16 (高)
_mm_mullo_pi16	乗算	WMULSL/WMULUL	4/16、4/16 (低)
_mm_mac_pi16	乗算と累算、符号付き	WMACS	4/16, 4/16

_mm_mac_pu16	**乗算と累算**	WMACU	4/16, 4/16
_mm_macz_pi16	**乗算と累算、符号付き**	WMACSZ	4/16, 4/16
_mm_macz_pu16	**乗算と累算**	WMACUZ	4/16, 4/16
_mm_acc_pu16	**累算**	WACCH	4/16, 1/16
_mm_acc_pu32	**累算**	WACCW	2/32. 2/32
_mm_acc_pu8	**累算**	WACCB	8/8, 1/8
_mm_mia_si64	**乗算と累算、符号付き**	TMIA	1/64
_mm_miabb_si64	**乗算と累算、符号付き**	TMIABB	1/64
_mm_miabt_si64	**乗算と累算、符号付き**	TMIABT	1/64
_mm_miaph_si64	**乗算と累算、符号付き**	TMIAPH	1/64
_mm_miatb_si64	**乗算と累算、符号付き**	TMIATB	1/64
_mm_miatt_si64	**乗算と累算、符号付き**	TMIATT	1/64

関連項目

その他のリソース

WMMX **組み込み関数**

WMMX シフト組み込み関数

次の表に示した組み込み関数は、シフト演算を実行します。

組み込み関数名	シフトの方向	シフトの種類	WMMX 命令
_mm_sll_pi16	左	論理	WSLLH
_mm_slli_pi16	左	論理	複合
_mm_sll_pi32	左	論理	WSLLW
_mm_slli_pi32	左	論理	複合
_mm_sll_si64	左	論理	WSLLD
_mm_slli_si64	左	論理	複合
_mm_sra_pi16	右	算術	WSRAH
_mm_srai_pi16	右	算術	複合
_mm_sra_pi32	右	算術	WSRAW
_mm_srai_pi32	右	算術	複合
_mm_sra_pi64	右	算術	WSRAD
_mm_srai_pi64	右	算術	複合
_mm_srl_pi16	右	論理	WSRLH
_mm_srli_pi16	右	論理	複合
_mm_srl_pi32	右	論理	WSRLW
_mm_srli_pi32	右	論理	複合
_mm_srl_si64	右	論理	WSRLD
_mm_srli_si64	右	論理	複合
_mm_ror_pi16	右回転	論理	WRORH
_mm_rori_pi16	右回転	論理	WRORW
_mm_ror_pi32	右回転	論理	WRORD
_mm_rori_pi32	右回転	論理	複合
_mm_ror_si64	右回転	論理	複合
_mm_rori_si64	右回転	論理	複合

関連項目

その他のリソース

WMMX **組み込み関数**

WMMX 論理組み込み関数

次の表に示した組み込み関数は、論理演算を実行します。

組み込み関数名	演算	WMMX 命令
_mm_and_si64	Bitwise AND	WAND
_mm_andnot_si64	Logical NOT	WANDN
_mm_or_si64	Bitwise OR	WOR
_mm_xor_si64	Bitwise exclusive OR	WXOR

関連項目

その他のリソース

WMMX 組み込み関数

WMMX 比較組み込み関数

次の表に示した組み込み関数は、比較を実行します。

組み込み関数名	比較	要素数	要素のビット サイズ	WMMX 命令
_mm_cmpeq_pi8	等しい	8	8	WCMPEQB
_mm_cmpeq_pi16	等しい	4	16	WCMPEQH
_mm_cmpeq_pi32	等しい	2	32	WCMPEQW
_mm_cmpgt_pi8	より大きい、符号付き	8	8	WCMPGTSB
_mm_cmpgt_pu8	より大きい、符号なし	8	8	WCMPGTUB
_mm_cmpgt_pi16	より大きい、符号付き	4	16	WCMPGTSH
_mm_cmpgt_pu16	より大きい、符号なし	4	16	WCMPGTUH
_mm_cmpgt_pi32	より大きい、符号付き	2	32	WCMPGTSW
_mm_cmpgt_pu32	より大きい、符号なし	2	32	WCMPGTUW

関連項目

その他のリソース

WMMX 組み込み関数

WMMX パック/アンパック組み込み関数

次の表に示した組み込み関数は、パックとアンパック処理を実行します。

組み込み関数名	処理	WMMX 命令
_mm_packs_pi16	パック、符号付き、飽和。	WPACKHSS
_mm_packs_pi32	パック、符号付き、飽和。	WPACKWSS
_mm_packs_pu16	パック、飽和。	WPACKHUS
_mm_unpackhi_pi8	インターリーブ。	WUNPCKIHB
_mm_unpackhi_pi16	インターリーブ。	WUNPCKIHH
_mm_unpackhi_pi32	インターリーブ。	WUNPCKIHW
_mm_unpacklo_pi8	インターリーブ。	WUNPCKILB
_mm_unpacklo_pi16	インターリーブ。	WUNPCKILH
_mm_unpacklo_pi32	インターリーブ。	WUNPCKILW
_mm_packs_si64	パック、符号付き、飽和。	WPACKDSS
_mm_packs_su64	パック、飽和。	WPACKDUS
_mm_packs_pu32	パック、飽和。	WPACKWUS
_mm_unpackeh_pi8	インターリーブ、符号付き、拡張。	WUNPCKEHSB
_mm_unpackeh_pi16	インターリーブ、符号付き、拡張。	WUNPCKEHSH
_mm_unpackeh_pi32	インターリーブ、符号付き、拡張。	WUNPCKEHSW
_mm_unpackeh_pu8	インターリーブ、符号なし、拡張。	WUNPCKEHUB
_mm_unpackeh_pu16	インターリーブ、符号なし、拡張。	WUNPCKEHUH
_mm_unpackeh_pu32	インターリーブ、符号なし、拡張。	WUNPCKEHUW
_mm_unpackel_pi8	インターリーブ、符号付き、拡張。	WUNPCKELSB
_mm_unpackel_pi16	インターリーブ、符号付き、拡張。	WUNPCKELSH
_mm_unpackel_pi32	インターリーブ、符号付き、拡張。	WUNPCKELSW
_mm_unpackel_pu8	インターリーブ、拡張。	WUNPCKELUB
_mm_unpackel_pu16	インターリーブ、拡張。	WUNPCKELUH
_mm_unpackel_pu32	インターリーブ、拡張。	WUNPCKELUW

関連項目

その他のリソース

WMMX **組み込み関数**

WMMX セット組み込み関数

次の表に示した組み込み関数は、複合命令から成ります。

組み込み関数名	処理	要素数	符号付き	逆順	WMMX 命令
_mm_setzero_si64	ゼロに設定	1	いいえ	いいえ	WZERO
_mm_set_pi32	整数値を設定	2	いいえ	いいえ	複合
_mm_set_pi16	整数値を設定	4	いいえ	いいえ	複合
_mm_set_pi8	整数値を設定	8	いいえ	いいえ	複合
_mm_set1_pi32	整数値を設定	2	はい	いいえ	TBCSTW
_mm_set1_pi16	整数値を設定	4	はい	いいえ	TBCSTH
_mm_set1_pi8	整数値を設定	8	はい	いいえ	TBCSTB
_mm_setr_pi32	整数値を設定	2	いいえ	はい	複合
_mm_setr_pi16	整数値を設定	4	いいえ	はい	複合
_mm_setr_pi8	整数値を設定	8	いいえ	はい	複合
_mm_setwcx	コントロール レジスタを設定	--	--	--	TMCR
_mm_getwcx	コントロール レジスタを取得	--	--	--	TMRC

関連項目

その他のリソース

WMMX 組み込み関数

WMMX の一般的なサポート組み込み関数

次の表に記載されている組み込み関数は、一般的なサポート操作を提供します。

組み込み関数名	操作	WMMX 命令
_mm_extract_pi8	8 バイトの抽出	TEXTRMSB
_mm_extract_pi16	4 ハーフ ワードの抽出	TEXTRMSH
_mm_extract_pi32	2 ワードの抽出	TEXTRMSW
_mm_extract_pu8	8 バイトの抽出	TEXTRMUB
_mm_extract_pu16	4 ハーフ ワードの抽出	TEXTRMUL
_mm_extract_pu32	2 ワードの抽出	TEXTRMUW
_mm_insert_pi8	1 バイトの挿入	TINSRB
_mm_insert_pi16	1 ハーフ ワードの挿入	TINSRH
_mm_insert_pi32	1 ワードの挿入	TINSRW
_mm_max_pi8	最大値の計算	WMAXSB
_mm_max_pi16	最大値の計算	WMAXSH
_mm_max_pi32	最大値の計算	WMAXUB
_mm_max_pu8	最大値の計算、符号なし	WMAXUB
_mm_max_pu16	最大値の計算、符号なし	WMAXUH
_mm_max_pu32	最大値の計算、符号なし	WMAXUW
_mm_min_pi8	最小値の計算	WMINXSB
_mm_min_pi16	最小値の計算	WMINSH
_mm_min_pi32	最小値の計算	WMINUB
_mm_min_pu8	最小値の計算、符号なし	WMINUB
_mm_min_pu16	最小値の計算、符号なし	WMINUH
_mm_min_pu32	最小値の計算、符号なし	WMINUW
_mm_movemask_pi8	8 ビット マスクの作成	TMOVMSKB
_mm_movemask_pi16	4 ハーフ ワード マスクの作成	TMOVMSKH

_mm_movemask_pi32	2 ワード マスクの作成	TMOVMSKW
_mm_shuffle_pi8	4 ワードの組み合わせのリターン	WSHUFH
_mm_avg_pu8	四捨五入した平均値の計算	WAV2BR
_mmavg_pu16	四捨五入した平均値の計算	WAV2HR
_mmavg2_pu8	平均値の計算	WAVG2B
_mmavg2_pu16	平均値の計算	WAVG2H
_mm_sada_pu8	差分の絶対和の計算	WSADB
_mm_sada_pu16	差分の絶対和の計算	WSADH
_mm_align_si64	64 ビット値の抽出	WALIGN/WALIGNR
_mm_cvtsi32_si64	int から 64 ビット __m64 オブジェクトへの変換	TMCRR
_mm_cvtsi64_si32	__m64 オブジェクトから int への変換	TMRRC
_mm_empty	MM 状態の解除	---

関連項目

その他のリソース

WMMX 組み込み関数

ARM コンパイラ オプション

次の表に、ARM マイクロプロセッサのコンパイラ オプションを示します。

オプション	説明
/QRArch – ターゲットのアーキテクチャの指定	ターゲットの ARM マイクロプロセッサを指定します。
/QRimplicit-import – ヘルパーのインポートの無効化	コンパイラのヘルパー関数に静的にリンクします。
/QRinterwork-return – 相互動作の有効化	ARM モードと Thumb モードの間で相互動作するコードを生成します。
/QRxscale – XSCALE のターゲットの指定	Intel XScale マイクロプロセッサ用のコードを生成します。
/QRxscalesched	XScale パイプラインに対して内部スケジューラの最適化を有効にするコードを生成します。
/QRthumb	Thumb モード用のコードを生成します。

関連項目

その他のリソース

ARM ファミリ プロセッサ

/QRArch – ターゲットのアーキテクチャの指定

このオプションは、コンパイラのターゲットとなる ARM アーキテクチャを指定します。このオプションの使い方は次のとおりです。

```
/Qrarch{4|5} [T]
```

このスイッチには、次のオプションを使用できます。

4

ARM4 アーキテクチャを示します。

5

ARM5 アーキテクチャを示します。

T

ARM と Thumb の相互動作命令をサポートするアーキテクチャを示します。

関連項目

その他のリソース

ARM ファミリ プロセッサ

/QRimplicit-import – ヘルパーのインポートの無効化

このオプションは、静的にリンクされたヘルパー関数を使用するようコンパイラに強制します。

コンパイラのヘルパー関数は通常、DLL 内にあります。これらの関数を呼び出すときのサンク オーバーヘッドを減らすために、コンパイラでは dllimport 呼び出し機構が使用されます。OS またはドライバ コードがヘルパー関数と静的にリンクされている場合は、**/QRimplicit-import** フラグを使用して、通常の呼び出し機構を使用するようコンパイラに強制します。

関連項目

その他のリソース

ARM コーリング シーケンスの仕様

/QRinterwork-return – 相互動作の有効化

このオプションは、プログラムで ARM **モードまたは** Thumb **モードから関数を呼び出し、正しく結果を返すことができるようにするコードの生成をコンパイラに命令します。**

関連項目
概念
THUMB **対応** ARM **の実装**

/QRxscale – XSCALE のターゲットの指定

このオプションは、Intel XScale プロセッサで実行するコードをコンパイラが生成するように指定します。このオプションによって XScale MAC 組み込み関数が有効になり、スケジューラで XScale パイプラインが最適化されます。

関連項目

参照情報

ARM XSCALE 組み込み関数

/QRxscalesched

このオプションを使用すると、生成されたコード内の命令の順序を再度並べ替えて、XScale 固有の方法で内部の命令実行スケジュールを最適化できます。

このオプションで提供される機能は、/QRxscale – XSCALE のターゲットの指定オプションの機能の一部です。

関連項目

参照情報

ARM XSCALE 組み込み関数

/QRthumb

このオプションによって、ARM Thumb 命令セットが有効になります。

関連項目
概念
THUMB 対応 ARM の実装

ARM コーリング シーケンスの仕様

ARM デバイス コンパイラの ARM コーリング シーケンスの仕様には、ARM ファミリのマイクロプロセッサ用のコンパイラとアセンブリ言語プログラムの開発のガイドラインが示されています。

このセクションのトピック

ARM レジスタ

割り当てられているレジスタの役割を示します。

ARM スタック フレームのレイアウト

ARM マイクロプロセッサでスタック レイアウトが指定されるようすを説明します。

ARM プロローグおよびエピローグ

構造化例外処理に必要なプロローグおよびエピローグのコード セグメントについて説明します。

THUMB 対応 ARM の実装

ARM と Thumb の相互動作の実装方法について説明します。

ARM アセンブラ

プロローグとエピローグ、ディレクティブ、およびコマンドライン オプションの ARM アセンブラ マクロについて説明します。

関連セクション

ARM コンパイラ オプション

ARM コンパイラのコンパイラ オプションに関する参照情報を提供します。

ARM マイクロプロセッサの組み込み関数

ARM コンパイラでのみサポートされる組み込み関数に関する参照情報を提供します。

デスクトップ コンパイラとデバイス コンパイラの違い

デスクトップ コンパイラとデバイス コンパイラの実装における重要な相違点について説明します。

ARM レジスタ

ARM マイクロプロセッサには、16 個の汎用レジスタがあります。

THUMB には、8 個の汎用レジスタ **R0 ～ R7** があり、ハイ レジスタ **R8 ～ R15** にアクセスできます。

次の表に、レジスタに割り当てられた役割を示します。

レジスタ	説明
R0	引数 1、戻り値 一時レジスタ
R1	引数 2、戻り値が double または int の場合は 2 番目の 32 ビット 一時レジスタ
R2 ～ R3	引数 一時レジスタ
R4 ～ R10	**R7** は THUMB フレーム ポインタ 永続レジスタ
R11	ARM フレーム ポインタ 永続レジスタ
R12	一時レジスタ
R13	スタック ポインタ 永続レジスタ
R14	リンク レジスタ 永続レジスタ
R15	プログラム カウンタ

レジスタ **R0 ～ R3** は、入力引数の最初の 4 ワードを保持します。マイクロプロセッサは、残りの引数を、**R0 ～ R3** がスピルしない呼び出し関数の引数作成領域で構築します。

次の表に、ベクトル浮動小数点 (VFP) と WMMX 用にあらかじめ定義されたその他のレジスタを示します。

レジスタ	説明
s0 ～ s32	VFP 単精度レジスタ
d0 ～ d16	VFP 倍精度レジスタ
fpsid	VFP システム ID レジスタ
fpscr	VFP ステータスおよびコントロール レジスタ
fpexc	VFP 例外レジスタ
wr0 ～ wr16	WMMX SIMD データ レジスタ

wc0 ～ wc16	WMMX ステータスおよびコントロール レジスタ
wcid	WMMX コプロセッサ ID レジスタ、**wc0** と同じ
wcon	WMMX コントロール レジスタ、**wc1** と同じ
wcssf	WMMX 飽和 SIMD フラグ、**wc2** と同じ
wcasf	WMMX 算術 SIMD フラグ、**wc3** と同じ
wcgr0 ～ wcgr3	WMMX コントロール汎用レジスタ、**wc8 ～ wc11** と同じ

関連項目

参照情報

ARM アセンブラ マクロ

概念

ARM スタック フレームのレイアウト

ARM スタック フレームのレイアウト

次の一覧は、ARM マイクロプロセッサのスタック フレームのレイアウトに関する詳細情報を示しています。

- **レジスタ保存領域** (RSA) には、関数によって使用される永続レジスタの保存値が格納されます。また、関数のリターン アドレスも格納されます。
- **ローカルおよび一時**領域は、ローカル変数およびコンパイラによって生成された一時レジスタに割り当てられるスタック領域を表します。
- スタックの最上位の最初の 4 ワードには、**R0-R3** に渡される値を格納することができます。これらのうちのいずれかの値がない場合もあります。レジスタに関数全体の引数を格納できない場合や、引数のアドレスが使用中の場合は、値を **R0-R3** に保存する必要があります。ルーチンで引数の最初の 4 ワードの保存領域が必要な場合は、呼び出された関数スタックの最上位に保存領域を作成し、初期化します。
  レジスタに引数の有効範囲の引数が格納されている場合、スタック フレーム内には引数に関連付けられている保存領域はありません。

ARM のフレーム ポインタとスタック ポインタ

フレーム ポインタは、レジスタ ディスプレイスメント アドレッシング オフセットを指定するビット フィールドのサイズが限られているという問題を緩和する上で役立ちます。フレーム ポインタは通常、スタック フレームの RSA またはローカルおよび一時領域内の固定フレーム オフセットをポイントしますが、フレーム内の他のオフセットをポイントすることもできます。大規模なスタック フレーム内のデータにより効率的にアクセスするために、ルーチンは別のフレーム ポインタを確立することができます。

- 永続レジスタを保存したり、4 ワードを超えるローカルまたは出力引数領域に領域を割り当てる必要がない限り、ルーチンでスタック フレームを設定する必要はありません。スタック ポインタとフレーム ポインタのアドレスは、4 バイト境界に配置されます。
- ルーチンに **alloca()** ローカルがある場合、ARM の仕様では、入力引数およびローカルにアクセスするには別のフレーム ポインタ レジスタが必要です。
  R11 は ARM 用に割り当てられたフレーム ポインタで、**R7** は THUMB 用に割り当てられたフレーム ポインタです。
  リーフ ルーチンでは、任意の空き整数レジスタをフレーム ポインタとして使用できます。非リーフ ルーチンでは永続レジスタを使用する必要があります。ルーチンでは、プロローグとエピローグ間でフレーム ポインタ レジスタを変更することはできません。
- ルーチンで **alloca()** を使用する場合、**alloca()** 領域より下位のアドレスにあるフレーム内のすべてが **R13** に関して参照され、**alloca()** の呼び出し時には定義された値は含まれません。そのため、**alloca()** 操作ではスタック フレームのこの部分をコピーする必要はなく、データの再配置の問題は生じません。
  alloca() 領域より上位のアドレスにあるフレーム内のすべては、フレーム ポインタ (ARM の場合は **R11**、THUMB の場合は **R7**) に関して参照されます。

関連項目

概念

ARM レジスタ

ARM プロローグおよびエピローグ

ARM マイクロプロセッサ用に構造化例外処理 (SEH) を実装するには、ARM プロローグおよびエピローグのコード セグメントが必要です。ARM プロローグはルーチンのスタック フレームを設定するコード セグメントです。エピローグでは、設定したルーチン フレームが削除され、ルーチンから戻ります。

このセクションのトピック

ARM プロローグ

ARM プロローグの必要条件について説明し、例を示します。

ARM エピローグ

ARM エピローグの必要条件について説明し、例を示します。

THUMB プロローグ

THUMB プロローグの必要条件について説明し、例を示します。

THUMB エピローグ

THUMB エピローグの必要条件について説明し、例を示します。

ARM アセンブラ マクロ

ARM および THUMB のプロローグおよびエピローグ シーケンスの定義に使用する ARM アセンブラ マクロに関する参照情報を提供します。

関連セクション

ARM コーリング シーケンスの仕様

ARM コーリング シーケンスの仕様に関する参照情報へのリンクを提供します。

RISC 環境の SEH

ARM などの RISC 環境での構造化例外処理の実装について説明します。

ARM プロローグ

ARM プロローグは 3 つの部分で構成されます。すべての部分は連続して記述し、間に命令を挿入しません。このガイドラインに従ってプロローグを記述すると、**仮想アンワインダ**でプロローグを逆実行できます。

ARM プロローグに必要な 3 つの部分は次の一覧のとおりです。

1. **R0**、**R1**、**R2**、および **R3** からの入力引数値を引数のホーム ロケーションにプッシュし、**R13** を新しいスタック トップに更新する 0 または 1 つの命令のシーケンス。
   このシーケンスでは、引数レジスタの保存に続いて、すべての永続レジスタがスタック フレームの一番上から降順で保存されます。
2. フレーム ポインタを確立する場合に、フレーム ポインタを設定する 1 つ以上の命令のシーケンス。
   プロローグでは、初期レジスタの保存前にスタック ポインタ **R13** が **R12** にコピーされ、**R12** を使用してフレーム ポインタ **R11** の値が計算されます。
3. **R13** から 4 バイトの整列オフセットを差し引くことによって、ローカル変数、コンパイラによって生成される一時レジスタ、および引数作成領域用に残りのスタック フレーム領域を割り当てる 0 個以上の命令のシーケンス。
   オフセットが即値フィールドに表すには大きすぎる場合、プロローグではスクラッチ レジスタ **R12** を使用してオフセットが保持されます。この場合、プロローグでは別の命令によって **R12** に値が設定されます。

例

ARM プロローグを構築するいくつかの例を次に示します。

- **R11** にフレームがある ARM プロローグ

```
MOV    r12, r13        ; Save stack on entry if needed.
STMDB  r13!, {r0-r3}    ; As needed
STMDB  r13!, {r4-r12, r14}  ; As needed
SUB    r11, r12, #16    ; Sets frame past args
<stack link if needed>
```

- フレームがない ARM プロローグ

```
MOV    r12, r13
STMDB  r13!, {r0-r3}        ; As needed
STMDB  r13! {[r4-r12,]|[r13,]r14}  ; As needed
<stack link if needed>
<note: r12 is not used if the stack (r13) is the first register saved>
```

関連項目

概念

RISC 環境の SEH
ARM エピローグ

ARM エピローグ

ARM エピローグは、以下の操作を行う一連の命令です。

- 保存された永続レジスタを復元します。
- スタック ポインタを関数開始時のその値にリセットします。
- 関数の呼び出し元関数に戻ります。

次の一覧は、エピローグの実装におけるガイドラインを示しています。

- すべての部分は連続して記述し、間に命令を挿入しません。
- フレーム ポインタが設定されている場合、エピローグはそのフレームをベースとして使用するただ 1 つの命令となり、プログラム カウンタやスタックを含むすべての不揮発性レジスタを更新します。
  フレームが設定されていない場合、エピローグはリンクされていないスタックで構成され、必要に応じて、複数のレジスタを復元する命令や、リンク レジスタ **R14** をプログラム カウンタにコピーする命令がその後に続きます。
  関数でフレーム ポインタ (ARM エピローグに対して **R11** の値を持つ) が確立されている場合、その関数では、プロローグの最後の命令が完了してからエピローグの最初の命令の実行が開始されるまでの間、ポインタの値を変更できません。
  関数でフレーム ポインタ (**R13** の値を持つ) が確立されていない場合、その関数では、プロローグの最後の命令が完了してからエピローグの最初の命令の実行が開始されるまでの間、スタック ポインタを変更できません。
- 不揮発性レジスタを変更しない、相互動作しないルーチンでは、プログラム カウンタへのリンク レジスタのコピーだけがエピローグに格納されます。
- 最後の命令が別のルーチンへの分岐であるルーチンでは、不揮発性レジスタを変更していない場合、空のエピローグを持つことができます。
- 常に **R13** の値を持つスタック ポインタに格納されているアドレスは、レジスタ保存領域の復元されていないレジスタ値の最下位アドレスを超えることはできません。
  これにより、永続レジスタに保存されている値が、コンテキスト スイッチあるいは、プロローグまたはエピローグの実行中に発生する可能性のあるその他の非同期的なイベントによって破損するのを防ぐことができます。

例

次の例は、さまざまな ARM エピローグを示しています。

- **R11** にフレームがある ARM エピローグ。

```
<no stack unlink>
LDMDB   r11, {r4-r11, r13, r15}
```

- フレームがない ARM エピローグ。

```
<stack unlink if needed>
LDMIA   r13, {r4-R11, r13, r15}
```

- 相互動作のリターンがある ARM エピローグ。

```
<stack unlink if needed>
LDMIA    r13, {r4-r11, r13, LR}
BX       LR
```

関連項目

概念

RISC 環境の SEH
ARM プロローグ

THUMB プロローグ

THUMB プロローグには、次のような特定の部分があります。すべての部分は連続して記述し、間に命令を挿入しません。

- **R0**、**R1**、**R2**、および **R3** の入力引数値を引数のホーム ロケーションにプッシュし、**R13** を新しいスタック トップに更新する 0 または 1 つの命令のシーケンス。
  関数で **R8**、**R9**、**R10** および **R11** などのハイ レジスタを使用しない場合、この命令シーケンスによって **R4** ～ **R7** またはリンク レジスタ **R14** がスタックにプッシュされます。
  ルーチンが、保存されているハイ レジスタのないリーフの場合、関数によってリンク レジスタがプッシュされることはありません。
- **R13** から整列オフセットを差し引くことによって、ローカル変数、コンパイラによって生成される一時レジスタ、および引数作成領域用に残りのスタック フレーム領域を割り当てる 0 個以上の命令のシーケンス。
- フレーム ポインタを確立する場合に、フレーム ポインタを設定する 1 つの命令。
  関数は、スタックをリンクした直後、**R13** のスタック ポインタの値を **R7** にコピーします。

例

次の例は、さまざまな THUMB プロローグを示しています。

- フレームがない THUMB プロローグ。

```
PUSH    {r0-r3}    ; As needed
PUSH    {r4-r7, LR}  ; As needed
SUB    SP, SP, #4  ; Stack link (as needed)
```

- **R7** にフレームがある THUMB プロローグ。

```
PUSH    {r0-r3}      ; As needed
PUSH    {r4-r7, LR}    ; As needed
SUB    SP, SP, #4    ; Stack link (as needed)
MOV    r7, SP       ; Set frame
```

- 相互動作するリターンがある THUMB プロローグ。

```
PUSH    {r4-r7, LR}
```

- ハイ レジスタを保存する THUMB プロローグ。

```
PUSH    {r0-r3}      ; Save r0-r3 as needed
PUSH    {LR}
BL    __savegpr_9    ; Routine for saving {r4-r11}
SUB    SP, SP, #4    ; Stack link (as needed)
```

- 大規模なスタック フレームがある THUMB プロローグ。

```
PUSH    {r7}
LDR    r7, [PC, #20]
NEG    r7, r7
ADD    SP, r7
```

関連項目

概念

RISC 環境の SEH
THUMB エピローグ

THUMB エピローグ

THUMB エピローグは、以下の操作を行う一連の命令です。

- 保存された永続レジスタを復元します。
- スタック ポインタを関数開始時のその値にリセットします。
- 関数の呼び出し元関数に戻ります。

次の一覧は、エピローグに関する重要情報です。

- フレーム ポインタがプロローグによって設定されている場合、エピローグはフレーム ポインタからスタック ポインタを復元します。エピローグは 1 つの **mov** 命令を使用して **R7** を **R13** にコピーします。このコピーによって、プロローグがスタックをリンクした直後の値に **R13** が復元されます。スタックのリンクを解除すると、**R13** が復元されます。
- {**R4**-**R7**} を復元する場合、およびハイ レジスタを復元しない場合は、{**R4**-**R7**} をポップする命令をエピローグに含める必要があります。この命令によって、スタック ポインタが更新されます。**アンワインダ**によってハイ レジスタが復元されると、**R4**-**R7** も復元されます。
- エピローグには、スタックから **R3** へのリンク レジスタをポップするゼロまたは 2 つの命令のシーケンスが含まれます。
  ルーチンがリーフの場合、またはルーチンでハイ レジスタが復元される場合、エピローグではこのシーケンスが省略されます。
  エピローグでハイ レジスタを復元する場合は、分岐およびリンクを使用して、**R4** から **R7** までとリンク レジスタを復元するレジスタ復元ルーチンが呼び出されます。また、プロローグが {**R0**-**R3**} を保存した直後に保持していた値にスタック ポインタが復元されます。{**R0**-**R3**} が保存されていた場合は、関数の開始時にスタック ポインタが保持していた値に復元されます。
- {**R4**-**R7**,**LR**} 以外のレジスタを保存する必要のない相互動作しないルーチンでは、エピローグは pop {**R4**-**R7**, **PC**} で終了します。{**R4**-**R7**} を保存する必要がない場合は、pop {**PC**} で終了します。相互動作しないルーチンで他のレジスタを保存する必要がある場合、エピローグは **PC** への **R3** のコピーで終了します。
  相互動作するルーチンでは、エピローグは **R3** への分岐と交換 (**BX**) で終了します。

例

次の例は、さまざまな THUMB エピローグを示しています。

- フレームがない THUMB エピローグ。

```
ADD    SP, SP, #4    ; Stack link (as needed)
POP    {r4-r7}
POP    {r3}        ; POP link register into r3
ADD    SP, SP, #16    ; POP r0-r3 as needed
MOV    PC, r3
```

- **R7** にフレームがある THUMB エピローグ。

```
MOV    r13, r7
ADD    SP, SP, #4    ; Stack unlink (as needed)
POP    {r4-r7}
POP    {r3}        ; POP link register into r3
ADD    SP, SP, #16    ; POP r0-r3 as needed
MOV    PC, r3
```

- ハイ レジスタを復元する THUMB エピローグ。

```
ADD    SP, SP, #4    ; Stack link (as needed)
BL    __restgpr_9
ADD    SP, SP, #16    ; POP r0-r3 as needed
BX    LR
```

- 大規模なスタック フレームがある THUMB エピローグ。

```
LDR    r7, [PC, #4]
ADD    SP, r7
POP    {r7}
MOV    PC, LR
```

関連項目

概念

RISC 環境の SEH

THUMB プロローグ

ARM アセンブラ マクロ

プロローグおよびエピローグのコード セグメントを実装するには、ARM アセンブラ レベルのマクロが必要です。

ARM マイクロプロセッサでは、以下のアセンブラ レベルのマクロが使用可能です。

マクロ	説明
ALTERNATE_ENTRY (ARM)	ルーチンの代替エントリを宣言します。
END_REGION (ARM)	一連のテキストまたはデータ範囲の最後を指定します。
ENTRY_END (ARM)	前の NESTED_ENTRY (ARM) によって指定されたルーチンを終了します。
EXCEPTION_HANDLER (ARM)	指定された例外ハンドラを次に続く NESTED_ENTRY (ARM) と関連付けます。
EXCEPTION_HANDLER_DATA (ARM)	指定された例外ハンドラとハンドラ データを次に続く NESTED_ENTRY (ARM) と関連付けます。
LEAF_ENTRY (ARM)	プロローグ コードを必要としないルーチンの最初を宣言します。
NESTED_ENTRY (ARM)	既存のスタック フレームがあるか、またはスタック フレームを作成するルーチンの最初を宣言します。
PROLOG_END (ARM)	プロローグ領域の最後を指定します。
START_REGION (ARM)	一連のテキストまたはデータ範囲の最初を指定します。

関連項目

概念

ARM レジスタ

ARM スタック フレームのレイアウト

その他のリソース

ARM プロローグおよびエピローグ

ALTERNATE_ENTRY (ARM)

このマクロによって、種類が NESTED_ENTRY (ARM) または LEAF_ENTRY (ARM) のルーチンの代替エントリが宣言されます。

```
ALTERNATE_ENTRY Name[,[Section=]SectionName]
```

パラメータ

Name

エントリ ポイント。

SectionName

そのエントリが含まれるセクションの名前。「備考」を参照してください。

戻り値

なし。

備考

Name にはグローバル名前空間に含まれる名前を使用します。

ALTERNATE_ENTRY マクロでは、*SectionName* パラメータは使用されません。このパラメータは受け入れられますが、NESTED_ENTRY (ARM) および LEAF_ENTRY (ARM) との整合性を保つために無視されます。

このパラメータを使用する場合は、**ALTERNATE_ENTRY** 呼び出しをルーチンの本体に含める必要があります。

関連項目

参照情報

NESTED_ENTRY (ARM)
LEAF_ENTRY (ARM)

END_REGION (ARM)

このマクロでは、一連のテキストまたはデータ範囲の終了が指定されます。

```
END_REGION NameEnd
```

パラメータ
NameEnd

NameEnd **は、範囲の最後にラベルを付けます。**

戻り値

なし。

備考

NameEnd **にはグローバル名前空間に含まれる名前を使用します。**

関連項目
参照情報
START_REGION (ARM)

ENTRY_END (ARM)

このマクロは、NESTED_ENTRY (ARM) または LEAF_ENTRY (ARM) によって指定されている現在のルーチンを終了します。

```
ENTRY_END [Name]
```

パラメータ
Name

「備考」を参照してください。

戻り値

なし。

備考

Name には、**NESTED_ENTRY** マクロまたは **LEAF_ENTRY** マクロで使用されるものと同じ名前を指定します。

ENTRY_END マクロでは *Name* が無視されます。

関連項目
参照情報
NESTED_ENTRY (ARM)
LEAF_ENTRY (ARM)

EXCEPTION_HANDLER (ARM)

このマクロでは、例外ハンドラ *Handler* を次に続く NESTED_ENTRY (ARM) または LEAF_ENTRY (ARM) ルーチンと関連付けます。

```
EXCEPTION_HANDLER Handler
```

パラメータ
Handler

例外ハンドラの名前。

戻り値

なし。

備考

この関連付けは、対応する ENTRY_END (ARM) マクロをコンパイラが検出するまで有効です。

関連項目
参照情報
NESTED_ENTRY (ARM)
LEAF_ENTRY (ARM)
ENTRY_END (ARM)
EXCEPTION_HANDLER_DATA (ARM)

EXCEPTION_HANDLER_DATA (ARM)

このマクロでは、例外ハンドラ *Handler* と *HandlerData* が、次に続く NESTED_ENTRY (ARM) **または** LEAF_ENTRY (ARM) **ルーチンに関連付けられます。**

```
EXCEPTION_HANDLER_DATA Handler,     HandlerData
```

パラメータ

Handler

例外ハンドラの名前。

HandlerData

例外ハンドラに関連付けられた文字列。

戻り値

なし。

備考

この関連付けは、一致する ENTRY_END (ARM) **マクロをコンパイラが検出するまで有効です。**

関連項目

参照情報

NESTED_ENTRY (ARM)
LEAF_ENTRY (ARM)
ENTRY_END (ARM)
EXCEPTION_HANDLER (ARM)

LEAF_ENTRY (ARM)

このマクロでは、プロローグ コードを必要としないルーチンの開始が宣言されます。

```
LEAF_ENTRY Name[,[Section=]SectionName]
```

パラメータ

Name

ルーチン名。

SectionName

省略可。そのエントリが含まれているセクションの名前。既定では .text。

戻り値

なし。

備考

Name **にはグローバル名前空間に含まれる名前を使用します。**

LEAF_ENTRY **には、関連付けられた** ENTRY_END (ARM) **が必要です。**

関連項目

参照情報

ENTRY_END (ARM)
PROLOG_END (ARM)

NESTED_ENTRY (ARM)

このマクロでは、既存のフレームを持つルーチン、またはスタック フレームを作成するルーチンの最初が宣言されます。

```
NESTED_ENTRY Name[,[Section=]SectionName]
```

パラメータ

Name

ルーチン名。

SectionName

省略可。そのエントリが含まれているセクションの名前。既定では text。

戻り値

なし。

備考

Name にはグローバル名前空間に含まれる名前を使用します。

NESTED_ENTRY には、関連付けられた PROLOG_END (ARM) および ENTRY_END (ARM) が必要です。

関連項目

参照情報

ENTRY_END (ARM)
PROLOG_END (ARM)

PROLOG_END (ARM)

このマクロは、プロローグ領域の最後を指定します。

```
PROLOG_END
```

パラメータ
なし。

戻り値

なし。

備考

このマクロは、NESTED_ENTRY (ARM) **マクロまたは** LEAF_ENTRY (ARM) **マクロに続けて使用する必要があります。**

このマクロは、プロローグ領域の後から、対応する ENTRY_END (ARM) **マクロの前までの間に使用されます。**

関連項目
参照情報
NESTED_ENTRY (ARM)
LEAF_ENTRY (ARM)
ENTRY_END (ARM)

START_REGION (ARM)

このマクロでは、一連のテキストまたはデータ範囲の開始が指定されます。

```
START_REGION NameBegin
```

パラメータ
NameBegin

NameBegin **は範囲の開始を表すラベルです。***NameBegin* **にはグローバル名前空間に含まれる名前を使用します。**

戻り値

なし。

関連項目
参照情報
END_REGION (ARM)

THUMB 対応 ARM の実装

THUMB 対応 ARM マイクロプロセッサは、通常の 32 ビット ARM 命令だけではなく 16 ビット THUMB 命令も実行できます。

THUMB 命令を実行するには、マイクロプロセッサを THUMB モードに切り替えます。ARM 命令の実行を再開するには、マイクロプロセッサを ARM モードに戻します。モード切り替えを実行する命令は分岐および交換 (**BX**) 命令です。

ARM コンパイラによってコンパイルされた関数内から THUMB コンパイラによってコンパイルされた関数を呼び出すには、呼び出し側と呼び出される側に対する **BX** 命令をコード シーケンスに含める必要があります。

BX 命令は、**BX Rx,** の形式で、**Rx,** はアドレスを格納するレジスタです。

小ビットの **BX** ターゲット アドレスが設定されている場合は、**BX** によって THUMB モードに切り替わります。

小ビットの **BX** ターゲット アドレスが設定されていない場合は、**BX** によって ARM モードに切り替わります。

リンカはリンク時に小ビットの THUMB 関数アドレスを設定します。

ARM C/C++ コンパイラは、関数呼び出しおよび ARM と THUMB の相互動作へのリターンをサポートします。相互動作をサポートする機構は次のとおりです。

- コンパイラ フラグ
- 相互動作する declspec 修飾子のための言語拡張
- リンク時のサンク ルーチンの生成

相互動作するコーリング シーケンスを使用するとモード切り替えが可能になりますが、必ず必要というわけではありません。

また、相互動作するコーリング シーケンスを使用すると、THUMB 関数は別の THUMB 関数を呼び出すことができ、ARM 関数は別の ARM 関数を呼び出すことができます。

関連項目

参照情報

相互動作用のコンパイラ フラグ

概念

修飾子

リンカで生成されるサンク ルーチン

コード サイズに関する考慮事項

相互動作用のコンパイラ フラグ

次のコンパイラ フラグは、相互動作プログラムの作成に適用されます。

- /QRinterwork-return – **相互動作の有効化**フラグを使用すると、1 つのファイル内のすべての関数のリターンで相互動作が行われます。
  このフラグは関数呼び出しには影響しません。
- /QRArch – **ターゲットのアーキテクチャの指定**フラグを使用すると、THUMB 対応のマイクロプロセッサ (**BX** など) でのみ使用可能な命令をコンパイラで生成できます。
  THUMB の場合、このフラグは既定でオンになっています。
  ARM の場合、このフラグは既定でオフになっています。
  /QRinterwork-return フラグによって **/QRarch** フラグが有効になります。

関連項目
概念
THUMB 対応 ARM の実装

修飾子

ARM および THUMB では、相互動作のための Microsoft 固有の修飾子がサポートされます。iwcall、iw16、および iw32 declspec を使用すると、相互動作する関数呼び出しを選択できます。

iwcall および iw16 **declspec** は相互呼び出しで常に、リンカによって生成されるサンク ルーチンより高速のコードを生成します。**declspec** によって実行パスに追加される **MOV** 命令は 1 つだけですが、リンカによって生成されるサンク ルーチンではメモリからの読み込みと分岐の 2 つの命令が追加されます。

関連項目

概念

THUMB 対応 ARM の実装
リンカで生成されるサンク ルーチン

iw16

ARM コンパイラでコンパイルした場合、**iw16 declspec** は、iwcall **declspec** と同じ機能を持ちます。つまり、相互動作呼び出しを行います。

THUMB コンパイラでコンパイルした場合、**iw16 declspec** は機能しません。

iw16 declspec を使用するのは、指定された関数を ARM と THUMB のどちらのコンパイラでコンパイルするかがわかっている場合のみにしてください。

iw16 declspec は、/QRArch – **ターゲットのアーキテクチャの指定**フラグも使用しない限り、ARM コンパイラでは機能しません。

/QRinterwork-return - **相互動作の有効化**フラグを使用すると、**/QRarch** フラグも自動的に有効になります。

関連項目
参照情報
iwcall
iw32
/QRinterwork-return - **相互動作の有効化**
/QRArch - **ターゲットのアーキテクチャの指定**
概念
修飾子

iw32

THUMB コンパイラでコンパイルした場合、**iw32 declspec** は iwcall **declspec** と同じ機能を持ちます。つまり、相互動作呼び出しを行います。

ARM コンパイラでコンパイルした場合、**iw32 declspec** は機能しません。

iw32 declspec を使用するのは、指定された関数を ARM と THUMB のどちらのコンパイラでコンパイルするかがわかっている場合のみにしてください。

関連項目

参照情報

iwcall

iw16

概念

修飾子

iwcall

iwcall declspec を使用すると、ARM または THUMB のいずれのコンパイラを使用しても、プログラマは相互動作する関数呼び出しを選択できます。

iwcall を使用すると、相互動作するサンク ルーチンのリンク時の生成を回避できます。

iwcall は、呼び出し元が見ることができる関数プロトタイプで使用されている場合にのみ有効です。

iwcall declspec を使用すると、関連付けられた関数で相互動作のリターンが発生しません。相互動作のリターンを発生させるには、-/QRinterwork-return – **相互動作の有効化**フラグを使用します。

ユーザーが指定した関数のプロトタイプで **__declspec(iwcall)** を使用すると、どちらの場合でもライブラリで処理できるようになります。

次のコード例は、**iwcall declspec** を使用して、ユーザーが指定した関数 **myfunction()** を有効にする方法を示しています。

```
__declspec(iwcall) int myfunction();
int main()
{
  return myfunction();
}
```

次の一覧は、/QRArch – **ターゲットのアーキテクチャの指定**を使用したコンパイル時に、この **declspec** によって生成されるコードを示しています。

```
    str       lr, [sp, #4]!
    ldr       r3, [pc, #8]
    mov       lr, pc
    bx        r3
    ldmia     sp!, {pc}
    DCD       |myfunction|
```

次の一覧は、/QRthumb **オプション**を使用して ARM コンパイラで生成されたコードを示しています。

```
    push      {lr}
    ldr       r3, [pc, #0x2]
    mov       r12, r3
    bl        __r12_indirect
    pop       {pc}
    DCD       |myfunction|
```

関連項目

参照情報

/QRinterwork-return – **相互動作の有効化**

/QRArch – **ターゲットのアーキテクチャの指定**

概念

修飾子

リンカで生成されるサンク ルーチン

declspec 呼び出しで、相互動作する関数呼び出しが有効にならない場合、リンカはサンク ルーチンを生成し、ARM と THUMB 間で必要な呼び出し側モード切り替えを実行します。リンカは、他方のモードが必要なオブジェクトから呼び出された一意の関数ごとに、サンク ルーチンを生成します。

関数からのリターン時に必要なモード切り替えの場合は、プログラマがコンパイル時に /QRinterwork-return – **相互動作の有効化フラグを使用する必要があります。**

次のコード例では、ARM **モードから** THUMB **モードに切り替える場合にリンカで生成されるサンク ルーチンを示しています。**

```
0xe59fc000    ldr  r12, [pc]
0xe12fff1c    bx   r12
0x00000000    DCD |destination|
```

次のコード例では、THUMB **モードから** ARM **モードに切り替える場合にリンカで生成されるサンク ルーチンを示しています。**

```
0xb408        push  {r3}
0x4b02        ldr  r3, [pc, #8]
0x469c        mov r12, r3
0xbc08        pop  {r3}
0x4760        bx   r12
0x0000        pad
0x00000000    DCD |destination|
```

関連項目

参照情報

/QRinterwork-return – **相互動作の有効化**
/QRArch - **ターゲットのアーキテクチャの指定**

コード サイズに関する考慮事項

declspec 修飾子とリンカで生成されるサンク ルーチンでは、どちらも 32 ビット ワードのテキスト領域命令が追加されます。

通常、プログラムに相互動作する関数が少ししか含まれていないのに、相互動作する関数が多数の場所から呼び出される場合、リンカで生成されたサンク ルーチンによって、全体のコード サイズが小さくなることが多いです。

プログラムにさまざまな相互動作関数が多く含まれるのに、その相互動作する関数が少数の場所からしか呼び出されない場合、iwcall または iw16 を使用すると、リンカで生成されたサンク ルーチンに必要なコード サイズと同様のサイズになります。

コード サイズの効率性が重要な場合は、次のワードの要件も考慮してください。

- **iwcall** および **iw16 declspecs** には、1 回の相互動作呼び出しにつき 32 ビット ワードのテキスト領域命令を追加する必要があります。
  関数アドレスはリテラル プールに配置して、リテラル プールから読み込む必要があるため、関数アドレスでは、リンカで生成されたサンク ルーチンに必要なスペースに 32 ビット ワードのテキスト領域命令を 2 つまで追加できます。
  複数の異なる読み込み命令で所定のリテラル プール エントリを共有できるため、関数呼び出しによっては、declspec にワードを追加しないものもあります。複数の異なる **BX** 命令で使用するレジスタに、そのレジスタを再度読み込まずに、リテラル プール エントリを読み込むことができます。
- ARM コンパイラに対してリンカでサンク ルーチンを生成できるようにするには、呼び出された関数ごとに追加の 3 つの 32 ビット ワードが必要になりますが、呼び出しサイトでは追加のワードは必要はありません。THUMB コンパイラに対してリンカでサンク ルーチンを生成するには、3 つではなく 4 つのワードが必要です。

注:
リンカは、関数呼び出し側の種類の数にかかわらず、1 回の関数呼び出しにつき相互動作サンク ルーチンを 1 つのみ生成します。

関連項目

概念

リンカで生成されるサンク ルーチン
修飾子

ARM アセンブラ

ARM アセンブラ (armasm) は 2 パス アセンブラで、ソース ファイルを 2 回処理することによって保持する必要のある内部状態の量を削減します。

ARM アセンブラは、ARM と THUMB の両方のアセンブリ言語を COFF (Common Object File Format) の Microsoft 実装にコンパイルします。

このセクションのトピック

ARM アセンブラ コマンドライン オプション

ARM アセンブラを呼び出すコマンドライン構文について説明し、コマンドライン オプションを示します。

あらかじめ宣言された ARM レジスタ名

ARM アセンブラがあらかじめ宣言するレジスタ名に関する参照情報を提供します。

組み込み ARM アセンブラ変数

ARM アセンブラに定義が組み込まれている変数に関する参照情報を提供します。

ARM アセンブラ ディレクティブ

アセンブラのディレクティブについて説明します。

関連セクション

ARM ファミリ プロセッサ

ARM アセンブラ コマンドライン オプション

ARM アセンブラを呼び出すコマンドの構文を以下に示します。

```
armasm {options} sourcefile objectfile
```

次の表は、ARM アセンブラのコマンドライン オプションを示しています。

オプション	説明
-archarchitecture	ターゲットのアーキテクチャを設定します。適正な値は 4、4T、5、および 5T です。 マイクロプロセッサ固有の命令によっては、不適切なターゲット アーキテクチャに対してアセンブルするとエラーまたは警告が表示されるものがあります。
-checkreglist	**LDM** および **STM** レジスタの一覧をチェックし、レジスタ番号が昇順になっていることを確認します。 昇順になっていない場合は、警告が表示されます。 `-checkreglist` を使用すると、レジスタのシンボリック名の誤用を検出できます。
-cpu ARMcore	ターゲットの ARM コアを設定します。 適正な値の例を以下に示します。 • ARM7TM • ARM720t • ARM920t • ARM8 • ARM10 • ARM10T • ARM10200 • StrongARM または StrongARM1 • XSCALE • PXA270 マイクロプロセッサ固有の命令によっては、不適切な ARM コアに対してアセンブルすると警告が表示されるものがあります。
-errors errorfile	エラー メッセージをエラー ファイルに出力します。
-help	コマンドライン オプションの概要を示します。
-i dir{,dir}	含めるディレクトリを指定します。 ディレクトリの指定は、INCLUDE 環境変数を使用して行うこともできます。
-ignore 0241 -archarchitecture	指定したターゲットのアーキテクチャに命令が実装されていないことを知らせる警告メッセージ 0241 を無視するようにビルド プロセスに指示します。
-list listingfile-noterse	terse フラグをオフにします。 terse フラグをオンにすると、条件付きアセンブリが原因でスキップされた行が一覧に表示されなくなります。 terse フラグをオフにすると、これらの行が一覧に表示されます。 既定ではオンになっています。

-list listingfile-width n	一覧ページの幅を設定します。 既定値は 79 文字です。
-list listingfile-length n	一覧ページの長さを設定します。 長さ 0 はページ付けされない一覧を意味します。 既定値は 66 行です。
-list listingfile-xref	シンボルの相互参照情報の一覧を表示します。 マクロ内およびマクロ外の定義場所と使用場所を示します。 既定ではオフになっています。
-noesc	C スタイルの特殊文字 (\n、\t など) を無視します。
-noregs	暗黙のレジスタ名 **r0 ～ r15、a1 ～ a4、v1 ～ v6、c0 ～ c15、p0 ～ p15、sl、fp、ip、sp、lr、pc** を事前定義しないようにアセンブラに指示します。
-nowarn	警告メッセージをオフにします。
-predefine "directive"	SETx ディレクティブを事前実行します。 これにより、対応する GBLx ディレクティブが暗黙的に実行されます。 このオプションにはスペースが含まれるため、SETx 引数全体を引用符で囲みます。次に例を示します。 `-predefine "Version SETA 44"`
-Via file	*file* を開いて追加の armasm コマンドライン引数を読み取ります。

関連項目

参照情報

あらかじめ宣言された ARM レジスタ名

組み込み ARM アセンブラ変数

その他のリソース

ARM アセンブラ

あらかじめ宣言された ARM レジスタ名

既定では、アセンブラによって次のレジスタ名があらかじめ宣言されます。

- **R0** ～ **R15** および **r0** ～ **r15**
- **c0** ～ **c15** のコプロセッサ レジスタ
- **p0** ～ **p15** のコプロセッサ レジスタ
- **a1** ～ **a4** のスクラッチ レジスタ、**r0** ～ **r3** と同義
- **v1** ～ **v8** の変数レジスタ、**r4** ～ **r11** と同義
- **sb** および **SB** のスタック ベース、**r9** と同義
- **sl** および **SL** のスタック ベース、**r10** と同義
- **fp** および **FP** のフレーム ポインタ、**r11** と同義
- **ip** および **IP** のプロシージャ内呼び出しスクラッチ レジスタ、**r12** と同義
- **sp** および **SP** のスタック ポインタ、**r13** と同義
- **lr** および **LR** のリンク レジスタ、**r14** と同義
- **pc** および **PC** のプログラム カウンタ、**r15** と同義
- **s0** ～ **s32** の VFP 単精度レジスタ
- **d0** ～ **d16** の VFP 倍精度レジスタ
- **fpsid** VFP システム ID レジスタ
- **fpscr** VFP ステータスおよびコントロール レジスタ
- **fpexc** VFP 例外レジスタ
- **wr0** ～ **wr16** の WMMX SIMD データ レジスタ
- **wc0** ～ **wc16** の WMMX ステータスおよびコントロール レジスタ
- **wcid** WMMX コプロセッサ ID レジスタ、**wc0** と同義
- **wcon** WMMX コントロール レジスタ、**wc1** と同義
- **wcssf** WMMX 飽和 SIMD フラグ、**wc2** と同義
- **wcasf** WMMX 算術 SIMD フラグ、**wc3** と同義
- **wcgr0** ～ **wcgr3** の WMMX コントロール汎用レジスタ、**wc8** ～ **wc11** と同義

関連項目

参照情報

組み込み ARM アセンブラ変数

その他のリソース

ARM アセンブラ

組み込み ARM アセンブラ変数

ARM アセンブラには、プログラマの利便性のために組み込み変数が用意されています。

次の表は、組み込み変数の定義を示しています。

変数	説明
{PC} または .	プログラム ロケーション カウンタの現在の値を格納します。
{VAR} または @	記憶域ロケーション カウンタの現在の値を格納します。
{TRUE}	論理定数 true を格納します。
{FALSE}	論理定数 false を格納します。
{OPT}	現在設定されている一覧オプションの値を格納します。 OPT ディレクティブを使用して、現在の一覧オプションの保存、一覧オプションにおける変更の強制、またはその元の値の復元を行うことができます。
{CONFIG}	アセンブラが 32 ビット プログラム カウンタ モードの場合、値は 32 です。26 ビット モードの場合、値は 26 です。
{ENDIAN}	アセンブラがビッグ エンディアン モードの場合、値は "big" です。リトル エンディアン モードの場合、値は "little" です。
{CODESIZE}	Thumb コードをコンパイルする場合、値は 16 です。それ以外の場合、値は 32 です。
{CPU}	選択された CPU の名前を格納します。CPU が指定されていない場合は、汎用 ARM の名前を格納します。
{ARCHITECTURE}	選択された ARM アーキテクチャの値を格納します。4 または 4T です。

関連項目

参照情報

あらかじめ宣言された ARM レジスタ名

その他のリソース

ARM アセンブラ

ARM アセンブラ ディレクティブ

ARM アセンブラでは一揃いのアセンブラ ディレクティブがサポートされており、これらを使用してソース ファイルのアセンブリを制御および管理することができます。

このセクションのトピック

ARM 初期化およびレイアウト ディレクティブ

ARM リンク ディレクティブ

ARM 診断ディレクティブ

ARM 条件付きアセンブリ ディレクティブ

ARM 動的一覧ディレクティブ オプション

ARM-Thumb 相互動作ディレクティブ

ARM の定数と変数の宣言

ARM アセンブラ マクロ定義ディレクティブ

その他の ARM ディレクティブ

関連セクション

ARM アセンブラ コマンドライン オプション

あらかじめ宣言された ARM レジスタ名

組み込み ARM アセンブラ変数

ARM アセンブラ エラー メッセージ

ARM 初期化およびレイアウト ディレクティブ

AREA ディレクティブは、アセンブラに対して新規のコードまたはデータ セクションのアセンブルの実行を指示します。使用する式の形式は次のとおりです。

```
AREA sectionname{,attr}{,attr}...
```

AREA *sectionname* 式は、1 つ以上のコンマ区切り属性で修飾します。

次の表に、AREA ディレクティブの有効な属性を示します。

属性	説明
ALIGN	セクションの境界を定義します。セクションは、$2^{expression}$ バイトの境界に整列します。 *expression* には、0 ～ 31 の整数値を使用します。コードのセクションに `ALIGN=0` または `ALIGN=1` を使用しないでください。 `ALIGN` 属性は、`ALIGN` ディレクティブとは異なります。
CODE	コンピュータの命令が含まれます。既定値は、`READONLY` です。
DATA	命令ではなく、データが含まれます。既定値は、`READWRITE` です。
NOINIT	データ セクションが初期化されていない、または 0 に初期化されていることを示します。これには、初期化値 0 と共に、領域予約ディレクティブである `SPACE` または `DCB`、`DCD< DCDU`、`DCQ`、`DCQU`、または `DCWU` のみが含まれます。
READONLY	セクションが書き込み禁止であることを示します。これは `CODE` 領域の既定値です。
READWRITE	セクションが読み取り、書き込み可能であることを示します。これは `DATA` 領域の既定値です。

関連項目

参照情報

ARM 初期化およびレイアウト ディレクティブ

ARM リンク ディレクティブ

次に示す ARM アセンブリ ディレクティブによって、他のファイルは現在のアセンブリにリンクされます。

ディレクティブ	構文	説明
EXPORT	`EXPORT symbol{[FPREGARGS,DATA,LEAF]}`	他の個々のオブジェクト ファイルによるリンク時に使用する *symbol* を宣言します。 *FPPREGARGS* は、fp レジスタで渡された fp 引数を要求する関数を定義します。 *DATA* は、関数やプロシージャではなくコードセグメント データを定義します。 *LEAF* は、他の関数を呼び出すアサートを生成します。 GLOBAL ディレクティブと同じ機能を持っています。
GET	`GET filename`	現在のファイル アセンブリ内に *filename* という名前のファイルを追加します。 追加されたファイル自体に、GET ディレクティブを使用してさらにファイルを追加することができます。 追加されたファイルのアセンブリが完了すると、**GET** ディレクティブに続く行でアセンブリが続行します。
GLOBAL	`GLOBAL symbol{[FPREGARGS,DATA,LEAF]}`	他の個々のオブジェクト ファイルによるリンク時に使用する *symbol* を宣言します。 *FPPREGARGS* は、fp レジスタで渡された fp 引数を要求する関数を定義します。 *DATA* は、関数やプロシージャではなくコードセグメント データを定義します。 *LEAF* は、他の関数を呼び出すアサートを生成します。 EXPORT ディレクティブと同じ機能を持っています。
IMPORT	`IMPORT symbol{WEAK=weak-symbol, {type=n}}`	アセンブラに *symbol* という名前を提供します。 アセンブリは *symbol* の定義を含みませんが、個々のオブジェクト ファイルで定義されたシンボルへのリンク時にこの定義を解決します。 このルーチンは *symbol* をプログラム アドレスとして扱います。
INCLUDE	`INCLUDE filename`	GET の類義語を指定します。

関連項目

その他のリソース

ARM アセンブラ ディレクティブ

ARM 診断ディレクティブ

次の ARM アセンブリ ディレクティブは、診断情報を生成します。

ディレクティブ	構文	説明
ASSERT	`ASSERT logical-expression`	診断の生成をサポートします。 *logical-expression* が FALSE を返した場合、アセンブラはアセンブリの 2 番目のパスのときに診断メッセージを生成します。 `ASSERT` はマクロの内部でも外部でも使用できます。
!	`! arithmetic-expression, string-expression`	`ASSERT` に関連しますが、アセンブリの両方のパスのときに検査され、カスタム エラー メッセージを作成する柔軟な方法を提供します。 *arithmetic-expression* が 0 の場合、アセンブラはパス 1 のときにアクションを実行しませんが、パス 2 のときに警告として *string-expression* を出力します。 *arithmetic-expression* が 0 でない場合、アセンブラは診断メッセージとして *string-expression* を出力します。アセンブリは パス 1 の後停止します。 このディレクティブは、`INFO` ディレクティブとまったく同じです。
INFO	`INFO arithmetic-expression, string-expression`	*arithmetic-expression* が 0 の場合、アセンブラはパス 1 のときにアクションを実行しません。 アセンブラは、ソース ファイルと行番号を *string-expression* のプレフィックスとして追加し、パス 2 のときに警告として結果を出力します。 *arithmetic-expression* が 0 でない場合、アセンブラは診断メッセージとして *string-expression* を出力します。アセンブリは パス 1 の後停止します。 このディレクティブは、`!` ディレクティブとまったく同じです。

関連項目

その他のリソース

ARM アセンブラ ディレクティブ

ARM 条件付きアセンブリ ディレクティブ

ARM アセンブラでは、ソース ファイルのセクションについては、条件付きアセンブリがサポートされます。条件が真の場合に、指定されたセクションがアセンブルされます。

また、反復テーブルを生成するときに役立つ条件付きループについては、WHILE...WEND ディレクティブがサポートされます。

条件付きループでは、実行時ループではなくアセンブリ時ループが生成されます。WHILE 条件の評価はループの最初に実行されるため、アセンブリ時にコードが生成されない場合がありますが、条件付きアセンブリについては行が表示されます。

条件付きおよび反復アセンブリには、次の ARM アセンブラ ディレクティブが必要です。

[または IF

条件の開始を指定します。

] または ENDIF

条件の終了を指定します。

| または ELSE

else 構文を挿入します。

WHILE

反復条件の開始を指定します。

WEND

反復条件の終了を指定します。

文字の [、|、および] は、行の先頭に記述できません。これらの文字の前にスペースまたはタブを挿入する必要があります。

条件付きアセンブリ構文の例を次に示します。

```
[ logical-expression
...code...
|
...code...
]
```

反復アセンブリ構文の例を次に示します。

```
WHILE logical-expression
...code...
WEND
```

関連項目

その他のリソース

ARM アセンブラ ディレクティブ

ARM 動的一覧ディレクティブ オプション

OPT ディレクティブは、ソース コード内から一覧オプションを設定する際に使用されます (一覧がオンの場合)。

既定の設定では、変数宣言、マクロ展開、呼び出し条件付きディレクティブ、および MEND ディレクティブを含む通常の一覧が生成されます。ただし、最初のパス時には一覧は生成されません。

これらの設定を変更するには、次の表に示すように、一覧から適切な値を追加して **OPT** ディレクティブと共に使用します。

1

通常の一覧をオンにします。

2

通常の一覧をオフにします。

4

ページ投入 : 直ちにフォーム フィードを発行し、新規ページを開始します。

8

行番号カウンタを 0 にリセットします。

16

SET、GBL、および LCL ディレクティブの一覧をオンにします。

32

SET、GBL、および LCL ディレクティブの一覧をオフにします。

64

マクロ展開の一覧をオンにします。

128

マクロ展開の一覧をオフにします。

256

マクロ呼び出しの一覧をオンにします。

512

マクロ呼び出しの一覧をオフにします。

1024

最初のパス一覧をオンにします。

2048

最初のパス一覧をオフにします。

4096

条件付きディレクティブの一覧をオンにします。

8192

条件付きディレクティブの一覧をオフにします。

16384

MEND ディレクティブの一覧をオンにします。

32768

MEND ディレクティブの一覧をオフにします。

関連項目

その他のリソース

ARM アセンブラ ディレクティブ

ARM-Thumb 相互動作ディレクティブ

次の表は、ARM-THUMB 相互動作ディレクティブを示しています。

CODE16

次に続く命令を 16 ビット (Thumb) 命令として解釈するようにアセンブラに指示します。

CODE32

次に続く命令を 32 ビット (ARM) 命令として解釈するようにアセンブラに指示します。

DATA

アセンブラに、ラベルが "data-in-code" ラベルであること、つまり、コード セグメント内のデータの領域を定義するように指示します。

Thumb コード領域のデータを定義する場合は、このディレクティブを指定する必要があります。

関連項目

その他のリソース

ARM アセンブラ ディレクティブ

ARM の定数と変数の宣言

次の ARM アセンブリ ディレクティブは定数の設定に使用されます。

ディレクティブ	構文	説明
*	`label *expression`	EQU と同じ。 固定式またはプログラム相対式にシンボリック ラベルを指定します。
CN	`label CN numeric-expression`	コプロセッサ レジスタ番号を指定します。c0 から c15 はあらかじめ定義されており、ラベルとしては使用できません。
CP	`label CP numeric-expression`	コプロセッサ番号に名前を指定します。可能であれば 0 から 15 までの範囲内にする必要があります。 p0 から p15 までの名前はあらかじめ定義されており、ラベルとしては使用できません。
EQU	`label EQU expression`	固定式またはプログラム相対式にシンボリック ラベルを指定します。
FN	`label FN numeric-expression`	可能であれば浮動小数点レジスタの名前を定義します。 F0 から F7 および f0 から f7 までの名前はあらかじめ定義されています。 あらかじめ定義されているレジスタ名はラベルとしては使用できませんが、数値式としては使用できます。
RN	`label RN numeric-expression`	レジスタ名を定義します。名前でのみレジスタを参照します。 R0 から R15、r0 から r15、PC、pc、LR および lr の名前はあらかじめ定義されています。 あらかじめ定義されているレジスタ名はラベルとしては使用できませんが、数値式としては使用できます。

このアセンブラではグローバル変数とローカル変数がサポートされます。グローバル変数のスコープは、ソース ファイル全体ですが、ローカル変数のスコープはマクロの特定のインスタンス化に限定されます。

次の表に、ローカル変数とグローバル変数の設定に使用される ARM アセンブリ ディレクティブを示します。

ディレクティブ	構文	説明
GBLA	`GBLA variable-name`	グローバル算術変数を定義します。 算術変数の値は、32 ビットの符号なし整数です。
GBLL	`GBLL variable-name`	グローバル論理変数を定義します。
GBLS	`GBLS variable-name`	グローバル文字列変数を定義します。
LCLA	`LCLA variable-name`	初期状態が 0 のローカル算術変数を定義します。
LCLL	`LCLL variable-name`	初期状態が FALSE のローカル論理変数を定義します。
LCLS	`LCLS variable-name`	初期状態が NULL 文字列のローカル文字列変数を定義します。

SETA	`variable-name SETA expression`	算術変数の値を設定します。
SETL	`variable-name SETL expression`	論理変数の値を設定します。
SETS	`variable-name SETS expression`	文字列変数の値を設定します。

文字列変数の値を設定するときは、引用符を使用する必要があります。

ローカル変数は、マクロ内からのみ宣言することができます。また、変数を宣言した後はその名前を他の用途に使用できません。

次に示すように、変数の中にはアセンブラによって値が置き換えられるものがあります。

- 変数名に $ 文字のプレフィックスがある場合、アセンブラは行の構文を確認する前に変数値を置き換えます。
- 変数が論理変数か算術変数の場合、アセンブラは変数に対して .STR 演算を実行し、変数をその演算の結果に置き換えます。

関連項目

その他のリソース

ARM アセンブラ ディレクティブ

ARM アセンブラ マクロ定義ディレクティブ

マクロは、一連の命令またはディレクティブが何度も必要な場合に使用すると便利です。

ARM アセンブラによって、マクロ名がその定義に置き換えられます。

マクロは、他のマクロへの呼び出しを格納でき、最大 255 レベルのネストが可能です。

マクロの定義には、**MACRO** と **MEND** の 2 つのディレクティブを使用します。

```
MACRO
{$label}    macroname {$parameter1}{,$parameter2}{,$parameter3}..
...code...
MEND
```

備考

マクロ プロトタイプのステートメントは、**MACRO** ディレクティブに続く最初の行に記述する必要があります。プロトタイプでは、マクロの名前 *macroname* とそのパラメータをアセンブラに指示します。

ラベルは省略可能ですが、マクロで内部ラベルを定義する場合は、ラベルを使用すると便利です。

任意の数のパラメータを使用できます。各パラメータには、通常のプログラム シンボルと区別するため、先頭に $ を付ける必要があります。

マクロ本文では、*$label*、*$parameter* などを他の変数と同様に使用できます。

$label パラメータは、マクロへの別のパラメータとして処理されます。マクロには、どのラベルがどこに定義されているかを記述します。ラベルは、マクロを展開する最初の命令を示しません。

ラベルは、たとえば、ループなど複数の内部ラベルを使用するマクロで、異なるサフィックスを持つベース ラベルとして各内部ラベルを定義する場合に便利です。

マクロ パラメータまたはラベルに値が追加されることがあります。この値はドットで区切ります。アセンブラは、パラメータおよびラベルの終了点を認識すると、そのドットを無視します。次に例を示します。

```
$label.1
$label.loop
$label.$count
```

パラメータに既定値を設定するには、パラメータの後に等号記号と既定値を記述します。既定値の前や後にスペースがある場合は、次のコード例に示すように、既定値全体を二重引用符で囲みます。

```
...{$parameter="default value"}
```

MEND ディレクティブは、マクロ定義の終了点を示します。マクロに WHILE/WEND ループまたは条件付きアセンブリが含まれる場合、WHILE/WEND ループは、**MEND** ディレクティブに達する前に終了する必要があります。

マクロの展開は、**MEXIT** ディレクティブを使用して終了することもできます。

関連項目

その他のリソース

ARM アセンブラ ディレクティブ

その他の ARM ディレクティブ

次の表に、ARM アセンブラのその他のディレクティブを示します。

ディレクティブ	構文	説明
ALIGN	`ALIGN {power-of-two {,offset-expression}}`	命令の位置を次のワード境界に設定します。 省略可能な power-of-two パラメータは、より粗いバイト境界との整列に使用できます。また、offset-expression パラメータは、その境界からのバイト オフセットの定義に使用できます。
END	`END`	ソース ファイルの処理を停止します。 GET ディレクティブがファイルのアセンブリを呼び出した場合、アセンブラは GET ディレクティブの後に戻り、処理を続行します。 アセンブラが最初のパス時に最上位のソース ファイルの END ディレクティブに達し、エラーがない場合、2 番目のパスが開始します。 END ディレクティブがない場合は、エラーが発生します。
KEEP	`KEEP {symbol}`	アセンブラのシンボル テーブルにローカル シンボルを保持します。 既定では、アセンブラはローカル シンボルをその出力オブジェクト ファイルに記述しません。 symbol パラメータなしで KEEP ディレクティブを指定した場合は、アセンブラですべてのシンボルが保持されます。 特定のシンボルを保持するには、その名前で指定します。
LTORG	`LTORG`	アセンブラに対し、アセンブリの直後に現在のリテラル プールをアセンブルするように指示します。 既定の LTORG は、ネストされたアセンブリの一部ではないすべての END ディレクティブで実行されます。 大きなプログラムでは複数のリテラル プールをそれぞれのリテラルに近い位置に配置し、LDR の 4 KB オフセット制限に違反しないようにする必要があります。
NOFP	`NOFP`	浮動小数点命令を無効にします。 ターゲットのハードウェアまたはソフトウェアで浮動小数点命令がサポートされていない場合もあります。
RLIST	`label RLIST list-of-registers`	LDM または STM によって転送される一連のレジスタに名前を割り当てます。 List-of-registers は、コンマで区切って中かっこで囲んだ一連のレジスタの名前の一覧または範囲です。 -CheckReglist オプションを選択した場合は、List-of-registers リストを昇順のレジスタ番号を付けて指定する必要があります。
ROUT	`{name}ROUT`	ローカル ラベルのスコープの境界を指定します。 名前は、スコープに割り当てる名前を示します。 ROUT を使用すると、ローカル ラベルのスコープを制限できます。これにより、誤ったラベルへの参照を容易に回避できます。ローカル ラベルのスコープに ROUT ディレクティブがない場合、そのスコープは領域全体を示します。
SUBT	`SUBT title`	新しいサブタイトルが指定されるまで、すべてのページに挿入するサブタイトルを指定します。

TTL	`TTL title`	**新しいタイトルが指定されるまで、すべてのページに挿入するタイトルを指定します。**

関連項目

その他のリソース

ARM アセンブラ ディレクティブ

ARM アセンブラ エラー メッセージ

このセクションでは、ARM アセンブラのエラー メッセージについて説明します。

このセクションのトピック

ARM エラー メッセージ 1 ～ 50

次の表は、ARM エラー メッセージ 1 ～ 50 の一覧です。

番号	エラー メッセージ	警告/エラー	引数
1		エラー	なし
2	:CHR:operator:truncation applied to a char（char に適用された切り捨て）	警告	なし
3	missing substitution operator $ for assembler variable %s (アセンブラ変数 %s に置換演算子 $ がありません。)	警告	なし
4	reserved symbol used in an expression: %s (式で、予約されているシンボルが使用されました: %s。)	警告	シンボル名
5	improper line syntax: %s (不適切な行の構文: %s。)	エラー	不適切な行の構文
6	improper program syntax: %s (不適切なプログラムの構文: %s。)	エラー	エラーの説明
7	unimplemented class of instructions (実行不可能な命令のクラス。)	エラー	なし
8	ARM register expected: %s (ARM レジスタが必要です: %s。)	エラー	アセンブラがレジスタを読み取ろうとした文字列
9	coprocessor number expected: %s (コプロセッサ番号が必要です: %s。)	エラー	アセンブラがコプロセッサ番号を読み取ろうとした文字列
10	ARM coprocessor register expected: %s (ARM コプロセッサ レジスタが必要です: %s。)	エラー	アセンブラがコプロセッサ レジスタを読み取ろうとした文字列
11	ARM floating point register expected: %s (ARM 浮動小数点レジスタが必要です: %s。)	エラー	アセンブラが浮動小数点レジスタを読み取ろうとした文字列
12	wrong operand type for %s (%s のオペランドの種類が正しくありません。)	エラー	オペランドを含む文字列
13	divide by zero (0 で除算。)	エラー	なし
14	%s cannot be applied; different sections or different base registers (%s は適用できません。セクションが異なるか、基底レジスタが異なります。)	エラー	オペランドのテキスト
15	illegal symbol %s; AREA name expected (無効なシンボル %s。AREA 名が必要です。)	エラー	なし
15	illegal symbol %s; AREA name expected (無効なシンボル %s。AREA 名が必要です。)	エラー	シンボル名
16	illegal flag(s) %s (無効な 1 つ以上のフラグ %s。)	エラー	トークン
17	illegal $ substitution, non assembler variable (無効な $ 置換。アセンブラ変数ではありません。)	エラー	なし

18	wrong character constant (**文字定数が正しくありません。**)	**エラー**	**なし**
19	malformed string constant; missing closing (**文字列定数の形式が正しくありません。終了が指定されていません。**)	**エラー**	**なし**
19	malformed string constant; missing closing (**文字列定数の形式が正しくありません。終了が指定されていません。**)	**エラー**	**なし**
20	wrong escape constant; null value not permitted inside a string (**エスケープ定数が正しくありません。文字列内に** NULL **値は許可されていません。**)	**エラー**	**なし**
20	wrong escape constant; null value not permitted inside a string (**エスケープ定数が正しくありません。文字列内に** NULL **値は許可されていません。**)	**エラー**	**なし**
21	unexpected char (**予期しない文字。**)	**エラー**	**なし**
22	wrong :: operator (:: **演算子が正しくありません。**)	**エラー**	**なし**
23	special bar id not closed (**特殊バー** ID **が終了していません。**)	**エラー**	**なし**
24	wrong built-in variable (**組み込み変数が正しくありません。**)	**エラー**	**なし**
25	wrong shift name (**シフト名が正しくありません。**)	**エラー**	**なし**
26	unimplemented instruction set (**実行不可能な命令セット。**)	**エラー**	**なし**
27	unaligned absolute value (**絶対値が整列されていません。**)	**警告**	**なし**
28	truncation applied, absolute value should be word aligned: 0x%x (**切り捨てが適用されました。絶対値は整列されたワードである必要があります**: 0x%x。)	**警告**	**オフセット値**
29	truncation applied, absolute value should be half-word aligned: 0x%x (**切り捨てが適用されました。絶対値は整列されたハーフワードである必要があります**: 0x%x。)	**警告**	**オフセット値**
30	address 0x%x is too large to represent in %d bits (**アドレス** 0x%x **が、**%d **ビットで表すには大きすぎます。**)	**エラー**	**アドレス**
31	address out of range in a different section (**アドレスが範囲外の別のセクションにあります。**)	**エラー**	**なし**
32	immediate value %d cannot be represented in %d bits (**即値** %d **を** %d **ビットで表すことはできません。**)	**エラー**	**オフセット**
33	floating value %e cannot be represented in single precision (**浮動小数点** %e **を単精度で表すことはできません。**)	**エラー**	**浮動小数点値**

34	undefined symbol: %s (**未定義のシンボル**: %s。)	**エラー**	**シンボル テキスト**
35	illegal expression type: %s (**無効な式の型**: %s。)	**エラー**	**式の値**
36	<%s> mnemonic qualifier is ignored (<%s> **ニーモニック修飾子は無視されます。**)	**警告**	**命令のサフィックス**
37	operand restriction violation (undefined behavior): %s (**オペランド制限違反** (**未定義の動作**): %s。)	**警告**	**命令のオペランド**
38	multiple use of the same register in a register list (1 **つのレジスター覧で同一レジスタが複数使用されています。**)	**警告**	**なし**
39	unsafe use of R15 as base register (R15 **を基底レジスタとして安全ではない方法で使用しました。**)	**警告**	**なし**
40	unsafe use of write-back operator (**書き戻し演算子の使い方が安全ではありません。**)	**警告**	**なし**
41	unknown LDM/STM operation (**不明な** LDM/STM **処理。**)	**エラー**	**なし**
42	thumb register expected; %s (Thumb **レジスタが必要です**: %s。)	**エラー**	**予期しないテキスト**
43	multiple symbol definition or incompatibility: %s (**シンボルの複数定義または非互換性**: %s。)	**エラー**	**シンボル名**
44	cannot open file: %s (**ファイルを開くことができません**: %s。)	**エラー**	**ファイル名**
45	possible infinite loop due to recursive file inclusion: %s (**再帰ファイルが含まれることによる無限ループの可能性があります**: %s。)	**警告**	**ファイル名**
46	unknown command-line argument or argument value %s (**不明なコマンドライン引数または引数値** %s。)	**警告**	**不明な引数**
47	command-line option not implemented; %s ignored (**コマンドライン オプションは実装されていません。**%s **は無視されました。**)	**警告**	**コマンドラインの引数**
48	multiple use of the same option: %s (**同一オプションが複数使用されています**: %s。)	**警告**	**コマンドラインの引数**
49	only one source file can be assembled: %s (**アセンブルできるソース ファイルは** 1 **つだけです**: %s。)	**エラー**	**ソース ファイル名**
50	missing input source file (**入力ソース ファイルがありません。**)	**エラー**	**なし**

関連項目

その他のリソース

ARM アセンブラ エラー メッセージ

ARM エラー メッセージ 51 ～ 100

次の表に、ARM エラー メッセージ 51 ～ 100 を示します。

番号	エラー メッセージ	警告/エラー	引数
51	unknown opcode: %s (不明な命令コード: %s。)	エラー	OPCODE の文字列
52	minimum requested width of 34 assumed (最小必要幅の想定値は 34 です。)	警告	なし
53	wrong page length: %d, 0 (non paged) assumed (ページの長さ %d が正しくありません。想定値は 0 (ページ付けなし) です。)	警告	ページの長さ
54	failed to close file %s (ファイル %s の終了に失敗しました。)	エラー	ファイル名
55	illegal line start symbol: %s for directive: %s (ディレクティブ %s の行の開始シンボル %s が無効です。)	エラー	開始シンボル、ディレクティブ
56	unimplemented directive %s (ディレクティブ %s が実装されていません。)	警告	ディレクティブ名
57	%s attribute does not pertain to a relocatable module; ignored (%s 属性は再配置可能モジュールに属していないため、無視されました。)	警告	ディレクティブ名
58	%s directive does not pertain to a relocatable module; ignored (%s ディレクティブは再配置可能モジュールに属していないため、無視されました。)	警告	ディレクティブ名
59	unknown section flag: %s (不明なセクション フラグ: %s。)	エラー	領域名
60	illegal combination of section flags: %s (無効なセクション フラグの組み合わせ: %s。)	警告	セクション名およびフラグ
61	redefinition of section flags ignored (セクション フラグの再定義が無視されました。)	警告	なし
62	ignored lines after END directive (END ディレクティブの後の行が無視されました。)	警告	なし
63	missing END directive (END ディレクティブが見つかりません。)	警告	なし
64	code inside data section (データ セクション内にコードがあります。)	エラー	なし
65	unknown or bad symbol type: %s (不明または無効のシンボルの種類: %s。)	エラー	シンボル

66	out of memory: section data: no object file will be generated (セクション データでメモリが不足しています。オブジェクト ファイルは生成されません。)	エラー	なし
66	out of memory: section data: no object file will be generated (セクション データでメモリが不足しています。オブジェクト ファイルは生成されません。)	エラー	なし
67	initialized data in an uninitialized data section %s (初期化されていないデータ セクション %s のデータが初期化されました。)	エラー	なし
68	assembler internal error; sync. error on local labels (アセンブラの内部エラー。ローカル ラベルの同期エラーです。)	エラー	なし
68	assembler internal error; sync. error on local labels (アセンブラの内部エラー。ローカル ラベルの同期エラーです。)	エラー	なし
69	error while generating object file: seek error (オブジェクト ファイル生成時にシーク エラーが発生しました。)	エラー	なし
69	error while generating object file: seek error (オブジェクト ファイル生成時にシーク エラーが発生しました。)	エラー	なし
70	input error: line is too long after expanding an assembler variable; truncated (入力エラー。アセンブラ変数の展開後の行が長すぎるため、切り捨てられました。)	エラー	なし
70	input error: line is too long after expanding an assembler variable; truncated (入力エラー。アセンブラ変数の展開後の行が長すぎるため、切り捨てられました。)	エラー	なし
71	macro error: END directive cannot appear inside a macro (マクロ エラー。END ディレクティブはマクロ内に配置できません。)	エラー	なし
71	macro error: END directive cannot appear inside a macro (マクロ エラー。END ディレクティブはマクロ内に配置できません。)	エラー	なし
72	wrong directive usage: local assembler variables can only be defined within a macro (ディレクティブの使い方が正しくありません。ローカル アセンブラ変数はマクロ内でのみ定義可能です。)	エラー	なし
73	wrong local label reference; reference expected after %% (usage: %%{x}{y}n{name}) (ローカル ラベルの参照が正しくありません。参照は %% の後でなければなりません (使用法: %%{x}{y}n{name})。)	エラー	なし
74	improper argument for <-predefine> command-line option: %s; %s (<-predefine> コマンドライン オプションの引数 %s、%s が不適切です。)	エラー	なし
75	non increasing order in the register list (レジスター覧が昇順になっていません。)	警告	なし
76	the selected architecture and CPU (or mode) are conflicting: %s versus %s (選択したアーキテクチャと CPU (またはモード) が競合しています (%s と %s)。)	警告	アーキテクチャ、CPU
77	could not delete %s file (%s ファイルを削除できませんでした。)	エラー	ファイル名
78	missing ENDP directive in section %s (セクション %s に ENDP ディレクティブがありません。)	エラー	セクション名
79	%s	エラー	内部使用

80	%s %s	エラー	内部使用
81	error (エラー)	エラー	なし
82	warning (警告)	警告	なし
83	illegal expression type; expected absolute numeric (式の型が無効です。絶対数値を指定してください。)	エラー	なし
84	illegal expression type; expected absolute numeric or program relative (式の型が無効です。絶対数値またはプログラム相対を指定してください。)	エラー	なし
85	illegal expression type; expected absolute numeric, program relative or register relative (式の型が無効です。絶対数値、プログラム相対、またはレジスタ相対を指定してください。)	エラー	なし
86	illegal expression type; expected program relative or register relative (式の型が無効です。プログラム相対またはレジスタ相対を指定してください。)	エラー	なし
87	illegal expression type; expected program relative (式の型が無効です。プログラム相対を指定してください。)	エラー	なし
88	illegal expression type; expected absolute numeric or string (式の型が無効です。絶対数値または文字列を指定してください。)	エラー	なし
89	illegal expression type; expected absolute numeric or Boolean (式の型が無効です。絶対数値またはブール値を指定してください。)	エラー	なし
90	illegal expression type; expected bool (式の型が無効です。ブール値を指定してください。)	エラー	なし
91	illegal expression type; expected string (式の型が無効です。文字列を指定してください。)	エラー	なし
92	no immediate rotate operand can be created: %d (即時回転オペランドを作成できません。%d。)	エラー	回転する値
93	immediate value %d out of range; expected values: %s (即値 %d は範囲外です。期待値は %s です。)	警告	即値。期待値を含む文字列
94	improper alignment value %d; expected: [1, 2, 4, 8, ... 32768] (配列の値 %d が不適切です。使用可能な値は [1, 2, 4, 8, ... 32768] です。)	エラー	値
95	improper alignment value %d; alignment offset should be positive and smaller than alignment; truncated (配置の値 %d が不適切です。配置オフセットは配置よりも小さい正の数でなければなりません。値は切り捨てられました。)	警告	値
96	immediate value %d out of range; positive value expected (即値 %d は範囲外です。正の値を指定してください。)	エラー	即値
97	immediate value %d out of range; truncated to 16 bits (即値 %d は範囲外のため、16 ビットに切り捨てられました。)	警告	即値

98	immediate value %d out of range; truncated to 8 bits (**即値 %d は範囲外のため、8 ビットに切り捨てられました。**)	**警告**	**即値**
99	literal pool too far, or too close: %d; address is 8 bit offset; use LTORG (**リテラル プール %d は遠すぎるか、または近すぎます。アドレスは 8 ビット オフセットです。**LTORG **を使用してください。**)	**エラー**	**なし**
100	literal pool too far, or too close: %d; address is 12 bit offset; use LTORG (**リテラル プール %d は遠すぎるか、または近すぎます。アドレスは 12 ビット オフセットです。**LTORG **を使用してください。**)	**エラー**	**なし**

関連項目

その他のリソース

ARM アセンブラ エラー メッセージ

ARM エラー メッセージ 101-150

次の表は、ARM エラー メッセージ 101 ～ 150 の一覧です。

番号	エラー メッセージ	警告/エラー	引数
101	improper shift amount: %d; must fit in 5 bits (不適切なシフト量: %d。5 ビット以内に収める必要があります。)	エラー	即値
102	improper rotate amount: %d; rotate amount shifted right by 1 must fit 4bits (不適切な回転量: %d。1 つ右にシフトする回転量は 4 ビット以内に収める必要があります。)	エラー	即値
103	improper rotate amount: %d; only even rotate amounts can be specified (不適切な回転量: %d。偶数の回転量しか指定できません。)	エラー	即値
104	immediate value out of range: %d; coprocessor information must fit in 3 bits (範囲外の即値: %d。コプロセッサ情報は 3 ビット以内に収める必要があります。)	エラー	即値
105	immediate value out of range: %d; coprocessor opcode must fit in 4 bits (範囲外の即値: %d。コプロセッサ opcode は 4 ビット以内に収める必要があります。)	エラー	即値
106	internal assembler error: unused error code (内部アセンブラ エラー: 未使用エラー コード。)	エラー	なし
107	improper line syntax, only <ldr> can be used to generate literals (不適切な行の構文。リテラルを生成する場合は <ldr> のみが使用可能です。)	エラー	なし
108	improper line syntax (不適切な行の構文。)	エラー	なし
109	improper line syntax; EOL or comma expected (不適切な行の構文。EOL またはコンマが必要です。)	エラー	なし
110	improper line syntax; wrong offset (不適切な行の構文。オフセットが正しくありません。)	エラー	なし
111	improper line syntax; expected: %s (不適切な行の構文。%s が必要です。)	エラー	適切な構文に必要なトークン。
112	improper line syntax; SP or PC required as second register in this format (不適切な行の構文。この形式では、2 番目のレジスタとして SP または PC が必要です。)	エラー	なし

113	improper line syntax; high registers are not accepted for SUB instruction (**不適切な行の構文。SUB 命令にハイ レジスタは使用できません。**)	**エラー**	**なし**
114	improper line syntax; only SP register accepted for this format (**不適切な行の構文。この形式では、SP レジスタのみが使用可能です。**)	**エラー**	**なし**
115	improper line syntax; unexpected comma (**不適切な行の構文。予期しないコンマが見つかりました。**)	**エラー**	**なし**
116	improper line syntax; high registers not allowed in this context (**不適切な行の構文。このコンテキストでは、ハイ レジスタは使用できません。**)	**エラー**	**なし**
117	improper line syntax; wrong combination of mnemonic and operands (**不適切な行の構文。ニーモニックとオペランドの組み合わせが正しくありません。**)	**エラー**	**なし**
118	improper line syntax; write back '!', expected (assumed) (**不適切な行の構文。'!' と書いて戻す必要があります。**)	**エラー**	**なし**
119	improper line syntax; opcode, directive or macro call expected (**不適切な行の構文。opcode、ディレクティブ、またはマクロの呼び出しが必要です。**)	**エラー**	**なし**
120	improper line syntax; wrong use of a local label (**不適切な行の構文。ローカル ラベルの使用方法が正しくありません。**)	**エラー**	**なし**
121	improper line syntax; %s directive cannot define a starting line symbol (**不適切な行の構文。%s ディレクティブでは行の開始シンボルを定義できません。**)	**エラー**	**ディレクティブ名**
122	improper line syntax; symbol expected (**不適切な行の構文。シンボルが必要です。**)	**エラー**	**なし**
123	improper line syntax; missing weak alias; symbol defined as external (**不適切な行の構文。弱いエイリアスがありません。シンボルが外部として定義されています。**)	**警告**	**なし**
124	improper line syntax; include file name expected (**不適切な行の構文。必要なファイル名を含めてください。**)	**エラー**	**なし**
125	improper line syntax; string expected (**不適切な行の構文。文字列が必要です。**)	**警告**	**なし**
126	improper line syntax; %s option requires %s section name (**不適切な行の構文。%s オプションには %s セクション名が必要です。**)	**警告**	**オプション、セクション名**

127	improper line syntax; usage: %s (**不適切な行の構文。使用法**: %s。)	**警告**	**行**
128	improper line syntax; selection must be 5 if <assoc= ..> option is used (**不適切な行の構文。**<assoc= ..> **オプションを使用する場合、選択は** 5 **でなければなりません。**)	**警告**	**なし**
129	improper line syntax; bad macro call; macro definition does not have a label parameter (**不適切な行の構文。マクロの呼び出しが正しくありません。マクロ定義にラベル パラメータがありません。**)	**警告**	**なし**
130	improper line syntax; macro call with too many actual parameters (**不適切な行の構文。マクロの呼び出しに実パラメータが多すぎます。**)	**警告**	**なし**
131	improper line syntax; malformed macro call (**不適切な行の構文。マクロの呼び出しの形式が正しくありません。**)	**警告**	**なし**
132	improper line syntax; register list symbol or proper register list ({ .. }) expected (**不適切な行の構文。レジスター覧シンボルまたは適切なレジスター覧** ({ .. }) **が必要です。**)	**エラー**	**なし**
133	improper line syntax; malformed register list construction (**不適切な行の構文。レジスター覧の構造の形式が正しくありません。**)	**エラー**	**なし**
134	improper line syntax; (**不適切な行の構文。**)	**エラー**	**なし**
135	improper line syntax; (**不適切な行の構文。**)	**エラー**	**なし**
136	improper line syntax; (**不適切な行の構文。**)	**エラー**	**なし**
137	unexpected end of file in an include file/macro or missing END directive (**インクルード ファイルまたはマクロに予期しないファイルの終点が見つかりました。または** END **ディレクティブがありません。**)	**エラー**	**なし**
138	improper program syntax; missing ENDP directive or nested function definition (**不適切なプログラムの構文。**ENDP **ディレクティブがありません。または関数定義がネストされています。**)	**警告**	**なし**
139	improper program syntax; unexpected ENDP directive (**不適切なプログラムの構文。予期しない** ENDP **ディレクティブが見つかりました。**)	**警告**	**なし**
140	improper program syntax; unexpected MEND/MEXIT directive (**不適切なプログラムの構文。予期しない** MEND/MEXIT **ディレクティブが見つかりました。**)	**警告**	**なし**

141	improper program syntax; conditional if structure crosses file (include) structure (**不適切なプログラムの構文。構造がファイル (インクルード) 構造と不整合になる場合は条件が課されます。**)	**警告**	**なし**
142	improper program syntax; (malformed/not closed) conditional if structure (**不適切なプログラムの構文。(無効な形式/未終了) 構造が表示された場合は条件が課されます。**)	**エラー**	**なし**
143	improper program syntax; end of file or macro in WHILE loop, or missing WEND (**不適切なプログラムの構文。**WHILE **ループにファイルまたはマクロの終点が見つかりました。または** WEND **がありません。**)	**エラー**	**なし**
144	improper program syntax; a WEND directive without a preceding WHILE (**不適切なプログラムの構文。**WEND **ディレクティブの前に** WHILE **がありません。**)	**エラー**	**なし**
145	improper program syntax; unbalanced WHILE-WEND construction, possibly due to conditionals and includes (**不適切なプログラムの構文。**WHILE-WEND **構造が対になっていません。条件付きおよびインクルードが原因であると考えられます。**)	**警告**	**なし**
146	improper program syntax; an ENDIF directive without a corresponding IF (**不適切なプログラムの構文。**END IF **ディレクティブに対応する** IF **がありません。**)	**エラー**	**なし**
147	improper program syntax; an ELSE directive without a corresponding IF (**不適切なプログラムの構文。**ELSE **ディレクティブに対応する** IF **がありません。**)	**エラー**	**なし**
148	improper program syntax; malformed conditional structure, possibly two ELSE keywords in an IF construct (**不適切なプログラムの構文。条件付き構造の形式が正しくありません。**IF **構造に** 2 **つの** ELSE **キーワードがある可能性があります。**)	**エラー**	**なし**
149	improper program syntax; nested macro definitions are not supported (**不適切なプログラムの構文。ネストされたマクロ定義はサポートされません。**)	**エラー**	**なし**
150	improper program syntax; read error or missing macro definition line (**不適切なプログラムの構文。読み取りエラーです。またはマクロ定義行がありません。**)	**エラー**	**なし**

関連項目

その他のリソース

ARM アセンブラ エラー メッセージ

ARM エラー メッセージ 151 ～ 200

次の表に、ARM エラー メッセージ 151 ～ 200 を示します。

番号	エラー メッセージ	警告/エラー	引数
151	improper program syntax; MEND, MEXIT directives do not have any label or prefixing text (不適切なプログラム構文。MEND および MEXIT ディレクティブにラベルまたはプレフィックスのテキストが指定されていません。)	警告	なし
152	improper program syntax; read error, missing MEND directive or end of file inside a macro definition (不適切なプログラム構文。読み取りエラー。マクロ定義内に MEND ディレクティブまたはファイルの終了が指定されていません。)	エラー	なし
153	improper program syntax; malformed macro definition (unrecoverable) (不適切なプログラム構文。マクロ定義の形式が正しくありません (回復不可)。)	エラー	なし
154	improper program syntax; associated COMDAT section not found (不適切なプログラム構文。関連付けられている COMDAT セクションが見つかりません。)	エラー	なし
155	improper program syntax; attempt to redefine the associated COMDAT section; ignored (不適切なプログラム構文。関連付けられている COMDAT セクションの再定義が試行されます。無視されました。)	エラー	なし
156	routine name does not match the enclosing routine structure (ルーチン名が、それを含むルーチン構造に一致しません。)	エラー	なし
157	improper argument for <-predefine> command-line option: name expected (<-predefine> コマンドライン オプションの引数が不適切です。名前が必要です。)	エラー	なし
158	improper argument for <-predefine> command-line option: %s; symbol name expected (<-predefine> コマンドライン オプションの引数が不適切です: %s。シンボル名が必要です。)	エラー	コマンドライン オプション
159	improper argument for <-predefine> command-line option: SETx expected (<-predefine> コマンドライン オプションの引数が不適切です。SETx が必要です。)	エラー	なし
160	improper argument for <-predefine> command-line option; undefined symbol in expression (<-predefine> コマンドライン オプションの引数が不適切です。式に未定義のシンボルがあります。)	エラー	なし
161	improper argument for <-predefine> command-line option; malformed or unrecognized expression (<-predefine> コマンドライン オプションの引数が不適切です。式の形式が正しくないか、認識されない式です。)	エラー	なし
162	improper argument for <-predefine> command-line option: expression type mismatch (<-predefine> コマンドライン オプションの引数が不適切です。式の型が一致しません。)	エラー	なし

163	assembler internal error; Sync. error on ROUT areas (**アセンブラ内部エラー。ROUT 領域で同期エラーが発生しました。**)	**エラー**	**なし**
164	assembler internal error; symbol sync. error in function symbol line numbers (**アセンブラ内部エラー。関数シンボルの行番号でシンボルの同期エラーが発生しました。**)	**エラー**	**なし**
165	assembler internal error; address sync. error in function symbol line numbers (**アセンブラ内部エラー。関数シンボルの行番号でアドレス同期エラーが発生しました。**)	**エラー**	**なし**
166	assembler internal error; INCLUDE path cannot be created, check INCLUDE env. variable and <-i> command-line option (**アセンブラ内部エラー。**INCLUDE **パスを作成できません。**INCLUDE **環境変数と** <-i> **コマンドライン オプションを確認してください。**)	**警告**	**なし**
167	assembler internal error; source input sync. error; bad use of include or macros (**アセンブラ内部エラー。ソース入力同期エラー。インクルード ファイルまたはマクロの使用が正しくありません。**)	**エラー**	**なし**
168	assembler internal error; bad input stack (**アセンブラ内部エラー。入力スタックが正しくありません。**)	**エラー**	**なし**
169	assembler internal error; bad macro structure in token input (**アセンブラ内部エラー。トークン入力のマクロ構造が正しくありません。**)	**エラー**	**なし**
170	assembler internal error; section raw data area exceeded (**アセンブラ内部エラー。セクションの生データ領域が超過しました。**)	**エラー**	**なし**
171	assembler internal error; PC sync error while generating literal pool (**アセンブラ内部エラー。リテラル プールの生成中に** PC **同期エラーが発生しました。**)	**エラー**	**なし**
172	ROUT tag of a local label does not match the enclosed ROUT name (**ローカル ラベルの** ROUT **タグが、それに囲まれる** ROUT **名に一致しません。**)	**エラー**	**なし**
173	unknown command-line argument or argument value; error file name expected (**不明なコマンドライン引数または引数値。エラー ファイル名が必要です。**)	**エラー**	**なし**
174	unknown command-line argument or argument value; include path expected (**不明なコマンドライン引数または引数値。インクルード パスが必要です。**)	**エラー**	**なし**
175	unknown command-line argument or argument value; list file name expected (**不明なコマンドライン引数または引数値。リスト ファイル名が必要です。**)	**エラー**	**なし**
176	unknown command-line argument or argument value; obj file name expected (**不明なコマンドライン引数または引数値。オブジェクト ファイル名が必要です。**)	**エラー**	**なし**

177	unknown command-line argument or argument value; width value expected (**不明なコマンドライン引数または引数値。幅の値が必要です。**)	エラー	なし
178	unknown command-line argument or argument value; length value expected (**不明なコマンドライン引数または引数値。長さの値が必要です。**)	エラー	なし
179	unknown command-line argument or argument value; SET directive expected (**不明なコマンドライン引数または引数値。**SET **ディレクティブが必要です。**)	エラー	なし
180	unknown command-line argument or argument value; <-via> file name expected (**不明なコマンドライン引数または引数値。**<-via> **ファイル名が必要です。**)	エラー	なし
181	wrong string constant; $ character must be doubled (**文字列定数が正しくありません。$ 文字は二重にする必要があります。**)	エラー	なし
182	attribute does not pertain to a relocatable module; ignored, weak expected (**属性が、再配置可能モジュールに属していません。無視されました。弱いエイリアスが必要です。**)	エラー	なし
183	multiple symbol definition or incompatibility; weak alias must be external (**複数のシンボルが定義されているか、互換性がありません。弱いエイリアスは外部に指定する必要があります。**)	エラー	なし
184	multiple symbol definition or incompatibility; procedure name must have address 0 if it is an area COMDAT symbol (**複数のシンボルが定義されているか、互換性がありません。領域の** COMDAT **シンボルの場合、プロシージャ名のアドレスは** 0 **である必要があります。**)	エラー	なし
185	multiple symbol definition or incompatibility; multiple definitions of section COMDAT symbol as function name (**複数のシンボルが定義されているか、互換性がありません。セクションの** COMDAT **シンボルが関数名として複数定義されています。**)	エラー	なし
186	multiple symbol definition or incompatibility; an assembler variable cannot be exported; export attribute deleted (**複数のシンボルが定義されているか、互換性がありません。アセンブラ変数をエクスポートできません。エクスポート属性は削除されました。**)	エラー	なし
187	multiple symbol definition or incompatibility; a register relative symbol cannot be exported; export attribute deleted (**複数のシンボルが定義されているか、互換性がありません。レジスタ相対シンボルをエクスポートできません。エクスポート属性は削除されました。**)	エラー	なし
188	multiple symbol definition or incompatibility; program counter symbol cannot be exported; export attribute deleted (**複数のシンボルが定義されているか、互換性がありません。プログラム カウンタ シンボルをエクスポートできません。エクスポート属性は削除されました。**)	エラー	なし
189	multiple symbol definition or incompatibility; a string symbol cannot be exported; export attribute deleted (**複数のシンボルが定義されているか、互換性がありません。文字列シンボルをエクスポートできません。エクスポート属性は削除されました。**)	エラー	なし
190	multiple symbol definition or incompatibility; truncation possible on a float absolute exported symbol (**複数のシンボルが定義されているか、互換性がありません。エクスポートされた浮動小数点絶対シンボルで切り捨てが生じる可能性があります。**)	エラー	なし

191	illegal symbol %s; a built-in variable cannot be used as a COMDAT symbol; ignored (**無効なシンボル %s。組み込み変数は COMDAT シンボルとして使用できません。無視されました。**)	**エラー**	**シンボル**
192	illegal symbol %s; COMDAT symbol name expected (**無効なシンボル %s。COMDAT シンボル名が必要です。**)	**警告**	**シンボル**
193	illegal symbol %s; AREA attribute expected (**無効なシンボル %s。AREA 属性が必要です。**)	**エラー**	**シンボル**
194	illegal symbol %s; expected : '=' (**無効なシンボル %s。'=' が必要です。**)	**エラー**	**シンボル**
195	illegal symbol %s; has no type (**無効なシンボル %s。種類が指定されていません。**)	**エラー**	**シンボル**
196	illegal symbol %s; wrong type (**無効なシンボル %s。種類が正しくありません。**)	**エラー**	**シンボル**
197	illegal symbol %s; absolute value expected (**無効なシンボル %s。絶対値が必要です。**)	**エラー**	**シンボル**
198	MEND directive before a proper macro definition (**適切なマクロ定義の前に MEND ディレクティブが必要です。**)	**エラー**	**なし**
199	illegal symbol %s; bad macro name (**無効なシンボル %s。マクロ名が正しくありません。**)	**エラー**	**シンボル**
200	illegal symbol %s; < argument > expected (**無効なシンボル %s。< argument > が必要です。**)	**エラー**	**シンボル**

関連項目

その他のリソース

ARM アセンブラ エラー メッセージ

ARM エラー メッセージ 201 ～ 250

次の表に、ARM エラー メッセージ 201 ～ 250 を示します。

番号	エラー メッセージ	警告/エラー	引数
201	illegal symbol %s; null macro formal parameter (シンボル %s は無効です。NULL マクロのフォーマル パラメータが必要です。)	エラー	シンボル
202	illegal symbol %s; initialization string expected (シンボル %s は無効です。初期化文字列が必要です。)	エラー	シンボル
203	illegal macro syntax: %s; comma or end of line expected (マクロ構文 %s は無効です。コンマまたは行末が必要です。)	エラー	行
204	illegal macro syntax: %s; improper component of a macro definition (マクロ構文 %s は無効です。マクロ定義のコンポーネントが不適切です。)	エラー	行
205	illegal symbol %s; macro name multiply defined (シンボル %s が無効です。マクロ名が重複して定義されています。)	エラー	シンボル
206	illegal symbol %s; unknown token (シンボル %s が無効です。不明なトークンです。)	エラー	シンボル
207	illegal symbol %s; duplicate formal parameter; ignored (シンボル %s は無効です。フォーマル パラメータが重複しています。%s は無視されました。)	警告	シンボル
208	wrong expression operand: %d; positive value expected (式のオペランド %d が正しくありません。正の数を指定してください。)	エラー	式の値
209	built-in variable not implemented yet (組み込み変数がまだ実装されていません。)	エラー	なし
210	error while reading input file: input stack overflow, possible infinite recursion on file include or macro calls (入力ファイルの読み取り中にエラーが発生しました。入力スタック オーバーフローです。インクルード ファイルまたはマクロ呼び出しの無限再帰の可能性があります。)	エラー	なし
211	error while reading input file: seek error (入力ファイルの読み取り中のエラー。シーク エラー。)	エラー	なし

212	error while reading input file: line too long on input file (**入力ファイルの読み取り中のエラー。入力ファイルの行が長すぎます。**)	**エラー**	**なし**
213	error while reading input file: macro expanded line too long (**入力ファイルの読み取り中のエラーです。マクロ展開行が長すぎます。**)	**エラー**	**なし**
214	error while reading input file: input stack overflow, possible infinite recursion on file include or macro calls (**入力ファイルの読み取り中にエラーが発生しました。入力スタック オーバーフローです。インクルード ファイルまたはマクロ呼び出しの無限再帰の可能性があります。**)	**エラー**	**なし**
215	malformed escape constant; too many characters (**エスケープ定数の形式が正しくありません。文字が多すぎます。**)	**エラー**	**なし**
216	malformed constant; unknown escape sequence (**定数の形式が正しくありません。不明なエスケープ シーケンスです。**)	**エラー**	**なし**
217	error while generating object file: write error (**オブジェクト ファイルの生成中に書き込みエラーが発生しました。**)	**エラー**	**なし**
218	assertion failed (**アサーション エラー。**)	**エラー**	**なし**
219	the selected architecture and CPU (or mode) are conflicting %s; BX instruction not defined for this architecture (**選択したアーキテクチャと** CPU (**またはモード**) **が競合しています。**%s**。このアーキテクチャに対して** BX **命令が定義されていません。**)	**エラー**	**なし**
220	the selected architecture and CPU (or mode) are conflicting; <Thumb Area option> and <command-line arch: not 4T> (**選択したアーキテクチャと** CPU (**またはモード**) **が競合しています。**<Thumb **領域オプション> と <コマンドラインのアーキテクチャが** 4T **でない>。**)	**警告**	**なし**
221	illegal flag(s); only <cpsr_f> or <spsr_f> can be used with immediate value (**フラグ (複数可) が無効です。即値で使用できるのは** <cpsr_f> **または** <spsr_f> **だけです。**)	**警告**	**なし**
222	operand restriction violation (undefined behavior): same source and dest reg (**オペランドの制限違反です (未定義の動作)。ソース レジスタと出力先レジスタが同じです。**)	**警告**	**なし**
223	operand restriction violation (undefined behavior): same base and offset reg (**オペランドの制限違反です (未定義の動作)。基底レジスタとオフセット レジスタが同じです。**)	**警告**	**なし**
224	mnemonic qualifier is ignored; suffix following <ldr> (**ニーモニック修飾子は無視されます。**<ldr> **の後にサフィックスがあります。**)	**警告**	**なし**
225	command-line option not implemented; %s assumed; %s ignored (**コマンドライン オプションは実装されません。**%s **が必要です。**%s **は無視されました。**)	**警告**	**必要なコマンドライン引数、または実装されていないコマンドライン引数**

226	illegal combination of section flags: section flags cannot be inferred, code and data/uninitialized, readonly/readwrite (セクション フラグの組み合わせが無効です。セクション フラグを推定できません。コードおよびデータ/未初期化、読み取り専用/読み取りまたは書き込み。)	警告	なし
227	syntax error in expression (式の構文エラー。)	エラー	なし
228	assembler internal error: stack overflow in expression (アセンブラ内部エラー。式のスタック オーバーフロー。)	エラー	なし
229	expression; %s (式: %s。)	エラー	式の文字列
230	condition codes are not accepted for this form of blx instruction; cc ignored (条件コードはこの形式の blx 命令では受け入れられません。cc は無視されました。)	警告	なし
231	unknown floating point system register: %s; expected: FPSID, FPSCR or FPEXC (浮動小数点システム レジスタ %s が不明です。FPSID、FPSCR、または FPEXC が必要です。)	エラー	浮動小数点レジスタを含む文字列
232	wrong floating point register list; a compact group of registers is expected (浮動小数点レジスター覧が正しくありません。レジスタの圧縮グループを使用してください。)	エラー	なし
233	the maximum amount of transfer is 16 words (最大転送量は 16 ワードです。)	警告	なし
234	improper line syntax; expected comma (行の構文が不適切です。コンマが必要です。)	エラー	なし
235	Improper syntax: s must follow condition code (構文が不適切です。条件コードの後に s が必要です。)	エラー	なし
236	fldmdb|fstmdb require increment flag on first register (fldmdb|fstmdb には最初のレジスタにインクリメント フラグが必要です。)	エラー	なし
237	The destination register must be even (出力先レジスタは偶数でなければなりません。)	警告	なし
238	The use of r15 (pc) has unpredictable results (r15 (pc) を使用すると予期しない結果が生じる可能性があります。)	警告	なし
239	Accumulator values greater than zero not supported (ゼロを超えるアキュムレータ値はサポートされていません。)	警告	なし

240	cpu argument (%s) not supported (cpu **引数** (%s) **はサポートされていません。**)	**エラー**	**引数**
241	Instruction %s not supported for -cpu %s (**命令** %s **は** -cpu %s **でサポートされていません。**)	**エラー**	**なし**
242	-cpu option overriding -arch option (-cpu **オプションによって** -arch **オプションが上書きされます。**)	**警告**	**なし**
243	The following usage is no longer supported (**次の使用法はサポートされなくなりました。**): `armasm [<options>] sourcefile objectfile\n` Please use the ""-o"" option (**次のように、**""-o""**オプションを使用してください。**): `armasm [<options>] -o objectfile sourcefile\n`	**警告**	**なし**
244	Use of undefined symbol %s in EQU expression (EQU **式で未定義のシンボル** %s **が使用されています。**)	**警告**	**シンボル**
245	Literal pool entry could not be found in second pass. Check to make sure the expression and all dependent symbols are fully defined before their use. (2 **番目のパスにリテラル プール エントリが見つかりません。式をチェックしてすべての従属シンボルが完全に定義されていることを確認してからこれらのシンボルを使用してください。**)	**エラー**	**なし**

関連項目

その他のリソース

ARM アセンブラ エラー メッセージ

Renesas ファミリ プロセッサ

Renesas Technology は、高性能で電力効率の高い RISC マイクロプロセッサを設計、製造しています。

Renesas SuperH SH-4 RISC コアでは、効率を高める 16 ビットの固定命令長、16 個の 32 ビット汎用レジスタによる読み込みおよび保存のアーキテクチャ、および 1 クロック サイクルに 1 命令を実行する 5 段階パイプラインが使用されます。

Renesas では特定のアプリケーションに対して各プロセッサ ファミリが最適化されます。

このセクションのトピック

Renesas マイクロプロセッサの組み込み関数

組み込み関数の参照情報を提供します。

Renesas コンパイラ オプション

Renesas マイクロプロセッサのみに使用できるコンパイラとリンカのオプションの参照情報を提供します。

Renesas SH-4 コーリング シーケンスの仕様

アセンブラ、レジスタ割り当て、およびスタック レイアウトについて説明しています。

関連セクション

デスクトップ コンパイラとデバイス コンパイラの違い

コンパイラ オプションと組み込み関数を比較した情報を記載しています。

Renesas マイクロプロセッサの組み込み関数

Renesas SH-4 マイクロプロセッサには、プログラムの高速化にそのまま使用できる多数の関数が組み込まれています。これらの関数は、数値計算に役立ちます。

組み込み関数を使用するプログラムには関数呼び出しのオーバーヘッドがなく、追加コードが作成されるため、プログラムの規模拡大が可能です。

関数を組み込み (インライン) 形式に置き換えるには、/Oi (組み込み関数の生成) オプションまたは **#pragma** 組み込み関数を使用します。

次の表に、組み込み関数として実装される SH-4 マイクロプロセッサ アセンブリ言語関数を示します。

SH-4 命令	説明
_Convolve	2 つのベクトルの総和を計算します。
_Dot3dVW0	"w" 座標を強制的に 0 にして、3 次元または 4 次元のベクトルのペアの内積を計算します。
_Dot3dVW1	"w" 座標を強制的に 1 にして、3 次元または 4 次元のベクトルのペアの内積を計算します。
_Dot4dV	1 対の 4 次元ベクトルの内積を計算します。
__fmac	2 つの浮動小数点値を掛けて、第 3 の浮動小数点値に加算します。
_LoadMatrix	4 × 4 行列を拡張浮動小数点レジスタ バンクに読み込みます。
__movca	キャッシュ ブロックの割り当てに従って移動します。
_Multiply4dM	4 × 4 の行列に 4 × 4 の行列を掛けます。
_SaveMatrix	拡張浮動小数点レジスタ バンクを 4 × 4 行列に格納します。
_XDMultMatrix	4 × 4 の行列に 4 × 4 の行列を掛けます。
_XDXform3dV	3 次元ベクトルに 3 × 3 の行列を掛けます。
_XDXform4dV	4 次元ベクトルに 4 × 4 の行列を掛けます。
_Xform3dV	3 次元ベクトルに 3 × 3 の行列を掛けます。
_Xform4dV	4 次元ベクトルに 4 × 4 の行列を掛けます。

関連項目

その他のリソース

Renesas ファミリ プロセッサ

__fmac

2 つの浮動小数点値を掛けて、第 3 の浮動小数点値に加算します。

```
Float __fmac(
  float Arg1,
  float Arg1,
  float Arg1
);
```

パラメータ

Arg1

[in] 乗算演算の第 1 オペランド。

Arg2

[in] 乗算演算の第 2 オペランド。

Arg3

[in] *Arg1* と *Arg2* の積が加算される値。

戻り値

Arg1 と *Arg2* の積に *Arg3* を加えた合計。

備考

次のコード例は、_fmac の使用方法を示しています。

```
/*********************************************************/
#include <stdio.h>
#include <shintr.h>
void main()
{
     int i;
     float sum=0;
     float v1[4] = {2.0, 2.0, 2.0, 2.0};
     float v2[4] = {8.0, 8.0, 8.0, 8.0};
         for (i = 0; i < 4; i++)
          sum = __fmac(v1[i], v2[i], sum);

     printf("sum = %f\n", sum);
}
```

この例によって、以下の出力が生成されます。

```
sum = 64.000000
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
__fmac	<shintr.h>	SH-4

関連項目

参照情報

Renesas マイクロプロセッサの組み込み関数

__movca

キャッシュ ブロックの割り当てに従って移動します。

```
void __movca(
  unsigned long value,
  unsigned long* addr
);
```

パラメータ

value

[in] メモリに格納するロングワードを指定します。

addr

[in] アクセス先のメモリ内のアドレス。

戻り値

なし。

備考

次のコード例は、**_moveca** と共に _prefetch を使用する方法を示しています。

```
#include <stdio.h>
#include <shintr.h>

#pragma intrinsic(__prefetch, __movca)
void main()
{
     unsigned long addr[1]={0xffff};
     unsigned long value = 0x100;
//
     printf("before prefetch addr value = %x\n", addr[0]);
//
     __prefetch(addr);
     printf("after prefetch addr value = %x\n", addr[0]);

     if (addr[0] != 0xffff)
          printf("error\n");

     __movca(value, addr);
     printf("after movca addr value = %x\n", addr[0]);

     if (addr[0] != value)
          printf("error\n");
}
```

この例によって、以下の出力が生成されます。

```
before prefetch addr value = ffff
after prefetch addr value = ffff
after movca addr value = 100
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ

__movca	<shintr.h>	SH-4

関連項目

参照情報

Renesas **マイクロプロセッサの組み込み関数**

_prefetch

_Convolve

2 つのベクトルの総和を計算します。

```
Float _Convolve(
  int nelement,
  float* pstart,
  float* pend,
  float* pdata,
  float* pfilter
);
```

パラメータ

nelement

[in] 処理する要素の数。

pstart

[in] データ バッファの先頭へのポインタ。

pend

[in] データ バッファの最後へのポインタ。

pdata

[in] 現在のデータ バッファへのポインタ。

pfilter

[in] フィルタ バッファへのポインタ。

戻り値

2 つのベクトルの総和。

備考

pdata パラメータは、データ バッファ内の任意の位置をポイントすることができます。*pfilter* パラメータは、フィルタ バッファ内の任意の位置をポイントすることができます。*nelement* パラメータは、*pfilter* と *nelement* の合計バッファ サイズを超過できません。

この関数を実装するには、コンパイル時に -Qsh4 -Oi フラグを使用します。

次のコード例は、**_Convolve** の使用方法を示しています。

```
/*************************************************************
_Convolve:  summation of two vector

pstart   | A0 |      pfilter   | X0 |
         | A1 |             | X1   |
         | .. |             | .. |
pdata    | Ai |             | .. |
         | .. |             | .. |
pend     | An |             | Xn |
```

次に例を示します。

```
nelement = 4;
```

```
result =    (Ai * X3) + (Ai+1 * X2) + (Ai+2 * x1) + (Ai+3 * x0)

nelement = n+i;
result =    (Ai * Xn) + (Ai+1 * Xn-1) + .. + (Ai-1 * X0)

*****************************************************************/

#include <stdio.h>
#include <shintr.h>

#include <stdio.h>

void main()
{
   float pdata[5] = {1.0,2.0,3.0,4.0,5.0};
   float filter[5] = {1.0,2.0,3.0,4.0,5.0};
   float output;
   float *pstart = pdata;
   float *pend = pdata+4;

   output = _Convolve(5, pstart, pend, pdata, filter);
   printf("output = %f\n", output);

   output = _Convolve(5, pstart, pend, pdata+1, filter);
   printf("output = %f\n", output);

   output = _Convolve(5, pstart, pend, pdata+2, filter);
   printf("output = %f\n", output);

   output = _Convolve(5, pstart, pend, pdata+3, filter);
   printf("output = %f\n", output);

   output = _Convolve(5, pstart, pend, pdata+4, filter);
   printf("output = %f\n", output);

}
```

出力

```
output = 35.000000
output = 45.000000
output = 50.000000
output = 50.000000
output = 45.000000
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_Convolve	<shintr.h>	SH-4

関連項目

参照情報

Renesas マイクロプロセッサの組み込み関数

_Dot3dVW0

"w" 座標を強制的に 0 にして、3 次元または 4 次元のベクトルのペアの内積を計算します。

```
float _Dot3dVW0(
  float* vector1,
  float* vector2
);
```

パラメータ

Vector1

[in] 3 次元または 4 次元のソース ベクトルへのポインタ。

Vector2

[in] 3 次元または 4 次元の宛先ベクトルへのポインタ。

戻り値

内積によって生成されるスカラ。

備考

ソース ベクトルの 4 番目の座標は、常に強制的に 0 になります。

この関数を実装するには、コンパイル時に /Qsh4 /Oi (組み込み関数を生成) フラグを使用します。

次のコード サンプルは、**Dot3dVW0** を使用して 3 次元または 4 次元のベクトルの内積を計算する方法を示しています。

```
/*********************************************************/
#include <stdio.h>
#include <shintr.h>
void main()
{
     float result;
     float v1[4]={1.0,2.0,3.0,4.0};
     float v2[4]={2.0,3.0,4.0,5.0};
     result = _Dot3dVW0(v1,v2);
/*********************************************************/
     printf("result=%f\n", result);
}
```

この例によって、以下の出力が生成されます。

```
retval=20.000000
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_Dot3dVW0	<shintr.h>	SH-4

関連項目

参照情報

_Dot4dV

_Dot3dVW1

Renesas マイクロプロセッサの組み込み関数

_Dot3dVW1

"w" 座標を強制的に 1 にして、3 次元または 4 次元のベクトルのペアの内積を計算します。

```
float _Dot3dVW1(
  float* vector1,
  float* vector2
);
```

パラメータ

vector1

[in] 3 次元または 4 次元のソース ベクトルへのポインタ。

vector2

[in] 3 次元または 4 次元の宛先ベクトルへのポインタ。

戻り値

内積によって生成されるスカラ。

備考

ソース ベクトルの 4 番目の座標は、常に強制的に 1 になります。

この関数を実装するには、コンパイル時に -Qsh4 /Oi (組み込み関数を生成) フラグを使用します。

次のコード例は、3 次元または 4 次元のベクトルの内積を計算する方法を示しています。

```
/*********************************************************/
#include <stdio.h>
#include <shintr.h>
void main()
{
     float retval;
     float v1[3] = {1.0, 2.0, 3.0};
     float v2[3] = {2.0, 3.0, 4.0};
     retval = _Dot3dVW1(v1, v2);
/*********************************************************/
     printf("retval=%f\n", retval);
}
```

この例によって、以下の出力が生成されます。

```
retval=21.000000
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_Dot3dVW1	<shintr.h>	SH-4

関連項目

参照情報

Renesas マイクロプロセッサの組み込み関数

_Dot4dV

_Dot3dVW0

_Dot4dV

1 対の 4 次元ベクトルの内積を計算します。

```
float _Dot4dV(
  float* vector1,
  float* vector2
);
```

パラメータ

vector1

[in] 4 次元のソース ベクトルへのポインタ。

vector2

[in] 4 次元の宛先ベクトルへのポインタ。

戻り値

内積によって生成されるスカラ。

備考

次のコード例は、4 次元ベクトルの内積を計算する方法を示しています。

```
/*********************************************************/
#include <stdio.h>
#include <shintr.h>
void main()
{
     float retval;
     float v1[4] = {1.0, 2.0, 3.0, 4.0};
     float v2[4] = {2.0, 3.0, 4.0, 5.0};
     retval = _Dot4dV(v1, v2);
     printf("retval=%f\n", retval);
}
```

この例によって、以下の出力が生成されます。

```
retval=40.000000
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_Dot4dV	<shintr.h>	SH-4

関連項目

参照情報

Renesas マイクロプロセッサの組み込み関数

_Dot3dVW0

_Dot3dVW1

_LoadMatrix

4 × 4 行列を拡張浮動小数点レジスタ バンクに読み込みます。

```
float* _LoadMatrix(
  float* matrix
);
```

パラメータ

matrix

[in] 配列のインデックスが [行][列] の値の 4 × 4 行列になるように並べられた浮動小数点値の配列へのポインタ。

戻り値

読み込まれた 4 × 4 行列へのポインタ。

備考

次のコードは、**_LoadMatrix** の使用例を示しています。

```
#include <stdio.h>
#include <shintr.h>

void main()
{
     int i,j;
     float result[4][4];
// ############################################################
     float m1[4][4] =          {1.0,1.0,1.0,1.0,
                               2.0,2.0,2.0,2.0,
                               3.0,3.0,3.0,3.0,
                               4.0,4.0,4.0,4.0};
// ############################################################
     float m2[4][4] =          {2.0,2.0,2.0,2.0,
                               2.0,2.0,2.0,2.0,
                               2.0,2.0,2.0,2.0,
                               2.0,2.0,2.0,2.0};
// ############################################################
   _LoadMatrix(m1);        //Load m1 matrix into XD bank regs
   _XDMultMatrix(m2);      //[m1]x[m2] Saved result into XD bank regs
   _SaveMatrix(result);  //Load XD bank regs into result buffer

     // Print out result
     printf("Result of [m1]x[m2] = \n");
     for (i = 0; i < 4; i++)
     {
          printf("| ");
          for (j =0; j < 4; j++)
               printf("%8.4f ", result[i][j]);
          printf(" |\n");
     }
}
```

この例によって、以下の出力が生成されます。

```
Result of [m1]x[m2] =
|   8.0000   8.0000   8.0000   8.0000  |
|  16.0000  16.0000  16.0000  16.0000  |
|  24.0000  24.0000  24.0000  24.0000  |
|  32.0000  32.0000  32.0000  32.0000  |
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_LoadMatrix	<shintr.h>	SH-4

関連項目

参照情報

Renesas マイクロプロセッサの組み込み関数

_SaveMatrix

_Multiply4dM

4 × 4 の行列に 4 × 4 の行列を掛けます。

```
float* _Multiply4dM(
  float* result,
  float* matrix1,
  float* matrix2
);
```

パラメータ

result

[out] 配列のインデックスが [行][列] の値の 4 × 4 行列になるように並べられた浮動小数点値の配列へのポインタ。この行列は演算の結果を受け取ります。

matrix1

[in] 配列のインデックスが [行][列] の値の 4 × 4 行列になるように並べられた浮動小数点値の配列へのポインタ。

matrix2

[in] 配列のインデックスが [行][列] の値の 4 × 4 行列になるように並べられた浮動小数点値の配列へのポインタ。

戻り値

4 × 4 結果行列へのポインタ。

備考

次のコード例は、_Multiply4DM の使用方法を示しています。

```
void main()
{
   int i,j;
   float result[4][4];
   float m1[4][4] =    {1.0,1.0,1.0,1.0,
                  2.0,2.0,2.0,2.0,
                  3.0,3.0,3.0,3.0,
                  4.0,4.0,4.0,4.0};
   float m2[4][4] =    {2.0,2.0,2.0,2.0,
                  2.0,2.0,2.0,2.0,
                  2.0,2.0,2.0,2.0,
                  2.0,2.0,2.0,2.0};
   Multiply4dM(result, m1, m2);
   printf("Result of [m1]x[m2] = \n");
   for (i = 0; i < 4; i++)
   {
      printf("| ");
      for (j = 0; j < 4; j++)
         printf("%8.4f ",result[i][j]);
      printf(" |\n");
   }
}
```

出力

```
Result of [m1]x[m2] =
|   8.0000   8.0000   8.0000   8.0000  |
|  16.0000  16.0000  16.0000  16.0000  |
```

```
|  24.0000  24.0000  24.0000  24.0000  |
|  32.0000  32.0000  32.0000  32.0000  |
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_Multiply4dM	<shintr.h>	SH-4

関連項目

参照情報

Renesas マイクロプロセッサの組み込み関数

_XDMultMatrix

_SaveMatrix

拡張浮動小数点レジスタ バンクを 4 × 4 行列に格納します。

```
float* _SaveMatrix(
  float* matrix
);
```

パラメータ

matrix

[out] **配列のインデックスが [行][列] の値の 4 × 4 行列になるように並べられた浮動小数点値の配列へのポインタ。**

戻り値

格納された 4 × 4 行列へのポインタ。

備考

次のコードに、_SaveMatrix の使用例を示します。

```
void main()
{
   int i,j;
   float result[4][4];
   float m1[4][4] =    {1.0,1.0,1.0,1.0,
                  2.0,2.0,2.0,2.0,
                  3.0,3.0,3.0,3.0,
                  4.0,4.0,4.0,4.0};

   float m2[4][4] =    {2.0,2.0,2.0,2.0,
                  2.0,2.0,2.0,2.0,
                  2.0,2.0,2.0,2.0,
                  2.0,2.0,2.0,2.0};

   _LoadMatrix(m1);      //Load m1 matrix into XD bank regs
   _XDMultMatrix(m2);   //[m1]x[m2] -> result saved into XD bank regs
   _SaveMatrix(result);      //Load XD bank regs into result buffer

   //
   // Print out the result
   //
   printf("Result of [m1]x[m2] = \n");
   for (i = 0; i < 4; i++)
   {
         printf("| ");
         for (j = 0; j < 4; j++)
            printf("%8.4f ",result[i][j]);
         printf(" |\n");
   }

   _XDMultMatrix(m2);   //[m1]x[m2]x[m2]->saved results into XD bank
   _SaveMatrix(result);   //Load XD bank regs into result buffer

   //
   // Print out the result
   //
   printf("\nResult of [m1]x[m2]x[m2] = \n");
   for (i = 0; i < 4; i++)
      {
         printf("| ");
         for (j = 0; j < 4; j++)
            printf("%8.4f ",result[i][j]);
         printf(" |\n");
```

```
    }
}
```

出力

```
Result of [m1]x[m2] =
|   8.0000   8.0000   8.0000   8.0000  |
|  16.0000  16.0000  16.0000  16.0000  |
|  24.0000  24.0000  24.0000  24.0000  |
|  32.0000  32.0000  32.0000  32.0000  |

Result of [m1]x[m2]x[m2] =
|  64.0000  64.0000  64.0000  64.0000  |
| 128.0000 128.0000 128.0000 128.0000  |
| 192.0000 192.0000 192.0000 192.0000  |
| 256.0000 256.0000 256.0000 256.0000  |
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_SaveMatrix	<shintr.h>	SH-4

関連項目

参照情報

Renesas マイクロプロセッサの組み込み関数

_LoadMatrix

_XDMultMatrix

4 × 4 の行列に 4 × 4 の行列を掛けます。この命令では、拡張浮動小数点レジスタ バンクに行列の 1 つをあらかじめ読み込んでおく必要があります。パラメータとして渡された行列が演算の結果を受け取ります。

```
void _XDMultMatrix(
  float* matrix
);
```

パラメータ

matrix

[in, out] **配列のインデックスが** [行][列] **の値の** 4 × 4 **行列になるように並べられた浮動小数点値の配列へのポインタ。**

戻り値

なし。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_XDMultMatrix	<shintr.h>	SH-4

関連項目

参照情報

Renesas **マイクロプロセッサの組み込み関数**

_Multiply4dM

_LoadMatrix

_SaveMatrix

_XDXform3dV

3 次元ベクトルに 3 × 3 の行列を掛けます。この命令では、対象の行列を拡張浮動小数点レジスタ バンクに前もって読み込んでおく必要があります。

```
float* _XDXform3dV(
  float* src,
  float* dst
);
```

パラメータ

src

[in] 3 次元のソース ベクトルへのポインタ。

dst

[out] 3 次元の宛先ベクトルへのポインタ。

戻り値

3 次元の宛先ベクトルへのポインタ。

備考

次のコードは、**_XDXform3dV** の使用方法を示しています。

```
void print_matrix(float *matrix, float *src, float *dest)
{
    int row, col;

    printf("Matrix:\t\t\tSrc:\tDest:\n");

    for (row = 0; row < 3; row++)
    {
        printf("| ");
        for (col = 0; col < 3; col++)
        {
            printf("%6.2f", *matrix++);

        }
        printf(" |");
        printf(" |%6.2f|", *src++);
        printf(" |%10.4f|\n", *dest++);
    }

}

void main()
{
    int i;
    float dest[6];

    float src[6] =    {1.0,2.0,3.0,
               4.0,5.0,6.0,};

   float matrix[16]=   {1.0, 2.0,  3.0,  4.0,
                  5.0,  6.0,  7.0,  8.0,
                  9.0,  10.0, 11.0, 12.0,
                  13.0, 14.0, 15.0, 16.0};

    _LoadMatrix(matrix); //Load matrix into XD bank registers

    for (i = 0; i < 6; i +=3)
```

```
        _XDXform3dV(src+i, dest+i);   //[matrix]x[src[i]] --> dest[i]

    print_matrix(matrix, src, dest);
    printf("\n");
    print_matrix(matrix, src+3, dest+3);
}
```

出力

```
Matrix:                   Src:      Dest:
|   1.00  2.00  3.00 |   |  1.00|   |   14.0000|
|   4.00  5.00  6.00 |   |  2.00|   |   38.0000|
|   7.00  8.00  9.00 |   |  3.00|   |   62.0000|

Matrix:                     Src:      Dest:
|   1.00  2.00  3.00 |   |  4.00|   |   32.0000|
|   4.00  5.00  6.00 |   |  5.00|   |   92.0000|
|   7.00  8.00  9.00 |   |  6.00|   |  152.0000|
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_XDXform3dV	<shintr.h>	SH-4

関連項目

参照情報

Renesas マイクロプロセッサの組み込み関数
_Xform3dV
_Xform4dV
_XDXform4dV
_LoadMatrix
_SaveMatrix

_XDXform4dV

4 次元ベクトルに 4 × 4 の行列を掛けます。この命令では、対象の行列を拡張浮動小数点レジスタ バンクに前もって読み込んでおく必要があります。

```
float* _XDXform4dV(
  float* src,
  float* dst
);
```

パラメータ

src

[in] 3 次元のソース ベクトルへのポインタ。

dst

[out] 3 次元の宛先ベクトルへのポインタ。

戻り値

4 次元の宛先ベクトルへのポインタ。

備考

次のコードは **_XDXform4dV** の使用方法を示しています。

```
#pragma intrinsic(_XDXform4dV, _LoadMatrix)
void print_matrix(float *matrix, float *src, float *dest)
{
       int row, col;
       printf("Matrix:\t\t\t\tSrc:\tDest:\n");

       for (row = 0; row < 4; row++)
    {
           printf("| ");
           for (col = 0; col < 4; col++)
             printf("%6.2f", *matrix++);
           printf(" |");
           printf(" |%6.2f|", *src++);
           printf("\t|%10.4f|\n", *dest++);
       }
}

void main()
{
       int i;
       float dest[8];

   float src[8] =    {1.0, 2.0 ,3.0, 4.0,
              5.0, 6.0, 7.0, 8.0};

   float matrix[16]={  1.0,  2.0,  3.0,   4.0,
                  5.0,  6.0,  7.0,   8.0,
                  9.0,  10.0,    11.0, 12.0,
                  13.0, 14.0,    15.0, 16.0};

   LoadMatrix(matrix);       //Load matrix into XD bank registers

       for (i = 0;  i < 8;  i +=4)
         XDXform4dV(src+i, dest+i);     //[matrix]x[src[i]] --> dest[i]

       print_matrix(matrix, src, dest);
```

```
        printf("\n");
        print_matrix(matrix, src+4, dest+4);
}
```

出力

```
Matrix:                          Src:         Dest:
|   1.00 2.00   3.00 4.00 |     |  1.00 |     |  30.0000|
|   5.00 6.00   7.00 8.00 |     |  2.00 |     |  70.0000|
|   9.00 10.00 11.00 12.00 |    |  3.00 |     |  110.0000|
|  13.00 14.00 15.00 16.00 |    |  4.00 |     |  150.0000|
Matrix:                           Src:        Dest:
|  1.00   2.00  3.00  4.00 |     |  5.00|     |   70.0000|
|  5.00   6.00  7.00  8.00 |     |  6.00|     |  174.0000|
|  9.00    10.00 11.00 12.00 |   |  7.00|     |  278.0000|
|  13.00 14.00 15.00 16.00 |     |  8.00|     |  382.0000|
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_XDXform4dV	<shintr.h>	SH-4

関連項目

参照情報

Renesas マイクロプロセッサの組み込み関数
_Xform3dV
_Xform4dV
_LoadMatrix
_SaveMatrix

_Xform3dV

3 次元ベクトルに 3 × 3 の行列を掛けます。この組み込み関数は、Bank1 浮動小数点レジスタ (**fr12**、**fr13**、**fr14**、**fr15**) と Bank0 浮動小数点レジスタ (**fr3**) に定数 0 を読み込みます。

```
float* _Xform3dV(
  float* dst,
  float* src,
  float* matrix
);
```

パラメータ

dst

[out] 3 次元の宛先ベクトルへのポインタ。

src

[in] 3 次元のソース ベクトルへのポインタ。

matrix

[in] 配列のインデックスが [行][列] の値の 3× 3 行列になるように並べられた浮動小数点値の配列へのポインタ。

戻り値

3 次元の宛先ベクトルへのポインタ。

備考

次のコードは、**_Xform3dV** の使用方法を示しています。

```
void print_matrix(float *matrix, float *src, float *dest)
{
    int row, col;

    printf("Matrix:\t\t\tSrc:\tDest:\n");
    for (row = 0; row < 3; row++)
    {
        printf("| ");
        for (col = 0; col < 3; col++)
        {
            printf("%6.2f", *matrix++);
        }
        printf(" |");
        printf(" |%6.2f|", *src++);
        printf("\t|%10.4f|\n", *dest++);
    }
}

void main()
{
    int i;
    float dest[3];
    float src[3] = {1.0, 2.0, 3.0}

    float matrix[9]={1.0, 2.0, 3.0,
                     4.0, 5.0, 6.0,
                     7.0, 8.0, 9.0};

      _Xform3dV(dest, src, matrix);   // [matrix]x[src[i]]  Edest[i]
      print_matrix(matrix, src, dest);
}
```

```
Output
Matrix:                  Src:    Dest:
|   1.00  2.00  3.00 | |  1.00| |   14.0000|
|   4.00  5.00  6.00 | |  2.00| |   32.0000|
|   7.00  8.00  9.00 | |  3.00| |   50.0000|
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_Xform3dV	<shintr.h>	SH-4

関連項目

参照情報

Renesas マイクロプロセッサの組み込み関数
_Xform4dV
_XDXform4dV
_XDXform3dV
_LoadMatrix
_SaveMatrix

_Xform4dV

4 次元ベクトルに 4 x 4 の行列を掛けます。

```
float* _Xform4dV(
  float* dst,
  float* src,
  float* matrix
);
```

パラメータ

dst

[out] 4 次元の宛先ベクトルへのポインタ。

src

[in] 4 次元のソース ベクトルへのポインタ。

matrix

[in] 配列のインデックスが [行][列] の値の 4 × 4 行列になるように並べられた浮動小数点値の配列へのポインタ。

戻り値

4 次元の宛先ベクトルへのポインタ。

備考

次のコードは、**_Xform4dV** の使用方法を示しています。

```
void print_matrix(float *matrix, float *src, float *dest)
{
    int row, col;

    printf("Matrix:\t\t\t\tSrc:\tDest:\n");

    for (row = 0; row < 4; row++)
    {
        printf("| ");
        for (col = 0; col < 4; col++)
        {
            printf("%6.2f", *matrix++);

        }
        printf(" |");
        printf(" |%6.2f|", *src++);
        printf("\t|%10.4f|\n", *dest++);
    }

}

void main()
{
    int i;
    float dest[4];

    float src[4] = {1.0,2.0,3.0,4.0};

    float matrix[16]={ 1.0,  2.0,  3.0,  4.0,
                       5.0,  6.0,  7.0,  8.0,
                       9.0, 10.0, 11.0, 12.0,
                      13.0, 14.0, 15.0, 16.0};
```

```
    _Xform4dV(dest, src, matrix); //[matrix]x[src[i]] --> dest[i]

    print_matrix(matrix, src, dest);

}
Output
Matrix:                          Src:    Dest:
|   1.00  2.00  3.00  4.00 | |  1.00|   |   30.0000|
|   5.00  6.00  7.00  8.00 | |  2.00|   |   70.0000|
|   9.00 10.00 11.00 12.00 | |  3.00|   |  110.0000|
|  13.00 14.00 15.00 16.00 | |  4.00|   |  150.0000|
```

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_Xform4dV	<shintr.h>	SH-4

関連項目

参照情報

Renesas **マイクロプロセッサの組み込み関数**

_Xform3dV

_XDXform4dV

_XDXform3dV

_LoadMatrix

_SaveMatrix

Renesas コンパイラ オプション

Renesas コンパイラでは、以下のコマンドライン オプションがサポートされています。

- -QSfast、-QSfastd – **下位互換性の提供**
- -Qsh4
- -QSimplicit-import – **ヘルパーのインポートの無効化**

関連項目
概念
固有のビルド オプション

-QSfast、-QSfastd – 下位互換性の提供

オプション **-QSfast** および **-QSfastd** は下位互換性を提供します。既定は **-QSfastd** です。

デバイス コンパイラは、非正規化オペランドの使用も含めて、IEEE に完全に準拠したハードウェアの動作に対応しています。このため、デバイス コンパイラでは、既定で単精度と倍精度双方の浮動小数点の演算に Renesas SuperH SH-4 ハードウェア命令を使用できます。

-QSfast オプションを使用すると、コンパイラは浮動小数点の演算すべてに対して CRT エミュレーション ルーチンを呼び出します。

-QSfast オプションを使用すると、コンパイラで単精度の SH-4 浮動小数点命令を使用することができます。また、コンパイラが倍精度浮動小数点定数を検出すると警告が生成されます。

倍精度定数は演算の種類を単精度から倍精度に不必要に昇格させる場合があるため、パフォーマンスが重要な場合は倍精度定数を使用しないでください。

浮動小数点定数は倍精度を既定とするため注意が必要です。単精度浮動小数点定数は、3.14159f のように浮動小数点サフィックス f または F を使用すると指定できます。

既定の **-QSfast** オプションを使用すると、コンパイラで単精度および倍精度の SH-4 浮動小数点命令を使用することができます。

-QSfastd を指定した場合、コンパイラでは倍精度定数の警告が生成されません。**-QSfastd** コマンドライン オプションを使用すると、コンパイラから生成されるコードが少し大きくなる場合があります。

関連項目

参照情報

Renesas コンパイラ オプション

-Qsh4

このオプションは、Renesas SH-4 **マイクロプロセッサのコンパイルを指定します。**

関連項目

参照情報

Renesas コンパイラ オプション

-QSimplicit-import – ヘルパーのインポートの無効化

このオプションは、静的にリンクされたヘルパー関数を使用するようコンパイラに強制します。コンパイラのヘルパー関数は通常、DLL 内にあります。

これらの関数を呼び出すときのサンク オーバーヘッドを減らすために、コンパイラでは dllimport 呼び出し機構が使用されます。OS またはドライバ コードがヘルパー関数と静的にリンクされている場合は、**-QSimplicit-import**- フラグを使用して、通常の呼び出し機構を使用するようコンパイラに強制します。

関連項目

参照情報

Renesas コンパイラ オプション

Renesas SH-4 コーリング シーケンスの仕様

Renesas SuperH SH-4 コーリング シーケンスの仕様は、SH-4 マイクロプロセッサのコンパイラとアセンブリ言語プログラムの開発の指針を示します。

また、この標準では、ツール、デバッガ、および呼び出しスタックの自動スキャンを実行するオペレーティング システム ユーティリティを開発することもできます。

このセクションのトピック

Renesas SH-4 レジスタ

レジスタ割り当てとパラメータの受け渡し規則について説明します。

Renesas SH-4 スタック フレームのレイアウト

スタック フレームのレイアウトについて説明します。

Renesas SH-4 プロローグおよびエピローグ

アンワインディング用にスタック フレームを設定する際のコード シーケンスの仕様について説明します。

関連セクション

Renesas ファミリ プロセッサ

Visual Studio のこのリリースでサポートされる Renesas マイクロプロセッサのコンパイラ オプション、組み込み関数、および呼び出し仕様の概要を示します。

Renesas SH-4 レジスタ

Renesas SuperH SH-4 には、16 個の汎用レジスタが備えられています。次の表は、これらのレジスタに割り当てられた役割を示しています。

レジスタ	説明
R0	アセンブリ言語の疑似命令の展開時に一時レジスタとして機能し、関数の戻り値を格納します。 R0 はまた、バイト演算および 16 ビット演算における暗黙的なソースまたは出力先としても機能します。 保存されません。
R1 ～ R3	一時レジスタとして機能します。 保存されません。
R4 ～ R7	整数の最初の 4 ワードと非スカラ入力引数を格納します。引数作成領域は、R4 ～ R7 に格納しきれない引数を保管する領域を提供します。 保存されません。
R8 ～ R13	永続レジスタとして機能します。 保存されます。
R14	既定のフレーム ポインタとして機能します。他の永続レジスタをフレーム ポインタとして使用することもできます。また、リーフ ルーチンでは、一時レジスタをフレーム ポインタとして使用することができます。 保存されます。
R15	スタック ポインタまたは永続レジスタとして機能します。 保存されます。

SH-4 は、16 個の浮動小数点レジスタに Bank0 および Bank1 と指定されている 2 つのバンクを備えています。この指定では、Bank1 の使用は定義されません。この呼び出し規則では、SH-4 Bank0 浮動小数点レジスタに次の役割が割り当てられます。

倍精度レジスタ	単精度レジスタ	説明
DR0	FR0 FR1	関数の戻り値を格納します。DR0 は、単精度レジスタ FR0 と FR1 のペアとしての別名です。
DR2	FR2 ～ FR3	一時レジスタとして機能します。DR2 は、FR2 と FR3 のペアとしての別名です。 保存されません。
DR4 ～ DR10	FR4 ～ FR11	単精度または倍精度の浮動小数点引数を格納します。引数作成領域は、浮動小数点レジスタに格納しきれない引数を保管する領域も提供します。
DR12 ～ DR14	FR12 ～ FR15	永続レジスタとして機能します。 保存されます。

浮動小数点のステータス制御は、一部の浮動小数点命令の動作に影響を与えます。プロローグとエピローグ内の PR、SZ、および FR ビットの使用の詳細については、「Renesas SH-4 プロローグおよびエピローグ」を参照してください。

関連項目

概念

Renesas SH-4 スタック フレームのレイアウト

その他のリソース

Renesas SH-4 スタック フレームのレイアウト

Renesas SuperH SH-4 スタック フレームのレイアウトでは、4 つの指定領域を使用して、関数で使用されるレジスタ領域と変数用の領域が保持されます。

- **レジスタ保存領域** (RSA) には、関数によって使用される永続レジスタすべての保存値が格納されます。また、関数のリターン アドレスも格納されます。
- **ローカルおよび一時** 領域は、ローカル変数およびコンパイラによって生成された一時レジスタに割り当てられるスタック領域を表します。
- **alloca()** ローカル領域は関数の実行時に alloca() 組み込み関数を使用することによって動的に割り当てられます。コンパイラによって生成されるコードを使用して、出力引数の構造体を管理する領域を動的に割り当てることもできます。
- **出力引数** 領域は、レジスタに渡されるすべての引数を含め、別の関数を呼び出す際に渡されるすべての引数を保持できるだけの大きさが必要です。この領域は、alloca() の規則が監視されている限りは、呼び出し前に動的に割り当てたり、拡張したりできます。

SH-4 のフレーム ポインタとスタック ポインタ

次の一覧は、スタック ポインタとフレーム ポインタに関する詳細情報です。

- スタック ポインタとフレーム ポインタのアドレスは、4 バイト境界に配置されます。
- ルーチンで alloca() ローカルが指定されている場合、または他の理由によってスタック フレーム領域が動的に割り当てられる場合、別のフレーム ポインタ レジスタが入力引数およびローカルにアクセスします。リーフ ルーチンでは任意の空き整数レジスタをフレーム ポインタとして使用できます。非リーフ ルーチンでは永続レジスタを使用する必要があります。
- 通常、別のフレーム ポインタを確立するルーチンでは R14 を使用します。ただし、大規模なスタック フレーム内のデータへのアクセスをより効率よくできるように、ルーチンでフレーム ポインタをもう 1 つ確立できます。ルーチンでフレーム ポインタを確立しない場合は、R15 をプロローグの終了からエピローグの開始までの間、変更なく保持する必要があります。

関連項目

概念

Renesas SH-4 レジスタ

その他のリソース

Renesas SH-4 プロローグおよびエピローグ

Renesas SH-4 プロローグおよびエピローグ

Renesas マイクロプロセッサに構造化例外処理 (SEH) を実装するには、Renesas SuperH SH-4 プロローグおよびエピローグのコード セグメントが必要です。詳細については、「RISC 環境の SEH」を参照してください。

プロローグには、ルーチンのスタック フレームを設定するコードが格納されます。エピローグには、ルーチンのフレームを削除してから呼び出し元関数に戻るコードが格納されます。

このセクションのトピック

SH-4 プロローグ

SH-4 プロローグ シーケンスの必要条件について説明します。

SH-4 エピローグ

SH-4 エピローグ シーケンスの必要条件について説明します。

SH-4 プロローグおよびエピローグの例

プロローグ シーケンスとエピローグ シーケンスのペアの例を示します。

SH-4 プロローグ

SH-4 プロローグには、間に命令が入ることなく直接つながった 4 つの部分が格納されます。次の一覧に、これらの必要な部分を示します。

1. R4 ～ R7 および FR4 ～ FR11 の入力引数の値を引数のホーム ロケーションに保存する 0 個以上の命令のシーケンス。
   通常、このシーケンスでは、基底アドレス レジスタとして R15 を使った fmov 命令が使用されます。このシーケンスでは、スタック ポインタ R15 は変更されません。
   また、レジスタ間接および前置デクリメント レジスタ間接アドッシング モードで fmov 命令を使用して、浮動小数点引数レジスタをスタックに保存することができます。
   1 つ以上のこのような fmov 命令のシーケンスの前には、汎用レジスタへの一定の移動と、それに続くレジスタへの sp の追加の 2 つの命令から成るアドレス レジスタ セットアップ シーケンスを使用することができます。次はその例です。

   ```
   mov #36, r0
   add sp, r0
   fmov fr5, @r0
   fmov fr4, @-r0
   ```

2. アドレス レジスタとして R15 を使ったプリデクリメント付きレジスタ間接アドレッシング モードを使用して、保存対象のすべての永続レジスタと、リターン アドレス (PR) 保存する場合はこれをスタックにプッシュする 0 個以上の命令のシーケンス。
   R14 などの永続レジスタをフレーム ポインタとして使用する場合は、それを最初にプッシュします。
   PR を保存する場合は、それを最後にプッシュします。
   プロローグでは、mov.l 命令を使用してリターン アドレス以外のレジスタを保存し、sts.l 命令を使用してリターン アドレスを保存します。
   また、プロローグでは、fmov.s 命令を使用して浮動小数点レジスタを保存します。さらに、最初に FPSCR で SZ ビットを設定することによって、倍精度および移動拡張 fmov 命令を呼び出す場合もあります。
3. フレーム ポインタを設定する 1 個以上の命令のシーケンス (フレーム ポインタを確立する場合)。
   フレーム ポインタを設定するには、まずステップ 3 のシーケンスで R15 の内容をフレーム ポインタ レジスタにコピーし、次に、フレーム ポインタ アドレスに対するオフセット量をフレーム ポインタ レジスタに追加するか、またはレジスタから差し引きます。
   フレーム ポインタ アドレスが R15 の現在の値より少ない場合、プロローグでフレーム ポインタ アドレスに対するオフセットを R15 から差し引き、R15 をフレーム ポインタ レジスタにコピーします。この方法を使用する場合、その後ルーチンでは R15 に対するデクリメントのサイズを考慮する必要があります。
   ルーチンがリーフ ルーチンでない場合、関数ではフレーム ポインタとして永続レジスタ (通常は R14) を使用する必要があります。
4. R15 から 4 バイトの整列オフセットを差し引くことによって、ローカル変数、コンパイラによって生成される一時レジスタ、および引数作成領域用に残りのスタック フレーム領域を割り当てる 0 個以上の命令のシーケンス。

仮想アンワインダは、このシーケンスの最後の命令をプロローグの最後の命令と見なします。

必要に応じて、一時レジスタ R1 には、大きすぎて加算命令の即値フィールドに表すことができないオフセットが保持されます。

関連項目

概念

SH-4 エピローグ
SH-4 プロローグおよびエピローグの例

その他のリソース

Renesas SH-4 プロローグおよびエピローグ

SH-4 エピローグ

SH-4 エピローグは、保存済み永続レジスタを復元し、スタック ポインタを関数エントリの値にリセットし、関数の呼び出し関数に戻る連続した命令のシーケンスです。

また、特に指定がない限り、**仮想アンワインダ**からエピローグに次の命令を含めるよう求められます。

1. フレーム ポインタを増分する 1 つの加算命令。この命令は、項目 2 (省略可能) で定義されている命令シーケンスの直前に挿入する必要があります。
2. 命令の出力先オペランド、または命令のポストインクリメント付きメモリ アドレス オペランドで参照することによって **R15** を変更する命令シーケンス。このシーケンスは、項目 3 の rts 命令の直前に挿入します。このシーケンスの命令はすべて、**lds**、**mov**、**fmov**、**fschg**、または **add** 命令である必要があります。
   この項目は省略可能です。これは、復元する RSA 領域を持たない関数には記述しません。
3. **rts** 命令とその遅延スロット命令。**rts** の遅延スロット命令は、エピローグの一部とみなされますが、項目 2 で示した規則に準拠する必要はありません。

関数をアンワインドする場合、**仮想アンワインダ**では、現在実行している命令がプロローグまたはエピローグにあるかどうかを判断する必要があります。現在のコントロール ポイントがプロローグまたはエピローグにある場合、**仮想アンワインダ**では、関数をアンワインドする特別な手段を講じる必要があります。

仮想アンワインダでは、次の 3 つの条件を守る必要があります。

- 関数でフレーム ポインタを確立する場合、その関数は、最後のプロローグ命令が完了してから、最初のエピローグ命令が開始するまでに、フレーム ポインタ値を変更できません。
  関数でフレーム ポインタを確立しない場合、その関数は、最後のプロローグ命令が完了してから、最初のエピローグ命令が開始するまでに、**R15** の値を変更できません。
- スタック ポインタ (常に **R15**) に含まれるアドレスは、レジスタ保存領域内の復元されていないレジスタ値の最下位アドレスを上回ることはできません。
- **FPSCR** レジスタの **SZ**、**PR** および **FR** ビットは、プロローグおよびエピローグを開始、終了する場合、常に 0 に設定する必要があります。**PR** および **FR** ビットは、プロローグまたはエピローグ内で変更できません。この条件により、仮想アンワインダでは浮動小数点命令を正しく逆実行できます。

関連項目

概念

SH-4 プロローグ
SH-4 プロローグおよびエピローグの例

その他のリソース

Renesas SH-4 プロローグおよびエピローグ

SH-4 プロローグおよびエピローグの例

次のコード例は、プロローグとエピローグを使用して特定のタスクを実行する方法を示しています。

- R14、R8、および PR を保存するためのスタック フレームの割り当て
  この例ではスタック フレームを割り当てて、R14、R8、および PR を保存し、alloca() 呼び出しを許可します。また、4 ワードの引数作成領域も割り当てます。ローカル変数および一時レジスタにはスタック領域は必要ありません。

```
// Prolog

NESTED_ENTRY Function

mov.l R14, @-R15 // Save old frame pointer.
mov.l R8, @-R15 // Save a permanent register.
sts.l PR, @-R15 // Save return address.
mov  R15, R14  // Set up new frame pointer.
add  #12, R14
add  #-16, R15  // Allocate argument save area

PROLOG_END

// Routine body


// Epilog

add  #-12, R14  // Find base of RSA.
mov  R14, R15
lds.l @R15+, PR  // Restore return address.
mov.l @R15+, R8  // Restore R8.
rts        // Return.
 mov.l @R15+, R14 // Restore R14.

ENTRY_END Function
```

- リーフ ルーチンのスタック フレームの割り当て
  この例では、ローカル変数および一時レジスタ用に 40 バイトを必要とするリーフ ルーチンのスタック フレームを割り当てます。ここでは、永続レジスタ R8、R9、および R10 が使用されます。このルーチンには _alloca ローカル変数が含まれないため、フレーム ポインタは不要です。

```
//  Prolog

NESTED_ENTRY Function

mov.l R8, @-R15
mov.l R9, @-R15
mov.l R10, @-R15
add  #-40, R15

PROLOG_END

// Routine body

add  #40, R15
mov.l @R15+, R10
mov.l @R15+, R9
rts
```

```
    mov.l @R15+, R8

    ENTRY_END Function
```

関連項目

参照情報

SH-4 アセンブラ マクロ

その他のリソース

Renesas SH-4 プロローグおよびエピローグ

SH-4 アセンブラ マクロ

SH-4 アセンブラ レベルのマクロは、プロローグおよびエピローグのコード シーケンスの実装に必要です。

次の表に SH-4 マイクロプロセッサ用に定義されたマクロを示します。

マクロ名	説明
ALTERNATE_ENTRY (SH-4)	ルーチンの代替エントリを宣言します。
END_REGION (SH-4)	一連のテキストまたはデータ範囲の最後を指定します。
ENTRY_END (SH-4)	前の NESTED_ENTRY によって指定されたルーチンを終了します。
EXCEPTION_HANDLER (SH-4)	指定された例外ハンドラを次に続く NESTED_ENTRY と関連付けます。
EXCEPTION_HANDLER_DATA (SH-4)	指定された例外ハンドラとハンドラ データを次に続く NESTED_ENTRY と関連付けます。
LEAF_ENTRY (SH-4)	プロローグ コードを必要としないルーチンの最初を宣言します。
NESTED_ENTRY (SH-4)	既存のスタック フレームがあるか、または新規のスタック フレームを作成するルーチンの最初を宣言します。
PROLOG_END (SH-4)	プロローグ領域の最後を指定します。
START_REGION (SH-4)	一連のテキストまたはデータ範囲の最初を指定します。

関連項目

その他のリソース

Renesas SH-4 コーリング シーケンスの仕様

ALTERNATE_ENTRY (SH-4)

このマクロによって、種類が NESTED_ENTRY (SH-4) または LEAF_ENTRY (SH-4) のルーチンの代替エントリが宣言されます。

```
ALTERNATE_ENTRY Name[,
[Section=]SectionName]
```

パラメータ

Name

Name は、エントリ ポイントです。これにはグローバル名前空間に含まれる名前を使用します。

SectionName

SectionName は、そのエントリが含まれるセクションの名前です。「備考」を参照してください。

戻り値

なし。

備考

ALTERNATE_ENTRY マクロでは、*SectionName* パラメータは使用されません。このパラメータは受け入れられますが、**NESTED_ENTRY** および **LEAF_ENTRY** との整合性を保つために無視されます。

このパラメータを使用する場合は、**ALTERNATE_ENTRY** 呼び出しをルーチンの本体に含める必要があります。

関連項目

参照情報

NESTED_ENTRY (SH-4)
LEAF_ENTRY (SH-4)

END_REGION (SH-4)

このマクロでは、一連のテキストまたはデータ範囲の終了が指定されます。

```
END_REGION NameEnd
```

パラメータ
NameEnd

NameEnd **は、範囲の最後にラベルを付けます。**

戻り値

なし。

備考

NameEnd **にはグローバル名前空間に含まれる名前を使用します。**

関連項目
参照情報
START_REGION (SH-4)

ENTRY_END (SH-4)

このマクロは、NESTED_ENTRY (SH-4) または LEAF_ENTRY (SH-4) で指定されている現在のルーチンを終了します。

```
ENTRY_END [Name]
```

パラメータ
Name

Name はエントリ ポイントです。これにはグローバル名前空間に含まれる名前を使用します。「備考」を参照してください。

戻り値

なし。

備考

Name には、**NESTED_ENTRY** マクロまたは **LEAF_ENTRY** マクロで使用されるものと同じ名前を指定します。

現在、**ENTRY_END (SH-4)** マクロでは *Name* が無視されます。

関連項目
参照情報
NESTED_ENTRY (SH-4)
LEAF_ENTRY (SH-4)

EXCEPTION_HANDLER (SH-4)

このマクロでは、例外ハンドラ *Handler* が、次に続く NESTED_ENTRY (SH-4) または LEAF_ENTRY (SH-4) ルーチンと関連付けられます。

```
EXCEPTION_HANDLER Handler
```

パラメータ
Handler

例外ハンドラの名前。

戻り値

なし。

備考

この関連付けは、対応する ENTRY_END (SH-4) マクロをコンパイラが検出するまで有効です。

関連項目
参照情報
NESTED_ENTRY (SH-4)
LEAF_ENTRY (SH-4)
ENTRY_END (SH-4)

EXCEPTION_HANDLER_DATA (SH-4)

このマクロは、例外ハンドラ *Handler* と *HandlerData* を次に続く NESTED_ENTRY (SH-4) または LEAF_ENTRY (SH-4) ルーチンに関連付けます。

```
EXCEPTION_HANDLER_DATA Handler,
     HandlerData
```

パラメータ

Handler

例外ハンドラの名前。

HandlerData

例外ハンドラに関連付けられた文字列。

戻り値

なし。

備考

この関連付けは、対応する ENTRY_END (SH-4) マクロがコンパイラによって検出されるまで有効です。

関連項目

参照情報

NESTED_ENTRY (SH-4)
LEAF_ENTRY (SH-4)
ENTRY_END (SH-4)
EXCEPTION_HANDLER (SH-4)

LEAF_ENTRY (SH-4)

このマクロでは、プロローグ コードを必要としないルーチンの最初が宣言されます。

```
LEAF_ENTRY Name[,
[Section=]SectionName]
```

パラメータ
Name

Name **は、ルーチン名です。これにはグローバル名前空間に含まれる名前を使用します。**

SectionName

SectionName **は、このエントリが含まれるセクションの名前です。これは省略可能で、既定値は** .text **です。**

戻り値

なし。

備考

LEAF_ENTRY **には、関連付けられた** ENTRY_END (SH-4) **が必要です。**

関連項目
参照情報
ENTRY_END (SH-4)
PROLOG_END (SH-4)

NESTED_ENTRY (SH-4)

このマクロでは、既存のフレームを持つルーチン、またはスタック フレームを作成するルーチンの最初が宣言されます。

```
NESTED_ENTRY Name[,
[Section=]SectionName]
```

パラメータ

Name

Name は、ルーチン名です。これにはグローバル名前空間に含まれる名前を使用します。

SectionName

SectionName は、このエントリが含まれるセクションの名前です。これは省略可能で、既定値は .text です。

戻り値

なし。

備考

NESTED_ENTRY には、PROLOG_END (SH-4) および ENTRY_END (SH-4) を関連付けておく必要があります。

関連項目

参照情報

ENTRY_END (SH-4)
PROLOG_END (SH-4)

PROLOG_END (SH-4)

このマクロは、プロローグ領域の最後を指定します。

```
PROLOG_END
```

パラメータ
なし。

戻り値

なし。

備考

このマクロは、NESTED_ENTRY (SH-4) マクロまたは LEAF_ENTRY (SH-4) マクロに続けて使用する必要があります。

このマクロは、プロローグ領域の後から、対応する ENTRY_END (SH-4) **マクロの前までの間に使用します。**

関連項目
参照情報
NESTED_ENTRY (SH-4)
LEAF_ENTRY (SH-4)
ENTRY_END (SH-4)

START_REGION (SH-4)

このマクロでは、一連のテキストまたはデータ範囲の開始が指定されます。

```
START_REGION NameBegin
```

パラメータ
NameBegin

NameBegin **は範囲の開始を表すラベルです。***NameBegin* **にはグローバル名前空間に含まれる名前を使用します。**

戻り値

なし。

関連項目
参照情報
END_REGION (SH-4)

Renesas SH-4 シリーズ アセンブラ

SH-4 シリーズ アセンブラ (SHASM) は、アセンブリ言語で書かれたソース プログラムを SH マイクロプロセッサで実行できる形式に変換し、結果をオブジェクト モジュールおよびアセンブラー一覧として出力します。SHASM は COFF (Microsoft Common Object File Format) の形式で出力を生成します。

SHASM にはプログラミング環境を改善し、互換性の問題を解決するためのいくつもの拡張機能が搭載されています。

SHASM は Renesas のネイティブ アセンブラによく似ていますが、まったく同じものではありません。たとえば、SHASM で生成されるエラーや警告メッセージには、エラー番号だけでなくエラーに関する説明も含まれています。

このセクションのトピック

SH-4 アセンブラ コマンドライン オプション

SHASM で使用できるコマンドライン オプションについて説明します。

SH-4 アセンブラ ディレクティブ

アセンブラのディレクティブについて説明します。

SH-4 アセンブラ エラー メッセージ

SH-4 アセンブラで表示されるエラー メッセージの一覧を示します。

関連セクション

Renesas SH-4 コーリング シーケンスの仕様

レジスタの割り当ておよびスタックの配置に関する概要とリンクを紹介します。

SH-4 アセンブラ コマンドライン オプション

SH-4 アセンブラを開始するには、ホスト コンピュータのオペレーティング システムが入力待機状態のときに次の形式でコマンドラインを入力します。

```
shasm <input source file> [,<input source file>...][command line options>...]
```

コマンドラインで複数のソース ファイルを指定すると、指定したファイルが指定した順番で連結することにより、アセンブラでアセンブリ処理のユニットが作成されます。このため、.END ディレクティブは最後のファイルにのみ指定してください。

コマンドライン オプションは、- (ハイフン) または / (スラッシュ) のいずれかで開始できます。ファイル名は、スラッシュまたは円記号で指定することができます。

アセンブラで引数をコマンドライン オプションとファイル名のいずれかとして解釈できる場合、その引数はコマンドライン オプションとして解釈されます。たとえば、`shasm /source` と記述してファイル /source.src をアセンブルすると、アセンブラは /source をスイッチとして解釈し、入力ファイルがないことを伝えるメッセージを発行します。

コマンドライン オプションを使用すると、アセンブリ処理の追加の指定を行うことができます。次の表は、SH-4 アセンブラのコマンドライン オプションを示しています。

コマンドライン オプション	説明
-OBJECT	オブジェクト モジュールの出力を指定します。既定値です。
-NOOBJECT	オブジェクト モジュールの出力を抑止します。
-DEBUG	デバッグ情報の出力を指定します。
-NODEBUG	デバッグ情報の出力を抑止します。
-LIST	アセンブラー一覧の出力を指定します。既定値です。
-NOLIST	アセンブラー一覧の出力を抑止します。
-SHOW	プリプロセッサ関数のソース プログラムの出力を指定します。
-NOSHOW	ソース プログラム一覧の、指定したプリプロセッサ関数のソース ステートメントおよびオブジェクト コードの表示行の出力を抑止します。
-LINES	アセンブラー一覧の行数を設定します。
-COLUMNS	アセンブラー一覧の列数を設定します。
-FoOBJPATH	オブジェクトを書き込むパスを設定します。-o=OBJPATH オプションの同義語です。Microsoft ベースのツールとの互換性を提供します。
-wide[_listing]	アセンブラー一覧で 1 行につき 8 バイトまでのマシン コードまたはデータを表示します。既定値は 1 行につき 4 バイトです。
-nowide[_listing]	アセンブラー一覧で 1 行につき 4 バイトまでのマシン コードまたはデータを表示します。これは既定の動作です。
-tab[_expand][=<number>]	アセンブラー一覧のタブ文字を展開します。数値が指定されている場合、その数値はタブ位置の間隔を示します。既定値は 8 です。
-Notab[_expand]	タブ文字を変更せずにアセンブラー一覧に書き込みます。これは既定の動作です。

-maxerr[ors]=<number>	<number> エラーの報告後、アセンブリを中止します。既定値は 100 です。
-Qsh<version>[r<revision>]	_M_SH および _M_SH_REV 定義済みシンボルの設定を制御する SH マイクロプロセッサのバージョンとリビジョンを選択します。既定のバージョンは SH-4 です。
-nologo	ロゴ バナーの実行時にロゴ バナーを出力しないようにします。
-help -usage	コマンドラインの詳細な使用法のメッセージを出力して直ちに終了します。
-D<name>[=<number>]	-DTERM は TERM に相当するなど、同等のものをあらかじめ定義します。
-CPU=SH<version>	アセンブルするソース プログラムのターゲット CPU を選択します。有効なオプションは次のとおりです。 • SH1 • SH2 • SH4

アセンブラー一覧は、アセンブリ処理の結果が出力される一覧です。

.asm や .src ではなく .s で終わるアセンブリ ソース ファイルをコンパイルすると、出力オブジェクト ファイルのファイル名の最後に .obj が連結されます。たとえば、ファイル名が test.s の場合は test.s.obj にアセンブルされます。

リターン コード

アセンブラは、アセンブリ処理が正常に終了したかどうかを伝えるリターン コードを配信します。次の表は、終了時の動作状況を表す戻り値を示しています。

動作状況	戻り値
正常終了	0
警告が発生	0
エラーが発生	2
致命的なエラーが発生	4

関連項目

その他のリソース

Renesas SH-4 シリーズ アセンブラ

SH-4 アセンブラ ディレクティブ

SH-4 アセンブラ (SHASM) は、アセンブラが解釈して実行する多数のアセンブラ ディレクティブを備えています。

このセクションのトピック

SH-4 アセンブラ マクロ ディレクティブ

SH-4 アセンブラには、次のマクロ関数ディレクティブが用意されています。

ディレクティブ	構文	説明
.MACRO	`.MACRO <macro name>[ <formal parameter>[=<default>] [,<formal parameter>...]]`	マクロを定義します。
.ENDM	`ENDM`	マクロ定義の終了を示します。
.EXITM	`.EXITM`	マクロの展開を終了します。

関連項目

その他のリソース

SH-4 アセンブラ ディレクティブ

SH-4 アセンブラ セクションおよびロケーション ディレクティブ

次の表は、セクション割り当ての SH-4 アセンブラ ディレクティブとロケーション カウンタの一覧です。

ディレクティブ	構文	説明
.SECTION	`.SECTION <section name> [,<section type>[, <section attributes>...]]`	セクションを宣言します。
.ORG	`.ORG <location-counter-value>`	ロケーション カウンタの値を設定します。 ロケーション カウンタの値は、前方参照のシンボルを持たない絶対値でなければなりません。
.ALIGN	`.ALIGN <boundary alignment value>`	ロケーション カウンタの値を境界配置の値の倍数に調整します。 境界配置の値は、前方参照のシンボルを持たない絶対値でなければなりません。

また、アセンブラはセクションが宣言されていない場合に既定のセクションを提供します。

次の一覧は、SHASM が既定のセクションを提供するステートメントと命令の種類を示します。

- 実行可能な命令
- データ予約アセンブラ ディレクティブ
- .ALIGN アセンブラ ディレクティブ
- .ORG アセンブラ ディレクティブ
- ロケーション カウンタへの参照
- ラベル フィールドだけで構成されるステートメント

関連項目

その他のリソース

SH-4 アセンブラ ディレクティブ

SH-4 アセンブラ シンボル処理ディレクティブ

次の表は、SH-4 アセンブラのシンボル処理ディレクティブを示しています。

ディレクティブ	構文	説明
.EQU	`<symbol>[:] .EQU <symbol value>`	シンボル値を設定します。.EQU ディレクティブで定義されたシンボルは再定義できません。
.ASSIGN	`<symbol>[:] .ASSIGN <symbol value>`	再定義できるシンボル値を設定します。
.REG	`<symbol>[:] .REG  <register name><symbol>[:] .REG (<register name>)`	レジスタ名のエイリアスを定義します。 .REG で定義されたレジスタ名のエイリアスは再定義できません。
.EXPORT	`.EXPORT <symbol>[,<symbol>...]`	エクスポート シンボルを宣言します。 この宣言により、現在のファイルで定義されたシンボルが他のファイルで参照できるようになります。
.IMPORT	`.IMPORT <symbol>[,<symbol>...]`	インポート シンボルを宣言します。 この宣言により、他のファイルで定義されたシンボルが現在のファイルで参照できるようになります。
.GLOBAL	`.GLOBAL <symbol>[,<symbol>...]`	エクスポート シンボルとインポート シンボルを宣言します。 この宣言により、現在のファイルで定義されたシンボルが他のファイルで参照できるようになり、他のファイルで定義されたシンボルが現在のファイルで参照できるようになります。

関連項目

その他のリソース

SH-4 アセンブラ ディレクティブ

SH-4 アセンブラ データおよびデータ領域ディレクティブ

次の表は、データおよびデータ領域を予約するための SH-4 アセンブラ ディレクティブを示しています。

ディレクティブ	構文	説明
.COMMON	`.COMMON <symbol name>[,<size>]`	既定の初期値 0 を使用してデータ領域を予約します。
.DATA	`[<symbol>[:]] .DATA[.<operation size>] <integer data>[,<integer data>...]`	整数型データを予約します。 データ サイズの記号には次のものがあります。 ● B – バイト ● W – ワード、2 バイト ● L – ロングワード、4 バイト、既定
.DATAB	`[<symbol>[:]] .DATAB[.<operation size>]<block count>,<integer data>`	整数型データ ブロックを予約します。 ブロック数の指定は、前方参照シンボルのない絶対値である必要があります。
.FDATA	`[<symbol>[:]] .FDATA[.<operation size>] <floating point data>[,<floating point data>...]`	浮動小数点型データを予約します。
.FDATAB	`[<symbol>[:]] .FDATAB[.<operation size>] <block count>,<floating point data>`	浮動小数点型データ ブロックを予約します。
.FRES	`[<symbol>[:]] .FRES[.<operation size>] <area count>`	浮動小数点型データ領域を予約します。 領域数の指定は、前方参照シンボルのない絶対値である必要があります。
.SDATA	`[<symbol>[:]] .SDATA <character string>[,<character string>...]`	文字列型データを予約します。
.SDATAB	`[<symbol>[:]] .SDATAB <block count>,<character string>`	文字列型データ ブロックを予約します。
.SDATAC	`[<symbol>[:]] .SDATAC <character string>[,<character string>...]`	文字列型データを長さ付きで予約します。
.SDATACB	`[<symbol>[:]] .SDATACB <block count>,<character string>`	指定した文字列型データ ブロック数を長さ付きで予約します。 長さ付き文字列とは、文字列の長さを示すバイトを先頭に挿入した文字列です。
.SDATAZ	`[<symbol>[:]] .SDATAZ <character string>[,<character string>...]`	0 終端記号付き文字列型データを予約します。

.SDATAZB	[<symbol>[:]] .SDATAZB <block count>,<character string>	指定した文字列型データ ブロック数を 0 終端記号付きで予約します。
.RES	[<symbol>[:]] .RES[.<operation size>] <area count>	データ領域を予約します。 領域数の指定は、前方参照シンボルのない絶対値である必要があります。
.SRES	[<symbol>[:]] .SRES <character string area size>[,<character string area size>...]	文字列型データ領域を予約します。 文字列型データ領域のサイズの指定は、前方参照シンボルのない絶対値である必要があります。
.SRESC	[<symbol>[:]] .SRESC <character string area size>[,<character string area size>...]	文字列型データ領域を長さ付きで予約します。 長さ付き文字列とは、文字列の長さを示すバイトを先頭に挿入した文字列です。
.SRESZ	[<symbol>[:]] .SRESZ <character string area size>[,<character string area size>...]	0 終端記号付き文字列型データ領域を予約します。

.SDATA のディレクティブ ファミリの文字列定数には、次に示すコードを 1 つ以上、任意に組み合わせることができます。

```
"quoted string"
<control char expression>
```

アセンブラでは、.SDATA や関連するディレクティブの文字列比較などのより複雑な式の一部として文字列定数を使用する場合は、かっこで囲む必要があります。

それ以外の場合は、文字列定数は二重引用符のみで囲み、かっこなしで使用できます。

関連項目

その他のリソース

SH-4 アセンブラ ディレクティブ

SH-4 アセンブラ関数定義ディレクティブ

C++ および構造化例外処理の必要条件と、デバッガでのスタック トレースの提供に対応するため、関数の開始および終了位置に関する情報と、各関数のどの部分がその関数のスタック フレームの設定を行っているかに関する情報をオブジェクト ファイルに含める必要があります。

次の表は、関数を定義する SH アセンブラ ディレクティブを示しています。

ディレクティブ	構文	説明
.ENTRY	`<symbol>[:] .ENTRY`	関数の開始を指定します。
.BIGENTRY	`<symbol>[:] .BIGENTRY`	関数の開始を指定します。
.ENDF	`.ENDF`	関数の終了を指定します。
.PROLOG	`.PROLOG`	関数のプロローグ コードの終了を指定します。 関数プロローグは、関数のスタック フレームを設定する、関数の最初に置かれたコードです。 .PROLOG ディレクティブは、関数のプロローグ コードの最後の行と、非プロローグ コードの最初の行との間に挿入する必要があります。
.PDATA	`.PDATA <handler address> [,<handler data>]`	現在の関数の言語固有の例外処理情報を指定します。 .PDATA ディレクティブは適用先の .ENTRY または .BIGENTRY セクションの前に挿入する必要があります。

関連項目

その他のリソース

SH-4 アセンブラ ディレクティブ

SH-4 アセンブラ デバッグ情報ディレクティブ

次の表は、シンボルおよびオブジェクト モジュールの出力を制御する SH-4 アセンブラ ディレクティブを示しています。

ディレクティブ	構文	説明
.OUTPUT	`.OUTPUT <output specifier>[,<output specifier>]`	オブジェクト モジュールとデバッグ情報の出力を制御します。
.DEBUG	`.DEBUG <output specifier>`	シンボリック デバッグ情報の出力を制御します。

デバッグ情報の出力に関する指定は、オブジェクト モジュールが出力されるときにのみ有効です。

次の表は、出力指定子に使用可能な値を示しています。

出力指定子	説明
OBJ (既定)	オブジェクト モジュールが出力されます。
NOOBJ	オブジェクト モジュールは出力されません。
DBG	デバッグ情報がオブジェクト モジュールに出力されます。
NODBG (既定)	デバッグ情報はオブジェクト モジュールに出力されません。

.DEBUG ディレクティブの指定は、オブジェクト モジュールとデバッグ情報の両方が出力されるときにのみ有効です。

関連項目

その他のリソース

SH-4 アセンブラ ディレクティブ

SH-4 アセンブラー一覧ディレクティブ

次の表は、一覧を制御する SH-4 アセンブラ ディレクティブを示しています。

ディレクティブ	構文	説明
.PRINT	.PRINT <output specifier>[,<output specifier>...]	アセンブラー一覧の出力を制御します。 詳細については、「SH-4 アセンブラ PRINT ディレクティブ指定子」を参照してください。
.LIST	.LIST <output specifier>[,<output specifier>...]Output specifier:{ON|OFF|COND | NOCOND|DEF|NODEF| CALL|NOCALL|EXP| NOEXP|CODE|NOCODE}	ソース プログラム一覧の出力を制御します。 詳細については、「SH-4 アセンブラ LIST ディレクティブ指定子」を参照してください。
.FORM	.FORM <size specifier>[,<size specifier>...]	アセンブラー一覧の行数と列数を設定します。 詳細については、「SH-4 アセンブラ FORM ディレクティブ指定子」を参照してください。
.HEADING	.HEADING [<expression>[,<expression>...]]	ソース プログラム一覧のヘッダーを設定します。
.PAGE	.PAGE	ソース プログラム一覧に新規のページを挿入します。
.SPACE	.SPACE[<line count>]	ソース プログラム一覧に空白行を出力します。

アセンブラー一覧の出力に関するコマンドライン オプションの指定には、アセンブラによって優先順位が設定されます。

関連項目

その他のリソース

SH-4 アセンブラ ディレクティブ

SH-4 アセンブラ PRINT ディレクティブ指定子

次の表は、PRINT ディレクティブ出力指定子に使用可能な値を示しています。

出力指定子	アセンブラの処理
LIST	アセンブラー覧が出力されます。
NOLIST (既定)	アセンブラー覧は出力されません。

関連項目

参照情報

SH-4 アセンブラー覧ディレクティブ

SH-4 アセンブラ LIST ディレクティブ指定子

.LIST は、ソース プログラム一覧の出力を制御するアセンブラ ディレクティブです。LIST ディレクティブ出力に選択できるものは、次のとおりです。

- ソース ステートメント
- 条件付きアセンブリおよびマクロ関数に関連したソース ステートメント
- オブジェクト コード行

次の表に、LIST ディレクティブ出力指定子に使用可能な値を示します。

種類	出力 (既定)	出力なし	オブジェクト	説明
1	ON	OFF	ソース ステートメント	このディレクティブに続くソース ステートメント。
2	COND	NOCOND	条件に失敗	条件に失敗した .AIF ディレクティブ ステートメント。
該当なし	DEF	NODEF	定義	マクロ定義ステートメント、.AREPEAT、.AWHILE、.ASSIGNA、および .ASSIGNC。
該当なし	CALL	NOCALL	呼び出し	マクロ呼び出しステートメント、.AIF および .AENDI。
該当なし	EXP	NOEXP	展開	マクロ展開ステートメント、.AREPEAT および .AWHILE。
3	CODE	NOCODE	オブジェクト コード行	ソース ステートメント行を超えているオブジェクト コード行。

.LIST ディレクティブの指定は、アセンブラー一覧が出力されている場合のみ有効です。

関連項目
参照情報
SH-4 アセンブラー一覧ディレクティブ

SH-4 アセンブラ FORM ディレクティブ指定子

以下の仕様によって、アセンブラー覧の行数と列数が決まります。

サイズ指定子	一覧のサイズ
LIN=<line count>	この指定値は、1 ページあたりの行数を設定します。
COL=<column count>	この指定値は、1 行あたりの列数を設定します。

関連項目

参照情報

SH-4 アセンブラー覧ディレクティブ

SH-4 アセンブラ条件付きアセンブリ ディレクティブ

次の表は、条件付きアセンブリ ディレクティブを示しています。

ディレクティブ	構文	説明
.ASSIGNA	<preprocessor variable>[:].ASSIGNA <value>	整数のプリプロセッサ変数を定義します。定義済みの変数を再定義することができます。
.ASSIGNC	<preprocessor variable>[:].ASSIGNCD<character string>	文字のプリプロセッサ変数を定義します。定義済みの変数を再定義することができます。
.AIF.AELSIF.AELSE.AENDI	.AIF <condition1><Source statements assembled if condition1 is satisfied>.AELSIF <condition2><Source statements assembled if condition1 is not satisfied,but condition2 is satisfied>.AELSE<Source statements assembled if neither condition1nor condition2 is satisfied>.AENDI	指定した条件に従ってソース プログラムの一部をアセンブルするかどうかを決定します。 条件が満たされている場合、.AIF より後のステートメントをアセンブルします。 条件が満たされていない場合、.AELSE より後のステートメントをアセンブルします。
.ERROR または .AERROR	.ERROR[<expression>[,<expression>...]]	ユーザーのコードによってアセンブリ時に検出されたエラーを報告します。 代替スペルとして .AERROR も使用できます。
.AREPEAT.AENDR	.AREPEATD<count><Source statements iteratively assembled>.AENDR	.AREPEAT と .AENDR の間にあるソース プログラムの一部のアセンブリを、指定した回数だけ繰り返します。
.AWHILE.AENDW	.AWHILE <condition><Source statements iteratively assembled>.AENDW	指定した条件が満たされている間、.AWHILE と .AENDW の間にあるソース プログラムの一部を繰り返しアセンブルします。
.EXITM	.EXITM	.AREPEAT または .AWHILE の反復展開を終了します。
.ALIMIT	.ALIMITD<maximum count>	プリプロセッサでの AWHILE ディレクティブの展開の上限値を指定します。

関連項目

その他のリソース

SH-4 アセンブラ ディレクティブ

SH-4 アセンブラのその他のディレクティブ

次の表に、SH-4 アセンブラのその他のディレクティブを示します。

ディレクティブ	構文	説明
.RADIX	`.RADIX <radix specifier>`	基数指定のない整数定数に基数を設定します。 特に指定のない限り、整数定数は 10 進数と見なされます。
.WEAK	`.WEAK <weak name>,<default name>[,<type>]`	弱い外部シンボルを宣言します。 弱い外部シンボルとは、リンク時に柔軟性を与えるプログラムの機構の 1 つです。
.END	`.END`	ソース プログラムの最後を宣言します。
.INCLUDE	`.INCLUDE <file name>`	現在のアセンブリに、指定されたファイルを含めます。 ファイル名にはディレクトリを含めることができます。 ファイルが現在のディレクトリで見つからず、絶対パスで指定されていない場合、アセンブラは INCLUDE 環境変数に指定されている各ディレクトリ内を検索します。
.LEN	`.LEN[ ](<character string>)`	文字列の長さを計算します。
.INSTR	`.INSTR[ ](<character string 1>,<character string 2>[,<start position>])`	文字列を検索します。
.SUBSTR	`.SUBSTR[ ](<character string>,<start position>,<extraction length>)`	文字列を抽出します。

次の表に、RADIX の仕様を示します。

Radix の仕様	説明
B (または b)	2 進
Q (または q)	8 進
O (または o)	8 進
D (または d) (既定)	10 進
H (または h)	16 進
X (または x)	16 進

関連項目

その他のリソース

SH-4 アセンブラ ディレクティブ

SH-4 アセンブラ エラー メッセージ

ここでは、SH-4 アセンブラ エラー メッセージについて説明します。

関連項目

その他のリソース

Renesas SH-4 シリーズ アセンブラ

SH-4 アセンブラ エラー メッセージ 1 ～ 41

次の表は、SH-4 アセンブラ エラー メッセージ 1 ～ 41 の一覧です。

メッセージ番号と重要度	メッセージのテキスト	説明
1 致命的	Internal error: %s (内部エラー: %s。)	アセンブラにバグがあると、このメッセージが表示されます。メッセージを正確に報告し、メッセージが表示されたプログラムのサンプルを提供してください。
2 致命的	Too many errors (%d) -- stopping assembly (エラーが多すぎます (%d) -- アセンブリを停止します。)	エラーの最大数を超えています。エラーの最大数を制御するには、-maxerrors コマンドライン オプションを使用します。
3 致命的	%s not yet implemented (%s はまだ実装されていません。)	ソース プログラムで、アセンブラでまだ実装されていないテクノロジを使用しようとすると、このメッセージが表示されます。メッセージを正確に報告してください。
10 エラー	No input files were specified -- nothing to do (入力ファイルが指定されていません -- 何の処理も行われません。)	すべてのコマンドライン オプションの解析後、検索して開くことができる入力ファイルがアセンブラにありません。
21 エラー	Error reading file: %s (ファイルの読み取りエラー: %s。)	正常に開いたファイルの入力内容の読み取りをアセンブラが突然続行できなくなったことを示します。
30 エラー	Unknown command line switch "%s" (不明なコマンドライン スイッチ "%s"。)	アセンブラが認識またはサポートしないコマンドライン オプションが指定されています。
31 エラー	Syntax error in command line switch "%s": %s (コマンドライン スイッチ "%s" の構文エラー: %s。)	認識されたコマンドライン オプションが正しく指定されていません。
32 エラー	An argument is required for %s (%s には引数が必要です。)	引数を必要とするコマンドライン オプションに引数が指定されていません。
33 エラー	A numeric argument is required for %s (%s には数値引数が必要です。)	数値引数を必要とするコマンドライン オプションに引数が指定されていないか、または数値以外の引数が指定されています。
34 エラー	%s is out of range; must be >= %d and <= %d (%s は範囲外です。%d 以上、%d 以下でなければなりません。)	有効な範囲内に含まれない数値引数が指定されています。このエラー メッセージは、その引数の有効な範囲を示します。
35 エラー	Invalid -show/-noshow option list: "%s" (無効な -show/-noshow オプションの一覧: "%s"。)	アセンブラは、-show または -noshow コマンドライン オプションの一部を解析できませんでした。
36 警告	Ignoring unimplemented option "%s" (実装されていないオプション "%s" は無視されます。)	アセンブラに現在実装されていないテクノロジを指定するコマンドライン オプションまたはアセンブラ ディレクティブが使用されました。

37 警告	Interpreting option "%s" as "%s" (オプション "%s" は "%s" として解釈されます。)	複数の方法でアセンブラが解釈可能な方法でコマンドライン オプションが指定されています。 このメッセージは、オプションを解釈するためにアセンブラが選択する方法を示します。 たとえば、-DEBUG は 引数 EBUG を指定した -D オプション (つまり、EBUG を 1 として定義) の意味にも、スイッチ -debug (デバッグ情報の出力の有効化) の意味にも解釈できます。 この場合は、デバッグ (または EBUG の同等物として -DEBUG=1) の使用によって回避できることがあります。
40 致命的	Insufficient memory%s (メモリ %s が不足しています。)	アセンブラは何らかの理由でメモリを割り当てることができませんでした。
41 致命的	Insufficient memory; cannot allocate %u more%s (メモリが不足しています。%u を割り当てることができません。%s を増やしてください。)	アセンブラは何らかの理由でメモリを割り当てることができませんでした。

関連項目

その他のリソース

Renesas SH-4 シリーズ アセンブラ

SH-4 アセンブラ エラー メッセージ 70-153

次の表は、SH-4 アセンブラ エラー メッセージ 70 ～ 153 の一覧です。

メッセージ番号と重要度	メッセージのテキスト	説明
70 エラー	Character %c was not expected (文字 %c は使用できません。)	アセンブラがソース ファイルで予期しない文字を検出しました。
71 エラー	Nonascii character (^%c) in source (ソースに非 ASCII 文字 (^%c) が見つかりました。)	アセンブラがソース ファイルで予期しない文字を検出しました。
72 エラー	Nonascii character (0x%02d) in source (ソースに非 ASCII 文字 (0x%02d) が見つかりました。)	アセンブラがソース ファイルで予期しない文字を検出しました。
73 エラー	What do you mean by "%s"? ("%s" の意味がわかりません。)	アセンブラが理解できない文字の組み合わせを検出しました。
74 エラー	"%c" is not a valid radix specifier (use HDQBXO) ("%c" は有効な基数指定子ではありません (HDQBXO を使用してください)。)	アセンブラが認識できない基数指定子が使用されています。H、D、Q、B、X、O、h、d、q、b、x、または o のいずれかを使用してください。
75 エラー	"%s" is not a valid radix specifier (use HDQBXO) ("%s" は有効な基数指定子ではありません (HDQBXO を使用してください)。)	アセンブラが認識できない基数指定子が使用されています。H、D、Q、B、X、O、h、d、q、b、x、または o のいずれかを使用してください。
76 エラー	Overflow in numeric constant (数値定数のオーバーフロー。)	32 ビットで表すことができない数値定数が記述されています。有効な範囲は、-2,147,483,648 から 4,294,967,295 または H'00000000 から H'FFFFFFFF です。
77 エラー	A backquoted symbol name cannot span multiple lines (単一引用符で囲まれたシンボル名を複数行にわたって使用することはできません。)	通常は許可されていない文字が名前に含まれるシンボルを囲むために使用される単一引用符 (`) が記述されていますが、対応する単一引用符が同じ行に見つかりません。 これらの引用符は対で使用し、その間にシンボル名を挿入する必要があります。
78 エラー	A backquoted symbol name cannot be null (``) ( 単一引用符で囲まれたシンボル名を NULL (``) にすることはできません。)	名前が NULL (ゼロ長) のシンボルを表す 2 つの隣接する単一引用符 (``) が記述されていますが、このようなシンボル名は無効です。 また、単一引用符 (`) が行の最後の文字として記述されている場合も、アセンブラはこのエラー メッセージを発行します。
80 エラー	Parser error: %s (パーサー エラー: %s。)	このメッセージが表示された場合、アセンブラにバグがあります。アセンブラでエラーを修復し、より有用なエラー メッセージが発行されるようにする必要があります。 正確なメッセージを報告してください。また、可能であれば、アセンブラでこのメッセージが発行される原因となったソース ファイルを提供してください。
81 エラー	Failed to parse input (入力を解析できませんでした。)	このメッセージが表示された場合、アセンブラにバグがあります。アセンブラでエラーを修復し、より有用なエラー メッセージが発行されるようにする必要があります。 正確なメッセージを報告してください。また、可能であれば、アセンブラでこのメッセージが発行される原因となったソース ファイルを提供してください。

101 エラー	Syntax error (構文エラー。)	エラーが報告された行に構文上無効な箇所があります。
150 エラー	Cannot put a %s instruction in a delay slot (遅延スロットに %s 命令を配置することはできません。)	遅延スロットで使用できない命令が遅延スロットに配置されています。
151 エラー	Cannot compute PC displacement in a delay slot (遅延スロットの PC ディスプレイスメントを計算することはできません。)	アセンブラでは、遅延スロット内の PC 相対アドレッシング モードの PC ディスプレイスメントを計算することはできません。このようなディスプレイスメントは分岐対象の PC に相対していなければなりません。
152 エラー	Cannot use "%s #x" in section (%s) with align < %d (配置が %d より小さいセクション (%s) で "%s #x" を使用することはできません。)	配置が 4 より小さいセクションで MOV.L #imm,Rn を使用することはできません。また、配置が 2 より小さいセクションで MOV.W #imm,Rn を使用することもできません。
153 エラー	Instruction at odd address (奇数アドレスの命令。)	奇数アドレスの命令は許可されていません。実行しようとすると、実行時にプロセッサの例外が発生します。

関連項目

その他のリソース

Renesas SH-4 シリーズ アセンブラ

SH-4 アセンブラ エラー メッセージ 200 ～ 312

次の表は、SH-4 アセンブラ エラー メッセージ 200 ～ 312 の一覧です。

メッセージ番号と重要度	メッセージのテキスト	説明
200 エラー	Symbol "%s", first used at %s, is never defined (%s で最初に使用されたシンボル "%s" は未定義です。)	定義されていないシンボルが参照されました。このシンボルの定義 (EQU、ラベルなどを使用) またはインポート (.IMPORT または .GLOBAL を使用) を行ってください。
202 エラー	Illegal symbol "%s" (シンボル "%s" は無効です。)	ユーザー定義のシンボルは \& または . (ピリオド) で開始できません。
203 エラー	Illegal label symbol "%s" (ラベル シンボル "%s" は無効です。)	ユーザー定義のシンボルは \& または . (ピリオド) で開始できません。
205 警告	%s has already been declared %s (%s では既に %s が宣言されています。)	既に宣言または定義されているシンボルが宣言または定義されました。
206 エラー	%s has already been declared %s (%s では既に %s が宣言されています。)	既に宣言または定義されているシンボルが宣言または定義されました。
207 エラー	Preprocessor symbol "%s" has not been defined yet (プリプロセッサ シンボル "%s" はまだ定義されていません。)	プリプロセッサ シンボルが定義前に使用されました。
210 エラー	A string ("%s") is not valid here (文字列 ("%s") はここでは有効ではありません。)	文字定数などの文字列をサポートしていないコンテキストで文字列定数か文字列値のシンボル、または式が使用されました。
211 エラー	Cannot %s a %s symbol (%s) (%s シンボルに %s を実行できません (%s)。)	インポートまたはエクスポートできないシンボル (.ASSIGN、.ASSIGNA、.ASSIGNC、.REG、または .SECTION を使用して定義されたシンボル、または内部レジスタ シンボル) が .EXPORT または .GLOBAL アセンブラ ディレクティブで使用されました。
212 エラー	Symbol "%s" has not been defined yet (シンボル "%s" はまだ定義されていません。)	前方参照が許可されていないコンテキストでシンボルが参照されました。このシンボルはまだ定義されていないか、またはシンボルの値が不明です。
213 エラー	The value of symbol "%s" is not known yet (シンボル "%s" の値はまだ不明です。)	前方参照が許可されていないコンテキストでシンボルが参照されました。このシンボルはまだ定義されていないか、またはシンボルの値が不明です。
214 エラー	"%s" is an .IMPORT symbol; its value is unknown ("%s" は .IMPORT シンボルです。この値が不明です。)	外部参照が許可されていないコンテキストで .IMPORT シンボルが使用されました。
301 エラー	Too many operands; expecting %s (オペランドが多すぎます。%s が必要です。)	命令またはアセンブラ ディレクティブに対して指定されているオペランドが多すぎます。
302 エラー	"%s" is not an opcode, directive, or macro ("%s" が命令コード、ディレクティブ、またはマクロではありません。)	命令コード、ディレクティブ、または定義済みマクロでない操作名が指定されています。スペル チェックを行ってください。操作名がマクロの場合は、最初の使用前に定義済みであることを確認します。

304 エラー	Not enough operands; expecting %s (オペランドが不足しています。%s が必要です。)	命令またはアセンブラ ディレクティブに必要な数のオペランドが指定されていません。
307 エラー	This addressing mode is not legal here (このアドレッシング モードはここでは無効です。)	命令または命令/修飾子の組み合わせに対して無効なアドレッシング モードが指定されているか、または命令に対して無効な修飾子が指定されています。
308 エラー	Invalid addressing mode (incompatible operands) (無効なアドレッシング モードです (互換性のないオペランド)。)	命令または命令/修飾子の組み合わせに対して無効なアドレッシング モードの組み合わせが指定されているか、またはアドレッシング モードと命令の組み合わせに対して無効な修飾子が指定されています。
309 エラー	Invalid addressing mode (illegal register %s): "%s" (無効なアドレッシング モード (無効なレジスタ %s) : "%s"。)	そのレジスタ (またはレジスタの種類) が無効な場所でレジスタ名が指定されています。
310 エラー	Cannot use %s here; only r0 is valid (ここでは %s を使用できません。r0 のみ有効です。)	r0 のみ有効なコンテキストで r0 以外のレジスタが指定されています。
311 エラー	"%s" (%s) is not a register ("%s" (%s) はレジスタではありません。)	.REG ディレクティブのオペランドとして指定されているシンボルは有効なレジスタ名ではありません。
312 エラー	This would require %s relocation (%s の再配置が必要です。)	現在定義されている MS-COFF オブジェクト形式では表すことができない操作を実行しようとしました。

関連項目

その他のリソース

Renesas SH-4 シリーズ アセンブラ

SH-4 アセンブラ エラー メッセージ 400 ～ 453

次の表は、SH-4 アセンブラ エラー メッセージ 400 ～ 453 の一覧です。

メッセージ番号と重要度	メッセージのテキスト	説明
400 エラー	Character constant longer than 4 characters: "%s" (4 文字より長い文字定数: "%s"。)	文字定数は 4 文字より長くすることはできません。 このメッセージは、数値が想定されるコンテキストで文字列 (または文字列値シンボルか式) が入力された場合にも発行されることがあります。
402 エラー	%d (0x%x) is out of range; must be >= %d and <= %u (%d (0x%x) は範囲外です。%d 以上 %u 以下でなければなりません。)	オペランドの値が正しい範囲に含まれていません。 メッセージで有効な範囲が示されます。 このメッセージは、PC 関連の操作 (分岐や MOV @(disp, PC),Rn など) を使用して、あまりに距離が離れた場所を参照しようとしたときや、前方参照しかサポートしていない命令で後方参照である場所を参照しようとしたときにも発行される場合があります。
403 エラー	Cannot %s when one or more operands are relocated (1 つ以上のオペランドが再配置される場合、%s は実行できません。)	相対値に対しては、特定の算術演算しかサポートされません。
404 エラー	Relocated value not valid here; must be absolute (再配置された値はここでは無効です。絶対値でなければなりません。)	絶対値のみ許可されるコンテキストで、相対値または相対値シンボルか式が使用されました。
405 エラー	Cannot %s import symbols or between sections (インポート シンボルに対して、またはセクション間では %s は実行できません。)	相対値および外部シンボルに対しては、特定の算術演算しかサポートされません。
406 エラー	Cannot %s expr w/%d reloc%s and expr w/%d reloc%s (%s expr w/%d reloc%s および expr w/%d reloc%s は実行できません。)	相対値および外部シンボルに対しては、特定の算術演算しかサポートされません。
408 エラー	Cannot divide by zero (0 では除算できません。)	0 による除算は数学的に不可能です。
413 エラー	Unknown relational operator: "%s" (不明な関係演算子: "%s"。)	通常の C 演算子の他に使用できる関係演算子は、コンテキストにかかわらず、EQ、NE、GT、LT、GE、および LE のみです。
420 エラー	Syntax error in operand (オペランドの構文エラー。)	アセンブラが命令のオペランドを認識できません。
421 エラー	Syntax error in expression (式の構文エラー。)	アセンブラが式を認識できません。
422 エラー	Cannot use a %s symbol here (ここでは %s シンボルは使用できません。)	式でレジスタを使用しようとしました。 これは限られた特定の状況以外、許可されません。
423 エラー	Expression must be relative (or use #immediate) (式は相対的でなければなりません (または #immediate を使用)。)	MOV symbol,Rn または MOVA symbol,Rn (symbol は .EQU シンボルなどの絶対シンボル) のような命令が記述されました。 シンボルの値をレジスタに移動するには、MOV #symbol,Rn を使用してください。

424 エラー	The "%s" operator (%s) is not supported ("%s" 演算子 (%s) はサポートされていません。)	認識はされますが、サポートされていない演算子 (++ や -- など) が使用されました。
425 エラー	Expression result overflow: %s %d (0x%x) (式結果のオーバーフロー: %s %d (0x%x)。)	算術演算の結果がオーバーフローしました (重要なビットの喪失)。
426 エラー	Expression result overflow: %d (0x%x) %s %d (0x%x) (式結果のオーバーフロー: %d (0x%x) %s %d (0x%x)。)	算術演算の結果がオーバーフローしました (重要なビットの喪失)。
427 エラー	Expression result overflow: %s %u (0x%x) (式結果のオーバーフロー: %s %u (0x%x)。)	算術演算の結果がオーバーフローしました (重要なビットの喪失)。
453 エラー	%s needs a literal pool before here (%s は、ここより前にリテラル プールを必要とします。)	メッセージのテキストに示されているソース行 (上記の %s) に、エラー メッセージ行の最初に示されているソース行のアドレスまで到達できないリテラル プールの参照が含まれています。 2 つのソース行の間のどこかに .POOL ディレクティブまたは無条件分岐を挿入してください。

関連項目

その他のリソース

Renesas SH-4 シリーズ アセンブラ

SH-4 アセンブラ エラー メッセージ 500 ～ 555

次の表は、SH-4 アセンブラ エラー メッセージ 500 ～ 555 の一覧です。

メッセージ番号と重要度	メッセージのテキスト	説明
500 エラー	Label required here (ここにはラベルが必要です。)	ラベルが必要なディレクティブにラベルが指定されていません。このため、ディレクティブが無視される可能性があります。
501 エラー	Illegal value for .ORG%s (.ORG%s の値が無効です。)	.ORG ディレクティブを使用して、ロケーション カウンタが現在値より低いアドレスに戻されるか 1 MB 以上進められています。 ロケーション カウンタを 1 MB 以上進める場合、MS-COFF オブジェクト ファイル形式ではオブジェクト ファイル内の領域を占有することになります。 別の方法が適しています。
503 エラー	Exported symbol "%s" not defined (エクスポートされたシンボル "%s" は定義されていません。)	メッセージで指定されたシンボルは、.EXPORT ディレクティブで宣言されていますが、ソース ファイルのどこにも定義されていません。
516 エラー	This %s directive conflicts with a previous %s (この %s ディレクティブは、前の %s と競合します。)	.PRINT ディレクティブの 1 つのオペランドが前の .PRINT ディレクティブの 1 つのオペランドと競合しているか、.OUTPUT ディレクティブの 1 つのオペランドが、前の .OUTPUT ディレクティブの 1 つのオペランドと競合しています。
517 エラー	The result of this expression is not known yet (この式の結果はまだ不明です。)	アセンブラが初めてファイルをパススルーするときに値が既知であることが要求されるコンテキストで式が指定されましたが、式の値が未定義です。
519 エラー	String is %u characters long; maximum is 255 (文字列が %u 文字長です。最大文字数は 255 です。)	.SDATAC または .SDATACB ディレクティブに指定される文字列は、長さが 1 バイトで示されるため 255 文字以下にする必要があります。
520 エラー	A string or string expression is required %s (%s には文字列または文字列式が必要です。)	文字列が必要なコンテキストで、数値定数、数値シンボル、または数値式が指定されています。
521 警告	A literal pool would never be put here anyway (ここにはリテラル プールを挿入できません。)	アセンブラがリテラル プールを生成しない場所に .NOPOOL ディレクティブが挿入されています。
522 警告	.POOL illegal in delay slot of %s (%s の遅延スロットの .POOL が無効です。)	リテラル プールが遅延分岐命令の後にワインドされています。アセンブラは自動的に NOP 命令を指定して遅延スロットを埋めます。
531 エラー	Invalid section type "%s" (セクションの種類 "%s" が無効です。)	指定されたセクションの種類が無効です。有効なセクションの種類は、CODE、DATA、DUMMY、および BSS です。大文字と小文字は区別されません。
532 エラー	Section type (%s) must follow section name (%s) (セクションの種類 (%s) はセクション名 (%s) の後にする必要があります。)	セクションの種類をセクション名の前に指定することはできません。
533 エラー	Section type (%s) cannot have a value (セクションの種類 (%s) には値を指定できません。)	このセクションの種類には値を指定できません。
534 エラー	Invalid option "%s" ("%s" は無効なオプションです。)	ディレクティブに無効なキーワード オプションが指定されています。

535 エラー	Option "%s" cannot have a value (オプション "%s" には値を指定できません。)	値を指定できないオプションに値が指定されています。
536 エラー	Option "%s" must have a numeric value (オプション "%s" には数値を指定する必要があります。)	数値を必要とするオプションに値が指定されていないか、数値以外の値が指定されています。
537 エラー	Alignment must be a power of 2 between 1 and 8192. (配置は 1 から 8192 までの 2 の累乗にする必要があります。)	.ALIGN ディレクティブを参照してください。
538 エラー	"%s": MS-COFF doesn't support absolute sections ("%s": MS-COFF は絶対セクションをサポートしていません。)	このメッセージは、MS-COFF オブジェクト形式の制限事項によりこのアセンブラではサポートされていない、Renesas アセンブラが提供する特定のテクノロジを使用しようとした場合にのみ出力されます。
539 エラー	"%s": MS-COFF Cannot export non-relocatable .EQU's ("%s": MS-COFF では、再配置できない .EQU をエクスポートできません。)	相対値だけをエクスポートできます。絶対シンボルはエクスポートできません。
540 エラー	"%s": MS-COFF Cannot export syms from DUMMY section ("%s": MS-COFF では、DUMMY セクションからシンボルをエクスポートできません。)	種類が DUMMY のセクションで定義されたシンボルはエクスポートできません。
541 エラー	Illegal alignment %u; must be a power of two (配置 %u は無効です。2 の累乗にしてください。)	配置の値は 2 の累乗 (2^0 から 2^6) にする必要があります。
542 エラー	Block count too large (max %u) -- directive ignored (ブロック数が多すぎます (最大値 %u) — ディレクティブが無視されます。)	データが 4 GB を超えることになるブロック数が指定されています。
543 エラー	Cannot export "%s"; its value depends on "%s" ("%s" をエクスポートできません。この値は "%s" に依存します。)	外部 (.IMPORT) シンボルに値が依存する .EQU シンボルがエクスポートされています。
544 警告	Code not in function; need a .ENTRY (関数にコードがありません。.ENTRY が必要です。)	コードはすべて .ENTRY ディレクティブと .ENDF ディレクティブの間に挿入する必要があります。
545 警告	Missing .PROLOG between .ENTRY/.ENDF for "%s" ("%s" の .ENTRY/.ENDF の間に .PROLOG がありません。)	各関数 (.ENTRY と .ENDF のペア) 内の任意の場所に .PROLOG ディレクティブを指定する必要があります。
546 警告	Missing .ENDF; closing function "%s" (関数 "%s" を終了する .ENDF がありません。)	コードはすべて .ENTRY ディレクティブと .ENDF ディレクティブの間に挿入する必要があります。
547 エラー	No matching .ENTRY for .ENDF (.ENDF に対応する .ENTRY がありません。)	アセンブラによって、対応する .ENTRY ディレクティブがない .ENDF ディレクティブが検出されました。
548 エラー	No active .ENTRY for .PROLOG (.PROLOG に対してアクティブな .ENTRY がありません。)	アセンブラによって、アクティブな .ENTRY ディレクティブがない (.ENDF ディレクティブによって終了していない) .PROLOG ディレクティブが検出されました。
549 エラー	.PROLOG already specified for function "%s" (関数 "%s" には既に .PROLOG が指定されています。)	1 つの関数 (.ENTRY と .ENDF のペア) 内で .PROLOG ディレクティブが複数回指定されています。
550 警告	.END encountered; ignoring files starting with %s (.END が検出されました。%s で始まるファイルは無視されます。)	コマンド ラインで最終のファイルとして指定されていないファイル内で .END ディレクティブが検出されました。 コマンド ラインで指定された残りのファイルは無視されます。

551 エラー	.SECTION ASSOC= must be followed by a section name (.SECTION ASSOC= の後ろにはセクション名が必要です。)	.SECTION ディレクティブの ASSOC オプションにはセクション名の値が必要です (".text{_main}" などの COMDAT スタイル セクション名を含む)。
552 エラー	Section "%s" must be COMDAT to allow option "%s" (オプション "%s" を使用するには、セクション "%s" を COMDAT にする必要があります。)	現在の .SECTION ディレクティブで宣言されたセクション、または 現在の .SECTION ディレクティブの ASSOC= オプションのターゲットが COMDAT スタイルのセクションではありません。
553 エラー	.PDATA is illegal between .ENTRY and .ENDF (.ENTRY と .ENDF の間の .PDATA は無効です。)	.ENTRY ディレクティブの後および .ENDF ディレクティブの前に .PDATA ディレクティブを指定することは有効ではなくなりました。.PDATA ディレクティブは、適用先の .ENTRY の前に指定する必要があります。
554 エラー	Prolog is too long (%u, max %u); use .BIGENTRY (プロローグが長すぎます (%u、最大値 %u)。.BIGENTRY を使用してください。)	プロローグが 510 バイト長を超える関数には、.ENTRY ディレクティブの代わりに .BIGENTRY ディレクティブを使用する必要があります。
555 エラー	Function is too long (%u, max %u); use .BIGENTRY (関数が長すぎます (%u、最大値 %u)。.BIGENTRY を使用してください。)	関数が 16,777,214 バイト長を超える場合は、.ENTRY ディレクティブの代わりに .BIGENTRY ディレクティブを使用する必要があります。

関連項目

その他のリソース

Renesas SH-4 シリーズ アセンブラ

SH-4 アセンブラ エラー メッセージ 611 ～ 673

次の表は、SH-4 アセンブラ エラー メッセージ 611 ～ 673 の一覧です。

メッセージ番号と重要度	メッセージのテキスト	説明
611 エラー	A macro name is required for .MACRO (.MACRO にマクロ名が必要です。)	.MACRO ディレクティブを使用してマクロ名を指定する必要があります。 具体的には、マクロ名は .MACRO ディレクティブの右側 (ラベルの位置ではなく最初のオペランドの位置) に指定します。
612 エラー	Syntax error in macro name for .MACRO (.MACRO のマクロ名の構文エラー。)	アセンブラは、MACRO ディレクティブの直後にシンボル名ではないと思われるものを検出しました。
619 エラー	Invalid macro parameter name "%s" (無効なマクロ パラメータ名 "%s"。)	マクロ パラメータ名の先頭に数値が使用されています。これは許可されません。
622 エラー	No matching paren for macro expansion exclusion (マクロ展開排他のかっこが一致しません。)	アセンブラは、マクロの展開で検出した \(に対応する) を検出できませんでした。 マクロの展開の排他構文 \(...) は、2 行にわたって使用できません。マクロの 1 行内に収める必要があります。
624 エラー	Too many macro parameters (max %d) (マクロのパラメータが多すぎます (最大値 %d)。)	使用しているマクロにパラメータ (オペランド) が多すぎます。
631 エラー	%s without matching %s (%s に、対応する %s がありません。)	指定されている .AENDI、.AENDR または .AENDW に、対応する .AIF、.AREPEAT または .AWHILE がないようです。 このエラーは、これらの構文が別の構文の中にネストされ、内部の構文の終了ディレクティブが省略されている場合も表示されます。
632 エラー	Cannot have %s after .AELSE (.AELSE の後には %s を指定できません。)	.AELSIF または .AELSE は、同じ .AIF ブロックの .AELSE の後に配置できません。
633 エラー	%s still active at end of "%s" macro expansion ("%s" マクロ展開の最後で %s がアクティブなままになっています。)	.AIF、.AREPEAT、.AWHILE、または .MACRO (マクロ定義) が、マクロの最後に到達してもまだ進行中でした。 これらの構文は、マクロの境界をまたぐことはできません。
634 エラー	%s from line %d still active at end of file %s (ファイル %s の最後で行 %d の %s がアクティブなままになっています。)	.AIF、.AREPEAT、.AWHILE または .MACRO (マクロ定義) が、ファイルの最後に到達してもまだ進行中でした。 これらの構文は、ファイルの境界をまたぐことはできません。
635 エラー	.EXITM outside of any .AREPEAT, .AWHILE, or macro (.EXITM が、.AREPEAT、.AWHILE、またはマクロの外にあります。)	アセンブラが、抜け出すループまたはマクロがなく、孤立していると思われる .EXITM ディレクティブを検出しました。 このメッセージは、前のエラーによってアセンブラがこのような構文の最初を無視した場合にも発行されることがあります。
636 エラー	.END with %s still active (%s の .END がアクティブのままです。)	.AIF、.AREPEAT、.AWHILE またはマクロの展開の進行中に、アセンブラで .END ディレクティブが検出されました。この場合、.END ディレクティブは無視されます。
670 エラー	This syntax is only legal in a macro: "%s" (この構文はマクロ内でのみ使用できます : "%s"。)	たとえば、マクロ本文でマクロが展開されている間にのみ使用できる特別な機能が使用されました。

671 エラー	Syntax error in macro arguments (マクロ引数の構文エラー。)	アセンブラで、マクロに対して 1 つ以上の引数を解析できませんでした。 アセンブラは可能な限り続行しようとします。
672 エラー	End of file reached while processing %s (%s の処理中にファイルの最後に到達しました。)	.AIF、.AREPEAT、.AWHILE または .MACRO (マクロ定義) が、ファイルの最後に到達してもまだ進行中でした。 これらの構文は、ファイルの境界をまたぐことはできません。 このエラー メッセージよりもエラー メッセージ 634 の方が出力される場合が多いです。
673 エラー	Unknown macro parameter name "%s" (不明なマクロ パラメータ名 "%s"。)	現在展開されているマクロに定義されていないマクロ パラメータ名が参照されました。 ネストされたマクロの展開の場合、最も内側のアクティブなマクロの引数のみがアクセス可能です。

関連項目

その他のリソース

Renesas SH-4 シリーズ アセンブラ

SH-4 アセンブラ エラー メッセージ 801 ～ 888

次の表は、SH-4 アセンブラ エラー メッセージ 801 ～ 888 の一覧です。

メッセージ番号と重要度	メッセージのテキスト	説明
801 警告	Symbol %s already defined as a %s (シンボル %s は既に %s として定義されています。)	既に宣言または定義されているシンボルが再度宣言または定義されました。
802 エラー	Symbol %s already defined as a %s (シンボル %s は既に %s として定義されています。)	既に宣言または定義されているシンボルが再度宣言または定義されました。
807 警告	Invalid size specification "%c" (サイズの指定 "%c" が無効です。)	サイズの指定 (修飾子) が無効です。有効な修飾子は B、W、L、b、w、l (L の小文字) に限られます。
808 警告	A size specification ("%c") is not allowed here (サイズの指定 ("%c") はここでは使用できません。)	サイズを指定できない命令またはディレクティブにサイズ指定 (修飾子) が行われました。
810 警告	Too many operands; expecting %s (オペランドが多すぎます。%s が必要です。)	命令またはディレクティブに、使用可能な数を超えたオペランドが指定されました。
811 警告	Label not allowed here; ignoring "%s" (ここではラベルを使用できません。"%s" は無視されます。)	ラベルを指定できないディレクティブにラベルが指定されました。このラベルは無視され、定義されません。さらにエラー メッセージが出力される場合があります。
813 警告	Section "%s" has already been declared %s (セクション "%s" では既に %s が宣言されています。)	.SECTION ディレクティブを使用して一度宣言されたセクションの種類または属性を変更することはできません。
816 警告	Unaligned %s size data at 0x%x (0x%x にある %s サイズ データが整列されていません。)	奇数アドレスへのワードの配置または、4 の倍数ではないアドレスへのロングワードの配置を実行しようとする .DATA、.RES、または関連ディレクティブが書き込まれました。 注意してください。1 つの命令でこのようなデータにアクセスしようとすると、プロセッサの例外が発生します。
817 警告	Beware of possibly unaligned code (整列されていないコードがないか確認してください。)	コード セクションで 2 未満の配置が指定されました。 これによって、奇数アドレスにコードが配置される可能性があります。 奇数アドレスのコードを実行しようとすると、プロセッサの例外が発生します。
818 警告	Cannot guarantee align > section "%s"'s align (%d) (セクション "%s" の配置 (%d) より大きい配置は保証できません。)	.ALIGN ディレクティブに現在のセクションの配置より大きい値が指定されました。このような配置は保証できません。
826 警告	Instructions valid only in Code section; "%s" is %s (命令はコード セクションでのみ有効です。"%s" は %s です。)	実行可能命令または拡張命令が CODE 以外のセクションに書き込まれました。これは許可されません。
827 警告	%s invalid in a %s section (%s) -- ignored (%s は %s セクションで無効です (%s) -- 無視されます。)	初期化されていないデータが、CODE セクションで予約されたか、DUMMY または BSS セクションで初期化されました。これは許可されません。

838 エラー	A character string may not span multiple lines (文字列は複数の行にまたがることができません。)	一般にこのメッセージは閉じる引用符がないか、二重化されていない文字列に埋め込み引用符が存在することを示します。
839 警告	You may not specify a value for %s; it is ignored (%s には値を指定できません。値は無視されます。)	値を指定できないキーワード オプションに値が指定されました。
840 警告	Block count is zero -- directive ignored (ブロック カウントがゼロです -- ディレクティブが無視されます。)	.RESB、.DATAB、または SDATAB ディレクティブがゼロのブロック カウントで書き込まれました。これによりディレクティブが事実上無効化されます。
841 警告	%u is too large to fit in a character; using %u (%u が大きすぎて一文字に収まりません。%u を使用します。)	制御文字式の結果が 255 より大きい値になりました。
842 警告	"%s" is abs; expr is location, not displacement ("%s" は abs です。expr はロケーションで、ディスプレイスメントではありません。)	分岐が絶対セクションのシンボルを参照する場合、シンボルはディスプレイスメントではなくロケーションを示すと見なされます。絶対セクションはサポートされていません。
851 警告	Overriding %s from .PDATA %s (.PDATA %s から %s が上書きされます。)	.PDATA ディレクティブが前の .PDATA ディレクティブを上書きしました。 これは、有効な .ENDF が間にない同一セクション内に .PDATA ディレクティブが 2 つ書き込まれた場合にのみ発生する可能性があります。
852 警告	%s from .PDATA at %s(%d) was never used (%s(%d) の .PDATA の %s は使用されませんでした。)	同一セクション内に有効な .ENDF が後続していない .PDATA ディレクティブが書き込まれました。
870 警告	Unaligned displacement value %d; truncating to %d (整列されていないディスプレイスメント値 %d。%d に切り捨てられます。)	ワードを参照するディスプレイスメント値が奇数であったか、ロングワードを参照するディスプレイスメント値が 4 の倍数以外になっていました。 孤立した最下位ビットがドロップされます。
871 警告	PC-relative in delay slot is branch-dest relative (遅延スロットの PC 相対は分岐先相対です。)	PC 相対命令を遅延スロットに書き込む場合、この命令が相対する PC は分岐先の PC です。
876 警告	Inserted BRA and NOP before literal pool (リテラル プールの前に BRA と NOP が挿入されています。)	リテラル プールが無条件分岐の遅延スロット命令に後続していないことを検出したため、アセンブラはリテラル プールをスキップするために BRA を自動的に挿入し、BRA の遅延スロットを埋めるために NOP を挿入しました。
880 警告	Function end after delayed branch (関数が遅延分岐後に終了しています。)	アセンブラが、分岐の遅延スロットがまだ埋められていない状態の .ENDF ディレクティブ (または入力ファイルかセクションの終了) を検出しました。 アセンブラは NOP を挿入して遅延スロットを埋めます。
886 警告	.END missing -- pretending there was one (.END がありません -- 存在するとして処理しています。)	入力の最後に達するまでアセンブラは .END ディレクティブを検出しませんでした。
887 警告	.END reached -- ignoring to EOF (.END に達しました -- EOF まで無視します。)	アセンブラは .END ディレクティブを検出しましたが、その後に空白行以外のものがあることも検出しました。 アセンブラは、これらをすべて無視しました。
888 警告	Entry point not supported (エントリ ポイントはサポートされません。)	.END ディレクティブにはオペランドを指定できません。

関連項目

その他のリソース

Renesas SH-4 シリーズ アセンブラ

SH-4 アセンブラ エラー メッセージ 903 ～ 998

次の表は、SH-4 アセンブラ エラー メッセージ 903 ～ 998 の一覧です。

メッセージ番号と重要度	メッセージのテキスト	説明
903 エラー	%s: %s	アセンブル一覧ファイルを開くとき、またはこのファイルに書き込むときに問題が発生しました。
904 エラー	%s: %s	オブジェクト ファイルを開くとき、またはこのファイルに書き込むときに問題が発生しました。
907 エラー	Cannot allocate %u bytes of memory for section "%s" (セクション "%s" に %u バイトのメモリを割り当てることができません。)	おそらく .ORG ディレクティブの結果として、アセンブラでは指定されたセクションにデータを格納できるだけのメモリ割り当てができません。 可能であれば、アセンブリの続行が試行されます。
998 エラー	%s	このメッセージは、.AERROR または .ERROR ディレクティブがアセンブルされたときに出力されます。

関連項目

その他のリソース

Renesas SH-4 シリーズ アセンブラ

MIPS ファミリ プロセッサ

MIPS Technologies, Inc. は、高性能で低電力の 32 ビットおよび 64 ビット RISC (縮小命令セット コンピューティング) マイクロプロセッサ アーキテクチャおよびコアを組み込みシステム用に設計しています。MIPS のテクノロジは、半導体企業やシステム OEM にライセンス供与されています。

MIPS RISC アーキテクチャの CPU は、50 MHz、32 ビット、R3000 ベースのデバイスから 266 MHz、64 ビット、R5000 ベースの CPU まで多岐にわたっています。

このセクションのトピック

MIPS マイクロプロセッサの組み込み関数

MIPS 固有の組み込み関数に関する参照情報を提供します。

MIPS コンパイラ オプション

MIPS 固有のコンパイラ オプションに関する参照情報を提供します。

MIPS コーリング シーケンスの仕様

レジスタ、スタック フレームのレイアウト、およびアセンブラについて説明します。

関連セクション

デスクトップ コンパイラとデバイス コンパイラの違い

ビルド オプション、組み込み関数、および例外処理におけるデスクトップ コンパイラとデバイス コンパイラの重要な違いについて説明します。

MIPS マイクロプロセッサの組み込み関数

MIPS デバイス コンパイラが認識する関数は、サポート組み込み関数と呼ばれます。これらの組み込み関数は、C ランタイム (CRT) ルーチンへの呼び出し、または 1 つ以上の一連の命令にコンパイラで変換されます。

コンパイラで常に一連の命令に変換される組み込み関数が展開される関数です。たとえば、コンパイラはすべての MIPS ターゲットの共通組み込み関数として **sqrt** を認識します。**sqrt** は浮動小数点 (FP) 引数を受け取り、FP 値を返す FP 組み込み関数です。

FP ユニットを搭載した MIPS マイクロプロセッサでは、**sqrt** 命令がサポートされます。つまり、FP ユニットを搭載した MIPS マイクロプロセッサの場合、組み込み関数 **sqrt** が、マイクロプロセッサに対してネイティブの **sqrt** 命令にコンパイラで変換されます。

FP ユニットを搭載していない MIPS マイクロプロセッサの場合、**sqrt** はコンパイラで CRT ルーチンへの呼び出しに変換されます。このように、**sqrt** 組み込み関数はすべての MIPS ターゲットに適用されますが、MIPS II FP および MIPS IV FP の場合にのみ命令に展開されます。**sqrt** の詳細については、「ランタイム ライブラリ リファレンス」を参照してください。

次の表は、指定された特定の MIPS ISA でコンパイラが認識するサポート組み込み関数を示しています。

関数	MIPS 16 ASE	MIPS II ISA	MIPS II ISA FP ユニット	MIPS IV ISA FP ユニット
__emul	○	○	○	○
__emulu	○	○	○	○
__ll_lshift	○	○	○	○
__ll_rshift	○	○	○	○
__ull_rshift	○	○	○	○
_enable	○	○	○	○
_disable	○	○	○	○
__regsize	○	○	○	○
_InterlockedDecrement	○	○	○	○
_InterlockedExchangeAdd	○	○	○	○
_InterlockedCompareExchange	○	○	○	○
_InterlockedIncrement	○	○	○	○
_InterlockedExchange	○	○	○	○

関連項目

その他のリソース

ランタイム ライブラリ リファレンス

__emul

この関数では、レジスタ **RT** の 32 ビット値とレジスタ **RS** の 32 ビット値が乗算され、64 ビットの結果が取得されます。

```
__int64 __cdecl __emul(
  int1,
  int2
);
```

パラメータ

int1

[in] **RT の値、乗算の第** 1 **項。**

int2

[in] **RS の値、乗算の第** 2 **項。**

戻り値

2 **進演算の結果。**

備考

この関数は、コンパイラで **mult** 命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
__emul	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目

参照情報

MIPS マイクロプロセッサの組み込み関数

__emulu

この関数は、レジスタ RT の符号なし 32 **ビット値とレジスタ RS の符号なし** 32 **ビット値を乗算し、**64 **ビットの結果を取得します。**

```
__int64 __cdecl __emulu(
  int1,
  int2
);
```

パラメータ

int1

[in] **RT の値、乗算の第** 1 **項。**

int2

[in] **RS の値、乗算の第** 2 **項。**

戻り値

2 **進演算の結果。**

備考

この関数は、コンパイラで multu 命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
__emulu	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目

参照情報

MIPS マイクロプロセッサの組み込み関数

__emul

__ll_lshift

この関数は、64 ビット ワードを、指定したビット数だけ左にシフトします。

```
__int64 __cdecl __ll_lshift(
  int64,
  int
);
```

パラメータ

int64

[in] シフトする値。

int

[in] シフトするビット数。

戻り値

なし。

備考

この関数は、コンパイラで **SLL** 命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
__ll_lshift	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目

参照情報

MIPS マイクロプロセッサの組み込み関数

__ll_rshift

この関数は、64 ビット ワードを、指定したビット数だけ右にシフトします。

```
__int64 __cdecl __ll_rshift(
  int64,
  int
);
```

パラメータ

int64

[in] シフトする値。

int

[in] シフトするビット数。

戻り値

なし。

備考

この関数は、コンパイラで **SRL** 命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
__ll_rshift	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目

参照情報

MIPS マイクロプロセッサの組み込み関数

__regsize

この関数はレジスタ サイズを返します。

```
int __cdecl __regsize(void);
```

パラメータ
なし。

戻り値

ターゲットのレジスタのサイズ。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
__regsize	<stdlib.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目
参照情報
MIPS マイクロプロセッサの組み込み関数

__ull_rshift

この関数は 64 ビットの符号なしワードを、指定したビット数だけ右にシフトします。

```
unsigned __int64 __cdecl __ull_rshift(
  unsigned__int64,
  int
);
```

パラメータ

unsigned_int64

[in] シフトする値。

int

[in] シフトするビット数。

戻り値

なし。

備考

この関数は、コンパイラで **SRA** 命令に変換されます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
__ull_rshift	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目

参照情報

MIPS マイクロプロセッサの組み込み関数

_enable

この関数は MIPS16 ASE **を有効にします。**

```
void _enable(void);
```

パラメータ
なし。

戻り値

なし。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_enable	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目
参照情報
MIPS マイクロプロセッサの組み込み関数

_disable

この関数は、MIPS16 ASE を無効にします。

```
void _disable(void);
```

パラメータ
なし。

戻り値

なし。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_disable	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目
参照情報
MIPS マイクロプロセッサの組み込み関数

_InterlockedCompareExchange

この関数では変数値が比較値とアトミックに比較され、値が同一の場合は *Destination* の値が *Exchange* の値と交換されます。

この関数によって、複数のスレッドが同じ変数を同時に使用できないようになります。

```
InterlockedCompareExchange(
  PVOID* Destination,
  PVOID ExChange,
  PVOID Comperand
);
```

パラメータ

Destination

[out] **関数が** *Comperand* **と比較する値へのポインタ。**

ExChange

[in] *Destination* **の内容の置換に使用される値** (**必要な場合**)。

Comperand

[in] *Destination* **の値との比較に使用される標準値。**

戻り値

Destination **がポイントする変数の初期値。**

備考

インターロックされた関数では、複数のスレッドに共有される変数への同時アクセスの単純な機構が提供されます。変数が共有メモリにある場合、異なるプロセスのスレッドでこの機構を使用することができます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_InterlockedCompareExchange	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目

参照情報

MIPS マイクロプロセッサの組み込み関数

_InterlockedDecrement

この関数は加数変数に対して減分値のアトミック加算を実行します。この関数によって、複数のスレッドが同じ変数を同時に使用できないようになります。

```
InterlockedDecrement(
  PLONG Addend
);
```

パラメータ

Addend

[in] **減分する値。**

戻り値

2 **進演算の結果。**

備考

インターロックされた関数では、複数のスレッドに共有される変数への同時アクセスの単純な機構が提供されます。変数が共有メモリにある場合、異なるプロセスのスレッドでこの機構を使用することができます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_InterlockedDecrement	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目

参照情報

MIPS マイクロプロセッサの組み込み関数

_InterlockedExchangeAdd

この関数は、加数変数に対して増分値のアトミック加算を実行します。この関数によって、複数のスレッドが同じ変数を同時に使用できないようになります。

```
InterlockedExchangeAdd(
  PLONG Addend,
  LONG Increment
);
```

パラメータ

Addend

[in, out] **増分する値。**

Increment

[in] *Addend* **の増分に使用する値。**

戻り値

Addend **がポイントする変数の初期値。**

備考

インターロックされた関数では、複数のスレッドに共有される変数への同時アクセスの単純な機構が提供されます。変数が共有メモリにある場合、異なるプロセスのスレッドでこの機構を使用することができます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_InterlockedExchangeAdd	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目

参照情報

MIPS **マイクロプロセッサの組み込み関数**

_InterlockedIncrement

この関数は、加数変数に対して増分値のアトミック加算を実行します。この関数によって、複数のスレッドが同じ変数を同時に使用できないようになります。

```
InterlockedIncrement(
  PLONG Addend
);
```

パラメータ

Addend

[in] **増分する値。**

戻り値

なし。

備考

インターロックされた関数では、複数のスレッドに共有される変数への同時アクセスの単純な機構が提供されます。

変数が共有メモリにある場合、異なるプロセスのスレッドでこの機構を使用することができます。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_interlockedIncrement	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目

参照情報

MIPS マイクロプロセッサの組み込み関数

_InterlockedExchange

この関数は、32 ビット値のペアをアトミックに交換します。この関数によって、複数のスレッドが同じ変数を同時に使用できないようになります。

```
LONG InterlockedExchange(
  LONG volatile* Target,
  LONG Value
);
```

パラメータ

Target

[in, out] 交換する値へのポインタ。関数はこの変数を Value に設定し、その前の値を返します。

Value

[in] Target がポイントする値と交換する値。

戻り値

この関数はターゲットの初期値を返します。

備考

インターロックされた関数では、複数のスレッドに共有される変数への同時アクセスの単純な機構が提供されます。変数が共有メモリにある場合、異なるプロセスのスレッドでこの機構を使用することができます。

Target パラメータによってポイントされる変数を 32 ビットの境界に配置する必要があります。そうでない場合、この関数はマルチプロセッサ x86 システムおよびすべての非 x86 システムで失敗します。

この関数は、PAGE_NOCACHE 修飾子を使用して割り当てられたメモリでは使用できません。

必要条件

ルーチン	必須のヘッダー	アーキテクチャ
_InterlockedExchange	<winnt.h>	MIPS16、MIPSII、MIPSIII、MIPS IV、MIPS 32

関連項目

参照情報

MIPS マイクロプロセッサの組み込み関数

MIPS コンパイラ オプション

次の表に、MIPS マイクロプロセッサのコンパイラ スイッチを示します。

オプション	説明
/QMmipsNN – 特定の MIPS ISA 用のコードの生成	MIPS I、II、III、IV、V、32 および 64 ISA (命令セット アーキテクチャ) のコードを生成します。
/QMmips16 – MIPS16 ASE 用コードの生成	MIPS16 ASE 用のコードを生成します。
/QMFPE – 浮動小数点のエミュレーション	浮動小数点ハードウェアを使用して、浮動小数点エミュレーションを有効にします。
/QMRnnnn – 特定の MIPS チップ用に最適化	Phillips PR3900、NEC VR4100、NEC VR4200、および NEC VR4300 のチップ固有のインライン アセンブラを有効にして、対応する MIPS ISA 用のコードを生成します。

関連項目

その他のリソース

MIPS コーリング シーケンスの仕様

/QMmips16 – MIPS16 ASE 用コードの生成

このオプションでは、MIPSII ISA の MIPS16 ASE (特定アプリケーション向け拡張機能) 用コードが生成されます。MIPS16 は、特にメモリが貴重な組み込みシステムで役立ちます。

関連項目

その他のリソース

MIPS コーリング シーケンスの仕様

/QMFPE – 浮動小数点のエミュレーション

/QMFPE オプションは、浮動小数点演算をサポートする際に、コンパイラの期待値を制御します。

次の表は、このオプションの詳細を示しています。

オプション	説明	既定のステータス
/QMFPE	浮動小数点のエミュレーションをオンにします。	**/QMmipsNN** オプションが指定されていないか、または **/QMmips1** オプションか **/QMmips2** オプションが指定されている場合は既定で設定されます。
/QMFPE -	浮動小数点ハードウェア装置の命令がコンパイラで生成されます。	**/QMmips3**、**/QMmips4**、**/QMmips5**、または **/QMmips64** オプションが指定されている場合は、既定で設定されます。

関連項目

参照情報

/QMmipsNN – 特定の MIPS ISA 用のコードの生成

その他のリソース

MIPS コーリング シーケンスの仕様

/QMmipsNN – 特定の MIPS ISA 用のコードの生成

次のオプションは、コンパイラが生成するコードの MIPS ISA を指定します。

スイッチ	説明	既定のステータス
/QMmips1	古い MIPS I 命令セットのコードを生成します。	既定値は、/QMFPE です。
/QMmips2	MIPS II 命令セットのコードを生成します。	既定値は、/QMFPE です。
/QMmips3	MIPS III ISA のコードを生成します。	既定値は、/QMFPE- です。
/QMmips4	MIPS IV ISA のコードを生成します。	既定値は、/QMFPE- です。
/QMmips5	MIPS V ISA のコードを生成します。	既定値は、/QMFPE- です。
/QMmips32	MIPS 32 ISA のコードを生成します。	既定値なし。
/QMmips64	MIPS 64 ISA のコードを生成します。	既定値なし。

関連項目

参照情報

/QMFPE – 浮動小数点のエミュレーション

/QMRnnnn – 特定の MIPS チップ用に最適化

このオプションによってコンパイラは特定の MIPS ISA に対して最適化されたコードを生成し、マイクロプロセッサ固有のインライン アセンブリ命令を許可します。次に、コンパイラはそのプロセッサに固有の命令または命令シーケンスを選択し、他のプロセッサと互換性のない実行可能ファイルを生成します。

次の表に、/QMRX000 スイッチの実装を示します。

オプション	同意義のもの	プロセッサ
/QMR3900	-QMmips2 -D_M_MRX000=3900	R3900
/QMR4100	-QMmips2 -D_M_MRX000=4100	R4100
/QMR4200	-QMmips2 -QMFPE- -D_M_MRX000=4200	R4200
/QMR4300	-QMmips2 -QMFPE- -D_M_MRX000=4300	R4300
/QMR5400	-QMmips4 -QMFPE- -D_m_MRX000=5400	R5400

これらのスイッチと /QMmips16 – MIPS16 ASE 用コードの生成スイッチは相互に排他的です。

関連項目

その他のリソース

MIPS コーリング シーケンスの仕様

MIPS コーリング シーケンスの仕様

MIPS コーリング シーケンスの仕様は、MIPS マイクロプロセッサのコンパイラとアセンブリ言語プログラムの開発の指針を示します。

また、この標準では、ツール、デバッガ、および呼び出しスタックの自動スキャンを実行するオペレーティング システム ユーティリティを開発することもできます。

このセクションのトピック

MIPS レジスタ

MIPS I、II、III、および IV アーキテクチャのレジスタ割り当てを指定します。

MIPS スタック フレームのレイアウト

MIPS スタック フレームのレイアウトについて説明します。

MIPS プロローグおよびエピローグ

コード シーケンスに関する参照情報と、MIPS マイクロプロセッサでの構造化例外処理の実装に必要なマクロを提供します。

関連セクション

デスクトップ コンパイラとデバイス コンパイラの違い

組み込み関数、ビルド オプション、および配置に関する問題における相違点について説明します。

RISC 環境の SEH

RISC マイクロプロセッサ環境で構造化例外処理が行われるしくみについて説明します。

MIPS レジスタ

MIPS マイクロプロセッサには 32 個の汎用レジスタが備えられています。

- MIPS I 命令セット アーキテクチャ (ISA) および MIPS II ISA のレジスタは 32 ビットです。
- MIPS III 以上の ISA では、32 ビット モード実行時には 32 ビット レジスタが使用され、64 ビット モード実行時には 64 ビット レジスタが使用されます。
- レジスタ $1、$26、$27、$29 は、アセンブラ、コンパイラ、およびオペレーティング システムによって特別な目的のために予約されています。
- レジスタ $0 は値 0 にハードワイヤードされています。$31 はジャンプおよびリンク命令用のリンク レジスタですが、他の命令で慎重に使用することもできます。

次の表は、これらのレジスタの使用規則の概要です。

レジスタ名	共通名	説明
$0	ゼロ	常に 0 の値を持ちます。このレジスタへの書き込みはすべて無視されます。
$1	at	アセンブラー時レジスタ。
$2-$3	v0-v1	関数結果のレジスタ。 関数は、32 ビット レジスタの使用時、整数の結果を v0 で返し、64 ビット整数の結果を v0 および v1 で返します。 浮動小数点ハードウェアが存在しない場合や、コンパイラのオプションで浮動小数点のエミュレーションが有効になっている場合、関数は 32 ビット レジスタの使用時、単精度浮動小数点の結果を v0 で返し、倍精度浮動小数点の結果を v0 および v1 で返します。 v0 および v1 は一時レジスタとして使用できます。 複数の関数呼び出しにわたって保持されることはありません。
$4-$7	a0-a3	整数型引数の最初の 4 ワードを格納する関数引数レジスタ。 関数は、これらのレジスタを使用して浮動小数点引数を格納します。 浮動小数点ハードウェアが存在しない場合や、コンパイラのオプションで浮動小数点のエミュレーションが有効になっている場合、関数は a0 を使用して最初の単精度浮動小数点引数を格納し、a1 を使用して 2 番目の単精度浮動小数点引数を格納します。 関数は、a0-a1 を使用して最初の倍精度浮動小数点引数を格納し、a2-a3 を使用して 2 番目の倍精度浮動小数点引数を格納します。 複数の関数呼び出しにわたって保持されることはありません。
$8-$15, $24-$25	t0-t9	必要に応じて使用できる一時レジスタ。複数の関数呼び出しにわたって保持されることはありません。
$16-$23, $30	s0-s8	自由に使用するために保存されているレジスタ。 複数の関数呼び出しにわたって保持されます。これらのレジスタは、呼び出された関数によって使用される前に保存しておく必要があります。
$26-$27	k0-k1	オペレーティング システムのカーネルによる使用と例外のリターンのために予約されています。

$28	gp	グローバル ポインタ。 Windows CE では使用されません。呼び出された関数の保存レジスタとして使用される場合があります。
$29	sp	スタック ポインタ。
$31	ra	呼び出し元関数によって保存されるリターン アドレスのレジスタ。保存後の使用が可能です。
$f0	該当なし	関数呼び出しから浮動小数点値および倍精度値を返すときに使用される、関数のリターン レジスタ。
($f12、$f13) および ($f14、$f15)	該当なし	浮動小数点値と倍精度値のパラメータを関数に渡すときに使用されるレジスタの 2 つのペア。 レジスタのペアは倍精度値を渡すため、かっこで囲む必要があります。 浮動小数点値の受け渡しには、$f12 と $f14 だけが使用されます。

次の一覧は、浮動小数点レジスタに関する追加情報です。

- MIPS ISA I、MIPS ISA II、および MIPS III で 32 ビット モードで ISA を実行している場合は、$f4、$f6、$f8、$f10、$f16、$f18 の一時レジスタのみを使用できます。
  倍精度の命令でこれらのレジスタを操作すると、上位 32 ビットが暗黙の奇数レジスタに書き込まれます。この奇数レジスタには直接アクセスできません。
- 永続レジスタ $f20、$f22、$f24、$f26、$f28、および $f30 では、複数の関数呼び出しにわたって値が保持されます。
- MIPS アーキテクチャ III を 64 ビット モードで実行している場合は、次のレジスタも一時レジスタとして使用できます。$f1、$f3、$f5、$f7、$f9、$f11、$f17、$f19、$f21、$f23、$f25、$f27、$f29、$f31。

関連項目

参照情報

MIPS スタック フレームのレイアウト

その他のリソース

MIPS コーリング シーケンスの仕様

MIPS スタック フレームのレイアウト

スタック フレームは次の 4 つの領域で構成されます。

- パラメータまたは引数領域
- ローカル変数領域
- レジスタ保存領域
- 引数作成領域

呼び出し元の関数はすべての引数に対してスタック上のスペースを割り当てます (ただし、そのうち一部の引数はレジスタで渡される場合があります)。このため、呼び出し元の関数では、その関数からの呼び出しで必要な引数の最大値に対応できるだけのスペースをスタック上に予約する必要があります。

呼び出し元の関数では、渡すパラメータの数が 3 つ以下であっても、少なくとも 4 ワード分のスペースを割り当てる必要があります。

関数では、その関数が引数をレジスタで渡すかどうかにかかわらず、すべての引数に対してスペースを割り当てる必要があります。これによって、引数レジスタを保存する必要がある場合に、そのための保存領域が呼び出し先の関数に提供されます。

呼び出し元の関数は最初の引数用に引数レジスタを割り当てます。次に、2 番目の引数に残りの引数レジスタを割り当てます。このようにして、すべての引数レジスタを使用してしまうか、引数一覧の引数がなくなるまで同様に繰り返します。引数の残りの部分と、残りの引数は、すべてスタックに渡されます。

引数レジスタの割り当てでは、メモリの場合と同様に、同一の配置が維持されます。複数の引数一覧のマッピング時に、引数レジスタによっては、メモリ内の埋め込み領域と同様に、関連するものが何も含まれない場合があります。

関連項目

その他のリソース

MIPS コーリング シーケンスの仕様

MIPS プロローグおよびエピローグ

MIPS マイクロプロセッサに SEH を実装するには、プロローグおよびエピローグのコード セグメントが必要です。

MIPS プロローグには、ルーチンにスタック フレームを設定するコードが含まれます。MIPS エピローグには、ルーチンのフレームを削除して呼び出し元関数に戻るコードが含まれます。これらのタスクを実行するプロローグおよびエピローグ コードは MIPS コンパイラで生成されますが、アセンブリ コード関数を記述する場合は、ユーザーがプロローグおよびエピローグ コードを記述する必要があります。

このセクションのトピック

MIPS アセンブラ マクロ

MIPS プロローグ

MIPS エピローグ

関連セクション

RISC 環境の SEH

MIPS スタック フレームのレイアウト

MIPS プロローグ

MIPS プロローグは、直接つながった複数の部分で構成されており、これは MIPS 16 ビット モード、MIPS 32 ビット モード、および MIPS 64 ビット モードのいずれが有効であるかにかかわりません。

次のステップでは、MIPS プロローグの必須要素を示します。

次のコード例は、MIPSII に適用されます。MIPS IV の場合は、**sw** 命令を **sd** に、**lw** 命令を **ld** に置き換えます。

1. プロローグを定義し、エントリを設定します。

```
.ent <routine_name>
<routine_name>:
```

2. スタック フレームの領域を予約します。

```
Addiu sp, -<frame size>
```

フレーム サイズが 1024 を超える場合は、4 バイトの拡張 **addiu** 命令が生成されます。
フレーム サイズが 32768 を超える場合は、次の命令シーケンスを使用して、スタック ポインタをリテラル プールの定数で更新します。

```
hw  $3, <frame size> constant offset(pc)
move  $2, $29
subu  $2, $2, $3
move  $29, $2
```

スタック フレームのサイズは、8 の倍数である必要があります。これには、ローカル変数と一時レジスタ、保存済みレジスタ、およびリーフ以外のルーチンのプロシージャ コール引数領域のためのスペースが含まれます。
レジスタ $16 ～ $23 または $30 を使用するルーチン、または呼び出された関数で保存する浮動小数点レジスタを使用するルーチンでは、これらのレジスタを保存する必要があります。
プロシージャ コールの引数領域には、リーフ以外のルーチンで呼び出されるプロシージャの引数に必要な最大バイト数が含まれます。この必要なバイト数には、N32 呼び出し規則が使用されていない限り、関数がレジスタで渡す引数が含まれます。

3. 仮想フレーム ポインタを設定します。仮想フレーム ポインタは、フレーム サイズに追加される **sp**($29) です。

```
.frame framereg (usually $29), framesize, returnreg (usually $31)
```

4. 保存済みの汎用レジスタそれぞれのビットマスクにビットを設定します。ビットはリトル エンディアンの順序で設定します。フレーム オフセットは、レジスタ保存領域が始まる仮想フレーム ポインタからの負のオフセット値です。

```
.mask mask, <frame offset>
```

5. 保存する必要があるレジスタを保存します。
たとえば、**ra** は保存する必要があります。

```
sw ra, <frame size> + <frame offset>(sp)
```

次に続く、より小さい番号のレジスタを保存する必要がある場合、そのオフセットは MIPS モードに依存します。次のレジスタが MIPS16 レジスタの場合は、次のとおりです。

```
sw  $<mips16 register>, <frame size> + <frame offset> - n(sp)
```

そうでない場合は、次のとおりです。

```
move  $2, <mipsii register>
sw  $2, <frame size> + <frame offset> - n(sp)
```

N は 4 であり、次に続くより小さい番号のレジスタを保存するたびに 4 ずつ増加します。
<frame size> + <frame offset> が 32767 より大きい場合は、レジスタ保存ループを実行する前に、次の 3 つの命令のシーケンスが

必要です。

```
lw  $2, <frame size> + <frame offset>constant offset(pc)
move  $2, $29
addu  $3, $2, $3
```

sw 命令は、MIPS レジスタのモードに依存します。次のレジスタが MIPS16 レジスタの場合は、次のとおりです。

```
sw  $<mips16 register>, -n($3)
```

そうでない場合は、次のとおりです。

```
move  $2,$<mipsii register>
sw  $2, -n($3)
```

6. プロローグの終了を指定します。

```
.prologue 0
```

関連項目

参照情報

MIPS アセンブラ マクロ

その他のリソース

MIPS プロローグおよびエピローグ

MIPS エピローグ

各プロシージャに含まれるプロローグは 1 つだけですが、プロシージャで複数の終了ポイントを使用する場合は、そのプロシージャに任意の数のエピローグを含むことができます。各エピローグを特定の部分で構成する必要があります。すべての部分を連続して記述し、間に命令を挿入しません。

次の手順は、MIPS ISA に対して呼び出された関数によって保存されたレジスタの復元方法を示しています。MIPS IV では、**lw** 命令を **ld** に置き換えます。

MIPS16 エピローグでは、呼び出された関数によって保存されたレジスタの復元に若干異なるガイドラインが適用されます。

1. プロローグに保存されているレジスタごとに復元の命令を発行します。

   ```
   lw $31, framesize+frameoffset($29)      ; restore return addresslw reg, framesize+fram
   eoffset-N($29)    ; restore integer register
   ldc1 reg, framesize+frameoffset-N($29)  ; restore float register
   ```

   N は 4 であり、保存されている後続のより小さい番号のレジスタごとに 4 ずつ増加します。
2. プロシージャから戻ります。

   ```
   j $31
   ```

3. ルーチンを終了します。

   ```
   .end routine_name
   ```

   リーフ ルーチンの適切な最小のエピローグに、リターン ジャンプと **.end** を含みます。さらに、非リーフ ルーチンの場合は、レジスタ $31 を読み込む必要があります。

   ```
   jr scratch
   ```

関連項目

参照情報

MIPS アセンブラ マクロ

その他のリソース

MIPS プロローグおよびエピローグ

MIPS アセンブラ マクロ

アセンブラ レベルのマクロは、アセンブラ ディレクティブの詳細からプログラマを分離します。

次の表は、MIPS マイクロプロセッサに定義されたマクロを示しています。

マクロ	説明
ALTERNATE_ENTRY (MIPS)	ルーチンの代替エントリを宣言します。
EXCEPTION_HANDLER (MIPS)	指定された例外ハンドラを次に続く **NESTED_ENTRY** に関連付けます。
LEAF_ENTRY (MIPS)	プロローグ コードを必要としないルーチンの最初を宣言します。
NESTED_ENTRY (MIPS)	既存のスタック フレームがあるか、または新規のスタック フレームを作成するルーチンの最初を宣言します。
PROLOGUE_END	プロローグ領域の最後を指定します。

関連項目

その他のリソース

MIPS コーリング シーケンスの仕様

ALTERNATE_ENTRY (MIPS)

このマクロによって、種類が NESTED_ENTRY (MIPS) または LEAF_ENTRY (MIPS) のルーチンの代替エントリが宣言されます。

```
ALTERNATE_ENTRY Name[,
[Section=]SectionName]
```

パラメータ

Name

Name はエントリ ポイントです。これにはグローバル名前空間に含まれる名前を使用します。

SectionName

SectionName はそのエントリが含まれるセクションの名前です。「備考」を参照してください。

戻り値

なし。

備考

ALTERNATE_ENTRY マクロでは、*SectionName* パラメータは使用されません。このパラメータは受け入れられますが、**NESTED_ENTRY (MIPS)** および **LEAF_ENTRY (MIPS)** との整合性を保つために無視されます。このパラメータを使用する場合は、**ALTERNATE_ENTRY** 呼び出しをルーチンの本体に含める必要があります。

関連項目

参照情報

NESTED_ENTRY (MIPS)
LEAF_ENTRY (MIPS)

EXCEPTION_HANDLER (MIPS)

このマクロでは、例外ハンドラ *Handler* が、次に続く NESTED_ENTRY (MIPS) **または** LEAF_ENTRY (MIPS) **と関連付けられます。**

```
EXCEPTION_HANDLER Handler
```

パラメータ
Handler

例外ハンドラの名前。

戻り値

なし。

関連項目
参照情報
NESTED_ENTRY (MIPS)
LEAF_ENTRY (MIPS)

LEAF_ENTRY (MIPS)

このマクロでは、プロローグ コードを必要としないルーチンの最初が宣言されます。

```
LEAF_ENTRY Name[,[Section=]SectionName]
```

パラメータ

Name

Name **は、ルーチン名です。これにはグローバル名前空間に含まれる名前を使用します。**

SectionName

SectionName **は、このエントリが含まれるセクションの名前です。これは省略可能で、既定値は** .text **です。**

戻り値

なし。

備考

LEAF_ENTRY **には、関連付けられた** PROLOGUE_END **が必要です。**

関連項目

参照情報

NESTED_ENTRY (MIPS)
EXCEPTION_HANDLER (MIPS)
PROLOGUE_END

NESTED_ENTRY (MIPS)

このマクロでは、既存のフレームを持つルーチン、またはスタック フレームを作成するルーチンの最初が宣言されます。

```
NESTED_ENTRY Name[,[Section=]SectionName]
```

パラメータ

Name

Name **は、ルーチン名です。これにはグローバル名前空間に含まれる名前を使用します。**

SectionName

SectionName **は、このエントリが含まれるセクションの名前です。これは省略可能で、既定値は** .text **です。**

戻り値

なし。

備考

NESTED_ENTRY **には、**PROLOGUE_END **を関連付けておく必要があります。**

関連項目

参照情報

LEAF_ENTRY (MIPS)
EXCEPTION_HANDLER (MIPS)
PROLOGUE_END

PROLOGUE_END

このマクロは、MIPS マイクロプロセッサ ファミリのプロローグ領域の最後を指定します。

```
PROLOGUE_END
```

パラメータ
なし。

戻り値

なし。

備考

このマクロは、NESTED_ENTRY (MIPS) マクロまたは LEAF_ENTRY (MIPS) マクロに続けて使用する必要があります。

関連項目
参照情報
NESTED_ENTRY (MIPS)
LEAF_ENTRY (MIPS)

MIPS アセンブラ

MIPS のライセンス付きマイクロプロセッサでは、スタンドアロンのアセンブラ、インライン アセンブリ、およびアセンブラ マクロがサポートされます。

このセクションのトピック

スタンドアロン MIPS アセンブラ

スタンドアロンの MIPS アセンブラを簡単に説明します。

MIPS インライン アセンブリ言語

MIPS インライン アセンブリの規則を説明します。

関連セクション

MIPS ファミリ プロセッサ

主要機能、コンパイラのオプション、および組み込み関数の概要を示します。

スタンドアロン MIPS アセンブラ

MIPS エディションの Visual C++ には、スタンドアロン アセンブラが含まれています。

このアセンブラには、インライン アセンブリにあるような制限がまったくありません。このため、特にアセンブリ コードのみで記述された関数を作成する場合は、スタンドアロン アセンブラが唯一の手段になります。

ただし、スタンドアロン アセンブラにはあまり便利ではない面もあります。たとえば、C および C++ のソース コードのオブジェクトを参照するのは容易ではありません。

MIPS スタンドアロン アセンブラ ドライバは Mipsasm.exe です。Mipsasm はアセンブリ言語ソース ファイルをファイル拡張子 .asm または .s で認識します。

次のコード例では、コマンド ラインから 1 回のステップで prog.asm を実行可能ファイルにアセンブルしてリンクします。

```
mipsasm -DMIPS -QMmips2 prog.asm /link /entry:mainACRTStartup
    /subsystem:windowsce /nodefaultlib corelibc.lib coredll.lib
```

次のコード例では、"-c" アセンブラ スイッチを使用して、リンクせずに、アセンブリ言語ソース ファイルをオブジェクト ファイルにアセンブルする方法を示します。

```
mipsasm -DMIPS -QMmips2 -c prog.asm
```

その他のアセンブラ オプションを表示するには、コマンド ラインに次のように入力します。

```
mipsasm /?
```

関連項目

概念

MIPS インライン アセンブリ言語

その他のリソース

MIPS コーリング シーケンスの仕様

MIPS インライン アセンブリ言語

インライン アセンブリ言語を使用すると、アセンブリやリンクのステップを別途に実行しなくても、アセンブリ言語命令を C および C++ のソース プログラムに埋め込むことができます。

インライン アセンブラがコンパイラに組み込まれるため、別途にアセンブリする必要がありません。

インライン アセンブラには追加のステップが必要ないため、特にプロセッサ リソースにアクセスする必要がある場合には、別のアセンブラを使用するよりも便利なことがあります。

インライン アセンブリ コードでは、インライン アセンブラ関数のコンパイラ最適化が無効になります。プログラムで最適化が重要な場合は、組み込み関数の使用を検討してください。

インライン アセンブリ コードを使用するには、特定の ISA (命令セット アーキテクチャ) に適切なコンパイラ スイッチを選択する必要があります。

これらのスイッチの詳細については、「/QMmipsNN – 特定の MIPS ISA 用のコードの生成」を参照してください。

制限事項

__asm ステートメントで指定する文字列には、スタンドアロン アセンブラで受け入れられる有効なアセンブリ コードを使用できます。プロシージャまたは関数を別途に定義するために使用しない限り、サポートされている擬似命令を使用することができます。

完全なアセンブリ言語プロシージャを作成するには、「スタンドアロン MIPS アセンブラ」で説明されているスタンドアロン アセンブラを使用します。

関連項目

参照情報

MIPS コンパイラ オプション

その他のリソース

MIPS コーリング シーケンスの仕様

MIPS インライン アセンブリの _asm キーワード

_asm キーワードはインライン アセンブラを呼び出します。このキーワードは、C または C++ ステートメントの任意の場所で使用できます。コンパイラは、**_asm** ステートメントを組み込み関数呼び出しとして扱います。**_asm** キーワードを宣言する **#pragma** 組み込みステートメントを含める必要があります。

次の C++ コードは、**#pragma** ディレクティブで _asm を宣言する方法を示しています。

```
extern "C" {
void _asm (char*,...);
};
#pragma intrinsic (__asm)
```

この宣言は、C ソース モジュールで実行する必要はありません。

_asm 宣言では、文字列引数は 1 つ以上のアセンブリ言語ステートメントで構成される NULL 終了文字列です。コンパイラは、最初の文字列引数によって記述されたアセンブリ コードに後続の引数を渡します。

次の例では、4 に 5 を掛けて結果を **t0** レジスタに格納する命令を生成します。この例では、**%0** への参照で値 "4" が生成済みのアセンブリ コードの最初の引数に適用されます。**%1** では、値 "5" が生成済みのアセンブリ コードの 2 番目の引数に適用されます。

```
__asm("mul t0, %0, %1", 4, 5);
```

_asm プロトタイプの文字列引数内では、埋め込まれたセミコロン (;) または改行 (\n) が複数のアセンブリ言語ステートメントを区切るために使用されます。次の例は、セミコロンを使用して複数のアセンブリ言語ステートメントを表す方法を示しています。

```
__asm(".set noat;"
"mult $1, $4, $5; "
".set at ");
```

インライン アセンブリでの関数アドレスの読み込み

MIPS インライン アセンブラでは、疑似命令 **la** を使用して関数のアドレスをレジスタに読み込みます。/Og (グローバルな最適化) を指定してコンパイルすると、コンパイラで次のエラー メッセージが生成されます。

```
Error C2759: inline assember reports function literal used with call optimization
```

/Og (グローバルな最適化) を指定してコードをコンパイルする場合は、コンパイラ エラーが発生しない他の構造に **la** 疑似命令を置き換える必要があります。

次のコードはこの置換例を示しています。

- このコードの構造を

  ```
  __asm("la v0, f");
  ```

  次のコードの構造に置き換えます。

  ```
  __asm("move v0,%0", f);
  ```

関連項目

概念

MIPS インライン アセンブリ言語

MIPS _asm ステートメント レジスタ

次の表に、MIPS マクロ名と、関連付けられているレジスタの説明を示します。

MIPS16 マクロ名	MIPSII マクロ名	関連付けられているレジスタ	説明
zero	zero	$0	常に zero。このレジスタへの書き込みは無視されます。
該当なし	AT	$1	アセンブラー時レジスタ。
v0	v0	$2	戻り値の保持に使用します。
v1	v1	$3	戻り値の保持に使用します。
a0	a0	$4	引数レジスタ。整数引数の最初の 4 ワードを渡すために使用します。
a1	a1	$5	引数レジスタ。整数引数の最初の 4 ワードを渡すために使用します。
a2	a2	$6	引数レジスタ。整数引数の最初の 4 ワードを渡すために使用します。
a3	a3	$7	引数レジスタ。整数引数の最初の 4 ワードを渡すために使用します。
該当なし	t0	$8	一時レジスタ。任意に変更できます。
該当なし	t1	$9	一時レジスタ。任意に変更できます。
該当なし	t2	$10	一時レジスタ。任意に変更できます。
該当なし	t3	$11	一時レジスタ。任意に変更できます。
該当なし	t4	$12	一時レジスタ。任意に変更できます。
該当なし	t5	$13	一時レジスタ。任意に変更できます。
該当なし	t6	$14	一時レジスタ。任意に変更できます。
該当なし	t7	$15	一時レジスタ。任意に変更できます。
該当なし	s0	$16	保存済みレジスタ。関数の呼び出し用に保存しておく必要があります。
該当なし	s1	$17	保存済みレジスタ。関数の呼び出し用に保存しておく必要があります。
該当なし	s2	$18	保存済みレジスタ。関数の呼び出し用に保存しておく必要があります。
該当なし	s3	$19	保存済みレジスタ。関数の呼び出し用に保存しておく必要があります。
該当なし	s4	$20	保存済みレジスタ。関数の呼び出し用に保存しておく必要があります。
該当なし	s5	$21	保存済みレジスタ。関数の呼び出し用に保存しておく必要があります。
該当なし	s6	$22	保存済みレジスタ。関数の呼び出し用に保存しておく必要があります。
該当なし	s7	$23	保存済みレジスタ。関数の呼び出し用に保存しておく必要があります。

t8	t8	$24	追加の一時レジスタ。
該当なし	t9	$25	追加の一時レジスタ。
該当なし	k0	$26	カーネルの予約済みレジスタ。
該当なし	k1	$27	カーネルの予約済みレジスタ。
該当なし	gp	$28	グローバル ポインタ。
sp	sp	$29	スタック ポインタ。
該当なし	s8	$30	追加の保存済みレジスタ。
ra	ra	$31	リターン アドレス レジスタ。

レジスタのマクロ名の代わりに、ドル記号のプレフィックスをつけた数値名を使用できます。たとえば、次の 2 つのステートメントは同じです。

```
__asm("add v0, a0, $16");
__asm("add $2, $4, s0");
```

関連項目
概念
MIPS インライン アセンブリ内の _asm キーワード

MIPS アセンブラ エラー メッセージ

ここでは、MIPS アセンブラのエラー メッセージについて説明します。

このセクションのトピック

MIPS エラー メッセージ 1 ～ 525

次の表に、MIPS エラー メッセージ 1 ～ 525 を示します。

番号	重要度	メッセージのテキスト
1	致命的	INTERNAL ASSEMBLER ERROR (assembler file '%s', line %d) (アセンブラ内部エラー (アセンブラ ファイル '%s'、行 %d)。)
1	エラー	INTERNAL ASSEMBLER ERROR (assembler file '%s', line %d) (アセンブラ内部エラー (アセンブラ ファイル '%s'、行 %d)。)
2	致命的	Assembler is out of heap space (アセンブラのヒープ領域が不足しています。)
3	致命的	NTERNAL ASSEMBLER ERROR: Insufficient memory (assembler file '%s', line %d) (アセンブラ内部エラー。メモリ不足 (アセンブラ ファイル '%s'、行 %d)。)
6	致命的	Invalid error number: %s(%d, ...) (無効なエラー番号: %s(%d, ...)。)
8	致命的	No input file specified (入力ファイルが指定されていません。)
9	致命的	Image file '%Fs' is not a PE image file (イメージ ファイル '%Fs' が PE イメージファイルではありません。)
10	致命的	Invalid Coff object '%Fs': swapped magic = 0x%4x (無効な COFF オブジェクト '%Fs'。magic = 0x%4x が切り替えられました。)
12	致命的	Symbol '%Fs' not found during disassembly of '%Fs' ('%Fs' の逆アセンブリ時に、シンボル '%Fs' が見つかりませんでした。)
13	致命的	Invalid EOF reading file '%Fs' (無効な EOF 読み取りファイル '%Fs'。)
15	警告	Unable to reach line number %d of file '%Fs' (ファイル '%Fs' の行番号 %d に到達できません。)
18	警告	Line number %d is greater than maximum of %d for file '%Fs (行番号 %d が、ファイル '%Fs の最大値 %d を超えています。)
20	エラー	Local label number is out of range (ローカル ラベル番号が範囲外です。)
32	致命的	%s (%d):WIN32 '%s' failed; %s (WIN32 '%s' が失敗しました。%s。)
32	警告	%s (%d):WIN32 '%s' failed; %s (WIN32 '%s' が失敗しました。%s。)
63	致命的	Unknown option '%c' in '%s' ('%s' に不明なオプション '%c' があります。)

83	致命的	Cannot open %Fs file: '%Fs': %Fs (%Fs ファイルを開くことができません: '%Fs': %Fs。)
83	警告	Cannot open %Fs file: '%Fs': %Fs (%Fs ファイルを開くことができません: '%Fs': %Fs。)
83	エラー	Cannot open %Fs file: '%Fs': %Fs (%Fs ファイルを開くことができません: '%Fs': %Fs。)
84	警告	Cannot read %Fs file: '%Fs': %Fs (%Fs ファイルを読み取ることができません: '%Fs': %Fs。)
84	エラー	Cannot read %Fs file: '%Fs': %Fs (%Fs ファイルを読み取ることができません: '%Fs': %Fs。)
85	致命的	Cannot write %Fs file: '%Fs': %Fs (%Fs ファイルを書き込むことができません: '%Fs': %Fs。)
110	エラー	Warnings treated as error - no object file generated (警告がエラーとして処理されました。オブジェクト ファイルは生成されませんでした。)
112	致命的	Cannot unlink %Fs file: '%Fs': %Fs (%Fs ファイルのリンクを解除できません: '%Fs': %Fs。)
120	致命的	Product ID mismatch between '%s' version '%ld' and '%s' version '%ld' ('%s' バージョン '%ld' と '%s' バージョン '%ld' で、プロダクト ID が一致しません。)
213	エラー	Invalid numerical argument '%s' (無効な数値引数 '%s'。)
213	致命的	Invalid numerical argument '%s' (無効な数値引数 '%s'。)
418	エラー	Extended-precision floating point not supported (拡張精度浮動小数点はサポートされません。)
502	致命的	%s (%d):Unexpected opcode '%d' (予期しない命令コード '%d'。)
503	致命的	%s (%d):.ent or .aent not aligned on word boundary (.ent または .aent がワード境界に整列されていません。)
504	致命的	%s (%d):tried to unput over buffer boundary (バッファ境界を超えて unput を実行しようとしました。)
505	致命的	%s (%d):not expecting arg size of %d (必要な引数サイズ %d ではありません。)
506	エラー	Obsolete or corrupt binasm '%s' (binasm '%s' が古いか破損しています。)
507	致命的	Exceeded 64k relocation entries. (64 K の再配置エントリを超えました。)Recompile with /Gy (/Gy を使用して再コンパイルしてください。)
508	エラー	Illegal branch-label '%s' reference at line (%d): (行 (%d) で無効な分岐ラベル '%s' が参照されました。)branch and label are in different sections (分岐とラベルは別々のセクションに含まれます。)

511	エラー	Redefinition of symbol '%s' (シンボル '%s' の再定義。)
512	エラー	.repeat only valid in assembler (.repeat はアセンブラでのみ有効です。)
513	エラー	.endr only valid in assembler (.endr はアセンブラでのみ有効です。)
514	情報	Unsupported option type (%d) in binasm (binasm ではサポートされていないオプションの種類 (%d) です。)
515	エラー	.noalias and .alias require gp registers (.noalias と .alias には gp レジスタが必要です。)
516	エラー	.cpload must be inside noreorder (.cpload は noreorder 内に配置する必要があります。)
517	エラー	.half not on halfword boundary (.half がハーフワード境界にありません。)
518	エラー	.word not on word boundary (.word がワード境界にありません。)
519	エラー	.dword not on double-word boundary (.dword がダブルワード境界にありません。)
520	エラー	.float or .double not on double-word boundary (.float または .double がダブルワード境界にありません。)
521	警告	NOP required for mtc0/mfc0 (mtc0/mfc0 に NOP が必要です。)
522	エラー	Both .cpalias and .cprestore used in the same .ent/.end pair (.cpalias と .cprestore の両方が、同じ .ent/.end のペアで使用されました。)
523	エラー	.cpalias registers do not match between .ent/.end pair (.cpalias レジスタが .ent/.end のペアの間で一致しません。)
524	エラー	.cprestore offsets do not match between .ent/.end pair (.cprestore オフセットが .ent/.end のペアの間で一致しません。)
525	エラー	.cprestore or .cpalias can only be placed in text (.cprestore または .cpalias は、text 内にのみ配置できます。)

関連項目

参照情報

MIPS エラー メッセージ 526 ～ 610
MIPS エラー メッセージ 611 ～ 660
MIPS エラー メッセージ 661 ～ 910
MIPS エラー メッセージ 911 ～ 980

その他のリソース

MIPS ファミリ プロセッサ

MIPS エラー メッセージ 526 ～ 610

次の表に、MIPS エラー メッセージ 526 ～ 610 を示します。

番号	重要度	メッセージのテキスト
526	エラー	Code can only be placed in text (コードは text 内にだけ配置できます。)
527	エラー	Bad instruction opcode (命令の命令コードが正しくありません。)
528	情報	Preparing gp-tables (gp-table を準備しています。)
529	情報	Preparing gp-tables (gp-table を構築しています。)
530	エラー	Label referenced but not defined: %s (ラベルが参照されていますが定義されていません: %s。)
531	エラー	Not all branch-label symbols were defined (branch-label シンボルに定義されていないものがあります。)
532	エラー	C0 opcode must be inside .set noreorder section (C0 命令コードは .set noreorder セクション内になければなりません。)
533	エラー	Byte address of jump target must fit in 28 bits (ジャンプ先のバイト アドレスは 28 ビットに収める必要があります。)
534	エラー	Branch target is not word-aligned (ジャンプ先がワード整列されていません。)
536	エラー	Register must differ from base (レジスタはベース以外にしてください。)
537	エラー	Operand 1 should be fp reg (オペランド 1 は fp reg でなければなりません。)
538	エラー	ulwu instruction not implemented (ulwu 命令は実装されていません。)
539	エラー	coproc doubles not implemented (コプロセッサ倍精度は実装されていません。)
540	エラー	li.e instruction not implemented (li.e 命令は実装されていません。)
542	エラー	Exponent out of range: %s (範囲外の指数です: %s。)
543	エラー	Only \\""1\\"" or \\""0\\"" allowed left of binary point: %s (バイナリ ポイントの左側には \\""1\\"" または \\""0\\"" だけを使用できます: %s。)

544	エラー	Illegally denormalized fraction: %s (非正規化分数が正しくありません: %s。)
545	エラー	Floating point underflow: %s (浮動小数点アンダーフロー: %s。)
546	エラー	-M forbidden when assembler runs on a MIPS machine (アセンブラが MIPS コンピュータで実行される場合 -M は許可されません。)
547	エラー	-M does not support \\"".extended\\"" (-M は \\"".extended\\"" をサポートしません。)
548	エラー	-M does not support hex floating point (-M は 16 進浮動小数点をサポートしません。)
549	エラー	Floating point literal too long (浮動小数点リテラルが長すぎます。)
550	エラー	Too many float literals — compile with \\""-Wb,-nopool\\"" (浮動小数点リテラルが多すぎます。\\""-Wb,-nopool\\"" でコンパイルしてください。)
551	エラー	Cannot label a pseudo-op in text section (text セクションの擬似命令にラベルを指定できません。)
552	エラー	Shift amount not 0..31 (0 から 31 以外のシフト量です。)
553	エラー	Shift amount not 0..63 (0 から 63 以外のシフト量です。)
554	エラー	gp-relative segments together exceed 64k bytes: %d bytes (gp-relative セグメントが合計で 64 KB を超えています: %d バイト。)
558	エラー	MIPS16 move must use a MIPSII gp register (MIPS16 move は MIPSII gp レジスタを使用する必要があります。)
559	エラー	MIPS16 addiu immediate limited to 15 bits (MIPS16 addiu 即値 は 15 ビットに制限されます。)
560	エラー	Doubleword opcode not supported by mips2 or earlier (mips2 およびそれ以前はダブルワード命令コードをサポートしません。)
575	エラー	MIPS16 load with symbolic offset must use different base & dest (シンボリック オフセットと共に読み込まれた MIPS16 は別のベースと宛先を使用しなければなりません。)
576	エラー	MIPS16 store may not use symbolic offset in base-offset mode (MIPS16 ストアはベースオフセット モードのシンボリック オフセットを使用できません。)
577	エラー	Non-MIPS16 instruction used in MIPS16 text section (MIPS16 text セクションで MIPS16 以外の命令が使用されています。)
578	エラー	MIPS16 instruction used in non-MIPS16 text section (MIPS16 以外の text セクションで MIPS16 の命令が使用されています。)

579	エラー	2nd register invalid in MIPS16 dsrl and dsra (MIPS16 dsrl と dsra の第 2 レジスタが無効です。)
580	エラー	Branch offset not specified (分岐オフセットが指定されていません。)
581	エラー	Relative offset not multiple of 4 (相対オフセットが 4 の倍数ではありません。)
582	エラー	Relative offset beyond 32768 (相対オフセットが 32768 を超えています。)
583	エラー	cprestore offset must be positive and divisible by 4 (cprestore オフセットは 4 で割れる正の数にする必要があります。)
584	エラー	.cpalias requires a register argument (.cpalias にはレジスタ引数が必要です。)
585	エラー	Number out of range: %s (範囲外の数値です: %s。)
586	エラー	MIPS16 general purpose register expected (MIPS16 汎用レジスタが必要です。)
587	エラー	2-register MIPS16 jalr must take $ra as first operand (2-レジスタの MIPS16 jalr には、第 1 オペランドとして $ra を指定する必要があります。)
602	エラー	Assembler op/directive expected (アセンブラにオペランドまたはディレクティブが必要です。)
603	エラー	Undefined symbol in expression (式内に未定義のシンボルがあります。)
604	エラー	Undefined symbol in expression: %s (式内のシンボルが無効です: %s。)
605	エラー	Symbol must have absolute value: %s (シンボルには絶対値が必要です: %s。)
607	エラー	No such label (そのようなラベルはありません。)
608	エラー	Too many local labels (ローカル ラベルが多すぎます。)
609	エラー	Overflow (オーバーフロー。)
610	警告	Large decimal set sign bit (10 進セット符号ビットが大きいです。)

関連項目

参照情報

MIPS エラー メッセージ 1 ～ 525
MIPS エラー メッセージ 611 ～ 660
MIPS エラー メッセージ 661 ～ 910

MIPS エラー メッセージ 911 ～ 980

その他のリソース

MIPS ファミリ プロセッサ

MIPS エラー メッセージ 611 ～ 660

次の表に、MIPS エラー メッセージ 611 ～ 660 を示します。

番号	重要度	メッセージのテキスト
611	エラー	Badly delimited numeric literal (数値リテラルの区切りが正しくありません。)
612	エラー	Badly delimited hexadecimal literal (16 進数リテラルの区切りが正しくありません。)
613	エラー	Hex digit in decimal literal (10 進数リテラルに 16 進数があります。)
614	エラー	Hex floating point literal too long (16 進浮動小数点リテラルが長すぎます。)
615	エラー	Missing exponent in floating-point literal (浮動小数点リテラルに指数がありません。)
616	エラー	Truncating token (トークンが切り捨てられます。)
618	エラー	Literal string not terminated (リテラル文字列が終了していません。)
619	エラー	Literal string too long (リテラル文字列が長すぎます。)
620	エラー	Number in string too large (文字列の数値が大きすぎます。)
621	エラー	Missing \\"" at end of string (文字列の最後に \\"" がありません。)
622	エラー	Expected cpp-generated line number (cpp 生成の行番号が必要です。)
623	エラー	Expected cpp-generated file name (cpp 生成のファイル名が必要です。)
624	警告	Truncating cpp-generated filename (cpp 生成のファイル名が切り捨てられます。)
625	エラー	Section not declared for symbol '%s' (シンボル '%s' にセクションが宣言されていません。)
626	エラー	Address symbol expected (アドレス シンボルが必要です。)
627	エラー	Register expected (レジスタが必要です。)
628	エラー	Undefined symbol '%s' (シンボル '%s' が未定義です。)
629	エラー	')' expected: %s (')' が必要です: %s。)
630	エラー	String within expression may have only one character: %s (式内の文字列には 1 文字だけ使用できます: %s。)
631	エラー	Immediate value out of range (範囲外の即値です。)
632	エラー	Conflicting definition of symbol '%s' (シンボル '%s' の定義が競合しています。)
633	エラー	Should be gp register (gp レジスタでなければなりません。)
634	エラー	Should be even gp register (偶数の gp レジスタでなければなりません。)
635	エラー	Should be coprocessor register (コプロセッサ レジスタでなければなりません。)

636	**エラー**	Should be even coprocessor register (**偶数のコプロセッサ レジスタでなければなりません。**)
637	**エラー**	Should be floating point register (**浮動小数点レジスタでなければなりません。**)
638	**エラー**	Should be even floating point register (**偶数の浮動小数点レジスタでなければなりません。**)
639	**エラー**	Should be multiple-of-2 register (2 **の倍数のレジスタでなければなりません。**)
640	**エラー**	Should be multiple-of-4 register (4 **の倍数のレジスタでなければなりません。**)
641	**エラー**	Should be multiple-of-4 floating point register (4 **の倍数の浮動小数点レジスタでなければなりません。**)
642	**エラー**	Should be fp double or gp single register (fp **倍精度または** gp **単精度のレジスタでなければなりません。**)
643	**情報**	No global in text_localize(%d) '%s' (text_localize(%d) '%s' **にグローバル変数がありません。**)
644	**エラー**	Expression required (**式が必要です。**)
645	**エラー**	Break operand out of range (**範囲外の** break **オペランドです。**)
646	**エラー**	Ill-formed symbolic expression (**シンボリック式の形式が正しくありません。**)
647	**エラー**	Bad id in expression (**式の** ID **が正しくありません。**)
648	**エラー**	Anticipating a string for .dword's value (.dword **の値には文字列が必要です。**)
649	**エラー**	Invalid memory tag (**メモリ タグが無効です。**)
650	**致命的**	%s (%d): .text name mismatch, '%s' : '%s' (%s (%d): .text **名が一致しません。**'%s': '%s'**。**)
651	**エラー**	Invalid external expression (**外部式が無効です。**)
652	**エラー**	Cache operation has invalid value (**キャッシュ演算に無効な値が含まれます。**)
653	**エラー**	Floating point constant expected (**浮動小数点定数が必要です。**)
654	**エラー**	2nd register not allowed in non-mips3 architecture (mips3 **以外のアーキテクチャでは第** 2 **レジスタを使用できません。**)
655	**エラー**	lui expression not in to 65535 range (lui **式が** 65535 **までの範囲にありません。**)
656	**エラー**	Invalid syntax in statement (**ステートメントに無効な構文が含まれます。**)
657	**エラー**	Shift value not 0 to 31 (inclusive) (**シフト値が** 0 **から** 31 **までの範囲にありません。**)
658	**エラー**	Shift value not 0 to 63 (inclusive) (**シフト値が** 0 **から** 63 **までの範囲にありません。**)
659	**エラー**	Operation requires immediate operand (**演算には即値オペランドが必要です。**)
660	**エラー**	Immediate operand not 0 to 65535 (**即値オペランドが** 0 **から** 65535 **の範囲にありません。**)

関連項目

参照情報

MIPS エラー メッセージ 1 ～ 525
MIPS エラー メッセージ 526 ～ 610

MIPS エラー メッセージ 661 ～ 910
MIPS エラー メッセージ 911 ～ 980

その他のリソース

MIPS ファミリ プロセッサ

MIPS エラー メッセージ 661 ～ 910

次の表に、MIPS エラー メッセージ 661 ～ 910 を示します。

番号	重要度	メッセージのテキスト
661	エラー	Immediate operand not -32768 to 32767 (即値オペランドが -32768 ～ 32767 の範囲にありません。)
662	エラー	Immediate operand not allowed on fp (即値オペランドは fp で使用できません。)
663	エラー	Statement extends past logical end (ステートメントが論理限界を超えています。)
664	エラー	Should be floating point CC reg (浮動小数点 CC レジスタでなければなりません。)
665	エラー	Invalid symbol for address (アドレスのシンボルが無効です。)
666	エラー	Label expected (ラベルが必要です。)
667	エラー	Symbol is not a label (シンボルがラベルでありません。)
668	エラー	Label or number expected (ラベルまたは番号が必要です。)
669	エラー	Cannot find symbol (シンボルが見つかりません。)
670	エラー	.shift_addr expression not 1 (.shift_addr 式が 1 でありません。)
671	エラー	.align expression not to 12 (.align 式が 12 になっていません。)
672	エラー	String literal expected (文字列リテラルが必要です。)
673	エラー	Byte value must be -128 to 255 (バイト値は -128 ～ 255 でなければなりません。)
674	エラー	Identifier expected (識別子が必要です。)
675	エラー	Number expected (番号が必要です。)
676	エラー	Invalid symbol for .comm/.lcomm/.extern (.comm/.lcomm/.extern のシンボルが無効です。)

677	**エラー**	Undefined assembler operation (**アセンブラの処理が定義されていません。**)
678	**エラー**	.err directive encountered (.err **ディレクティブが検出されました。**)
680	**エラー**	Invalid symbol for .[a]ent: %s (.[a]ent **の無効なシンボル**: %s。)
681	**エラー**	Invalid symbol for .globl: %s (.globl **の無効なシンボル**: %s。)
682	**エラー**	Invalid section name: %s (**無効なセクション名**: %s。)
683	**エラー**	Invalid id in expression (**式の** ID **が無効です。**)
684	**エラー**	Value must be -32768 to 65535 (**値は** -32768 **～** 65535 **でなければなりません。**)
685	**エラー**	.repeat not allowed in struct (**構造で** .repeat **は使用できません。**)
686	**エラー**	.repeat may not nest (.repeat **はネストできません。**)
687	**エラー**	.endr without preceding .repeat (.repeat **の前に** .endr **がありません。**)
688	**エラー**	Reserved name used as label (**予約済みの名前がラベルとして使用されています。**)
689	**エラー**	Colon must follow a local label (**ローカル ラベルの後にコロンが必要です。**)
690	**エラー**	Malformed statement (**ステートメントの形式が正しくありません。**)
691	**エラー**	%hi/%lo/%gprel not followed by ((%hi/%lo/%gprel **の後に** (**がありません。**)
692	**警告**	Invalid expression with %hi/%lo: %s (%hi/%lo **の無効な式**: %s。)
693	**エラー**	Expected \\""(\\"" before base register (**基底レジスタの前に** \\""(\\"" **が必要です。**)
694	**エラー**	Expected \\"")\\"" after base register (**基本レジスタの後に** \\"")\\"" **が必要です。**)
695	**エラー**	Macro expansion needs at register after .set noat (**マクロの展開は** .set noat **の後のレジスタで行う必要があります。**)

696	エラー	Code field in trap instruction not 0..1023 (トラップ命令のコード フィールドが 0 ～ 1023 でありません。)
697	エラー	+= expected after . (. の後に += が必要です。)
698	エラー	.text block missing .ent (.text ブロックに .ent がありません。)
699	エラー	.end without matching .ent (.end に対応する .ent がありません。)
897	警告	Opcode used without -QMmips64 or -QMViper (-QMmips64 または -QMViper を指定しないで命令コードを使用しました。)
898	警告	MIPSII opcode used without -QMmips32 (-QMmips32 を指定しないで MIPSII 命令コードを使用しました。)
899	警告	Opcode used without -QMmips32 or -QMmips4 (-QMmips32 または -QMmips4 を指定しないで命令コードを使用しました。)
900	警告	Opcode used without -QMmips32 or -QMmips64 or -QMViper (-QMmips32、-QMmips64、または -QMViper を指定しないで命令コードを使用しました。)
901	エラー	Shift value not 32 to 63 (inclusive) (シフト値が 32 ～ 63 の範囲にありません。)
905	エラー	Should be vector (ベクトルでなければなりません。)
906	エラー	Should be debug reg (デバッグ レジスタでなければなりません。)
907	警告	MIPS V opcode used without -QMmips5 (-QMmips5 を指定しないで MIPS V 命令コードを使用しました。)
908	警告	Mips viper opcode used without -QMViper (-QMViper を指定しないで Mips viper 命令コードを使用しました。)
909	警告	Mips R5400 opcode used without -QMR5400 (-QMR5400 を指定しないで Mips R5400 命令コードを使用しました。)
910	警告	Using unimplemented unaligned-access opcode (未実装で未整列アクセスの命令コードを使用しています。)

関連項目

参照情報

MIPS エラー メッセージ 1 ～ 525
MIPS エラー メッセージ 526 ～ 610
MIPS エラー メッセージ 611 ～ 660
MIPS エラー メッセージ 911 ～ 980

その他のリソース

MIPS ファミリ プロセッサ

MIPS エラー メッセージ 911 ～ 980

次の表に、MIPS エラー メッセージ 911 ～ 980 を示します。

番号	重要度	メッセージのテキスト
911	警告	Zero length string (文字列の長さが 0 です。)
912	警告	Register alu instruction converted to immediate instruction (レジスタ alu 命令は即値命令に変換されました。)
916	警告	opcode used without /QMRnnnn where nnnn=3900, 32, or 64 (/QMRnnnn を指定しないで命令コードを使用しました (nnnn は 3900、32、または 64)。)
917	警告	Floating point opcode used with -QMFPE (-QMFPE を指定して浮動小数点命令コードを使用しました。)
918	警告	Mark II opcode not available on R4100 used with -QMR4100 (-QMR4100 を指定して、R4100 で使用できない Mark II 命令コードを使用しました。)
919	警告	Mark II opcode not available on all Windows CE architectures (Mark II 命令コードは、すべての Windows CE アーキテクチャで使用できません。)
920	警告	Mark II opcode not available on R3910 used with -QMR3910 (-QMR3910 を指定して、R3910 で使用できない Mark II 命令コードを使用しました。)
922	警告	Mips R4100 opcode used without -QMR4100 (-QMR4100 を指定しないで Mips R4100 命令コードを使用しました。)
923	警告	wait instruction used without /QMmips32, /QMmips64, or /QMViper (/QMmips32、/QMmips64、または /QMViper を指定しないで待機命令を使用しました。)
924	警告	INTERNAL COMPILER WARNING - Invalid version stamp Please choose the Technical Support command on the Visual C++ Help menu, or open the Technical Support help file for more information (内部コンパイラ警告 – 無効なバージョン スタンプ。詳細については、Visual C++ のヘルプ メニューでテクニカル サポートのコマンドを選択するか、テクニカル サポートのヘルプ ファイルを開いてください。)
926	警告	Length of .lcomm was less than 1: %s (.lcomm の長さが 1 未満: %s。)
927	警告	Errata 52: DIV in branch delay slot may not work on a R4000 chip (エラー 52: 分岐遅延スロットの DIV は R4000 チップでは動作しない場合があります。)
928	警告	Division by zero (0 で除算しました。)
929	警告	Cross-assembler ignores -fli (クロスアセンブラは -fli を無視します。)
930	警告	Decimal .extended limited to 55 bit precision: %s (10 進の .extended は 55 ビットの精度に制限されています: %s。)
931	警告	Floating under/over-flow in conversion to binary (バイナリへの変換時の浮動小数点アンダー/オーバーフロー。)
932	警告	Floating exception in conversion to binary (バイナリへの変換時の浮動小数点例外。)
932	警告	Floating exception in conversion to binary (バイナリへの変換時の浮動小数点例外。)
933	警告	Cannot correct .e alignment (.e 配置を修正できません。)
934	警告	Cannot do floating-point load on unaligned data (未整列のデータには浮動小数点を読み込めません。)

935	警告	Improperly aligned register (レジスタの配置が不適切です。)
936	警告	sData offset exceeded $gp padding threshold (sData オフセットが $gp 埋め込みしきい値を超えています。)
937	警告	Extern offset exceeded $gp padding threshold (Extern オフセットが $gp 埋め込みしきい値を超えています。)
938	警告	Bss offset exceeded $gp padding threshold (Bss オフセットが $gp 埋め込みしきい値を超えています。)
939	警告	Branch target is not word-aligned (分岐ターゲットがワード整列されていません。)
940	警告	Mark II opcode used without -QMmips2 (-QMmips2 を指定しないで Mark II 命令コードを使用しました。)
941	警告	Mark III opcode used without -QMmips3 (-QMmips3 を指定しないで Mark III 命令コードを使用しました。)
942	警告	MIPS IV opcode used without -QMmips4 (-QMmips4 を指定しないで MIPS IV 命令コードを使用しました。)
943	警告	Too many file names '%s' (ファイル名 '%s' が多すぎます。)
944	警告	Unknown option '%s' (不明なオプション '%s'。)
945	警告	Macro instruction used (マクロ命令を使用しました。)
946	警告	Macro instruction used in branch delay slot (分岐遅延スロットでマクロ命令を使用しました。)
947	警告	.set nomove is obsolete (.set nomove は古い形式です。)
948	警告	nomacro requires noreorder (nomacro には noreorder が必要です。)
949	警告	reorder requires macro (reorder には macro が必要です。)
950	警告	No code generated for '%Fs' ('%Fs' にコードが生成されていません。)
951	警告	.loc should precede .ent for /Od (/Od では .loc を .ent の前に置いてください。)
952	エラー	Cannot handle misaligned 64 bit const (不適切な配置の 64 ビット const を処理できません。)
953	警告	nop must be inside .set noreorder section (nop は .set noreorder セクション内になければなりません。)
954	警告	Macro instruction used t (マクロ命令で t が使用されました。)
956	警告	Symbol %s is not in comdat section yet it has the comdat attribute (comdat 属性があるにもかかわらず、シンボル %s が comdat セクション内にありません。)
964	警告	Image file '%Fs' has unfamiliar optional header size %d (イメージ ファイル '%Fs' のオプション ヘッダー サイズ %d が異常です。)
972	エラー	duplicate public symbol '%s' defined for COMDAT '%s' (-Gy), linked image may not run (パブリック シンボル '%s' が COMDAT '%s' (-Gy) に対して重複して定義されました。リンク先のイメージが実行されない場合があります。)
972	警告	duplicate public symbol '%s' defined for COMDAT '%s' (-Gy), linked image may not run (パブリック シンボル '%s' が COMDAT '%s' (-Gy) に対して重複して定義されました。リンク先のイメージが実行されない場合があります。)
975	警告	Directive not implemented (ディレクティブが実装されていません。)

976	警告	$31 not allowed in conditional branch and link ($31 は条件分岐およびリンクでは使用できません。)
977	エラー	Coprocessor operation > 25 bits (コプロセッサ演算 > 25 ビット。)
978	警告	Extra filename on command line (コマンドライン上に余分なファイル名があります。)
979	警告	Line too long (行が長すぎます。)
980	警告	Used t without .set noat (.set noat を指定しないで t を使用しました。)
981	警告	Cannot redefine symbol (シンボルは再定義できません。)
982	警告	Load/store of an undefined symbol: %s (未定義のシンボルの読み込み/保存: %s。)
983	警告	JAL should not use same register twice (JAL では同じレジスタを 2 回使用できません。)
984	警告	JAL should not use $31 alone or any register twice (JAL では $31 の単独使用またはレジスタの 2 回使用はできません。)
985	警告	Optional argument not S, ignored (オプションの引数が S でないため、無視されました。)
986	警告	.option name expected (.option の名前が必要です。)
987	警告	.ent missing preceding .end (.end の前に .ent がありません。)
988	警告	.end missing at end of assembly (アセンブリの終わりに .end がありません。)
989	エラー	.ent/.end block never defined the procedure name (.ent/.end ブロックでプロシージャ名が定義されていません。)
990	警告	.aent must be inside .ent/.end block (.aent は .ent/.end ブロック内になければなりません。)
991	警告	sym directive not implemented (sym ディレクティブが実装されていません。)
992	警告	.line directive not implemented (.line ディレクティブが実装されていません。)
993	警告	.dead directive not implemented (.dead ディレクティブが実装されていません。)
994	警告	Unknown name in .option (.option に不明な名前があります。)
995	警告	.set option expected (.set オプションが必要です。)
996	警告	Unknown option in .set (.set に不明なオプションがあります。)
997	警告	Hint field not in range 0..31 (範囲 0 ～ 31 にヒント フィールドがありません。)
998	警告	Multiply accumulate instruction used without /QMRnnnn - Optimize for Specific MIPS chip, where nnnn = 4121, Viper, 5400, 32, or 64. (/QMRnnnn – 特定の MIPS チップ用に最適化を指定しないで乗算累算命令を使用しました (nnnn は 4121、Viper、5400、32 または 64)。)

関連項目

参照情報

MIPS エラー メッセージ 1 ～ 525
MIPS エラー メッセージ 526 ～ 610
MIPS エラー メッセージ 611 ～ 660
MIPS エラー メッセージ 661 ～ 910

その他のリソース

デバイス用のユーザー インターフェイス リファレンス

ほとんどの Visual Studio ユーザー インターフェイス機能は、デスクトップ用にデザインされているかデバイス用にデザインされているかに関係なく同じです。関連情報については、『一般的なユーザー インターフェイス要素 (Visual Studio)』のヘルプ トピックを参照してください。

このセクションのトピックでは、デバイス アプリケーションの開発のためだけに作成されているインターフェイス要素について説明します。

このセクションの内容

すべてのデバイス プロジェクトに共通

[エミュレータ ブートストラップの構成] ダイアログ ボックス

[TCP/IP トランスポートの構成] ダイアログ ボックス

[デバイスへの接続] ダイアログ ボックス

[配置] ダイアログ ボックス (デバイス)

[デバイスのプロパティ] ダイアログ ボックス

[デバイス] ([オプション] ダイアログ ボックス - [デバイス ツール])

[証明書の選択] ダイアログ ボックス (デバイス)

マネージ デバイス プロジェクト

[ターゲット プラットフォームの変更] ダイアログ ボックス (デバイス)

[<データベース> への接続] ダイアログ ボックス

接続文字列プロパティ、ファイル プロパティのダイアログ ボックス (デバイス)

[デバッグ] ペイン (プロジェクト デザイナ)

[色の作成] ダイアログ ボックス(デバイス)

[デバイス コントロール] タブ (ツールボックス)

[デバイス] ペイン (プロジェクト デザイナ)

[フォント] ダイアログ ボックス (デバイス)

[フォーム ファクタのプロパティ] ダイアログ ボックス (デバイス)

[フォーム ファクタ] ([オプション] ダイアログ ボックス - [デバイス ツール])

[全般] ([オプション] ダイアログ ボックス - [デバイス ツール])

[出力ファイル フォルダ] ダイアログ ボックス (デバイス)

[プロパティ] ウィンドウ (スマート デバイス マネージ プロジェクト)

ネイティブ デバイス プロジェクト

[高度な機能] (MFC スマート デバイス アプリケーション ウィザード)

[アプリケーションの設定] (ATL スマート デバイス プロジェクト ウィザード)

[アプリケーションの設定] (MFC スマート デバイス ActiveX コントロール ウィザード)

[アプリケーションの設定] (MFC スマート デバイス DLL ウィザード)

[アプリケーションの設定] (Win32 スマート デバイス プロジェクト ウィザード)

[アプリケーションの種類] (MFC スマート デバイス アプリケーション ウィザード)

ATL スマート デバイス プロジェクト ウィザード

[コントロール名] (MFC スマート デバイス ActiveX コントロール ウィザード)

[コントロールの設定] (MFC スマート デバイス ActiveX コントロール ウィザード)

[デバッグ] ([<プロジェクト名> プロパティ ページ] ダイアログ ボックス - [構成プロパティ]) (デバイス)

[配置] ([<プロジェクト名> プロパティ ページ] ダイアログ ボックス - [構成プロパティ]) (デバイス)

[ドキュメント テンプレート文字列] (MFC スマート デバイス アプリケーション ウィザード)

[全般] ([<プロジェクト名> プロパティ ページ] ダイアログ ボックス - [構成プロパティ] - [Authenticode 署名をする]) (デバイス)

[生成されたクラス] (MFC スマート デバイス アプリケーション ウィザード)

MFC スマート デバイス ActiveX コントロール ウィザード

MFC スマート デバイス アプリケーション ウィザード

MFC スマート デバイス DLL ウィザード

[プラットフォーム] (ATL スマート デバイス プロジェクト ウィザード)

[プラットフォーム] (MFC スマート デバイス ActiveX コントロール ウィザード)

[プラットフォーム] (MFC スマート デバイス アプリケーション ウィザード)

[プラットフォーム] (MFC スマート デバイス DLL ウィザード)

[プラットフォーム] (Win32 スマート デバイス プロジェクト ウィザード)

[ユーザー インターフェイスの機能] (MFC スマート デバイス アプリケーション ウィザード)

Win32 スマート デバイス プロジェクト ウィザード

デバイス セットアップ プロジェクト

[ビルド] (配置プロジェクトの [プロパティ] ダイアログ ボックス - [構成プロパティ])

[配置] ([<プロジェクト名> プロパティ ページ] ダイアログ ボックス - [構成プロパティ]) (デバイス)

[プロパティ] ウィンドウ (スマート デバイス CAB プロジェクト)

関連するセクション

一般的なユーザー インターフェイス要素 (Visual Studio)

Visual Studio のダイアログ ボックスやウィンドウに表示されるオプションについて説明します。

接続の追加/変更 (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

データ ソースとプロバイダを指定し、接続のテスト方法を提供します。

次の表は、Visual Studio 2005 SP1 に合わせて更新されています。

項目	説明
[データ ソース] ボックス	データ ソースとデータ プロバイダを指定します。
[変更]	[データ ソースの変更] ダイアログ ボックスを開きます。このダイアログ ボックスでは、データ ソースとデータ プロバイダを選択できます。
[データ ソース] オプション ボタン	データ ソースが開発用コンピュータ上にあるか、または ActiveSync によって接続されているデバイス上にあるかを指定します。
[データベース]	データベースのファイル名を指定します。
[作成]	[新しいデータベースを作成する] ダイアログ ボックスを開きます。このダイアログ ボックスでは、新しいデータベースのファイル名とパスワードを指定できます。
[参照]	[データベース ファイルを選択してください。] ダイアログ ボックスを開きます。このダイアログ ボックスでは、既存の SQL Server Mobile Edition データベースまたは SQL Server Compact Edition データベースを選択できます。
[パスワード]	データベースのパスワードを指定します。
[詳細]	[プロパティの詳細] ダイアログ ボックスを開きます。このダイアログ ボックスでは、データベース ファイルの名前とパスを変更したり、パスワードを指定したりできます。
[接続のテスト]	[データベース] ボックスで指定したデータベースへの接続をテストします。

参照

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[高度な機能] (MFC スマート デバイス アプリケーション ウィザード)

MFC スマート デバイス アプリケーション ウィザードの [高度な機能] ページについて説明します。

このページには、ActiveX コントロールや Windows Sockets など、アプリケーションの追加機能用のオプションが用意されています。各セクションで、これらの高度な機能の追加サポートを指定します。

[高度な機能]

[Windows ヘルプ]

この機能は、このリリースではサポートされていません。

[印刷と印刷プレビュー]

この機能は、このリリースではサポートされていません。

MFC ライブラリから CView クラス内のメンバ関数を呼び出すことによって、印刷、印刷設定、および印刷プレビューのコマンドを処理するコードが生成されます。これらの関数に対するコマンドも、ウィザードによってアプリケーションのメニューに追加されます。印刷サポートが使用できるのは、ウィザードの [アプリケーションの種類] ページの [ドキュメント ビュー アーキテクチャ サポート] が指定されたアプリケーションだけです。既定では、ドキュメント/ビュー アプリケーションには印刷サポートが含まれます。

[ActiveX コントロール]

ActiveX コントロールをサポートします (既定値)。このオプションを選択せず、後で ActiveX コントロールをプロジェクトに追加する必要が生じた場合には、アプリケーションのメンバ関数 InitInstance に AfxEnableControlContainer の呼び出しを追加する必要があります。

[Windows ソケット]

この機能は、このリリースではサポートされていません。

Windows Sockets をサポートします。Windows Sockets を使用すると、TCP/IP ネットワーク上で通信するアプリケーションを作成できます。

[[最近使ったファイル] のファイル数]

この機能は、このリリースではサポートされていません。

最近使ったファイルの一覧に表示するファイル数を指定します。既定値は 4 です。

参照

関連項目

MFC スマート デバイス アプリケーション ウィザード

[アプリケーションの設定] (ATL スマート デバイス プロジェクト ウィザード)

新しい ATL スマート デバイス プロジェクトのプラットフォーム設定以外の設定を指定します。

メモ :
スマート デバイス プロジェクトでコード ウィザードを実行中に属性を選択することはサポートされていません。このリリースでは、スマート デバイス プロジェクトの属性付きコードはサポートされていません。

[サーバーの種類]

[ダイナミック リンク ライブラリ (DLL)]

サーバーがダイナミック リンク ライブラリ (DLL: Dynamic Link Library) であり、したがってインプロセス サーバーであることを指定します。

[実行可能なアプリケーション (EXE)]

サーバーが実行可能なアプリケーション (EXE) であり、したがってローカルのアウトプロセス サーバーであることを指定します。このオプションでは、MFC のサポートは提供されません。

[追加のオプション]

[MFC サポート]

サーバーに MFC のサポートを含めることを指定します。このオプションを使用すると、プロジェクトと MFC ライブラリがリンクされ、ライブラリに含まれるクラスや関数にアクセスできます。

このオプションは、プロジェクト内部から MFC 固有のクラスを使用する必要がある場合にのみ選択します。**CString**、CRect クラス、CSize クラス、および CPoint クラスなどのユーティリティ クラスでは、ATL プロジェクトに MFC サポートを追加する必要はありません。

このオプションを選択する場合は、MFC を使用する COM メソッド、Windows プロシージャ、およびエクスポートされた関数すべてに対して、個々の先頭に次の行を追加する必要があります。

```
AFX_MANAGE_STATE(AfxGetStaticModuleState());
```

[マージされたプロキシ/スタブ の生成]

MIDL で生成されるプロキシおよびスタブ コードを有効にします。通常は **DCOM** を使用する SDK で使用します。Pocket PC や Smartphone など、DCOM をサポートしていないプラットフォームではこのオプションを使用できません。

参照

関連項目

ATL スマート デバイス プロジェクト ウィザード

その他の技術情報

デバイス用の ATL リファレンス

デバイス用のユーザー インターフェイス リファレンス

[アプリケーションの設定] (MFC スマート デバイス ActiveX コントロール ウィザード)

MFC スマート デバイス ActiveX コントロール ウィザードの [アプリケーションの設定] ページについて説明します。

新しい MFC スマート デバイス ActiveX プロジェクトのデザインを変更したり、基本機能を追加したりするには、MFC スマート デバイス ActiveX コントロール ウィザードのこのページを使用します。設定値はアプリケーション本体に適用され、コントロールの特定の機能や要素には適用されません。

アプリケーション設定

[ランタイム ライセンス]

コントロールと一緒に配布するユーザー ライセンス ファイルを生成するには、このオプションを選択します。ライセンスは、*projname*.lic というテキスト ファイルです。デザイン時環境でコントロールのインスタンスを作成するには、このファイルをコントロールの DLL と同じディレクトリに置く必要があります。通常、このファイルはコントロールと一緒に配布しますが、ユーザーはこのファイルを配布しません。

参照

関連項目

MFC スマート デバイス ActiveX コントロール ウィザード

[アプリケーションの設定] (MFC スマート デバイス DLL ウィザード)

MFC スマート デバイス DLL ウィザードの [アプリケーションの設定] ページについて説明します。

MFC スマート デバイス DLL ウィザードのこのページを使用して、新しい MFC スマート デバイス DLL プロジェクトの基本機能のデザインと追加を行います。

[DLL の種類]

作成する MFC Smart Device DLL の種類を選択します。

[共有 MFC DLL を使用する通常の DLL]

MFC ライブラリを共有 DLL としてプログラムにリンクする場合は、このオプションを選択します。このオプションを使用すると、作成した DLL と呼び出し元のアプリケーション間で MFC オブジェクトを共有できません。プログラムは実行時に MFC ライブラリを呼び出します。このオプションを使用すると、プログラムが MFC ライブラリを使用する複数の実行可能ファイルで構成される場合に、必要なディスクとメモリの量が削減されます。Windows CE プログラムと MFC プログラムは、どちらも DLL 内の関数を呼び出すことができます。この種類のプロジェクトを使用する場合は、MFC DLL を再配布する必要があります。

[MFC をスタティックにリンクした通常の DLL]

プログラムと MFC ライブラリをビルド時に静的にリンクする場合は、このオプションを選択します。Windows CE プログラムと MFC プログラムは、どちらも DLL 内の関数を呼び出すことができます。このオプションを選択するとプログラムのサイズは大きくなりますが、この種類のプロジェクトでは MFC DLL を再配布する必要がありません。作成した DLL と呼び出し元のアプリケーション間では MFC オブジェクトを共有できません。

[MFC 拡張 DLL]

実行時にプログラムから MFC ライブラリを呼び出す場合、および作成した DLL と呼び出し元のアプリケーション間で MFC オブジェクトを共有する場合は、このオプションを選択します。このオプションを使用すると、プログラムが MFC ライブラリを使用する複数の実行可能ファイルで構成される場合に、必要なディスクとメモリの量が削減されます。作成した DLL の関数を呼び出すことができるのは、MFC プログラムだけです。この種類のプロジェクトを使用する場合は、MFC DLL を再配布する必要があります。

[追加の機能]

MFC DLL でオートメーションをサポートする必要があるかどうか、および Windows ソケットをサポートする必要があるかどうかを選択します。

[オートメーション]

[オートメーション] を選択すると、他のプログラムに実装されたオブジェクトをプログラムで操作できます。また、[オートメーション] を選択することによって、プログラムが他のオートメーション クライアントに公開されます。詳細については、「オートメーション」を参照してください。

[Windows ソケット]

このオプションを選択すると、プログラムで Windows Sockets がサポートされます。Windows Sockets を使用すると、TCP/IP ネットワーク上で通信するプログラムを作成できます。Windows ソケットをサポートする MFC DLL を作成すると、CWinApp::InitInstance によってソケットのサポートが初期化され、StdAfx.h に AfxSock.h という MFC ヘッダー ファイルがインクルードされます。

参照

関連項目

MFC スマート デバイス DLL ウィザード

[アプリケーションの設定] (Win32 スマート デバイス プロジェクト ウィザード)

ウィザードのこのページを使用して、Windows CE プロジェクトのオプションを設定します。

[アプリケーションの種類]

指定したタイプのアプリケーションが作成されます。

オプション	説明
[Windows アプリケーション]	Windows CE API の呼び出しを使用して Visual C または Visual C++ で記述された実行可能アプリケーション (.exe) を作成し、グラフィカル ユーザー インターフェイスを作成します。
[コンソール アプリケーション]	アプリケーションのコンソール アプリケーション アーキテクチャを作成します。このオプションを使用できるのは、コンソール アプリケーションをサポートするプラットフォームの場合だけです。
[DLL]	MFC クラスではなく Windows CE API クラスの呼び出しを使用する、Visual C または Visual C++ で記述されたバイナリ ファイル (DLL) を作成します。このファイルは、複数のアプリケーションで同時に使用できる関数の共有ライブラリとして機能します。DLL がシンボルをエクスポートするように指定できます。
[スタティック ライブラリ]	実行可能ファイルの作成時にプログラムにリンクされるオブジェクト (およびそのオブジェクトの関数とデータ) を含むファイルを作成します。スタティック ライブラリは、MFC ベースのプログラムおよび非 MFC のプログラムの両方にリンクできます。スタティック ライブラリ プロジェクトには、ATL のサポートを追加できません。

[追加オプション]

アプリケーションのサポートとオプションを定義します。定義内容はアプリケーションの種類によって異なります。

オプション	説明
[空のプロジェクト]	プロジェクト ファイルは空白にします。一連のソース コード ファイル (.cpp ファイル、ヘッダー ファイル、アイコン、ツール バー、ダイアログ ボックスなど) があり、Visual C++ 開発環境でプロジェクトを作成する場合は、最初に空のプロジェクトを作成してから、そのプロジェクトにファイルを追加する必要があります。このオプションはスタティック ライブラリ プロジェクトには使用できません。
[シンボルのエクスポート]	DLL プロジェクトがシンボルをエクスポートするように指定します。
[プリコンパイル済みのヘッダー]	スタティック ライブラリ プロジェクトでプリコンパイル済みヘッダーを使用するように指定します。

[サポートの追加]

Visual C++ に用意されているライブラリの 1 つに対するサポートを追加します。

オプション	説明
[ATL]	そのデバイス用に ATL (Active Template Library) のクラスのサポートがプロジェクトに組み込まれます。このオプションは、スタティック ライブラリ プロジェクトには使用できません。
[MFC]	そのデバイス用に MFC (Microsoft Foundation Class) ライブラリのサポートがプロジェクトに組み込まれます。

参照

関連項目

Win32 スマート デバイス プロジェクト ウィザード

[アプリケーションの種類] (MFC スマート デバイス アプリケーション ウィザード)

新しい MFC スマート デバイス プロジェクトのデザインを変更したり、基本機能を追加したりするには、MFC スマート デバイス アプリケーション ウィザードのこのページを使用します。

[アプリケーションの種類]

アプリケーションで作成するドキュメント サポートの種類を指定します。選択したアプリケーション タイプに応じて、アプリケーションで使用できるユーザー インターフェイスのオプションが決定されます。ユーザー インターフェイスのオプションの詳細については、「[ユーザー インターフェイスの機能] (MFC スマート デバイス アプリケーション ウィザード)」を参照してください。ドキュメントの種類の詳細については、以下のトピックを参照してください。

- SDI と MDI
- フレーム ウィンドウ
- フレーム ウィンドウ クラス
- ドキュメント、ビュー、フレームワーク
- ダイアログ ボックス

オプション	説明
[シングル ドキュメント]	シングル ドキュメント インターフェイス (SDI: Single Document Interface) アーキテクチャのアプリケーションを作成します。ビュー クラスは CView に基づいています。ビューの基本クラスの変更は、このウィザードの [生成されたクラス] ページで行います。たとえば、フォーム ベースのアプリケーションを作成するには、ビュー クラスとして CFormView を使用できます。 このタイプのアプリケーションでは、ドキュメントのフレーム ウィンドウでフレーム内に保持できるドキュメントは 1 つだけです。
[ダイアログ ベース]	CDialog に基づくダイアログ クラスを使用し、アプリケーションに対してダイアログ ベース アーキテクチャを作成します。
[doc list を持つシングル ドキュメント]	CDocList クラスに基づいて、アプリケーション用に単一のドキュメント アーキテクチャを作成します。このオプションは、Pocket PC など、このクラスをサポートするプラットフォームでのみ使用できます。 **メモ :** CdocList は、このリリースでは使用できません。

[ドキュメント ビュー アーキテクチャ サポート]

CDocument クラス 基本クラスおよび CView クラス 基本クラス (既定の場合) を使用して、アプリケーションにドキュメント/ビュー アーキテクチャを組み込みます。たとえば、非 MFC アプリケーションを移植する場合、またはコンパイル済みの実行可能ファイルのサイズを縮小する場合には、このチェック ボックスをオフにします。ドキュメント/ビュー アーキテクチャを含まないアプリケーションは、既定では CWinApp クラス から派生し、ディスク ファイルのドキュメントを開くために必要な MFC サポートを含んでいません。

[リソース言語]

リソースで使用する言語を設定します。Visual Studio のインストール内容に応じて、システムで利用できる言語が一覧表示されます。システム言語以外の言語を選択する場合は、その言語に対応するテンプレート フォルダを事前にインストールしておく必要があります。[リソース言語] ボックスで使用できる既定の言語以外の言語リソースのインストールの詳細については、「ウィザードでサポートされるその他の言語」を参照してください。選択した言語は、このウィザードの [ドキュメント テンプレート文字列] ページの [ローカライズされる文字列] に反映されます。

[MFC の使用法]

MFC ライブラリとのリンク方法を指定します。既定では、MFC は共有 DLL としてリンクされます。

オプション	説明

[共有 DLL で MFC を使用]	MFC ライブラリを共有 DLL としてアプリケーションにリンクします。アプリケーションは実行時に MFC ライブラリを呼び出します。このオプションを使用すると、MFC ライブラリを使用する複数の実行可能ファイルでアプリケーションが構成される場合に、必要なディスクとメモリの量が削減されます。Windows CE アプリケーションおよび MFC アプリケーションは両方とも DLL 内の関数を呼び出すことができます。
[スタティック ライブラリで MFC を使用]	ビルド時にアプリケーションをスタティック MFC ライブラリにリンクします。

参照

関連項目

MFC スマート デバイス アプリケーション ウィザード

ATL スマート デバイス プロジェクト ウィザード

ATL (Active Template Library) は、テンプレート ベースの一連の C++ クラスです。これらの C++ クラスを使用することにより、COM オブジェクトを簡単に作成できます。ATL スマート デバイス プロジェクト ウィザードでは、COM オブジェクトを格納するための構造体を持つプロジェクトを作成します。

[概要]

作成している ATL プロジェクトの現在のプロジェクト設定を表示します。プロジェクトの既定の設定は以下のとおりです。

- プロジェクトの既定のターゲット プラットフォームは、プラットフォームの一覧の最初のプラットフォームです。既定のインストールでの既定のプラットフォームは Pocket PC 2003 ですが、Windows CE 5.0 SDK をインストールおよびアンインストールすると、新しいアプリケーションの既定のターゲットが変更される場合があります。
- ダイナミック リンク ライブラリ。使用しているサーバーが DLL であり、インプロセス サーバーになることを示します。

[プラットフォーム]

ウィザードの左の列で [プラットフォーム] をクリックして、[プラットフォーム] (ATL スマート デバイス プロジェクト ウィザード) ページを表示します。このページを使用して、現在のプロジェクトの SDK を選択します。

[アプリケーションの設定]

ウィザードの左の列で [アプリケーションの設定] をクリックして、[アプリケーションの設定] (ATL スマート デバイス プロジェクト ウィザード) ページを表示します。このページを使用して、サーバーの種類 (DLL または EXE) および特定の追加オプションを選択します。

メモ :
ATL EXE プロジェクトでは、空白を含む長いファイル名の付いたディレクトリ (たとえば、\Program Files\My Server\) に登録されている COM ローカル サーバーは、COM によってインスタンス化されたとき起動しません。 この状況を回避するには、空白を含まない短いファイル名を使用してクラスを登録します。詳細については、「方法 : プライマリ プロジェクト出力のリモート パスを指定する」を参照してください。既定値には空白が含まれています。

参照

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス
デバイス用の ATL リファレンス

[ビルド] (配置プロジェクトの [プロパティ] ダイアログ ボックス - [構成プロパティ])

現在選択しているスマート デバイス配置プロジェクトのビルド設定を指定します。プロジェクトの名前はタイトル バーに表示されます。

[配置プロジェクトのプロパティ] ダイアログ ボックスを表示するには、ソリューション エクスプローラでプロジェクトを右クリックし、[プロパティ] をクリックします。

[構成]

[Debug] および [Retail] など、ビルドの構成を指定します。

[プラットフォーム]

アクティブなプロジェクトが対象とするプラットフォームを指定します。

[構成プロパティ] フォルダ

右側のパネルに表示するプロパティのカテゴリを指定します。

[構成マネージャ]

[構成マネージャ] ダイアログ ボックスを開きます。

[出力ファイル名]

ビルドの実行時に CAB ファイルを配置する場所を指定します。既定以外の場所を選択するには、[参照] をクリックします。

[Authenticode で署名]

Authenticode 署名を使用して出力ファイルに署名するかどうかを指定します。詳細については、「デバイス プロジェクトにおけるセキュリティ」を参照してください。

[証明書]

ファイルへの署名に使用する Authenticode の証明書ファイルを指定します。証明書ファイルを選択するには、[ストアから選択] をクリックします。

[ストアから選択]

[証明書の選択] ダイアログ ボックスを開きます。このダイアログ ボックスでは、使用する証明書を選択できます。詳細については、「方法 : デバイス プロジェクトで証明書をインポートおよび適用する」を参照してください。

参照

関連項目

[構成マネージャ] ダイアログ ボックス

その他の技術情報

デバイス プロジェクトにおけるセキュリティ

[ターゲット プラットフォームの変更] ダイアログ ボックス (デバイス)

マネージ デバイス プロジェクトのターゲット プラットフォームを、.NET Compact Framework の同一バージョンが対象となる他のプラットフォームに変更します。

このダイアログ ボックスを開くには、[プロジェクト] をクリックして [ターゲット プラットフォームの変更] をクリックします。このメニュー項目は、Visual Studio IDE でマネージ プロジェクトが開かれている場合だけ表示されます。

[Current platform]

アクティブなプロジェクトのプラットフォームを表示します。

[変更]

現在のプラットフォームが対象にする .NET Compact Framework と同じバージョンが対応しているその他のプラットフォームの一覧を表示します。

参照

その他の技術情報

.NET Compact Framework を使用したデバイスのプログラミング
デバイス用のユーザー インターフェイス リファレンス

[構成マネージャ] (スマート デバイス プロジェクト ウィザード)

このダイアログ ボックスは、ソリューション ビルド構成やプロジェクト構成を作成および編集する場合に使用します。ソリューション ビルド構成に加えた変更はすべて、[ソリューション <ソリューション名> プロパティ ページ] ダイアログ ボックスの [構成] ページに反映されます。[構成マネージャ] には、[ビルド] メニュー、[ソリューション <ソリューション名> プロパティ ページ] ダイアログ ボックス、またはメイン ツール バーにある [ソリューション構成] ドロップダウン リストからアクセスできます。

[アクティブ ソリューション構成]

有効なソリューション ビルド構成を表示します。このドロップダウン リストまたはメイン ツール バーの [構成] ドロップダウン リストは、アクティブなソリューション構成を変更する場合に使用します。新規のソリューション構成を作成するには、ドロップダウン リストの [<新規作成...>] を選択します。既存のソリューション構成を変更するには、ドロップダウン リストの [<編集...>] を選択します。

[アクティブ ソリューション プラットフォーム]

ソリューションのビルドに使用できるプラットフォームを表示します。アクティブ ソリューションのプラットフォームを変更すると、この変更はソリューション内のすべてのプロジェクトに適用されます。新規のソリューション プラットフォームを作成するには、ドロップダウン リストの [<新規作成...>] を選択します。既存のソリューション プラットフォームを変更するには、ドロップダウン リストの [<編集...>] を選択します。

[プロジェクトのコンテキスト (ビルドまたは配置するプロジェクト構成をチェック)]

選択したソリューション ビルド構成の [プロジェクトのコンテキスト] (ビルドまたは配置するプロジェクト構成をチェック) の下にある各エントリには、プロジェクト名、構成の種類とプラットフォームのドロップダウン リスト、およびビルドと配置 (有効な場合) を行うプロジェクトを選択するためのチェック ボックスが含まれます。選択した構成の種類とプラットフォームの組み合わせによって、使用されるプロジェクト構成が決まります。列ヘッダーをクリックして、グリッド内の列を並べ替えます。

[プロジェクト]

現在のソリューションに存在するプロジェクトの名前を表示します。

[構成]

必要なプロジェクトのビルドの種類と使用できるすべてのビルドの種類が表示されます。新しいプロジェクトのビルドの種類を作成するには、このドロップダウン リストの [<新規作成...>] を選択します。既存のプロジェクトのビルドの種類の名前を変更するには、[<編集...>] を選択します。

[プラットフォーム]

目的のビルドを実行するプラットフォームと、プロジェクトで使用できるすべてのプラットフォームの一覧が表示されます。新しいプラットフォームを追加するには、このドロップダウン リストの [<新規作成...>] を選択します。既存のプラットフォームを編集するには、[<編集...>] を選択します。対象とするプロジェクトの種類によって、複数のプラットフォームを追加できるかどうかや、プロジェクトに追加できるプラットフォームの種類が決まります。プロジェクトのプラットフォームを追加すると、新規プロジェクト構成が作成されます。

[ビルド]

プロジェクトが現在のソリューション構成によってビルドされるかどうかを指定します。選択されていないプロジェクトは、プロジェクトの依存関係にかかわらずビルドされません。ビルドの実行が指定されていないプロジェクトも、デバッグ、実行、パッケージ化、およびソリューションの配置は可能です。

[配置]

有効な場合、選択したソリューション ビルド構成の [実行] コマンドか [配置] コマンドを使用した場合に、プロジェクトを配置するかどうかを指定します。このチェック ボックスは、配置プロジェクトの場合だけに表示されます。

参照

その他の技術情報

.NET Compact Framework を使用したデバイスのプログラミング
デバイス用のユーザー インターフェイス リファレンス

[エミュレータ ブートストラップの構成] ダイアログ ボックス

1 つ以上の XML ファイルを Pocket PC と Smartphone の構成インフラストラクチャに挿入します。

このダイアログ ボックスを表示するには、まず [ツール] メニューの [オプション] をクリックし、[デバイス ツール]、[デバイス] の順にクリックします。次に、エミュレータを選択し、[プロパティ] をクリックします。最後に、ブートストラップを選択し、[構成] をクリックします。

[XML プロビジョニング ファイル]

1 つ以上の XML ファイルを Pocket PC と Smartphone の構成インフラストラクチャに挿入して、ネットワーク、UI、およびセキュリティに関する設定を調整します。

参照

処理手順

方法 : Visual Studio でデバイス エミュレータを起動する

関連項目

[デバイス] ([オプション] ダイアログ ボックス - [デバイス ツール])

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[TCP/IP トランスポートの構成] ダイアログ ボックス

デバイス プロジェクトのトランスポートの特性を指定するためのオプションが用意されています。

このダイアログ ボックスを表示するには、まず [ツール] メニューの [オプション] をクリックします。次に、[デバイス ツール] をクリックし、[デバイス] をクリックし、[トランスポート] ボックスの横の [構成] をクリックします。[構成] ボタンが淡色表示されている場合、選択したトランスポートは構成できません。

[固定ポート番号を使用する]

常にこのポートが使用されます。デバイス上のインターネット プロトコル (IP: Internet Protocol) 番号またはポート番号を変更しない場合は、このオプションを選択します。

[ActiveSync を使用して IP アドレスを自動的に取得]

IP 設定を自動的に割り当てます。この設定には、デバイスと開発用コンピュータとの間に ActiveSync 通信が必要になります。

[特定の IP アドレスを使用]

使用する IP アドレスを選択します。既定のアドレスはローカル ホスト アドレスです。

参照

関連項目

[デバイス] ([オプション] ダイアログ ボックス - [デバイス ツール])
[配置] ダイアログ ボックス (デバイス)

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[デバイスへの接続] ダイアログ ボックス

接続する物理デバイスまたはエミュレータの指定をユーザーに求めます。

このダイアログ ボックスを表示するには、[ツール] メニューの [デバイスへの接続] をクリックします。

[プラットフォーム]

作業中のプロジェクトでは使用できないものも含めて、インストールされているすべてのプラットフォームの一覧が表示されます。デバイス プロジェクトが開かれている場合は、そのプラットフォームが自動的に選択されます。

[すべてのプラットフォーム] を選択すると、インストールされているすべてのデバイスの一覧が表示されます。

[デバイス]

[プラットフォーム] ボックスで選択したプラットフォームにインストールされているデバイスの一覧を表示します。

[接続]

[デバイス] ボックスで選択したデバイスへの接続を開きます。デバイスがエミュレータである場合、エミュレータは、開発用コンピュータのデスクトップ上に開かれます。

参照

関連項目

[デバイス] ([オプション] ダイアログ ボックス - [デバイス ツール])

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[<データベース> への接続] ダイアログ ボックス

既存の接続にデータベースのアクセスに必要なパスワードが含まれていない場合に、パスワードを入力するように求めます。

このダイアログ ボックスは、データ ソース構成ウィザードでパスワードが必要な場合に表示されます。

[データベース]

データベースのファイル名 (*.sdf)。

[ファイル]

データベース ファイルへの絶対パス。

[パスワード]

データベースにアクセスするために必要なパスワード。

参照

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

接続文字列プロパティ、ファイル プロパティのダイアログ ボックス (デバイス)

このトピックは、Visual Studio 2005 SP1 に合わせて更新されています。

.xsd ファイルのプロパティとして、デバイスで実行されているアプリケーションの実行時の接続文字列を指定します。

実行時の接続文字列は、自動生成された接続文字列が適切でない場合に使用される、オプションのプロパティです。生成される接続文字列は、.xsd スキーマ ファイル内の接続文字列が使用され、デザイン時の接続文字列と見なされます。デザイン時の文字列によって、Visual Studio から、デザイン時にデータベースに接続し、スキーマ情報を取得できます。このデザイン時の接続文字列が実行時に適切ではない場合、実行時の [接続文字列] プロパティを使用して、生成されたコードにシリアル化されるカスタムの接続文字列を指定できます。

以下は、Visual Studio 2005 SP1 に合わせて更新されています。

たとえば、Microsoft SQL Mobile および Microsoft SQL Server 2005 Compact Edition の接続文字列には、データベースのパスが含まれます。この場合、生成される接続文字列は、データベースが実行バイナリと同じフォルダにあることを前提とした接続文字列に変換されます。Microsoft SQL Mobile データベースまたは SQL Server Compact Edition データベースが、デバイス上の異なる場所にある場合、.xsd ファイルの [接続文字列] プロパティを使用して、生成された接続文字列をカスタムの接続文字列でオーバーライドできます。

.xsd スキーマ ファイルの [接続文字列] プロパティでカスタムの接続文字列を指定しても、.xsd スキーマ ファイルに格納されているデザイン時の接続文字列は変更されません。またデザイン時の機能は通常どおりに動作します。詳細については、「方法 : デザイン時の接続文字列を変更する (デバイス)」を参照してください。

このウィンドウの他のファイル プロパティは、スマート デバイス プロジェクト固有のものではありません。詳細については、「ファイルのプロパティ」を参照してください。

[プロパティ ウィンドウ] を開くには、ソリューション エクスプローラで .xsd ファイルを選択し、[表示] メニューの [プロパティ ウィンドウ] ウィンドウをクリックします。

[接続文字列]

実行時にデバイス上のデータベースに接続する接続文字列を指定します。

ボックスを空にすると、.xsd スキーマ ファイルに格納されている既定の接続文字列が使用されます。

参照

処理手順

方法 : 実行時の接続文字列を変更する (デバイス)

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[コントロール名] (MFC スマート デバイス ActiveX コントロール ウィザード)

このページでは、作成するコントロールに対してコントロール クラスとプロパティ ページ クラスのクラス名、タイプ名、およびタイプ ID を指定します。[短い名前] 以外のすべてのフィールドは個別に編集できます。[短い名前] のテキストを変更すると、このページにある他のすべてのフィールドの名前に変更内容が反映されます。コントロールの作成時に簡単に識別できるように、すべての名前がこの方法で付けられています。

[短い名前]

コントロールの省略名を指定します。既定では、この名前は [新しいプロジェクト] ダイアログ ボックスで指定したプロジェクト名によって決まります。個別にフィールドを変更しない限り、指定した名前に基づいてクラス名、タイプ名、およびタイプ ID が決まります。

[コントロール クラス名]

既定のコントロール クラス名は、短い形式の名前に基づき、先頭に **C**、末尾に **Ctrl** が付きます。たとえば、コントロールの短い形式の名前が **Price** の場合、コントロール クラスの名前は **CPriceCtrl** です。

[コントロール .h ファイル]

既定のヘッダー ファイル名は、短い形式の名前に基づき、末尾に **Ctrl** が付きます。また、拡張子 **.h** も付きます。たとえば、コントロールの短い形式の名前が **Price** の場合、ヘッダー ファイルの名前は **PriceCtrl.h** です。このフィールドの名前は、コントロール クラス名と一致する必要があります。

[コントロール .cpp ファイル]

既定のヘッダー ファイル名は、短い形式の名前に基づき、末尾に **Ctrl** が付きます。また、拡張子 **.cpp** も付きます。たとえば、コントロールの短い形式の名前が **Price** の場合、ヘッダー ファイルの名前は **PriceCtrl.cpp** です。このフィールドの名前は、ヘッダー名と一致する必要があります。

[コントロールのタイプ名]

既定のコントロールのタイプ名は、短い形式の名前に基づき、末尾に **Control** が付きます。たとえば、コントロールの短い形式の名前が **Price** の場合、コントロール クラスのタイプ名は **Price Control** です。このフィールドの値を変更する場合は、名前に継承が示されていることを確認してください。

[コントロールのタイプ ID]

コントロール クラスのコントロール タイプ ID を設定します。コントロールをプロジェクトに追加すると、この文字列はレジストリに書き込まれます。コンテナ アプリケーションは、この文字列を使用してコントロールのインスタンスを作成します。既定のコントロールのタイプ ID は、[新しいプロジェクト] ダイアログ ボックスで指定したプロジェクト名と、短い形式の名前によって決まります。このフィールドの名前は、タイプ名と同じであることが必要です。既定のコントロールのタイプ ID は、*ProjectName.ShortName*Ctrl.1 という形式になります。このダイアログ ボックスで短い形式の名前を変更した場合、コントロールのタイプ ID は、*ProjectName.NewShortName*Ctrl.1 という形式になります。

[プロパティ ページ クラス名]

既定のプロパティ ページ クラス名は、短い形式の名前に基づき、先頭に **C**、末尾に **PropPage** が付きます。たとえば、コントロールの短い形式の名前が **Price** の場合、プロパティ ページ クラス名は **CPricePropPage** です。このフィールドの名前は、末尾に **PropPage** を付けたコントロール クラス名と同じであることが必要です。

[プロパティ ページ .h ファイル]

既定のプロパティ ページのヘッダー ファイル名は短い形式の名前に基づき、末尾に **PropPage** が付きます。また、拡張子 **.h** も付きます。たとえば、コントロールの短い形式の名前が **Price** の場合、プロパティ ページのヘッダー ファイル名は **PricePropPage.h** です。この名前は、クラス名と同じである必要があります。

[プロパティ ページ .cpp ファイル]

既定のプロパティ ページの実装ファイル名は短い形式の名前に基づき、末尾に **PropPage** が付きます。また、拡張子 **.cpp** も付きます。たとえば、コントロールの短い形式の名前が **Price** の場合、プロパティ ページのヘッダー ファイル名は **PricePropPage.cpp** です。この名前は、ヘッダー ファイル名と同じである必要があります。

[プロパティ ページのタイプ名]

既定のプロパティ ページのタイプ名は、短い形式の名前に基づき、末尾に **Property Page** が付きます。たとえば、コントロールの短い形式の名前が **Price** の場合、プロパティ ページのタイプ名は **Price Property Page** です。このフィールドの値を変更する場合は、名前にコントロール クラスが示されていることを確認してください。

[プロパティ ページのタイプ ID]

プロパティ ページ クラスの ID を設定します。コントロールがプロジェクトに適用されると、この文字列がレジストリに書き込まれます。コンテナ ア

プリケーションは、この文字列を使用してコントロールのプロパティ ページのインスタンスを作成します。既定のプロパティ ページのタイプ ID は、[新しいプロジェクト] ダイアログ ボックスで指定したプロジェクト名と、短い形式の名前によって決まります。このフィールドの名前は、タイプ名と同じであることが必要です。既定のプロパティ ページのタイプ ID は、*ProjectName.ShortName*PropPage.1 という形式になります。このダイアログ ボックスで短い形式の名前を変更した場合、プロパティ ページのタイプ ID は、*ProjectName.NewShortName*PropPage.1 という形式になります。

参照

関連項目

MFC スマート デバイス ActiveX コントロール ウィザード

[コントロールの設定] (MFC スマート デバイス ActiveX コントロール ウィザード)

MFC スマート デバイス ActiveX コントロール ウィザードの [コントロールの設定] ページについて説明します。

このページのオプションを使用して、コントロールの動作を指定します。たとえば、既存の標準 Windows CE コントロールのタイプに基づいてコントロールを作成したり、コントロールの動作や外観を最適化したりできます。また、コントロールが他のコントロールのコンテナとして機能するように指定することもできます。

コントロールの効果を最大限にするためのこのページのオプション選択の詳細については、「MFC ActiveX コントロール : 最適化」を参照してください。

[次に基づいてコントロールを作成]

コントロールが継承するコントロールのタイプを一覧から選択できます。一覧には、commctrl.h の MFC アプリケーションに公開された追加のコモン コントロールが含まれています。選択項目に応じて、*ProjName*Ctrl.cpp ファイルの **PreCreateWindow** 関数のコントロールのスタイルが決まります。詳細については、「MFC ActiveX コントロール : Windows コントロールのサブクラス化」を参照してください。

BUTTON	ボタン コントロール
COMBOBOX	コンボ ボックス コントロール
EDIT	エディット コントロール
LISTBOX	リスト ボックス コントロール
SCROLLBAR	スクロール バー コントロール
STATIC	スタティック コントロール
msctls_progress32	プログレス バー コモン コントロール
msctls_statusbar32	ステータス バー コモン コントロール
msctls_trackbar32	トラック バー コモン コントロール
msctls_updown32	スピン ボタン コモン コントロール
SysHeader32	ヘッダー コモン コントロール
SysListView32	リスト ビュー コモン コントロール
SysTabControl32	タブ コモン コントロール
SysTreeView32	ツリー ビュー コモン コントロール

[追加の機能]

[表示時にアクティブ]

コントロールが画面に表示されたときに、そのコントロールのウィンドウを作成するように指定します。[表示時にアクティブ] は、既定ではオンになっています。コンテナでコントロールが (マウス クリックなどによって) 必要になるまでコントロールをアクティブにしない場合は、この機能をオフにします。この機能をオフにすると、必要になるまでウィンドウを作成しないため、コントロールが最適化されます。このオプションの詳細については、「[表示時にアクティブ] オプションのオフ」を参照してください。

[ランタイムで非表示]

実行時にコントロールのユーザー インターフェイスを表示しないことを指定します。表示が不要なコントロールとしては、たとえばタイマ コント

ロールなどがあります。

[[バージョン情報] ボックス ダイアログあり]

コントロールで標準の Windows CE の [バージョン情報] ダイアログ ボックスを使用して、バージョン番号と著作権情報を表示するように指定します。

メモ：
ユーザーがコントロールのヘルプにアクセスする方法は、ヘルプの実装方法、およびコンテナのヘルプがコントロールのヘルプに統合されているかどうかによって異なります。ヘルプの統合の詳細については、「MFC ActiveX コントロールへの状況依存ヘルプの追加」を参照してください。

このオプションをオンにすると、プロジェクトのコントロール クラス (*CProjName*Ctrl.cpp) に **AboutBox** コントロール メソッドが追加され、プロジェクトのディスパッチ マップに [バージョン情報] が追加されます。既定では、このオプションが設定されています。

[最適化された描画コード]

同じデバイス コンテキストに描画されるコンテナのコントロールの描画がすべて完了した後で、コンテナの **GDI** オブジェクトを自動的に元の状態に復元するように指定します。この機能の詳細については、「コントロールの描画の最適化」を参照してください。

[ウィンドウなしでアクティブ]

コントロールがアクティブになったときにコントロールのウィンドウを作成しないように指定します。この機能をオンにすると、四角形以外のコントロールや透過的なコントロールを使用できます。また、ウィンドウなしのコントロールには、ウィンドウ付きのコントロールに必要なシステム オーバーヘッドが不要です。ウィンドウなしのコントロールでは、クリッピングを行わないデバイス コンテキストやちらつきなしのアクティベーションは使用できません。1996 年以前に作成されたコンテナでは、ウィンドウなしのアクティベーションをサポートしません。このオプションの使用方法の詳細については、「ウィンドウなしのアクティベーション」を参照してください。

[クリッピングを行わないデバイス コンテキスト]

コントロール ヘッダー (projnamectrl.h) の COleControl::GetControlFlags をオーバーライドし、**COleControl** による IntersectClipRect の呼び出しを無効にします。[クリッピングを行わないデバイス コンテキスト] をオンにすると、処理速度が多少向上します。[ウィンドウなしでアクティブ] をオンにした場合は、この機能を使用できません。詳細については、「クリッピングを行わないデバイス コンテキストの使用」を参照してください。

[ちらつきなしでアクティブ化]

コントロールがアクティブな状態とアクティブでない状態との間で発生する描画処理とそれに伴うちらつきをなくします。[ウィンドウなしでアクティブ] をオンにした場合は、この機能を使用できません。このオプションをオンにすると、COleControl::GetControlFlags によって返されるフラグに **noFlickerActivate** フラグが含まれます。詳細については、「ちらつきなしのアクティベーションの提供」を参照してください。

[[オブジェクトの挿入] で使用]

有効なコンテナに対してコントロールを [オブジェクトの挿入] ダイアログ ボックスで使用できることを指定します。このオプションをオンにすると、AfxOleRegisterControlClass によって返される一連のフラグに **afxRegInsertable** フラグが含まれます。[オブジェクトの挿入] ダイアログ ボックスでは、ユーザーが新規または既存のオブジェクトを複合ドキュメントに追加できます。

[非アクティブ時のマウス ポインタ通知]

コントロールがアクティブかどうかにかかわらず、コントロールでマウス ポインタの通知を処理できるように指定します。このオプションをオンにすると、COleControl::GetControlFlags によって返される一連のフラグに **pointerInactive** フラグが含まれます。このオプションの使用方法の詳細については、「コントロールがアクティブでないときのマウスとの対話」を参照してください。

[シンプル フレーム コントロールとして動作]

コントロールの **OLEMISC_SIMPLEFRAME** ビットを設定することにより、そのコントロールが他のコントロールのコンテナであることを指定します。詳細については、「Simple Frame Site Containment」を参照してください。

[非同期でプロパティを読み込む]

以前の非同期データのリセットを有効にし、コントロールの非同期プロパティの新規読み込みを実行します。

参照

関連項目

MFC スマート デバイス ActiveX コントロール ウィザード

[デバッグ] ペイン (プロジェクト デザイナ)

コマンド ライン引数、作業ディレクトリ、およびスタートアップ プロジェクトを指定するためのユーザー入力が用意されています。

プロジェクト デザイナを開くには、ソリューション エクスプローラで [<プロジェクト名>] を右クリックし、ショートカット メニューの [プロパティ] をクリックします。

[スタート プロジェクト]

現在のプロジェクトがスタートアップ プロジェクトであることを指定します。

[外部プログラムの開始]

現在のプロジェクトをスタートアップ プロジェクトとして指定しない場合、代わりに起動時の処理として機能する外部プログラムを指定します。

[コマンドライン引数]

コマンド ライン引数を指定します。

2 つのドロップダウン リストがあります。

- 構成
- [プラットフォーム]

ドロップダウン リスト

[構成] ドロップダウン リスト

[Debug] および [Release] など、選択できる他の構成の一覧を表示します。

[プラットフォーム] ドロップダウン リスト

[アクティブ (Any CPU)] など、使用可能なプラットフォームの一覧を表示します。

参照

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[デバッグ] ([<プロジェクト名 > プロパティ ページ] ダイアログ ボックス - [構成プロパティ]) (デバイス)

[構成プロパティ] フォルダの [デバッグ] プロパティ ページでは、デバッガのプロパティをいくつか指定します。

[起動するデバッガ]

デバッグ セッション用に起動できるデバッガの一覧が表示されたドロップダウン リスト。

[リモート実行ファイル]

デバッグ セッションを開始するときに使用するリモートの実行可能ファイルの名前。

[コマンド引数]

リモートの実行可能ファイルに渡す引数。

2 つのドロップダウン リストと 1 つのボタンがあります。

- [構成マネージャ]
- [プラットフォーム]

ドロップダウン リスト

[構成] ドロップダウン リスト

[Debug] および [Release] など、選択できる他の構成の一覧を表示します。

[プラットフォーム] ドロップダウン リスト

Smartphone 2003 や Pocket PC 2003 など、使用できる他のプラットフォームの一覧を表示します。

[構成マネージャ] ボタン

[構成マネージャ] ボタンをクリックすると、プロジェクトの [構成マネージャ] ダイアログ ボックスにアクセスできます。

参照

関連項目

プロパティ ページ (C++)

[色の作成] ダイアログ ボックス(デバイス)

色合いを指定したり、カスタムの色合いを追加するときに使用します。色合いを変更すると、それに合わせて赤、緑、および青の値が適切に変更されます。値は 0 から 239 の範囲です。

このダイアログ ボックスをデザイン ビューから表示するには、色のプロパティを持つ任意のコンポーネント (TextBox コントロールなど) を右クリックし、ショートカット メニューの [プロパティ] をクリックし、[プロパティ] ウィンドウで [ForeColor] などの色のプロパティを選択します。その後、プロパティ リストの右側にあるドロップダウン矢印をクリックし、[カスタム] をクリックし、[カスタム] タブの下部にある空のタイルのいずれかを右クリックします。

色の割り当ては、インストールされた各プラットフォームごとに、リソースとしてプラットフォーム デザイナ アセンブリに格納されます。RGB および名前付きの色はデスクトップ プラットフォームとデバイス プラットフォームの両方で同じですが、システム カラーは異なります。システム カラーは、デザイン時に変換されません。選択できる色は、.NET Compact Framework でサポートされている色に制限されているため、プラットフォームによって異なる場合があります。

プラットフォームのカラー マップ スキーマが見つからない場合は、デスクトップの既定値が使用されます。

マウスまたは Tab キーを使用して、[色合い]、[鮮やかさ]、または [明るさ] ボックスにカーソルを配置します。F1 キーを押して、追加情報を示すツールヒントを表示します。

参照

関連項目

[フォント] ダイアログ ボックス (デバイス)

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[配置] ダイアログ ボックス (デバイス)

デバイス プロジェクトに関連付けられているプラットフォームでアプリケーションを実行できる、インストール済みのデバイスの一覧を表示します。

このダイアログ ボックスが開かれるのは、次の場合です。

- [デバイス ツール] オプション ([ツール] の [オプション] をクリックすると使用できます) の [全般] ページで [デバイス プロジェクトの配置前に選択できるデバイスを表示] を選択した場合。
- [配置] コマンドを発行 ([ビルド] メニューから明示的に発行、または [デバッグ] メニューの [開始] の実行後に暗黙的に発行) した場合。

[デバイス]

エミュレータ イメージなど、インストールされているデバイスの一覧を表示します。

デバイスを選択して、このダイアログ ボックスを閉じると、そのデバイスがプロジェクトの対象デバイスとして選択されます。

[アプリケーションの配置時にこのダイアログを表示する]

オンにした場合は、プロジェクトを配置するたびに [<プロジェクト名> の配置] ダイアログ ボックスが表示されます。このオプションは、[全般] ([オプション] ダイアログ ボックス - [デバイス ツール])のチェック ボックスを使用して選択することもできます。

[配置]

ターゲット デバイスに接続します。

[開始] から暗黙的に開かれた場合、デバイスへの接続後にアプリケーションを起動します。

参照

処理手順

方法 : 既定のデバイスを変更する (マネージ プロジェクト)
方法 : 既定のデバイスを変更する (ネイティブ プロジェクト)

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[配置] ([<プロジェクト名 > プロパティ ページ] ダイアログ ボックス - [構成プロパティ]) (デバイス)

マルチプラットフォーム デバイス配置の導入により、デバイスごとにデバイス配置プロパティを設定できます。[構成プロパティ] フォルダの [配置] プロパティ ページは、ソリューション エクスプローラでプロジェクト ノードを選択したときに、プロジェクトのプロパティ ページに表示されます。このページには、以下の 2 つのセクションに分類されるプロパティと、2 つのドロップダウン リストと 1 つのボタンが含まれています。

- [全般]
- [サーバー側のアクション]

[全般]

[配置デバイス]

配置先のデバイスを指定します。

追加ファイル

デバイスに配置する追加ファイルを、ソース ディレクトリとターゲット ディレクトリ、および、そのファイルを対象のデバイスに登録するかどうかを示す値と共に、セミコロンで区切って指定します。プライマリ プロジェクト出力と配置可能なコンテンツとしてマークされたプライマリ プロジェクト ファイルは含めません。このダイアログ ボックスで使用できるマクロの詳細については、「ビルドのコマンドとプロパティのマクロ」を参照してください。

[リモート ディレクトリ]

アプリケーションのコピー先となる配置デバイス上のディレクトリを指定します。既定では、%CSIDL_PROGRAM_FILES%\ になります。この場合、アプリケーションが Smartphone または Pocket PC 上の適切な場所に配置されます。

[サーバー側のアクション]

[出力の登録]

プライマリ プロジェクト出力の DLL を対象のデバイスに登録するかどうかを指定します。このドロップダウン リストでは、[はい] または [いいえ] を選択できます。MFC ActiveX プロジェクトおよび ATL プロジェクトの場合は、[はい] を選択してください。

2 つのドロップダウン リストと 1 つのボタンがあります。

- [構成マネージャ]
- [プラットフォーム]

ドロップダウン リスト ボックス

[構成] ボックス

[Debug] および [Release] など、選択できる他の構成の一覧を表示します。

[プラットフォーム] ボックス

Smartphone 2003 や Pocket PC 2003 など、使用できる他のプラットフォームの一覧を表示します。

[構成マネージャ] ボタン

[構成マネージャ] ボタンをクリックすると、プロジェクトの [構成マネージャ] ダイアログ ボックスにアクセスできます。

参照

関連項目

プロパティ ページ (C++)

デバイスの [コンポーネント] タブ (ツールボックス)

表示されるコンポーネントは、Windows フォームにドラッグできますが、スマート デバイス プロジェクトの *visual* 要素ではないものです。Visual Studio 2005 には、[ポインタ]、[SerialPort]、[タイマー]、[InputPanel]、[Notification]、[HardwareButton]、[MessageQueue]、および [BindingSource] が含まれます。

参照

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[デバイス コントロール] タブ (ツールボックス)

対象となるプラットフォームでサポートされているコントロールを表示します。このタブは、スマート デバイス プロジェクトがアクティブなときに使用できます。

[デバイス コントロール] タブが表示されていない場合は、ツールボックスを右クリックし、[ツールボックスのリセット] または [すべて表示] を選択します。

ツールボックスには、[表示] メニューからアクセスできます。

参照

関連項目

ツールボックス

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[デバイスのプロパティ] ダイアログ ボックス

デバイス (エミュレータと物理デバイスの両方を含む) のプロパティを指定するためのオプションを示します。

このダイアログ ボックスを表示するには、[ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックして、[デバイス] をクリックします。次に、デバイスを選択し、[プロパティ] をクリックします。

[デバイス上の既定の出力場所]

アプリケーションを配置するリモート デバイス上のパスを指定します。

[トランスポート]

トランスポートを指定します。

Visual Studio には、エミュレータ用の DMA トランスポートと、物理デバイス用の TCP 接続トランスポートが用意されています。

メモ：
DMA トランスポートは、エミュレータのための高速で信頼性の高い接続を実現します。エミュレータでも TCP トランスポートを使用できますが、DMA トランスポートよりも信頼性に劣ります。エミュレータでは、特別な理由がある場合にのみ TCP トランスポートを使用してください。

[構成]

[トランスポートの構成] ダイアログ ボックスを開きます。トランスポートが TCP/IP の場合は、固定ポート番号を使用することを選択したり、IP アドレスの決定方法を指定したりできます。[構成] ボタンが淡色表示されている場合、選択したトランスポートは構成できません。

[ブートストラップ]

ブートストラップを指定します。

[構成]

物理デバイスの場合は、[手動ブートストラップの構成] ダイアログ ボックスを開きます。このダイアログ ボックスでは、ポート番号とデバイス IP アドレスを指定できます。

[構成] ボタンが淡色表示されている場合、選択したブートストラップは構成できません。

[デバイスの切断を検出]

デバイスが接続されているかどうかをすばやく検出できます。

カーネル独立トランスポート層 (KITL: Kernel Independent Transport Layer) を使用してデバッグするには、このチェック ボックスをオフにします。

このプロパティへの変更は、次にデバイスに接続したときに反映されます。

[エミュレータ オプション]

[エミュレータのプロパティ] ダイアログ ボックスを開きます。このダイアログ ボックスでは、表示や接続の特性など、エミュレータのプロパティを設定できます。物理デバイスを選択した場合、このオプションは表示されません。

参照

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[デバイス] ペイン (プロジェクト デザイナ)

デバイスを選択したり、スマート デバイス プロジェクトの特性を指定したりするためのコントロールが用意されています。[デバイス] ペインはデバイス プロジェクトでのみ表示されます。

プロジェクト デザイナを開くには、ソリューション エクスプローラでプロジェクトを右クリックし、ショートカット メニューの [プロパティ] をクリックします。

UI

[ターゲット デバイス]

アプリケーションの対象デバイスを指定します。デバイスには、物理的なハードウェアだけでなく、エミュレータも含まれます。

[ファイル フォルダの出力]

アプリケーションのダウンロード先とするリモート デバイス上のパスを指定します。

[Service Pack を含む最新バージョンの .NET Compact Framework を配置]

開発用コンピュータ上の .NET Compact Framework のバージョンがデバイスまたはエミュレータ上のものよりも新しい場合は、新しい方のバージョンを配置します。

[この証明書を使用してプロジェクト出力を署名]

一覧に表示されている証明書を使用して署名する出力ファイルを指定します。

ボックスに 1 つ以上の証明書を設定するには、[証明書の選択] をクリックします。

[証明書の選択]

[証明書の選択] ダイアログ ボックスを開きます。このダイアログ ボックスでは、使用する証明書を選択できます。

詳細については、「方法 : デバイス プロジェクトで証明書をインポートおよび適用する」を参照してください。

[証明書をデバイスにプロビジョニング]

デバイスを準備するかどうかを指定します。デバイスを準備する場合は、証明書を特権のある証明書ストアと特権のない証明書ストアのどちらに配置するかを指定します。

詳細については、「方法 : Visual Basic プロジェクトまたは Visual C# プロジェクトでデバイスを用意する」を参照してください。

参照

その他の技術情報

デバイス プロジェクトにおけるセキュリティ
.NET Compact Framework を使用したデバイスのプログラミング
デバイス用のユーザー インターフェイス リファレンス

[デバイス] ([オプション] ダイアログ ボックス - [デバイス ツール])

デバイス プロジェクトの構成設定を変更するためのコントロールについて説明します。

このダイアログ ボックスを表示するには、[ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックして、[デバイス] をクリックします。

[デバイスを表示するプラットフォーム]

作業中のプロジェクトでは使用できないものも含めて、インストールされているすべてのプラットフォームの一覧が表示されます。デバイス プロジェクトが開かれている場合は、そのプラットフォームが自動的に選択されます。

[すべてのプラットフォーム] を選択すると、インストールされているすべてのデバイスの一覧が表示されます。

[デバイス]

[デバイスを表示するプラットフォーム] ボックスで選択したプラットフォーム用にインストールされているデバイスの一覧から、構成するデバイスを選択します。デバイス プロジェクトが開かれている場合は、そのデバイスが自動的に選択されます。

[既定のデバイス]

[デバイスを表示するプラットフォーム] ボックスで選択したプラットフォームに使用できるデバイスから、既定の配置先を設定します。[すべてのプラットフォーム] が選択されている場合、このコントロールは淡色表示になり、使用できません。既定値への変更は保存され、新しいすべてのプロジェクトに適用されますが、現在アクティブなプロジェクトには影響しません。

[名前を付けて保存]

[名前を付けて保存] ダイアログ ボックスを開きます。このダイアログ ボックスでは、[デバイス] ボックスで選択したデバイスのコピーを作成できます。

[名前の変更]

[デバイス] ボックスで選択したデバイスの名前を変更します。

[削除]

[デバイス] ボックスで選択したデバイスを削除します。

[削除] ボタンが淡色表示されている場合、選択したデバイスは削除できません。以下の場合、デバイスは削除できません。

- プロジェクトと一緒にインストールされた場合
- プラットフォームの既定のデバイスの場合
- 現在開いているプロジェクトの対象のデバイスとして選択されている場合

[プロパティ]

[デバイス] ボックスで選択したデバイスの [デバイスのプロパティ] ダイアログ ボックスを開きます。

参照

関連項目

[配置] ダイアログ ボックス (デバイス)
[全般] ([オプション] ダイアログ ボックス - [デバイス ツール])
[フォーム ファクタ] ([オプション] ダイアログ ボックス - [デバイス ツール])

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[ドキュメント テンプレート文字列] (MFC スマート デバイス アプリケーション ウィザード)

MFC スマート デバイス アプリケーション ウィザードの [ドキュメント テンプレート文字列] ページについて説明します。

MFC スマート デバイス アプリケーション ウィザードのこのページでは、ドキュメントの管理とローカリゼーションに役立つ次のオプションを指定または変更します。[アプリケーションの種類] の [ドキュメント ビュー アーキテクチャ サポート] を含むアプリケーションでは、ドキュメント テンプレート文字列を使用できます。ダイアログ ボックスでは使用できません。ほとんどのドキュメント テンプレート文字列は表示される文字列で、アプリケーションのユーザーによって使用されるため、このウィザードの [アプリケーションの種類] ページで指定した [リソース言語] の言語にローカライズされます。

[ローカライズされない文字列]

ユーザー ドキュメントを作成するアプリケーションが対象になります。ファイル拡張子とファイル型 ID を指定すると、ユーザーはドキュメントを簡単に開いたり、印刷したり、保存したりできます。これらのアイテムはユーザーではなくシステムで使用されるため、ローカライズされません。

オプション	説明
[ファイル拡張子]	アプリケーションの使用時に、ユーザーが保存するドキュメントに関連付けられるファイル拡張子を設定します。たとえば、プロジェクトの名前が Widget の場合は、ファイル拡張子 .wgt を指定できます。ファイル拡張子を入力する場合、ピリオドは付けないでください。 ファイル拡張子を指定した場合は、ユーザーがアプリケーションのドキュメント アイコンをプリンタ アイコンにドロップすると、そのアプリケーションを起動せずにエクスプローラからドキュメントを印刷できます。 ここで拡張子を指定しない場合は、ファイルの保存時にユーザーがファイル拡張子を指定する必要があります。ウィザードでは既定のファイル拡張子は用意されていません。
[ファイルの種類の ID]	ドキュメントの種類のラベルをレジストリに設定します。

[ローカライズされる文字列]

アプリケーションのユーザーが読んだり使用したりする、アプリケーションやドキュメントに関連付けられた文字列を生成するため、文字列はローカライズされます。

オプション	説明
[言語]	[ローカライズされる文字列] の下にあるすべてのボックスの文字列の言語を指定します。このボックスの値を変更するには、MFC スマート デバイス アプリケーション ウィザードの [アプリケーションの種類] ページの [リソース言語] で適切な言語を選択します。
[メイン フレームのキャプション]	メイン アプリケーション フレームの上部に表示されるテキストを設定します。既定ではプロジェクト名になります。
[ドキュメントの種類の名前]	アプリケーションのドキュメントをグループ化できるドキュメントの種類を識別します。既定ではプロジェクト名になります。既定値を変更しても、このダイアログ ボックスのほかのオプションには影響ありません。
[フィルタ名]	ユーザーが特定のファイル型のファイルを検索するときに指定する名前を設定します。このオプションは、Windows 標準の [開く] ダイアログ ボックスおよび [名前を付けて保存] ダイアログ ボックスの [ファイルの種類] で使用できます。既定の名前は、プロジェクト名 + Files + [ファイル拡張子] で指定した拡張子です。たとえば、プロジェクト名が Widget で、ファイル拡張子が .wgt の場合、既定の [フィルタ名] は Widget Files (*.wgt) になります。

[ファイルの新しい短い名前]	複数の新規ドキュメント テンプレートがある場合に、Windows 標準の [新規作成] ダイアログ ボックスに表示される名前を設定します。アプリケーションがオートメーション サーバーの場合、この名前はオートメーション オブジェクトの短い形式の名前として使用されます。既定ではプロジェクト名になります。
[ファイルの種類の長い名前]	ファイル型名をレジストリに設定します。アプリケーションがオートメーション サーバーの場合、この名前はオートメーション オブジェクトの長い形式の名前として使用されます。既定の名前は、プロジェクト名 + Document です。

参照

関連項目

MFC スマート デバイス アプリケーション ウィザード

[フォント] ダイアログ ボックス (デバイス)

デバイス プロジェクトのフォントを指定するためのオプションが用意されています。

デザイン ビューからこのダイアログ ボックスを表示するには、フォント プロパティを持つコンポーネント (たとえば、フォーム自体) を右クリックし、ショートカット メニューの [プロパティ] をクリックします。その後、[プロパティ] ウィンドウの Font を選択し、[(...)] をクリックします。

このダイアログ ボックスは、デスクトップのこれに相当するダイアログ ボックスとは少し異なります。具体的には、[すべてのフォントを表示] チェック ボックスは、開発用コンピュータで使用できるフォントの一部がターゲット デバイスではサポートされていない可能性があるということに対応しています。このチェック ボックスをオフにすると、ターゲット デバイス用に定義されているフォントだけが表示されます。

メモ :
また、独自のフォントを開発用コンピュータから追加することもできます。独自のフォントを追加するには、追加するフォントを選択し、フォント ファイルをデスクトップからデバイス上の Windows\Fonts ディレクトリにコピーします。

[フォント]

使用できるフォントの一覧が表示されます。

[スタイル]

指定したフォントに対して使用できるスタイルの一覧が表示されます。

[サイズ]

指定したフォントに対して使用できるポイント数の一覧が表示されます。

[すべてのフォントを表示]

アクティブなデバイス プロジェクトで使用できるフォントの一覧が表示されます。このチェック ボックスをオンにすると、開発用コンピュータで使用できるすべてのフォントの一覧が表示されます。

[取り消し線]

フォントに取り消し線を引くかどうかを指定し、フォントに使用できる色を指定します。

[下線]

フォントに下線を引くかどうかを指定し、フォントに使用できる色を指定します。

[サンプル]

指定したフォント設定でのテキストの表示状態を示すサンプルが表示されます。

各プラットフォームの既定のフォントは次のとおりです。

- Pocket PC: Courier New、Tahoma
- Smartphone: Nina
- Windows CE: Arial、Tahoma

参照

関連項目

[色の作成] ダイアログ ボックス(デバイス)

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[フォーム ファクタのプロパティ] ダイアログ ボックス (デバイス)

デバイス プロジェクトでフォーム ファクタの設定を変更するためのオプションが用意されています。

プロジェクトが開いているときに変更を加えた場合、デザイナ ウィンドウを閉じて再び開くまで、表示には変更が反映されません。

このダイアログ ボックスを表示するには、[ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックして、[フォーム ファクタ] をクリックします。フォーム ファクタを選択し、[プロパティ] をクリックします。

[スキンのプレビュー]

選択したスキンのプレビューを表示します。

[スキン]

選択したフォーム ファクタに関連付けられたスキンを設定します。別のスキンを選択するには、[参照] ボタンをクリックして適切な XML ファイルを開きます。

[回転サポートを有効にする]

デザイン時の回転サポートを有効にします。

[水平解像度]

Windows フォーム デザイナでこのフォーム ファクタを対象とするフォームの水平解像度を設定します。

[垂直解像度]

Windows フォーム デザイナでこのフォーム ファクタを対象とするフォームの垂直解像度を設定します。

[スキンの表示]

デザイン時およびエミュレート時に、選択したフォーム ファクタのスキンをフォームの周囲に表示します。この既定値を選択しても、デザイナでは、[オプション] ダイアログ ボックスの [デバイス ツール] の [全般] で [スキンの表示] を選択している場合しかスキンが表示されません。

[画面の幅]

開発用コンピュータでのエミュレータ表示の幅のピクセル数を設定します。

[スキンの表示] を選択している場合は、エミュレータ表示の幅が既に設定されているため、このコントロールは淡色表示になります。

[画面の高さ]

開発用コンピュータでのエミュレータ表示の高さのピクセル数を設定します。

[スキンの表示] を選択している場合は、エミュレータ表示の高さが既に設定されているため、このコントロールは淡色表示になります。

[色の深度]

エミュレータ表示のピクセルあたりのビット数を設定します。通常、この値は 8、16、および 32 になります。

[スキンの表示] を選択している場合は、エミュレータ表示の色深度が既に設定されているため、このコントロールは淡色表示になります。

参照

処理手順

方法 : フォームの向きおよび解像度を変更する (デバイス)

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[フォーム ファクタ] ([オプション] ダイアログ ボックス - [デバイス ツール])

マネージ デバイス プロジェクトのフォーム ファクタおよびそのプロパティを指定するためのオプションが用意されています。

各プラットフォームには既定のフォーム ファクタがあります。これは、プロジェクトでの必要に応じて変更できます。たとえば、必要に応じて、画面のサイズおよび解像度のどちらも選択できます。

プロジェクトが開いているときに変更を加えた場合、デザイナ ウィンドウを閉じて再び開くまで、表示には変更が反映されません。

このダイアログ ボックスを表示するには、[ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックして、[フォーム ファクタ] をクリックします。

[プラットフォームのフォーム ファクタを表示する]

選択したプラットフォーム用にインストールされているフォーム ファクタの一覧を表示します。[すべてのプラットフォーム] を選択すると、インストールされているすべてのプラットフォームのすべてのフォーム ファクタが表示されます。

[フォーム ファクタ]

[プラットフォームのフォーム ファクタを表示する] で選択したプラットフォーム用にインストールされているフォーム ファクタの一覧を表示します。

[既定のフォーム ファクタ]

[プラットフォームのフォーム ファクタを表示する] で選択したプラットフォームを対象とする新しいすべてのプロジェクトの既定のデザイン時のフォーム ファクタを設定します ([すべてのプラットフォーム] を選択した場合、このオプションは淡色表示になります)。

[名前を付けて保存]

[名前を付けて保存] ダイアログ ボックスを開きます。このダイアログ ボックスでは、選択したフォーム ファクタのコピーを保存できます。

[名前の変更]

選択したフォーム ファクタの名前を変更します。

[削除]

選択したフォーム ファクタを削除します。

[削除] ボタンが淡色表示されている場合、選択したフォーム ファクタは削除できません。次の場合、フォーム ファクタは削除できません。

- プロジェクトと一緒にインストールされた場合
- プラットフォームの既定のフォーム ファクタの場合

[プロパティ]

選択したフォーム ファクタの [フォーム ファクタのプロパティ] ダイアログ ボックスを開きます。

参照

関連項目

[フォーム ファクタのプロパティ] ダイアログ ボックス (デバイス)

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[全般] ([<プロジェクト名 > プロパティ ページ] ダイアログ ボックス - [構成プロパティ] - [Authenticode 署名をする]) (デバイス)

ソリューション エクスプローラでプロジェクト ノードを選択するとプロジェクトのプロパティ ページに表示される、[構成プロパティ] フォルダの [Authenticode 署名をする] プロパティ ページでは、プロパティが次の 1 つのセクションに分類されています。

- [全般]

[全般]

[Authenticode で署名]

プロジェクトのプライマリ出力に署名するかどうかを指定します。

[デバイスのプロビジョニング]

ターゲット デバイスを用意するかどうかを指定します。また、ターゲット デバイスを用意する場合に証明書を格納する場所を指定します。

2 つのドロップダウン リストと 1 つのボタンがあります。

- [構成マネージャ]
- [プラットフォーム]

ドロップダウン リスト

[構成] ドロップダウン リスト

[Debug] および [Release] など、選択できる他の構成の一覧を表示します。

[プラットフォーム] ドロップダウン リスト

Smartphone 2003 や Pocket PC 2003 など、使用できる他のプラットフォームの一覧を表示します。

[構成マネージャ] ボタン

[構成マネージャ] ボタンをクリックすると、プロジェクトの [構成マネージャ] ダイアログ ボックスにアクセスできます。

参照

関連項目

プロパティ ページ (C++)

[全般] ([オプション] ダイアログ ボックス - [デバイス ツール])

マネージ デバイス プロジェクトの全般オプションを設定します。

このダイアログ ボックスを表示するには、[ツール] メニューの [オプション] をクリックし、[デバイス ツール] をクリックして、[全般] をクリックします。

[デバイス プロジェクトの配置前に選択できるデバイスを表示]

プロジェクトを配置するたびに [<プロジェクト名> の配置] ダイアログ ボックスが表示されます。たとえば、物理デバイスとそのエミュレータの間などでデバイスを頻繁に切り替える場合は、このオプションをオンにします。1 つのデバイスのみに配置する場合は、このチェック ボックスをオフにすることで、[<プロジェクト名> の配置] ダイアログ ボックスを配置を開始するたびに表示しないようにできます。

このオプションは、[<プロジェクト名> の配置] ダイアログ ボックスで選択することもできます。

[Windows フォーム デザイナでスキンを表示する]

このオプションが [フォーム ファクタのプロパティ] ダイアログ ボックス (デバイス)で有効になっている場合、デザイン時にフォームの周りにスキンが表示されます。

参照

関連項目

[デバイス] ([オプション] ダイアログ ボックス - [デバイス ツール])
[配置] ダイアログ ボックス (デバイス)
[デバイス] ([オプション] ダイアログ ボックス - [デバイス ツール])

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[生成されたクラス] (MFC スマート デバイス アプリケーション ウィザード)

MFC スマート デバイス アプリケーション ウィザードの [生成されたクラス] ページについて説明します。

このページには、プロジェクトで生成される基本クラスとファイルの名前が表示されます。既定では、表示される名前は [新しいプロジェクト] ダイアログ ボックスで指定したプロジェクト名によって決まります。以下で説明するように、これらの名前のほとんどを変更できます。

[生成されたクラス]

プロジェクトに対して作成されるクラスの名前を指定します。既定では、この名前はプロジェクト名によって決まります。既定の MFC プロジェクトでは、C*ProjName*View、C*ProjName*App、C*ProjName*Doc、および C*ProjName*MainFrame の各クラスが作成されます。このページの他のすべてのボックスは、[生成されたクラス] ボックスの一覧で現在選択されているクラスの情報を反映します。

クラス名を変更するには、[クラス名] ボックスを使用します。

[クラス名]

[生成されたクラス] ボックスの一覧で現在選択されているクラスの名前を指定します。このボックスがアクティブな場合は、クラス名を変更できます。[クラス名] ボックスからフォーカスを移すと、選択したクラス名に対する変更内容が [生成されたクラス] ボックスの一覧にすべて表示されます。

[.h ファイル]

[生成されたクラス] ボックスの一覧で現在選択されているクラスのヘッダー ファイル名を指定します。このテキスト ボックスがアクティブな場合は、ヘッダー ファイル名を変更できます。

[基本クラス]

[生成されたクラス] ボックスの一覧で現在選択されているクラスの基本クラスの名前を指定します。このボックスがアクティブな場合は、基本クラスとして別のクラスを一覧から選択できます。

[.cpp ファイル]

選択したクラスに関連付けられているソース コード ファイルの名前を指定します。このテキスト ボックスがアクティブな場合は、実装ファイル名を変更できます。

参照

関連項目

MFC スマート デバイス アプリケーション ウィザード

MFC スマート デバイス ActiveX コントロール ウィザード

MFC スマート デバイス ActiveX コントロール ウィザードについて説明します。

ActiveX コントロールとは特殊なオートメーション サーバーであり、再利用できるコンポーネントです。ActiveX コントロールをホストするアプリケーションは、そのコントロールのオートメーション クライアントです。再利用できるコンポーネントを作成する場合は、このウィザードを使用してコントロールを作成します。詳細については、「MFC ActiveX コントロール」を参照してください。

また、MFC スマート デバイス アプリケーション ウィザードを使用して、オートメーション サーバー MFC スマート デバイス アプリケーションを作成することもできます。

MFC 起動プログラムには、C++ ソース (.cpp) ファイル、リソース (.rc) ファイル、およびプロジェクト (.vcproj) ファイルが含まれます。これらの初期ファイルで生成されたコードは、MFC に基づいています。

概要

ウィザードのこのページでは、作成する MFC ActiveX コントロール プロジェクトの現在のアプリケーション設定が示されます。既定では、このウィザードで、次のようなプロジェクトが作成されます。

- プロジェクトの既定のターゲット プラットフォームは、プラットフォームの一覧の最初のプラットフォームです。既定のインストールでは、既定のプラットフォームは Pocket PC 2003 です。Windows CE 5.0 SDK をインストールしてアンインストールすると、新しいアプリケーションの既定のターゲットを変更でき、Smartphone 2003 などの他のプラットフォームを追加できます。
- 既定のプロジェクトでは、ランタイム ライセンスが生成されません。
- プロジェクトには、プロジェクト名に基づくコントロール クラスとプロパティ ページ クラスが含まれます。
- このコントロールは既存のどの Windows CE コントロールにも基づいていません。このコントロールは画面に表示されるとアクティブになり、ユーザー インターフェイスを備え、[バージョン情報] ダイアログ ボックスを含みます。

ターゲット プラットフォームを変更するには、ウィザードの左の列にある [プラットフォーム] をクリックし、必要な変更を加えます。

アプリケーション設定を変更するには、ウィザードの左の列にある [アプリケーションの設定] をクリックし、必要な変更を加えます。

コントロール名を変更するには、ウィザードの左の列にある [コントロール名] をクリックし、必要な変更を加えます。

コントロールの設定を変更するには、ウィザードの左の列にある [コントロールの設定] をクリックし、必要な変更を加えます。

新しいプロジェクトを作成した後に、コンパイラが **_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA** の定義について警告を発した場合は、メイン ヘッダー ファイル内にこのフラグを定義する必要があります。

```
#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA
```

これは特に、Windows Mobile 内の Web サービスを使用して Windows Mobile プラットフォーム上で COM オブジェクトを作成したり、ATL COM オブジェクトを作成したりするシナリオに当てはまります。

参照

その他の技術情報

Visual C++ デバイス プロジェクトの作成と移植

MFC スマート デバイス アプリケーション ウィザード

MFC スマート デバイス アプリケーション ウィザードでは、組み込み機能を持つアプリケーションが生成されます。このアプリケーションをコンパイルすると、Windows CE 実行可能アプリケーション (.exe) の基本機能が実装されます。

[概要]

MFC スマート デバイス アプリケーション ウィザードの [概要] ページでは、作成する MFC スマート デバイス アプリケーションの現在のアプリケーション設定が示されます。既定では、このウィザードで、次のようなプロジェクトが作成されます。

[プラットフォーム]

- プロジェクトの既定のターゲット プラットフォームは、プラットフォームの一覧の最初のプラットフォームです。既定のインストールでの既定のプラットフォームは Pocket PC 2003 ですが、Windows CE 5.0 SDK をインストールおよびアンインストールすると、新しいアプリケーションの既定のターゲットを変更したり、Smartphone 2003 などの新しいターゲットを追加したりできます。

[アプリケーションの種類]

- SDI と MDIを含むプロジェクトが作成されます。
- プロジェクトで、ドキュメント/ビュー アーキテクチャを使用します。
- プロジェクトで、共有 DLL 内の MFC を使用します。

[ドキュメント テンプレート文字列]

- 既定のドキュメント テンプレート文字列のプロジェクト名が使用されます。

[ユーザー インターフェイスの機能]

- プロジェクトにはコマンド バーが実装されます。

[高度な機能]

- プロジェクトでは高度な機能はサポートされません。

[生成されたクラス]

- プロジェクトのビュー クラスは CView クラス から派生します。
- プロジェクトのアプリケーション クラスは CWinApp クラス から派生します。
- プロジェクトのドキュメント クラスは CDocument クラス から派生します。
- プロジェクトのメイン フレーム クラスは CFrameWnd クラス から派生します。

新しいプロジェクトを作成した後に、コンパイラが **_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA** の定義について警告を発した場合は、メイン ヘッダー ファイル内にこのフラグを定義する必要があります。

```
#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA
```

これは特に、Windows Mobile 内の Web サービスを使用して Windows Mobile プラットフォーム上で COM オブジェクトを作成したり、ATL COM オブジェクトを作成したりするシナリオに当てはまります。

参照

その他の技術情報

Visual C++ を使用したデバイスのプログラミング

MFC スマート デバイス DLL ウィザード

MFC スマート デバイス DLL ウィザード、特に [概要] ページについて説明します。

MFC DLL ウィザードで MFC DLL プロジェクトを作成すると、組み込みの機能を備えた、動作可能な初期アプリケーションが作成されます。このアプリケーションをコンパイルすると、DLL の基本機能が実装されます。MFC 起動プログラムには、C++ ソース (.cpp) ファイル、リソース (.rc) ファイル、およびプロジェクト (.vcproj) ファイルが含まれます。これらの初期ファイルで生成されたコードは、MFC に基づいています。詳細については、Visual Studio でプロジェクト用に作成された Readme.txt のファイルの説明、および「MFC DLL ウィザードによって生成されるクラスと関数」を参照してください。

概要

ウィザードのこのページには、作成する MFC スマート デバイス DLL プロジェクトの現在のプロジェクト設定が表示されます。既定では、プロジェクトに次のオプションが設定されています。

- プロジェクトの既定のターゲット プラットフォームは、プラットフォームの一覧の最初のプラットフォームです。既定のインストールでの既定のプラットフォームは Pocket PC 2003 ですが、Windows CE 5.0 SDK をインストールおよびアンインストールすると、新しいアプリケーションの既定のターゲットが変更される場合があります。
- このプロジェクトは、MFC を静的にリンクした標準 DLL として作成されます。

既定のプラットフォームを変更するには、ウィザードの左の列にある [プラットフォーム] をクリックし、必要な変更を加えます。

既定のアプリケーション設定を変更するには、ウィザードの左の列にある [アプリケーションの設定] をクリックし、必要な変更を加えます。

新しいプロジェクトを作成した後に、コンパイラが **_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA** の定義について警告を発した場合は、メイン ヘッダー ファイル内にこのフラグを定義する必要があります。

```
#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA
```

これは特に、Windows Mobile 内の Web サービスを使用して Windows Mobile プラットフォーム上で COM オブジェクトを作成したり、ATL COM オブジェクトを作成したりするシナリオに当てはまります。

参照

その他の技術情報

Visual C++ を使用したデバイスのプログラミング

[出力ファイル フォルダ] ダイアログ ボックス (デバイス)

デバイスのファイル システム内でプロジェクトの出力を配置する場所を指定します。

[出力ファイル フォルダ] ダイアログ ボックスを開くには、ソリューション エクスプローラで [<プロジェクト名>] を右クリックし、ショートカット メニューの [プロパティ] をクリックします。[デバイス] ペインを選択し、[出力ファイル フォルダ] ボックスの右にある省略記号 ([...]) ボタンをクリックします。

[デバイス上の出力場所]

配置ディレクトリがあるデバイスのルート ディレクトリを指定します。

[サブディレクトリ]

プロジェクトの出力を配置するディレクトリを指定します。

[出力ファイル フォルダ (生成済み)]

プロジェクトの出力を配置する場所の完全修飾ディレクトリを表示します。

参照

その他の技術情報

.NET Compact Framework を使用したデバイスのプログラミング
デバイス用のユーザー インターフェイス リファレンス

[プラットフォーム] (ATL スマート デバイス プロジェクト ウィザード)

ATL スマート デバイス プロジェクト ウィザードの [プラットフォーム] ページでは、新しい ATL プロジェクトでサポートされるプラットフォームと命令セットを指定します。

[プラットフォーム] ウィンドウ

プロジェクトのターゲット プラットフォームを指定します。使用できるプラットフォームは、Visual Studio 2005 と共にインストールされるか、別個の Windows Mobile ソフトウェア開発キット (SDK: Software Development Kit) または Windows CE ソフトウェア開発キットと共にインストールされます。

[命令セットとデバイス パネル]

[プラットフォーム] ウィンドウで選択したプラットフォームでサポートされている命令セットおよびデバイスの一覧を表示します。

[インストール済み SDK]

Smartphone 2003 や Pocket PC 2003 など、使用できるその他のプラットフォームの一覧を表示します。ボタンを使用して、1 つまたは全部を [選択された SDK] に追加できます。ボタンを使用して、1 つまたは全部を [選択された SDK] から削除することもできます。

[選択された SDK]

Smartphone 2003 や Pocket PC 2003 など、選択されている対象プラットフォームの一覧を表示します。

メモ :

プロジェクトを作成した後、デバイス プロジェクトの作成後にアプリケーション ウィザードを使用していつでも Smartphone 2003 などのプラットフォームを追加できます。ただし、最初の作成後に新しいプラットフォームをプロジェクトに追加した場合、新しいプラットフォームの Additional Files 構成プロパティには、依存するランタイム DLL は追加されません。

参照

関連項目

ATL スマート デバイス プロジェクト ウィザード

[プラットフォーム] (MFC スマート デバイス ActiveX コントロール ウィザード)

MFC スマート デバイス ActiveX コントロール ウィザードの [プラットフォーム] ページでは、新しい ActiveX プロジェクトでサポートされるプラットフォームと命令セットを指定します。

[プラットフォーム] ウィンドウ

プロジェクトのターゲット プラットフォームを指定します。使用できるプラットフォームは、Visual Studio 2005 と共にインストールされるか、別個の Windows Mobile SDK または Windows CE SDK と共にインストールされます。

[命令セットとデバイス パネル]

[プラットフォーム] ウィンドウで選択したプラットフォームでサポートされている命令セットおよびデバイスの一覧を表示します。

[インストール済み SDK]

Smartphone 2003 や Pocket PC 2003 など、使用できるその他のプラットフォームの一覧を表示します。ボタンを使用して、1 つまたは全部を [選択された SDK] に追加できます。ボタンを使用して、1 つまたは全部を [選択された SDK] から削除することもできます。

[選択された SDK]

Smartphone 2003 や Pocket PC 2003 など、選択されている対象プラットフォームの一覧を表示します。

メモ :

プロジェクトを作成した場合、デバイス プロジェクトの作成後にアプリケーション ウィザードを使用していつでも Smartphone 2003 などのプラットフォームを追加できます。ただし、最初の作成後に新しいプラットフォームをプロジェクトに追加した場合、新しいプラットフォームの Additional Files 構成プロパティには、依存するランタイム DLL は追加されません。

参照

関連項目

MFC スマート デバイス ActiveX コントロール ウィザード

[プラットフォーム] (MFC スマート デバイス アプリケーション ウィザード)

MFC スマート デバイス プロジェクト ウィザードの [プラットフォーム] ページでは、新しい MFC プロジェクトでサポートされるプラットフォームと命令セットを指定します。

[プラットフォーム] ウィンドウ

プロジェクトのターゲット プラットフォームを指定します。使用できるプラットフォームは、Visual Studio 2005 と共にインストールされるか、別個の Windows Mobile SDK または Windows CE SDK と共にインストールされます。

[命令セットとデバイス パネル]

[プラットフォーム] ウィンドウで選択したプラットフォームでサポートされている命令セットおよびデバイスの一覧を表示します。

[インストール済み SDK]

Smartphone 2003 や Pocket PC 2003 など、使用できるその他のプラットフォームの一覧を表示します。ボタンを使用して、1 つまたは全部を [選択された SDK] に追加できます。ボタンを使用して、1 つまたは全部を [選択された SDK] から削除することもできます。

[選択された SDK]

Smartphone 2003 や Pocket PC 2003 など、選択されている対象プラットフォームの一覧を表示します。

メモ :

デバイス プロジェクトを作成した場合、最初の作成後にいつでも Smartphone 2003 などのプラットフォームを追加できます。ただし、最初の作成後に新しいプラットフォームをプロジェクトに追加した場合、追加されたプラットフォームの [追加ファイル] 構成プロパティには、依存するランタイム DLL は追加されません。

参照

関連項目

MFC スマート デバイス アプリケーション ウィザード

[プラットフォーム] (MFC スマート デバイス DLL ウィザード)

MFC スマート デバイス DLL ウィザードの [プラットフォーム] ページでは、新しい MFC スマート デバイス DLL プロジェクトでサポートされるプラットフォームと命令セットを指定します。

[プラットフォーム] ウィンドウ

プロジェクトのターゲット プラットフォームを指定します。使用できるプラットフォームは、Visual Studio 2005 と共にインストールされるか、別個の Windows Mobile SDK または Windows CE SDK と共にインストールされます。

[命令セットとデバイス パネル]

[プラットフォーム] ウィンドウで選択したプラットフォームでサポートされている命令セットおよびデバイスの一覧を表示します。

[命令セットとデバイス パネル]

[プラットフォーム] ウィンドウで選択したプラットフォームでサポートされている命令セットおよびデバイスの一覧を表示します。

[インストール済み SDK]

Smartphone 2003 や Pocket PC 2003 など、使用できるその他のプラットフォームの一覧を表示します。ボタンを使用して、1 つまたは全部を [選択された SDK] に追加できます。ボタンを使用して、1 つまたは全部を [選択された SDK] から削除することもできます。

[選択された SDK]

Smartphone 2003 や Pocket PC 2003 など、選択されている対象プラットフォームの一覧を表示します。

メモ :

プロジェクトを作成した場合、デバイス プロジェクトの作成後にアプリケーション ウィザードを使用していつでも Smartphone 2003 などのプラットフォームを追加できます。ただし、最初の作成後に新しいプラットフォームをプロジェクトに追加した場合、新しいプラットフォームの Additional Files 構成プロパティには、依存するランタイム DLL は追加されません。

参照

関連項目

MFC スマート デバイス DLL ウィザード

[プラットフォーム] (Win32 スマート デバイス プロジェクト ウィザード)

Windows CE アプリケーション ウィザードの [プラットフォーム] ページでは、新しい Windows CE プロジェクトでサポートされるプラットフォームと命令セットを指定します。

[プラットフォーム] ウィンドウ

プロジェクトのターゲット プラットフォームを指定します。使用できるプラットフォームは、Visual Studio 2005 と共にインストールされるか、別個の Windows Mobile SDK または Windows CE SDK と共にインストールされます。

[命令セットとデバイス パネル]

[プラットフォーム] ウィンドウで選択したプラットフォームでサポートされている命令セットおよびデバイスの一覧を表示します。

[インストール済み SDK]

Smartphone 2003 や Pocket PC 2003 など、使用できるその他のプラットフォームの一覧を表示します。ボタンを使用して、1 つまたは全部を [選択された SDK] に追加できます。ボタンを使用して、1 つまたは全部を [選択された SDK] から削除することもできます。

[選択された SDK]

Smartphone 2003 や Pocket PC 2003 など、選択されている対象プラットフォームの一覧を表示します。

メモ　デバイス プロジェクトを作成した後に、最初の作成後にいつでも Smartphone 2003 などのプラットフォームを追加できます。ただし、最初の作成後に新しいプラットフォームをプロジェクトに追加した場合、追加されたプラットフォームの [追加ファイル] 構成プロパティには、依存するランタイム DLL は追加されません。

参照

関連項目

Win32 スマート デバイス プロジェクト ウィザード

[プロパティ] ウィンドウ (スマート デバイス CAB プロジェクト)

この [プロパティ] ウィンドウを使用して、スマート デバイス CAB プロジェクトのプロパティを指定します。

このウィンドウを開くには、ソリューション エクスプローラでスマート デバイス CAB プロジェクトを選択し、[表示] メニューの [プロパティ ウィンドウ] をクリックします。

[CE Setup DLL]

この .cab ファイルに使用する Windows CE セットアップ DLL を指定します。Setup.dll は、アプリケーションのインストール中および削除中にカスタム操作を実行するオプション ファイルです。詳細については、「デバイス ソリューションをパッケージ化する方法の概要」を参照してください。

[Compress]

この .cab ファイルを圧縮する必要があるかどうかを指定します。

SDK によっては、圧縮された .cab ファイルがサポートされていません。たとえば、Pocket PC 2003 ではサポートされていません。詳細については、SDK のドキュメントを参照してください。

Manufacturer

アプリケーションまたはコンポーネントのメーカー名を指定します。

[NoUninstall]

この .cab ファイルをアンインストールできるかどうかを指定します。このオプションは Compact Framework 2.0 以降で使用されます。

[OSVersionMax]

.cab ファイルをインストールできる OS の最大のバージョン番号を指定します。

[OSVersionMin]

.cab ファイルをインストールできる OS の最小のバージョン番号を指定します。

ProductName

アプリケーションまたはコンポーネントのパブリック名を指定します。これは、アプリケーション名とも呼ばれます。

参照

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[プロパティ] ウィンドウ (スマート デバイス マネージ プロジェクト)

このウィンドウを使用して、スマート デバイス マネージ プロジェクトのプロパティを指定します。この [プロパティ] ウィンドウに表示されても、ここに示されてないプロパティは、デスクトップで共通です。

このウィンドウを開くには、ソリューション エクスプローラでスマート デバイス プロジェクトを選択し、[表示] メニューの [プロパティ ウィンドウ] をクリックします。

[出力ファイル フォルダ]

アプリケーションが配置されるリモート デバイス上のフォルダです。

[プラットフォーム]

アプリケーションがターゲットとするプラットフォームです。

[ターゲット デバイス]

アプリケーションがターゲットとするリモート デバイスまたはエミュレータです。

[Framework バージョン]

このプロジェクトがターゲットとする .NET Compact Framework のバージョンです。

参照

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[証明書の選択] ダイアログ ボックス (デバイス)

デバイス アプリケーションに署名するための証明書を指定できます。また、[証明書の管理] ダイアログ ボックスと証明書のインポート ウィザードへのポータルとしての役割を果たします。

詳細については、「方法 : デバイス プロジェクトで証明書をインポートおよび適用する」を参照してください。

証明書ペイン

使用できる証明書が一覧表示されます。

[証明書の管理]

新しい証明書をインポートできる [証明書の管理] ダイアログ ボックスを開きます。[中間証明機関]、[信頼されたルート証明機関]、および [信頼されたパブリッシャと依頼されていないパブリッシャ] など、他の情報やオプションも使用できます。

参照

概念

セキュリティの概要 (デバイス)

その他の技術情報

デバイス用のユーザー インターフェイス リファレンス

[ユーザー インターフェイスの機能] (MFC スマート デバイス アプリケーション ウィザード)

このページには、アプリケーションの外観と操作性を指定するためのオプションが用意されています。プロジェクトで使用できるユーザー インターフェイス機能は、MFC アプリケーション ウィザードの [アプリケーションの種類] ページで指定したアプリケーション タイプによって異なります。

[コマンド バー]

アプリケーションのコマンド バーの詳細を指定します。

オプション	説明
[メニューのみ]	アプリケーションに基本コマンド バーを追加します。
[メニューとボタン]	ボタンが含まれた基本コマンド バーを追加します。

[ステータス バー]

このオプションは、このリリースではサポートされていません。

テキスト出力用のペインの行を持つコントロール バーを実装します。ダイアログ ベースのアプリケーションには使用できません。

[ダイアログ タイトル]

CDialog ベースのアプリケーションの場合にのみ、このタイトルがダイアログ ボックスのタイトル バーに表示されます。このフィールドを編集するには、[アプリケーションの種類] ページの [アプリケーションの種類] で、[ダイアログ ベース] が選択されている必要があります。詳細については、「[アプリケーションの種類] (MFC スマート デバイス アプリケーション ウィザード)」を参照してください。

参照

関連項目

MFC スマート デバイス アプリケーション ウィザード

Win32 スマート デバイス プロジェクト ウィザード

Win32 スマート デバイス プロジェクト ウィザードを使用すると、4 種類のプロジェクトのうちいずれかを作成し、Windows CE オペレーティング システムで実行できます。いずれの場合でも、開くプロジェクトの種類に応じて追加のオプションを指定できます。

[概要]

ウィザードのこのページでは、作成する Windows CE プロジェクトの現在のプロジェクト設定が表示されます。既定では、プロジェクトに次のオプションが設定されています。

- プロジェクトの既定のターゲット プラットフォームは、プラットフォームの一覧の最初のプラットフォームである。既定のインストールでの既定のプラットフォームは Pocket PC 2003 ですが、Windows CE 5.0 SDK をインストールおよびアンインストールすると、新しいアプリケーションの既定のターゲットを変更できます。
- プロジェクトが Windows アプリケーションである。Windows アプリケーションは、**WinMain** 関数が記述された .cpp ファイルを含むプロジェクトとして定義されます。
- プロジェクトが空ではない。
- プロジェクトにエクスポート シンボルが含まれない。
- プロジェクトではプリコンパイル済みヘッダーを使用しない。
- プロジェクトでは MFC または ATL がサポートされていない。

既定のプラットフォームを変更するには、ウィザードの左の列にある [プラットフォーム] をクリックし、必要な変更を加えます。

他の既定値を変更するには、ウィザードの左の列にある [アプリケーションの設定] をクリックし、必要な変更を加えます。

新しいプロジェクトを作成した後に、コンパイラが **_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA** の定義について警告を発した場合は、メイン ヘッダー ファイル内にこのフラグを定義する必要があります。

```
#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA
```

これは特に、Windows Mobile 内の Web サービスを使用して Windows Mobile プラットフォーム上で COM オブジェクトを作成したり、ATL COM オブジェクトを作成したりするシナリオに当てはまります。

参照

その他の技術情報

Visual C++ デバイス プロジェクトの作成と移植

デバイス対応の Visual Basic の言語リファレンス

デバイス アプリケーション開発者が使用できる Visual Basic の言語プログラミング要素は、.NET Compact Framework をターゲットとした場合と少し異なります。Visual Basic リファレンス には、デバイス アプリケーション開発でサポートされないそれらの要素が記載されています。詳細については、「.NET Compact Framework 開発デスクトップとの違い」を参照してください。

新機能

Visual Studio 2005 のデバイス アプリケーション開発者が使用できる新しいプログラミング要素を次に示します。

- AudioPlayMode 列挙型
- SByte 型 (Visual Basic)、UInteger データ型、ULong データ型 (Visual Basic)、および UShort 型 (Visual Basic)
- TextFieldParser オブジェクト
- IsNot 演算子

サポートされる要素

次のプログラミング要素がサポートされます。

- Visual Basic のすべてのデータ型、ディレクティブ、メソッド、オブジェクト、演算子、およびステートメント
- サポートされない要素として次のセクションに示されていない属性、定数と列挙型、関数、キーワード、およびプロパティ

サポートされない要素

.NET Compact Framework を対象とする場合、Visual Basic の言語プログラミング要素の一部は使用できません。次の段落では、スマート デバイス アプリケーションの開発でサポートされないプログラミング要素の概要を示します。

属性

VBFixedStringAttribute クラス

定数と列挙型

FileAttribute 列挙型、CallType 列挙型、VbStrConv 列挙型、および VariantType 列挙型。

関数

AppActivate 関数, Beep 関数, CallByName 関数, ChDir 関数, ChDrive 関数, CreateObject 関数 (Visual Basic), CurDir 関数, DeleteSetting 関数, Dir 関数, Environ 関数, EOF 関数, FileAttr 関数, FileClose 関数, FileCopy 関数, FileDateTime 関数, FileGet 関数, FileGetObject 関数, FileLen 関数, FileOpen 関数, FilePut 関数, FilePutObject 関数, FileWidth 関数, FreeFile 関数, GetAllSettings 関数, GetAttr 関数, GetObject 関数 (Visual Basic), GetSetting 関数, Input 関数, InputString 関数, Kill 関数, LineInput 関数, Loc 関数, Lock 関数、Unlock 関数, LOF 関数, MkDir 関数, Print 関数、PrintLine 関数, Rename 関数, Reset 関数, RmDir 関数, SaveSetting 関数, Seek 関数, SetAttr 関数, Shell 関数, SPC 関数, StrConv 関数, TAB 関数, Lock 関数、Unlock 関数, VarType 関数 (Visual Basic), Write 関数、WriteLine 関数

キーワード

Ansi、Auto、End (Visual Basic)、および Unicode (Visual Basic) キーワード

プロパティ

LastDllError プロパティ (Err オブジェクト)、ScriptEngine プロパティ、ScriptEngineBuildVersion プロパティ、ScriptEngineMajorVersion プロパティ、および ScriptEngineMinorVersion プロパティ プロパティ

部分的にサポートされる要素

Today プロパティ、TimeOfDay プロパティ、DateString プロパティ、および TimeString プロパティ プロパティは、日付と時刻を取得するために使用できますが、設定はできません。

参照

概念

.NET Compact Framework 開発デスクトップとの違い

その他の技術情報

.NET Compact Framework を使用したデバイスのプログラミング
Visual Basic リファレンス
リファレンス (デバイス)

エラー メッセージ (デバイス)

一部のエラー メッセージはスマート デバイス開発にのみ適用されます。次のようなものがあります。

このセクションの内容

参照

その他の技術情報

リファレンス (デバイス)

<名前> への参照を追加できませんでした。

読み込もうとしているプロジェクトが、Visual Studio のインストールでサポートするように構成されていないデバイス プラットフォームを対象にしています。

このエラーを解決するには

- 読み込むプロジェクトをサポートするプラットフォームをインストールします。

  たとえば、Windows Mobile 5.0 プロジェクトを読み込もうとしていて、Windows Mobile 5.0 SDK がインストールされていない場合は、Windows Mobile 5.0 SDK をインストールします。

参照

その他の技術情報

エラー メッセージ (デバイス)

スキンを読み込むことができない場合 (デバイス)

次のエラーは、[フォーム ファクタのプロパティ] ダイアログ ボックスの [スキン] ボックスでファイルを選択したときに発生することがあります。

メッセージ	一般的な原因
スキンを読み込めません。XML を開くことができません。	ファイル名またはパスが誤っている場合、XML ファイルではない場合、別のプロセスでファイルがロックされている場合、ユーザーの権限が十分ではない場合。
スキンを読み込めません。XML ファイルによって参照されたイメージ ファイルを開くことができません。	上記のいずれかの場合、イメージが BMP または PNG でない場合 (この 2 種類のイメージ形式のみをサポート)、イメージ ファイルが壊れている場合。
スキンを読み込めません。XML ファイルが有効なスキン スキーマを含んでいません。	有効なスキン スキーマの詳細を確認するには、デバイス エミュレータの [ヘルプ] メニューを参照してください。

このエラーを解決するには

1. [OK] をクリックしてエラー メッセージ ボックスを閉じます。
2. [フォーム ファクタのプロパティ] ダイアログ ボックスで、有効なスキン ファイルを選択します。

参照

関連項目

[フォーム ファクタのプロパティ] ダイアログ ボックス (デバイス)

ユーザーのデータストアから情報を取得中にエラーが発生しました。(デバイス)

このエラーは、データ ストアが破損したときに発生します。データ ストアを再作成するには、次の手順に従って、データ ストアを削除し、ソリューションを開き直します。

注意
このプロセスでは、それまでにデータ ストアに追加したカスタム項目 (既存のデバイスからコピーしたデバイス、エミュレータ構成プロパティなど) をすべて削除します。

データ ストアを削除してソリューションを開き直すには

1. 作業内容を保存し、すべてのリモート ツールと Visual Studio を閉じます。
2. Corecon ディレクトリの内容をすべて削除します。

   このディレクトリは、%USERPROFILE%\Local Settings\Application Data\Microsoft にあります。

   たとえば、`del Documents and Settings\myname\Local Settings\Application Data\Microsoft\Corecon\*.*` などです。
3. Visual Studio を開き直します。

   新しいデータ ストアが作成されます。

参照

その他の技術情報

エラー メッセージ (デバイス)

ジェネリック配置エラー (デバイス)

このエラー メッセージは、配置エラーの性質を特定できる情報がないときに表示されます。ほとんどの配置エラーでは、具体的な情報が生成されます。この汎用的メッセージはめったに表示されません。

参照

その他の技術情報

エラー メッセージ (Visual Studio)

.NET Compact Framework 1.0 をターゲットとするプロジェクトには、このコンピュータでは検出されない .NET Framework のバージョン 1.1 が必要

.NET Compact Framework バージョン 1.0 をターゲットとする Visual Studio 2005 プロジェクトには、(デスクトップの) .NET Framework のバージョン 1.1 が必要です。Smartphone 2003 プロジェクトは、このカテゴリに分類されます。

.NET Framework のバージョン 1.1 は、開発コンピュータでは検出されませんでした。

.NET Framework のバージョン 1.1 をインストールするには

- .NET Framework のバージョン 1.1 をダウンロードし、インストールします。

  方法については、http://www.microsoft.com/japan/msdn/netframework/downloads/howtoget.asp を参照してください。

参照

概念

デバイス機能と必要な開発ツール

その他の技術情報

エラー メッセージ (デバイス)

出力ファイル名が無効です。(デバイス Cab プロジェクト)

このエラーは、Cab プロジェクト内の出力ファイルの名前が、名前付け要件に合致しない場合に発生します。

このエラーを解決するには

- 有効な文字と .cab 拡張子を使用して、出力ファイルの名前を変更します。

参照

その他の技術情報

配置用のデバイス ソリューションのパッケージ化
エラー メッセージ (デバイス)

新しいトランスポートに使用する前に、デバイスをリセットしてください。

この警告は、開発用コンピュータとデバイスとを結ぶ既存の接続のトランスポートを変更したときに発生します (たとえば、デバイス エミュレータのトランスポートを DMA から TCP/IP に変更するなど)。

デバイス側コンポーネントとデスクトップ コンポーネントが確実に同じプロトコルを使用するように、デバイスをリセットしてください。すると、デスクトップが適切なプロトコルを使用して新しい接続を確立します。

デバイス エミュレータをリセットするには

- デバイス エミュレータで、[ファイル] メニューの [リセット] をポイントし、[ソフト] または [ハード] を選択します。

  ソフト リセットでもかまいません。

  または

- デバイス エミュレータ マネージャの [利用可能なエミュレータ] ウィンドウで、アクティブなエミュレータを右クリックし、ショートカット メニューの [リセット] をクリックします。

物理デバイスをリセットするには

- メーカーの指示に従います。

参照

関連項目

[デバイスのプロパティ] ダイアログ ボックス

その他の技術情報

スマート デバイスを開発コンピュータに接続する

ビジュアル継承は現在無効になっています。

このメッセージの全文を次に示します。

"基本コントロールにデバイス特定コントロールがあるため、ビジュアル継承は現在無効になっています。"

このメッセージは、基本コントロールまたは基本フォームにデバイス固有のコントロールやコンポーネントが含まれている場合に、マネージ プロジェクトで表示されることがあります。

このメッセージが表示される状況を次に示します。

- 継承されたフォームまたはユーザー コントロールの親にデバイス固有のコントロールが含まれている場合。この場合、継承されたフォームまたはユーザー コントロールはデザイナに表示されません。
- デバイス固有のコントロールを含むフォームまたはユーザー コントロールが別のフォームまたはユーザー コントロールから継承されている場合。この場合、継承されたフォームまたはユーザー コントロールはデザイナに表示されません。
- プロジェクトで Microsoft.WindowsCE.Forms.dll などのデバイス固有のアセンブリを参照する場合。
- プロジェクト、またはそのプロジェクトが参照するアセンブリでプラットフォーム呼び出しを使用する場合。プラットフォーム呼び出しがデザイン時に実行されないことがわかっている場合は、親フォームまたは親ユーザー コントロールに **DesktopCompatible(true)** 属性を設定することで、安全にビジュアル継承を有効にできます。

参照

その他の技術情報

エラー メッセージ (デバイス)

.NET Compact Framework を使用したデバイスのプログラミング