Visual Studio 2005 Smart Device Development

本文档中的内容已停用，将不再更新且不支持。某些链接可能无效。已停用的内容表示此内容的最近更新版本。

Visual Studio

Visual Studio 是一套完整的工具，用于生成桌面和基于团队的企业级 Web 应用程序。除了生成高性能的桌面应用程序外，还可以使用 Visual Studio 基于组件的强大开发工具和其他技术，简化基于团队的企业级解决方案的设计、开发和部署。

本节内容

Visual Studio 简介

找到有关 Visual Studio 新增功能的更多信息，了解有关 .NET Framework 的更多信息，并查找指向此版本 Visual Studio 入门的指针。

Visual Studio 集成开发环境

找到有关设计、开发、调试、测试、部署和管理用 Visual Studio 创建的应用程序的信息。

基于 Windows 的应用程序、组件和服务

确定生成应用程序和组件时使用的工具和技术，以及可使用 Visual Studio 创建的项。

Visual Studio 中的 .NET Framework 编程

了解在 Visual Basic、Visual C# 和 Visual J# 中开发应用程序时如何使用 .NET Framework。

Visual Basic

了解 Visual Basic 的新增功能，并研究如何使用 Visual Basic 开发应用程序。

Visual C#

了解 Visual C# 的新增功能，并研究如何使用 Visual C# 开发应用程序。

Visual C++

找到有关 Visual C++ 的新增功能的信息，以及发现如何使用 Visual C++ 开发应用程序。

Visual J#

找到有关 Visual J# 的信息，Visual J# 是一种工具，供 Java 语言程序员用于生成在 .NET Framework 上运行的应用程序和服务。

JScript

找到有关 JScript .NET（真正面向对象的脚本语言）的信息。

Visual Web Developer

了解 Visual Web Developer 并研究如何使用 Visual Web Developer 创建 Web 应用程序。

Visual Studio Team System

了解 Visual Studio 2005 Team System，这是一个高效、集成且可扩展的软件开发生命周期工具平台，可以帮助软件团队在整个软件开发过程中提高沟通和协作能力。

Visual Studio Tools for Office

了解如何创建商业应用程序，以利用 Microsoft Office 2003 的强大功能对信息进行收集、分析、操作或呈现。

智能设备开发

了解如何开发在基于 Windows CE 的智能设备（如 Pocket PC 和 Smartphone）上运行的软件。

工具和功能

了解 Crystal Reports、Windows Server 功能编程以及应用程序验证工具。

.NET Framework 示例

定位此版本的 Visual Studio 中提供的最新示例应用程序和示例。

快速入门

了解 .NET Framework SDK 附带的教程。

.NET Framework 词汇表

了解 .NET Framework 中使用的常用术语的定义。

相关章节

Visual Studio SDK

找到有关 Visual Studio 软件开发工具包 (SDK)（提供扩展性选项和工具包）的信息。

智能设备开发

Visual Studio 2005 为开发在基于 Windows CE 的智能设备(如 Pocket PC 和 Smartphone)上运行的软件提供丰富的集成支持。您可以使用 Visual C# 或 Visual Basic 来编写在 .NET Compact Framework 上运行的托管应用程序，或可以使用 Visual C++ 来编写本机应用程序。无论选择何种语言，您都将使用开发 PC 程序时所使用的相同代码编辑器、设计器和调试器界面。直接从可用于您选择的语言的智能设备项目模板中选择一个模板，然后开始编码。

Visual Studio 还提供仿真程序，让您可以在开发计算机上运行和调试您的代码；Visual Studio 也提供工具，简化将应用程序及其资源打包到 CAB 文件以便部署到最终用户设备的过程。

有关智能设备项目的最新信息，请访问"Mobile Developer Center"(移动开发人员中心)。

本节内容

智能设备项目入门

提供针对设备应用程序开发问题的概述信息，包括 Visual Studio 2005 的新增功能、Visual Studio for Devices 与其他 Windows Mobile SDK 和工具有什么关系，以及如何设置您的 PC 以便进行设备软件开发。

智能设备开发中的设计注意事项

提供有关选择项目类型、选择开发语言，以及自定义仿真程序外观的信息。

将智能设备连接到开发计算机上

描述连接方法和选项。

使用 .NET Compact Framework 进行设备编程

解释在使用 Visual C# 或 Visual Basic 和 .NET Compact Framework 来开发智能设备软件时的通用过程。

使用 Visual C++ 进行设备编程

解释在使用 Visual C++ 来开发本机设备应用程序时的通用过程。

调试设备项目

解释与桌面调试的区别，并提供对本机代码和托管代码组成的调试解决方案的说明。

打包设备解决方案以便进行部署

提供将您所开发的设备应用程序打包并将其传输到一台或多台目标设备的说明。

设备项目中的安全性

描述如何使用安全证书和供应设备来对您的文件签名。

示例和演练(智能设备项目)

提供完整的项目来阐释用于解决设备编程问题的语法、结构和技术。

参考(设备)

包括设备的 ATL 和 MFC 的参考主题、设备项目的用户界面参考及错误信息等。

相关章节

.NET Compact Framework

介绍设备应用程序的编程方法。.NET Compact Framework 使设备具有了 .NET Framework 的功能。将 .NET Compact Framework 与 .NET Framework 进行比较，描述关键组件，阐释常见编程任务并列出支持的类。

Visual Studio 简介

描述 Visual Studio 中的新增功能。

Visual Studio 集成开发环境

提供有关设计、开发、调试、测试和管理用 Visual Studio 创建的应用程序的信息。

智能设备项目入门

Visual Studio 2005 包括面向 Pocket PC、Smartphone 以及其他基于 Windows CE .NET 的平台开发应用程序所需的工具和框架。如果没有智能设备，可以使用仿真技术在不离开 Visual Studio 集成开发环境的情况下创建和测试智能设备应用程序。

Visual Studio 2005 支持使用 Visual Basic .NET、Visual C# 和 Visual C++ 语言开发智能设备应用程序。

本节内容

智能设备项目中的新增功能

介绍面向智能设备项目的 Visual Studio 2005 的新功能。

应用程序开发概述（设备）

讨论基于 Windows CE 的解决方案、小型设备的设计问题、现有解决方案的移植、编程语言的选择、与桌面开发的比较以及术语问题。

设备功能和所需的开发工具

介绍智能设备的各种版本，以及为每个版本开发所需的工具。

智能设备项目的硬件和软件要求

列出在开发阶段对于生成设备应用程序、连接到设备以及在设备上运行应用程序提出的要求。

设备项目的远程工具

介绍可用于智能设备项目的特殊工具。

如何：优化智能设备开发帮助

描述如何以最佳方式使用设备项目的帮助，包括筛选器的使用。

如何：在 Visual Studio 中启动设备仿真程序

介绍如何使用设备仿真程序来替代物理设备。

更新由以前的工具创建的项目

描述如何移植用早期版本和工具开发的项目。

如何实现 - 智能设备开发

提供有关智能设备常见问题的帮助链接。

请参见

其他资源

智能设备开发

Mobile Developer Center（移动开发人员中心）

智能设备可编程性

智能设备项目中的新增功能

本主题已针对 Visual Studio 2005 SP1 进行了更新。

Visual Studio 2005 中包含下列新增或扩展的功能。

SP1 中的新增功能

此部分已针对 Visual Studio 2005 SP1 进行了更新。

eMbedded Visual C++ 到 Visual Studio 2005 升级向导

eMbedded Visual C++ 项目升级向导已得到改进。

设备支持的桌面 MFC 类的列表

设备 MFC 库中添加了 15 个 MFC 类。eMbedded Visual C++ 中曾包含下列类，但 Visual Studio 2005 中不包含这些类。

CBitmapButton Class

CDialogBar Class

CEditView Class

CFindReplaceDialog Class

CHttpConnection Class

CHttpFile Class

CInternetConnection Class

CInternetException Class

CInternetFile Class

CInternetSession Class

COleSafeArray Class

CReBar Class

CReBarCtrl Class

CRecentFileList Class

CSplitterWnd Class

准备 SQL Server Compact Edition

Microsoft SQL Server 2005 Compact Edition 替换 SQL Server 2005 Mobile Edition。Visual Studio IDE 中的对话框会显示此更改。

数据访问概述(托管设备项目)

创建 SQL Server Compact Edition 数据库，更改架构以及执行其他数据库管理任务，所有这些都可以在 Visual Studio IDE 中进行。将业务对象、SQL Server 数据库、SQL Server Compact Edition 数据库或 Web 服务用作数据源。

Windows CE 6.0 发布于 2006 年 11 月 1 日。

支持 Windows CE 6.0 应用程序开发。

Visual Studio 2005 中的新增功能

Visual C++ 设备应用程序开发中的新增功能

使用 Microsoft 基础类、活动模板库 (ATL) 和 Windows CE 本机 API 开发智能设备应用程序。迁移 eMbedded Visual C++ 4.0 项目。

托管设备项目中的新增内容

用于托管开发的新增功能包括集成的 Smartphone 支持、新增控件、锚定和停靠等等。

如何:在 Visual Studio 中启动设备仿真程序

您可以使用为了运行针对 ARM 处理器编译的代码而从头创建的全新设备仿真程序，来运行、测试和调试运行时图像。

数据访问概述（托管设备项目）

创建 SQL Mobile 数据库，更改架构以及执行其他数据库管理任务，所有这些都可以在 Visual Studio IDE 中进行。将业务对象、SQL Server 数据库、SQL Mobile 数据库或 Web 服务用作数据源。

设备解决方案打包概述

将应用程序打包，以便使用智能设备 CAB 项目以及文件系统编辑器和注册表编辑器进行部署。

在设备项目中切换平台

使单个源以 Pocket PC 和 Smartphone 等多个平台为目标。

设备项目的远程工具

在基于 Windows CE 的设备上远程执行编程和调试任务。

安全概述（设备）

使用适当的证书和供应设备对文件进行签名。

连接方法选择

通过 USB、以太网、802.11b/g、蓝牙、串行端口或红外端口将开发计算机连接到设备。

Mobile Developer Center（移动开发人员中心）网站

访问智能设备项目信息的中心位置，其中包括入门指南、代码示例、常见问题、培训机会以及其他资源。

相关章节

Visual Studio 2005 中的新增功能

.NET Compact Framework 2.0 中的新增功能

新增功能（《SQL Server 联机丛书》）

应用程序开发概述(设备)

Visual Studio 支持两种为设备开发应用程序的方法。

- 可以开发在 Web 服务器上运行的移动 Web 应用程序，这些应用程序可以用不同的格式提供，以支持各种配备了浏览器的移动设备。
- 可以开发在设备本身上运行的、基于 Windows CE 和 Windows Mobile 的富客户端应用程序。这后一种方法就是所说的“智能”设备应用程序开发。

智能设备解决方案和 Windows CE

若要更好地理解 Windows CE、Pocket PC、Smartphone 和 Windows Mobile™ 软件之间的关系，请参见“Microsoft Mobile Developer Center”(Microsoft 移动开发人员中心)上的“Mobile Developer Center Editor's Note”(移动开发人员中心编辑器说明)。

版本兼容性

若要确定在开发设备应用程序时要一起使用哪些版本的工具和技术，请参见“Introduction to Development Tools for Windows Mobile-based Pocket PCs and Smartphones”(用于基于 Windows Mobile 的 Pocket PC 和 Smartphone 的开发工具简介)。在 Visual Studio 2005 托管项目中，除非另行说明，否则所有平台都以 .NET Compact Framework 2.0 版为目标。例如，在“新建项目”对话框中，如果智能设备模板针对的是 .NET Compact Framework 1.0 版本，则这些模板以“(1.0)”进行标记。

设计准则

设备应用程序的设计决定着用户完成任务的难易程度、速度和效率。通过对应用程序进行优化，使其充分利用不同设备的功能，您可以创建可用性、一致性、响应能力和可访问性更强的应用程序，使之用起来得心应手。有关特定界面功能的详细设计准则，请参见设备的软件开发工具包 (SDK)。

设备仿真程序

设备仿真程序是专为 Visual Studio 2005 设备项目设计的。它运行为 ARM 指令集编译的应用程序，并作为用户模式进程运行。Visual Studio 现在提供直接内存访问 (DMA) 传输以与仿真程序进行通信。DMA 传输胜过传统 TCP/IP 传输，该类型的传输速度快并且不依赖于网络连接或其他外部因素，并能够提供确定的连接和断开连接。

Visual Studio 2005 包含用于 Pocket PC 2003 SE、Pocket PC 2003 SE Square、Pocket PC 2003 SE Square VGA、Pocket PC 2003 VGA、Smartphone 2003 SE 和 Smartphone 2003 SE QVGA 的仿真程序映像。

在仿真程序菜单栏上单击“帮助”，可查看支持该仿真程序的所有帮助主题。

若要打开仿真程序，请单击“工具”，单击“连接到设备”，选择要打开的仿真程序，然后单击“连接”。

注意

设备仿真程序支持 Direct3D 和 DirectPlay 库。但是，仿真程序不支持任何形式的硬件加速。运行在仿真程序上的 Direct3D 和 DirectPlay 应用程序的性能将无法准确反映运行在实际硬件上的应用程序的性能。此外，实际硬件可能支持也可能不支持硬件加速。强烈建议您在实际出厂设备上测试 Direct3D 和 DirectPlay 应用程序。

安全性

设备应用程序的远程连接方面造成了其他安全性问题。有关更多信息，请参见设备项目中的安全性、.NET Compact Framework 中的安全和本机代码和 .NET Framework 代码的安全性。

移植现有解决方案

有关移植和迁移的提示，请参见下面的内容：

- 创建和开发托管设备项目
- eMbedded Visual C++ 到 Visual Studio 2005 升级向导

设备与桌面

开发设备应用程序时使用的 Visual Studio 环境与开发桌面应用程序时所使用的相同，但面向具体设备时会出现某些显著差异。例如：

- Visual Studio 环境为设备连接和设备调试提供了更多的工具。
- 创建项目时，除选择项目的类型和模板外，还必须选择要在哪个设备上运行和调试应用程序。该设备可以是连接到开发计算机的物理设备、网络上的设备或运行在开发计算机上的设备模拟器。
- 类的数量和成员都与开发桌面应用程序时使用的有所不同。在使用 .NET Compact Framework 的托管对象中，可用于设备的类更少，且不同平台上的类补集通常不同。本机项目也是如此，在这些项目中，只有 Windows API、MFC 类或 ATL 组件的子集可用。您可以通过查看文档、使用 IntelliSense 或在项目处于活动状态时使用 Visual Studio 对象浏览器来确定哪些类可用。
- 与桌面应用程序一样，可以使用平台调用访问本机代码。.NET Compact Framework 对 COM interop 提供有限的支持。它不支持在托管代码中创建 COM 对象或与 ActiveX 控件的交互。
- 某些语言项可能不同；例如，并非所有用于桌面开发的 Visual Basic 关键字都受支持。
- Visual Studio 文档中为桌面项目提供的有些代码段在设备项目中可能会产生生成错误。
- 设备开发存在一些桌面开发不需要考虑的设计注意事项，例如设备的格式参数、电源使用、内存限制及其他细节。

其他资源

有关更多信息，请参见“Mobile Developer Center”(移动开发人员中心)。

请参见

其他资源

智能设备项目入门

设备功能和所需的开发工具

本主题已针对 Visual Studio 2005 SP1 进行了更新。

Visual Studio 2005 支持针对设备(这些设备运行 Windows Mobile 5.0 版、Windows Mobile 2003 及 2003 Second Edition)和基于 Windows CE 的硬件(运行 Windows CE 5.0)进行应用程序开发。

但是，存在许多旧式设备。此情况可能会导致不清楚需要哪些开发工具、何种 .NET Compact Framework 版本及基础 Windows CE 操作系统。

工具比较表

下表提供了有关各种智能设备硬件、硬件功能和开发工具的概要说明。这些列表可能会随时间发生变化。您可以查看 MSDN Library 中的技术文章Introduction to Development Tools for Windows Mobile-based Devices，以获取完整的最新信息。

IDE 功能概述

此表提供有关不同 IDE 的功能的概述。列标题的缩写如下：

- eVT3C = eMbedded Visual C++ 3.0
- eVT3V = eMbedded Visual Basic 3.0
- eVC4 = eMbedded Visual C++ 4.0 及 Service Pack 4.0
- VS2003 = Visual Studio .NET 2003
- VS2005 = Visual Studio 2005

		eVT3C	eVT3V	eVC4	VS2003	VS2005
代码类型	本机代码	X		X		X
	解释型代码		X			
	托管代码				X	X
	服务器端代码				X	X
设备 SDK	Pocket PC 2000 和 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					X

.NET Compact Framework 工具和 OS 支持

此表提供支持 .NET Compact Framework 1.0 和 2.0 版的工具版本及 Windows Mobile 软件版本的概述。

		版本 1.0	版本 2.0
工具	Visual Studio .NET 2003	X	
	Visual Studio 2005	X	X
Windows Mobile 软件版本	Windows Mobile 5.0	ROM 中 (1.0 SP3)	用户可安装
	Windows Mobile 2003 Second Edition	ROM 中 (1.0 SP2)	用户可安装(仅限 Pocket PC)

	Windows Mobile 2003	ROM 中 (1.0 SP1)	用户可安装(仅限 Pocket PC)
	Smartphone 2002		
	Pocket PC 2002	用户可安装	
	Pocket PC 2000	用户可安装	

数据库技术支持

下表已针对 Visual Studio 2005 SP1 进行了更新。

此表提供 Windows Mobile 不同版本支持的数据库技术的概述。

	SQL Server 2005 Compact Edition 或 SQL Server 2005 Mobile Edition	SQL CE 2.0	EDB	CEDB	ADOCE
Windows Mobile 5.0	用户可安装	用户可安装(仅限 Pocket PC)	ROM 中	ROM 中*	不支持用户安装
Windows Mobile 2003 Second Edition	用户可安装(仅限 Pocket PC)	用户可安装(仅限 Pocket PC)	不可用	ROM 中	ROM 中
Windows Mobile 2003	用户可安装(仅限 Pocket PC)	用户可安装(仅限 Pocket PC)	不可用	ROM 中	ROM 中
Smartphone 2002	不可用	不可用	不可用	ROM 中	不可用
Pocket PC 2002	不可用	用户可安装(仅限 Pocket PC)	不可用	ROM 中	ROM 中
Pocket PC 2000	不可用	用户可安装(仅限 Pocket PC)	不可用	ROM 中	ROM 中(大多数设备)

* 在 Windows Mobile 5.0 中，CEDB 在 ROM 中，但已被否决。开发人员应改用 EDB。

说明

- 有关将设备升级到 Windows CE 或 Windows Mobile 的更新版本的信息，请与设备制造商联系。Microsoft 不为最终用户提供特定设备的升级。
- Visual Studio 2005 速成版不包括对智能设备项目的支持。
- 不再支持 eMbedded Visual Basic 工具。eMbedded Visual Basic 运行库不再包含在设备 ROM 中。
- 可以从"Mobile Developer Center"(移动开发人员中心)下载 eMbedded Visual C++ 4.0 和 eMbedded Visual Basic 4.0。
- 如果 ROM 中尚不存在 .NET Compact Framework 1.0 版，可以在 Pocket PC 2000、2002、2003 和 2003 SE 设备的 RAM 中安装它。如果 ROM 中尚不存在 2.0 版，可以在 Pocket PC 2003、Windows CE 5.0 和 Windows Mobile 5.0 的 RAM 或永久存储区中安装它。
- Compact Framework 的当前版本是 2.0，可以从"Mobile Developer Center"(移动开发人员中心)下载以安装到 RAM 中。

请参见

概念

更新由以前的工具创建的项目

其他资源

智能设备项目入门

智能设备项目的硬件和软件要求

下面的内容指定对开发计算机、目标设备以及两者之间的连接的要求。

开发计算机

安装 Visual Studio 2005 时选择“智能设备可编程技术”(默认情况下选择该功能)会增加约 900 MB 的硬盘空间占用量。如果不开发智能设备应用程序，可以通过卸载“智能设备可编程技术”来释放此空间。可以在控制面板中的“添加/删除程序”选项卡上执行卸载过程，方法是：选择您安装的 Visual Studio，单击“更改/删除”，然后按照显示的步骤执行操作。

如果设备项目中使用了仿真程序，则至少还需要 64 MB 的额外 RAM。

设备

目标设备必须支持开发所面向的平台。Visual Studio 2005 提供对 Pocket PC 2003、Smartphone 2003、Windows CE 5.0 及更高平台的支持。

如果 .NET Compact Framework 未安装在 ROM 中，则还需要在设备上为其提供大约 2 MB 的 RAM。

连接

Visual Studio 2005 具有下列连接要求：

硬件

除非您的物理设备具有无线连接功能，并且开发计算机启用了此功能，否则，需要使用设备制造商提供的串行或 USB 电缆将设备连接到开发计算机。您必须根据设备制造商提供的说明设置开发计算机和设备后才能使用此连接。

如果使用仿真程序作为设备，则无需任何其他硬件。

软件

Microsoft ActiveSync 4.0 或更高版本。

请参见

概念

安装和设置要点

其他资源

智能设备项目入门

Visual Studio 版本

设备项目的远程工具

下面列出的远程工具可以在 eMbedded Visual C++ 4.0 中找到，Visual Studio 2005 也附带这些工具以帮助您开发和调试设备应用程序。

若要启动这些独立的工具，请在 Windows 桌面上单击“开始”，指向“所有程序”，再指向“Microsoft Visual Studio 2005”，单击“Visual Studio Remote Tools”，然后从菜单中选择一个远程工具。

要执行的操作	使用
查看和管理目标设备上的文件系统（包括导出和导入）	远程文件查看器
显示有关目标设备上运行的每个进程的堆标识符和标志的信息	远程堆浏览程序
显示有关目标设备上运行的每个进程的信息	远程进程查看器
显示和管理远程设备的注册表	远程注册表编辑器
显示与目标设备上运行的应用程序关联的窗口收到的消息	远程监视程序
以位图 (.bmp) 文件格式从目标设备上捕获屏幕图像	远程屏幕截图程序

请参见

其他资源

智能设备项目入门

如何：优化智能设备开发帮助

如果在安装 Visual Studio 的 MSDN Library 时选择了“最小”或“自定义”，则可能未必有可供智能设备开发人员使用的所有 Visual Studio 2005 帮助文件。具体地说，“最小”安装不包括库的“移动和嵌入式开发”一节。

“移动和嵌入式开发”一节包含的材料有 Pocket PC 和 Smartphone SDK 文档、Windows CE 信息，等等。该节中的主题经常被 Visual Studio 的“智能设备开发”一节中的主题引用。“智能设备开发”一节始终会安装，它重点介绍关于使用 Visual Studio 环境来开发智能设备应用程序的信息。下列过程演示了如何确定所安装的帮助、如何添加“移动和嵌入式开发”帮助，以及如何筛选帮助。

验证是否安装了“移动和嵌入式开发”帮助部分

- 单击 Pocket PC 开发人员指南。

  如果显示此主题，则表示安装了“移动和嵌入式开发”一节。

  如果未发现该主题，您可以通过执行下列步骤来添加“移动和嵌入式开发”一节。

包括“移动和嵌入式开发”主题

1. 在 Windows 控制面板中，单击“添加/删除程序”。
2. 选择“MSDN Library for Visual Studio 2005”，然后单击“更改”。
3. 在“安装向导”的“欢迎使用”页上，单击“下一步”。
4. 在“程序维护”页上单击“修改”。
5. 在“自定义安装”页上，选择“移动和嵌入式开发”。
6. 从下拉按钮中的可用快捷菜单上，选择“此功能以及所有子功能将被安装在本地硬盘上”。
7. 单击“下一步”。
8. 在“准备修改程序”页上，单击“安装”。

筛选帮助主题

Visual Studio 为目录和索引提供智能设备开发筛选器。应用此筛选器时，可见主题会减少为仅包括对于智能设备开发人员而言最基本的文档。更重要的是，如果要仅显示运行库参考主题中与智能设备有关的部分，包括 .NET Compact Framework 以及 MFC 和 ATL 库，则应用该筛选器是一种绝佳的方法。

如果应用语言筛选器，如 Visual C#，则可查看的主题中不包括智能设备开发主题。为此，最好使用“智能设备开发”筛选器，或者根本不使用筛选器。另请注意，如果您已经登录到使用语言开发设置（如 Visual C# 开发设置）的 Visual Studio，则系统会默认指定一种语言筛选器。

设置智能设备开发筛选器

1. 在 Visual Studio 的“帮助”菜单上，单击“目录”或“索引”。
2. 在“筛选依据”框中，选择“智能设备开发”。

更改开发设置

1. 在 Visual Studio 的“工具”菜单上单击“导入和导出设置”。
2. 在向导的“欢迎使用”页上，单击“重置所有设置”，然后单击“下一步”。

   如果不确定应选择哪些选项，请按 F1 打开向导的帮助主题。
3. 在“保存当前设置”页上，选择是否保存当前设置，然后单击“下一步”。
4. 在“选择一个默认设置集合”页上，选择“常规开发设置”，然后单击“完成”。

请参见

任务

如何：使用 .NET Compact Framework 的类库

概念

Visual Studio 的帮助筛选器

其他资源

智能设备项目入门

如何：在 Visual Studio 中启动设备仿真程序

在智能设备项目开发周期的大部分阶段，设备仿真程序都可以充当物理设备的替代品。设备仿真程序支持许多选项，例如，指定 RAM 大小，但并非所有 SDK 都支持所有选项。有关详细信息，请参阅 SDK 文档。

有关更多信息，请启动设备仿真程序，然后单击“帮助”。

使用“连接到设备”对话框启动设备仿真程序

1. 在 Visual Studio 的“工具”菜单上，单击“连接到设备”。
2. 在“连接到设备”对话框中，从“设备”框中选择一个仿真程序，然后单击“连接”。

使用设备仿真程序管理器启动设备仿真程序

1. 在 Visual Studio 的“工具”菜单上，单击“设备仿真程序管理器”。
2. 在“设备仿真程序管理器”窗口中，右击要启动的仿真程序。
3. 在快捷菜单上单击“连接”。

请参见

其他资源

智能设备项目入门

更新由以前的工具创建的项目

Microsoft Visual Studio 2005 为智能设备开发人员提供了强大的开发工具。现在，使用 C# 和 Visual Basic 的程序员可以利用 Visual Studio 2005 提供的改进工具，并且可以使用与创建桌面应用程序相同的开发环境为 Pocket PC、Smartphone 或 Windows CE 设备创建 C++ 项目。

Visual Studio 2005 开发环境的改进包括：

- C# 源代码重构工具。有关更多信息，请参见重构。
- 改进的调试功能。有关更多信息，请参见调试设备项目。
- C# 和 Visual Basic 的代码段。有关更多信息，请参见创建和使用 IntelliSense 代码段。
- 改进的部署工具。有关更多信息，请参见设备解决方案打包概述。
- 改进的智能设备仿真程序。有关更多信息，请参见如何：在 Visual Studio 中启动设备仿真程序。

将项目从 eMbedded Visual C++ 迁移到 Visual Studio 2005

可以使用迁移向导来迁移 eMbedded Visual C++ 项目。有关更多信息，请参见 eMbedded Visual C++ 到 Visual Studio 2005 升级向导。

将项目从 eMbedded Visual Basic 迁移到 Visual Studio 2005

在 eMbedded Visual Basic (eVB) 中创建的项目不会自动转换为 Visual Studio 2005 项目。您必须将现有的源文件和资源文件添加到在 Visual Studio 2005 中创建的新 Visual Basic 智能设备项目中。

注意
Windows Mobile 2003 设备不将 eMbedded Visual Basic 运行库包含在 ROM 中；即使运行库可能已在 RAM 安装的过程中下载到设备，也不支持此配置。

有关使用 Visual Studio 2005 来转换 eVB 项目的更多信息，请参见下列主题：

- 将 eVB 文件控件迁移到 Visual Basic .NET
- 将 eVB 窗体迁移到 Visual Basic .NET

将 Visual Studio .NET 2003 中的托管项目迁移到 Visual Studio 2005

可以将在 Visual Studio .NET 2003 中开发的 Visual C# 和 Visual Basic 智能设备项目导入 Visual Studio 2005。Visual Studio 转换向导会对项目进行任何必要的更改。

将项目从 2002 设备更新到 2003 设备

有关将项目从 Windows Mobile 2000 更新到 Windows Mobile 2003 的更多信息，请参见 Windows Mobile Platform Migration FAQ for Developers（Windows Mobile 平台迁移开发人员常见问题）。

请参见

概念

设备功能和所需的开发工具

其他资源

智能设备项目入门

迁移到 eVC 4.0 环境

如何实现 - 智能设备开发

本主题已针对 Visual Studio 2005 SP1 进行了更新。

Visual Studio 2005 对于开发在基于 Windows CE 和 Windows Mobile 智能设备(如 Pocket PC 和 Smartphone)上运行的软件,提供了大量的集成支持。您可以使用 Visual C# 或 Visual Basic 编写在 .NET Compact Framework 上运行的托管应用程序,也可以通过 Visual C++ 编写本机应用程序。无论选择何种语言,使用的代码编辑器、设计器和调试器界面都与您在为 PC 开发应用程序时使用的相同。只要选择一种可供所选语言使用的智能设备项目模板,然后开始编写代码即可。

Visual Studio 还提供了仿真程序,所以您无需使用物理设备便可在开发计算机上运行并调试代码。

入门(如何实现 - 智能设备)

Windows 窗体应用程序... 启动设备仿真程序... 工具所支持的版本... 选择开发语言... 使用帮助筛选器... 更多...

设备连接(如何实现 - 智能设备)

虚拟 PC 会话... DMA... 蓝牙... 无 ActiveSync... 疑难解答... 更多...

Visual Basic 和 Visual C#(如何实现 - 智能设备)

创建项目... 共享源... 更改平台... 代码段... 更改默认目标... 更多...

Visual C++(如何实现 - 智能设备)

创建 C++ 设备项目... 迁移 eMbedded Visual C++... 添加 SQL Server Mobile Edition 或 SQL Server Compact Edition... 开发多平台解决方案... 创建 MFC ActiveX 宿主... 更多...

调试(如何实现 - 智能设备)

附加到进程... 调试混合解决方案... 更改设备注册表... 更多...

数据(如何实现 - 智能设备)

创建和管理数据库... 向项目添加数据源... 生成查询... 处理主/从关系... 生成结果集或数据集... 更多...

打包(如何实现 - 智能设备)

创建 cab 项目... 创建快捷方式... 编辑设备注册表... 更多...

安全性(如何实现 - 智能设备开发)

导入证书... 查询安全模型... 给文件签名... 提供设备... 更多...

请参见

其他资源

智能设备开发

入门(如何实现 - 智能设备)

此页链接至有关智能设备应用程序开发入门的常见问题的帮助。

智能设备项目中的新增功能

描述 Visual Studio 2005 中有关设备应用程序开发的新功能及增强功能。

设备功能和所需的开发工具

描述智能设备的各种版本，以及支持每个版本所需的工具。

设备项目的远程工具

列出用于设备应用程序开发的特殊工具，并提供有关每个工具的详细帮助主题的链接。

如何:优化智能设备开发帮助

描述如何以最佳方式使用设备项目的帮助，包括筛选器的使用。

如何:在 Visual Studio 中启动设备仿真程序

描述如何打开此对物理设备进行模拟的常用工具。

更新由以前的工具创建的项目

描述如何移植用早期版本和工具开发的项目。

选择开发语言

将 Visual Basic、Visual C# 和 Visual C++ 作为设备项目的开发语言进行比较。

演练:创建用于设备的 Windows 窗体应用程序

提供开发 Windows 窗体应用程序并将其运行于设备仿真程序的分步说明。

请参见

概念

如何实现 - 智能设备开发

Visual Studio 入门

其他资源

Visual Studio 集成开发环境

设备连接(如何实现 - 智能设备)

此页链接到公用设备连接任务的帮助。

如何:从虚拟 PC 会话连接到设备仿真程序

描述在缺少 TCP/IP 的情况下的连接技术。

如何:访问 Smartphone 仿真程序文件系统

描述如何访问 Smartphone 仿真程序的文件系统,该仿真程序没有自己的文件查看器。

如何:使用蓝牙连接

描述如何使用蓝牙进行连接。

如何:使用 IR 进行连接

描述如何使用红外线进行连接。

如何:在不使用 ActiveSync 的情况下连接到 Windows CE 设备

描述在无法使用 ActiveSync 服务时连接到设备所需的步骤。

如何:从仿真程序访问开发计算机文件

描述如何使用共享文件夹从仿真程序访问开发计算机文件。

如何:设置连接选项(设备)

描述在何处可以查找用于设置连接选项的通用对话框。

请参见

任务

连接疑难解答(设备)

概念

如何实现 - 智能设备开发

连接方法选择

Visual Basic 和 Visual C#(如何实现 - 智能设备)

此页链接至有关 .NET Compact Framework 常见开发任务的帮助。

如何:使用 Visual Basic 或 Visual C# 创建设备应用程序

描述如何创建设备应用程序,以及该过程与创建桌面应用程序的不同之处。

如何:跨平台共享源代码(设备)

描述如何使用编译器常量,以跨不同的平台共享相同的源代码。

如何:在设备项目中更改平台

描述在同一项目中如何在平台之间进行前后切换。

如何:将项目升级到更高版本的 .NET Compact Framework

描述如果安装了更高版本的平台,如何升级现有项目的平台。

在设备项目中管理代码段

描述如何使用专属于设备项目的代码段。

如何:在设备项目中验证代码的平台支持

描述如何确保目标平台支持您的代码。

如何:处理 HardwareButton 事件(设备)

描述如何在 Pocket PC 上重写应用程序键。

如何:更改窗体的方向和分辨率(设备)

描述如果方向和分辨率的默认值不正确或缺少,如何更改它们。

如何:更改默认设备(托管项目)

描述如何在项目开发期间更改目标设备。

如何:优化智能设备开发帮助

描述如何使用智能设备帮助筛选器仅显示设备应用程序开发所支持的那些 .NET Framework 元素。

请参见

参考

数据(如何实现 - 智能设备)

概念

如何实现 - 智能设备开发

.NET Compact Framework 帮助主题

其他资源

使用 .NET Compact Framework 进行设备编程

.NET Compact Framework

托管设备项目中的数据

Visual C++（如何实现 - 智能设备）

此页链接到有关常见智能设备 C++ 任务的帮助。

常规

桌面项目的设备支持

提供可帮助您面向包括桌面在内的多个平台的相关主题列表。

在本机设备项目中进行资源编辑

解释用于设备的资源编辑器与用于桌面的资源编辑器之间的相似点和不同点。

如何：向本机设备项目添加数据库

解释如何开发包括 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库在内的 C++ 智能设备应用程序。

如何：指定主项目输出的远程路径

解释如何设置远程路径。

如何：更改默认设备（本机项目）

解释如何更改 C++ 设备项目中的默认目标。

创建/迁移

如何：创建新的 Visual C++ 设备项目

说明如何创建用于设备的 C++ 应用程序，并与创建桌面上运行的 C++ 应用程序进行比较。

如何：创建多平台设备项目 (Visual C++)

说明如何将同一 C++ 开发项目用于多个设备。

从 eVC 移植所带来的已知问题

说明如何将 eMbedded Visual C++ 4.0 项目迁移到 Visual Studio。

MFC

演练：为智能设备创建多平台 MFC 应用程序

重点讨论如何以多个平台为目标。

演练：为智能设备创建 MFC 多平台 ActiveX 控件

重点讨论如何使用设备专用的 MFC 创建 ActiveX 控件。

如何：查找有关设备支持的 MFC 类和方法的帮助

说明如何使用智能设备帮助筛选器仅显示设备项目所支持的那些 MFC 元素。

ATL

演练：为智能设备创建 ATL 多平台 ActiveX 控件

重点讨论如何使用设备专用的 ATL 创建 ActiveX 控件。

如何：查找有关设备支持的 ATL 类和方法的帮助

说明如何使用智能设备帮助筛选器仅显示设备项目所支持的那些 ATL 元素。

请参见

概念

如何实现 - 智能设备开发

其他资源

Visual C++

调试(如何实现 - 智能设备)

此页提供一些帮助链接，这些帮助是有关智能设备的调试任务以及这些任务与桌面项目的类似任务的不同之处。

设备和桌面调试器之间的差异

列出不支持或以有限方式支持的功能，并描述用于设备调试的 cordbg.exe 的其他信息。

如何：附加到托管设备进程

描述如何不使用调试器即附加到已在运行的进程。

如何：更改设备注册表设置

描述如何使用远程注册表编辑器编辑设备的注册表设置。

演练：调试同时包含托管代码和本机代码的解决方案

描述如何调试这些混合的解决方案。

请参见

概念

如何实现 - 智能设备开发

其他资源

使用 Visual Studio 进行调试

数据(如何实现 - 智能设备)

本主题已针对 Visual Studio 2005 SP1 进行了更新。

此页链接到有关常见智能设备数据绑定任务的帮助。

常规

如何:生成 SqlCeResultSet 代码(设备)

描述如何生成结果集而不是数据集。

如何:更改设计时连接字符串(设备)

描述如何在设计时更改 Visual Studio 用以连接到 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库的字符串。

如何:更改运行时连接字符串(设备)

描述如何在运行时更改应用程序用以连接到 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库的字符串。

如何:添加导航按钮(设备)

描述在 .NET Compact Framework 中不受支持的 **DataNavigator** 类的替代类。

如何:将数据更改持久地保存到数据库中(设备)

描述如何将数据集中的更改应用于数据库。

添加数据源

如何:创建数据库(设备)

描述如何在项目内或项目外使用 Visual Studio 环境创建 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库。

如何:向设备项目添加数据库

描述如何将服务器资源管理器中可用的 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库作为 Visual Basic 或 Visual C# 项目的数据源添加。

如何:添加 SQL Server 数据库作为数据源(设备)

描述如何将 SQL Server 数据库作为 Visual Basic 或 Visual C# 项目的数据源添加。

如何:添加业务对象作为数据源(设备)

描述如何将业务对象作为 Visual Basic 或 Visual C# 项目的数据源添加。

如何:添加 Web 服务作为数据源(设备)

描述如何将 Web 服务作为 Visual Basic 或 Visual C# 项目的数据源添加。

管理数据源

如何:管理数据库中的表(设备)

描述如何添加和移除表,以及如何编辑现有表的架构。

如何:管理数据库中的列(设备)

描述如何添加和移除列,以及编辑列的属性。

如何:管理数据库中的索引(设备)

描述如何添加和移除索引,以及如何更改索引的排序顺序属性。

如何:管理数据库密码(设备)

描述如何为新的 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库设置密码,以及如何更改现有数据库的密码。

如何:缩小和修复数据库(设备)

描述如何缩小和修复 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库。

生成查询

如何：创建参数化查询（设备）

描述如何创建参数化查询。

演练：参数化查询应用程序

提供对端对端项目的分步说明，该项目包括生成参数化查询。

处理主/从关系
如何：创建主/从应用程序（设备）

描述如何实现主/从关系。

演练：数据库主/从应用程序

提供对端对端项目的分步说明，该项目包括创建和运行一个主/从应用程序。

查看和编辑数据
如何：预览数据库中的数据（设备）

描述用于查看数据库中数据的一些选项。

如何：为数据应用程序生成摘要视图和编辑视图（设备）

描述如何使用数据窗体查看和编辑数据网格中的单行数据。

请参见
概念
如何实现 - 智能设备开发
其他资源
托管设备项目中的数据
Overview of SQL Server 2005 Mobile Edition
访问数据 (Visual Studio)

打包(如何实现 - 智能设备)

本页链接到有关设备项目的打包任务和设置任务的帮助。

演练:打包智能设备解决方案以便进行部署

提供有关将应用程序及其资源打包的分步说明。

请参见

概念

如何实现 - 智能设备开发

支持将设备应用程序打包的 IDE 功能

设备解决方案打包概述

安全性(如何实现 - 智能设备开发)

此页链接到有关常见智能设备安全任务的帮助。

常规

如何:在设备项目中导入和应用证书

说明如何有效使用"选择证书"对话框对设备项目进行签名。

如何:将 Signtool.exe 作为生成后事件启动(设备)

说明如何在生成后事件已更改原始二进制文件后对项目进行签名。

如何:向设备查询其安全模型

说明如何确定设备证书存储区中已安装了哪些证书。

签名

如何:对 Visual Basic 或 Visual C# 应用程序进行签名(设备)

列出对针对 .NET Compact Framework 编写的应用程序进行签名的步骤。

如何:对 Visual Basic 或 Visual C# 程序集进行签名(设备)

列出对项目程序集进行签名的步骤。

如何:在 Visual C++ 项目中对项目输出进行签名(设备)

列出对 Visual C++ 项目输出进行签名的步骤。

如何:对 CAB 文件进行签名(设备)

列出对设备 CAB 项目进行签名的步骤。

配置

如何:在 Visual Basic 或 Visual C# 项目中提供设备

列出向托管项目的设备存储区中添加数字证书的步骤。

如何:在 Visual C++ 项目中提供设备

列出向本机项目的设备存储区中添加数字证书的步骤。

如何:为设备提供安全模型

说明如何使用 RapiConfig.exe 配置具有安全模型的设备。

请参见

概念

如何实现 - 智能设备开发

其他资源

本机代码和 .NET Framework 代码的安全性

智能设备开发中的设计注意事项

Visual Studio 2005 可为智能设备开发提供三种不同的编程语言，以及多种不同的项目类型。此部分概括介绍了这些项目类型，并就如何选择更合适的项目语言提出了部分建议。

本节内容

选择一个智能设备项目类型

提供 Visual Studio 2005 中可以使用的多种不同智能设备项目以及项目所支持的不同硬件平台的概述。

选择开发语言

提供有关如何选择最佳项目开发语言的建议。

自定义外观（设备）

描述智能设备项目中的外观及如何自定义这些外观。

请参见

其他资源

智能设备开发

智能设备可编程性

Mobile Developer Center（移动开发人员中心）

选择一个智能设备项目类型

Visual Studio 2005 提供以下项目模板用于创建新的智能设备项目。标记有“(1.0)”的模板指示基于 .NET Compact Framework 1.0 版的项目。其余模板表示基于 2.0 版的项目。

Visual C# 和 Visual Basic

模板名	支持的设备	注释
设备应用程序	Pocket PC 2003、Windows CE 5.0	用于创建 .NET Compact Framework 2.0 Windows 窗体应用程序的项目。
控件库	Pocket PC 2003、Windows CE 5.0	用于创建 .NET Compact Framework 2.0 控件的项目。
类库	Pocket PC 2003、Windows CE 5.0	用于创建 .NET Compact Framework 2.0 类库 (DLL) 的项目。
控制台应用程序	Pocket PC 2003、Windows CE 5.0	用于创建 .NET Compact Framework 2.0 非图形应用程序的项目。
空项目	Pocket PC 2003、Windows CE 5.0	用于创建 .NET Compact Framework 2.0 应用程序的空项目。
设备应用程序 (1.0)	Pocket PC 2003、Smartphone 2003	用于创建 .NET Compact Framework 1.0 Windows 窗体应用程序的项目。
类库 (1.0)	Pocket PC 2003、Smartphone 2003	用于创建 .NET Compact Framework 1.0 类库 (DLL) 的项目。
控制台应用程序 (1.0)	Pocket PC 2003、Smartphone 2003	用于创建 .NET Compact Framework 1.0 非图形应用程序的项目。
空项目 (1.0)	Pocket PC 2003、Smartphone 2003	用于创建 .NET Compact Framework 1.0 应用程序的空项目。

Visual C++

模板名	支持的设备	注释
ATL 智能设备项目	Pocket PC 2003、Smartphone 2003 和 Windows CE 5.0	使用活动模板库创建应用程序的项目。
MFC 智能设备项目	Pocket PC 2003、Smartphone 2003 和 Windows CE 5.0	使用 Microsoft 基础类库创建应用程序的项目。
MFC 智能设备 ActiveX 控件	Pocket PC 2003、Smartphone 2003 和 Windows CE 5.0	使用 Microsoft 基础类库创建 ActiveX 控件的项目。
MFC 智能设备 DLL	Pocket PC 2003、Smartphone 2003 和 Windows CE 5.0	使用 Microsoft 基础类库创建动态链接库的项目。
Win32 项目	Pocket PC 2003、Smartphone 2003 和 Windows CE 5.0	用于创建 Win32 应用程序的项目。

其他项目类型

模板名	支持的设备	注释
智能设备 CAB 项目	Pocket PC 2003、Smartphone 2003 和 Windows CE 5.0	创建用于部署智能设备应用程序的 CAB 文件的项目。

请参见

概念

选择开发语言

选择开发语言

在开发部署于智能设备上的应用程序、控件或库时，有三种编程语言可供选择：Visual C#、Visual Basic 和 Visual C++。

Visual C#

C# 是面向对象的现代编程语言。其垃圾回收功能和对 .NET Compact Framework 类的支持使其成为开发可靠、安全的移动应用程序的理想语言。Visual C# for Smart Devices 包括大量用于快速创建图形用户界面 (GUI) 的控件，而 Compact Framework 类则支持 GDI+、XML 和 Web 服务等功能。Visual C# 还可以在 .NET Compact Framework 无法提供支持的情况下调用本机 Windows CE 函数。

有关使用 Visual C# 进行开发以及访问本机 Windows CE 函数的更多信息，请参见：

- Getting Started with Visual Studio .NET and the Microsoft .NET Compact Framework（Visual Studio .NET 和 Microsoft .NET Compact Framework 入门）
- C# 参考
- An Introduction to P/Invoke and Marshaling on the Microsoft .NET Compact Framework（Microsoft .NET Compact Framework 上的 P/Invoke 和封送处理简介）
- Creating a P/Invoke Library（创建 P/Invoke 库）
- Advanced P/Invoke on the Microsoft .NET Compact Framework（Microsoft .NET Compact Framework 上的高级 P/Invoke）

Visual Basic

Visual Basic for Smart Devices 是 Visual Basic 的全面实现，它比以前的开发工具 eMbedded Visual Basic 功能更为强大。Visual Basic 可以大大简化将桌面应用程序移植到移动设备的任务，并且可以快速创建胖客户端 (Rich Client) 应用程序。与 Visual C# 一样，Visual Basic 也使用 .NET Compact Framework。熟悉 Visual Basic 的开发人员能够迅速移植现有应用程序或创建新的应用程序。与 C# 一样，Visual Basic 也可以访问本机 Windows CE 函数。

有关使用 Visual Basic 进行开发的更多信息，请参见：

- Getting Started with Visual Studio .NET and the Microsoft .NET Compact Framework（Visual Studio .NET 和 Microsoft .NET Compact Framework 入门）
- Visual Basic 参考

Visual C++

在性能为关键考虑因素，或者要开发系统级应用程序、设备驱动程序或者“今日”或“主页”屏幕插件时，Visual C++ 是首选智能设备开发语言。Visual C++ 不支持 .NET Compact Framework，但提供了 Win32 API 集的一个子集。用托管 C# 或 Visual Basic 代码编写的应用程序可以通过互操作来访问 DLL 中包含的 C++ 代码。

有关使用 Visual C++ 进行开发的更多信息，请参见：

- C/C++ Languages

请参见

其他资源

智能设备项目入门

自定义外观(设备)

外观是一种图形，它环绕设备仿真程序或 Visual Studio 设计器中应用程序的矩形窗体和视区。在应用程序开发过程中使用外观时，可以更好地看到应用程序在实际设备上的外观效果。可以在 Visual Studio 设计器和设备仿真程序中使用相同的外观。

外观不仅增强了可视性，还可以提供功能(如处理硬件按钮和软键的鼠标事件)。

注意
如果与 Visual Studio 一起安装的外观文件损坏，可以通过修复 Visual Studio 2005 安装来重新安装它们。有关更多信息，请参见如何：注册 Visual Studio。

本节内容

外观技术(设备)

描述外观的 XML 外观定义文件和图形文件要求。

如何：创建外观文件(设备)

描述创建简单外观的步骤。

如何：更改外观的可视特性(设备)

描述可以对外观定义文件做出的修改，从而影响仿真程序的标题栏、视区的位置和尺寸、外观是否显示在框架中或是否透明。

如何：处理鼠标事件(设备)

描述如何使用颜色映射来指定响应鼠标事件的外观区域(又称为作用点)。

如何：公开工具提示(设备)

描述如何为外观作用点提供工具提示。

如何：将外观和设备仿真程序结合使用

描述设备仿真程序用户界面中的可用选项(包括外观选择)、如何将外观与 OS 映像保存在一起以及如何显示或隐藏工具提示。

如何：在 Visual Studio 2005 中使用外观(设备)

描述 Visual Studio 用户界面中的可用选项，包括设置默认值、指定分辨率、启用旋转以及其他属性。

外观定义文件详细信息(设备)

描述外观定义文件中出现的各个元素(必需元素和可选元素)并提供示例值。

外观定义文件示例(设备)

显示一个完整的外观定义文件。

请参见

其他资源

智能设备开发中的设计注意事项

Mobile Developer Center(移动开发人员中心)

外观技术(设备)

外观由最多三个位图 (BMP) 或可移植网络图形 (PNG) 文件和包含可扩展标记语言 (XML) 的单个外观定义文件组成。外观定义文件和关联的 BMP 或 PNG 文件必须在同一个目录中。

- XML 文件描述外观的工作机制以及在何处查找 BMP 或 PNG 文件。XML 文件还根据映射文件中显示的颜色代码来定义按钮操作。有关更多信息，请参见外观定义文件示例(设备)。
- 一个 BMP 或 PNG 文件(被称为正常图形文件)显示仿真程序外观的默认外观。标准图形文件是外观所必需的唯一文件。
- 可选的第二个 BMP 或 PNG 文件(被称为按下图形文件)显示按下所有按钮时仿真程序外观的外形。

第三个可选的 BMP 或 PNG 文件(被称作映射文件)显示外观上每一个按钮的颜色代码。各按钮的颜色代码在映射文件中表示为完全覆盖按钮的单色区域。外观用户不会看到您在映射文件中用作代码的颜色。

请参见

其他资源

自定义外观(设备)

如何：创建外观文件（设备）

您可以创建新的外观或使用相同的技术自定义现有外观。OEM 使用高性能图形工具来设计和开发外观所需的图形文件。尽管如此，您可以使用更简单的工具（甚至 Microsoft 画图）获得令人满意的结果。您可以使用 Microsoft 记事本之类的简单工具来编写外观定义文件（XML 文件）。

若要自定义现有外观，请使用新的名称保存现有外观文件，然后再编辑它们。当您要使用新的外观时，请确保外观的所有文件，即外观定义文件及其附带的图形文件共享同一个目录。

以下步骤描述如何创建新外观。有关更多信息，请参见外观技术（设备）。

创建新外观

1. 创建显示默认外观的位图 (BMP) 或可移植网络图形 (PNG) 文件。
2. 创建显示所有按钮按下时外观的 BMP 或 PNG 文件。
3. 创建显示每一个按钮的区域使用单一色填充的 BMP 或 PNG 文件。

   这些按钮颜色表示作用点并用来处理鼠标事件。如果您希望每一个按钮的外形可以独立于外观上的其他按钮而更改，请使用不同的颜色来填充每一个按钮区域。有关更多信息，请参见如何：处理鼠标事件（设备）。

注意
若要取得最佳可见性，请勿使用白色或黑色填充这些区域。

4. 创建外观定义文件。

   有关更多信息，请参见外观定义文件详细信息（设备）和外观定义文件示例（设备）。

5. 将三个 BMP 或 PNG 文件和 XML 文件保存在一个目录中。

请参见

其他资源

自定义外观（设备）

如何：更改外观的可视特性（设备）

您可以更改外观的一些可视特性。

注意

视区这个术语描述的是外观中的窗口区域，其中显示应用程序的用户界面。

注意

显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

更改设备仿真程序的标题栏中的用语

- 在外观定义文件中，改变 `titleBar` 元素的用语。例如：

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

更改外观中视区的位置

- 在外观定义文件中，更改分配给 x 轴和 y 轴的值。

  例如，默认的 Pocket PC 2003 Second Edition 架构列出 displayPosX="51" 和 displayPosY="47"。当增加 `displayPosX` 值时，视区将向右移动。当增加 `displayPoxY` 值时，视区将向下移动。

更改视区的高度和宽度

- 在外观定义文件中，更改分配给 `displayWidth` 和 `displayHeight` 元素的值。

  例如，默认的 Pocket PC 2003 Second Edition 架构列出 `displayWidth="240"` 和 `displayHeight="320"`。当增加 `displayWidth` 的值时，视区的水平尺寸将向右延伸。当增加 `displayHeight` 的值时，视区的垂直尺寸将向下延伸。

更改视区的颜色深度

- 在外观定义文件中，更改分配给 `displayDepth` 元素的值。

  例如，默认的 Pocket PC 2003 Second Edition 架构列出 `displayDepth="16"`。

  注意

  颜色深度又称为位深度，它表示定义每个像素时可用的位数。智能设备外观的颜色深度通常为 16。

实现背景透明

- 将抬起或正常图形文件的左下角像素设置为透明色。Microsoft 目前为 Smartphone 和 Pocket PC 提供的外观使用颜色值 FEFFFF 来实现这一目的。

  注意

  使用此技术可以使外观没有边框。换言之，设备仿真程序和 Visual Studio 设计器中的图形将由外观的可视元素绑定，而不是显示在背景色可能不同于外观所在窗口的背景色的矩形中。

请参见

其他资源

自定义外观（设备）

如何：处理鼠标事件（设备）

除了可以使用外观提供真实设备的可见副本外，还可以使用外观来处理鼠标事件，使得对真实设备的仿真更加逼真。

通过向外观定义文件中的每个按钮区域分配唯一的颜色 (*mappingColor*)，可以指定在外观的任何按钮上方悬停、单击或按下并保持光标不动时所要发生的事件。用户界面中看不到此颜色。它只用于为设备仿真程序和 Visual Studio 设计器中的事件处理提供唯一的指示器。

例如，如果使用图形工具查看默认安装在 \Program Files\Microsoft Visual Studio 8\SmartDevices\Skins\PocketPC_2003\PocketPC_2003\2052 中的文件 PocketPC_2003_Mask.PNG，就可以看到每个按钮以不同的颜色显示。

处理 onClick 事件

1. 在外观定义文件的按钮标记中，为 `mappingColor` 分配颜色值。

   下面的示例源于 Pocket PC 2003 外观定义文件：

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

2. 将击键分配给 `onClick` 事件。

   有关更多信息，请参见以下将按钮与击键关联的步骤。

   如果单击颜色为 0xF26C4F 的按钮，将处理在该按钮区域中指定的 `onClick` 事件。外观定义文件中指定的击键将传递到引擎。

处理 onPressAndHold 事件

1. 在外观定义文件的按钮标记中，为 `mappingColor` 分配颜色值。

   下面的示例源于 Pocket PC 2003 外观定义文件：

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

2. 将击键分配给 `onPressAndHold` 事件。

   有关更多信息，请参见以下将按钮与击键关联的步骤。

   如果单击颜色为 0xED145B 的任何按钮，将处理在该按钮区域中指定的 `onPressAndHold` 事件。

将按钮与击键关联

- 使用以前示例中的键盘扫描代码或一组预定义常数（如 Key_Down）。

  有关更多信息，请参见 MSDN 联机库中的 Emulator Skin XML Schema（仿真程序外观 XML 架构）。

请参见

其他资源

自定义外观（设备）

如何：公开工具提示（设备）

外观定义文件中的每个按钮标记均可以包括一行工具提示。如果外观上的某个作用点已分配了工具提示，则当鼠标悬停在该点时会弹出工具提示，显示作用点所表示的功能的信息。作用点通常是按钮，例如，“功能键 1”。工具提示显示外观定义文件中的 ToolTip 元素中指定的按钮名称。

注意

显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

在外观上公开工具提示

1. 在外观定义文件的按钮标记中，为 `mappingColor` 分配颜色值。

   下面的示例源于 Pocket PC 2003 外观定义文件：

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

2. 为 `toolTip` 分配一个文本字符串。

   如果鼠标悬停在具有 0xF26C4F 颜色的按钮上，则显示带有文字“功能键 1”的工具提示。

请参见

其他资源

自定义外观（设备）

如何：将外观和设备仿真程序结合使用

Microsoft Device Emulator 提供一个对话框供您选择“外观定义文件”，如果选择从命令行启动仿真程序，它还提供多个外观选项。

可以从“仿真程序属性”对话框中的“显示”选项卡中找到用于选择仿真程序“外观定义文件”的图形界面。

某些选项仅当您从命令行启动设备仿真程序时才可用。有关更多信息，请单击设备仿真程序中的“帮助”。

注意
显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

打开设备仿真程序

1. 在 Visual Studio 的“工具”菜单上单击“连接到设备”。
2. 在“连接到设备”对话框中选择一个仿真程序，再单击“连接”。

从设备仿真程序打开“仿真程序属性”对话框

- 在设备仿真程序的“文件”菜单上单击“配置”，再单击“显示”选项卡。

从 Visual Studio 2005 打开“仿真程序属性”对话框

1. 在 Visual Studio 的“工具”菜单上，单击“选项”。
2. 展开“设备工具”节点，再单击“设备”。
3. 在“设备”框中选择一个仿真程序，再单击“属性”。
4. 在“<EmulatorName> 仿真程序属性”对话框中单击“仿真程序选项”，再单击“显示”选项卡。

将外观应用于设备仿真程序

- 使用“仿真程序属性”对话框中的“显示”选项卡中的“外观”框。

  - 或 -

- 从命令行启动仿真程序时，请使用 **/skin** 开关。

  - 或 -

- 应用先前与仿真程序 OS 映像一起保存的外观，具体方法是使用 **/s** 开关从命令行启动。

将外观文件与仿真程序 OS 映像一起保存

- 在设备仿真程序的“文件”菜单上单击“保存状态并退出”。

  将生成并保存一个保存状态文件（其中包含外观）。有关更多信息，请在仿真程序的“帮助”系统中查找“保存状态文件”。

显示或隐藏工具提示

- 在“仿真程序属性”对话框的“显示”选项卡上选择或清除“启用工具提示”。

注意
仅当“外观定义文件”中指定工具提示时才可以显示工具提示。

请参见

其他资源

自定义外观（设备）

如何：在 Visual Studio 2005 中使用外观（设备）

Visual Studio 2005（速成版除外）提供若干用户界面元素帮助您在设计器中管理外观。通常可以在“外观设置选项”和“外观设置属性”对话框中找到这些功能。

注意

为方便起见，您可以使用 Visual Studio 环境访问仿真程序和 Visual Studio 设计器的外观属性，而不必打开设备仿真程序。有关更多信息，请参见“访问设备仿真程序的外观属性”部分。

注意

显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

打开“外观设置选项”对话框

1. 在 Visual Studio 的“工具”菜单上，单击“选项”。
2. 展开“设备工具”节点，再单击“外观设置”。

打开“外观设置属性”对话框

- 在“外观设置选项”对话框中选择“外观设置”，再单击“属性”。

选择设计器的外观

1. 在“外观设置属性”对话框中，使用“外观”框选择 XML 文件，该文件表示您要在设计器中使用的外观定义文件。

   外观的图形表现形式显示在“外观”框左边的面板中。
2. 单击“确定”返回到“外观设置选项”对话框。
3. 单击“另存为”将此新设置存储在新的名称下。

   新命名的外观设置包含自定义外观。

   注意

   如果您已打开一个项目，您必须关闭该窗体并重新打开，更改才能生效。

将特定外观设置为默认外观

- 在“外观设置选项”对话框中，使用“默认外观设置”框选择您要设为默认值的外观设置。

  您选择的默认值即成为新项目的默认外观设置。

  注意

  只有当您已经打开一个项目时，才能设置此默认值。

启用旋转支持

- 在“外观设置属性”对话框中选择“启用旋转支持”。

设置水平和垂直分辨率

- 在“外观设置属性”对话框中，输入水平和垂直分辨率的值（像素/英寸）。

隐藏或显示外观

- 在“外观设置属性”对话框中清除或选择“显示外观”。

  如果您选择“显示外观”，则因为外观确定了高度、宽度和颜色深度，所以这些属性将灰显且无法更改。

设置视区的高度和宽度

- 在“外观设置属性”对话框中，使用“屏幕宽度”和“屏幕高度”框输入像素数目值。

  如果您选择“显示外观”，则因为外观设定了这些属性，所以这些属性将灰显且无法更改。

设置颜色深度

- 在“外观设置属性”对话框中，使用“颜色深度”框选择每个像素的位数。

  如果您选择了“显示外观”，则因为外观设定了此属性，所以该属性将灰显且无法更改。

隐藏工具提示

- 从外观定义文件中移除工具提示元素。

  注意

  如果外观定义文件已定义了工具提示，则 Visual Studio 在设计器中不提供任何用于隐藏工具提示的用户界面选项。请将此工具提示行为与应用于设备仿真程序的外观的工具提示行为区别开来，在设备仿真程序中可以启用或禁用工具提示。有关更多信息，请参见如何：将外观和设备仿真程序结合使用。

访问设备仿真程序的外观属性

1. 在 Visual Studio 的“工具”菜单上，单击“选项”。
2. 展开“设备工具”节点，再单击“设备”。
3. 在“设备”框中选择一个仿真程序，再单击“属性”。
4. 在“<EmulatorName> 仿真程序属性”对话框中单击“仿真程序选项”，再选择“显示”选项卡。

   此对话框与单击设备仿真程序的“文件”菜单上的“配置”时显示的对话框相同。有关更多信息，请参见如何：将外观和设备仿真程序结合使用。

请参见

其他资源

自定义外观（设备）

外观定义文件详细信息(设备)

下表描述了设备的外观定义文件的示例元素和值。有关更多信息，请参见外观定义文件示例(设备)。

元素

XML 标记和元素	说明
<skin> 标记	封装仿真程序外观的架构。在每个 XML 文件中只能使用一个 <skin> 标记。
<view> 标记	包含仿真程序外观的架构。每个 <skin> 标记只能使用一个 <view> 标记。
titleBar ="My Emulator skin" 标题栏元素	指定仿真程序的窗口的标题。
displayPosX="10" 和 displayPosY="149" 元素	指定一个位置，以在仿真程序外观的窗口中定位包含仿真程序显示的窗口。若要使该显示不可见，请选择屏幕以外的坐标。
displayWidth="272" 和 displayHeight="224" 元素	指定仿真程序显示的宽度和高度。对于宽度，请选择 80 和 1024 之间可被 8 整除的整数。对于高度，请选择 64 和 768 之间的整数。
displayDepth="8" 元素	指定仿真程序显示的颜色深度。对于颜色深度，可选择 8、16 或 32。
normalImage="up.bmp" 元素	指定仿真程序外观的正常图形文件(它是必需的)。正常图形文件指定仿真程序的窗口大小和仿真程序外观的显示。
mappingImage="map.bmp" 元素	指定仿真程序外观的映射文件。映射文件是一个可选文件，它定义按钮在仿真程序外观中占据的区域。
downImage="down.bmp" 元素	指定仿真程序外观的按下图形文件。按下图形文件是一个可选文件，它指定按下按钮时按钮在仿真程序外观中的显示。
<button> 标记	包含仿真程序外观上的按钮的说明。
mappingColor="0x00FF00" 元素	指定映射文件中用于按钮的 RGB 颜色。映射图像中使用这个颜色的所有像素表示您可以在仿真程序外观中单击该按钮的区域。该区域的行为类似于掩码，当您按下按钮时通过它显示按下图形文件。
toolTip="This is my ToolTip." 元素	可选元素。指定将指针移动到按钮上方时要显示的文本。
onClick=" DOWN:Key_LeftShift Key_Z 0x00000015 UP: Key_LeftShift Key_A"	可选元素。指定按下按钮时要传递到引擎的键盘按键。使用与原始键盘扫描代码对应的十六进制值或整数值。

<button toolTip="Soft Key 1" onPressAndHold="0x3B" mappingColor="0xF26C4F" KeyEvent="F1" Comment="Not handled when menu is present."	指定按下仿真程序外观上的按钮时要重复的键盘事件。如果指定 KeyEvent，则设计器生成按钮的代码。如果指定 Comment，则将它作为注释添加到生成的代码中。默认情况下，工具提示用作注释。 此功能支持所有密钥代码(SHUTDOWN 密钥代码除外)。

请参见

其他资源

自定义外观(设备)

外观定义文件示例(设备)

下面的代码是外观定义文件 PocketPC_2003_Skin.xml(用于 Pocket PC 2003 外观并采用纵向格式)。除速成版外，Visual Studio 2005 在默认情况下将这个外观文件和其他外观文件安装在 \Program Files\Microsoft Visual Studio 8\SmartDevices\Skins 中。

可以将相同的外观文件用于设备仿真程序和 Visual Studio 设计器。有关更多信息，请参见外观定义文件详细信息(设备)。

代码

```
<?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>
```

请参见

其他资源

自定义外观(设备)

将智能设备连接到开发计算机上

本节中的主题讨论如何在开发计算机和目标设备(无论是物理设备还是仿真程序)之间建立安全、可靠的连接。

本节内容

连接方法选择

介绍各种连接方法。

如何:设置连接选项(设备)

介绍如何更改默认连接设置。

如何:在不使用 ActiveSync 的情况下连接到 Windows CE 设备

介绍 ActiveSync 不可用于支持连接时进行连接的步骤。

如何:使用蓝牙连接

介绍建立蓝牙连接的步骤。

如何:使用 IR 进行连接

介绍建立 IR 连接的步骤。

如何:从虚拟 PC 会话连接到设备仿真程序

演示如何将开发计算机连接到 VPC 会话中运行的仿真程序。

如何:访问 Smartphone 仿真程序文件系统

介绍访问 Smartphone 仿真程序的文件系统的步骤,该仿真程序没有文件资源管理器。

如何:从仿真程序访问开发计算机文件

介绍如何使用共享文件夹在仿真程序和开发计算机之间移动文件。

连接疑难解答(设备)

介绍干扰正确连接的问题,以及如何解决这些问题。

请参见

其他资源

智能设备开发

Mobile Developer Center(移动开发人员中心)

连接方法选择

在开发过程中，在智能设备和开发计算机之间建立快速、可靠的连接非常重要。尽管几乎可以在开发过程的所有阶段使用智能设备仿真程序，但是在实际硬件上测试应用程序是开发周期中不可或缺的重要组成部分。

有多种连接选项可用，本文稍后将进行总结。最通用的配置是：

- 使用 DMA 传输连接到设备仿真程序。此传输可消除与网络相关的连接问题，通常作为默认传输提供。除非有重要的原因需要使用另一种传输，否则请始终使用设备仿真程序的 DMA 传输。
- 使用 ActiveSync 4.x 和 USB 端口连接到物理设备。

可以从 Visual Studio 工具菜单中访问这些及其他的选项。有关更多信息，请参见如何：设置连接选项（设备）。

ActiveSync 4.x

ActiveSync 4.x 通过使用电缆、底座、蓝牙或红外连接，提供开发计算机与设备之间的安全连接。它还提供一种设备，通过该设备可将所需连接和安全文件自动下载到设备。当将设备插入底座时，ActiveSync 会关闭所有其他网卡，以确保设备只与开发计算机进行通信。ActiveSync 是开发设备应用程序时的标准连结机制。

如果您的设备不支持 ActiveSync，请参见如何：在不使用 ActiveSync 的情况下连接到 Windows CE 设备。

连接选项

Pocket PC、Smartphone 以及其他基于 Windows CE 的硬件提供链接设备和计算机的多种方式。在这一部分中，将讨论各种连接选项及其优缺点。

根据所涉及的硬件设备，可以使用下列一种或多种连接方法。

USB 连接

最简单的连接形式，所有 Pocket PC 和 Smartphone 设备都支持 USB 连接。尽管在速度上不像以太网连接或无线 802.11b/g 连接那样快，但 USB 连接使用简单，是最常用的连接选项。许多设备还将通过 USB 端口供电，可为您带来更多的便利。

有线以太网

如果不使用附加硬件，默认情况下，Pocket PC 和 Smartphone 设备不支持以太网连接。但是，由于此连接标准具有更高的速度，使其成为执行调试和其他数据密集型操作的首选方式。

无线 802.11b/g 网络

Pocket PC 可以使用无线网卡，目前一些型号的 Pocket PC 集成了无线网络功能。无线网络具有与有线以太网连接同样快的速度。

蓝牙

许多 Pocket PC 和 Smartphone 设备都提供蓝牙无线联网功能。完成配对后，智能设备只要位于桌面计算机的信号覆盖范围内，智能设备就可以通过 ActiveSync 与桌面计算机进行连接。蓝牙在速度上不如 802.11b/g 无线网络快，建议不要在调试时使用。

串行连接

如果没有可用的 USB、有线或无线网络选项，则可以采用串行端口将智能设备连接到开发计算机，但此连接方式速度很慢。

红外连接

红外连接无需使用附加缆线，Pocket PC 和 Smartphone 设备的标准配置都带有 IrDA 端口。然而，红外连接需要位于可视范围内才能可靠地工作，即使这样在性能上也无法满足调试要求。但是，IrDA 可以作为向设备复制文件的最后技术手段。

请参见

任务

连接疑难解答（设备）

其他资源

将智能设备连接到开发计算机上

如何：设置连接选项（设备）

Visual Studio 2005 提供用于将开发计算机连接到设备的许多选项。使用“设备属性”对话框可管理这些连接。

Visual Studio 附带默认使用 DMA 传输的仿真程序连接以及默认使用 TCP/IP 的物理设备。

设置连接选项

1. 在 Visual Studio 的“工具”菜单上，单击“选项”。
2. 展开“设备工具”，单击“设备”，选择一个设备或仿真程序，然后单击“属性”。

   使用关联的对话框选择和配置连接。

请参见

参考

“设备属性”对话框

概念

连接方法选择

其他资源

将智能设备连接到开发计算机上

如何：在不使用 ActiveSync 的情况下连接到 Windows CE 设备

当 ActiveSync 不可用时，Visual Studio 2005 不会将所需的连接文件自动复制到设备。使用下面的步骤将这些文件安装到设备上，修改 Visual Studio 连接配置，并建立设备安全。

前两个步骤，即准备设备和 Visual Studio，只需执行一次。最后一组步骤，即设置安全和建立连接，每当要从 Visual Studio 的新实例连接时都必须重复执行。

准备设备进行连接

1. 使用与您的设备的任何连接，并将以下文件复制到设备上的 \Windows\ 文件夹中。默认情况下，这些文件位于开发计算机上的 \Program Files\Common Files\Microsoft Shared\CoreCon\1.0\Target\wce400\<CPU> 中。
   - Clientshutdown.exe
   - ConmanClient2.exe
   - CMaccept.exe
   - eDbgTL.dll
   - TcpConnectionA.dll
2. 从设备的命令提示处，运行 conmanclient2.exe。
3. 确定设备的 IP 地址。

准备 Visual Studio 进行连接

1. 在 Visual Studio 的“工具”菜单上，依次单击“选项”、“设备工具”和“设备”。
2. 选择“Windows CE 设备”，再单击“属性”。
3. 在“传输”框的右侧单击“配置”。
4. 在“配置 TCP/IP 传输”对话框中选择“使用特定 IP 地址”，再键入设备 IP 地址。
5. 关闭对话框。

   可能会出现一个消息框，提示您重置设备。如果出现该消息框，则软重置即可。

设置安全和建立连接

1. 在设备上的命令提示符处，运行 cMaccept.exe。
2. 在三分钟内连接到设备。

   如果您在三分钟内建立初次连接，那么只要您使用同一个 Visual Studio 实例，则可以无限期地继续部署和调试。如果需要从另一个 Visual Studio 实例进行连接，您就需要再次执行这些安全步骤。

安全注意
您可以通过禁用设备安全，来消除 cMaccept 步骤。为此，请使用远程注册表编辑器设置 `HLKM\System\CoreConOverride Security = 1` DWORD 值。禁用安全会向恶意攻击公开您的设备，我们不建议这样，除非您提供了适当的安全保护措施。

请参见

概念

设备项目的远程工具

其他资源

将智能设备连接到开发计算机上

如何：使用蓝牙连接

以下步骤描述了如何使用蓝牙将设备连接到开发计算机。若要完成此过程，必须安装 ActiveSync 4.0 或更高版本。

使用蓝牙进行连接

1. 在开发计算机上，确保已正确插入并安装蓝牙天线。
2. 在开发计算机上的“蓝牙属性”对话框中，将“可发现”更改为“开”。
3. 在设备上，点击蓝牙图标，再点击“打开蓝牙”。
4. 点击“蓝牙管理器”。
5. 在蓝牙窗口菜单上，单击“新建”启动对启用蓝牙的设备的搜索。
6. 选择要使用 ActiveSync 连接的开发计算机。
7. 提示时，请键入一个临时配对码，然后立即在开发计算机上键入同一配对码。

   现在已与您的设备建立蓝牙连接。若要设置 ActiveSync，必须为该蓝牙连接创建一个虚拟 COM 端口。请使用以下步骤。

设置 ActiveSync

1. 在桌面计算机上转入蓝牙配置，并为蓝牙连接添加一个传入 COM 端口。
2. 在 ActiveSync 中打开“连接设置”对话框，然后选择“允许连接下列之一”。
3. 选择蓝牙设备的 COM 端口，再单击“确定”。
4. 在设备上启动 ActiveSync。
5. 在 ActiveSync 菜单上，点击“通过蓝牙连接”。

   现在已建立 ActiveSync 连接。

请参见

其他资源

将智能设备连接到开发计算机上

如何：使用 IR 进行连接

下列步骤描述了如何使用红外端口 (IR) 将设备连接到您的开发计算机。若要完成此过程，必须安装 ActiveSync 4.0 或更高版本。

设置开发计算机

1. 在开发计算机上打开 Microsoft ActiveSync。
2. 在 ActiveSync 的“文件”菜单上单击“连接设置”。
3. 选择“允许连接到下列之一”。
4. 选择“红外端口 (IR)”，并单击“确定”。

设置设备

1. 确保将设备 IR 端口在有效范围内指向开发计算机 IR 端口。
2. 在设备上打开 ActiveSync。
3. 在 ActiveSync“工具”菜单上点击“通过 IR 连接”。

   连接已建立。

请参见

任务

连接疑难解答（设备）

其他资源

将智能设备连接到开发计算机上

如何：从虚拟 PC 会话连接到设备仿真程序

在虚拟 PC (VPC) 会话中无法使用 TCP/IP 连接到设备仿真程序，因为 VPC 不支持虚拟网络交换机驱动程序。仿真程序需要此驱动程序以建立 TCP/IP 连接。请使用下面的步骤来代替。

注意

显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

在虚拟 PC 会话中连接到仿真程序

1. 在虚拟 PC 映像中安装 Microsoft ActiveSync 4.x。
2. 在 ActiveSync 中的“文件”菜单上，单击“连接设置”。
3. 在“连接设置”对话框中，将连接传输更改为“DMA”。
4. 启动仿真程序。
5. 在 Visual Studio 中的“工具”菜单上，单击“设备仿真程序管理器”。
6. 在“可用的仿真程序”框中，右击所需的仿真程序，然后在快捷菜单中单击“插入底座”。
7. 按照 ActiveSync 向导中的说明建立业务合作关系。

请参见

任务

如何：在 Visual Studio 中启动设备仿真程序

其他资源

智能设备项目入门

将智能设备连接到开发计算机上

如何：访问 Smartphone 仿真程序文件系统

Smartphone 仿真程序没有资源管理器。使用以下技术来访问 Smartphone 文件系统。若要完成此过程，必须安装 ActiveSync 4.0 或更高版本。

在试图将 ActiveSync 与仿真程序一起使用之前，请确保您的桌面计算机上未连接任何设备，并且 ActiveSync 中已禁用 USB 连接。

注意
完成下列步骤以后，如果关闭设备仿真程序管理器或关闭仿真程序，则 ActiveSync 连接也会关闭。

访问 Smartphone 仿真程序文件系统

1. 在 Visual Studio 的“工具”菜单上，单击“设备仿真程序管理器”。
2. 在“可用的仿真程序”框中，选择要访问其文件系统的仿真程序。
3. 在设备仿真程序管理器的“操作”菜单上，单击“连接”。

   在选定的仿真程序旁会出现一个图标，指示已进行了连接。
4. 右击选定的仿真程序，然后单击“插入底座”。

   图标更改，显示仿真程序已插入底座。
5. 打开 ActiveSync。
6. 在 ActiveSync 的“文件”菜单上单击“连接设置”。
7. 选择“允许连接到下列端口之一”复选框。
8. 从端口列表中选择“DMA”，然后单击“确定”。

   ActiveSync 将立即使用仿真程序启动伙伴关系。请按“新建伙伴关系向导”提供的说明操作。

注意
如果伙伴关系未能自动启动，请单击“连接设置”对话框中的“连接”，然后按“建立连接向导”中的提示操作。

9. 完成 ActiveSync 伙伴关系步骤之后，请单击 ActiveSync 工具栏上的“浏览”以访问 Smartphone 仿真程序文件系统。

注意
每当您要通过 ActiveSync 使用与 Visual Studio 连接的仿真程序时，请使用相应平台的设备目标，而不是仿真程序目标。

请参见

其他资源

将智能设备连接到开发计算机上

如何：从仿真程序访问开发计算机文件

下面的步骤演示如何使用共享文件夹从设备仿真程序访问开发计算机上的文件。

注意
Smartphone 仿真程序通常没有文件查看器，因此后面的 Smartphone 测试步骤中使用了远程文件查看器。有关更多信息，请参见远程文件查看器。

设置共享文件夹

1. 在开发计算机上，创建一个在开发计算机和设备仿真程序之间共享的文件夹。
2. 在设备仿真程序的“文件”菜单上单击“配置”。
3. 在“常规”选项卡的“共享文件夹”框中，键入或定位到开发计算机上的共享文件夹。
4. 单击“确定”。

测试 Pocket PC 仿真程序上的共享文件夹

1. 在 Pocket PC 仿真程序中打开“资源管理器”。
2. 点击并选择“我的设备”。

   “存储卡”项就是共享文件夹。

测试 Smartphone 仿真程序上的共享文件夹

1. 在 Windows“开始”菜单上指向“所有程序”，指向“Microsoft Visual Studio 2005”，再指向“Visual Studio 远程工具”，然后单击“远程文件查看器”。
2. 在“选择 Windows CE 设备”中选择 Smartphone 仿真程序，然后单击“确定”打开“Windows CE 远程文件查看器”窗口。

   “存储卡”项就是共享文件夹。

请参见

其他资源

将智能设备连接到开发计算机上

连接疑难解答(设备)

开发计算机与设备之间的大多数连接问题都是由安全或网络问题引起的。以下段落将帮助您识别并解决一些较常见的连接问题，并提供建立可靠、安全的连接所需的步骤。

连接到设备仿真程序

使用 Visual Studio 2005 提供的、用于连接到设备仿真程序的 DMA 传输。此传输几乎可以消除开发计算机与仿真程序之间的所有连接问题。

要点
只有当您有某些重要的特定原因时才可使用 TCP/IP 传输。若要解决对仿真程序使用 TCP/IP 而引发的问题，请查看以下步骤。有关更多信息，请参见"Mobile Developer Center"(移动开发人员中心)。

未能打开虚拟交换驱动程序

如果试图使用仿真的 NE2000 或 CS8900 卡将设备仿真程序连接到网络，则需要虚拟交换驱动程序。您可以从"Mobile Developer Center"(移动开发人员中心)下载驱动程序。

可以导致打开驱动程序时发生错误的原因有多种，其中包括：

- 缺少驱动程序。
- 开发计算机上的网卡没有安装驱动程序。
- 安装驱动程序的过程中出现问题。
- 驱动程序处于禁用状态。
- 开发计算机没有网卡。

使用以下步骤来诊断确切原因。

诊断导致失败的确切原因

1. 查看"仿真程序属性"对话框的"网络"选项卡。

   如果启用了 NE2000 和/或 CS8900 卡，请验证它们绑定到的网卡是否存在并且已连接。若要打开"仿真程序属性"对话框，请单击仿真程序"文件"菜单上的"配置"。

2. 查看适配器的网络属性以验证"虚拟机网络服务"项是否存在、已启用并且版本正确(即 2.6.465.224 或更高版本)。

3. 如果这些步骤未能解决问题，请重新安装驱动程序。

部署到仿真程序错误

如果开发计算机具有无线网络连接并且您使用的是 TCP 传输，您可能需要执行其他步骤(如安装 Microsoft 环回适配器)。有关更多信息，请参见"Mobile Developer Center"(移动开发人员中心)。

注意
除非您有某些重要的特定原因而要使用 TCP 传输，否则请使用 DMA 传输以避免网络问题。

切换传输后无法调试

您可以更改仿真程序的传输，但仿真程序在您对设备进行软重置之前不会绑定到新的传输。

注意
DMA 传输是设备仿真程序的首选传输。只有当您有某些重要的特定原因需要使用 TCP/IP 传输时才能这么做。

切换传输

1. 在 Visual Studio 的"工具"菜单上，依次单击"选项"、"设备工具"和"设备"。

2. 选择一个仿真程序，再单击"属性"。

3. 在“传输”框中选择一个不同的传输。

   如果要切换到 TCP/IP，请单击“配置”以设置附加选项。

4. 单击“确定”关闭对话框。

在虚拟 PC 会话中运行时无法连接到仿真程序

通过对仿真程序使用 DMA 传输可以避免此连接问题。有关更多信息，请参见如何：从虚拟 PC 会话连接到设备仿真程序。

修复设备仿真程序的安装

指示未能连接到设备仿真程序的错误通常不是安装错误。但是，您可以使用以下步骤来修复设备仿真程序的安装。要做到这一点，您需要原始安装媒体。修复 Visual Studio 2005 安装并不会修复设备仿真程序安装。

修复设备仿真程序安装

1. 定位到原始 Visual Studio 2005 安装媒体上的 wcu\ARM。

   此文件夹的位置（在光盘 1、光盘 2 等光盘中）根据 Visual Studio 的版本而有所不同。

2. 双击“vs_emulator.exe”打开“设备仿真程序设置向导”，然后按照说明进行操作。

附加提示

设备仿真程序独立的“帮助”系统提供附加提示。有关更多信息，请单击设备仿真程序“帮助”菜单并在“目录”或“索引”选项卡中查找“连接问题疑难解答”。

连接到物理设备
设备上缺少正确的证书

出于安全原因，某些设备（包括 Smartphone 2003 及更高版本）要求在设备上安装正确的证书。Visual Studio 2005 中包含了用于日常开发工作的证书以及证书安装工具。

安装所需证书

1. 使用任何可用的连接机制连接到设备。

2. 将 SDKCERTS.cab 从开发计算机复制到设备。

   默认情况下，SDKCERTS.cab 位于 \Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools 下。

3. 在设备上将 sdkcerts.cab 解压缩以安装证书。

缺少预备 Windows CE 5.0 设备

没有 ActiveSync 支持的 Windows CE 5.0 设备需要执行一些准备步骤，然后才能与 Visual Studio 实例建立连接。有关更多信息，请参见如何：在不使用 ActiveSync 的情况下连接到 Windows CE 设备。

部署期间的意外行为

如果开发计算机通过 ActiveSync 连接到一台设备，然后再试图与一台 Windows CE 设备（以其为例）建立 TCP/IP 连接，此时发生连接错误，那么开发计算机将与通过 ActiveSync 连接的设备相连，且不警告 TCP/IP 连接失败。

无线连接

虽然 Visual Studio 2005 支持使用无线技术连接到设备，但是无线技术会带来一些其他会对成功的和可维护的连接造成负面影响的因素。这些因素包括 IR 端口未对准、RF 连接中的信号受阻或衰减等。

请参见
其他资源
将智能设备连接到开发计算机上

使用 .NET Compact Framework 进行设备编程

此节提供有关使用 Visual Basic 或 C# 语言以及 .NET Compact Framework 开发智能设备应用程序的信息。

本节内容

托管设备项目中的新增内容

列出自 Visual Basic .NET 2003 以来的重要更改。

.NET Compact Framework 开发中与台式机的不同之处

描述与开发桌面应用程序之间的区别。

.NET Compact Framework 版本的变体

描述 .NET Compact Framework 的几个版本（包括 Service Pack）。

Pocket PC 2003 控件与 Smartphone 2003 控件之间的区别

提供一个表，其中列出了 Pocket PC 2003 开发支持的控件以及 Smartphone 2003 开发支持的控件。

托管设备项目中的控件和组件

描述设备开发可用的其他控件和组件。

设备项目中的自定义控件和用户控件

描述设备项目可用的功能。

设备的 COM 互操作性

描述如何在托管设备项目中创建 COM Interop 程序集。

创建和开发托管设备项目

提供常见设备开发任务的分步介绍。

托管设备项目中的数据

描述如何为 Visual Studio 2005 中的设备管理数据。

演练：创建简单应用程序

提供有关创建简单的 Visual Basic 或 Visual C# 设备项目的分步介绍。

请参见

任务

演练：创建用于设备的 Windows 窗体应用程序

其他资源

.NET Compact Framework

智能设备开发

.NET Compact Framework

常见问题

Mobile Developer Center（移动开发人员中心）

托管设备项目中的新增内容

下面的列表描述自 Visual Studio .NET 2003 以来，托管设备应用程序开发所添加的增强功能。

注意
虽然可以使用 Visual Studio 2005 针对 .NET Compact Framework 1.0 版和 2.0 版进行编程设计，但在下面列出的功能中，标有星号 (*) 的功能在 1.0 版中不受支持。

新增功能

下列功能可在 Visual Studio 2005 中使用。

功能	详细信息
*在 Windows 窗体上锚定控件	如何：在 Windows 窗体上锚定控件
使用 Visual Studio 安装和部署项目生成 CAB 文件。此过程消除了手动自定义 .inf 文件的需要。	设备解决方案打包概述
筛选不受支持的属性/方法/事件。	功能包括筛选的 IntelliSense、生成验证步骤，以及转换不受支持的控件的能力。
颜色编辑器	显示在设备上的颜色。 “定义颜色”对话框（设备）
自定义控件 *用户控件	完全支持。使用设计和查看类与类型添加自定义设计器属性。
*在 Windows 窗体上停靠控件	如何：在 Windows 窗体上停靠控件
*数据设计支持	托管设备项目中的数据
字体编辑器	当前平台所支持的字体。“字体编辑器”对话框（设备）
外观设置支持，包括方向和分辨率。 *自动缩放	“外观设置属性”对话框（设备）
用代码创建窗体外观	“外观设置属性”对话框（设备）
*工具箱中的新控件和新组件	WebBrowser, Notification, DocumentList, DateTimePicker, MonthCalendar, LinkLabel, Splitter, BindingSource
平台特定的 WYSIWYG	改进的设计器可为每个平台提供准确的外观。
以 .NET Compact Framework 1.0 版为目标	Smartphone 2003 和 Pocket PC 2003 支持
使用相同的源代码更改平台	如何：跨平台共享源代码（设备）

请参见

其他资源

.NET Compact Framework 2.0 中的新增功能
开发环境中的新增功能
使用 .NET Compact Framework 进行设备编程

.NET Compact Framework 开发中与台式机的不同之处

在开始一个设备项目之前，首先要了解使用 .NET Framework 进行桌面应用程序开发和使用 .NET Compact Framework 进行设备应用程序开发之间的主要差别，这一点很重要。

Visual Basic 中的编程元素

当使用 Visual Basic 针对 .NET Compact Framework 进行编程时，可用的编程元素列表（例如函数和关键字）与针对完整版本的 .NET Framework 进行编程时并不相同。用于设备应用程序开发的 Visual Basic 语言参考中总结了这些不同之处，Visual Basic 参考中单独讨论这些元素的主题也对此进行了说明。

使用 My 进行开发

Visual Studio 2005 包含对 My.Resources、My.Forms 和 My.WebServices 的支持。它不包含对 My.Application、My.Computer、My.User 或 My.Settings 的支持。有关更多信息，请参见 My Reference。

文件输入和输出

Visual Basic 提供两个文件输入/输出 (I/O) 选项：

- 标准 .NET Framework System.IO 命名空间。公共语言运行库 (CLR) 中的所有语言都支持这些库。
- 一组 Visual Basic 特定的库，可提供与早期版本的 Visual Basic 的库相似的开发体验。

设备项目仅支持 .NET Framework **System.IO** 命名空间。不支持使用 FileSystem 命名空间的文件 I/O，因为：

- 设备上不存在 **FileSystem** 命名空间的几项常用功能。例如，设备上不存在当前目录或当前驱动器的概念。因此，无法使用 ChDir 和 ChDrive 功能。
- 仅支持 .NET Framework 的 **System.IO** 命名空间可减少 Visual Basic 帮助器库的大小。这样可以释放设备上的宝贵空间。

隐式后期绑定

在 Visual Basic 中，如果将对象赋给声明为 Object 数据类型类型的变量，则该对象被后期绑定。该类型的对象在运行时绑定。可以给对象赋值，并从中检索值。但却不能使用圆点约定指定对象变量的方法或属性。下面这段试图获取对象属性的代码会引发编译器错误：

```
dim a as object = "automobile"

dim i as integer = a.horsepower
```

COM Interop

在按其自身的步调过渡到 NET Framework 时，桌面应用程序开发人员使用 COM 互操作性来利用现有的 COM 对象。设备项目仅支持某些 COM 互操作性方案。有关更多信息，请参见设备的 COM 互操作性。

调试

附加到正在运行的进程的方法与桌面应用程序有所不同。有关更多信息，请参见如何：附加到托管设备进程。

请参见

参考

用于设备应用程序开发的 Visual Basic 语言参考
System.IO Namespace
确定要使用的技术和工具
My Reference

概念

早期绑定和后期绑定
Visual Basic 中的 Me、My、MyBase 和 MyClass
设备的 COM 互操作性

其他资源

使用 Visual Basic 访问文件
Visual Studio 中的 .NET Framework 编程

.NET Compact Framework 版本的变体

在开发用于智能设备的应用程序时，需要考虑到 Compact Framework 安装版本的可能变体，这一点很重要。

.NET Compact Framework 1.0 版

- Compact Framework 的最初版本。

.NET Compact Framework 1.0 版 SP1

- 1.0 版的 bug 修复程序。
- 支持 Smartphone。

.NET Compact Framework 1.0 版 SP2

- 1.0 版 SP1 的 bug 修复程序。
- 性能得到了改进。
- 支持使用横向显示模式的设备。
- 支持自动滚动，自动支持横向模式的对话框。

.NET Compact Framework 1.0 版 SP3

- 1.0 版 SP2 的 bug 修复程序。

.NET Compact Framework 2.0 版

已升级为 .NET Compact Framework 2.0 版的智能设备将具备以下功能：

- 性能得到了改进。
- 支持泛型类。
- 支持 COM Interop。
- 改进了对控件的支持。
- 支持移动设备的 Direct3D 和 DirectDraw。

请参见

其他资源

使用 .NET Compact Framework 进行设备编程
.NET Compact Framework 2.0 中的新增功能
.NET Compact Framework FAQ - .NET Compact Framework 2.0（.NET Compact Framework 常见问题 - .NET Compact Framework 2.0）

Pocket PC 2003 控件与 Smartphone 2003 控件之间的区别

下表对 Windows Mobile Pocket PC 2003 和 Windows Mobile Smartphone 2003 间的控件支持的差异进行了总结。一些控件对 Smartphone 没有意义，因为 Smartphone 不具备某些功能，如没有触摸屏。

控件	Pocket PC 2003	Smartphone 2003
Label	是	是
LinkLabel	是	否
Button	是	否
TextBox	是	是
MainMenu	是	是
CheckBox	是	是
RadioButton	是	否
PictureBox	是	是
Panel	是	是
DataGrid	是	否
BindingSource	是	否
ListBox	是	否
ComboBox	是	是
ListView	是	是
TreeView	是	是
TabControl	是	否
DateTimePicker	是	否
MonthCalendar	是	否
HScrollBar	是	是
VScrollBar	是	是
Timer	是	是
Splitter	是	否
DomainUpDown	是	否

NumericUpDown	是	否
TrackBar	是	否
ProgressBar	是	是
ImageList	是	是
ContextMenu	是	否
ToolBar	是	否
StatusBar	是	否
OpenFileDialog	是	否
SaveFileDialog	是	否
WebBrowser	是	否
InputPanel	是	否
通知	是	否
DocumentList	是	否

可以跨平台共享源代码。有关更多信息，请参见如何：跨平台共享源代码（设备）。

请参见

任务

如何：将项目升级到更高版本的 .NET Compact Framework

其他资源

使用 .NET Compact Framework 进行设备编程

托管设备项目中的控件和组件

托管设备项目有自己的控件和组件集，并在 Visual Studio 工具箱中拥有自己的选项卡。当设备项目处于活动状态时，可以对此"设备"选项卡进行拖放操作，就如同在桌面项目中一样。

有关更多信息，请参见 .NET Compact Framework 中的 Windows 窗体控件

请参见

其他资源

创建和开发托管设备项目

设备项目中的自定义控件和用户控件

.NET Compact Framework 为创建和自定义 Windows 窗体项目的控件提供了强劲的支持。

用户控件

您可以创建用户控件并将它们添加到托管设备项目，其操作与桌面项目的此类操作相同。有关更多信息，请参见如何：创作复合控件。

自定义控件

.NET Compact Framework 提供了许多自定义控件的方法。有关更多信息，请参见自定义控件开发。

请参见

任务

演练：为设备创作用户控件

演练：向用户控件添加简单属性

其他资源

设计时开发 Windows 窗体控件

设计和查看类与类型

使用 .NET Compact Framework 进行设备编程

设备的 COM 互操作性

.NET Compact Framework 支持 COM 对象的"运行库可调用包装"(也称为"互操作程序集")。此功能包括复杂类型的封送处理。设备的 COM Interop 基于桌面实现。同样，组件也必须在桌面上注册。

支持的方案

在 Visual Studio 2005 中，设备项目支持以下方案：

- 可以将现有 COM 组件作为引用添加到托管项目。此操作会创建一个互操作程序集，并自动将程序集作为引用添加。然后便可以像使用任何托管程序集一样使用互操作程序集，同时对象的属性、方法和事件也可以用于 Intellisense 和在对象浏览器中使用。可添加的合法文件类型有 DLL、EXE 和 TLB。
- 可以创建一个本机项目来生成一个 COM 组件，然后再在同一解决方案中创建托管项目来使用该 COM 组件。此过程与适用于桌面项目的过程相同：
    - 设置本机项目以生成 TLB 输出。
    - 编译本机项目以生成 DLL。
    - 在托管项目中，添加对 DLL 的引用。此操作可生成互操作程序集。

不受支持的方案

以下方案在 Visual Studio 2005 中不受支持：

- 从托管项目中引用现有的 ActiveX COM 组件
- 具有非系统子组件的 COM 对象
- 从数据源向导中以业务对象的形式引用 COM 对象。

请参见

任务

演练：Hello World：智能设备的 COM Interop 示例

演练：调试同时包含托管代码和本机代码的解决方案

概念

COM Interop 介绍

运行库可调用包装

封送选定接口

其他资源

.NET Framework 应用程序中的 COM 互操作性

使用 .NET Compact Framework 进行设备编程

.NET Compact Framework 中的互操作性

创建和开发托管设备项目

开发托管设备项目与开发托管桌面项目非常类似。主要的不同之处在于，.NET Compact Framework 是 .NET Framework 全功能版的子集，因此有些方面不受支持。有关更多信息，请参见 .NET Compact Framework。

本节内容

请参见

其他资源

使用 .NET Compact Framework 进行设备编程

Visual Studio 中的 .NET Framework 编程

如何：使用 Visual Basic 或 Visual C# 创建设备应用程序

除了必须要选择一个项目要在其上运行的目标平台（例如 Pocket PC 2003）和 .NET Compact Framework 版本（如 2.0 版）外，为设备创建 Visual Basic 和 Visual C# 托管项目的过程与创建桌面项目的过程大体相同。

Visual Studio 2005 中增强的“新建项目”对话框取代了 Visual Studio .NET 2003 的智能设备应用程序向导。在 Visual Studio 2005 中，您可以通过“新建项目”对话框进行涉及项目类型和模板的所有选择。

注意
对话框中提供的选项和菜单命令会随当前设置（又称配置文件）的不同而有所不同。下面的过程是使用 Visual Basic 开发设置和 Visual C# 开发设置编写的。若要查看或更改设置，请在“工具”菜单上选择“导入和导出设置”。

创建设备项目

1. (Visual Basic) 在 Visual Studio 2005 中的“文件”菜单上，单击“新建项目”。

   - 或 -

   (Visual C#) 在 Visual Studio 2005 的“文件”菜单上，指向“新建”，然后单击“项目”。

2. 在“新建项目”对话框的“项目类型”之下，展开“Visual Basic 项目”或“Visual C# 项目”，展开“智能设备”，再单击所需的平台（例如，“Pocket PC 2003”）。

   如果开始时并未出现您需要的语言，请展开“其他语言”。此显示受开发设置的控制。若要查看或更改设置，请在“工具”菜单上单击“导入和导出设置”。

3. 在“模板”下单击适于执行所需任务的模板，例如，“Pocket PC 2003 类库”。

注意
名称末尾追加有 (1.0) 的模板是针对 .NET Compact Framework 1.0 版而设计的。其他模板针对 2.0 版设计。

4. 在“名称”框中键入项目名称。

   如果出现“位置”框，请确认项目文件的目标存储位置，然后单击“确定”。

向现有解决方案中添加项目

- 在“文件”菜单上指向“添加”，再单击“新建项目”或“现有项目”。

  “新建项目”命令会打开“新建项目”对话框，以便可以创建新的项目。“现有项目”命令会打开“添加现有项目”对话框，以便可以选择现有项目并将其包含在当前的解决方案中。

移植现有项目

- 有关移植在早期版本的 Visual Studio 中创建的项目的步骤，请参见用于管理项目的 Visual Studio 常规文档。有关更多信息，请参见项目和向后兼容性和使用 .NET Framework 的多个版本。

请参见

任务

演练：创建用于设备的 Windows 窗体应用程序

参考

确定要使用的技术和工具

其他资源

.NET Compact Framework 2.0 中的新增功能

管理解决方案、项目和文件

创建和开发托管设备项目

如何：在 Visual Basic 项目中显示“设备”工具栏

如果将“开发设置”设为“Visual Basic 开发设置”，默认情况下不会显示“设备”工具栏。

显示“设备”工具栏

- 在 Visual Studio 的“视图”菜单上指向“工具栏”，再单击“设备”。

请参见

其他资源

创建和开发托管设备项目

如何：在 Visual Basic 项目中显示配置管理器（设备）

如果您选择了“Visual Basic 开发设置”，则默认情况下不会出现“配置管理器”。例如，要从“调试”切换到“发布”配置时，使用“配置管理器”。下面的步骤演示如何显示“配置管理器”。

在 Visual Basic 设备项目中显示配置管理器

1. 在 Visual Studio“工具”菜单上，单击“选项”，展开“项目和解决方案”，再单击“常规”。
2. 单击“显示高级生成配置”。

请参见

其他资源

创建和开发托管设备项目

如何:跨平台共享源代码(设备)

通过使用编译器常数来区分那些依赖于目标平台的代码段,可以跨平台共享源代码。允许的常数有 *PocketPC*、*Smartphone* 和 *WindowsCE*。平台必须针对相同版本的 .NET Compact Framework。

下面的步骤提供了该技术的简单示例。创建一个 Visual Basic Pocket PC 应用程序,添加编译器指令,运行应用程序,关闭应用程序,并将应用程序更改为一个 Smartphone 应用程序。然后运行该 Smartphone 应用程序,查看标题栏文字是否更改。

注意
显示的对话框和菜单命令可能会与帮助中的描述不同,具体取决于您现用的设置或版本。若要更改设置,请在"工具"菜单上选择"导入和导出设置"。有关更多信息,请参见 Visual Studio 设置。

创建和运行 Pocket PC 版本

1. 在 Visual Studio"文件"菜单上,指向"新建",然后单击"项目"。
2. 在"项目类型"窗格中,依次展开"Visual Basic"和"智能设备",然后单击"Pocket PC 2003"。
3. 在"模板"窗格中,单击"设备应用程序(1.0)",然后单击"确定"。

   附加的"(1.0)"指示这是 .NET Compact Framework 1.0 版本的项目。
4. 在设计器中右击窗体,然后在快捷菜单上单击"属性"。
5. 清除窗体的"文本"属性值,也就是说,使其为空。
6. 在"解决方案资源管理器"中右击"Form1.vb",然后在快捷菜单上单击"查看代码"。
7. 展开"Windows 窗体设计器生成代码"区域。
8. 在 Public Sub New() 中的 InitializeComponent() 之后插入下面的代码:

   ```
   #If PocketPC Then
      Me.Text = "PPC2003"
   #Else
      Me.Text = "Smartphone"
   #Endif
   ```
9. 在"调试"菜单上单击"开始调试"。
10. 在"部署 <Projectname>"对话框中单击"Pocket PC 2003 SE 仿真程序",再单击"部署"。

    Pocket PC 应用程序便会在仿真程序中运行,而窗体的标题栏中则显示有"PPC2003"。

创建并运行 Smartphone 版本

1. 关闭仿真程序,但不保存状态。

   如果出现一条消息,该消息指示已丢失连接,则单击"确定"。
2. 在"项目"菜单上单击"更改目标平台"。
3. 在"更改目标平台"对话框的"更改为"框中,选择"Smartphone2003",再单击"确定"。
4. 在提示项目即将被关闭和重新打开的消息框中,单击"是"。

   注意,工具栏上的"目标设备"框中现在显示"Smartphone 2003 SE 仿真程序"。
5. 在"调试"菜单上单击"开始调试"。
6. 在"部署 <Projectname>"对话框中单击"Smartphone 2003 SE 仿真程序",再单击"部署"。

   Smartphone 应用程序便会在仿真程序中运行,而窗体的标题栏中则显示有"Smartphone"。

请参见

任务

如何:在设备项目中更改平台

其他资源

创建和开发托管设备项目

如何:在设备项目中更改平台

在同一项目中，可以在平台之间进行前后切换。例如，如果目标平台为 Pocket PC，只要新平台的目标 .NET Compact Framework 版本与原始目标平台的目标版本相同，就可以切换到以 Windows CE 作为目标平台。

在设备项目中更改目标平台

1. 在“项目”菜单上单击“更改目标平台”。
2. 在“更改为”框中，选择一个不同的平台，然后单击“确定”。

   只有目标 .NET Compact Framework 版本与活动项目相同的平台才可使用。

请参见

参考

“更改目标平台”对话框(设备)

其他资源

创建和开发托管设备项目

如何：将项目升级到更高版本的 .NET Compact Framework

如果开发计算机上安装了更高版本的 .NET Compact Framework，可使用此过程升级现有项目的 .NET Compact Framework 版本。

将项目升级到 .NET Compact Framework 的更高版本

1. 在“项目”菜单上单击“升级项目”。

注意
如果“升级项目”命令没有出现在菜单上，则表示开发计算机上没有安装更高版本的平台，升级过程不可用于当前项目。

2. 按照出现的提示中的说明进行操作。

请参见

其他资源

创建和开发托管设备项目

如何：在设备项目中验证代码的平台支持

Visual Studio 始终验证目标平台是否支持代码，如果不支持，则生成警告。

可以规定将所有警告都视为错误，从而阻止部署不受支持的代码，使生成失败。

例如，下面的 Visual Basic 代码虽然可以编译，但会在 Smartphone 应用程序中生成警告或错误，因为 Smartphone 不支持按钮。

```
Dim btn as System.Windows.Forms.Button
btn = new System.Windows.Forms.Button()
...
btn.Caption = "MyButton"
```

将 Visual Basic 中的所有警告都视为错误

1. 在“解决方案资源管理器”中右击“<Projectname>”，再单击快捷菜单上的“属性”。
2. 在“编译”选项卡上，选择“将所有警告视为错误”。

将 Visual C# 中的所有警告都视为错误

1. 在“解决方案资源管理器”中右击“<Projectname>”，再单击快捷菜单上的“属性”。
2. 在“生成”选项卡上，选择“将警告视为错误”区域中的“全部”。

请参见

其他资源

创建和开发托管设备项目

如何：处理 HardwareButton 事件（设备）

使用“HardwareButton”控件重写 Pocket PC 上的应用程序键。

将 HardwareButton 组件分配给特定的应用程序键

1. 从“工具箱”的“设备组件”选项卡中，将“HardwareButton”组件拖动到 Windows 窗体上或设计器的组件栏中。
2. 在组件栏中右击“HardwareButton”控件，再单击快捷菜单上的“属性”。
3. 将“AssociatedControl”属性设置为窗体（例如，“Form1”）。
4. 将“HardwareKey”属性设置为要重写的键（例如，“ApplicationKey1”）。
5. 单击设计器外观上的按钮（例如，“软键 1”）。

   代码编辑器在 `Form_KeyDown` 事件处理程序中打开。
6. 插入下列代码：

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

   通常使用 `//TODO` 节启动应用程序。

请参见

任务

如何：使用 HardwareButton 组件

参考

HardwareButton

其他资源

创建和开发托管设备项目

如何：更改窗体的方向和分辨率（设备）

已经为利用 Visual Studio 安装的平台设置了默认方向（旋转）属性。如果需要更改属性，或是安装的 SDK 中缺少这些属性或是这些属性不正确，请使用以下步骤。

更改方向

1. 在“工具”菜单上依次单击“选项”、“设备工具”和“外观设置”。
2. 为项目中的窗体选择外观设置（例如，“Pocket PC 2003 竖幅”），再单击“属性”。
3. 选择“在 Windows 窗体设计器中显示外观”和“启用旋转支持”。
4. 单击“确定”，关闭“外观设置属性”对话框，然后再次单击“确定”，关闭“选项”对话框。

   注意

   如果更改选项时设计器处于打开状态，请关闭设计器并重新打开，以使所做的更改生效。

5. 在设计器中右击窗体或外观，再选择“向左旋转”或“向右旋转”，旋转外观。

   设备旋转时窗体是否旋转取决于当前平台和窗体属性设置，如下所示：

   - Smartphone：窗体总是随设备一起旋转。
   - Pocket PC：默认情况下，窗体会随设备一起旋转。若要防止窗体随设备旋转，请将窗体的“WindowState”属性设置为 *Normal*，并将“FormBorderStyle”属性设置为 *None*。
   - Windows CE：默认情况下，窗体不随设备一起旋转。若要使窗体随设备旋转，请将“WindowState”属性设置为 *Maximized*。

更改分辨率

1. 在“工具”菜单上依次单击“选项”、“设备工具”和“外观设置”。
2. 为项目中的窗体选择外观设置（例如，“Pocket PC 2003 竖幅”），再单击“属性”。
3. 在“外观设置属性”对话框中设置垂直分辨率和水平分辨率。
4. 单击“确定”，关闭“外观设置属性”对话框，然后再次单击“确定”，关闭“选项”对话框。

   注意

   如果更改选项时设计器处于打开状态，请关闭设计器并重新打开，以使所做的更改生效。

请参见

参考

“外观设置属性”对话框（设备）

“选项”对话框 ->“设备工具”->“外观设置”

如何：更改默认设备（托管项目）

使用以下步骤更改托管项目中的默认目标设备。

为托管项目选择新的默认目标设备

1. 在“解决方案资源管理器”中，右击“<项目名称>”。
2. 在快捷菜单上单击“属性”，然后选择“设备”窗格。
3. 使用“目标设备”框中的下拉列表选择新的默认目标设备。

请参见

任务

如何：更改默认设备（本机项目）

参考

“部署”对话框（设备）

其他资源

创建和开发托管设备项目

如何:使用平台调用播放波形文件(设备)

下面的代码示例阐释如何在移动设备上使用平台调用 (PInvoke) 播放波形声音文件。

示例

此代码示例使用 `PlaySound` 在移动设备上播放声音文件。此代码使用 System.Runtime.InteropServices 调用 Compact FrameWork 的 CoreDll.DLL 的 `PlaySound` 方法。

```
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));
        }

    }
}
```

编译代码

- 在 Visual Studio 中创建一个新的 C# Smartphone 应用程序项目，并将它命名为**“MobileSoundPInvoke”**。
- 复制前一示例中的代码，然后在控制台应用程序中，将该代码粘贴至**“MobileSoundPInvoke”**项目的 Form1.cs 文件中。

可靠编程

- 确保将适当的参数传递给 `CMobilePlaySound (string szSound, IntPtr hMod, int flags)` 函数，包括 .wav 文件的路径和文件名。

安全

有关安全性的更多信息，请参见“.NET Framework Security”(.NET Framework 安全性)。

请参见

任务

平台调用技术示例

其他资源

智能设备示例

智能设备演练

在 C++ 中使用显式 PInvoke(DllImport 属性)

Creating a P/Invoke Library(创建 P/Invoke 库)

http://pinvoke.net/

在设备项目中管理代码段

Visual Basic 包括一个 IntelliSense 代码段代码库，只需单击几次鼠标便可将这类代码段插入应用程序。每个代码段都可执行一个完整的编程任务。同时，您还可以创建自己的代码段，将其添加到库中，并在需要时使用它们。

Visual Studio 已添加了一些设备项目专属的代码段。这些代码段的快捷方式都是以字符“sd”开始。有关更多信息，请参见 Visual Basic IntelliSense 代码段。

请参见

其他资源

创建和开发托管设备项目

托管设备项目中的数据

本主题已针对 Visual Studio 2005 SP1 进行了更新。

Visual Studio 2005 为设备解决方案提供了新的数据库管理工具，其中包括新建数据库，修改其架构和处理其他管理任务（例如设置密码和压缩数据库）的功能。

下面一节已针对 Visual Studio 2005 SP1 进行了更新。

本节内容

数据访问概述（托管设备项目）

描述 Visual Studio 中可用于处理设备项目中数据应用程序的一些功能。

结果集与数据集（设备）

描述这两种技术之间的差别，以及使用一种技术或另一种技术的原因。

生成类型化 ResultSet

更详细地描述 ResultSet 所提供的功能。

在设备项目中管理数据源

列出常见任务的步骤，如创建 SQL Server Mobile 或 SQL Server Compact Edition 数据库、修改表架构、管理密码等等。

请参见

任务

演练：数据库主/从应用程序

概念

“连接到 Visual Studio 中的数据”概述

其他资源

创建客户端数据应用程序

数据演练

使用 .NET Compact Framework 进行设备编程

SQL Server Mobile Edition Books Online Home Page

使用 .NET Compact Framework 进行设备编程

数据访问概述(托管设备项目)

本主题已针对 Visual Studio 2005 SP1 进行了更新。

用于开发执行数据操作的设备项目的 Visual Studio 环境与开发桌面数据应用程序的环境类似。设备的托管数据应用程序依赖于 .NET Compact Framework 支持的 ADO.NET 命名空间。这两种特点使得您能够创建这样一些应用程序:在这种应用程序中,设备上的数据存储区通常与服务器上的数据断开连接,只是定期进行数据同步。

设备上的数据:Visual Studio 2005 中的新增功能

下面一节已针对 Visual Studio 2005 SP1 进行了更新。

- 使用 Visual Studio IDE 可以创建 SQL Server Mobile 或 Microsoft SQL Server 2005 Compact Edition 数据库、分配密码、改变架构,以及执行其他基本数据库管理任务。
- 可以将类型化数据集、ResultSet、业务对象、SQL Server 数据库、SQL Mobile 数据库、SQL Server Compact Edition 数据库或 Web 服务用作数据源。
- 可以将这些数据源从"数据源"窗口拖放到 Windows 窗体上,以生成想要的数据绑定控件。

注意
"数据源配置向导"不可用于针对 .NET Compact Framework 1.0 版本的项目。

SQL Server Mobile Edition 和 SQL Server Compact Edition 体系结构

下面一节已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异:

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

SQL Server Mobile 和 SQL Server Compact Edition 对仅偶尔连接的设备而言,是最佳的本地数据引擎。可以使用 .NET Compact Framework 进行托管应用程序的编程开发,也可以使用 Microsoft Visual C++ 进行本机设备应用程序的编程开发。有关更多信息,请参见 SQL Server Mobile Architecture。

SQL Server Mobile Edition 和 SQL Server Compact Edition 的典型应用

下面一节已针对 Visual Studio 2005 SP1 进行了更新。

SQL Server Mobile 和 SQL Server Compact Edition 提供了一种针对移动设备上偶尔连接的数据访问方案的解决方案。当连接不可用时,经常需要通过移动企业方案来处理数据。SQL Server Mobile 和 SQL Server Compact Edition 通过提供在连接可用时能够与 SQL Server 进行同步的强大的关系存储区,满足了这些方案的需求。有关更多信息,请参见 Typical Uses of SQL Server Mobile。

注意
可以将 SQL Server Mobile 和 SQL Server Compact Edition 用作 Tablet PC 应用程序和设备应用程序的数据存储区。它们还可以在便携式计算机和桌面计算机上运行,前提是安装了 Visual Studio 2005 或 SQL Server 2005。有关更多信息,请参见Building a SQL Server Mobile Application for Tablet PCs。

SQL Server Mobile 和 SQL Server Compact Edition 的功能

下面一节已针对 Visual Studio 2005 SP1 进行了更新。

SQL Server Mobile 和 SQL Server Compact Edition 提供了丰富的功能,既可以作为 .NET Compact Framework 应用程序的一部分,也可以独立安装在智能设备上。您可以脱机操作其中的数据,并在以后与服务器同步。有关更多信息,请参见 SQL Server Mobile Features。

设计和管理 SQL Server 数据库

为了开发有效的设备数据应用程序,需要对优秀的数据库设计和 SQL Server 数据库引擎有所了解。同时应当掌握以下知识:如

何维护数据库，如何保证数据库的安全，如何访问和修改数据库中的数据，如何有效地查询数据以及如何最大化数据库的性能。有关更多信息，请参见Working with Databases (SQL Server Mobile)和Enhancing Performance (SQL Server Mobile)。

与服务器的连接

下面一节已针对 Visual Studio 2005 SP1 进行了更新。

SQL Server Mobile 和 SQL Server Compact Edition 支持在服务器上进行合并复制、远程数据访问以及安全规划和实现。有关更多信息，请参见Managing Connectivity (SQL Server Mobile)。

以编程方式实现常见任务

有关以编程方式实现常见任务的步骤，请参见How To (SQL Server Mobile)。

本地安全

SQL Server Mobile 和 SQL Server Compact Edition 数据库引擎提供密码保护和加密，以便保证设备上的本地数据库的安全。它们还为连接提供了安全选项。有关更多信息，请参见Securing Databases (SQL Server Mobile)。

请参见

任务

演练：数据库主/从应用程序
如何：安装示例数据库

概念

数据中的新增功能
保护连接字符串
“连接到 Visual Studio 中的数据”概述

其他资源

SQL Server Mobile Edition Books Online Home Page
创建客户端数据应用程序
数据演练
使用 .NET Compact Framework 进行设备编程
托管设备项目中的数据
访问数据 (Visual Studio)

结果集与数据集(设备)

当使用数据源向导创建新数据源时，Visual Studio 2005 同时生成数据集和结果集。结果集代码比数据集代码性能更强，而且相对要精简得多。另一方面，从某种意义上讲，数据集内容更丰富，可以保存多个表，同时还可以维护有关这些表之间的关系的信息。编程模型在这两种技术之间有显著区别。

可以根据需要有选择地生成任意一种结构，或者同时生成这两种结构。此功能供高级开发人员使用。有关更多信息，请参见如何：生成 SqlCeResultSet 代码(设备)。

请参见

任务

如何：生成 SqlCeResultSet 代码(设备)

概念

数据访问策略建议

其他资源

SQL Server Compact Edition 门户

托管设备项目中的数据

生成类型化 ResultSet

本主题已针对 Visual Studio 2005 SP1 进行了更新。

当 XSD 架构文件上的 **CustomTool** 属性设置为 **MSResultSetGenerator** 时，会生成类型化 ResultSet 数据源对象，而不是通常的类型化 **DataSet** 数据源对象。ResultSet 是快速的数据库游标，支持用户界面数据绑定、在数据中向后和向前滚动以及更新数据库中的数据。作为一种始终保持连接的模型，ResultSet 保持与数据库的活动连接。

下面一节已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

类型化 ResultSet 功能

生成的类型化 ResultSet 对象提供对数据库表的类型安全访问，这与类型化 **DataSets** 非常类似。这样，生成的代码确保应用程序和数据库之间传输的数据在编译时与数据库架构正确匹配。类型化 ResultSet 生成的代码扩展 SQL Server Mobile Edition 或 SQL Server Compact Edition ResultSet 基类，以提供特定于目标数据库表的属性和方法。

以下段落摘要介绍了为类型化 ResultSet 生成的代码的功能。

- **构造函数** 生成的类型化 ResultSet 包含两个构造函数。默认构造函数可用于设计时和基本运行时方案。Visual Studio 的设计时要求默认构造函数连接驻留在本地开发计算机上的数据库。这样，生成的代码就包含一个到本地 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库文件的连接字符串。对于运行时而言，默认构造函数从执行应用程序所在的同一个目录切换到本地数据库文件。这是基本运行时方案。
- **连接字符串** 当运行时方案更加复杂时，如当 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库文件驻留在一个不是执行目录的设备位置时，或连接字符串以其他某种方式变化（如密码）时，则可以使用重载构造函数。重载构造函数采用两个参数：一个自定义连接字符串和一个自定义 **ResultSetOptions** 标志。
- **连接属性** 连接属性是类型化 ResultSet 用来访问数据库数据的活动 **SqlCeConnection** 对象。连接属性是公共的，允许管理类型化 ResultSet 的连接状态。例如，有些事务可能要求关闭连接。
- **对表列的类型安全访问** 生成的代码为数据表中的每一列创建一个属性访问器。通过将基础数据库类型映射到 .NET Framework 类型来确定属性类型。例如，将 **nvarchar** 映射到 .NET Framework 字符串。生成的属性同时支持 **Get** 和 **Set** 访问器，让您可以使用 .NET Framework 字符串而不是基础数据库的 **nvarchar** 来推拉数据。每列还有 **IsxxxDBNull** 和 **SetxxxDBNull** 方法（其中，xxx 为列名），以方便获取 DBNull 并将其设置到数据库。
- **AddxxxRecord 方法**（其中 xxx 为表名） 此方法允许向数据库表中添加新行。**AddxxxRecord** 方法对数据表中的每一列都有一个参数，自动增加的列除外，这些列的值由数据库引擎分配。每个参数又再次声明为 .NET Framework 类型，这样就能够方便地用于应用程序，并抽象出基础数据库类型。在调用此方法后不必调用更新。调用之后，新行将提交到数据库。
- **DeletexxxRecord 方法**（其中 xxx 是表名） 此方法从数据库中删除当前行。与数据集不同，此转换不进行缓存，而且一旦调用，则从数据库中删除此行。在调用此方法后不必调用更新。
- **Bind 方法** Bind 方法采用一个参数，即 BindingSource，它被数据绑定到窗体上的一个或多个用户界面控件。然后，此方法将 **BindingSource** 数据绑定到类型化 ResultSet 实例，完成数据绑定链，并使控件能够直接显示数据库中的数据。

> **注意**
>
> 设计时方案需要一个硬编码的到 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库的连接字符串。某些时候，例如开发人员之间共享项目时，此连接字符串可能过期。因此，“数据源”窗口和/或 Windows 窗体设计器可能无法打开该项目。您可以重新生成类型化 ResultSet 代码以纠正这种情况。在“解决方案资源管理器”中，右击 XSD 架构文件，再单击“运行 CustomTool”。

请参见

参考

SqlCeResultSet

概念

结果集与数据集（设备）

其他资源

托管设备项目中的数据

在设备项目中管理数据源

以下几节描述如何在设备项目中实现常见数据任务。

注意
“数据源配置向导”不可用于针对 .NET Compact Framework 1.0 版本的项目。

本节内容

请参见

概念

“连接到 Visual Studio 中的数据”概述

其他资源

SQL Server Mobile Edition Books Online Home Page

托管设备项目中的数据

使用 SQL Server Mobile Edition 和 SQL Server Compact Edition 数据库

本主题是 Visual Studio 2005 SP1 中的新主题。

Visual Studio 2005 包括智能设备的开发人员在创建数据库应用程序时可使用的 SQL Server Mobile Edition。Visual Studio 2005 Service Pack 1 使开发人员能够使用 Microsoft SQL Server 2005 Compact Edition(它是同样包括针对桌面开发人员的功能的新版本)。对于智能设备的开发人员来说，SQL Server Compact Edition 和 SQL Server Mobile Edition 提供了一组相同的功能。

但是，为了利用 Visual Studio Service Pack 1 中丰富的设计时数据功能，必须使用 SQL Server Compact Edition 3.1 和 .NET Compact Framework 2.0 来编写数据应用程序。Visual Studio 2005 SP1 要求您同时安装 Visual Studio SP1 和 SQL Server Compact Edition 工具。有关更多信息，请参见如何：安装示例数据库

下表链接到的信息使您能够使用 Visual Studio 集成开发环境 (IDE) 来与数据库进行有效交互。

常见数据库任务

如何：创建数据库(设备)	如何：为数据应用程序生成摘要视图和编辑视图(设备)
如何：创建主/从应用程序(设备)	如何：管理数据库中的列(设备)
如何：创建参数化查询(设备)	如何：管理数据库密码(设备)
如何：生成 SqlCeResultSet 代码(设备)	如何：预览数据库中的数据(设备)
如何：缩小和修复数据库(设备)	如何：向设备项目添加数据库
如何：管理数据库中的表(设备)	如何：管理数据库中的索引(设备)
如何：添加导航按钮(设备)	如何：将数据更改持久地保存到数据库中(设备)

请参见

概念

带有 .NET Compact Framework 的 SQL Server Compact Edition

如何：创建数据库（设备）

本主题已针对 Visual Studio 2005 SP1 进行了更新。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

无论是否打开了项目，都可以创建 SQL Server Mobile 或 SQL Server Compact Edition 数据库。如果要使该数据库成为项目的一部分，则可以考虑在项目中将此数据库作为该项目的数据源进行创建。即使是在项目外创建该数据库，也可以在以后将其添加到项目。有关更多信息，请参见如何：向设备项目添加数据库。

注意
两个或多个进程无法通过网络共享同时访问同一 SQL Server Mobile 或 SQL Server Compact Edition 数据库，因为第二个进程将导致文件共享冲突错误。但是，一个进程可打开多个与数据库的连接。例如，进程可以同时在一个连接上运行“插入”查询，并在另一个连接上运行“选择”查询。通过网络共享打开数据库文件时，使用 DB_MODE_SHARE_EXCLUSIVE 文件模式。有关更多信息，请参见How to: Set the File Mode When Opening a Database with OLE DB (Programmatically)。

在项目外创建数据库

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 右击“数据连接”，然后单击“添加连接”，打开“添加连接”对话框。
3. 单击“更改”，打开“更改数据源”对话框。
4. 在“数据源”框中，选择“Microsoft SQL Server Mobile Edition”，然后单击“确定”。

   或者

   在“数据源”框中，选择“Microsoft SQL Server Compact Edition”，然后单击“确定”。
5. 选择“Microsoft SQL Server Mobile Edition”或“Microsoft SQL Server Compact Edition”，然后单击“确定”。
6. 从下面的“继续操作并配置连接”继续。

在项目内创建数据库

1. 打开一个项目后，在“数据”菜单上单击“添加新数据源”。

   将打开“数据源配置向导”。
2. 在“选择数据源类型”页上，选择“数据库”，然后单击“下一步”。
3. 在“选择您的数据连接”页上单击“新建连接”，打开“添加连接”对话框。
4. 单击“更改”，打开“更改数据源”对话框。
5. 选择“Microsoft SQL Server Mobile Edition”或“Microsoft SQL Server Compact Edition”，然后单击“确定”。
6. 从下面的“继续操作并配置连接”继续。

继续操作并配置连接

1. 在“添加连接”对话框中，选择“我的电脑”。
2. 单击“创建”。
3. 在“创建新的 SQL Server 2005 Mobile Edition 数据库”对话框中，键入新数据库的完全限定路径，如“c:\MyDB”。

   或者

   在“创建新的 SQL Server 2005 Compact Edition 数据库”对话框中，键入新数据库的完全限定路径（如“c:\MyDB”）。

4. 在“新密码”和“确认密码”框中键入新数据库的密码（如“MyPassword”），然后单击“确定”。

安全注意
对于将要在实际应用程序中使用的项目，请选择强密码。

5. 单击“确定”返回到“添加连接”对话框。
6. 在“添加连接”对话框中，单击“测试连接”以确保已建立连接。

   出现一条消息指示测试连接成功。
7. 单击“确定”，返回到“添加连接”对话框，然后单击“确定”关闭该对话框。

   仅当要在项目内创建数据库和数据集时，才完成以下步骤：
8. 在“选择您的数据连接”页上，选择“是，在连接字符串中包含敏感数据”。

安全注意
对于将要在实际应用程序中使用的项目，应选择从连接字符串中 *exclude* 敏感数据。至少，应确保敏感数据经过加密。

9. 单击“下一步”。

   出现“本地数据库文件”消息框，询问是否要在当前项目中包括数据文件。

   单击“是”。
10. 在“选择您的数据库对象”页上，选择要在项目中包含的表或其他对象。
11. 单击“完成”。

    这时通过在“数据”菜单上单击“显示数据源”，可以在“数据源”窗口中将新数据库作为数据集进行查看。若要向数据库中添加表、列、约束等，请参见在设备项目中管理数据源。

请参见

任务

如何：安装示例数据库

其他资源

Create Database Dialog (SQL Server Mobile)

在设备项目中管理数据源

如何：创建主/从应用程序（设备）

本主题已针对 Visual Studio 2005 SP1 进行了更新。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

以下步骤假定在“数据源”窗口中有可用的 SQL Server Mobile 或 SQL Server Compact Edition 数据库，且该数据库具有表关系。有关更多信息，请参见如何：创建数据库（设备）。

拖动详细信息表时，应仅拖动所需的那些列，而不是整个网格。可通过单击表名称右侧的箭头做出此选择。

以下过程假定您已打开了设备项目并配置了数据源。

创建主/从应用程序

1. 将主表从“数据源”窗口拖动到设计器中的窗体上。
2. 在“数据源”窗口中，展开该主表以公开详细信息表。
3. 将在主表节点下找到的详细信息表拖动到窗体上。

注意
这是显示在主表之内的详细信息表，而不是与该主表处于相同树级别的详细信息表。

   设计器从外键约束中自动检测主/从关系。有关更多信息，请参见演练：数据库主/从应用程序。
4. 调整窗体上的控件使其适合您的应用程序。

请参见

任务

演练：数据库主/从应用程序

如何：安装示例数据库

其他资源

在设备项目中管理数据源

如何：创建参数化查询（设备）

本主题已针对 Visual Studio 2005 SP1 进行了更新。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

以下步骤假定在“数据源”窗口中有可用的 SQL Server Mobile 或 SQL Server Compact Edition 数据库。有关更多信息，请参见如何：创建数据库（设备）或如何：向设备项目添加数据库。

如果需要用户能为一个参数输入不同的值，在设计查询时，请将问号（“?”）用作该参数。如果在 Windows 窗体设计器上使用此智能标记创建查询（如下面的一组步骤所示），则会在 Windows 窗体中自动生成一个用户界面。如果在数据集设计器中从“TableAdapter”创建查询（如最后的一组步骤所示），则不会自动生成用户界面。

使用 Windows 窗体设计器为指定参数进行设置

1. 从“数据源”窗口中将一个 Datagrid 格式或详细信息格式的表拖动到设计器中的窗体上。

   单击表名右侧的箭头可选择格式。

2. 单击被拖动组件上的智能标记，再在快捷菜单上单击“添加查询”。

   如果没有使用鼠标，则可以通过键盘快捷键 Shift+Alt+F10 来打开“任务”对话框。

3. 在“查询标准生成器”对话框中选择“新查询名称”。

   使用默认名称或自己创建一个名称。

4. 现在，可以通过更改“查询文本”框中的 SQL 语句或单击“查询生成器”来指定参数。

使用“查询文本”框指定参数

1. 将 `WHERE` 子句添加到 `SELECT` 语句的末尾。
2. 单击“确定”关闭“查询标准生成器”对话框。

   设计器中的窗体上出现一个查询绑定按钮。

使用“查询生成器”指定参数

1. 在“查询生成器”对话框中执行以下操作：

   - 在“SQL 语句”窗格中添加一个 `WHERE` 子句。

     - 或 -

   - 在相应的“列”列表中的“筛选器”下输入参数。

     此方法可在“SQL 语句”窗格中编写 `WHERE` 子句。

2. 单击“确定”，关闭“查询生成器”对话框。
3. 单击“确定”关闭“查询标准生成器”对话框。

   设计器中的窗体上出现一个查询绑定按钮。

使用数据集设计器指定参数

1. 在“解决方案资源管理器”中，右击 .xsd 文件，然后单击“打开”。
2. 在“数据集设计器”中，右击“TableAdapter”，指向“添加”，然后在快捷菜单上单击“查询”。
3. 在 TableAdapter 查询配置向导中，选择“使用 SQL 语句”，然后单击“下一步”。

4. 在“选择查询类型”页上，选择“SELECT(返回单个值)”，然后单击“下一步”。
5. 在“指定 SQL SELECT 语句”页上，单击“查询生成器”。

   如果需要，可以在此添加 WHERE 子句。
6. 按此主题前面所述的方法使用查询生成器。

注意
当使用“TableAdapter 查询配置向导”创建查询时，不会自动生成用户界面元素。

请参见

任务

如何：向 Windows 应用程序中的窗体添加参数化查询

演练：参数化查询应用程序

如何：安装示例数据库

参考

“选择标准生成器”对话框

概念

查询和视图设计器工具

其他资源

在设备项目中管理数据源

如何：生成 SqlCeResultSet 代码（设备）

本主题已针对 Visual Studio 2005 SP1 进行了更新。

下面一节已针对 Visual Studio 2005 SP1 进行了更新。

以下步骤假定在“数据源”窗口中有可用的 SQL Server Mobile 或 Microsoft SQL Server 2005 Compact Edition 数据库。有关更多信息，请参见如何：创建数据库（设备）或如何：向设备项目添加数据库。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

在设备项目中，Visual Studio 在默认情况下仅为数据集生成代码。您可以更改此默认值，以生成结果集代码，或同时生成这两种代码。有关这两种类型间差别的更多信息，请参见 结果集与数据集（设备）。

只有针对 SQL Server Mobile 或 SQL Server Compact Edition 连接所创建的 .xsd 文件才支持结果集和结果集/数据集选项。所有连接都支持数据集选项，而且是默认生成的代码。

注意
如果想要将现有的数据集应用程序转换为结果集应用程序，请将“自定义工具”属性设置为“MSDataSetResultSetGenerator”。此设置同时生成这两种数据访问类。然后，可以将代码从一种类型迁移至另一种类型，且不会出现生成错误。在迁移代码后，请将“自定义工具”属性设置为“MSResultSetGenerator”。进行重新生成，确认已从代码中移除所有的数据集使用。

只为结果集生成代码

1. 在“解决方案资源管理器”中右击数据库 .xsd 文件，再单击快捷菜单上的“属性”。
2. 将“自定义工具”值设置为“MSResultSetGenerator”。
3. 重新生成解决方案。

只为数据集生成代码

1. 在“解决方案资源管理器”中右击数据库 .xsd 文件，再单击快捷菜单上的“属性”。
2. 将“自定义工具”值设置为“MSDataSetGenerator”。

   此值为默认值。
3. 重新生成解决方案。

同时为结果集和数据集生成代码

1. 在“解决方案资源管理器”中右击数据库 .xsd 文件，再单击快捷菜单上的“属性”。
2. 将“自定义工具”值设置为“MSDataSetResultSetGenerator”。
3. 重新生成解决方案。

请参见

任务

如何：安装示例数据库

概念

结果集与数据集（设备）

其他资源

在设备项目中管理数据源

如何：为数据应用程序生成摘要视图和编辑视图（设备）

本主题已针对 Visual Studio 2005 SP1 进行了更新。

使用数据窗体查看和编辑数据网格中的单行数据。

数据窗体用户界面包括两个对话框："视图"对话框和"编辑"对话框，前者用于显示选定数据网格行的摘要视图，后者用于编辑数据行。

- 在"设备仿真程序"中双击数据网格中的某一行，或者在设备上用笔针点击某一行，便可在运行的应用程序中打开"视图"对话框。
- 显示数据网格时，通过单击（用笔针点击）"新建"可打开"编辑"对话框（此操作会在数据网格中新建一行）；或者在显示"视图"对话框时，通过单击（用笔针点击）"编辑"也可打开该对话框。

数据窗体被设计为自定义的模板。您应将用于验证对数据库的更改和提交这些更改的适当代码添加为此自定义的一部分。

通过使用数据窗体所做的更改不会保留在数据库中。有关如何保留这些更改的更多信息，请参见如何：将数据更改持久地保存到数据库中（设备）。

注意
只有针对 SQL Server Mobile、SQL Server Compact Edition 和服务器数据库数据集创建的数据源支持该功能。它不受结果集、业务对象或 Web 服务支持。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

以下步骤假定在"数据源"窗口中有可用的 SQL Server Mobile 或 SQL Server Compact Edition 数据库。有关更多信息，请参见如何：创建数据库（设备）。本主题也描述了如何将现有 SQL Server 数据库作为数据源添加到您的项目中。

生成数据窗体

1. 从"数据源"窗口中将一个数据网格形式的表拖动到设计器中的窗体上。

   单击表名右侧的箭头可选择格式。

2. 单击被拖动组件上的智能标记，然后在快捷菜单上选择"生成数据窗体"。（如果没有使用鼠标，则可以通过键盘快捷键 Shift+Alt+F10 来打开"任务"对话框）。

注意
在设计器中生成对话框时，这些对话框会短暂地显示在屏幕上。若要验证对话框是否已成为项目的组成部分，请在"解决方案资源管理器"中查找它们。

在运行的应用程序中修改数据

1. 启动数据应用程序。

   已填充了数据的数据网格将显示。

2. 双击某个数据行。

   该行的摘要视图便会出现在"视图"对话框中。此视图由具有内容的各列的标签和数据组成。也就是说，"视图"对话框会隐藏值为 DBNULL 的所有列。

3. 在主菜单上单击"编辑"，打开"编辑"对话框。

   使用"编辑"对话框（其中显示了所有列）修改数据，再单击确定。

修改后的数据显示在数据网格中。

在运行的应用程序的数据网格中创建新行

1. 在数据网格已经打开的前提下，单击主菜单上的“新建”。

   便会出现“编辑”对话框。使用此对话框添加新的数据行。

2. 单击“确定”。

   新行便被添加到数据网格中。

请参见

其他资源

在设备项目中管理数据源

如何：管理数据库中的列（设备）

本主题已针对 Visual Studio 2005 SP1 进行了更新。

以下步骤假定在“服务器资源管理器”窗口中有可用的 SQL Mobile 或 Microsoft SQL Server 2005 Compact Edition 数据库。有关更多信息，请参见如何：创建数据库（设备）。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

添加列

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”窗口中，展开数据连接以显示其表。
3. 右击要向其中添加列的表，再单击快捷菜单上的“编辑表架构”。
4. 在“编辑表”窗口中添加列及其属性，再单击“确定”。

编辑列属性

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”窗口中，展开数据连接以显示其表。
3. 右击包含要编辑其属性的列的表，然后在快捷菜单上单击“编辑表架构”。
4. 进行编辑并单击“确定”。

注意
所做的编辑不能破坏引用完整性（例如，当列由另一约束引用时，试图将“主键”属性更改为“否”，则会破坏引用完整性）。

从表中移除列

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”窗口中，展开数据连接以显示其表。
3. 右击要从中移除列的表，再单击快捷菜单上的“编辑表架构”。
4. 在“编辑表”窗口中选择要删除的列，单击“删除”，然后单击“确定”。

注意
只有删除对一列的所有引用之后，才能够移除该列。

请参见

任务

如何：安装示例数据库

其他资源

Column Properties

在设备项目中管理数据源

如何：管理数据库密码（设备）

本主题已针对 Visual Studio 2005 SP1 进行了更新。

可以在创建数据库时设置密码，还可以更改现有数据库中的密码。

安全注意
在连接字符串中包含密码会带来一定的安全风险。有关更多信息，请参见Securing Databases (SQL Server Mobile)。

下面的示例步骤假定您已经打开了一个 Pocket PC Windows 窗体应用程序。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

在创建数据库时设置密码

1. 在“数据”菜单上单击“添加新数据源”。
2. 在“选择数据源类型”页上，单击“数据库”，然后单击“下一步”。
3. 在“选择您的数据连接”页上单击“新建连接”，打开“添加连接”对话框。
4. 单击“更改”，单击“Microsoft SQL Server Mobile Edition”，然后单击“确定”。

   或者

   单击“更改”，单击“Microsoft SQL Server Compact Edition”，然后单击“确定”。
5. 在“添加连接”对话框中，单击“我的电脑”。
6. 单击“创建”。
7. 在“创建新的 SQL Server 2005 Mobile Edition 数据库”对话框中，键入新数据库的完全限定路径（例如，c:\MyDatabase.sdf）。

   或者

   在“创建新的 Microsoft SQL Server Compact Edition 数据库”对话框中，键入新数据库的完全限定路径（例如，c:\MyDatabase.sdf）。
8. 在“新密码”和“确认密码”框中键入新数据库的密码，然后单击“确定”。

更改现有数据库中的密码

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”中，右击要更改其密码的数据源。
3. 在快捷菜单上单击“数据库属性”。
4. 在“数据库属性”对话框中单击“设置密码”。
5. 根据提示键入旧密码和新密码，然后单击“确定”。

请参见

任务

如何：创建数据库（设备）

其他资源

Securing Databases (SQL Server Mobile)

在设备项目中管理数据源

如何：预览数据库中的数据（设备）

本主题已针对 Visual Studio 2005 SP1 进行了更新。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

以下步骤假定在“服务器资源管理器”窗口中有可用的 SQL Server Mobile 或 SQL Server Compact Edition 数据库。有关更多信息，请参见如何：创建数据库（设备）。

可以使用“服务器资源管理器”在项目外预览数据，或使用“数据源”窗口在项目内预览数据，在该窗口中还可以对产生视图的查询进行参数化处理。有关更多信息，请参见如何：创建参数化查询（设备）。

使用服务器资源管理器查看数据

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”窗口中展开数据连接，以公开表列表。
3. 右击要查看其数据的表，再单击快捷菜单上的“打开”。

使用项目中的“数据源”窗口查看数据

1. 在“数据”菜单上单击“显示数据源”。
2. 在“数据源”窗口中右击要查看其数据的数据源（例如一个表）。
3. 在快捷菜单上单击“预览数据”。
4. 在“数据预览”对话框中单击“预览”。

使用 DataGrid 控件上的智能标记查看数据

1. 在设计器中，单击 DataGrid 控件上的智能标记。
2. 在“DataGrid 任务”快捷菜单上单击“预览数据”。
3. 在“预览数据”对话框中，选择要预览的对象，然后单击“预览”。

   数据出现在“结果”框中。“预览数据”对话框的底部还显式网格中的列数和行数。

请参见

任务

如何：安装示例数据库

其他资源

在设备项目中管理数据源

如何：向设备项目添加数据库

本主题已针对 Visual Studio 2005 SP1 进行了更新。

以下步骤假定在“服务器资源管理器”窗口中有可用的 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库。有关更多信息，请参见如何：创建数据库（设备）。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

将 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库作为数据源添加

1. 在智能设备项目打开的前提下，单击“数据”菜单上的“添加新数据源”。
2. 在“选择数据源类型”页上，选择“数据库”，然后单击“下一步”。
3. 在“选择您的数据连接”页上，选择包含数据库名称的数据连接字符串，再单击“下一步”。

   如果数据库已在服务器资源管理器中，此连接应显示在下拉框中。如果数据库连接没有出现在列表中，请单击“新建连接”以打开“添加连接”对话框，单击“浏览”打开“选择数据库文件”对话框，导航至您的数据库，然后单击“打开”。单击“确定”关闭“添加连接”对话框。
4. 在“选择您的数据连接”页上单击“下一步”。

   出现“本地数据库文件”消息框，询问是否要在当前项目中包括数据文件。

   单击“是”。
5. 在“选择数据库对象”页上选择要在项目中用作数据源的对象。
6. 单击“完成”。

   从 SQL Server Mobile 或 SQL Server Compact Edition 数据库中选择的对象便会作为“数据源”窗口中项目的数据源显示。若要打开该窗口，单击“数据”菜单上的“显示数据源”。

请参见

其他资源

在设备项目中管理数据源

如何：缩小和修复数据库（设备）

本主题已针对 Visual Studio 2005 SP1 进行了更新。

以下步骤假定在“服务器资源管理器”窗口中有可用的 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库。有关更多信息，请参见如何：创建数据库（设备）。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

缩小和修复数据库

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”窗口中右击要缩小和修复的数据连接，再在快捷菜单上单击“数据库属性”。
3. 在“选择页”窗格中单击“缩小和修复”。
4. 从“缩小和修复”页上的选项中进行选择。

   页底部显示所选选项的说明。
5. 单击“确定”，启动所选的缩小和修复选项，或单击“取消”，保持数据库的当前状态。

请参见

其他资源

Shrink & Repair

在设备项目中管理数据源

如何：管理数据库中的表（设备）

本主题已针对 Visual Studio 2005 SP1 进行了更新。

以下步骤假定在“服务器资源管理器”窗口中有可用的 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库。有关更多信息，请参见如何：创建数据库（设备）。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

添加表

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”窗口中展开要在其中添加表的数据连接。
3. 右击“表”，然后单击创建表。
4. 在“新建表”对话框的“名称”框中键入表名称。
5. 在第一列中，输入“列名”、“数据类型”、“长度”、“允许空”、“唯一”和“主键”的值。继续其他列。
6. 单击“确定”。

   新表显示在数据连接的表列表中。

编辑现有的表架构

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”窗口中展开可在其中找到要编辑的表架构的数据连接。
3. 右击要编辑的表，然后在快捷菜单上单击“编辑表架构”。
4. 在“编辑表 <Tablename>”对话框中做出更改，然后单击“确定”。

移除表

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”窗口中展开要从中删除表的数据连接。
3. 在“表”之下右击要删除的表，再在快捷菜单上单击“删除表”。
4. 在“删除对象”对话框中单击“移除”，再单击“确定”。

注意
在删除对一张表的所有引用之前无法移除该表。

请参见

其他资源

New Table
Edit Table (SQL Server Mobile)
Table Properties
在设备项目中管理数据源

如何：管理数据库中的索引（设备）

本主题已针对 Visual Studio 2005 SP1 进行了更新。

以下步骤假定在“服务器资源管理器”窗口中有可用的 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库。有关更多信息，请参见如何：创建数据库（设备）。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

创建 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库的索引

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”窗口中，展开要在其中创建新索引的数据源。
3. 展开要为其创建新索引的表，再右击“索引”文件夹。
4. 在快捷菜单上单击“创建索引”。
5. 在“新建索引”对话框中键入索引的名称，再单击“添加”。
6. 在“从 <Table> 选择列”对话框中选择要添加到索引键的列，再单击“确定”。
7. 单击“确定”关闭“新建索引”对话框。

仅编辑索引的排序顺序属性

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”窗口中，展开数据连接以及包含要编辑其属性的索引的表。
3. 右击索引，再单击快捷菜单上的“索引属性”。
4. 在此对话框中可以更改排序顺序。

   不能够更改索引的其他属性，除非将现有索引替换为新的索引。

从 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库中删除索引

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在“服务器资源管理器”窗口中，展开要从中删除索引的数据连接。
3. 展开具有索引的表，再展开“索引”，然后右击要删除的索引。
4. 在快捷菜单上单击“删除索引”。
5. 在“删除对象”对话框中单击“移除”，再单击“确定”。

请参见

其他资源

New Index
Index Properties (SQL Server Mobile)
在设备项目中管理数据源

如何：添加导航按钮（设备）

使用这些过程可以提供导航按钮，以便查看数据源中的不同数据行。此技术可解决 .NET Compact Framework 对 .NET Framework 的 **DataNavigator** 类支持不足的问题。

下面的步骤是用 C# 编写的，它们依赖于 Northwind 数据库的 Customers 表，并假定“数据源”窗口中有一个数据集或结果集。有关更多信息，请参见如何：向设备项目添加数据库。在实际项目中，可能会包括绑定检查，但此处的代码示例中并未显示。

设置以添加导航按钮

1. 将表从“数据源”窗口拖放到 Windows 窗体上。
2. 将按钮拖放到窗体上。
3. 相应地设置按钮的“Text”属性（例如“Next”）。
4. 双击窗体上的按钮，打开代码编辑器并自动定位到按钮单击事件处理程序处。
5. 使用下面的代码示例，编写“第一行”、“下一行”、“上一行”和“最后一行”按钮事件处理程序的代码。

编写“第一行”按钮的代码

- 键入 `this.customersBindingSource.MoveFirst();`

编写“下一行”按钮的代码

- 键入 `this.customersBindingSource.MoveNext();`

编写“上一行”按钮的代码

- 键入 `this.customersBindingSource.MovePrevious();`

编写“最后一行”按钮的代码

- 键入 `this.customersBindingSource.MoveLast();`

请参见

任务

如何：生成 SqlCeResultSet 代码（设备）

其他资源

在设备项目中管理数据源

如何：将数据更改持久地保存到数据库中（设备）

使用这些过程可保持对设备项目中的数据所做的更改。

下面的步骤是用 C# 编写的，它们依赖于 Northwind 数据库的 Customers 表，并假定“数据源”窗口中已有一个数据集（而不是任何其他类型的数据源）。有关更多信息，请参见如何：向设备项目添加数据库。

保持对数据库的数据更改

1. 将一个表从“数据源”窗口中拖动到 Windows 窗体上。
2. 将一个按钮拖动到窗体上。
3. 将按钮的“文本”属性更改为“保存”。
4. 双击窗体上的按钮，打开代码编辑器并自动定位到按钮单击事件处理程序处。
5. 键入以下代码：

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

请参见

其他资源

在设备项目中管理数据源

如何:更改设计时连接字符串(设备)

本主题已针对 Visual Studio 2005 SP1 进行了更新。

默认情况下,为项目创建 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据源时,会创建设计时连接字符串。Visual Studio 在设计时使用此字符串连接到数据库以检索架构信息。在项目开发过程中,您可能希望更改此字符串。

安全注意
在连接字符串中包含密码会带来一定的安全风险。有关更多信息,请参见Securing Databases (SQL Server Mobile)。

注意
请将此字符串与运行时连接字符串区分开。有关更多信息,请参见文件属性对话框中的连接字符串属性(设备)。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异:

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

更改设计时连接字符串

1. 在"解决方案资源管理器"中,双击 .xsd 文件打开"数据集设计器"。
2. 右击"TableAdapter",然后在快捷菜单上单击"属性"。
3. 展开"Connection"属性。
4. 为"ConnectionString"键入新的值。

请参见

任务

如何:编辑连接字符串

其他资源

在设备项目中管理数据源

如何：更改运行时连接字符串（设备）

为项目添加数据源时，将在 .xsd 文件中生成一个连接字符串。该字符串被视为设计时连接字符串，适用于在设计时将 Visual Studio 连接到该数据源。同时，该字符串还充当默认的运行时连接字符串。

如果当应用程序在设备上运行时需要一个不同的字符串，可以使用下列步骤指定运行时字符串。有关更多信息，请参见文件属性对话框中的连接字符串属性（设备）。

安全注意
在连接字符串中包含密码会带来一定的安全风险。有关更多信息，请参见Securing Databases (SQL Server Mobile)。

更改运行时连接字符串

1. 在“解决方案资源管理器”中选择 .xsd 文件。
2. 在“视图”菜单上单击“属性窗口”。
3. 为“连接字符串”属性键入运行时连接字符串。

请参见

任务

如何：更改设计时连接字符串（设备）

其他资源

在设备项目中管理数据源

如何：添加业务对象作为数据源（设备）

可以将业务对象作为数据源添加到智能设备项目中。有关更多信息，请参见Visual Database Tools。

从对象中创建新的数据源

1. 在“数据”菜单上单击“添加新数据源”。
2. 在“选择数据源类型”页上选择“对象”。
3. 在“选择希望绑定到的对象”页面上选择应用程序中已有的对象。

   注意

   您可能需要在对象出现在向导中之前生成包含该对象的项目。单击“添加引用”并在“添加引用”对话框中查找所需的程序集，还可以添加一个对当前不在应用程序中的对象的引用。程序集将被添加到树视图中。

4. 展开包含要绑定到的对象的程序集，然后再在树视图中选择对象。
5. 单击“完成”。

   该数据源将被添加到“数据源”窗口中。

向窗体中添加数据源

1. 在“数据”菜单上单击“显示数据源”，打开“数据源”窗口。
2. 在“数据源”窗口中选择项，并将它们拖动到 Windows 窗体上，以创建绑定到对象中的属性的控件。有关更多信息，请参见“显示数据”概述。

请参见

其他资源

在设备项目中管理数据源

如何：添加 Web 服务作为数据源（设备）

在智能设备项目中，可以将 Web 服务作为数据源添加。

将应用程序连接到 Web 服务

1. 在“数据”菜单上单击“添加新数据源”。
2. 在“选择数据源类型”页上，选择“Web 服务”，然后单击“下一步”。
3. 使用“添加 Web 引用”对话框添加一个对所需的 Web 服务的引用。
4. 单击“完成”。

   该数据源将被添加到“数据源”窗口中。

向窗体中添加数据源

- 在“数据源”窗口中选择项并将其拖动到一个 Windows 窗体上，从而创建绑定控件。有关更多信息，请参见“显示数据”概述。

请参见

其他资源

在设备项目中管理数据源

如何：添加 SQL Server 数据库作为数据源（设备）

可以将 SQL Server 数据库作为数据源在托管设备项目中使用。

向“服务器资源管理器”中添加 SQL Server 数据连接

1. 在“视图”菜单上单击“服务器资源管理器”。
2. 在服务器资源管理器窗口中右击“数据连接”，再单击快捷菜单上的“添加连接”。
3. 在“添加连接”对话框中单击“更改”。
4. 在“更改数据源”对话框中选择“Microsoft SQL Server”，再单击“确定”，重新打开“添加连接”对话框。
5. 在“服务器名称”框中指定数据源所在的服务器的名称。
6. 登陆到服务器。

   如果服务器使用 SQL Server 身份验证，则需要提供用户名和密码。
7. 在“选择或输入数据库名称”框中选择或键入数据库名称，再单击“确定”。

   新的数据连接便会显示在“服务器资源管理器”中。

将 SQL Server 数据连接用作项目中的数据源

1. 系统必备：在 Visual Studio IDE 中必须具有已经打开的 .NET Compact Framework 智能设备项目。
2. 在“数据”菜单上，单击“添加新数据源”打开“数据源配置向导”。
3. 在“选择数据源类型”页上，选择“数据库”，然后单击“下一步”。
4. 在“选择您的数据连接”页上，单击“新建连接”。
5. 在“添加连接”对话框中单击“更改”。
6. 在“更改数据源”对话框中选择“Microsoft SQL Server”，再单击“确定”，重新打开“添加连接”对话框。
7. 在“服务器名称”框中键入或选择数据源所在的服务器的名称。
8. 登陆到服务器。

   如果服务器使用 SQL Server 身份验证，则需要提供用户名和密码。
9. 在“选择或输入数据库名称”框中选择或键入数据库名称，再单击“确定”。
10. 在“选择您的数据连接”页上，选择从连接字符串中排除敏感数据，然后单击“下一步”。

安全注意
在连接字符串中包含敏感数据会带来一定的安全风险。

11. 在“选择数据库对象”页上选择要用作数据源的对象，然后单击“完成”。

    现在数据连接便会作为数据源出现在“数据源”窗口中。

请参见

其他资源

在设备项目中管理数据源

演练：创建简单应用程序

此演练演示如何使用 Visual Studio 2005，通过 Visual C# 或 Visual Basic 创建简单的智能设备项目。此演练还演示如何在 PC 的仿真程序中或在与 PC 连接的实际设备上运行应用程序。

注意

若要在一台实际设备上永久安装一个完成的应用程序，必须先将该应用程序打包到一个 CAB 文件中。有关更多信息，请参见演练：打包智能设备解决方案以便进行部署。

注意

对话框中提供的选项和菜单命令会随开发设置的不同而有所不同。此演练是使用 Visual Basic 开发设置和 Visual C# 开发设置编写的。若要查看或更改设置，请选择“工具”菜单上的“导入和导出设置”，单击“重置 IDE 设置”，选择所需的设置，再单击“重置设置”。

本演练由四项主要任务组成：

- 选择目标设备。
- 创建设备项目。
- 在窗体中添加按钮和事件处理程序。
- 在目标设备上运行应用程序。

选择目标设备

可以针对仿真程序或物理设备执行本演练。甚至可以在演练期间来回切换。为了确保每次部署解决方案时系统提示您选择设备，请完成以下过程。

在部署时提示选择设备

1. 在“工具”菜单上依次单击“选项”、“设备工具”和“常规”。如果看不到“设备工具”，请选择“选项”对话框底部的“显示所有设置”。
2. 选择“部署设备项目前显示设备选项”复选框。

创建设备项目

在本节中，您将生成一个简单的 Windows 窗体应用程序。以下过程包括创建一个 Windows 窗体项目，向窗体中添加一个控件，为控件添加事件处理，最后生成并测试应用程序。此处未给出创建 C++ 项目的步骤。如果您是在 C++ 环境中开发，则可以创建一个基本的 Hello World 应用程序，生成一个发布版本，然后跳至“生成、测试和保存应用程序”一节。

创建设备项目

1. (Visual Basic) 在 Visual Studio 的“文件”菜单上单击“新建项目”。

   - 或 -

   (Visual C#) 在 Visual Studio 的“文件”菜单上，指向“新建”，再单击“项目”。
2. 在“新建项目”对话框的“项目类型”之下，展开“Visual Basic”或“Visual C#”，再展开“智能设备”，然后再单击“Pocket PC 2003”。

   如果开始时未显示您需要的语言，请展开“其他语言”。该显示受配置文件设置的控制。
3. 在“模板”下，单击“设备应用程序”。
4. 在“名称”框中键入 **AppForDeployment**。
5. （仅适用于 Visual C#）在“位置”框中，确认要用于存储项目文件的位置。
6. 单击“确定”。

   Pocket PC 设备的表示形式便会出现在 Windows 窗体设计器中。

向窗体添加控件

1. 将一个“Button”控件从“工具箱”中拖动到窗体上。

   如果看不到“工具箱”，请在“视图”菜单上单击“工具箱”。

   如果在“工具箱”中看不到“设备控件”选项卡，请右击“工具箱”并选择“全部显示”，或选择“重置工具箱”。

2. 右击“Button”控件，再单击“属性”。

3. 在“属性”窗口中键入 **Say Hello**，再按 Enter 设置“Text”属性。

为 Button 控件添加事件处理

1. 双击窗体上的按钮。

   随即打开代码编辑器，并且插入点位于事件处理程序中。

2. 插入以下 Visual Basic 代码：

   ```
   MessageBox.Show("Hello, World!")
   ```

   - 或 -

   插入以下 C# 代码：

   ```
   MessageBox.Show("Hello, World!");
   ```

生成、测试和保存应用程序

1. 在“调试”菜单上单击“启动”。

2. 在“部署”对话框中选择“Pocket PC 2003 SE 仿真程序”，再单击“部署”。

   “输出”窗口、“状态”栏和仿真程序屏幕上都将显示进度消息。

3. 当应用程序在仿真程序上运行时，点击“Say Hello”按钮，确信显示了“Hello, World!”。

4. 在仿真程序的“文件”菜单上单击“退出”，以在进入本演练的下一节前关闭仿真程序。

5. 在“设备仿真程序”对话框中单击“否”，对“退出前保存状态”提示作出响应。

6. 在 Visual Studio 的“文件”菜单上单击“全部保存”。

   如果仿真程序尚未完全关闭，便会显示提示“要停止调试吗?”。单击“是”。

7. 在“保存项目”对话框中，将“AppForDeployment”项目保存到所选位置。

请参见

其他资源

使用 .NET Compact Framework 进行设备编程

使用 Visual C++ 进行设备编程

Visual Studio 2005 为使用 Visual C++ 开发设备应用程序提供支持。

本节内容

Visual C++ 设备应用程序开发中的新增功能

提供与用于设备的 Visual C++ 的新功能有关的信息。

创建并移植 Visual C++ 设备项目

解释如何创建一个新的 Visual C++ 设备项目，以及如何将现有 Visual C++ 桌面项目和 eMbedded Visual C++ 4.0 项目移植到 Visual C++ 设备项目中。

开发 Visual C++ 设备项目

描述如何使用 Visual Studio 2005 的功能来对使用 Visual C++ 的设备应用程序进行开发。

生成并调试 Visual C++ 设备项目

解释如何生成并调试 Visual C++ 设备项目。

请参见

其他资源

智能设备开发

Mobile Developer Center（移动开发人员中心）

Visual C++ 设备应用程序开发中的新增功能

Visual Studio 2005 在 Visual C++ 中包含了设备开发。以前，使用 Visual C++ 开发设备项目需要 eMbedded Visual C++ 版本。以下部分详细介绍的新增功能是对 eMbedded Visual C++ 的改进。

集成开发环境 (IDE)

面向多个操作系统

Visual Studio 2005 工具集提供的功能可以面向 Pocket PC 2003、Smartphone 2003、基于 Windows CE 5.0 的自定义 SDK 以及 SDK 将来的版本。

项目系统

项目系统现在将平台与其受支持的 CPU 结构关联。eMbedded Visual C++ 的早期版本允许选择当前活动项目不支持的 CPU 结构。

IntelliSense

在 Visual Studio 2005 中，IntelliSense 在各个软件开发工具包 (SDK) 头文件中进行反映，以提供有关目标平台的准确信息。

混合模式解决方案

Visual Studio 2005 让设备开发人员可以在同一个解决方案中同时包含托管的 Visual C# 或 Visual Basic 代码和非托管的 Visual C++ 代码。它还允许桌面和设备 Visual C++ 代码包含在同一个项目中。

应用程序安装

允许开发人员将其设备应用程序打包并创建桌面安装程序，将应用程序分发给设备。

自定义应用程序和类向导

Visual Studio 2005 使为设备项目编写应用程序和类向导更加容易。

资源编辑器

“资源编辑器”包括设备特定资源，例如“输入面板的状态”和“CAPEdit”控件。

调试

Visual Studio 2005 C++ 设备调试器具有以下新功能和改进：

- 在以前的设备应用程序调试器的基础上改进了可靠性和性能。
- 分离和重新附加到进程的能力。
- 无需可执行文件也可附加到进程的能力。
- 支持共享 DLL 中的断点。
- 多个进程调试。
- 改进的表达式计算，包含更多表达式调试器窗口。

编译器

Visual Studio 2005 C++ 设备编译器具有以下新功能和改进：

- 支持 /LTCG（链接时代码生成）。
- 增加的编译器限制。
- 对缓冲区溢出检测的 **/GS** 切换支持。
- 支持 __restrict 关键字。
- 进行更新，以更高程度符合 C++ 标准。

库

用于设备的 Visual C++ 库具有以下新功能和改进：

- 用于设备的 Microsoft 活动模板库 (ATL) 和 Microsoft 基础类 (MFC) 已更新至 8.0 基本代码。

- 用于设备的 MFC、用于设备的 ATL、用于设备的 C 运行时 (CRT) 库以及用于设备的标准 C++ 库中具有扩展的安全性、性能、稳定性和标准遵从性。
- 合并了通用的 ATL 和 MFC 功能。
- 向用于设备的标准 C++ 库中添加了流支持。

ATL

设备的 ATL 中添加了以下功能：

- ActiveX 控件宿主。
- 使用 Web 服务的功能。
- 具有 **CImage** 类的位图支持。
- 用于管理数组、列表和目录树的新类。
- 增强的字符串处理和转换功能。
- 扩展的对 IPv6 的套接字支持。

MFC

设备的 MFC 中添加了以下功能：

- 现在可用于 Smartphone。
- ActiveX 控件宿主和创建。

请参见

其他资源

使用 Visual C++ 进行设备编程

创建并移植 Visual C++ 设备项目

本主题已针对 Visual Studio 2005 SP1 进行了更新。

Visual Studio 2005 提供开始 C++ 设备项目的多种方法。本节包括有关创建新的 Visual C++ 设备项目，以及允许现有项目以 Visual Studio 2005 环境中的设备为目标的各个主题。

下表已针对 Visual Studio 2005 SP1 进行了更新。

本节内容

如何：创建新的 Visual C++ 设备项目

介绍如何创建新的 ATL、MFC 和本机 Windows CE 应用程序项目。

eMbedded Visual C++ 到 Visual Studio 2005 升级向导

介绍如何使用升级向导，将使用 eMbedded Visual C++ 3.0 和 4.0 编写的项目导入 Visual Studio 2005。

从 eVC 移植所带来的已知问题

介绍在将使用 eMbedded Visual C++ 3.0 和 4.0 编写的项目迁移到 Visual Studio 2005 时要查找的内容。

桌面项目的设备支持

介绍如何将桌面 C++ 项目设置为以设备为目标。

在本机设备项目中面向多个平台

介绍使单个项目以多个平台为目标的两种方法。

在本机设备项目中进行资源编辑

介绍设备的资源编辑器与桌面的资源编辑器之间的相似点。

创建本机设备项目后的限制

介绍在创建项目后添加其他平台的限制。

如何：向本机设备项目添加数据库

介绍如何使用 Visual C++ 开发 SQL Server Mobile 或 SQL Server Compact Edition 设备应用程序。

相关章节

使用 Visual C++ 进行设备编程

如何：创建新的 Visual C++ 设备项目

从 eVC 移植所带来的已知问题

开发 Visual C++ 设备项目

生成并调试 Visual C++ 设备项目

如何：创建新的 Visual C++ 设备项目

创建设备项目与创建桌面项目之间有几处显著的不同：

- 设备仅支持桌面项目模板中的一部分。执行下面的过程时，可以看到这些模板。
- 在您选择创建的项目的应用程序向导中，需要从向导的“平台”页上提供的列表中选择一个或多个目标平台。此平台列表是基于您当前安装的 SDK 生成的。

> **注意**
>
> 如果编译器发出有关定义 `CE_ALLOW_SINGLE_THREADED_OBJECTS_IN` 的警告，则应定义 `_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA`，并在 stdafx.h 中定义该标志。为此，请打开 stdafx.h 头文件，在文件顶部附近添加 `#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA` 。

创建 Visual C++ 设备项目

1. 在“文件”菜单上指向“新建”，然后单击“项目”。
2. 在“项目类型”窗格中，展开“Visual C++”，然后单击“智能设备”。“模板”窗格显示设备支持的所有项目模板。
3. 单击要使用的项目模板。
4. 在“名称”框中键入项目的名称，然后单击“确定”。
5. 使用应用程序向导完成创建项目的过程。

请参见

任务

使用应用程序向导创建项目

其他资源

创建并移植 Visual C++ 设备项目

eMbedded Visual C++ 到 Visual Studio 2005 升级向导

Visual Studio 2005 提供了升级向导，可以将 eMbedded Visual C++ 3.0 和 eMbedded Visual C++ 4.0 项目迁移到 Visual Studio 2005。

该升级向导可：

- 创建一个 Visual Studio 2005 解决方案和项目，该解决方案或项目包含从 eMbedded VC++ 迁移的源代码、头文件和资源。
- 添加 MFC DLL 以部署迁移的 MFC 项目的列表。
- 迁移项目设置，如编译器开关。
- 将 eVC 支持但 Visual Studio 2005 不支持的任何体系结构映射到 Visual Studio 2005 支持的体系结构中。

使用 eVC 到 Visual Studio 2005 的升级向导

使用该升级向导将 eVC 项目迁移到 Visual Studio 2005 中

1. 在“文件”菜单上单击“打开”，然后单击“项目/解决方案”。
2. 定位到 eVC 项目所在的目录。如果 eVC 工作区仅包含一个项目，则可以选择 .vcw 或 .vcp 文件；如果 eVC 工作区包含多个项目并且您想要迁移所有项目，请选择 .vcw 文件。
3. 单击“确定”。

注意
迁移向导执行就地迁移过程，例如不会创建源代码副本，而只创建 Visual Studio 2005 项目。由于迁移而创建的 Visual Studio 2005 项目将包括原始 eVC 项目所包括的相同源文件。

映射体系结构

受 eMbedded Visual C++ 支持的一些设备体系结构在 Visual Studio 2005 中不再被支持。这是因为 Visual Studio 2005 所针对的较新平台支持更新的体系结构。幸运的是，所有旧的体系结构都可以映射到较新的设备体系结构。升级向导可自动为您执行此映射。下表阐释了 eMbedded Visual C++ 支持的设备体系结构与 Visual Studio 2005 支持的设备体系结构：

eVC 体系结构	兼容的 Visual Studio 2005 体系结构
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
仿真程序	X86
X86	X86

当使用向导升级 eVC 项目时，在 Visual Studio 2005 中创建的新项目将面向支持新项目中的体系结构的所有已安装 SDK。迁移后的体系结构从其中一种 eVC 体系结构中继承其设置。下表阐释了 eMbedded Visual C++ 支持的设备体系结构与 Visual Studio 2005 支持的设备体系结构之间的映射关系。

原始结构	映射到	说明
非 ARM/ARMV4/ARMV4I	请参见"映射体系结构"中的表	
ARM（不包括 ARMV4i）	ARMV4 和 ARMV4i	ARMV4i 配置设置继承自 eVC 中的 ARM 配置。
ARMV4（不包括 ARMV4i）	ARMV4 和 ARMV4i	ARMV4i 配置设置继承自 eVC 中的 ARMV4 配置。
ARM/ARMV4 和 ARMV4i	ARMV4 和 ARMV4i	ARMV4i 配置设置继承自 eVC 中的 ARMV4i 配置。

默认情况下，Embedded Visual C++ 4.0 版会将 MFC Pocket PC 应用程序的对话框样式设置为 `DS_MODALFRAME`。MFC 8.0 不支持此样式。

注意

如果收到一条错误消息指出"没有与此项目文件的原始平台匹配的可用平台"，则您可能需要安装配置原始项目所用的兼容版本的 SDK。

请参见

参考

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

从 eVC 移植所带来的已知问题

从 eVC 移植所带来的已知问题

可以借助多种 C++ 工具和资源，将现有的 Embedded Visual C++ 项目转换为 Visual Studio 2005。有关更多信息，请参见 eMbedded Visual C++ 到 Visual Studio 2005 升级向导。

自 eVC 以来，活动模板库 (ATL)、Microsoft 基础类 (MFC) 和标准 C++ 库已更新和更改。某些类不再受支持。请参见从 MFC 3.0 升级到 8.0 后不受支持的 eVC 类的列表。调用这些类的代码需要先进行修改，然后才能在 Visual Studio 2005 中编译。从 eVC 移植时通常会出现下列问题。

问题	说明/解决方法
对于 CE 3.0 以上的设备，不会在 MFC 中调用 **CCeSocket::OnReceive()** 方法。	有关详细解决方案，请访问以下网址：http://support.microsoft.com/default.aspx?scid=kb;zh-cn;253945。
不支持 CArchive Class 类。	许多 eVC 项目包含对 **CArchive Class** 类的引用。要解决此问题，需要移除对 **CArchive** 的引用。

某些集合类（包括 CObArray、CMapPtrToPtr 等等）是在 CE 5.0 中使用 CArray<>、CMap<> 等的模板化版本实现的。在 Embedded Visual C++ 4.0 版和桌面 C++ 库中，这些类型是作为常规、非模板化的类实现的。因此，对这些模板化的类调用 IMPLEMENT_SERIAL 会导致编译错误：
错误 C2039:“classCObArray”: 不是“CArray<TYPE,ARG_TYPE>”的成员。
错误 C2065:“classCObArray”: 未声明的标识符
	若要解决实现中的此差异，请更改 IMPLEMENT_SERIAL 宏以使用 CObject，而不是 CObArray、CMapPtrToPtr 等等。
	也就是说，请不要编写以下代码：
	`IMPLEMENT_SERIAL(CYourClass, CObArray, 0)`
	而使用以下代码：
	`IMPLEMENT_SERIAL(CYourClass, CObject, 0)`

默认情况下，Embedded Visual C++ 4.0 版会将 MFC Pocket PC 应用程序的对话框样式设置为 **DS_MODALFRAME**。MFC 8.0 不支持此样式。

示例

此节简要介绍一些更常见的错误，在将项目从 eMbedded Visual C++ 迁移到 Visual Studio 2005 时，可能会遇到这些错误。有关更多信息，请参见 Migrating Microsoft eMbedded Visual C++ Projects to Visual Studio 2005（将 Microsoft eMbedded Visual C++ 项目迁移到 Visual Studio 2005）。

- 编译错误：无法打开包含文件“wceres.rc”

  右击项目资源 (RC) 文件，选择“查看代码”，然后注释掉以下行：

  ```
  //#include "wceres.rc"
  ```

- 未定义 NUM_TOOL_TIP

  在头文件中，为 Pocket PC 配置定义 `#define _WIN32_WCE_PSPC`，为 Smartphone 配置定义 `_WIN32_WCE_WFSP`。

- 无法打开 OLDNAMES.lib 文件

  在“解决方案资源管理器”中，右击项目文件，然后单击“属性”。

  单击“链接器”。编辑“忽略特定库”属性，方法是添加 **OLDNAMES.LIB**。

- 重载不明确

  标准 C++ 库 (SCL) 和 ATL 具有同时存在于设备 SDK 中的 API。使用诸如 **::** 的命名空间消除歧义性。

- 模块计算机类型“THUMB”与目标计算机类型“ARM”冲突

  在“解决方案资源管理器”中，右击项目文件，然后选择“属性”。

  在“配置属性”下，展开“链接器”，然后单击“命令行”属性。对于每个 Windows Mobile 5.0 配置，在“属性”页中从命令行移除 **/MACHINE:THUMB** 开关。

- 资源字符串未正确分隔

  您可能遇到未正确分隔来自所移植应用程序的资源字符串的问题。在“解决方案资源管理器”中，右击项目文件，然后单击“属性”。在“配置属性”下，展开“资源”，然后选择“命令行”属性。将 **-n** 开关添加到资源编译器命令行中。

- 对话框中需要 BEGIN 错误

  发生该错误后通常会发生“未找到文件”这类错误，例如，“找不到文件: 0x1”。请转至错误所指示的 RC 文件，然后编辑代码以便在 **FONT** 声明前后使用 `#ifdef` 语句，如下面的代码示例所示。

  原来的代码：

```
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
```

修改后的代码:

```
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
```

请参见

概念

eMbedded Visual C++ 到 Visual Studio 2005 升级向导

其他资源

Windows Mobile Platform Migration FAQ for Developers(Windows Mobile 平台迁移开发人员常见问题)

桌面项目的设备支持

Visual Studio 2005 支持一个项目面向多个平台，其中包括桌面平台以及 Pocket PC 和 Smartphone 设备等智能设备平台。可以使用“设备项目的 C++ 类向导”，在桌面项目所在的解决方案中创建设备项目。有关更多信息，请参见设备项目的 C++ 类向导。然后，可以使用下列资源，将桌面应用程序植入新创建的设备项目中：

- 将代码向导用于设备项目
- 如何：创建新的 Visual C++ 设备项目
- 在本机设备项目中面向多个平台
- 在本机设备项目中进行资源编辑
- 生成并调试 Visual C++ 设备项目
- 设备项目的资源编辑器
- 调试设备项目
- 打包设备解决方案以便进行部署
- 使用多个平台上的资源 .

请参见

参考

用于设备应用程序开发的 Visual Basic 语言参考

概念

设备项目的 .NET Compact Framework 引用
设备的标准 C++ 库参考
针对设备的 C 运行时库参考

其他资源

设备的 ATL 引用
设备的 MFC 参考

在本机设备项目中面向多个平台

使用 Visual Studio 2005，您可以创建一个面向多个设备平台（如 Pocket PC 2003 和 Smartphone 2003）的设备项目。

可以有两种方式设置设备项目，使其面向多个平台：

- 在项目创建时使用应用程序向导。这是较简单的方法。有关更多信息，请参见如何：使用向导创建多平台设备项目。
- 在创建项目之后。有关更多信息，请参见如何：向设备项目添加新平台。

如果在项目应用程序向导的“平台”页上选择多个平台，则会为您的每个平台生成并配置一个资源文件。但是，如果在项目创建后添加平台，则需要手动添加平台和资源文件。有关更多信息，请参见使用多个平台上的资源。

请参见

任务

使用应用程序向导创建项目

概念

MFC for Windows CE 向导 (C++)

其他资源

创建并移植 Visual C++ 设备项目

在本机设备项目中进行资源编辑

由于 Visual Studio 2005 具有多平台特性，因此会为您选择要包含的每个平台（例如，Pocket PC 和 Smartphone）生成一个单独的资源文件。设备项目的资源编辑器几乎与桌面项目的资源编辑器完全相同。每个编辑器均受支持，并且只有对话框编辑器具有显著的不同。有关更多信息，请参见使用设备项目的资源编辑器和资源编辑器。

在创建项目时可以面向所有平台；因此，您可以轻松地根据设备格式的因素自定义应用程序的用户界面，并且可以为多平台应用程序维护一个项目。在演练：为智能设备创建多平台 MFC 应用程序中创建的示例项目同时具有 Pocket PC 资源文件和 Smartphone 资源文件。请注意，Smartphone 资源文件上有一个“无生成”图标，因为项目的当前活动配置为 Pocket PC 2003 (Armv4)。可以使用“智能设备项目向导”的“配置管理器设置”将活动配置更改为 Smartphone 2003 (ArmV4)。这将使“无生成”图标从 Smartphone 2003 资源文件上消失，而改为出现在 Pocket PC 2003 (Armv4) 资源文件上。

请参见

其他资源

创建并移植 Visual C++ 设备项目

创建本机设备项目后的限制

本节列出了与桌面的本机应用程序相比，应用于智能设备的本机 C++ 应用程序的限制。

桌面项目和设备本机项目的区别

有关创建 C++ 设备项目的更多信息，请参见设备项目的 C++ 类向导。

- 有关"帮助"主题和参考的筛选，请参见如何：优化智能设备开发帮助。
- 有关项目之间的用户界面区别，请参见设备的用户界面参考。
- 有关 ATL 设备项目开发，请参见设备的 ATL 和标准 ATL 之间的差异。
- 有关 MFC 设备项目开发，请参见设备的 MFC C++ 和标准 MFC 之间的差异和设备类的唯一 MFC。
- 有关 C++ 设备开发，请参见设备的标准 C++ 库参考。
- 有关对设备项目使用资源，请参见使用多个平台上的资源。
- 有关编译器开关，请参见用于智能设备的编译器。
- 有关设备项目调试，请参见调试设备项目。
- 有关设备项目打包，请参见打包设备解决方案以便进行部署。

请参见

任务

如何：查找有关设备支持的 ATL 类和方法的帮助

如何：查找有关设备支持的 MFC 类和方法的帮助

概念

eMbedded Visual C++ 到 Visual Studio 2005 升级向导

其他资源

创建并移植 Visual C++ 设备项目

如何：向本机设备项目添加数据库

本主题已针对 Visual Studio 2005 SP1 进行了更新。

通过使用 Visual Studio 开发环境，可以开发使用 SQL Server Mobile Edition 或 SQL Server Compact Edition 的智能设备应用程序。

有关使用针对设备的 Visual C++ 开发 SQL Server Mobile Edition 或 SQL Server Compact Edition 应用程序的说明，请参见Installing a Development Environment。

如果在本机项目中使用 SQL Server Mobile Edition 或 SQL Server Compact Edition，则部署项目时安装 CAB 文件不会自动下载到设备上。可以将 CAB 文件添加到您的项目中。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

将 SQL Server Mobile Edition 或 SQL Server Compact Edition CAB 文件添加到您的项目中

1. 右击项目，然后在“项目”快捷菜单上单击“添加现有项”。
2. 定位到 CAB 文件。

   默认情况下，CAB 文件位于 %Program Files%\Microsoft Visual Studio 8\SmartDevices\SDK\SQL Server\Mobile\v3.0\wce500\<设备处理器>\。

   - 或 -

3. 将 CAB 文件复制到智能设备。

   默认情况下，CAB 文件位于 %Program Files%\Microsoft Visual Studio 8\SmartDevices\SDK\SQL Server\Mobile\v3.0\wce500\<设备处理器>\。

4. 通过在设备上运行 CAB 文件来安装 SQL Server Mobile Edition 或 SQL Server Compact Edition。

请参见

其他资源

创建并移植 Visual C++ 设备项目

开发 Visual C++ 设备项目

本部分包含的主题介绍了开发设备应用程序特有的信息。

本节内容

在设备项目中切换平台

描述如何使用相同的项目切换目标。

设备项目的 C++ 类向导

提供有关如何使用面向设备项目的 C++ 向导的信息。

设备项目的资源编辑器

提供有关面向设备项目的资源编辑器的信息。

用于智能设备的编译器

描述面向智能设备中使用的微处理器的编译器。

代码解释：Visual C++ 设备项目中由向导生成的代码

描述设备项目中由 Visual C++ 向导生成的代码。

如何：指定主项目输出的远程路径

描述如何指定项目在目标设备上的部署位置。

如何：更改默认设备（本机项目）

描述如何更改本机项目的默认目标。

请参见

参考

欢迎使用 Windows Mobile

其他资源

使用 Visual C++ 进行设备编程

在设备项目中切换平台

您可以轻松在目标平台之间进行切换。例如，使用相同项目，分别将 Pocket PC 2003 平台或 Smartphone 平台作为目标。

若要在 Visual C++ 中创建以多个平台为目标的项目，请参见：

- 在本机设备项目中面向多个平台
- “智能设备项目向导”的“配置管理器设置”.

有关在配置管理器中设置目标平台的信息，请参见：

- 如何：创建多平台设备项目 (Visual C++)
- “智能设备项目向导”的“配置管理器设置”.

有关在 Visual C++ 项目中使用多个平台上的资源的其他信息，请参见使用多个平台上的资源。

请参见

概念

eMbedded Visual C++ 到 Visual Studio 2005 升级向导

从 eVC 移植所带来的已知问题

其他资源

智能设备开发

设备项目的 C++ 类向导

Visual C++ 设备项目支持桌面 Visual C++ 项目所支持的类向导的子集。由于 Windows 与 Windows CE 平台之间存在区别，因此设备项目不支持某些向导。有关更多信息，请参见用代码向导添加功能。

本节内容

将代码向导用于设备项目

介绍哪些 C++ 类向导受支持，以及如何访问这些向导。

本机设备项目中的向导选项

提供一些主题的链接，这些主题介绍特定 C++ 类向导中不受支持的向导选项。

"项目属性"对话框中不受支持的选项

描述"项目属性"对话框中与桌面项目不同的行为。

不是所有智能设备本机应用程序向导都允许您同时选择静态链接和动态链接。下表概述了智能设备应用程序向导在运行库链接方面的行为：

向导	说明
Win32 智能设备项目 – Windows 应用程序	静态链接。创建项目时不提供动态/静态链接选项
Win32 智能设备项目 – 控制台应用程序	静态链接。创建项目时不提供动态/静态链接选项
Win32 智能设备项目 - DLL	静态链接。创建项目时不提供动态/静态链接选项
Win32 智能设备项目 – 静态库	静态链接。创建项目时不提供动态/静态链接选项
ATL 智能设备项目 - DLL	静态链接。创建项目时不提供动态/静态链接选项
ATL 智能设备项目 - EXE	静态链接。创建项目时不提供动态/静态链接选项
MFC 智能设备应用程序 - SDI	静态链接。创建项目时不提供动态/静态链接选项
MFC 智能设备应用程序 – SDI w. **DocList**	静态链接。创建项目时不提供动态/静态链接选项
MFC 智能设备应用程序 - 基于对话框	静态链接。创建项目时不提供动态/静态链接选项
MFC 智能设备 DLL – 常规 DLL	静态链接。创建项目时不提供动态/静态链接选项
MFC 智能设备 ActiveX 控件	静态链接。创建项目时不提供动态/静态链接选项
MFC 智能设备 DLL – 扩展 DLL	"动态"链接。创建项目时不提供动态/静态链接选项

上表使用 F5 快捷键指代部署。应用程序的安装将在下面进行介绍：

- 当为使用 C++ 编写的应用程序创建"智能设备 CAB"项目时，如果要动态链接到这些 DLL，必须手动向 CAB 项目添加所有依赖项，如 **atl80.dll**、**mfc80U.dll** 和/或 **msvcrt.dll**。如果采用的是动态链接，并需要在 cab 中重新发布 DLL，请不要将 DLL 安装到设备上的系统目录（如"\windows"）中，而应将 DLL 安装到本地应用程序目录中。如果重新发布的应用程序套件中的所有应用程序都动态链接到 ATL/MFC 运行库，建议您将所有应用程序和运行库 DLL 安装到单个应用程序目录中，并提供应用程序快捷方式（这些快捷方式可以放在应用程序自己的文件夹中）。这样有助于节省空间大小，并避免了系统目录中的 DLL 以后被应用程序的另一个安装所替换，从而破坏动态链接到 DLL 的应用程序的危险性。
- 为了减少 MFC/ATL DLL 的依赖项，强烈建议您使用静态链接。如果采用的是静态链接，则不应随同应用程序一起重新发布 DLL。

请参见

其他资源

开发 Visual C++ 设备项目

将代码向导用于设备项目

设备项目可以利用 C++ 桌面项目使用的多个相同代码向导。其中的部分向导不受支持，部分向导受支持但具有不受支持的选项。有关特定代码向导的不受支持选项的更多信息，请参见本机设备项目中的向导选项。

支持的代码向导

Visual C++ 设备项目支持下列向导：

- ATL 简单对象向导。有关此向导中不支持的选项的更多信息，请参见 ATL 简单对象向导的选项页中的向导选项。
- ATL 控件向导。有关此向导中不支持的选项的更多信息，请参见 ATL 控件向导的“选项”中的向导选项、ATL 控件向导的“界面”中的向导选项和 ATL 控件向导的常用属性页中不受支持的向导选项。
- ATL 对话框向导 .
- 向 MFC 项目添加 ATL 支持
- ATL 属性页向导。有关此向导中不支持的选项的更多信息，请参见 ATL 属性页向导的选项页中不受支持的向导选项。
- 一般 C++ 类向导
- MFC 类向导。有关此向导中不支持的选项的更多信息，请参见 MFC 类向导中不受支持的向导选项。

访问 Visual C++ 智能设备代码向导

Visual C++ 智能设备代码向导位于“添加类”对话框对话框中。可以通过下面的方法访问“添加类”对话框：

- 在“项目”菜单上单击“添加类”。
- 在解决方案资源管理器中，右击任意文件夹，单击“添加”，然后单击“类”。
- 在类视图窗口中，右击适当的节点，单击“添加”，然后单击“类”。

若要查看可用于智能设备项目的向导，请在“添加类”对话框的“类别”窗格中展开目录结构，再单击“智能设备”。可用模板将显示在“模板”窗格中。

请参见

其他资源

设备项目的 C++ 类向导

本机设备项目中的向导选项

由于设备项目与桌面项目之间存在差异，因此，一些 C++ 类向导所包含的选项在设备上不受支持，或者在设备项目中具有不同的行为。

下面的主题介绍在当前项目面向设备时不受支持的向导选项。

本节内容

ATL 简单对象向导的选项页中的向导选项

介绍 ATL 简单对象向导的“选项”页中不受支持的向导选项。

ATL 控件向导的“选项”中的向导选项

介绍 ATL 控件向导的“选项”页中不受支持的向导选项。

ATL 控件向导的“界面”中的向导选项

介绍 ATL 控件向导的“接口”页中不受支持的向导选项。

不受支持的用于外观的向导选项 - ATL 控件向导（设备）

介绍 ATL 控件向导的“外观”页中不受支持的向导选项。

ATL 控件向导的常用属性页中不受支持的向导选项

介绍 ATL 控件向导的“常用属性”页中不受支持的向导选项。

ATL 属性页向导的选项页中不受支持的向导选项

介绍 ATL 属性页向导的“选项”页中不受支持的向导选项。

MFC 类向导中不受支持的向导选项

介绍 MFC 类向导中不受支持的向导选项。

ATL 智能设备项目向导的“应用程序设置”页中不支持的向导选项

介绍 ATL 智能设备向导的“应用程序设置”页中不受支持的向导选项。

请参见

其他资源

设备项目的 C++ 类向导

ATL 简单对象向导的选项页中的向导选项

描述在“ATL 简单对象向导”的“选项”页中，对于智能设备不受支持的向导选项。此向导页上的某些元素要么在设备上不受支持，要么在设备项目中具有不同的行为。

不支持的选项和支持方式不同的选项

下表描述了在设备项目上具有不同行为的元素。

节	行为
线程模型	线程模型设置取决于目标平台是否支持 DCOM。如果目标平台不支持 DCOM，则无论用户选择何种线程模型，生成的代码中使用的线程模型始终为“Free”。 Windows Mobile 2003 软件不支持 DCOM。 有关 Windows CE 上的 COM、DCOM 和线程模型的更多信息，请参见 Windows CE 5.0 文档中的组件服务（COM 和 DCOM）和 COM 线程和进程。
接口	“自动化兼容”复选框在设备项目中不受支持。
支持	“IObjectWithSite（IE 对象支持）”复选框在设备项目中不受支持。

请参见

其他资源

设备项目的 C++ 类向导

ATL 控件向导的“选项”中的向导选项

此向导页上的某些元素要么在设备上不受支持，要么在设备项目中具有不同的行为。

不支持的选项和支持方式不同的选项

下表描述了在设备项目上具有不同行为的元素。

节	行为
线程模型	线程模型设置将取决于目标平台是否支持 DCOM。如果目标平台不支持 DCOM，则无论用户选择何种线程模型，生成的代码中使用的线程模型始终为“Free”。 Windows Mobile 2003 软件不支持 DCOM。 有关 Windows CE 上的 COM、DCOM 和线程模型的更多信息，请参见 Windows CE 5.0 文档中的组件服务（COM 和 DCOM）和 COM 线程和进程。
控件类型	设备项目不支持“DHTML 控件”选项。
支持	设备项目不支持“已授权”复选框。

请参见

其他资源

设备项目的 C++ 类向导

ATL 控件向导的“界面”中的向导选项

介绍 ATL 控件向导的“界面”页上智能设备不支持的界面。

此向导页上的某些界面要么在设备上不受支持，要么在设备项目中具有不同的行为。

不支持的选项和支持方式不同的选项

下表描述了在设备项目中具有不同行为的界面。

界面	行为
IDataObject	Windows CE 不支持此界面。
IPersistStorage	只有支持 DCOM 的平台支持此界面。 Windows Mobile™ 2003 软件不支持 DCOM。 有关 Windows CE 上的 COM 和 DCOM 的更多信息，请参见 Windows CE 5.0 文档中的组件服务（COM 和 DCOM）。

请参见

其他资源

设备项目的 C++ 类向导

不受支持的用于外观的向导选项 - ATL 控件向导(设备)

此向导页上的一些元素不是完全受设备项目支持的。

支持方式不同的选项

下表介绍在各个设备项目中受支持情况不同的元素。

节	差异
添加的控件基于	只有以下项受设备平台支持: • Button • ComboBox • 编辑 • ListBox • Scrollbar • 静态 • RichEdit(只在某些 Windows CE 平台上受支持，而不是在所有 Windows CE 平台上都受支持)

请参见

其他资源

本机设备项目中的向导选项

ATL 控件向导的常用属性页中不受支持的向导选项

由于缺少 GUID CLSID_StockPicturePage、CLSID_StockColorPage 和 CLSID_StockFontPage，因此该向导页上的某些常用属性在设备上不受支持。

不支持的属性

下列常用属性在设备上不受支持：

- BackColor
- BorderColor
- FillColor
- Font
- ForeColor
- MouseIcon
- Picture

请参见

其他资源

设备项目的 C++ 类向导

ATL 属性页向导的选项页中不受支持的向导选项

此向导页上的某些元素要么在设备上不受支持，要么在设备项目中具有不同的行为。

下表描述了在设备项目上具有不同行为的元素。

节	行为
线程模型	线程模型设置取决于目标平台是否支持 DCOM。如果目标平台不支持 DCOM，则无论用户选择何种线程模型，生成的代码中使用的线程模型始终为“Free”。 Windows Mobile 2003 软件不支持 DCOM。 有关 Windows CE 上的 COM、DCOM 和线程模型的更多信息，请参见 Windows CE 5.0 文档中的组件服务（COM 和 DCOM）和 COM 线程和进程。
界面	“自动化兼容”复选框在设备项目中不受支持。
支持	“IObjectWithSite”（IE 对象支持）复选框在设备项目中不受支持。

请参见

其他资源

设备项目的 C++ 类向导

MFC 类向导中不受支持的向导选项

此向导页上的某些元素要么在设备上不受支持，要么在设备项目中具有不同的行为。

不受支持的选项

设备项目不支持下面的元素：

- Active Accessibility
- DHTML 资源 ID
- .HTM 文件
- 自动化
- 生成 DocTemplate 资源
- 文档模板字符串

请参见

其他资源

设备项目的 C++ 类向导

ATL 智能设备项目向导的“应用程序设置”页中不支持的向导选项

介绍 ATL 智能设备项目向导的“应用程序设置”页中智能设备不支持的向导选项。

此向导页上的某些元素要么在设备上不受支持，要么在设备项目中具有不同的行为。

不受支持的选项

下表描述了在设备项目中具有不同行为的元素。

节	行为
（无）	设备项目不支持“属性化”复选框。

ATL 智能设备项目向导不实现类型库注销

由于 Windows Mobile 不实现从注册表移除类型库的 COM 功能，因此 ATL 智能设备项目向导会生成以不同的方式实现 `DllUnregisterServer` 函数的代码：

```
// DllUnregisterServer - Removes entries from the system registry
STDAPI DllUnregisterServer(void)
{
    HRESULT hr = _AtlModule.DllUnregisterServer(false);
    return hr;
}
```

将 **false** 传递给 `DllUnregisterServer` 函数会指示 COM 不注销类型库。如果将该参数更改为 **true**，则对 DllUnregisterServer 的所有调用都将失败，并会产生 `E_NOTIMPL`。

请参见

其他资源

本机设备项目中的向导选项

“项目属性”对话框中不受支持的选项

“项目属性”对话框上的某些元素要么在设备上不受支持，要么在设备项目中具有不同的行为。

不支持的选项和支持方式不同的选项

下表描述了在设备项目中的行为与在桌面 PC 上的行为不同的元素。

节	行为
项目默认值	尽管在“ATL 的使用”设置为“动态链接到 ATL”的同时，将“MFC 的使用”设置为“在静态库中使用 MFC”是可能的，但此方案在设备项目中不受支持。
项目默认值	**注意** 此项仅适用于 MFC 智能设备应用程序项目和 MFC 智能设备 DLL 项目。 如果将“MFC 的使用”从“在共享 DLL 中使用 MFC”(默认)更改为“在静态库中使用 MFC”，则还应该将 **mfc80ud.dll**(调试)和/或 **mfc80u.dll**(发布)从“部署”属性页“常规”部分的“附加文件”中移除。 同样，如果将“MFC 的使用”从“在静态库中使用 MFC”更改为“在共享 DLL 中使用 MFC”，则还应该将 **mfc80ud.dll**(调试)和/或 **mfc80u.dll**(发布)添加到“部署”属性页“常规”部分的“附加文件”中。 添加 **mfc80u.dll** 或 **mfc80ud.dll** 时，还应当添加 **atl80.dll** 以及 **msvcr80.dll** 或 **msvcr80d.dll**。

请参见

参考

属性页 (C++)

其他资源

本机设备项目中的向导选项

设备项目的资源编辑器

对设备项目使用资源编辑器与对桌面 Visual C++ 项目使用资源编辑器类似。本节详细说明桌面资源编辑器与设备资源编辑器之间的一些差别。有关更多信息，请参见资源编辑器。

设备 SDK 可以定义它们自己的用户界面 (UI) 模型，该模型可用于筛选对话框编辑器中显示的控件列表，从而仅显示该平台所支持的控件。Visual Studio 2005 附带包括 Windows Mobile 2003 SDK 在内的 Windows CE UI 模型。

本节内容

使用设备项目的资源编辑器

概要介绍如何将资源编辑器与设备项目结合使用。

使用多个平台上的资源

解释如何在面向多个平台的项目中使用资源。

设备对话框控件

说明设备项目支持的对话框控件。

请参见

其他资源

开发 Visual C++ 设备项目

使用设备项目的资源编辑器

设备项目的资源编辑器几乎与桌面项目的资源编辑器完全相同。每个编辑器均受支持，并且只有对话框编辑器具有显著的不同。有关更多信息，请参见资源编辑器。

Visual Studio 2005 中的本机智能设备项目支持下列资源类型：

- 快捷键
- 位图
- 光标
- 对话框
- 图标
- 菜单
- 注册表
- 字符串表
- 工具栏
- 版本

对话框编辑器

设备对话框编辑器与桌面对话框编辑器在以下方面具有不同：

- 桌面控件比设备控件少，且设备支持的控件与对应的桌面控件在属性上稍有不同。有关更多信息，请参见设备对话框控件。
- 新增了一些对话框模板，用于常见的设备窗体外观设置。
- 对话框控件的行为和属性来源于安装的每个软件开发工具包 (SDK) 附带的用户界面 (UI) 模型。该 UI 模型提供了一组正确的控件，这些控件用于当前面向的平台。如果 SDK 未定义 UI 模型，则对话框编辑器默认使用 Windows CE UI 模型。
- 有两个控件是设备项目所特有的："输入面板的状态"控件 和 CAPEdit 控件。

RC2 文件

某些应用程序向导除了生成标准的资源文件 (.RC) 外，还生成一个 .RC2 资源文件。该 .RC2 文件不是供资源编译器编译的；它实际上包含资源编译器并不处理的资源。例如 HI_RES_AWARE 自定义资源和菜单资源数据 (RCDATA) 就是这样的资源。对于您不希望资源编译器为您进行编辑的其他自定义资源，.RC2 文件是理想的放置地点。

有关如何为 Smartphone 创建菜单资源的更多信息，请参见"How to: Create a Soft Key Bar"（如何：创建软键栏）。若要创建 Smartphone 菜单，请确保您具有 RCDATA 节。通常，该节在 .RC2 文件中。资源 ID 的值应大于或等于 100。这些 ID 是在资源头文件（对于 Smartphone，为 resourcesp.h）中设置的。按钮应将 NOMENU 作为它们的索引 (IDR_MENU RCDATA)。下面的示例阐释了这一点：

```
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
```

使用用于设备的资源编辑器时，您可能会收到由于下列原因产生的错误：

- 您编辑的 RESX 项属于另一个项目项，如窗体或用户控件。
- Windows 窗体设计器自动丢弃了任何不链接到控件的项。它还移除所有注释，不支持链接的项，并且无法加载窗体或用户控件（如果已经向资源编辑器中的 RESX 文件添加了一个窗体或用户控件）。

- Windows CE 不支持某些资源类型，如 Tiff 文件。
- 还由于资源文件的格式不受支持，文件为空或者格式已损坏而生成了错误。

请参见

其他资源

设备项目的资源编辑器

使用多个平台上的资源

Visual Studio 2005 允许一个设备项目面向多个平台，如 Pocket PC 和 Smartphone。由于平台之间在用户界面 (UI) 上的差异，因此每个平台都需要在项目中具有自己的资源脚本文件 (.rc)。

多个资源文件

有两种将设备项目设置为面向多个平台的方式：

- 在创建项目时使用应用程序向导。
- 创建项目之后。

如果在项目应用程序向导的“平台”页上选择了多个平台，将为每一个平台生成并配置一个资源文件。例如，如果选择 Pocket PC 和 Smartphone 作为目标平台，那么在面向 Smartphone 平台进行生成时将排除 Pocket PC 资源文件，在面向 Pocket PC 平台进行生成时将排除 Smartphone 资源文件。

但是，如果在创建项目后添加平台，则需要手动添加平台和资源文件。

添加新平台

添加新平台

1. 在“生成”菜单上单击“配置管理器”。
2. 在“活动解决方案平台”框中，单击“<新建...>”。
3. 选择要添加到项目的平台，再选择要从中复制设置的平台，然后单击“确定”。

   注意

   如果从“<默认>”复制设置，则该平台的项目属性将为空。建议从类似的平台复制设置，然后根据需要更改项目属性。例如，如果要添加 Smartphone 作为平台，则应从 Pocket PC 平台复制设置。

4. 单击“关闭”。

添加新的资源文件

现在，您有一个新的平台，您需要为该平台添加资源文件。

为新平台添加资源文件

1. 在“项目”菜单上单击“添加新项”。
2. 在“添加新项”对话框中，单击“资源”，然后在“模板”窗格中，单击“资源文件 (.rc)”。
3. 在“名称”框中，键入文件的名称，然后单击“添加”。

此时，向您的项目添加一个新的头文件 (.h)，该文件与新的资源脚本文件 (.rc) 对应。

从生成中排除资源文件

针对目标平台生成项目时，您不会希望包含来自其他平台的资源文件。可以基于目标平台从生成中排除文件。

从生成中排除资源文件

1. 右击资源脚本文件 (.rc)，然后单击“属性”。
2. 在“平台”框中，选择列表中的第一个平台。
3. 如果希望在面向选定平台生成项目时不包括该 **.rc** 文件，请在“常规”属性页上的“从生成中排除”框中选择“是”。
4. 对每个平台配置重复上一步骤，确保仅排除不属于当前选定平台的资源文件。
5. 对项目中的每个 .rc 文件重复前面的所有步骤 (1-4)。

在“解决方案资源管理器”中，您会注意到要从面向当前选定平台的生成中排除的每个文件的图标上都有一个红色标记。

更改新平台配置的项目属性

现在，资源文件已经为您的平台设置好了，接下来您需要确保项目属性对于您的新平台配置而言是正确的。如果设置是从类似平台中复制的，您可能没有太多要更改的配置，但如果您选择的是“<默认>”，则必须手动添加所有设置。对于本示例，您可以假设向项目中添加了一个新的“Smartphone 2003 (ARMV4)”平台，而设置是从“Pocket PC 2003 (ARMV4)”平台复制的。

更改项目属性

1. 在“项目”菜单上单击“属性”。
2. 展开“C/C++”节点，然后单击“预处理器”。
3. 在“预处理器定义”框中，将 **POCKETPC2003_UI_MODEL** 更改为 **SMARTPHONE2003_UI_MODEL**，然后单击“确定”。

注意
如果您添加了一个不同的平台，或者从一个不同的平台复制了设置，则可能需要更改更多设置。

向头文件中添加 #ifdef 指令

项目的主头文件需要检查您在前面的过程中设置的 UI 模型预处理器定义，并且仅包括相应的资源文件。

向头文件中添加 #ifdef 指令

1. 打开 *ProjectName*.h。
2. 在原始平台的 UI 模型的 **#ifdef** 后面，添加以下代码：

```
#ifdef SMARTPHONE2003_UI_MODEL
   #include "ResourceFileName.h"
#endif
```

3.

请参见

其他资源

设备项目的资源编辑器

设备对话框控件

对话框编辑器使您能够以用于 Visual C++ 桌面项目的相同方式为 Visual C++ 设备项目创建或编辑对话框资源。此节描述了两个设备特定的控件以及设备项目的对话框编辑器中支持的公共桌面控件。

本节内容

设备所支持的对话框控件

描述设备同样支持的公共桌面控件。

设备特有的对话框控件

描述设备独有的两个控件。

请参见

其他资源

设备项目的资源编辑器

设备所支持的对话框控件

Visual Studio 2005 设备对话框编辑器支持两个唯一的设备控件，以及桌面项目支持的大部分公共控件。

受支持的控件

设备的对话框编辑器支持以下公共控件：

Button	Vertical Scroll Bar
Check Box	Slider Control
Edit Control	Progress Control
Combo Box	List Control
List Box	Tree Control
Group Box	Tab Control
Radio Button	Date Time Picker
Static Text	Month Calendar
Picture Control	Custom Control
Horizontal Scroll Bar	

若要查看设备上受支持的对话框控件的最新列表，您可以创建一个本机 Win32 .exe 设备项目，双击 RC 文件，再展开“对话框”节点并将其打开。

设备独有的控件

Visual Studio 2005 对话框编辑器还为设备开发添加两个新控件。

控件	说明
CAPEdit 控件	“CAPEdit”控件是编辑控件，将控件中每个单词的首字母变成大写。
“输入面板的状态”控件	“State of Input Panel”控件会引发支持该控件的平台上的输入面板。

请参见

其他资源

设备对话框控件

设备特有的对话框控件

“输入面板的状态”和“CAPEdit 控件”是设备独有的对话框。这些对话框将利用设备独有的特性。

本节内容

CAPEdit 控件

“CAPEdit”控件是编辑控件，它将控件中键入的第一个字母改为大写。

“输入面板的状态”控件

“输入面板的状态控件”显示任何对话框上的输入面板。

请参见

其他资源

设备对话框控件

CAPEdit 控件

除了“CAPEdit”控件中的第一个字母会自动大写之外，该控件的行为与 CEdit Class 控件的行为相似。该控件简化了输入长字符串的过程，它通常在用户需要输入总是大写的信息（如名称和位置）的情况下使用。

请参见

参考

CDialog Class

其他资源

设备特有的对话框控件

创建编辑控件

“输入面板的状态”控件

“输入面板的状态”控件是一个非可视控件，用于显示支持它的平台（如 Pocket PC）上的输入面板。如果对话框中包含该控件，则只要输入控件接收到焦点，就会显示输入面板。当控件失去焦点时，输入面板就会减弱显示。

注意
在一个对话框中添加多个“输入面板的状态”控件会导致应用程序在设备上挂起。此问题有待在将来的版本中得到解决。

请参见

概念

设备所支持的对话框控件

其他资源

设备特有的对话框控件

用于智能设备的编译器

Visual Studio 2005 包括下列面向智能设备中使用的微处理器的编译器：

- 用于编译和链接 32 位 ARM C 和 C++ 程序的 32 位 C/C++ 编译器。
- 用于编译和链接 32 位 Renesas SH-4 C 和 C++ 程序的 32 位 C/C++ 编译器。
- 用于编译和链接 MIPS16、MIPS32、MIPS64 C 和 C++ 程序的 C/C++ 编译器。

这些编译器生成通用对象文件格式对象文件。编译器程序每次编译都编译每个源文件，并且还编译一个对象文件（除非另外指定）。编译器中包含有命令行 (CL)、CL 环境变量以及任何指定的响应文件中列出的选项。有关更多信息，请参见参考主题用于智能设备的编译器。

Visual Studio 2005 桌面编译器与设备编译器之间的差异

差异	说明
“高级”选项卡，“为结构编译”下拉列表。	设备项目在“C/C++”节点下的“项目属性”的“高级”选项卡的“为结构编译”下拉列表的下拉框中有以下选项：“Arm4 (/QRarch4)”、“ARM5 (/QRarch5)”、“Arm4t (/QRarch4t)”、“ARM5t (/QRarch5t)”。
“高级”选项卡，“使 ARM 和 ARM Thumb 交互工作”下拉列表。	设备项目在“C/C++”节点下的“项目属性”的“高级”选项卡的“使 ARM 和 ARM Thumb 交互工作”下拉列表的下拉框中有以下选项：“是(/QRInterwork-return)”和“否”选项。当设置为“是”时，编译器生成形式转换代码，以使 ARM 16 和 32 位代码交互工作。
“高级”选项卡，“启用浮点模拟”下拉列表。	设备项目在“C/C++”节点下的“项目属性”的“高级”选项卡的“启用浮点模拟”下拉列表的下拉框中有以下选项：“是”和“否”选项。当设置为“是”时，编译器启用浮点操作模拟。
“预处理器”选项卡，“预处理器定义”输入框。	设备项目在“C/C++”节点下的“项目属性”的“预处理器”选项卡的“预处理器定义”输入框中有一个“从父级或项目默认设置继承”复选框和一个用于添加宏的“宏”按钮。
“优化”选项卡，“浮点一致性定义”下拉列表。	设备项目在“C/C++”节点下“项目属性”的“优化”选项卡的“浮点一致性定义”下拉列表中选择“默认一致性”或“改善一致性(/Op)”。

有关更多信息，请参见按字母顺序列出的编译器选项。

Visual Studio 2003 和 Visual Studio 2005 使用的编译器不同

由于设备编译器基于桌面计算机 Visual C++ 编译器，因此，了解了桌面编译器的各版本之间的不同，便可以很好地了解 eMbedded Visual C++ 设备编译器与 Visual Studio 2005 设备编译器之间的不同。有关 Visual Studio 6.0 与 Visual Studio 2003 之间的不同的信息，请参见 Standard Compliance Issues in Visual C++（Visual C++ 中的标准符合性问题）。

下表概述了 Visual Studio 2003 与 Visual Studio 2005 使用的编译器的不同点。

问题	说明
指向成员的指针现在要求使用限定名和 address-of 运算符 (**&**)，并且要求调用函数时必须使用括号。	针对仅使用方法名的以前版本的编译器编写的代码现在提供编译器错误 C3867 或编译器警告 C4867。此诊断是 ISO C++ 标准所必需的。要创建指向成员函数的指针，必须使用 address-of 运算符 (**&**) 以及方法的完全限定名。如果不要求 **&** 运算符和方法的完全限定名，或者函数调用中缺少括号，则可能会导致错误。在没有参数列表的情况下使用函数的名称会导致一个函数指针可转换为几种类型。因此，代码可能会在运行时产生意外的行为。
某个类必须可供 **friend** 声明访问。	对于以前的 Visual C++ 编译器而言，如果某个类在包含友元声明的类的范围内不可访问，则编译器将启用该友元声明。在 Visual C++ 2005 中，这些情况会导致编译器生成编译器错误 C2248。为了解除此错误，应更改在友元声明中指定的类的可访问性。此更改是为了符合 ISO C++ 标准。
不允许将显式专用化作为复制构造函数和复制赋值运算符。	依赖于复制构造函数或复制赋值运算符的显式模板专用化的代码现在将生成编译器错误 C2299。ISO C++ 标准禁止这种用法。此更改是出于符合性方面的考虑，其目的是提高代码可移植性。

不能将非专用化的类模板用作基类列表中的模板参数。	在类定义的基类列表中使用非专用化的模板类名将导致编译器错误 C3203。将非专用化的模板类名用作基类列表中的模板参数是无效的。当要将某个模板类型参数用作基类列表中的模板参数时，必须显式地将该模板类型参数添加到模板类名中。此更改是出于符合性方面的考虑，其目的是提高代码可移植性。
不再允许嵌套类型的 using 声明。	具有嵌套类型的 using 声明的代码现在将生成编译器错误 C2885。要解决此问题，需要完全限定对嵌套类型的引用，将类型放入命名空间中，或者创建 `typedef`。此更改是出于符合性方面的考虑，其目的是提高代码可移植性。
/YX 编译器选项已被移除。	**/YX** 编译器选项生成自动的预编译头支持。它是默认从开发环境使用的。如果将 **/YX**编译器选项从生成配置中移除，有助于加快生成的速度。除了性能问题，**/YX** 编译器选项还引入了在运行时产生意外行为的可能性。首选方案是使用 **/Yc**"创建预编译头文件"和 **/Yu**"使用预编译头文件"，在这两种情况下，可以更好地控制预编译头文件的使用方式。
/Oa 和 **/Ow** 编译器选项已被移除。	**/Oa** 和 **/Ow** 编译器选项已被移除，并且将被忽略。请使用 **noalias** 或 **restrictdeclspec** 修饰符来指定编译器命名别名的方式。
/Op 编译器选项已被移除。	**/Op** 编译器选项已被移除。可以改为使用 **/fp:precise**。
/ML 和 **/MLd**编译器选项已被移除。	Visual C++ 2005 不再提供静态链接的单线程 CRT 库支持。可以改为使用 **/MT** 和 **/MTd**。
/G3、**/G4**、**/G5**、**/G6**、**/G7** 和 **/GB**编译器选项已被移除。	编译器现在使用混合模型，以试图创建对于所有结构而言均为最佳的输出文件。
/Gf 编译器选项已被移除。	可以改为使用 **/GF**。**/GF** 将池中的字符串放入只读部分，该部分比 **/Gf** 在其中添加这些字符串的可写部分安全。
/GS 编译器选项现在默认启用。	缓冲区溢出检查功能现在默认为打开状态。可以使用 **/GS-** 关闭缓冲区溢出检查功能。
/Zc:wchar_t 变量现在默认启用。	这是 ISO C++ 标准行为：wchar_t 变量将默认为内置类型，而不是短的无符号整数。当客户端代码与在未使用 **/Zc:wchar_t** 的情况下编译的库链接时，此更改会破坏二进制兼容性。可以使用 **/Zc:wchar_t-** 恢复为原来的非标准行为。引入此更改是为了默认创建具有符合性的代码。
/Zc:forScope 变量现在默认启用。	这是 ISO C++ 标准行为：现在，如果代码依赖于 **for** 循环中声明的变量在 **for** 循环范围结束后的使用，则此代码在编译时将失败。可以使用 **/Zc:forScope** 恢复为原来的非标准行为。引入此更改是为了默认创建具有符合性的代码。
强制对 Visual C++ 属性进行参数检查。	具有以下特点的代码现在将生成编译器错误 C2065 或编译器警告（级别 1）C4581：当类型不是字符串时，在将命名属性传递给属性构造函数时使用引号；当类型是字符串时，在将命名属性传递给属性构造函数时不使用引号。以前，所有编译器属性都被解析为字符串，并且编译器在需要时插入缺少的引号。属性支持通过增加参数检查验证而得到增强。此更改有助于防止由于向属性构造函数传递了错误的参数而产生意外的行为。
编译器将不会在声明中插入 **int** 类型作为默认类型。	声明中缺少该类型的代码将不再默认使用 **int** 类型。编译器将生成编译器警告 C4430 或编译器警告（级别 4）C4431。ISO C++ 标准不支持默认的 **int**，此更改有助于确保您获得显式指定的类型。

有关更多信息，请参见 Breaking Changes in the Visual C++ 2005 Compiler。

请参见

其他资源

开发 Visual C++ 设备项目

代码解释：Visual C++ 设备项目中由向导生成的代码

Visual C++ 的功能和向导简化了生成多平台设备应用程序、配置文件和项目文件的大多数常见任务。此演练描述由 Windows 32 设备应用程序向导自动生成的代码。以此方式，您可以根据需要扩展并修改您的 Windows 应用程序。

向导生成的 Windows 32 设备应用程序的代码

如果您熟悉 Windows (win32) 桌面编程，则可以很容易识别出向导生成的主例程。

在典型的 Windows 设备应用程序中，主入口点位于 <yourprojectname>.cpp 文件中。下文给出了此文件的示例。

粗略的来看，应用程序具有以下主要函数：

- **WinMain** 函数
- **MyRegisterClass** 函数
- **InitInstance** 函数
- **About Dialog** 函数

下面将对每一个函数做进一步的阐释：

- **WinMain** 函数：`int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)`.。有关更多信息，请参见 WinMain Function（WinMain 函数）。

```
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPTSTR    lpCmdLine,
                   int       nCmdShow)
{
      MSG msg;

      // Perform application initialization:
      if (!InitInstance(hInstance, nCmdShow))
      {
            return FALSE;
      }
```

- Win32 平台。有关更多信息，请参见 Win32 Platforms。

```
#ifndef WIN32_PLATFORM_WFSP
        HACCEL hAccelTable;
        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_WIN32SMARTDEVICE);
#endif // !WIN32_PLATFORM_WFSP
```

- 消息循环。有关更多信息，请参见 Message Handling and Command Targets。

```
        // 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** 函数用于注册该窗口应用程序。

```
//  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**：`FUNCTION InitInstance(HANDLE, int)`。有关更多信息，请参见 InitInstance Member Function。

  `WIN32_PLATFORM_WFSP` 用作 Smartphone 的条件，`WIN32_PLATFORM_PSPC` 用作 Pocket PC 的条件。如果需要进一步区分，可以随时定义自己的条件。

```
// 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;
}
```

- 还将为应用程序生成一个"关于"对话框，作为指导您如何生成其他对话框的示例: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
```

- 已包含一些主消息(如 `WM_COMMAND`)以提供可扩展性。包含 `WinProc` 以用于处理系统及用户输入消息:`LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam. ` 。有关 `WinProc` 的更多信息，请参见 Windows Overview(Windows 概述)。

```
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
                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
```

请参见

任务

演练：创建简单的 Windows 窗体

参考

“选项”对话框 ->“设备工具”->“常规”

工具箱

其他资源

智能设备演练

开发 Visual C++ 设备项目

如何：指定主项目输出的远程路径

可以使用设备项目的属性页为项目的部署设置“远程目录”。“远程目录”指定部署应用程序输出的设备目录。

默认情况下，“远程目录”设置为 %CSIDL_PROGRAM_FILES%\<projectname>。第一部分是一个 **CSIDL** 值，第二部分是项目名。**CSIDL** 值提供了一种标识“远程目录”文件夹的方法，该方法与系统无关。这些值取代了用于此目的的环境变量。下表列出了对“远程目录”参数而言有效的 **CSIDL** 值。值的两侧需加上 % 符号。

CSIDL	值	说明
CSIDL_DESKTOP	0x0000	在 Smartphone 上不受支持。
CSIDL_FAVORITES	0x0006	文件系统目录，它充当用户的收藏夹项的公共储存库。
CSIDL_FONTS	0x0014	包含字体的虚拟文件夹。
CSIDL_PERSONAL	0x0005	文件系统目录，它充当文档的公共储存库。
CSIDL_PROGRAM_FILES	0x0026	Program Files 文件夹。
CSIDL_PROGRAMS	0x0002	文件系统目录，它包含用户的程序组，这些程序组也是文件系统目录。
CSIDL_STARTUP	0x0007	与用户的“启动”程序组对应的文件系统目录。当设备接通电源时，系统启动这些程序。
CSIDL_WINDOWS	0x0024	Windows 文件夹。

为主项目输出指定远程路径

1. 在“解决方案资源管理器”中，右击“<Projectname>”，然后单击快捷菜单上的“属性”。
2. 展开“配置属性”节点，单击“部署”。
3. 在最右侧窗格中，设置项目的“远程目录”属性。

请参见

其他资源

开发 Visual C++ 设备项目

如何：更改默认设备（本机项目）

要更改本机项目中的默认目标设备，请执行下列步骤。

为本机项目选择一个新的默认目标设备

1. 在“解决方案资源管理器”中，右击“<Projectname>”。
2. 在快捷菜单上单击“属性”。
3. 在“属性”对话框中，展开“配置属性”，然后单击“部署”。
4. 在下拉列表中，更改“部署”属性的值，选择一个新的默认目标设备。

请参见

任务

如何：更改默认设备（托管项目）

参考

“部署”对话框（设备）

其他资源

开发 Visual C++ 设备项目

生成并调试 Visual C++ 设备项目

生成并调试设备项目与生成并调试桌面项目略有不同。本节包含解释这些区别的主题。

以下是生成和调试技术的列表：

- 设备项目的线程模型在默认情况下是释放的。但是，如果在生成 CE OS 映像时未选择 DCOM 选项，则 Windows CE 不完全支持 COM 封送处理和关联的定义。因此，在某些 CE 平台上，编译器可能会生成有关 DCOM 支持和单线程与多线程定义的警告。该警告建议您在自己的代码中处理线程和同步。例如，当编译 ATL 设备项目时，编译器可能会发出有关定义 **_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA** 的警告。在某些情形，如创建 COM 对象、使用 Web 服务，以及 Windows Mobile 平台上的 ATL COM 对象时，也会发生这种情况。您可以按如下所示，在主头文件中为单线程对象定义此标志：`#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA`。如果您的代码正在进行多线程处理，您可以放心地忽略该警告。有关 Windows CE 上 COM、DCOM 和线程模型的更多信息，请参见 Windows CE 5.0 文档中的“COM Threads and Processes”(COM 线程和进程)与“Component Services (COM and DCOM)”(组件服务(COM 和 DCOM))。
- 本机设备应用程序在默认情况下是使用静态链接创建的。如果您选择切换到动态链接，则可以将以下运行时 DLL 添加到您的“附加文件”项目属性：
    - MFC 应用程序发布生成配置：将以下运行时 DLL 添加到您的“附加文件”项目属性：
      msvcr80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;atl80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;MFC80U.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;
    - MFC 应用程序调试生成配置：将以下运行时 DLL 添加到您的“附加文件”项目属性：
      msvcr80d.dll|$(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|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;
    - ATL 应用程序发布生成配置：将以下运行时 DLL 添加到您的“附加文件”项目属性：
      msvcr80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;atl80.dll|$(BINDIR)\$(INSTRUCTIONSET)\|%CSIDL_PROGRAM_FILES%\<projectname>|0;
    - ATL 应用程序调试生成配置：将以下运行时 DLL 添加到您的“附加文件”项目属性：
      msvcr80d.dll|$(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;

      有关 Windows CE DLL 加载的更多信息，请参见 Windows CE DLL 加载 LoadLibrary。

注意
同时从不同的目录加载多个同名 DLL(例如 \Windows\aDLL.dll 和 **\Program** Files\aDLL.dll)可能会导致不可预知的结果。建议您一次加载一个 DLL 副本或应调用第一个加载的 DLL。

以下是一些其他注意事项：

- 移植到 MFC 8.0 # 时，请定义 _WIN32_WCE_PSPC。在 MFC 8.0 中，默认情况下不会定义此标志。
- 在“为结构编译”下拉列表中以 Pocket PC 2003 或 Smartphone 2003 为目标时，不支持“ARM4T”选项。该列表位于“<project name> 属性页”对话框中(单击“配置属性”，再单击“C/C++”，再单击“高级”，然后选择“为结构编译”)。
- GAPI 函数在 C++ 中可用，但在 C 中不可用。将 gx.h 包含在代码中的做法只有在 C++ 中才会起作用。如果您在代码中使用 GAPI，请勿使用 **/TC** 编译器选项编译。
- 在用于设备的 MFC 8.0 中，您控制 **CommandBar** 的创建和插入。不再支持 **CDialog::m_pWndEmptyCB** 成员。该成员用来指向为对话框创建的空的 CCommandBar 类(在 Pocket PC 上也称为 **MenuBar**)，您可以引用它来插入自己的 **MenuBar**。
- **Checked::_strlwr_s、_strlwr_s_l、_mbslwr_s, _mbslwr_s_l、_wcslwr_s、_wcslwr_s_l、Checked::tcslwr_s** 和 **Checked::gcvt_s** 为 Windows CE 平台而提供，在将来的版本中，基础 CRT 方法将更为安全，以尽可能确保安全。
- 不再定义 **_WIN32_WCE_PSPC** 标志；您可以使用 **_WIN32_WCE_PSPC WIN32_PLATFORM_PSPC** 标志来作为变通方法。
- 对于 STL 应用程序的移植，请包括 <deque>，而不是 `#include <deque.h>`。
- 由于开发环境的多平台特点，在设备项目中承载 MFC 或 ATL ActiveX 对象时，在对话框资源中，您应在等效桌面 ActiveX 项目

中创建和注册等效控件，以便在资源编辑器中可以将它添加到设备对话框模板并且正确运行。ActiveX 控件的桌面和设备版本应具有相同的 GUID 和设计时属性，如背景色。

本节内容

Visual C++ 设备项目的调试和部署设置

解释 Visual C++ 设备项目独有的调试和部署属性。

请参见

参考

"<Projectname> 属性页"对话框 ->"配置属性"->"调试"(设备)

概念

Visual C++ 设备项目的调试和部署设置

其他资源

调试设备项目

创建并移植 Visual C++ 设备项目

开发 Visual C++ 设备项目

Visual C++ 设备项目的调试和部署设置

Visual C++ 设备项目支持 Visual C++ 桌面项目不支持的属性。在项目的属性页上设置这些属性。

独有的属性页

Visual C++ 设备项目具有以下独有的属性页：

- "<Projectname> 属性页"对话框 ->"配置属性"->"调试"（设备）
- 支持将设备应用程序打包的 IDE 功能
- 项目设计器 ->"调试"窗格
- 有关属性页的更多信息，请参见生成并调试 Visual C++ 设备项目和属性页 (C++)。

访问 Visual C++ 设备项目属性

可以用以下方法访问设备项目属性页：

- 在解决方案资源管理器中右击项目，再单击"属性"。
- 在"解决方案资源管理器"中单击项目名称，然后在"项目"菜单上单击"属性"。

注意
调试 ATL .exe 项目（如部署目标为带有 DCOM 支持的 Windows CE 5.0 SDK）时，将"项目属性"中的"注册输出"设置为"是"，并将"/RegServer"添加到"命令"属性。这会在开始调试时注册 .exe。有关更多信息，请参见"<Projectname> 属性页"对话框 ->"配置属性"->"部署"（设备）。

注意
部署本机设备项目时，依赖项目可能不会随本机设备项目自动部署，可能需要单独部署它们。

请参见

其他资源

生成并调试 Visual C++ 设备项目

调试设备项目

调试智能设备上运行的项目与调试桌面上运行的项目非常相似。一个明显差异是，在设备项目中用于调试的处理器不在开发计算机上，所以连接问题会影响设备调试。

本节中的主题汇总了与调试桌面项目的差异，并提供有关在设备上进行调试的一些有用提示。

安全注意
在包含机密或敏感信息的设备或仿真程序上调试应用程序可能带来安全风险。

本节内容

设备和桌面调试器之间的差异

描述设备调试体验与桌面调试体验之间的各方面差异。

如何：附加到托管设备进程

描述如何附加到已在运行的托管进程。

如何：更改设备注册表设置

描述在设备上使用远程注册表编辑器更改注册表设置。

演练：调试同时包含托管代码和本机代码的解决方案

描述交替启动本机和托管调试器的步骤。

请参见

其他资源

智能设备开发

调试和分析应用程序

Mobile Developer Center（移动开发人员中心）

设备和桌面调试器之间的差异

桌面调试器支持的大多数功能，设备调试器同样支持，但以下情况除外。

不支持编辑并继续

设备调试器不支持在中断模式下编辑源代码并继续的能力。如果要在调试时修改代码，则应先停止调试，编辑代码，然后使用修改后的源代码重新启动。如果试图在中断模式下更改代码，调试器将会发出警告。

本机调试器不支持函数计算

本机设备调试器不支持函数运算。您无法键入包含函数的表达式，也无法计算函数和返回结果。

托管设备调试器支持函数运算。

互操作调试限制

您无法在调试器的单个实例中调试本机代码和托管代码。

若要对混合本机代码和托管代码（或使用 pInvoke 的托管代码）的应用程序进行调试，请在需要开始单步执行代码的每一部分中设置断点。然后，为某一部分（如托管部分）附加任何所需的调试器。当需要另一调试器时，分离前一调试器并附加该调试器。可以根据需要不断重复这些分离/附加步骤，逐句通过整个程序。有关更多信息，请参见演练：调试同时包含托管代码和本机代码的解决方案。

当前不支持在同一进程中同时使用这两个调试实例。

不支持基于属性的调试

.NET Compact Framework 当前不支持基于属性的调试。因此，可视化工具等定义属性的能力对于设备调试器的用户不可用。

不支持桌面调试

无法使用设备调试器来调试为桌面编写的应用程序。请改用桌面调试器。

不支持内核调试

无法使用设备调试器进行内核调试。

不支持“仅我的代码”调试

不能使用“仅我的代码”调试。

运行时调试器 (Cordbg.exe) 添加

运行库调试器可以帮助工具供应商和应用程序开发人员查找并修复以 .NET Framework 公共语言运行库 (CLR) 为目标的程序中的 bug。设备项目将一个新命令和一个新的模式参数添加到运行时调试器。新命令和模式参数（在 Cordbg.exe 会话内部）的语法在下表中进行了说明。

有关更多信息和完整语法，请参见运行库调试器 (Cordbg.exe)。

命令	说明
m[ode] EmbeddedCLR {0|1}	EmbeddedCLR 是将调试器设置为目标设备项目的模式参数。若要控制此设置，请指定 *on* 为 1，或指定 *off* 为 0。
conn[ect] *machine_name port*	连接到远程嵌入式 CLR 设备。 参数： *Machine_name* 必选。远程计算机的名称或 IP 地址。 *Port* 必选。用来连接到远程计算机的端口。

连接问题

调试器运行时关闭设备将导致调试器由于连接失败而关闭。发生连接失败的原因是应用程序仍然在设备上后台运行。Pocket PC 上"X"按钮是一个智能最小化功能，它不关闭应用程序。相反，该按钮会将应用程序设置为后台运行。

若要正确关闭在 Pocket PC 后台运行的应用程序，请在"开始"菜单上敲击"设置"，敲击"系统"选项卡，然后敲击"内存"。在"正在运行程序"选项卡上，敲击要关闭的应用程序，然后敲击"停止"。

请参见

其他资源

调试器指南

调试设备项目

如何：附加到托管设备进程

附加到设备中的进程时采用的方式与附加到桌面上的进程基本相同，只是当进程已运行而未附加调试器时，必须在设备中设置一个注册表项以启用托管调试功能。此项的设置将一直保持，直到您更改该设置；或者，在使用仿真程序的情况下，直到在不保存设置的情况下关闭仿真程序。

注意
设置设备调试项会降低性能。在不进行调试时，请重置该项。

如果试图附加两个调试器，或者在未设置设备注册表项的情况下试图附加托管调试器，则将显示错误信息。

可以通过多种方式（包括文件资源管理器、命令行等等）启动进程。在下面的步骤中，通过“调试”菜单来启动进程。还可以在不附加托管调试器的情况下启动进程，然后在以后附加托管调试器。

如果面向由 Platform Builder 生成的 Windows CE 平台，则需要使用 toolhelp.dll 库填充“可用进程”窗格。此库包含于 Windows Mobile SDK 中。

注意
显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

调试托管进程

调试托管进程

- 在“调试”菜单上单击“启动”。

注意
如果与从“调试”菜单启动的进程分离，则在执行下列步骤（以便在进程运行后附加到该进程）之前，将无法重新附加。也就是说，必须在设备上设置注册表项。

附加到已运行的托管进程

如果计划附加到已运行的进程（例如，通过单击“开始执行(不调试)”，然后附加到正在运行的托管进程），则必须在启动进程且在试图使用“附加到进程”对话框执行附加操作之前设置设备注册表项。下面的步骤对此过程进行了详细说明。

设置设备注册表项以便能够附加到正在运行的进程

1. 在 Windows“开始”菜单上指向“所有程序”，指向“Microsoft Visual Studio 2005”，指向“Visual Studio Tools”，然后单击“远程注册表编辑器”。
2. 使用远程注册表编辑器连接到设备。
3. 定位至或创建以下注册表项：HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETCompactFramework\Managed Debugger
4. 设置或创建名为 `AttachEnabled` 的 DWORD 值。
5. 将该值的数据设置为 1。

注意
设置设备调试项会显著降低性能。在不进行调试时，请通过将数据值重置为 0 或删除 `AttachEnabled` 值来禁用托管附加。

6. 关闭远程注册表编辑器。

   现在已启用了托管附加，可以在不附加调试器的情况下启动进程，然后使用“附加到进程”对话框附加到该进程。

在托管进程运行后附加到该进程

1. 按照上述步骤设置注册表项之后，在不附加调试器的情况下启动进程。
2. 在“工具”菜单上单击“附加到进程”。
3. 在“传输”框中单击“智能设备”。
4. 在“限定符”框中单击“浏览”。

注意
“限定符”框预先设置了当前会话中最常用的设备。

5. 在“连接到设备”对话框中选择平台，选择设备，然后单击“连接”。
6. 在“可用进程”窗格中选择一个或多个要附加到的进程，然后单击“附加”。

注意
默认情况下，代码类型自动设置为“托管(.NET Compact Framework)”(如果可用)，否则设置为“本机(智能设备)”。若要重写默认设置，请单击“选择”以打开“选择代码类型”对话框。注意，您不能同时选中二者。

注意
不支持互操作调试。即，不能同时调试托管和本机代码类型。

与进程分离或终止进程

与进程分离或终止进程

1. 在“调试”菜单上指向“窗口”，再单击“进程”。
2. 在“进程”窗口中右击要与之分离或终止的进程。
3. 在快捷菜单上单击“终止进程”或“与进程分离”。

注意
可以使用相同的快捷菜单重新打开“附加到进程”对话框。

填充“可用进程”窗格

在 Windows CE 项目中填充“可用进程”窗格

- 在 Windows CE OS 映像中包括文件 toolhelp.dll。

  - 或 -

  将文件 toolhelp.dll 手动复制到目标设备。

请参见

任务

演练：调试同时包含托管代码和本机代码的解决方案

其他资源

调试设备项目

生成并调试 Visual C++ 设备项目

如何：更改设备注册表设置

使用远程注册表编辑器更改设备上的注册表设置。

更改注册表设置

1. 在 Windows“开始”菜单上指向“所有程序”，指向，指向“Visual Studio Remote Tools”，然后单击“远程注册表编辑器”。
2. 在“选择 Windows CE 设备”窗口中单击要编辑其注册表的目标设备。

   此时，在“注册表编辑器”的左窗格中显示一个表示目标设备的注册表的节点。
3. 展开该节点进行更改。

请参见

其他资源

调试设备项目

演练：调试同时包含托管代码和本机代码的解决方案

此演练提供对既包含托管 .NET Compact Framework 又包含本机组件的解决方案进行调试的步骤。Visual Studio 2005 不支持对此类设备应用程序进行互操作调试。即，不能同时附加本机调试器和托管调试器。

对既包含本机元素又包含托管元素的解决方案进行调试的建议方法如下：首先任意附加某个节（如托管节）所需的一个调试器，然后分离该调试器并根据需要附加另一个调试器。可以根据需要不断重复这些分离/附加步骤，逐句通过整个程序。

注意
显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

此演练是使用 Visual C# 开发设置编写的。它包含下列部分：

- 启用附加托管调试器
- 启动应用程序
- 在本机代码中设置断点
- 附加本机调试器
- 运行到本机断点
- 附加托管调试器
- 在托管代码中设置断点
- 运行到托管断点
- 结束语

先决条件

此演练依赖于使用另一个演练（演练：Hello World：智能设备的 COM Interop 示例）生成的解决方案。确保已成功生成并运行这个 Hello World 演练。

启用附加托管调试器

默认情况下，包括仿真程序在内的设备不允许将托管调试器附加到已在运行的进程。在既包含托管代码又包含本机代码的设备解决方案中，经常会遇到将托管调试器附加到已在运行的进程的情况。

那么，您应首先将设备设置为允许将托管调试器附加到已在运行的进程。可以通过设置设备上的注册表项来实现这一点。

注意
设置注册表项仅影响附加到已在运行的托管进程。这并不影响使用“启动并进行调试”(F5) 来启动项目。但是，如果在“启动并进行调试”后分离，则需要此进程来重新附加并再次启动调试。

使托管调试器能附加到已在运行的进程

1. 在 Windows 的“开始”菜单上依次指向“所有程序”、“Visual Studio 2005”和“Visual Studio 远程工具”，再单击“远程注册表编辑器”。
2. 在“选择 Windows CE 设备”窗口中，展开“Pocket PC 2003”，再单击“Pocket PC 2003 SE 仿真程序”。这是本演练的目标设备。
3. 单击“确定”。

   将打开“正在连接到设备”进度窗口，然后打开设备仿真程序和 Windows CE 远程注册表编辑器。
4. 在注册表编辑器中，展开“Pocket PC 2003 SE 仿真程序”，再创建以下项：
   HKEY_LOCAL_MACHINE\SOFTWARE\Microsot\.NETCompactFramework\Managed Debugger。

   创建此项的方法是：右击“.NETCompactFramework”，指向“新建”，再单击“项”。

请注意，“Managed”与“Debugger”之间有一个空格。

5. 创建一个名为 `AttachEnabled` 的 DWORD。

   创建 DWORD 的方法是：右击“Managed Debugger”，指向“新建”，再单击“DWORD 值”。

6. 将“名称”设置为 *AttachEnabled*，将“值”设置为 *1*。

注意
设置这个设备调试项将显著降低性能。不进行调试操作时，请将数据值重置为 0 以禁用此功能。

7. 为保留注册表设置，不要关闭设备仿真程序，剩余步骤仍将使用它。可以关闭注册表编辑器。

启动应用程序

下一步是启动 InteropSolution 应用程序。

启动应用程序

1. 打开在演练：Hello World：智能设备的 COM Interop 示例中创建的解决方案。

   确保工具栏上的“目标设备”框中显示“Pocket PC 2003 SE 仿真程序”。

2. 在 Visual Studio 的“调试”菜单上单击“启动调试”或按 F5。

   此步骤立即将本机项目 HelloCOMObject 部署到仿真程序而无需用户进一步干预。

3. 当“部署 SayHello”对话框打开时，选择“Pocket PC 2003 SE 仿真程序”，再单击“部署”。

   此步骤将部署托管项目。

   应用程序在仿真程序中打开。暂时不要单击按钮。

在本机代码中设置断点

下一步是在本机代码中设置断点以准备附加本机调试器。

在本机代码中设置断点

1. 在“解决方案资源管理器”中，右击“Hello.cpp”，然后在快捷菜单上单击“查看代码”。

2. 通过单击“代码编辑器”左侧的空白处，在以 `*text` 开头的行中插入断点。

   断点符号显示为带有感叹号的空心圆圈，表明暂时无法解析此断点。这是因为它目前缺少适当的符号和源。

3. 在 Visual Studio 的“调试”菜单上指向“窗口”，再单击“模块”。

   “模块”窗口显示到目前为止加载的所有模块（例如，托管应用程序 SayHello.exe）。请注意，由于还没有单击应用程序中的按钮，所以尚未加载本机 HelloCOMObject.dll。

附加本机调试器

下一步是分离托管调试器，以便可以附加本机调试器。记住不能为设备项目同时附加两种调试器。以下是每当要从托管调试器切换到本机调试器时使用的步骤。

附加本机调试器

1. 在 Visual Studio 的“调试”菜单上单击“全部分离”。

   此步骤将分离托管调试器，但应用程序仍可继续运行。

2. 在“调试”菜单上单击“附加到进程”。

3. 在“传输”框中选择“智能设备”。

4. 若要填充“限定符”框，请单击“浏览”。

5. 在“连接到设备”对话框中选择“Pocket PC 2003 SE 仿真程序”，再单击“连接”。

6. 若要填充“附加到”框，请单击“选择”。

7. 在“选择代码类型”对话框中选择“调试以下代码类型”，清除“托管”复选框，选择“本机”复选框，再单击“确定”。
8. 在“可用进程”框中，选择“SayHello.exe”，再单击“附加”。

   现在已附加本机调试器。

运行到本机断点

现在即可前进到在本机代码中设置的断点。再次查看“模块”窗口时，您将看到本机模块。但是，由于还没有单击 `button1`，所以尚未加载 HelloCOMObject.dll。

注意
如果以前运行过此演练，则可能已加载调试符号，您即可跳过这些步骤。如果尚未运行过此演练，以下部分提供了加载它们的步骤。

向前执行到本机断点

1. 在“设备仿真程序”窗体中单击 `button1`。

   窗体中将显示“Hello World!”消息，并且“模块”窗口中显示“hellocomobject.dll”。

   如果“hellocomobject.dll”的“符号状态”列显示“未加载符号”，请按照下列步骤操作：

   a. 右击“hellocomobject.dll”，然后在快捷菜单上单击“加载符号”。

   b. 在“查找符号”对话框中，定位到 InteropSolution\HelloCOMObject\Pocket PC 2003 (ARMV4)\Debug\HelloCOMObject.pdb。

   c. 单击“打开”。

      “符号状态”列由“未加载符号”更改为“已加载符号”，并且断点指示器现在显示断点已解析。
2. 在“设备仿真程序”窗体上单击“Hello World!”窗口上的“确定”，然后再次单击 `button1`。

   断点指示器显示执行停止在断点处。
3. 在“调试”菜单上单击“逐语句”或按 F11。

   请注意，执行将移动到下一行。这表示您现在可以逐句通过解决方案的本机部分。

附加托管调试器

下一步是分离本机调试器，以便可以附加托管调试器。记住不能为设备项目同时附加两种调试器。以下是每当要从本机调试器切换到托管调试器时使用的步骤。

附加托管调试器

1. 在 Visual Studio 的“调试”菜单上单击“全部分离”。

   此步骤将分离本机调试器，但应用程序仍可继续运行。
2. 在“调试”菜单上单击“附加到进程”并确保“传输”框包含“智能设备”。
3. 使用以下方法填充“限定符”框：单击“选择”，再选择“Pocket PC 2003 SE 仿真程序”，再单击“连接”。
4. 若要填充“附加到”框，请单击“选择”，再选择“调试以下代码类型”，再选中“托管”框，清除“本机”框，再单击“确定”。

   如果显示消息提醒您托管调试与本机调试不兼容，请单击“确定”。
5. 在“可用进程”框中，选择“SayHello.exe”，再单击“附加”。

   现在已附加托管调试器。

在托管代码中设置断点

下一步是在托管代码中设置断点以准备附加托管调试器。

在托管代码中设置断点

1. 在“解决方案资源管理器”中右击“Form1.cs”，然后在快捷菜单上单击“查看代码”。

2. 在 `string text;` 行上插入断点。

运行到托管断点

现在即可前进到托管代码中设置的断点。

向前执行到托管断点

- 在"设备仿真程序"中单击 `button1`。

  执行在断点处停止。

结束语

出于性能原因，当您不再需要将托管调试器附加到已在运行的进程时，请将设备注册表项重置为 0。

请参见

任务

如何：附加到托管设备进程

如何：更改设备注册表设置

其他资源

调试设备项目

使用 Visual Studio 进行调试

打包设备解决方案以便进行部署

需要创建 CAB 文件才能将设备应用程序分发给最终用户。

本节内容

设备解决方案打包概述

描述对智能设备应用程序进行打包以分发给最终用户所需的进程。

支持将设备应用程序打包的 IDE 功能

描述用于打包的 Visual Studio 环境元素以及取决于目标平台的打包方式的差异。

演练：打包智能设备解决方案以便进行部署

提供对应用程序及其源进行打包的分步介绍。

相关章节

智能设备开发

Mobile Developer Center（移动开发人员中心）

设备解决方案打包概述

部署是将应用程序或组件传输到要运行该程序或组件的目标设备上的过程。在部署解决方案之前，必须将其打包到 CAB 文件中。CAB 文件是一种可执行存档文件，它可以包含应用程序、DLL 等依赖项、资源、帮助文件以及要包括的任何其他文件。在生成 CAB 项目时，Microsoft Visual Studio 2005 会生成用于创建 CAB 的 INF 文件。INF 文件说明每个文件要安装到哪一个文件夹，应用程序要运行于哪一个 Windows CE 版本，以及是否允许卸载应用程序等等。还可以在 CAB 文件中包含一个自定义本机 DLL，用于执行任何自定义安装步骤，例如检查 Windows CE 或 .NET Compact Framework 的版本号，确定是否存在其他组件等等。在将 CAB 文件复制到目标设备上之后，用户点击该文件可以开始安装过程。这称为解开 CAB。

注意

Microsoft Visual Studio 2005 提供了打包 CAB 文件的工具。但未提供任何将 CAB 文件部署到目标设备的工具。对于简单部署方案，可以使用 ActiveSync 连接将 CAB 文件从桌面计算机拖到设备。对于更为复制的部署方案，有多种第三方部署解决方案可供选用。有关更多信息，请访问“Mobile and Embedded Application Developer Center”（移动和嵌入式应用程序开发中心）。

Visual Studio 2005 使得在大多数情况下直接在 Visual Studio 集成开发环境 (IDE) 中执行所有必需的打包任务成为可能。可以采用以下方法创建 CAB 文件：首选为现有的解决方案添加智能设备 CAB 项目，然后使用与桌面安装项目相同的用户界面向项目中添加文件、快捷方式和注册表项。在生成安装项目时，将创建 CAB 文件。

为 Pocket PC 应用程序创建的 CAB 文件与为 Smartphone 应用程序创建的 CAB 文件之间存在一些差异。基于 Windows Mobile 2003SE 和早期版本的 Pocket PC 不支持压缩或签名的 CAB 文件。Smartphone CAB 文件必须进行压缩，在将这些文件安装到设备上之前，必须对 EXE 或 DLL 文件以及 CAB 文件本身进行数字签名。

在使用 Visual Studio 创建 CAB 文件之后，下一步就是使用任意一种以下常用文件传输方法将该文件传输到目标设备上：通过从设备发出的 FTP 或 HTTP 请求，使用 Windows 资源管理器手动将文件从桌面开发计算机复制到连接的设备上的文件夹中，对 Smartphone 使用无线 (OTA) 传输等等。

请参见

任务

演练：打包智能设备解决方案以便进行部署

概念

支持将设备应用程序打包的 IDE 功能

其他资源

智能设备开发

部署中的文件安装管理

支持将设备应用程序打包的 IDE 功能

要对解决方案进行打包以部署到智能设备上，需要使用与桌面解决方案相同或类似的 Microsoft Visual Studio 2005 集成开发环境 (IDE) 功能。下表描述了这些功能。

功能	如何查找	备注
智能设备 CAB 项目模板	在“文件”菜单上指向“添加”，单击“新建项目”，然后单击“其他项目类型”，再单击“安装和部署”。	单击此图标可以为现有的解决方案添加新的 CAB 项目。请注意，这是此对话框中唯一可用于智能设备的项目类型。为 CAB 项目选择名称之后，单击“确定”，该项目即添加到解决方案，并显示在“解决方案资源管理器”中。
文件系统编辑器	在“解决方案资源管理器”中右击 CAB 项目名称，单击“视图”，再单击“文件系统”。	使用此编辑器可以指定要添加到 CAB 中的文件，以及这些文件应安装到的设备文件夹。
注册表编辑器	在“解决方案资源管理器”中右击 CAB 项目名称，单击“视图”，再单击“注册表”。	使用此编辑器可以指定应用程序所需的全部特殊注册表项。
CAB 项目的属性窗口	在“解决方案资源管理器”中选择 CAB 项目，再单击“视图”菜单上的“属性窗口”。	使用此窗口可以指定 CE 安装程序 DLL（如果有）的名称、应用程序的制造商名称、可运行此应用程序的最低和最高 Windows CE 版本，以及其他选项。
项目属性页	在“解决方案资源管理器”中右击 CAB 项目名称，再单击“属性”。	使用此对话框可以指定配置（例如，*Debug*）、输出文件名和安全证书。

注意

由于这些编辑器同时也用于桌面安装项目，因此可能会对智能设备 CAB 项目禁用某些选项。

在某些情况下，您可能会编写仅设计用于运行在特定平台（如 Windows Mobile 2003 SE 和更高版本）上的应用程序。在这些情况下，可以阻止 CAB 文件安装到指定的不受支持的平台，但要实现此功能，必须手动编辑 INF 文件，然后使用命令行工具重新打包该 CAB。如果使用 Visual Studio 重新打包 CAB，将会覆盖所做的更改。

Pocket PC 与 Smartphone 的对比

在 Windows Mobile 2003 SE 和更早版本上，用于 Pocket PC 的 CAB 文件与用于 Smartphones 的 CAB 文件之间的主要区别在于，Pocket PC 不支持压缩或签名的 CAB 文件。Smartphone CAB 文件必须进行压缩，并且 .exe 或 .dll 文件以及 CAB 文件本身在安装到设备上之前，必须先进行数字签名。有关更多信息，请参见设备项目中的安全性。

本机应用程序与托管应用程序的对比

既可以为使用 C++ 编写的应用程序创建智能设备 CAB 项目，也可以为使用 Visual C# 或 Visual Basic 编写的应用程序创建智能设备 CAB 项目，这两者之间的唯一区别在于，对于本机应用程序，必须手动为 CAB 项目添加系统依赖项 **atl80.dll**、**mfc80U[d].dll** 和/或 **msvcrt[d].dll**。对于托管应用程序，则无需为 CAB 文件添加任何 .NET Compact Framework DLL。如果您面向的是 1.0 版，则所有的 DLL 都已存在于任何基于 Windows Mobile 2003SE 和更高版本的设备上。如果您面向的是 2.0 版的 .NET Compact Framework，则需要确定设备是否已安装了此版本。为此，可以编写一个 Windows CE 安装程序 DLL 来检查目标设备上的 mscoree.dll 的版本。如果 2.0 版的 .NET Compact Framework 不存在，那么可以部署随 Visual Studio 一起提供的相应 CAB，该 CAB 位于 Microsoft Visual Studio 8\SmartDevices\SDK\CompactFramework\2.0\WindowsCE 路径下。既可以在应用程序 CAB 之外单独部署此 CAB，也可以将此 CAB 嵌入到应用程序 CAB 内部。

警告

如果重新发布动态链接到 MFC/ATL 的本机应用程序，并将 MFC/ATL 运行时 DLL 部署到应用程序目录，应用程序可能不会链接到该目录中的 DLL。在 Windows CE 上，如果两个 DLL 具有相同的文件名和不同的路径，则只加载具有该文件名的第一个 DLL。而不会加载后面具有相同文件名的 DLL。实际上，应用程序将链接至之前已被另一个应用程序加载的具有该文件名的 DLL。

若要确保应用程序链接至其目录中的 DLL，请确保没有其他应用程序正在使用相同文件名的 DLL。

智能设备部署与桌面部署的对比

在“新建项目”对话框中，通过单击“其他项目类型”，再单击“安装和部署”，可以同时访问桌面安装项目和设备安装项目。部署桌面应用程序时，可以从“安装项目”、“合并模块项目”、“CAB 项目”、“Web 安装项目”和“安装向导”中进行选择。上述所有项目类型都不可用于设备应用程序。智能设备不支持 ClickOnce 部署。若要创建可部署到所有基于 Windows CE 的设备（包括 Smartphone 和 Pocket PC）的 CAB 文件，必须使用智能设备 CAB 项目。

请参见

任务

演练：打包智能设备解决方案以便进行部署

概念

设备解决方案打包概述

演练：打包智能设备解决方案以便进行部署

本演练演示如何使用 Visual Studio 2005 将应用程序及其资源打包到一个 CAB 文件中，以便可以将它部署到最终用户的智能设备上。

注意
显示的对话框和菜单命令可能会与“帮助”中描述的不同，具体取决于当前的设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

在本演练中，您可以从任何用 Visual Basic 2005、Visual C# 2005 或 Visual C++ 2005 编写的智能设备解决方案开始。有关更多信息，请参见演练：创建简单应用程序。

本演练演示如何执行下列操作：

- 将一个 CAB 项目添加到解决方案中。
- 更改产品名称。
- 更改输出路径。
- 用应用程序的主输出填充 CAB 文件。
- 在必要时添加依赖项。
- 创建应用程序的快捷方式。
- 编辑注册表项。

先决条件

一个现有的智能设备解决方案。对于本打包演练，可以考虑创建并生成一个简单的项目，例如，演练：创建用于设备的 Windows 窗体应用程序中介绍的项目。

注意
显示的对话框和菜单命令可能会与“帮助”中描述的不同，具体取决于当前的设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

安装 CAB 项目

为解决方案添加智能设备 CAB 项目

1. 打开现有的智能设备项目，并确保“解决方案资源管理器”可见。
2. 在“文件”菜单上指向“添加”，然后单击“新建项目”。

   出现“添加新项目”对话框。
3. 在左侧的“项目类型”窗格中展开“其他项目类型”节点，再单击“安装和部署”。
4. 在右侧的“模板”窗格下选择“智能设备 CAB 项目”。

   这是唯一可用于智能设备的 CAB 项目类型。其他项目类型仅用于桌面解决方案。
5. 在“名称”框中，键入 **CABProject**，然后单击“确定”。

   此 CAB 项目即会添加到您的解决方案，并显示在“解决方案资源管理器”中。现在显示“文件系统编辑器”的两个窗格。

自定义 CAB 项目

更改产品名称和其他项目属性

1. 在“解决方案资源管理器”中选择“CABProject”（如果尚未选定）。
2. 在“视图”菜单上单击“属性窗口”，或打开“属性”窗口。

3. 在属性网格的“ProductName”字段中，将值更改为“MyProduct”。

   “ProductName”属性的值决定了在文件夹名称和“添加或删除程序”对话框中为应用程序显示的名称。

   - 还可以使用此窗口更改制造商的名称，以及指定允许的最低和最高操作系统版本。
   - 可以将“OSVersionMin”属性设置为 4.21，以指示您的 Pocket PC 应用程序具有屏幕方向感知功能。但是，将此属性设置为 4.21 会阻止应用程序安装到基于 Windows Mobile 2003 或更早版本的 Pocket PC 上。若要允许在此类设备上安装该程序，并通知较新的设备该程序具有屏幕方向感知功能，则必须手动编辑 .inf 文件，将“BuildMax”属性设置为下列值之一：

     0xA0000000，指示应用程序支持方形屏幕（240x240 像素）

     0xC0000000，指示应用程序支持屏幕旋转

     - 或 -

     0xE0000000，指示应用程序支持方形屏幕和屏幕旋转。

     有关更多信息，请参见 MSDN 白皮书“Developing Screen Orientation-Aware Applications”（开发识别屏幕方向的应用程序）。
   - 对于基于 Windows Mobile 2003SE 和更早版本的 Pocket PC 解决方案，“Compress”属性和“NoUninstall Device Deployment”属性必须为 False。请注意，对于配备了 Compact Framework 2.0 的设备，此选项可以设置为 **true**。有关更多信息，请参见“智能设备 Cab 项目”->“属性”窗口。
   - 如果您使用的是 Windows CE 安装程序 DLL，使用此属性网格可以指定文件名和位置。有关 Windows CE 安装程序 DLL 的更多信息，请参见 Pocket PC 或 Smartphone SDK 文档。

更改 CAB 文件的名称并添加身份验证

1. 在“解决方案资源管理器”中右击“CABProject”，然后单击“属性”。

   出现 CAB 项目的“属性页”对话框。在“输出文件名”框中，将 CAB 文件的名称和路径更改为 **Debug\MyApp.cab**，然后单击“确定”。

2. 还可以使用此属性页为项目添加身份验证。身份验证对于 Smartphone 解决方案是必需的，但是在基于 Windows Mobile 2003 SE 和更早版本的 Pocket PC 解决方案上，身份验证不受支持。有关更多信息，请参见设备项目中的安全性。

为 CAB 项目添加设备项目应用程序

1. 在“文件系统编辑器”的左窗格中，选择“应用程序文件夹”节点，以指定下列步骤中选择的文件将安装到目标设备上的此文件夹中。

   如果“文件系统”编辑器不可见，请在“解决方案资源管理器”中右击 CAB 项目名称，选择“视图”，再单击“文件系统”。

2. 在 Visual Studio 中的“操作”菜单上指向“添加”，然后单击“项目输出”。
3. 在“添加项目输出组”对话框中，从“项目”下拉列表中选择您的智能设备项目。
4. 从输出列表中选择“主输出”，然后单击“确定”。

注意

为使用 C++ 编写的应用程序创建智能设备 CAB 项目时，如果要动态链接到 DLL，必须手动向 CAB 项目添加所有依赖项，如 atl80.dll、mfc80U.dll 和/或 msvcr.dll。但是，为了减少 MFC/ATL DLL 的依赖项，强烈建议您使用静态链接。如果采用的是静态链接，则无需随同应用程序一起重新发布 DLL。如果采用的是动态链接，并需要在 CAB 中重新发布 DLL，请不要将 DLL 安装到设备上的系统目录（如 \windows）中，而应将 DLL 安装到本地应用程序目录中。如果重新发布一个应用程序套件，而套件中的所有应用程序都动态地链接到 ATL/MFC 运行库，建议您将所有应用程序和运行库 DLL 安装到一个单独的应用程序目录中，并为可以放置在其自己的文件夹中的应用程序提供快捷方式。这样可以避免发生系统目录中的 DLL 在以后被替换，从而破坏动态链接到这些 DLL 的任何应用程序的危险，同时还可以节省一些空间。

向 CAB 项目中添加依赖项（仅限 C++ 项目）

1. 在“解决方案资源管理器”中右击 CAB 项目的名称，指向“添加”，再单击“文件”。
2. 导航至 <Visual Studio 安装文件夹>\VC\ce\dll\<平台>。

3. 选择要添加的文件。

   - 对于 MFC 项目，请按 Ctrl，再单击 MFC80U.DLL、atl80.dll 和 msvcr80.dll。如果应用程序需要特定于 MFC 语言的资源，您可能还需要单击一个或多个特定于语言的 DLL。
   - 对于 ATL 项目，请按 Ctrl，再单击 atl80.dll 和 msvcr80.dll。如果 ATL 解决方案支持 MFC，还需要单击 MFC80U.DLL。
   - 对于 Win32 项目，请单击 msvcr80.dll。

4. 在“添加文件”对话框中单击“打开”，将文件添加到 CAB 项目中。
5. 在“文件系统编辑器”的左窗格中，右击“目标计算机上的文件系统”。
6. 单击“添加特殊文件夹”，然后单击“Windows 文件夹”。
7. 在“文件系统编辑器”的左窗格中，单击包含主输出的文件夹。默认情况下，DLL 已添加到与主输出相同的文件夹中。若要将这些 DLL 移动到 Windows 文件夹中，请在“文件系统编辑器”的中间窗格中选择这些文件，然后将它们拖到“Windows 文件夹”图标上。
8. 使用相同的过程添加解决方案需要的其他任何依赖项。可以将依赖项添加到任何文件夹中；而不必将它们添加到“Windows”文件夹。

为设备项目应用程序创建快捷方式

1. 在“文件系统编辑器”的右窗格中，选择“<your application project name> 的主输出”。
2. 在“操作”菜单上选择“创建 <your application project name> 的主输出的快捷方式”。

   此命令将在“输出”项的下面添加一个快捷方式项。
3. 右击该快捷方式项，单击“重命名”，将此快捷方式重命名为适用于快捷方式的内容。

添加注册表项

1. 在“解决方案资源管理器”中选择 CAB 项目。
2. 在“视图”菜单上指向“编辑器”，然后单击“注册表”。
3. 在“注册表编辑器”中，右击 HKEY_CURRENT_USER，然后单击快捷菜单上的“新建项”。
4. 当“注册表编辑器”中显示“新建项”项时，将其重命名为“SOFTWARE”。
5. 右击此新项，指向“新建”，然后单击“项”。
6. 当“注册表编辑器”中显示“新建项”项时，将其重命名为“MyCompany”。
7. 右击“MyCompany”项，再单击快捷菜单上的“属性窗口”。

   “名称”值已更改为“MyCompany”。

生成和部署 CAB 文件

生成 CAB 文件

1. 在“生成”菜单上单击“生成 CABProject”。

   - 或 -

   在“解决方案资源管理器”中右击“CABProject”，再单击“生成”。
2. 在“文件”菜单上单击“全部保存”。

   Smartphone 解决方案的 CAB 文件在部署到最终用户的设备上之前必须进行数字签名。基于 Windows Mobile 2003SE 和更早版本的 Pocket PC 解决方案不支持数字签名。有关更多信息，请参见如何：对 CAB 文件进行签名（设备）。

将 CAB 文件部署到设备上

1. 在“Windows 资源管理器”中定位到存储此解决方案的文件夹。可以在解决方案的“CABProject\Release”文件夹中找到此 CAB 文件。

2. 将 CAB 文件复制到与 ActiveSync 4.0 或更高版本连接的设备上。

   当用户在设备上的“资源管理器”中点击此 CAB 文件名时，Windows CE 将解开该 CAB，并将应用程序安装到设备上。

   有关更多信息，请参见 Smartphone 和 Pocket PC SDK 文档。

请参见

其他资源

智能设备演练

打包设备解决方案以便进行部署

设备项目中的安全性

设备根据安全策略设置和安全角色来限制应用程序的安装和执行以及对系统资源的访问。此节包括的主题解释了如何设置应用程序以在使用不同安全模型的设备上运行。

本节内容

安全概述(设备)

描述证书、安全模型、配置和其他常规信息。

设备签名过程的图形化流程图

以图形方式显示如何在设备项目中使用签名。

如何:在设备项目中导入和应用证书

描述如何使用 Visual Studio 环境来定位并应用证书。

如何:对 Visual Basic 或 Visual C# 应用程序进行签名(设备)

描述如何使用 Visual Studio 环境在使用 .NET Compact Framework 的项目中将证书应用到 EXE 和 DLL 文件。

如何:对 Visual Basic 或 Visual C# 程序集进行签名(设备)

描述如何使用 Visual Studio 环境将证书应用到根据 .NET Compact Framework 编写的项目中。

如何:在 Visual C++ 项目中对项目输出进行签名(设备)

描述如何使用 Visual Studio 环境将证书应用到 Visual C++ 项目中的项目输出。

如何:对 CAB 文件进行签名(设备)

描述如何使用 Visual Studio 环境将证书应用到智能设备 Cab 项目。

如何:在 Visual Basic 或 Visual C# 项目中提供设备

描述如何使用 Visual Studio 环境将证书添加到项目的证书存储中(项目使用 .NET Compact Framework)。

如何:在 Visual C++ 项目中提供设备

描述如何使用 Visual Studio 环境将证书添加到项目的证书存储中(项目以 Visual C++ 编写)。

如何:为设备提供安全模型

描述如何设置设备的安全模型。

如何:向设备查询其安全模型

描述如何发现设备上安装的证书。

如何:将 Signtool.exe 作为生成后事件启动(设备)

描述生成后事件更改的原始二进制文件后如何应用证书。

参考

安全性(C# 编程指南)

安全和 Visual Basic 开发

C++ 安全性最佳做法

.NET Compact Framework 中的安全

请参见

其他资源

智能设备开发

Mobile Developer Center(移动开发人员中心)

安全概述(设备)

设备根据安全策略设置和安全角色来限制应用程序的安装和执行，以及对系统资源的访问。为了使应用程序可以在特定设备上正确运行，必须用适当的证书对这些设备进行签名，而且该证书必须存在于设备的证书存储区中。

应对哪些文件签名？

最佳实践要求对 EXE、DLL、CAB 和 MUI(多语言用户界面)文件进行签名。在托管项目中，程序集文件也应签名。

安全模型

设备在经过设置后，还可以在几种安全模型中运行。不同的安全模型可以限制对那些不具备适当授权的应用程序的访问。有关设备安全模型的更多信息，请参见 Windows Mobile-Based Smartphone Developer's Guide(基于 Windows Mobile 的 Smartphone 开发人员指南)和如何：为设备提供安全模型。

在生成后二进制文件更改后签名

如果运行更改二进制文件的生成后步骤，则需要对二进制文件进行重新签名；也就是说，必须禁用项目属性中的“Authenticode 签名”，并改为按生成后步骤签名。这一操作是必需的，因为签名后对二进制文件所做的任何更改都会使签名无效。因此，必须对二进制文件进行重新签名。

提供

提供设备指的是向设备的证书存储区中添加数字证书。试图在设备上安装或运行应用程序时，设备的操作系统会检查对应用程序签名所用的证书是否在设备的证书存储区中。有关证书存储区的更多信息，请参见 A Practical Guide to the Smartphone Application Security and Code Signing Model for Developers(Smartphone 应用程序安全和代码签名模型开发人员实践指南)。

- Smartphone 由移动运营商预配置。
- 基于 Windows Mobile 5.0 的 Pocket PC 由 OEM 预先配置；而较早的 Pocket PC 通常则不会。
- Visual Studio 中的“设置仿真程序”的映像已预先配置。

有关配置的更多信息，请参见 Smartphone 或 Pocket PC SDK 文档。

证书文件

为了帮助开发针对各种安全模型的应用程序，Visual Studio 2005 中包含以下证书文件，这些文件默认位于 \Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools\TestCertificates 中：

- TestCert_Privileged.pfx
- TestCert_UnPrivileged.pfx

这些 .pfx 文件包含证书和相应的私钥。(如果尝试用没有相应私钥的证书对应用程序进行签名，会导致签名过程失败。)这些文件都没有密码。

将这些 .pfx 文件导入台式计算机的个人证书存储区。如果尝试将它们导入另一个存储区(如受信任根证书颁发机构存储区)，Visual Studio 便会显示详细的“安全警告”。有关更多信息，请参见如何：在设备项目中导入和应用证书。

请参见

任务

如何：在 Visual C++ 项目中提供设备

如何：在 Visual Basic 或 Visual C# 项目中提供设备

概念

保护连接字符串

其他资源

设备项目中的安全性

设备签名过程的图形化流程图

下面的图形化表示形式显示了签名过程中各个部分之间的相互关系。

项目签名过程中的第一个步骤是选择证书。证书可以是证书存储区中的现有证书，也可以导入新证书并进行选择。

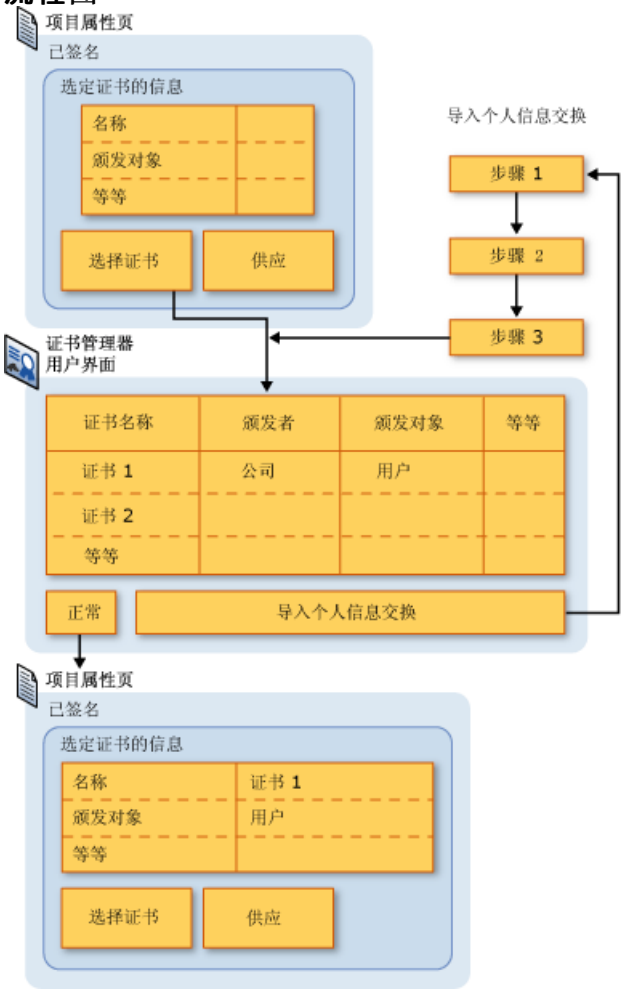
选择证书后，必须决定是否向设备提供证书。如果提供设备，则必须选择设备上将要接收证书的证书存储区（特权存储区或非特权存储区）。

每次生成时，都会用所选证书为项目签名。每次部署（例如按 F5）时，都会向设备提供所选证书（位于设备的选定证书存储区中）。

在 Visual Studio IDE 中，通常可在项目的一个属性页上设置签名选项。

如果不希望项目用 Authenticode 签名，则此流程图中的任何一部分都不会应用于项目。

流程图



请参见

任务

如何：对 Visual Basic 或 Visual C# 应用程序进行签名（设备）

如何：对 Visual Basic 或 Visual C# 程序集进行签名（设备）

如何：在 Visual C++ 项目中对项目输出进行签名（设备）

如何：对 CAB 文件进行签名（设备）

其他资源

设备项目中的安全性

如何:在设备项目中导入和应用证书

"选择证书"对话框是对设备项目进行签名的中心门户。正如下面的步骤所述，它提供了通向"管理证书"对话框和"证书导入向导"的入口。

注意
显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在"工具"菜单上选择"导入和导出设置"。有关更多信息，请参见 Visual Studio 设置。

显示"选择证书"对话框

"选择证书"对话框的访问方式取决于要签名的项目类型。

显示"选择证书"对话框

1. 在"解决方案资源管理器"中，右击"<Projectname>"，然后单击快捷菜单上的"属性"。
2. 通过使用下面的任意一个过程继续:
   - 对于 Visual Basic 和 Visual C# 项目:在"项目设计器"中单击"设备"，选择"Authenticode 签名"，再单击"选择证书"。
   - 在 Visual C++ 项目中，选择"Authenticode 签名"，再单击"证书"属性行中的省略号按钮。
   - 在智能设备 CAB 项目中，选择"Authenticode 签名"，再单击"从存储区选择"。

为设备项目选择证书

按照先前步骤显示"选择证书"对话框后，可以选择所需的证书。

使用"选择证书"对话框为项目选择证书

- 如果"选择证书"对话框显示了您要为项目使用的证书，则可以选择该证书并单击"确定"。

  当生成项目时，会使用该证书对项目进行签名。
- 如果"选择证书"对话框中没有显示您要为项目使用的证书，则可以使用"证书导入向导"导入一个证书。

为设备项目导入证书

下面的步骤演示如何通过导入 Visual Studio 提供的测试证书并将这些证书应用于项目来填充"选择证书"对话框。如果想应用其他的证书，可以按照相同的过程进行操作。

Visual Studio 为证书导入任务提供了三个用户界面元素，以便于将证书应用于项目:

- "选择证书"对话框，用于指定要应用于当前项目的证书。
- "管理证书"对话框，用于列出开发计算机上可用的证书文件。
- "证书导入向导"，用于引导您选择证书文件并指定其目标存储位置。

使用证书导入向导导入测试证书

1. 在"选择证书"对话框中单击"管理证书"。

   "管理证书"对话框便会显示开发计算机上存储的证书的列表。
2. 单击"导入"，打开"证书导入向导"。
3. 单击"下一步"，打开向导的"要导入的文件"页。
4. 单击"浏览"，定位至 Visual Studio 中的"TestCertificates"文件夹。

   默认情况下，此文件夹位于 \Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools 下。
5. 将"文件类型"选项更改为"所有文件(*.*)"，选择 *TestCert_Privileged.pfx* 或 *TestCert_Unprivileged.pfx*，再单击"打开"。

6. 在向导的“要导入的文件”页中单击“下一步”，打开“密码”页。

   将“密码”框保留为空。这些测试证书都没有密码。

7. 单击“下一步”，打开“证书存储区”页。确保在“证书存储区”框中选择“个人”。

8. 单击“下一步”，显示完成页，再单击“完成”。

   随即出现“导入成功”消息。

9. 单击“确定”关闭该消息。

   现在，证书便出现在“管理证书”列表中。单击“关闭”，返回到“选择证书”对话框。

10. 选择所需的证书，然后单击“确定”。

    所选的证书便会在开始时的属性页中列出。

请参见

任务

如何：对 Visual Basic 或 Visual C# 应用程序进行签名（设备）

如何：对 Visual Basic 或 Visual C# 程序集进行签名（设备）

如何：在 Visual C++ 项目中对项目输出进行签名（设备）

如何：对 CAB 文件进行签名（设备）

概念

安全概述（设备）

其他资源

设备项目中的安全性

如何：对 Visual Basic 或 Visual C# 应用程序进行签名（设备）

以下步骤假定解决方案中已有一个智能设备 Visual Basic 或 Visual C# 项目。有关更多信息，请参见使用 .NET Compact Framework 进行设备编程。

对于 EXE 和 DLL 项目，这些步骤都是相同的。

注意
显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

为 Visual Basic 或 Visual C# 设备项目签名

1. 在“解决方案资源管理器”中右击 Visual Basic 或 Visual C# 项目，再单击快捷菜单上的“属性”。
2. 在“设备”页上单击“Authenticode 签名”。
3. 单击“选择证书”。
4. 在“选择证书”对话框中：
   - 如果所需的证书显示在列表中，请选择该证书，然后单击“确定”关闭对话框。
   - 如果所需的证书未出现在列表中，请单击“管理证书”，打开“管理证书”对话框。有关更多信息，请参见如何：在设备项目中导入和应用证书。

     完成证书获取后，在“选择证书”对话框中单击“确定”。证书详细信息便会出现在“设备”页的“应用程序签名”窗口中。

请参见

其他资源

设备项目中的安全性

如何：对 Visual Basic 或 Visual C# 程序集进行签名（设备）

以下步骤假定解决方案中已有一个智能设备 Visual Basic 或 Visual C# 项目。有关创建这些项目的更多信息，请参见使用 .NET Compact Framework 进行设备编程。

对于 EXE 和 DLL 项目，这些步骤都是相同的。

注意
显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

为 Visual Basic 或 Visual C# 设备项目中的程序集签名

1. 在“解决方案资源管理器”中右击 Visual Basic 或 Visual C# 项目，再单击快捷菜单上的“属性”。
2. 在“签名”页上单击“为程序集签名”。
3. 在“选择强名称密钥文件”框中：
   - 如果要使用现有的强名称密钥文件，请单击“<浏览...>”，打开“选择文件”对话框。
   - 如果要创建新的强名称密钥文件，请单击“新建”，打开“创建强名称密钥”对话框。

延迟为程序集签名

- 完成以上步骤后，单击“仅延迟签名”。

  当没有所需私钥的访问权限时，可使用此功能。延迟签名可以提供公钥并延迟添加私钥，直至程序集提交为止。有关更多信息，请参见如何：延迟为程序集签名 (Visual Studio)。

请参见

概念

ClickOnce 应用程序的强名称签名

其他资源

设备项目中的安全性

如何：在 Visual C++ 项目中对项目输出进行签名（设备）

以下步骤假定解决方案中已有一个智能设备 Visual C++ 项目。有关更多信息，请参见使用 Visual C++ 进行设备编程。

注意
显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

在 Visual C++ 设备项目中为项目输出签名

1. 在“解决方案资源管理器”中右击 Visual C++ 项目，再单击快捷菜单上的“属性”。
2. 展开“Authenticode 签名”页。
3. 对于“Authenticode 签名”属性，选择“是”。
4. 对于“证书”属性，单击省略号按钮（“...”）。
5. 在“选择证书”对话框中：
   - 如果所需的证书出现在列表中，请选择该证书，然后单击“确定”，关闭对话框。
   - 如果所需的证书未出现在列表中，请单击“管理证书”，打开“管理证书”对话框。有关更多信息，请参见如何：在设备项目中导入和应用证书。

     完成证书获取后，在“选择证书”对话框中单击“确定”。证书便会出现在“Authenticode 签名”页的“证书”行中。
6. 在“Authenticode 签名”页上单击“确定”。

请参见

其他资源

设备项目中的安全性

如何：对 CAB 文件进行签名（设备）

以下步骤假定您的解决方案中有一个智能设备 Cab 项目。有关更多信息，请参见设备解决方案打包概述。

注意

显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

为智能设备 .cab 文件签名

1. 在“解决方案资源管理器”中右击智能设备 Cab 项目，然后在快捷菜单上单击“属性”。
2. 在“生成”页上选择“Authenticode 签名”。
3. 单击“从存储区选择”。
4. 在“选择证书”对话框中：
   - 如果所需的证书显示在列表中，请选择该证书，然后单击“确定”关闭对话框。
   - 如果所需的证书未出现在列表中，请单击“管理证书”，打开“管理证书”对话框。有关更多信息，请参见如何：在设备项目中导入和应用证书。

     完成证书获取后，在“选择证书”对话框中单击“确定”。该证书出现在“生成”页的“证书”框中。
5. 在“生成”页上单击“确定”。

请参见

其他资源

设备项目中的安全性

如何：在 Visual Basic 或 Visual C# 项目中提供设备

提供设备指的是向设备的证书存储区中添加数字证书。有关更多信息，请参见 A Practical Guide to the Smartphone Security and Code Signing Model for Developers（《Smartphone 安全和代码签名模型开发人员实践指南》）。

以下步骤假定：

- 解决方案中已有一个智能设备 Visual Basic 或 Visual C# 项目。有关更多信息，请参见使用 .NET Compact Framework 进行设备编程。
- 已经为应用程序签名。有关更多信息，请参见如何：对 Visual Basic 或 Visual C# 应用程序进行签名（设备）。

注意

显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

在托管设备项目中提供设备

1. 在“解决方案资源管理器”中右击 Visual Basic 或 Visual C# 项目，再单击快捷菜单上的“属性”。
2. 打开“设备”页。
3. 在“设备提供”框中，选择以下任一选项：
   - 不提供目标设备
   - 提供特权存储区
   - 提供非特权存储区

请参见

其他资源

设备项目中的安全性

如何：在 Visual C++ 项目中提供设备

为了完成下面的过程：

- 解决方案中必须有一个智能设备 Visual C++ 项目。有关更多信息，请参见使用 Visual C++ 进行设备编程。
- 必须已为 Visual C++ 项目的项目输出签名。有关为 Visual C++ 项目签名的更多信息，请参见如何：在 Visual C++ 项目中对项目输出进行签名（设备）。

注意

显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

在 Visual C++ 设备项目中提供设备

1. 在“解决方案资源管理器”中右击 Visual C++ 项目，再单击快捷菜单上的“属性”。
2. 展开“Authenticode 签名”页。
3. 对于“提供设备”属性，可选择“特权证书存储区”或“非特权证书存储区”。
4. 在“Authenticode 签名”页上单击“确定”。

请参见

其他资源

设备项目中的安全性

如何：为设备提供安全模型

可以显式地设置设备的安全模型，以便在各种不同的安全模型下测试应用程序。如果设备已被原始设备制造商 (OEM) 锁定，则不可能再提供不同的安全模型。但是，如果设备未被锁定，则可为它提供任何安全模型。

以下安全模型 XML 文件包含在 Visual Studio 2005 中。默认位置为 \Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools\SecurityModels。

- Locked.xml 设置以下两层的安全模型：
  - 在运行应用程序前给予提示。
  - 不运行未签名的应用程序。
- Prompt.xml 设置以下两层的安全模型：
  - 在运行应用程序前给予提示。
  - 将未签名的应用程序作为非特权应用程序运行。
- Open.xml 设置以下一层的安全模型：
  - 不给予提示。
  - 将所有应用程序（包括未签名应用程序）作为特权应用程序运行。

有关更多信息，请参见为基于 Windows Mobile 的设备提供的模型。

要为设备提供安全模型，需要以下组件：

- ActiveSync。
- RapiConfig.exe，默认位置为 \Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools。
- 安全 XML 文件。

为设备提供安全模型

1. 建立到设备的 ActiveSync 连接。
2. 在命令提示符处键入以下命令，其中 *securityfile.xml* 为安全模型 XML 文件：

```
RapiConfig.exe /P /M <securityfile.xml>
```

请参见

其他资源

设备项目中的安全性

如何：向设备查询其安全模型

您可以查询设备，以查看设备证书存储区中已安装的证书。可以使用这些信息确定要用来为应用程序签名的证书。

运行 RapiConfig.exe，并传入包含证书存储区查询的 StoreQuery XML 文件即可完成查询。然后 RapiConfig.exe 输出包含查询结果的 XML 文件。

RapiConfig.exe、CertStoreQuery.xml 以及一些示例 xml 查询文件的默认位置为 \Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SDKTools。

查询设备证书存储区需要使用以下组件：

- ActiveSync。
- RapiConfig.exe。
- 证书存储区查询 XML 文件 (CertStoreQuery.xml)。

向设备查询其安全模型

1. 建立到设备的 ActiveSync 连接。
2. 在命令提示符处键入以下命令，其中 *certstorequery.xml* 为证书存储区查询 XML 文件：

   ```
   Rapiconfig.exe /P /M <certstorequery.xml>
   ```

3. 查看生成的 RapiConfigOut.xml 文件。

请参见

其他资源

设备项目中的安全性

如何：将 Signtool.exe 作为生成后事件启动（设备）

当其他生成后事件更改原始二进制文件时，可将 signtool.exe 作为生成后事件运行。对已签名的二进制文件所做的更改会使原始签名失效。

注意

显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

在 Visual Basic 和 Visual C# 设备项目中将 signtool.exe 作为生成后事件启动。

1. 在“解决方案资源管理器”中，右击项目，再单击快捷菜单上的“属性”。
2. 在“生成事件”页 (Visual C#) 或“编译”页 (Visual Basic) 上单击“编辑生成后事件”。
3. 在“生成后事件命令行”对话框中，键入带有所选选项的 signtool 命令行。

   有关从命令行运行 signtool 的更多信息，请参见 SignTool。

在 Visual C++ 设备项目中将 signtool.exe 作为生成后事件启动

1. 在“解决方案资源管理器”中，右击项目，再单击快捷菜单上的“属性”。
2. 在“配置属性”之下展开“生成事件”节点。
3. 单击“生成后事件”。
4. 选择“命令行”属性，再单击“省略号”按钮（“...”），打开“命令行”对话框。
5. 键入带有所选选项的 signtool 命令行。

   有关从命令行运行 signtool 的更多信息，请参见 SignTool。

请参见

其他资源

设备项目中的安全性

示例和演练(智能设备项目)

下面的示例项目阐释了用于解决各种设备编程问题的语法、结构和技术。

本节内容

智能设备示例

提供要检查的已生成项目。

智能设备演练

提供有关如何创建不同类型的应用程序和组件的分步介绍。既包括托管项目，也包括本机项目。

相关章节

Visual Studio 示例

提供演示应用程序的一些示例。

Visual Studio 演练

演示如何用各种编程语言创建不同类型的应用程序和组件。

请参见

其他资源

智能设备开发

Mobile Developer Center(移动开发人员中心)

智能设备演练

本主题已针对 Visual Studio 2005 SP1 进行了更新。

这些演练提供了如何在智能设备项目中实现各种任务的逐步骤说明。每个演练提供了特定类型的设备应用程序的开发步骤。如果不修改设备应用程序，为桌面应用程序设计的 Visual Studio 演练通常不起作用，这主要是因为 .NET Compact Framework 中缺少 .NET Framework 中可用的一个或多个类。

下面的演练中使用的默认目标设备是模拟器，因此不需要物理设备就可以完成演练。如果您有与开发计算机相连或通信的物理设备，您可以面向该设备而不是模拟器。有关更多信息，请参见智能设备项目的硬件和软件要求。

注意
一些界面元素（例如，“新建项目向导”）会遮蔽演练的文字。一种解决方法是将帮助查看器从 Visual Studio 集成开发环境 (IDE) 浏览器切换到外部窗口。为此，可以在“工具”菜单上单击“选项”，单击“环境”，然后单击“帮助”。选择“外部帮助”并重新启动 IDE 以使更改生效。

下表已针对 Visual Studio 2005 SP1 进行了更新。

本节内容

演练：创建用于设备的 Windows 窗体应用程序

描述如何开发一个简单的 Windows 窗体应用程序以及如何将其下载到智能设备。

演练：为设备创作用户控件

演示如何在托管设备项目中创作一个用户控件。

演练：向用户控件添加简单属性

描述如何在设备项目中自定义用户控件。

演练：数据库主/从应用程序

描述如何建立作为数据源的 SQL Server Mobile 或 SQL Server Compact Edition 数据库、如何将绑定到控件的数据对象拖动到 Windows 窗体上、如何配置主/从关系等。

演练：参数化查询应用程序

描述如何建立作为数据源的 SQL Server Mobile 或 SQL Server Compact Edition 数据库、如何将绑定到控件的数据对象拖动到 Windows 窗体上，以及如何在应用程序中包含参数化查询。

演练：Hello World：智能设备的 COM Interop 示例

演示托管项目如何可以承载 COM 对象。

如何：创建多平台设备项目 (Visual C++)

描述如何面向多个设备。

演练：为智能设备创建多平台 MFC 应用程序

描述如何针对多个目标平台生成 MFC 应用程序。

演练：为智能设备创建 MFC 多平台 ActiveX 控件

描述如何生成 MFC ActiveX 控件。

演练：为智能设备创建 ATL 多平台 ActiveX 控件

描述如何创建多平台智能设备 ATL 项目和简单 ATL 控件。

相关章节

演练：调试同时包含托管代码和本机代码的解决方案

描述交替启动本机和托管调试器的步骤。

演练：打包智能设备解决方案以便进行部署

描述如何生成 CAB 文件分发给最终用户用于在设备上进行安装，以及如何使用 IDE（而非手动配置 .inf 文件）来修改各种设

置。

智能设备示例

提供一些演示基于 .NET Compact Framework 的应用程序的完整项目，以及如何使用本机代码的示例的链接。

Visual Studio 演练

提供各种类型的桌面应用程序的逐步骤说明。

请参见

其他资源

示例和演练（智能设备项目）

演练：创建用于设备的 Windows 窗体应用程序

在此演练中，您将使用 Visual Basic 或 Visual C# 生成一个简单的 Windows 窗体应用程序，然后在 Pocket PC 仿真程序上运行该应用程序。此演练演示了桌面编程与设备编程之间的主要差异，也就是说，您必须指定目标设备。在此演练中，设备是指 Pocket PC 2003 的内置仿真程序。

注意
显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

此演练是使用 Visual Basic 开发设置和 Visual C# 开发设置编写的。

本演练由五个主要任务组成：

- 创建使用 Windows 窗体的设备项目
- 向窗体添加控件。
- 向控件添加事件处理。
- 选择运行项目的设备。
- 生成应用程序并将其部署到设备。

选择目标设备

为了确保在部署解决方案时系统提示您选择设备，请完成以下过程。

在部署时提示选择设备

1. 在“工具”菜单上，单击“选项”，单击“设备工具”，然后单击“常规”。（如果“设备工具”不可见，请选择“选项”对话框底部的“显示所有设置”。）
2. 选择“部署设备项目前显示设备选项”复选框。

创建应用程序

创建 Windows 窗体项目和添加控件及事件处理的过程对于设备项目和桌面项目来说是相同的。主要的差异是 .NET Compact Framework 中可用类的数量较少。

创建使用 Windows 窗体的设备项目

1. (Visual Basic) 在 Visual Studio 2005 中的“文件”菜单上，单击“新建项目”。

   - 或 -

   (Visual C#) 在 Visual Studio 2005 中的“文件”菜单上，指向“新建”，然后单击“项目”。
2. 在“新建项目”对话框中的“项目类型”下，展开“Visual Basic”或“Visual C#”，展开“智能设备”，然后单击“Pocket PC 2003”。

   如果开始并未出现您需要的语言，请展开“其他语言”。此显示由开发设置进行控制。
3. 在“模板”下，单击“设备应用程序”。
4. 在“名称”框中，键入“DeviceSample”，然后单击“确定”。
5. （仅适用于 Visual C#）在“位置”框中，确认要用于存储项目文件的位置，然后单击“确定”。

   Pocket PC 设备的一种表示形式将显示在 Windows 窗体设计器中。

向窗体添加控件

1. 将一个“Button”控件从“工具箱”中拖到窗体上。

   如果“工具箱”不可见，请在“视图”菜单上单击“工具箱”。

   如果在“工具箱”中“设备控件”选项卡不可见，请右击“工具箱”，然后单击“全部显示”。

2. 右击“Button”控件，再单击“属性”。
3. 在“属性”窗口中，键入 **Say Hello**，然后按 Enter 设置“Text”属性。

为 Button 控件添加事件处理

1. 双击窗体上的按钮。

   随即打开代码编辑器，并且插入点位于事件处理程序中。

2. 插入以下 Visual Basic 代码：

   ```
   MessageBox.Show("Hello, World!")
   ```

   - 或 -

   插入以下 C# 代码：

   ```
   MessageBox.Show("Hello, World!");
   ```

生成并测试应用程序

此时，您会发现与桌面项目的不同之处。在设备项目中，通常可以从若干个目标之间选择运行项目的位置。在此演练中，选择 Pocket PC 仿真程序。如果开发计算机中已经有一个受支持的物理设备，也可以选择该物理设备。

生成并测试应用程序

1. 在“调试”菜单上，单击“开始”（或“开始调试”）。
2. 在“部署”对话框中，选择“Pocket PC 2003 SE 仿真程序”，然后单击“部署”。

   可以在进度栏中查看进度。

3. 当应用程序运行于仿真程序上时，点击按钮以确保出现“Hello, World!”。

准备进行其他演练

如果您打算进行其他演练或打开其他项目，则需要完全关闭该仿真程序并退出此解决方案。

关闭仿真程序和解决方案

1. 在仿真程序的“文件”菜单上单击“退出”。
2. 在“设备仿真程序”消息框中，对“退出前保存状态?”消息单击“否”。
3. 在 Visual Studio 的“调试”菜单上，单击“停止调试”。
4. 如果出现指示丢失连接的消息，则单击“确定”。
5. 在“文件”菜单上单击“关闭解决方案”。

请参见

任务

演练：创建简单的 Windows 窗体

参考

“选项”对话框 ->“设备工具”->“常规”

工具箱

其他资源

智能设备演练

演练：为设备创作用户控件

在本演练中，您将创建一个控件库以及一个要放入该库中的用户控件。用户控件是单个可重用单元中的 Windows 窗体控件的组合，自定义控件是需要无法通过标准控件实现的 UI 或功能的控件，您应当将两者区分开来。

本演练中的用户控件是一个用于设备的简单时钟显示工具，该控件是在完成类似的桌面演练（例如演练：使用 Visual Basic 创作复合控件和演练：使用 Visual C# 创作复合控件）之后建模的。

本演练由四项主要任务组成：

- 创建一个控件库和一个控件。
- 重命名该库和控件。
- 向该控件添加组件。
- 在设备仿真程序上测试控件。

您可以在本演练中使用 Visual Basic 或 Visual C#。虽然两种语言使用相同的文件名，但具有各自的语言特定扩展名，为此，本演练使用一条竖线来提示您根据使用的语言选择相应的扩展名，如 **filename.vb|cs**。

注意
显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。 此演练是使用 Visual Basic 开发设置和 Visual C# 开发设置编写的。

先决条件

无。

选择目标设备

为了确保在部署解决方案时系统提示您选择设备，请完成以下过程。

在部署时提示选择设备

1. 在“工具”菜单上，依次单击“选项”、“设备工具”和“常规”。

   （如果“设备工具”不可见，请选择“选项”对话框底部的“显示所有设置”。）

2. 选中“部署设备项目前显示设备选项”复选框。

创建项目

指定给新项目的名称同时还设置根命名空间和程序集名称。

创建控件库和控件

1. (Visual Basic) 在“文件”菜单上单击“新建项目”。

   - 或 -

   (Visual C#) 在“文件”菜单上指向“新建”，然后单击“项目”。

2. 在“新建项目”对话框中的“项目类型”下，展开“Visual Basic”或“Visual C#”，再展开“智能设备”，然后单击“Pocket PC 2003”。

   如果开始并未出现您需要的语言，请展开“其他语言”。此显示由开发设置控制。

3. 在“模板”之下单击“控件库”。
4. 在“名称”框中键入“ctlDevClockLib”。
5. （仅适用于 Visual C#）在“位置”框中，确认要用于存储项目文件的位置。
6. 单击“确定”。

   此时会显示“组件设计器”。

您已经指定了项目名、根命名空间和程序集名称 (ctlDevClockLib)。但是，项目中的组件具有 Visual Studio 指定的默认名称，例如，"UserControl1"。通常，你需要将这些名称更改为更有意义的术语。

重命名该库和控件

1. 在"解决方案资源管理器"中，右击"UserControl1.vb|cs"，然后单击上下文菜单中的"属性"。
2. 将"文件名"更改为**"ctlDevClock.vb|cs"**。
3. （仅限 Visual C#）在询问是否要重命名对此代码元素的所有引用的消息框中，单击"是"。

   "属性"窗口中的名称更改现在传播到了其他引用，如类名和构造函数。

接下来，从"工具箱"中添加组件，以便为您的用户控件提供功能和用户交互。在本演练中，您添加一个访问系统时间的**"计时器"**控件和一个显示时间的**"标签"**控件。

添加组件并更改它们的属性

1. 在"工具箱"中，双击"标签"。

   将向"组件设计器"中的用户控件添加一个标签控件。
2. 在该标签控件的"属性"窗口中，执行下列操作：

属性	更改为
Name	**lblDisplay**
Text	（空白）
TextAlign	TopCenter
Font.Size	14

1. 在"工具箱"中，双击"计时器"。

   组件栏中会出现一个**"计时器"**控件。
2. 在该计时器控件的"属性"窗口中，执行下列操作：

属性	更改为
Interval	1000
Enabled	True

1.

在下面的步骤中，添加一个事件处理程序以便在**"标签"**控件中显示时钟刻度。

添加事件处理程序

1. 在组件栏中，双击"timer1"打开代码编辑器（此时编辑器显示 tick 事件）。
2. (Visual Basic) 插入下面的事件处理代码：

   ```
   lblDisplay.Text = Format(Now, "hh:mm:ss")
   ```

   - 或 -

   (Visual C#)

   ```
   lblDisplay.Text = DateTime.Now.ToLongTimeString();
   ```

3. (Visual Basic) 将访问修饰符从 **Private** 更改为 **Protected** 并添加 **Overridable** 关键字，使处理程序代码如下所示：

```
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
```

- 或 -

(Visual C#) 将访问修饰符从 **private** 更改为 **protected** 并添加 **virtual** 关键字，使处理程序代码如下所示：

```
protected virtual void timer1_Tick(object sender, System.EventArgs
   e)
{
   // Causes the label to display the current time.
   lblDisplay.Text = DateTime.Now.ToLongTimeString();
}
```

4. 在“文件”菜单上单击“全部保存”。
5. （仅适用于 Visual Basic）在“保存项目”对话框中，将项目以**“ctlDevClockLib”**的名称保存到您选择的位置。

测试控件

该简单设备应用程序用作您的控件的测试容器。

生成此控件并创建一个测试容器

1. 在“生成”菜单上单击“生成”（或“生成 ctlDevClockLib”）。
2. 在“文件”菜单上指向“添加”，然后单击“新建项目”。
3. 在“添加新项目”对话框中，单击“项目类型”窗格中的“Pocket PC 2003”，然后单击“模板”窗格中的“设备应用程序”。
4. 在“名称”框中键入“Test”，然后单击“确定”。
5. 在“解决方案资源管理器”中，右击“Test”项目，然后单击“设为启动项目”。
6. 再次右击“Test”项目，然后单击“添加引用”。
7. 在“添加引用”对话框中，单击“项目”选项卡，然后双击“ctlDevClockLib”。
8. 在“工具箱”中，找到“ctlDevClockLib 组件”选项卡，然后双击“ctlDevClock”组件。

   此控件即加载到设计器中。
9. 在“调试”菜单上，单击“开始”（或“开始调试”）。
10. 在“部署”对话框中，选择“Pocket PC 2003 SE 仿真程序”，然后单击“部署”。

请参见

概念

控件类型建议

其他资源

设计时开发 Windows 窗体控件

演练：向用户控件添加简单属性

此演练演示如何向设备项目中的用户控件添加属性。具体地说，添加使控件属性 (property) 在设计时可见的自定义属性 (attribute)。您可能希望在项目中添加此功能以防止属性值被更改。

该过程与桌面的对应功能相似，只是设备项目将此信息存储在单独的元数据文件 (.xmta) 中。

注意

显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

此演练是使用 Visual C# 开发设置编写的。

创建 UserControl1 类

1. 在“文件”菜单上指向“新建”，然后单击“项目”。
2. 在“项目类型”窗格中，依次展开“Visual C#”和“智能设备”，然后单击“Pocket PC 2003”。
3. 在“模板”窗格中，单击“控件库”。
4. 在“名称”框中，键入**“MyControlLibrary”**，然后单击“确定”。

   设计器将打开，并显示一个表示新建用户控件类的正方形。

添加属性

1. 在“解决方案资源管理器”中，右击“UserControl1.cs”，然后在快捷菜单上单击“查看类关系图”。

   将打开表示类关系图的圆角矩形。
2. 右击该类关系图，然后在快捷菜单上单击“类详细信息”。
3. 在“类详细信息”窗口的“属性”部分中，在“<添加属性>”提示下，键入 **MyProperty**。
4. 在“类型”列中，用“string”替换“int”。
5. 右击 MyProperty 行开头的图标，然后在快捷菜单上单击“属性”。
6. 若要指定“自定义属性”属性的值，请单击省略号按钮 (...) 以打开“自定义属性”对话框。
7. 键入 **Browsable(false)**，然后单击“确定”。

   “解决方案资源管理器”中显示包含该自定义属性的设计时属性 .xmta 文件 (DesignTimeAttributes.xmta)。

生成控件库

1. 在“解决方案资源管理器”中，右击“UserControl1.cs”，然后在快捷菜单上单击“查看代码”。
2. 注释掉引发 `System.NotImplementedException` 的行，改为插入 `return "";` 作为 `get` 操作。
3. 在“生成”菜单上单击“生成 MyControlLibrary”。

测试 MyProperty 是否未显示在属性浏览器中

1. 在“解决方案资源管理器”中，右击“MyControlLibrary”，在快捷菜单上指向“添加”，然后单击“新建项”.
2. 在“添加新项”对话框中，选择“Windows 窗体”，然后单击“添加”。
3. 将“UserControl1”从“工具箱”拖到窗体上。
4. 右击该窗体上的用户控件图像，然后在快捷菜单上单击“属性”。

   “MyProperty”未显示在“属性”浏览器中。
5. 在解决方案资源管理器中，双击该 .xmta 文件，然后将“false”替换为“true”。

6. 重复这些步骤，以查看“属性”网格。注意，“MyProperty”现在显示出来。

请参见

其他资源

智能设备演练

演练：数据库主/从应用程序

本主题已针对 Visual Studio 2005 SP1 进行了更新。

此演练显示如何使用 Visual Studio 2005 环境连接到数据库、选择要包含在项目中的数据库对象和创建数据绑定控件以便在智能设备应用程序中显示数据。

注意
显示的对话框和菜单命令可能会与“帮助”中描述的不同，具体取决于当前的设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。 此演练是使用 Visual Basic 开发设置和 Visual C# 开发设置编写的。

先决条件

SQL Server Mobile Edition 的 Northwind 数据库（包含在 Visual Studio 2005 中）。

Microsoft SQL Server 2005 Compact Edition 的 Northwind 数据库（安装 SQL Server Compact Edition 以用于 Visual Studio 2005 SP1 时已包括）。

注意
如果您不是开发计算机的管理员，则不能在“Northwind.sdf”文件的默认位置 (\Program Files\Microsoft Visual Studio 8\Smart Devices\SDK\SQL Server\Mobile\v3.0) 打开该文件。请在提示您时，将该文件复制到桌面上或“我的文档”中，然后打开它。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

选择目标设备

为了确保在部署解决方案时系统提示您选择设备，请完成以下过程。

在部署时提示选择设备

1. 在“工具”菜单上，单击“选项”，单击“设备工具”，然后单击“常规”。
2. 选择“部署设备项目前显示设备选项”复选框。

创建应用程序

这是一个简单的 Windows 窗体应用程序，用于承载此演练的数据功能。

创建 Windows 窗体设备项目

1. (Visual Basic) 在 Visual Studio 2005 中的“文件”菜单上，单击“新建项目”。

   - 或 -

   (Visual C#) 在 Visual Studio 2005 中的“文件”菜单上，指向“新建”，然后单击“项目”。
2. 在“新建项目”对话框中的“项目类型”下，展开“Visual Basic”或“Visual C#”，展开“智能设备”，然后单击“Pocket PC 2003”。

   如果开始并未出现您需要的语言，请展开“其他语言”。此显示由开发设置进行控制。
3. 在“模板”下，单击“设备应用程序”。

注意
请不要选择“设备应用程序(1.0)”，该选项依赖于 .NET Compact Framework 的版本 1，不适用于此演练。

4. 在“名称”框中键入“MasterDetailSample”。
5. （仅适用于 Visual C#）在“位置”框中，确认要用于存储项目文件的位置。
6. 单击“确定”。

   Pocket PC 设备的一种表示形式将显示在 Windows 窗体设计器中。

添加数据功能

本节包括以下任务:

- 选择数据源类型
- 选择并配置数据连接
- 选择数据库对象
- 向窗体添加数据绑定控件

选择数据源类型

1. 在“数据”菜单上，单击“添加新数据源”打开“数据源配置向导”。
2. 在“选择数据源类型”页上，选择“数据库”，然后单击“下一步”。

选择并配置数据连接

1. 在“选择您的数据连接”页上，单击“新建连接”。
2. 在“选择数据源”对话框中，选择“Microsoft SQL Server Mobile Edition”，然后单击“继续”。

   - 或 -

   在“选择数据源”对话框中，选择“Microsoft SQL Server Compact Edition”，然后单击“继续”。

注意
根据设置和早期项目，可能会显示“添加连接”对话框而不是“选择数据源”对话框。如果出现这种情况，请在“添加连接”对话框中单击“更改”打开“更改数据源”对话框。然后选择“Microsoft SQL Server Mobile Edition”或“Microsoft SQL Server Compact Edition”，并单击“确定”。

3. 在“添加连接”对话框中，选择“我的电脑”。
4. 同样在“添加连接”对话框中，单击“浏览”。
5. 在“选择 SQL Server Mobile Edition 数据库文件”对话框中，选择“Northwind.sdf”，然后单击“打开”。

   - 或 -

   在“选择 SQL Server Compact Edition 数据库文件”对话框中，选择“Northwind.sdf”，然后单击“打开”。
6. 在“添加连接”对话框中，保留“密码”框为空。

   此数据库没有密码。
7. 单击“测试连接”验证该连接。

注意
如果访问“Northwind.sdf”文件被拒绝，请将该文件复制到桌面上，然后浏览到该副本以打开它。如果在开发计算机上没有足够的权限以在该文件的默认位置（列于此演练的开头部分）打开该文件，则可能发生此情况。

8. 在显示连接已成功的消息框中单击“确定”，然后单击“确定”关闭“添加连接”对话框。
9. 单击“下一步”，关闭“选择您的数据连接”页。
10. 在询问是否要将文件复制到项目的消息框中，单击“确定”。

选择数据库对象

1. 在“选择数据库对象”页上，展开“表”节点，然后选择“Customers”和“Orders”表。
2. 单击“完成”。

   此时已创建 NorthwindDataset。在“数据”菜单上选择“显示数据源”，即可查看此数据源。

向窗体添加数据绑定控件

1. 在“数据源”窗口中选择“Customers”表，单击下拉箭头并选择“DataGrid”选项。
2. 将“Customers”表从“数据源”窗口拖到设计器中的窗体上。

   在放置该网格时，使其朝向窗口顶部。
3. 在“数据源”窗口中，展开“Customers”表以显示“Orders”表

注意
这是“Customers”表中显示的“Orders”表，而不是与“Customers”表在树中处于同一级的“Orders”表。

4. 单击此“Orders”表的下拉箭头，然后选择“DataGrid”选项。
5. 将此“Orders”表从“数据源”窗口拖到设计器中的窗体上。

   在放置该网格时，使其朝向窗口底部。

测试应用程序

在本节中，您将生成应用程序，将其下载到 Pocket PC 2003 SE 仿真程序，然后验证该应用程序是否工作正常。

测试应用程序

1. 在“调试”菜单上，单击“开始”或“开始调试”。
2. 在“部署”对话框中，选择“Pocket PC 2000 SE 仿真程序”，然后单击“部署”。

   “状态”栏中会显示部署进度。部署到仿真程序中可能需要较长时间。
3. 当应用程序在仿真程序中运行时，使用键盘上的向上键和向下键或仿真程序上的导航控件更改“Customers”网格中的选定记录。验证选定记录在“Orders”网格中是否已被更改。

准备进行其他演练

如果您打算进行其他演练或打开其他项目，则需要完全关闭该仿真程序并退出此解决方案。

关闭仿真程序和解决方案

1. 在仿真程序的“文件”菜单上单击“退出”。
2. 在“设备仿真程序”消息框中，单击“否”答复询问您是否要保存仿真程序状态的消息。
3. 在提示连接丢失的消息框中，单击“确定”。
4. (Visual Basic) 在“文件”菜单上单击“关闭项目”。

   在提示您保存该项目或解决方案时，如果希望今后还要使用它，请单击“保存”；否则，单击“放弃”，这将不保存您的文件。

   - 或 -

   (Visual C#) 在“文件”菜单上单击“关闭解决方案”。

请参见

任务

如何：安装示例数据库

参考

数据源配置向导

其他资源

数据演练

访问数据 (Visual Studio)

演练:参数化查询应用程序

本主题已针对 Visual Studio 2005 SP1 进行了更新。

此演练显示如何使用 Visual Studio 2005 环境开发简单的参数化查询应用程序。将为您自动生成数据绑定和大部分的用户界面。此应用程序依赖于您熟悉的 Northwind 数据库，专为智能设备用户需要在仅知道订单号时确定装运国家/地区的情况而提供。使用此处生成的应用程序，用户可输入订单号，且随之显示对应的装运国家/地区。

注意
显示的对话框和菜单命令可能会与“帮助”中描述的不同，具体取决于当前的设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。 此演练是使用 Visual Basic 开发设置和 Visual C# 开发设置编写的。

以下过程已针对 Visual Studio 2005 SP1 进行了更新。

安装的数据库会根据您已安装的 Visual Studio 的版本不同而异：

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

先决条件

SQL Server Mobile 或 SQL Server Compact Edition 的 Northwind 数据库（包含在 Visual Studio 2005 中）。

注意
如果您不是开发计算机上的管理员，则不能在“Northwind.sdf”文件的默认位置 (\Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SQL Server\Mobile\v3.0) 打开该文件。请将该文件复制到桌面或“我的文档”，然后在提示您时在该位置中打开它。

选择目标设备

为了确保在部署解决方案时系统提示您选择设备，请完成以下过程。

在部署时提示选择设备

1. 在“工具”菜单上，单击“选项”，单击“设备工具”，然后单击“常规”。
2. 选择“部署设备项目前显示设备选项”复选框。

创建应用程序

这是一个简单的 Windows 窗体应用程序，用于承载此演练的数据功能。

创建 Windows 窗体设备项目

1. (Visual Basic) 在 Visual Studio 2005 中的“文件”菜单上，单击“新建项目”。

   - 或 -

   (Visual C#) 在 Visual Studio 2005 中的“文件”菜单上，指向“新建”，然后单击“项目”。
2. 在“新建项目”对话框中的“项目类型”下，展开“Visual Basic”或“Visual C#”，展开“智能设备”，然后单击“Pocket PC 2003”。

   如果开始并未出现您需要的语言，请展开“其他语言”。此显示由开发设置进行控制。
3. 在“模板”下，单击“设备应用程序”。

注意
请不要选择“设备应用程序 (1.0)”，该选项依赖于 .NET Compact Framework 的版本 1.0，不适用于此演练。

4. 在“名称”框中，键入 **ParamQuerySample**。

5. (仅适用于 Visual C#)在"位置"框中，确认要用于存储项目文件的位置。
6. 单击"确定"。

   Pocket PC 设备的一种表示形式将显示在 Windows 窗体设计器中。

添加数据功能

本节包括以下任务:

- 选择数据源类型。
- 选择并配置数据连接。
- 选择数据库对象。
- 向窗体添加数据绑定控件。

选择数据源类型

1. 在"数据"菜单上，单击"添加新数据源"打开"数据源配置向导"。
2. 在"选择数据源类型"页上，选择"数据库"，然后单击"下一步"。

选择并配置数据连接

1. 在"选择您的数据连接"页上，单击"新建连接"。
2. 在"选择数据源"对话框中，选择"Microsoft SQL Server Mobile Edition"，然后单击"继续"。

   - 或 -

   在"选择数据源"对话框中，选择"Microsoft SQL Server Compact Edition"，然后单击"继续"。

注意
根据设置和早期项目，可能会显示"添加连接"对话框而不是"选择数据源"对话框。如果出现这种情况，请在"添加连接"对话框中单击"更改"打开"更改数据源"对话框。然后选择"Microsoft SQL Server Mobile Edition"或"Microsoft SQL Server Compact Edition"，并单击"确定"。

3. 在"添加连接"对话框中，选择"我的电脑"。
4. 同样在"添加连接"对话框中，单击"浏览"。
5. 在"选择 SQL Server Mobile Edition 数据库文件"对话框中，选择"Northwind.sdf"，然后单击"打开"。

   - 或 -

   在"选择 SQL Server Compact Edition 数据库文件"对话框中，选择"Northwind.sdf"，然后单击"打开"。
6. 在"添加连接"对话框中，保留"密码"框为空。

   此数据库没有密码。
7. 单击"测试连接"验证该连接。

注意
如果访问"Northwind.sdf"文件被拒绝，请将该文件复制到桌面上，然后浏览到该副本以打开它。如果在开发计算机上没有足够的权限以在该文件的默认位置(列于此演练的开头部分)打开该文件，则可能发生此情况。

8. 在显示连接已成功的消息框中单击"确定"，然后单击"确定"关闭"添加连接"对话框。
9. 单击"下一步"，关闭"选择您的数据连接"页。
10. 在询问是否要将文件复制到项目的消息框中，单击"确定"。

选择数据库对象

1. 在“选择数据库对象”页上，展开“表”节点，然后选择“Orders”表。
2. 单击“完成”。

   此时已创建 NorthwindDataset。在“数据”菜单上选择“显示数据源”，即可查看此数据源。

创建查询

1. 在“数据源”窗口中，展开“Orders”表。
2. 单击“Ship Country”列，单击下拉箭头，再选择“标签”选项。
3. 将“Ship Country”列拖到设计器中的窗体上。
4. 在设计器中的标签控件上，单击智能标记，然后在快捷菜单上单击“添加查询”。
5. 在“查询标准生成器”对话框中，单击“查询生成器”。
6. 在“Order ID”行的“Filter”列中，键入问号 (**?**)。

   此符号指示应用程序用户将必须输入 Order ID 的值。
7. 单击“确定”。

   此时，“查询文本”框中的 `WHERE` 子句应读取 `([Order ID]=@PARAM1)`。
8. 单击“确定”关闭“查询标准生成器”对话框。

   一个面板将显示在设计器中的窗体上。

改进用户界面

1. 在设计器中右击“PARAM1”标签控件，然后在快捷菜单上单击“属性”。
2. 将“Text”属性更改为订单 **ID**。
3. 选择“FillBy”按钮，然后将其文本属性更改为“显示国家/地区”。
4. 展开该面板和控件，以消除滚动栏并显示所有文本。要特别小心，“Ship_CountryLabel”及其文本框没有隐藏在“FillByPanel”及其控件的后面。

测试应用程序

在本节中，您将生成应用程序，将其下载到 Pocket PC 2003 SE 仿真程序，然后验证该应用程序是否工作正常。

测试应用程序

1. 在“调试”菜单上，单击“开始”或“开始调试”。
2. 在“部署”对话框中，选择“Pocket PC 2000 SE 仿真程序”，然后单击“部署”。

   “状态”栏中会显示部署进度。部署到仿真程序中可能需要较长时间。
3. 当该应用程序在仿真程序上运行时，请键入订单号（在 Northwind 数据库中，该订单号为 10000 到 11077），然后单击“显示国家/地区”。

   标签控件中将显示该订单的装运国家/地区。

准备进行其他演练

如果您打算进行其他演练或打开其他项目，则需要完全关闭该仿真程序并退出此解决方案。

关闭仿真程序和解决方案

1. 在仿真程序的“文件”菜单上单击“退出”。
2. 在“设备仿真程序”消息框中，单击“否”答复询问您是否要保存仿真程序状态的消息。
3. (Visual Basic) 在“文件”菜单上单击“关闭项目”。

   - 或 -

(Visual C#) 在“文件”菜单上单击“关闭解决方案”。

在提示您保存该项目或解决方案时，如果希望今后还要使用它，请单击“保存”；否则，单击“放弃”，这将不保存您的文件。

请参见

任务

如何：创建参数化查询（设备）

如何：安装示例数据库

参考

数据源配置向导

其他资源

智能设备演练

托管设备项目中的数据

演练：Hello World：智能设备的 COM Interop 示例

本演练将一个简单的 COM 对象和一个托管应用程序合并到一个解决方案中。

注意

显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

此演练是使用 Visual C++ 开发设置编写的。

创建 COM 对象

创建智能设备 ATL 项目

1. 在“文件”菜单上指向“新建”，单击“项目”，在“项目类型”窗格中展开“Visual C++”节点，然后单击“智能设备”。
2. 在“模板”窗格中单击“ATL 智能设备项目”。
3. 在“名称”框中，键入**“HelloCOMObject”**。
4. 在“解决方案名称”框中键入**“InteropSolution”**。
5. 单击“确定”启动“ATL 智能设备项目向导”。
6. 单击“完成”关闭向导。

   对于本演练，您不需要更改向导中的任何默认设置。

添加类

1. 在“解决方案资源管理器”中，右击“HelloCOMObject”项目，指向“添加”，然后单击“类”打开“添加类”对话框。
2. 在“类别”窗格中单击“智能设备”。
3. 在“模板”窗格中单击“ATL 简单对象”，然后单击“添加”打开“ATL 简单对象向导”。
4. 在“简称”框中键入**“Hello”**。
5. 在左侧窗格中，单击“选项”打开“选项”页。
6. 在“线程模型”下，选择“自由”，然后单击“完成”。

向类添加方法

1. 从桌面上的选项卡或从“视图”菜单中打开“类视图”窗口。
2. 展开“HelloCOMObject”以显示“IHello”界面。
3. 右击“IHello”，指向“添加”，然后单击“添加方法”打开“添加方法向导”。
4. 在“方法名称”框中，键入**“HelloWorld”**。
5. 在“参数类型”框中，选择“BSTR*”。
6. 在“参数名”框中键入**“text”**。
7. 在“参数属性”下，选择“out”。
8. 单击“添加”。

   方法框将显示 `[out] BSTR* text`。
9. 单击“完成”关闭“添加方法向导”。

   方法 `STDMETHOD(HelloWorld)(BSTR* text)` 显示在文件 Hello.h 中。

向方法添加实现

1. 在“解决方案资源管理器”中双击“Hello.cpp”，在代码编辑器中打开该文件。
2. 在 STDMETHODIMP 节中的返回语句之前插入以下实现代码：

```
*text = SysAllocString(L"Hello World!");
```

3. 在“生成”菜单上单击“生成 HelloCOMObject”。

   COM 对象现在已成为解决方案的一部分。至此，本演练的第一部分已完成。

创建托管项目

向解决方案添加托管项目

1. 在“解决方案资源管理器”中右击“InteropSolution”，指向“添加”，然后单击“新建项目”。
2. 在“项目类型”窗格中，定位到 Visual C# 节点，展开该节点，再展开“智能设备”节点，然后单击“Pocket PC 2003”。
3. 在“模板”窗格中，单击“设备应用程序”。
4. 在“名称”框中，键入**“SayHello”**，然后单击“确定”。

   SayHello 托管项目即创建为解决方案的一部分，并且“设计器”窗口中出现一个 Pocket PC 窗体。

将 COM 对象作为托管项目中的引用添加

将 COM 对象作为托管项目中的引用添加

1. 在“解决方案资源管理器”中，右击“SayHello”项目，然后单击快捷菜单上的“添加引用”。
2. 在“添加引用”对话框中，单击“浏览”。

   “SayHello”文件夹即显示在“查找范围”框中。
3. 定位到父文件夹（在本演练中为“InteropSolution”）。
4. 在显示文件夹内容的窗口中，依次双击“HelloCOMObject”、“Pocket PC 2003 (ARMV4)”、“调试”，然后单击“HelloCOMObject.dll”。
5. 单击“确定”关闭“添加引用”对话框。
6. 在“解决方案资源管理器”中，右击“Form1.cs”，然后在快捷菜单上单击“查看代码”。
7. 在文件顶部的 Using directives 区域，添加以下代码：

```
using HelloCOMObjectLib;
```

向托管项目添加事件处理

向托管项目添加事件处理并生成该事件处理

1. 打开 Form1 设计器。
2. 将一个按钮从“工具箱”中拖动到窗体上。
3. 双击该按钮打开代码编辑器（此时编辑器显示 click 事件）。
4. 为该按钮插入以下事件处理代码：

```
string text;
HelloClass h = new HelloClass();
h.HelloWorld(out text);
MessageBox.Show(text);
```

5. 在“生成”菜单上单击“生成 SayHello”。

对解决方案进行最后的调整

配置解决方案以进行部署

1. 在“解决方案资源管理器”中，右击“SayHello”项目，然后单击快捷菜单上的“设为启动项目”。
2. 在“解决方案资源管理器”中，右击“InteropSolution”，然后单击快捷菜单上的“项目依赖项”。
3. 在“项目依赖项”对话框的“项目”框中选择“SayHello”，然后在“依赖于”框中选择“HelloCOMObject”。
4. 单击“确定”。

   解决方案此时已可以进行部署。

部署混合解决方案

部署解决方案

1. 在“调试”菜单上单击“开始执行(不调试)”。
2. 在“部署”对话框中，选择“Pocket PC 2003 SE 仿真程序”，然后单击“部署”。

保存该解决方案，以便在演练：调试同时包含托管代码和本机代码的解决方案中使用。

请参见
概念
设备的 COM 互操作性
其他资源
智能设备演练

如何：创建多平台设备项目 (Visual C++)

使用针对设备的 C++ 可以在同一开发项目中面向多种设备。使用 Visual Studio 2005，可以创建面向多种设备平台（如 Pocket PC 和 Smartphone）的单一设备项目，还可以将设备项目添加至桌面项目。有关更多信息，请参见桌面项目的设备支持。

有两种将设备项目设置为面向多种平台的方式：

1. 在创建项目时使用“智能设备项目向导”。这是一项更加高效的技术。

   “智能设备项目向导”为选定的所有设备平台生成项目文件，例如，源文件、头文件和资源文件。通过安装平台或设备的 SDK，可以安装其他新平台。

   使用 Visual Studio 2005，可以安装支持 Smartphone 或 Pocket PC 2003 或更高版本以及任何 Windows CE 设备 5.0 版或更高版本的任何设备 SDK。有关更多信息，请参见如何：使用向导创建多平台设备项目。

2. 在创建项目后，使用“智能设备项目向导”的“配置管理器设置”对话框的“新建平台”窗格。有关更多信息，请参见如何：向设备项目添加新平台。

请参见

任务

如何：更改默认设备（本机项目）

使用多个平台上的资源

其他资源

How to Maintain a Single Binary for Pocket PC and Smartphone（如何为 Pocket PC 和 Smartphone 维护单一的二进制代码）

如何：使用向导创建多平台设备项目

在创建项目时使用“智能设备应用程序向导”创建多平台设备项目，要比创建了项目后再添加平台具有更多的优点。这些优点如下：

- 在应用程序向导的“平台”页上选择多种平台时，将为每一平台生成并配置资源文件。但是，如果您在创建项目后添加平台，则需要手动添加平台和资源文件。有关更多信息，请参见使用多个平台上的资源。
- 当您在应用程序向导中的“平台”页上选择多个平台时，生成的文件会在源中嵌入 `#if defined` 语句，以适应多平台编码。但是，如果您在创建项目后添加平台，则需要手动将 `#if defined` 语句添加到代码的相应节中。

下面的过程演示如何使用其中的每一种技术。有关更多信息，请参见代码解释：Visual C++ 设备项目中由向导生成的代码。

注意

显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

使用新建项目向导创建多平台设备项目

1. 在“文件”菜单上指向“新建”，然后单击“项目”。

   出现“新建项目”对话框。

2. 在“项目类型”之下展开“Visual C++”节点，单击“智能设备”，然后单击“MFC 智能设备应用程序”或者单击另一个可用的项目类型。

3. 在“名称”框中键入**“MultiPlatformProject”**。

4. 在“位置”框中验证您要存储项目文件的位置，然后单击“确定”。

   出现“<Project Type> 智能设备应用程序向导”。

5. 单击“下一步”，然后在“平台”窗格上选择您要面向的平台，如“Pocket PC 2003”和“Smartphone 2003”。

6. 单击“完成”。

   将在“解决方案资源管理器”中显示您的项目。

使用配置管理器添加新设备配置

1. 在 Visual Studio“生成”菜单上单击“配置管理器”。

   出现“配置管理器”对话框。

2. 在“活动解决方案配置”下拉列表上选择“新建”以添加新配置。

   即会出现“新建解决方案配置”对话框。

3. 在“名称”框中，键入新配置的名称。

4. 在“从此处复制设置”框中，指定一个现有配置。

5. 单击“确定”。

请参见

任务

如何：向设备项目添加新平台

如何：更改默认设备（本机项目）

使用多个平台上的资源

其他资源

How to Maintain a Single Binary for Pocket PC and Smartphone（如何为 Pocket PC 和 Smartphone 维护单一的二进制代码）

如何：向设备项目添加新平台

以下过程演示如何将平台添加到现有的设备项目中。

除了使用向导从头创建多平台项目之外，还可以使用这种方法创建多平台项目。使用向导是一种高级方法。有关更多信息，请参见如何：使用向导创建多平台设备项目。

注意

显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

使用配置管理器添加新的平台

1. 在设备项目的“生成”菜单上单击“配置管理器”。

   出现“配置管理器”对话框。

2. 在“活动解决方案平台”下拉列表中选择“新建”以添加新的配置。

   出现“新建解决方案平台”对话框。

3. 在“名称”下选择或键入要添加的平台，例如“Smartphone 2003”。.

4. 您可以从“从此处复制设置”下拉列表中选择现有平台来为新平台复制设置，或用空设置来创建新的平台。对于后一种情况，必须手动设置项目设置。

5. 确保选中“创建新的项目平台”复选框。

   如果未选中此复选框，您就必须管理和添加配置文件、向导可以为您生成的与平台无关的任何代码以及资源等其他项目文件。

6. 单击“确定”。

   “新建解决方案平台”对话框为新添加的平台生成项目文件，如源文件、头文件和资源文件。

7. 单击“确定”以关闭“配置管理器”对话框。

   注意

   创建设备项目后，您可以在第一次创建之后不断添加更多平台。但是，第一次创建之后将新的平台添加到项目并不会将其他依赖运行时 DLL 添加到“附加文件”配置属性中。例如，如果应用程序动态链接到 MFC，则需要为新添加的平台将以下 DLL 添加到“附加文件”属性中：`Mfc80u.dll`、`Atl80.dll` 和 `Msvcr80.dll`.（此示例假设为发布配置）。

请参见

任务

如何：更改默认设备（本机项目）

使用多个平台上的资源

演练：为智能设备创建多平台 MFC 应用程序

可以使用 Visual C++ 编写面向多种设备的代码。下面的演练阐释如何生成多平台 MFC 应用程序。有关更多信息，请参见 MFC 智能设备应用程序向导。

创建 MFC 多平台项目

本演练由三项主要任务组成：

- 创建多平台智能设备 MFC 项目。
- 向多平台 **OnDraw()** 方法添加代码。
- 部署多平台解决方案。

注意

显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

本演练使用 Visual C++ 开发设置编写。

创建多平台智能设备 MFC 项目

1. 在“文件”菜单上指向“新建”，单击“项目”，在“项目类型”窗格中展开“Visual C++”节点，然后单击“智能设备”。
2. 在“模板”窗格中，单击“MFC 智能设备应用程序”。
3. 在“名称”框中键入“HelloMFC”。
4. 单击“确定”以启动“MFC 智能设备应用程序向导”。
5. 单击“下一步”以选择要添加到当前项目的 Platform SDK。
6. 从“已安装的 SDK”窗格中选择要添加到当前项目中的平台，例如“Smartphone 2003”和“Pocket PC 2003”。
7. 单击“下一步”以打开“应用程序类型”页。
8. 选中“单文档”和“在静态库中使用 MFC”复选框。保持“文档/视图结构支持”复选框处于选中状态。
9. 单击“完成”以完成并关闭向导，或者单击“下一步”接受向导中的所有剩余选项的默认值。

注意

如果已创建了设备项目，您仍然可以添加其他平台；但是，在初始创建后向项目添加新的平台不会向所添加平台的“附加文件”配置属性添加附加的依赖运行时 DLL。例如，如果您的应用程序动态链接到 MFC，则需要在所添加新平台的配置中向“附加文件”属性添加下列 DLL：Mfc80u.dll、Atl80.dll 和 Msvcr80.dll．此示例假设采用发布配置。

向多平台 OnDraw() 方法添加代码

向 OnDraw() 方法添加代码

1. 在“解决方案资源管理器”中展开“源文件”节点。双击 HelloMFCView.cpp 以在编辑器中打开源文件。
2. 在 `OnDraw(CDC* pDC)` 方法中将 `OnDraw` 签名修改为取消注释的 `pDC`。结果行如下所示：

```
void CHelloMFCView::OnDraw(CDC* pDC)
```

3. 在 `OnDraw` 方法中的 `//TODO` 注释后插入以下代码：

```
// 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. 在"生成"菜单上单击"重新生成解决方案"。

选择目标设备

为了确保在部署解决方案时系统提示您选择设备，请完成以下过程。

在部署时提示选择设备

1. 在"工具"菜单上，单击"选项"，单击"设备工具"，然后单击"常规"。

   如果"设备工具"不可见，请选择"选项"对话框底部的"显示所有设置"。

2. 选择"部署设备项目前显示设备选项"复选框。

部署多平台 MFC 解决方案

部署解决方案

1. 在 Visual Studio 工具栏上的"目标设备"下拉列表中选择目标，例如"Pocket PC 2003 SE 仿真程序"或"Pocket PC 2003 设备"。

2. 在"生成"菜单上单击"部署"。

有关为此演练生成的代码的更多信息，请参见代码解释：Hello World：智能设备的多平台 MFC 应用程序。

请参见

其他资源

智能设备演练

如何：创建多平台设备项目 (Visual C++)

代码解释：Hello World：智能设备的多平台 MFC 应用程序

无论是将 Visual Studio C++ 用于 Windows CE（移动）及其他流行的移动设备，还是使用嵌入式 Visual C++ 开发设备应用程序，您将发现针对智能设备的 C++ 多平台 MFC 应用程序向导可简化生成项目文件所需的大多数常见任务。某些文件包括资源文件及多平台项目配置文件。该向导还引入了 jumpstart 代码，使您能够专注于开发核心业务应用程序功能。

此演练帮助您熟悉由针对智能设备的多平台 MFC 应用程序向导自动生成的代码，这样您便能根据需要对应用程序进行轻松扩展和修改。

有关更多信息，请参见演练：为智能设备创建多平台 MFC 应用程序、设备的 MFC 参考和 MFC Reference。

智能设备 C++ 的多平台 MFC 应用程序代码清单

向导生成的代码包括以下部分：

- **include** 部分。

```
#include "stdafx.h"
```

- stdafx.h。有关更多信息，请参见预编译头文件。

```
#include "MFCHello1.h"
#include "MFCHello1Doc.h"
#include "MFCHello1View.h"
```

- // 调试配置。

```
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

IMPLEMENT_DYNCREATE(CMFCHello1View, CView)
```

- MFC 消息映射和更具体的信息 BEGIN_MESSAGE_MAP。
- IMPLEMENT_DYNCREATE，请参见 IMPLEMENT_DYNCREATE。

```
BEGIN_MESSAGE_MAP(CMFCHello1View, CView)
END_MESSAGE_MAP()
```

- 构造代码。有关更多信息，请参见构造函数设计。有关异常处理，请参见 Handling Exceptions（处理异常）。

```
// 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);
}
```

- 请记住，使用 CView::OnDraw 方法时，在屏幕上绘图（屏幕绘制）。使用该方法，您可以控制图形设备接口 (GDI)，即通过作为参数提供的代码 `CDC* pDC`。切勿忘记取消对该代码的注释。例如，您可以在应用程序中为各项内容使用从文本到绘制动画游戏中的图形的全部 GDI 功能。在此 GDI 示例中，CDC::DrawTextCDC::DrawText 方法随后用于在由 CWnd::GetClientRect 定义的矩形内绘制 Hello World 文本。
- BEGIN_MESSAGE_MAP.
- CDC 类.
- 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 及其他相关方法。有关更多信息，请参见 CDC Class。

```
        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
```

由向导创建的自述文件中的可用信息

该应用程序向导已为您创建此 MFCHello1 应用程序。此应用程序不仅演示了使用 Microsoft 基础类的基本知识，而且还可作为编写应用程序的起点。

此处列出了生成的文件，以及每个文件中发现的内容摘要。这些文件共同构成 MFC 应用程序开发的 jumpstart 点。

一个示例使用了名称 HelloMFC。您可能想要使用自己的项目名称。

HelloMFC.vcproj

使用应用程序向导生成的 VC++ 项目的主项目文件。它包含生成文件的 Visual C++ 版本信息，以及有关用应用程序向导选择的平台、配置和项目功能的信息。

HelloMFC.h

应用程序的主头文件。它包括其他特定于项目的头文件，并声明 CMFCHello1App 应用程序类。

HelloMFC.cpp

应用程序的主源文件，该文件包含应用程序类 CMFCHello1App 的类定义。

HelloMFCppc.rc

当编译用于 *Pocket PC platform* 或支持同一用户界面模型的平台时，该项目使用的项目的所有 Microsoft Windows 资源的主资源文件列表。它包括存储在 RES 子目录中的图标、位图和光标。此文件可以直接在 Microsoft Visual C++ 中编辑。项目资源位于 2052。如果保留 .rc 文件，数据节中的定义将保留为它们所定义为的数值的十六进制版本，而不是定义的友好名称。

res\HelloMFCppc.rc2

此文件包含不是由 Microsoft Visual C++ 编辑的资源。您应将所有不可由此资源编辑器编辑的资源放置在此文件中。

HelloMFCsp.rc

当编译用于 *Smartphone platform* 或支持同一用户界面模型的平台时，该项目使用的项目的所有 Microsoft Windows 资源的主资源文件列表。它包括存储在 RES 子目录中的图标、位图和光标。此文件可以直接在 Microsoft Visual C++ 中编辑。项目资源位于 2052。如果保留 .rc 文件，数据节中的定义将保留为它们所定义为的数值的十六进制版本，而不是定义的友好名称。

res\HelloMFCsp.rc2

此文件包含不是由 Microsoft Visual C++ 编辑的资源。您应将所有不可由此资源编辑器编辑的资源放置在此文件中。

res\HelloMFC.ico

一个图标文件，用作应用程序的图标。此图标包含在主资源文件中。

MainFrm.h、MainFrm.cpp

包含框架类 CMainFrame 的文件，这些类从 CFrameWnd 派生，并控制所有的 SDI 框架功能。

应用程序向导还创建一个 MFC 文档类型和一个 MFC 视图：

- **HelloMFCDoc.h、HelloMFCDoc.cpp**

  包含 HelloMFCDoc 类的文件。编辑这些文件，以添加特殊的文档数据，并实现文件保存和加载（使用 CMFCHello1Doc::Serialize）。

- **HelloMFCView.h、HelloMFCView.cpp**

  包含 HelloMFCView 类的文件。HelloMFCView 对象用以查看 HelloMFCDoc 对象。

StdAfx.h、StdAfx.cpp

这两个文件用于生成名为 HelloMFC.pch 的预编译头 (PCH) 文件和名为 StdAfx.obj 的预编译类型文件。

Resourceppc.h 和 Resourcesp.h

标准头文件，用于定义新的资源 ID。Microsoft Visual C++ 读取并更新此文件。

该应用程序向导使用 `TODO:` 指示可以添加到或自定义的源代码部分。

如果应用程序在共享 DLL 中使用 MFC，且应用程序使用的语言与操作系统的当前语言不同，则可能需要将相应本地化资源的 MFC80XXX.DLL 复制到应用程序的目录中。在此名称中，"XXX"代表语言缩写。例如，MFC80DEU.DLL 包含已翻译为德语的资源。否则，应用程序的某些 UI 元素将保留为操作系统的语言。

请参见

其他资源

示例和演练(智能设备项目)

演练：为智能设备创建 MFC 多平台 ActiveX 控件

可以使用 Visual C++ 编写面向多种设备的 MFC ActiveX 控件代码。本演练阐释如何生成用于多种设备的 C++ 多平台 MFC ActiveX 控件。

创建 MFC ActiveX 多平台控件项目

本演练由三项主要任务组成：

- 创建多平台智能设备 MFC ActiveX 控件项目。
- 为 MFC ActiveX 控件的 `OnDraw()` 方法添加代码。
- 部署多平台 MFC ActiveX 控件解决方案以用于测试。

有关更多信息，请参见 MFC 智能设备 ActiveX 控件向导。

注意
显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

本演练使用 Visual C++ 开发设置编写。

创建多平台智能设备 MFC ActiveX 控件项目

1. 在“文件”菜单上指向“新建”，单击“项目”，在“项目类型”窗格中展开“Visual C++”节点，然后单击“智能设备”。
2. 在“模板”窗格中单击“MFC 智能设备 ActiveX 控件”。
3. 在“名称”框中键入**“MFCAX”**。
4. 在“解决方案”框中，接受默认选项“创建解决方案的目录”。
5. 单击“确定”启动“MFC 智能设备 ActiveX 控件向导”。
6. 在“MFC 智能设备应用程序向导欢迎”页上单击“下一步”。出现“MFC 智能设备 ActiveX 控件向导”的“平台”，此时您可以选择要添加到当前项目的平台。

   从“已安装的 SDK”窗格中选择要用于和添加到当前项目的平台，例如“Smartphone 2003”和“Pocket PC 2003”。若要添加平台，请在左窗格中选择该平台（如“Smartphone 2003”），然后单击右箭头（“>”）按钮。若要移除平台，请在右窗格中选择该平台（如“Pocket PC 2003”），然后单击左箭头（“<”）按钮。
7. 单击“完成”以完成并关闭向导，或者单击“下一步”接受向导中的所有剩余选项的默认值。

注意
创建设备项目后，您可以在第一次创建之后不断添加更多平台。但是，将新的平台添加到现有的项目并不会将其他依赖性运行时 DLL 添加到“附加文件”配置属性中。例如，如果您的应用程序动态链接到 MFC，则需要在新添加的平台的“附加文件”属性中包含下列 DLL：Mfc80u.dll、Atl80.dll 和 Msvcr80.dll。此示例假设为发布配置。
8.

为多平台 MFC 控件的 OnDraw() 方法添加代码

为 MFC ActiveX 控件的 OnDraw 方法添加代码

1. 在“解决方案资源管理器”中展开“源文件”节点，然后选择 MFCAXCtrl.cpp 源文件并将其在编辑器打开。
2. 将 `OnDraw` 方法的代码替换为下面的代码，尤其是最后三行：

```
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. 在“生成”菜单上单击“重新生成解决方案”。

部署多平台解决方案

部署解决方案

1. 要运行部署的解决方案，首先需要在目标设备上部署和注册 ActiveX 控件项目。
2. 在 Visual Studio 工具栏上的“目标设备”下拉列表中选择目标，例如“Pocket PC 2003 SE 仿真程序”或“Pocket PC 2003 设备”。
3. 在“生成”菜单上单击“部署”。

选择目标设备

为了确保在部署解决方案时系统提示您选择设备，请完成以下过程。

在部署时提示选择设备

1. 在“工具”菜单上，单击“选项”，单击“设备工具”，然后单击“常规”。如果“设备工具”不可见，请选择“选项”对话框底部的“显示所有设置”。
2. 选择“部署设备项目前显示设备选项”复选框。

有关更多信息，请参见 Mobile Developer Center（移动开发人员中心）。

请参见

其他资源

智能设备演练

如何：创建多平台设备项目 (Visual C++)

代码解释：Hello World：智能设备的多平台 ActiveX 控件

智能设备 C++ ActiveX MFC 向导使您从生成针对智能设备项目的多平台 MFC ActiveX 控件的大多数常见任务（如生成资源文件和多平台项目配置文件）中解脱出来。向导会生成这些基本文件。

此演练帮助您熟悉在使用针对智能设备的多平台 MFC ActiveX 控件向导时自动生成的代码，这样您便能根据需要对应用程序进行轻松扩展和修改。

代码演练

通常，熟悉 MFC ActiveX 控件编程或 ActiveX 编程的人员都可以轻松地发现由该向导生成的主要功能。该向导代码包含以下部分：

- **include** 部分：

```
#include "stdafx.h"
```

- stdafx.h。有关更多信息，请参见 预编译头文件。

```
#include "stdafx.h"
#include "<PROJECTNAME>.h"
#include "MFCAXCtrl.h"
#include "MFCAX1PropPage.h"
```

- 调试配置：

```
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

- IMPLEMENT_DYNCREATE。有关更多信息，请参见 IMPLEMENT_DYNCREATE。
- MFC 消息映射。有关更多信息，请参见 BEGIN_MESSAGE_MAP。

```
// Message map

BEGIN_MESSAGE_MAP(CMFCAXCtrl, COleControl)
    ON_OLEVERB(AFX_IDS_VERB_PROPERTIES, OnProperties)
END_MESSAGE_MAP()
```

- MFC 调度映射。有关更多信息，请参见 BEGIN_DISPATCH_MAP。

```
// 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 事件映射。有关更多信息，请参见 BEGIN_EVENT_MAP。

```
    // 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)
```

由向导创建的自述文件中的可用信息

该应用程序向导已为您创建此 jumpstart 应用程序。此应用程序不仅演示了使用 Microsoft 基础类的基本知识，而且还可作为编写 ActiveX 控件应用程序的起点。

此处列出了生成的文件，以及每个文件中发现的内容摘要。这些文件共同构成 MFC ActiveX 开发的 jumpstart 点。

一个示例使用了名称 MFCAX1。您可能想要使用自己的项目名称。

MFCAXDLL.vcproj

该向导为 MFCAX1 ActiveX 控件 DLL 创建了此项目，其中包含 1 个控件。

MFCAX1.vcproj

使用应用程序向导生成的 VC++ 项目的主项目文件。其中包含生成该文件的 Visual C++ 版本信息，以及有关在应用程序向导中选择的平台、配置和项目功能的信息。

MFCAX1.h

ActiveX 控件 DLL 的主包含文件。它包括其他特定于项目的包含文件。

MFCAX1.cpp

包含 DLL 初始化、终止和其他薄记代码的主源文件。

MFCAX1ppc.rc

当编译用于 Pocket PC 平台或支持同一用户界面模型的平台时，该项目使用的所有 Microsoft Windows 资源的列表。如果保留 .rc 文件，数据节中的 **#defines** 实例将保留为它们所定义为的数值的十六进制版本，而不是定义的友好名称。

MFCAX1sp.rc

当编译用于 Smartphone 平台或支持同一用户界面模型的平台时，该项目使用的所有 Microsoft Windows 资源的列表。如果保留 .rc 文件，数据节中的 **#defines** 实例将保留为它们所定义为的数值的十六进制版本，而不是定义的友好名称。

MFCAX1.rc2

此文件包含不是由 Microsoft Visual C++ 编辑的资源。您应将所有不可由此资源编辑器编辑的资源放置在此文件中。

MFCAX1.def

此文件包含有关 ActiveX 控件 DLL 的信息，必须提供此控件才可以在 Microsoft Windows 上运行。

MFCAX1.idl

此文件包含控件类型库的对象描述语言源代码。

C MFCAX1Ctrl 控件文件：

- **MFCAX1Ctrl.h**：此文件包含 C<PROJECTNAME>Ctrl C++ 类的声明。
- **MFCAX1Ctrl.cpp**：此文件包含 C **MFCAX1**Ctrl C++ 类的实现。
- **MFCAX1PropPage.h**：此文件包含 C **MFCAX1**PropPage C++ 类的声明。
- **MFCAX1PropPage.cpp**：此文件包含 **C MFCAX1PropPage** C++ 类的实现。
- **CMFCAX1Ctrl.bmp**：此文件包含位图，容器将在 **CMFCAX1Ctrl** 控件出现在工具面板上时使用该位图来表示此控件。该位图包含在主资源文件 (.rc) 中。

其他标准文件：

- **stdafx.h、stdafx.cpp**：这些文件用于生成名为 **MFCAX1.pch** 的预编译头 (PCH) 文件，以及名为 **stdafx.obj** 的预编译类型 (PCT) 文件。
- **resourceppc.h**：标准头文件，用于定义新资源 ID。Visual C++ 资源编辑器将读取和更新此文件。
- **resourcesp.h**：标准头文件，用于定义新资源 ID。Visual C++ 资源编辑器将读取和更新此文件。控件向导使用 **TODO** 指示应添加到或自定义的源代码部分。

请参见

其他资源

示例和演练（智能设备项目）

演练：为智能设备创建 ATL 多平台 ActiveX 控件

可以使用用于设备的 Visual C++ 编写面向多种设备的 ActiveX 控件。下面的演练阐释如何生成多平台 ATL ActiveX 控件。

在本演练中，执行以下主要任务：

- 创建多平台智能设备 ATL 项目。
- 使用向导将 ActiveX 控件添加至项目。注意，大部分基本结构和代码都是由向导生成的。
- 修改 **stdafx.h** 和 **samplecontrol.h** 文件中的代码，以定义线程模型和避免编译器警告。
- 部署多平台解决方案。注意，还生成了一个 Internet Explorer 文件，以便测试和运行该控件。

本演练使用 Visual C++ 开发设置编写。

注意
显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

创建多平台 ATL ActiveX 控件

创建多平台 ATL ActiveX 控件

1. 在“文件”菜单上指向“新建”，单击“项目”，在“项目类型”窗格中展开“Visual C++”节点，然后单击“智能设备”。
2. 在“模板”窗格中单击“ATL 智能设备项目”。
3. 在“名称”框中键入**“ATLAXControl”**，再单击“确定”。

   此时，将启动“ATL 智能设备项目向导”。
4. 在“ATL 智能设备项目向导”的“欢迎”页上单击“下一步”。

   显示“ATL 智能设备项目向导”的“平台”，此时您可以选择要添加到当前项目的 Platform SDK。
5. 从“已安装的 SDK”列表中选择要添加到当前项目中的平台，例如“Smartphone 2003”和“Pocket PC 2003”。若要添加平台，请在左窗格中选择该平台（如“Smartphone 2003”），然后单击右箭头（“>”）按钮。若要移除平台，请在右边窗格中选择该平台（如“Pocket PC 2003”），然后单击左箭头（“<”）按钮。
6. 单击“完成”以完成并关闭向导。

向项目添加 ActiveX 控件

向项目添加 ActiveX 控件

1. 在“解决方案资源管理器”中右击“ATLAXControl”，指向“添加”，再单击“类”。
2. 在“类别”窗格中单击“智能设备”。
3. 在“模板”窗格中，单击“ATL 控件”，再单击“添加”。

   将出现“ATL 控件向导”对话框。
4. 在“简称”文本框中，键入 **samplecontrol**。
5. 单击“完成”以完成并关闭向导。

修改头文件中的代码

修改 stdafx.h 中的代码

1. 在“解决方案资源管理器”中，双击 stdafx.h 以在编辑器中打开该文件。
2. 在 `#pragma once` 后面添加以下定义 `#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA`，如下所示：

```
    // Add this define after
    #pragma once
#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA
```

3. 按下面的步骤演示，将 ActiveX 控件添加至项目。

向项目添加 ActiveX 控件

1. 在“解决方案资源管理器”中，双击 samplecontrol.h 以在编辑器中打开该文件。
2. 在定义 `Isamplecontrol` 的代码中，将字符串 `ATL 8.0 : samplecontrol` 替换为 `Hello World ActiveX Control`。

注意

面向 DCOM 平台的 ActiveX 控件必须在生成时标记为单元模型线程。这是 ATL 控件向导的默认设置。可以放心忽略编译过程中生成的警告。同样，对于 ATL、GUI 和 EXE 项目（例如已向 ATL EXE 项目中添加了 atlwin.h、atlctl.h 或 atlhost.h 的项目），在包含 ATL 头文件之前，应在 stdafx.h 中定义 `_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA`。这种做法与开发桌面控件时相同。有关更多信息，请参见生成并调试 Visual C++ 设备项目。

部署多平台 ATL 解决方案

部署解决方案

1. 在“生成”菜单上单击“重新生成解决方案”以生成控件。
2. 在“生成”菜单上单击“部署解决方案”。
3. 在 Visual Studio 工具栏上的“目标设备”下拉列表中选择目标，例如“Pocket PC 2003 SE 仿真程序”或“Pocket PC 2003 设备”。
4. 在“生成”菜单上单击“部署”。

选择目标设备

为了确保在部署解决方案时系统提示您选择设备，请完成以下过程。

在部署时提示选择设备

1. 在“工具”菜单上单击“选项”，展开“设备工具”节点，再单击“常规”。
2. 如果“设备工具”不可见，请选择“选项”对话框底部的“显示所有设置”。
3. 选择“部署设备项目前显示设备选项”复选框，再单击“确定”。

若要运行控件，请在设备上使用文件资源管理器定位至“Program Files\ATLAXControl”，再双击 Internet Explorer 文件“ATLAXControl”。此时，将显示一个或多个安全消息。单击“是”以运行该页。

请参见

其他资源

智能设备演练

如何：创建多平台设备项目 (Visual C++)

如何：在对话框资源中承载 ActiveX 控件

当使用 Visual Studio 2005 设计设备的 ActiveX 控件时，需要添加几个额外步骤。由于“资源编辑器”依赖于在桌面计算机上注册的控件在设计时对其进行操作，以及您无法在桌面计算机上注册设备控件的原因，以下步骤提供了另一种设计时体验。以下过程假定您已拥有 ActiveX 控件项目和宿主项目，并且要在对话框中承载 ActiveX 控件。

注意

显示的对话框和菜单命令可能会与“帮助”中描述的不同，具体取决于当前的设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

使用对话框编辑器添加 ActiveX 控件

1. 在对话框编辑器中，打开宿主项目的对话框。
2. 将一个自定义控件从“工具箱”中拖动到该对话框。
3. 调整自定义控件在对话框上的位置和大小，以反映您希望 ActiveX 控件显示的样式。
4. 右击该自定义控件，然后单击“属性”。
5. 在“类”属性中，粘贴 ActiveX 控件的 GUID。请记住包括大括号“{...}”。
6. 在“解决方案资源管理器”中，右击项目 Project Name.RC2 文件，然后单击“查看代码”。
7. 在“在此处添加手动编辑的资源”部分，添加以下代码。自定义控件要求正确显示对话框 Init 部分。不使用实际对话框 Init 部分的内容。请记住用项目的名称替换 <项目名称>。

```
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. 生成和运行宿主项目。请记住在目标设备上部署和注册 ActiveX 控件。

使用承载 ActiveX 控件的替代方法

1. 通过在应用程序中的某个位置调用 **AtlAxWinInit** 来注册 AtlAxWin80 窗口类。ATL 应用程序在模块初始化代码中执行此操作。Win32 应用程序应在 **WinMain** 函数中调用该函数。对于 MFC 应用程序，请按照下列步骤操作：
   a. 右击“解决方案资源管理器”中的项目节点，单击“添加”，然后单击“类”。
   b. 单击“向 MFC 添加 ATL 支持”（在“智能设备”标题下）。
   c. 将 **AtlAxWinInit** 调用添加到宿主应用程序类的 **InitInstance** 方法的顶部。
2. 在对话框资源（如 ATL 对话框或复合控件，或 MFC 对话框）中：
   a. 从“工具箱”拖动一个自定义控件。
   b. 将窗口类属性设置为 AtlAxWin80。
   c. 将标题设置为用大括号括起的 GUID，或设置为 ProgID。
3. 对于 MFC，请将 atl.lib 作为附加链接输入添加。
4. 对于 MFC，请将这些行添加到“部署”|“附加文件”选项。对于动态链接库，这些行已添加；对于 MFC 静态链接库，则需要添加这些行。

```
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
```

请参见

其他资源

使用 Visual C++ 进行设备编程
智能设备演练

智能设备示例

您可以访问 Mobile Developer Center（移动开发人员中心）在线查找多个设备示例。

下面的 Visual C# 示例将公开 .NET Compact Framework 的功能。这些本机示例使用 Visual C++ 演示常见的设备应用程序方案。

本节内容

创建一个自定义控件 (Visual C#)

提供一个双行 ListView 自定义控件，包括 WYSIWYG 设计时。

基于 .NET Compact Framework 2.0 版。

CompactNav (Visual C#)

提供一个 Smartphone 文件系统浏览器，它显示诸如调用非托管 API 等技术。

基于 .NET Compact Framework 1.0 版。

本机示例（设备）

提供 Mobile Developer Center 站点上三个示例本机应用程序的链接。

请参见

其他资源

使用 .NET Compact Framework 进行设备编程

示例和演练（智能设备项目）

创建一个自定义控件 (Visual C#)

Download sample

此 Visual C# 示例演示一个包含两行的 ListView 自定义控件，提供所见即所得的设计时体验。

可以使用 .NET Compact Framework 的版本 1.0 或版本 2.0 为 Pocket PC 或 Smartphone 生成该示例。

安全注意
提供该代码示例是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该代码示例。对于因将该代码示例用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

运行此示例

1. 单击“下载示例”。

   出现“文件下载”消息框。

2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

   “提取向导”打开。

3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

   请确保选中了“显示提取的文件”复选框，并单击“完成”。

4. 双击该示例的 .sln 文件。

   示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

此示例演示的内容

此示例演示如何以完全的设计时支持，使用自定义绘制实现“TwoLineListBox”自定义控件，包括下面的技术：

- 使用 System.Drawing 和 System.Drawing.Imaging 命名空间中的类，自定义绘制控件的外观。
- 处理事件以重新绘制自定义控件。
- 使用数据绑定从数据源中加载数据，以填充控件。
- 在自定义控件中显示图像。
- 通过使用类关系图和 DesignTimeAttributes，添加对说明、类别以及其他属性和事件的设计时支持。
- 添加将控件绑定到现有数据源的设计时支持。
- 通过 DesignTimeAttributes 将 DataSourceListEditor 和 DataMemberFieldEditor 应用于属性（DataSource、Data）。
- 通过 DesignTimeAttributes 将自定义编辑器（联系人编辑器）应用于属性 (Context)。
- 设置一些属性（如 Line1DisplayMember 和 Line2DisplayMember）的默认值。
- 将集合属性从自定义编辑器（联系人编辑器）中序列化至代码。
- 将集合属性从代码中反序列化至自定义编辑器（联系人编辑器）。
- 将图像从自定义编辑器（联系人编辑器）中序列化至 .resx 文件。
- 将图像从 .resx 文件中反序列化至自定义编辑器（联系人编辑器）。

CompactNav (Visual C#)

Download sample

CompactNav 示例是一个 Smartphone 文件/系统浏览器，该浏览器是使用 .NET Compact Framework 版本 1.0 在托管代码中编写的。

安全注意
提供该代码示例是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该代码示例。对于因将该代码示例用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

运行此示例

1. 单击“下载示例”。

   出现“文件下载”消息框。

2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

   “提取向导”打开。

3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

   请确保选中了“显示提取的文件”复选框，并单击“完成”。

4. 双击该示例的 .sln 文件。

   示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

此示例演示的内容

该示例阐释了如何实现基本的文件/系统导航器，其中包括下面的技术：

- 使用平台调用 (pInvoke) 调用非托管 API，如 **CreateProcess**。
- 异常处理。
- 如何从文件系统加载映像。
- 选择并执行可执行文件（使用 **CreateProcess**）。
- 通过选择目录并按操作键进入目录。
- 通过使用菜单项 (UpDir) 或选择常规“..”目录访问父目录。
- 实现退出选项（仅为了测试方便，一般不用于或不建议用于实际的 Smartphone 应用程序）。

本机示例(设备)

有关使用 Visual C++ 的设备应用程序的示例，请访问 Mobile Developer Center(移动开发人员中心)。这些示例包括：

示例	说明
PhoneMenu	演示 Smartphone 应用程序的菜单。
TextEditor	演示 Microsoft 基础类的基本使用方法，并提供编写自己的 MFC 应用程序的起点。
PoomViewer	在查询并显示约会、联系人和任务数据库中所有字段的程序中演示一个独立于设备的用户界面。

有关这些示例和它们所展示的功能的更多信息，请参阅每个示例附带的 Readme.txt 文件。

请参见

其他资源

智能设备示例

参考(设备)

本节提供智能设备项目的参考信息。相关章节提供链接，这些链接指向设备项目的针对语言的参考主题。

本节内容

设备项目的 .NET Compact Framework 引用

提供有关 .NET Compact Framework 的信息。

设备的 ATL 引用

提供如何显示在设备项目中受支持的 ATL 参考主题的说明。

设备的 MFC 参考

提供设备的 MFC 的参考主题。

针对设备的 C 运行时库参考

提供设备的 C 运行时库的参考主题。

设备的标准 C++ 库参考

提供可用于设备应用程序开发的标准 C++ 库子集的参考主题。

用于智能设备的编译器

描述 ARM、Renesas 和 MIPS 系列处理器，并列出桌面编译器与设备编译器之间的区别。

设备的用户界面参考

列出仅适用于智能设备应用程序开发的界面元素。

用于设备应用程序开发的 Visual Basic 语言参考

总结了智能设备项目中可用的 Visual Basic 编程元素之间的区别。

错误信息(设备)

提供有关解决特定错误的信息。

请参见

其他资源

智能设备开发

设备项目的 .NET Compact Framework 引用

.NET Compact Framework 是 .NET Framework 类库的子集，还包含专门为它设计的类。它继承了公共语言运行库和托管代码执行的 .NET Framework 全功能版的体系结构。

有关更多信息，请参见 .NET Compact Framework。

请参见

其他资源

.NET Framework

设备的 ATL 引用

由于设备限制，设备的 ATL 不支持标准 ATL 所支持的所有内容。此节包含对区别进行解释的主题。

本节内容

如何：查找有关设备支持的 ATL 类和方法的帮助

描述如何使用“帮助”的筛选机制来显示有关设备的 ATL 引用、ATL 类以及在设备上受支持的方法的最新信息。

设备的 ATL 和标准 ATL 之间的差异

描述标准 ATL 和设备的 ATL 之间的区别。

相关章节

ATL Reference

如果您从未使用过 ATL，本主题将您带入桌面计算机的活动模板库。

请参见

其他资源

参考（设备）

如何：查找有关设备支持的 ATL 类和方法的帮助

Visual Studio 帮助提供目录和索引的“智能设备开发”筛选器。应用此筛选器时，可见主题会减少为仅包括对于智能设备开发人员而言必不可少的文档。如果要仅显示参考主题中与智能设备有关的部分（包括 ATL 库），则应用此筛选器是一种绝佳的方法。

要查看 C++ 主题和参考文档，应使用“Visual C++ 设置”。

如果应用语言筛选器，如 Visual C#，则可查看的主题中不包括智能设备开发主题。为此，最好使用“智能设备开发”筛选器，或者根本不使用筛选器。使用语言开发设置（如“Visual C# 开发设置”）启动 Visual Studio。可以将此设置更改为“Visual C++ 设置”。

如何设置智能设备开发筛选器

设置智能设备开发筛选器

1. 在 Visual Studio 的“帮助”菜单上，单击“目录”或“索引”。
2. 在“筛选依据”框中，选择“智能设备开发”。

更改开发设置

1. 在 Visual Studio 的“工具”菜单上单击“导入和导出设置”。
2. 在向导的“欢迎使用”页上，单击“重置所有设置”，然后单击“下一步”。

   如果不确定应选择哪些选项，请按 F1 打开向导的帮助主题。
3. 在“保存当前设置”页上，选择是否保存当前设置，然后单击“下一步”。
4. 在“选择一个默认设置集合”页上，选择“Visual C++ 设置”，然后单击“完成”。

请参见

任务

如何：查找有关设备支持的 MFC 类和方法的帮助

概念

Visual Studio 的帮助筛选器

其他资源

智能设备项目入门

设备的 ATL 和标准 ATL 之间的差异

ATL 传统上用于开发基于 COM 的应用程序。ATL 8.0 的功能可以使 COM 编程更简单，如字符串处理和转换类、管理数组类，以及列表和目录树的类。ATL 设备开发人员将在 ATL 8.0 中发现一些与 eMbedded Visual C++ 的区别，包括 Web 服务客户支持、扩展的套接字支持 (IPv6) 及改进的安全性。但是，设备的 ATL 8.0 将不具有所有桌面 ATL 功能。尽管 ATL 设备版本中包括 Web 服务使用，但在设备版本中不包括安全、服务、ATL 数据和 ATL Server。

此节包含的主题是解释设备的 ATL 和标准 ATL 之间的差异和相似性。有关筛选设备的 ATL 参考的帮助主题的更多信息，请参见如何：查找有关设备支持的 ATL 类和方法的帮助。

本节内容

受支持的 ATL 类的列表

提供对于设备项目完全支持的所有 ATL 类的列表

唯一的设备 ATL 类

提供仅不支持用于设备的 ATL 类的列表。

请参见

参考

受支持的 ATL 类的列表

其他资源

设备的 ATL 引用

唯一的设备 ATL 类

受支持的 ATL 类的列表

以下是用于设备的受支持 ATL 类。由于设备的限制，桌面上可用的类功能在设备上可能不可用。若要查看设备上受支持的类和方法的详细信息，可以使用开发环境"帮助"的筛选机制。

有关筛选开发环境的帮助以查看 ATL 主题和参考（包括用于设备的 ATL 方法）的信息，请参见如何：查找有关设备支持的 ATL 类和方法的帮助。

用于设备受支持的 ATL 类包括：

- allocator Class
- AtlHtmlAttrs Class
- CAdapt Class
- CAtlBaseAuthObject Class
- CAtlBaseModule Class
- CAtlComModule Class
- CAtlException Class
- CAtlFile Class
- CAtlFileMappingBase Class
- CAtlHttpClientT Class
- CAtlNavigateData Class
- CAtlTemporaryFile Class
- CAtlWinModule Class
- CAutoPtr Class
- CAutoPtrArray Class
- CAutoPtrElementTraits Class
- CAutoPtrList Class
- CAutoVectorPtr Class
- CAutoVectorPtrElementTraits Class
- CBasicAuthObject Class
- CCacheDataBase Class
- CComAggObject Class
- CComAllocator Class
- CComAutoCriticalSection Class
- CComBSTR Class
- CComCachedTearOffObject Class
- CComClassFactory Class
- CComClassFactory2 Class
- CComClassFactorySingleton Class
- CComContainedObject Class
- CComCriticalSection Class
- CComCritSecLock Class

- CComDynamicUnkArray Class
- CComFakeCriticalSection Class
- CComHeap Class
- CComHeapPtr Class
- CComModule Class
- CComMultiThreadModel Class
- CComMultiThreadModelNoCS Class
- CComObject Class
- CComObjectGlobal Class
- CComObjectNoLock Class
- CComObjectRootEx Class
- CComObjectStack Class
- CComPolyObject Class
- CComPtr Class
- CComPtrBase Class
- CComQIPtr Class
- CComSafeArrayBound Class
- CComTearOffObject Class
- CComUnkArray Class
- CComVariant Class
- CContainedWindowT Class
- CCriticalSection Class
- CCRTAllocator Class
- CCRTHeap Class
- CCryptDerivedKey Class
- CCryptHash Class
- CCryptHMACHash Class
- CCryptImportKey Class
- CCryptKey Class
- CCryptKeyedHash Class
- CCryptMACHash Class
- CCryptMD2Hash Class
- CCryptMD4Hash Class
- CCryptMD5Hash Class
- CCryptProv Class
- CCryptRandomKey Class
- CCryptSHAHash Class
- CCryptSSL3SHAMD5Hash Class

- CCryptUserExKey Class
- CCryptUserSigKey Class
- CDefaultCharTraits Class
- CDefaultCompareTraits Class
- CDefaultElementTraits Class
- CDefaultHashTraits Class
- CDynamicChain Class
- CElementTraits Class
- CElementTraitsBase Class
- CEvent Class
- CExpireCuller Class
- CFileTime Class
- CFileTimeSpan Class
- CFirePropNotifyEvent Class
- CFixedLifetimeCuller Class
- CGlobalHeap Class
- CHandle Class
- CHtmlGen Class
- CHtmlGenBase Class
- CHttpResponse Class
- CImage Class
- CLifetimeCuller Class
- CLocalHeap Class
- CLOUFlusher Class
- CLRUFlusher Class
- CMutex Class
- CNoDllCachePeer Class
- CNoExpireCuller Class
- CNoFileCachePeer Class
- CNoFlusher Class
- CNonStatelessWorker Class
- CNoStatClass Class
- CNoWorkerThread Class
- CNTLMAuthObject Class
- COldFlusher Class
- COleDateTime Class
- COleDateTimeSpan Class
- COrCullers Class

- COrFlushers Class
- CPoint Class
- CPrimitiveElementTraits Class
- CRect Class
- CRegKey Class
- CSemaphore Class
- CSimpleArrayEqualHelper Class
- CSimpleArrayEqualHelperFalse Class
- CSimpleDialog Class
- CSimpleMap Class
- CSimpleMapEqualHelper Class
- CSimpleMapEqualHelperFalse Class
- CSocketAddr Class
- CStrBufT Class
- CStreamFormatter Class
- CStreamOnWriteStream Class
- CStringRefElementTraits Class
- CTime Class
- CTimeSpan Class
- CUrl Class
- CWin32Heap Class
- CWindow Class
- CWriteStreamHelper Class
- Win32ThreadTraits Class

请参见

其他资源

设备的 ATL 和标准 ATL 之间的差异

设备的 ATL 引用

唯一的设备 ATL 类

设备的 ATL 有一个独有的类。本节描述此类。

本节内容

CAtlCEValidateThreadIDDefault 方法

提供与线程相关的宏和方法的类，这些宏和方法是设备项目独有的。

请参见

任务

如何：查找有关设备支持的 ATL 类和方法的帮助

其他资源

设备的 ATL 引用

CAtlCEValidateThreadIDDefault 类

介绍 **CAtlCEValidateThreadIDDefault** 类，此类提供与线程相关的宏和方法，这些宏和方法是设备项目独有的。

要求

Windows CE 5.0 版及更高版本。

头文件：在 atlcore.h 中声明。

请参见

概念

CAtlCEValidateThreadIDDefault 方法

其他资源

唯一的设备 ATL 类

CAtlCEValidateThreadIDDefault 方法

下表列出了 **CAtlCEValidateThreadIDDefault** 方法：

CAtlCEValidateThreadIDDefault::CAtlCEValidateThreadIDDefault	创建 **CAtlCEValidateThreadIDDefault** 对象的实例，并设置 **m_ThreadId** 私有数据成员。
CAtlCEValidateThreadIDDefault::Validate	指示线程标识符（用作调用线程的句柄）与原始调用线程相同，还是当前线程标识符有所不同。

下表列出了 **CAtlCEValidateThreadIDDefault** 宏：

CE_VALIDATE_THREADID_ASSERT 宏	在调试版本中，当 CAtlCEValidateThreadIDDefault::Validate 方法返回 false 时，CE_VALIDATE_THREADID_ASSERT 生成一个调试报告。
CE_VALIDATE_THREADID_RETURN 宏	如果线程标识符已验证，则返回宏的参数。
CE_VALIDATE_THREADID_THROW 宏	如果线程标识符已验证，则引发异常。

请参见

概念

CAtlCEValidateThreadIDDefault 类

其他资源

唯一的设备 ATL 类

CAtlCEValidateThreadIDDefault::CAtlCEValidateThreadIDDefault

创建 **CAtlCEValidateThreadIDDefault** 对象的实例并将 **m_ThreadId** 私有数据成员设置为当前线程标识符，该线程标识符用作调用线程的句柄。

```
CAtlCEValidateThreadIDDefault();
```

要求

Windows CE 5.0 版及更高版本。

头文件：在 atlcore.h 中声明。

请参见

概念

CAtlCEValidateThreadIDDefault 方法

其他资源

唯一的设备 ATL 类

CAtlCEValidateThreadIDDefault::Validate

指示线程标识符(用作调用线程的句柄)与原始调用线程相同，还是当前线程标识符有所不同。

```
bool Validate ();
```

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 atlcore.h 中声明。

请参见

概念

CAtlCEValidateThreadIDDefault 方法

其他资源

唯一的设备 ATL 类

CE_VALIDATE_THREADID_ASSERT 宏

在调试版本中，**CE_VALIDATE_THREADID_ASSERT** 在 CAtlCEValidateThreadIDDefault::Validate 方法返回 false 时生成一个调试报告。

```
CE_VALIDATE_THREADID_ASSERT();
```

要求

Windows CE 5.0 版及更高版本。

头文件：在 atlcore.h 中声明。

请参见

概念

CAtlCEValidateThreadIDDefault 方法

其他资源

唯一的设备 ATL 类

CE_VALIDATE_THREADID_RETURN 宏

如果线程标识符已验证，则返回宏的参数。

```
CE_VALIDATE_THREADID_RETURN(r);
```

参数

下表描述了 **CAtlCEValidateThreadIDDefault** 类的 **CE_VALIDATE_THREADID_RETURN** 宏的参数。

参数	说明
r	在线程已验证的情况下返回的参数。

要求

Windows CE 5.0 版及更高版本。

头文件：在 atlcore.h 中声明。

请参见

参考

CAtlCEValidateThreadIDDefault::Validate

概念

CAtlCEValidateThreadIDDefault 方法

其他资源

唯一的设备 ATL 类

CE_VALIDATE_THREADID_THROW 宏

如果线程标识符已验证，则引发异常。

```
CE_VALIDATE_THREADID_THROW(e);
```

参数

下表描述了 **CAtlCEValidateThreadIDDefault** 类的 CE_VALIDATE_THREADID_THROW 宏的参数。

参数	说明
e	要引发的异常。

要求

Windows CE 5.0 版及更高版本。

头文件：在 atlcore.h 中声明。

请参见

参考

CAtlCEValidateThreadIDDefault::Validate

概念

CAtlCEValidateThreadIDDefault 方法

其他资源

唯一的设备 ATL 类

设备的 MFC 参考

在 Visual Studio 2005 中，智能设备项目使用设备的 MFC 8.0 版本。设备的 MFC 8.0 与标准桌面 MFC 和 eMbedded Visual C++ MFC 3.0 均不同。本节将介绍在何处获取帮助。

设备文件 MFC 位于文件夹 %Program Files%\Microsoft Visual Studio 8\VC\ce 下。

本节内容

MFC Classes

大多数设备文档 MFC 的位置。

请注意，设备 MFC 与标准桌面 MFC 共享文档。

“智能设备开发人员说明”部分的主题中指出了偏离标准桌面 MFC 的行为。

设备类的唯一 MFC

记下智能设备独有的 MFC 类的位置。只有几个独有的类。

如何：查找有关设备支持的 MFC 类和方法的帮助

描述如何使用筛选机制来显示有关设备的 MFC/ATL 引用的最新信息。

设备支持的桌面 MFC 类的列表

列出设备支持的标准桌面 MFC 类。

设备不支持的桌面 MFC 类的列表

列出设备不支持的标准桌面 MFC 类。

从 MFC 3.0 升级到 8.0 后不受支持的 eVC 类的列表

描述 eVC MFC 3.0 与设备的 Visual Studio 2005 MFC 8.0 之间的区别。

设备的 MFC C++ 和标准 MFC 之间的差异

用于说明在桌面和 MFC C++ 设备上都得到支持的类的图表。

请参见

概念

eMbedded Visual C++ 到 Visual Studio 2005 升级向导

设备的标准 C++ 库参考

其他资源

参考（设备）

MFC for Windows CE 向导 (C++)

使用 Visual C++ 向导，可以编写面向多个 Windows CE 设备的 MFC 应用程序。Visual C++ 向导包括 MFC 智能设备应用程序、MFC 智能设备 ActiveX 控件和 MFC 智能设备 DLL。

MFC for Windows CE 向导 (C++)

MFC for Windows CE 向导 (C++) 包括：

- MFC 智能设备应用程序向导

  MFC 智能设备应用程序向导生成具有 Windows CE 可执行文件 (.exe) 应用程序的内置功能和基本功能的应用程序。有关更多信息，请参见：

- “MFC 智能设备应用程序向导”的“高级功能”
- “MFC 智能设备应用程序向导”的“应用程序类型”
- “MFC 智能设备应用程序向导”的“平台”
- “MFC 智能设备应用程序向导”的“用户界面功能”
- “MFC 智能设备应用程序向导”的“文档模板字符串”
- “MFC 智能设备应用程序向导”的“生成的类”
- MFC 智能设备 ActiveX 控件向导

  MFC 向导生成的项目包括 C++ 源文件 (.cpp)、资源文件 (.rc) 和一个项目文件 (.vcproj)。有关更多信息，请参见：

- “MFC 智能设备 ActiveX 控件向导”的“控件设置”
- “MFC 智能设备 ActiveX 控件向导”的“控件名称”
- “MFC 智能设备 ActiveX 控件向导”的“平台”
- “MFC 智能设备 ActiveX 控件向导”的“应用程序设置”
- MFC 智能设备 DLL 向导

  可以使用 MFC 智能设备控件 DLL 项目创建智能设备 DLL。有关更多信息，请参见：

- “MFC 智能设备 DLL 向导”的“应用程序设置”
- “MFC 智能设备 DLL 向导”的“平台”.

请参见

任务

演练：为智能设备创建多平台 MFC 应用程序

如何：查找有关设备支持的 MFC 类和方法的帮助

Visual Studio 帮助提供目录和索引的“智能设备开发”筛选器。应用此筛选器时，可见主题会减少为仅包括对于智能设备开发人员而言必不可少的文档。如果要仅显示参考主题中与智能设备有关的部分，包括 MFC 库，则应用此筛选器是一种绝佳的方法。

要查看 C++ 主题和参考文档，应使用“Visual C++ 设置”。

如果应用语言筛选器，如 Visual C#，则可查看的主题中不包括智能设备开发主题。为此，最好使用“智能设备开发”筛选器，或者根本不使用筛选器。

使用语言开发设置（如“Visual C# 开发设置”）启动 Visual Studio。可以将此默认设置更改为“Visual C++ 设置”。

如何设置智能设备开发筛选器

设置智能设备开发筛选器

1. 在 Visual Studio 的“帮助”菜单上，单击“目录”或“索引”。
2. 在“筛选依据”框中，选择“智能设备开发”。

更改开发设置

1. 在 Visual Studio 的“工具”菜单上单击“导入和导出设置”。
2. 在向导的“欢迎使用”页上，单击“重置所有设置”，然后单击“下一步”。

   如果不确定应选择哪些选项，请按 F1 打开向导的帮助主题。
3. 在“保存当前设置”页上，选择是否保存当前设置，然后单击“下一步”。
4. 在“选择一个默认设置集合”页上，选择“Visual C++ 设置”，然后单击“完成”。

请参见

概念

设备不支持的桌面 MFC 类的列表

Visual Studio 的帮助筛选器

其他资源

智能设备项目入门

从 MFC 3.0 升级到 8.0 后不受支持的 eVC 类的列表

本主题已针对 Visual Studio 2005 SP1 进行了更新。

Embedded Visual C++

- CDaoFieldExchange Class
- CDBVariant Class
- CFieldExchange Class
- CFontDialog Class
- CLongBinary Class
- COleCmdUI Class
- COleCurrency Class
- COleDataObject Class
- COlePropertyPage Class
- CPrintDialog Class
- CPrintInfo Members
- CSocketFile Class

在 Visual Studio 2005 SP1 中，设备 MFC 库中添加了 15 个 eVC MFC 类，以改善嵌入式 Visual C++ 项目移植。以下类在设备的 Visual Studio 2005 MFC 中不受支持，但在设备的 Visual Studio 2005 SP1 中受支持。

- CBitmap Class
- CDialogBar Class
- CEditView Class
- CFindReplaceDialog Class
- CHttpConnection Class
- CHttpFile Class
- CInternetConnection Class
- CInternetException Class
- CInternetFile Class
- CInternetSession Class
- COleSafeArray Class
- CReBar Class
- CReBarCtrl Class
- CRecentFileList Class
- CSplitterWnd Class

下面的类是 **typedef**，它们使用模板类提供等效的功能：

- CByteArray Class
- CDWordArray Class
- CMapPtrToPtr Class
- CMapPtrToWord Class

- CMapStringToOb Class
- CMapStringToPtr Class
- CMapStringToString Class
- CMapWordToOb Class
- CMapWordToPtr Class
- CObArray Class
- CObList Class
- COleSafeArray Class
- CPtrArray Class
- CPtrList Class
- CStringArray Class
- CStringList Class
- CUIntArray Class
- CWordArray Class

从 MFC 3.0 升级到 MFC 8.0 后 API 的行为差异

- CDocument::SaveModified 对话框类和关联资源已经从所有平台的 MFC 8.0 中删除。因此，在 Pocket PC 2003 和 Smartphone 2003 平台上，**DoSave** 和 **SaveModified** 方法在使用时没有默认文件名，这些方法也没有对文件名的默认提示（如自动生成的文件名）。但是，在 Pocket PC 2003 平台上提供了一个可以重写此行为并提示文件名的选项。在 Smartphone 平台上，如果您希望提示文件名，则可以调用 **CDocManager::DoPromptFileName,**。**DoSave** 和 **SaveModified** 方法的默认文件名行为在 Windows CE 平台上受支持，并且功能和在桌面上的相同。
- 设备的 MFC 8.0 不提供停靠支持。例如，不支持 **CCommandBar::m_pDockBar** 和 **CCommandBar::m_pDockContext** 成员。有关更多信息，请参见 CCommandBar 类。有关停靠支持的更多信息，请参见 Docking and Floating Toolbars。
- 在设备的 MFC 8.0 中，CDC::FrameRect 不再是 CDC Class 的成员。
- 在设备的 MFC 8.0 中，CCeDocList 被重命名为 CDocList 类。
- 在设备的 MFC 8.0 中，CCeSocket 功能封装在 CAsyncSocket Class 中。
- 在设备的 MFC 8.0 中，CFont::CreateFont 不受支持，可以改用 CFont::CreatePointFont。
- 在设备的 MFC 8.0 中，不再支持 **CCommandBar::m_pDockBar** 和 **CCommandBar::m_pDockContext** 成员。
- 在设备的 MFC 8.0 中，**LPINLINEIMAGEINFO** 结构被 INLINEIMAGEINFO 替换。
- Visual Studio 2005 向导生成的资源遵循 Windows Mobile 5.0 用户界面 (UI) 指南。这意味着所有应用程序的 **MenuBar** 类始终将符合惯例的“新建”按钮放在左边，而将“菜单”放在右边。因此，设备的 MFC 8.0 不支持 **m_bShowSharedNewButton** 变量。例如，如果您的应用程序代码使用的是 `wndCommandBar.m_bShowSharedNewButton = TRUE;`，则可以将该代码行注释掉并将您的应用程序移植到设备的 MFC 8.0。
- 如果应用程序代码使用的是 `ON_NOTIFY(DLN_CE_CREATE, AFXCE_ID_DOCLIST` 或 `OnCreateDocList,`，将会得到下面的编译错误：
- MainFrm.cpp(42) : 错误 C2065: 'DLN_CE_CREATE' : 未声明的标识符
- MainFrm.cpp(42) : 错误 C2065: 'AFXCE_ID_DOCLIST' : 未声明的标识符
- 在 MFC 8.0 中，您可以安全地使用 DLN_DOCLIST_CREATE、DLN_DOCLIST_DESTROY 和 AFX_ID_DOCLIST。
- 使用 MFC 8.0 时，您将无法链接到标准 CRT 库。
- 移植到 MFC 8.0 时，请包含 `# define _WIN32_WCE_PSPC`。在 MFC 8.0 中，默认情况下不会定义此标志。
- 有关更多信息，请参见设备不支持的桌面 MFC 类的列表。

请参见

概念

设备的 MFC C++ 和标准 MFC 之间的差异

其他资源

设备类的唯一 MFC

设备的 MFC C++ 和标准 MFC 之间的差异

由于内存等设备约束原因，用于设备的 MFC 并不支持标准桌面 MFC 所支持的所有类和功能。下表描述了用于设备的 Visual Studio 最新版本（MFC 8.0 版）中支持和不支持的类。

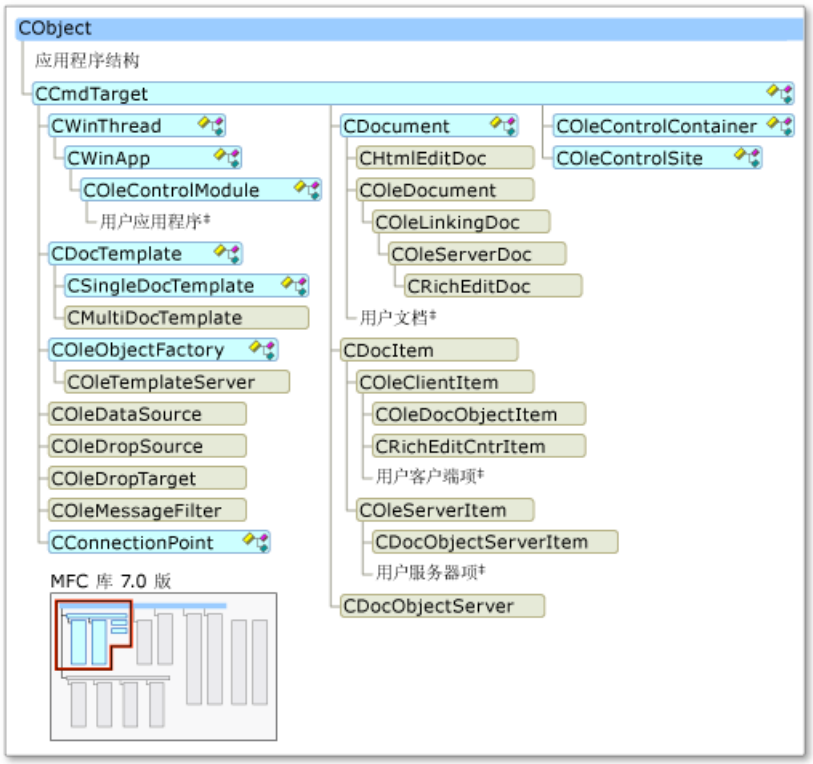
您也可以使用"帮助"功能的筛选机制来获得在桌面和设备上均受支持的 MFC 类的列表。

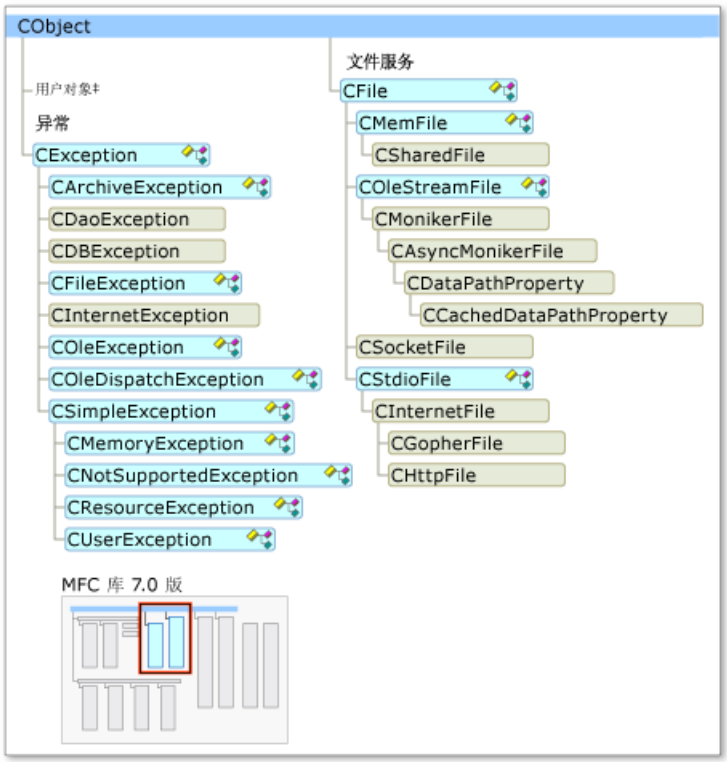
- 有关筛选用于设备的"MFC 参考"的帮助主题的更多信息（包括受支持的类和方法），请参见如何：查找有关设备支持的 MFC 类和方法的帮助。
- 有关设备支持的 MFC 类的更多信息，请参见设备支持的桌面 MFC 类的列表。
- 有关设备不支持的 MFC 类的更多信息，请参见设备不支持的桌面 MFC 类的列表。
- 下面的图表阐释了在桌面和 MFC C++ 设备上均受支持的类。

表示用于设备的 MFC 8.0 和标准桌面 MFC 7.0 之间区别的图表

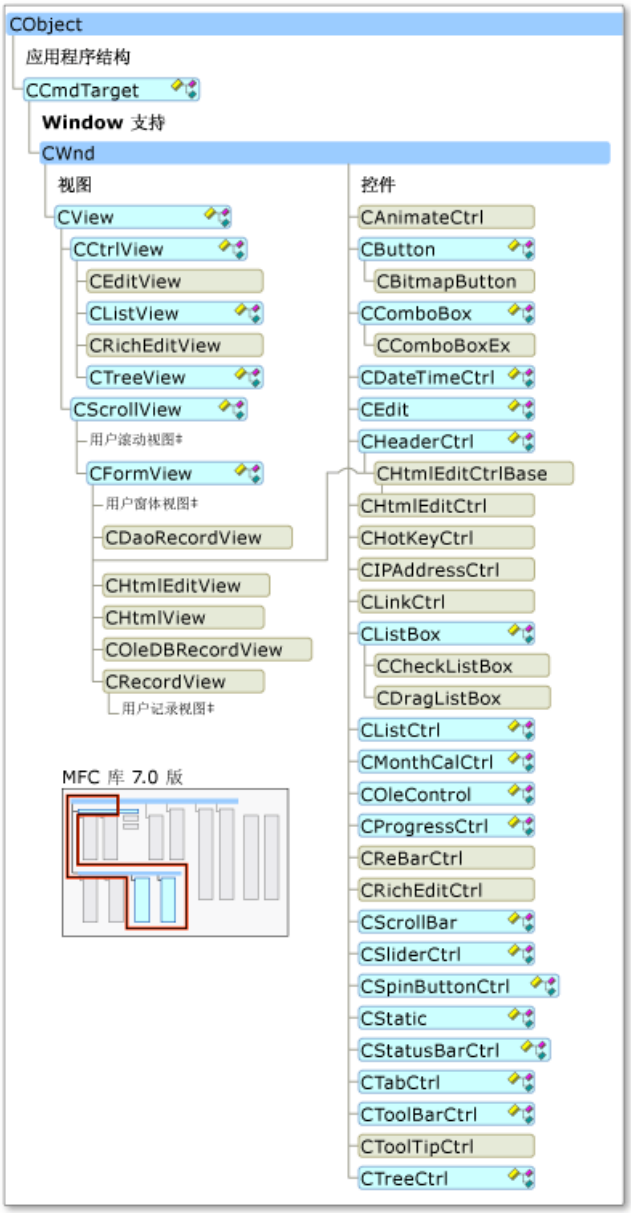
Legend

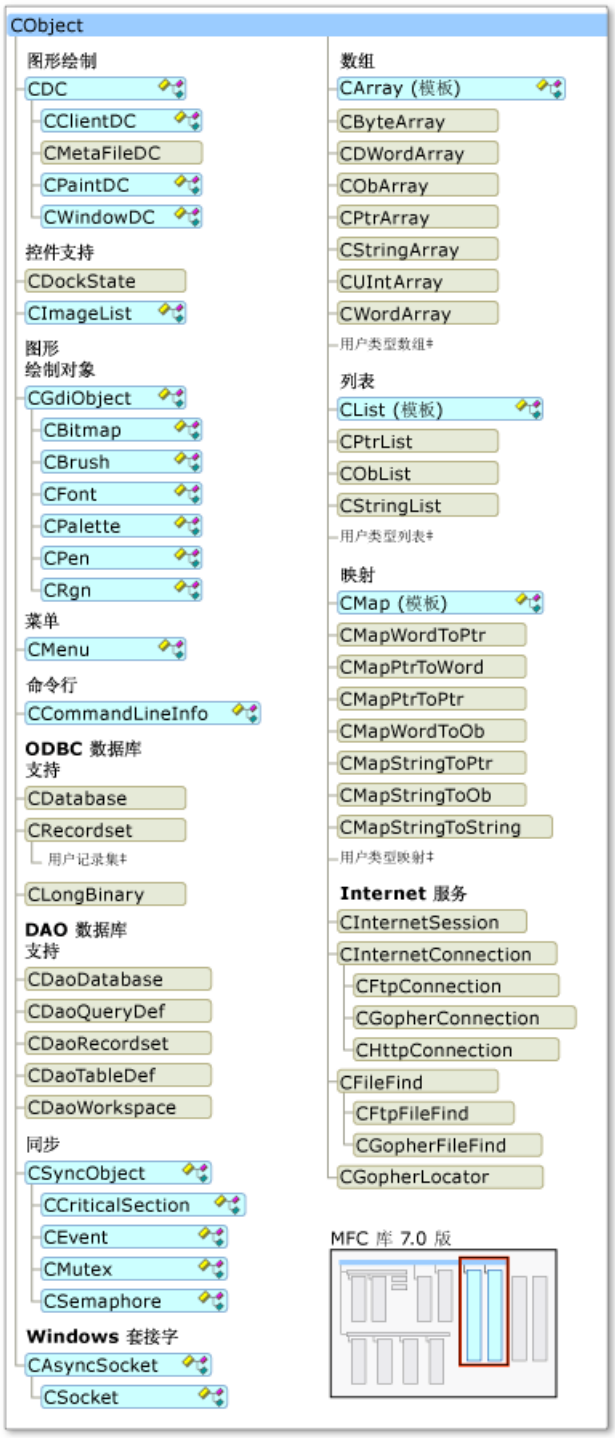
符号	说明
	用于设备的 MFC 8.0 中支持的标准桌面 MFC 7.0 类
	用于设备的 MFC 8.0 中不支持的标准桌面 MFC 7.0 类
‡	类的分类




CObject
用户对象‡
异常
CException
CArchiveException
CDaoException
CDBException
CFileException
CInternetException
COleException
COleDispatchException
CSimpleException
CMemoryException
CNotSupportedException
CResourceException
CUserException
文件服务
CFile
CMemFile
CSharedFile
COleStreamFile
CMonikerFile
CAsyncMonikerFile
CDataPathProperty
CCachedDataPathProperty
CSocketFile
CStdioFile
CInternetFile
CGopherFile
CHttpFile
MFC 库 7.0 版


CObject
应用程序结构
CCmdTarget
Window 支持
CWnd
框架窗口
CFrameWnd
CMDIChildWnd
用户 MDI 窗口‡
CMDIFrameWnd
用户 MDI 工作区‡
CMiniFrameWnd
用户 SDI 窗口‡
COleIPFrameWnd
CSplitterWnd
控制条
CControlBar
CDialogBar
COleResizeBar
CReBar
CStatusBar
CToolBar
属性表
CPropertySheet
对话框
CDialog
CCommonDialog
CColorDialog
CFileDialog
CFindReplaceDialog
CFontDialog
COleDialog
COleBusyDialog
COleChangeIconDialog
COleChangeSourceDialog
COleConvertDialog
COleInsertDialog
COleLinksDialog
COleUpdateDialog
COlePasteSpecialDialog
COlePropertiesDialog
CPageSetupDialog
CPrintDialog
CPrintDialogEx
COlePropertyPage
CPropertyPage
用户对话框‡
CDHtmlDialog
CMultiPageDHtmlDialog
MFC 库 7.0 版


CObject
应用程序结构
CCmdTarget
Window 支持
CWnd
视图
CView
CCtrlView
CEditView
CListView
CRichEditView
CTreeView
CScrollView
用户滚动视图‡
CFormView
用户窗体视图‡
CDaoRecordView
CHtmlEditView
CHtmlView
COleDBRecordView
CRecordView
用户记录视图‡
控件
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
MFC 库 7.0 版



请参见

其他资源

设备的 MFC 参考

设备类的唯一 MFC

设备支持的桌面 MFC 类的列表

本主题已针对 Visual Studio 2005 SP1 进行了更新。

以下是用于设备的 Visual Studio 最新版本所支持的 MFC 类。由于设备的限制，桌面上可用的某个类的部分功能在设备上不可用。

下面一节已针对 Visual Studio 2005 SP1 进行了更新。

用于设备的所支持的 MFC 类包括：

- CArchive Class
- CArchiveException Class
- CArray Class
- CAsyncSocket Class
- CBitmap Class
- CBitmap Class（在 Visual Studio 2005 SP1 中受支持）
- CBrush Class
- CButton Class
- CClientDC Class
- CCmdTarget Class
- CCmdUI Class
- CColorDialog Class
- CComboBox Class
- CCommandBar 类
- CCommandLineInfo Class
- CCommonDialog Class
- CConnectionPoint Class
- CControlBar Class
- CCriticalSection Class
- CCtrlView Class
- CDataExchange Class
- CDateTimeCtrl Class
- CDC Class
- CDialog Class
- CDialogBar Class（在 Visual Studio 2005 SP1 中受支持）
- CDocList 类
- CDocListDocTemplate 类
- CDocTemplate Class
- CDocument Class
- CEdit Class
- CEditView Class（在 Visual Studio 2005 SP1 中受支持）

- CEvent Class
- CException Class
- CFile Class
- CFileDialog Class
- CFileException Class
- CFindReplaceDialog Class（在 Visual Studio 2005 SP1 中受支持）
- CFixedStringT Class
- CFont Class
- CFontHolder Class
- CFormView Class
- CFrameWnd Class
- CGdiObject Class
- CHeaderCtrl Class
- CHttpConnection Class（在 Visual Studio 2005 SP1 中受支持）
- CHttpFile Class（在 Visual Studio 2005 SP1 中受支持）
- CImageList Class
- CInternetConnection Class（在 Visual Studio 2005 SP1 中受支持）
- CInternetException Class（在 Visual Studio 2005 SP1 中受支持）
- CInternetFile Class（在 Visual Studio 2005 SP1 中受支持）
- CInternetSession Class（在 Visual Studio 2005 SP1 中受支持）
- CInvalidArgException Class
- CList Class
- CListBox Class
- CListCtrl Class
- CListView Class
- CMap Class
- CMemFile Class
- CMemoryException Class
- CMenu Class
- CMonthCalCtrl Class
- CMultiLock Class
- CMutex Class
- CNotSupportedException Class
- CObject Class
- COccManager Class
- COleControl Class
- COleControlContainer Class
- COleControlModule Class

- COleControlSite Class
- COleDateTime Class
- COleDispatchDriver Class
- COleDispatchException Class
- COleException Class
- COleObjectFactory Class
- COlePropertyPage Class
- COleSafeArray Class（在 Visual Studio 2005 SP1 中受支持）
- COleStreamFile Class
- COleVariant Class
- CPaintDC Class
- CPalette Class
- CPen Class
- CProgressCtrl Class
- CPropertyPage Class
- CPropertySheet Class
- CPropExchange Class
- CReBar Class（在 Visual Studio 2005 SP1 中受支持）
- CReBarCtrl Class（在 Visual Studio 2005 SP1 中受支持）
- CRecentFileList Class（在 Visual Studio 2005 SP1 中受支持）
- CRectTracker Class
- CResourceException Class
- CRgn Class
- CScrollBar Class
- CScrollView Class
- CSemaphore Class
- CSimpleException Class
- CSimpleStringT Class
- CSingleDocTemplate Class
- CSingleLock Class
- CSliderCtrl Class
- CSocket Class
- CSocketFile Class
- CSpinButtonCtrl Class
- CSplitterWnd Class（在 Visual Studio 2005 SP1 中受支持）
- CStatic Class
- CStatusBar Class
- CStatusBarCtrl Class

- CStdioFile Class
- CStringT Class
- CSyncObject Class
- CTabCtrl Class
- CTime Class
- CTimeSpan Class
- CToolBar Class
- CToolBarCtrl Class
- CTreeCtrl Class
- CTreeView Class
- CTypedPtrList Class
- CUserException Class
- CView Class
- CWinApp Class
- CWindowDC Class
- CWinThread Class
- CWnd Class

这些类之间存在一些行为差异。有关更多信息，请参见从 MFC 3.0 升级到 8.0 后不受支持的 eVC 类的列表。

要查看受支持的 MFC 主题和 MFC 参考的详细信息，可以使用开发环境的“帮助”的筛选机制。有关此筛选机制的更多信息，请参见如何：查找有关设备支持的 MFC 类和方法的帮助。

请参见

概念

从 MFC 3.0 升级到 8.0 后不受支持的 eVC 类的列表

其他资源

设备的 MFC 参考

设备类的唯一 MFC

设备不支持的桌面 MFC 类的列表

本主题已针对 Visual Studio 2005 SP1 进行了更新。

由于设备的限制，在桌面上可用的 MFC 类的功能可能在设备上不可用。有关设备支持的 MFC 类的列表，请参见设备支持的桌面 MFC 类的列表。在设备的 MFC 中不支持的标准桌面 MFC 类包括：

- CAnimateCtrl Class
- CAsyncMonikerFile Class
- CByteArray Class
- CCachedDataPathProperty Class
- CCheckListBox Class
- CComboBoxEx Class
- CDaoDatabase Class
- CDaoException Class
- CDaoFieldExchange Class
- CDaoQueryDef Class
- CDaoRecordset Class
- CDaoRecordView Class
- CDaoTableDef Class
- CDaoWorkspace Class
- CDatabase Class
- CDataPathProperty Class
- CDBException Class
- CDBVariant Class
- CDHtmlDialog Class
- CDocItem Class
- CDockState Class
- CDocObjectServer Class
- CDocObjectServerItem Class
- CDragListBox Class
- CDumpContext Class
- CDWordArray Class
- CFieldExchange Class
- CFileFind Class
- CFileTime Class
- CFileTimeSpan Class
- CFontDialog Class
- CFtpConnection Class
- CFtpFileFind Class

- CGopherConnection Class
- CGopherFile Class
- CGopherFileFind Class
- CGopherLocator Class
- CHotKeyCtrl Class
- CHtmlEditCtrl Class
- CHtmlEditCtrlBase Class
- CHtmlEditDoc Class
- CHtmlEditView Class
- CHtmlStream Class
- CHtmlView Class
- CHttpArg Structure
- CHttpArgList Class
- CHttpFilter Class
- CHttpFilterContext Class
- CHttpServer Class
- CHttpServerContext Class
- CImage Class
- CIPAddressCtrl Class
- CLinkCtrl Class
- CLongBinary Class
- CMapPtrToPtr Class
- CMapPtrToWord Class
- CMapStringToOb Class
- CMapStringToPtr Class
- CMapStringToString Class
- CMapWordToOb Class
- CMapWordToPtr Class
- CMDIChildWnd Class
- CMDIFrameWnd Class
- CMetaFileDC Class
- CMiniFrameWnd Class
- CMonikerFile Class
- CMultiDocTemplate Class
- CMultiPageDHtmlDialog Class
- CObArray Class
- CObList Class
- COleBusyDialog Class

- COleChangeIconDialog Class
- COleChangeSourceDialog Class
- COleClientItem Class
- COleCmdUI Class
- COleConvertDialog Class
- COleCurrency Class
- COleDataObject Class
- COleDataSource Class
- COleDateTimeSpan Class
- COleDBRecordView Class
- COleDialog Class
- COleDocObjectItem Class
- COleDocument Class
- COleDropSource Class
- COleDropTarget Class
- COleInsertDialog Class
- COleIPFrameWnd Class
- COleLinkingDoc Class
- COleLinksDialog Class
- COleMessageFilter Class
- COlePasteSpecialDialog Class
- COlePropertiesDialog Class
- COleResizeBar Class
- COleServerDoc Class
- COleServerItem Class
- COleTemplateServer Class
- COleUpdateDialog Class
- CPageSetupDialog Class
- CPictureHolder Class
- CPrintDialog Class
- CPrintDialogEx Class
- CPrintInfo Members
- CPoint Class
- CPtrArray Class
- CPtrList Class
- CRecordset Class
- CRecordView Class
- CRect Class

- CRichEditCntrItem Class
- CRichEditCtrl Class
- CRichEditDoc Class
- CRichEditView Class
- CSharedFile Class
- CSize Class
- CStrBufT Class
- CStringArray Class
- CStringData Class
- CStringList Class
- CToolTipCtrl Class
- CTypedPtrArray Class
- CTypedPtrMap Class
- CUIntArray Class
- CWaitCursor Class
- CWinFormsControl Class
- CWinFormsDialog Class
- CWinFormsView Class
- CWordArray Class

在 Visual Studio 2005 SP1 中，设备 MFC 库中添加了 15 个桌面 MFC 类。以下类在设备的 Visual Studio 2005 MFC 中不受支持，但在设备的 Visual Studio 2005 SP1 中受支持。

- CBitmap Class
- CDialogBar Class
- CEditView Class
- CFindReplaceDialog Class
- CHttpConnection Class
- CHttpFile Class
- CInternetConnection Class
- CInternetException Class
- CInternetFile Class
- CInternetSession Class
- COleSafeArray Class
- CReBar Class
- CReBarCtrl Class
- CRecentFileList Class
- CSplitterWnd Class

请参见

参考

设备支持的桌面 MFC 类的列表

概念

从 MFC 3.0 升级到 8.0 后不受支持的 eVC 类的列表

其他资源

设备的 MFC 参考

设备类的唯一 MFC

MFC Classes

设备类的唯一 MFC

设备所特有的具有若干唯一类的 MFC。本节即介绍这些类。

本节内容

CCommandBar 类

结合菜单栏和工具栏的功能，且专为具有小显示屏的设备而设计。

CDocList 类

将 **DocList** 窗口类型的操作系统实现以及将与 MFC 文档/视图结构集成的其他功能包装在一起。

CDocListDocTemplate 类

SDI 应用程序类型的优化，由 CSingleDocTemplate 帮助实现。

AfxEnableDRA

启用设备分辨率识别功能的函数。

相关章节

设备的 MFC 参考

如何：查找有关设备支持的 MFC 类和方法的帮助

CCommandBar 类

使用该类可以创建和修改命令栏。命令栏是一个工具栏，它可以包含一个菜单栏以及一个“关闭”按钮、“帮助”按钮和“确定”按钮。命令栏可以包含菜单、组合框、按钮和分隔符。分隔符是一段空白，用于将其他元素分组，或者在命令栏中保留空白。创建 **CCommandBar** 对象后，可以使用 InsertMenuBar、InsertComboBox 和 InsertSeparator 方法来分别插入菜单栏、组合框和分隔符。将其他所有元素添加到命令栏之后，使用 AddAdornments 方法来添加“取消”按钮，还可以添加“帮助”和“确定”按钮。每次修改命令栏上的菜单后，都会使用 DrawMenuBar 方法来重绘命令栏。

在 Windows CE 5.0 中，不再支持 **CDialog::m_pWndEmptyCB** 成员，您可以控制创建和插入过程。以前，该成员变量用于指向 Pocket PC 上的空 **CommandBar** 或 **MenuBar**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCommandBar 方法

下表列出了 **CCommandBar** 方法：

CCommandBar::AddAdornments	将“关闭”按钮以及可选的“帮助”和“确定”按钮添加到命令栏。
CCommandBar::AddBitmap	该方法将一个或多个图像添加到可用在命令栏中的按钮图像的列表中。
CCommandBar::AddButtons	向工具栏控件添加一个或多个按钮。
CCommandBar::CCommandBar	创建和初始化一个 **CCommandBar** 对象。
CCommandBar::Create	实例化新的命令栏。
CCommandBar::DrawMenuBar	修改命令栏中的菜单后，重定位和重绘命令栏。
CCommandBar::GetMenu	检索命令栏上的菜单的句柄。
CCommandBar::Height	检索命令栏的高度（以像素为单位）。
CCommandBar::InsertButton	此方法向工具栏控件中添加一个或多个按钮。
CCommandBar::InsertComboBox	向命令栏中添加组合框。
CCommandBar::InsertMenuBar	向命令栏中添加菜单栏。
CCommandBar::InsertSeparator	向命令栏中添加分隔符。
CCommandBar::Show	显示或隐藏命令栏。

CCeDocList 在 MFC 8.0 中，类已变成 CDocList 类。下面是来自 MFC 3.0 应用程序的代码：

```
CCommandBar* pDocListCB = pDocList->GetCommandBar();
```

此代码生成以下错误：

MainFrm.cpp(115): 错误 C2039:“GetCommandBar”: 不是“CDocList”的成员

可以在 MFC 8.0 中创建 **CDocListCommandBar** 对象来代替使用此代码。

请参见

参考

CCommandBar 类

其他资源

设备类的唯一 MFC

CCeCommandBar

CCommandBar::AddAdornments

将“关闭”按钮以及可选的“帮助”和“确定”按钮添加到命令栏。

```
BOOL AddAdornments(
Dword dwFlags = 0 );
```

参数

下表描述了 CCommandBar 类的 **AddAdornments** 方法的参数。

参数	说明
dwFlags	指定要添加到命令栏的可选按钮。如果只需要“关闭”按钮，则该值为零；如果要创建其他按钮，则为下列值的组合：

值	说明
CMDBAR_HELP	向命令栏中添加“帮助”按钮。单击该按钮后，会发送 WM_HELP 消息。
CMDBAR_OK	向命令栏中添加“确定”按钮。选定该按钮后，会发送消息标识符为 IDOK 的 WM_COMMAND 消息。

返回值

如果成功，则为 **true**；否则为 **false**。

备注

此方法至少创建两个按钮：将选定修饰与命令栏右侧对齐的分隔符以及选定的修饰。

当用户单击“关闭”、“确定”或“帮助”按钮时，与该按钮关联的消息放在应用程序的消息队列中。每个命令栏都必须有“关闭”按钮。“确定”按钮和“帮助”按钮是可选的。

在其他所有元素（如菜单、按钮和组合框）都添加到命令栏后，调用此方法。

应重新设计现有的应用程序以去除对 **AddAdornments** 的依赖性。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCommandBar::AddBitmap

将一个或多个图像添加到可用在命令栏中的按钮图像的列表中。

```
int AddBitmap (
int nBitMapID,
int nNumImages );
```

参数

下表描述了 CCommandBar 类的 **AddBitmap** 方法的参数。

nBitMapID

指定包含要添加的按钮图像的位图的资源标识符。

nNumImages

指定位图中的按钮图像的数目。

返回值

如果成功，则为第一个新图像的从零开始的索引，否则为 –1。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCommandBar::AddButtons

向工具栏控件添加一个或多个按钮。

```
BOOL AddButtons(
UINT uNumButtons,
LPTBBUTTON lpButtons );
```

参数

下表描述了 CCommandBar 类的 **AddButtons** 方法的参数。

uNumButtons

指定要添加的按钮的数目。

lpButtons

指定 TBBUTTON 结构的数组的地址，该数组包含有关要添加的按钮的信息。数组中的元素的数目必须与 *nNumButtons* 指定的按钮的数目相同。

返回值

如果成功，则为 **true**；否则为 **false**。

备注

lpButtons 指针指向 TBBUTTON 结构的数组。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCeCommandBar

CCommandBar::CCommandBar

创建 **CCommandBar** 对象的一个实例。

```
CCommandBar ( );
```

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCommandBar::Create

实例化一个新的命令栏。

```
BOOL CommandBar::Create(
        CWnd* pWndParent,
    DWORD dwStyle,
    UINT nBarID = AFX_IDW_TOOLBAR
);
```

参数

下表描述了 CCommandBar 类的 **Create** 方法的参数。

pWndParent

指向命令栏对象的父级的指针。

dwStyle

指定命令栏样式。请参见备注。

nBarID

可选参数。指定工具栏的子窗口 ID = AFX_IDW_TOOLBAR。

返回值

如果成功，则为 **true**；否则为 **false**。

备注

Create 函数创建一个空命令栏。在 Pocket PC 和 Smartphone 上，命令栏是在窗口底部创建的。在 Windows CE 5.0 中，不再支持 **CDialog::m_pWndEmptyCB**，而您控制创建和插入进程。以前，此成员变量用于指向 Pocket PC 上的空 **CommandBar** 或 **MenuBar**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCommandBar::DrawMenuBar

修改命令栏中的菜单后，重定位和重绘命令栏。

```
BOOL DrawMenuBar(
WORD wButton )
const;
```

参数

下表描述了 CCommandBar 类的 **DrawMenuBar** 方法的参数。

wButton

指定命令栏中的菜单栏的索引(从 0 开始)。

返回值

如果成功，则为 **true**；否则为 **false**。

备注

修改命令栏中的菜单后总是调用该方法。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCeCommandBar

CCommandBar::GetMenu

检索命令栏上的菜单的句柄。

```
HMENU CommandBar_GetMenu();
```

返回值

菜单的句柄表示成功。**NULL** 表示失败。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCeCommandBar

CCommandBar::Height

检索命令栏的高度(以像素为单位)。

```
int Height();
```

返回值

返回命令栏的高度(以像素为单位)。

备注

命令栏占据了应用程序主窗口的部分工作区，可以将调用此函数作为确定工作区实际大小的一种方法。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCommandBar::InsertButton

向工具栏控件添加一个或多个按钮。

```
BOOL InsertButton(
int nButton,
LPTBBUTTON lpButton );
```

参数

下表描述了 CCommandBar 类的 **InsertButton** 方法的参数。

nButton

指定按钮的从零开始的索引。此方法将新按钮插入到此按钮的左侧。

lpButton

指定一个 TBUTTON 结构的地址，该结构包含有关要插入的按钮的信息。

返回值

如果成功，则为 **true**；否则为 **false**。

备注

lpButton 指针指向 TBUTTON 结构。

要求

Windows CE 5.0 版或更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCeCommandBar

CCommandBar::InsertComboBox

向命令栏中插入组合框。

```
CComboBox* InsertComboBox(
int nWidth,
DWORD dwStyle,
WORD wComboBoxID,
WORD wButton );
```

参数

下表描述了 CCommandBar 类的 **InsertComboBox** 方法的参数。

nWidth

以像素为单位指定组合框的宽度。

dwStyle

指定应用于组合框的窗口样式。WS_VISIBLE 和 WS_CHILD 样式是自动应用的。默认样式为 CBS_DROPDOWNLIST 和 WS_SCROLL。

wComboBoxID

指定组合框的标识符。

wButton

指定命令栏中的按钮的索引(从零开始)。

返回值

如果成功，则为指向由 *wComboBoxID* 标识的菜单的指针；否则为 **null**。

备注

此方法将组合框插入由 w*Button* 标识的按钮的左侧。

要求

Windows CE 5.0 版或更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCeCommandBar

CCommandBar::InsertMenuBar

将菜单栏插入命令栏。

```
BOOL InsertMenuBar(
WORD wMenuID,
WORD wButton );
```

参数

下表描述了 CCommandBar 类的 **InsertMenuBar** 方法的参数。

wMenuID

指定 **MenuBar** 的资源 ID。

wButton

指定命令栏中的按钮的索引(从零开始)。

返回值

如果成功，则为 **true**；否则为 **false**。

备注

该方法将 **MenuBar** 放在 *wButton* 所指示的按钮位置。

要求

Windows CE 5.0 版或更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCeCommandBar

CCommandBar::InsertSeparator

在命令栏上插入分隔符按钮。

```
BOOL InsertSeparator(
int nWidth = 6,
WORD wButton );
```

参数

下表描述了 CCommandBar 类的 **InsertSeparator** 方法的参数。

nWidth

指定分隔符按钮的宽度，以像素为单位。

wButton

指定命令栏中的按钮的索引(从零开始)，该索引大于或等于零，但小于命令栏中的按钮的数目。CMDBAR_END 指定 **CommandBar** 的末尾。

返回值

如果成功，则为 **true**；否则为 **false**。

备注

该方法将分隔符放在 *wButton* 所指示的按钮位置。分隔符不接收用户输入。

要求

Windows CE 5.0 版或更高版本。

头文件：在 afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCeCommandBar

CCommandBar::Show

显示或隐藏命令栏。

```
BOOL Show(BOOL bShow)
```

参数

下表描述了 CCommandBar 类的 **Show** 方法的参数。

bShow

指定要显示还是隐藏命令栏的布尔值。将此参数设置为 **false** 以显示命令栏，或者将它设置为 **true** 以隐藏命令栏。

返回值

返回 **CCommandBar** 以前的显示状态。如果以前显示命令栏，则返回 **true**，如果以前隐藏命令栏，则返回 **false**。

备注

创建命令栏时，其显示状态设置为 **true**。若要隐藏命令栏，请调用 `Show(false)`。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CCommandBar 方法

其他资源

设备类的唯一 MFC

CCeCommandBar

CDocList 类

使用 **CDocList** 可以一致的方式显示文档列表。**CDocList** 类将 **DocList** 窗口类型的操作系统实现以及将与 MFC 文档/视图结构集成的其他功能包装在一起。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

其他资源

设备类的唯一 MFC

CDocList 方法

CCeDocList

CDocList 方法

下面是 CDocList 类的方法，该类包装 **DocList** 窗口类型的操作系统实现。

CDocList::CDocList	构造 **CDocList** 对象。
CDocList::Create	创建并初始化与 **CDocList** 对象关联的列表。
CDocList::DeleteSelection	删除所选项。
CDocList::DisableUpdate	禁用 **CDocList** 的更新。
CDocList::EnableUpdate	启用 **CDocList** 的更新。
CDocList::GetFilterIndex	为设置当前筛选器而获取当前文件筛选器选择，例如，设置组合框的当前筛选器。
CDocList::GetItemCount	获取项的计数。
CDocList::GetNextItem	获取 **CDocList** 中的下一项。
CDocList::GetNextWaveFile	选择下一个 .wav 文件。
CDocList::GetPrevWaveFile	选择上一个 .wav 文件。
CDocList::GetSelectCount	获取所选项的数目。
CDocList::GetSelectedPathname	获取与所选项关联的路径名。
CDocList::ReceiveIR	接收红外信号。
CDocList::Refresh	刷新 **CDocList**。
CDocList::RenameMoveSelectedItems	显示所选项的“重命名/移动”对话框。
CDocList::SelectItem	选择与路径关联的项。
CDocList::SendEMail	发送电子邮件。
CDocList::SendIR	发送红外信号。
CDocList::SetFilterIndex	指示文件筛选器已更改；例如，用户已更改“选项”对话框中的设置。
CDocList::SetFolder	设置一个文件夹。
CDocList::SetItemState	更改 **CDocList** 中某项的状态。
CDocList::SetSelectedPathname	使与路径关联的项在下一次刷新时被选定。
CDocList::SetSelection	按索引选择项。
CDocList::SetSortOrder	将 **CDocList** 排序。

CDocList::Update	刷新 **CDocList**，但不还原选择。

CCeDocList 在 MFC 8.0 中，类已变成 CDocList 类。下面是来自 MFC 3.0 应用程序的代码：

```
CCommandBar* pDocListCB = pDocList->GetCommandBar();
```

此代码生成以下错误：

MainFrm.cpp(115): 错误 C2039:“GetCommandBar”: 不是“CDocList”的成员

在 MFC 8.0 中，改为创建 **CDocListCommandBar** 的实例。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::CDocList

创建 **CDocList** 对象的实例。

```
CDocList();
```

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CDocList::Create

创建并初始化与 **CDocList** 对象关联的列表。

```
BOOL Create (
CWnd* pParentWnd,
LPCTSTR lpszFilterList,
LPCTSTR lpszFolder,
DWORD dwFlags = DLF_SHOWEXTENSION ) ;
```

参数

下表描述了 CDocList 类的 **Create** 方法的参数。

参数	说明
pParentWnd	指向父窗口的指针 - 必选。
lpszFilterList	指向筛选器列表的指针，这是一个以竖线分隔、以 ||\0 终止的列表。
lpszFolder	指向文件夹的指针。
dwFlags	DLF_SHOWEXTENSION 是当前版本中唯一允许的值。

返回值

如果已成功创建并显示列表，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CDocList::DeleteSelection

删除所选项。

```
HRESULT DeleteSelection();
```

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CDocList::DisableUpdate

禁止更新文档列表。

```
void DisableUpdate();
```

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::EnableUpdate

允许更新文档列表。

```
void EnableUpdate();
```

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::GetFilterIndex

获取当前的文件筛选器选择。

```
int GetFilterIndex();
```

返回值

当前筛选器选择在以竖线 (|) 分隔的筛选器列表中的索引。该索引从 1 开始而不是从 0 开始。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::GetItemCount

获取项的计数。

```
int GetItemCount();
```

返回值

文档列表中的文档数。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::GetNextItem

获取文档列表中的下一项。

```
int GetNextItem(
int nStart,
UINT nFlags );
```

参数

下表描述了 CDocList 类的 **GetNextItem** 方法的参数。

参数	说明
nStart	指定文档列表中搜索开始处的项编号。
nFlags	指定所请求的项与指定的项的几何关系以及所请求的项的状态。该几何关系可以是下列值之一： 值 / 说明 LVNI_ABOVE / 搜索指定项上面的项。 LVNI_ALL / 按索引搜索后续项，这是默认值。 LVNI_BELOW / 搜索指定项下面的项。 LVNI_TOLEFT / 搜索指定项左侧的项。 LVNI_TORIGHT / 搜索指定项右侧的项。 另外，您还可以指定项的状态，该状态可以是零，也可以是下面的一个或多个值： 值 / 说明 LVNI_DROPHILITED / 该项已设置 LVIS_DROPHILITED 状态标志。 LVNI_FOCUSED / 该项已设置 LVIS_FOCUSED 状态标志。 LVNI_SELECTED / 该项已设置 LVIS_SELECTED 状态标志。

备注

如果某项尚未设置任何指定状态标志，则会继续搜索下一项。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::GetNextWaveFile

获取下一个 .wav 文件。

```
BOOL GetNextWaveFile(
int* pnItem );
```

参数

下表描述了 CDocList 类的 **GetNextWaveFile** 方法的参数。

参数	说明
pnItem	指向列表中下一个 .wav 文件的指针。

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CDocList::GetPrevWaveFile

获取上一个 .wav 文件。

```
BOOL GetPrevWaveFile(
int* pnItem );
```

参数

下表描述了 CDocList 类的 **GetPrevWaveFile** 方法的参数。

参数	说明
pnItem	指向列表中的上一个 .wav 文件的指针。

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CDocList::GetSelectCount

获取选定项的数目。

```
int GetSelectCount();
```

返回值

文档列表中选定项的数目。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CDocList::GetSelectedPathname

获取与所选项关联的路径。

```
BOOL GetSelectedPathname(
LPTSTR pszPath,
WORD nSize );
```

参数

下表描述了 CDocList 类的 **GetSelectedPathname** 方法的参数。

pszPath

指向文档列表中所选项的路径的指针。

nSize

以字符为单位，指定接收路径的缓冲区的长度。

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CDocList::ReceiveIR

用于接收红外信息。

```
void ReceiveIR(
LPTSTR pszPath );
```

参数

下表描述了 CDocList 类的 **ReceiveIR** 方法的参数。

pszPath

指向包含路径的字符串的指针。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::Refresh

刷新文档列表。

```
BOOL Refresh();
```

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::RenameMoveSelectedItems

显示所选项的"重命名/移动"对话框。

```
BOOL RenameMoveSelectedItems();
```

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CDocList::SelectItem

导致与此路径关联的项被选中。

```
HRESULT SelectItem(
LPTSTR pszPath,
BOOL fVisible );
```

参数

下表描述了 CDocList 类的 **SelectItem** 方法的参数。

参数	说明
pszPath	指向包含路径的字符串的指针。
fVisible	指示文档已变为可见还是不可见。

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::SendEMail

用于发送电子邮件。

```
void SendEMail(
LPTSTR pszPath );
```

参数

下表描述了 CDocList 类的 **SendEMail** 方法的参数。

参数	说明
pszPath	指向包含路径的字符串的指针。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::SendIR

用于发送红外信息。

```
void SendIR(
LPTSTR pszPath );
```

参数

下表描述了 CDocList 类的 **SendIR** 方法的参数。

参数	说明
pszPath	指向包含路径的字符串的指针。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::SetFilterIndex

指示文件筛选器已更改。

```
HRESULT SetFilterIndex(
int nIndex );
```

参数

下表描述了 CDocList 类的 **SetFilterIndex** 方法的参数。

参数	说明
nIndex	以竖线 (|) 分隔的筛选器列表索引。该索引从 1 开始而不是从 0 开始。

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::SetFolder

设置一个文件夹。

```
void SetFolder(
LPTSTR pszFolder );
```

参数

下表描述了 CDocList 类的 **SetFolder** 方法的参数。

参数	说明
pszFolder	指向包含文件夹名称的字符串的指针。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::SetItemState

更改文档列表中某项的状态。

```
BOOL SetItemState(
int nItem,
UINT nState,
UINT nStateMask );
```

参数

下表描述了 CDocList 类的 **SetItemState** 方法的参数。

参数	说明
nItem	项的索引号。
nState	指定要更改的状态的值。
nStateMask	指定要更改的状态。

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::SetSelectedPathname

使与路径关联的项在下一次刷新时被选定。

```
void SetSelectedPathname(
LPTSTR pszPath );
```

参数

下表描述了 CDocList 类的 **SetSelectedPathname** 方法的参数。

参数	说明
pszPath	指向包含路径的字符串的指针。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::SetSelection

按索引选择项。

```
BOOL SetSelection(
int wItem );
```

参数

下表描述了 CDocList 类的 **SetSelection** 方法的参数。

参数	说明
wItem	项的索引号。

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CDocList::SetSortOrder

排序文档列表。排序顺序始终是升序。

```
BOOL SetSortOrder();
```

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocList::Update

刷新文档列表，但不还原选择。

```
HRESULT Update();
```

返回值

如果成功，则为 **true**；否则为 **false**。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

参考

CDocList 类

其他资源

设备类的唯一 MFC

CCeDocList

CDocListDocTemplate 类

此类是 SDI 应用程序类型的优化，由 CSingleDocTemplate Class 帮助实现。通过使用此类，可以在不打开文档的情况下将应用程序置于 **DocList** 模式。

将此文档类型添加到应用程序后，调用 **ShowDocList** 成员最初会创建 **CDocList** 类的实例。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

其他资源

设备类的唯一 MFC

CDocListDocTemplate 方法

CDocListDocTemplate 类是与桌面 **CDocListDocTemplate** 类似的一个类，后者是 SDI 应用程序类型的优化形式。

该类使应用程序可以在没有文档存在的情况下处于文档模式下。

将此文档类型添加到应用程序后，调用 **ShowDocList** 成员会创建一个 **CDocList** 实例。

本节内容

CDocListDocTemplate::CDocListDocTemplate

构造 **CDocListDocTemplate** 对象。

CDocListDocTemplate::ShowDocList

显示文档列表。调用 **CDocListDocTempate::ShowDocList** 时，会自动创建 **CDocList**。

请参见

其他资源

设备类的唯一 MFC

CCeDocListDocTemplate

CDocListDocTemplate::CDocListDocTemplate

构造 **CDocListDocTemplate** 对象。使用此类，可以动态分配 **CDocListDocTemplate** 对象，并将它从应用程序类的 **InitInstance** 方法传递到 **CWinApp::AddDocTemplate**。

```
CDocListDocTemplate(
UINT nIDResource,
CRuntimeClass* pDocClass,
CRuntimeClass* pFrameClass,
CRuntimeClass* pViewClass,
LPCTSTR lpszFilterList,
LPCTSTR lpszFolder );
```

参数

下表描述了 CDocListDocTemplate 类的 **CDocListDocTemplate** 方法的参数。

参数	说明
nIDResource	指定与文档类型一起使用的资源标识符。这可能包括菜单、图标、快捷键对应表以及字符串资源。
pDocClass	指向文档类的 **CRuntimeClass** 对象的指针。该类是一个 **CDocument** 派生类，定义它是为了表示文档。
pFrameClass	指向框架窗口类的 **CRuntimeClass** 对象的指针。该类可以是 **CFrameWnd** 派生类，如果您需要主框架窗口的默认行为，那么它还可以是 **CFrameWnd**。
pViewClass	指向视图类的 **CRuntimeClass** 对象的指针。该类是一个 **CView** 派生类，定义它是为了显示文档。
lpszFilterList	指向筛选器列表的指针，这是一个以竖线分隔、以 ||\0 终止的列表。
lpszFolder	指向文件夹的指针。

备注

有关更多信息，请参见 CDocListDocTemplateCDocListDocTemplate。

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxext.h 中声明。

请参见

其他资源

设备类的唯一 MFC

CCeDocListDocTemplate

CDocListDocTemplate::ShowDocList

用作 **CWinApp::InitInstance** 发出的调用以便最初显示文档列表。

```
virtual void ShowDocList();
```

要求

Windows CE 5.0 版及更高版本。

头文件：在 Afxwin.h 中声明。

请参见

其他资源

设备类的唯一 MFC

AfxEnableDRA

此函数在设备应用程序项目中启用设备分辨率识别功能。

```
void AfxEnableDRA(BOOL bEnable);
```

参数

bEnable

指定 **TRUE** 将启用设备分辨率识别功能；指定 **FALSE** 或不调用该函数，将禁用设备分辨率识别功能。

备注

设备分辨率识别功能使应用程序可以在运行时响应分辨率的更改，例如，从纵向模式更改为横向模式。

请在直接实例化 **CDialog** 时使用 **AfxEnableDRA()** 函数。在此情况下，您使用的是在 dlgcore.cpp 中定义、在 MFC DLL 和 LIB 中实现的 **OnSize** 方法。在这些库版本中，**AfxIsDRAEnabled()** 用于执行运行时检查，以便确定是否调用 **DRA::RelayoutDialog(...)**。只有在先调用了 **`AfxEnableDRA(TRUE)`** 的情况下，**`AfxIsDRAEnabled()`** 才返回 `True`。

注意

当您使用向导创建用于设备的 MFC 项目时，生成的代码会为 **CDialog** 的派生类实现重写（**`CDialog::OnSize(int, int)`**）。随后在编译时检查设备分辨率识别功能，并确定调用 **`DRA::RelayoutDialog(...)`**。

示例

```
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);
    }
}
```

请参见

其他资源

设备类的唯一 MFC

针对设备的 C 运行时库参考

用于设备的 Microsoft C 运行时库支持用于桌面的完整 Microsoft C 运行时库中可用的函数的子集。

由于仅仅支持完整库的子集，因此应用程序所运行在的设备具有的资源要比桌面计算机具有的资源有限。有关筛选用于设备的运行时库参考的帮助主题的更多信息，请参见如何：查找有关设备支持的 MFC 类和方法的帮助和如何：查找有关设备支持的 ATL 类和方法的帮助。

C 运行时库

有关用于设备的 C 运行时库的完整文档，请参见用于 Windows CE 的 Microsoft C 运行时库。

安全 API

Visual Studio 2005 中的设备项目已添加下面的 API。

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	

请参见

其他资源

参考（设备）

设备的标准 C++ 库参考

标准 C++ 库的设备版本是完整的 Standard C++ Library Reference 的子集。

新功能

流支持已添加到这个版本的标准 C++ 库。

不支持的功能

- 用于设备的标准 C++ 库不包括区域设置支持。
- 只有 Windows CE 5.0 和更高版本(包括 Windows Mobile 2005 平台)才支持 `uncaught_exception`。

不受支持的标头

标准 C++ 库的设备版本不支持下列标头:

- <cerrno>
- <csignal>
- <locale>

请参见

其他资源

参考(设备)

Visual C++ Libraries Reference

Compilers for Smart Devices

Compilers for Smart Devices

Visual Studio 2005 includes the following compilers that target microprocessors used in smart devices:

- A 32-bit C/C++ compiler used to compile and link 32-bit ARM C and C++ programs.
- A 32-bit C/C++ compiler used to compile and link 32-bit Renesas SH-4 C and C++ programs.
- A C/C++ compiler used to compile and link MIPS16, MIPS32, MIPS64 C and C++ programs.

The compilers produce Common Object File Format (COFF) object files.

In This Section

Supported Device Microprocessors

List the microprocessor families that device compilers support.

Differences Between Desktop and Device Compilers*s*

Provides reference information about compiler options, intrinsic functions, and predefined macros. In addition, describes differences in data alignment and Structured Exception Handling that may be important when programming for smart devices.

ARM Family Processors

Provides information about ARM technology compiler options, intrinsic functions, and call specifications.

Renesas Family Processors

Provides information about Renesas technology compiler options, intrinsic functions, and call specifications.

MIPS Family Processors

Provides information about MIPS technology compiler options, intrinsic functions, and call specifications.

Related Sections

Online Help for Smart Device Projects

Describes Visual Studio 2005 support for smart device programming.

Supported Device Microprocessors

Supported Device Microprocessors

The device compilers support a wide variety of microprocessors used for devices. Support includes compilers, intrinsic functions that allow full optimization, and assemblers for tasks that cannot be supported in other ways.

Supported Microprocessors

The following list shows supported microprocessor families:

- ARM licensed technologies for architectures v4, v4T, Thumb, v5TE, and Intel XScale
  For information about ARM-licensed microprocessors, see ARM Web site.
  For information about ARM-licensed Intel microprocessors, see Intel Web site.
- Renesas SuperH SH4 microprocessors
  For information about Renesas microprocessors, see Renesas Web site.
- MIPS licensed technologies developed by NEC, Toshiba, Philips Semiconductor, Integrated Device Technologies, LSI Logic, and Quantum Effect Design
  - For information about MIPS-licensed technologies, see MIPS Web site
  - For information about MIPS-licensed NEC microprocessors, see NEC Web site
  - For information about MIPS-licensed Toshiba microprocessors, see Toshiba Web site
  - For information about MIPS-licensed Phillips Semiconductor microprocessors, see Philips Semiconductor Web site

See Also

Other Resources

Differences Between Desktop and Device Compilers

Differences Between Desktop and Device Compilers

Differences Between Desktop and Device Compilers

The following topics provide descriptions of the differences between desktop and device compiler operation, including detailed reference information about intrinsic functions and compiler errors.

In This Section

Unique Build Options

Describes build options uniquely defined for device compilers,

Shared Build Options

Lists build options shared with desktop compilers.

Intrinsic Functions for Device Compilers

Provided detailed reference information about the intrinsic functions common to all device compilers.

Pre-defined Macros

Provides reference information about macros included with device compilers.

RISC Processor Data Alignment

Provides programmer information about data alignment issues that vary across different microprocessors.

Exception Handling for Device Compilers

Describes the unique ways RISC-based microprocessor compilers manage exception handling.

Device Compiler Error Messages

Provides information about compiler and linker error messages that you might encounter when working with device compilers.

Related Sections

ARM Family Processors

Provides information about ARM technology compiler options, intrinsic functions, and call specifications.

Renesas Family Processors

Provides information about Renesas technology compiler options, intrinsic functions, and call specifications.

MIPS Family Processors

Provides information about MIPS technology compiler options, intrinsic functions, and call specifications.

Unique Build Options

Unique Build Options

Nearly all of the build options offered for desktop compilers are also available for device compilers, and are implemented in identical ways. Linker option implementation is identical in both environments.

For a list compiler options that are common to both desktop compilers and device compilers, see Shared Build Options.

The following table shows compiler options that support device compilers in a way that differs significantly from desktop implementations.

Option	Description
/callcap - Enable callcap profiling	Inserts callcap profiling hooks at the beginning and end of each function.
/GS - Enable Security Checks	Enables stack checking to detect buffer overrun attacks.

In addition, each supported microprocessor family has processor-specific compiler options that implement unique functionality.

See Also

Reference

ARM Compiler Options
Renesas Compiler Options
MIPS Compiler Options

/callcap - Enable callcap profiling

/callcap - Enable callcap profiling

The **/callcap** option causes the compiler to insert calls to profiling hooks at the beginning and end of each function.

You must compile profiling hooks without the **/callcap** switch. If you compile the profiling hook functions with the **/callcap** switch, the functions will perform infinite recursive calls to themselves.

The following code example, callcaphooks.c, shows a profiling hook function, **_CAP_Enter_Function**, for compilation without callcap.

The **/callcap** switch inserts calls to the profiling hooks for all functions compiled with the **/callcap** compiler switch.

```
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;
}
```

See Also

Concepts

Unique Build Options

/GS - Enable Security Checks

/GS - Enable Security Checks

When you build with the **/GS** build option on, the compiler attempts to detect any direct buffer overruns into the return address. When a buffer overrun overwrites a return address, it provides an opportunity to exploit code that does not enforce buffer size restrictions.

The following sample overruns a buffer. It displays a message box and terminates the process when built with **/GS**.

```
#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);
}
```

Remarks

Buffer overruns are more easily exploited on machines, such as x86, with calling conventions that pass the return address of function calls on the stack.

To prevent buffer overrun exploitation when a function is compiled with **/GS**, the compiler identifies functions that might be subject to buffer overrun problems, and inserts a security cookie on the stack before the associated return address. If, on function exit, the security cookie has changed, then the compiler reports an error and terminates the process.

When working on smart devices, however, the security cookie can become an issue. If an EXE or DLL that does not use one of the default CRT entry points, but instead calls **_cinit** through CRT startup code, the compiler will report an error because **_cinit** resets the expected value of the security cookie. If **_cinit** changes the value of the security cookie, the compiler falsely detects a buffer overrun, reports the error, and terminates the process.

To avoid this issue:

- Do not use arrays or **_alloca** in any functions that call **_cinit**.
- Initialize the CRT normally with a default entry point, such as **WinMainCRTStartup** or **_DllMainCRTStartup**.

/GS does not protect against all buffer overrun security attacks. For example, buffer overrun attacks are still possible by overwriting into the parameters area.

Even if you use **/GS**, you should strive to write secure code. That is, make sure that your code has no buffer overruns. You can inject security checks into compiled code by enforcing buffer size restrictions.

See Also

Other Resources

Differences Between Desktop and Device Compilers

Shared Build Options

Shared Build Options

The following table shows options that are supported by all supported compilers, including those that target desktop workstations and smart devices. For additional documentation on these options, see the Visual C++ reference in Visual Studio .NET combined documentation.

/?	/c (Compile Without Linking)	/C (Preserve Comments During Preprocessing))
/D (Preprocessor Definitions)	/E (Preprocess to stdout)	/EH (Exception Handling Model)
/EP (Preprocess to stdout Without #line Directives)	/F (Set Stack Size)	/FA, /Fa (Listing File)
/Fd (Program Database File Name)	/Fe (Name EXE File)	/FI (Name Forced Include File)
/Fm (Name Mapfile)	/Fo (Object File Name)	/Fp (Name .pch File)
/FR, /Fr (Create .sbr File)	/GF (Eliminate Duplicate Strings))	/GL (Whole Program Optimization)
/GR (Enable Run-Time Type Information)	/GX (Enable Exception Handling)	/Gy (Enable Function-Level Linking)
/GZ (Enable Stack Frame Run-Time Error Checking)	/H (Restrict Length of External Names)	/HELP (Compiler Command-Line Help)
/I (Additional Include Directories)	/J (Default char Type Is unsigned)	/link (Pass Options to Linker)
/MD, /MT, /LD (Use Run-Time Library)	/nologo (Suppress Startup Banner) (C/C++)	/O1, /O2 (Minimize Size, Maximize Speed)
/Ob (Inline Function Expansion)	/Od (Disable (Debug))	/Og (Global Optimizations)
/Oi (Generate Intrinsic Functions)	/Os, /Ot (Favor Small Code, Favor Fast Code)	/Ox (Full Optimization)
/P (Preprocess to a File)	/RTC (Run-Time Error Checks)	/showIncludes (List Include Files)
/Tc, /Tp, /TC, /TP (Specify Source File Type)	/U, /u (Undefine Symbols)	/V (Version Number)
/vd (Disable Construction Displacements)	/vmb, /vmg (Representation Method)	/vmm, /vms, /vmv (General Purpose Representation)
/w, /Wn, /WX, /Wall, /wln, /wdn, /wen, /won (Warning Level)	/WL (Enable One-Line Diagnostics)	/Wp64 (Detect 64-Bit Portability Issues)
/X (Ignore Standard Include Paths)	/Y (Precompiled Headers)	/Yc (Create Precompiled Header File)
/Yd (Place Debug Information in Object File)	/Yu (Use Precompiled Header File)	/YX (Automatic Use of Precompiled Headers)
/Z7, /Zd, /Zi, /ZI (Debug Information Format)	/Za, /Ze (Disable Language Extensions)	/Zc (Conformance)
/Zg (Generate Function Prototypes)	/ZI (Omit Default Library Name)	/Zm (Specify Precompiled Header Memory Allocation Limit)
/Zp (Struct Member Alignment)	/Zs (Syntax Check Only)	..

See Also

Reference

ARM Compiler Options

Renesas Compiler Options

MIPS Compiler Options

Intrinsic Functions for Device Compilers

Intrinsic Functions for Device Compilers

The device compilers support a wide variety of intrinsic functions. Some intrinsic functions are supported by all device compilers. In addition, the compilers for each microprocessor family support additional intrinsic functions.

Common intrinsic functions perform simple and useful operations that are difficult to express concisely in C or C++. They are called common because all device compilers support them.

Using intrinsic functions to access assembly instructions instead of inline assembly results in code that can still be fully optimized by the compiler. All of the intrinsic functions are permanent. Using them in **#pragma** function generates an error message during compilation.

You can enable these functions by including Cmnintrin.h.

In This Section

Common Intrinsic Functions for Device Compilers

Provides reference information about intrinsic functions supported for device compilers.

Intrinsic Forms of CRT Functions

Provides reference information about CRT functions that are supported as intrinsic functions by the device compilers.

Unsupported Intrinsic Functions

Lists the intrinsic functions that are supported by desktop compilers, but not supported by device compilers.

Macros for Common Intrinsics

Provided detailed reference information about macros that assist with processor-specific requirements.

Related Sections

Intrinsic Functions for ARM Microprocessors

Provides reference materials about intrinsic functions specifically defined for the ARM microprocessor compiler.

Intrinsic Functions for MIPS Microprocessors

Provides reference materials about intrinsic functions specifically defined for the MIPS microprocessor compiler.

Intrinsic Functions for Renesas Microprocessors

Provides reference materials about intrinsic functions specifically defined for the SH microprocessor compiler.

Common Intrinsic Functions for Device Compilers

Common Intrinsic Functions for Device Compilers

The following table shows intrinsic functions supported by all device compilers:

__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

See Also

Reference

Unsupported Intrinsic Functions

Intrinsic Forms of CRT Functions

_debugbreak

_debugbreak

This function causes a debug breakpoint exception to be inserted.

```
void __cdecl __debugbreak(void);
```

Parameters
None.

Return Values

None.

Remarks

The breakpoint transfers control to the exception handler.

Requirements

Routine	Required header	Architecture
_debugbreak	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also
Other Resources
Intrinsic Functions for Device Compilers

_prefetch

_prefetch

This function loads the data cache from main memory, if possible.

```
void __cdecl __prefetch(
  void *
);
```

Parameters

*

[in] Pointer to cache line.

Return Value

None.

Remarks

This function might do nothing if the requested functionality is not available on the target hardware platform.

Requirements

Routine	Required header	Architecture
_prefetch	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

__trap

__trap

This function inserts a trap instruction.

```
int __cdecl __trap(
  arg1,
);
```

Parameters

arg1

[in] The trap number.

Return Value

The integer value provided by the trap handler. If the trap handler provides no return value, the return value is undefined.

Remarks

The compiler backend can enforce restrictions on the arguments, including the trap number, and can define a special calling convention for the trap.

The interpretation of the trap number and the actions taken by the trap handler are not defined.

Requirements

Routine	Required header	Architecture
__trap	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_CacheRelease

_CacheRelease

This function writes the cache line containing the address referenced by the pointer to main memory, and then marks the cache line as empty.

```
void __cdecl __CacheRelease(
  void *
);
```

Parameters

*

[in] Pointer to cache line.

Return Value

None.

Remarks

This function might do nothing if the requested functionality is not available on the target hardware platform.

Requirements

Routine	Required header	Architecture
_CacheRelease	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_CacheWriteBack

_CacheWriteBack

This function writes the cache line containing the address referenced by the pointer to main memory.

```
void __cdecl __CacheWriteback(
  void *
);
```

Parameters

*

[in] Pointer to cache line.

Return Value

None.

Remarks

This function might do nothing if the requested functionality is not available on the target hardware platform.

Requirements

Routine	Required header	Architecture
_CacheWriteBack	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_ICacheRefresh

_ICacheRefresh

This function releases the cache line containing the address referenced by the pointer from the instruction cache.

```
void __cdecl __ICacheRefresh(
  void *
);
```

Parameters

*

[in] Pointer to cache line.

Return Value

None.

Remarks

The extent of the instruction cache refreshed is implementation-dependent. It is usually at least 16 bytes, but can be the entire instruction cache. In addition, prefetch of the instructions referenced by the pointer is implementation-dependent.

This function is not supported for SH and MIPS microprocessors.

Requirements

Routine	Required header	Architecture
_ICacheRefresh	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_ReadWriteBarrier, _WriteBarrier

_ReadWriteBarrier, _WriteBarrier

_ReadWriteBarrier forces all previous memory accesses to complete before subsequent memory access is started.

_WriteBarrier forces all previous memory write operations to complete before any subsequent write operation is started.

```
void __cdecl _ReadWriteBarrier(void);
void __cdecl _WriteBarrier(void);
```

Parameters
None.

Return Value

None.

Remarks

_WriteBarrier is usually used for writing device drivers to make sure that a set of commands has been sent to the device before further commands are issued. The compiler does not reschedule memory writes across an invocation of **_WriteBarrier**, even on hardware platforms without explicit synchronization instructions.

_ReadWriteBarrier is usually used for writing device drivers to make sure that commands have been sent to the device before status is read.

The compiler does not reschedule memory reads and writes across an invocation of **_ReadWriteBarrier**, even on hardware platforms without explicit synchronization instructions.

Requirements

Routine	Required header	Architecture
_WriteBarrier	<cmnintrin.h>	x86, ARM, SH-4, MIPS
_ReadWriteBarrier	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also
Other Resources
Intrinsic Functions for Device Compilers

_ReturnAddress

_ReturnAddress

This function provides the return address of the instruction in the calling function that will be executed after control returns to the caller.

```
void _ReturnAddress(void);
```

Parameters
None.

Return Value

None.

Remarks

Build the program in the Example section, and step through it in the debugger.

As you step through the program, note the address that is returned from **_ReturnAddress**.

Then, immediately after returning from the function where **_ReturnAddress** was used, open the Disassembly window and note that the address of the next instruction to be executed matches the address returned from **_ReturnAddress**.

Optimizations such as inlining can affect the return address.

For example, if the following sample program is compiled with **/Ob (Inline Function Expansion)** with n=1, inline_func is inlined into the calling function, main. Therefore, the calls to **_ReturnAddress** from inline_func and main each produce the same value.

Example

```
// 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;
}
```

Requirements

Routine	Required header	Architecture

_ReturnAddress	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_CopyDoubleFromInt64

_CopyDoubleFromInt64

This function copies a floating-point double long integer into a long integer register.

```
double _CopyInt64FromDouble(
  __int64 arg1
);
```

Parameters

arg1

[in] The long integer argument that the function acts on.

Return Values

The floating-point double result of converting *arg1*.

Remarks

This function can be implemented by copying the source value using a temporary memory location.

Requirements

Routine	Required header	Architecture
_CopyDoubleFromInt64	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_CopyFloatFromInt32

_CopyFloatFromInt32

This function copies an integer value to a floating-point register.

```
float _CopyFloatFromInt32(
  ___int32 arg1
);
```

Parameters

arg1

[in] The integer value acted upon by the function.

Return Values

The floating point conversion of *arg1*.

Remarks

This function can be implemented by copying the source value using a temporary memory location.

Requirements

Routine	Required header	Architecture
_CopyFloatFromInt32	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_CopyInt32FromFloat

_CopyInt32FromFloat

This function copies a floating-point number into an integer register.

```
__int32 _CopyInt32FromFloat(
  float arg1
);
```

Parameters

arg1

[in] The floating-point value acted upon by the function.

Return Values

The integer conversion of *arg1*.

Remarks

This function can be implemented by copying the source value using a temporary memory location.

Requirements

Routine	Required header	Architecture
_CopyInt32FromFloat	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_CopyInt64FromDouble

_CopyInt64FromDouble

This function copies a floating-point double to a long integer register.

```
__int64 _CopyInt64FromDouble(
  double arg1
);
```

Parameters

arg1

[in] The floating-point double argument acted on by the function.

Return Values

The long integer result of the conversion.

Remarks

This function can be implemented by copying the source value using a temporary memory location.

Requirements

Routine	Required header	Architecture
_CopyInt64FromDouble	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_CountLeadingOnes, _CountLeadingOnes64

_CountLeadingOnes, _CountLeadingOnes64

This function returns the number of contiguous one bits starting with the most significant bit.

```
unsigned __cdecl _CountLeadingOnes(
  long arg1
);

unsigned __cdecl _CountLeadingOnes64(
  __int64 arg1
);
```

Parameters

arg1

[in] The long integer argument that the function examines to find one bits.

Return Values

The number of contiguous one bits in *arg1*. If all bits in *arg1* are set, the return value of **_CountLeadingOnes** is 32, or the value of **_CountLeadingOnes64** is 64..

Remarks

This function can be implemented by calling a runtime function.

Requirements

Routine	Required header	Architecture
_CountLeadingOnes	<cmnintrin.h>	x86, ARM, SH-4, MIPS
_CountLeadingOnes64	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_CountLeadingSigns, _CountLeadingSigns64

_CountLeadingSigns, _CountLeadingSigns64

This function returns the number of contiguous bits that match the sign bit, starting with the next most significant bit.

```
unsigned __cdecl _CountLeadingSigns(
  long arg1
);

unsigned __cdecl _CountLeadingSigns64(
  __int64 arg1
);
```

Parameters

arg1

[in] The unsigned integer value that the function operates on.

Return Values

The number of contiguous bits in *arg1* that match the sign bit.

Remarks

This function can be implemented by calling a runtime function.

Requirements

Routine	Required header	Architecture
_CountLeadingSigns	<cmnintrin.h>	x86, ARM, SH-4, MIPS
_CountLeadingSigns64	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_CountLeadingZeros, _CountLeadingZeros64

_CountLeadingZeros, _CountLeadingZeros64

This function returns the number of contiguous zero bits starting with the most significant bit in the argument.

```
unsigned __cdecl _CountLeadingZeros(
  long arg1
);

unsigned __cdecl _CountLeadingZeros64(
  __int64 arg1
);
```

Parameters

arg1

[in] The unsigned integer to be examined by the function.

Return Values

The number of contiguous zero bits in *arg1*. If none of the bits in *arg1* are set, the return value is of **_CountLeadingZeros** is 32, or the value of **_CountLeadingZeros64** is 64.

Remarks

This function can be implemented by calling a runtime function.

Requirements

Routine	Required header	Architecture
_CountLeadingZeros	<cmnintrin.h>	x86, ARM, SH-4, MIPS
_CountLeadingZeros	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_CountOneBits, _CountOneBits64

_CountOneBits, _CountOneBits64

This function returns the number of one bits in the argument.

```
unsigned __cdecl _CountOneBits(
  long arg1
);

unsigned __cdecl _CountOneBits64(
  __int64 arg1
);
```

Parameters

arg1

[in] The long integer value that the function examines for one bits.

Return Values

The number of one bits.

Remarks

This function can be implemented by calling a runtime function.

Requirements

Routine	Required header	Architecture
_CountOneBits	<cmnintrin.h>	x86, ARM, SH-4, MIPS
_CountOneBits64	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_isunordered, _isunorderedf

_isunordered, _isunorderedf

_isunordered compares two double precision numbers to determine if they are unordered.

_isunorderedf compares two floating-point numbers to determine if they are unordered.

```
int __cdecl _isunordered(
  double arg1,
  double arg2
);

int __cdecl _isunorderedf(
  float arg1,
  float arg2
);
```

Parameters

arg1

[in] The value to be compared to *arg2*.

arg2

[in] The value to be compared to *arg1*.

Return Values

Returns a Boolean value.

TRUE indicates that *arg1* and *arg2* are unordered.

Remarks

IEEE-754 floating-point comparison can have four separate result values: less-than, equal-to, greater-than or unordered.

The first three conditions can be tested using normal C operators, and this function is used to test for the last condition.

Two values are unordered if either is a NaN. This means that a NaN is not equal to any value, even itself.

The C++ compiler returns a *bool* value instead of an *int*.

Requirements

Routine	Required header	Architecture
_isunordered	<cmnintrin.h>	x86, ARM, SH-4, MIPS
_isunorderedf	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

MulDiv

MulDiv

The **MulDiv** function multiplies two 32-bit values and then divides the 64-bit result by a third 32-bit value. The return value is rounded up or down to the nearest integer.

```
int MulDiv(
  int nNumber,
  int nNumerator,
  int nDenominator
);
```

Parameters

nNumber

[in] Multiplicand.

nNumerator

[in] Multiplier.

nDenominator

[in] Number by which the result of the multiplication (nNumber * nNumerator) is to be divided.

Return Values

If the function succeeds, the return value is the result of the multiplication and division. If either an overflow occurred or nDenominator was 0, the return value is â€”1.

Requirements

Routine	Required header	Architecture
MulDiv	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

_MulHigh, _MulUnsignedHigh

_MulHigh, _MulUnsignedHigh

This function returns the high-order 32-bit result of multiplying two arguments.

```
long __cdecl _MulHigh(
  long arg1,
  long arg2
);

unsigned long __cdecl _MulUnsignedHigh(
  unsigned long arg1,
  unsigned long arg2
);
```

Parameters

arg1

[in] The first argument in the product.

arg2

[in] The second argument in the product.

Return Values

The long integer result of multiplying *arg1* and *arg2*.

Remarks

This function can be useful for detecting overflow. **_MulHigh** is useful for multiplying integers scaled to represent [-0.5..0.5), and **_MulUnsignedHigh** is useful for multiplying integers scaled to represent 0..1).

Requirements

Routine	Required header	Architecture
_MulHigh	<cmnintrin.h>	x86, ARM, SH-4, MIPS
_MulUnsignedHigh	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

__assume

__assume

The __**assume** intrinsic function passes a hint to the optimizer.

```
__assume(expression)
```

Parameters
expression

Condition to be tested.

Return Value

None.

Remarks

The optimizer assumes that the condition represented by expression is true at the point where the keyword appears and remains true until expression is altered (for example, by assignment to a variable). Selective use of hints passed to the optimizer by __**assume** can improve optimization.

Example

```
//
// 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.
  }
}
```

Requirements

Routine	Required header	Architecture
__**assume**	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

__noop

__noop

__noop gives you a function name to use when you want the function to be ignored and the argument list unevaluated.

```
__noop(functionname)
```

Parameters

Functionname

Name of function to be ignored

Return Value

None.

Example

```
   // 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");
}
```

Requirements

Routine	Required header	Architecture
__noop	<cmnintrin.h>	x86, ARM, SH-4, MIPS

See Also

Other Resources

Intrinsic Functions for Device Compilers

Intrinsic Forms of CRT Functions

Intrinsic Forms of CRT Functions

The following table lists intrinsic forms of CRT functions that are supported by device compilers:

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

Intrinsic Forms of Math Library Functions

The following table lists intrinsic forms of math library functions that are supported by device compilers:

acos	asin	atan	atan
ceil, ceilf	cos, cosf, cosh, coshf	fmod, fmodf	exp, log, and log
floor, floorf	pow	sin, sinf, sinh, sinhf	sqrt
tan, tanf, tanh, tanhf	.	.	.

See Also

Reference

Unsupported Intrinsic Functions

Common Intrinsic Functions for Device Compilers

Unsupported Intrinsic Functions

Unsupported Intrinsic Functions

Visual Studio desktop compilers for x86, AMD64, or Intel Itanium (IPF) architectures support the functions in the following list. Device compilers do not support these intrinsic functions.

__break	__cpuid
__dsrlz	__faststorefence
__fc	__fci
__fclrf	__flushrs
__fsetc	__fwb
__getCFS	__getcx86, AMD64, IA64, IPFerseflags
__getPSP	__getReg
__inbyte	__inbytestring
__indword	__indwordstring
__int2c	__invalat
__invlpg	__Inword, __inwordstring
__isNat	__isrlz
__lfetch, __lfetch_excl, __lfetchfault, __lfetchfaultexcl	__load128, __load128_acq
__movsb, __movsd,__movsq, __movsw	__mul128
__mulh	__outbyte
__outbytestring	__outword
__outwordstring	__ptcg, __ptcga
__ptcl	__ptrcl, __ptri
__rdteb	__rdtsc
__readcr0, __readcr2, __readcr3, __readcr4, __readcr8	__readfsbyte, __readfsdword, __readfsqword, __readfsword
__readgsbyte, __readgsdword, __readgsqword, __readgsword	__readmsr
__readpmc	__rsm
__rum	__segmentlimit

__setReg	__shiftleft128
__shiftright128	__ssm
__store128, __store128_rel	__stosb, __stosd, __stosq, __stosw
__sum	__synci
__writefsbyte, __writefsdword, __writefsqword, __writefsword	__writegsbyte, __writegsdword, __writegsqword, __writegsword
__writemsr	__yield
_AcquireSpinLock	_AddressOfReturnAddress
_BitScanForward, _BitScanForward64	_BitScanReverse, _BitScanReverse64
_bittest, _bittest64	_bittestandcomplement,_bittestandcomplement64
_bittestandreset, _bittestandreset64	_bittestandset, _bittestandset64
_InterlockedAdd	_InterlockedAddLargeStatistic
_InterlockedAnd	_InterlockedAnd, _InterlockedAnd64
_interlockedbittestandreset, _interlockedbittestandreset64	_InterlockedCompare64Exchange
_InterlockedCompareExchange16	_InterlockedCompareExchange64
_InterlockedDecrement16	_InterlockedExchangePointer
_InterlockedIncrement16	_InterlockedOr
_InterlockedXor	_ReadBarrier
_ReleaseSpinLock	_thash
_ttag	_umul128
_umulh	_wbinvd
_writecr0, _writecr2, _writecr3, _writecr4, _writecr8,	..

The following table lists intrinsic forms of math library functions that are not supported by device compilers:

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

See Also

Other Resources

Intrinsic Functions for Device Compilers

Macros for Common Intrinsics

Macros for Common Intrinsics

All microprocessor families that support common intrinsic functions have a machine-dependent description of support in the header file, cmnintr.h. The machine-dependent description does the following:

- Declares the functions with the correct prototype
- Enables the intrinsic version
- Defines macros that classify how the function is supported

You can use the intrinsic function unconditionally or you can use one of the macros to classify the intrinsic support in your CPU environment.

The following table shows descriptions of the macros for common intrinsic functions.

Macro	Description
_INTRINSIC_IS_HELPER	This macro determines if an intrinsic function is instantiated through a call to the C Run-time Library (CRT).
_INTRINSIC_IS_INLINE	This macro tests to determine if the compiler can expand a specified intrinsic to one or more lines.
_INTRINSIC_IS_SAFE	This macro determines if a specified intrinsic function is instantiated independent of the OS.
_INTRINSIC_IS_SUPPORTED	This macro determines if a specified intrinsic function is supported.

See Also

Reference

Common Intrinsic Functions for Device Compilers

_INTRINSIC_IS_HELPER

_INTRINSIC_IS_HELPER

This macro determines if a specified intrinsic function is supported.

```
_INTRINSIC_IS_SUPPORTED(arg)
```

Parameters

Arg

The name of the intrinsic function of interest.

Return Value

A nonzero return value indicates that the specified intrinsic is supported by the compiler.

See Also

Other Resources

Intrinsic Functions for Device Compilers

_INTRINSIC_IS_INLINE

_INTRINSIC_IS_INLINE

This macro determines if a specified intrinsic function is supported.

```
_INTRINSIC_IS_SUPPORTED(arg)
```

Parameters

Arg

[in] The name of the intrinsic function of interest.

Return Value

A nonzero return value indicates that the specified intrinsic is supported by the compiler.

Example

```
   //  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
```

See Also

Other Resources

Intrinsic Functions for Device Compilers

_INTRINSIC_IS_SAFE

_INTRINSIC_IS_SAFE

This macro determines if a specified intrinsic function is instantiated independent of the OS.

```
_INTRINSIC_IS_SAFE(arg)
```

Parameters

Arg

[in] The name of the intrinsic function of interest.

Return Value

A nonzero return value indicates that the specified intrinsic does require invocation of an OS feature.

Remarks

An intrinsic function expansion sometimes requires the compiler to invoke an operating system feature because the compiler itself cannot perform the particular task. Because the compiler cannot control whether the OS provides the required feature, such an intrinsic is considered unsafe.

For example, the __trap intrinsic assumes that an OS handler is available to take an action. The compiler cannot guarantee that such a handler is present, so the **__trap** intrinsic is considered unsafe.

See Also

Other Resources

Intrinsic Functions for Device Compilers

_INTRINSIC_IS_SUPPORTED

_INTRINSIC_IS_SUPPORTED

This macro determines if an intrinsic function is instantiated through a call to the Microsoft C Run-Time Library (CRT).

```
_INTRINSIC_IS_HELPER(arg)
```

Parameters

Arg

[in] The name of the intrinsic function of interest.

Return Value

A nonzero return value indicates that intrinsic named by *arg* is instantiated with a call to the CRT.

Example

```
    //
    // 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
  //
```

See Also

Reference

Common Intrinsic Functions for Device Compilers

Pre-defined Macros

Pre-defined Macros

The device compilers recognize seven predefined ANSI C macros and the Microsoft C++ implementation provides several more.

These macros take no arguments and cannot be redefined. Their value, except for **__LINE__** and **__FILE__**, must be constant throughout compilation.

The following table provides additional information about predefined macros, some of which are defined with multiple values.

Macro	Description
__DATE__	The compilation date of the current source file. The date is a string literal of the form Mmm dd yyyy.
__FILE__	The name of the current source file. **__FILE__** expands to a string surrounded by double quotation marks.
__FUNCTION__	Valid only within a function and returns the undecorated name of the enclosing function (as a string). **__FUNCTION__** is not expanded if you use **/EP (Preprocess to stdout Without #line Directives)** or **/P (Preprocess to a File)** compiler options.
__LINE__	The line number in the current source file. The line number is a decimal integer constant. It can be altered with a **#line** directive.
__STDC__	Indicates full conformance with the ANSI C standard. Defined as the integer constant 1 only if the **/Za, /Ze (Disable Language Extensions)** compiler option is given and you are not compiling C++ code; otherwise is undefined.
__TIME__	The most recent compilation time of the current source file. The time is a string literal of the form hh:mm:ss.
__TIMESTAMP__	The date and time of the last modification of the current source file, expressed as a string literal in the form Ddd Mmm Date hh:mm:ss yyyy, where Ddd is the abbreviated day of the week and Date is an integer from 1 to 31.

Microsoft-specific Macros

The following table shows additional predefined macros that are Microsoft-specific.

_CHAR_UNSIGNED

Default char type is unsigned. Defined when **/J (Default char Type Is unsigned)** is specified.

__cplusplus

Defined for C++ programs only.

_CPPRTTI

Defined for code compiled with **/GR (Enable Run-Time Type Information)**.

_CPPUNWIND

Defined for code compiled with **/GX (Enable Exception Handling)**.

_MFC_VER

Defines the MFC version. Defined as 0x0600 for Microsoft Foundation Class Library 6.0 or later. Always defined.

_MSC_EXTENSIONS

This macro is defined when compiling with the **/Za, /Ze (Disable Language Extensions)** compiler option (the default). Its value, when defined, is 1.

_MSC_VER

Defines the compiler version. Defined as 1200 for Microsoft Visual C++ 6.0 or later. Always defined.

_WIN32

Defined for applications for Win32. Always defined.

See Also

Other Resources

Differences Between Desktop and Device Compilers

RISC Processor Data Alignment

RISC Processor Data Alignment

Alignment of data is an important issue when porting or writing code on some target architectures, especially architectures other than x86.

Depending on the architecture, unaligned operations with primitive data types can cause poor application performance, or can cause your application to fault and terminate abnormally.

This section provides information to help you avoid such pitfalls.

In This Section

About Data Alignment

Describes how data alignment issues impact device programming.

Avoiding Alignment Errors

Provides guidelines for keeping primitive types properly aligned.

Working with Packing Structures

Describes how structure packing interacts with alignment.

__unaligned keyword

Provides reference information about using the __unaligned modifier to deal with alignment issues.

About Data Alignment

About Data Alignment

Many CPUs, such as those based on Alpha, IA-64, MIPS, and SuperH architectures, refuse to read misaligned data. When a program requests that one of these CPUs access data that is not aligned, the CPU enters an exception state and notifies the software that it cannot continue. On ARM, MIPS, and SH device platforms, for example, the operating system default is to give the application an exception notification when a misaligned access is requested.

Misaligned memory accesses can incur enormous performance losses on targets that do not support them in hardware.

Alignment

Alignment is a property of a memory address, expressed as the numeric address modulo a power of 2. For example, the address 0x0001103F modulo 4 is 3; that address is said to be aligned to 4n+3, where 4 indicates the chosen power of 2. The alignment of an address depends on the chosen power of two. The same address modulo 8 is 7.

An address is said to be aligned to X if its alignment is *Xn+0*.

CPUs execute instructions that operate on data stored in memory, and the data are identified by their addresses in memory. In addition to its address, a single datum also has a size. A datum is called naturally aligned if its address is aligned to its size, and misaligned otherwise. For example, an 8-byte floating-point datum is naturally aligned if the address used to identify it is aligned to 8.

Compiler handling of data alignment

Device compilers attempt to allocate data in a way that prevents data misalignment.

For simple data types, the compiler assigns addresses that are multiples of the size in bytes of the data type. Thus, the compiler assigns addresses to variables of type **long** that are multiples of four, setting the bottom two bits of the address to zero.

In addition, the compiler pads structures in a way that naturally aligns each element of the structure. Consider the structure **struct x_** in the following code example:

```
struct x_
{
   char a;     // 1 byte
   int b;      // 4 bytes
   short c;    // 2 bytes
   char d;     // 1 byte
} MyStruct;
```

The compiler pads this structure to enforce alignment naturally.

Example

The following code example shows how the compiler places the padded structure in memory:

```
// 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
}
```

Both declarations return **sizeof(struct x_)** as 12 bytes.

The second declaration includes two padding elements:

- **char _pad0[3]** to align the **int** b member on a four-byte boundary
- **char _pad1[1]** to align the array elements of the structure **struct _x bar[3]**;

The padding aligns the elements of **bar[3]** in a way that allows natural access.

The following code example shows the **bar[3]** array layout:

```
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];
```

See Also

Reference

__unaligned keyword

Concepts

Working with Packing Structures

Avoiding Alignment Errors

Avoiding Alignment Errors

All data types are either primitive types, or complex types that consist of arrays, structures, or unions of simpler types.

Alignment of primitive types

In general, alignment errors can be avoided by following these rules:

- Do not enable structure packing.
- Do not access a small-aligned address by using a recast pointer of larger alignment.

For example, treating the address of a **char** data type as a pointer to a **long** can cause an alignment error because this might mean executing a four-byte move from an address that is not a multiple of four.

The following example illustrates this style of coding:

```
char a[10];
char *p = &a[1];
long l = *(long *)p;   // ERROR!; Attempt to move a long from
                       // the address of a char.
```

Accessing a large-aligned address

Accessing a large-aligned address with a recast pointer of smaller alignment is safe. For example, you could use a **char *** cast to access the first byte, or any byte, of a long variable.

If you need to pack structures or move data to unaligned addresses, use the __unaligned keyword.

This keyword cannot resolve alignment problems of pre-existing classes such as in inherited code, or in the Microsoft Foundation Class Library.

You might need to use structure packing in programs that use large arrays of structures, or to read a pre-existing data format.

In such cases, you can still use packed structures if you also carefully unpack members of a packed structure before using data in the program. This technique might involve copying the data in a structure member-by-member, or element-by-element, or field-by-field, into a temporary location that is correctly aligned.

Syntactically, **__unaligned** is a type qualifier like **const** and **volatile**; it controls the meaning of what the pointer points to. **__unaligned** has meaning only when used in a pointer declaration.

The following code example shows the use of **__unaligned** pointer declarations:

```
int __unaligned *p1;             // p1 is a pointer to unaligned int
struct {int i;} __unaligned *p2; // p2 is a pointer to unaligned struct
```

The following code example illustrates the correct and incorrect use of **__unaligned** and **#pragma pack** in conjunction with integer operations. In the first section, a fault is generated because the **__unaligned** qualifier is not used, whereas in the second section, the **__unaligned** qualifier is used correctly.

```
#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);
}
```

The output from this example appears in the following machine code example. In the output, function **f_improper** shows the code generated by the improper handling of unaligned data, and function **g_proper** shows the extra code generated when **__unaligned** is used.

In function **g_proper**, more than double the number of instructions are generated to handle the unaligned data, but an alignment fault cannot occur.

```
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
```

Because there is a performance penalty for accessing data through an **__unaligned** pointer, use the **__unaligned** keyword only when needed.

To guarantee that there are no alignment errors, the compiler must access the de-referenced data as a series of smaller pieces.

If the data is already aligned, this technique for accessing data is not necessary.

Alignment of Complex Types

The following list summarizes the rules for alignment of complex types:

- The alignment of an array is the same as the alignment of the base type.
- The alignment of a structure or object is the maximum of the alignment factors of all members of the structure. Further, as long as structure packing is turned off, the compiler pads the structure so each member is placed at the next available properly aligned address for that member.

The alignment of a union is also the maximum of the alignment factors of all members. Each member's address is that of the union itself, so there is no padding.

See Also

Reference

__unaligned keyword

Concepts

Working with Packing Structures

Working with Packing Structures

Working with Packing Structures

Problems can occur when a structure requires more bytes than the programmer intended, especially when space requirements are paramount.

Structure Packing and Alignment

Structure packing interacts with compiler alignment behavior as follows.

- If the packsize is set equal to or greater than the default alignment, the packsize is ignored.
- If the packsize is set smaller than the default alignment, the compiler aligns according to the packsize value.

Thus, if the packsize is set to four, data types having a size of four, eight, or 16 bytes are aligned on addresses that are multiples of four. However, there is no guarantee that data types eight bytes in size (64 bits) are aligned on addresses that are a multiple of eight. The packsize has no effect on data types outside of a structure.

In addition, packing affects the alignment of the entire packed structure. For example, in a structure declared under **#pragma pack(1)**, the alignment of all members are forced to one, regardless of whether they would have been naturally aligned even without packing.

The following techniques set a packsize, in bytes:

- The command-line option **/Zp (Struct Member Alignment)** sets the packsize to *n*, in which *n* can be 1, 2, 4, 8, or 16, and in which 8 is the default.
- The compiler directive **#pragma pack([n])** sets the packsize to n, in which n can be 1, 2, 4, 8, or 16. If n is not specified, **#pragma pack** resets the packsize to its value at the beginning of compilation: either the value specified by **/Zp (Struct Member Alignment)**, or the default, which is 8 on most platforms.
  The pragma applies only from the point at which it occurs in the source. For example, the **/Zp1** option sets the packsize to 1, which causes the compiler to use no padding within structures. To avoid this problem, turn off structure packing or use the __unaligned keyword when accessing unaligned members of such structures through pointers.

Guidelines for packing structures

The following list shows possible solutions to the structure issue.

- Reordering structure members
  If space requirements are critical, reorder the members of the structure so the same-sized elements are next to each other and pack tightly. Usually, you start with the largest members and work your way down to the smallest ones.
  Note that reordering a structure assumes that the user has full control of the data structure, but the user might not have the freedom to rearrange the members. For example, the data structure might represent the layout of fields in a file on disk.
  Consider the following code example:

```
struct x_
{
   char a;     // 1 byte
   int b;      // 4 bytes
   short c;    // 2 bytes
   char d;     // 1 byte
} MyStruct;
```

  If you reorganize the members of this structure, as shown in the following code example, the reorganized structure aligns all members on natural boundaries, and the size of the structure is eight bytes instead of 12.

```
struct x_
{
   int b;     // 4 bytes
   short c;   // 2 bytes
   char d;    // 1 byte
   char a;    // 1 byte
```

```
} MyStruct;
```

- Padding the structure
  A different kind of problem can arise when the structure size requires padding to make sure array elements have the same alignment, but the user needs to ensure that there is no padding between the array elements. For example, the user might need to restrict memory usage or read data from a fixed-format source.
  If the structure requires padding, you can use the compiler directive **#pragma pack**. However, **#pragma pack** causes elements of the structures to be unaligned, and it requires the use of the __unaligned keyword qualifier to generate the code needed to access this data without causing alignment faults.
  The following example code uses **#pragma pack** to tell the compiler that the pointer px points to data that is not naturally aligned, and tells the compiler to generate the appropriate sequence of load, merge, and store operations to do the assignment efficiently.

```
# 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;
}
```

  The __unaligned keyword should only be used as a last resort, because the generated code is less efficient than accessing naturally aligned data. However, the __unaligned keyword is clearly preferable to alignment faults.
  If at all possible, arrange the members of a data structure to preserve alignment and minimize space at the same time.

See Also

Reference

__unaligned keyword

__unaligned keyword

__unaligned keyword

The **__unaligned** keyword is a type modifier in pointer definitions. It indicates to the compiler that the data pointed to might not be properly aligned on a correct address.

To be properly aligned, the address of an object must be a multiple of the size of the type. For example, two-byte objects must be aligned on even addresses.

When data is accessed through a pointer declared **__unaligned**, the compiler generates the additional code necessary to load or store, or read or write the data without causing alignment errors.

It is best to avoid using unaligned data, but in some cases the usage can be justified by the need to access packed structures such as shared disk structures or I/O hardware.

Note: **UNALIGNED** is a Win32 macro that expands to **__unaligned** on hardware platforms that require it. **UNALIGNED** expands to nothing on platforms that do not require **__unaligned**.

For portability, use **UNALIGNED** instead of the **__unaligned** keyword. When you use the UNALIGNED macro, include the windef.h header, as in the following example:

```
#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
```

See Also

Concepts

Working with Packing Structures

Exception Handling for Device Compilers

Exception Handling for Device Compilers

Exception handling works differently on RISC-based microprocessors (such ARM, SH-4, and MIPS) than on x86 microprocessors. Under many circumstances, the compiler hides the differences; however, the differences become critical when your code contains assembly language segments.

In simplified terms, in an x86 environment, the data structures associated with exception handling are written onto the stack at runtime. The OS looks at these structures to locate appropriate exception-handling routines.

In a RISC environment, many data structures associated with exception handling are calculated at compile time and are written to the data sections of the module being built. RISC-based microprocessors support a table-based mechanism using PDATA Structures to handle exception processing that depends on the context, the frame and stack pointers, and the program counter. Because the compiler generates the code segments that set up the stack frame whose address limits are associated with the function table entry, only code that the compiler handles can access the table entry automatically.

Therefore, in a RISC environment, you must write code that manages exception-related processing for any assembly code functions you include.

In This Section

SEH in x86 Environments

Provides a brief description of how exception handling works in an x86 processor environment.

SEH in RISC Environments

Provides a brief description of how exception handling works in a RISC processor environment.

PDATA Structures

Provides reference information about data structures used in RISC exception handling.

Related Sections

ARM Prolog and Epilog

Provides guidelines and examples for creating prolog and epilog code sequences for ARM microprocessor compilers.

Renesas SH-4 Prolog and Epilog

Provides guidelines and examples for creating prolog and epilog code sequences for Renesas microprocessor compilers.

MIPS Prolog and Epilog

Provides guidelines and examples for creating prolog and epilog code sequences for MIPS microprocessor compilers.

SEH in x86 Environments

SEH in x86 Environments

In simplified terms, data structures associated with exception handling in an x86 environment are written onto the stack at runtime. The OS looks at these structures to locate approriate exception-handling routines.

Exception-handling structures

In an x86 environment, the **FS** register points to the current value of the Thread Information Block (TIB) structure. One element in the TIB structure is a pointer to an **EXCEPTION_RECORD** structure, which in turn contains a pointer to an exception handling callback function. Thus, each thread has its own exception callback function.

The x86 compiler builds exception-handling structures on the stack as it processes functions. The **FS** register always points to the TIB, which in turn contains a pointer to an **EXCEPTION_RECORD** structure. The **EXCEPTION_RECORD** structure points to the exception handler function.

EXCEPTION_RECORD structures form a linked list: the new **EXCEPTION_RECORD** structure contains a pointer to the previous **EXCEPTION_RECORD** structure, and so on. On Intel-based machines, the head of the list is always pointed to by the first DWORD in the thread information block, **FS:[0]** .

Unwinding

When an exception occurs, the system walks the list of **EXCEPTION_RECORD** structures until it finds a handler for the exception. When a handler is found, the system walks the list again, up to the node that handles the exception. During this second traversal, the system calls each handler function a second time with the exception flag set to EH_UNWINDING.

The API builds a dummy exception record structure containing the context and the exception callback function. After each callback, the corresponding exception frame is removed and the API moves on to the next frame. The API stops unwinding when it gets to the frame whose address was passed in as the first parameter.

After an exception is handled and all previous exception frames have been called to unwind, execution continues where the handling callback function indicates.

The code where execution resumes expects that the stack and frame pointers, the **ESP** and **EBP** registers on Intel CPUs, are set to their values within the stack frame that handled the exception.

Therefore, the handler that accepts an exception is responsible for setting the stack and frame pointers to values they had in the stack frame that contains the SEH code that handled the exception.

See Also

Concepts

SEH in RISC Environments

SEH in RISC Environments

SEH in RISC Environments

In a RISC environment, data structures associated with exception processing are calculated at compile time, and written to the data sections of the module being built.

To locate appropriate handlers when an exception occurs in Win32 environments other than x86, the system first determines the frames that reside on the callstack, along with their associated functions in code. Any function can have a handler associated with it. If so, the system gives the handler associated with the function an opportunity to handle the exception.

As with x86, a RISC system invokes handlers in reverse order; that is, it first invokes the handler whose corresponding frames were most recently pushed onto the stack.

To determine the frames on the stack, the system simulates the execution of a portion of each function's code in reverse. This simulation creates a CPU context similar to the state the real CPU context held at the point of entry to that function.

This process of reverse execution is known as Virtual Unwinding, because the stack unwind is only being simulated, not actually performed.

Code elements for unwinding

The portion of the code that is reversed is known as the Prolog of the function. It consists of instructions that modify the stack pointer and set up the stack frame immediately upon entry to the function.

When a frame has been virtually unwound, the virtual context contains the stack pointer for the previous frame and the return address for the current function. The return address is very near the place where control left the previous function, so it corresponds to the program counter of the previous frame.

With each successive program counter and stack pointer, the unwind process can iterate until there are no frames left on the stack.

To virtually unwind, the system needs a small amount of information about each function. This information is contained in data structures called PDATA Structures.

A PDATA structure marks where a function begins and ends in the code stream, as well as the location of the function prolog.

Given a program counter associated with a specific stack frame, the Unwinder searches the table of PDATA for the entry corresponding to the containing function. When found, the Unwinder can unwind the function frame.

The PDATA structure also locates an exception handling routine associated with the function, if one exists.

The compiler generates correct Prolog and Epilog sequences, and PDATA for functions that it compiles, but you must create appropriate code and PDATA for functions you write in assembly language.

The prolog and epilog sequence must adhere to strict guidelines for Virtual Unwinding to work.

For details on acceptable prologs and epilogs, see the documentation for your target platform.

See Also

Reference

Prolog-Epilog Example

Concepts

Prolog

Virtual Unwinding

Other Resources

Prolog and Epilog

Prolog

Prolog

A prolog has several immediately contiguous parts, with no intervening instructions.

Typically, a prolog segment contains separate sequences of instructions that perform the following tasks:

- Allocate a stack frame.
- Save incoming argument registers.
- Set up the frame pointer, if one is to be established. The prolog copies the stack pointer to a designated register before the initial register saves; then it uses this value to compute the value of the frame pointer.
- Save the link register with return address.
- Allocate space for compiler-generated temporaries, local variables, and an argument build area.
- Indicate the end of the prolog code.

Be sure your prolog code is succinct and only performs the necessary operations.

See Also

Reference

Prolog-Epilog Example

Concepts

SEH in RISC Environments

Epilog

Epilog

Although each procedure has only one prolog, a procedure can contain many epilogs if the procedure uses multiple exit points.

Each epilog is required to have certain specific parts. All parts are contiguous, with no intervening instructions.

Typically, an epilog segment contains separate sequences of instructions that perform the following tasks:

- Restore the frame pointer register, if it was saved in the prolog
- Restore nonvolatile registers, including the Program Counter and the stack
- Restore the return address
- Deallocate the local frame
- Return to the calling function

See Also

Reference

Prolog-Epilog Example

Concepts

SEH in RISC Environments

Prolog-Epilog Example

Prolog-Epilog Example

The following code example allocates a stack frame that requires 64 KB of memory for an SH-4 microprocessor. The code segment for an ARM or MIPS microprocessor is similar, except for the names of registers used.

The prolog segment in this example saves the argument register to the incoming argument save areas. No separate frame pointer is required. The segment then saves the return address, saves the permanent register, and allocates the stack frame. It declares an exception handler.

The epilog segment removes the stack frame and recovers the return address.

```
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
```

Virtual Unwinding

Virtual Unwinding

To reconstruct the context that existed on entry to a routine, SEH for RISC processors uses a process called Virtual Unwinding to emulate a small subset of instructions in prolog and epilog code.

Virtual unwinding provides a syntactically efficient way of transferring control from the kernel exception handler to user-mode code.

In virtual unwinding, the kernel traverses the call stack to find an appropriate exception handler.

Starting with a CPU context record and an instruction address, the unwinding process interprets instructions in the prolog or epilog to reconstruct the context, as it existed before the function call.

Code elements for unwinding

The **Virtual Unwinder** uses a PDATA Structures to determine the procedure start, the procedure end, and the prolog end. The **PDATA** structure can also contain a pointer to an exception handler.

The subset of prolog and epilog code that the **Virtual Unwinder** emulates includes the following:

- Adding or subtracting a value from a register
- Loading or storing a register on the stack frame
- Loading integer constants into registers
- Moving between registers

The **Virtual Unwinder** ignores other instructions found in the prolog or epilog sequences.

Virtual Unwinder process

The following list shows the steps the **Virtual Unwinder** performs:

1. Search the prolog for an instruction that saves the frame pointer, the stack pointer, or the link register.
   If the instruction is present, the instruction saves all permanent registers the **Virtual Unwinder** must restore.
   If the instruction is not present, the link register contains the return address, and the **Virtual Unwinder** updates only the program counter.
2. Search for an instruction in the prolog that writes the frame pointer. The unwinding process restores all registers from this address down, starting from the lowest numbered register to the highest numbered register.
3. Search for an instruction that writes the stack. If such an instruction exists, the unwinding process must reverse-execute the stack link. The right operand to this subtract is the stack size, which is a constant immediate value.
4. If execution stops inside a prolog, the **Virtual Unwinder** determines if an instruction that saves the permanent registers executed, and if a stack link executed.

   - If the function has not saved the permanent registers, the **Virtual Unwinder** copies the value in the link register to the program counter register.
   - If the function saved the register values, and if no stack link executed, the **Virtual Unwinder** updates the permanent registers from the stack pointer.
   - If execution stopped in a prolog with a linked stack, the **Virtual Unwinder** reverse-executes the prolog.

Note:
All functions that move the stack pointer must have an associated PDATA structure for SEH to work. These include any function that allocates stack space, calls other functions, saves permanent registers, or has an exception handler. A leaf function (that is, a function that calls no other functions) that does not modify a permanent register does not need PDATA. In this case, the **Virtual Unwinder** updates the program counter from the link register and continues to the next frame.

See Also

Reference

Prolog-Epilog Example
PDATA Structures

Concepts

SEH in RISC Environments

PDATA Structures

PDATA Structures

ARM, MIPS, and SHx device compilers use PDATA structures to aid in stack walking at run-time. This structure aids in debugging and exception processing.

The compilers associate one PDATA structure with each procedure.

The data structure is a table stored in a COFF .pdata section. The .pdata section contains an array of function table entries for exception handling, and is pointed to by the exception table entry in the image data directory.

The MIPS calling standard supports an uncompressed PDATA format, _IMAGE_ALPHA_RUNTIME_FUNCTION_ENTRY. In most cases, MIPSII currently uses an uncompressed 20 bytes for each function for the _IMAGE_ALPHA_RUNTIME_FUNCTION_ENTRY entry.

Leaf functions that do not have associated exception-handling routines do not have an associated pdata entry.

The following table shows the function table entry format for MIPSII images.

Offset	Size	Field	Description
0	4	Begin Address	Virtual address of the corresponding function.
4	4	End Address	Virtual address of the end of the function.
8	4	Exception Handler	Pointer to the exception handler to be executed.
12	4	Handler Data	Pointer to additional information to be passed to the handler.
16	4	Prolog End Address	Virtual address of the end of the function prolog.

ARM and SH-4 Device compilers support a compressed PDATA structure, _IMAGE_CE_RUNTIME_FUNCTION_ENTRY.

The following table shows the COFF-specified function-table entry format used for the ARM and SH-4 hardware platforms.

Offset	Size	Field	Description
0	4	Begin Address	Virtual address of the corresponding function.
4	8 bits	Prolog Length	Number of instructions in the function's prolog.
4	22 bits	Function Length	Number of instructions in the function.
4	1 bit	32-bit Flag	Set if the function is comprised of 32-bit instructions, cleared for a 16-bit function.
4	1 bit	Exception Flag	Set if an exception handler exists for the function.

If an exception handler exists or the function length is zero, an additional PDATA_EH structure precedes the function in the .text section. The function uses PDATA_EH when the function has an associated exception handler or handler data.

In most cases, **PDATA** structure occupies only eight bytes per function. For functions that have an exception handler, the PDATA_EHstructure requires an additional eight bytes.

The exception-handling data record and the prolog and function length record are guaranteed to be 4-byte aligned. This implies that any function associated with one or more of these records is 4-byte aligned.

See Also

Concepts

Virtual Unwinding

_IMAGE_CE_RUNTIME_FUNCTION_ENTRY

_IMAGE_CE_RUNTIME_FUNCTION_ENTRY

This structure contains detailed information about runtime exception processing.

_IMAGE_CE_RUNTIME_FUNCTION_ENTRY is used only by ARM and Renesas microprocessor families. It does not apply to MIPS microprocessors.

```
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;
```

Parameters

FuncStart

Address of the first instruction in the function. It is the function's entry address.

PrologLen

Length of the prolog in instructions, based on the instruction size indicated in the *ThirtyTwoBit* flag. *PrologLen* is set to 1 for ARM functions, and to 0 for THUMB and SH-4 functions.

FuncLen

Total function length in instructions; see *PrologLen*, above. A function with 200 ARM instructions would have a *FuncLen* of 200, or 800 bytes.

ThirtyTwoBit

Size of instructions in a function. *ThirtyTwoBit* can hold one of the following values: 1Represents ARM functions, each of which consists of 32-bit instructions, or 4 bytes. 0Represents THUMB functions, each of which consists of 16-bit instructions, or 2 bytes.

ExceptionFlag

Associated exception handler or handler data. When its value is 1, the PDATA_EH is present in the **.text** section. When the *ExceptionFlag* is 0, no PDATA_EH is present.

Remarks

The **IMAGE_CE_RUNTIME_FUNCTION_ENTRY** data structure is also called as **PDATA**. A table containing these records is stored in a section called .pdata. The .pdata section aids in debugging and exception processing.

If the *ExceptionFlag* is set, or if the *FuncLen* is set to 0, an additional PDATA_EH structure exists that precedes the function in the **.text** section.

The data record containing this information appears in the **.text** section, immediately preceding the function, if and only if the *ExceptionFlag* bit is set.

This record is used when the function has an associated exception handler or handler data.

See Also

Reference

PDATA Structures

PDATA_EH

_IMAGE_ALPHA_RUNTIME_FUNCTION_ENTRY

_IMAGE_ALPHA_RUNTIME_FUNCTION_ENTRY

This structure contains detailed information about runtime exception processing.

This structure has an uncompressed 20-byte format.

```
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;
```

Parameters

BeginAddress

Address of the first instruction in the function. It is the function's entry address.

EndAddress

Address of the last instruction in the function. It is the function's end address.

ExceptionHandler

Address of the exception handler for the function.

HandlerData

Address of the exception handler data record for the function.

PrologEndAddress

Address of the last instruction in the prolog.

See Also

Reference

PDATA Structures

PDATA_EH

PDATA_EH

This structure holds detailed information about an associated exception handler function. This is an internal data structure used in OS exception processing.

```
struct PDATA_EH {  unsigned int* pHandler;  unsigned int* pHandlerData;};
```

Parameters

pHandler

Address of the exception handler for the function.

pHandlerData

Address of the exception handler data record for the function.

See Also

Reference

PDATA Structures

_IMAGE_CE_RUNTIME_FUNCTION_ENTRY

Device Compiler Error Messagesr

Device Compiler Error Messagesr

The following table describes unique error messages for device compilers.

Error	Description
Compiler Error C2729	Indicates an intrinsic function not supported in Thumb mode.
Compiler Error C2759	Indicates an error in inline assembly.
Compiler Error C2822	Indicates premature exit from guarded section.
Compiler Error C2880	Indicates an attempt to create a namespace alias failed because the namespace already exis ts.
Compiler Error C2887	Indicates too many arguments for a __**swi** intrinsic.
Compiler Warning (level 2) C4720	Indicates one of a variety of SH-specific issues.
Compiler Warning (level 1) C4721	Indicates an unknown intrinsic function.
Compiler Warning (level 1) C4732	Indicates an intrinsic function not supported in the target architecture.
Compiler Warning (Level 1) C4567	Indicates that the linker encountered incompatible object files from different versions of th e compiler.

See Also

Other Resources

Compilers for Smart Devices

Compiler Error C2729

```
intrinsic not allowed in Thumb mode
```

This error indicates that the compiler attempted to compile an intrinsic function not supported in Thumb mode, such as _prefetch.

Note that if source is compiled with the /GL (Whole Program Optimization) switch, this error will not be output until the referenced object is linked.

Compiler Error C2759

```
in-line assembler reports: "various"
```

An error occurred in inline assembly code that prevented compilation.

Compiler Error C2822

```
local unwind is not supported on this platform
```

The implementation of Structured Exception Handling on this platform does not support a local unwind operation, which is required when prematurely leaving either the guarded section or the termination handler of a **try-finally** statement. If you need to leave the guarded section, use the **__leave** keyword. Leaving the termination handler prematurely can have undefined behavior and should be avoided.

The following code demonstrates two ways this error message can be generated.

```
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;
}
```

Compiler Error C2880

```
__swi requires a valid constant as first argument (SWI number)
```

The _swi intrinsic function did not receive an integer constant as expected for the first argument. The integer constant must be in the range [0 - 16777215] for ARM, or [0 - 255] for Thumb microprocessors.

Example

```
int test_intrinsic(int x)
{

     return __swi(x, 12, 14, 13, 12); // error C2880
}
```

Compiler Error C2887

```
__swi cannot have more than five arguments (SWI number r0 - r3)
```

__swi intrinsic must have five or fewer arguments.

Example

```
#pragma intrinsic(__swi)
int test_intrinsic()
{
     return __swi(10, 12, 14, 13, 15, 12);
 // error cannot have more than 5 args
}
```

Compiler Warning (Level 2) C4720

```
in-line assembler reports: 'message'
```

This SH-specific warning applies to multiple situations that might occur for SH inline assembly.

This warning is not visible when compiling with the -GL Whole Program Optimization option.

In the following example code, the compiler issues this warning to indicate a branch in a delay slot.

Example

```
/* 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;
}
```

Compiler Warning (Level 1) C4721

```
'function' : not available as an intrinsic
```

The compiler encountered an unknown intrinsic function. The use of #pragma intrinsic('function') will be ignored.

Compiler Warning (Level 1) C4732

```
instrinsic ' %s' is not supported in this architecture
```

The compiler encountered an intrinsic function that is not supported in the target architecture.

The __trap intrinsic is not supported under the MIPS 16 ISA. As a result, the following code example causes the compiler to generate compiler warning C4732.

Example

```
#include <cmnintrin.h>

int main()
{
  int returnCode = 1;
      #if (_INTRINSIC_IS_SUPPORTED(__trap))
              __trap(1);
      #else
              return 1;
      #endif
  return 0;
}
```

Compiler Warning (Level 1) C4567

Compiler Warning (Level 1) C4567

```
'function' : behavior change due to parameter 'parameter': calling convention incompatible
with previous compiler versions
```

This warning indicates that the compiler encountered code that may not execute correctly if it is linked with code compiled by an earlier compiler version.

This warning is specific to the C++ compiler for the ARM(R) Architecture. Versions of the compiler older than 14.00 use a different calling convention than newer compilers when passing certain function parameters by value. The two calling conventions are not compatible, and linking object files from an older compiler with newer object files can result in unpredictable behavior and crashes if such a parameter is passed by value between the old and new object files.

An object of class, struct, or union type with a user-defined copy constructor is subject to this calling convention change if it is passed by value. Objects passed by reference are not affected.

If you are linking to object files from older compilers, use this warning to find places in your code where the calling convention has changed. If objects with user-defined copy constructors are passed by value between old and new object files, the old object files must be recompiled with a compiler version 14.00 or later.

This warning is off by default.

Example

```
// 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;
}
```

See Also

Other Resources

Differences Between Desktop and Device Compilers

ARM Family Processors

ARM Family Processors

The ARM microprocessor range provides solutions for the following applications:

- Open hardware platforms running complex operating systems with wireless, consumer, and imaging applications.
- Embedded, real-time systems for mass storage, automotive, industrial, and networking applications.
- Secure applications, including smart cards and Single Inline Memory (SIM) modules.

The ARM Instruction Set Architecture (ISA) includes several technology extensions, such as THUMB technology, that enable optimum functionality and performance.

In This Section

Intrinsic Functions for ARM Microprocessors

Provides tables and detailed reference information about intrinsic functions supported by key ARM microprocessor families.

ARM Compiler Options

Provides reference information about compiler options specific to ARM microprocessors and compilers.

ARM Calling Sequence Specification

Provides information about ARM register and stack frame layout, ARM prologs and epilogs for SEH, and reference information about the ARM assembler.

Related Sections

Intrinsic Functions for Device Compilers

Provides reference information about intrinsic functions supported by all device compilers.

RISC Processor Data Alignment

Provides guidelines for data alignment for RISC microprocessors.

SEH in RISC Environments

Describes the key differences between Structured Exception Handling in RISC environments.

Intrinsic Functions for ARM Microprocessors

Intrinsic Functions for ARM Microprocessors

The ARM device compiler supports a set of intrinsic functions that are defined for specific ARM microprocessors.

Different sets of intrinsic functions are available for the ARM10, the ARM DSP, the ARM XSCALE, and the Intel PXA270 architectures.

The ARM compiler uses the **/QR** compiler option to determine which set of intrinsic functions to chose for a particular compilation. For example, the **/QRxscale** enables the XScale MAC intrinsic functions. For more information about the **/QR** flag, see ARM Compiler Options.

If appropriate target architecture is not defined, some intrinsic functions, such as XScale MAC functions, will fail.

To guard an intrinsic function, or to ensure that an intrinsic function is called only if a specific target architecture is defined, use the system management function **IsProcessorFeaturePresent.**

In This Section

ARM10 Intrinsic Functions

ARM10 Intrinsic Functions

The following ARM10 instructions are supported through intrinsic functions.

Instruction	Description
CLZ	Counts leading zeroes before first 1-bit. The common intrinsic _CountLeadingZeros, _CountLeadingZeros64accesses the **CLZ** instruction.
BKPT	Creates soft breakpoint. The common intrinsic __trap accesses the BKPT instruction.
_swi	Generates a call to the OS using the SWI software interrupt instruction.
__emit	Inserts a specified instruction into the instruction stream.
_MoveFromCoProcessor, _MoveFromCoProcessor2	Reads data from the ARM coprocessor.
_MoveToCoProcessor, MoveToCoprocessor2	Writes data to the ARM coprocessor.

See Also

Reference

ARM Compiler Options

CLZ

CLZ

This ARM10 instruction counts the number of binary zero bits before the first binary one bit in a register value. The common _CountLeadingZeros, _CountLeadingZeros64intrinsic supports **CLZ**.

```
unsigned cdecl _CountLeadingZeros(
  long Arg1
);
```

Parameters

Arg1

[in] The value for which the leading zero bits should be determined.

Return Values

Number of binary zero bits.

Remarks

To generate the **CLZ** instruction for the **_CountLeadingZeros** intrinsic, use the **-QRarch5** or **-QRarch5T** flag.

If you are compiling on an ARM4 microprocessor, the compiler generates a call to a library name, or to some other sequence of ARM 4 code.

Requirements

Routine	Required header	Architecture
CLZ	<armintr.h>	ARM

See Also

Reference

ARM10 Intrinsic Functions

/QRArch - Specify Target Architecture

Other Resources

Common Intrinsic Function Reference

BKPT

BKPT

This ARM10 instruction causes a software breakpoint to occur. The common __trap intrinsic supports **BKPT**.

```
int __trap(
  int Arg1
);
```

Parameters

Arg1

[in] Address of breakpointed instruction.

Return Values

None.

Requirements

Routine	Required header	Architecture
BKPT	<armintr.h>	ARM

See Also

Reference

ARM10 Intrinsic Functions

Other Resources

Common Intrinsic Function Reference

__emit

__emit

This intrinsic function inserts a specified instruction into the stream of instructions output by the compiler.

```
void __emit(
   const unsigned __int32 opcode
);
```

Parameters

opcode

Instruction word to be inserted.

Return Values

None

Remarks

The value of *opcode* must be a constant expression known at compile time.

The compiler makes no attempt to interpret the contents of *opcode* and does not guarantee a CPU or memory state before the inserted instruction is executed.

The compiler assumes that the CPU and memory states are unchanged after the inserted instruction is executed. Therefore, instructions that do change state can have a detrimental impact on normal code generated by the compiler.

For this reason, use **__emit** only to insert instructions that affect a CPU state that the compiler does not normally process, such as coprocessor state, or to implement functions declared with **__declspec(naked).**

When generating ARM instructions, the size of an instruction word is 32 bits. When generating Thumb instructions, as when /QRthumb is specified, the size of an instruction word is 16 bits and the most significant 16 bits of *opcode* are ignored.

Requirements

Routine	Required header	Architecture
__emit	<armintr.h>	ARM

See Also

Reference

ARM10 Intrinsic Functions

/QRthumb

_MoveFromCoProcessor, _MoveFromCoProcessor2

_MoveFromCoProcessor, _MoveFromCoProcessor2

These intrinsic functions read data from ARM coprocessors via the coprocessor data transfer instructions.

```
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
);
```

Parameters

coproc

Coprocessor number in the range 0 to 15.

opcode1

Coprocessor-specific opcode in the range 0 to 7.

crn

Coprocessor register number in the range 0 to 15, which specifies the first operand to the instruction.

crm

Coprocessor register number in the range 0 to 15, which specifies an additional source or destination operand.

opcode2

Additional coprocessor-specific opcode in the range 0 to 7.

Return Values

The value read from the coprocessor.

Remarks

The values of all five parameters to this intrinsic must be constant expressions known at compile time.

_MoveFromCoprocessor uses the MRC instruction; **_MoveFromCoprocessor2** uses MRC2. The parameters correspond to bitfields encoded directly into the instruction word. The interpretation of the parameters is coprocessor-dependent. For more information, see the manual for the coprocessor in question.

These intrinsics are not available when generating Thumb instructions, such as when /QRthumb is specified.

Requirements

Routine	Required header	Architecture
_MoveFromCoprocessor	<armintr.h>	ARM

Routine	Required header	Architecture
_MoveFromCoprocessor2	<armintr.h>	ARM

See Also
Reference
ARM10 Intrinsic Functions
/QRthumb

_MoveToCoProcessor, MoveToCoprocessor2

_MoveToCoProcessor, MoveToCoprocessor2

These intrinsic functions write data to ARM coprocessors via the coprocessor data transfer instructions.

```
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
);
```

Parameters

value

Value to be written to the coprocessor.

coproc

Coprocessor number in the range 0 to 15.

opcode1

Coprocessor-specific opcode in the range 0 to 7.

crn

Coprocessor register number in the range 0 to 15, which specifies the first operand to the instruction.

crm

Coprocessor register number in the range 0 to 15, which specifies and additional source or destination operand.

opcode2

Additional coprocessor-specific opcode in the range 0 to 7.

Return Values

None

Remarks

The values of *coproc*, *opcode1*, *crn*, *crm*, and *opcode2* must be constant expressions known at compile time.

_MoveToCoprocessor uses the **MCR** instruction; **_MoveToCoprocessor2** uses **MCR2**.

The parameters correspond to bitfields encoded directly into the instruction word. The interpretation of the parameters is coprocessor-dependent. For more information, see the manual for the coprocessor in question.

These intrinsics are not available when generating Thumb instructions, such as when /QRthumb is specified.

Requirements

Routine	Required header	Architecture

_MoveToCoprocessor , **_MoveToCoprocessor2**	<armintr.h>	ARM

See Also

Reference

ARM10 Intrinsic Functions

/QRthumb

_swi

_swi

This intrinsic function generates a call to an OS routine using the software interrupt instruction **SWI**.

```
unsigned int __swi(
   unsigned swi_number,
   arg2,
   arg3,
   arg4
);
```

Parameters

swi_number

Software Interrupt number

arg2-arg4

Additional arguments for passing

Return Values

The **_swi** intrinsic function returns the value left in register R0 when control is returned to the instruction following the SWI.

Remarks

The **_swi** intrinsic applies to the ARM or the Thumb instruction set, depending on whether the compiler is generating 32-bit or 16-bit code.

The first parameter, *swi_number,* is encoded directly into the immediate field of the instruction. It must be an integer constant in the range [0 - 16777215] for ARM or [0 - 255] for Thumb.

If additional arguments are included, the function passes the values according to the standard ARM calling convention with one exception: no arguments or parts of arguments may be passed in memory, that is, on the stack.

Therefore, all arguments must be able to be passed using only registers **R0, R1, R2,** and **R3**.

Requirements

Routine	Required header	Architecture
_swi	<armintr.h>	ARM

See Also

Reference

/QRArch - Specify Target Architecture

/QRthumb

ARM DSP-enhanced Intrinsic Functions

ARM DSP-enhanced Intrinsic Functions

All ARM DSP instructions are supported as intrinsic functions. To use the ARM DSP intrinsics, include the armintr.h header.

The following ARM DSP Instructions have intrinsic functions defined for them.

Function	Corresponding ARM DSP instruction	Description
_SmulAddLo_SW_SL _SmulAddHi_SW_SL _SmulAddHiLo_SW_SL _SmulAddLoHi_SW_SL	**SMLAxy**	A signed-integer multiply and accumulate operation: 16x16-bit multiply followed by a 32-bit add.
_SmulAddWLo_SW_SL _SmulAddWHi_SW_SL	**SMLAWy**	A 32x16-bit multiply operation, followed by a 32-bit add of the upper 32 bits of the 48 bit product.
_SmulAddHi_SW_SQ _SmulAddLo_SW_SQ _SmulAddHiLo_SW_SQ _SmulAddLoHi_SW_SQ	**SMLALxy**	A 16x16-bit multiply operation, followed by a 64-bit add of the product, with a 64-bit integer.
_SmulLo_SW_SL _SmulHi_SW_SL _SmulHiLo_SW_SL _SmulLoHi_SW_SL	**SMULxy**	A signed-integer 16x16-bit multiply operation.
_SmulWLo_SW_SL _SmulWHi_SW_SL	**SMULWy**	A signed-integer 32x16-bit multiply operation, returning the upper 32-bits.
_AddSatInt	**QADD**	A saturating add instruction.
_DSubSatInt	**QSUB**	A saturating subtract instruction.
_DAddSatInt	**QDADD**	An instruction to double an integer and saturate, and then add to a second integer and saturate.
_DSubSatInt	**QDSUB**	An instruction to double an integer and saturate, and then subtract from a second integer and saturate.
_ReadCoProcessor _WriteCoProcessor	**MRRC,** **MCRR**	An operation to transfer values from a coprocessor to two ARM registers.

See Also

Reference

ARM XSCALE Intrinsic Functions

Other Resources

Intrinsic Functions for Device Compilers

_SmulAddLo_SW_SL

_SmulAddLo_SW_SL

This ARM DSP-enhanced, signed-integer multiply-accumulate operation multiplies the bottom half of register **Rm** and the bottom half of register **Rs**, producing a 32-bit product. The operation then performs a 32-bit accumulation with **Rn**.

```
int _SmulAddLo_SW_SL(
  int Arg1,
  int Arg2,
  int Arg3
);
```

Parameters

Arg1

Contents of **Rn**, the value added to the product of *Arg2* and *Arg3*.

Arg2

[in] The contents of **Rm**, the first term multiplied.

Arg3

[in] The contents of **Rs**, the second term multiplied.

Return Values

The result of multiplication and accumulation.

Remarks

The compiler translates this instruction into the **smlabb** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddLo_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulAddHi_SW_SL
_SmulAddHiLo_SW_SL
_SmulAddLoHi_SW_SL

_SmulAddHi_SW_SL

_SmulAddHi_SW_SL

This ARM DSP-enhanced, signed-integer multiply-accumulate operation multiplies the top half of register **Rm** and the top half of register **Rs**, producing a 32-bit product. The operation then performs a 32-bit accumulation with **Rn**.

```
int _SmulAddHi_SW_SL(
  int Arg1,
  int Arg2,
  int Arg3
);
```

Parameters

Arg1

The contents of **Rn**, the value added to the product of *Arg2* and *Arg3*.

Arg2

[in] The contents of **Rm**, the first term multiplied.

Arg3

[in] The contents of **Rs**, the second term multiplied.

Return Values

The result of the multiplication and accumulation.

Remarks

The compiler translates this instruction into the **smlatt** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddHi_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulAddLo_SW_SL
_SmulAddHiLo_SW_SL
_SmulAddLoHi_SW_SL

_SmulAddHiLo_SW_SL

_SmulAddHiLo_SW_SL

This ARM DSP-enhanced, signed-integer multiply-accumulate operation multiplies the top half of register **Rm** and the bottom half of register **Rs** to produce a 32-bit product. The operation then performs a 32-bit accumulation with **Rn**.

```
int _SmulAddHiLo_SW_SL(
  int Arg1,
  int Arg2,
  int Arg3
);
```

Parameters

Arg1

The contents of **Rn**, the value added to the product of *Arg2* and *Arg3*.

Arg2

[in] The contents of **Rm**, the first term multiplied.

Arg3

[in] The contents of **Rs**, the second term multiplied.

Return Values

The result of multiplication and accumulation.

Remarks

The compiler translates this instruction into the **smlatb** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddHiLo_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulAddHi_SW_SL
_SmulAddLo_SW_SL
_SmulAddLoHi_SW_SL

_SmulAddLoHi_SW_SL

_SmulAddLoHi_SW_SL

This ARM DSP-enhanced, signed-integer multiply-accumulate operation multiplies the bottom half of register **Rm** and the top half of register **Rs**, producing a 32-bit product. The operation then performs a 32-bit accumulation with **Rn**.

```
int _SmulAddLoHi_SW_SL(
  int Arg1,
  int Arg2,
  int Arg3
);
```

Parameters

Arg1

The contents of **Rn**, the value added to the product of *Arg2* and *Arg3*.

Arg2

[in] The contents of **Rm**, the first term multiplied.

Arg3

[in] The contents of **Rs**, the second term multiplied.

Return Values

The integer result of multiplication.

Remarks

The compiler translates this instruction into the **smlabt** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddLoHi_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulAddLo_SW_SL
_SmulAddHiLo_SW_SL
_SmulAddHi_SW_SL

_SmulAddWLo_SW_SL

_SmulAddWLo_SW_SL

This ARM DSP-enhanced, signed integer multiply-accumulate operation multiplies **Rm** with the bottom 16 bits of **Rs**; then it accumulates in **Rn**. The operation adds the upper 32 bits of the 48-bit product to the 32-bit **Rn**.

```
int _SmulAddWLo_SW_SL(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] The contents of **Rm**, the first term in the product.

Arg2

[in] The contents of **Rs**, the second term in the product.

Return Values

The integer result of the multiplication and accumulation.

Remarks

The compiler translates this instruction into the **smlawb** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddW_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions

_SmulAddWHi_SW_SL

_SmulAddWHi_SW_SL

_SmulAddWHi_SW_SL

This ARM DSP-enhanced, signed integer multiply-accumulate operation multiplies **Rm** with the top 16 bits of **Rs** then accumulates in **Rn**. The operation adds the upper 32 bits of the 48-bit product to the 32-bit **Rn**.

```
int _SmulAddWHi_SW_SL(
  int Arg1,
  int Arg2,
  int Arg3
);
```

Parameters

Arg1

[in] The contents of **Rn**, the value added to the product of *Arg2* and *Arg3*.

Arg2

[in] The contents of **Rm**, the first term in the product.

Arg3

[in] The contents of **Rs**, the second term in the product.

Return Values

The integer result of the multiplication and accumulation.

Remarks

The compiler translates this instruction into the **smlawt** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddWHi_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions

_SmulAddWLo_SW_SL

_SmulAddHi_SW_SQ

_SmulAddHi_SW_SQ

This ARM DSP-enhanced, signed integer multiply-accumulate operation first performs a multiply on two 16-bit source operands from the top half of register **Rm** and the top half of **Rs**. This is followed with a 64 bit accumulate with the 32-bit registers RdLo and RdHi.

```
__int64 _SmulAddHi_SW_SQ(
  __int64 Arg1,
  int Arg2,
  int Arg3
);
```

Parameters

Arg1

Pointers to a 64-bit accumulate that contains RdHi and RdLo.

Arg2

[in] The contents of **Rm**, the first term in the product.

Arg3

[in] The contents of **Rs**, the second term in the product.

Return Values

The long integer result of the multiplication and accumulation.

Remarks

The compiler translates this instruction into the **smlaltt** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddHi_SW_SQ	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulAddLo_SW_SQ
_SmulAddHiLo_SW_SQ
_SmulAddLoHi_SW_SQ

_SmulAddLo_SW_SQ

_SmulAddLo_SW_SQ

This ARM DSP-enhanced, signed integer multiply-accumulate operation first performs a multiply on two 16-bit source operands from the bottom half of register **Rm** and the bottom half of **Rs**. This is followed with a 64 bit accumulate with the 32-bit registers RdLo and RdHi.

```
__int64 _SmulAddLo_SW_SQ(
  __int64 Arg1,
  int Arg2,
  int Arg3
);
```

Parameters

Arg1

A pointer to a 64-bit variable used to accumulate the contents of RdHi and RdLo.

Arg2

[in] The contents of **Rm**, the first term in the product.

Arg3

[in] The contents of **Rs**, the second term in the product.

Return Values

The result of multiplication and accumulation.

Remarks

The compiler translates this instruction into the **smlalbb** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddLo_SW_SQ	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulAddHi_SW_SQ
_SmulAddHiLo_SW_SQ
_SmulAddLoHi_SW_SQ

_SmulAddHiLo_SW_SQ

_SmulAddHiLo_SW_SQ

This ARM DSP-enhanced, signed integer multiply-accumulate operation multiplies the top half of register **Rm** and the bottom half of **Rs**. This is followed with a 64 bit accumulate with the 32-bit registers RdLo and RdHi.

```
__int64 _SmulAddHiLo_SW_SQ(
  __int64 Arg1,
  int Arg2,
  int Arg3
);
```

Parameters

Arg1

Pointer to a 64-bit variable used to accumulate the contents of RdHi and RdLo.

Arg2

[in] The contents of **Rm**, the first term in the product.

Arg3

[in] The contents of **Rs**, the second term in the product.

Return Values

The long integer result of multiplication and accumulation.

Remarks

The compiler translates this instruction into the **smlaltb** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddHiLo_SW_SQ	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulAddLo_SW_SQ
_SmulAddLoHi_SW_SQ
_SmulAddHi_SW_SQ

_SmulAddLoHi_SW_SQ

_SmulAddLoHi_SW_SQ

This ARM DSP-enhanced, signed integer multiply-accumulate operation multiplies the bottom half of register **Rm** and the top half of **Rs**. This is followed with a 64 bit accumulate with the 32-bit registers RdLo and RdHi.

```
__int64 _SmulAddLoHi_SW_SQ(
  __int64 Arg1,
  int Arg2,
  int Arg3
);
```

Parameters

Arg1

A pointer to a 64-bit variable used to accumulate the contents of RdHi and RdLo.

Arg2

[in] The contents of **Rm**, the first term in the product.

Arg3

[in] The contents of **Rs**, the second term in the product.

Return Values

The long integer result of the multiplication and accumulation.

Remarks

The compiler translates this instruction into the **smlalbt** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddLoHi_SW_SQ	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulAddLo_SW_SQ
_SmulAddHiLo_SW_SQ
_SmulAddHi_SW_SQ

_SmulHi_SW_SL

_SmulHi_SW_SL

This ARM DSP-enhanced, signed integer multiply operation multiplies the top half of register **Rm** times the top half of register **Rs**, producing a 32-bit result in **Rd**.

```
int _SmulHi_SW_SL(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] The contents of **Rm**, the first term in the product.

Arg2

[in] The contents of **Rs**, the second term in the product.

Return Values

The integer result of the multiplication.

Remarks

The compiler translates this instruction into the **smultt** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulHi_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulLo_SW_SL
_SmulLoHi_SW_SL
_SmulHiLo_SW_SL

_SmulLo_SW_SL

_SmulLo_SW_SL

This ARM DSP-enhanced, signed integer multiply operation multiplies the bottom half of register **Rm** times the bottom half of register **Rs**, producing a 32-bit result in **Rd**.

```
int _SmulLo_SW_SL(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] The contents of **Rm**, the first term in the product.

Arg2

[in] The contents of **Rs**, the second term in the product.

Return Values

The integer result of the multiplication.

Remarks

The compiler translates this instruction into the **smulbb** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulLo_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulLoHi_SW_SL
_SmulHiLo_SW_SL
_SmulHi_SW_SL

_SmulHiLo_SW_SL

_SmulHiLo_SW_SL

This ARM DSP-enhanced, signed integer multiply operation multiplies the top half of register **Rm** times the bottom half of register **Rs**, producing a 32-bit result in **Rd**.

```
int _SmulHiLo_SW_SL(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] The contents of **Rm**, the first term in the product.

Arg2

[in] The contents of **Rs**, the second term in the product.

Return Values

The integer result of the multiplication.

Remarks

The compiler translates this instruction into the **smultb** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulHiLo_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulLo_SW_SL
_SmulLoHi_SW_SL
_SmulHi_SW_SL

_SmulLoHi_SW_SL

_SmulLoHi_SW_SL

This ARM DSP-enhanced, signed integer multiply operation multiplies the bottom half of register **Rm** times the top half of register **Rs**, producing a 32-bit result in **Rd**.

```
int _SmulLoHi_SW_SL(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] The contents of **Rm**, the first term in the product.

Arg2

[in] The contents of **Rs**, the second term in the product.

Return Values

The integer result of the multiplication.

Remarks

The compiler translates this instruction into the **smulbt** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulLoHi_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
_SmulLo_SW_SL
_SmulHiLo_SW_SL
_SmulHi_SW_SL

_SmulWHi_SW_SL

_SmulWHi_SW_SL

This ARM DSP-enhanced, signed-integer multiplication operation performs a 32x16 bit multiply on the 32-bit operand in **Rm** and the 16-bit source operand from the top half of register **Rs**. It then takes the upper 32 bits of the 48-bit product.

```
int _SmulWHi_SW_SL(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] The first term in the product, the contents of **Rm**.

Arg2

[in] The second term in the product | the contents of **Rs**.

Return Values

The integer result of the multiplication.

Remarks

The compiler translates this instruction into the **smulwt** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulWHi_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions

_SmulWLo_SW_SL

_SmulWLo_SW_SL

_SmulWLo_SW_SL

This ARM DSP-enhanced, signed-integer multiplication operation performs a 32x16 bit multiply on the 32-bit operand in **Rm** and the 16-bit source operand from the bottom half of register **Rs**. It then takes the upper 32 bits of the 48-bit product.

```
int _SmulWLo_SW_SL(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] The contents of **Rm**, the first term in the product.

Arg2

[in] The contents of **Rs**, the second term in the product.

Return Values

The integer result of the multiplication and accumulation.

Remarks

The compiler translates this instruction into the **smlawb** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulWLo_SW_SL	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions

_SmulAddWHi_SW_SL

_AddSatInt

_AddSatInt

This ARM DSP-enhanced operation performs a saturating add instruction. It adds registers **Rm** and **Rn**, and places the result in register **Rd**. It affects the sticky-overflow bit 'Q' if overflow occurs in the addition.

```
int _AddSatInt(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] The contents of **Rm**, the first term in the sum.

Arg2

[in] The contents of **Rn**, the second term in the sum.

Return Values

The result of the binary arithmetic.

Remarks

The compiler translates this instruction into the **qadd** assembly instruction.

Requirements

Routine	Required header	Architecture
_AddSatInt	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions

_SubSatInt

_DAddSatInt

_DAddSatInt

This operation calculates SAT(**Rm** + SAT(**Rn***2)); that is, the operation first saturates the double of **Rn**, adds the result to **Rm**, then saturates the sum.

```
int _DAddSatInt(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] This integer is added to the doubled *Arg2*. It is analogous to the value in **Rm**.

Arg2

[in] This integer is the term that is doubled. It is analogous to the value in **Rn**.

Return Values

The result of the binary arithmetic.

Remarks

The compiler translates this instruction into the **qdadd** assembly instruction.

Saturation can occur on the doubling operation, on the addition, or both. If saturation occurs on the doubling operation, but not on the addition, the Q flag is set and the the final result is unsaturated.

Requirements

Routine	Required header	Architecture
_DAddSatInt	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions

_DSubSatInt

_SubSatInt

_SubSatInt

This ARM DSP-enhanced operation performs a saturating subtract instruction. It subtracts the value in **Rn** from the value in **Rm**, and places the result in register **Rd**. This operation affects the sticky-overflow bit 'Q' if overflow occurs in the subtraction.

```
int _SubSatInt(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] The first term in the difference, the contents of **Rm**.

Arg2

[in] The second term in the difference, the contents of **Rn**.

Return Values

The result of the binary arithmetic.

Remarks

The compiler translates this intrinsic into the **qsub** assembly instruction.

Requirements

Routine	Required header	Architecture
_SubSatInt	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions

_AddSatInt

_DSubSatInt

_DSubSatInt

This operation doubles **Rn** and saturates, then subtracts the result from **Rm** and saturates. This operation affects the sticky-overflow bit 'Q' if overflow occurs in the subtraction.

```
int _DSubSatInt(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] The doubled *Arg2* is subtracted from this integer. It is the contents of the value in **Rm**.

Arg2

[in] This integer is doubled. It is the contents of the value in **Rn**.

Return Values

The value that results from the arithmetic performed.

Remarks

The compiler translates this instruction into the **qdsub** assembly instruction.

Requirements

Routine	Required header	Architecture
_DSubSatInt	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions

_DAddSatInt

_ReadCoProcessor

_ReadCoProcessor

This instruction causes the specified coprocessor registers to transfer values to two ARM registers.

```
__int64 _ReadCoProcessor(
  int Arg1
);
```

Parameters

Arg1

[in] Coprocessor number, equivalent to cp_num.

Return Values

Value held in coprocessor register.

Remarks

The compiler translates this instruction into the **MRRC** assembly instruction for ARM DSP-enhanced processors, and into the **MRA** assembly instruction for ARM XScale processors. **MRA** is disassembled as the **MRCC** instruction.

The XScale and the DSP-enhanced ARM microprocessors each implement this instruction in a different way:

- For the ARM XScale implementation, this instruction does the following:
  - Moves 64 bits of data to ARM registers from Coprocessor registers
  - Moves the 40-bit accumulator value (**acc0**) into two registers
  - Moves bits [31:0] of the value in acc0 into the register RdLo
  - Sign-extends bits [39:32] of the value in acc0 to 32 bits and moves them into the register RdHi
- For the ARM DSP-enhanced implementation, this instruction causes the coprocessor to transfer values to the two general-purpose registers **Rd** and **Rn**.

Requirements

Routine	Required header	Architecture
_ReadCoProcessor	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM DSP-enhanced Intrinsic Functions

ARM XSCALE Intrinsic Functions

_WriteCoProcessor

_WriteCoProcessor

_WriteCoProcessor

This instruction causes the specified ARM registers to transfer values to coprocessor registers.

```
void _WriteCoProcessor(
  __int64 Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] Values to be written to coprocessor.

Arg2

[in] Coprocessor number. This should be zero.

Return Values

None.

Remarks

For the ARM XScale processor, the compiler translates this intrinsic into the **MAR** assembly instruction. **MAR** is disassembled as the **MCRR** instruction.

For the ARM DSP-enhanced processor, the compiler translates this intrinsic into the **MCRR** assembly instruction.

The XScale and the DSP-enhanced ARM microprocessors implement this instruction in two slightly different ways.

- ARM XScale implementation
  This instruction moves the value in register RdLo to bits [31:0] of the 40-bit accumulator (**acc0**), and moves bits [7:0] of the value in register RdHi into bits [39:32] of acc0.
- ARM DSP-enhanced implementation
  This instruction causes the two general-purpose registers **Rd** and **Rn** to transfer values to the coprocessor.

Requirements

Routine	Required header	Architecture
_WriteCoProcessor	<armintr.h>	ARM10, ARM-DSP

See Also

Reference

ARM XSCALE Intrinsic Functions
ARM DSP-enhanced Intrinsic Functions
_ReadCoProcessor

ARM XSCALE Intrinsic Functions

ARM XSCALE Intrinsic Functions

To increase the performance and precision of the audio processing algorithms, the Intel 80200(XScale) microprocessor implementation of ARM adds a Digital Signal Processing (DSP) coprocessor.

This coprocessor contains a 40-bit accumulator and new instructions.

To implement the intrinsic functions for the ARM XScale instruction set, use the /QRxscale - Specify XSCALE Target compiler option when compiling your code.

The following XScale instructions are implemented through intrinsic functions.

Function	ARM XScale instruction	Description
_SmulAdd_SL_ACC	**MIA**	Multiplies the signed value in register **Rs** by the signed value in register **Rm**, and then adds the result to the 40-bit accumulator.
_SmulAddPack_2SW_ACC	**MIAPH**	Performs two 16x16 signed multiplications on packed half-word data and accumulates these to a single 40-bit accumulator.
_SmulAddLo_SW_ACC _SmulAddHi_SW_ACC _SmulAddLoHi_SW_ACC _SmulAddHiLo_SW_ACC	**MIAxy**	Performs one 16-bit signed multiplication and accumulates the result to a single 40-bit accumulator.
_ReadCoProcessor	**MAR**	Moves 64 bits of data from ARM registers to coprocessor registers.
_WriteCoProcessor	**MRA**	Moves 64 bits of data to ARM registers from coprocessor registers.
_PreLoad	**PLD**	This instruction is used as a hint to the memory system that a memory access from the specified address will occur shortly.

See Also

Reference

ARM10 Intrinsic Functions

ARM XSCALE Intrinsic Functions

_PreLoad

_PreLoad

This instruction is a soft preload instruction; that is, this instruction indicates to the memory system that a memory access from the specified address will occur shortly.

```
void _PreLoad(
  unsigned long* addr
);
```

Parameters

addr

Location of memory access.

Return Values

None.

Remarks

The compiler translates this instruction into the **PLD** assembly instruction.

Requirements

Routine	Required header	Architecture
_PreLoad	<armintr.h>	ARM10, ARM-DSP, ARM XSCALE

See Also

Reference

ARM XSCALE Intrinsic Functions

_SmulAdd_SL_ACC

_SmulAdd_SL_ACC

This operation multiplies the signed value in register **Rs** by the signed value in register **Rm** and then adds the result to the 40-bit accumulator, acc0.

```
void _SmulAdd_SL_ACC(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] Values in **Rs** to be written to coprocessor.

Arg2

[in] Values in **Rm** to be written to coprocessor.

Return Values

None.

Remarks

The compiler translates this instruction into the **mia** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAdd_SL_ACC	<armintr.h>	ARM10, ARM-DSP, ARM XSCALE

See Also

Reference

ARM XSCALE Intrinsic Functions

_SmulAddHi_SW_ACC

_SmulAddHi_SW_ACC

This instruction multiplies the top half of **Rm** and the top half of **Rs** and accumulates the result to a single 40-bit accumulator.

The instruction does not support unsigned multiplication, but interprets all arguments as signed data values.

```
void _SmulAddHi_SW_ACC(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] Value in **Rm**.

Arg2

[in] Value in **Rs**.

Return Values

None.

Remarks

The compiler translates this instruction into the **miatt** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddHi_SW_ACC	<armintr.h>	ARM10, ARM-DSP, ARM XSCALE

See Also

Reference

ARM XSCALE Intrinsic Functions
_SmulAddLo_SW_ACC
_SmulAddHiLo_SW_ACC
_SmulAddLoHi_SW_ACC

_SmulAddHiLo_SW_ACC

_SmulAddHiLo_SW_ACC

This ARM XScale instruction multiplies the top half of **Rm** and the bottom half of **Rs** and accumulates the result to a single 40-bit accumulator.

The instruction does not support unsigned multiplication, but interprets all arguments as signed data values.

```
void _SmulAddHiLo_SW_ACC(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] Value in **Rm**.

Arg2

[in] Value in **Rs**.

Return Values

None.

Remarks

The compiler translates this instruction into the **miatb** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddHiLo_SW_ACC	<armintr.h>	ARM10, ARM-DSP, ARM XSCALE

See Also

Reference

ARM XSCALE Intrinsic Functions
_SmulAddLo_SW_ACC
_SmulAddHi_SW_ACC
_SmulAddLoHi_SW_ACC

_SmulAddLo_SW_ACC

_SmulAddLo_SW_ACC

This ARM XScale instruction multiplies the bottom half of **Rm** and the bottom half of **Rs** and accumulates the result to a single 40-bit accumulator.

The instruction does not support unsigned multiplication, but interprets all arguments as signed data values.

```
void _SmulAddLo_SW_ACC(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] Value in **Rm**.

Arg2

[in] Value in **Rs**.

Return Values

None.

Remarks

The compiler translates this instruction into the **miabb** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddLo_SW_ACC	<armintr.h>	ARM10, ARM-DSP, ARM XSCALE

See Also

Reference

ARM XSCALE Intrinsic Functions
_SmulAddHi_SW_ACC
_SmulAddHiLo_SW_ACC
_SmulAddLoHi_SW_ACC

_SmulAddLoHi_SW_ACC

_SmulAddLoHi_SW_ACC

This ARM XScale instruction multiplies the bottom half of **Rm** and the top half of **Rs** and accumulates the result to a single 40-bit accumulator.

The instruction does not support unsigned multiplication, but interprets all arguments as signed data values.

```
void _SmulAddLoHi_SW_ACC(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] Value in **Rm**.

Arg2

[in] Value in **Rs**.

Return Values

None.

Remarks

The compiler translates this instruction into the **miabt** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddLoHi_SW_ACC	<armintr.h>	ARM10, ARM-DSP, ARM XSCALE

See Also

Reference

ARM DSP-enhanced Intrinsic Functions
ARM XSCALE Intrinsic Functions
_SmulAddLo_SW_ACC
_SmulAddHi_SW_ACC
_SmulAddHiLo_SW_ACC

_SmulAddPack_2SW_ACC

_SmulAddPack_2SW_ACC

This ARM XScale instruction performs two 16x16 signed multiplication on packed half word data and accumulates these to a single 40-bit accumulator.

```
void _SmulAddPack_2SW_ACC(
  int Arg1,
  int Arg2
);
```

Parameters

Arg1

[in] Value in **Rm**.

Arg2

[in] Value in **Rs**.

Return Values

None.

Remarks

First, this instruction multiplies the lower 16 bits of the value in **Rm** with the lower 16 bits of the value in **Rs**.

It then performs a multiplication with the upper 16 bits of **Rs** and **Rm**.

The instruction sign-extends both signed 32-bit products and then adds them to the value in the 40-bit accumulator (acc0).

The compiler translates this instruction into the **miaph** assembly instruction.

Requirements

Routine	Required header	Architecture
_SmulAddPack_2SW_ACC	<armintr.h>	ARM10, ARM-DSP, ARM XSCALE

See Also

Reference

ARM XSCALE Intrinsic Functions

WMMX Intrinsic Functions

WMMX Intrinsic Functions

The ARM-compliant Intel processors such as the PXA270 processor with Wireless MMX (WMMX) technology have instructions to enable development of optimized multimedia applications. The instructions are implemented as co-processor instructions.

This technology uses the single-instruction, multiple-data (SIMD) technique. By processing data elements in parallel, applications with media-rich bit streams can significantly improve performance by using SIMD instructions.

For complete details of the hardware instructions and intrinsic functions, data types, and registers can be found in the Intel Wireless MMX Technology Developer Guide, number 251793-001
http://www.intel.com/design/pca/prodbref/251669_devguide.pdf.

In This Section

External Resources

Intel Wireless MMX Technology Developer Guide, number 251793-001

WMMX Technology Overview

WMMX Technology Overview

Intel's wireless MMX (WMMX) technology is an extended implementation of the Intel architecture (IA) MMX instruction set. The technology uses a single-instruction, multiple-data (SIMD) technique to speed up multimedia and communications software by processing data elements in parallel.

The WMMX instruction set adds 57 opcodes and a 64-bit data type. In addition, there are sixteen 64-bit MMX technology registers, each of which can be directly addressed using the register names wR0 to wR15.

Additional information and details about the MMX instructions, data types, and registers can be found in the Intel Wireless MMX Technology Developer Guide, number 251793-001. This guide is available online at Intel hardware design.

New Registers

The WMMX technology intrinsic functions provide sixteen registers (wR0 to wR15) that are 64 bits long (0 to 63).

These new data registers enable the processing of data elements in parallel. Because each register can hold more than one data element, the processor can process more than one data element simultaneously. This processing capability is also known as SIMD processing. To enable SIMD processing with the C/C++ compiler, new data types are defined to exploit the expanded size of the new registers.

Using intrinsic functions allows you to code with the syntax of C function calls and variables instead of with the assembly language. For each computational and data manipulation instruction in the new extension sets, there is a corresponding C intrinsic that directly implements that instruction. This frees you from managing registers and assembly programming. Further, the compiler optimizes the instruction scheduling so that your executable runs faster.

__m64 data type

The **__m64** data type is used to represent the contents of a WMMX register, which is the register used by the WMMX technology intrinsic functions. The **__m64** data type can hold eight 8-bit values, four 16-bit values, two 32-bit values, or one 64-bit value.

New Data Types Usage Guidelines

The new **__m64** data type is not a basic ANSI C data type, and therefore you must observe the following usage restrictions:

- Use **__m64** only on the left side of an assignment as a return value or as a parameter. You cannot use it with other arithmetic expressions (" + ", " ", and so on).
- Use **__m64** as objects in aggregates, such as unions, to access the byte elements and structures.
- Use **__m64** only with the WMMX intrinsic functions.

Data Alignment

Many of the WMMX intrinsic functions have data alignment requirements. If these intrinsic functions are used and data is not appropriately aligned, the program will throw an exception that must be handled by the program; otherwise, the program will fault. To support the use of WMMX intrinsic functions, the user must take a more active role to guarantee that alignment issues are appropriately addressed.

For more information, see RISC Processor Data Alignment.

See Also
Other Resources
WMMX Intrinsic Functions

WMMX Arithmetic Intrinsics

WMMX Arithmetic Intrinsics

The intrinsics listed in the following table perform packed arithmetic operations.

Intrinsic name	Operation	WMMX Instruction	Argument and result values/bits
_mm_add_pi16	Adds	WADDH	4/16, 4/16
_mm_add_pi32	Adds	WADDW	2/32, 2/32
_mm_add_pi8	Adds	WADDB	8/8, 8/8
_mm_adds_pi16	Adds, signed	WADDHSS	4/16, 4/16
_mm_adds_pi32	Adds, signed	WADDWSS	2/32, 2/32
_mm_adds_pi8	Adds, signed	WADDBSS	8/8, 8/8
_mm_adds_pu16	Adds	WADDHUS	4/16, 4/16
_mm_adds_pu32	Adds	WADDWUS	2/32, 2/32
_mm_adds_pu8	Adds	WADDBUS	8/8, 8/8
_mm_sub_pi16	Subtracts	WSUBH	4/16, 4/16
_mm_sub_pi32	Subtracts	WSUBW	2/32, 2/32
_mm_sub_pi8	Subtracts	WSUBB	8/8, 8/8
_mm_subs_pi16	Subtracts, signed	WSUBHSS	4/16, 4/16
_mm_subs_pi32	Subtracts, signed	WSUBWSS	2/32, 2/32
_mm_subs_pi8	Subtracts, signed	WSUBBSS	8/8, 8/8
_mm_subs_pu16	Subtracts	WSUBHS	4/16, 4/16
_mm_subs_pu32	Subtracts	WSUBWSS	2/32, 2/32
_mm_subs_pu8	Subtracts	WSUBBUS	8/8, 8/8
_mm_madd_pi16	Multiplies	WMADDS	4/16, 2/32
_mm_madd_pu16	Multiplies	WMADDU	4/16, 2/32
_mm_mulhi_pi16	Multiplies, signed	WMULSH	4/16, 4/16 (high)
_mm_mulhi_pu16	Multiplies, signed	WMULUH	4/16, 4/16 (high)
_mm_mullo_pi16	Multiplies	WMULSL/WMULUL	4/16, 4/16 (low)

_mm_mac_pi16	Multiply-accumulate, signed	WMACS	4/16, 4/16
_mm_mac_pu16	Multiply-accumulate	WMACU	4/16, 4/16
_mm_macz_pi16	Multiply-accumulate, signed	WMACSZ	4/16, 4/16
_mm_macz_pu16	Multiply-accumulate	WMACUZ	4/16, 4/16
_mm_acc_pu16	Accumulate	WACCH	4/16, 1/16
_mm_acc_pu32	Accumulate	WACCW	2/32. 2/32
_mm_acc_pu8	Accumulate	WACCB	8/8, 1/8
_mm_mia_si64	Multiply-accumulate, signed	TMIA	1/64
_mm_miabb_si64	Multiply-accumulate, signed	TMIABB	1/64
_mm_miabt_si64	Multiply-accumulate, signed	TMIABT	1/64
_mm_miaph_si64	Multiply-accumulate, signed	TMIAPH	1/64
_mm_miatb_si64	Multiply-accumulate, signed	TMIATB	1/64
_mm_miatt_si64	Multiply-accumulate, signed	TMIATT	1/64

See Also
Other Resources
WMMX Intrinsic Functions

WMMX Shift Intrinsics

WMMX Shift Intrinsics

The intrinsics listed in the following table perform shift operations.

Intrinsic name	Shift direction	Shift type	WMMX instruction
_mm_sll_pi16	Left	Logical	WSLLH
_mm_slli_pi16	Left	Logical	Composite
_mm_sll_pi32	Left	Logical	WSLLW
_mm_slli_pi32	Left	Logical	Composite
_mm_sll_si64	Left	Logical	WSLLD
_mm_slli_si64	Left	Logical	Composite
_mm_sra_pi16	Right	Arithmetic	WSRAH
_mm_srai_pi16	Right	Arithmetic	Composite
_mm_sra_pi32	Right	Arithmetic	WSRAW
_mm_srai_pi32	Right	Arithmetic	Composite
_mm_sra_pi64	Right	Arithmetic	WSRAD
_mm_srai_pi64	Right	Arithmetic	Composite
_mm_srl_pi16	Right	Logical	WSRLH
_mm_srli_pi16	Right	Logical	Composite
_mm_srl_pi32	Right	Logical	WSRLW
_mm_srli_pi32	Right	Logical	Composite
_mm_srl_si64	Right	Logical	WSRLD
_mm_srli_si64	Right	Logical	Composite
_mm_ror_pi16	Rotate right	Logical	WRORH
_mm_rori_pi16	Rotate right	Logical	WRORW
_mm_ror_pi32	Rotate right	Logical	WRORD
_mm_rori_pi32	Rotate right	Logical	Composite
_mm_ror_si64	Rotate right	Logical	Composite

_mm_rori_si64	Rotate right	Logical	Composite

See Also

Other Resources

WMMX Intrinsic Functions

WMMX Logical Intrinsics

WMMX Logical Intrinsics

The intrinsics listed in the following table perform logical operations.

Intrinsic name	Operation	WMMX instruction
_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

See Also

Other Resources

WMMX Intrinsic Functions

WMMX Compare Intrinsics

WMMX Compare Intrinsics

The intrinsics listed in the following table perform comparisons.

Intrinsic name	Comparison	Number of elements	Element bit size	WMMX instruction
_mm_cmpeq_pi8	Equals	8	8	WCMPEQB
_mm_cmpeq_pi16	Equals	4	16	WCMPEQH
_mm_cmpeq_pi32	Equals	2	32	WCMPEQW
_mm_cmpgt_pi8	Greater than, signed	8	8	WCMPGTSB
_mm_cmpgt_pu8	Greater than, unsigned	8	8	WCMPGTUB
_mm_cmpgt_pi16	Greater than, signed	4	16	WCMPGTSH
_mm_cmpgt_pu16	Greater than, unsigned	4	16	WCMPGTUH
_mm_cmpgt_pi32	Greater than, signed	2	32	WCMPGTSW
_mm_cmpgt_pu32	Greater than, unsigned	2	32	WCMPGTUW

See Also

Other Resources

WMMX Intrinsic Functions

WMMX Pack/Unpack Intrinsics

WMMX Pack/Unpack Intrinsics

The intrinsic functions listed in the following table provide pack and unpack operations.

Intrinsic name	Operation	WMMX Instruction
_mm_packs_pi16	Packs, signed and saturated	WPACKHSS
_mm_packs_pi32	Packs, signed and saturated	WPACKWSS
_mm_packs_pu16	Packs, saturated	WPACKHUS
_mm_unpackhi_pi8	Interleaves	WUNPCKIHB
_mm_unpackhi_pi16	Interleaves	WUNPCKIHH
_mm_unpackhi_pi32	Interleaves	WUNPCKIHW
_mm_unpacklo_pi8	Interleaves	WUNPCKILB
_mm_unpacklo_pi16	Interleaves	WUNPCKILH
_mm_unpacklo_pi32	Interleaves	WUNPCKILW
_mm_packs_si64	Packs, signed and saturated	WPACKDSS
_mm_packs_su64	Packs, saturated	WPACKDUS
_mm_packs_pu32	Packs, saturated	WPACKWUS
_mm_unpackeh_pi8	Interleaves, signed and extended	WUNPCKEHSB
_mm_unpackeh_pi16	Interleaves, signed and extended	WUNPCKEHSH
_mm_unpackeh_pi32	Interleaves, signed and extended	WUNPCKEHSW
_mm_unpackeh_pu8	Interleaves, unsigned and extended	WUNPCKEHUB
_mm_unpackeh_pu16	Interleaves, unsigned and extended	WUNPCKEHUH
_mm_unpackeh_pu32	Interleaves, unsigned and extended	WUNPCKEHUW
_mm_unpackel_pi8	Interleaves, signed and extended	WUNPCKELSB
_mm_unpackel_pi16	Interleaves, signed and extended	WUNPCKELSH
_mm_unpackel_pi32	Interleaves, signed and extended	WUNPCKELSW
_mm_unpackel_pu8	Interleaves, extended	WUNPCKELUB
_mm_unpackel_pu16	Interleaves, extended	WUNPCKELUH

_mm_unpackel_pu32	Interleaves, extended	WUNPCKELUW

See Also

Other Resources

WMMX Intrinsic Functions

WMMX Set Intrinsics

WMMX Set Intrinsics

The intrinsics listed in the following table are made up of composite instructions.

Intrinsic name	Operation	Number of elements	Signed	Reverse order	WMMX Instruction
_mm_setzero_si64	Sets to zero	1	No	No	WZERO
_mm_set_pi32	Sets integer values	2	No	No	Composite
_mm_set_pi16	Sets integer values	4	No	No	Composite
_mm_set_pi8	Sets integer values	8	No	No	Composite
_mm_set1_pi32	Sets integer values	2	Yes	No	TBCSTW
_mm_set1_pi16	Sets integer values	4	Yes	No	TBCSTH
_mm_set1_pi8	Sets integer values	8	Yes	No	TBCSTB
_mm_setr_pi32	Sets integer values	2	No	Yes	Composite
_mm_setr_pi16	Sets integer values	4	No	Yes	Composite
_mm_setr_pi8	Sets integer values	8	No	Yes	Composite
_mm_setwcx	Set control register	--	--	--	TMCR
_mm_getwcx	Get control register	--	--	--	TMRC

See Also

Other Resources

WMMX Intrinsic Functions

WMMX General Support Intrinsics

WMMX General Support Intrinsics

The intrinsic functions listed in the following table provide general support operations.

Intrinsic name	Operation	WMMX Instruction
_mm_extract_pi8	Extracts on 8 bytes	TEXTRMSB
_mm_extract_pi16	Extracts on 4 half words	TEXTRMSH
_mm_extract_pi32	Extracts on two words	TEXTRMSW
_mm_extract_pu8	Extracts on 8 bytes	TEXTRMUB
_mm_extract_pu16	Extracts on 4 half words	TEXTRMUL
_mm_extract_pu32	Extracts on two words	TEXTRMUW
_mm_insert_pi8	Inserts a byte.	TINSRB
_mm_insert_pi16	Inserts a half word	TINSRH
_mm_insert_pi32	Inserts a word	TINSRW
_mm_max_pi8	Computes maximum	WMAXSB
_mm_max_pi16	Computes maximum	WMAXSH
_mm_max_pi32	Computes maximum	WMAXUB
_mm_max_pu8	Computes maximum, unsigned	WMAXUB
_mm_max_pu16	Computes maximum, unsigned	WMAXUH
_mm_max_pu32	Computes maximum, unsigned	WMAXUW
_mm_min_pi8	Computes minimum	WMINXSB
_mm_min_pi16	Computes minimum	WMINSH
_mm_min_pi32	Computes minimum	WMINUB
_mm_min_pu8	Computes minimum, unsigned	WMINUB
_mm_min_pu16	Computes minimum, unsigned	WMINUH
_mm_min_pu32	Computes minimum, unsigned	WMINUW
_mm_movemask_pi8	Creates an 8-bit mask	TMOVMSKB
_mm_movemask_pi16	Creates a 4 half-word mask	TMOVMSKH

_mm_movemask_pi32	Creates a two word mask	TMOVMSKW
_mm_shuffle_pi8	Returns a combination of 4 words	WSHUFH
_mm_avg_pu8	Computes rounded average	WAV2BR
_mmavg_pu16	Computes rounded average	WAV2HR
_mmavg2_pu8	Computes average	WAVG2B
_mmavg2_pu16	Computes average	WAVG2H
_mm_sada_pu8	Computes absolute sum of differences	WSADB
_mm_sada_pu16	Computes absolute sum of differences	WSADH
_mm_align_si64	Extracts a 64-bit value	WALIGN/WALIGNR
_mm_cvtsi32_si64	Converts from int to a 64-bit __m64 object	TMCRR
_mm_cvtsi64_si32	Converts from__m64 object to int	TMRRC
_mm_empty	Empties MM state	---

See Also

Other Resources

WMMX Intrinsic Functions

ARM Compiler Options

ARM Compiler Options

The following table shows compiler options for ARM microprocessors.

Option	Description
/QRArch - Specify Target Architecture	Specifies target ARM microprocessors.
/QRimplicit-import - Disable Importing of Helpers	Statically links compiler helper functions.
/QRinterwork-return - Enable Interworking	Generates code that can interwork between ARM and Thumb mode.
/QRxscale - Specify XSCALE Target	Generates code for Intel XScale microprocessors.
/QRxscalesched	Generates code that enables internal scheduler optimizations for the XScale pipeline.
/QRthumb	Generates code for Thumb mode.

See Also

Other Resources

ARM Family Processors

/QRArch - Specify Target Architecture

/QRArch - Specify Target Architecture

This option specifies which ARM architecture the compiler will target. The following line shows the usage of this option:

```
/Qrarch{4|5} [T]
```

The following options are available for this switch.

4

Indicates the ARM4 architecture.

5

Indicates the ARM5 architecture.

T

Indicates that the architecture supports the ARM/Thumb interworking instructions.

See Also

Other Resources

ARM Family Processors

/QRimplicit-import - Disable Importing of Helpers

/QRimplicit-import - Disable Importing of Helpers

This option forces the compiler to use statically linked helper functions.

Compiler helper functions are typically located in a DLL. To reduce the thunk overhead when calling these functions, the compiler uses the dllimport calling mechanism. If OS or driver code is linked statically with the helper functions, the **/QRimplicit-import**- flag is used to force the compiler to use typical calling mechanisms.

See Also
Other Resources
ARM Calling Sequence Specification

/QRinterwork-return - Enable Interworking

/QRinterwork-return - Enable Interworking

This option instructs the compiler to generate code that enables programs to call functions from ARM or Thumb mode and return correctly.

See Also
Concepts
THUMB-enabled ARM Implementation

/QRxscale - Specify XSCALE Target

/QRxscale - Specify XSCALE Target

This option specifies that the compiler should generate code that runs on an Intel XScale processor. This enables the XScale MAC intrinsics and causes the scheduler to optimize for the XScale pipeline.

See Also

Reference

ARM XSCALE Intrinsic Functions

/QRxscalesched

/QRxscalesched

This option enables a re-ordering of instructions in the generated code to take advantage of XScale-specific optimizations for internal scheduling of instruction execution.

The functionality offered by this option is a proper subset of the functionality of the /QRxscale - Specify XSCALE Target option.

See Also

Reference

ARM XSCALE Intrinsic Functions

/QRthumb

/QRthumb

This option enables the ARM Thumb instruction set.

See Also

Concepts

THUMB-enabled ARM Implementation

ARM Calling Sequence Specification

ARM Calling Sequence Specification

The ARM Calling Sequence Specification for the ARM device compiler provides direction for the development of compilers and assembly language programs for the Arm family of microprocessors.

In This Section

ARM Registers

Shows the assigned register roles.

ARM Stack Frame Layout

Describes how ARM microprocessors specify the stack layout.

ARM Prolog and Epilog

Describes the prolog and epilog code segments required for Structured Exception Handling.

THUMB-enabled ARM Implementation

Describes how to implement ARM-Thumb interworking.

ARM Assembler

Describes ARM Assembler macros for prolog and epilog, directives, and command-line options.

Related Sections

ARM Compiler Options

Provides reference information about compiler options for ARM compilers.

Intrinsic Functions for ARM Microprocessors

Provides reference information about intrinsic functions supported only by ARM compilers

Differences Between Desktop and Device Compilers

Describes critical differences in implementation between desktop and device compilers.

ARM Registers

ARM Registers

The ARM microprocessor has 16 general-purpose registers.

THUMB has eight general-purpose registers, **R0-R7**, and access to the high registers, **R8-R15**.

The following table shows the assigned register roles.

Register	Description
R0	Argument1, Return Value Temporary register
R1	Argument2, Second 32-bits if double/int Return Value Temporary register
R2-R3	Arguments Temporary registers
R4-R10	**R7** is THUMB frame pointer Permanent registers.
R11	ARM frame pointer Permanent register
R12	Temporary register
R13	Stack pointer Permanent register
R14	Link register Permanent register
R15	Program Counter

Note that registers **R0** through **R3** hold the first four words of incoming arguments. The microprocessor constructs remaining arguments in the calling function's argument build area, which does not provide space into which **R0** through **R3** can be spilled.

The following table shows additional predefined registers for Vector Floating Point and for WMMX.

Register	Description
s0-s32	VFP single-precision registers
d0-d16	VFP double-precision registers
fpsid	VFP system ID register
fpscr	VFP status and control register
fpexc	VFP exception register

wr0-wr16	WMMX SIMD data registers
wc0-wc16	WMMX status and control registers
wcid	WMMX coprocessor ID register, synonymous with **wc0**
wcon	WMMX control register, synonymous with **wc1**
wcssf	WMMX saturation SIMD flags, synonymous with **wc2**
wcasf	WMMX arithmetic SIMD flags, synonymous with **wc3**
wcgr0-wcgr3	WMMX control general-purpose registers, synonymous with **wc8-wc11**

See Also

Reference

ARM Assembler Macros

Concepts

ARM Stack Frame Layout

ARM Stack Frame Layout

ARM Stack Frame Layout

The following list gives more information about ARM microprocessor stack frame layout.

- The **Register Save Area** (RSA) holds the preserved values of permanent registers used by the function. It also contains the function return address.
- The **Locals and Temporaries** area represents the stack space allocated for local variables and compiler-generated temporaries.
- The first four words at the top of the stack can contain the values passed in **R0-R3**. Any of these values could be missing. The values should be stored in the **R0-R3** if registers cannot hold the arguments for the entire function, or if the addresses for the arguments are in use.
  If a routine needs storage space for the first four words of arguments, it creates and initializes the storage at the top of the called function stack.
  If a register keeps an argument for the argument live range, the argument has no associated storage in the stack frame.

ARM Frame and Stack Pointers

A frame pointer helps mitigate problems with the limited size of the bit field that specifies register-displacement-addressing offset. The frame pointer typically points to a fixed frame offset in the RSA or Local and Temporaries areas of the stack frame, but the pointer can point to other offsets within the frame. To more efficiently access data in large stack frames, a routine can establish another frame pointer.

- A routine does not need to set up a stack frame unless it needs to save permanent registers, or to allocate space for locals or outgoing argument areas that are bigger than four words. The stack pointer and frame pointer addresses align on 4-byte boundaries.
- If a routine has **alloca()** locals, the ARM specification requires a separate frame pointer register to access incoming arguments and locals.
  R11 is the assigned frame pointer for ARM, and **R7** is the assigned frame pointer for THUMB.
  A leaf routine can use any free integer register as the frame pointer. A nonleaf routine must use a permanent register. The routine must not modify the frame pointer register between the prolog and epilog.
- If a routine uses **alloca()**, everything in the frame at a lower address than the **alloca()** area is referenced relative to **R13** and never contains a defined value at the time of an **alloca()** call. Thus, the **alloca()** operation never needs to copy this part of the stack frame, and no data relocation problems arise.
  Everything in the frame at an address higher than the **alloca()** area is referenced relative to the frame pointer, **R11** for ARM or **R7** for THUMB.

See Also

Concepts

ARM Registers

ARM Prolog and Epilog

ARM Prolog and Epilog

ARM prolog and epilog code segments are required to implement Structured Exception Handling (SEH) for ARM microprocessors. The ARM prolog is a code segment that sets up the stack frame for a routine. The epilog removes the routine frame and returns from the routine.

In This Section

ARM Prolog

Describes the requirements of an ARM prolog, and provides an example.

ARM Epilog

Describes the requirements of an ARM epilog, and provides an example.

THUMB Prolog

Describes the requirements of an THUMB prolog, and provides an example.

THUMB Epilog

Describes the requirements of an THUMB epilog, and provides an example.

ARM Assembler Macros

Provides reference information about the ARM Assembler macros used to define the ARM and THUMB prolog and epilog sequences.

Related Sections

ARM Calling Sequence Specification

Provides links to reference information about the ARM calling sequence specification.

SEH in RISC Environments

Describes the implementation of Structured Exception Handling in RISC environments such as ARM.

ARM Prolog

ARM Prolog

The ARM prolog has three constituent parts. All parts are immediately contiguous, with no intervening instructions. When the prolog follows this guideline, the **Virtual Unwinder** can reverse-execute the prolog.

The following list shows the three required parts of an ARM prolog.

1. A sequence of zero or one instructions that push the incoming argument values from **R0, R1, R2,** and **R3** to the argument home locations, and updates **R13** to the new stack top.
   This sequence saves all permanent registers in descending order at the top of the stack frame, following any saved argument registers.
2. A sequence of one or more instructions that set up the frame pointer, if one is to be established.
   The prolog copies stack pointer **R13** to **R12** before the initial register saves and uses **R12** to compute the value of the frame pointer, **R11**.
3. A sequence of zero or more instructions that allocate the remaining stack frame space for local variables, compiler-generated temporaries, and the argument build area by subtracting a 4-byte aligned offset from **R13**.
   If an offset is too wide to represent in the immediate field, the prolog uses the scratch register **R12** to hold the offset. In this case, it sets the value in **R12** with a different instruction.

Examples

The following examples show how to construct several ARM prologs.

- ARM Prolog with frame in **R11**.

```
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>
```

- ARM Prolog with no frame.

```
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>
```

See Also

Concepts

SEH in RISC Environments

ARM Epilog

ARM Epilog

ARM Epilog

The ARM epilog is a contiguous sequence of instructions that does the following:

- Restores the saved permanent registers
- Resets the stack pointer to its value on function entry
- Returns to the function's calling function

The following list shows the guidelines for implementing an epilog:

- All parts are immediately contiguous, with no intervening instructions.
- If a frame pointer is set up, the epilog is a single instruction that uses the frame as the base and updates all nonvolatile registers, including the Program Counter and the stack.
  If no frame is set up, the epilog consists of a stack unlink, if needed, followed by an instruction to restore multiple registers or to copy the link register **R14** to the program counter.
  If the function establishes a frame pointer (which has the value of **R11** for an ARM epilog), the function must not modify the pointer value during the interval between the completion of the prolog's last instruction and the beginning of the execution of the first instruction of the epilog.
  If the function does not establish a frame pointer (which has the value of **R13**), the function must not modify the stack pointer during the interval between the completion of the prolog's last instruction and the beginning of the execution of the first instruction of the epilog.
- In a routine that does not modify nonvolatile registers and is not interworking, the epilog contains only a copy of the link register to the program counter.
- A routine whose last instruction is a branch to another routine can have an empty epilog if it does not modify nonvolatile registers.
- The address contained in the stack pointer, which always has the value of **R13**, must never be greater than the lowest address of any unrestored register value in the Register Save Area.
  This prevents the preserved values of the permanent registers from being corrupted by a context switch or any other asynchronous event that might occur during the execution of a prolog or epilog.

Example

The following examples show a variety of ARM epilogs.

- ARM Epilog with frame in **R11**.

```
<no stack unlink>
LDMDB  r11, {r4-r11, r13, r15}
```

- ARM Epilog with no frame.

```
<stack unlink if needed>
LDMIA  r13, {r4-R11, r13, r15}
```

- ARM Epilog with interworking return.

```
<stack unlink if needed>
LDMIA   r13, {r4-r11, r13, LR}
BX      LR
```

See Also

Concepts

SEH in RISC Environments
ARM Prolog

THUMB Prolog

THUMB Prolog

A THUMB prolog has the following specific parts. All parts are immediately contiguous with no intervening instructions.

- A sequence of zero or one instructions that push the incoming argument values in **R0, R1, R2,** and **R3** to the argument home locations, and updates **R13** to the new stack top.
  If the function does not use high registers such as **R8, R9, R10,** and **R11**, a sequence of instructions pushes **R4-R7** or the link register **R14** to the stack.
  The function does not push the link register if the routine is a leaf with no high registers saved.
- A sequence of zero or more instructions that allocate the remaining stack frame space for local variables, compiler-generated temporaries, and the argument build area by subtracting an aligned offset from **R13**.
- A single instruction that sets the frame pointer if one is to be established.
  Immediately after it links the stack, the function copies the value of the stack pointer in **R13** to **R7**.

Example

The following examples show a variety of THUMB prologs.

- THUMB Prolog with no frame.

```
PUSH    {r0-r3}    ; As needed
PUSH    {r4-r7, LR}  ; As needed
SUB    SP, SP, #4  ; Stack link (as needed)
```

- THUMB Prolog with frame in **R7**.

```
PUSH    {r0-r3}      ; As needed
PUSH    {r4-r7, LR}    ; As needed
SUB    SP, SP, #4    ; Stack link (as needed)
MOV    r7, SP      ; Set frame
```

- THUMB Prolog with interworking return.

```
PUSH    {r4-r7, LR}
```

- THUMB Prolog saving high registers.

```
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)
```

- THUMB Prolog with a large stack frame.

```
PUSH    {r7}
LDR    r7, [PC, #20]
NEG    r7, r7
ADD    SP, r7
```

See Also

Concepts

SEH in RISC Environments
THUMB Epilog

THUMB Epilog

THUMB Epilog

The THUMB epilog is a contiguous sequence of instructions that does the following:

- Restores the saved permanent registers
- Resets the stack pointer to its value on function entry
- Returns to the function's calling function

The following list shows important information about an epilog.

- If a frame pointer has been set up by the prolog, the epilog restores the stack pointer from the frame pointer. It uses a single **mov** instruction to copy **R7** to **R13**. This copy restores **R13** to the value it had just after the prolog linked the stack.Unlinking the stack restores **R13**.
- If {**R4-R7**} are to be restored, and if high registers are not to be restored, the epilog must contain an instruction to pop {**R4-R7**}. This instruction updates the stack pointer. If the **Unwinder** restores high registers, it also restores **R4-R7**.
- The epilog contains a sequence of zero or two instructions that pop the link register from the stack into **R3**.
  If the routine is a leaf or if the routine restores high registers, the epilog omits this sequence.
  If the epilog restores high registers, it uses a branch and link to call a restore-register routine that restores **R4** through **R7** and the link register. It also restores the stack pointer to the value it held just after the prolog saved {**R0-R3**}, or to the value the stack pointer held on entry of the function if { **R0-R3**} were saved.
- In a non-interworking routine that does not need to save registers other than { **R4-R7,LR**}, the epilog ends with pop {**R4-R7, PC**} or pop { **PC** } if { **R4-R7**} do not need to be saved. If a noninterworking routine needs to save other registers, the epilog ends with a copy of **R3** to the **PC**.
  In an interworking routine, the epilog ends with a branch and exchange (**BX**) to **R3**.

Example

The following examples show a variety of THUMB epilogs.

- THUMB Epilog with no frame.

```
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
```

- THUMB Epilog with frame in **R7**.

```
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
```

- THUMB Epilog restoring high registers.

```
ADD    SP, SP, #4    ; Stack link (as needed)
BL    __restgpr_9
ADD    SP, SP, #16    ; POP r0-r3 as needed
BX    LR
```

- THUMB Epilog with a large stack frame.

```
LDR    r7, [PC, #4]
ADD    SP, r7
POP    {r7}
```

```
    MOV     PC, LR
```

See Also

Concepts

SEH in RISC Environments

THUMB Prolog

ARM Assembler Macros

ARM Assembler Macros

ARM assembler-level macros are required to implement prolog and epilog code segments.

The following assembler level macros are available for ARM microprocessors.

Macro	Description
ALTERNATE_ENTRY (ARM)	Declares an alternate entry to a routine.
END_REGION (ARM)	Marks the end of a contiguous range of text or data.
ENTRY_END (ARM)	Ends the routine that was specified by a prior NESTED_ENTRY (ARM).
EXCEPTION_HANDLER (ARM)	Associates a named exception handler with the subsequent NESTED_ENTRY (ARM).
EXCEPTION_HANDLER_DATA (ARM)	Associates a named exception handler and the handler data with the subsequent NESTED_ENTRY (ARM).
LEAF_ENTRY (ARM)	Declares the beginning of a routine that does not require prolog code.
NESTED_ENTRY (ARM)	Declares the beginning of a routine that either has an existing stack frame or that creates a stack frame.
PROLOG_END (ARM)	Marks the end of the prolog area.
START_REGION (ARM)	Marks the beginning of a contiguous range of text or data.

See Also

Concepts

ARM Registers

ARM Stack Frame Layout

Other Resources

ARM Prolog and Epilog

ALTERNATE_ENTRY (ARM)

ALTERNATE_ENTRY (ARM)

This macro declares an alternate entry to a routine of type NESTED_ENTRY (ARM) or LEAF_ENTRY (ARM).

```
ALTERNATE_ENTRY Name[,[Section=]SectionName]
```

Parameters

Name

The entry point.

SectionName

The name of the section where the entry appears; see Remarks.

Return Value

None.

Remarks

Name is in the global name space.

The **ALTERNATE_ENTRY** macro does not use the *SectionName* parameter. The parameter is accepted and ignored for consistency with NESTED_ENTRY (ARM) and LEAF_ENTRY (ARM).

If used, an **ALTERNATE_ENTRY** call must appear in the body of a routine.

See Also

Reference

NESTED_ENTRY (ARM)

LEAF_ENTRY (ARM)

END_REGION (ARM)

END_REGION (ARM)

This macro marks the end of a contiguous range of text or data.

```
END_REGION NameEnd
```

Parameters
NameEnd

NameEnd labels the end of the range.

Return Values

None.

Remarks

NameEnd is in the global name space.

See Also
Reference
START_REGION (ARM)

ENTRY_END (ARM)

ENTRY_END (ARM)

This macro ends the current routine specified by NESTED_ENTRY (ARM) or LEAF_ENTRY (ARM).

```
ENTRY_END [Name]
```

Parameters
Name

See Remarks.

Return Values

None.

Remarks

Name should be the same name used in the **NESTED_ENTRY** or **LEAF_ENTRY** macros.

The **ENTRY_END** macro ignores *Name*.

See Also
Reference
NESTED_ENTRY (ARM)
LEAF_ENTRY (ARM)

EXCEPTION_HANDLER (ARM)

EXCEPTION_HANDLER (ARM)

This macro associates an exception handler *Handler* with a subsequent NESTED_ENTRY (ARM) or LEAF_ENTRY (ARM) routine.

```
EXCEPTION_HANDLER Handler
```

Parameters
Handler

Name of the exception handler.

Return Values

None.

Remarks

This association is in effect until the compiler encounters a matching ENTRY_END (ARM) macro.

See Also
Reference
NESTED_ENTRY (ARM)
LEAF_ENTRY (ARM)
ENTRY_END (ARM)
EXCEPTION_HANDLER_DATA (ARM)

EXCEPTION_HANDLER_DATA (ARM)

EXCEPTION_HANDLER_DATA (ARM)

This macro associates an exception handler *Handler* and *HandlerData* with a subsequent NESTED_ENTRY (ARM) or LEAF_ENTRY (ARM) routine.

```
EXCEPTION_HANDLER_DATA Handler,     HandlerData
```

Parameters

Handler

Name of the exception handler.

HandlerData

String associated with the exception handler.

Return Values

None.

Remarks

This association is in effect until compiler encounters a matching ENTRY_END (ARM) macro.

See Also

Reference

NESTED_ENTRY (ARM)
LEAF_ENTRY (ARM)
ENTRY_END (ARM)
EXCEPTION_HANDLER (ARM)

LEAF_ENTRY (ARM)

LEAF_ENTRY (ARM)

This macro declares the beginning of a routine that does not require prolog code.

```
LEAF_ENTRY Name[,[Section=]SectionName]
```

Parameters

Name

The routine name.

SectionName

Optional. The name of the section where the entry appears.Defaults to .text.

Return Values

None.

Remarks

Name is in the global name space.

A LEAF_ENTRY must have an associated ENTRY_END (ARM).

See Also

Reference

ENTRY_END (ARM)

PROLOG_END (ARM)

NESTED_ENTRY (ARM)

NESTED_ENTRY (ARM)

This macro declares the beginning of a routine that either has an existing frame or creates a stack frame.

```
NESTED_ENTRY Name[,[Section=]SectionName]
```

Parameters

Name

The routine name.

SectionName

Optional. The name of the section where the entry appears.Defaults to text.

Return Values

None.

Remarks

Name is in the global name space.

A NESTED_ENTRY must have an associated PROLOG_END (ARM) and ENTRY_END (ARM).

See Also

Reference

ENTRY_END (ARM)
PROLOG_END (ARM)

PROLOG_END (ARM)

PROLOG_END (ARM)

This macro marks the end of the prolog area.

```
PROLOG_END
```

Parameters
None.

Return Values

None.

Remarks

This macro must appear following a NESTED_ENTRY (ARM) or LEAF_ENTRY (ARM) macro.

It appears after the prolog area and before the matching ENTRY_END (ARM) macro.

See Also
Reference
NESTED_ENTRY (ARM)
LEAF_ENTRY (ARM)
ENTRY_END (ARM)

START_REGION (ARM)

START_REGION (ARM)

This macro marks the beginning of a contiguous range of text or data.

```
START_REGION NameBegin
```

Parameters
NameBegin

NameBegin labels the beginning of the range. *NameBegin* is in the global name space.

Return Values

None.

See Also
Reference
END_REGION (ARM)

THUMB-enabled ARM Implementation

THUMB-enabled ARM Implementation

THUMB-enabled ARM microprocessors can execute 16-bit THUMB instructions in addition to the usual 32-bit ARM instructions.

To execute THUMB instructions, switch the microprocessor into THUMB mode. To resume executing ARM instructions, switch the microprocessor back into ARM mode. The instruction that accomplishes the mode switch is the branch and exchange (**BX**) instruction.

To call a function compiled by the THUMB compiler from within a function compiled by the ARM compiler, the code sequence must include a **BX** instruction on the caller side and on the callee side.

The **BX** instruction is of the form **BX Rx,** where **Rx,**is a register containing an address.

If the low bit of the **BX** target address is set, **BX** causes a switch into THUMB mode.

If the low bit of the **BX** target address is not set, **BX** causes a switch into ARM mode.

The linker sets the low bit of THUMB function addresses at link time.

The ARM C/C++ compiler supports function calls and returns for ARM/THUMB interworking. The following mechanisms support interworking.

- Compiler flags
- Language extensions by way of interworking declspec modifiers
- Link-time generation of thunking routines

The use of an interworking calling sequence allows but does not require a mode switch to occur.

In addition, a THUMB function can call another THUMB function and an ARM function can call another ARM function through an interworking calling sequence.

See Also

Reference

Compiler Flags for Interworking

Concepts

Modifiers

Linker-Generated Thunking Routines

Code Size Considerations

Compiler Flags for Interworking

Compiler Flags for Interworking

The following compiler flags apply to the creation of interworking programs.

- The /QRinterwork-return - Enable Interworking flag causes all function returns in a file to be made interworking. This flag has no effect on function calls.
- The /QRArch - Specify Target Architecture flag allows the compiler to generate instructions that are only available on THUMB-enabled microprocessors, such as **BX**.
  For THUMB, this flag is on by default.
  For ARM, this flag is off by default.
  The **/QRinterwork-return** flag enables the **/QRarch** flag.

See Also
Concepts
THUMB-enabled ARM Implementation

Modifiers

Modifiers

ARM and THUMB support Microsoft-specific modifiers for interworking. The iwcall, iw16, and iw32 declspecs allow you to select which function calls are interworking.

Compared to a linker-generated thunking routine, the iwcall or iw16 **declspecs** always result in faster code for interworking calls. The **declspecs** add only a single **MOV** instruction to the execution path, while a linker-generated thunking routine adds two instructions: a load from memory, and a branch.

See Also

Concepts

THUMB-enabled ARM Implementation
Linker-Generated Thunking Routines

iw16

iw16

When compiled by the ARM compiler, the **iw16 declspec** has the same effect as the iwcall **declspec**; that is, it makes an interworking call.

When compiled by the THUMB compiler, the **iw16 declspec** has no effect.

Use the **iw16 declspec** only when you know which compiler, ARM or THUMB, will compile the designated function.

iw16 declspec has no effect on the ARM compiler unless you also use the /QRArch - Specify Target Architecture flag.

Using the /QRinterwork-return - Enable Interworking flag automatically enables the **/QRarch** flag.

See Also

Reference

iwcall
iw32
/QRinterwork-return - Enable Interworking
/QRArch - Specify Target Architecture

Concepts

Modifiers

iw32

iw32

When compiled by the THUMB compiler, the **iw32 declspec** has the same effect as the iwcall **declspec**; that is, it makes an interworking call.

When compiled by the ARM compiler, the **iw32 declspec** has no effect.

Use the **iw32 declspec** only when you know which compiler, ARM or THUMB, will compile the designated function.

See Also

Reference

iwcall

iw16

Concepts

Modifiers

iwcall

iwcall

The **iwcall declspec** allows the programmer to select which function calls are interworking, regardless of whether the ARM or THUMB compiler is used.

Using **iwcall** allows you to avoid link-time generation of interworking thunking routines.

Note that **iwcall** has no effect unless it is used on a function prototype visible to the caller.

The **iwcall declspec** does not cause the associated function to have an interworking return. To make an interworking return, use the **-**/QRinterwork-return - Enable Interworking flag.

You can use **__declspec(iwcall)** on the prototype of the user-supplied function to enable the library to handle either case.

The following code example shows how to use of **iwcall declspec** to enable a user-supplied function **myfunction()**.

```
__declspec(iwcall) int myfunction();
int main()
{
  return myfunction();
}
```

The following listing shows the code generated from this **declspec** when compiled with /QRArch - Specify Target Architecture.

```
    str       lr, [sp, #4]!
    ldr       r3, [pc, #8]
    mov       lr, pc
    bx        r3
    ldmia     sp!, {pc}
    DCD       |myfunction|
```

The following listing shows the code generated with the ARM compiler using the /QRthumboption.

```
    push      {lr}
    ldr       r3, [pc, #0x2]
    mov       r12, r3
    bl        __r12_indirect
    pop       {pc}
    DCD       |myfunction|
```

See Also

Reference

/QRinterwork-return - Enable Interworking

/QRArch - Specify Target Architecture

Concepts

Modifiers

Linker-Generated Thunking Routines

Linker-Generated Thunking Routines

If no **declspec** call enables interworking function calls, the linker generates a thunking routine to perform necessary caller-side mode changes from ARM to THUMB or vice versa. The linker generates a thunking routine for each unique function called from an object that requires the opposite mode.

For the required mode switch to occur on the return from the function, the programmer must use the /QRinterwork-return - Enable Interworking flag at compile time.

The following code example shows a linker-generated thunking routine generated when switching from ARM to THUMB mode.

```
0xe59fc000    ldr  r12, [pc]
0xe12fff1c    bx   r12
0x00000000    DCD |destination|
```

The following code example shows the linker-generated thunking routine generated when switching from THUMB to ARM mode.

```
0xb408        push  {r3}
0x4b02        ldr  r3, [pc, #8]
0x469c        mov r12, r3
0xbc08        pop  {r3}
0x4760        bx   r12
0x0000        pad
0x00000000    DCD |destination|
```

See Also

Reference

/QRinterwork-return - Enable Interworking
/QRArch - Specify Target Architecture

Code Size Considerations

Code Size Considerations

Both declspec modifiers and linker-generated thunking routines add additional 32-bit words of text-space instruction.

In general, if a program contains only a few interworking functions, but the interworking functions are called from many places; a linker-generated thunking routine probably results in smaller overall code size.

If a program contains many different interworking functions, but the interworking functions are called from only a few places, using iwcall or iw16 results in code size similar to that required by a linker-generated thunking routine.

If code size efficiency is important, keep in mind the following additional word requirements:

- The **iwcall** and **iw16 declspecs** require an additional 32-bit word of text-space instruction per interworking call. Because the function address must be placed into and loaded from the literal pool, the function address can add up to two 32-bit words of text space instruction to the space required by the linker-generated thunking routine.
  Because different load instructions can share a given literal pool entry, some function calls do not add words for a declspec. The literal pool entry can be loaded into a register that different **BX** instructions use without reloading the register.
- Allowing the linker to generate a thunking routine for the ARM compiler requires three additional 32-bit words per called function but no additional words at the call sites. A linker-generated thunking routine for the THUMB compiler requires four words instead of three.

Note:
Regardless of how many different callers call a function, the linker generates only one interworking thunking routine p er function call.

See Also

Concepts

Linker-Generated Thunking Routines
Modifiers

ARM Assembler

ARM Assembler

The ARM assembler (armasm) is a two-pass assembler, processing its source files twice to reduce the amount of internal state that it needs to keep.

The ARM assembler compiles both ARM and Thumb assembly language into the Microsoft implementation of the Common Object File Format (COFF).

In This Section

ARM Assembler Command-Line Options

Describes the command-line syntax for invoking the ARM Assembler, and describes the command-line options.

Predeclared ARM Register Names

Provides reference information about register names that the ARM Assembler pre-declares.

Built-In ARM Assembler Variables

Provides reference information about variables that have built-in definitions for the ARM Assembler.

ARM Assembler Directives

Describes the assembler directives.

Related Sections

ARM Family Processors

ARM Assembler Command-Line Options

ARM Assembler Command-Line Options

The syntax of a command to invoke the ARM assembler is as follows:

```
armasm {options} sourcefile objectfile
```

The following table shows command-line options for the ARM assembler.

Option	Description
-archarchitecture	Sets the target architecture. Legitimate values are 4, 4T, 5, and 5T. Some microprocessor-specific instructions produce errors or warnings if assembled for the wrong target architecture.
-checkreglist	Checks **LDM** and **STM** register lists to ensure that register number order is increasing. If not, a warning is given. You can use `-checkreglist` to detect misuse of symbolic register names.
-cpu ARMcore	Sets the target ARM core. Legitimate values include the following: • ARM7TM • ARM720t • ARM920t • ARM8 • ARM10 • ARM10T • ARM10200 • StrongARM or StrongARM1 • XSCALE • PXA270 Some microprocessor-specific instructions produce warnings if assembled for the wrong ARM core.
-errors errorfile	Outputs error messages to errorfile.
-help	Displays a summary of the command-line options.
-i dir{,dir}	Specifies directories to include. Directories can also be specified by using the INCLUDE environment variable.
-ignore 0241 -archarchitecture	Instructs the build process to ignore warning message 0241 for an instruction not implemented on the specified target architecture.
-list listingfile-noterse	Turns the terse flag off. When the terse flag is on, lines skipped due to conditional assembly do not appear in the listing. When the terse flag is off, these lines appear in the listing. The default is on.

-list listingfile-width n	Sets the listing page width. The default is 79 characters.
-list listingfile-length n	Sets the listing page length. Length 0 means an unpaged listing. The default is 66 lines.
-list listingfile-xref	Lists cross-referencing information on symbols. Provides definition location and usage location inside and outside of macros. The default is off.
-noesc	Ignores C-style special characters (\n, \t, and so on).
-noregs	Tells the assembler not to predefine implicit register names **r0â€“r15, a1â€“a4, v1â€“v6, c0â€“c15, p0â€–p15, sl, fp, ip, sp, lr, pc**.
-nowarn	Turns off warning messages.
-predefine "directive"	Pre-executes a SETx directive. This implicitly executes a corresponding GBLx directive. Because it contains spaces, place the full SETx argument in quotation marks. For example: `-predefine "Version SETA 44"`
-Via file	Opens *file* and reads in more armasm command-line arguments.

See Also

Reference

Predeclared ARM Register Names

Built-In ARM Assembler Variables

Other Resources

ARM Assembler

Predeclared ARM Register Names

Predeclared ARM Register Names

By default, the assembler predeclares the following register names:

- **R0â€“R15** and **r0â€“r15**
- **c0â€“c15** coprocessor registers
- **p0â€“p15** coprocessor registers
- **a1-a4** scratch registers, synonymous with **r0-r3**
- **v1-v8** variable registers, synonymous with **r4-r11**
- **sb and SB** stack base, synonymous with **r9**
- **sl** and **SL** stack base, synonymous with **r10**
- **fp** and **FP** frame pointer, synonymous with **r11**
- **ip** and **IP** intra-procedure call scratch register, synonymous with **r12**
- **sp** and **SP** stack pointer, synonymous with **r13**
- **lr** and **LR** link register, synonymous with **r14**
- **pc** and **PC** program counter, synonymous with **r15**
- **s0-s32** VFP single precision registers
- **d0-d16** VFP double precision registers
- **fpsid** VFP system ID register
- **fpscr** VFP status and control register
- **fpexc** VFP exception register
- **wr0-wr16** WMMX SIMD data registers
- **wc0-wc16** WMMX status and control registers
- **wcid** WMMX coprocessor ID register, synonymous with **wc0**
- **wcon** WMMX control register, synonymous with **wc1**
- **wcssf** WMMX saturation SIMD flags, synonymous with **wc2**
- **wcasf** WMMX arithmetic SIMD flags, synonymous with **wc3**
- **wcgr0-wcgr3** WMMX control general purpose registers, synonymous with **wc8-wc11**

See Also

Reference

Built-In ARM Assembler Variables

Other Resources

ARM Assembler

Built-In ARM Assembler Variables

Built-In ARM Assembler Variables

The ARM assembler contains built-in variables for programmer convenience.

The following table shows the built-in variable definitions.

Variable	Description
{PC} or .	Contains the current value of the program location counter.
{VAR} or @	Contains the current value of the storage area location counter.
{TRUE}	Contains the logical constant true.
{FALSE}	Contains the logical constant false.
{OPT}	Contains the value of the currently set listing option. The OPT directive can be used to save the current listing option, force a change in it, or restore its original val ue.
{CONFIG}	Has the value 32 if the assembler is in 32-bit program counter mode, and the value 26 if it is in 26-bit mode.
{ENDIAN}	Has the value "big" if the assembler is in big-endian mode and the value "little" if it is in little-endian mode.
{CODESIZE}	Has the value 16 if compiling Thumb code. Otherwise, it is 32.
{CPU}	Has the name of the selected CPU or generic ARM if no CPU is specified.
{ARCHITECTURE }	Has the value of the selected ARM architecture: 4 or 4T.

See Also

Reference

Predeclared ARM Register Names

Other Resources

ARM Assembler

ARM Assembler Directives

ARM Assembler Directives

The ARM assembler supports a full array of assembler directives that allow you to control and direct source file assembly.

In This Section

ARM Initialization and Layout Directives

ARM Linking Directives

ARM Diagnostic Directives

ARM Directives for Conditional Assembly

ARM Dynamic Listing Directive Options

ARM-Thumb Interworking Directives

ARM Constant and Variable Declarations

ARM Assembler Directives for Macro Definition

Miscellaneous ARM Directives

Related Sections

ARM Assembler Command-Line Options

Predeclared ARM Register Names

Built-In ARM Assembler Variables

ARM Assembler Error Messages

ARM Initialization and Layout Directives

ARM Initialization and Layout Directives

The AREA directive instructs the assembler to assemble a new code or data sections, used in an expression of the following form:

```
AREA sectionname{,attr}{,attr}...
```

The AREA *sectionname* expression is modified by one or more comma-delimited attributes.

The following table shows the valid attributes for an AREA directive.

Attribute	Description
ALIGN	Defines the section boundary, where the section is aligned on a $2^{expression}$ -byte boundary. *expression* can have any integer value from 0 to 31. Do not use `ALIGN=0` or `ALIGN=1` for code sections. Note that the `ALIGN` attribute is not the same as the `ALIGN` directive.
CODE	Contains machine instructions. `READONLY` is the default.
DATA	Contains data, not instructions. `READWRITE` is the default.
NOINIT	Indicates that the data section is uninitialized, or initialized to zero. It contains only space reservation directives `SPACE` or `DCB`, `DCD< DCDU`, `DCQ`, `DCQU`, or `DCWU` with initialized values of zero.
READONLY	Indicates that this section should not be written to. This is the default for `CODE` areas.
READWRITE	Indicates that this section can be read from and written to. This is the default for `DATA` areas.

See Also

Reference

ARM Initialization and Layout Directives

ARM Linking Directives

ARM Linking Directives

The following ARM assembly directives link other files to the current assembly.

Directive	Syntax	Description
EXPORT	`EXPORT symbol{[FPREGARGS,DATA,LEAF]}`	Declares a *symbol* for use at link time by other, separate object files. *FPPREGARGS* defines a function that expects fp arguments passed in fp registers. *DATA* defines a code-segment datum rather than a function or procedure. *LEAF* causes asserts that call other functions. Identical to the `GLOBAL` directive.
GET	`GET filename`	Includes an additional file named *filename* within the current file assembly. The included file can in turn use `GET` directives to include more files. When assembly of the included file is complete, assembly continues at the line following the **GET** directive.
GLOBAL	`GLOBAL symbol{[FPREGARGS,DATA,LEAF]}`	Declares a *symbol* for use at link time by other, separate object files. *FPPREGARGS* defines a function that expects fp arguments passed in fp registers. *DATA* defines a code-segment datum rather than a function or procedure. *LEAF* causes asserts that call other functions. Identical to the `EXPORT` directive.
IMPORT	`IMPORT symbol{WEAK=weak-symbol, {type=n}}`	Provides the assembler with a name, *symbol*. The assembly does not contain a definition for *symbol*, but resolves it at link time to a symbol defined in a separate object file. The routine treats *symbol* as a program address.
INCLUDE	`INCLUDE filename`	Specifies a synonym for `GET`.

See Also

Other Resources

ARM Assembler Directives

ARM Diagnostic Directives

ARM Diagnostic Directives

The following ARM assembly directives generate diagnostic information.

Directive	Syntax	Description
ASSERT	`ASSERT logical-expression`	Supports diagnostic generation. If *logical-expression* returns FALSE, the assembler generates a diagnostic message during the second pass of the assembly. `ASSERT` can be used inside and outside macros.
!	`! arithmetic-expression, string-expression`	Related to `ASSERT`, but is inspected on both passes of the assembly, providing a more flexible means for creating custom error messages. If *arithmetic-expression* equals 0, the assembler takes no action during pass one, but prints *string-expression* as a warning during pass two. If *arithmetic-expression* does not equal 0, the assembler prints *string-expression* as a diagnostic message. The assembly halts after pass one. This directive is identical to the `INFO` directive.
INFO	`INFO arithmetic-expression, string-expression`	If *arithmetic-expression* equals 0, the assembler takes no action during pass one. It adds source file and line number as a prefix to *string-expression,* and prints the result as a warning during pass two. If *arithmetic-expression* does not equal 0, the assembler prints *string-expression* as a diagnostic message. The assembly halts after pass one. This directive is identical to the `!` directive.

See Also

Other Resources

ARM Assembler Directives

ARM Directives for Conditional Assembly

ARM Directives for Conditional Assembly

The ARM assembler supports conditional assemblies for sections of a source file; that is, it assembles the specified sections if certain conditions are true.

In addition, the assembler supports WHILE...WEND directives for conditional looping that are useful for generating repetitive tables.

Conditional looping produces an assembly-time loop, not a run-time loop. Because the test for the WHILE condition is made at the top of the loop, it is possible that no code is generated during assembly; lines are listed as for conditional assembly.

The following ARM assembler directives are required for conditional and repetitive assembly.

[or IF

Marks the start of the condition.

] or ENDIF

Marks the end of the condition.

| or ELSE

Provides an else construct.

WHILE

Marks the start of the repetitive condition.

WEND

Marks the end of the repetitive condition.

Note that the characters [, |, and] cannot be the first character in a line. These characters must be preceded by a space or a tab.

The following code shows the syntax for conditional assembly.

```
[ logical-expression
...code...
|
...code...
]
```

The following code shows the syntax for repetitive assembly.

```
WHILE logical-expression
...code...
WEND
```

See Also

Other Resources

ARM Assembler Directives

ARM Dynamic Listing Directive Options

ARM Dynamic Listing Directive Options

The **OPT** directive is used to set listing options from within the source code, if listing is turned on.

The default setting is to produce a normal listing, including the declaration of variables, macro expansions, call-conditioned directives, and MEND directives, but without producing a listing during the first pass.

These settings can be altered by adding the appropriate values from the list and using them with the **OPT** directive, as shown in the following table.

1

Turns on normal listing.

2

Turns off normal listing.

4

Page throw: issues an immediate form feed and starts a new page.

8

Resets the line number counter to 0.

16

Turns on the listing of SET, GBL, and LCL directives.

32

Turns off the listing of SET, GBL, and LCL directives.

64

Turns on the listing of macro expansions.

128

Turns off the listing of macro expansions.

256

Turns on the listing of macro calls.

512

Turns off the listing of macro calls.

1024

Turns on the first pass listing.

2048

Turns off the first pass listing.

4096

Turns on the listing of conditional directives.

8192

Turns off the listing of conditional directives.

16384

Turns on the listing of MEND directives.

32768

Turns off the listing of MEND directives.

See Also

Other Resources

ARM Assembler Directives

ARM-Thumb Interworking Directives

ARM-Thumb Interworking Directives

The following table shows the ARM-THUMB interworking directives.

CODE16

Tells the assembler that subsequent instructions are to be interpreted as 16-bit (Thumb) instructions.

CODE32

Tells the assembler that subsequent instructions are to be interpreted as 32-bit (ARM) instructions.

DATA

Tells the assembler that the label is a "data-in-code" label; that is, that it defines an area of data within a code segment. You must specify this directive if you are defining data in a Thumb code area.

See Also

Other Resources

ARM Assembler Directives

ARM Constant and Variable Declarations

ARM Constant and Variable Declarations

The following ARM Assembly directives are for setting constants.

Directive	Syntax	Description
*	`label *expression`	Identical to EQU. Gives a symbolic label to a fixed or program-relative expression.
CN	`label CN numeric-expression`	Names a coprocessor register number; c0 to c15 are predefined and cannot be used as labels.
CP	`label CP numeric-expression`	Gives a name to a coprocessor number, if available, that must be within the range 0 to 15. The names p0 - p15 are predefined and cannot be used as labels.
EQU	`label EQU expression`	Gives a symbolic label to a fixed or program-relative expression.
FN	`label FN numeric-expression`	Defines the names of floating-point registers, if available. The names F0-F7 and f0-f7 are predefined. The predefined register names cannot be used as labels but can be used as numeric expressions.
RN	`label RN numeric-expression`	Defines register names. Refer to registers by name only. The names R0-R15, r0-r15, PC, pc, LR, and lr are predefined. The predefined register names cannot be used as labels but can be used as numeric expressions.

The assembler supports global and local variables. The scope of global variables extends across the entire source file, while that of local variables is restricted to a particular instantiation of a macro.

The following table ARM assembly directives are for setting local and global variables.

Directive	Syntax	Description
GBLA	`GBLA variable-name`	Defines a global arithmetic variable. Values of arithmetic variables are 32-bit unsigned integers.
GBLL	`GBLL variable-name`	Defines a global logical variable.
GBLS	`GBLS variable-name`	Defines a global string variable.
LCLA	`LCLA variable-name`	Defines a local arithmetic variable with an initial state of 0.
LCLL	`LCLL variable-name`	Defines a local logical variable with an initial state of FALSE.

LCLS	`LCLS variable-name`	Defines a local string variable with an initial state of NULL string.
SETA	`variable-name SETA expression`	Sets the value of an arithmetic variable.
SETL	`variable-name SETL expression`	Sets the value of a logical variable.
SETS	`variable-name SETS expression`	Sets the value of a string variable.

Note that when you set the value of a string variable, you must use quotes.

You can declare local variables only from within a macro. In addition, after you declare a variable, you cannot use its name for any other purpose.

The assembler substitutes values for some variables:

- If a variable name has a $ character prefix, the assembler substitutes the variable value before it checks the line syntax.
- If the variable is a logical or arithmetic variable, the assembler performs an .STR operation on the variable, and replaces the variable with the result of the operation.

See Also

Other Resources

ARM Assembler Directives

ARM Assembler Directives for Macro Definition

ARM Assembler Directives for Macro Definition

Macros are useful when you need a group of instructions or directives frequently.

The ARM assembler replaces the macro name with its definition.

Macros can contain calls to other macros, nested up to 255 levels.

Two directives define a macro, **MACRO** and **MEND**.

```
MACRO
{$label}Â Â Â  macroname {$parameter1}{,$parameter2}{,$parameter3}..
...code...
MEND
```

Remarks

A macro prototype statement must appear on the first line following the **MACRO** directive. The prototype tells the assembler the name of the macro, *macroname,* and its parameters.

A label is optional, but is useful if the macro defines internal labels.

Any number of parameters can be used. Each must begin with $ to distinguish it from ordinary program symbols.

Within the macro body, *$label, $parameter,* and so on, can be used in the same way as other variables.

The *$label* parameter is treated as another parameter to the macro. The macro describes which labels are defined where. The label does not represent the first instruction in the macro expansion.

For example, it is useful in a macro that uses several internal labels, such as a loop, to define each internal label as the base label with a different suffix.

Sometimes, a value appends a macro parameter or label. Separate the appended value by a dot. After the assembler recognizes the end of the parameter and label, the assembler ignores the dot. For example:

```
$label.1
$label.loop
$label.$count
```

Default values can be set for parameters by following them with an equal sign and the default value. If the default has a leading or trailing space, the whole value should appear in quotes, as shown in the following code example.

```
...{$parameter="default value"}
```

The **MEND** directive signifies the end of the macro definition. If the macro contains WHILE/WEND loops, or contains conditional assembly, the WHILE/WEND loop must close before execution reaches the **MEND** directive.

You can also terminate macro expansion with the **MEXIT** directive.

See Also

Other Resources

ARM Assembler Directives

Miscellaneous ARM Directives

Miscellaneous ARM Directives

The following table describes miscellaneous directives for the ARM assembler.

Directive	Syntax	Description
ALIGN	`ALIGN {power-of-two {,offset-expression}}`	Sets the instruction location to the next word boundary. The optional power-of-two parameter can be used to align with a coarser byte boundary and the offset-expression parameter to define a byte offset from that boundary.
END	`END`	Stops the processing of a source file. If a GET directive invoked assembly of the file, the assembler returns and continues after the GET directive. If the assembler reaches an END directive in the top-level source file during the first pass and there are no errors, the second pass begins. Absence of an END directive causes an error.
KEEP	`KEEP {symbol}`	Retains a local symbol in the assembler's symbol table. By default, the assembler does not describe local symbols in its output object file. If the KEEP directive appears without a symbol parameter, the assembler keeps all symbols. To keep a specific symbol, specify it by name.
LTORG	`LTORG`	Directs the assembler to assemble the current literal pool that immediately follows the assembly. A default LTORG is executed at every END directive that is not part of a nested assembly. Large programs might need several literal pools, each closer to the location of their literals to avoid violating LDR's 4-KB offset limit.
NOFP	`NOFP`	Disables floating point instructions. In some circumstances, there is no support in the target hardware or software for floating-point instructions.
RLIST	`label RLIST list-of-registers`	Assigns a name to a set of registers to be transferred by LDM or STM. List-of-registers is a comma-separated list of register names or ranges enclosed in braces. If you select the -CheckReglist option, you must also supply a List-of-registers list, in increasing order of register number.
ROUT	`{name}ROUT`	Marks the boundaries of the scope of local labels. name indicates the name to be assigned to the scope. Use ROUT to limit the scope of local labels. This makes it easier for you to avoid referring to a wrong label by accident. The scope of local labels is the whole area if there are no ROUT directives in it.
SUBT	`SUBT title`	Specifies a subtitle to be inserted on all pages until a new subtitle appears.

TTL	`TTL title`	Specifies a title to be inserted on all pages until a new title appears.

See Also

Other Resources

ARM Assembler Directives

ARM Assembler Error Messages

ARM Assembler Error Messages

This section contains descriptions of the ARM Assembler error messages.

In This Section

ARM Error Messages 1-50

ARM Error Messages 1-50

The following table shows ARM error messages 1-150.

No.	Error Message	Warning/Error	Arguments
1		Error	None
2	:CHR: operator: truncation applied to a char	Warning	None
3	missing substitution operator $ for assembler variable %s	Warning	None
4	reserved symbol used in an expression: %s	Warning	Symbol name
5	improper line syntax: %s	Error	Line with improper syntax
6	improper program syntax: %s	Error	Description of error
7	unimplemented class of instructions	Error	None
8	ARM register expected: %s	Error	String where the assembler tried to read a register
9	coprocessor number expected: %s	Error	String where the assembler tried to read a coprocessor number
10	ARM coprocessor register expected: %s	Error	String where the assembler tried to read a coprocessor register
11	ARM floating point register expected: %s	Error	String where the assembler tried to read a floating-point register
12	wrong operand type for %s	Error	String with operand
13	divide by zero	Error	None
14	%s cannot be applied; different sections or different base registers	Error	Text of operand
15	illegal symbol %s; AREA name expected	ErrorÂ	None
15	illegal symbol %s; AREA name expected	Error	Symbol name
16	illegal flag(s) %s	Error	Token
17	illegal $ substitution, non assembler variable	Error	None
18	wrong character constant	Error	None
19	malformed string constant; missing closing	Error	None
19	malformed string constant; missing closing	Error	None

20	wrong escape constant; null value not permitted inside a string	Error	None
20	wrong escape constant; null value not permitted inside a string	Error	None
21	unexpected char	Error	None
22	wrong :: operator	Error	None
23	special bar id not closed	Error	None
24	wrong built-in variable	Error	None
25	wrong shift name	Error	None
26	unimplemented instruction set	Error	None
27	unaligned absolute value	Warning	None
28	truncation applied, absolute value should be word aligned: 0x%x	Warning	Offset value
29	truncation applied, absolute value should be half-word aligned: 0x%x	Warning	Offset value
30	address 0x%x is too large to represent in %d bits	Error	Address
31	address out of range in a different section	Error	None
32	immediate value %d cannot be represented in %d bits	Error	Offset
33	floating value %e cannot be represented in single precision	Error	Floating-point value
34	undefined symbol: %s	Error	Symbol text
35	illegal expression type: %s	Error	Expression value
36	<%s> mnemonic qualifier is ignored	Warning	Instruction suffix
37	operand restriction violation (undefined behavior): %s	Warning	Instruction operand
38	multiple use of the same register in a register list	Warning	None
39	unsafe use of R15 as base register	Warning	None
40	unsafe use of write-back operator	Warning	None
41	unknown LDM/STM operation	Error	None
42	thumb register expected; %s	Error	Unexpected text
43	multiple symbol definition or incompatibility: %s	Error	Symbol name

44	cannot open file: %s	Error	File name
45	possible infinite loop due to recursive file inclusion: %s	Warning	File name
46	unknown command-line argument or argument value %s	Warning	Unknown argument
47	command-line option not implemented; %s ignored	Warning	Command-line argument
48	multiple use of the same option: %s	Warning	Command-line argument
49	only one source file can be assembled: %s	Error	Source file name
50	missing input source file	Error	None

See Also

Other Resources

ARM Assembler Error Messages

ARM Error Messages 51-100

ARM Error Messages 51-100

The following table shows ARM error messages 51-100.

No.	Error Message	Warning/ Error	Arguments
51	unknown opcode: %s	Error	String for OPCODE
52	minimum requested width of 34 assumed	Warning	None
53	wrong page length: %d, 0 (non paged) assumed	Warning	Page length
54	failed to close file %s	Error	File name
55	illegal line start symbol: %s for directive: %s	Error	Start symbol, directive
56	unimplemented directive %s	Warning	Directive name
57	%s attribute does not pertain to a relocatable module; ignored	Warning	Directive name
58	%s directive does not pertain to a relocatable module; ignored	Warning	Directive name
59	unknown section flag: %s	Error	Area name
60	illegal combination of section flags: %s	Warning	Section name and flags
61	redefinition of section flags ignored	Warning	None
62	ignored lines after END directive	Warning	None
63	missing END directive	Warning	None
64	code inside data section	Error	None
65	unknown or bad symbol type: %s	Error	Symbol
66	out of memory: section data: no object file will be generated	Error	None
66	out of memory: section data: no object file will be generated	Error	None
67	initialized data in an uninitialized data section %s	Error	None
68	assembler internal error; sync. error on local labels	Error	None
68	assembler internal error; sync. error on local labels	Error	None
69	error while generating object file: seek error	Error	None
69	error while generating object file: seek error	Error	None
70	input error: line is too long after expanding an assembler variable; truncated	Error	None

70	input error: line is too long after expanding an assembler variable; truncated	Error	None
71	macro error: END directive cannot appear inside a macro	Error	None
71	macro error: END directive cannot appear inside a macro	Error	None
72	wrong directive usage: local assembler variables can only be defined within a macro	Error	None
73	wrong local label reference; reference expected after %% (usage: %%{x}{y}n{name})	Error	None
74	improper argument for <-predefine> command-line option: %s; %s	Error	None
75	non increasing order in the register list	Warning	None
76	the selected architecture and CPU (or mode) are conflicting: %s versus %s	Warning	Architecture, CPU
77	could not delete %s file	Error	File name
78	missing ENDP directive in section %s	Error	Section name
79	%s	Error	Internal use
80	%s %s	Error	Internal use
81	error	Error	None
82	warning	Warning	None
83	illegal expression type; expected absolute numeric	Error	None
84	illegal expression type; expected absolute numeric or program relative	Error	None
85	illegal expression type; expected absolute numeric, program relative or register relative	Error	None
86	illegal expression type; expected program relative or register relative	Error	None
87	illegal expression type; expected program relative	Error	None
88	illegal expression type; expected absolute numeric or string	Error	None
89	illegal expression type; expected absolute numeric or Boolean	Error	None
90	illegal expression type; expected bool	Error	None
91	illegal expression type; expected string	Error	None
92	no immediate rotate operand can be created: %d	Error	Values to rotate
93	immediate value %d out of range; expected values: %s	Warning	Immediate value, string contain expected values

94	improper alignment value %d; expected: [1, 2, 4, 8,Â Â Â ... 32768]	Error	Value
95	improper alignment value %d; alignment offset should be positive and smaller than alignment; truncated	Warning	Value
96	immediate value %d out of range; positive value expected	Error	Immediate value
97	immediate value %d out of range; truncated to 16 bits	Warning	Immediate value
98	immediate value %d out of range; truncated to 8 bits	Warning	Immediate value
99	literal pool too far, or too close: %d; address is 8 bit offset; use LTORG	Error	None
100	literal pool too far, or too close: %d; address is 12 bit offset; use LTORG	Error	None

See Also

Other Resources

ARM Assembler Error Messages

ARM Error Messages 101-150

ARM Error Messages 101-150

The following table shows ARM error messages 101-150.

No.	Error Message	Warning/Error	Arguments
101	improper shift amount: %d; must fit in 5 bits	Error	Immediate value
102	improper rotate amount: %d; rotate amount shifted right by 1 must fit 4bits	Error	Immediate value
103	improper rotate amount: %d; only even rotate amounts can be specified	Error	Immediate value
104	immediate value out of range: %d; coprocessor information must fit in 3 bits	Error	Immediate value
105	immediate value out of range: %d; coprocessor opcode must fit in 4 bits	Error	Immediate value
106	internal assembler error: unused error code	Error	None
107	improper line syntax, only <ldr> can be used to generate literals	Error	None
108	improper line syntax	Error	None
109	improper line syntax; EOL or comma expected	Error	None
110	improper line syntax; wrong offset	Error	None
111	improper line syntax; expected: %s	Error	Expected tokens for proper syntax
112	improper line syntax; SP or PC required as second register in this format	Error	None
113	improper line syntax; high registers are not accepted for SUB instruction	Error	None
114	improper line syntax; only SP register accepted for this format	Error	None
115	improper line syntax; unexpected comma	Error	None
116	improper line syntax; high registers not allowed in this context	Error	None

117	improper line syntax; wrong combination of mnemonic and operands	Error	None
118	improper line syntax; write back '!', expected (assumed)	Error	None
119	improper line syntax; opcode, directive or macro call expected	Error	None
120	improper line syntax; wrong use of a local label	Error	None
121	improper line syntax; %s directive cannot define a starting line symbol	Error	Directive name
122	improper line syntax; symbol expected	Error	None
123	improper line syntax; missing weak alias; symbol defined as external	Warning	None
124	improper line syntax; include file name expected	Error	None
125	improper line syntax; string expected	Warning	None
126	improper line syntax; %s option requires %s section name	Warning	Option, section name
127	improper line syntax; usage: %s	Warning	Line
128	improper line syntax; selection must be 5 if <assoc= ..> option is used	Warning	None
129	improper line syntax; bad macro call; macro definition does not have a label parameter	Warning	None
130	improper line syntax; macro call with too many actual parameters	Warning	None
131	improper line syntax; malformed macro call	Warning	None
132	improper line syntax; register list symbol or proper register list ({ .. }) expected	Error	None
133	improper line syntax; malformed register list construction	Error	None
134	improper line syntax;	Error	None

135	improper line syntax;	Error	None
136	improper line syntax;	Error	None
137	unexpected end of file in an include file/macro or missing END directive	Error	None
138	improper program syntax; missing ENDP directive or nested function definition	Warning	None
139	improper program syntax; unexpected ENDP directive	Warning	None
140	improper program syntax; unexpected MEND/MEXIT directive	Warning	None
141	improper program syntax; conditional if structure crosses file (include) structure	Warning	None
142	improper program syntax; (malformed/not closed) conditional if structure appears	Error	None
143	improper program syntax; end of file or macro in WHILE loop, or missing WEND	Error	None
144	improper program syntax; a WEND directive without a preceding WHILE	Error	None
145	improper program syntax; unbalanced WHILE-WEND construction, possibly due to conditionals and includes	Warning	None
146	improper program syntax; an ENDIF directive without a corresponding IF	Error	None
147	improper program syntax; an ELSE directive without a corresponding IF	Error	None
148	improper program syntax; malformed conditional structure, possibly two ELSE key words in an IF construct	Error	None
149	improper program syntax; nested macro definitions are not supported	Error	None
150	improper program syntax; read error or missing macro definition line	Error	None

See Also
Other Resources
ARM Assembler Error Messages

ARM Error Messages 151-200

ARM Error Messages 151-200

The following table shows ARM error messages 151-200.

No.	Error Message	Warning/Error	Arguments
151	improper program syntax; MEND, MEXIT directives do not have any label or prefixing text	Warning	None
152	improper program syntax; read error, missing MEND directive or end of file inside a macro definition	Error	None
153	improper program syntax; malformed macro definition (unrecoverable)	Error	None
154	improper program syntax; associated COMDAT section not found	Error	None
155	improper program syntax; attempt to redefine the associated COMDAT section; ignored	Error	None
156	routine name does not match the enclosing routine structure	Error	None
157	improper argument for <-predefine> command-line option: name expected	Error	None
158	improper argument for <-predefine> command-line option: %s; symbol name expected	Error	Command-line option
159	improper argument for <-predefine> command-line option: SETx expected	Error	None
160	improper argument for <-predefine> command-line option; undefined symbol in expression	Error	None
161	improper argument for <-predefine> command-line option: malformed or unrecognized expression	Error	None
162	improper argument for <-predefine> command-line option: expression type mismatch	Error	None
163	assembler internal error; Sync. error on ROUT areas	Error	None
164	assembler internal error; symbol sync. error in function symbol line numbers	Error	None
165	assembler internal error; address sync. error in function symbol line numbers	Error	None
166	assembler internal error; INCLUDE path cannot be created, check INCLUDE env. variable and <-i> command-line option	Warning	None

167	assembler internal error; source input sync. error; bad use of include or macros	Error	None
168	assembler internal error; bad input stack	Error	None
169	assembler internal error; bad macro structure in token input	Error	None
170	assembler internal error; section raw data area exceeded	Error	None
171	assembler internal error; PC sync error while generating literal pool	Error	None
172	ROUT tag of a local label does not match the enclosed ROUT name	Error	None
173	unknown command-line argument or argument value; error file name expected	Error	None
174	unknown command-line argument or argument value; include path expected	Error	None
175	unknown command-line argument or argument value; list file name expected	Error	None
176	unknown command-line argument or argument value; obj file name expected	Error	None
177	unknown command-line argument or argument value; width value expected	Error	None
178	unknown command-line argument or argument value; length value expected	Error	None
179	unknown command-line argument or argument value; SET directive expected	Error	None
180	unknown command-line argument or argument value; <-via> file name expected	Error	None
181	wrong string constant; $ character must be doubled	Error	None
182	attribute does not pertain to a relocatable module; ignored, weak expected	Error	None
183	multiple symbol definition or incompatibility; weak alias must be external	Error	None
184	multiple symbol definition or incompatibility; procedure name must have address 0 if it is an area COMDAT symbol	Error	None

185	multiple symbol definition or incompatibility; multiple definitions of section COMDAT symbol as function name	Error	None
186	multiple symbol definition or incompatibility; an assembler variable cannot be exported; export attribute deleted	Error	None
187	multiple symbol definition or incompatibility; a register relative symbol cannot be exported; export attribute deleted	Error	None
188	multiple symbol definition or incompatibility; program counter symbol cannot be exported; export attribute deleted	Error	None
189	multiple symbol definition or incompatibility; a string symbol cannot be exported; export attribute deleted	Error	None
190	multiple symbol definition or incompatibility; truncation possible on a float absolute exported symbol	Error	None
191	illegal symbol %s; a built-in variable cannot be used as a COMDAT symbol; ignored	Error	Symbol
192	illegal symbol %s; COMDAT symbol name expected	Warning	Symbol
193	illegal symbol %s; AREA attribute expected	Error	Symbol
194	illegal symbol %s; expected : '='	Error	Symbol
195	illegal symbol %s; has no type	Error	Symbol
196	illegal symbol %s; wrong type	Error	Symbol
197	illegal symbol %s; absolute value expected	Error	Symbol
198	MEND directive before a proper macro definition	Error	None
199	illegal symbol %s; bad macro name	Error	Symbol
200	illegal symbol %s; < argument > expected	Error	Symbol

See Also
Other Resources
ARM Assembler Error Messages

ARM Error Messages 201-250

ARM Error Messages 201-250

The following table shows ARM error messages 201-242.

No.	Error Message	Warning/Error	Arguments
201	illegal symbol %s; null macro formal parameter	Error	Symbol
202	illegal symbol %s; initialization string expected	Error	Symbol
203	illegal macro syntax: %s; comma or end of line expected	Error	Line
204	illegal macro syntax: %s; improper component of a macro definition	Error	Line
205	illegal symbol %s; macro name multiply defined	Error	Symbol
206	illegal symbol %s; unknown token	Error	Symbol
207	illegal symbol %s; duplicate formal parameter; ignored	Warning	Symbol
208	wrong expression operand: %d; positive value expected	Error	Value in expression
209	built-in variable not implemented yet	Error	None
210	error while reading input file: input stack overflow, possible infinite recursion on file include or macro calls	Error	None
211	error while reading input file: seek error	Error	None
212	error while reading input file: line too long on input file	Error	None

213	error while reading input file: macro expanded line too long	Error	None
214	error while reading input file: input stack overflow, possible infinite recursion on file include or macro calls	Error	None
215	malformed escape constant; too many characters	Error	None
216	malformed constant; unknown escape sequence	Error	None
217	error while generating object file: write error	Error	None
218	assertion failed	Error	None
219	the selected architecture and CPU (or mode) are conflicting %s; BX instruction not defined for this architecture	Error	None
220	the selected architecture and CPU (or mode) are conflicting; <Thumb Area option> and <command-line arch: not 4T>	Warning	None
221	illegal flag(s); only <cpsr_f> or <spsr_f> can be used with immediate value	Warning	None
222	operand restriction violation (undefined behavior): same source and dest reg	Warning	None
223	operand restriction violation (undefined behavior): same base and offset reg	Warning	None
224	mnemonic qualifier is ignored; suffix following <ldr>	Warning	None
225	command-line option not implemented; %s assumed; %s ignored	Warning	Command-line argument that is assumed, or that is not implemented
226	illegal combination of section flags: section flags cannot be inferred, code and data/uninitialized, readonly/readwrite	Warning	None

227	syntax error in expression	Error	None
228	assembler internal error: stack overflow in expression	Error	None
229	expression; %s	Error	Expression string
230	condition codes are not accepted for this form of blx instruction; cc ignored	Warning	None
231	unknown floating point system register: %s; expected: FPSID, FPSCR or FPEXC	Error	String with floating register
232	wrong floating point register list; a compact group of registers is expected	Error	None
233	the maximum amount of transfer is 16 words	Warning	None
234	improper line syntax; expected comma	Error	None
235	Improper syntax: s must follow condition code	Error	None
236	fldmdb|fstmdb require increment flag on first register	Error	None
237	The destination register must be even	Warning	None
238	The use of r15 (pc) has unpredictable results	Warning	None
239	Accumulator values greater than zero not supported	Warning	None
240	cpu argument (%s) not supported	Error	Argument

241	Instruction %s not supported for -cpu %s	Error	None
242	-cpu option overriding -arch option	Warning	None
243	The following usage is no longer supported : `armasm [<options>] sourcefile objectfile\n` Please use the ""-o"" option: `armasm [<options>] -o objectfile sourcefile\n`	Warning	None
244	Use of undefined symbol %s in EQU expression	Warning	Symbol
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.	Error	None

See Also
Other Resources
ARM Assembler Error Messages

Renesas Family Processors

Renesas Family Processors

Renesas Technology designs and manufactures high-performance, power-efficient RISC microprocessors.

The Renesas SuperH SH-4 RISC core uses a fixed 16-bit instruction length for improved efficiency, load-store architecture with 16 32-bit general-purpose registers, and a 5-stage pipeline that executes one instruction per clock cycle.

Renesas optimizes each processor family for specific applications.

In This Section

Intrinsic Functions for Renesas Microprocessors

Provides reference information about intrinsic functions.

Renesas Compiler Options

Provides reference information about compiler and linker options available only to Renesas microprocessors.

Renesas SH-4 Calling Sequence Specification

Provides information about assemblers, register assignments, and stack layout.

Related Sections

Differences Between Desktop and Device Compilers

Provides a comparative reference for compiler options and intrinsic functions.

Intrinsic Functions for Renesas Microprocessors

Intrinsic Functions for Renesas Microprocessors

The Renesas SH-4 microprocessor includes a number of functions that you can use intrinsically for faster programs. These functions aid mathematical computations.

Programs that use intrinsic functions do not have the overhead of function calls, but they can be larger because of the additional code created.

To replace functions with their intrinsic (inline) forms, use the /Oi (Generate Intrinsic Functions) option or the **#pragma** intrinsic.

The following table shows SH-4 microprocessor assembly language functions implemented as intrinsic functions.

SH-4 instruction	**Description**
_Convolve	Computes the summation of two vectors.
_Dot3dVW0	Computes the inner product of a pair of three- or four-dimensional vectors with the "w" coordinate forced to 0.
_Dot3dVW1	Computes the inner product of a pair of three- or four-dimensional vectors with the "w" coordinate forced to 1.
_Dot4dV	Computes the inner product of a pair of four-dimensional vectors.
__fmac	Multiplies two float values and adds to a third float value.
_LoadMatrix	Loads a 4x4 matrix into the extended floating-point register bank.
__movca	Moves with cache block allocation.
_Multiply4dM	Multiplies a 4x4 matrix by a 4x4 matrix.
_SaveMatrix	Stores the extended floating-point register bank into a 4x4 matrix.
_XDMultMatrix	Multiplies a 4x4 matrix by a 4x4 matrix.
_XDXform3dV	Multiplies a three-dimensional vector by a 3x3 matrix.
_XDXform4dV	Multiplies a four-dimensional vector by a 4x4 matrix.
_Xform3dV	Multiplies a three-dimensional vector by a 3x3 matrix.
_Xform4dV	Multiplies a four-dimensional vector by a 4x4 matrix.

See Also

Other Resources

Renesas Family Processors

__fmac

__fmac

Multiplies two float values and add to a third float value.

```
Float __fmac(
  float Arg1,
  float Arg1,
  float Arg1
);
```

Parameters

Arg1

[in] First operand of the multiplication operation.

Arg2

[in] Second operand of the multiplication operation.

Arg3

[in] Value to which the product of *Arg1*and *Arg2* is added.

Return Values

The sum of the product of *Arg1* and *Arg2* with *Arg3*.

Remarks

The following code example shows how to use **_fmac**.

```
/*********************************************************/
#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);
}
```

This example results in the following output.

```
sum = 64.000000
```

Requirements

Routine	Required header	Architecture
__fmac	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors

__movca

__movca

Move with cache block allocation.

```
void __movca(
  unsigned long value,
  unsigned long* addr
);
```

Parameters

value

[in] Specifies longword to store in memory.

addr

[in] Address in memory to be accessed.

Return Values

None.

Remarks

The following code example shows how to use _prefetch with **_moveca**.

```
#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");
}
```

This example results in the following output.

```
before prefetch addr value = ffff
after prefetch addr value = ffff
after movca addr value = 100
```

Requirements

Routine	Required header	Architecture

__movca	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors

_prefetch

_Convolve

_Convolve

Computes the summation of two vectors.

```
Float _Convolve(
  int nelement,
  float* pstart,
  float* pend,
  float* pdata,
  float* pfilter
);
```

Parameters

nelement

[in] Number of elements to be processed.

pstart

[in] Pointer to the beginning of data buffer.

pend

[in] Pointer to the end of data buffer.

pdata

[in] Pointer to the current data buffer.

pfilter

[in] Pointer to the filter buffer.

Return Values

The summation of two vectors.

Remarks

The *pdata* parameter can point to any position in the data buffer. The *pfilter* parameter can point to any position in the filter buffer. The *nelement* parameter must not exceed *pfilter+nelement* buffer size.

To implement this function, use the -Qsh4 -Oi flag when compiling.

The following code example shows how to use **_Convolve.**

```
/*************************************************************
_Convolve:  summation of two vector

pstart   | A0 |      pfilter   | X0 |
         | A1 |            | X1   |
         | .. |            | .. |
pdata    | Ai |            | .. |
         | .. |            | .. |
pend     | An |            | Xn |
```

For example:

```
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);

}
```

Output

```
output = 35.000000
output = 45.000000
output = 50.000000
output = 50.000000
output = 45.000000
```

Requirements

Routine	Required header	Architecture
_Convolve	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors

_Dot3dVW0

_Dot3dVW0

Computes the inner product of a pair of three or four-dimensional vectors with the "w" coordinate forced to 0.

```
float _Dot3dVW0(
  float* vector1,
  float* vector2
);
```

Parameters

Vector1

[in] Pointer to a three or four-dimensional source vector.

Vector2

[in] Pointer to a three or four-dimensional destination vector.

Return Values

The scalar resulting from the inner product.

Remarks

The fourth coordinate of a source vector will always force to 0.

To implement this function, use the /Qsh4 /Oi (Generate Intrinsic Functions) flag when compiling.

The following code example shows how to use **Dot3dVW0** to compute the inner products of three or four dimensional vectors.

```
/*********************************************************/
#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);
}
```

This example results in the following output.

```
retval=20.000000
```

Requirements

Routine	Required header	Architecture
_Dot3dVW0	<shintr.h>	SH-4

See Also

Reference

_Dot4dV

_Dot3dVW1

Intrinsic Functions for Renesas Microprocessors

_Dot3dVW1

_Dot3dVW1

Computes the inner product of a pair of three or four-dimensional vectors with the "w" coordinate forced to 1.

```
float _Dot3dVW1(
  float* vector1,
  float* vector2
);
```

Parameters

vector1

[in] Pointer to a three or four-dimensional source vector.

vector2

[in] Pointer to a three or four-dimensional destination vector.

Return Values

The scalar resulting from the inner product.

Remarks

The fourth coordinate of the source vector will always force to 1.

To implement this function, use the -QSh4 /Oi (Generate Intrinsic Functions) flag when compiling.

The following code example shows how to compute the inner products of three or four dimensional vectors.

```
/*********************************************************/
#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);
}
```

This example results in the following output.

```
retval=21.000000
```

Requirements

Routine	Required header	Architecture
_Dot3dVW1	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors
_Dot4dV
_Dot3dVW0

_Dot4dV

_Dot4dV

Computes the inner product of a pair of four-dimensional vectors.

```
float _Dot4dV(
  float* vector1,
  float* vector2
);
```

Parameters

vector1

[in] Pointer to a four-dimensional source vector.

vector2

[in] Pointer to a four-dimensional destination vector.

Return Values

The scalar resulting from the inner product.

Remarks

The following code example shows how to compute the inner product of four-dimensional vectors.

```
/*********************************************************/
#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);
}
```

This example results in the following output.

```
retval=40.000000
```

Requirements

Routine	Required header	Architecture
_Dot4dV	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors
_Dot3dVW0
_Dot3dVW1

_LoadMatrix

_LoadMatrix

Loads a 4x4 matrix into the extended floating-point register bank.

```
float* _LoadMatrix(
  float* matrix
);
```

Parameters

matrix

[in] Pointer to an array of float values arranged such that the indices of the array are the [row][column] values of the 4x4 matrix.

Return Values

Pointer to the 4x4 matrix that has been loaded.

Remarks

The following code example shows how to use **_LoadMatrix**.

```
#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");
     }
}
```

This example results in the following output.

```
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  |
```

Requirements

Routine	Required header	Architecture
_LoadMatrix	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors

_SaveMatrix

_Multiply4dM

_Multiply4dM

Multiplies a 4x4 matrix by a 4x4 matrix.

```
float* _Multiply4dM(
  float* result,
  float* matrix1,
  float* matrix2
);
```

Parameters

result

[out] Pointer to an array of float values arranged such that the indices of the array are the [row][column] values of the 4x4 matrix. This matrix receives the result of the operation.

matrix1

[in] Pointer to an array of float values arranged such that the indices of the array are the [row][column] values of the 4x4 matrix.

matrix2

[in] Pointer to an array of float values arranged such that the indices of the array are the [row][column] values of the 4x4 matrix.

Return Values

Pointer to the 4x4 result matrix.

Remarks

The following code example shows how to use **_Multiply4DM**.

```
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");
   }
}
```

Output

```
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  |
```

Requirements

Routine	Required header	Architecture
_Multiply4dM	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors

_XDMultMatrix

_SaveMatrix

_SaveMatrix

Stores the extended floating-point register bank into a 4x4 matrix.

```
float* _SaveMatrix(
  float* matrix
);
```

Parameters

matrix

[out] Pointer to an array of float values arranged such that the indices of the array are the [row][column] values of the 4x4 matrix.

Return Values

Pointer to the 4x4 matrix that has been stored.

Remarks

The following code example shows how to use **_SaveMatrix**.

```
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] â€”> 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");
   }
}
```

Output

```
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  |
```

Requirements

Routine	Required header	Architecture
_SaveMatrix	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors

_LoadMatrix

_XDMultMatrix

_XDMultMatrix

Multiplies a 4x4 matrix by a 4x4 matrix. This instruction requires one of the matrices to have been previously loaded into the extended floating-point register bank. The matrix passed as the parameter receives the result of the operation.

```
void _XDMultMatrix(
  float* matrix
);
```

Parameters

matrix

[in, out]Pointer to an array of float values arranged such that the indices of the array are the [row][column] values of the 4x4 matrix.

Return Values

None.

Requirements

Routine	Required header	Architecture
_XDMultMatrix	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors
_Multiply4dM
_LoadMatrix
_SaveMatrix

_XDXform3dV

_XDXform3dV

Multiplies a three-dimensional vector by a 3x3 matrix. This instruction requires that the matrix has been previously loaded into the extended floating-point register bank.

```
float* _XDXform3dV(
  float* src,
  float* dst
);
```

Parameters

src

[in] Pointer to a three-dimensional source vector.

dst

[out] Pointer to a three-dimensional destination vector.

Return Values

A pointer to the three-dimensional destination vector.

Remarks

The following code shows how to use **_XDXform3dV**.

```
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);
}
```

Output

```
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|
```

Requirements

Routine	Required header	Architecture
_XDXform3dV	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors
_Xform3dV
_Xform4dV
_XDXform4dV
_LoadMatrix
_SaveMatrix

_XDXform4dV

_XDXform4dV

Multiplies a four-dimensional vector by a 4x4 matrix. This instruction requires that the matrix has been previously loaded into the extended floating-point register bank.

```
float* _XDXform4dV(
  float* src,
  float* dst
);
```

Parameters

src

[in] Pointer to a three-dimensional source vector.

dst

[out] Pointer to a three-dimensional destination vector.

Return Values

Pointer to the four-dimensional destination vector.

Remarks

The following code shows how to use **_XDXform4dV**.

```
#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);
}
```

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|
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|
```

Requirements

Routine	Required header	Architecture
_XDXform4dV	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors
_Xform3dV
_Xform4dV
_LoadMatrix
_SaveMatrix

_Xform3dV

_Xform3dV

Multiplies a three-dimensional vector by a 3x3 matrix. This intrinsic will load a constant zero into Bank1 floating-point registers (**fr12, fr13, fr14, fr15**) and Bank0 floating-point register (**fr3**).

```
float* _Xform3dV(
  float* dst,
  float* src,
  float* matrix
);
```

Parameters

dst

[out] Pointer to a three-dimensional destination vector.

src

[in] Pointer to a three-dimensional source vector.

matrix

[in] Pointer to an array of float values arranged such that the indices of the array are the [row][column] values of the 3x3 matrix.

Return Values

Pointer to the three-dimensional destination vector.

Remarks

The following code shows how to use **_Xform3dV**.

```
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]] Ã  dest[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|
```

Requirements

Routine	Required header	Architecture
_Xform3dV	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors
_Xform4dV
_XDXform4dV
_XDXform3dV
_LoadMatrix
_SaveMatrix

_Xform4dV

_Xform4dV

Multiplies a four-dimensional vector by a 4x4 matrix.

```
float* _Xform4dV(
  float* dst,
  float* src,
  float* matrix
);
```

Parameters

dst

[out] Pointer to a four-dimensional destination vector.

src

[in] Pointer to a four-dimensional source vector.

matrix

[in] Pointer to an array of float values arranged such that the indices of the array are the [row][column] values of the 4x4 matrix.

Return Values

Pointer to the four-dimensional destination vector.

Remarks

The following code shows how to use **_Xform4dV**.

```
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|
```

Requirements

Routine	Required header	Architecture
_Xform4dV	<shintr.h>	SH-4

See Also

Reference

Intrinsic Functions for Renesas Microprocessors
_Xform3dV
_XDXform4dV
_XDXform3dV
_LoadMatrix
_SaveMatrix

Renesas Compiler Options

Renesas Compiler Options

The Renesas compiler supports the following command-line options:

- -QSfast, -QSfastd - Provide Backward Compatibility
- -Qsh4
- -QSimplicit-import- Disable Importing of Helpers

See Also
Concepts
Unique Build Options

-QSfast, -QSfastd - Provide Backward Compatibility

-QSfast, -QSfastd - Provide Backward Compatibility

These options, **-QSfast** and **-QSfastd** provide backward compatibility. **-QSfastd** is the default.

The device compilers support full IEEE compliant hardware operations including the use of denormal operands. This allows the compiler to default to use of the Renesas SuperH SH-4 hardware instructions for both single and double precision floating-point operations.

The **-QSfast** option causes the compiler to call CRT emulation routines for all floating-point operations.

The **-QSfast** option allows the compiler to use single precision SH-4 floating-point instructions. It will also generate a warning if the compiler finds a double floating point constant.

Do not use double constants when performance is a consideration, because double constants can unnecessarily promote an operation-type from single to double precision.

Note that floating-point constants default to type double. You may specify single precision floating point constants with a floating suffix f or F such as 3.14159f.

The default **-QSfastd** option allows the compiler to use single and double precision SH-4 floating-point instructions.

The compiler does not generate warnings for double constants when you specify **-QSfastd**. The **-QSfastd** command line option may cause the compiler to produce slightly larger code.

See Also

Reference

Renesas Compiler Options

-QSh4

-QSh4

This option specifies compilation for the Renesas SH-4 microprocessor.

See Also

Reference

Renesas Compiler Options

-QSimplicit-import- Disable Importing of Helpers

-QSimplicit-import- Disable Importing of Helpers

This option forces the compiler to use statically linked helper functions. Compiler helper functions are typically located in a DLL.

To reduce the thunk overhead when calling these functions, the compiler uses the dllimport calling mechanism. If OS or driver code is linked statically with the helper functions, the **-QSimplicit-import**- flag is used to force the compiler to use typical calling mechanisms.

See Also

Reference

Renesas Compiler Options

Renesas SH-4 Calling Sequence Specification

Renesas SH-4 Calling Sequence Specification

The Renesas SuperH SH-4 Calling Sequence Specification provides direction for the development of compilers and assembly language programs for the SH-4 microprocessor.

In addition, the standard enables the development of tools, debuggers, and operating system utilities that perform automated walking of the call stack.

In This Section

Renesas SH-4 Registers

Describes register assignments and parameter passing conventions.

Renesas SH-4 Stack Frame Layout

Describes the stack frame layout.

Renesas SH-4 Prolog and Epilog

Describes the code sequence specifications for setting up a stack frame for unwinding.

Related Sections

Renesas Family Processors

Provides an overview of compiler options, intrinsic functions, and call specifications for Renesas microprocessors supported in this release of Visual Studio.

Renesas SH-4 Registers

Renesas SH-4 Registers

The Renesas SuperH SH-4 has 16 general-purpose registers. The following table shows the roles assigned to these registers.

Register	Description
R0	Serves as a temporary register when expanding assembly language pseudo-instructions, and holds function return values. In addition, R0 serves as an implicit source or destination in byte and 16-bit operations. Not preserved.
R1-R3	Serve as temporary registers Not preserved
R4-R7	Hold the first four words of integer and non-scalar incoming arguments. The argument build area provides space into which R4 through R7 holding arguments may spill. Not preserved.
R8-R13	Serve as permanent registers Preserved
R14	Serves as the default frame pointer. Any other permanent register may serve as the frame pointer, and leaf routines may use a temporary register as the frame pointer. Preserved.
R15	Serves as the stack pointer or as a permanent register Preserved.

The SH-4 has two banks of 16 floating-point registers designated Bank0 and Bank1. This specification does not define the use of Bank1. This calling convention assigns the following roles to the SH-4 Bank0 floating-point registers.

Double-Precision Register	Single-Precision Register	Description
DR0	FR0 FR1	Hold function return values. DR0 is another name for the single-precision registers FR0 and FR1 as a pair.
DR2	FR2-FR3	Serve as temporary registers. DR2 is another name for FR2 and FR3 as a pair. Not preserved
DR4-DR10	FR4-FR11	Hold single- or double-precision floating-point arguments. The argument build area also provides space into which floating-point registers holding arguments may spill.
DR12-DR14	FR12-FR15	Serve as permanent registers. Preserved

The floating-point status control affects the behavior of some floating-point instructions. For information about the use of the PR, SZ, and FR bits within prologs and epilogs, see Renesas SH-4 Prolog and Epilog.

See Also

Concepts

Renesas SH-4 Stack Frame Layout

Other Resources

Renesas SH-4 Prolog and Epilog

Renesas SH-4 Stack Frame Layout

Renesas SH-4 Stack Frame Layout

The Renesas SuperH SH-4 stack frame layout uses four designated areas to hold register areas used by functions and space for variables.

- The **Register Save Area** (RSA) holds the preserved values of any permanent registers used by the function. It also contains the function's return address.
- The **Locals and Temporaries** area represents the stack space allocated for local variables and compiler-generated temporaries.
- An **alloca()** locals area is dynamically allocated during function execution by the use of the alloca() intrinsic function. Compiler-generated code may also dynamically allocate space to manage the construction of outgoing arguments.
- The **Outgoing Arguments** area must be large enough to hold all the arguments passed when calling another function, including all arguments passed in registers. This area may be dynamically allocated or extended prior to a call if the rules for alloca() are observed.

SH-4 Frame and Stack Pointers

The following list gives more information about stack and frame pointers:

- The stack pointer and frame pointer addresses are aligned on 4-byte boundaries.
- If a routine has alloca() locals, or dynamically allocates stack frame space for any other reason, a separate frame pointer register accesses incoming arguments and locals. A leaf routine may use any free integer register as the frame pointer. A non-leaf routine must use a permanent register.
- By convention, a routine that establishes a separate frame pointer should use R14. However, a routine may establish another frame pointer to more efficiently access data in large stack frames. If a routine establishes no frame pointer, R15 must remain unchanged between the end of prolog and the beginning of the epilog.

See Also

Concepts

Renesas SH-4 Registers

Other Resources

Renesas SH-4 Prolog and Epilog

Renesas SH-4 Prolog and Epilog

Renesas SH-4 Prolog and Epilog

Renesas SuperH SH-4 prolog and epilog code segments are required to implement Structured Exception Handling (SEH) on Renesas microprocessors. For more information, see SEH in RISC Environments.

The prolog contains the code that sets up the stack frame for a routine. The epilog contains the code that removes the routine's frame and then returns to the calling function.

In This Section

SH-4 Prolog

Describes the requirements for an SH-4 prolog sequence.

SH-4 Epilog

Describes the requirements for an SH-4 epilog sequence.

SH-4 Prolog and Epilog Examples

Provides examples of paired prolog and epilog sequences.

SH-4 Prolog

SH-4 Prolog

The SH-4 prolog contains four parts that are immediately contiguous with no intervening instructions. The following list shows the required parts.

1. A sequence of zero or more instructions that save the incoming argument values from R4 - R7 and FR4 - FR11 to the argument home locations.
   This sequence typically uses the fmov instruction with R15 as the base address register. This sequence will not change the stack pointer, R15.
   In addition, floating-point argument registers can be stored onto the stack using the fmov instruction with the register-indirect and register-indirect-pre-decrement addressing modes.
   A sequence of one or more such fmov instructions may be preceded by a two-instruction address register setup sequence composed of a constant move to a general register, followed by an add of the sp to the register. For example:

   ```
   mov #36, r0
   add sp, r0
   fmov fr5, @r0
   fmov fr4, @-r0
   ```

2. A sequence of zero or more instructions that push all permanent registers to be saved and the return address (PR), if it is to be saved, onto the stack using the pre-decrement indirect register addressing mode with R15 as the address register.
   If a permanent register such as R14 is to be used as the frame pointer, it is pushed first.
   If the PR is to be saved, it is pushed last.
   The prolog uses mov.l instructions to save registers other than the return address, and the sts.l instruction to save the return address.
   The prolog also uses the fmov.s instruction to save floating-point registers, and may invoke double-precision and move-extension fmov instructions by first setting the SZ bit in the FPSCR.
3. A sequence of one or more instructions that set up the frame pointer, if one is to be established.
   To set up the frame pointer, the step 3 sequence first copies the contents of R15 to the frame pointer register, then adds or subtracts the amount of offset to the frame pointer address from the frame pointer register.
   If the frame pointer address is less than the current value in R15, the prolog subtracts the offset to the frame pointer address from R15, and copies R15 to the frame pointer register. If this method is used, the routine must later take the size of the decrement to R15 into account.
   If the routine is not a leaf routine, the function must use a permanent register, typically R14, as the frame pointer.
4. A sequence of zero or more instructions that allocate the remaining stack frame space for local variables, compiler-generated temporaries, and the argument-build area by subtracting a 4-byte aligned offset from R15.

The Virtual Unwinder considers the last instruction in this sequence the last instruction of the prolog.

If necessary, the temporary register R1 holds an offset too large to be represented in the immediate field of an add instruction.

See Also

Concepts

SH-4 Epilog
SH-4 Prolog and Epilog Examples

Other Resources

Renesas SH-4 Prolog and Epilog

SH-4 Epilog

SH-4 Epilog

The SH-4 epilog is a contiguous sequence of instructions that restores the saved permanent registers, resets the stack pointer to its value on function entry, and returns to the function's calling function.

In addition, the **Virtual Unwinder** requires the epilog to have the following parts, except where noted:

1. A single add instruction that increments the frame pointer. This instruction must immediately precede the sequence of instructions defined for part 2. (Optional)
2. A sequence of instructions that modify **R15** by referencing it in the destination operand of the instruction or in a post-increment memory address operand of the instruction. This sequence immediately precedes the rts instruction of part 3. All instructions in this sequence must be **lds, mov, fmov, fschg,** or **add** instructions.
   This item is optional. It does not appear in functions that have no RSA area to restore.
3. The **rts** instruction and its delay slot instruction. The instruction in the delay slot of the **rts** is considered part of the epilog but is not required to conform to the rules listed for item 2.

When unwinding out of a function, the **Virtual Unwinder** must determine if the currently executing instruction is in the prolog or epilog. If the current point of control is in the prolog or in the epilog, the **Virtual Unwinder** must take special measures to unwind out of the function.

The following list shows three conditions that the **Virtual Unwinder** must maintain:

- If the function establishes a frame pointer, the function may not modify the frame pointer value during the interval between the completion of the last prolog instruction and the beginning of the first instruction of the epilog.
  If the function does not establish a frame pointer, the function must not modify value in **R15** during the interval between the completion of the last prolog instruction and the beginning of the first instruction of the epilog.
- The address contained in the stack pointer, which is always **R15**, must never be greater than the lowest address of any unrestored register value in the Register Save Area.
- The **SZ, PR,** and **FR** bits of the **FPSCR** register must always be set to zero on entering and exiting the prolog and epilog. The **PR** and **FR** bits must not be modified within the prolog or epilog. This condition enables the Virtual Unwinder to correctly reverse-execute floating-point instructions.

See Also

Concepts

SH-4 Prolog

SH-4 Prolog and Epilog Examples

Other Resources

Renesas SH-4 Prolog and Epilog

SH-4 Prolog and Epilog Examples

SH-4 Prolog and Epilog Examples

The following code examples show how to use prolog and epilog to perform certain tasks.

- Allocate a stack frame to save R14, R8, and PR
  This example allocates a stack frame to save R14, R8, and PR, and to allow alloca() calls. It also allocates a 4-word argument build area. Local variables and temporaries do not need stack space.

```
// 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
```

- Allocate a stack frame for a leaf routine
  This example allocates a stack frame for a leaf routine that requires 40 bytes for local variables and temporaries. It uses permanent registers R8, R9, and R10. This routine has no _alloca locals, so no frame pointer is required.

```
//  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
```

See Also

Reference

SH-4 Assembler Macros

Other Resources

Renesas SH-4 Prolog and Epilog

SH-4 Assembler Macros

SH-4 Assembler Macros

SH-4 assembler-level macros are reuired to implement prolog and epilog code sequences.

The following table shows the macros defined for SH-4 microprocessors.

Macro name	Description
ALTERNATE_ENTRY (SH-4)	Declares an alternate entry to a routine.
END_REGION (SH-4)	Marks the end of a contiguous range of text or data.
ENTRY_END (SH-4)	Ends the routine that was specified by a prior NESTED_ENTRY.
EXCEPTION_HANDLER (SH-4)	Associates a named exception handler with the subsequent NESTED_ENTRY.
EXCEPTION_HANDLER_DATA (SH-4)	Associates a named exception handler and the handler data with the subsequent NESTED_ENTRY.
LEAF_ENTRY (SH-4)	Declares the beginning of a routine that does not require any prolog code.
NESTED_ENTRY (SH-4)	Declares the beginning of a routine that either has an existing stack frame or creates a new stack frame.
PROLOG_END (SH-4)	Marks the end of the prolog area.
START_REGION (SH-4)	Marks the beginning of a contiguous range of text or data.

See Also

Other Resources

Renesas SH-4 Calling Sequence Specification

ALTERNATE_ENTRY (SH-4)

ALTERNATE_ENTRY (SH-4)

This macro declares an alternate entry to a routine of type NESTED_ENTRY (SH-4) or LEAF_ENTRY (SH-4).

```
ALTERNATE_ENTRY Name[,
[Section=]SectionName]
```

Parameters

Name

Name is the entry point and is in the global name space.

SectionName

SectionName is the name of the section in which the entry will appear; see Remarks.

Return Values

None.

Remarks

The **ALTERNATE_ENTRY** macro does not use the *SectionName* parameter. The parameter is accepted and ignored for consistency with **NESTED_ENTRY** and **LEAF_ENTRY**.

If used, an **ALTERNATE_ENTRY** call must appear in the body of a routine.

See Also

Reference

NESTED_ENTRY (SH-4)

LEAF_ENTRY (SH-4)

END_REGION (SH-4)

END_REGION (SH-4)

This macro marks the end of a contiguous range of text or data.

```
END_REGION NameEnd
```

Parameters
NameEnd

NameEnd labels the end of the range.

Return Values

None.

Remarks

NameEnd is in the global name space.

See Also
Reference
START_REGION (SH-4)

ENTRY_END (SH-4)

ENTRY_END (SH-4)

This macro ends the current routine specified by NESTED_ENTRY (SH-4) or LEAF_ENTRY (SH-4).

```
ENTRY_END [Name]
```

Parameters
Name

Name is the entry point and is in the global name space.See Remarks.

Return Values

None.

Remarks

Name should be the same name used in the **NESTED_ENTRY** or **LEAF_ENTRY** macros.

The **ENTRY_END (SH-4)** macro currently ignores *Name*.

See Also
Reference
NESTED_ENTRY (SH-4)
LEAF_ENTRY (SH-4)

EXCEPTION_HANDLER (SH-4)

EXCEPTION_HANDLER (SH-4)

This macro associates an exception handler *Handler* with a subsequent NESTED_ENTRY (SH-4) or LEAF_ENTRY (SH-4) routine.

```
EXCEPTION_HANDLER Handler
```

Parameters
Handler

Name of exception handler.

Return Values

None.

Remarks

This association is in effect until the compiler encounters a matching ENTRY_END (SH-4) macro.

See Also
Reference
NESTED_ENTRY (SH-4)
LEAF_ENTRY (SH-4)
ENTRY_END (SH-4)

EXCEPTION_HANDLER_DATA (SH-4)

EXCEPTION_HANDLER_DATA (SH-4)

This macro associates an exception handler *Handler* and the *HandlerData* with a subsequent NESTED_ENTRY (SH-4) or LEAF_ENTRY (SH-4) routine.

```
EXCEPTION_HANDLER_DATA Handler,
     HandlerData
```

Parameters

Handler

Name of exception handler.

HandlerData

String associated with exception handler.

Return Values

None.

Remarks

This association is in effect until compiler encounters a matching ENTRY_END (SH-4) macro.

See Also

Reference

NESTED_ENTRY (SH-4)
LEAF_ENTRY (SH-4)
ENTRY_END (SH-4)
EXCEPTION_HANDLER (SH-4)

LEAF_ENTRY (SH-4)

LEAF_ENTRY (SH-4)

This macro declares the beginning of a routine that does not require any prolog code.

```
LEAF_ENTRY Name[,
[Section=]SectionName]
```

Parameters

Name

Name is the routine name and is in the global name space.

SectionName

SectionName is the name of the section in which the entry will appear; it is optional and defaults to .text.

Return Values

None.

Remarks

A LEAF_ENTRY must have an associated ENTRY_END (SH-4).

See Also

Reference

ENTRY_END (SH-4)

PROLOG_END (SH-4)

NESTED_ENTRY (SH-4)

NESTED_ENTRY (SH-4)

This macro declares the beginning of a routine that either has an existing frame or creates a stack frame.

```
NESTED_ENTRY Name[,
[Section=]SectionName]
```

Parameters

Name

Name is the routine name and is in the global name space.

SectionName

SectionName is the name of the section in which the entry will appear; it is optional and defaults to text.

Return Values

None.

Remarks

A **NESTED_ENTRY** must have an associated PROLOG_END (SH-4) and ENTRY_END (SH-4).

See Also

Reference

ENTRY_END (SH-4)

PROLOG_END (SH-4)

PROLOG_END (SH-4)

PROLOG_END (SH-4)

This macro marks the end of the prolog area.

```
PROLOG_END
```

Parameters
None.

Return Values

None.

Remarks

This macro must appear following a **NESTED_ENTRY (SH-4)** or **LEAF_ENTRY (SH-4)** macro.

It appears after the prolog area and before the matching ENTRY_END (SH-4) macro.

See Also
Reference
NESTED_ENTRY (SH-4)
LEAF_ENTRY (SH-4)
ENTRY_END (SH-4)

START_REGION (SH-4)

START_REGION (SH-4)

This macro marks the beginning of a contiguous range of text or data.

```
START_REGION NameBegin
```

Parameters
NameBegin

NameBegin labels the beginning of the range. *NameBegin* is in the global name space.

Return Values

None.

See Also
Reference
END_REGION (SH-4)

Renesas SH-4 Series Assembler

Renesas SH-4 Series Assembler

The SH-4 Series Assembler (SHASM) converts source programs written in assembly language into a format that can be handled by SH microprocessors, outputting the result as an object module and as an assembler listing. SHASM generates output in Microsoft Common Object File Format (COFF).

SHASM contains numerous enhancements designed to enhance the programming environment and to address compatibility issues.

Although SHASM closely resembles the Renesas native assembler, it is not identical. For example, SHASM generates error and warning messages that contain not only an error number but also a descriptive message.

In This Section

SH-4 Assembler Command-Line Options

Describes the command-line options available to SHASM

SH-4 Assembler Directives

Describes assembler directives

SH-4 Assembler Error Messages

Lists the error messages displayed by the SH-4 assembler.

Related Sections

Renesas SH-4 Calling Sequence Specification

Provides an overview and links to information about register assignments and stack layout

SH-4 Assembler Command-Line Options

SH-4 Assembler Command-Line Options

To start the SH-4 assembler, enter a command line with the following format when the host computer operating system is in the input wait state.

```
shasm <input source file> [,<input source file>...][command line options>...]
```

When you specify multiple source files on the command line, the assembler creates a unit of assembly processing by concatenating the specified files in the specified order. Because of this, the .END directive must appear only in the last file.

Command line options can begin with either a - (hyphen) or a / (slash). File names can be specified with forward or backward slashes.

If the assembler has the option of interpreting an argument as either a command line option or a file name, the argument will be interpreted as a command line option. For example, if a file called /source.src is assembled by writing `shasm /source`, the assembler interprets /source as a switch, and issues a message that there are no input files.

Command line options provide additional specifications for the assembly processing. The following table shows the command line options for the SH-4 assembler.

Command line option	**Description**
-OBJECT	Specifies output of an object module. Default value.
-NOOBJECT	Suppresses output of an object module.
-DEBUG	Specifies output of debug information.
-NODEBUG	Suppresses output of debug information.
-LIST	Specifies output of an assembler listing. Default value.
-NOLIST	Suppresses output of an assembler listing.
-SHOW	Specifies output of the preprocessor function source program.
-NOSHOW	Suppresses output of specified preprocessor function source statements and object code display lines in the source program listing.
-LINES	Sets the number of lines in the assembler listing.
-COLUMNS	Sets the number of columns in the assembler listing.
-FoOBJPATH	Sets the path to which the object should be written. A synonym for the -o=OBJPATH option. Provided for compatibility with Microsoft-based tools.
-wide[_listing]	Shows up to eight bytes of machine code or data per line in the assemble listing. Default is four bytes per line.
-nowide[_listing]	Shows up to four bytes of machine code or data per line in the assemble listing. This is the default behavior.
-tab[_expand][=<number>]	Expands tab characters in the assemble listing. The number, if specified, is the spacing of tab stops. Default is eight.

-Notab[_expand]	Writes tab characters into the assemble listing unchanged. This is the default behavior.
-maxerr[ors]=<number>	Aborts the assembly after <number> errors have been reported. Default is 100.
-Qsh<version>[r<revision>]	Selects SH microprocessor version and revision that control the setting of the _M_SH and _M_SH_REV predefined symbols. Default is version SH-4.
-nologo	Instructs not to print the logo banner when it runs.
-help -usage	Output a detailed command line usage message and exit immediately.
-D<name>[=<number>]	Predefines an equate; for example, -DTERM is equivalent to TERM.
-CPU=SH<version>	Selects target CPU for the source program being assembled. Valid options are: • SH1 • SH2 • SH4

The assembler listing is a listing to which the results of the assembly processing are output.

Note that when compiling an assembly source file that ends in .s, in contrast to .asm or .src, then the output object file concatenates .obj to the end of the file name. For example, a file named test.s is assembled to test.s.obj.

Return Codes

The assembler delivers a return code that reports whether or not the assembly processing terminated normally. The following table shows the return values that indicate the activity at termination.

Activity	**Return value**
Normal termination	0
Warnings occurred	0
Errors occurred	2
Fatal error occurred	4

See Also

Other Resources

Renesas SH-4 Series Assembler

SH-4 Assembler Directives

SH-4 Assembler Directives

The SH-4 assembler (SHASM) includes numerous assembler directives that the assembler interprets and executes.

In This Section

SH-4 Assembler Macro Directives

SH-4 Assembler Macro Directives

The assembler provides the following macro function directives.

Directive	Syntax	Description
.MACRO	`.MACRO <macro name>[ <formal parameter>[=<default>] [,<formal parameter>...]]`	Defines a macro.
.ENDM	`ENDM`	Indicates the end of a macro definition.
.EXITM	`.EXITM`	Terminates macro expansion.

See Also

Other Resources

SH-4 Assembler Directives

SH-4 Assembler Section and Location Directives

SH-4 Assembler Section and Location Directives

The following table lists shows SH-4 assembler directives for Section assignment and the location counters.

Directive	Syntax	Description
.SECTION	`.SECTION <section name> [,<section type>[ ,<section attributes>...]]`	Declares a section.
.ORG	`.ORG <location-counter-value>`	Sets the value of the location counter. location counter value must be an absolute value with no forward reference symbols.
.ALIGN	`.ALIGN <boundary alignment value>`	Adjusts the value of the location counter to a multiple of boundary alignment value. boundary alignment value must be an absolute value wit h no forward reference symbols.

The assembler also provides a default section for the situations in which no section has been declared.

The following list shows the kinds of statements and instructions for which SHASM provides a default section.

- Executable instructions
- Data reservation assembler directives
- .ALIGN assembler directive
- .ORG assembler directive
- Reference to the location counter
- Statements consisting of only the label field

See Also

Other Resources

SH-4 Assembler Directives

SH-4 Assembler Symbol Handling Directives

SH-4 Assembler Symbol Handling Directives

The following table shows SH-4 assembler directives for symbol handling.

Directive	Syntax	Description
.EQU	`<symbol>[:] .EQU <symbol value>`	Sets a symbol value. Symbols defined with the .EQU directive cannot be redefined.
.ASSIGN	`<symbol>[:] .ASSIGN <symbol value>`	Sets a symbol value that can be redefined.
.REG	`<symbol>[:] .REG <register name><symbol>[:] .REG (<register name>)`	Defines the alias of a register name. The alias of a register name defined with .REG cannot be redefined.
.EXPORT	`.EXPORT <symbol>[,<symbol>...]`	Declares export symbols. This declaration allows symbols defined in the current file to be referenced in other files.
.IMPORT	`.IMPORT <symbol>[,<symbol>...]`	Declares import symbols. This declaration allows symbols defined in other files to be referenced in the current file.
.GLOBAL	`.GLOBAL <symbol>[,<symbol>...]`	Declares export and import symbols. This declaration allows symbols defined in the current file to be referenced in other files and allows symbols defined in other files to be referenced in the current file.

See Also

Other Resources

SH-4 Assembler Directives

SH-4 Assembler Data and Data Area Directives

SH-4 Assembler Data and Data Area Directives

The following table shows SH-4 assembler directives for data and data area reservation.

Directive	Syntax	Description
.COMMON	`.COMMON <symbol name>[,<size>]`	Reserves data area with default initialization to zeros.
.DATA	`[<symbol>[:]] .DATA[.<operation size>] <integer data>[,<integer data>...]`	Reserves integer data. The following list shows the data size of symbol. • B - Byte • W - Word, 2 bytes • L - Longword, 4 bytes, default
.DATAB	`[<symbol>[:]] .DATAB[.<operation size>]<block count>,<integer data>`	Reserves integer data blocks. The block count specification must be an absolute value with no forward reference symbols.
.FDATA	`[<symbol>[:]] .FDATA[.<operation size>] <floating point data>[,<floating point data>...]`	Reserves floating-point data.
.FDATAB	`[<symbol>[:]] .FDATAB[.<operation size>] <block count>,<floating point data>`	Reserves floating-point data blocks.
.FRES	`[<symbol>[:]] .FRES[.<operation size>] <area count>`	Reserves floating-point area. The area count specification must be an absolute value with no forward reference symbols.
.SDATA	`[<symbol>[:]] .SDATA <character string>[,<character string>...]`	Reserves character string data.
.SDATAB	`[<symbol>[:]] .SDATAB <block count>,<character string>`	Reserves character string data blocks.
.SDATAC	`[<symbol>[:]] .SDATAC <character string>[,<character string>...]`	Reserves character string data with length.
.SDATACB	`[<symbol>[:]] .SDATACB <block count>,<character string>`	Reserves specified number of character string data blocks with length. A character string with length is a character string with an inserted leading byte that indicates the length of the string.

.SDATAZ	`[<symbol>[:]] .SDATAZ <character string>[,<character string>...]`	Reserves character string data with zero terminator.
.SDATAZB	`[<symbol>[:]] .SDATAZB <block count>,<character string>`	Reserves specified number of character string data blocks with zero terminator.
.RES	`[<symbol>[:]] .RES[.<operation size>] <area count>`	Reserves data area. The area count specification must be an absolute value with no forward reference symbols.
.SRES	`[<symbol>[:]] .SRES <character string area size>[,<character string area size>...]`	Reserves character string data area. The character string area size specification must be an absolute value with no forward reference symbols.
.SRESC	`[<symbol>[:]] .SRESC <character string area size>[,<character string area size>...]`	Reserves character string data area with length. A character string with length is a character string with an inserted leading byte that indicates the length of the string.
.SRESZ	`[<symbol>[:]] .SRESZ <character string area size>[,<character string area size>...]`	Reserves character string data area with zero terminator.

String constants in the .SDATA family of directives are allowed to be one or more of the following in any combination:

```
"quoted string"
<control char expression>
```

The assembler requires that parentheses be put around string constants if they are to be used as part of more complicated expressions such as string comparisons in .SDATA and related directives.

In all other contexts, string constants can only be double-quoted strings, and can be used without parentheses.

See Also

Other Resources

SH-4 Assembler Directives

SH-4 Assembler Function-Definition Directives

SH-4 Assembler Function-Definition Directives

To support the requirements of C++ and structured exception handling as well as provision of stack traces in the debugger, object files must contain information regarding where functions begin and end and what part of each function is devoted to setting up the function's stack frame.

The following table shows SH assembler directives that define functions.

Directive	Syntax	Description
.ENTRY	`<symbol>[:] .ENTRY`	Marks beginning of a function.
.BIGENTRY	`<symbol>[:] .BIGENTRY`	Marks beginning of a function.
.ENDF	`.ENDF`	Marks end of a function.
.PROLOG	`.PROLOG`	Marks end of a function's prolog code. The function prolog is the code at the beginning of the function that sets up the function stack frame. The .PROLOG directive should appear between the last line of prolog code and the first line of non-prolog code for the function.
.PDATA	`.PDATA <handler address>[,<handler data>]`	Specifies language-specific exception-handling information for current function. The .PDATA directive must appear before the .ENTRY or .BIGENTRY section to which it applies.

See Also

Other Resources

SH-4 Assembler Directives

SH-4 Assembler Debug Information Directives

SH-4 Assembler Debug Information Directives

The following table shows SH-4 assembler directives that control output of symbols and object modules.

Directive	Syntax	Description
.OUTPUT	`.OUTPUT <output specifier>[,<output specifier>]`	Controls object module and debug information output.
.DEBUG	`.DEBUG <output specifier>`	Controls the output of symbolic debug information.

Specifications concerning debug information output are only valid when an object module is output.

The following table shows possible values for output specifier.

Output Specifier	Description
OBJ (default)	An object module is output.
NOOBJ	No object module is output.
DBG	Debug information is output in the object module.
NODBG (default)	No debug information is output in the object module.

The .DEBUG directive specification is valid only when both an object module and debug information are output.

See Also

Other Resources

SH-4 Assembler Directives

SH-4 Assembler Listing Directives

SH-4 Assembler Listing Directives

The following table shows SH-4 assembler directives that control listings.

Directive	Syntax	Description
.PRINT	.PRINT <output specifier>[,<output specifier>...]	Controls assembler-listing output. For more information, see SH-4 Assembler PRINT Directive Specifiers.
.LIST	.LIST <output specifier>[,<output specifier>...]Output specifier:{ ON|OFF|COND| NOCOND|DEF|NODEF| CALL|NOCALL|EXP| NOEXP|CODE|NOCODE}	Controls the output of the source program listing. For more information, see SH-4 Assembler LIST Directive Specifiers.
.FORM	.FORM <size specifier>[,<size specifier>...]	Sets the number of lines and columns in the assembler listing. For more information, see SH-4 Assembler FORM Directive Specifiers.
.HEADING	.HEADING [<expression>[,<expression>...]]	Sets the header for the source program listing.
.PAGE	.PAGE	Inserts a new page in the source program listing.
.SPACE	.SPACE[<line count>]	Outputs blank lines to the source program listing.

The assembler gives priority to command line option specifications concerning assembler-listing output.

See Also

Other Resources

SH-4 Assembler Directives

SH-4 Assembler PRINT Directive Specifiers

SH-4 Assembler PRINT Directive Specifiers

The following table shows possible values for the PRINT directive output specifier.

Output Specifier	Assembler Action
LIST	An assembler listing is output
NOLIST (default)	No assembler listing is output

See Also

Reference

SH-4 Assembler Listing Directives

SH-4 Assembler LIST Directive Specifiers

SH-4 Assembler LIST Directive Specifiers

.LIST is the assembler directive that controls output of the source program listing. The following list shows the possible LIST directive output selections.

- Source statements.
- Source statements related to conditional assembly and macro functions.
- Object code lines.

The following table shows the possible values for LIST directive output specifiers.

Type	Output(default)	Not Output	Object	Description
1	ON	OFF	Source statements	The source statements following this directive.
2	COND	NOCOND	Failed condition	Condition-failed .AIF directive statements.
n/a	DEF	NODEF	Definition	Macro definition statements, .AREPEAT, .AWHILE, .ASSIGNA, and .ASSIGNC.
n/a	CALL	NOCALL	Call	Macro call statements, .AIF, and .AENDI.
n/a	EXP	NOEXP	Expansion	Macro expansion statements, .AREPEAT, and .AWHILE.
3	CODE	NOCODE	Object code lines	The object code lines exceeding the source statement lines.

The specification of the .LIST directive is only valid when an assembler listing is output.

See Also

Reference

SH-4 Assembler Listing Directives

SH-4 Assembler FORM Directive Specifiers

SH-4 Assembler FORM Directive Specifiers

These specifications determine the number of lines and columns in the assembler listing.

Size Specifier	Listing Size
LIN=<line count>	The specified value is set to the number of lines per page.
COL=<column count>	The specified value is set to the number of columns per line.

See Also

Reference

SH-4 Assembler Listing Directives

SH-4 Assembler Conditional Assembly Directives

SH-4 Assembler Conditional Assembly Directives

The following table shows the conditional assembly directives.

Directive	Syntax	Description
.ASSIGNA	`<preprocessor variable>[:].ASSIGNA <value>`	Defines an integer preprocessor variable; the defined variable can be redefined.
.ASSIGNC	`<preprocessor variable>[:].ASSIGNCD<character string>`	Defines a character preprocessor variable; the defined variable can be redefined.
.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`	Determines whether to assemble a part of a source program according to the specified condition. When the condition is satisfied, the statements after the .AIF are assembled. When the condition is not satisfied, the statements after the .AELSE are assembled.
.ERROR or .AERROR	`.ERROR[<expression>[,<expression>...]]`	Reports error detected at assembly time by user code. The alternate spelling .AERROR is also supported
.AREPEAT.AENDR	`.AREPEATD<count><Source statements iteratively assembled>.AENDR`	Repeats assembly of a part of a source program between .AREPEAT and .AENDR the specified number of times.
.AWHILE.AENDW	`.AWHILE <condition><Source statements iteratively assembled>.AENDW`	Assembles a part of a source program between .AWHILE and .AENDW iteratively while the specified condition is satisfied.
.EXITM	`.EXITM`	Terminates .AREPEAT or .AWHILE iterated expansion.
.ALIMIT	`.ALIMITD<maximum count>`	Specifies the upper limit value for expansion of the AWHILE directive in the preprocessor.

See Also

Other Resources

SH-4 Assembler Directives

SH-4 Assembler Miscellaneous Directives

SH-4 Assembler Miscellaneous Directives

The following table shows miscellaneous other SH-4 assembler directives.

Directive	Syntax	Description
.RADIX	`.RADIX <radix specifier>`	Sets the radix, or base, for integer constants with no radix specification. Unless otherwise specified, integer constants are assumed decimal numbers.
.WEAK	`.WEAK <weak name>,<default name>[,<type>]`	Declares weak external symbols. Weak externals are a mechanism for programs allowing flexibility at link time.
.END	`.END`	Declares the end of the source program.
.INCLUDE	`.INCLUDE <file name>`	Includes the specified file in the current assembly. The file name can include the directory. If the file is not found in the current directory and is not specified with an absolute path, the assembler looks in each of the directories specified in the INCLUDE environment variable.
.LEN	`.LEN[ ](<character string>)`	Counts the length of a character string.
.INSTR	`.INSTR[ ](<character string 1>,<character string 2>[,<start position>])`	Searches for a character string.
.SUBSTR	`.SUBSTR[ ](<character string>,<start position>,<extraction length>)`	Extracts a character string.

The following table shows valid RADIX specifications.

Radix Specification	**Description**
B (or b)	Binary
Q (or q)	Octal
O (or o)	Octal
D (or d) (default)	Decimal
H (or h)	Hexadecimal
X (or x)	Hexadecimal

See Also

Other Resources

SH-4 Assembler Error Messages

SH-4 Assembler Error Messages

This section contains descriptions of the SH-4 Assembler error messages.

See Also

Other Resources

Renesas SH-4 Series Assembler

SH-4 Assembler Error Messages 1-41

SH-4 Assembler Error Messages 1-41

The following table shows the SH-4 assembler error messages 1-41.

Message # and Severity	Message Text	Explanation
1 FATAL	Internal error: %s	If this message appears, there is a bug in the assembler. Please report the exact message, and supply a sample program that caused the message.
2 FATAL	Too many errors (%d) -- stopping assembly	The maximum number of errors was exceeded. The maximum number of errors can be controlled using the -maxerrors command line option.
3 FATAL	%s not yet implemented	If this message appears, a source program has attempted to use a technology that is not yet implemented in the assembler. Please report the exact message.
10 ERROR	No input files were specified -- nothing to do	After parsing all of the command line options, the assembler has no input files that it was able to find and open.
21 ERROR	Error reading file: %s	Indicates that the assembler was suddenly unable to continue reading input from a file that it successfully opened.
30 ERROR	Unknown command line switch "%s"	A command line option was specified that the assembler does not recognize or support.
31 ERROR	Syntax error in command line switch "%s": %s	A recognized command line option was specified incorrectly.
32 ERROR	An argument is required for %s	An argument was not supplied for a command line option that requires one.
33 ERROR	A numeric argument is required for %s	An argument was not supplied or a non-numeric argument was supplied for a command line option that requires a numeric argument.
34 ERROR	%s is out of range; must be >= %d and <= %d	A numeric argument was specified that is not within the valid range. The error message will show the valid range for the argument.
35 ERROR	Invalid -show/-noshow option list: "%s"	The assembler could not parse some part of a -show or -noshow command line option.
36 WARN	Ignoring unimplemented option "%s"	A command line option or assembler directive was used that specifies a technology that is not currently implemented in the assembler.
37 WARN	Interpreting option "%s" as "%s"	A command line option was specified in a way that could be interpreted in more than one way by the assembler. This message tells which way the assembler chose to interpret the option. For instance, -DEBUG could mean the -D option with the argument EBUG (meaning to define EBUG as 1) or it could mean the switch -debug (enabling output of debugging information). This particular case could be avoided by using debug (or -DEBUG=1 for an EBUG equate).
40 FATAL	Insufficient memory%s	The assembler was unable to allocate memory for some purpose.

41 FATAL	Insufficient memory; cannot allocate %u more %s	The assembler was unable to allocate memory for some purpose.

See Also

Other Resources

Renesas SH-4 Series Assembler

SH-4 Assembler Error Messages 70-153

SH-4 Assembler Error Messages 70-153

The following table shows the SH-4 assembler error messages 70-153.

Message # and Severity	Message Text	Explanation
70 ERROR	Character %c was not expected	The assembler encountered a character that it did not expect to see in a source file.
71 ERROR	Nonascii character (^%c) in source	The assembler encountered a character that it did not expect to see in a source file.
72 ERROR	Nonascii character (0x%02d) in source	The assembler encountered a character that it did not expect to see in a source file.
73 ERROR	What do you mean by "%s"?	The assembler encountered a combination of characters that it does not understand.
74 ERROR	"%c" is not a valid radix specifier (use HDQBXO)	A radix specifier was used that the assembler does not recognize. Use one of H, D, Q, B, X, O, h, d, q, b, x, or o.
75 ERROR	"%s" is not a valid radix specifier (use HDQBXO)	A radix specifier was used that the assembler does not recognize. Use one of the following: H, D, Q, B, X, O, h, d, q, b, x, or o.
76 ERROR	Overflow in numeric constant	A numeric constant was written that cannot be represented in 32 bits. The valid range is -2,147,483,648 to 4,294,967,295 or H'00000000 to H'FFFFFFFF.
77 ERROR	A backquoted symbol name cannot span multiple lines	A backquote character was written (`), which can be used to enclose a symbol whose name contains characters that are not ordinarily allowed, but no matching backquote character was found on the same line. These characters must come in pairs with a symbol name between them.
78 ERROR	A backquoted symbol name cannot be null (``)	Two adjacent backquote characters (``) were written, which would represent a symbol whose name is null (zero-length), but such a symbol name is illegal. The assembler will also issue this error message if a backquote (`) was written as the last character on a line.
80 ERROR	Parser error: %s	If this message appears, there is a bug in the assembler because it should recover from the error and issue a more helpful error message. Please report the exact message, and if possible, provide a source file that causes the assembler to issue the message.
81 ERROR	Failed to parse input	If this message appears, there is a bug in the assembler because it should recover from the error and issue a more helpful error message). Please report the exact message, and if possible, provide a source file that causes the assembler to issue the message.
101 ERROR	Syntax error	Something is syntactically invalid on the line for which this error is reported.

150 ERROR	Cannot put a %s instruction in a delay slot	An instruction was placed in a delay slot that is not allowed in a delay slot.
151 ERROR	Cannot compute PC displacement in a delay slot	The assembler is not able to compute the PC displacement for a PC-relative addressing mode in a delay slot because such a displacement must be relative to the PC of the branch target.
152 ERROR	Cannot use "%s #x" in section (%s) with align < %d	A MOV.L #imm,Rn cannot be used in a section whose alignment is less than 4, nor a MOV.W #imm,Rn in a section whose alignment is less than 2.
153 ERROR	Instruction at odd address	An instruction at an odd address is not permitted. Attempting to execute it will cause a processor exception at run time.

See Also
Other Resources
Renesas SH-4 Series Assembler

SH-4 Assembler Error Messages 200-312

SH-4 Assembler Error Messages 200-312

The following table shows the SH-4 assembler error messages 200-312.

Message # and Severity	Message Text	Explanation
200 ERROR	Symbol "%s", first used at %s, is never defined	A symbol was referenced without providing a definition for it. Either define it (using .EQU, a label, and so on), or import it (using .IMPORT or .GLOBAL).
202 ERROR	Illegal symbol "%s"	User-defined symbols are not permitted to begin with \& or . (a period).
203 ERROR	Illegal label symbol "%s"	User defined symbols are not permitted to begin with \& or . (a period).
205 WARN	%s has already been declared %s	A symbol was declared or defined that has already been declared or defined.
206 ERROR	%s has already been declared %s	A symbol was declared or defined that has already been declared or defined.
207 ERROR	Preprocessor symbol "%s" has not been defined yet	A preprocessor symbol was used before defining it.
210 ERROR	A string ("%s") is not valid here	A string constant or a string-valued symbol or expression was used in a context in which strings, including character constants, are not supported.
211 ERROR	Cannot %s a %s symbol (%s)	A symbol that cannot be imported or exported (one defined using .ASSIGN, .ASSIGNA, .ASSIGNC, .REG, or .SECTION, or an internal register symbol) was used in an .EXPORT or .GLOBAL assembler directive.
212 ERROR	Symbol "%s" has not been defined yet	A symbol was referenced in a context where forward references are not permitted, but the symbol has not yet been defined, or its value is not yet known.
213 ERROR	The value of symbol "%s" is not known yet	A symbol was referenced in a context where forward references are not permitted, but the symbol has not yet been defined, or its value is not yet known.
214 ERROR	"%s" is an .IMPORT symbol; its value is unknown	An .IMPORT symbol was used in a context where external references are not permitted.
301 ERROR	Too many operands; expecting %s	Too many operands were supplied for an instruction or an assembler directive.
302 ERROR	"%s" is not an opcode, directive, or macro	An operation name was specified that is not an opcode or directive, nor a defined macro. Check the spelling, or if it is a macro, make sure it has been defined before the first use.
304 ERROR	Not enough operands; expecting %s	Fewer operands were supplied than are required for an instruction or an assembler directive.

307 ERROR	This addressing mode is not legal here	An addressing mode was specified that is not legal for the instruction or the instruction/modifier combination, or a modifier was specified that is not legal for the instruction.
308 ERROR	Invalid addressing mode (incompatible operands)	A combination of addressing modes was specified that is not legal for the instruction or the instruction/modifier combination, or a modifier was specified that is not legal for the combination of addressing modes and the instruction.
309 ERROR	Invalid addressing mode (illegal register %s): "%s"	A register name was specified in a place where that register (or register type) is not legal.
310 ERROR	Cannot use %s here; only r0 is valid	A register other than r0 was specified in a context where only r0 is legal.
311 ERROR	"%s" (%s) is not a register	The symbol specified as the operand of a .REG directive is not a valid register name.
312 ERROR	This would require %s relocation	An operation was attempted that cannot be represented in the MS-COFF object format as currently defined.

See Also

Other Resources

Renesas SH-4 Series Assembler

SH-4 Assembler Error Messages 400-453

SH-4 Assembler Error Messages 400-453

The following table shows the SH-4 assembler error messages 400-453.

Message # and Severity	Message Text	Explanation
400 ERROR	Character constant longer than 4 characters: "%s"	A character constant may be no longer than four characters. This message might also be issued if a string (or a string-valued symbol or expression) is supplied in a context where a number is expected.
402 ERROR	%d (0x%x) is out of range; must be >= %d and <= %u	An operand value is outside of the legal range. The message tells you what the valid range is. This message can also be issued when an attempt is made to use a PC-relative operation (such as a branch, or a MOV @(disp, PC),Rn) to refer a location that is too far away, or that is a backward reference for an instruction that can only support forward references.
403 ERROR	Cannot %s when one or more operands are relocated	Only certain arithmetic operations are supported on relative values.
404 ERROR	Relocated value not valid here; must be absolute	A relative value or relative-valued symbol or expression was used in a context where only absolute values are allowed.
405 ERROR	Cannot %s import symbols or between sections	Only certain arithmetic operations are supported on relative values and external symbols.
406 ERROR	Cannot %s expr w/ %d reloc%s and expr w/%d reloc%s	Only certain arithmetic operations are supported on relative values and external symbols.
408 ERROR	Cannot divide by zero	It is mathematically impossible to divide by zero.
413 ERROR	Unknown relational operator: "%s"	The only supported relational operators besides the regular C operators are EQ, NE, GT, LT, GE, and LE, regardless of case.
420 ERROR	Syntax error in operand	The assembler cannot recognize an operand of an instruction.
421 ERROR	Syntax error in expression	The assembler cannot recognize an expression.
422 ERROR	Cannot use a %s symbol here	An attempt was made to use a register in an expression. This is permitted only in a few specific situations.
423 ERROR	Expression must be relative (or use #immediate)	An instruction was written like MOV symbol,Rn or MOVA symbol,Rn where symbol is an absolute symbol (such as a .EQU symbol might be). To move the value of the symbol into the register, use MOV #symbol,Rn.

424 ERROR	The "%s" operator (%s) is not supported	An operator was used that is recognized but not supported, such as ++ or --.
425 ERROR	Expression result overflow: %s %d (0x%x)	An arithmetic operation resulted in overflow (loss of important bits).
426 ERROR	Expression result overflow: %d (0x%x) %s %d (0x%x)	An arithmetic operation resulted in overflow (loss of important bits).
427 ERROR	Expression result overflow: %s %u (0x%x)	An arithmetic operation resulted in overflow (loss of important bits).
453 ERROR	%s needs a literal pool before here	The source line noted in the text of the message (where the %s is above) contains a literal pool reference that cannot reach as far as the address of the source line noted at the beginning of the error message line. Insert a .POOL directive or an unconditional branch somewhere between the two source lines.

See Also

Other Resources

Renesas SH-4 Series Assembler

SH-4 Assembler Error Messages 500-555

SH-4 Assembler Error Messages 500-555

The following table shows the SH-4 assembler error messages 500-555.

Message # and Severity	Message Text	Explanation
500 ERROR	Label required here	A label was not supplied for a directive that requires one. As a result, the directive might be ignored.
501 ERROR	Illegal value for .ORG%s	An .ORG directive was used to move the location counter backward to a lower address than its current value or more than 1 MB forward. If the intent was to move the location counter forward by more than 1 MB, consider that this would occupy that amount of space in the object file, due to the MS-COFF object file format. Another approach may be best.
503 ERROR	Exported symbol "%s" not defined	The symbol named in the message was declared in a .EXPORT directive, but was not defined anywhere in the source file.
516 ERROR	This %s directive conflicts with a previous %s	An operand of a .PRINT directive conflicts with an operand of a previous .PRINT directive, or an operand of an .OUTPUT directive conflicts with an operand of a previous .OUTPUT directive.
517 ERROR	The result of this expression is not known yet	An expression was supplied whose value is not yet defined in a context where the value must be known on the assembler's first pass through the file.
519 ERROR	String is %u characters long; maximum is 255	The string(s) supplied for a .SDATAC or .SDATACB directive must be no more than 255 characters long, because the length is represented in a single byte.
520 ERROR	A string or string expression is required %s	A numeric constant or a numeric-valued symbol or expression was supplied in a context where a string is required.
521 WARN	A literal pool would never be put here anyway	A .NOPOOL directive was put in a place where the assembler would not have generated a literal pool anyway.
522 WARN	.POOL illegal in delay slot of %s	A literal pool wound up after a delayed branch instruction; the assembler automatically supplies a NOP instruction to fill the delay slot.
531 ERROR	Invalid section type "%s"	An invalid section type was specified The valid section types are CODE, DATA, DUMMY, and BSS, without regard to case.
532 ERROR	Section type (%s) must follow section name (%s)	The section type cannot be specified before the section name.
533 ERROR	Section type (%s) cannot have a value	A value cannot be specified for the section type.
534 ERROR	Invalid option "%s"	A keyword option was supplied for a directive for which it is not valid.
535 ERROR	Option "%s" cannot have a value	A value was supplied for an option that cannot have a value.

536 ERROR	Option "%s" must have a numeric value	A value was not supplied, or a non-numeric value was supplied, for an option that requires a numeric value.
537 ERROR	Alignment must be a power of 2 between 1 and 8192.	See the .ALIGN directive.
538 ERROR	"%s": MS-COFF doesn't support absolute sections	This message only occurs if an attempt is made to use a particular technology provided by other Renesas assemblers that is not supported in this assembler due to restrictions of the MS-COFF object format.
539 ERROR	"%s": MS-COFF Cannot export non-relocatable .EQU's	Only relative values may be exported; absolute symbols cannot be exported.
540 ERROR	"%s": MS-COFF Cannot export syms from DUMMY section	Symbols defined in a section of type DUMMY cannot be exported.
541 ERROR	Illegal alignment %u; must be a power of two	An alignment value must be a power of two (2^0 through 2^6).
542 ERROR	Block count too large (max %u) -- directive ignored	A block count was specified that would result in more than 4 GB of data.
543 ERROR	Cannot export "%s"; its value depends on "%s"	An .EQU symbol was exported whose value is dependent on an external (.IMPORT) symbol.
544 WARN	Code not in function; need a .ENTRY	All code should appear between an .ENTRY directive and an .ENDF directive.
545 WARN	Missing .PROLOG between .ENTRY/.ENDF for "%s"	Each function (.ENTRY and .ENDF pair) should have a .PROLOG directive inside it somewhere.
546 WARN	Missing .ENDF; closing function "%s"	All code should appear between a .ENTRY directive and a .ENDF directive.
547 ERROR	No matching .ENTRY for .ENDF	The assembler encountered a .ENDF directive that does not seem to have a matching .ENTRY directive.
548 ERROR	No active .ENTRY for .PROLOG	The assembler encountered a .PROLOG directive when there is no .ENTRY directive active (that is, not yet terminated by a .ENDF directive).
549 ERROR	.PROLOG already specified for function "%s"	A .PROLOG directive was specified more than once in a single function (.ENTRY and .ENDF pair).
550 WARN	.END encountered; ignoring files starting with %s	A .END directive was encountered in a file that was not the last file specified on the command line. The rest of the files specified on the command line will be ignored.
551 ERROR	.SECTION ASSOC= must be followed by a section name	The ASSOC option of the .SECTION directive must have a value that is a section name (including a COMDAT style section name, such as ".text{_main}").

552 ERROR	Section "%s" must be COMDAT to allow option "%s"	Either the section being declared with the current .SECTION directive or the target of an ASSOC= option of the current .SECTION directive is not a COMDAT style section.
553 ERROR	.PDATA is illegal between .ENTRY and .ENDF	It is no longer valid to specify a .PDATA directive after a .ENTRY directive and before a .ENDF directive; the .PDATA directive must now be specified before the .ENTRY to which it should apply.
554 ERROR	Prolog is too long (%u, max %u); use .BIGENTRY	The .BIGENTRY directive must be used in place of the .ENTRY directive for a function whose prolog is more than 510 bytes long.
555 ERROR	Function is too long (%u, max %u); use .BIGENTRY	The .BIGENTRY directive must be used in place of the .ENTRY directive for a function that is more than 16,777,214 bytes long.

See Also
Other Resources
Renesas SH-4 Series Assembler

SH-4 Assembler Error Messages 611-673

SH-4 Assembler Error Messages 611-673

The following table shows SH-4 assembler error messages 611-673.

Message # and Severity	Message Text	Explanation
611 ERROR	A macro name is required for .MACRO	A macro name must be specified with the .MACRO directive. In particular, the macro name goes on the right side of the .MACRO directive (in the first operand position, not in the label position).
612 ERROR	Syntax error in macro name for .MACRO	The assembler encountered something that does not look like a symbol name immediately following a .MACRO directive.
619 ERROR	Invalid macro parameter name "%s"	A macro parameter name was specified that begins with a number. This is not permitted.
622 ERROR	No matching paren for macro expansion exclusion	The assembler was not able to find a) to match a \(which it encountered in a macro expansion. The macro expansion exclusion syntax \(...) is not allowed to start on one line and end on another line; it must all occur on a single line of the macro.
624 ERROR	Too many macro parameters (max %d)	The macro used has too many parameters (operands).
631 ERROR	%s without matching %s	The .AENDI or .AENDR or .AENDW given does not seem to have a matching .AIF, .AREPEAT, or .AWHILE. This error also appears if one of these constructs is nested inside another, and the end directive for the inner one is omitted.
632 ERROR	Cannot have %s after .AELSE	An .AELSIF or an .AELSE cannot be put after an .AELSE in the same .AIF block.
633 ERROR	%s still active at end of "%s" macro expansion	An .AIF, .AREPEAT, .AWHILE, or .MACRO (macro definition) was still in progress when the end of a macro was reached. These constructs cannot span macro boundaries.
634 ERROR	%s from line %d still active at end of file %s	An .AIF, .AREPEAT, .AWHILE, or .MACRO (macro definition) was still in progress when the end of a file was reached. These constructs cannot span file boundaries.
635 ERROR	.EXITM outside of any .AREPEAT, .AWHILE, or macro	The assembler encountered an .EXITM directive that seems to be stranded without any loop or macro to exit from. This message might also be issued if a previous error caused the assembler to ignore the beginning of such a construct.
636 ERROR	.END with %s still active	The assembler encountered an .END directive while an .AIF, .AREPEAT, .AWHILE, or macro expansion was still in progress. The assembler ignores the .END directive in this case.
670 ERROR	This syntax is only legal in a macro: "%s"	A special facility was used that is only available when a macro is being expanded, for example, in a macro body).

671 ERROR	Syntax error in macro arguments	The assembler was unable to parse one or more of the arguments to a macro. It tries to continue as best it can.
672 ERROR	End of file reached while processing %s	An .AIF, .AREPEAT, .AWHILE, or .MACRO (macro definition) was still in progress when the end of a file was reached. These constructs cannot span file boundaries. Error 634 is more expected than this one.
673 ERROR	Unknown macro parameter name "%s"	A macro parameter name was referenced that is not defined for the macro currently being expanded. In the case of nested macro expansion, only the arguments of the innermost active macro are accessible.

See Also
Other Resources
Renesas SH-4 Series Assembler

SH-4 Assembler Error Messages 801-888

SH-4 Assembler Error Messages 801-888

The following table shows SH-4 assembler error messages 801-888.

Message # and Severity	Message Text	Explanation
801 WARN	Symbol %s already defined as a %s	A symbol was redeclared or redefined that has already been declared or defined.
802 ERROR	Symbol %s already defined as a %s	A symbol was redeclared or redefined that has already been declared or defined.
807 WARN	Invalid size specification "%c"	An illegal size specification (modifier) was supplied. The only legal modifiers are B, W, L, b, w, and l (lowercase L).
808 WARN	A size specification ("%c") is not allowed here	A size specification (modifier) was supplied for an instruction or directive that cannot have one.
810 WARN	Too many operands; expecting %s	More operands were supplied than can be used by the instruction or directive.
811 WARN	Label not allowed here; ignoring "%s"	A label was supplied on a directive that cannot have one. The label is ignored, and will not be defined. This can lead to further error messages.
813 WARN	Section "%s" has already been declared %s	The type or attributes of a section cannot be changed after the first time it is declared (by using the .SECTION directive).
816 WARN	Unaligned %s size data at 0x%x	A .DATA or .RES or related directive was written that is attempting to place a word at an odd address or a longword at an address that is not a multiple of 4. Be careful. Attempting to access such data in a single instruction causes a processor exception.
817 WARN	Beware of possibly unaligned code	An alignment was specified of less than 2 in a code section. This can result in code being placed at an odd address. An attempt to execute code at an odd address causes a processor exception.
818 WARN	Cannot guarantee align > section "%s"'s align (%d)	A value was specified for an .ALIGN directive that is greater than the alignment of the current section. Such an alignment cannot be guaranteed.
826 WARN	Instructions valid only in Code section; "%s" is %s	An executable or extended instruction was written in a section that is not a CODE section. This is not permitted.
827 WARN	%s invalid in a %s section (%s) -- ignored	Uninitialized data was reserved in a CODE section, or initialized data in a DUMMY or BSS section. This is not permitted.
838 ERROR	A character string may not span multiple lines	This message generally indicates a missing closing quote or the presence of an embedded quote in the string that was not doubled.

839 WARN	You may not specify a value for %s; it is ignored	A value was specified for a keyword option that cannot have a value.
840 WARN	Block count is zero -- directive ignored	A .RESB or .DATAB or .SDATAB directive was written with a zero block count. This effectively nullifies the directive.
841 WARN	%u is too large to fit in a character; using %u	A control character expression resulted in a value greater than 255.
842 WARN	"%s" is abs; expr is location, not displacement	When a branch refers to a symbol in an absolute section, the symbol is assumed to represent a location, not a displacement. Absolute sections are not supported.
851 WARN	Overriding %s from .PDATA %s	A .PDATA directive has overridden a previous .PDATA directive. This can only happen if two .PDATA directives are written in the same section without a valid .ENDF between them.
852 WARN	%s from .PDATA at %s(%d) was never used	A .PDATA directive was written that did not have a valid .ENDF following it in the same section.
870 WARN	Unaligned displacement value %d; truncating to %d	A displacement value that refers to a word was an odd number, or a displacement value that refers to a longword was not a multiple of 4. The stray least significant bits are dropped on the floor.
871 WARN	PC-relative in delay slot is branch-dest relative	When a PC-relative instruction is written in a delay slot, the PC that it is relative to is the PC of the branch destination
876 WARN	Inserted BRA and NOP before literal pool	The assembler automatically inserted a BRA to skip over a literal pool (and a NOP to fill the BRA's delay slot) because it observed that the literal pool did not follow the delay slot instruction of an unconditional branch.
880 WARN	Function end after delayed branch	The assembler encountered an .ENDF directive (or the end of the input file or section) with a branch's delay slot still unfilled. The assembler supplies a NOP to fill the delay slot.
886 WARN	.END missing -- pretending there was one	The assembler reached the end of the input without seeing an .END directive.
887 WARN	.END reached -- ignoring to EOF	The assembler encountered an .END directive but noticed that there was something more than blank lines after it. Whatever it was, the assembler ignored it.
888 WARN	Entry point not supported	The .END directive may not have an operand.

See Also
Other Resources
Renesas SH-4 Series Assembler

SH-4 Assembler Error Messages 903-998

SH-4 Assembler Error Messages 903-998

The following table shows SH-4 assembler error messages 903-998.

Message # and Severity	Message Text	Explanation
903 ERROR	%s: %s	There was some problem opening or writing to the assemble listing file.
904 ERROR	%s: %s	There was some problem opening or writing to the object file.
907 ERROR	Cannot allocate %u bytes of memory for section "%s"	Probably as a result of a .ORG directive, the assembler is unable to allocate enough memory to store the data for the section named. It will try to continue the assembly, if possible.
998 ERROR	%s	This message occurs when an .AERROR or .ERROR directive is assembled.

See Also

Other Resources

Renesas SH-4 Series Assembler

MIPS Family Processors

MIPS Family Processors

MIPS Technologies, Inc. designs high-performance, low-power, 32-bit and 64-bit reduced instruction-set computing (RISC) microprocessor architectures and cores for embedded systems. MIPS licenses technology to semiconductor companies and system OEMs.

The MIPS RISC architecture CPUs range from 50 MHz 32-bit R3000-based devices up to 266 MHZ 64-bit R5000-based CPUs.

In This Section

Intrinsic Functions for MIPS Microprocessors

Provides reference information about MIPS-specific intrinsic functions.

MIPS Compiler Options

Provides reference information about MIPS-specific compiler options.

MIPS Calling Sequence Specification

Describes registers, stack frame layout, and assemblers.

Related Sections

Differences Between Desktop and Device Compilers

Describes important differences between desktop compilers and device compilers in build options, intrinsic functions, and exception handling.

Intrinsic Functions for MIPS Microprocessors

Intrinsic Functions for MIPS Microprocessors

The functions that the MIPS device compiler recognizes are known as supported intrinsic functions. The compiler translates such intrinsic functions into a call to a C run-time (CRT) routine, or into a series of one or more instructions.

Intrinsic functions that the compiler always translates into a series of instructions are expanded functions. For example, the compiler recognizes **sqrt** as a common intrinsic function for all MIPS targets. **sqrt** is a floating-point (FP) intrinsic function which takes an FP argument and returns an FP value.

MIPS microprocessors with a FP unit support the **sqrt** instruction. That is, for MIPS microprocessors with an FP unit the compiler translates the intrinsic function **sqrt** into a **sqrt** instruction native to the microprocessor.

For MIPS microprocessors without a FP unit, the compiler translates **sqrt** into a call to a CRT routine. In this way, **sqrt** is an intrinsic function for all MIPS targets, but is expanded into an instruction only for MIPS II FP and MIPS IV FP. For more information about **sqrt**, see the Run-Time Library Reference.

The following table shows supported intrinsic functions that the compiler recognizes for specific indicated MIPS ISAs.

Function	MIPS 16 ASE	MIPS II ISA	MIPS II ISA FP unit	MIPS IV ISA FP unit
__emul	Y	Y	Y	Y
__emulu	Y	Y	Y	Y
__ll_lshift	Y	Y	Y	Y
__ll_rshift	Y	Y	Y	Y
__ull_rshift	Y	Y	Y	Y
_enable	Y	Y	Y	Y
_disable	Y	Y	Y	Y
__regsize	Y	Y	Y	Y
_InterlockedDecrement	Y	Y	Y	Y
_InterlockedExchangeAdd	Y	Y	Y	Y
_InterlockedCompareExchange	Y	Y	Y	Y
_InterlockedIncrement	Y	Y	Y	Y
_InterlockedExchange	Y	Y	Y	Y

See Also

Other Resources

Run-Time Library Reference

__emul

__emul

This function multiplies a 32-bit value in register **RT** times a 32-bit value in register **RS**, and obtains a 64-bit result.

```
__int64 __cdecl __emul(
  int1,
  int2
);
```

Parameters

int1

[in] Value in **RT** the first term in the multiplication.

int2

[in] Value in **RS**, the second term in the multiplication.

Return Values

Result of binary arithmetic.

Remarks

The compiler translates this function into the **mult** instruction.

Requirements

Routine	Required header	Architecture
__emul	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also

Reference

Intrinsic Functions for MIPS Microprocessors

__emulu

__emulu

This function multiplies a 32-bit unsigned value in register **RT** times a 32-bit unsigned value in register **RS**, and obtains a 64-bit result.

```
__int64 __cdecl __emulu(
  int1,
  int2
);
```

Parameters

int1

[in] Value in **RT**, the first term in the multiplication.

int2

[in] Value in **RS**, the second term in the multiplication.

Return Values

Result of binary arithmetic.

Remarks

The compiler translates this function into the **multu** instruction.

Requirements

Routine	Required header	Architecture
__emulu	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also

Reference

Intrinsic Functions for MIPS Microprocessors

__emul

__ll_lshift

__ll_lshift

This function shifts a 64-bit word to the left a specified number of bits.

```
__int64 __cdecl __ll_lshift(
  int64,
  int
);
```

Parameters

int64

[in] The value to shift.

int

[in] The number of bits to shift.

Return Values

None.

Remarks

The compiler translates this function into the **SLL** instruction.

Requirements

Routine	Required header	Architecture
__ll_lshift	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also

Reference

Intrinsic Functions for MIPS Microprocessors

__ll_rshift

__ll_rshift

This function shifts a 64-bit word to the right a specified number of bits.

```
__int64 __cdecl __ll_rshift(
  int64,
  int
);
```

Parameters

int64

[in] The value to shift.

int

[in] The number of bits to shift.

Return Values

None.

Remarks

The compiler translates this function into the **SRL** instruction.

Requirements

Routine	Required header	Architecture
__ll_rshift	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also

Reference

Intrinsic Functions for MIPS Microprocessors

__regsize

__regsize

This function returns the register size.

```
int __cdecl __regsize(void);
```

Parameters
None.

Return Values

The size of the target register.

Requirements

Routine	Required header	Architecture
__regsize	<stdlib.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also
Reference
Intrinsic Functions for MIPS Microprocessors

__ull_rshift

__ull_rshift

This function shifts a 64-bit unsigned word to the right a specified number of bits.

```
unsigned __int64 __cdecl __ull_rshift(
  unsigned__int64,
  int
);
```

Parameters

unsigned_int64

[in] The value to shift.

int

[in] The number of bits to shift.

Return Values

None.

Remarks

The compiler translates this function into the **SRA** instruction.

Requirements

Routine	Required header	Architecture
__ull_rshift	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also

Reference

Intrinsic Functions for MIPS Microprocessors

_enable

_enable

This function enables MIPS16 ASE.

```
void _enable(void);
```

Parameters
None.

Return Values

None.

Requirements

Routine	Required header	Architecture
_enable	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also
Reference
Intrinsic Functions for MIPS Microprocessors

_disable

_disable

This function disables MIPS16 ASE.

```
void _disable(void);
```

Parameters
None.

Return Values

None.

Requirements

Routine	Required header	Architecture
_disable	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also
Reference
Intrinsic Functions for MIPS Microprocessors

_InterlockedCompareExchange

_InterlockedCompareExchange

This function atomically compares a variable value to a comparison value and exchanges the value of *Destination* with *Exchange* if they are the same.

The function prevents more than one thread from using the same variable simultaneously.

```
InterlockedCompareExchange(
  PVOID* Destination,
  PVOID ExChange,
  PVOID Comperand
);
```

Parameters

Destination

[out] Pointer to the value that the function compares to *Comperand*.

ExChange

[in] Value used to replace contents of *Destination*, if necessary.

Comperand

[in] Standard value used for comparison to the value in *Destination*.

Return Values

Initial value of the variable pointed to by *Destination*.

Remarks

The interlocked functions provide a simple mechanism for synchronizing access to a variable that is shared by multiple threads. The threads of different processes can use this mechanism if the variable is in shared memory.

Requirements

Routine	Required header	Architecture
_InterlockedCompareExchange	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also

Reference

Intrinsic Functions for MIPS Microprocessors

_InterlockedDecrement

_InterlockedDecrement

This function performs an atomic addition of a decrement value to an addend variable. The function prevents more than one thread from using the same variable simultaneously.

```
InterlockedDecrement(
  PLONG Addend
);
```

Parameters
Addend

[in] Value to be decremented.

Return Values

Result of binary arithmetic.

Remarks

The interlocked functions provide a simple mechanism for synchronizing access to a variable that is shared by multiple threads. The threads of different processes can use this mechanism if the variable is in shared memory.

Requirements

Routine	Required header	Architecture
_InterlockedDecrement	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also
Reference
Intrinsic Functions for MIPS Microprocessors

_InterlockedExchangeAdd

_InterlockedExchangeAdd

This function performs an atomic addition of an increment value to an addend variable. The function prevents more than one thread from using the same variable simultaneously.

```
InterlockedExchangeAdd(
  PLONG Addend,
  LONG Increment
);
```

Parameters

Addend

[in, out] Value to be incremented.

Increment

[in] Value used to increment *Addend*.

Return Values

The initial value of the variable pointed to by *Addend*.

Remarks

The interlocked functions provide a simple mechanism for synchronizing access to a variable that is shared by multiple threads. The threads of different processes can use this mechanism if the variable is in shared memory.

Requirements

Routine	Required header	Architecture
_InterlockedExchangeAdd	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also

Reference

Intrinsic Functions for MIPS Microprocessors

_InterlockedIncrement

_InterlockedIncrement

This function performs an atomic addition of an increment value to an addend variable. The function prevents more than one thread from using the same variable simultaneously

```
InterlockedIncrement(
  PLONG Addend
);
```

Parameters
Addend

[in] Value to be incremented.

Return Values

None.

Remarks

The interlocked functions provide a simple mechanism for synchronizing access to a variable that is shared by multiple threads.

The threads of different processes can use this mechanism if the variable is in shared memory.

Requirements

Routine	Required header	Architecture
_interlockedIncrement	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also
Reference
Intrinsic Functions for MIPS Microprocessors

_InterlockedExchange

_InterlockedExchange

This function atomically exchanges a pair of 32-bit values. The function prevents more than one thread from using the same variable simultaneously.

```
LONG InterlockedExchange(
  LONG volatile* Target,
  LONG Value
);
```

Parameters

Target

[in, out] Pointer to the value to be exchanged. The function sets this variable to Value, and returns its prior value.

Value

[in] Value to be exchanged with the value pointed to by Target.

Return Values

The function returns the initial value of the target.

Remarks

The interlocked functions provide a simple mechanism for synchronizing access to a variable that is shared by multiple threads. The threads of different processes can use this mechanism if the variable is in shared memory.

The variable pointed to by the Target parameter must be aligned on a 32-bit boundary; otherwise, this function will fail on multiprocessor x86 systems and any non-x86 systems.

This function should not be used on memory allocated with the PAGE_NOCACHE modifier.

Requirements

Routine	Required header	Architecture
_InterlockedExchange	<winnt.h>	MIPS16, MIPSII, MIPSIII, MIPS IV, MIPS 32

See Also

Reference

Intrinsic Functions for MIPS Microprocessors

MIPS Compiler Options

MIPS Compiler Options

The following table shows compiler switches for MIPS microprocessors.

Option	Description
/QMmipsNN - Generate Code for Specific MIPS ISA	Generates code for MIPS I, II, III, IV, V, 32, and 64 Instruction Set Architectures (ISA)s.
/QMmips16 - Generate Code for MIPS16 ASE	Generates code for MIPS16 ASE.
/QMFPE - Floating Point Emulation	Enables floating-point emulation, using floating-point hardware.
/QMRnnnn - Optimize for Specific MIPS chip	Enables chip-specific inline assembler for Phillips PR3900, NEC VR4100, NEC VR4200. and NEC VR4300, and generates code for corresponding MIPS ISA.

See Also

Other Resources

MIPS Calling Sequence Specification

/QMmips16 - Generate Code for MIPS16 ASE

/QMmips16 - Generate Code for MIPS16 ASE

This option generates code for the MIPS16 Application-Specific Extension (ASE) to the MIPSII ISA. MIPS16 is especially advantageous for embedded systems where memory is of a premium.

See Also

Other Resources

MIPS Calling Sequence Specification

/QMFPE - Floating Point Emulation

/QMFPE - Floating Point Emulation

The **/QMFPE** options control compiler expectations for floating point arithmetic support.

The following table details this option.

Option	Description	Default status
/QMFPE	Turns on floating point emulation.	Default when any of the following are true:.No **/QMmipsNN** option specified -or-**/QMmips1** or the **/QMmips2** option specified.
/QMFPE-	Causes the compiler to generate instructions for a floating-point hardware unit.	Default when **/QMmips3**, **/QMmips4**, **/QMmips5**, or **/QMmips64** option specified.

See Also

Reference

/QMmipsNN - Generate Code for Specific MIPS ISA

Other Resources

MIPS Calling Sequence Specification

/QMmipsNN - Generate Code for Specific MIPS ISA

/QMmipsNN - Generate Code for Specific MIPS ISA

The following options specify which MIPS ISA the compiler generates code for.

Switch	Description	Default status
/QMmips1	Generates code for the older MIPS I instruction set.	/QMFPE is default.
/QMmips2	Generates code for the MIPS II instruction set.	/QMFPE is default.
/QMmips3	Generate code for MIPS III ISA.	/QMFPE- is default.
/QMmips4	Generate code for MIPS IV ISA.	/QMFPE- is default.
/QMmips5	Generate code for MIPS V ISA.	/QMFPE- is default.
/QMmips32	Generate code for MIPS 32 ISA	Not specified.
/QMmips64	Generate code for MIPS 64 ISA	Not specified.

See Also

Reference

/QMFPE - Floating Point Emulation

/QMRnnnn - Optimize for Specific MIPS chip

/QMRnnnn - Optimize for Specific MIPS chip

These options cause the compiler to generate code optimized for a particular MIPS ISA, and allow microprocessor-specific inline assembly instructions. The compiler may then choose instructions or sequences of instructions specific to that particular processor, and produce an executable that is incompatible with any other processor.

The following table shows the implementation of /QMRX000 switches.

Option	Equivalent to	Processor
/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

These switches and the /QMmips16 - Generate Code for MIPS16 ASE switch are mutually exclusive.

See Also

Other Resources

MIPS Calling Sequence Specification

MIPS Calling Sequence Specification

MIPS Calling Sequence Specification

The MIPS Calling Sequence Specification for provides direction for the development of compilers and assembly language programs for the MIPS microprocessor.

In addition, the standard enables the development of tools, debuggers, and operating system utilities that perform automated walking of the call stack.

In This Section

MIPS Registers

Specifies the register assignment for MIPS I, II, III, and IV architectures.

MIPS Stack Frame Layout

Describes the MIPS stack frame layout.

MIPS Prolog and Epilog

Provideds reference information about code sequences and macros needed to implement Structured Exception Handling on MIPS microprocessors

Related Sections

Differences Between Desktop and Device Compilers

Describes differences in intrinsic functions, build options, and alignment issues.

SEH in RISC Environments

Describes how structured exception handling occurs in RISC microprocessor environments.

MIPS Registers

MIPS Registers

The MIPS microprocessor has 32 general-purpose registers.

- Registers are 32 bits for MIPS I instruction set architecture (ISA) and II ISA.
- MIPS III and higher ISAs have 32-bit registers when running in 32-bit mode, and 64-bit registers when running in 64-bit mode.
- Registers $1, $26, $27, $29 are reserved for special purposes by the assembler, compiler and operating system.
- Register $0 is hard wired to the value zero, and $31 is the link register for jump and link instructions but can be used with other instructions with caution.

The following table summarizes the usage convention for these registers.

Register Name	Common Name	Description
$0	zero	Always has the value 0. Any writes to this register are ignored.
$1	at	Assembler temporary.
$2-$3	v0-v1	Function result registers. Functions return integer results in v0, and 64-bit integer results in v0 and v1 when using 32-bit registers. In cases where floating-point hardware is not present, or when compiler options enable floating-point emulation, functions return single precision floating-point results in v0 and double precision floating-point results in v0 and v1 when using 32-bit registers. v0 and v1 can be temporary registers. Not preserved across function calls.
$4-$7	a0-a3	Function argument registers that hold the first four words of integer type arguments. Functions use these registers to hold floating-point arguments. When floating-point hardware is not present, or compiler options enable floating-point emulation, functions use a0 to hold the first single precision floating-point argument and a1 to hold the second single precision floating-point argument. Functions use a0-a1 for the first double precision floating-point argument, and a2-a3 to hold the second double precision floating-point argument. Not preserved across function calls.
$8-$15, $24-$25	t0-t9	Temporary registers you can use as you want. Not preserved across function calls.
$16-$23, $30	s0-s8	Saved registers to use freely. Preserved across function calls. These registers must be saved before use by the called function.
$26-$27	k0-k1	Reserved for use by the operating system kernel and for exception return.

$28	gp	Global pointer. Not used in Windows CE and may be used as save register for called functions.
$29	sp	Stack pointer.
$31	ra	Return address register, saved by the calling function. Available for use after saving.
$f0	n/a	Function return register used to return float and double values from function calls.
($f12, $f13) and ($f14, $f15)	n/a	Two pairs of registers used to pass float and double valued parameters to functions. Pairs of registers are parenthesized because they have to pass double values. To pass float values, only $f12 and $f14 are used.

The following list contains additional information on floating-point registers:

- In MIPS ISAs I, II, and in MIPS III and up ISAs running in 32-bit mode, only $f4, $f6, $f8, $f10, $f16, and $f18 temporary registers are available.
  When manipulating these registers with double precision instructions, the high-order 32-bits are in the implied odd register. The odd registers are not directly accessible.
- Permanent registers $f20, $f22, $f24, $f26, $f28, and $f30 are registers where values are preserved across function calls.
- In MIPS architectures III and up running in 64-bit mode, the following registers are also available as temporary registers: $f1, $f3, $f5, $f7, $f9, $f11, $f17, $f19, $f21, $f23, $f25, $f27, $f29, $f31.

See Also

Reference

MIPS Stack Frame Layout

Other Resources

MIPS Calling Sequence Specification

MIPS Stack Frame Layout

MIPS Stack Frame Layout

The stack frame consists of four areas:

- Parameter or argument area
- Local variable area
- Register save area
- Argument build area

The calling function allocates space on the stack for all arguments, even though it may pass some of the arguments in registers. The calling function should reserve enough space on the stack for the maximum argument list required by calls from the calling function.

The function must allocate space for at least four words, even if it passes fewer parameters.

Functions should allocate space for all arguments, regardless of whether the function passes the arguments in registers. This provides a save area for the called function for saving argument registers if these registers need to be preserved.

The function allocates argument registers for the first argument. It allocates any argument registers remaining to the second argument, and so on until it uses all the argument registers or exhausts the argument list. All remaining parts of an argument and remaining arguments go on the stack.

The argument register allocation preserves that same alignment as if in memory. When mapping some argument lists some argument registers may not contain anything relevant the same as padding space in memory.

See Also

Other Resources

MIPS Calling Sequence Specification

MIPS Prolog and Epilog

MIPS Prolog and Epilog

Prolog and epilog code segments are required to implement SEH for MIPS microprocessors.

The MIPS prolog contains code that sets up the stack frame for a routine, and the epilog contains the code that removes the routine's frame and then returns to the calling function. The MIPS compiler generates prolog and epilog code to perform these tasks, but you must write the prolog and epilog code for any assembly code functions that you write.

In This Section

Related Sections

MIPS Prolog

MIPS Prolog

The MIPS prolog has several immediately contiguous parts, regardless of whether MIPS 16-bit mode, MIPS 32-bit mode, or MIPS 64-bit mode is in force.

The following steps show the required elements of a MIPS prolog.

The code examples shown apply to MIPSII. For MIPS IV, replace the **sw** instruction with **sd**, and replace the **lw** instruction with **ld**.

1. Define the prolog and set up the entry.

```
.ent <routine_name>
<routine_name>:
```

2. Reserve space for the stack frame.

```
Addiu sp, -<frame size>
```

An extended **addiu** instruction of four bytes is generated for frame sizes greater than 1024.
If the frame size exceeds 32768, update the stack pointer with a constant from the literal pool using the following sequence of instructions.

```
hw  $3, <frame size> constant offset(pc)
move  $2, $29
subu  $2, $2, $3
move  $29, $2
```

The size of the stack frame must be a multiple of eight. This includes space for local variables and temporaries, saved registers, and a procedure call argument area for non-leaf routines.
Any routine that uses registers $16-$23, or $30, or any floating-point register that the called function saves, must save these registers.
The procedure call argument area contains the maximum number of bytes required for the arguments of any procedure called in a non-leaf routine. This number of required bytes includes those arguments that the function can pass in registers, unless the N32 calling convention is in use.

3. Set up a virtual frame pointer. The virtual frame pointer is **sp**($29) added to the frame size.

```
.frame framereg (usually $29), framesize, returnreg (usually $31)
```

4. Set a bit in the bitmask for each general-purpose register saved. Set bits in little endian order. The frame offset is the offset, a negative number, from the virtual frame pointer where the register save area begins.

```
.mask mask, <frame offset>
```

5. Store any registers that need to be saved.
For example, if **ra** needs to be saved:

```
sw ra, <frame size> + <frame offset>(sp)
```

If subsequent lower number registers need to be saved, the offset depends on the MIPS mode. If the next register is a MIPS16 register,

```
sw  $<mips16 register>, <frame size> + <frame offset> - n(sp)
```

Otherwise

```
move  $2, <mipsii register>
sw  $2, <frame size> + <frame offset> - n(sp)
```

Where N is four and is incremented by four for each subsequent lower number register to be saved.
If the <frame size> + <frame offset> is greater than 32767, then the following 3 instruction sequence is required before executing the register save loop:

```
lw  $2, <frame size> + <frame offset>constant offset(pc)
move  $2, $29
addu  $3, $2, $3
```

The **sw** instructions depend on the mode of the MIPS register. If the next register is a MIPS16 register:

```
sw  $<mips16 register>, -n($3)
```

Otherwise

```
move  $2,$<mipsii register>
sw  $2, -n($3)
```

6. Mark the end of the prolog.

```
.prologue 0
```

See Also
Reference
MIPS Assembler Macros
Other Resources
MIPS Prolog and Epilog

MIPS Epilog

MIPS Epilog

While each procedure has only one prolog, a procedure may contain any number of epilogs if the procedure uses multiple exit points. Each epilog is required to have certain specific parts. All parts are contiguous with no intervening instructions.

The following steps shows how to restore the registers saved by the called function for a MIPS ISA. For MIPS IV, replace the **lw** instruction with **ld.**

Note that MIPS16 epilogs employ slightly different guidelines to restore the registers saved by the called function.

1. Issue a restore for each register saved in the prologue

```
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
```

   Where N is four and incremented by four for each subsequent lower number register saved.

2. Return from the procedure.

```
j $31
```

3. End the routine.

```
.end routine_name
```

   The minimum proper epilog for a leaf routine includes the return jump and the **.end.** Additionally, for a nonleaf routine, loading register $31 is required.

```
jr scratch
```

See Also

Reference

MIPS Assembler Macros

Other Resources

MIPS Prolog and Epilog

MIPS Assembler Macros

MIPS Assembler Macros

Assembler-level macros isolate the programmer from the details of assembler directives.

The following table shows the macros defined for MIPS microprocessors.

Macro	Description
ALTERNATE_ENTRY (MIPS)	Declares an alternate entry to a routine
EXCEPTION_HANDLER (MIPS)	Associates a named exception handler with the subsequent **NESTED_ENTRY**
LEAF_ENTRY (MIPS)	Declares the beginning of a routine that does not require any prolog code
NESTED_ENTRY (MIPS)	Declares the beginning of a routine that has an existing stack frame or creates a new stack fram e
PROLOGUE_END	Marks the end of the prolog area

See Also

Other Resources

MIPS Calling Sequence Specification

ALTERNATE_ENTRY (MIPS)

ALTERNATE_ENTRY (MIPS)

This macro declares an alternate entry to a routine of type NESTED_ENTRY (MIPS) or LEAF_ENTRY (MIPS).

```
ALTERNATE_ENTRY Name[,
[Section=]SectionName]
```

Parameters

Name

Name is the entry point and is in the global name space.

SectionName

SectionName is the name of the section in which the entry will appear; see Remarks.

Return Values

None.

Remarks

The **ALTERNATE_ENTRY** macro does not use the *SectionName* parameter. The parameter is accepted and ignored for consistency with **NESTED_ENTRY (MIPS**) and **LEAF_ENTRY (MIPS)**. If used, an **ALTERNATE_ENTRY** call must appear in the body of a routine.

See Also

Reference

NESTED_ENTRY (MIPS)

LEAF_ENTRY (MIPS)

EXCEPTION_HANDLER (MIPS)

EXCEPTION_HANDLER (MIPS)

This macro associates an exception handler *Handler* with a subsequent NESTED_ENTRY (MIPS) or LEAF_ENTRY (MIPS).

```
EXCEPTION_HANDLER Handler
```

Parameters
Handler

Name of exception handler.

Return Values

None.

See Also
Reference
NESTED_ENTRY (MIPS)
LEAF_ENTRY (MIPS)

LEAF_ENTRY (MIPS)

LEAF_ENTRY (MIPS)

This macro declares the beginning of a routine that does not require any prolog code.

```
LEAF_ENTRY Name[,[Section=]SectionName]
```

Parameters

Name

Name is the routine name and is in the global name space.

SectionName

SectionName is the name of the section in which the entry will appear; it is optional and defaults to .text.

Return Values

None.

Remarks

A LEAF_ENTRY must have an associated PROLOGUE_END.

See Also

Reference

NESTED_ENTRY (MIPS)
EXCEPTION_HANDLER (MIPS)
PROLOGUE_END

NESTED_ENTRY (MIPS)

NESTED_ENTRY (MIPS)

This macro declares the beginning of a routine that has an existing frame or that creates a stack frame.

```
NESTED_ENTRY Name[,[Section=]SectionName]
```

Parameters

Name

Name is the routine name and is in the global name space.

SectionName

SectionName is the name of the section in which the entry will appear; it is optional and defaults to. text.

Return Values

None.

Remarks

A NESTED_ENTRY must have an associated PROLOGUE_END.

See Also

Reference

LEAF_ENTRY (MIPS)
EXCEPTION_HANDLER (MIPS)
PROLOGUE_END

PROLOGUE_END

PROLOGUE_END

This macro marks the end of the prolog area for the MIPS microprocessor family.

```
PROLOGUE_END
```

Parameters
None.

Return Values

None.

Remarks

This macro must appear following a NESTED_ENTRY (MIPS) or LEAF_ENTRY (MIPS) macro.

See Also
Reference
NESTED_ENTRY (MIPS)
LEAF_ENTRY (MIPS)

MIPS Assembler

MIPS Assembler

Support for MIPS-licensed microprocessor includes a standalone assembler, inline assembly, and assembler macros.

In This Section

Stand-alone MIPS Assembler

Provides a brief description of the stand-alone MIPS assembler.

MIPS Inline Assembly Language

Describes the conventions of MIPS inline assembly

Related Sections

MIPS Family Processors

Provides an overview of key functionality, compiler options, and intrinsic functions.

Stand-alone MIPS Assembler

Stand-alone MIPS Assembler

The MIPS edition of Visual C++ includes a stand-alone assembler.

The assembler does not have all of the restrictions that inline assembly does; in particular, using the stand-alone assembler is the only way to create functions written only in assembly code.

However, in some respects the stand-alone assembler is less convenient, because you cannot refer as easily to objects in C and C++ source code.

MIPS stand-alone assembler driver is Mipsasm.exe. Mipsasm recognizes assembly language source files that end with .asm or .s file extensions.

The following code example shows how to assemble and link prog.asm into an executable in one step from the command line.

```
mipsasm -DMIPS -QMmips2 prog.asm /link /entry:mainACRTStartup
    /subsystem:windowsce /nodefaultlib corelibc.lib coredll.lib
```

The next code example shows how to assemble an assembly language source file into an object file without linking, using the "-c" assembler switch.

```
mipsasm -DMIPS -QMmips2 -c prog.asm
```

To see more assembler options, type the following at the command line:

```
mipsasm /?
```

See Also

Concepts

MIPS Inline Assembly Language

Other Resources

MIPS Calling Sequence Specification

MIPS Inline Assembly Language

MIPS Inline Assembly Language

Inline assembly language allows you to embed assembly-language instructions in your C and C++ source programs without extra assembly and link steps.

The inline assembler is built into the compiler and does not require separate assembly.

Because the inline assembler requires no extra steps, it is sometimes more convenient than using a separate assembler, especially in situations where you require access to processor resources.

Inline assembly code disables compiler optimization for the inline assembler function. If optimization is important for your program, you should consider using intrinsic functions.

To use inline assembly code, you must select the appropriate compiler switches for the specific Instruction Set Architecture (ISA).

For more information about these switches, see /QMmipsNN - Generate Code for Specific MIPS ISA.

Restrictions

The string specified in the **__asm** statement can be any valid assembly code accepted by the stand-alone assembler. You can use any of the pseudo-ops supported, as long as you do not use pseudo-ops to define separate procedures or functions.

To create complete assembly-language procedures, use the stand-alone assembler, as explained in Stand-alone MIPS Assembler.

See Also

Reference

MIPS Compiler Options

Other Resources

MIPS Calling Sequence Specification

_asm Keyword in MIPS Inline Assembly

_asm Keyword in MIPS Inline Assembly

The **_asm** keyword invokes the inline assembler, and can appear anywhere in a C or C++ statement. The compiler treats the **_asm** statement as an intrinsic function call. You must include a **#pragma** intrinsic statement that declares an **_asm** keyword.

The following C++ code shows how to declare _asm with a **#pragma** directive.

```
extern "C" {
void _asm (char*,...);
};
#pragma intrinsic (__asm)
```

It is not necessary to perform these declarations in C source modules.

In an **_asm** declaration, the string argument is a null terminated character string consisting of one or more assembly language statements. The compiler passes any subsequent arguments to the assembly code described by the first string argument.

The following example generates instructions to multiply 4 and 5, and store the result in the **t0** register. In this example, the reference to **%0** refers the value "4" to the first argument in the generated assembly code. **%1** refers the value "5" to the second argument in the generated assembly code.

```
__asm("mul t0, %0, %1", 4, 5);
```

Within the string argument of the **_asm** prototype, an embedded semicolon (;) or newline (\n) is used to separate multiple assembly language statements. The following example shows how to use a semicolon to signify multiple assembly language statements.

```
__asm(".set noat;"
"mult $1, $4, $5; "
".set at ");
```

Loading function addresses with inline assembly

The MIPS inline assembler uses the pseudo-op **la** to load the address of a function to a register. When compiling with /Og (Global Optimizations)**,** this causes the compiler to generate the following error message:

```
Error C2759: inline assember reports function literal used with call optimization
```

If you want to compile your code with /Og (Global Optimizations), you will need to replace the **la** pseudo-op with with other constructions that do not cause a compiler error.

The following code example shows such a replacement.

- Replace this code construction

  ```
  __asm("la v0, f");
  ```

  with the following code construction:

  ```
  __asm("move v0,%0", f);
  ```

See Also

Concepts

MIPS Inline Assembly Language

MIPS _asm Statement Registers

MIPS _asm Statement Registers

The following table gives MIPS macro names and associated register descriptions.

MIPS16 macro name	MIPSII macro name	Associated Register	Description
zero	**zero**	**$0**	Always zero; writes to this register are ignored.
N/A	**AT**	**$1**	Assembler temporary.
v0	**v0**	**$2**	Used to hold return value.
v1	**v1**	**$3**	Used to hold return value.
a0	**a0**	**$4**	Argument registers; used to pass first four words of integer arguments.
a1	**a1**	**$5**	Argument registers; used to pass first four words of integer arguments.
a2	**a2**	**$6**	Argument registers; used to pass first four words of integer arguments.
a3	**a3**	**$7**	Argument registers; used to pass first four words of integer arguments.
N/A	**t0**	**$8**	Temporary registers; may be freely changed.
N/A	**t1**	**$9**	Temporary registers; may be freely changed.
N/A	**t2**	**$10**	Temporary registers; may be freely changed.
N/A	**t3**	**$11**	Temporary registers; may be freely changed.
N/A	**t4**	**$12**	Temporary registers; may be freely changed.
N/A	**t5**	**$13**	Temporary registers; may be freely changed.
N/A	**t6**	**$14**	Temporary registers; may be freely changed.
N/A	**t7**	**$15**	Temporary registers; may be freely changed.
N/A	**s0**	**$16**	Saved registers; must be preserved across function calls.
N/A	**s1**	**$17**	Saved registers; must be preserved across function calls.
N/A	**s2**	**$18**	Saved registers; must be preserved across function calls.
N/A	**s3**	**$19**	Saved registers; must be preserved across function calls.
N/A	**s4**	**$20**	Saved registers; must be preserved across function calls.

N/A	**s5**	**$21**	Saved registers; must be preserved across function calls.
N/A	**s6**	**$22**	Saved registers; must be preserved across function calls.
N/A	**s7**	**$23**	Saved registers; must be preserved across function calls.
t8	**t8**	**$24**	Additional temporary registers.
N/A	**t9**	**$25**	Additional temporary registers.
N/A	**k0**	**$26**	Kernel reserved registers.
N/A	**k1**	**$27**	Kernel reserved registers.
N/A	**gp**	**$28**	Global pointer.
sp	**sp**	**$29**	Stack pointer.
N/A	**s8**	**$30**	Additional saved register.
ra	**ra**	**$31**	Return address register.

The register macro names are interchangeable with their numeric names using a dollar sign prefix. For example, the following two statements are equivalent:

```
__asm("add v0, a0, $16");
__asm("add $2, $4, s0");
```

See Also

Concepts

_asm Keyword in MIPS Inline Assembly

MIPS Assembler Error Messages

MIPS Assembler Error Messages

This section contains descriptions of the MIPS Assembler error messages.

In This Section

MIPS Error Messages 1-525

MIPS Error Messages 1-525

The following table shows MIPS error messages number 1 through 525.

No.	Severity	Message text
1	FATAL	INTERNAL ASSEMBLER ERROR (assembler file '%s', line %d)
1	ERROR	INTERNAL ASSEMBLER ERROR (assembler file '%s', line %d)
2	FATAL	Assembler is out of heap space
3	FATAL	INTERNAL ASSEMBLER ERROR: Insufficient memory (assembler file '%s', line %d)
6	FATAL	Invalid error number: %s(%d, ...)
8	FATAL	No input file specified
9	FATAL	Image file '%Fs' is not a PE image file
10	FATAL	Invalid Coff object '%Fs': swapped magic = 0x%4x
12	FATAL	Symbol '%Fs' not found during disassembly of '%Fs'
13	FATAL	Invalid EOF reading file '%Fs'
15	WARN	Unable to reach line number %d of file '%Fs'
18	WARN	Line number %d is greater than maximum of %d for file '%Fs
20	ERROR	Local label number is out of range
32	FATAL	%s (%d): WIN32 '%s' failed; %s
32	WARN	%s (%d): WIN32 '%s' failed; %s
63	FATAL	Unknown option '%c' in '%s'
83	FATAL	Cannot open %Fs file: '%Fs': %Fs
83	WARN	Cannot open %Fs file: '%Fs': %Fs
83	ERROR	Cannot open %Fs file: '%Fs': %Fs
84	WARN	Cannot read %Fs file: '%Fs': %Fs
84	ERROR	Cannot read %Fs file: '%Fs': %Fs
85	FATAL	Cannot write %Fs file: '%Fs': %Fs
110	ERROR	Warnings treated as error - no object file generated

112	FATAL	Cannot unlink %Fs file: '%Fs': %Fs
120	FATAL	Product ID mismatch between '%s' version '%ld' and '%s' version '%ld'
213	ERROR	Invalid numerical argument '%s'
213	FATAL	Invalid numerical argument '%s'
418	ERROR	Extended-precision floating point not supported
502	FATAL	%s (%d): Unexpected opcode '%d'
503	FATAL	%s (%d): .ent or .aent not aligned on word boundary
504	FATAL	%s (%d): tried to unput over buffer boundary
505	FATAL	%s (%d): not expecting arg size of %d
506	ERROR	Obsolete or corrupt binasm '%s'
507	FATAL	Exceeded 64k relocation entries. Recompile with /Gy
508	ERROR	Illegal branch-label '%s' reference at line (%d): branch and label are in different sections
511	ERROR	Redefinition of symbol '%s'
512	ERROR	.repeat only valid in assembler
513	ERROR	.endr only valid in assembler
514	INFO	Unsupported option type (%d) in binasm
515	ERROR	.noalias and .alias require gp registers
516	ERROR	.cpload must be inside noreorder
517	ERROR	.half not on halfword boundary
518	ERROR	.word not on word boundary
519	ERROR	.dword not on double-word boundary
520	ERROR	.float or .double not on double-word boundary
521	WARN	NOP required for mtc0/mfc0
522	ERROR	Both .cpalias and .cprestore used in the same .ent/.end pair
523	ERROR	.cpalias registers do not match between .ent/.end pair
524	ERROR	.cprestore offsets do not match between .ent/.end pair
525	ERROR	.cprestore or .cpalias can only be placed in text

See Also

Reference

MIPS Error Messages 526-610
MIPS Error Messages 611-660
MIPS Error Messages 661-910
MIPS Error Messages 911-980

Other Resources

MIPS Family Processors

MIPS Error Messages 526-610

MIPS Error Messages 526-610

The following table shows MIPS error messages number 526 through 610.

No.	Severity	Message text
526	ERROR	Code can only be placed in text
527	ERROR	Bad instruction opcode
528	INFO	Preparing gp-tables
529	INFO	Building gp-tables
530	ERROR	Label referenced but not defined: %s
531	ERROR	Not all branch-label symbols were defined
532	ERROR	C0 opcode must be inside .set noreorder section
533	ERROR	Byte address of jump target must fit in 28 bits
534	ERROR	Jump target is not word-aligned
536	ERROR	Register must differ from base
537	ERROR	Operand 1 should be fp reg
538	ERROR	ulwu instruction not implemented
539	ERROR	coproc doubles not implemented
540	ERROR	li.e instruction not implemented
542	ERROR	Exponent out of range: %s
543	ERROR	Only \\""1\\"" or \\""0\\"" allowed left of binary point: %s
544	ERROR	Illegally denormalized fraction: %s
545	ERROR	Floating point underflow: %s
546	ERROR	-M forbidden when assembler runs on a MIPS machine
547	ERROR	-M does not support \\"".extended\\""
548	ERROR	-M does not support hex floating point
549	ERROR	Floating point literal too long
550	ERROR	Too many float literals â€" compile with \\""-Wb,-nopool\\""

551	ERROR	Cannot label a pseudo-op in text section
552	ERROR	Shift amount not 0..31
553	ERROR	Shift amount not 0..63
554	ERROR	gp-relative segments together exceed 64k bytes: %d bytes
558	ERROR	MIPS16 move must use a MIPSII gp register
559	ERROR	MIPS16 addiu immediate limited to 15 bits
560	ERROR	Doubleword opcode not supported by mips2 or earlier
575	ERROR	MIPS16 load with symbolic offset must use different base & dest
576	ERROR	MIPS16 store may not use symbolic offset in base-offset mode
577	ERROR	Non-MIPS16 instruction used in MIPS16 text section
578	ERROR	MIPS16 instruction used in non-MIPS16 text section
579	ERROR	2nd register invalid in MIPS16 dsrl and dsra
580	ERROR	Branch offset not specified
581	ERROR	Relative offset not multiple of 4
582	ERROR	Relative offset beyond 32768
583	ERROR	cprestore offset must be positive and divisible by 4
584	ERROR	.cpalias requires a register argument
585	ERROR	Number out of range: %s
586	ERROR	MIPS16 general purpose register expected
587	ERROR	2-register MIPS16 jalr must take $ra as first operand
602	ERROR	Assembler op/directive expected
603	ERROR	Undefined symbol in expression
604	ERROR	Invalid symbol in expression: %s
605	ERROR	Symbol must have absolute value: %s
607	ERROR	No such label
608	ERROR	Too many local labels
609	ERROR	Overflow

610	WARN	Large decimal set sign bit

See Also

Reference

MIPS Error Messages 1-525

MIPS Error Messages 611-660

MIPS Error Messages 661-910

MIPS Error Messages 911-980

Other Resources

MIPS Family Processors

MIPS Error Messages 611-660

MIPS Error Messages 611-660

The following table shows MIPS error messages number 611 through 660.

No.	Severity	Message text
611	ERROR	Badly delimited numeric literal
612	ERROR	Badly delimited hexadecimal literal
613	ERROR	Hex digit in decimal literal
614	ERROR	Hex floating point literal too long
615	ERROR	Missing exponent in floating-point literal
616	ERROR	Truncating token
618	ERROR	Literal string not terminated
619	ERROR	Literal string too long
620	ERROR	Number in string too large
621	ERROR	Missing \\"" at end of string
622	ERROR	Expected cpp-generated line number
623	ERROR	Expected cpp-generated file name
624	WARN	Truncating cpp-generated filename
625	ERROR	Section not declared for symbol '%s'
626	ERROR	Address symbol expected
627	ERROR	Register expected
628	ERROR	Undefined symbol '%s'
629	ERROR	')' expected: %s
630	ERROR	String within expression may have only one character: %s
631	ERROR	Immediate value out of range
632	ERROR	Conflicting definition of symbol '%s'
633	ERROR	Should be gp register
634	ERROR	Should be even gp register

635	ERROR	Should be coprocessor register
636	ERROR	Should be even coprocessor register
637	ERROR	Should be floating point register
638	ERROR	Should be even floating point register
639	ERROR	Should be multiple-of-2 register
640	ERROR	Should be multiple-of-4 register
641	ERROR	Should be multiple-of-4 floating point register
642	ERROR	Should be fp double or gp single register
643	INFO	No global in text_localize(%d) '%s'
644	ERROR	Expression required
645	ERROR	Break operand out of range
646	ERROR	Ill-formed symbolic expression
647	ERROR	Bad id in expression
648	ERROR	Anticipating a string for .dword's value
649	ERROR	Invalid memory tag
650	FATAL	%s (%d): .text name mismatch, '%s' : '%s'
651	ERROR	Invalid external expression
652	ERROR	Cache operation has invalid value
653	ERROR	Floating point constant expected
654	ERROR	2nd register not allowed in non-mips3 architecture
655	ERROR	lui expression not in to 65535 range
656	ERROR	Invalid syntax in statement
657	ERROR	Shift value not 0 to 31 (inclusive)
658	ERROR	Shift value not 0 to 63 (inclusive)
659	ERROR	Operation requires immediate operand
660	ERROR	Immediate operand not 0 to 65535

See Also
Reference

MIPS Error Messages 1-525
MIPS Error Messages 526-610
MIPS Error Messages 661-910
MIPS Error Messages 911-980

Other Resources

MIPS Family Processors

MIPS Error Messages 661-910

MIPS Error Messages 661-910

The following table shows MIPS error messages number 661 through 910.

No.	Severity	Message text
661	ERROR	Immediate operand not -32768 to 32767
662	ERROR	Immediate operand not allowed on fp
663	ERROR	Statement extends past logical end
664	ERROR	Should be floating point CC reg
665	ERROR	Invalid symbol for address
666	ERROR	Label expected
667	ERROR	Symbol is not a label
668	ERROR	Label or number expected
669	ERROR	Cannot find symbol
670	ERROR	.shift_addr expression not 1
671	ERROR	.align expression not to 12
672	ERROR	String literal expected
673	ERROR	Byte value must be -128 to 255
674	ERROR	Identifier expected
675	ERROR	Number expected
676	ERROR	Invalid symbol for .comm/.lcomm/.extern
677	ERROR	Undefined assembler operation
678	ERROR	.err directive encountered
680	ERROR	Invalid symbol for .[a]ent: %s
681	ERROR	Invalid symbol for .globl: %s
682	ERROR	Invalid section name: %s
683	ERROR	Invalid id in expression
684	ERROR	Value must be -32768 to 65535

685	ERROR	.repeat not allowed in struct
686	ERROR	.repeat may not nest
687	ERROR	.endr without preceding .repeat
688	ERROR	Reserved name used as label
689	ERROR	Colon must follow a local label
690	ERROR	Malformed statement
691	ERROR	%hi/%lo/%gprel not followed by (
692	WARN	Invalid expression with %hi/%lo: %s
693	ERROR	Expected \\""(\\"" before base register
694	ERROR	Expected \\"")\\"" after base register
695	ERROR	Macro expansion needs at register after .set noat
696	ERROR	Code field in trap instruction not 0..1023
697	ERROR	+= expected after .
698	ERROR	.text block missing .ent
699	ERROR	.end without matching .ent
897	WARN	Opcode used without -QMmips64 or -QMViper
898	WARN	MIPSII opcode used without -QMmips32
899	WARN	Opcode used without -QMmips32 or -QMmips4
900	WARN	Opcode used without -QMmips32 or -QMmips64 or -QMViper
901	ERROR	Shift value not 32 to 63 (inclusive)
905	ERROR	Should be vector
906	ERROR	Should be debug reg
907	WARN	MIPS V opcode used without -QMmips5
908	WARN	Mips viper opcode used without -QMViper
909	WARN	Mips R5400 opcode used without -QMR5400
910	WARN	Using unimplemented unaligned-access opcode

See Also
Reference

Other Resources

MIPS Error Messages 911-980

MIPS Error Messages 911-980

The following table shows MIPS error messages number 911 through 980.

No.	Severity	Message text
911	WARN	Zero length string
912	WARN	Register alu instruction converted to immediate instruction
916	WARN	opcode used without /QMRnnnn where nnnn=3900, 32, or 64
917	WARN	Floating point opcode used with -QMFPE
918	WARN	Mark II opcode not available on R4100 used with -QMR4100
919	WARN	Mark II opcode not available on all Windows CE architectures
920	WARN	Mark II opcode not available on R3910 used with -QMR3910
922	WARN	Mips R4100 opcode used without -QMR4100
923	WARN	wait instruction used without /QMmips32, /QMmips64, or /QMViper
924	WARN	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
926	WARN	Length of .lcomm was less than 1: %s
927	WARN	Errata 52: DIV in branch delay slot may not work on a R4000 chip

928	WARN	Division by zero
929	WARN	Cross-assembler ignores -fli
930	WARN	Decimal .extended limited to 55 bit precision: %s
931	WARN	Floating under/over-flow in conversion to binary
932	WARN	Floating exception in conversion to binary
932	WARN	Floating exception in conversion to binary
933	WARN	Cannot correct .e alignment
934	WARN	Cannot do floating-point load on unaligned data
935	WARN	Improperly aligned register
936	WARN	sData offset exceeded $gp padding threshold
937	WARN	Extern offset exceeded $gp padding threshold
938	WARN	Bss offset exceeded $gp padding threshold
939	WARN	Branch target is not word-aligned
940	WARN	Mark II opcode used without -QMmips2

941	WARN	Mark III opcode used without -QMmips3
942	WARN	MIPS IV opcode used without -QMmips4
943	WARN	Too many file names '%s'
944	WARN	Unknown option '%s'
945	WARN	Macro instruction used
946	WARN	Macro instruction used in branch delay slot
947	WARN	.set nomove is obsolete
948	WARN	nomacro requires noreorder
949	WARN	reorder requires macro
950	WARN	No code generated for '%Fs'
951	WARN	.loc should precede .ent for /Od
952	ERROR	Cannot handle misaligned 64 bit const
953	WARN	nop must be inside .set noreorder section
954	WARN	Macro instruction used t

956	WARN	Symbol %s is not in comdat section yet it has the comdat attribute
964	WARN	Image file '%Fs' has unfamiliar optional header size %d
972	ERROR	duplicate public symbol '%s' defined for COMDAT '%s' (-Gy), linked image may not run
972	WARN	duplicate public symbol '%s' defined for COMDAT '%s' (-Gy), linked image may not run
975	WARN	Directive not implemented
976	WARN	$31 not allowed in conditional branch and link
977	ERROR	Coprocessor operation > 25 bits
978	WARN	Extra filename on command line
979	WARN	Line too long
980	WARN	Used t without .set noat
981	Warn	Cannot redefine symbol
982	Warn	Load/store of an undefined symbol: %s
983	Warn	JAL should not use same register twice
984	Warn	JAL should not use $31 alone or any register twice

985	Warn	Optional argument not S, ignored
986	Warn	.option name expected
987	Warn	.ent missing preceding .end
988	Warn	.end missing at end of assembly
989	Error	.ent/.end block never defined the procedure name
990	Warn	.aent must be inside .ent/.end block
991	Warn	sym directive not implemented
992	Warn	.line directive not implemented
993	Warn	.dead directive not implemented
994	Warn	Unknown name in .option
995	Warn	.set option expected
996	Warn	Unknown option in .set
997	Warn	Hint field not in range 0..31
998	Warn	Multiply accumulate instruction used without /QMRnnnn - Optimize for Specific MIPS chip, where nnnn = 4121, Viper, 5400, 32, or 64.

See Also
Reference

Other Resources

设备的用户界面参考

无论是用于桌面设计还是设备设计，大多数 Visual Studio 用户界面功能都是相同的。您可以在常规用户界面元素 (Visual Studio) 的帮助主题中找到相关信息。

本节中的主题描述专为开发设备应用程序而创建的界面元素。

本节内容

所有设备项目通用

"配置仿真引导程序"对话框

"配置 TCP/IP 传输"对话框

"连接到设备"对话框

"部署"对话框(设备)

"设备属性"对话框

"选项"对话框 ->"设备工具"->"设备"

"选择证书"对话框(设备)

托管设备项目

"更改目标平台"对话框(设备)

"连接到 <数据库>"对话框

文件属性对话框中的连接字符串属性(设备)

项目设计器 ->"调试"窗格

"定义颜色"对话框(设备)

"工具箱"->"设备控件"选项卡

"项目设计器"->"设备"窗格

"字体编辑器"对话框(设备)

"外观设置属性"对话框(设备)

"选项"对话框 ->"设备工具"->"外观设置"

"选项"对话框 ->"设备工具"->"常规"

"输出文件文件夹"对话框(设备)

"智能设备托管项目"->"属性"窗口

本机设备项目

"MFC 智能设备应用程序向导"的"高级功能"

"ATL 智能设备项目向导"的"应用程序设置"

"MFC 智能设备 ActiveX 控件向导"的"应用程序设置"

"MFC 智能设备 DLL 向导"的"应用程序设置"

"Win32 智能设备项目向导"的"应用程序设置"

"MFC 智能设备应用程序向导"的"应用程序类型"

ATL 智能设备项目向导

"MFC 智能设备 ActiveX 控件向导"的"控件名称"

"MFC 智能设备 ActiveX 控件向导"的"控件设置"

"<Projectname> 属性页"对话框 ->"配置属性"->"调试"(设备)

"<Projectname> 属性页"对话框 ->"配置属性"->"部署"(设备)

"MFC 智能设备应用程序向导"的"文档模板字符串"

"<Projectname> 属性页"对话框 ->"配置属性"->"Authenticode 签名"->"常规"(设备)

"MFC 智能设备应用程序向导"的"生成的类"

MFC 智能设备 ActiveX 控件向导

MFC 智能设备应用程序向导

MFC 智能设备 DLL 向导

"ATL 智能设备项目向导"的"平台"

"MFC 智能设备 ActiveX 控件向导"的"平台"

"MFC 智能设备应用程序向导"的"平台"

"MFC 智能设备 DLL 向导"的"平台"

"Win32 智能设备项目向导"的"平台"

"MFC 智能设备应用程序向导"的"用户界面功能"

Win32 智能设备项目向导

设备安装项目

"部署项目属性"对话框 ->"配置设置"->"生成"

"<Projectname> 属性页"对话框 ->"配置属性"->"部署"(设备)

"智能设备 Cab 项目"->"属性"窗口

相关章节

常规用户界面元素 (Visual Studio)

说明在 Visual Studio 对话框和窗口中出现的选项。

添加/修改连接(设备)

本主题已针对 Visual Studio 2005 SP1 进行了更新。

指定数据源和数据提供程序，并提供测试连接的方式。

下表已针对 Visual Studio 2005 SP1 进行了更新。

项	说明
“数据源”框	指定数据源和数据提供程序。
更改	打开“更改数据源”对话框，可以在此选择数据源和数据提供程序。
“数据源”单选按钮	指定数据源是位于开发计算机上，还是位于 ActiveSync 所连接的设备上。
数据库	指定数据库的文件名。
创建	打开“创建新数据库”对话框，可以在此指定新数据库的文件名和密码。
浏览	打开“选择数据库文件”对话框，可以在此选择现有的 SQL Server Mobile Edition 或 SQL Server Compact Edition 数据库。
密码	指定数据库的密码
高级	打开“高级属性”对话框，可以在此修改数据库文件的名称和路径，以及指定密码。
测试连接	测试“数据库”框中指定的数据库连接。

请参见

其他资源

设备的用户界面参考

“MFC 智能设备应用程序向导”的“高级功能”

描述“MFC 智能设备应用程序向导”的“高级功能”页。

此页为应用程序提供了附加功能选项，如 ActiveX 控件和 Windows 套接字。在每个部分，指定这些高级功能的附加支持。

高级功能

Windows 帮助

在此版本中不支持此功能。

打印和打印预览

在此版本中不支持此功能。

通过从 MFC 库调用 CView 类中的成员函数，生成处理打印、打印设置和打印预览命令的代码。向导也向应用程序的菜单添加这些函数的命令。打印支持仅对在向导的应用程序类型页中指定“文档/视图结构支持”的应用程序可用。默认情况下，文档/视图应用程序有打印支持。

ActiveX 控件

支持 ActiveX 控件（默认值）。如果不选择此选项，以后需要向项目插入 ActiveX 控件时，必须在应用程序的 InitInstance 成员函数中添加 AfxEnableControlContainer 调用。

Windows 套接字

在此版本中不支持此功能。

支持 Windows 套接字。Windows 套接字允许编写通过 TCP/IP 网络通信的应用程序。

最近文件列表上的文件数

在此版本中不支持此功能。

指定最近使用的列表中列出的文件数。默认数是 4。

请参见

参考

MFC 智能设备应用程序向导

“ATL 智能设备项目向导”的“应用程序设置”

为新的 ATL 智能设备项目指定平台设置之外的其他设置。

注意
不支持在智能设备项目上运行代码向导时，应选择“属性化”。此版本不支持智能设备项目中的属性化代码。

服务器类型

动态链接库 (DLL)

指定服务器为动态链接库 (DLL) 并且因此是进程内服务器。

可执行文件 (EXE)

指定服务器为可执行文件 (EXE) 并且因此是本地进程外服务器。此选项不支持 MFC。

附加选项

支持 MFC

指定服务器包含 MFC 支持。此选项将项目链接到 MFC 库，以便可以访问它们包含的任何类和函数。

只有需要在项目中使用特定于 MFC 的类时才选择此选项。实用工具类（如 **CString**、CRect Class、CSize Class 和 CPoint Class）不需要您向 ATL 项目添加 MFC 支持。

如果选择了此选项，则必须将以下行的内容添加到每个使用 MFC 的 COM 方法、Windows 过程和导出函数的开头：

```
AFX_MANAGE_STATE(AfxGetStaticModuleState());
```

支持合并的代理/存根

通常为使用 **DCOM** 的 SDK 启用 MIDL 生成的代理和存根代码。不支持此选项的平台不能使用此选项，这些平台包括 Pocket PC 和 Smartphone。

请参见

参考

ATL 智能设备项目向导

其他资源

设备的 ATL 引用

设备的用户界面参考

“MFC 智能设备 ActiveX 控件向导”的“应用程序设置”

描述“MFC 智能设备 ActiveX 控件向导”的“应用程序设置”页。

使用 MFC 智能设备 ActiveX 控件向导中的此页可为新的 MFC 智能设备 ActiveX 项目设计和添加基本功能。这些设置应用于应用程序本身，不应用于控件的任何特定功能或元素。

应用程序设置

运行时许可证

选择此选项以生成和控件一起分发的用户许可证文件。该许可证是一个文本文件，即 *projname*.lic。此文件必须与控件的 DLL 位于同一目录中，以便可以在设计时环境中创建控件实例。您通常和控件一起分发该文件，但您的客户不分发它。

请参见

参考

MFC 智能设备 ActiveX 控件向导

“MFC 智能设备 DLL 向导”的“应用程序设置”

描述“MFC 智能设备 DLL 向导”的“应用程序设置”页。

使用 MFC 智能设备 DLL 向导中的此页为新的 MFC 智能设备 DLL 项目设计和添加基本功能。

DLL 类型

选择要创建的 MFC 智能设备 DLL 的类型。

使用共享 MFC DLL 的规则 DLL

选择此选项将 MFC 库作为共享 DLL 链接到您的程序。使用此选项不能在 DLL 和调用应用程序之间共享 MFC 对象。程序在运行时调用 MFC 库。如果程序由多个使用 MFC 库的执行文件组成，则此选项可降低程序的磁盘和内存需求。Windows CE 和 MFC 程序都可调用 DLL 中的函数。必须将 MFC DLL 与此项目类型一起重新发布。

带静态链接 MFC 的规则 DLL

选择此选项在生成时将您的程序静态链接到 MFC 库。Windows CE 和 MFC 程序都可调用 DLL 中的函数。虽然此选项会增加程序的大小，但不必将 MFC DLL 与此项目类型一起重新分布。不能在 DLL 和调用应用程序之间共享 MFC 对象。

MFC 扩展 DLL

如果希望程序在运行时调用 MFC 库和希望在 DLL 和调用应用程序之间共享 MFC 对象，则选择此选项。如果程序由多个使用 MFC 库的执行文件组成，则此选项可降低程序的磁盘和内存需求。只有 MFC 程序可以在 DLL 中调用函数。必须将 MFC DLL 与此项目类型一起重新发布。

附加功能

选择 MFC DLL 是否应支持自动化和 Windows 套接字。

自动化

选择“自动化”使您的程序可以操作在其他程序中实现的对象。选择“自动化”还会向其他自动化客户端公开您的程序。有关更多信息，请参见 Automation。

Windows 套接字

选择此选项以指示程序支持 Windows 套接字。Windows 套接字允许编写通过 TCP/IP 网络通信的程序。创建具有 Windows 套接字支持的 MFC DLL 时，CWinApp::InitInstance 会初始化套接字支持，并且 MFC 头文件 StdAfx.h 中将包含 AfxSock.h。

请参见

参考

MFC 智能设备 DLL 向导

“Win32 智能设备项目向导”的“应用程序设置”

使用此向导页设置 Windows CE 项目的选项。

应用程序类型

创建指定的应用程序类型。

选项	说明
Windows 应用程序	通过调用 Windows CE API 创建图形用户界面，从而创建用 Visual C 或 Visual C++ 编写的可执行应用程序 (.exe)。
控制台应用程序	为应用程序创建控制台应用程序结构。此选项仅在支持控制台应用程序的平台上可用。
DLL	创建用 Visual C 或 Visual C++ 编写的、调用 Windows CE API（而不是 MFC 类）的二进制文件 (DLL)。它与函数的共享库类似，可由多个应用程序同时使用。可以指示 DLL 导出符号。
静态库	生成可执行文件时创建包含链接到程序中的对象（及其函数和数据）的文件。可以将静态库链接到基于 MFC 的程序或者非 MFC 程序。不能向静态库项目添加 ATL 支持。

附加选项

根据应用程序的类型定义其支持和选项。

选项	说明
空项目	指定项目文件为空。如果有一组源代码文件（如 .cpp 文件、头文件、图标、工具栏或对话框）并且需要在 Visual C++ 开发环境中创建项目，则必须首先创建一个空项目，然后向该项目中添加这些文件。此选项对静态库项目不可用。
导出符号	指定 DLL 项目导出符号。
预编译头	指定静态库项目使用预编译头。

添加支持，以用于

为 Visual C++ 中提供的某个库提供支持。

选项	说明
ATL	在项目中内置对设备活动模板库 (ATL) 中类的支持。此选项不可用于静态库项目。
MFC	在项目中内置对设备 Microsoft 基础类 (MFC) 库的支持。

请参见

参考

Win32 智能设备项目向导

“MFC 智能设备应用程序向导”的“应用程序类型”

使用 MFC 智能设备应用程序向导中的此页可为新的 MFC 智能设备项目设计和添加基本功能。

应用程序类型

指示要在应用程序中创建的文档支持类型。选择的应用程序类型决定应用程序可用的用户界面选项。有关用户界面选项的更多信息，请参见“MFC 智能设备应用程序向导”的“用户界面功能”。有关文档类型的更多信息，请参见：

- SDI and MDI
- Frame Windows
- Frame-Window Classes
- Documents, Views, and the Framework
- Dialog Boxes

选项	说明
单文档	用基于 CView 的视图类创建应用程序的单文档界面 (SDI) 结构。可以在向导的生成的类页中更改视图的基类。例如，若要创建基于窗体的应用程序，可以使用视图类的 CFormView。 在此类应用程序中，文档的框架窗口只能容纳框架中的一个文档。
基于对话框	通过基于“CDialog”的对话框类为您的应用程序创建一个基于对话框的结构。
带有文档列表的单文档	基于 CDocList 类为应用程序创建单文档结构。此选项仅在支持此类的平台上可用，如 Pocket PC。 **注意** CdocList 在此版本中不可用。

文档/视图结构支持

使用 CDocument Class 和 CView Class 基类在应用程序中包含文档/视图结构（默认）。如果正在移植非 MFC 应用程序，或者如果想减少编译的可执行文件的大小，则清除此复选框。默认情况下，无文档/视图结构的应用程序从 CWinApp Class 派生，不包含用于从磁盘文件中打开文档的 MFC 支持。

资源语言

为资源设置要使用的语言。列表显示系统上由 Visual Studio 安装的可用语言。如果要选择系统语言以外的语言，则必须已经安装了该语言的适当模板文件夹。有关安装“资源语言”列表中默认的可用语言资源之外的语言资源的更多信息，请参见其他语言的向导支持。选择的语言反映在向导的文档模板字符串页的“本地化字符串”选项中。

MFC 的使用

指示如何链接到 MFC 库。默认情况下，MFC 作为共享 DLL 链接。

选项	说明
使用共享 DLL 中的 MFC	将 MFC 库作为共享 DLL 链接到应用程序。应用程序在运行时调用 MFC 库。如果应用程序由多个使用 MFC 库的可执行文件组成，则此选项会降低应用程序的磁盘和内存需求。Windows CE 和 MFC 应用程序都可以调用 DLL 中的函数。
使用静态库中的 MFC	生成时将应用程序链接到静态 MFC 库。

请参见

参考

MFC 智能设备应用程序向导

ATL 智能设备项目向导

活动模板库 (ATL) 是一组基于模板的 C++ 类，用以简化 COM 对象的编写。“ATL 智能设备项目向导”以包含 COM 对象的结构创建项目。

概述

显示正在创建的 ATL 项目的当前项目设置。默认情况下，项目有以下设置：

- 项目的默认目标平台是平台列表中的第一个平台。在默认安装中，默认平台是 Pocket PC 2003，但安装和卸载 Windows CE 5.0 SDK 可能会更改新应用程序的默认目标平台。
- 动态链接库，指定服务器是 DLL 并且因此是进程内服务器。

平台

单击向导左列中的“平台”，显示“ATL 智能设备项目向导”的“平台”页。使用此页选择适合当前项目的 SDK。

应用程序设置

单击向导左列中的“应用程序设置”，显示“ATL 智能设备项目向导”的“应用程序设置”页。使用此页选择服务器类型（DLL 或 EXE）以及某些附加选项。

注意
在 ATL EXE 项目中，如果 COM 本地服务器注册时所在的目录有一个包含空格的长文件名（例如 \Program Files\My Server\），则在 COM 实例化 COM 本地服务器时，COM 本地服务器不启动。 使用没有空格的短文件名注册类可以避免这种情况发生。有关更多信息，请参见如何：指定主项目输出的远程路径。（默认值包含空格。）

请参见

其他资源

设备的用户界面参考

设备的 ATL 引用

“部署项目属性”对话框 ->“配置设置”->“生成”

指定当前选定的智能设备部署项目的生成设置。项目名称在标题栏中显示。

若要显示“部署项目属性”对话框，请右击“解决方案资源管理器”中的项目，再单击“属性”。

配置

指定生成配置，如“调试”、“发布”，等等。

平台

指定活动项目的目标平台。

配置属性文件夹

指定要在右侧面板中显示的属性类别。

配置管理器

打开“配置管理器”对话框。

输出文件名

指定生成时放置 CAB 文件的位置。单击“浏览”选择不同于默认位置的位置。

Authenticode 签名

指定是否将使用 Authenticode 签名为输出文件签名。有关更多信息，请参见设备项目中的安全性。

证书

指定用于对文件进行签名的 Authenticode 证书文件。单击“从存储区选择”选择证书文件。

从存储区选择

打开“选择证书”对话框，可在此选择要使用的证书。有关更多信息，请参见如何：在设备项目中导入和应用证书。

请参见

参考

“配置管理器”对话框

其他资源

设备项目中的安全性

“更改目标平台”对话框(设备)

将托管设备项目的目标平台更改为相同 .NET Compact Framework 版本的另一个平台。

若要打开此对话框，请单击“项目”，再单击“更改目标平台”。仅当在 Visual Studio IDE 中打开托管项目时，此菜单项才会出现。

当前平台

显示活动项目平台。

更改为

显示与当前平台具有相同 .NET Compact Framework 版本的其他可用平台的列表。

请参见

其他资源

使用 .NET Compact Framework 进行设备编程

设备的用户界面参考

“智能设备项目向导”的“配置管理器设置”

使用此对话框创建并编辑解决方案生成配置和项目配置。对解决方案生成配置所做的任何更改都在“解决方案属性页”对话框的“配置”页上反映出来。可以通过以下几种方式访问“配置管理器”：从“生成”菜单，从“解决方案属性页”对话框，或从主工具栏中的解决方案配置下拉列表。

活动的解决方案配置

显示可用的解决方案生成配置。使用此下拉列表或主工具栏中的“配置”下拉列表更改活动解决方案配置。若要创建新的解决方案配置和修改现有的解决方案配置，请从该下拉列表中选择“<新建...>”或“<编辑...>”。

活动解决方案平台

显示为其生成解决方案的可用平台。更改活动解决方案平台时，会将更改应用于解决方案中的所有项目。若要创建新的解决方案平台以及修改现有的解决方案平台，请从该下拉列表中选择“<新建...>”或“<编辑...>”。

项目上下文

选定的解决方案生成配置中“项目上下文”下的每一项均包括项目名称、配置类型和平台的下拉列表，以及用于选择要生成和要部署（如果已启用）的那些项目的复选框。所选的类型和平台组合确定将使用的项目配置。单击列标头对网格中的列进行排序。

项目

显示在当前解决方案中找到的项目名称。

配置

显示所需的项目版本类型，并列出所有可用的版本类型。若要创建新的项目版本类型或重命名现有的项目版本类型，请从此下拉列表中选择“<新建...>”或“<编辑...>”。

平台

显示所需版本必须在其上运行的平台，并列出该项目的所有可用平台。若要添加新平台或编辑现有平台，请从此下拉列表中选择“<新建...>”或“<编辑...>”。所使用的项目类型确定是否可以添加多个平台，以及哪些平台可以添加到项目中。当为项目添加平台时，新的项目配置将创建。

生成

指定该项目是否将由当前解决方案配置生成。未选定的项目不生成，不管其上有任何项目依赖项。未选定要生成的项目仍包含在解决方案的调试、运行、打包和部署中。

部署

如果已启用，则指定在对选定的解决方案生成配置使用“运行”或“部署”命令时，是否部署该项目。此复选框只对可部署的项目出现。

请参见

其他资源

使用 .NET Compact Framework 进行设备编程

设备的用户界面参考

“配置仿真引导程序”对话框

向 Pocket PC 和 Smartphone 配置基础结构中插入一个或多个 XML 文件。

若要显示此对话框，请依次单击“工具”菜单上的“选项”、“设备工具”和“设备”，选择仿真程序，单击“属性”，选择“引导程序”，再单击“配置”。

XML 配置文件

向 Pocket PC 和 Smartphone 配置基础结构中插入一个或多个 XML 文件，以调整网络、UI 和安全设置。

请参见

任务

如何：在 Visual Studio 中启动设备仿真程序

参考

“选项”对话框 ->“设备工具”->“设备”

其他资源

设备的用户界面参考

“配置 TCP/IP 传输”对话框

提供用于指定设备项目中传输特性的选项。

若要显示此对话框，请依次单击“工具”菜单上的“选项”、“设备工具”和“设备”，再单击“传输”框旁边的“配置”按钮。如果“配置”按钮灰显，则无法配置所选的传输。

使用固定端口号

始终使用此端口。如果不希望更改设备上的 Internet 协议 (IP) 或端口号，请选择此选项。

使用 ActiveSync 自动获取 IP 地址

导致自动分配 IP 设置。此设置要求在设备和开发计算机之间进行 ActiveSync 通信。

使用特定 IP 地址

选择要使用的 IP 地址。默认地址为本地主机的地址。

请参见

参考

“选项”对话框 ->“设备工具”->“设备”

“部署”对话框（设备）

其他资源

设备的用户界面参考

“连接到设备”对话框

提示输入要连接到的物理设备或仿真程序。

若要显示此对话框，请单击“工具”菜单上的“连接到设备”。

平台

列出已安装的所有平台，其中包括那些不能用于活动项目的平台。如果打开了某一设备项目，则自动选择其平台。

选择“所有平台”会显示所有已安装设备的列表。

设备

列出在“平台”框中所选平台的已安装设备。

连接

打开到“设备”框中所选设备的连接。如果该设备为仿真程序，仿真程序便会在开发计算机的桌面上打开。

请参见

参考

“选项”对话框 ->“设备工具”->“设备”

其他资源

设备的用户界面参考

“连接到 <数据库>”对话框

现有连接中未包含访问数据库所需的密码时，提示输入密码。

当数据源配置向导中需要密码时，此对话框便会出现。

数据库

数据库的文件名 (*.sdf)。

文件

数据库文件的完整路径。

密码

访问数据库所需的密码。

请参见

其他资源

设备的用户界面参考

文件属性对话框中的连接字符串属性(设备)

本主题已针对 Visual Studio 2005 SP1 进行了更新。

作为 .xsd 文件的属性，为设备上运行的应用程序指定运行时连接字符串。

运行时连接字符串是可选属性，在自动生成的连接字符串无法满足需要时使用。生成的连接字符串是从 .xsd 架构文件中的连接字符串获取的，它被视为设计时连接字符串。通过此设计时字符串，Visual Studio 可以在设计时连接到数据库并检索架构信息。如果此设计时连接字符串在运行时不能满足需要，可以使用运行时"Connection String"属性指定要序列化到所生成代码中的自定义连接字符串。

以下段落已针对 Visual Studio 2005 SP1 进行了更新。

例如，Microsoft SQL Mobile 和 Microsoft SQL Server 2005 Compact Edition 连接字符串包含数据库的路径。此时，所生成的连接字符串被转换为假定该数据库与执行二进制文件位于同一文件夹中的连接字符串。如果 Microsoft SQL Mobile 或 SQL Server Compact Edition 数据库实际驻留在设备上的不同位置，可以使用 .xsd 文件中的"Connection String"属性，以使用自定义连接字符串覆盖生成的连接字符串。

在 .xsd 架构文件的"Connection String"属性中指定自定义连接字符串时，不会更改 .xsd 架构文件中存储的设计时连接字符串，而且设计时功能继续正常运行。有关更多信息，请参见如何：更改设计时连接字符串(设备)。

此窗口中的其余文件属性并不特定于智能设备项目。有关更多信息，请参见文件属性。

若要打开"属性窗口"，请在"解决方案资源管理器"中选择 .xsd 文件，然后在"视图"菜单上单击"属性窗口"。

连接字符串

指定用于在运行时连接到设备上的数据库的连接字符串。

如果该框为空，则应用程序使用 .xsd 架构文件中存储的默认连接字符串。

请参见

任务

如何：更改运行时连接字符串(设备)

其他资源

设备的用户界面参考

“MFC 智能设备 ActiveX 控件向导”的“控件名称”

使用此页指定控件类和属性页类的名称、类型名称和控件的类型标识符。除“简称”字段外，所有其他字段都可单独进行编辑。如果更改“简称”的文本，该更改会反映在此页的所有其他字段的名称中。此命名行为旨在使所有名称在开发控件时易于识别。

简称

提供控件的缩写名称。默认情况下，该名称基于在新建项目对话框中提供的项目名称。提供的名称决定类名、类型名称和类型标识符，除非分别更改那些字段。

控件类名

默认情况下，控件类的名称基于简称，并以 **C** 为前缀，以 **Ctrl** 为后缀。例如，如果控件的简称为 **Price**，则控件类名为 **CPriceCtrl**。

控件 .h 文件

默认情况下，该头文件的名称基于简称，并以 **Ctrl** 为后缀，以 **.h** 为文件扩展名。例如，如果控件的简称为 **Price**，则头文件名为 **PriceCtrl.h**。此字段中的名称应与控件类名相匹配。

控件 .cpp 文件

默认情况下，该头文件的名称基于简称，并以 **Ctrl** 为后缀，以 **.cpp** 为文件扩展名。例如，如果控件的简称为 **Price**，则头文件名为 **PriceCtrl.cpp**。此字段中的名称应与头名相匹配。

控件类型名称

默认情况下，控件类型的名称基于简称，后跟 **Control**。例如，如果控件的简称是 **Price**，则控件类的类型名称是 **Price Control**。如果更改该字段中的值，确保名称中体现继承。

控件类型 ID

设置控件类的控件类型 ID。当控件被添加到项目时，它将该字符串写入注册表。容器应用程序使用该字符串创建控件的实例。默认情况下，控件类型 ID 基于在“新建项目”对话框中指定的项目名称和简称。此名称应匹配类型名。默认情况下，控件类型 ID 以如下形式出现：*ProjectName.ShortName*Ctrl.1；如果更改了此对话框中的简称，则控件类型 ID 便会以如下形式出现：*ProjectName.NewShortName*Ctrl.1

属性页类名

默认情况下，属性页类的名称基于简称，并以 **C** 为前缀，以 **PropPage** 为后缀。例如，如果控件的简称是 **Price**，则属性页类的名称为 **CPricePropPage**。该名称应匹配控件类的名称并追加 **PropPage**。

属性页 .h 文件

默认情况下，属性页头文件的名称基于简称，并以 **PropPage** 为后缀，以 **.h** 为文件扩展名。例如，如果控件的简称为 **Price**，则属性页的头文件名为 **PricePropPage.h**。此名称应与类名称相匹配。

属性页 .cpp 文件

默认情况下，属性页实现文件的名称基于简称，并以 **PropPage** 为后缀，以 **.cpp** 为文件扩展名。例如，如果控件的简称为 **Price**，则属性页的头文件名为 **PricePropPage.cpp**。此名称应与头文件名相匹配。

属性页类型名称

默认情况下，属性页类型名称基于简称，后跟 **Property Page**。例如，如果控件的简称是 **Price**，则属性页类型名称为 **Price Property Page**。如果更改该字段中的值，确保名称中体现控件类。

属性页类型 ID

设置属性页类的 ID。当控件应用到项目时，它将该字符串写入注册表。容器应用程序使用该字符段创建控件属性页的实例。默认情况下，属性页类型 ID 基于在“新建项目”对话框中指定的项目名称和简称。此名称应匹配类型名。默认情况下，属性页类型 ID 以如下形式出现：*ProjectName.ShortName*PropPage.1；如果更改了此对话框中的简称，则属性页类型 ID 便会以如下形式出现：*ProjectName.NewShortName*PropPage.1

请参见

参考

MFC 智能设备 ActiveX 控件向导

“MFC 智能设备 ActiveX 控件向导”的“控件设置”

描述“MFC 智能设备 ActiveX 控件向导”的“控件设置”页。

使用该页中的选项指定所需的控件行为。例如，可以基于现有的标准 Windows CE 控件类型创建控件，优化控件的行为和外观，或者指示控件可以用作其他控件的容器。

有关选择此页中的选项以最大程度提高控件效率的更多信息，请参见 MFC ActiveX Controls: Optimization。

创建的控件基于

从列表中，可以选择控件应从中继承的控件类型。该列表包括 commctrl.h 中向 MFC 应用程序公开的附加公共控件。您的选择将决定 *ProjName* Ctrl.cpp 文件中 **PreCreateWindow** 函数中的控件样式。有关更多信息，请参见 MFC ActiveX Controls: Subclassing a Windows Control。

BUTTON	按钮控件
COMBOBOX	组合框控件
EDIT	编辑控件
LISTBOX	列表框控件
SCROLLBAR	滚动条控件
STATIC	静态控件
msctls_progress32	进度栏公用控件
msctls_statusbar32	状态栏公用控件
msctls_trackbar32	跟踪栏公用控件
msctls_updown32	数值调节钮公共控件
SysHeader32	标头公用控件
SysListView32	列表视图公用控件
SysTabControl32	选项卡公用控件
SysTreeView32	目录树视图公用控件

附加功能

可见时激活

指定当控件变得可见时为它创建窗口。默认情况下设置“可见时激活”选项。如果想将控件的激活推迟到容器需要它时（例如，用户通过鼠标单击），则取消选中此功能。关闭此功能可优化控件，因为只有在需要该控件时才会招致创建窗口的开销。有关此选项的更多信息，请参见 Turning off the Activate When Visible Option。

运行时不可见

指定在运行时控件不出现用户界面。计时器可能是您不希望在界面中看到的一类控件。

有“关于”对话框

指定该控件具有标准的 Windows CE“关于”对话框，并在对话框中显示版本号和版权信息。

注意

用户如何访问控件的帮助取决于帮助的实现方式以及控件帮助是否已与容器帮助集成。有关集成帮助的更多信息，请参见 Adding Context-Sensitive Help to an MFC ActiveX Control。

设置此选项将在项目的控件类 (*CProjName*Ctrl.cpp) 中插入 **AboutBox** 控件方法，并将 AboutBox 添加到项目的调度映射中。默认情况下设置该选项。

优化的绘图代码

指定绘制到同一设备上下文的所有容器控件全都绘制完后，容器自动还原原始的 **GDI** 对象。有关此功能的更多信息，请参见 Optimizing Control Drawing。

无窗口激活

指定控件在激活时不产生窗口。无窗口激活考虑了非矩形控件或透明控件，而且无窗口控件不需要窗口控件所需的系统开销。无窗口激活没有考虑未剪辑的设备上下文和无闪烁激活。1996 年以前创建的容器不支持无窗口激活。有关使用此选项的更多信息，请参见 Providing Windowless Activation。

未剪辑的设备上下文

重写控件头文件 (projnamectrl.h) 中的 COleControl::GetControlFlags，以禁用 **COleControl** 对 IntersectClipRect 的调用。选择“未剪辑的设备上下文”可使速度得到少许提高。如果选择“无窗口激活”，则此功能不可用。有关更多信息，请参见 Using an Unclipped Device Context。

无闪烁激活 (Flicker-Free Activation)

消除在控件的活动与非活动状态间发生的绘图操作和伴随的视觉闪烁。如果选择“无窗口激活”，则此功能不可用。设置此选项后，**noFlickerActivate** 标志包含在 COleControl::GetControlFlags 所返回的标志中。有关更多信息，请参见 Providing Flicker-Free Activation。

在“插入对象”对话框中可用

指定控件在已启用容器的“插入对象”对话框中可用。选择此选项后，**afxRegInsertable** 标志包含在 AfxOleRegisterControlClass 所返回的标志集中。“插入对象”对话框允许用户将新创建或现有的对象插入到复合文档中。

不活动时有鼠标指针通知

无论控件是否是活动的，都使控件可以处理鼠标指针通知。选择此选项后，**pointerInactive** 标志包含在 COleControl::GetControlFlags 所返回的标志集中。有关使用此选项的更多信息，请参见 Providing Mouse Interaction While Inactive。

作为简单框架控件

通过为控件设置 **OLEMISC_SIMPLEFRAME** 位来指定该控件是其他控件的容器。有关更多信息，请参见简单框架站点包容。

异步加载属性

允许重置任何以前的异步数据并开始新的控件异步属性加载。

请参见

参考

MFC 智能设备 ActiveX 控件向导

项目设计器 ->“调试”窗格

提供用于指定命令行参数、工作目录和启动项目的用户输入。

若要打开项目设计器，请在“解决方案资源管理器”中右击“<Projectname>”，再单击快捷菜单上的“属性”。

启动项目

指定当前项目为启动项目。

启动外部程序

如果当前项目未被指定为启动项目，则指定要启动的外部程序。

命令行参数

指定命令行参数。

有两个下拉列表。

- 配置
- 平台

下拉列表

配置下拉列表

显示其他可选择的可用配置的列表，如“调试”和“发布”。

平台下拉列表

显示可用平台的列表，如“活动(Any CPU)”。

请参见

其他资源

设备的用户界面参考

“<Projectname> 属性页”对话框 ->“配置属性”->“调试”(设备)

“配置属性”文件夹中的“调试”属性页用于指定一些调试器属性。

要启动的调试器

要为调试会话启动的可用调试器的下拉列表。

远程可执行文件

用于启动调试会话的远程可执行文件的名称。

命令参数

要传递给远程可执行文件的参数。

有两个下拉列表和一个按钮。

- 配置管理器
- 平台

下拉列表

配置下拉列表

显示其他可选择的可用配置的列表，如“调试”和“发布”。

平台下拉列表

显示其他可用平台的列表，如 Smartphone 2003 或 Pocket PC 2003。

“配置管理器”按钮

“配置管理器”按钮访问项目的“配置管理器”对话框。

请参见

参考

属性页 (C++)

“定义颜色”对话框(设备)

指定颜色的色调，可用以添加自定义色调。如果更改色调，则“红”、“绿”和“蓝”的值将发生更改以与之匹配。值的范围为从 0 到 239。

若要在“设计”视图中显示此对话框，请右击任何具有颜色属性的组件(如 TextBox 控件)，单击快捷菜单中的“属性”，然后在“属性”窗口中选择一个颜色属性，如“Forecolor”。然后，单击该属性列表右侧的下拉箭头，单击“自定义”，再右击“自定义”选项卡底部的空白块之一。

每个已安装平台的颜色映射作为资源存储在平台设计器程序集中。对于桌面平台和设备平台而言，RGB 和已命名的颜色相同，而系统颜色不同。系统颜色在设计时进行转换。可选择的颜色仅限于 .NET Compact Framework 中支持的颜色，并且根据平台不同会有所不同。

如果没有找到某平台的颜色映射方案，则采用桌面默认设置。

可以使用鼠标或 Tab 键，将插入点置于“色调”、“饱和度”和“亮度”框中。然后单击 F1 显示带有更多信息的工具提示。

请参见

参考

“字体编辑器”对话框(设备)

其他资源

设备的用户界面参考

“部署”对话框(设备)

提供一个已安装设备的列表;这些设备将运行面向与设备项目关联的平台的应用程序。

此对话框在以下情况下打开:

- 当“设备工具”选项(单击“工具”菜单上的“选项”便会出现)的“常规”页上的“部署设备项目前显示设备选项”被选中时;以及
- 已发出“部署”命令(该命令可从“生成”菜单上显式发出,也可在发出“调试”菜单上的“启动”命令后隐式发出)。

设备

显示已安装设备的列表,包括仿真程序映像。

选定某个设备然后关闭此对话框后,该设备就成为了此项目的选定设备。

显示选项

如果选定此选项,则每次部署项目时都显示“部署”对话框。使用“选项”对话框 ->“设备工具”->“常规”中的复选框同样可以选定此选项。

部署

连接到目标设备。

如果由“启动”命令隐式打开,则在连接到设备后启动应用程序。

请参见

任务

如何:更改默认设备(托管项目)

如何:更改默认设备(本机项目)

其他资源

设备的用户界面参考

“<Projectname> 属性页”对话框 ->“配置属性”->“部署”(设备)

随着多平台设备部署的引入，现在可以为每一设备设置设备的部署属性。在“解决方案资源管理器”中选定一个项目节点时，“配置属性”文件夹中的“部署”属性页显示在项目的属性页中，它包含两个属性区域、两个下拉列表和一个按钮：

- 常规
- 服务器端操作

常规

部署设备

指定要部署到的设备。

附加文件

包含以下内容的分号分隔列表：要部署到设备的附加文件，以及源目录、目标目录和指示是否应在目标设备上注册文件的值。不要包含被标记为可部署内容的主项目输出或文件。有关可以在此对话框中使用的宏的更多信息，请参见用于生成命令和属性的宏。

远程目录

指定要将应用程序复制到的部署设备上的目录。默认情况下所提供的 %CSIDL_PROGRAM_FILES%\ 将部署到 Smartphone 或 Pocket PC 上的正确位置。

服务器端操作

注册输出

指定是否要在目标设备上注册主项目输出 DLL。该下拉列表允许选择“是”或“否”。为 MFC ActiveX 项目和 ATL 项目选择“是”。

有两个下拉列表和一个按钮。

- 配置管理器
- 平台

下拉列表框

“配置”下拉列表框

显示其他可选择的可用配置的列表，如“调试”和“发布”。

“平台”下拉列表框

显示其他可用平台的列表，如 Smartphone 2003 或 Pocket PC 2003。

“配置管理器”按钮

“配置管理器”按钮访问项目的 “配置管理器”对话框。

请参见

参考

属性页 (C++)

工具箱中的设备组件选项卡

显示可拖到 Windows 窗体上但不属于智能设备项目的 *visual* 元素的组件。在 Visual Studio 2005 中，这些组件包括“指针”、“SerialPort”、“Timer”、“InputPanel”、“Notification”、“HardwareButton”、“MessageQueue”和“BindingSource”。

请参见

其他资源

设备的用户界面参考

“工具箱”->“设备控件”选项卡

显示目标平台支持的控件。仅在智能设备项目处于活动状态时，此选项卡才可用。

如果“设备控件”选项卡不可见，则请在“工具箱”中右击并选择“重置工具箱”或“全部显示”。

可以从“视图”菜单访问“工具箱”。

请参见

参考

工具箱

其他资源

设备的用户界面参考

“设备属性”对话框

提供指定设备属性的选项，这些设备包括仿真程序和物理设备。

若要显示此对话框，请依次单击“工具”菜单上的“选项”、“设备工具”和“设备”，选择设备，再单击“属性”。

设备上的默认输出位置

指定在远程设备上部署应用程序的路径。

传输

指定传输。

Visual Studio 为仿真程序提供 DMA 传输，为物理设备提供 TCP 连接传输。

注意
DMA 传输为仿真程序提供高速和可靠的连接。虽然 TCP 传输可用于仿真程序，但不及 DMA 传输可靠。只有出于极其特殊的原因才对仿真程序使用 TCP 传输。

配置

打开“配置传输”对话框。如果传输为 TCP/IP，可以选择使用固定的端口号，并指定如何确定 IP 地址。如果“配置”按钮灰显，则无法配置选定的传输。

引导程序

指定引导程序。

配置

对于物理设备：打开“配置手动引导程序”对话框，可以在此指定端口号和设备 IP 地址。

如果“配置”按钮灰显，则无法配置选定的引导程序。

检测设备何时断开连接

快速检测设备是否处于连接状态。

若要使用“内核独立传输层”(KITL) 进行调试，请清除此复选框。

对此属性所做的更改会在设备下一次连接时生效。

仿真程序选项

打开仿真程序属性对话框，可以在此设置仿真程序的属性，如显示和连接特性。如果已经选择了物理设备，则此选项不会出现。

请参见

其他资源

设备的用户界面参考

“项目设计器”->“设备”窗格

提供用于进行设备选择和指定智能设备项目的某些特征的控件。“设备”窗格仅出现在设备项目中。

若要打开“项目设计器”，请在“解决方案资源管理器”中右击项目，再单击快捷菜单上的“属性”。

UI

目标设备

指定该应用程序的目标设备。设备不仅包括物理硬件，而且还包括模拟器。

输出文件文件夹

指定将从中下载应用程序的远程设备上的路径。

部署最新版本的 .NET Compact Framework（包括 Service Pack）

如果开发计算机的 .NET Compact Framework 版本比设备或仿真程序上找到的版本新，则部署较新版本。

使用此证书对项目输出进行签名

指定将使用列出的证书对输出文件进行签名。

若要用一个或多个证书填充该框，请单击“选择证书”。

选择证书

打开“选择证书”对话框，可在此选择要使用的证书。

有关更多信息，请参见如何：在设备项目中导入和应用证书。

向设备提供证书

指定是否要向设备提供证书，如果要提供，是否应将该证书放入特许或非特许证书存储区。

有关更多信息，请参见如何：在 Visual Basic 或 Visual C# 项目中提供设备。

请参见

其他资源

设备项目中的安全性

使用 .NET Compact Framework 进行设备编程

设备的用户界面参考

“选项”对话框 ->“设备工具”->“设备”

提供用于更改设备项目配置设置的控件。

若要显示此对话框，请依次单击“工具”菜单上的“选项”、“设备工具”和“设备”。

显示用于以下平台的设备

列出已安装的所有平台，其中包括那些不能用于活动项目的平台。如果打开了某一设备项目，则自动选择其平台。

选择“所有平台”会显示所有已安装设备的列表。

设备

从在“显示用于以下平台的设备”框中选定的平台上已安装的设备的列表中，选择要配置的设备。如果打开了某一设备项目，则自动选择该项目的设备。

默认设备

从在“显示用于以下平台的设备”框中选定的平台的可用设备中设置默认部署目标。如果已选择了“所有平台”，则该控件灰显且不可用。对默认设备所做的更改为所有新项目保留，但对当前活动的项目没有影响。

另存为

打开“将设备另存为”对话框，可在此创建在“设备”框中选定的设备的副本。

重命名

更改在“设备”框中选定的设备的名称。

删除

删除在“设备”框中选定的设备。

如果“删除”按钮灰显，则无法移除选定的设备。在以下情况下，设备不能移除

- 该设备是与产品一起安装的；
- 该设备是某个平台的默认设备；或
- 该设备已在当前打开的项目中被选为目标设备。

属性

打开在“设备”框中选定的设备的“设备属性”对话框。

请参见

参考

“部署”对话框（设备）
“选项”对话框 ->“设备工具”->“常规”
“选项”对话框 ->“设备工具”->“外观设置”

其他资源

设备的用户界面参考

“MFC 智能设备应用程序向导”的“文档模板字符串”

描述“MFC 智能设备应用程序向导”的“文档模板字符串”页

在 MFC 智能设备应用程序向导的此页中，提供或者精炼了以下帮助文档管理和本地化的选项。文档模板字符串对于在应用程序类型中包含“文档/视图结构支持”的应用程序可用。它们对于对话框不可用。由于大多数文档模板字符串是可见的且由应用程序的用户使用，因此它们被本地化为向导的“应用程序类型”页中指示的“资源语言”。

非本地化字符串

应用于创建用户文档的应用程序。如果提供文件扩展名和文件类型 ID，用户可以更容易地打开、打印和保存文档。这些项由系统而非用户使用，因此没有本地化。

选项	说明
文件扩展名	设置与用户在使用应用程序时保存的文档关联的文件扩展名。例如，如果项目名为 Widget，可以将文件扩展名命名为 .wgt。（输入文件扩展名时，不要包含句号。） 如果提供文件扩展名，当用户在打印机图标上放下文档图标时，资源管理器不必启动应用程序就可以打印应用程序的文档。 如果不指定扩展名，用户在保存文件时必须指定文件扩展名。向导不提供默认文件扩展名。
文件类型 ID	在系统注册表中设置文档类型的标签。

本地化字符串

生成与应用程序的用户读取和使用的应用程序和文档关联的字符串，因此字符串被本地化。

选项	说明
语言	指示为“本地化字符串”下的所有框显示字符串所使用的语言。若要更改此框中的值，请在 MFC 智能设备应用程序向导的应用程序类型页中的“资源语言”下选择适当的语言。
主框架标题	设置出现在主应用程序框架顶部的文本。默认情况下，为项目名。
文档类型名称	标识应用程序的文档可归入的文档类型。默认情况下，为项目名。更改默认值不会更改此对话框中的任何其他选项。
筛选器名	设置用户可以用来指示查找特定文件类型的文件的名称。此选项可以从标准 Windows“打开”和“另存为”对话框的“文件类型”和“另存为类型”选项中得到。默认情况下，筛选器名为项目名称加上 Files，后跟“文件扩展名”中提供的扩展名。例如，如果项目名称为 Widget，并且文件扩展名为 .wgt，则默认情况下，“筛选器名”就应该为 Widget Files (*.wgt)。
文件的新简称	如果有不止一个新文档模板，就需要对出现在标准 Windows“新建”对话框中的名称进行设置。如果应用程序是自动化服务器，则此名称用作自动化对象的简称。默认情况下，为项目名。

文件类型全名	在系统注册表中设置文件类型名称。如果应用程序是自动化服务器，则此名称用作自动化对象的全称。默认情况下，项目名称加上 .Document。

请参见

参考

MFC 智能设备应用程序向导

“字体编辑器”对话框(设备)

提供用于指定设备项目中的字体的选项。

若要从设计视图中显示此对话框，请右击具有“Font”属性的任何组件(例如，窗体本身)，在快捷菜单中单击“属性”，在“属性”窗口中选择“Font”，然后单击“浏览(...)”。

此对话框与其桌面副本稍有差别。具体说来，“显示所有字体”复选框表明一个事实，即开发计算机上的所有可用字体可能在目标设备上并不受支持。清除该复选框则仅显示为目标设备预定义的那些字体。

注意
还可以通过选择该字体并将其文件从桌面复制到设备的 Windows\Fonts 目录，从而将自己的字体从开发计算机添加到该设备。

字体

列出可用的字体。

字体样式

列出指定字体的可用样式。

大小

列出指定字体的可用磅值。

显示所有字体

列出活动设备项目的可用字体。选择该复选框可列出开发计算机上的所有可用字体。

删除线

指定是否对字体加删除线，并为字体指定可用的颜色。

下划线

指定是否对字体加下划线，并为字体指定可用的颜色。

示例

显示使用指定字体设置的文本将是什么样子的示例。

每个平台的默认字体如下：

- Pocket PC：Courier New、Tahoma
- Smartphone：Nina
- Windows CE：Arial、Tahoma

请参见

参考

“定义颜色”对话框(设备)

其他资源

设备的用户界面参考

“外观设置属性”对话框(设备)

提供一些选项，以更改设备项目中的外观设置。

如果在进行更改时打开了一个项目，则在关闭并重新打开设计器窗口之前不会出现更改后的效果。

若要显示此对话框，请在“工具”菜单上依次单击“选项”、“设备工具”和“外观设置”，选择要更改的外观设置，然后单击“属性”。

外观预览

显示选定的外观的预览。

外观

设置与选定的外观设置关联的外观。若要选择不同的外观，请单击“浏览”按钮打开相应的 XML 文件。

启用旋转支持

启用旋转的设计时支持。

水平分辨率

在 Windows 窗体设计器中设置以此外观设置为目标的窗体的水平分辨率。

垂直分辨率

在 Windows 窗体设计器中设置以此外观设置为目标的窗体的垂直分辨率。

显示外观

设计和仿真时在窗体周围显示具有选定的外观设置的外观。即使选择了此默认选项，设计器也仅在当您在“选项”->“设备工具”->“常规”对话框中选择了“显示外观”之后才会显示外观。

屏幕宽度

设置构成开发计算机上仿真程序显示宽度的像素数。

如果已经选择了“显示外观”，则已设置了仿真程序显示宽度，此控件灰显。

屏幕高度

设置构成开发计算机上仿真程序显示高度的像素数。

如果已经选择了“显示外观”，则已设置了仿真程序显示高度，此控件灰显。

颜色深度

为仿真程序显示设置其每个像素的位深度。典型值为 8、16 和 32。

如果已经选择了“显示外观”，则已设置了仿真程序显示的颜色深度，此控件灰显。

请参见

任务

如何:更改窗体的方向和分辨率(设备)

其他资源

设备的用户界面参考

“选项”对话框 ->“设备工具”->“外观设置”

提供一些选项，以指定托管设备项目中的外观设置及其属性。

每个平台都有一个默认的外观设置，可对其进行更改以使其适合您的项目需要。例如，可以任意选择需要的屏幕大小和分辨率。

如果在进行更改时打开了一个项目，则在关闭并重新打开设计器窗口之前不会出现更改后的效果。

若要显示此对话框，请依次单击“工具”菜单上的“选项”、“设备工具”和“外观设置”。

显示用于以下平台的外观设置

为选定的平台列出已安装的外观设置。选择“所有平台”将显示所有已安装平台上的所有外观设置。

外观设置

为在“显示用于以下平台的外观设置”中选择的平台列出已安装的外观设置。

默认外观设置

为所有新项目设置默认设计时外观设置，这些项目以在“显示用于以下平台的外观设置”中选择的平台为目标平台。（如果选择了“所有平台”，则此选项灰显。）

另存为

打开“另存为”对话框，可在此保存选定的外观设置的副本。

重命名

更改选定的外观设置的名称。

删除

删除选定的外观设置。

如果“删除”按钮灰显，则无法移除选定的外观设置。在以下情况下，外观设置不能移除

- 该外观设置是与产品一起安装的；
- 该外观设置是平台的默认外观设置。

属性

启动选定的外观设置的“外观设置属性”对话框。

请参见

参考

“外观设置属性”对话框（设备）

其他资源

设备的用户界面参考

“<Projectname> 属性页”对话框 ->“配置属性”->“Authenticode 签名”->“常规”(设备)

在“解决方案资源管理器”中选定一个项目节点时，“配置属性”文件夹中的“Authenticode 签名”属性页显示在项目的属性页中，它包含一个属性区域：

- 常规

常规

Authenticode 签名

指定是否为项目主输出签名。

提供设备

指定是否提供目标设备，如果提供，则指定要放置证书的存储区。

有两个下拉列表和一个按钮。

- 配置管理器
- 平台

下拉列表

配置下拉 列表

显示其他可选择的可用配置的列表，如“调试”和“发布”。

平台下拉列表

显示其他可用平台的列表，如 Smartphone 2003 或 Pocket PC 2003。

“配置管理器”按钮

“配置管理器”按钮访问项目的“配置管理器”对话框。

请参见

参考

属性页 (C++)

“选项”对话框 ->“设备工具”->“常规”

为托管设备项目设置常规选项。

若要显示此对话框，请依次单击“工具”菜单上的“选项”、“设备工具”和“常规”。

部署设备项目前显示设备选项

每次部署项目时都打开“部署”对话框。如果需要经常在各个设备之间切换（例如，在物理设备和它的模拟器之间），请选择此选项。如果仅部署到一个设备，则可以清除该复选框，这样“部署”对话框就不会在每次启动部署时都打开。

也可以使用“部署”对话框选择此选项。

在 Windows 窗体设计器中显示外观

如果在“外观设置属性”对话框（设备）中启用了此选项，则在设计时显示窗体的外观。

请参见

参考

“选项”对话框 ->“设备工具”->“设备”
“部署”对话框（设备）
“选项”对话框 ->“设备工具”->“设备”

其他资源

设备的用户界面参考

“MFC 智能设备应用程序向导”的“生成的类”

描述“MFC 智能设备应用程序向导”的“生成的类”页。

此页列出项目生成的基类和文件的名称。默认情况下，名称基于在新建项目对话框中指定的项目名称。可以更改这些名称中的大多数，详见下文。

生成的类

指示为项目创建的类的名称。默认情况下，名称基于项目名称。默认的 MFC 项目创建一个 C*ProjName*View 类、一个 C*ProjName*App 类、一个 C*ProjName*Doc 类和一个 C*ProjName*MainFrame 类。此页中的所有其他框反映有关“生成的类”列表框中当前选定的类的信息。

若要更改类名，请使用“类名”框。

类名

指示“生成的类”列表框中当前选定的类的名称。如果此框是活动的，则可以更改类名。改变“类名”框中的焦点时，选定类名的任何改变都显示在“生成的类”列表框中。

.h 文件

指示“生成的类”列表框中当前选定的类的头文件名称。如果此文本框是活动的，则可以更改头文件的名称。

基类

指示“生成的类”列表框中当前选定的类的基类名称。如果此框是活动的，则可以从列表中为基类选择另一个类。

.cpp 文件

指示与选定的类关联的源代码文件名。如果此文本框是活动的，则可以更改实现文件的名称。

请参见

参考

MFC 智能设备应用程序向导

MFC 智能设备 ActiveX 控件向导

描述 MFC 智能设备 ActiveX 控件向导。

ActiveX 控件是Automation Servers的特定类型；是可重用组件。承载 ActiveX 控件的应用程序是该控件的Automation Clients。如果目的是创建一个可重用组件，则请使用此向导创建控件。有关更多信息，请参见 MFC ActiveX Controls。

或者，可以使用 MFC 智能设备应用程序向导创建自动化服务器 MFC 智能设备应用程序。

MFC 起始程序包括 C++ 源文件 (.cpp)、资源文件 (.rc) 和一个项目文件 (.vcproj)。这些起始文件中生成的代码基于 MFC。

概述

该向导页描述正在创建的 MFC ActiveX 控件项目的当前应用程序设置。默认情况下，该向导按如下方式创建项目：

- 项目的默认目标平台是平台列表中的第一个平台。在默认安装中，默认平台为 Pocket PC 2003。通过安装或卸载 Windows CE 5.0 SDK，可以更改新应用程序的默认目标平台和/或添加 Smartphone 2003 等其他平台。
- 默认项目不生成运行时许可证。
- 项目包括基于项目名称的控件类和属性页类。
- 控件不基于现有的 Windows CE 控件，当变为可见时激活，它有用户界面并且包括"关于"对话框。

若要更改目标平台，请单击向导左列中的平台并做所需的更改。

若要更改应用程序设置，请单击向导左列中的应用程序设置并做所需的更改。

若要更改控件名称，请单击向导左列中的控件名称并做所需的更改。

若要更改控件设置，请单击向导左列中的控件设置并做所需的更改。

在创建新项目之后，如果编译器对定义 **_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA** 发出警告，则您必须在主头文件中定义此标志。

```
#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA
```

特别是在 Windows Mobile 平台上创建 COM 对象，在 Windows Mobile 中使用 Web 服务以及使用 ATL COM 对象时，更容易出现这种情况。

请参见

其他资源

创建并移植 Visual C++ 设备项目

MFC 智能设备应用程序向导

MFC 智能设备应用程序向导可生成具有内置功能的应用程序，在编译后可实现 Windows CE 可执行文件 (.exe) 应用程序的基本功能。

概述

"MFC 智能设备应用程序向导"的"概述"页描述正在创建的 MFC 智能设备应用程序的当前应用程序设置。默认情况下，该向导按如下方式创建项目：

平台

- 项目的默认目标平台是平台列表中的第一个平台。在默认安装中，默认平台是 Pocket PC 2003，但安装和卸载 Windows CE 5.0 SDK 可能会更改新应用程序的默认目标平台，或添加诸如 Smartphone 2003 等新目标平台。

应用程序类型

- 项目通过 SDI and MDI 创建。
- 项目使用 Document/View Architecture。
- 项目使用共享 DLL 中的 MFC。

文档模板字符串

- 项目使用默认文档模板字符串的项目名称。

用户界面功能

- 项目实现命令栏。

高级功能

- 项目不支持高级功能。

生成的类

- 项目的视图类从 CView Class 派生。
- 项目的应用程序类从 CWinApp Class 派生。
- 项目的文档类从 CDocument Class 派生。
- 项目的主框架类从 CFrameWnd Class 派生。

在创建新项目之后，如果编译器对定义 **_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA** 发出警告，则您必须在主头文件中定义此标志。

```
#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA
```

特别是在 Windows Mobile 平台上创建 COM 对象，在 Windows Mobile 中使用 Web 服务以及使用 ATL COM 对象时，更容易出现这种情况。

请参见

其他资源

使用 Visual C++ 进行设备编程

MFC 智能设备 DLL 向导

描述 MFC 智能设备 DLL 向导，尤其是"概述"页。

当使用 MFC DLL 向导创建一个 MFC DLL 项目时，会得到一个具有内置功能的工作起始应用程序，该功能在编译后将实现 DLL 的基本功能。MFC 起始程序包括 C++ 源文件 (.cpp)、资源文件 (.rc) 和一个项目文件 (.vcproj)。这些起始文件中生成的代码基于 MFC。有关更多详细信息，请参见为您的 Visual Studio 中的项目生成的 Readme.txt 中的文件详细信息，并请参见 MFC DLL 向导生成的类和函数。

概述

此向导页显示正在创建的 MFC 智能设备 DLL 项目的当前项目设置。默认情况下，项目有以下选项设置：

- 项目的默认目标平台是平台列表中的第一个平台。在默认安装中，默认平台是 Pocket PC 2003，但安装和卸载 Windows CE 5.0 SDK 可能会更改新应用程序的默认目标平台。
- 该项目作为一个带静态链接 MFC 的规则 DLL 创建。

若要更改默认平台，请单击向导左列中的平台并做所需的更改。

若要更改默认应用程序设置，请单击向导左列中的应用程序设置并做所需的更改。

在创建新项目之后，如果编译器对定义 **_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA** 发出警告，则您必须在主头文件中定义此标志。

```
#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA
```

特别是在 Windows Mobile 平台上创建 COM 对象，在 Windows Mobile 中使用 Web 服务以及使用 ATL COM 对象时，更容易出现这种情况。

请参见

其他资源

使用 Visual C++ 进行设备编程

“输出文件文件夹”对话框(设备)

指定要在设备文件系统中放置项目输出的位置。

若要打开“输出文件文件夹”对话框，在“解决方案资源管理器”中右击“<Projectname>”，单击快捷菜单上的“属性”，选择“设备”窗格，然后单击“输出文件文件夹”框右侧的“省略号”(“...”)按钮。

设备上的输出位置

指定包含部署目录的设备根目录。

子目录

指定要在其中放置项目输出的目录。

输出文件文件夹(生成的)

显示要在其中放置项目输出的完全限定目录。

请参见

其他资源

使用 .NET Compact Framework 进行设备编程

设备的用户界面参考

“ATL 智能设备项目向导”的“平台”

“ATL 智能设备项目向导”的“平台”页用于指定新 ATL 项目要支持的平台和指令集。

平台窗口

指定项目的目标平台。可用平台通过 Visual Studio 2005 安装，或通过单独的 Windows Mobile 或 Windows CE 软件开发工具包 (SDK) 安装。

指令集和设备面板

为在“平台”窗口中选择的平台列出受支持的指令集和设备。

已安装的 SDK

显示其他可用平台（如 Smartphone 2003 和 Pocket PC 2003）的列表。可以使用按钮将一个或所有平台添加到“选择的 SDK”中。也可以使用按钮从“选择的 SDK”中移除一个或所有平台。

选择的 SDK

显示选择的目标平台（如 Smartphone 2003、Pocket PC 2003）的列表。

注意
创建项目后，您总是可以添加其他平台，例如在使用应用程序向导创建设备项目后添加 Smartphone 2003；但是，在首次创建项目后向项目添加新平台不会导致将依赖运行时 DLL 添加至该新平台的“附加文件”配置属性。

请参见

参考

ATL 智能设备项目向导

“MFC 智能设备 ActiveX 控件向导”的“平台”

“MFC 智能设备 ActiveX 控件向导”的“平台”页用于指定新的 ActiveX 项目要支持的平台和指令集。

平台窗口

指定项目的目标平台。可用平台通过 Visual Studio 2005 安装，或通过单独的 Windows Mobile 或 Windows CE SDK 安装。

指令集和设备面板

为在“平台”窗口中选择的平台列出受支持的指令集和设备。

已安装的 SDK

显示其他可用平台（如 Smartphone 2003 和 Pocket PC 2003）的列表。可以使用按钮将一个或所有平台添加到“选择的 SDK”中。也可以使用按钮从“选择的 SDK”中移除一个或所有平台。

选择的 SDK

显示选择的目标平台（如 Smartphone 2003、Pocket PC 2003）的列表。

注意
创建项目后，您总是可以添加其他平台，例如在使用应用程序向导创建设备项目后添加 Smartphone 2003；但是，在首次创建项目后向项目添加新平台不会导致将依赖运行时 DLL 添加至该新平台的“附加文件”配置属性。

请参见

参考

MFC 智能设备 ActiveX 控件向导

“MFC 智能设备应用程序向导”的“平台”

“MFC 智能设备项目向导”的“平台”页用于指定新 MFC 项目要支持的平台和指令集。

平台窗口

指定项目的目标平台。可用平台通过 Visual Studio 2005 安装，或通过单独的 Windows Mobile 或 Windows CE SDK 安装。

指令集和设备面板

为在“平台”窗口中选择的平台列出受支持的指令集和设备。

已安装的 SDK

显示其他可用平台（如 Smartphone 2003 和 Pocket PC 2003）的列表。可以使用按钮将一个或所有平台添加到“选择的 SDK”。也可以使用按钮从“选择的 SDK”中移除一个或所有平台。

选择的 SDK

显示选择的目标平台（如 Smartphone 2003 和 Pocket PC 2003）的列表。

注意
创建设备项目后，您总是可以在初始创建后添加更多平台（如 Smartphone 2003），但是，在初始创建后向项目添加新的平台不会向所添加平台的“附加文件”配置属性添加附加的依赖运行时 DLL。

请参见

参考

MFC 智能设备应用程序向导

“MFC 智能设备 DLL 向导”的“平台”

“MFC 智能设备 DLL 向导”的“平台”页用于指定新 MFC 智能设备 DLL 项目要支持的平台和指令集。

平台窗口

指定项目的目标平台。可用平台通过 Visual Studio 2005 安装，或通过单独的 Windows Mobile 或 Windows CE SDK 安装。

指令集和设备面板

为在“平台”窗口中选择的平台列出受支持的指令集和设备。

指令集和设备面板

为在“平台”窗口中选择的平台列出受支持的指令集和设备。

已安装的 SDK

显示其他可用平台（如 Smartphone 2003 和 Pocket PC 2003）的列表。可以使用按钮将一个或所有平台添加到“选择的 SDK”中。也可以使用按钮从“选择的 SDK”中移除一个或所有平台。

选择的 SDK

显示选择的目标平台（如 Smartphone 2003 和 Pocket PC 2003）的列表。

注意
创建项目后，您总是可以添加其他平台，例如在使用应用程序向导创建设备项目后添加 Smartphone 2003；但是，在首次创建项目后向项目添加新平台不会导致将依赖运行时 DLL 添加至该新平台的“附加文件”配置属性。

请参见

参考

MFC 智能设备 DLL 向导

“Win32 智能设备项目向导”的“平台”

“Windows CE 应用程序向导”的“平台”页用于指定新 Windows CE 项目要支持的平台和指令集。

平台窗口

指定项目的目标平台。可用平台通过 Visual Studio 2005 安装，或通过单独的 Windows Mobile 或 Windows CE SDK 安装。

指令集和设备面板

为在“平台”窗口中选择的平台列出受支持的指令集和设备。

已安装的 SDK

显示其他可用平台（如 Smartphone 2003 和 Pocket PC 2003）的列表。可以使用按钮将一个或所有平台添加到“选择的 SDK”。也可以使用按钮从“选择的 SDK”中移除一个或所有平台。

选择的 SDK

显示选择的目标平台（如 Smartphone 2003、Pocket PC 2003）的列表。

注意 创建设备项目后，您总是可以在初始创建后添加更多平台（如 Smartphone 2003），但是，在初始创建后向项目添加新的平台不会向所添加平台的“附加文件”配置属性添加附加的依赖运行时 DLL。

请参见

参考

Win32 智能设备项目向导

“智能设备 Cab 项目”->“属性”窗口

使用此“属性”窗口可以指定智能设备 Cab 项目的属性。

若要打开此窗口，请在“解决方案资源管理器”中选择“智能设备 Cab”项目，然后在“视图”菜单上单击“属性窗口”。

CE Setup DLL

指定要为此 .cab 文件使用的 Windows CE 安装程序 DLL。Setup.dll 是可选文件，该文件在安装和移除应用程序的过程中执行自定义操作。有关更多信息，请参见设备解决方案打包概述。

Compress

指定是否应压缩此 .cab 文件。

并非所有 SDK 都支持压缩的 .cab 文件（例如，Pocket PC 2003 就不受支持）。有关更多信息，请参见 SDK 文档。

Manufacturer

指定应用程序或组件的制造商名称。

NoUninstall

指定是否可卸载此 .cab 文件。此选项用于 Compact Framework 2.0 和更高版本。

OSVersionMax

指定可安装此 .cab 文件的最大操作系统版本号。

OSVersionMin

指定可安装此 .cab 文件的最小操作系统版本号。

ProductName

指定描述应用程序或组件的公共名称。也称为“应用程序名”。

请参见

其他资源

设备的用户界面参考

“智能设备托管项目”->“属性”窗口

使用此窗口可指定智能设备托管项目的属性。显示在此“属性窗口”中但未在此处列出的属性是桌面的通用属性。

要打开此窗口，请在“解决方案资源管理器”中选择“智能设备”项目，然后在“视图”菜单上单击“属性窗口”。

输出文件文件夹

远程设备上放置应用程序的文件夹。

平台

应用程序的目标平台。

目标设备

应用程序的目标远程设备或仿真程序。

Framework 版本

此项目的目标 .NET Compact Framework 的版本。

请参见

其他资源

设备的用户界面参考

“选择证书”对话框(设备)

提供一些为设备应用程序签名而指定证书的方式。同时还可作为“管理证书”对话框和“证书导入向导”的入口。

有关更多信息，请参见如何:在设备项目中导入和应用证书。

证书窗格

列出可用的证书。

管理证书

打开“管理证书”对话框，可以在此导入新证书。其他可用的信息和选项包括“中级证书颁发机构”、“受信任的根证书颁发机构”和“受信任和不受信任的发行者”。

请参见

概念

安全概述(设备)

其他资源

设备的用户界面参考

“MFC 智能设备应用程序向导”的“用户界面功能”

此页提供用来指示应用程序的外观和行为的选项。项目的可用用户界面功能取决于在 MFC 应用程序向导的应用程序类型页中指定的应用程序类型。

命令栏

指定应用程序命令栏的详细信息。

选项	说明
仅菜单	向应用程序中添加基本命令栏。
菜单和按钮	添加带有按钮的基本命令栏。

状态栏

在此版本中不支持此选项。

实现带有一行文本输出窗格的控件栏。对基于对话框的应用程序类型不可用。

对话框标题

仅对基于 CDialog 的应用程序，此标题出现在对话框的标题栏中。若要编辑此字段，必须选择“应用程序”类型下的“基于对话框”选项。有关更多信息，请参见“MFC 智能设备应用程序向导”的“应用程序类型”。

请参见

参考

MFC 智能设备应用程序向导

Win32 智能设备项目向导

使用 Win32 智能设备项目向导，可创建四种能够在 Windows CE 操作系统上运行的项目中的任意一种。每种类型中，都可以指定适合打开的项目类型的附加选项。

概述

此向导页显示正在创建的 Windows CE 项目的当前项目设置。默认情况下，项目有以下选项设置：

- 项目的默认目标平台是平台列表中的第一个平台。在默认安装中，默认平台是 Pocket PC 2003，但安装和卸载 Windows CE 5.0 SDK 可能会更改新应用程序的默认目标平台。
- 项目是 Windows 应用程序。Windows 应用程序被定义为包含 .cpp 文件的项目，这种文件包含一个 **WinMain** 函数。
- 项目非空。
- 项目不包含导出符号。
- 项目不使用预编译头文件。
- 项目不包括对 MFC 或 ATL 的支持。

若要更改默认平台，请单击向导左列中的平台并做所需的更改。

若要更改其他的默认值，请单击向导左列中的应用程序设置并做所需的更改。

在创建新项目之后，如果编译器对定义 **_CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA** 发出警告，则您必须在主头文件中定义此标志。

```
#define _CE_ALLOW_SINGLE_THREADED_OBJECTS_IN_MTA
```

特别是在 Windows Mobile 平台上创建 COM 对象，在 Windows Mobile 中使用 Web 服务以及使用 ATL COM 对象时，更容易出现这种情况。

请参见

其他资源

创建并移植 Visual C++ 设备项目

用于设备应用程序开发的 Visual Basic 语言参考

设备应用程序开发人员可用的 Visual Basic 语言编程元素与面向 .NET Compact Framework 时可用的编程元素不完全相同。Visual Basic 参考记录了设备应用程序开发不支持的那些元素。有关更多信息，请参见 .NET Compact Framework 开发中与台式机的不同之处。

新增功能

Visual Studio 2005 中的下列编程元素对于设备应用程序开发人员而言是新内容：

- AudioPlayMode 枚举
- SByte 数据类型 (Visual Basic)、UInteger 数据类型、ULong 数据类型 (Visual Basic) 和 UShort 数据类型 (Visual Basic)
- TextFieldParser 对象
- IsNot 运算符

支持的元素

下列编程元素受支持：

- 所有 Visual Basic 数据类型、指令、方法、对象、运算符和语句。
- 未在以下各节中作为不受支持元素列出的属性 (Attribute)、常数和枚举、函数、关键字以及属性 (Property)。

不支持的元素

当面向 .NET Compact Framework 时，某些 Visual Basic 语言编程元素是不可用的。下列段落概述了开发智能设备应用程序时不支持的编程元素：

属性

VBFixedStringAttribute 类

常数和枚举

FileAttribute 枚举、CallType 枚举、VbStrConv 枚举和 VariantType 枚举。

函数

AppActivate 函数 , Beep 函数 , CallByName 函数 , ChDir 函数 , ChDrive 函数 , CreateObject 函数 (Visual Basic), CurDir 函数 , DeleteSetting 函数 , Dir 函数 , Environ 函数 , EOF 函数 , FileAttr 函数 , FileClose 函数 , FileCopy 函数 , FileDateTime 函数 , FileGet 函数 , FileGetObject 函数 , FileLen 函数 , FileOpen 函数 , FilePut 函数 , FilePutObject 函数 , FileWidth 函数 , FreeFile 函数 , GetAllSettings 函数 , GetAttr 函数 , GetObject 函数 (Visual Basic), GetSetting 函数 , Input 函数 , InputString 函数 , Kill 函数 , LineInput 函数 , Loc 函数 , Lock、Unlock 函数 , LOF 函数 , MkDir 函数 , Print、Printline 函数 , Rename 函数 , Reset 函数 , RmDir 函数 , SaveSetting 函数 , Seek 函数 , SetAttr 函数 , Shell 函数 , SPC 函数 , StrConv 函数 , TAB 函数 , Lock、Unlock 函数 , VarType 函数 (Visual Basic), Write、WriteLine 函数 .

关键字

Ansi、Auto、End (Visual Basic) 和 Unicode (Visual Basic) 关键字

属性

LastDllError 属性(Err 对象)、ScriptEngine 属性、ScriptEngineBuildVersion 属性、ScriptEngineMajorVersion 属性和 ScriptEngineMinorVersion 属性属性。

部分支持的元素

可以使用下列属性获取(但不能设置)日期和时间：Today 属性、TimeOfDay 属性、DateString 属性和 TimeString 属性。

请参见

概念

.NET Compact Framework 开发中与台式机的不同之处

其他资源

使用 .NET Compact Framework 进行设备编程

Visual Basic 参考

参考(设备)

错误信息(设备)

某些错误信息仅在开发智能设备时显示，其中包括以下信息。

本节内容

请参见

其他资源

参考(设备)

未能添加对 <Name> 的引用

您试图加载的项目以 Visual Studio 安装未配置为支持的设备平台为目标。

更正此错误

- 安装支持您要加载的项目的平台。

  例如，如果您试图加载 Windows Mobile 5.0 项目，但未安装 Windows Mobile 5.0 SDK，请安装 Windows Mobile 5.0 SDK。

请参见

其他资源

错误信息(设备)

无法加载外观(设备)

在"外观设置属性"对话框的"外观"框中选择文件时，可能发生下列错误。

消息	常见原因
无法加载外观;无法打开 XML 文件	文件名或路径不正确;不是真正的 XML 文件;文件被另一进程锁定;没有足够的用户特权。
无法加载外观;无法打开 XML 文件引用的图像文件	上述任何之一;图像不是 BMP 或 PNG(仅支持这两种图像格式);图像文件已损坏。
无法加载外观;XML 文件不包含有效的外观架构	要查看有效外观架构的详细信息，请参见设备仿真程序上的"帮助"菜单。

更正此错误

1. 单击"确定"关闭错误信息框。
2. 在"外观设置属性"对话框中选择有效的外观文件。

请参见

参考

"外观设置属性"对话框(设备)

从用户数据存储中检索信息时发生错误(设备)

当数据存储已损坏时将发生此错误。若要重新创建数据存储，应该按照以下步骤删除该数据存储并重新打开解决方案。

警告
此过程删除已添加至数据存储的任何自定义项，例如从现有设备中复制的设备、仿真程序配置属性等。

删除数据存储并重新打开解决方案

1. 保存您的工作，然后关闭任何远程工具和 Visual Studio。
2. 删除 Corecon 目录中的全部内容。

   此目录位于 %USERPROFILE%\Local Settings\Application Data\Microsoft 下。

   例如 `del Documents and Settings\myname\Local Settings\Application Data\Microsoft\Corecon\*.*`。
3. 重新打开 Visual Studio。

   此时，即创建了一个新的数据存储。

请参见

其他资源

错误信息(设备)

泛型部署失败(设备)

在没有特定信息可用来标识部署错误的特性时会显示此错误信息。大多数部署错误都会生成特定的信息。此常规消息很少出现。

请参见

其他资源

错误消息 (Visual Studio)

面向 .NET Compact Framework 1.0 的项目需要 1.1 版的 .NET Framework，但在此计算机上未检测到此版本

面向 .NET Compact Framework 1.0 版的 Visual Studio 2005 项目需要 1.1 版的（桌面）.NET Framework。Smartphone 2003 项目就属于这一类。

在开发计算机上未检测到 1.1 版的 .NET Framework。

安装 1.1 版的 .NET Framework

- 下载并安装 1.1 版的 .NET Framework。

  有关说明在 http://msdn.microsoft.com/netframework/technologyinfo/howtoget/default.aspx 上提供

请参见

概念

设备功能和所需的开发工具

其他资源

错误信息（设备）

无效的输出文件名(设备 CAB 项目)

如果 CAB 项目中的输出文件的名称不符合命名要求，就会发生此错误。

更正此错误

- 使用有效字符和 .cab 扩展名重命名输出文件。

请参见

其他资源

打包设备解决方案以便进行部署

错误信息(设备)

对设备使用新传输前需重置该设备

当更改开发计算机和设备之间的现有连接的传输（例如，将设备仿真程序的传输由 DMA 改至 TCP/IP）时，会发生此警告。

若要确保设备端和桌面组件正使用相同的协议，请重置设备。然后，桌面会使用合适的协议建立新的连接。

重置设备仿真程序

- 在设备仿真程序的“文件”菜单上指向“重置”，然后单击“软”或“硬”。

  进行软重置就足够了。

  - 或 -

- 在设备仿真程序管理器的“可用的仿真程序”窗口中，右击活动的仿真程序，然后在快捷菜单上单击“重置”。

重置物理设备

- 请遵循制造商的说明。

请参见

参考

“设备属性”对话框

其他资源

将智能设备连接到开发计算机上

当前禁用 Visual 继承

该消息的全文为：

因为基控件有设备特定的控件，所以当前禁用 Visual 继承。

当基控件或基窗体具有设备特定的控件或设备特定的组件时，托管项目中可能出现该消息。

可以导致这种状态的情况包括以下几种：

- 继承窗体的父窗体或继承用户控件的父控件包含设备特定的控件。在这种情况下，无法看到继承窗体或继承用户控件的设计器。
- 窗体或用户控件包含设备特定的控件，此控件从另一个窗体或用户控件继承。在这种情况下，无法看到继承窗体或继承用户控件的设计器。
- 项目引用设备特定的程序集，如 Microsoft.WindowsCE.Forms.dll。
- 项目或项目引用的程序集包括平台调用。如果确定设计时不执行平台调用，通过将 **DesktopCompatible(true)** 属性放置在父窗体或父用户控件中可以安全地启用 Visual 继承。

请参见

其他资源

错误信息（设备）

使用 .NET Compact Framework 进行设备编程