emWin

图形库
图形用户界面

版本 5.12

手册修订版 0

www.segger.com
嵌入式软件的解决方案

免责声明

本文中所含的规格信息据信是准确的，但不担保完全没有错误。本手册中所含信息可能会因功能或性能改善而变更，恕不另行通知。请确保您手中的手册是最新版本。虽然此处所提供信息据信是准确的，但如有任何错误或疏漏， SEGGER Microcontroller GmbH & Co. KG（制造商）概不承担任何责任。制造商不作（用户因而也不会获得）任何担保或条件，无论是明示的、暗含的、法定的还是以任何函件形式与用户确认的。特别地，制造商拒绝承担任何适销性或特定目的适用性的担保。

版权声明

商标

本手册中提及的名称可能是其各自所属公司的商标。
各品牌和产品名称均为其各自所有者的商标或注册商标。

注册

请通过电子邮件注册软件。这样，我们就能确保在有更新时，您会立即收到更新或者更新通知。
注册时，请提供以下信息：

- 公司名称和地址
- 您的姓名
- 您的职位
- 您的电子邮件地址和电话号码
- 产品名称和版本

请将这些信息发送至：register@segger.com

联系地址

SEGGER Microcontroller GmbH & Co. KG
In den Weiden 11
D-40721 Hilden
德国
电话：+49 2103-2878-0
传真：+49 2103-2878-28
电子邮件：support@segger.com
网址：http://www.segger.com

手册版本

本手册讲述的是最新软件版本。软件的版本编号请见本章节稍后的 “软件版本”表。如有任何差错，请通知我们，我们将尽快尽力为您提供帮助。
有关任何未明确的其他信息和程序，请联系我们。
印刷日期：12/9/11

版本	日期	作者	描述
5.12R0	110621	AS JE	第 17 章 “窗口对象 （小工具）” - 添加新函数 LISTVIEW_SetHeaderHeight()。 - 添加新函数 ICONVIEW_AddStreamedBitmapItem()。 - 添加新函数 ICONVIEW_GetItemText()。 - 添加新函数 ICONVIEW_GetItemUserData()。 - 添加新函数 ICONVIEW_GetNumItems()。 - 添加新函数 ICONVIEW_InsertBitmapItem()。 - 添加新函数 ICONVIEW_InsertStreamedBitmapItem()。 - 添加新函数 ICONVIEW_SetBitmapItem()。 - 添加新函数 ICONVIEW_SetFrame()。 - 添加新函数 ICONVIEW_SetItemText()。 - 添加新函数 ICONVIEW_SetItemUserData()。 - 添加新函数 ICONVIEW_SetSpace()。 - 添加新函数 ICONVIEW_SetStreamedBitmapItem()。 - 添加新函数 ICONVIEW_SetTextAlign()。 - 添加新函数 TEXT_GetNumLines()。 第 30 章 “显示驱动” - 添加新显示驱动： GUIDRV_Dist GUIDRV_SPage - GUIDRV_CompactColor_16 支持的新显示控制器： 66709：所罗门 SSD1961 - LCD_SetDevFunc()：添加 LCD_DEVFUNC_COPYRECT。 - GUIDRV_Lin：添加对 LCD_DEVFUNC_COPYRECT 的支持。
5.10R1	110531	AS JE	第 30 章 “显示驱动” - 新显示驱动：GUIDRV_FlexColor
5.10R0	110329	AS JE	第 14 章 “存储设备” - GUI_USE_MEMDEV_1BPP_FOR_SCREEN 的默认值是 1。 - 添加新函数 GUI_MEMDEV_MarkDirty()。 添加第 19 章 “GUIBuilder”。 第 30 章 “显示驱动” - GUIDRV_CompactColor_16 支持的新显示控制器： 66708: Ilitek ILI9328 66709: Sitronix ST7715 66772: Ilitek ILI9221 - 添加新函数 GUIDRV_BitPlains_Config()。

版本	日期	作者	描述
5.08R0	110112	AS JE	第 9 章 “2D 图形库” - 添加新函数 GUI_CreateBitmapFromStreamRLEAlpha()。 - 添加新函数 GUI_CreateBitmapFromStreamRLE32()。 - 函数 GUI_CreateBitmapFromStream() 支持额外格式。 - 添加新函数 GUI_UC_EnableBIDI()。 第 12 章 “位图转换器” - 添加新格式 “Alpha 通道，压缩”。 - 添加新格式 “带 Alpha 通道的真彩色，压缩”。 - 添加新功能 Image/Convert Into/Best Palette + transparency。 第 14 章 “存储设备” - 添加新函数 GUI_MEMDEV_SetAnimationCallback()。 - 添加新函数 GUI_MEMDEV_ShiftInWindow()。 - 添加新函数 GUI_MEMDEV_ShiftOutWindow()。 第 15 章 “执行模型” - 添加新函数 GUI_SetSignalEventFunc()。 - 添加新函数 GUI_SetWaitEventFunc()。 - 添加新函数 GUI_SetWaitEventTimedFunc()。 - 编译时间配置宏的定义变更。 第 16 章 “窗口管理器” - 添加新函数 WM_MULTIBUF_Enable()。 - 添加新消息 WM_PRE_PAINT 和 WM_POST_PAINT。 第 17 章 “小工具” - LISTVIEW_SetUserData() 重命名为 LISTVIEW_SetUserDataRow()。 - LISTVIEW_GetUserData() 重命名为 LISTVIEW_GetUserDataRow()。 - 为所有小工具添加新函数 <WIDGET>_SetUserData。 - 为所有小工具添加新函数 <WIDGET>_GetUserData。 - 为所有小工具添加新函数 <WIDGET>_CreateUser。 - 添加新函数 BUTTON_GetTextAlign()。 - 添加新函数 BUTTON_SetReactOnLevel()。 - 添加新函数 ICONVIEW_CreateIndirect()。 - 添加新函数 ICONVIEW_DeleteItem()。 - 添加新函数 LISTWHEEL_CreateIndirect()。 - 添加新函数 SCROLLBAR_SetThumbSizeMin()。 - 添加新函数 SCROLLBAR_GetThumbSizeMin()。 - 添加新函数 TREEVIEW_ITEM_CollapseAll()。 - 添加新函数 TREEVIEW_ITEM_ExpandAll()。 第 19 章 “皮肤设置” - 添加新编译时间配置宏 WIDGET_USE_FLEX_SKIN。 - 新消息 WIDGET_ITEM_GET_RADIUS 添加到帧窗口皮肤。 第 20 章 “多重缓冲”。 - 添加新函数 GUI_MULTIBUF_Begin()。 - 添加新函数 GUI_MULTIBUF_End()。 - 添加新函数 GUI_MULTIBUF_Config()。
5.06R0	100907	JE	第 9 章 “字体”： - 添加新函数 GUI_SetDefaultFont()。 第 12 章 “存储设备”： - 添加新函数 GUI_MEMDEV_FadeDevices()。 第 15 章 “小工具”： - 添加新函数 SCROLLBAR_GetNumItems()。 - 添加新函数 SCROLLBAR_GetPageSize()。 - 添加新函数 BUTTON_SetReactOnLevel()。 - 添加新函数 LISTWHEEL_SetPos()。 - 添加新函数 GRAPH_DATA_XY_SetOwnerDraw()。 - 添加新函数 LISTVIEW_SetItemBitmap()。 新第 17 章 “皮肤设置”： - 为最通用小工具添加皮肤设置功能。 第 26 章 “显示驱动”： - 添加新函数 GUI_SetOrientation()（旋转设备）。 - 给 GUIDRV_Lin 添加用于 16、 24 和 32 bpp 的新 OXY 定向。

版本	日期	作者	描述
5.04R2	100526	AS	- 章节 “小工具”中的新函数 LISTVIEW_SetItemBitmap() - 章节 “小工具”中的新函数 GRAPH_DATA_XY_SetOwnerDraw() - 章节 “字体”中的新函数 GUI_SetDefaultFont() - 章节 “2-D 图形库”中的新函数 GUI_GetPixelIndex() - 章节 “执行模型”中的新函数 GUITASK_SetMaxTask() - GUIDRV_CompactColor_16: 添加对以下显示控制器的支持： Himax HX8353、LGDP4551、Orisetech SPFD54124C、Renesas R61505、矽创 ST7735 和 ST7787、所罗门 SSD1284 和 SSD2119。 - 给每个使用驱动宏的驱动添加宏。
5.04R1	100505	AS	添加驱动 “GUIDRV_S1D15G00”和 “GUIDRV_SLin” 各种修正 章节 “2-D 图形库”： - 新函数 GUI_DrawGradientRoundedV() - 新函数 GUI_DrawGradientRoundedH() - 新函数 GUI_DrawRoundedFrame() 第 12 章 “存储设备”： - 新函数 GUI_MEMDEV_MoveInWindow() - 新函数 GUI_MEMDEV_MoveOutWindow() - 新函数 GUI_MEMDEV_FadeInWindow() - 新函数 GUI_MEMDEV_FadeOutWindow() 章节 “模拟” - 新函数 SIM_GUI_SetCallback() - 新函数 SIM_GUI_ShowDevice()
5.04R0	100104	JE	第 26 章 “VNC 服务器”： - 添加新函数 GUI_VNC_EnableKeyboardInput()。 - 添加新函数 GUI_VNC_GetNumConnections()。 - 添加新函数 GUI_VNC_SetPassword()。 - 添加新函数 GUI_VNC_SetProgName()。 - 添加新函数 GUI_VNC_SetSize()。 - 添加新函数 GUI_VNC_RingBell()。

版本	日期	作者	描述
5.04R0	100104	JE	第 5 章 “显示驱动”： - 添加新函数 GUI_DispStringInRectWrap()。 - 添加新函数 GUI_WrapGetNumLines()。 第 7 章 “2-D 图形库”： - 添加新函数 GUI_EnableAlpha()。 - 添加新函数 GUI_RestoreUserAlpha()。 - 添加新函数 GUI_SetUserAlpha()。 - 添加新函数 GUI_CreateBitmapFromStream()。 - 添加新函数 GUI_DrawStreamedBitmapEx()。 - 添加新函数 GUI_GetStreamedBitmapInfo()。 - 添加新函数 GUI_GetStreamedBitmapInfoEx()。 - 添加新函数 GUI_SetStreamedBitmapHook()。 - 添加新函数 GUI_CreateBitmapFromStreamIDX()。 - 添加新函数 GUI_CreateBitmapFromStreamRLE4()。 - 添加新函数 GUI_CreateBitmapFromStreamRLE8()。 - 添加新函数 GUI_CreateBitmapFromStream565()。 - 添加新函数 GUI_CreateBitmapFromStreamM565()。 - 添加新函数 GUI_CreateBitmapFromStream555()。 - 添加新函数 GUI_CreateBitmapFromStreamM555()。 - 添加新函数 GUI_CreateBitmapFromStreamRLE16()。 - 添加新函数 GUI_CreateBitmapFromStreamRLEM16()。 - 添加新函数 GUI_CreateBitmapFromStream24()。 - 添加新函数 GUI_CreateBitmapFromStreamAlpha()。 第 9 章 “字体”： - 添加新字体 F20F_ASCII （框架式）。 - 添加新字体 F6x8_ASCII 和 F6x8_1。 - 添加新字体 F8x8_ASCII 和 F8x8_1。 - 添加新字体 F8x16_ASCII 和 F8x16_1。 - 添加对扩展 AA2 和扩展 AA4 新字体格式的支持。 第 12 章 “存储设备”： - 添加对多图层 / 显示的考量。 第 14 章 “窗口管理器”： - WM_DeleteWindow() 现在还删除了所有相关定时器。 第 15 章 “小工具”： - 添加新函数 WINDOW_SetBkColor()。 第 19 章 “指针输入装置”： - 添加 PID 缓冲区。 - 修订对触摸校准的解释。 第 20 章 “键盘”： - 添加键盘缓冲区。 第 25 章 “显示驱动”： - 添加新驱动 GUIDRV_BitPlains。 - 添加新驱动 GUIDRV_SLin。 - 添加新驱动 GUIDRV_SSD1926。 - 添加驱动 GUIDRV_1611。 - 添加驱动 GUIDRV_6331。 - 添加驱动 GUIDRV_7529。 - 添加驱动 GUIDRV_Page1bpp。 - GUIDRV_CompactColor_16：添加对以下显示控制器的支持：奇景 HX8340 和 HX8352、所罗门 SSD1298、 SSD1355 和 SSD1963、爱普生 S1D19122、Orisetech SPFD5414D、 Ilitek ILI9320 和 ILI9326
5.00R1	090409	JE	第 3 章 “模拟器”： - 全面修订。 第 8 章 “显示位图文件” - 添加 PNG 支持。
5.00R0	090326	JE	软件已全面修订。有关各早期版本的修订历史记录，请参阅较早文档。

SEGGER Microcontroller GmbH & Co. KG 开发和经销用于嵌入式系统的软件开发工具和ANCI C软件组件（中间件），涉及电信、医疗技术、消费电子、汽车和工业自动化等多种行业。

SEGGER 的目标是通过提供小型灵活并易于使用的中间件，使开发人员可以集中精力于具体应用，从而缩短嵌入式应用的软件开发时间。

我们最受欢迎的产品有 emWin 和 embOS，前者是一款用于嵌入式应用的通用图形软件包，后者是一种小型、高效的实时内核。emWin 完全采用 ANSI C 编写，可轻松用于任何 CPU 和几乎所有显示器。以下 PC 工具是其的完美补充：位图转换器、字体转换器、模拟器和查看器。 embOS 支持大多数 8/16/32 位 CPU。其存储器占用面积小，因此适于单芯片应用。

除了重点关注软件工具外，SEGGER 还研发和生产用于闪存宏控制器的编程工具，以及辅助研发、调试和生产的 JTAG 仿真器 J-Link，它已迅速成为调试 ARM 内核存取的行业标准。

办公地址：
http://www.segger.com

美国办事处：
http://www.segger-us.com

嵌入式软件（中间件）

emWin

图形软件和 GUI

emWin 设计用于提供高效且独立于处理器和显示控制器的图形用户界面 (GUI)，用于任何使用图形显示进行操作的应用。我们提供有初学者套装、评估版和试用版。

embOS

实时操作系统

embOS 是一种实时操作系统 (RTOS)，设计用于为复杂的实时应用提供占用资源最小的完整多任务系统。其中包含仿形PC工具embOSView。

emFile

文件系统

emFile 是支持 FAT12、FAT16 和 FAT32 的嵌入式文件系统。 emFile 已经过优化，可在保持高速度的同时耗用最少的 RAM 和 ROM。可使用各种不同的设备驱动，如 NAND 和 NOR 闪存的驱动、 SD/MMC 和 CompactFlash 卡的驱动。

USB 协议栈

USB 设备协议栈

USB协议栈设计用于在任何使用USB客户端控制器的嵌入式系统中工作。支持块通信和大多数标准设备类别。

SEGGER 工具

Flasher

闪存编程器

主要用于微控制器的闪存编程工具。

J-Link

ARM 内核的 JTAG 仿真器

用于 ARM 内核的 USB 驱动 JTAG 界面。

J-Trace

带跟踪的 JTAG 仿真器

用于 ARM 内核的 USB 驱动 JTAG 界面带跟踪存储器，支持 ARM ETM （嵌入式跟踪宏单元）。

J-Link / J-Trace 相关软件

与 SEGGER 盏男幸当曜糐 TAG 仿真器一同使用的附加软件，包括闪存编程软件和闪存断点。

目录

第 1 章

emWin 简介

本章将介绍与本文档相关的一些信息，也将概述 emWin 的构成特点及其要求。

1.1 本文档的目的

本指南介绍如何安装、配置和使用用于嵌入式应用程序的 emWin 图形用户界面，还将说明软件的内部结构。

1.2 假定

本指南假定您已经具备 C 程序设计语言的基础知识。如果认为自己的 C 程序设计语言知识不够，建议您阅读 Kernighan 和 Richie 编写的 *"C" Programming Language*（C 程序设计语言），该书介绍了编程标准，在其较新版本还包括了 ANSI C 标准。无需汇编编程的知识。

1.3 如何使用本手册

本手册介绍如何安装、配置和使用 emWin。手册说明了软件的内部结构和 emWin 提供的所有功能（应用程序接口或 API）。在实际使用 emWin 前，应阅读或至少浏览一遍本手册以便对软件有所了解。然后建议按以下步骤进行操作：

- 将 emWin 文件复制到计算机中。
- 通读 “入门指南”（第 31 页）。
- 使用模拟器以便更熟悉软件的功能 （请参阅 “模拟”（第 39 页））。
- 参考手册的其它内容，对程序进行扩展。

1.4 语法句子的印刷规范

本手册使用以下印刷规范：

样式	用途
正文	正文文字。
关键字	在命令提示符后输入的或在显示屏上显示的文字 （即系统函数、文件或路径名）。
参数	API 函数中的参数。
示例	示例程序中的示例代码。
新示例	要添加到现有示例程序中的示例代码。
警告	重要的警告或提醒。

1.5 要求

用 emWin 开发软件时无需目标系统；大多数软件都可使用模拟器进行开发。当然，最终目的通常是能在目标系统上运行软件。

1.5.1 目标系统 （硬件）

目标系统必须具有：

- 一个 CPU （8/16/32/64 位）
- 一个具有最小内存的 RAM 和 ROM
- 一个完整图形显示器 （任何类型和任何分辨率）

存储器要求取决于使用的是软件的哪部分以及目标编译器的效率。因此不可能指定精确的值，但是以下值适用于典型的系统。

小系统 （无窗口管理器）

- RAM：100 字节
- 堆栈：600 字节
- ROM：10-25 kb （取决于所使用的功能）

大系统 （包含窗口管理器和小工具）

- RAM：2-6 kb （取决于所需的窗口数）
- 堆栈：1200-1800 字节 （取决于所使用的功能）
- ROM：30-60 kb （取决于所使用的功能）

请注意，如果应用程序使用了很多字体，则对 ROM 的要求会提高。上述所有值都是粗略估算值，不保证能正常使用。

1.5.2 开发环境 （编译器）

所使用的CPU不重要；只需要一个至少满足以下国际标准之一的、符合ANSI标准的C语言编译器：

- ISO/IEC/ANSI 9899:1990 (C90)，支持 C++ 样式注释 (//)
- ISO/IEC 9899:1999 (C99)
- ISO/IEC 14882:1998 (C++)

如果您的编译器有某些限制，请告诉我们，我们将告诉您这些限制是否会在编译软件时导致问题。据我们所知，任何 16/32/64 位 CPU 或 DSP 的编译器都可使用；大多数 8 位编译器也可使用。无需 C++ 编译器，但是可以使用。需要时，应用程序当然也可用 C++ 进行编程。

1.6 特性

emWin 设计用于提供高效且独立于处理器和显示控制器的图形用户界面，用于任何使用图形显示进行操作的应用。它与单任务和多任务环境、专用操作系统或具有任何商业 RTOS 兼容， emWin 的发货形式为 C 语言源代码。它可适用于任何尺寸的、具有任何显示控制器和 CPU 的物理和虚拟显示器。其特性包括：

一般特性

- 支持使用任何控制器的任何 （单色、灰度或彩色）显示器 （如果有正确的驱动）。
- 在较小显示器上无显示控制器也可运行。
- 使用配置宏可支持任何接口。
- 显示尺寸可配置。
- 可在显示器上的任何点 （而不仅仅是在偶数位字节地址）上写入字符和位图。
- 已针对尺寸和速度优化了各种例程。
- 利用编译时间切换可进行不同优化。
- 对于较慢的显示控制器，可在存储器中缓存显示，将存取操作减到最少，从而获得非常高的速度。
- 结构清晰。
- 支持虚拟显示；虚拟显示可大于实际显示。

图形库

- 支持不同色深的位图。
- 可使用位图转换器。
- 绝对无浮点使用。
- 快速线 / 点绘制 （不使用浮点）。
- 圆形 / 多边形绘制非常快速。
- 不同的绘图模式。

字体

- 基本软件配备多种不同的字体：4*6、6*8、6*9、8*8、8*9、8*16、8*17、8*18、24*32 以及像素高度为 8、 10、 13、 16 的比例字体。有关详细信息，请参阅字体一章。
- 可以定义新的字体并只需简单链接。
- 只有应用程序使用的字体才实际链接到生成的可执行程序，从而使 ROM 使用最小。
- 字体可分别在 X 和 Y 方向完全缩放。
- 可使用字体转换器；主机系统 （即 Microsoft Windows）上有的任何字体都可以转换。

字符串 / 值输出例程

- 例程可以十进制、二进制、十六进制、任何字体形式显示值。
- 例程可以十进制、二进制、十六进制、任何字体形式编辑值。

窗口管理器 (WM)

- 完整的窗口管理操作，包括裁剪。窗口的客户区以外的区域不可能被覆盖。
- 窗口可以移动和调整大小。
- 支持回调例程 （可选择是否使用）。
- WM 使用最小的 RAM （每个窗口大约 50 字节）。

PC 界面外观的可选小工具

- 提供各种小工具 （窗口对象，也称为控件）。它们通常自动操作并且简单易用。

支持触摸屏和鼠标

- 对于按钮小工具等窗口对象， emWin 提供触摸屏和鼠标支持。

PC 工具

- 模拟脉冲查看器。
- 位图转换器。
- 字体转换器。

1.7 示例和演示

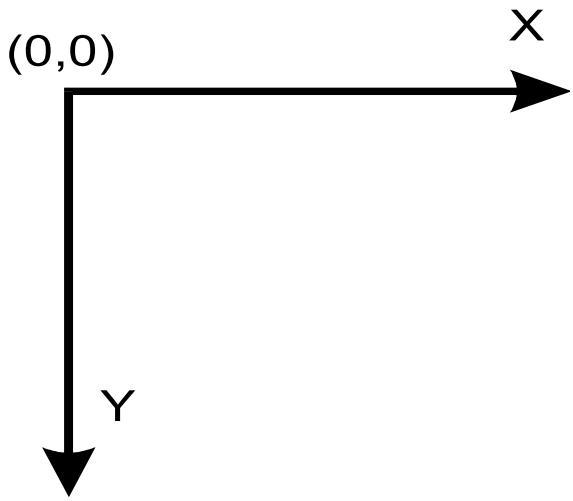
为了让您更好地了解 emWin 的功能，我们准备了各种不同演示，其形式为“即时可用”的模拟可执行程序，在 Sample\EXE 下。示例程序的源代码位于 Sample 文件夹中。文件夹 Sample\GUIDemo 包含一个显示 emWin 许多特性的应用程序。所有示例也可从 www.segger.com 获得。

1.8 初学者套装

完整的初学者套装包括一个带显示器的演示电路板、一个 C 编译器和一个示例项目。有关详细信息，请查看网站 www.segger.com。

1.9 屏幕和坐标

屏幕由许多可以单独控制的点组成，这些点称为像素。emWin 在其 API 中提供给用户程序的绝大部分文本和绘图功能，都可在任何指定的像素上进行书写或绘制。
水平尺度称为 X 轴，而垂直尺度称为 Y 轴。坐标定义为一对由 X 和 Y 值组成的值 (X, Y)。在需要 X 和 Y 坐标的例程中 X 坐标始终在前。显示屏（或窗口）左上角的坐标默认为 (0,0)。正的 X 值始终向右，正的 Y 值始终向下。上图说明了坐标系以及 X 和 Y 轴的方向。传递给 API 函数的所有坐标始终以像素为单位指定。



1.10 如何将显示器连接到微控制器

emWin 处理对显示器的所有访问。事实上可支持任何显示控制器，而与其访问方式无关。有关详细信息，请参阅“配置”（第 905 页）。如果不支持您的显示控制器，请与我们联系。我们当前正为市场上的所有显示控制器编写驱动，可能已经具有您要使用的显示控制器的成熟驱动。通常可非常简单地将访问显示器的例程（或宏）写入应用程序。SEGGER Microcontroller GmbH & Co. KG 可为您提供这些定制服务，如果需要，还可为您的目标硬件提供定制服务。
显示器如何与系统连接无关紧要，只要软件可通过某种方式对其进行访问，这有多种不同的方式。这些接口中的大多数通过以源代码形式提供的驱动支持。此类驱动通常不需要修改，而是通过在文件 LCDConf.h 中进行更改以便根据硬件进行配置。在“显示驱动”（第 813 页）一章中给出了如何在必要时为硬件定制驱动的详细信息。访问显示器最通用的方式如下所述。如果您只想了解如何使用 emWin，则可跳过本节。

带存储器映射显示控制器的显示器

显示控制器直接与系统的数据总线连接，即意味着可像 RAM 一样访问控制器。这是一种访问显示控制器的高效方式，一般都建议使用它。将显示器地址定义到段 LCDSEG，而且为了能够访问显示器，只需告诉链接器 / 定位器如何定位此段即可。该位置必须与物理地址空间中的访问地址相同。可提供用于此类型接口和不同显示控制器的驱动。

显示控制器连接到端口 / 缓冲器的显示器

对于在快速处理器上使用的较慢显示控制器，使用端口线可能是唯一的解决方案。这种访问显示器的方法有一个缺点，比直接总线接口稍微慢些，但是缓存能最小化对显示器的访问，因此显示更新不会显著减慢。所有需要做的事情就是定义例程或宏，设置或读取显示器所连接的硬件端口 / 缓冲器。此类型接口也可由不同显示控制器的不同驱动支持。

专有解决方案：无显示控制器的显示器

也可连接没有显示控制器的显示器。在此情况下，显示数据通常由控制器通过 4 或 8 位移位寄存器直接提供。这些专有硬件解决方案的优点是成本不高，但缺点是会用掉很多可用计算时间。时间可在 20% 到 100%之间，取决于 CPU。对于较慢的 CPU，这是完全不可能的。此类型接口不需要特定的显示驱动，因为 emWin 只是将所有显示数据放入显示缓存中。用户必须自己编写硬件相关的部分，从而定期将高速缓存中的数据传输到显示器。
将视频图像传输到显示器中的示例代码，以 C 语言和 M16C 和 M16C/80 的优化汇编程序两种形式提供。

1.11 数据类型

由于 C 语言不提供在其他所有平台上都相同的固定长度的数据类型，因此大多数情况下， emWin 使用自己的数据类型，如下表所示：

数据类型	定义	描述
I8	带符号字符	带符号的 8 位值
U8	不带符号字符	不带符号的 8 位值
I16	带符号的短数	带符号的 16 位值
U16	不带符号的短数	不带符号的 16 位值
I32	带符号的长数	带符号的 32 位值
U32	不带符号的长数	不带符号的 32 位值
I16P	带符号的短数	带符号的 16（或更多）位值
U16P	不带符号的短数	不带符号的 16（或更多）位值

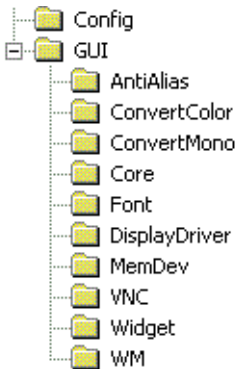
对于大多数 16/32 位控制器，这些设置都运行良好。但是，如果在程序的其他段有类似定义，则需要更改或重新定位它们。建议位置是在文件 Global.h 中。

第 2 章

入门指南

下面一章概述了在您的目标系统上设置和配置 emWin 的基础步骤，其中还包括简单的程序示例。后面章节中对大多数主题进行了更为详细的探讨，如果有不确定的地方请参阅以后部分。在您开始更复杂的编程之前，您最有可能需要参考手册的其它部分。

2.1 推荐目录结构



我们建议 emWin 与您的应用程序文件保持分开。较好的做法是将所有程序文件（包括头文件）都放在项目根目录的 GUI 子目录中。目录结构应类似右侧图示。此做法的优点是可通过简单地替换 GUI\ directories，非常轻松地更新 emWin 版本。您的应用程序文件可存储在任何位置。

2.1.1 子目录

下表显示所有 GUI 子目录的内容：

目录	内容
Config	配置文件
GUI\AntiAlias	支持抗锯齿 *
GUI\ConvertMono	用于灰度显示器的颜色转换例程 *
GUI\ConvertColor	用于彩色显示器的颜色转换例程 *
GUI\Core	emWin core files
GUI\Font	字体文件
GUI\DisplayDriver	显示驱动
GUI\MemDev	存储设备支持 *
GUI\VNC	VNC 支持 *
GUI\Widget	小工具库 *
GUI\WM	窗口管理器 *

(* = 可选）

2.1.2 包含目录

您应确保包含路径含有以下目录 （包含顺序不重要）：

- Config
- GUI\Core
- GUI\DisplayDriver
- GUI\Widget （如果使用小工具库）
- GUI\WM （如果使用窗口管理器）

警示：请始终确保每个文件只有唯一版本！

如果因包括了旧文件而具有了不同版本，则在更新到 emWin 新版本时通常会导致严重问题。如果将 emWin 保持在建议目录下 （且仅在这些目录下），则不会发生此类问题。当更新到较新版本时，您应能够保留您的配置文件并使其处于未更改状态。处于安全考虑，我们建议在更新之前备份（或至少重命名） GUI\ directories。

2.2 添加 emWin 到目标程序

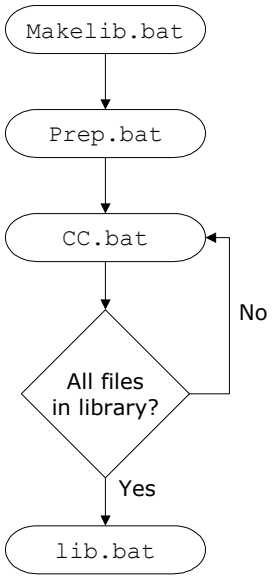
最基本的，您可以选择仅包括您在项目中要实际使用的源文件 （随后进行编译和链接），或者创建一个库并链接库文件。如果您的工具链支持“智能”链接（仅链入所引用的模块，而非未引用的模块），则根本不必创建库，因为只会链接需要的函数和数据结构。如果您的工具链不支持 “智能”链接，则库非常有意义，因为否则会链接所有项目，使程序极其庞大。对于一些 CPU，我们提供了示例项目以帮助您入门。

2.3 创建库

从源构建库的步骤非常简单。第一步是将批处理文件（位于 SampleExample\Makelib 下）复制到您的根目录，然后进行所有必要的更改。总共有四个批处理文件需要复制，如下表所述。主文件 Makelib.bat 对于所有系统都是相同的，不需要任何更改。要从您的目标系统构建库，通常需要对其它三个较小的文件稍作改动。最后，启动文件 Makelib.bat，创建库。批处理文件假定您的 GUI 和 Config 子目录是按建议设置的。

右侧流程图说明了创建库的步骤。Makelib.bat 文件首先调用 Prep.bat，为工具链准备环境。然后为要包括在库中的每个文件调用 CC.bat。按照需要的次数执行该操作。CC.bat 将每个目标文件添加到 lib.bat 将使用的列表中。当要添加到库中的所有文件都已列出后，Makelib.bat 调用 lib.bat，lib.bat 使用库管理程序将列出的目标文件放入实际库中。当然您还可以选择另一种方式创建库。

不建议创建包含编译时间可配置显示驱动的 emWin 库。有关显示驱动的可配置性的更多信息，请参见 “现有显示驱动”（第 814 页）。



文件	描述
Makelib.bat	主批处理文件。不需要任何修改。
Prep.bat	由 Makelib.bat 调用，为要使用的工具链准备环境。
CC.bat	由 Makelib.bat 为要添加到库中的每个文件调用；创建一个这些目标文件的列表，随后将在下一步中由 lib.bat 文件中的库管理程序使用。
lib.bat	由 Makelib.bat 调用，以将 CC.bat 列出的目标文件放入库中。

文件出厂时，假定 Microsoft 编译器安装在其默认位置。如果所有批处理文件都复制到根目录下（直接在 GUI 之上）且未作任何更改，则将为 emWin 模拟生成模拟库。但为了创建目标库，有必要修改 Prep.bat、 CC.bat 和 lib.bat。

2.3.1 改编库批处理文件以适应不同系统

下面将通过 Mitsubishi M32C CPU 的改编示例显示如何改编文件。

改编 Prep.bat

Prep.bat 在 Makelib.bat 开始时调用。如上所述，其工作是设置所用工具的环境变量并设置环境变量 PATH，以便批处理文件无需指定绝对路径就可调用工具。假定编译器安装在文件夹 C:\MTOOL 中，文件 Prep.bat 类似如下：

```
@ECHO OFF
SET TOOLPATH=C:\MTOOL
REM ****************************************************************
REM   Set the variable PATH to be able to call the tools
SET PATH=%TOOLPATH%\BIN;%TOOLPATH%\LIB308;%PATH%
REM ****************************************************************
REM   Set the tool internal used variables
SET BIN308=%TOOLPATH%\BIN
SET INC308=%TOOLPATH%\INC308
SET LIB308=%TOOLPATH%\LIB308
SET TMP308=%TOOLPATH%\TMP
```

改编 CC.bat

CC.bat 的作用是编译传递的源文件并将目标文件的文件名添加到链接列表中。在启动 MakeLib.bat 时，它会创建与自身位置相关的如下子目录：

目录	内容
Lib	此文件夹应包含构建过程之后的库文件。
Temp\Output	将包含所有的编译器输出和链接列表文件。将在构建过程后删除。
Temp\Source	MakeLib.bat 使用此文件夹复制所有用于构建过程的源文件和头文件。将在构建过程后删除。

目标文件应创建（或移动）到 Temp\Output，这确保所有输出都在构建过程后删除。链接列表也应位于 output 文件夹中。如下为 Mitsubishi 编译器的示例：

```
@ECHO OFF
GOTO START
REM ****************************************************************
REM   Explanation of the used compiler options:
-silent : Suppresses the copyright message display at startup
-M82    : Generates object code for M32C/80 Series (Remove this switch
          for M16C80 targets)
-c      : Creates a relocatable file (extension .r30) and ends processing
-I      : Specifies the directory containing the file(s) specified in #include
-dir    : Specifies the destination directory
-OS     : Maximum optimization of speed followed by ROM size
-fFRAM  : Changes the default attribute of RAM data to far
-fETI   : Performs operation after extending char-type data to the int type
          (Extended according to ANSI standards)
:START
REM ****************************************************************
REM   Compile the passed source file with the Mitsubishi NC308 compiler
NC308 -silent -M82 -c -IInc -dir Temp\Output -OS -fFRAM -fETI Temp\Source\%1.c
REM ****************************************************************
REM   Pause if any problem occurs
IF ERRORLEVEL 1 PAUSE
REM ****************************************************************
REM   Add the file name of the object file to the link list
ECHO Temp\Output\%1.R30>>Temp\Output\Lib.dat
```

改编 Lib.bat

所有源文件编译完后，将从 MakeLib.bat 调用 Lib.bat，其作用是使用 CC.bat 创建的链接列表创建库文件。库文件的目标文件夹应是 MakeLib.bat 创建的 Lib 文件夹。以下为 Mitsubishi 库管理程序的示例：


```
@ECHO OFF
GOTO START
REM ********************************************************************
REM   Explanation of the used options:
-C : Creates new library file
@  : Specifies command file
:START
REM ********************************************************************
REM   Create the first part of the linker command file
ECHO -C Lib\GUI>Temp\Output\PARA.DAT
REM ********************************************************************
REM   Merge the first part with the link list to the linker command file
COPY Temp\Output\PARA.DAT+Temp\Output\Lib.dat Temp\Output\LINK.DAT
REM ********************************************************************
REM   Call the Mitsubishi librarian
LB308 @Temp\Output\LINK.DAT
REM ********************************************************************
REM   Pause if any problem occurs
IF ERRORLEVEL 1 PAUSE
```

2.4 要包含在项目中的 C 文件

一般而言，需要包含 emWin 的内核 C 文件、显示驱动、要使用的所有字体文件以及连同 emWin 一起定购的所有可选模块：

- 文件夹 Config 中的所有 C 文件
- 文件夹 GUI\Core 中的所有 C 文件
- 要使用的字体 （位于 GUI\Font 中）
- 显示驱动：文件夹 GUI\DisplayDriver 中的所有 C 文件。

其他软件包

如果要使用其他可选模块，也必须包含它们的 C 文件：

- 灰度转换功能：位于 GUI\ConvertMono 中的所有 C 文件
- 色彩转换功能：位于 GUI\ConvertColor 中的所有 C 文件
- 抗锯齿：位于 GUI\AntiAlias 中的所有 C 文件
- 存储设备：位于 GUI\MemDev 中的所有 C 文件
- VNC 支持：位于 GUI\VNC 中的所有 C 文件
- 小工具库：位于 GUI\Widget 中的所有 C 文件
- 窗口管理器：位于 GUI\WM 中的所有 C 文件

目标细节

- 对于通过端口/缓冲器访问的显示器，必须定义接口例程。所需例程的示例可在Samples\LCD_X中找到。
- 必须包含 GUI_X_embOS.c 或 GUI_X.c，以便使用 embOS 或其他 RTOS 进行定时以及与多任务相关的功能。

请确保将 GUI.h 包含在访问 emWin 的所有源文件中。

2.5 emWin 的配置

Config 文件夹应包含所有配置文件。“配置”一章详细说明了如何配置 emWin。
有以下类型的配置宏可用：

二进制开关 “B”

开关的值可为 0 或 1，0 表示去激活，而 1 表示激活（实际上除 0 以外的任何值都起作用，但是使用 1 时读取配置文件更容易）。这些开关可启用或禁用某些功能或行为。开关是最简单形式的配置宏。

数值 “N”

数值用在代码中某个位置，代替数值常数。典型示例是在显示器分辨率的配置中。

选择开关 “S”

选择开关用于在只能从多个选项中选择一项时选择一个选项。典型示例是选择要用的显示控制器类型，这时选定的数字表示哪个源代码 （在哪个显示驱动中）用于生成对象代码。

别名 “A”

像简单文本替换一样进行操作的宏。示例为定义 U8，预处理器将替换为不带符号字符。

函数替换 “F”

基本上宏可以像常规函数一样进行处理，尽管有某些限制，因为宏仍可像简单文本替换一样被置入代码中。函数替换主要用于向具有高度硬件依赖性的模块 （如对显示器的访问）添加特定的功能。此类型的宏通常用括号 （和可选参数）进行声明。

2.6 初始化 emWin

应使用以下函数来初始化和 “取消初始化” emWin，以便启动配置过程 （请参阅 “配置”（第 905 页）一章），或再次从存储器清除内部数据。

例程	描述
GUI_Init()	初始化 emWin 内部数据结构和变量。
GUI_Exit()	从存储器清除 emWin 内部数据。

GUI_Init()

描述

初始化 emWin 内部数据结构和变量。

原型

```
int GUI_Init(void);
```

返回值

如果成功，返回 0 ；如果显示驱动初始化失败，则返回其他值。

其他信息

使用任何 emWin 功能前强制执行此功能。唯一例外是为窗口设置创建标记（请参阅“WM_SetCreateFlags()”（第329页））。如果已使用窗口管理器，则在GUI_Init()内创建背景窗口。因此，如果在调用GUI_Init()前已设置创建标记，则会依据它们创建背景窗口。

GUI_Exit()

描述

从存储器清除 emWin 内部数据，以便可以进一步调用 GUI_Init()。

原型

```
void GUI_Exit(void);
```


其他信息

如果 emWin 代表不继续使用的应用程序部分，应使用此函数，因此必须能再次打开和关闭。请注意，调用 GUI_Exit 后， emWin 将不能正常工作，直至再次调用 GUI_Init()。

2.7 有目标硬件时使用 emWin

以下只是用emWin开始编程时要采取的通用步骤的基本概述。在后续章节中会进一步说明所有步骤。

第 1 步：配置 emWin

第一步通常是定制 emWin。有关位图转换器的详细信息，请参阅 “配置”（第 905 页）。

第 2 步：定义访问地址或访问例程

对于存储器映射显示控制器，只需在显示控制器的配置文件中定义显示器的访问地址。对于通过端口 / 缓冲器访问的显示器控制器，必须定义接口例程。所需例程的示例可在 Samples\LCD_X 中找到。

第 3 步：编译、链接和测试示例代码

emWin 附带有单任务和多任务环境的示例代码。编译、链接和测试这些小示例程序，直至熟悉这些操作为止。

第 4 步：修改示例程序

对示例程序进行简单修改。添加其他命令，如在显示器上显示不同尺寸的文本、显示直线等。

第 5 步：在多任务应用程序中：改编以适应您的 OS （必要时）

如果多个任务能同时访问显示器，则宏 GUI_MAXTASK 和 GUI_OS 以及文件 GUITask.c 发挥作用。有关详细信息和改编示例，请参阅 “配置”（第 905 页）。

第 6 步：使用 emWin 编写自己的应用程序

到目前为止，您应该已经清楚了解如何使用 emWin。考虑如何构建应用要求的程序，并通过调用适当的例程来使用 emWin。请查阅本手册稍后的参考章节，这些章节讨论了具体的 emWin 功能和可用的配置宏。

2.8 “Hello world”示例程序

下面所示为 “Hellow world”示例程序。如果希望查看更多基于 emWin 的应用程序示例以及进一步的简单教程应用程序，请查看 emWin 随附的 Sample 文件夹或访问 www.segger.com 上的 “emWin Samples”部分。

“Hello world”程序在早期就已用作 C 语言编程的起点，因为它基本上是可以编写的最小程序。以下所示为一个 emWin “Hello world”程序，在 emWin 随附的 Sample\Tutorial 文件夹中名为 BASIC_HelloWorld.c。
该程序的全部目的是将 “Hello world”写入显示器的左上角。为此，必须首先初始化应用程序的硬件、显示控制器和 GUI。只需在程序开始调用 GUI_Init() 即可初始化 emWin。在本示例中，我们假定您应用程序的硬件已经初始化。
Hello world 程序如下所示：

```
#include "GUI.h"

void MainTask(void) {
  GUI_Init();
  GUI_DispString("Hello world!");
  while(1);
}
```

向 “Hello world”程序添加功能

这个小程序目前能做的还不算很多，现在可以将功能扩展一点：在显示 “Hello world”以后，我们想要程序开始在屏幕上记数，以便估算向显示器的输出速度。只需将少量代码添加到主程序末尾的循环中，其作用实际上是调用显示十进制值的函数。
该示例在 Sample 文件夹中，名称为 BASIC_Hello1.c。

```
#include "GUI.h"

void MainTask(void) {
  int i=0;
  GUI_Init();
  GUI_DispString("Hello world!");
  while(1) {
    GUI_DispDecAt( i++, 20,20,4);
    if (i > 9999) {
      i = 0;
    }
  }
}
```

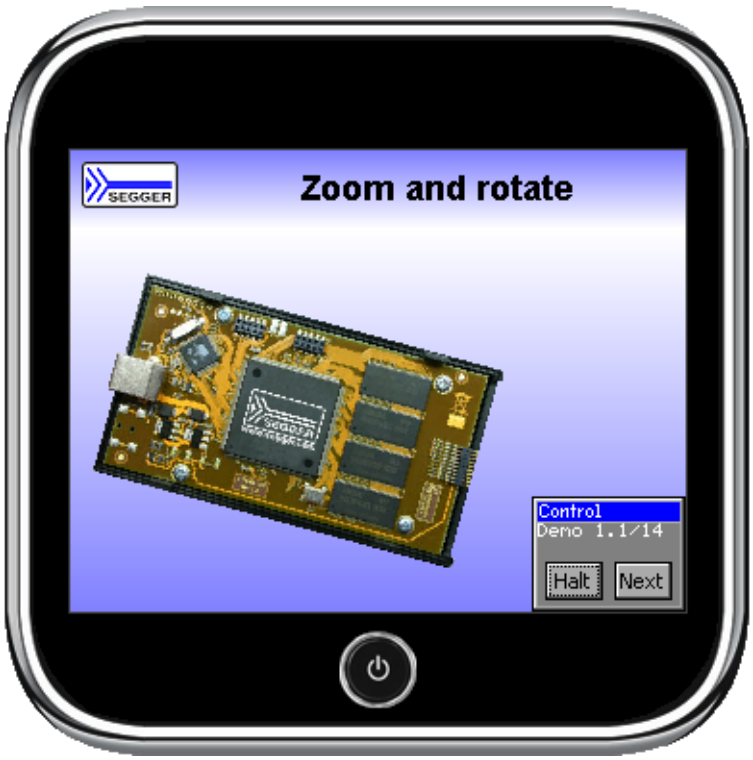

第 3 章

模拟

emWin 的 PC 模拟允许您在自己的 Windows PC 上使用本机（通常为 Microsoft）编译器编译同一 C 源代码，并为自己的应用程序创建可执行文件。这样做可允许以下操作：

- 在您的 PC 上设计用户界面 （无需硬件！）；
- 调试用户界面程序；
- 创建应用程序演示，该演示可用于讨论用户界面。

产生的可执行文件可通过电子邮件轻松发送。



3.1 使用模拟

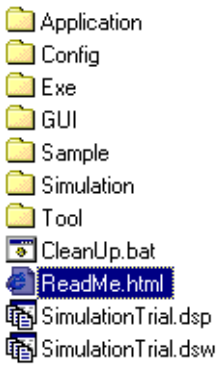
emWin 模拟需要 Microsoft Visual C++ （6.00 或更高版本）和随附的集成开发环境 (IDE)。您将在 PC 屏上看到 LCD 模拟，其在 X 和 Y 上具有相同的分辨率，且正确配置后可显示与您的 LCD 完全相同的颜色。模拟的整个图形库 API 和窗口管理器 API 与您目标系统上的相同。因为模拟使用与目标系统相同的 C 源代码，因此所有功能的执行方式与在目标硬件上完全相同。差别仅在于软件的较低级别：LCD 驱动。PC 模拟不使用实际的 LCD 驱动，而是使用写入到位图的模拟驱动。然后使用模拟的第二个线程将位图显示在屏幕上。此第二线程对应用程序不可见，其操作就像 LCD 例程直接写入显示器。

3.1.1 通过 emWin 试用版使用模拟

emWin 试用版包含一个可让您评估 emWin 所有可用功能的完整库。其中还包括 emWin 查看器（用于调试应用程序），以及字体转换器和位图转换器的演示版本。请记住，使用试用版时您不能查看 emWin 或模拟的源代码，但您仍然可通过试用版熟悉 emWin 的功能。

3.1.1.1 目录结构

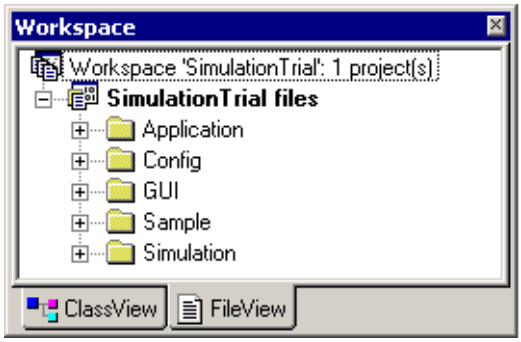
试用版中模拟的目录结构如右侧所示。下表说明了文件夹的内容：



目录	内容
Application	演示程序的源代码。
Config	用于构建库的配置文件。注意，头文件的更改对预编译库没有任何影响！
Exe	即时可用的演示程序。
GUI	使用库所需的库文件和包含文件。
Sample	模拟示例。
Simulation	模拟需要的文件。
Tool	emWin 查看器、位图转换器的演示版本和字体转换器的演示版本。

3.1.1.2 Visual C++ 工作空间

上面显示的根目录包括 Microsoft Visual C++ 工作空间 (`SimulationTrial.dsw`) 和项目文件 (`SimulationTrial.dsp`)。双击工作空间文件即可打开 Microsoft IDE。
Visual C++ 工作空间的目录结构与右侧显示的目录结构类似。



3.1.1.3 编译演示程序

演示程序的源文件位于 Application 目录中，为即用模拟，意味着您只需重建并启动它。注意，需要安装 Microsoft Visual C++ （6.00 或更高版本）后才能重建可执行文件。

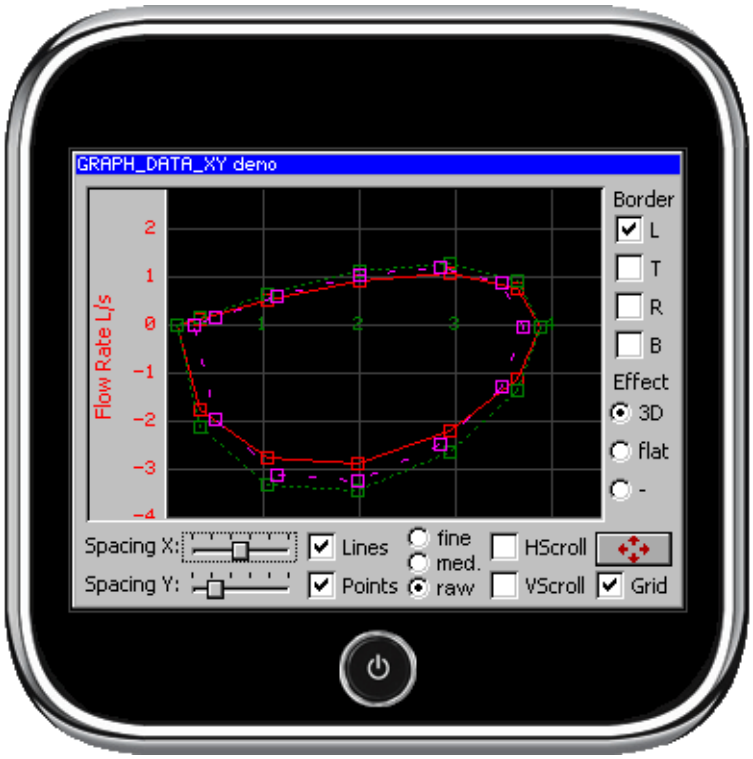
- 第 1 步：双击 SimulationTrial.dsw，打开 Visual C++ 工作空间。
- 第 2 步：从菜单选择 Build/Rebuild All （或按 F7），重建项目。
- 第 3 步：从菜单选择 Build/Start Debug/Go （或按 F5），启动模拟。

演示项目开始运行，可通过右键点击该项目并选择 “退出”随时将其关闭。

3.1.1.4 编译示例

Sample 目录包含演示 emWin 不同功能的即用示例，并提供一些典型应用示例。为构建其中任一个可执行文件，必须在项目中 “激活”其 C 源代码。使用以下步骤可轻松完成此操作：

- 第 1 步：右击工作空间的 Application 文件夹然后选择 Settings\General\Exclude from build，将 Application 文件夹从构建过程中排除。
- 第 2 步：双击工作空间的 Sample 文件夹，将其打开。右击添加要使用的示例并从构建中取消选择 Settings\General\Exclude。
- 第 3 步：如果示例包含其自身的配置文件 （LCDConf.c 和 / 或 SIMConf.c），则需要从构建过程中排除位于 config 文件夹下的默认配置文件。
- 第 4 步：从菜单选择 Build/Rebuild All （或按 F7），重建示例。
- 第 5 步：从菜单选择 Build/Start Debug/Go （或按 F5），启动模拟。上面所选示例的结果图示如下：



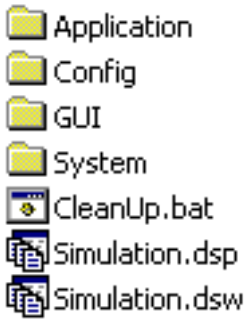
3.1.2 通过 emWin 源代码使用模拟

3.1.2.1 目录结构

模拟的根目录可位于您 PC 上的任何位置，例如 C:\Work\emWinSim。出现的目录结构将如右侧所示。此结构与我们为您的目标应用推荐的结构非常相似 （更多详细信息请参见第 3 章：“使用入门”）。
下表为文件夹的内容：

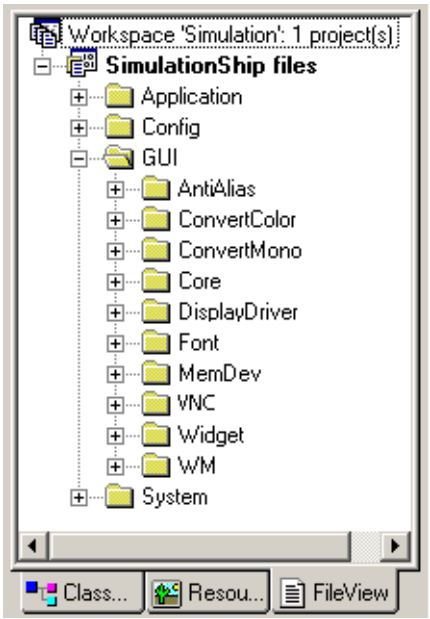
目录	内容
Doc	包含 emWin 文档
Sample	代码示例，在此文档后面介绍
Start	使用 emWin 创建新项目需要的所有项目
Tool	emWin 发货随附的工具

可通过创建 Start 文件夹副本开始新项目。它包含新项目的所有必需文件。包含 emWin 源代码的子目录位于 Start\GUI 文件夹下，并将包含与用于您目标（交叉）编译器同名目录完全相同的文件。GUI 子目录的文件不应更改，因为更改操作会使 emWin 很难更新到更新的版本。
Start\Config 目录包含配置文件，需要修改这些文件以反映您的目标硬件设置 （主要是可显示的 LCD 尺寸和颜色）。



3.1.2.2 Visual C++ 工作空间

上面显示的根目录包括 Microsoft Visual C++ 工作空间 (Simulation.dsw) 和项目文件 (Simulation.dsp)。利用工作空间可以修改应用程序，并在目标系统上进行编译之前对其进行调试。
Visual C++ 工作空间的目录结构外观类似右侧显示的目录结构。此处，GUI 文件夹是打开的，以显示 emWin 子目录。注意，您的 GUI 目录可能与图示不完全类似，具体取决于您所拥有的 emWin 的附加功能。Core、Font 和 LCDDriver 文件夹是基础 emWin 包的一部分，并将始终显示在工作空间目录中。



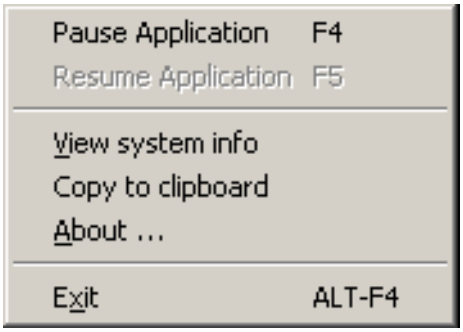
3.1.2.3 编译应用程序

此模拟包含一个或多个应用程序 C 文件（位于 Application 目录下），可以修改或移除，还可将其他文件添加到项目中。然后应在 Visual C++ 工作空间内重建程序以对其进行测试 / 调试。您对结果感到满意并想在自己的应用中使用该程序时，应能够在目标系统上编译这些相同的文件，并在目标显示器上获取相同的结果。使用模拟的常规步骤如下：

- 第 1 步：双击 Simulation.dsw，打开 Visual C++ 工作空间。
- 第 2 步：从菜单选择 Build/Rebuild All （或按 F7），编译项目。
- 第 3 步：从菜单选择 Build/Start Debug/Go （或按 F5），执行模拟。
- 第 4 步：使用您自己的标识或图片替换位图。
- 第 5 步：根据需要，通过编辑源代码或添加 / 删除文件进一步修改应用程序。
- 第 6 步：在 Visual C++ 内编译并运行应用程序以测试结果。根据需要继续进行修改和调试。
- 第 7 步：在目标系统上编译并运行应用程序。

3.1.3 模拟的高级功能

单击鼠标右键，会显示带有几个高级功能的上下文菜单：

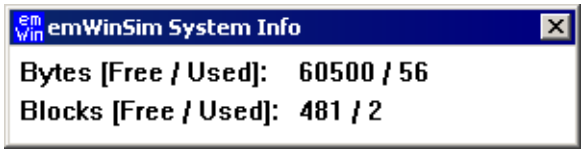


3.1.3.1 暂停和恢复

这些菜单项允许暂停并恢复模拟中当前运行的应用程序。通过按 <F4> 或 <F5> 可执行相同操作。尝试暂停一个已暂停的应用程序或尝试恢复一个已运行的应用程序都会导致错误消息。

3.1.3.2 查看系统信息

此菜单项打开另一个窗口，该窗口带有应用程序当前使用存储器的信息。该窗口通过显示可用字节和已用字节以及存储块的可用数目和已用数目，持续显示存储器消耗的当前状态。有关存储器管理器的详细信息，请参阅 “配置”（第 905 页）。



3.1.3.3 复制到剪贴板

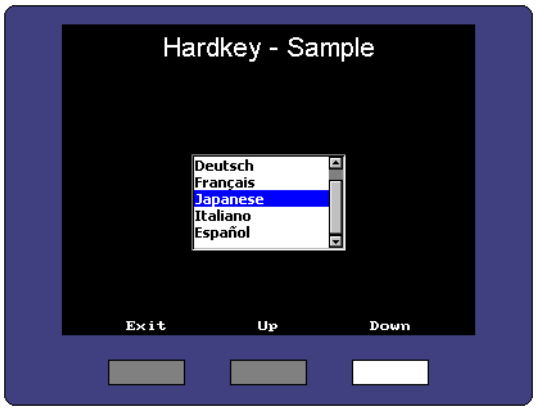
此菜单项将显示器的当前内容复制到剪贴板。此功能使用户可轻松记录其他应用程序。

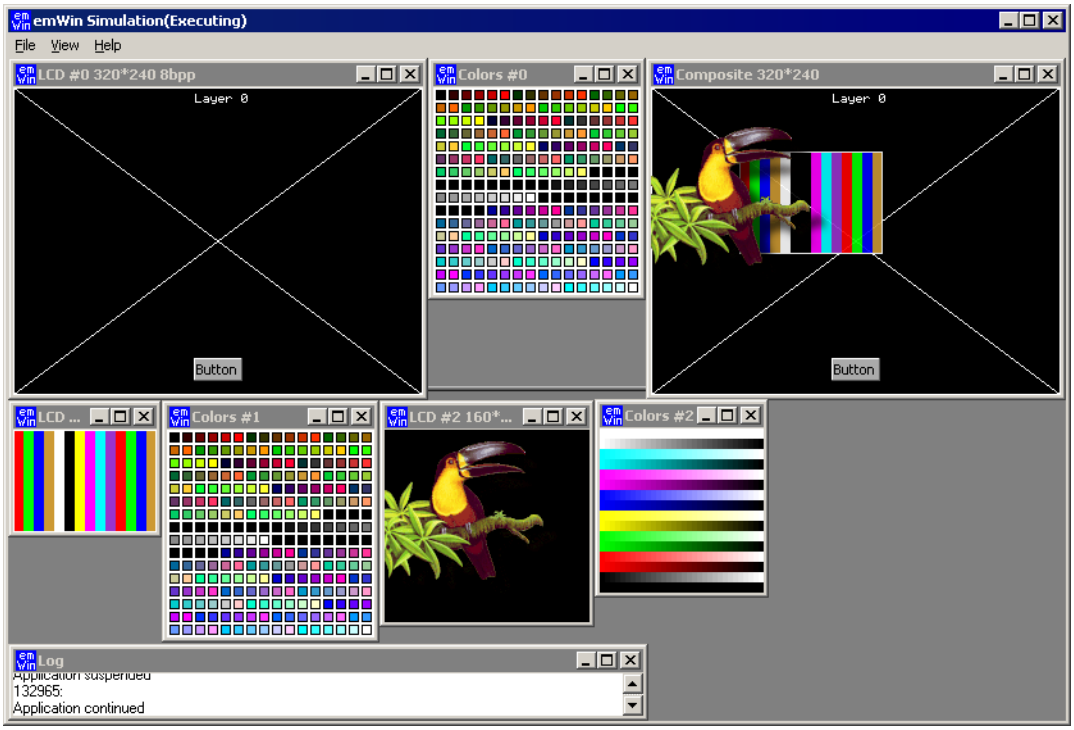
3.2 设备模拟

设备模拟支持 3 种视图：

- 生成的框架视图
- 自定义位图视图
- 窗口视图

下表显示了这三种不同的视图：

生成的框架视图	自定义位图视图
	

窗口视图


下面将详细介绍如何使用每个选项。

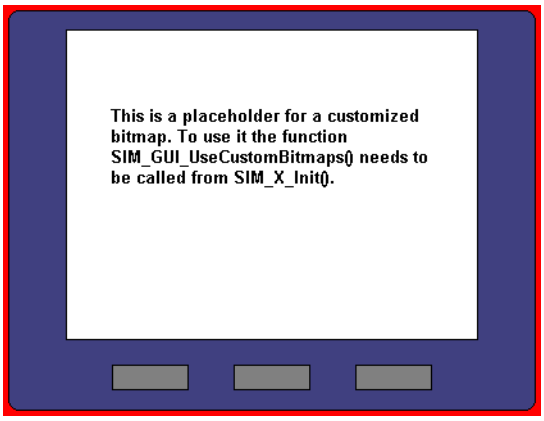
3.2.1 生成的框架视图

模拟将显示器显示在一个自动生成的包围该显示器的框架内。框架包含一个小按钮，该按钮默认情况下关闭应用程序。对于单层系统，这是模拟的默认操作。“单层系统”的意思是仅第一层进行了初始化。

3.2.2 自定义位图视图

模拟可在您选择的位图中显示模拟显示器，通常是您的目标设备。位图可用于模拟整个目标设备的操作。要模拟设备的外观，必需使用位图。

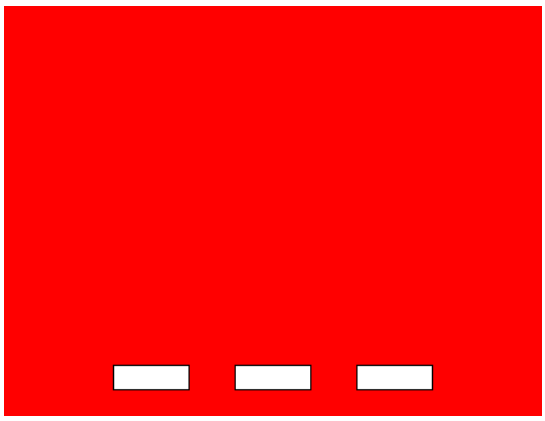
设备位图



第一个位图通常是一张设备照片（顶视图），并需要命名为 Device.bmp。此位图可以是单独的文件（与可执行文件在同一目录下），或者也可以将其作为应用程序中的资源包括。详细操作方法请见本章后面。此文件应提供一个像素大小与物理显示屏分辨率相同的模拟显示器区域。
如果存在任何要模拟的硬键，则位图还应显示处于未按下状态的所有这些硬键。
透明区域需要使用函数 SIM_GUI_SetTransColor() 定义的颜色着色，通常为鲜红色 (0xFF0000)。这些区域不必须是矩形。它们可以为任意形 （可达到一定复杂度，具体取决于您操作系统的限制，但通常都有足够的随意性）。鲜红色是透明区域的默认颜色，主要是因为大多数位图通常都不包含此颜色。要使用鲜红色位图，可使用函数SIM_GUI_SetTransColor()更改默认的透明颜色。

硬键位图

需要第二个位图文件定义硬键，且必须将其命名为 Device1.bmp。它包含处于按下状态的按钮。无硬键区域需要使用透明色填充。这仅是一个简短说明。有关如何模拟硬键的更多详细信息，请参见 “硬键模拟”（第 52 页）。



使用单独的文件

启动模拟时，它会检查可执行文件的目录是否包含位图文件 Device.bmp 和 Device1.bmp。如果这些文件可用，则会自动使用并忽略资源位图。注意，这仅在单层系统下有效。

添加位图到应用程序资源文件

模拟的资源文件在 System\Simulation\Res\Simulation.rc 下，它包含以下部分：

```
/////////////////////////////////////////////////////////////////////////
//
// Customizable bitmaps
//
IDB_DEVICE              BITMAP  DISCARDABLE     "Device.bmp"
IDB_DEVICE1             BITMAP  DISCARDABLE     "Device1.bmp"
```

本节可用于设置自定义设备文件。有关更多信息，请参见 Win32 文档。

3.2.3 窗口视图

模拟多层系统的默认方法是在单独的窗口显示每一层，而不使用位图或生成的框架。

3.3 设备模拟 API

所有的设备模拟 API 函数都应在设置阶段调用。调用应从例程 SIM_X_Config() 内部执行，该函数位于配置文件夹的文件 SIMConf.c 中。下面的示例在设置中调用 SIM_SetLCDPos()：

```
#include "LCD_SIM.h"

void SIM_X_Config() {
  SIM_GUI_SetLCDPos(50, 20); // Define the position of the LCD in the bitmap}
}
```

下表按字母顺序列出可用的设备模拟相关例程。例程的详细说明如下：

例程	描述
SIM_GUI_ShowDevice()	管理设备位图的可见性。
SIM_GUI_SetCallback()	设置用于接收模拟窗口句柄的回调函数。
SIM_GUI_SetCompositeColor()	设置复合窗口的背景色。（仅用于多层系统）
SIM_GUI_SetCompositeSize()	设置复合窗口的尺寸。（仅用于多层系统）
SIM_GUI_SetLCDColorBlack()	将要使用的颜色设置为黑色 （单色显示器）。
SIM_GUI_SetLCDColorWhite()	将要使用的颜色设置为白色 （单色显示器）。
SIM_GUI_SetLCDPos()	在目标设备位图中设置模拟 LCD 的位置。
SIM_GUI_SetMag()	设置 X 和 / 或 Y 轴的放大系数。
SIM_GUI_SetTransColor()	设置要用于透明区域的颜色 （默认：0xFF0000）。
SIM_GUI_UseCustomBitmaps()	指示模拟使用来自应用程序资源文件的自定义位图。

SIM_GUI_ShowDevice()

描述

此函数可用于管理模拟的周围设备位图的可见性。

原型

```
void SIM_GUI_ShowDevice(int OnOff);
```

参数	描述
OnOff	1 表示显示位图， 0 表示隐藏位图。

其他信息

在多层系统上，设备位图默认情况下不显示，在单层系统上则位图可见。如果需要不同的特性，则此函数可用于设置设备位图的可见性。

SIM_GUI_SetCallback()

描述

如果不只是需要模拟显示器窗口或硬键，则可设置回调函数来接收模拟的窗口句柄。这可实现在显示窗口外侧添加附加控件 （如 LED 或滑块）等功能。请注意不能在此处使用 emWin 函数。

原型

```
void SIM_GUI_SetCallback(int (* _pfInfoCallback)(SIM_GUI_INFO * pInfo));
```

参数	描述
_pfInfoCallback	指向回调函数的指针。此函数必须有一个指向 SIM_GUI_INFO 结构的指针作为参数

SIM_GUI_INFO 结构的内容		
类型	名称	描述
HWND	hWndMain	主窗口句柄
HWND	ahWndLCD[16]	显示层的句柄数组
HWND	ahWndColor[16]	图层句柄的数组

SIM_GUI_SetCompositeColor()

描述

在模拟多层系统时，每层都可在其自己的窗口中显示。但物理显示器只有一个区域，它显示层混合的结果。模拟在复合窗口中显示结果，该窗口可具有其自己的尺寸，与层无关。在复合窗口内每层都可有其自己的位置和尺寸。这意味着层不必覆盖整个区域。这种情况下 （还针对透明效果），此函数设置复合窗口的默认背景色。

原型

```
void SIM_GUI_SetCompositeColor(U32 Color);
```

参数	描述
Color	要使用的背景颜色。

SIM_GUI_SetCompositeSize()

描述

如上所述，在 SIM_GUI_SetCompositeColor() 下面，复合窗口的尺寸与层的尺寸无关。此函数用于设置复合窗口的尺寸。

原型

```
void SIM_GUI_SetCompositeSize(int xSize, int ySize);
```

参数	描述
xSize	水平像素尺寸。
ySize	垂直像素尺寸。

示例

下面显示了一个典型应用 （对于多层系统）：

```
void SIM_X_Config() {
  SIM_GUI_SetCompositeSize(240, 320);  // Set size of composite window
  SIM_GUI_SetCompositeColor(0x800000); // Define background color of composite window
}
```


SIM_GUI_SetLCDColorBlack(), SIM_GUI_SetLCDColorWhite()

描述

分别将要使用的颜色设置为黑色或白色 （在单色显示器上）。

原型

```
int SIM_GUI_SetLCDColorBlack(int DisplayIndex, int Color);
int SIM_GUI_SetLCDColorWhite(int DisplayIndex, int Color);
```

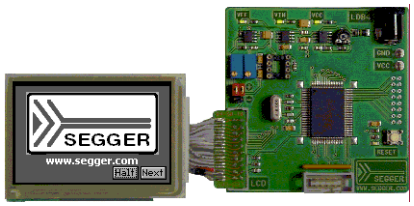
参数	描述
DisplayIndex	留待将来使用，必须为 0。
Color	颜色的 RGB 值。

其他信息

这些函数可用于模拟显示器的真实背景色。
默认颜色值为黑色和白色，或 0x000000 和 0xFFFFFF。

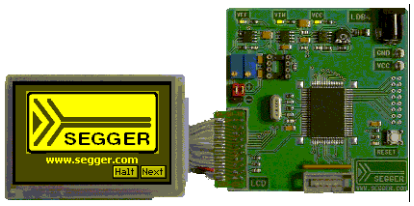
使用默认设置的示例

```
void SIM_X_Config() {
  SIM_GUI_SetLCDPos(14,84);                 // Define the position of the LCD
                                            // in the bitmap
  SIM_GUI_SetLCDColorBlack (0, 0x000000);   // Define the color used as black
  SIM_GUI_SetLCDColorWhite (0, 0xFFFFFF);   // Define the color used as white
  (used for colored monochrome displays)
}
```



使用黄色替代白色的示例

```
void SIM_X_Config() {
  SIM_GUI_SetLCDPos(14,84);                 // Define the position of the LCD
                                            // in the bitmap
  SIM_GUI_SetLCDColorBlack (0, 0x000000);   // Define the color used as black
  SIM_GUI_SetLCDColorWhite (0, 0x00FFFF);   // Define the color used as white
  (used for colored monochrome displays)
}
```



SIM_GUI_SetLCDPos()

描述

在目标设备位图中设置模拟 LCD 的位置。

原型

```
void SIM_GUI_SetLCDPos(int x, int y);
```

参数	描述
x	模拟 LCD 左上角的 X 位置 （像素值）。
y	模拟 LCD 左上角的 Y 位置 （像素值）。

其他信息

X 和 Y 位置与目标设备位图相关，因此位置 (0,0) 指向位图而不是实际 LCD 的左上角 （原点）。
只有被模拟屏幕的原点需要指定。您显示器的分辨率应已反映在 Config 目录下的配置文件中。
使用此函数可启用位图 Device.bmp 和 Device1.bmp。注意，这些值需要 >= 0 才能启用位图。
如果应禁用设备位图，则在 SIM_X_Init() 中删除对此函数的调用。

SIM_GUI_SetMag()

描述

设置 X 和 / 或 Y 轴的放大系数。

原型

```
void SIM_GUI_SetMag(int MagX, int MagY);
```

参数	描述
MagX	X 轴的放大系数。
MagY	Y 轴的放大系数。

其他信息

默认情况下，对于模拟显示器的每个像素模拟操作都使用 PC 上的一个像素。此函数的使用对小型显示器非常有用。如果使用了设备位图并且放大率 >1，则设备位图需要修改以适应放大率。设备位图不自动放大。

SIM_GUI_SetTransColor()

描述

设置设备或硬键位图透明区域要使用的颜色。

原型

```
I32 SIM_GUI_SetTransColor(I32 Color);
```

参数	描述
Color	颜色的 RGB 值，采用格式 00000000RRRRRRRRGGGGGGGGBBBBBBBB。

其他信息

透明区的默认设置为鲜红色 (0xFF0000)。
如果您的位图包含同样的红色阴影，您通常只需更改此设置。

SIM_GUI_UseCustomBitmaps()

描述

如本章前面所述，可以使用来自应用程序资源文件的设备位图。此函数指示模拟操作使用来自应用程序资源文件的设备和硬键位图，且不生成默认的框架位图。

原型

```
void SIM_GUI_UseCustomBitmaps(void);
```

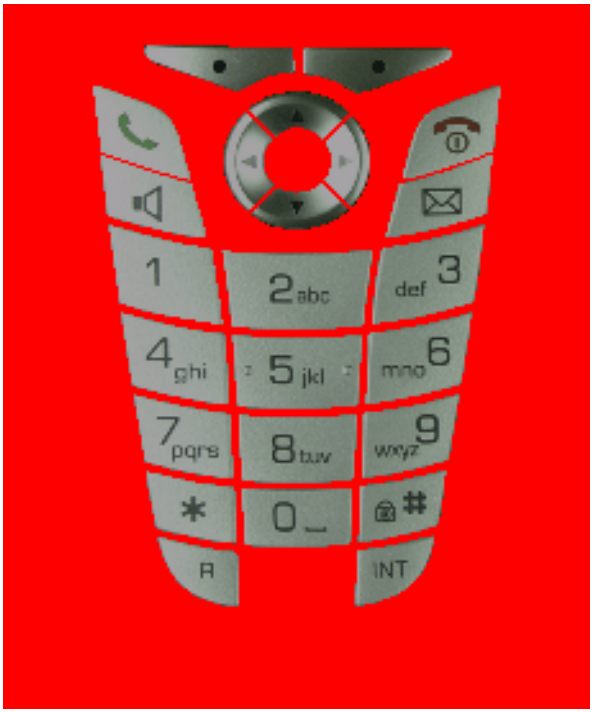
其他信息

emWin发货时默认包含2个位图Device.bmp和Device1.bmp，位于Start\System\Simulation\Res中，可用作您自己位图的起点。

3.4 硬键模拟

硬键模拟只能在自定义位图视图中使用。硬键还可以作为设备的一部分进行模拟，并可使用鼠标指针选择。该理念能够区分模拟设备上的键或按钮是否被按下。只要按住鼠标按钮，则认为硬键被“按下”；释放鼠标按钮或将指针从硬键上移走会“取消按下”该键。按下和取消按下之间的切换操作还可使用例程 SIM_HARDKEY_SetMode() 指定。

为了模拟硬键，您需要设备的第二个位图，该位图除了键本身（处于按下状态）外，其他部分为透明。如本章之前所述，此位图可以是目录中的单独文件，或作为可执行文件中的资源包括。硬键可为任意形状，只要它们在 Device.bmp 和 Device1.bmp 中的像素尺寸完全相同。以下示例说明了这一点：

设备位图：未按下硬键状态 (Device.bmp)	设备硬键位图：按下硬键状态 (Device1.bmp)
	

使用鼠标“按下”键时，硬键位图 (Device1.bmp) 的相应部分将覆盖设备位图，以显示处于按下状态的该键。

这些键可以定期轮询，以确定其状态（按下 / 未按下）是否更改以及是否需要更新。或者，回调例程可设置为在硬键状态更改时触发特定操作。

3.4.1 硬键模拟 API

硬键模拟函数是随 emWin 发货的标准模拟程序的一部分。如果使用用户定义的 emWin 模拟，则这些函数可能不可用。下表按字母顺序在各自类别下列出可用的硬键模拟相关例程。例程的详细说明如下：

例程	描述
SIM_HARDKEY_GetNum()	返回可用硬键的数目。
SIM_HARDKEY_GetState()	返回指定硬键的状态 （0：未按下， 1：按下）。
SIM_HARDKEY_SetCallback()	设置在指定硬键的状态更改时要执行的回调例程。
SIM_HARDKEY_SetMode()	设置指定硬键的特性 （默认值 = 0：不切换）。
SIM_HARDKEY_SetState()	设置指定硬键的状态 （0：未按下， 1：按下）。

SIM_HARDKEY_GetNum()

描述

返回可用硬键的数目。

原型

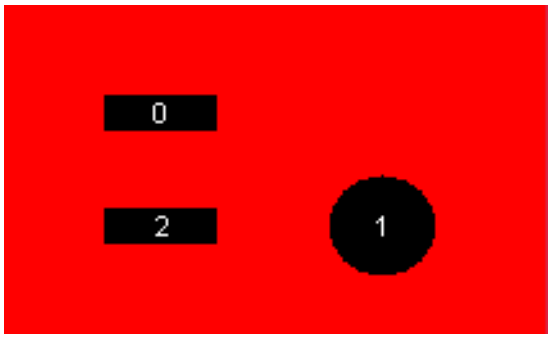
```
int SIM_HARDKEY_GetNum(void);
```

返回值

位图中找到的可用硬键的数目。

其他信息

硬键的编号顺序为标准的读取顺序 （从左到右，然后从顶至底）。因此最先找到的是硬键最顶部的像素，无论其水平位置如何。例如，在下面的位图中，对硬键进行了标记，使它们在其他函数中被 KeyIndex 参数引用：



建议调用此函数以验证位图是否正确加载。

SIM_HARDKEY_GetState()

描述

返回指定硬键的状态。

原型

```
int SIM_HARDKEY_GetState(unsigned int KeyIndex);
```

参数	描述
KeyIndex	硬键的索引（0 = 第一个键的索引）。

返回值

指定硬键的状态：
0：未按下
1：按下

SIM_HARDKEY_SetCallback()

描述

设置在指定硬键的状态更改时要执行的回调例程。

原型

```
SIM_HARDKEY_CB * SIM_HARDKEY_SetCallback(unsigned int KeyIndex,
                                         SIM_HARDKEY_CB * pfCallback);
```

参数	描述
KeyIndex	硬键的索引（0 = 第一个键的索引）。
pfCallback	指向回调例程的指针。

返回值

指向上一回调例程的指针。

其他信息

注意，如果需要在回调函数中调用 GUI 函数，则必须启用多任务支持。没有多任务支持，则只应使用可以在中断例程中调用的 GUI 函数。
回调例程必须具有以下原型：

原型

```
typedef void SIM_HARDKEY_CB(int KeyIndex, int State);
```

参数	描述
KeyIndex	硬键的索引（0 = 第一个键的索引）。
State	指定硬键的状态（见下表）。

参数 State 的允许值	
0	未按下。
1	按下。

SIM_HARDKEY_SetMode()

描述

设置指定硬键的特性。

原型

```
int SIM_HARDKEY_SetMode(unsigned int KeyIndex, int Mode);
```

参数	描述
KeyIndex	硬键的索引 （0 = 第一个键的索引）。
Mode	操作模式 （见下表）。

参数 Mode 的允许值	
0	正常操作 （默认）。
1	切换操作。

其他信息

正常 （默认）的硬键操作表示只要鼠标按钮按在键上，就认为键被按下。当鼠标释放或从硬键上移走时，认为键被取消按下。
对于切换操作，每次单击鼠标会将硬键的状态在按下和取消按下间切换。也就是说，如果您在硬键上单击鼠标且其变为按下状态，则该硬键将保持按下状态直到您再次在其上单击鼠标。

SIM_HARDKEY_SetState()

描述

设置指定硬键的状态。

原型

```
int SIM_HARDKEY_SetState(unsigned int KeyIndex, int State);
```

参数	描述
KeyIndex	硬键的索引 （0 = 第一个键的索引）。
State	指定硬键的状态 （见下表）。

参数 State 的允许值	
0	未按下。
1	按下。

其他信息

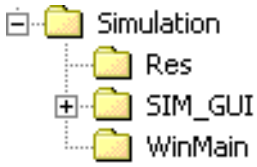
仅当 SIM_HARDKEY_SetMode() 设置为 1 （切换模式）时此函数才可用。

3.5 集成 emWin 模拟到现有模拟中

要将 emWin 模拟集成到现有模拟中，模拟的源代码不是必需的。模拟的源代码通常不随 emWin 一同发货。它是单独 （可选）的软件项目，不包括在 emWin 基础包中。
通常不需要 emWin 模拟的源代码，但可作为可选软件项目提供。如本章前面所述，基础包和试用版包含模拟库。例如，如果要将 emWin 模拟添加到现有硬件或实时内核 (RTOS) 模拟中，则可使用此库的 API 函数。
要将 emWin 模拟添加到现有模拟中（采用 C 或 C++ 编写，使用 Win32 API），只需添加几行代码。

3.5.1 目录结构

System 文件夹的子文件夹 Simulation 包含 emWin 模拟。目录结构显示在右侧。下表说明了该子文件夹的内容：



目录	内容
Simulation	要与模拟源代码一起和不一起使用的模拟源文件和头文件。该文件夹还包含即用模拟库。
Res	资源文件。
SIM_GUI	GUI 模拟源代码 （可选）。
WinMain	包含 WinMain 例程。

3.5.2 使用模拟库

以下步骤显示如何使用模拟库将 emWin 模拟集成到现有模拟中：

- 第 1 步：将模拟库 GUISim.lib 添加到项目中。
- 第 2 步：按第 2.1.1 章 “子目录”中所述将所有 GUI 文件添加到项目中。
- 第 3 步：按第 2.1.2 章 “包含目录”中所述将包含目录添加到项目中。
- 第 4 步：修改 WinMain。

3.5.2.1 修改 WinMain

每个 Windows Win32 程序都以 WinMain() 开始（与普通的命令行 C 程序相反，其以 main() 开始）。所有要做的就是将几行代码添加到此例程中。
需要添加以下函数调用 （通常按照其在下面应用程序代码示例中显示的顺序）：

- SIM_GUI_Init
- SIM_GUI_CreateLCDWindow
- CreateThread
- SIM_GUI_Exit

3.5.2.2 应用程序示例

下面的应用程序在 Sample\WinMain\SampleApp.c 下提供，显示了如何将 emWin 模拟集成到现有应用程序中：

```
#include <windows.h>
#include "GUI_SIM_Win32.h"
void MainTask(void);

/*********************************************************************
*
*       _Thread
*/
static DWORD __stdcall _Thread(void* Parameter) {
  MainTask();
  return 0;
}

/*********************************************************************
*
*       _WndProcMain
*/
static LRESULT CALLBACK _WndProcMain(HWND   hWnd,   UINT   message,
                                     WPARAM wParam, LPARAM lParam) {
  SIM_GUI_HandleKeyEvents(message, wParam);
  switch (message) {
  case WM_DESTROY:
    PostQuitMessage(0);
    break;
  }
  return DefWindowProc(hWnd, message, wParam, lParam);
}

/*********************************************************************
*
*       _RegisterClass
*/
static void _RegisterClass(HINSTANCE hInstance) {
  WNDCLASSEX wcex;
  memset(&wcex, 0, sizeof(wcex));
  wcex.cbSize        = sizeof(WNDCLASSEX);
  wcex.hInstance     = hInstance;
  wcex.style         = CS_HREDRAW | CS_VREDRAW;
  wcex.lpfnWndProc   = (WNDPROC)_WndProcMain;
  wcex.hIcon         = 0;
  wcex.hCursor       = LoadCursor(NULL, IDC_ARROW);
  wcex.hbrBackground = (HBRUSH)(COLOR_APPWORKSPACE + 1);
  wcex.lpszMenuName  = 0;
  wcex.lpszClassName = "GUIApplication";
  RegisterClassEx(&wcex);
}

/*********************************************************************
*
*       WinMain
*/
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine, int nCmdShow) {
  DWORD ThreadID;
  MSG   Msg;
  HWND  hWndMain;
  /* Register window class */
  _RegisterClass(hInstance);
  /* Create main window    */
  hWndMain = CreateWindow("GUIApplication", "Application window",
                          WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN | WS_VISIBLE,
                          0, 0, 328, 267, NULL, NULL, hInstance, NULL);
  /* Initialize the emWin simulation and create a LCD window */
  SIM_GUI_Init(hInstance, hWndMain, lpCmdLine, "embOSIAR - emWin Simulation");
  SIM_GUI_CreateLCDWindow(hWndMain, 0, 0, 320, 240, 0);
  /* Create a thread which executes the code to be simulated */
  CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)_Thread, NULL, 0, &ThreadID);
  /* Main message loop */
  while (GetMessage(&Msg, NULL, 0, 0)) {
    TranslateMessage(&Msg);
    DispatchMessage(&Msg);
  }
  SIM_GUI_Exit();
}
```


3.5.3 集成到 embOS 模拟中

3.5.3.1 WinMain

以下示例代码显示如何修改 emWin 模拟的现有 WinMain 以集成 embOS 模拟。红色行将添加到 WinMain 以初始化 emWin 模拟、创建模拟窗口并退出 emWin 模拟：

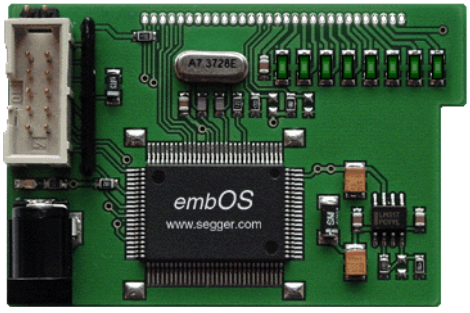
```
...
#include "GUI_SIM_Win32.h"
...
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine, int nCmdShow) {
MSG    Msg;
HACCEL hAccelTable;
  HWND   hWndMain;
  BITMAP BmpDevice;
  DWORD  ThreadID;
  /* Init global data */
  _StopHyperThreading();
  _hInst = hInstance;
  /* Register main window class */
  _RegisterClass();
  /* Load bitmap */
  _hBmpDevice = (HBITMAP)LoadImage(_hInst,
                                   (LPCTSTR) IDB_DEVICE,
                                   IMAGE_BITMAP,  0, 0, 0);
  _hMenuPopup = LoadMenu(_hInst, (LPCSTR)IDC_CONTEXTMENU);
  _hMenuPopup = GetSubMenu(_hMenuPopup, 0);
  /* Create main window    */
  GetObject(_hBmpDevice, sizeof(BmpDevice), &BmpDevice);
  hWndMain = CreateWindowEx(WS_EX_TOPMOST, _sWindowClass,
                            "embOSIAR Simulation",
                            WS_SYSMENU | WS_CLIPCHILDREN | WS_POPUP | WS_VISIBLE,
                            10, 20, BmpDevice.bmWidth, BmpDevice.bmHeight,
                            NULL, NULL, _hInst, NULL);
  if (!hWndMain) {
    return 1;   /* Error */
  }
  /* Init emWin simulation and create window */
  SIM_GUI_Init(hInstance, hWndMain, lpCmdLine, "embOSIAR - emWin Simulation");
  SIM_GUI_CreateLCDWindow(hWndMain, 80, 50, 128, 64, 0);
  /* Show main window */
  ShowWindow(hWndMain, nCmdShow);
  /* Load accelerator table */
  hAccelTable = LoadAccelerators(_hInst, (LPCTSTR)IDC_WINMAIN);
  /* application initialization:*/
  CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Thread, NULL, 0, &ThreadID);
  /* main message loop */
  if (SIM_Init(hWndMain) == 0) {
    while (GetMessage(&Msg, NULL, 0, 0)) {
      if (!TranslateAccelerator(Msg.hwnd, hAccelTable, &Msg)) {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
      }
    }
  }
  /* Exit emWin simulation */
  SIM_GUI_Exit();
  return 0;
}
```


3.5.3.2 目标程序 (main)

emWin API 可从一个或多个目标线程调用。没有 RTOS 时， WIN32 API 函数 CreateThread 通常用于创建调用 emWin API 的目标线程。在 RTOS 模拟内，目标任务 / 线程（由模拟 RTOS 创建）用于调用 emWin API。换句话说：使用 OS_CreateTask 创建用户界面任务。下面是修改后的 embOS 启动应用程序：

```
#include "RTOS.H"
#include "HW_LED.h"
#include "GUI.h"
OS_STACKPTR int Stack0[128], Stack1[128], Stack2[2000]; /* Task stacks */
OS_TASK         TCB0,        TCB1,        TCB2;         /* Task-control-blocks */

void Task0(void) {
  while(1) {
    HW_LED_Toggle0();
    OS_Delay(100);
  }
}

void Task1(void) {
  while(1) {
    HW_LED_Toggle1();
    OS_Delay(500);
  }
}

void main(void) {
  GUI_COLOR aColor[] = {GUI_RED, GUI_YELLOW};
  GUI_Init();
  while(1) {
    int i;
    for (i = 0; i < 2; i++) {
      GUI_Clear();
      GUI_SetColor(aColor[i]);
      GUI_SetFont(&GUI_FontComic24B_ASCII);
      GUI_DispStringAt("Hello world!", 1, 1);
      OS_Delay(200);
    }
  }
}

/*************************************************************
*
*       main
*/
#include <windows.h>
void main(void) {
  OS_IncDI();           /* Initially disable interrupts  */
  OS_InitKern();        /* initialize OS                 */
  OS_InitHW();          /* initialize Hardware for OS    */
  /* You need to create at least one task here !*/
  OS_CREATETASK(&TCB0, "HP Task",  Task0,    100, Stack0);
  OS_CREATETASK(&TCB1, "LP Task",  Task1,     50, Stack1);
  OS_CREATETASK(&TCB2, "GUI Task", MainTask,  80, Stack2);
  OS_Start();           /* Start multitasking            */
}
```

下表显示集成 emWin 模拟之前和之后的模拟：

之前	之后
	

3.5.4 GUI 模拟 API

下表按字母顺序在各自类别下列出用户定义模拟程序可使用的例程。这些函数仅可用于 emWin 模拟的源代码。例程的详细说明如下：

例程	描述
SIM_GUI_CreateLCDInfoWindow()	创建一个窗口，显示尺寸和位置给定的选定层的可用颜色。
SIM_GUI_CreateLCDWindow()	创建尺寸和位置给定的 LCD 窗口。
SIM_GUI_Exit()	停止 GUI 模拟。
SIM_GUI_Init()	初始化 GUI 模拟。
SIM_GUI_SetLCDWindowHook()	设置 LCD 窗口接收消息时要调用的挂钩函数。

SIM_GUI_CreateLCDInfoWindow()

描述

创建一个窗口，显示选定层的可用颜色。

原型

```
HWND SIM_GUI_CreateLCDInfoWindow(HWND hParent,
                                 int x, int y, int xSize, int ySize
                                 int LayerIndex);
```

参数	描述
hParent	父窗口的句柄。
x	父坐标中的 X 位置。
y	父坐标中的 Y 位置。
xSize	新窗口的 X 像素尺寸。如果使用 1 和 8 之间的颜色深度，应为 160，如果在高色彩模式下工作，则为 128。
ySize	新窗口的 Y 像素尺寸。如果使用 1 和 8 之间的颜色深度，应为 160，如果在高色彩模式下工作，则为 128。
LayerIndex	要显示层的索引。

其他信息

创建的颜色窗口没有框架、标题栏和按钮。

示例

```
SIM_GUI_CreateLCDInfoWindow(hWnd, 0, 0, 160, 160, 0);
```

截屏

SIM_GUI_CreateLCDWindow()

描述

创建一个窗口，模拟在给定位置具有给定尺寸的 LCD 显示器。

原型

```
HWND SIM_GUI_CreateLCDWindow(HWND hParent,
                             int x, int y, int xSize, int ySize
                             int LayerIndex);
```

参数	描述
hParent	父窗口的句柄。
x	父坐标中的 X 位置。
y	父坐标中的 Y 位置。
xSize	新窗口的 X 像素尺寸。
ySize	新窗口的 Y 像素尺寸。
LayerIndex	要显示层的索引。

其他信息

所有到指定层的显示输出都将在此窗口显示。窗口尺寸应与 LCDConf.c 中的配置相同。
创建的模拟窗口没有框架、标题栏和按钮。

SIM_GUI_Exit()

描述

此函数应在模拟返回到调用进程之前调用。

原型

```
void SIM_GUI_Exit(void);
```

SIM_GUI_Init()

描述

此函数初始化 emWin 模拟，并应在任何其他 SIM_GUI... 函数调用之前调用。

原型

```
int SIM_GUI_Init(HINSTANCE hInst, HWND hWndMain,
                 char * pCmdLine, const char * sAppName);
```

参数	描述
hInst	传递到 WinMain 的当前实例的句柄。
hWndMain	模拟主窗口的句柄。
pCmdLine	指向传递到 WinMain 的命令行的指针
sAppName	指向包含应用程序名称的字符串的指针。

其他信息

要显示消息框，则使用参数 hWndMain 和 sAppName。

SIM_GUI_SetLCDWindowHook()

描述

设置如果 LCD 窗口接收消息时要从模拟调用的挂钩函数。

原型

```
void SIM_GUI_SetLCDWindowHook(SIM_GUI_tfHook * pfHook);
```

参数	描述
pfHook	指向挂钩函数的指针。

挂钩函数的原型

```
int Hook(HWND hWnd, UINT Message, WPARAM wParam, LPARAM lParam,
         int * pResult);
```

参数	描述
hWnd	LCD 窗口的句柄。
Message	从操作系统接收的消息。
wParam	系统传递的 wParam 消息参数。
lParam	系统传递的 lParam 消息参数。
pResult	指向用作返回代码的整数的指针，该返回代码表示消息是否已被挂钩函数处理。

返回值

如果消息已处理，则挂钩函数将返回 0。此时， GUI 模拟忽略该消息。

第 4 章

查看器

如果调试应用程序时使用模拟操作，则在对源代码执行单步调试时将无法看到显示输出。查看器的主要作用就是解决这一问题。模拟调试时，它会显示模拟屏幕的内容。

查看器可以提供以下额外功能：

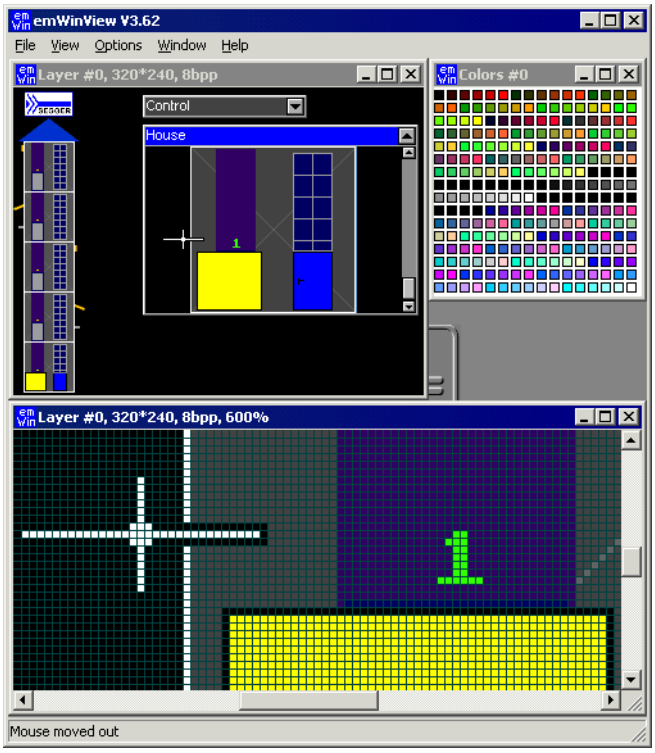
- 每个图层都有多个窗口
- 在单个窗口中查看整个虚拟图层
- 放大每个图层窗口
- 使用多个图层时复合视图

4.1 使用查看器

查看器可以：

- 为任一图层 / 显示打开多个窗口
- 放大某个图层 / 显示的任一区域
- 查看各个图层/显示的内容以及多层配置的复合视图
- 使用虚拟屏幕支持时，查看虚拟屏幕和可见显示的内容

屏幕截图显示查看器显示单层配置的输出。左上角显示模拟显示。右上角的窗口展示显示配置的可用颜色。查看器底部是展示模拟显示放大区域的辅助显示窗口。开始调试应用程序时，查看器会为每个图层展示一个显示窗口和颜色窗口。多图层配置时，复合视图窗口也可见。



4.1.1 使用模拟和查看器

如果调试应用程序时使用模拟操作，则在对源代码执行单步调试时将无法看到显示输出。这是由于 Win32 的限制：如果一个线程（正在进行调试）停止，该进程的其他所有线程也会停止。这包括在屏幕上输出模拟显示的线程。

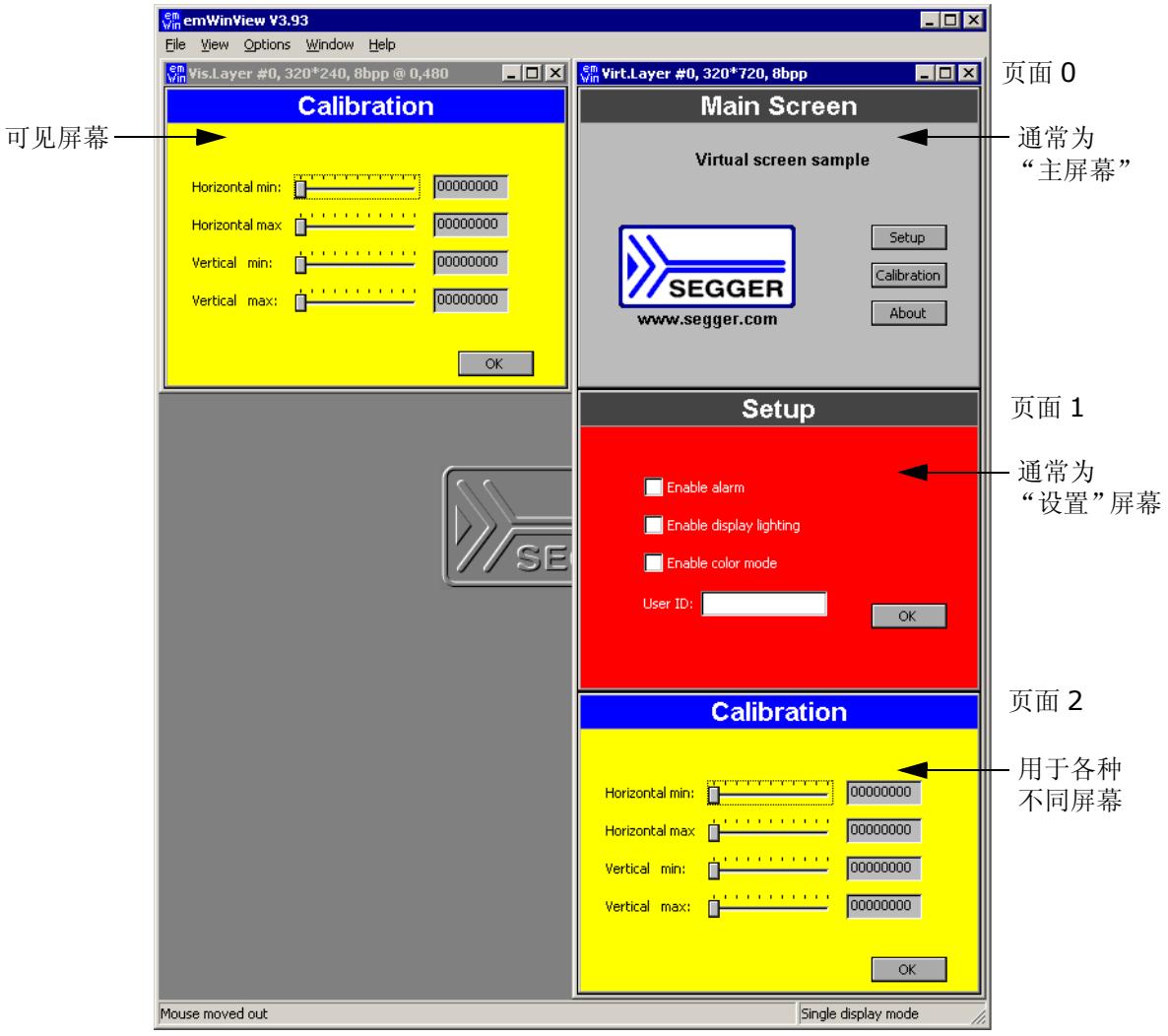
emWin 查看器解决这一问题的方式是：在独立流程中分别显示模拟的显示窗口和颜色窗口。我们可以自行决定是在调试应用程序之前还是在调试过程中启动查看器。我们建议：

- 第 1 步：启动查看器。模拟开始前，没有显示窗口或颜色窗口出现。
- 第 2 步：打开 Visual C++ 工作空间。
- 第 3 步：编译和运行应用程序。
- 第 4 步：按前述方法调试应用程序。

优点是可以在 LCD 窗口中一步步跟踪所有绘制操作。

4.1.2 使用带虚拟页面的查看器

查看器默认为每个图层打开一个窗口，显示视频 RAM （通常为屏幕）的可见部分。如果配置的虚拟视频 RAM 比屏幕大，则可使用指令 `View/Virtual Layer/Layer (0...4)` 在单个窗口中显示全部视频 RAM。如果使用函数 `GUI_SetOrg()`，则可见屏幕的内容将发生改变，但虚拟图层窗口保持不变：



更多关于虚拟屏幕的信息，请参见 “虚拟屏幕 / 虚拟页面”（第 739 页）。

4.1.3 总在顶部显示

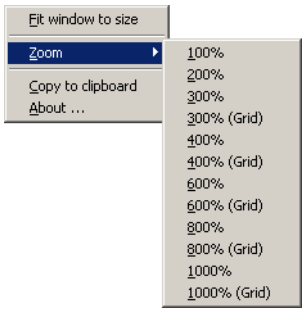
查看器窗口默认为总在顶部显示。可以从菜单中选择 `Options\Always on top` 改变这一方式。

4.1.4 打开显示输出的更多窗口

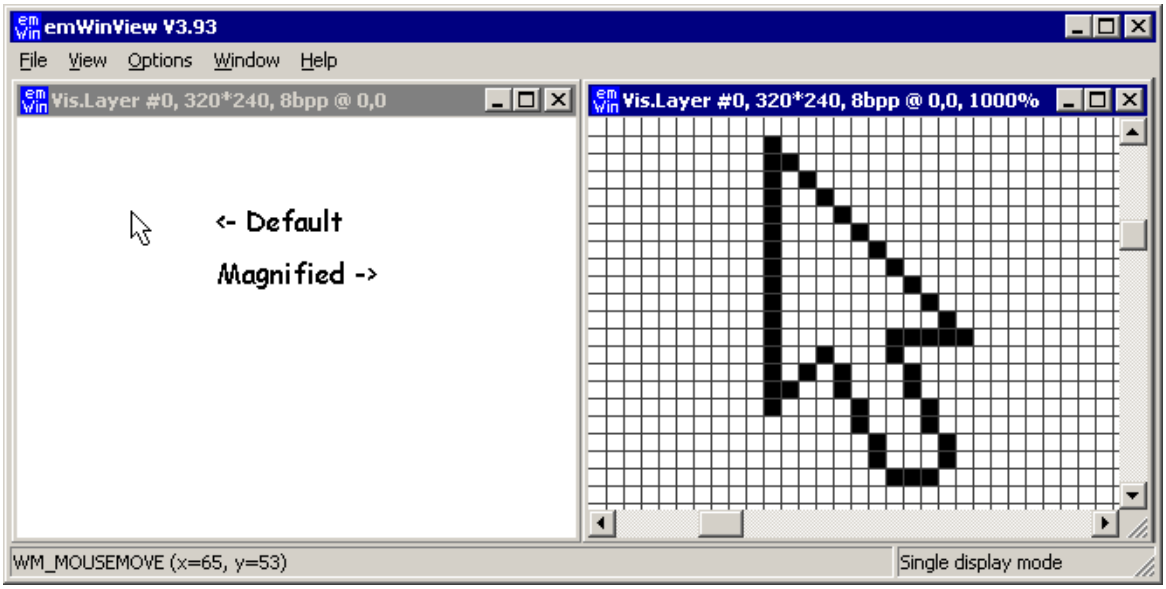
如需显示 LCD 输出的放大区域或多图层配置的复合视图，可以打开多个输出窗口。可以通过 `View/Visible Layer/Layer (1...4)`、`View/Virtual Layer/Layer (1...4)` 或 `View/Composite` 实现该功能。

4.1.5 缩放

放大或缩小很方便：
右击图层或复合窗口，打开 “缩放”弹出菜单。
选择一个缩放选项：



使用网格



如果将 LCD 输出放大 >= 300%，可以选择是否使用网格显示。可更改网格的颜色。这可以通过选择 “菜单”，单击 Options/Grid color 实现。

调节窗口大小

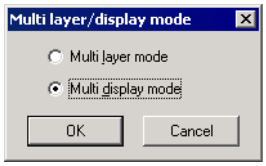
如需根据放大图像调节窗口大小，从第一个弹出菜单选择 “将窗口缩放为屏幕大小”。

4.1.6 将输出复制到剪贴板

右击 LCD 窗口或复合视图，选择 “复制到剪贴板”。例如，现在可以将剪贴板的内容粘贴到 “mspaint”应用中。

4.1.7 使用带多种显示的查看器

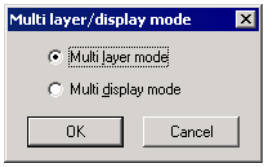
如果要使用多种显示，则可以通过使用指令 Options/Multi layer/display 将查看器设置为“多种显示模式”。



启动调试器时，查看器将为每种显示打开一个显示窗口和一个颜色窗口：

4.1.8 使用带多种显示的查看器

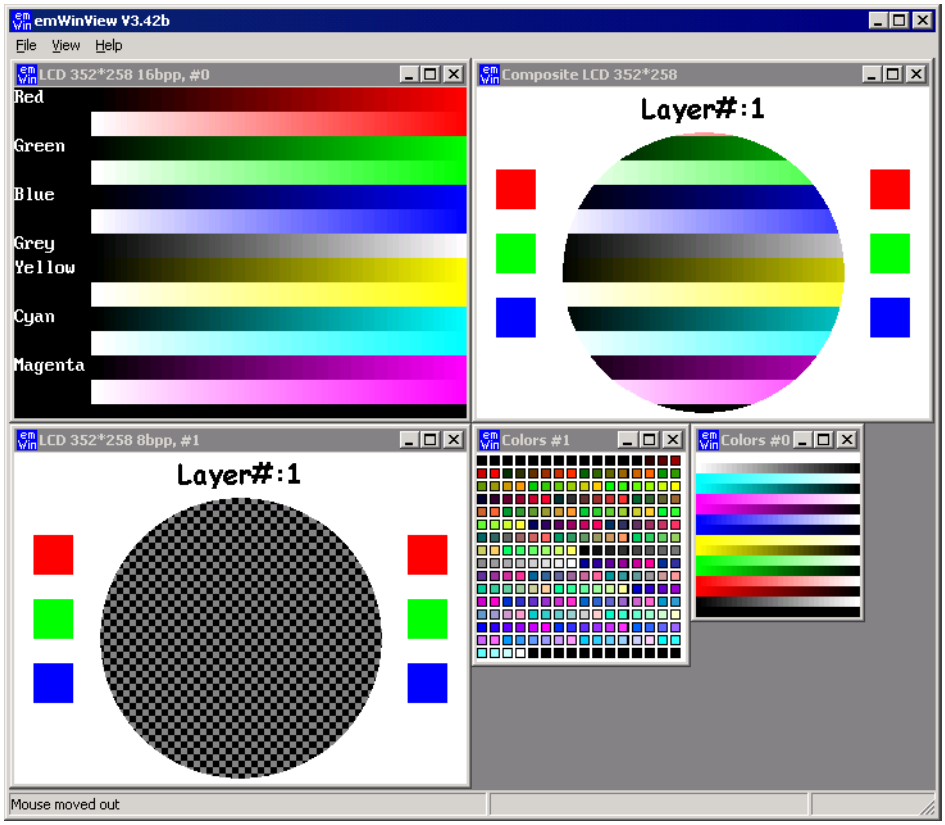
如果要使用多种显示，则可以通过使用指令 Options/Multi layer/display 将查看器设置为“多种显示模式”。



启动调试器时，查看器将为每个图层打开一个 LCD 窗口和一个颜色窗口，并打开一个复合窗口显示结果。

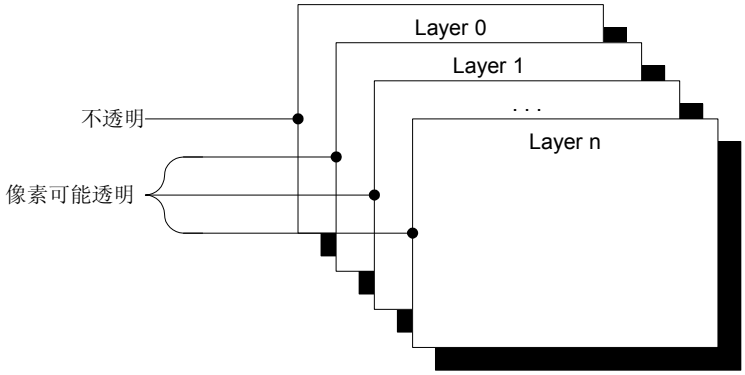
示例

以下示例为带两个图层的查看器屏幕截图。图层 0 显示的是高色彩配置的彩色条。图层 1 显示的是白色背景上的透明圆圈，带有色长方块。复合窗口显示的是屏幕上实际可见的结果



透明

查看器复合窗口显示所有图层。高索引图层位于低索引图层的上方，可以使用透明像素：



第 5 章

显示文本

使用 emWin 显示文本很容易。仅需要很少的例程知识，就可以使用任何有效的字体进行文本书写，然后显示在任一位置上。我们首先简单介绍一下文本显示，然后分别对各个可用的例程进行详细描述。

5.1 基本例程

要在 LCD 上显示文本，只需调用例程 GUI_DispString() 并以要显示的文本作为参数即可。例如：

```
GUI_DispString("Hello world!");
```

上述代码将在当前文本位置显示文本 “Hello world”。然而，正如我们将看到的，有很多例程可以使用不同字体或在特定位置显示文本。此外，它不仅可以写字符串，还可以写十进制、十六进制以及二进制数值进行显示。尽管图形显示通常以字节为向导，文本可以定位在屏幕的任何像素上，而不仅仅按字节定位。

控制字符

控制字符是指字符代码小于 32 的字符。控制字符被定义为 ASCII 代码的一部分。emWin 忽略了下表所列字符以外的所有控制字符：

字符代码	ASCII 代码	C	描述
10	LF	\n	换行。 当前文本位置改变至下一行的开始。默认为：X = 0。 Y + = 字体 - 距离 （单位：像素）（如例程 GUI_GetFontDistY() 计算得出）。
13	CR	\r	回车。 当前文本位置改变至当前行的开始。默认为：X = 0。

控制字符 LF 的用法在字符串中非常方便。换行符可以作为字符串的一部分，这样，字符串就可以拆分为多行，且只需调用一个例程就能显示。

将文本置于所选位置

这可以通过例程 GUI_GotoXY() 实现，如下面的例子所示：

```
GUI_GotoXY(10,10);//  Set text position (in pixels)
GUI_DispString("Hello world!");//  Show text
```


5.2 文本 API

下表按字母顺序列出了相应类别中与文本相关的可用例程。各例程将在后续章节中详细描述。

例程	描述
显示文本的例程	
GUI_DispChar()	在当前位置显示单个字符。
GUI_DispCharAt()	在指定位置显示单个字符。
GUI_DispChars()	按指定次数显示字符。
GUI_DispNextLine()	将光标移至下一行的开始。
GUI_DispString()	在当前位置显示字符串。
GUI_DispStringAt()	在指定位置显示字符串。
GUI_DispStringAtCEOL()	在指定位置显示字符串，并清除至行末。
GUI_DispStringHCenterAt()	在指定位置水平居中显示字符串。
GUI_DispStringInRect()	在指定的矩形区域中显示字符串。
GUI_DispStringInRectEx()	在指定的矩形区域中显示字符串，并可旋转 （可选）。
GUI_DispStringInRectWrap()	在指定的矩形区域中显示字符串，并可自动换行 （可选）。
GUI_DispStringLen()	在当前位置显示指定字符数的字符串。
GUI_WrapGetNumLines()	为指定自动换行模式获取文本行数。
选择文本绘制模式	
GUI_GetTextMode()	返回当前文本模式。
GUI_SetTextMode()	设置文本绘制模式。
GUI_SetTextStyle()	设置要使用的文本样式。
选择文本对齐模式	
GUI_GetTextAlign()	返回当前文本对齐模式。
GUI_SetLBorder()	设置换行后的左边界。
GUI_SetTextAlign()	设置文本对齐模式。
设置当前文本位置	
GUI_GotoX()	设置当前 X 位置。
GUI_GotoXY()	设置当前 (X,Y) 位置。
GUI_GotoY()	设置当前 Y 位置。
返回当前文本位置	
GUI_GetDispPosX()	返回当前 X 位置。
GUI_GetDispPosY()	返回当前 Y 位置。
清除窗口或部分窗口的例程	
GUI_Clear()	清除活动窗口 （如果背景是活动窗口，则清除整个显示）
GUI_DispCEOL()	清除从当前文本位置到行末的显示内容。

5.3 显示文本的例程

GUI_DispChar()

描述

在当前窗口的当前文本位置处，使用当前字体显示单个字符。

原型

```
void GUI_DispChar(U16 c);
```

参数	描述
c	显示的字符。

其他信息

这是显示单个字符的基本例程。所有其他显示例程 （GUI_DispCharAt()、 GUI_DispString() 等）都要调用这个例程来输出单个字符。
字符是否可用取决于所选择的字体。如果当前字体中该字符不可用，则不会有任何显示。

示例

在屏幕上显示一个大写 “A”：

```
GUI_DispChar('A');
```

相关主题

GUI_DispChars()、 GUI_DispCharAt()

GUI_DispCharAt()

描述

在当前窗口的指定位置处，使用当前字体显示单个字符。

原型

```
void GUI_DispCharAt(U16 c, I16P x, I16P y);
```

参数	描述
c	显示的字符。
x	要写入的客户端窗口 X 位置 （单位：像素）。
y	要写入的客户端窗口 Y 位置 （单位：像素）。

附加信息

所显示字符的左上角在指定的 (X,Y) 位置。
使用例程 GUI_DispChar() 写字符。
如果当前字体中该字符不可用，则不会有任何显示。

示例

在屏幕左上角显示一个大写 “A”：

```
GUI_DispCharAt('A',0,0);
```

相关主题

GUI_DispChar()、 GUI_DispChars()

GUI_DispChars()

描述

在当前窗口的当前文本位置处，使用当前字体按指定次数显示字符。

原型

```
void GUI_DispChars(U16 c, int Cnt);
```

参数	描述
c	显示的字符。
Cnt	重复次数 (0 <= Cnt <= 32767)。

其他信息

使用例程 GUI_DispChar() 写字符。
如果当前字体中该字符不可用，则不会有任何显示。

示例

在屏幕上显示 “******************************”：

```
GUI_DispChars('*', 30);
```

相关主题

GUI_DispChar()、 GUI_DispCharAt()

GUI_DispNextLine()

描述

将光标移至下一行的开始。

原型

```
void GUI_DispNextLine(void);
```

相关主题

GUI_SetLBorder()

GUI_DispString()

描述

在当前窗口的当前文本位置处，使用当前字体显示作为参数的字符串。

原型

```
void GUI_DispString(const char GUI_FAR * s);
```

参数	描述
s	显示的字符串。

其他信息

字符串可以包括控制字符 “\n”。该控制字符把当前文本位置移至下一行的开始。

示例

在屏幕上显示 “Hello world” 并在下一行显示 “Next line”：

```
GUI_DispString("Hello world");  //Disp text
GUI_DispString("\nNext line");  //Disp text
```

相关主题

GUI_DispStringAt()、 GUI_DispStringAtCEOL()、 GUI_DispStringLen()

GUI_DispStringAt()

描述

在当前窗口的指定位置处，使用当前字体显示作为参数的字符串。

原型

```
void GUI_DispStringAt(const char GUI_FAR * s, int x, int y);
```

参数	描述
s	显示的字符串。
x	要写入的客户端窗口 X 位置 （单位：像素）。
y	要写入的客户端窗口 Y 位置 （单位：像素）。

示例

在屏幕位置 (50,20) 处显示 “Position 50,20”：

```
GUI_DispStringAt("Position 50,20", 50, 20);  // Disp text
```

相关主题

GUI_DispString()、 GUI_DispStringAtCEOL()、 GUI_DispStringLen()

GUI_DispStringAtCEOL()

描述

该例程使用的参数与 GUI_DispStringAt() 完全相同。它也执行同样的操作：在指定的位置显示所给出的字符串。但是，完成此操作后，它会调用 GUI_DispCEOL() 例程清除本行剩下部分内容直至行末。如果某个字符串要覆盖其他字符串，同时该字符串长度比原先的字符串要短，则使用该例程会很方便。

GUI_DispStringHCenterAt()

描述

在当前窗口的指定位置处，使用当前字体水平居中显示作为参数的字符串。

原型

```
void GUI_DispStringHCenterAt(const char GUI_FAR * s, int x, int y);
```

参数	描述
s	显示的字符串。
x	要写入的客户端窗口 X 位置 （单位：像素）。
y	要写入的客户端窗口 Y 位置 （单位：像素）。

GUI_DispStringInRect()

描述

在当前窗口指定的矩形区域内的指定位置处，使用当前字体显示作为参数的字符串。

原型

```
void GUI_DispStringInRect(const char GUI_FAR * s,
                          GUI_RECT *           pRect,
                          int                  Align);
```

参数	描述
s	显示的字符串。
pRect	要写入的客户端窗口矩形区域 （单位：像素）。
Align	对齐标记，可以通过 “OR” 操作进行组合。垂直对齐和水平对齐标记应该组合使用。可用的标记有： 垂直对齐：GUI_TA_TOP、 GUI_TA_BOTTOM、 GUI_TA_VCENTER。 水平对齐：GUI_TA_LEFT、 GUI_TA_RIGHT、 GUI_TA_HCENTER。

示例

在当前窗口的水平和垂直居中位置显示字 “Text”：

```
GUI_RECT rClient;
GUI_GetClientRect(&rClient);
GUI_DispStringInRect("Text", &rClient, GUI_TA_HCENTER | GUI_TA_VCENTER);
```

其他信息

如果指定的矩形区域太小，文本会被裁剪。

相关主题

GUI_DispString()、 GUI_DispStringAtCEOL()、 GUI_DispStringLen()

GUI_DispStringInRectEx()

描述

在当前窗口指定的矩形区域内的指定位置处，使用当前字体显示作为参数的字符串，并可旋转（可选）。

原型

```
void GUI_DispStringInRectEx(const char *          s,
                            GUI_RECT *            pRect,
                            int                   TextAlign,
                            int                   MaxLen,
                            const GUI_ROTATION *  pLCD_Api);
```

参数	描述
s	显示的字符串。
pRect	要写入的客户端窗口矩形区域 （单位：像素）。
TextAlign	对齐标记，可以通过 “OR” 操作进行组合。垂直对齐和水平对齐标记应该组合使用。可用的标记有： 垂直对齐：GUI_TA_TOP、GUI_TA_BOTTOM、GUI_TA_VCENTER。 水平对齐：GUI_TA_LEFT、GUI_TA_RIGHT、GUI_TA_HCENTER。
MaxLen	显示的最大字符数。
pLCD_Api	（参见下表）

参数 pLCD_Api 的允许值	
GUI_ROTATE_0	不旋转文本。从左到右显示。
GUI_ROTATE_180	180 度旋转文本。
GUI_ROTATE_CCW	逆时针旋转文本。
GUI_ROTATE_CW	顺时针旋转文本。

示例

在指定的矩形区域内水平和垂直居中处显示字 “Text”：

```
GUI_RECT Rect = {10, 10, 40, 80};
char acText[] = "Rotated\ntext";
GUI_SetTextMode(GUI_TM_XOR);
GUI_FillRectEx(&Rect);
GUI_DispStringInRectEx(acText,
                       &Rect,
                       GUI_TA_HCENTER | GUI_TA_VCENTER,
                       strlen(acText),
                       GUI_ROTATE_CCW);
```

上述示例的屏幕截图

其他信息

如果指定的矩形区域太小，文本会被裁剪。
为了让例程有效，配置开关 GUI_SUPPORT_ROTATION 必须激活 （默认）。

GUI_DispStringInRectWrap()

描述

在当前窗口指定的矩形区域内的指定位置处，使用当前字体显示字符串，并可自动换行 （可选）。

原型

```
void GUI_DispStringInRectWrap(const char GUI_UNI_PTR * s,
                              GUI_RECT *                pRect,
                              int                       TextAlign,
                              GUI_WRAPMODE              WrapMode);
```

参数	描述
s	显示的字符串。
pRect	要写入的客户端窗口矩形区域 （单位：像素）。
TextAlign	对齐标记，可以通过 “OR” 操作进行组合。垂直对齐和水平对齐标记应该组合使用。可用的标记有： 垂直对齐：GUI_TA_TOP、 GUI_TA_BOTTOM、 GUI_TA_VCENTER。 水平对齐：GUI_TA_LEFT、 GUI_TA_RIGHT、 GUI_TA_HCENTER。
WrapMode	（参见下表）

参数 WrapMode 的允许值	
GUI_WRAPMODE_NONE	不执行自动换行。
GUI_WRAPMODE_WORD	根据字对文本进行自动换行。
GUI_WRAPMODE_CHAR	根据字符对文本进行自动换行。

其他信息

如需执行字换行，而指定的矩形区域又太小，则会对该字进行字符换行。

示例

在指定的矩形区域内水平和垂直居中处显示字 “Text”，并进行字换行：

```
int i;
char acText[]      = "This example demonstrates text wrapping";
GUI_RECT Rect      = {10, 10, 59, 59};
GUI_WRAPMODE aWm[] = {GUI_WRAPMODE_NONE,
                      GUI_WRAPMODE_CHAR,
                      GUI_WRAPMODE_WORD};
GUI_SetTextMode(GUI_TM_TRANS);
for (i = 0; i < 3; i++) {
  GUI_SetColor(GUI_BLUE);
  GUI_FillRectEx(&Rect);
  GUI_SetColor(GUI_WHITE);
  GUI_DispStringInRectWrap(acText, &Rect, GUI_TA_LEFT, aWm[i]);
  Rect.x0 += 60;
  Rect.x1 += 60;
}
```

上述示例的屏幕截图

GUI_DispStringLen()

描述

在当前窗口的当前文本位置处，使用当前字体按指定字符数显示作为参数的字符串。

原型

```
void GUI_DispStringLen(const char GUI_FAR * s, int Len);
```

参数	描述
s	显示的字符串。应该以 “\0” 作为 8 位字符数组的结束标记。允许用 NULL 作为参数。
Len	显示的字符数。

其他信息

如果字符串的字符数少于指定的数量 （短些），则用空格填满。
如果多于指定的数量 （长些），则实际仅会显示指定数量的字符。
文本消息可能以不同语言显示 （长度自然不同）时，该函数尤为有用，但仅有一定数量的字符可以显示。

相关主题

GUI_DispString()、 GUI_DispStringAt()、 GUI_DispStringAtCEOL()

GUI_WrapGetNumLines()

描述

返回设置自动换行模式的指定文本的行数。

原型

```
int GUI_WrapGetNumLines(const char GUI_UNI_PTR * pText,
                        int                      xSize,
                        GUI_WRAPMODE             WrapMode);
```

参数	描述
pText	显示的字符串。应该以 “\0” 作为 8 位字符数组的结束标记。允许用 NULL 作为参数。
xSize	X-size 用以绘制文本。
WrapMode	（参见下表）

参数 WrapMode 的允许值	
GUI_WRAPMODE_NONE	不执行自动换行。
GUI_WRAPMODE_WORD	根据字对文本进行自动换行。
GUI_WRAPMODE_CHAR	根据字符对文本进行自动换行。

其他信息

请记住，绘制文本所需的行数取决于当前所选的字体。

5.4 选择文本绘制模式

通常，在所选窗口的当前文本位置，使用所选字体以正常文本模式写入文本。正常文本模式意思是指文本覆盖已经显示的任何内容，在这种情况下，字符掩码中设定的位元在屏幕上被设定。在这种模式下，活动位元使用前景颜色写入，而非活动位元用背景颜色写入。但在某些情况下，需要更改这种默认模式。 emWin 为此提供了四种可以组合使用的标记 （一种默认加三种修改标记）：

正常文本

通过指定 GUI_TEXTMODE_NORMAL 或 0，可以正常显示文本。

反转文本

通过指定GUI_TEXTMODE_REV，可以反转显示文本。通常的黑底白字显示方式将变为白底黑字显示。

透明文本

通过指定 GUI_TEXTMODE_TRANS，可以显示为透明文本。透明文本表示文本写在屏幕上已经可见的内容之上。不同之处在于，屏幕上原有的内容仍然可见，而在正常文本中，背景会替换为当前选择的背景色。

异或文本

通过指定 GUI_TEXTMODE_XOR，可以使用异或模式显示文本。通常情况下，用白色绘制的 （实际字符）显示是反相的。如果背景色是黑色，效果与默认模式（正常文本）是一样的。如果背景是白色，输出与反转文本一样。如果使用彩色，反相的像素由下式计算：
新像素颜色 = 颜色的值 – 实际像素颜色 –1。

透明反转文本

通过指定 GUI_TEXTMODE_TRANS | GUI_TEXTMODE_REV，可以显示为透明反转文本。与透明文本一样，它不会覆盖背景，而且和反转文本一样，该文本会反转显示。

其他信息

请注意，还可以使用缩写形式：例如， GUI_TM_NORMAL

示例

显示正常、反转、透明、异或以及透明反转文本：

```
GUI_SetFont(&GUI_Font8x16);
GUI_SetBkColor(GUI_BLUE);
GUI_Clear();
GUI_SetPenSize(10);
GUI_SetColor(GUI_RED);
GUI_DrawLine(80, 10, 240, 90);
GUI_DrawLine(80, 90, 240, 10);
GUI_SetBkColor(GUI_BLACK);
GUI_SetColor(GUI_WHITE);
GUI_SetTextMode(GUI_TM_NORMAL);
GUI_DispStringHCenterAt("GUI_TM_NORMAL"            , 160, 10);
GUI_SetTextMode(GUI_TM_REV);
GUI_DispStringHCenterAt("GUI_TM_REV"               , 160, 26);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DispStringHCenterAt("GUI_TM_TRANS"             , 160, 42);
GUI_SetTextMode(GUI_TM_XOR);
GUI_DispStringHCenterAt("GUI_TM_XOR"               , 160, 58);
GUI_SetTextMode(GUI_TM_TRANS | GUI_TM_REV);
GUI_DispStringHCenterAt("GUI_TM_TRANS | GUI_TM_REV", 160, 74);
```

上述示例的屏幕截图

GUI_GetTextMode()

描述

返回当前选定的文本模式。

原型

```
int GUI_GetTextMode(void);
```

返回值

当前选定的文本模式。

GUI_SetTextMode()

描述

按照指定的参数设置文本模式。

原型

```
int GUI_SetTextMode(int TextMode);
```

参数	描述
TextMode	设置的文本模式，可以是 TEXTMODE 标记的任意组合。

参数 TextMode 的允许值 （可以通过 “OR”操作进行组合）	
GUI_TEXTMODE_NORMAL	设置为显示正常文本。这是默认设置，该数值等同于 0。
GUI_TEXTMODE_REV	设置为显示反转文本。
GUI_TEXTMODE_TRANS	设置为显示透明文本。
GUI_TEXTMODE_XOR	设置为反相显示的文本。

返回值

之前选定的文本模式。

示例

屏幕位置 (0,0) 处显示 “The value is”，以反转文本显示一个值，再将其设回正常模式：

```
int i = 20;
GUI_DispStringAt("The value is", 0, 0);
GUI_SetTextMode(GUI_TEXTMODE_REV);
GUI_DispDec(20, 3);
GUI_SetTextMode(GUI_TEXTMODE_NORMAL);
```


GUI_SetTextStyle()

描述

按照指定的参数设置文本样式。

原型

```
char GUI_SetTextStyle(char Style);
```

参数	描述
Style	设置的文本样式 （参见下表）。

参数 Style 的允许值	
GUI_TS_NORMAL	显示正常文本 （默认）。
GUI_TS_UNDERLINE	显示带下划线的文本。
GUI_TS_STRIKETHRU	显示带删除线的文本。
GUI_TS_OVERLINE	显示带顶线的文本。

返回值

之前选定的文本样式。

5.5 选择文本对齐模式

GUI_GetTextAlign()

描述

返回当前文本对齐模式。

原型

```
int GUI_GetTextAlign(void);
```

GUI_SetLBorder()

描述

设置当前窗口换行后的左边界。

原型

```
void GUI_SetLBorder(int x)
```

参数	描述
x	新的左边界 （单位：像素， 0 表示左边界）。

GUI_SetTextAlign()

描述

为当前窗口的字符串输出设置文本对齐模式。

原型

```
int GUI_SetTextAlign(int TextAlign);
```

参数	描述
TextAlign	设置的文本对齐模式，可以是水平和垂直对齐标记的组合。

参数 TextAlign 的允许值
（水平和垂直标记可以通过 “OR”操作进行组合）

水平对齐	
GUI_TA_LEFT	X 轴方向左对齐 （默认）。
GUI_TA_HCENTER	X 轴方向居中。
GUI_TA_RIGHT	X 轴方向右对齐。
垂直对齐	
GUI_TA_TOP	在字符 Y 轴方向顶部对齐 （默认）。
GUI_TA_VCENTER	Y 轴方向居中。
GUI_TA_BOTTOM	在字体 Y 轴底部像素线对齐。

返回值

所选的文本对齐模式。

其他信息

GUI_SetTextAllign() 不影响以 GUI_DispChar() 开始的字符输出例程。请注意，该例程的设置仅对单一字符串有效。

示例

在位置 (100,100) 处显示数值 1234，采用居中模式：

```
GUI_SetTextAlign(GUI_TA_HCENTER | GUI_TA_VCENTER);
GUI_DispDecAt(1234,100,100,4);
```

5.6 设置当前文本位置

每个任务都有一个当前文本位置。该位置以窗口的原点（通常是 (0,0)）为参考，如果调用文本输出例程，下一个字符就会写在该位置上。最初，该位置是 (0,0)，即当前窗口的左上角。有三个函数可以用于设置当前文本位置。

GUI_GotoXY(), GUI_GotoX(), GUI_GotoY()

描述

设置当前文本写入位置。

原型

```
char GUI_GotoXY(int x, int y);
char GUI_GotoX(int x);
char GUI_GotoY(int y);
```

参数	描述
x	新的 X 轴位置 （单位：像素， 0 表示左边界）。
y	新的 Y 轴位置 （单位：像素， 0 表示顶部边界）。

返回值

通常为 0。
如果返回值 != 0，则当前文本位置超出窗口范围（到了右边或下方），这样后续的写入操作可能被忽略。

其他信息

GUI_GotoXY() 对当前文本位置的 X 和 Y 位置同时设置。
GUI_GotoX() 仅对当前文本位置的 X 位置进行设置， Y 位置保持不变。
GUI_GotoY() 仅对当前文本位置的 X 位置进行设置， X 位置保持不变。

示例

在屏幕位置 (20,20) 处显示 “(20,20)”：

```
GUI_GotoXY(20,20)
GUI_DispString("The value is");
```

5.7 返回当前文本位置

GUI_GetDispPosX()

描述

返回当前 X 位置。

原型

```
int GUI_GetDispPosX(void);
```

GUI_GetDispPosY()

描述

返回当前 Y 位置。

原型

```
int GUI_GetDispPosY(void);
```

5.8 清除窗口或部分窗口的例程

GUI_Clear()

描述

清除当前窗口。

原型

```
void GUI_Clear(void);
```

其他信息

如果没有定义窗口，则当前窗口为整个显示区。在这种情况下，整个显示区都会被清除。

示例

在屏幕上显示 “Hello world”，等待 1 秒种，然后清除显示内容：

```
GUI_DispStringAt("Hello world", 0, 0); // Disp text
GUI_Delay(1000);                       // Wait 1 second (not part of emWin)
GUI_Clear();                           // Clear screen
```

GUI_DispCEOL()

描述

清除当前窗口 （或显示）从当前文本位置到行末的内容，高度为当前字体高度。

原型

```
void GUI_DispCEOL(void);
```

示例

在屏幕上显示“Hello world”，等待 1 秒钟，然后在同一位置处显示“Hi”，代替原先显示的字符串：

```
GUI_DispStringAt("Hello world", 0, 0);// Disp text
Delay (1000);
GUI_DispStringAt("Hi", 0, 0);
GUI_DispCEOL();
```


第 6 章

显示数值

上一章节描述了如何在屏幕上显示字符串。当然，我们也可以使用字符串和标准 C 库的函数来显示数值。然而，有时候这会是件困难的事。通常，较为容易（也更为有效）的是调用一个例程显示所需结构的数值。 emWin 支持各种十进制、十六进制和二进制输出。本章将对这些例程进行逐一描述。所有函数不需要使用浮点库，并对速度和大小进行了优化。当然“Sprintf”可以用于任何系统。使用本章介绍的例程，有时可以简化操作，节省 ROM 空间和执行时间。

6.1 评估 API

下表按字母顺序列出了相应类别中与数值相关的可用例程。各例程将在后续章节中详细描述。

例程	描述
显示十进制数值	
GUI_DispDec()	在当前位置显示指定字符数的十进制数值。
GUI_DispDecAt()	在指定位置显示指定字符数的十进制数值。
GUI_DispDecMin()	在当前位置显示最小字符数的十进制数值。
GUI_DispDecShift()	在当前位置显示指定字符数、带小数点的十进制长数值。
GUI_DispDecSpace()	在当前位置显示指定字符数的十进制数值，用空格代替首位的 0。
GUI_DispSDec()	在当前位置显示指定字符数的十进制数值并显示符号。
GUI_DispSDecShift()	在当前位置显示指定字符数、带小数点的十进制长数值并显示符号。
显示浮点数值	
GUI_DispFloat()	显示指定字符数的浮点数值。
GUI_DispFloatFix()	显示指定小数点右边位数的浮点数值。
GUI_DispFloatMin()	显示最小字符数的浮点数值。
GUI_DispSFloatFix()	显示指定小数点右边位数的浮点数值并显示符号。
GUI_DispSFloatMin()	显示最小字符数的浮点数值并显示符号。
显示二进制数值	
GUI_DispBin()	在当前位置显示二进制数值。
GUI_DispBinAt()	在指定位置显示二进制数值。
显示十六进制数值	
GUI_DispHex()	在当前位置显示十六进制数值。
GUI_DispHexAt()	在指定位置显示十六进制数值。
emWin 版	
GUI_GetVersionString()	返回 emWin 的当前版本。

6.2 显示十进制数值

GUI_DispDec()

描述

在当前窗口的当前文本位置处，使用当前字体显示指定字符数的十进制数值。

原型

```
void GUI_DispDec(I32 v, U8 Len);
```

参数	描述
V	显示的数值。 最小值为 -2147483648 (= -2^31)。 最大值为 2147483647 (= 2^31 -1)。
Len	显示的位数 （最大为 10）。

其他信息

支持首位为 0 的格式 （显示为 0）。
如果数值为负，则会显示一个负号。

示例

```
// Display time as minutes and seconds
  GUI_DispString("Min:");
  GUI_DispDec(Min,2);
  GUI_DispString(" Sec:");
  GUI_DispDec(Sec,2);
```

相关主题

GUI_DispSDec()、 GUI_DispDecAt()、 GUI_DispDecMin()、 GUI_DispDecSpace()

GUI_DispDecAt()

描述

在当前窗口的指定位置处，使用当前字体显示指定字符数的十进制数值。

原型

```
void GUI_DispDecAt(I32 v, I16P x, I16P y, U8 Len);
```

参数	描述
V	显示的数值。 最小值为 -2147483648 (= -2^31)。 最大值为 2147483647 (= 2^31 -1)。
x	要写入的客户端窗口 X 位置 （单位：像素）。
y	要写入的客户端窗口 Y 位置 （单位：像素）。
Len	显示的位数 （最大为 10）。

其他信息

支持首位为 0 的格式。
如果数值为负，则会显示一个负号。

示例

```
// Update seconds in upper right corner
GUI_DispDecAT(Sec, 200, 0, 2);
```

相关主题

GUI_DispDec()、 GUI_DispSDec()、 GUI_DispDecMin()、 GUI_DispDecSpace()

GUI_DispDecMin()

描述

在当前窗口的当前文本位置处，使用当前字体显示十进制数值。不需要指定数值长度。会自动使用最小长度值。

原型

```
void GUI_DispDecMin(I32 v);
```

参数	描述
V	显示的数值。 最小值：-2147483648 (= -2^31)，最大值为 2147483647 (= 2^31 -1)。

其他信息

显示的最大位数为 10。如果数值必须对齐，但位数不一样，则不应该使用该函数。在这种情况下，应该使用一个需要指定位数的函数。

示例

```
// Show result
GUI_DispString("The result is :");
GUI_DispDecMin(Result);
```

相关主题

GUI_DispDec()、 GUI_DispDecAt()、 GUI_DispSDec()、 GUI_DispDecSpace()

GUI_DispDecShift()

描述

在当前窗口的当前文本位置处，使用当前字体显示指定字符数的十进制长数值 （带小数点）。

原型

```
void GUI_DispDecShift(I32 v, U8 Len, U8 Shift);
```

参数	描述
V	显示的数值。 最小值：-2147483648 (= -2^31)，最大值：2147483647 (= 2^31 -1)。
Len	显示的位数 （最大为 10）。
Shift	小数点右边显示的位数。

其他信息

注意：显示的最大字符数为 9 （包括符号和小数点）。

GUI_DispDecSpace()

描述

在当前窗口的当前文本位置处，使用当前字体显示十进制数值。不支持首位为 0 的格式（用空格替代）。

原型

```
void DispDecSpace(I32 v, U8 MaxDigits);
```

参数	描述
V	显示的数值。 最小值：-2147483648 (= -2^31)，最大值：2147483647 (= 2^31 -1)。
MaxDigits	显示的位数，包括首位空格。 显示的最大位数为 10 （不包括首位空格）。

其他信息

如果数值必须对齐，但数字位数不一样，使用该函数是一个不错的选择。

示例

```
// Show result
GUI_DispString("The result is :");
GUI_DispDecSpace(Result, 200);
```

相关主题

GUI_DispDec()、 GUI_DispDecAt()、 GUI_DispSDec()、 GUI_DispDecMin()

GUI_DispSDec()

描述

在当前窗口的当前文本位置处，使用当前字体显示指定字符数的十进制数值 （带符号）。

原型

```
void GUI_DispSDec(I32 v, U8 Len);
```

参数	描述
V	显示的数值。 最小值：-2147483648 (= -2^31)，最大值：2147483647 (= 2^31 -1)。
Len	显示的位数 （最大为 10）。

其他信息

支持首位为 0 的格式。
该函数与 GUI_DispDec 相似，但在数值的前面总显示符号，即使这个数值为正。

相关主题

GUI_DispDec()、 GUI_DispDecAt()、 GUI_DispDecMin()、 GUI_DispDecSpace()

GUI_DispSDecShift()

描述

在当前窗口的当前文本位置处，使用当前字体显示指定字符数的十进制长数值 （带符号和小数点）。

原型

```
void GUI_DispSDecShift(I32 v, U8 Len, U8 Shift);
```

参数	描述
V	显示的数值。 最小值：-2147483648 (= -2^31)，最大值：2147483647 (= 2^31 -1)。
Len	显示的位数 （最大为 10）。
Shift	小数点右边显示的位数。

其他信息

数值前面总显示符号。
注意：显示的最大字符数为 9 （包括符号和小数点）。

示例

```
void DemoDec(void) {
  long l = 12345;
  GUI_Clear();
  GUI_SetFont(&GUI_Font8x8);
  GUI_DispStringAt("GUI_DispDecShift:\n",0,0);
  GUI_DispSDecShift(l, 7, 3);
  GUI_SetFont(&GUI_Font6x8);
  GUI_DispStringAt("Press any key",0,GUI_VYSIZE-8);
  WaitKey();
}
```

上述示例的屏幕截图

6.3 显示浮点数值

GUI_DispFloat()

描述

在当前窗口的当前文本位置处，使用当前字体显示指定字符数的浮点数值。

原型

```
void GUI_DispFloat(float v, char Len);
```

参数	描述
V	显示的数值。 最小值为 1.2 E-38，最大值为 3.4 E38。
Len	显示的位数 （最大为 10）。

其他信息

不支持首位为 0 的格式。小数点作为一个字符处理。
如果数值为负，则会显示一个负号。

示例

```
/* Shows all features for displaying floating point values */
void DemoFloat(void) {
  float f = 123.45678;
  GUI_Clear();
  GUI_SetFont(&GUI_Font8x8);
  GUI_DispStringAt("GUI_DispFloat:\n",0,0);
  GUI_DispFloat (f,9);
  GUI_GotoX(100);
  GUI_DispFloat (-f,9);
  GUI_DispStringAt("GUI_DispFloatFix:\n",0,20);
  GUI_DispFloatFix (f,9,2);
  GUI_GotoX(100);
  GUI_DispFloatFix (f,9,2);
  GUI_DispStringAt("GUI_DispSFloatFix:\n",0,40);
  GUI_DispSFloatFix (f,9,2);
  GUI_GotoX(100);
  GUI_DispSFloatFix (-f,9,2);
  GUI_DispStringAt("GUI_DispFloatMin:\n",0,60);
  GUI_DispFloatMin (f,3);
  GUI_GotoX(100);
  GUI_DispFloatMin (-f,3);
  GUI_DispStringAt("GUI_DispSFloatMin:\n",0,80);
  GUI_DispSFloatMin (f,3);
  GUI_GotoX(100);
  GUI_DispSFloatMin (-f,3);
  GUI_SetFont(&GUI_Font6x8);
  GUI_DispStringAt("Press any key",0,GUI_YSIZE-8);
  WaitKey();
}
```

上述示例的屏幕截图

GUI_DispFloatFix()

描述

在当前窗口的当前文本位置处，使用当前字体显示指定总字符数和小数点右边字符数的浮点数值。

原型

```
void GUI_DispFloatFix (float v, char Len, char Decs);
```

参数	描述
V	显示的数值。 最小值为 1.2 E-38，最大值为 3.4 E38。
Len	显示的位数 （最大为 10）。
Decs	小数点右边显示的位数。

其他信息

支持首位为 0 的格式。
如果数值为负，则会显示一个负号。

GUI_DispFloatMin()

描述

在当前窗口的当前文本位置处，使用当前字体显示小数点右边十进制数最少的浮点数值。

原型

```
void GUI_DispFloatMin(float f, char Fract);
```

参数	描述
V	显示的数值。 最小值为 1.2 E-38，最大值为 3.4 E38。
Fract	显示最少的字符数。

其他信息

不支持首位为 0 的格式。
如果数值为负，则会显示一个负号。
不需要指定数值长度，会自动选择最小的长度值。如果数值必须对齐，但数字位数不一样，使用该函数不是一个好的选择。可以尝试一个指定数字位数的函数。

GUI_DispSFloatFix()

描述

在当前窗口，使用当前字体显示指定总字符数和小数点右边字符数的浮点数值 （带符号）。

原型

```
void GUI_DispSFloatFix(float v, char Len, char Decs);
```

参数	描述
V	显示的数值。 最小值为 1.2 E-38，最大值为 3.4 E38。
Len	显示的位数 （最大为 10）。
Decs	小数点右边显示的位数。

其他信息

支持首位为 0 的格式。
数值前面总显示符号。

GUI_DispSFloatMin()

描述

在当前窗口的当前文本位置处，使用当前字体显示小数点右边十进制数最少的浮点数值 （带符号）。

原型

```
void GUI_DispSFloatMin(float f, char Fract);
```

参数	描述
V	显示的数值。 最小值为 1.2 E-38，最大值为 3.4 E38。
Fract	显示最少的位数。

其他信息

不支持首位为 0 的格式。
数值前面总显示符号。
不需要指定数值长度，会自动选择最小的长度值。如果数值必须对齐，但数字位数不一样，使用该函数不是一个好的选择。可以尝试一个指定数字位数的函数。

6.4 显示二进制数值

GUI_DispBin()

描述

在当前窗口的当前文本位置处，使用当前字体显示二进制数值。

原型

```
void GUI_DispBin(U32 v, U8 Len);
```

参数	描述
V	显示的数值， 32 位。
Len	显示的数字位数 （包括首位的 0）。

其他信息

与十进制和十六进制一样，最低有效位在最右边。

示例

```
//
// Show binary value 7, result:000111
//
  U32 Input = 0x7;
  GUI_DispBin(Input, 6);
```

相关主题

GUI_DispBinAt()

GUI_DispBinAt()

描述

在当前窗口的指定位置处，使用当前字体显示二进制数值。

原型

```
void DispBinAt(U32  v, I16P y, I16P x, U8 Len);
```

参数	描述
V	显示的数值， 16 位。
x	要写入的客户端窗口 X 位置 （单位：像素）。
y	要写入的客户端窗口 Y 位置 （单位：像素）。
Len	显示的数字位数 （包括首位的 0）。

其他信息

与十进制和十六进制一样，最低有效位在最右边。

示例

```
//
// Show binary input status
//
  GUI_DispBinAt(Input, 0,0, 8);
```

相关主题

GUI_DispBin(), GUI_DispHex()

6.5 显示十六进制数值

GUI_DispHex()

描述

在当前窗口的当前文本位置处，使用当前字体显示十六进制数值。

原型

```
void GUI_DispHex(U32 v, U8 Len);
```

参数	描述
V	显示的数值， 16 位。
Len	显示的数字位数。

其他信息

与十进制和二进制一样，最低有效位在最右边。

示例

```
/* Show value of AD-converter */
GUI_DispHex(Input, 4);
```

相关主题

GUI_DispDec()、 GUI_DispBin()、 GUI_DispHexAt()

GUI_DispHexAt()

描述

在当前窗口的指定位置处，使用当前字体显示十六进制数值。

原型

```
void GUI_DispHexAt(U32 v, I16P x, I16P y, U8 Len);
```

参数	描述
V	显示的数值， 16 位。
x	要写入的客户端窗口 X 位置 （单位：像素）。
y	要写入的客户端窗口 Y 位置 （单位：像素）。
Len	显示的数字位数。

其他信息

与十进制和二进制一样，最低有效位在最右边。

示例

```
//
// Show value of AD-converter at specified position
//
GUI_DispHexAt(Input, 0, 0, 4);
```

相关主题

GUI_DispDec()、 GUI_DispBin()、 GUI_DispHex()

6.6 emWin 版

GUI_GetVersionString()

描述

返回一个包含 emWin 当前版本的字符串。

原型

```
const char * GUI_GetVersionString(void);
```

示例

```
//
// Displays the current version at the current cursor position
//
GUI_DispString(GUI_GetVersionString());
```

第 7 章

2-D 图形库

emWin 包含完整的 2-D 图形库，可供大多数应用程序充分使用。 emWin 提供的例程的使用具有（或没有）裁剪 （请参阅 “窗口管理器 (WM)”（第 289 页）），它们基于快速高效的算法。当前，只有 GUI_DrawArc() 函数需要浮点计算。

7.1 图形 API

下表按字母顺序列出了各自类别内可用的与图形相关的例程。有关详细描述，请参阅后面的章节。

例程	描述
GUI_GetPixelIndex()	返回给定位置的颜色指数。
绘制模式	
GUI_GetDrawMode()	返回当前的绘制模式。
GUI_SetDrawMode()	设置绘制模式。
画笔大小	
GUI_GetPenSize()	返回当前的画笔大小 （像素）。
GUI_SetPenSize()	设置画笔大小 （像素）。
查询当前客户区矩形	
GUI_GetClientRect()	返回当前可用的绘制区域。
基本绘制例程	
GUI_ClearRect()	为矩形区域填充背景颜色。
GUI_CopyRect()	复制显示器中的一个矩形区域。
GUI_DrawGradientH()	绘制用水平颜色梯度填充的矩形。
GUI_DrawGradientV()	绘制用垂直颜色梯度填充的矩形。
GUI_DrawGradientRoundedH()	绘制用水平颜色梯度填充的圆角矩形。
GUI_DrawGradientRoundedV()	绘制用垂直颜色梯度填充的圆角矩形。
GUI_DrawPixel()	绘制单个像素。
GUI_DrawPoint()	绘制点。
GUI_DrawRect()	绘制矩形。
GUI_DrawRectEx()	绘制矩形。
GUI_DrawRoundedFrame()	绘制圆角框。
GUI_DrawRoundedRect()	绘制圆角矩形。
GUI_FillRect()	绘制填充的矩形。
GUI_FillRectEx()	绘制填充的矩形。
GUI_FillRoundedRect()	绘制填充的圆角矩形。
GUI_InvertRect()	倒转矩形区域。
Alpha 混合	
GUI_EnableAlpha()	启用 / 禁用自动 Alpha 混合。
GUI_RestoreUserAlpha()	恢复之前的用户 Alpha 混合状态。
GUI_SetAlpha()	设置当前的 Alpha 混合值。（弃用）。
GUI_SetUserAlpha()	设置其他值，以用于计算要使用的实际 Alpha 混合值。
绘制位图	
GUI_DrawBitmap()	绘制位图。
GUI_DrawBitmapEx()	绘制缩放的位图。
GUI_DrawBitmapHWAlpha()	在具有硬件 Alpha 混合支持的系统上绘制带 Alpha 混合信息的位图。
GUI_DrawBitmapMag()	绘制放大的位图。
绘制流位图	
GUI_CreateBitmapFromStream()	基于给定的任何类型的流创建位图。
GUI_CreateBitmapFromStreamIDX()	根据基于指数的位图流创建位图。
GUI_CreateBitmapFromStreamRLE4()	基于 RLE4 位图流创建位图。
GUI_CreateBitmapFromStreamRLE8()	基于 RLE8 位图流创建位图。
GUI_CreateBitmapFromStream565()	基于 16 位位图流创建位图。
GUI_CreateBitmapFromStreamM565()	基于红色和蓝色交换的 16 位位图流创建位图。
GUI_CreateBitmapFromStream555()	基于 15 位位图流创建位图。
GUI_CreateBitmapFromStreamM555()	基于红色和蓝色交换的 15 位位图流创建位图。
GUI_CreateBitmapFromStreamRLE16()	基于 RLE16 位图流创建位图。
GUI_CreateBitmapFromStreamRLEM16()	基于红色和蓝色交换的 RLE16 位图流创建位图。

例程	描述
GUI_CreateBitmapFromStream24()	基于 24 位位图流创建位图。
GUI_CreateBitmapFromStreamAlpha()	基于 32 位位图流创建位图。
GUI_CreateBitmapFromStreamRLEAlpha()	基于 RLE 压缩的 8 位 Alpha 位图流创建位图。
GUI_CreateBitmapFromStreamRLE32()	基于 RLE32 位图流创建位图。
GUI_DrawStreamedBitmap()	绘制位图流。
GUI_DrawStreamedBitmapEx()	绘制位图流，而不加载完整的图像。
GUI_GetStreamedBitmapInfo()	返回有关给定流的信息。
GUI_GetStreamedBitmapInfoEx()	返回有关可能位于任何媒介的给定流的信息。
GUI_SetStreamedBitmapHook()	为 GUI_DrawStreamedBitmapEx() 设置挂钩函数。
绘制线条	
GUI_DrawHLine()	绘制水平线。
GUI_DrawLine()	绘制从某个指定起点到指定终点的线条 （绝对坐标）。
GUI_DrawLineRel()	绘制从当前位置到按 X 和 Y 距离指定的终点的线条 （相对坐标）。
GUI_DrawLineTo()	绘制从当前位置到指定终点的线条。
GUI_DrawPolyLine()	绘制折线。
GUI_DrawVLine()	绘制垂直线。
GUI_GetLineStyle()	返回当前的线条样式。
GUI_MoveRel()	相对于其当前位置移动线条指针。
GUI_MoveTo()	将线条指针移动到给定位置。
GUI_SetLineStyle()	设置当前的线条样式。
绘制多边形	
GUI_DrawPolygon()	绘制多边形的轮廓。
GUI_EnlargePolygon()	扩展多边形。
GUI_FillPolygon()	绘制填充的多边形。
GUI_MagnifyPolygon()	放大多边形。
GUI_RotatePolygon()	按指定角度旋转多边形。
绘制圆	
GUI_DrawCircle()	绘制圆的轮廓。
GUI_FillCircle()	绘制填充的圆。
绘制椭圆	
GUI_DrawEllipse()	绘制椭圆的轮廓。
GUI_FillEllipse()	绘制填充的椭圆。
绘制弧线	
GUI_DrawArc()	绘制弧线。
绘制线图	
GUI_DrawGraph()	绘制线图。
绘制饼图	
GUI_DrawPie()	绘制圆形扇区。
保存和恢复 GUI 环境	
GUI_RestoreContext()	恢复 GUI 环境。
GUI_SaveContext()	保存 GUI 环境。
裁剪	
GUI_SetClipRect()	设置用于裁剪的矩形。

GUI_GetPixelIndex()

描述

返回给定位置的颜色指数。

原型

```
unsigned GUI_GetPixelIndex(int x, int y);
```

参数	描述
x	像素的绝对 x 位置
y	像素的绝对 y 位置

7.2 绘制模式

emWin 可在 NORMAL 或 XOR 模式中进行绘制。默认为 NORMAL 模式，在该模式下，显示的内容将被超量绘制。在 XOR 模式中，超量绘制时显示的内容方向将倒转。

GUI_DRAWMODE_XOR 相关的限制

- XOR 模式仅在活动窗口或屏幕内使用两种显示颜色时比较有用。
- emWin 的部分绘制函数在此绘制模式下无法正确运行。一般而言，此模式仅适用于一像素的画笔大小。这意味着，在 XOR 模式下运行时，使用 GUI_DrawLine()、GUI_DrawCircle()、GUI_DrawRect() 等函数之前，必须确保画笔大小设置为 “1”。
- 使用大于 1 位 / 像素 (bpp) 的色深绘制位图时，此绘制模式则不起作用。
- 使用 GUI_DrawPolyLine() 等绘制函数或多次调用 GUI_DrawLineTo() 时，支点将进行两次倒转。其结果是这些像素依然留在背景颜色中。

GUI_GetDrawMode()

描述

返回当前的绘制模式。

原型

```
GUI_DRAWMODE GUI_GetDrawMode(void);
```

返回值

当前选择的绘制模式。

其他信息

有关绘制模式的详情，请参阅 “GUI_GetDrawMode()”（第 100 页）函数。

GUI_SetDrawMode()

描述

选择指定的绘制模式。

原型

```
GUI_DRAWMODE GUI_SetDrawMode(GUI_DRAWMODE mode);
```

参数	描述
mode	要设置的绘制模式。可能是任何例程返回的值，该例程设置绘制模式或以下常数之一。

参数 mode 的允许值	
GUI_DM_NORMAL	默认：绘制点、线、区域、位图。
GUI_DM_XOR	在显示器中重写其他对象的颜色时倒转点、线、区域。

返回值

所选的绘制模式。

其他信息

除设置绘制模式外，此例程还可用于恢复之前更改的绘制模式。
如果使用颜色，倒转的像素的计算方式如下：
新像素颜色 = 颜色的值 - 实际像素颜色 -1。

示例

```
//
// Showing two circles, the second one XOR-combined with the first:
//
GUI_Clear();
GUI_SetDrawMode(GUI_DRAWMODE_NORMAL);
GUI_FillCircle(120, 64, 40);
GUI_SetDrawMode(GUI_DRAWMODE_XOR);
GUI_FillCircle(140, 84, 40);
```

上述示例的屏幕截图

7.3 查询当前客户区矩形

GUI_GetClientRect()

描述

当前的客户区矩形取决于是否在使用窗口管理器。如果正在使用窗口管理器，则可使用函数 WM_GetClientRect 来检索客户区矩形。如果未使用窗口管理器，则客户区矩形与整个 LCD 显示器相对应。

原型

```
void GUI_GetClientRect(GUI_RECT * pRect);
```

参数	描述
pRect	指向存储结果的 GUI_RECT-structure。

7.4 画笔大小

画笔大小决定以下矢量绘制操作的厚度：

- GUI_DrawPoint()
- GUI_DrawLine()
- GUI_DrawLineRel()
- GUI_DrawLineTo()
- GUI_DrawPolyLine()
- GUI_DrawPolygon()
- GUI_DrawEllipse()
- GUI_DrawArc()

请注意，无法合并画笔大小 >1 的线条样式。

GUI_GetPenSize()

描述

返回当前的画笔大小。

原型

```
U8 GUI_GetPenSize(void);
```

GUI_SetPenSize()

描述

设置用于更多绘制操作的画笔大小。

原型

```
U8 GUI_SetPenSize(U8 PenSize);
```

参数	描述
PenSize	使用的画笔大小 （单位：像素）。

返回值

之前的画笔大小。

附加信息

画笔大小应 >= 1。

7.5 基本绘制例程

基本绘制例程可以在显示器中的任意位置绘制单个点、水平线和垂直线以及图形。所有可用的绘制模式均能使用。由于大多数应用程序会频繁调用这些例程，因此对它们的速度尽可能进行了优化。例如，水平线和垂直线函数不需要使用单点例程。

GUI_ClearRect()

描述

通过为矩形区域填充背景颜色，清除当前窗口中指定位置的矩形区域。

原型

```
void GUI_ClearRect(int x0, int y0, int x1, int y1);
```

参数	描述
x0	左上角 X 位置。
y0	左上角 Y 位置。
x1	右下角 X 位置。
y1	右下角 Y 位置。

相关主题

GUI_InvertRect()、 GUI_FillRect()

GUI_CopyRect()

描述

将给定矩形区域的内容复制到指定位置。

原型

```
void GUI_CopyRect(int x0, int y0, int x1, int y1, int xSize, int ySize);
```

参数	描述
x0	源矩形的左上角 X 位置。
y0	源矩形的左上角 Y 位置。
x1	目标矩形的左上角 X 位置。
y1	目标矩形的左上角 Y 位置。
xSize	矩形的 X 大小。
ySize	矩形的 Y 大小。

其他信息

源矩形和目标矩形可能会彼此重叠。

GUI_DrawGradientH()

描述

绘制用水平颜色梯度填充的矩形。

原型

```
void GUI_DrawGradientH(int x0, int y0, int x1, int y1,
```

```
GUI_COLOR Color0, GUI_COLOR Color1);
```

参数	描述
x0	左上角 X 位置。
y0	左上角 Y 位置。
x1	右下角 X 位置。
y1	右下角 Y 位置。
Color0	矩形最左侧要绘制的颜色。
Color1	矩形最右侧要绘制的颜色。

示例

```
GUI_DrawGradientH(0, 0, 99, 99, 0x0000FF, 0x00FFFF);
```

上述示例的屏幕截图

GUI_DrawGradientV()

描述

绘制用垂直颜色梯度填充的矩形。

原型

```
void GUI_DrawGradientV(int x0, int y0, int x1, int y1,
                       GUI_COLOR Color0, GUI_COLOR Color1);
```

参数	描述
x0	左上角 X 位置。
y0	左上角 Y 位置。
x1	右下角 X 位置。
y1	右下角 Y 位置。
Color0	矩形最顶端要绘制的颜色。
Color1	矩形最底端要绘制的颜色。

示例

```
GUI_DrawGradientV(0, 0, 99, 99, 0x0000FF, 0x00FFFF);
```

上述示例的屏幕截图

GUI_DrawGradientRoundedH()

描述

绘制用水平颜色梯度填充的圆角矩形。

原型

```
void GUI_DrawGradientRoundedH(int x0, int y0, int x1, int y1, int rd
                              GUI_COLOR Color0, GUI_COLOR Color1);
```

参数	描述
x0	左上角 X 位置。
y0	左上角 Y 位置。
x1	右下角 X 位置。
y1	右下角 Y 位置。
rd	
Color0	矩形最左侧要绘制的颜色。
Color1	矩形最右侧要绘制的颜色。

示例

```
GUI_DrawGradientRoundedH(0, 0,  99, 99, 25, 0x0000FF, 0x00FFFF);
```

上述示例的屏幕截图

GUI_DrawGradientRoundedV()

描述

绘制用垂直颜色梯度填充的圆角矩形。

原型

```
void GUI_DrawGradientRoundedV(int x0,   int y0, int x1,   int y1,
                              GUI_COLOR Color0, GUI_COLOR Color1);
```

参数	描述
x0	左上角 X 位置。
y0	左上角 Y 位置。
x1	右下角 X 位置。
y1	右下角 Y 位置。
Color0	矩形最左侧要绘制的颜色。
Color1	矩形最右侧要绘制的颜色。

示例

```
GUI_DrawGradientRoundedV(0, 0, 99, 99, 25, 0x0000FF, 0x00FFFF);
```

上述示例的屏幕截图

GUI_DrawPixel()

描述

在当前窗口中的指定位置绘制像素。

原型

```
void GUI_DrawPixel(int x, int y);
```

参数	描述
x	像素的 X 位置。
y	像素的 Y 位置。

相关主题

GUI_DrawPoint()

GUI_DrawPoint()

描述

在当前窗口中的指定位置以当前画笔大小绘制点。

原型

```
void GUI_DrawPoint(int x, int y);
```

参数	描述
x	点的 X 位置。
y	点的 Y 位置。

相关主题

GUI_DrawPixel()

GUI_DrawRect()

描述

在当前窗口中的指定位置绘制矩形。

原型

```
void GUI_DrawRect(int x0, int y0, int x1, int y1);
```

参数	描述
x0	左上角 X 位置。
y0	左上角 Y 位置。
x1	右下角 X 位置。
y1	右下角 Y 位置。

GUI_DrawRectEx()

描述

在当前窗口中的指定位置绘制矩形。

原型

```
void GUI_DrawRectEx(const GUI_RECT * pRect);
```

参数	描述
pRect	指向包含矩形坐标的 GUI_RECT-structure

GUI_DrawRoundedFrame()

描述

在当前窗口中的指定位置绘制指定宽度的圆角框。

原型

```
void GUI_DrawRoundedFrame(int x0, int y0, int x1, int y1, int r, int w);
```

参数	描述
x0	左上角 X 位置。
y0	左上角 Y 位置。
x1	右下角 X 位置。
y1	右下角 Y 位置。
r	圆角使用的半径。
w	框的绘制宽度。

GUI_DrawRoundedRect()

描述

在当前窗口中的指定位置绘制圆角矩形。

原型

```
void GUI_DrawRoundedRect(int x0, int y0, int x1, int y1, int r);
```

参数	描述
x0	左上角 X 位置。
y0	左上角 Y 位置。
x1	右下角 X 位置。
y1	右下角 Y 位置。
r	圆角使用的半径。

GUI_FillRect()

描述

在当前窗口中的指定位置绘制填充的矩形区域。

原型

```
void GUI_FillRect(int x0, int y0, int x1, int y1);
```

参数	描述
x0	左上角 X 位置。
y0	左上角 Y 位置。
x1	右下角 X 位置。
y1	右下角 Y 位置。

其他信息

使用当前的绘制模式，通常意味着矩形内的所有像素均已设置。

相关主题

GUI_InvertRect()、 GUI_ClearRect()

GUI_FillRectEx()

描述

在当前窗口中的指定位置绘制填充的矩形。

原型

```
void GUI_FillRectEx(const GUI_RECT * pRect);
```

参数	描述
pRect	指向包含矩形坐标的 GUI_RECT-structure

GUI_FillRoundedRect()

描述

在当前窗口中的指定位置绘制填充的圆角矩形。

原型

```
void GUI_FillRoundedRect(int x0, int y0, int x1, int y1, int r);
```

参数	描述
x0	左上角 X 位置。
y0	左上角 Y 位置。
x1	右下角 X 位置。
y1	右下角 Y 位置。
r	圆角使用的半径。

GUI_InvertRect()

描述

在当前窗口中的指定位置绘制倒转的矩形区域。

原型

```
void GUI_InvertRect(int x0, int y0, int x1, int y1);
```

参数	描述
x0	左上角 X 位置。
y0	左上角 Y 位置。
x1	右下角 X 位置。
y1	右下角 Y 位置。

相关主题

GUI_FillRect()、 GUI_ClearRect()

7.6 Alpha 混合

Alpha 混合是一种合并前景图像与背景来创建半透明效果的方法。Alpha 值决定可见的像素数以及背景透露的像素数。

颜色信息

emWin 内部适用于 32 位的颜色信息：

- 0-7 位：红色
- 8-15 位：绿色
- 16-23 位：蓝色
- 24-31 位：Alpha 信息

Alpha 值为 0 表示不透明，值为 255 表示完全透明。

工作原理

Alpha 混合完全自动执行。唯一需要做的就是使用 GUI_EnableAlpha() 启用 Alpha 混合。在此，8 位以上的颜色信息将作为 Alpha 值管理。

示例

下面的小示例显示了它的工作原理：

```
GUI_EnableAlpha(1);
GUI_SetBkColor(GUI_WHITE);
GUI_Clear();
GUI_SetColor(GUI_BLACK);
GUI_DispStringHCenterAt("Alphablending", 45, 41);
GUI_SetColor((0x40uL << 24) | GUI_RED);
GUI_FillRect(0, 0, 49, 49);
GUI_SetColor((0x80uL << 24) | GUI_GREEN);
GUI_FillRect(20, 20, 69, 69);
GUI_SetColor((0xC0uL << 24) | GUI_BLUE);
GUI_FillRect(40, 40, 89, 89);
```

旧版本

在旧版本中，需要使用函数 GUI_SetAlpha() 来混合前景与当前的背景颜色信息。现在依然有用，但不再是必需的。

GUI_EnableAlpha()

描述

启用或禁用自动 Alpha 混合。

原型

```
unsigned GUI_EnableAlpha(unsigned OnOff);
```

参数	描述
OnOff	1 表示启用自动 Alpha 混合， 0 表示禁用。

返回值

旧状态。

其他信息

启用自动 Alpha 混合后，每个对象的颜色信息将自动决定其透明度。

GUI_SetAlpha()

（弃用）

描述

为所有后续绘制操作启用软件 Alpha 混合。

原型

```
unsigned GUI_SetAlpha(U8 Value);
```

参数	描述
Alpha	用于所有后续绘制操作的 Alpha 值。默认为 0，表示无 Alpha 混合。

返回值

Alpha 混合之前使用的值。

其他信息

该函数设置用于所有后续绘制操作的 Alpha 值。参数 Alpha 值为 0 表示不透明（禁用 Alpha 混合），值为 255 表示完全透明 （不可见）。
请注意，软件 Alpha 混合会增加 CPU 负荷。此外，强烈建议在绘制操作完成后，将 Alpha 值设回默认值。

示例

```
extern const GUI_BITMAP _LogoBitmap;

GUI_SetColor(GUI_BLUE);
GUI_FillCircle(100, 50, 49);
GUI_SetColor(GUI_YELLOW);
for (i = 0; i < 100; i++) {
  U8 Alpha;
  Alpha = (i * 255 / 100);
  GUI_SetAlpha(Alpha);
  GUI_DrawHLine(i, 100 - i, 100 + i);
}
GUI_SetAlpha(0x80);
GUI_DrawBitmap(&_LogoBitmap, 30, 30);
GUI_SetColor(GUI_MAGENTA);
GUI_SetFont(&GUI_Font24B_ASCII);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DispStringHCenterAt("Alphablending", 100, 3);
GUI_SetAlpha(0);                          /* Set back to default (opaque) */
```

上述示例的屏幕截图

GUI_SetUserAlpha()

描述

设置其他值以用于计算要使用的实际 Alpha 值。
实际 Alpha 值的计算方式如下：

Alpha = AlphaFromObject + ((255 - AlphaFromObject) * UserAlpha) / 255

原型

```
U32 GUI_SetUserAlpha(GUI_ALPHA_STATE * pAlphaState, U32 UserAlpha);
```

参数	描述
pAlphaState	指向用于保存当前状态的 GUI_ALPHA_STATE 结构。
UserAlpha	要使用的值。

返回值

之前的用户 Alpha 值。

其他信息

以下函数 GUI_RestoreUserAlpha() 可用于恢复函数之前的状态。

GUI_RestoreUserAlpha()

描述

恢复用户 Alpha 混合之前的状态。保存在所指向的结构中。

原型

```
U32 GUI_RestoreUserAlpha(GUI_ALPHA_STATE * pAlphaState);
```

参数	描述
pAlphaState	指向包含要恢复的之前状态信息的 GUI_ALPHA_STATE 结构。

返回值

当前的用户 Alpha 值。

示例

```
{
  GUI_ALPHA_STATE AlphaState;

  GUI_EnableAlpha(1);
  GUI_SetBkColor(GUI_WHITE);
  GUI_Clear();
  GUI_SetColor(GUI_BLACK);
  GUI_DispStringHCenterAt("Alphablending", 45, 41);
  GUI_SetUserAlpha(&AlphaState, 0xC0);
  GUI_SetColor(GUI_RED);
  GUI_FillRect(0, 0, 49, 49);
  GUI_SetColor(GUI_GREEN);
  GUI_FillRect(20, 20, 69, 69);
  GUI_SetColor(GUI_BLUE);
  GUI_FillRect(40, 40, 89, 89);
  GUI_RestoreUserAlpha(&AlphaState);
}
```

7.7 绘制位图

一般而言， emWin 能够在任何显示器位置显示所有位图图像。在 16 位 CPU(sizeof(int) == 2) 中，默认一个位图的大小限制为 64 kb。如果需要使用 16 位 CPU 显示更大的位置，请参阅“配置”（第 905 页）。

GUI_DrawBitmap()

描述

在当前窗口中的指定位置绘制位图图像。

原型

```
void GUI_DrawBitmap(const GUI_BITMAP * pBM, int x, int y);
```

参数	描述
pBM	指向要显示的位图。
x	显示器中位图左上角的 X 位置。
y	显示器中位图左上角的 Y 位置。

其他信息

图片数据的解释为从第一个字节的最高有效位 (msb) 开始的位流。
新行始终在偶数字节地址开始，就像位图的第 n 行从偏移 n*BytesPerLine 开始。在客户区的任何位置都能看到位图。
通常，使用位图转换器生成位图。更多详细信息，请参阅 “位图转换器”（第 213 页）。

示例

```
extern const GUI_BITMAP bmSeggerLogoBlue;  /* declare external Bitmap */

void main() {
  GUI_Init();
  GUI_DrawBitmap(&bmSeggerLogoBlue, 45, 20);
}
```

上述示例的屏幕截图

GUI_DrawBitmapEx()

描述

此例程可以在显示器中缩放和 / 或镜像位图。

原型

```
void GUI_DrawBitmapEx(const GUI_BITMAP * pBitmap,
                      int x0,      int y0,
                      int xCenter, int yCenter,
                      int xMag,    int yMag);
```

参数	描述
pBM	指向要显示的位图。
x0	显示器中定位点的 X 位置。
y0	显示器中定位点的 Y 位置。
xCenter	位图中定位点的 X 位置。
yCenter	位图中定位点的 Y 位置。
xMag	X 方向的比例因子。
yMag	Y 方向的比例因子。

其他信息

xMag 参数为负值将在 X 轴镜像位图，yMag 参数为负值将在 Y 轴镜像位图。xMag 和 yMag 的单位为千分之一。 xCenter 和 yCenter 给定的位置指定在显示器中 x0/y0 位置（不考虑比例或镜像）显示的位图的像素。
不能使用此函数绘制 RLE 压缩的位图。

GUI_DrawBitmapHWAlpha()

描述

在具有硬件 Alpha 混合支持的多层系统上绘制带 Alpha 信息的位图。

原型

```
void GUI_DrawBitmapHWAlpha(const GUI_BITMAP GUI_UNI_PTR * pBM,
                           int x0, int y0);
```

参数	描述
pBM	指向要显示的位图。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。

其他信息

在 emWin 中，逻辑颜色作为 32 位值处理。较低的 24 位用于颜色信息，较高的 8 位用于管理 Alpha 值。 Alpha 值为 0 表示图像不透明，值为 0xFF 表示完全透明 （不可见）。
在硬件支持 Alpha 混合的系统上，需要将 Alpha 值写入执行 Alpha 混合的显示控制器中。
通常，硬件的 Alpha 格式与上述 emWin 中的 Alpha 定义不同。大多情况下，值为 0 表示完全透明，值越高表示像素可见度越高。
因此，大多数情况下，需要定制颜色转换例程将逻辑颜色转换为所需的硬件格式。“样本”文件夹中包含 ALPHA_DrawBitmapHWAlpha 示例，它显示了如何考虑定制颜色转换的要求。

GUI_DrawBitmapMag()

描述

此例程可以放大显示器中的位图。

原型

```
void GUI_DrawBitmapMag(const GUI_BITMAP * pBM,
                       int x0,   int y0,
                       int XMul, int YMul);
```

参数	描述
pBM	指向要显示的位图。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。
XMul	X 方向的放大系数。
YMul	Y 方向的放大系数。

7.8 绘制流位图

与 C 文件格式的位图相反，流位图可放在任何位置。位图流可用于创建位图。这些位图可按与 C 文件格式位图相同的方式使用。此外， emWin 支持直接绘制基于调色板的位图流，而不必在可寻址区域（RAM 或 ROM）放置完整的图像。

GUI_CreateBitmapFromStream()

描述

该函数通过传输任何类型的位图流创建位图结构。

原型

```
int  GUI_CreateBitmapFromStream(GUI_BITMAP *      pBMP,
                                GUI_LOGPALETTE * pPAL,
                                const void *     p);
```

参数	描述
pBMP	指向该函数初始化的 GUI_BITMAP 结构。
pPAL	指向该函数初始化的 GUI_LOGPALETTE 结构。
p	指向数据流。

返回值

0 表示成功， 1 表示错误。

其他信息

如果数据流可能包含多种位图格式或未知，应使用此函数。使用此函数的缺点在于它占用很大的存储器空间。如果需要顾虑存储器使用量 (ROM)，则最好使用以下特定格式的函数。

GUI_CreateBitmapFromStreamIDX(), GUI_CreateBitmapFromStreamRLE4(), GUI_CreateBitmapFromStreamRLE8(), GUI_CreateBitmapFromStream565(), GUI_CreateBitmapFromStreamM565(), GUI_CreateBitmapFromStream555(), GUI_CreateBitmapFromStreamM555(), GUI_CreateBitmapFromStreamRLE16(), GUI_CreateBitmapFromStreamRLEM16(), GUI_CreateBitmapFromStream24(), GUI_CreateBitmapFromStreamAlpha(), GUI_CreateBitmapFromStreamRLEAlpha(), GUI_CreateBitmapFromStreamRLE32()

描述

这些函数可通过传输已知格式的位图流，创建位图结构。

原型

```
int  GUI_CreateBitmapFromStream<FORMAT>(GUI_BITMAP *      pBMP,
                                        GUI_LOGPALETTE * pPAL,
                                        const void *     p);
```

参数	描述
pBMP	指向该函数初始化的 GUI_BITMAP 结构。
pPAL	指向该函数初始化的 GUI_LOGPALETTE 结构。
p	指向数据流。

支持的数据流格式

下表显示了每个函数支持的数据流格式：

函数	支持的数据流格式
GUI_CreateBitmapFromStreamIDX()	基于指数的位图流。
GUI_CreateBitmapFromStreamRLE4()	RLE4 压缩的位图流。
GUI_CreateBitmapFromStreamRLE8()	RLE8 压缩的位图流。
GUI_CreateBitmapFromStream565()	高色彩位图流 (565)。
GUI_CreateBitmapFromStreamM565()	高色彩位图流 (M565)。
GUI_CreateBitmapFromStream555()	高色彩位图流 (555)。
GUI_CreateBitmapFromStreamM555()	高色彩位图流 (M565)。
GUI_CreateBitmapFromStreamRLE16()	RLE16 压缩的位图流。
GUI_CreateBitmapFromStreamRLEM16()	RLE16 压缩的位图流，红色和蓝色交换。
GUI_CreateBitmapFromStream24()	24 bpp 位图流 （真彩色）。
GUI_CreateBitmapFromStreamAlpha()	32 bpp 位图流 （真彩色，带 Alpha 通道）。
GUI_CreateBitmapFromStreamRLEAlpha()	RLE 压缩的 8 bpp Alpha 位图流。
GUI_CreateBitmapFromStreamRLE32()	RLE32 压缩的位图流 （真彩色，带 Alpha 通道）。

返回值

0 表示成功， 1 表示错误。

其他信息

如果数据流包含已知的格式，则应使用这些函数。这样，可避免链接未用的代码并使二进制编码较小。

GUI_DrawStreamedBitmap()

描述

基于数据位图数据流绘制位图。

原型

```
void GUI_DrawStreamedBitmap(const void * p, int x, int y);
```

参数	描述
p	指向数据流。
x	显示器中位图左上角的 X 位置。
y	显示器中位图左上角的 Y 位置。

其他信息

您可以使用位图转换器（见“位图转换器”（第 213 页））来创建位图数据流。这些流的格式与 bmp 文件的格式不相同。

GUI_DrawStreamedBitmapEx()

描述

如果没有足够的 RAM 或 ROM 来保存可寻址存储器 （RAM 或 ROM）内的整个文件，可使用此函数来绘制数据流。图形用户界面 (GUI) 库将调用参数 pfGetData 所指向的函数来读取数据。GetData 函数需要返回请求的字节数。 GUI 请求的最大字节数为绘制一行图像所需的字节数。

原型

```
int GUI_DrawStreamedBitmapEx(GUI_GET_DATA_FUNC * pfGetData,
                             const void         * p, int x, int y);
```

参数	描述
pfGetData	指向获取数据调用的函数。有关 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
x	显示器中位图左上角的 X 位置。
y	显示器中位图左上角的 Y 位置。

返回值

0 表示成功， 1 表示错误。

其他信息

另请参阅函数 GUI_SetStreamedBitmapHook()。

GUI_GetStreamedBitmapInfo()

描述

返回附带给定数据流相关信息的结构。

原型

```
void GUI_GetStreamedBitmapInfo(const void *            p,
                               GUI_BITMAPSTREAM_INFO * pInfo);
```

参数	描述
p	指向数据流。
pInfo	指向由该函数填充的 GUI_BITMAPSTREAM_INFO 结构。

GUI_BITMAPSTREAM_INFO 的元素

数据类型	元素	描述
int	XSize	图像的 X 像素大小。
int	YSize	图像的 Y 像素大小。
int	BitsPerPixel	每像素的位数。
int	NumColors	基于指数的图像的颜色数。
int	HasTrans	对于基于指数的图像， 1 表示存在透明度， 0 表示不存在。

GUI_GetStreamedBitmapInfoEx()

描述

返回附带给定数据流相关信息的结构，该数据流无需位于 CPU 的可寻址 ROM 或 RAM 中。

原型

```
int GUI_GetStreamedBitmapInfoEx(GUI_GET_DATA_FUNC *     pfGetData,
                                const void *            p,
                                GUI_BITMAPSTREAM_INFO * pInfo);
```

参数	描述
pfGetData	指向获取数据调用的函数。有关 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
pInfo	指向由该函数填充的 GUI_BITMAPSTREAM_INFO 结构。

返回值

0 表示成功， 1 表示错误。

GUI_BITMAPSTREAM_INFO 的元素

请参阅 GUI_GetStreamedBitmapInfo()。

GUI_SetStreamedBitmapHook()

描述

设置挂钩函数，用于管理未位于 CPU 可寻址区域中的流位图的调色板。挂钩函数在执行 GUI_DrawStreamedBitmapEx() 时调用。

原型

```
void GUI_SetStreamedBitmapHook(
                              GUI_BITMAPSTREAM_CALLBACK pfStreamedBitmapHook);
```

参数	描述
pfStreamedBitmapHook	GUI_DrawStreamedBitmapEx() 调用的挂钩函数。

挂钩函数的原型

```
void * Hook(GUI_BITMAPSTREAM_PARAM * pParam);
```

参数	描述
pParam	指向 GUI_BITMAPSTREAM_PARAM 结构。

GUI_BITMAPSTREAM_PARAM 的元素

数据类型	元素	描述
int	Cmd	要执行的命令。
U32	V	取决于要执行的命令。
void *	p	取决于要执行的命令。

参数 Cmd 支持的值	
GUI_BITMAPSTREAM_GET_BUFFER	收到此命令后，应用程序可以占用位图流调色板的缓冲区。参数： p – 指示缓冲区或 NULL v – 请求缓冲区大小
GUI_BITMAPSTREAM_RELEASE_BUFFER	如果应用程序已占用调色板的缓冲区，在此应释放该缓冲区。参数： p – 指示要释放的缓冲区 v – 未使用
GUI_BITMAPSTREAM_MODIFY_PALETTE	在加载调色板之后但绘制图像之前发送此命令，以便能够修改流图像的调色板。参数： p – 指示调色板数据 v – 调色板中的颜色数

示例

```
static void * _cbStreamedBitmapHook(GUI_BITMAPSTREAM_PARAM * pParam) {
  void * p = NULL;
  int    i, NumColors;
  U32    Color;
  U32  * pColor;

  switch (pParam->Cmd) {
  case GUI_BITMAPSTREAM_GET_BUFFER:
    //
    // Allocate buffer for palette data
    //
    p = malloc(pParam->v);
    break;
  case GUI_BITMAPSTREAM_RELEASE_BUFFER:
    //
    // Release buffer
    //
    free(pParam->p);
    break;
  case GUI_BITMAPSTREAM_MODIFY_PALETTE:
    //
    // Do something with the palette...
    //
    NumColors = pParam->v;
    pColor    = (U32 *)pParam->p;
    Color     = *(pColor + pParam->v - 1);
    for (i = NumColors - 2; i >= 0; i--) {
      *(pColor + i + 1) = *(pColor + i);
    }
    *pColor = Color;
    break;
  }
  return p;
}
```

7.9 绘制线条

最常用的绘制例程是绘制从一点到另一点的线的例程。

GUI_DrawHLine()

描述

在当前窗口中绘制从某个指定起点到某个指定终点的 1 像素厚的水平直线。

原型

```
void GUI_DrawHLine(int y, int x0, int x1);
```

参数	描述
y	Y 位置。
x0	X 起始位置。
x1	X 结束位置。

其他信息

如果 x1 < x0，则不会显示任何内容。
大多数 LCD 控制器能够非常迅速地执行此例程，因为一次可以设置多个像素，且无需进行计算。如果明确要绘制水平直线，则此例程的执行速度比 GUI_DrawLine() 例程快。

GUI_DrawLine()

描述

在当前窗口中绘制从某个指定起点到某个指定终点之间的线 （绝对坐标）。

原型

```
void GUI_DrawLine(int x0, int y0, int x1, int y1);
```

参数	描述
x0	X 起始位置。
y0	Y 起始位置。
x1	X 结束位置。
y1	Y 结束位置。

其他信息

如果部分线因没有位于当前窗口中而不可见或当前部分窗口不可见，这是由于裁剪的原因。

GUI_DrawLineRel()

描述

在当前窗口中绘制从当前 (X,Y) 位置到按 X 和 Y 距离指定的终点之间的线 （相对坐标）。

原型

```
void GUI_DrawLineRel(int dx, int dy);
```

参数	描述
dx	X 方向到要绘制的线的终点的距离。
dy	Y 方向到要绘制的线的终点的距离。

GUI_DrawLineTo()

描述

在当前窗口中绘制从当前 (X,Y) 位置到按 X 和 Y 坐标指定的终点之间的线。

原型

```
void GUI_DrawLineTo(int x, int y);
```

参数	描述
x	X 结束位置。
y	Y 结束位置。

GUI_DrawPolyLine()

描述

在当前窗口中连接预定义的点列表和线。

原型

```
void GUI_DrawPolyLine(const GUI_POINT * pPoint, int NumPoints,
                      int x, int y);
```

参数	描述
pPoint	指向要显示的折线。
NumPoints	点列表中指定的点数量。
x	原点的 X 位置。
y	原点的 Y 位置。

其他信息

折线的起点和终点不需要一样。

GUI_DrawVLine()

描述

在当前窗口中绘制从某个指定起点到某个指定终点的 1 像素厚的垂直线。

原型

```
void GUI_DrawVLine(int x, int y0, int y1);
```

参数	描述
x	X 位置。
y0	Y 起始位置。
y1	Y 结束位置。

其他信息

如果 y1 < y0，则不会显示任何内容。
大多数 LCD 控制器能够非常迅速地执行此例程，因为一次可以设置多个像素，且无需进行计算。如果明确要绘制水平直线，则此例程的执行速度比 GUI_DrawLine() 例程快。

GUI_GetLineStyle()

描述

返回函数 GUI_DrawLine 当前使用的线条样式。

原型

```
U8 GUI_GetLineStyle(void);
```

返回值

函数 GUI_DrawLine 当前使用的线条样式。

GUI_MoveRel()

描述

相对于其当前位置移动当前的线条指针。

原型

```
void GUI_MoveRel(int dx, int dy);
```

参数	描述
dx	X 方向移动的距离。
dy	Y 方向移动的距离。

相关主题

GUI_DrawLineTo()、 GUI_MoveTo()

GUI_MoveTo()

描述

将当前的线条指针移动到给定位置。

原型

```
void GUI_MoveTo(int x, int y);
```

参数	描述
x	X 的新位置。
y	Y 的新位置。

GUI_SetLineStyle()

描述

设置函数 GUI_DrawLine 当前使用的线条样式。

原型

```
U8 GUI_SetLineStyle(U8 LineStyle);
```

参数	描述
LineStyle	要使用的新线条样式 （见下表）。

参数 LineStyle 的允许值	
GUI_LS_SOLID	按实线样式绘制线条 （默认）。
GUI_LS_DASH	按虚线样式绘制线条。
GUI_LS_DOT	按圆点样式绘制线条。
GUI_LS_DASHDOT	按虚线和圆点交替的样式绘制线条。
GUI_LS_DASHDOTDOT	按虚线和双点交替的样式绘制线条。

返回值

函数 GUI_DrawLine 之前使用的线条样式。

其他信息

此函数仅设置 GUI_DrawLine 使用的线条样式。该样式仅用于画笔大小为 1 的情况。

7.10 绘制多边形

多边形绘制例程对于绘制矢量化符号非常有帮助。

GUI_DrawPolygon()

描述

在当前窗口中绘制按点列表定义的多边形的轮廓。

原型

```
void GUI_DrawPolygon(const GUI_POINT * pPoint, int NumPoints,
                     int x, int y);
```

参数	描述
pPoint	指向要显示的多边形。
NumPoints	点列表中指定的点数量。
x	原点的 X 位置。
y	原点的 Y 位置。

其他信息

通过将终点连接到起点，绘制的多边形将自动闭合。

GUI_EnlargePolygon()

描述

按指定的长度 （像素）全方位扩展多边形。

原型

```
void GUI_EnlargePolygon(GUI_POINT       * pDest,
                        const GUI_POINT * pSrc,
                        int               NumPoints,
                        int               Len);
```

参数	描述
pDest	指向目标多边形。
pSrc	指向源多边形。
NumPoints	点列表中指定的点数量。
Len	扩展多边形的长度 （像素）。

其他信息

确保点的目标数组等于或大于源数组。

示例

```
const GUI_POINT aPoints[] = {
  { 40, 20},
  {  0, 20},
  { 20,  0}
};

GUI_POINT aEnlargedPoints[GUI_COUNTOF(aPoints)];

void Sample(void) {
  int i;
  GUI_Clear();
  GUI_SetDrawMode(GUI_DM_XOR);
  GUI_FillPolygon(aPoints, GUI_COUNTOF(aPoints), 140, 110);
  for (i = 1; i < 10; i++) {
    GUI_EnlargePolygon(aEnlargedPoints, aPoints, GUI_COUNTOF(aPoints), i * 5);
    GUI_FillPolygon(aEnlargedPoints, GUI_COUNTOF(aPoints), 140, 110);
  }
}
```

上述示例的屏幕截图

GUI_FillPolygon()

描述

在当前窗口中绘制按点列表定义的填充的多边形。

原型

```
void GUI_FillPolygon(const GUI_POINT * pPoint, int NumPoints, int x, int y);
```

参数	描述
pPoint	指向要显示和填充的多边形。
NumPoints	点列表中指定的点数量。
x	原点的 X 位置。
y	原点的 Y 位置。

其他信息

通过将终点连接到起点，绘制的多边形将自动闭合。终点无需接触到多边形的轮廓。
通过为多边形的各个 y 位置绘制一条或多条水平直线，可对多边形进行渲染。默认情况下，为一个 y 位置绘制水平直线使用的最大点数量为 12（意味着每个 y 位置需要 6 条直线）。如果需要增加此值，可使用宏 GUI_FP_MAXCOUNT 设置最大点数量。

示例

```
#define GUI_FP_MAXCOUNT 50
```

GUI_MagnifyPolygon()

描述

按指定系数放大多边形。

原型

```
void GUI_MagnifyPolygon(GUI_POINT *       pDest,
                        const GUI_POINT * pSrc,
                        int               NumPoints,
                        int               Mag);
```

参数	描述
pDest	指向目标多边形。
pSrc	指向源多边形。
NumPoints	点列表中指定的点数量。
Mag	放大多边形使用的系数。

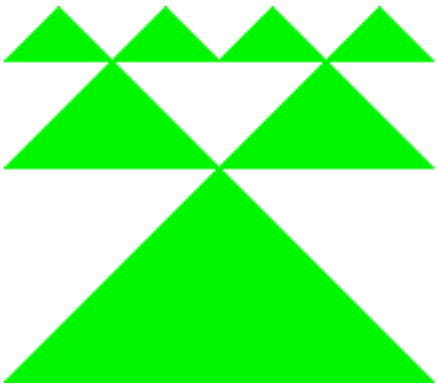
其他信息

确保点的目标数组等于或大于源数组。
请注意扩展和放大多边形之间的区别。调用函数 GUI_EnlargePolygon()（参数 Len = 1）将使多边形的所有边扩展 1 像素，而调用 GUI_MagnifyPolygon()（参数 Mag = 1）则没有效果。

示例

```
const GUI_POINT aPoints[] = {
  {  0, 20},
  { 40, 20},
  { 20,  0}
};

GUI_POINT aMagnifiedPoints[GUI_COUNTOF(aPoints)];

void Sample(void) {
  int Mag, y = 0, Count = 4;
  GUI_Clear();
  GUI_SetColor(GUI_GREEN);
  for (Mag = 1; Mag <= 4; Mag *= 2, Count /= 2) {
    int i, x = 0;
    GUI_MagnifyPolygon(aMagnifiedPoints, aPoints, GUI_COUNTOF(aPoints), Mag);
    for (i = Count; i > 0; i--, x += 40 * Mag) {
      GUI_FillPolygon(aMagnifiedPoints, GUI_COUNTOF(aPoints), x, y);
    }
    y += 20 * Mag;
  }
}
```

上述示例的屏幕截图



GUI_RotatePolygon()

描述

按指定角度旋转多边形。

原型

```
void GUI_RotatePolygon(GUI_POINT *       pDest,
                       const GUI_POINT * pSrc,
                       int               NumPoints,
```



```
                                float               Angle);
```

参数	描述
pDest	指向目标多边形。
pSrc	指向源多边形。
NumPoints	点列表中指定的点数量。
Angle	旋转多边形使用的角度 （弧度）。

其他信息

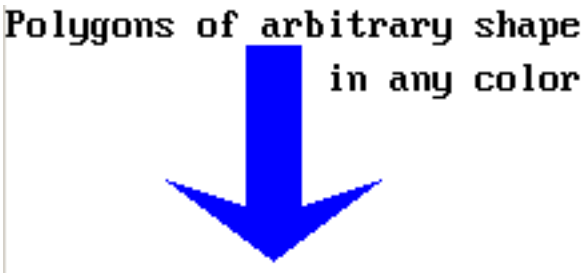
确保点的目标数组等于或大于源数组。

示例

以下示例显示了如何绘制多边形。可像 2DGL_DrawPolygon.c 一样在 emWin 随附的示例中找到该示例。

```
#include "gui.h"
/*********************************************************************
*
*           The points of the arrow
*/
static const GUI_POINT aPointArrow[] = {
  {  0,  -5},
  {-40, -35},
  {-10, -25},
  {-10, -85},
  { 10, -85},
  { 10, -25},
  { 40, -35},
};

/*********************************************************************
*
*               Draws a polygon
*/
static void DrawPolygon(void) {
  int Cnt =0;
  GUI_SetBkColor(GUI_WHITE);
  GUI_Clear();
  GUI_SetFont(&GUI_Font8x16);
  GUI_SetColor(0x0);
  GUI_DispStringAt("Polygons of arbitrary shape ", 0, 0);
  GUI_DispStringAt("in any color", 120, 20);
  GUI_SetColor(GUI_BLUE);
  /* Draw filled polygon */
  GUI_FillPolygon (&aPointArrow[0],7,100,100);
}

/*********************************************************************
*
*                 main
*/
void main(void) {
  GUI_Init();
  DrawPolygon();
  while(1)
    GUI_Delay(100);
}
```

上述示例的屏幕截图



7.11 绘制圆

GUI_DrawCircle()

描述

在当前窗口中的指定位置绘制指定尺寸的圆的轮廓。

原型

```
void GUI_DrawCircle(int x0, int y0, int r);
```

参数	描述
x0	客户端窗口的圆心的 X 位置 （像素）。
y0	客户端窗口的圆心的 Y 位置 （像素）。
r	圆的半径 （直径的一半）。 最小：0 （将成为点）；最大：180。

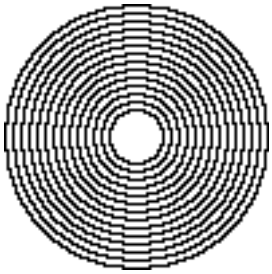
其他信息

此例程无法处理大于 180 的半径，因为它使用整数计算，否则将导致溢出。不过，对于大多数嵌入式应用程序而言，这不是问题，因为直径为 360 的圆比显示器还要大。

示例

```
// Draw concentric circles
void ShowCircles(void) {
  int i;
  for (i=10; i<50; i += 3)
    GUI_DrawCircle(120, 60, i);
}
```

上述示例的屏幕截图



GUI_FillCircle()

描述

在当前窗口中的指定位置绘制指定尺寸的填充的圆。

原型

```
void GUI_FillCircle(int x0, int y0, int r);
```

参数	描述
x0	客户端窗口的圆心的 X 位置 （像素）。
y0	客户端窗口的圆心的 Y 位置 （像素）。
r	圆的半径 （直径的一半）。 最小：0 （将成为点）；最大：180。

其他信息

此例程无法处理大于 180 的半径。

示例

```
GUI_FillCircle(120,60,50);
```

上述示例的屏幕截图

7.12 绘制椭圆

GUI_DrawEllipse()

描述

在当前窗口中的指定位置绘制指定尺寸的椭圆的轮廓。

原型

```
void GUI_DrawEllipse(int x0, int y0, int rx, int ry);
```

参数	描述
x0	客户端窗口的圆心的 X 位置 （像素）。
y0	客户端窗口的圆心的 Y 位置 （像素）。
rx	椭圆的 X 半径 （直径的一半）。 最小：0；最大：180。
ry	椭圆的 Y 半径 （直径的一半）。 最小：0；最大：180。

其他信息

此例程无法处理大于 180 的 rx/ry 参数，因为它使用整数计算，否则将导致溢出。

示例

请参见 GUI_FillEllipse() 示例。

GUI_FillEllipse()

描述

在当前窗口中的指定位置绘制指定尺寸的填充的椭圆。

原型

```
void GUI_FillEllipse(int x0, int y0, int rx, int ry);
```

参数	描述
x0	客户端窗口的圆心的 X 位置 （像素）。
y0	客户端窗口的圆心的 Y 位置 （像素）。
rx	椭圆的 X 半径 （直径的一半）。 最小：0；最大：180。
ry	椭圆的 Y 半径 （直径的一半）。 最小：0；最大：180。

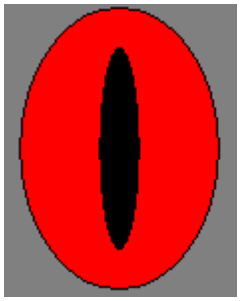
其他信息

此例程无法处理大于 180 的 rx/ry 参数。

示例

```
// Demo ellipses
GUI_SetColor(0xff);
GUI_FillEllipse(100, 180, 50, 70);
GUI_SetColor(0x0);
GUI_DrawEllipse(100, 180, 50, 70);
GUI_SetColor(0x000000);
GUI_FillEllipse(100, 180, 10, 50);
```

上述示例的屏幕截图



7.13 绘制弧线

GUI_DrawArc()

描述

在当前窗口中的指定位置绘制指定尺寸的弧线。弧线是圆轮廓的一部分。

原型

```
void GUI_DrawArc(int xCenter, int yCenter, int rx, int ry, int a0, int a1);
```

参数	描述
xCenter	客户端窗口中心的水平位置 （像素）。
yCenter	客户端窗口中心的垂直位置 （像素）。
rx	X 半径 （像素）。
ry	Y 半径 （像素）。
a0	起始角度 （度）。
a1	结束角度 （度）。

局限

当前未使用 ry 参数，而使用 rx 参数代替。

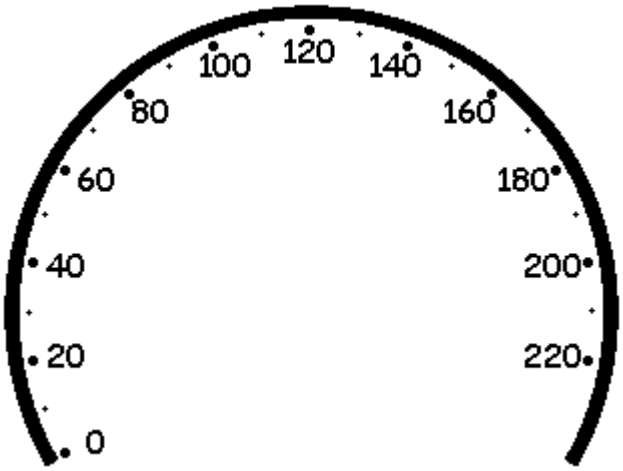
其他信息

GUI_DrawArc() 使用浮点库。它无法处理大于 180 的 rx/ry 参数，因为它使用整数计算，否则将导致溢出。

示例

```
void DrawArcScale(void) {
  int x0 = 160;
  int y0 = 180;
  int i;
  char ac[4];
  GUI_SetBkColor(GUI_WHITE);
  GUI_Clear();
  GUI_SetPenSize( 5 );
  GUI_SetTextMode(GUI_TM_TRANS);
  GUI_SetFont(&GUI_FontComic18B_ASCII);
  GUI_SetColor(GUI_BLACK);
  GUI_DrawArc( x0,y0,150, 150,-30, 210 );
  GUI_Delay(1000);
  for (i=0; i<= 23; i++) {
    float a = (-30+i*10)*3.1415926/180;
    int x = -141*cos(a)+x0;
    int y = -141*sin(a)+y0;
    if (i%2 == 0)
      GUI_SetPenSize( 5 );
    else
      GUI_SetPenSize( 4 );
    GUI_DrawPoint(x,y);
    if (i%2 == 0) {
      x = -123*cos(a)+x0;
      y = -130*sin(a)+y0;
      sprintf(ac, "%d", 10*i);
      GUI_SetTextAlign(GUI_TA_VCENTER);
      GUI_DispStringHCenterAt(ac,x,y);
    }
  }
}
```

上述示例的屏幕截图



7.14 绘制线图

GUI_DrawGraph()

描述

立即绘制线图。

原型

```
void GUI_DrawGraph(I16 * paY, int NumPoints, int x0, int y0);
```

参数	描述
paY	指向包含线图 Y 值的数组。
NumPoints	要显示的 Y 值的数量。
x0	X 的起始点。
y0	y 的起始点。

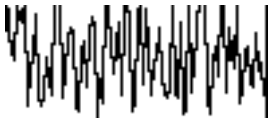
其他信息

该函数首先设置到使用 x0、y0 和给定数组的第一个 Y 值所指定位置的直线光标。然后，将开始绘制到 x0 + 1、y0 + *(paY + 1)、x0 + 2、y0 + *(paY + 2) 等位置的线。

示例

```
#include "GUI.h"
#include <stdlib.h>

I16 aY[100];

void main(void) {
  int i;
  GUI_Init();
  for (i = 0; i < GUI_COUNTOF(aY); i++) {
    aY[i] = rand() % 50;
  }
  GUI_DrawGraph(aY, GUI_COUNTOF(aY), 0, 0);
}
```

上述示例的屏幕截图



7.15 绘制饼图

GUI_DrawPie()

描述

绘制圆形扇区。

原型

```
void GUI_DrawPie(int x0, int y0, int r, int a0, int a1, int Type);
```

参数	描述
x0	客户端窗口的圆心的 X 位置 （像素）。
y0	客户端窗口的圆心的 Y 位置 （像素）。
r	圆的半径 （直径的一半）。
a0	起始角度 （度）。
a1	结束角度 （度）。
Type	（留待将来使用，应为 0）

示例

```
int i, a0, a1;
const unsigned aValues[]  = { 100, 135, 190, 240, 340, 360};
const GUI_COLOR aColors[] = { GUI_BLUE, GUI_GREEN,   GUI_RED,
                              GUI_CYAN, GUI_MAGENTA, GUI_YELLOW };
for (i = 0; i < GUI_COUNTOF(aValues); i++) {
  a0 = (i == 0) ?0 :aValues[i - 1];
  a1 = aValues[i];
  GUI_SetColor(aColors[i]);
  GUI_DrawPie(100, 100, 50, a0, a1, 0);
}
```

上述示例的屏幕截图

7.16 保存和恢复 GUI 环境

GUI_RestoreContext()

描述

该函数将恢复 GUI 环境。

原型

```
void GUI_RestoreContext(const GUI_CONTEXT * pContext);
```

参数	描述
pContext	指向包含新环境的 GUI_CONTEXT 结构。

其他信息

GUI 环境包含 GUI 的当前状态，例如文本游标位置、当前字体指针等。有时，保存当前状态并在以后进行恢复非常有用。对此，您可以使用这些函数。

GUI_SaveContext()

描述

该函数将保存当前的 GUI 环境。（另请参见 GUI_RestoreContext）

原型

```
void GUI_SaveContext(GUI_CONTEXT * pContext);
```

参数	描述
pContext	指向用于保存当前环境的 GUI_CONTEXT 结构。

7.17 裁剪

GUI_SetClipRect()

描述

设置用于限制输出的裁剪矩形。

原型

```
void GUI_SetClipRect(const GUI_RECT * pRect);
```

参数	描述
pRect	指向应用于裁剪的矩形。恢复默认值应使用空指针。

其他信息

默认情况下，裁剪区域限制为配置的 （虚拟）显示器大小。
某些情况下，使用通过此函数可以设置的较小的裁剪矩形比较有用。
引用的矩形应保持不变，直到使用空指针再次调用该函数。

示例

以下示例显示了如何使用该函数：

```
GUI_RECT Rect = {10, 10, 100, 100};
GUI_SetClipRect(&Rect);
.
. /* Use the clipping area ...*/
.
GUI_SetClipRect(NULL);
```

第 8 章

显示位图文件

显示编译时已知位图建议采用且最高效的方式是使用位图转换器将其转换为 C 文件，并添加到项目 / 生成文件中。有关位图转换器的详细信息，请参阅 “位图转换器”（第 213 页）。
如果应用程序需要显示编译时未知的图像，该图像需要有受 emWin 支持的图形文件格式。在这种情况下，该图像文件可位于存储器或其他存储设备中，即使可用的存储量低于该图像文件的大小，亦可以对其显示。
emWin 当前支持 BMP、 JPEG、 GIF 和 PNG 文件格式。

8.1 BMP 文件支持

尽管可用于 emWin 的位图通常作为 C 文件编译和链接到应用程序，但有时候使用这些类型的结构可能并不理想。常见的示例即不断引用用户下载的位图等新图像的应用程序。下列函数支持加载到存储器的 bmp 文件。
对于您计划再利用（即公司图标）的图像，将其作为 emWin 可直接使用的 C 文件编译和链接则更为便捷。使用位图转换器可轻松完成此任务。

8.1.1 支持的格式

BMP 文件格式由 Microsoft 定义。下表显示了许多不同的格式：

每像素位数	索引	压缩	支持
1	是	否	是
4	是	否	是
4	是	是	是
8	是	否	是
8	是	是	是
16	否	否	是
24	否	否	是
32	否	否	是

8.1.2 BMP 文件 API

下表按字母顺序列出了可用的 BMP 文件相关的例程。以下为详细描述：

例程	描述
GUI_BMP_Draw()	绘制已加载到存储器的 BMP 文件。
GUI_BMP_DrawEx()	绘制无需加载到存储器的 BMP 文件。
GUI_BMP_DrawScaled()	绘制已加载到存储器的带比例的 BMP 文件。
GUI_BMP_DrawScaledEx()	绘制无需加载到存储器的带比例的 BMP 文件。
GUI_BMP_GetXSize()	返回加载到存储器的 BMP 文件的 X 大小。
GUI_BMP_GetXSizeEx()	返回无需加载到存储器的 BMP 文件的 X 大小。
GUI_BMP_GetYSize()	返回加载到存储器的位图的 Y 大小。
GUI_BMP_GetYSizeEx()	返回无需加载到存储器的 BMP 文件的 Y 大小。
GUI_BMP_Serialize()	创建 BMP 文件。
GUI_BMP_SerializeEx()	基于给定的矩形创建 BMP 文件。

GUI_BMP_Draw()

描述

在当前窗口中的指定位置绘制已加载到存储器的 Windows bmp 文件。

原型

```
int GUI_BMP_Draw(const void * pFileData, int x0, int y0);
```

参数	描述
pFileData	指向 bmp 文件所在的存储器区域的起始位置。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。

其他信息

本章开头的表显示了受支持的 BMP 文件格式。示例 2DGL_DrawBMP.c 显示了如何使用该函数。

GUI_BMP_DrawEx()

描述

在当前窗口中的指定位置绘制不必加载到存储器的 bmp 文件。

原型

```
int GUI_BMP_DrawEx(GUI_GET_DATA_FUNC * pfGetData, void * p, int x0, int y0);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。

返回值

0 表示成功，非 0 表示该函数失败。

其他信息

如果存储器没有足够的 RAM 加载整个文件，可使用此函数来绘制 bmp 文件。图形用户界面 (GUI) 库将调用参数 pfGetData 所指向的函数来读取数据。 GetData 函数需要返回请求的字节数。 GUI 请求的最大字节数为绘制一行图像所需的字节数。

GUI_BMP_DrawScaled()

描述

在当前窗口中的指定位置通过缩放比例绘制已加载到存储器的 bmp 文件。

原型

```
int GUI_BMP_DrawScaled(const void * pFileData,
                       int x0, int y0, int Num, int Denom);
```

参数	描述
pFileData	指向 bmp 文件所在的存储器区域的起始位置。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。
Num	用于缩放比例的分子。
Denom	用于缩放比例的分母。

返回值

0 表示成功，非 0 表示该函数失败。

其他信息

该函数通过使用给定的分子和分母构建分数来缩放图像。例如，如果图像应缩减至 2/3 大小，则 Num 应为 2， Denom 应为 3。

GUI_BMP_DrawScaledEx()

描述

在当前窗口中的指定位置通过缩放比例绘制不必加载到存储器的 bmp 文件。

原型

```
int  GUI_BMP_DrawScaledEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                          int                 x0,        int    y0,
                          int                 Num,       int    Denom);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。
Num	用于缩放比例的分子。
Denom	用于缩放比例的分母。

返回值

0 表示成功，非 0 表示该函数失败。

其他信息

该函数通过使用给定的分子和分母构建分数来缩放图像。例如，如果图像应缩减至 2/3 大小，则 Num 应为 2， Denom 应为 3。
更多详细信息，请参阅 “GUI_BMP_DrawEx()”（第 143 页）。

GUI_BMP_GetXSize()

描述

返回已加载到存储器的指定位图的 X 大小。

原型

```
int GUI_BMP_GetXSize(const void * pFileData);
```

参数	描述
pFileData	指向 bmp 文件所在的存储器区域的起始位置。

返回值

位图的 X 大小。

GUI_BMP_GetXSizeEx()

描述

返回不必加载到存储器的指定 bmp 文件的 X 大小。

原型

```
int GUI_BMP_GetXSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。

返回值

位图的 X 大小。

GUI_BMP_GetYSize()

描述

返回已加载到存储器的指定位图的 Y 大小。

原型

```
int GUI_BMP_GetYSize(const void * pFileData);;
```

参数	描述
pFileData	指向 bmp 文件所在的存储器区域的起始位置。

返回值

位图的 Y 大小。

GUI_BMP_GetYSizeEx()

描述

返回不必加载到存储器的指定 bmp 文件的 X 大小。

原型

```
int GUI_BMP_GetYSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。

返回值

位图的 Y 大小。

GUI_BMP_Serialize()

描述

该函数创建包含 LCD 完整内容的 BMP 文件。

原型

```
void GUI_BMP_Serialize(GUI_CALLBACK_VOID_U8_P * pfSerialize, void * p);
```

参数	描述
pfSerialize	指向串行化函数
p	指向传递到串行化函数的用户定义数据

其他信息

以下示例将显示如何在窗口中创建 BMP 文件。

```
static void _DrawSomething(void) {
  /* Draw something */
  GUI_DrawLine(10, 10, 100, 100);
}

static void _WriteByte2File(U8 Data, void * p) {
  U32 nWritten;
  WriteFile(*((HANDLE *)p), &Data, 1, &nWritten, NULL);
}

static void _ExportToFile(void) {
  HANDLE hFile = CreateFile("C:\\GUI_BMP_Serialize.bmp",
                            GENERIC_WRITE, 0, 0,
                            CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);
  GUI_BMP_Serialize(_WriteByte2File, &hFile);
  CloseHandle(hFile);
}

void main(void) {
  GUI_Init();
  _DrawSomething();
  _ExportToFile();
}
```

GUI_BMP_SerializeEx()

描述

该函数将创建包含给定区域的 BMP 文件。

原型

```
void GUI_BMP_SerializeEx(GUI_CALLBACK_VOID_U8_P * pfSerialize,
                         int    x0, int y0, int xSize, int ySize,
                         void * p);
```

参数	描述
pfSerialize	指向串行化函数。
x0	创建 BMP 文件的 X 起始位置。
y0	创建 BMP 文件的 Y 起始位置。
xSize	X 的大小。
ySize	Y 的大小。
p	指向传递到串行化函数的用户定义数据。

其他信息

请参阅 “GUI_BMP_Serialize()”（第 146 页）。

8.2 JPEG 文件支持

JPEG（读音为“jay-peg”）是全彩和灰度图像的标准压缩方法。 JPEG 用于压缩“真实世界”的景象、线条画、卡通，其他非现实图像并不是其强项。 JPEG 会有损耗，意指输出图像与输入图像并不完全相同。因此，如果您必须达到完全相同的输出位，则不能使用 JPEG。不过，对于常见的照片图像，可以得到非常好的压缩级别，看不出变化。并且如果您能容忍低质量的图像，则可以实现相当高的压缩级别。

8.2.1 支持的 JPEG 压缩方法

此软件执行 JPEG 基准、可扩展顺序以及渐进式的压缩流程。尽管尚未实施某些不常见的参数设置，但它支持这些流程的所有变体。出于法律原因，不得分发 JPEG 算术编码变体的代码。 JPEG 规范的算术编码选项似乎属于归 IBM、 AT&T 和 Mitsubishi 所有的专利。因此，从法律上讲，如未获得一个或多个许可，则不能使用算术编码。因此，尚未包含对算术编码的支持。（由于算术编码相对于未获专利的 Huffman 模式仅具有限界收益，因此不太可能有太多实施支持它。）
JPEG 文件支持不包含提供标准中定义的层次式或无损流程。

8.2.2 将 JPEG 文件转换为 C 源

某些情况下，将 JPEG 文件作为 C 文件添加到项目中非常有用。这时，首先需要将 JPEG 文件转换为 C 文件。使用 emWin 随附的工具 Bin2C.exe 可完成此任务。在“工具”子文件夹中可找到该工具。它可以将给定的二进制文件（在本例中为 JPEG 文件）转换为 C 文件。C 文件的文件名与二进制文件名称相同，采用文件扩展名“.c”。
以下步骤将显示如何使用 Bin2C 嵌入 JPEG 文件：

- 启动 Bin2C.exe，并选择要转换为 C 文件的 JPEG 文件（例如“Image.jpeg”），然后将其转换为 C 文件。
- 将该 C 文件添加到项目中。

示例

以下示例显示如何显示转换的 JPEG 文件：

```
#include "GUI.h"
#include "Image.c" /* Include the converted C file */

void main(void) {
  GUI_Init();
  GUI_JPEG_Draw(acImage, sizeof(acImage), 0, 0);
  ...
}
```

8.2.3 显示 JPEG 文件

图形库首先对图形信息进行解码。如果必须绘制图像，解码流程将花费相当长的时间。如果在窗口管理器经常调用的 callback 例程中使用 JPEG 文件，则解码流程可能花费相当长的时间。通过使用存储设备可缩短计算时间。最好的方法是先将图像绘制到存储设备中。在这种情况下，将只进行一次解压缩。更多关于存储设备的信息，请参见“存储设备”（第 247 页）。

8.2.4 存储器使用

JPEG 解压缩大约需要 33 Kb RAM 用于与图像大小无关的解压缩和依赖大小的字节量。RAM 要求可按以下方式计算：

App. 大约 RAM 要求 = 图像的 X 大小 * 80 字节 + 33 千字节

依赖于 X 大小的量取决于 JPEG 文件的压缩类型。下表显示了部分示例：

压缩	图像大小 （像素）	RAM 使用 [千字节]	RAM 使用，大小依赖量 [千字节]
H1V1	160x120	45	12
H2V2	160x120	46	13
GRAY	160x120	38	4

解压缩所需的存储器由 emWin 存储器管理系统动态分配。绘制 JPEG 图像后，将释放整个 RAM。

8.2.5 渐进式 JPEG 文件

与基准和可扩展顺序的 JPEG 文件相反，渐进式 JPEG 包含多次扫描。每次扫描均基于上次扫描，并优化 JPEG 图像的外观。这要求即使只有一行需要解压缩，也要对整个文件进行扫描。
如果针对全部图像数据配置了足够的 RAM，则解压缩只需进行一次。如果配置的 RAM 较少， JPEG 解码器将使用 “banding” 绘制图像。所需的带越多，图像需要进行解压缩的次数越多，且性能越慢。换句话说：RAM 越多，性能越好。

8.2.6 JPEG 文件 API

下表按字母顺序列出了可用的 JPEG 文件相关的例程。以下为详细描述：

例程	描述
GUI_JPEG_Draw()	绘制已加载到存储器的 JPEG 文件。
GUI_JPEG_DrawEx()	绘制无需加载到存储器的 JPEG 文件。
GUI_JPEG_DrawScaled()	绘制已加载到存储器的带比例的 JPEG 文件。
GUI_JPEG_DrawScaledEx()	绘制无需加载到存储器的带比例的 JPEG 文件。
GUI_JPEG_GetInfo()	基于已加载到存储器的 JPEG 文件填充 GUI_JPEG_INFO 结构。
GUI_JPEG_GetInfoEx()	基于无需加载到存储器的 JPEG 文件填充 GUI_JPEG_INFO 结构。

GUI_JPEG_Draw()

描述

在当前窗口中的指定位置绘制已加载到存储器的 jpeg 文件。

原型

```
int GUI_JPEG_Draw(const void * pFileData, int DataSize, int x0, int y0);
```

参数	描述
pFileData	指向 jpeg 文件所在的存储器区域的起始位置。
DataSize	Jpeg 文件的字节数。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。

返回值

0 表示成功，非 0 表示该函数失败。（当前的实施始终返回 0）

其他信息

“样本”文件夹中包含示例 2DGL_DrawJPG.c，它显示了如何使用该函数。

GUI_JPEG_DrawEx()

描述

在当前窗口中的指定位置绘制不必加载到存储器的 jpeg 文件。

原型

```
int GUI_JPEG_DrawEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                    int                 x0,         int    y0);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。

返回值

0 表示成功，非 0 表示该函数失败。（当前的实施始终返回 0）

其他信息

如果存储器没有足够的 RAM 加载整个文件，可使用此函数来绘制 jpegs 文件。然后，JPEG 库将调用参数 pfGetData 所指向的函数来读取数据。
GetData 函数应返回可用的字节数。这可能小于或等于请求的字节数。该函数至少需要返回 1 个新字节。“样本”文件夹中包含示例 2DGL_DrawJPGScaled.c，它显示了如何使用 GetData 函数。

GUI_JPEG_DrawScaled()

描述

在当前窗口中的指定位置通过缩放比例绘制已加载到存储器的 jpeg 文件。

原型

```
int  GUI_JPEG_DrawScaled(const void * pFileData, int DataSize,
                         int x0, int y0, int Num, int Denom);
```

参数	描述
pFileData	指向 jpeg 文件所在的存储器区域的起始位置。
DataSize	Jpeg 文件的字节数。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。
Num	用于缩放比例的分子。
Denom	用于缩放比例的分母。

返回值

0 表示成功，非 0 表示该函数失败。（当前的实施始终返回 0）

其他信息

该函数通过使用给定的分子和分母构建分数来缩放图像。例如，如果图像应缩减至 2/3 大小，则 Num 应为 2， Denom 应为 3。
“样本”文件夹中包含示例 2DGL_DrawJPGScaled.c，它显示了如何绘制缩放的 JPEG 文件。

GUI_JPEG_DrawScaledEx()

描述

在当前窗口中的指定位置通过缩放比例绘制不必加载到存储器的 jpeg 文件。

原型

```
int  GUI_JPEG_DrawScaledEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                           int x0, int y0, int Num, int Denom);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。
Num	用于缩放比例的分子。
Denom	用于缩放比例的分母。

返回值

0 表示成功，非 0 表示该函数失败。（当前的实施始终返回 0）

其他信息

该函数通过使用给定的分子和分母构建分数来缩放图像。例如，如果图像应缩减至 2/3 大小，则 Num 应为 2， Denom 应为 3。
更多详细信息，请参阅 “GUI_JPEG_DrawEx()”（第 150 页）。
“样本”文件夹中包含示例 2DGL_DrawJPGScaled.c，它显示了如何使用该函数。

GUI_JPEG_GetInfo()

描述

利用已加载到存储器的 jpeg 文件的相关信息填充 GUI_JPEG_INFO 结构。

原型

```
int GUI_JPEG_GetInfo(const void * pFileData,
                     int DataSize,
                     GUI_JPEG_INFO * pInfo);
```

参数	描述
pFileData	指向 jpeg 文件所在的存储器区域的起始位置。
DataSize	Jpeg 文件的字节数。
pInfo	指向该函数要填充的 GUI_JPEG_INFO 结构。

返回值

0 表示成功，非 0 表示该函数失败。

GUI_JPEG_INFO 的元素

数据类型	元素	描述
int	XSize	图像的 X 像素大小。
int	YSize	图像的 Y 像素大小。

其他信息

“样本”文件夹中包含示例 2DGL_DrawJPG.c，它显示了如何使用该函数。

GUI_JPEG_GetInfoEx()

描述

利用不必加载到存储器的 jpeg 文件的相关信息填充 GUI_JPEG_INFO 结构。

原型

```
int GUI_JPEG_GetInfoEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                       GUI_JPEG_INFO     * pInfo);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
pInfo	指向该函数要填充的 GUI_JPEG_INFO 结构。

返回值

0 表示成功，非 0 表示该函数失败。

其他信息

有关该函数以及参数 pfGetData 和 p 的更多详细信息，请参阅 “GUI_JPEG_GetInfo()”（第 153 页）和 “GUI_JPEG_DrawEx()”（第 150 页）。
“样本”文件夹中包含示例 2DGL_DrawJPGScaled.c，它显示了如何使用该函数。

8.3 GIF 文件支持

20 世纪 80 年代，CompuServe Information Service 开发出了 GIF 文件格式（图形交换格式）。它设计用于跨数据网络传输图像。
GIF 标准支持隔行扫描、透明、应用定义数据、动画以及原始文本渲染。 emWin 将忽略原始文本或应用特定数据等不受支持的数据。
GIF 文件使用 LZW (Lempel-Zif-Welch) 文件压缩方法来压缩图像数据。这种压缩方法运行起来不会丢失数据。输出图像与输入图像完全相同。

8.3.1 将 GIF 文件转换为 C 源

某些情况下，将 GIF 文件作为 C 文件添加到项目中非常有用。对此，可完全按照前面介绍的“JPEG 文件支持”下的相同方式来执行。

8.3.2 显示 GIF 文件

图形库首先对图形信息进行解码。如果必须绘制图像，解码流程将花费相当长的时间。如果在窗口管理器经常调用的 callback 例程中使用 GIF 文件，则解码流程可能花费相当长的时间。通过使用存储设备可缩短计算时间。最好的方法是先将图像绘制到存储设备中。在这种情况下，将只进行一次解压缩。更多关于存储设备的信息，请参见 “存储设备”（第 247 页）。

8.3.3 存储器使用

emWin 的 GIF 解压缩例程大约需要 16 千字节动态分配的 RAM 进行解压缩。绘制图像后，将释放用于解压缩的 RAM。

8.3.4 GIF 文件 API

下表按字母顺序列出了可用的 GIF 文件相关的例程。以下为详细描述：

例程	描述
GUI_GIF_Draw()	绘制已加载到存储器的 GIF 文件的第一个图像。
GUI_GIF_DrawEx()	绘制无需加载到存储器的 GIF 文件的第一个图像。
GUI_GIF_DrawSub()	绘制已加载到存储器的 GIF 文件的给定子图像。
GUI_GIF_DrawSubEx()	绘制无需加载到存储器的 GIF 文件的给定子图像。
GUI_GIF_DrawSubScaled()	绘制已加载到存储器的带比例的 GIF 文件的给定子图像。
GUI_GIF_DrawSubScaledEx()	绘制无需加载到存储器的带比例的 GIF 文件的给定子图像。
GUI_GIF_GetComment()	返回已加载到存储器的 GIF 文件的给定注释。
GUI_GIF_GetCommentEx()	返回无需加载到存储器的 GIF 文件的给定注释。
GUI_GIF_GetImageInfo()	返回已加载到存储器的 GIF 文件给定子图像的相关信息。
GUI_GIF_GetImageInfoEx()	返回无需加载到存储器的 GIF 文件给定子图像的相关信息。
GUI_GIF_GetInfo()	返回已加载到存储器的 GIF 文件的相关信息。
GUI_GIF_GetInfoEx()	返回无需加载到存储器的 GIF 文件的相关信息。
GUI_GIF_GetXSize()	返回加载到存储器的位图的 X 大小。
GUI_GIF_GetXSizeEx()	返回无需加载到存储器的位图的 X 大小。
GUI_GIF_GetYSize()	返回加载到存储器的位图的 Y 大小。
GUI_GIF_GetYSizeEx()	返回无需加载到存储器的位图的 Y 大小。

GUI_GIF_Draw()

描述

在当前窗口中的指定位置绘制已加载到存储器的 gif 文件的第一个图像。

原型

```
int GUI_GIF_Draw(const void * pGIF, U32 NumBytes, int x0, int y0);
```

参数	描述
pGIF	指向 gif 文件所在的存储器区域的起始位置。
NumBytes	Gif 文件的字节数。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。

返回值

0 表示成功， != 0 表示错误。

其他信息

如果该文件包含多个图像，此函数将仅显示该文件的第一个图像。支持透明和隔行扫描的图像。

GUI_GIF_DrawEx()

描述

在当前窗口中的指定位置绘制不必加载到存储器的 gif 文件。

原型

```
int GUI_GIF_DrawEx(GUI_GET_DATA_FUNC * pfGetData, void * p, int x0, int y0);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。

返回值

0 表示成功，非 0 表示该函数失败。

其他信息

如果存储器没有足够的 RAM 加载整个文件，可使用此函数来绘制 gif 文件。然后，库会调用参数 pfGetData 所指的函数来读取数据。
GetData 函数应返回可用的字节数。这可能小于或等于请求的字节数。该函数至少需要返回 1 个新字节。

GUI_GIF_DrawSub()

描述

在当前窗口中的指定位置绘制已加载到存储器的 gif 文件的给定子图像。

原型

```
int GUI_GIF_DrawSub(const void * pGIF, U32 NumBytes,
                    int x0, int y0, int Index);
```

参数	描述
pGIF	指向 gif 文件所在的存储器区域的起始位置。
NumBytes	Gif 文件的字节数。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。
Index	要显示的子图像基于 0 的索引。

返回值

0 表示成功， != 0 表示错误。

其他信息

该函数管理当前图像和上一图像之间的背景像素。例如，如果子图像 #3 应按 x20/y20 偏移量绘制，大小为 w10/h10，而上一个子图像在 x15/y15 位置以 w20/h20 大小显示，且需要重新绘制背景，该函数将使用背景颜色填充图像之间的像素。
“样本”文件夹的 2DGL_DrawGIF.c 文件显示了如何使用该函数。

GUI_GIF_DrawSubEx()

描述

在当前窗口中的指定位置绘制不必加载到存储器的 gif 文件的给定子图像。

原型

```
int GUI_GIF_DrawSubEx(GUI_GET_DATA_FUNC * pfGetData,
                      void * p, int x0, int  y0, int Index);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。
Index	要显示的子图像基于 0 的索引。

返回值

0 表示成功，非 0 表示该函数失败。

其他信息

如果存储器没有足够的 RAM 加载整个文件，可使用此函数来绘制 gif 图像。图形用户界面 (GUI) 库将调用参数 pfGetData 所指向的函数来读取数据。
更多详细信息，请参阅 “GUI_GIF_DrawEx()”（第 156 页）。

GUI_GIF_DrawSubScaled()

描述

在当前窗口中的指定位置绘制已加载到存储器的 gif 文件的给定子图像。

原型

```
int GUI_GIF_DrawSubScaled(const void * pGIF, U32 NumBytes, int x0, int y0,
                          int Index, int Num, int Denom);
```

参数	描述
pGif	指向 gif 文件所在的存储器区域的起始位置。
NumBytes	Gif 文件的字节数。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。
Index	要显示的子图像基于 0 的索引。
Num	用于缩放比例的分子。
Denom	用于缩放比例的分母。

返回值

0 表示成功，非 0 表示该函数失败。

其他信息

该函数通过使用给定的分子和分母构建分数来缩放图像。例如，如果图像应缩减至 2/3 大小，则 Num 应为 2， Denom 应为 3。

GUI_GIF_DrawSubScaledEx()

描述

在当前窗口中的指定位置通过缩放比例绘制已加载到存储器的 gif 文件的给定子图像。

原型

```
int GUI_GIF_DrawSubScaledEx(GUI_GET_DATA_FUNC * pfGetData,
                            void * p,     int x0,  int y0,
                            int    Index, int Num, int Denom);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。
Index	要显示的子图像基于 0 的索引。
Num	用于缩放比例的分子。
Denom	用于缩放比例的分母。

返回值

0 表示成功，非 0 表示该函数失败。

其他信息

该函数通过使用给定的分子和分母构建分数来缩放图像。例如，如果图像应缩减至 2/3 大小，则 Num 应为 2， Denom 应为 3。

GUI_GIF_GetComment()

描述

返回已加载到存储器的 GIF 图像的给定注释。

原型

```
int GUI_GIF_GetComment(const void * pGIF, U32 NumBytes,
                       U8 * pBuffer, int MaxSize, int Index);
```

参数	描述
pGIF	指向 gif 文件所在的存储器区域的起始位置。
NumBytes	Gif 文件的字节数。
pBuffer	指向填充了注释的缓冲区。
MaxSize	缓冲区的大小。
Index	要返回的注释基于 0 的索引。

返回值

0 表示成功， != 0 表示错误。

其他信息

GIF 文件可以包含 1 个或多个注释。该函数会将注释复制到给定的缓冲区。如果注释大小大于给定缓冲区，则只会复制适合该缓冲区的字节数。
“样本”文件夹的 2DGL_DrawGIF.c 文件显示了如何使用该函数。

GUI_GIF_GetCommentEx()

描述

返回不必加载到存储器的 GIF 图像的给定注释。

原型

```
int GUI_GIF_GetCommentEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                         U8 * pBuffer, int MaxSize, int Index);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
pBuffer	指向填充了注释的缓冲区。
MaxSize	缓冲区的大小。
Index	要返回的注释基于 0 的索引。

返回值

0 表示成功， != 0 表示错误。

其他信息

更多详细信息，请参阅 “GUI_GIF_GetComment()”（第 158 页）。

GUI_GIF_GetImageInfo()

描述

返回已加载到存储器的 GIF 文件给定子图像的相关信息。

原型

```
int GUI_GIF_GetImageInfo(const void * pGIF, U32 NumBytes,
                         GUI_GIF_IMAGE_INFO * pInfo, int Index);
```

参数	描述
pGIF	指向 gif 文件所在的存储器区域的起始位置。
NumBytes	Gif 文件的字节数。
pInfo	指向该函数要填充的 GUI_GIF_IMAGE_INFO 结构。
Index	子图像基于 0 的索引。

返回值

0 表示成功， != 0 表示错误。

GUI_GIF_IMAGE_INFO 的元素

数据类型	元素	描述
int	xPos	上次所绘制图像的 X 位置。
int	yPos	上次所绘制图像的 Y 位置。
int	xSize	上次所绘制图像的 X 大小。
int	ySize	上次所绘制图像的 Y 大小。
int	Delay	图像应在电影中显示的时间 （1/100 秒）。

其他信息

如果需要将图像作为电影显示，则应使用此函数获取子图像可见的时间以及下一个子图像显示的时间。如果 Delay 元素为 0，则图像可见的时间应为 1/10 秒。

GUI_GIF_GetImageInfoEx()

描述

返回无需加载到存储器的 GIF 文件给定子图像的相关信息。

原型

```
int GUI_GIF_GetImageInfoEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                           GUI_GIF_IMAGE_INFO * pInfo, int Index);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
pInfo	指向该函数要填充的 GUI_GIF_IMAGE_INFO 结构。
Index	子图像基于 0 的索引。

返回值

0 表示成功， != 0 表示错误。

其他信息

更多详细信息，请参阅 “GUI_GIF_GetImageInfo()”（第 159 页）。

GUI_GIF_GetInfo()

描述

返回已加载到存储器的给定 GIF 文件的子图像的信息结构，包含子图像的大小和数量等相关信息。

原型

```
int GUI_GIF_GetInfo(const void * pGIF, U32 NumBytes, GUI_GIF_INFO * pInfo);
```

参数	描述
pGIF	指向 gif 文件所在的存储器区域的起始位置。
NumBytes	Gif 文件的字节数。
pInfo	指向此函数要填充的 GUI_GIF_INFO 结构。

返回值

0 表示成功， != 0 表示错误。

GUI_GIF_INFO 的元素

数据类型	元素	描述
int	XSize	图像的 X 像素大小。
int	YSize	图像的 Y 像素大小。
int	NumImages	文件中的子图像数。

GUI_GIF_GetInfoEx()

描述

返回无需加载到存储器的给定 GIF 文件的子图像的信息结构，包含子图像的大小和数量等相关信息。

原型

```
int GUI_GIF_GetInfoEx(GUI_GET_DATA_FUNC * pfGetData, void * p,
                      GUI_GIF_INFO * pInfo);;
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
pInfo	指向此函数要填充的 GUI_GIF_INFO 结构。

返回值

0 表示成功， != 0 表示错误。

GUI_GIF_INFO 的元素

数据类型	元素	描述
int	XSize	图像的 X 像素大小。
int	YSize	图像的 Y 像素大小。
int	NumImages	文件中的子图像数。

GUI_GIF_GetXSize()

描述

返回已加载到存储器的指定 GIF 图像的 X 大小。

原型

```
int GUI_GIF_GetXSize(const void * pGIF);
```

参数	描述
pGIF	指向 gif 文件所在的存储器区域的起始位置。

返回值

GIF 图像的 X 大小。

GUI_GIF_GetXSizeEx()

描述

返回无需加载到存储器的 GIF 图像的 X 大小。

原型

```
int GUI_GIF_GetXSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。

返回值

GIF 图像的 X 大小。

GUI_GIF_GetYSize()

描述

返回已加载到存储器的指定 GIF 图像的 Y 大小。

原型

```
int GUI_GIF_GetYSize(const void * pGIF);;
```

参数	描述
pGIF	指向 bmp 文件所在的存储器区域的起始位置。

返回值

GIF 图像的 Y 大小。

GUI_GIF_GetYSizeEx()

描述

返回无需加载到存储器的 GIF 图像的 Y 大小。

原型

```
int GUI_GIF_GetYSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。

返回值

GIF 图像的 Y 大小。

8.4 PNG 文件支持

PNG（可移植的网络图形）格式是一种图像格式，它利用非专利的数据压缩方法提供无损的数据压缩和 Alpha 混合。 PNG 1.0 版规范于 1996 年发布。到 2003 年末， PNG 成为国际标准 (ISO/IEC 15948)。
emWin 对 PNG 支持的实施基于来自 Glenn Randers-Pehrson、Guy Eric Schalnat 和 Andreas Dilger 的 “libpng” 库，该库可在 www.libpng.org 下免费获得。 emWin 对该库的使用符合 GUI\PNG\png.h 中的版权通知，通知中允许使用该库，而没有任何限制。
emWin 的 PNG 库可从 www.segger.com/link/emwin_png.zip 获得。

8.4.1 将 PNG 文件转换为 C 源

某些情况下，将 PNG 文件作为 C 文件添加到项目中非常有用。对此，可完全按照前面介绍的 “JPEG 文件支持” 下的相同方式来执行。此外，位图转换器能够加载 PNG 文件并将它们转换为 C 位图文件。

8.4.2 显示 PNG 文件

图形库首先对图形信息进行解码。如果必须绘制图像，解码流程将花费相当长的时间。如果在窗口管理器经常调用的 callback 例程中使用 PNG 文件，则解码流程可能花费相当长的时间。通过使用存储设备可缩短计算时间。最好的方法是先将图像绘制到存储设备中。在这种情况下，将只进行一次解压缩。更多关于存储设备的信息，请参见 “存储设备”（第 247 页）。

8.4.3 存储器使用

PNG 解压缩大约需要 21 Kb RAM 用于与图像大小无关的解压缩和依赖大小的字节量。RAM 要求可按以下方式计算：

大约 RAM 要求 = (X-Size + 1)* Y 大小 * 4 + 21Kbytes

8.4.4 PNG 文件 API

下表按字母顺序列出了可用的 PNG 文件相关的例程。以下为详细描述：#

例程	描述
GUI_PNG_Draw()	绘制已加载到存储器的 PNG 文件。
GUI_PNG_DrawEx()	绘制无需加载到存储器的 PNG 文件。
GUI_PNG_GetXSize()	返回加载到存储器的位图的 X 大小。
GUI_PNG_GetXSizeEx()	返回无需加载到存储器的位图的 X 大小。
GUI_PNG_GetYSize()	返回加载到存储器的位图的 Y 大小。
GUI_PNG_GetYSizeEx()	返回无需加载到存储器的位图的 Y 大小。

GUI_PNG_Draw()

描述

在当前窗口中的指定位置绘制已加载到存储器的 png 文件。

原型

```
int GUI_PNG_Draw(const void * pFileData, int FileSize, int x0, int y0);
```

参数	描述
pFileData	指向 png 文件所在的存储器区域的起始位置。
FileSize	png 文件的字节数。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。

返回值

0 表示成功，非 0 表示该函数失败。（当前的实施始终返回 0）

其他信息

“样本”文件夹中包含示例 2DGL_DrawPNG.c，它显示了如何使用该函数。

GUI_PNG_DrawEx()

描述

在当前窗口中的指定位置绘制不必加载到存储器的 png 文件。

原型

```
int GUI_PNG_DrawEx(GUI_GET_DATA_FUNC * pfGetData, void * p, int x0, int y0);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。
x0	显示器中位图左上角的 X 位置。
y0	显示器中位图左上角的 Y 位置。

返回值

0 表示成功，非 0 表示该函数失败。

其他信息

如果存储器没有足够的 RAM 加载整个文件，可使用此函数来绘制 png 文件。然后，PNG 库会调用参数 pfGetData 所指的函数来读取数据。
GetData 函数应返回可用的字节数。这可能小于或等于请求的字节数。该函数至少需要返回 1 个新字节。请注意， PNG 库会在内部为整个图像分配缓冲区。使用此函数无法避免此操作。

GUI_PNG_GetXSize()

描述

返回已加载到存储器的指定 PNG 图像的 X 大小。

原型

```
int GUI_PNG_GetXSize(const void * pFileData, int FileSize);
```

参数	描述
pFileData	指向 png 文件所在的存储器区域的起始位置。
FileSize	文件的大小（字节）。

返回值

PNG 图像的 X 大小。

GUI_PNG_GetXSizeEx()

描述

返回无需加载到存储器的 PNG 图像的 X 大小。

原型

```
int GUI_PNG_GetXSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。

返回值

PNG 图像的 X 大小。

GUI_PNG_GetYSize()

描述

返回已加载到存储器的指定 PNG 图像的 Y 大小。

原型

```
int GUI_PNG_GetYSize(const void * pFileData, int FileSize);
```

参数	描述
pFileData	指向 png 文件所在的存储器区域的起始位置。
FileSize	文件的大小（字节）。

返回值

PNG 图像的 Y 大小。

GUI_PNG_GetYSizeEx()

描述

返回无需加载到存储器的 PNG 图像的 X 大小。

原型

```
int GUI_PNG_GetYSizeEx(GUI_GET_DATA_FUNC * pfGetData, void * p);
```

参数	描述
pfGetData	指向获取数据调用的函数。更多 GetData 函数的详细信息，请参阅“使用 ...Ex() 函数获取数据”（第 167 页）。
p	传递到 pfGetData 所指函数的 Void 指针。

返回值

PNG 图像的 Y 大小。

8.5 使用 ...Ex() 函数获取数据

同流位图一样，使用 BMP、 GIF、 JPEG 和 PNG 文件也能够运行，而不必将整个图像加载到 RAM。对于此种情况，可使用 ...Ex() 函数。所有这些函数的共同之处在于都使用 “GetData” 函数。请注意，根据使用 “GetData” 函数的实际任务不同，该函数的运行也稍有差异。请参见以下参数和示例表。

“GetData” 函数的原型

```
int GUI_GET_DATA_FUNC(void     * p,         const U8 * * ppData,
                      unsigned   NumBytes, U32          Off)
```

参数	描述
p	应用程序定义的 Void 指针。
ppData	BMP、 GIF 和 JPEG： 指向到 U8 的指针。“GetData” 函数必须将指针设置到请求数据所在的位置。 流位图和 PNG： 指向 U8。必须使用 “GetData” 函数填充指针指向的位置。
NumBytesReq	请求的字节数。
StartOfFile	如果此标记为 1，应将数据指针设置到数据流的起始位置。

示例 （BMP、 GIF 和 JPEG）

下列代码节选显示了如何实施 “GetData” 函数以用于 BMP、 GIF 和 JPEG 数据：

```
static char _acBuffer[0x200];

int APP_GetData(void * p, const U8 * * ppData, unsigned NumBytesReq, U32 Off) {
  HANDLE * phFile;
  DWORD    NumBytesRead;

  phFile = (HANDLE *)p;
  //
  // Check buffer size
  //
  if (NumBytesReq > sizeof(_acBuffer)) {
    NumBytesReq = sizeof(_acBuffer);
  }
  //
  // Set file pointer to the required position
  //
  SetFilePointer(*phFile, Off, 0, FILE_BEGIN);
  //
  // Read data into buffer
  //
  ReadFile(*phFile, _acBuffer, NumBytesReq, &NumBytesRead, NULL);
  //
  // Set data pointer to the beginning of the buffer
  //
  *ppData = _acBuffer;
  //
  // Return number of available bytes
  //
  return NumBytesRead;
}
```

示例 （PNG 和流位图）

以下代码节选显示了如何实施 “GetData”函数以用于 PNG 和流位图数据：

```
static char acBuffer[0x200];

int APP_GetData(void * p, const U8 * * ppData, unsigned NumBytesReq, U32 Off) {
  HANDLE * phFile;
  DWORD    NumBytesRead;
  U8     * pData;

  pData  = (U8 *)*ppData;
  phFile = (HANDLE *)p;
  //
  // Set file pointer to the required position
  //
  SetFilePointer(*phFile, Off, 0, FILE_BEGIN);
  //
  // Read data into buffer
  //
  ReadFile(*phFile, pData, NumBytesReq, &NumBytesRead, NULL);
  //
  // Return number of available bytes
  //
  return NumBytesRead;
}
```


第 9 章

字体

本章介绍 emWin 中字体支持的各种方法。emWin 随附的最常用的字体为 C 字体文件。所有文件均包含 ASCII 字符集，大部分文件还包含 ISO 8859-1 字符。实际上，您可能会发现这些字体足够支持您的应用。有关各种字体的详细信息，请参阅 “标准字体”（第 192 页）。
emWin 采用 8 位字符编译，最多允许 256 种不同的字符代码，其中前 32 中将保留作为控制字符。可用的字符取决于所选的字体。
为访问含 65536 种可能字符的完整的 Unicode 区域， emWin 支持 UTF8 解码，在 “外语支持”（第 801 页）中对此进行了介绍。

9.1 简介

字体支持的第一种方法是，可以使用字体定义中含每个字符的 1 bpp 像素信息位图的 C 文件。这种字体支持仅限于使用应用编译的字体。
随着时间推移，字体支持在字体质量、 ROM 要求、性能、可扩展性和运行时添加更多字体的能力方面进行了改进。
同时， emWin 字体包含抗锯齿、复合字符的图样 （例如泰语中的要求）、位于外部非可寻址介质中的字体以及 TrueType 支持。除 TrueType 字体 （矢量字体）格式外，所有其他字体均为位图字体。

9.2 字体类型

emWin 支持自行定义的内部类型不同的字体以及常用的 TrueType 字体。

等宽位图字体

等宽位图字体的每个字符大小相同。在比例字体中，每个字符有自己的宽度，而等宽字体的宽度只需定义一次。像素信息保存为 1 bpp，涵盖整个字符区域。

比例位图字体

比例位图字体的每个字符高度相同、宽度可能不同。像素信息保存为 1 bpp，涵盖整个字符区域。

抗锯齿字体，包含 2 bpp 抗锯齿信息

每个字符高度相同、宽度可能不同。像素信息保存为 2 bpp 抗锯齿信息，涵盖整个字符区域。

抗锯齿字体，包含 4 bpp 抗锯齿信息

每个字符高度相同、宽度可能不同。像素信息保存为 4 bpp 抗锯齿信息，涵盖整个字符区域。

扩展比例位图字体

扩展比例位图字体的每个字符拥有自己的高度和宽度。像素信息保存为 1 bpp，仅涵盖字形位图区域。

扩展比例位图字体，包含 2 bpp 抗锯齿信息

每个字符高度相同、宽度可能不同。像素信息保存为 2 bpp 抗锯齿信息，仅涵盖字形位图区域。

扩展比例位图字体，包含 4 bpp 抗锯齿信息

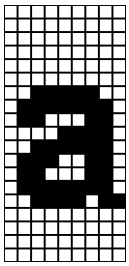
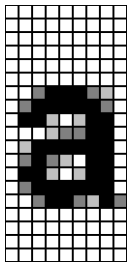
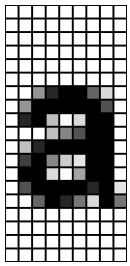
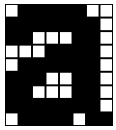
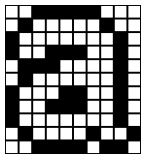
每个字符高度相同、宽度可能不同。像素信息保存为 4 bpp 抗锯齿信息，仅涵盖字形位图区域。

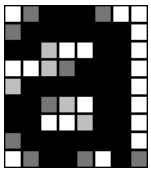
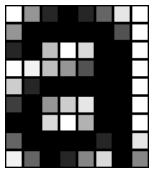
扩展比例位图字体，带边框

在某些情况下 （例如编译时背景颜色未知），可以使用带边框的字体。带边框的字体始终在透明模式下绘制，与当前设置无关。字符像素按当前所选的前景颜色绘制，边框按背景颜色绘制。前景颜色和背景颜色之间明显的对比可以确保能够阅读文本，而与背景无关。
请注意，这种类型的字体不适用于复合字符，例如泰语。另外，也不适用于 Arabic 字体。以下图片显示了照片前某些带边框的文本：

字体类型表

下表显示了各字体类型之间的差异。图片仅显示字体文件中保存的像素信息：

比例位图字体	比例位图字体，AA2	比例位图字体，AA2	扩展比例位图字体	扩展比例位图字体，带边框
				

扩展比例位图字体， AA2	扩展比例位图字体， AA4
	

TrueType 矢量字体

emWin 的 TrueType 字体支持表示，支持本章后面部分介绍的 TrueType 字体文件格式。

9.3 字体格式

下面解释了所支持的字体格式之间的差异、何时使用它们以及使用它们的要求。

9.3.1 C 文件格式

这是最常见的字体使用方式。使用 C 文件格式的字体时，我们建议作为库模块来编译所有可用的字体和链接它们，或将所有字体对象文件置于您可以链接到应用的库中。这样，可确保实际只链接了您的应用所需的字体。可以使用字体转换器 （将在单独手册中介绍）来创建其他字体。

何时使用

在编译时字体已知以及对于字体数据拥有足够的可寻址内存的情况下，可以使用这种格式。

要求

为了能够在应用中使用字体 C 文件，必须满足以下要求：

- 字体文件为与 emWin 兼容的 C 文件、对象文件或库格式。
- 字体文件与您的应用链接。
- 应用中包含字体声明。

格式说明

字体 C 文件首先包含字体所含全部字符的像素信息。之后，是有关每个字符大小信息的字符信息表。此表之后是字体文件中所含各个字符相邻区域的范围信息结构，而每个结构均指向下一个结构。请注意，如果使用许多单独的字符，这种方法可以大幅扩展字体文件。在范围信息结构之后，是 GUI_FONT 结构，其中包含字体的类型、像素大小等主要信息。

9.3.2 系统独立字体 (SIF) 格式

系统独立字体是包含字体信息的二进制数据块。可以使用字体转换器来创建系统独立字体。此工具不在基本封装之列。本章后面将对此进行简单的介绍。

何时使用

在编译时字体未知以及对于字体数据拥有足够的可寻址内存的情况下，可以使用这种格式。

要求

为了能够在应用中使用 SIF 字体文件，需要在可寻址存储器 （ROM 或 RAM）中保存整个文件。

格式说明

SIF 文件的结构与 C 文件大致相同。它在二进制格式中包含相同的信息。文件要素的顺序反之亦然：常规字体信息，之后是范围信息结构、字符信息表和所有字符的最小像素信息。

9.3.3 外部位图字体 (XBF) 格式

同 SIF 字体一样，XBF 字体也是包含字体信息的二进制数据块，并且可以使用字体转换器来创建 XBF 文件。字体转换器不在 emWin 基本封装之列。有关如何创建外部二进制字体的详细信息，请参阅字体转换器文档。
与其他字体不同的是，XBF 字体在使用时不必位于存储器中，而 emWin 的所有其他字体均需要完全驻存在存储器中。 XBF 字体文件在使用时可保存在任意外部介质中。通过 “GetData” 回调函数可以进行数据访问，而所有其他字体都需要驻存在可寻址存储器 （RAM 或 ROM）中。 XBF 字体的优势在于，可以在内存很小的系统上使用非常大的字体。

何时使用

如果对于字体数据没有足够的可寻址内存，并且可以使用任何其他种类的外部介质来存储字体的情况下，可以使用这种格式。

要求

为了能够在应用中使用 XBF 字体，需要使用 “GetData” 回调函数来负责获取字体数据。

格式说明

这种格式总体上与 SIF 和 C 文件格式有所不同。首先，它包含小块常规字体信息，其中包含最低和最高的字符代码。之后，是包含最低和最高字符代码之间每个字符的偏移和数据大小信息的访问表。如果不存在字符，则相应字符的此信息为零。访问表之后是所有字符的字符信息，包含像素数据和字符大小信息。

9.3.4 TrueType 字体 (TTF) 格式

TrueType 是 Apple Computer 开发的轮廓字体标准。它为字体开发人员提供对在各种字体高度下字体显示方式的高度控制。与位图字体（基于每个字符的位图）不同，TrueType 字体基于矢量图形。矢量表示的优势在于无损的可扩展性。
这意味着，每个字符在绘制前需要光栅化为位图。为避免每次绘制字符时都进行光栅化，通常用字体引擎缓存位图数据。这要求 CPU 速度快、 RAM 足够。
发货时不含 emWin TTF 包。该项内容可在 www.segger.com/link/emwin_freetype.zip 下免费获得。

许可

emWin对TTF支持的实施基于来自David Turner、Robert Wilhelm和Werner Lembergr的FreeType字体库，该库可在 www.freetype.org 下免费获得。emWin 对该库的使用符合 GUI\TrueType\FTL.txt 下的 FreeType 许可。 emWin 对该库进行了少许改编，添加了带有 GUI 函数的 “粘贴”层。

何时使用

如果在运行时需要对字体进行缩放，可使用这种格式。

要求

- CPU：TTF 支持仅适用于 32 位 CPU。我们对 32 位 CPU 的定义为：`sizeof(int) = 4`。
- ROM：TTF 引擎的 ROM 要求大约为 250 K。确切大小取决于 CPU、编译器以及编译器的优化水平。
- RAM：该库的 RAM 要求主要取决于使用的字体。TTF 引擎的基本 RAM 要求大约为 50 K。在使用 `GUI_TTF_CreateFont()` 创建 GUI 字体时，字体引擎会加载生成字符所需的 TTF 文件中定义的所有字体表。不同字体之间的表大小有所差异。创建字体额外要求的 RAM 量可能介于几个 KB 到 1 MB 以上之间。一般字体需要 80-300 kb。取决于使用的字体文件需要多少 RAM。至少，TTF 引擎需要位图缓存。默认情况下，引擎使用 200 K 的缓存。足够大多数应用使用。TTF 引擎通过非 emWin 函数 `malloc()` 和 `free()` 分配内存。使用 TTF 引擎之前，必须确保能够运行这些函数。

格式说明

有关 TTF 格式的详细信息，请参阅 www.apple.com 下可用的信息。

9.4 将 TTF 文件转换为 C 源

某些情况下，将 TTF 文件作为 “C” 文件添加到项目中非常有用，例如在没有文件系统可用时。使用 emWin 随附的工具 `Bin2C.exe` 可完成此任务。在 “工具”子文件夹中可找到该工具。它可以将给定的二进制文件 （在本例中为 TTF 文件）转换为 “C” 文件。

9.5 声明自定义字体

声明自定义字体原型最推荐的方法是将它们置于应用定义的头文件中。使用这些字体的每个应用源文件中均应包含此内容。如以下示例所示：

```
#include "GUI.h"

extern GUI_CONST_STORAGE GUI_FONT GUI_FontApp1;
extern GUI_CONST_STORAGE GUI_FONT GUI_FontApp2;
```

请注意，如果这些字体要用于 BUTTON_FONT_DEFAULT 或类似的 emWin 配置宏，这种声明原型不起作用。在这种情况下，字体需要在配置文件GUIConf.h中声明。这种情况下的声明如以下示例所示：

```
typedef struct GUI_FONT GUI_FONT;

extern const GUI_FONT GUI_FontApp1;

#define BUTTON_FONT_DEFAULT &GUI_FontApp1
#define EDIT_FONT_DEFAULT   &GUI_FontApp1
```

由于前面emWin包含GUIConf.h的位置未定义GUI_FONT结构，所以需要类型定义符(typedef)。

9.6 选择字体

emWin提供不同的字体，始终选择其中之一。通过调用函数GUI_SetFont()或GUI_XXX_CreateFont()函数之一可更改此选项，它们将选择用于当前任务所有文本输出的字体。
如果您的应用没有选择任何字体，则将使用默认字体。默认字体在 GUIConf.h 中配置，且可以更改。您应确保默认字体是在应用中实际使用的字体，因为默认字体将链接到您的应用，因而占用 ROM 内存。

9.7 字体 API

下表按字母顺序列出了各自类别内可用的与字体相关的例程。有关详细描述，请参阅后面的章节。

例程	描述
C 文件相关的字体函数	
GUI_SetDefaultFont()	设置默认字体
GUI_SetFont()	设置当前字体
“SIF”文件相关的字体函数	
GUI_SIF_CreateFont()	通过将指针传递至系统独立字体数据创建和选择字体。
GUI_SIF_DeleteFont()	删除使用 GUI_SIF_CreateFont() 创建的字体
“TTF”文件相关的字体函数	
GUI_TTF_CreateFont()	基于 TTF 字体文件创建 GUI 字体。
GUI_TTF_DestroyCache()	解除 TTF 引擎的缓存。
GUI_TTF_Done()	释放 TTF 引擎动态分配的所有内存。
GUI_TTF_GetFamilyName()	返回字体的系列名称。
GUI_TTF_GetStyleName()	返回字体的样式名称。
GUI_TTF_SetCacheSize()	可用于设置 TTF 缓存的默认大小。
“XBF”文件相关的字体函数	
GUI_XBF_CreateFont()	通过将指针传递至负责从 XBF 字体文件获取数据的回调函数创建和选择字体。
GUI_XBF_DeleteFont()	删除使用 GUI_XBF_CreateFont() 创建的字体
常见字体相关的函数	
GUI_GetCharDistX()	返回当前字体中指定字符的宽度像素 （X 大小）。
GUI_GetFont()	返回当前选择的字体的指针。
GUI_GetFontDistY()	返回当前字体的 Y 间距。
GUI_GetFontInfo()	返回包含字体信息的结构。
GUI_GetFontSizeY()	返回当前字体的高度像素 （Y 大小）。
GUI_GetLeadingBlankCols()	返回给定字符的前导空格像素列数。
GUI_GetStringDistX()	返回使用当前字体的文本的 X 大小。
GUI_GetTextExtend()	评估使用当前字体的文本的大小。
GUI_GetTrailingBlankCols()	返回给定字符的后导空格像素列数。
GUI_GetYDistOfFont()	返回特定字体的 Y 间距。
GUI_GetYSizeOfFont()	返回特定字体的 Y 大小。
GUI_IsInFont()	评估特定字体中是否存在指定的字符。
GUI_SetDefaultFont()	设置 GUI_Init() 之后使用的默认字体。

9.8 C 文件相关的字体函数

GUI_SetDefaultFont()

描述

设置文本输出默认情况下使用的字体。

原型

```
void GUI_SetDefaultFont(const GUI_FONT GUI_UNI_PTR * pFont);
```

参数	描述
pFont	指向默认情况下选择的字体。

其他信息

此函数旨在用于 GUI_X_Config()。定义 GUI_DEFAULT_FONT 不再是必需的。如果既没有定义 GUI_DEFAULT_FONT，也没有调用 GUI_SetDefaultFont，则会将 GUI_Font6x8 设置为默认字体。如果不使用任何 emWin 字体，则 GUI_DEFAULT_FONT 必需定义为 NULL，并需要使用此函数将自定义字体设为默认。

GUI_SetFont()

描述

设置文本输出使用的字体。

原型

```
const GUI_FONT * GUI_SetFont(const GUI_FONT * pNewFont);
```

参数	描述
pFont	指向要选择和使用的字体。

返回值

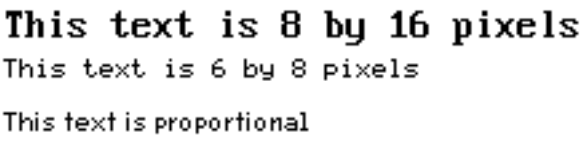
返回之前选择的字体的指针以便进行缓冲。

示例

使用 3 种不同的大小显示示例文本，然后恢复之前的字体：

```
const GUI_FONT GUI_FLASH * OldFont;
OldFont = GUI_SetFont(&GUI_Font8x16);                // Buffer old font
GUI_DispStringAt("This text is 8 by 16 pixels",0,0);
GUI_SetFont(&GUI_Font6x8);
GUI_DispStringAt("This text is 6 by 8 pixels",0,20);
GUI_SetFont(&GUI_Font8);
GUI_DispStringAt("This text is proportional",0,40);
GUI_SetFont(OldFont);                                 // Restore old font
```

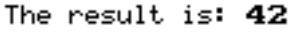
上述示例的屏幕截图：



使用不同的字体显示文本和值：

```
GUI_SetFont(&GUI_Font6x8);
GUI_DispString("The result is:"); // Disp text
GUI_SetFont(&GUI_Font8x8);
GUI_DispDec(42,2);                // Disp value
```

上述示例的屏幕截图：



9.9 “SIF”文件相关的字体函数

GUI_SIF_CreateFont()

描述

通过将指针传递至系统独立字体数据设置使用的字体。

原型

```
void GUI_SIF_CreateFont(void                * pFontData,
                        GUI_FONT            * pFont,
                        const GUI_SIF_TYPE * pFontType);
```

参数	描述
pFontData	指向系统独立字体数据。
pFont	指向 RAM 中由该函数填充的 GUI_FONT 结构。
pFontType	请参见下表。

元素 pFontType 的允许值	
GUI_SIF_TYPE_PROP	在参数 pFont 指向比例字体时使用。
GUI_SIF_TYPE_PROP_EXT	在参数 pFont 指向扩展比例字体时使用。
GUI_SIF_TYPE_PROP_FRM	在参数 pFont 指向扩展带边框比例字体时使用。
GUI_SIF_TYPE_PROP_AA2	在参数 pFont 指向使用 2 bpp 抗锯齿的比例字体时使用。
GUI_SIF_TYPE_PROP_AA4	在参数 pFont 指向使用 4 bpp 抗锯齿的比例字体时使用。
GUI_SIF_TYPE_PROP_AA2_EXT	在参数 pFont 指向使用 2 bpp 抗锯齿的扩展比例字体时使用。
GUI_SIF_TYPE_PROP_AA4_EXT	在参数 pFont 指向使用 4 bpp 抗锯齿的扩展比例字体时使用。

其他信息

与必须编译和链接应用程序的 emWin 标准字体不同，系统独立字体 (SIF) 是包含字体信息的二进制数据块。可以使用字体转换器来创建系统独立字体。此工具不在基本封装之列。本章后面将对此进行简单的介绍。有关如何创建系统独立字体的详细信息，请参阅字体转换器文档。
使用此函数时， emWin 需要用字体信息填充 GUI_FONT 结构。在参数 pFont 中，用户需要将指针传递至此结构。此结构的内容在字体使用期间必须保持有效。
该函数不知道应创建何种字体。要告知函数创建的字体类型，必须在参数 pFontType 中进行传递。这样，可避免链接不需要的代码。

示例

```
static GUI_FONT _Font; /* Font structure in RAM */

void main(void) {
  GUI_Init();
  GUI_SIF_CreateFont(_DownloadedFont, &_Font, GUI_SIF_TYPE_PROP);
  GUI_DispString("Hello World!");
  while (1) {
    GUI_Exec();
  }
}
```


GUI_SIF_DeleteFont()

描述

删除参数 pFont 指示的字体。

原型

```
void GUI_SIF_DeleteFont(GUI_FONT * pFont);
```

参数	描述
pFont	指向要删除的字体。

其他信息

使用 GUI_SIF_CreateFont() 创建的字体后，如果不再使用该字体，应将其删除。

示例

```
GUI_FONT _Font; /* Font structure in RAM */
GUI_SIF_CreateFont(_DownloadedFont, &_Font, GUI_SIF_TYPE_PROP);
/*
  Use the font
*/
GUI_SIF_DeleteFont(&_Font);
```

9.10 “TTF”文件相关的字体函数

emWin 对 TTF 文件支持的实施基于来自 David Turner、 Robert Wilhelm 和 Werner Lemberg 的 FreeType 字体库。更多详细信息，请参阅 “TrueType 字体 (TTF) 格式”（第 174 页）。

GUI_TTF_CreateFont()

描述

使用 TTF 字体文件创建和选择 emWin 字体。

原型

```
int GUI_TTF_CreateFont(GUI_FONT * pFont, GUI_TTF_CS * pCS);
```

参数	描述
pFont	指向 RAM 中由该函数填充的 GUI_FONT 结构。
pCS	指向包含创建参数的 GUI_TTF_CS 结构。

返回值

0 表示成功， 1 表示错误。

GUI_TTF_CS 的元素

数据类型	元素	描述
GUI_TTF_DATA *	pTTF	指向包含要使用的字体文件位置和大小的 GUI_TTF_DATA 结构。
PixelHeight	PixelHeight	新字体的像素高度。表示字形 “g” 和 “f” 之间周围矩形的高度。请注意，这并非两行文本之间的距离。换句话说，GUI_GetFontSizeY() 返回的值与此值并不完全相同。
FaceIndex	FaceIndex	有些字体文件可能包含多种字体风格。对于多种风格，此索引指定使用基于零的风格索引来创建字体。通常为 0。

GUI_TTF_DATA 的元素

数据类型	元素	描述
const void *	pData	指向可寻址存储器区域中的 TTF 字体文件。
NumBytes	NumBytes	文件的大小 （字节）。

其他信息

第一次使用该函数时，它会对 TTF 引擎和内部缓存系统进行初始化。如果缓存需要使用默认定义的其他值，则需要在首次调用此函数之前进行配置。有关如何配置缓存的详细信息，请参阅 “GUI_TTF_SetCacheSize()”（第 182 页）。

内部数据缓存管理创建字体和缓存位图数据的整个机制。字体风格在缓存中按参数 pTTF 和 FaceIndex 中给定的地址进行唯一标识，通常为 0。例如，如果应使用相同的字体文件来创建不同大小的字体，则参数 pTTF 应指向 GUI_TTF_DATA 结构的相同位置。

参数 PixelHeight 指定字形 “g” 和 “f” 之间周围矩形的高度。请注意 PixelHeight 的值与一行文本到另一行文本之间的偏移量不同。

示例

```
GUI_TTF_CS   Cs0, Cs1;
GUI_TTF_DATA Data;
GUI_FONT     Font0, Font1;
/* Set parameters for accessing the font file */
Data.pData      = aTTF;          /* Address */
Data.NumBytes   = sizeof(aTTF); /* Size */
/* Set creation parameters of first font */
Cs0.pTTF        = &Data;         /* Use address of GUI_TTF_DATA */
Cs0.PixelHeight = 24;            /* Pixel height */
Cs0.FaceIndex   = 0;             /* Initialize to 0 */
/* Set creation parameters of second font */
Cs1.pTTF        = &Data;         /* Use address of GUI_TTF_DATA */
Cs1.PixelHeight = 48;            /* Pixel height */
Cs1.FaceIndex   = 0;             /* Initialize to 0 */
/* Create 2 fonts */
GUI_TTF_CreateFont(&Font0, &Cs0);
GUI_TTF_CreateFont(&Font1, &Cs1);
/* Draw something using the fonts */
GUI_SetFont(&Font0);
GUI_DispString("Hello world\n");
GUI_SetFont(&Font1);
GUI_DispString("Hello world");
```


GUI_TTF_DestroyCache()

描述

此函数会释放 TTF 缓存系统分配的所有内存并解除该缓存。

原型

```
void GUI_TTF_DestroyCache(void);
```

其他信息

下次使用 GUI_TTF_CreateFont() 时， emWin 将自动创建和初始化新的缓存。

GUI_TTF_Done()

描述

此函数会释放 TTF 引擎及其内部缓存系统分配的所有内存。

原型

```
void GUI_TTF_Done(void);
```

其他信息

下次使用GUI_TTF_CreateFont()时，emWin将自动初始化TTF引擎，并创建和初始化新的缓存。

GUI_TTF_GetFamilyName()

描述

此函数将返回字体文件中定义的字体系列名称。

原型

```
int GUI_TTF_GetFamilyName(GUI_FONT * pFont, char * pBuffer, int NumBytes);
```

参数	描述
pFont	指向使用 GUI_TTF_CreateFont() 创建的 GUI_FONT 结构。
pBuffer	要填充系列名称的缓冲区。
NumBytes	缓冲区的大小 （字节）。

返回值

0 表示成功， 1 表示错误。

GUI_TTF_GetStyleName()

描述

此函数将返回字体文件中定义的样式名称 （粗体、常规等）。

原型

```
int GUI_TTF_GetStyleName(GUI_FONT * pFont, char * pBuffer, int NumBytes);
```

参数	描述
pFont	指向使用 GUI_TTF_CreateFont() 创建的 GUI_FONT 结构。
pBuffer	要填充样式名称的缓冲区。
NumBytes	缓冲区的大小 （字节）。

返回值

0 表示成功， 1 表示错误。

GUI_TTF_SetCacheSize()

描述

设置第一次调用 GUI_TTF_CreateFont() 时用于创建缓存的大小参数。

原型

```
void GUI_TTF_SetCacheSize(unsigned MaxFaces,
                          unsigned MaxSizes, U32 MaxBytes);
```

参数	描述
MaxFaces	缓存应能够同时处理的最大字体风格数。 0 选择默认值。
MaxSizes	缓存应能够同时处理的最大字体对象数。 0 选择默认值。
MaxBytes	位图缓存使用的最大字节数。 0 选择默认值。

其他信息

例如，如果应使用 3 种字体风格，每种有 2 种大小，则缓存应能够处理 6 种字体对象。
TTF 引擎使用的默认值是：2 种风格、 4 种大小对象和 200 K 位图数据缓存。

9.11 “XBF”文件相关的字体函数

GUI_XBF_CreateFont()

描述

通过将指针传递至负责从 XBF 字体文件获取数据的回调函数创建和选择字体。

原型

```
int GUI_XBF_CreateFont(GUI_FONT               * pFont,
                       GUI_XBF_DATA           * pXBF_Data,
                       const GUI_XBF_TYPE     * pFontType,
                       GUI_XBF_GET_DATA_FUNC  * pfGetData,
                       void                   * pVoid);
```

参数	描述
pFont	指向 RAM 中由该函数填充的 GUI_FONT 结构。
pXBF_Data	指向 RAM 中由该函数填充的 GUI_XBF_DATA 结构。
pFontType	请参见下表。
pfGetData	指向负责从字体文件获取数据的回调函数。请参见以下原型。
pVoid	传递到 “GetData” 回调函数的应用定义的指针。

元素 pFontType 的允许值	
GUI_XBF_TYPE_PROP	在参数 pFont 指向比例字体时使用。
GUI_XBF_TYPE_PROP_EXT	在参数 pFont 指向扩展比例字体时使用。
GUI_XBF_TYPE_PROP_FRM	在参数 pFont 指向扩展带边框比例字体时使用。
GUI_XBF_TYPE_PROP_AA2_EXT	在参数 pFont 指向使用 2 bpp 抗锯齿的扩展比例字体时使用。
GUI_XBF_TYPE_PROP_AA4_EXT	在参数 pFont 指向使用 4 bpp 抗锯齿的扩展比例字体时使用。

GUI_XBF_GET_DATA_FUNC – 描述

```
int GUI_XBF_GET_DATA_FUNC(U32    Off,   U16    NumBytes,
                          void * pVoid, void * pBuffer);
```

该函数应设置 pBuffer，指向请求数据所在的位置。例如，一个 “GetData” 函数，请参见 “使用 ...Ex() 函数获取数据”（第 167 页）。

其他信息

参数 pfGetData 应指向负责从字体文件获取数据的应用定义的回调函数。在请求字体数据时，参数 pVoid 将传递给回调函数。例如，可使用它将文件句柄传递给回调函数。
该函数要求指向 GUI_FONT 结构和 GUI_XBF_DATA 结构。该函数将使用字体信息填充这些结构。这些结构的内容在字体使用期间必须保持有效。该函数不知道必须要创建的 XBF 字体类型，因此必须使用参数 pFontType 告知该函数要创建的字体类型。这样，可避免链接不必要的代码。

默认情况下，每个字符的数据字节最大值限制为 200。这应该满足大部分要求。如果加载使用更多字节的字符，在调试版本中将生成警告。通过向文件 GUIConf.h 添加以下定义可增加默认值：

```
#define GUI_MAX_XBF_BYTES 500 // Sets the maximum number of bytes/chars to 500
```

示例

```
static  GUI_FONT     Font;     /* GUI_FONT structure in RAM */
static  GUI_XBF_DATA XBF_Data; /* GUI_XBF_DATA structure in RAM */

static int _cbGetData(U32 Off, U16 NumBytes, void * pVoid, void * pBuffer) {
  /* The pVoid pointer may be used to get a file handle */
  .../* TBD */
  /* Set file pointer to the given position */
  .../* TBD */
  /* Read the required number of bytes into the given buffer */
  .../* TBD */
}

void CreateXBF_Font(void * pVoid) {
  GUI_XBF_CreateFont(&Font,             /* Pointer to GUI_FONT structure */
                     &XBF_Data,         /* Pointer to GUI_XBF_DATA structure */
                     GUI_XBF_TYPE_PROP, /* Font type to be created */
                     _cbGetData,        /* Pointer to callback function */
                     pVoid);            /* Pointer to be passed to callback */

}
```

GUI_XBF_DeleteFont()

描述

删除参数 pFont 指示的 XBF 字体。

原型

```
void GUI_XBF_DeleteFont(GUI_FONT * pFont);
```

参数	描述
pFont	指向要删除的字体。

其他信息

使用 GUI_XBF_CreateFont() 创建的字体后，如果不再使用该字体，应将其删除。

9.12 常见字体相关的函数

GUI_GetFont()

描述

返回当前选择的字体的指针。

原型

```
const GUI_FONT * GUI_GetFont(void)
```

GUI_GetCharDistX()

描述

返回用于显示当前所选字体中指定字符的宽度像素 （X 大小）。

原型

```
int  GUI_GetCharDistX(U16 c);
```

参数	描述
c	计算宽度所基于的字符。

GUI_GetFontDistY()

描述

返回当前所选字体的 Y 间距。

原型

```
int GUI_GetFontDistY(void);
```

其他信息

Y 间距是两行相邻文本之间的垂直距离 （像素）。
返回的值是为当前所选字体输入的 YDist 值。
返回的值对比例字体和等宽字体都有效。

GUI_GetFontInfo()

描述

计算特定字体的 GUI_FONTINFO 结构的指针。

原型

```
void GUI_GetFontInfo(const GUI_FONT*pFont, GUI_FONTINFO* pfi);
```

参数	描述
pFont	指向该字体。
pfi	指向 GUI_FONTINFO 结构。

其他信息

GUI_FONTINFO 结构的定义如下所示：

```
typedef struct {
  U16 Flags;
} GUI_FONTINFO;
```

元素变量标记可采用以下值：

```
GUI_FONTINFO_FLAG_PROP
GUI_FONTINFO_FLAG_MONO
GUI_FONTINFO_FLAG_AA
GUI_FONTINFO_FLAG_AA2
GUI_FONTINFO_FLAG_AA4
```

示例

获取 GUI_Font6x8 的信息。计算后，FontInfo.Flags 包含标记 GUI_FONTINFO_FLAG_MONO。

```
GUI_FONTINFO FontInfo;
GUI_GetFontInfo(&GUI_Font6x8, &FontInfo);
```

GUI_GetFontSizeY()

描述

返回当前所选字体的高度像素 （Y 大小）。

原型

```
int GUI_GetFontSizeY(void);
```

其他信息

返回的值是为当前所选字体输入的 YSize 值。此值小于或等于 GUI_GetFontDistY() 函数返回的 Y 间距。
返回的值对比例字体和等宽字体都有效。

GUI_GetLeadingBlankCols()

描述

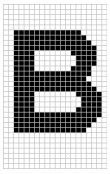
返回当前所选字体中给定字符的前导空格像素列数。

原型

```
int GUI_GetLeadingBlankCols(U16 c);
```

参数	描述
c	要使用的字符。

示例



以上屏幕截图中显示的字符 “B” 的结果应为 2。

GUI_GetStringDistX()

描述

返回用于显示当前所选字体中指定字符串的 X 大小。

原型

```
int  GUI_GetStringDistX(const char GUI_FAR *s);
```

参数	描述
s	指向该字符串。

GUI_GetTextExtend()

描述

计算使用当前字体的给定字符串的大小。

原型

```
void GUI_GetTextExtend(GUI_RECT* pRect, const char* s, int Len);
```

参数	描述
pRect	指向存储结果的 GUI_RECT-structure。
s	指向该字符串。
Len	字符串的字符数。

GUI_GetTrailingBlankCols()

描述

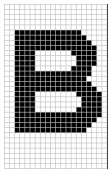
返回当前所选字体中给定字符的后导空格像素列数。

原型

```
int GUI_GetTrailingBlankCols(U16 c);
```

参数	描述
c	要使用的字符。

示例



以上屏幕截图中显示的字符 “B”的结果应为 1。

GUI_GetYDistOfFont()

描述

返回特定字体的 Y 间距。

原型

```
int  GUI_GetYDistOfFont(const GUI_FONT* pFont);
```

参数	描述
pFont	指向该字体。

其他信息

（见 GUI_GetFontDistY()）

GUI_GetYSizeOfFont()

描述

返回特定字体的 Y 大小。

原型

```
int  GUI_GetYSizeOfFont(const GUI_FONT* pFont);
```

参数	描述
pFont	指向该字体。

其他信息

（见 GUI_GetFontSizeY()）

GUI_IsInFont()

描述

评估特定字体中是否包含指定的字符。

原型

```
char GUI_IsInFont(const GUI_FONT * pFont, U16 c);
```

参数	描述
pFont	指向该字体。
c	要搜索的字符。

其他信息

如果指针 pFont 设置为 0，则使用当前所选的字体。

示例

评估字体 GUI_FontD32 中是否包含 “X”。

```
if (GUI_IsInFont(&GUI_FontD32, 'X') == 0) {
  GUI_DispString("GUI_FontD32 does not contains 'X'");
}
```

GUI_SetDefaultFont()

描述

设置 GUI_Init() 之后使用的默认字体。

原型

```
void GUI_SetDefaultFont(const GUI_FONT GUI_UNI_PTR * pFont);
```

参数	描述
pFont	指向要使用的字体。

9.13 字符集

9.13.1 ASCII

emWin 支持完整的 ASCII 字符集。以下是从 32 到 127 的 96 种字符：

十六进制	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2x		!		"#	$	%	&		'(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x		`a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	V	w	x	y	z	{	|	}	~	

遗憾的是，由于 ASCII 代表美国信息交换标准代码，所以它是针对美国需求而设计。它不包含欧洲语言中使用的任何特殊字符，例如 Ä, Ö, Ü, á, à, 以及其他字符。这些 “欧洲扩展” 的 ASCII 字符集没有唯一的标准，而是存在多个不同的标准。互联网以及大多数 Windows 程序使用的标准是 ISO 8859-1，即 ASCII 字符集的超集。

9.13.2 ISO 8859-1 西方拉丁字符集

emWin 支持 ISO 8859-1，它定义了以下所列的字符：

代码	描述	字符
160	不间断空格	
161	倒置的感叹号	¡
162	分符号	¢
163	英镑	£
164	一般货币符号	¤
165	日元符号	¥
166	断开的竖线	¦
167	章节符号	§
168	变元音（分音符）	¨
169	版权	©
170	阴性顺序	ª
171	左角引号，guillemotleft	«
172	否定符号	¬
173	软连字号	
174	注册商标	®
175	长音符	¯
176	角度符号	°
177	加号或减号	±
178	上标 2	²
179	上标 3	³
180	尖音符	´
181	微符号	µ
182	段落符号	¶
183	中间点	·
184	变音符	¸
185	上标 1	¹
186	masculine ordinal	º
187	右角引号，guillemot right	»
188	分数四分之一	¼
189	分数二分之一	½
190	分数四分之三	¾
191	倒置的问号	¿
192	大写 A，重音符	À
193	大写 A，尖音符	Á
194	大写 A，抑扬音符	Â
195	大写 A，鼻音化符号	Ã
196	大写 A，分音符或变元音符号	Ä
197	大写 A，环形	Å
198	大写 A，双元音（连字）	Æ
199	大写 C，变音符	Ç
200	大写 E，重音符	È
201	大写 A，尖音符	É
202	大写 E，抑扬音符	Ê
203	大写 E，分音符或变元音符号	Ë
204	大写 I，重音符	Ì
205	大写 I，尖音符	Í
206	大写 I，抑扬音符	Î
207	大写 I，分音符或变元音符号	Ï
208	Eth，冰岛语	Ð
209	N，鼻音化符号	Ñ
210	大写 O，重音符	Ò
211	大写 O，尖音符	Ó
212	大写 O，抑扬音符	Ô
213	大写 O，鼻音化符号	Õ
214	大写 O，分音符或变元音符号	Ö
215	乘号	×

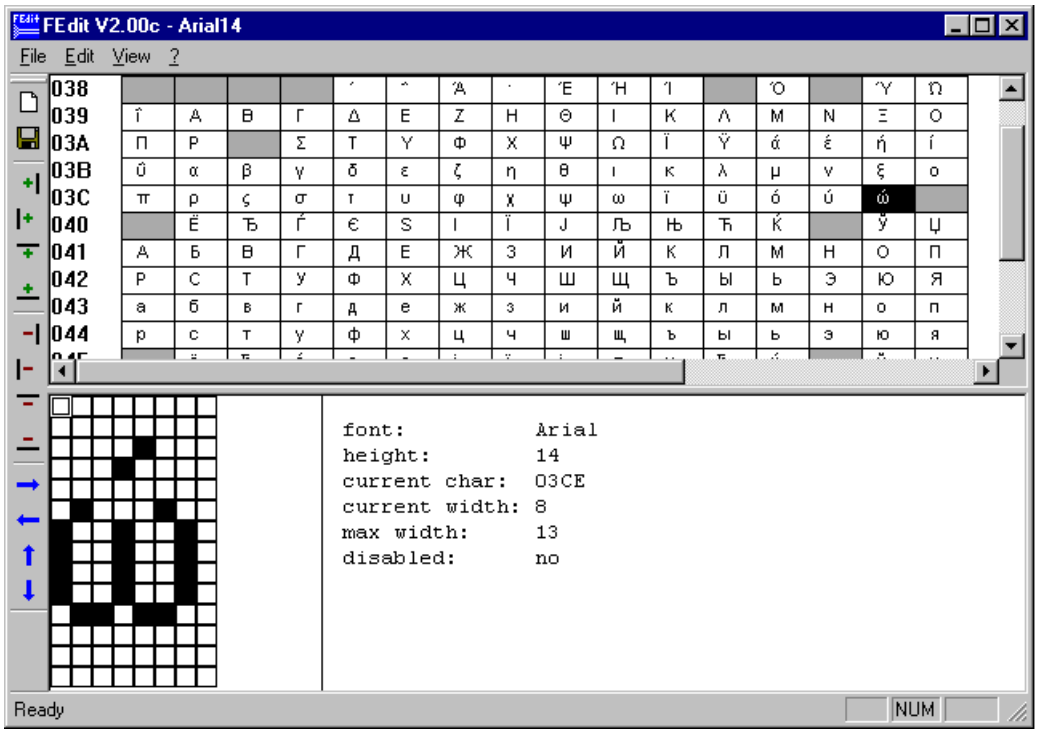
代码	描述	字符
216	大写 O，斜线	Ø
217	大写 U，重音符	Ù
218	大写 U，尖重音符	Ú
219	大写 U，抑扬音符	Û
220	大写 U，分音符或变元音符号	Ü
221	大写 Y，尖音符	Ý
222	THORN，冰岛语	Þ
223	急转的 s, 德语 （s-z 连字）	ß
224	小写 a，重音符	à
225	小写 a，尖音符	á
226	小写 a，抑扬音符	â
227	小写 a，鼻音化符号	ã
228	小写 a，分音符或变元音符号	ä
229	小写 a，环形	å
230	小写 ae 双元音 （连字）	æ
231	变音符	ç
232	小写 e，重音符	è
233	小写 e，尖音符	é
234	小写 e，抑扬音符	ê
235	小写 e，分音符或变元音符号	ë
236	小写 i，重音符	ì
237	小写 i，尖音符	í
238	小写 i，抑扬音符	î
239	小写 i，分音符或变元音符号	ï
240	小写 eth，冰岛语	ð
241	小写 n，鼻音化符号	ñ
242	小写 o，重音符	ò
243	小写 o，尖音符	ó
244	小写 o，抑扬音符	ô
245	小写 o，鼻音化符号	õ
246	小写 o，分音符或变元音符号	ö
247	除号	÷
248	小写 o，斜线	ø
249	小写 u，重音符	ù
250	小写 u，尖音符	ú
251	小写 u，抑扬音符	û
252	小写 u，分音符或变元音符号	ü
253	小写 y，尖音符	ý
254	小写 thorn，冰岛语	þ
255	小写 y，分音符或变元音符号	ÿ

9.13.3 Unicode

Unicode 是最终的字符解码方法。它是基于 ASCII 和 ISO 8859-1 国际标准。与 ASCII 不同的是，UNICODE 要求 16 位字符，因为所有字符都拥有自己的代码。当前，定义了 30,000 多种不同的字符。不过，并非所有字符图像在 emWin 中都进行了定义。用户可自行定义其他相关字符。请与 SEGGER Microcontroller GmbH & Co. KG 或您的经销商联系，我们可能已拥有您所需的字符集。

9.14 字体转换器

emWin 可以使用的字体必须作为 C 中的 GUI_FONT 结构进行定义。这些结构或它们所引用的字体数据可以相当大。手动生成这些字体非常耗时且效率低下。因此，我们建议使用字体转换器，它可以基于字体自动生成 C 文件。
字体转换器属于简单的 Windows 程序。您只需将安装的 Windows 字体加载到该程序，根据需要对其编辑，将其另存为 C 文件。然后，则可以编译 C 文件，允许在需求 emWin 的显示器中显示该字体。默认情况下，字符代码 - 0x1F 和 0x80 - 0x9F 被禁用。以下是加载了字体的字体转换器的屏幕截图示例：



字体转换器将在单独手册中介绍，通过 SEGGER Microcontroller GmbH & Co. KG () 可请求获得该文档。

9.14.1 添加字体

创建字体文件并将其链接到项目后，作为外部常量 GUI_FONT 对链接的字体进行声明，如以下示例所示：

示例

```
extern const GUI_FONT GUI_FontNew;

int main(void) {
  GUI_Init();
  GUI_Clear();
  GUI_SetFont(&GUI_FontNew);
  GUI_DispString("Hello world\n");
  return 0;
}
```

9.15 标准字体

emWin 随附一系列字体，可满足您的大部分需求。标准字体包包含不同大小和样式的等宽字体和比例字体。**等宽字体**指字符宽度固定的字体，其中所有字符的宽度像素相同。**比例字体**指其中每个字符有自己单独的宽度像素的字体。
本章将概述标准的 emWin 字体。

9.15.1 字体标识符命名约定

所有标准字体的命名如下所示。然后，表中解释了命名约定的元素：

```
GUI_Font[<style>][<width>x]<height>[x<MagX>x<MagY>][H][B][_<characterset>]
```

元素	描述
GUI_Font	emWin 随附的所有字体的标准前缀。
<style>	指定非标准字体样式。示例：Comic style in GUI_FontComic18B_ASCII.
<width>	字符宽度，仅包含在等宽字体中。
<height>	字体的高度 （像素）。
<MagX>	X 的放大系数，仅包含在放大字体中。
<MagY>	Y 的放大系数，仅包含在放大字体中。
H	“high”的缩写。仅在有多种相同高度的字体时使用。这意味着，该字体的显示高度比其他字体“高”。
B	“bold”的缩写。用于加粗字体。
<characterset>	指定字符的内容： ASCII：仅 ASCII 字符 0x20-0x7E (0x7F)。 1：ASCII 字符和欧洲扩展 0xA0 - 0xFF。 HK：平假名和片假名。 1 HK：ASCII、欧洲扩展、平假名和片假名。 D：数字字体，字符集：+-.0123456789.

示例 1

```
GUI_Font16_ASCII
```

元素	描述
GUI_Font	标准字体前缀。
16	高度 （像素）。
ASCII	字体仅包含 ASCII 字符。

示例 2

```
GUI_Font8x15B_ASCII
```

元素	描述
GUI_Font	标准字体前缀。
8	字符的宽度。
x15	高度 （像素）。
B	加粗字体。
ASCII	字体仅包含 ASCII 字符。

示例 3

GUI_Font8x16x1x2

元素	描述
GUI_Font	标准字体前缀。
8	字符的宽度。
x16	高度 （像素）。
x1	X 方向的放大系数。
x2	Y 方向的放大系数。

9.15.2 字体文件命名约定

字体文件的名称与字体本身的名称相似。文件的命名如下所示：
F[<width>]<height>[H][B][<characterset>]

元素	描述
F	emWin 随附的所有字体文件的标准前缀。
<width>	字符宽度，仅包含在等宽字体中。
<height>	字体的高度 （像素）。
H	“high”的缩写。仅在有多种相同高度的字体时使用。这意味着，该字体的显示高度比其他字体“高”。
B	“bold”的缩写。用于加粗字体。
<characterset>	指定字符的内容： ASCII：仅 ASCII 字符 0x20-0x7E (0x7F)。 1：ASCII 字符和欧洲扩展 0xA0 - 0xFF。 HK：平假名和片假名。 1 HK：ASCII、欧洲扩展、平假名和片假名。 D：数字字体。

9.15.3 字体的计量、 ROM 大小和字符集

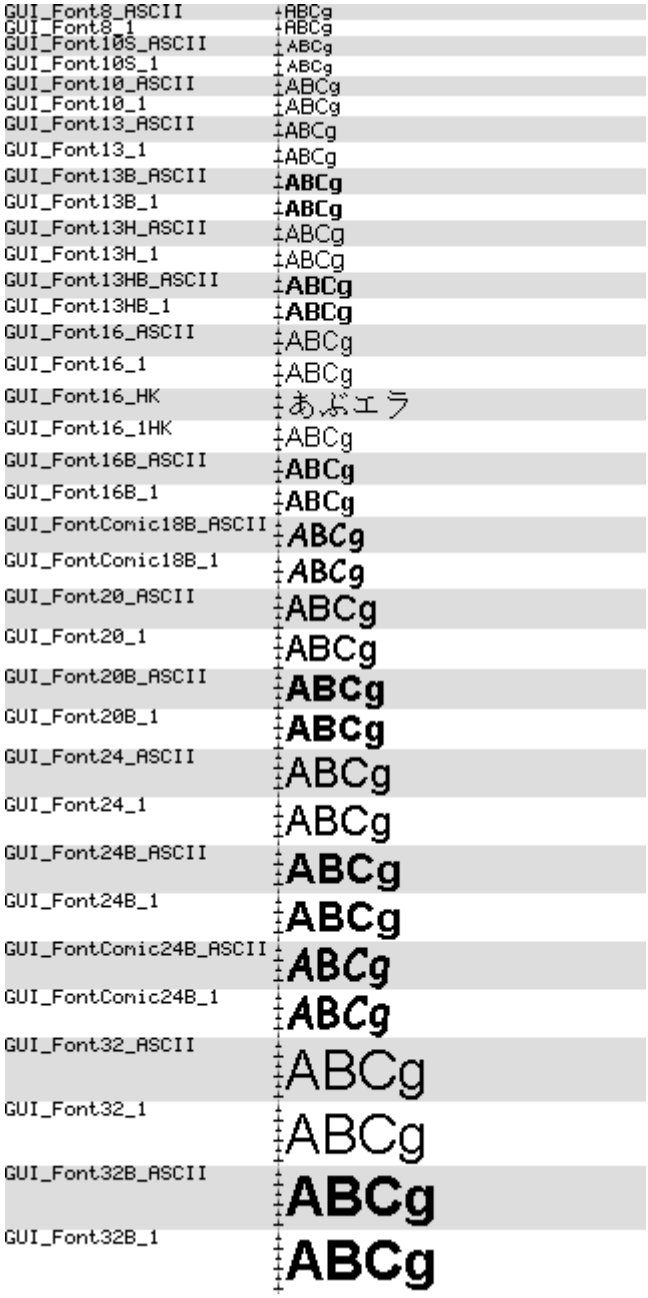
以下内容将介绍 emWin 随附的标准字体。对于每种字体，都有一张计量图表，需要包括所有字符的概述并使用包含 ROM 大小 （字节）的表以及字体文件。
在计量图表中将使用以下参数：

元素	描述
F	Y 的字体大小。
B	距字体顶部的基线的距离。
C	大写字符的高度。
L	小写字符的高度。
U	“g”、“j”或“y”等字母使用的底部长度大小。

9.15.4 比例字体

9.15.4.1 概述

以下屏幕截图概述了所有可用的比例字体：



9.15.4.2 计量、 ROM 大小和使用的文件

下表显示了字体的计量、 ROM 大小和使用的文件：

字体名称	计量	ROM 大小（字节）	使用的文件
GUI_Font8_ASCII	F: 8, B: 7, C: 7, L: 5, U: 1	1562	F08_ASCII.c
GUI_Font8_1	F: 8, B: 7, C: 7, L: 5, U: 1	1562+ 1586	F08_ASCII.c F08_1.c
GUI_Font10S_ASCII	F: 10, B: 8, C: 6, L: 4, U: 2	1760	F10S_ASCII.c

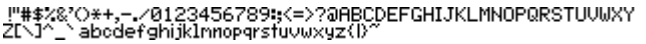
字体名称	计量	ROM 大小（字节）	使用的文件
GUI_Font10S_1	F: 10, B: 8, C: 6, L: 4, U: 2	1760+ 1770	F10_ASCII.c F10_1.c
GUI_Font10_ASCII	F: 10, B: 9, C: 8, L: 6, U: 1	1800	F10_ASCII
GUI_Font10_1	F: 10, B: 9, C: 8, L: 6, U: 1	1800+ 2456	F10_ASCII.c F10_1.c
GUI_Font13_ASCII	F: 13, B: 11, C: 8, L: 6, U: 2	2076	F13_ASCII.c
GUI_Font13_1	F: 13, B: 11, C: 8, L: 6, U: 2	2076+ 2149	F13_ASCII.c F13_1.c
GUI_Font13B_ASCII	F: 13, B: 11, C: 8, L: 6, U: 2	2222	F13B_ASCII.c
GUI_Font13B_1	F: 13, B: 11, C: 8, L: 6, U: 2	2222+ 2216	F13B_ASCII.c F13B_1.c
GUI_Font13H_ASCII	F: 13, B: 11, C: 9, L: 7, U: 2	2232	F13H_ASCII.c
GUI_Font13H_1	F: 13, B: 11, C: 9, L: 7, U: 2	2232+ 2291	F13H_ASCII.c F13H_1.c
GUI_Font13HB_ASCII	F: 13, B: 11, C: 9, L: 7, U: 2	2690	F13HB_ASCII.c
GUI_Font13HB_1	F: 13, B: 11, C: 9, L: 7, U: 2	2690+ 2806	F13HB_ASCII.c F13HB_1.c
GUI_Font16_ASCII	F: 16, B: 13, C: 10, L: 7, U: 3	2714	F16_ASCII.c
GUI_Font16_1	F: 16, B: 13, C: 10, L: 7, U: 3	2714+ 3850	F16_ASCII.c F16_1.c
GUI_Font16_HK	F: 16, B: 13, C: 10, L: 7, U: 3	6950	F16_HK.c
GUI_Font16_1HK	F: 16, B: 13, C: 10, L: 7, U: 3	120+ 6950+ 2714+ 3850	F16_1HK.c F16_HK.c F16_ASCII.c F16_1.c
GUI_Font16B_ASCII	F: 16, B: 13, C: 10, L: 7, U: 3	2690	F16B_ASCII.c
GUI_Font16B_1	F: 16, B: 13, C: 10, L: 7, U: 3	2690+ 2790	F16B_ASCII.c F16B_1.c
GUI_FontComic18B_ASCII	F: 18, B: 15, C: 12, L: 9, U: 3	3572	FComic18B_ASCII.c
GUI_FontComic18B_1	F: 18, B: 15, C: 12, L: 9, U: 3	3572+ 4334	FComic18B_ASCII.c FComic18B_1.c
GUI_Font20_ASCII	F: 20, B: 16, C: 13, L: 10, U: 4	4044	F20_ASCII.c
GUI_Font20_1	F: 20, B: 16, C: 13, L: 10, U: 4	4044+ 4244	F20_ASCII.c F20_1.c
GUI_Font20B_ASCII	F: 20, B: 16, C: 13, L: 10, U: 4	4164	F20B_ASCII.c
GUI_Font20B_1	F: 20, B: 16, C: 13, L: 10, U: 4	4164+ 4244	F20B_ASCII.c F20B_1.c
GUI_Font24_ASCII	F: 24, B: 20, C: 17, L: 13, U: 4	4786	F24_ASCII.c
GUI_Font24_1	F: 24, B: 20, C: 17, L: 13, U: 4	4786+ 5022	F24_ASCII.c F24_1.c
GUI_Font24B_ASCII	F: 24, B: 19, C: 15, L: 11, U: 5	4858	F24B_ASCII.c
GUI_Font24B_1	F: 24, B: 19, C: 15, L: 11, U: 5	4858+ 5022	F24B_ASCII.c F24B_1.c
GUI_FontComic24B_ASCII	F: 24, B: 20, C: 17, L: 13, U: 4	6146	FComic24B_ASCII
GUI_FontComic24B_1	F: 24, B: 20, C: 17, L: 13, U: 4	6146+ 5598	FComic24B_ASCII FComic24B_1

字体名称	计量	ROM 大小（字节）	使用的文件
GUI_Font32_ASCII	F: 32, B: 26, C: 20, L: 15, U: 6	7234	F32_ASCII.c
GUI_Font32_1	F: 32, B: 26, C: 20, L: 15, U: 6	7234+ 7734	F32_ASCII.c F32_1.c
GUI_Font32B_ASCII	F: 32, B: 25, C: 20, L: 15, U: 7	7842	F32B_ASCII.c
GUI_Font32B_1	F: 32, B: 25, C: 20, L: 15, U: 7	7842+ 8118	F32B_ASCII.c F32B_1.c

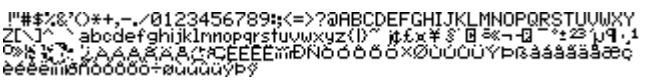
9.15.4.3 字符

下面显示了所有比例标准字体的全部字符：

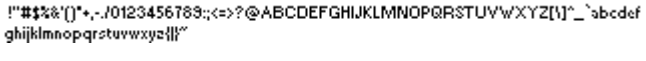
GUI_Font8_ASCII



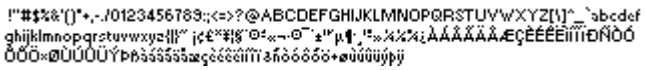
GUI_Font8_1



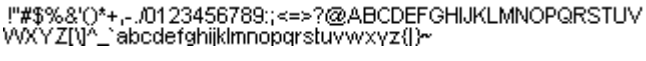
GUI_Font10S_ASCII



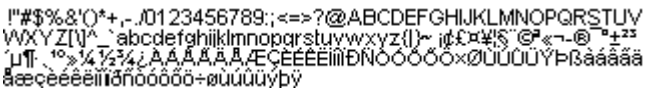
GUI_Font10S_1



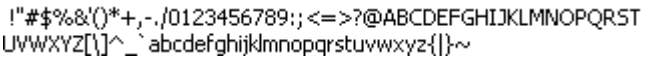
GUI_Font10_ASCII



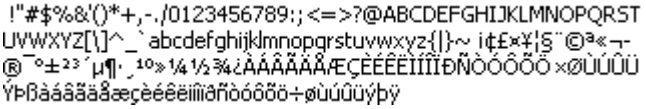
GUI_Font10_1



GUI_Font13_ASCII

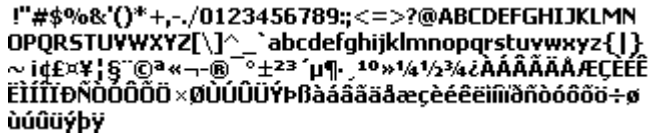


GUI_Font13_1

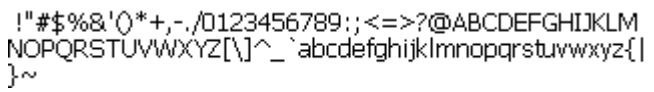


GUI_Font13B_ASCII

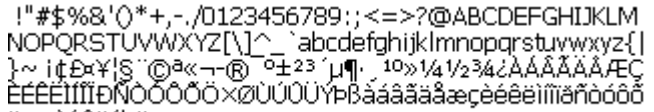
GUI_Font13B_1



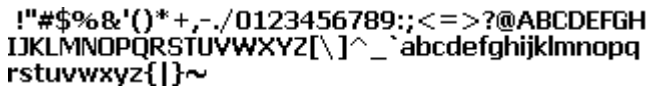
GUI_Font13H_ASCII



GUI_Font13H_1

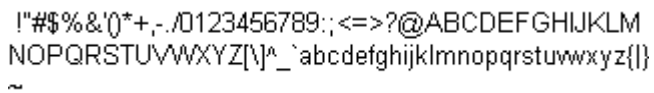


GUI_Font13HB_ASCII

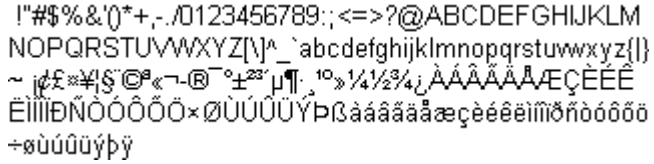


GUI_Font13HB_1

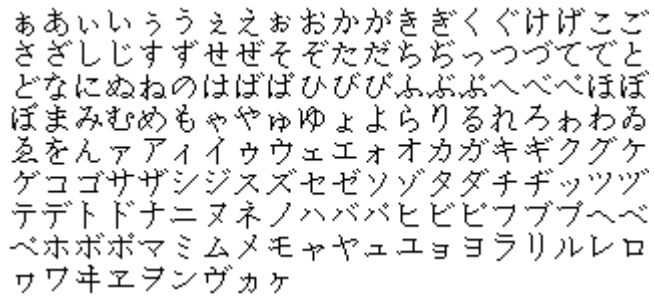
GUI_Font16_ASCII



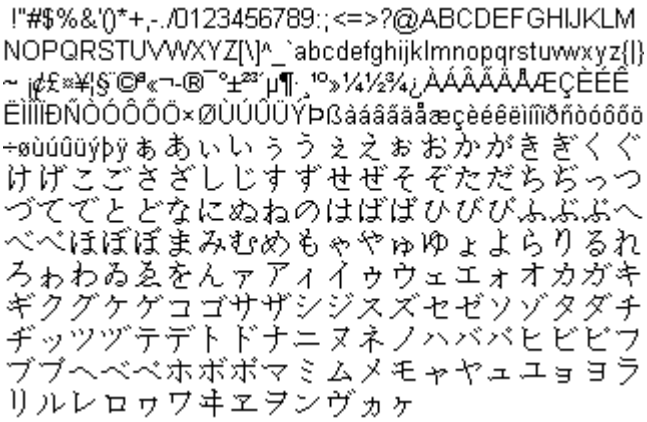
GUI_Font16_1



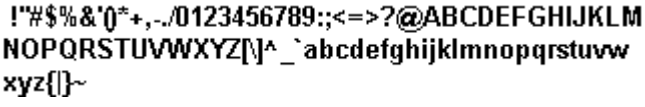
GUI_Font16_HK



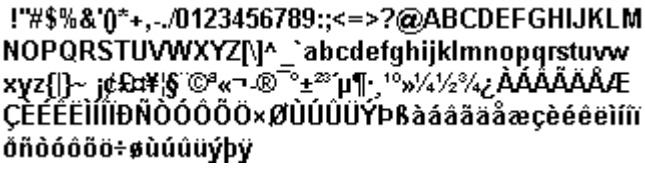
GUI_Font16_1HK



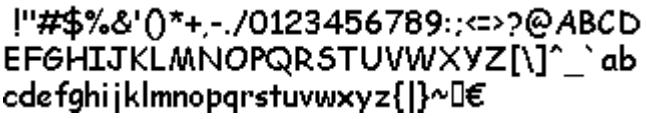
GUI_Font16B_ASCII



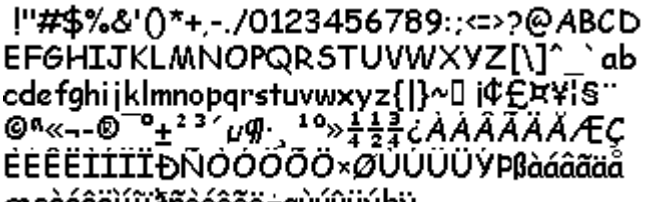
GUI_Font16B_1



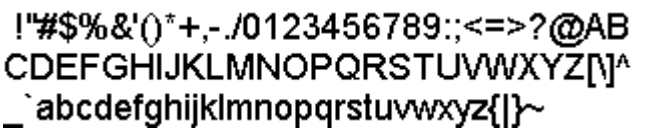
GUI_FontComic18B_ASCII



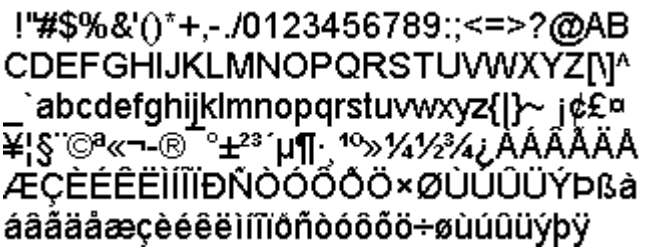
GUI_FontComic18B_1



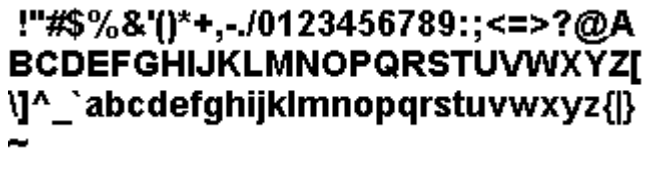
GUI_Font20_ASCII



GUI_Font20_1

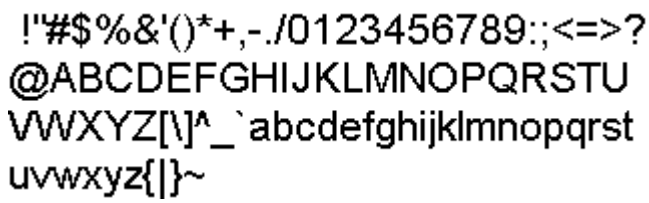


GUI_Font20B_ASCII

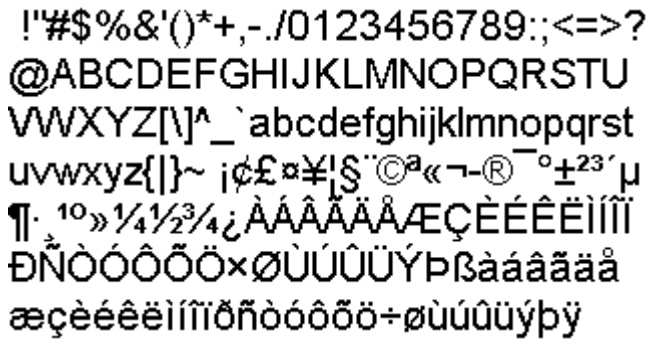


GUI_Font20B_1

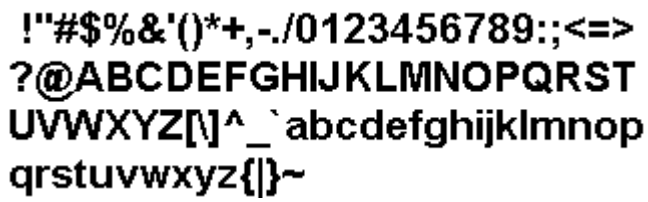
GUI_Font24_ASCII



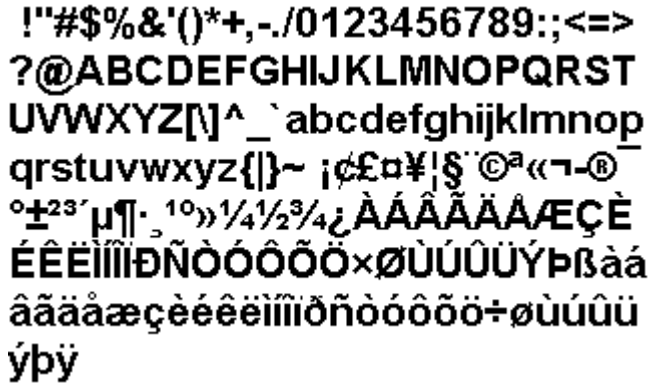
GUI_Font24_1



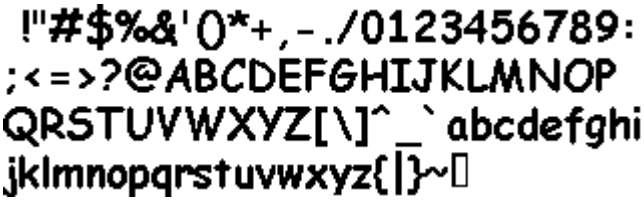
GUI_Font24B_ASCII



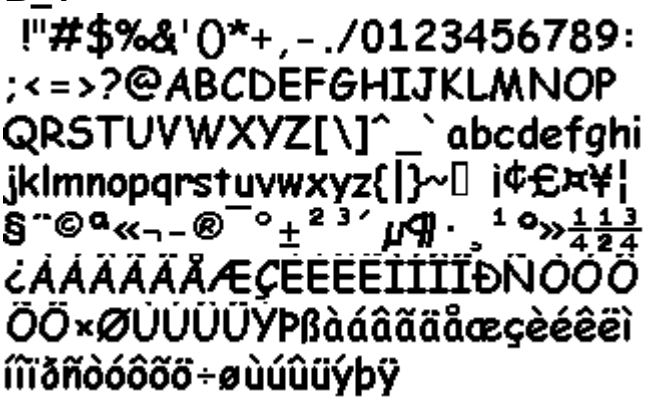
GUI_Font24B_1



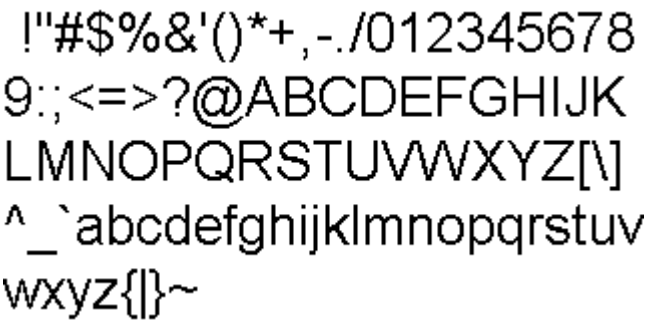
GUI_FontComic24B_ASCII



GUI_FontComic24B_1

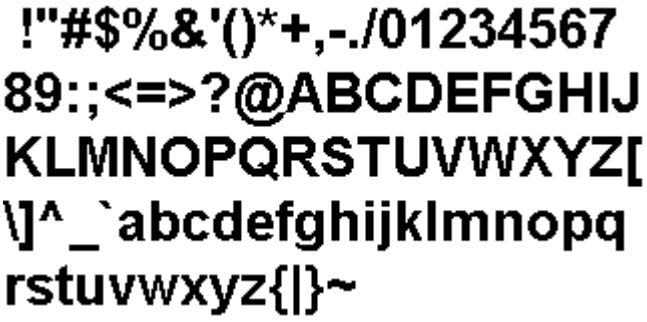


GUI_Font32_ASCII



GUI_Font32_1

GUI_Font32B_ASCII



GUI_Font32B_1

9.15.5 比例字体，带边框

9.15.5.1 概述

以下屏幕截图概述了所有可用的带边框比例字体：

9.15.5.2 计量、 ROM 大小和使用的文件

下表显示了字体的计量、 ROM 大小和使用的文件：

字体名称	计量	ROM 大小（字节）	使用的文件
GUI_Font20F_ASCII	F: 20, B: 19, C: 19, L: 19, U: 1	5248	F20F_ASCII.c

9.15.5.3 字符

下面显示了所有带边框的比例字体的全部字符：

GUI_Font20F_ASCII

9.15.6 等宽字体

9.15.6.1 概述

以下屏幕截图概述了所有可用的等宽字体：

9.15.6.2 计量、 ROM 大小和使用的文件

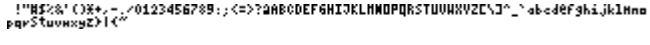
下表显示了字体的计量、 ROM 大小和使用的文件：

字体名称	计量	ROM 大小（字节）	使用的文件
GUI_Font4x6	F: 6, B: 5, C: 5, L: 4, U: 1	620	F4x6.c
GUI_Font6x8	F: 8, B: 7, C: 7, L: 5, U: 1	1840	F6x8.c
GUI_Font6x8_ASCII	F: 8, B: 7, C: 7, L: 5, U: 1	1568	F6x8_ASCII.c
GUI_Font6x8_1	F: 8, B: 7, C: 7, L: 5, U: 1	1568+ 1584	F6x8_ASCII.c F6x8_1.c
GUI_Font6x9	F: 9, B: 7, C: 7, L: 5, U: 2	1840 （与 GUI_Font6x8 的 ROM 位置相同）	F6x8.c
GUI_Font8x8	F: 8, B: 7, C: 7, L: 5, U: 1	1840	F8x8.c
GUI_Font8x8_ASCII	F: 8, B: 7, C: 7, L: 5, U: 1	1568	F8x8_ASCII.c
GUI_Font8x8_1	F: 8, B: 7, C: 7, L: 5, U: 1	1568+ 1584	F8x8_ASCII.c F8x8_1.c
GUI_Font8x9	F: 9, B: 7, C: 7, L: 5, U: 2	1840 （与 GUI_Font8x8 的 ROM 位置相同）	F8x8.c
GUI_Font8x10_ASCII	F: 10, B: 9, C: 9, L: 7, U: 1	1770	F8x10_ASCII.c
GUI_Font8x12_ASCII	F: 12, B: 10, C: 9, L: 6, U: 2	1962	F8x12_ASCII.c
GUI_Font8x13_ASCII	F: 13, B: 11, C: 9, L: 6, U: 2	2058	F8x13_ASCII.c
GUI_Font8x13_1	F: 13, B: 11, C: 9, L: 6, U: 2	2058+ 2070	F8x13_ASCII.c F8x13_1.c
GUI_Font8x15B_ASCII	F: 15, B: 12, C: 9, L: 7, U: 3	2250	F8x15_ASCII.c
GUI_Font8x15B_1	F: 15, B: 12, C: 9, L: 7, U: 3	2250+ 2262	F8x15B_ASCII.c F8x15B_1.c
GUI_Font8x16	F: 16, B: 12, C: 10, L: 7, U: 4	3304	F8x16.c
GUI_Font8x17	F: 17, B: 12, C: 10, L: 7, U: 5	3304 （与 GUI_Font8x16 的 ROM 位置相同）	F8x16.c
GUI_Font8x18	F: 18, B: 12, C: 10, L: 7, U: 6	3304 （与 GUI_Font8x16 的 ROM 位置相同）	F8x16.c
GUI_Font8x16x1x2	F: 32, B: 24, C: 20, L: 14, U: 8	3304 （与 GUI_Font8x16 的 ROM 位置相同）	F8x16.c
GUI_Font8x16x2x2	F: 32, B: 24, C: 20, L: 14, U: 8	3304 （与 GUI_Font8x16 的 ROM 位置相同）	F8x16.c
GUI_Font8x16x3x3	F: 48, B: 36, C: 30, L: 21, U: 12	3304 （与 GUI_Font8x16 的 ROM 位置相同）	F8x16.c
GUI_Font8x16_ASCII	F: 16, B: 12, C: 10, L: 7, U: 4	2328	F8x16_ASCII.c
GUI_Font8x16_1	F: 16, B: 12, C: 10, L: 7, U: 4	2328+ 2352	F8x16_ASCII.c F8x16_1.c

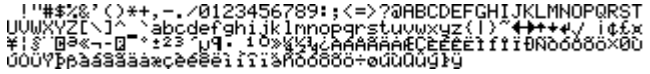
9.15.6.3 字符

下面显示了所有等宽标准字体的全部字符：

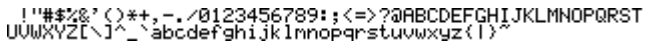
GUI_Font4x6



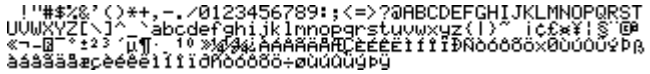
GUI_Font6x8



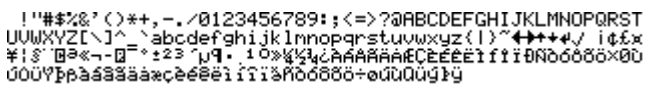
GUI_Font6x8_ASCII



GUI_Font6x8_1

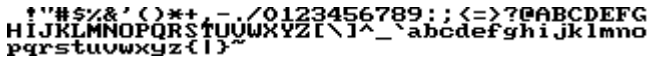


GUI_Font6x9



GUI_Font8x8

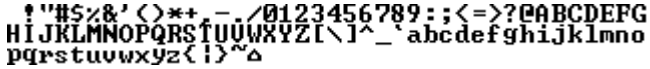
GUI_Font8x8_ASCII



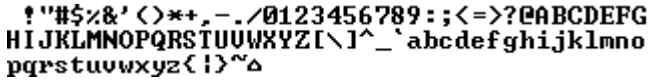
GUI_Font8x8_1

GUI_Font8x9

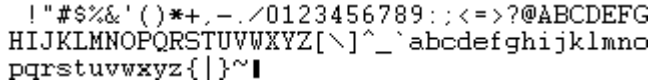
GUI_Font8x10_ASCII



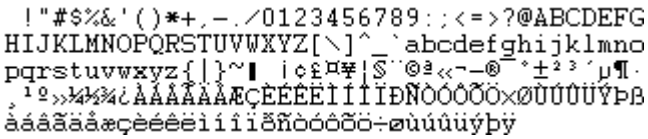
GUI_Font8x12_ASCII



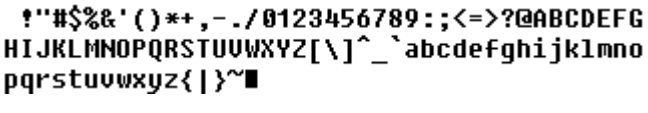
GUI_Font8x13_ASCII



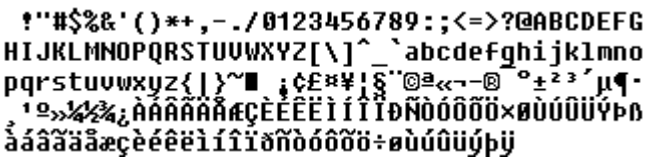
GUI_Font8x13_1



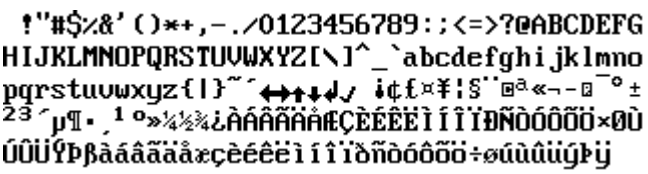
GUI_Font8x15B_ASCII



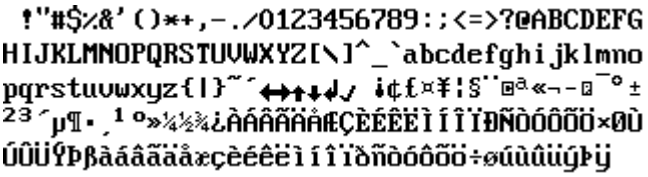
GUI_Font8x15B_1



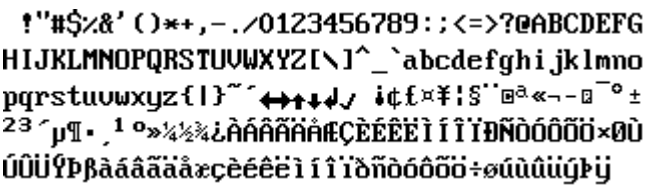
GUI_Font8x16



GUI_Font8x17

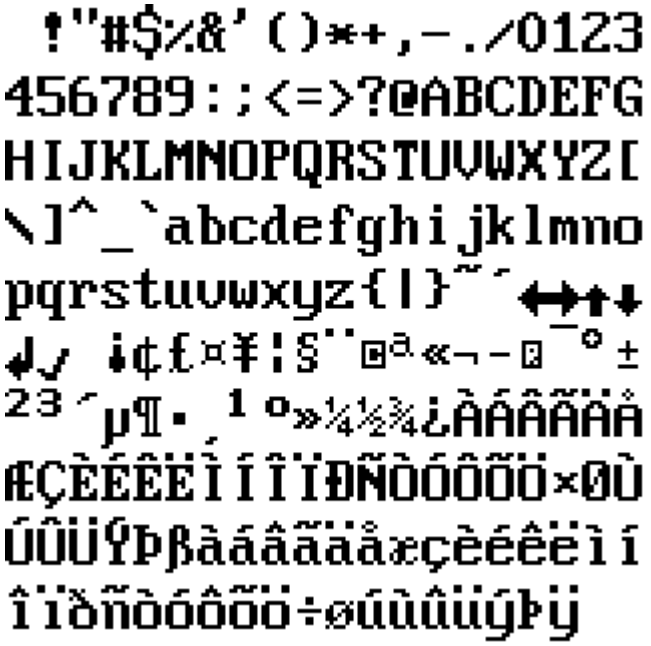


GUI_Font8x18

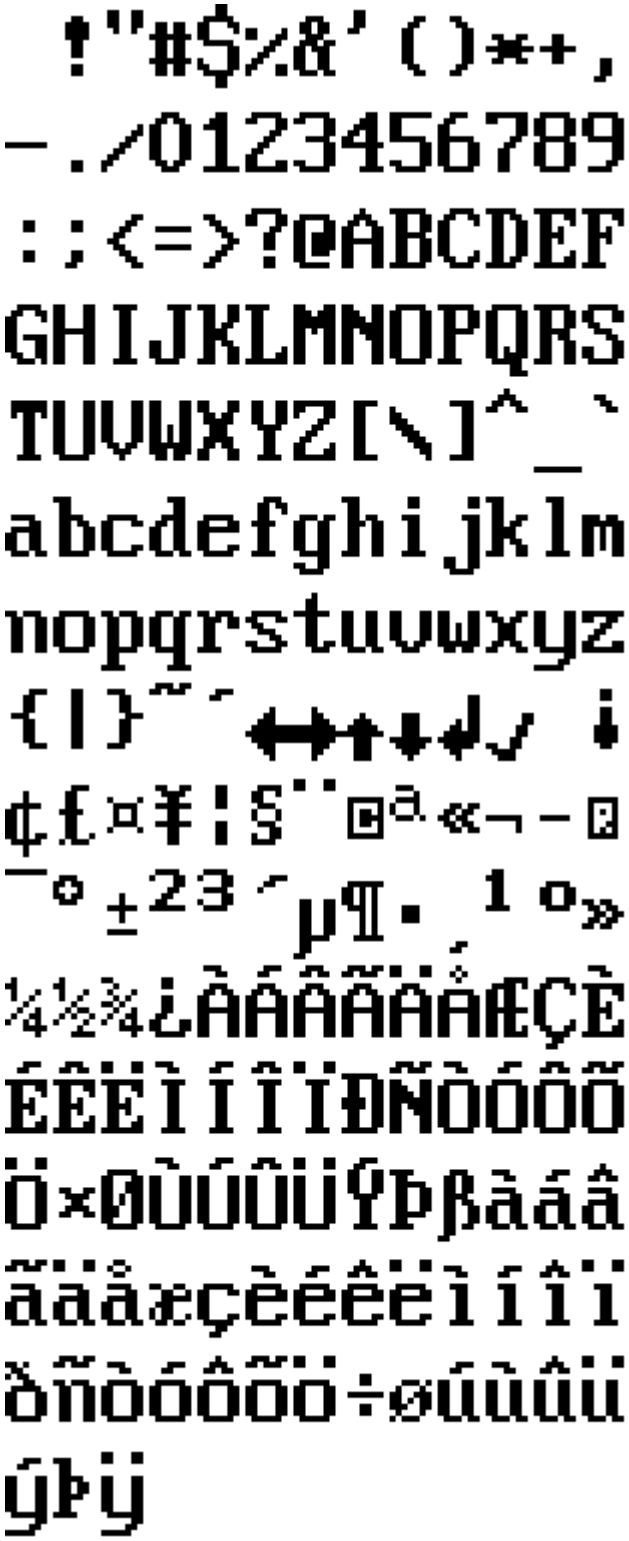


GUI_Font8x16x1x2

GUI_Font8x16x2x2



GUI_Font8x16x3x3



GUI_Font8x16_ASCII

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~
```

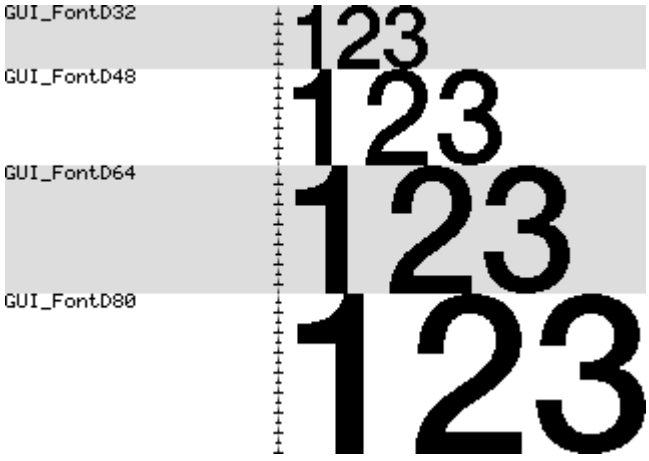
GUI_Font8x16_1

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸
¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßà
áâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ
```

9.15.7 数字字体 （比例）

9.15.7.1 概述

以下屏幕截图概述了所有可用的比例数字字体：



9.15.7.2 计量、 ROM 大小和使用的文件

下表显示了字体的计量、 ROM 大小和使用的文件：

字体名称	计量	ROM 大小（字节）	使用的文件
GUI_FontD32	F: 32, C: 31	1574	FD32.c
GUI_FontD48	F: 48, C: 47	3512	FD48.c
GUI_FontD64	F: 64, C: 63	5384	FD64.c
GUI_FontD80	F: 80, C: 79	8840	FD80.c

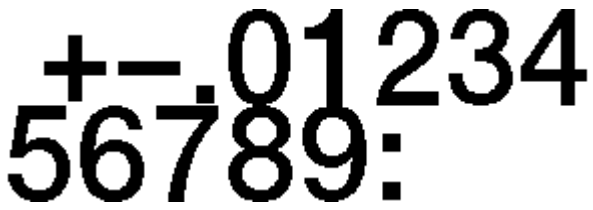
9.15.7.3 字符

下面显示了所有比例数字字体的全部字符：

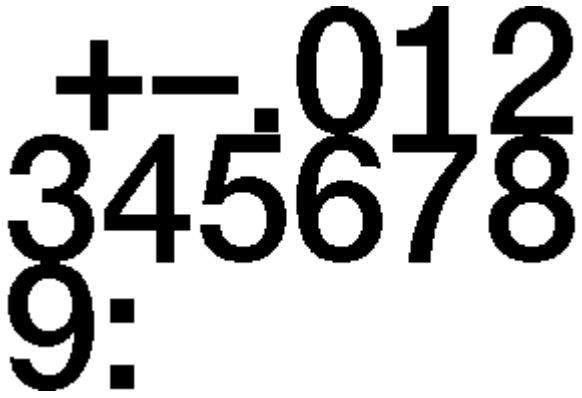
GUI_FontD32

```
+-.0123456789:
```

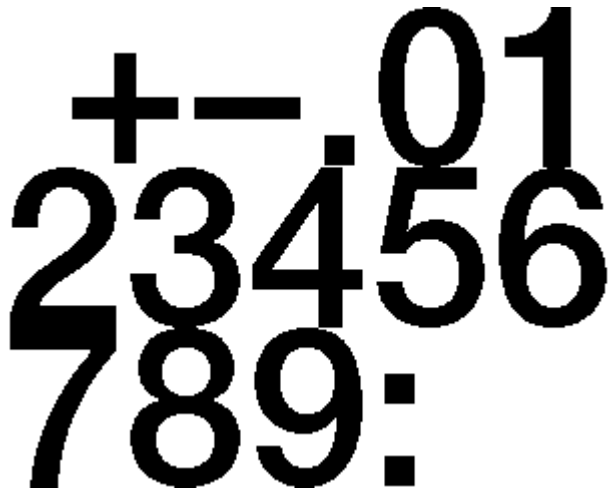
GUI_FontD48



GUI_FontD64



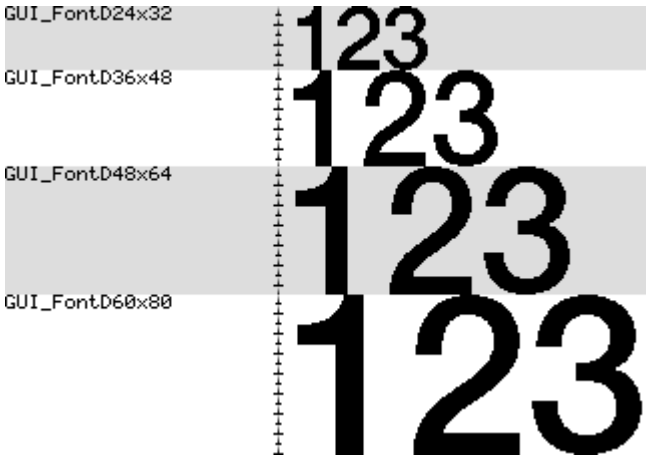
GUI_FontD80



9.15.8 数字字体 （等宽）

9.15.8.1 概述

以下屏幕截图概述了所有可用的等宽数字字体：



9.15.8.2 计量、 ROM 大小和使用的文件

下表显示了字体的计量、 ROM 大小和使用的文件：

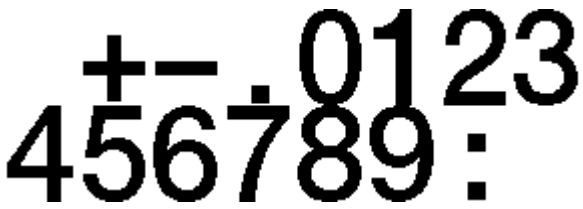
字体名称	计量	ROM 大小（字节）	使用的文件
GUI_FontD24x32	F: 32, C: 31	1606	FD24x32.c
GUI_FontD36x48	F: 48, C: 47	3800	FD36x48.c
GUI_FontD48x64	F: 64, C: 63	5960	FD48x60.c
GUI_FontD60x80	F: 80, C: 79	9800	FD60x80.c

9.15.8.3 字符

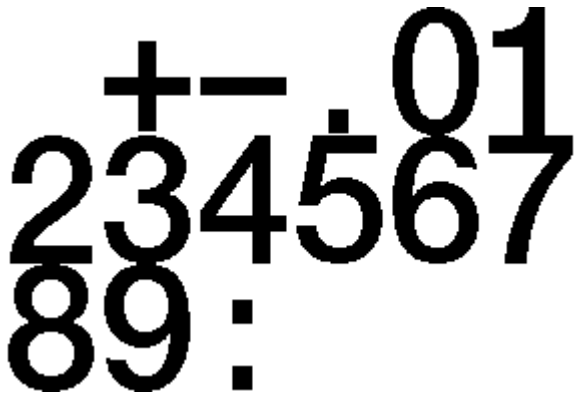
下面显示了所有等宽数字字体的全部字符：

GUI_FontD24x32

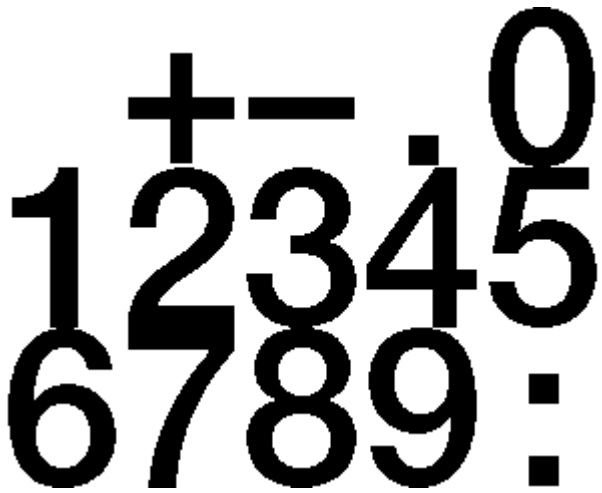
GUI_FontD36x48



GUI_FontD48x64



GUI_FontD60x80

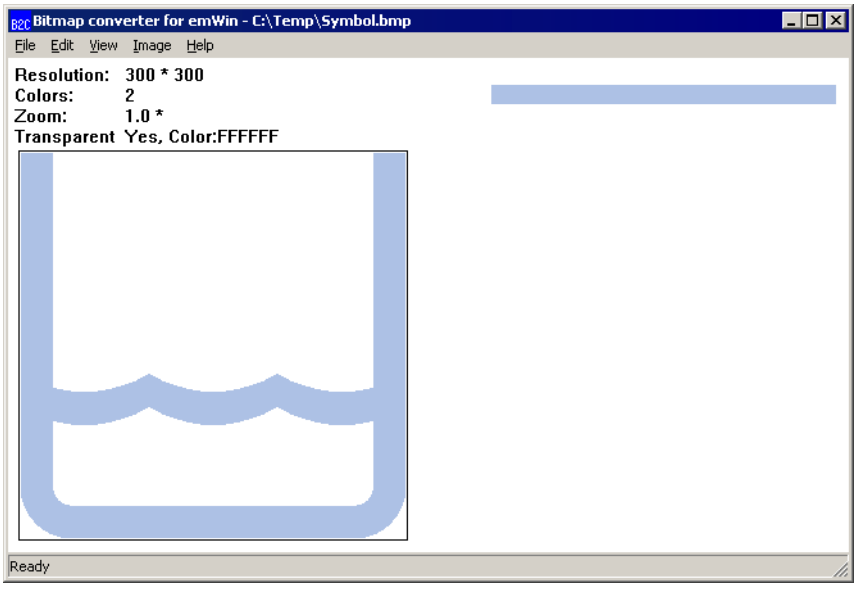


第 10 章

位图转换器

位图转换器是一个简单易用的 Windows 程序，只需将位图 （bmp 或 gif 格式的文件）加载到应用程序中。在需要或必须时可转换色彩格式，并通过另存为相应的格式，将其转换为 C 文件。然后即可对 C 文件进行编译，使图像可在具有 emWin 的显示器中显示。

位图转换器的屏幕截图



10.1 功能

位图转换器的主要作用是将位图从 PC 格式转换为 C 文件。emWin 可使用的位图在 C 文件中通常定义为 GUI_BITMAP 结构。这些结构 （更确切地说是这些结构所引用的图像数据）会非常大。手动生成这些位图很费时并且效率很低。因此建议使用位图转换器，自动从位图生成 C 文件。
它的另一项有用功能是将图像另存为 C 流文件。与常规 C 文件相比，其优势在于这些数据流可位于需要将 C 文件置于的可寻址 CPU 区域的任何媒体中的任何位置。
还具有色彩转换功能，因此可以减小所生成的 C 代码。通常可以减少每个像素的位数以减少内存消耗。位图转换器显示已转换的图像。
位图转换器可执行很多简单的功能，包括缩放尺寸、水平或垂直裁剪位图、旋转以及转换位图索引或色彩（这些功能可在“图像”菜单中找到）。对图像的任何进一步修改都必须在位图处理程序如 Adobe Photoshop 或 Corel Photopaint 中进行。通常比较好的做法是，在这类程序中进行图像修改，而仅使用位图转换器进行转换。

10.2 加载位图

10.2.1 支持的输入文件格式

位图转换器主要支持 Windows 位图文件 (*.bmp)、“图形交换格式” (*.gif) 和 “可移植的网络图形” (*.png)：

Windows 位图文件 (BMP)

位图转换器支持最通用的位图文件格式，它可打开以下格式的位图文件：

- 每像素 1、 4 或 8 位 (bpp)，带调色板；
- 16、 24 或 32bpp，无调色板 （全彩模式，其中每种颜色分配一个 RGB 值）；
- RLE4 和 RLE8。

尝试读取其他格式的位图文件时，位图转换器会显示一条错误消息。

图形交换格式 (GIF)

位图转换器支持每个 GIF 文件读取一个图像。例如，如果文件包含由多个图像组成的电影，则转换器只读取第一个图像。
转换器支持透明和隔行扫描的 GIF 图像。

可移植的网络图形 (PNG)

PNG 格式是用 Alpha 混合处理创建图像时最常推荐的格式。位图转换器支持读取具有 Alpha 通道的 PNG 图像。

10.2.2 从文件加载

选择 File/Open，可在位图转换器中直接打开所支持格式的图像文件。

10.2.3 使用剪贴板

可用其他程序打开其他任何类型的位图 （即 .jpg, .jpeg, .png, .tif），将其复制到剪贴板，然后粘贴到位图转换器中。此过程与直接从文件加载位图具有相同的效果。

10.3 从位图生成 C 文件

位图转换器的主要功能是将 PC 格式的位图转换为 emWin 可用的 C 文件。但在转换前，通常需要修改图像的调色板，使生成的 C 文件不会过大。
位图可以另存为 bmp 或 gif 文件 （可以重新加载和使用，或加载到其他位图处理程序中），或另存为 C 文件。 C 文件将用作 C 编译器的输入文件，保存时可包含调色板 （设备无关位图，或 DIB）或不包含调色板 （设备相关位图，或 DDB）。建议使用 DIB，因为它可在任何显示器中正常显示；而 DDB 只能在使用与位图相同的调色板的显示器中正常显示。
生成的 C 文件可以是 “带调色板的 C”、“不带调色板的 C”、“带调色板的 C，压缩”或 “不带调色板的 C，压缩”。有关压缩文件的详细信息，请参见 “压缩位图”一节以及本章末尾的示例。

10.3.1 支持的位图格式

下表列出了 C 文件当前可用的输出格式：

格式	色彩深度	压缩	透明性	调色板
每像素 1 位	1bpp	否	是	是
每像素 2 位	2bpp	否	是	是
每像素 4 位	4bpp	否	是	是
每像素 8 位	8bpp	否	是	是
压缩， RLE4	4bpp	是	是	是
压缩， RLE8	8bpp	是	是	是
高彩 555	15bpp	否	否	否
高彩 555，红色和蓝色交换	15bpp	否	否	否
高彩 565	16bpp	否	否	否
高彩 565，红色和蓝色交换	16bpp	否	否	否
高彩 565，压缩	16bpp	是	否	否
高彩 565，红色和蓝色交换，压缩	16bpp	是	否	否
真彩 888	24bpp	否	否	否
真彩 8888，带 Alpha 通道	32bpp	否	是	否
Alpha 通道，压缩	8bpp	是	是	否
真彩，带 Alpha 通道，压缩	32bpp	是	是	否

10.3.2 调色板信息

位图调色板是一个具有 24 位 RGB 颜色表项的阵列。色彩深度从 1 - 8bpp 的位图可以保存为带调色板信息 （设备无关位图， DIB）或不带调色板信息 （设备相关位图 DDB）。

设备无关位图 (DIB)

颜色信息以颜色阵列索引的形式存储。 emWin 绘制 DIB 前，将位图调色板的 24 位 RGB 色彩转换为硬件调色板的颜色索引。使用 DIB 的优点是它们与硬件无关，可在具有不同颜色配置的系统中正常绘制。缺点是调色板要求额外的 ROM，并且由于色彩转换，性能有所降低。

设备相关位图 (DDB)

DDB 的像素信息是显示器硬件调色板的索引，绘制 DDB 前无需进行转换。优点是对 ROM 的要求较少，性能更高。缺点是这些位图不能在使用其他颜色配置的系统中正常显示。

10.3.3 透明性

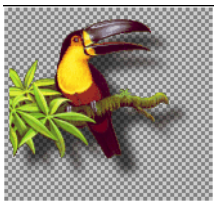
可将基于位图的调色板转换为透明位图。透明性意味着索引为 0 的每个像素都将不会产生任何输出。可使用 `Image/Transparency` 命令来选择要透明的颜色。选择透明颜色后，将重新计算图像的像素索引，使选定的颜色在位图调色板的位置 0 上。将位图保存为 C 文件时，其透明属性将一同保存。

10.3.4 Alpha 混合

Alpha 混合是将图像和背景混合在一起，产生半透明效果的一种方法。像素的 Alpha 值决定了其透明性。位图绘制后，像素的颜色是以前的颜色和位图中颜色值的混合。在 emWin 中，逻辑颜色作为 32 位值处理。较低的 24 位用于颜色信息，较高的 8 位用于管理 Alpha 值。 Alpha 值为 0 表示图像不透明，值为 0xFF 表示完全透明。然而，BMP 和 GIF 文件不支持 Alpha 混合，PNG 文件支持 Alpha 混合。因此，创建具有 Alpha 混合的位图文件最简单的方法是加载 PNG 文件。使用 BMP 和 / 或 GIF 文件时，位图转换器最初没有关于 Alpha 值的信息。

加载 PNG 文件

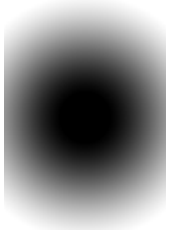
这是创建具有 Alpha 掩码位图的最推荐方法：

加载后


PNG 文件包含所有需要的信息。

从 Alpha 掩码位图加载 Alpha 值

此方法从单独的文件加载 Alpha 值。Alpha 掩码文件的黑色像素表示不透明，白色像素表示透明。下表为一个示例：

起点	Alpha 掩码	结果
		

可使用命令 File/Load Alpha Mask 来加载 Alpha 掩码。

从两个位图创建 Alpha 值

此方法用两个图像像素之间的差值来计算Alpha值。第一个图像应在黑色背景上显示项目，第二个图像应在白色背景上显示同一项目。下表示例显示了如何使用命令 File/Create Alpha 来创建 Alpha 值：

起点	黑色背景	白色背景	结果
			

命令 File/Create Alpha 可用于创建 Alpha 值。

10.3.5 选择最佳格式

emWin 支持生成的 C 文件的各种格式。这取决于成为兆罢颜格式的数个条件，没有可用的基本规则。色彩深度、压缩、调色板和透明性都会影响绘制性能和 / 或位图的 ROM 要求。

色彩深度

通常，色彩深度越低，位图的 ROM 要求越小。每个显示器驱动都是针对绘制 1bpp 位图 （文本）和具有与显示器相同色彩深度的位图进行优化的。

压缩

支持 RLE 压缩方法，这种方法对具有许多水平序列的等着色像素的位图具有最佳效果，本章稍后将详细说明。性能通常稍慢于绘制不压缩的位图。

调色板

调色板的 ROM 要求是每种颜色 4 字节。因此，256 色调色板使用 1 千字节。另外，emWin 还需要在绘制位图前转换调色板的颜色。优点：位图与设备无关，可在任何显示器上显示，并与其色彩深度和格式无关。

透明性

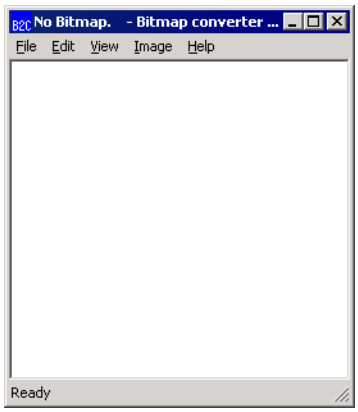
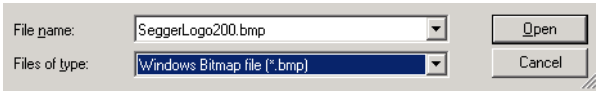
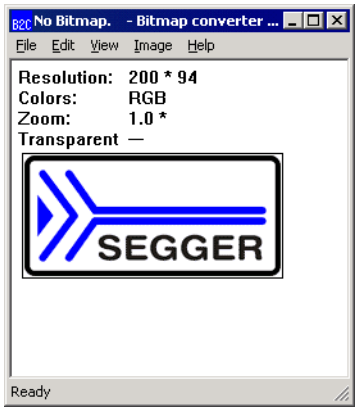
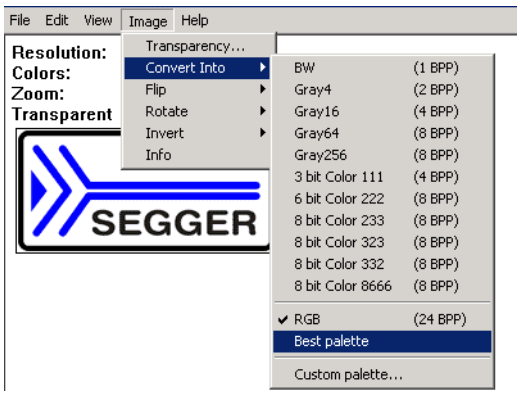
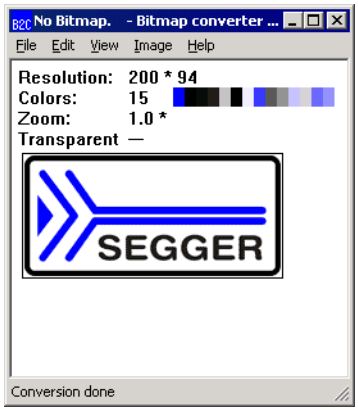
透明位图的 ROM 要求与不透明位图的要求相同。透明位图的性能稍慢于不透明位图。

高彩和真彩位图

需要特别注意这两种格式的位图。通常，这两种格式仅在色彩深度为 15 位及以上的显示器上才有用。另外，强烈建议将 C 文件保存为与硬件所用格式完全相同的格式。请记住，使用正确的格式对绘制性能具有积极影响。例如，如果高彩位图要在色彩深度为 16bpp、红色和蓝色分量交换的系统中显示，则最佳格式为 “高彩 565，红色和蓝色交换” 。对于每个像素都需进行色彩转换的其他格式稍有影响，但是，具有正确格式的位图可以非常快速的渲染，而不用进行色彩转换。在此情况下，绘制性能的差距将是 10 倍及以上。

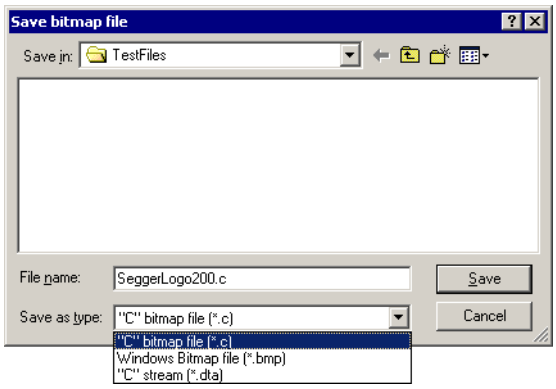
10.3.6 保存文件

使用位图转换器的基本步骤如下所示：

第 1 步：启动应用程序。 位图转换器打开，显示一个空窗口。	
第 2 步：将位图加载到位图转换器中。 选择 File/Open。 找到要打开的文件并点击打开（必须是 bmp 文件）。 在本例中，选择 Logo200.bmp。  位图转换器显示已加载的位图。	 在本例中，加载的位图为全彩模式。它必须先转换为调色板格式，然后才能生成 C 文件。
第 3 步：必要时转换图像。 选择 Image/Convert Into。 选择所需的调色板。 在本例中，选择选项最佳调色板。  位图转换器显示已转换的位图。	 图像外观没有变化，但使用更少的内存，因为调色板仅使用 15 色，而不是使用全彩模式。这 15 色仅是显示此特定图像的实际要求。

第 4 步：将位图另存为 C 文件。

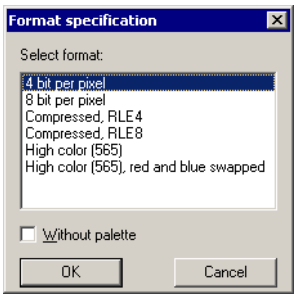
选择 File/Save As。
为 C 文件选择目的地并命名。
选择文件类型。在本例中，文件另存为 C 位图文件。
点击保存。



第 5 步：指定位图格式。

如果要将位图另存为 C 文件，现在即可指定格式。请使用对话框中显示的可用格式之一。如果要不带调色板保存位图，则激活复选框 “不带调色板”。

位图转换器将在指定目的地创建一个单独文件，包含该位图的 C 源代码。

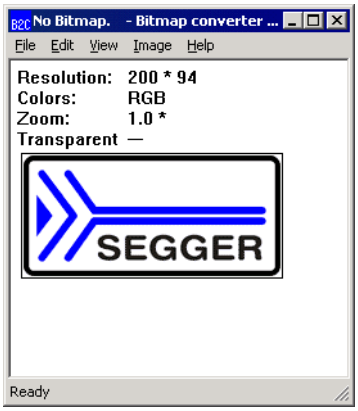


10.4 色彩转换

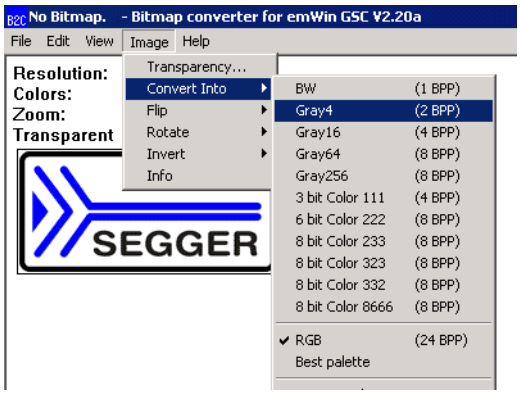
转换位图颜色格式的主要原因是为了减少内存消耗。最常用的方法是使用上例所示的选项 “最佳调色板”，它可定制特定位图的调色板，只包含图像中要用的颜色。对于为了使调色板尽可能小，同时还要完全支持图像的全彩位图，此选项尤其有用。在位图转换器中打开位图文件后，只需从菜单中选择 Image/Convert Into/Best palette。如果需要保持透明性，请选择 Image/Convert Into/Best palette + transparency。
对于某些应用，可能使用固定颜色调色板更高效，请在 Image/Convert Into 菜单下选择。例如，假如要在仅支持四种灰度的显示器上显示全彩模式的位图。此时使图像保持为原始格式会浪费内存，因为在显示器上只显示为四种灰度。将全彩位图转换为四灰度、 2bpp 位图可获得最高效率。
转换步骤如下所示：

按上一示例的步骤 1 和 2 打开位图转换器并加载同一文件。

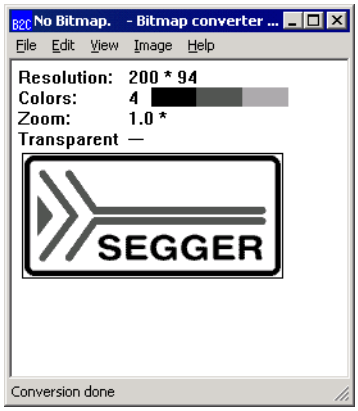
位图转换器显示已加载的位图。



选择 Image/Convert Into/Gray4。



位图转换器显示已转换的位图。



在本例中，图像使用较少的内存，因为调色板仅使用4灰度，而不是全彩模式。如果目标显示器仅支持 4 灰度，采用更高像素深度没有用，因为它仅会浪费内存。

10.5 生成 C 流文件

C 流文件由与 C 文件相同的信息组成。它与 C 文件相反，数据流可位于任何地方，无需编译或与项目链接。C 文件支持的所有输出格式对 C 流文件也可用。emWin 支持从数据流创建位图并直接绘制数据流。有关 C 流文件支持的详细信息，请参见 “绘制位图”（第 113 页）。

10.6 压缩位图

位图转换器和 emWin 支持在生成的源代码文件中对位图进行游程编码 (RLE) 压缩。如果位图包含许多水平序列的等着色像素，则 RLE 压缩方法最有效。高效压缩的位图将节省大量的空间。但是，不建议对摄影图像进行压缩，因为它们通常不具有相同像素的序列。也应注意，压缩图像的显示时间会稍微更长。

如果要保存使用 RLE 压缩的位图，可在另存为 C 文件时，选择一种压缩输出格式：“带调色板的 C，压缩”或 “不带调色板的 C，压缩”。显示压缩位图无需特殊功能；与显示不压缩位图的方法相同。

压缩比

获得的压缩比将随所用位图的不同而不同。图像水平均匀性越好，压缩比越高。每个像素的位数越高，产生压缩比也越高。

在上一示例使用的位图中，图像中的像素总数为 (200*94) = 18,800。

因为 2 像素存储在 1 字节中，因此未压缩图像总的大小为 18,800/2 = 9,400 字节。

此 18,800 像素的特定位图的总压缩大小为 3,803 字节 （参见本章末尾的示例）。

因此可计算出压缩比为 9,400/3,803 = 2.47。

10.7 使用定制调色板

将位图转换为定制调色板并保存为不带调色板信息的位图，可节省内存并提高位图绘制操作的性能。

更有效的内存利用

默认情况下，每个位图都包含自己的调色板。即使是最小的位图也包含多达 256 色的大调色板。在很多情况下，位图只使用了调色板的一小部分。如果使用很多这种位图，则调色板所用的内存量将快速增加。
因此，如果将 emWin 所用的位图转换为可用的硬件调色板，并将它们保存为不带调色板信息的设备相关位图 (DDB)，将会节省很多 ROM。

更高的位图绘制性能

emWin 绘制位图前，需要将每个设备无关位图调色板转换为可用的硬件调色板。这是必需的，因为位图文件的像素索引指向设备无关位图调色板的索引，而非可用的硬件调色板。
将位图转换为 DDB，意味着运行时无需色彩转换，因此能加速绘制。

10.7.1 保存调色板文件

位图转换器可将当前加载的位图的调色板保存到调色板文件中，该文件可用命令 `Image/Convert Into/Custom palette` 来转换其他位图。这就要求当前文件是基于调色板的文件而不是 RGB 文件。要保存文件，可使用命令 `File/Save palette...`。

10.7.2 C 文件格式

定制调色板文件是定义转换可使用颜色的简单文件。由以下部分组成：

- 文件头 （8 字节)。
- NumColors （U32， 4 字节）。
- 0 （4 字节)。
- U32 颜色 [NumColors]（NumColors*4 字节，类型 GUI_COLOR）。

因此整个文件大小为：16+(NumColors*4) 字节。 8 色的定制调色板文件大小将为 16+(8*4) = 48 字节。此时，必须使用二进制编辑器来创建这种文件。
所支持的最多颜色是 256 色，最少是 2 色。

示例

本示例文件将定义一个包含 2 种颜色 - 红色和白色的调色板：

```
0000: 65 6d 57 69 6e 50 61 6c 02 00 00 00 00 00 00 00
0010: ff 00 00 00 ff ff ff 00
```

8 个文件头构成第一行的头八个字节。U32 首先存储 lsb（大端）并代表后面四个字节，随后是四个 0 字节。以每个颜色用 1 个字节存储，其中第 4 个字节是 0，如下所示：RRGGBB00. 第二行代码定义本示例中使用的两种颜色。

10.7.3 固定调色板模式的调色板文件

使用定制调色板功能甚至在最常用的固定调色板模式下也很有用，而不是仅限于定制硬件调色板。对于大多数基于调色板的固定调色板模式，可在文件夹 `Sample\Palette` 中找到调色板文件。

10.7.4 转换位图

使用命令 Image/Convert Into/Custom palette，可将当前加载的位图转换为定制调色板。位图转换器尝试为当前加载位图的每个像素查找调色板文件的最接近颜色。

10.8 命令行用法

也可使用命令提示符来操作位图转换器。位图转换器菜单中可用的所有转换功能都可通过命令执行，并且在位图中执行的任何数量的功能都可在一个命令行中完成。

10.8.1 命令的格式

命令使用以下格式输入：

```
BmpCvt <filename>.bmp <-command>
```

（如果使用多个命令，则在各个命令之间键入一个空格）。
例如，将名为 logo.bmp 的位图转换为最佳调色板格式并另存为名为 logo.bmp 的 C 文件，在命令提示符后输入以下代码即可一次完成：

```
BmpCvt logo.bmp -convertintobestpalette -saveaslogo,1 -exit
```

请注意，要加载到位图转换器中的文件始终包含其 bmp 扩展名，但在 -saveas 命令中不用写入文件扩展名，而是用一个整数来指定所需的文件类型。上述 -saveas 命令行中的整数 1 指派“带调色板的 C”。 -exit 命令在完成后自动关闭程序。有关详细信息，请参见下表。

10.8.2 有效的命令行选项

下表列出了所有允许的位图转换器命令。也可通过在命令提示符后输入 BmpCvt -?，随时查看这些命令。

命令	描述
-convertintobw	转换为 BW。
-convertintogray4	转换为 4 灰度。
-convertintogray16	转换为 16 灰度。
-convertintogray64	转换为 64 灰度。
-convertintogray256	转换为 256 灰度。
-convertinto111	转换为 111。
-convertinto222	转换为 222。
-convertinto233	转换为 233。
-convertinto323	转换为 323。
-convertinto332	转换为 332。
-convertinto8666	转换为 8666。
-convertintorgb	转换为 RGB。
-convertintobestpalette	转换为最佳调色板。
-convertintotranspalette	转换为具有透明性的最佳调色板。
-convertintocustompalette<filename>	转换为定制调色板。
<filename>	所需定制调色板的用户指定的文件名。
-exit	自动终止 PC 程序。
-fliph	水平翻转图像。
-flipv	垂直翻转图像。
-help	显示此对话框。
-invertindices	反转索引。
-rotate90cw	顺时针旋转图像 90 度。
-rotate90cc	逆时针旋转图像 90 度。

命令	描述
-rotate180	180 度旋转图像。
-saveas<filename>,<type>[,<fmt>[,<noplt>]]	将文件另存为“filename”。
<filename>	用户指定的文件名，包括文件扩展名。
<type>	必须是 1 到 4 的整数，如下所示： 1：带调色板的 C （.c 文件） 2：Windows 位图文件 （bmp 文件） 3：C 流文件 （.dta 文件） 4：GIF 格式 （gif 文件）
<fmt>	指定位图格式 （仅在类型 == 1 时）： 1：每像素 1 位 2：每像素 2 位 4：每像素 4 位 5：每像素 8 位 6：RLE4 压缩 7：RLE8 压缩 8：高彩 565 9：高彩 565，红色和蓝色交换 10：高彩 555 11：高彩 555，红色和蓝色交换 12：RLE16 压缩 13：RLE16 压缩，红色和蓝色交换 15：真彩 32bpp，压缩 16：真彩 32bpp 17：真彩 24bpp 18：Alpha 通道 8bpp，压缩 如果没有指定此参数，则位图转换器使用以下默认格式，取决于位图的颜色数： 颜色数 <= 2：每像素 1 位 颜色数 <= 4：每像素 2 位 颜色数 <= 16：每像素 4 位 颜色数 <= 256：每像素 8 位 RGB：高彩 565
<noplt>	保存位图，带或不带调色板 （仅在类型 == 1 时） 0：带调色板保存位图 （默认） 1：不带调色板保存位图
-transparency<RGB-Color>	设置透明颜色。
<RGB-Color>	要用作透明颜色的 RGB 颜色。
-?	显示此对话框。

10.9 转换位图示例

使用位图转换器的典型示例是将公司徽标转换为 C 位图。再看看下图所示的示例位图：

位图被加载到位图转换器中，转换为 “最佳调色板”，然后另存为 “带调色板的 C” 文件。生成的 C 源代码如下所示 （有些数据没有显示，以节省空间）。

生成的 C 代码 （由位图转换器生成）

```
/*********************************************************************
*                SEGGER MICROCONTROLLER SYSTEME GmbH                 *
*        Solutions for real time microcontroller applications        *
*                           www.segger.com                           *
**********************************************************************
*
* C-file generated by
*
*        Bitmap converter for emWin V5.05.
*        Compiled Feb 26 2010, 14:49:28
*        (C) 1998 - 2005 Segger Microcontroller Systeme GmbH
*
**********************************************************************
*
* Source file:SeggerLogo200
* Dimensions:200 * 100
* NumColors:33
*
**********************************************************************
*/

#include <stdlib.h>

#include "GUI.h"

#ifndef GUI_CONST_STORAGE
  #define GUI_CONST_STORAGE const
#endif

/*   Palette
The following are the entries of the palette table.
Every entry is a 32-bit value (of which 24 bits are actually used)
the lower   8 bits represent the Red component,
the middle  8 bits represent the Green component,
the highest 8 bits (of the 24 bits used) represent the Blue component
as follows:   0xBBGGRR
*/

static GUI_CONST_STORAGE GUI_COLOR ColorsSeggerLogo200[] = {
     0xFFFFFF,0x353537,0x9C4B37,0xCDCDCD
    ,0x9A9A9B,0xE6D2CD,0xB57869,0x686869
    ,0xA25644,0xCEA59B,0xF9F4F3,0x424244
    ,0xECDDDA,0xF2F2F3,0xAF6D5D,0xD9D9DA
    ,0x818182,0x5B5B5D,0xB3B3B4,0x4E4E50
    ,0xD4B0A8,0x747476,0xA7A7A8,0xF3E9E6
    ,0xC79A8F,0xBB8376,0xDABCB4,0xE0C7C1
    ,0x8D8D8F,0xE6E6E6,0xA86250,0xC18F82
    ,0xC0C0C1
};

static GUI_CONST_STORAGE GUI_LOGPALETTE PalSeggerLogo200 = {
  33,/* number of entries */
  0, /* No transparency */
  &ColorsSeggerLogo200[0]
};

static GUI_CONST_STORAGE unsigned char acSeggerLogo200[] = {
  0x00, 0x00,       /* Not all data is shown in this example */
  0x00, 0x92,
.
.
.
0xC6, 0x22
```



```
0x0A, 0x22
};

extern GUI_CONST_STORAGE GUI_BITMAP bmSeggerLogo200;

GUI_CONST_STORAGE GUI_BITMAP bmSeggerLogo200 = {
  200, /* XSize */
  100, /* YSize */
  200, /* BytesPerLine */
  8, /* BitsPerPixel */
  acSeggerLogo200,  /* Pointer to picture data (indices) */
  &PalSeggerLogo200  /* Pointer to palette */
};

/* *** End of file *** */
```

压缩文件

可使用同一位图图像来创建压缩的 C 文件，只需按如前所述加载并转换位图，然后将其另存为“带调色板的 C，压缩”。源代码如下所示 （有些数据没有显示，以节省空间）。

可在文件的末尾看见压缩图像的大小，大小为 18,800 像素 3,730 字节。

生成的压缩 C 代码 （由位图转换器生成）

```
/*********************************************************************
*                SEGGER MICROCONTROLLER SYSTEME GmbH                 *
*        Solutions for real time microcontroller applications        *
*                           www.segger.com                           *
**********************************************************************
*
* C-file generated by
*
*        Bitmap converter for emWin V5.05.
*        Compiled Feb 26 2010, 14:49:28
*        (C) 1998 - 2005 Segger Microcontroller Systeme GmbH
*
**********************************************************************
*
* Source file:SeggerLogo200_comp
* Dimensions:200 * 100
* NumColors:33
*
**********************************************************************
*/

#include <stdlib.h>

#include "GUI.h"

#ifndef GUI_CONST_STORAGE
  #define GUI_CONST_STORAGE const
#endif

/*   Palette
The following are the entries of the palette table.
Every entry is a 32-bit value (of which 24 bits are actually used)
the lower   8 bits represent the Red component,
the middle  8 bits represent the Green component,
the highest 8 bits (of the 24 bits used) represent the Blue component
as follows:   0xBBGGRR
*/

static GUI_CONST_STORAGE GUI_COLOR ColorsSeggerLogo200_comp[] = {
     0xFFFFFF,0x353537,0x9C4B37,0xCDCDCD
    ,0x9A9A9B,0xE6D2CD,0xB57869,0x686869
    ,0xA25644,0xCEA59B,0xF9F4F3,0x424244
    ,0xECDDDA,0xF2F2F3,0xAF6D5D,0xD9D9DA
    ,0x818182,0x5B5B5D,0xB3B3B4,0x4E4E50
    ,0xD4B0A8,0x747476,0xA7A7A8,0xF3E9E6
    ,0xC79A8F,0xBB8376,0xDABCB4,0xE0C7C1
    ,0x8D8D8F,0xE6E6E6,0xA86250,0xC18F82
    ,0xC0C0C1
};

static GUI_CONST_STORAGE GUI_LOGPALETTE PalSeggerLogo200_comp = {
```



```
  33,/* number of entries */
  0, /* No transparency */
  &ColorsSeggerLogo200_comp[0]
};

static GUI_CONST_STORAGE unsigned char acSeggerLogo200_comp[] = {
  /* RLE:006 Pixels @ 000,000*/ 6, 0x00,
  /* RLE:188 Pixels @ 006,000*/ 188, 0x01,
.
.
.
  /* RLE:188 Pixels @ 006,099*/ 188, 0x01,
  /* RLE:006 Pixels @ 194,099*/ 6, 0x00,


  0};  /* 3730 for 20000 pixels */

extern GUI_CONST_STORAGE GUI_BITMAP bmSeggerLogo200_comp;

GUI_CONST_STORAGE GUI_BITMAP bmSeggerLogo200_comp = {
  200, /* XSize */
  100, /* YSize */
  200, /* BytesPerLine */
  GUI_COMPRESS_RLE8, /* BitsPerPixel */
  acSeggerLogo200_comp,  /* Pointer to picture data (indices) */
  &PalSeggerLogo200_comp  /* Pointer to palette */
 ,GUI_DRAW_RLE8
};

/* *** End of file *** */
```


第 11 章

颜色

emWin 支持黑色 / 白色、灰度 （具有不同强度的单色）和彩色显示器。同一用户程序可用于任何显示器；只有 LCD 显示器的配置才需要进行更改。颜色管理功能会尝试查找与要显示的任何颜色最接近的匹配颜色。

逻辑颜色是应用程序处理的颜色。逻辑颜色始终按 RGB 值进行定义，是一个 24 位的值，每种颜色包含 8 位，如下所示：

0xBBGGRR。因此，白色为 0xFFFFFF，黑色为 0x000000，鲜红色为 0xFF。

物理颜色是可用显示器实际显示的颜色。按与逻辑颜色相同的 24 位 RGB 格式进行定义。在运行时，逻辑颜色映射到物理颜色。

对于有几种颜色的显示器（如单色显示器或 8/16 色 LCD），emWin 使用优化版的“最小二乘偏差搜索”对其进行转换。它将要显示的颜色 （逻辑颜色）与 LCD 可实际显示的所有可用颜色 （物理颜色）进行比较，并使用 LCD 认为最接近的颜色。

11.1 预定义的颜色

除了自定义颜色外， emWin 中也预定义了一些标准颜色，如下表所示：

GUI_BLUE		0xFF0000
GUI_GREEN		0x00FF00
GUI_RED		0x0000FF
GUI_CYAN		0xFFFF00
GUI_MAGENTA		0xFF00FF
GUI_YELLOW		0x00FFFF
GUI_LIGHTBLUE		0xFF8080
GUI_LIGHTGREEN		0x80FF80
GUI_LIGHTRED		0x8080FF
GUI_LIGHTCYAN		0xFFFF80
GUI_LIGHTMAGENTA		0xFF80FF
GUI_LIGHTYELLOW		0x80FFFF
GUI_DARKBLUE		0x800000
GUI_DARKGREEN		0x008000
GUI_DARKRED		0x000080
GUI_DARKCYAN		0x808000
GUI_DARKMAGENTA		0x800080
GUI_DARKYELLOW		0x008080
GUI_WHITE		0xFFFFFF
GUI_LIGHTGRAY		0xD3D3D3
GUI_GRAY		0x808080
GUI_DARKGRAY		0x404040
GUI_BLACK		0x000000
GUI_BROWN		0x2A2AA5

示例

```
/* Set background color to magenta */
GUI_SetBkColor(GUI_MAGENTA);
GUI_Clear();
```

11.2 颜色条测试例程

颜色条示例程序用于显示 13 种颜色条，如下所述：

黑色 -> 红色、白色 -> 红色、黑色 -> 绿色、白色 -> 绿色、黑色 -> 蓝色、白色 -> 蓝色、黑色 -> 白色、黑色 -> 黄色、白色 -> 黄色、黑色 -> 青色、白色 -> 青色、黑色 -> 洋红和白色 -> 洋红。

此小例程可以任何颜色格式在所有显示器上使用。当然，结果取决于可显示的颜色；例程要求显示其尺寸为 320*240，以便显示所有颜色。该例程用于演示显示器不同颜色设置的效果。测试程序也可用它来验证显示器的功能，检查可用的颜色和灰度，以及校正色彩转换。这些屏幕截图取自 windows 模拟，看起来与设置和硬件工作正常时显示器的实际输出完全一样。该例程以 emWin 随附示例中的 COLOR_ShowColorBar.c 的形式提供。

11.3 固定调色板模式

下表列出了可用的固定调色板颜色模式以及创建驱动或存储设备时需要使用的必要标识符。详细信息如下所述。

标识符	可用的颜色数	掩码
GUICC_1	2（黑白）	0x01
GUICC_2	4（灰度）	0x03
GUICC_4	16（灰度）	0x0F
GUICC_5	32（灰度）	0x1F
GUICC_111	8	BGR
GUICC_M111	8	RGB
GUICC_222	64	BBGGRR
GUICC_M222	64	RRGGBB
GUICC_233	256	BBGGGRRR
GUICC_M233	256	RRGGGBBB
GUICC_323	256	BBBGGRRR
GUICC_M323	256	RRRGGBBB
GUICC_332	256	BBBGGGRR
GUICC_M332	256	RRRGGGBB
GUICC_44412	4096	0000BBBBGGGGRRRR
GUICC_M444_12	4096	0000RRRRGGGGBBBB
GUICC_444121	4096	BBBBGGGGRRRR0000
GUICC_44416	4096	0BBBB0GGGG0RRRR0
GUICC_M44416	4096	0RRRR0GGGG0BBBB0
GUICC_555	32768	0BBBBBGGGGGRRRRR
GUICC_M555	32768	0RRRRRGGGGGBBBBB
GUICC_556	65536	BBBBBGGGGGRRRRRR
GUICC_M556	65536	RRRRRGGGGGBBBBBB
GUICC_565	65536	BBBBBGGGGGGRRRRR
GUICC_M565	65536	RRRRRGGGGGGBBBBB
GUICC_655	65536	BBBBBBGGGGGRRRRR
GUICC_M655	65536	RRRRRRGGGGGBBBBB
GUICC_666	262144	BBBBBBGGGGGGRRRRRR
GUICC_M666	262144	RRRRRRGGGGGGBBBBBB
GUICC_666_9	262144	0000000BBBBBBGGG0000000GGGRRRRRR
GUICC_M666_9	262144	0000000RRRRRRGGG0000000GGGBBBBBB
GUICC_822216	256	0xFF
GUICC_84444	240	0xFF
GUICC_8666	232	0xFF
GUICC_86661	233（232 + 透明）	0xFF
GUICC_888	16777216	BBBBBBBBGGGGGGGGRRRRRRRR

标识符	可用的颜色数	掩码
GUICC_M888	16777216	RRRRRRRRGGGGGGGGBBBBBBBB
GUICC_8888	16777216 + 8 位 Alpha 混合	AAAAAAAABBBBBBBBGGGGGGGGRRRRRRRR
GUICC_M8888	16777216 + 8 位 Alpha 混合	AAAAAAAARRRRRRRRGGGGGGGGBBBBBBBB
GUICC_0	x	x

11.4 固定调色板模式的详细说明

以下为每种预定义的固定调色板模式中可用颜色的详细说明。

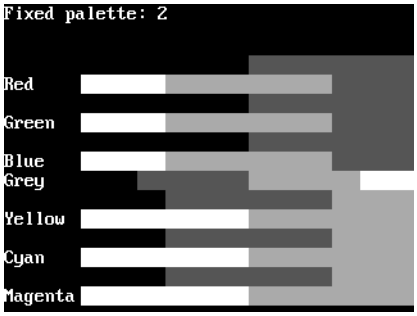
GUICC_1：1bpp （黑白）

对于每像素 1 位的单色显示器有必要使用此模式。

可用颜色：2:

GUICC_2：2 bpp （4 灰度）

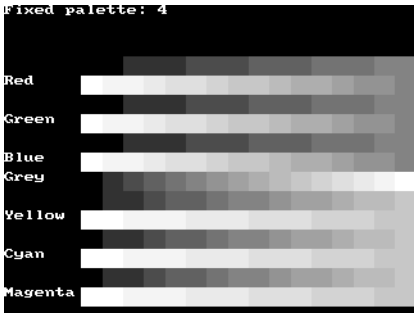
对于每像素 2 位的单色显示器有必要使用此模式。



可用颜色：2 x 2 = 4：

GUICC_4：4 bpp （16 灰度）

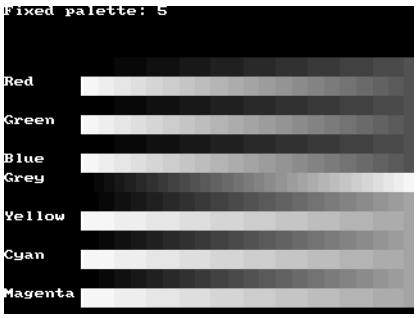
对于每像素 4 位的单色显示器有必要使用此模式。



可用颜色：2 x 2 x 2 x 2 = 16：

GUICC_5：5 bpp （32 灰度）

对于每像素 5 位的单色显示器有必要使用此模式。



可用颜色：2 x 2 x 2 x 2 x 2 = 32：

GUICC_111：3 bpp （每种颜色 2 个级别）

如果 8 种基本颜色已经足够、硬件仅支持每像素和颜色一位或没有足够的视频存储器用于更高的色彩深度，请使用此模式。
颜色掩码：BGR

可用颜色：2 x 2 x 2 = 8：

GUICC_M111:3 bpp （每种颜色 2 个级别），红色和蓝色交换

如果 8 种基本颜色已经足够、硬件仅支持每像素和颜色一位或没有足够的视频存储器用于更高的色彩深度，请使用此模式。可用的颜色与 111 模式中可用颜色相同。
颜色掩码：RGB
可用颜色：2 x 2 x 2 = 8：

GUICC_222：6 bpp （每种颜色 4 个级别）

如果硬件没有用于每种单独颜色的调色板，此模式是好选择。为每像素和每颜色保留 2 位；通常使用 1 字节存储一个像素。
颜色掩码：BBGGRR

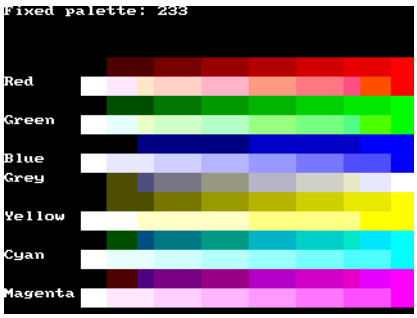
可用颜色：4 x 4 x 4 = 64：

GUICC_M222:6 bpp （每种颜色 4 个级别），红色和蓝色交换

如果硬件没有用于每种单独颜色的调色板，此模式是好选择。为每像素和每颜色保留 2 位；通常使用 1 字节存储一个像素。可用的颜色与 222 模式中可用颜色相同。
颜色掩码：RRGGBB
可用颜色：4 x 4 x 4 = 64：

GUICC_233：8 bpp

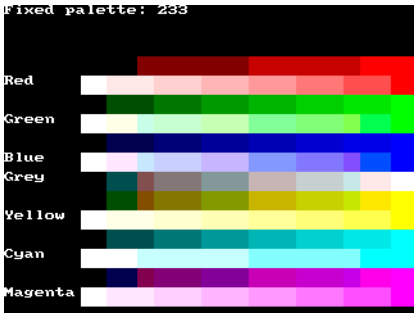
此模式支持 256 色。3 位用于颜色的红色和绿色分量，2 位用于蓝色分量。如图中所示，结果是用于绿色和红色的 8 个等级以及用于蓝色的 4 个等级。不建议使用此模式，因为它不包含真实的灰度等级。
颜色掩码：BBGGGRRR



可用颜色：4 x 8 x 8 = 256：

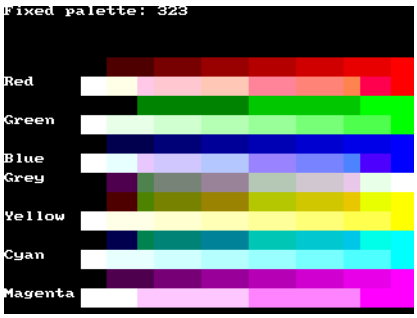
GUICC_M233：8bpp，红色和蓝色交换

此模式支持 256 色。3 位用于颜色的红色和绿色分量，2 位用于蓝色分量。结果是用于绿色和蓝色的 8 个等级和用于红色的 4 个等级。不建议使用此模式，因为它不包含真实的灰度等级。
颜色掩码：RRGGGBBB



可用颜色：4 x 8 x 8 = 256：

GUICC_323：8 bpp



此模式支持 256 色。3 位用于颜色的红色和蓝色分量，2 位用于绿色分量。如图中所示，结果是用于蓝色和红色的 8 个等级和用于绿色的 4 个等级。不建议使用此模式，因为它不包含真实的灰度等级。
颜色掩码：BBBGGRRR

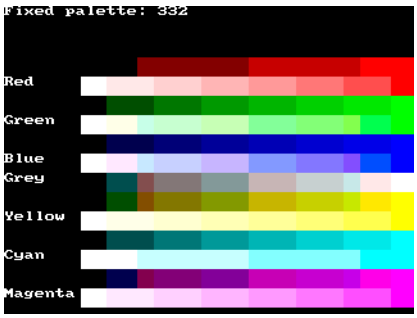
可用颜色：8 x 4 x 8 = 256：

GUICC_M323:8bpp，红色和蓝色交换

此模式支持 256 色。 3 位用于颜色的红色和蓝色分量， 2 位用于绿色分量。可用的颜色与 323 模式中可用颜色相同。结果是用于绿色和蓝色的 8 个等级和用于红色的 4 个等级。不建议使用此模式，因为它不包含真实的灰度等级。
颜色掩码：RRRGGBBB
可用颜色：8 x 4 x 8 = 256：

332 模式：8 bpp

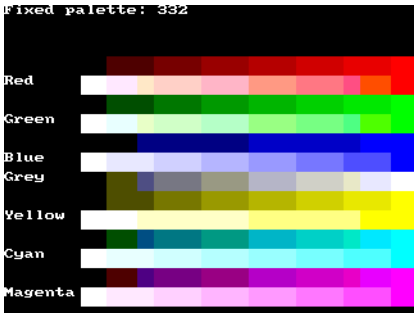
此模式支持 256 色。3 位用于颜色的蓝色和绿色分量，2 位用于红色分量。如图中所示，结果是用于绿色和蓝色的 8 个等级和用于红色的 4 个等级。不建议使用此模式，因为它不包含真实的灰度等级。
颜色掩码：BBBGGGRR



可用颜色：8 x 8 x 4 = 256：

GUICC_332：8bpp，红色和蓝色交换

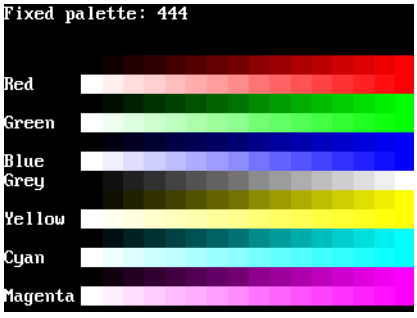
此模式支持 256 色。3 位用于颜色的红色和绿色分量，2 位用于蓝色分量。结果是用于红色和绿色的 8 个等级和用于蓝色的 4 个等级。不建议使用此模式，因为它不包含真实的灰度等级。
颜色掩码：RRRGGGBB



可用颜色：8 x 8 x 4 = 256：

GUICC_44412:

红、绿和蓝分量都是 4 位。
颜色掩码：0000BBBBGGGGRRRR
可用颜色：16 x 16 x 16 = 4096。



GUICC_44416:

红、绿和蓝分量都是4位。各颜色分量之间的一位不使用。可用的颜色与44412模式中可用颜色相同。
颜色掩码：0BBBB0GGGG0RRRR0
可用颜色：16 x 16 x 16 = 4096。

GUICC_44412：红色和蓝色交换

红、绿和蓝分量都是 4 位。可用的颜色与 44412 模式中可用颜色相同。
可用颜色：16 x 16 x 16 = 4096。
颜色掩码：RRRRGGGGBBBB

GUICC_44416：红色和蓝色交换

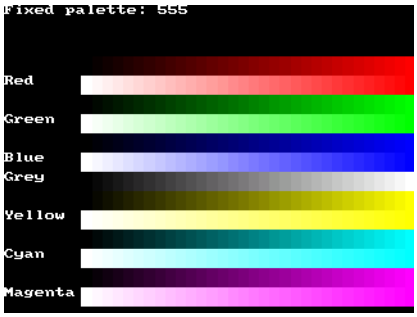
红、绿和蓝分量都是4位。各颜色分量之间的一位不使用。可用的颜色与44412模式中可用颜色相同。
颜色掩码：0RRRR0GGGG0BBBB0
可用颜色：16 x 16 x 16 = 4096。

GUICC_444121：

红、绿和蓝分量都是 4 位。颜色掩码的低 4 位不使用。可用的颜色与 44412 模式中可用颜色相同。
颜色掩码：BBBBGGGGRRRR0000
可用颜色：16 x 16 x 16 = 4096。

GUICC_555：15 bpp

对于支持色彩深度为 15bpp 的 RGB 色的显示控制器，必须使用此模式。红、绿和蓝分量都是 5 位。
颜色掩码：BBBBBGGGGGRRRRR
可用颜色：32 x 32 x 32 = 32768。

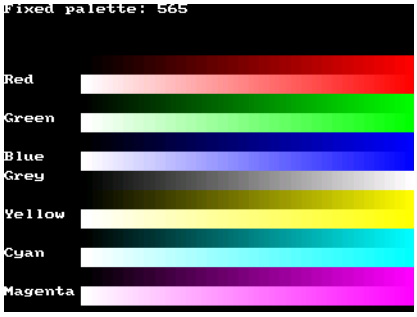


GUICC_M555:15bpp，红色和蓝色交换

对于支持色彩深度为 15bpp 的 RGB 色的显示控制器，必须使用此模式。红、绿和蓝分量都是 5 位。可用的颜色与 555 模式中可用颜色相同。
颜色掩码：RRRRRGGGGGBBBBB
可用颜色：32 x 32 x 32 = 32768。

GUICC_565：16 bpp

对于支持色彩深度为 16bpp 的 RGB 色的显示控制器，必须使用此模式。红色和绿色分量为 5 位，蓝色分量为 6 位。
颜色掩码：BBBBBGGGGGGRRRRR
可用颜色：32 x 64 x 32 = 65536。



GUICC_M565:16bpp，红色和蓝色交换

对于支持色彩深度为 16bpp 的 RGB 色的显示控制器，必须使用此模式。可用的颜色与 565 模式中可用颜色相同。
颜色序列：RRRRRGGGGGGBBBBB
可用颜色：32 x 64 x 32 = 65536。

GUICC_556：16 bpp

对于支持色彩深度为 16bpp 的 RGB 色的显示控制器，必须使用此模式。蓝色和绿色分量为 5 位，红色分量为 6 位。
颜色掩码：BBBBBGGGGGRRRRR
可用颜色：64 x 32 x 32 = 65536。

GUICC_M556：16bpp，红色和蓝色交换

对于支持色彩深度为 16bpp 的 RGB 色的显示控制器，必须使用此模式。红色和绿色分量为 5 位，蓝色分量为 6 位。
颜色掩码：RRRRRGGGGGBBBBB
可用颜色：64 x 32 x 32 = 65536。

GUICC_655：16 bpp

对于支持色彩深度为 16bpp 的 RGB 色的显示控制器，必须使用此模式。红色和绿色分量为 5 位，蓝色分量为 6 位。
颜色掩码：BBBBBBGGGGGRRRRR
可用颜色：64 x 32 x 32 = 65536。

GUICC_M655：16bpp，红色和蓝色交换

对于支持色彩深度为 16bpp 的 RGB 色的显示控制器，必须使用此模式。蓝色和绿色分量为 5 位，红色分量为 6 位。
颜色掩码：RRRRRRGGGGGBBBBB
可用颜色：64 x 32 x 32 = 65536。

GUICC_666：18 bpp

对于支持色彩深度为 18bpp 的 RGB 色的显示控制器，必须使用此模式。红、绿和蓝分量都是 6 位。
颜色掩码：BBBBBBGGGGGGRRRRRR
可用颜色：64 x 64 x 64 = 262144。

GUICC_M666：18bpp，红色和蓝色交换

对于支持色彩深度为 18bpp 的 RGB 色的显示控制器，必须使用此模式。红、绿和蓝分量都是 6 位。
颜色掩码：RRRRRRGGGGGGBBBBBB
可用颜色：64 x 64 x 64 = 262144。

GUICC_666_9：18 bpp

对于支持色彩深度为 18bpp 的 RGB 色的显示控制器，必须使用此模式。红、绿和蓝分量都是 6 位。
颜色掩码：0000000BBBBBBGGG0000000GGGRRRRRR
可用颜色：64 x 64 x 64 = 262144。

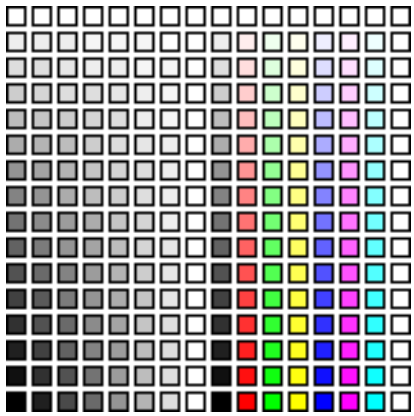
GUICC_M666_9：18bpp，红色和蓝色交换

对于支持色彩深度为 18bpp 的 RGB 色的显示控制器，必须使用此模式。红、绿和蓝分量都是 6 位。
颜色掩码：RRRRRRGGGGGGBBBBBB
可用颜色：64 x 64 x 64 = 262144。

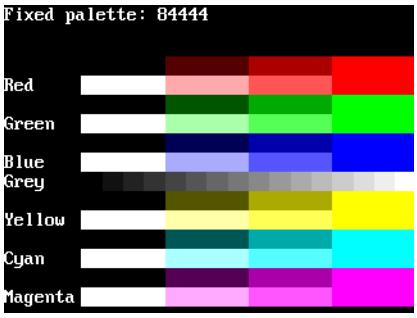
GUICC_822216：8bpp，每种颜色 2 个级别 + 8 个灰度 + 16 个级别的 Alpha 混合

此模式可与可编程色彩查询表 (LUT) 一起使用，支持总共 256 种可能颜色并支持 Alpha 混合。对于每种 color / grayscale，它支持 8 种基本颜色、 8 个灰度和 16 个级别的 Alpha 混合。换句话说，如果仅要求几种颜色，但要求更多级别的 Alpha 混合时，可使用它。

可用颜色：(2 x 2 x 2 + 8) * 16 = 256



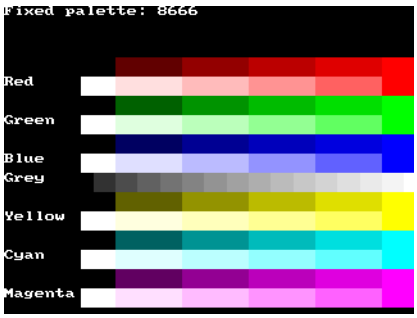
GUICC_84444：8bpp，每种颜色 4 个级别 + 16 个灰度 + 4(3) 个级别的 Alpha 混合



此模式可与可编程色彩查询表 (LUT) 一起使用，支持总共 256 种可能颜色并支持 Alpha 混合。每种 color / grayscale 除了有 16 个灰度和 4 个级别的 Alpha 混合外，每种颜色还有 4 个强度级别可用。换句话说，如果仅要求几个级别的 Alpha 混合和不同颜色阴影，可使用它。

可用颜色：(4 x 4 x 4 + 16) * 3 = 240

GUICC_8666：8bpp，每种颜色 6 个级别 + 16 个灰度



此模式最常与可编程色彩查询表 (LUT) 一起使用，支持总共 256 种使用调色板的可能颜色。所示屏幕截图为可用颜色的一个大概；此模式包含用于常规应用程序的最佳选择。除 16 个灰度外，每种颜色还有六个级别的强度可用。

可用颜色：6 x 6 x 6 + 16 = 232：

GUICC_86661:8bpp，每种颜色 6 个级别 + 16 个灰度 + 透明性

此模式最常与多层配置和可编程色彩查询表 (LUT) 一起使用，支持总共 256 种使用调色板的可能颜色。 8666 和 86661 的不同之处在于 86661 模式的第一个颜色索引不使用。因此色彩转换例程 GUI_Color2Index 从不返回用于透明性的 0。
可用颜色：6 x 6 x 6 + 16 = 232。

GUICC_888:24 bpp

对于支持色彩深度为 24bpp 的 RGB 色的显示控制器，必须使用此模式。红、绿和蓝分量都是 8 位。
颜色掩码：BBBBBBBBGGGGGGGGRRRRRRRR
可用颜色：256 x 256 x 256 = 16777216。

GUICC_M888:24bpp，红色和蓝色交换

对于支持色彩深度为 24bpp 的 RGB 色的显示控制器，必须使用此模式。红、绿和蓝分量都是 8 位。
颜色掩码：RRRRRRRRGGGGGGGGBBBBBBBB
可用颜色：256 x 256 x 256 = 16777216。

GUICC_8888：32 bpp

对于支持色彩深度为 32bpp 的 RGB 色的显示控制器，必须使用此模式。其中低位 3 字节用于颜色分量，高位字节用于 Alpha 混合。红、绿、蓝和 Alpha 混合分量都是 8 位。
颜色掩码：AAAAAAAABBBBBBBBGGGGGGGGRRRRRRRR
可用颜色：256 x 256 x 256 = 16777216。

GUICC_M8888：32bpp，红色和蓝色交换

对于支持色彩深度为 32bpp 的 RGB 色的显示控制器，必须使用此模式。其中低位 3 字节用于颜色分量，高位字节用于 Alpha 混合。红、绿、蓝和 Alpha 混合分量都是 8 位。
颜色掩码：AAAAAAAARRRRRRRRGGGGGGGGBBBBBBBB
可用颜色：256 x 256 x 256 = 16777216。

GUICC_0：定制调色板模式

（本章稍后将详细说明）

11.5 应用程序定义的色彩转换

如果没有固定调色板模式与色彩转换的需求相匹配，则利用此模式就可使用应用程序定义的色彩转换例程。使用这些例程的目的是将 RGB 值转换为硬件的索引值，反之亦然。

定义定制色彩转换例程的示例

以下示例说明了其工作原理：

```
static unsigned _Color2Index_User(LCD_COLOR Color) {
  unsigned Index;
  /* Add code for converting the RGB value to an index value for the hardware */
  return Index;
}

static LCD_COLOR _Index2Color_User(unsigned Index) {
  LCD_COLOR Color;
  /* Add code for converting the index value into an RGB value */
  return Color;
}

static unsigned _GetIndexMask_User(void) {
  return 0xffff; /* Example for using 16 bits */
}

const LCD_API_COLOR_CONV LCD_API_ColorConv_User = {
  _Color2Index_User,
  _Index2Color_User,
  _GetIndexMask_User
};
```

如果要将 RGB 值转换为显示控制器的索引值，则 emWin 调用函数 LCD_Color2Index_User()，而要将索引值转换为 RGB 值时， emWin 调用函数 LCD_Index2Color_User()。
LCD_GetIndexMask_User() 将返回一个位掩码值，将显示控制器所用的每个位都设置为 1，未使用的位设置为 0。例如， GUICC_44416 模式的索引掩码为 0BBBB0GGGG0RRRR0，其中 0 代表未使用的位。此模式的位掩码为 0x7BDE。

使用定制色彩转换例程的示例

如“配置”一章中所述，创建显示器驱动设备需要一个指向 API 表的指针。如上例所示，API 表由指向色彩转换例程的函数指针组成。
API 表和色彩转换例程的理想位置在 LCDConf.c 文件夹的配置文件 Config 中。这些例程可用于负责创建显示器驱动设备的函数 LCD_X_Config() 中，如下所示：

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_16, &LCD_API_ColorConv_User, 0, 0);
  .
  .
  .
}
```


11.6 定制调色板模式

如果没有固定调色板模式能满足应用程序的要求，则 emWin 可使用定制调色板。定制调色板只是按顺序列出了所有可用的颜色，该顺序与硬件使用这些颜色时的顺序相同。这意味着无论 LCD 控制器 / 显示器组合可显示什么颜色， emWin 都将能在 PC 模拟中模拟它们，并在目标系统中正确处理这些颜色。使用定制调色板要求色彩深度 <= 8bpp。
定制调色板通常在负责创建并配置显示器驱动设备的函数 LCD_X_Config() 初始化期间使用。

示例

以下示例显示了定制调色板的使用方法。它将调色板传递给函数：

```
static const LCD_COLOR _aColors_16[] = {
  0x000000,  0x0000FF,  0x00FF00,  0x00FFFF,
  0xFF0000,  0xFF00FF,  0xFFFF00,  0xFFFFFF,
  0x000000,  0x000080,  0x008000,  0x008080,
  0x800000,  0x800080,  0x808000,  0x808080,
};

static const LCD_PHYSPALETTE _aPalette_16 = {
  COUNTOF(_aColors_16), _aColors_16
};

void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  .
  .
  .
  //
  // Set user palette data (only required if no fixed palette is used)
  //
  LCD_SetLUTEx(0, _aPalette_16);
}
```

11.7 颜色 API

下表按字母顺序在各自类别中列出了可用的颜色相关函数，各例程将在后续章节中详细描述。

例程	描述
基本颜色函数	
GUI_GetBkColor()	返回当前背景色。
GUI_GetBkColorIndex()	返回当前背景色的索引。
GUI_GetColor()	返回当前前景色。
GUI_GetColorIndex()	返回当前前景色的索引。
GUI_SetBkColor()	设置当前背景色。
GUI_SetBkColorIndex()	设置当前背景色的索引。
GUI_SetColor()	设置当前前景色。
GUI_SetColorIndex()	设置当前前景色的索引。
索引和色彩转换	
GUI_CalcColorDist()	返回 2 种颜色之间的差值
GUI_CalcVisColorError()	将差值返回给下一可用颜色
GUI_Color2Index()	将色彩转换为颜色索引。
GUI_Color2VisColor()	返回最接近的可用颜色
GUI_ColorIsAvailable()	检查给定的颜色是否可用
GUI_Index2Color()	将颜色索引转换为颜色。

11.8 基本颜色函数

GUI_GetBkColor()

描述
返回当前背景色。

原型
```
GUI_COLOR GUI_GetBkColor(void);
```

返回值
当前背景色。

GUI_GetBkColorIndex()

描述
返回当前背景色的索引。

原型
```
int GUI_GetBkColorIndex(void);
```

返回值
当前背景色索引。

GUI_GetColor()

描述
返回当前前景色。

原型
```
GUI_COLOR GUI_GetColor(void);
```

返回值
当前前景色。

GUI_GetColorIndex()

描述
返回当前前景色的索引。

原型
```
int GUI_GetColorIndex(void);
```

返回值
当前前景色索引。

GUI_SetBkColor()

描述

设置当前背景色。

原型

```
GUI_COLOR GUI_SetBkColor(GUI_COLOR Color);
```

参数	描述
Color	背景的颜色， 24 位 RGB 值。

返回值

选定的背景色。

GUI_SetBkColorIndex()

描述

设置当前背景色的索引。

原型

```
int GUI_SetBkColorIndex(int Index);
```

参数	描述
Index	要使用颜色的索引。

返回值

选定的背景色索引。

GUI_SetColor()

描述

设置当前前景色。

原型

```
void GUI_SetColor(GUI_COLOR Color);
```

参数	描述
Color	前景的颜色， 24 位 RGB 值。

返回值

选定的前景色。

GUI_SetColorIndex()

描述

设置当前前景色的索引。

原型

```
void GUI_SetColorIndex(int Index);
```

参数	描述
Index	要使用颜色的索引。

返回值

选定的前景色索引。

11.9 索引和色彩转换

GUI_CalcColorDist()

计算 2 种颜色之间的差值。差值应通过红色、绿色和蓝色分量的差值平方值相加进行计算：
差值 = $(Red1 - Red0)^2 + (Green1 - Green0)^2 + (Blue1 - Blue0)^2$

原型

```
U32 GUI_CalcColorDist(GUI_COLOR Color0, GUI_COLOR Color1))
```

参数	描述
Color0	第一种颜色的 RGB 值。
Color1	第二种颜色的 RGB 值。

返回值

如上所述的差值。

GUI_CalcVisColorError()

计算与下一可用颜色的差值。有关拖放的详细信息，请参阅“GUI_CalcColorDist()”（第 244 页）。

原型

```
U32 GUI_CalcVisColorError(GUI_COLOR color)
```

参数	描述
Color	要计算的颜色的 RGB 值。

返回值

到下一可用颜色的差值。

GUI_Color2Index()

返回指定 RGB 颜色值的索引。

原型

```
int GUI_Color2Index(GUI_COLOR Color)
```

参数	描述
Color	要转换颜色的 RGB 值。

返回值

颜色索引。

GUI_Color2VisColor()

将系统的下一可用颜色作为 RGB 颜色值返回。

原型

```
GUI_COLOR GUI_Color2VisColor(GUI_COLOR color)
```

参数	描述
Color	颜色的 RGB 值。

返回值

最接近的可用颜色的 RGB 颜色值。

GUI_ColorIsAvailable()

检查给定的颜色是否可用。

原型

```
char GUI_ColorIsAvailable(GUI_COLOR color)
```

参数	描述
Color	颜色的 RGB 值。

返回值

颜色可用时为 1，否则为 0。

GUI_Index2Color()

返回指定索引的 RGB 颜色值。

原型

```
int GUI_Index2Color(int Index)
```

参数	描述
Index	要转换颜色的索引

返回值

RGB 颜色值。

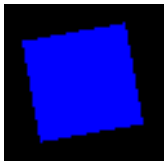
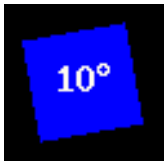
第 12 章

存储设备

存储设备可在各种情况下使用，主要用于防止在绘制重叠项目时出现显示器闪烁。其基本思想很简单。不使用存储设备时，绘制操作直接写入显示器。屏幕在执行绘制操作时随时更新，从而在进行各种更新时使屏幕闪烁。例如，如果要在背景中绘制一个位图，在前景中绘制一些透明文本，应首先绘制位图，然后绘制文本。效果将是文本出现闪烁。
但是，如果在此过程中使用存储设备，则所有绘制操作都在存储器中执行。仅在所有操作都完成后才将最终结果显示在屏幕上，其优点是没有闪烁。在下节的示例中可以看到这种差异，该示例展示了在使用和不使用存储设备时的一系列绘制操作。
差别可归纳如下：如果不使用存储设备，则可以看到一步步的绘制操作效果，缺点是会出现显示器闪烁。使用存储设备时，一次可见到所有例程的效果，就象单次操作一样，不能实际看见中间步骤。如上所述，其优点是完全消除了显示器的闪烁，这也是通常所期望的。
存储设备是附加的 （可选）软件项目，不随 emWin 基本包一起发货。存储设备的软件位于子目录 GUI\Memdev 中。

12.1 使用存储设备：图示

下表所示为同一操作在使用和不使用存储设备时的屏幕截图。两种情况下的目的是相同的：旋转工件并分别标志旋转角度（此处为 10 度）。第一种情况下（不使用存储设备），屏幕必须清除，然后在新位置重绘多边形，并写入带新标志的字符串。第二种情况下（使用存储设备），在存储器中执行相同的操作，但屏幕此时不更新。仅在调用 GUI_MEMDEV_CopyToLCD() 例程时出现更新，并且此更新一次就反映所有操作。请注意，这两种操作步骤的初始状态和最终输出是相同的。

API 函数	无存储设备	有存储设备
第 0 步：初始状态		
第 1 步：GUI_Clear()		
第 2 步：GUI_DrawPolygon()		
第 3 步：GUI_DispString()		
第 4 步：GUI_MEMDEV_CopyToLCD()（仅在使用存储设备时）		

12.2 支持的色彩深度 (bpp)

存储设备有 4 种不同的色彩深度可用：
1bpp、8bpp、16bpp 和 32bpp。

创建与显示器 “兼容”的存储设备

有两种创建存储设备的方法。如果要避免闪烁，则应创建与显示器兼容的存储设备。“兼容”的存储设备必须具有与显示器相同或更高的色彩深度。如果使用了函数 GUI_MEMDEV_Create()、GUI_MEMDEV_CreateEx()，则 emWin 会自动为显示器选择 “正确”类型的存储设备。
窗口管理器也能为系统中的某些或全部窗口使用存储设备，同样也使用这些函数。
这样，会自动使用具有最低色彩深度 （使用最少存储量）的存储设备。

创建用于其他用途的存储设备

任何类型的存储设备都可使用 GUI_MEMDEV_CreateFixed() 创建，典型应用是使用存储设备进行打印，如本章稍后所述。

12.3 存储设备和窗口管理器

窗口管理器可与存储设备完美搭配。每个窗口都有一个标记，告诉窗口管理器是否应使用存储设备进行渲染。此标记可以在创建窗口时指定，也可在任何时候进行设置 / 重设。
如果为特定窗口设置了存储设备标记，则 WM 在绘制窗口时自动使用存储设备。它会在绘制窗口前创建一个存储设备，然后在绘制操作完成后将其删除。如果有足够的内存可用，会将整个窗口装入 WM 所创建存储设备的内存中。如果没有足够的内存可用于将整个窗口装入一个存储设备中，则 WM 使用 “分段”来绘制窗口。有关 “分段”的详细信息，在文档的 “存储设备 \ 分段存储设备”一章中说明。用于绘制操作的内存仅在绘制操作期间分配。如果在绘制 （重绘）窗口时没有足够的内存可用，则不使用存储设备重绘窗口。

12.4 存储设备和多层

存储设备 API 函数没有指定层的选项。请注意，在创建存储设备时，存储设备与当前选定的层相关联。存储设备也自动使用当前选定层的色彩转换设置。

示例

```
//
// Create a memory device associated with layer 1
//
GUI_SelectLayer(1);
hMem = GUI_MEMDEV_Create(0, 0, 100, 100);
GUI_MEMDEV_Select(hMem);
GUI_DrawLine(0, 0, 99, 99);
GUI_MEMDEV_Select(0);
//
// Select layer 0
//
GUI_SelectLayer(0);
//
// The following line copies the memory device to layer 1 and not to layer 0
//
GUI_MEMDEV_CopyToLCD(hMem);
```

12.5 内存要求

创建存储设备时，要求的字节数取决于存储设备的色彩深度以及是否需要支持透明。

无透明支持的内存使用情况

下表所示为无透明支持的存储设备取决于系统色彩深度的内存要求。

存储设备的色彩深度	系统色彩深度 (LCD_BITSPERPIXEL)	内存使用
1 bpp	1 bpp	1 字节 / 8 像素： (XSIZE + 7) / 8 * YSIZE
8 bpp	2、4 和 8bpp	XSIZE * YSIZE
16 bpp	12 和 16bpp	2 字节 / 像素： XSIZE * YSIZE * 2
32 bpp	18、24 和 32bpp	4 字节 / 像素： XSIZE * YSIZE * 4

示例：

要创建 X 方向 111 像素、Y 方向 33 像素的存储设备，它还应兼容使用 12bpp 色彩深度的显示器并支持透明。所需的字节数计算如下：

所需字节数 = (111 * 2 + (111 + 7) / 8) * 33 = 7788 字节

有透明支持的内存使用情况

如果存储设备要支持透明性，则每 8 像素需要一个额外字节来进行内部管理。

存储设备的色彩深度	系统色彩深度 (LCD_BITSPERPIXEL)	内存使用
1 bpp	1 bpp	2 字节 /8 像素： (XSIZE + 7) / 8 * YSIZE * 2
8 bpp	2、4 和 8bpp	1 字节 / 像素 + 1 字节 /8 像素： (XSIZE + (XSIZE + 7) / 8) * YSIZE
16 bpp	12 和 16bpp	2 字节 / 像素 + 1 字节 /8 像素： (XSIZE * 2 + (XSIZE + 7) / 8) * YSIZE
32 bpp	18、24 和 32bpp	4 字节 / 像素 + 1 字节 /8 像素： (XSIZE * 4 + (XSIZE + 7) / 8) * YSIZE

示例：

要创建 X 方向 200 像素、Y 方向 50 像素的存储设备，它还应兼容 4bpp 色彩深度的显示器并支持透明。所需的字节数计算如下：

所需字节数 = (200 + (200 + 7) / 8) * 50 = 11250 字节

12.6 性能

使用存储设备通常不会显著影响性能。使用存储设备时，驱动的工作就更容易了：只需将位图传输给显示控制器。在具有较慢驱动的系统中 （例如通过串口连接的显示器），如果使用存储设备性能会更高；在具有快速驱动的系统中 （如内存映射的显示存储器、 GUIDRV_Lin 以及其他），使用存储设备会牺牲某些性能。
如果需要 “分段”，则绘制窗口所用的时间将随着分段数增加而增加。存储设备可用的内存越多，性能也越好。

12.7 基本函数

以下例程是在使用存储设备时通常会调用的，基本用法非常简单：

1. 创建存储设备 （使用 GUI_MEMDEV_Create()）。
2. 激活它 （使用 GUI_MEMDEV_Select()）。
3. 执行绘制操作。
4. 将结果复制到显示器中 （使用 GUI_MEMDEV_CopyToLCD()）。
5. 不再需要它时，删除该存储设备 （使用 GUI_MEMDEV_Delete()）。

12.8 使用存储设备的准备操作 ...

存储设备默认是启用的。为了优化软件的性能，可在配置文件 GUIConf.h 中包含以下行，关闭对存储设备的支持：

```
#define GUI_SUPPORT_MEMDEV        0
```

如果配置文件中有此行，而你想使用存储设备，则需要删除该行或者将定义改为 1。

12.9 多层 / 多显示配置

如本章之前部分所述，与显示器“兼容”的存储设备需要具有与显示器相同或更高的色彩深度。创建与显示器兼容的存储设备时， emWin “知道”当前选定层 / 显示器的色彩深度，并自动使用最低的色彩深度。

12.10 配置选项

类型	宏	默认值	描述
B	GUI_USE_MEMDEV_1BPP_FOR_SCREEN	1	启用 1bpp 存储设备的使用，显示器色彩深度 1bpp。

12.10.1 GUI_USE_MEMDEV_1BPP_FOR_SCREEN

在显示器色彩深度 <= 8bpp 的系统中，与显示器兼容的存储设备的默认色彩深度为 8bpp。要对色彩深度为 1bpp 的显示器启用 1bpp 存储设备，应将以下行添加到配置文件 GUIConf.h 中：

```
#define GUI_USE_MEMDEV_1BPP_FOR_SCREEN 0
```

12.11 存储设备 API

下表列出了 emWin 存储设备 API 的可用例程。
所有功能都在其各自类别中按字母顺序列出。各例程将在后续章节中详细描述。

例程	描述
基本函数	
GUI_MEMDEV_Clear()	将存储设备内容标志为未更改。
GUI_MEMDEV_CopyFromLCD()	将 LCD 内容复制到存储设备。
GUI_MEMDEV_CopyToLCD()	将存储设备内容复制到 LCD。
GUI_MEMDEV_CopyToLCDAA()	复制存储设备内容并抗锯齿处理
GUI_MEMDEV_CopyToLCDAt()	复制存储设备内容到 LCD 的指定位置
GUI_MEMDEV_Create()	创建存储设备 （第一步）
GUI_MEMDEV_CreateEx()	创建具有额外创建标记的存储设备
GUI_MEMDEV_CreateFixed()	创建具有给定色彩深度的存储设备
GUI_MEMDEV_Delete()	释放存储设备使用的内存

例程	描述
GUI_MEMDEV_DrawPerspectiveX()	将透视变形的给定存储设备绘制到当前选定设备
GUI_MEMDEV_GetDataPtr()	返回指向数据直接操作区域的指针
GUI_MEMDEV_GetXSize()	返回存储设备的 X 尺寸 （宽度）
GUI_MEMDEV_GetYSize()	返回存储设备的 Y 尺寸 （高度）
GUI_MEMDEV_MarkDirty()	将某矩形区域标志为脏污。
GUI_MEMDEV_ReduceYSize()	减小存储设备的 Y 尺寸
GUI_MEMDEV_Rotate()	旋转并缩放存储设备，并将结果写入使用 “最近邻居”法的存储设备
GUI_MEMDEV_RotateHQ()	旋转并缩放存储设备，并将结果写入使用“高品质”法的存储设备
GUI_MEMDEV_Select()	选择作为绘制操作目标的存储设备
GUI_MEMDEV_SetOrg()	改变存储设备在 LCD 上的原点
GUI_MEMDEV_Write()	将存储设备的内容写入存储设备
GUI_MEMDEV_WriteAlpha()	将存储设备的内容写入使用 Alpha 混合处理的存储设备
GUI_MEMDEV_WriteAlphaAt()	将存储设备的内容写入使用给定位置和Alpha混合处理的存储设备
GUI_MEMDEV_WriteAt()	将存储设备的内容写入存储设备的给定位置
GUI_MEMEDV_WriteEx()	将存储设备的内容写入使用 Alpha 混合处理和缩放的存储设备
GUI_MEMDEV_WriteExAt()	将存储设备的内容写入使用 Alpha 混合处理和缩放的存储设备的给定位置
GUI_SelectLCD()	选择作为绘制操作目标的 LCD
分段存储设备	
GUI_MEMDEV_Draw()	使用存储设备进行绘制
自动设备对象函数	
GUI_MEMDEV_CreateAuto()	创建自动设备对象
GUI_MEMDEV_DeleteAuto()	删除自动设备对象
GUI_MEMDEV_DrawAuto()	使用 GUI_AUTODEV 对象进行绘制
测量设备对象函数	
GUI_MEASDEV_ClearRect()	清除测量矩形
GUI_MEASDEV_Create()	创建测量设备
GUI_MEASDEV_Delete()	删除测量设备
GUI_MEASDEV_GetRect()	检索测量结果
GUI_MEASDEV_Select()	选择测量设备，作为绘制操作的目标
动画函数	
GUI_MEMDEV_FadeDevices()	执行从一个存储设备到另一存储设备的淡变。
GUI_MEMDEV_SetAnimationCallback()	设置在处理动画时要调用的用户定义函数。
动画函数 （需要使用窗口管理器）	
GUI_MEMDEV_FadeInWindow()	通过减小 Alpha 值淡入窗口
GUI_MEMDEV_FadeOutWindow()	通过增加 Alpha 值淡出窗口
GUI_MEMDEV_MoveInWindow()	通过放大 （也可选择使用旋转）将窗口从指定位置移动到其实际位置
GUI_MEMDEV_MoveOutWindow()	通过缩小 （也可选择使用旋转）将窗口从其实际位置移动到指定位置
GUI_MEMDEV_ShiftInWindow()	在指定方向移动窗口，至屏幕中其实际位置处。
GUI_MEMDEV_ShiftOutWindow()	在指定方向移动窗口，从其实际位置移出屏幕。
GUI_MEMDEV_ShiftWindow()	在指定方向移动窗口，至屏幕中其实际位置处，并移出相应窗口区域中的旧内容。

12.12 基本函数

GUI_MEMDEV_Clear()

描述

将存储设备的整个内容标志为 “未更改”。

原型

```
void GUI_MEMDEV_Clear(GUI_MEMDEV_Handle hMem);
```

参数	描述
hMem	存储设备的句柄。

其他信息

使用 GUI_MEMDEV_CopyToLCD() 的下一绘制操作仅会写入在 GUI_MEMDEV_Clear() 和 GUI_MEMDEV_CopyToLCD() 之间修改的字节数。

GUI_MEMDEV_CopyFromLCD()

描述

将存储设备的内容从 LCD 数据（视频存储器）复制到存储设备。换句话说：将 LCD 的内容读回存储设备。

原型

```
void GUI_MEMDEV_CopyFromLCD(GUI_MEMDEV_Handle hMem);
```

参数	描述
hMem	存储设备的句柄。

GUI_MEMDEV_CopyToLCD()

描述

将存储设备的内容从内存复制到 LCD。

原型

```
void GUI_MEMDEV_CopyToLCD(GUI_MEMDEV_Handle hMem);
```

参数	描述
hMem	存储设备的句柄。

其他信息

请勿在窗口管理器调用的着色回调函数中使用此函数，因为它将禁用窗口管理器的裁剪区域。应使用函数 GUI_MEMDEV_WriteAt 代替。

GUI_MEMDEV_CopyToLCDAA()

描述

将存储设备 （已抗锯齿处理）的内容从内存复制到 LCD。

原型

```
void GUI_MEMDEV_CopyToLCDAA(GUI_MEMDEV_Handle MemDev);
```

参数	描述
hMem	存储设备的句柄。

其他信息

设备数据被作为抗锯齿数据处理，2x2 像素阵列转换为 1 像素。所得到像素的强度取决于阵列中设置了多少像素。

示例

创建存储设备并选择它用于输出，然后设置大字体并将文本写入存储设备：

```
GUI_MEMDEV_Handle hMem = GUI_MEMDEV_Create(0,0,60,32);
GUI_MEMDEV_Select(hMem);
GUI_SetFont(&GUI_Font32B_ASCII);
GUI_DispString("Text");
GUI_MEMDEV_CopyToLCDAA(hMem);
```

上述示例的屏幕截图

GUI_MEMDEV_CopyToLCDAt()

描述

将存储设备的内容从内存复制到 LCD 的指定位置。

原型

```
void GUI_MEMDEV_CopyToLCDAt(GUI_MEMDEV_Handle hMem, int x, int y)
```

参数	描述
hMem	存储设备的句柄。
x	X 方向位置。
y	Y 方向位置。

GUI_MEMDEV_Create()

描述

创建存储设备。

原型

```
GUI_MEMDEV_Handle GUI_MEMDEV_Create(int x0, int y0, int XSize, int YSize)
```

参数	描述
x0	存储设备的 X 位置。
y0	存储设备的 Y 位置。
xsize	存储设备的 X 尺寸。
ysize	存储设备的 Y 尺寸。

返回值

已创建存储设备的句柄。例程失败时返回值 0。

GUI_MEMDEV_CreateEx()

描述

创建存储设备。

原型

```
GUI_MEMDEV_Handle GUI_MEMDEV_CreateEx(int x0, int y0,
                                      int XSize, int YSize
                                      int Flags))
```

参数	描述
x0	存储设备的 X 位置。
y0	存储设备的 Y 位置。
xsize	存储设备的 X 尺寸。
ysize	存储设备的 Y 尺寸。
Flags	（见下表）。

参数 Flags 的允许值	
GUI_MEMDEV_HASTRANS (recommended)	默认：使用透明性标记创建存储设备，该标记确保正确绘制背景。
GUI_MEMDEV_NOTRANS	创建存储设备，无透明性。用户必须确保正确绘制背景。这样可将存储设备用于非矩形区域。另一优势是速度较高：使用此标记可加速存储设备约 30 - 50%。

返回值

已创建存储设备的句柄。例程失败时返回值 0。

GUI_MEMDEV_CreateFixed()

描述

创建固定尺寸、色彩深度 (bpp) 和指定色彩转换的存储设备。

原型

```
GUI_MEMDEV_Handle GUI_MEMDEV_CreateFixed(
                                  int x0, int y0, int xsize, int ysize,
                                  int Flags,
                                  const tLCDDEV_APIList     * pMemDevAPI,
                                  const LCD_API_COLOR_CONV * pColorConvAPI)
```

参数	描述
x0	存储设备的 X 位置。
y0	存储设备的 Y 位置。
xsize	存储设备的 X 尺寸。
ysize	存储设备的 Y 尺寸。
Flags	（见下表）。
pMemDevAPI	（见下表）。
pColorConvAPI	（见下表）。

参数 Flags 的允许值	
GUI_MEMDEV_HASTRANS (recommended)	默认：使用透明性标记创建存储设备，该标记确保正确绘制背景。
GUI_MEMDEV_NOTRANS	创建存储设备，无透明性。用户必须确保正确绘制背景。这样可将存储设备用于非矩形区域。另一优势是速度较高：使用此标记可加速存储设备约 30 - 50%。

参数 pMemDevAPI	
定义存储设备的色彩深度，单位 bpp。存储设备的色彩深度应等于或大于色彩转换例程要求的位数。例如，色彩转换 (GUI_COLOR_CONV_1) 为 1bpp 的存储设备要求色彩深度至少 1bpp 的存储设备。可用的存储设备为 1bpp、8bpp、16bpp 和 32bpp 存储设备，因此应使用 1bpp 的存储设备。 如果使用每像素 4 位的色彩转换 (GUI_COLOR_CONV_4)，则至少需要 4bpp 的存储设备。在此例中，应使用 8bpp 的存储设备。	
允许值	
GUI_MEMDEV_APILIST_1	创建色彩深度为 1bpp 的存储设备 （每 8 像素 1 字节）用于指定色彩转换要求 1bpp 时。
GUI_MEMDEV_APILIST_8	创建色彩深度为 8bpp 的存储设备 （每像素 1 字节）用于指定色彩转换要求 8bpp 或更低时。
GUI_MEMDEV_APILIST_16	创建色彩深度为 16bpp 的存储设备 （每像素 1 U16）用于指定色彩转换要求高于 8bpp 时 （高彩模式）。
GUI_MEMDEV_APILIST_32	创建色彩深度为 32bpp 的存储设备 （每像素 1 U32）用于指定色彩转换要求高于 16bpp 时 （真彩模式）。

参数 pColorConvAPI	
此参数定义所需的色彩转换。有关所使用的每像素位数和色彩转换的详细信息，请参阅 “颜色” （第 227 页）。	
允许值	
GUI_COLOR_CONV_1	与固定调色板模式 1 （黑 / 白）相同的色彩转换。
GUI_COLOR_CONV_2	与固定调色板模式 2 （4 灰度）相同的色彩转换。
GUI_COLOR_CONV_4	与固定调色板模式 4 （16 灰度）相同的色彩转换。
GUI_COLOR_CONV_565	与固定调色板模式 565 相同的色彩转换。
GUI_COLOR_CONV_M565	与固定调色板模式 M565 相同的色彩转换。
GUI_COLOR_CONV_8666	与固定调色板模式 8666 相同的色彩转换。
GUI_COLOR_CONV_888	与固定调色板模式 888 相同的色彩转换。
GUI_COLOR_CONV_8888	与固定调色板模式 8888 相同的色彩转换。

返回值

已创建存储设备的句柄。例程失败时返回值 0。

其他信息

如果要创建具有指定色彩转换的存储设备，可使用此函数。例如，要在打印设备上打印在某些项目时，可使用它。文件夹 Sample 包含的示例代码 MEMDEV_Printing.c 显示了如何使用该函数，以 1bpp 色彩转换模式打印一些项目。

示例

下例所示为如何创建具有 1bpp 色彩深度的存储设备：

```
GUI_MEMDEV_Handle hMem;
hMem = GUI_MEMDEV_CreateFixed(0, 0, 128, 128, 0,
                              GUI_MEMDEV_APILIST_1, /* Used API list */
                             GUI_COLOR_CONV_1);    /* Black/white color conversion */
GUI_MEMDEV_Select(hMem);
```

GUI_MEMDEV_Delete()

描述

删除存储设备。

原型

```
void GUI_MEMDEV_Delete(GUI_MEMDEV_Handle MemDev);
```

参数	描述
hMem	存储设备的句柄。

返回值

已删除存储设备的句柄。

GUI_MEMDEV_DrawPerspectiveX()

描述

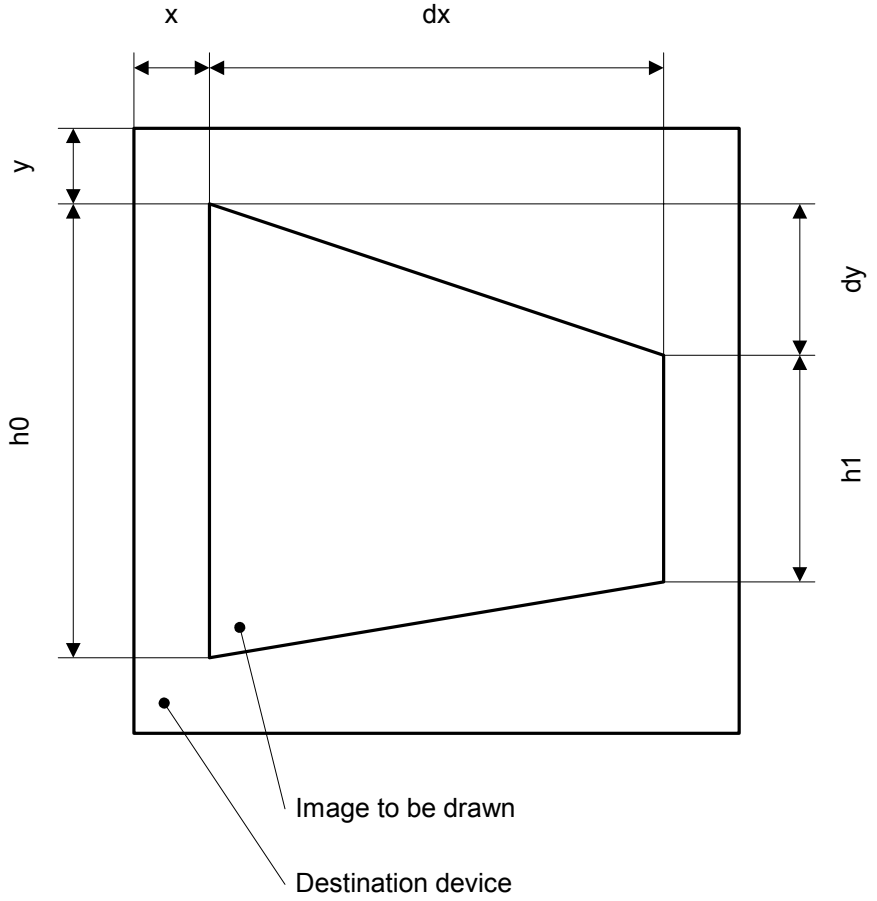
将透视变形的给定存储设备绘制到当前选定设备中。

原型

```
void GUI_MEMDEV_DrawPerspectiveX(GUI_MEMDEV_Handle hMem, int x, int y,
                                 int h0, int h1, int dx, int dy);
```

参数	描述
hMem	带有要绘制图像的源存储设备的句柄。
x	水平起始位置，单位像素。
y	垂直起始位置，单位像素。
h0	要绘制图像的最左边的高度。
h1	要绘制图像的最右边的高度。
dx	要绘制图像的宽度。
dy	y 方向上位置，从右侧的最上像素到左侧的最上像素。

下图更详细地描述了这些参数：



其他信息

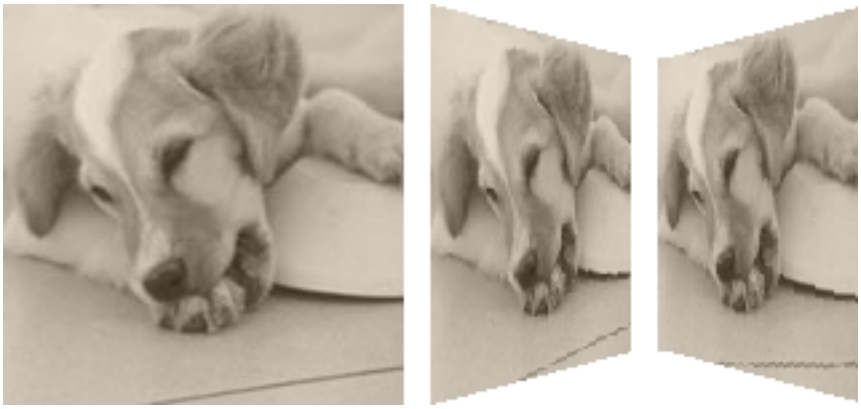
该函数将给定存储设备的内容绘制到当前选定的设备中，源设备的原点应为 (0,0)。新图像的大小和变形由参数 dx、 dy、 h0 和 h1 定义。
请注意，该函数目前仅用于色彩深度为 32 bpp 的存储设备和色彩深度为 32 bpp 的系统。

示例

以下示例显示了如何使用该函数：

```
GUI_MEMDEV_Handle hMem0, hMem1, hMem2;
hMem0 = GUI_MEMDEV_CreateFixed(0, 0, 150, 150, GUI_MEMDEV_NOTRANS,
                               GUI_MEMDEV_APILIST_32,
                               GUI_COLOR_CONV_888);
hMem1 = GUI_MEMDEV_CreateFixed(0, 0,  75, 150, GUI_MEMDEV_HASTRANS,
                               GUI_MEMDEV_APILIST_32,
                               GUI_COLOR_CONV_888);
hMem2 = GUI_MEMDEV_CreateFixed(0, 0,  75, 150, GUI_MEMDEV_HASTRANS,
                               GUI_MEMDEV_APILIST_32,
                               GUI_COLOR_CONV_888);
GUI_MEMDEV_Select(hMem0);
GUI_JPEG_Draw(_aJPEG, sizeof(_aJPEG), 0, 0);
GUI_MEMDEV_Select(hMem1);
GUI_MEMDEV_DrawPerspectiveX(hMem0, 0,  0, 150, 110, 75,  20);
GUI_MEMDEV_Select(hMem2);
GUI_MEMDEV_DrawPerspectiveX(hMem0, 0, 20, 110, 150, 75, -20);
GUI_MEMDEV_CopyToLCDAt(hMem0,   0, 10);
GUI_MEMDEV_CopyToLCDAt(hMem1, 160, 10);
GUI_MEMDEV_CopyToLCDAt(hMem2, 245, 10);
```


上述示例的屏幕截图



GUI_MEMDEV_GetDataPtr()

描述

返回存储设备的数据区域（图像区域）的指针。然后即可不使用 GUI 函数来处理此数据区域。例如，它可用作 JPEG 或视频解压缩例程的输出缓冲。

原型

```
void* GUI_MEMDEV_GetDataPtr(GUI_MEMDEV_Handle hMem);
```

参数	描述
hMem	存储设备的句柄。

其他信息

此类设备数据存储在返回地址之后。修改此类数据的应用程序必须极其小心，避免覆盖此数据区域外的内存。如果对此数据区域使用 emWin 默认内存管理，则只要在使用指针，内存区域就必须保持锁定。

数据区域的组织：

像素以 “本地” 模式存储到目标显示器 （或层）。对于 8 bpp 或更低的层，每像素使用 8 位 （1 字节）；对于大于 8 且小于等于 16 bpp 的层，每像素使用 16 位值 (U16)。内存按读取顺序组织，这意味着：第一个字节 （或 U16）存储在起始地址，代表左上角 (y=0, x=0) 的像素色彩索引；下一个像素保存紧接第一个存储，是左边一个 (y=0, x=1) 的索引。（除非存储设备区域的宽度仅为 1 像素）。下一行紧接第一行存储在内存中，无任何类型的间隔。Endian 模式是不相关的，它假定 16 位单元就被作为 16 位单元访问，而非 2 个单独的字节。数据区域由 (xSize * ySize) 个像素组成，因此 xSize * ySize 个字节用于 8bpp 或更低存储设备，
2 * xSize * ySize 个字节 （作为 16 位的 xSize * ySize 单元访问）用于 16bpp 的存储设备。

GUI_MEMDEV_GetXSize()

描述

返回存储设备的 X 尺寸 （宽度）。

原型

```
int  GUI_MEMDEV_GetXSize(GUI_MEMDEV_Handle hMem);
```

参数	描述
hMem	存储设备的句柄。

GUI_MEMDEV_GetYSize()

描述

返回存储设备的 Y 尺寸 （高度），单位像素。

原型

```
int  GUI_MEMDEV_GetYSize(GUI_MEMDEV_Handle hMem);
```

参数	描述
hMem	存储设备的句柄。

GUI_MEMDEV_MarkDirty()

描述

将某矩形区域标志为脏污。

原型

```
void GUI_MEMDEV_MarkDirty(GUI_MEMDEV_Handle hMem,
                          int x0, int y0, int x1, int y1);
```

参数	描述
hMem	存储设备的句柄。
x0	左上角的 x 坐标。
y0	左上角的 y 坐标。
x1	右下角的 x 坐标。
y1	右下角的 y 坐标。

GUI_MEMDEV_ReduceYSize()

描述

减小存储设备的 Y 尺寸。

原型

```
void GUI_MEMDEV_ReduceYSize(GUI_MEMDEV_Handle hMem, int YSize);
```

参数	描述
hMem	存储设备的句柄。
YSize	存储设备的新 Y 尺寸。

其他信息

更改存储设备的尺寸比删除然后再重新创建它效率高。

GUI_MEMDEV_Rotate(), GUI_MEMDEV_RotateHQ()

描述

这些函数旋转并缩放给定源存储设备。源设备将围绕其中心旋转和缩放，然后按给定的像素数移动。结果保存到给定的目标存储设备中。

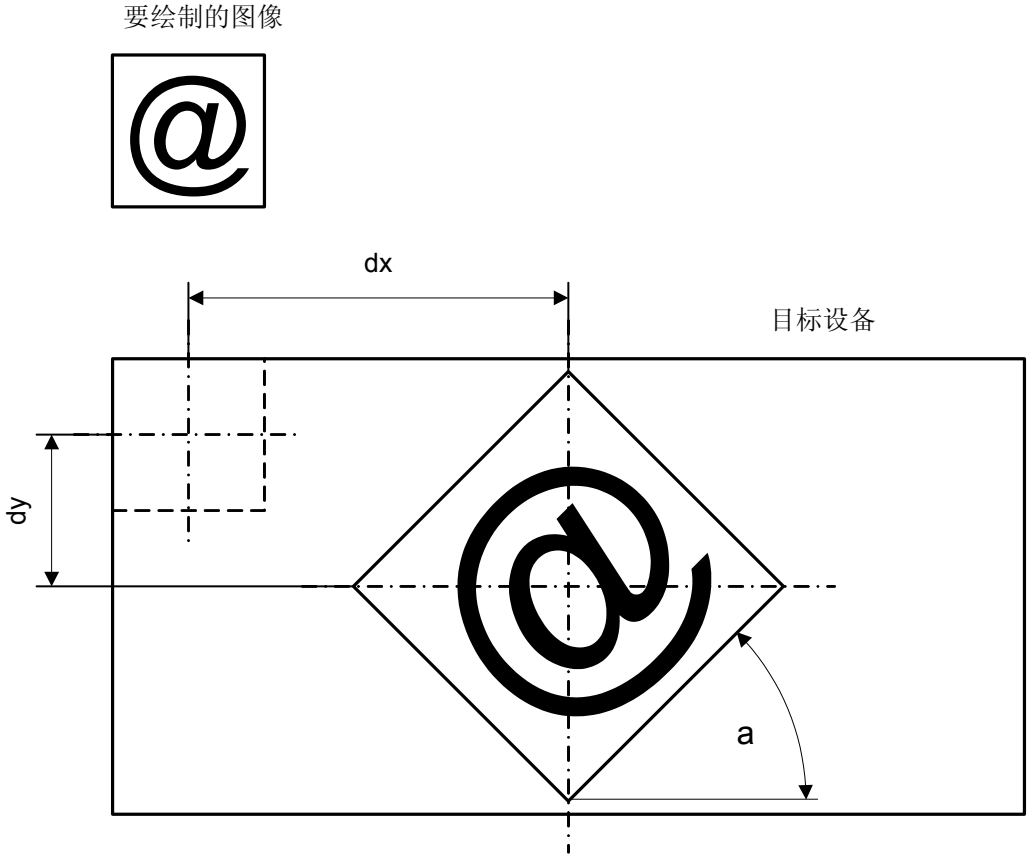
两个函数之间的不同在于计算目标像素数据的算法。GUI_MEMDEV_Rotate() 使用“最近邻居”法，较快但不够准确。GUI_MEMDEV_RotateHQ() 使用更复杂的方法，非常准确但比“最近邻居”法慢。有关此两函数间差别的更详细信息，请见本函数描述末尾的两个屏幕截图。

原型

```
void GUI_MEMDEV_RotateHQ(GUI_MEMDEV_Handle hSrc, GUI_MEMDEV_Handle hDst,
                         int dx, int dy, int a, int Mag);

void GUI_MEMDEV_Rotate(GUI_MEMDEV_Handle hSrc, GUI_MEMDEV_Handle hDst,
                       int dx, int dy, int a, int Mag);
```

参数	描述
hSrc	要旋转和缩放的存储设备的句柄。
hDst	目标设备的句柄。
dx	在 X 方向上移动图像的距离，单位像素。
dy	在 Y 方向上移动图像的距离，单位像素。
a	要旋转的角度，单位 “度 * 1000”。
Mag	放大系数 * 1000

下图为各参数的更详细直观描述：



其他信息

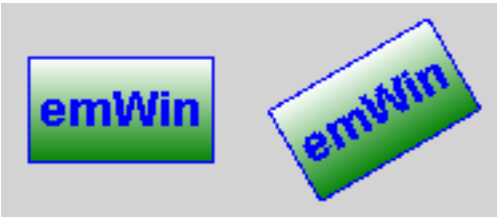
两个存储设备 （源和目标）都需要是 32 位存储设备。任何其它情况下，函数都将立即返回。Sample 文件夹中还包含了示例 MEMDEV_ZoomAndRotate.c，详细描述了如何使用该函数。

示例

```
GUI_MEMDEV_Handle hMemSource;
GUI_MEMDEV_Handle hMemDest;
GUI_RECT RectSource = {0, 0, 69, 39};
GUI_RECT RectDest   = {0, 0, 79, 79};
hMemSource = GUI_MEMDEV_CreateFixed(RectSource.x0, RectSource.y0,
                                    RectSource.x1 - RectSource.x0 + 1,
                                    RectSource.y1 - RectSource.y0 + 1,
                                    GUI_MEMDEV_NOTRANS,
                                    GUI_MEMDEV_APILIST_32, GUI_COLOR_CONV_888);
hMemDest   = GUI_MEMDEV_CreateFixed(RectDest.x0,   RectDest.y0,
                                    RectDest.x1   - RectDest.x0   + 1,
                                    RectDest.y1   - RectDest.y0   + 1,
                                    GUI_MEMDEV_NOTRANS,
                                    GUI_MEMDEV_APILIST_32, GUI_COLOR_CONV_888);
GUI_MEMDEV_Select(hMemSource);
GUI_DrawGradientV(RectSource.x0, RectSource.y0,
                  RectSource.x1, RectSource.y1,
                  GUI_WHITE, GUI_DARKGREEN);
GUI_SetColor(GUI_BLUE);
GUI_SetFont(&GUI_Font20B_ASCII);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DispStringInRect("emWin", &RectSource, GUI_TA_HCENTER | GUI_TA_VCENTER);
GUI_DrawRect(0, 0, RectSource.x1, RectSource.y1);
GUI_MEMDEV_RotateHQ(hMemSource, hMemDest,
                    (RectDest.x1 - RectSource.x1) / 2,
                    (RectDest.y1 - RectSource.y1) / 2,
                    30 * 1000,
                    1000);
GUI_MEMDEV_CopyToLCDAt(hMemSource,  10, (RectDest.y1 - RectSource.y1) / 2);
GUI_MEMDEV_CopyToLCDAt(hMemDest,   100, 0);
```

上述使用 GUI_MEMDEV_RotateHQ() 的示例的屏幕截图

上述使用 GUI_MEMDEV_Rotate() 的示例的屏幕截图



GUI_MEMDEV_Select()

描述

激活存储设备 （如果句柄为 0 则激活 LCD）。

原型

```
void GUI_MEMDEV_Select(GUI_MEMDEV_Handle hMem)
```

参数	描述
hMem	存储设备的句柄。

GUI_MEMDEV_SetOrg()

描述

改变存储设备在 LCD 上的原点。

原型

```
void GUI_MEMDEV_SetOrg(GUI_MEMDEV_Handle hMem, int x0, int y0);
```

参数	描述
hMem	存储设备的句柄。
x0	水平位置 （左上像素的）。
y0	垂直位置 （左上像素的）。

其他信息

将同一设备用于屏幕的不同区域时，或者要将存储设备的内容复制到不同区域中时，可使用此例程。更改存储设备的原点比删除然后再重新创建它效率高。

GUI_MEMDEV_Write()

描述

将给定存储设备的内容写入当前选定的存储设备中。

原型

```
void GUI_MEMDEV_Write(GUI_MEMDEV_Handle hMem);
```

参数	描述
hMem	存储设备的句柄。

GUI_MEMDEV_WriteAlpha()

描述

将给定存储设备的内容写入使用 Alpha 混合处理的当前选定存储设备中。

原型

```
void GUI_MEMDEV_WriteAlpha(GUI_MEMDEV_Handle hMem, int Alpha);
```

参数	描述
hMem	存储设备的句柄。
Alpha	Alpha 混合系数， 0 - 255

其他信息

Alpha 混合意味着用给定强度混合 2 种色彩。利用此函数，可以从一个存储设备将半透明写入其它存储设备。 Alpha 参数指定写入当前选定存储设备时使用的强度。

GUI_MEMDEV_WriteAlphaAt()

描述

将给定存储设备的内容写入使用 Alpha 混合处理的当前选定存储设备的指定位置。

原型

```
void GUI_MEMDEV_WriteAlphaAt(GUI_MEMDEV_Handle hMem,
                             int Alpha, int x, int y);
```

参数	描述
hMem	存储设备的句柄。
Alpha	Alpha 混合系数， 0 - 255
x	X 方向位置。
y	Y 方向位置。

其他信息

（参见 GUI_MEMDEV_WriteAlpha）

GUI_MEMDEV_WriteAt()

描述

将给定存储设备的内容写入当前选定存储设备的指定位置中。

原型

```
void GUI_MEMDEV_WriteAt(GUI_MEMDEV_Handle hMem, int x, int y);
```

参数	描述
hMem	存储设备的句柄。
x	X 方向位置。
y	Y 方向位置。

GUI_MEMDEV_WriteEx()

描述

将给定存储设备的内容写入使用 Alpha 混合处理和缩放的当前选定存储设备的位置 (0,0)。

原型

```
void GUI_MEMDEV_WriteEx(GUI_MEMDEV_Handle hMem,
                        int xMag, int yMag, int Alpha);
```

参数	描述
hMem	存储设备的句柄。
xMag	X 轴的缩放系数 * 1000。
yMag	Y 轴的缩放系数 * 1000。
Alpha	Alpha 混合系数， 0 – 255。

其他信息

负的缩放系数将镜像输出。请参阅 “GUI_MEMDEV_WriteExAt()”（第 264 页）。

GUI_MEMDEV_WriteExAt()

描述

将给定存储设备的内容写入使用 Alpha 混合处理和缩放的当前选定存储设备的指定位置。

原型

```
void GUI_MEMDEV_WriteExAt(GUI_MEMDEV_Handle hMem,
                          int x, int y, int xMag, int yMag, int Alpha);
```

参数	描述
hMem	存储设备的句柄。
x	X 方向位置。
y	Y 方向位置。
xMag	X 轴的缩放系数 * 1000。
yMag	Y 轴的缩放系数 * 1000。
Alpha	Alpha 混合系数， 0 – 255。

其他信息

负的缩放系数将镜像输出。

示例

下列创建了 2 个存储设备：hMem0 (40x10) 和 hMem1 (80x20)。在 hMem0 和 hMem1 的左上位置绘制了一小段白色文本，然后函数 GUI_MEMDEV_WriteEx() 使用镜像和放大将 hMem0 的内容写入 hMem1。

```
GUI_MEMDEV_Handle hMem0, hMem1;
GUI_Init();
hMem0 = GUI_MEMDEV_Create(0, 0, 40, 10);
hMem1 = GUI_MEMDEV_Create(0, 0, 80, 20);
GUI_MEMDEV_Select(hMem0);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DispString("Text");
GUI_MEMDEV_Select(hMem1);
GUI_SetBkColor(GUI_RED);
GUI_Clear();
GUI_DispStringAt("Text", 0, 0);
GUI_MEMDEV_WriteExAt(hMem0, 0, 0, -2000, -2000, 160);
GUI_MEMDEV_CopyToLCD(hMem1);
```

上述示例的屏幕截图

GUI_SelectLCD()

描述

选择作为绘制操作目标的 LCD。

原型

```
void GUI_SelectLCD(void))
```

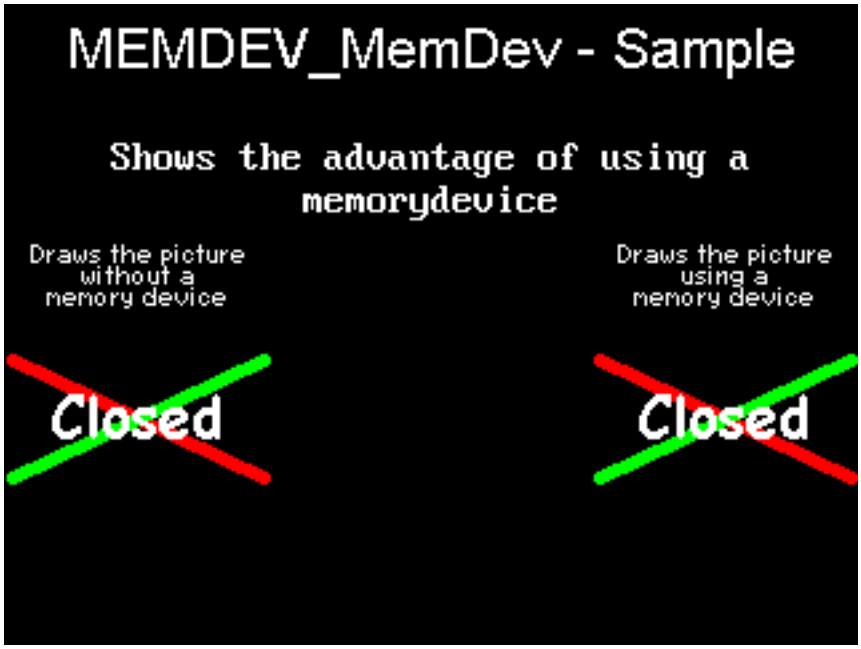
使用存储设备的示例

Sample 文件夹包含显示如何使用存储设备的如下示例：

- MEMDEV_MemDev.c

此示例说明存储设备的使用。将一些项目写入存储设备中然后复制到显示器中。请注意，有几个其他示例也使用了存储设备，也有助于熟悉存储设备。

上述示例的屏幕截图：



12.13 分段存储设备

首先通过执行指定绘制函数来填充存储设备，填充后，内容即被绘制到 LCD 中。可能没有用于一次性存储整个输出区域的内存可用，这取决于与配置 （参见 “配置”（第 905 页）章节 “配置”中的 GUI_ALLOC_SIZE 配置宏）。分段存储设备将绘制区域分成多段，每段包含当前可用内存可包含的最多行。

GUI_MEMDEV_Draw()

描述

基本绘制函数，防止显示器闪烁。

原型

```
int GUI_MEMDEV_Draw(GUI_RECT* pRect,
                    GUI_CALLBACK_VOID_P* pfDraw,
                    void* pData,
                    int NumLines,
                     int Flags)
```

参数	描述
pRect	所使用 LCD 区域的 GUI_RECT 结构的指针。
pfDraw	执行绘制的回调函数的指针。
pData	用作回调函数参数的数据结构的指针。
NumLines	0 （推荐）或存储设备的行数。
Flags	（见下表）。

参数 Flags 的允许值	
GUI_MEMDEV_HASTRANS (recommended)	默认：使用透明性标记创建存储设备，该标记确保正确绘制背景。
GUI_MEMDEV_NOTRANS	创建存储设备，无透明性。用户必须确保正确绘制背景。仅应用于优化目的。

返回值

成功时为 0，例程失败则为 1。

其他信息

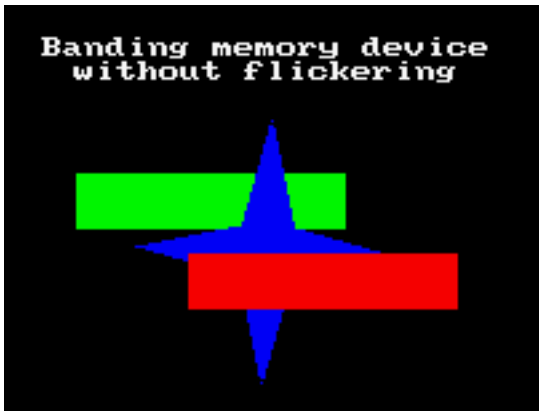
如果参数 NumLines 为 0，则每段的行数由该函数自动计算。然后函数通过移动存储设备的原点，逐段迭代输出区域。

使用分段存储设备的示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- MEMDEV_Banding.c

上述示例的屏幕截图



12.14 自动设备对象

必须更新显示器以反映项目的移动或更改时可使用存储设备，因为在此类应用中，防止 LCD 闪烁非常重要。自动设备对象以分段存储设备为基础，对于一次仅更新显示器一小部分的应用（如移动指示器）而言可能更高效。

该设备会自动识别包含固定对象的显示器区域，以及包含必须更新的移动或更改对象的区域。第一次调用绘制函数时，会绘制所有项目，每一次后续调用都将仅更新移动或更改对象使用的空间。实际绘制操作使用分段存储设备，但仅限在必要空间内。使用自动设备对象的主要优势（相对于使用分段存储设备而言）是计算时间减少，因为它不始终更新整个显示器。

GUI_MEMDEV_CreateAuto()

描述

创建自动设备对象。

原型

```
int GUI_MEMDEV_CreateAuto(GUI_AUTODEV * pAutoDev);
```

参数	描述
pAutoDev	GUI_AUTODEV 对象的指针。

返回值

目前 0，留待以后使用。

GUI_MEMDEV_DeleteAuto()

描述

删除自动设备对象。

原型

```
void GUI_MEMDEV_DeleteAuto(GUI_AUTODEV * pAutoDev);
```

参数	描述
pAutoDev	GUI_AUTODEV 对象的指针。

GUI_MEMDEV_DrawAuto()

描述

使用分段存储设备执行指定绘制例程。

原型

```
int GUI_MEMDEV_DrawAuto(GUI_AUTODEV          * pAutoDev,
                        GUI_AUTODEV_INFO     * pAutoDevInfo,
                        GUI_CALLBACK_VOID_P * pfDraw,
                        void                 * pData);
```

参数	描述
pAutoDev	GUI_AUTODEV 对象的指针。
pAutoDevInfo	GUI_AUTODEV_INFO 对象的指针。
pfDraw	要执行的用户定义绘制函数的指针。
pData	传递给绘制函数的数据结构的指针。

返回值

成功时为 0，例程失败则为 1。

其他信息

GUI_AUTODEV_INFO 结构包含用户函数必须绘制哪些项目的信息：

```
typedef struct {
  char DrawFixed;
} GUI_AUTODEV_INFO;
```

如果必须绘制所有项目，则 DrawFixed 设置为 1。仅绘制移动或更改对象时，设置为 0。使用此功能时，建议采用以下步骤：

```
typedef struct {
  GUI_AUTODEV_INFO AutoDevInfo; /* Information about what has to be drawn */
  /* Additional data used by the user function */
  ...
} PARAM;

static void Draw(void * p) {
  PARAM * pParam = (PARAM *)p;
  if (pParam->AutoDevInfo.DrawFixed) {
    /* Draw fixed background */
    ...
  }
  /* Draw moving objects */
  ...
  if (pParam->AutoDevInfo.DrawFixed) {
    /* Draw fixed foreground (if needed) */
    ...
  }
}

void main(void) {
  PARAM Param;                    /* Parameters for drawing routine */
```

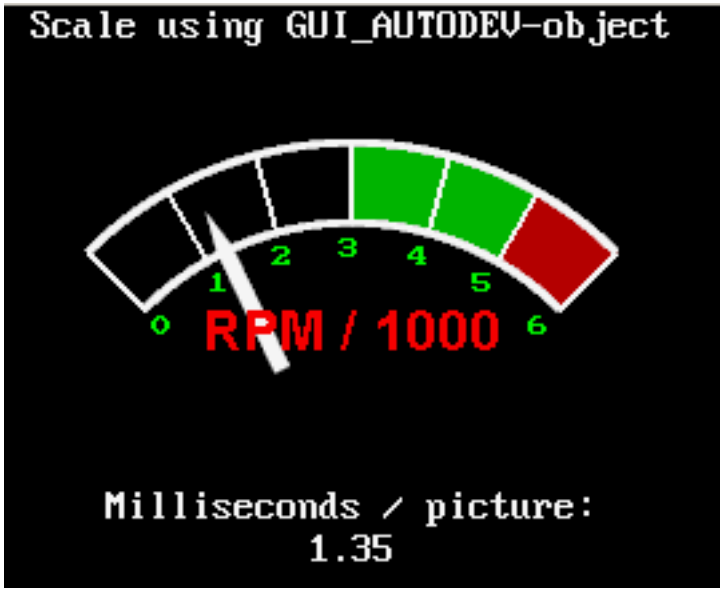


```
  GUI_AUTODEV AutoDev;             /* Object for banding memory device */
  /* Set/modify informations for drawing routine */
  ...
  GUI_MEMDEV_CreateAuto(&AutoDev); /* Create GUI_AUTODEV-object */
  GUI_MEMDEV_DrawAuto(&AutoDev,    /* Use GUI_AUTODEV-object for drawing */
                      &Param.AutoDevInfo,
                      &Draw,
                      &Param);
  GUI_MEMDEV_DeleteAuto(&AutoDev); /* Delete GUI_AUTODEV-object */
}
```

使用自动设备对象的示例

示例 MEMDEV_AutoDev.c 说明了如何使用自动设备对象。其相应文件为 MEMDEV_AutoDev.c，在背景中绘制一个带移动针的刻度，在前景中写入一小段文本。针使用 emWin 的抗锯齿功能绘制。为了改善移动针的外观，在此使用了高分辨率的抗锯齿功能。更多详细信息，请参阅 “抗锯齿”（第 787 页）。

上述示例的屏幕截图



12.15 测量设备对象

需要知道绘制区域面积时，可使用测量设备。创建并选择测量设备，作为绘制操作的目标，就可以检索用于绘制操作的矩形。

GUI_MEASDEV_Create()

描述

创建测量设备。

原型

```
GUI_MEASDEV_Handle GUI_MEASDEV_Create(void);
```

返回值

测量设备的句柄。

GUI_MEASDEV_ClearRect()

描述

调用此函数可清除给定测量设备的测量矩形。

原型

```
void GUI_MEASDEV_ClearRect(GUI_MEASDEV_Handle hMem);
```

参数	描述
hMem	测量设备的句柄。

GUI_MEASDEV_Delete()

描述

删除测量设备。

原型

```
void GUI_MEASDEV_Delete (GUI_MEASDEV_Handle hMem);
```

参数	描述
hMem	测量设备的句柄。

GUI_MEASDEV_GetRect()

描述

检索绘制操作的结果。

原型

```
void GUI_MEASDEV_GetRect(GUI_MEASDEV_Handle hMem, GUI_RECT *pRect);
```

参数	描述
hMem	测量设备的句柄。
pRect	用于保存结果的 GUI_RECT 结构的指针。

GUI_MEASDEV_Select()

描述

选择测量设备，作为绘制操作的目标。

原型

```
void GUI_MEASDEV_Select (GUI_MEASDEV_Handle hMem);
```

参数	描述
hMem	测量设备的句柄。

示例

下例说明了如何使用测量设备，它创建了一个测量设备、绘制一条线然后显示测量设备的结果：

```
void MainTask(void) {
  GUI_MEASDEV_Handle hMeasdev;
  GUI_RECT Rect;
  GUI_Init();
  hMeasdev = GUI_MEASDEV_Create();
  GUI_MEASDEV_Select(hMeasdev);
  GUI_DrawLine(10, 20, 30, 40);
  GUI_SelectLCD();
  GUI_MEASDEV_GetRect(hMeasdev, &Rect);
  GUI_MEASDEV_Delete(hMeasdev);
```



```
  GUI_DispString("X0:");
  GUI_DispDec(Rect.x0, 3);
  GUI_DispString(" Y0:");
  GUI_DispDec(Rect.y0, 3);
  GUI_DispString(" X1:");
  GUI_DispDec(Rect.x1, 3);
  GUI_DispString(" Y1:");
  GUI_DispDec(Rect.y1, 3);
}
```

上述示例的屏幕截图：

```
X0:010 Y0:020 X1:030 Y1:040
```

12.16 动画函数

可使用动画将一些实际情况映射到应用程序中，这些动画始终有助于使用户顺利观察到应用程序的进展。

GUI_MEMDEV_FadeDevices()

描述

执行从一个存储设备到另一存储设备的淡变。

原型

```
int GUI_MEMDEV_FadeDevices(GUI_MEMDEV_Handle hMem0,
                           GUI_MEMDEV_Handle hMem1,
                           int               Period);
```

参数	描述
hMem0	必须淡出的存储设备的句柄。
hMem1	必须淡入的存储设备的句柄。
Period	处理淡变的时间期间。

返回值

成功时为 0，函数失败则为 1。

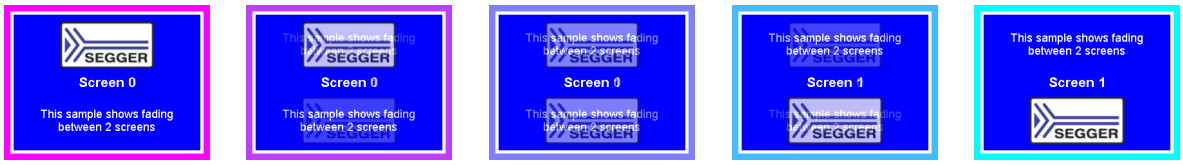
其他信息

请注意，此函数仅在 hMem0 和 hMem1 大小相同且位于屏幕上同一位置时进行处理。

示例

有关使用淡变函数的示例，请参阅“emWin\Sample\Tutorial”中的“MEMDEV_FadingPerformance.c”。

屏幕截图



GUI_MEMDEV_SetAnimationCallback()

描述

设置在处理动画时要调用的用户定义回调函数。该函数应包含确定是继续还是中止当前动画进程的代码。

原型

```
void GUI_MEMDEV_SetAnimationCallback(
                                  GUI_ANIMATION_CALLBACK_FUNC * pCbAnimation,
                                  void                        * pVoid);
```

参数	描述
pCbAnimation	指向用户定义回调函数的指针。
pVoid	数据指针。

其他信息

该函数在动画函数每次刚复制实际步骤到屏幕时调用。

示例

以下为使用 GUI_ANIMATION_CALLBACK_FUNC 的示例，它能够对 PID 事件作出反应：

```
static int _cbAnimation(int TimeRem, void * pVoid) {
  int           Pressed;

  if (TimeRem /* Insert Condition */) {
    /* ... React on remaining Time ...*/
  }
  Pressed = _GetButtonState();
  if (Pressed) {
    return 1;   // Button was     pressed, stop     animation
  } else {
    return 0;   // Button was not pressed, continue animation
  }
}

void main(void) {
  GUI_Init();
  GUI_MEMDEV_SetAnimationCallback(_cbAnimation, (void *)&_Pressed);
  while (1) {
    /* Do animations...*/
  }
}
```

12.17 动画函数 （需要使用窗口管理器）

以下动画函数要求使用窗口管理器。

GUI_MEMDEV_FadeInWindow()

GUI_MEMDEV_FadeOutWindow()

描述

通过减小 / 增加 Alpha 值淡入 / 淡出窗口。

原型

```
int GUI_MEMDEV_FadeInWindow (WM_HWIN hWin, int Period);
int GUI_MEMDEV_FadeOutWindow(WM_HWIN hWin, int Period);
```

参数	描述
hWin	必须淡入 / 淡出的窗口的句柄
Period	处理淡变的时间期间

返回值

成功时为 0，函数失败则为 1。

其他信息

请注意，调用此函数后，当前桌面及其子窗口的状态为 “有效”。

示例

有关使用淡变函数的示例，请参阅 “emWin\Sample” 文件夹中的 “SKINNING_NestedModal.c”。

屏幕截图

GUI_MEMDEV_MoveInWindow()

GUI_MEMDEV_MoveOutWindow()

描述

将窗口移入 / 移出屏幕。首先在指定位置 / 实际位置绘制最小化 / 最大化的窗口，然后移动到其实际位置 / 指定位置，同时放大到其实际尺寸 / 缩小。窗口在移动时可顺时针和逆时针旋转。

原型

```
int GUI_MEMDEV_MoveInWindow (WM_HWIN hWin, int x,     int y,
                                           int a180, int Period);
int GUI_MEMDEV_MoveOutWindow(WM_HWIN hWin, int x,     int y,
                                           int a180, int Period);
```

参数	描述
hWin	必须移动的窗口的句柄
x	离被移动窗口的 x 距离
y	离被移动窗口的 y 距离
a180	窗口将旋转的度数： a180 = 0 -> 无旋转 a180 > 0 -> 顺时针 a180 < 0 -> 逆时针
Period	处理移动的时间期间

返回值

成功时为 0，函数失败则为 1。

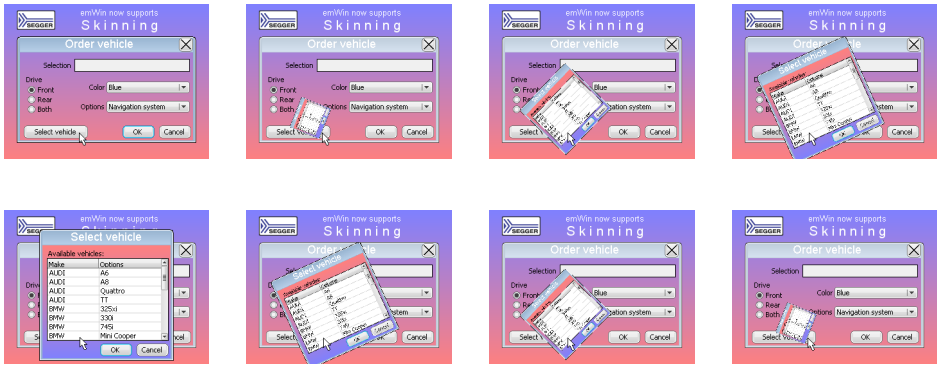
其他信息

请注意，调用此函数后，当前桌面及其子窗口的状态为 “有效”。 GUI_MEMDEV_MoveInWindow()/ GUI_MEMDEV_MoveOutWindow() 要求约 1 MB 的动态内存来以 QVGA 模式正确运行。

示例

有关使用 GUI_MEMDEV_MoveInWindow() 和 GUI_MEMDEV_MoveOutWindow() 的示例，请参阅文件夹 “emWin\Sample” 中的 “SKINNING_NestedModal.c”。

屏幕截图



GUI_MEMDEV_ShiftInWindow()

GUI_MEMDEV_ShiftOutWindow()

描述

在指定方向移动窗口，从其实际位置移出屏幕，或移入其在屏幕上的实际位置。

原型

```
int GUI_MEMDEV_ShiftInWindow (WM_HWIN hWin, int Period, int Direction);
int GUI_MEMDEV_ShiftOutWindow(WM_HWIN hWin, int Period, int Direction);
```

参数	描述
hWin	必须移动的窗口的句柄
Period	处理移动的时间期间
Direction	窗口的移动方向。

返回值

成功时为 0，函数失败则为 1。

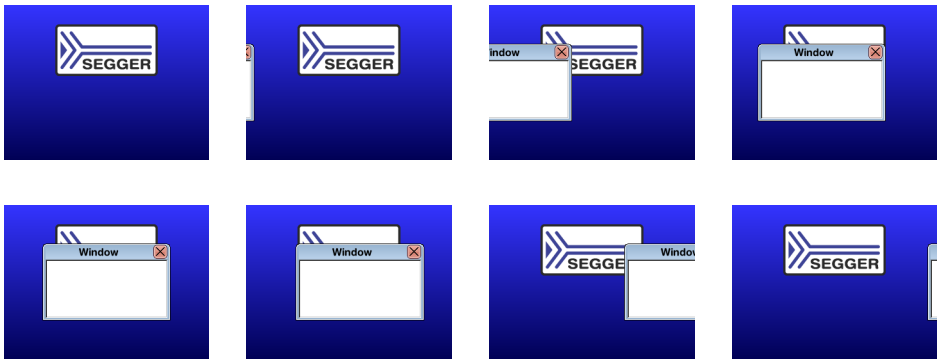
其他信息

请注意，移动窗口后，当前桌面及其子窗口的状态为 “有效”。 GUI_MEMDEV_ShiftInWindow() 和 GUI_MEMDEV_ShiftOutWindow() 要求约 1 MB 的动态内存来以 QVGA 模式正确运行。

示例

有关使用 GUI_MEMDEV_ShiftInWindow() 和 GUI_MEMDEV_ShiftOutWindow() 的示例，请参阅文件夹 “emWin\Sample” 中的 “SKINNING_Notepad.c”。

屏幕截图



GUI_MEMDEV_ShiftWindow()

描述

在指定方向移动窗口，至屏幕中其实际位置处，并移出相应窗口区域中的旧内容。

原型

```
int GUI_MEMDEV_ShiftWindow(WM_HWIN hWin, int Period, int Edge);
```

参数	描述
hWin	必须移动的窗口的句柄
Period	处理移动的时间期间
Edge	窗口的移动方向。

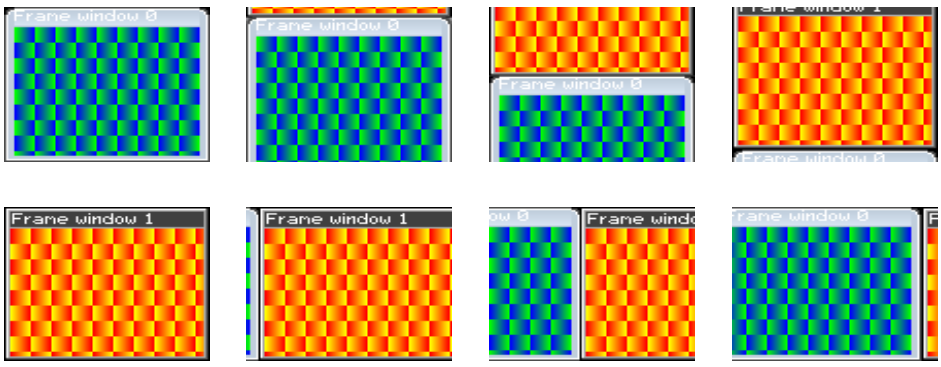
返回值

成功时为 0，函数失败则为 1。

其他信息

请注意，交换窗口后，当前桌面及其子窗口的状态为 “有效”。GUI_MEMDEV_ShiftWindow() 要求约 1 MB 的动态内存来以 QVGA 模式正确运行。

屏幕截图



第 13 章

执行模型：单任务 / 多任务

emWin 最初就设计为与不同类型的环境兼容。它在单任务和多任务应用中作业，使用专用操作系统或任何商用 “实时操作系统 (RTOS)”，如 embOS 或 uC/OS。

13.1 支持的执行模型

我们必须从根本上区分 3 种不同的执行模型：

单任务系统 （超循环）

整个程序在一个超循环中运行。通常，所有软件组件都周期性调用。软件的实时部分必须使用中断，因为未使用实时内核。

多任务系统：一个任务调用 emWinemWin

使用实时内核 (RTOS)，但只有一个任务调用 emWin 函数。从图形软件的视点看，它与在单任务系统中的使用是相同的。

多任务系统：多个任务调用 emWin

使用实时内核 (RTOS)，并且多个任务调用 emWin 函数。只要软件是线程安全的，则调用操作不会发生任何问题，这通过在配置中启用多任务支持并改编内核接口例程实现。对于普通内核，内核接口例程是即用的。

13.2 单任务系统 （超循环）

13.2.1 描述

整个程序在一个超循环中运行。通常，软件的所有组件都进行周期性调用。因为未使用实时内核，所以软件的实时部分必须使用中断。此类型系统主要用于小型系统，或者实时操作特性无关紧要时。

13.2.2 超循环示例 （无 emWin）

```
void main (void) {
  HARDWARE_Init();

  /* Init software components */
  XXX_Init();
  YYY_Init();

  /* Superloop:call all software components regularily */
  while (1) {
    /* Exec all compontents of the software */
    XXX_Exec();
    YYY_Exec();
  }
}
```

13.2.3 优势

因为未使用实时内核 （-> 较小的 ROM 大小，仅一个堆栈 -> 用于堆栈的 RAM 较少），所以不存在抢先 / 同步问题。

13.2.4 缺点

超循环类型程序的大小如果超出某个值，会变得很难维护。实时特性较差，因为一个软件组件无法被任何其它组件中断 （只能通过中断）。这意味着一个软件组件的反应时间取决于系统中所有其它组件的执行时间。

13.2.5 使用 emWin

关于 emWin 的使用没有实际的限定。照例，需要在调用 GUI_Init() 之后才能使用该软件。之后，任何 API 函数都可使用。如果使用了窗口管理器的回调机制，则必须定期调用 emWin 更新函数。这通常通过从超循环内调用GUI_Exec() 来完成。各种模块化函数，如GUI_Delay() 和GUI_ExecDialog() 不应在循环中使用，因为它们会阻断其它软件模块。
可使用不支持多任务 (#define GUI_OS 0) 的默认配置；不需要内核接口例程。

13.2.6 超循环示例 （有 emWin）

```
void main (void) {
  HARDWARE_Init();

  /* Init software components */
  XXX_Init();
  YYY_Init();
  GUI_Init();         /* Init emWin */

  /* Superloop:call all software components regularily */
  while (1) {
    /* Exec all compontents of the software */
    XXX_Exec();
    YYY_Exec();
    GUI_Exec();         /* Exec emWin for functionality like updating windows */
  }
}
```

13.3 多任务系统：一个任务调用 emWin

13.3.1 描述

使用了实时内核 (RTOS)。用户程序分成不同部分，在不同的任务中执行，这些任务通常具有不同的优先级。通常，实时关键任务 （需要一定的反应时间）将具有最高优先级。**一个单任务**用于用户界面，它调用 emWin 函数。此任务在系统中通常具有最低优先级，或至少是最低优先级任务中的一个（一些统计任务或简单的空闲时间处理可能具有更低的优先级）。
中断可以但不是必须用于软件的实时部分。

13.3.2 优势

该系统的实时特性非常优秀。任务的实时特性仅受以较高优先级运行的任务的影响。这意味着对在低优先级任务中运行的程序组件的更改完全不会影响实时特性。如果从低优先级任务执行用户界面，则意味着用户界面的更改不影响实时特性。因此，使用这种系统可轻松将软件的不同组件分配给开发团队的不同成员，成员彼此间可以高度独立地工作。

13.3.3 缺点

用户需要拥有实时内核 (RTOS)，这需要资金并会耗费 ROM 和 RAM （用于堆栈）。此外，还必须考虑任务的同步，以及如何将信息从一个任务传输到另一个任务。

13.3.4 使用 emWin

如果使用了窗口管理器的回调机制，则必须从调用 emWin 的任务定期调用 emWin 更新函数（通常为 `GUI_Exec()`、`GUI_Delay()`）。由于 emWin 仅由一个任务调用，所以对于 emWin 而言，它与在单任务系统中的使用是相同的。
可使用不支持多任务 (`#define GUI_OS 0`) 的默认配置；不需要内核接口例程。可使用任何实时内核，商用的或专用的。

13.4 多任务系统：多个任务调用 emWin

13.4.1 描述

使用了实时内核 (RTOS)。用户程序分成不同部分，在不同任务中执行，这些任务通常具有不同的优先级。通常，实时关键任务 （需要一定的反应时间）将具有最高优先级。**多个任务**用于用户界面，调用 emWin 函数。这些任务在系统中通常具有低优先级，所以它们不影响系统的实时特性。
中断可以但不是必须用于软件的实时部分。

13.4.2 优势

该系统的实时特性非常优秀。任务的实时特性仅受以较高优先级运行的任务的影响。这意味着对在低优先级任务中运行的程序组件的更改完全不会影响实时特性。如果从低优先级任务执行用户界面，则意味着用户界面的更改不影响实时特性。因此，使用这种系统可轻松将软件的不同组件分配给开发团队的不同成员，成员彼此间可以高度独立地工作。

13.4.3 缺点

用户必须拥有实时内核 (RTOS)，这需要资金并会耗费部分 ROM 和 RAM （用于堆栈）。此外，还必须考虑任务的同步，以及如何将信息从一个任务传输到另一个任务。

13.4.4 使用 emWin

如果使用了窗口管理器的回调机制，则必须从调用 emWin 的一个或多个任务定期调用 emWin 更新函数 （通常为 GUI_Exec()、 GUI_Delay()）。
不可使用不支持多任务 (#define GUI_OS 0) 的默认配置。配置需要启用多任务支持并定义从中调用 emWin 的任务的最大数目 （摘自 GUIConf.h）：

```
#define GUI_OS 1            // Enable multitasking support
#define GUI_MAX_TASK 5      // Max. number of tasks that may call emWin
```

内核接口例程是必需的，并需要与正在使用的内核匹配。可使用任何实时内核，商用的或专用的。宏和例程将下面章节中讨论。

13.4.5 建议

- 仅从一个任务调用 emWin 更新函数（即 GUI_Exec()、GUI_Delay()），这有助于保持程序结构清晰。如果在您的系统中有足够的 RAM，则指定一个任务 （具有最低优先级）来更新 emWin。如下面示例所示，此任务将不断调用 GUI_Exec()，且不进行任何其它操作。
- 保持实时任务（它们确定 I/O、接口、网络等方面的系统特性）与调用 emWin 的任务分开，这有助于确保最佳的实时性能。
- 如果可能，仅为用户界面使用一个任务，这有助于保持程序结构简单并简化调试工作。（当然，这不是必须的，而且可能在一些系统中不适合。）

13.4.6 示例

此摘录显示了专用的 emWin 更新任务。它摘自示例 MT_Multitasking，该示例包括在随 emWin 发货的示例中：


```
/*********************************************************************
*
*          GUI background processing
*
* This task does the background processing.
* The main job is to update invalid windows, but other things such as
* evaluating mouse or touch input may also be done.
*/
void GUI_Task(void) {
  while(1) {
    GUI_Exec();           /* Do the background work ...Update windows etc.) */
    GUI_X_ExecIdle();     /* Nothing left to do for the moment ...Idle processing */
  }
}
```

13.5 支持多任务的 GUI 配置函数

下表所示为有多项任务调用 emWin 的多任务系统的可用配置函数：

例程	描述
GUI_SetSignalEventFunc()	设置发送事件信号的函数。
GUI_SetWaitEventFunc()	设置等待事件的函数。
GUI_SetWaitEventTimedFunc()	设置在给定时间段内等待事件的函数。

GUI_SetSignalEventFunc()

描述

设置发送事件信号的函数。

原型

```
void GUI_SetSignalEventFunc(GUI_SIGNAL_EVENT_FUNC pfSignalEvent);
```

参数	描述
pfSignalEvent	发送事件信号函数的指针。

GUI_SIGNAL_EVENT_FUNC 的定义

```
typedef void (* GUI_SIGNAL_EVENT_FUNC)(void);
```

其他信息

默认情况下，GUI 需要定期检查事件，除非定义了一个等待函数且是一个会触发事件的函数。此函数设置触发事件的函数。它仅在与 GUI_SetWaitEventFunc() 和 GUI_SetWaitEventTimedFunc() 结合使用才有意义。使用这些函数替代轮询的优点是，在等待输入时可将等待任务的 CPU 负载减少到 0%。如果已按建议指定函数，且用户给系统提供了输入（键盘或指针输入设备），则指定的函数将发送事件信号。
建议指定函数 GUI_X_SignalEvent() 用于该作业。

示例

```
GUI_SetSignalEventFunc(GUI_X_SignalEvent);
```

GUI_SetWaitEventFunc()

描述

设置等待事件的函数。

原型

```
void GUI_SetWaitEventFunc(GUI_WAIT_EVENT_FUNC pfWaitEvent);
```

参数	描述
pfWaitEvent	等待事件的函数的指针。

GUI_SIGNAL_EVENT_FUNC 的定义

```
typedef void (* GUI_WAIT_EVENT_FUNC)(void);
```

其他信息

默认情况下，GUI 需要定期检查事件，除非定义了一个等待函数且是一个会触发事件的函数。此函数设置等待事件的函数。此函数只有与 GUI_SetSignalEventFunc() 和 GUI_SetWaitEventTimedFunc() 结合使用才有意义。使用这些函数替代轮询的优点是，在等待输入时可将等待任务的CPU负载减少到0%。如果已按建议指定函数，且系统等待用户输入，则定义的函数应等待从 GUI_SetSignalEventFunc() 指定的函数发出信号的事件。
建议指定函数 GUI_X_WaitEvent() 用于该作业。

示例

```
GUI_SetWaitEventFunc(GUI_X_WaitEvent);
```

GUI_SetWaitEventTimedFunc()

描述

定义在专用时间段等待事件的函数。

原型

```
void GUI_SetWaitEventTimedFunc(GUI_WAIT_EVENT_TIMED_FUNC pfWaitEventTimed);
```

参数	描述
pfWaitEventTimed	等待事件的函数的指针。

GUI_WAIT_EVENT_TIMED_FUNC 的定义

```
typedef void (* GUI_WAIT_EVENT_TIMED_FUNC)(int Period);
```

参数	描述
Period	等待事件的时段，单位为 ms。

其他信息

默认情况下，GUI 需要定期检查事件，除非定义了一个等待函数且是一个会触发事件的函数。如果定时器处于激活状态，则此函数设置等待事件的函数。此函数只有与 GUI_SetSignalEventFunc() 和 GUI_SetWaitEventFunc() 结合使用才有意义。如果已按建议指定函数，且系统在定时器激活期间等待用户输入，则定义的函数应等待直到定时器过期，或从 GUI_SetSignalEventFunc() 设置的函数发出事件信号。
建议指定函数 GUI_X_WaitEventTimed() 用于该作业。

示例

```
GUI_SetWaitEventTimedFunc(GUI_X_WaitEventTimed);
```

13.6 支持多任务的 GUI 配置宏

下表所示为用于有多项任务调用 emWin 的多任务系统的配置宏：

类型	宏	默认值	描述
N	GUI_MAXTASK	4	定义在多任务支持启用时，从其中调用 emWin 的任务的最大数目。
B	GUI_OS	0	激活以启用多任务支持。
F	GUI_X_SIGNAL_EVENT	-	定义发送事件信号的函数。（弃用）
F	GUI_X_WAIT_EVENT	GUI_X_ExecIdle	定义等待事件的函数。（弃用）
F	GUI_X_WAIT_EVENT_TIMED	-	定义在专用时间段等待事件的函数。（弃用）

GUI_MAXTASK

描述

定义从其中调用 emWin 以访问显示器的任务的最大数目。

类型

数字值。

其他信息

仅当激活 GUI_OS 时此符号才有意义。如果使用预编译库，则应使用函数 GUITASK_SetMaxTask()。更多详细信息，请参阅 “GUITASK_SetMaxTask()”（第 908 页）。

GUI_OS

描述

通过激活模块 GUITask 启用多任务支持。

类型

二进制开关
0：未激活，多任务支持禁用 （默认）
1：激活，多任务支持启用

GUI_X_SIGNAL_EVENT

描述

定义发送事件信号的函数。

类型

函数替换

其他信息

默认情况下， GUI 需要定期检查事件，除非定义了一个等待函数且是一个会触发事件的函数。此宏定义触发事件的函数，它只有与 GUI_X_WAIT_EVENT 结合使用才有意义。使用宏 GUI_X_SIGNAL_EVENT 和 GUI_X_WAIT_EVENT 替代轮询的优点是，在其等待输入时可将等待任务的 CPU 负载减少到 0%。如果已按建议定义宏，且用户给系统提供了输入（键盘或指针输入设备），则定义的函数将发送事件信号。建议指定函数 GUI_X_SignalEvent() 用于该作业。

示例

```
#define GUI_X_SIGNAL_EVENT GUI_X_SignalEvent
```

GUI_X_WAIT_EVENT

描述

定义等待事件的函数。

类型

函数替换

其他信息

默认情况下， GUI 需要定期检查事件，除非定义了一个等待函数且是一个会触发事件的函数。此宏定义等待事件的函数，只有与 GUI_X_SIGNAL_EVENT 结合使用才有意义。使用宏 GUI_X_SIGNAL_EVENT 和 GUI_X_WAIT_EVENT 替代轮询的优点是，在其等待输入时可将等待任务的 CPU 负载减少到 0%。如果已按建议定义宏，且系统等待用户输入，则定义的函数应等待宏 GUI_X_SIGNAL_EVENT 定义的函数发出信号的事件。
建议指定函数 GUI_X_WaitEvent() 用于该作业。

示例

```
#define GUI_X_WAIT_EVENT GUI_X_WaitEvent
```

GUI_X_WAIT_EVENT_TIMED

描述

定义在专用时间段等待事件的函数。

类型

函数替换

其他信息

默认情况下， GUI 需要定期检查事件，除非定义了一个等待函数且是一个会触发事件的函数。如果定时器处于激活状态，则此宏定义等待事件的函数。只有与 GUI_X_SIGNAL_EVENT 结合使用才有意义。如果已按建议定义宏，且系统在定时器激活期间等待用户输入，则定义的函数应等待直到定时器过期，或从宏 GUI_X_SIGNAL_EVENT 定义的函数发出事件信号。
建议指定函数 GUI_X_WaitEventTimed() 用于该作业。

示例

```
#define GUI_X_WAIT_EVENT_TIMED GUI_X_WaitEventTimed
```

13.7 内核接口 API

RTOS 通常提供一种称为资源信号量的机制，在这种机制中，使用特定资源的任务在实际使用该资源之前要对其进行声明。显示器即是需要使用资源信号量保护的资源的一个示例。 emWin 在访问显示器之前或在使用关键内部数据结构之前，使用宏 GUI_USE 调用函数 GUI_Use()。类似地，它在访问显示器或使用数据结构之后调用 GUI_Unuse()，该操作在模块 GUITask.c 中完成。

GUITask.c 则使用下表所示的 GUI 内核接口例程。这些例程前缀以 GUI_X_，因为它们是高级（硬件相关的）函数。必须根据所用的实时内核对它们进行改编，以确保 emWin 任务 （或线程）安全。下面是例程的详细描述以及如何针对不同内核进行改编的示例。

例程	描述
GUI_X_GetTaskId()	返回当前任务 / 线程的唯一的 32 位标识符。
GUI_X_InitOS()	初始化内核接口模块 （创建资源信号量 / 互斥体）。
GUI_X_Lock()	锁定 GUI （阻止资源信号量 / 互斥体）。
GUI_X_SignalEvent()	发送事件信号。
GUI_X_Unlock()	解锁 GUI （取消阻止资源信号量 / 互斥体）。
GUI_X_WaitEvent()	等待事件。

GUI_X_GetTaskID()

描述

返回当前任务的唯一 ID。

原型

```
U32 GUI_X_GetTaskID(void);
```

返回值

当前任务的 ID，是一个 32 位整数。

其他信息

该函数用于实时操作系统。
只要返回值对于使用 emWin API 的每个任务 / 线程是唯一的，且只要该值对于每个特定线程始终相同，则返回值是多少并不重要。

GUI_X_InitOS()

描述

创建通常由 GUI_X_Lock() 和 GUI_X_Unlock() 使用的资源信号量或互斥体。

原型

```
void GUI_X_InitOS(void)
```

GUI_X_Lock()

描述

锁定 GUI。

原型

```
void GUI_X_Lock(void);
```

其他信息

此例程在 GUI 访问显示器之前或在使用关键内部数据结构之前由 GUI 调用，它使用资源信号量 / 互斥体阻止其它线程进入同一临界区，直到 GUI_X_Unlock() 已被调用。
在使用实时操作系统时，通常必须按增量增加计数资源信号量。

GUI_X_SignalEvent()

描述
发送事件信号。

原型
```
void GUI_X_SignalEvent(void);
```

其他信息
此函数为可选函数，仅通过宏 GUI_X_SIGNAL_EVENT 或函数 GUI_SetSignalEventFunc() 使用。

GUI_X_Unlock()

描述
解锁 GUI。

原型
```
void GUI_X_Unlock(void);
```

其他信息
此例程在访问显示器之后或在使用关键内部数据结构之后由 GUI 调用。
在使用实时操作系统时，通常必须减量计数资源信号量。

GUI_X_WaitEvent()

描述
等待事件。

原型
```
void GUI_X_WaitEvent(void);
```

其他信息
此函数为可选函数，仅通过宏 GUI_X_WAIT_EVENT 或函数 GUI_SetWaitEventFunc() 使用。

GUI_X_WaitEventTimed()

描述
在给定时段内等待事件。

原型
```
void GUI_X_WaitEventTimed(int Period);
```

参数	描述
Period	要使用的时段，单位为 ms。

其他信息
此函数为可选函数，仅通过宏 GUI_X_WAIT_EVENT_TIMED 或函数 GUI_SetWaitEventTimedFunc() 使用。

13.7.1 示例

Win32 的内核接口例程 embOS

以下示例显示了 embOS 改编 （摘自位于文件夹 Sample\GUI_X 中的文件 GUI_X_embOS.c）：

```
#include "RTOS.H"

static OS_TASK* _pGUITask;
static OS_RSEMA _RSema;

void GUI_X_InitOS(void)     { OS_CreateRSema(&_RSema);     }
void GUI_X_Unlock(void)     { OS_Unuse(&_RSema);           }
void GUI_X_Lock(void)       { OS_Use(&_RSema);             }
U32  GUI_X_GetTaskId(void) { return (U32)OS_GetTaskID(); }

void GUI_X_WaitEvent(void) {
  _pGUITask = OS_GetpCurrentTask();
  OS_WaitEvent(1);
}

void GUI_X_SignalEvent(void) {
  if (_pGUITask) {
    OS_SignalEvent(1, _pGUITask);
  }
}

void GUI_X_WaitEventTimed(int Period) {
  static OS_TIMER Timer;
  static int Initialized;

  if (Period > 0) {
    if (Initialized != 0) {
      OS_DeleteTimer(&Timer);
    }
    Initialized = 1;
    OS_CreateTimer(&Timer, GUI_X_SignalEvent, Period);
    OS_StartTimer(&Timer);
    GUI_X_WaitEvent();
  }
}
```

uC/OS 的内核接口例程

以下示例显示了 uC/OS 改编 （摘自位于文件夹 Sample\GUI_X 中的文件 GUI_X_uCOS.c）：

```
#include "INCLUDES.H"

static OS_EVENT * pDispSem;
static OS_EVENT * pGUITask;

U32   GUI_X_GetTaskId(void) { return ((U32)(OSTCBCur->OSTCBPrio)); }
void  GUI_X_Unlock(void)    { OSSemPost(pDispSem);                  }

void  GUI_X_InitOS(void)    {
  pDispSem = OSSemCreate(1);
  pGUITask = OSSemCreate(0);
}

void  GUI_X_Lock(void)      {
  INT8U err;
  OSSemPend(pDispSem, 0, &err);
}
```

Win32 的内核接口例程

下面内容摘自 emWin 的 Win32 模拟。（在使用 emWin 模拟时，无需添加这些例程，因为它们已在库中。）
注：为简明起见，已省略清理代码。

```
/*********************************************************************
*
*      emWin - Multitask interface for Win32
*
**********************************************************************
```



```
  The following section consisting of 4 routines is used to make
  emWin thread safe with WIN32
*/

static HANDLE hMutex;

void GUI_X_InitOS(void) {
  hMutex = CreateMutex(NULL, 0, "emWinSim - Mutex");
}

unsigned int GUI_X_GetTaskId(void) {
  return GetCurrentThreadId();
}

void GUI_X_Lock(void) {
  WaitForSingleObject(hMutex, INFINITE);
}

void GUI_X_Unlock(void) {
  ReleaseMutex(hMutex);
}
```


第 14 章

窗口管理器 (WM)

使用 emWin 窗口管理器 (WM) 时，显示器上出现的任何内容都包含在窗口中 – 屏幕上的一个矩形区域。窗口可以为任何尺寸，并且可在屏幕上一次显示多个窗口，甚至部分或整个窗口在其他窗口的前面也可。
窗口管理器提供一组例程，利用这些例程可以很容易地对窗口进行创建、移动、调整大小，另外还能操控任意数量的窗口。它还通过管理显示器中窗口的分层来提供更低级别的支持，并通过提醒应用程序来显示影响其窗口的更改。
emWin是单独（可选）的软件项目，不包括在emWin基础包中。存储设备的软件位于子目录GUI\WM中。

14.1 术语说明

窗口是矩形的，由其原点（左上角的 X 和 Y 坐标）以及 X 和 Y 值（分别为宽度和高度）定义。emWin 中的窗口：

- 是矩形的。
- 具有 Z 位置。
- 可以隐藏或显示。
- 可具有有效和 / 或无效区域。
- 可以透明或不透明。
- 可以具有或不具有回调例程。

活动窗口

当前正用于绘制操作的窗口称为活动窗口，不一定就是最上面的窗口。

回调例程

回调例程由用户程序定义，指示在特定事件出现时图形系统调用特定的函数。它们通常用于在窗口内容更改时自动重绘窗口。

子窗口 / 父窗口 , 同属窗口

子窗口是相对于其他窗口（称为父窗口）定义的。只要父窗口移动，其子窗口就会相应移动。子窗口始终完全包含在其父窗口中，并在必要时会被裁剪。具有相同父窗口的多个子窗口被视为“同属”窗口。

客户区

窗口的客户区就是其可用区域。如果窗口包含边框或标题栏，则客户区是内部的矩形区域。如果没有这种边框，则客户区的坐标与窗口自身的坐标相同。

裁剪 , 裁剪区域

裁剪是将输出限制为一个窗口或窗口一部分的过程。
窗口的裁剪区域是其可见区域。它是窗口区域减去被更高 Z 轴阶层的同属窗口遮挡的区域，然后减去没有放入父窗口可见区域的任何部分。

坐标

坐标通常是 2 维坐标，以像素单位表示。坐标由 2 个值组成。第一个值指定水平分量 （也称为 x 坐标），第二个值指定垂直分量 （也称为 y 坐标）。

桌面坐标

桌面坐标是桌面窗口的坐标。显示器的左上角位置 （原点）为 (0,0)。

桌面窗口

桌面窗口由窗口管理器自动创建，并且始终覆盖整个显示区域。它始终是最底层的窗口，在没有定义其他窗口时，它是默认 （活动）窗口。所有窗口都是桌面窗口的后代窗口 （子窗口、孙窗口等）。

句柄

创建新窗口后，WM 会分配一个称为句柄的唯一标识符。句柄用于在该特定窗口中执行的其他任何操作。

隐藏 / 显示窗口

隐藏的窗口不可见，尽管仍然存在 （有一个句柄）。创建窗口时，如果不指定创建标记，默认情况下是隐藏的。显示窗口使其可见，隐藏窗口则使其不可见。

父坐标

父坐标是与父窗口相关的窗口坐标。窗口的左上角位置 （原点）为 (0,0)。

透明性

具有透明区域的窗口包含不与窗口其余部分一起重绘的区域。这些区域就象其背后窗口 “透过” 它们显示一样。在此情况下，在有透明区域的窗口之前重绘背后窗口非常重要。 WM 自动按正确的顺序进行重绘。

有效化 / 无效化

有效窗口是不需要重绘的完全更新窗口。
无效窗口不会反映所有更新，因此需要完全或部分重绘。作出的更改影响了特定窗口时， WM 将该窗口标记为无效。下次窗口重绘时 （手动或通过回调例程），将进行验证。

窗口坐标

窗口坐标是窗口的坐标。窗口的左上角位置 （原点）为 (0,0)。

Z 位置 , 底部 / 顶部

尽管窗口显示在以 X 和 Y 表示的二维屏幕上，但是 WM 也管理所谓的 Z 位置或深度坐标 -- 虚拟的第三维上的位置，该坐标确定从背景到前景的位置。各窗口因此可在其他窗口之上或之下出现。
将某窗口设置为底部，会将该窗口置于其所有同属窗口 （如果有）的 “底部”；设置为顶部，则将其置于其同属窗口的 “顶部”。创建窗口时，如果不指定创建标记，默认情况下设置为顶部。

14.2 回调机制，无效化和渲染

WM 可在有或无回调例程时使用。大多数情况下，最好使用回调。
emWin 为窗口和窗口对象 （小工具）提供回调机制的根本概念是一个事件驱动系统。因为在大多数窗口式系统中，其原理是控制流不仅仅是从用户程序到图形系统，而且还从用户程序到图形系统，然后再通过用户程序提供的回调例程返回用户程序。此机制通常称为 “好莱坞原则”（“不要打电话给我们，我们会给你打电话的！”），窗口管理器需要它的主要目的是触发窗口重绘。这与传统编程相反，但是它能利用窗口管理器的无效化逻辑。

14.2.1 不使用回调渲染

回调例程不是必须使用的，但是如果这样做， WM 会失去管理窗口重绘（更新）的能力。也可能混合使用，例如让有些窗口使用回调，有些不使用。当然，如果窗口不使用回调机制，则由应用程序负责更新其内容。

警示：不使用回调机制时，您有责任管理屏幕更新！

14.2.2 使用回调渲染

要创建带回调的窗口，必须有一个回调例程。创建窗口（cb 参数）时该例程用作 WM_CreateWindow() 函数的一部分。
回调例程必须具有以下原型：

原型

```
void callback(WM_MESSAGE * pMsg);
```

参数	描述
pMsg	指向类型 WM_MESSAGE 的数据结构指针。

回调例程执行的操作取决于其收到的消息的类型。上述原型通常后跟一个 switch 语句，它使用一个或多个 case 语句为不同消息定义不同的行为 （通常至少是 WM_PAINT）。

处理 WM_PAINT 消息

窗口收到 WM_PAINT 消息时，应重绘自身。将此消息发送到窗口前， WM 确保它已被选定。
非透明窗口 （默认！）必须重绘其整个无效区域。
最简单的方式是重新着色窗口的整个区域。WM 的裁剪机制确保了仅重绘无效区域。为了加速绘制过程，仅重绘无效区域非常有用。本章稍后描述了如何获得无效区域 （信息是消息的一部分）。
另一方面，**透明窗口**不必重绘整个无效区域；它可让窗口区域部分不受影响。此不受影响的区域然后会变成透明。
WM 发送 WM_PAINT 消息到透明窗口之前，以下区域已经重绘 （通过发送一条 WM_PAINT 消息到下面窗口）。

警示：处理 WM_PAINT 时，不得执行某些操作

处理 WM_PAINT 消息时，回调例程除了重绘窗口的内容外，不得执行其它操作。处理 WM_PAINT 事件时，下列函数不能调用：WM_SelectWindow()、 WM_Paint()、 WM_DeleteWindow() 和 WM_CreateWindow()。更改窗口属性的其他任何函数也不能调用：WM_Move()、WM_Resize()、...

示例

创建一个自动重绘窗口的回调例程：

```
void WinHandler(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_SetBkColor(0xFF00);
    GUI_Clear();
    GUI_DispStringAt("Hello world",0,0);
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}
```

请注意， WM_PRE_PAINT 和 WM_POST_PAINT 消息在 WM_PAINT 消息发送前和后立即处理。

14.2.3 背景窗口重绘和回调

初始化窗口管理器期间，会创建一个包含整个 LCD 区域的窗口作为背景窗口。此窗口的句柄为 WM_HBKWIN。 WM 不会自动重绘背景窗口的区域，因为没有默认的背景颜色。也就是说如果创建了另一个窗口然后将其删除，则删除的窗口仍然可见。需要指定例程 WM_SetDesktopColor() 以便为重绘背景窗口设置颜色。

也可设置回调函数来处理此问题。如果如前所述创建然后删除了一个窗口，则回调例程将触发 WM 来确认背景窗口不再有效，并自动重绘它。有关使用回调例程重绘背景的详细信息，请参阅本章末尾的示例。

14.2.4 无效化

无效化窗口或窗口的一部分会告诉WM该窗口的无效区域在下次调用GUI_Exec()或GUI_Delay()时应重绘。 emWin 的无效化例程不会重绘窗口的无效部分，只管理窗口的无效区域。

窗口的无效区域

对于每个窗口， WM 只使用一个矩形来存储包含整个无效区域的最小矩形。例如，如果左上角的一小部分和右下角的一小部分变为无效，则整个窗口即无效。

使用无效化的原因

使用窗口无效化而非立即重绘每个窗口的优点是只绘制窗口一次，即使其被无效化多次。例如，如果窗口的多个属性需要更改（如背景颜色、字体和窗口大小），与所有属性都更改后重绘一次窗口相比，每个属性更改后立即重绘窗口需要更多的次数。

重绘无效窗口

函数GUI_Exec()重绘所有无效窗口。这通过向每个无效窗口发送一条或多条WM_PAINT消息完成。

14.2.5 渲染透明窗口

如果需要绘制透明窗口，则 WM 会自动确保在透明窗口收到 WM_PAINT 消息前绘制窗口的背景。方法为，在向透明窗口发送 WM_PAINT 消息前，首先重绘透明窗口无效区域下面的所有窗口区域。
为确保窗口管理器能执行透明窗口的重绘，有必要响应 WM_PAINT 消息并重绘窗口。否则，不能保证透明窗口的外观是正确的。
使用透明窗口比使用不透明窗口需要更多消耗 CPU。如果需要考虑性能问题，尝试避免使用透明窗口或许是一个选择。

14.2.6 自动使用存储设备

窗口管理器的默认特性是向每个需要重绘的窗口发送一条 WM_PAINT。这会导致闪变效应。为抑制这些每个窗口的闪变效应，可为重绘操作自动使用存储设备。方法为，在创建窗口时设置标记WM_CF_MEMDEV、使用函数 WM_SetCreateFlags() 设置默认创建标记，或使用函数 WM_EnableMemdev()。 WM 然后会将 WM_PAINT 消息输出重定向到存储设备中，再复制到显示器中。如果整个窗口的内存不够，会自动使用分段。存储设备只是临时使用，在绘制操作完成后会移除。更多详细信息，请参阅“存储设备”（第 247 页）。

14.2.7 自动使用多帧缓冲

WM 可自动使用多帧缓冲 （如果可用），这可通过函数 WM_MULTIBUF_Enable() 实现。启用后，在绘制无效窗口前，窗口管理器会将所有绘制函数的输出重定向到不可见的后置缓冲。绘制最后一个无效窗口后，WM 使后置缓冲可见。请注意，仅在显示驱动支持多缓冲，并且有至少够 2 帧缓冲使用的足够 RAM 时，该功能才可用。更多详细信息，请参阅 “多缓冲”（第 729 页）。

14.2.8 自动使用显示驱动缓存

WM 自动使用显示驱动缓存 （如果可用）。如果可用，它会在开始绘制无效窗口前锁定缓冲。最后一个窗口绘制完成后， WM 解锁缓存。

14.3 消息

以下几节列出了emWin使用的系统消息、如何使用这些消息数据以及如何使用应用程序定义的消息。

14.3.1 消息结构

回调例程被调用时，它会收到以其 pMsg 参数指定的消息。此消息实际上是一个 WM_MESSAGE 数据结构，其元素定义如下。

WM_MESSAGE 的元素

数据类型	元素	描述
int	MsgId	消息的类型 （参见下表）。
WM_HWIN	hWin	目标窗口。
WM_HWIN	hWinSrc	源窗口。
void*	Data.p	数据指针。
int	Data.v	数据值。

14.3.2 消息清单

以下是由 emWin 定义的消息。

消息 Id (MsgId)	描述
系统定义的消息	
WM_CREATE	窗口创建后立即发送，使窗口可以初始化并创建任何子窗口。
WM_DELETE	要删除窗口前发送，告诉窗口释放其数据结构 （如果有）。
WM_GET_ID	发送到窗口，请求窗口的 Id。
WM_INIT_DIALOG	创建对话框后立即发送到对话框窗口。
WM_KEY	按下某个键后发送到当前包含焦点的窗口。
WM_MOVE	窗口移动后立即发送到窗口。
WM_NOTIFY_PARENT	告知父窗口，其子窗口中发生了某些改变。
WM_NOTIFY_VIS_CHANGED	窗口可见性更改时发送到窗口。
WM_PAINT	窗口变为无效并应重绘时，发送到窗口。
WM_POST_PAINT	最后一条 WM_PAINT 消息处理后发送到窗口。
WM_PRE_PAINT	第一条 WM_PAINT 消息发出前发送到窗口。
WM_SET_FOCUS	获得或失去输入焦点时发送到窗口。
WM_SET_ID	发送到窗口以更改窗口 Id。
WM_SIZE	窗口大小更改后发送到窗口。
WM_TIMER	定时器到期后发送到窗口。
指针输入设备 (PID) 消息	
WM_MOUSEOVER	指针输入设备接触到窗口轮廓时发送到窗口。仅在启用鼠标支持时发送。
WM_MOUSEOVER_END	指针输入设备已移出窗口轮廓时发送到窗口。仅在启用鼠标支持时发送。
WM_PID_STATE_CHANGED	按下状态已更改时，发送到指针输入设备指向的窗口。
WM_TOUCH	指针输入设备接触到处于按下状态的窗口轮廓时发送到窗口。
WM_TOUCH_CHILD	指针输入设备已接触到子窗口时发送到父窗口。
通知代码	
WM_NOTIFICATION_CHILD_DELETED	此通知消息将在某窗口被删除前，从该窗口发送到其父窗口。
WM_NOTIFICATION_CLICKED	此通知消息将在点击窗口后发送。
WM_NOTIFICATION_LOST_FOCUS	此通知消息将在窗口失去焦点时发送。
WM_NOTIFICATION_MOVED_OUT	此通知消息将在指针移出窗口并点击时发送。
WM_NOTIFICATION_RELEASED	此通知消息将在被点击的小工具已被释放时发送。

消息 Id (MsgId)	描述
WM_NOTIFICATION_SCROLL_CHANGED	此通知消息将在附加的 SCROLLBAR（滚动条）小工具的滚动位置更改时发送。
WM_NOTIFICATION_SCROLLBAR_ADDED	此通知消息在将 SCROLLBAR 小工具添加到窗口时发送。
WM_NOTIFICATION_SEL_CHANGED	此通知消息将在小工具选择已更改时发送。
WM_NOTIFICATION_VALUE_CHANGED	此通知消息将在小工具的特定值已更改时发送。
用户定义的消息	
WM_USER	应用程序可使用 WM_USER 常数来定义私人消息，通常形式为 WM_USER+X，其中 X 为整数值。

14.3.3 系统定义的消息

此类型的消息由 GUI 库发送。请勿从用户应用程序向窗口或小工具发送系统定义的消息。

WM_CREATE

描述

此消息在窗口创建后立即发出，使窗口可以初始化并创建任何子窗口。

数据

此消息不包含数据。

WM_DELETE

描述

此消息在窗口要删除前发出，告诉窗口释放其数据结构 （如果有）。

数据

此消息不包含数据。

WM_GET_ID

描述

此消息发送到窗口以请求其 Id。所有 emWin 小工具都处理此消息。应用程序定义的窗口应在其回调例程中处理此消息，否则此消息将被忽略。

数据

窗口的回调例程应将 Id 存储在 Data.v 值中。

WM_INIT_DIALOG

描述

此消息在对话框创建后、显示前发送到窗口。对话框程序通常使用此消息初始化小工具，并执行影响对话框外观的其他任何初始化任务。

数据

此消息不包含数据。

WM_KEY

描述

按下某个键后发送到当前包含焦点的窗口。

数据

指向 WM_KEY_INFO 结构的消息的 Data.p 指针。

WM_KEY_INFO 的元素

数据类型	元素	描述
int	Key	已按下的键。
int	PressedCount	>0（如果键已按下），0（如果键已释放）。

WM_MOVE

描述

窗口移动后立即发送到窗口。
如果窗口有任何子窗口，则将首先移动它们。每个子窗口移动后也会收到此消息。
无论窗口是否可见消息都会发送。

数据

此消息不包含数据。

WM_NOTIFY_PARENT

描述

告知父窗口，其子窗口中发生了某些改变。这些消息通常由小工具发送到其父窗口，让它们可以对事件作出反应。

数据

消息的 Data.v 值包含消息的通知代码。有关通知代码的详细信息，请参阅相应的小工具。

WM_NOTIFY_VIS_CHANGED

描述

如果窗口的可见性发生改变，则此消息发送到窗口并将配置开关 WM_SUPPORT_NOTIFY_VIS_CHANGED 设置为 1。遮挡改变时窗口的可见性改变：

- 窗口部分或整个被更高层级的窗口 （在窗口顶部显示的窗口）覆盖或露出，
- 窗口被删除或
- 窗口从不隐藏变成隐藏或相反。

典型应用

使用硬件解码器在窗口中播放视频的应用。如果包含视频的窗口完全可见，则硬件解码器可直接写入显示器，绕过 emWin。如果可见性更改，则硬件解码器需要重新编程。

示例

以下为此消息的典型反应：

```
case WM_NOTIFY_VIS_CHANGED:
  if (WM_IsCompletelyVisible(WM_GetClientWindow(pMsg->hWin))) {
    ...
  }
```

emWin 的 “样本”文件夹中包含示例 WM_Video.c，它显示了如何使用该函数。

数据

此消息不包含数据。

WM_PAINT

描述

窗口变为无效 （部分或全部）并需要绘制时， WM 将此消息发送到窗口。窗口收到 WM_PAINT 消息时，应重绘自身。将此消息发送到窗口前，WM 确保它已被选定。有关 WM_PAINT 消息反应的详细信息，在本章稍早的 “使用回调例程”中已进行了描述。

数据

该消息的 Data.p 指针指向在屏幕坐标中包含窗口的无效矩形的 GUI_RECT 结构。此类信息可用于优化着色功能。

WM_POST_PAINT

描述

在最后一条 WM_PAINT 消息处理后， WM 立即将此消息发送到窗口。

数据

此消息不包含数据。

WM_PRE_PAINT

描述

第一条 WM_PAINT 消息发出前， WM 将此消息发送到窗口。

数据

此消息不包含数据。

WM_SET_FOCUS

描述

获得或失去输入焦点时发送到窗口。

数据

如果窗口获得输入焦点，则 Data.v 值设置为 1。如果窗口 “接受”该输入焦点，则应将 Data.v 值设置为 0，作为对此消息的反应。
如果窗口失去输入焦点，则 Data.v 值设置为 0。

WM_SET_ID

描述

发送到窗口以更改 Id。所有 emWin 小工具都处理此消息。应用程序定义的窗口应在其回调例程中处理此消息，否则此消息将被忽略。

数据

Data.v 值包含窗口的新 Id。

WM_SIZE

描述

窗口大小更改后发送到窗口。使窗口可以重新定位其子窗口 （如果有）。

数据

此消息不包含数据。

WM_TIMER

描述

WM_CreateTimer() 创建的定时器到期时，此消息发送到窗口。

数据

Data.v 包含到期定时器的句柄。

14.3.4 指针输入设备 (PID) 消息

此类消息由 GUI 库发送，作为对 PID 输入的反应。请勿从用户应用程序向窗口或小工具发送此消息。

WM_MOUSEOVER

描述

指针输入设备接触到窗口轮廓时，WM_MOUSEOVER 消息发送到窗口。仅在启用了鼠标支持时发送。此消息不发送到禁用的窗口。
要启用鼠标支持，请将以下代码行添加到文件 GUIConf.h 中：

```
#define GUI_SUPPORT_MOUSE 1
```

数据

指向 GUI_PID_STATE 结构的消息的 Data.p 指针。

GUI_PID_STATE 的元素

数据类型	元素	描述
int	x	窗口坐标中 PID 的水平位置。
int	y	窗口坐标中 PID 的垂直位置。
U8	Pressed	收到 WM_MOUSEOVER 消息时始终设置为 0。

WM_MOUSEOVER_END

描述

鼠标指针已移出窗口时， WM_MOUSEOVER_END 消息发送到窗口。仅在启用了鼠标支持时发送。此消息不发送到禁用的窗口。

数据

指向 GUI_PID_STATE 结构的消息的 Data.p 指针。有关绘制模式的详情，请参阅 WM_MOUSEOVER 函数。

WM_PID_STATE_CHANGED

描述

按下状态已更改时，发送到受指针输入设备影响的窗口。受影响的窗口是在输入位置的可见窗口。换句话说：例如，如果用户在窗口上离开了触摸屏，则按下状态从 1 （按下）变成 0 （未按下）。在此情况下，WM_PID_STATE_CHANGED 消息发送到窗口。此消息在触摸消息发送前发送。不可见窗口不会接收此消息。透明窗口的处理方式与可见窗口一样。
此消息不发送到禁用的窗口。

数据

指向 WM_PID_STATE_CHANGED_INFO 结构的消息的 Data.p 指针。

WM_PID_STATE_CHANGED_INFO 的元素

数据类型	元素	描述
int	x	窗口坐标中 PID 的水平位置。
int	y	窗口坐标中 PID 的垂直位置。
U8	State	按下状态 （>0，如果 PID 已按下）。
U8	StatePrev	前一按下状态

WM_TOUCH

描述

出现以下情况时， WM_TOUCH 消息发送到窗口

- PID 处于按下状态，且指针位于窗口可见部分上
- 或 PID 刚刚释放。

此消息不发送到禁用的窗口。

数据

指向GUI_PID_STATE结构的消息的Data.p指针。有关位图转换器的详细信息，请参阅WM_MOUSEOVER。

GUI_PID_STATE 的元素

数据类型	元素	描述
int	x	窗口坐标中 PID 的水平位置。
int	y	窗口坐标中 PID 的垂直位置。
U8	Pressed	如果消息由触摸屏引起，此值可以为 0 （未按下）或 1 （按下）。 如果消息由鼠标引起，则每位代表一个鼠标按钮 （0 表示未按下， 1 表示按下）： - 位 0 代表第一个按钮 （通常是左按钮） - 位 1 代表第二个按钮 （通常是右按钮） - 位 2 代表第三个按钮 （通常是中间按钮） 余下的位可用于其他按钮。

WM_TOUCH_CHILD

描述

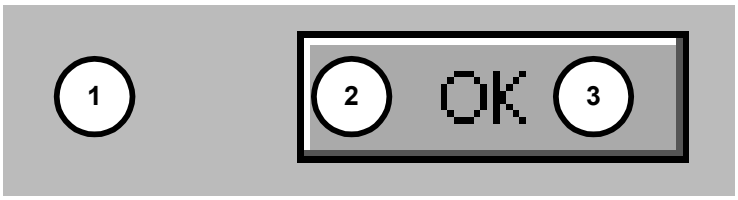
子窗口的轮廓被处于按下或未按下状态的指针输入设备接触时，此消息发送到父窗口。
此消息不发送到禁用的窗口。

数据

该消息的Data.p指针指向发送到子窗口的触摸消息。有关消息数据的详细信息，请参阅“WM_TOUCH”（第 301 页）。

示例

以下示例描述了指针输入设备从有按钮的对话框上拖过时的情况：



位置	描述
1	指针输入设备 (PID) 在此位置按下，从而导致 WM 将以下 WM_PID_STATE_CHANGED 消息发送到在此位置的窗口： x = 桌面坐标的水平位置。 y = 桌面坐标的垂直位置。 State = 1 StatePrev = 0 WM 也将具有相同 x 和 y 坐标的 WM_TOUCH 消息发送到窗口： x = 桌面坐标的水平位置。 y = 桌面坐标的垂直位置。 按下 = 1
2	PID 拖动到此位置。下面窗口 （按钮）将不会收到 WM_PID_STATE_CHANGED 消息，因为 PID 保持在按下状态。 WM 只将一条 WM_TOUCH 消息发送到窗口： x = 桌面坐标的水平位置。 y = 桌面坐标的垂直位置。 按下 = 1
3	PID 在此位置松开。从而导致 WM 将以下 WM_PID_STATE_CHANGED 消息发送到在此位置的窗口： x = 桌面坐标的水平位置。 y = 桌面坐标的垂直位置。 State = 0 StatePrev = 1 WM 也将具有相同 x 和 y 坐标的 WM_TOUCH 消息发送到窗口： x = 桌面坐标的水平位置。 y = 桌面坐标的垂直位置。 按下 = 0

14.3.5 系统定义的通知代码

此类型消息从窗口发送到其父窗口，通知父窗口子窗口有变化，这使得父窗口可以对此事件作出反应。该消息包含 hWinSrc 元素，它是引发消息的小工具的句柄。有关各种小工具可发送哪些通知消息的详细信息，请参阅 “小工具”一章中的相应小工具。

注：请勿从用户应用程序将系统定义的通知代码发送到窗口。

WM_NOTIFICATION_CHILD_DELETED

此通知消息将在某窗口被删除前，从该窗口发送到其父窗口。

WM_NOTIFICATION_CLICKED

此通知消息将在点击窗口后发送。

WM_NOTIFICATION_LOST_FOCUS

此通知消息将在窗口失去焦点时发送。

WM_NOTIFICATION_MOVED_OUT

此通知消息将在指针移出窗口并点击时发送。

WM_NOTIFICATION_RELEASED

此通知消息将在被点击的小工具已被释放时发送。

WM_NOTIFICATION_SCROLL_CHANGED

此通知消息将在附加的 SCROLLBAR （滚动条）小工具的滚动位置更改时发送。

WM_NOTIFICATION_SCROLLBAR_ADDED

此通知消息在将 SCROLLBAR 小工具添加到窗口时发送。

WM_NOTIFICATION_SEL_CHANGED

此通知消息将在小工具选择已更改时发送。

WM_NOTIFICATION_VALUE_CHANGED

此通知消息将在小工具的特定值已更改时发送。

14.3.6 应用定义的消息

应用程序可根据自己的应用定义其它消息。为确保它们不会使用与 emWin 相同的消息标识，用户定义的消息在 WM_USER 后开始编号。可按如下方式定义自己的消息：

```
#define MY_MESSAGE_AAA WM_USER+0
#define MY_MESSAGE_BBB WM_USER+1
```

等等。

14.4 配置选项

类型	宏	默认值	描述
B	WM_SUPPORT_NOTIFY_VIS_CHANGED	0	使 WM 在窗口可见性变化时将 WM_NOTIFY_VIS_CHANGED 消息发送到窗口。
B	WM_SUPPORT_TRANSPARENCY	1	启用对透明窗口的支持。如果设置为 0，则不包括透明性支持的附加代码。

WM_SUPPORT_NOTIFY_VIS_CHANGED

默认情况下，emWin不会通知窗口其可见性是否已更改。启用后，WM发送WM_NOTIFY_VIS_CHANGED消息。

WM_SUPPORT_TRANSPARENCY

默认情况下， emWin 支持透明窗口。这意味着默认情况下，如果使用 WM，则链接用于处理透明窗口的附加代码。如果应用程序不使用透明窗口，且 WM_SUPPORT_TRANSPARENCY 设置为 0，则应用程序的内存要求可减少。

14.5 WM API

下表列出了 emWin 存储设备 API 的可用例程。
所有功能都在其各自类别中按字母顺序列出。各例程的详细描述请见本章稍后部分。

例程	描述
基本函数	
WM_Activate()	激活窗口管理器。
WM_AttachWindow()	将窗口附加到一个新的父窗口。
WM_AttachWindowAt()	在给定位置将窗口附加到一个新的父窗口。
WM_BroadcastMessage()	发送消息到所有现有窗口。
WM_BringToBottom()	将窗口放在其同属窗口后面。
WM_BringToTop()	将窗口放在其同属窗口前面。
WM_ClrHasTrans()	清除“有透明性”标记。
WM_CreateWindow()	创建窗口。
WM_CreateWindowAsChild()	创建子窗口。
WM_Deactivate()	去激活窗口管理器。
WM_DefaultProc()	句柄消息的默认例程。
WM_DeleteWindow()	删除窗口。
WM_DetachWindow()	断开窗口与其父窗口的关联。
WM_DisableWindow()	将小工具状态设置为禁用。
WM_EnableWindow()	将窗口状态设置为启用 （默认）。
WM_Exec()	通过执行回调重绘无效窗口 （所有工作）。
WM_Exec1()	通过执行一个回调重绘一个无效窗口 （仅一项工作）。
WM_ForEachDesc()	迭代窗口的所有后代。
WM_GetActiveWindow()	返回活动窗口的句柄。
WM_GetCallback()	返回窗口回调函数的指针。
WM_GetClientRect()	返回活动窗口的尺寸。
WM_GetClientRectEx()	返回窗口的尺寸。
WM_GetDesktopWindow()	返回桌面窗口的窗口句柄
WM_GetDesktopWindowEx()	返回指定桌面窗口的窗口句柄
WM_GetDialogItem()	返回对话框项目 （小工具）的窗口句柄。
WM_GetFirstChild()	返回窗口第一个子窗口的句柄。
WM_GetFocussedWindow()	返回具有输入焦点的窗口的句柄。
WM_GetHasTrans()	返回 “ 有透明性”标记的当前值。
WM_GetInvalidRect()	返回给定窗口的无效矩形。
WM_GetNextSibling()	返回指定页面的窗口句柄。
WM_GetOrgX()	返回活动窗口 X 方向的原点。
WM_GetOrgY()	返回活动窗口 Y 方向的原点。
WM_GetParent()	返回窗口的父窗口的句柄。
WM_GetPrevSibling()	返回窗口前一同属窗口的句柄。
WM_GetStayOnTop()	返回 “保持在顶部”标记的当前值。
WM_GetUserData()	检索窗口的用户数据
WM_GetWindowOrgX()	返回窗口 X 方向的原点。
WM_GetWindowOrgY()	返回窗口 Y 方向的原点。
WM_GetWindowRect()	返回活动窗口的屏幕坐标。
WM_GetWindowRectEx()	返回窗口的屏幕坐标。
WM_GetWindowSizeX()	返回窗口的水平尺寸 （宽度）。
WM_GetWindowSizeY()	返回窗口的垂直尺寸 （高度）。
WM_HasCaptured()	检查给定窗口是否已捕获鼠标和触摸屏输入。
WM_HasFocus()	检查给定窗口是否具有输入焦点。
WM_HideWindow()	使窗口不可见。

例程	描述
WM_InvalidateArea()	使显示器的某些部分无效。
WM_InvalidateRect()	使部分窗口无效。
WM_InvalidateWindow()	使窗口无效。
WM_IsCompletelyCovered()	检查窗口是否完全被覆盖。
WM_IsCompletelyVisible()	检查窗口是否完全可见。
WM_IsEnabled()	返回指定页面的启用状态。
WM_IsVisible()	返回表示窗口是否可见的值。
WM_IsWindow()	确定指定句柄是否是有效的窗口句柄。
WM_MakeModal()	将窗口变成“模态”窗口。
WM_MoveChildTo()	设置窗口在窗口坐标中的位置。
WM_MoveTo()	设置窗口在桌面坐标中的位置。
WM_MoveWindow()	将窗口移动到其他位置。
WM_NotifyParent()	发送 WM_NOTIFY_PARENT 消息到给定窗口的父窗口。
WM_Paint()	立即绘制或重绘窗口。
WM_PaintWindowAndDescs()	立即绘制给定窗口和所有后代窗口。
WM_ReleaseCapture()	停止捕获鼠标和触摸屏输入。
WM_ResizeWindow()	改变窗口尺寸。
WM_SelectWindow()	设置要用于绘制操作的活动窗口。
WM_SendMessage()	向窗口发送消息。
WM_SendMessageNoPara()	向窗口发送无参数的消息。
WM_SendToParent()	将给定消息发送到给定窗口的父窗口。
WM_SetDesktopColor()	设置桌面窗口颜色。
WM_SetDesktopColorEx()	设置给定桌面的桌面窗口颜色。
WM_SetCallback()	设置窗口的回调例程。
WM_SetCapture()	开始捕获鼠标和触摸屏输入。
WM_SetCreateFlags()	设置在创建新窗口时用作默认的标记。
WM_SetFocus()	将输入焦点设置到指定窗口。
WM_SetHasTrans()	设置有透明性标记。
WM_SetId()	将 WM_SET_ID 消息发送到给定窗口。
WM_SetpfPollPID()	设置 WM 轮询 PID 而要调用的函数。
WM_SetSize()	设置窗口的新尺寸。
WM_SetWindowPos()	设置窗口的尺寸和位置。
WM_SetXSize()	设置窗口的新 X 尺寸。
WM_SetYSize()	设置窗口的新 Y 尺寸。
WM_SetStayOnTop()	设置保持在顶部标记。
WM_SetTransState()	设置或清除 WM_CF_HASTRANS 和 WM_CF_CONST_OUTLINE 标记。
WM_SetUserClipRect()	临时缩小裁剪区域。
WM_SetUserData()	设置给定窗口的用户数据。
WM_ShowWindow()	使窗口可见。
WM_Update()	绘制给定窗口的无效部分。
WM_UpdateWindowAndDescs()	绘制给定窗口的无效部分和所有后代窗口的无效部分。
WM_ValidateRect()	有效化部分窗口。
WM_ValidateWindow()	有效化窗口。
存储设备支持 （可选）	
WM_DisableMemdev()	禁用存储设备用于重绘。
WM_EnableMemdev()	启用存储设备用于重绘。
定时器相关	
WM_CreateTimer()	创建向窗口发送 WM_TIMER 消息的定时器。
WM_DeleteTimer()	删除定时器。
WM_GetTimerId()	获取给定定时器的 Id。
WM_RestartTimer()	重启定时器。

例程	描述
与小工具相关的函数	
WM_GetClientWindow()	返回客户端窗口的句柄。
WM_GetId()	返回小工具的 ID。
WM_GetInsideRect()	返回活动窗口减去边界后的尺寸。
WM_GetInsideRectEx()	返回窗口减去边界后的尺寸。
WM_GetScrollPosH()	返回窗口水平滚动条的滚动位置。
WM_GetScrollPosV()	返回窗口垂直滚动条的滚动位置。
WM_GetScrollState()	获取滚动条小工具的状态。
WM_SetScrollPosH()	设置窗口水平滚动条的滚动位置。
WM_SetScrollPosV()	设置窗口垂直滚动条的滚动位置。
WM_SetScrollState()	设置滚动条小工具的状态。

14.5.1 使用 WM API 函数

WM 的许多函数都有窗口句柄为参数，使用句柄时请遵守以下规则：

- 窗口句柄可以为 0。在此情况下，函数立即返回，除非函数的文档另有规定。
- 如果窗口句柄为 != 0，则应是一个有效句柄。 WM 不会检查给定的句柄是否有效。如果向函数提供了无效句柄，则函数会失败，甚至崩溃。

14.6 基本函数

WM_Activate()

描述

激活窗口管理器。

原型

```
void WM_Activate(void);
```

其他信息

默认情况下，WM 在初始化后即被激活。此函数仅在先前已调用了 WM_Deactivate() 时才需调用。

WM_AttachWindow()

描述

给定窗口将从父窗口分离，并附加到新的父窗口。新父窗口在窗口坐标中的新原点将与旧父窗口在窗口坐标中的旧原点相同。

原型

```
void WM_AttachWindow(WM_HWIN hWin, WM_HWIN hParent);
```

参数	描述
hWin	窗口句柄。
hWinParent	新父窗口的窗口句柄。

其他信息

如果给定的窗口句柄为 0 或两个句柄相同，则函数立即返回。
如果只有给定父窗口的句柄为 0，则该函数分离给定的窗口并返回；该窗口保持独立。

WM_AttachWindowAt()

描述

给定窗口将从父窗口分离，并附加到新的父窗口。给定的位置将用于设置该窗口在父窗口的窗口坐标中的原点。

原型

```
void WM_AttachWindowAt(WM_HWIN hWin, WM_HWIN hParent, int x, int y);
```

参数	描述
hWin	窗口句柄。
hWinParent	新父窗口的窗口句柄。
x	窗口在父窗口的窗口坐标中的 X 位置。
y	窗口在父窗口的窗口坐标中的 Y 位置。

其他信息

如果给定的窗口句柄为 0 或两个句柄相同，则函数立即返回。
如果只有一个给定的父窗口句柄为 0，则函数分离给定的窗口，将其移到新位置并返回；窗口保持独立。

WM_BringToBottom()

描述

将指定窗口放在其同属窗口下面。

原型

```
void WM_BringToBottom(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

窗口将放置在所有其他同属窗口的下面，但将留在其父窗口的前面。

WM_BringToTop()

描述

将指定窗口放在其同属窗口顶部。

原型

```
void WM_BringToTop(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

该窗口将放置在其他所有同属窗口及父窗口的顶部。

WM_BroadcastMessage()

描述

发送给定消息到所有现有窗口。

原型

```
int WM_BroadcastMessage(WM_MESSAGE * pMsg);
```

参数	描述
pMsg	指向要发送消息的指针。

其他信息

窗口不会删除自己或父窗口，作为对广播消息的反应。

WM_ClrHasTrans()

描述

清除有透明性标记 （设置为 0）。

原型

```
void WM_ClrHasTrans(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

在设置时，此标记告诉窗口管理器窗口包含不重绘的部分，因此将为透明。从而使 WM 知道需要在重绘窗口之前重绘背景，以确保透明部分正确还原。
用 WM_ClrHasTrans() 清除标记时， WM 不会在重绘窗口之前自动重绘背景。

WM_CreateWindow()

描述

在指定位置创建指定尺寸的窗口。

原型

```
WM_HWIN WM_CreateWindow(int x0,              int           y0,
                        int width,           int           height,
                        U8  Style,           WM_CALLBACK * cb
                        int NumExtraBytes);
```

参数	描述
x0	桌面坐标中的左上 X 位置。
y0	桌面坐标中的左上 Y 位置。
width	窗口的 X 尺寸。
height	窗口的 Y 尺寸。
Style	窗口创建标记，列在下面。
cb	回调例程的指针，或不使用回调时为 NULL。
NumExtraBytes	要分配的额外字节数，通常为 0。

参数 Style 的允许值 （可以通过 “OR”操作进行组合）	
WM_CF_ANCHOR_BOTTOM	相对于父窗口的底边固定新窗口的底边。如果父窗口底边的位置由于尺寸变化要进行调整，则新窗口的位置也进行调整。
WM_CF_ANCHOR_LEFT	相对于父窗口的左边固定新窗口的左边 （默认）。如果父窗口左边的位置由于尺寸变化要进行调整，则新窗口的位置也进行调整。
WM_CF_ANCHOR_RIGHT	相对于父窗口的右边固定新窗口的右边。如果父窗口右边的位置由于尺寸变化要进行调整，则新窗口的位置也进行调整。
WM_CF_ANCHOR_TOP	相对于父窗口的顶边固定新窗口的顶边 （默认）。如果父窗口顶边的位置由于尺寸变化要进行调整，则新窗口的位置也进行调整。
WM_CF_BGND	创建后将窗口置于背景中。
WM_CF_CONST_OUTLINE	此标记是对透明窗口的优化。它使窗口管理器有机会优化透明窗口的重绘和无效化。透明窗口通常作为背景的一部分进行重绘，其效率低于单独重绘窗口。但是，如果窗口半透明 （与背景 Alpha 混合 / 抗锯齿）或轮廓 （形状）改变时，此标记 “不会”使用。通常可以使用；在有疑问时不要使用它。
WM_CF_FGND	创建后将窗口置于前景中 （默认）。
WM_CF_HASTRANS	有透明性标记。对于客户区没有完全填充的窗口，必须定义。
WM_CF_HIDE	创建后隐藏窗口 （默认）。
WM_CF_LATE_CLIP	此标记可用于告诉 WM，应在绘制例程中进行裁剪 （迟裁剪）WM 的默认特性是早裁剪。也就是说在将 WM_PAINT 消息发送到窗口前，将计算裁剪矩形。可能有必要向窗口发送多条 WM_PAINT 消息，取决于其他现有窗口。如果使用 WM_CF_LATE_CLIP，WM 将确保只向无效窗口发送一条消息，裁剪将通过绘制例程完成。emWin 的 Sample 文件夹中包含展示效果的示例 WM_LateClipping.c。
WM_CF_MEMDEV	重绘时自动使用存储设备。 在大多数情况下，这可避免闪烁，也能提高输出速度，因为裁剪被简化。请注意，必须要有存储设备包 （并要在配置中启用）才能使用此标记。如果存储设备没有启用，则此标记被忽略。
WM_CF_MEMDEV_ON_REDRAW	第一次绘制窗口后， WM 会自动使用存储设备进行重绘。此标记可用作 WM_CF_MEMDEV 的替代。通常能加速窗口的初始渲染，但会保持无闪烁更新的优点。
WM_CF_SHOW	创建后显示窗口。
WM_CF_STAYONTOP	确保窗口停留在所有不用此标记创建的同属窗口顶部。

返回值

已创建窗口的句柄。

其他信息

使用 (OR) 运算符可将多个创建标记组合起来。
可使用负的位置坐标。

示例

创建带回调的窗口：

```
hWin2 = WM_CreateWindow(100, 10 ,180, 100, WM_CF_SHOW, &WinHandler, 0);
```

创建不带回调的窗口：

```
hWin2 = WM_CreateWindow(100, 10 ,180, 100,WM_CF_SHOW, NULL, 0);
```

WM_CreateWindowAsChild()

描述

将窗口创建为子窗口。

原型

```
WM_HWIN WM_CreateWindowAsChild(int x0, int y0,
                               int width, int height,
                               WM_HWIN hWinParent,
                               U8 Style,
                               WM_CALLBACK* cb
                                 int NumExtraBytes);
```

参数	描述
x0	父窗口在窗口坐标中的左上 X 位置。
y0	父窗口在窗口坐标中的左上 Y 位置。
width	窗口的 X 尺寸。如果为 0，则用父窗口客户区的 X 尺寸。
height	窗口的 Y 尺寸。如果为 0，则用父窗口客户区的 Y 尺寸。
hWinParent	父窗口的句柄。
Style	窗口创建标识（请参阅 WM_CreateWindow()）。
cb	回调例程的指针，或不使用回调时为 NULL。
NumExtraBytes	要分配的额外字节数，通常为 0。

返回值

子窗口的句柄。

其他信息

如果 hWinParent 参数设置为 0，则将背景窗口用作父窗口。
默认情况下，子窗口放置在父窗口和先前任何同属窗口的顶部，因此如果它们的 Z 位置不改变，则“最新的”窗口始终在最上面。
同属窗口的 Z 位置可以更改，尽管它们始终留在父窗口的顶部，无论其顺序如何。

WM_Deactivate()

描述

去激活窗口管理器。

原型

```
void WM_Deactivate(void);
```

其他信息

调用此函数后，裁剪区域被设置为整个 LCD 区域，且 WM 将不执行窗口回调函数。

WM_DefaultProc()

描述

默认的消息句柄。

原型

```
void WM_DefaultProc(WM_MESSAGE* pMsg)
```

参数	描述
pMsg	消息指针。

其他信息

使用此函数可处理未处理的消息，如下例所示：

```
static WM_RESULT cbBackgroundWin(WM_MESSAGE* pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_Clear();
  default:
    WM_DefaultProc(pMsg);
  }
}
```

WM_DeleteWindow()

描述

删除指定窗口。

原型

```
void WM_DeleteWindow(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

删除窗口前，它接收一条 WM_DELETE 消息。此消息通常用于删除所使用的任何对象 （小工具），并释放窗口动态分配的内存。
如果指定的窗口具有任何现有子窗口，则在删除窗口前自动删除这些子窗口。因此不必分别删除子窗口。
窗口删除前，会发送一条 WM_NOTIFICATION_CHILD_DELETED 消息到其父窗。

WM_DetachWindow()

描述

断开窗口与其父窗口的关联。窗口管理器将不会重绘分离的窗口。

原型

```
void WM_DetachWindow(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

WM_DisableWindow()

描述

将指定窗口设置为禁用状态。 WM 不会将指针输入设备 (PID) 消息 （触摸、鼠标、操纵手柄 ...）发送到被禁用的窗口。

原型

```
void WM_DisableWindow(WM_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

其他信息

禁用的小工具通常显示为灰色，且不接受来自用户的输入。当然，实际外观可能会有所不同 （取决于 widget/configuration settings 等）。
禁用的窗口将不会接收以下消息：WM_TOUCH、WM_TOUCH_CHILD、WM_PID_STATE_CHANGED 和 WM_MOUSEOVER。

WM_EnableWindow()

描述

将指定的窗口设置为启用状态。启用的窗口接收来自 WM 的指针输入设备 (PID) 消息（触摸、鼠标、操纵手柄 ...）。

原型

```
void WM_EnableWindow(WM_Handle hObj);
```

参数	描述
hObj	窗口的句柄。

其他信息

这是所有小工具的默认设置。

WM_Exec()

描述

通过执行回调函数重绘无效窗口 （所有作业）。

原型

```
int WM_Exec(void);
```

返回值

0，如果没有已执行作业。
1，如果已执行作业。

其他信息

此函数将自动重复调用 WM_Exec1()，直至完成所有作业 – 实质是直至返回 0 值为止。
建议调用函数 GUI_Exec() 作为代替。
通常，此函数无需由用户应用程序调用，它自动由 GUI_Delay() 调用。如果使用的是多任务系统，建议通过单独的任务执行此函数，如下所示：

```
void ExecIdleTask(void) {
  while(1) {
    WM_Exec();
  }
}
```

WM_Exec1()

描述

通过执行一个回调函数重绘无效窗口 （仅一项工作）。

原型

```
int WM_Exec1(void);
```

返回值

0，如果没有已执行作业。
1，如果已执行作业。

其他信息

此例程可重复调用，直至返回 0，即意味着所有作业都已完成。
建议调用函数 GUI_Exec1() 作为代替。
此函数自动由 WM_Exec() 调用。

WM_ForEachDesc()

描述

迭代给定窗口的所有后代。窗口的后代是一个子窗口，或孙窗口，或孙窗口的子窗口，或 ....

原型

```
void WM_ForEachDesc(WM_HWIN hWin, WM_tfForEach * pcb, void * pData);
```

参数	描述
hWin	窗口句柄。
pcb	要由 WM_ForEachDesc 调用的回调函数的指针。
pData	要传递给回调函数的用户数据。

其他信息

此函数调用由给定窗口的每个后代窗口的指针 pcb 指定的回调函数。参数 pData 将传递给用户函数，并可用于指向用户定义的数据。

回调函数的原型

```
void CallbackFunction(WM_HWIN hWin, void * pData);
```

示例

下例显示了如何使用该函数。它创建了 3 个窗口，第一个作为桌面的子窗口，第二个作为第一个窗口的子窗口，第三个作为第二个窗口的子窗口。窗口创建后，使用 WM_ForEachDesc() 在其父窗口中移动各个窗口：


```
#include "GUI.h"
#include "WM.h"

/*********************************************************************
*
*       _cbWin
*/
static void _cbWin(WM_MESSAGE * pMsg) {
  GUI_COLOR Color;
  switch (pMsg->MsgId) {
  case WM_PAINT:
    WM_GetUserData(pMsg->hWin, &Color, 4);
    GUI_SetBkColor(Color);
    GUI_Clear();
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

/*********************************************************************
*
*       _cbDoSomething
*/
static void _cbDoSomething(WM_HWIN hWin, void * p) {
  int Value = *(int *)p;
  WM_MoveWindow(hWin, Value, Value);
}

/*********************************************************************
*
*       MainTask
*/
void MainTask(void) {
  WM_HWIN hWin_1, hWin_2, hWin_3;
  int Value = 10;
  GUI_COLOR aColor[] = {GUI_RED, GUI_GREEN, GUI_BLUE};
  GUI_Init();
  WM_SetDesktopColor(GUI_BLACK);
  hWin_1 = WM_CreateWindow( 10, 10, 100, 100, WM_CF_SHOW, _cbWin, 4);
  hWin_2 = WM_CreateWindowAsChild(10, 10, 80, 80, hWin_1, WM_CF_SHOW, _cbWin, 4);
  hWin_3 = WM_CreateWindowAsChild(10, 10, 60, 60, hWin_2, WM_CF_SHOW, _cbWin, 4);
  WM_SetUserData(hWin_1, &aColor[0], 4);
  WM_SetUserData(hWin_2, &aColor[1], 4);
  WM_SetUserData(hWin_3, &aColor[2], 4);
  while (1) {
    WM_ForEachDesc(WM_HBKWIN, _cbDoSomething, (void *)&Value);
    Value *= -1;
    GUI_Delay(500);
  }
}
```

WM_GetCallback()

描述

返回给定窗口的回调函数的指针

原型

```
WM_CALLBACK * WM_GetCallback(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

指向给定窗口的回调函数的类型 WM_CALLBACK 的指针。如果窗口没有回调函数，则返回 NULL。

WM_GetActiveWindow()

描述

返回用于绘制操作的活动窗口的句柄。

原型

```
WM_HWIN WM_GetActiveWindow(void);
```

返回值

活动窗口的句柄。

WM_GetClientRect()

描述

返回活动窗口的客户区在窗口坐标中的坐标。也就是说，GUI_RECT 结构的 x0 和 y0 将为 0，x1 和 y1 对应于尺寸 -1。

原型

```
void WM_GetClientRect(GUI_RECT* pRect);
```

参数	描述
pRect	GUI_RECT 结构的指针。

WM_GetClientRectEx()

描述

返回窗口的客户区在窗口坐标中的坐标。也就是说，GUI_RECT 结构的 x0 和 y0 将为 0，x1 和 y1 对应于尺寸 -1。

原型

```
void WM_GetClientRectEx(WM_HWIN hWin, GUI_RECT* pRect);
```

参数	描述
hWin	窗口句柄。
pRect	GUI_RECT 结构的指针。

WM_GetDesktopWindow()

描述

返回桌面窗口的句柄。

原型

```
WM_HWIN WM_GetDesktopWindow(void);
```

返回值

桌面窗口的句柄。

其他信息

桌面窗口始终是最底层的窗口，任何进一步创建的窗口都是其后代。

WM_GetDesktopWindowEx()

描述

在多层环境中工作时，返回指定桌面窗口的句柄。

原型

```
WM_HWIN WM_GetDesktopWindowEx(unsigned int LayerIndex);
```

参数	描述
LayerIndex	层的索引

返回值

指定桌面窗口的句柄。

WM_GetDialogItem()

描述

返回对话框项目 （小工具）的窗口句柄。

原型

```
WM_HWIN WM_GetDialogItem(WM_HWIN hDialog, int Id);
```

参数	描述
hDialog	对话框的句柄。
Id	小工具的窗口 Id。

返回值

小工具的窗口句柄。

其他信息

创建对话框时始终使用此函数，因为对话框中使用的小工具的窗口 Id 必须转换成其句柄，然后才能使用。

WM_GetFirstChild()

描述

返回指定窗口的第一个子窗口的句柄。

原型

```
void WM_GetFirstChild(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

窗口的第一个子窗口的句柄；如果没有子窗口，则为 0。

其他信息

窗口的第一个子窗口是该特定父窗口首先创建的子窗口。如果各窗口的 Z 位置都没有改变，则它就是在指定父窗口直接顶部的窗口。

WM_GetFocussedWindow()

描述

返回具有输入焦点的窗口的句柄。

原型

```
WM_HWIN WM_GetFocussedWindow(void);
```

返回值

有输入焦点的窗口的句柄，如果任何窗口都没有输入焦点，则为 0。

WM_GetHasTrans()

描述

返回有透明性标记的当前值。

原型

```
U8 WM_GetHasTrans(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

0：不透明
1：窗口有透明性

其他信息

在设置时，此标记告诉窗口管理器窗口包含不重绘的部分，因此将为透明。从而使 WM 知道需要在重绘窗口之前重绘背景，以确保透明部分正确还原。

WM_GetInvalidRect()

描述

返回窗口在桌面坐标中的无效矩形。

原型

```
int WM_GetInvalidRect(WM_HWIN hWin, GUI_RECT * pRect);
```

参数	描述
hWin	窗口句柄。
pRect	用于存储无效矩形的 GUI_RECT 结构的指针。

返回值

没有无效矩形时为 0，否则为 1。

WM_GetNextSibling()

描述

返回指定页面的窗口句柄。

原型

```
void WM_GetNextSibling(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

窗口的下一同属窗口的句柄；如果没有则为 0。

其他信息

窗口的下一同属窗口是相对于同一父窗口创建的下一个子窗口。如果各窗口的 Z 位置都没有改变，则它就是在指定窗口直接顶部的窗口。

WM_GetOrgX(), WM_GetOrgY()

描述

分别返回活动窗口的原点在桌面坐标中的 X 或 Y 位置。

原型

```
int WM_GetOrgX(void);
int WM_GetOrgY(void);
```

返回值

活动窗口的原点在桌面坐标中的 X 或 Y 位置。

WM_GetParent()

描述

返回指定窗口的父窗口的句柄。

原型

```
void WM_GetParent(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

窗口的父窗口的句柄；如果没有则为 0。

其他信息

仅在桌面窗口的句柄用作参数时，才不存在父窗口。

WM_GetPrevSibling()

描述

返回指定窗口的前一同属窗口的句柄。

原型

```
void WM_GetPrevSibling(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

窗口的前一同属窗口的句柄；如果没有则为 0。

其他信息

窗口的前一同属窗口是相对于同一父窗口创建的前一个子窗口。如果各窗口的 Z 位置都没有改变，则它就是在指定窗口直接下面的窗口。

WM_GetStayOnTop()

描述

返回保持在顶部标记的当前值。

原型

```
int WM_GetStayOnTop(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

0：保持在顶部标记未设置
1：保持在顶部标记已设置

WM_GetUserData()

描述

检索通过 WM_SetUserData() 所设置的数据。

原型

```
int WM_GetUserData(WM_HWIN hWin, void* pDest, int SizeofBuffer);
```

参数	描述
hWin	窗口句柄。
pDest	指向缓冲区的指针。
SizeOfBuffer	缓冲区的大小。

返回值

检索到的字节数。

其他信息

此函数返回的最大字节数是在创建窗口时指定的额外字节数。

WM_GetWindowOrgX(), WM_GetWindowOrgY()

描述

分别返回指定窗口的原点在桌面坐标中的 X 或 Y 位置。

原型

```
int WM_GetWindowOrgX(WM_HWIN hWin);
int WM_GetWindowOrgY(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

客户区的 X 或 Y 位置 （单位：像素）。

WM_GetWindowRect()

描述

返回活动窗口在桌面坐标中的坐标。

原型

```
void WM_GetWindowRect(GUI_RECT* pRect);
```

参数	描述
pRect	GUI_RECT 结构的指针。

WM_GetWindowRectEx()

描述

返回窗口在桌面坐标中的坐标。

原型

```
void WM_GetWindowRectEx(WM_HWIN hWin, GUI_RECT* pRect);
```

参数	描述
hWin	窗口句柄。
pRect	GUI_RECT 结构的指针。

其他信息

如果给定窗口的句柄为 0，或 GUI_RECT 结构的给定指针为 NULL，则此函数立即返回。

WM_GetWindowSizeX(), WM_GetWindowSizeY()

描述

分别返回指定窗口的 X 或 Y 尺寸。

原型

```
int WM_GetWindowSizeX(WM_HWIN hWin);
int WM_GetWindowSizeY(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

窗口的 X 或 Y 尺寸 （单位：像素）。
在水平方向定义为 x1-x0+1，在垂直方向定义为 y1-y0+1，其中 x0、x1、y0、y1 是窗口的最左 / 最右 / 最上 / 最下位置。
如果给定窗口的句柄为 0，则函数返回桌面窗口的尺寸。

WM_HasCaptured()

描述

给定窗口捕获到鼠标和触摸屏输入时返回 1，否则返回 0。

原型

```
int WM_HasCaptured(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

给定窗口捕获到鼠标和触摸屏输入时为 1，否则为 0。

其他信息

如果给定窗口句柄无效或为 0，则函数返回错误结果。

WM_HasFocus()

描述

检查给定窗口是否具有输入焦点。

原型

```
int WM_HasFocus(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

如果给定窗口具有输入焦点则为 1，否则为 0。

其他信息

如果给定窗口句柄无效或为 0，则函数返回错误结果。

WM_HideWindow()

描述

使指定窗口不可见。

原型

```
void WM_HideWindow(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

调用此函数后，窗口不会立即 “不可见”。在执行 WM_Exec() 时，其它窗口的无效区域（出现在要隐藏窗口 “后面”的区域）将重绘。如果您需要立即隐藏 （移走）某窗口，则应调用 WM_Paint() 重绘其它窗口。

WM_InvalidateArea()

描述

使显示器的指定矩形区域无效。

原型

```
void WM_InvalidateArea(GUI_RECT* pRect);
```

参数	描述
pRect	指向带有桌面坐标的 GUI_RECT 结构的指针。

其他信息

调用此函数将告诉 WM 指定区域未更新。
此函数可用于使重叠交叉区域的任何窗口或窗口的任何部分无效。 GUI_RECT 结构的坐标必须使用桌面坐标。

WM_InvalidateRect()

描述

使窗口的指定矩形区域无效。

原型

```
void WM_InvalidateRect(WM_HWIN hWin, GUI_RECT* pRect);
```

参数	描述
hWin	窗口句柄。
pRect	指向带父窗口的窗口坐标的 GUI_RECT 结构的指针。

其他信息

调用此函数将告诉 WM 指定区域未更新。
下一次调用 WM_Paint() 重绘窗口时，该区域也将被重绘。GUI_RECT 结构的坐标必须使用窗口坐标。

WM_InvalidateWindow()

描述

使指定窗口无效。

原型

```
void WM_InvalidateWindow(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

调用此函数会告诉 WM 指定的窗口未更新。

WM_IsCompletelyCovered()

描述

检查给定窗口是否完全被覆盖。

原型

```
char WM_IsCompletelyCovered(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

如果给定窗口完全被覆盖则为 1，否则为 0。

其他信息

如果给定窗口句柄无效或为 0，则函数返回错误结果。

WM_IsCompletelyVisible()

描述

检查给定窗口是否完全可见。

原型

```
char WM_IsCompletelyVisible(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值
如果给定窗口完全可见则为 1，否则为 0。

其他信息
如果给定窗口句柄无效或为 0，则函数返回错误结果。

WM_IsEnabled()

描述
此函数返回表示窗口是否启用的值。

原型
```
int WM_IsEnabled(WM_HWIN hObj);
```

参数	描述
hObj	窗口的句柄。

返回值
如果窗口启用则为 1，否则为 0。

其他信息
禁用的小工具通常显示为灰色，且不接受来自用户的输入。当然，实际外观可能会有所不同（取决于 widget/configuration settings 等）。

WM_IsVisible()

描述
确定指定窗口是否可见。

原型
```
int WM_IsVisible(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值
0: 窗口不可见
1: 窗口可见

WM_IsWindow()

描述
确定指定句柄是否是有效的窗口句柄。

原型
```
void WM_IsWindow(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值
0: 句柄不是有效的窗口句柄
1: 句柄是有效的窗口句柄

其他信息

仅在绝对必要时才使用此函数。如果给定句柄是一个窗口，则存在的窗口越多，用于评估的时间越长。

WM_MakeModal()

描述

此函数使窗口在“模态”模式下作业。这意味着指针设备输入将仅发送到“模态”窗口，或者如果输入位置在模态窗口的矩形内则仅发送到其子窗口。

原型

```
void WM_MakeModal(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

WM_MoveChildTo()

描述

将指定窗口移动到某个位置。

原型

```
void WM_MoveChildTo(WM_HWIN hWin, int x, int y);
```

参数	描述
hWin	窗口句柄。
x	父窗口的窗口坐标中新的 X 位置。
y	父窗口的窗口坐标中新的 Y 位置。

WM_MoveTo()

描述

将指定窗口移动到某个位置。

原型

```
void WM_MoveTo(WM_HWIN hWin, int x, int y);
```

参数	描述
hWin	窗口句柄。
x	桌面坐标中新的 X 位置。
y	桌面坐标中新的 Y 位置。

WM_MoveWindow()

描述

将指定窗口移动某段距离。

原型

```
void WM_MoveWindow(WM_HWIN hWin, int dx, int dy);
```

参数	描述
hWin	窗口句柄。
dx	要移动的水平距离。
dy	要移动的垂直距离。

WM_NotifyParent()

描述

将 WM_NOTIFY_PARENT 消息发送到给定窗口。

原型

```
void WM_NotifyParent(WM_HWIN hWin, int Notification);
```

参数	描述
hWin	窗口句柄。
Notification	要发送到父窗口的值。

其他信息

Notification- 参数将在消息的 Data.v 元素中发送。宏 WM_NOTIFOCATION_USER 可用于定义应用程序定义的消息：

```
#define NOTIFICATION_1 (WM_NOTIFOCATION_USER + 0)
#define NOTIFICATION_2 (WM_NOTIFOCATION_USER + 1)
```

WM_Paint()

描述

立即绘制或重绘指定窗口。

原型

```
void WM_Paint(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

窗口的重绘将反映自上次绘制以来到当前所做的所有更新。

WM_PaintWindowAndDescs()

描述

给指定窗口及其所有子代窗口着色。

原型

```
void WM_PaintWindowAndDescs(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

此函数在绘制整个窗口区域时，先将其变为无效，然后再绘制。

WM_ReleaseCapture()

描述

释放捕获的鼠标和触摸屏输入。

原型

```
void WM_ReleaseCapture(void);
```


其他信息

使用 WM_SetCapture() 可将所有的鼠标和触摸屏输入发送到特定窗口。

WM_ResizeWindow()

描述

通过增加 （或减少）给定差别更改指定窗口的尺寸。

原型

```
void WM_ResizeWindow(WM_HWIN hWin, int XSize, int YSize);
```

参数	描述
hWin	窗口句柄。
dx	X 的变化量。
dy	Y 的变化量。

WM_SelectWindow()

描述

设置要用于绘制操作的活动窗口。

原型

```
WM_HWIN WM_SelectWindow(WM_HWIN  hWin);
```

参数	描述
hWin	窗口句柄。

返回值

选定的窗口。

其他信息

此函数不应在窗口管理器调用的着色函数内调用。如果窗口管理器发送 WM_PAINT 消息，则目标窗口已选择。
在使用多层配置时，该函数还切换到给定窗口的顶层父窗口的层上。
如果给定窗口句柄为 0，则该函数选择最先创建的窗口，通常为第一个桌面窗口。

示例

将句柄为 hWin2 的窗口设置为活动窗口，设置背景色，然后清除该窗口：

```
WM_SelectWindow(hWin2);
GUI_SetBkColor(0xFF00);
GUI_Clear();
```

WM_SendMessage()

描述

将消息发送到指定窗口。

原型

```
void WM_SendMessage(WM_HWIN hWin, WM_MESSAGE* pMsg)
```

参数	描述
hWin	窗口句柄。
pMsg	消息指针。

WM_SendMessageNoPara()

描述

将不带参数的消息发送到指定窗口。

原型

```
void WM_SendMessageNoPara(WM_HWIN hWin, int MsgId)
```

参数	描述
hWin	窗口句柄。
MsgId	要发送消息的 Id。

其他信息

如果仅消息 Id 要发送到窗口，则应使用此函数完成，因为其不需要指向 WM_MESSAGE 结构的指针。注意，接收窗口除了消息 Id 外未获取任何其它信息。

WM_SendToParent()

描述

将给定消息发送到给定窗口的父窗口。

原型

```
void WM_SendToParent(WM_HWIN hWin, WM_MESSAGE* pMsg);
```

参数	描述
hWin	窗口句柄。
pMsg	指向 WM_MESSAGE 结构的指针。

WM_SetDesktopColor()

描述

设置桌面窗口的颜色。

原型

```
GUI_COLOR WM_SetDesktopColor(GUI_COLOR Color);
```

参数	描述
Color	桌面窗口的颜色， 24 位 RGB 值。

返回值

之前选择的桌面窗口颜色。

其他信息

桌面窗口的默认设置不用于自身重新着色。如果不调用此函数，则桌面窗口不进行重绘；因此其它窗口将保持可见，即使在将它们删除之后。
一旦使用此函数指定了颜色，则桌面窗口将进行自身重新着色。为了恢复默认设置，请调用此函数并指定 GUI_INVALID_COLOR。

WM_SetDesktopColorEx()

描述

在多层环境下设置桌面窗口的颜色。

原型

```
GUI_COLOR WM_SetDesktopColorEx(GUI_COLOR Color, unsigned int LayerIndex);
```

参数	描述
Color	桌面窗口的颜色， 24 位 RGB 值。
LayerIndex	层的索引。

返回值

之前选择的桌面窗口颜色。

其他信息

（参见 WM_SetDesktopColor）。

WM_SetCallback()

描述

设置窗口管理器要执行的回调例程。

原型

```
WM_CALLBACK* WM_SetCallback (WM_HWIN hWin, WM_CALLBACK* cb)
```

参数	描述
hWin	窗口句柄。
cb	指向回调例程的指针。

返回值

指向上一回调例程的指针。

其他信息

给定窗口将变为无效，这确保窗口可进行重绘。

WM_SetCapture()

描述

将所有鼠标和触摸屏消息发送到给定窗口。

原型

```
void WM_SetCapture(WM_HWIN hObj, int AutoRelease);
```

参数	描述
hWin	窗口句柄。
AutoRelease	如果捕获操作应在用户释放触摸时结束，则为 1。

WM_SetCreateFlags()

描述

设置在创建新窗口时用作默认的标记。

原型

```
U8 WM_SetCreateFlags (U8 Flags)
```

参数	描述
Flags	窗口创建标识（请参阅 WM_CreateWindow()）。

返回值

此参数以前的值。

其他信息

此处指定的标记为二进制 ORed，标记在 WM_CreateWindow() 和 WM_CreateWindowAsChild() 例程中指定。
标记 WM_CF_MEMDEV 频繁用于启用所有窗口的存储设备。请注意，在调用 GUI_Init() 之前允许设置创建标记，这将导致背景窗口也受到创建标记的影响。

示例

```
WM_SetCreateFlags(WM_CF_MEMDEV);  /* Auto. use memory devices on all windows */
```

WM_SetFocus()

描述

将输入焦点设置到指定窗口。

原型

```
void WM_SetFocus(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

如果窗口接受焦点则为 0；如果不能接受则为 0 以外的值。

其他信息

窗口接收为其指定输入焦点的 WM_SETFOCUS 消息。如果由于某原因窗口无法接受焦点，则不会发生任何操作。

WM_SetHasTrans()

描述

设置有透明性 标记（将其设置为 1）。

原型

```
void WM_SetHasTrans(WM_HWIN hWin);
```

参数	描述
HWin	窗口句柄。

其他信息

在设置时，此标记告诉窗口管理器窗口包含不重绘的部分，因此将为透明。从而使 WM 知道需要在重绘窗口之前重绘背景，以确保透明部分正确还原。

WM_SetId()

描述

此函数将 WM_SET_ID 消息发送到给定窗口。

原型

```
void WM_SetId(WM_HWIN hObj, int Id);
```

参数	描述
hObj	窗口句柄。
Id	要发送到窗口的 Id。

其他信息

此函数可用于更改小工具的 Id，对每个小工具都有效。当此函数用于应用程序定义的窗口时，窗口的回调函数应处理消息，否则会将其忽略。

WM_SetpfPollPID()

描述

设置将由窗口管理器调用的函数，以轮询指针输入设备 （触摸屏或鼠标）。

原型

```
WM_tfPollPID * WM_SetpfPollPID(WM_tfPollPID * pf);
```

参数	描述
pf	指向 WM_tfPollPID 类型函数的指针。

其他信息

该函数类型定义如下：

```
typedef void WM_tfPollPID(void);
```

示例

将触摸屏作为设备处理的示例：

```
void ReadTouch(void) {
  // ...read touchscreen
}

WM_SetpfPollPID(ReadTouch);
```

WM_SetSize()

描述

设置窗口的新尺寸。

原型

```
void WM_SetSize(WM_HWIN hWin, int XSize, int YSize);
```

参数	描述
hWin	窗口句柄。
XSize	X 的新尺寸。
YSize	Y 的新尺寸。

WM_SetWindowPos()

描述

设置窗口的尺寸和位置。

原型

```
void WM_SetWindowPos(WM_HWIN hWin,
                     int xPos, int yPos,
                     int xSize, int ySize);
```

参数	描述
hWin	窗口句柄。
xPos	桌面坐标中 X 的新位置。
yPos	桌面坐标中 Y 的新位置。
xSize	X 的新尺寸。
ySize	Y 的新尺寸。

WM_SetXSize()

描述

设置窗口的新 X 尺寸。

原型

```
void WM_SetXSize(WM_HWIN hWin, int XSize);
```

参数	描述
hWin	窗口句柄。
XSize	X 的新尺寸。

WM_SetYSize()

描述

设置窗口的新 Y 尺寸。

原型

```
void WM_SetYSize(WM_HWIN hWin, int YSize);
```

参数	描述
hWin	窗口句柄。
YSize	Y 的新尺寸。

WM_SetStayOnTop()

描述

设置保持在顶部标记。

原型

```
void WM_SetStayOnTop(WM_HWIN hWin, int OnOff);
```

参数	描述
hWin	窗口句柄。
OnOff	（参见下表）

参数 OnOff 的允许值	
0	保持在顶部标记将被清除。
1	保持在顶部标记将被设置。

WM_SetTransState()

描述

此函数设置或清除给定窗口的标记 WM_CF_HASTRANS 和 WM_CF_CONST_OUTLINE。

原型

```
void WM_SetTransState(WM_HWIN hWin, unsigned State);
```

参数	描述
hWin	窗口句柄。
State	标记 WM_CF_HASTRANS 和 WM_CF_CONST_OUTLINE 的组合。

其他信息

有关标记 WM_CF_CONST_OUTLINE 的详细信息，请参见“WM_CreateWindow()”（第 309 页）。

WM_SetUserClipRect()

描述

临时将当前窗口的裁剪区缩小为指定矩形。

原型

```
const GUI_RECT* WM_SetUserClipRect(const GUI_RECT* pRect);
```

参数	描述
pRect	指向 GUI_RECT 结构的指针，该结构采用桌面坐标定义裁剪区。

返回值

指向前一裁剪矩形的指针。

其他信息

可传递 NULL 指针，以恢复默认设置。裁剪矩形将在使用回调时由 WM 自动重置。
指定的矩形必须与当前窗口相关，不能将裁剪矩形扩大到超出当前窗口矩形。
应用程序必须确保指定的矩形保留其值直到不再需要它，即，直到指定了其它裁剪矩形或直到传递了 NULL 指针为止。这意味着如果裁剪矩形在返回操作之前始终保持活动状态，则作为参数传递的矩形结构不应为自动变量 （通常位于堆栈上）。此时，应使用静态变量。

示例

此示例取自进度指示器的绘制例程。进度指示器必须在进度条顶部写入文本，且进度条左侧和右侧部分的文本颜色必须不同。这意味着一半数字采用某种颜色，而另一半数字采用其它颜色。最佳的做法是在绘制进度条每一部分时临时减小裁剪区，如下所示：

```
/* Draw left part of the bar */
    r.x0=0; r.x1=x1-1; r.y0=0; r.y1 = GUI_YMAX;
    WM_SetUserClipRect(&r);
    GUI_SetBkColor(pThis->ColorBar[0]);
    GUI_SetColor(pThis->ColorText[0]);
    GUI_Clear();
    GUI_GotoXY(xText,yText); GUI_DispDecMin(pThis->v); GUI_DispChar('%');
/* Draw right part of the bar */
    r.x0=r.x1; r.x1=GUI_XMAX;
    WM_SetUserClipRect(&r);
    GUI_SetBkColor(pThis->ColorBar[1]);
    GUI_SetColor(pThis->ColorText[1]);
    GUI_Clear();
    GUI_GotoXY(xText,yText); GUI_DispDecMin(pThis->v); GUI_DispChar('%');
```

进度条的屏幕截图

WM_SetUserData()

描述

设置窗口的额外数据。建立窗口时，用 NumExtraBytes 参数保留额外数据存储。

原型

```
int WM_SetUserData(WM_HWIN hWin, void * pSrc, int NumBytes);
```

参数	描述
hWin	窗口句柄。
pSrc	指向缓冲区的指针。
NumBytes	缓冲区的大小。

返回值

写入的字节数。

其他信息

用于存储用户数据的字节数最多不能超过建立窗口时指定的 ExtraBytes 数。

WM_ShowWindow()

描述

使指定的窗口可见。

原型

```
void WM_ShowWindow(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

调用此函数后，窗口不会立即显示。它会在执行 WM_Exec() 命令时重绘。如果需要立即显示窗口，应该调用 WM_Paint() 函数。

WM_Update()

描述

立即绘制指定窗口的无效部分。

原型

```
void WM_Update(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

更新窗口后，其整个区域都被标记为有效。

WM_UpdateWindowAndDescs()

描述

对给定窗口的无效部分及其所有后代窗口的无效部分着色。

原型

```
void WM_UpdateWindowAndDescs(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

该函数只绘制无效的窗口区域。

WM_ValidateRect()

描述

使窗口的指定矩形区域有效。

原型

```
void WM_ValidateRect  (WM_HWIN hWin, GUI_RECT* pRect);
```

参数	描述
hWin	窗口句柄。
pRect	指向带父窗口的窗口坐标的 GUI_RECT 结构的指针。

其他信息

调用此函数将会通知 WM 指定窗口已更新。
通常，此函数由内部调用，不需要通过用户应用程序调用。 GUI_RECT 结构的坐标必须使用桌面坐标。

WM_ValidateWindow()

描述

使指定的窗口有效。

原型

```
void WM_ValidateWindow  (WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

其他信息

调用此函数将会通知 WM 指定窗口已更新。
通常，此函数由内部调用，不需要通过用户应用程序调用。

14.7 存储设备支持 （可选）

使用存储设备来重绘窗口时，所有绘制操作都会自动传送到存储设备环境中并在存储器中执行。仅当所有绘制操作都执行完毕后，窗口才会重绘在 LCD 上，并立即反映出所有更新。使用存储设备的优点是可以消除所有的闪变效应 （通常发生在执行绘制操作的同时屏幕持续更新时）。
更多关于存储设备的信息，请参见 “存储设备”（第 247 页）。

WM_DisableMemdev()

描述

禁止使用存储设备来重绘窗口。

原型

```
void WM_DisableMemdev (WM_HWIN hWin)
```

参数	描述
hWin	窗口句柄。

WM_EnableMemdev()

描述

启用使用存储设备来重绘窗口。

原型

```
void WM_EnableMemdev (WM_HWIN hWin)
```

参数	描述
hWin	窗口句柄。

14.8 定时器相关函数

WM_CreateTimer()

描述

创建定时器，其功能是经过指定周期后，向指定窗口发送消息。该定时器与指定窗口相关联。

原型

```
WM_HTIMER WM_CreateTimer(WM_HWIN hWin, int UserId, int Period, int Mode);
```

参数	描述
hWin	接受信息的窗口的句柄。
UserId	用户定义的 Id。如果不对同一窗口使用多个定时器，此值可以设置为零。
Period	周期，此周期过后指定窗口应收到消息。
Mode	（留待将来使用，应为 0）

返回值

定时器的句柄。

其他信息

该函数建立了一个 “单次定时器”，向指定窗口发送 WM_TIMER 消息。该定时器的周期过后，定时器对象仍然有效，可使用WM_RestartTimer()函数重启，或者使用WM_DeleteTimer()函数删除它。请注意，窗口管理器在删除窗口时会自动连带删除与其相关的定时器。

示例

```
static void _cbWin(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case WM_TIMER:
    /*
      ... do something ...
    */
    WM_RestartTimer(pMsg->Data.v, 1000);
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

static void _DemoTimer(void) {
  WM_HWIN hWin;
  WM_HTIMER hTimer;
  hWin   = WM_CreateWindow(10, 10, 100, 100, WM_CF_SHOW, _cbWin, 0);
  hTimer = WM_CreateTimer(hWin, 0, 1000, 0);
  while (1) {
    GUI_Exec();
  }
}
```

WM_DeleteTimer()

描述

删除指定定时器。

原型

```
void WM_DeleteTimer(WM_HTIMER hTimer);
```

参数	描述
hTimer	要删除的定时器的句柄。

其他信息

当定时器的周期过后，定时器对象仍然有效，不会被自动删除。如果以后不需要它了，应该用此函数删除它。
请注意，窗口管理器在删除窗口时会自动删除定时器。

WM_GetTimerId()

描述

获取给定定时器的 Id。

原型

```
int WM_GetTimerId(WM_HTIMER hTimer);
```

参数	描述
hTimer	要删除的定时器的句柄。

返回值

定时器的 Id，定时器是之前通过 WM_CreateTimer() 函数设置的。

WM_RestartTimer()

描述

重启具有指定周期的定时器。

原型

```
void WM_RestartTimer(WM_HTIMER hTimer, int Period);
```

参数	描述
hTimer	要重启的定时器的句柄。
Period	要使用的新周期。

其他信息

指定周期过后将会向分配给该定时器的窗口发送一条 WM_TIMER 消息。更多详细信息，请参阅“WM_CreateTimer()”（第 336 页）。

14.9 小工具相关函数

WM_GetClientWindow()

描述

返回客户端窗口的句柄。此函数向活动窗口发送一条消息，以检索客户端窗口的句柄。如果该窗口不处理该消息，将返回当前窗口的句柄。

原型

```
WM_HWIN WM_GetClientWindow(WM_HWIN hObj);
```

参数	描述
hWin	小工具的句柄。

返回值

客户端窗口的句柄。

其他信息

使用此函数可以检索 FRAMEWIN 小工具的客户端窗口句柄。

WM_GetId()

描述

返回指定小工具窗口的 ID。

原型

```
int WM_GetId(WM_HWIN hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

创建时指定的小工具 ID。
如果指定窗口不是小工具，返回 0。

WM_GetInsideRect()

描述

返回客户区的坐标，该区域由活动小工具尺寸减去边界尺寸确定。此函数向活动窗口发送一条消息，检索内部矩形。如果小工具不处理此消息（也即意味着小工具没有边界），则需使用 WM_GetClientRect 函数计算出矩形。结果通过窗口坐标给出。也即， GUI_RECT 结构中的 x0 和 y0 相当于 x 和 y 的边界尺寸， x1 和 y1 相当于窗口尺寸减去边界尺寸 -1。

原型

```
void WM_GetInsideRect(GUI_RECT* pRect);
```

参数	描述
pRect	GUI_RECT 结构的指针。

WM_GetInsideRectEx()

描述

返回窗口尺寸减去边界尺寸后的坐标。更多详细信息，请参阅“WM_GetInsideRect()”（第 339 页）。

原型

```
void WM_GetInsideRectEx(WM_HWIN hObj, GUI_RECT* pRect);
```

参数	描述
hObj	小工具的句柄。
pRect	GUI_RECT 结构的指针。

WM_GetScrollPosH()

描述

如果窗口附加有水平滚动条，则此函数将返回其滚动位置。

原型

```
int WM_GetScrollPosH(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

附加到窗口的水平滚动条的滚动位置。

其他信息

如果窗口没有附加水平滚动条，则此函数返回。
更多详细信息，请参阅 “SCROLLBAR：滚动条小工具”（第 617 页）。

WM_GetScrollPosV()

描述

如果窗口附加有垂直滚动条，则此函数将返回其滚动位置。

原型

```
int WM_GetScrollPosV(WM_HWIN hWin);
```

参数	描述
hWin	窗口句柄。

返回值

附加到窗口的垂直滚动条的滚动位置。

其他信息

如果窗口没有附加垂直滚动条，则此函数返回 0。
更多详细信息，请参阅 “SCROLLBAR：滚动条小工具”（第 617 页）。

WM_GetScrollState()

描述

使用指定滚动条小工具窗口的当前状态信息填充数据结构。

原型

```
void WM_GetScrollState(WM_HWIN hObj, WM_SCROLL_STATE* pScrollState);
```

参数	描述
hObj	滚动条小工具的句柄。
pScrollState	指向 WM_SCROLL_STATE 类型的数据结构的指针。

其他信息

由于滚动条的状态是由多个值定义的，所以此函数不返回值。
它对其他类型的小工具和窗口没有影响。
更多详细信息，请参阅 “SCROLLBAR：滚动条小工具”（第 617 页）。

WM_SCROLL_STATE 的元素

数据类型	元素	描述
int	NumItems	项目数量。
int	v	当前值。
int	PageSize	一页中可见项目的数量。

WM_SetScrollPosH()

描述

如果窗口附加有水平滚动条，则此函数将设置其滚动位置。

原型

```
void WM_SetScrollPosH(WM_HWIN hWin, unsigned ScrollPos);
```

参数	描述
hWin	窗口句柄。
ScrollPos	滚动条的新滚动位置。

其他信息

如果窗口没有附加水平滚动条，则此函数返回。
更多详细信息，请参阅 “SCROLLBAR：滚动条小工具”（第 617 页）。

WM_SetScrollPosV()

描述

如果窗口附加有垂直滚动条，则此函数将设置其滚动位置。

原型

```
void WM_SetScrollPosV(WM_HWIN hWin, unsigned ScrollPos);
```

参数	描述
hWin	窗口句柄。
ScrollPos	滚动条的新滚动位置。

其他信息

如果窗口没有附加垂直滚动条，则此函数返回。
更多详细信息，请参阅 “SCROLLBAR：滚动条小工具”（第 617 页）。

WM_SetScrollState()

描述

设置指定滚动条小工具的状态。

原型

```
void WM_SetScrollState(WM_HWIN hObj, const WM_SCROLL_STATE* pState);
```

参数	描述
hObj	滚动条小工具的句柄。

14.10 示例

以下示例描述了使用回调例程重绘背景和不使用回调例程的区别，它还描述了如何设置自己的回调函数。该示例以 emWin 随附示例中的 WM_Redraw.c 的形式提供：

```
/*********************************************************************
*                SEGGER MICROCONTROLLER SYSTEME GmbH                 *
*        Solutions for real time microcontroller applications        *
*                                                                    *
*                      emWin example code                            *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File        :WM_Redraw.c
Purpose     :Demonstrates the redrawing mechanism of the window manager
----------------------------------------------------------------------
*/

#include "GUI.H"

/*********************************************************************
*
*             Callback routine for background window
*
**********************************************************************
*/

static void cbBackgroundWin(WM_MESSAGE* pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_Clear();
  default:
    WM_DefaultProc(pMsg);
  }
}

/*********************************************************************
*
*             Callback routine for foreground window
*
**********************************************************************
*/

static void cbForegroundWin(WM_MESSAGE* pMsg) {
  switch (pMsg->MsgId) {
  case WM_PAINT:
    GUI_SetBkColor(GUI_GREEN);
    GUI_Clear();
    GUI_DispString("Foreground window");
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}

/*********************************************************************
```



```
*
*         Demonstrates the redraw mechanism of emWin
*
**********************************************************************
*/

static void DemoRedraw(void) {
  GUI_HWIN hWnd;
  while(1) {
    /* Create foreground window */
    hWnd = WM_CreateWindow(10, 10, 100, 100, WM_CF_SHOW, cbForegroundWin, 0);
    /* Show foreground window */
    GUI_Delay(1000);
    /* Delete foreground window */
    WM_DeleteWindow(hWnd);
    GUI_DispStringAt("Background of window has not been redrawn", 10, 10);
    /* Wait a while, background will not be redrawn */
    GUI_Delay(1000);
    GUI_Clear();
    /* Set callback for Background window */
    WM_SetCallback(WM_HBKWIN, cbBackgroundWin);
    /* Create foreground window */
    hWnd = WM_CreateWindow(10, 10, 100, 100,WM_CF_SHOW, cbForegroundWin, 0);
    /* Show foreground window */
    GUI_Delay(1000);
    /* Delete foreground window */
    WM_DeleteWindow(hWnd);
    /* Wait a while, background will be redrawn */
    GUI_Delay(1000);
    /* Delete callback for Background window */
    WM_SetCallback(WM_HBKWIN, 0);
 }}

/*********************************************************************
*
*                 main
*
**********************************************************************
*/

void main(void) {
  GUI_Init();
  DemoRedraw();
}
```


第 15 章

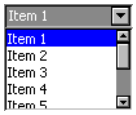
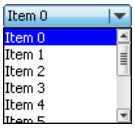
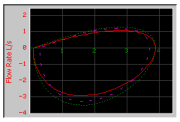
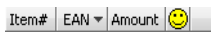
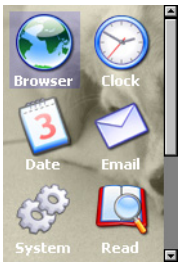
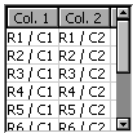
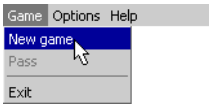
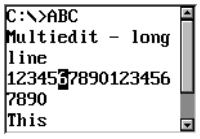
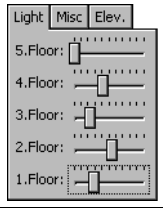
窗口对象 （小工具）

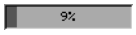
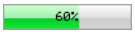
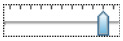
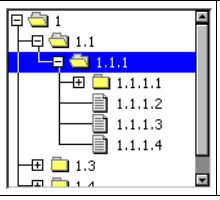
小工具是具有对象类型属性的各种窗口；在窗口界被称为控件，是组成用户界面的元素。它们可自动对某些事件作出反应；例如，按下某按钮后，它可以不同状态显示。小工具需要创建，具有可在存续期间随时更改的属性，并通常在不再需要时被删除。正如窗口一样，小工具通过其创建函数返回的句柄进行引用。
小工具要求使用窗口管理器。小工具创建后，可像其他任何窗口一样处理它； WM 确保了在必要时随时正确显示 （并重绘）它。小工具不是编写应用程序或用户界面所必需的，但它们可大大简化编程。

15.1 基础知识

15.1.1 可用小工具

下表显示了当前可用的各种小工具的外观。其中一些支持皮肤设置。这种更改外观的方法将在章节“皮肤设置”中详细解释。第二个屏幕截图所示为小工具启用了皮肤设置后的外观：

名称	屏幕截图（经典）	屏幕截图（皮肤设置）	描述
BUTTON			可按下的按钮。按钮上可显示文本或位图。
CHECKBOX			可被选中或取消选中的复选框。
DROPDOWN			下拉列表，按下后会打开一个列表框。
EDIT			单行编辑字段，提示用户键入数字或文本。
FRAMEWIN			框架窗口。创建典型的 GUI 外观。
GRAPH			图形小工具，用于显示曲线或测量值。
HEADER			标题控件，用于管理各栏。
ICONVIEW			图标视图小工具。对于基于图标的平台 （如在常见的手持式设备中）非常有用。
LISTBOX			突出显示用户选择项目的列表框。
LISTVIEW			Listview 小工具用于创建表。
LISTWHEEL			Listwheel 小工具。可使用指针输入设备移动或加速数据。
MENU			菜单小工具用于创建水平和垂直菜单。
MULTIEDIT			Multiedit 小工具用于编辑多行文本。
MULTIPAGE			Multipage 小工具用于创建具有多个页面的对话框。

名称	屏幕截图（经典）	屏幕截图（皮肤设置）	描述
PROGBAR			用于可视化的进度条。
RADIOBUTTON			可以选择的单选按钮，一次只能选择一个按钮。
SCROLLBAR			水平或者垂直的滚动条。
SLIDER			用于更改值的滑动条。
TEXT			静态文本控件，一般用于对话框。
TREEVIEW			用于管理分层列表的树形图小工具。

15.1.2 了解重绘机制

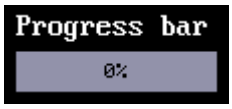
小工具根据其属性绘制自身，此操作在调用 WM_Exec()、GUI_Exec() 或 GUI_Delay() 时执行。在多任务环境中，通常由后台任务来调用 WM_Exec() 并更新小工具（以及所有其他具有回调函数的窗口）。

小工具的属性更改后，该小工具的窗口（或部分窗口）会被标记为无效，但不会立即重绘。因此，代码段的执行速度会非常快。重绘由 WM 在稍后执行，或者通过为该小工具调用 WM_Paint() 强制执行（或直到重绘所有窗口时调用 WM_Exec()）。

15.1.3 如何使用小工具

假设我们要显示一个进度条。所需的只是以下代码：

```
PROGBAR_Handle hProgBar;
GUI_DispStringAt("Progress bar", 100, 20);
hProgBar = PROGBAR_Create(100, 40, 100, 20, WM_CF_SHOW);
```



其中第一行为小工具的句柄保留内存，最后一行实际创建该小工具。稍后或者在某单独任务中调用 WM_Exec() 时，窗口管理器会自动绘制此小工具。

每种类型的小工具都有若干能够修改其外观的成员函数。小工具创建后，可通过调用其成员函数之一更改其属性。这些函数使用小工具的句柄作为第一个自变量。要使上述创建的进度条显示 45% 并将条颜色从默认设置（深灰 / 浅灰）更改为绿色 / 红色，可使用以下代码段：

```
PROGBAR_SetBarColor(hProgBar, 0, GUI_GREEN);
PROGBAR_SetBarColor(hProgBar, 1, GUI_RED);
PROGBAR_SetValue(hProgBar, 45);
```

默认配置

所有小工具都还具有一个或多个配置宏，它们定义各种默认设置（如所使用的字体和颜色）。在本章稍后每个小工具的各自章节中，列出了每个小工具的可用配置选项。

小工具如何通信

小工具通常作为子窗口创建，父窗口可为任何窗口类型，甚至是另一种小工具。为了确保同步，无论何时父窗口的任何子项有任何事件发生，通常都应通知父窗口。有事件发生时，子窗口小工具通过发送 WM_NOTIFY_PARENT 消息与其父窗口通信。作为消息一部分发送的通知代码取决于事件。
大多数小工具都有定义不同类型事件的一个或多个通知代码，每种小工具可用的通知代码 （如果有）在其各自章节中列出。

皮肤设置

如前所述，小工具的外观可使用其各自的成员函数进行修改，其中一些支持皮肤设置。如果对某小工具使用了皮肤设置，则 “皮肤”将决定该小工具的外观，且一些成员函数将失效。有关详细信息，请参阅 “皮肤设置”一章。

小工具的动态内存使用

在嵌入式应用中，由于分裂效应，通常都不太希望使用动态内存。可使用许多不同策略来避免此情况，但只要内存区域被应用程序中的指针引用，这些策略的使用都有局限性。因此， emWin 使用不同的方法：所有对象 （以及运行时存储的所有数据）都存储在被句柄引用的内存区域中。这使它能够在运行时重新分配已分配的内存区域，因此避免了使用指针时会发生的长期分配问题。所以所有小工具都使用句柄引用。

确定小工具的类型

没有像 WM_GetWidgetType() 一样的函数来确定小工具的类型，其类型仅可通过将特定小工具的回调函数与小工具 API 的公共回调函数进行比较来确定。在回调函数未被覆盖时此方法很有效。如下所示为如何确定小工具类型的简短示例。如果回调函数被覆盖，则应相应调整方法：

```
WM_CALLBACK * pCb = WM_GetCallback(hWidget);
if          (pCb == BUTTON_Callback) {
  /* Widget is a button */
} else if (pCb == DROPDOWN_Callback) {
  /* Widget is a dropdown */
} else if (pCb == LISTBOX_Callback) {
  /* Widget is a listbox */
} else if (...) {
  ...
}
```

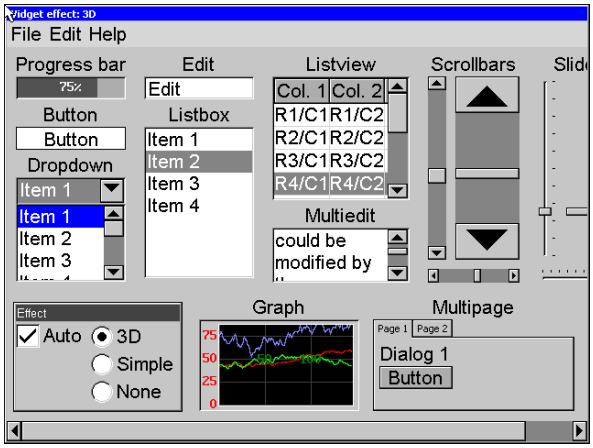

15.2 配置选项

类型	宏	默认值	描述
B	WIDGET_USE_PARENT_EFFECT	0	设置为 1 时，小工具的每个“子小工具”都具有与其父小工具相同的效果。例如，如果列表框需要创建一个滚动条，则新滚动条具有与列表框相同的效果。
B	WIDGET_USE_SCHEME_LARGE	0	设置为 1 时，小工具的默认外观为大尺寸。这意味着所有显示文本的小工具都配置为使用大尺寸的默认字体。
B	WIDGET_USE_SCHEME_MEDIUM	0	设置为 1 时，小工具的默认外观为中等尺寸。这意味着所有显示文本的小工具都配置为使用中等尺寸的默认字体。
B	WIDGET_USE_SCHEME_SMALL	1	设置为 1 时，小工具的默认外观为小尺寸。这意味着所有显示文本的小工具都配置为使用小尺寸的默认字体。
B	WIDGET_USE_FLEX_SKIN	0	设置为 1 时，默认使用 Flex 皮肤来绘制小工具。有关皮肤设置的详细信息，请参阅 “皮肤设置”一章。

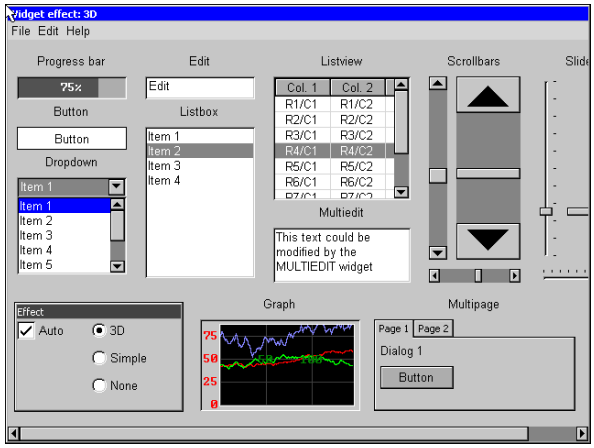
WIDGET_USE_SCHEME...

下表所示为小工具配置的默认外观：

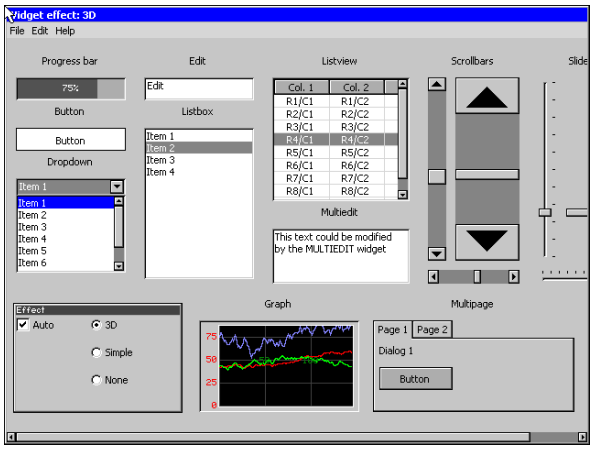
WIDGET_USE_SCHEME_LARGE



WIDGET_USE_SCHEME_MEDIUM



WIDGET_USE_SCHEME_SMALL



15.3 通用小工具 API

15.3.1 用于小工具的 WM 例程

由于小工具本质上就是窗口，所以它们与任何窗口管理器 API 例程都兼容。小工具的句柄用作 hWin 参数，且小工具的处理方式与其他任何窗口都相同。最常用于小工具的 WM 函数如下所列：

例程	描述
WM_DeleteWindow()	删除窗口。
WM_DisableMemdev()	禁止使用存储设备进行重绘。
WM_EnableMemdev()	启用存储设备用于重绘。
WM_InvalidateWindow()	使窗口无效。
WM_Paint()	立即绘制或重绘窗口。

有关 WM 相关函数的完整列表，请参见 “窗口管理器 (WM)”（第 289 页）。

15.3.2 常用例程

下表按字母顺序列出了可用的小工具相关例程。这些函数对于所有小工具都是通用的，在此列出以避免重复。这些例程的详细说明如下。每个小工具可用的额外成员函数在稍后各章节中介绍。

例程	描述
<WIDGET>_Callback()	默认回调函数。
<WIDGET>_CreateIndirect()	用于对话框中的自动创建。
<WIDGET>_CreateUser()	使用额外字节作为用户数据创建小工具。
<WIDGET>_GetUserData()	检索用 <WIDGET>_SetUserData 设置的数据。
<WIDGET>_SetUserData()	设置小工具的额外数据。
WIDGET_GetDefaultEffect()	返回用于小工具的默认效果。
WIDGET_SetDefaultEffect()	设置用于小工具的默认效果。
WIDGET_SetEffect()	设置用于给定小工具的效果。

<WIDGET>_Callback()

描述

要从被覆盖回调函数内部使用的小工具的默认回调函数。

原型

```
void <WIDGET>_Callback(WM_MESSAGE * pMsg);
```

参数	描述
pMsg	指向类型为 WM_MESSAGE 的数据结构的指针。

其他信息

不应直接调用小工具的默认回调函数，仅应从被覆盖回调函数内部使用它。
有关 WM_MESSAGE 数据结构的详细信息，请参阅 “消息”（第 295 页）。

<WIDGET>_CreateIndirect()

描述

创建要在对话框中使用的小工具。

原型

```
<WIDGET>_Handle <WIDGET>_CreateIndirect(
  const GUI_WIDGET_CREATE_INFO * pCreateInfo,
  WM_HWIN                        hParent,
  int                            x0,
  int                            y0,
  WM_CALLBACK                  * cb
);
```

参数	描述
pCreateInfo	指向 GUI_WIDGET_CREATE_INFO 结构的指针 （见下文）。
hParent	父窗口的句柄。
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
cb	指向回调函数的指针。

其他信息

任何小工具都可使用适宜的前缀间接创建。例如：
BUTTON_CreateIndirect() 可间接创建一个按钮小工具，
CHECKBOX_CreateIndirect() 可间接创建一个复选框小工具，等等。

仅在小工具要被包含在对话框中时，才需要间接创建它。否则，可使用 <WIDGET>_Create() 函数直接创建它。有关对话框的更多信息，请参见 “对话框”（第 665 页）。

资源表

GUI_WIDGET_CREATE_INFO 数据结构在对话框资源表中定义如下：

```
typedef struct {
  GUI_WIDGET_CREATE_FUNC * pfCreateIndirect; // Create function
  const char * pName;                        // Text (not used for all widgets)
  I16 Id;                                    // Window ID of the widget
  I16 x0, y0, xSize, ySize;                  // Size and position of the widget
  I16 Flags;                                 // Widget-specific flags (or 0)
  I32 Para;                                  // Widget-specific parameter (or 0)
  U32 NumExtraBytes;                         // Number of extra bytes usable
                                             // with <WIDGET>_SetUserData &
                                             //      <WIDGET>_GetUserData
} GUI_WIDGET_CREATE_INFO;
```

小工具标记和参数是可选的，并随小工具类型不同而不同。每个小工具可用的标记和参数 （如果有）将列在本章后面的各相关章节中。

<WIDGET>_CreateUser()

描述

使用额外字节作为用户数据创建小工具。此函数与相应小工具的 <WIDGET>_CreateEx() 函数在每一方面都相似，除了额外参数 NumExtraBytes 以外。

原型

```
<WIDGET>_Handle <WIDGET>_CreateUser(int x0, int y0, ..., int Id,
```

```
                                    int NumExtraBytes);
```

参数	描述
NumBytes	要分配的额外字节数。

返回值

已创建小工具的句柄；函数失败时为 0。

其他信息

有关其他参数的更多信息，可参阅相应的 <WIDGET>_CreateEx() 函数。

<WIDGET>_GetUserData()

描述

检索用 <WIDGET>_SetUserData 设置的数据。

原型

```
int <WIDGET>_GetUserData(<WIDGET>_Handle   hObj,
                         void            * pDest,
                         int               NumBytes)
```

参数	描述
hObj	小工具的句柄
pDest	指向缓冲区的指针
NumBytes	要读取的字节数

返回值

已读取的字节数。

其他信息

此函数返回的最大字节数是在创建小工具时使用 <WIDGET>_CreateUser() 或 <WIDGET>_CreateIndirect() 指定的额外字节数。

<WIDGET>_SetUserData()

描述

设置小工具的额外数据。

原型

```
int <WIDGET>_GetUser(<WIDGET>_Handle   hObj,
                     void            * pDest,
                     int               NumBytes)
```

参数	描述
hObj	小工具的句柄
pDest	指向缓冲区的指针
NumBytes	要写入的字节数

返回值

写入的字节数。

其他信息

此函数返回的最大字节数是在创建小工具时使用 <WIDGET>_CreateUser() 或 <WIDGET>_CreateIndirect() 指定的额外字节数。

WIDGET_GetDefaultEffect()

描述

返回用于小工具的默认效果。

原型

```
const WIDGET_EFFECT * WIDGET_GetDefaultEffect(void);
```

返回值

此函数的结果是 WIDGET_EFFECT 结构的指针。

其他信息

更多详细信息，请参阅 “WIDGET_SetDefaultEffect()”（第 353 页）。

WIDGET_SetDefaultEffect()

描述

设置用于小工具的默认效果。

原型

```
const WIDGET_EFFECT * WIDGET_SetDefaultEffect(const WIDGET_EFFECT* pEffect);
```

参数	描述
pEffect	WIDGET_EFFECT 结构的指针。参见下表。

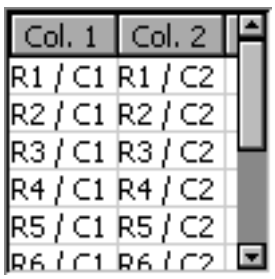
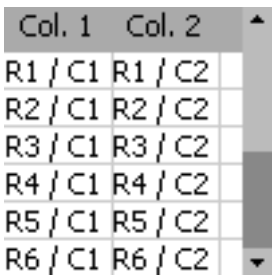
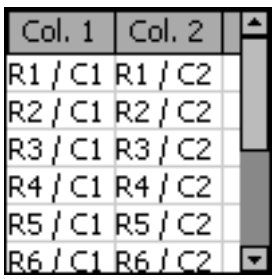
元素 pEffect 的允许值	
&WIDGET_Effect_3D	将默认效果设置为 “3D”。
&WIDGET_Effect_None	将默认效果设置为 “无”。
&WIDGET_Effect_Simple	将默认效果设置为 “简单”。

返回值

先前使用的默认效果。

其他信息

下表所示为使用各种效果后一些小工具的外观：

“3D”	“无”	“简单”
		
		
		

WIDGET_SetEffect()

描述

设置给定小工具的效果。

原型

```
void WIDGET_SetEffect(WM_HWIN hObj, const WIDGET_EFFECT* pEffect);
```

参数	描述
hObj	小工具的句柄。
pEffect	WIDGET_EFFECT结构的指针。更多详细信息，请参阅“WIDGET_SetDefaultEffect()”（第353页）。

15.3.3 用户绘制小工具

一些小工具支持自画功能，例如 LISTBOX 小工具。小工具的用户绘制模式激活后，将调用 WIDGET_DRAW_ITEM_FUNC 类型的应用程序定义函数来绘制该小工具（项目）。应用程序定义的自画函数的原型应按如下定义：

原型

```
int WIDGET_DRAW_ITEM_FUNC(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo)
```

参数	描述
pDrawItemInfo	指向 WIDGET_ITEM_DRAW_INFO 结构的指针。

WIDGET_ITEM_DRAW_INFO 的元素

数据类型	元素	描述
WM_HWIN	hWin	小工具的句柄
int	Cmd	（参见下表）
int	ItemIndex	要绘制项目的以零为基准的索引
int	x0	绘制项目时要使用的在窗口坐标中的 X 位置
int	y0	绘制项目时要使用的在窗口坐标中的 Y 位置

元素 Cmd 的允许值	
WIDGET_ITEM_GET_XSIZE	该函数返回给定项目的 x 尺寸 （宽度），单位为像素。
WIDGET_ITEM_GET_YSIZE	该函数返回给定项目的 y 尺寸 （高度），单位为像素。
WIDGET_ITEM_DRAW	函数在指定位置绘制指定项目。
WIDGET_DRAW_BACKGROUND	要绘制的小工具的背景。
WIDGET_DRAW_OVERLAY	此命令在所有其他绘制操作都完成后发出，目的是可以在小工具之上绘制一些覆盖项目。

返回值

取决于给定命令。

对命令的反应

该函数必须对在WIDGET_ITEM_DRAW_INFO结构中给定的命令作出反应。以下列2种方式之一执行：

- 调用特定小工具提供的适当默认函数 （例如， LISTBOX_OwnerDraw()）
- 提供作出相应反应的代码

命令

该函数支持并应对以下所列命令作出反应。如上所描述，应为所有未处理函数调用默认自画函数。这可减小代码大小（例如，如果高度与默认高度相同），并确保代码与将来的软件新版本和引入的其他命令保持兼容。

WIDGET_ITEM_GET_XSIZE

必须返回给定项目的 X 尺寸，以像素为单位。

WIDGET_ITEM_GET_YSIZE

必须返回给定项目的 Y 尺寸（高度），以像素为单位。

WIDGET_ITEM_DRAW

必须绘制给定项目。 WIDGET_ITEM_DRAW_INFO 结构的 x0 和 y0 指定项目在窗口坐标中的位置。该项目必须填充其整个矩形，该矩形由提供给函数的起始位置 x0、 y0 以及函数反应命令 WIDGET_ITEM_GET_YSIZE、WIDGET_ITEM_GET_XSIZE 返回的尺寸定义。此矩形区域的任何部分都不得留空。它不能绘制到此矩形外，因为在调用该函数之前已设置了裁剪矩形。

15.4 BUTTON：按钮小工具

按钮小工具通常用作触摸屏的主要用户界面元素。如果按钮有输入焦点，它还会对键 GUI_KEY_SPACE 和 GUI_KEY_ENTER 作出反应。按钮可显示文本 （如下所示），或显示位图。

所有 BUTTON 相关例程都位于文件 BUTTON*.c、BUTTON.h 中。所有标识符的前缀都是 BUTTON。

皮肤设置 ...

... 可用于该小工具。上述屏幕截图显示使用默认皮肤的小工具。有关详细信息，请参阅“皮肤设置”一章。

15.4.1 配置选项

类型	宏	默认值	描述
N	BUTTON_3D_MOVE_X	1	处于按下状态的文本 / 位图在水平方向上移动的像素数。
N	BUTTON_3D_MOVE_Y	1	处于按下状态的文本 / 位图在垂直方向上移动的像素数。
N	BUTTON_ALIGN_DEFAULT	GUI_TA_HCENTER | GUI_TA_VCENTER	用于显示按钮文本的对齐方式。
N	BUTTON_BKCOLOR0_DEFAULT	0xAAAAAA	背景色，未按下状态。
N	BUTTON_BKCOLOR1_DEFAULT	GUI_WHITE	背景色，按下状态。
N	BUTTON_FOCUSCOLOR_DEFAULT	GUI_BLACK	显示聚焦框所使用的默认颜色。
S	BUTTON_FONT_DEFAULT	&GUI_Font13_1	按钮文本使用的字体。
B	BUTTON_REACT_ON_LEVEL	0	参见以下描述。
N	BUTTON_TEXTCOLOR0_DEFAULT	GUI_BLACK	文本颜色，未按下状态。
N	BUTTON_TEXTCOLOR1_DEFAULT	GUI_BLACK	文本颜色，按下状态。

BUTTON_REACT_ON_LEVEL

默认情况下，按钮对每一触摸消息作出反应。例如，如果使用指针输入设备 (PID) 不准确地触摸一个按钮，然后将 PID 以非按下状态移过该按钮，则按钮从未按下状态改变为按下状态。此特性在使用触摸面板时有用。
如果某按钮只应对水平更改作出反应，则 BUTTON_REACT_ON_LEVEL 应设置为 1。这时，该按钮仅在 PID 在该按钮上按下然后释放时才改变状态。然后，如果将处于按下状态的 PID 移过该按钮，它将没有反应。在对话框应对 WM_NOTIFICATION_CLICKED 作出反应时，可以使用此特性。
示例 (BUTTON_REACT_ON_LEVEL = 0)：一个对话框 （对话框 2）显示在另一对话框 （对话框 1）之上。对话框 2 的 “关闭”按钮的位置与对话框 1 的相同。现在按下对话框 2 的 “关闭”按钮，移除对话框 2。PID 现在处于按下状态。如果现在移动按钮然后再释放它，则对话框 1 的按钮将从未按下状态变为按下状态。
这种不必要的特性可通过将 BUTTON_REACT_ON_LEVEL 设置为 1 避免，
或者可对此配置选项使用函数 BUTTON_SetReactOnLevel()。

BUTTON_BKCOLOR0_DEFAULT, BUTTON_BKCOLOR1_DEFAULT

按钮的默认设置是在按下状态时使用白色背景，这样做是有意的，因为它使得被按下的按钮非常明显，无论在什么显示器上。如果想使按钮在按下和未按下状态下的背景色相同，请将 BUTTON_BKCOLOR1_DEFAULT 更改为 BUTTON_BKCOLOR0_DEFAULT。

15.4.2 通知代码

以下事件是按钮小工具作为 WM_NOTIFY_PARENT 消息的一部分发送给其父窗口的：

消息	描述
WM_NOTIFICATION_CLICKED	按钮已被点击。
WM_NOTIFICATION_RELEASED	按钮已被释放。
WM_NOTIFICATION_MOVED_OUT	按钮已被点击，且指针已移出按钮并且没有释放。

15.4.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_ENTER	此类键按下后，按钮认为它被按下并立即释放，然后作出相应反应。
GUI_KEY_SPACE	此类键按下后，按钮状态改变为已按下。键释放后，按钮状态改变为未按下。

15.4.4 BUTTON API

下表按字母顺序列出了可用的 emWin BUTTON 相关例程。这些例程的详细说明如下。

例程	描述
BUTTON_Create()	创建 BUTTON 小工具。（弃用）
BUTTON_CreateAsChild()	将 BUTTON 小工具创建为子窗口。（弃用）
BUTTON_CreateEx()	创建 BUTTON 小工具。
BUTTON_CreateIndirect()	从资源表项创建 BUTTON 小工具。
BUTTON_CreateUser()	使用额外字节作为用户数据创建 BUTTON 小工具。
BUTTON_GetBitmap()	返回指向 BUTTON 位图的指针。
BUTTON_GetBkColor()	返回 BUTTON 的背景色。
BUTTON_GetDefaultBkColor()	返回 BUTTON 小工具的默认背景色。
BUTTON_GetDefaultFont()	返回 BUTTON 小工具的默认字体。
BUTTON_GetDefaultTextAlign()	返回 BUTTON 小工具的默认文本对齐方式。
BUTTON_GetDefaultTextColor()	返回 BUTTON 小工具的默认文本颜色。
BUTTON_GetFont()	返回 BUTTON 小工具字体的指针。
BUTTON_GetText()	检索指定 BUTTON 的文本。
BUTTON_GetTextAlign()	返回 BUTTON 文本的对齐方式。
BUTTON_GetTextColor()	返回指定 BUTTON 的文本颜色。
BUTTON_GetUserData()	检索用 BUTTON_SetUserData() 设置的数据。
BUTTON_IsPressed()	返回表示按钮是否按下的值。
BUTTON_SetBitmap()	设置显示 BUTTON 时使用的位图。
BUTTON_SetBitmapEx()	设置显示 BUTTON 时使用的位图。
BUTTON_SetBkColor()	设置按钮的背景色。
BUTTON_SetBMP()	设置显示 BUTTON 时使用的位图。
BUTTON_SetBMPEx()	设置显示 BUTTON 时使用的位图。
BUTTON_SetDefaultBkColor()	设置 BUTTON 小工具的默认背景色。
BUTTON_SetDefaultFont()	设置 BUTTON 小工具的默认字体。
BUTTON_SetDefaultTextAlign()	设置 BUTTON 小工具的默认文本对齐方式。
BUTTON_SetDefaultTextColor()	设置 BUTTON 小工具的默认文本颜色。
BUTTON_SetFocussable()	设置接收输入焦点的能力。

例程	描述
BUTTON_SetFont()	选择文本的字体。
BUTTON_SetPressed()	将按钮的状态设置为按下或未按下。
BUTTON_SetReactOnLevel()	将所有 BUTTON 小工具设置为对水平作出反应。
BUTTON_SetStreamedBitmap()	设置显示 BUTTON 小工具时使用的位图。
BUTTON_SetStreamedBitmapEx()	设置显示 BUTTON 小工具时使用的位图。
BUTTON_SetText()	设置文本。
BUTTON_SetTextAlign()	设置 BUTTON 文本的对齐方式。
BUTTON_SetTextColor()	设置文本的颜色。
BUTTON_SetTextOffset()	根据当前的文本对齐方式设置，调节按钮文本的位置。
BUTTON_SetUserData()	设置 BUTTON 小工具的额外数据。

BUTTON_Create()

（弃用，应使用 BUTTON_CreateEx() 代替）

描述

在指定位置创建指定尺寸的 BUTTON 小工具。

原型

```
BUTTON_Handle BUTTON_Create(int x0,    int y0,
                            int xsize, int ysize,
                            int Id,    int Flags);
```

参数	描述
x0	按钮的最左像素 （在父坐标中）。
y0	按钮的最上像素 （在父坐标中）。
xsize	按钮的水平尺寸 （单位：像素）。
ysize	按钮的垂直尺寸 （单位：像素）。
Id	按钮按下时要返回的 ID。
Flags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。

返回值

已创建 BUTTON 小工具的句柄；函数失败时为 0。

BUTTON_CreateAsChild()

（弃用，应使用 BUTTON_CreateEx 代替）

描述

将 BUTTON 小工具创建为子窗口。

原型

```
BUTTON_Handle BUTTON_CreateAsChild(int     x0,      int y0,
                                   int     xsize,   int ysize,
                                   WM_HWIN hParent, int Id,
                                   int     Flags);
```


参数	描述
x0	按钮相对于父窗口的 X 位置。
y0	按钮相对于父窗口的 Y 位置。
xsize	按钮的水平尺寸 （单位：像素）。
ysize	按钮的垂直尺寸 （单位：像素）。
hParent	父窗口的句柄。如果为 0，则 BUTTON 小工具将是桌面 （顶级窗口）的子项。
Id	按钮按下时要返回的 ID。
Flags	窗口创建标记 （请参见 BUTTON_Create()）。

返回值

已创建 BUTTON 小工具的句柄；函数失败时为 0。

BUTTON_CreateEx()

描述

在指定位置创建指定尺寸的 BUTTON 小工具。

原型

```
BUTTON_Handle BUTTON_CreateEx(int     x0,       int y0,
                              int     xsize,    int ysize,
                              WM_HWIN hParent, int WinFlags,
                              int     ExFlags, int Id);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则 BUTTON 小工具将是桌面 （顶级窗口）的子项。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW（有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	未使用，保留供日后使用。
Id	小工具的窗口 ID。

返回值

已创建 BUTTON 小工具的句柄；函数失败时为 0。

其他信息

如果存储用户数据的能力比较重要，则应使用函数 BUTTON_CreateUser()。

BUTTON_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。资源的元素 Flags 和 Para 将省略，因为未使用参数。

BUTTON_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细描述，可参见函数 BUTTON_CreateEx()。

BUTTON_GetBitmap()

描述

返回可选 BUTTON 位图的指针。

原型

```
const GUI_BITMAP * BUTTON_GetBitmap(BUTTON_Handle hObj,
                                    unsigned int  Index);
```

参数	描述
hObj	小工具的句柄。
Index	所需位图的索引 （见下表）。

参数 Index 的允许值	
BUTTON_BI_DISABLED	用于禁用状态的位图。
BUTTON_BI_PRESSED	用于按下状态的位图。
BUTTON_BI_UNPRESSED	用于未按下状态的位图。

返回值

位图的指针，如果无位图，则返回 0。

其他信息

有关如何设置按钮位图的详细信息，请参见 “BUTTON_SetBitmap()”（第 363 页）和 “BUTTON_SetBitmapEx()”（第 363 页）。

BUTTON_GetBkColor()

描述

返回给定 BUTTON 小工具的背景色。

原型

```
GUI_COLOR BUTTON_GetBkColor(BUTTON_Handle hObj, unsigned int Index);
```

参数	描述
hObj	小工具的句柄。
Index	颜色的索引 （见下表）。

参数 Index 的允许值	
BUTTON_CI_DISABLED	用于禁用状态的颜色。
BUTTON_CI_PRESSED	用于按下状态的颜色。
BUTTON_CI_UNPRESSED	用于未按下状态的颜色。

返回值

给定 BUTTON 小工具的背景色。

BUTTON_GetDefaultBkColor()

描述

返回 BUTTON 小工具的默认背景色。

原型

```
GUI_COLOR BUTTON_GetDefaultBkColor(unsigned Index);
```

参数	描述
Index	颜色的索引 （见下表）。

参数 Index 的允许值	
BUTTON_CI_DISABLED	用于禁用状态的颜色。
BUTTON_CI_PRESSED	用于按下状态的颜色。
BUTTON_CI_UNPRESSED	用于未按下状态的颜色。

返回值

BUTTON 小工具的默认背景色。

BUTTON_GetDefaultFont()

描述

返回用于显示 BUTTON 小工具文本的字体的指针。

原型

```
const GUI_FONT * BUTTON_GetDefaultFont(void);
```

返回值

用于显示 BUTTON 小工具文本的字体的指针。

BUTTON_GetDefaultTextAlign()

描述

返回用于显示 BUTTON 小工具文本的默认文本对齐方式。

原型

```
int BUTTON_GetDefaultTextAlign(void);
```

返回值

用于显示 BUTTON 小工具文本的默认文本对齐方式。

BUTTON_GetDefaultTextColor()

描述

返回用于显示 BUTTON 小工具文本的默认文本颜色。

原型

```
GUI_COLOR BUTTON_GetDefaultTextColor(unsigned Index);
```

参数	描述
Index	颜色的索引 （见下表）。

参数 Index 的允许值	
BUTTON_CI_DISABLED	用于禁用状态的颜色。
BUTTON_CI_PRESSED	用于按下状态的颜色。
BUTTON_CI_UNPRESSED	用于未按下状态的颜色。

返回值

用于显示 BUTTON 小工具文本的默认文本颜色。

BUTTON_GetFont()

描述

返回用于显示给定 BUTTON 小工具文本的字体的指针。

原型

```
const GUI_FONT * BUTTON_GetFont(BUTTON_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

用于显示给定 BUTTON 小工具文本的字体的指针。

BUTTON_GetText()

描述

检索指定 BUTTON 小工具的文本。

原型

```
void BUTTON_GetText(BUTTON_Handle hObj, char * pBuffer, int MaxLen);
```

参数	描述
hObj	小工具的句柄。
pBuffer	指向缓冲区的指针。
MaxLen	缓冲区的大小。

BUTTON_GetTextAlign()

描述

返回 BUTTON 文本的对齐方式。

原型

```
int BUTTON_GetTextAlign(BUTTON_Handle hObj);
```

参数	描述
hObj	BUTTON 小工具的句柄。

返回值

BUTTON 文本的对齐方式。

BUTTON_GetTextColor()

描述

返回给定 BUTTON 小工具的文本颜色。

原型

```
GUI_COLOR BUTTON_GetTextColor(BUTTON_Handle hObj, unsigned int Index);
```

参数	描述
hObj	小工具的句柄。
Index	颜色的索引 （见下表）。

参数 Index 的允许值	
BUTTON_CI_DISABLED	用于禁用状态的颜色。
BUTTON_CI_PRESSED	用于按下状态的颜色。
BUTTON_CI_UNPRESSED	用于未按下状态的颜色。

返回值

给定 BUTTON 小工具的文本颜色。

BUTTON_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

BUTTON_IsPressed()

描述

返回表示 BUTTON 是否按下的值。

原型

```
unsigned BUTTON_IsPressed(BUTTON_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

如果按钮按下则为 1，否则为 0。

BUTTON_SetBitmap()

描述

设置显示指定按钮时要使用的位图。

原型

```
void BUTTON_SetBitmap(BUTTON_Handle      hObj,
                      unsigned int       Index,
                      const GUI_BITMAP * pBitmap);
```

参数	描述
hObj	按钮的句柄。
Index	位图的索引（见下表）。
pBitmap	位图结构的指针。

参数 Index 的允许值	
BUTTON_BI_DISABLED	用于禁用状态的位图。
BUTTON_BI_PRESSED	用于按下状态的位图。
BUTTON_BI_UNPRESSED	用于未按下状态的位图。

其他信息

如果仅设置了未按下状态的位图，则当按钮按下或者禁用时，按钮也将显示该位图。

BUTTON_SetBitmapEx()

描述

设置显示指定按钮时要使用的位图。

原型

```
void BUTTON_SetBitmapEx(BUTTON_Handle      hObj,
                        unsigned int       Index,
                        const GUI_BITMAP * pBitmap,
                        int                x,
                        int                y);
```

参数	描述
hObj	按钮的句柄。
Index	位图的索引 （请参见 BUTTON_SetBitmap()）。
pBitmap	位图结构的指针。
x	位图相对于按钮的 X 位置。
y	位图相对于按钮的 Y 位置。

其他信息

如果仅设置了未按下状态的位图，则当按钮按下或者禁用时，按钮也将显示该位图。

BUTTON_SetBkColor()

描述

设置按钮背景颜色。

原型

```
void BUTTON_SetBkColor(BUTTON_Handle hObj,   unsigned int Index,
                       GUI_COLOR     Color);
```

参数	描述
hObj	按钮的句柄。
Index	颜色的索引 （见下表）。
Color	要设置的背景色。

参数 Index 的允许值	
BUTTON_CI_DISABLED	设置按钮禁用时要使用的颜色。
BUTTON_CI_PRESSED	设置按钮按下时要使用的颜色。
BUTTON_CI_UNPRESSED	设置按钮未按下时要使用的颜色。

BUTTON_SetBMP()

描述

设置显示指定按钮时要使用的位图。

原型

```
void BUTTON_SetBMP(BUTTON_Handle hObj, unsigned int Index,
                   const void * pBitmap);
```

参数	描述
hObj	小工具的句柄。
Index	位图的索引 （见下表）。
pBitmap	位图文件数据的指针

参数 Index 的允许值	
BUTTON_BI_DISABLED	设置按钮禁用时要使用的位图。
BUTTON_BI_PRESSED	设置按钮按下时要使用的位图。
BUTTON_BI_UNPRESSED	设置按钮未按下时要使用的位图。

其他信息

如果仅设置了未按下状态的位图，则当按钮按下或者禁用时，按钮也将显示该位图。
有关位图文件的其他信息，请参阅 “BMP 文件支持”（第 142 页）。

BUTTON_SetBMPEx()

描述

设置显示指定按钮时要使用的位图。

原型

```
void BUTTON_SetBMPEx(BUTTON_Handle   hObj,    unsigned int Index,
                     const void    * pBitmap, int          x,
                     int             y);
```

参数	描述
hObj	小工具的句柄。
Index	位图的索引（请参见 BUTTON_SetBitmap()）。
pBitmap	位图文件数据的指针
x	位图相对于按钮的 X 位置。
y	位图相对于按钮的 Y 位置。

其他信息

如果仅设置了未按下状态的位图，则当按钮按下或者禁用时，按钮也将显示该位图。
有关位图文件的其他信息，请参阅 “BMP 文件支持”（第 142 页）。

BUTTON_SetDefaultBkColor()

描述

设置用于 BUTTON 小工具的默认背景色。

原型

```
void BUTTON_SetDefaultBkColor(GUI_COLOR Color, unsigned Index);
```

参数	描述
Color	要使用的颜色。
Index	颜色的索引（见下表）。

参数 Index 的允许值	
BUTTON_CI_DISABLED	用于禁用状态的颜色。
BUTTON_CI_PRESSED	用于按下状态的颜色。
BUTTON_CI_UNPRESSED	用于未按下状态的颜色。

BUTTON_SetDefaultFocusColor()

描述

设置 BUTTON 小工具的默认焦点矩形颜色。

原型

```
GUI_COLOR BUTTON_SetDefaultFocusColor(GUI_COLOR Color);
```

参数	描述
Color	用于 BUTTON 小工具的默认颜色。

返回值

先前的默认聚焦框颜色。

其他信息

更多详细信息，请参阅 “BUTTON_SetFocusColor()”（第 366 页）。

BUTTON_SetDefaultFont()

描述

设置用于显示 BUTTON 小工具文本的 GUI_FONT 结构的指针。

原型

```
void BUTTON_SetDefaultFont(const GUI_FONT * pFont);
```

参数	描述
pFont	要使用的 GUI_FONT 结构的指针。

BUTTON_SetDefaultTextAlign()

描述

设置用于显示 BUTTON 小工具文本的默认文本对齐方式。

原型

```
void BUTTON_SetDefaultTextAlign(int Align);
```

参数	描述
Align	要使用的文本对齐方式。更多详细信息，请参阅 “GUI_SetTextAlign()”（第 82 页）。

BUTTON_SetDefaultTextColor()

描述

设置用于显示 BUTTON 小工具文本的默认文本颜色。

原型

```
void BUTTON_SetDefaultTextColor(GUI_COLOR Color, unsigned Index);
```

参数	描述
Color	要使用的默认文本颜色。
Index	颜色的索引 （见下表）。

参数 Index 的允许值	
BUTTON_CI_DISABLED	用于禁用状态的颜色。
BUTTON_CI_PRESSED	用于按下状态的颜色。
BUTTON_CI_UNPRESSED	用于未按下状态的颜色。

BUTTON_SetFocusColor()

之前	之后
	

描述

设置用于渲染 BUTTON 小工具焦点矩形的颜色。

原型

```
GUI_COLOR BUTTON_SetFocusColor(BUTTON_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄。
Color	聚焦框所使用的颜色。

返回值

先前的聚焦框颜色。

其他信息

仅当此小工具具有输入焦点时，聚焦框才可见。

BUTTON_SetFocussable()

描述

设置接收输入焦点的能力。

原型

```
void BUTTON_SetFocussable(BUTTON_Handle hObj, int State);
```

参数	描述
hWin	窗口句柄。
State	见下表

参数 State 的允许值	
1	按钮能接收输入焦点
0	按钮不能接收输入焦点

BUTTON_SetFont()

描述

设置按钮字体。

原型

```
void BUTTON_SetFont(BUTTON_Handle hObj, const GUI_FONT* pFont);
```

参数	描述
hObj	按钮的句柄。
pFont	字体的指针。

其他信息

如果未选择字体，则将使用 BUTTON_FONT_DEF。

BUTTON_SetPressed()

描述

将按钮的状态设置为按下或未按下。

原型

```
void BUTTON_SetPressed(BUTTON_Handle hObj, int State);
```

参数	描述
hObj	按钮的句柄。
State	状态，对于按下为 1，对于未按下为 0

BUTTON_SetReactOnLevel()

描述

将所有 BUTTON 小工具设置为对水平作出反应。

原型

```
void BUTTON_SetReactOnLevel(void);
```

其他信息

或者可对此函数使用配置选项 BUTTON_REACT_ON_LEVEL。

BUTTON_SetStreamedBitmap()

描述

设置显示指定按钮对象时要使用的流位图。

原型

```
void BUTTON_SetStreamedBitmap(BUTTON_Handle             hObj,
                              unsigned int              Index,
                              const GUI_BITMAP_STREAM * pBitmap);
```

参数	描述
hObj	按钮的句柄。
Index	位图的索引 （请参见 BUTTON_SetBitmap()）。
pBitmap	位图流的指针。

其他信息

有关流位图的详情，请见 “GUI_DrawStreamedBitmap()”（第 118 页）。

BUTTON_SetStreamedBitmapEx()

描述

设置显示指定按钮对象时要使用的流位图。

原型

```
void BUTTON_SetStreamedBitmapEx(BUTTON_Handle             hObj,
                                unsigned int              Index,
                                const GUI_BITMAP_STREAM * pBitmap,
                                int                       x,
                                int                       y);
```

参数	描述
hObj	按钮的句柄。
Index	位图的索引 （请参见 BUTTON_SetBitmap()）。
pBitmap	位图流的指针。
x	位图相对于按钮的 X 位置。
y	位图相对于按钮的 Y 位置。

其他信息

有关流位图的详情，请见 “GUI_DrawStreamedBitmap()”（第 118 页）。

BUTTON_SetText()

描述

设置要显示在按钮上的文本。

原型

```
void BUTTON_SetText(BUTTON_Handle hObj, const char * s);
```

参数	描述
hObj	按钮的句柄。
s	要显示的文本。

BUTTON_SetTextAlign()

描述

设置按钮文本的对齐方式。

原型

```
void BUTTON_SetTextAlign(BUTTON_Handle hObj, int Align);
```

参数	描述
hObj	按钮的句柄。
Align	要设置的文本对齐方式 （请参见 “GUI_SetTextAlign()”（第 82 页））

其他信息

文本对齐方式的默认值是 GUI_TA_HCENTER | GUI_TA_VCENTER。

BUTTON_SetTextColor()

描述

设置按钮文本颜色。

原型

```
void BUTTON_SetTextColor(BUTTON_Handle hObj, unsigned int Index,
                         GUI_COLOR Color);
```

参数	描述
hObj	按钮的句柄。
Index	文本颜色的索引 （请参见 BUTTON_SetBkColor()）。
Color	要设置的文本颜色。

BUTTON_SetTextOffset()

描述

根据当前的文本对齐方式设置，调节按钮文本的位置。

原型

```
void BUTTON_SetTextOffset(BUTTON_Handle hObj, int xPos, int yPos);
```

参数	描述
hObj	按钮的句柄。
xPos	要使用的 x 轴偏移。默认值为 0。
yPos	要使用的 y 轴偏移。默认值为 0。

BUTTON_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

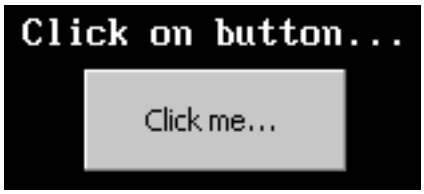
15.4.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_ButtonSimple.c
- WIDGET_ButtonPhone.c
- WIDGET_ButtonRound.c

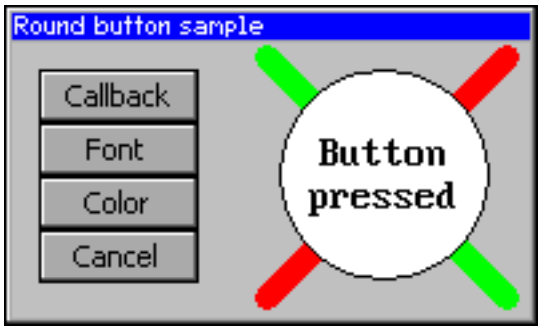
需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_ButtonSimple.c 的屏幕截图：



WIDGET_ButtonPhone.c 的屏幕截图：

WIDGET_ButtonRound.c 的屏幕截图：



15.5 CHECKBOX：复选框小工具

复选框是选择各种不同选项的最常用小工具之一。用户可选中或取消选中复选框，并且可一次选中任意个框。使用键盘界面时，被聚焦的复选框的状态可使用 <SPACE> 键切换。禁用的框显示为灰色，如下表所示，其中描述了每种可能的复选框的外观：

	取消选中	选中	第三种状态
启用			
禁用			

所有 CHECKBOX 相关例程都位于文件 CHECKBOX*.c、 CHECKBOX.h 中。所有标识符的前缀都是 CHECKBOX。

皮肤设置 ...

... 可用于该小工具。上述屏幕截图显示使用默认皮肤的小工具。有关详细信息，请参阅“皮肤设置”一章。

15.5.1 配置选项

类型	宏	默认值	描述
N	CHECKBOX_BKCOLOR_DEFAULT	0xC0C0C0	默认背景色。
N	CHECKBOX_BKCOLOR0_DEFAULT	0x808080	默认图像的背景色，禁用状态。
N	CHECKBOX_BKCOLOR1_DEFAULT	GUI_WHITE	默认图像的背景色，启用状态。
N	CHECKBOX_FGCOLOR0_DEFAULT	0x101010	默认图像的前景色，禁用状态。
N	CHECKBOX_FGCOLOR1_DEFAULT	GUI_BLACK	默认图像的前景色，启用状态。
N	CHECKBOX_FOCUSCOLOR_DEFAULT	GUI_BLACK	用于渲染焦点矩形的前一颜色。
S	CHECKBOX_FONT_DEFAULT	&GUI_Font13_1	用于显示可选复选框文本的默认字体。
S	CHECKBOX_IMAGE0_DEFAULT	（参见上表）	用于绘制小工具 （如果选中）的位图的指针，禁用状态。
S	CHECKBOX_IMAGE1_DEFAULT	（参见上表）	用于绘制小工具 （如果选中）的位图的指针，启用状态。
N	CHECKBOX_SPACING_DEFAULT	4	用于在框旁边显示可选复选框文本的间距。
N	CHECKBOX_TEXTALIGN_DEFAULT	GUI_TA_LEFT | GUI_TA_VCENTER	可选复选框文本的默认对齐方式。
N	CHECKBOX_TEXTCOLOR_DEFAULT	GUI_BLACK	用于显示可选复选框文本的默认颜色。

15.5.2 通知代码

以下事件是复选框小工具作为 WM_NOTIFY_PARENT 消息的一部分发送给其父窗口的：

消息	描述
WM_NOTIFICATION_CLICKED	复选框已被点击。
WM_NOTIFICATION_RELEASED	复选框已被释放。
WM_NOTIFICATION_MOVED_OUT	复选框已被点击，且指针已移出框并且没有释放。
WM_NOTIFICATION_VALUE_CHANGED	复选框的状态已改变。

15.5.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_SPACE	切换小工具的选中状态。

15.5.4 CHECKBOX API

下表按字母顺序列出了可用的 emWin CHECKBOX 相关例程。这些例程的详细说明如下。

例程	描述
CHECKBOX_Check()	将复选框的状态设置为选中。（弃用）
CHECKBOX_Create()	创建 CHECKBOX 小工具。（弃用）
CHECKBOX_CreateEx()	创建 CHECKBOX 小工具。
CHECKBOX_CreateIndirect()	从资源表项创建 CHECKBOX 小工具。
CHECKBOX_CreateUser()	使用额外字节作为用户数据创建 CHECKBOX 小工具。
CHECKBOX_GetDefaultBkColor()	返回 CHECKBOX 小工具的默认背景色。
CHECKBOX_GetDefaultFont()	返回用于显示 CHECKBOX 小工具文本的默认字体。
CHECKBOX_GetDefaultSpacing()	返回 CHECKBOX 小工具文本和框之间的默认间距。
CHECKBOX_GetDefaultTextAlign()	返回用于显示 CHECKBOX 小工具文本的默认对齐方式。
CHECKBOX_GetDefaultTextColor()	返回用于显示 CHECKBOX 小工具文本的默认文本颜色。
CHECKBOX_GetState()	返回复选框的当前状态。
CHECKBOX_GetText()	返回复选框的文本。
CHECKBOX_GetUserData()	检索使用 CHECKBOX_SetUserData() 设置的数据。

例程	描述
CHECKBOX_IsChecked()	返回复选框的当前状态（选中或未选中）。
CHECKBOX_SetBkColor()	设置给定 CHECKBOX 小工具的背景色。
CHECKBOX_SetBoxBkColor()	设置框区域的背景色。
CHECKBOX_SetDefaultBkColor()	设置 CHECKBOX 小工具的默认背景色。
CHECKBOX_SetDefaultFocusColor()	设置 CHECKBOX 小工具的默认焦点矩形颜色。
CHECKBOX_SetDefaultFont()	设置用于显示 CHECKBOX 小工具文本的默认字体。
CHECKBOX_SetDefaultImage()	设置复选框被选中后要显示的默认图像。
CHECKBOX_SetDefaultSpacing()	设置 CHECKBOX 小工具文本和框之间的默认间距。
CHECKBOX_SetDefaultTextAlign()	设置用于显示复选框文本的默认对齐方式。
CHECKBOX_SetDefaultTextColor()	设置用于显示 CHECKBOX 小工具文本的默认文本颜色。
CHECKBOX_SetFocusColor()	设置聚焦框的颜色。
CHECKBOX_SetFont()	设置复选框字体。
CHECKBOX_SetImage()	设置复选框被选中后要显示的图像。
CHECKBOX_SetNumStates()	设置复选框的可能状态的数量（2 或 3）。
CHECKBOX_SetSpacing()	设置框和复选框文本之间的间距。
CHECKBOX_SetState()	设置 CHECKBOX 小工具的状态。
CHECKBOX_SetText()	设置 CHECKBOX 小工具的文本。
CHECKBOX_SetTextAlign()	设置用于显示 CHECKBOX 小工具文本的对齐方式。
CHECKBOX_SetTextColor()	设置用于显示 CHECKBOX 小工具文本的颜色。
CHECKBOX_SetUserData()	设置 CHECKBOX 小工具的额外数据。
CHECKBOX_Uncheck()	将复选框的状态设置为未选中。（弃用）

CHECKBOX_Check()

（弃用，应使用 CHECKBOX_SetState() 代替）

之前	之后
	

描述

将指定复选框设置为选中状态。

原型

```
void CHECKBOX_Check(CHECKBOX_Handle hObj);
```

参数	描述
hObj	复选框的句柄。

CHECKBOX_Create()

（弃用，应使用 CHECKBOX_CreateEx 代替）

描述

在指定位置创建指定尺寸的 CHECKBOX 小工具。

原型

```
CHECKBOX_Handle CHECKBOX_Create(int     x0,      int y0,
                                int     xsize,   int ysize,
                                WM_HWIN hParent, int Id,
                                int     Flags);
```

参数	描述
x0	复选框的最左像素 （在父坐标中）。
y0	复选框的最上像素 （在父坐标中）。
xsize	复选框的水平尺寸 （单位：像素）。
ysize	复选框的垂直尺寸 （单位：像素）。
hParent	父窗口的句柄。
Id	将返回的 ID。
Flags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。

返回值

已创建 CHECKBOX 小工具的句柄；函数失败时为 0。

其他信息

如果参数 xsize 或 ysize 为 0，则将使用位图的尺寸作为复选框的默认尺寸。

CHECKBOX_CreateEx()

描述

在指定位置创建指定尺寸的 CHECKBOX 小工具。

原型

```
CHECKBOX_Handle CHECKBOX_CreateEx(int     x0,      int y0,
                                  int     xsize,   int ysize,
                                  WM_HWIN hParent, int WinFlags,
                                  int     ExFlags, int Id);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新的 CHECKBOX 小工具将是桌面 （顶级窗口）的子项。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	未使用，保留供日后使用。
Id	小工具的窗口 ID。

返回值

已创建 CHECKBOX 小工具的句柄；函数失败时为 0。

其他信息

如果参数 xsize 或 ysize 为 0，则将使用默认复选标记位图的尺寸 （11 x 11 像素）加上效果尺寸作为复选框的默认尺寸。如果所需的复选框尺寸不同于默认尺寸，则可使用函数 CHECKBOX_SetImage() 来设置用户定义的复选标记图像。
如果要与该小工具一同显示复选框文本，则尺寸应足够大到可显示 “框 + 文本 + 框和文本之间的间距”。

CHECKBOX_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。资源的元素 Flags 和 Para 将省略，因为未使用参数。

CHECKBOX_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细描述，可参见函数 CHECKBOX_CreateEx()。

CHECKBOX_GetDefaultBkColor()

描述

返回新复选框小工具的默认背景色。

原型

```
GUI_COLOR CHECKBOX_GetDefaultBkColor(void);
```

返回值

新复选框小工具的默认背景色。

其他信息

此函数返回的背景色不是框中显示的背景色，而是该小工具其余部分的背景色。
更多详细信息，请参阅 “CHECKBOX_SetBoxBkColor()”（第 378 页）。

CHECKBOX_GetDefaultFont()

描述

返回用于显示新复选框小工具的复选框文本的 GUI_FONT 结构的指针。

原型

```
const GUI_FONT * CHECKBOX_GetDefaultFont(void);
```

返回值

用于显示新复选框小工具的复选框文本的 GUI_FONT 结构的指针。

其他信息

更多详细信息，请参阅 “CHECKBOX_SetFont()”（第 381 页）。

CHECKBOX_GetDefaultSpacing()

描述

返回框和文本 （用于显示新复选框小工具的复选框文本）之间的默认间距。

原型

```
int CHECKBOX_GetDefaultSpacing(void);
```

返回值

框和文本 （用于显示新复选框小工具的复选框文本）之间的默认间距。

其他信息

更多详细信息，请参阅 “CHECKBOX_SetSpacing()”（第 383 页）。

CHECKBOX_GetDefaultTextAlign()

描述

返回用于显示新复选框小工具的复选框文本的默认对齐方式。

原型

```
int CHECKBOX_GetDefaultAlign(void);
```

返回值

用于显示复选框文本的默认对齐方式。

其他信息

更多详细信息，请参阅 “CHECKBOX_SetTextAlign()”（第 384 页）。

CHECKBOX_GetDefaultTextColor()

描述

返回用于显示新复选框小工具的复选框文本的默认文本颜色。

原型

```
GUI_COLOR CHECKBOX_GetDefaultTextColor(void);
```

返回值

用于显示新复选框小工具的复选框文本的默认文本颜色。

其他信息

更多详细信息，请参阅 “CHECKBOX_SetTextColor()”（第 384 页）。

CHECKBOX_GetState()

描述

返回给定复选框的当前状态。

原型

```
int CHECKBOX_GetState(CHECKBOX_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

给定复选框的当前状态。

其他信息

默认情况下，复选框有 2 种状态，选中 (1) 和取消选中 (0)。使用函数 CHECKBOX_SetNumStates() 可将可能状态增加到 3。如果复选框处于第三种状态，则该函数返回 2。
更多详细信息，请参阅 “CHECKBOX_SetNumStates()”（第 382 页）。

CHECKBOX_GetText()

描述

返回给定复选框的可选文本。

原型

```
int CHECKBOX_GetText(CHECKBOX_Handle hObj, char * pBuffer, int MaxLen);
```

参数	描述
hObj	小工具的句柄。
pBuffer	复制文本所在缓冲区的指针。
MaxLen	缓冲区大小 （以字节为单位）。

返回值

复制到缓冲区的文本长度。

其他信息

如果复选框不含文本，则该函数返回 0 且缓冲区保持不变。

CHECKBOX_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

CHECKBOX_IsChecked()

描述

返回指定 CHECKBOX 小工具的当前状态 （选中或未选中）。

原型

```
int CHECKBOX_IsChecked(CHECKBOX_Handle hObj);
```

参数	描述
hObj	复选框的句柄。

返回值

0：未选中
1：选中

CHECKBOX_SetBkColor()

之前	之后
	

描述

设置用于显示复选框背景的背景色。

原型

```
void CHECKBOX_SetBkColor(CHECKBOX_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄。
Color	用于绘制背景的颜色，或者要在透明模式下操作 GUI_INVALID_COLOR。

其他信息

如果复选框应在透明模式下操作，则应使用 GUI_INVALID_COLOR。

CHECKBOX_SetBoxBkColor()

之前	之后
	

描述

设置框区域的背景色。

原型

```
GUI_COLOR CHECKBOX_SetBoxBkColor(CHECKBOX_Handle hObj,
                                 GUI_COLOR       Color,
                                 int             Index);
```

参数	描述
hObj	小工具的句柄。
Color	要使用的颜色。
Index	（参见下表）

参数 Index 的允许值	
CHECKBOX_CI_DISABLED	用于禁用状态的背景色。
CHECKBOX_CI_ENABLED	用于启用状态的背景色。

返回值

前一背景色。

其他信息

如果小工具使用的图像是透明的或者没有使用图像，则仅此函数设置的颜色可见。此小工具的默认图像是透明的。

CHECKBOX_SetDefaultBkColor()

描述

设置用于新复选框小工具的默认背景色。

原型

```
void CHECKBOX_SetDefaultBkColor(GUI_COLOR Color);
```

参数	描述
Color	要使用的颜色，对于透明则为 GUI_INVALID_COLOR。

其他信息

更多详细信息，请参阅 “CHECKBOX_SetBkColor()”（第 378 页）。

CHECKBOX_SetDefaultFocusColor()

描述

设置用于渲染新复选框小工具的焦点矩形的颜色。

原型

```
GUI_COLOR CHECKBOX_SetDefaultFocusColor(GUI_COLOR Color);
```

参数	描述
Color	要使用的颜色。

返回值

用于渲染焦点矩形的前一颜色。

其他信息

更多信息，请参见 “CHECKBOX_SetFocusColor()”（第 381 页）。

CHECKBOX_SetDefaultFont()

描述

设置用于显示新复选框小工具的复选框文本的 GUI_FONT 结构的指针。

原型

```
void CHECKBOX_SetDefaultFont(const GUI_FONT * pFont);
```

参数	描述
pFont	要使用的 GUI_FONT 结构的指针。

其他信息

更多详细信息，请参阅 “CHECKBOX_SetFont()”（第 381 页）。

CHECKBOX_SetDefaultImage()

描述

设置新复选框被选中后，用于显示它们的图像。

原型

```
void CHECKBOX_SetDefaultImage(const GUI_BITMAP * pBitmap,
```

```
                                unsigned int       Index);
```

参数	描述
pBitmap	位图的指针。
Index	（参见下表）

参数 Index 的允许值	
CHECKBOX_BI_INACTIV_UNCHECKED	设置复选框被取消选中并禁用时显示的位图。
CHECKBOX_BI_ACTIV_UNCHECKED	设置复选框被取消选中并启用时显示的位图。
CHECKBOX_BI_INACTIV_CHECKED	设置复选框被选中并禁用时显示的位图。
CHECKBOX_BI_ACTIV_CHECKED	设置复选框被选中并启用时显示的位图。
CHECKBOX_BI_INACTIV_3STATE	设置复选框处于第三种状态并禁用时显示的位图。
CHECKBOX_BI_ACTIV_3STATE	设置复选框处于第三种状态并启用时显示的位图。

其他信息

图像必须填充复选框的整个内部区域。

CHECKBOX_SetDefaultSpacing()

描述

设置框和文本 （用于显示新复选框小工具的复选框文本）之间的默认间距。

原型

```
void CHECKBOX_SetDefaultSpacing(int Spacing);
```

参数	描述
Spacing	用于新复选框小工具的框和文本之间的像素数量。

其他信息

更多详细信息，请参阅 “CHECKBOX_SetSpacing()”（第 383 页）。

CHECKBOX_SetDefaultTextAlign()

描述

设置用于显示新复选框小工具的复选框文本的默认对齐方式。

原型

```
void CHECKBOX_SetDefaultTextAlign(int Align);
```

参数	描述
Align	用于显示新复选框小工具文本的对齐方式。

其他信息

更多详细信息，请参阅 “CHECKBOX_SetTextAlign()”（第 384 页）。

CHECKBOX_SetDefaultTextColor()

描述

设置用于显示新复选框小工具的复选框文本的默认文本颜色。

原型

```
void CHECKBOX_SetDefaultTextColor(GUI_COLOR Color);
```

参数	描述
Color	要使用的颜色。

其他信息

更多详细信息，请参阅 “CHECKBOX_SetTextColor()”（第 384 页）。

CHECKBOX_SetFocusColor()

之前	之后
	

描述

设置用于渲染焦点矩形的颜色。

原型

```
GUI_COLOR CHECKBOX_SetFocusColor(CHECKBOX_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄。

返回值

先前的聚焦框颜色。

其他信息

仅当此小工具具有输入焦点时，聚焦框才可见。

CHECKBOX_SetFont()

描述

设置复选框字体。

原型

```
void CHECKBOX_SetFont(CHECKBOX_Handle               hObj,
                      const GUI_FONT GUI_UNI_PTR * pFont);
```

参数	描述
hObj	复选框的句柄。
pFont	字体的指针。

CHECKBOX_SetImage()

之前	之后
 	 

描述

设置复选框被选中后显示的图像。

原型

```
void CHECKBOX_SetImage(CHECKBOX_Handle     hObj,
                       const GUI_BITMAP * pBitmap,
                       unsigned int        Index);
```

参数	描述
hObj	复选框的句柄。
pBitmap	位图的指针。
Index	（请参见 CHECKBOX_SetDefaultImage 下的表）

其他信息

图像必须填充复选框的整个内部区域。使用此函数时，要确保用于创建小工具的复选框的尺寸足以显示位图和 （可选的）文本。

CHECKBOX_SetNumStates()

描述

此函数设置给定复选框的可能状态的数量。

原型

```
void CHECKBOX_SetNumStates(CHECKBOX_Handle hObj, unsigned NumStates);
```

参数	描述
hObj	小工具的句柄。
NumStates	给定复选框的当前状态的数量。当前支持 2 或 3 种状态。

其他信息

默认情况下，复选框支持 2 种状态：选中 (1) 和未选中 (0)。如果复选框要支持第三种状态，可将可能状态增加到 3 种。

CHECKBOX_SetSpacing()

之前	之后
	

描述

设置给定复选框小工具的框和文本之间的像素数量。

原型

```
void CHECKBOX_SetSpacing(CHECKBOX_Handle hObj, unsigned Spacing);
```

参数	描述
hObj	小工具的句柄。
Spacing	框和文本之间的像素数量。

其他信息

默认间距为4像素。可使用函数CHECKBOX_SetDefaultSpacing()或配置宏CHECKBOX_SPACING_DEFAULT来设置默认值。

CHECKBOX_SetState()

之前	之后
	 

描述

设置给定复选框小工具的新状态。

原型

```
void CHECKBOX_SetState(CHECKBOX_Handle hObj, unsigned State);
```

参数	描述
hObj	小工具的句柄。
State	基于零的新状态编号。

参数 State 的允许值	
0	取消选中
1	选中
2	第三种状态

其他信息

传递的状态不得大于用 CHECKBOX_SetNumStates() 设置的可能状态数量减 1。

CHECKBOX_SetText()

之前	之后
	

描述

设置复选框旁显示的可选文本。

原型

```
void CHECKBOX_SetText(CHECKBOX_Handle hObj, const char * pText);
```

参数	描述
hObj	小工具的句柄。
pText	指向要显示在复选框旁边的文本的指针。

其他信息

点击复选框旁文本的作用与点击框内相同。

CHECKBOX_SetTextAlign()

之前	之后
	

描述

设置用于显示复选框旁边文本的对齐方式。

原型

```
void CHECKBOX_SetTextAlign(CHECKBOX_Handle hObj, int Align);
```

参数	描述
hObj	小工具的句柄。
Align	所需的文本对齐方式。

其他信息

默认情况下，文本对齐方式是 GUI_TA_LEFT | GUI_TA_VCENTER。可使用函数 CHECKBOX_SetDefaultTextAlign() 或配置宏 CHECKBOX_TEXTALIGN_DEFAULT 来设置用户定义的默认值。

CHECKBOX_SetTextColor()

之前	之后
	

描述

设置用于显示复选框文本的颜色。

原型

```
void CHECKBOX_SetTextColor(CHECKBOX_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄。
Color	所需的复选框文本颜色。

其他信息

默认情况下，复选框文本的颜色为 GUI_BLACK。可使用函数 CHECKBOX_SetDefaultTextColor() 或配置宏 CHECKBOX_TEXTCOLOR_DEFAULT 来设置用户定义的默认颜色。

CHECKBOX_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

CHECKBOX_Uncheck()

（弃用，应使用 CHECKBOX_SetState() 代替）

之前	之后
	

描述

将指定复选框的状态设置为未选中。

原型

```
void CHECKBOX_Uncheck(CHECKBOX_Handle hObj);
```

参数	描述
hObj	复选框的句柄。

其他信息

这是复选框的默认设置。

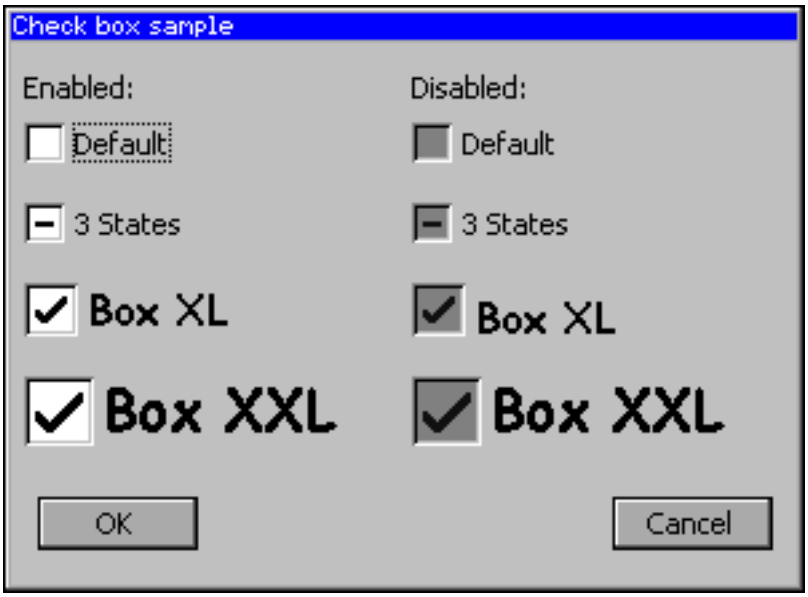
15.5.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_Checkbox.c

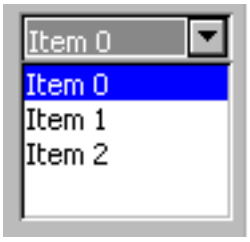
需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_Checkbox.c 的屏幕截图：



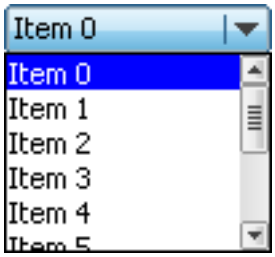
15.6 DROPDOWN：下拉列表小工具

DROPDOWN 小工具用于从具有若干栏的列表中选择一个元素，它以非打开状态显示当前选择的项目。如果用户打开 DROPDOWN 小工具，就会出现一个选择新项目的 LISTBOX。

DROPDOWN 关闭	DROPDOWN 打开
	

如果已启用鼠标支持，则打开的列表会对移过它的鼠标作出反应。

皮肤设置 ...



... 可用于该小工具。上述屏幕截图显示使用默认皮肤的小工具。有关详细信息，请参阅“皮肤设置”一章。

15.6.1 配置选项

类型	宏	默认值	描述
N	DROPDOWN_ALIGN_DEFAULT	GUI_TA_LEFT	用于显示关闭状态下拉文本的对齐方式。
S	DROPDOWN_FONT_DEFAULT	&GUI_Font13_1	默认字体
N	DROPDOWN_BKCOLOR0_DEFAULT	GUI_WHITE	背景颜色，未选定状态。
N	DROPDOWN_BKCOLOR1_DEFAULT	GUI_GRAY	背景色，选中状态， 无焦点。
N	DROPDOWN_BKCOLOR2_DEFAULT	GUI_BLUE	背景色，选中状态， 有焦点。
N	DROPDOWN_KEY_EXPAND	GUI_KEY_SPACE	可用于展开下拉列表的键。
N	DROPDOWN_KEY_SELECT	GUI_KEY_ENTER	可用于从打开的下拉列表中选择项目的键。
N	DROPDOWN_TEXTCOLOR0_DEFAULT	GUI_BLACK	文本颜色，未选定状态。
N	DROPDOWN_TEXTCOLOR1_DEFAULT	GUI_BLACK	文本颜色，选定状态，无焦点。
N	DROPDOWN_TEXTCOLOR2_DEFAULT	GUI_WHITE	启用 3D 支持。

15.6.2 通知代码

以下事件将作为 WM_NOTIFY_PARENT 消息的一部分从小工具发送至其父窗口：

消息	描述
WM_NOTIFICATION_CLICKED	已单击小工具。
WM_NOTIFICATION_RELEASED	已释放小工具。
WM_NOTIFICATION_MOVED_OUT	已单击小工具，并且指针已移出小工具，但没有释放。
WM_NOTIFICATION_SCROLL_CHANGED	已打开的下拉列表小工具的可选滚动条的滚动位置已改变。
WM_NOTIFICATION_SEL_CHANGED	下拉列表的选择已更改。

15.6.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_ENTER	从打开的下拉列表中选择项目，然后关闭列表。
GUI_KEY_SPACE	打开下拉列表。

15.6.4 DROPDOWN API

下表按字母顺序列出了可用的 emWin DROPDOWN 相关例程。这些例程的详细说明如下。

例程	描述
DROPDOWN_AddString()	给 DROPDOWN 列表添加元素。
DROPDOWN_Collapse()	关闭下拉列表。
DROPDOWN_Create()	创建 DROPDOWN 小工具。（弃用）
DROPDOWN_CreateEx()	创建 DROPDOWN 小工具。
DROPDOWN_CreateIndirect()	从资源表项创建 DROPDOWN 小工具。
DROPDOWN_CreateUser()	使用额外字节作为用户数据创建 DROPDOWN 小工具。
DROPDOWN_DecSel()	减小选定范围。
DROPDOWN_DecSelExp()	递减处于展开状态的选取项。
DROPROWN_DeleteItem()	删除 DROPDOWN 列表的项目。
DROPDOWN_Expand()	打开下拉列表。
DROPDOWN_GetDefaultFont()	返回用于创建 DROPDOWN 小工具的默认字体。
DROPDOWN_GetItemDisabled()	返回给定项目的状态。
DROPDOWN_GetListbox()	返回处于展开状态的附加 LISTBOX 的句柄。
DROPDOWN_GetNumItems()	返回下拉列表中项目的数量。

例程	描述
DROPDOWN_GetSel()	返回当前选定元素的数量。
DROPDOWN_GetSelExp()	返回处于展开状态的当前选定元素的数量。
DROPDOWN_GetUserData()	检索使用 DROPDOWN_SetUserData() 设置的数据。
DROPDOWN_IncSel()	增加选定范围。
DROPDOWN_IncSelExp()	递增处于展开状态的选取项。
DROPDOWN_InsertString()	给下拉列表插入字符串。
DROPDOWN_SetAutoScroll()	启用在下拉列表中自动使用滚动条。
DROPDOWN_SetBkColor()	设置背景颜色。
DROPDOWN_SetColor()	设置 DROPDOWN 小工具的箭头和按钮的颜色。
DROPDOWN_SetDefaultColor()	设置 DROPDOWN 小工具的箭头和按钮的默认颜色。
DROPDOWN_SetDefaultFont()	设置 DROPDOWN 小工具的默认字体。
DROPDOWN_SetDefaultScrollbarColor()	设置下拉列表中可选滚动条的默认颜色。
DROPDOWN_SetFont()	设置给定 DROPDOWN 小工具的字体。
DROPDOWN_SetItemDisabled()	设置给定项目的状态。
DROPDOWN_SetItemSpacing()	设置下拉列表中各项目之间的间距。
DROPDOWN_SetScrollbarColor()	设置下拉列表中滚动条的颜色。
DROPDOWN_SetSel()	设置当前选定内容。
DROPDOWN_SetSelExp()	设置处于展开状态的当前选择。
DROPDOWN_SetTextAlign()	设置用于显示关闭状态下拉文本的对齐方式。
DROPDOWN_SetTextColor()	设置给定 DROPDOWN 小工具的文本颜色。
DROPDOWN_SetTextHeight()	设置用于显示关闭状态下拉文本的矩形的高度。
DROPDOWN_SetUserData()	设置 DROPDOWN 小工具的额外数据。

DROPDOWN_AddString()

描述

给下拉列表添加新元素。

原型

```
void DROPDOWN_AddString(DROPDOWN_Handle hObj, const char * s);
```

参数	描述
hObj	小工具的句柄
s	要添加的字符串的指针

DROPDOWN_Collapse()

描述

关闭 DROPDOWN 小工具的下拉列表。

原型

```
void DROPDOWN_Collapse(DROPDOWN_Handle hObj);
```

参数	描述
hObj	小工具的句柄

DROPDOWN_Create()

（弃用，应使用 DROPDOWN_CreateEx() 代替）

描述

在指定位置创建指定尺寸的 DROPDOWN 小工具。

原型

```
DROPDOWN_Handle DROPDOWN_Create(WM_HWIN hWinParent,
                                int     x0,          int y0,
                                int     xsize,       int ysize,
                                int     Flags);
```

参数	描述
hWinParent	父窗口的句柄
x0	DROPDOWN 小工具的最左像素 （在父坐标中）。
y0	DROPDOWN 小工具的最上像素 （在父坐标中）。
xsize	DROPDOWN 小工具的水平尺寸 （单位：像素）。
ysize	处于打开状态的 DROPDOWN 小工具的垂直尺寸 （单位：像素）。
Flags	窗口创建标记。为使小工具立即可见，通常使用 WM_CF_SHOW （有关可用参数值的清单，请参见“WM_CreateWindow()”（第 309 页））。

返回值

已创建的 DROPDOWN 小工具的句柄；函数失败时为 0。

其他信息

该小工具处于关闭状态时的 ysize 取决于创建小工具时使用的字体。不能设置处于关闭状态的 DROPDOWN 小工具的 ysize。

DROPDOWN_CreateEx()

描述

在指定位置创建指定尺寸的 DROPDOWN 小工具。

原型

```
DROPDOWN_Handle DROPDOWN_CreateEx(int     x0,       int y0,
                                  int     xsize,    int ysize,
                                  WM_HWIN hParent,  int WinFlags,
                                  int     ExFlags,  int Id);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	处于打开状态的小工具的垂直尺寸 （单位：像素）。
hParent	父窗口的句柄。如果为 0，则新的 DROPDOWN 小工具将是桌面 （顶级窗口）的子项。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅“窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	（参见下表）
Id	小工具的窗口 ID。

参数 ExFlags 的允许值	
0	无功能。
DROPDOWN_CF_AUTOSCROLLBAR	启用自动使用滚动条。更多详细信息，请参阅“DROPDOWN_SetAutoScroll()”（第 394 页）。
DROPDOWN_CF_UP	创建一个在上方打开下拉列表的 DROPDOWN 小工具。小工具下方的空间不足以显示下拉列表时，可以使用此标记。

返回值

已创建的 DROPDOWN 小工具的句柄；函数失败时为 0。

DROPDOWN_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。

DROPDOWN_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细描述，可参见函数 DROPDOWN_CreateEx()。

DROPDOWN_DecSel()

描述

递减选取，将指定 DROPDOWN 小工具的选取项向上移动一个项目。

原型

```
void DROPDOWN_DecSel(DROPDOWN_Handle hObj);
```

参数	描述
hObj	小工具的句柄

DROPDOWN_DecSelExp()

描述

递减处于展开状态的附加 LISTBOX 的选取项。

原型

```
void DROPDOWN_DecSelExp(DROPDOWN_Handle hObj);
```

参数	描述
hObj	小工具的句柄

DROPDOWN_DeleteItem()

描述

删除下拉列表的给定项目。

原型

```
void DROPDOWN_DeleteItem(DROPDOWN_Handle hObj, unsigned int Index);
```

参数	描述
hObj	小工具的句柄。
Index	要删除的项目的以零为基准的索引。

其他信息

如果索引大于项目数量，则函数立即返回 <1。

DROPDOWN_Expand()

描述

打开小工具的下拉列表。

原型

```
void DROPDOWN_Expand(DROPDOWN_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

其他信息

下拉列表保持打开，直到已选择一个元素或者焦点丢失。

DROPDOWN_GetItemDisabled()

描述

返回给定项目的状态。

原型

```
unsigned DROPDOWN_GetItemDisabled(DROPDOWN_Handle hObj, unsigned Index);
```

参数	描述
hObj	小工具的句柄。
Index	项目的以零为基准的索引。

返回值

给定项目禁用时为 1，否则为 0。

DROPDOWN_GetListbox()

描述

返回处于展开状态的附加 LISTBOX 小工具的句柄。

原型

```
LISTBOX_Handle DROPDOWN_GetListbox(DROPDOWN_Handle hObj);
```

参数	描述
hObj	小工具的句柄

返回值

处于展开状态的附加 LISTBOX 小工具的句柄， DROPDOWN 处于折叠状态时为 0。

DROPDOWN_GetNumItems()

描述

返回给定 DROPDOWN 小工具中项目的数量。

原型

```
int DROPDOWN_GetNumItems(DROPDOWN_Handle hObj);
```

参数	描述
hObj	小工具的句柄

返回值

给定 DROPDOWN 小工具中项目的数量。

DROPDOWN_GetSel()

描述

返回指定 DROPDOWN 小工具中当前选定项目的数量。

原型

```
int DROPDOWN_GetSel(DROPDOWN_Handle hObj);
```

参数	描述
hObj	小工具的句柄

返回值

当前选定项目的数量。

DROPDOWN_GetSelExp()

描述

返回处于展开状态的附加 LISTBOX 中当前选定项目的数量。

原型

```
int DROPDOWN_GetSelExp(DROPDOWN_Handle hObj);
```

参数	描述
hObj	小工具的句柄

返回值

当前选定项目的数量。

DROPDOWN_GetUserData()

在本章的开头已按 <WIDGET>_GetUserData() 对原型进行了描述。

DROPDOWN_IncSel()

描述

递增选取，将指定 DROPDOWN 小工具的选取项向下移动一个项目。

原型

```
void DROPDOWN_IncSel(DROPDOWN_Handle hObj);
```

参数	描述
hObj	小工具的句柄

DROPDOWN_IncSelExp()

描述

递增处于展开状态的附加 LISTBOX 的选取项。

原型

```
void DROPDOWN_IncSelExp(DROPDOWN_Handle hObj);
```

参数	描述
hObj	小工具的句柄

DROPDOWN_InsertString()

描述

将字符串插入下拉列表的给定位置。

原型

```
void DROPDOWN_InsertString(DROPDOWN_Handle   hObj,
                           const char      * s,
                           unsigned int      Index);
```

参数	描述
hObj	小工具的句柄。
s	指向要插入的字符串的指针。
Index	位置的以零为基准的索引。

其他信息

如果给定索引大于项目数量，则字符串将附加到下拉列表的末端。

DROPDOWN_SetAutoScroll()

描述

启用在下拉列表中自动使用垂直滚动条。

原型

```
void DROPDOWN_SetAutoScroll(DROPDOWN_Handle hObj, int OnOff);
```

参数	描述
hObj	小工具的句柄。
OnOff	（参见下表）

参数 OnOff 的允许值	
0	禁用自动使用滚动条。
1	启用自动使用滚动条。

其他信息

启用后，下拉列表将检查是否所有元素都能放入列表框中。如果未启用，则将添加垂直滚动条。

DROPDOWN_SetBkColor()

之前	之后
	

描述

设置给定 DROPDOWN 小工具的背景色。

原型

```
void DROPDOWN_SetBkColor(DROPDOWN_Handle hObj,
                         unsigned int    Index,
                         GUI_COLOR       Color);
```

参数	描述
hObj	小工具的句柄
Index	背景颜色的索引（参见下表）。
Color	要设置的颜色。

参数 Index 的允许值	
DROPDOWN_CI_UNSEL	未选定元素。
DROPDOWN_CI_SEL	选定元素，无焦点。
DROPDOWN_CI_SELFOCUS	选定元素，有焦点。

DROPDOWN_SetColor()

之前	之后
	

描述

设置给定 DROPDOWN 小工具的按钮或箭头的颜色。

原型

```
void DROPDOWN_SetColor(DROPDOWN_Handle hObj,
                       unsigned int    Index,
                       GUI_COLOR       Color);
```

参数	描述
hObj	小工具的句柄
Index	目标项的索引（参见下表）。
Color	要使用的颜色。

参数 Index 的允许值	
DROPDOWN_CI_ARROW	按钮内小箭头的颜色。
DROPDOWN_CI_BUTTON	按钮颜色。

DROPDOWN_SetDefaultColor()

描述

设置新 DROPDOWN 小工具的箭头和按钮的默认颜色。

原型

```
GUI_COLOR DROPDOWN_SetDefaultColor(int Index, GUI_COLOR Color);
```

参数	描述
Index	请参阅 “DROPDOWN_SetColor()”（第 395 页）。
Color	要使用的颜色。

DROPDOWN_SetDefaultFont()

描述

设置新 DROPDOWN 小工具的默认字体。

原型

```
void DROPDOWN_SetDefaultFont(const GUI_FONT * pFont);
```

参数	描述
pFont	GUI_FONT 结构的指针。

DROPDOWN_SetDefaultScrollbarColor()

描述

设置下拉列表中可选滚动条的默认颜色。

原型

```
GUI_COLOR DROPDOWN_SetDefaultScrollbarColor(int Index, GUI_COLOR Color);
```

参数	描述
Index	请参阅 “DROPDOWN_SetScrollbarColor()”（第 397 页）。
Color	要使用的颜色。

DROPDOWN_SetFont()

描述

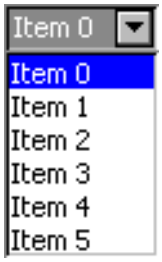
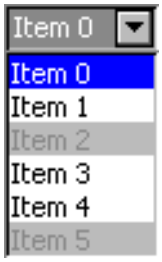
设置用于显示给定 DROPDOWN 小工具的字体。

原型

```
void DROPDOWN_SetFont(DROPDOWN_Handle hObj, const GUI_FONT * pFont);
```

参数	描述
hObj	小工具的句柄
pFont	字体的指针。

DROPDOWN_SetItemDisabled()

之前	之后
	

描述

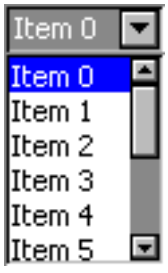
设置给定项目的启用状态。

原型

```
void DROPDOWN_SetItemDisabled(DROPDOWN_Handle hObj,
                              unsigned        Index,
                              int             OnOff);
```

参数	描述
hObj	小工具的句柄
Index	项目的以零为基准的索引。
OnOff	启用为 1，禁用为 0。

DROPDOWN_SetScrollbarColor()

之前	之后
	

描述

设置下拉列表中可选滚动条的颜色。

原型

```
void DROPDOWN_SetScrollbarColor(DROPDOWN_Handle hObj,
                                unsigned int    Index,
                                GUI_COLOR       Color);
```

参数	描述
hObj	小工具的句柄
Index	目标项的索引 （参见下表）。
Color	要使用的颜色。

参数 Index 的允许值	
SCROLLBAR_CI_THUMB	缩略图区域的颜色。
SCROLLBAR_CI_SHAFT	轴的颜色。
SCROLLBAR_CI_ARROW	箭头颜色。

DROPDOWN_SetScrollbarWidth()

描述

设置给定 DROPDOWN 小工具的下拉列表使用的滚动条的宽度。

原型

```
void DROPDOWN_SetScrollbarWidth(DROPDOWN_Handle hObj, unsigned Width);
```

参数	描述
hObj	小工具的句柄。
Width	给定 DROPDOWN 小工具的下拉列表使用的滚动条的宽度。

DROPDOWN_SetSel()

描述

设置指定 DROPDOWN 小工具的选定项目。

原型

```
void DROPDOWN_SetSel(DROPDOWN_Handle hObj, int Sel);
```

参数	描述
hObj	小工具的句柄
Sel	要选择的元素。

DROPDOWN_SetSelExp()

描述

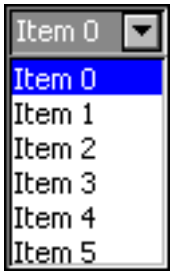
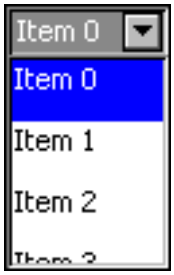
设置处于展开状态的附加 LISTBOX 的选定项。

原型

```
void DROPDOWN_SetSelExp(DROPDOWN_Handle hObj, int Sel);
```

参数	描述
hObj	小工具的句柄
Sel	要选择的元素。

DROPDOWN_SetItemSpacing()

之前	之后
	

描述

设置下拉列表项目下面的额外间距。

原型

```
void DROPDOWN_SetItemSpacing(DROPDOWN_Handle hObj, unsigned Value);
```

参数	描述
hObj	小工具的句柄
Value	项目和下拉列表之间额外间距的像素数量。

DROPDOWN_SetTextAlign()

之前	之后
	

描述

设置用于显示关闭状态下拉文本的对齐方式。

原型

```
void DROPDOWN_SetTextAlign(DROPDOWN_Handle hObj, int Align);
```

参数	描述
hObj	小工具的句柄
Align	用于显示关闭状态下拉文本的对齐方式。

DROPDOWN_SetTextColor()

描述

设置给定 DROPDOWN 小工具的背景色。

原型

```
void DROPDOWN_SetTextColor(DROPDOWN_Handle hObj,
                           unsigned int    Index,
```

```
                                    GUI_COLOR        Color);
```

参数	描述
hObj	小工具的句柄
Index	背景颜色的索引 （参见下表）。
Color	要设置的颜色。

参数 Index 的允许值	
DROPDOWN_CI_UNSEL	未选定元素。
DROPDOWN_CI_SEL	选定元素，无焦点。
DROPDOWN_CI_SELFOCUS	选定元素，有焦点。

DROPDOWN_SetTextHeight()

之前	之后
	

描述

设置用于显示关闭状态 DROPDOWN 文本的矩形的高度。

原型

```
void DROPDOWN_SetTextHeight(DROPDOWN_Handle hObj, unsigned TextHeight);
```

参数	描述
hObj	小工具的句柄
TextHeight	用于显示关闭状态文本的矩形的高度。

其他信息

默认情况下，DROPDOWN 小工具的高度取决于所使用的字体。使用此函数时，如果 TextHeight>0，则表示应使用给定值。 Text Height= 0 表示应使用默认值。

DROPDOWN_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

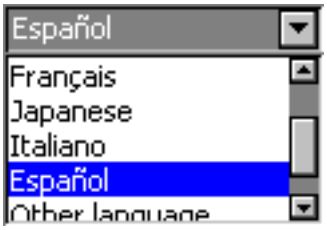
15.6.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_Dropdown.c

需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_Dropdown.c 的屏幕截图：



15.7 EDIT：“编辑”小工具

编辑字段通常用作输入文本的主要用户界面：

空白编辑字段	带用户输入的编辑字段
	

也可使用编辑字段以二进制、十进制或十六进制模式输入值。十进制模式编辑字段的外观与下表所列类似。与复选框类似，编辑字段在禁用时显示为灰色：

带用户输入的编辑字段 （十进制）	已禁用的编辑字段
	

所有 EDIT 相关例程都位于文件 EDIT*.c、 EDIT.h 中。所有标识符的前缀都是 EDIT。

15.7.1 配置选项

类型	宏	默认值	描述
S	EDIT_ALIGN_DEFAULT	GUI_TA_RIGHT | GUI_TA_VCENTER	编辑字段文本的对齐方式。
N	EDIT_BKCOLOR0_DEFAULT	0xc0c0c0	背景色，禁用状态。
N	EDIT_BKCOLOR1_DEFAULT	GUI_WHITE	背景色，启用状态。
N	EDIT_BORDER_DEFAULT	1	边框宽度 （单位：像素）。
S	EDIT_FONT_DEFAULT	&GUI_Font13_1	编辑字段文本使用的字体。
N	EDIT_TEXTCOLOR0_DEFAULT	GUI_BLACK	文本颜色，禁用状态。
N	EDIT_TEXTCOLOR1_DEFAULT	GUI_BLACK	文本颜色，启用状态。
N	EDIT_XOFF	2	在 X 方向上，文本偏离编辑字段左边界的距离。

可用的对齐标记有：
水平对齐：GUI_TA_LEFT、 GUI_TA_RIGHT、 GUI_TA_HCENTER。
垂直对齐：GUI_TA_TOP、 GUI_TA_BOTTOM、 GUI_TA_VCENTER。

15.7.2 通知代码

以下事件是编辑小工具作为 WM_NOTIFY_PARENT 消息的一部分发送给其父窗口的：

消息	描述
WM_NOTIFICATION_CLICKED	已单击小工具。
WM_NOTIFICATION_RELEASED	已释放小工具。
WM_NOTIFICATION_MOVED_OUT	已单击小工具，并且指针已移出小工具，但没有释放。
WM_NOTIFICATION_VALUE_CHANGED	编辑小工具的值 （内容）已更改。

15.7.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_UP	上升当前字符。例如，如果当前字符 （光标下的字符）为 “A”，则变为 “B”。
GUI_KEY_DOWN	下降当前字符。例如，如果当前字符为为 “B”，则变为 “A”。
GUI_KEY_RIGHT	将光标向右移动一个字符。
GUI_KEY_LEFT	将光标向左移动一个字符。
GUI_KEY_BACKSPACE	如果小工具在文本模式下操作，则删除光标前的字符。
GUI_KEY_DELETE	如果小工具在文本模式下操作，则删除当前字符。
GUI_KEY_INSERT	如果小工具在文本模式下操作，则此键在 GUI_EDIT_MODE_OVERWRITE 和 GUI_EDIT_MODE_INSERT 之间切换编辑模式。更多详细信息，请参阅 “EDIT_SetInsertMode()”（第 413 页）。

15.7.4 EDIT API

下表按字母顺序列出了可用的 emWin EDIT 相关例程。这些例程的详细说明如下。

例程	描述
EDIT_AddKey()	按键输入例程。
EDIT_Create()	创建 EDIT 小工具。（弃用）
EDIT_CreateAsChild()	创建 EDIT 小工具，作为子窗口。（弃用）
EDIT_CreateEx()	创建 EDIT 小工具。
EDIT_CreateIndirect()	从资源表项创建 EDIT 小工具。
EDIT_CreateUser()	使用额外字节作为用户数据创建 EDIT 小工具。
EDIT_EnaBleBlink()	启用 / 禁用闪烁光标
EDIT_GetCursorCharPos()	返回光标位置的字符编号。
EDIT_GetCursorPixelPos()	返回光标的像素位置。
EDIT_GetDefaultBkColor()	返回默认背景颜色。
EDIT_GetDefaultFont()	返回默认的字体。
EDIT_GetDefaultTextAlign()	返回默认的文本对齐方式。
EDIT_GetDefaultTextColor()	返回默认文本颜色。
EDIT_GetFloatValue()	按浮点值返回当前值。
EDIT_GetNumChars()	返回给定编辑小工具的字符数。
EDIT_GetText()	获取用户输入。
EDIT_GetUserData()	检索用 EDIT_SetUserData() 设置的数据。
EDIT_GetValue()	返回当前值。
EDIT_SetBinMode()	启用二进制编辑模式。
EDIT_SetBkColor()	设置编辑字段的背景色。
EDIT_SetCursorAtChar()	将编辑小工具光标设置到指定的字符位置。
EDIT_SetCursorAtPixel()	将编辑小工具光标设置到指定的像素位置。
EDIT_SetDecMode()	启用十进制编辑模式。
EDIT_SetDefaultBkColor()	设置默认背景色。
EDIT_SetDefaultFont()	设置用于编辑字段的默认字体。
EDIT_SetDefaultTextAlign()	设置编辑字段的默认文本对齐方式。
EDIT_SetDefaultTextColor()	设置默认文本颜色。
EDIT_SetFloatMode()	启用浮点编辑模式。
EDIT_SetFloatValue()	设置使用浮点编辑模式时的浮点值。
EDIT_SetFont()	选择文本的字体。
EDIT_SetHexMode()	启用十六进制编辑模式。
EDIT_SetInsertMode()	启用或禁用插入模式。
EDIT_SetMaxLen()	设置编辑字段的最大字符数。

例程	描述
EDIT_SetpfAddKeyEx()	设置添加字符时要调用的函数。
EDIT_SetSel()	设置当前选定内容。
EDIT_SetText()	设置文本。
EDIT_SetTextAlign()	设置编辑字段的文本对齐方式。
EDIT_SetTextColor()	设置文本的颜色。
EDIT_SetTextMode()	将小工具的编辑模式设置回文本模式。
EDIT_SetValue()	设置当前值。
EDIT_SetUlongMode()	启用不带符号的长十进制编辑模式。
EDIT_SetUserData()	设置 EDIT 小工具的额外数据。
GUI_EditBin()	编辑当前光标位置处的二进制值。
GUI_EditDec()	编辑当前光标位置处的十进制值。
GUI_EditHex()	编辑当前光标位置处的十六进制值。
GUI_EditString()	编辑当前光标位置处的字符串。

EDIT_AddKey()

描述

向指定的编辑字段添加用户输入。

原型

```
void EDIT_AddKey(EDIT_Handle hObj, int Key);
```

参数	描述
hObj	编辑字段的句柄。
Key	要添加的字符。

其他信息

指定字符被添加到EDIT小工具的用户输入中。如果要清除最后一个字符，可使用键GUI_KEY_BACKSPACE。如果已达到最大字符数，则不会添加其他字符。

EDIT_Create()

（弃用，应使用 EDIT_CreateEx() 代替）

描述

在指定位置创建指定尺寸的 EDIT 小工具。

原型

```
EDIT_Handle EDIT_Create(int x0, int y0,     int xsize,  int ysize,
                        int Id, int MaxLen, int Flags);
```

参数	描述
x0	编辑字段的最左像素 （在父坐标中）。
y0	编辑字段的最上像素 （在父坐标中）。
xsize	编辑字段的水平尺寸 （单位：像素）。
ysize	编辑字段的垂直尺寸 （单位：像素）。
Id	将返回的 ID。
MaxLen	最大字符数。
Flags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅“窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。

返回值

已创建的 EDIT 小工具的句柄；函数失败时为 0。

EDIT_CreateAsChild()

（弃用，应使用 EDIT_CreateEx 代替）

描述

创建 EDIT 小工具，作为子窗口。

原型

```
EDIT_Handle EDIT_CreateAsChild(int     x0,      int y0,
                               int     xsize,   int ysize,
                               WM_HWIN hParent, int Id,
                               int     Flags,   int MaxLen);
```

参数	描述
x0	编辑字段相对于父窗口的 X 位置。
y0	编辑字段相对于父窗口的 Y 位置。
xsize	编辑字段的水平尺寸 （单位：像素）。
ysize	编辑字段的垂直尺寸 （单位：像素）。
hParent	父窗口的句柄。
Id	将返回的 ID。
Flags	窗口创建标记 （请参见 EDIT_Create()）。
MaxLen	最大字符数。

返回值

已创建的 EDIT 小工具的句柄；函数失败时为 0。

EDIT_CreateEx()

描述

在指定位置创建指定尺寸的 EDIT 小工具。

原型

```
EDIT_Handle EDIT_CreateEx(int     x0,      int y0,
                          int     xsize,   int ysize,
                          WM_HWIN hParent, int WinFlags,
                          int     ExFlags, int Id,
                          int     MaxLen);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新的 EDIT 小工具将是桌面 （顶级窗口）的子项。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW（有关可用参数值的列表，请参阅“窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	未使用，保留供日后使用。
Id	小工具的窗口 ID。
MaxLen	最大字符数。

返回值

已创建的 EDIT 小工具的句柄；函数失败时为 0。

EDIT_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。以下标记可以用作以参数来进行传递的资源 Flags 元素：

允许的间接创建标记 （可以通过 “OR”操作进行组合）	
EDIT_CF_LEFT	水平对齐：左对齐
EDIT_CF_RIGHT	水平对齐：右对齐
EDIT_CF_HCENTER	水平对齐：居中对齐
EDIT_CF_TOP	垂直对齐：顶端对齐
EDIT_CF_BOTTOM	垂直对齐：底部对齐
EDIT_CF_VCENTER	垂直对齐：居中对齐

Para 元素是要显示的字符串的最大长度 / 以十进制、二进制或十六进制模式使用时为最大位数。

EDIT_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细描述，可参见函数 EDIT_CreateEx()。

EDIT_EnableBlink()

描述

启用 / 禁用闪烁光标。

原型

```
void EDIT_EnableBlink(EDIT_Handle hObj, int Period, int OnOff);
```

参数	描述
hObj	编辑字段的句柄
Period	闪烁周期
OnOff	1 表示启用闪烁； 0 表示禁用闪烁

其他信息

此函数将调用 GUI_X_GetTime()。

EDIT_GetCursorCharPos()

描述

返回光标位置的字符编号。

原型

```
int EDIT_GetCursorCharPos(EDIT_Handle hObj);
```

参数	描述
hObj	编辑字段的句柄。

返回值

光标位置的字符编号。

其他信息

无论小工具是否具有焦点，它都将返回字符位置。这就意味着，如果光标当前在小工具中不可见，将同时返回光标位置。

EDIT_GetCursorPixelPos()

描述

返回光标在窗口坐标中的像素位置。

原型

```
void EDIT_GetCursorPixelPos(EDIT_Handle hObj, int * pxPos, int * pyPos);
```

参数	描述
hObj	编辑字段的句柄。
pxPos	窗口坐标中 X 轴位置的整数变量指针。
pyPos	窗口坐标中 Y 轴位置的整数变量指针。

其他信息

无论小工具是否具有焦点，它都将返回像素位置。这就意味着，如果光标当前在小工具中不可见，将同时返回光标位置。

EDIT_GetDefaultBkColor()

描述

返回用于 EDIT 小工具的默认背景色。

原型

```
GUI_COLOR EDIT_GetDefaultBkColor(unsigned int Index);
```

参数	描述
Index	（参见下表）

参数 Index 的允许值	
0	小工具未激活时的颜色。
1	小工具激活时的颜色。

返回值

用于 EDIT 小工具的默认背景色。

EDIT_GetDefaultFont()

描述

返回 EDIT 小工具所用的默认字体。

原型

```
const GUI_FONT* EDIT_GetDefaultFont(void);
```

返回值

EDIT 小工具所用的默认字体。

EDIT_GetDefaultTextAlign()

描述

返回 EDIT 小工具所用的默认文本对齐方式。

原型

```
int EDIT_GetDefaultTextAlign(void);
```

返回值

EDIT 小工具所用的默认文本对齐方式。

EDIT_GetDefaultTextColor()

描述

返回 EDIT 小工具所用的默认文本颜色。

原型

```
GUI_COLOR EDIT_GetDefaultTextColor(unsigned int Index);
```

参数	描述
Index	必须为 0，留待将来使用。

返回值

EDIT 小工具所用的默认文本颜色。

EDIT_GetFloatValue()

描述

按浮点值返回编辑字段的当前值。

原型

```
float EDIT_GetFloatValue(EDIT_Handle hObj);
```

参数	描述
hObj	编辑字段的句柄。

返回值

当前值。

其他信息

仅在编辑字段处于浮点编辑模式时，使用此函数才有意义。

EDIT_GetNumChars

描述

返回指定编辑字段的字符数。

原型

```
int EDIT_GetNumChars(EDIT_Handle hObj);
```

参数	描述
hObj	编辑字段的句柄。

返回值

指定编辑字段的字符数。

EDIT_GetText()

描述

检索指定编辑字段的用户输入。

原型

```
void EDIT_GetText(EDIT_Handle hObj, char * sDest, int MaxLen);
```

参数	描述
hObj	编辑字段的句柄。
sDest	指向缓冲区的指针。
MaxLen	缓冲区的大小。

EDIT_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

EDIT_GetValue()

描述

返回编辑字段的当前值。当前值仅在编辑字段处于二进制、十进制或十六进制模式时有用。

原型

```
I32  EDIT_GetValue(EDIT_Handle hObj);
```

参数	描述
hObj	编辑字段的句柄。

返回值

当前值。

EDIT_SetBinMode()

描述

启用编辑字段的二进制编辑模式。可在给定范围内修改给定值。

原型

```
void EDIT_SetBinMode(EDIT_Handle hObj, U32 Value, U32 Min, U32 Max);
```

参数	描述
hObj	编辑字段的句柄。
Value	要修改的值。
Min	最小值。
Max	最大值。

EDIT_SetBkColor()

描述

设置编辑字段的背景色。

原型

```
void EDIT_SetBkColor(EDIT_Handle hObj, unsigned int Index, GUI_COLOR Color);
```

参数	描述
hObj	编辑字段的句柄。
Index	背景颜色的索引（参见下表）。
Color	要设置的颜色。

参数 Index 的允许值	
EDIT_CI_DISABLED	禁用状态。
EDIT_CI_ENABLED	启用状态。

EDIT_SetCursorAtChar()

描述

将编辑小工具光标设置到指定的字符位置。

原型

```
void EDIT_SetCursorAtChar(EDIT_Handle hObj, int xPos);
```

参数	描述
hObj	编辑字段的句柄。
xPos	要将光标设置到的字符位置。

其他信息

字符位置按如下确定：
0：第一个 （最左）字符的左边，
1：第一个和第二个字符之间，
2：第二个和第三个字符之间，
如此类推。

EDIT_SetCursorAtPixel()

描述

将编辑小工具光标设置到指定的像素位置。

原型

```
void EDIT_SetCursorAtPixel(EDIT_Handle hObj, int Pos);
```

参数	描述
hObj	编辑字段的句柄。
Pos	要将光标设置到的像素位置。

EDIT_SetDecMode()

描述

启用编辑字段的十进制编辑模式。可在给定范围内修改给定值。

原型

```
void EDIT_SetDecMode(EDIT_Handle hEdit, I32 Value, I32 Min,
                     I32         Max,   int Shift, U8  Flags);
```

参数	描述
hObj	编辑字段的句柄。
Value	要修改的值。
Min	最小值。
Max	最大值。
Shift	如果 >0，指定小数点的位置。
Flags	参见下表。

参数 Flags 的允许值 （可以通过 “OR” 操作进行组合）	
0 （默认值）	在正常模式下编辑，符号仅在 Value 为负值时显示。
GUI_EDIT_SIGNED	显示 “+” 和 “-” 号。

EDIT_SetDefaultBkColor()

描述

设置用于编辑小工具的默认背景色。

原型

```
void EDIT_SetDefaultBkColor(unsigned int Index, GUI_COLOR Color);
```

参数	描述
Index	（参见下表）
Color	要使用的颜色。

参数 Index 的允许值	
0	小工具未激活时的颜色。
1	小工具激活时的颜色。

EDIT_SetDefaultFont()

描述

设置用于编辑字段的默认字体。

原型

```
void EDIT_SetDefaultFont(const GUI_FONT * pFont);
```

参数	描述
pFont	要设置成默认值的字体指针。

EDIT_SetDefaultTextAlign()

描述

设置编辑字段的默认文本对齐方式。

原型

```
void EDIT_SetDefaultTextAlign(int Align);
```

参数	描述
Align	默认的文本对齐方式。更多详细信息，请参阅 “EDIT_SetTextAlign()”（第 415 页）。

EDIT_SetDefaultTextColor()

描述

设置编辑小工具所用的默认文本颜色。

原型

```
void EDIT_SetDefaultTextColor(unsigned int Index, GUI_COLOR Color);
```

参数	描述
Index	必须为 0，留待将来使用。
Color	要使用的颜色。

EDIT_SetFloatMode()

描述

启用编辑字段的浮点编辑模式。可在给定范围内修改给定值。

原型

```
void  EDIT_SetFloatMode(EDIT_Handle hObj, float Value, float Min,
                        float       Max,  int   Shift, U8    Flags);
```

参数	描述
hObj	编辑字段的句柄。
Value	要修改的值。
Min	最小值。
Max	最大值。
Shift	小数点后的位数。
Flags	参见下表。

参数 Flags 的允许值 （可以通过 “OR”操作进行组合）	
0 （默认值）	在正常模式下编辑，符号仅在 Value 为负值时显示。
GUI_EDIT_SIGNED	显示 “+”和 “-”号。
GUI_EDIT_SUPPRESS_LEADING_ZEROES	不显示前导零。

EDIT_SetFloatValue()

描述

使用浮点模式时，该函数可用于设置编辑字段的浮点值。

原型

```
void EDIT_SetFloatValue(EDIT_Handle hObj, float Value);
```

参数	描述
hObj	编辑字段的句柄。
Value	编辑字段的新浮点值。

其他信息

仅在编辑字段处于浮点模式时，使用此函数才有意义。此函数在文本模式中无效。未定义使用二进制、十进制或十六进制模式时的编辑字段特性。

EDIT_SetFont()

描述

设置编辑字段字体。

原型

```
void EDIT_SetFont(EDIT_Handle hObj, const GUI_FONT * pFont);
```

参数	描述
hObj	编辑字段的句柄。
pFont	字体的指针。

EDIT_SetHexMode()

描述

启用编辑字段的十六进制编辑模式。可在给定范围内修改给定值。

原型

```
void EDIT_SetHexMode(EDIT_Handle hObj, U32 Value, U32 Min, U32 Max);
```

参数	描述
hObj	编辑字段的句柄。
Value	要修改的值。
Min	最小值。
Max	最大值。

EDIT_SetInsertMode()

描述

启用或禁用编辑小工具的插入模式。

原型

```
int EDIT_SetInsertMode(EDIT_Handle hObj, int OnOff);
```

参数	描述
hObj	编辑字段的句柄。
OnOff	参见下表。

参数 OnOff 的允许值	
0	禁用插入模式。
1	启用插入模式。

返回值

返回先前的插入模式状态。

其他信息

只有编辑小工具在文本模式或任何用户定义的模式中进行操作时，使用此函数才有意义。如果在十六进制、二进制、浮点或十进制模式中，除了可更改光标的外观外，使用此函数没有任何作用。

EDIT_SetMaxLen()

描述

设置要通过给定编辑字段编辑的最大字符数。

原型

```
void EDIT_SetMaxLen(EDIT_Handle hObj, int MaxLen);
```

参数	描述
hObj	编辑字段的句柄。
MaxLen	字符数。

EDIT_SetpfAddKeyEx()

描述

设置函数指针， EDIT 小工具使用该指针来调用负责添加字符的函数。

原型

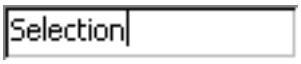
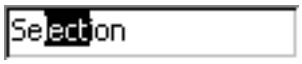
```
void EDIT_SetpfAddKeyEx(EDIT_Handle hObj, tEDIT_AddKeyEx * pfAddKeyEx);
```

参数	描述
hObj	编辑字段的句柄。
pfAddKeyEx	要用于添加字符的函数指针。

其他信息

如果处于文本模式（默认）或编辑值的模式之一，则编辑小工具使用自身的例程添加字符。仅在编辑小工具的默认特性需要更改时，使用此函数才有意义。如果已用此函数设置了函数指针，则应用程序负责文本缓冲区的内容。

EDIT_SetSel()

之前	之后
	

描述

用于设置编辑字段的当前选择。

原型

```
void EDIT_SetSel(EDIT_Handle hObj, int FirstChar, int LastChar);
```

参数	描述
hObj	编辑字段的句柄。
FirstChar	要选择的第一个字符的以零为基准的索引。不应选择字符时为 -1。
LastChar	要选择的最后一个字符的以零为基准的索引。应选择从第一个字符到最后一个字符的所有字符时为 -1。

其他信息

选定的字符通常反色显示。设置取消选择所有字符的光标位置。

示例

```
EDIT_SetSel(0, -1) /* Selects all characters of the widget */
EDIT_SetSel(-1, 0) /* Deselect all characters */
EDIT_SetSel(0, 2)  /* Selects the first 3 characters */
```

EDIT_SetText()

描述

设置要在编辑字段中显示的文本。

原型

```
void EDIT_SetText(EDIT_Handle hObj, const char* s)
```

参数	描述
hObj	编辑字段的句柄。
s	要显示的文本。

EDIT_SetTextAlign()

描述

设置编辑字段的文本对齐方式。

原型

```
void EDIT_SetTextAlign(EDIT_Handle hObj, int Align);
```

参数	描述
hObj	编辑字段的句柄。
Align	要设置的文本对齐方式 （请参见 “GUI_SetTextAlign()”（第 82 页））

EDIT_SetTextColor()

描述

设置编辑字段的文本颜色。

原型

```
void EDIT_SetTextColor(EDIT_Handle hObj,   unsigned int Index,
                       GUI_COLOR   Color);
```

参数	描述
hObj	编辑字段的句柄。
Index	文本颜色的索引 （参见下表）。
Color	要设置的颜色。

参数 OnOff 的允许值	
EDIT_CI_DISABLED	设置禁用状态的文本颜色。
EDIT_CI_ENABLED	设置启用状态的文本颜色。

EDIT_SetTextMode()

描述

将小工具的编辑模式设置回文本模式。

原型

```
void EDIT_SetTextMode(EDIT_Handle hEdit);
```

参数	描述
hObj	小工具的句柄。

其他信息

如果已经使用 EDIT_SetBinMode()、EDIT_SetDecMode()、EDIT_SetFloatMode() 或 EDIT_SetHexMode() 函数之一，将编辑字段设置为数字编辑模式之一，则此函数会将编辑模式设置回文本模式，还会清除小工具的内容。

EDIT_SetUlongMode()

描述

启用编辑字段的不带符号的长十进制编辑模式。可在给定范围内修改给定值。

原型

```
void EDIT_SetUlongMode(EDIT_Handle hEdit, U32 Value, U32 Min, U32 Max);
```

参数	描述
hObj	编辑字段的句柄。
Value	要修改的值。
Min	最小值。
Max	最大值。

EDIT_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

EDIT_SetValue()

描述

设置编辑字段的当前值。仅在设置为二进制、十进制或十六进制编辑模式时有用。

原型

```
void EDIT_SetValue(EDIT_Handle hObj, I32 Value);
```

参数	描述
hObj	编辑字段的句柄。
Value	新值。

GUI_EditBin()

描述

编辑当前光标位置处的二进制值。

原型

```
U32 GUI_EditBin(U32 Value, U32 Min, U32 Max, int Len, int xsize);
```

参数	描述
Value	要修改的值。
Min	最小值。
Max	最大值。
Len	要编辑的位数。
xsize	编辑字段 X 方向的像素大小。

返回值

按下 <ENTER> 键时将返回新值。如果按下 <ESC> 键，则返回旧值。

其他信息

按 <ENTER> 或 <ESC> 键后返回例程。仅在按下 <ENTER> 键时才会修改给定文本的内容。

GUI_EditDec()

描述

编辑当前光标位置处的十进制值。

原型

```
U32 GUI_EditDec(I32 Value, I32 Min, I32 Max, int Len, int xsize,
                int Shift, U8 Flags);
```

参数	描述
Value	要修改的值。
Min	最小值。
Max	最大值。
Len	要编辑的位数。
xsize	编辑字段 X 方向的像素大小。
Shift	如果 >0，指定小数点的位置。
Flags	请参见 EDIT_SetDecMode()。

返回值

按下 <ENTER> 键时将返回新值。如果按下 <ESC> 键，则返回旧值。

其他信息

按 <ENTER> 或 <ESC> 键后返回例程。仅在按下 <ENTER> 键时才会修改给定文本的内容。

GUI_EditHex()

描述

编辑当前光标位置处的十六进制值。

原型

```
U32 GUI_EditHex(U32 Value, U32 Min, U32 Max, int Len, int xsize);
```

参数	描述
Value	要修改的值。
Min	最小值。
Max	最大值。
Len	要编辑的位数。
xsize	编辑字段 X 方向的像素大小。

返回值

按下 <ENTER> 键时将返回新值。如果按下 <ESC> 键，则返回旧值。

其他信息

按 <ENTER> 或 <ESC> 键后返回例程。仅在按下 <ENTER> 键时才会修改给定文本的内容。

GUI_EditString()

描述

编辑当前光标位置处的字符串。

原型

```
void GUI_EditString(char * pString, int Len, int xsize);
```

参数	描述
pString	指向要编辑的字符串的指针。
Len	最大字符数。
xsize	编辑字段 X 方向的像素大小。

其他信息

按 <ENTER> 或 <ESC> 键后返回例程。仅在按下 <ENTER> 键时才会修改给定文本的内容。

15.7.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_Edit.c
- WIDGET_EditWinmode.c

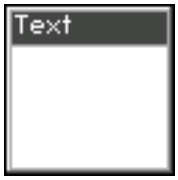
需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_Edit.c 的屏幕截图：

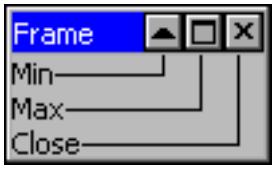
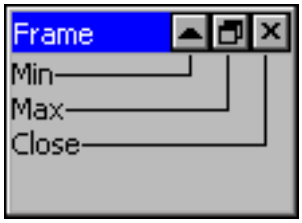
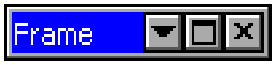
WIDGET_EditWinmode.c 的屏幕截图：

15.8 FRAMEWIN：框架窗口小工具

框架窗口为您的应用提供一个 PC 应用程序的窗口外观。这些窗口由周围框架、标题栏和用户区组成。标题栏的颜色改变以显示窗口是否激活，如下图所示：

激活的框架窗口	未激活的框架窗口
	

如下图所示，可将预定义的按钮附加到标题栏，也可将自己的按钮附加到标题栏：

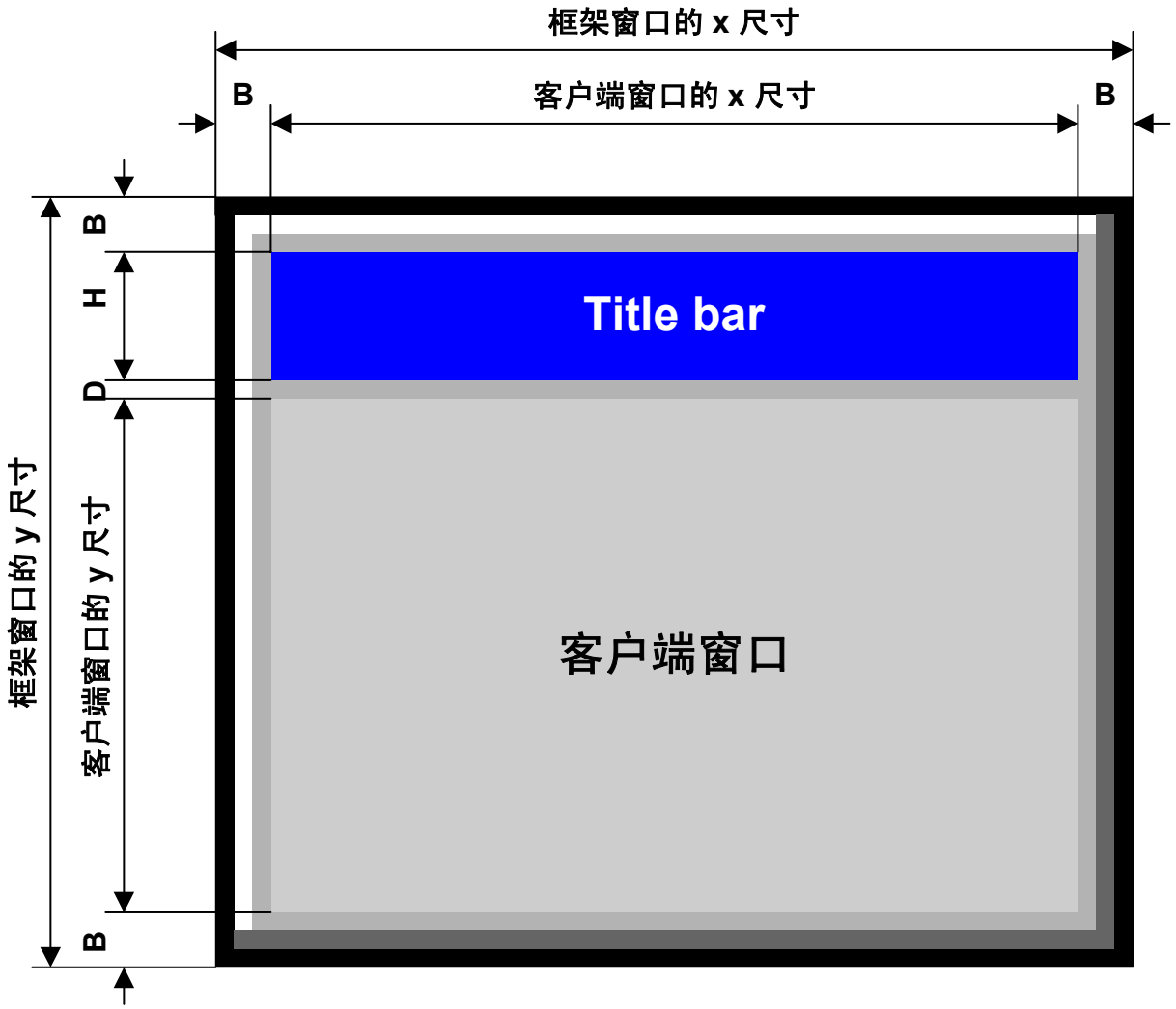
描述	框架窗口
具有 “最小化”、“最大化”和 “关闭”按钮的框架窗口。	
处于最大化状态，具有 “最小化”、“最大化”和 “关闭”按钮的框架窗口。	
处于最小化状态，具有 “最小化”、“最大化”和 “关闭”按钮的框架窗口。	

皮肤设置 ...

... 可用于该小工具。上述屏幕截图显示使用默认皮肤的小工具。有关详细信息，请参阅 “皮肤设置”一章。

15.8.1 框架窗口的结构

下图显示了框架窗口的详细结构和外观：



框架窗口实际上由 2 种窗口组成；主窗口和子窗口。子窗口称为客户端窗口。处理回调函数时意识到此点非常重要：存在具有 2 种不同回调函数的 2 种窗口。创建子窗口时，这些子窗口通常作为客户端窗口的子窗口而创建；它们的父窗口因此是客户端窗口。

细节	描述
B	框架窗口的边框尺寸。默认的边框尺寸为 3 像素。
H	标题栏的高度。取决于标题所用字体的尺寸。
D	标题栏和客户端窗口的间距。（1 像素）
标题栏	标题栏是框架窗口的一部分，不是单独的窗口。
客户端窗口	客户端窗口是一个单独的窗口，作为框架窗口的子窗口创建。

15.8.2 配置选项

类型	宏	默认值	描述
B	FRAMEWIN_ALLOW_DRAG_ON_FRAME	1	允许在周围框架上拖动小工具。
N	FRAMEWIN_BARCOLOR_ACTIVE_DEFAULT	0xff0000	标题栏颜色，激活状态。
N	FRAMEWIN_BARCOLOR_INACTIVE_DEFAULT	0x404040	标题栏颜色，未激活状态。
N	FRAMEWIN_BORDER_DEFAULT	3	外边框宽度 （单位：像素）。
N	FRAMEWIN_CLIENTCOLOR_DEFAULT	0xc0c0c0	客户端窗口区域的颜色。
S	FRAMEWIN_DEFAULT_FONT	&GUI_Font8_1	标题栏文本所用的字体。
N	FRAMEWIN_FRAMECOLOR_DEFAULT	0xaaaaaa	框架颜色。
N	FRAMEWIN_IBORDER_DEFAULT	1	内边框宽度 （单位：像素）。
N	FRAMEWIN_TITLEHEIGHT_DEFAULT	0	标题栏的默认高度。

15.8.3 键盘反应

该小工具无法获取输入焦点，并且对键盘输入无任何反应。

15.8.4 FRAMEWIN API

下表按字母顺序列出了可用的 emWin HEADER 相关例程。这些例程的详细说明如下。

例程	描述
FRAMEWIN_AddButton()	在标题栏中添加一个按钮。
FRAMEWIN_AddCloseButton()	在标题栏中添加一个 “关闭” 按钮。
FRAMEWIN_AddMaxButton()	在标题栏中添加一个 “最大化” 按钮。
FRAMEWIN_AddMenu()	向框架窗口添加一个 “菜单” 小工具。
FRAMEWIN_AddMinButton()	在标题栏中添加一个 “最小化” 按钮。
FRAMEWIN_Create()	创建一个 FRAMEWIN 小工具。（弃用）
FRAMEWIN_CreateAsChild()	创建一个 FRAMEWIN 小工具作为子窗口。（弃用）
FRAMEWIN_CreateEx()	创建一个 FRAMEWIN 小工具。
FRAMEWIN_CreateIndirect()	从资源表项创建 BUTTON 小工具。
FRAMEWIN_CreateUser()	使用额外字节作为用户数据创建一个 FRAMEWIN 小工具。
FRAMEWIN_GetActive()	返回框架窗口是否处于激活状态。
FRAMEWIN_GetBarColor()	返回标题栏的颜色。
FRAMEWIN_GetBorderSize()	返回边框的尺寸。
FRAMEWIN_GetDefaultBarColor()	返回标题栏的默认颜色。
FRAMEWIN_GetDefaultBorderSize()	返回默认的边框尺寸
FRAMEWIN_GetDefaultClientColor()	返回客户区的默认颜色。
FRAMEWIN_GetDefaultFont()	返回标题栏使用的默认字体
FRAMEWIN_GetDefaultTextColor()	返回标题的默认文本颜色。
FRAMEWIN_GetDefaultTitleHeight()	返回标题栏的默认尺寸
FRAMEWIN_GetFont()	返回标题文本所用的字体。
FRAMEWIN_GetText()	返回标题文本。
FRAMEWIN_GetTextAlign()	返回标题文本的对齐方式。
FRAMEWIN_GetTitleHeight()	返回标题栏的高度。
FRAMEWIN_GetUserData()	检索用 FRAMEWIN_SetUserData() 设置的数据。
FRAMEWIN_IsMinimized()	返回表示框架窗口是否最小化的值。
FRAMEWIN_IsMaximized()	返回表示框架窗口是否最大化的值。
FRAMEWIN_Maximize()	将框架窗口放大到父窗口的尺寸。
FRAMEWIN_Minimize()	隐藏框架窗口的客户区。
FRAMEWIN_OwnerDraw()	用于绘制标题栏的默认函数。
FRAMEWIN_Restore()	恢复最小化或最大化的框架窗口。
FRAMEWIN_SetActive()	设置框架窗口的状态。（弃用）

例程	描述
FRAMEWIN_SetBarColor()	设置标题栏的颜色。
FRAMEWIN_SetBorderSize()	设置框架窗口的边框尺寸。
FRAMEWIN_SetClientColor()	设置客户区的颜色
FRAMEWIN_SetDefaultBarColor()	设置标题栏的默认颜色
FRAMEWIN_SetDefaultBorderSize()	设置默认的边框尺寸
FRAMEWIN_SetDefaultClientColor()	设置客户区的默认颜色。
FRAMEWIN_SetDefaultFont()	设置标题栏使用的默认字体。
FRAMEWIN_SetDefaultTextColor()	设置标题的默认文本颜色。
FRAMEWIN_SetDefaultTitleHeight()	设置标题栏的默认高度
FRAMEWIN_SetFont()	选择标题文本所用的字体。
FRAMEWIN_SetMoveable()	将框架窗口设置为可移动 / 不可移动状态。
FRAMEWIN_SetOwnerDraw()	启用要自画的框架窗口。
FRAMEWIN_SetResizeable()	将框架窗口设置为可缩放状态。
FRAMEWIN_SetText()	设置标题文本。
FRAMEWIN_SetTextAlign()	设置标题文本的对齐方式。
FRAMEWIN_SetTextColor()	设置标题文本的颜色。
FRAMEWIN_SetTextColorEx()	设置标题文本的颜色。
FRAMEWIN_SetTitleHeight()	设置标题栏的高度。
FRAMEWIN_SetTitleVis()	设置标题栏的可见性标记
FRAMEWIN_SetUserData()	设置 FRAMEWIN 小工具的额外数据。

FRAMEWIN_AddButton()

之前	之后
	

描述

将按钮添加到框架窗口的标题栏。

原型

```
WM_HWIN FRAMEWIN_AddButton(FRAMEWIN_Handle hObj, int Flags,
                           int              Off,  int Id);
```

参数	描述
hObj	框架窗口的句柄。
Flags	（参见下表）
Off	用于创建 BUTTON 小工具的 X 偏移
Id	BUTTON 小工具的 ID

参数 Flags 的允许值	
0	BUTTON 将创建在左侧。
FRAMEWIN_BUTTON_RIGHT	BUTTON 将创建在右侧。

返回值

BUTTON 小工具的句柄。

其他信息

按钮将作为框架窗口的子窗口创建，因此窗口管理器确保在框架窗口删除时按钮也会被删除。
按钮可创建在标题栏的左侧或右侧，取决于参数 Flags。参数 Offset 指定按钮与框架窗口的边框之间的间距，或与先前创建的按钮之间的间距。

FRAMEWIN_AddCloseButton()

之前	之后
	

描述

将 “关闭”按钮添加到框架窗口的标题栏。

原型

```
WM_HWIN FRAMEWIN_AddCloseButton(FRAMEWIN_Handle hObj, int Flags, int Off);
```

参数	描述
hObj	框架窗口的句柄。
Flags	（参见下表）
Off	用于创建 BUTTON 小工具的 X 偏移

参数 Index 的允许值	
0	BUTTON 将创建在左侧。
FRAMEWIN_BUTTON_RIGHT	BUTTON 将创建在右侧。

返回值

“关闭”按钮的句柄。

其他信息

用户按下 “关闭”按钮时，框架窗口及其所有子窗口都将被删除。

FRAMEWIN_AddMaxButton()

之前	之后	最大化
		

描述

将 “最大化” 按钮添加到框架窗口的标题栏。

原型

```
WM_HWIN FRAMEWIN_AddMaxButton(FRAMEWIN_Handle hObj, int Flags, int Off);
```

参数	描述
hObj	框架窗口的句柄。
Flags	（参见下表）
Off	用于创建 BUTTON 小工具的 X 偏移

参数 Index 的允许值	
0	BUTTON 将创建在左侧。
FRAMEWIN_BUTTON_RIGHT	BUTTON 将创建在右侧。

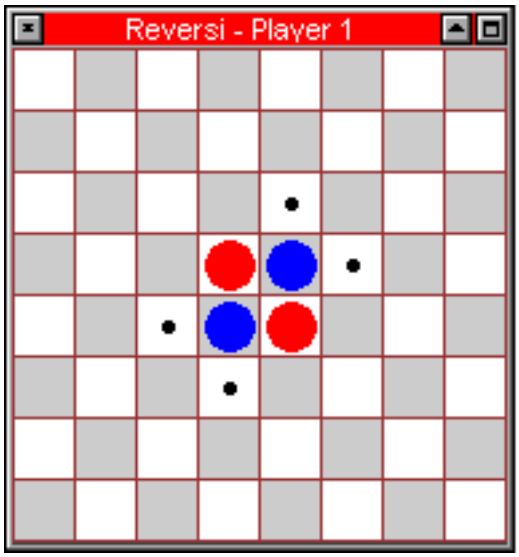
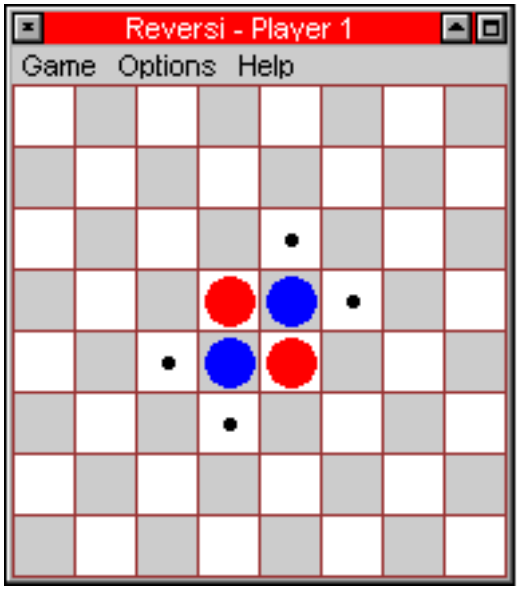
返回值

“最大化” 按钮的句柄。

其他信息

用户第一次按 “最大化” 按钮时，框架窗口将放大到父窗口的尺寸。第二次按该按钮时，将框架窗口缩小到原来尺寸并恢复到原来位置。

FRAMEWIN_AddMenu()

之前	之后
	

描述

给框架窗口添加给定菜单。菜单显示在标题栏下。

原型

```
void FRAMEWIN_AddMenu(FRAMEWIN_Handle hObj, WM_HWIN hMenu);
```

参数	描述
hObj	框架窗口的句柄。
hMenu	要添加的菜单小工具的句柄。

其他信息

添加的菜单作为框架窗口的子项附加。如果已用回调例程创建了框架窗口，则该函数确保将 WM_MENU 消息传递到框架窗口的客户端窗口。

FRAMEWIN_AddMinButton()

之前	之后	最小化窗口
		

描述

将 “最小化”按钮添加到框架窗口的标题栏。

原型

```
WM_HWIN FRAMEWIN_AddMinButton(FRAMEWIN_Handle hObj, int Flags, int Off);
```

参数	描述
hObj	框架窗口的句柄。
Flags	（参见下表）
Off	用于创建 BUTTON 小工具的 X 偏移

参数 Index 的允许值	
0	BUTTON 将创建在左侧。
FRAMEWIN_BUTTON_RIGHT	BUTTON 将创建在右侧。

返回值

“最小化”按钮的句柄。

其他信息

用户第一次按 “最小化”按钮时，框架窗口的客户区将隐藏，只有标题栏可见。第二次按该按钮时，框架窗口将恢复到原来尺寸。

FRAMEWIN_Create()

（弃用，应使用 FRAMEWIN_CreateEx() 代替）

描述

在指定位置创建指定尺寸的 FRAMEWIN 小工具。

原型

```
FRAMEWIN_Handle FRAMEWIN_Create(const char * pTitle, WM_CALLBACK * cb,
                                int          Flags,
                                int          x0,     int          y0,
                                int          xsize,  int          ysize);
```

参数	描述
pTitle	显示在标题栏中的标题。
cb	指向客户区回调例程的指针。
Flags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
x0	框架窗口的最左像素 （在父坐标中）。
y0	框架窗口的最上像素 （在父坐标中）。
xsize	框架窗口的水平尺寸 （单位：像素）。
ysize	框架窗口的垂直尺寸 （单位：像素）。

返回值

已创建的 FRAMEWIN 小工具的句柄；函数失败时为 0。

FRAMEWIN_CreateAsChild()

（弃用，应使用 FRAMEWIN_CreateEx 代替）

描述

创建一个 FRAMEWIN 小工具作为子窗口。

原型

```
FRAMEWIN_Handle FRAMEWIN_CreateAsChild(int          x0,      int y0,
                                       int          xsize,   int ysize,
                                       WM_HWIN      hParent,
                                       const char * pText,
```



```
                                        WM_CALLBACK * cb,      int Flags);
```

参数	描述
x0	框架窗口相对于父窗口的 X 位置。
y0	框架窗口相对于父窗口的 Y 位置。
xsize	框架窗口的水平尺寸 （单位：像素）。
ysize	框架窗口的垂直尺寸 （单位：像素）。
hParent	父窗口的句柄。
pText	要在标题栏中显示的文本。
cb	指向回调例程的指针。
Flags	窗口创建标记 （请参见 FRAMEWIN_Create()）。

返回值

已创建的 FRAMEWIN 小工具的句柄；函数失败时为 0。

FRAMEWIN_CreateEx()

描述

在指定位置创建指定尺寸的 FRAMEWIN 小工具。

原型

```
FRAMEWIN_Handle FRAMEWIN_CreateEx(int           x0,       int y0,
                                  int           xsize,    int ysize,
                                  WM_HWIN       hParent,  int WinFlags,
                                  int           ExFlags,  int Id,
                                  const char  * pTitle,
                                  WM_CALLBACK * cb);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新的 FRAMEWIN 小工具将是桌面 （顶级窗口）的子项。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW（有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	（参见下表）
Id	小工具的窗口 ID。
pTitle	显示在标题栏中的标题。
cb	指向用户回调例程的指针。此用户回调例程由客户端窗口的窗口回调例程调用。

参数 ExFlags 的允许值	
0	无功能。
FRAMEWIN_CF_MOVEABLE	将新框架窗口设置为可移动状态。更多详细信息，请参阅 “FRAMEWIN_SetMoveable()”（第 437 页）。

返回值

已创建的 FRAMEWIN 小工具的句柄；函数失败时为 0。

其他信息

用户回调例程通常用于 2 个目的：

- 着色客户端窗口 （在填充了不想要的颜色时）
- 对子窗口 （通常是对话框元素）的消息作出反应

客户端窗口的正常操作是自我着色，用客户端颜色填充整个窗口。
如果用户回调也填充客户端窗口 （或部分窗口），最好是将客户端颜色设置为 GUI_INVALID_COLOR，使窗口回调不填充客户端窗口。
客户端窗口的用户回调不会接收发送到客户端窗口的所有消息；一些系统消息完全由窗口回调例程处理，并且不会传递给用户回调。所有通知消息以及 WM_PAINT 和所有用户消息都被发送到用户回调例程。
用户回调所接收的句柄是框架窗口（客户端窗口的父窗口）的句柄，接收客户端窗口句柄的 WM_PAINT 消息除外。

FRAMEWIN_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。
资源的元素 Flags 和 Para 将省略，因为未使用参数。

FRAMEWIN_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细描述，可参见函数 FRAMEWIN_CreateEx()。

FRAMEWIN_GetActive()

描述

返回表示给定框架窗口是否处于激活状态的值。

原型

```
GUI_COLOR FRAMEWIN_GetBarColor(FRAMEWIN_Handle hObj, unsigned Index);
```

参数	描述
hObj	框架窗口的句柄。

返回值

框架窗口处于激活状态时为 1，否则为 0。

FRAMEWIN_GetBarColor()

描述

返回给定框架窗口标题栏的颜色。

原型

```
GUI_COLOR FRAMEWIN_GetBarColor(FRAMEWIN_Handle hObj, unsigned Index);
```

参数	描述
hObj	框架窗口的句柄。
Index	（参见下表）

参数 Index 的允许值	
0	返回框架窗口未激活时标题栏的颜色。
1	返回框架窗口激活时标题栏的颜色。

返回值

标题栏的颜色以 RGB 值表示。

FRAMEWIN_GetBorderSize()

描述

返回给定框架窗口的边框尺寸。

原型

```
int FRAMEWIN_GetBorderSize(FRAMEWIN_Handle hObj);
```

参数	描述
hObj	框架窗口的句柄。

返回值

给定框架窗口的边框尺寸。

FRAMEWIN_GetDefaultBarColor()

描述

返回框架窗口标题栏的默认颜色。

原型

```
const GUI_COLOR* FRAMEWIN_GetDefaultBarColor(unsigned int Index);
```

参数	描述
Index	（参见下表）

参数 Index 的允许值	
0	返回框架窗口未激活时标题栏的颜色。
1	返回框架窗口激活时标题栏的颜色。

返回值

指向用于指定状态框架窗口默认标题栏颜色的指针。

FRAMEWIN_GetDefaultBorderSize()

描述

返回框架窗口边框的默认尺寸。

原型

```
int FRAMEWIN_GetDefaultBorderSize(void);
```

返回值

框架窗口边框的默认尺寸。

FRAMEWIN_GetDefaultClientColor()

描述

返回框架窗口中客户区的默认颜色。

原型

```
const GUI_COLOR* FRAMEWIN_GetDefaultClientColor(void);
```

返回值

指向默认客户区颜色的指针。

FRAMEWIN_GetDefaultFont()

描述

返回框架窗口标题所用的默认字体。

原型

```
const GUI_FONT* FRAMEWIN_GetDefaultFont(void);
```

返回值

指向框架窗口标题所用默认字体的指针。

FRAMEWIN_GetDefaultTextColor()

描述

返回标题的默认文本颜色。

原型

```
GUI_COLOR FRAMEWIN_GetDefaultTextColor(unsigned Index);
```

参数	描述
Index	（参见下表）

参数 Index 的允许值	
0	框架窗口未激活时使用的颜色。
1	框架窗口激活时使用的颜色。

返回值

标题的默认文本颜色。

FRAMEWIN_GetFont()

描述

返回指向标题文本所用字体的指针。

原型

```
const GUI_FONT GUI_UNI_PTR * FRAMEWIN_GetFont(FRAMEWIN_Handle hObj);
```

参数	描述
hObj	框架窗口的句柄。

返回值

指向标题文本所用字体的指针。

FRAMEWIN_GetText()

描述

返回标题文本。

原型

```
void FRAMEWIN_GetText(FRAMEWIN_Handle hObj, char * pBuffer, int MaxLen);
```

参数	描述
hObj	框架窗口的句柄。
pBuffer	指向要填充标题文本的缓冲区的指针。
MaxLen	缓冲区大小 （以字节为单位）。

其他信息

如果缓冲区大小小于标题文本，则该函数复制 MaxLen。

FRAMEWIN_GetTextAlign()

描述

返回标题文本的文本对齐方式。

原型

```
int FRAMEWIN_GetTextAlign(FRAMEWIN_Handle hObj);
```

参数	描述
hObj	框架窗口的句柄。

返回值

当前所用的文本对齐方式。有关文本对齐方式的详细信息，请参阅“GUI_SetTextAlign()”（第82页）。

FRAMEWIN_GetDefaultTitleHeight()

描述

返回框架窗口标题栏的默认高度。

原型

```
int FRAMEWIN_GetDefaultCaptionSize(void);
```

返回值

默认标题栏高度。有关标题高度的详细信息，请参见 “FRAMEWIN_SetDefaultTitleHeight()”（第 437 页）。

FRAMEWIN_GetTitleHeight()

描述

返回给定框架窗口标题栏的高度。

原型

```
int FRAMEWIN_GetTitleHeight(FRAMEWIN_Handle hObj);
```

参数	描述
hObj	框架窗口的句柄。

返回值

给定框架窗口标题栏的高度。有关标题高度的详细信息，请参见“FRAMEWIN_SetDefaultTitleHeight()”（第 437 页）。

FRAMEWIN_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

FRAMEWIN_IsMaximized()

描述

返回表示框架窗口是否最大化的值。

原型

```
int FRAMEWIN_IsMaximized(FRAMEWIN_Handle hObj);
```

参数	描述
hObj	框架窗口的句柄。

返回值

框架窗口最大化时为 1，否则为 0。

FRAMEWIN_IsMinimized()

描述

返回表示框架窗口是否最小化的值。

原型

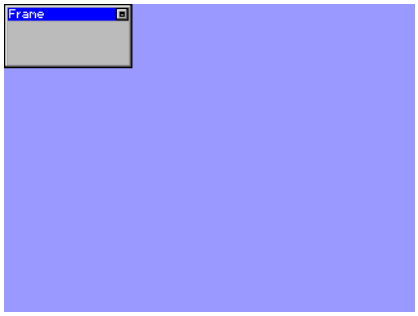
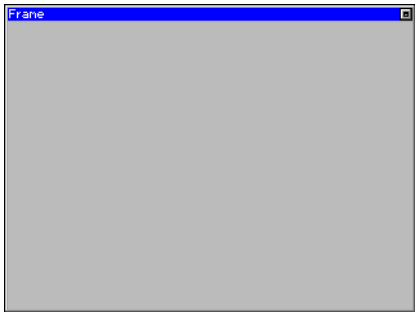
```
int FRAMEWIN_IsMinimized(FRAMEWIN_Handle hObj);
```

参数	描述
hObj	框架窗口的句柄。

返回值

框架窗口最小化时为 1，否则为 0。

FRAMEWIN_Maximize()

之前	之后
	

描述

将框架窗口放大到父窗口的尺寸。

原型

```
void FRAMEWIN_Maximize(FRAMEWIN_Handle hObj);
```

参数	描述
hObj	框架窗口的句柄。

其他信息

调用此函数时，框架窗口的操作与用户按 “最大化”按钮时相同。框架窗口将放大到父窗口尺寸。

FRAMEWIN_Minimize()

之前	之后
	

描述

隐藏给定框架窗口的客户区。

原型

```
void FRAMEWIN_Minimize(FRAMEWIN_Handle hObj);
```

参数	描述
hObj	框架窗口的句柄。

其他信息

调用此函数时，框架窗口的操作与用户按“最小化”按钮时相同。框架窗口的客户区将隐藏，只有标题栏可见。

FRAMEWIN_OwnerDraw()

描述

绘制框架窗口标题栏的默认函数。

原型

```
int FRAMEWIN_OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

参数	描述
pDrawItemInfo	WIDGET_ITEM_DRAW_INFO 结构的指针。

其他信息

此函数在使用 FRAMEWIN_SetOwnerDraw() 时有用。对于所有传递给自画函数的未处理命令，都应调用它。有关详细信息，请参见自画小工具和 FRAMEWIN_SetOwnerDraw() 的描述部分。

FRAMEWIN_Restore()

之前	之后
	

描述

将最小化或最大化的框架窗口恢复到原有尺寸和位置。

原型

```
void FRAMEWIN_Restore(FRAMEWIN_Handle hObj);
```

参数	描述
hObj	框架窗口的句柄。

其他信息

如果给定的窗口既没最大化也没最小化，则函数不起作用。

FRAMEWIN_SetActive()

之前	之后
	

描述

设置指定框架窗口的状态。标题栏的颜色将根据状态改变。

原型

```
void FRAMEWIN_SetActive(FRAMEWIN_Handle hObj, int State);
```

参数	描述
hObj	框架窗口的句柄。
State	框架窗口的状态 （见下表）。

参数 State 的允许值	
0	框架窗口未激活。
1	框架窗口激活。

其他信息

此函数弃用。如果输入设备指向框架窗口的子项，则框架窗口将自动激活。不建议使用此函数。如果使用此函数将框架窗口置于激活状态，则不能保证它在其他框架窗口激活时变成未激活。

FRAMEWIN_SetBarColor()

之前	之后
	

描述

设置指定框架窗口标题栏的颜色。

原型

```
void FRAMEWIN_SetBarColor(FRAMEWIN_Handle hObj,   unsigned int Index,
                          GUI_COLOR       Color);
```

参数	描述
hObj	框架窗口的句柄。
Index	框架窗口状态的索引 （见下表）。
Color	要设置的颜色。

参数 Index 的允许值	
0	设置框架窗口未激活时使用的颜色。
1	设置框架窗口激活时使用的颜色。

FRAMEWIN_SetBorderSize()

之前	之后
	

描述

设置指定框架窗口的边框尺寸。

原型

```
void FRAMEWIN_SetBorderSize(FRAMEWIN_Handle hObj, unsigned Size);
```

参数	描述
hObj	框架窗口的句柄。
Size	框架窗口的新边框尺寸。

FRAMEWIN_SetClientColor()

之前	之后
	

描述

设置指定框架窗口客户端窗口区域的颜色。

原型

```
void FRAMEWIN_SetClientColor(FRAMEWIN_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	框架窗口的句柄。
Color	要设置的颜色。

FRAMEWIN_SetDefaultBarColor()

描述

设置框架窗口标题栏的默认颜色。

原型

```
void FRAMEWIN_SetDefaultBarColor(unsigned int Index, GUI_COLOR Color);
```

参数	描述
Index	框架窗口状态的索引 （见下表）。
Color	要设置的颜色。

参数 Index 的允许值	
0	设置框架窗口未激活时使用的颜色。
1	设置框架窗口激活时使用的颜色。

FRAMEWIN_SetDefaultBorderSize()

描述

设置框架窗口的默认边框尺寸。

原型

```
void FRAMEWIN_SetDefaultBorderSize(int BorderSize);
```

参数	描述
BorderSize	要设置的尺寸。

FRAMEWIN_SetDefaultClientColor()

描述

设置框架窗口中客户区的默认颜色。

原型

```
void FRAMEWIN_SetDefaultClientColor(GUI_COLOR Color);
```

参数	描述
Color	要设置的颜色。

FRAMEWIN_SetDefaultFont()

描述

设置在框架窗口中显示标题的默认字体。

原型

```
void FRAMEWIN_SetDefaultFont(const GUI_FONT * pFont);
```

参数	描述
pFont	指向要用作默认字体的指针。

FRAMEWIN_SetDefaultTextColor()

描述

设置标题的默认文本颜色。

原型

```
void FRAMEWIN_SetDefaultTextColor(unsigned Index, GUI_COLOR Color);
```

参数	描述
Index	（参见下表）
Color	要使用的颜色。

参数 Index 的允许值	
0	框架窗口未激活时使用的颜色。
1	框架窗口激活时使用的颜色。

FRAMEWIN_SetDefaultTitleHeight()

描述

设置标题栏 Y 方向的尺寸。

原型

```
void FRAMEWIN_SetDefaultTitleHeight(int Height);
```

参数	描述
Height	要设置的尺寸

其他信息

标题高度的默认值为 0，这意味着标题的高度取决于显示标题文本所用的字体。如果默认值设置为 >0，则每个新框架窗口都将使用此高度 （而非标题的字体高度）作为标题高度。

FRAMEWIN_SetFont()

之前	之后
	

描述

设置标题字体。

原型

```
void FRAMEWIN_SetFont(FRAMEWIN_Handle hObj, const GUI_FONT * pfont);
```

参数	描述
hObj	框架窗口的句柄。
pFont	字体的指针。

FRAMEWIN_SetMoveable()

描述

将框架窗口设置为可移动或固定状态。

原型

```
void FRAMEWIN_SetMoveable(FRAMEWIN_Handle hObj, int State);
```

参数	描述
hObj	框架窗口的句柄。
State	框架窗口的状态 （见下表）。

参数 State 的允许值	
0	框架窗口固定 （不可移动）。
1	框架窗口可移动。

其他信息

框架窗口创建后的默认状态为固定。
可移动状态意味着可用指针输入设备 (PID) 拖动框架窗口。要移动框架窗口，首先需用处于按下状态的 PID 在标题区域中接触该窗口。此时，移动按下的 PID 即可移动小工具。
如果配置宏 FRAMEWIN_ALLOW_DRAG_ON_FRAME 为 1（默认值），则框架窗口也可在周围框架上拖动。此功能仅在框架窗口不处于可缩放状态时才有效。

FRAMEWIN_SetOwnerDraw()

描述

启用要自画的框架窗口。

原型

```
void FRAMEWIN_SetOwnerDraw(FRAMEWIN_Handle          hObj,
                           WIDGET_DRAW_ITEM_FUNC * pfDrawItem);
```

参数	描述
hObj	框架窗口的句柄。
pfDrawItem	自绘函数的指针。

其他信息

则此函数设置在必须绘制框架窗口时，指向小工具将调用的函数的指针。这样就可以绘制一个完全定制的标题栏，而非仅纯机器码。 pfDrawItem 是一个指向类型为 WIDGET_DRAW_ITEM_FUNC 的应用程序定义函数的指针，该类型已在本章的开头描述。

示例

以下是一个典型的自画函数：

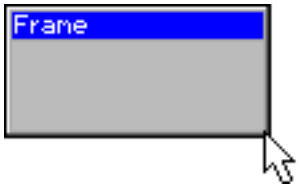
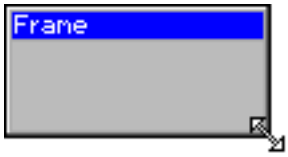
```
int _OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  GUI_RECT Rect;
  char acBuffer[20];
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_DRAW:
    Rect.x0 = pDrawItemInfo->x0 + 1;
    Rect.x1 = pDrawItemInfo->x1 - 1;
    Rect.y0 = pDrawItemInfo->y0 + 1;
    Rect.y1 = pDrawItemInfo->y1;
    FRAMEWIN_GetText(pDrawItemInfo->hWin, acBuffer, sizeof(acBuffer));
    GUI_DrawGradientH(pDrawItemInfo->x0, pDrawItemInfo->y0,
                      pDrawItemInfo->x1, pDrawItemInfo->y1,
                      GUI_RED, GUI_GREEN);
    GUI_SetFont(FRAMEWIN_GetFont(pDrawItemInfo->hWin));
    GUI_SetTextMode(GUI_TM_TRANS);
    GUI_SetColor(GUI_YELLOW);
    GUI_DispStringInRect(acBuffer, &Rect,
                         FRAMEWIN_GetTextAlign(pDrawItemInfo->hWin));
    return 0;
  }
```



```
  return FRAMEWIN_OwnerDraw(pDrawItemInfo);
}

void CreateFrameWindow(void) {
  ...
  FRAMEWIN_SetOwnerDraw(hWin, _OwnerDraw);
  ...
}
```

以上示例的屏幕截图

FRAMEWIN_SetResizeable()

之前	之后
	

描述

设置给定框架窗口的可缩放状态。

原型

```
void FRAMEWIN_SetResizeable(FRAMEWIN_Handle hObj, int State);
```

参数	描述
hObj	框架窗口的句柄。
State	框架窗口可缩放时为 1，否则为 0。

其他信息

如果框架窗口处于可缩放状态，则可通过拖动边框改变其大小。如果指针输入设备在边框上移动，光标将变成调整大小光标 （如果光标打开并且启用了可选的鼠标支持）。
如果指向边框的边缘，则窗口的 X 和 Y 尺寸可同时改变。

FRAMEWIN_SetText()

之前	之后
	

描述

设置标题文本。

原型

```
void FRAMEWIN_SetText(FRAMEWIN_Handle hObj, const char * s);
```

参数	描述
hObj	框架窗口的句柄。
s	要显示的标题文本。

FRAMEWIN_SetTextAlign()

之前	之后
	

描述

设置标题栏的文本对齐方式。

原型

```
void FRAMEWIN_SetTextAlign(FRAMEWIN_Handle hObj, int Align);
```

参数	描述
hObj	框架窗口的句柄。
Align	标题的对齐属性 （见下表）。

参数 Align 的允许值	
GUI_TA_HCENTER	居中标题 （默认）。
GUI_TA_LEFT	标题显示到左侧。
GUI_TA_RIGHT	标题显示到右侧。

其他信息

如果不调用此函数，则默认行为将是居中显示文本。

FRAMEWIN_SetTextColor()

之前	之后
	

描述

设置激活和未激活两种状态下标题文本的颜色。

原型

```
void FRAMEWIN_SetTextColor(FRAMEWIN_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	框架窗口的句柄。
Color	要设置的颜色。

FRAMEWIN_SetTextColorEx()

之前	之后
	

描述

设置给定状态的文本颜色。

原型

```
void FRAMEWIN_SetTextColorEx(FRAMEWIN_Handle hObj,   unsigned Index,
                             GUI_COLOR       Color);
```

参数	描述
hObj	框架窗口的句柄。
Index	（参见下表）
Color	要使用的颜色。

参数 Index 的允许值	
0	框架窗口未激活时使用的颜色。
1	框架窗口激活时使用的颜色。

FRAMEWIN_SetTitleHeight()

之前	之后
	

描述

设置标题栏的高度。

原型

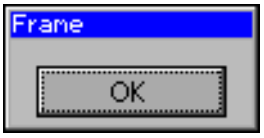
```
int FRAMEWIN_SetTitleHeight(FRAMEWIN_Handle hObj, int Height);
```

参数	描述
hObj	框架窗口的句柄。
Height	标题栏的高度。

其他信息

标题栏的默认高度取决于显示标题文本所用字体的大小。使用 FRAMEWIN_SetTitleHeight 时，高度将固定为给定的值。改变字体对标题栏的高度没有影响。使用值 0 会恢复到默认的高度。

FRAMEWIN_SetTitleVis()

之前	之后
	

描述

设置标题栏的可见性标记。

原型

```
void FRAMEWIN_SetTitleVis(FRAMEWIN_Handle hObj, int Show);
```

参数	描述
hObj	框架窗口的句柄。
Show	可见时为 1 （默认），隐藏时为 0

FRAMEWIN_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

15.8.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_FrameWin.c

需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_FrameWin.c 的屏幕截图：

15.9 GRAPH：图形小工具

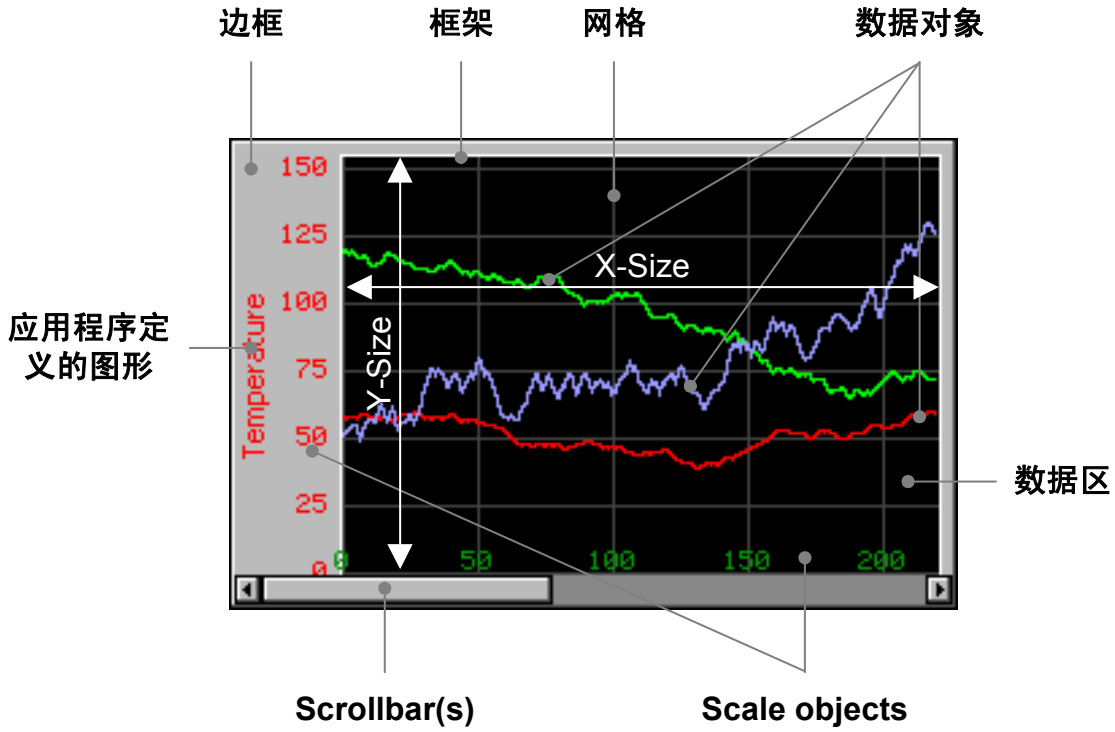
图形小工具可用于可视化数据。图形小工具的典型应用是显示测量值或函数图形的曲线，可同时显示多条曲线。可使用水平和垂直刻度来标记曲线。可在背景上显示具有不同水平和垂直间距的网格。如果数据阵列不能容纳进图形的可见区域，小工具可自动显示滚动条，从而可在大的数据阵列中进行滚动。

15.9.1 图形小工具的结构

图形小工具由不同种类的对象组成：

- 图形小工具自身，可以附加数据对象和刻度对象。
- 可选的一个或多个数据对象。
- 可选的一个或多个刻度对象。

下图显示了图形小工具的详细结构：



下表描述了上图的细节：6

细节	描述
Border	可选边框是图形小工具的一部分。
Frame	数据区四周的细线，是图形小工具的一部分。
Grid	显示在数据区的背景中，是图形小工具的一部分。
Data area	网格和数据对象的显示区域。
Data object(s)	对于每条曲线，应向图形小工具添加一个数据对象。
Application defined graphic	应用程序定义的一种回调函数，可用于绘制任何应用程序定义的文本和 / 或图形。
Scrollbar(s)	如果数据对象的范围大于图形小工具的数据区，则图形小工具可自动显示水平和 / 或垂直滚动条。
Scale object(s)	可附加到图形小工具的水平和垂直刻度。
X-Size	数据区的 X 尺寸。
Y-Size	数据区的 Y 尺寸。

15.9.2 创建和删除图形小工具

创建图形小工具的过程如下：

- 创建图形小工具并设置所需的属性。
- 创建数据对象。
- 将数据对象附加到图形小工具。
- 创建可选的刻度对象。
- 将刻度对象附加到图形小工具。

附加到图形小工具后，就不需要通过应用程序来删除数据和刻度对象，而由图形小工具完成。

示例

以下是如何创建和删除图形小工具的小示例：

```
GRAPH_DATA_Handle  hData;
GRAPH_SCALE_Handle hScale;
WM_HWIN hGraph;
hGraph = GRAPH_CreateEx(10, 10, 216, 106, WM_HBKWIN, WM_CF_SHOW, 0, GUI_ID_GRAPH0);
hData  = GRAPH_DATA_YT_Create(GUI_DARKGREEN, NumDataItems, aData0, MaxNumDataItems);
GRAPH_AttachData(hGraph, hData);
hScale = GRAPH_SCALE_Create(28, GUI_TA_RIGHT, GRAPH_SCALE_CF_VERTICAL, 20);
GRAPH_AttachScale(hGraph, hScale);
/*
  Do something with the widget...
*/
WM_DeleteWindow(hGraph);
```

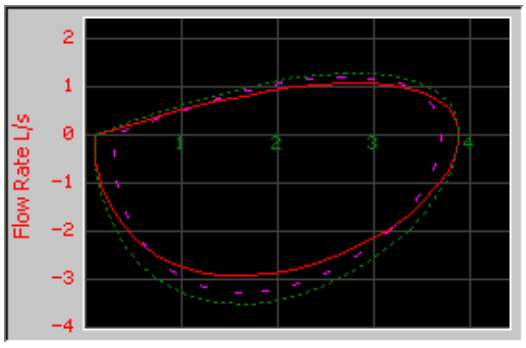
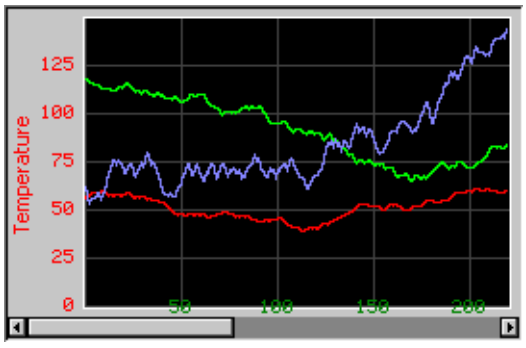
15.9.3 绘制过程

如前所述，图形小工具由不同部分和 “子” 对象组成。以下描述了绘制小工具的顺序：

1. 用背景色填充背景。
2. 调用可选的回调例程。例如，这样就能绘制用户定义的网格。
3. 绘制网格 （如已启用）。
4. 绘制数据对象和边框区域。
5. 绘制刻度对象。
6. 调用可选的回调例程。例如，这样就能绘制用户定义的刻度，或一些其他的文本和 / 或图形。

15.9.4 支持的曲线类型

显示测量值连续更新的曲线的要求不同于显示具有 X/Y 坐标的函数图形的要求。因此，小工具当前支持 2 种数据对象，如下表所示：

GRAPH_DATA_XY	GRAPH_DATA_YT
	

15.9.4.1 GRAPH_DATA_XY

此数据对象用于显示由点阵列组成的曲线，对象数据绘制为折线。此数据对象的典型应用是绘制函数图形。

15.9.4.2 GRAPH_DATA_YT

此数据对象用于显示图形上每个 X 位置都具有一个 Y 值的曲线。此数据对象的典型应用是显示测量值连续更新的曲线。

15.9.5 配置选项

15.9.5.1 图形小工具

类型	宏	默认值	描述
N	GRAPH_BKCOLOR_DEFAULT	GUI_BLACK	数据区的默认背景色。
N	GRAPH_BORDERCOLOR_DEFAULT	0xC0C0C0	边框的默认背景色。
N	GRAPH_FRAMECOLOR_DEFAULT	GUI_WHITE	细框架线的默认颜色。
N	GRAPH_GRIDCOLOR_DEFAULT	GUI_DARKGRAY	绘制网格的默认颜色。
N	GRAPH_GRIDSPACING_X_DEFAULT	50	网格的默认水平间距。
N	GRAPH_GRIDSPACING_Y_DEFAULT	50	网格的默认垂直间距。
N	GRAPH_BORDER_L_DEFAULT	0	左边框的默认尺寸。
N	GRAPH_BORDER_T_DEFAULT	0	上边框的默认尺寸。
N	GRAPH_BORDER_R_DEFAULT	0	右边框的默认尺寸。
N	GRAPH_BORDER_B_DEFAULT	0	下边框的默认尺寸。

15.9.5.2 刻度对象

类型	宏	默认值	描述
N	GRAPH_SCALE_TEXTCOLOR_DEFAULT	GUI_WHITE	默认文本颜色。
S	GRAPH_SCALE_FONT_DEFAULT	&GUI_Font6x8	绘制值使用的默认字体。

15.9.6 键盘反应

该小工具无法获取输入焦点，并且对键盘输入无任何反应。

15.9.7 图形 API

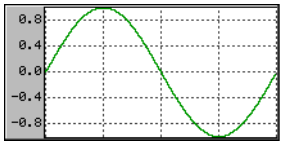
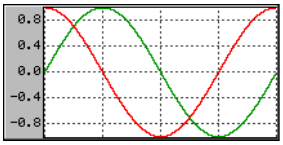
下表按字母顺序列出了可用的 emWin GRAPH 相关例程。这些例程的详细说明如下。

例程	描述
常用例程	
GRAPH_AttachData()	将数据对象附加到 GRAPH 小工具。
GRAPH_AttachScale()	将刻度对象附加到 GRAPH 小工具。
GRAPH_CreateEx()	创建 GRAPH 小工具。
GRAPH_CreateIndirect()	从资源表项创建 GRAPH 小工具。
GRAPH_CreateUser()	使用额外字节作为用户数据创建 GRAPH 小工具。
GRAPH_DetachData()	从 GRAPH 小工具分离数据对象。
GRAPH_DetachScale()	从 GRAPH 小工具分离刻度对象。
GRAPH_GetUserData()	检索由 GRAPH_SetUserData() 设置的数据。
GRAPH_SetBorder()	设置边框的尺寸 （上、下、左、右）。
GRAPH_SetColor()	设置 GRAPH 小工具的颜色。
GRAPH_SetGridDistX()	设置水平网格间距。
GRAPH_SetGridDistY()	设置垂直网格间距。

例程	描述
GRAPH_SetGridFixedX()	固定 X 轴方向的网格。
GRAPH_SetGridVis()	启用网格绘制。
GRAPH_SetLineStyleH()	设置水平网格线的线型。
GRAPH_SetLineStyleV()	设置垂直网格线的线型。
GRAPH_SetUserData()	设置 GRAPH 小工具的额外数据。
GRAPH_SetUserDraw()	设置用户回调函数。
GRAPH_SetVSizeX()	设置 GRAPH 小工具的水平范围。
GRAPH_SetVSizeY()	设置 GRAPH 小工具的垂直范围。
GRAPH_DATA_YT 相关的例程	
GRAPH_DATA_YT_AddValue()	向 GRAPH_DATA_YT 对象添加数据项。
GRAPH_DATA_YT_Clear()	清除 GRAPH_DATA_YT 对象的所有数据项。
GRAPH_DATA_YT_Create()	创建 GRAPH_DATA_YT 对象。
GRAPH_DATA_YT_Delete()	删除 GRAPH_DATA_YT 对象。
GRAPH_DATA_YT_MirrorX()	镜像 x 轴。
GRAPH_DATA_YT_SetAlign()	设置给定 GRAPH_DATA_YT 对象的对齐方式。
GRAPH_DATA_YT_SetOffY()	设置用于绘制数据的垂直偏移。
GRAPH_DATA_XY 相关的例程	
GRAPH_DATA_XY_AddPoint()	向 GRAPH_DATA_XY 对象添加一个点。
GRAPH_DATA_XY_Create()	创建 GRAPH_DATA_XY 对象。
GRAPH_DATA_XY_Delete()	删除 GRAPH_DATA_XY 对象。
GRAPH_DATA_XY_SetLineStyle()	设置用于绘制数据的线型。
GRAPH_DATA_XY_SetOffX()	设置用于绘制数据的水平偏移。
GRAPH_DATA_XY_SetOffY()	设置用于绘制数据的垂直偏移。
GRAPH_DATA_XY_SetOwnerDraw()	设置自回调函数。
GRAPH_DATA_XY_SetPenSize()	设置绘制数据所用的画笔尺寸。
刻度相关的例程	
GRAPH_SCALE_Create()	创建 GRAPH_SCALE 对象。
GRAPH_SCALE_Delete()	删除 GRAPH_SCALE 对象。
GRAPH_SCALE_SetFactor()	设置用于计算从像素转换到所需单位的计算因子。
GRAPH_SCALE_SetFixed()	避免刻度滚动。
GRAPH_SCALE_SetFont()	设置用于绘制编号的字体。
GRAPH_SCALE_SetNumDecs()	设置小数部分的位数。
GRAPH_SCALE_SetOff()	设置添加到编号的可选偏移。
GRAPH_SCALE_SetPos()	设置刻度的水平或垂直位置。
GRAPH_SCALE_SetTextColor()	设置刻度的文本颜色。
GRAPH_SCALE_SetTickDist()	设置刻度线之间的像素距离。

15.9.7.1 常用例程

GRAPH_AttachData()

之前	之后
	

描述

将数据对象附加到现有图形小工具。

原型

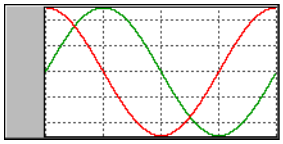
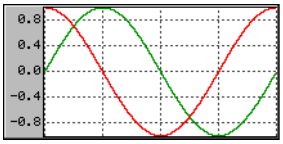
```
void GRAPH_AddGraph(GRAPH_Handle hObj, GRAPH_DATA_Handle hData);
```

参数	描述
hObj	小工具的句柄
hData	要添加到小工具的数据对象的句柄。数据对象应由 GRAPH_DATA_YT_Create() 或 GRAPH_DATA_XY_Create() 创建。

其他信息

附加到图形小工具后，应用程序无需破坏刻度对象。
在删除图形小工具时会删除所有附加的刻度对象。
有关如何创建数据对象的详细信息，请参见 “GRAPH_DATA_YT_Create()”（第 456 页）和 “GRAPH_DATA_XY_Create()”（第 459 页）。

GRAPH_AttachScale()

之前	之后
	

描述

将刻度对象附加到现有图形小工具。

原型

```
void GRAPH_AttachScale(GRAPH_Handle hObj, GRAPH_SCALE_Handle hScale);
```

参数	描述
hObj	小工具的句柄
hScale	要添加的刻度的句柄。

其他信息

附加到图形小工具后，应用程序无需破坏刻度对象。在删除图形小工具时会删除所有附加的刻度对象。
有关如何创建刻度对象的详细信息，请参见 “GRAPH_SCALE_Create()”（第 463 页）。

GRAPH_CreateEx()

描述

在指定位置创建指定尺寸的新 GRAPH 小工具。

原型

```
GRAPH_Handle GRAPH_CreateEx(int     x0,      int y0,
                            int     xsize,   int ysize,
                            WM_HWIN hParent, int WinFlags,
                            int     ExFlags, int Id);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新按钮窗口将是桌面 （顶级窗口）的子窗口。
WinFlags	窗口创建标记。为使小工具立即可见，通常使用 WM_CF_SHOW （有关可用参数值的清单，请参见 “WM_CreateWindow()”（第 309 页））。
ExFlags	（参见下表）
Id	小工具的窗口 ID。

参数 ExFlags 的允许值	
GRAPH_CF_GRID_FIXED_X	此标记 “固定” X 轴方向上的网格。这意味着如果使用水平滚动，则网格保持在原位。

返回值

已创建的 GRAPH 小工具的句柄；函数失败时为 0。

GRAPH_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。

GRAPH_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细描述，请参见函数 GRAPH_CreateEx()。

GRAPH_DetachData()

描述

从图形小工具分离数据对象。

原型

```
void GRAPH_DetachData(GRAPH_Handle hObj, GRAPH_DATA_Handle hData);
```

参数	描述
hObj	小工具的句柄
hData	要从小工具分离的数据对象的句柄。

其他信息

从图形小工具分离后，需要使用应用程序来破坏数据对象，
分离数据对象时不会将其删除。有关如何删除数据对象的详细信息，请参见“GRAPH_DATA_YT_Delete()”（第 457 页）和“GRAPH_DATA_XY_Delete()”（第 460 页）。

GRAPH_DetachScale()

描述

从图形小工具分离刻度对象。

原型

```
void GRAPH_DetachScale(GRAPH_Handle hObj, GRAPH_SCALE_Handle hScale);
```

参数	描述
hObj	小工具的句柄
hScale	要从小工具分离的刻度对象的句柄。

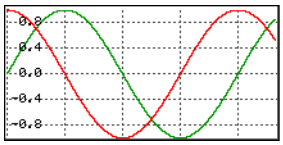
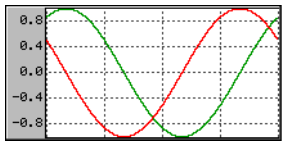
其他信息

从图形小工具分离后，需要使用应用程序来破坏刻度对象，
分离刻度对象时不会将其删除。有关如何删除刻度对象的详细信息，请参见“GRAPH_SCALE_Delete()”（第 463 页）。

GRAPH_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

GRAPH_SetBorder()

之前	之后
	

描述

设置给定图形小工具的左、上、右和下边框。

原型

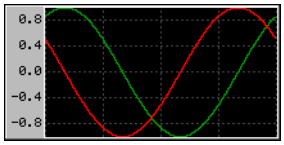
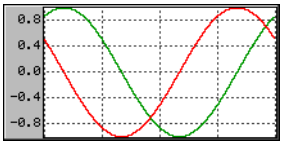
```
void GRAPH_SetBorder(GRAPH_Handle hObj,
                     unsigned     BorderL, unsigned BorderT,
                     unsigned     BorderR, unsigned BorderB);
```

参数	描述
hObj	小工具的句柄。
BorderL	距离左边框的尺寸（单位：像素）。
BorderT	距离上边框的尺寸（单位：像素）。
BorderR	距离右边框的尺寸（单位：像素）。
BorderB	距离下边框的尺寸（单位：像素）。

其他信息

边框尺寸是小工具有效框架和图形小工具数据区之间的像素数。框架即是围绕数据区的细线，仅在边框尺寸至少为一个像素时可见。有关如何设置边框和细框架颜色的详细信息，请参见 “GRAPH_SetColor()” （第 450 页）。

GRAPH_SetColor()

之前	之后
	

描述

设置给定图形小工具的所需颜色。

原型

```
GUI_COLOR GRAPH_SetColor(GRAPH_Handle hObj,   GUI_COLOR Color,
                         unsigned     Index);
```

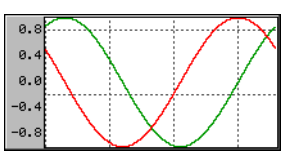
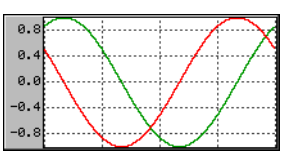
参数	描述
hObj	小工具的句柄。
Color	要用于所需项目的颜色。
Index	（参见下表）

参数 Index 的允许值	
GRAPH_CI_BK	设置背景颜色。
GRAPH_CI_BORDER	设置边框区域的颜色。
GRAPH_CI_FRAME	设置细框架线的颜色。
GRAPH_CI_GRID	设置网格的颜色。

返回值

所需项目先前所用的颜色。

GRAPH_SetGridDistX(), GRAPH_SetGridDistY()

之前	之后
	

描述

这些函数设置从一条网格线到下一网格线的距离。

原型

```
unsigned GRAPH_SetGridDistX(GRAPH_Handle hObj, unsigned Value);
```



```
unsigned GRAPH_SetGridDistY(GRAPH_Handle hObj, unsigned Value)
```

参数	描述
hObj	小工具的句柄
Value	以像素为单位的从一条网格线到下一网格线的距离，默认值为 50 像素。

返回值

先前的网格线距离。

其他信息

第一条垂直网格线绘制在数据区的最左位置，第一条水平网格线绘制在数据区的底部位置，使用偏移时除外。

GRAPH_SetGridFixedX()

描述

固定 X 轴方向的网格。

原型

```
unsigned GRAPH_SetGridFixedX(GRAPH_Handle hObj, unsigned OnOff);
```

参数	描述
hObj	小工具的句柄。
OnOff	要在 X 轴方向固定网格时为 1，否则为 0 （默认值）。

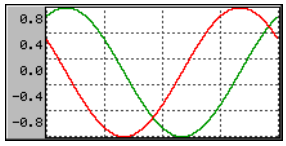
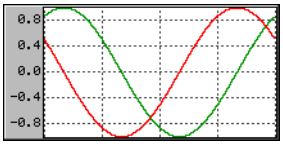
返回值

先前使用的值

其他信息

在某些情况下，在 X 轴方向固定网格非常有用。典型应用是绘制 YT 图，此时连续添加新值，可能会水平滚动。在此情况下，就希望在背景中固定网格。
有关如何为图形小工具激活滚动的详细信息，请参见第 454 页的 “GRAPH_SetVSizeX(), GRAPH_SetVSizeY()”（第 454 页）。

GRAPH_SetGridOffY()

之前	之后
	

描述

添加用于显示水平网格线的偏移。

原型

```
unsigned GRAPH_SetGridOffY(GRAPH_Handle hObj, unsigned Value);
```

参数	描述
hObj	小工具的句柄。
Value	要使用的偏移。

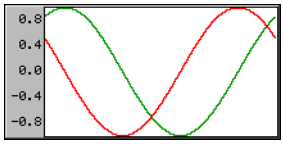
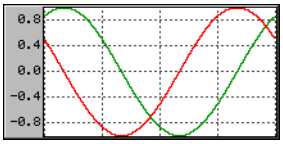
返回值

先前用于绘制水平网格线的偏移。

其他信息

渲染网格时，小工具使用当前间距，从数据区的底部开始绘制水平网格线。有时会出现零点位于 Y 轴中间的情况，此时中间没有网格线。在此情况下，可通过此函数添加偏移，在 Y 轴方向偏移网格。正值向下偏移网格，负值向上偏移网格。
有关如何设置网格间距的详细信息，请参见 “GRAPH_SetGridDistX(), GRAPH_SetGridDistY()” （第 450 页）的函数。

GRAPH_SetGridVis()

之前	之后
	

描述

设置网格线的可见性。

原型

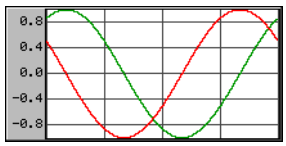
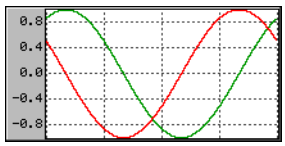
```
unsigned GRAPH_SetGridVis(GRAPH_Handle hObj, unsigned OnOff);
```

参数	描述
hObj	小工具的句柄。
OnOff	网格应可见时为 1，否则为 0 （默认值）。

返回值

先前网格可见性的值。

GRAPH_SetLineStyleH(), GRAPH_SetLineStyleV()

之前	之后
	

描述

这些函数用于设置水平和垂直网格线的线型。

原型

```
U8 GRAPH_SetLineStyleH(GRAPH_Handle hObj, U8 LineStyle);
U8 GRAPH_SetLineStyleV(GRAPH_Handle hObj, U8 LineStyle);
```

参数	描述
hObj	小工具的句柄。
LineStyle	要使用的线型。有关支持的线型的详细信息，请参见 “GUI_SetLineStyle()”（第 125 页）。默认值为 GUI_LS_SOLID。

返回值

先前用于绘制水平 / 垂直网格线的线型。

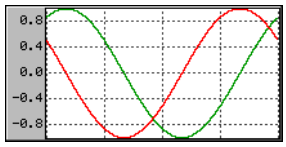
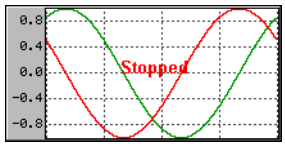
其他信息

请注意，使用 GUI_LS_SOLID 以外的其他线型要花更多时间来显示网格。

GRAPH_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

GRAPH_SetUserDraw()

之前	之后
	

描述

设置用户绘制函数。此函数由小工具在绘制过程中调用，使应用程序能够绘制用户定义的数据。

原型

```
void GRAPH_SetUserDraw(GRAPH_Handle hObj,
                       void (* pUserDraw)(WM_HWIN hObj, int Stage));
```

参数	描述
hObj	小工具的句柄
pUserDraw	指向绘制过程中要由小工具调用的应用程序函数的指针。

参数 Stage 的允许值	
GRAPH_DRAW_FIRST	填充数据区的背景后调用函数。例如，使应用程序能绘制用户定义的网格。
GRAPH_DRAW_LAST	绘制所有图形项目后调用函数。例如，使应用程序能用用户定义的刻度标记数据。

其他信息

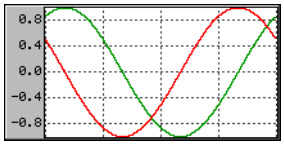
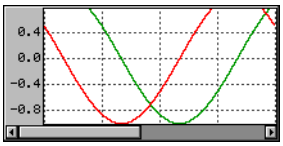
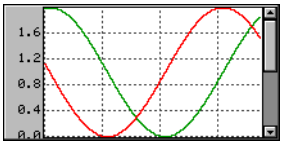
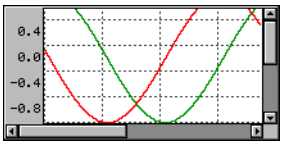
如本章开始时所述，在填充数据区背景以及绘制所有图形项目后，开始调用该用户绘制函数。第一次调用时裁剪区限制在数据区。最后一次调用时，裁剪区限制在除有效框架外的整个图形小工具区域。

示例

以下是如何使用用户绘制函数的小示例：

```
static void _UserDraw(WM_HWIN hWin, int Stage) {
  switch (Stage) {
  case GRAPH_DRAW_FIRST:
    /* Draw for example a user defined grid...*/
    break;
  case GRAPH_DRAW_LAST:
    /* Draw for example a user defined scale or additional text...*/
    break;
  }
}

static void _CreateGraph(void) {
  WM_HWIN hGraph;
  hGraph = GRAPH_CreateEx(10, 10, 216, 106, WM_HBKWIN, WM_CF_SHOW, 0, GUI_ID_GRAPH0);
  GRAPH_SetUserDraw(hGraph, _UserDraw); /* Enable user draw */
  ...
}
```

GRAPH_SetVSizeX(), GRAPH_SetVSizeY()

之前	之后
	  

描述

这两个函数设置 X 和 Y 轴的虚拟尺寸。

原型

```
unsigned GRAPH_SetVSizeX(GRAPH_Handle hObj, unsigned Value);
```



```
unsigned GRAPH_SetVSizeY(GRAPH_Handle hObj, unsigned Value);
```

参数	描述
hObj	小工具的句柄。
Value	以像素为单位的 X 或 Y 轴的虚拟尺寸。

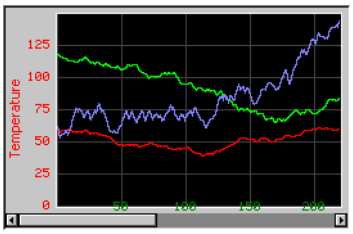
返回值

先前小工具数据区在 X 或 Y 轴方向的虚拟尺寸。

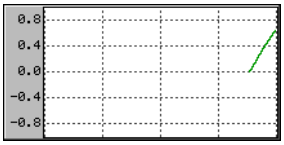
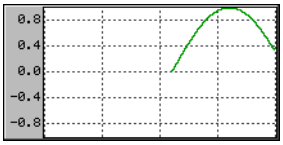
其他信息

如果小工具虚拟尺寸大于数据区的可见尺寸，则小工具自动显示滚动条。例如，如果函数 GRAPH_DATA_YT_Create() 创建的数据对象包含的数据多于可在数据区中显示的数据，则可使用函数 GRAPH_SetVSizeX() 启用滚动。只要数据项的数量大于可见数据区的 X 尺寸，GRAPH_SetVSizeX(NumDataItems) 等类似函数调用就可启用水平滚动条。

15.9.7.2 GRAPH_DATA_YT 相关的例程



GRAPH_DATA_YT_AddValue()

之前	之后
	

描述

向 GRAPH_DATA_YT 对象添加新数据项。

原型

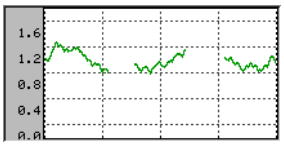
```
void GRAPH_DATA_YT_AddValue(GRAPH_DATA_Handle hDataObj, I16 Value);
```

参数	描述
hDataObj	数据对象的句柄。
Value	要添加到数据对象的值。

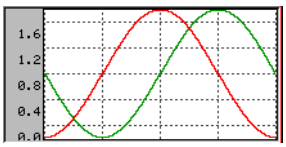
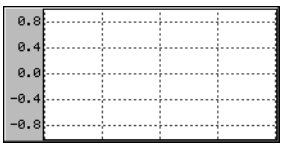
其他信息

给定的数据值添加到数据对象。如果该数据对象 “已满”，即意味着它包含的数据项与创建时在参数 MaxNumItems 中指定的项数相同，在添加新值前会首先移动一个数据项。因此，向 “已满”对象添加数据项时，第一个数据项被移出。

可使用值 0x7FFF 来处理无效数据值，绘制图形时这些值会被排除。以下屏幕截图显示了具有 2 个无效数据缺口的图形：



GRAPH_DATA_YT_Clear()

之前	之后
	

描述

清除数据对象的所有数据项。

原型

```
void GRAPH_DATA_YT_Clear(GRAPH_DATA_Handle hDataObj);
```

参数	描述
hDataObj	数据对象的句柄。

GRAPH_DATA_YT_Create()

描述

创建 GRAPH_DATA_YT 对象。此类对象要求 x 轴上的每个点在 y 轴上都有一个值，通常用于与时间相关的图形。

原型

```
GRAPH_DATA_Handle GRAPH_DATA_YT_Create(GUI_COLOR   Color,
                                       unsigned    MaxNumItems,
                                       I16       * pData,
                                       unsigned    NumItems);
```

参数	描述
Color	绘制数据要使用的颜色。
MaxNumItems	数据项的最大数。
pData	指向要添加到对象的数据的指针。该指针应指向一组 I16 值。
NumItems	要添加的数据项数。

返回值

创建成功时数据对象的句柄，否则为 0。

其他信息

最后一个数据项显示在数据区的最右列。如果数据对象包含的数据多于图形小工具数据区可显示的数据，则可使用函数 GRAPH_SetVSizeX() 来显示滚动条，从而可在较大的数据对象中滚动。
附加到图形小工具后，就不需要使用应用程序来删除数据对象，在删除图形小工具时会自动删除。

GRAPH_DATA_YT_Delete()

描述

删除给定的数据对象。

原型

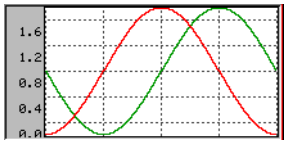
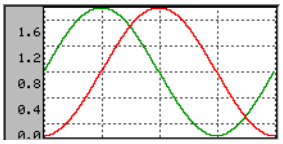
```
void GRAPH_DATA_YT_Delete(GRAPH_DATA_Handle hDataObj);
```

参数	描述
hDataObj	要删除的数据对象。

其他信息

删除图形小工具时，会删除所有当前附加的数据对象。因此应用程序只需删除未附加的数据对象。

GRAPH_DATA_YT_MirrorX()

之前	之后
	

描述

镜像小工具的 x 轴。

原型

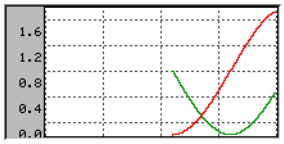
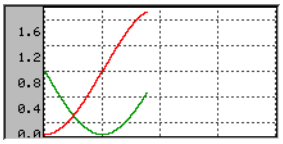
```
void GRAPH_DATA_YT_MirrorX(GRAPH_DATA_Handle hDataObj, int Value);
```

参数	描述
hDataObj	数据对象的句柄。
OnOff	1 表示镜像 x 轴， 0 表示默认视图。

其他信息

默认情况下，数据从右向左绘制。调用此函数后，数据从左向右绘制。

GRAPH_DATA_YT_SetAlign()

之前	之后
	

描述

设置数据的对齐方式。

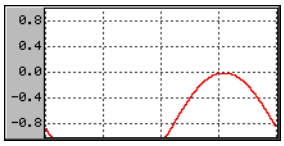
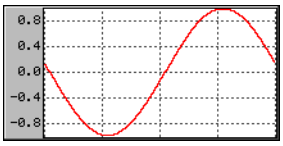
原型

```
void GRAPH_DATA_YT_SetAlign(GRAPH_DATA_Handle hDataObj, int Align);
```

参数	描述
hDataObj	数据对象的句柄。
Align	（参见下表）

参数 Align 的允许值	
GRAPH_ALIGN_RIGHT	数据右对齐 （默认）。
GRAPH_ALIGN_LEFT	数据左对齐。

GRAPH_DATA_YT_SetOffY()

之前	之后
	

描述

设置用于绘制对象数据的垂直偏移。

原型

```
void GRAPH_DATA_YT_SetOffY(GRAPH_DATA_Handle hDataObj, int Off);
```

参数	描述
hDataObj	数据对象的句柄。
Off	要用于绘制数据的垂直偏移。

其他信息

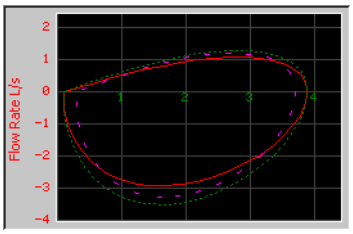
数据对象所显示的数据垂直范围，范围为 (0) -（数据区的 Y 尺寸 – 1）。使用滚动条时，当前滚动位置会添加到范围中。

示例

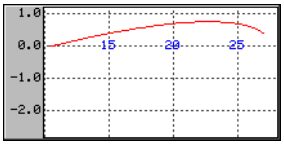
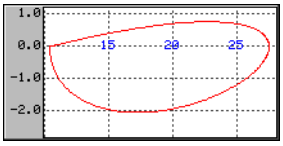
例如，如果可见数据范围为 -200 到 -100，则数据应正向偏移 200 像素：

```
GRAPH_DATA_YT_SetOffY(hDataObj, 200);
```

15.9.7.3 GRAPH_DATA_XY 相关的例程



GRAPH_DATA_XY_AddPoint()

之前	之后
	

描述

向 GRAPH_DATA_XY 对象添加新数据项。

原型

```
void GRAPH_DATA_XY_AddPoint(GRAPH_DATA_Handle hDataObj, GUI_POINT * pPoint);
```

参数	描述
hDataObj	数据对象的句柄。
pPoint	指向要添加到数据对象的 GUI_POINT 结构的指针。

其他信息

给定的点被添加到数据对象。如果该数据对象 “已满”，即意味着它包含的点数与创建时在参数 MaxNumItems 中指定的点数相同，在添加新点前会首先移动一个数据项。因此将新点添加到 “已满” 对象时，第一个点会被移出。

GRAPH_DATA_XY_Create()

描述

创建 GRAPH_DATA_XY 对象。此类对象能存储任何值对，这些值对将通过添加命令进行连接。

原型

```
GRAPH_DATA_Handle GRAPH_DATA_XY_Create(GUI_COLOR   Color,
                                       unsigned    MaxNumItems,
                                       GUI_POINT * pData,
                                       unsigned    NumItems);
```

参数	描述
Color	绘制数据要使用的颜色。
MaxNumItems	最大点数。
pData	指向要添加到对象的数据的指针。应指向 GUI_POINT 阵列的指针。
NumItems	要添加的点数。

返回值

创建成功时数据对象的句柄，否则为 0。

其他信息

附加到图形小工具后，就不需要使用应用程序来删除数据对象，在删除图形小工具时会自动删除。

GRAPH_DATA_XY_Delete()

描述

删除给定的数据对象。

原型

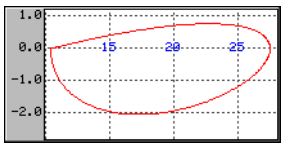
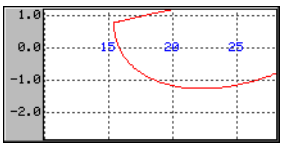
```
void GRAPH_DATA_XY_Delete(GRAPH_DATA_Handle hDataObj);
```

参数	描述
hDataObj	要删除的数据对象。

其他信息

删除图形小工具时，会删除所有当前附加的数据对象。因此应用程序只需删除未附加的数据对象。

GRAPH_DATA_XY_SetOffX(), GRAPH_DATA_XY_SetOffY()

之前	之后
	

描述

设置用于绘制折线的垂直或水平偏移。

原型

```
void GRAPH_DATA_XY_SetOffX(GRAPH_DATA_Handle hDataObj, int Off);
void GRAPH_DATA_XY_SetOffY(GRAPH_DATA_Handle hDataObj, int Off);
```

参数	描述
hDataObj	数据对象的句柄。
Off	要用于绘制折线的水平 / 垂直偏移。

其他信息

数据对象显示的数据范围为 (0, 0) -（数据区的 X 尺寸 - 1，数据区的 Y 尺寸 - 1）。使用滚动条时，当前滚动位置会添加到各自的范围中。要使其他范围的数据可见，应使用此函数设置偏移，使数据处于可见区域。

示例

例如，要使可见数据范围为 (100, -1200) - (200, -1100)，应使用以下偏移：

```
GRAPH_DATA_XY_SetOffX(hDataObj, -100);
GRAPH_DATA_XY_SetOffY(hDataObj, 1200);
```


GRAPH_DATA_XY_SetOwnerDraw()

描述

设置自回调函数。此函数由小工具在绘制过程中调用，使应用程序能在小工具顶部绘制附加项目。

原型

```
void GRAPH_DATA_XY_SetOwnerDraw(GRAPH_DATA_Handle hDataObj,
  void (* pOwnerDraw)(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo));
```

参数	描述
hDataObj	要删除的数据对象。
pOwnerDraw	指向绘制过程中要由小工具调用的应用程序函数的指针。

其他信息

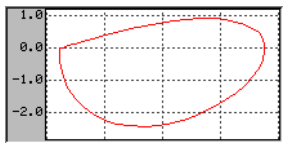
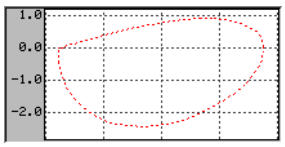
自画函数在绘制背景、刻度和网格线后调用。

示例

以下代码段为自画函数的示例：

```
static int _cbData(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_DRAW:
    GUI_DrawRect(pDrawItemInfo->x0 - 3, pDrawItemInfo->y0 - 3,
                 pDrawItemInfo->x0 + 3, pDrawItemInfo->y0 + 3);
    break;
  }
  return 0;
}

void MainTask(void) {
  WM_HWIN            hGraph;
  GRAPH_DATA_Handle  hData;
  GUI_Init();
  hGraph = GRAPH_CreateEx (140, 100, 171, 131, 0, WM_CF_SHOW, 0, GUI_ID_GRAPH0);
  hData  = GRAPH_DATA_XY_Create(USER_DEFINED_COLOR, 126, 0, 0);
  GRAPH_DATA_XY_SetOwnerDraw(hData, _cbData);
}
```

GRAPH_DATA_XY_SetLineStyle()

之前	之后
	

描述

设置绘制折线所用的线型。

原型

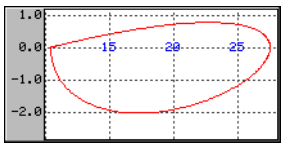
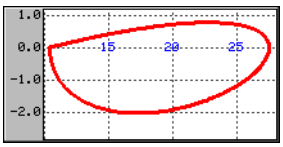
```
void GRAPH_DATA_XY_SetLineStyle(GRAPH_DATA_Handle hDataObj, U8 LineStyle);
```

参数	描述
hDataObj	数据对象的句柄。
LineStyle	要使用的新线型。有关支持的线型的详细信息，请参见 “GUI_SetLineStyle()”（第 125 页）。

局限

请注意，仅线型为 GUI_LS_SOLID （默认）的曲线可用画笔尺寸 >1 的画笔绘制。

GRAPH_DATA_XY_SetPenSize()

之前	之后
	

描述

设置绘制折线所用的画笔尺寸。

原型

```
void GRAPH_DATA_XY_SetPenSize(GRAPH_DATA_Handle hDataObj, U8 PenSize);
```

参数	描述
hDataObj	数据对象的句柄。
PenSize	绘制折线要使用的画笔尺寸。

局限

请注意，仅线型为 GUI_LS_SOLID （默认）的曲线可用画笔尺寸 >1 的画笔绘制。

15.9.7.4 刻度相关的例程

图形小工具支持用于标记的水平和垂直刻度。以下描述了使用刻度时可用的函数。

GRAPH_SCALE_Create()

描述

创建刻度对象。

原型

```
GRAPH_SCALE_Handle GRAPH_SCALE_Create(int      Pos,   int      Align,
                                      unsigned Flags, unsigned TickDist);
```

参数	描述
Pos	相对于图形小工具的左边 / 顶边的位置。
TextAlign	用于绘制编号的文本对齐方式。更多详细信息，请参阅 “GUI_GetTextAlign()”（第 81 页）。
Flags	（参见下表）
TickDist	从一个刻度线到下一刻度线的距离。

参数 Flags 允许值	
GRAPH_SCALE_CF_HORIZONTAL	创建水平刻度对象。
GRAPH_SCALE_CF_VERTICAL	创建垂直刻度对象。

返回值

创建成功时刻度对象的句柄，否则为 0。

其他信息

水平刻度对象从数据区的底边开始向顶部进行标记，垂直刻度对象从左边（水平刻度）开始向右边进行标记，其中第一个位置是零点。参数 TickDist 指定编号之间的距离。
对于水平刻度，参数 Pos 指定从图形小工具顶边到刻度文本的垂直距离，以像素为单位。对于垂直刻度，该参数指定从图形小工具左边到水平文本位置的水平距离。请注意，实际文本位置也取决于用参数 TextAlign 指定的文本对齐方式。
刻度对象为数据区内的每个位置绘制一个编号。对于水平刻度，有一个例外：如果第一个位置是 0，则在此位置不绘制编号。
附加到图形小工具后，就不需要使用应用程序来删除刻度对象，在删除图形小工具时会自动删除。

GRAPH_SCALE_Delete()

描述

删除给定的刻度对象。

原型

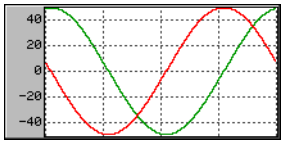
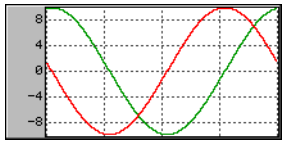
```
void GRAPH_SCALE_Delete(GRAPH_SCALE_Handle hScaleObj);
```

参数	描述
hScaleOBj	要删除的刻度对象。

其他信息

删除图形小工具时，会删除所有当前附加的刻度对象。因此应用程序只需删除未附加的刻度对象。

GRAPH_SCALE_SetFactor()

之前	之后
	

描述

设置用于计算要绘制的编号的因子。

原型

```
float GRAPH_SCALE_SetFactor(GRAPH_SCALE_Handle hScaleObj, float Factor);
```

参数	描述
hScaleOBj	刻度对象的句柄。
Factor	要用于计算编号的因子。

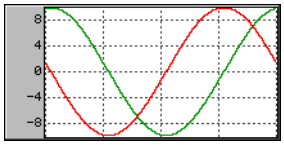
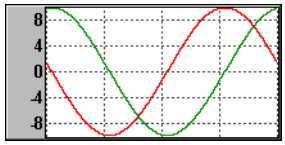
返回值

用于计算编号的旧因子。

其他信息

不使用因子时，刻度对象的单位为 “像素”。因此给定因子应将像素值转换为所需的单位。

GRAPH_SCALE_SetFont()

之前	之后
	

描述

设置用于绘制刻度编号的字体。

原型

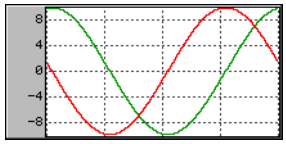
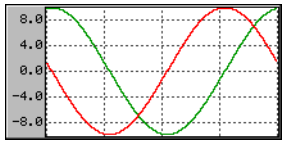
```
const GUI_FONT * GRAPH_SCALE_SetFont(GRAPH_SCALE_Handle   hScaleObj,
                                     const GUI_FONT     * pFont);
```

参数	描述
hScaleOBj	刻度对象的句柄。
pFont	要用的字体。

返回值

先前用于绘制编号的字体。

GRAPH_SCALE_SetNumDecs()

之前	之后
	

描述

设置要显示的小数点后的位数。

原型

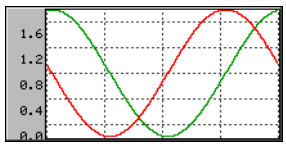
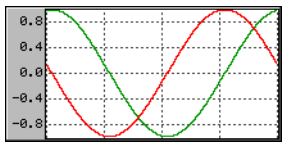
```
int GRAPH_SCALE_SetNumDecs(GRAPH_SCALE_Handle hScaleObj, int NumDecs);
```

参数	描述
hScaleOBj	刻度对象的句柄。
NumDecs	小数点后的位数。

返回值

先前的小数点后位数。

GRAPH_SCALE_SetOff()

之前	之后
	

描述

设置用于在正或负方向上 “移动” 刻度对象的偏移。

原型

```
int GRAPH_SCALE_SetOff(GRAPH_SCALE_Handle hScaleObj, int Off);
```

参数	描述
hScaleOBj	刻度对象的句柄。
Off	用于绘制刻度的偏移。

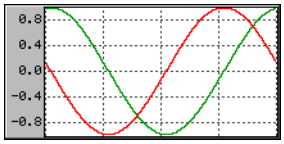
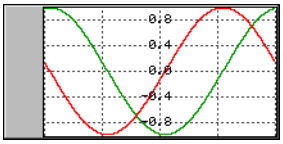
返回值

先前所用的偏移。

其他信息

如函数 GRAPH_SCALE_Create() 章节中所述，水平刻度对象从数据区的底边开始向顶部进行标记，垂直刻度对象从左边 （水平刻度）开始向右边进行标记，其中第一个位置是零点。在许多情况下，不希望第一个位置是零点。如果刻度要向正方向 “移动”，应添加正偏移，要向负方向 “移动”，应添加负值。

GRAPH_SCALE_SetPos()

之前	之后
	

描述

设置在图形小工具中显示刻度对象的位置。

原型

```
int GRAPH_SCALE_SetPos(GRAPH_SCALE_Handle hScaleObj, int Pos);
```

参数	描述
hScaleOBj	刻度对象的句柄。
Pos	要显示刻度的位置。

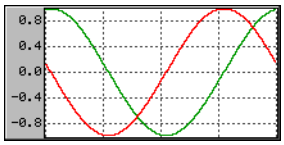
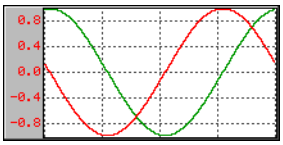
返回值

先前的刻度对象位置。

其他信息

对于水平刻度，参数 Pos 指定从图形小工具顶边到刻度文本的垂直距离，以像素为单位。对于垂直刻度，该参数指定从图形小工具左边到水平文本位置的水平距离。请注意，实际文本位置也取决于刻度对象的文本对齐方式。

GRAPH_SCALE_SetTextColor()

之前	之后
	

描述

设置用于绘制编号的文本颜色。

原型

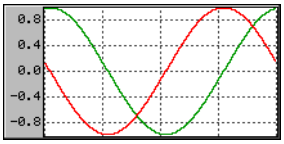
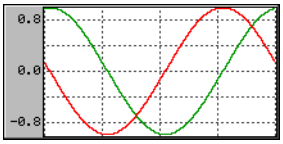
```
GUI_COLOR GRAPH_SCALE_SetTextColor(GRAPH_SCALE_Handle hScaleObj,
                                   GUI_COLOR          Color);
```

参数	描述
hScaleOBj	刻度对象的句柄。
Color	要用于显示编号的颜色。

返回值

先前用于显示编号的颜色。

GRAPH_SCALE_SetTickDist()

之前	之后
	

描述

设置从一个编号到下一编号的距离。

原型

```
unsigned GRAPH_SCALE_SetTickDist(GRAPH_SCALE_Handle hScaleObj,
                                 unsigned           Dist);
```

参数	描述
hScaleOBj	刻度对象的句柄。
Dist	两个编号之间以像素为单位的距离。

返回值

先前编号之间的距离。

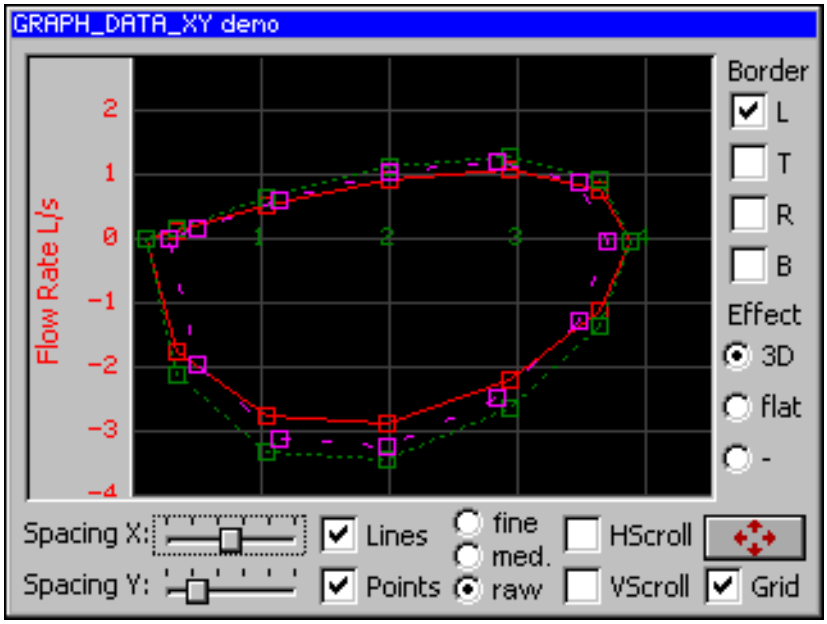
15.9.8 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

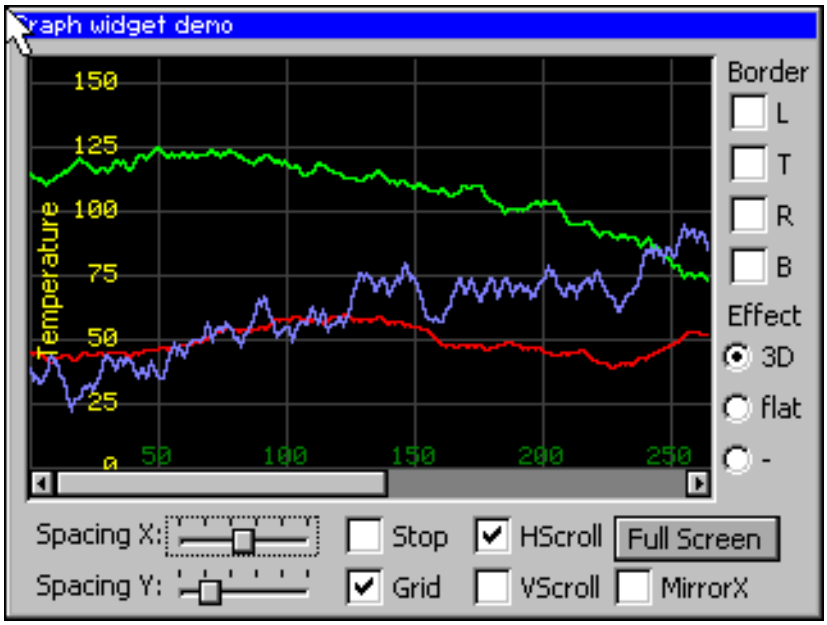
- WIDGET_GraphXY.c
- WIDGET_GraphYT.c

需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_GraphXY.c 的屏幕截图：



WIDGET_GraphYT.c 的屏幕截图：



15.10 HEADER：标题小工具

HEADER 小工具用于标记表格的列：

如果使用了指针输入设备 （PID），可通过使用 PID 拖动分隔线来调节标题项目的宽度。

使用鼠标时的特性

如果鼠标支持已启用，则光标出现并且 PID 在分隔线附近移动时，光标就会变成信号，可以在当前位置拖动分隔线。

使用触屏时的特性

如果在分隔线附近按下该小工具，光标出现并变成信号，此时即可拖动分隔线。

可拖动分隔线的屏幕截图

预定义的光标

2 种预定义光标如下所示：

GUI_CursorHeaderM （默认）	GUI_CursorHeaderMI
	

在使用 HEADER 小工具的时候，也可以创建并使用自己的光标，相关内容请参见本章后面。

皮肤设置 ...

... 可用于该小工具。上述屏幕截图显示使用默认皮肤的小工具。有关详细信息，请参阅“皮肤设置”一章。

15.10.1 配置选项

类型	宏	默认值	描述
N	HEADER_BKCOLOR_DEFAULT	0xAAAAAA	背景色的默认值。
S	HEADER_CURSOR_DEFAULT	&GUI_CursorHeaderM	默认光标
S	HEADER_FONT_DEFAULT	&GUI_Font13_1	默认字体
N	HEADER_BORDER_H_DEFAULT	2	文本和边框之间的垂直间距
N	HEADER_BORDER_V_DEFAULT	0	文本和边框之间的垂直间距
B	HEADER_SUPPORT_DRAG	1	启用 / 禁用拖动支持
N	HEADER_TEXTCOLOR_DEFAULT	GUI_BLACK	文本颜色的默认值

15.10.2 通知代码

以下事件是 HEADER 小工具作为 WM_NOTIFY_PARENT 消息的一部分发送给其父窗口的：

消息	描述
WM_NOTIFICATION_CLICKED	已单击小工具。
WM_NOTIFICATION_RELEASED	已释放小工具。
WM_NOTIFICATION_MOVED_OUT	已单击小工具，并且指针已移出小工具，但没有释放。

15.10.3 键盘反应

该小工具无法获取输入焦点，并且对键盘输入无任何反应。

15.10.4 HEADER API

下表按字母顺序列出了可用的 emWin HEADER 相关例程。这些例程的详细说明如下。

例程	描述
HEADER_AddItem()	在右侧增加一个项目
HEADER_Create()	创建 HEADER 小工具 （弃用）
HEADER_CreateAttached()	创建附加于窗口的 HEADER 小工具
HEADER_CreateEx()	创建 HEADER 小工具
HEADER_CreateIndirect()	从资源表项创建 HEADER 小工具
HEADER_CreateUser()	使用额外字节作为用户数据创建 HEADER 小工具。
HEADER_GetDefaultBkColor()	返回默认背景色
HEADER_GetDefaultBorderH()	返回水平间距值。
HEADER_GetDefaultBorderV()	返回垂直间距值。
HEADER_GetDefaultCursor()	返回指向默认光标的指针。
HEADER_GetDefaultFont()	返回指向默认字体的指针。
HEADER_GetDefaultTextColor()	返回默认文本颜色。
HEADER_GetHeight()	返回小工具的高度
HEADER_GetItemWidth()	返回项目的宽度。
HEADER_GetNumItems()	返回项目数。
HEADER_GetUserData()	检索使用 HEADER_SetUserData() 设置的数据。
HEADER_SetBitmap()	设置显示给定项目时使用的位图。
HEADER_SetBitmapEx()	设置显示给定项目时使用的位图。
HEADER_SetBkColor()	设置小工具的背景色。
HEADER_SetBMP()	设置显示给定项目时使用的位图。
HEADER_SetBMPEx()	设置显示给定项目时使用的位图。
HEADER_SetDefaultBkColor()	设置默认背景色。
HEADER_SetDefaultBorderH()	设置水平间距的默认值。
HEADER_SetDefaultBorderV()	设置垂直间距的默认值。

例程	描述
HEADER_SetDefaultCursor()	设置默认光标。
HEADER_SetDefaultFont()	设置默认字体。
HEADER_SetDefaultTextColor()	设置默认文本颜色。
HEADER_SetDragLimit()	将拖动项目限制设置为开或关。
HEADER_SetFont()	设置小工具的字体。
HEADER_SetHeight()	设置小工具的高度。
HEADER_SetItemText()	设置给定项目的文本。
HEADER_SetItemWidth()	设置给定项目的宽度。
HEADER_SetStreamedBitmap()	设置显示给定项目时使用的位图。
HEADER_SetStreamedBitmapEx()	设置显示给定项目时使用的位图。
HEADER_SetTextAlign()	设置给定项目的对齐方式。
HEADER_SetTextColor()	设置小工具的文本颜色。
HEADER_SetUserData()	设置 HEADER 小工具的额外数据。

HEADER_AddItem()

描述

向已存在的 HEADER 小工具添加项目。

原型

```
void HEADER_AddItem(HEADER_Handle   hObj, int Width,
                    const char    * s,    int Align);
```

参数	描述
hObj	小工具的句柄
Width	新项目的宽度
s	要显示的文本
Align	要设置的文本对齐模式。可能为水平对齐标记和垂直对齐标记。

参数 Align 的允许值 （水平和垂直标记可以通过 “OR” 操作进行组合）	
水平对齐	
GUI_TA_LEFT	X 轴方向左对齐 （默认）。
GUI_TA_HCENTER	X 轴方向对中。
GUI_TA_RIGHT	X 轴方向右对齐 （默认）。
垂直对齐	
GUI_TA_TOP	将 Y 轴位置与字符的顶部对齐 （默认）。
GUI_TA_VCENTER	在 Y 轴方向对中。
GUI_TA_BOTTOM	将 Y 轴位置与字体的底部像素行对齐。

其他信息

Width 参数可以为 0。如果 Width = 0，新项目的宽度将由给定文本与其水平间距的默认值计算得出。

HEADER_Create()

（弃用，应使用 HEADER_CreateEx() 代替。）

描述

在指定位置创建指定尺寸的 HEADER 小工具。

原型

```
HEADER_Handle HEADER_Create(int     x0,      int y0,
                            int     xsize,   int ysize,
                            WM_HWIN hParent, int Id,
                            int     Flags,   int SpecialFlags);
```

参数	描述
x0	HEADER 小工具最左侧的像素 （在父坐标中）。
y0	HEADER 小工具最顶端的像素 （在父坐标中）。
xsize	HEADER 小工具的水平尺寸 （以像素为单位）。
ysize	HEADER 小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄
Id	新创建的 HEADER 小工具的 Id
Flags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow() ）。
SpecialFlags	（保留供日后使用）

返回值

已创建 HEADER 小工具的句柄；函数失败时为 0。

HEADER_CreateAttached()

描述

创建附加于现有窗口的 HEADER 小工具。

原型

```
HEADER_Handle HEADER_CreateAttached(WM_HWIN hParent,
                                    int     Id,
                                    int     SpecialFlags);
```

参数	描述
hObj	小工具的句柄
Id	HEADER 小工具的 Id
SpecialFlags	（未使用，保留供日后使用）

返回值

已创建 HEADER 小工具的句柄；函数失败时为 0。

其他信息

附加的 HEADER 小工具实质上是定位于父窗口上并相应操作的子窗口。

HEADER_CreateEx()

描述

在指定位置创建指定尺寸的 HEADER 小工具。

原型

```
HEADER_Handle HEADER_CreateEx(int     x0,      int y0,
                              int     xsize,   int ysize,
                              WM_HWIN hParent, int WinFlags,
```



```
                                    int     ExFlags, int Id);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新的 HEADER 小工具将是桌面 （顶级窗口）的子项。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW（有关可用参数值的列表，请参阅“窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	未使用，保留供日后使用。
Id	小工具的窗口 ID。

返回值

已创建 HEADER 小工具的句柄；函数失败时为 0。

HEADER_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。

HEADER_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细描述，请参见函数 HEADER_CreateEx()。

HEADER_GetDefaultBkColor()

描述

返回创建 HEADER 小工具时使用的默认背景色。

原型

```
GUI_COLOR HEADER_GetDefaultBkColor(void);
```

返回值

创建 HEADER 小工具时使用的默认背景色。

HEADER_GetDefaultBorderH()

描述

返回创建 HEADER 小工具时使用的水平间距值。水平间距指文本和项目水平边框之间的水平距离，单位像素。

原型

```
int HEADER_GetDefaultBorderH(void);
```

返回值

创建 HEADER 小工具时使用的水平间距值。

其他信息

仅当新项目的给定宽度为 0 时，水平间距才有效。

HEADER_GetDefaultBorderV()

描述

返回创建 HEADER 小工具时使用的垂直间距值。垂直间距指文本和 HEADER 小工具垂直边框之间的垂直距离，单位像素。

原型

```
int HEADER_GetDefaultBorderV(void);
```

返回值

创建 HEADER 小工具时使用的垂直间距值。

HEADER_GetDefaultCursor()

描述

返回指向拖动项目宽度时显示的光标的指针。

原型

```
const GUI_CURSOR * HEADER_GetDefaultCursor(void);
```

返回值

指向拖动项目宽度时显示的光标的指针。

HEADER_GetDefaultFont()

描述

返回指向创建 HEADER 小工具时使用的默认字体的指针。

原型

```
const GUI_FONT * HEADER_GetDefaultFont(void);
```

返回值

指向创建 HEADER 小工具时使用的默认字体的指针。

HEADER_GetDefaultTextColor()

描述

返回创建 HEADER 小工具时使用的默认文本颜色。

原型

```
GUI_COLOR HEADER_GetDefaultTextColor(void);
```

返回值

创建 HEADER 小工具时使用的默认文本颜色。

HEADER_GetHeight()

描述

返回给定 HEADER 小工具的高度

原型

```
int HEADER_GetHeight(HEADER_Handle hObj);
```

参数	描述
hObj	小工具的句柄

返回值

给定 HEADER 小工具的高度。

HEADER_GetItemWidth()

描述

返回给定 HEADER 小工具的项目宽度。

原型

```
int HEADER_GetItemWidth(HEADER_Handle hObj, unsigned int Index);
```

参数	描述
hObj	小工具的句柄
Index	项目的索引

返回值

项目的宽度。

HEADER_GetNumItems()

描述

返回给定 HEADER 小工具的项目数量。

原型

```
int HEADER_GetNumItems(HEADER_Handle hObj);
```

参数	描述
hObj	小工具的句柄

返回值

给定 HEADER 小工具的项目数量。

HEADER_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

HEADER_SetBitmap()

描述

设置显示指定项目时使用的位图。

原型

```
void HEADER_SetBitmap(HEADER_Handle      hObj,
                      unsigned int       Index,
                      const GUI_BITMAP * pBitmap);
```

参数	描述
hObj	小工具的句柄
Index	项目的索引
pBitmap	指向要显示的位图结构的指针

其他信息

HEADER 小工具的一个项目可同时包含文本和位图。（请阅读 HEADER_SetBitmapEx 下面的示例）。

HEADER_SetBitmapEx()

描述

设置显示指定项目时使用的位图。

原型

```
void HEADER_SetBitmapEx(HEADER_Handle      hObj,    unsigned int Index,
                        const GUI_BITMAP * pBitmap,
                        int                x,       int          y);
```

参数	描述
hObj	小工具的句柄
Index	项目的索引
pBitmap	指向要显示的位图结构的指针
x	X 方向上的额外偏移
y	Y 方向上的额外偏移

其他信息

HEADER 小工具的一个项目可同时包含文本和位图。

示例：

```
...
HEADER_Handle hHeader;
GUI_Init();
HEADER_SetDefaultTextColor(GUI_YELLOW);
HEADER_SetDefaultFont(&GUI_Font8x8);
hHeader = HEADER_Create(10, 10, 100, 40, WM_HBKWIN, 1234, WM_CF_SHOW, 0);
HEADER_AddItem(hHeader, 50, "Phone", GUI_TA_BOTTOM | GUI_TA_HCENTER);
HEADER_AddItem(hHeader, 50, "Code" , GUI_TA_BOTTOM | GUI_TA_HCENTER);
HEADER_SetBitmapEx(hHeader, 0, &bmPhone, 0, -15);
HEADER_SetBitmapEx(hHeader, 1, &bmCode,  0, -15);
...
```

上述示例的屏幕截图：

HEADER_SetBkColor()

描述

设置给定 HEADER 小工具的背景色。

原型

```
void HEADER_SetBkColor(HEADER_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄
Color	要设置的背景色

HEADER_SetBMP()

描述

设置显示指定项目时使用的位图。

原型

```
void HEADER_SetBMP(HEADER_Handle   hObj,     unsigned int Index,
```



```
                                const void    * pBitmap);
```

参数	描述
hObj	小工具的句柄
Index	HEADER 项目的索引
pBitmap	位图文件数据的指针

其他信息

有关位图文件的更多信息，参见 “显示位图文件”（第 141 页）。

HEADER_SetBMPEx()

描述

设置显示指定项目时使用的位图。

原型

```
void HEADER_SetBMPEx(HEADER_Handle   hObj,    unsigned int Index,
                     const void    * pBitmap,
                     int             x,       int          y);
```

参数	描述
hObj	小工具的句柄
Index	项目的索引
pBitmap	位图文件数据的指针
x	X 方向上的额外偏移
y	Y 方向上的额外偏移

其他信息

有关位图文件的更多信息，参见 “显示位图文件”（第 141 页）。

HEADER_SetDefaultBkColor()

描述

设置创建 HEADER 小工具时使用的默认背景色。

原型

```
GUI_COLOR HEADER_SetDefaultBkColor(GUI_COLOR Color);
```

参数	描述
Color	要使用的背景色

返回值

前一默认背景色。

HEADER_SetDefaultBorderH()

描述

设置创建 HEADER 小工具时使用的水平间距值。水平间距指文本和项目水平边框之间的水平距离，单位像素。

原型

```
int HEADER_SetDefaultBorderH(int Spacing);
```

参数	描述
Spacing	要使用的值

返回值

先前的默认值。

其他信息

仅当新项目的给定宽度为 0 时，水平间距才有效。

HEADER_SetDefaultBorderV()

描述

设置创建 HEADER 小工具时使用的垂直间距值。垂直间距指文本和 HEADER 小工具垂直边框之间的垂直距离，单位像素。

原型

```
int HEADER_SetDefaultBorderV(int Spacing);
```

参数	描述
Spacing	要使用的值

返回值

先前的默认值。

HEADER_SetDefaultCursor()

描述

设置拖动 HEADER 项目宽度时要显示的光标。

原型

```
const GUI_CURSOR * HEADER_SetDefaultCursor(const GUI_CURSOR * pCursor);
```

参数	描述
pCursor	指向拖动 HEADER 项目宽度时将要显示的光标的指针

返回值

指向前一默认光标的指针。

其他信息

在本章开头介绍了 2 种预定义光标。

HEADER_SetDefaultFont()

描述

设置创建 HEADER 小工具时使用的默认字体。

原型

```
const GUI_FONT * HEADER_SetDefaultFont(const GUI_FONT * pFont);
```

参数	描述
pFont	指向要使用字体的指针

返回值

指向前一默认字体的指针。

HEADER_SetDefaultTextColor()

描述

返回创建 HEADER 小工具时使用的默认文本颜色。

原型

```
GUI_COLOR HEADER_SetDefaultTextColor(GUI_COLOR Color);
```

参数	描述
Color	要使用的颜色

返回值

先前的默认值。

HEADER_SetDragLimit()

描述

将拖动分隔线限制设置为开或关。如果限制打开，则只能在小工具区域内拖动分隔线。如果关闭限制，可将其拖出小工具区域。

原型

```
void HEADER_SetDragLimit(HEADER_Handle hObj, unsigned OnOff);
```

参数	描述
hObj	小工具的句柄
OnOff	1 将拖动限制设置为开，0 为关

HEADER_SetFont()

描述

设置在显示给定 HEADER 小工具时使用的字体。

原型

```
void HEADER_SetFont(HEADER_Handle hObj, const GUI_FONT * pFont);
```

参数	描述
hObj	小工具的句柄
pFont	指向要使用字体的指针

HEADER_SetHeight()

描述

设置给定 HEADER 小工具的高度。

原型

```
void HEADER_SetHeight(HEADER_Handle hObj, int Height);
```

参数	描述
hObj	小工具的句柄
Height	新高度

HEADER_SetItemText()

描述

设置显示指定项目时使用的文本。

原型

```
void HEADER_SetItemText(HEADER_Handle   hObj, unsigned int Index,
                        const char    * s);
```

参数	描述
hObj	小工具的句柄
Index	HEADER 项目的索引
s	指向要显示的字符串的指针

其他信息

一个 HEADER 项目可同时包括字符串和位图。

HEADER_SetItemWidth()

描述

设置指定 HEADER 项目的宽度。

原型

```
void HEADER_SetItemWidth(HEADER_Handle hObj, unsigned int Index, int Width);
```

参数	描述
hObj	小工具的句柄
Index	HEADER 项目的索引
Width	新的宽度

HEADER_SetStreamedBitmap()

描述

设置显示指定项目时使用的位图。

原型

```
void HEADER_SetStreamedBitmap(HEADER_Handle             hObj,
                              unsigned int              Index,
                              const GUI_BITMAP_STREAM * pBitmap);
```

参数	描述
hObj	小工具的句柄
Index	项目的索引
pBitmap	指向要显示的流位图数据的指针

其他信息

有关流位图文件的更多信息，请参见 “2-D 图形库”（第 97 页）一章。

HEADER_SetStreamedBitmapEx()

描述

设置显示指定项目时使用的位图。

原型

```
void HEADER_SetStreamedBitmapEx(HEADER_Handle             hObj,
                                unsigned int              Index,
                                const GUI_BITMAP_STREAM * pBitmap,
                                int                       x,
                                int                       y);
```

参数	描述
hObj	小工具的句柄
Index	项目的索引
pBitmap	指向要显示的流位图数据的指针
x	X 方向上的额外偏移
y	Y 方向上的额外偏移

其他信息

有关流位图文件的更多信息，请参见 “2-D 图形库”（第 97 页）一章。

HEADER_SetTextAlign()

描述

设置指定 HEADER 项目的文本对齐方式。

原型

```
void HEADER_SetTextAlign(HEADER_Handle hObj, unsigned int Index, int Align);
```

参数	描述
hObj	小工具的句柄
Index	HEADER 项目的索引
Align	要设置的文本对齐模式。可能为水平对齐标记和垂直对齐标记的组合。

参数 Align 的允许值 （水平和垂直标记可以通过 “OR” 操作进行组合）	
水平对齐	
GUI_TA_LEFT	X 轴方向左对齐 （默认）。
GUI_TA_HCENTER	X 轴方向对中。
GUI_TA_RIGHT	X 轴方向右对齐 （默认）。
垂直对齐	
GUI_TA_TOP	将 Y 轴位置与字符的顶部对齐 （默认）。
GUI_TA_VCENTER	在 Y 轴方向对中。
GUI_TA_BOTTOM	将 Y 轴位置与字体的底部像素行对齐。

HEADER_SetTextColor()

描述

设置显示小工具时使用的文本颜色。

原型

```
void HEADER_SetTextColor(HEADER_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄
Color	要使用的颜色

HEADER_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

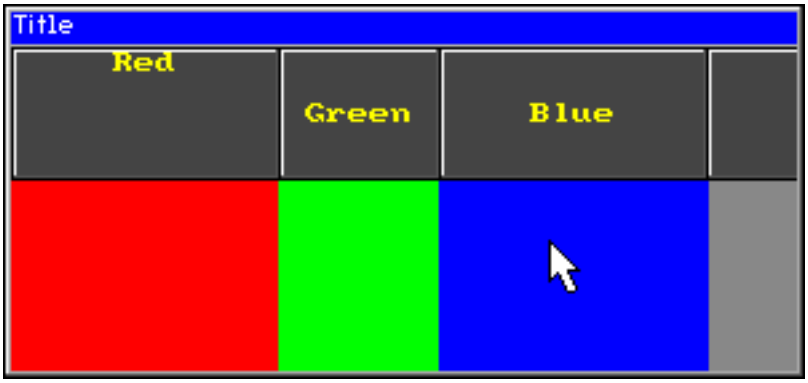
15.10.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_Header.c

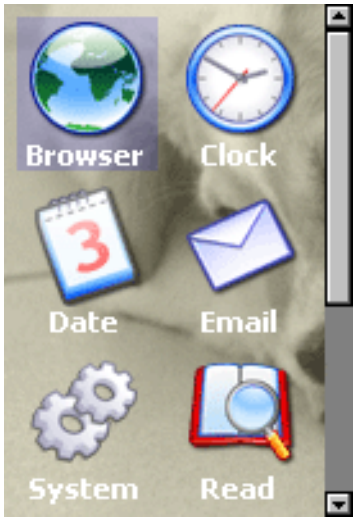
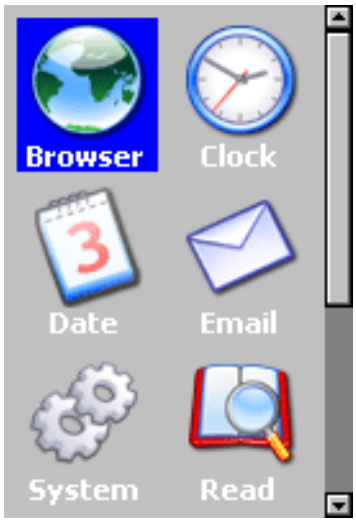
需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_Header.c 的屏幕截图：



15.11 ICONVIEW：图标视图小工具

图标视图小工具可用于基于图标的菜单，手持式设备（如移动电话或便携式管理器）常常需要使用这种菜单。它显示一系列的图标，每个图标都可标注可选文本。图标视图小工具支持透明度及 alpha 混合处理。所以其背景可显示任何内容。当前选定的图标会用纯色或 alpha 混合效果突出显示，后者使背景通体闪亮。需要时可以显示滚动条。

具有透明性的 ICONVIEW	不具有透明性的 ICONVIEW
	

所有 ICONVIEW 相关例程都位于文件 `ICONVIEW*.c`、`ICONVIEW*.h` 中。所有标识符的前缀都是 ICONVIEW。

15.11.1 配置选项

类型	宏	默认值	描述
N	ICONVIEW_BKCOLOR0_DEFAULT	GUI_WHITE	背景颜色，未选定状态。
N	ICONVIEW_BKCOLOR1_DEFAULT	GUI_BLUE	背景色，选中状态。
N	ICONVIEW_TEXTCOLOR0_DEFAULT	GUI_WHITE	文本颜色，未选定状态。
N	ICONVIEW_TEXTCOLOR1_DEFAULT	GUI_WHITE	文本颜色，选中状态。
S	ICONVIEW_FONT_DEFAULT	GUI_Font13_1	绘制标签要使用的字体。
N	ICONVIEW_FRAMEX_DEFAULT	5	图标和小工具左右边框的空白空间。
N	ICONVIEW_FRAMEY_DEFAULT	5	图标和小工具上下边框的空白空间。
N	ICONVIEW_SPACEX_DEFAULT	5	图标之间的水平空白空间。
N	ICONVIEW_SPACEY_DEFAULT	5	图标之间的垂直空白空间。
N	ICONVIEW_ALIGN_DEFAULT	GUI_TA_HCENTER | GUI_TA_BOTTOM	绘制标签要使用的默认对齐方式。

15.11.2 通知代码

以下事件是 ICONVIEW 小工具作为 WM_NOTIFY_PARENT 消息的一部分发送给其父窗口的：

消息	描述
WM_NOTIFICATION_CLICKED	已单击小工具。
WM_NOTIFICATION_RELEASED	已释放小工具。
WM_NOTIFICATION_MOVED_OUT	小工具已被点击，且指针已移出小工具区域并且没有释放。
WM_NOTIFICATION_SCROLL_CHANGED	可选滚动条的滚动位置已更改。
WM_NOTIFICATION_SEL_CHANGED	小工具的选择已经改变。

15.11.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

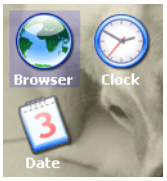
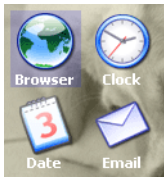
按键	反应
GUI_KEY_RIGHT	将选择移动到下一图标。
GUI_KEY_LEFT	将选择移动到上一图标。
GUI_KEY_DOWN	向下移动选择。
GUI_KEY_UP	向上移动选择。
GUI_KEY_HOME	将选择移动到第一个图标。
GUI_KEY_END	将选择移动到最后一个图标。

15.11.4 ICONVIEW API

下表按字母顺序列出了可用的 emWin ICONVIEW 相关例程。例程的详细说明如下：

例程	描述
ICONVIEW_AddBitmapItem()	向 ICONVIEW 小工具添加新图标。
ICONVIEW_AddStreamedBitmapItem()	向使用流位图的 ICONVIEW 小工具添加新图标。
ICONVIEW_CreateEx()	创建 ICONVIEW 小工具。
ICONVIEW_CreateIndirect()	从资源表项创建 ICONVIEW 小工具。
ICONVIEW_CreateUser()	使用用户数据作为额外字节创建 ICONVIEW 小工具。
ICONVIEW_DeleteItem()	删除现有项目。
ICONVIEW_GetItemText()	检索指定图标视图项目的文本。
ICONVIEW_GetItemUserData()	从指定项目中检索先前存储的用户数据。
ICONVIEW_GetNumItems()	返回给定图标视图中的项目数量。
ICONVIEW_GetSel()	返回当前选定图标的索引。
ICONVIEW_GetUserData()	检索用 ICONVIEW_SetUserData() 设置的数据。
ICONVIEW_InsertBitmapItem()	在给定位置向图标视图小工具插入新图标。
ICONVIEW_InsertStreamedBitmapItem()	使用流位图在给定位置向图标视图小工具插入新图标。
ICONVIEW_SetBitmapItem()	设置指定项目要使用的位图。
ICONVIEW_SetBkColor()	设置背景颜色。
ICONVIEW_SetFont()	设置要用于绘制标签的字体。
ICONVIEW_SetFrame()	设置小工具边框和图标之间的框架大小。
ICONVIEW_SetItemText()	设置指定项目的文本。
ICONVIEW_SetItemUserData()	在指定项目中存储用户数据。
ICONVIEW_SetSel()	设置当前选定内容。
ICONVIEW_SetSpace()	设置图标在 x 或 y 方向上的间距。
ICONVIEW_SetStreamedBitmapItem()	设置当前选定内容。
ICONVIEW_SetTextAlign()	设置要用于绘制标签的对齐方式。
ICONVIEW_SetTextColor()	设置要用于绘制标签的颜色。
ICONVIEW_SetUserData()	设置 ICONVIEW 小工具的额外数据。

ICONVIEW_AddBitmapItem()

之前	之后
	

描述

向小工具添加新的位图图标。

原型

```
int ICONVIEW_AddBitmapItem(ICONVIEW_Handle    hObj,
                           const GUI_BITMAP * pBitmap,
                           const char       * pText);
```

参数	描述
hObj	小工具的句柄。
pBitmap	指向绘制图标所使用的位图结构的指针。
pText	要用于标记图标的文本。

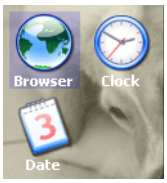
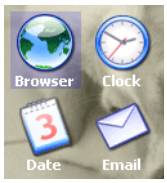
返回值

0 表示成功， != 0 表示错误。

其他信息

注意，位图指针需要保持有效。

ICONVIEW_AddStreamedBitmapItem()

之前	之后
	

描述

向小工具添加新的流位图图标。

原型

```
int ICONVIEW_AddStreamedBitmapItem(ICONVIEW_Handle   hObj,
                                   const void      * pStreamedBitmap,
                                   const char      * pText);
```

参数	描述
hObj	小工具的句柄。
pStreamedBitmap	指向绘制图标所使用的位图流的指针。
pText	要用于标记图标的文本。

返回值

0 表示成功， != 0 表示错误。

其他信息

位图流指针需要保持有效。

ICONVIEW_CreateEx()

描述

在指定位置创建指定尺寸的 ICONVIEW 小工具。

原型

```
ICONVIEW_Handle ICONVIEW_CreateEx(int     x0,         int y0,
                                  int     xSize,      int ySize,
                                  WM_HWIN hParent,    int WinFlags,
                                  int     ExFlags,    int Id,
                                  int     xSizeItem, int ySizeItem);
```

参数	描述
x0	小工具的最左像素 （在父坐标中）。
y0	小工具的最上像素 （在父坐标中）。
xSize	小工具的水平尺寸 （单位：像素）。
ySize	小工具的垂直尺寸 （单位：像素）。
hParent	父窗口的句柄。如果为 0，则新小工具将成为桌面 （顶级窗口）的子窗口。
WinFlags	窗口创建标记。为使小工具立即可见，通常使用 WM_CF_SHOW （有关可用参数值的清单，请参见 “WM_CreateWindow()”（第 309 页））。
ExFlags	（参见下表）
Id	小工具的窗口 ID。
xSizeItem	图标的水平尺寸 （单位：像素）。
ySizeItem	图标的垂直尺寸 （单位：像素）。

参数 ExFlags 的允许值	
0	（默认）
ICONVIEW_CF_AUTOSCROLLBAR_V	如果小工具区域太小，不足以显示所有图标，则将增加垂直滚动条。

返回值

新小工具的句柄；函数失败时为 0。

其他信息

如果小工具应是透明的，则应使用 OR 组合参数 WinFlags 和 WM_CF_HASTRANS。

ICONVIEW_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。

ICONVIEW_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细描述，可参见函数 ICONVIEW_CreateEx()。

ICONVIEW_DeleteItem()

描述

删除 ICONVIEW 小工具的现有项目。

原型

```
void ICONVIEW_DeleteItem(ICONVIEW_Handle hObj, unsigned Index);
```

参数	描述
hObj	小工具的句柄。
Index	要删除的项目的索引。

ICONVIEW_GetItemText()

描述

检索指定图标视图项目的文本。

原型

```
int ICONVIEW_GetItemText(ICONVIEW_Handle hObj,    int Index,
                         char *          pBuffer, int MaxSize);
```

参数	描述
hObj	小工具的句柄。
Index	要删除的项目的索引。
pBuffer	用于检索文本的缓冲区。
MaxSize	复制到缓冲区的文本的最大长度。

返回值

返回实际复制的文本的长度。

ICONVIEW_GetItemUserData()

描述

从指定项目中检索先前存储的用户数据。

原型

```
U32 ICONVIEW_GetItemUserData(ICONVIEW_Handle hObj, int Index);
```

参数	描述
hObj	小工具的句柄。
Index	项目的索引。

返回值

项目中存储的用户数据，如 U32。

ICONVIEW_GetNumItems()

描述

返回给定图标视图中的项目数量。

原型

```
int ICONVIEW_GetNumItems(ICONVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

项目数量。

ICONVIEW_GetSel()

描述

返回当前选定图标的以零为基准的索引。

原型

```
int ICONVIEW_GetSel(ICONVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

当前选定图标的以零为基准的索引。

ICONVIEW_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

ICONVIEW_InsertBitmapItem()

描述

向小工具中插入新的位图图标。有关屏幕截图，请参见 “ICONVIEW_AddBitmapItem()”（第 485 页）。

原型

```
int ICONVIEW_InsertBitmapItem(ICONVIEW_Handle    hObj,
                              const GUI_BITMAP * pBitmap,
                              const char       * pText
                              int                Index);
```

参数	描述
hObj	小工具的句柄。
pBitmap	指向绘制图标所使用的位图结构的指针。
pText	要用于标记图标的文本。
Index	插入项目的索引位置。

返回值

0 表示成功， != 0 表示错误。

其他信息

注意，位图指针需要保持有效。

ICONVIEW_InsertStreamedBitmapItem()

描述

向小工具中插入新的流位图图标。有关屏幕截图，请参见 “ICONVIEW_AddBitmapItem()”（第 485 页）。

原型

```
int ICONVIEW_InsertStreamedBitmapItem(ICONVIEW_Handle   hObj,
                                      const void      * pStreamedBitmap,
                                      const char      * pText,
                                      int               Index);
```

参数	描述
hObj	小工具的句柄。
pStreamedBitmap	指向绘制图标所使用的位图流的指针。
pText	要用于标记图标的文本。
Index	插入项目的索引位置。

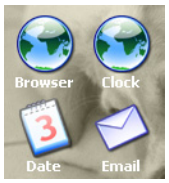
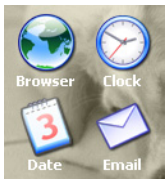
返回值

0 表示成功， != 0 表示错误。

其他信息

位图流指针需要保持有效。

ICONVIEW_SetBitmapItem()

之前	之后
	

描述

设置指定项目要使用的位图。

原型

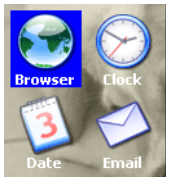
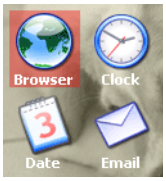
```
void ICONVIEW_SetBitmapItem(ICONVIEW_Handle    hObj,
                            int                Index,
                            const GUI_BITMAP * pBitmap);
```

参数	描述
hObj	小工具的句柄。
Index	项目的索引。
pBitmap	指向要使用的位图的指针。

其他信息

位图结构的指针需要保持有效。

ICONVIEW_SetBkColor()

之前	之后
	

描述

设置小工具的背景色。

原型

```
void ICONVIEW_SetBkColor(ICONVIEW_Handle hObj, int Index, GUI_COLOR Color);
```

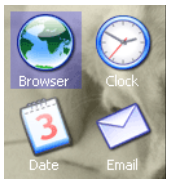
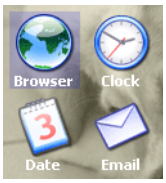
参数	描述
hObj	小工具的句柄。
Index	（参见下表）
Color	绘制背景要使用的颜色。

参数 Index 的允许值	
ICONVIEW_CI_BK	绘制小工具背景要使用的颜色。
ICONVIEW_CI_SEL	用于突出显示当前选定项目的颜色。

其他信息

32 位颜色值的前 8 位可用于 alpha 混合处理效果。有关 alpha 混合处理的更多信息，请参见 “GUI_SetAlpha()”（第 111 页）。

ICONVIEW_SetFont()

之前	之后
	

描述

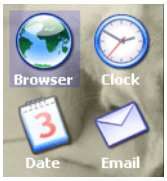
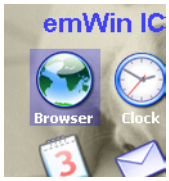
设置要用于绘制图标标签的字体。

原型

```
void ICONVIEW_SetFont(ICONVIEW_Handle               hObj,
                      const GUI_FONT GUI_UNI_PTR * pFont);
```

参数	描述
hObj	小工具的句柄。
pFont	指向绘制图标标签要使用的 GUI_FONT 结构的指针。

ICONVIEW_SetFrame()

之前	之后
	

描述

设置小工具边框和图标之间的框架大小。

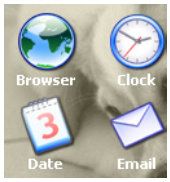
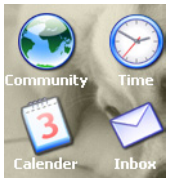
原型

```
void ICONVIEW_SetFrame(ICONVIEW_Handle hObj,
                       int             Coord,
                       int             Value);
```

参数	描述
hObj	小工具的句柄。
Coord	此参数的允许值见下文。
Value	要设置的距离。

参数 Coord 的允许值	
GUI_COORD_X	X 方向。
GUI_COORD_Y	Y 方向。

ICONVIEW_SetItemText()

之前	之后
	

描述

设置指定项目的文本。

原型

```
void ICONVIEW_SetItemText(ICONVIEW_Handle   hObj,
                          int               Index,
                          const char      * pText);
```

参数	描述
hObj	小工具的句柄。
Index	项目的索引。
pText	指向要使用的文本的指针。

ICONVIEW_SetItemUserData()

描述

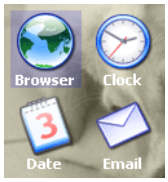
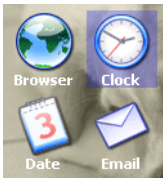
在指定项目中存储用户数据。

原型

```
void ICONVIEW_SetItemUserData(ICONVIEW_Handle hObj,
                              int             Index,
                              U32             UserData);
```

参数	描述
hObj	小工具的句柄。
Index	项目的索引。
UserData	要存储的 32 位用户数据。

ICONVIEW_SetSel()

之前	之后
	

描述

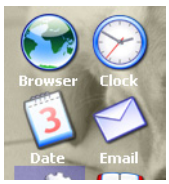
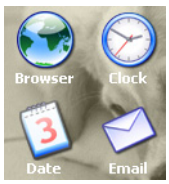
设置当前选定内容。

原型

```
void ICONVIEW_SetSel(ICONVIEW_Handle hObj, int Sel);
```

参数	描述
hObj	小工具的句柄。
Sel	新选择。

ICONVIEW_SetSpace()

之前	之后
	

描述

设置图标在 x 或 y 方向上的间距。

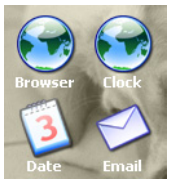
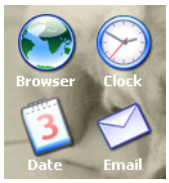
原型

```
void ICONVIEW_SetSpace(ICONVIEW_Handle hObj, int Coord, int Value);
```

参数	描述
hObj	小工具的句柄。
Coord	此参数的允许值见下文。
Value	要设置的距离。

参数 Coord 的允许值	
GUI_COORD_X	X 方向。
GUI_COORD_Y	Y 方向。

ICONVIEW_SetStreamedBitmapItem()

之前	之后
	

描述

设置指定项目要使用的流位图。

原型

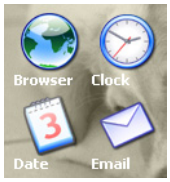
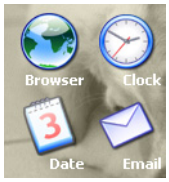
```
void ICONVIEW_SetStreamedBitmapItem(ICONVIEW_Handle   hObj,
                                    int               Index,
                                    const void      * pStreamedBitmap);
```

参数	描述
hObj	小工具的句柄。
Index	项目的索引。
pStreamedBitmap	指向要使用的位图流的指针。

其他信息

位图流指针需要保持有效。

ICONVIEW_SetTextAlign()

之前	之后
	

描述

设置要用于绘制标签的颜色。

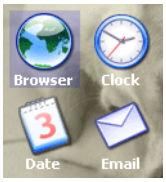
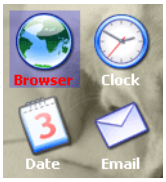
原型

```
void ICONVIEW_SetTextAlign(ICONVIEW_Handle hObj, int TextAlign);
```

参数	描述
hObj	小工具的句柄。
TextAlign	参见下表。

参数 TextAlign 的允许值 （水平和垂直标记可以通过 “OR” 操作进行组合）	
水平对齐	
GUI_TA_LEFT	X 轴方向左对齐 （默认）。
GUI_TA_HCENTER	X 轴方向对中。
GUI_TA_RIGHT	X 轴方向右对齐 （默认）。
垂直对齐	
GUI_TA_TOP	将 Y 轴位置与字符的顶部对齐 （默认）。
GUI_TA_VCENTER	在 Y 轴方向对中。
GUI_TA_BOTTOM	将 Y 轴位置与字体的底部像素行对齐。

ICONVIEW_SetTextColor()

之前	之后
	

描述

设置要用于绘制标签的颜色。

原型

```
void ICONVIEW_SetTextColor(ICONVIEW_Handle hObj,   int Index,
                           GUI_COLOR       Color);
```

参数	描述
hObj	小工具的句柄
Index	（参见下表）
Color	要使用的颜色

参数 Index 的允许值	
ICONVIEW_CI_UNSEL	绘制处于未选定状态的标签所使用的颜色。
ICONVIEW_CI_SEL	绘制处于被选定状态的标签所使用的颜色。

ICONVIEW_SetUserData()

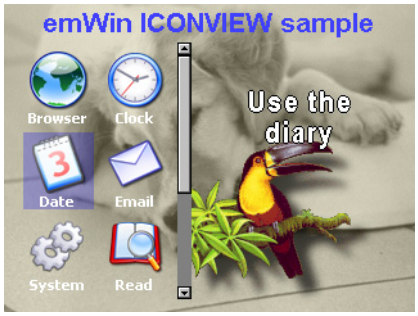
在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

15.11.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_IconView

WIDGET_Iconview.c 的屏幕截图：



15.12 LISTBOX：列表框小工具

列表框用于选择列表的一个元素。创建的列表框可以没有环绕的框架窗口，如下所示，或者作为 FRAMEWIN 小工具的子窗口建立 （参见本节末尾的其他屏幕截图）。列表框中的项目被选定后，会突出显示。注意，所选项目的背景色取决于列表框窗口是否有输入焦点。

有焦点的列表框	无焦点的列表框
	

所有LISTBOX相关例程都位于文件LISTBOX*.c、LISTBOX.h中。所有标识符的前缀都是LISTBOX。

15.12.1 配置选项

类型	宏	默认值	描述
N	LISTBOX_BKCOLOR0_DEFAULT	GUI_WHITE	背景颜色，未选定状态。
N	LISTBOX_BKCOLOR1_DEFAULT	GUI_GRAY	背景颜色，选定状态，无焦点。
N	LISTBOX_BKCOLOR2_DEFAULT	GUI_BLUE	背景颜色，选定状态，有焦点。
S	LISTBOX_FONT_DEFAULT	&GUI_Font13_1	所使用的字体。
N	LISTBOX_TEXTCOLOR0_DEFAULT	GUI_BLACK	文本颜色，未选定状态。
N	LISTBOX_TEXTCOLOR1_DEFAULT	GUI_WHITE	文本颜色，选定状态，无焦点。
N	LISTBOX_TEXTCOLOR2_DEFAULT	GUI_WHITE	文本颜色，选定状态，有焦点。

15.12.2 通知代码

以下事件是列表框小工具作为 WM_NOTIFY_PARENT 消息的一部分发送给其父窗口的：

消息	描述
WM_NOTIFICATION_CLICKED	列表框已被点击。
WM_NOTIFICATION_RELEASED	列表框已被释放。
WM_NOTIFICATION_MOVED_OUT	列表框已被点击，且指针已从列表框处移开，没有释放。
WM_NOTIFICATION_SCROLL_CHANGED	可选滚动条的滚动位置已更改。
WM_NOTIFICATION_SEL_CHANGED	列表框的选择内容已更改。

15.12.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_SPACE	如果小工具在多选择模式下操作，此键切换当前选定项目的状态。
GUI_KEY_RIGHT	如果列表框项目的最大X尺寸超过了列表框本身，此键将列表框中的内容滚动至左边。
GUI_KEY_LEFT	如果列表框项目的最大X尺寸超过了列表框本身，此键将列表框中的内容滚动至右边。
GUI_KEY_DOWN	选择栏下移。
GUI_KEY_UP	选择栏上移。

15.12.4 LISTBOX API

下表按字母顺序列出了可用的 emWin LISTBOX 相关例程。例程的详细说明如下：

例程	描述
LISTBOX_AddString()	向列表框添加项目。
LISTBOX_Create()	创建 LISTBOX 小工具。（弃用）
LISTBOX_CreateAsChild()	创建一个 LISTBOX 小工具作为子窗口。（弃用）
LISTBOX_CreateEx()	创建 LISTBOX 小工具。
LISTBOX_CreateIndirect()	从资源表项创建 LISTBOX 小工具。
LISTBOX_CreateUser()	使用额外字节作为用户数据创建 LISTBOX 小工具。
LISTBOX_DecSel()	减小选定范围。
LISTBOX_DeleteItem()	删除某元素。
LISTBOX_GetDefaultBkColor()	返回 LISTBOX 小工具的默认背景色。
LISTBOX_GetDefaultFont()	返回 LISTBOX 小工具的默认字体。
LISTBOX_GetDefaultScrollStepH()	返回要水平滚动的默认像素数值。
LISTBOX_GetDefaultTextAlign()	返回新列表框的默认文本对齐方式。
LISTBOX_GetDefaultTextColor()	返回新列表框的默认文本颜色。
LISTBOX_GetFont()	返回列表框的字体。
LISTBOX_GetItemDisabled()	返回给定项目的禁用状态。
LISTBOX_GetItemSel()	返回 LISTBOX 项目的选取状态。
LISTBOX_GetItemText()	返回列表框项目的文本。
LISTBOX_GetMulti()	返回表示多选择模式是否激活的值。
LISTBOX_GetNumItems()	返回列表框的项目数。
LISTBOX_GetScrollStepH()	返回要水平滚动的像素数值。
LISTBOX_GetSel()	返回选定的项目数。
LISTBOX_GetTextAlign()	返回 LISTBOX 的文本对齐方式。
LISTBOX_GetUserData()	检索用 LISTBOX_SetUserData() 设置的数据。
LISTBOX_IncSel()	增加选定范围。
LISTBOX_InsertString()	插入某元素。
LISTBOX_InvalidateItem()	使自画 LISTBOX 的项目失效。
LISTBOX_OwnerDraw()	用于绘制 LISTBOX 项目的默认函数。
LISTBOX_SetAutoScrollH()	激活自动使用水平滚动条。
LISTBOX_SetAutoScrollV()	激活自动使用垂直滚动条。
LISTBOX_SetBkColor()	设置背景颜色。
LISTBOX_SetDefaultBkColor()	设置 LISTBOX 小工具的默认背景色。
LISTBOX_SetDefaultFont()	改变 LISTBOX 小工具的默认字体。
LISTBOX_SetDefaultScrollStepH()	设置水平滚动的默认像素数值。
LISTBOX_SetDefaultTextAlign()	设置新创建的 LISTBOX 小工具所使用的默认文本对齐方式。
LISTBOX_SetDefaultTextColor()	设置 LISTBOX 小工具的默认文本颜色。
LISTBOX_SetFont()	选择字体。
LISTBOX_SetItemDisabled()	设置给定项目的禁用状态。
LISTBOX_SetItemSel()	设置给定项目的选择状态。
LISTBOX_SetItemSpacing()	设置项目之间的间距。
LISTBOX_SetMulti()	设置多选择模式的开 / 关。
LISTBOX_SetOwnerDraw()	启用列表框为自画。
LISTBOX_SetScrollbarColor()	设置可选滚动条的颜色。
LISTBOX_SetScrollbarWidth()	设置 LISTBOX 使用的滚动条的宽度。
LISTBOX_SetScrollStepH()	设置水平滚动的像素数值。
LISTBOX_SetSel()	设置选择的项目。
LISTBOX_SetString()	设置元素的文本。

例程	描述
LISTBOX_SetTextAlign()	设置 LISTBOX 的文本对齐方式。
LISTBOX_SetTextColor()	设置前景颜色。
LISTBOX_SetUserData()	设置 LISTBOX 小工具的额外数据。

LISTBOX_AddString()

描述

向已存在的列表框添加项目。

原型

```
void LISTBOX_AddString(LISTBOX_Handle hObj, const char * s);
```

参数	描述
hObj	列表框的句柄。
s	要显示的文本。

LISTBOX_Create()

描述

在指定位置创建指定尺寸的 LISTBOX 小工具。

原型

```
LISTBOX_Handle LISTBOX_Create(const GUI_ConstString * ppText,
                              int   x0, int y0,
                              int   xSize, int ySize,
                              int   Flags);
```

参数	描述
ppText	包含要显示的元素的字符串阵列指针。
x0	列表框的最左像素 （在父坐标中）。
y0	列表框的最上像素 （在父坐标中）。
xSize	列表框的水平尺寸 （单位：像素）。
ySize	列表框的垂直尺寸 （单位：像素）。
Flags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。

返回值

已创建 LISTBOX 小工具的句柄；函数失败时为 0。

其他信息

如果参数 ySize 大于绘制小工具内容所要求的空间，则 Y 尺寸将被降低到所需值。这对参数 Xsize 同样适用。

LISTBOX_CreateAsChild()

描述

创建一个 LISTBOX 小工具作为子窗口。

原型

```
LISTBOX_Handle LISTBOX_CreateAsChild(const   GUI_ConstString * ppText,
                                     WM_HWIN hParent,
                                     int     x0,       int y0,
                                     int     xSize,    int ySize,
                                     int     Flags);
```

参数	描述
ppText	包含要显示的元素的字符串阵列指针。
hParent	父窗口的句柄。
x0	列表框相对于父窗口的 X 位置。
y0	列表框相对于父窗口的 Y 位置。
xSize	列表框的水平尺寸 （单位：像素）。
ySize	列表框的垂直尺寸 （单位：像素）。
Flags	窗口创建标记 （参见 LISTBOX_Create()）。

返回值

已创建 LISTBOX 小工具的句柄；函数失败时为 0。

其他信息

如果参数 ySize 大于绘制小工具内容所要求的空间，则 Y 尺寸将被降低到所需值。如果 ySize = 0，则小工具的 Y 尺寸将设置为父窗口客户区的 Y 尺寸。这对参数 Xsize 同样适用。

LISTBOX_CreateEx()

描述

在指定位置创建指定尺寸的 LISTBOX 小工具。

原型

```
LISTBOX_Handle LISTBOX_CreateEx(int     x0,       int y0,
                                int     xsize,    int ysize,
                                WM_HWIN hParent, int WinFlags,
                                int     ExFlags, int Id,
                                const   GUI_ConstString * ppText);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新的 HEADER 小工具将是桌面 （顶级窗口）的子项。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW（有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	未使用，保留供日后使用。
Id	小工具的窗口 ID。
ppText	包含要显示的元素的字符串阵列指针。

返回值

已创建 LISTBOX 小工具的句柄；函数失败时为 0。

LISTBOX_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。资源的元素 Flags 和 Para 将省略，因为未使用参数。

LISTBOX_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细描述，可参见函数 LISTBOX_CreateEx()。

LISTBOX_DecSel()

描述

递减列表框选取 （将指定列表框的选择条向上移动一个项目）。

原型

```
void LISTBOX_DecSel(LISTBOX_Handle hObj);
```

参数	描述
hObj	列表框的句柄。

其他信息

需要注意的是，项目编号始终从顶部以0值开始；因此，减小选定范围实际上会将选定范围上移一行。

LISTBOX_DeleteItem()

描述

从列表框删除一个元素。

原型

```
void LISTBOX_DeleteItem(LISTBOX_Handle hObj, unsigned int Index);
```

参数	描述
hObj	列表框的句柄。
Index	要删除元素的以零为基准的索引。

LISTBOX_GetDefaultBkColor()

描述

返回新 LISTBOX 小工具的默认背景色。

原型

```
GUI_COLOR LISTBOX_GetDefaultBkColor(unsigned Index);
```

参数	描述
Index	以零为基准的背景颜色索引 （参见下表）

参数 Index 的允许值	
LISTBOX_CI_UNSEL	未选定元素。
LISTBOX_CI_SEL	选定元素，无焦点。
LISTBOX_CI_SELFOCUS	选定元素，有焦点。

返回值

新 LISTBOX 小工具的默认背景色。

LISTBOX_GetDefaultFont()

描述

返回创建 LISTBOX 小工具所使用的默认字体。

原型

```
const GUI_FONT * LISTBOX_GetDefaultFont(void);
```

返回值

默认字体的指针。

LISTBOX_GetDefaultScrollStepH()

描述

返回创建 LISTBOX 小工具所使用的默认水平滚动级别。水平滚动级别定义了所需滚动的像素数。

原型

```
int LISTBOX_GetDefaultScrollStepH(void);
```

返回值

默认水平滚动级别。

LISTBOX_GetDefaultTextAlign()

描述

返回新创建的 LISTBOX 小工具所使用的默认文本对齐方式。

原型

```
int LISTBOX_GetDefaultTextAlign(void);
```

返回值

新创建的 LISTBOX 小工具所使用的默认文本对齐方式。

其他信息

更多详细信息，请参阅 “LISTBOX_SetTextAlign()”（第 512 页）。

LISTBOX_GetDefaultTextColor()

描述

返回新创建的 LISTBOX 小工具所使用的默认文本颜色。

原型

```
GUI_COLOR LISTBOX_GetDefaultTextColor(unsigned Index);
```

参数	描述
Index	以零为基准的文本颜色索引（参见下表）

参数 Index 的允许值	
LISTBOX_CI_UNSEL	未选定元素。
LISTBOX_CI_SEL	选定元素，无焦点。
LISTBOX_CI_SELFOCUS	选定元素，有焦点。

返回值

新创建的 LISTBOX 小工具所使用的默认文本颜色。

LISTBOX_GetFont()

描述

返回显示列表框文本所使用的字体指针。

原型

```
const GUI_FONT * LISTBOX_GetFont(LISTBOX_Handle hObj);
```

参数	描述
hObj	列表框的句柄。

返回值

显示列表框文本所使用的字体指针。

LISTBOX_GetItemDisabled()

描述

返回指定的列表框项目是否已禁用。

原型

```
int LISTBOX_GetItemDisabled(LISTBOX_Handle hObj, unsigned Index);
```

参数	描述
hObj	列表框的句柄。
Index	以零为基准的项目索引。

返回值

如果项目已禁用，则返回值为 1 ；如果项目未禁用，则返回值为 0。

LISTBOX_GetItemSel()

描述

返回指定列表框项目的选择状态。 LISTBOX 项目的选择状态只能在多重选择模式下进行修改。

原型

```
int LISTBOX_GetItemSel(LISTBOX_Handle hObj, unsigned int Index);
```

参数	描述
hObj	列表框的句柄。
Index	以零为基准的项目索引。

返回值

如果项目已选定，则返回值为 1 ；如果项目未选定，则返回值为 0。

LISTBOX_GetItemText()

描述

返回指定列表框项目的文本。

原型

```
void LISTBOX_GetItemText(LISTBOX_Handle   hObj,    unsigned Index,
                         char           * pBuffer, int      MaxSize);
```

参数	描述
hObj	列表框的句柄。
Index	以零为基准的项目索引。
pBuffer	存储项目文本所使用的缓冲区指针。
MaxSize	缓冲区大小。

其他信息

该函数将所指定的列表框项目的文本复制到指定的缓冲区中。

LISTBOX_GetMulti()

描述

返回指定列表框的多重选择模式是否处于活动状态。

原型

```
int LISTBOX_GetMulti(LISTBOX_Handle hObj);
```

参数	描述
hObj	列表框的句柄。

返回值

如果处于活动状态，则返回值为 1；如果未处于活动状态，则返回值为 0。

LISTBOX_GetNumItems()

描述

返回指定列表框中的项目数。

原型

```
unsigned LISTBOX_GetNumItems(LISTBOX_Handle hObj);
```

参数	描述
hObj	列表框的句柄。

返回值

列表框中的项目数。

LISTBOX_GetScrollStepH()

描述

返回指定列表框的水平滚动级别。

原型

```
int LISTBOX_GetScrollStepH(LISTBOX_Handle hObj);
```

参数	描述
hObj	列表框的句柄。

返回值

指定列表框的水平滚动级别。

LISTBOX_GetSel()

描述

返回在指定列表框中当前所选项的索引 （以零为基准）。在多重选择模式下，该函数将返回焦点元素的索引。

原型

```
int LISTBOX_GetSel(LISTBOX_Handle hObj);
```

参数	描述
hObj	列表框的句柄。

返回值

当前所选项的索引 （以零为基准）。

其他信息

如果未选定任何元素，则函数将返回 -1。

LISTBOX_GetTextAlign()

描述

返回指定的 LISTBOX 小工具所使用的文本对齐方式。

原型

```
int LISTBOX_GetTextAlign(LISTBOX_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

指定的 LISTBOX 小工具所使用的文本对齐方式。

其他信息

更多详细信息，请参阅 “LISTBOX_SetTextAlign()”（第 512 页）。

LISTBOX_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

LISTBOX_IncSel()

描述

增加列表框选定范围 （将指定列表框的选择栏下移一个项目）。

原型

```
void LISTBOX_IncSel(LISTBOX_Handle hObj);
```

参数	描述
hObj	列表框的句柄。

其他信息

需要注意的是，项目编号始终从顶部以0值开始；因此，增加选定范围实际上会将选定范围下移一行。

LISTBOX_InsertString()

描述

将元素插入列表框。

原型

```
void LISTBOX_InsertString(LISTBOX_Handle hObj,   const char * s,
                          unsigned int   Index);
```

参数	描述
hObj	列表框的句柄。
s	要插入的字符串指针。
Index	要插入的以零为基准的元素索引。

LISTBOX_InvalidateItem()

描述

使自绘列表框的项目无效。

原型

```
void LISTBOX_InvalidateItem(LISTBOX_Handle hObj, int Index);
```

参数	描述
hObj	列表框的句柄。
Index	即将无效的以零为基准的元素索引或 LISTBOX_ALL_ITEMS （如果所有项目都将无效。）

其他信息

只有在自绘列表框项已更改的情况下，才需要调用该函数。如果已使用列表框 API 函数（比如：LISTBOX_SetString()）修改了列表框项，则无需调用 LISTBOX_InvalidateItem()。如果用户确定（比如：项目的垂直大小已更改），则需要调用相关函数。换言之，如果未使用列表框 API 函数对项目进行修改，则需要调用该函数。

LISTBOX_ALL_ITEMS

如果列表框的所有项都无效，则将该项定义作为索引参数。

LISTBOX_OwnerDraw()

描述

处理 LISTBOX 条目的默认函数。

原型

```
int LISTBOX_OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

参数	描述
pDrawItemInfo	WIDGET_ITEM_DRAW_INFO 结构的指针。

其他信息

如果已使用了 LISTBOX_SetOwnerDraw()，则该函数十分有用。在绘图函数中，可使用该函数来检索 LISTBOX 条目的 x 轴的原始大小和 / 或显示 LISTBOX 条目的文本，并且所有未处理的命令都应当调用该函数。
有关详细信息，请参阅介绍自绘小工具LISTBOX_SetOwnerDraw()的相关章节以及所提供的示例。

LISTBOX_SetAutoScrollH()

描述

启用 / 禁用自动使用水平滚动条。

原型

```
void LISTBOX_SetAutoScrollH(LISTBOX_Handle hObj, int OnOff);
```

参数	描述
hObj	列表框的句柄。
OnOff	（参见下表）

参数 OnOff 的允许值	
0	禁用自动使用水平滚动条。
1	启用自动使用水平滚动条。

其他信息

如果启用，则列表框会检查是否所有元素都符合列表框。如果未启用该功能，则水平滚动条将附加到窗口。

LISTBOX_SetAutoScrollV()

描述

启用 / 禁用自动使用垂直滚动条。

原型

```
void LISTBOX_SetAutoScrollV(LISTBOX_Handle hObj, int OnOff);
```

参数	描述
hObj	列表框的句柄。
OnOff	（参见下表）

参数 OnOff 的允许值	
0	禁用自动使用垂直滚动条。
1	启用自动使用垂直滚动条。

其他信息

如果启用，则列表框会检查是否所有元素都符合列表框。如果未启用，则将添加垂直滚动条。

LISTBOX_SetBkColor()

描述

设置列表框的背景颜色。

原型

```
void LISTBOX_SetBkColor(LISTBOX_Handle hObj, unsigned int Index,
                        GUI_COLOR Color);
```

参数	描述
hObj	列表框的句柄。
Index	背景颜色的索引 （参见下表）。
Color	要设置的颜色。

参数 Index 的允许值	
LISTBOX_CI_UNSEL	未选定元素。
LISTBOX_CI_SEL	选定元素，无焦点。
LISTBOX_CI_SELFOCUS	选定元素，有焦点。
LISTBOX_CI_DISABLED	禁用的元素。

LISTBOX_SetDefaultBkColor()

描述

设置新创建的 LISTBOX 小工具所使用的默认背景颜色。

原型

```
void LISTBOX_SetDefaultBkColor(unsigned Index, GUI_COLOR Color);
```

参数	描述
Index	以零为基准的背景颜色索引（参见下表）
Color	所需的背景颜色。

参数 Index 的允许值	
LISTBOX_CI_UNSEL	未选定元素。
LISTBOX_CI_SEL	选定元素，无焦点。
LISTBOX_CI_SELFOCUS	选定元素，有焦点。
LISTBOX_CI_DISABLED	禁用的元素。

LISTBOX_SetDefaultFont()

描述

设置创建 LISTBOX 小工具所使用的默认字体。

原型

```
void LISTBOX_SetDefaultFont(const GUI_FONT * pFont)
```

参数	描述
pFont	字体的指针。

LISTBOX_SetDefaultScrollStepH()

描述

设置创建 LISTBOX 小工具时所使用的默认水平滚动级别。

原型

```
void LISTBOX_SetDefaultScrollStepH(int Value);
```

参数	描述
Value	要滚动的像素数。

LISTBOX_SetDefaultTextAlign()

描述

设置新创建的 LISTBOX 小工具所使用的默认文本对齐方式。

原型

```
void LISTBOX_SetDefaultTextAlign(int Align);
```

参数	描述
Align	新创建的 LISTBOX 小工具所使用的默认文本对齐方式。

其他信息

更多详细信息，请参阅 “LISTBOX_SetTextAlign()”（第 512 页）。

LISTBOX_SetDefaultTextColor()

描述

设置新创建的 LISTBOX 小工具所使用的默认文本颜色。

原型

```
void LISTBOX_SetDefaultTextColor(unsigned Index, GUI_COLOR Color);
```

参数	描述
Index	以零为基准的文本颜色索引 （参见下表）
Color	所需的文本颜色。

参数 Index 的允许值	
LISTBOX_CI_UNSEL	未选定元素。
LISTBOX_CI_SEL	选定元素，无焦点。
LISTBOX_CI_SELFOCUS	选定元素，有焦点。

LISTBOX_SetFont()

描述

设置列表框的字体。

原型

```
void LISTBOX_SetFont(LISTBOX_Handle hObj, const GUI_FONT* pfont);
```

参数	描述
hObj	列表框的句柄。
pFont	字体的指针。

LISTBOX_SetItemDisabled()

描述

修改指定列表框项目的禁用状态。

原型

```
void LISTBOX_SetItemDisabled(LISTBOX_Handle hObj, unsigned Index,
                             int            OnOff);
```

参数	描述
hObj	列表框的句柄。
Index	以零为基准的列表框项的索引。
OnOff	1 代表禁用， 0 代表未禁用。

其他信息

在滚动列表框时，将跳过禁用的项目。无法滚动至已禁用的列表框项。

LISTBOX_SetItemSel()

描述

修改指定列表框项的选择状态。

原型

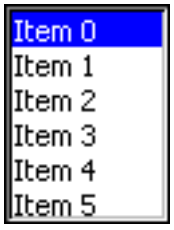
```
void LISTBOX_SetItemSel(LISTBOX_Handle hObj, unsigned Index, int OnOff);
```

参数	描述
hObj	列表框的句柄。
Index	以零为基准的列表框项的索引。
OnOff	1 代表选定， 0 代表未选定。

其他信息

只有在使用多重选择模式时，才能设置列表框项的选择状态。另请参见 LISTBOX_SetMulti()。

LISTBOX_SetItemSpacing()

之前	之后
	

描述

设置列表框项下方的额外间距。

原型

```
void LISTBOX_SetItemSpacing(LISTBOX_Handle hObj, unsigned Value);
```

参数	描述
hObj	列表框的句柄。
Value	作为各项目间额外间距的像素数。

LISTBOX_SetMulti()

描述

开启或关闭 LISTBOX 的多重选择模式。

原型

```
void LISTBOX_SetMulti(LISTBOX_Handle hObj, int Mode);
```

参数	描述
hObj	列表框的句柄。
Mode	0 代表关闭， 1 代表开启。

其他信息

如果处于多重选择模式，则列表框就具有多个选定元素。如果使用空格键，则可切换列表框项的选择状态。

LISTBOX_SetOwnerDraw()

描述

将列表框设置为自绘列表框。

原型

```
void LISTBOX_SetOwnerDraw(LISTBOX_Handle          hObj,
                          WIDGET_DRAW_ITEM_FUNC * pfDrawItem);
```

参数	描述
hObj	列表框的句柄。
pfDrawItem	自绘函数的指针。

其他信息

如果必须绘制列表框项并且需要该列表框项 x 轴或 y 轴的大小，则该函数可将函数指针设置为由小工具调用的函数。通过该函数可以将任何内容绘制为列表框项，而不仅仅是纯机器码。pfDrawItem 是 WIDGET_DRAW_ITEM_FUNC 型的应用程序定义的函数指针，该内容在本章的开头部分已作了介绍。

用户定义的自绘函数的结构

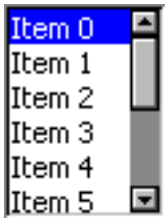
典型自绘函数的结构如下所示。假设LISTBOX条目比默认函数所绘制的项目宽30像素，但高度相同：

```
static int _OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_GET_XSIZE:
    return LISTBOX_OwnerDraw(pDrawItemInfo) + 30; /* Returns the default xsize+10 */
  case WIDGET_ITEM_DRAW:

    /* Your code to be added to draw the LISTBOX item */

    return 0;
  }
  return LISTBOX_OwnerDraw(pDrawItemInfo);    /* Def. function for unhandled cmds */
}
```

示例

该示例的源代码在这些示例中为 WIDGET_ListBoxOwnerDraw。

LISTBOX_SetScrollbarColor()

之前	之后
	

描述

设置可选滚动条的颜色。

原型

```
void LISTBOX_SetScrollbarColor(LISTBOX_Handle hObj,
                               unsigned int   Index,
                               GUI_COLOR      Color);
```

参数	描述
hObj	小工具的句柄
Index	目标项的索引 （参见下表）。
Color	要使用的颜色。

参数 Index 的允许值	
SCROLLBAR_CI_THUMB	缩略图区域的颜色。
SCROLLBAR_CI_SHAFT	轴的颜色。
SCROLLBAR_CI_ARROW	箭头颜色。

LISTBOX_SetScrollbarWidth()

描述

设置指定列表框所使用的滚动条宽度。

原型

```
void LISTBOX_SetScrollbarWidth(LISTBOX_Handle hObj, unsigned Width);
```

参数	描述
hObj	列表框的句柄。
Width	指定列表框所使用的滚动条宽度。

LISTBOX_SetScrollStepH()

描述

设置指定列表框的水平滚动级别。水平滚动级别定义了所需滚动的像素数。

原型

```
void LISTBOX_SetScrollStepH(LISTBOX_Handle hObj, int Value);
```

参数	描述
hObj	列表框的句柄。
Value	要滚动的像素数。

LISTBOX_SetSel()

描述

设置指定列表框的选定项目。

原型

```
void LISTBOX_SetSel(LISTBOX_Handle hObj, int Sel);
```

参数	描述
hObj	列表框的句柄。
Sel	要选择的元素。

LISTBOX_SetString()

描述

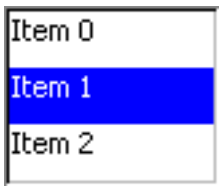
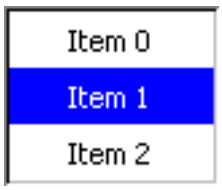
设置指定项目的内容。

原型

```
void LISTBOX_SetString(LISTBOX_Handle hObj,   const char * s,
                       unsigned int   Index);
```

参数	描述
hObj	列表框的句柄。
s	包含新内容的字符串的指针。
Index	要更改的元素索引 （以零为基准）。

LISTBOX_SetTextAlign()

之前	之后
	

描述

该函数设置显示每个列表框项所使用的文本对齐方式。

原型

```
void LISTBOX_SetTextAlign(LISTBOX_Handle hObj, int Align);
```

参数	描述
hObj	小工具的句柄。
Align	要使用的文本对齐方式。

参数 Align 的允许值 （水平和垂直标记可以通过 “OR” 操作进行组合）	
水平对齐	
GUI_TA_LEFT	X 轴方向左对齐 （默认）。
GUI_TA_HCENTER	X 轴方向对中。
GUI_TA_RIGHT	X 轴方向右对齐 （默认）。
垂直对齐	

参数 Align 的允许值 （水平和垂直标记可以通过 “OR” 操作进行组合）	
GUI_TA_TOP	将 Y 轴位置与字符的顶部对齐 （默认）。
GUI_TA_VCENTER	在 Y 轴方向对中。
GUI_TA_BOTTOM	将 Y 轴位置与字体的底部像素行对齐。

其他信息

列表框的默认对齐方式为 GUI_TA_LEFT。按照默认设置，每项的高度取决于呈现列表框项的字体高度。因此，只有在使用 LISTBOX_SetItemSpacing() 函数设置项目下方的额外间距时，垂直文本对齐方式才有效。

LISTBOX_SetTextColor()

描述

设置列表框的文本颜色。

原型

```
void LISTBOX_SetTextColor(LISTBOX_Handle hObj,   unsigned int Index,
                          GUI_COLOR      Color);
```

参数	描述
hObj	列表框的句柄。
Index	文本颜色的索引 （参见 LISTBOX_SetBackColor()）。
Color	要设置的颜色。

LISTBOX_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

15.12.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_SimpleListBox.c
- WIDGET_ListBox.c

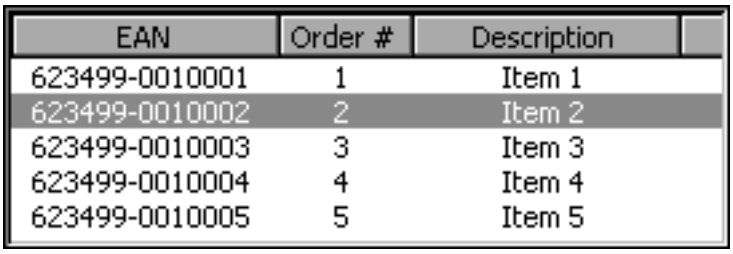
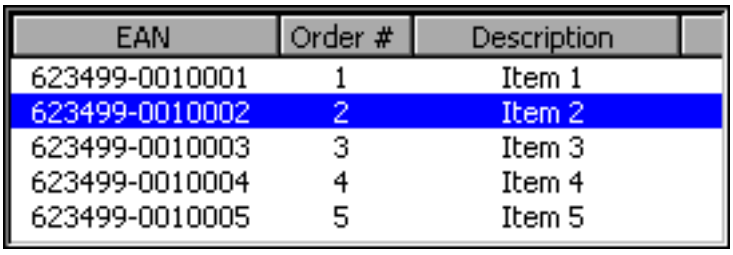
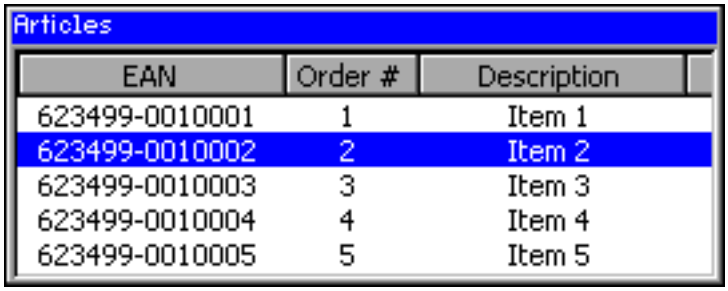
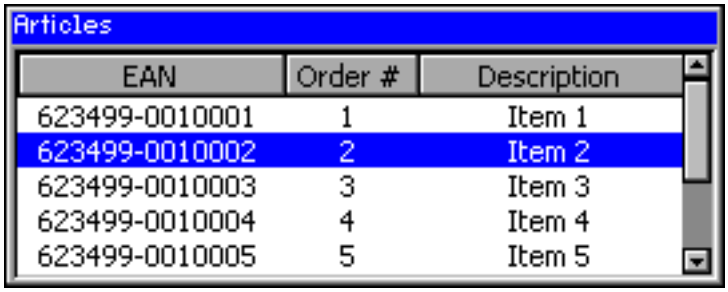
需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

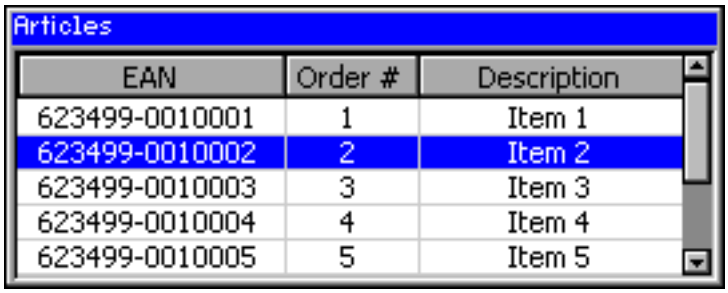
WIDGET_SimpleListBox.c 的屏幕截图：

WIDGET_ListBox.c 的屏幕截图：

15.13 LISTVIEW：Listview 小工具

LISTVIEW 小工具可在具有多个列的列表中选择某个元素。由于LISTVIEW 小工具包含了一个HEADER 小工具，因此可对列加以管理。所创建的 LISTVIEW 既可以无环绕型框架窗口，也可以作为 FRAMEWIN 小工具的一个子窗口。一旦选定列表视图中的项目，就会突出显示有关项目。需要注意的是，所选项目的背景颜色取决于 LISTVIEW 窗口是否具有输入焦点。下表显示的是 LISTVIEW 小工具的外观：

描述	LISTVIEW 小工具
无焦点 无环绕型 FRAMEWIN 未附加滚动条 网格线不可见	
具有输入焦点 无环绕型 FRAMEWIN 未附加滚动条 网格线不可见	
具有输入焦点 带环绕型 FRAMEWIN 未附加滚动条 网格线不可见	
具有输入焦点 带环绕型 FRAMEWIN 附加滚动条 网格线不可见	

描述	LISTVIEW 小工具
具有输入焦点 带环绕型 FRAMEWIN 附加滚动条 网格线可见	

15.13.1 配置选项

类型	宏	默认值	描述
S	LISTVIEW_FONT_DEFAULT	&GUI_Font13_1	默认字体
N	LISTVIEW_BKCOLOR0_DEFAULT	GUI_WHITE	背景颜色，未选定状态。
N	LISTVIEW_BKCOLOR1_DEFAULT	GUI_GRAY	背景颜色，选定状态，无焦点。
N	LISTVIEW_BKCOLOR2_DEFAULT	GUI_BLUE	背景颜色，选定状态，有焦点。
N	LISTVIEW_SCROLLSTEP_H_DEFAULT	10	根据需要定义要滚动的像素数。
N	LISTVIEW_TEXTCOLOR0_DEFAULT	GUI_BLACK	文本颜色，未选定状态。
N	LISTVIEW_TEXTCOLOR1_DEFAULT	GUI_WHITE	文本颜色，选定状态，无焦点。
N	LISTVIEW_TEXTCOLOR2_DEFAULT	GUI_WHITE	文本颜色，选定状态，有焦点。
N	LISTVIEW_GRIDCOLOR_DEFAULT	GUI_LIGHTGRAY	网格线的颜色 （如显示）
N	LISTVIEW_ALIGN_DEFAULT	GUI_TA_VCENTER |GUI_TA_HCENTER	默认的文本对齐方式

15.13.2 通知代码

以下事件将作为 WM_NOTIFY_PARENT 消息的一部分从 LISTVIEW 小工具发送至其父窗口：

消息	描述
WM_NOTIFICATION_CLICKED	已单击小工具。
WM_NOTIFICATION_RELEASED	已释放小工具。
WM_NOTIFICATION_MOVED_OUT	已单击小工具，并且指针已移出小工具，但没有释放。
WM_NOTIFICATION_SCROLL_CHANGED	可选滚动条的滚动位置已更改。
WM_NOTIFICATION_SEL_CHANGED	列表框的选择内容已更改。

15.13.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_UP	选择栏上移。
GUI_KEY_DOWN	选择栏下移。
GUI_KEY_RIGHT	如果总列宽 > 列表视图的内部区域，则内容将滚动到左侧。
GUI_KEY_LEFT	如果总列宽 > 列表视图的内部区域，则内容将滚动到右侧。

15.13.4 LISTVIEW API

下表按字母顺序列出可用的 emWin LISTVIEW 相关例程。这些例程的详细说明如下。

例程	描述
LISTVIEW_AddColumn()	向 LISTVIEW 添加列。
LISTVIEW_AddRow()	向 LISTVIEW 添加行。
LISTVIEW_CompareDec()	比较 2 个整数值所使用的比较函数。
LISTVIEW_CompareText()	比较 2 个字符串所使用的比较函数。
LISTVIEW_Create()	创建 LISTVIEW 小工具。（弃用）
LISTVIEW_CreateAttached()	创建附加到窗口的 LISTVIEW 小工具。
LISTVIEW_CreateEx()	创建 LISTVIEW 小工具。
LISTVIEW_CreateIndirect()	从资源表条目创建 LISTVIEW 小工具。
LISTVIEW_CreateUser()	使用额外字节作为用户数据来创建 LISTVIEW 小工具。
LISTVIEW_DecSel()	减小选定范围。
LISTVIEW_DeleteColumn()	删除指定的列。
LISTVIEW_DeleteRow()	删除指定的行。
LISTVIEW_DisableRow()	将指定行的状态设置为禁用。
LISTVIEW_DisableSort()	禁用 LISTVIEW 排序功能。
LISTVIEW_EnableRow()	将指定行的状态设置为启用。
LISTVIEW_EnableSort()	启用 LISTVIEW 排序功能。
LISTVIEW_GetBkColor()	返回 LISTVIEW 的背景颜色。
LISTVIEW_GetFont()	返回 LISTVIEW 的字体。
LISTVIEW_GetHeader()	返回附加 HEADER 小工具的句柄。
LISTVIEW_GetItemText()	返回指定单元格的文本。
LISTVIEW_GetNumColumns()	返回列数。
LISTVIEW_GetNumRows()	返回行数。
LISTVIEW_GetSel()	返回选定的项目数。
LISTVIEW_GetSelUnsorted()	返回未排序状态下选定的项目数。
LISTVIEW_GetTextColor()	返回 LISTVIEW 的文本颜色。
LISTVIEW_GetUserData()	检索通过 LISTVIEW_SetUserData() 所设置的数据。
LISTVIEW_GetUserDataRow()	返回指定行的用户数据。
LISTVIEW_IncSel()	增加选定范围。
LISTVIEW_InsertRow()	在指定位置新插入一行。
LISTVIEW_SetAutoScrollH()	启用自动使用水平滚动条。
LISTVIEW_SetAutoScrollV()	启用自动使用垂直滚动条。
LISTVIEW_SetBkColor()	设置背景颜色。
LISTVIEW_SetColumnWidth()	设置列宽。
LISTVIEW_SetCompareFunc()	设置指定列的比较函数。
LISTVIEW_SetDefaultBkColor()	设置 HEADER 小工具的默认背景颜色。
LISTVIEW_SetDefaultFont()	设置 HEADER 小工具的默认字体。
LISTVIEW_SetDefaultGridColor()	设置 HEADER 小工具的默认文本颜色。
LISTVIEW_SetDefaultTextColor()	设置 HEADER 小工具网格线的默认颜色。
LISTVIEW_SetFixed()	固定指定的列数。
LISTVIEW_SetFont()	设置 LISTVIEW 的字体。
LISTVIEW_SetGridVis()	设置网格线的可见性标记。
LISTVIEW_SetHeaderHeight()	设置标题的高度。
LISTVIEW_SetItemBitmap()	将某个位图设置为 LISTVIEW 单元格的背景
LISTVIEW_SetItemBkColor()	设置 LISTVIEW 单元格的背景颜色
LISTVIEW_SetItemText()	设置 LISTVIEW 单元格的文本
LISTVIEW_SetItemTextColor()	设置 LISTVIEW 单元格的文本颜色
LISTVIEW_SetLBorder()	设置左边框所使用的像素数。
LISTVIEW_SetRBorder()	设置右边框所使用的像素数。

例程	描述
LISTVIEW_SetRowHeight()	设置 LISTVIEW 的行高。
LISTVIEW_SetSel()	设置当前选定内容。
LISTVIEW_SetSelUnsorted()	设置未排序状态下的当前选定内容。
LISTVIEW_SetSort()	设置列以及排序依据的排序顺序。
LISTVIEW_SetTextAlign()	设置列的文本对齐方式。
LISTVIEW_SetTextColor()	设置文本颜色。
LISTVIEW_SetUserData()	设置 LISTVIEW 小工具的额外数据。
LISTVIEW_SetUserDataRow()	设置指定行的用户数据。

LISTVIEW_AddColumn()

描述

向 LISTVIEW 小工具新增一列。

原型

```
void LISTVIEW_AddColumn(LISTVIEW_Handle   hObj, int Width,
                        const char      * s,    int Align);
```

参数	描述
hObj	小工具的句柄
Width	新增列的列宽
s	在 HEADER 小工具中将要显示的文本
Align	要设置的文本对齐模式。可能为水平对齐标记和垂直对齐标记的组合

参数 Align 的允许值
（水平和垂直标记可以通过 “OR” 操作进行组合）

水平对齐	
GUI_TA_LEFT	X 轴方向左对齐。
GUI_TA_HCENTER	X 轴方向对中。
GUI_TA_RIGHT	X 轴方向右对齐。
垂直对齐	
GUI_TA_TOP	将 Y 轴位置与字符的顶部对齐。
GUI_TA_VCENTER	在 Y 轴方向对中。
GUI_TA_BOTTOM	将 Y 轴位置与字体的底部像素行对齐。

其他信息

Width 参数可以为 0。如果 Width 的参数 =0，则将由指定文本和水平间距的默认值来计算新增列的列宽。
只能将列添加到 “空的” LISTVIEW 小工具中。如果它包含至少一行，则无法新增一列。

LISTVIEW_AddRow()

描述

向 LISTVIEW 小工具新增一行。

原型

```
void LISTVIEW_AddRow(LISTVIEW_Handle hObj, const GUI_ConstString * ppText);
```

参数	描述
hObj	小工具的句柄
ppText	包含 LISTVIEW 单元格文本的阵列指针

其他信息

ppText 阵列应包含每列的一个项目。如果它所包含的项目较少，则其余单元格会留空。

LISTVIEW_CompareDec()

描述

比较 2 个整数值所使用的比较函数。

原型

```
int LISTVIEW_CompareDec(const void * p0, const void * p1);
```

参数	描述
p0	第一个值的无效指针：
p1	第二个值的无效指针：

返回值

如果单元格 0 的值大于单元格 1 的值，则返回值小于 <0。
0 如果单元格 0 的值与单元格 1 的值相同，则返回值为 0。
如果单元格 0 的值小于单元格 1 的值，则返回值大于 >0。

其他信息

如果单元格文本代表的是整数值，则列表视图排序算法将使用该函数。
有关如何使用该函数进行排序的详情，另请参阅“LISTVIEW_SetCompareFunc()”（第528页）。
“样本”文件夹中包含示例 WIDGET_SortedListview.c，它显示了如何使用该函数。

LISTVIEW_CompareText()

描述

比较 2 个字符串所使用的函数。

原型

```
int LISTVIEW_CompareText(const void * p0, const void * p1);
```

参数	描述
p0	第一个文本的无效指针：
p1	第二个文本的无效指针：

返回值

如果单元格 0 的文本大于单元格 1 的文本，则返回值大于 >0。
0 如果单元格 0 的文本与单元格 1 的文本相同，则返回值为 0。
如果单元格 0 的文本小于单元格 1 的文本，则返回值小于 <0。

其他信息

列表视图排序算法将使用该函数。

有关如何使用该函数进行排序的详情，另请参阅“LISTVIEW_SetCompareFunc()”（第528页）。“样本”文件夹中包含示例 WIDGET_SortedListview.c，它显示了如何使用该函数。

LISTVIEW_Create()

（弃用，应使用 LISTVIEW_CreateEx() 来代替）

描述

在指定位置创建规定大小的 LISTVIEW 小工具。

原型

```
LISTVIEW_Handle LISTVIEW_Create(int     x0,      int y0,
                                int     xsize,   int ysize,
                                WM_HWIN hParent, int Id,
                                int     Flags,   int SpecialFlags);
```

参数	描述
x0	HEADER 小工具最左侧的像素（在父坐标中）。
y0	HEADER 小工具最顶端的像素（在父坐标中）。
xsize	HEADER 小工具的水平尺寸（以像素为单位）。
ysize	HEADER 小工具的垂直尺寸（以像素为单位）。
hParent	父窗口的句柄
Id	新创建的 HEADER 小工具的 Id
Flags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW（有关可用参数值的列表，请参阅“窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）
SpecialFlags	（保留供日后使用）

返回值

所创建的 LISTVIEW 小工具的句柄；如果函数执行失败，则返回值为 0。

LISTVIEW_CreateAttached()

描述

创建附加到现有窗口的 LISTVIEW 小工具。

原型

```
LISTVIEW_Handle LISTVIEW_CreateAttached(WM_HWIN hParent,       int Id,
                                        int     SpecialFlags);
```

参数	描述
hObj	小工具的句柄
Id	新创建的 LISTVIEW 小工具的 Id
SpecialFlags	（未使用，保留供日后使用）

返回值

所创建的 LISTVIEW 小工具的句柄；如果函数执行失败，则返回值为 0。

其他信息

附加的 LISTVIEW 小工具实质上是一个子窗口，它将自己放置在父窗口上并进行相应的操作。

LISTVIEW_CreateEx()

描述

在指定位置创建规定大小的 LISTVIEW 小工具。

原型

```
LISTVIEW_Handle LISTVIEW_CreateEx(int     x0,      int y0,
                                  int     xsize,   int ysize,
                                  WM_HWIN hParent, int WinFlags,
                                  int     ExFlags, int Id);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新创建的 LISTVIEW 小工具将作为桌面 （顶层窗口）的子窗口。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）
ExFlags	未使用，保留供日后使用。
Id	小工具的窗口 ID。

返回值

所创建的 LISTVIEW 小工具的句柄；如果函数执行失败，则返回值为 0。

LISTVIEW_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。

LISTVIEW_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细说明，可参阅函数 LISTVIEW_CreateEx()。

LISTVIEW_DecSel()

描述

减小列表视图选定范围 （如果可能的话，将指定列表视图的选择栏上移一个项目）。

原型

```
void LISTVIEW_DecSel(LISTVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄

其他信息

需要注意的是，项目编号始终从顶部以0值开始；因此，减小选定范围实际上会将选定范围上移一行。

LISTVIEW_DeleteColumn()

描述

删除列表视图的指定列。

原型

```
void LISTVIEW_DeleteColumn(LISTVIEW_Handle hObj, unsigned Index);
```

参数	描述
hObj	小工具的句柄
Index	要删除的列的索引（以零为基准）。

其他信息

需要注意的是，项目编号始终从左以 0 值开始。

LISTVIEW_DeleteRow()

描述

删除列表视图的指定行。

原型

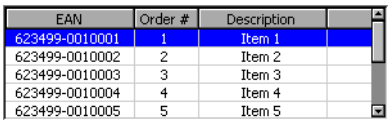
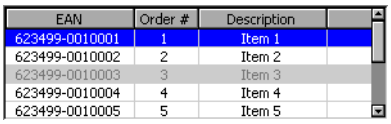
```
void LISTVIEW_DeleteRow(LISTVIEW_Handle hObj, unsigned Index);
```

参数	描述
hObj	小工具的句柄
Index	要删除的行的索引（以零为基准）。

其他信息

需要注意的是，项目编号始终从顶部以 0 值开始。

LISTVIEW_DisableRow()

之前	之后
	

描述

该函数可将指定行的状态设置为禁用。

原型

```
void LISTVIEW_DisableRow(LISTVIEW_Handle hObj, unsigned Row);
```

参数	描述
hObj	小工具的句柄
Row	要禁用的行的索引（以零为基准）。

其他信息

在滚动列表视图时，将跳过禁用的项目。您无法滚动至已禁用的列表视图项。

LISTVIEW_DisableSort()

描述

禁用指定列表视图的排序功能。调用该函数后，列表视图的内容将以未排序状态显示。

原型

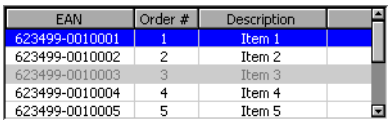
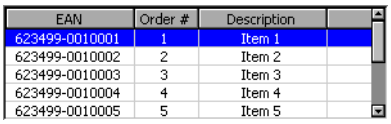
```
void LISTVIEW_DisableSort(LISTVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄

其他信息

有关如何在列表视图小工具中使用排序功能的详情，请参阅 “LISTVIEW_SetCompareFunc()”（第 528 页）和 “LISTVIEW_SetSort()”（第 536 页）。
“样本”文件夹中包含示例 WIDGET_SortedListview.c，它显示了如何使用该函数。

LISTVIEW_EnableRow()

之前	之后
	

描述

该函数可将指定行的状态设置为启用。

原型

```
void LISTVIEW_EnableRow(LISTVIEW_Handle hObj, unsigned Row);
```

参数	描述
hObj	小工具的句柄
Row	要禁用的行的索引 （以零为基准）。

其他信息

请参阅 “LISTVIEW_DisableRow()”（第 521 页）。

LISTVIEW_EnableSort()

描述

启用指定列表视图的排序功能。调用该函数后，单击所需列的标题项，列表视图的内容即可以排序状态呈现，且列表视图将以该顺序排序其数据。需要注意的是，只有设置所需列的比较函数后，该操作才有效。

原型

```
void LISTVIEW_EnableSort(LISTVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄

其他信息

有关如何设置比较函数的详情，请参阅 “LISTVIEW_SetCompareFunc()”（第 528 页）。
“样本”文件夹中包含示例 WIDGET_SortedListview.c，它显示了如何使用该函数。

LISTVIEW_GetBkColor()

描述

返回指定列表视图的背景颜色。

原型

```
GUI_COLOR LISTVIEW_GetBkColor(LISTVIEW_Handle hObj, unsigned Index);
```

参数	描述
hObj	小工具的句柄
Index	颜色的索引 （参见下表）

参数 Index 的允许值	
LISTVIEW_CI_UNSEL	未选定元素。
LISTVIEW_CI_SEL	选定元素，无焦点。
LISTVIEW_CI_SELFOCUS	选定元素，有焦点。

返回值

指定列表视图的背景颜色。

LISTVIEW_GetFont()

描述

返回显示列表视图文本所使用的字体指针。

原型

```
const GUI_FONT * LISTVIEW_GetFont(LISTVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

显示列表视图文本所使用的字体。

LISTVIEW_GetHeader()

描述

返回 HEADER 小工具的句柄。

原型

```
HEADER_Handle LISTVIEW_GetHeader(LISTVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄

返回值

HEADER 小工具的句柄。

其他信息

每个 LISTVIEW 小工具都包含一个 HEADER 小工具，可对列加以管理。使用该句柄可以更改 LISTVIEW-HEADER 的属性，例如，更改 HEADER 小工具的文本颜色。

示例：

```
LISTVIEW_Handle hListView = LISTVIEW_Create(10, 80, 270, 89, 0, 1234, WM_CF_SHOW, 0);
HEADER_Handle hHeader = LISTVIEW_GetHeader(hListView);
HEADER_SetTextColor(hHeader, GUI_GREEN);
```

LISTVIEW_GetItemText()

描述

将指定列表视图单元格的文本复制到指定的缓冲区，即可将其返回。

原型

```
void LISTVIEW_GetItemText(LISTVIEW_Handle hObj,     unsigned   Column,
                          unsigned        Row,      char     * pBuffer,
                          unsigned        MaxSize);
```

参数	描述
hObj	小工具的句柄
Column	单元格列的索引 （以零为基准）
Row	单元格行的索引 （以零为基准）
pBuffer	由例程填充的缓冲区指针。
MaxSize	缓冲区大小 （以字节为单位）。

其他信息

如果单元格的文本不适合缓冲区，则参数 MaxSize 指定的字节数将复制到缓冲区。

LISTVIEW_GetNumColumns()

描述

返回指定的 LISTVIEW 小工具的列数。

原型

```
unsigned LISTVIEW_GetNumColumns(LISTVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄

返回值

指定的 LISTVIEW 小工具的列数。

LISTVIEW_GetNumRows()

描述

返回指定的 LISTVIEW 小工具的行数。

原型

```
unsigned LISTVIEW_GetNumRows(LISTVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄

返回值

指定的 LISTVIEW 小工具的行数。

LISTVIEW_GetSel()

描述

返回当前在指定的 LISTVIEW 小工具中所选的行数。

原型

```
int LISTVIEW_GetSel(LISTVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄

返回值

当前所选的行数。

LISTVIEW_GetSelUnsorted()

描述

返回未排序状态下当前所选行的索引。

原型

```
int LISTVIEW_GetSelUnsorted(LISTVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄

返回值

未排序状态下当前所选行的索引。

其他信息

该函数会返回所选行的实际索引，而函数 LISTVIEW_GetSel() 只会返回已排序行的索引。实际（未排序）行索引应在函数调用中用作行索引。
“样本”文件夹中包含示例 WIDGET_SortedListview.c，它显示了如何使用该函数。

LISTVIEW_GetTextColor()

描述

返回指定的列表视图的文本颜色。

原型

```
GUI_COLOR LISTVIEW_GetTextColor(LISTVIEW_Handle hObj, unsigned Index);
```

参数	描述
hObj	小工具的句柄
Index	颜色的索引 （参见下表）

参数 Index 的允许值	
LISTVIEW_CI_UNSEL	未选定元素。
LISTVIEW_CI_SEL	选定元素，无焦点。
LISTVIEW_CI_SELFOCUS	选定元素，有焦点。

返回值

指定的列表视图的文本颜色。

LISTVIEW_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

LISTVIEW_GetUserDataRow()

描述

返回指定行的用户数据。

原型

```
U32 LISTVIEW_GetUserData(LISTVIEW_Handle hObj, unsigned Row);
```

参数	描述
hObj	小工具的句柄
Row	以零为基准的行的索引

返回值

指定行的用户数据。

其他信息

有关如何设置行的用户数据的详情，请参阅 “LISTVIEW_SetUserDataRow()”（第 537 页）。

LISTVIEW_IncSel()

描述

扩大列表框选定范围 （将指定 LISTVIEW 的选择栏下移一个项目）。

原型

```
void LISTVIEW_IncSel(LISTVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄

LISTVIEW_InsertRow()

描述

在列表视图的指定位置新插入一行。

原型

```
int LISTVIEW_InsertRow(LISTVIEW_Handle          hObj,    unsigned Index,
                       const GUI_ConstString * ppText);
```

参数	描述
hObj	小工具的句柄
Index	新增列的索引
ppText	包含新增行单元格文本的字符串阵列指针

返回值

如果函数成功执行，则返回值为 0 ；如果发生错误，则返回值为 1。

其他信息

ppText 阵列应包含每列的一个项目。如果它所包含的项目较少，则其余单元格会留空。
如果指定的索引 >= 当前行数，则可使用函数 LISTVIEW_AddRow() 来新增一行。
“样本”文件夹中包含示例 WIDGET_SortedListview.c，它显示了如何使用该函数。

LISTVIEW_SetAutoScrollH()

描述

启用 / 禁用自动使用水平滚动条。

原型

```
void LISTVIEW_SetAutoScrollH(LISTVIEW_Handle hObj, int OnOff);
```

参数	描述
hObj	小工具的句柄
OnOff	（参见下表）

参数 OnOff 的允许值	
0	禁用自动使用水平滚动条。
1	启用自动使用水平滚动条。

其他信息

如果启用，则列表视图会检查所有列是否适合小工具区域。如果未启用，则将添加水平滚动条。

LISTVIEW_SetAutoScrollV()

描述

启用 / 禁用自动使用垂直滚动条。

原型

```
void LISTVIEW_SetAutoScrollV(LISTVIEW_Handle hObj, int OnOff);
```

参数	描述
hObj	小工具的句柄
OnOff	（参见下表）

参数 OnOff 的允许值	
0	禁用自动使用垂直滚动条。
1	启用自动使用垂直滚动条。

其他信息

如果启用，则列表视图会检查所有行是否适合小工具区域。如果未启用，则将添加垂直滚动条。

LISTVIEW_SetBkColor()

描述

设置指定的 LISTVIEW 小工具的背景颜色。

原型

```
void LISTVIEW_SetBkColor(LISTVIEW_Handle hObj,   unsigned int Index,
                         GUI_COLOR       Color);
```

参数	描述
hObj	小工具的句柄
Index	背景颜色的索引（参见下表）
Color	要设置的颜色

参数 Index 的允许值	
LISTVIEW_CI_UNSEL	未选定元素。
LISTVIEW_CI_SEL	选定元素，无焦点。
LISTVIEW_CI_SELFOCUS	选定元素，有焦点。
LISTVIEW_CI_DISABLED	禁用的元素。

其他信息

如果要设置单个单元格的背景颜色，则应使用函数 LISTVIEW_SetItemBkColor()。
“样本”文件夹中包含示例 WIDGET_SortedListview.c，它显示了如何使用该函数。

LISTVIEW_SetColumnWidth()

描述

设置指定列的宽度。

原型

```
void LISTVIEW_SetColumnWidth(LISTVIEW_Handle hObj,   unsigned int Index,
                             int             Width);
```

参数	描述
hObj	小工具的句柄
Index	列数
Width	新的宽度

LISTVIEW_SetCompareFunc()

描述

设置指定列的比较函数。如果列表视图小工具按指定列排序，则需要设置比较函数。

原型

```
void LISTVIEW_SetCompareFunc(LISTVIEW_Handle hObj, unsigned Column,
                      int (* fpCompare)(const void * p0, const void * p1));
```

参数	描述
hObj	小工具的句柄
Column	目标列 （应当为该列设置比较函数）的索引。
fpCompare	比较函数的函数指针。

其他信息

如果使用列表视图小工具的排序功能，则小工具将使用比较函数，以确定一个单元格的内容是否大于、等于或小于另一个单元格的内容。
在默认情况下，不会为列表视图列设置比较函数。对于排序所使用的每个列，需要设置比较函数。
列表视图小工具的单元格包含文本。但文本有时为其他类型的数据，如日期、整数等。因此，排序需要不同的比较函数。 emWin 提供 2 个比较函数：

LISTVIEW_CompareText()：该函数可用来比较包含文本的单元格。

LISTVIEW_CompareDec()：该函数可用来比较包含文本内容为整数值的单元格。

如果第二个单元格的内容大于第一个单元格的内容，则比较函数的返回值将 >0；如果第二个单元格的内容小于第一个单元格的内容，则返回值将 <0；或如果第二个单元格的内容等于第一个单元格的内容，则返回值为 0。
另外，还可以使用用户定义的比较函数。应用程序定义的函数原型，其定义应当如下：

原型

```
int APPLICATION_Compare(const void * p0, const void * p1);
```

参数	描述
p0	第一个单元格 NULL 终止字符串数据的指针。
p1	第二个单元格 NULL 终止字符串数据的指针。

示例

```
int APPLICATION_Compare(const void * p0, const void * p1) {
  return strcmp((const char *)p1, (const char *)p0);
}

void SetAppCompareFunc(WM_HWIN hListView, int Column) {
  LISTVIEW_SetCompareFunc(hListView, Column, APPLICATION_Compare);
}
```

“样本”文件夹中包含示例 WIDGET_SortedListview.c，它显示了如何使用该函数。

LISTVIEW_SetDefaultBkColor()

描述

设置新创建的 LISTVIEW 小工具的默认背景颜色。

原型

```
GUI_COLOR LISTVIEW_SetDefaultBkColor(unsigned int Index, GUI_COLOR Color);
```

参数	描述
Index	默认背景颜色的索引 （参见下表）
Color	要设置为默认值的颜色

参数 Index 的允许值	
LISTVIEW_CI_UNSEL	未选定元素。
LISTVIEW_CI_SEL	选定元素，无焦点。
LISTVIEW_CI_SELFOCUS	选定元素，有焦点。
LISTVIEW_CI_DISABLED	禁用的元素。

返回值

先前的默认值。

LISTVIEW_SetDefaultFont()

描述

设置新创建的 LISTVIEW 小工具的默认字体。

原型

```
const GUI_FONT * LISTVIEW_SetDefaultFont(const GUI_FONT * pFont);
```

参数	描述
pFont	新创建的 LISTVIEW 小工具所使用的字体指针

返回值
先前的默认值。

LISTVIEW_SetDefaultGridColor()

描述
设置新创建的 LISTVIEW 小工具网格线的默认颜色。

原型
```
GUI_COLOR LISTVIEW_SetDefaultGridColor(GUI_COLOR Color);
```

参数	描述
Color	新默认值

返回值
先前的默认值

LISTVIEW_SetDefaultTextColor()

描述
设置新创建的 LISTVIEW 小工具的默认文本颜色。

原型
```
GUI_COLOR LISTVIEW_SetDefaultTextColor(unsigned int Index, GUI_COLOR Color);
```

参数	描述
Index	默认文本颜色的索引 （参见下表）
Color	要设置为默认值的颜色

参数 Index 的允许值	
0	未选定元素。
1	选定元素，无焦点。
2	选定元素，有焦点。

返回值
先前的默认值。

LISTVIEW_SetFixed()

描述
将指定的列数固定在其水平位置。

原型
```
unsigned LISTVIEW_SetFixed(LISTVIEW_Handle hObj, unsigned Fixed);
```

参数	描述
hObj	列表视图的句柄。
Fixed	固定在其水平位置的指定列数。

其他信息
只有在滚动操作期间一个或多个列保持在其水平位置时，使用该函数才有意义。

LISTVIEW_SetFont()

描述

设置列表视图的字体。

原型

```
void LISTVIEW_SetFont(LISTVIEW_Handle hObj, const GUI_FONT * pFont);
```

参数	描述
hObj	列表视图的句柄。
pFont	字体的指针。

LISTVIEW_SetGridVis()

描述

设置网格线的可见性标记。创建 LISTVIEW 时，网格线默认为禁用。

原型

```
int LISTVIEW_SetGridVis(LISTVIEW_Handle hObj, int Show);
```

参数	描述
hObj	小工具的句柄
Show	设置网格线的可见性

参数 Show 的允许值	
0	不可见
1	可见

返回值

可见性标记的上一个值。

LISTVIEW_SetHeaderHeight()

描述

设置附加标题小工具的高度。

原型

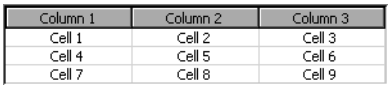
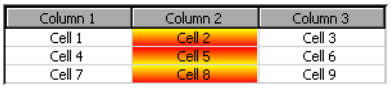
```
void LISTVIEW_SetHeaderHeight(LISTVIEW_Handle hObj, unsigned HeaderHeight);
```

参数	描述
hObj	LISTVIEW 小工具的句柄。
Show	要设置的附加标题小工具的高度。

其他信息

如果将高度设置为 0，则将不显示标题小工具。

LISTVIEW_SetItemBitmap()

之前	之后
	

描述

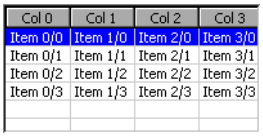
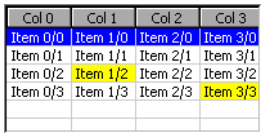
将位图设置为指定单元格的背景。

原型

```
void LISTVIEW_SetItemBitmap(LISTVIEW_Handle hObj,
                            unsigned        Column, unsigned Row,
                            int             xOff,   int      yOff,
                            const GUI_BITMAP GUI_UNI_PTR * pBitmap);
```

参数	描述
hObj	Listview 小工具的句柄
Column	列数
Row	行数
xOff	要绘制的位图最左边像素的偏移
yOff	要绘制的位图最顶端像素的偏移
pBitmap	位图的指针

LISTVIEW_SetItemBkColor()

之前	之后
	

描述

设置指定单元格的背景颜色。

原型

```
void LISTVIEW_SetItemBkColor(LISTVIEW_Handle hObj,
                             unsigned        Column, unsigned  Row,
                             unsigned int    Index,  GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄
Column	列数
Row	行数
Index	背景颜色的索引 （见下表）
Color	要使用的颜色

参数 Index 的允许值	
LISTVIEW_CI_UNSEL	未选定元素。

参数 Index 的允许值	
LISTVIEW_CI_SEL	选定元素，无焦点。
LISTVIEW_CI_SELFOCUS	选定元素，有焦点。
LISTVIEW_CI_DISABLED	禁用的元素。

其他信息

此函数会覆盖由 LISTVIEW_SetBkColor() 为指定单元格所设置的默认背景颜色。

LISTVIEW_SetItemText()

描述

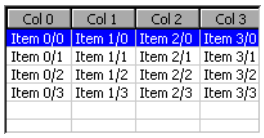
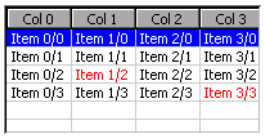
设置由行和列所指定的 LISTVIEW 小工具的单元格文本。

原型

```
void LISTVIEW_SetItemText(LISTVIEW_Handle hObj, unsigned     Column,
                          unsigned        Row,  const char * s);
```

参数	描述
hObj	小工具的句柄。
Column	列数。
Row	行数。
s	要在表格单元格中显示的文本。

LISTVIEW_SetItemTextColor()

之前	之后
	

描述

设置指定单元格的文本颜色。

原型

```
void LISTVIEW_SetItemTextColor(LISTVIEW_Handle hObj,
                               unsigned        Column, unsigned  Row,
                               unsigned int    Index,  GUI_COLOR Color);
```

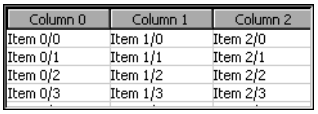
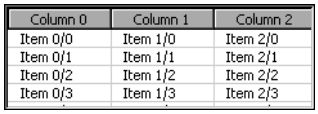
参数	描述
hObj	小工具的句柄
Column	列数
Row	行数
Index	文本颜色的索引 （参见下表）
Color	要使用的颜色

参数 Index 的允许值	
LISTVIEW_CI_UNSEL	未选定元素。
LISTVIEW_CI_SEL	选定元素，无焦点。
LISTVIEW_CI_SELFOCUS	选定元素，有焦点。

其他信息

此函数会覆盖由 LISTVIEW_SetTextColor() 为指定单元格所设置的默认文本颜色。

LISTVIEW_SetLBorder()

之前	之后
	

描述

设置列表视图每个单元格内左边框所使用的像素数。

原型

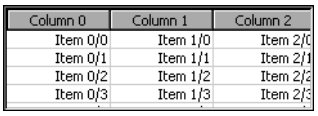
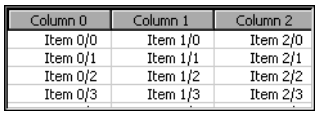
```
void LISTVIEW_SetLBorder(LISTVIEW_Handle hObj, unsigned BorderSize);
```

参数	描述
hObj	小工具的句柄。
BorderSize	要使用的像素数。

其他信息

使用该函数对列表视图所使用的标题小工具没有任何影响。

LISTVIEW_SetRBorder()

之前	之后
	

描述

设置列表视图每个单元格内右边框所使用的像素数。

原型

```
void LISTVIEW_SetRBorder(LISTVIEW_Handle hObj, unsigned BorderSize);
```

参数	描述
hObj	小工具的句柄。
BorderSize	要使用的像素数。

其他信息

使用该函数对列表视图所使用的标题小工具没有任何影响。

LISTVIEW_SetRowHeight()

描述

设置列表视图每一行高度所使用的像素数。

原型

```
unsigned LISTVIEW_SetRowHeight(LISTVIEW_Handle hObj, unsigned RowHeight);
```

参数	描述
hObj	小工具的句柄。

返回值

由该函数所设置的行高的上一个值。如果返回值为 0，则行的高度取决于小工具所使用的字体的高度。

其他信息

默认情况下，行的高度取决于所使用字体的高度。

LISTVIEW_SetSel()

描述

设置指定的 LISTVIEW 小工具已选定的行。

原型

```
void LISTVIEW_SetSel(LISTVIEW_Handle hObj, int Sel);
```

参数	描述
hObj	小工具的句柄
Sel	要选择的元素

LISTVIEW_SetSelUnsorted()

描述

设置未排序状态下当前所选行的索引。

原型

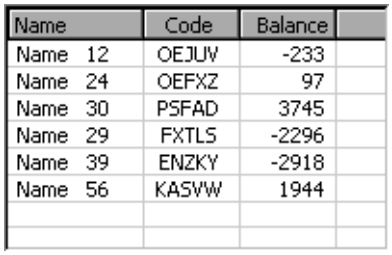
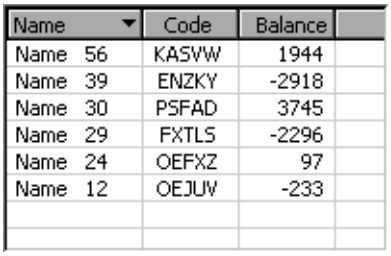
```
void LISTVIEW_SetSelUnsorted(LISTVIEW_Handle hObj, int Sel);
```

参数	描述
hObj	小工具的句柄。
Sel	未排序状态下以零为基准的选择索引。

其他信息

该函数设置所选行的实际索引，而函数 LISTVIEW_SetSel() 则设置已排序行的索引。实际（未排序）行索引应在函数调用中用作行索引。
“样本”文件夹中包含示例 WIDGET_SortedListview.c，它显示了如何使用该函数。

LISTVIEW_SetSort()

之前	之后
	

描述

该函数设置作为排序依据的列以及排序顺序。

原型

```
unsigned LISTVIEW_SetSort(LISTVIEW_Handle hObj,     unsigned Column,
                          unsigned        Reverse);
```

参数	描述
hObj	小工具的句柄。
Column	作为排序依据的列。
Reverse	0 表示正常排序顺序 （最大的元素位于顶部）， 1 表示反向顺序。

返回值

如果函数成功，则返回值为 0 ；如果不成功，则返回值为 1。

其他信息

调用该函数前，需要为目标列设置比较函数。有关如何设置比较函数的详情，请参阅 “LISTVIEW_SetCompareFunc()”（第 528 页）。
“样本”文件夹中包含示例 WIDGET_SortedListview.c，它显示了如何使用该函数。

LISTVIEW_SetTextAlign()

描述

设置指定列的对齐方式。

原型

```
void LISTVIEW_SetTextAlign(LISTVIEW_Handle hObj,   unsigned int Index,
                           int             Align);
```

参数	描述
hObj	小工具的句柄
Index	列数
Align	要设置的文本对齐模式。可能为水平对齐标记和垂直对齐标记的组合

参数 Align 的允许值 （水平和垂直标记可以通过 “OR”操作进行组合）	
水平对齐	
GUI_TA_LEFT	X 轴方向左对齐 （默认）。
GUI_TA_HCENTER	X 轴方向对中。
GUI_TA_RIGHT	X 轴方向右对齐 （默认）。
垂直对齐	

参数 Align 的允许值 （水平和垂直标记可以通过 “OR”操作进行组合）	
GUI_TA_TOP	将 Y 轴位置与字符的顶部对齐 （默认）。
GUI_TA_VCENTER	在 Y 轴方向对中。
GUI_TA_BOTTOM	将 Y 轴位置与字体的底部像素行对齐。

LISTVIEW_SetTextColor()

描述

设置指定的 LISTVIEW 小工具的文本颜色。

原型

```
void LISTVIEW_SetTextColor(LISTVIEW_Handle hObj,   unsigned int Index,
                           GUI_COLOR       Color);
```

参数	描述
hObj	小工具的句柄
Index	文本颜色的索引 （参见下表）
Color	要设置的颜色

参数 Index 的允许值	
LISTVIEW_CI_UNSEL	未选定元素。
LISTVIEW_CI_SEL	选定元素，无焦点。
LISTVIEW_CI_SELFOCUS	选定元素，有焦点。

LISTVIEW_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

LISTVIEW_SetUserDataRow()

描述

设置指定行的用户数据。

原型

```
void LISTVIEW_SetUserData(LISTVIEW_Handle hObj,      unsigned Row,
                          U32             UserData);
```

参数	描述
hObj	小工具的句柄
Row	要为其设置用户数据的行
UserData	要与行关联的值

其他信息

设置与行关联的 32 位值。每一行都具有相应的 32 位值，以供应用程序使用。

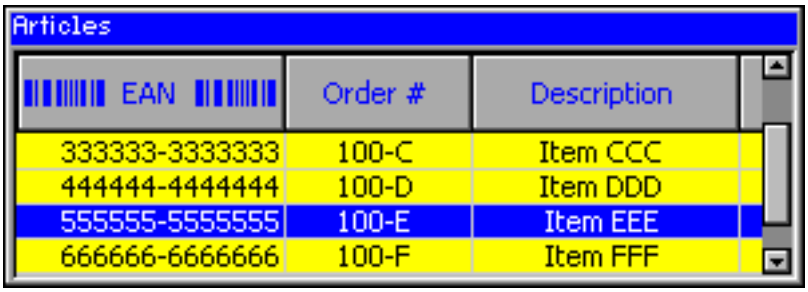
15.13.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_ListView.c

需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_ListView.c 屏幕截图：



15.14 LISTWHEEL：Listwheel 小工具

该小工具与本章前文所述的 LISTBOX 小工具相似。但是， LISTBOX 的数据是使用键盘通过移动光标的方式来进行选择的，或使用滚动条进行选择，而 LISTWHEEL 的工作方式则与之完全不同：整个数据区的移动可通过指针输入设备 (PID) 来实现。从上到下触发小工具（反之亦然），可将数据向上或向下移动。在数据区移动过程中，如果释放 PID，则会放慢数据区的运动速度，并且它最终会在吸附位置吸入一个新的数据项并停下来。此外，数据以循环方式显示。第一个数据项会在最后一项之后继续显示，这就如同链条一样。因此，数据可如同在转轮上一样进行 “旋转”：

描述	LISTWHEEL 小工具
显示用于选择日期的三个转轮的应用示例。该示例使用自绘机构来给小工具叠加自定义 alpha 蒙板，以创建遮光效果。	

上表显示了示例 WIDGET_ListWheel.c 的屏幕截图，该示例位于 emWin 的 “示例” 文件夹 Sample\Tutorial\ 中。

15.14.1 配置选项

类型	宏	默认值	描述
S	LISTWHEEL_FONT_DEFAULT	GUI_Font13_1	所使用的字体。
N	LISTWHEEL_BKCOLOR0_DEFAULT	GUI_WHITE	普通文本的背景颜色。
N	LISTWHEEL_BKCOLOR1_DEFAULT	GUI_WHITE	所选文本的背景颜色。
N	LISTWHEEL_TEXTCOLOR0_DEFAULT	GUI_BLACK	普通文本的文本颜色。
N	LISTWHEEL_TEXTCOLOR1_DEFAULT	GUI_BLUE	所选文本的文本颜色。
N	LISTWHEEL_TEXTALIGN_DEFAULT	GUI_TA_LEFT	默认的文本对齐方式

15.14.2 通知代码

以下事件将作为 WM_NOTIFY_PARENT 消息的一部分从小工具发送至其父窗口：

消息	描述
WM_NOTIFICATION_CLICKED	已单击小工具。
WM_NOTIFICATION_RELEASED	已释放小工具。
WM_NOTIFICATION_MOVED_OUT	已单击小工具，并且指针已移出小工具，但没有释放。
WM_NOTIFICATION_SEL_CHANGED	已在吸附位置吸附一个项。

15.14.3 键盘反应

该小工具目前不会对键盘输入作出反应。

15.14.4 LISTWHEEL API

（可能会更改）
下表按字母顺序列出可用的 emWin LISTWHEEL 相关例程。例程的详细说明如下。

例程	描述
LISTWHEEL_AddString()	添加新字符串。
LISTWHEEL_CreateEx()	创建 LISTWHEEL 小工具。
LISTWHEEL_CreateIndirect()	从资源表条目创建 LISTWHEEL 小工具。
LISTWHEEL_CreateUser()	使用作为用户数据的额外字节来创建 LISTWHEEL 小工具。
LISTWHEEL_GetFont()	返回绘制数据所使用的字体。
LISTWHEEL_GetItemText()	返回所请求的数据项的文本。
LISTWHEEL_GetLBorder()	返回左边框的像素大小。
LISTWHEEL_GetLineHeight()	返回某个数据项的高度。
LISTWHEEL_GetNumItems()	返回数据项的个数。
LISTWHEEL_GetPos()	返回当前所选数据项的索引。
LISTWHEEL_GetRBorder()	返回右边框的像素大小。
LISTWHEEL_GetSel()	返回当前所选数据项。
LISTWHEEL_GetTextAlign()	返回绘制数据项所使用的文本对齐方式。
LISTWHEEL_GetUserData()	检索通过 LISTWHEEL_SetUserData() 所设置的数据。
LISTWHEEL_MoveToPos()	将 LISTWHEEL 移到指定位置。
LISTWHEEL_OwnerDraw()	绘制小工具所使用的默认函数。
LISTWHEEL_SetBkColor()	设置背景所使用的颜色。
LISTWHEEL_SetFont()	设置绘制项目文本所使用的字体。
LISTWHEEL_SetItemData()	将自定义的无确切类型指针分配给指定数据项。
LISTWHEEL_SetLBorder()	设置左边框的像素大小。
LISTWHEEL_SetLineHeight()	设置绘制某个数据项所使用的高度。
LISTWHEEL_SetOwnerDraw()	设置绘制小工具所使用的自绘函数。
LISTWHEEL_SetPos()	将 LISTWHEEL 固定到指定位置。
LISTWHEEL_SetRBorder()	设置右边框的像素大小。
LISTWHEEL_SetSel()	设置当前所选数据项。
LISTWHEEL_SetSnapPosition()	从小工具顶部设置的吸附位置 （以像素为单位）。
LISTWHEEL_SetText()	设置小工具的内容。
LISTWHEEL_SetTextAlign()	设置绘制数据项所使用的对齐方式。
LISTWHEEL_SetTextColor()	设置绘制数据项所使用的颜色。
LISTWHEEL_SetUserData()	设置 LISTWHEEL 小工具的额外数据。
LISTWHEEL_SetVelocity()	开始以指定速度移动转轮。

LISTWHEEL_AddString()

描述

将新数据项 （通常为字符串）添加到小工具。

原型

```
void LISTWHEEL_AddString(LISTWHEEL_Handle hObj, const char * s);
```

参数	描述
hObj	小工具的句柄。
s	要添加的字符串指针。

其他信息

指定文本的宽度应适合小工具水平区域。否则，文本在绘制操作过程中将被裁剪。

LISTWHEEL_CreateEx()

描述

在指定位置创建指定大小的 LISTWHEEL 小工具。

原型

```
LISTWHEEL_Handle LISTWHEEL_CreateEx(int     x0,       int y0,
                                    int     xSize,    int ySize,
                                    WM_HWIN hParent,  int WinFlags,
                                    int     ExFlags,  int Id,
                                    const GUI_ConstString * ppText);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新创建的 LISTVIEW 小工具将作为桌面 （顶层窗口）的子窗口。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “WM_CreateWindow()”（第 309 页））。
ExFlags	未使用，保留供日后使用。
Id	小工具的窗口 ID。
ppText	包含要显示的元素的字符串阵列指针。

返回值

所创建的 LISTWHEEL 小工具的句柄；如果函数执行失败，则返回值为 0。

其他信息

如果使用参数 ppText，则阵列的最后一个元素必须为 NULL 元素。

示例

```
char * apText[] = {
  "Monday",
  "Tuesday",
  "Wednesday",
  "Thursday",
  "Friday",
  "Saturday",
  "Sunday",
  NULL
};

LISTWHEEL_CreateEx(10, 10, 100, 100, WM_HBKWIN, WM_CF_SHOW,
                   0, GUI_ID_LISTWHEEL0, apText);
```

LISTWHEEL_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。

LISTWHEEL_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细说明，可参阅函数 LISTWHEEL_CreateEx()。

LISTWHEEL_GetFont()

描述

返回绘制指定小工具数据项所使用的字体。

原型

```
const GUI_FONT GUI_UNI_PTR * LISTWHEEL_GetFont(LISTWHEEL_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

绘制数据项所使用的 GUI_FONT 结构的指针。

LISTWHEEL_GetItemText()

描述

返回所请求的数据项的文本。

原型

```
void LISTWHEEL_GetItemText(LISTWHEEL_Handle   hObj,    unsigned Index,
                           char             * pBuffer, int      MaxSize);
```

参数	描述
hObj	小工具的句柄。
Index	所请求的数据项的索引。
pBuffer	存储文本所使用的缓冲区。
MaxSize	缓冲区大小 （以字节为单位）。

其他信息

该函数可将指定数据项的文本复制到指定的缓冲区中。如果缓冲区太小，将会裁剪文本。

LISTWHEEL_GetLBorder()

描述

返回小工具左边框与文本开头之间的像素大小。

原型

```
int LISTWHEEL_GetLBorder(LISTWHEEL_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

左边框与文本之间的像素数。

LISTWHEEL_GetLineHeight()

描述

返回某个数据项的高度。

原型

```
unsigned LISTWHEEL_GetLineHeight(LISTWHEEL_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值
某个数据项的高度。

其他信息
该函数会返回函数 LISTWHEEL_SetLineHeight() 所设置的值。如果返回值为零，则意味着数据项的高度取决于当前字体的大小。更多详细信息，请参阅 “LISTWHEEL_SetLineHeight()”（第547页），“LISTWHEEL_GetFont()”（第541页），和“GUI_GetYSizeOfFont()”（第187页）。

LISTWHEEL_GetNumItems()

描述
返回指定小工具的数据项的个数。

原型
```
int LISTWHEEL_GetNumItems(LISTWHEEL_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值
指定小工具的数据项的个数。

LISTWHEEL_GetPos()

描述
返回当前吸入项的索引（以零为基准）。

原型
```
int LISTWHEEL_GetPos(LISTWHEEL_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值
当前吸入项的索引。

其他信息
吸附数据项的位置可通过函数 LISTWHEEL_SetSnapPosition() 进行设置。更多详细信息，请参阅 “LISTWHEEL_SetSnapPosition()”（第 550 页）。

LISTWHEEL_GetRBorder()

描述
返回小工具右边框与文本结尾之间的像素大小。

原型
```
int LISTWHEEL_GetRBorder(LISTWHEEL_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值
右边框与文本之间的像素数。

LISTWHEEL_GetSel()

描述

返回当前所选项的索引 （以零为基准）。

原型

```
int LISTWHEEL_GetSel(LISTWHEEL_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

当前所选项的索引。

其他信息

更多详细信息，请参阅 “LISTWHEEL_SetSel()”（第 549 页）。

LISTWHEEL_GetSnapPosition()

描述

返回离开小工具顶部的位置 （以像素为单位），在该位置应当 “吸入”数据项。

原型

```
int LISTWHEEL_GetSnapPosition(LISTWHEEL_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

离开小工具上边缘的吸附位置 （以像素为单位）。

其他信息

默认值为 0。

LISTWHEEL_GetTextAlign()

描述

返回指定小工具的文本对齐方式。

原型

```
int LISTWHEEL_GetTextAlign(LISTWHEEL_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

指定小工具的文本对齐方式。

其他信息

更多详细信息，请参阅 “LISTWHEEL_SetTextAlign()”（第 551 页）。

LISTWHEEL_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

LISTWHEEL_MoveToPos()

描述

将小工具的数据区移到指定位置。

原型

```
void LISTWHEEL_MoveToPos(LISTWHEEL_Handle hObj, unsigned int Index);
```

参数	描述
hObj	小工具的句柄。
Index	以零为基准的项目 （“转轮”要移动到的项）索引。

其他信息

小工具将选择最短的路线开始移动。例如，如果有 7 个项可用，其中第 2 项当前已吸附，则小工具应移到最后一个项，从这里开始向后移动，直至到达第 7 项。
请参阅 “LISTWHEEL_SetPos()”（第 548 页）。

LISTWHEEL_OwnerDraw()

描述

默认函数，可管理某个数据项的绘制操作。

原型

```
int LISTWHEEL_OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

参数	描述
hObj	小工具的句柄。

返回值

该返回值取决于由pDrawItemInfo所指定的WIDGET_ITEM_DRAW_INFO结构的Cmd元素中的命令。

其他信息

如果使用 LISTWHEEL_SetOwnerDraw()，则该函数相当有用。该函数可检索数据项的原始大小和 / 或绘制数据项的文本，并且应当由未经应用程序定义的自绘函数所管理的所有命令来对其进行调用。以下命令由默认函数管理：

- WIDGET_ITEM_GET_XSIZE
- WIDGET_ITEM_GET_YSIZE
- WIDGET_ITEM_DRAW

有关更多信息，请参阅“用户绘制小工具”（第 354 页）、“LISTWHEEL_SetOwnerDraw()”（第 548 页），并参阅提供的示例。

LISTWHEEL_SetBkColor()

之前	之后
	

描述

为已选定和未选定的项设置指定的背景颜色。

原型

```
void LISTWHEEL_SetBkColor(LISTWHEEL_Handle hObj,   unsigned int Index,
                          GUI_COLOR        Color);
```

参数	描述
hObj	小工具的句柄。
Index	参见下面的元素列表。
Color	新的背景颜色。

元素 Index 的允许值	
LISTWHEEL_CI_UNSEL	更改所有未选定项的背景颜色。
LISTWHEEL_CI_SEL	更改已选定项的背景颜色。

LISTWHEEL_SetFont()

之前	之后
	

描述

设置绘制数据项应当使用的字体。

原型

```
void LISTWHEEL_SetFont(LISTWHEEL_Handle              hObj,
                       const GUI_FONT GUI_UNI_PTR * pFont);
```

参数	描述
hObj	小工具的句柄。
pFont	GUI_FONT 结构的指针。

LISTWHEEL_SetLBorder()

之前	之后
	

描述

设置小工具左边缘与文本开头之间的边框大小。

原型

```
void LISTWHEEL_SetLBorder(LISTWHEEL_Handle hObj, unsigned BorderSize);
```

参数	描述
hObj	小工具的句柄。
BorderSize	所需的边框大小。

其他信息

边框大小的默认值为 0。

LISTWHEEL_SetLineHeight()

之前	之后
	

描述

设置绘制数据项所使用的行高。

原型

```
void LISTWHEEL_SetLineHeight(LISTWHEEL_Handle hObj, unsigned LineHeight);
```

参数	描述
hObj	小工具的句柄。
LineHeight	所需的高度。如果默认值为 0，则意味着字体大小将决定行的高度。

其他信息

默认情况下，行高取决于所使用的字体。该函数所设置的值会 “覆盖” 此默认行为。

LISTWHEEL_SetOwnerDraw()

之前	之后
	

描述

为小工具设置应用程序定义的自绘函数，该函数负责绘制小工具项。

原型

```
void LISTWHEEL_SetOwnerDraw(LISTWHEEL_Handle         hObj,
                            WIDGET_DRAW_ITEM_FUNC * pfOwnerDraw);
```

参数	描述
hObj	小工具的句柄。
pfOwnerDraw	自绘函数的指针。

其他信息

该函数将为应用程序定义的函数设置一个指针，必须绘制数据项或需要项的 x 轴或 y 轴大小时，小工具将调用该函数。通过该函数可将任何对象（而不仅仅是纯机器码）绘制为数据项。pfDrawItem 是应用程序定义的 WIDGET_DRAW_ITEM_FUNC 类函数的指针，本章开始部分对此类函数作了解释。执行以下命令。WIDGET_ITEM_GET_YSIZE、WIDGET_ITEM_DRAW、WIDGET_DRAW_BACKGROUND 和 WIDGET_DRAW_OVERLAY。

示例

以下示例例程在小工具上方绘制 2 个红色指示行：

```
static int _OwnerDraw(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_DRAW_OVERLAY:
    GUI_SetColor(GUI_RED);
    GUI_DrawHLine(40, 0, 99);
    GUI_DrawHLine(59, 0, 99);
    break;
  default:
    return LISTWHEEL_OwnerDraw(pDrawItemInfo);
  }
  return 0;
}
```

LISTWHEEL_SetPos()

描述

将小工具的数据区固定到指定位置。

原型

```
void LISTWHEEL_SetPos(LISTWHEEL_Handle hObj, unsigned int Index);
```

参数	描述
hObj	小工具的句柄。
Index	以零为基准的项目 （“转轮”要移动到的项）索引。

其他信息

请参阅 “LISTWHEEL_MoveToPos()”（第 545 页）。

LISTWHEEL_SetRBorder()

之前	之后
	

描述

设置小工具左边缘与文本开头之间的边框大小。

原型

```
void LISTWHEEL_SetRBorder(LISTWHEEL_Handle hObj, unsigned BorderSize);
```

参数	描述
hObj	小工具的句柄。
BorderSize	所需的边框大小。

其他信息

边框大小的默认值为 0。

LISTWHEEL_SetSel()

之前	之后
	

描述

该函数设置已选定的项。

原型

```
void LISTWHEEL_SetSel(LISTWHEEL_Handle hObj, int Sel);
```

参数	描述
hObj	小工具的句柄。
Sel	要选择的项的索引（以零为基准）。

其他信息

仅能选择一个项。默认情况下，将选择包含索引 0 的项。

LISTWHEEL_SetSnapPosition()

之前	之后
	

描述

此函数设置离开小工具顶部的相对位置，在该处应当吸入项。默认情况下，吸附位置为 0，这意味着在小工具的顶部吸入项。

原型

```
void LISTWHEEL_SetSnapPosition(LISTWHEEL_Handle hObj, int SnapPosition);
```

参数	描述
hObj	小工具的句柄。
SnapPosition	离开小工具顶部的相对位置 （以像素为单位），在该位置应当吸入项。

其他信息

函数 LISTWHEEL_GetPos() 可用于获取当前已吸入项的索引 （以零为基准）。

LISTWHEEL_SetText()

之前	之后
	

描述

此函数会删除任何现有项，并添加函数传递的指定项。

原型

```
void LISTWHEEL_SetText(LISTWHEEL_Handle         hObj,
                       const GUI_ConstString * ppText);
```

参数	描述
hObj	小工具的句柄。
ppText	字符串阵列的指针。最后一个项必须为空指针。

其他信息

需要注意的是， ppText 指向的最后一个元素必须为空指针。

示例

以下示例说明函数的使用方法：

```
static char * _apText[] = {
```



```
  "Monday",
  "Tuesday",
  "Wednesday",
  "Thursday",
  "Friday",
  "Saturday",
  "Sunday",
  NULL
};

static void _SetContents(void) {
  LISTWHEEL_SetText(hWin, _apText);
}
```

LISTWHEEL_SetTextAlign()

之前	之后
	

描述

设置绘制小工具的项所使用的文本对齐方式。

原型

```
void LISTWHEEL_SetTextAlign(LISTWHEEL_Handle hObj, int Align);
```

参数	描述
hObj	小工具的句柄。
Align	绘制小工具的项所使用的对齐方式。

其他信息

有关文本对齐方式的详细信息，请参阅 “GUI_SetTextAlign()”（第 82 页）。

LISTWHEEL_SetTextColor()

之前	之后
	

描述

设置绘制文本所使用的颜色。

原型

```
void LISTWHEEL_SetTextColor(LISTWHEEL_Handle hObj,
```

```
                                unsigned int     Index, GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄。
Index	（参见下表）
Color	要使用的颜色。

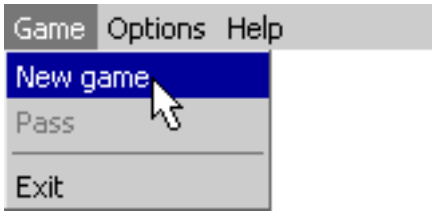
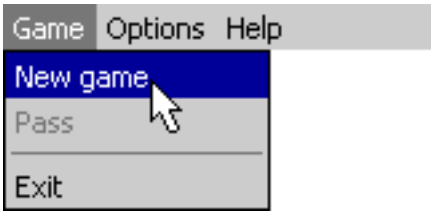
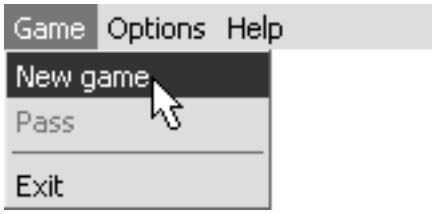
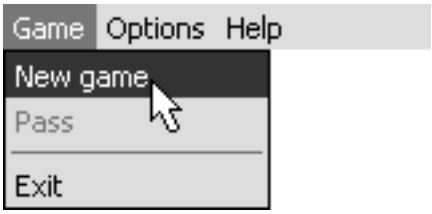
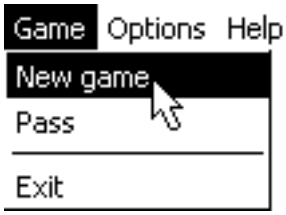
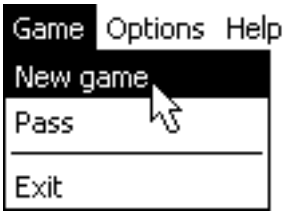
参数 Index 的允许值	
LISTWHEEL_CI_UNSEL	设置未选定文本的颜色。
LISTWHEEL_CI_SEL	设置已选定文本的颜色。

LISTWHEEL_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

15.15 MENU：菜单小工具

MENU 小工具可用于创建若干种菜单。每个菜单项代表一个应用程序命令或子菜单。MENU 可水平显示和 / 或垂直显示。菜单项可使用分隔符进行分组。水平菜单和垂直菜单均支持分隔符。选择一个菜单项会发送 WM_MENU 消息给菜单的所有者，或打开一个子菜单。如果已启用鼠标支持，则 MENU 小工具会对菜单项上方的鼠标移动作出反应。
装运的emWin包含说明MENU小工具使用方法的应用示例。该示例可在Sample\Application\Reversi.c目录下找到。
下表显示水平 MENU 小工具带垂直菜单的外观：

描述	使用 WIDGET_Effect_3D1L 时的菜单	使用 WIDGET_Effect_Simple 时的菜单
彩色显示屏（8666 色模式）		
单色显示屏 （16 灰度级）		
黑 / 白显示屏		

上表显示菜单小工具使用其默认效果 WIDGET_Effect_3D1L 和使用 WIDGET_Effect_Simple 时的相应外观。菜单小工具还可以与所有其他效果配合使用。

15.15.1 菜单消息

为了通知其所有者有关选择一个项或打开一个子菜单的信息，菜单小工具将发送 WM_MENU 类型的消息给其所有者。

WM_MENU

描述

发送此消息的目的是通知菜单的所有者有关选择一个项或打开一个子菜单的信息。已禁用的菜单项将不发送此消息。

数据

指向 MENU_MSG_DATA 结构的消息的 Data.p 指针。

MENU_MSG_DATA 的元素

数据类型	元素	描述
U16	MsgType	（参见下表）
U16	ItemId	菜单项的 Id。

元素 MsgType 的允许值	
MENU_ON_INITMENU	菜单打开之前会立即将此消息发送给菜单的所有者。这使得应用程序有机会在菜单打开之前对其进行修改。
MENU_ON_ITEMACTIVATE	菜单项被高亮显示之后，菜单的所有者窗口将收到此消息。将子菜单高亮显示之后，则不会发送此消息。
MENU_ON_ITEMPRESSED	按下菜单项之后，将发送此消息给小工具的所有者窗口。对于已禁用的菜单项，也将发送此消息。
MENU_ON_ITEMSELECT	选择菜单项之后将立即发送此消息给菜单的所有者。ItemId 元素包含已按下的菜单项的 Id。

示例

以下示例说明如何对 WM_MENU 消息作出反应：

```
void Callback(WM_MESSAGE * pMsg) {
  MENU_MSG_DATA * pData;
  WM_HWIN hWin = pMsg->hWin;
  switch (pMsg->MsgId) {
  case WM_MENU:
    pData = (MENU_MSG_DATA *)pMsg->Data.p;
    switch (pData->MsgType) {
    case MENU_ON_ITEMACTIVATE:
      _UpdateStatusbar(pData->ItemId);
      break;
    case MENU_ON_INITMENU:
      _OnInitMenu();
      break;
    case MENU_ON_ITEMSELECT:
      switch (pData->ItemId) {
      case ID_MENU_ITEM0:
        ... /* React on selection of menu item 0 */
        break;
      case ID_MENU_ITEM1:
        ... /* React on selection of menu item 1 */
        break;
      case ...
        ...
      }
      break;
    }
    break;
  default:
    MENU_Callback(pMsg);
  }
}
```


15.15.2 数据结构

下文说明与菜单小工具相关的数据结构。

MENU_ITEM_DATA

此结构充当一个容器，用于设置或检索有关菜单项的信息。

MENU_ITEM_DATA 的元素

数据类型	元素	描述
const char *	pText	菜单项文本。
U16	Id	菜单项的 Id。
U16	Flags	（参见下表）
MENU_Handle	hSubmenu	如果该菜单项代表一个子菜单，则此元素包含子菜单的句柄。

元素 Flags 的允许值	
MENU_IF_DISABLED	菜单项已被禁用。
MENU_IF_SEPARATOR	菜单项为分隔符。

15.15.3 配置选项

类型	宏	默认值	描述
N	MENU_BKCOLOR0_DEFAULT	GUI_LIGHTGRAY	已启用且未选定的项的背景颜色。
N	MENU_BKCOLOR1_DEFAULT	0x980000	已启用且已选定的项的背景颜色。
N	MENU_BKCOLOR2_DEFAULT	GUI_LIGHTGRAY	已禁用项的背景颜色。
N	MENU_BKCOLOR3_DEFAULT	0x980000	已禁用且已选定的项的背景颜色。
N	MENU_BKCOLOR4_DEFAULT	0x7C7C7C	活动子菜单项的背景颜色。
N	MENU_BORDER_BOTTOM_DEFAULT	2	菜单项文本与菜单项底部之间的边框。
N	MENU_BORDER_LEFT_DEFAULT	4	菜单项文本与菜单项左边缘之间的边框。
N	MENU_BORDER_RIGHT_DEFAULT	4	菜单项文本与菜单项右边缘之间的边框。
N	MENU_BORDER_TOP_DEFAULT	2	菜单项文本与菜单项顶部之间的边框。
S	MENU_EFFECT_DEFAULT	WIDGET_Effect_3D1L	默认效果。
S	MENU_FONT_DEFAULT	GUI_Font13_1	所使用的字体。
N	MENU_TEXTCOLOR0_DEFAULT	GUI_BLACK	已启用且未选定的菜单项的文本颜色。
N	MENU_TEXTCOLOR1_DEFAULT	GUI_WHITE	已启用且已选定的菜单项的文本颜色。
N	MENU_TEXTCOLOR2_DEFAULT	0x7C7C7C	已禁用菜单项的文本颜色。
N	MENU_TEXTCOLOR3_DEFAULT	GUI_LIGHTGRAY	已禁用且已选定的菜单项的文本颜色。
N	MENU_TEXTCOLOR4_DEFAULT	GUI_WHITE	活动子菜单项的文本颜色。

15.15.4 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_RIGHT	- 如果菜单为水平菜单，则选定范围将向右移动一个项。 - 如果菜单为垂直菜单并且当前项为子菜单，则子菜单将会打开，并且输入焦点将移到该子菜单。 - 如果菜单为垂直菜单，但当前项非子菜单，并且顶层菜单为水平菜单，则顶层菜单的下一个项会打开，并且输入焦点将移到该菜单项。
GUI_KEY_LEFT	- 如果菜单为水平菜单，则选定范围将向左移动一个项。 - 如果菜单为垂直菜单而非顶层菜单，则当前菜单会关闭，并且输入焦点将移到上一个菜单。如果上一个菜单为水平菜单，则其上一个子菜单将会打开，并且输入焦点将移到上一个子菜单。
GUI_KEY_DOWN	- 如果菜单为水平菜单并且当前菜单项为子菜单，则此子菜单将会打开。 - 如果菜单为垂直菜单，则选定范围将移到下一个项。
GUI_KEY_UP	- 如果菜单为垂直菜单，则选定范围将移到上一个项。
GUI_KEY_ESCAPE	- 如果菜单不是顶层菜单，则当前菜单将关闭，并且输入焦点将移到上一个菜单。 - 如果菜单为顶层菜单，则当前菜单项将变为未选定。
GUI_KEY_ENTER	- 如果当前菜单项为子菜单，则子菜单将会打开，并且输入焦点将移到该子菜单。 - 如果当前菜单项不是子菜单，则顶层菜单的所有子菜单将会关闭，并且将发送 MENU_ON_ITEMSELECT 消息给菜单的所有者。

15.15.5 “菜单” API

下表按字母顺序列出可用的 emWinMENU 相关例程。例程的详细说明如下：

例程	描述
MENU_AddItem()	将一个项添加到现有菜单。
MENU_Attach()	在指定位置将具有指定大小的菜单附加到指定窗口。
MENU_CreateEx()	创建 MENU 小工具。
MENU_CreateIndirect()	从资源表条目创建 MENU 小工具。
MENU_CreateUser()	使用额外字节作为用户数据创建 MENU 小工具。
MENU_DeleteItem()	删除指定菜单项。
MENU_DisableItem()	禁用指定菜单项。
MENU_EnableItem()	启用指定菜单项。
MENU_GetDefaultBkColor()	返回新菜单的默认背景颜色。
MENU_GetDefaultBorderSize()	返回新菜单的默认边框大小。
MENU_GetDefaultEffect()	返回新菜单的默认效果。
MENU_GetDefaultFont()	返回显示新菜单菜单项文本所使用的默认字体指针。
MENU_GetDefaultTextColor()	返回新菜单的默认文本颜色。
MENU_GetItem()	检索有关指定菜单项的信息。
MENU_GetItemText()	返回指定菜单项的文本。
MENU_GetNumItems()	返回指定菜单的菜单项个数。
MENU_GetOwner()	返回指定菜单的所有者窗口。
MENU_GetUserData()	检索通过 MENU_SetUserData() 所设置的数据。
MENU_InsertItem()	插入一个菜单项。
MENU_Popup()	在指定位置打开弹出菜单。
MENU_SetBkColor()	设置指定菜单的背景颜色。
MENU_SetBorderSize()	设置指定菜单的边框大小。
MENU_SetDefaultBkColor()	设置新菜单的默认背景颜色。
MENU_SetDefaultBorderSize()	设置新菜单的默认边框大小。
MENU_SetDefaultEffect()	设置新菜单的默认效果。
MENU_SetDefaultFont()	设置显示新菜单的菜单项文本所使用的默认字体指针。
MENU_SetDefaultTextColor()	设置新菜单的默认文本颜色。
MENU_SetFont()	设置显示指定菜单的菜单项所使用的字体。

例程	描述
MENU_SetItem()	更改有关指定菜单项的信息。
MENU_SetOwner()	设置将接受菜单通知的窗口。
MENU_SetTextColor()	设置指定菜单的文本颜色。
MENU_SetUserData()	设置 MENU 小工具的额外数据。

MENU_AddItem()

之前	之后
	

描述

此函数会将一个新菜单项添加到指定菜单的末尾。

原型

```
void MENU_AddItem(MENU_Handle hObj, const MENU_ITEM_DATA * pItemData);
```

参数	描述
hObj	小工具的句柄。
pItemData	包含新菜单项信息的 MENU_ITEM_DATA 结构的指针。

其他信息

如果要配合若干个子菜单使用菜单，则菜单项的 Id 必须是唯一的。不同子菜单不能包含具有相同 Id 的菜单项。
给菜单添加项，并且没有使用固定大小时，将调整菜单的大小。
请参阅 “MENU_ITEM_DATA”（第 555 页）。

MENU_Attach()

描述

在指定位置将具有指定大小的指定菜单附加到指定窗口。

原型

```
void MENU_Attach(MENU_Handle hObj,    WM_HWIN hDestWin,
                 int         x,       int     y,
                 int         xSize,   int     ySize,
                 int         Flags);
```

参数	描述
hObj	小工具的句柄。
hDestWin	应附加菜单的窗口的句柄。
x	在菜单的窗口坐标中的 X 轴位置。
y	在菜单的窗口坐标中的 Y 轴位置。
xSize	菜单的固定 X 轴大小。更多详细信息，请参阅 “MENU_CreateEx()”（第 558 页）。
ySize	菜单的固定 Y 轴大小。更多详细信息，请参阅 “MENU_CreateEx()”（第 558 页）。
Flags	保留供日后使用

其他信息

创建菜单小工具之后，此函数可用于将菜单附加到现有窗口。

MENU_CreateEx()

描述

在指定位置创建指定大小的 MENU 小工具。

原型

```
MENU_Handle MENU_CreateEx(int     x0,       int y0,
                          int     xSize,    int ySize,
                          WM_HWIN hParent, int WinFlags,
                          int     ExFlags, int Id);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xSize	小工具的固定水平大小 （以像素为单位）。如果菜单应控制 xSize，则返回值为 0。
ySize	小工具的固定垂直大小 （以像素为单位）。如果菜单应控制 ySize，则返回值为 0。
hParent	父窗口的句柄。如果为 0，则新小工具将作为桌面 （顶层窗口）的子窗口。 在某些情况下，在“未附加”状态下创建菜单小工具并在稍后将其附加到现有窗口将很有用。对于这种情况，可将 WM_UNATTACHED 用作参数。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅“WM_CreateWindow()”（第 309 页））。
ExFlags	（参见下表）
Id	小工具的窗口 ID。

参数 ExFlags 的允许值	
MENU_CF_HORIZONTAL	创建水平菜单。
MENU_CF_VERTICAL	创建垂直菜单。

返回值

所创建的 MENU 小工具的句柄；如果函数执行失败，则返回值为 0。

其他信息

参数 xSize 和 / 或 ySize 指定菜单是否应使用固定宽度和 / 或高度。
如果这些参数 >0，则应使用固定大小。例如，如果菜单应作为水平菜单附加到窗口顶部，则有必要使用能够覆盖整个窗口顶部的固定 X 轴大小。在这种情况下，参数 xSize 可用于设置菜单的固定 X 轴大小。附加或删除具有固定大小的菜单项时，小工具的大小不会改变。
如果值为 0，则菜单自行控制其大小。这意味着菜单的大小取决于菜单的当前菜单项的大小。如果添加或删除项，将调整小工具的大小。

MENU_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。

MENU_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细说明，可参阅函数 MENU_CreateEx()。

MENU_DeleteItem()

之前	之后
	

描述

将指定菜单条目从菜单中删除。

原型

```
void MENU_DeleteItem(MENU_Handle hObj, U16 ItemId);
```

参数	描述
hObj	小工具的句柄。
ItemId	要删除的菜单项的 Id。

其他信息

如果菜单项不存在，则函数将立即返回。
将菜单项从菜单中删除并且没有使用固定大小时，将调整窗口大小。

MENU_DisableItem()

之前	之后
	

描述

禁用指定菜单项。

原型

```
void MENU_DisableItem(MENU_Handle hObj, U16 ItemId);
```

参数	描述
hObj	小工具的句柄。
ItemId	要禁用的菜单项的 Id。

其他信息

如果选择已禁用的菜单项，则菜单小工具将不会发送 WM_MENU 消息给所有者。不能打开已禁用的子菜单项。

MENU_EnableItem()

之前	之后
	

描述

启用指定菜单。

原型

```
void MENU_EnableItem(MENU_Handle hObj, U16 ItemId);
```

参数	描述
hObj	小工具的句柄。
ItemId	要启用的菜单项的 Id。

其他信息

更多详细信息，请参阅 “MENU_DisableItem()”（第 559 页）。

MENU_GetDefaultBkColor()

描述

返回绘制新菜单项所使用的默认背景颜色。

原型

```
GUI_COLOR MENU_GetDefaultBkColor(unsigned ColorIndex);
```

参数	描述
ColorIndex	将返回的颜色的索引 （见下表）。

参数 ColorIndex 的允许值	
MENU_CI_ACTIVE_SUBMENU	活动子菜单项的背景颜色。
MENU_CI_DISABLED	已禁用的菜单项的背景颜色。
MENU_CI_DISABLED_SEL	已禁用且已选定的菜单项的背景颜色。
MENU_CI_ENABLED	已启用且未选定的菜单项的背景颜色。
MENU_CI_SELECTED	已启用且已选定的菜单项的背景颜色。

返回值

绘制新菜单项所使用的默认背景颜色。

其他信息

更多详细信息，请参阅 “MENU_SetBkColor()”（第 564 页）。

MENU_GetDefaultBorderSize()

描述

返回新菜单小工具所使用的默认边框大小。

原型

```
U8 MENU_GetDefaultBorderSize(unsigned BorderIndex);
```

参数	描述
BorderIndex	（参见下表）

参数 BorderIndex 的允许值	
MENU_BI_BOTTOM	菜单项文本与菜单项底部之间的边框。
MENU_BI_LEFT	菜单项文本与菜单项左边缘之间的边框。
MENU_BI_RIGHT	菜单项文本与菜单项右边缘之间的边框
MENU_BI_TOP	菜单项文本与菜单项顶部之间的边框。

返回值

新菜单小工具所使用的默认边框大小。

其他信息

更多详细信息，请参阅 “MENU_SetBorderSize()”（第 565 页）。

MENU_GetDefaultEffect()

描述

返回新菜单的默认效果。

原型

```
const WIDGET_EFFECT * MENU_GetDefaultEffect(void);
```

返回值

此函数的结果是 WIDGET_EFFECT 结构的指针。

其他信息

更多详细信息，请参阅 “WIDGET_SetDefaultEffect()”（第 353 页）。

MENU_GetDefaultFont()

描述

返回显示新菜单菜单项文本所使用的默认字体指针。

原型

```
const GUI_FONT * MENU_GetDefaultFont(void);
```

返回值

显示新菜单菜单项所使用的默认字体指针。

MENU_GetDefaultTextColor()

描述

返回新菜单的默认文本颜色。

原型

```
GUI_COLOR MENU_GetDefaultTextColor(unsigned ColorIndex);
```

参数	描述
ColorIndex	将返回的颜色的索引 （见下表）

参数 ColorIndex 的允许值	
MENU_CI_ACTIVE_SUBMENU	活动子菜单项的文本颜色。
MENU_CI_DISABLED	已禁用的菜单项的文本颜色。
MENU_CI_DISABLED_SEL	已禁用且已选定的菜单项的文本颜色。
MENU_CI_ENABLED	已启用且未选定的菜单项的文本颜色。
MENU_CI_SELECTED	已启用且已选定的菜单项的文本颜色。

返回值

新菜单的默认文本颜色。

其他信息

更多详细信息，请参阅 “MENU_SetDefaultTextColor()”（第 567 页）。

MENU_GetItem()

描述

检索有关指定菜单项的信息。

原型

```
void MENU_GetItem(MENU_Handle hObj, U16 ItemId, MENU_ITEM_DATA * pItemData);
```

参数	描述
hObj	小工具的句柄。
ItemId	请求的菜单项的 Id。
pItemData	由该函数填充的 MENU_ITEM_DATA 结构的指针。

其他信息

如果要配合若干个子菜单使用菜单，则小工具的句柄必须包含请求项的菜单 / 子菜单，或者必须为更高层菜单 / 子菜单的句柄。
此函数会将 MENU_ITEM_INFO 数据结构的元素 pText 设置为 0。如果要检索菜单项文本，则应使用函数 MENU_GetItemText()。
有关 MENU_ITEM_INFO 数据结构的详细信息，请参阅 “菜单”一章的开始部分。

MENU_GetItemText()

描述

返回指定菜单项的文本。

原型

```
void MENU_GetItemText(MENU_Handle   hObj,    U16      ItemId,
```



```
                                char        * pBuffer, unsigned BufferSize);
```

参数	描述
hObj	小工具的句柄。
ItemId	请求的菜单项的 Id。
pBuffer	将由该函数填充的缓冲区。
BufferSize	将检索的最大字节数。

MENU_GetNumItems()

描述

返回指定菜单的菜单项个数。

原型

```
unsigned MENU_GetNumItems(MENU_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

指定菜单的菜单项个数。

MENU_GetOwner()

描述

返回指定菜单的所有者窗口。

原型

```
WM_HWIN MENU_GetOwner(MENU_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

指定菜单的所有者窗口。

MENU_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

MENU_InsertItem()

之前	之后
	

描述

在指定位置插入一个菜单项。

原型

```
void MENU_InsertItem(MENU_Handle hObj, U16 ItemId,
```



```
                                const MENU_ITEM_DATA * pItemData);
```

参数	描述
hObj	小工具的句柄。
ItemId	菜单项 （在其之前应插入新菜单项）的 Id。
pItemData	包含新菜单项信息的 MENU_ITEM_DATA 结构的指针。

其他信息

有关 MENU_ITEM_INFO 数据结构的详细信息，请参阅 “菜单”一章的开始部分。

MENU_Popup()

描述

在指定位置打开指定菜单。选择一个菜单项或触按菜单之外的显示屏后，弹出菜单将关闭。

原型

```
void MENU_Popup(MENU_Handle hObj,    WM_HWIN hDestWin,
                int         x,       int     y,
                int         xSize,   int     ySize,
                int         Flags);
```

参数	描述
hObj	小工具的句柄。
hDestWin	应附加菜单的窗口的句柄。
x	在菜单的窗口坐标中的 X 轴位置。
y	在菜单的窗口坐标中的 Y 轴位置。
xSize	菜单的固定 X 轴大小。更多详细信息，请参阅 “MENU_CreateEx()”（第 558 页）。
ySize	菜单的固定 Y 轴大小。更多详细信息，请参阅 “MENU_CreateEx()”（第 558 页）。
Flags	保留供日后使用

其他信息

选择一个菜单项或触按弹出菜单之外的显示屏后，菜单将关闭。需要注意的是，该菜单不会被自动删除。“样本”文件夹中包含示例 WIDGET_PopupMenu.c，它显示了如何使用该函数。

MENU_SetBkColor()

之前	之后
	

描述

设置指定菜单的背景颜色。

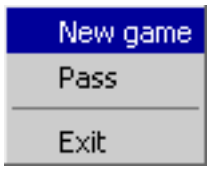
原型

```
void MENU_SetBkColor(MENU_Handle hObj,   unsigned ColorIndex,
                     GUI_COLOR   Color);
```

参数	描述
hObj	小工具的句柄。
ColorIndex	颜色的索引（参见下表）
Color	要使用的颜色。

参数 ColorIndex 的允许值	
MENU_CI_ACTIVE_SUBMENU	活动子菜单项的背景颜色。
MENU_CI_DISABLED	已禁用的菜单项的背景颜色。
MENU_CI_DISABLED_SEL	已禁用且已选定的菜单项的背景颜色。
MENU_CI_ENABLED	已启用且未选定的菜单项的背景颜色。
MENU_CI_SELECTED	已启用且已选定的菜单项的背景颜色。

MENU_SetBorderSize()

之前	之后
	

将在上述屏幕截图之间执行以下代码：

```
MENU_SetBorderSize(hMenuGame, MENU_BI_LEFT, 20);
```

之前	之后
	

将在上述屏幕截图之间执行以下代码：

```
MENU_SetBorderSize(hMenu, MENU_BI_LEFT, 10);
MENU_SetBorderSize(hMenu, MENU_BI_RIGHT, 10);
```

描述

设置指定菜单的边框大小。

原型

```
void MENU_SetBorderSize(MENU_Handle hObj,         unsigned BorderIndex,
                        U8          BorderSize);
```

参数	描述
hObj	小工具的句柄。
BorderIndex	（参见下表）
BorderSize	要使用的大小。

参数 BorderIndex 的允许值	
MENU_BI_BOTTOM	菜单项文本与菜单项底部之间的边框
MENU_BI_LEFT	菜单项文本与菜单项左边缘之间的边框
MENU_BI_RIGHT	菜单项文本与菜单项右边缘之间的边框
MENU_BI_TOP	菜单项文本与菜单项顶部之间的边框

MENU_SetDefaultBkColor()

描述

设置绘制新菜单项所使用的默认背景颜色。

原型

```
void MENU_SetDefaultBkColor(unsigned ColorIndex, GUI_COLOR Color);
```

参数	描述
ColorIndex	将返回的颜色的索引 （见下表）。
Color	要使用的颜色。

参数 ColorIndex 的允许值	
MENU_CI_ACTIVE_SUBMENU	活动子菜单项的背景颜色。
MENU_CI_DISABLED	已禁用的菜单项的背景颜色。
MENU_CI_DISABLED_SEL	已禁用且已选定的菜单项的背景颜色。
MENU_CI_ENABLED	已启用且未选定的菜单项的背景颜色。
MENU_CI_SELECTED	已启用且已选定的菜单项的背景颜色。

其他信息

更多详细信息，请参阅 “MENU_SetBkColor()”（第 564 页）。

MENU_SetDefaultBorderSize()

描述

设置新菜单小工具所使用的默认边框大小。

原型

```
void MENU_SetDefaultBorderSize(unsigned BorderIndex, U8 BorderSize);
```

参数	描述
BorderIndex	（参见下表）
BorderSize	要使用的边框大小。

参数 BorderIndex 的允许值	
MENU_BI_BOTTOM	菜单项文本与菜单项底部之间的边框
MENU_BI_LEFT	菜单项文本与菜单项左边缘之间的边框
MENU_BI_RIGHT	菜单项文本与菜单项右边缘之间的边框
MENU_BI_TOP	菜单项文本与菜单项顶部之间的边框

其他信息

更多详细信息，请参阅 “MENU_SetBorderSize()”（第 565 页）。

MENU_SetDefaultEffect()

描述

设置新菜单的默认效果。

原型

```
void MENU_SetDefaultEffect(const WIDGET_EFFECT * pEffect);
```

参数	描述
pEffect	WIDGET_EFFECT 结构的指针。

其他信息

更多详细信息，请参阅 “WIDGET_SetDefaultEffect()”（第 353 页）。

MENU_SetDefaultFont()

描述

设置显示新菜单菜单项文本所使用的默认字体指针。

原型

```
void MENU_SetDefaultFont(const GUI_FONT * pFont);
```

参数	描述
pFont	要使用的 GUI_FONT 结构的指针。

其他信息

更多详细信息，请参阅 “MENU_SetFont()”（第 568 页）。

MENU_SetDefaultTextColor()

描述

设置新菜单的默认文本颜色。

原型

```
void MENU_SetDefaultTextColor(unsigned ColorIndex, GUI_COLOR Color);
```

参数	描述
ColorIndex	要使用的颜色的索引 （见下表）
Color	要使用的颜色

参数 ColorIndex 的允许值	
MENU_CI_ACTIVE_SUBMENU	活动子菜单项的文本颜色。
MENU_CI_DISABLED	已禁用的菜单项的文本颜色。
MENU_CI_DISABLED_SEL	已禁用且已选定的菜单项的文本颜色。
MENU_CI_ENABLED	已启用且未选定的菜单项的文本颜色。
MENU_CI_SELECTED	已启用且已选定的菜单项的文本颜色。

其他信息

更多详细信息，请参阅 “MENU_SetTextColor()”（第 569 页）。

MENU_SetFont()

之前	之后
	

描述

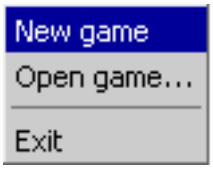
设置显示新菜单菜单项文本所使用的默认字体指针。

原型

```
void MENU_SetFont(MENU_Handle hObj, const GUI_FONT * pFont);
```

参数	描述
hObj	小工具的句柄。
pFont	要使用的 GUI_FONT 结构的指针。

MENU_SetItem()

之前	之后
	

描述

设置指定菜单项的菜单项信息。

原型

```
void MENU_SetItem(MENU_Handle            hObj,       U16 ItemId,
                  const MENU_ITEM_DATA * pItemData);
```

参数	描述
hObj	小工具的句柄。
ItemId	要更改的菜单项的 Id。
pItemData	包含新信息的 MENU_ITEM_DATA 结构的指针。

MENU_SetOwner()

描述

设置将接受小工具通知的菜单所有者。

原型

```
void MENU_SetOwner(MENU_Handle hObj, WM_HWIN hOwner);
```

参数	描述
hObj	小工具的句柄。
hOwner	应接受菜单 WM_MENU 消息的所有者窗口的句柄。

其他信息

如果未设置任何所有者，则菜单的父窗口将接收 WM_MENU 消息。而在某些情况下，可不将消息发送至菜单的父窗口。此时，该函数可用来设置 WM_MENU 消息的收件人。

MENU_SetSel()

之前	之后
	

描述

设置指定菜单已选定的项。

原型

```
void MENU_SetSel(MENU_Handle hObj, int Sel);
```

参数	描述
hObj	小工具的句柄。
Sel	要选择的菜单项的索引（以零为基准）。

返回值

此函数将返回上一个已选定的菜单项的索引（以零为基准）。

其他信息

如果参数 Sel 的值 <0，则会取消选择菜单项。

MENU_SetTextColor()

之前	之后
	

描述

设置指定菜单的文本颜色。

原型

```
void MENU_SetTextColor(MENU_Handle hObj,   unsigned ColorIndex,
                       GUI_COLOR   Color);
```

参数	描述
hObj	小工具的句柄。
ColorIndex	要使用的颜色的索引 （见下表）
Color	要使用的颜色。

参数 ColorIndex 的允许值	
MENU_CI_ACTIVE_SUBMENU	活动子菜单项的文本颜色。
MENU_CI_DISABLED	已禁用的菜单项的文本颜色。
MENU_CI_DISABLED_SEL	已禁用且已选定的菜单项的文本颜色。
MENU_CI_ENABLED	已启用且未选定的菜单项的文本颜色。
MENU_CI_SELECTED	已启用且已选定的菜单项的文本颜色。

MENU_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

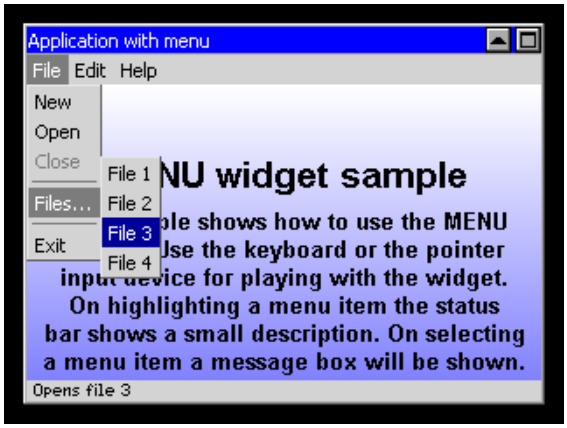
15.15.6 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_Menu.c

需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_Menu.c 的屏幕截图：



15.16 MESSAGEBOX：消息框小工具

使用 MESSAGEBOX 小工具可在带有标题栏和 “确定” 按钮 （必须按下才能关闭窗口）的框架窗口中显示消息。创建消息框或者创建并执行消息框仅需一行代码。所有 MESSAGEBOX 相关例程包含于文件 MESSAGEBOX*.c、MESSAGEBOX.h 和 GUI.h 中。下表显示 MESSAGEBOX 小工具的外观：

简单消息框


15.16.1 配置选项

类型	宏	默认值	描述
N	MESSAGEBOX_BORDER	4	消息框的元素与客户端窗口框的元素之间的距离。
N	MESSAGEBOX_XSIZEOK	50	“确定” 按钮的 X 轴大小。
N	MESSAGEBOX_YSIZEOK	20	“确定” 按钮的 Y 轴大小。
S	MESSAGEBOX_BKCOLOR	GUI_WHITE	客户端窗口背景的颜色。

15.16.2 键盘反应

小工具包含框窗口、文本和按钮小工具。执行消息框时，按钮将获得输入焦点。更多详细信息，请参阅 “键盘反应”（第 357 页）。

15.16.3 MESSAGEBOX API

下表按字母顺序列出可用的 emWin MESSAGEBOX 相关例程。例程的详细说明如下。

例程	描述
GUI_MessageBox()	创建并显示消息框。
MESSAGEBOX_Create()	创建消息框。

GUI_MessageBox()

描述

创建并显示消息框。

原型

```
int GUI_MessageBox(const char* sMessage, const char* sCaption, int Flags);
```

参数	描述
sMessage	要显示的消息。
sCaption	框架窗口的标题栏的说明。
Flags	（参见下表）

参数 Flags 的允许值	
GUI_MESSAGEBOX_CF_MOVEABLE	可以通过拖动标题栏或窗口框来移动消息框。
0	无功能。

其他信息

通过该函数只需一行代码就能创建并执行消息框。“样本”文件夹中包含示例 DIALOG_MessageBox.c，它显示了如何使用该函数。
有关拖放的详细信息，请参阅 “FRAMEWIN_SetMoveable()”（第 437 页）。

MESSAGEBOX_Create()

描述

创建消息框。

原型

```
WM_HWIN GUI_MessageBox(const char* sMessage, const char* sCaption, int
Flags);
```

参数	描述
sMessage	要显示的消息。
sCaption	框架窗口的标题栏的说明。
Flags	（参见下表）

参数 Flags 的允许值	
GUI_MESSAGEBOX_CF_MODAL	创建模态消息框。默认选择是创建非模态消息框。

返回值

消息框窗口的句柄。

其他信息

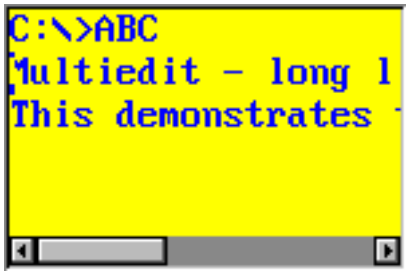
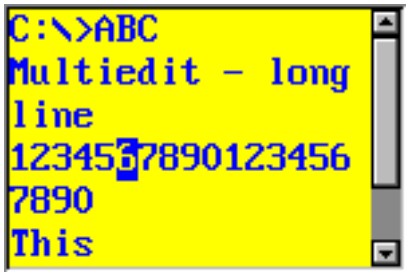
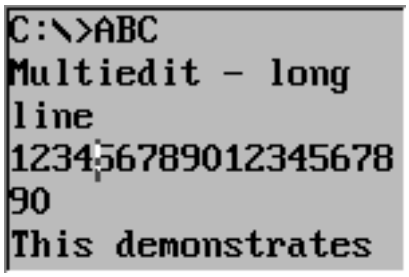
该函数创建的消息框将包含标题栏带有说明的框架窗口、带有消息文本的文本小工具和代表 “确定”按钮的按钮小工具。创建消息框之后，可以通过用户定义的回调函数来更改对话框行为，或通过小工具 API 函数来修改对话框项的属性。可使用下列 Id 来访问这些项：

Id	描述
GUI_ID_TEXT0	包含消息文本的 “文本”小工具的 Id。
GUI_ID_OK	“确定”按钮小工具的 Id。

通过该函数返回的句柄可访问框架窗口。
应使用函数 GUI_ExecCreatedDialog() 来执行消息框。

15.17 MULTIEDIT：多行文本小工具

通过 MULTIEDIT 小工具可编辑多行文本。它既可以被用作简单的文本编辑器，也可以用来显示静态文本。该小工具支持带滚动条和不带滚动条的滚动。所有MULTIEDIT相关例程包含于文件MULTIEDIT*.c和 MULTIEDIT.h 中。所有标识符均以 MULTIEDIT 为前缀。下表显示 MULTIEDIT 小工具的外观：

描述	框架窗口
编辑模式， 自动的水平滚动条， 无换行模式， 插入模式，	
编辑模式， 自动的垂直滚动条， 字词换行模式， 覆盖模式，	
只读模式， 字词换行模式	

15.17.1 配置选项

类型	宏	默认值	描述
S	MULTIEDIT_FONT_DEFAULT	GUI_Font13_1	所使用的字体。
N	MULTIEDIT_BKCOLOR0_DEFAULT	GUI_WHITE	背景颜色。
N	MULTIEDIT_BKCOLOR2_DEFAULT	0xC0C0C0	背景颜色只读模式。
N	MULTIEDIT_TEXTCOLOR0_DEFAULT	GUI_BLACK	文本颜色。
N	MULTIEDIT_TEXTCOLOR2_DEFAULT	GUI_BLACK	文本颜色只读模式。

15.17.2 通知代码

以下事件将作为 WM_NOTIFY_PARENT 消息的一部分从小工具发送至其父窗口：

消息	描述
WM_NOTIFICATION_CLICKED	已单击小工具。
WM_NOTIFICATION_RELEASED	已释放小工具。
WM_NOTIFICATION_MOVED_OUT	已单击小工具，并且指针已移出小工具，但没有释放。
WM_NOTIFICATION_SCROLL_CHANGED	可选滚动条的滚动位置已更改。
WM_NOTIFICATION_VALUE_CHANGED	小工具的文本已更改。

15.17.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_UP	将光标向上移动一行。
GUI_KEY_DOWN	将光标向下移动一行。
GUI_KEY_RIGHT	将光标向右移动一个字符。
GUI_KEY_LEFT	将光标向左移动一个字符。
GUI_KEY_END	将光标移到当前行的末尾。
GUI_KEY_HOME	将光标移到当前行的开头。
GUI_KEY_BACKSPACE	如果小工具在读 / 写模式下工作，此按键将删除光标之前的字符。
GUI_KEY_DELETE	如果小工具在读 / 写模式下工作，此按键将删除光标之下的字符。
GUI_KEY_INSERT	在插入模式和覆盖模式之间切换。
GUI_KEY_ENTER	如果小工具在读 / 写模式下工作，此按键将在当前位置插入一个新行 ('\n')。如果小工具在只读模式下工作，光标将移到下一行的开头。

15.17.4 MULTIEDIT API

下表按字母顺序列出可用的 emWin MULTIEDIT 相关例程。例程的详细说明如下：

例程	描述
MULTIEDIT_AddKey()	按键输入例程。
MULTIEDIT_AddText()	在当前光标位置添加更多文本。
MULTIEDIT_Create()	创建 MULTIEDIT 小工具。（弃用）
MULTIEDIT_CreateEx()	创建 MULTIEDIT 小工具。
MULTIEDIT_CreateIndirect()	从资源表条目创建 MULTIEDIT 小工具。
MULTIEDIT_CreateUser()	使用作为用户数据的额外字节来创建 MULTIEDIT 小工具。
MULTIEDIT_EnableBlink()	启用 / 禁用闪烁光标。
MULTIEDIT_GetCursorCharPos()	返回光标位置的字符编号。
MULTIEDIT_GetCursorPixelPos()	返回光标的像素位置。
MULTIEDIT_GetPrompt()	返回提示符的文本。
MULTIEDIT_GetText()	返回文本。
MULTIEDIT_GetTextSize()	返回当前文本所使用的缓冲区大小。

例程	描述
MULTIEDIT_GetUserData()	检索通过 MULTIEDIT_SetUserData() 设置的数据。
MULTIEDIT_SetAutoScrollH()	激活自动使用水平滚动条。
MULTIEDIT_SetAutoScrollV()	激活自动使用垂直滚动条。
MULTIEDIT_SetBkColor()	设置背景颜色。
MULTIEDIT_SetBufferSize()	设置用于文本和提示符的缓冲区大小。
MULTIEDIT_SetCursorOffset()	将光标固定到指定字符。
MULTIEDIT_SetFont()	设置字体。
MULTIEDIT_SetInsertMode()	启用 / 禁用插入模式。
MULTIEDIT_SetMaxNumChars()	设置最大字符 （包括提示符）数。
MULTIEDIT_SetPasswordMode()	启用 / 禁用密码模式。
MULTIEDIT_SetPrompt()	设置提示文本。
MULTIEDIT_SetReadOnly()	启用 / 禁用只读模式。
MULTIEDIT_SetText()	设置文本。
MULTIEDIT_SetTextAlign()	设置文本对齐方式。
MULTIEDIT_SetTextColor()	设置文本颜色。
MULTIEDIT_SetUserData()	设置 MULTIEDIT 小工具的额外数据。
MULTIEDIT_SetWrapWord()	启用 / 禁用字词换行。
MULTIEDIT_SetWrapNone()	启用 / 禁用无换行模式。

MULTIEDIT_AddKey()

描述

将用户输入添加到指定的 multiedit 小工具。

原型

```
void MULTIEDIT_AddKey(MULTIEDIT_HANDLE hObj, int Key);
```

参数	描述
hObj	Multiedit 小工具的句柄。
Key	要添加的字符。

其他信息

Multiedit 小工具的用户输入中将增加指定字符。如果已达到最大字符数，则不会添加其他字符。

MULTIEDIT_AddText()

描述

在当前光标位置添加指定文本。

原型

```
int MULTIEDIT_AddText(MULTIEDIT_HANDLE hObj, const char * s);
```

参数	描述
hObj	Multiedit 小工具的句柄。
s	要添加的文本 （以空终止符结尾）的指针。

其他信息

如果字符数超出通过函数 MULTIEDIT_SetMaxNumChars() 所设置的最大限值，则该函数将仅添加适合该小工具的文本字符。

MULTIEDIT_Create()

（弃用，应转而使用 MULTIEDIT_CreateEx()）

描述

在指定位置创建指定大小的 MULTIEDIT 小工具。

原型

```
MULTIEDIT_HANDLE MULTIEDIT_Create(int          x0,      int y0,
                                  int          xsize,   int ysize,
                                  WM_HWIN      hParent, int Id,
                                  int          Flags,   int ExFlags,
                                  const char * pText,   int MaxLen);
```

参数	描述
x0	Multiedit 小工具最左侧的像素 （在父坐标中）。
y0	Multiedit 小工具最顶端的像素 （在父坐标中）。
xsize	Multiedit 小工具的水平大小 （以像素为单位）。
ysize	Multiedit 小工具的垂直大小 （以像素为单位）。
hParent	Multiedit 小工具的父窗口。
Id	Multiedit 小工具的 ID。
Flags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	（参见下表）
pText	要使用的文本。
MaxLen	用于文本和提示符的字节的最大个数。

参数 ExFlags 的允许值	
MULTIEDIT_CF_AUTOSCROLLBAR_H	自动使用水平滚动条。
MULTIEDIT_CF_AUTOSCROLLBAR_V	自动使用垂直滚动条。
MULTIEDIT_CF_INSERT	启用插入模式。
MULTIEDIT_CF_READONLY	启用只读模式。

返回值

所创建的 MULTIEDIT 小工具的句柄；如果函数执行失败，则返回值为 0。

MULTIEDIT_CreateEx()

描述

在指定位置创建指定大小的 MULTIEDIT 小工具。

原型

```
MULTIEDIT_HANDLE MULTIEDIT_CreateEx(int          x0,          int y0,
                                    int          xsize,       int ysize,
                                    WM_HWIN      hParent,     int WinFlags,
                                    int          ExFlags,     int Id,
                                    int          BufferSize,
```



```
                                    const char * pText);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新创建的 MULTIEDIT 小工具将作为桌面 （顶层窗口）的子窗口。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW（有关可用参数值的列表，请参阅“窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	（参见下表）
Id	小工具的窗口 ID。
BufferSize	小工具的初始文本缓冲区大小。使用 MULTIEDIT_SetMaxNumChars 来设置最大字符数。
pText	要使用的文本。

参数 ExFlags 的允许值	
MULTIEDIT_CF_AUTOSCROLLBAR_H	自动使用水平滚动条。
MULTIEDIT_CF_AUTOSCROLLBAR_V	自动使用垂直滚动条。
MULTIEDIT_CF_INSERT	启用插入模式。
MULTIEDIT_CF_READONLY	启用只读模式。

返回值

所创建的 MULTIEDIT 小工具的句柄；如果函数执行失败，则返回值为 0。

MULTIEDIT_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。

MULTIEDIT_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细说明，可参阅函数 MULTIEDIT_CreateEx()。

MULTIEDIT_EnableBlink()

描述

启用 / 禁用闪烁光标。

原型

```
void MULTIEDIT_EnableBlink(EDIT_Handle hObj, int Period, int OnOff);
```

参数	描述
hObj	编辑字段的句柄
Period	闪烁周期
OnOff	1 表示启用闪烁； 0 表示禁用闪烁

其他信息

此函数将调用 GUI_X_GetTime()。

MULTIEDIT_GetCursorCharPos()

描述

返回光标位置的字符编号。

原型

```
int MULTIEDIT_GetCursorCharPos(MULTIEDIT_Handle hObj);
```

参数	描述
hObj	Multiedit 小工具的句柄。

返回值

光标位置的字符编号。

其他信息

无论小工具是否具有焦点，它都将返回字符位置。这就意味着，如果光标当前在小工具中不可见，将同时返回光标位置。

MULTIEDIT_GetCursorPixelPos()

描述

返回光标在窗口坐标中的像素位置。

原型

```
void MULTIEDIT_GetCursorPixelPos(MULTIEDIT_Handle   hObj,
                                 int              * pxPos, int * pyPos);
```

参数	描述
hObj	Multiedit 小工具的句柄。
pxPos	窗口坐标中 X 轴位置的整数变量指针。
pyPos	窗口坐标中 Y 轴位置的整数变量指针。

其他信息

无论小工具是否具有焦点，它都将返回像素位置。这就意味着，如果光标当前在小工具中不可见，将同时返回光标位置。

MULTIEDIT_GetPrompt()

描述

返回当前的提示文本。

原型

```
void MULTIEDIT_GetPrompt(MULTIEDIT_HANDLE hObj, char * sDest, int MaxLen);
```

参数	描述
hObj	Multiedit 小工具的句柄。
sDest	要返回的提示文本的缓冲区。
MaxLen	要复制到 sDest 的最大字节数。

其他信息

此函数会将当前的提示文本复制到sDest指定的缓冲区。复制到缓冲区的最大字节数由MaxLen指定。

MULTIEDIT_GetText()

描述

返回当前文本。

原型

```
void MULTIEDIT_GetText(MULTIEDIT_HANDLE hObj, char * sDest, int MaxLen);
```

参数	描述
hObj	Multiedit 小工具的句柄。
sDest	要返回的文本的缓冲区。
MaxLen	要复制到 sDest 的最大字节数。

其他信息

该函数会将当前文本复制到 sDest 指定的缓冲区。复制到缓冲区的最大字节数由 MaxLen 指定。

MULTIEDIT_GetTextSize()

描述

返回用于存储当前文本 （和提示符）的缓冲区大小。

原型

```
int MULTIEDIT_GetTextSize(MULTIEDIT_HANDLE hObj);
```

参数	描述
hObj	Multiedit 小工具的句柄。

返回值

用于存档当前文本 （和提示符）的缓冲区大小。

MULTIEDIT_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

MULTIEDIT_SetAutoScrollH()

描述

启用 / 禁用自动使用水平滚动条。

原型

```
void MULTIEDIT_SetAutoScrollH(MULTIEDIT_HANDLE hObj, int OnOff);
```

参数	描述
hObj	Multiedit 小工具的句柄。
OnOff	（参见下表）

参数 OnOff 的允许值	
0	禁用自动使用水平滚动条。
1	启用自动使用水平滚动条。

其他信息

启用自动使用水平滚动条仅适用于无换行模式 （本章后续部分将加以解释） 。如果启用该功能，则 multiedit 小工具将检查无换行文本的宽度是否适合客户端区域。如果未启用该功能，则水平滚动条将附加到窗口。

MULTIEDIT_SetAutoScrollV()

描述

启用 / 禁用自动使用垂直滚动条。

原型

```
void MULTIEDIT_SetAutoScrollV(MULTIEDIT_HANDLE hObj, int OnOff);
```

参数	描述
hObj	Multiedit 小工具的句柄。
OnOff	（参见下表）

参数 OnOff 的允许值	
0	禁用自动使用垂直滚动条。
1	启用自动使用垂直滚动条。

其他信息

如果启用该功能，则 multiedit 小工具将检查文本的高度是否适合客户端区域。如果未启用该功能，则垂直滚动条将附加到窗口。

MULTIEDIT_SetBkColor()

描述

设置指定 multiedit 小工具的背景颜色。

原型

```
void MULTIEDIT_SetBkColor(MULTIEDIT_HANDLE hObj,   unsigned int Index,
                          GUI_COLOR        Color);
```

参数	描述
hObj	Multiedit 小工具的句柄。
Index	（参见下表）
Color	要使用的背景颜色。

参数 Index 的允许值	
MULTIEDIT_CI_EDIT	编辑模式。
MULTIEDIT_CI_READONLY	只读模式。

MULTIEDIT_SetBufferSize()

描述

设置文本和提示符所使用的最大字节数。

原型

```
void MULTIEDIT_SetBufferSize(MULTIEDIT_HANDLE hObj, int BufferSize);
```

参数	描述
hObj	Multiedit 小工具的句柄。
BufferSize	最大字节数。

其他信息

该函数会清除 multiedit 小工具的当前内容，并为文本和提示符分配指定字节数。

MULTIEDIT_SetCursorOffset()

描述

将光标位置固定到指定字符。

原型

```
void MULTIEDIT_SetCursorOffset(MULTIEDIT_HANDLE hObj, int Offset);
```

参数	描述
hObj	Multiedit 小工具的句柄。
Offset	新的光标位置。

其他信息

必须将提示符所使用的字符数添加到参数 Offset。如果使用提示符，则参数 Offset 的值不能小于提示符所使用的字符数。

MULTIEDIT_SetFont()

描述

设置显示文本和提示符所使用的字体。

原型

```
void MULTIEDIT_SetFont(MULTIEDIT_HANDLE hObj, const GUI_FONT * pFont);
```

参数	描述
hObj	Multiedit 小工具的句柄。
pFont	要使用的字体的指针。

MULTIEDIT_SetInsertMode()

描述

启用 / 禁用插入模式。默认行为是覆盖模式。

原型

```
void MULTIEDIT_SetInsertMode(MULTIEDIT_HANDLE hObj, int OnOff);
```

参数	描述
hObj	Multiedit 小工具的句柄。
OnOff	（参见下表）

参数 OnOff 的允许值	
0	禁用插入模式。
1	启用插入模式。

MULTIEDIT_SetMaxNumChars()

描述

设置文本和提示符所使用的最大字符数。

原型

```
void MULTIEDIT_SetMaxNumChars(MULTIEDIT_HANDLE hObj, unsigned MaxNumChars);
```

参数	描述
hObj	Multiedit 小工具的句柄。
MaxNumChars	最大字符数。

MULTIEDIT_SetPasswordMode()

描述

启用 / 禁用密码模式。

原型

```
void MULTIEDIT_SetPasswordMode(MULTIEDIT_HANDLE hObj, int OnOff);
```

参数	描述
hObj	Multiedit 小工具的句柄。
OnOff	（参见下表）

参数 OnOff 的允许值	
0	禁用密码模式。
1	启用密码模式。

其他信息

使用密码模式可隐藏用户输入。

MULTIEDIT_SetPrompt()

描述

设置提示文本。

原型

```
void MULTIEDIT_SetPrompt(MULTIEDIT_HANDLE hObj, const char * sPrompt);
```

参数	描述
hObj	Multiedit 小工具的句柄。
sPrompt	新的提示文本的指针。

其他信息

提示文本首先显示。光标不能移到提示符中。

MULTIEDIT_SetReadOnly()

描述

启用 / 禁用只读模式。

原型

```
void MULTIEDIT_SetReadOnly(MULTIEDIT_HANDLE hObj, int OnOff);
```

参数	描述
hObj	Multiedit 小工具的句柄。
OnOff	（参见下表）

参数 OnOff 的允许值	
0	禁用只读模式。
1	启用只读模式。

其他信息

如果已设置只读模式，小工具将不会更改文本。此时，只能移动光标。

MULTIEDIT_SetText()

描述

设置将由小工具控制的文本。

原型

```
void MULTIEDIT_SetText(MULTIEDIT_HANDLE hObj, const char * s);
```

参数	描述
hObj	Multiedit 小工具的句柄。
s	将由 multiedit 小工具控制的文本指针。

其他信息

此函数会将指定文本复制到创建小工具时所分配的或由 MULTIEDIT_SetMaxSize() 分配的缓冲区。文本可由 MULTIEDIT_GetText() 检索。

MULTIEDIT_SetTextAlign()

描述

设置指定 MULTIEDIT 小工具的文本对齐方式。

原型

```
void MULTIEDIT_SetTextAlign(MULTIEDIT_HANDLE hObj, int Align);
```

参数	描述
hObj	Multiedit 小工具的句柄。
Align	（参见下表）

参数 Align 的允许值	
GUI_TA_LEFT	文本左对齐。
GUI_TA_RIGHT	文本右对齐。

MULTIEDIT_SetTextColor()

描述

设置文本颜色。

原型

```
void MULTIEDIT_SetTextColor(MULTIEDIT_HANDLE hObj,   unsigned int Index,
                            GUI_COLOR        Color);
```

参数	描述
hObj	Multiedit 小工具的句柄。
Index	（参见下表）
Color	要使用的文本颜色。

参数 Index 的允许值	
MULTIEDIT_CI_EDIT	编辑模式。
MULTIEDIT_CI_READONLY	只读模式。

MULTIEDIT_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

MULTIEDIT_SetWrapWord()

描述

启用字词换行模式。

原型

```
void MULTIEDIT_SetWrapWord(MULTIEDIT_HANDLE hObj);
```

参数	描述
hObj	Multiedit 小工具的句柄。

其他信息

如果已设置字词换行模式，则位于行尾的文本将在最后一个词的开始部分换行 （如果可能的话）。

MULTIEDIT_SetWrapNone()

描述

启用无换行模式。

原型

```
void MULTIEDIT_SetWrapNone(MULTIEDIT_HANDLE hObj);
```

参数	描述
hObj	Multiedit 小工具的句柄。

其他信息

“无换行”意味着将仅在新行处进行换行。如果文本的水平大小超出客户端区域的大小，则将滚动文本。

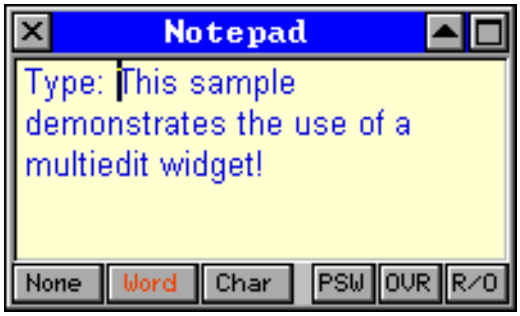
15.17.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_MultiEdit.c

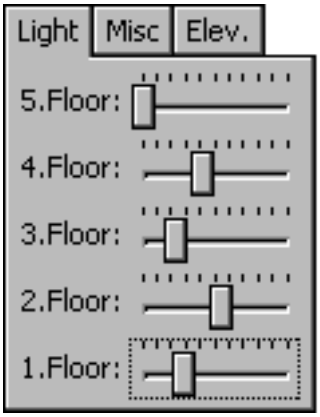
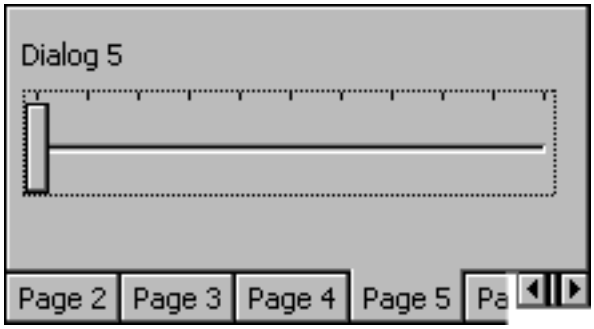
需要注意的是，其他几个示例可能也使用此小工具，这些示例可能有助于熟悉该小工具。

WIDGET_Multiedit.c 的屏幕截图：



15.18 MULTIPAGE：“多页”小工具

MULTIPAGE 类似于笔记本中的分隔卡或文件柜中的标签。通过使用 MULTIPAGE 小工具，应用程序可为窗口或对话框的相同区域定义多个页面。每个页面包含特定类型的信息或用户选择相应页面时应用程序会显示的一组小工具。如果要选择一个页面，则必须单击该页面的选项卡。如果无法显示所有选项卡，则 MULTIPAGE 小工具会自动在边缘处显示一个小滚动条以滚动页面。
“样本”文件夹中包含文件WIDGET_Multipage.c，它显示了如何创建并使用MULTIPAGE小工具。
下表显示 MULTIPAGE 小工具的外观：

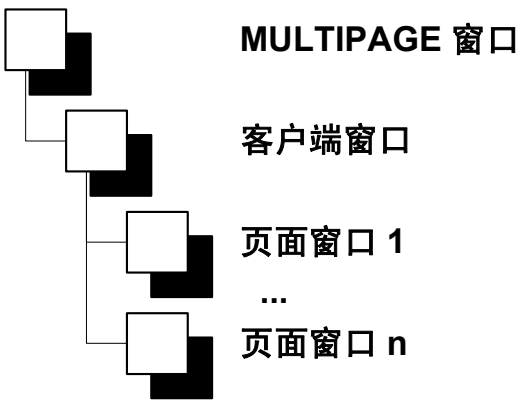
描述	MULTIPAGE 小工具
包含 3 个页面的 MULTIPAGE 小工具，左上对齐。	
包含 6 个页面的 MULTIPAGE 小工具，右下对齐。	

MULTIPAGE 小工具的结构

包含 n 个页面的 MULTIPAGE 小工具由 n+2 个窗口组成：

- 1 个 MULTIPAGE 窗口
- 1 个客户端窗口
- N 个页面窗口

页面窗口将被添加到小工具的客户端窗口。右图显示了小工具的结构。



15.18.1 配置选项

类型	宏	默认值	描述
N	MULTIPAGE_ALIGN_DEFAULT	MULTIPAGE_ALIGN_LEFT | MULTIPAGE_ALIGN_TOP	默认对齐方式。
N	MULTIPAGE_BKCOLOR0_DEFAULT	0xD0D0D0	禁用状态下的页面默认背景颜色。
N	MULTIPAGE_BKCOLOR1_DEFAULT	0xC0C0C0	启用状态下的页面默认背景颜色。
S	MULTIPAGE_FONT_DEFAULT	&GUI_Font13_1	小工具所使用的默认字体。
N	MULTIPAGE_TEXTCOLOR0_DEFAULT	0x808080	禁用状态下的页面默认文本颜色。
N	MULTIPAGE_TEXTCOLOR1_DEFAULT	0x000000	启用状态下的页面默认文本颜色。

15.18.2 通知代码

以下事件将作为 WM_NOTIFY_PARENT 消息的一部分从小工具发送至其父窗口：

消息	描述
WM_NOTIFICATION_CLICKED	已单击小工具。
WM_NOTIFICATION_RELEASED	已释放小工具。
WM_NOTIFICATION_MOVED_OUT	已单击小工具，并且指针已移出小工具，但没有释放。
WM_NOTIFICATION_VALUE_CHANGED	小工具的文本已更改。

15.18.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_PGUP	切换到下一页。
GUI_KEY_PGDOWN	切换到上一页。

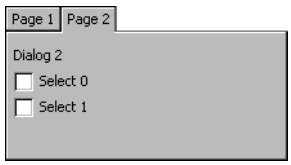
15.18.4 MULTIPAGE API

下表按字母顺序列出可用的 emWin MULTIPAGE 相关例程。例程的详细说明如下：

例程	描述
MULTIPAGE_AddPage()	将页面添加到 MULTIPAGE 小工具。
MULTIPAGE_CreateEx()	创建 MULTIPAGE 小工具。
MULTIPAGE_CreateIndirect()	从资源表条目创建 MULTIPAGE 小工具。
MULTIPAGE_CreateUser()	使用作为用户数据的额外字节来创建 MULTIPAGE 小工具。
MULTIPAGE_DeletePage()	将页面从 MULTIPAGE 小工具中删除。
MULTIPAGE_DisablePage()	禁用 MULTIPAGE 小工具中的页面。
MULTIPAGE_EnablePage()	启用 MULTIPAGE 小工具中的页面。
MULTIPAGE_GetDefaultAlign()	返回 MULTIPAGE 小工具的默认对齐方式。
MULTIPAGE_GetDefaultBkColor()	返回 MULTIPAGE 小工具的默认背景颜色。
MULTIPAGE_GetDefaultFont()	返回用于 MULTIPAGE 小工具的默认字体。
MULTIPAGE_GetDefaultTextColor()	返回用于 MULTIPAGE 小工具的默认文本颜色。
MULTIPAGE_GetSelection()	返回当前选择内容。
MULTIPAGE_GetUserData()	检索通过 MULTIPAGE_SetUserData() 设置的数据。
MULTIPAGE_GetWindow()	返回指定页面的窗口句柄。
MULTIPAGE_IsPageEnabled()	返回指定页面的启用状态。
MULTIPAGE_SelectPage()	选择指定页面。
MULTIPAGE_SetAlign()	设置选项卡的对齐方式。
MULTIPAGE_SetBkColor()	设置背景颜色。
MULTIPAGE_SetDefaultAlign()	设置新创建的 MULTIPAGE 小工具的默认对齐方式。
MULTIPAGE_SetDefaultBkColor()	设置新创建的 MULTIPAGE 小工具的默认背景颜色。

例程	描述
MULTIPAGE_SetDefaultFont()	设置新创建的 MULTIPAGE 小工具所使用的默认字体。
MULTIPAGE_SetDefaultTextColor()	设置新 MULTIPAGE 小工具所使用的默认文本颜色。
MULTIPAGE_SetFont()	选择小工具的字体。
MULTIPAGE_SetRotation()	设置小工具的旋转模式。
MULTIPAGE_SetText()	设置在 MULTIPAGE 小工具的选项卡中显示的文本。
MULTIPAGE_SetTextColor()	设置文本颜色。
MULTIPAGE_SetUserData()	设置 MULTIPAGE 小工具的额外数据。

MULTIPAGE_AddPage()

之前	之后
	

描述

将新页面添加到指定的 MULTIPAGE 小工具。

原型

```
void MULTIPAGE_AddPage(MULTIPAGE_Handle   hObj,   WM_HWIN hWin ,
                       const char       * pText);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。
hWin	要在指定页面中显示的窗口句柄。
pText	要在页面的选项卡中显示的文本指针。

其他信息

处理 WM_PAINT 消息时，建议添加到 MULTIPAGE 小工具的所有窗口控制 WM_PAINT 小工具的整个客户端区域。

MULTIPAGE_CreateEx()

描述

在指定位置创建指定大小的 MULTIPAGE 小工具。

原型

```
MULTIPAGE_Handle MULTIPAGE_CreateEx(int     x0,      int y0,
                                    int     xsize,   int ysize,
                                    WM_HWIN hParent, int WinFlags,
                                    int     ExFlags, int Id);
```

参数	描述
x0	小工具的 X 轴位置 （在父坐标中）。
y0	小工具的 Y 轴位置 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新创建的小工具将作为桌面 （顶层窗口）的子窗口。

参数	描述
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅“WM_CreateWindow()”（第 309 页））。
ExFlags	未使用，保留供日后使用。
Id	小工具的窗口 ID。

返回值

新小工具的句柄。

其他信息

选项卡的大小取决于 MULTIPAGE 小工具所使用的字体的大小。

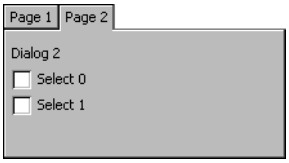
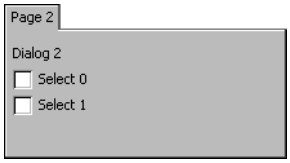
MULTIPAGE_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。

MULTIPAGE_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细说明，可参阅函数 MULTIPAGE_CreateEx()。

MULTIPAGE_DeletePage()

之前	之后
	

描述

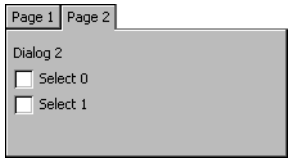
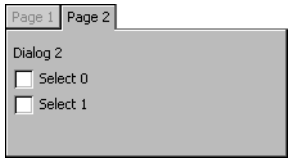
将页面从 MULTIPAGE 小工具中删除并有选择性地删除窗口。

原型

```
void MULTIPAGE_DeletePage(MULTIPAGE_Handle hObj,    unsigned Index,
                          int               Delete);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。
Index	要从 MULTIPAGE 小工具中删除的页面索引（以零为基准）。
Delete	如果 >0，则附加到页面的窗口将被删除。

MULTIPAGE_DisablePage()

之前	之后
	

描述

禁用 MULTIPAGE 小工具中的页面。

原型

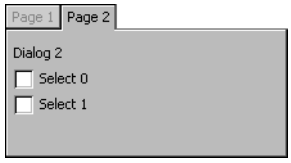
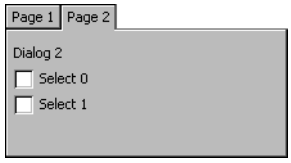
```
void MULTIPAGE_DisablePage(MULTIPAGE_Handle hObj, unsigned Index);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。
Index	要禁用的页面的索引 （以零为基准）。

其他信息

无法通过单击页面的选项卡来选择窗口已禁用的页面。 MULTIEDIT 页面的默认状态为 “启用”。

MULTIPAGE_EnablePage()

之前	之后
	

描述

启用 MULTIPAGE 小工具的页面。

原型

```
void MULTIPAGE_EnablePage(MULTIPAGE_Handle hObj, unsigned Index);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。

其他信息

MULTIEDIT 页面的默认状态为 “启用”。

MULTIPAGE_GetDefaultAlign()

描述

返回新创建的 MULTIPAGE 小工具的默认选项卡对齐方式。

原型

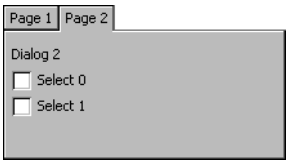
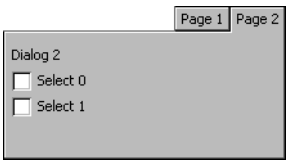
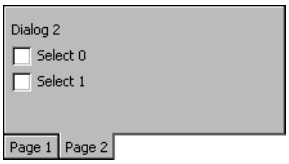
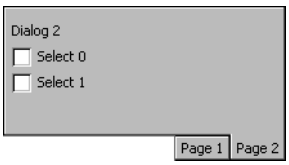
```
unsigned MULTIPAGE_GetDefaultAlign(void);
```

返回值

新创建的 MULTIPAGE 小工具的默认选项卡对齐方式。

其他信息

下表显示此函数返回的对齐值：

对齐	MULTIPAGE 小工具的外观
MULTIPAGE_ALIGN_LEFT | MULTIPAGE_ALIGN_TOP	
MULTIPAGE_ALIGN_RIGHT | MULTIPAGE_ALIGN_TOP	
MULTIPAGE_ALIGN_LEFT | MULTIPAGE_ALIGN_BOTTOM	
MULTIPAGE_ALIGN_RIGHT | MULTIPAGE_ALIGN_BOTTOM	

MULTIPAGE_GetDefaultBkColor()

描述

返回新创建的 MULTIPAGE 小工具的默认背景颜色。

原型

```
GUI_COLOR MULTIPAGE_GetDefaultBkColor(unsigned Index);
```

参数	描述
Index	参见下表。

参数 Index 的允许值	
0	返回禁用状态下的页面默认背景颜色。
1	返回启用状态下的页面默认背景颜色。

返回值

新创建的 MULTIPAGE 小工具的默认背景颜色。

MULTIPAGE_GetDefaultFont()

描述

返回显示新创建的 MULTIPAGE 小工具的选项卡中文本所使用的字体指针。

原型

```
const GUI_FONT * MULTIPAGE_GetDefaultFont(void);
```

返回值

显示新创建的 MULTIPAGE 小工具的选项卡中文本所使用的字体指针。

MULTIPAGE_GetDefaultTextColor()

描述

返回显示新创建的 MULTIPAGE 小工具选项卡中文本所使用的默认文本颜色。

原型

```
GUI_COLOR MULTIPAGE_GetDefaultTextColor(unsigned Index);
```

参数	描述
Index	参见下表。

参数 Index 的允许值	
0	返回禁用状态下的页面默认文本颜色。
1	返回启用状态下的页面默认文本颜色。

返回值

显示新创建的 MULTIPAGE 小工具选项卡中文本所使用的默认文本颜色。

MULTIPAGE_GetSelection()

描述

返回 MULTIPAGE 小工具当前所选页面的索引 （以零为基准）。

原型

```
int MULTIPAGE_GetSelection(MULTIPAGE_Handle hObj);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。

返回值

MULTIPAGE 小工具当前所选页面的索引 （以零为基准）。

MULTIPAGE_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

MULTIPAGE_GetWindow()

描述

返回在指定页面中显示的窗口的句柄。

原型

```
WM_HWIN MULTIPAGE_GetWindow(MULTIPAGE_Handle hObj, unsigned Index);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。
Index	页面的索引（以零为基准）。

返回值

在指定页面中显示的窗口的句柄。

MULTIPAGE_IsPageEnabled()

描述

返回 MULTIEDIT 小工具的指定页面的启用状态。

原型

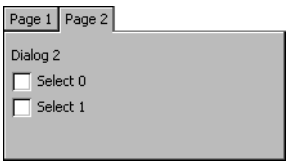
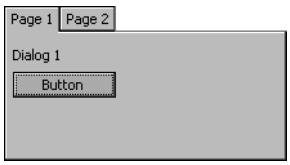
```
int MULTIPAGE_IsPageEnabled (MULTIPAGE_Handle hObj, unsigned Index);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。
Index	请求的页面的索引（以零为基准）。

返回值

如果指定页面已启用，则返回值为 1；否则，返回值为 0。

MULTIPAGE_SelectPage()

之前	之后
	

描述

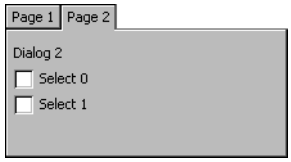
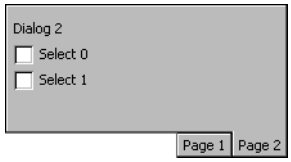
设置 MULTIPAGE 小工具的当前所选页面。

原型

```
void MULTIPAGE_SelectPage(MULTIPAGE_Handle hObj, unsigned Index);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。
Index	要选择的页面的索引（以零为基准）。

MULTIPAGE_SetAlign()

之前	之后
	

描述

设置指定 MULTIPAGE 小工具的选项卡对齐方式。

原型

```
void MULTIPAGE_SetAlign(MULTIPAGE_Handle hObj, unsigned Align);
```

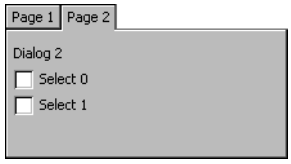
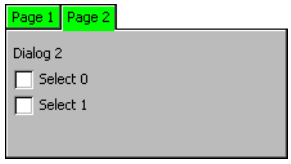
参数	描述
hObj	MULTIPAGE 小工具的句柄。
Align	参见下表。

参数 Index 的允许值 （水平标记和垂直标记可以通过 “OR” 操作进行组合）	
MULTIPAGE_ALIGN_BOTTOM	将选项卡向右侧对齐。
MULTIPAGE_ALIGN_LEFT	将选项卡向左侧对齐。
MULTIPAGE_ALIGN_RIGHT	将选项卡向小工具的顶部对齐。
MULTIPAGE_ALIGN_TOP	将选项卡向小工具的底部对齐。

其他信息

更多详细信息，请参阅 “MULTIPAGE_GetDefaultAlign()”（第 590 页）。

MULTIPAGE_SetBkColor()

之前	之后
	

描述

设置指定 MULTIPAGE 小工具的背景颜色。

原型

```
void MULTIPAGE_SetBkColor(MULTIPAGE_Handle hObj,   GUI_COLOR Color,
                          unsigned         Index);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。
Color	要使用的颜色。
Index	参见下表。

参数 Index 的允许值	
MULTIPAGE_CI_DISABLED	设置已禁用的页面的默认文本颜色。
MULTIPAGE_CI_ENABLED	设置已启用的页面的默认文本颜色。

其他信息

该函数仅设置 MULTIPAGE 小工具的背景颜色。添加到小工具的子窗口不受影响。这就意味着，如果整个客户端区域由添加到小工具的窗口绘制，则仅选项卡的背景颜色会更改。

MULTIPAGE_SetDefaultAlign()

描述

设置新创建的 MULTIPAGE 小工具的默认选项卡对齐方式。

原型

```
void MULTIPAGE_SetDefaultAlign(unsigned Align);
```

参数	描述
Align	新创建的 MULTIPAGE 小工具所使用的选项卡对齐方式。

其他信息

有关选项卡对齐方式的更多信息，“MULTIPAGE_GetDefaultAlign()”（第 590 页）和“MULTIPAGE_SetAlign()”（第 594 页）。

MULTIPAGE_SetDefaultBkColor()

描述

设置新创建的 MULTIPAGE 小工具所使用的默认背景颜色。

原型

```
void MULTIPAGE_SetDefaultBkColor(GUI_COLOR Color, unsigned Index);
```

参数	描述
Color	要使用的颜色。
Index	参见下表。

参数 Index 的允许值	
0	设置禁用状态下的页面默认背景颜色。
1	设置启用状态下的页面默认背景颜色。

MULTIPAGE_SetDefaultFont()

描述

设置显示新创建的 MULTIPAGE 小工具的选项卡中文本所使用的默认字体。

原型

```
void MULTIPAGE_SetDefaultFont(const GUI_FONT * pFont);
```

参数	描述
pFont	要使用的 GUI_FONT 结构的指针。

其他信息

选项卡的水平和垂直大小取决于所使用的字体的大小。

MULTIPAGE_SetDefaultTextColor()

描述

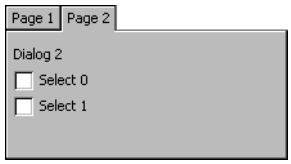
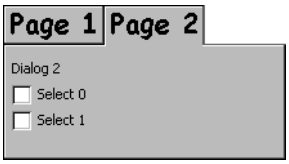
返回显示新创建的 MULTIPAGE 小工具的选项卡中文本所使用的默认文本颜色。

原型

```
void MULTIPAGE_SetDefaultTextColor(GUI_COLOR Color, unsigned Index);
```

参数	描述
Color	要使用的颜色。
Index	参见下表。

MULTIPAGE_SetFont()

之前	之后
	

描述

设置显示指定 MULTIPAGE 小工具的选项卡中文本所使用的字体。

原型

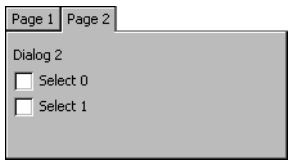
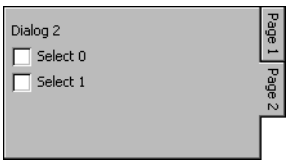
```
void MULTIPAGE_SetFont(MULTIPAGE_Handle hObj, const GUI_FONT * pFont);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。
pFont	显示选项卡中文本的 GUI_FONT 结构所使用的指针。

其他信息

选项卡的垂直和水平大小取决于所使用的字体和选项卡中显示的文本大小。

MULTIPAGE_SetRotation()

之前	之后
	

描述

设置指定小工具的旋转模式。

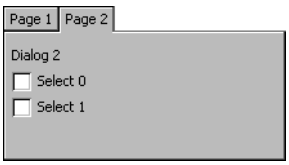
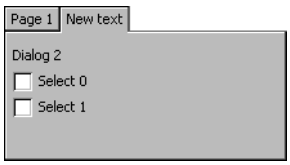
原型

```
void MULTIPAGE_SetRotation(MULTIPAGE_Handle hObj, unsigned Rotation);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。
Rotation	旋转模式 （见下表）

参数 Index 的允许值	
MULTIPAGE_CF_ROTATE_CW	垂直排列选项卡，并将选项卡文本顺时针旋转 90 度。
0	默认的水平模式。

MULTIPAGE_SetText()

之前	之后
	

描述

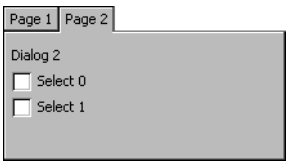
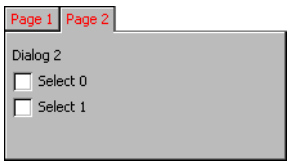
设置在指定页面的选项卡中所显示的文本。

原型

```
void MULTIPAGE_SetText(MULTIPAGE_Handle hObj,   const char * pText,
                       unsigned         Index);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。
pText	要显示的文本的指针。
Index	以零为基准的页面索引。

MULTIPAGE_SetTextColor()

之前	之后
	

描述

设置显示 MULTIPAGE 小工具的选项卡中文本所使用的颜色。

原型

```
void MULTIPAGE_SetTextColor(MULTIPAGE_Handle hObj,   GUI_COLOR Color,
                            unsigned         Index);
```

参数	描述
hObj	MULTIPAGE 小工具的句柄。
Color	要使用的颜色。
Index	参见下表。

参数 Index 的允许值	
0	设置禁用状态下的页面文本颜色。
1	设置启用状态下的页面文本颜色。

MULTIPAGE_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

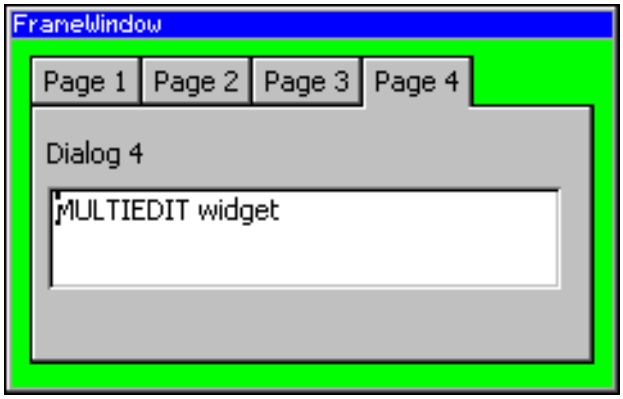
15.18.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_Multipage.c

需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

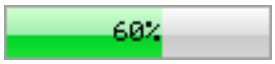
WIDGET_Multipage.c 的截屏：



15.19 PROGBAR：进度条小工具

进度条通常在应用程序中用于实现虚拟化；例如，油罐液位指示器或油压指示器。您可在本章的开始部分及本节的末尾找到作为示例的屏幕截图。所有 PROGBAR 相关例程包含于文件 PROGBAR*.c 和 PROGBAR.h 中。所有识别符均为带有前缀的 PROGBAR。

皮肤设置 ...



... 可用于该小工具。上述屏幕截图显示使用默认皮肤的小工具。有关详细信息，请参阅“皮肤设置”一章。

15.19.1 配置选项

类型	宏	默认值	描述
S	PROGBAR_DEFAULT_FONT	GUI_DEFAULT_FONT	所使用的字体。
N	PROGBAR_DEFAULT_BARCOLOR0	0x555555（深灰色）	左侧进度条的颜色。
N	PROGBAR_DEFAULT_BARCOLOR1	0xAAAAAA（浅灰色）	右侧进度条的颜色。
N	PROGBAR_DEFAULT_TEXTCOLOR0	0xFFFFFF	文本颜色，左侧进度条。
N	PROGBAR_DEFAULT_TEXTCOLOR1	0x000000	文本颜色，右侧进度条。

15.19.2 键盘反应

该小工具无法获取输入焦点，并且对键盘输入无任何反应。

15.19.3 PROGBAR API

下表按字母顺序列出可用的 emWin PROGBAR 相关例程。例程的详细说明如下：

例程	描述
PROGBAR_Create()	创建 PROGBAR 小工具。（弃用）
PROGBAR_CreateAsChild()	创建 PROGBAR 小工具，将其作为子窗口。（弃用）
PROGBAR_CreateEx()	创建 PROGBAR 小工具。
PROGBAR_CreateIndirect()	从资源表条目创建 PROGBAR 小工具。
PROGBAR_CreateUser()	使用作为用户数据的额外字节来创建 PROGBAR 小工具。
PROGBAR_GetUserData()	检索通过 PROGBAR_SetUserData() 设置的数据。
PROGBAR_SetBarColor()	设置进度条的颜色。
PROGBAR_SetFont()	选择文本的字体。
PROGBAR_SetMinMax()	设置进度条所使用的最小值和最大值。
PROGBAR_SetText()	设置进度条图形的文本 （可选）。
PROGBAR_SetTextAlign()	设置文本对齐方式 （默认为居中）。
PROGBAR_SetTextColor()	设置文本的颜色。
PROGBAR_SetTextPos()	设置文本位置 （默认为 0,0）。
PROGBAR_SetUserData()	设置 PROGBAR 小工具的额外数据。
PROGBAR_SetValue()	设置条图形的值 （如果未分配任何文本，则设置百分比）。

PROGBAR_Create()

（弃用，应转而使用 PROGBAR_CreateEx()）

描述

在指定位置创建指定大小的 PROGBAR 小工具。

原型

```
PROGBAR_Handle PROGBAR_Create(int x0,     int y0,
                              int xsize, int ysize, int Flags);
```

参数	描述
x0	进度条最左边的像素 （在父坐标中）。
y0	进度条最顶端的像素 （在父坐标中）。
xsize	进度条的水平大小 （以像素为单位）。
ysize	进度条的垂直大小 （以像素为单位）。
Flags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。

返回值

所创建的 PROGBAR 小工具的句柄；如果函数执行失败，则返回值为 0。

PROGBAR_CreateAsChild()

（弃用，应转而使用 PROGBAR_CreateEx）

描述

创建 PROGBAR 小工具，将其作为子窗口。

原型

```
PROGBAR_Handle PROGBAR_CreateAsChild(int     x0,      int y0,
                                     int     xsize,   int ysize,
                                     WM_HWIN hParent, int Id,
                                     int     Flags);
```

参数	描述
x0	相对于父窗口的进度条的 X 轴位置
y0	相对于父窗口的进度条的 Y 轴位置
xsize	进度条的水平大小 （以像素为单位）。
ysize	进度条的垂直大小 （以像素为单位）。
hParent	父窗口的句柄。
Id	将返回的 ID。
Flags	窗口创建标识 （请参阅 PROGBAR_Create()）。

返回值

所创建的 PROGBAR 小工具的句柄；如果函数执行失败，则返回值为 0。

PROGBAR_CreateEx()

描述

在指定位置创建指定大小的 PROGBAR 小工具。

原型

```
PROGBAR_Handle PROGBAR_CreateEx(int     x0,      int y0,
                                int     xsize,   int ysize,
                                WM_HWIN hParent, int WinFlags,
                                int     ExFlags, int Id);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。

参数	描述
hParent	父窗口的句柄。如果为 0，则新创建的 PROGBAR 小工具将作为桌面 （顶层窗口）的子窗口。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW（有关可用参数值的列表，请参阅“窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	（参见下表）
Id	小工具的窗口 ID。

参数 ExFlags 的允许值	
PROGBAR_CF_VERTICAL	将创建垂直进度条。
PROGBAR_CF_HORIZONTAL	将创建水平进度条。

返回值

所创建的 PROGBAR 小工具的句柄；如果函数执行失败，则返回值为 0。

PROGBAR_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。资源的元素 Flags 和 Para 将省略，因为未使用参数。

PROGBAR_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细说明，可参阅函数 PROGBAR_CreateEx()。

PROGBAR_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

PROGBAR_SetBarColor()

描述

设置进度条的颜色。

原型

```
void PROGBAR_SetBarColor(PROGBAR_Handle hObj,   unsigned int Index,
                         GUI_COLOR      Color);
```

参数	描述
hObj	进度条的句柄。
Index	参见下表。不允许使用其他值。
Color	要设置的颜色 （24 位 RGB 值）。

参数 Index 的允许值	
0	进度条的左侧部分。
1	进度条的右侧部分。

PROGBAR_SetFont()

描述

选择在进度条内部显示文本所使用的字体。

原型

```
void PROGBAR_SetFont(PROGBAR_Handle hObj, const GUI_FONT* pFont);
```

参数	描述
hObj	进度条的句柄。
pFont	字体的指针。

其他信息

如果不调用此函数，则将使用进度条的默认字体 （GUI 默认字体）。不过，在 GUIConf.h 文件中可更改进度条默认字体。
仅需输入 # 即可如下定义默认字体 （示例）：

```
#define PROGBAR_DEFAULT_FONT &GUI_Font13_ASCII
```

PROGBAR_SetMinMax()

描述

设置进度条所使用的最小值和最大值。

原型

```
void PROGBAR_SetMinMax(PROGBAR_Handle hObj, int Min, int Max);
```

参数	描述
hObj	进度条的句柄。
Min	最小值 范围：-16383< Min <= 16383。
Max	最大值 范围：-16383< Max <= 16383。

其他信息

如果不调用此函数，则将使用 Min = 0 且 Max = 100 的默认值。

PROGBAR_SetText()

描述

设置在进度条内部所显示的文本。

原型

```
void PROGBAR_SetText(PROGBAR_Handle hObj, const char* s);
```

参数	描述
hObj	进度条的句柄。
s	要显示的文本。允许使用空指针；在这种情况下，将显示百分比值。

其他信息

如果不调用此函数，则将默认显示百分比值。如果根本不想显示任何文本，则应设置空字符串。

PROGBAR_SetTextAlign()

描述

设置文本对齐方式。

原型

```
void PROGBAR_SetTextAlign(PROGBAR_Handle hObj, int Align);
```

参数	描述
hObj	进度条的句柄。
Align	文本的水平对齐属性 （见下表）。

参数 Align 的允许值	
GUI_TA_HCENTER	居中标题 （默认）。
GUI_TA_LEFT	标题显示到左侧。
GUI_TA_RIGHT	标题显示到右侧。

其他信息

如果不调用此函数，则默认行为将是居中显示文本。

PROGBAR_SetTextColor()

描述

设置进度条的文本颜色。

原型

```
void PROGBAR_SetTextColor(PROGBAR_Handle hObj,   unsigned int Index,
                          GUI_COLOR      Color);
```

参数	描述
hObj	进度条的句柄。
Index	参见下表。不允许使用其他值。
Color	要设置的颜色 （24 位 RGB 值）。

参数 Index 的允许值	
0	文本的左侧部分。
1	文本的右侧部分。

PROGBAR_SetTextPos()

描述

设置文本位置 （以像素为单位）。

原型

```
void PROGBAR_SetTextPos(PROGBAR_Handle hObj, int XOff, int YOff);
```

参数	描述
hObj	进度条的句柄。
XOff	在水平方向上移动文本所使用的像素数。 正数将使文本向右移动。
YOff	在垂直方向上移动文本所使用的像素数。 正数将使文本向下移动。

其他信息

这些数值会使文本在小工具内移动指定的像素数。通常，默认值 (0,0) 应当已经足够。

PROGBAR_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

PROGBAR_SetValue()

描述

设置进度条的值。

原型

```
void PROGBAR_SetValue(PROGBAR_Handle hObj, int v);
```

参数	描述
hObj	进度条的句柄。
v	要设置的值。

其他信息

将依据最大值 / 最小值计算条指示器。如果自动显示百分比，则也将如下使用最小值 / 最大值计算百分比：
p = 100% * (v-Min)/(Max-Min)
创建小工具之后的默认值为 0。

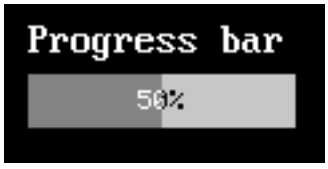
15.19.4 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_SimpleProgbar.c
- WIDGET_Progbar.c

需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_SimpleProgbar.c 的屏幕截图：



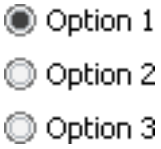
WIDGET_Progbar.c 的屏幕截图：

15.20 RADIO：单选按钮小工具

如同复选框一样，单选按钮也可用来选择选项。打开或选择单选按钮时，将出现圆点。与复选框的差别是，用户一次只能选择一个单选按钮。选择一个按钮时，小工具中的其他按钮将关闭 （如右图所示）。一个单选按钮小工具可能包含任意数量的按钮 ，这些按钮始终处于垂直排列状态。
所有与 RADIO 相关的例程均在 RADIO*.c、RADIO.h 文件中。所有识别符均为带有前缀的 RADIO。下表显示的是 RADIO 按钮的默认外观：

	选定	未选定
启用	Radio button	Radio button
已禁用	Radio button	Radio button

皮肤设置 ...



... 可用于该小工具。上述屏幕截图显示使用默认皮肤的小工具。有关详细信息，请参阅“皮肤设置”一章。

15.20.1 配置选项

类型	宏	默认值	描述
S	RADIO_IMAGE0_DEFAULT	（参见上表）	显示已禁用单选按钮所使用的默认外部图像。
S	RADIO_IMAGE1_DEFAULT	（参见上表）	显示已启用单选按钮所使用的默认外部图像。
S	RADIO_IMAGE_CHECK_DEFAULT	（参见上表）	标记所选项目所使用的默认内部图像。
N	RADIO_FONT_DEFAULT	&GUI_Font13_1	显示单选按钮文本所使用的默认字体。
N	RADIO_DEFAULT_TEXT_COLOR	GUI_BLACK	单选按钮文本的默认文本颜色。
N	RADIO_DEFAULT_BKCOLOR	0xC0C0C0	未使用透明状效果时，单选按钮的默认背景颜色。
N	RADIO_FOCUSCOLOR_DEFAULT	GUI_BLACK	显示聚焦框所使用的默认颜色。

15.20.2 通知代码

以下事件将作为 WM_NOTIFY_PARENT 消息的一部分从单选按钮小工具发送至其父窗口：

消息	描述
WM_NOTIFICATION_CLICKED	已单击单选按钮。
WM_NOTIFICATION_RELEASED	已释放单选按钮。
WM_NOTIFICATION_MOVED_OUT	已单击单选按钮，并且指针已移出单选按钮，但没有释放。
WM_NOTIFICATION_VALUE_CHANGED	单选按钮小工具的值 （选择内容）已更改。

15.20.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_RIGHT	选定范围增加 1。
GUI_KEY_DOWN	选定范围增加 1。
GUI_KEY_LEFT	选定范围减小 1。
GUI_KEY_UP	选定范围减小 1。

15.20.4 RADIO API

下表按字母顺序列出可用的 emWin RADIO 相关例程。这些例程的详细说明如下。

例程	描述
RADIO_Create()	创建 RADIO 小工具。（弃用）
RADIO_CreateEx()	创建 RADIO 小工具。
RADIO_CreateIndirect()	从资源表条目创建 RADIO 小工具。
RADIO_CreateUser()	使用作为用户数据的额外字节来创建 RADIO 小工具。
RADIO_Dec()	将按钮选定范围的值减小 1。
RADIO_GetDefaultFont()	返回显示新单选按钮文本所使用的默认字体。
RADIO_GetDefaultTextColor()	返回显示新单选按钮文本所使用的默认文本颜色。
RADIO_GetText()	返回单选按钮项目文本。
RADIO_GetUserData()	检索带 RADIO_SetUserData() 的数据集。
RADIO_GetValue()	返回当前按钮选定内容。
RADIO_Inc()	将按钮选定范围的值增加 1。
RADIO_SetBkColor()	设置单选按钮的背景颜色。
RADIO_SetDefaultFocusColor()	设置新单选按钮的默认聚焦框颜色。
RADIO_SetDefaultFont()	设置显示新单选按钮文本所使用的默认字体。
RADIO_SetDefaultImage()	设置新单选按钮将使用的图像。
RADIO_SetDefaultTextColor()	设置显示新单选按钮文本所使用的默认文本颜色。
RADIO_SetFocusColor()	设置聚焦框的颜色。
RADIO_SetFont()	设置显示单选按钮文本所使用的字体。
RADIO_SetGroupId()	设置指定单选框小工具的群组 ID。
RADIO_SetImage()	设置显示单选按钮所使用的图像。
RADIO_SetText()	设置文本。
RADIO_SetTextColor()	设置显示单选按钮文本所使用的文本颜色。
RADIO_SetUserData()	设置 RADIO 小工具的额外数据。
RADIO_SetValue()	设置按钮选定内容。

RADIO_Create()

（弃用，应转而使用 RADIO_CreateEx()）

描述

在指定位置创建指定大小的 RADIO 小工具。

原型

```
RADIO_Handle RADIO_Create(int     x0,      int       y0,
                          int     xsize,   int       ysize,
                          WM_HWIN hParent, int       Id,
                          int     Flags,   unsigned  Para);
```

参数	描述
x0	单选按钮小工具最左侧的像素 （在父坐标中）。
y0	单选按钮小工具最顶端的像素 （在父坐标中）。
xsize	单选按钮小工具的水平尺寸 （以像素为单位）。
ysize	单选按钮小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。
Id	将返回的 ID。
Flags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW（有关可用参数值的列表，请参阅“窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
Para	群组中按钮的数量。

返回值

所创建的 RADIO 小工具的句柄；如果函数执行失败，则返回值为 0。

RADIO_CreateEx()

描述

在指定位置创建指定大小的 RADIO 小工具。

原型

```
RADIO_Handle RADIO_CreateEx(int     x0,       int y0,
                            int     xsize,    int ysize,
                            WM_HWIN hParent,  int WinFlags,
                            int     ExFlags,  int Id,
                            int     NumItems, int Spacing);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新创建的 RADIO 小工具将作为桌面 （顶层窗口）的子窗口。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW（有关可用参数值的列表，请参阅“窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	未使用，保留供日后使用。
Id	小工具的窗口 ID。
NumItems	单选框小工具的项目数。（默认值为 2）
Spacing	单选框小工具每个项目所使用的垂直像素数。

返回值

所创建的 RADIO 小工具的句柄；如果函数执行失败，则返回值为 0。

其他信息

如果创建单选框小工具，则需确保指定 ysize 足以用来显示所有项目，该值至少应为 NumItems * Spacing。如果 NumItems 的给定值 <= 0，那么使用默认值 2。

RADIO_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。未使用作为参数传递的资源元素 Flags。下表包含资源元素 Para 的使用：

位	描述
0 - 7	单选框小工具的项目数。如果为 0，则使用默认项目值 2。
8 - 15	用于每个项目的垂直像素数。如果为 0，则使用默认图像的高度。
16 - 23	未使用，保留供日后使用。
24 - 31	未使用，保留供日后使用。

RADIO_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细说明，可参阅函数 RADIO_CreateEx()。

RADIO_Dec()

之前	之后
	

描述

将选定范围的值减小 1。

原型

```
void RADIO_Dec(RADIO_Handle hObj);
```

参数	描述
hObj	单选按钮小工具的句柄。

其他信息

需要注意的是，按钮编号始终从顶部以 0 值开始；因此，降低选定范围实际上会将选定范围上移一个按钮。

RADIO_GetDefaultFont()

描述

返回显示新单选按钮旁边可选文本所使用的默认字体。

原型

```
const GUI_FONT * RADIO_GetDefaultFont(void);
```

返回值

显示单选按钮旁边可选文本所使用的默认字体。

其他信息

有关如何将文本添加至单选框小工具的信息，请参阅 “RADIO_SetText()”（第 614 页）。

RADIO_GetDefaultTextColor()

描述

返回显示新单选按钮旁边可选文本所使用的默认文本颜色。

原型

```
GUI_COLOR RADIO_GetDefaultTextColor (void);
```

返回值

显示新单选按钮旁边可选文本所使用的默认文本颜色。

其他信息

有关如何将文本添加至单选框小工具的信息，请参阅 “RADIO_SetText()”（第 614 页）。

RADIO_GetText()

描述

返回指定单选按钮的可选文本。

原型

```
int RADIO_GetText(RADIO_Handle   hObj,    unsigned Index,
                  char         * pBuffer, int      MaxLen);
```

参数	描述
hObj	小工具的句柄。
Index	所需项目的索引。
pBuffer	复制文本所在缓冲区的指针。
MaxLen	缓冲区大小 （以字节为单位）。

返回值

复制到缓冲区的文本长度。

其他信息

如果单选按钮的目标项不包含任何文本，则函数将返回 0，且缓冲区保持不变。

RADIO_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

RADIO_GetValue()

描述

返回当前按钮选定内容。

原型

```
void RADIO_GetValue(RADIO_Handle hObj);
```

参数	描述
hObj	单选按钮小工具的句柄。

返回值

当前所选按钮的值。如果未选定任何按钮 （在使用单选按钮群组的情况下），则返回值为 -1。

其他信息

有关如何使用单选按钮群组的信息，请参阅 “RADIO_SetGroupID()”（第 613 页）。

RADIO_Inc()

之前	之后
	

描述

将选定范围的值增加 1。

原型

```
void RADIO_Inc(RADIO_Handle hObj);
```

参数	描述
hObj	单选按钮小工具的句柄。

其他信息

需要注意的是，按钮编号始终从顶部以 0 值开始；因此，增加选定范围实际上会将选定范围下移一个按钮。

RADIO_SetBkColor()

之前	之后
	

描述

设置单选框小工具的背景颜色。

原型

```
void RADIO_SetBkColor(RADIO_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	单选按钮小工具的句柄。
Color	要使用的背景颜色。 （范围 0x000000 和 0xFFFFFF 或有效颜色定义） GUI_INVALID_COLOR 可使背景透明

其他信息

该小工具的背景可以填充任何可用颜色或透明色。如果指定一个有效的 RGB 颜色，则背景将填充该颜色，否则背景 （通常是父窗口的内容）是可见的。如果背景是透明的，则该小工具被视为透明窗口，否则为不透明窗口。需要注意的是，使用背景颜色可使显示更加有效 （速度更快）。

RADIO_SetDefaultFocusColor()

描述

设置新单选按钮的默认聚焦框颜色。

原型

```
GUI_COLOR RADIO_SetDefaultFocusColor(GUI_COLOR Color);
```

参数	描述
Color	新单选按钮所使用的默认颜色。

返回值

先前的默认聚焦框颜色。

其他信息

更多详细信息，请参阅 “RADIO_SetFocusColor()”（第 612 页）。

RADIO_SetDefaultFont()

描述

设置显示新单选按钮旁边可选文本所使用的默认字体。

原型

```
void RADIO_SetDefaultFont(const GUI_FONT * pFont);
```

参数	描述
pFont	显示新单选框小工具文本所使用的 GUI_FONT 结构指针。

其他信息

有关如何将文本添加至单选框小工具的信息，请参阅 “RADIO_SetText()”（第 614 页）。

RADIO_SetDefaultImage()

描述

设置描述新单选按钮所使用的图像。

原型

```
void RADIO_SetDefaultImage(const GUI_BITMAP * pBitmap, unsigned int Index);
```

参数	描述
pBitmap	位图的指针。
Index	（参见下表）

参数 Index 的允许值	
RADIO_BI_INACTIV	显示已禁用单选按钮所使用的外部图像。
RADIO_BI_ACTIV	显示已启用单选按钮所使用的外部图像。
RADIO_BI_CHECK	标记所选项目所使用的内部图像。

其他信息

两个图像都用于显示单选按钮。一个图像用来绘制显示未选定单选按钮所使用的外框。根据当前状态，该图像将是 RADIO_BI_ACTIV （默认）或 RADIO_BI_ACTIV 所引用的位图。第二个图像（由 RADIO_BI_CHECK 引用）则用于标记当前选定的按钮。

RADIO_SetDefaultTextColor()

描述

设置显示新单选按钮旁边可选文本所使用的默认文本颜色。

原型

```
void RADIO_SetDefaultTextColor(GUI_COLOR TextColor);
```

参数	描述
TextColor	要使用的新颜色。

其他信息

有关如何将文本添加至单选框小工具的信息，请参阅 “RADIO_SetText()”（第 614 页）。

RADIO_SetFocusColor()

之前	之后
	

描述

设置显示单选按钮聚焦框所使用的颜色。

原型

```
GUI_COLOR RADIO_SetFocusColor(RADIO_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄。
Color	聚焦框所使用的颜色。

返回值

先前的聚焦框颜色。

其他信息

仅当此小工具具有输入焦点时，聚焦框才可见。

RADIO_SetFont()

之前	之后
	

描述

设置显示单选按钮旁边可选文本所使用的字体。

原型

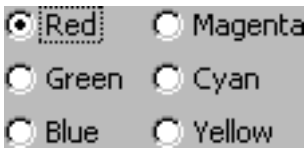
```
void RADIO_SetFont(RADIO_Handle hObj, const GUI_FONT * pFont);
```

参数	描述
hObj	单选按钮小工具的句柄。
pFont	显示文本所使用的 GUI_FONT 结构的指针。

其他信息

有关如何将文本添加至单选框小工具的信息，请参阅 “RADIO_SetText()”（第 614 页）。

RADIO_SetGroupID()

之前	之后
	

描述

设置单选框小工具的群组 ID。

原型

```
void RADIO_SetGroupID(RADIO_Handle hObj, U8 GroupID);
```

参数	描述
hObj	单选按钮小工具的句柄。
GroupID	单选按钮群组的 ID。必须在 1 和 255 之间。如果值为 0，则不能将单选框小工具分配给单选按钮群组。

其他信息

此命令可用于创建单选按钮群组。一个群组的作用与一个单选按钮的作用相同。例如，这样就可以并排创建 2 个 RADIO 小工具 （带 3 个按钮），并将它们创建为一个群组。

示例

以下示例说明该如何创建含有 2 个 RADIO 小工具的群组，如功能说明开始部分的屏幕截图所示：

```
hRadio_0 = RADIO_CreateEx(10, 10, 60, 0, WM_HBKWIN, WM_CF_SHOW, 0, 1234, 3, 20);
RADIO_SetText(hRadio_0, "Red", 0);
RADIO_SetText(hRadio_0, "Green", 1);
RADIO_SetText(hRadio_0, "Blue", 2);
hRadio_1 = RADIO_CreateEx(65, 10, 60, 0, WM_HBKWIN, WM_CF_SHOW, 0, 1234, 3, 20);
RADIO_SetText(hRadio_1, "Magenta", 0);
RADIO_SetText(hRadio_1, "Cyan", 1);
RADIO_SetText(hRadio_1, "Yellow", 2);
RADIO_SetGroupID(hRadio_0, 1);
RADIO_SetGroupID(hRadio_1, 1);
```

RADIO_SetImage()

描述

设置绘制单选按钮所使用的图像。

原型

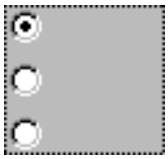
```
void RADIO_SetImage(RADIO_Handle hObj,   const GUI_BITMAP * pBitmap,
                    unsigned int Index);
```

参数	描述
hObj	单选按钮小工具的句柄。
pBitmap	位图的指针。
Index	（参见 RADIO_SetDefaultImage 下方所示表格）

其他信息

（参见 RADIO_SetDefaultImage）。

RADIO_SetText()

之前	之后
	

描述

设置单选按钮旁边所示的可选文本。

原型

```
void RADIO_SetText(RADIO_Handle hObj, const char * pText, unsigned Index);
```

参数	描述
hObj	单选按钮小工具的句柄。
pText	将显示在指定单选按钮旁边的文本指针。
Index	以零为基准的单选按钮的索引。

其他信息

如果使用不含文本（古体字）的 RADIO 小工具，则聚焦框显示在小工具按钮的四周。如果使用单选按钮文本，则聚焦框显示在当前所选小工具单选按钮文本的四周。

示例

以下示例说明如何添加上述屏幕截图中所显示的文本：


```
RADIO_SetText(hRadio_0, "Red",   0);
RADIO_SetText(hRadio_0, "Green", 1);
RADIO_SetText(hRadio_0, "Blue", 2);
```

RADIO_SetTextColor()

之前	之后
	

描述

设置显示单选按钮旁边可选文本所使用的文本颜色。

原型

```
void RADIO_SetTextColor(RADIO_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	单选按钮小工具的句柄。
Color	显示文本所使用的颜色。

其他信息

有关如何将文本添加至单选框小工具的信息，请参阅 “RADIO_SetText()”（第 614 页）。

RADIO_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

RADIO_SetValue()

描述

设置当前按钮选定内容。

原型

```
void RADIO_SetValue(RADIO_Handle hObj, int v);
```

参数	描述
hObj	单选按钮小工具的句柄。
v	要设置的值。

其他信息

RADIO 小工具中最顶端的单选按钮始终具有 0 值，向下一个按钮始终为 1，再下一个为 2，以此类推。

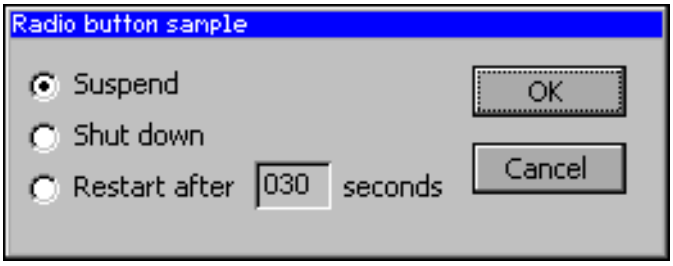
15.20.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- DIALOG_Radio.c

需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

DIALOG_Radio.c 的屏幕截图：



15.21 SCROLLBAR：滚动条小工具

滚动条用于滚动列表框或任何其他类型的窗口。它们可以水平 （如下所示）或垂直创建。

通常，滚动条将附加到现有窗口，例如下图所示的列表框：

所有与 SCROLLBAR 相关的例程均在 SCROLLBAR*.c、SCROLLBAR.h 文件中。所有识别符均为带有前缀的 SCROLLBAR。

皮肤设置 ...

... 可用于该小工具。上述屏幕截图显示使用默认皮肤的小工具。有关详细信息，请参阅“皮肤设置”一章。

15.21.1 配置选项

类型	宏	默认值	描述
N	SCROLLBAR_COLOR_SHAFT_DEFAULT	0x808080	轴的颜色。
N	SCROLLBAR_COLOR_ARROW_DEFAULT	GUI_BLACK	箭头颜色。
N	SCROLLBAR_COLOR_THUMB_DEFAULT	0xc0c0c0	缩略图区域的颜色。
N	SCROLLBAR_THUMB_SIZE_MIN_DEFAULT	4	最小缩略图尺寸。

15.21.2 通知代码

以下事件将作为 WM_NOTIFY_PARENT 消息的一部分从滚动条小工具发送至其父窗口：

消息	描述
WM_NOTIFICATION_CLICKED	已单击滚动条。
WM_NOTIFICATION_RELEASED	已释放滚动条。
WM_NOTIFICATION_SCROLLBAR_ADDED	滚动条刚刚被添加 （附加）至现有窗口。需要通知窗口，以便使其能初始化滚动条。
WM_NOTIFICATION_VALUE_CHANGED	滚动条的值可以通过移动缩略图或按下箭头按钮进行更改。

15.21.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_RIGHT	将滚动条的当前值增加 1。
GUI_KEY_DOWN	将滚动条的当前值增加 1。
GUI_KEY_PGDOWN	将滚动条的当前值增加 1 页。
GUI_KEY_LEFT	将滚动条的当前值减小 1。
GUI_KEY_UP	将滚动条的当前值减小 1。
GUI_KEY_PGUP	将滚动条的当前值减小 1 页。

15.21.4 SCROLLBAR API

下表按字母顺序列出可用的 emWin SCROLLBAR 相关例程。这些例程的详细说明如下。

例程	描述
SCROLLBAR_AddValue()	按指定值增加或减小滚动条的值。
SCROLLBAR_Create()	创建 SCROLLBAR 小工具。（弃用）
SCROLLBAR_CreateAttached()	创建附加至窗口的 SCROLLBAR 小工具。
SCROLLBAR_CreateEx()	创建 SCROLLBAR 小工具。
SCROLLBAR_CreateIndirect()	从资源表条目创建 SCROLLBAR 小工具。
SCROLLBAR_CreateUser()	使用作为用户数据的额外字节来创建滚动条小工具。
SCROLLBAR_Dec()	将滚动条的值减小 1。
SCROLLBAR_GetDefaultWidth()	返回滚动条的默认宽度。
SCROLLBAR_GetNumItems()	返回项目数。
SCROLLBAR_GetPageSize()	返回页面大小 （以项目数为单位）。
SCROLLBAR_GetThumbSizeMin()	返回最小缩略图尺寸 （以像素为单位）。
SCROLLBAR_GetUserData()	检索通过 SCROLLBAR_SetUserData() 设置的数据。
SCROLLBAR_GetValue()	返回当前项目值。
SCROLLBAR_Inc()	将滚动条的值增加 1。
SCROLLBAR_SetColor()	设置滚动条的颜色。
SCROLLBAR_SetDefaultColor()	设置新滚动条的默认颜色。
SCROLLBAR_SetDefaultWidth()	设置滚动条的默认宽度。
SCROLLBAR_SetNumItems()	设置滚动项目数。
SCROLLBAR_SetPageSize()	设置页面大小 （以项目数为单位）。
SCROLLBAR_SetState()	设置滚动条的状态。
SCROLLBAR_SetThumbSizeMin()	设置最小缩略图尺寸 （以像素为单位）。
SCROLLBAR_SetUserData()	设置 SCROLLBAR 小工具的额外数据。
SCROLLBAR_SetValue()	设置滚动条的当前值。
SCROLLBAR_SetWidth()	设置滚动条的宽度。

SCROLLBAR_AddValue()

定义

按指定值增加或减小滚动条的值。

原型

```
void SCROLLBAR_AddValue(SCROLLBAR_Handle hObj, int Add);
```

参数	描述
hObj	滚动条的句柄。
Add	一次增加或减小的项目数。

其他信息

滚动条不能超过 SCROLLBAR_SetNumItems() 中设置的值。例如，如果一个窗口包含 200 个项目，且滚动条当前在值 195，那么将滚动条增加 3 个项目就会使其移动至值 198。但是，增加 10 个项目最多也只能将滚动条移动至值 200，这是该特定窗口的最大值。

SCROLLBAR_Create()

（弃用，应转而使用 SCROLLBAR_CreateEx()）

描述

在指定位置创建指定大小的 SCROLLBAR 小工具。

原型

```
SCROLLBAR_Handle SCROLLBAR_Create(int x0,          int y0,
                                  int xsize,       int ysize
                                  WM_HWIN hParent, int Id,
                                  int WinFlags,    int SpecialFlags);
```

参数	描述
x0	滚动条最左侧的像素 （在父坐标中）。
y0	滚动条最顶端的像素 （在父坐标中）。
xsize	滚动条的水平尺寸 （以像素为单位）。
ysize	滚动条的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。
Id	将返回的 ID。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow() ）
SpecialFlags	特殊创建标记 （参见 SCROLLBAR_CreateIndirect() 下方的间接创建标记）。

返回值

所创建的 SCROLLBAR 小工具的句柄；如果函数执行失败，则返回值为 0。

SCROLLBAR_CreateAttached()

描述

创建附加到现有窗口的滚动条。

原型

```
SCROLLBAR_Handle SCROLLBAR_CreateAttached(WM_HWIN hParent,
                                          int     SpecialFlags);
```

参数	描述
hParent	父窗口的句柄。
SpecialFlags	特殊创建标记 （参见 SCROLLBAR_CreateIndirect() 下方的间接创建标记）。

返回值

所创建的 SCROLLBAR 小工具的句柄；如果函数执行失败，则返回值为 0。

其他信息

附加的滚动条实质上是一个子窗口，它将自己放置在父窗口上并进行相应的操作。
垂直附加滚动条将自动位于父窗口右侧；水平滚动条位于底部。因为只有一个水平滚动条和一个垂直滚动条可附加到父窗口，所以 ID 需要作为参数传递。以下固定 ID 将在创建附加滚动条时自动分配：
GUI_ID_HSCROLL 用于水平滚动条，而
GUI_ID_VSCROLL 则用于垂直滚动条。

示例

创建带附加滚动条的列表框：

```
LISTBOX_Handle hListBox;
hListBox = LISTBOX_Create(ListBox, 50, 50, 100, 100, WM_CF_SHOW);
SCROLLBAR_CreateAttached(hListBox, SCROLLBAR_CF_VERTICAL);
```

上述示例的屏幕截图

左侧图片显示列表框创建后的样式。右侧图片显示带有附加垂直滚动条：

之后	创建后带有附加垂直滚动条
	

SCROLLBAR_CreateEx()

描述

在指定位置创建指定大小的 SCROLLBAR 小工具。

原型

```
SCROLLBAR_Handle SCROLLBAR_CreateEx(int     x0,      int y0,
                                    int     xsize,   int ysize,
                                    WM_HWIN hParent, int WinFlags,
                                    int     ExFlags, int Id);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新创建的 SCROLLBAR 小工具将作为桌面 （顶层窗口）的子窗口。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	特殊创建标记 （参见 SCROLLBAR_CreateIndirect() 下方的间接创建标记）。
Id	小工具的窗口 ID。

返回值

所创建的 SCROLLBAR 小工具的句柄；如果函数执行失败，则返回值为 0。

SCROLLBAR_CreateIndirect()

在本章开始部分解释的原型。以下标记可以用作以参数来进行传递的资源 Flags 元素：

允许的间接创建标记 （可以通过 “OR” 操作进行组合）	
SCROLLBAR_CF_VERTICAL	创建垂直滚动条 （默认为水平滚动条）。
SCROLLBAR_CF_FOCUSSABLE	为滚动条指定输入焦点。

资源表中未使用 Para 元素。

SCROLLBAR_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细说明，可参阅函数 SCROLLBAR_CreateEx()。

SCROLLBAR_Dec()

描述

将滚动条的当前值减小 1。

原型

```
void SCROLLBAR_Dec(SCROLLBAR_Handle hObj);
```

参数	描述
hObj	滚动条的句柄。

其他信息

“项目”的定义为特定应用，尽管在多数情况下就是一行。项目是以 0 值开始从顶端到底部、从左侧到右侧进行编号。

SCROLLBAR_GetDefaultWidth()

描述

返回创建滚动条所使用的默认宽度。

原型

```
int SCROLLBAR_GetDefaultWidth(void);
```

返回值

创建滚动条所使用的默认宽度。

SCROLLBAR_GetNumItems()

描述

返回滚动条项目数。

原型

```
int SCROLLBAR_GetNumItems(SCROLLBAR_Handle hObj);
```

参数	描述
hObj	滚动条的句柄

返回值

滚动条项目数。

SCROLLBAR_GetPageSize()

描述

返回页面大小。

原型

```
int SCROLLBAR_GetValue(SCROLLBAR_Handle hObj);
```

参数	描述
hObj	滚动条的句柄。

返回值

项目数指定为一页。

SCROLLBAR_GetThumbSizeMin()

描述

返回最小缩略图尺寸 （以像素为单位）。

原型

```
int SCROLLBAR_GetThumbSizeMin(void);
```

返回值

最小缩略图尺寸 （以像素为单位）。

SCROLLBAR_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

SCROLLBAR_GetValue()

描述

返回当前项的值。

原型

```
int SCROLLBAR_GetValue(SCROLLBAR_Handle hObj);
```

参数	描述
hObj	滚动条的句柄。

返回值

当前项的值。

SCROLLBAR_Inc()

描述

将滚动条的当前值增加 1。

原型

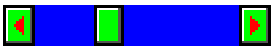
```
void SCROLLBAR_Inc(SCROLLBAR_Handle hObj);
```

参数	描述
hObj	滚动条的句柄。

其他信息

“项目”的定义为特定应用，尽管在多数情况下就是一行。项目是以 0 值开始从顶端到底部、从左侧到右侧进行编号。

SCROLLBAR_SetColor()

之前	之后
	

描述

设置滚动条的指定颜色属性。

原型

```
GUI_COLOR SCROLLBAR_SetColor(SCROLLBAR_Handle hObj, int Index,
                             GUI_COLOR        Color);
```

参数	描述
hObj	滚动条的句柄。
Index	（参见下表）
Color	要使用的颜色。

参数 Index 的允许值	
SCROLLBAR_CI_THUMB	缩略图区域的颜色。
SCROLLBAR_CI_SHAFT	轴的颜色。
SCROLLBAR_CI_ARROW	箭头颜色。

返回值

先前用于指定索引的颜色。

SCROLLBAR_SetDefaultColor()

描述

为新滚动条设置默认颜色属性。

原型

```
GUI_COLOR SCROLLBAR_SetDefaultColor(GUI_COLOR Color, unsigned int Index);
```

参数	描述
Color	用作新滚动条默认值的颜色。
Index	（参见 SCROLLBAR_SetColor() 下方的表格）

返回值

先前默认颜色。

SCROLLBAR_SetDefaultWidth()

描述

设置创建滚动条所使用的默认宽度。

原型

```
int SCROLLBAR_SetDefaultWidth(int DefaultWidth);
```

参数	描述

返回值

先前的默认宽度。

SCROLLBAR_SetNumItems()

描述

设置滚动项目数。

原型

```
void SCROLLBAR_SetNumItems(SCROLLBAR_Handle hObj, int NumItems);
```

参数	描述
hObj	滚动条的句柄。
NumItems	要设置的项目数。

其他信息

“项目”的定义为特定应用，尽管在多数情况下就是一行。
指定的项目数为最大值；滚动条不能超过该值。

SCROLLBAR_SetPageSize()

描述

设置页面大小。

原型

```
void SCROLLBAR_SetPageSize(SCROLLBAR_Handle hObj, int PageSize);
```

参数	描述
hObj	滚动条的句柄。
PageSize	页面大小 （以项目数为单位）。

其他信息

页面大小被指定为一页的项目数。如果用户使用键盘或通过在滚动条区域点击鼠标向上或向下翻页，则窗口按照为一页指定的项目数向上或向下滚动。

SCROLLBAR_SetState()

描述

设置滚动条的状态。

原型

```
void SCROLLBAR_SetState(SCROLLBAR_Handle        hObj,
                        const WM_SCROLL_STATE * pState);
```

参数	描述
hObj	滚动条的句柄。
pState	类型 WM_SCROLL_STATE 的数据结构指针。

其他信息

数据结构定义如下：

```
typedef struct {
  int NumItems;
  int v;
  int PageSize;
} WM_SCROLL_STATE;
```

SCROLLBAR_SetThumbSizeMin()

描述

设置最小缩略图尺寸 （以像素为单位）。

原型

```
int SCROLLBAR_SetThumbSizeMin(int ThumbSizeMin);
```

参数	描述
ThumbSizeMin	要设置的最小缩略图尺寸。

返回值

旧的最小缩略图尺寸 （以像素为单位）。

SCROLLBAR_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

SCROLLBAR_SetValue()

描述

设置滚动条的当前值。

原型

```
void SCROLLBAR_SetValue(SCROLLBAR_Handle hObj, int v);
```

参数	描述
hObj	滚动条的句柄。
v	要设置的值。

SCROLLBAR_SetWidth()

描述

设置滚动条的宽度。

原型

```
void SCROLLBAR_SetWidth(SCROLLBAR_Handle hObj, int Width);
```

参数	描述
hObj	滚动条的句柄。
Width	要设置的宽度。

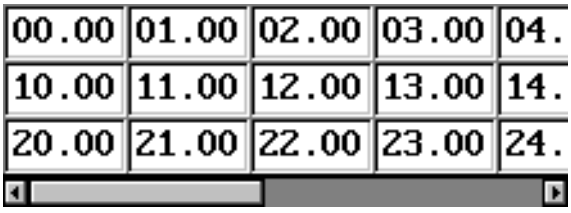
15.21.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- WIDGET_ScrollbarMove.c

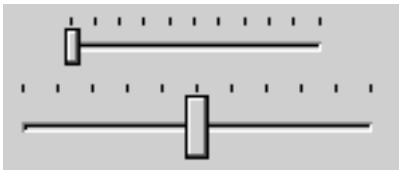
需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_ScrollbarMove.c 的屏幕截图：



15.22 SLIDER：滑块小工具

滑块小工具的常见用途是：使用滑动条来修改各项数值。小工具包含滑动条和滑动条旁边的刻度标记。在拖动滑动条时，这些刻度标记可规定光标按指定间距移动。有关如何使用刻度标记规定光标按指定间距移动的详细信息，请参阅函数 SLIDER_SetRange()。



所有与SLIDER相关的例程均在SLIDER*.c、SLIDER.h文件中。所有标识符均以SLIDER为前缀。

皮肤设置 ...

... 可用于该小工具。上述屏幕截图显示使用默认皮肤的小工具。有关详细信息，请参阅“皮肤设置”一章。

15.22.1 配置选项

类型	宏	默认值	描述
N	SLIDER_BKCOLOR0_DEFAULT	0xc0c0c0	背景颜色。
N	SLIDER_COLOR0_DEFAULT	0xc0c0c0	滑块 （缩略图）颜色。
N	SLIDER_FOCUSCOLOR_DEFAULT	GUI_BLACK	显示聚焦框所使用的默认颜色。

15.22.2 通知代码

以下事件将作为 WM_NOTIFY_PARENT 消息的一部分从滑块小工具：

消息	描述
WM_NOTIFICATION_CLICKED	已单击滑块小工具。
WM_NOTIFICATION_RELEASED	已释放滑块小工具。
WM_NOTIFICATION_VALUE_CHANGED	滑块小工具的值可以通过移动缩略图进行更改。

15.22.3 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_RIGHT	将滑动条的当前值增加一个项目。
GUI_KEY_LEFT	将滑动条的当前值减小一个项目。

15.22.4 SLIDER API

下表按字母顺序列出可用的 emWin SLIDER 相关例程。这些例程的详细说明如下。

例程	描述
SLIDER_Create()	创建 SLIDER 小工具。（弃用）
SLIDER_CreateEx()	创建 SLIDER 小工具。
SLIDER_CreateIndirect()	从资源表条目创建 SLIDER 小工具。
SLIDER_CreateUser()	使用作为用户数据的额外字节来创建 SLIDER 小工具。
SLIDER_Dec()	减小滑动条的值。
SLIDER_GetUserData()	检索通过 SLIDER_SetUserData() 设置的数据。

例程	描述
SLIDER_GetValue()	返回滑动条的当前值。
SLIDER_Inc()	增加滑动条的值。
SLIDER_SetBkColor()	设置滑动条的背景颜色。
SLIDER_SetDefaultFocusColor()	设置新滑动条的默认聚焦框颜色。
SLIDER_SetFocusColor()	设置聚焦框的颜色。
SLIDER_SetNumTicks()	设置滑动条的刻度标记数。
SLIDER_SetRange()	设置滑块值的范围。
SLIDER_SetUserData()	设置 SLIDER 小工具的额外数据。
SLIDER_SetValue()	设置滑动条的当前值。
SLIDER_SetWidth()	设置滑动条的宽度。

SLIDER_Create()

（弃用，应使用 SLIDER_CreateEx() 来代替）

描述

在指定位置创建指定大小的 SLIDER 小工具。

原型

```
SLIDER_Handle SLIDER_Create(int     x0,       int y0,
                            int     xsize,    int ysize,
                            WM_HWIN hParent,  int Id,
                            int     WinFlags, int SpecialFlags);
```

参数	描述
x0	滑块最左侧的像素 （在父坐标中）。
y0	滑块最顶端的像素 （在父坐标中）。
xsize	滑块的水平尺寸 （以像素为单位）。
ysize	滑块的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。
Id	将返回的 ID
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅“窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
SpecialFlags	特殊创建标记 （参见 SLIDER_CreateIndirect() 下方的间接创建标记）。

返回值

所创建的 SLIDER 小工具的句柄；如果函数执行失败，则返回值为 0。

SLIDER_CreateEx()

描述

在指定位置创建指定大小的 SLIDER 小工具。

原型

```
SLIDER_Handle SLIDER_CreateEx(int     x0,       int y0,
                              int     xsize,    int ysize,
                              WM_HWIN hParent,  int WinFlags,
                              int     ExFlags,  int Id);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新 SLIDER 小工具将作为桌面 （顶层窗口）的子窗口。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	特殊创建标记 （参见 SLIDER_CreateIndirect() 下方的间接创建标记）。
Id	小工具的窗口 ID。

返回值

所创建的 SLIDER 小工具的句柄；如果函数执行失败，则返回值为 0。

SLIDER_CreateIndirect()

在本章开始部分解释的原型。以下标记可以用作以参数来进行传递的资源 Flags 元素：

允许的间接创建标记	
SLIDER_CF_VERTICAL	创建垂直滑块 （默认为水平滑块）。

资源表中未使用 Para 元素。

SLIDER_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细说明，可参阅函数 SLIDER_CreateEx()。

SLIDER_Dec()

描述

将滑动条的当前值减小一个项目。

原型

```
void SLIDER_Dec(SLIDER_Handle hObj);
```

参数	描述
hObj	滑块小工具的句柄。

SLIDER_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

SLIDER_GetValue()

描述

返回滑动条的当前值。

原型

```
int SLIDER_GetValue(SLIDER_Handle hObj);
```

参数	描述
hObj	滑块小工具的句柄。

返回值

滑块的当前值。

SLIDER_Inc()

描述

将滑动条的当前值增加一个项目。

原型

```
void SLIDER_Inc(SLIDER_Handle hObj);
```

参数	描述
hObj	滑块小工具的句柄。

SLIDER_SetBkColor()

描述

设置滑块的背景颜色。

原型

```
void SLIDER_SetBkColor(SLIDER_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	滑块小工具的句柄。
Color	要使用的背景颜色。 （范围 0x000000 和 0xFFFFFF 或有效颜色定义） GUI_INVALID_COLOR 可使背景透明

其他信息

该小工具的背景可以填充任何可用颜色或透明色。如果指定一个有效的 RGB 颜色，则背景将填充该颜色，否则背景 （通常是父窗口的内容）是可见的。如果背景是透明的，则该小工具被视为透明窗口，否则为不透明窗口。需要注意的是，使用背景颜色可使显示更加有效 （速度更快）。

按照默认设置，该小工具为透明窗口。透明窗口背景的外观取决于父窗口的外观。如果透明窗口需要重新绘制，则首先要通过向父窗口发送 WM_PAINT 消息描述背景。
如果配合有效颜色使用此函数，则窗口状态将从透明更改为不透明；而如果窗口需要重新描述，则背景将填充指定颜色。
如果 GUI_INVALID_COLOR 传递到该函数，则状态将从不透明更改为透明。

SLIDER_SetDefaultFocusColor()

描述

设置新滑动条的默认聚焦框颜色。

原型

```
GUI_COLOR SLIDER_SetDefaultFocusColor(GUI_COLOR Color);
```

参数	描述
Color	新滑动条所使用的默认颜色。

返回值

先前的默认聚焦框颜色。

其他信息

更多详细信息，请参阅 “SLIDER_SetFocusColor()”（第 630 页）。

SLIDER_SetFocusColor()

之前	之后
	

描述

设置显示滑动条聚焦框所使用的颜色。

原型

```
GUI_COLOR SLIDER_SetFocusColor(SLIDER_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄。
Color	聚焦框所使用的颜色。

返回值

先前的聚焦框颜色。

其他信息

仅当此小工具具有输入焦点时，聚焦框才可见。

SLIDER_SetNumTicks()

之前	之后
	

描述

设置滑动条的刻度标记数。

原型

```
void SLIDER_SetNumTicks(SLIDER_Handle hObj, int NumTicks);
```

参数	描述
hObj	滑块小工具的句柄。
NumTicks	描述的刻度标记数。

其他信息

滑块小工具创建后，默认刻度标记数为 10。刻度标记对拖动滑动条时规定光标按指定间距移动无作用。

SLIDER_SetRange()

描述

设置滑块的范围。

原型

```
void SLIDER_SetRange(SLIDER_Handle hObj, int Min, int Max);
```

参数	描述
hObj	滑块小工具的句柄。
Min	最小值。
Max	最大值。

其他信息

滑块小工具创建后，默认范围设置为 0-100。

示例

如果应在 0-2499 范围内修改数值，则设置范围如下：

```
SLIDER_SetRange(hSlider, 0, 2499);
```

如果应在 100-499 范围内修改数值，则设置范围如下：

```
SLIDER_SetRange(hSlider, 100, 499);
```

如果应在 0-5000 范围内修改数值，且滑动条应按步长 250 更改数值，则设置范围和刻度标记如下。SLIDER_GetValue() 返回的结果应乘以 250：

```
SLIDER_SetRange(hSlider, 0, 20);
SLIDER_SetNumTicks(hSlider, 21);
```

SLIDER_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

SLIDER_SetValue()

描述

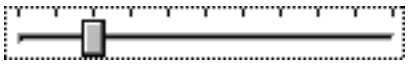
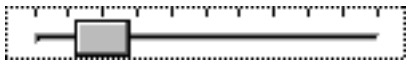
设置滑动条的当前值。

原型

```
void SLIDER_SetValue(SLIDER_Handle hObj, int v);
```

参数	描述
hObj	滑块小工具的句柄。
v	要设置的值。

SLIDER_SetWidth()

之前	之后
	

描述

设置滑动条的宽度。

原型

```
void SLIDER_SetWidth(SLIDER_Handle hObj, int Width);
```

参数	描述
hObj	滑块小工具的句柄。
Width	要设置的宽度。

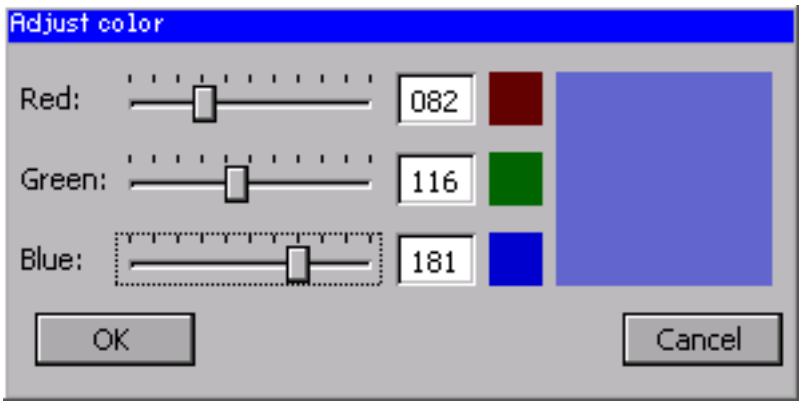
15.22.5 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：

- DIALOG_SliderColor.c

需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

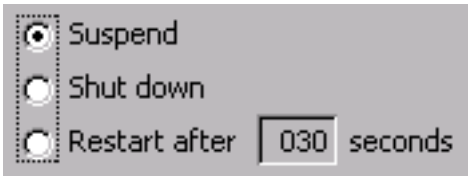
DIALOG_SliderColor.c 的屏幕截图：



15.23 文本文本小工具

文本小工具通常用于显示对话框中的文本字段，如下图消息框所示：

当然，文本字段也可用于描述其他小工具，如下所示：



所有与 TEXT 相关的例程均在 TEXT*.c、 TEXT.h 文件中。所有标识符均以 TEXT 为前缀。

15.23.1 配置选项

类型	宏	默认值	描述
N	TEXT_DEFAULT_BK_COLOR	GUI_INVALID_COLOR	默认设置为透明背景
N	TEXT_DEFAULT_TEXT_COLOR	GUI_BLACK	默认文本颜色
N	TEXT_DEFAULT_WRAPMODE	GUI_WRAPMODE_NONE	默认覆盖模式
S	TEXT_FONT_DEFAULT	&GUI_Font13_1	所使用的字体

15.23.2 键盘反应

该小工具无法获取输入焦点，并且对键盘输入无任何反应。

15.23.3 文本 API

下表按字母顺序列出可用的 emWin TEXT 相关例程。这些例程的详细说明如下。

例程	描述
TEXT_Create()	创建 TEXT 小工具。（弃用）
TEXT_CreateAsChild()	创建一个 TEXT 小工具作为子窗口。（弃用）
TEXT_CreateEx()	创建 TEXT 小工具。
TEXT_CreateIndirect()	从资源表条目创建 TEXT 小工具。
TEXT_CreateUser()	使用作为用户数据的额外字节来创建 TEXT 小工具。
TEXT_GetDefaultFont()	返回文本的默认字体。
TEXT_GetNumLines()	返回当前在小工具中所显示的线条数。
TEXT_GetUserData()	检索通过 TEXT_SetUserData() 设置的数据。
TEXT_SetBkColor()	设置文本的背景颜色。
TEXT_SetDefaultFont()	设置文本的默认字体。
TEXT_SetDefaultTextColor()	设置文本的默认文本颜色。
TEXT_SetDefaultWrapMode()	设置新文本小工具的默认覆盖模式。
TEXT_SetFont()	设置指定文本小工具所使用的字体。
TEXT_SetText()	设置指定文本小工具的文本。
TEXT_SetTextAlign()	设置指定文本小工具的文本对齐方式。
TEXT_SetTextColor()	设置指定小工具的文本颜色。
TEXT_SetUserData()	设置 TEXT 小工具的额外数据。
TEXT_SetWrapMode()	设置指定文本小工具的覆盖模式。

TEXT_Create()

（弃用，应使用 TEXT_CreateEx() 来代替）

描述

在指定位置创建指定大小的 TEXT 小工具。

原型

```
TEXT_Handle TEXT_Create(int          x0,    int y0,
                        int          xsize, int ysize,
                        int          Id,    int Flags,
                        const char * s,     int Align);
```

参数	描述
x0	文本小工具最左侧的像素 （在父坐标中）。
y0	文本小工具最顶端的像素 （在父坐标中）。
xsize	文本小工具的水平尺寸 （以像素为单位）。
ysize	文本小工具的垂直尺寸 （以像素为单位）。
Id	将返回的 ID。
Flags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
s	要显示的文本的指针。
Align	文本的对齐属性 （参见 TEXT_CreateIndirect() 下方的间接创建标记）。

返回值

所创建的 TEXT 小工具的句柄；如果函数执行失败，则返回值为 0。

TEXT_CreateAsChild()

（弃用，应使用 TEXT_CreateEx 来代替）

描述

创建一个 TEXT 小工具作为子窗口。

原型

```
TEXT_Handle TEXT_CreateAsChild(int     x0,       int          y0,
                               int     xsize,    int          ysize,
                               WM_HWIN hParent,  int          Id,
                               int     Flags,    const char * s,
                               int     Align);
```

参数	描述
x0	相对于父窗口的进度条的 X 轴位置
y0	相对于父窗口的进度条的 Y 轴位置
xsize	文本小工具的水平尺寸 （以像素为单位）。
ysize	文本小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。
Id	将返回的 ID。
Flags	窗口创建标记 （参见 TEXT_Create()）。
s	要显示的文本的指针。
Align	文本的对齐属性 （参见 TEXT_CreateIndirect() 下方的间接创建标记）。

返回值

所创建的 TEXT 小工具的句柄；如果函数执行失败，则返回值为 0。

TEXT_CreateEx()

描述

在指定位置创建指定大小的 TEXT 小工具。

原型

```
TEXT_Handle TEXT_CreateEx(int          x0,      int y0,
                          int          xsize,   int ysize,
                          WM_HWIN      hParent, int WinFlags,
                          int          ExFlags, int Id,
                          const char * pText);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新创建的 TEXT 小工具将作为桌面 （顶层窗口）的子窗口。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW（有关可用参数值的列表，请参阅“窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	文本的对齐属性 （参见 TEXT_CreateIndirect() 下方的间接创建标记）。
Id	TEXT 小工具的窗口 ID。
pText	要显示的文本的指针。

返回值

所创建的 TEXT 小工具的句柄；如果函数执行失败，则返回值为 0。

TEXT_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。以下标记可以用作以参数来进行传递的资源 Flags 元素：

允许的间接创建标记 （可以通过 “OR”操作进行组合）	
TEXT_CF_LEFT	水平对齐：左对齐
TEXT_CF_RIGHT	水平对齐：右对齐
TEXT_CF_HCENTER	水平对齐：居中对齐
TEXT_CF_TOP	垂直对齐：顶端对齐
TEXT_CF_BOTTOM	垂直对齐：底部对齐
TEXT_CF_VCENTER	垂直对齐：居中对齐

资源表中未使用 Para 元素。

TEXT_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细说明，可参阅函数 TEXT_CreateEx()。

TEXT_GetDefaultFont()

描述

返回文本小工具所使用的默认字体。

原型

```
const GUI_FONT* TEXT_GetDefaultFont(void);
```

返回值

文本小工具所使用的默认字体指针。

TEXT_GetNumLines()

描述

返回当前在小工具中所显示的线条数。

原型

```
int TEXT_GetNumLines(TEXT_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

返回值

线条数。

TEXT_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

TEXT_SetBkColor()

描述

设置文本小工具的背景颜色。

原型

```
void TEXT_SetBkColor(TEXT_Handle hObj, GUI_COLOR Color);
```

参数	描述
hObj	文本小工具的句柄。
Color	要使用的背景颜色。 （范围 0x000000 和 0xFFFFFF 或有效颜色定义） GUI_INVALID_COLOR 可使背景透明

其他信息

该小工具的背景可以填充任何可用颜色或透明色。如果指定一个有效的 RGB 颜色，则背景将填充该颜色，否则背景 （通常是父窗口的内容）是可见的。如果背景是透明的，则该小工具被视为透明窗口，否则为不透明窗口。需要注意的是，使用背景颜色可使显示更加有效 （速度更快）。

TEXT_SetDefaultFont()

描述

设置文本小工具所使用的默认字体。

原型

```
void TEXT_SetDefaultFont(const GUI_FONT * pFont);
```

参数	描述
pFont	要设置成默认值的字体指针。

TEXT_SetDefaultTextColor()

描述

设置文本小工具所使用的默认文本颜色。

原型

```
void TEXT_SetDefaultTextColor(GUI_COLOR Color);
```

参数	描述
Color	要使用的颜色。

TEXT_SetDefaultWrapMode()

描述

设置新文本小工具所使用的默认文本覆盖模式。

原型

```
GUI_WRAPMODE TEXT_SetDefaultWrapMode(GUI_WRAPMODE WrapMode);
```

参数	描述
WrapMode	新文本小工具所使用的默认文本覆盖模式。参见下表。

参数 WrapMode 的允许值	
GUI_WRAPMODE_NONE	未执行任何覆盖。
GUI_WRAPMODE_WORD	文本按字方式覆盖。
GUI_WRAPMODE_CHAR	文本按字符方式覆盖。

返回值

先前的默认文本覆盖模式。

其他信息

TEXT 小工具的默认覆盖模式为 GUI_WRAPMODE_NONE。有关文本小工具中文本覆盖的详细信息，请参阅 “TEXT_SetWrapMode()”（第 638 页）。

TEXT_SetFont()

描述

设置指定文本小工具所使用的字体。

原型

```
void TEXT_SetFont(TEXT_Handle hObj, const GUI_FONT * pFont);
```

参数	描述
hObj	文本小工具的句柄。
pFont	要使用的字体指针。

TEXT_SetText()

描述

设置指定文本小工具所使用的文本。

原型

```
void TEXT_SetText(TEXT_Handle hObj, const char * s);
```

参数	描述
hObj	文本小工具的句柄。
s	要显示的文本。

TEXT_SetTextAlign()

描述

设置指定文本小工具的文本对齐方式。

原型

```
void TEXT_SetTextAlign(TEXT_Handle hObj, int Align);
```

参数	描述
hObj	文本小工具的句柄。
Align	文本对齐方式 （参见 TEXT_Create()）。

TEXT_SetTextColor()

描述

设置指定文本小工具的文本颜色。

原型

```
void TEXT_SetTextColor(TEXT_Handle pObj, GUI_COLOR Color);
```

参数	描述
hObj	文本小工具的句柄。
Color	新文本颜色。

TEXT_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

TEXT_SetWrapMode()

描述

设置指定文本小工具的覆盖模式。

原型

```
void TEXT_SetWrapMode(TEXT_Handle hObj, GUI_WRAPMODE WrapMode);
```

参数	描述
hObj	文本小工具的句柄。
WrapMode	（参见下表）

参数 WrapMode 的允许值	
GUI_WRAPMODE_NONE	未执行任何覆盖。
GUI_WRAPMODE_WORD	文本按字方式覆盖。
GUI_WRAPMODE_CHAR	文本按字符方式覆盖。

其他信息

TEXT 小工具的默认覆盖模式为 GUI_WRAPMODE_NONE。有关文本对齐方式的详细信息，请参阅“GUI_DispStringInRectWrap()”（第 77 页）。

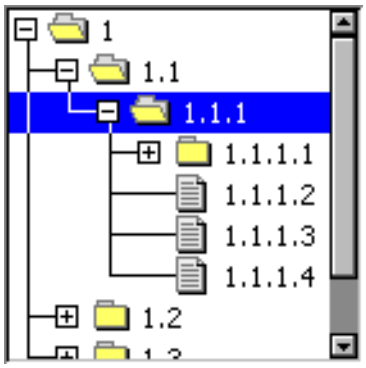
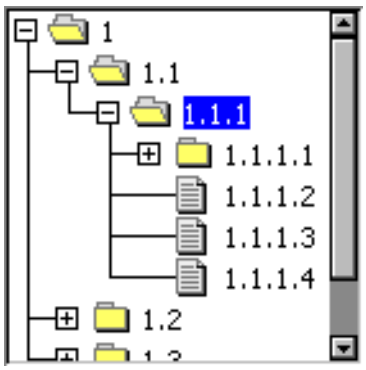
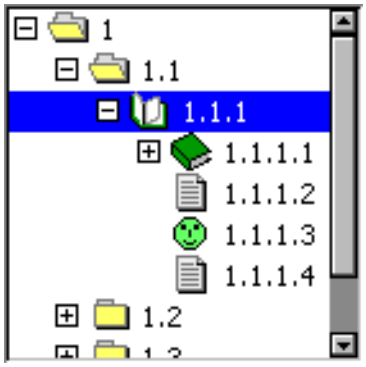
15.23.4 示例

此小工具没有特别示例。很多示例都使用此小工具：

- DIALOG_Count.c
- DIALOG_Radio.c
- WIDGET_GraphXY.c
- ...

15.24 TREEVIEW：树形视图小工具

树形视图小工具可用于显示信息的分层视图 （如目录中的文件或索引的项目），而每个项目都可以是节点或树叶。每个节点都可以具有若干子项目，也可以关闭或打开。
一个节点包含一个按钮图像 （关闭状态下显示正号，或打开状态下显示负号）、两个项目图像 （一个用于关闭状态，而一个用于打开状态）和项目文本。按下按钮图像或双击项目图像可以切换节点状态（打开或关闭）。
树叶包含项目图像和项目文本。
可以通过突出显示项目文本或整行来标记当前选定内容。按照默认设置，树形的所有项目均由线条连接。
所有与 TREEVIEW 相关的例程均在 TREEVIEW*.c、 TREEVIEW*.h 文件中。所有识别符均为带有前缀的 TREEVIEW。下表显示的是 TREEVIEW 小工具的外观：

描述	TREEVIEW 小工具
启用行选择的树形视图小工具。	
启用文本选择的树形视图小工具。	
关闭了某些定义位图和线条的应用程序的树形视图小工具。	

15.24.1 术语说明

项目

该词表示树形视图项目，可以是树叶或节点。

树叶

树叶是树形视图项目，不能拥有任何子节点。它由树叶位图和项目文本表示。

节点

节点是树形视图项目，可以拥有子节点。它由按钮位图、节点位图和项目文本表示。节点状态可以通过按下按钮位图、双击节点位图或项目选定区域进行切换。在打开状态下，可以在节点下方下一个缩进级别看见子节点。

按钮位图

该位图在节点处可见，可按下该位图来切换节点状态。

项目位图

项目位图在项目文本左侧显示。显示哪一个位图取决于项目 （树叶或节点），如果有节点，那还要取决于状态 （折叠或展开）。

展开状态

在展开状态下，节点的子节点可见，而且在按钮位图中显示负号。

折叠状态

在折叠状态下，节点的子节点隐藏，而且在按钮位图中显示正号。

连接线条

用于连接树形项目的线条。线条根据树形的层次结构将节点的按钮位图与树叶的项目位图连接在一起。

15.24.2 配置选项

类型	宏	默认值	描述
	TREEVIEW_FONT_DEFAULT	&GUI_Font13_1	绘制文本所使用的默认字体。
	TREEVIEW_BKCOLOR0_DEFAULT	GUI_WHITE	未选定状态的背景颜色。
	TREEVIEW_BKCOLOR1_DEFAULT	GUI_BLUE	选定状态的背景颜色。
	TREEVIEW_BKCOLOR2_DEFAULT	0xC0C0C0	禁用状态的背景颜色。
	TREEVIEW_TEXTCOLOR0_DEFAULT	GUI_BLACK	未选定状态的文本颜色。
	TREEVIEW_TEXTCOLOR1_DEFAULT	GUI_WHITE	选定状态的文本颜色。
	TREEVIEW_TEXTCOLOR2_DEFAULT	GUI_GRAY	禁用状态的文本颜色。
	TREEVIEW_LINECOLOR0_DEFAULT	GUI_BLACK	未选定状态的线条颜色。
	TREEVIEW_LINECOLOR1_DEFAULT	GUI_WHITE	选定状态的线条颜色。
	TREEVIEW_LINECOLOR2_DEFAULT	GUI_GRAY	禁用状态的线条颜色。
	TREEVIEW_IMAGE_CLOSED_DEFAULT		关闭状态下节点的项目图像。
	TREEVIEW_IMAGE_OPEN_DEFAULT		打开状态下节点的项目图像。
	TREEVIEW_IMAGE_LEAF_DEFAULT		树叶的项目图像。
	TREEVIEW_IMAGE_PLUS_DEFAULT		正号
	TREEVIEW_IMAGE_MINUS_DEFAULT		负号
	TREEVIEW_INDENT_DEFAULT	16	缩进所使用的像素数。
	TREEVIEW_TEXT_INDENT_DEFAULT	20	缩进文本所使用的像素数。

15.24.3 通知代码

以下事件将作为 WM_NOTIFY_PARENT 消息的一部分从树形视图小工具发送至其父窗口：

消息	描述
WM_NOTIFICATION_CLICKED	已单击树形视图。
WM_NOTIFICATION_RELEASED	已释放树形视图。
WM_NOTIFICATION_MOVED_OUT	已单击树形视图，且指针已移出小工具区域，但没有释放。
WM_NOTIFICATION_SEL_CHANGED	树形视图小工具的值 （选择内容）已更改。

15.24.4 键盘反应

如果小工具具有输入焦点，则它将对下列各键做出反应：

按键	反应
GUI_KEY_RIGHT	如果光标位于关闭节点，则节点为打开状态。 如果光标位于打开节点，则光标将移至节点的第一个子节点。
GUI_KEY_DOWN	光标移至当前位置下面的下一个可见项目。
GUI_KEY_LEFT	如果光标位于树叶处，则光标将移至项目的父节点。 如果光标位于打开节点，则该节点将关闭。 如果光标位于关闭节点，则光标移至下一个父节点。
GUI_KEY_UP	光标移至当前位置上面的上一个可见项目。

15.24.5 TREEVIEW API

下表按字母顺序列出可用的 emWin TREEVIEW 相关例程。这些例程的详细说明如下。

例程	描述
常用例程	
TREEVIEW_AttachItem()	将现有项目附加到指定树形视图。
TREEVIEW_CreateEx()	创建 TREEVIEW 小工具。
TREEVIEW_CreateIndirect()	从资源表创建 TREEVIEW 小工具。
TREEVIEW_CreateUser()	使用作为用户数据的额外字节来创建 TREEVIEW 小工具。
TREEVIEW_DecSel()	将光标移至上一个可见项目。
TREEVIEW_GetDefaultBkColor()	返回默认背景颜色。
TREEVIEW_GetDefaultFont()	返回绘制项目文本所使用的默认字体。
TREEVIEW_GetDefaultLineColor()	返回默认线条颜色。
TREEVIEW_GetDefaultTextColor()	返回默认文本颜色。
TREEVIEW_GetItem()	返回所需项目。
TREEVIEW_GetSel()	返回当前选定项目。
TREEVIEW_GetUserData()	检索通过 TREEVIEW_SetUserData() 设置的数据。
TREEVIEW_IncSel()	将光标移至下一个可见项目。
TREEVIEW_InsertItem()	在指定位置插入指定项目。
TREEVIEW_SetAutoScrollH()	管理自动使用水平滚动条。
TREEVIEW_SetAutoScrollV()	管理自动使用垂直滚动条。
TREEVIEW_SetBitmapOffset()	设置正号 / 负号位图的偏置。
TREEVIEW_SetBkColor()	设置背景颜色。
TREEVIEW_SetDefaultBkColor()	设置 TREEVIEW 小工具的默认背景颜色。
TREEVIEW_SetDefaultFont()	设置 TREEVIEW 小工具的默认字体。
TREEVIEW_SetDefaultLineColor()	设置 TREEVIEW 小工具的默认线条颜色。
TREEVIEW_SetDefaultTextColor()	设置 TREEVIEW 小工具的默认文本颜色。
TREEVIEW_SetFont()	设置绘制项目文本所使用的字体。
TREEVIEW_SetHasLines()	管理连接线条的可见性。
TREEVIEW_SetImage()	设置绘制树形视图项目所使用的图像。
TREEVIEW_SetIndent()	设置树形视图项目的缩进距离。
TREEVIEW_SetLineColor()	设置绘制连接线条所使用的颜色。
TREEVIEW_SetOwnerDraw()	启用自绘的树形视图。
TREEVIEW_SetSel()	设置树形视图的选定内容。
TREEVIEW_SetSelMode()	管理当前突出显示的选定内容。
TREEVIEW_SetTextColor()	设置绘制树形视图项目所使用的颜色。
TREEVIEW_SetTextIndent()	设置项目文本的缩进距离。
TREEVIEW_SetUserData()	设置 TREEVIEW 小工具的额外数据。
项目相关例程	
TREEVIEW_ITEM_Collapse()	折叠指定节点。
TREEVIEW_ITEM_CollapseAll()	折叠指定节点及所有子节点。
TREEVIEW_ITEM_Create()	创建新树形视图项目。
TREEVIEW_ITEM_Delete()	删除指定树形视图项目。
TREEVIEW_ITEM_Detach()	分离指定项目，但不删除。
TREEVIEW_ITEM_Expand()	展开指定节点。
TREEVIEW_ITEM_ExpandAll()	展开指定节点及所有子节点。
TREEVIEW_ITEM_GetInfo()	返回指定项目的信息结构。
TREEVIEW_ITEM_GetText()	返回项目文本。
TREEVIEW_ITEM_GetUserData()	返回树形视图项目的 UserData 值。
TREEVIEW_ITEM_SetImage()	设置绘制单个指定项目所使用的图像。
TREEVIEW_ITEM_SetText()	设置指定项目的文本。
TREEVIEW_ITEM_SetUserData()	设置树形视图项目的 UserData 值。

15.24.5.1常用例程

TREEVIEW_AttachItem()

描述

将现有项目附加到树形视图小工具。

原型

```
int TREEVIEW_AttachItem(TREEVIEW_Handle      hObj,
                        TREEVIEW_ITEM_Handle hItem,
                        TREEVIEW_ITEM_Handle hItemAt, int Position);
```

参数	描述
hObj	小工具的句柄。
hItem	将附加的项目的句柄。
hItemAt	当前附加项目 （指定所要使用的位置）的句柄。
Position	（参见下表）

参数 Position 的允许值	
TREEVIEW_INSERT_ABOVE	在与指定位置处于相同缩进级别的指定位置上方附加项目。
TREEVIEW_INSERT_BELOW	在与指定位置处于相同缩进级别的指定位置下方附加项目。
TREEVIEW_INSERT_FIRST_CHILD	通过缩进在指定位置下方附加项目。指定位置需要位于一个节点级别。

返回值

如果函数执行成功，则返回值为 0 ；否则返回值为 1。

其他信息

此函数可用于附加单一项目，也可用于附加完整树。需要注意的是，如果附加树，树的根项目需要作为 hItem 传递。如果将第一个项目附加到空的树形视图，则参数 hItem 和 Position 应为 0。

TREEVIEW_CreateEx()

描述

在指定位置创建指定大小的 TREEVIEW 小工具。

原型

```
TREEVIEW_Handle TREEVIEW_CreateEx(int     x0,      int y0,
                                  int     xsize,   int ysize,
                                  WM_HWIN hParent, int WinFlags,
                                  int     ExFlags, int Id);
```

参数	描述
x0	小工具最左侧的像素 （在父坐标中）。
y0	小工具最顶端的像素 （在父坐标中）。
xsize	小工具的水平尺寸 （以像素为单位）。
ysize	小工具的垂直尺寸 （以像素为单位）。
hParent	父窗口的句柄。如果为 0，则新创建的 TEXT 小工具将作为桌面 （顶层窗口）的子窗口。
WinFlags	窗口创建标记。为了使小工具立即可见，通常为 WM_CF_SHOW （有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	（参见下表）
Id	小工具的窗口 ID。

参数 ExFlags 的允许值	
TREEVIEW_CF_HIDELINES	未显示连接线条。
TREEVIEW_CF_ROWSELECT	激活行选择模式。
TREEVIEW_CF_AUTOSCROLLBAR_H	启用自动水平滚动条的应用。
TREEVIEW_CF_AUTOSCROLLBAR_V	启用自动垂直滚动条的应用。

返回值

新创建的小工具的句柄；如果函数执行失败，则返回值为 0。

其他信息

参数 ExFlags 的值能够以 OR 组合。

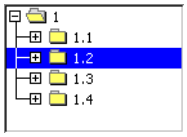
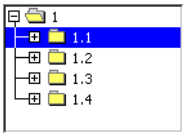
TREEVIEW_CreateIndirect()

在本章开始部分解释为 <WIDGET>_CreateIndirect() 的原型。未使用资源表的 Para 元素。

TREEVIEW_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细说明，可参阅函数 TREEVIEW_CreateEx()。

TREEVIEW_DecSel()

之前	之后
	

描述

将光标移至指定树形视图的上一个可见项目。

原型

```
void TREEVIEW_DecSel(TREEVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

其他信息

如果没有上一个可见项目，则光标仍然处于当前位置。

TREEVIEW_GetDefaultBkColor()

描述

返回新树形视图小工具所使用的默认背景颜色。

原型

```
GUI_COLOR TREEVIEW_GetDefaultBkColor(int Index);
```

参数	描述
Index	（参见下表）

参数 Index 的允许值	
TREEVIEW_CI_UNSEL	未选定元素的背景颜色。
TREEVIEW_CI_SEL	选定元素的背景颜色。
TREEVIEW_CI_DISABLED	禁用元素的背景颜色。

返回值

新创建的树形视图小工具所使用的默认背景颜色。

TREEVIEW_GetDefaultFont()

描述

返回绘制新创建的树形视图小工具项目文本所使用的默认字体。

原型

```
const GUI_FONT GUI_UNI_PTR * TREEVIEW_GetDefaultFont(void);
```

返回值

绘制新创建的树形视图小工具项目文本所使用的默认字体。

TREEVIEW_GetDefaultLineColor()

描述

返回绘制新创建的树形视图小工具连接线条所使用的默认颜色。

原型

```
GUI_COLOR TREEVIEW_GetDefaultLineColor(int Index);
```

参数	描述
Index	（参见下表）

参数 Index 的允许值	
TREEVIEW_CI_UNSEL	未选定元素的线条颜色。
TREEVIEW_CI_SEL	选定元素的线条颜色。
TREEVIEW_CI_DISABLED	禁用元素的线条颜色。

返回值

绘制新创建的树形视图小工具连接线条所使用的默认颜色。

TREEVIEW_GetDefaultTextColor()

描述

返回绘制新创建的树形视图小工具项目文本所使用的默认文本颜色。

原型

```
GUI_COLOR TREEVIEW_GetDefaultTextColor(int Index);
```

参数	描述
Index	（参见下表）

参数 Index 的允许值	
TREEVIEW_CI_UNSEL	未选定元素的文本颜色。
TREEVIEW_CI_SEL	选定元素的文本颜色。
TREEVIEW_CI_DISABLED	禁用元素的文本颜色。

返回值

绘制新创建的树形视图小工具项目文本所使用的默认文本颜色。

TREEVIEW_GetItem()

描述

返回所需树形视图项目的句柄。

原型

```
TREEVIEW_ITEM_Handle TREEVIEW_GetItem(TREEVIEW_Handle      hObj,
                                      TREEVIEW_ITEM_Handle hItem,
                                      int                  Flags);
```

参数	描述
hObj	小工具的句柄。
hItem	指定开始搜索位置的树形视图项目的句柄。
Flags	（参见下表）

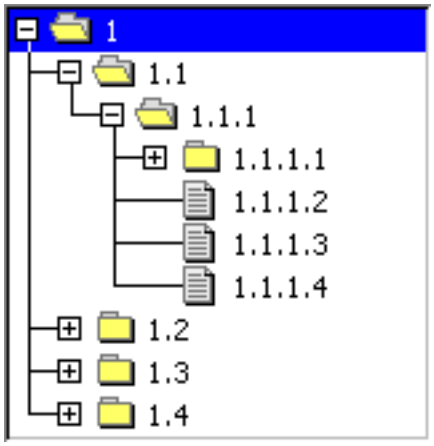
参数 Flags 的允许值	
TREEVIEW_GET_FIRST	返回树形视图小工具的第一个项目。不需要参数hItem，且可以为0。
TREEVIEW_GET_LAST	返回树形视图小工具的最后一个项目。不需要参数hItem，且可以为0。
TREEVIEW_GET_NEXT_SIBLING	返回 hItem 父节点的下一个子项目。
TREEVIEW_GET_PREV_SIBLING	返回 hItem 父节点的上一个子项目。
TREEVIEW_GET_FIRST_CHILD	返回指定节点的第一个子节点。
TREEVIEW_GET_PARENT	返回指定项目的父节点。

返回值

如果函数执行成功，则返回所需树形视图项目的句柄；否则返回值为 0。

示例

图片显示了含有几个项目的树形视图小工具。如何将参数 Flags 用于获得与参数 hItem 相应的树形视图项目，如下所示：



- TREEVIEW_GET_NEXT_SIBLING
  “1.1”的下一个同级节点为 “1.2”。
- TREEVIEW_GET_PREV_SIBLING
  “1.2”的上一个同级节点为 “1.1”。
- TREEVIEW_GET_FIRST_CHILD
  “1.1.1”的第一个子项目为 “1.1.1.1”。
- TREEVIEW_GET_PARENT
  “1.1”的父项目为 “1”。

使用 TREEVIEW_GET_FIRST 和 TREEVIEW_GET_LAST 应显而易见。如果所需项目不存在，则函数的返回值为 0。

TREEVIEW_GetSel()

描述

返回当前所选树形视图项目的句柄。

原型

```
TREEVIEW_ITEM_Handle TREEVIEW_GetSel(TREEVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

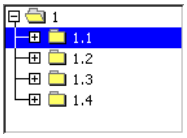
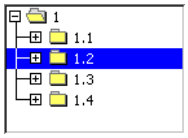
返回值

当前所选树形视图项目的句柄。如果未选择任何项目，则返回值为 0。

TREEVIEW_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

TREEVIEW_IncSel()

之前	之后
	

描述

将光标移至指定树形视图的下一个可见项目。

原型

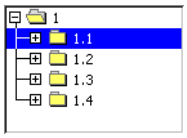
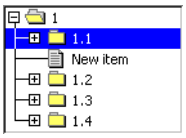
```
void TREEVIEW_IncSel(TREEVIEW_Handle hObj);
```

参数	描述
hObj	小工具的句柄。

其他信息

如果没有下一个可见项目，则光标仍然处于当前位置。

TREEVIEW_InsertItem()

之前	之后
	

描述

此函数可以创建并插入一个与指定项目相应的新树形视图项目。

原型

```
TREEVIEW_ITEM_Handle TREEVIEW_InsertItem(TREEVIEW_Handle         hObj,
                                         int                     IsNode,
                                         TREEVIEW_ITEM_Handle    hItemPrev,
                                         int                     Position,
                                         const char GUI_UNI_PTR * s);
```

参数	描述
hObj	小工具的句柄。
IsNode	（参见下表）
hItemPrev	指定新项目位置的树形视图项目的句柄。
Position	（参见下表）
s	新树形视图项目的文本。

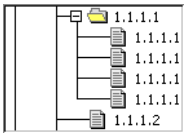
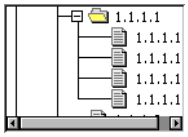
参数 IsNode 的允许值	
TREEVIEW_ITEM_TYPE_LEAF	新项目为 “树叶”。
TREEVIEW_ITEM_TYPE_NODE	新项目为 “节点”。

参数 Position 的允许值	
TREEVIEW_INSERT_FIRST_CHILD	应用于树形视图节点的第一个项目。
TREEVIEW_INSERT_ABOVE	在具有相同缩进级别的指定项目上方插入项目。
TREEVIEW_INSERT_BELOW	在具有相同缩进级别的指定项目下方插入项目。

返回值

如果函数执行成功，则返回新项目的句柄；否则返回值为 0。

TREEVIEW_SetAutoScrollH()

之前	之后
	

描述

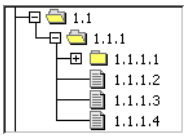
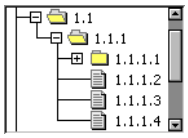
启用或禁用自动水平滚动条的应用。

原型

```
void TREEVIEW_SetAutoScrollH(TREEVIEW_Handle hObj, int State);
```

参数	描述
hObj	小工具的句柄。
State	1 代表启用自动水平滚动条，而 0 则代表禁用。

TREEVIEW_SetAutoScrollV()

之前	之后
	

描述

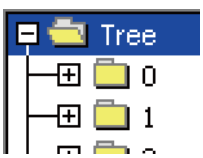
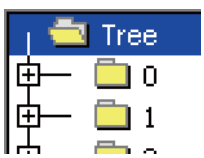
启用或禁用自动垂直滚动条的应用。

原型

```
void TREEVIEW_SetAutoScrollV(TREEVIEW_Handle hObj, int State);
```

参数	描述
hObj	小工具的句柄。
State	1 代表启用自动垂直滚动条，而 0 则代表禁用。

TREEVIEW_SetBitmapOffset()

之前	之后
	

描述

设置正号 / 负号位图的偏置。

原型

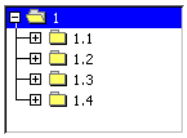
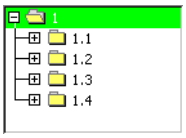
```
void TREEVIEW_SetBitmapOffset(TREEVIEW_Handle hObj, int Index,
                              int             xOff, int yOff);
```

参数	描述
hObj	小工具的句柄。
Index	该参数当前唯一允许值为 TREEVIEW_BI_PM。
xOff	水平偏移。
yOff	垂直偏移。

其他信息

如果将 xOff 和 yOff 设为 0（默认值），则正号 / 负号位图水平和垂直位于实际项目缩进空间左侧的中间位置。缩进空间与父项目（如果存在）有关，或与小工具的左边框有关。参阅 “TREEVIEW_SetIndent()”（第 654 页）的 “之前 / 之后” 屏幕截图。

TREEVIEW_SetBkColor()

之前	之后
	

描述

设置指定小工具的背景颜色。

原型

```
void TREEVIEW_SetBkColor(TREEVIEW_Handle hObj, int Index, GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄。
Index	（参见下表）
Color	要使用的颜色。

参数 Index 的允许值	
TREEVIEW_CI_UNSEL	未选定项的颜色。
TREEVIEW_CI_SEL	选定项的颜色。
TREEVIEW_CI_DISABLED	禁用项的颜色。

TREEVIEW_SetDefaultBkColor()

描述

设置新创建的树形视图小工具所使用的默认背景颜色。

原型

```
void TREEVIEW_SetDefaultBkColor(int Index, GUI_COLOR Color);
```

参数	描述
Index	请参阅 “TREEVIEW_SetBkColor()”（第 651 页）。
Color	要使用的颜色。

TREEVIEW_SetDefaultFont()

描述

设置新创建的树形视图小工具所使用的默认字体。

原型

```
void TREEVIEW_SetDefaultFont(const GUI_FONT GUI_UNI_PTR * pFont);
```

参数	描述
pFont	要使用的 GUI_FONT 结构的指针。

TREEVIEW_SetDefaultLineColor()

描述

设置新创建的树形视图小工具所使用的默认线条颜色。

原型

```
void TREEVIEW_SetDefaultLineColor(int Index, GUI_COLOR Color);
```

参数	描述
Index	请参阅 “TREEVIEW_SetBkColor()”（第 651 页）。
Color	要使用的颜色。

TREEVIEW_SetDefaultTextColor()

描述

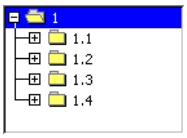
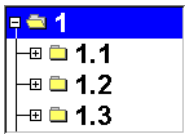
设置新创建的树形视图小工具所使用的默认文本颜色。

原型

```
void TREEVIEW_SetDefaultTextColor(int Index, GUI_COLOR Color);
```

参数	描述
Index	请参阅 “TREEVIEW_SetBkColor()”（第 651 页）。
Color	要使用的颜色。

TREEVIEW_SetFont()

之前	之后
	

描述

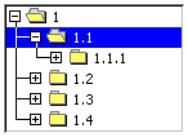
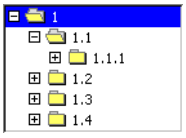
设置绘制指定树形视图小工具项目文本所使用的字体。

原型

```
void TREEVIEW_SetFont(TREEVIEW_Handle                hObj,
                      const GUI_FONT GUI_UNI_PTR * pFont);
```

参数	描述
hObj	小工具的句柄。
pFont	要使用的 GUI_FONT 结构的指针。

TREEVIEW_SetHasLines()

之前	之后
	

描述

管理树形视图项目之间连接线条的可见性。

原型

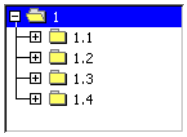
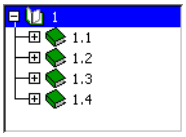
```
void TREEVIEW_SetHasLines(TREEVIEW_Handle hObj, int State);
```

参数	描述
hObj	小工具的句柄。
State	1 代表显示线条，而 0 则代表不显示线条。

其他信息

按照默认设置，显示线条。

TREEVIEW_SetImage()

之前	之后
	

描述

设置绘制树形视图项目所使用的图像。

原型

```
void TREEVIEW_SetImage(TREEVIEW_Handle    hObj,     int Index,
                       const GUI_BITMAP * pBitmap);
```

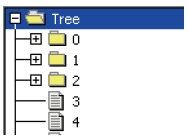
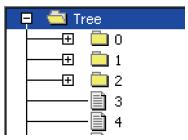
参数	描述
hObj	小工具的句柄。
Index	（参见下表）
pBitmap	要使用的位图结构指针。

参数 Index 的允许值	
TREEVIEW_BI_CLOSED	关闭节点的图像。
TREEVIEW_BI_OPEN	打开节点的图像。
TREEVIEW_BI_LEAF	树叶图像。
TREEVIEW_BI_PLUS	关闭节点的正号。
TREEVIEW_BI_MINUS	打开节点的负号。

其他信息

函数 TREEVIEW_SetItemImage() 可用于为每个项目设置单个图像。

TREEVIEW_SetIndent()

之前	之后
	

描述

设置树形视图项目的缩进距离 （以像素为单位）。缩进的默认值为 16 像素。

原型

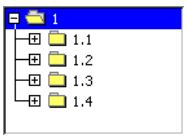
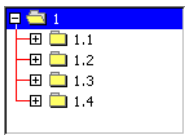
```
int TREEVIEW_SetIndent(TREEVIEW_Handle hObj, int Indent);
```

参数	描述
hObj	小工具的句柄。
Indent	缩进树形视图项目的距离 （以像素为单位）。

返回值

先前的缩进距离。

TREEVIEW_SetLineColor()

之前	之后
	

描述

设置绘制树形视图项目之间连接线条所使用的颜色。

原型

```
void TREEVIEW_SetLineColor(TREEVIEW_Handle hObj, int Index,
                           GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄。
Index	请参阅 “TREEVIEW_SetBkColor()”（第 651 页）。
Color	要使用的颜色。

TREEVIEW_SetOwnerDraw()

描述

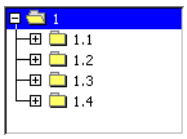
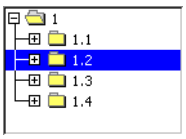
启用自绘的树形视图。

原型

```
void TREEVIEW_SetOwnerDraw(TREEVIEW_Handle          hObj,
                           WIDGET_DRAW_ITEM_FUNC * pfDrawItem);
```

参数	描述
hObj	小工具的句柄。
pfDrawItem	自绘函数的指针。请参阅 “用户绘制小工具”（第 354 页）。

TREEVIEW_SetSel()

之前	之后
	

描述

设置树形视图当前所选项目。

原型

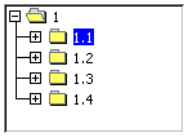
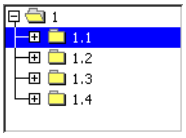
```
void TREEVIEW_SetSel(TREEVIEW_Handle hObj, TREEVIEW_ITEM_Handle hItem);
```

参数	描述
hObj	小工具的句柄。
hItem	要选择的树形视图项目的句柄。

其他信息

如果指定树形视图项目是关闭节点的子节点，则调用此函数后，任何选定内容均不可见。

TREEVIEW_SetSelMode()

之前	之后
	

描述

设置树形视图小工具选择模式。

原型

```
void TREEVIEW_SetSelMode(TREEVIEW_Handle hObj, int Mode);
```

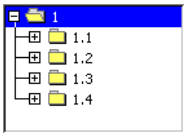
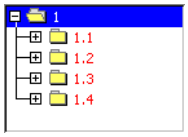
参数	描述
hObj	小工具的句柄。
Mode	（参见下表）

参数 Mode 的允许值	
TREEVIEW_SELMODE_ROW	激活行选择模式。
TREEVIEW_SELMODE_TEXT	激活文本选择模式。

其他信息

默认选择模式为文本选择。如果行选择被激活，则整行都可用于选择项目。如果文本选择有效，则仅项目文本和项目位图可用于选择。

TREEVIEW_SetTextColor()

之前	之后
	

描述

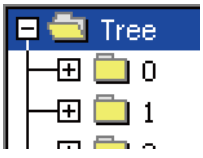
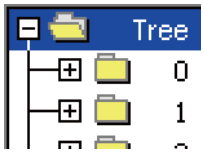
设置绘制指定小工具树形视图项目所使用的颜色。

原型

```
void TREEVIEW_SetTextColor(TREEVIEW_Handle hObj, int Index,
                           GUI_COLOR       Color);
```

参数	描述
hObj	小工具的句柄。
Index	请参阅 “TREEVIEW_SetBkColor()”（第 651 页）。
Color	要使用的颜色。

TREEVIEW_SetTextIndent()

之前	之后
	

描述

设置项目文本的缩进距离 （以像素为单位）。文本缩进的默认值为 20 像素。

原型

```
int TREEVIEW_SetTextIndent(TREEVIEW_Handle hObj, int TextIndent);
```

参数	描述
hObj	小工具的句柄。
TextIndent	要使用的文本缩进距离。

返回值

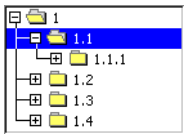
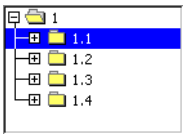
先前的文本缩进距离。

TREEVIEW_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

15.24.5.2项目相关例程

TREEVIEW_ITEM_Collapse()

之前	之后
	

描述

折叠指定节点之后显示正号。

原型

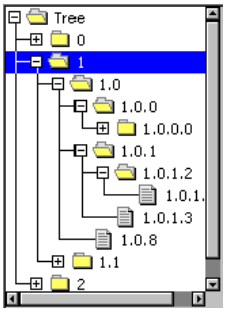
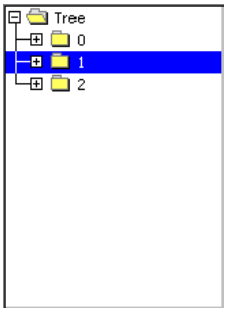
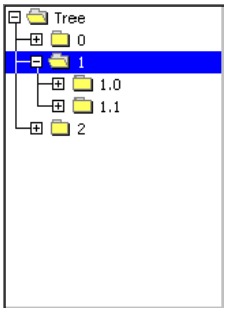
```
void TREEVIEW_ITEM_Collapse(TREEVIEW_ITEM_Handle hItem);
```

参数	描述
hItem	要折叠的项目的句柄。

其他信息

指定项目需要位于一个节点。否则函数立即返回。

TREEVIEW_ITEM_CollapseAll()

之前	所有折叠节点	再次展开节点
		

描述

折叠指定节点及所有子节点之后显示正号。

原型

```
void TREEVIEW_ITEM_CollapseAll(TREEVIEW_ITEM_Handle hItem);
```

参数	描述
hItem	要折叠的项目的句柄。

其他信息

此函数可折叠所有子节点，因此即使指定节点再次展开，所有子节点也仍处于折叠状态。

TREEVIEW_ITEM_Create()

描述

创建新树形视图项目。

原型

```
TREEVIEW_ITEM_Handle TREEVIEW_CreateItem(int                    IsNode,
                                         const char GUI_UNI_PTR * s,
                                         U32                    UserData);
```

参数	描述
IsNode	（参见下表）
s	要显示的项目文本指针。
UserData	应用程序将使用的 32 位值。

参数 IsNode 的允许值	
TREEVIEW_ITEM_TYPE_NODE	用于创建节点。
TREEVIEW_ITEM_TYPE_LEAF	用于创建树叶。

返回值

如果函数执行成功，则返回新项目的句柄；否则返回值为 0。

其他信息

树形视图项目创建后包含文本副本。

TREEVIEW_ITEM_Delete()

描述

删除指定树形视图项目。

原型

```
void TREEVIEW_ITEM_Delete(TREEVIEW_ITEM_Handle hItem);
```

参数	描述
hItem	要删除的项目的句柄。

其他信息

如果项目当前未附加到任何树形视图，则参数 hObj 应为 0。此函数可用于删除单一项目，也可用于删除完整树。如果删除树，则树的根元素应传递到该函数。

TREEVIEW_ITEM_Detach()

描述

从树形视图小工具分离指定树形视图项目。

原型

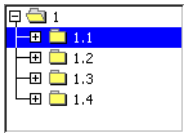
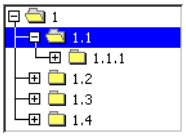
```
void TREEVIEW_ITEM_Detach(TREEVIEW_ITEM_Handle hItem);
```

参数	描述
hItem	要分离的项目的句柄。

其他信息

此函数可以从树形视图分离指定项目及其所有子节点。

TREEVIEW_ITEM_Expand()

之前	之后
	

描述

展开指定节点之后显示负号。

原型

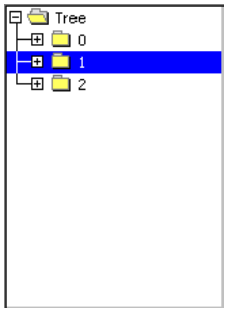
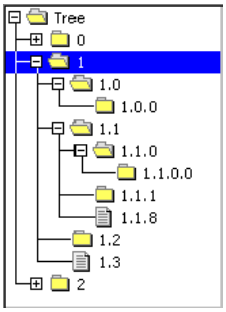
```
void TREEVIEW_ITEM_Expand(TREEVIEW_ITEM_Handle hItem);
```

参数	描述
hItem	要展开的节点的句柄。

其他信息

指定项目需要位于一个节点。否则函数立即返回。

TREEVIEW_ITEM_ExpandAll()

之前	之后
	

描述

展开指定节点及所有子节点之后显示负号。

原型

```
void TREEVIEW_ITEM_ExpandAll(TREEVIEW_ITEM_Handle hItem);
```

参数	描述
hItem	要展开的项目的句柄。

TREEVIEW_ITEM_GetInfo()

描述

返回含有指定项目信息的结构。

原型

```
void TREEVIEW_ITEM_GetInfo(TREEVIEW_ITEM_Handle   hItem,
```

```
                                TREEVIEW_ITEM_INFO    * pInfo);
```

参数	描述
hItem	树形视图项目的句柄。
pInfo	将由函数填充的 TREEVIEW_ITEM_INFO 结构指针。

TREEVIEW_ITEM_INFO 的元素

数据类型	元素	描述
int	IsNode	如果项目是一个节点，则为 1；如果不是，则为 0。
int	IsExpanded	如果项目 （节点）打开，则为 1；如果关闭，则为 0。
int	HasLines	如果连接线条可见，则为 1；如果不可见，则为 0。
int	HasRowSelect	如果行选择有效，则为 1；如果无效，则为 0。
int	Level	项目的缩进级别。

TREEVIEW_ITEM_GetText()

描述

返回指定树形视图项目的项目文本。

原型

```
void TREEVIEW_ITEM_GetText(TREEVIEW_ITEM_Handle hItem,
                           U8 * pBuffer, int MaxNumBytes);
```

参数	描述
hItem	树形视图项目的句柄。
pBuffer	由函数填充的缓冲区指针。
MaxNumBytes	缓冲区大小 （以字节为单位）。

其他信息

如果 MaxNumBytes 小于项目文本长度，则缓冲区将填充项目文本的第一个 MaxNumBytes。

TREEVIEW_ITEM_GetUserData()

描述

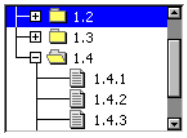
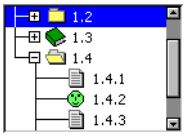
此函数可返回与指定树形视图项目相关的 32 位值，应用程序也可使用。

原型

```
U32 TREEVIEW_ITEM_GetUserData(TREEVIEW_ITEM_Handle hItem);
```

参数	描述
hItem	树形视图项目的句柄。

TREEVIEW_ITEM_SetImage()

之前	之后
	

描述

此函数仅使用指定树形视图项目设置将使用的图像。

原型

```
void TREEVIEW_ITEM_SetImage(TREEVIEW_ITEM_Handle   hItem,    int Index,
                            const GUI_BITMAP     * pBitmap);
```

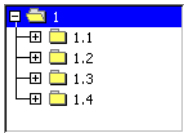
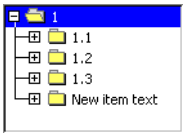
参数	描述
hItem	树形视图项目的句柄。
Index	（参见下表）
pBitmap	要使用的位图结构指针。

参数 Index 的允许值	
TREEVIEW_BI_CLOSED	关闭节点的图像。
TREEVIEW_BI_OPEN	打开节点的图像。
TREEVIEW_BI_LEAF	树叶图像。

其他信息

此函数可 “覆盖”小工具的默认图像。如果未设置任何单个图像，则使用默认图像。

TREEVIEW_ITEM_SetText()

之前	之后
	

描述

此函数可设置指定项目的文本。

原型

```
TREEVIEW_ITEM_Handle TREEVIEW_ITEM_SetText(TREEVIEW_ITEM_Handle     hItem,
                                           const char GUI_UNI_PTR * s);
```

参数	描述
hItem	树形视图项目的句柄。
s	要使用的文本指针。

返回值

带新文本树形视图项目的句柄。

其他信息

该文本将复制到树形视图项目中。需要注意的是，使用该函数会更改该项目的句柄。调用此函数后，需要使用新句柄。

TREEVIEW_ITEM_SetUserData()

描述

此函数可设置与指定树形视图项目相关的 32 位值，应用程序也可使用。

原型

```
void TREEVIEW_ITEM_SetUserData(TREEVIEW_ITEM_Handle hItem, U32 UserData);
```

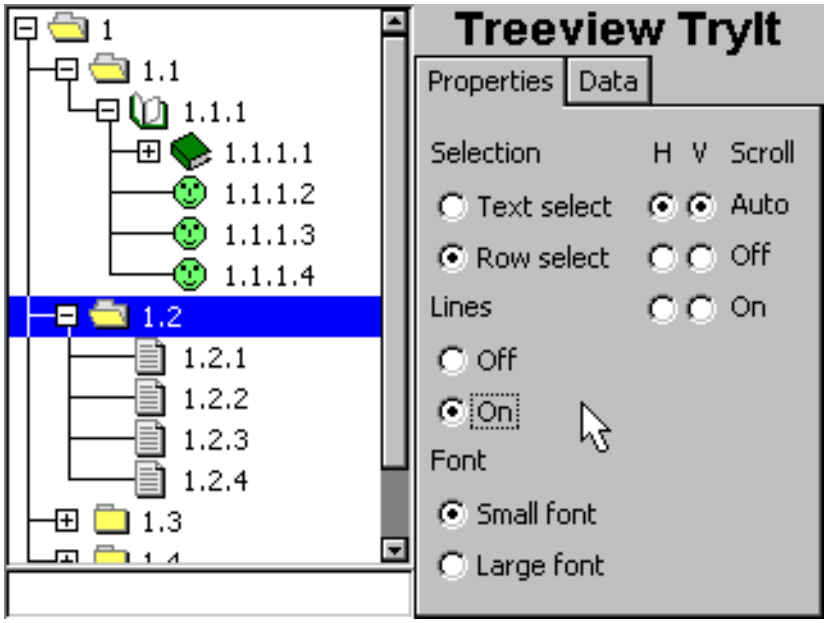
参数	描述
hItem	树形视图项目的句柄。
UserData	应用程序将使用的 32 位值。

15.24.6 示例

“示例”文件夹包含以下示例，这些示例说明了该如何使用该小工具：WIDGET_TreeviewTryit.c

需要注意的是，其他几个示例也使用该小工具，这些示例可有助于熟悉该小工具。

WIDGET_TreeviewTryit.c 的屏幕截图：



15.25 WINDOW：窗口小工具

WINDOW 小工具的作用是从资源表创建对话框窗口。如果对话框看起来不象框架窗口，则应当使用 WINDOW 小工具。窗口小工具充当的是背景以及子窗口的容器：它可包含子窗口，通常采用灰色来作为背景的填充色。
它类似于没有框架和标题栏的框架窗口，供对话框所使用。

15.25.1 配置选项

类型	宏	默认值	描述
S	WINDOW_BKCOLOR_DEFAULT	0xC0C0C0	新 WINDOW 小工具的默认背景色。

15.25.2 键盘反应

该小工具无法获取输入焦点，并且对键盘输入无任何反应。

15.25.3 WINDOW API

下表按字母顺序列出可用的 emWin WINDOW 相关程序。这些例程的详细说明如下。

例程	描述
WINDOW_CreateEx()	创建 WINDOW 小工具。
WINDOW_CreateIndirect()	从资源表项创建 WINDOW 小工具。
WINDOW_CreateUser()	使用作为用户数据的额外字节来创建 WINDOW 小工具。
WINDOW_GetUserData()	检索用 WINDOW_SetUserData() 设置的数据。
WINDOW_SetBkColor()	设置给定 WINDOW 小工具的背景色。
WINDOW_SetDefaultBkColor()	设置 WINDOW 小工具的默认背景色。
WINDOW_SetUserData()	设置 WINDOW 小工具的额外数据。

WINDOW_CreateEx()

描述

在指定位置创建指定大小的 WINDOW 小工具。

原型

```
WINDOW_Handle WINDOW_CreateEx(int           x0,       int y0,
                              int           xsize,    int ysize,
                              WM_HWIN       hParent,  int WinFlags,
                              int           ExFlags,  int Id,
                              WM_CALLBACK * cb);
```

参数	描述
x0	WINDOW 小工具的最左像素 （在父坐标中）。
y0	WINDOW 小工具的最上像素 （在父坐标中）。
xsize	X 轴中 WINDOW 小工具的大小
ysize	Y 轴中 WINDOW 小工具的大小
hParent	父窗口的句柄
WinFlags	窗口创建标记。为了使小工具立即可见，通常为WM_CF_SHOWWM_CF_SHOW（有关可用参数值的列表，请参阅 “窗口管理器 (WM)”（第 289 页）一章中的 WM_CreateWindow()）。
ExFlags	尚未使用，留待将来使用。
Id	WINDOW 小工具的窗口 ID。
cb	指向回调例程的指针。

返回值

已创建 WINDOW 小工具的句柄；函数失败时为 0。

WINDOW_CreateIndirect()

在本章开始部分解释为<WIDGET>_CreateIndirect()的原型。示例文件夹包含的文件WIDGET_Window.c说明了在对话资源中该如何使用 WINDOW 小工具。

WINDOW_CreateUser()

在本章开始部分解释为 <WIDGET>_CreateUser() 的原型。有关参数的详细描述，可参见函数 WINDOW_CreateEx()。

WINDOW_GetUserData()

在本章开始部分解释为 <WIDGET>_GetUserData() 的原型。

WINDOW_SetBkColor()

描述

设置给定 WINDOW 小工具的背景色。

原型

```
void WINDOW_SetBkColor(WM_HWIN hObj, GUI_COLOR Color);
```

参数	描述
hObj	小工具的句柄。
Color	要使用的背景颜色。

WINDOW_SetDefaultBkColor()

描述

设置用于 WINDOW 小工具的默认背景色。

原型

```
void WINDOW_SetDefaultBkColor(GUI_COLOR Color);
```

参数	描述
Color	要使用的颜色。

WINDOW_SetUserData()

在本章开始部分解释为 <WIDGET>_SetUserData() 的原型。

第 16 章

对话框

小工具可以创建并独立使用，因为它们本质上就是窗口。但是，通常需要使用对话框，它是包含一个或多个小工具的窗口。
对话框通常是一个窗口，它在出现时会要求用户输入信息。它可能包含多个小工具，要求用户根据各种选择来提供信息，或者以消息框的形式，仅提供信息（比如向用户提供注意事项或警告）和一个“确定”按钮。

16.1 对话框的基本原理

输入焦点

窗口管理器能记住用户使用触摸屏、鼠标、键盘或用其他方式最终所选择的窗口或窗口对象。该窗口会收到键盘输入消息，即具有输入焦点。
追踪输入焦点的主要原因是为了确定键盘命令的发送目的地。具有输入焦点的窗口会接收由键盘所生成的事件。
如果要将对话框内的输入焦点移至下一个焦点对话框项目，可以使用 GUI_KEY_TAB 键。如果要向后移动，则可以使用 GUI_KEY_BACKTAB。

阻塞式和非阻塞式对话框

对话框窗口可以分为阻塞式和非阻塞式。
阻塞式对话框会阻塞执行的线程。默认情况下，它有输入焦点，用户必须先关闭它，线程才能继续执行。阻塞式对话框不会同时禁用所显示的其他对话框。换言之，阻塞式对话框并非模式对话框。
如果对话框为阻塞式，则表示只有在对话框关闭后，所使用的函数（GUI_ExecDialogBox() 或 GUI_ExecCreatedDialog()）才会返回值。
而非阻塞式对话框则不会阻塞调用的线程 -- 在它为可见时，可允许任务继续运行。创建对话框后，函数会立即返回值。
需要注意的是，切勿从回调函数中调用阻塞式函数。否则，可能会导致应用程序故障。

对话框过程函数

对话框就是一个窗口，它接收消息的方式与系统中其他所有窗口一样。大多数消息由对话框的窗口回调程序自动处理，而其他消息则传递到建立对话框时所指定的回调程序，这便称为对话框过程函数。

对话框消息

发送到对话框过程函数的两种附加消息为：WM_INIT_DIALOG 和 WM_NOTIFY_PARENT。在显示对话框前， WM_INIT_DIALOG 消息会立即发送到对话框过程函数。对话框过程函数通常使用该消息来初始化小工具，并执行其他任何会影响对话框外观的初始化任务。 WM_NOTIFY_PARENT 消息则通过对话框的子窗口发送到对话框，通知任何事件的父窗口以确保同步化。通过子窗口发送的事件取决于其类型，并针对每个类型的小工具单独记录。

16.2 创建对话框

创建对话框需要两个基本要素：资源表和对话框过程；前者定义所要包括的小工具，后者定义小工具的初始值及其行为。一旦具备这两个要素，则只需进行单个函数调用（GUI_CreateDialogBox() 或 GUI_ExecDialogBox()）就能创建对话框。

16.2.1 资源表

对话框可以基于阻塞（使用 GUI_ExecDialogBox()）或非阻塞（使用 GUI_CreateDialogBox()）方式创建。必须首先定义一个资源表，以指定在对话框中所要包括的所有小工具。下面的示例说明了创建资源表的方法：


```
static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
  { FRAMEWIN_CreateIndirect, "Dialog", 0,          10,  10, 180, 230, FRAMEWIN_CF_MOVEABLE, 0 },
  { BUTTON_CreateIndirect,   "OK",     GUI_ID_OK,     100,  5,   60,  20 },
  { BUTTON_CreateIndirect,   "Cancel", GUI_ID_CANCEL, 100,  30,  60,  20 },
  { TEXT_CreateIndirect,     "LText",  0,               10,  55,  48,  15, TEXT_CF_LEFT },
  { TEXT_CreateIndirect,     "RText",  0,               10,  80,  48,  15, TEXT_CF_RIGHT },
  { EDIT_CreateIndirect,     NULL,     GUI_ID_EDIT0,    60,  55, 100,  15, 0, 50 },
  { EDIT_CreateIndirect,     NULL,     GUI_ID_EDIT1,    60,  80, 100,  15, 0, 50 },
  { TEXT_CreateIndirect,     "Hex",    0,               10, 100,  48,  15, TEXT_CF_RIGHT },
  { EDIT_CreateIndirect,     NULL,     GUI_ID_EDIT2,    60, 100, 100,  15, 0, 6 },
  { TEXT_CreateIndirect,     "Bin",    0,               10, 120,  48,  15, TEXT_CF_RIGHT },
  { EDIT_CreateIndirect,     NULL,     GUI_ID_EDIT3,    60, 120, 100,  15 },
  { LISTBOX_CreateIndirect,  NULL,     GUI_ID_LISTBOX0,10,  10,  48,  40 },
  { CHECKBOX_CreateIndirect, NULL,     GUI_ID_CHECK0,  10, 140,   0,   0 },
  { CHECKBOX_CreateIndirect, NULL,     GUI_ID_CHECK1,  30, 140,   0,   0 },
  { SLIDER_CreateIndirect,   NULL,     GUI_ID_SLIDER0, 60, 140, 100,  20 },
  { SLIDER_CreateIndirect,   NULL,     GUI_ID_SLIDER1, 10, 170, 150,  30 }
};
```

对话框中所要包括的任何小工具都必须使用 <WIDGET>_CreateIndirect() 函数来间接创建。更多详细信息，请参阅 “窗口对象 （小工具）”（第 345 页） .。

16.2.2 对话框过程函数

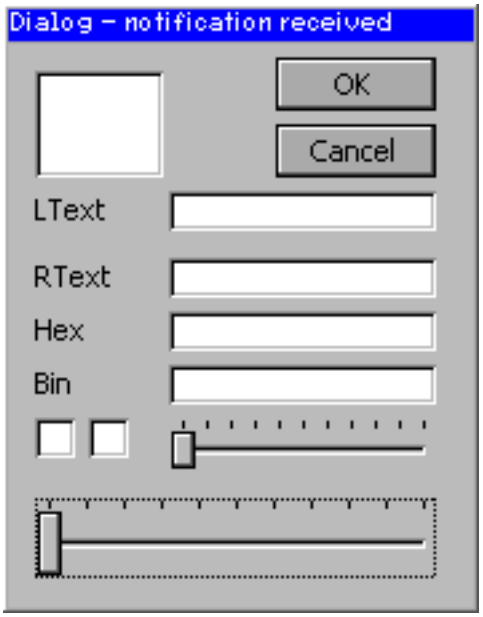
上述示例使用如下所示的空白对话框过程函数创建。在创建任何对话框过程函数时，该基本模板都将作为起始点：

```
/*********************************************************************
*
对话框过程函数
*/
static void _cbCallback(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
    default:
      WM_DefaultProc(pMsg);
  }
}
```

对于该示例，对话框显示时会有下列代码行：

```
GUI_ExecDialogBox(_aDialogCreate, GUI_COUNTOF(_aDialogCreate),
                  &_cbCallback, 0, 0, 0);
```

所生成的对话框看起来如下图一样或者与之类似 （实际外观则取决于配置和默认设置）：



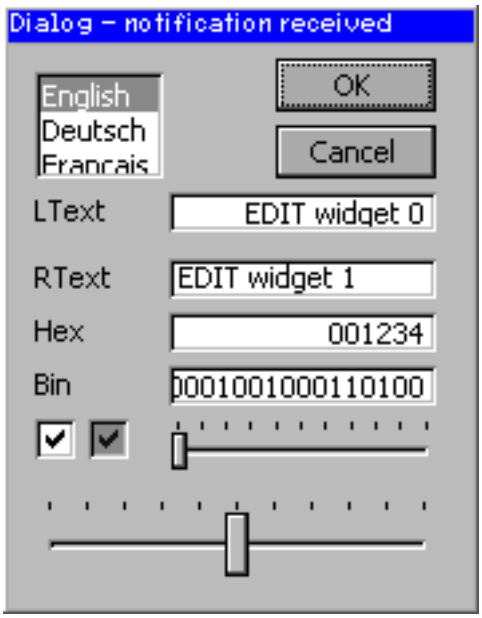
创建对话框后，所有资源表中的小工具都将可见。尽管这些小工具在上面的屏幕截图中可见，但它们是以“空”的形式出现的。这是因为对话框过程函数尚未包含初始化单个元素的代码。小工具的初始值、由它们所引起的行为以及它们之间的交互作用都需要在对话框过程中进行定义。

16.2.2.1 初始化对话框

下一步通常是使用它们各自的初始值对小工具进行初始化。在对话框过程函数中，这是对WM_INIT_DIALOG消息做出反应时的通常做法。下面的程序摘要说明了这个过程：

```
/*********************************************************************
*
对话框过程函数
*/
static void _cbCallback(WM_MESSAGE * pMsg) {
  int NCode, Id;
  WM_HWIN hEdit0, hEdit1, hEdit2, hEdit3, hListBox;
  WM_HWIN hWin = pMsg->hWin;
  switch (pMsg->MsgId) {
    case WM_INIT_DIALOG:
      /* Get window handles for all widgets */
      hEdit0   = WM_GetDialogItem(hWin, GUI_ID_EDIT0);
      hEdit1   = WM_GetDialogItem(hWin, GUI_ID_EDIT1);
      hEdit2   = WM_GetDialogItem(hWin, GUI_ID_EDIT2);
      hEdit3   = WM_GetDialogItem(hWin, GUI_ID_EDIT3);
      hListBox = WM_GetDialogItem(hWin, GUI_ID_LISTBOX0);
      /* Initialize all widgets */
      EDIT_SetText(hEdit0, "EDIT widget 0");
      EDIT_SetText(hEdit1, "EDIT widget 1");
      EDIT_SetTextAlign(hEdit1, GUI_TA_LEFT);
      EDIT_SetHexMode(hEdit2, 0x1234, 0, 0xffff);
      EDIT_SetBinMode(hEdit3, 0x1234, 0, 0xffff);
      LISTBOX_SetText(hListBox, _apListBox);
      WM_DisableWindow (WM_GetDialogItem(hWin, GUI_ID_CHECK1));
      CHECKBOX_Check(  WM_GetDialogItem(hWin, GUI_ID_CHECK0));
      CHECKBOX_Check(  WM_GetDialogItem(hWin, GUI_ID_CHECK1));
      SLIDER_SetWidth( WM_GetDialogItem(hWin, GUI_ID_SLIDER0), 5);
      SLIDER_SetValue( WM_GetDialogItem(hWin, GUI_ID_SLIDER1), 50);
      break;
    default:
      WM_DefaultProc(pMsg);
  )
}
```

经过初始化的对话框现在如下所示，所有小工具都包含了它们各自的初始值：



16.2.2.2 定义对话框行为

一旦对话框得到初始化，则剩下的所有工作便是向对话框过程函数添加代码，这样可定义小工具的行为，从而使其能充分操作。继续同一个示例，最终的对话框过程函数如下所示：

```
/*********************************************************************
*
对话框过程函数
*/
static void _cbCallback(WM_MESSAGE * pMsg) {
  int NCode, Id;
  WM_HWIN hEdit0, hEdit1, hEdit2, hEdit3, hListBox;
  WM_HWIN hWin = pMsg->hWin;
  switch (pMsg->MsgId) {
    case WM_INIT_DIALOG:
      /* Get window handles for all widgets */
      hEdit0   = WM_GetDialogItem(hWin, GUI_ID_EDIT0);
      hEdit1   = WM_GetDialogItem(hWin, GUI_ID_EDIT1);
      hEdit2   = WM_GetDialogItem(hWin, GUI_ID_EDIT2);
      hEdit3   = WM_GetDialogItem(hWin, GUI_ID_EDIT3);
      hListBox = WM_GetDialogItem(hWin, GUI_ID_LISTBOX0);
      /* Initialize all widgets */
      EDIT_SetText(hEdit0, "EDIT widget 0");
      EDIT_SetText(hEdit1, "EDIT widget 1");
      EDIT_SetTextAlign(hEdit1, GUI_TA_LEFT);
      EDIT_SetHexMode(hEdit2, 0x1234, 0, 0xffff);
      EDIT_SetBinMode(hEdit3, 0x1234, 0, 0xffff);
      LISTBOX_SetText(hListBox, _apListBox);
      WM_DisableWindow (WM_GetDialogItem(hWin, GUI_ID_CHECK1));
      CHECKBOX_Check(  WM_GetDialogItem(hWin, GUI_ID_CHECK0));
      CHECKBOX_Check(  WM_GetDialogItem(hWin, GUI_ID_CHECK1));
      SLIDER_SetWidth( WM_GetDialogItem(hWin, GUI_ID_SLIDER0), 5);
      SLIDER_SetValue( WM_GetDialogItem(hWin, GUI_ID_SLIDER1), 50);
      break;
    case WM_KEY:
      switch (((WM_KEY_INFO*)(pMsg->Data.p))->Key) {
      case GUI_ID_ESCAPE:
        GUI_EndDialog(hWin, 1);
        break;
      case GUI_ID_ENTER:
        GUI_EndDialog(hWin, 0);
        break;
      }
      break;
    case WM_NOTIFY_PARENT:
      Id    = WM_GetId(pMsg->hWinSrc);    /* Id of widget */
      NCode = pMsg->Data.v;               /* Notification code */
      switch (NCode) {
        case WM_NOTIFICATION_RELEASED:/* React only if released */
          if (Id == GUI_ID_OK) {          /* OK Button */
            GUI_EndDialog(hWin, 0);
          }
          if (Id == GUI_ID_CANCEL) {      /* Cancel Button */
            GUI_EndDialog(hWin, 1);
          }
          break;
        case WM_NOTIFICATION_SEL_CHANGED:/* Selection changed */
          FRAMEWIN_SetText(hWin, "Dialog - sel changed");
          break;
        default:
          FRAMEWIN_SetText(hWin, "Dialog - notification received");
      }
      break;
    default:
      WM_DefaultProc(pMsg);
  }
}
```

有关该完整示例的更多详情，请参阅 emWin 随附示例中的 Dialog.c。

16.3 对话框 API

下表在其各自类别内按字母顺序列出与对话框相关的可用程序。程序的详细描述可参阅下面的章节：

程序	描述
GUI_CreateDialogBox()	创建非阻塞式对话框。
GUI_ExecCreatedDialog()	执行已创建的对话框。
GUI_ExecDialogBox()	创建并执行对话框。
GUI_EndDialog()	结束对话框。

16.4 对话框

GUI_CreateDialogBox()

描述

创建非阻塞式对话框。

原型

```
WM_HWIN GUI_CreateDialogBox(const GUI_WIDGET_CREATE_INFO * paWidget,
                            int                            NumWidgets,
                            WM_CALLBACK                  * cb,
                            WM_HWIN                        hParent,
                            int                            x0,
                            int                            y0);
```

参数	描述
paWidget	定义对话框中所要包含的小工具的资源表的指针。
NumWidgets	对话框中所包含的小工具的总数。
cb	应用程序特定回调函数 （对话框过程函数）的指针。
hParent	父窗口的句柄 （0 表示没有父窗口）。
x0	对话框相对于父窗口的 X 轴位置。
y0	对话框相对于父窗口的 Y 轴位置。

GUI_ExecCreatedDialog()

描述

执行已创建的对话框。

原型

```
int GUI_ExecCreatedDialog(WM_HWIN hDialog);
```

参数	描述
hDialog	对话框的句柄。

返回值

从 GUI_EndDialog 返回的值。

GUI_ExecDialogBox()

描述

创建并执行对话框。

原型

```
int GUI_ExecDialogBox(const GUI_WIDGET_CREATE_INFO * paWidget,
                      int                            NumWidgets,
                      WM_CALLBACK                  * cb,
                      WM_HWIN                        hParent,
                      int                            x0,
                      int                            y0);
```

参数	描述
paWidget	定义对话框中所要包含的小工具的资源表的指针。
NumWidgets	对话框中所包含的小工具的总数。
cb	应用程序特定回调函数 （对话框过程函数）的指针。
hParent	父窗口的句柄 （0 表示没有父窗口）。
x0	对话框相对于父窗口的 X 轴位置。
y0	对话框相对于父窗口的 Y 轴位置。

返回值

从 GUI_EndDialog 返回的值。

GUI_EndDialog()

描述

结束 （关闭）对话框。

原型

```
void GUI_EndDialog(WM_HWIN hDialog, int r);
```

参数	描述
hDialog	对话框的句柄。
r	由 GUI_ExecDialogBox 返回的值。

返回值

指定从创建对话框的函数返回至调用线程的值 （通常只与 GUI_ExecDialogBox() 相关）。
对于非阻塞式对话框，没有等待的应用程序线程，并将忽略返回值。

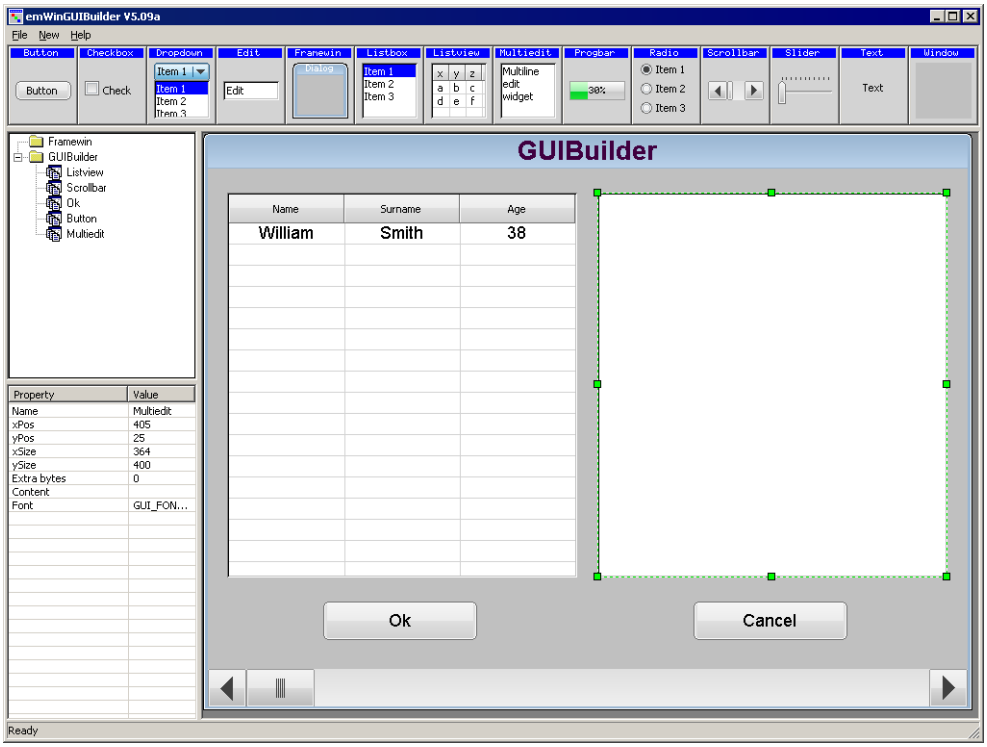
其他信息

需要注意的是，在调用该函数后，句柄 hDialog 便不再有效。如上所述，如果该函数结束对话框，则存储器中的该对话框将被删除。对于指定窗口的所有子窗口来说，这也同样适用。

第 17 章

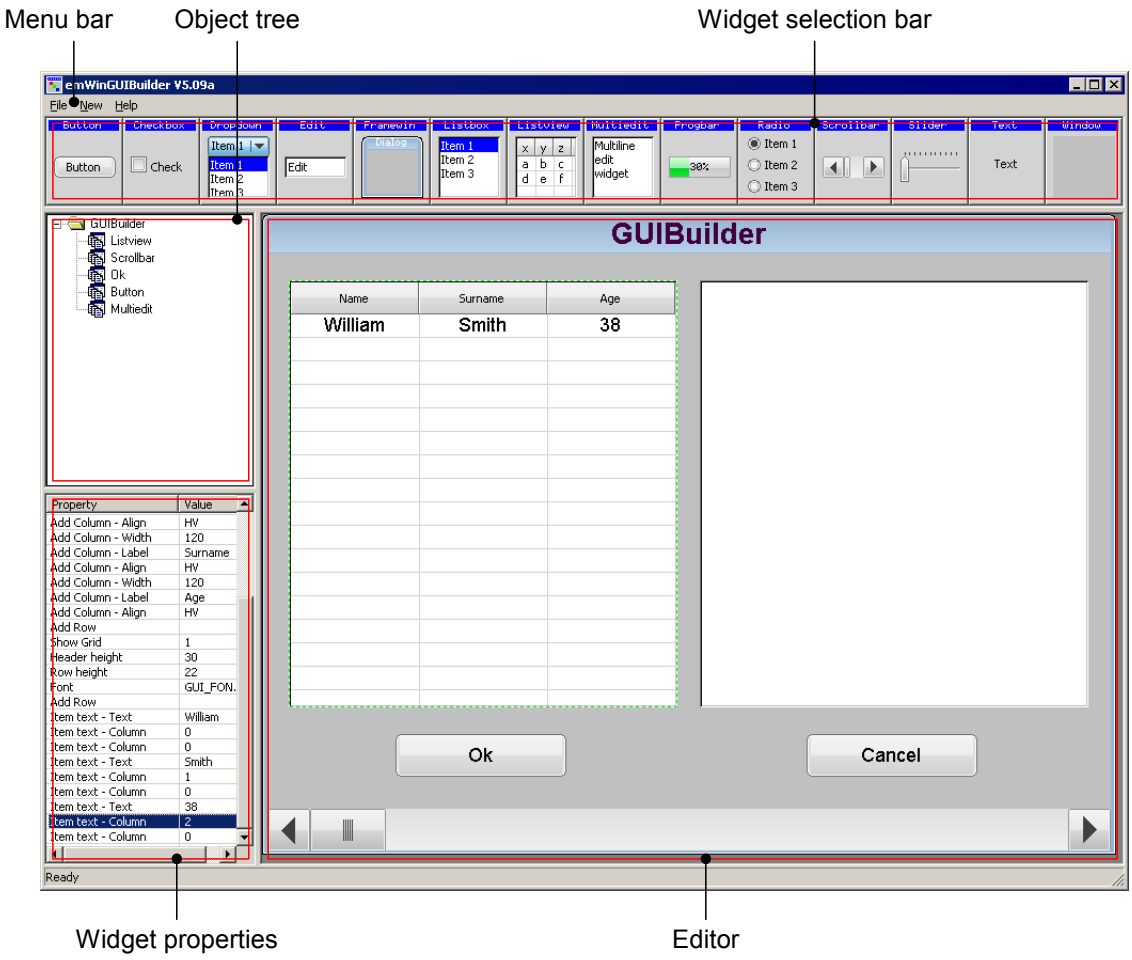
GUIBuilder

GUIBuilder应用程序是一款无需使用C编程语言即可创建对话框的工具。小工具的放置和大小调整，可通过拖放操作来实现，而无需编写源代码。根据上下文菜单，能添加其他各种属性。对这些小工具的属性进行编辑后，就能实现微调。该操作无需使用 C 编程语言。对话框可以另存为 C 文件，在添加用户定义的代码后可对其进行增强。当然， GUIBuilder 可以加载并修改具有嵌入式用户代码的这些 C 文件。



17.1 简介

下图给出了 GUI- Builder 图形用户界面的各要素：



小工具选择栏

该选择栏包含 GUIBuilder 所有可用的小工具。只要单击所需小工具上的选择栏或将其拖入编辑器区域，即可进行添加。

对象树

该区域显示所有当前加载的对话框及其子小工具。只要单击相应条目，即可选择小工具。

小工具属性

它显示每个小工具的属性，并可以用于编辑属性。

编辑器

编辑器窗口显示当前所选的对话框。它可用于放置对话框及其小工具，并调整它们的大小。

17.2 入门指南

在开始一个项目前， GUIBuilder 需要知道项目路径。根据默认设置，该路径是 GUIBuilder 的应用程序路径。所有文件都保存在该文件夹内。

设置项目路径

在首次执行项目后， GUIBuilder 目录会包含配置文件 GUIBuilder.ini。在该文件内，可以通过编辑值 ProjectPath 来更改项目路径：

```
[Settings]
ProjectPath="C:\Work\MyProject\"
```

17.3 创建对话框

下面介绍如何创建对话框以及如何修改所使用的小工具的属性。

17.3.1 选择父小工具

每个对话框都需要一个有效的父小工具。因此，必须从能够充当父小工具的小工具开始。目前，有 2 个可使用的小工具：

框架窗口小工具	窗口小工具
	

上表显示小工具选择栏的相应按钮。如果要将小工具添加到编辑器，可以使用鼠标单击有关按钮然后将该按钮拖入编辑器窗口，或使用 “ New”菜单来创建按钮。

17.3.2 在编辑器中调整大小和定位

在将小工具放入编辑器区域后，可以使用鼠标或键盘的箭头键来移动小工具。拖动时标即可调整大小。

17.3.3 修改小工具属性

Property	Value
Name	Framewin
xPos	0
yPos	0
xSize	320
ySize	240
Extra bytes	0

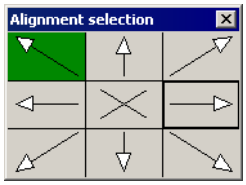
GUIBuilder 的左下方为属性窗口。创建新小工具后，该窗口会显示小工具的默认属性：名称、位置、大小和额外字节。这些属性适用于所有类型的小工具，且无法移除。与默认属性相反，所有其他属性可以通过上下文菜单或在选定相应行时按 <DEL> 进行移除。如果要更改数值，则可按下键盘上的 <ENTER>（如果目标行已选定且窗口具有焦点）或单击数值字段来选定数值。另外，可以使用右键单击出现的上下文“Edit”条目来启动编辑操作。 <ESC> 可以用于中止编辑操作。

17.3.4 向小工具添加其他函数

如果要通过小工具的可用函数获取上下文菜单，则可以右键单击目标小工具上的编辑器窗口或右键单击对象树。选择函数后，会向小工具添加新属性并开始所选函数的编辑操作。如果为数值或字母数字值，则将在属性窗口内执行编辑操作。
如果选择字体、文本对齐方式或颜色，则将出现单独的选择窗口。

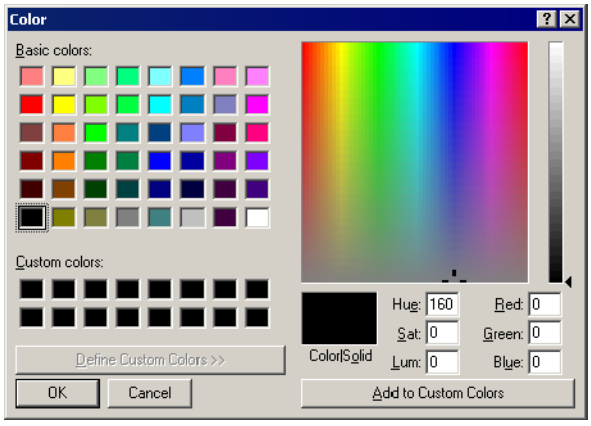
对齐方式选择

对齐方式选择对话框会以绿色显示先前选择的对齐方式。在对话框内单击可选择新的对齐方式。按下 <ESC> 键可中止选择操作。



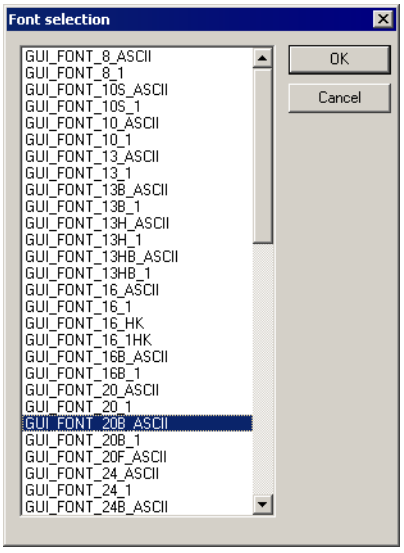
颜色选择

如果要选择颜色，则会出现 Windows 默认颜色选择对话框。按下 <ESC> 键可中止选择操作。



字体选择

字体选择对话框显示 GUIBuilder 的所有可用字体。单击目标字体即可将其选定。按下 <ESC> 键可中止选择操作。



17.3.5 删除小工具属性

使用属性窗口的上下文菜单或按下 <DEL> 键（如果小工具属性窗口中的所需属性具有焦点），即可轻松删除小工具属性。

17.3.6 删除小工具

如果在编辑器窗口已激活小工具，则按下 <DEL> 键可删除小工具。在对象树窗口中选中小工具，然后按 <DEL> 键，也可以移除小工具。
需要注意的是，如果删除父小工具，则其所有子窗口也会被删除。

17.4 保存当前的对话框

使用 “File/Save...” 菜单项可将所有当前加载的对话框保存在项目文件夹内。有关如何设置项目文件夹的详情，请参阅 “入门指南”（第 675 页）。
每个对话框都将另存为单独的 C 文件。需要注意的是，文件名将根据父小工具的小工具名称自动生成。文件名构建如下：

<Widget name>DLG.c

例如，小工具的名称为 “Framewin”，则文件将命名为 FramewinDLG.c。

17.5 GUIBuilder 的输出

如上所述， GUIBuilder 的结果只可能是 C 文件。下文提供该工具生成的一个小范例：

```
/*********************************************************************
*                                                                    *
*                SEGGER Microcontroller GmbH & Co. KG                *
*        Solutions for real time microcontroller applications        *
*                                                                    *
**********************************************************************
*                                                                    *
* C-file generated by:*
*                                                                    *
*        GUI_Builder for emWin version 5.09                          *
*        Compiled Mar 23 2011, 09:52:04                              *
*        (c) 2011 Segger Microcontroller GmbH & Co. KG               *
*                                                                    *
**********************************************************************
*                                                                    *
*        Internet:www.segger.com  Support:support@segger.com       *
*                                                                    *
**********************************************************************
*/

// USER START (Optionally insert additional includes)
// USER END

#include "DIALOG.h"

/*********************************************************************
*
*       Defines
*
**********************************************************************
*/

#define ID_FRAMEWIN_0   (GUI_ID_USER + 0x0A)
#define ID_BUTTON_0     (GUI_ID_USER + 0x0B)

// USER START (Optionally insert additional defines)
// USER END

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

// USER START (Optionally insert additional static data)
// USER END

/*********************************************************************
*
*       _aDialogCreate
*/
static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
  { FRAMEWIN_CreateIndirect, "Framewin", ID_FRAMEWIN_0, 0, 0, 320, 240, 0, 0, 0 },
  { BUTTON_CreateIndirect, "Button", ID_BUTTON_0, 5, 5, 80, 20, 0, 0, 0 },
  // USER START (Optionally insert additional widgets)
  // USER END
};

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

// USER START (Optionally insert additional static code)
// USER END

/*********************************************************************
*
*       _cbDialog
*/
static void _cbDialog(WM_MESSAGE * pMsg) {
  WM_HWIN hItem;
  int Id, NCode;
```



```
  // USER START (Optionally insert additional variables)
  // USER END

  switch (pMsg->MsgId) {
  case WM_INIT_DIALOG:
    //
    // Initialization of 'Framewin'
    //
    hItem = pMsg->hWin;
    FRAMEWIN_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
    FRAMEWIN_SetFont(hItem, GUI_FONT_24_ASCII);
    //
    // Initialization of 'Button'
    //
    hItem = WM_GetDialogItem(pMsg->hWin, ID_BUTTON_0);
    BUTTON_SetText(hItem, "Press me...");
    // USER START (Opt. insert additional code for further widget initialization)
    // USER END
    break;
  case WM_NOTIFY_PARENT:
    Id    = WM_GetId(pMsg->hWinSrc);
    NCode = pMsg->Data.v;
    switch(Id) {
    case ID_BUTTON_0:// Notifications sent by 'Button'
      switch(NCode) {
      case WM_NOTIFICATION_CLICKED:
        // USER START (Optionally insert code for reacting on notification message)
        // USER END
        break;
      case WM_NOTIFICATION_RELEASED:
        // USER START (Optionally insert code for reacting on notification message)
        // USER END
        break;
      // USER START (Opt. insert additional code for further notification handling)
      // USER END
      }
      break;
    // USER START (Optionally insert additional code for further Ids)
    // USER END
    }
    break;
  // USER START (Optionally insert additional message handling)
  // USER END
  default:
    WM_DefaultProc(pMsg);
    break;
  }
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/
/*********************************************************************
*
*       CreateFramewin
*/
WM_HWIN CreateFramewin(void) {
  WM_HWIN hWin;

  hWin = GUI_CreateDialogBox(_aDialogCreate,
                             GUI_COUNTOF(_aDialogCreate), &_cbDialog, WM_HBKWIN, 0, 0);
  return hWin;
}

// USER START (Optionally insert additional public code)
// USER END

/************************* End of file ****************************/
```


17.6 修改 C 文件

如示例代码所示，它包含自定义代码的许多部分。这些部分如下：

```
// USER START (选择性插入 ...)
// USER END
```

在上述两行之间，可以添加任何代码。需要注意的是，需要在这两行之间添加代码。注释行本身不允许修改。下面说明它如何运行：

```
// USER START (选择性插入其他包括内容)
#ifndef WIN32
  #include <ioat91sam9261.h>
#endif
// USER END
```

17.7 如何使用 C 文件

如示例输出所示，代码不包含任何使用对话框或具有可使其在屏幕上可见的其他字词的代码。每个文件结尾都有一个名为 Create<Widget name>() 的创建程序。这些程序会创建相应的对话框。只要调用这些程序，屏幕上就会显示有关对话框。

示例

以下代码显示如何在屏幕上描述上一个输出示例的对话框：

```
#include "DIALOG.h"

/*********************************************************************
*
*       Externals
*
**********************************************************************
*/
WM_HWIN CreateFramewin(void);

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/
/*********************************************************************
*
*       MainTask
*/
void MainTask(void) {
  WM_HWIN hDlg;
  GUI_Init();
  //
  // Call creation function for the dialog
  //
  hDlg = CreateFramewin();
  //
  // May do anything with hDlg
  //
  ...
  //
  // Keep program allive...
  //
  while (1) {
    GUI_Delay(10);
  }
}
```


第 18 章

换肤

换肤是一种更改一个或多个小工具外观的方法。它允许使用定义小工具呈现方式的专用皮肤来更改外观。通过这样一种仅更改皮肤的相似方法，可轻松更改整个小工具组的外观。
如果无换肤功能，则必须使用小工具成员函数来更改每个小工具的外观，或必须覆盖回调函数。

18.1 “皮肤”是什么?

皮肤就是一个简单的回调函数，用来描述小工具的所有详细信息。它的运行方式与小工具的“自绘函数”完全相同，后者是实现 “换肤”之前使用的一种小工具自定义方法。

18.2 从使用 API 函数到换肤

有多种不同的方法可更改小工具的外观。小工具 API 函数、自绘函数、换肤和覆盖回调函数可用于修改小工具的外观。应根据所要更改的内容来决定所要使用的方法。下面介绍使用每种方法可实现的效果。

使用小工具 API 函数

默认 API 函数的作用是更改各种属性（比如：大小、颜色、字体等）或者使用经典设计来绘制小工具的位图。显示可实现的效果的典型示例：

之前	之后
	

一些属性可以更改，但基本的外观将保持不变。

用户描述的函数

一些小工具（如 LISTBOX、FRAMEWIN、GRAPH 或 BUTTON 小工具）具有自绘函数。这些函数可用于描述一些其他的详细信息或替代一些项目的默认描述方法。以下屏幕截图显示框架窗口的自绘标题区域。自绘函数将在标题区域呈现渐变，而小工具 API 函数无法实现此效果：

之前	之后
	

换肤

与上述方法相反，换肤涉及整个小工具的绘制，而不仅仅是部分详细信息。通过换肤还能提升可换肤小工具的外观，使其看起来更为时尚，成为一种经典的小工具设计。下表显示与新的默认皮肤相比上述示例中 “关于”方框的外观：

经典设计	默认皮肤
	

覆盖小工具的回调函数

在实现换肤之前，更改小工具整个外观的唯一方法是覆盖小工具的回调函数。使用该方法可完全控制小工具的全部消息处理。它可以配合其他方法使用。覆盖回调函数的主要缺点是用户需要编写大量代码。此过程易于出错。

18.3 可换肤的小工具

只有小工具包含多个小工具特定详细信息时，换肤才有意义。因此，并非每个类型的小工具都需要进行换肤。例如， TEXT 小工具不需要单独的皮肤，因为它只包含文本自身。
目前，以下小工具支持换肤：

- BUTTON
- CHECKBOX
- DROPDOWN
- FRAMEWIN
- HEADER
- PROGBAR
- RADIO
- SCROLLBAR
- SLIDER

18.4 使用皮肤

emWin 发货包含所有上述可换肤小工具的即用型默认皮肤。它们名为 `<WIDGET>_SKIN_FLEX`。下表显示所有可换肤小工具可用的默认皮肤：

小工具	默认皮肤
BUTTON	BUTTON_SKIN_FLEX
CHECKBOX	CHECKBOX_SKIN_FLEX
DROPDOWN	DROPDOWN_SKIN_FLEX
FRAMEWIN	FRAMEWIN_SKIN_FLEX
HEADER	HEADER_SKIN_FLEX
PROGBAR	PROGBAR_SKIN_FLEX

小工具	默认皮肤
RADIO	RADIO_SKIN_FLEX
SCROLLBAR	SCROLLBAR_SKIN_FLEX
SLIDER	SLIDER_SKIN_FLEX

18.4.1 运行时间配置

如果要使用这些皮肤，则可以使用函数 <WIDGET>_SetSkin(<WIDGET>_SKIN_FLEX)。另外，可以通过 <WIDGET>_SetDefaultSkin() 来设置默认皮肤，以供每个新创建的小工具自动使用。

从经典设计切换至皮肤

使用皮肤时，最值得推荐的方法是首先设置小工具行为，然后再创建小工具。

示例

以下示例显示该如何使用皮肤：

```
BUTTON_SetSkin(hButton, BUTTON_SKIN_FLEX); // Sets the skin for the given widget
BUTTON_SetDefaultSkin(BUTTON_SKIN_FLEX);   // Sets the default skin for new widgets
```

18.4.2 编译时间配置

如果将换肤作为默认行为，则存在可使用的编译时间配置宏。如果要使用换肤，则根据默认设置，可以使用宏 WIDGET_USE_FLEX_SKIN。

示例

如果要使用换肤，则根据默认设置，应将该宏添加到文件 GUIConf.h 中：

```
#define WIDGET_USE_FLEX_SKIN 1
```

18.5 简单更改 “Flex”皮肤外观

API 函数可更改经典外观的属性，与之类似，“Flex”皮肤的属性也可以更改。在更改时，无需了解换肤的所有详细信息。
函数 <WIDGET>_SetSkinFlexProps()（在本章后面将详细介绍）可用于更改属性。每个皮肤都存在用于获取和设置属性的函数。

示例

以下代码显示如何更改按钮皮肤的属性：

```
BUTTON_GetSkinFlexProps(&Props, BUTTON_SKINFLEX_FOCUSSED);
Props.aColorFrame[0] = 0x007FB13C;
Props.aColorFrame[1] = 0x008FfF8F;
Props.Radius         = 6;
BUTTON_SetSkinFlexProps(&Props, BUTTON_SKINFLEX_FOCUSSED);
WM_InvalidateWindow(hWin);
```

需要注意的是，需要使受皮肤影响的窗口变为无效。每个小工具都需要调用小工具 API 函数，与之相反，皮肤并不 “知道”将使用皮肤的小工具。因此，在更改皮肤属性时，不会出现小工具自动无效并重新绘制的情况。

屏幕截图

之前	之后
	

18.6 对 “Flex”皮肤外观所作的重大更改

如果没有换肤功能，则默认设计的自绘机制对于应用程序设计人员而言相当于 “黑匣子”。如果默认外观无需重大更改，则换肤的情况也如此。如果更改默认皮肤的属性后，无法实现所需的外观，则需要了解换肤自绘机制的详细信息。

18.6.1 换肤回调机制

所有可换肤小工具的自绘机制都非常相似，如下所示：

```
int <WIDGET>_DrawSkin(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_DRAW_BACKGROUND:
    /* Draw the background */
    break;
  case WIDGET_ITEM_DRAW_TEXT:
    /* Draw the text */
    break;
  case WIDGET_ITEM_CREATE:
    /* Additional function calls required to create the widget */
    break;
  ...
}
```

Elements of WIDGET_ITEM_DRAW_INFO

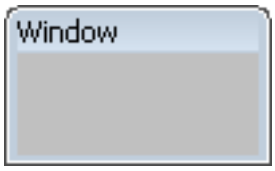
数据类型	元素	描述
WM_HWIN	hWin	小工具的句柄。
int	Cmd	要处理的命令。
int	ItemIndex	要绘制的项目的索引。
int	x0	窗口坐标中最左侧的坐标。
int	y0	窗口坐标中最顶端的坐标。
int	x1	窗口坐标中最右侧的坐标。
int	y1	窗口坐标中最底部的坐标。
void *	p	小工具特定信息的数据指针。

此机制与所有可换肤小工具相同。回调函数会收到 WIDGET_ITEM_DRAW_INFO 结构的指针。pDrawItemInfo 指出的结构包含必须处理的命令、小工具的句柄以及其含义可能因小工具而异的其他信息。换肤回调函数必须通过绘制专用的详细信息或返回专用值来作出反应。
本章后面将详细介绍如何使用描述信息。

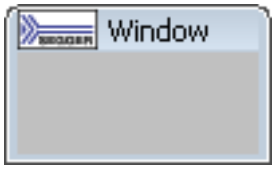
18.6.2 更改默认皮肤的外观

了解上述回调机制相当重要，因为从现有皮肤派生新皮肤即可轻松更改皮肤。小型示例将说明更改小工具默认皮肤外观的方法。

由于图标将显示在标题栏的左侧，因此就认为需要更改框架窗口皮肤的默认外观。FRAMEWIN 皮肤的默认外观如下：



该外观将更改为以下外观：



只要使用从默认外观派生的自定义皮肤，即可轻松进行更改。以下代码显示如何实现此效果。它显示了自定义换肤回调函数，该函数通过函数 FRAMEWIN_SetSkin() 用作皮肤。由于将在框架窗口的文本区域描述图标，此函数将覆盖文本描述的默认行为：

```
  case WIDGET_ITEM_DRAW_TEXT:
    ...
```

所有其他任务将由默认皮肤执行：

```
  default:
    return FRAMEWIN_DrawSkinFlex(pDrawItemInfo);
```

示例

```
static int _DrawSkinFlex_FRAME(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  char acBuffer[20];
  GUI_RECT Rect;

  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_DRAW_TEXT:
    //
    // Draw icon at the left side
    //
    GUI_DrawBitmap(&_bmLogo_30x15, pDrawItemInfo->x0, pDrawItemInfo->y0);
    //
    // Draw text beneath
    //
    FRAMEWIN_GetText(pDrawItemInfo->hWin, acBuffer, sizeof(acBuffer));
    GUI_SetColor(GUI_BLACK);
    Rect.x0 = pDrawItemInfo->x0   // Default position of text
            + _bmLogo_30x15.XSize // + X-size of icon
            + 4;                  // + small gap between icon and text
    Rect.y0 = pDrawItemInfo->y0;
    Rect.x1 = pDrawItemInfo->x1;
    Rect.y1 = pDrawItemInfo->y1;
    GUI_DispStringInRect(acBuffer, &Rect, GUI_TA_VCENTER);
    break;
  default:
     //
     // Use the default skinning routine for processing all other commands
     //
    return FRAMEWIN_DrawSkinFlex(pDrawItemInfo);
  }
  return 0;
}

void _SetSkin(WM_HWIN) {
  //
  // Set the derived
  //
  FRAMEWIN_SetSkin(hFrame, _DrawSkinFlex_FRAME);
}
```

18.6.3 命令列表

如上所述，换肤程序会收到 WIDGET_ITEM_DRAW_INFO 结构的指针。此结构的 Cmd 成员包含需要处理的命令。有多个命令将发送至小工具的换肤程序。需要注意的是，并非将所有命令都发送至所有小工具。另外，详细的含义将因小工具而异。有关如何响应命令的详细描述，将在小工具特定详细信息的后面介绍。下表概述了将发送至换肤程序的命令：

命令 Id (Cmd)	描述
创建消息	
WIDGET_ITEM_CREATE	在绘制之前和创建之后，发送至每个可换肤的小工具。
信息消息	
WIDGET_ITEM_GET_BORDERSIZE_B	用于获取下边框的大小。
WIDGET_ITEM_GET_BORDERSIZE_L	用于获取左边框的大小。
WIDGET_ITEM_GET_BORDERSIZE_R	用于获取右边框的大小。
WIDGET_ITEM_GET_BORDERSIZE_T	用于获取上边框的大小。
WIDGET_ITEM_GET_BUTTONSIZE	用于获取按钮大小。
WIDGET_ITEM_GET_XSIZE	用于获取 X 轴大小。
WIDGET_ITEM_GET_YSIZE	用于获取 Y 轴大小。
绘制消息	
WIDGET_ITEM_DRAW_ARROW	用于绘制箭头。
WIDGET_ITEM_DRAW_BACKGROUND	用于绘制背景。
WIDGET_ITEM_DRAW_BITMAP	用于绘制位图。
WIDGET_ITEM_DRAW_BUTTON	用于绘制按钮区域。
WIDGET_ITEM_DRAW_BUTTON_L	用于绘制左侧按钮区域。
WIDGET_ITEM_DRAW_BUTTON_R	用于绘制右侧按钮区域。
WIDGET_ITEM_DRAW_FOCUS	用于绘制聚焦框。
WIDGET_ITEM_DRAW_FRAME	用于绘制小工具的框架。
WIDGET_ITEM_DRAW_OVERLAP	用于绘制重叠区域。
WIDGET_ITEM_DRAW_SEP	用于绘制分隔符。
WIDGET_ITEM_DRAW_SHAFT	用于绘制轴区域。
WIDGET_ITEM_DRAW_SHAFT_L	用于绘制左侧轴区域。
WIDGET_ITEM_DRAW_SHAFT_R	用于绘制右侧轴区域。
WIDGET_ITEM_DRAW_TEXT	用于绘制文本。
WIDGET_ITEM_DRAW_THUMB	用于绘制缩略图区域。
WIDGET_ITEM_DRAW_TICKS	用于绘制刻度标记。

18.7 常用的换肤 API

下表按字母顺序列出与换肤相关的可用程序。这些函数是所有可换肤小工具所共有的，并在此列出以免重复。这些程序的详细说明如下。每个小工具可用的其他换肤成员函数可在后面章节中找到。

程序	描述
<WIDGET>_DrawSkinFlex()	默认皮肤的换肤回调函数。
<WIDGET>_GetSkinFlexProps()	返回皮肤的当前属性。
<WIDGET>_SetDefaultSkin()	设置新创建的小工具所使用的默认皮肤。
<WIDGET>_SetDefaultSkinClassic()	将经典设计设置为新创建的小工具的默认设计。
<WIDGET>_SetSkin()	设置指定小工具的皮肤。
<WIDGET>_SetSkinClassic()	设置指定小工具的经典设计。
<WIDGET>_SetSkinFlexProps()	设置皮肤的属性。

<WIDGET>_DrawSkinFlex()

描述

这些函数为默认皮肤的换肤回调函数，负责绘制完整的小工具。

原型

```
int <WIDGET>_DrawSkinFlex(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

参数	描述
pDrawItemInfo	类型 WIDGET_ITEM_DRAW_INFO 的数据结构的指针。

其他信息

派生皮肤可使用该函数来绘制默认皮肤的详细信息。

<WIDGET>_GetSkinFlexProps()

描述

使用这些函数可返回默认皮肤的属性。本章后面的小工具特定说明将详细介绍皮肤属性。

原型

```
void <WIDGET>_GetSkinFlexProps(<WIDGET>_SKINFLEX_PROPS * pProps,
                               int Index);
```

参数	描述
pProps	类型 <WIDGET>_SKINFLEX_PROPS 的皮肤特定配置结构的指针。
Index	小工具状态 （已按下、活动状态、已选定 ...），需对其详细信息进行检索。

<WIDGET>_SetDefaultSkin()

描述

这些函数设置新创建的小工具 （专用类型）所使用的默认皮肤。

原型

```
void <WIDGET>_SetDefaultSkin(WIDGET_DRAW_ITEM_FUNC * pfDrawSkin);
```

参数	描述
pfDrawSkin	类型 WIDGET_DRAW_ITEM_FUNC 的换肤回调函数的指针。

其他信息

指定的指针指向所有新创建的小工具所使用的换肤回调程序。有关更多详情，另请参阅本章后面介绍的函数 <WIDGET>_SetSkin()。

<WIDGET>_SetDefaultSkinClassic()

描述

这些函数设置所有新创建的小工具 （专用类型）的经典设计。

原型

```
void <WIDGET>_SetDefaultSkinClassic(void);
```

其他信息

使用经典设计的小工具，其行为与实施换肤功能前的行为完全一样。

<WIDGET>_SetSkin()

描述

这些函数可用于设置指定小工具的皮肤。

原型

```
void <WIDGET>_SetSkin(<WIDGET>_Handle          hObj,
                      WIDGET_DRAW_ITEM_FUNC * pfDrawSkin);
```

参数	描述
hObj	专用小工具的句柄。
pfDrawSkin	类型 WIDGET_DRAW_ITEM_FUNC 的换肤回调函数的指针。

WIDGET_DRAW_ITEM_FUNC

```
typedef int  WIDGET_DRAW_ITEM_FUNC(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo);
```

其他信息

需要注意的是，如果使用皮肤，小工具的部分默认 API 函数将无效。

<WIDGET>_SetSkinClassic()

描述

这些函数将切换至经典设计，但不会为指定小工具换肤。

原型

```
void <WIDGET>_SetSkinClassic(<WIDGET>_Handle hObj);
```

参数	描述
hObj	专用小工具的句柄。

其他信息

另请参阅函数 <WIDGET>_SetDefaultSkinClassic()。

<WIDGET>_SetSkinFlexProps()

描述

使用这些函数，可以更改默认皮肤的一些属性，但不会派生自己的皮肤。本章后面的小工具特定说明将详细介绍可更改的内容。

原型

```
void <WIDGET>_SetSkinFlexProps(const <WIDGET>_SKINFLEX_PROPS * pProps,
                               int Index);
```

参数	描述
pProps	类型 <WIDGET>_SKINFLEX_PROPS 的皮肤特定配置结构的指针。
Index	状态 （已按下、活动状态、已选定 ...）的详细信息，且有关详细信息将有效。

18.8 BUTTON_SKIN_FLEX

下图显示皮肤的详细信息：

按钮皮肤由圆角边框和矩形内部区域组成，其填充方法采用的是两色渐变。环绕边框采用 2 种颜色进行绘制。

详细信息	描述
A	顶部渐变的顶部颜色。
B	顶部渐变的底部颜色。
C	底部渐变的顶部颜色。
D	底部渐变的底部颜色。
E	环绕框架的外部颜色。
F	环绕框架的内部颜色。
G	环绕框架与内部矩形区之间部分的颜色。
R	圆角的半径。
T	可选文本。

18.8.1 配置结构

如果要设置皮肤的默认外观或在运行时更改外观，则请使用类型 BUTTON_SKINFLEX_PROPS 的配置结构：

BUTTON_SKINFLEX_PROPS 的元素

数据类型	元素	描述
U32	aColorFrame[3]	[0] - 环绕框架的外部颜色。 [1] - 环绕框架的内部颜色。 [2] - 框架与内部区域之间区域的颜色。
U32	aColorUpper[2]	[0] - 上层渐变的第一种 （上层）颜色。 [1] - 上层渐变的第二种 （下层）颜色。
U32	aColorLower[2]	[0] - 下层渐变的第一种 （上层）颜色。 [1] - 下层渐变的第二种 （下层）颜色。
int	Radius	圆角的半径。

18.8.2 配置选项

在 GUIConf.h 中设置上述类型的自定义配置结构，可以决定皮肤的默认外观。下表显示可用的配置选项：

宏	描述
BUTTON_SKINPROPS_PRESSED	定义已按下状态所使用的默认属性。
BUTTON_SKINPROPS_FOCUSSED	定义已聚焦状态所使用的默认属性。
BUTTON_SKINPROPS_ENABLED	定义启用状态所使用的默认属性。
BUTTON_SKINPROPS_DISABLED	定义禁用状态所使用的默认属性。

18.8.3 换肤 API

下表按字母顺序列出了可用的程序：

程序	描述
BUTTON_DrawSkinFlex()	BUTTON_SKIN_FLEX 的换肤回调函数。（本章开始部分已介绍）
BUTTON_GetSkinFlexProps()	返回指定按钮皮肤的属性。（本章开始部分已介绍）
BUTTON_SetDefaultSkin()	设置新按钮小工具所使用的默认皮肤。（本章开始部分已介绍）
BUTTON_SetDefaultSkinClassic()	将经典设计设置为新按钮小工具的默认设计。（本章开始部分已介绍）
BUTTON_SetSkin()	设置指定按钮小工具的皮肤。（本章开始部分已介绍）
BUTTON_SetSkinClassic()	设置指定按钮小工具的经典设计。（本章开始部分已介绍）
BUTTON_SetSkinFlexProps()	设置指定按钮皮肤的属性。

BUTTON_SetSkinFlexProps()

之前	之后
	

描述

使用该函数可更改皮肤的属性。

原型

```
void BUTTON_SetSkinFlexProps(const BUTTON_SKINFLEX_PROPS * pProps,
                             int Index);
```

参数	描述
pProps	类型 BUTTON_SKINFLEX_PROPS 的结构的指针。
Index	（参见下表）

参数 Index 的允许值	
BUTTON_SKINFLEX_PI_PRESSED	已按下状态的属性。
BUTTON_SKINFLEX_PI_FOCUSSED	已聚焦状态的属性。
BUTTON_SKINFLEX_PI_ENABLED	启用状态的属性。
BUTTON_SKINFLEX_PI_DISABLED	禁用状态的属性。

其他信息

该函数会将指针传递至 BUTTON_SKINFLEX_PROPS 结构。它可用于设置皮肤的颜色和半径。
使用函数 BUTTON_GetSkinFlexProps() 则可获取皮肤的当前属性。

18.8.4 命令列表

换肤程序会收到 WIDGET_ITEM_DRAW_INFO 结构的指针。此结构的 Cmd 成员包含需要处理的命令。下表显示传递至 BUTTON_SKIN_FLEX 回调函数的所有命令：

命令	描述
WIDGET_ITEM_CREATE	创建小工具后立即发送。
WIDGET_ITEM_DRAW_BACKGROUND	换肤函数将绘制背景。
WIDGET_ITEM_DRAW_BITMAP	换肤函数将绘制可选的按钮位图。
WIDGET_ITEM_DRAW_TEXT	换肤函数将绘制可选的按钮文本。

本章开始部分已介绍 WIDGET_ITEM_DRAW_INFO 结构。

WIDGET_ITEM_CREATE

如有必要，换肤程序将设置与皮肤相关的属性，如透明度或文本对齐方式。

WIDGET_ITEM_DRAW_BACKGROUND

将绘制小工具的背景。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	（参见下表）。
x0	窗口坐标中最左侧的坐标，通常为 0。
y0	窗口坐标中最顶端的坐标，通常为 0。
x1	窗口坐标中最右侧的坐标。
y1	窗口坐标中最底部的坐标。

元素 ItemIndex 的允许值	
BUTTON_SKINFLEX_PI_PRESSED	已按下小工具。
BUTTON_SKINFLEX_PI_FOCUSSED	未按下小工具但已将其聚焦。
BUTTON_SKINFLEX_PI_ENABLED	未聚焦小工具但已将其启用。
BUTTON_SKINFLEX_PI_DISABLED	已禁用小工具。

WIDGET_ITEM_DRAW_BITMAP

将绘制可选的按钮位图。

WIDGET_ITEM_DRAW_INFO 结构的内容

请参阅 WIDGET_ITEM_DRAW_BACKGROUND。

其他信息

使用函数 BUTTON_GetBitmap() 可获取可选的按钮位图。

WIDGET_ITEM_DRAW_TEXT

将绘制可选的按钮文本。

WIDGET_ITEM_DRAW_INFO 结构的内容

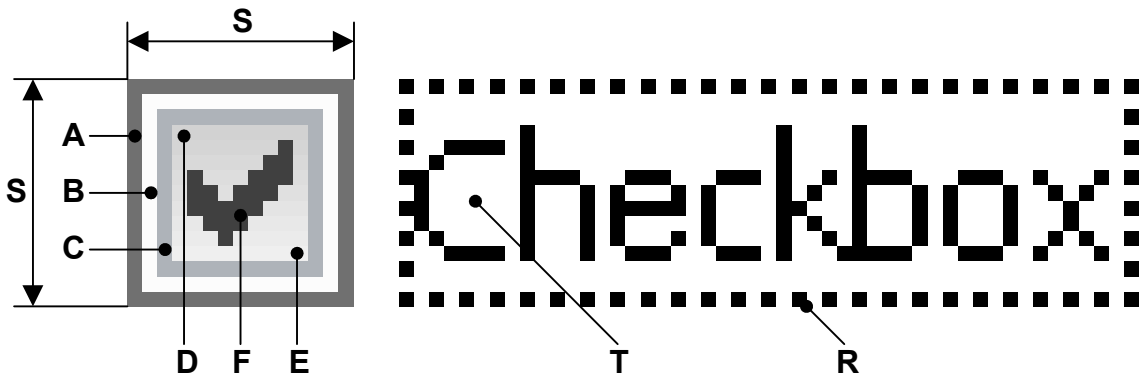
请参阅 WIDGET_ITEM_DRAW_BACKGROUND。

其他信息

使用函数 BUTTON_GetText() 可获取可选的文本。

18.9 CHECKBOX_SKIN_FLEX

下图显示皮肤的详细信息：



复选框皮肤的按钮区域由一个框架和矩形内部区域组成，其采用的填充方法为单色渐变。框架采用 3 种颜色进行绘制。如果选中，则方框中央将显示选中标记。

详细信息	描述
A	框架的第一种颜色。
B	框架的第二种颜色。
C	框架的第三种颜色。
D	渐变的上层颜色。
E	渐变的下层颜色。
F	选中标记的颜色。
R	聚焦框。
S	按钮区域的大小 （以像素为单位）。
T	可选文本。

18.9.1 配置结构

如果要设置皮肤的默认外观或在运行时更改外观，则请使用类型 CHECKBOX_SKINFLEX_PROPS 的配置结构：

CHECKBOX_SKINFLEX_PROPS 的元素

数据类型	元素	描述
U32	aColorFrame[3]	[0] - 框架的外部颜色。 [1] - 框架的中间颜色。 [2] - 框架的内部颜色。
U32	aColorInner[2]	[0] - 渐变的第一种 （上层）颜色。 [1] - 渐变的第二种 （下层）颜色。
U32	ColorCheck	选中标记的颜色。
int	Size	按钮区域的大小 （以像素为单位）。

18.9.2 配置选项

在 GUIConf.h 中设置上述类型的自定义配置结构，可以决定皮肤的默认外观。下表显示可用的配置选项：

宏	描述
CHECKBOX_SKINPROPS_ENABLED	定义启用状态所使用的默认属性。
CHECKBOX_SKINPROPS_DISABLED	定义禁用状态所使用的默认属性。

18.9.3 换肤 API

下表按字母顺序列出了可用的程序：

程序	描述
CHECKBOX_DrawSkinFlex()	CHECKBOX_SKIN_FLEX 的换肤回调函数。（本章开始部分已介绍）
CHECKBOX_GetSkinFlexProps()	返回指定复选框皮肤的属性。（本章开始部分已介绍）
CHECKBOX_SetDefaultSkin()	设置新创建的复选框小工具所使用的默认皮肤。（本章开始部分已介绍）
CHECKBOX_SetDefaultSkinClassic()	将经典设计设置为新创建的复选框小工具的默认设计。（本章开始部分已介绍）
CHECKBOX_SetSkin()	设置指定复选框小工具的皮肤。（本章开始部分已介绍）
CHECKBOX_SetSkinClassic()	设置指定复选框小工具的经典设计。（本章开始部分已介绍）
CHECKBOX_SetSkinFlexProps()	设置指定复选框皮肤的属性。

CHECKBOX_SetSkinFlexProps()

之前	之后
	

描述

使用该函数可更改皮肤的属性。

原型

```
void CHECKBOX_SetSkinFlexProps(const CHECKBOX_SKINFLEX_PROPS * pProps,
                               int Index);
```

参数	描述
pProps	类型 CHECKBOX_SKINFLEX_PROPS 的结构的指针。
Index	（参见下表）

参数 Index 的允许值	
CHECKBOX_SKINFLEX_PI_ENABLED	启用状态的属性。
CHECKBOX_SKINFLEX_PI_DISABLED	禁用状态的属性。

其他信息

该函数会将指针传递至 CHECKBOX_SKINFLEX_PROPS 结构。它可以用于设置皮肤的颜色。
需要注意的是，如果新按钮大小与旧按钮大小不同，则无法更改使用皮肤的小工具大小。无法通过皮肤更改小工具的大小，因为皮肤并不“知道”使用皮肤的小工具。如果需要，则应通过应用程序（如通过 WM_ResizeWindow()）来调整大小。
使用函数 CHECKBOX_GetSkinFlexProps() 则可获取皮肤的当前属性。

18.9.4 命令列表

换肤程序会收到 WIDGET_ITEM_DRAW_INFO 结构的指针。此结构的 Cmd 成员包含需要处理的命令。下表显示传递至 CHECKBOX_SKIN_FLEX 回调函数的所有命令：

命令	描述
WIDGET_ITEM_CREATE	创建小工具后立即发送。
WIDGET_ITEM_DRAW_BUTTON	将绘制按钮区域的背景。
WIDGET_ITEM_DRAW_BITMAP	将绘制按钮区域的选中标记。
WIDGET_ITEM_DRAW_FOCUS	将绘制聚焦框。
WIDGET_ITEM_DRAW_TEXT	将绘制可选文本。

WIDGET_ITEM_CREATE

如有必要，换肤程序将设置与皮肤相关的属性，如透明度或文本对齐方式。

WIDGET_ITEM_DRAW_BUTTON

将绘制无选中标记的小工具按钮区域。通常在小工具区域的左侧绘制按钮区域。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
x0	窗口坐标中小工具区域最左侧的坐标，通常为 0。
y0	窗口坐标中小工具区域最顶端的坐标，通常为 0。
x1	窗口坐标中小工具区域最右侧的坐标。
y1	窗口坐标中小工具区域最底部的坐标。

此皮肤所有命令的 hWin、 x0、 y0、 x1 和 y1 的内容均相同。

WIDGET_ITEM_DRAW_BITMAP

将在按钮区域的中心绘制选中标记。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
ItemIndex	1 - 已选中小工具。 2 - 在使用 3 种状态按钮时第二个选中的状态。
hWin,x0,y0,x1,y1	（请参阅 WIDGET_ITEM_DRAW_BUTTON）

WIDGET_ITEM_DRAW_FOCUS

将在文本周围显示聚焦框。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
p	无效指针指向小工具的零终止可选文本。
hWin,x0,y0,x1,y1	（请参阅 WIDGET_ITEM_DRAW_BUTTON）

其他信息

元素 p 可以转换为文本指针。有关详情，请参阅 WIDGET_ITEM_DRAW_TEXT。

WIDGET_ITEM_DRAW_TEXT

将绘制可选文本。通常在按钮区域的右侧绘制文本。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
p	无效指针指向小工具的零终止可选文本。
hWin,x0,y0,x1,y1	（请参阅 WIDGET_ITEM_DRAW_BUTTON）

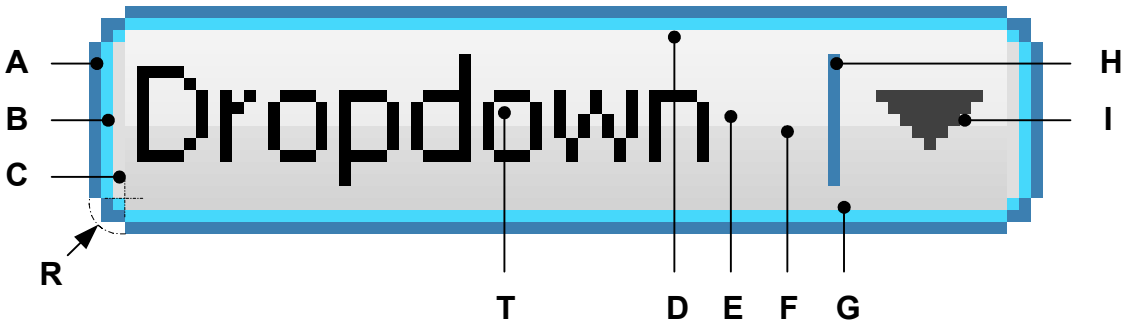
其他信息

如果要获取文本指针，则元素 p 可以转换为文本指针：

```
char * s;
s = (char *)pDrawItemInfo->p;
GUI_DispString(s);
```

18.10 DROPDOWN_SKIN_FLEX

下图显示皮肤的详细信息：



下拉皮肤由圆角框架和矩形内部区域组成，其填充方法采用的是两色渐变。圆角框架采用 3 种颜色进行绘制。将在右侧绘制一个小三角形。将在文本和三角形之间绘制小分隔符：

详细信息	描述
A	框架的第一种颜色。
B	框架的第二种颜色。
C	框架的第三种颜色。
D	顶部渐变的顶部颜色。
E	顶部渐变的底部颜色。
F	底部渐变的顶部颜色。
G	底部渐变的底部颜色。
H	文本与三角形之间的分隔符。
I	三角形。
R	圆角的半径。
T	可选文本。

处于开启状态的下拉小工具包含一个额外的列表框。需要注意的是，此列表框不会受皮肤的影响。

18.10.1 配置结构

如果要设置皮肤的默认外观或在运行时更改外观，则请使用类型 DROPDOWN_SKINFLEX_PROPS 的配置结构：

DROPDOWN_SKINFLEX_PROPS 的元素

数据类型	元素	描述
U32	aColorFrame[3]	[0] - 环绕框架的外部颜色。 [1] - 环绕框架的内部颜色。 [2] - 框架与内部区域之间区域的颜色。
U32	aColorUpper[2]	[0] - 顶部渐变的顶部颜色。 [1] - 顶部渐变的底部颜色。
U32	aColorLower[2]	[0] - 底部渐变的顶部颜色。 [1] - 底部渐变的底部颜色。
U32	ColorArrow	绘制箭头所使用的颜色。
U32	ColorText	绘制文本所使用的颜色。
U32	ColorSep	绘制分隔符所使用的颜色。
int	Radius	圆角的半径。

18.10.2 配置选项

在 GUIConf.h 中设置上述类型的自定义配置结构，可以决定皮肤的默认外观。下表显示可用的配置选项：

宏	描述
DROPDOWN_SKINPROPS_OPEN	定义开启状态所使用的默认属性。
DROPDOWN_SKINPROPS_FOCUSSED	定义已聚焦状态所使用的默认属性。
DROPDOWN_SKINPROPS_ENABLED	定义启用状态所使用的默认属性。
DROPDOWN_SKINPROPS_DISABLED	定义禁用状态所使用的默认属性。

18.10.3 换肤 API

下表按字母顺序列出了可用的程序：

程序	描述
DROPDOWN_DrawSkinFlex()	DROPDOWN_SKIN_FLEX 的换肤回调函数。（本章开始部分已介绍）
DROPDOWN_GetSkinFlexProps()	返回指定下拉皮肤的属性。（本章开始部分已介绍）
DROPDOWN_SetDefaultSkin()	设置新创建的下拉小工具所使用的默认皮肤。（本章开始部分已介绍）
DROPDOWN_SetDefaultSkinClassic()	将经典设计设置为新创建的下拉小工具的默认设计。（本章开始部分已介绍）
DROPDOWN_SetSkin()	设置指定下拉小工具的皮肤。（本章开始部分已介绍）
DROPDOWN_SetSkinClassic()	设置指定下拉小工具的经典设计。（本章开始部分已介绍）
DROPDOWN_SetSkinFlexProps()	设置指定下拉皮肤的属性。

DROPDOWN_SetSkinFlexProps()

之前	之后
	

描述

使用该函数可更改皮肤的属性。

原型

```
void DROPDOWN_SetSkinFlexProps(const DROPDOWN_SKINFLEX_PROPS * pProps,
                               int Index);
```

参数	描述
pProps	类型 DROPDOWN_SKINFLEX_PROPS 的结构的指针。
Index	（参见下表）

参数 Index 的允许值	
DROPDOWN_SKINFLEX_PI_OPEN	开启状态的属性。
DROPDOWN_SKINFLEX_PI_FOCUSSED	已聚焦状态的属性。
DROPDOWN_SKINFLEX_PI_ENABLED	启用状态的属性。
DROPDOWN_SKINFLEX_PI_DISABLED	禁用状态的属性。

其他信息

该函数会将指针传递至 DROPDOWN_SKINFLEX_PROPS 结构。它可用于设置皮肤的颜色和半径。使用函数 DROPDOWN_GetSkinFlexProps() 则可获取皮肤的当前属性。

18.10.4 命令列表

换肤程序会收到 WIDGET_ITEM_DRAW_INFO 结构的指针。此结构的 Cmd 成员包含需要处理的命令。下表显示传递至 DROPDOWN_SKIN_FLEX 回调函数的所有命令：

命令	描述
WIDGET_ITEM_CREATE	创建小工具后立即发送。
WIDGET_ITEM_DRAW_ARROW	换肤函数将绘制箭头。
WIDGET_ITEM_DRAW_BACKGROUND	换肤函数将绘制背景。
WIDGET_ITEM_DRAW_TEXT	换肤函数将绘制可选的按钮文本。

WIDGET_ITEM_CREATE

如有必要，换肤程序将设置与皮肤相关的属性，如透明度或文本对齐方式。

WIDGET_ITEM_DRAW_ARROW

将在右侧绘制三角形 （箭头）。

WIDGET_ITEM_DRAW_INFO 结构的内容：

（请参阅 WIDGET_ITEM_DRAW_BACKGROUND）

WIDGET_ITEM_DRAW_BACKGROUND

将绘制小工具的背景。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	（参见下表）。
x0	窗口坐标中最左侧的坐标，通常为 0。
y0	窗口坐标中最顶端的坐标，通常为 0。
x1	窗口坐标中最右侧的坐标。
y1	窗口坐标中最底部的坐标。

元素 ItemIndex 的允许值	
DROPDOWN_SKINFLEX_PI_EXPANDED	小工具已展开。
DROPDOWN_SKINFLEX_PI_FOCUSSED	未按下小工具但已将其聚焦。
DROPDOWN_SKINFLEX_PI_ENABLED	未聚焦小工具但已将其启用。
DROPDOWN_SKINFLEX_PI_DISABLED	已禁用小工具。

WIDGET_ITEM_DRAW_TEXT

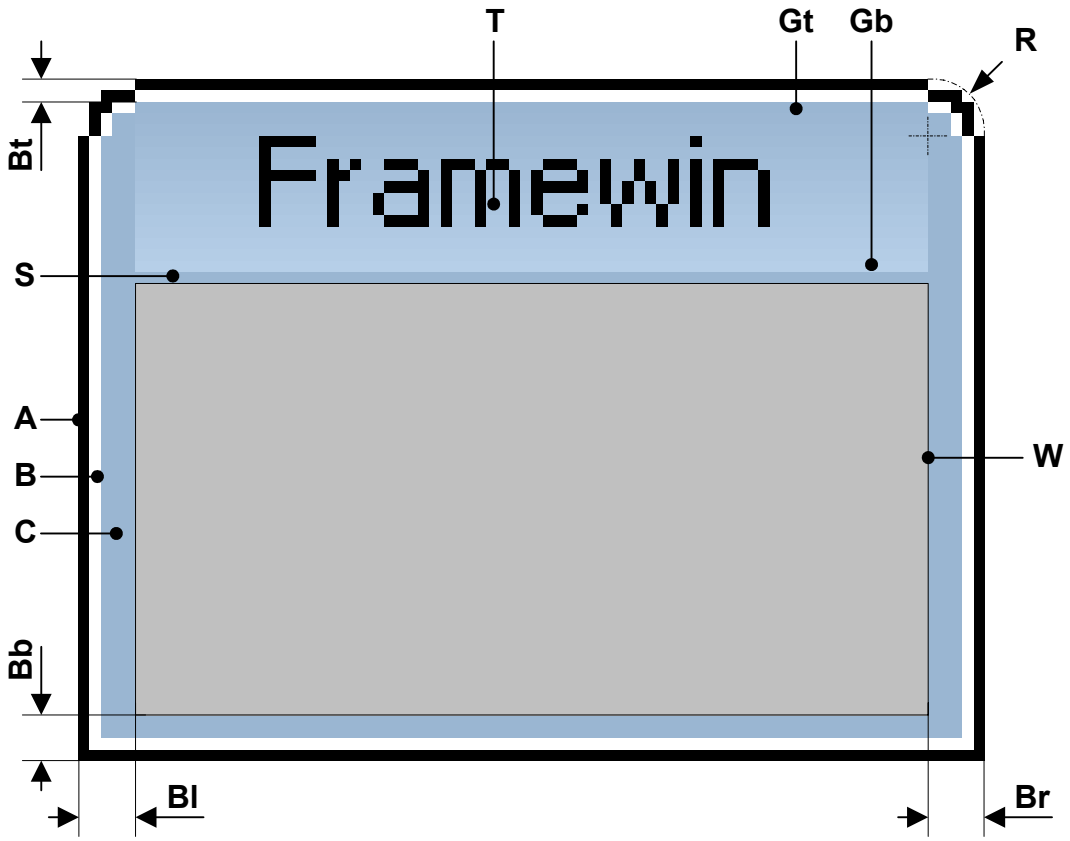
将在下拉小工具的按钮区域绘制当前所选字符串的文本。通常在按钮区域的左侧绘制文本。

WIDGET_ITEM_DRAW_INFO 结构的内容：

（请参阅 WIDGET_ITEM_DRAW_BACKGROUND）

18.11 FRAMEWIN_SKIN_FLEX

下图显示皮肤的详细信息：



上图显示 Framewin 皮肤的详细信息。它包括标题栏、顶部的圆角、绘制标题栏背景所使用的渐变、大小可配置的边框以及标题栏与工作区之间的分隔符：

详细信息	描述
A	环绕框架的外部颜色。
B	环绕框架的内部颜色。
C	框架与内部区域之间区域的颜色。
Gt	顶部标题栏渐变的顶部颜色。
Gb	标题栏渐变的底部颜色。
Bt	边框的顶部大小。
Bb	边框的底部大小。
Bl	边框的左侧大小。
Br	边框的右侧大小。
W	客户端窗口区域。
R	圆角的半径。
T	可选文本。

18.11.1 配置结构

如果要设置皮肤的默认外观或在运行时更改外观，则请使用类型 FRAMEWIN_SKINFLEX_PROPS 的配置结构：

FRAMEWIN_SKINFLEX_PROPS 的元素

数据类型	元素	描述
U32	aColorFrame[3]	[0] - 环绕框架的外部颜色。 [1] - 环绕框架的内部颜色。 [2] - 框架与内部区域之间区域的颜色。
U32	aColorTitle[2]	[0] - 顶部标题栏渐变的顶部颜色。 [1] - 标题栏渐变的底部颜色。
int	Radius	圆角的半径。
int	SpaceX	X 轴中标题文本与标题渐变边框之间的可选空间。
int	BorderSizeL	边框的左侧大小。
int	BorderSizeR	边框的右侧大小。
int	BorderSizeT	边框的顶部大小。
int	BorderSizeB	边框的底部大小。

18.11.2 配置选项

在 GUIConf.h 中设置上述类型的自定义配置结构，可以决定皮肤的默认外观。下表显示可用的配置选项：

宏	描述
FRAMEWIN_SKINPROPS_ACTIVE	定义活动状态所使用的默认属性。
FRAMEWIN_SKINPROPS_INACTIVE	定义非活动状态所使用的默认属性。

18.11.3 换肤 API

下表按字母顺序列出了可用的程序：

程序	描述
FRAMEWIN_DrawSkinFlex()	FRAMEWIN_SKIN_FLEX 的换肤回调函数。（本章开始部分已介绍）
FRAMEWIN_GetSkinFlexProps()	返回指定 FRAMEWIN 皮肤的属性。（本章开始部分已介绍）
FRAMEWIN_SetDefaultSkin()	设置新创建的 Framewin 小工具所使用的默认皮肤。（本章开始部分已介绍）
FRAMEWIN_SetDefaultSkinClassic()	将经典设计设置为新 Framewin 小工具的默认设计。（本章开始部分已介绍）
FRAMEWIN_SetSkin()	设置指定 Framewin 小工具的皮肤。（本章开始部分已介绍）
FRAMEWIN_SetSkinClassic()	设置指定 Framewin 小工具的经典设计。（本章开始部分已介绍）
FRAMEWIN_SetSkinFlexProps()	设置指定 Framewin 皮肤的属性。

FRAMEWIN_SetSkinFlexProps()

之前	之后
	

描述

使用该函数可更改皮肤的属性。

原型

```
void FRAMEWIN_SetSkinFlexProps(const FRAMEWIN_SKINFLEX_PROPS * pProps,
                               int Index);
```

参数	描述
pProps	类型 FRAMEWIN_SKINFLEX_PROPS 的结构的指针。
Index	（参见下表）

参数 Index 的允许值	
FRAMEWIN_SKINFLEX_PI_ACTIVE	活动状态的属性。
FRAMEWIN_SKINFLEX_PI_INACTIVE	非活动状态的属性。

其他信息

该函数会将指针传递至 FRAMEWIN_SKINFLEX_PROPS 结构。它可以用于设置皮肤的颜色、半径和边框大小。
使用函数 FRAMEWIN_GetSkinFlexProps() 则可获取皮肤的当前属性。
在使用此皮肤创建框架窗口时，非活动状态的值将用于计算客户端窗口的大小和位置。

18.11.4 命令列表

换肤程序会收到 WIDGET_ITEM_DRAW_INFO 结构的指针。此结构的 Cmd 成员包含需要处理的命令。下表显示传递至 FRAMEWIN_SKIN_FLEX 回调函数的所有命令：

命令	描述
WIDGET_ITEM_CREATE	创建小工具后立即发送。
WIDGET_ITEM_DRAW_BACKGROUND	换肤函数将绘制标题背景。
WIDGET_ITEM_DRAW_FRAME	换肤函数将绘制框架。
WIDGET_ITEM_DRAW_SEP	换肤函数将绘制分隔符。
WIDGET_ITEM_DRAW_TEXT	换肤函数将绘制标题文本。
WIDGET_ITEM_GET_BORDERSIZE_L	换肤函数将返回左边框大小。
WIDGET_ITEM_GET_BORDERSIZE_R	换肤函数将返回右边框大小。
WIDGET_ITEM_GET_BORDERSIZE_T	换肤函数将返回上边框大小。
WIDGET_ITEM_GET_BORDERSIZE_B	换肤函数将返回下边框大小。
WIDGET_ITEM_GET_RADIUS	换肤函数将返回圆角的半径。

WIDGET_ITEM_CREATE

如有必要，换肤程序将设置与皮肤相关的属性，如透明度或文本对齐方式。

WIDGET_ITEM_DRAW_BACKGROUND

换肤程序将绘制标题区域的背景。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	（参见下表）。
x0	窗口坐标中标题区域最左侧的坐标。
y0	窗口坐标中标题区域最顶端的坐标。
x1	窗口坐标中标题区域最右侧的坐标。
y1	窗口坐标中标题区域最底部的坐标。

元素 ItemIndex 的允许值	
FRAMEWIN_SKINFLEX_PI_ACTIVE	小工具处于活动状态。
FRAMEWIN_SKINFLEX_PI_INACTIVE	小工具处于非活动状态。

WIDGET_ITEM_DRAW_FRAME

换肤程序将绘制整个边框，但不包括标题区域和分隔符。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	（参见下表）。
x0	窗口坐标中最左侧的坐标，通常为 0。
y0	窗口坐标中最顶端的坐标，通常为 0。
x1	窗口坐标中最右侧的坐标 （窗口的 x 轴大小 - 1）。
y1	窗口坐标中最右侧的坐标 （窗口的 y 轴大小 - 1）。

元素 ItemIndex 的允许值	
FRAMEWIN_SKINFLEX_PI_ACTIVE	小工具处于活动状态。
FRAMEWIN_SKINFLEX_PI_INACTIVE	小工具处于非活动状态。

WIDGET_ITEM_DRAW_SEP

换肤程序将在标题区域与客户端窗口之间绘制分隔符。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	（参见下表）。
x0	窗口坐标中分隔符最左侧的坐标。
y0	窗口坐标中分隔符最顶端的坐标。
x1	窗口坐标中分隔符最右侧的坐标。
y1	窗口坐标中分隔符最底部的坐标。

元素 ItemIndex 的允许值	
FRAMEWIN_SKINFLEX_PI_ACTIVE	小工具处于活动状态。
FRAMEWIN_SKINFLEX_PI_INACTIVE	小工具处于非活动状态。

WIDGET_ITEM_DRAW_TEXT

换肤程序将绘制标题文本。

WIDGET_ITEM_DRAW_INFO 结构的内容：

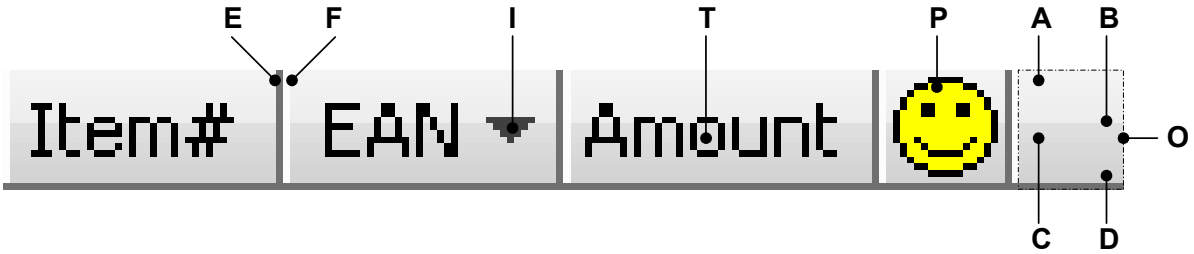
（请参阅 WIDGET_ITEM_DRAW_BACKGROUND）

WIDGET_ITEM_GET_BORDERSIZE_L、WIDGET_ITEM_GET_BORDERSIZE_R、WIDGET_ITEM_GET_BORDERSIZE_T、WIDGET_ITEM_GET_BORDERSIZE_B

换肤程序将返回相应边框的大小。

18.12 HEADER_SKIN_FLEX

下图显示皮肤的详细信息：



上图显示皮肤的详细信息。它包括带窄边框的栏，其中分为不同的项目。该栏的背景由顶部和底部渐变组成。每个项目都可能具有文本、位图及可用于显示排序顺序的指示器：

详细信息	描述
A	顶部渐变的顶部颜色。
B	顶部渐变的底部颜色。
C	底部渐变的顶部颜色。
D	底部渐变的底部颜色。
E	框架的第一种颜色。
F	框架的第二种颜色。
I	指示器。
T	文本 （可选）。
P	位图 （可选）。

18.12.1 配置结构

如果要设置皮肤的默认外观或在运行时更改外观，则请使用类型 HEADER_SKINFLEX_PROPS 的配置结构：

HEADER_SKINFLEX_PROPS 的元素

数据类型	元素	描述
U32	aColorFrame[2]	[0] - 框架和分隔符的第一种颜色。 [1] - 框架和分隔符的第二种颜色。
U32	aColorUpper[2]	[0] - 顶部渐变的顶部颜色。 [1] - 顶部渐变的底部颜色。
U32	aColorLower[2]	[0] - 底部渐变的顶部颜色。 [1] - 底部渐变的底部颜色。
U32	ColorArrow	指示器的颜色。

18.12.2 配置选项

在 GUIConf.h 中设置上述类型的自定义配置结构，可以决定皮肤的默认外观。下表显示可用的配置选项：

宏	描述
HEADER_SKINPROPS	定义绘制皮肤所使用的默认属性。

18.12.3 换肤 API

下表按字母顺序列出了可用的程序：

程序	描述
HEADER_DrawSkinFlex()	HEADER_SKIN_FLEX 的换肤回调函数。（本章开始部分已介绍）
HEADER_GetSkinFlexProps()	返回指定 HEADER 皮肤的属性。（本章开始部分已介绍）
HEADER_SetDefaultSkin()	设置新创建的 HEADER 小工具所使用的默认皮肤。（本章开始部分已介绍）
HEADER_SetDefaultSkinClassic()	将经典设计设置为新创建的 HEADER 小工具的默认设计。（本章开始部分已介绍）
HEADER_SetSkin()	设置指定 HEADER 小工具的皮肤。（本章开始部分已介绍）
HEADER_SetSkinClassic()	设置指定 HEADER 小工具的经典设计。（本章开始部分已介绍）
HEADER_SetSkinFlexProps()	设置指定 HEADER 皮肤的属性。

HEADER_SetSkinFlexProps()

之前	之后
	

描述

使用该函数可更改皮肤的属性。

原型

```
void HEADER_SetSkinFlexProps(const HEADER_SKINFLEX_PROPS * pProps,
                             int Index);
```

参数	描述
pProps	类型 HEADER_SKINFLEX_PROPS 的结构的指针。
Index	将为 0。

其他信息

该函数会将指针传递至 HEADER_SKINFLEX_PROPS 结构。它可以用于设置皮肤的颜色。
使用函数 HEADER_GetSkinFlexProps() 则可获取皮肤的当前属性。

18.12.4 命令列表

换肤程序会收到 WIDGET_ITEM_DRAW_INFO 结构的指针。此结构的 Cmd 成员包含需要处理的命令。
下表显示传递至 HEADER_SKIN_FLEX 回调函数的所有命令：

命令	描述
WIDGET_ITEM_CREATE	创建小工具后立即发送。
WIDGET_ITEM_DRAW_ARROW	将绘制标头项目的指示器箭头。
WIDGET_ITEM_DRAW_BACKGROUND	将绘制标头项目的背景。
WIDGET_ITEM_DRAW_BITMAP	将绘制标头项目的位图。
WIDGET_ITEM_DRAW_OVERLAP	将绘制小工具的重叠区域。
WIDGET_ITEM_DRAW_TEXT	将绘制标头项目的测试。

WIDGET_ITEM_CREATE

如有必要，换肤程序将设置与皮肤相关的属性，如透明度或文本对齐方式。

WIDGET_ITEM_DRAW_ARROW

换肤程序将绘制可选的方向指示器。只有标头项目的指示器启用时，才会发送消息。

WIDGET_ITEM_DRAW_INFO 结构的内容：

（请参阅 WIDGET_ITEM_DRAW_BACKGROUND）

WIDGET_ITEM_DRAW_BACKGROUND

换肤程序将绘制项目区域的背景。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	始终为 0。
x0	窗口坐标中项目区域最左侧的坐标。
y0	窗口坐标中项目区域最顶端的坐标。
x1	窗口坐标中项目区域最右侧的坐标。
y1	窗口坐标中项目区域最底部的坐标。

WIDGET_ITEM_DRAW_BITMAP

换肤程序将绘制可选的项目位图。只有存在位图时，才会发送消息。

WIDGET_ITEM_DRAW_INFO 结构的内容：

（请参阅 WIDGET_ITEM_DRAW_BACKGROUND）

WIDGET_ITEM_DRAW_OVERLAP

换肤程序将绘制重叠区域。

WIDGET_ITEM_DRAW_INFO 结构的内容：

（请参阅 WIDGET_ITEM_DRAW_BACKGROUND）

WIDGET_ITEM_DRAW_TEXT

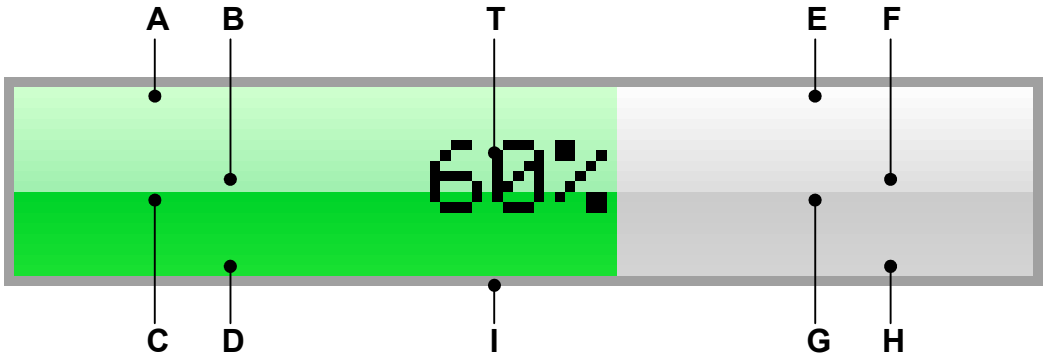
换肤程序将绘制可选的项目文本。只有存在文本时，才会发送消息。

WIDGET_ITEM_DRAW_INFO 结构的内容：

（请参阅 WIDGET_ITEM_DRAW_BACKGROUND）

18.13 PROGBAR_SKIN_FLEX

下图显示皮肤的详细信息：



上图显示皮肤的详细信息。它包括带窄边框的栏。绘制背景采用的是四色渐变的方法（即左右两侧的顶部与底部渐变）以及根据默认设置显示当前状态的文本。

详细信息	描述
A	左上方渐变的顶部颜色。
B	左上方渐变的底部颜色。
C	左下方渐变的顶部颜色。
D	左下方渐变的底部颜色。
A	右上方渐变的顶部颜色。
B	右上方渐变的底部颜色。
C	右下方渐变的顶部颜色。
D	右下方渐变的底部颜色。
I	框架的颜色。
T	文本（可选）。

18.13.1 配置结构

如果要设置皮肤的默认外观或在运行时更改外观，则请使用类型 PROGBAR_SKINFLEX_PROPS 的配置结构：

PROGBAR_SKINFLEX_PROPS 的元素

数据类型	元素	描述
U32	aColorUpperL[2]	[0] - 顶部渐变的顶部颜色。 [1] - 顶部渐变的底部颜色。
U32	aColorLowerL[2]	[0] - 底部渐变的顶部颜色。 [1] - 底部渐变的底部颜色。
U32	aColorUpperR[2]	[0] - 顶部渐变的顶部颜色。 [1] - 顶部渐变的底部颜色。
U32	aColorLowerR[2]	[0] - 底部渐变的顶部颜色。 [1] - 底部渐变的底部颜色。
U32	ColorFrame	框架的颜色。
U32	ColorText	文本的颜色。

18.13.2 配置选项

在 GUIConf.h 中设置上述类型的自定义配置结构，可以决定皮肤的默认外观。下表显示可用的配置选项：

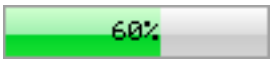
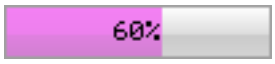
宏	描述
PROGBAR_SKINPROPS	定义绘制皮肤所使用的默认属性。

18.13.3 换肤 API

下表按字母顺序列出了可用的程序：

程序	描述
PROGBAR_DrawSkinFlex()	PROGBAR_SKIN_FLEX 的换肤回调函数。（本章开始部分已介绍）
PROGBAR_GetSkinFlexProps()	返回指定 PROGBAR 皮肤的属性。（本章开始部分已介绍）
PROGBAR_SetDefaultSkin()	设置新创建的 PROGBAR 小工具所使用的默认皮肤。（本章开始部分已介绍）
PROGBAR_SetDefaultSkinClassic()	将经典设计设置为新创建的 PROGBAR 小工具的默认设计。（本章开始部分已介绍）
PROGBAR_SetSkin()	设置指定 PROGBAR 小工具的皮肤。（本章开始部分已介绍）
PROGBAR_SetSkinClassic()	设置指定 PROGBAR 小工具的经典设计。（本章开始部分已介绍）
PROGBAR_SetSkinFlexProps()	设置指定 PROGBAR 皮肤的属性。

PROGBAR_SetSkinFlexProps()

之前	之后
	

描述

使用该函数可更改皮肤的颜色。

原型

```
void PROGBAR_SetSkinFlexProps(const PROGBAR_SKINFLEX_PROPS * pProps,
                              int Index);
```

参数	描述
pProps	类型 PROGBAR_SKINFLEX_PROPS 的结构的指针。
Index	将为 0。

其他信息

该函数会将指针传递至 PROGBAR_SKINFLEX_PROPS 结构。它可以用于设置皮肤的颜色。
使用函数 PROGBAR_GetSkinFlexProps() 可获取皮肤的当前属性。

18.13.4 命令列表

换肤程序会收到 WIDGET_ITEM_DRAW_INFO 结构的指针。此结构的 Cmd 成员包含需要处理的命令。
下表显示传递至 PROGBAR_SKIN_FLEX 回调函数的所有命令：

命令	描述
WIDGET_ITEM_CREATE	创建小工具后立即发送。
WIDGET_ITEM_DRAW_BACKGROUND	换肤函数将绘制背景。
WIDGET_ITEM_DRAW_FRAME	换肤函数将绘制框架。
WIDGET_ITEM_DRAW_TEXT	换肤函数将绘制文本。

WIDGET_ITEM_CREATE

如有必要，换肤程序将设置与皮肤相关的属性，如透明度或文本对齐方式。

WIDGET_ITEM_DRAW_BACKGROUND

换肤程序将绘制背景。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	始终为 0。
x0	窗口坐标中小工具区域最左侧的坐标。
y0	窗口坐标中小工具区域最顶端的坐标。
x1	窗口坐标中小工具区域最右侧的坐标。
y1	窗口坐标中小工具区域最底部的坐标。
p	PROGBAR_SKINFLEX_INFO 结构的指针。

PROGBAR_SKINFLEX_INFO 的元素

数据类型	元素	描述
int	IsVertical	如果为水平进度条，则为 0；如果为垂直进度条，则为 1。
int	Index	（参见下表）
const char *	pText	要绘制的文本的指针。

元素 Index 的允许值	
PROGBAR_SKINFLEX_L	水平进度条：将绘制左侧部分。 垂直进度条：将绘制顶部部分。
PROGBAR_SKINFLEX_R	水平进度条：将绘制右侧部分。 垂直进度条：将绘制底部部分。

其他信息

消息将发送两次，一次针对进度条的左侧/顶部部分，一次针对右侧/底部部分。PROGBAR_SKINFLEX_INFO 结构中 p 元素指出的 WIDGET_ITEM_DRAW_INFO 结构中的信息可以用于获取将绘制内容的信息。WIDGET_ITEM_DRAW_INFO 结构的参数 x0、y0、x1 和 y1 只会标记将绘制的区域：左侧 / 右侧或顶部 / 底部。

WIDGET_ITEM_DRAW_FRAME

换肤程序将绘制环绕框架。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	始终为 0。
x0	窗口坐标中小工具区域最左侧的坐标。
y0	窗口坐标中小工具区域最顶端的坐标。
x1	窗口坐标中小工具区域最右侧的坐标。
y1	窗口坐标中小工具区域最底部的坐标。

WIDGET_ITEM_DRAW_TEXT

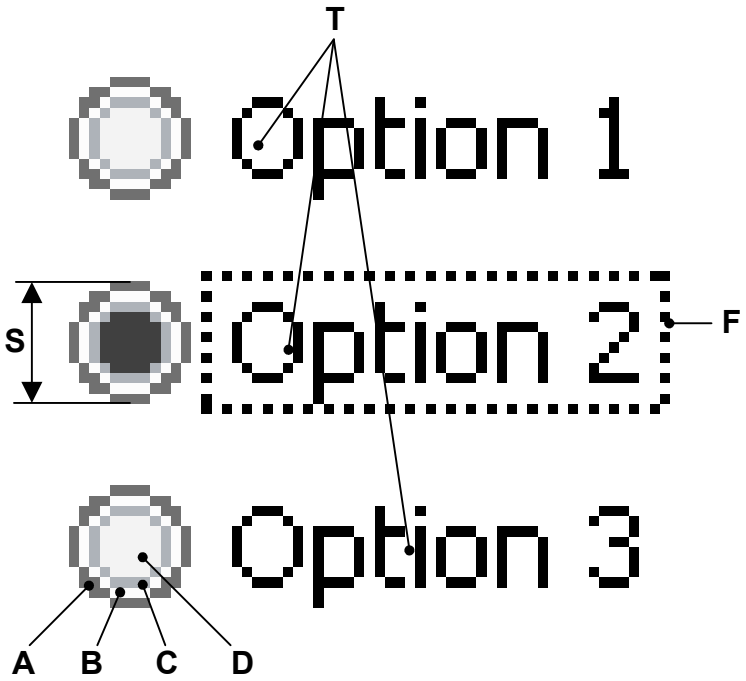
换肤程序将绘制文本。

WIDGET_ITEM_DRAW_INFO 结构的内容：

（请参阅 WIDGET_ITEM_DRAW_FRAME）

18.14 RADIO_SKIN_FLEX

下图显示皮肤的详细信息：



上图显示皮肤的详细信息。它由可配置按钮和每个项目的文本组成。如果小工具具有输入焦点，则当前所选项目文本将由聚焦框环绕：

详细信息	描述
A	按钮框架的外部颜色。
B	按钮框架的中间颜色。
C	按钮框架的内部颜色。
D	按钮的内部颜色。
F	聚焦框。
S	按钮的大小。
T	项目文本。

18.14.1 配置结构

如果要设置皮肤的默认外观或在运行时更改外观，则请使用类型 `RADIO_SKINFLEX_PROPS` 的配置结构：

RADIO_SKINFLEX_PROPS 的元素

数据类型	元素	描述
U32	aColorButton[4]	[0] - 按钮框架的外部颜色。 [1] - 按钮框架的中间颜色。 [2] - 按钮框架的内部颜色。 [3] - 按钮的内部颜色。
int	ButtonSize	按钮的大小 （以像素为单位）。

18.14.2 配置选项

在 GUIConf.h 中设置上述类型的自定义配置结构，可以决定皮肤的默认外观。下表显示可用的配置选项：

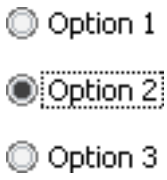
宏	描述
RADIO_SKINPROPS_CHECKED	定义已选中状态所使用的默认属性。
RADIO_SKINPROPS_UNCHECKED	定义未选中状态所使用的默认属性。

18.14.3 换肤 API

下表按字母顺序列出了可用的程序：

程序	描述
RADIO_DrawSkinFlex()	RADIO_SKIN_FLEX 的换肤回调函数。（本章开始部分已介绍）
RADIO_GetSkinFlexProps()	返回指定 RADIO 皮肤的属性。（本章开始部分已介绍）
RADIO_SetDefaultSkin()	设置新创建的 RADIO 小工具所使用的默认皮肤。（本章开始部分已介绍）
RADIO_SetDefaultSkinClassic()	将经典设计设置为新创建的 RADIO 小工具的默认设计。（本章开始部分已介绍）
RADIO_SetSkin()	设置指定 RADIO 小工具的皮肤。（本章开始部分已介绍）
RADIO_SetSkinClassic()	设置指定 RADIO 小工具的经典设计。（本章开始部分已介绍）
RADIO_SetSkinFlexProps()	设置指定 RADIO 皮肤的属性。

RADIO_SetSkinFlexProps()

之前	之后
	

描述

此函数可用于更改皮肤的颜色及按钮的大小。

原型

```
void RADIO_SetSkinFlexProps(const RADIO_SKINFLEX_PROPS * pProps,
                            int Index);
```

参数	描述
pProps	类型 RADIO_SKINFLEX_PROPS 的结构的指针。
索引	将为 0。

其他信息

该函数会将指针传递至 RADIO_SKINFLEX_PROPS 结构。它可以用于设置皮肤的颜色和按钮大小。使用函数 RADIO_GetSkinFlexProps() 可获取皮肤的当前属性。

18.14.4 命令列表

换肤程序会收到 WIDGET_ITEM_DRAW_INFO 结构的指针。此结构的 Cmd 成员包含需要处理的命令。下表显示传递至 RADIO_SKIN_FLEX 回调函数的所有命令：

命令	描述
WIDGET_ITEM_CREATE	创建小工具后立即发送。
WIDGET_ITEM_DRAW_BUTTON	换肤函数将绘制项目的按钮。
WIDGET_ITEM_DRAW_FOCUS	换肤函数将绘制聚焦框。
WIDGET_ITEM_DRAW_TEXT	换肤函数将绘制项目的文本。
WIDGET_ITEM_GET_BUTTONSIZE	换肤函数将返回按钮大小。

WIDGET_ITEM_CREATE

如有必要，换肤程序将设置与皮肤相关的属性，如透明度或文本对齐方式。

WIDGET_ITEM_DRAW_BUTTON

换肤程序将绘制项目的按钮。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	要绘制的项目的索引。
x0	窗口坐标中按钮区域最左侧的坐标。
y0	窗口坐标中按钮区域最顶端的坐标。
x1	窗口坐标中按钮区域最右侧的坐标。
y1	窗口坐标中按钮区域最底部的坐标。

WIDGET_ITEM_DRAW_FOCUS

换肤程序将在当前所选项目的文本周围绘制聚焦框。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	要绘制的项目的索引。
x0	窗口坐标中聚焦框最左侧的坐标。
y0	窗口坐标中聚焦框最顶端的坐标。
x1	窗口坐标中聚焦框最右侧的坐标。
y1	窗口坐标中聚焦框最底部的坐标。

其他信息

x0、 y0、 x1 和 y1 中指定的矩形区域将考虑字体设置和项目文本。

WIDGET_ITEM_DRAW_TEXT

换肤程序将绘制项目的文本。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	要绘制的项目的索引。
x0	窗口坐标中文本区域最左侧的坐标。
y0	窗口坐标中文本区域最顶端的坐标。
x1	窗口坐标中文本区域最右侧的坐标。
y1	窗口坐标中文本区域最底部的坐标。

其他信息

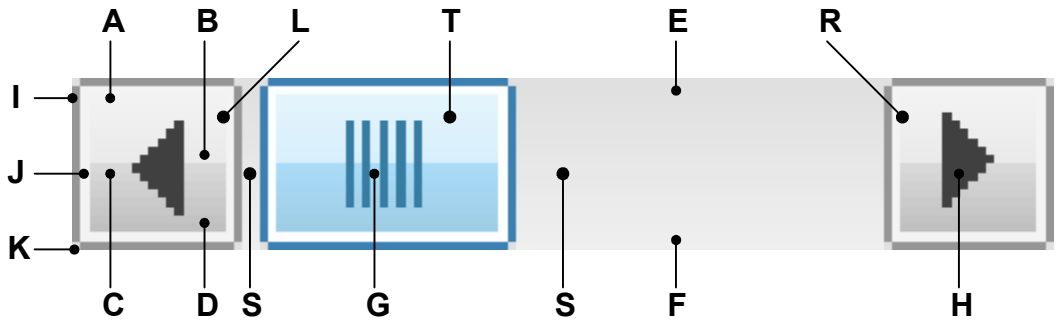
x0、 y0、 x1 和 y1 中指定的矩形区域将考虑字体设置和项目文本。

WIDGET_ITEM_GET_BUTTONSIZE

换肤程序将返回按钮大小。

18.15 SCROLLBAR_SKIN_FLEX

下图显示皮肤的详细信息：



上图显示皮肤的详细信息。它包括带箭头的一个左侧按钮和一个右侧按钮、轴区域及带抓取工具的缩略图：

详细信息	描述
A	顶部渐变的顶部颜色。
B	顶部渐变的底部颜色。
C	底部渐变的顶部颜色。
D	底部渐变的底部颜色。
E	轴渐变的顶部颜色。
F	轴渐变的底部颜色。
G	缩略图区域的抓取工具。
H	按钮箭头。
I	外部框架颜色。
J	内部框架颜色。
K	框架边缘的颜色。
L	左侧按钮。
T	缩略图。
R	右侧按钮。
S	轴区域。

18.15.1 配置结构

如果要设置皮肤的默认外观或在运行时更改外观，则请使用类型 SCROLLBAR_SKINFLEX_PROPS 的配置结构：

RADIO_SKINFLEX_PROPS 的元素

数据类型	元素	描述
U32	aColorFrame[3]	[0] - 外部框架颜色。 [1] - 内部框架颜色。 [2] - 框架边缘的颜色。
U32	aColorUpper[2]	[0] - 顶部渐变的顶部颜色。 [1] - 顶部渐变的底部颜色。
U32	aColorLower[2]	[0] - 底部渐变的顶部颜色。 [1] - 底部渐变的底部颜色。
U32	aColorShaft[2]	[0] - 轴渐变的顶部颜色。 [1] - 轴渐变的底部颜色。
U32	ColorArrow	按钮箭头的颜色。
U32	ColorGrasp	抓取工具的颜色。

18.15.2 配置选项

在 GUIConf.h 中设置上述类型的自定义配置结构，可以决定皮肤的默认外观。下表显示可用的配置选项：

宏	描述
SCROLLBAR_SKINPROPS_PRESSED	定义已按下状态所使用的默认属性。
SCROLLBAR_SKINPROPS_UNPRESSED	定义未按下状态所使用的默认属性。

18.15.3 换肤 API

下表按字母顺序列出了可用的程序：

程序	描述
SCROLLBAR_DrawSkinFlex()	SCROLLBAR_SKIN_FLEX 的换肤回调函数。（本章开始部分已介绍）
SCROLLBAR_GetSkinFlexProps()	返回指定 SCROLLBAR 皮肤的属性。（本章开始部分已介绍）
SCROLLBAR_SetDefaultSkin()	设置新创建的 SCROLLBAR 小工具所使用的默认皮肤。（本章开始部分已介绍）
SCROLLBAR_SetDefaultSkinClassic()	将经典设计设置为新创建的 SCROLLBAR 小工具的默认设计。（本章开始部分已介绍）
SCROLLBAR_SetSkin()	设置指定 SCROLLBAR 小工具的皮肤。（本章开始部分已介绍）
SCROLLBAR_SetSkinClassic()	设置指定 SCROLLBAR 小工具的经典设计。（本章开始部分已介绍）
SCROLLBAR_SetSkinFlexProps()	设置指定 SCROLLBAR 皮肤的属性。

SCROLLBAR_SetSkinFlexProps()

之前	之后
	

描述

使用该函数可更改皮肤的颜色。

原型

```
void SCROLLBAR_SetSkinFlexProps(const SCROLLBAR_SKINFLEX_PROPS * pProps,
                                int Index);
```

参数	描述
pProps	类型 SCROLLBAR_SKINFLEX_PROPS 的结构的指针。
索引	（参见下表）。

参数 Index 的允许值	
SCROLLBAR_SKINFLEX_PI_PRESSED	已按下状态的属性。
SCROLLBAR_SKINFLEX_PI_UNPRESSED	未按下状态的属性。

其他信息

该函数会将指针传递至 SCROLLBAR_SKINFLEX_PROPS 结构。它可以用于设置皮肤的颜色。使用函数 SCROLLBAR_GetSkinFlexProps() 可获取皮肤的当前属性。

18.15.4 命令列表

换肤程序会收到 WIDGET_ITEM_DRAW_INFO 结构的指针。此结构的 Cmd 成员包含需要处理的命令。下表显示传递至 SCROLLBAR_SKIN_FLEX 回调函数的所有命令：

命令	描述
WIDGET_ITEM_CREATE	创建小工具后立即发送。
WIDGET_ITEM_DRAW_BUTTON_L	换肤函数将绘制左侧按钮。
WIDGET_ITEM_DRAW_BUTTON_R	换肤函数将绘制右侧按钮。
WIDGET_ITEM_DRAW_OVERLAP	换肤函数将绘制重叠区域。
WIDGET_ITEM_DRAW_SHAFT_L	换肤函数将绘制轴的左侧部分。
WIDGET_ITEM_DRAW_SHAFT_R	换肤函数将绘制轴的右侧部分。
WIDGET_ITEM_DRAW_THUMB	换肤函数将绘制缩略图。
WIDGET_ITEM_GET_BUTTONSIZE	换肤函数将返回按钮大小。

WIDGET_ITEM_DRAW_BUTTON_L、 WIDGET_ITEM_DRAW_BUTTON_R

换肤程序将绘制按钮。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	要绘制的项目的索引。
x0	窗口坐标中按钮最左侧的坐标。
y0	窗口坐标中按钮最顶端的坐标。
x1	窗口坐标中按钮最右侧的坐标。
y1	窗口坐标中按钮最底部的坐标。
p	SCROLLBAR_SKINFLEX_INFO 结构的指针。

SCROLLBAR_SKINFLEX_INFO 的元素

数据类型	元素	描述
int	IsVertical	如果为水平进度条，则为 0；如果为垂直进度条，则为 1。
int	State	（参见下表）

元素 State 的允许值	
PRESSED_STATE_NONE	未按下任何按钮。
PRESSED_STATE_RIGHT	按下右侧按钮。
PRESSED_STATE_LEFT	按下左侧按钮。
PRESSED_STATE_THUMB	按下缩略图。

WIDGET_ITEM_DRAW_OVERLAP

换肤程序将绘制缩略图。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	要绘制的项目的索引。
x0	窗口坐标中重叠区域最左侧的坐标。
y0	窗口坐标中重叠区域最顶端的坐标。
x1	窗口坐标中重叠区域最右侧的坐标。
y1	窗口坐标中重叠区域最底部的坐标。

其他信息

如果对话框在边框具有垂直和水平滚动条，则可能存在重叠区域。通常，重叠区域看起来与轴区域相同。

示例

以下屏幕截图显示含 2 个滚动条的窗口，而 2 个滚动条在客户端窗口的右下角具有重叠区域：

WIDGET_ITEM_DRAW_SHAFT_L、 WIDGET_ITEM_DRAW_SHAFT_R

换肤程序将绘制轴区域。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	要绘制的项目的索引。
x0	窗口坐标中轴区域最左侧的坐标。
y0	窗口坐标中轴区域最顶端的坐标。
x1	窗口坐标中轴区域最右侧的坐标。
y1	窗口坐标中轴区域最底部的坐标。

WIDGET_ITEM_DRAW_THUMB

换肤程序将绘制缩略图。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	要绘制的项目的索引。
x0	窗口坐标中缩略图区域最左侧的坐标。
y0	窗口坐标中缩略图区域最顶端的坐标。
x1	窗口坐标中缩略图区域最右侧的坐标。
y1	窗口坐标中缩略图区域最底部的坐标。
p	SCROLLBAR_SKINFLEX_INFO 结构的指针。

SCROLLBAR_SKINFLEX_INFO 的元素

请参阅 WIDGET_ITEM_DRAW_BUTTON_L。

WIDGET_ITEM_GET_BUTTONSIZE

换肤程序将返回按钮大小。按钮大小含义如下：

- 水平滚动条将返回滚动条的高度。
- 垂直滚动条将返回滚动条的宽度。

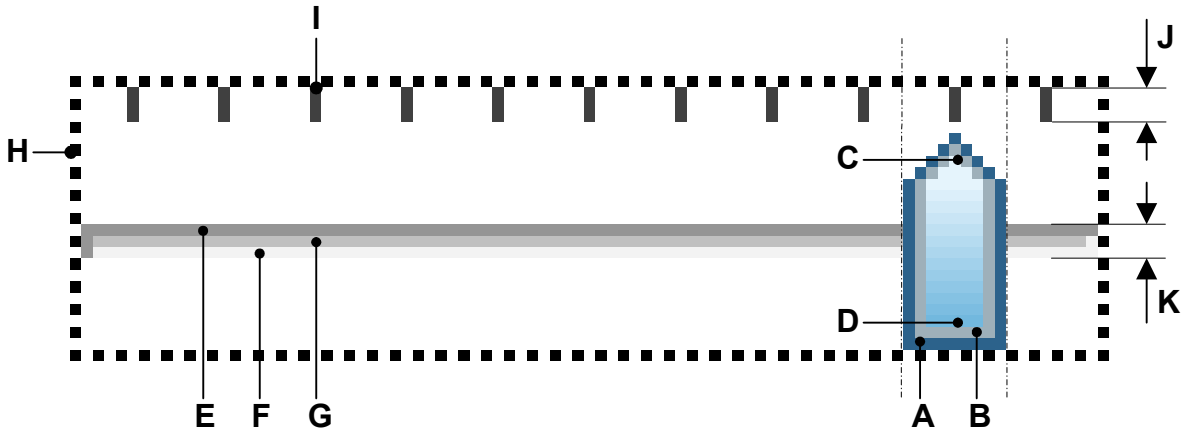
示例

在多数情况下，以下代码可用于返回正确的值：

```
int _SkinningCallback(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
  SCROLLBAR_SKINFLEX_INFO * pSkinInfo;

  pSkinInfo = (SCROLLBAR_SKINFLEX_INFO *)pDrawItemInfo->p;
  switch (pDrawItemInfo->Cmd) {
  case WIDGET_ITEM_GET_BUTTONSIZE:
    return (pSkinInfo->IsVertical) ?
           pDrawItemInfo->x1 - pDrawItemInfo->x0 + 1 :
           pDrawItemInfo->y1 - pDrawItemInfo->y0 + 1;
    ...
  }
}
```

18.16 SLIDER_SKIN_FLEX

下图显示皮肤的详细信息：



上图显示皮肤的详细信息。它由含滑块的轴和上述刻度标记组成。另外，如果小工具具有输入焦点，则将显示聚焦框。滑块由框架和渐变绘制：

详细信息	描述
A	滑块框架的外部颜色。
B	滑块框架的内部颜色。
C	渐变的顶部颜色。
D	渐变的底部颜色。
E	轴的第一种颜色。
F	轴的第二种颜色。
G	轴的第三种颜色。
H	聚焦框。
I	刻度标记。
J	刻度标记的大小。
K	轴的大小。

18.16.1 配置结构

如果要设置皮肤的默认外观或在运行时更改外观，则请使用类型 SLIDER_SKINFLEX_PROPS 的配置结构：

RADIO_SKINFLEX_PROPS 的元素

数据类型	元素	描述
U32	aColorFrame[2]	[0] - 外部框架颜色。 [1] - 内部框架颜色。
U32	aColorInner[2]	[0] - 渐变的顶部颜色。 [1] - 渐变的底部颜色。
U32	aColorShaft[3]	[0] - 轴的第一种框架颜色。 [1] - 轴的第二种框架颜色。 [2] - 轴的内部颜色。
U32	ColorTick	刻度标记的颜色。
U32	ColorFocus	聚焦框的颜色。
int	TickSize	刻度标记的大小。
int	ShaftSize	轴的大小。

18.16.2 配置选项

在 GUIConf.h 中设置上述类型的自定义配置结构，可以决定皮肤的默认外观。下表显示可用的配置选项：

宏	描述
SLIDER_SKINPROPS_PRESSED	定义已按下状态所使用的默认属性。
SLIDER_SKINPROPS_UNPRESSED	定义未按下状态所使用的默认属性。

18.16.3 换肤 API

下表按字母顺序列出了可用的程序：

程序	描述
SLIDER_DrawSkinFlex()	SLIDER_SKIN_FLEX 的换肤回调函数。（本章开始部分已介绍）
SLIDER_GetSkinFlexProps()	返回指定 SLIDER 皮肤的属性。（本章开始部分已介绍）
SLIDER_SetDefaultSkin()	设置新创建的 SLIDER 小工具所使用的默认皮肤。（本章开始部分已介绍）
SLIDER_SetDefaultSkinClassic()	将经典设计设置为新创建的 SLIDER 小工具的默认设计。（本章开始部分已介绍）
SLIDER_SetSkin()	设置指定 SLIDER 小工具的皮肤。（本章开始部分已介绍）
SLIDER_SetSkinClassic()	设置指定 SLIDER 小工具的经典设计。（本章开始部分已介绍）
SLIDER_SetSkinFlexProps()	设置指定 SLIDER 皮肤的属性。

SLIDER_SetSkinFlexProps()

之前	之后
	

描述

使用该函数可更改皮肤的颜色、刻度标记和轴的大小。

原型

```
void SLIDER_SetSkinFlexProps(const SLIDER_SKINFLEX_PROPS * pProps,
                             int Index);
```

参数	描述
pProps	类型 SLIDER_SKINFLEX_PROPS 的结构的指针。
Index	（参见下表）。

参数 Index 的允许值	
SLIDER_SKINFLEX_PI_PRESSED	已按下状态的属性。
SLIDER_SKINFLEX_PI_UNPRESSED	未按下状态的属性。

其他信息

该函数会将指针传递至 SLIDER_SKINFLEX_PROPS 结构。它可以用于设置皮肤的颜色。
使用函数 SLIDER_GetSkinFlexProps() 则可获取皮肤的当前属性。

18.16.4 命令列表

换肤程序会收到 WIDGET_ITEM_DRAW_INFO 结构的指针。此结构的 Cmd 成员包含需要处理的命令。下表显示传递至 SLIDER_SKIN_FLEX 回调函数的所有命令：

命令	描述
WIDGET_ITEM_CREATE	创建小工具后立即发送。
WIDGET_ITEM_DRAW_FOCUS	换肤函数将绘制聚焦框。
WIDGET_ITEM_DRAW_SHAFT	换肤函数将绘制轴。
WIDGET_ITEM_DRAW_THUMB	换肤函数将绘制滑块。
WIDGET_ITEM_DRAW_TICKS	换肤函数将绘制刻度标记。

WIDGET_ITEM_CREATE

如有必要，换肤程序将设置与皮肤相关的属性，如透明度或文本对齐方式。

WIDGET_ITEM_DRAW_FOCUS

换肤程序将绘制聚焦框。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	要绘制的项目的索引。
x0	窗口坐标中小工具最左侧的坐标。
y0	窗口坐标中小工具最顶端的坐标。
x1	窗口坐标中小工具最右侧的坐标。
y1	窗口坐标中小工具最底部的坐标。

WIDGET_ITEM_DRAW_SHAFT

换肤程序将绘制轴。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	要绘制的项目的索引。
x0	窗口坐标中小工具 + 1 最左侧的坐标。
y0	窗口坐标中小工具 + 1 最顶端的坐标。
x1	窗口坐标中小工具 -1 最右侧的坐标。
y1	窗口坐标中小工具 -1 最底部的坐标。

WIDGET_ITEM_DRAW_THUMB

换肤程序将绘制滑块本身。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	要绘制的项目的索引。
x0	窗口坐标中滑块最左侧的坐标。
y0	窗口坐标中滑块最顶端的坐标。
x1	窗口坐标中滑块最右侧的坐标。
y1	窗口坐标中滑块最底部的坐标。
p	SLIDER_SKINFLEX_INFO 结构的指针。

SLIDER_SKINFLEX_INFO 的元素

数据类型	元素	描述
int	Width	滑块的宽度。
int	IsPressed	如果按下滑块，则为 1；如果未按下，则为 0。
int	IsVertical	如果为水平滑块，则为 0；如果为垂直滑块，则为 1。

WIDGET_ITEM_DRAW_TICKS

换肤程序将绘制刻度标记。

WIDGET_ITEM_DRAW_INFO 结构的内容：

元素	描述
hWin	小工具的句柄。
ItemIndex	要绘制的项目的索引。
x0	窗口坐标中小工具 + 1 最左侧的坐标。
y0	窗口坐标中小工具 + 1 最顶端的坐标。
x1	窗口坐标中小工具 -1 最右侧的坐标。
y1	窗口坐标中小工具 -1 最底部的坐标。
p	SLIDER_SKINFLEX_INFO 结构的指针。

SLIDER_SKINFLEX_INFO 的元素

数据类型	元素	描述
int	Width	滑块的宽度。
int	NumTicks	要绘制的刻度数。
int	Size	刻度标记线条的长度。
int	IsPressed	如果按下滑块，则为 1；如果未按下，则为 0。
int	IsVertical	如果为水平滑块，则为 0；如果为垂直滑块，则为 1。

第 19 章

多缓冲

多缓冲是一种使用多个帧缓冲器的方法。其基本原理如下：在启用多个缓冲器的情况下，由显示控制器所使用的前置缓冲器会在屏幕上产生图像，同时，一个或多个后置缓冲器则用于绘图操作。绘图操作完成后，后置缓冲器成为可见的前置缓冲器。

如果使用两个缓冲器（即一个前置缓冲器和一个后置缓冲器），通常称之为“双缓冲”；如果使用两个后置缓冲器和一个前置缓冲器，则称之为“三缓冲”。

通常，这种方法可以避免多种无用效果：

- 可见的屏幕逐项绘图过程
- 绘图操作重叠导致的闪烁效果
- 垂直消隐期之外的写操作导致的撕裂效果

下一节详细解释了这种方法的工作原理、使用该功能所需满足的条件、emWin 的配置方法以及“三缓冲”相对于“双缓冲”的优势。此外，该方法还说明如何配置可选窗口管理器，以自动使用“多缓冲”功能。

19.1 工作原理

由于多缓冲方法使用多个帧缓冲器，因此，即便绘图操作仍在进行中，屏幕画面也是完全渲染的结果。启动绘图过程时，前置缓冲器的当前内容会被复制到一个后置缓冲器中。在该操作完成后，所有绘图操作只对该后置缓冲器起作用。绘图操作完成后，后置缓冲器成为前置缓冲器。如果要使后置缓冲器成为可见的前置缓冲器，通常只需修改显示控制器的帧缓冲器起始地址寄存器即可。
可以认为，显示器的持续刷新是通过显示控制器的应用程序得以实现的。每秒 60 次。每个周期完成之后有一个垂直同步信号，通常称之为 VSYNC 信号。使后置缓冲器成为前置缓冲器的最佳时机是该信号出现之时。如果不考虑 VSYNC 信号，则可能产生撕裂效果。

撕裂效果：

19.1.1 双缓冲

如果是双缓冲，则采用两个缓冲器：一个前置缓冲器和一个后置缓冲器。启动绘图操作时，前置缓冲器的当前内容被复制到后置缓冲器中。绘图操作完成后，后置缓冲器成为可见的前置缓冲器。
正如前文所解释的那样，后置缓冲器成为可见的前置缓冲器的最佳时机是在显示控制器出现 VSYNC 信号之时。这样就体现出了双缓冲相对于三缓冲所存在的劣势：帧缓冲器起始地址不是在绘图操作结束时立即发生变化，就是等到下一个 VSYNC 信号出现之后再发生变化。这就意味着，要么产生撕裂效果，要么会因等待下一个 VSYNC 信号而导致性能下降。

19.1.2 三缓冲

正如其名，该方法采用三个缓冲器：一个前置缓冲器和两个后置缓冲器。启动绘图操作时，前置缓冲器的当前内容被复制到第一个后置缓冲器中。绘图操作完成后，后置缓冲器成为可见的前置缓冲器。与双缓冲解决方案相反的是，该方法不需要立即切换到该缓冲器。在显示控制器出现下一个 VSYNC 信号时，可以切换到新的前置缓冲器，它是通过一个中断服务程序 (ISR) 来实现的。大多数能处理多个帧缓冲器的显示控制器提供作为中断源的 VSYNC 信号。处于等待中的前置缓冲器应在 ISR 中变成可见。在等待前置缓冲器变成可见之前，该缓冲器不会用于其他绘图操作。如果在等待前置缓冲器变为可见前启动另一次绘图操作，则第二个后置缓冲器将用于该绘图操作。如果在等待 VSYNC 信号时有新的缓冲器可用，则该缓冲器将成为新的等待前置缓冲器，如此等等，不一而足。这样可以始终保护前置缓冲器，避免对其进行写操作。

需要指出的是，在某些显示控制器中，只有在绘制下一帧图像时，对显示缓冲器起始地址所作的更改才有效。在这种情况下，无论解决方案是否带 ISR，它们的效果都相同。只有更改地址的操作立即生效时，才需要 ISR 来避免撕裂效果。

19.2 要求

下表给出了使用多个缓冲器所需满足的要求：

- 显示控制器应能支持多个帧缓冲器。
- 多个帧缓冲器应有足够的视频 RAM 可用。
- 如需避免撕裂效果，则必须能对显示控制器的 VSYNC 信号作出反应。为了实现最佳性能，建议使用三缓冲。

19.3 限制

非常遗憾的是，虚拟屏幕和多缓冲支持两者不可兼得。

19.4 配置

通常情况下，配置文件 LCDConf.c 中有 2 个程序需要修改，即显示配置程序 LCD_X_Config() 和驱动回调函数 LCD_X_DisplayDriver()。

19.4.1 LCD_X_Config()

通常，在该处所需做的是：启用多个缓冲器支持。

基本配置

创建显示驱动器件之前，首先需要配置多缓冲器接口。通常通过 LCD_X_Config() 进行配置。创建显示驱动器件之前，务必启用多缓冲，如以下代码片断所示：

```
void LCD_X_Config(void) {
  //
  // Initialize multibuffering
  //
  GUI_MULTIBUF_Config(NUM_BUFFERS);
  //
  // Set display driver and color conversion
  //
  GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER, COLOR_CONVERSION, 0, 0);
  ...
}
```

为复制缓冲器定制回调程序

另外，可以为缓冲器的复制设定一个回调程序。正如上文所说，在绘图操作开始时，必须将当前前置缓冲器的内容复制到后置缓冲器。通常可通过简单的 memcpy 操作来实现。但是，如果所使用的显示控制器包括一个能执行复制操作的 BitBLT 引擎，则可以利用该引擎来执行复制操作。否则，应当使用基于 DMA 的程序来完成复制操作。在这些情况下，可以通过一个定制的回调函数来完成该操作。该函数可在创建显示驱动器件后安装，如以下代码片断所示：


```
static void _CopyBuffer(int LayerIndex, int IndexSrc, int IndexDst) {
  unsigned long BufferSize, AddrSrc, AddrDst;

  //
  // Calculate the size of one frame buffer
  //
  BufferSize = (XSIZE * YSIZE * BITSPERPIXEL) / 8;
  //
  // Calculate source- and destination address
  //
  AddrSrc    = _VRamBaseAddr + BufferSize * IndexSrc;
  AddrDst    = _VRamBaseAddr + BufferSize * IndexDst;
  memcpy((void *)AddrDst, (void *)AddrSrc, BufferSize);
}

void LCD_X_Config(void) {
  //
  // Initialize multibuffering
  //
  GUI_MULTIBUF_Config(NUM_BUFFERS);
  //
  // Set display driver and color conversion
  //
  GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER, COLOR_CONVERSION, 0, 0);
  //
  // Set custom callback function for copy operation
  //
  LCD_SetDevFunc(0, LCD_DEVFUNC_COPYBUFFER, (void (*)())_CopyBuffer);
}
```

需要注意的是，上述示例通常不起作用，因为驱动默认使用简单的 memcpy() 操作。只有需要使用加速功能时，定制回调函数才有意义。

19.4.2 LCD_X_DisplayDriver()

绘图过程完成后，后置缓冲器应变成可见。显示驱动将 LCD_X_SHOWBUFFER 命令发送至显示驱动回调函数。回调函数收到命令时即作出反应，确保缓冲器变成可见。这既可以通过 ISR 实现，也可以通过直接向显示控制器的帧缓冲器起始地址中写入正确的地址来实现。

有 ISR 时

以下代码片断显示了一种示例实现方法：

```
static void _ISR_EndOfFrame(void) {
  unsigned long Addr, BufferSize;

  if (_PendingBuffer >= 0) {
    //
    // Calculate address of the given buffer
    //
    BufferSize = (XSIZE * YSIZE * BITSPERPIXEL) / 8;
    Addr       = _VRamBaseAddr + BufferSize * pData->Index;
    //
    // Make the given buffer visible
    //
    AT91C_LCDC_BA1 = Addr;
    //
    // Send a confirmation that the buffer is visible now
    //
    GUI_MULTIBUF_Confirm(_PendingBuffer);
    _PendingBuffer = -1;
  }
}

int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * p) {
  switch (Cmd) {
  case LCD_X_SHOWBUFFER:{
      LCD_X_SHOWBUFFER_INFO * pData;

      pData = (LCD_X_SHOWBUFFER_INFO *)p;
      //
      // Remember buffer index to be used by ISR
      //
      _PendingBuffer = pData->Index;
    }
    break;
  ...
  }
}
```


以上实现方法假定存在一个 ISR，可在出现下一个 VSYNC 信号时执行。

无 ISR 时

如果没有 ISR 可用，则可以直接设定地址，但可能导致撕裂效果。
以下代码片断显示了一种示例实现方法：

```
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * p) {
  unsigned long Addr, BufferSize;

  switch (Cmd) {
  ...
  case LCD_X_SHOWBUFFER:{
      LCD_X_SHOWBUFFER_INFO * pData;

      pData = (LCD_X_SHOWBUFFER_INFO *)p;
      //
      // Calculate address of the given buffer
      //
     BufferSize = (XSIZE * YSIZE * BITSPERPIXEL) / 8;
     Addr       = _VRamBaseAddr + BufferSize * pData->Index;
      //
      // Make the given buffer visible
      //
      AT91C_LCDC_BA1 = Addr;
      //
      // Send a confirmation that the buffer is visible now
      //
      GUI_MULTIBUF_Confirm(pData->Index);
    }
    break;
  ...
  }
}
```

19.5 通过窗口管理器自动使用多个缓冲器

可选的窗口管理器（WM）能够自动使用多缓冲器功能。使用函数 WM_MULTIBUF_Enable() 可启用该功能。如果该功能启用，则 WM 会首先切换至后置缓冲器，然后再重新绘制无效的窗口。绘制完所有无效窗口后，新屏幕变成可见。这样就可以隐藏屏幕逐窗绘图过程。

19.6 多缓冲 API

下表列出了多缓冲器功能可用的程序。

程序	描述
基本函数	
GUI_MULTIBUF_Begin()	需在启动绘图操作之前调用。
GUI_MULTIBUF_BeginEx()	同上，仅 LayerIndex 参数不同。
GUI_MULTIBUF_Config()	需调用以配置多缓冲器功能。
GUI_MULTIBUF_ConfigEx()	同上，仅 LayerIndex 参数不同。
GUI_MULTIBUF_Confirm()	应在等待前置缓冲器成为可见之后立即调用。
GUI_MULTIBUF_ConfirmEx()	同上，仅 LayerIndex 参数不同。
GUI_MULTIBUF_End()	需在绘图操作完成之后调用。
GUI_MULTIBUF_EndEx()	同上，仅 LayerIndex 参数不同。
GUI_MULTIBUF_GetNumBuffers()	返回所使用的缓冲器的个数。
GUI_MULTIBUF_GetNumBuffersEx()	同上，仅 LayerIndex 参数不同。
GUI_MULTIBUF_UseSingleBuffer()	使多缓冲功能为所有层使用一个帧。
可选窗口管理器函数	
WM_MULTIBUF_Enable()	启用或禁用可选 WM 的自动多缓冲支持。

（在以后的版本中，以上程序接口可能发生变化）

GUI_MULTIBUF_Begin()

描述

需在刚好启动绘图操作之前调用。

原型

```
void GUI_MULTIBUF_Begin(void);
```

其他信息

该函数可确保将当前的前置缓冲器复制到后置缓冲器，然后在所有后续绘图操作中使用该后置缓冲器。复制操作一般由显示驱动自行完成。如前文所述，这也可以通过定制回调函数来实现。

GUI_MULTIBUF_BeginEx()

描述

详见 GUI_MULTIBUF_Begin()。

原型

```
void GUI_MULTIBUF_BeginEx(int LayerIndex);
```

参数	描述
LayerIndex	要使用的层。

GUI_MULTIBUF_Config()

描述

在初始化过程中需要调用该函数，一般是在 LCD_X_Config() 中调用，可用来启用多缓冲器功能。

原型

```
void GUI_MULTIBUF_Config(int NumBuffers);
```

参数	描述
NumBuffers	要使用的缓冲器个数。可使用以下数值： 2 – 双缓冲 3 – 三缓冲

其他信息

该函数需在创建显示驱动器件之前调用。

GUI_MULTIBUF_ConfigEx()

描述

详见 GUI_MULTIBUF_Config()。

原型

```
void GUI_MULTIBUF_ConfigEx(int LayerIndex, int NumBuffers);
```

参数	描述
LayerIndex	要使用的层。
NumBuffers	要使用的缓冲器个数。可使用以下数值： 2 – 双缓冲 3 – 三缓冲

GUI_MULTIBUF_Confirm()

描述

在新的缓冲器变成可见之后，需立即调用该函数。

原型

```
void GUI_MULTIBUF_Confirm(int Index);
```

参数	描述
Index	已变成可见的缓冲器的索引。

其他信息

通常该函数既可由切换至新的前置缓冲器的 ISR 来调用，也可通过显示驱动回调函数调用。

GUI_MULTIBUF_ConfirmEx()

描述

详见 GUI_MULTIBUF_Confirm()。

原型

```
void GUI_MULTIBUF_ConfirmEx(int LayerIndex, int BufferIndex);
```

参数	描述
LayerIndex	要使用的层。
Index	已变成可见的缓冲器的索引。

GUI_MULTIBUF_End()

描述

该函数需在完全绘制好新的屏幕之后调用。

原型

```
void GUI_MULTIBUF_End(void);
```

其他信息

调用该函数时，显示驱动将一个 LCD_X_SHOWBUFFER 命令发往显示驱动回调程序，然后，由该程序使指定的缓冲器成为前置缓冲器。

GUI_MULTIBUF_EndEx()

描述

详见 GUI_MULTIBUF_End()。

原型

```
void GUI_MULTIBUF_EndEx(int LayerIndex);
```

参数	描述
LayerIndex	要使用的层。

GUI_MULTIBUF_GetNumBuffers()

描述

该函数返回当前层所配置的缓冲器个数。

原型

```
int GUI_MULTIBUF_GetNumBuffers(void);
```

返回值

当前层所配置的缓冲器个数。

GUI_MULTIBUF_GetNumBuffersEx()

描述

详见 GUI_MULTIBUF_GetNumBuffers()。

原型

```
int GUI_MULTIBUF_GetNumBuffersEx(int LayerIndex);
```

参数	描述
LayerIndex	要使用的层。

返回值

指定层所配置的缓冲器个数。

GUI_MULTIBUF_UseSingleBuffer()

描述

使多缓冲功能为所有层使用一个帧。

原型

```
void GUI_MULTIBUF_UseSingleBuffer(void);
```

其他信息

该函数需在创建显示驱动器件之前调用。

WM_MULTIBUF_Enable()

描述

使用该程序可启用自动使用多个缓冲器的功能，详见本节开始部分的描述。

原型

```
int WM_MULTIBUF_Enable(int OnOff);
```

参数	描述
OnOff	1 代表启用自动多缓冲支持， 0 则代表禁用该功能。

返回值

上一状态。

第 20 章

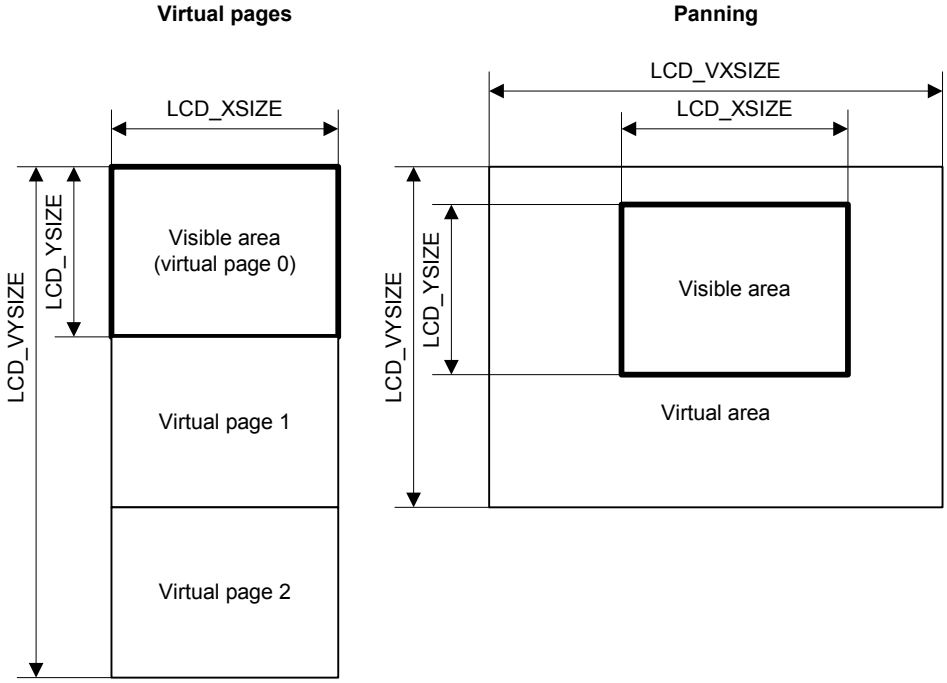
虚拟屏幕 / 虚拟页面

虚拟屏幕指比显示器实际尺寸大的显示区域。它要求额外的视频存储器，即使是在较慢的 CPU 上也能在不同屏幕之间快速切换。本章将探讨

- 使用虚拟屏幕的要求；
- emWin 的配置方法；
- 以及虚拟屏幕的使用方法。

如果配置了虚拟显示区，则可通过设置源改变显示器的可见部分。

20.1 简介



emWin 的虚拟屏幕支持功能可用于图像平移，或者用于在不同视频页面之间进行切换。

图像平移

如果在应用时，所使用的屏幕大于显示器尺寸，则可使用虚拟屏幕 API 函数来使目标区域可见。

虚拟页面

虚拟页面是把显示 RAM 当作多个页面进行使用的一种方式。举例来说，如果某个应用需要三种不同的屏幕，则每个屏幕就可以使用显示 RAM 中属于自己的页面。如果是这种情况，则在使用这些屏幕前，该应用就可以绘制第二个和第三个页面。此后，该应用可以通过 emWin 的虚拟屏幕 API 函数在不同页面之间进行快速切换。这些函数只需设定显示目标屏幕所需要的显示起始地址即可。这种情况下，虚拟尺寸 Y 一般为显示器尺寸 Y 的倍数。

20.2 要求

虚拟屏幕功能要求其硬件的显示 RAM 数要比单屏幕所要求的多，同时要求硬件能够更改显示输出的起始位置。

视频 RAM

所用显示控制器应能为虚拟区域提供视频 RAM 支持。例如，如果显示器的分辨率为 320x240，色彩深度为 16 位 / 像素，并且需要支持 2 个屏幕，则可按以下方式计算所需视频 RAM 的大小：

```
Size = LCD_XSIZE * LCD_YSIZE * LCD_BITSPERPIXEL / 8 * NUM_SCREENS
Size = 320 x 240 x 16 / 8 x 2
Size = 307200 Bytes
```


显示起始位置可配置

所使用的显示控制器需要一个可以配置的显示起始位置。这也就是说，显示驱动要么有一个用于设置帧缓冲器起始地址的寄存器，要么通过一个命令来设置左上部的显示起始位置。

20.3 配置

虚拟屏幕应在初始化过程中进行配置。定义虚拟显示尺寸需使用 LCD_SetVSizeEx() 函数。另外，还需要通过设置正确的帧缓冲器起始地址，来对驱动回调程序中的 LCD_X_SETORG 命令作出反应。

LCD_SetVSizeEx()

描述

设置虚拟显示尺寸。

原型

```
int LCD_SetVSizeEx(int LayerIndex, int xSize, int ySize);
```

参数	描述
LayerIndex	以零为基准的层索引，单层系统中一般为 0。
xSize	虚拟显示的水平分辨率。
ySize	虚拟显示的垂直分辨率。

返回值

如果成功，返回值为 0；如果发生错误，则返回值为 1。

20.4 示例

下面通过一些例子来说明如何在 emWin 中使用虚拟屏幕。

20.4.1 基本示例

以下示例说明了如何使用分辨率为 128x192 的虚拟屏幕以及分辨率为 128x64 的显示器，以便在 3 个不同屏幕之间进行即时切换。

配置

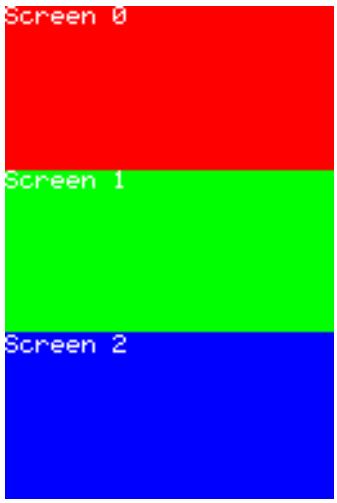
```
LCD_SetSizeEx (0, 128,  64);
LCD_SetVSizeEx(0, 128, 192);
```

应用

```
  GUI_SetColor(GUI_RED);
  GUI_FillRect(0,   0, 127,  63);
  GUI_SetColor(GUI_GREEN);
  GUI_FillRect(0,  64, 127, 127);
  GUI_SetColor(GUI_BLUE);
  GUI_FillRect(0, 128, 127, 191);
  GUI_SetColor(GUI_WHITE);
  GUI_SetTextMode(GUI_TM_TRANS);
  GUI_DispStringAt("Screen 0", 0,   0);
  GUI_DispStringAt("Screen 1", 0,  64);
  GUI_DispStringAt("Screen 2", 0, 128);
  GUI_SetOrg(0,  64); /* Set origin to screen 1 */
  GUI_SetOrg(0, 128); /* Set origin to screen 2 */
```

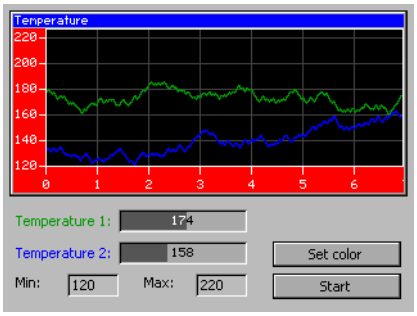
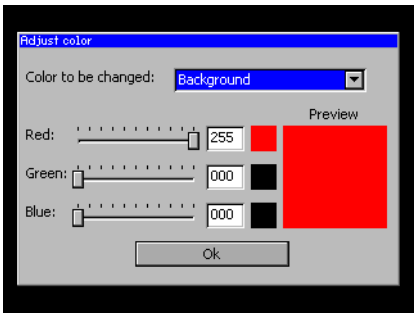
输出

下表所示为显示器的输出：

描述	显示器输出	虚拟区域的内容
执行前 GUI_SetOrg(0, 64)		
执行后 GUI_SetOrg(0, 64)		
执行后 GUI_SetOrg(0, 128)		

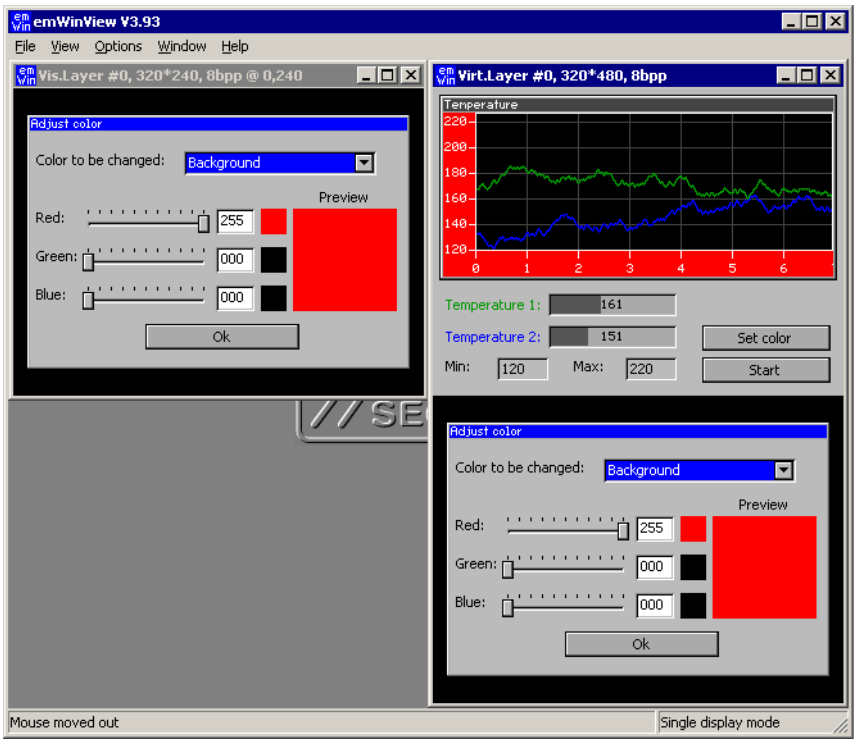
20.4.2 使用窗口管理器的实时示例

emWin 附有一个示例，展示了如何在实时应用中使用虚拟屏幕。该示例可在 `Sample\Tutorial\VSCREEN_RealTime.c` 目录下找到。

屏幕 0 / 页面 0	屏幕 1 / 页面 1
	

显示简短介绍后，该示例将在 2 个独立页面上创建 2 个屏幕（如上文所示）。第一个屏幕显示一个对话框，其中含有 2 条温度曲线的图形表示。按下“设置色彩”(Set color) 按钮时，应用将立即切换到第二个屏幕，即使是在较慢的 CPU 上也能快速切换。按下“调整颜色”(Adjust color) 对话框的“确定”按钮后，应用将返回第一个屏幕。
更多详情，请参见该示例的源代码。

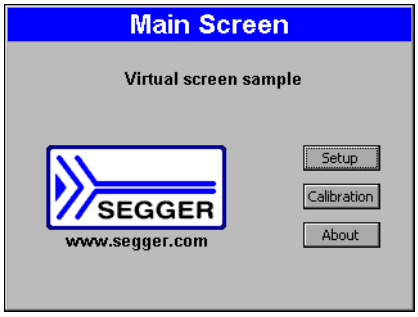
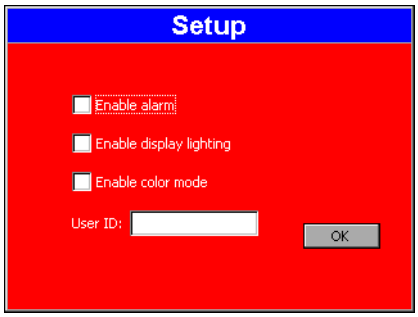
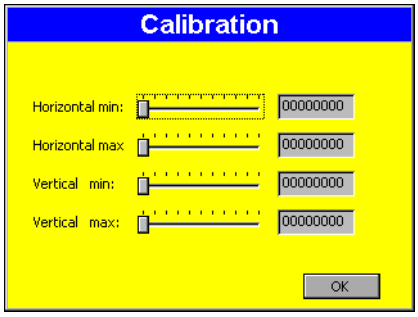
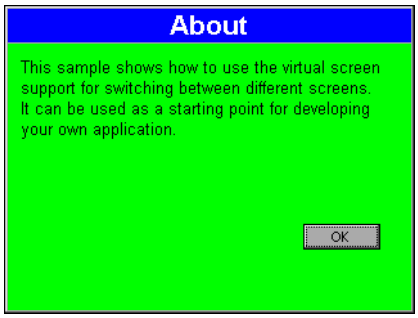
上例在查看器中的屏幕截图



如果使用查看器，则可以同时显示两个屏幕。在以上屏幕截图中，可见显示位于左侧，配置的整个虚拟显示 RAM 中的内容位于右侧。

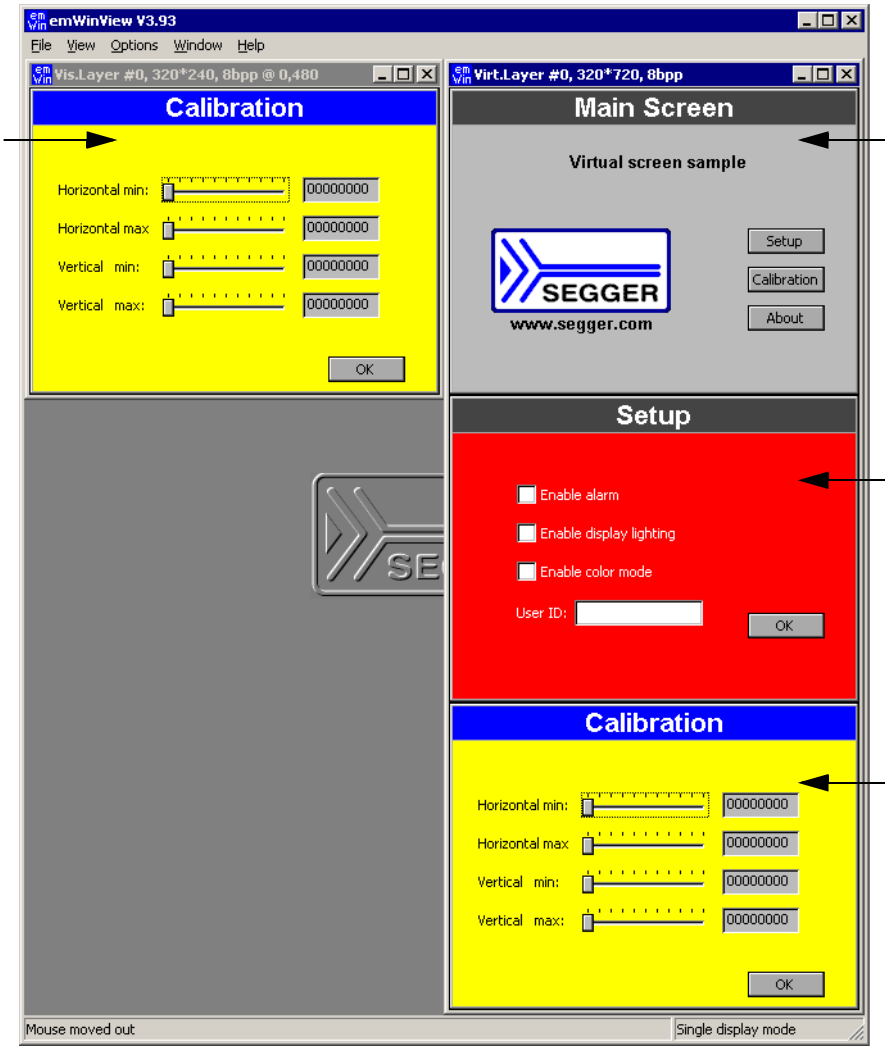
20.4.3 使用窗口管理器的对话框示例

第二个高级示例位于文件夹 “Sample\GUI\VSCREEN_MultiPage.” 下。该示例通过虚拟屏幕在 3 个不同的视频页面上显示 4 个屏幕。该应用由以下屏幕构成：

主屏幕 / 页面 0	设置屏幕 / 页面 1
	
校准屏幕 / 页面 2	**关于屏幕 / 页面 2**
	

在简短的介绍屏幕之后，显示器上将通过页面 0 显示 “主屏幕” (Main Screen)。按下 “设置” (Setup) 按钮之后，将在页面 1 上创建 “设置” 屏幕。该屏幕创建后，应用将切换至页面 1，从而使该屏幕可见。“校准” (Calibration) 和 “关于” (About) 两个屏幕都使用页面 2。如果用户按下 “校准” 或 “关于” 两个按钮之一，则应用将切换至页面 2 并显示对话框。

上例在查看器中的屏幕截图



查看器可以同时显示所有页面。在以上屏幕截图中，可见显示位于左侧，页面 0-2 显示的整个层（虚拟显示 RAM）的内容位于右侧。

20.5 虚拟屏幕 API

下表列出了虚拟屏幕功能可用的程序。

程序	描述
GUI_GetOrg()	返回显示起始位置。
GUI_SetOrg()	设置显示起始位置。

GUI_GetOrg()

描述

返回显示起始位置。

原型

```
void GUI_GetOrg(int * px, int * py);
```

参数	描述
px	int 类变量的指针，用于存储显示起始位置 X 值。
py	int 类变量的指针，用于存储显示起始位置 Y 值。

其他信息

该函数把当前显示起始位置存储到给定指针指向的变量之中。

GUI_SetOrg()

描述

设置显示起始位置。

原型

```
void GUI_SetOrg(int x, int y);
```

参数	描述
x	显示起始位置的新的 X 位置。
y	显示起始位置的新的 Y 位置。

第 21 章

多层 / 多显示支持

如果需要访问多个显示，或者显示控制器支持多层（且需要使用多层），则需要使用 emWin 的多层支持功能。
多层支持和多显示支持的原理相同。所访问的每一层 / 每个显示都可以有自己的颜色设置、尺寸和显示驱动。多层的初始化非常简单：最大可用层数 GUI_NUM_LAYERS 应在 GUIConf.h 中定义，每一层都需要一个显示驱动器件，该器件应在初始化过程中通过配置程序 LCD_X_Config() 创建。对可用层的最大数量没有限制。

21.1 简介

窗口可以放在任何层或显示中，绘图操作可以应用于任何层或任何显示。由于在这一方面两者的差异非常小，因此，多层和多显示均以相同方式处理 （使用相同的 API 程序），并一律简称为多层，即使具体的嵌入式系统使用的是多显示。通过 emWin 查看器，可以查看每一个层 （显示），但在多显示系统中，还可查看实际输出 （复合视图）。目前，可以使用多显示和多层系统，但无法模拟它们。

21.1.1 选择绘图操作所使用的层

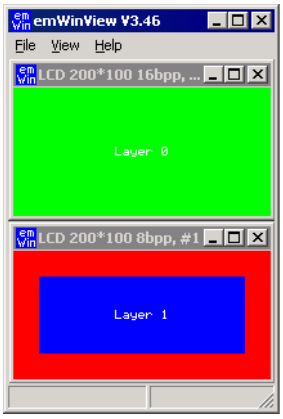
直接绘图时，使用的是默认层 0。其他层则可以通过 GUI_SelectLayer() 函数选择。

示例

以下示例展示了如何选择某个层来执行绘图操作：

```
void MainTask(void) {
  GUI_Init();
  /* Draw something on default layer 0 */
  GUI_SetBkColor(GUI_GREEN);
  GUI_Clear();
  GUI_DispStringHCenterAt("Layer 0", 100, 46);
  /* Draw something on layer 1 */
  GUI_SelectLayer(1); /* Select layer 1 */
  GUI_SetBkColor(GUI_RED);
  GUI_Clear();
  GUI_SetColor(GUI_BLUE);
  GUI_FillRect(20, 20, 179, 79);
  GUI_SetColor(GUI_WHITE);
  GUI_SetTextMode(GUI_TM_TRANS);
  GUI_DispStringHCenterAt("Layer 1", 100, 46);
  while(1) {
    GUI_Delay(100);
  }
}
```

上述示例的屏幕截图



21.1.2 选择窗口所使用的层

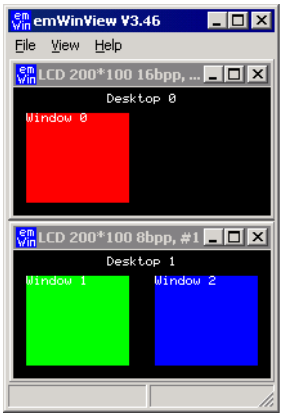
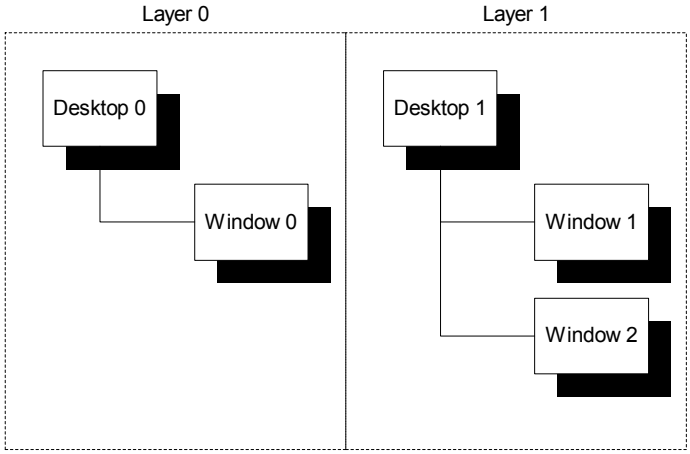
窗口管理器会自动跟踪哪些窗口位于哪个层。其实现方法非常简单：
如果使用窗口管理器，则每一层都有一个顶层 （桌面）窗口。
对于该层中的任何其他窗口，仅当其为这些桌面窗口之一的子代窗口 （子窗口或孙窗口等）时，才可见。某个窗口位于哪一层完全取决于其顶层窗口是哪一个桌面窗口。

示例

以下示例展示了如何在 2 个不同的桌面窗口上创建 3 个窗口：


```
/* Create 1 child window on destop 0 */
hWin0 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                                WM_GetDesktopWindowEx(0), WM_CF_SHOW, _cbWin0, 0);
/* Create 2 child windows on destop 1 */
hWin1 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                                WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin1, 0);
hWin2 = WM_CreateWindowAsChild(110, 20, 80, 70,
                                WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin2, 0);
```

下表展示了上例的屏幕截图和窗口层次关系：

屏幕截图	窗口层级关系
	

21.1.2.1 把窗口从一层移到另一层

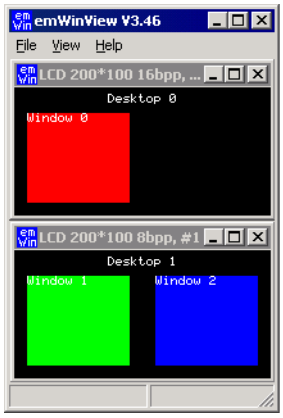
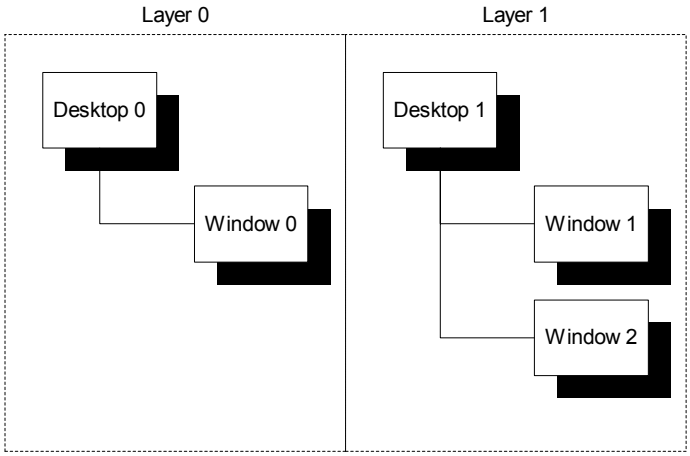
这种功能有时非常有用，可以轻松实现：如果某个窗口脱离其父窗口（某一层的桌面窗口，或者该桌面窗口的任何子代窗口），并附加到位于另一层的某个窗口上，则该窗口实际上从一层移到了另一层。

示例

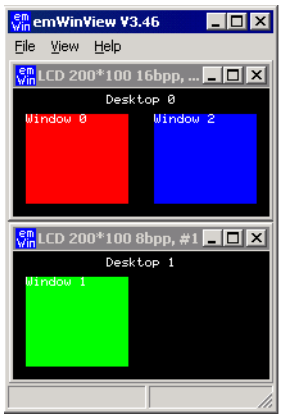
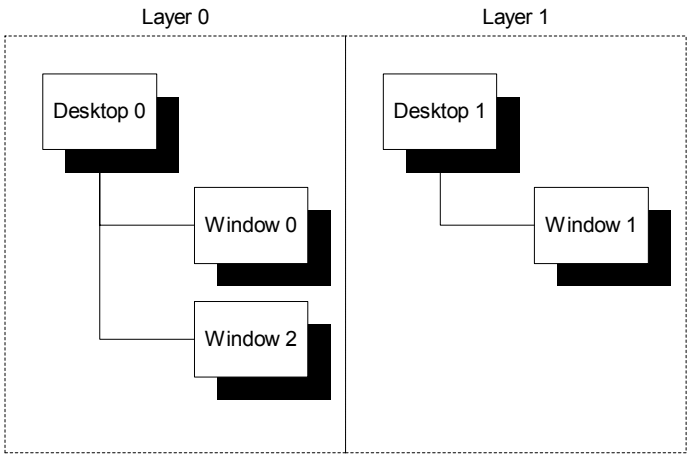
以下示例展示了如何将窗口附加到新的父窗口上：

```
/* Create 1 child window on destop 0 */
hWin0 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                                WM_GetDesktopWindowEx(0), WM_CF_SHOW, _cbWin0, 0);
/* Create 2 child windows on destop 1 */
hWin1 = WM_CreateWindowAsChild( 10, 20, 80, 70,
                                WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin1, 0);
hWin2 = WM_CreateWindowAsChild(110, 20, 80, 70,
                                WM_GetDesktopWindowEx(1), WM_CF_SHOW, _cbWin2, 0);
GUI_Delay(1000);
/* Detach window 2 from desktop 1 and attach it to desktop 0 */
WM_AttachWindow(hWin2, WM_GetDesktopWindowEx(0));
```

下表展示了在把窗口附加到新父窗口之前，上例中的屏幕截图和窗口层级关系：

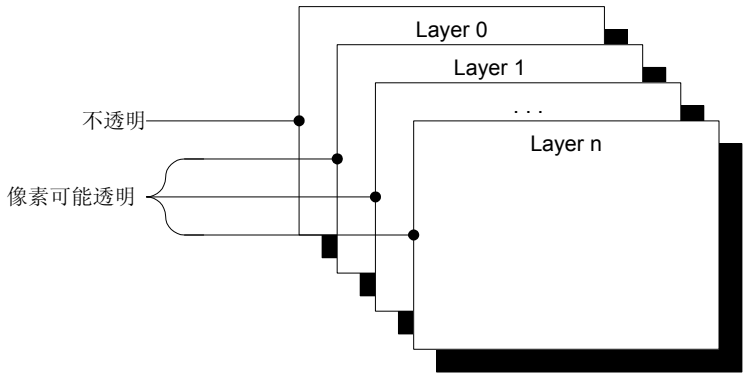
屏幕截图	窗口层级关系
	

下表展示了在把窗口附加到新父窗口之后，上例中的屏幕截图和窗口层级关系：

屏幕截图	窗口层级关系
	

21.2 使用多层支持

emWin 对多层和多显示不作区分。使用多层时，通常每一层的尺寸和驱动都是相同的。查看器用不同窗口显示每一层。查看器复合窗口显示所有图层。高索引图层位于低索引图层的上方，可以使用透明像素：



21.2.1 透明

透明是指，在大于 >0 的层中色指数为 0 的像素位置，可以看到背景层的颜色。对于第 0 层以外的所有层，指数 0 表示透明，因此，指数 0 不能用来显示颜色。这也意味着，色彩转换不能使 0 成为某种颜色的最佳匹配，因为这样会导致透明像素。这就是说，只得使用一些固定的调色板样式或定制调色板样式，而且在定义自己的调色板时必须特别注意。必须确保色彩转换 （24 位 RGB -> 指数）的结果不为 0。

固定调色板样式

86661 是目前支持透明的唯一可用固定调色板样式。有关详细信息，请参阅“颜色”（第 227 页）。

定制调色板样式

如果在大于 >0 的层中要使用定制调色板，则第一个颜色不得从色彩转换程序中调用。以下示例定义了一种拥有 15 个灰色标的定制调色板：

```
static const LCD_COLOR _aColors_16[] = {
  GUI_TRANSPARENT, 0x000000, 0x222222, 0x333333,
  0x444444, 0x555555, 0x666666, 0x777777,
  0x888888, 0x999999, 0xAAAAAA, 0xBBBBBB,
  0xCCCCCC, 0xDDDDDD, 0xEEEEEE, 0xFFFFFF
};

static const LCD_PHYSPALETTE _aPalette_16 = {
  16, _aColors_16

};
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  .
  .
  .
  //
  // Set user palette data (only required if no fixed palette is used)
  //
  LCD_SetLUTEx(1, _aPalette_16);
}
```

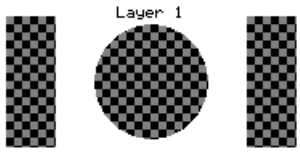
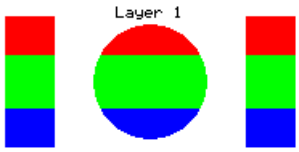
示例

以下示例展示了透明的使用方法。在第 0 层中绘制 3 个色条。第 1 层填充为白色，并绘制了 3 个透明项目。

```
GUI_SelectLayer(0);
GUI_SetColor(GUI_RED);
GUI_FillRect(0, 0, 199, 33);
GUI_SetColor(GUI_GREEN);
GUI_FillRect(0, 34, 199, 66);
GUI_SetColor(GUI_BLUE);
GUI_FillRect(0, 67, 199, 99);
GUI_SelectLayer(1);
GUI_SetBkColor(GUI_WHITE);
GUI_Clear();
GUI_SetColor(GUI_BLACK);
GUI_DispStringHCenterAt("Layer 1", 100, 4);
GUI_SetColor(GUI_TRANSPARENT);
GUI_FillCircle(100, 50, 35);
GUI_FillRect(10, 10, 40, 90);
GUI_FillRect(160, 10, 190, 90);
```

上例屏幕截图

下表展示了各层的内容及复合视图 （为显示结果）：

第 0 层	第 1 层	显示
		

21.2.2 Alpha 混合

Alpha 混合 (Alpha blending) 是将两种颜色相混合以产生透明效果的一种方法。假设根据 alpha 混合值 A（数值范围为 0 到 1，其中：0 表示不可见，1 表示 100% 可见）来混合 2 种颜色 C_0 和 C_1，则所产生的颜色 C_r 可借助以下等式计算：

$$C_r = C_0 * (1 - A) + C_1 * A$$

逻辑颜色内部处理为 32 位值。下 24 位用于颜色信息，上 8 位则用于管理 alpha 混合。若 alpha 值为 0x00，则表示不透明， 0xFF 表示完全透明 （不可见）。

不同的方法

alpha 信息有 3 种不同的管理方法：

- 层alpha混合：在采用层alpha混合的系统中，alpha值对层是固定的，可通过LCD_SetAlphaEx()函数设置。
- 查找表 (LUT)alpha 混合：这种 alpha 混合用 LUT 来管理 alpha 信息。
- 像素 alpha 混合：层中需要与背景结合的每个像素都由 alpha 信息构成。

固定调色板样式

对于 LUT alpha 混合，可使用 822216 和 84444 两种固定调色板样式。像素 alpha 混合仅受采用固定调色板样式 8888 的 32 bpp 样式支持。有关固定调色板样式的详情，请参见“颜色”（第 227 页）一章。

示例

以下示例展示了像素 alpha 混合的使用方法。在第 0 层中绘制一个圆，一个由水平线条构成的黄色三角形，加上 alpha 值构成的垂直梯度：

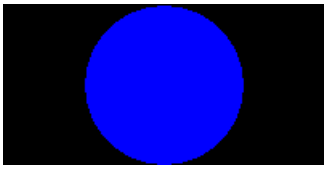
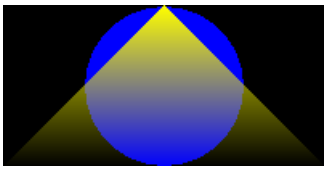
```
GUI_SetColor(GUI_BLUE);
GUI_FillCircle(100, 50, 49);
```



```
GUI_SelectLayer(1);
GUI_SetBkColor(GUI_TRANSPARENT);
GUI_Clear();
for (i = 0; i < 100; i++) {
  U32 Alpha;
  Alpha = (i * 255 / 100) << 24;
  GUI_SetColor(GUI_YELLOW | Alpha);
  GUI_DrawHLine(i, 100 - i, 100 + i);
}
```

上例屏幕截图

下表展示了各层的内容及复合视图 （为显示结果）：

第 0 层	第 1 层	显示
		

21.2.3 硬件游标

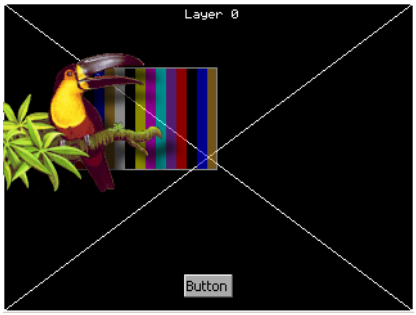
术语 “硬件游标” (Hardware cursor) 指在背景透明的独立层中使用游标图像。如果硬件支持多层并支持层定位，则可对 emWin 进行配置，以使用不同的层来管理游标。这种游标支持的主要优势在于可以获得更好的性能，因为移动时只需修改少量寄存器，并且能够对游标层中的图像进行定制。有关拖放的详细信息，请参阅 “GUI_AssignCursorLayer()”（第 755 页）。

21.2.4 多层示例

有关多层示例的详情，请参见 “模拟”（第 39 页）一章。另外，示例文件夹含有以下示例，展示了多层支持的使用方法：

- MULTILAYER_AlphaChromaMove.c

上述示例的屏幕截图



21.3 使用多显示支持

每个显示都可以有自己的驱动和设置。

21.3.1 启用多显示支持

为了启用多显示支持，必须在 GUIConf.h 中定义最大层数：

```
#define GUI_NUM_LAYERS 2 /* Enables support for 2 displays/layers */
```

另外需要为每一层创建和配置一个显示驱动器件。

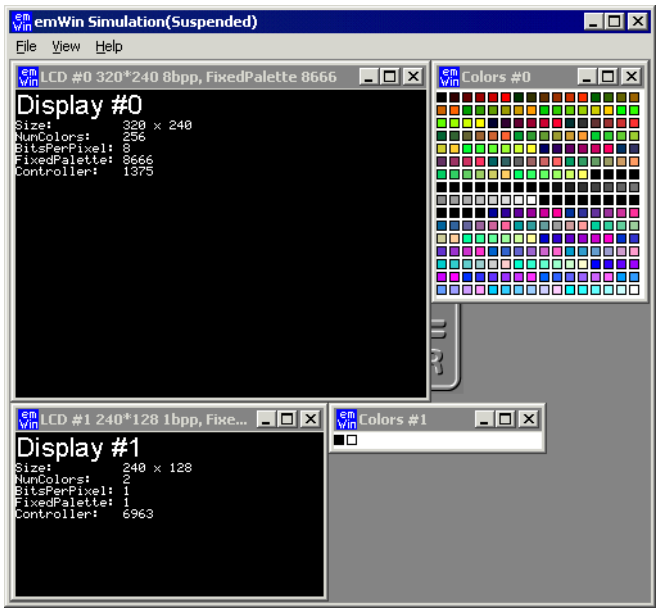
21.3.2 运行时间屏幕旋转

有些情况下，可能需要在运行过程中更改显示方向。通过多显示支持即可实现。此时， LCDConf.c 文件应含有针对每一种所需显示方向的显示配置。在此基础上，切换显示方向的方法如下：

- 通过 GUI_SelectLayer() 选择所需显示方向的配置。
- 如果旋转需要对显示控制器重新进行初始化，则需调用正确的驱动重新初始化函数。第 0 层为 LCD_L0_Init()，其他层为 LCD_L0_x_Init()，其中 “x” 表示零基配置索引。

21.3.3 多显示示例

以下示例展示了模拟的屏幕截图，其中有 2 个显示。第一显示为一个 8bpp 的彩色显示，尺寸为 320 x 240 像素。驱动为 LCD13XX.c，针对爱普生 S1D13705 LCD 控制器进行配置。第二显示为一个 1bpp 的黑白显示，尺寸为 240 x 128 像素。驱动为 LCDSlin.c，针对东芝 T6963 LCD 控制器进行配置：



21.4 配置多层支持

以上多层示例的 LCD 配置

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for first layer ...
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_16, // Display driver
                           GUICC_655,     // Color conversion
                           0, 0);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (0, 400,  234);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(0, (void *)0xc00000); // Video RAM start address
  //
  // Set display driver and color conversion for second layer ...
  //
```



```
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_8,  // Display driver
                           GUICC_86661,   // Color conversion
                           0, 1);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (1, 400,  234);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(1, (void *)0xc00000); // Video RAM start address
}
```

21.5 配置多显示支持

以上多显示示例的配置

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for first layer ...
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_8,  // Display driver
                           GUICC_8666,    // Color conversion
                           0, 0);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (0, 320,  240);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(0, (void *)0xc00000); // Video RAM start address
  //
  // Set display driver and color conversion for second layer ...
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_1,  // Display driver
                           GUICC_1,       // Color conversion
                           0, 1);
  //
  // ... and configure it
  //
  LCD_SetSizeEx    (1, 240,  128);        // Physical display size in pixels
  LCD_SetVRAMAddrEx(1, (void *)0x800000); // Video RAM start address
}
```

21.6 多层 API

下表按字母顺序列出了可用的多层相关程序。详细描述如下：

程序	描述
GUI_AssignCursorLayer()	指定管理硬件游标所使用的层。
GUI_SelectLayer()	选择输出操作的 layer/display。
GUI_SetLayerAlphaEx()	设置层 alpha 混合。
GUI_SetLayerPosEx()	设置给定层的位置。
GUI_SetLayerSizeEx()	设置给定层的尺寸。
GUI_SetLayerVisEx()	设置给定层的可见性。
LCD_GetNumLayers()	返回层数。

GUI_AssignCursorLayer()

描述

该函数指定用作游标层的层。

原型

```
void GUI_AssignCursorLayer(unsigned Index, unsigned CursorLayer);
```

参数	描述
索引	层索引。
CursorLayer	管理游标所使用的层。

其他信息

使用硬件游标则表示把某个层用作游标层。与默认游标处理相反（即绘制游标的视频存储器区与所有其他项目相同），硬件游标在不同的层中绘制。此时，emWin 将确保硬件游标的背景色设为透明，且选定的游标将绘制到该层中。
默认游标管理要求或多或少的计算时间，以绘制游标或管理背景，而移动硬件游标只需要修改几个寄存器即可。
需要注意的是，使用该函数要求显示驱动支持层定位。

GUI_SelectLayer()

描述

选择绘图操作所使用的层。

原型

```
unsigned int GUI_SelectLayer(unsigned int Index);
```

参数	描述
Index	层索引。

返回值

上一选定层的索引。

GUI_SetLayerAlphaEx()

描述

设置给定层的 alpha 混合。

原型

```
int GUI_SetLayerAlphaEx(unsigned Index, int Alpha);
```

参数	描述
Index	层索引。
Alpha	给定层的 alpha 混合值。

其他信息

如果要使用该函数，则硬件和所用显示驱动需要支持层 alpha 混合。如果驱动不支持这种功能，则函数将立即返回。
可用的 alpha 值范围取决于硬件。许多情况下，可用阿尔法值的范围有限，例如 0 - 0x3f。emWin 并不了解限制，会把给定值传递给驱动。应用需要负责确保给定的值未超出范围。

GUI_GetLayerPosEx()

描述

设置给定层的 X 位置和 Y 位置。

原型

```
void GUI_GetLayerPosEx(unsigned Index, GUI_POINT * pPos);
```

参数	描述
Index	层索引。
pPos	指向 GUI_POINT 结构 （含有待设置的位置）的指针。

其他信息

如果要使用该函数，则硬件和所用显示驱动需要支持层定位。如果驱动不支持这种功能，则函数将立即返回。

GUI_SetLayerSizeEx()

描述

设置给定层的 X 和 Y 尺寸。

原型

```
int GUI_SetLayerSizeEx(unsigned Index, int xSize, int ySize);
```

参数	描述
Index	层索引。
xSize	给定层新的水平尺寸 （单位：像素）。
ySize	给定层新的垂直尺寸 （单位：像素）。

其他信息

如果要使用该函数，则硬件和所用显示驱动需要支持层尺寸设置。如果驱动不支持这种功能，则函数将立即返回。

GUI_SetLayerVisEx()

描述

设置给定层的可见性。

原型

```
int GUI_SetLayerVisEx(unsigned Index, int OnOff);
```

参数	描述
Index	层索引。
OnOff	1 代表层可见， 0 则代表不可见。

其他信息

如果要使用该函数，则硬件和所用显示驱动需要支持该功能。如果驱动不支持这种功能，则函数将立即返回。

LCD_GetNumLayers()

描述

返回在配置文件中配置的层数。

原型

```
int LCD_GetNumLayers(void);
```

返回值

在配置文件中所配置的层数。

第 22 章

指针输入设备

emWin 支持指针输入设备，可以是触摸屏、鼠标或游戏操纵杆。emWin 基础版包括模拟触摸屏的驱动、PS2 鼠标驱动和示例游戏操纵杆驱动。只要有适当的驱动，也可以使用其他类型的触摸屏和鼠标设备。
输入设备的软件位于 GUI\Core 子目录中。

22.1 描述

指针输入设备指鼠标、触摸屏、游戏操纵杆等设备。单个应用中可以使用多个指针输入设备，以支持鼠标 / 触摸屏 / 游戏操纵杆的同时使用。一般地，指针输入设备 (PID) 驱动所做的是在检测到事件（如移动鼠标或者按下触摸屏等）时调用 GUI_PID_StoreState() 程序。
窗口管理器负责对 PID 事件作出正确反应；如果未使用窗口管理器，则由应用负责对 PID 事件作出反应。

22.2 指针输入设备 API

下表按字母顺序列出了指针输入设备程序。详细描述如下。
注：该 API 由 PID 驱动使用；如果使用 emWin 自带的 PID 驱动，则代码中无需调用这些程序。

程序	描述
GUI_PID_GetState()	返回 PID 的当前状态。
GUI_PID_StoreState()	存储 PID 的当前状态。

数据结构

pState 参数提到的 GUI_PID_STATE 类结构由程序用当前值填充。结构定义如下：

```
typedef struct {
  int x, y;
  U8  Pressed;
  U8  Layer;
} GUI_PID_STATE;
```

Elements of GUI_PID_STATE

数据类型	元素	描述
int	x	指针输入设备的 X 位置。
int	y	指针输入设备的 Y 位置。
U8	Pressed	若使用触摸屏，该值可为 0 （未按下）或 1 （按下）。 若使用鼠标，则 0 位用于左键的按下状态， 1 位用于右键。若按下按键，则两位为 1，否则为 0。
U8	Layer	用以描述 PID 状态的起始层

GUI_PID_GetState()

描述

返回是否按下输入设备，并用当前状态信息填充给定的 GUI_PID_STATE 结构。

原型

```
int GUI_PID_GetState(GUI_PID_STATE * pState);
```

参数	描述
pState	指向 GUI_PID_STATE 类结构 （由当前状态填充）的指针。

返回值

如果输入设备当前被按下，返回值为 1，否则返回值为 0。

示例

```
GUI_PID_STATE State;
GUI_PID_GetState(&State);
```

GUI_PID_StoreState()

描述

存储指针输入设备的当前状态。

原型

```
void GUI_PID_StoreState(const GUI_PID_STATE * pState);
```

参数	描述
pState	指向 GUI_PID_STATE 类结构的指针。

其他信息

该函数可从中断服务程序调用。
emWin 的 PID 输入管理器含有一个 FIFO 缓冲器，默认情况下最多可以保存 5 个 PID 事件。如果需要不同的尺寸，可以更改该值。更多详细信息，请参阅 “高级 GUI 配置选项”（第 914 页）。

22.3 鼠标驱动

鼠标支持由两 “层” 构成：一个通用层和一个鼠标驱动层。通用程序指始终存在的函数，与所用鼠标驱动的类型无关。另一方面，可用的鼠标驱动则会根据必要调用相应的通用程序，并且只能配合随 emWin 一起提供的 PS2 鼠标驱动使用。如果要写自己的驱动，则该驱动负责调用通用程序。而通用鼠标程序则会调用相应的 PID 程序。

22.3.1 通用鼠标 API

下表按字母顺序列出了通用鼠标程序。这些函数可以配合任何类型的鼠标驱动使用。详细描述如下。

程序	描述
GUI_MOUSE_GetState()	返回鼠标的当前状态。
GUI_MOUSE_StoreState()	存储鼠标的当前状态。

GUI_MOUSE_GetState()

描述

返回鼠标的当前状态。

原型

```
int GUI_MOUSE_GetState(GUI_PID_STATE * pState);
```

参数	描述
pState	指向 GUI_PID_STATE 类结构的指针。

返回值

如果鼠标当前被按下，返回值为 1，否则返回值为 0。

其他信息

该函数将调用 GUI_PID_GetState()。

GUI_MOUSE_StoreState()

描述

存储鼠标的当前状态。

原型

```
void GUI_MOUSE_StoreState(const GUI_PID_STATE *pState);
```

参数	描述
pState	指向 GUI_PID_STATE 类结构的指针。

其他信息

该函数将调用 GUI_PID_StoreState()。
该函数可从中断服务程序调用。

示例

```
GUI_PID_STATE State;
State.x = _MousepositionX; /* Screen position in X of mouse device */
State.y = _MousepositionY; /* Screen position in Y of mouse device */
State.Pressed = 0;
if (_LeftButtonPressed) {
  State.Pressed |= 1;      /* Set bit 0 if left button is pressed */
}
if (_RightButtonPressed) {
  State.Pressed |= 2;      /* Set bit 1 if right button is pressed */
}
GUI_MOUSE_StoreState(&State);
```

22.3.2 PS2 鼠标驱动

该驱动支持任何类型的 PS2 鼠标。

22.3.2.1 使用 PS2 鼠标驱动

该驱动使用非常方便。在启动代码中，需调用 init 函数 GUI_MOUSE_DRIVER_PS2_Init()。
在从鼠标收到一个字节时，应用需要以某种方式予以注意。此时，应调用 GUI_MOUSE_DRIVER_PS2_OnRx() 函数，并将收到的字节作为参数传递给该函数。然后，驱动根据需要调用 GUI_PID_StoreState。
字节的接收一般通过中断服务程序处理。

以下是一种可能的 ISR 示例：（注意，不同系统当然存在差异）

```
void interrupt OnRx(void) {
  char Data;
  Data  = UART_REG;                  // Read data from the hardware
  GUI_MOUSE_DRIVER_PS2_OnRx(Data);   // Pass it on to the driver
}
```

22.3.2.2 PS2 鼠标驱动

下表以字母顺序列出了可用的鼠标驱动。

程序	描述
GUI_MOUSE_DRIVER_PS2_Init()	初始化鼠标驱动。
GUI_MOUSE_DRIVER_PS2_OnRx()	由接收中断程序调用。

GUI_MOUSE_DRIVER_PS2_Init()

描述

初始化鼠标驱动。

原型

```
void GUI_MOUSE_DRIVER_PS2_Init(void);
```

GUI_MOUSE_DRIVER_PS2_OnRx()

描述

必须由接收中断程序调用。

原型

```
void GUI_MOUSE_DRIVER_PS2_OnRx(unsigned char Data);
```

参数	描述
Data	ISR 收到的数据字节。

其他信息

PS2 鼠标驱动是一种串口驱动，即是说它一次只能接收 1 个字节。
必须确保在每次收到一个字节 （1 个字符）时，通过接收中断程序调用该函数。

22.4 触摸屏驱动

如前所述，触摸屏驱动一般会简单地调用 GUI_PID_StoreState()。这种方式可以支持任何类型的触摸屏。您所要做的就是编写驱动代码 （一般来说都很简单）。
连接触摸屏最常见的方式是采用提供有驱动的 4 引脚模拟接口来进行连接。

22.4.1 通用型触摸屏 API

通用型触摸屏 API 可用于任何类型的驱动 （模拟、数字等）。驱动根据需要调用相应的程序。如果要编写自己的驱动，则该驱动需要调用通用程序。
下表按字母顺序列出了通用触摸屏程序。这些函数可以配合任何类型的触摸屏驱动使用。详细描述如下。

程序	描述
GUI_TOUCH_GetState()	返回触摸屏的当前状态。
GUI_TOUCH_StoreState()	用 X 和 Y 坐标存储触摸屏的当前状态。
GUI_TOUCH_StoreStateEx()	存储触摸屏的当前状态。

GUI_TOUCH_GetState()

描述

返回触摸屏的当前状态。

原型

```
int GUI_TOUCH_GetState(GUI_PID_STATE *pState);
```

参数	描述
pState	指向 GUI_PID_STATE 类结构的指针。

返回值

如果触摸屏当前被按下，返回值为 1，否则返回值为 0。

GUI_TOUCH_StoreState()

描述

以 X 和 Y 坐标为参数存储触摸屏的当前状态。

原型

```
void GUI_TOUCH_StoreState(int x, int y);
```

参数	描述
x	X 位置。
y	Y 位置。

其他信息

如果给定值之一为负数，则 GUI 会假定触摸屏未被按下。
该函数可从中断服务程序调用。
有关触摸屏处理程序的详细示例，请参见 Sample\GUI_X\GUI_X_Touch_StoreState.c。

示例

```
int x, y;
if (_TouchIsPressed) {
  x = _TouchPositionX; /* Current position in X of touch device */
  y = _TouchPositionY; /* Current position in Y of touch device */
} else {
  x = y = -1;          /* Use -1 if touch is not pressed */
}
GUI_TOUCH_StoreState(x, y);
```

GUI_TOUCH_StoreStateEx()

描述

存储触摸屏的当前状态。

原型

```
void GUI_TOUCH_StoreStateEx(const GUI_PID_STATE * pState);
```

参数	描述
pState	指向 GUI_PID_STATE 类结构的指针。

其他信息

该函数将调用 GUI_PID_GetState()。
有关触摸屏处理程序的详细示例，请参见 Sample\GUI_X\GUI_X_Touch_StoreState.c。

示例

```
GUI_PID_STATE State;
State.x = _TouchPositionX;
State.y = _TouchPositionY;
if (_TouchIsPressed) {
  State.Pressed = 1;
} else {
  State.Pressed = 0;
}
GUI_TOUCH_StoreStateEx(&State);
```

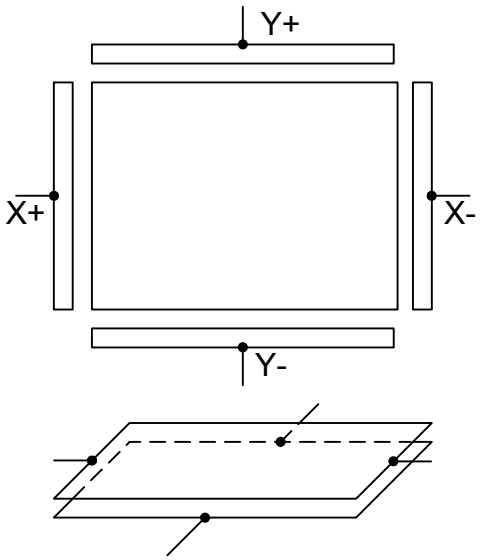
22.4.2 模拟触摸屏驱动

emWin 触摸屏驱动处理触摸屏的模拟输入（通过 8 位或更好的 A/D 转换器）、去抖和校准。

触摸屏驱动通过 GUI_TOUCH_Exec() 函数持续监控和更新触控面板，该函数在意识到执行了某项动作或发生什么变化时，会调用适当的通用触摸屏 API 程序。

模拟触摸屏的工作原理

触摸屏由 2 个超薄的玻璃传导层构成，二者之间一般是相互绝缘的。如果用户按下触控面板，则两层在该点处连接。如果向 Y 层应用一个电压，则在按下时，可以在 X+/X- 端电极处测到一个电压。该电压取决于触摸位置。同理，如果向 X 层应用一个电压，则在按下时，可以在 Y+/Y- 端电极处测到一个电压。



22.4.2.1 设置模拟触摸屏

触控面板的准备应遵循以下步骤：

- 应用硬件程序
- 对 GUI_TOUCH_Exec() 应用定期调用
- 用示波器验证工作是否正常
- 通过示例来确定校准值
- 使用上一步确定的值，在初始化程序 LCD_X_Config() 中添加一个对 GUI_TOUCH_Calibrate() 的调用

下面对每一步进行详细描述。

应用硬件程序

应用触摸屏的第一步应该是用代码填写硬件程序。这些程序为：

```
GUI_TOUCH_X_ActivateX(), GUI_TOUCH_X_ActivateY()
GUI_TOUCH_X_MeasureX(), GUI_TOUCH_X_MeasureY(
```

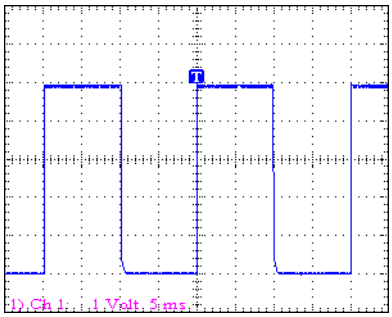
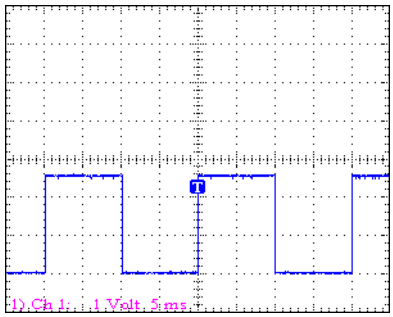
含有空白程序的模块 GUI_TOUCH_X.c 位于文件夹 Sample\GUI_X 中。您可以从这个模块开始。激活程序应接通测量电压以准备进行测量。例如，GUI_TOUCH_X_ActivateX() 应接通 X 的测量电压来准备对 Y 的测量。另外，该程序还应切断 Y 的电压并禁用对 X 的测量。
测量程序应返回 A/D 转换器的测量结果。本章后面将就硬件程序的应用提供一个示例。

对 GUI_TOUCH_Exec() 应用定期调用

应用触摸屏的第二步是确保 GUI_TOUCH_Exec() 函数被定期调用。应用应当每秒调用大约 100 次。如果使用的是实时操作系统，则确保该函数被定期调用最简单的办法是创建一个独立的任务。在未使用多任务系统的情况下，可以使用中断服务程序来完成这项工作。

用示波器验证工作是否正常

应用 GUI_TOUCH_Exec() 的调用之后，要确保硬件能正常工作。最简单的办法是用示波器测量触控面板的电源电压和测量电压。下表展示了一种典型结果。第一栏显示的是一个轴的电源电压，第二栏是在按下触控面板中间时测量到的测量电压。

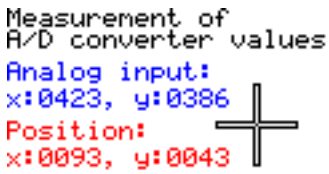
电源电压	测量电压
	

通过示例来确定校准值

第三步是取得 A/D 转换器的最小值和最大值。emWin 需要用这些值来把测量结果转换为以像素表示的触摸位。这 4 个值为：

值	获取方法
GUI_TOUCH_AD_TOP	按下触摸屏的顶部，写下 Y 轴模拟输入值。
GUI_TOUCH_AD_BOTTOM	按下触摸屏的底部，写下 Y 轴模拟输入值。
GUI_TOUCH_AD_LEFT	按下触摸屏的左侧，写下 X 轴模拟输入值。
GUI_TOUCH_AD_RIGHT	按下触摸屏的右侧，写下 X 轴模拟输入值。

emWin 的示例文件夹中有一个小程序，可用来获取触控面板的这些值。该程序位于文件夹Sample\Tutorial下，程序名称为TOUCH_Sample.c。在硬件上运行该示例。其输出应类似于右侧的屏幕截图。



以上述值调用 GUI_TOUCH_Calibrate()

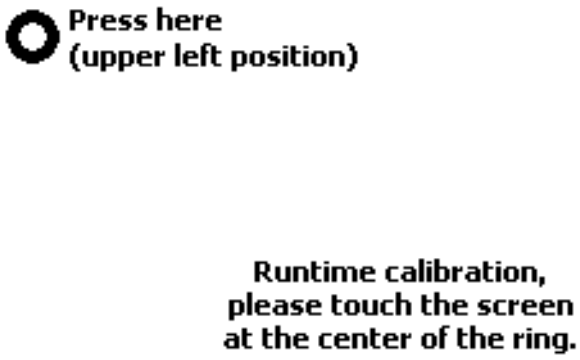
最后一步是用校准值添加对 GUI_TOUCH_Calibrate() 函数的调用。校准触摸屏的建议程序是初始化程序 LCD_X_Config()，位于 LCDConf.c 中，与以下示例相似：

```
#define GUI_TOUCH_AD_TOP     877
#define GUI_TOUCH_AD_BOTTOM  273
#define GUI_TOUCH_AD_LEFT    232
#define GUI_TOUCH_AD_RIGHT   918
.
.
.
void LCD_X_Config(void) {
  //
  // Initialize display driver
  //
  .
  .
  .
  //
  // Set orientation of touch screen (only required when using
  //
  TouchOrientation = (GUI_MIRROR_X * LCD_GetMirrorX()) |
                     (GUI_MIRROR_Y * LCD_GetMirrorY()) |
                     (GUI_SWAP_XY  * LCD_GetSwapXY()) ;
```

```
  GUI_TOUCH_SetOrientation(TouchOrientation);
  //
  // Calibrate touch screen
  //
  GUI_TOUCH_Calibrate(GUI_COORD_X, 0, 240, TOUCH_AD_TOP , TOUCH_AD_BOTTOM);
  GUI_TOUCH_Calibrate(GUI_COORD_Y, 0, 320, TOUCH_AD_LEFT, TOUCH_AD_RIGHT);
}
```

22.4.2.2 运行时校准

实际上，配置文件的确切值只能按照触控面板来确定。由于同一系列的器件之间存在细微差异，因此有必要在运行过程中对每一台设备进行校准。这可以通过 GUI_TOUCH_Calibrate() 函数来得以实现。示例文件夹含有示例 TOUCH_Calibrate.c，其中展示了如何在运行过程中校准触摸屏：



22.4.2.3 硬件程序

如果使用随 emWin 一起提供的驱动，则需把以下四个因硬件而异的函数添加到项目中，因为 GUI_TOUCH_Exec() 会在论询触控面板时调用它们。建议位置是在文件 GUI_X.c 中。功能如下：

程序	描述
GUI_TOUCH_X_ActivateX()	准备 Y 轴测量。
GUI_TOUCH_X_ActivateY()	准备 X 轴测量。
GUI_TOUCH_X_MeasureX()	返回 A/D 转换器的 X 轴测量结果。
GUI_TOUCH_X_MeasureY()	返回 A/D 转换器的 Y 轴测量结果。

GUI_TOUCH_X_ActivateX(), GUI_TOUCH_X_ActivateY()

描述

这些程序由GUI_TOUCH_Exec()调用，用于激活X轴和Y轴的测量。GUI_TOUCH_X_ActivateX()接通通向 X 轴的测量电压； GUI_TOUCH_X_ActivateY() 接通通向 Y 轴的测量电压。接通 X 轴的电压表示可以测量 Y 轴的值，反之亦然。

原型

```
void GUI_TOUCH_X_ActivateX(void);
void GUI_TOUCH_X_ActivateY(void);
```

GUI_TOUCH_X_MeasureX(), GUI_TOUCH_X_MeasureY()

描述

这些程序由 GUI_TOUCH_Exec() 调用，返回 A/D 转换器测得的 X 轴和 Y 轴的测量值。

原型

```
int GUI_TOUCH_X_MeasureX(void);
int GUI_TOUCH_X_MeasureY(void);
```

示例应用

以下示例展示了针对一种三菱 M16C/80 控制器的触摸硬件程序的应用方式：

```
void GUI_TOUCH_X_ActivateX(void) {
  U8 Data;
  asm("fclr i");                  /* Disable interrupts            */
  Data  = P10;                    /* Read port data                */
  Data |=   (1 << 2) | (1 << 3);  /* Switch on  power in X
                                     and  enable measurement in Y  */
  Data &= ~((1 << 4) | (1 << 5)); /* Switch off power in Y
                                     and disable measurement in X  */
  P10   = Data;                   /* Write port data               */
  asm("fset i");                  /* Enable interrupts             */
}

void GUI_TOUCH_X_ActivateY(void) {
  U8 Data;
  asm("fclr i");                  /* Disable interrupts            */
  Data  = P10;                    /* Read port data                */
  Data |=   (1 << 5) | (1 << 4);  /* Switch on power in Y
                                     and  enable measurement in X  */
  Data &= ~((1 << 3) | (1 << 2)); /* Switch off power in X
                                     and disable measurement in Y  */
  P10   = Data;                   /* Write port data               */
  asm("fset i");                  /* Enable interrupts             */
}

static void ReadADCx(int channel) {
  ADCON0  = channel               /* Select channel 0-7            */
          | (0 << 3)              /* One shot mode                 */
          | (0 << 6)              /* A-D conversion start (0=stop) */
          | (0 << 7);             /* FAD/4 select                  */
  ADCON1  =  (0 << 0)             /* A-D sweep select (XX)         */
          | (0 << 2)              /* No sweep mode                 */
          | (0 << 3)              /* 8 bit mode                    */
          | (0 << 4)              /* FAD4 select                   */
          | (1 << 5)              /* VRef connected                */
          | (0 << 6);             /* Anex0/1 not used              */
  ADCON2  =  (1 << 0);            /* Use example and hold          */
  ADIC    = 0;                    /* Reset IR flag                 */
  ADCON0 |= (1 << 6);             /* Start conversion              */
  while ((ADIC & (1 << 3)) == 0); /* Wait for end of conversion    */
  ADCON0 &= ~(6 << 0);            /* Start conversion = 0          */
}

int GUI_TOUCH_X_MeasureX(void) {
  ReadADCx(0);
  return AD0;
}

int GUI_TOUCH_X_MeasureY(void) {
  ReadADCx(1);
  return AD1;
}
```


22.4.2.4 模拟触摸屏的驱动 API

下表按字母顺序列出了可用的模拟触摸屏驱动。这些函数仅在使用 emWin 所附带的驱动时有效。

程序	描述
GUI_TOUCH_Calibrate()	更改校准。
GUI_TOUCH_Exec()	激活 X 轴和 Y 轴的测量，调用频率应为每秒 100 次左右。
GUI_TOUCH_SetOrientation()	设置逻辑显示方向。

GUI_TOUCH_Calibrate()

描述

在运行过程中更改校准。

原型

```
int GUI_TOUCH_Calibrate(int Coord, int Log0, int Log1,
                        int Phys0, int Phys1);
```

参数	描述
Coord	GUI_COORD_X，X 轴；GUI_COORD_Y，Y 轴。
Log0	逻辑值 0（单位：像素）。
Log1	逻辑值 1（单位：像素）。
Phys0	Log0 的 A/D 转换器值。
Phys1	Log1 的 A/D 转换器值。

其他信息

该函数的参数为待校准的轴、该轴的两个逻辑值（单位为像素）以及 A/D 转换器的两个对应物理值。

GUI_TOUCH_Exec()

描述

通过调用 TOUCH_X 程序以激活 X 轴和 Y 轴的测量，从而实现对触摸屏的论询。必须确保该函数的调用频率为每秒 100 次左右。

原型

```
void GUI_TOUCH_Exec(void);
```

其他信息

如果使用的是实时操作系统，则确保该函数被定期调用最简单的办法是创建一个独立的任务。在未使用多任务系统的情况下，可以使用中断服务程序来完成这项工作。

GUI_TOUCH_SetOrientation()

描述

该函数配置触摸屏方向。如果触摸屏已经配置为支持默认方向，但现在需要翻转显示或者镜像显示，则可以使用该函数来配置触摸驱动，使其使用与显示相同的方向，而不对硬件程序进行任何改动。

原型

```
void GUI_TOUCH_SetOrientation(unsigned Orientation);
```

参数	描述
Orientation	下表中的一个或多个 “OR” 组合值。

参数 Orientation 的允许值	
GUI_MIRROR_X	对 X 轴进行镜像显示
GUI_MIRROR_Y	对 Y 轴进行镜像显示
GUI_SWAP_XY	交换 X 轴和 Y 轴

22.4.2.5 配置模拟触摸屏驱动

触摸屏驱动完全支持运行时配置。需使用 GUI_TOUCH_Calibrate() 来为每个轴的 2 个位置指定 A/D 转换器返回的物理值。如果需要翻转显示或镜像显示，则可使用GUI_TOUCH_SetOrientation() 来设定新的显示方向，而不对硬件程序进行任何改动。
在 emWin 管理任何触摸输入之前，必须对触摸屏进行配置。

示例

```
#define TOUCH_AD_LEFT   0x3c0
#define TOUCH_AD_RIGHT  0x034
#define TOUCH_AD_TOP    0x3b0
#define TOUCH_AD_BOTTOM 0x034

Orientation = (GUI_MIRROR_X * LCD_GetMirrorXEx(0)) |
              (GUI_MIRROR_Y * LCD_GetMirrorYEx(0)) |
              (GUI_SWAP_XY  * LCD_GetSwapXYEx (0));
GUI_TOUCH_SetOrientation(Orientation);
GUI_TOUCH_Calibrate(GUI_COORD_X, 0, 239, TOUCH_AD_LEFT, TOUCH_AD_RIGHT);
GUI_TOUCH_Calibrate(GUI_COORD_Y, 0, 319, TOUCH_AD_TOP,  TOUCH_AD_BOTTOM);
```

22.5 游戏操纵杆输入示例

以下示例展示了如何使用指针输入设备 API 来处理来自游戏操纵杆的输入：

```
/*********************************************************************
*
*         _JoystickTask
*
* Purpose:
*   Periodically read the Joystick and inform emWin using
*   GUI_PID_StoreState.
*   It supports dynamic acceleration of the pointer.
*   The Joystick is a simple, standard 5 switch (digital) type.
*
*/
static void _JoystickTask(void) {
  int Stat;
  int StatPrev = 0;
  int TimeAcc = 0;   // Dynamic acceleration value
  while(1) {
    Stat = HW_ReadJoystick();
    //
    // Handle dynamic pointer acceleration
    //
    if (Stat == StatPrev) {
      if (TimeAcc < 10) {
        TimeAcc++;
      }
    } else {
      TimeAcc = 1;
    }
    if (Stat || (Stat != StatPrev)) {
      GUI_PID_STATE State;
      int Max;
      //
      // Compute the new coordinates
      //
      GUI_PID_GetState(&State);
      if (Stat & JOYSTICK_LEFT) {
        State.x -= TimeAcc;
      }
      if (Stat & JOYSTICK_RIGHT) {
        State.x += TimeAcc;
      }
      if (Stat & JOYSTICK_UP) {
        State.y -= TimeAcc;
      }
      if (Stat & JOYSTICK_DOWN) {
        State.y += TimeAcc;
      }
      //
      // Make sure coordinates are still in bounds
      //
      if (State.x < 0) {
        State.x = 0;
      }
      if (State.y < 0) {
        State.y = 0;
      }
      Max = LCD_GetXSize() - 1;
      if (State.x >= Max) {
        State.x = Max;
      }
      Max = LCD_GetYSize() - 1;
      if (State.y > Max) {
        State.y = Max;
      }
      //
      // Inform emWin
      //
      State.Pressed = (Stat & JOYSTICK_ENTER) ? 1: 0;
      GUI_PID_StoreState(&State);
      StatPrev = Stat;
    }
    OS_Delay(40);
  }
}
```


第 23 章

键盘输入

emWin 可以支持任何类型的键盘。 emWin 兼容任何类型的键盘驱动。
键盘输入的软件位于 GUI\Core 子目录中，同时也是基本版的一部分。

23.1 描述

键盘输入设备使用 ASCII 字符编码，以便区分不同的字符。例如，键盘上只有一个 “A” 键，但大写的 “A” 与小写的 “a” 拥有不同的 ASCII 编码 （分别为 0x41 和 0x61）。

emWin 预定义字符编码

emWin 同时定义了其他 “虚拟” 键盘操作的字符编码。这些编码如下表所示，由 GUI.h 中的标识符表中定义。因此，在 emWin 中，字符编码可以是任意扩展 ASCII 字符值，也可以是任意下列预定义的 emWin 值。

预定义的虚拟键码	描述
GUI_KEY_BACKSPACE	退格键。
GUI_KEY_TAB	制表键。
GUI_KEY_ENTER	回车键。
GUI_KEY_LEFT	左箭头键。
GUI_KEY_UP	上箭头键。
GUI_KEY_RIGHT	右箭头键。
GUI_KEY_DOWN	下箭头键。
GUI_KEY_HOME	本位键 （移至当前行的开头）。
GUI_KEY_END	结束键 （移至当前行的末尾）。
GUI_KEY_SHIFT	换挡键。
GUI_KEY_CONTROL	控制键。
GUI_KEY_ESCAPE	换码键。
GUI_KEY_INSERT	插入键。
GUI_KEY_DELETE	删除键。

23.1.1 驱动层 API

键盘驱动层处理键盘消息函数。这些程序会在具体键 （或组合键）被按下或松开时通知窗口管理器。下表按字母顺序列出了驱动层键盘程序。详细描述如下。

程序	描述
GUI_StoreKeyMsg()	把消息存储于指定键。
GUI_SendKeyMsg()	把消息发送至指定键。

GUI_StoreKeyMsg()

描述

把消息数据 (Key, PressedCnt) 存进键盘缓冲器。

原型

```
void GUI_StoreKeyMsg(int Key, int Pressed);
```

参数	描述
Key	可以是任意扩展 ASCII 字符 （范围为 0x20 至 0xFF）或者任意预定义的 emWin 字符编码。
Pressed	键状态 （见下表）。

参数 Pressed 的允许值	
1	按下状态。
0	松开 （未按下）状态。

其他信息

该函数可从中断服务程序调用。
emWin 的键盘输入管理器含有一个 FIFO 缓冲器，默认情况下最多可以保存 10 个键盘事件。如果需要不同的尺寸，可以更改该值。更多详细信息，请参阅 “高级 GUI 配置选项”（第 914 页）。

GUI_SendKeyMsg()

描述

把键盘数据发送到输入焦点所在窗口。如果没有窗口有输入焦点，则调用 GUI_StoreKeyMsg() 函数将数据存储至输入缓冲器之中。

原型

```
void GUI_SendKeyMsg(int Key, int Pressed);
```

参数	描述
Key	可以是任意扩展 ASCII 字符 （范围为 0x20 至 0xFF）或者任意预定义的 emWin 字符编码。
Pressed	键状态 （见 GUI_StoreKeyMsg()）。

其他信息

该函数不可从中断服务程序调用。

23.1.2 应用层 API

下表按字母顺序列出了应用层键盘程序。详细描述如下。

程序	描述
GUI_ClearKeyBuffer()	清除键缓冲器。
GUI_GetKey()	返回键缓冲器中的内容。
GUI_StoreKey()	把键存储于缓冲器中。
GUI_WaitKey()	等待键被按下。

GUI_ClearKeyBuffer()

描述

清除键缓冲器。

原型

```
void GUI_ClearKeyBuffer(void);
```

GUI_GetKey()

描述

返回键缓冲器的当前内容。

原型

```
int GUI_GetKey(void);
```

返回值

键缓冲器中的字符编码；若未缓冲键，则返回值为 0。

GUI_StoreKey()

描述

把键存储于缓冲器中。

原型

```
void GUI_StoreKey(int Key);
```

参数	描述
Key	可以是任意扩展 ASCII 字符 （范围为 0x20 至 0xFF）或者任意预定义的 emWin 字符编码。

其他信息

该函数一般由驱动调用，而不是由应用本身调用。

GUI_WaitKey()

描述

等待键被按下。

原型

```
int GUI_WaitKey(void);
```

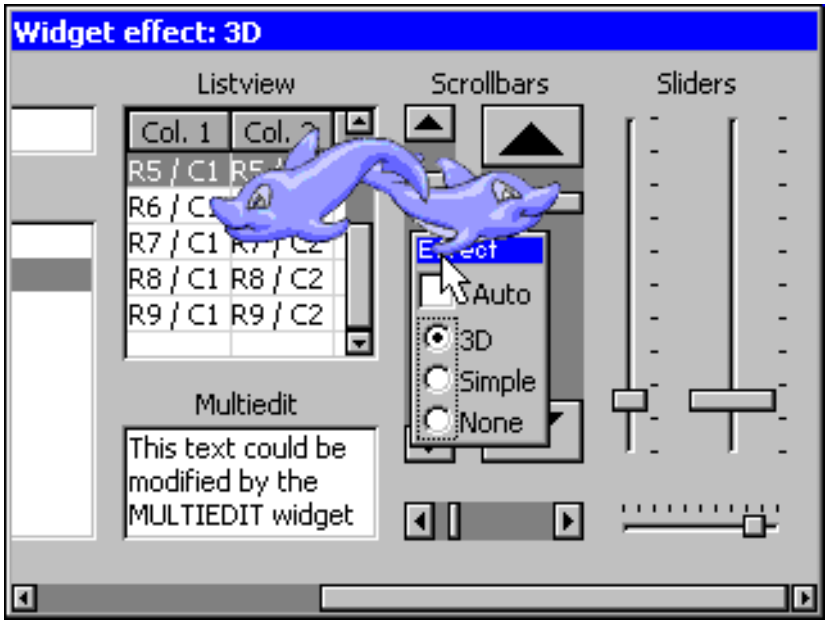
其他信息

应用被 “阻止”，也就是说，直到某个键被按下才会返回值。

第 24 章

Sprites

“sprite”这幅图像，可以显示在屏幕上的所有其他图形的上方。 Sprite 可以保护其所覆盖的屏幕区域。可以随时移动或者消除，从而完全恢复屏幕内容。可通过使用多幅图像来实现动画效果。
Sprite 完全独立于所有其他绘图操作和窗口操作：Sprite 不会影响绘图操作或窗口操作；绘图操作或窗口操作也不会影响 sprite。
可以把 Sprite 看作位于屏幕顶层的对象，类似于游标。



24.1 简介

emWin sprite 作为一种纯粹的软件解决方案实现。使用 emWin sprite 不需要额外的硬件。它们可以显示、移动和删除，而不影响当前可见的图形项目。

存储器要求

每个 sprite 都需要一个存储器区，用于保存 sprite 后面的显示数据，以便在移动或删除 sprite 时能恢复背景显示。另外需要一个存储区，用作颜色缓存。颜色缓存的大小取决于 sprite 图像中使用的颜色数量。因此， sprite 需要的全部字节数可以按以下等式计算：

```
SizeOfSpriteObject (~30 bytes) +
(XSize * YSize + NumberOfBitmapColors) * REQUIRED_BYTES_PER_PIXEL
```

Sprite 的最大数目

emWin 并不限制同时可见的 sprite 的数量。具体取决于可用存储器的大小。

性能

需要注意的是，绘制 sprite 比绘制简单的位图需要耗费更多计算资源，因为需要处理背景数据以及与其他 sprite 的交叉点。

叠置顺序

叠置顺序是对有重叠的二维对象 （此处为 sprite）的排序。当两个 sprite 重叠时，它们的叠置顺序决定了哪个 sprite 位于另一个 sprite 的上方。最后创建的 sprite 位于最上面。

24.2 Sprite API

下表按字母顺序列出了可用的 sprite 相关程序。详细描述如下：

程序	描述
GUI_SPRITE_Create()	创建 sprite。
GUI_SPRITE_CreateEx()	在给定层中创建 sprite。
GUI_SPRITE_Delete()	删除 sprite。
GUI_SPRITE_GetState()	返回 sprite 是否可见。
GUI_SPRITE_Hide()	隐藏 sprite。
GUI_SPRITE_SetBitmap()	设置 sprite 的新位图。
GUI_SPRITE_SetBitmapAndPosition()	设置 sprite 的新位图和位置。
GUI_SPRITE_SetPosition()	设置 sprite 的位置。
GUI_SPRITE_Show()	显示给定的 sprite。

GUI_SPRITE_Create()

描述

在当前层中的给定位置创建 sprite。

原型

```
GUI_HSPRITE GUI_SPRITE_Create(const GUI_BITMAP GUI_UNI_PTR * pBM,
                              int x, int y);
```

参数	描述
pBM	指向位图结构 （用于绘制 sprite）的指针。
x	Sprite 在屏幕坐标中的 X 位置。
y	Sprite 在屏幕坐标中的 Y 位置。

返回值

新 sprite 的句柄，如果失败，则返回值为 0。

其他信息

pBM 参数指向的位图需要符合以下要求：

- 不得压缩。
- 必须透明。
- 应是基于调色板的位置，为 1、 2、 4 或 8bpp。

其他位图或存储器不够会导致该函数失败。

GUI_SPRITE_CreateEx()

描述

在目标层中的给定位置创建 sprite。

原型

```
GUI_HSPRITE GUI_SPRITE_CreateEx(const GUI_BITMAP GUI_UNI_PTR * pBM,
                                int x, int y, int Layer);;
```

参数	描述
pBM	指向位图结构 （用于绘制 sprite）的指针。
x	Sprite 在屏幕坐标中的 X 位置。
y	Sprite 在屏幕坐标中的 Y 位置。
Layer	sprite 的层。

返回值

新 sprite 的句柄，如果失败，则返回值为 0。

GUI_SPRITE_Delete()

描述

删除给定的 sprite。

原型

```
void GUI_SPRITE_Delete(GUI_HSPRITE hSprite);
```

参数	描述
hSprite	待删除的 sprite 的句柄。

其他信息

该函数从存储器中删除 sprite，并自动恢复背景。

GUI_SPRITE_GetState()

描述

返回 sprite 是否可见。

原型

```
int GUI_SPRITE_GetState(GUI_HSPRITE hSprite);
```

参数	描述
hSprite	sprite 的句柄。

返回值

如果可见，返回值为 1，否则返回值为 0。

GUI_SPRITE_Hide()

描述

隐藏给定的 sprite。

原型

```
void GUI_SPRITE_Hide(GUI_HSPRITE hSprite);
```

参数	描述
hSprite	待隐藏的 sprite 句柄。

其他信息

该函数从可见 sprite 列表中删除给定的 sprite。

GUI_SPRITE_SetBitmap()

描述

为绘制 sprite 设置新图像。

原型

```
int GUI_SPRITE_SetBitmap(GUI_HSPRITE hSprite,
                         const GUI_BITMAP GUI_UNI_PTR * pBM);
```

参数	描述
hSprite	sprite 的句柄。
pBM	指向位图结构 （用于绘制 sprite）的指针。

返回值

如果成功，返回值为 0 ；如果程序失败，则返回值为 1。

其他信息

新的位图必须具有与上一位图完全相同的尺寸。把指针传向尺寸不同的位图会导致函数失败。
函数会立即替换屏幕上的可见 sprite 图像。显示新图像不需要进行其他操作。

GUI_SPRITE_SetBitmapAndPosition()

描述

同时设置位置和图像。

原型

```
int GUI_SPRITE_SetBitmapAndPosition(GUI_HSPRITE hSprite,
                                    const GUI_BITMAP GUI_UNI_PTR * pBM,
```



```
                                    int x, int y);
```

参数	描述
hSprite	sprite 的句柄。
pBM	指向新位图结构 （用于绘制 sprite）的指针。
x	在屏幕坐标中的新的 X 位置。
y	在屏幕坐标中的新的 Y 位置。

其他信息

先后使用 GUI_SPRITE_SetBitmap() 函数和 GUI_SPRITE_SetPosition() 函数，或者使用本函数，结果是不一样的。调用 GUI_SPRITE_SetBitmap() 和 GUI_SPRITE_SetPosition() 时，屏幕上的图像会渲染两次；使用本函数时，只会渲染一次，动画中可以善加利用。

GUI_SPRITE_SetPosition()

描述

把 sprite 移动到新的位置。

原型

```
void GUI_SPRITE_SetPosition(GUI_HSPRITE hSprite, int x, int y);
```

参数	描述
hSprite	sprite 的句柄。
x	在屏幕坐标中的新的 X 位置。
y	在屏幕坐标中的新的 Y 位置。

其他信息

该函数把给定的 sprite 移动到新的位置。

GUI_SPRITE_Show()

描述

显示给定的 sprite。

原型

```
void GUI_SPRITE_Show(GUI_HSPRITE hSprite);
```

参数	描述
hSprite	sprite 的句柄。

其他信息

该函数把给定的 sprite 添加到可见 sprite 列表中。

第 25 章

游标

窗口管理器支持包括一个全系统游标，可以更改为其他预定义的风格。尽管该游标始终存在，但默认是隐藏的，与窗口一样。在通过调用命令显示它之前，是不可见的，并且可以随时再次隐藏。

25.1 可用游标

目前提供以下游标。如果应用 GUI_CURSOR_Show() 调用，并且未在 GUI_CURSOR_Select() 中指定风格，则默认游标将是一个大小适中的箭头。

箭头游标		十字游标	
GUI_CursorArrowS 小箭头		GUI_CursorCrossS 小十字	
GUI_CursorArrowM 中箭头（默认游标）		GUI_CursorCrossM 中十字	
GUI_CursorArrowL 大箭头		GUI_CursorCrossL 大十字	
反箭头游标		**反十字游标**	
GUI_CursorArrowSI 小反箭头		GUI_CursorCrossSI 小反十字	
GUI_CursorArrowMI 中反箭头		GUI_CursorCrossMI 中反十字	
GUI_CursorArrowLI 大反箭头		GUI_CursorCrossLI 大反十字	

25.2 游标 API

下表按字母顺序列出了可用的游标相关程序。详细描述如下：

程序	描述
GUI_CURSOR_GetState()	返回游标是否可见。
GUI_CURSOR_Hide()	隐藏游标。
GUI_CURSOR_Select()	设置指定的游标。
GUI_CURSOR_SetPosition()	设置游标位置。
GUI_CURSOR_Show()	显示游标。

GUI_CURSOR_GetState()

描述

返回游标当前是否可见。

原型

```
int GUI_CURSOR_GetState(void);
```

返回值

如果游标可见，返回值为 1，否则返回值为 0。

GUI_CURSOR_Hide()

描述

隐藏游标。

原型

```
void GUI_CURSOR_Hide(void);
```

其他信息

这是默认的游标设置。如果希望游标可见，则必须调用 GUI_CURSOR_Show()。

GUI_CURSOR_Select()

描述

设置指定的游标风格。

原型

```
void GUI_CURSOR_Select(const GUI_CURSOR * pCursor);
```

参数	描述
pCursor	指向待选定的游标的指针。

参数 pCursor 的允许值	
GUI_CursorArrowS	小箭头。
GUI_CursorArrowM	中箭头。
GUI_CursorArrowL	大箭头。
GUI_CursorArrowSI	小反箭头。
GUI_CursorArrowMI	中反箭头。
GUI_CursorArrowLI	大反箭头。
GUI_CursorCrossS	小十字。
GUI_CursorCrossM	中十字。
GUI_CursorCrossL	大十字。
GUI_CursorCrossSI	小反十字。
GUI_CursorCrossMI	中反十字。
GUI_CursorCrossLI	大反十字。

其他信息

如果未调用该函数，则默认游标为中箭头。

GUI_CURSOR_SetPosition()

描述

设置游标位置。

原型

```
void GUI_CURSOR_SetPosition(int x, int y);
```

参数	描述
x	游标的 X 位置。
y	游标的 Y 位置。

其他信息

一般情况下，该函数由窗口管理器内部调用，无需从应用中调用。

GUI_CURSOR_Show()

描述

显示游标。

原型

```
void GUI_CURSOR_Show(void);
```

其他信息

游标的默认设置为隐藏；因此，如果希望游标可见，则必须调用该函数。

第 26 章

抗锯齿

线条由一系列必须位于显示坐标处的像素近似构成。因此，可能看起来呈锯齿状，尤其是近似水平线或近似垂直线。这种现象称为锯齿 (aliasing)。
抗锯齿是对线条和曲线进行平滑处理。可以减少不完全水平或垂直的线条的楼梯状锯齿。 emWin 支持不同的抗锯齿质量、无锯齿字体和高分辨率坐标。
抗锯齿功能由独立的软件提供，emWin 基本版中不包括。无锯齿处理软件位于 `GUI\AntiAlias` 子目录中。

26.1 简介

抗锯齿通过使背景色与前景色相“混合”的方法来平滑曲线和对角线。背景色与前景色之间使用的阴影越多，抗锯齿效果越好 （计算时间越长）。

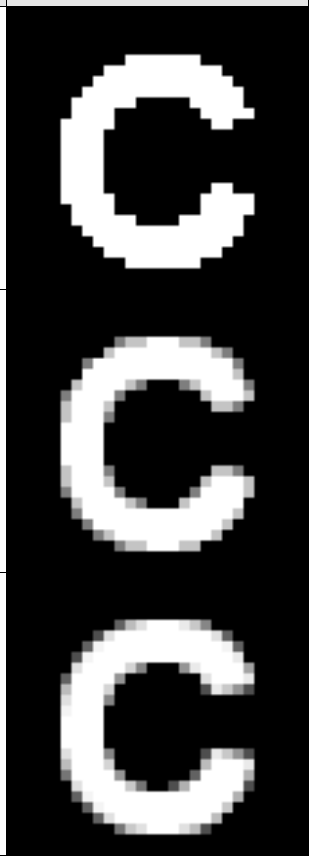
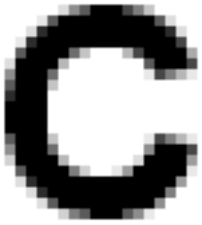
26.1.1 抗锯齿质量

抗锯齿处理的质量由 GUI_AA_SetFactor() 程序设定，详见本章后面部分。若要了解抗锯齿因子与对应结果之间的关系，请参看所给图示。

第一条线未经抗锯齿处理（因子 1）。第二条线以因子 2 进行了抗锯齿处理。也就是说，从前景到背景的阴影数为 2 x 2 = 4。下一条线以抗锯齿因子 3 绘制，因而有 3 x 3 = 9 个阴影，如此等等，不一而足。因子 4 足以处理多数应用。进一步增加抗锯齿因素不会大幅改善结果，但会增加计算时间。

26.1.2 无锯齿字体

支持低质量 (2bpp) 和高质量 (4bpp) 两种无锯齿字体。显示这些字体所需要的程序会在必要时自动链接。下表展示了在不进行抗锯齿处理以及采用两种无锯齿字体时绘制字母 C 的效果：

字体类型	白底黑字	白字黑底
标准 （无抗锯齿） 1 bpp 2 个阴影		
低质量 （经抗锯齿处理） 2 bpp 4 个阴影		
高质量 （经抗锯齿处理） 4 bpp 16 个阴影		

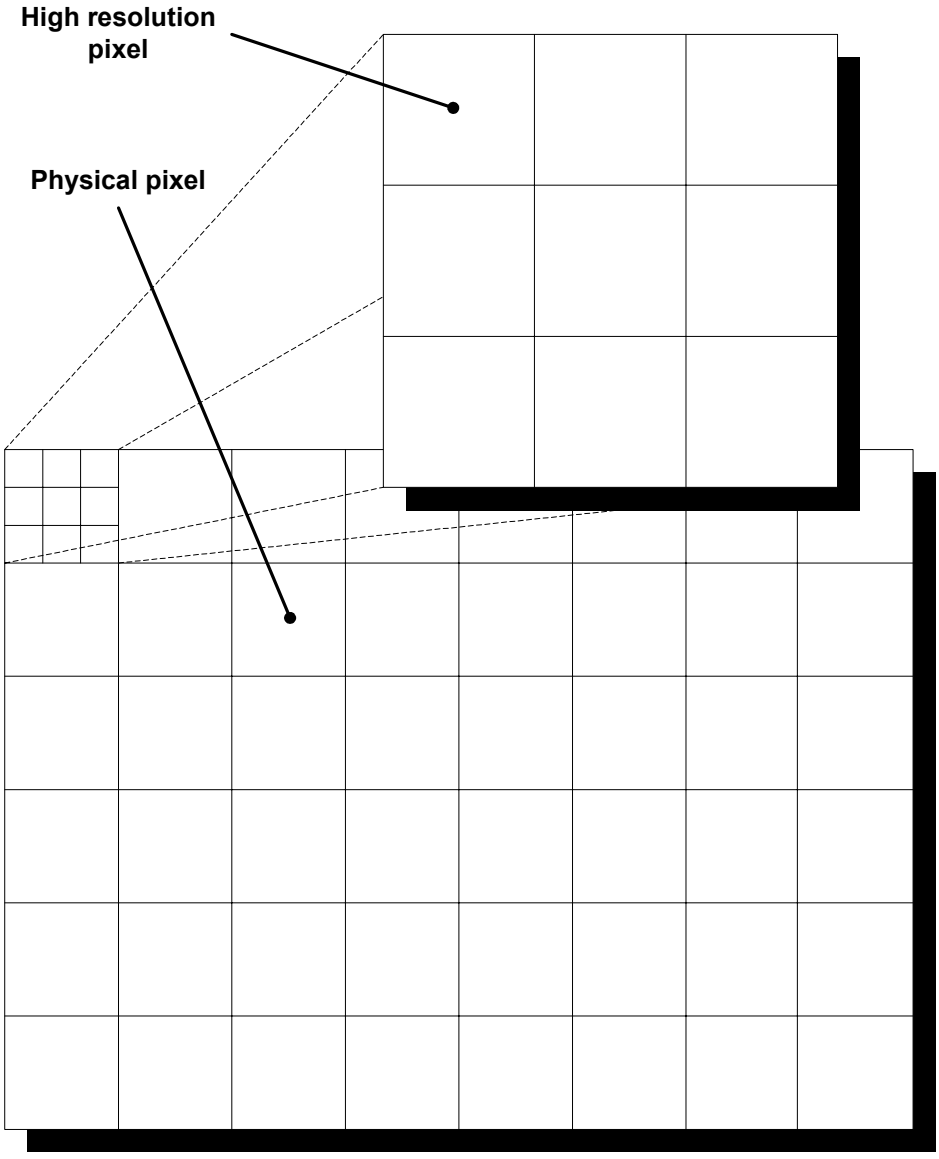
无锯齿字体可以通过 emWin 字体转换器创建。使用无锯齿字体的一般目的是改进文字的外观。虽然高质量抗锯齿比低质量搞锯齿处理看起来更好美观，但计算时间和存储器占用量也会相应增加。低质量 (2bpp) 字体需要两倍于无抗锯齿处理 (1bpp) 字体的存储器容量；高质量 (4bpp) 字体则需要四倍的存储器容量。

26.1.3 高分辨率坐标

在使用抗锯齿法绘制项目时，使用的是与常规（无抗锯齿处理）绘图程序相同的坐标。这是默认模式。在函数参数中无需考虑抗锯齿因子。例如，要从 (50, 100) 到 (100, 50) 绘制一条抗锯齿线条，则需写入以下代码：

```
GUI_AA_DrawLine(50, 100, 100, 50);
```

借助 emWin 的高分辨率功能，您可以使用取决于抗锯齿因子和显示尺寸的虚拟空间。使用高分辨率坐标的优势在于，项目不但可以置于显示器的物理位置上，而且可以置于物理位置之间。
下图展示的是一个高分辨率像素的虚拟空间，其抗锯齿因素为 3：



以抗锯齿因子3，从像素(50, 100)到像素(100, 50)绘制一条高分辨率线条，则需写入以下代码：

```
GUI_AA_DrawLine(150, 300, 300, 150);
```

高分辨率坐标必须通过 GUI_AA_EnableHiRes() 程序启用，并用 GUI_AA_DisableHiRes() 禁用。这两个函数将在本章后面部分介绍。
有关使用高分辨率功能的示程序序，请参见本单末尾的示例。

26.2 抗锯齿 API

下表以字母顺序列出了抗锯齿包中的可用程序 （按所属类别分）。有关程序的详细描述可参见后续各节。

程序	描述
控制函数	
GUI_AA_DisableHiRes()	禁用高分辨率坐标。
GUI_AA_EnableHiRes()	启用高分辨率坐标。
GUI_AA_GetFactor()	返回当前抗锯齿因子。
GUI_AA_SetFactor()	设置当前抗锯齿因子。
绘图函数	
GUI_AA_DrawArc()	绘制无锯齿弧形。
GUI_AA_DrawLine()	绘制无锯齿线条。
GUI_AA_DrawPolyOutline()	绘制最多 10 个点的无锯齿多边形轮廓。
GUI_AA_DrawPolyOutlineEx()	绘制无锯齿多边形轮廓。
GUI_AA_FillCircle()	绘制无锯齿圆形。
GUI_AA_FillPolygon()	绘制实心无锯齿多边形。

26.3 控制函数

GUI_AA_DisableHiRes()

描述

禁用高分辨率坐标。

原型

```
void GUI_AA_DisableHiRes(void);
```

其他信息

高分辨率坐标默认情况下被禁用。

GUI_AA_EnableHiRes()

描述

启用高分辨率坐标。

原型

```
void GUI_AA_EnableHiRes(void);
```


GUI_AA_GetFactor()

描述

返回当前抗锯齿质量因子。

原型

```
int  GUI_AA_GetFactor(void);
```

返回值

当前抗锯齿因子。

GUI_AA_SetFactor()

描述

设置抗锯齿质量因子。

原型

```
void GUI_AA_SetFactor(int Factor);
```

参数	描述
Factor	新的抗锯齿因子。 最小：1 （无抗锯齿处理）；最大：6。

其他信息

虽然允许把参数 Factor 设为 1，但这实际上会禁用抗锯齿处理功能，结果产生标准字体。
建议将抗锯齿质量因子设为 2-4。默认因子为 3。

26.4 绘图函数

GUI_AA_DrawArc()

描述

用当前的画笔尺寸和画笔形状，在当前窗口指定位置显示一个无锯齿弧形。

原型

```
void GUI_AA_DrawArc(int x0, int y0, int rx, int ry, int a0, int a1);
```

参数	描述
x0	中心水平位置。
y0	中心垂直位置。
rx	水平半径。
ry	垂直半径。
a0	起始角度 （度）。
a1	终止角度 （度）。

限制

目前不支持 ry 参数，而使用 rx 参数代替。

其他信息

如果采用高分辨率模式，位置和半径必须使用高分辨率坐标。否则，必须指定为像素值。

GUI_AA_DrawLine()

描述

用当前的画笔尺寸和画笔形状，在当前窗口指定位置显示一条无锯齿线。

原型

```
void GUI_AA_DrawLine(int x0, int y0, int x1, int y1);
```

参数	描述
x0	X 起始位置。
y0	Y 起始位置。
x1	X 终止位置。
y1	Y 终止位置。

其他信息

如果采用高分辨率模式，则必须使用高分辨率坐标。否则，必须指定为像素值。

GUI_AA_DrawPolyOutline()

描述

在当前窗口指定位置，以指定的厚度显示一个由一组点定义的无锯齿多边形的轮廓。定义点不超过 10 个。

原型

```
void GUI_AA_DrawPolyOutline(const GUI_POINT * pPoint,
                            int               NumPoints,
                            int               Thickness,
                            int               x,
                            int               y)
```

参数	描述
pPoint	指向待显示的多边形的指针。
指定点数量	定义点列表中指定点的数量。
厚度	轮廓的厚度。
x	X 原点位置。
y	Y 原点位置。

其他信息

通过连接终点和起点，可使绘制的多线条自动闭合。起点不得再次指定为终点。
如果采用高分辨率模式，则必须使用高分辨率坐标。否则，必须指定为像素值。
默认情况下，该函数处理的定义点不超过 10 个。如果多边形由 10 个以的点构成，则必须使用 GUI_AA_DrawPolyOutlineEx() 函数。

示例

```
#define countof(Array) (sizeof(Array) / sizeof(Array[0]))

static GUI_POINT aPoints[] = {
  {  0,  0 },
  { 15, 30 },
  {  0, 20 },
  {-15, 30 }
};

void Sample(void) {
  GUI_AA_DrawPolyOutline(aPoints, countof(aPoints), 3, 150, 40);
}
```

上述示例的屏幕截图

GUI_AA_DrawPolyOutlineEx()

描述

在当前窗口指定位置，以指定的厚度显示一个由一组点定义的无锯齿多边形的轮廓。

原型

```
void GUI_AA_DrawPolyOutlineEx(const GUI_POINT * pPoint,
                              int               NumPoints,
                              int               Thickness,
                              int               x,
                              int               y,
                              GUI_POINT       * pBuffer);
```

参数	描述
pPoint	指向待显示的多边形的指针。
NumPoints	定义点列表中指定点的数量。
厚度	轮廓的厚度。
x	X 原点位置。
y	Y 原点位置。
pBuffer	指向 GUI_POINT 元素缓冲器的指针。

其他信息

该函数不限制多边形定义点的数量。该函数内部需要一个 GUI_POINT 元素缓冲器，以便用于计算目的。缓冲器的点数应大于或等于多边形定义点数量。
更多详细信息，请参阅 “GUI_AA_DrawPolyOutline()”（第 792 页）。

GUI_AA_FillCircle()

描述

在当前窗口指定位置显示一个实心无锯齿圆形。

原型

```
void GUI_AA_FillCircle(int x0, int y0, int r);
```

参数	描述
x0	客户端窗口的圆心 X 位置 （单位：像素）。
y0	客户端窗口的圆心 Y 位置 （单位：像素）。
r	圆形半径 （直径的一半）。 最小值：0 （结果产生一个点）；最大：180。

其他信息

如果采用高分辨率模式，则必须使用高分辨率坐标。否则，必须指定为像素值。

GUI_AA_FillPolygon()

描述

在当前窗口指定位置填充一个由一组点定义的无锯齿圆形。

原型

```
void GUI_AA_FillPolygon(const GUI_POINT * pPoint,
                        int               NumPoints,
                        int               x,
                        int               y)
```

参数	描述
pPoint	指向待显示的多边形的指针。
NumPoints	定义点列表中指定点的数量。
x	X 原点位置。
y	Y 原点位置。

其他信息

通过连接终点和起点，可使绘制的多线条自动闭合。起点不得再次指定为终点。
如果采用高分辨率模式，则必须使用高分辨率坐标。否则，必须指定为像素值。

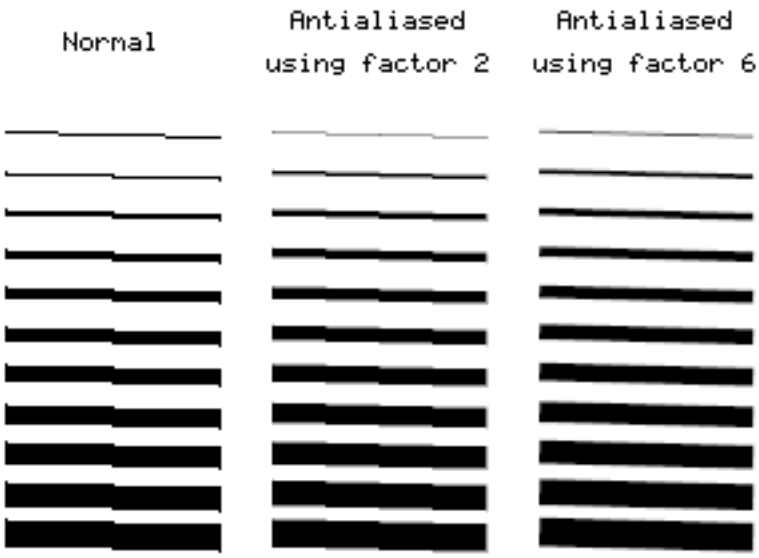
26.5 示例

不同的抗锯齿因子

以下示例将在无抗锯齿处理和有抗锯齿处理两种情况下创建对角形。源代码随 emWin 一起提供，文件名为 AA_Lines.c。

```
/*********************************************************************
*                 SEGGER Microcontroller GmbH & Co. KG               *
*        Solutions for real time microcontroller applications        *
*                                                                    *
*                     emWin example code                             *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File        :AA_Lines.c
Purpose     :Shows lines with different antialiasing qualities
----------------------------------------------------------------------
*/

#include "GUI.H"

/*********************************************************************
*
*         Show lines with different antialiasing qualities
*
**********************************************************************
*/

static void DemoAntialiasing(void) {
  int i, x1, x2;
  int y = 2;
  /* Set drawing attributes */
  GUI_SetColor(GUI_BLACK);
  GUI_SetBkColor(GUI_WHITE);
  GUI_SetPenShape(GUI_PS_FLAT);
  GUI_Clear();
  x1 = 10; x2 = 90;
  /* Draw lines without antialiasing */
  GUI_DispStringHCenterAt("\nNormal", (x1 + x2) / 2, 10);
  for (i = 1; i < 12; i++) {
    GUI_SetPenSize(i);
    GUI_DrawLine(x1, 40 + i * 15, x2, 40 + i * 15 +  y);
  }
  x1 = 110; x2 = 190;
  /* Draw lines with antialiasing quality faktor 2 */
  GUI_AA_SetFactor(2);
  GUI_DispStringHCenterAt("Antialiased\n\nusing factor 2", (x1 + x2) / 2, 10);
  for (i = 1; i < 12; i++) {
    GUI_SetPenSize(i);
    GUI_AA_DrawLine(x1, 40 + i * 15, x2, 40 + i * 15 +  y);
  }
  x1 = 210; x2 = 290;
  /* Draw lines with antialiasing quality faktor 6 */
  GUI_AA_SetFactor(6);
  GUI_DispStringHCenterAt("Antialiased\n\nusing factor 6", (x1 + x2) / 2, 10);
  for (i = 1; i < 12; i++) {
    GUI_SetPenSize(i);
    GUI_AA_DrawLine(x1, 40 + i * 15, x2, 40 + i * 15 +  y);
  }
}

/*********************************************************************
*
*                 main
*
**********************************************************************
*/

void main(void) {
  GUI_Init();
  DemoAntialiasing();
  while(1)
    GUI_Delay(100);
}
```


上述示例的屏幕截图



以高分辨率坐标绘制的线条

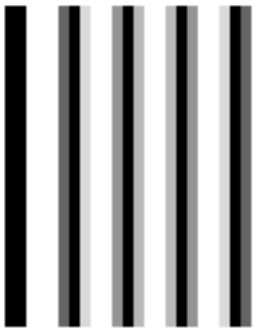
本例展示以高分辨率坐标绘制无锯齿线条。源代码见文件 AA_HiResPixels.c。

```
/*********************************************************************
*                 SEGGER Microcontroller GmbH & Co. KG               *
*         Solutions for real time microcontroller applications       *
*                                                                    *
*                     emWin example code                             *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File        :AA_HiResPixels.c
Purpose     :Demonstrates high resolution pixels
----------------------------------------------------------------------
*/

#include "GUI.H"

/*********************************************************************
*
*          Show lines placed on high resolution pixels
*
**********************************************************************
*/

static void ShowHiResPixels(void) {
  int i, Factor = 5;
  GUI_SetBkColor(GUI_WHITE);
  GUI_SetColor(GUI_BLACK);
  GUI_Clear();
  GUI_SetLBorder(50);
  GUI_DispStringAt("This example uses high resolution pixels.\n", 50, 10);
  GUI_DispString  ("Not only the physical pixels are used.\n");
  GUI_DispString  ("Enabling high resolution simulates more\n");
  GUI_DispString  ("pixels by using antialiasing.\n");
  GUI_DispString  ("Please take a look at the magnified output\n");
  GUI_DispString  ("to view the result.\n");
  GUI_SetPenSize(2);
  GUI_SetPenShape(GUI_PS_FLAT);
  GUI_AA_EnableHiRes();       /* Enable high resolution */
  GUI_AA_SetFactor(Factor);   /* Set quality factor */
  /* Drawing lines using high resolution pixels */
  for (i = 0; i < Factor; i++) {
    int x = (i + 1) * 5 * Factor + i - 1;
    GUI_AA_DrawLine(x, 50, x, 199);
  }
}
```



```
/*********************************************************************
*
*                 main
*
*********************************************************************
*/

void main(void) {
  GUI_Init();
  ShowHiResPixels();
  while(1) {
    GUI_Delay(100);
  }
```

上例的放大屏幕截图



利用高分辨率抗锯齿移动指针

在本例中，将通过绘制一个每一步转动 0.1 度的旋转点，来实现高分辨率抗锯齿。由于高分辨率抗锯齿的效果只能在指针移动过程中看见，所有本例没有屏幕截图。在标准分辨率下，指针看起来像是短程 “跳动”，而在高分辨率模式下，不存在明显的跳动。
本例由文件 AA_HiResAntialiasing.c 提供。

```
/*********************************************************************
*                 SEGGER Microcontroller GmbH & Co. KG                *
*        Solutions for real time microcontroller applications         *
*                                                                     *
*                     emWin example code                              *
*                                                                     *
**********************************************************************

----------------------------------------------------------------------
File        :AA_HiResAntialiasing.c
Purpose     :Demonstrates high resolution antialiasing
----------------------------------------------------------------------
*/

#include "GUI.H"

/*********************************************************************
*
*             Data
*
**********************************************************************
*/

#define countof(Obj) (sizeof(Obj)/sizeof(Obj[0]))

static const GUI_POINT aPointer[] = {
  {  0,  3},
  { 85,  1},
  { 90,  0},
  { 85, -1},
  {  0, -3}
};

static GUI_POINT aPointerHiRes[countof(aPointer)];
```



```
typedef struct {
  GUI_AUTODEV_INFO AutoInfo;
  GUI_POINT aPoints[countof(aPointer)];
  int Factor;
} PARAM;

/*********************************************************************
*
*                 Drawing routines
*
**********************************************************************
*/

static void DrawHiRes(void * p) {
  PARAM * pParam = (PARAM *)p;
  if (pParam->AutoInfo.DrawFixed) {
    GUI_ClearRect(0, 0, 99, 99);
  }
  GUI_AA_FillPolygon(pParam->aPoints,
                     countof(aPointer),
                     5 * pParam->Factor,
                     95 * pParam->Factor);
}

static void Draw(void * p) {
  PARAM * pParam = (PARAM *)p;
  if (pParam->AutoInfo.DrawFixed) {
    GUI_ClearRect(100, 0, 199, 99);
  }
  GUI_AA_FillPolygon(pParam->aPoints, countof(aPointer), 105, 95);
}

/*********************************************************************
*
*    Demonstrate high resolution by drawing rotating pointers
*
**********************************************************************
*/

static void ShowHiresAntialiasing(void) {
  int i;
  GUI_AUTODEV aAuto[2];
  PARAM Param;
  Param.Factor = 3;
  GUI_DispStringHCenterAt("Using\nhigh\nresolution\nmode", 50, 120);
  GUI_DispStringHCenterAt("Not using\nhigh\nresolution\nmode", 150, 120);
  /* Create GUI_AUTODEV objects */
  for (i = 0; i < countof(aAuto); i++) {
    GUI_MEMDEV_CreateAuto(&aAuto[i]);
  }
  /* Calculate pointer for high resolution */
  for (i = 0; i < countof(aPointer); i++) {
    aPointerHiRes[i].x = aPointer[i].x * Param.Factor;
    aPointerHiRes[i].y = aPointer[i].y * Param.Factor;
  }
  GUI_AA_SetFactor(Param.Factor); /* Set antialiasing factor */
  while(1) {
    for (i = 0; i < 1800; i++) {
      float Angle = (i >= 900) ?1800 - i :i;
      Angle *= 3.1415926f / 1800;
      /* Draw pointer with high resolution */
      GUI_AA_EnableHiRes();
      GUI_RotatePolygon(Param.aPoints, aPointerHiRes, countof(aPointer), Angle);
      GUI_MEMDEV_DrawAuto(&aAuto[0], &Param.AutoInfo, DrawHiRes, &Param);
      /* Draw pointer without high resolution */
      GUI_AA_DisableHiRes();
      GUI_RotatePolygon(Param.aPoints, aPointer, countof(aPointer), Angle);
      GUI_MEMDEV_DrawAuto(&aAuto[1], &Param.AutoInfo, Draw, &Param);
      GUI_Delay(2);
    }
  }
}

/*********************************************************************
*
*                 main
*
**********************************************************************
*/
```



```
void main(void) {
  GUI_Init();
  ShowHiresAntialiasing();
}
```


第 27 章

外语支持

阿拉伯语、泰语或汉语等外语文本一般含有 emWin 标准字体以外的字符。
本章将介绍一些基本知识，比如定义全球所有可用字符的 Unicode 标准，以及 UTF-8 编码方案，emWin 使用该方案来解码以 Unicode 字符编写的文本。
同时还将说明如何启用阿拉伯语支持，以及如何采用 Shift-JIS（日本工业标准）编码方案来显示文本。

27.1 Unicode

Unicode 标准是一种 16 位字符编码方案。全世界所有可用字符都包含在一个 16 位的字符集中（全球统一）。 Unicode 标准由统一码联盟 (Unicode Consortium) 定义。
emWin 能显示采用 Unicode 编码的单个字符或字符串，不过，最常见的情况是使用混合字符串，即一个 ASCII 字符串中有任意个 Unicode 序列。

27.1.1 UTF-8 编码方案

ISO/IEC 10646-1 定义了一种称为通用字符集 (UCS) 的多八位字符集，收入了全球大多数文字系统。然而，多八位字符并不兼容许多现有的应用程序和协议，因此就扩展出了一些各有特色的 UCS 转换格式 (UTF)。
UTF-8 具有多种特点，保留了全部 ASCII 字符，并且兼容依赖于 ASCII 值但对其他值透明的文件系统、解析器和其他软件。
在 emWin 中， UTF-8 字符以 1 至 3 个八位字节序列编码。如果将高序位设为 0，则剩下的 7 位将用来编码字符值。在一个由 n 个八位字节构成的序列中 (n>1)，起始八位字节第 n 个高序位设为 1，其后一位设为 0。该八位字节的剩余位用于存放待编码字符的值位。后续八位字节都将高序位设为 1，其后一位设为 0，各剩 6 位，用于保存待编码字符的值位。
下表显示了编码范围：

字符范围	UTF-8 八位字节序列
0000 - 007F	0xxxxxxx
0080 - 07FF	110xxxxx 10xxxxxx
0800 - FFFF	1110xxxx 10xxxxxx 10xxxxxx

编码示例

文本"Halöle"含有ASCII字符和欧洲扩展字符。以下十六进制转储将该文本显示为UTF-8编码文本：

```
48 61 6C C3 B6 6C 65
```

编程示例

如果需要显示含有非 ASCII 字符的文本，则可以通过手动计算字符串中的非 ASCII 字符的 UTF-8 编码来实现。
然而，如果编译器支持 UTF-8 编码 （有时称为多字节编码），则即使是非 ASCII 字符也可以在字符串中直接进行使用。

```
//
// Example using ASCII encoding:
//
GUI_UC_SetEncodeUTF8();       /* required only once to activate UTF-8*/
GUI_DispString("Hal\xc3\xb6le");

//
// Example using UTF-8 encoding:
//
GUI_UC_SetEncodeUTF8();       /* required only once to activate UTF-8*/
GUI_DispString("Halle");
```

27.1.2 Unicode 字符

emWin 使用的字符输出程序 (GUI_DispChar()) 将始终接受无符号的 16 位值 (U16)，并具有显示 Unicode 定义字符的基本能力。只需要含有待显示字符的字体就可以了。

27.1.3 UTF-8 字符串

这是显示 Unicode 的首选推荐方式。无需使用特殊函数即可实现。如果启用 UTF-8 编码，则 emWin 中处理字符串的每个函数都会以 UTF-8 文本来解码给定的文本。

27.1.3.1 使用 U2C.exe 将 UTF-8 文本转换为 C 编码

emWin 的 Tool 子目录含有 U2C.exe 工具，可将 UTF-8 文本转换为 C 编码。该工具先读入一个 UTF-8 文本文件，然后以 C 字符串创建一个 C 文件。以下步骤将展示如何把文本文件转换为 C 字符串，以及如何使用 emWin 来显示这些字符串：

第 1 步：创建一个 UTF-8 文本文件

把待转换的文本保存为UTF-8格式。这可以通过Notepad.exe来实现。用Notepad.exe加载文本：

```
Japanese:
1 - エンコーディング
2 - テキスト
3 - サポート
English:
1 - encoding
2 - text
3 - support
```

选择 File/Save As...。文件对话框含有一个组合框，用于设置编码格式。选择 “UTF-8”并保存文本文件。

第 2 步：把文本文件转换为 C 编码文件

启动 U2C.exe。启动程序后，需要选择待转换的文本文件。选定文本文件后，应选择 C 文件的名称。U2C.exe 输出：

```
"Japanese:"
"1 - \xe3\x82\xa8\xe3\x83\xb3\xe3\x82\xb3\xe3\x83\xbc
    "\xe3\x83\x87\xe3\x82\xa3\xe3\x83\xb3\xe3\x82\xb0"
"2 - \xe3\x83\x86\xe3\x82\xad\xe3\x82\xb9\xe3\x83\x88"
"3 - \xe3\x82\xb5\xe3\x83\x9d\xe3\x83\xbc\xe3\x83\x88"
"English:"
"1 - encoding"
"2 - text"
"3 - support"
```

第 3 步：在应用代码中使用输出

以下示例展示了如何使用 emWin 来显示 UTF-8 文本：

```
#include "GUI.h"

static const char * _apStrings[] = {
  "Japanese:",
  "1 - \xe3\x82\xa8\xe3\x83\xb3\xe3\x82\xb3\xe3\x83\xbc"
      "\xe3\x83\x87\xe3\x82\xa3\xe3\x83\xb3\xe3\x82\xb0",
  "2 - \xe3\x83\x86\xe3\x82\xad\xe3\x82\xb9\xe3\x83\x88",
  "3 - \xe3\x82\xb5\xe3\x83\x9d\xe3\x83\xbc\xe3\x83\x88",
  "English:",
  "1 - encoding",
  "2 - text",
  "3 - support"
};

void MainTask(void) {
  int i;
  GUI_Init();
  GUI_SetFont(&GUI_Font16_1HK);
  GUI_UC_SetEncodeUTF8();
  for (i = 0; i < GUI_COUNTOF(_apStrings); i++) {
    GUI_DispString(_apStrings[i]);
    GUI_DispNextLine();
  }
  while(1) {
    GUI_Delay(500);
  }
}
```


27.1.4 Unicode API

下表按字母顺序列出了可用的程序 （按所属类别分）。有关程序的详细描述可参见后续各节。

程序	描述
UTF-8 函数	
GUI_UC_ConvertUC2UTF8()	将 Unicode 字符串转换为 UTF-8 格式。
GUI_UC_ConvertUTF82UC()	将 UTF-8 字符串转换为 Unicode 格式。
GUI_UC_EnableBIDI()	启用 / 禁用双向字体支持。
GUI_UC_Encode()	以当前编码方案对给定的字符进行编码。
GUI_UC_GetCharCode()	返回解码的字符。
GUI_UC_GetCharSize()	返回编码给定字符所使用的字节数。
GUI_UC_SetEncodeNone()	禁用编码。
GUI_UC_SetEncodeUTF8()	启用 UTF-8 编码。
双字节函数	
GUI_UC_DispString()	显示双字节字符串。

27.1.4.1 UTF-8 函数

GUI_UC_ConvertUC2UTF8()

描述

将给定的双字节 Unicode 字符串转换为 UTF-8 格式。

原型

```
int GUI_UC_ConvertUC2UTF8(const U16 GUI_UNI_PTR * s, int Len,
                          char * pBuffer, int BufferSize);
```

参数	描述
s	指向待转换的 Unicode 字符串的指针。
Len	待转换的 Unicode 字符的数量。
pBuffer	指向用于写入结果的缓冲器的指针。
BufferSize	缓冲区大小 （以字节为单位）。

返回值

该函数返回写入缓冲器的字节数。

其他信息

采用UTF-8编码的字符最多可以使用3个字节。保险起见，建议缓冲器尺寸为：Unicode字符数*3。如果缓冲器无法存储全部结果，则函数会在缓冲器满载时返回。

GUI_UC_ConvertUTF82UC()

描述

将给定的 UTF-8 字符串转换为 Unicode 格式。

原型

```
int GUI_UC_ConvertUTF82UC(const char GUI_UNI_PTR * s, int Len,
                          U16 * pBuffer, int BufferSize);
```

参数	描述
s	指向待转换的 UFT-8 字符串的指针。
Len	待转换字符串的长度 （以字节表示）。
pBuffer	指向用于写入结果的缓冲器的指针。
BufferSize	缓冲器尺寸 （以字为单位）。

返回值

该函数返回写入缓冲器的 Unicode 字符数。

其他信息

如果缓冲器无法存储全部结果，则函数会在缓冲器满载时返回。

GUI_UC_EnableBIDI()

描述

该函数启用双向字体支持。

原型

```
int GUI_UC_EnableBIDI(int OnOff);
```

参数	描述
OnOff	1 启用 BIDI 支持， 2 禁用。

返回值

BIDI 支持的上一个状态。

其他信息

一旦链接该函数，则会使用大约 60 Kb 的额外 ROM 空间。

GUI_UC_Encode()

描述

该函数以当前编码设置对给定的字符进行编码。

原型

```
int GUI_UC_Encode(char* s, U16 Char);
```

参数	描述
s	指向用于存储编码字符的缓冲器的指针。
Char	待编码的字符。

返回值

存入缓冲器的字节数。

其他信息

该函数假定，缓冲器至少为结果准备了 3 个字节的空间。

GUI_UC_GetCharCode()

描述

该函数对给定文本的字符进行解码。

原型

```
U16 GUI_UC_GetCharCode(const char* s);
```

参数	描述
s	指向待编码的文本的指针。

返回值

编码的字符。

相关主题

GUI_UC_GetCharSize()

GUI_UC_GetCharSize()

描述

该函数返回编码给定字符所使用的字节数。

原型

```
int GUI_UC_GetCharSize(const char* s);
```

参数	描述
s	指向待编码的文本的指针。

返回值

编码给定字符所使用的字节数。

其他信息

该函数的作用是确定某个指针指向下一个字符需要增加多少字节。以下示例展示了本函数的使用方法：

```
static void _Display2Characters(const char * pText) {
  int Size;
  U16 Character;
  Size = GUI_UC_GetCharSize(pText);       /* Size to increment pointer */
  Character = GUI_UC_GetCharCode(pText); /* Get first character code */
  GUI_DispChar(Character);                /* Display first character */
  pText += Size;                          /* Increment pointer */
  Character = GUI_UC_GetCharCode(pText); /* Get next character code */
  GUI_DispChar(Character);                /* Display second character */
}
```

GUI_UC_SetEncodeNone()

描述

禁用字符编码。

原型

```
void GUI_UC_SetEncodeNone(void);
```

其他信息

调用该函数后，文本中的每个字节都将被当作一个字符处理。这是 emWin 的默认行为。

GUI_UC_SetEncodeUTF8()

描述

启用 UTF-8 编码。

原型

```
void GUI_UC_SetEncodeUTF8(void);
```


其他信息

调用 GUI_UC_SetEncodeUTF8 后，emWin 中每个字符串相关程序都会根据 UTF-8 转换规则对给定的字符串进行编码。

27.1.4.2 双字节函数

GUI_UC_DispString()

描述

该函数显示给定的双字节字符串。

原型

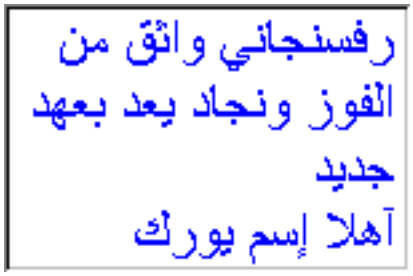
```
void GUI_UC_DispString(const U16 GUI_FAR *s);
```

参数	描述
s	指向双字节字符串的指针。

其他信息

如果需要显示双字节字符串，则需使用该函数。每个字符均需由一个 16 位值来进行定义。

27.2 阿拉伯语支持



西方语言与阿拉伯语之间的根本差异在于，阿拉伯语是从右往左写的，而且不区分大小写。另外，文本中的字符编码并不等同于字体中用来转换该字符的字符索引，因为字符的记号形式取决于其在文本中的位置。

27.2.1 记号形式

阿拉伯基本字符集由 Unicode 标准定义，范围为 0x0600 至 0x06FF。遗憾的是，这些字符编码将不能直接用于获取绘制这些字符的字体字符，因为记号形式取决于字符在文本中的位置。一个字符最多可以有 4 个不同的记号形式：

- 一，位于字的开头处 （字首）
- 二，位于字的末尾处 （字尾）
- 三，位于字的中间 （字中）
- 四，为独立字符 （独立）

但是，并非每个字符都可以附加到左边和右边 （双向附加）。例如，字符 “Hamza” 始终都是独立的，“Alef” 只能出现在末尾或以独立形式出现。字母 “Lam” 和 “Alef” 构成的字符组合需要转换为一个合体字符 (Ligature)。这就是说用一个字符替换 “Lam” 和 “Alef” 的组合。
从以上说明可以看出，记号形式一般并不等同于文本的字符编码。下表展示了 emWin 是如何根据文本位置将字符转换为记号形式的：

基本编码	独立	字尾	字首	字中	字符
0x0621	0xFE80	-	-	-	Hamza
0x0622	0xFE81	0xFE82	-	-	Alef 且 Madda 位于上方
0x0623	0xFE83	0xFE84	-	-	Alef 且 Hamza 位于上方
0x0624	0xFE85	0xFE86	-	-	Waw 且 Hamza 位于上方
0x0625	0xFE87	0xFE88	-	-	Alef 且 Hamza 位于下方
0x0626	0xFE89	0xFE8A	0xFE8B	0xFE8C	Yeh 且 Hamza 位于上方
0x0627	0xFE8D	0xFE8E	-	-	Alef
0x0628	0xFE8F	0xFE90	0xFE91	0xFE92	Beh
0x0629	0xFE93	0xFE94	-	-	Teh Marbuta
0x062A	0xFE95	0xFE96	0xFE97	0xFE98	Teh
0x062B	0xFE99	0xFE9A	0xFE9B	0xFE9C	Theh
0x062C	0xFE9D	0xFE9E	0xFE9F	0xFEA0	Jeem
0x062D	0xFEA1	0xFEA2	0xFEA3	0xFEA4	Hah
0x062E	0xFEA5	0xFEA6	0xFEA7	0xFEA8	Khah
0x062F	0xFEA9	0xFEAA	-	-	Dal
0x0630	0xFEAB	0xFEAC	-	-	Thal
0x0631	0xFEAD	0xFEAE	-	-	Reh
0x0632	0xFEAF	0xFEB0	-	-	Zain
0x0633	0xFEB1	0xFEB2	0xFEB3	0xFEB4	Seen
0x0634	0xFEB5	0xFEB6	0xFEB7	0xFEB8	Sheen
0x0635	0xFEB9	0xFEBA	0xFEBB	0xFEBC	Sad
0x0636	0xFEBD	0xFEBE	0xFEBF	0xFEC0	Dad

基本编码	独立	字尾	字首	字中	字符
0x0637	0xFEC1	0xFEC2	0xFEC3	0xFEC4	Tah
0x0638	0xFEC5	0xFEC6	0xFEC7	0xFEC8	Zah
0x0639	0xFEC9	0xFECA	0xFECB	0xFECC	Ain
0x063A	0xFECD	0xFECE	0xFECF	0xFED0	Ghain
0x0641	0xFED1	0xFED2	0xFED3	0xFED4	Feh
0x0642	0xFED5	0xFED6	0xFED7	0xFED8	Qaf
0x0643	0xFED9	0xFEDA	0xFEDB	0xFEDC	Kaf
0x0644	0xFEDD	0xFEDE	0xFEDF	0xFEE0	Lam
0x0645	0xFEE1	0xFEE2	0xFEE3	0xFEE4	Meem
0x0646	0xFEE5	0xFEE6	0xFEE7	0xFEE8	Noon
0x0647	0xFEE9	0xFEEA	0xFEEB	0xFEEC	Heh
0x0648	0xFEED	0xFEEE	-	-	Waw
0x0649	0xFEEF	0xFEF0	-	-	Alef Maksura
0x064A	0xFEF1	0xFEF2	0xFEF3	0xFEF4	Yeh
0x067E	0xFB56	0xFB57	0xFB58	0xFB59	Peh
0x0686	0xFB7A	0xFB7B	0xFB7C	0xFB7D	Tcheh
0x0698	0xFB8A	0xFB8B	-	-	Jeh
0x06A9	0xFB8E	0xFB8F	0xFB90	0xFB91	Keheh
0x06AF	0xFB92	0xFB93	0xFB94	0xFB95	Gaf
0x06CC	0xFBFC	0xFBFD	0xFBFE	0xFBFF	Farsi Yeh

27.2.2 合体字符

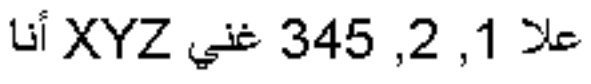
由 “Lam”和 “Alef”构成的字符组合需要转换为合体字符。下表展示了在第一个字母为 “Lam”（编码：0x0644）的情况下， emWin 将这些组合转换为合体字符的方法：

第二个字母	合体字符 （字尾）	合体字符 （其他位置）
0x622, Alef 且 Madda 位于上方	0xFEF6	0xFEF5
0x623, Alef 且 Hamza 位于上方	0xFEF8	0xFEF7
0x625, Alef 且 Hamza 位于下方	0xFEFA	0xFEF9
0x627, Alef	0xFEFC	0xFEFB

27.2.3 双向文本对齐

如上所述，阿拉伯语是从右到左书写的 (RTL)。但是，如果阿拉伯语文本所含有的数字超过一位数，则这些数字是从左向右书写的。如果阿拉伯语文本与欧洲语言文本混合，则需要遵守更多规则，以便在视觉上文本能够正确对齐。
统一码联盟在 Unicode 标准中定义了这些规则。如果启用双向文本支持，则 emWin 会遵循多数此类规则，以在绘制文本前获得正确的视觉顺序。
emWin 同时还支持对 RTL 对齐文本中的中性字符进行镜像处理。比如，在阿拉伯语文本含有括号的情况下，这点就非常重要。该处理方法是用图像符合镜像图像的镜像伴侣字符编码来取代需要镜像显示的字符编码。如需快速处理，则可使用一个包含有所有字符以及镜像伴侣字符的表。需要注意的是，不支持对其他字符的镜像处理支持。

以下示例展示了 emWin 转换双向文本的方式：

UTF-8 文本	转换
\xd8\xb9\xd9\x84\xd8\xa7 1, 2, 345 \xd8\xba\xd9\x86\xd9\x8a XYZ \xd8\xa3\xd9\x86\xd8\xa7	

27.2.4 要求

阿拉伯语支持随基本版提供。需要注意的是，emWin 的标准字体不包含支持阿拉伯字符的字体文件。如果要创建阿拉伯语字体文件，则必须使用字体转换器。

存储器

双向文本对齐和阿拉伯字符转换需要占用大约 60 KB 的 ROM 以及约 800 字节的额外堆栈。

27.2.5 如何启用阿拉伯语支持

默认情况下， emWin 始终从左到右写文本，不会进行上述阿拉伯语字符转换操作。如果要启用对双向文本和阿拉伯字符转换的支持，则必须在应用中添加以下一行代码：

```
GUI_UC_EnableBIDI(1);
```

启用后， emWin 会遵循统一码联盟规定的双向算法规则，以在绘制文本前获得正确的视觉顺序。

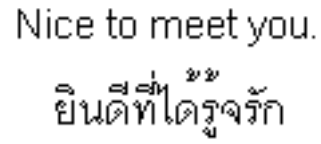
27.2.6 示例

示例文件夹中所包含的示例 FONT_Arabic 展示了该如何绘制阿拉伯语文本。在该示例中含有一种支持阿拉伯字符的 emWin 字体，同时还包括一些小的阿拉伯语示例文本。

27.2.7 配合阿拉伯语文本使用的字体文件

转换阿拉伯语所使用的字体文件至少需要包含阿拉伯语字符范围 0x600-0x6FF 中定义的所有字符，以及本章表格中所列出的记号形式和合体字符。

27.3 泰语支持



泰语字母系统使用了 44 个辅音和 15 个基本元音字符。这些字符从左到右按水平顺序书写，中间没有空格，构成音节、字和句。元音写在所修饰的辅音之上、之下、之前或之后，但在说话时，辅音始终首先发音。元音字符（以及少数辅音字符）可以按多种方式进行组合，从而形成各种各样的复合元音 （双元音和三元音）。

27.3.1 要求

如上所述，泰语使用大量复合字符。如果要在 emWin 中绘制复合字符，则需要使用一种新的字体，其中应含有所有必要的字符信息，比如图像尺寸、图像位置和游标增量值等。从 4.00 版起，emWin 开始支持含有这些信息的新型字体。这也意味着，较老的字体不能用于绘制泰语文本。
需要注意的是， emWin 的标准字体不包含支持泰字符的字体文件。如果要创建泰语字体文件，则必须使用 3.04 版或以上的字体转换器。

存储器

泰语支持不需要额外的 ROM 或 RAM。

27.3.2 如何启用泰语支持

泰语支持无需通过配置开关来启用。绘制泰语文本需要做的唯一一件事就是，用 3.04 版或以上的字体转换器创建一个 “扩展”类型的字体文件。

27.3.3 示例

示例文件夹中所包含的示例 `FONT_ThaiText.c` 展示了该如何绘制泰语文本。该示例中含有一种支持泰语字符的 emWin 字体，同时也包括一些小的泰语示例文本。

27.3.4 配合泰语文本使用的字体文件

转换泰语文本所使用的字体文件至少需要包含泰语字符范围 0xE00-0xE7F 中定义的所有字符。

27.4 Shift JIS 支持

Shift JIS（日本工业标准）是一种针对日语的字符编码方法。这是最常用的日语编码方法。 Shift JIS 编码大量使用 8 位字符，第一个字节的值用于区分单字节字符和多字节字符。
仅当需要转换采用 Shift JIS 编码文本时，才需要 emWin 支持 Shift JIS。
绘制 Shift JIS 字符串时无需调用特殊函数。关键是要有一个含有 Shift JIS 字符的字体文件。

27.4.1 创建 Shift JIS 字体

字体转换器可以从任意 Windows 字体产生 Shift JIS 字体，以供 emWin 使用。使用 Shift JIS 字体时，用于显示 Shift JIS 字符的函数会自动与字体库链接。
若要详细了解创建 Shift-JIS 字体的方法，请联系 SEGGER 微控制器公司 (info@segger.com)。单独的字体转换器文档中详细描述了如何在 emWin 工程中高效地应用 Shift JIS 字符。

第 28 章

显示驱动

显示驱动支持特定的显示控制器系列，以及连接至一个或多个该类控制器的所有显示器。只需修改驱动的配置文件即可对驱动进行配置，而无需修改驱动本身。配置文件含有程序器的所有必要信息，包括硬件访问方式，以及控制器与显示器的连接方式等。
本章将简要介绍 emWin 支持的显示驱动。对于各驱动，涵盖的内容有：

- 可以访问哪些显示控制器，以及支持的颜色深度和接口类型等。
- RAM 要求。
- 各驱动特定的函数。
- 硬件的访问方式。
- 特殊配置开关。
- 具体显示控制器的特殊要求。

28.1 现有显示驱动

驱动接口的改变始于 emWin V5。不再支持针对 emWin V4 或更早版本开发的老显示驱动。改变显示驱动接口的目的是为了能在运行过程中配置驱动。其原因在于， emWin 经常被用作一种预编译的库，在使用不同显示器时不应进行修改。

警示：如果创建一个预编译库 （该库中含有编译时可配置驱动的源文件），则会将用库进行配置的能力排除在外。

为了在短期内尽量多地支持显示控制器，我们把一些旧驱动移植到了新的接口中。需要注意的是，这些移植的显示驱动并不是完全可以在运行时间配置的。只有完全新开发的驱动支持运行时间配置。有关目前可用的驱动，请参见 28.1.2 及后续部分。

28.1.1 驱动文件命名规则

属于同一显示驱动的所有文件均应以驱动名开头。因此，称为 `<DriverName>*.*` 的所有文件即涵盖了整个驱动。

示例

下列文件描述的是 `GUIDRV_IST3088` 显示驱动：

- `GUIDRV_IST3088.c`
- `GUIDRV_IST3088.h`
- `GUIDRV_IST3088_4.c`
- `GUIDRV_IST3088_Private.h`
- `GUIDRV_IST3088_X_4.c`

28.1.2 运行时间可配置驱动

下表列出了针对 emWin 当前接口开发的现有运行时间可配置型驱动：

驱动	支持的显示控制器 / 驱动的作用	支持的位数 / 像素
GUIDRV_BitPlains	该驱动可用于不含显示控制器的解决方案。它负责管理每个颜色位的独立“位平面”。当初开发该驱动的目的是为了支持一种 R32C/111 解决方案，这种解决方案可以不用显示控制器驱动一个 TFT 显示器。可用于要求以独立平面表示颜色位的各种解决方案。	1 - 8
GUIDRV_Dist	该驱动支持带有多个控制器的显示器。	取决于实际使用的显示驱动。
GUIDRV_FlexColor	爱普生 S1D19122 奇景 HX8353、 HX8325A Ilitek ILI9320、 ILI9325、 ILI9328 LG 电子 LGDP4531、 LGDP4551 联咏 NT39122 OriseTech SPFD5408、 SPFD54124C、 SPFD5414D 瑞萨 R61505、 R61516、 R61580 矽创 ST7628、 ST7637、 ST7735 所罗门 SSD1355、 SSD1961、 SSD1963、 SSD2119 Syncoam SEPS525	16, 18
GUIDRV_IST3088	IST3088 IST3257	4
GUIDRV_Lin	该驱动向每个显示控制器提供线性可寻址视频存储器和一个直接 （全总线）接口。这就意味着，视频 RAM 可以由 CPU 的地址线直接寻址。驱动不含特殊控制器代码。因此，可以用于无显示控制器的解决方案，这种解决方案要求驱动仅负责管理视频 RAM。	1, 2, 4, 8, 16, 24, 32
GUIDRV_S1D13748	爱普生 S1D13748	16
GUIDRV_S1D15G00	爱普生 S1D15G00	12
GUIDRV_SLin	爱普生 S1D13700 （仅含间接接口！） 所罗门 SSD1848 东芝 T6963 晶宏 UC1617	1, 2
GUIDRV_SPage	晶宏 UC1611	2, 4
GUIDRV_SSD1926	所罗门 SSD1926	8

28.1.3 编译时可配置驱动

下表列出了已移植到最新版 emWin 的现有驱动：

驱动	支持的显示控制器 / 驱动的作用	支持的位数 / 像素
GUIDRV_CompactColor_16	晶采 FSA506 爱普生 S1D13742、S1D13743、S1D19122 奇景 HX8301、HX8312A、HX8325A、HX8340、HX8347、HX8352、HX8352B、HX8353 日立 HD66766、HD66772、HD66789 Ilitek ILI9161、ILI9220、ILI9221、ILI9320、ILI9325、ILI9326、ILI9328 LG 电子 LGDP4531、LGDP4551 美格纳 D54E4PA7551 联咏 NT39122、NT7573 OriseTech SPFD5408、SPFD54124C、SPFD5414D、SPFD5420A 瑞萨 R61505、R61509、R61516、R61580、R63401 三星 S6D0110A、S6D0117、S6D0129、S6D04H0 夏普 LCY-A06003、LR38825 矽创 ST7628、ST7637、ST7712、ST7715、ST7735、ST7787 所罗门 SSD1284、SSD1289、SSD1298、SSD1355、SSD1961、SSD1963、SSD2119 东芝 JBT6K71	16
GUIDRV_Fujitsu_16	Fujitsu MB87J2020 (Jasmine) Fujitsu MB87J2120 (Lavender)	1, 2, 4, 8, 16
GUIDRV_Page1bpp	爱普生 S1D10605、S1D15605、S1D15705、S1D15710、S1D15714、S1D15721、S1D15E05、S1D15E06、SED1520、SED1560、SED1565、SED1566、SED1567、SED1568、SED1569、SED1575 日立 HD61202 IST IST3020 新日本无线株式会社 NJU6676、NJU6679 联咏 NT7502、NT7534、NT7538、NT75451 飞利浦 PCF8810、PCF8811、PCF8535、PCD8544 三星 KS0108B、KS0713、KS0724、S6B0108B、S6B0713、S6B0719、S6B0724、S6B1713 中颖 SH1101A 矽创 ST7522、ST7565、ST7567 所罗门 SSD1303、SSD1805、SSD1815 意法半导体 ST7548、STE2001、STE2002 凌阳 SPLC501C 晶宏 UC1601、UC1606、UC1608、UC1701	1
GUIDRV_07X1	联咏 NT7506 三星 KS0711、KS0741、S6B0711、S6B0741 矽创 ST7541、ST7571 所罗门 SSD1854 意法半导体 STE2010 Tomato TL0350A	2
GUIDRV_1611	爱普生 S1D15719、S1D15E05、S1D15E06 晶宏 UC1611 晶宏 UC1610	2 4 2
GUIDRV_6331	三星 S6B33B0X、S6B33B1X、S6B33B2X	16
GUIDRV_7529	矽创 ST7529	1, 4, 5

28.1.4 尚未移植的现有驱动

下表列出了尚未移植到最新版 emWin 的新接口的全部现有驱动：

驱动	支持的显示控制器 / 驱动的作用	支持的位数 / 像素
GUIDRV_Mem	无控制器，写入主存储器 需要 ISR 或特殊硬件以刷新 LCD （单色显示）	1, 2
GUIDRV_MemC	无控制器，写入主存储器 需要 ISR 或特殊硬件以刷新 LCD （彩色显示）	3, 6
GUIDRV_Noritake	Noritake 显示 GU256X128C-3900	1
GUIDRV_Page4bpp	矽创 ST7528	4
GUIDRV_SLin (*1)	爱普生 SED1330、SED1335 RAIO 8822/8803、8835	1
GUIDRV_Vesa	任何 VESA 兼容硬件	8, 16
GUIDRV_Xylon	Xylon 公司基于 FPGA 的显示控制器	8, 16, 32
GUIDRV_0323	所罗门 SSD0323 OLED 控制器	4
GUIDRV_1200	Toppoly C0C0、C0E0	16
GUIDRV_13701	爱普生 S1D13701 OLED 控制器	9, 12
GUIDRV_159A	爱普生 SED159A 矽创 ST7632	8
GUIDRV_161620	NEC µPD161620	12
GUIDRV_1781	所罗门 SSD1768、SSD1781、SSD1783、SSD1797	16
GUIDRV_6642X	日立 HD66420、HD66421	2
GUIDRV_66750	日立 HD66750、HD66753	2
GUIDRV_7920	矽创 ST7920	1
GUIDRV_8822	Raio RA8822	2

*1: 目前有一个称为 “GUIDRV_Slin” 的新驱动。需要注意的是，该驱动目前不支持其未移植版支持的所有控制器。对这些控制器的支持可应要求快速添加。

28.1.5 特殊用途驱动

基本版包括 2 个不支持具体 LCD 控制器的驱动。可以用作新驱动模板或测量模板：

驱动	LCD 控制器	支持的位数 / 像素
GUIDRV_Template	驱动模板。 可以其为基础编写新的驱动。 随基本版提供	-

28.2 CPU / 显示控制器接口

不同的显示控制器可能有不同的 CPU 接口。基本而言，有两种不同的接口：

- 直接接口
- 间接接口

直接接口通过 CPU 地址总线直接存取视频存储器，间接接口则要求更加复杂的通信方式，由显示控制器来存取视频存储器。这可以通过不同类型的连接来实现：

- 并行存取
- 4 引脚 SPI 接口
- 3 引脚 SPI 接口
- I2C 总线接口

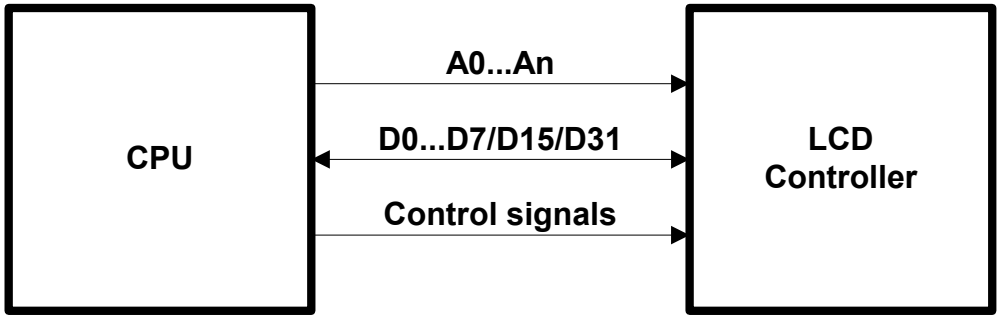
下面将说明这些接口及其配置方法。请注意，并非始终都要求使用所有的配置宏。要详细了解需要哪些宏，请参见 “显示驱动详细描述”（第 829 页）。

28.2.1 直接接口

有些显示控制器（尤其是针对高分辨率显示器的控制器）要求使用全地址总线，意即它们要至少连接至 14 个地址位。在直接接口配置中，视频存储器由 CPU 直接存取；地址总线与显示控制器相连。在配置直接接口时唯一需要了解的是地址范围信息 （将为 LCD 控制器产生一个 CHIP-SELECT 信号），以及是否需要使用 8 位、16 位或 32 位存取方式（即显示控制器的总线宽度）。换言之，您需要了解以下各项：

- 视频存储器存取的基本地址
- 寄存器存取的基本地址
- 相邻视频存储器位置之间的距离 （通常为 1/2/4 个节）
- 相邻寄存器位置之间的距离 （通常为 1/2/4 字节）
- 视频存储器的存取类型 （8/16/32 位）
- 寄存器的存取类型 （8/16/32 位）

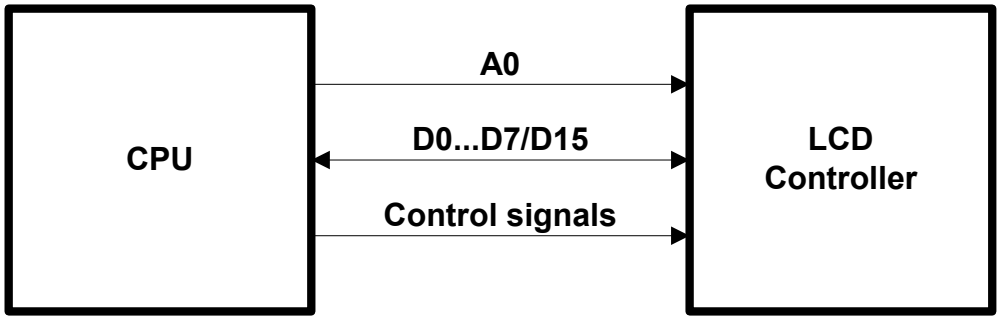
典型框图



28.2.2 间接接口 —— 并行总线

针对小型显示器的多数控制器用间接接口来连接至 CPU。使用间接接口时，只有 1 个地址位 （通常为 A0）与 LCD 控制器相连。有些这种控制器非常慢，因此，硬件设计师可以决定将其连接至输入 / 输出 (I/O) 引脚而不是地址总线。

典型框图



用 8 (16) 个数据位、1 个地址位和 2 或 3 条控制线把 CPU 与 1 个 LCD 控制器连接起来。用 4 个宏告诉 LCD 驱动如何存取所用的各个控制器。如果 LCD 控制器直接连接至 CPU 的地址总线，则配置非常简单，每个宏通常不超过 1 行。如果 LCD 控制器连接至 I/O 引脚，则必须模拟总线接口，每个宏需要大约 5-10 行程序（或者用函数调用总线接口的模拟程序）。信号 **A0** 也称为 **C/D**（命令 / 数据）、 **D/I** （数据 / 指令）或 **RS** （寄存器选择），具体取决于显示控制器。

28.2.2.1 I/O 引脚连接程序示例

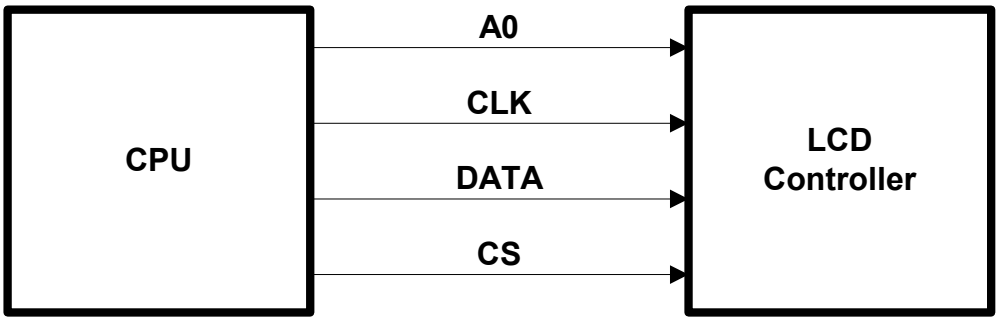
示例可在 Sample\LCD_X 文件夹下找到：

- LCD_X_6800.c，针对 6800 并行接口的端口程序。
- LCD_X_8080.c，针对 8080 并行接口的端口程序。

28.2.3 间接接口 ——4 引脚 SPI

4 引脚 SPI 接口的用法与并行接口非常相似。通过 4 引脚 SPI 接口连接 LCD 显示器，必须把 A0、CLK、 DATA 和 CS 4 条线连接至 CPU。

典型框图



28.2.3.1 I/O 引脚连接程序示例

示例可在 Sample\LCD_X 文件夹下找到：

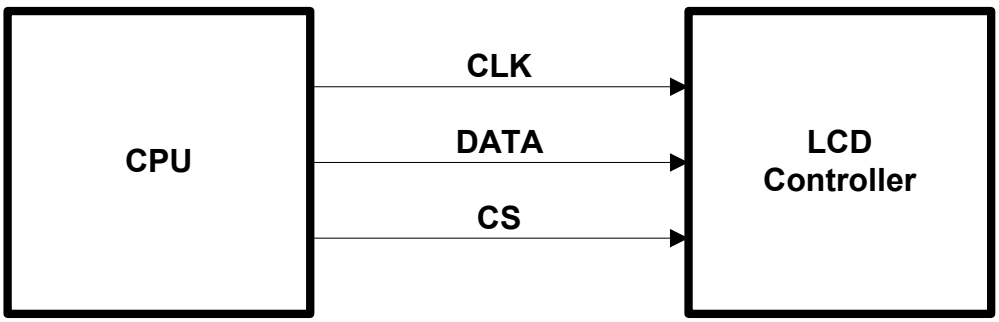
- LCD_X_SERIAL.c，针对串行接口的端口程序

需要注意的是，本例使用端口引脚来进行通信。但这在每一种 CPU 上都非常慢。客户需要对其进行优化，方法是使用 CPU 对这类通信的硬件支持功能。

28.2.4 间接接口 ——3 引脚 SPI

通过 4 引脚 SPI 接口连接 LCD 显示器，必须把 CLK、 DATA 和 CS 3 条线连接至 CPU。

典型框图



28.2.4.1 I/O 引脚连接程序示例

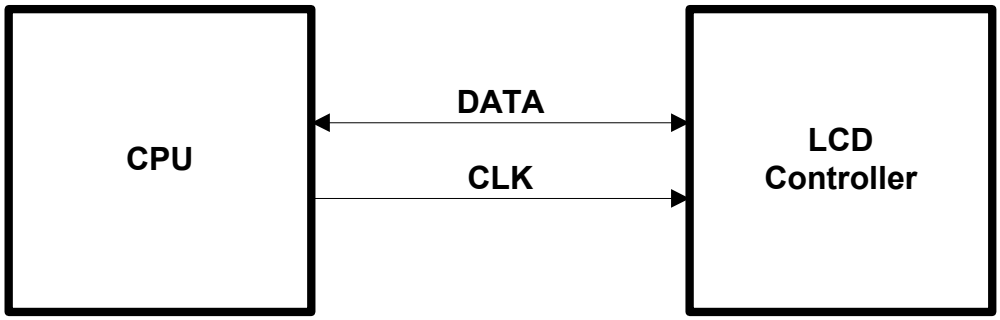
这种接口没有独立的线来区分传输至显示控制器的是数据还是命令。对此没有标准化的管理方法。有些控制器通过附加位来区分数据和命令，其他控制器则采用其他方式。
示例可在 Sample\LCD_X 文件夹下找到：

- LCD_X_Serial_3Pin.c，针对 3 引脚串行接口的端口程序
- LCD_X_Serial_3Wire.c，针对 3 引脚串行接口的端口程序

28.2.5 间接接口 ——I2C 总线

这种接口只采用 2 条线，同时使用标准协议与显示控制器进行通信。

典型框图



28.2.5.1 I/O 引脚连接程序示例

示例可在 Sample\LCD_X 文件夹下找到：

- LCD_X_I2CBUS.c，针对 I2C 总线接口的端口程序

与串行通信示例类似，本例通过端口线进行通信，速度不是很快。如果 CPU 支持这种通信，则应通过相应的硬件函数来优化这些程序。

28.3 硬件接口配置

下面说明如何配置显示驱动与显示控制器之间的硬件通信。

28.3.1 直接接口

使用直接接口对驱动进行硬件接口配置是通过指定视频存储器的地址来实现的。正常情况下，应使用 LCD_SetVRAMAddrEx() 程序。一般而言，启用驱动对视频存储器的存取不需要其他操作。更多详细信息，请参阅 “显示驱动 API”（第 883 页）。

28.3.2 间接接口

有 2 种显示驱动：

- 运行时间可配置驱动
- 编译时可配置驱动

这两种驱动的配置方式有所不同：

- 运行时间配置的意思是说，驱动可以编译而不配置。配置在运行过程中进行。这种驱动在存放于库中之后，在运行时仍然可以进行配置。
- 编译时配置驱动要求在配置头文件中进行配置，该文件在驱动编译时已包含于其中。

28.3.2.1 运行时间配置

运行时间可配置驱动在编译时不需要配置。因此，这种驱动可以用在预编译库中。
每种驱动都有自己的硬件接口设置函数。其方法是将一个指针传给 GUI_PORT_API 结构，该结构含有指向待使用的硬件程序的函数指针：

GUI_PORT_API 的元素

元素	数据类型	描述
8 位接口		
pfWrite8_A0	void (*)(U8 Data)	指向一个函数的指针，该函数向控制器写入一个字节，其中，C/D 线为低电平。
pfWrite8_A1	void (*)(U8 Data)	指向一个函数的指针，该函数向控制器写入一个字节，其中，C/D 线为高电平。
pfWriteM8_A0	void (*)(U8 * pData, int NumItems)	指向一个函数的指针，该函数向控制器写入多个字节，其中，C/D 线为低电平。
pfWriteM8_A1	void (*)(U8 * pData, int NumItems)	指向一个函数的指针，该函数向控制器写入多个字节，其中，C/D 线为高电平。
pfRead8_A0	U8 (*)(void)	指向一个函数的指针，该函数从控制器读取一个字节，其中，C/D 线为低电平。
pfRead8_A1	U8 (*)(void)	指向一个函数的指针，该函数从控制器读取一个字节，其中，C/D 线为高电平。
pfReadM8_A0	void (*)(U8 * pData, int NumItems)	指向一个函数的指针，该函数从控制器读取多个字节，其中，C/D 线为低电平。
pfReadM8_A1	void (*)(U8 * pData, int NumItems)	指向一个函数的指针，该函数从控制器读取多个字节，其中，C/D 线为高电平。
16 位接口		
pfWrite16_A0	void (*)(U16 Data)	指向一个函数的指针，该函数向控制器写入一个字，其中，C/D 线为低电平。
pfWrite16_A1	void (*)(U16 Data)	指向一个函数的指针，该函数向控制器写入一个字，其中，C/D 线为高电平。
pfWriteM16_A0	void (*)(U16 * pData, int NumItems)	指向一个函数的指针，该函数向控制器写入多个字，其中，C/D 线为低电平。
pfWriteM16_A1	void (*)(U16 * pData, int NumItems)	指向一个函数的指针，该函数向控制器写入多个字，其中，C/D 线为高电平。
pfRead16_A0	U16 (*)(void)	指向一个函数的指针，该函数从控制器读取一个字，其中，C/D 线为低电平。
pfRead16_A1	U16 (*)(void)	指向一个函数的指针，该函数从控制器读取一个字，其中，C/D 线为高电平。
pfReadM16_A0	void (*)(U16 * pData, int NumItems)	指向一个函数的指针，该函数从控制器读取多个字，其中，C/D 线为低电平。
pfReadM16_A1	void (*)(U16 * pData, int NumItems)	指向一个函数的指针，该函数从控制器读取多个字，其中，C/D 线为高电平。

该结构含有用于 8 位和 16 位存取的函数指针。每个驱动并不要求使用所有函数指针。如果要了解驱动要求使用哪些指针，请参见显示驱动文档。

示例

以下示例展示了 GUIDRV_Slin 驱动的配置。先创建和配置驱动，然后初始化 GUI_PORT_API 结构的必要函数指针，最后将其传给驱动：

```
GUI_DEVICE * pDevice;
CONFIG_SLIN Config = {0};
GUI_PORT_API PortAPI = {0};

//
// Set display driver and color conversion
//
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_SLIN_2, GUICC_2, 0, 0);
//
// Common display driver configuration
//
LCD_SetSizeEx (0, XSIZE, YSIZE);
LCD_SetVSizeEx(0, XSIZE, YSIZE);
//
// Driver specific configuration
//
Config.UseCache  = 1;
GUIDRV_SLin_Config(pDevice, &Config);
//
// Select display controller
//
GUIDRV_SLin_SetS1D13700(pDevice);
//
// Setup hardware access routines
//
PortAPI.pfWrite16_A0  = _Write0;
PortAPI.pfWrite16_A1  = _Write1;
PortAPI.pfWriteM16_A0 = _WriteM0;
PortAPI.pfRead16_A1   = _Read1;
GUIDRV_SLin_SetBus8(pDevice, &PortAPI);
```

有关详情，请参见运行时间可配置驱动的详细描述。

28.3.2.2 编译时间配置

编译时可配置驱动要求通过头文件进行配置。该配置文件在显示驱动编译过程中即包含于其中。编译时可配置驱动通过不同的宏来存取硬件。到底使用哪些宏取决于接口的详细情况。下面将说明哪些宏用于哪类接口。

简单型总线接口使用的宏

下表显示了使用的硬件存取宏：

类型	宏	描述
F	LCD_READ_A0	从 LCD 控制器读取一个字节，其中， A0 线为低电平。
F	LCD_READ_A1	从 LCD 控制器读取一个字节，其中， A0 线为高电平。
F	LCD_WRITE_A0	向显示控制器写入一个字节，其中， A0 线为低电平。
F	LCD_WRITE_A1	向显示控制器写入一个字节，其中， A0 线为高电平。
F	LCD_WRITEM_A1	向 LCD 控制器写入多个字节，其中， A0 线为高电平。

4 引脚 SPI 接口使用的宏

下表显示了使用的硬件存取宏：

类型	宏	描述
F	LCD_WRITE_A0	向显示控制器写入一个字节，其中， A0 (C/D) 线为低电平。
F	LCD_WRITE_A1	向显示控制器写入一个字节，其中， A0 (C/D) 线为高电平。
F	LCD_WRITEM_A1	向 LCD 控制器写入多个字节，其中， A0 (C/D) 线为高电平。

3 引脚 SPI 接口使用的宏

下表显示了使用的硬件存取宏：

类型	宏	描述
F	LCD_WRITE	向显示控制器写入一个字节。
F	LCD_WRITEM	向 LCD 控制器写入多个字节。

I2C 总线接口使用的宏

下表显示了使用的硬件存取宏：

类型	宏	描述
F	LCD_READ_A0	从 LCD 控制器读取一个状态字节。
F	LCD_READ_A1	从 LCD 控制器读取一个数据字节。
F	LCD_WRITE_A0	向显示控制器写入一个指令字节。
F	LCD_WRITE_A1	向显示控制器写入一个数据字节。
F	LCD_WRITEM_A1	向 LCD 控制器写入多个数据字节。

LCD_READ_A0

描述

从 LCD 控制器读取一个字节，其中， A0 (C/D) 线为低电平。

类型

函数替换

原型

```
#define LCD_READ_A0(Result)
```

参数	描述
Result	结果读取。这不是指针，而是值存储变量的占位符。

LCD_READ_A1

描述

从 LCD 控制器读取一个字节，其中， A0 (C/D) 线为高电平。

类型

函数替换

原型

```
#define LCD_READ_A1(Result)
```

参数	描述
Result	结果读取。这不是指针，而是值存储变量的占位符。

LCD_WRITE_A0

描述

向显示控制器写入一个字节，其中， A0 (C/D) 线为低电平。

类型

函数替换

原型

```
#define LCD_WRITE_A0(Byte)
```

参数	描述
Byte	待写入的字节。

LCD_WRITE_A1

描述

向显示控制器写入一个字节，其中， A0 (C/D) 线为高电平。

类型

函数替换

原型

```
#define LCD_WRITE_A1(Byte)
```

参数	描述
Byte	待写入的字节。

LCD_WRITEM_A1

描述

向 LCD 控制器写入多个字节，其中， A0 (C/D) 线为高电平。

类型

函数替换

原型

```
#define LCD_WRITEM_A1(paBytes, NumberOfBytes)
```

参数	描述
paBytes	指向第一个数据字节的占位符。
NumberOfBytes	待写入的数据字节数。

LCD_WRITE

描述

向 LCD 控制器写入一个字节。

类型

函数替换

原型

```
#define LCD_WRITE(Byte)
```

参数	描述
Byte	待写入的字节。

LCD_WRITEM

描述

向 LCD 控制器写入多个字节。

类型

函数替换

原型

```
#define LCD_WRITEM(paBytes, NumberOfBytes)
```

参数	描述
paBytes	指向第一个数据字节的占位符。
NumberOfBytes	待写入的数据字节数。

28.4 不可读取的显示器

有些配备间接接口的显示控制器不支持显示数据的回读。尤其是通过 SPI 接口连接的显示器一般都有这种限制。这种情况下，我们建议使用显示数据缓存。要详细了解如何启用显示数据缓存，请参见“显示驱动详细描述”（第 829 页）部分。
只有在 RAM 非常小的系统上，有时无法使用显示数据缓存。如果显示器不可读取，而且不能使用显示数据缓存，则 emWin 的部分功能将失去效用。以下列出这些功能：

- 游标和 Sprite
- XOR 操作， EDIT 和 MULTIEDIT 两种小工具中用于文本游标
- Alpha 混合
- 抗锯齿

这种情况适用于一个数据单元 （8 位或 16 位）代表一个像素的所有驱动。如果一个数据单位代表一个以上的像素，而且无显示数据缓存可用且显示器不可读取，则不能使用这种显示驱动。GUIDRV_Page1bpp 驱动即是其中之一，其中，一个字节代表 8 个像素。

28.5 显示方向

如果原始显示方面不符合要求，则可通过不同的方式来改变显示方向：
- 通过驱动配置目标显示方向
- 使用 GUI_SetOrientation()

28.5.1 通过驱动配置显示方向

如果显示驱动支持不同的显示方向，建议用驱动来设置正确的方向。显示方向的具体配置方法取决于所用的显示驱动。虽然多数常见驱动支持运行时间配置显示方向，但有些驱动需要在编译时进行配置。

28.5.1.1 运行时间配置

多数常见驱动的显示方向通过利用正确的宏在 LCD_X_Config() 中创建显示驱动器件来决定。请参见 “GUIDRV_Lin”（第 841 页）部分，了解可用来创建该驱动的所有可用标识符。其中列出了所有可用的宏及其方向。

28.5.1.2 编译时间配置

有些采用间接接口的驱动 （如 GUIDRV_CompactColor_16）的显示方向需要在编译时在驱动的配置文件中进行配置。

显示方向

共有 8 种可能的显示方向；显示器可以转动 0°、90°、180° 或 270°，也可从上或从下查看。默认方向为 0° 上视图。这些 4*2 = 8 种不同的显示方向也可表示为 3 个二进制开关的组合：X 镜像、Y 镜像和 X/Y 交换。为此，下列二进制配置宏可以用于每一种驱动，支持任意组合。如果显示方向欠佳，请查看下表中的配置开头，使其正常工作。方向处理方式如下：先生成 X 和 Y 镜像，然后交换（若选择）。

显示	驱动配置文件中的方向宏	显示	驱动配置文件中的方向宏
	无需方向宏		#define LCD_MIRROR_Y 1
	#define LCD_MIRROR_X 1		#define LCD_MIRROR_X 1 #define LCD_MIRROR_Y 1
	#define LCD_SWAP_XY 1		#define LCD_SWAP_XY 1 #define LCD_MIRROR_Y 1
	#define LCD_SWAP_XY 1 #define LCD_MIRROR_X 1		#define LCD_SWAP_XY 1 #define LCD_MIRROR_X 1 #define LCD_MIRROR_Y 1

如需详细了解如何同时使用多个方向，请参见 “运行时间屏幕旋转”（第 754 页）部分。

28.5.2 通过函数配置显示方向

另一种设置显示方向的方法是调用 GUI_SetOrientation()。如果显示驱动无法使用，则建议使用该函数。

GUI_SetOrientation()

描述

该函数利用一个旋转器件来改变显示方向。

原型

```
int GUI_SetOrientation(int Orientation);
```

参数	描述
Orientation	请参见下表，以了解有效值。

显示结果	GUI_SetOrientation() 使用的值	显示结果	GUI_SetOrientation() 使用的值
	0		GUI_MIRROR_Y
	GUI_MIRROR_X		GUI_MIRROR_X | GUI_MIRROR_Y
	GUI_SWAP_XY		GUI_SWAP_XY | GUI_MIRROR_Y
	GUI_SWAP_XY | GUI_MIRROR_X		GUI_SWAP_XY | GUI_MIRROR_X | GUI_MIRROR_Y

返回值

如果成功，返回值为 0；如果发生错误，则返回值为 1。

其他信息

旋转器件覆盖内部屏幕缓冲器中的整个虚拟屏幕。为此，使用该函数需要额外的存储器空间，以便存储额外的屏幕缓冲器。所需字节数可按以下公式计算：

虚拟 xSize * 虚拟 ySize * BytesPerPixel

对于 1-8bpp 的系统， BytesPerPixel （每像素字节数）为 1， 8bpp 至 16bpp 的系统为 2，超过 16bpp 的系统为 4。
各绘制操作首先更新该缓冲器。然后，受影响的像素被传给显示驱动器件。

GUI_SetOrientationEx()

描述

该函数利用一个旋转器件来改变指定层中的显示方向。

原型

```
int GUI_SetOrientation(int Orientation, int LayerIndex);
```

参数	描述
Orientation	请参见 “GUI_SetOrientation()”（第 825 页），以了解有效值。
LayerIndex	需要 （重新）配置方向的层的索引。

返回值

如果成功，返回值为 0；如果发生错误，则返回值为 1。

其他信息

请参阅 “GUI_SetOrientation()”（第 825 页）。

28.6 显示驱动回调函数

显示驱动要求使用回调函数。驱动需要在多个任务中调用该函数。其中一个任务是使显示驱动开始工作，另见“配置”一章。同时，要求执行硬件相关操作（如开启和关闭显示、设置查找表入口等）的其他任务也要调用该函数。

LCD_X_DisplayDriver()

描述

这是显示驱动的回调函数。显示驱动在多个任务中要调用该函数。将一个命令和一个指向某个数据结构的指针传给回调程序。该命令告诉回调函数需要做什么。如果命令需要参数，则通过数据指针 pData 传送。指向格式因命令而异的结构。

原型

```
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * pData);
```

参数	描述
LayerIndex	基数为零的层索引。
Cmd	待使执行的命令。详见后文。
pData	指向某种数据结构的指针。

返回值

如果发生错误，程序的返回值为 -2；如果函数未处理命令，则返回值为 -1；如果命令已成功执行，则返回值为 0。

28.6.1 传给回调函数的命令

以下将说明传给回调函数的常见命令。要了解有关显示驱动所用具体命令的详情，请参见“显示驱动详细描述”（第 829 页）部分。相关内容在 “其他回调命令”部分。

LCD_X_INITCONTROLLER

如上所述，应用必须在回调程序收到命令时初始化显示控制器，并使其开始工作。该命令不传递参数。通常情况下，收到该命令后，应调用用于初始化显示控制器寄存器的初始化程序。

参数

无。

LCD_X_SETVRAMADDR_INFO

该命令由驱动传递，用于告诉回调程序视频 RAM 的起始地址。典型的反应是把该地址写入帧缓冲器起始地址寄存器。

参数

pData 指向一种属于 LCD_X_SETVRAMADDR_INFO 类的数据结构：

数据类型	元素	描述
void *	pVRAM	指向视频 RAM 的起始地址。该地址一般写入显示控制器的视频 RAM 基地址寄存器。

LCD_X_ON

该命令开启显示器。

参数

无

LCD_X_OFF

该命令关闭显示器。

参数

无

LCD_X_SETLUTENTRY

需要设置查找表入口。典型反应是将入口写入显示控制器的查找表。

参数

pData 指向一种属于 LCD_X_SETLUTENTRY_INFO 类的数据结构：

数据类型	元素	描述
LCD_COLOR	Color	等写入 LUT 的 RGB 颜色值。需要注意的是，硬件要求的格式可能因 RGB 格式而异。
U8	Pos	待设置的 LUT 入口索引 （基数为零）。

LCD_X_SETORG

该函数用于虚拟屏幕。如果需要设置显示原点，则需调用该函数。典型反应是修改帧缓冲器起始地址。

参数

pData 指向一种属于 LCD_X_SETORG_INFO 类的数据结构：

数据类型	元素	描述
int	xPos	虚拟屏幕中物理显示位置的新的 X 位置。
int	yPos	虚拟屏幕中物理显示位置的新的 Y 位置。

28.7 显示驱动详细描述

28.7.1 GUIDRV_BitPlains

该驱动专门针对无显示控制器的系统而开发。它以独立的位平面管理每个颜色位。也就是说，如果色彩深度（比如）是每像素 4 位，则驱动会管理 4 个位平面，每个平面含 1 位。
该驱动的最初目的是通过 SPI 接口驱动配备 R323C/111 CPU 的单色和彩色 TFT。但是，该驱动也可用于类似的应用。
这个驱动只管理位平面的内容，不含因显示控制器而异的任何代码。

支持的硬件

控制器

无。

每像素位数

该驱动针对每像素 1 至 8 位的颜色深度而开发。

接口

需要写一个由应用决定的程序，该程序利用位平面的内容来产生显示用的颜色信号。该驱动附有针对 R32C/111 CPU 的示例，其中，利用 SPI 接口通过计时器中断程序来刷新显示器。

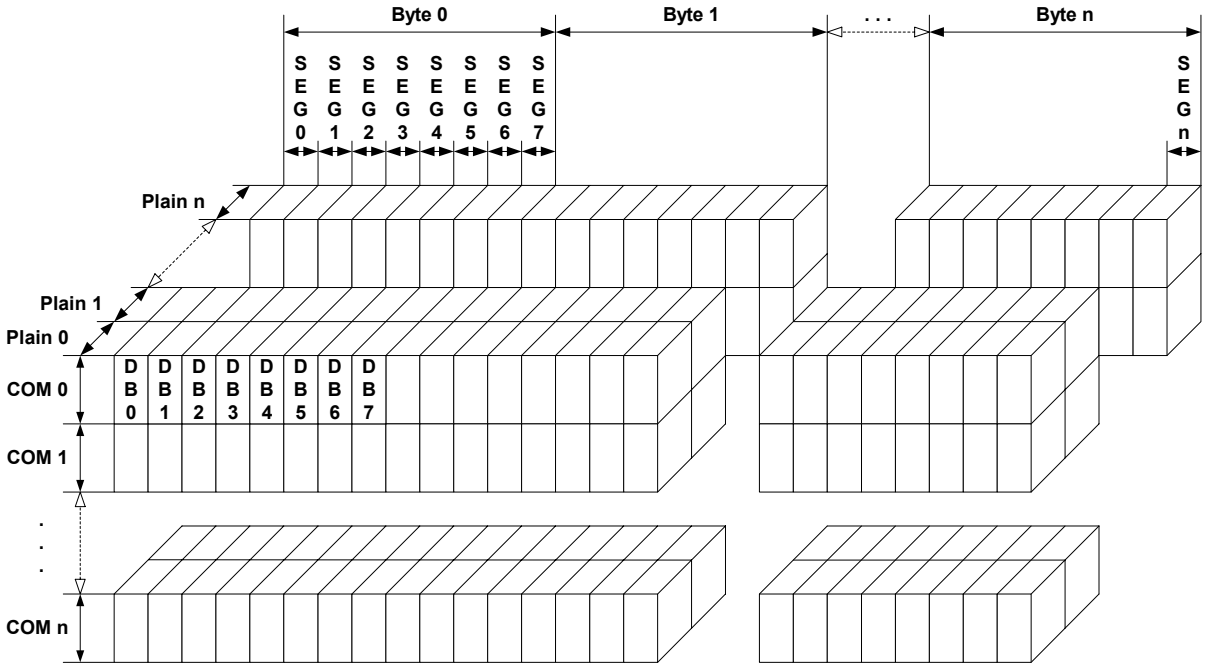
驱动选择

举例来说，若要针对给定的显示器使用 GUIDRV_BitPlains，可用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_BITPLAINS, GUICC_M111, 0, 0);
```

请参见 “颜色”（第 227 页），以了解有关利用正确的调色板模式的更多信息。

显示数据 RAM 的组织



上图展示了显示存储器与显示 SEG 和 COM 线之间的关系。显示存储器针对每位颜色分为独立的平面。这就是说，每个像素的第 0 位存储于平面 0 中，第 1 位存储于平面 1 中，以此类推，不一而足。这种方法的优势在于，可以快速访问显示数据的每个颜色位。

RAM 要求

显示存储器区所需大小可按以下公式计算：

```
大小 = BitsPerPixel * (LCD_XSIZE + 7) / 8 * LCD_YSIZE
```

需要注意的是，指向位平面区的指针需要传给驱动的配置程序。不是在驱动中分配，而是从应用中进行分配。

硬件配置

正常情况下，硬件接口是一种中断服务程序 (ISR)，其作用是更新显示器。该驱动附有一个以 "C" 语言写的示例。该程序可以当作一个使用案例。

其他运行时间配置

下表列出了该驱动的可用运行时间配置程序：

程序	说明
GUIDRV_BitPlains_Config()	将一个指向某种 CONFIG_BITPLAINS 结构的指针传递给驱动。
LCD_SetVRAMAddrEx()	将一个指向某种 CONFIG_VRAM_BITPLAINS 结构的指针传递给驱动。请参见下面的说明。有关该函数的描述可在第 890 页找到。

CONFIG_VRAM_BITPLAINS 的元素

数据类型	元素	描述
U8 *	apVRAM	指向驱动用于各个位平面的存储器位置的指针阵列。举例而言，如果驱动工作于 2bpp 模式，则只使用前 2 个指针 （每位颜色信息 1 个平面）。

GUIDRV_BitPlains_Config()

描述

该函数将一个指向某种 CONFIG_BITPLAINS 结构的指针传递给驱动。

原型

```
void GUIDRV_BitPlains_Config(GUI_DEVICE        * pDevice,
                             CONFIG_BITPLAINS * pConfig);
```

参数	描述
pDevice	指向驱动器件的指针。
pConfig	指向某种 CONFIG_BITPLAINS 结构的指针，说明见下文。

CONFIG_BITPLAINS 的元素

数据类型	元素	描述
int	Mirror	配置开关，用于产生显示数据位的镜像。

配置示例

```
//
// Data arrays to be used by the display driver
//
static U8 _aPlain_0[BYTES_PER_LINE * YSIZE_PHYS];
static U8 _aPlain_1[BYTES_PER_LINE * YSIZE_PHYS];
static U8 _aPlain_2[BYTES_PER_LINE * YSIZE_PHYS];

//
// Structure to be passed to the driver
//
static struct {
  U8 * apVRAM[8];
} _VRAM_Desc = {
  _aPlain_0,
  _aPlain_1,
  _aPlain_2,
};

void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_BITPLAINS, COLOR_CONVERSION, 0, 0);
  //
  // Display driver configuration, required for Lin-driver
  //
  if (LCD_GetSwapXY()) {
    LCD_SetSizeEx (0, YSIZE_PHYS, XSIZE_PHYS);
    LCD_SetVSizeEx(0, YSIZE_PHYS, XSIZE_PHYS);
  } else {
    LCD_SetSizeEx (0, XSIZE_PHYS, YSIZE_PHYS);
    LCD_SetVSizeEx(0, XSIZE_PHYS, YSIZE_PHYS);
  }
  //
  // Initialize VRAM access of the driver
  //
  LCD_SetVRAMAddrEx(0, (void *)&_VRAM_Desc);
}
```


28.7.2 GUIDRV_Dist

GUIDRV_Dist 用于支持含有多个控制器的显示器。可以支持多个显示区，每个区由独立的显示控制器驱动。分布式驱动将绘图操作传递至相应的显示驱动。同时支持交叠操作。这些情况下，基于每个受影响的控制器，将操作分成子操作。

支持的硬件

分布式驱动能支持各种运行时间可配置显示驱动。需要注意的是，要求配置的每一个显示驱动必须使用与分布式驱动相关的颜色转换模式。

驱动选择

为了使用该驱动，需要创建以下调用：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_DIST, COLOR_CONVERSION, 0, Layer);
```

RAM 要求

无。

运行时间配置

创建驱动之后，同时还须创建实际的显示驱动，并添加到分布式器件之中：

```
pDevice0 = GUI_DEVICE_Create(DISPLAY_DRIVER, COLOR_CONVERSION, 0, -1);
pDevice1 = GUI_DEVICE_Create(DISPLAY_DRIVER, COLOR_CONVERSION, 0, -1);
GUIDRV_Dist_AddDriver(pDevice, pDevice0, &Rect0);
GUIDRV_Dist_AddDriver(pDevice, pDevice1, &Rect1);
```

GUIDRV_Dist_AddDriver()

描述

将显示驱动添加到分布式驱动中。

原型

```
void GUIDRV_Dist_AddDriver(GUI_DEVICE * pDevice,
                           GUI_DEVICE * pDriver, GUI_RECT * pRect);
```

参数	描述
pDevice	指向已经创建的分布式器件的指针。
pDriver	指向待添加的已经创建的驱动器件的指针。
pRect	指向其中的输出将影响驱动的矩形的指针。

配置示例

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion for 1st layer
  //
  pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_DIST, COLOR_CONVERSION, 0, 0);
  //
  // Display size configuration
  //
  LCD_SetSizeEx (0, XSIZE_PHYS,  YSIZE_PHYS);
  LCD_SetVSizeEx(0, VXSIZE_PHYS, VYSIZE_PHYS);
  //
  // Create first display driver
  //
  pDevice0 = GUI_DEVICE_Create(DISPLAY_DRIVER, COLOR_CONVERSION, 0, -1);
  //
  // Configuration of first driver
  //
  ...
  //
  // Create second display driver
  //
```



```
  pDevice1 = GUI_DEVICE_Create(DISPLAY_DRIVER, COLOR_CONVERSION, 0, -1);
  //
  // Configuration of second driver
  //
  ...
  //
  // Add display drivers to distribution driver
  //
  Rect0.x0 = 0;
  Rect0.y0 = 160;
  Rect0.x1 = 223;
  Rect0.y1 = 319;
  GUIDRV_Dist_AddDriver(pDevice, pDevice0, &Rect0);
  Rect1.x0 = 0;
  Rect1.y0 = 0;
  Rect1.x1 = 223;
  Rect1.y1 = 159;
  GUIDRV_Dist_AddDriver(pDevice, pDevice1, &Rect1);
}
```

28.7.3 GUIDRV_FlexColor

支持的硬件

控制器

该驱动支持以下显示控制器：

- 爱普生 S1D19122
- 奇景 HX8353、HX8325A
- Ilitek ILI9320、ILI9325、ILI9328
- LG 电子 LGDP4531、LGDP4551
- 联咏 NT39122
- OriseTech SPFD5408、SPFD54124C、SPFD5414D
- 瑞萨 R61505、R61516、R61580
- 矽创 ST7628、ST7637、ST7735
- 所罗门 SSD1355、SSD1961、SSD1963、SSD2119
- Syncoam SEPS525

每像素位数

支持的颜色深度为 16 bpp 和 18 bpp。

接口

该驱动支持 8 位、9 位和 16 位间接接口。

驱动选择

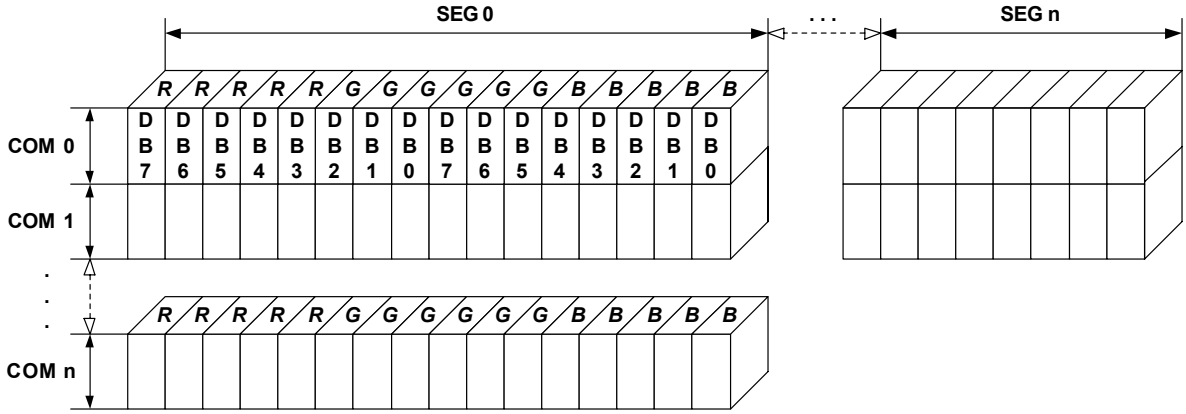
为了使用该驱动，需要创建以下调用：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_FLEXCOLOR,
                                   COLOR_CONVERSION, 0, Layer);
```

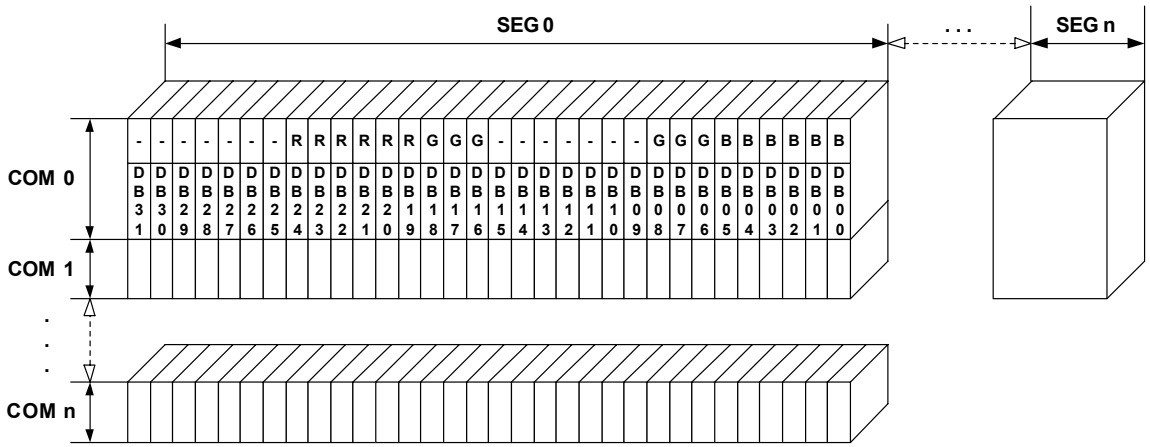
为了选择正确的颜色转换模式，请参见 “颜色”（第 227 页），以了解有关调色板模式的详细信息。

显示数据 RAM 的组织

16 bits per pixel, fixed palette = 565



18 bits per pixel, fixed palette = 666_9



RAM 要求

该显示驱动需要 500 字节的应用存储器才能正常工作。也可使用或不使用显示数据缓存，该缓存中含有显示数据 RAM 中内容的完整拷贝。缓存所用存储器大小为：`LCD_XSIZE * LCD_YSIZE * BytesPerPixel`。对于 16bpp 模式，BytesPerPixel 为 2，18bpp 模式为 4。使用缓存可以避免显示控制器在 XOR 绘图操作过程进行读操作，而且还能加快字符串输出操作。

配置程序

程序	描述
GUIDRV_FlexColor_SetFunc()	配置总线、缓存和硬件程序。
GUIDRV_FlexColor_Config()	配置 SEG 和 COM 线的方向和偏移。

GUIDRV_FlexColor_SetFunc()

描述

配置总线宽度、缓存用量和硬件程序。

原型

```
void GUIDRV_FlexColor_SetFunc(GUI_DEVICE   * pDevice,
                              GUI_PORT_API * pHW_API,
                              void (* pfFunc)(GUI_DEVICE * pDevice),
                              void (* pfMode)(GUI_DEVICE * pDevice));
```

参数	描述
pDevice	指向驱动器件结构的指针。
pHW_API	指向某种 GUI_PORT_API 结构的指针。参见下方的所需程序部分。
pfFunc	控制器选择宏。参见下表。
pfMode	参见下表。

参数 pfFunc 的允许值 支持的显示控制器	
GUIDRV_FLEXCOLOR_F66708	设置驱动，以使用下列控制器之一： - Ilitek ILI9320、ILI9325、ILI9328 - LG 电子 LGDP4531、LGDP4551 - OriseTech SPFD5408 - 瑞萨 R61505、R61580
GUIDRV_FLEXCOLOR_F66709	设置驱动，以使用下列控制器之一： - 爱普生 S1D19122 - 奇景 HX8353、HX8325A - 联咏 NT39122 - OriseTech SPFD54124C、SPFD5414D - 瑞萨 R61516 - 矽创 ST7628、ST7637、ST7735 - 所罗门 SSD1355、SSD1961、SSD1963
GUIDRV_FLEXCOLOR_F66714	设置驱动，以使用下列显示控制器： - 所罗门 SSD2119
GUIDRV_FLEXCOLOR_F66718	设置驱动，以使用下列显示控制器： - Syncoam SEPS525

上表列出的显示控制器为目前已知的与该驱动兼容的控制器。需要注意的是，选择宏使用的数值与驱动 GUIDRV_CompactColor_16 的部分 LCD_CONTROLLER 宏兼容。这样有利于从编译时可配置的 GUIDRV_CompactColor_16 移植到运行时间可配置的 GUIDRV_FlexColor。

参数 pfMode 的允许值	
GUIDRV_FLEXCOLOR_M16C0B8	16bpp，无缓存， 8 位总线
GUIDRV_FLEXCOLOR_M16C1B8	16bpp，有缓存， 8 位总线
GUIDRV_FLEXCOLOR_M16C0B16	16bpp，无缓存， 16 位总线
GUIDRV_FLEXCOLOR_M16C1B16	16bpp，有缓存， 16 位总线
GUIDRV_FLEXCOLOR_M18C0B9	18bpp，无缓存， 9 位总线
GUIDRV_FLEXCOLOR_M18C1B9	18bpp，有缓存， 9 位总线

每个控制器选择都支持不同的操作模式。下表所示为各控制器支持的模式：

选择宏	M16C0B8	M16C1B8	M16C0B16	M16C1B16	M18C0B9	M18C1B9
GUIDRV_FLEXCOLOR_F66708	X	X	X	X	-	-
GUIDRV_FLEXCOLOR_F66709	X	X	X	X	-	-
GUIDRV_FLEXCOLOR_F66714	X	X	X	X	X	X
GUIDRV_FLEXCOLOR_F66718	X	X	X	X	X	X

所需的 GUI_PORT_API 程序

具体需要的 GUI_PORT_API 程序取决于使用的接口。如果使用缓存，则各接口无需使用数据读取程序，其中：

8 位接口

元素	数据类型
pfWrite8_A0	void (*)(U8 Data)
pfWrite8_A1	void (*)(U8 Data)
pfWriteM8_A1	void (*)(U8 * pData, int NumItems)
pfReadM8_A1	void (*)(U8 * pData, int NumItems)

16 位接口

元素	数据类型
pfWrite16_A0	void (*)(U16 Data)
pfWrite16_A1	void (*)(U16 Data)
pfWriteM16_A1	void (*)(U16 * pData, int NumItems)
pfReadM16_A1	void (*)(U16 * pData, int NumItems)

9 位接口

使用 9 位接口且颜色深度为 18 bpp 时，显示控制器利用 D10-D17 线（8 位）来传递指令，用 D9-D17 线（9 位）传递数据。因此，这种情况下， D9-D17 线连接至 CPU 的接口线。为了尽量加快像素数据的处理速度，驱动对每个像素使用 2 个 16 位数据值（0000000R RRRRRGGG 和 0000000G GGBBBBBB），其中，只有前 9 位都含有像素数据，这些数据被传给硬件程序。
如上所述，命令和参数接口使用显示控制器的 D10-D17 线（8 位）。驱动将 16 位值传递给硬件程序，其中， 1-8 位已经含有显示控制器 D10-D17 线的信息（0 位和 9-15 位未使用）。因此，硬件程序中不需要移位操作。

元素	数据类型	描述
pfWrite16_A0	void (*)(U16 Data)	待设置的寄存器 (DB1-DB9)
pfWrite16_A1	void (*)(U16 Data)	待设置的参数 (DB1-DB9)
pfWriteM16_A1	void (*)(U16 * pData, int NumItems)	待写入的数据 (DB0-DB9)
pfReadM16_A1	void (*)(U16 * pData, int NumItems)	读取的数据 (DB0-DB9)

GUIDRV_FlexColor_Config()

描述

配置 SEG 和 COM 线的方向和偏移。

原型

```
void GUIDRV_FlexColor_Config(GUI_DEVICE       * pDevice,
                             CONFIG_FLEXCOLOR * pConfig);
```

参数	描述
pDevice	指向待配置的器件的指针。
pConfig	指向某种 CONFIG_FLEXCOLOR 结构的指针。参见下面的元素列表。

CONFIG_FLEXCOLOR 的元素

数据类型	元素	描述
int	FirstSEG	第一分段线。
int	FirstCOM	第一公用线。
int	Orientation	下表中的一个或多个 “OR” 组合值。
U16	RegEntryMode	正常情况下，显示控制器使用一个寄存器的 3 位来定义所需显示方向。通常为 ID0、ID1 和 AM 3 位。为控制其他位的内容，可以使 RegEntryMode 元素。驱动在初始化过程中将该值与所需方向位结合。

参数 Orientation 的允许值	
GUI_MIRROR_X	对 X 轴进行镜像显示
GUI_MIRROR_Y	对 Y 轴进行镜像显示
GUI_SWAP_XY	交换 X 轴和 Y 轴

28.7.4 GUIDRV_IST3088

支持的硬件

控制器

该驱动支持以下显示控制器：

- IST3088
- IST3257

每像素位数

支持的颜色深度为 4 bpp。

接口

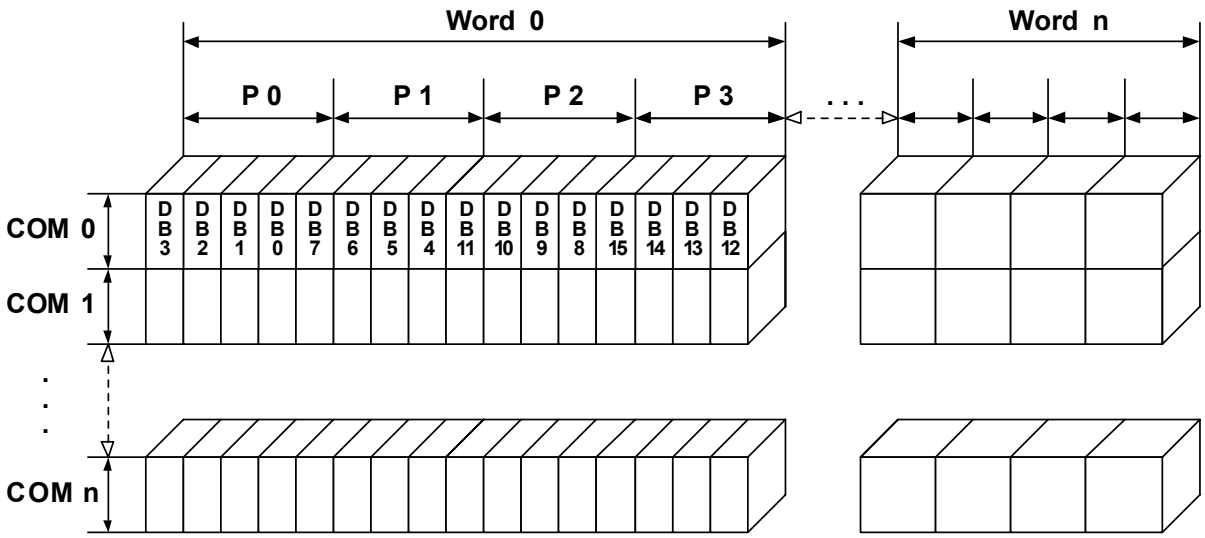
该驱动支持 16 位间接接口。

驱动选择

若要针对给定的显示器使用 GUIDRV_IST3088，应使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_IST3088_4, GUICC_4, 0, 0);
```

显示数据 RAM 的组织



上图展示了显示存储器与 LCD SEG 和 COM 线之间的关系。

RAM 要求

该显示驱动可以使用，也可不使用显示数据缓存，该缓存中含有显示数据 RAM 中内容的完整拷贝。缓存所用存储器大小为：LCD_XSIZE * LCD_YSIZE / 2。

其他运行时间配置

下表列出了该驱动的可用运行时间配置程序：

程序	说明
GUIDRV_IST3088_SetBus16	告知驱动使用 16 位间接接口，将把指向某种 GUI_PORT_API 结构的指针传给驱动。

GUIDRV_IST3088_SetBus16()

描述

告知驱动使用 16 位间接接口，将把指向某种 GUI_PORT_API 结构的指针传给驱动，该指针中含有待使用的硬件程序的函数指针。

原型

```
void GUIDRV_IST3088_SetBus16(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

参数	描述
pDevice	指向驱动器件的指针。
pHW_API	指向某种 GUI_PORT_API 结构的指针。参见下方的所需程序部分。

所需的 GUI_PORT_API 程序

元素	数据类型
pfWrite16_A0	void (*)(U16 Data)
pfWrite16_A1	void (*)(U16 Data)
pfWriteM16_A1	void (*)(U16 * pData, int NumItems)

特殊要求

该驱动需要工作于固定调色板模式 GUICC_4。驱动不支持其他调色板模式或固定调色板模式。颜色转换模式应使用 GUICC_4。

28.7.5 GUIDRV_Lin

该驱动支持所有配备了可通过直接接口存取的线性视频存储器的显示控制器。既可以使用显示控制器，也可以不使用。该驱动只管理视频存储器中的内容，它既不会向显示控制器发送任何命令，也不假定采用任何特定的寄存器。因此，该驱动独立于显示控制器的寄存器接口，其作用是管理每个线性映射视频存储器。

支持的硬件

控制器

该驱动支持所有配备线性映射视频存储器的系统。

每像素位数

支持的颜色深度为每像素 1 位、 2 位、 4 位、 8 位、 16 位、 24 位和 32 位。

接口

该驱动支持从CPU到视频存储器的全总线接口。视频存储器必须能以8位、16位或32位的模式存取。

颜色深度和显示方向

该驱动由几个文件构成。其命名方式为 _[O]_BPP.c，其中，可选的“O”表示目标显示方向，“BPP”表示颜色深度。下表列出了在初始化过程中用来创建和链接驱动的驱动文件及配置宏。

标识符	颜色深度和显示方向
GUIDRV_LIN_1	1bpp，默认方向
GUIDRV_LIN_2	2bpp，默认方向
GUIDRV_LIN_4	4bpp，默认方向
GUIDRV_LIN_8	8bpp，默认方向
GUIDRV_LIN_16	16bpp，默认方向
GUIDRV_LIN_OX_16	16bpp， X 轴镜像
GUIDRV_LIN_OXY_16	16bpp， X 轴和 Y 轴镜像
GUIDRV_LIN_OY_16	16bpp， Y 轴镜像
GUIDRV_LIN_OS_16	16bpp， X 轴和 Y 轴交换
GUIDRV_LIN_OSX_16	16bpp， X 轴镜像方式， X 轴和 Y 轴交换
GUIDRV_LIN_OSY_16	16bpp， Y 轴镜像， X 轴和 Y 轴交换
GUIDRV_LIN_24	24bpp，默认方向
GUIDRV_LIN_OX_24	24bpp， X 轴镜像
GUIDRV_LIN_OXY_24	24bpp， X 轴和 Y 轴镜像
GUIDRV_LIN_OY_24	24bpp， Y 轴镜像
GUIDRV_LIN_OS_24	24bpp， X 轴和 Y 轴交换
GUIDRV_LIN_OSX_24	24bpp， X 轴镜像， X 轴和 Y 轴交换
GUIDRV_LIN_OSY_24	24bpp， Y 轴镜像， X 轴和 Y 轴交换
GUIDRV_LIN_32	32bpp，默认方向
GUIDRV_LIN_OX_32	32bpp， X 轴镜像
GUIDRV_LIN_OXY_32	32bpp， X 轴和 Y 轴镜像
GUIDRV_LIN_OY_32	32bpp， Y 轴镜像
GUIDRV_LIN_OS_32	32bpp， X 轴和 Y 轴交换
GUIDRV_LIN_OSX_32	32bpp， X 轴镜像， X 轴和 Y 轴交换
GUIDRV_LIN_OSY_32	32bpp， Y 轴镜像， X 轴和 Y 轴交换

上表列出的标识符的作用是选择驱动。无论对驱动的方向和颜色深度进行怎样的组合，都是可行的。需要注意的是，目前驱动只随附了部分组合。如果还未获得所需的组合，请来函索取。

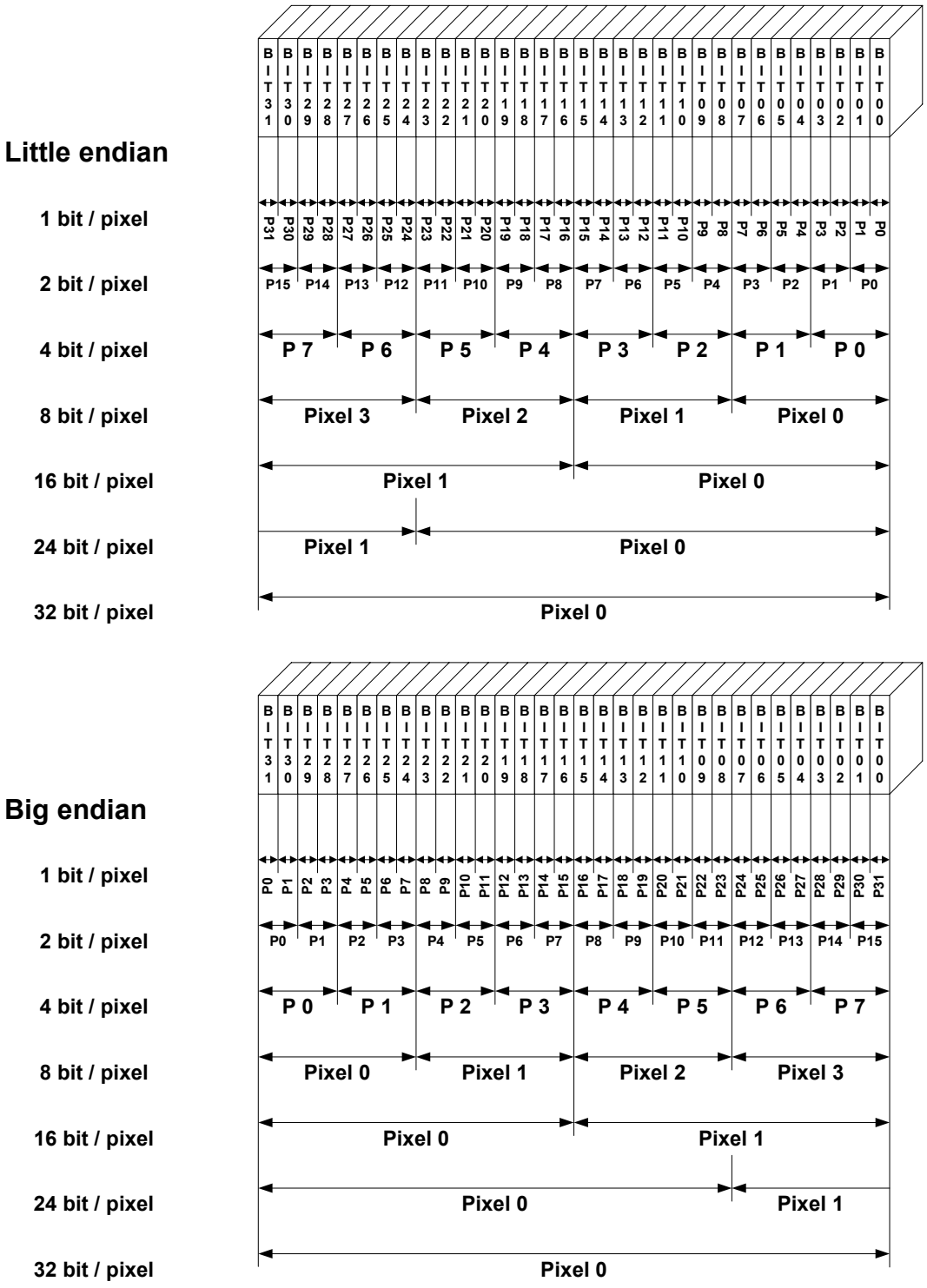
驱动选择

举例来说，如果指定的显示器需要使用该驱动，则可使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_LIN_OX_16, GUICC_565, 0, 0);
```

请参见 “颜色”（第 227 页），以了解有关使用正确的调色板模式的更多信息。

显示数据 RAM 的组织



上图展示了显示存储器与 LCD 像素在颜色深度和端字节序模式上的关系。

小端字节序视频模式

使用最低有效位，并首先输出。最低有效位用于第一个 （最左边）像素。

大端字节序视频模式

使用最高有效位，并首先输出。最大有效位用于第一个 （最左边）像素。

RAM 要求

无。

可用的配置宏 （编译时间配置）

下表列出了必须针对硬件存取进行定义的宏：

宏	描述
LCD_ENDIAN_BIG	大端字节序模式应设为 1，（默认）小端字节序模式设为 0。

可用的配置程序 （运行时间配置）

下表列出了可用的运行时间配置程序：

程序	描述
LCD_SetDevFunc()	可用于设置可选程序或定制程序。
LCD_SetSizeEx()	更改可见区的尺寸。
LCD_SetVRAMAddrEx()	更改视频 RAM 的起始地址。
LCD_SetVSizeEx()	更改虚拟显示区的尺寸。

LCD_SetDevFunc() 支持的值

下表展示了该函数支持的值：

值	描述
LCD_DEVFUNC_COPYBUFFER	可用于设置定制的缓冲器复制程序。需与多个缓冲器配合使用。
LCD_DEVFUNC_COPYRECT	可用于设置定制的显示器矩形区复制程序。
LCD_DEVFUNC_DRAWBMP_1BPP	可用于设置定制的 1bpp 位图绘图程序。仅适用于需要用 BitBLT 引擎来绘制文本和 1bpp 位图的情况。
LCD_DEVFUNC_FILLRECT	可用于设置定制的矩形填充程序。适用于需要用 BitBLT 引擎来执行填充操作等情况。

请参见 “LCD 层程序”（第 885 页），以了解有关使用正确的调色板模式的更多信息。

配置示例

以下示例展示了如何使用该驱动来创建显示驱动器件及其配置方法：

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_LIN_8,    // Display driver
                           GUICC_8666,      // Color conversion
                           0, 0);
  //
  // Display driver configuration
  //
  LCD_SetSizeEx    (0, 320,  240);          // Physical display size in pixels
  LCD_SetVSizeEx   (0, 320,  480);          // Virtual  display size in pixels
  LCD_SetVRAMAddrEx(0, (void *)0x20000000); // Video RAM start address
}
```

在有缓存的系统中使用 Lin 驱动

有几条非常简单的规则需要遵守：

规则 1

所有缓存 （若适用）应全面启用。这里指采用独立缓存的系统中的 I 缓存和 D 缓存。

规则 2

应将所有代码和数据置于可缓存区，以实现最佳性能。如果应用的其他部分要求将部分或分部数据置于不可缓存区中，虽然这不会构成问题，但会造成性能下降。

规则 3

规则 2 的唯一例外是，显示存储器（实际上是一个共享的存储器区， CPU 和 LCD 控制器 DMA 同时对其进行存取）应置于非缓存区中。在有 MMU 的许多系统中，可以通过把 RAM 两次映射到虚拟地址空间中来实现：在正常地址处， RAM 有缓存，在第二个地址处，无缓存。传给驱动在的 VRAM 地址应为非缓存地址。

28.7.6 GUIDRV_S1D13748

支持的硬件

控制器

该驱动已通过爱普生 S1D13748 的测试。

每像素位数

支持的颜色深度为 16 bpp。

接口

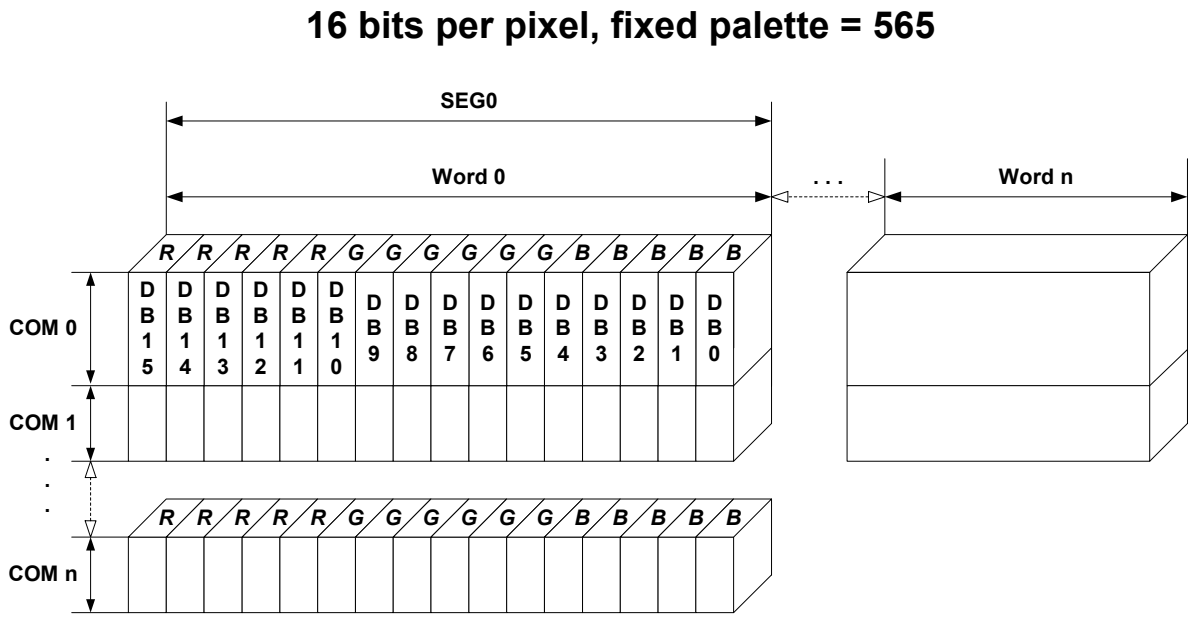
该驱动支持 16 位并行 （简单总线）接口。

驱动选择

如果指定的显示器需要使用 GUIDRV_S1D13748，则可使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_S1D13748, GUICC_M565, 0, 0);
```

显示数据 RAM 的组织



上图展示了显示存储器与 LCD SEG 和 COM 线之间的关系。

RAM 要求

约 500 字节。

其他运行时间配置

下表列出了该驱动的可用运行时间配置程序：

程序	说明
GUIDRV_S1D13748_Config	将一个指向某种 CONFIG_S1D13748 结构的指针传递给驱动。
GUIDRV_S1D13748_SetBus_16	告知驱动使用 16 位间接接口，将把指向某种 GUI_PORT_API 结构的指针传给驱动。

GUIDRV_S1D13748_Config()

描述

将一个指向某种 CONFIG_S1D13748 结构的指针传递给驱动。

原型

```
void GUIDRV_S1D13748_Config(GUI_DEVICE       * pDevice,
                            CONFIG_S1D13748 * pConfig);
```

参数	描述
pDevice	指向驱动器件的指针。
pConfig	指向某种 CONFIG_S1D13748 结构的指针，说明见下文。

CONFIG_S1D13748 的元素

数据类型	元素	描述
U32	BufferOffset	待定
int	UseLayer	待定

GUIDRV_S1D13748_SetBus_16()

描述

告知驱动使用 16 位间接接口，将把指向某种 GUI_PORT_API 结构的指针传给驱动，该指针中含有待使用的硬件程序的函数指针。

原型

```
void GUIDRV_S1D13748_SetBus_16(GUI_DEVICE    * pDevice,
                               GUI_PORT_API * pHW_API);
```

参数	描述
pDevice	指向驱动器件的指针。
pHW_API	指向某种 GUI_PORT_API 结构的指针。参见下方的所需程序部分。

所需的 GUI_PORT_API 程序

元素	数据类型
pfWrite16_A0	void (*)(U16 Data)
pfWrite16_A1	void (*)(U16 Data)
pfWriteM16_A1	void (*)(U16 * pData, int NumItems)
pfRead16_A1	U16 (*)(void)
pfReadM16_A1	U16 (*)(U16 * pData, int NumItems)

其他配置开关

无。

特殊要求

该驱动需要采用固定调色板模式 GUICC_M565。驱动不支持其他调色板模式或固定调色板模式。

28.7.7 GUIDRV_S1D15G00

支持的硬件

控制器

该驱动支持爱普生 S1D15G00 控制器。

每像素位数

支持的颜色深度为 12bpp。

接口

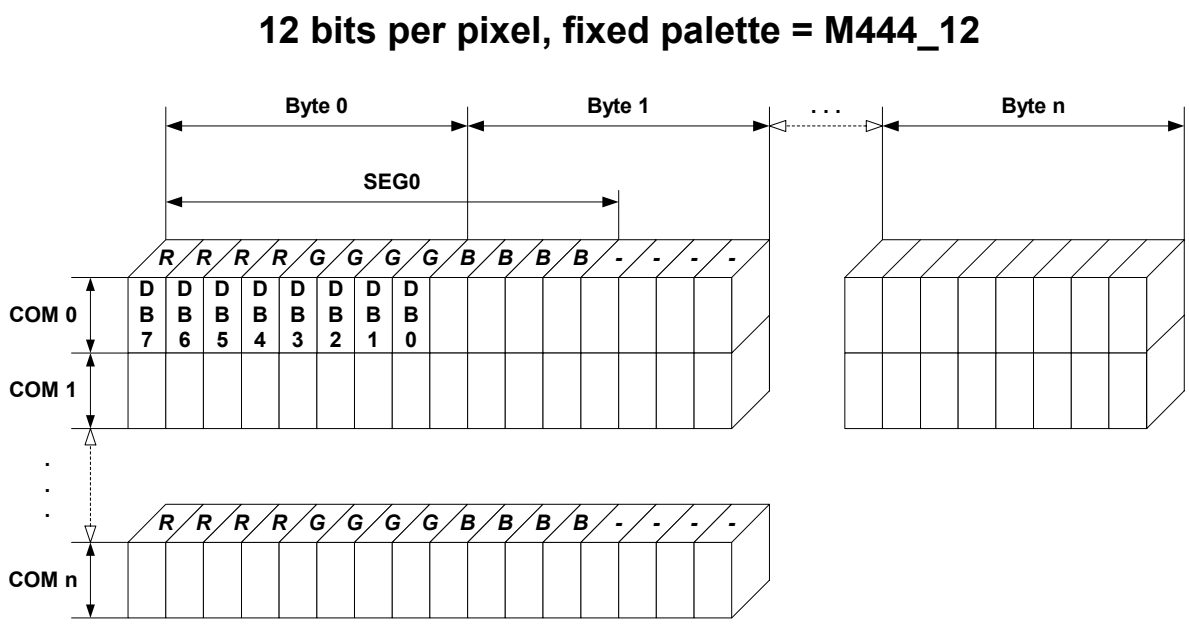
该驱动支持 8 位间接接口。

驱动选择

如果指定的显示器需要使用 GUIDRV_S1D15G00，则可使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_S1D15G00, GUICC_M444_12, 0, 0);
```

显示数据 RAM 的组织



上图展示了显示存储器与 LCD SEG 和 COM 线之间的关系。

RAM 要求

该 LCD 驱动可以使用显示数据缓存，也可以不使用。该缓存中含有 LCD 数据 RAM 中内容的完整拷贝。缓存所用存储器大小为：

LCD_XSIZE x LCD_YSIZE x 2 字节

仅当有大量绘图操作需要使用 XOR 绘图模式时，才建议使用缓存。这种情况下，使用缓存可以避免读取显示数据。正常情况下，不建议使用缓存。

其他运行时间配置

下表列出了该驱动的可用运行时间配置程序：

程序	说明
GUIDRV_S1D15G00_Config	将一个指向某种 CONFIG_S1D15G00 结构的指针传递给驱动。
GUIDRV_S1D15G00_SetBus8	告知驱动使用 8 位间接接口，将把指向某种 GUI_PORT_API 结构的指针传给驱动。

GUIDRV_S1D15G00_Config()

描述

将一个指向某种 CONFIG_S1D15G00 结构的指针传递给驱动。

原型

```
void GUIDRV_S1D15G00_Config(GUI_DEVICE       * pDevice,
                            CONFIG_S1D15G00 * pConfig);
```

参数	描述
pDevice	指向驱动器件的指针。
pConfig	指向某种 CONFIG_S1D15G00 结构的指针，说明见下文。

CONFIG_S1D15G00 的元素

数据类型	元素	描述
int	FirstSEG	显示控制器数据 RAM 使用的第一个段地址。其值可以通过实验决定，也可参考显示器文档。该值通常为 0。
int	FirstCOM	显示控制器数据 RAM 使用的第一个共用地址。其值可以通过实验决定，也可参考显示器文档。该值通常为 0。
int	UseCache	启用或禁用数据缓存功能。启用设为 1，禁用则设为 0。

GUIDRV_S1D15G00_SetBus8()

描述

告知驱动使用 8 位间接接口，将把指向某种 GUI_PORT_API 结构的指针传给驱动，该指针中含有待使用的硬件程序的函数指针。

原型

```
void GUIDRV_S1D15G00_SetBus8(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

参数	描述
pDevice	指向驱动器件的指针。
pHW_API	指向某种 GUI_PORT_API 结构的指针。参见下方的所需程序部分。

所需的 GUI_PORT_API 程序

元素	数据类型
pfWrite8_A0	void (*)(U8 Data)
pfWrite8_A1	void (*)(U8 Data)
pfWriteM8_A1	void (*)(U8 * pData, int NumItems)
pfRead8_A1	U8 (*)(void)

配置示例

```
#define XSIZE 130
#define YSIZE 130

GUI_PORT_API _PortAPI;

void LCD_X_Config(void) {
  GUI_DEVICE * pDevice;
  CONFIG_S1D15G00 Config = {0};

  //
  // Set display driver and color conversion for 1st layer
  //
  pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_S1D15G00, GUICC_M444_12, 0, 0);
  //
  // Display driver configuration, required for Lin-driver
  //
  LCD_SetSizeEx (0, XSIZE,  YSIZE);
  LCD_SetVSizeEx(0, XSIZE,  YSIZE);
  //
  // Driver specific configuration
  //
  Config.FirstCOM  = 2;
  GUIDRV_S1D15G00_Config(pDevice, &Config);
  //
  // Setup hardware access routines
  //
  _PortAPI.pfWrite8_A0  = _Write_A0;
  _PortAPI.pfWrite8_A1  = _Write_A1;
  _PortAPI.pfWriteM8_A1 = _WriteM_A1;
  GUIDRV_S1D15G00_SetBus8(pDevice, &_PortAPI);
}
```

28.7.8 GUIDRV_SLin

支持的硬件

控制器

该驱动支持以下显示控制器：

- 爱普生 S1D13700
- 所罗门 SSD1848
- 晶宏 UC1617
- 东芝 T6963

每像素位数

支持的颜色深度为 1bpp 和 2bpp。需要注意的是，东芝 Toshiba T6963 控制器只支持 1bpp 模式。

接口

该驱动支持 8 位间接接口。

颜色深度和显示方向

该驱动可以支持不同的方向和颜色深度。下表列出了在初始化过程中用来创建和链接驱动的配置宏。

标识符	颜色深度和显示方向
GUIDRV_SLIN_1	1bpp，默认方向
GUIDRV_SLIN_OY_1	1bpp， Y 轴镜像
GUIDRV_SLIN_OX_1	1bpp， X 轴镜像
GUIDRV_SLIN_OXY_1	1bpp， X 轴和 Y 轴镜像
GUIDRV_SLIN_OS_1	1bpp， X 轴和 Y 轴交换
GUIDRV_SLIN_OSY_1	1bpp， X 轴和 Y 轴交换， Y 轴镜像
GUIDRV_SLIN_OSX_1	1bpp， X 轴和 Y 轴交换， X 轴镜像
GUIDRV_SLIN_OSXY_1	1bpp， X 轴和 Y 轴交换， X 轴和 Y 轴镜像
GUIDRV_SLIN_2	2bpp，默认方向
GUIDRV_SLIN_OY_2	2bpp， Y 轴镜像
GUIDRV_SLIN_OX_2	2bpp， X 轴镜像
GUIDRV_SLIN_OXY_2	2bpp， X 轴镜像， Y 轴镜像
GUIDRV_SLIN_OS_2	2bpp， X 轴和 Y 轴交换
GUIDRV_SLIN_OSY_2	2bpp， X 轴和 Y 轴交换， Y 轴镜像
GUIDRV_SLIN_OSX_2	2bpp， X 轴和 Y 轴交换， X 轴镜像
GUIDRV_SLIN_OSXY_2	2bpp， X 轴和 Y 轴交换， Y 轴和 X 轴镜像

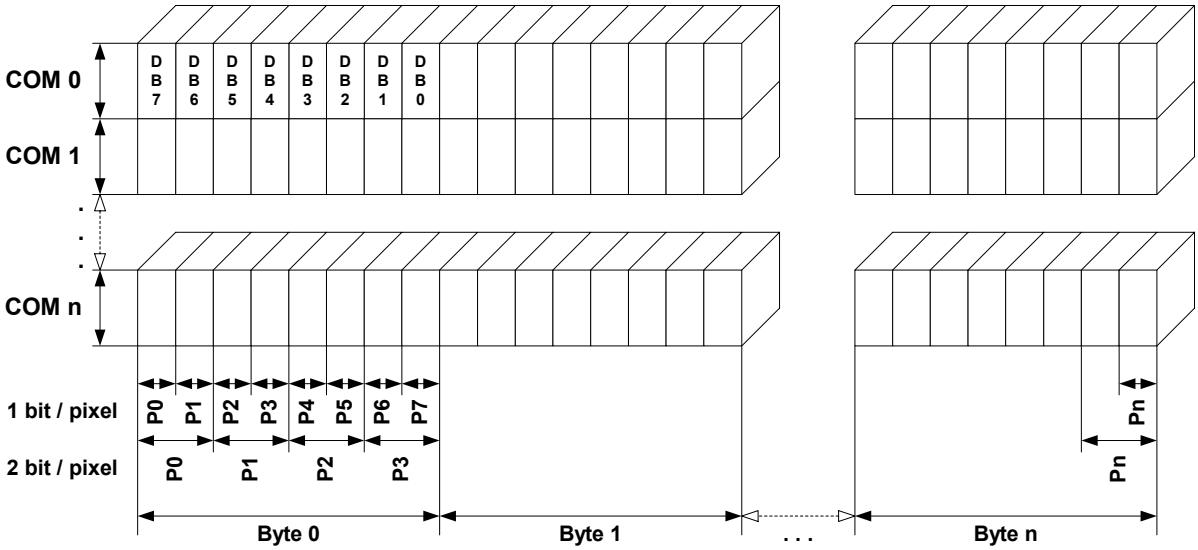
驱动选择

举例来说，如果指定的显示器需要使用 GUIDRV_SLin，则可使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_SLIN_OX_1, GUICC_1, 0, 0);
```

请参见 “颜色”（第 227 页），以了解有关利用正确的调色板模式的更多信息。

显示数据 RAM 的组织



上图展示了显示存储器与显示 SEG 和 COM 线之间的关系。

RAM 要求

该显示驱动可以使用显示数据缓存，也可以不使用。该缓存中含有 LCD 数据 RAM 的完整拷贝。如果不使用缓存，则无需增加 RAM。
建议配合该驱动使用一个数据缓存，以提高 LCD 存取速度。缓存所用存储器的大小可以通过以下等式计算：

Size of RAM (in bytes) = BitsPerPixel * (LCD_XSIZE + 7) / 8 * LCD_YSIZE

其他运行时间配置

下表列出了该驱动的可用运行时间配置程序：

程序	说明
GUIDRV_SLin_Config	将一个指向某种 CONFIG_SLIN 结构的指针传递给驱动。
GUIDRV_SLin_SetBus8	告知驱动使用 8 位间接接口，将把指向某种 GUI_PORT_API 结构的指针传给驱动。
GUIDRV_SLin_SetS1D13700	告知驱动使用爱普生 S1D13700 控制器。
GUIDRV_SLin_SetSSD1848	告知驱动使用所罗门 SSD1848 控制器。
GUIDRV_SLin_SetT6963	告知驱动使用东芝 T6963 控制器。
GUIDRV_SLin_SetUC1617	告知驱动使用晶宏 UC1617 控制器。

GUIDRV_SLin_Config()

描述

将一个指向某种 CONFIG_SLIN 结构的指针传递给驱动。

原型

```
void GUIDRV_SLin_Config(GUI_DEVICE * pDevice, CONFIG_SLIN * pConfig);
```

参数	描述
pDevice	指向驱动器件的指针。
pConfig	指向某种 CONFIG_SLIN 结构的指针，说明见下文。

CONFIG_SLIN 的元素

数据类型	元素	描述
int	FirstSEG	显示控制器数据 RAM 使用的第一个段地址。其值可以通过实验决定，也可参考显示器文档。该值通常为 0。
int	FirstCOM	显示控制器数据 RAM 使用的第一个共用地址。其值可以通过实验决定，也可参考显示器文档。该值通常为 0。
int	UseCache	启用或禁用数据缓存功能。启用设为 1，禁用则设为 0。
int	UseMirror	仅供 SSD1848 使用。一般为 1。

GUIDRV_SLin_SetBus8()

描述

告知驱动使用 16 位间接接口，将把指向某种 GUI_PORT_API 结构的指针传给驱动，该指针中含有待使用的硬件程序的函数指针。

原型

```
void GUIDRV_Slin_SetBus8(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

参数	描述
pDevice	指向驱动器件的指针
pHW_API	指向某种 GUI_PORT_API 结构的指针。参见下方的所需程序部分

所需的 GUI_PORT_API 程序

元素	数据类型
pfWrite8_A0	void (*)(U8 Data)
pfWrite8_A1	void (*)(U8 Data)
pfWriteM8_A0	void (*)(U8 * pData, int NumItems)
pfWriteM8_A1	void (*)(U8 * pData, int NumItems)
pfRead8_A1	U8 (*)(void)

GUIDRV_SLin_SetS1D13700()

描述

告知驱动应使用爱普生 S1D13700 控制器。

原型

```
void GUIDRV_SLin_SetS1D13700(GUI_DEVICE * pDevice);
```

参数	描述
pDevice	指向驱动器件的指针。

GUIDRV_SLin_SetSSD1848()

描述

告知驱动应使用所罗门 SSD1848 控制器。

原型

```
void GUIDRV_SLin_SetSSD1848(GUI_DEVICE * pDevice);
```

参数	描述
pDevice	指向驱动器件的指针。

GUIDRV_SLin_SetT6963()

描述

告知驱动应使用东芝 T6963 控制器。

原型

```
void GUIDRV_SLin_SetT6963(GUI_DEVICE * pDevice);
```

参数	描述
pDevice	指向驱动器件的指针。

GUIDRV_SLin_SetUC1617()

描述

告知驱动应使用晶宏 UC1617 控制器。

原型

```
void GUIDRV_SLin_SetUC1617(GUI_DEVICE * pDevice);
```

参数	描述
pDevice	指向驱动器件的指针。

配置示例

```
#define XSIZE 320
#define YSIZE 240

void LCD_X_Config(void) {
  GUI_DEVICE * pDevice;
  CONFIG_SLIN  Config  = {0};
  GUI_PORT_API PortAPI = {0};

  //
  // Set display driver and color conversion
  //
  pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_SLIN_2, GUICC_2, 0, 0);
  //
  // Common display driver configuration
  //
  LCD_SetSizeEx (0, XSIZE, YSIZE);
  LCD_SetVSizeEx(0, XSIZE, YSIZE);
  //
  // Driver specific configuration
  //
  Config.UseCache  = 1;
  GUIDRV_SLin_Config(pDevice, &Config);
  //
  // Select display controller
  //
  GUIDRV_SLin_SetS1D13700(pDevice);
  //
  // Setup hardware access routines
  //
  PortAPI.pfWrite16_A0  = _Write0;
  PortAPI.pfWrite16_A1  = _Write1;
  PortAPI.pfWriteM16_A0 = _WriteM0;
  PortAPI.pfRead16_A1   = _Read1;
  GUIDRV_SLin_SetBus8(pDevice, &PortAPI);
}
```

28.7.9 GUIDRV_SPage

支持的硬件

控制器

该驱动支持以下显示控制器：

- 晶宏 UC1611

每像素位数

该驱动目前支持 2bpp 和 4bpp 分辨率。

接口

该驱动支持显示控制器的 8 位间接接口。可以使用并行 4 引脚 SPI 或 I2C 总线。

颜色深度和显示方向

该驱动可以支持不同的方向和颜色深度。下表列出了在初始化过程中用来创建和链接驱动的配置宏。

标识符	颜色深度	缓存	Orientation
GUIDRV_SPAGE_2C0	2bpp	无	默认
GUIDRV_SPAGE_OY_2C0	2bpp	无	Y 轴镜像
GUIDRV_SPAGE_OX_2C0	2bpp	无	X 轴镜像
GUIDRV_SPAGE_OXY_2C0	2bpp	无	X 轴和 Y 轴镜像
GUIDRV_SPAGE_OS_2C0	2bpp	无	X 轴和 Y 轴交换
GUIDRV_SPAGE_OSY_2C0	2bpp	无	X 轴和 Y 轴交换， Y 轴镜像
GUIDRV_SPAGE_OSX_2C0	2bpp	无	X 轴和 Y 轴交换， X 轴镜像
GUIDRV_SPAGE_OSXY_2C0	2bpp	无	X 轴和 Y 轴交换， X 轴和 Y 轴镜像
GUIDRV_SPAGE_2C1	2bpp	有	默认
GUIDRV_SPAGE_OY_2C1	2bpp	有	Y 轴镜像
GUIDRV_SPAGE_OX_2C1	2bpp	有	X 轴镜像
GUIDRV_SPAGE_OXY_2C1	2bpp	有	X 轴和 Y 轴镜像
GUIDRV_SPAGE_OS_2C1	2bpp	有	X 轴和 Y 轴交换
GUIDRV_SPAGE_OSY_2C1	2bpp	有	X 轴和 Y 轴交换， Y 轴镜像
GUIDRV_SPAGE_OSX_2C1	2bpp	有	X 轴和 Y 轴交换， X 轴镜像
GUIDRV_SPAGE_OSXY_2C1	2bpp	有	X 轴和 Y 轴交换， X 轴和 Y 轴镜像
GUIDRV_SPAGE_4C0	4bpp	无	默认
GUIDRV_SPAGE_OY_4C0	4bpp	无	Y 轴镜像
GUIDRV_SPAGE_OX_4C0	4bpp	无	X 轴镜像
GUIDRV_SPAGE_OXY_4C0	4bpp	无	X 轴和 Y 轴镜像
GUIDRV_SPAGE_OS_4C0	4bpp	无	X 轴和 Y 轴交换
GUIDRV_SPAGE_OSY_4C0	4bpp	无	X 轴和 Y 轴交换， Y 轴镜像
GUIDRV_SPAGE_OSX_4C0	4bpp	无	X 轴和 Y 轴交换， X 轴镜像
GUIDRV_SPAGE_OSXY_4C0	4bpp	无	X 轴和 Y 轴交换， X 轴和 Y 轴镜像
GUIDRV_SPAGE_4C1	4bpp	有	默认
GUIDRV_SPAGE_OY_4C1	4bpp	有	Y 轴镜像
GUIDRV_SPAGE_OX_4C1	4bpp	有	X 轴镜像
GUIDRV_SPAGE_OXY_4C1	4bpp	有	X 轴和 Y 轴镜像
GUIDRV_SPAGE_OS_4C1	4bpp	有	X 轴和 Y 轴交换
GUIDRV_SPAGE_OSY_4C1	4bpp	有	X 轴和 Y 轴交换， Y 轴镜像
GUIDRV_SPAGE_OSX_4C1	4bpp	有	X 轴和 Y 轴交换， X 轴镜像
GUIDRV_SPAGE_OSXY_4C1	4bpp	有	X 轴和 Y 轴交换， X 轴和 Y 轴镜像

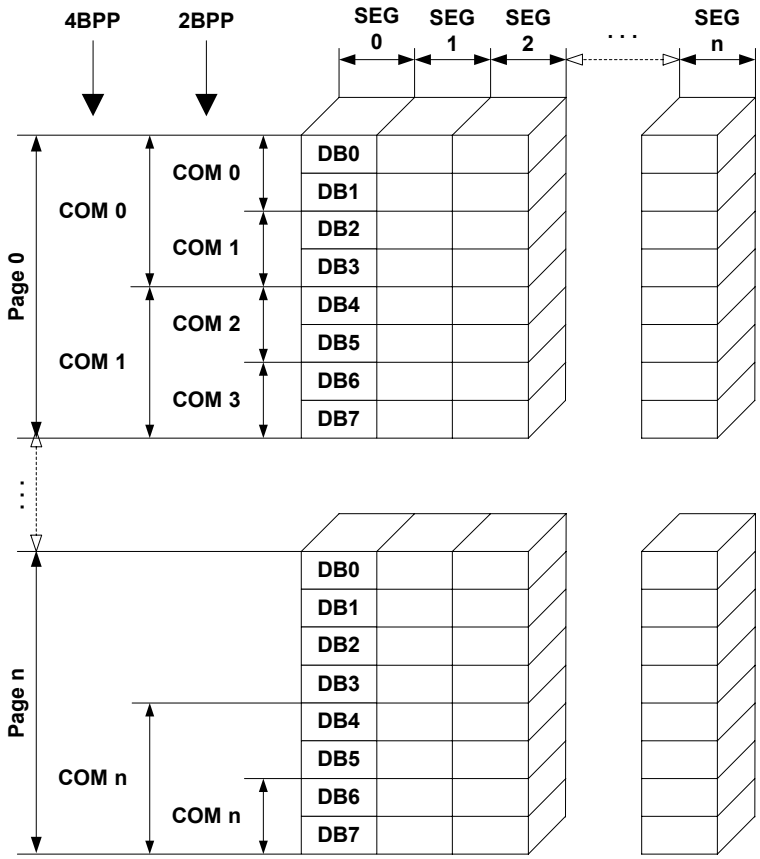
驱动选择

如果给定的显示器需要使用 GUIDRV_SPage，则函数 LCD_X_Config 可使用以下调用：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_SPAGE_4C0, GUICC_4, 0, 0);
```


请参见 “颜色”（第 227 页）一章，以了解有关利用正确的调色板模式的更多信息。

显示数据 RAM 的组织



上图展示了显示存储器与显示 SEG 和 COM 线之间的关系。

RAM 要求

该显示驱动可以使用显示数据缓存，也可以不使用。数据缓存含有 LCD 数据 RAM 的完整拷贝。如果不使用缓存，则无需增加 RAM。
强烈建议配合该驱动使用一个数据缓存，以提高 LCD 存取速度。不使用缓存会大幅降低该驱动的性能。缓存所用存储器的大小可以通过以下等式计算：

```
RAM 大小 （单位：字节） =
(LCD_YSIZE + (8 / LCD_BITSPERPIXEL - 1)) / 8 * LCD_BITSPERPIXEL * LCD_XSIZE
```

其他运行时间配置

下表列出了该驱动的可用运行时间配置程序：

程序	说明
GUIDRV_SPage_Config	传递一个指向某种 CONFIG_SPAGE 结构的指针
GUIDRV_SPage_SetBus8	告知驱动使用8位间接接口，将把指向某种GUI_PORT_API结构的指针传给驱动
GUIDRV_SPage_SetUC1611	告知驱动使用晶宏 UC1611 控制器

GUIDRV_SPage_Config()

描述

将一个指向某种 CONFIG_SPAGE 结构的指针传递给驱动。

原型

```
void GUIDRV_SPage_Config(GUI_DEVICE * pDevice, CONFIG_SPAGE * pConfig);
```

参数	描述
pDevice	指向驱动器件的指针。
pConfig	指向某种 CONFIG_SPAGE 结构的指针，说明见下文。

CONFIG_SPAGE 的元素

数据类型	元素	描述
int	FirstSEG	显示控制器数据 RAM 使用的第一个段地址。其值可以通过实验决定，也可参考显示器文档。该值通常为 0。
int	FirstCOM	显示控制器数据 RAM 使用的第一个共用地址。其值可以通过实验决定，也可参考显示器文档。该值通常为 0。

GUIDRV_SPage_SetBus8()

描述

告知驱动使用 8 位间接接口，将把指向某种 GUI_PORT_API 结构的指针传给驱动，该指针中含有待使用的硬件程序的函数指针。

原型

```
void GUIDRV_SPage_SetBus8(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

参数	描述
pDevice	指向驱动器件的指针。
pHW_API	指向某种 GUI_PORT_API 结构的指针。参见下方的所需程序部分。

所需的 GUI_PORT_API 程序

元素	数据类型
pfWrite8_A0	void (*)(U8 Data)
pfWrite8_A1	void (*)(U8 Data)
pfWriteM8_A1	void (*)(U8 * pData, int NumItems)
pfRead8_A1	U8 (*)(void)

配置示例

```
void LCD_X_Config(void) {
  CONFIG_SPAGE Config = {0};
  GUI_DEVICE * pDevice;
  GUI_PORT_API PortAPI = {0};

  //
  // Set display driver and color conversion for 1st layer
  //
  pDevice = GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER, COLOR_CONVERSION, 0, 0);
  //
  // Display size configuration
  //
  if (LCD_GetSwapXY()) {
    LCD_SetSizeEx (0, YSIZE_PHYS,   XSIZE_PHYS);
    LCD_SetVSizeEx(0, VYSIZE_PHYS,  VXSIZE_PHYS);
  } else {
    LCD_SetSizeEx (0, XSIZE_PHYS,   YSIZE_PHYS);
    LCD_SetVSizeEx(0, VXSIZE_PHYS,  VYSIZE_PHYS);
  }
  //
  // Driver configuration
  //
  Config.FirstSEG = 0;//256 - 224;
  GUIDRV_SPage_Config(pDevice, &Config);
  //
  // Configure hardware routines
  //
  PortAPI.pfWrite8_A0  = _Write8_A0;
  PortAPI.pfWrite8_A1  = _Write8_A1;
  PortAPI.pfWriteM8_A1 = _WriteM8_A1;
  PortAPI.pfReadM8_A1  = LCD_X_8080_8_ReadM01;
  GUIDRV_SPage_SetBus8(pDevice, &PortAPI);
  //
  // Controller configuration
  //
  GUIDRV_SPage_SetUC1611(pDevice);
}
```

GUIDRV_SPage_SetUC1611()

描述

告知驱动应当使用晶宏 UC1611 控制器。

原型

```
void GUIDRV_SPage_SetUC1611(GUI_DEVICE * pDevice);
```

参数	描述
pDevice	指向驱动器件的指针。

28.7.10 GUIDRV_SSD1926

支持的硬件

控制器

该驱动支持所罗门 SSD1926 显示控制器。

每像素位数

支持的颜色深度为 8bpp。该显示控制器最高可支持 32bpp。如果需要支持另一种颜色深度，则可根据需要扩展该驱动。

接口

该驱动支持 16 位间接接口。

颜色深度和显示方向

该驱动可以支持不同的显示方向。下表列出了在初始化过程中用来创建和链接驱动的配置宏。

标识符	颜色深度和显示方向
GUIDRV_SSD1926_8	8bpp，默认方向
GUIDRV_SSD1926_OY_8	8bpp， Y 轴镜像
GUIDRV_SSD1926_OX_8	8bpp， X 轴镜像
GUIDRV_SSD1926_OXY_8	8bpp， X 轴和 Y 轴镜像
GUIDRV_SSD1926_OS_8	8bpp， X 轴和 Y 轴交换
GUIDRV_SSD1926_OSY_8	8bpp， X 轴和 Y 轴交换， Y 轴镜像
GUIDRV_SSD1926_OSX_8	8bpp， X 轴和 Y 轴交换， X 轴镜像
GUIDRV_SSD1926_OSXY_8	8bpp， X 轴和 Y 轴交换， X 轴和 Y 轴镜像

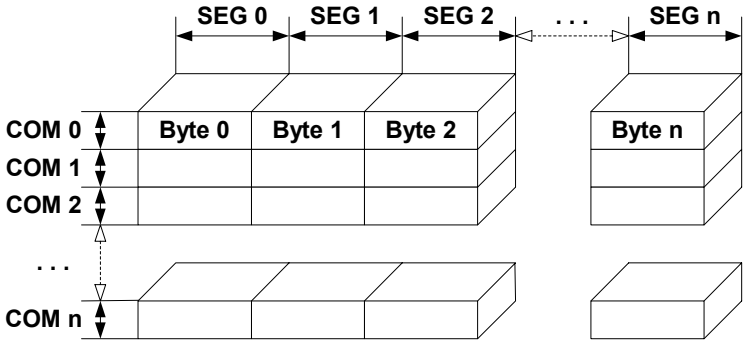
驱动选择

举例来说，如果指定的显示器需要使用 GUIDRV_SSD1926，则可使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_SSD1926, GUICC_323, 0, 0);
```

请参见 “颜色”（第 227 页）一章，以了解有关利用正确的调色板模式的更多信息。

显示数据 RAM 的组织



上图展示了显示存储器与显示 SEG 和 COM 线之间的关系。

RAM 要求

该显示驱动可以使用显示数据缓存，也可以不使用。该缓存中含有 LCD 数据 RAM 的完整拷贝。如果不使用缓存，则无需增加 RAM。
建议配合该驱动使用一个数据缓存，以提高 LCD 存取速度。缓存所用存储器的大小可以通过以下等式计算：

RAM 大小 （单位：字节） = LCD_XSIZE * LCD_YSIZE

其他运行时间配置

下表列出了该驱动的可用运行时间配置程序：

程序	说明
GUIDRV_SSD1926_Config	将一个指向某种 CONFIG_SSD1926 结构的指针传递给驱动。
GUIDRV_SSD1926_SetBus16	告知驱动使用 16 位间接接口，将把指向某种 GUI_PORT_API 结构的指针传给驱动。

GUIDRV_SSD1926_Config()

描述

将一个指向某种 CONFIG_SSD1926 结构的指针传递给驱动。

原型

```
void GUIDRV_SSD1926_Config(GUI_DEVICE * pDevice, CONFIG_SSD1926 * pConfig);
```

参数	描述
pDevice	指向驱动器件的指针。
pConfig	指向某种 CONFIG_SSD1926 结构的指针，说明见下文。

CONFIG_SSD1926 的元素

数据类型	元素	描述
int	FirstSEG	显示控制器数据 RAM 使用的第一个段地址。其值可以通过实验决定，也可参考显示器文档。该值通常为 0。
int	FirstCOM	显示控制器数据 RAM 使用的第一个共用地址。其值可以通过实验决定，也可参考显示器文档。该值通常为 0。
int	UseCache	启用或禁用数据缓存功能。启用设为 1，禁用则设为 0。

GUIDRV_SSD1926_SetBus16()

描述

告知驱动使用 16 位间接接口，将把指向某种 GUI_PORT_API 结构的指针传给驱动，该指针中含有待使用的硬件程序的函数指针。

原型

```
void GUIDRV_SSD1926_SetBus16(GUI_DEVICE * pDevice, GUI_PORT_API * pHW_API);
```

参数	描述
pDevice	指向驱动器件的指针。
pHW_API	指向某种 GUI_PORT_API 结构的指针。参见下方的所需程序部分。

所需的 GUI_PORT_API 程序

元素	数据类型
pfWrite16_A0	void (*)(U16 Data)
pfWrite16_A1	void (*)(U16 Data)
pfWriteM16_A0	void (*)(U16 * pData, int NumItems)
pfWriteM16_A1	void (*)(U16 * pData, int NumItems)
pfRead16_A1	U16 (*)(void)

配置示例

```
#define XSIZE 320L
#define YSIZE 240L

GUI_PORT_API _PortAPI;

void LCD_X_Config(void) {
  GUI_DEVICE * pDevice_0;
  CONFIG_SSD1926 Config_0 = {0};

  //
  // Set display driver and color conversion
  //
  pDevice_0 = GUI_DEVICE_CreateAndLink(GUIDRV_SSD1926_8, GUICC_8666, 0, 0);
  //
  // Common display driver configuration
  //
  LCD_SetSizeEx (0, XSIZE, YSIZE);
  LCD_SetVSizeEx(0, XSIZE, YSIZE);
  //
  // Set driver specific configuration items
  //
  Config_0.UseCache  = 1;
  //
  // Set hardware access routines
  //
  _PortAPI.pfWrite16_A0  = LCD_X_8080_16_Write00_16;
  _PortAPI.pfWrite16_A1  = LCD_X_8080_16_Write01_16;
  _PortAPI.pfWriteM16_A0 = LCD_X_8080_16_WriteM00_16;
  _PortAPI.pfWriteM16_A1 = LCD_X_8080_16_WriteM01_16;
  _PortAPI.pfRead16_A1   = LCD_X_8080_16_Read01_16;
  GUIDRV_SSD1926_SetBus16(pDevice, &_PortAPI);
  //
  // Pass configuration structure to driver
  //
  GUIDRV_SSD1926_Config(pDevice, &Config_0);
}
```


28.7.11 GUIDRV_CompactColor_16

支持的硬件

控制器

该驱动支持以下显示控制器：

- 晶采 FSA506
- 爱普生 S1D13742、 S1D13743、 S1D19122
- 奇景 HX8301、HX8312A、HX8325A、HX8340、HX8347、HX8352、HX8352B、HX8353
- 日立 HD66766、 HD66772、 HD66789
- Ilitek ILI9161、 ILI9220、 ILI9221、 ILI9320、 ILI9325、 ILI9326、 ILI9328
- LG 电子 LGDP4531、 LGDP4551
- 美格纳 D54E4PA7551
- 联咏 NT39122、 NT7573
- OriseTech SPFD5408、 SPFD54124C、 SPFD5414D、 SPFD5420A
- 瑞萨 R61505、 R61509、 R61516、 R61580、 R63401
- 三星 S6D0110A、 S6D0117、 S6D0129、 S6D04H0
- 夏普 LCY-A06003、 LR38825
- 矽创 ST7628、 ST7637、 ST7687、 ST7712、 ST7715、 ST7735、 ST7787
- 所罗门 SSD1284、SSD1289、SSD1298、SSD1355、SSD1961、SSD1963、SSD2119
- 东芝 JBT6K71

每像素位数

支持的颜色深度为 16bpp。

接口

该驱动支持间接接口 （8 位和 16 位）和 3 引脚 SPI 接口。默认为 8 位间接模式。

驱动选择和配置

如果要使用该驱动，则需要将以下宏定义添加至配置文件 LCDConf.h 中：

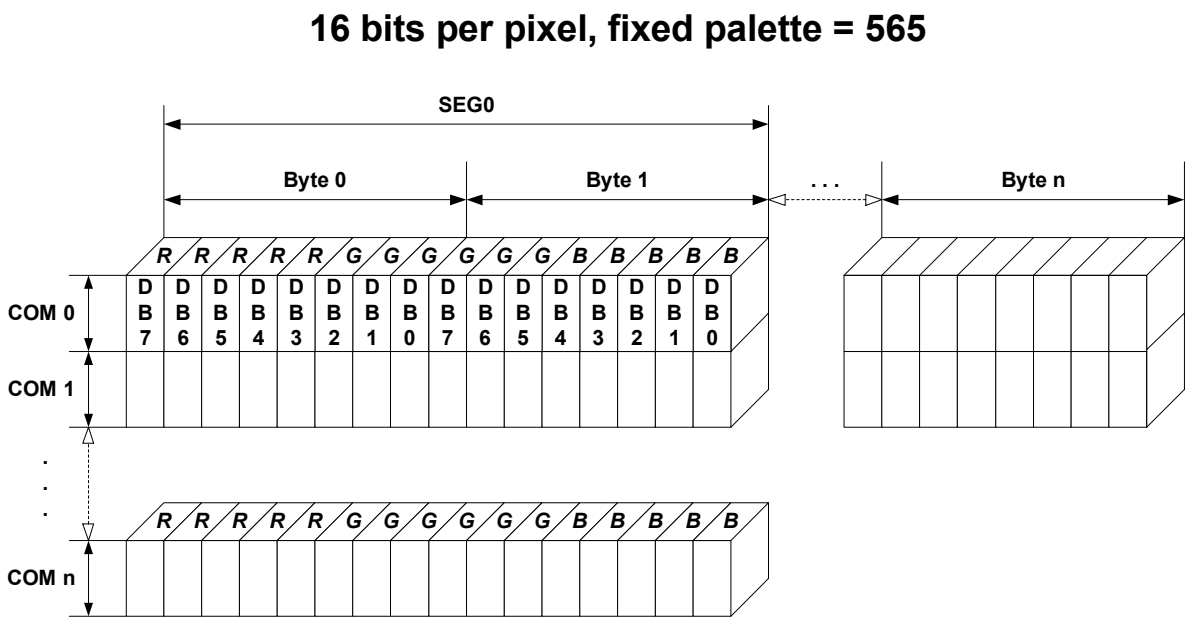
```
#define LCD_USE_COMPACT_COLOR_16
```

添加该定义语句后，显示驱动将采用配置文件夹中与驱动对应的配置文件 LCDConf_CompactColor_16.h。所有其他编译时配置宏都应在该文件中定义。如果要使用 GUIDRV_CompactColor_16 为给定的显示器创建驱动，则可以使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_COMPACT_COLOR_16,
                                   GUICC_565, 0, 0);
```

请参见 “颜色”（第 227 页）一章，以了解有关利用正确的调色板模式的更多信息。

显示数据 RAM 的组织



上图展示了显示存储器与 LCD SEG 和 COM 线之间的关系。

RAM 要求

该 LCD 驱动可以使用显示数据缓存，也可以不使用。该缓存中含有 LCD 数据 RAM 中内容的完整拷贝。缓存所用存储器大小为：LCD_XSIZE * LCD_YSIZE * 2 字节。仅当有大量绘图操作要使用 XOR 绘图模式时，才建议使用缓存。这种情况下，使用缓存可以避免读取显示数据。正常情况下，不建议使用缓存。

该驱动使用一个写缓冲器来绘制同一种颜色的多个像素。如果需要绘制同一种颜色的多个像素，则该驱动将首先填充缓冲器，然后执行对 LCD_WRITEM_A1 宏的单次调用，以将数据立即传递给显示控制器。默认的缓冲器大小为 500 字节。

可用的配置宏 （编译时间配置）

控制器选择

如果要选择所需的控制器，则应当在配置文件LCDConf_CompactColor_16.h中使用LCD_CONTROLLER宏。下表列出了用于选择正确的控制器的值：

编号	支持的控制器
66700	夏普 LR38825
66701	Ilitek ILI9326 OriseTech SPFD5420A 瑞萨 R61509、 R63401
66702	所罗门 SSD1284、 SSD1289、 SSD1298
66703	东芝 JBT6K71
66704	夏普 LCY-A06003
66705	瑞萨 R61505 三星 S6D0129
66706	美格纳 D54E4PA7551
66707	奇景 HX8312
66708	Ilitek ILI9320、 ILI9325、 ILI9328 LG 电子 LGDP4531、 LGDP4551 OriseTech SPFD5408 瑞萨 R61505、 R61580

编号	支持的控制器
66709	爱普生 S1D19122 奇景 HX8353 联咏 NT39122 Orisetech SPFD54124C、SPFD5414D 瑞萨 R61516 三星 S6D04H0 矽创 ST7628、ST7637、ST7715、ST7735 所罗门 SSD1355、SSD1961、SSD1963
66710	联咏 NT7573
66711	爱普生 S1D13742、S1D13743
66712	奇景 HX8347、HX8352
66713	奇景 HX8340
66714	所罗门 SSD2119
66715	奇景 HX8352B
66716	晶采 FSA506
66717	矽创 ST7787
66766	日立 HD66766 Ilitec ILI9161 三星 S6D0110A
66772	奇景 HX8301 日立 HD66772 Ilitec ILI9220、ILI9221 三星 S6D0117 矽创 ST7712
66789	日立 HD66789

显示配置

下表列出了可用的配置宏：

宏	描述
LCD_MIRROR_X	激活 X 轴镜像。
LCD_MIRROR_Y	激活 Y 轴镜像。
LCD_SWAP_XY	激活 X 轴和 Y 轴交换。

更多详细信息，请参阅 “显示方向”（第 824 页）。

硬件存取

下表列出了可用的配置宏，这些宏可在该文件中进行定义，以便配置硬件的存取方式：

宏	描述
LCD_NUM_DUMMY_READS	在需要执行读操作时要求的虚拟读取次数。默认值为 2。如果使用串行接口，则 HD66766 和 HD66772 显示控制器需要 5 次虚拟读操作。夏普 LR38825 在使用 8 位总线时，需要 3 次虚拟读取操作。
LCD_REG01	该宏仅当在采用奇景 HX8312A 时才需要使用。遗憾的是，寄存器 0x01（控制寄存器 1）同时含有因方向而异的设置和共用设置。因此，该宏应含有该寄存器的内容。
LCD_SERIAL_ID	采用一个串行 3 线接口，该宏定义器件 ID 代码的 ID 信号。应为 0（默认值）或 1。需要注意的是：该宏仅用于日立 HD66772、三星 S6D0117、奇景 HX8301 和 Ilitek ILI9220 的 3 线协议。
LCD_USE_SERIAL_3PIN	该配置宏是为了支持下列控制器的 3 线串行接口：日立 HD66772、三星 S6D0117、奇景 HX8301 和 Ilitek ILI9220。如果使用 3 线串行接口，应设为 1。默认值为 0。需要注意的是：其他显示控制器不要使用该宏！
LCD_USE_PARALLEL_16	如果使用 16 位并行接口，应设为 1。默认值为 0。
LCD_WRITE_BUFFER_SIZE	定义写缓冲器的大小。使用写缓冲器可以提高驱动的性能。如果要写入同一种颜色的多个像素，则驱动首先填充缓冲器，然后执行一次 LCD_WRITEM_A1 宏（而不是多次），写入缓冲器的内容。默认的缓冲器大小为 500 字节。
LCD_WRITE_A0	向显示控制器写入一个字节，其中， RS 线为低电平。
LCD_WRITE_A1	向显示控制器写入一个字节，其中， RS 线为高电平。

宏	描述
LCD_READM_A1	从显示控制器读取多个字节 （8 位并行接口）或多个字 （16 位并行接口），其中，RS 线为高电平。
LCD_WRITEM_A1	把多个字节 （8 位并行接口）或多个字 （16 位并行接口）写入显示控制器，其中，RS 线为高电平。
LCD_WRITEM_A0	把多个字节 （8 位并行接口）或多个字 （16 位并行接口）写入显示控制器，其中，RS 线为低电平。

该驱动会自行初始化 “驱动输出模式”和 “输入模式”寄存器。用户无需在 LCD_X_InitController() 中初始化该寄存器。

可用的配置程序 （运行时间配置）

下表列出了可用的运行时间配置程序：

程序	描述
LCD_SetSizeEx()	更改可见区的尺寸。

配置示例

以下示例展示了驱动的选择和配置方式：

LCDConf.h

如上所述，其中应包含以下语句，以便选择驱动：

```
#define LCD_USE_COMPACT_COLOR_16
```

LCDConf_CompactColor_16.h

该文件含有与显示驱动对应的配置，以下是一种可能的情况：

```
//
// General configuration of LCD
//
#define LCD_CONTROLLER      66709 // Renesas R61516
#define LCD_BITSPERPIXEL       16
#define LCD_USE_PARALLEL_16     1
#define LCD_MIRROR_Y            1
//
// Indirect interface configuration
//
void LCD_X_Write01_16(unsigned short c);
void LCD_X_Write00_16(unsigned short c);
void LCD_X_WriteM01_16(unsigned short * pData, int NumWords);
void LCD_X_WriteM00_16(unsigned short * pData, int NumWords);
void LCD_X_ReadM01_16 (unsigned short * pData, int NumWords);
#define LCD_WRITE_A1(Word) LCD_X_Write01_16(Word)
#define LCD_WRITE_A0(Word) LCD_X_Write00_16(Word)
#define LCD_WRITEM_A1(Word, NumWords) LCD_X_WriteM01_16(Word, NumWords)
#define LCD_WRITEM_A0(Word, NumWords) LCD_X_WriteM00_16(Word, NumWords)
#define LCD_READM_A1(Word, NumWords)  LCD_X_ReadM01_16(Word, NumWords)
```

LCDConf.c

以下示例展示了如何使用该驱动来创建显示驱动器件及其配置方法：

```
void LCD_X_Config(void) {
  //
  // Set display driver and color conversion
  //
  GUI_DEVICE_CreateAndLink(GUIDRV_COMPACT_COLOR_16, // Display driver
                           GUICC_M565,              // Color conversion
                           0, 0);
  //
  // Display driver configuration
  //
  LCD_SetSizeEx(0, 240, 320);                     // Physical display size in pixels
}
```

28.7.12 GUIDRV_Fujitsu_16

该驱动支持富士通图形显示控制器。已通过“Jasmine”型号的测试，但也应该支持“Lavender”，因为所有相关寄存器都是兼容的。

支持的硬件

控制器

该驱动支持以下显示控制器：

- 富士通 Jasmine
- 富士通 Lavender

每像素位数

支持的颜色深度为 1bpp、 2bpp、 4bpp、 8bpp 和 16bpp。

接口

该驱动已通过测试，支持32位CPU接口。如果使用16位接口，则可用2次16位存取代替32位存取。

驱动选择

举例来说，如果给定的显示器需要使用 GUIDRV_Fujitsu_16，则可以使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_FUJITSU_16, GUICC_556, 0, 0);
```

请参见 “颜色”（第 227 页）一章，以了解有关利用正确的调色板模式的更多信息。

可用的配置宏 （编译时间配置）

控制器选择

如果要选择所需的控制器，则应当在配置文件 LCDConf_Fujitsu_16.h 中使用 LCD_CONTROLLER 宏。下表列出了用于选择正确的控制器的值：

编号	支持的控制器
8720	富士通 Jasmine
8721	富士通 Lavender

显示数据 RAM 的组织

显示控制器以优化、非线性的方式利用 DRAM（详见富士通文档）。该驱动不使用直接存储器存取模式。

RAM 要求

大约需要 16 个字节，用于存储静态变量。

硬件配置

该驱动要求使用全总线硬件存取接口，如第 28 章 “配置”中所描述的那样。下表列出了必须针对硬件存取进行定义的宏：

宏	说明
LCD_READ_REG	读取显示控制器的一个寄存器。（作为 32 位值）（可选）
LCD_WRITE_REG	写入显示控制器的一个寄存器。（作为 32 位值）（可选）

该驱动含有默认的硬件存取宏，在富士通演示平台上配置 32 位存取模式（使用 MB91361 或 MB91362 以及一个 Jasmine 芯片，地址为 0x30000000）；如果目标硬件兼容这些设置，则不需要定义 LCD_READ_REG()、 LCD_WRITE_REG()。

颜色格式 （R/B 交换）

似乎有些目标系统中，红色和蓝色是交换的。如果在配置文件中切换配置开关 LCD_SWAP_RB，则可通过软件来更改这种情况。

硬件初始化

该显示控制器要求较复杂的初始化。富士通在 GDC 模块中提供有示例代码。该代码并不是驱动的一部分，因为它取决于实际使用的芯片，取决于时钟设置、显示器以及多种其他因素。
我们建议使用富士通提供的原始代码，因为根据文档信息，还不足以写出该代码。在调用 GUI_Init() 之前，应通过该代码初始化 GDC （一般以 GDC_Init(0xff)) 的形式调用）。

示例：

LCDConf.h for VGA display, 8bpp, Jasmine:

```
#define LCD_XSIZE 640 /* X-resolution of LCD, Logical color.*/
#define LCD_YSIZE 480 /* Y-resolution of LCD, Logical color.*/
#define LCD_BITSPERPIXEL 8
#define LCD_CONTROLLER 8720 /* Jasmine */
```

其他配置开关

下表列出了该驱动可以使用的可选配置宏：

宏	说明
LCD_ON	函数替换宏，用于打开显示器。
LCD_OFF	函数替换宏，用于关闭显示器。

特殊要求

无

28.7.13 GUIDRV_Page1bpp

支持的硬件

控制器

该驱动支持以下显示控制器：

- 爱普生 S1D10605、S1D15605、S1D15705、S1D15710、S1D15714、S1D15721
- 爱普生 S1D15E05、S1D15E06、SED1520、SED1560、SED1565、SED1566
- 爱普生 SED1567、SED1568、SED1569、SED1575
- 日立 HD61202
- IST IST3020
- 新日本无线株式会社 NJU6676、NJU6679
- 联咏 NT7502、NT7534、NT7538、NT75451
- 飞利浦 PCF8810、PCF8811、PCF8535、PCD8544
- 三星 KS0108B、KS0713、KS0724
- 三星 S6B0108B、S6B0713、S6B0719、S6B0724、S6B1713
- 中颖 SH1101A
- 矽创 ST7522、ST7565、ST7567
- 所罗门 SSD1303、SSD1805、SSD1815
- 意法半导体 ST7548、STE2001、STE2002
- 凌阳 SPLC501C
- 晶宏 UC1601、UC1606、UC1608、UC1701

可以假定，该驱动同时也能支持每种结构类似的控制器。

每像素位数

支持的颜色深度为 1bpp。

接口

该驱动支持显示控制器的 8 位间接接口。可以使用并行 4 引脚 SPI 或 I2C 总线。

驱动选择

如果给定的显示器要使用 GUIDRV_Page1bpp，则应当使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_PAGE1BPP, GUICC_1, 0, 0);
```


可用的配置宏 （编译时间配置）

控制器选择

如果要选择所需的控制器，则应当在配置文件 LCDConf_Page1bpp.h 中使用 LCD_CONTROLLER 宏。下表列出了用于选择正确的控制器的值：

编号	支持的控制器
1501	三星 KS0713、KS0724、S6B0713、S6B0724 晶宏 UC1601、UC1606
1502	三星 KS0108B、S6B0108B
1503	日立 HD61202
1504	飞利浦 PCF8810、PCF8811
1505	飞利浦 PCF8535
1506	新日本无线株式会社 NJU6679
1507	飞利浦 PCD8544
1508	爱普生 S1D15710
1509	所罗门 SSD1303 OLED 控制器
1510	爱普生 S1D15714 IST 3020 新日本无线株式会社 NJU6676 联咏 NT7538、NT75451 三星 S6B0719 中颖 SH1101A 矽创 ST7522、ST7565、ST7567 所罗门 SSD1805 晶宏 UC1608、UC1701
1511	爱普生 S1D15721
1512	爱普生 S1D15E05、S1D15E06
1513	意法半导体 ST7548、STE2001、STE2002
1520	爱普生 SED1520
1560	爱普生 SED1560
1565	爱普生 SED1565、S1D10605、S1D15605 联咏 NT7502、NT7534 三星 S6B1713 所罗门 SSD1815 凌阳 SPLC501C
1566	爱普生 SED1566
1567	爱普生 SED1567
1568	爱普生 SED1568
1569	爱普生 SED1569
1575	爱普生 SED1575、S1D15705

RAM 要求

多数情况下，该 LCD 显示驱动可以使用显示数据缓存，也可以不使用。如果一个显示器含有一个以上的 LCD 控制器，则不能禁用缓存。数据缓存含有 LCD 数据 RAM 中内容的完整拷贝。如果不使用缓存，则无需增加 RAM。

建议配合该驱动使用一个数据缓存，以提高 LCD 存取速度。缓存所用存储器的大小可以通过以下等式计算：

RAM 大小 （单位：字节） = (LCD_YSIZE + 7) / 8 * LCD_XSIZE

其他驱动函数

LCD_ControlCache

有关此函数的信息，请参见第 891 页。

硬件配置

如第 28 章 “配置”中所描述的那样，该驱动以间接接口存取硬件。下表列出了必须针对硬件存取进行定义的宏：

宏	说明
LCD_READ_A0	从 LCD 控制器读取一个字节，其中， A 线为低电平。
LCD_READ_A1	从 LCD 控制器读取一个字节，其中， A 线为高电平。
LCD_WRITE_A0	向 LCD 控制器写入一个字节，其中， A 线为低电平。
LCD_WRITE_A1	向 LCD 控制器写入一个字节，其中， A 线为高电平。
LCD_WRITEM_A1	向 LCD 控制器写入多个字节，其中， A 线为高电平。

显示方向

支持的部分显示控制器支持硬件镜像显示 x/y 轴。建议使用这些函数而不是 emWin 的显示方向宏。如果需要镜像显示 X 轴，则应在初始化宏中使用命令 0xA1（ADC 选择反向）。结果会使显示控制器颠倒指派给段输出的栏地址。如果 X 中的显示尺寸小于显示控制器的段输出数，则可利用宏 LCD_FIRSTSEG0 来给栏地址添加一个偏移，以确保存取 LCD 控制器的正确 RAM 地址。
如果需要镜像显示Y轴，则应在初始化宏中使用命令0xC8（SHL选择反向），同时用宏LCD_FIRSTCOM0定义偏移，以访问显示控制器的正确 RAM 地址。

其他配置开关

下表列出了该驱动可以使用的可选配置开关：

宏	说明
LCD_CACHE	设为 0 时，不使用显示数据缓存，结果会减慢驱动的速度。默认值为 1 （缓存激活）。
LCD_FIRSTCOM0	该宏可以用于定义显示控制器数据 RAM 中使用的第一个共用地址。其值可以通过实验决定，也可参考显示器文档。
LCD_FIRSTSEG0	该宏可以用于定义显示控制器数据 RAM 中使用的第一个段地址。其值可以通过实验决定，也可参考显示器文档。
LCD_SUPPORT_CACHECONTROL	设为 1 时，可使用 CD_ControlCache()。

某些 LCD 控制器的特殊要求

无。

28.7.14 GUIDRV_07X1

支持的硬件

控制器

该驱动支持以下 LCD 控制器：

- 联咏 NT7506
- 三星 KS0711、KS0741、S6B0711、S6B0741
- 所罗门 SSD1854
- 矽创 ST7541、ST7571
- 意法半导体 STE2010
- Tomato TL0350A

每像素位数

支持的颜色深度为 2bpp。

接口

该芯片支持 8 位并行 （简单总线）接口和 4 引脚或 3 引脚串行外设接口 (SPI)。驱动的最新版支持 8 位并行 （简单总线）接口或 4 引脚 SPI 接口。

驱动选择

举例来说，若要针对给定的显示器使用 GUIDRV_07X1，可用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_07X1, GUICC_2, 0, 0);
```

请参见 “颜色”（第 227 页）一章，以了解有关利用正确的调色板模式的更多信息。

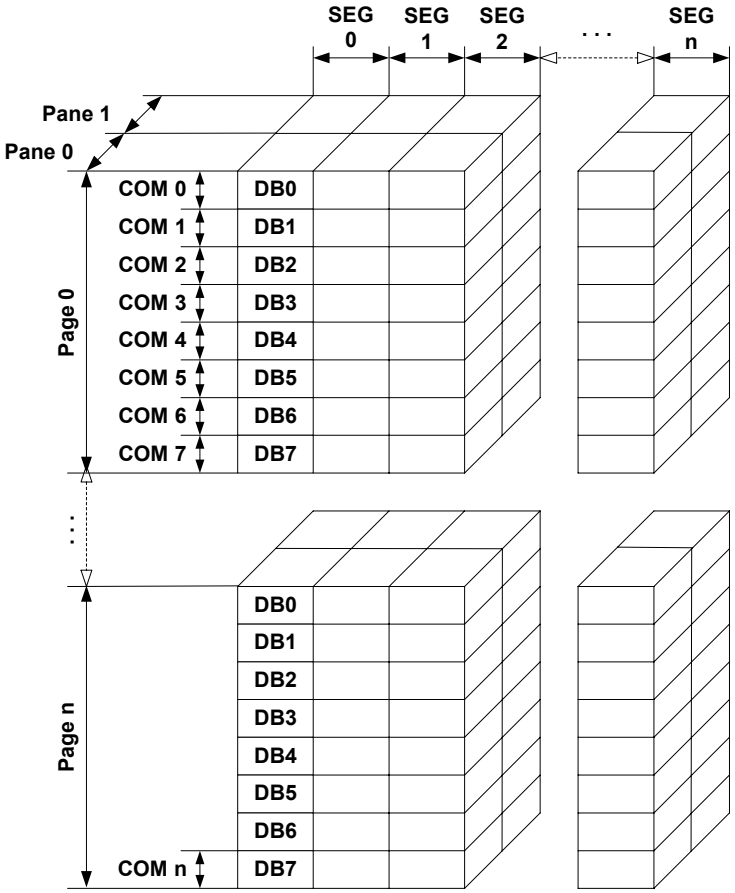
可用的配置宏 （编译时间配置）

控制器选择

如果要选择所需的控制器，则应当在配置文件 LCDConf_07X1.h 中使用 LCD_CONTROLLER 宏。下表列出了用于选择正确的控制器的值：

编号	支持的控制器
701	联咏 NT7506 所罗门 SSD1854
702	意法半导体 STE2010
711	三星 KS0711、S6B0711
741	三星 KS0741、S6B0741 矽创 ST7541、ST7571 Tomato TL0350A

显示数据 RAM 的组织



上图展示了显示存储器与 LCD SEG 和 COM 线之间的关系。每个像素所用显示存储器分为两个窗格。每个像素的低位存于窗格 0 中，高位则存于窗格 1 中。

RAM 要求

该 LCD 驱动可以使用显示数据缓存，也可以不使用，该缓存中含有 LCD 数据 RAM 中内容的完整拷贝。如果不使用缓存，则无需增加 RAM。

建议配合该驱动使用一个数据缓存，以提高 LCD 存取速度。缓存所用存储器的大小可以通过以下等式计算：

RAM 大小 （单位：字节） = (LCD_YSIZE + 7) / 8 * LCD_XSIZE * 2

其他驱动函数

LCD_ControlCache

有关此函数的信息，请参见第 891 页。

硬件配置

如“配置”（第 905 页）部分中所描述的那样，该驱动以简单总线接口存取硬件。下表列出了必须针对硬件存取进行定义的宏：

宏	说明
LCD_READ_A0	从 LCD 控制器读取一个字节，其中， A 线为低电平。（仅在无缓存时使用）
LCD_READ_A1	从 LCD 控制器读取一个字节，其中， A 线为高电平。（仅在无缓存时使用）
LCD_WRITE_A0	向 LCD 控制器写入一个字节，其中， A 线为低电平。
LCD_WRITE_A1	向 LCD 控制器写入一个字节，其中， A 线为高电平。
LCD_WRITEM_A1	向 LCD 控制器写入多个字节，其中， A 线为高电平。

显示方向

支持的显示控制器支持硬件镜像显示 x/y 轴。建议使用这些函数而不是 emWin 的显示方向宏。如果需要镜像显示 X 轴，则应在初始化宏中使用命令 0xA1 （ADC 选择反向）。结果会使显示控制器颠倒指派给段输出的栏地址。如果 X 中的显示尺寸小于显示控制器的段输出数，则可利用宏 LCD_FIRSTSEG0 来给栏地址添加一个偏移，以确保存取 LCD 控制器的正确 RAM 地址。
如果需要镜像显示Y轴，则应在初始化宏中使用命令0xC8（SHL选择反向），同时用宏LCD_FIRSTCOM0 定义偏移，以访问显示控制器的正确 RAM 地址。

其他配置开关

下表列出了该驱动可以使用的可选配置开关：

宏	说明
LCD_FIRSTCOM0	该宏可以用于定义显示控制器数据 RAM 中使用的第一个共用地址。其值可以通过实验决定，也可参考显示器文档。
LCD_FIRSTSEG0	该宏可以用于定义显示控制器数据 RAM 中使用的第一个段地址。其值可以通过实验决定，也可参考显示器文档。

某些 LCD 控制器的特殊要求

无。

28.7.15 GUIDRV_1611

支持的硬件

控制器

该驱动支持以下显示控制器：

- 爱普生 S1D15E05、S1D15E06、S1D15719
- 晶宏 UC1610、UC1611

每像素位数

支持的颜色深度为 2bpp（UC1610、S1D15E05、S1D15E06、S1D15719）和 4bpp（UC1611）。

接口

该驱动支持显示控制器的 8 位间接接口。可以使用并行 4 引脚 SPI 或 I2C 总线。

驱动选择

举例来说，如果要选择 GUIDRV_1611 作为应用所使用的驱动，则可以使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_1611, GUICC_2, 0, 0);
```

请参见 “颜色”（第 227 页）一章，以了解有关利用正确的调色板模式的更多信息。

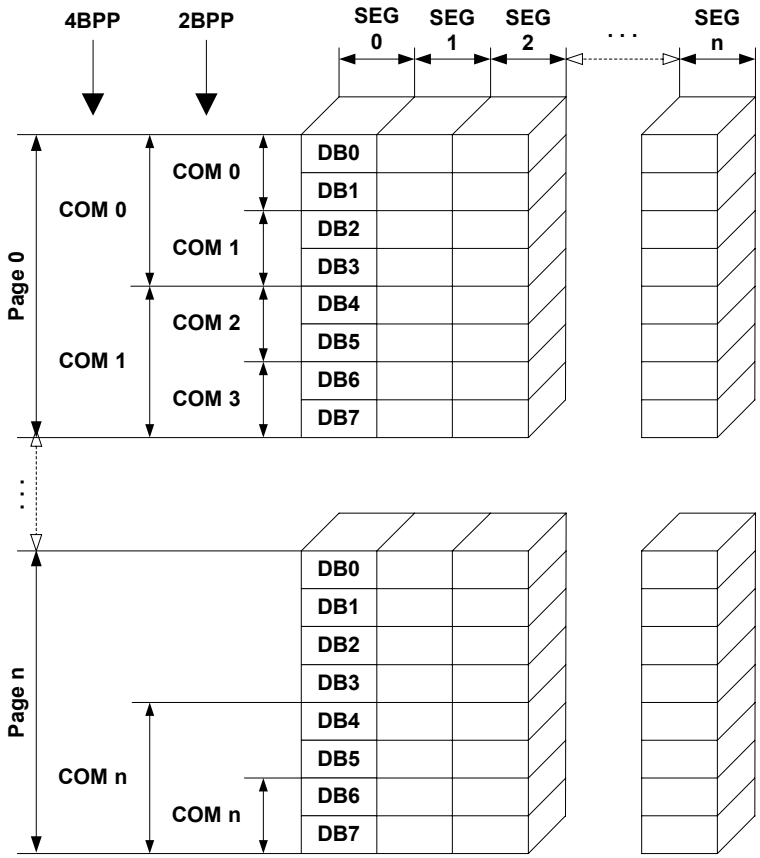
可用的配置宏 （编译时间配置）

控制器选择

如果要选择所需的控制器，则应当在配置文件 LCDConf_1611.h 中使用 LCD_CONTROLLER 宏。下表列出了用于选择正确的控制器的值：

编号	支持的控制器
1701	爱普生 S1D15E05
1702	爱普生 S1D15719
1800	晶宏 UC1611
1801	晶宏 UC1610
1802	晶宏 UC1611s

显示数据 RAM 的组织



上图展示了显示存储器与 LCD SEG 和 COM 线之间的关系。

RAM 要求

该显示驱动可以使用显示数据缓存，也可以不使用。数据缓存含有 LCD 数据 RAM 的完整拷贝。如果不使用缓存，则无需增加 RAM。
强烈建议配合该驱动使用一个数据缓存，以提高 LCD 存取速度。不使用缓存会大幅降低该驱动的性能。缓存所用存储器的大小可以通过以下等式计算：

```
RAM 大小 （单位：字节） =
(LCD_YSIZE + (8 / LCD_BITSPERPIXEL - 1)) / 8 * LCD_BITSPERPIXEL * LCD_XSIZE
```

硬件配置

该驱动以间接接口存取硬件。下表列出了需要针对硬件存取进行定义的宏：

宏	说明
LCD_READ_A0	从 LCD 控制器读取一个字节，其中， A 线为低电平。
LCD_READ_A1	从 LCD 控制器读取一个字节，其中， A 线为高电平。
LCD_WRITE_A0	向 LCD 控制器写入一个字节，其中， A 线为低电平。
LCD_WRITE_A1	向 LCD 控制器写入一个字节，其中， A 线为高电平。
LCD_WRITEM_A1	向 LCD 控制器写入多个字节，其中， A 线为高电平。

其他配置开关

下表列出了该驱动可以使用的可选配置开关：

宏	说明
LCD_CACHE	设为 0 时，不使用显示数据缓存，结果会减慢驱动的速度。默认值为 1 （缓存激活）。

某些 LCD 控制器的特殊要求

无。

28.7.16 GUIDRV_6331

支持的硬件

控制器

该驱动已通过三星 S6B33B1X 显示控制器的测试。

每像素位数

支持的颜色深度为 16bpp。

接口

该驱动支持显示控制器的 8 位间接接口。可以使用并行或 4 引脚 SPI 总线。

驱动选择

如果要选择 GUIDRV_6331 作为应用所使用的驱动，则可以使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_6331, GUICC_565, 0, 0);
```

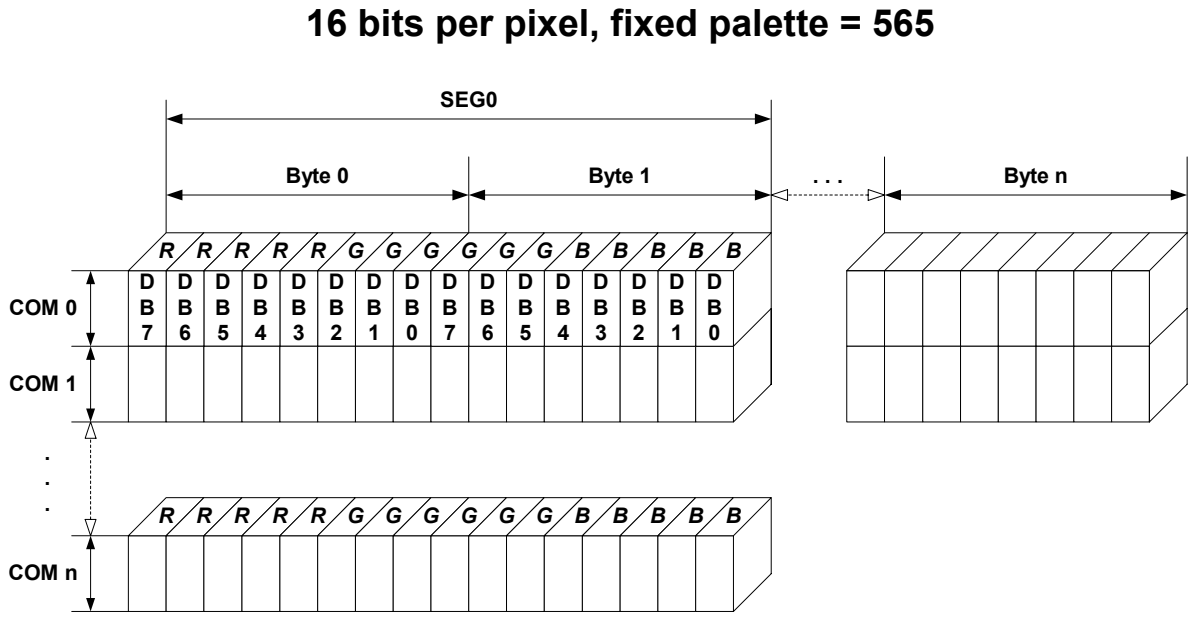
可用的配置宏 （编译时间配置）

控制器选择

如果要选择所需的控制器，则应当在配置文件 LCDConf_6331.h 中使用 LCD_CONTROLLER 宏。下表列出了用于选择正确的控制器的值：

编号	支持的控制器
6331	三星 S6B33B1X

显示数据 RAM 的组织



上图展示了显示存储器与 LCD SEG 和 COM 线之间的关系。

RAM 要求

该显示驱动可以使用显示数据缓存，也可以不使用，该缓存中含有 LCD 数据 RAM 的完整拷贝。缓存所用存储器大小为：LCD_XSIZE x LCD_YSIZE x 2 字节。

硬件配置

该驱动以间接接口存取硬件。下表列出了必须针对硬件存取进行定义的宏：

宏	说明
LCD_WRITE_A0	向显示控制器写入一个字节，其中， A 线为低电平。
LCD_WRITE_A1	向显示控制器写入一个字节，其中， A 线为高电平。
LCD_WRITEM_A1	向显示控制器写入多个字节，其中， A 线为高电平。
LCD_DRIVER_OUTPUT_MODE_DLN	“驱动输出模式设置”指令的 “显示线编号” (DLN) 选择位。有关详情，请参见显示控制器文档。
LCD_DRIVER_ENTRY_MODE_16B	“输入模式设置”指令的数据总线宽度选择位。有关详情，请参见显示控制器文档。

该驱动会自行初始化 “驱动输出模式”和 “输入模式”寄存器。用户无需在 LCD_X_InitController() 中初始化该寄存器。

其他配置开关

下表列出了该驱动可以使用的可选配置开关：

宏	说明
LCD_CACHE	设为 0 时，不使用显示数据缓存，结果会减慢驱动的速度。默认值为 1 （缓存激活）。

特殊要求

该驱动需要采用固定调色板模式 565。驱动不支持其他调色板模式或固定调色板模式。另外，该驱动需要交换色指数的红色和蓝色部分。应在配件文件 LCDConf.h 中使用以下宏定义：

```
#define LCD_FIXEDPALETTE 565
#define LCD_SWAP_RB        1
```

28.7.17 GUIDRV_7529

支持的硬件

控制器

该驱动支持矽创 ST7529 显示控制器。

每像素位数

支持的颜色深度为 5bpp （默认）、 4bpp 和 1bpp。

接口

该驱动支持显示控制器的间接接口（8位和16位）。可以使用并行、3引脚SPI或4引脚SPI存取模式。

驱动选择

举例来说，如果要选择 GUIDRV_7529 作为应用所使用的驱动，则可以使用以下命令：

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_7529, GUICC_5, 0, 0);
```

请参见 “颜色”（第 227 页）一章，以了解有关利用正确的调色板模式的更多信息。

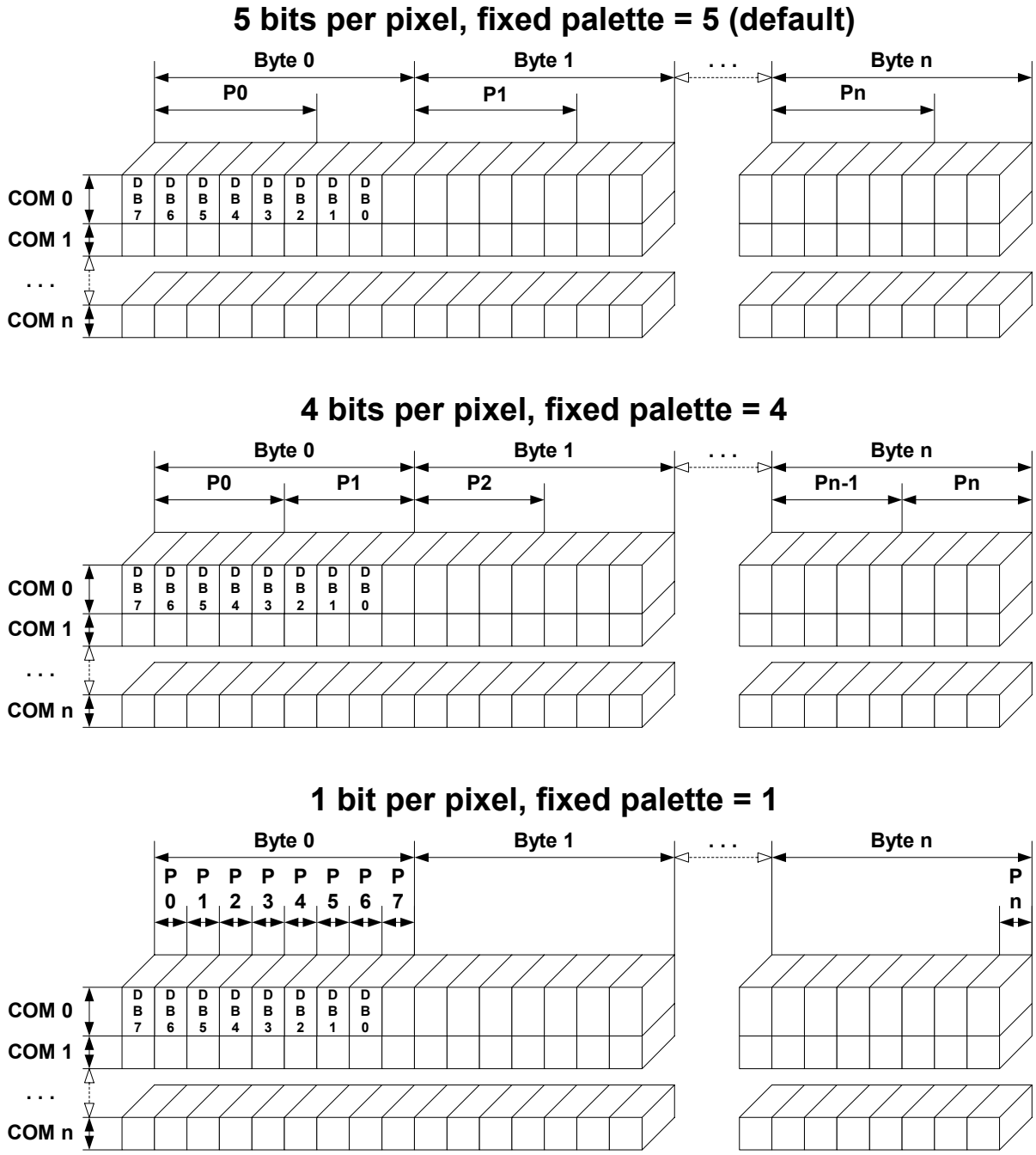
可用的配置宏 （编译时间配置）

控制器选择

如果要选择所需的控制器，则应当在配置文件 LCDConf_7529.h 中使用 LCD_CONTROLLER 宏。下表列出了用于选择正确的控制器的值：

编号	支持的控制器
7529	矽创 ST7529

显示数据 RAM 的组织



上图展示了显示存储器与 LCD SEG 和 COM 线之间的关系。

RAM 要求

该显示驱动可以使用显示数据缓存，也可以不使用，该缓存中含有 LCD 数据 RAM 的完整拷贝。如果不使用缓存，则无需增加 RAM。
可以选择 （但建议）配合该驱动使用一个数据缓存，以提高 LCD 存取速度。缓存所用存储器的大小可以通过以下等式计算：

5bpp 模式：
RAM 大小 （单位：字节） = (LCD_XSIZE + 2) / 3 * LCD_YSIZE * 3
4bpp 模式：
RAM 大小 （单位：字节） = ((LCD_XSIZE + 2) / 3 * 3 + 1) / 2 * LCD_YSIZE
1bpp 模式：
RAM 大小 （单位：字节） = ((LCD_XSIZE + 2) / 3 * 3 + 7) / 8 * LCD_YSIZE

硬件配置

该驱动以间接接口存取硬件。下表列出了必须针对硬件存取进行定义的宏：

宏	说明
LCD_WRITE_A0	向 LCD 控制器写入一个字节，其中， A 线为低电平。
LCD_WRITE_A1	向 LCD 控制器写入一个字节，其中， A 线为高电平。
LCD_WRITEM_A1	向显示控制器写入多个字节，其中， A 线为高电平。
LCD_READM_A1	从显示控制器读取多个字节，其中， A 线为高电平。仅当未配置显示数据缓存时才需使用。
LCD_FIRSTPIXEL0	如果 X 中的显示尺寸小于显示控制器的段输出数，则可利用该宏来定义显示器的第一个可见像素。如果显示控制器的第一段线未与显示器相连，则必须使用该宏。

其他配置开关

下表列出了该驱动可以使用的可选配置开关：

宏	说明
LCD_CACHE	设为 0 时，不使用显示数据缓存，结果会减慢驱动的速度。默认值为 1 （缓存激活）。

某些 LCD 控制器的特殊要求

无。

28.7.18 GUIDRV_Template—— 新驱动模板

该驱动随基本版提供，可轻松调整以适应各种显示控制器。其中含有显示驱动所需要的全部功能。

调整模板驱动

若要调整该驱动以适应当前不支持的显示控制器，只需要改动 _SetPixelIndex() 和 _GetPixelIndex() 两个程序即可。调用这两个程序的上层已经确保给定的坐标未超出范围，因而不需要对参数进行检验。
如果显示器不可读取，则函数 _GetPixelIndex() 将无法读回显示数据 RAM 中的内容。这种情况下，应在驱动中使用一个显示数据缓存，以使驱动知晓每个像素的内容。如果在这种情况下，没有数据缓存可用，则 emWin 有部分函数将无法正常工作。这些函数都需要反转像素。特别是， XOR 绘图模式和文本游标绘图（同样使用 XOR 绘图模式）不会正常工作。如果不调整函数 _GetPixelIndex()，不使用 XOR 绘图模式的简单应用同样会失效。
第二步则应对其进行优化，以提高绘图速度。

28.8 LCD 层和显示驱动 API

emWin 需要使用硬件驱动。本章将说明 emWin LCD 驱动所起的作用及其向 emWin 提供的程序（应用编程接口 API）。
多数情况下，可能不需要阅读本章内容，因为对 emWin LCD 层的调用多数都是通过 GUI 层实现的。事实上，我们建议在没有 GUI 等效工具时才调用 LCD 函数 （比如，如果希望直接修改 LCD 控制器的查找表）。其原因在于，与其 GUI 等效工具不一样，LCD 驱动函数不具线程安全性。因此，在多任务环境中不应直接调用。

28.8.1 显示驱动 API

下表以字母顺序列出了可用的 emWin LCD 相关程序。有关程序的详细描述可参见后续各节。

用户自定义程序

程序	描述
LCD_X_InitController()	由显示驱动调用，以使显示控制器初始化。

LCD 层程序

程序	描述
"Get" 组	
LCD_GetBitsPerPixel()	返回每像素位数。
LCD_GetBitsPerPixelEx()	返回给定层 / 显示器的每像素位数。
LCD_GetNumColors()	返回可用颜色数。
LCD_GetNumColorsEx()	返回给定层 / 显示器的可用颜色数。
LCD_GetVXSize()	返回 LCD 的虚拟 X 尺寸 （单位：像素）。
LCD_GetVXSizeEx()	返回给定层 / 显示器的虚拟 X 尺寸 （单位：像素）。
LCD_GetVYSize()	返回 LCD 的虚拟 Y 尺寸 （单位：像素）。
LCD_GetVYSizeEx()	返回给定层 / 显示器的虚拟 Y 尺寸 （单位：像素）。
LCD_GetXMag()	返回 x 轴放大倍数。
LCD_GetXMagEx()	返回给定层 / 显示器的 x 轴放大倍数。
LCD_GetXSize()	返回 LCD 的物理 X 尺寸 （单位：像素）。
LCD_GetXSizeEx()	返回给定层 / 显示器的物理 X 尺寸 （单位：像素）。
LCD_GetYMag()	返回 y 轴放大倍数。
LCD_GetYMagEx()	返回给定层 / 显示器的 y 轴放大倍数。
LCD_GetYSize()	返回 LCD 的物理 Y 尺寸 （单位：像素）。
LCD_GetYSizeEx()	返回给定层 / 显示器的物理 Y 尺寸 （单位：像素）。
配置组	
LCD_SetDevFunc()	为显示驱动设置可选或定制程序。
LCD_SetSizeEx()	设置给定层的物理尺寸 （单位：像素）。
LCD_SetVRAMAddrEx()	设置给定层视频 RAM 的地址。
LCD_SetVSizeEx()	设置给定层虚拟显示区的大小 （单位：像素）。
缓存组	
LCD_ControlCache()	锁定、解锁和清除显示控制器的缓存 （若支持）。

28.8.2 用户自定义程序

LCD_X_InitController()

描述

此函数由显示控制器调用。程序的作用是使显示控制器寄存器正确初始化。

原型

```
void LCD_X_InitController(unsigned LayerIndex);
```

参数	描述
LayerIndex	待初始化的层的索引。

其他信息

示例文件夹含有有关多种显示控制器的一些示例。此函数取代了老版本中使用的宏 LCD_INIT_CONTROLLER。

28.8.3 LCD 层程序

28.8.3.1 "Get" 组

LCD_GetBitsPerPixel()

描述

返回每像素位数。

原型

```
int LCD_GetBitsPerPixel(void);
```

返回值

每像素位数。

LCD_GetBitsPerPixelEx()

描述

返回每像素位数。

原型

```
int LCD_GetBitsPerPixelEx(int Index);
```

参数	描述
索引	层索引。

返回值

每像素位数。

LCD_GetNumColors()

描述

返回 LCD 当前可用的颜色数。

原型

```
int LCD_GetNumColors(void);
```

返回值

可用颜色数

LCD_GetNumColorsEx()

描述

返回 LCD 当前可用的颜色数。

原型

```
U32 LCD_GetNumColorsEx(int Index);
```

参数	描述
索引	层索引。

返回值

可用颜色数。

LCD_GetVXSize(), LCD_GetVYSize()

描述

分别返回 LCD 的虚拟 X 尺寸和 Y 尺寸 （单位：像素）。多数情况下，虚拟尺寸等于物理尺寸。

原型

```
int LCD_GetVXSize(void)
int LCD_GetVYSize(void)
```

返回值

显示器的虚拟 X/Y 尺寸。

LCD_GetVXSizeEx(), LCD_GetVYSizeEx()

描述

分别返回 LCD 的虚拟 X 尺寸和 Y 尺寸 （单位：像素）。大多情况下，虚拟尺寸等于物理尺寸。

原型

```
int LCD_GetVXSizeEx(int Index);
int LCD_GetVYSizeEx(int Index);
```

参数	描述
索引	层索引。

返回值

显示器的虚拟 X/Y 尺寸。

LCD_GetXMag(), LCD_GetYMag()

描述

分别返回 X 轴和 Y 轴的放大倍数。

原型

```
int LCD_GetXMag(int Index);
int LCD_GetYMag(int Index);
```

返回值

X 轴和 Y 轴的放大倍数。

LCD_GetXMagEx(), LCD_GetYMagEx()

描述

分别返回 X 轴和 Y 轴的放大倍数。

原型

```
int LCD_GetXMagEx(int Index);
```

参数	描述
索引	层索引。

返回值

X 轴和 Y 轴的放大倍数。

LCD_GetXSize(), LCD_GetYSize()

描述

分别返回 LCD 的物理 X 尺寸和 Y 尺寸 （单位：像素）。

原型

```
int LCD_GetXSize(void)
int LCD_GetYSize(void)
```

返回值

显示器的物理 X/Y 尺寸。

LCD_GetXSizeEx(), LCD_GetYSizeEx()

描述

分别返回 LCD 的物理 X 尺寸和 Y 尺寸 （单位：像素）。

原型

```
int LCD_GetXSizeEx(int Index);
int LCD_GetYSizeEx(int Index);
```

参数	描述
Index	层索引。

返回值

显示器的物理 X/Y 尺寸。

28.8.3.2 配置组

LCD_SetDevFunc()

描述

此函数设置显示驱动的附加和 / 或用户自定义函数。

原型

```
int LCD_SetDevFunc(int LayerIndex, int IdFunc, void (* pDriverFunc)(void));
```

参数	描述
LayerIndex	层索引。
IdFunc	（参见下表）
pDriverFunc	指向应使用的函数的指针。

元素 IdFunc 的允许值	
LCD_DEVFUNC_COPYBUFFER	可用于设置定制的缓冲器复制程序。需与多个缓冲器配合使用。
LCD_DEVFUNC_COPYRECT	可用于设置定制的矩形区复制程序。
LCD_DEVFUNC_DRAWBMP_1BPP	可用于设置定制的 1bpp 位图绘图程序。仅适用于需要用 BitBLT 引擎来绘制文本和 1bpp 位图的情况。
LCD_DEVFUNC_FILLRECT	可用于设置定制的矩形填充程序。适用于需要用 BitBLT 引擎来执行填充操作等情况。

LCD_DEVFUNC_COPYBUFFER

可用于设置一个函数，以将帧缓冲器复制到目标位置。如果有 BitBLT 引擎可用于完成该项任务，则可使用该函数。
pDriverFunc 指向的函数应类似于：

```
void CopyRect(int LayerIndex, int x0, int y0, int x1, int y1,
              int xSize,      int ySize)
```

参数	描述
LayerIndex	层索引。
IndexSrc	待复制的源帧缓冲器的索引。
IndexDst	待覆盖的目标帧缓冲器的索引。

LCD_DEVFUNC_COPYRECT

可用于设置一个函数，以将屏幕的一个矩形区复制到目标位置。如果有 BitBLT 引擎可用于完成该项任务，则可使用该函数。
pDriverFunc 指向的函数应类似于：

```
void CopyRect(int LayerIndex, int x0, int y0, int x1, int y1,
              int xSize,      int ySize);
```

参数	描述
LayerIndex	层索引。
x0	源矩形最左边的像素。
y0	源矩形最上边的像素。
x1	目标矩形最左边的像素。
y1	目标矩形最上边的像素。
xSize	矩形的 X 尺寸。
ySize	矩形的 Y 尺寸。

LCD_DEVFUNC_FILLRECT

可用于设置一个定制函数，用于执行填充操作。 pDriverFunc 指向的函数应类似于：

```
void FillRect(int LayerIndex,  int x0, int y0, int x1, int y1,
              U32 PixelIndex);
```

参数	描述
LayerIndex	层索引。
x0	屏幕坐标中待填充的最左边坐标。
y0	屏幕坐标中待填充的最上边坐标。
x1	屏幕坐标中待填充的最右边坐标。
y1	屏幕坐标中待填充的最下边坐标。
PixelIndex	待用于填充指定区域的颜色索引。

LCD_DEVFUNC_DRAWBMP_1BPP

可用于设置一个函数，用于绘制同时含有文本的 1bpp 位图。如果有 BitBLT 引擎可用于完成该项任务，则可使用该函数。

pDriverFunc 指向的函数应类似于：

```
void DrawBMP1(int LayerIndex,
              int x, int y, U8 const * p, int Diff, int xSize, int ySize,
               int BytesPerLine, const LCD_PIXELINDEX * pTrans);
```

参数	描述
LayerIndex	层索引。
x	待绘制的位图在屏幕坐标中的最左边坐标。
y	待绘制的位图在屏幕坐标中的最上边坐标。
p	指向位图像素数据的指针。
Diff	参数 p 指向的第一个像素的偏移值。支持的值为 0 至 7。
xSize	待绘制的位图的 xSize （单位：像素）。
ySize	待绘制的位图的 ySize （单位：像素）。
BytesPerLine	一行位图数据含有的字节数。
pTrans	指向用于绘制位图数据的颜色索引阵列的指针。第一个颜色索引定义背景颜色，第二个颜色索引定义前景颜色。

返回值

如果成功，返回值为 0；如果发生错误，则返回值为 1。

其他信息

需要注意的是，支持参数 IdFunc 的哪些值取决于显示驱动。

LCD_SetSizeEx()

描述

设置给定显示器 / 层可见区的物理尺寸。

原型

```
int LCD_SetSizeEx(int LayerIndex, int xSize, int ySize);
```

参数	描述
LayerIndex	层索引。
xSize	设置给定层可见区的 X 尺寸 （单位：像素）。
ySize	给定层可见区的 Y 尺寸 （单位：像素）。

返回值

如果成功，返回值为 0 ；如果发生错误，则返回值为 1。

其他信息

此函数要求使用一个能动态管理显示尺寸变化的显示驱动。如果显示驱动不支持这种功能，则函数将失效。

LCD_SetVRAMAddrEx()

描述

设置视频 RAM 的地址。

原型

```
int LCD_SetVRAMAddrEx(int LayerIndex, void * pVRAM);
```

参数	描述
LayerIndex	层索引。
pVRAM	指向视频 RAM 的起始地址的指针。

返回值

如果成功，返回值为 0 ；如果发生错误，则返回值为 1。

其他信息

此函数要求使用一个能动态管理视频 RAM 地址变化的显示驱动。如果显示驱动不支持这种功能，则函数将失效。

LCD_SetVSizeEx()

描述

设置虚拟显示区的尺寸。

原型

```
int LCD_SetVSizeEx(int LayerIndex, int xSize, int ySize);
```

参数	描述
LayerIndex	层索引。
xSize	给定层虚拟区的 X 尺寸 （单位：像素）。
ySize	给定层虚拟区的 Y 尺寸 （单位：像素）。

返回值

如果成功，返回值为 0 ；如果发生错误，则返回值为 1。

其他信息

此函数要求使用一个能动态管理虚拟显示尺寸变化的显示驱动。如果显示驱动不支持这种功能，则函数将失效。

28.8.3.3 缓存组

LCD_ControlCache()

描述

锁定、解锁和清除显示控制器的缓存 （若支持）。

原型

```
int LCD_ControlCache(int Cmd);
```

参数	描述
Cmd	含有需要用缓存完成的动作： LCD_CC_LOCK, LCD_CC_UNLOCK or LCD_CC_FLUSH

返回值

如果成功，返回值为 0；如果发生错误，则返回值为 1。

其他信息

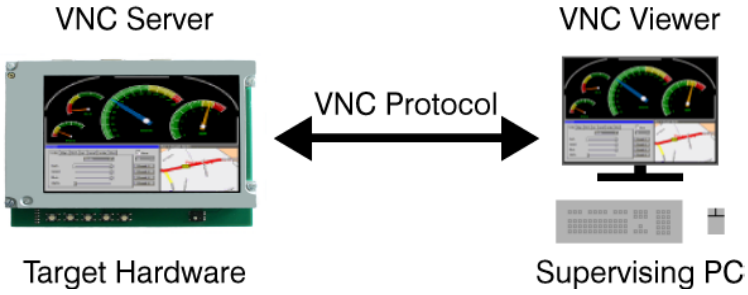
此函数要求使用一个能动态管理虚拟显示尺寸变化的显示驱动。如果显示驱动不支持这种功能，则函数将失效。
此函数被自动用于窗口和字符串的绘图操作。

第 29 章

VNC 服务器

emWin VNC 服务器可用于管理嵌入式目标以及多种其他用途，支持压缩 (hextile) 编码。
VNC 表示“虚拟网络计算”。它是基于一种简单的显示协议的客户端 - 服务器系统，允许用户从互联网的任何地方查看和控制 “桌面”计算环境，支持多种机器架构，采用的是 TCP/IP 通信协议。
换言之：嵌入式设备的显示内容可以在运行客户端的机器 （如 PC）的屏幕上看到；可以用您的鼠标和键盘来控制目标。

emWin 仿真和试用版都提供了该功能。
emWin VNC 支持包单独提供，不包含在基本版之中。



29.1 简介

VNC 由两类组件构成：一是服务器，生成显示，一是查看器，在您的屏幕上绘制显示内容。不但可以看到远程机器 （目标或仿真），而且可以通过鼠标和键盘进行控制。
服务器和查看器可以在不同的机器上，可以采用不同的架构。用于连接服务器和查看器的协议具有简单、开放且不受平台限制的特点。查看器不存储状态。断开查看器与服务器的连接，然后重新连接不会导致数据丢失。由于可以从其他地方重新连接，因此可以轻松实现移动性。使用 VNC 服务器，您可以从任何地方控制您的目标机器，您可以从 “实时”系统中截图 （比如，用于制作手册）。

29.1.1 要求

TCP/IP 堆栈

由于服务器与查看器之间的通信是基于 TCP/IP 连接的，因此，VNC 要求使用 TCP/IP 堆栈。在 Win32 仿真环境中，一般都有 TCP/IP (Winsock)。在目标机器中，需要有 TCP/IP 堆栈。emWin 并不提供 TCP/IP 堆栈。灵活的接口设计可以确保支持任何 TCP/IP 堆栈。

多任务

VNC 服务器需要以独立线程运行。因此，使用 emWin VNC 服务器要求采用多任务系统。

29.1.2 实现说明

支持的客户端到服务器消息

emWin VNC 服务器支持指针事件消息和键盘事件消息。

编码

服务器支持原始编码和 hextile 编码。

性能

多数查看器都支持 hextile 编码，该编码方案支持下降压缩。典型情况下，四分之一 VGA 屏幕要求 20 - 50 kb 的数据。运行于 ARM7 平台 （50 MHZ，带缓存）的实现方案更新整个屏幕大约需要 200 – 300 毫秒。
服务器处理增量更新；多数情况下，更新后的显示区比整个显示区要小得多，只需要传输较少的数据。因此，典型的 ARM7 系统支持实时更新。

多服务器

这种实现方案完全具有线程安全性和重入性；可以在同一个 CPU 上为不同的层或显示器启用多个 VNC 服务器。如果您的目标机器（同样适用于仿真）有多个显示器或多层，则可以使用这种选项。在任何给定时间，每层只能启用一个 VNC 服务器；当与查看器的连接终止时，可以连接另一个。

29.2 VNC 查看器

软件源

Tool 子文件夹中含有 AT&T 剑桥实验室所提供的 VNC 查看器。这是一款免费软件，以 GNU 公共许可证的方式发布。这款 VNC 软件的最新版可以从 http://www.uk.research.att.com/vnc 下载。该网站同时提供有关 VNC 协议的详细信息，以及针对不同平台的服务器版和客户端的源代码。

版本

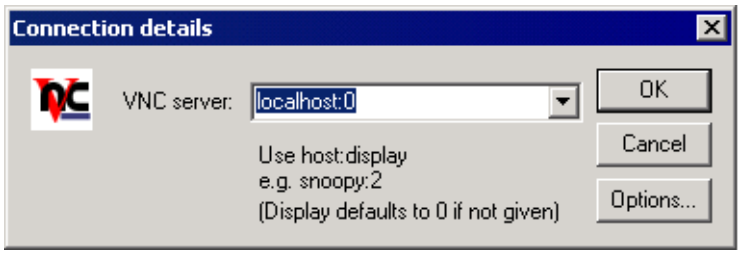
emWin 的 VNC 插件已通过 3.3.3R2 版的测试，应该能支持最新版和后续版本。

平台

查看器提供针对不同平台的版本。有关详情，请参见 AT&T 剑桥实验室网站。

29.2.1 启动 VNC 查看器

双击 Tool\VNCViewer.exe 文件即可启动查看器。将提示连接 VNC 服务器：



使用同一计算机上的仿真系统连接到 VNC 服务器

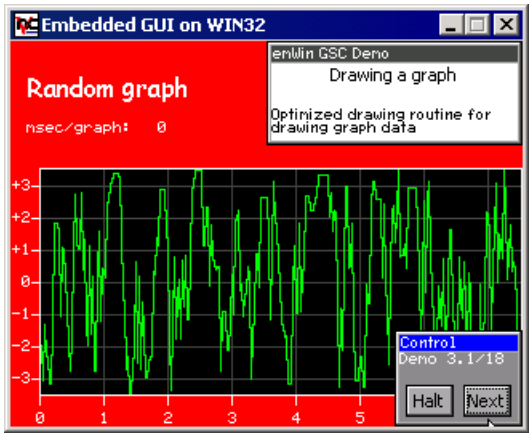
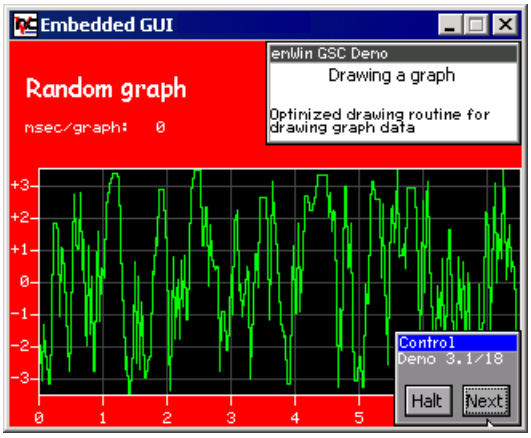
当在同一计算机上运行 VNC 查看器和仿真系统时，输入“localhost:0”即可连接。“:0”表示服务器索引 0。如果略去服务器索引，则查看器将假定使用服务器 0。因此，在多数情况下，通过输入“localhost”即可连接仿真系统。

连接至运行于不同计算机或目标系统上的 VNC 服务器

如果要连接至运行于不同计算机或目标系统上的 VNC 服务器，则请输入机器的名称或 IP 地址（可以选择在其后加上“:”和服务器编号）。比如，如果连接至计算机“Joerg”上的 VNC 服务器，其 IP 地址为“192.168.1.2:0”，则可以输入“192.168.1.2:0”、“Joerg:0”或“Joerg”。
如果要连接至 IP 地址为 192.168.1.254 的目标系统，则可输入“192.168.1.254”。

屏幕截图

以下截图展示了该查看器：

连接至仿真系统	连接至目标系统
	

29.3 emWinVNC 服务器

29.3.1 启动 emWin VNC 服务器

启动 VNC 服务器唯一要做的是调用 GUI_VNC_X_StartServer() 函数：

```
void MainTask(void) {
  GUI_Init();
  GUI_VNC_X_StartServer(0,    /* Layer index */
                        0);   /* Server index */
  ...
}
```

以上函数调用语句将创建一个线程，监听 5900 端口上的输入连接。检测到连接后，将调用 GUI_VNC_Process() 函数。

端口

VNC 服务器监听 590x 号端口，其中，x 为服务器索引。因此，对于多数 PC 服务器而言，该端口为 5900，因为它们默认情况下使用显示器 0。

示例

我们的网站上提供有一个现成的用例 （为可执行文件）。试用版也包括 VNC 服务器；激活该服务器只需要一行代码 （用 GUI_VNC_X_StartServer()）。

29.3.2 服务器的启动方式 ...

使用仿真系统时，只需调用 GUI_VNC_X_StartServer() 函数。结果将创建一个线程，对 590x 号端口进行监听，直到检测到输入连接为止，然后再调用 GUI_VNC_Process() （实际服务器的实现）。

29.3.3 VNC 服务器在目标系统上的集成

使用 GUI_VNC_X_StartServer() 函数之前，必须针对所用 TCP/IP 堆栈和多任务系统进行调整。文件夹 Sample\GUI_X\GUI_VNC_X_VNCServer.c 下有一个实现示例，只需作出很小的改动即可。示例文件未使用动态存储器分配模式来给后面将提到的 GUI_VNC_CONTEXT 结构分配存储器。因此，该实现方式只允许启动一个服务器。

29.4 要求

ROM

在 ARM7 平台上，有 hextile 编码大约需要 4.9 kb 的空间，无 hextile 编码则需要大约 3.5 kb。

RAM

VNC 不使用静态数据。对于每个实例，使用一个 GUI_VNC_CONTEXT 结构 （约 60 字节）。

其他

每个实例需要一个 TCP/IP 插口和一个线程。

29.5 配置选项

类型	宏	默认值	描述
N	GUI_VNC_BUFFER_SIZE	1000	帧缓冲器尺寸。该缓冲器位于堆栈上。典型情况下，增大尺寸只会略微提高速度。缓冲器的合理尺寸约为 200 字节。
B	GUI_VNC_LOCK_FRAME	0	若设为 1，则 GUI 将在帧发送至查看器期间锁定。如果需要对文档截图，则可使用该选项。
S	GUI_VNC_PROGNAME	（见说明）	该宏定义查看器标题栏中所示目标的名称。如果在仿真系统中使用查看器，则默认值为： "Embedded GUI on WIN32" 在目标系统中，默认值为： "Embedded GUI"
B	GUI_VNC_SUPPORT_HEXTILE	1	启用或禁用 hextile 编码。Hextile 编码速度更快，但代码尺寸更大 （约大 1.4 k）。

29.6 VNC API

下表按字母顺序列出了可用的 VNC 相关函数。有关程序的详细描述可参见后续各节。

程序	描述
GUI_VNC_AttachToLayer()	将一个 VNC 服务器附加到一个层。如果没有多显示配置，则给定的索引必须是 0。
GUI_VNC_EnableKeyboardInput()	启用或禁用基于 VNC 的键盘输入。
GUI_VNC_GetNumConnections()	返回至服务器的连接数。
GUI_VNC_Process()	实际 VNC 服务器；初始化与查看器的通信。
GUI_VNC_RingBell()	使客户端振铃 （若有）。
GUI_VNC_SetPassword()	设置服务器连接密码。
GUI_VNC_SetProgName()	设置显示在查看器标题栏中的文字。
GUI_VNC_SetSize()	设置待传输至客户端的区域。
GUI_VNC_X_StartServer()	待调用以启动 VNC 查看器的程序。

GUI_VNC_AttachToLayer()

描述

此函数将给定的层附加到 VNC 服务器。正常情况下，在单层配置中，该参数应为 0。

原型

```
void GUI_VNC_AttachToLayer(GUI_VNC_CONTEXT * pContext, int LayerIndex);
```

参数	描述
pContext	指向某种 GUI_VNC_CONTEXT 结构的指针。
LayerIndex	将由服务器处理的层的索引（基数为零）。

返回值

如果函数成功，返回值为 0；如果失败，则返回 != 0。

GUI_VNC_EnableKeyboardInput()

描述

启用或禁用基于 VNC 的键盘输入。

原型

```
void GUI_VNC_EnableKeyboardInput(int OnOff);
```

参数	描述
OnOff	1 启用键盘输入， 0 禁用。

GUI_VNC_GetNumConnections()

描述

返回至服务器的现有连接数。

原型

```
int GUI_VNC_GetNumConnections(void);
```

返回值

连接数。

GUI_VNC_Process()

描述

此函数设置数据发送和接收函数，并启动与查看器的通信。

原型

```
void GUI_VNC_Process(GUI_VNC_CONTEXT * pContext,
                     GUI_tSend pfSend,
                     GUI_tReceive pfReceive,
                     void * pConnectInfo);
```

参数	描述
pContext	指向某种 GUI_VNC_CONTEXT 结构的指针。
pfSend	指向服务器用来向查看器发送数据的函数的指针。
pfReceive	指向服务器用来从查看器读取数据的函数的指针。
pConnectInfo	待传递给发送和接收函数的指针。

其他信息

GUI_VNC_CONTEXT 结构由服务器用于存储连接状态信息。
发送和接收函数应返回查看器成功发送或接收的字节数。
指针 pConnectInfo 被传递至发送和接收程序。可用于传递指向含有连接信息的结构的指针，或用于传递插口编号。
以下类型被用作函数指针 （指向用于向 / 从查看器发送 / 接收字节的程序）：

```
typedef int (*GUI_tSend)    (const U8 * pData, int len, void* pConnectInfo);
typedef int (*GUI_tReceive)(      U8 * pData, int len, void* pConnectInfo);
```

示例

```
static GUI_VNC_CONTEXT _Context; /* Data area for server */

static int _Send(const U8* buf, int len, void * pConnectionInfo) {
  SOCKET Socket = (SOCKET)pConnectionInfo;
 ...
}
static int _Recv(U8* buf, int len, void * pConnectionInfo) {
  SOCKET Socket = (SOCKET)pConnectionInfo;
  ...
}
static void _ServerTask(void) {
  int Socket;
  ...
  GUI_VNC_Process(&_Context, _Send, _Recv, (void*)Socket);
  ...
}
```

GUI_VNC_RingBell()

描述

使客户端振铃 （若有）。

原型

```
void GUI_VNC_RingBell(void);
```

GUI_VNC_SetPassword()

描述

设置服务器连接密码。

原型

```
void GUI_VNC_SetPassword(U8 * sPassword);
```

参数	描述
sPassword	连接服务器需要的密码。

其他信息

默认情况不需要密码。

GUI_VNC_SetProgName()

描述

设置显示于客户端窗口标题栏的标题。

原型

```
void GUI_VNC_SetProgName(const char * sProgName);
```

参数	描述
sProgName	显示于客户端窗口标题栏的标题。

GUI_VNC_SetSize()

描述

设置待传输至客户端的区域的尺寸。

原型

```
void GUI_VNC_SetSize(unsigned xSize, unsigned ySize);
```

参数	描述
xSize	待使用的 X 尺寸。
ySize	待使用的 Y 尺寸。

其他信息

默认情况下服务器使用层尺寸。传递至此函数的尺寸可以小于或大于实际显示尺寸。

GUI_VNC_X_StartServer()

描述

以给定的服务器索引启动 VNC 查看器，以在查看器中显示给定的层。
函数必须由用户编写，因为具体实现取决于所使用的 TCP/IP 堆栈和操作系统。
emWin 附有一个实现示例，位文件夹 Sample\GUI_X\GUI_VNC_X_StartServer.c 中。可以该示例为基础，针对其他系统进行改编。

原型

```
int GUI_VNC_X_StartServer(int LayerIndex, int ServerIndex);
```

参数	描述
LayerIndex	查看器将显示的层。
ServerIndex	服务器索引。

其他信息

在仿真系统和目标系统中启动 VNC 服务器的方法是一样的。两种情况下都需要调用此函数。仿真系统含有一个该函数的实现示例，硬件实现需由客户完成。

第 30 章

与时间和执行相关的函数

有些小工具以及演示代码需要使用时间相关函数。 emWin 图形库的其他部分不需要时间函数。演示代码大量使用 GUI_Delay() 程序，其作用是延迟指定的时间。时间单位称为点。

30.1 时间和执行 API

下表按字母顺序列出了可用的时间以及与执行相关的程序。这些程序的详细说明如下。

程序	描述
GUI_Delay()	延迟指定的时间。
GUI_Exec()	执行回调函数 （所有任务）。
GUI_Exec1()	执行一个回调函数 （仅一个任务）。
GUI_GetTime()	返回当前系统时间。

GUI_Delay()

描述

延迟指定的时间。

原型

```
void GUI_Delay(int Period);
```

参数	描述
Period	函数在返回值前需要等待的点数。

其他信息

时间单位 （点）通常为毫秒 （取决于 GUI_X_ 函数）。
GUI_Delay() 仅在给定时间内执行空闲函数。如果使用窗口管理器，则利用延迟时间来更新无效窗口 （通过执行 WM_Exec()）。
此函数将调用 GUI_X_Delay()。

GUI_Exec()

描述

执行回调函数 （通常为重新绘制窗口）。

原型

```
int GUI_Exec(void);
```

返回值

如果未执行任务，则返回值为 0。
如果执行了一个任务，则返回值为 1。

其他信息

此函数将自动重复调用 GUI_Exec1()，直至完成所有作业 – 实质是直至返回 0 值为止。
正常情况下，用户应用不需要调用此函数。它自动由 GUI_Delay() 调用。

GUI_Exec1()

描述

执行一个回调函数 （仅一个任务 —— 通常为重新绘制窗口）。

原型

```
int GUI_Exec1(void);
```

返回值

如果未执行任务，则返回值为 0。
如果执行了一个任务，则返回值为 1。

其他信息

该程序可能会重复调用，直到返回 0 为止，返回值为 0 则表示已完成所有任务。
此函数由 GUI_Exec() 自动调用。

GUI_GetTime()

描述

返回当前系统时间。

原型

```
int GUI_GetTime(void);
```

返回值

以点表示的当前系统时间。

其他信息

此函数将调用 GUI_X_GetTime()。

第 31 章

配置

在目标系统上使用 emWin 之前，必须对软件进行配置。配置意味着修改一般位于 Config（子）文件夹下的配置文件。我们已尽量简化配置过程，但仍然需要修改一些配置程序，以使系统能够正常工作。需要配置以下项目：

- emWin 使用的存储区域
- 用于绘图操作的显示驱动
- 使用的颜色转换程序
- 显示控制器初始化

本章将详细介绍 emWin 的配置。

31.1 需要配置的项目

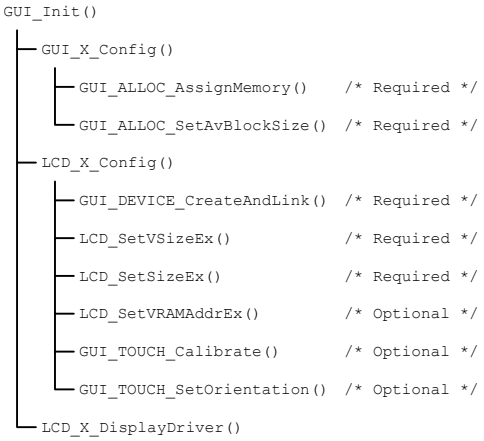
配置基本上可以分为两部分：GUI 配置和 LCD 配置。 GUI 配置表示对可用的功能、默认颜色、默认字体以及可用存储器进行配置。LCD 配置对硬件的依赖程度更高，需要定义将使用的显示器的物理尺寸、显示驱动和颜色转换程序。有关颜色转换程序的详细信息，请参阅 “颜色”（第 227 页）。
还有一部分是仿真系统的配置。但这不是目标硬件要求的配置，本章不作讨论。有关仿真系统配置的详细信息，请参阅 “模拟”（第 39 页）。

31.2 运行时间和编译时间的配置

需要配置 C 文件和 include 文件。头文件中的配置定于编译时，不能更改，而 C 文件中的配置则可在运行时更改。这样就可以创建在很大程度上独立于配置的库，并可支持任何显示器和驱动。这要求将本章描述的配置程序作为应用的一部分，而不是库的一部分。

31.3 emWin 的初始化程序

示意图所示为初始化过程。如果要初始化 emWin，则应用只需调用 GUI_Init() 即可。下面说明的配置程序将在内部初始化过程中被调用。



GUI_X_Config()

在初始化过程开始时即调用，以确保将存储器分配给 emWin。在该程序中，必须调用 GUI_ALLOC_AssignMemory() 和 GUI_ALLOC_SetAvBlockSize()，以将一个存储块分配给 emWin，并设置存储块的平均尺寸。这两个函数将在本章后面部分介绍。

LCD_X_Config()

此函数在 GUI_X_Config() 之后立即调用。该程序的主要目的是创建一个显示驱动器件，并选择颜色转换程序。另外，它还负责设置显示尺寸。如果使用触摸屏，则也需要在此进行配置。

LCD_X_DisplayDriver()

在初始化过程稍后时，要调用 LCD_X_DisplayDriver() 函数。该函数由显示驱动直接调用。在初始化过程中，该程序的任务是使显示控制器开始工作。本章后面将对该程序进行详细说明。

31.4 运行时间配置

下表列出了 Config 子文件夹中可用的运行时间配置文件：

配置文件	目的
GUIConf.c	配置可用存储器。
LCDConf.c	配置显示尺寸、显示驱动和颜色转换程序。
SIMConf.c	配置仿真系统 （不在本章范围之内）。
GUI_X.c	配置时间程序。

31.4.1 定制 GUIConf.c

该模块的目的是向 emWin 提供 GUI_X_Config() 函数，该函数负责向存储器管理系统分配一个存储块。这需要对所用元件的存储器要求有所了解。另外一章 “性能与资源占用”中含有各个 emWin 模块的存储器要求的详细说明 （RAM 和 ROM）。
默认情况下，GUIConf.c位于Config（子）文件夹中，含有负责向emWin分配存储块的GUI_X_Config()程序。并不要求将其留在 GUIConf.c 文件中。 GUI_X_Config() 程序可以位于应用中的任何位置。

GUI_X_Config()

描述

对此函数的调用是初始化过程做的第一件事。此函数负责向 emWin 分配一个存储块。该存储块由内存管理系统进行管理。存储块必须能以 8 位、 16 位或 32 位模式存取。

原型

```
void GUI_X_Config(void);
```

其他信息

需要注意的是，应用不可以使用整个存储块，因为管理系统本身会占用该存储块的一小部分。每个存储块都需要大约 12 字节的管理空间。

31.4.1.1 用于 GUI_X_Config() 的 API 函数

下表列出了必须在 GUI_X_Config() 中调用的 API 函数：

程序	描述
GUI_ALLOC_AssignMemory()	为存储器管理系统分配一个存储块。
GUI_ALLOC_SetAvBlockSize()	设置存储块的平均尺寸。该区越大，可用的存储块数量越少。
GUITASK_SetMaxTask()	设置在启用多任务时可用于访问 emWin 的最大任务数。

GUI_ALLOC_AssignMemory()

描述

此函数将内存管理系统使用的唯一一个存储块分配给 emWin。通常，该函数应当从 GUI_X_Config() 中调用。

原型

```
void GUI_ALLOC_AssignMemory(void * p, U32 NumBytes);
```

参数	描述
p	指向将由 emWin 使用的存储块的指针。
NumBytes	存储块的尺寸 （单位：字节）。

其他信息

需要注意的是，应用不可以使用整个存储块，因为管理系统本身会占用该存储块的一小部分。

GUI_ALLOC_SetAvBlockSize()

描述

设置由存储器管理系统分配的存储块的平均尺寸。块尺寸会影响可用块的最大数量。例如，如果应用要使用含有大量条目的列表视图，则可将块平均尺寸设置为较小的值。另一方面，如果应用主要将存储器管理用于一些存储器设备或图像解压，则应把平均尺寸设为较大的值。建议范围为 32 至 1024。具体值取决于应用。
通常，该函数应当从 GUI_X_Config() 中调用。

原型

```
void GUI_ALLOC_SetAvBlockSize(U32 BlockSize);
```

参数	描述
BlockSize	块平均尺寸。

其他信息

块的平均尺寸用于计算可用存储块的最大数量：

块的最大数量 = 存储器尺寸 （单位：字节） /(BlockSize + sizeof(BLOCK_STRUCT))

BlockStruct 指一种内部结构，其尺寸取决于 GUI_DEBUG_LEVEL。如果大于 0，则尺寸为 12 字节，否则为 8 字节。需要注意的是，结构尺寸同样取决于所用的编译器。

GUITASK_SetMaxTask()

描述

设置在启用多任务时可用于访问 emWin 的最大任务数。

原型

```
void GUITASK_SetMaxTask(int MaxTask);
```

参数	描述
MaxTask	访问 emWin 的最大任务数。

其他信息

此函数应从 GUI_X_Config() 中调用。使用预编译库时，需要使用该函数。否则，可以定义 wise GUI_MAXTASK。更多详细信息，请参阅 “GUI_MAXTASK”（第 283 页）。

31.4.2 定制 LCDConf.c

本模块的目的是为 emWin 提供必要的显示配置程序和显示驱动使用的回调函数。这些函数是：

程序	描述
LCD_X_Config()	配置程序，用于创建显示驱动器件，设置颜色转换程序和显示尺寸。
LCD_X_DisplayDriver()	由显示驱动调用的回调程序，用于使显示控制器开始工作。

LCD_X_Config()

描述

如上表所述，该程序负责创建一个显示驱动器件，设置正确的颜色转换程序，并配置物理显示尺寸。

原型

```
void LCD_X_Config(void);
```

其他信息

取决于使用的显示驱动，也可能需要设置视频 RAM 地址，初始化定制调色板等。有关任何其他要求的信息，请参见 “显示驱动详细描述”（第 829 页）。本章后面将列出和说明本程序可以使用的配置函数。

示例

以下示例展示了一种典型的实现实例：

```
//
// Set display driver and color conversion for 1st layer
//
GUI_DEVICE_CreateAndLink(GUIDRV_LIN_16, GUICC_565, 0, 0);
//
// Display driver configuration, required for Lin-driver
//
LCD_SetSizeEx    (0, 320, 240);
LCD_SetVSizeEx   (0, 320, 240);
LCD_SetVRAMAddrEx(0, (void *)0x200000);
```

LCD_X_DisplayDriver()

描述

这是显示驱动的回调函数。可以实现多种目的。与初始化过程相关的只有少数几个，实际上指显示控制器的初始化和视频 RAM 地址的设置。

原型

```
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * pData);
```

参数	描述
LayerIndex	基数为零的层索引。
Cmd	（参见下表）
pData	指向一种取决于 Cmd 的数据结构类型的指针。

元素 Cmd 在初始化过程中允许的值	
LCD_X_INITCONTROLLER	显示控制器需要初始化。通常情况下，收到该命令后，应调用用于初始化显示控制器寄存器的初始化程序。 pData 为 NULL
LCD_X_SETVRAMADDR	pData 指向的数据结构的 pVRAM 元素指向视频 RAM 的起始地址。该地址一般写入显示控制器的视频 RAM 基地址寄存器。 pData 指向一种 LCD_X_SETVRAMADDR_INFO 结构。

LCD_X_SETVRAMADDR 的元素：

数据类型	元素	描述
void *	pVRAM	指向视频 RAM 的起始地址的指针。

返回值

如果发生错误，程序的返回值为 -2；如果函数未处理命令，则返回值为 -1；如果命令已成功执行，则返回值为 0。

其他信息

若要详细了解显示驱动传递给程序的命令，请参见 “显示驱动”（第 813 页）。

示例

Sample\LCDConf\ 文件夹含有该程序的大量实现实例，可以其为基础针对应用进行改编。

31.4.2.1 用于 LCD_X_Config() 的 API 函数

下表列出了 LCD_X_Config() 中可用的配置 API 函数：

程序	描述
GUI_DEVICE_CreateAndLink()	创建显示驱动器件，并关联将使用的颜色转换程序。
GUI_TOUCH_SetOrientation()	设置触摸屏方向。仅当使用触摸屏且不以默认方向运行时，才需要此程序。
GUI_TOUCH_Calibrate()	校准触摸屏。
LCD_SetLUTEx()	用给定的调色板初始化查找表。仅当需要使用定制调色板时才需要此函数。
LCD_SetSizeEx()	用于设置显示器的物理尺寸。
LCD_SetVRAMAddrEx()	设置视频 RAM 的地址。仅当使用配备线性映射视频 RAM 的显示驱动时才需要此函数。
LCD_SetVSizeEx()	仅当虚拟显示尺寸与物理尺寸不同时才需要此函数。

有关 LCD_ 的详细信息，请参见 “显示驱动”（第 813 页）。
有关 GUI_TOUCH_ 的详细信息，请参见 “触摸屏驱动”（第 763 页）。

GUI_DEVICE_CreateAndLink()

描述

该程序创建显示驱动器件，设置用于存取显示器的颜色转换程序，并将驱动器件连接到给定层的器件列表。 LCD_X_Config() 在 GUI_X_Config() 之后立即调用。这样可以确保存储器配置已完成，且驱动可以分配存储器。
显示驱动器件需要的存储器大约是 50 字节 + 驱动的具体存储器。有关各种显示驱动对存储器的详细要求，请参见 “显示驱动”（第 813 页）。

原型

```
GUI_DEVICE * GUI_DEVICE_CreateAndLink(const GUI_DEVICE_API * pDeviceAPI,
                                      const LCD_API_COLOR_CONV * pColorConvAPI,
                                        U16 Flags, int LayerIndex);
```

参数	描述
pDeviceAPI	指向待使用的显示驱动的指针。“显示驱动”一章以表格列出了可用的显示驱动。
pColorConvAPI	指向待使用的颜色转换程序的指针。“颜色”一章以表格列出了可用的颜色转换程序。
Flags	应为零。
LayerIndex	应该由驱动管理的层。

返回值

若成功，返回指向创建的器件对象的指针，否则，返回 NULL。

其他信息

需要注意的是，某些情况下，所使用的驱动同时也决定了显示方向。具体因驱动而异。有关存储器管理器的详细信息，请参阅 “显示驱动”（第 813 页）。

31.4.3 定制 GUI_X.c

此文件含有时间程序、调试程序和核心内部程序：

31.4.3.1 时间程序

GUI_X_Delay()

描述

在指定时间后 （单位：毫秒）返回。

原型

```
void GUI_X_Delay(int Period)
```

参数	描述
Period	以毫秒表示的时间。

GUI_X_ExecIdle()

描述

仅从窗口管理吕的非阻塞函数中调用。

原型

```
void GUI_X_ExecIdle(void);
```

其他信息

在不再需要处理消息时调用。这种情况下， GUI 已更新。

GUI_X_GetTime()

描述

由 GUI_GetTime 用于返回当前系统时间 （单位：毫秒）。

原型

```
int GUI_X_GetTime(void)
```

返回值

以毫秒表示的当前系统时间，类型为整数。

31.4.3.2 调试程序

GUI_X_ErrorOut()、 GUI_X_Warn()、 GUI_X_Log()

描述

如果发现问题 （错误）或潜在问题， emWin 将以较高调试等级的调试信息调用这些程序。这些程序可以为空，并不是 emWin 正常工作所必须的。在目标系统中，发布 （生产）版本一般不要求这些程序，因为生产版本通常使用较低的调试等级。
如果 (GUI_DEBUG_LEVEL >= 3)，用 GUI_X_ErrorOut() 输出致命错误
如果 (GUI_DEBUG_LEVEL >= 4)，用 GUI_X_Warn() 输出警告
如果 (GUI_DEBUG_LEVEL >= 5)，用 GUI_X_Log() 输出消息

原型

```
void GUI_X_ErrorOut(const char * s);
void GUI_X_Warn(const char * s);
void GUI_X_Log(const char * s);
```

参数	描述
s	指向待发送的字符串的指针。

其他信息

emWin 调用该程序，以便传送错误消息或警告，如果启用日志功能，则需要使用该程序。 GUI 根据配置宏 GUI_DEBUG_LEVEL 调用该函数。下表列出了 GUI_DEBUG_LEVEL 允许的值：

值	符号名称	描述
0	GUI_DEBUG_LEVEL_NOCHECK	未执行运行时检验。
1	GUI_DEBUG_LEVEL_CHECK_PARA	已执行参数检验以避免故障。 （为目标系统的默认值）
2	GUI_DEBUG_LEVEL_CHECK_ALL	已执行参数检验和一致性检验。
3	GUI_DEBUG_LEVEL_LOG_ERRORS	已记录错误。
4	GUI_DEBUG_LEVEL_LOG_WARNINGS	已记录错误和警告。 （为计算机仿真系统的默认值）
5	GUI_DEBUG_LEVEL_LOG_ALL	已记录错误、警告和消息。

31.4.3.3 内核接口例程

有关这些程序的详细描述见 “执行模型：单任务 / 多任务”。

31.5 编译时间配置

下表列出了 Config 子文件夹中的可用编译时配置文件：

配置文件	目的
GUIConf.h	配置可用功能、层数、默认字体和默认颜色。
LCDConf.h	配置使用的显示驱动。

31.5.1 定制 GUIConf.h

如上所述，该文件含有对可用功能和默认字体的配置。每份 emWin 都配备了 GUIConf.h 文件，其中含有基本的配置，可以其为基础针对具体应用改编。

31.5.1.1 配置 emWin 的可用功能

下表列出了可用的配置宏：

类型	宏	默认值	描述
B	GUI_OS	0	激活后可以启用 emWin 的多任务支持（请参见“执行模型：单任务 / 多任务”（第 277 页））。
B	GUI_SUPPORT_CURSOR	（见说明）	默认情况下，如果已启用 GUI_SUPPORT_TOUCH 或 GUI_SUPPORT_MOUSE，则会启用游标。如果要显示游标而不启用这些选项之一，则应将其设为 1。
B	GUI_SUPPORT_MEMDEV	0	启用可选存储器件支持。
B	GUI_SUPPORT_MOUSE	0	启用可选鼠标支持。
B	GUI_SUPPORT_ROTATION	1	启用文字旋转支持。
B	GUI_SUPPORT_TOUCH	0	启用可选触摸屏支持。
B	GUI_WINSUPPORT	0	启用可选窗口管理器支持。

31.5.1.2 默认字体和默认颜色配置

下表列出了可用的配置宏：

类型	宏	默认值	描述
N	GUI_DEFAULT_BKCOLOR	GUI_BLACK	定义默认的背景颜色。
N	GUI_DEFAULT_COLOR	GUI_WHITE	定义默认的前景颜色。
S	GUI_DEFAULT_FONT	&GUI_Font6x8	定义默认情况下在 GUI_Init() 之后使用的字体。如果不使用默认字体，则可改成不同的默认值，因为默认字体用代码表示，因此始终都会关联。请参见 GUI_SetDefaultFont()，可用于在运行时配置默认字体。

也可配置作为可选窗口管理器一部分的小工具的默认颜色和字体。有关详细信息，请参阅 “窗口对象（小工具）”（第 345 页）。

31.5.1.3 高级 GUI 配置选项

下表列出了可用的配置宏：

类型	宏	默认值	描述
S	GUI_DEBUG_LEVEL	1 （目标系统） 4 （仿真系统）	定义调试等级，决定了 emWin 执行的检验数 （置位数），以及是否输出调试错误、警告和消息。 调试等级越高，产生的代码越大。
N	GUI_MAXTASK	4	定义在启用多任务时可用于访问 emWin 的最大任务数 （请参见 “执行模型：单任务 / 多任务”（第 277 页））。
F	GUI_MEMCPY	---	该宏允许取代 memcpy 函数。
F	GUI_MEMSET	---	取代 GUI 的 memset 函数。
N	GUI_NUM_LAYERS	1	定义可用层 / 显示器的最大数量。
B	GUI_TRIAL_VERSION	0	将编译器的输出标为评估版。
B	GUI_WINSUPPORT	0	启用可选窗口管理器支持。
N	GUI_PID_BUFFER_SIZE	5	输入缓冲器管理的 PID 事件的最大数量。
N	GUI_KEY_BUFFER_SIZE	10	输入缓冲器管理的键事件的最大数量。

GUI_MEMCPY

该宏允许取代 GUI 的 memcpy 函数。在许多系统中， memcpy 会花费大量的时间，因为编译器生产商并未对其进行优化。 emWin 含有一个替代 memcpy 程序，针对 32 位 CPU 进行过优化。在许多系统中，该程序能以比默认 memcpy 程序更快的速度生成代码。然而，这仍然是一种通用型 C 程序，在许多系统中可以由更快的代码所取代。其典型方法是使用不同的 C 程序 （针对具体 CPU 进行过更好的优化），或者以汇编语言写一个程序。
若要使用优化过的 emWin 程序，将在 GUIConf.h 文件中添加以下定义语句：

```
#define GUI_MEMCPY(pSrc, pDest, NumBytes) GUI__memcpy(pSrc, pDest, NumBytes)
```

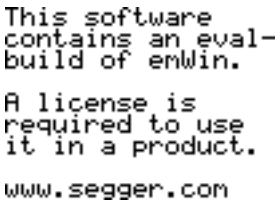
GUI_MEMSET

该宏允许取代 GUI 的 memset 函数。许多系统中，memset 会花费大量的时间，因为编译器生产商并未对其进行优化。我们尝试通过自己的 memset() 程序 GUI__memset 来解决这个问题。然而，这仍然是一种通用型 C 程序，在许多系统中可以由更快的代码所取代。其典型方法是使用不同的 C 程序（针对具体 CPU 进行过更好的优化），以汇编语言写一个程序，或者使用 DMA。
若希望使用自己的 memset 取代程序，请将定义添加到 GUIConf.h 文件中。

GUI_TRIAL_VERSION

该宏可用于将编译器的输出标记为评估版。如果将软件提供给第三方进行评估 （通常同时提供评估板），则需要定义。
需要注意的是，这样做需要一种特殊许可证；常用的许可证不允许以源代码或对象代码 （可重链）的形式分发 emWin。如有需要，请联系 sales@segger.com。

如果定义了 GUI_TRIAL_VERSION，则会在调用 GUI_Init() 时显示以下消息：



该消息始终显示在显示器的左上角，一般会持续 1 秒。该时间是通过调用 GUI_X_Delay(1000) 来实现的。如果激活该开关，则 emWin 的功能不受限制。

示例

```
#define GUI_TRIAL_VERSION 1
```

31.5.2 定制 LCDConf.h

该文件含有不需要在运行时更改的显示驱动的一般编译配置选项。可用配置选项取决于使用的显示驱动。有关位图转换器的详细信息，请参阅 “显示驱动”（第 813 页）。有关驱动的详细描述展示了各种显示驱动可用的配置选项。

31.6 请求可用存储器

以下函数允许在运行时控制存储器的使用，可用于避免存储器空间的浪费。

程序	描述
GUI_ALLOC_GetNumFreeBytes()	返回实际空闲的字节数。
GUI_ALLOC_GetNumUsedBytes()	返回应用实际使用的字节数。

GUI_ALLOC_GetNumFreeBytes()

描述

此函数返回可用于 emWin 函数的字节数。

原型

```
I32 GUI_ALLOC_GetNumFreeBytes(void);
```

返回值

空闲字节数。

GUI_ALLOC_GetNumUsedBytes()

描述

该函数返回 emWin 函数已经使用的字节数。

原型

```
I32 GUI_ALLOC_GetNumUsedBytes(void);
```

返回值

已使用的字节数。

第 32 章

性能与资源占用

高性能和低资源占用始终是设计时考虑的一个重要因素。emWin 运行于 8/16/32 位 CPU 平台。根据所使用的模块，emWin 甚至可以支持 ROM 不到 64 Kb、RAM 不到 2 Kb 的单芯片系统。实际性能和资源占用取决于多种因素（CPU、编译器、存储器模型、优化、配置、LCD 控制器的接口等）。本章含有关于典型系统中资源占用的基准和相关信息，用于对多数目标系统进行充分的估算。

32.1 性能

本节将展示不同目标系统上的驱动基准，以及绘图操作的性能值。

32.1.1 驱动基准

我们通过基准测试来测量显示驱动在现有目标系统上的速度。虽然该测试并不完整，但可以大概展示常用操作在不同目标系统上所需要的时间。

配置和性能表

CPU	LCD 控制器（驱动）	bpp	基准测试 1 填充	基准测试 2 小字体	基准测试 3 大字体	基准测试 4 位图 1bpp	基准测试 5 位图 2bpp	基准测试 6 位图 4bpp	基准测试 7 位图 8bpp	基准测试 8 DDP 位图
V850SB1 (20MHz)	S1D13806 (1300)	8	16.7M	339K	1.59M	1.52M	240K	459K	83K	1.25M
V850SB1 (20MHz)	S1D13806 (1300)	16	8.33M	326K	1.45M	1.49M	391K	388K	214K	806K
ARM720T (50MHz)	（内部） (3200)	16	7.14M	581K	1.85M	1.96M	694K	645K	410K	2.94M
ARM926EJ-S (200MHz)	（内部） (3200)	16	123M	3.79M	5.21M	7.59M	2.27M	2.21M	1.77M	15.2M

基准测试 1：填充

填充速度基准。以不同颜色填充一个大小为 64*64 像素的区域。

基准测试 2：小字体

小字符输出速度基准。以小字符文本填充一个大小为 60*64 像素的区域。

基准测试 3：大字体

大字符输出速度基准。以大字符文本填充一个大小为 65*48 像素的区域。

基准测试 4：位图 1bpp

1bbp 位图速度基准。以一个 1bbp 位图填充一个大小为 58*8 像素的区域。

基准测试 5：位图 2bpp

2bbp 位图速度基准。以一个 2bbp 位图填充一个大小为 32*11 像素的区域。

基准测试 6：位图 4bpp

4bbp 位图速度基准。以一个 4bbp 位图填充一个大小为 32*11 像素的区域。

基准测试 7：位图 8bpp

8bbp 位图速度基准。以一个 8bbp 位图填充一个大小为 32*11 像素的区域。

基准测试 8：因器件而异的位图， 8 或 16 bpp

8bpp 或 16bpp 位图速度基准。以一个位图填充一个大小为 64*8 像素的区域。测试用位图的颜色深度取决于配置。如果配置 <= 8bpp，则使用 8 bpp 位图； 16bpp 配置则使用 16 bpp 位图。

32.1.2 绘图性能

下表旨在展示 emWin 支持的不同图像格式的绘图性能。下表中的测量值是基于以下平台得到的结果：ARM922T CPU（夏普 LH7A404），时钟频率 200MHz，显示颜色深度 15 bpp（固定调色板 = 555），采用 LCDLin 驱动。

图像格式	百万像素 / 秒
内部位图格式：1bpp C 文件	17.186
内部位图格式：4bpp C 文件	3.897
内部位图格式：8bpp C 文件	4.017
内部位图格式：8bpp C 文件，无调色板	4.478
内部位图格式：16bpp C 文件，高彩色 555	13.363
内部位图格式：16bpp C 文件，高彩色 565	1.336
内部位图格式：24bpp C 文件，全彩色 888	1.671
内部位图格式：RLE4 C 文件	6.144
内部位图格式：RLE8 C 文件	6.806
内部位图格式：RLE16 C 文件	3.740
BMP 文件 8bpp	4.115
BMP 文件 16bpp	1.134
BMP 文件 24bpp	1.544
BMP 文件 32bpp	1.525
BMP 文件 RLE4	6.998
BMP 文件 RLE8	6.345
GIF 文件	1.285
JPEG 文件，灰色	0.516
JPEG 文件，灰色，渐近	0.438
JPEG 文件， H1V1	0.402
JPEG 文件， H1V1，渐近	0.280
JPEG 文件， H2V2	0.602
JPEG 文件， H2V2，渐近	0.431

32.2 存储器要求

emWin 的操作区存在较大差异，主要取决于所使用的应用和功能。以下各节将列出不同模块以及示例应用的存储器要求。 GUI 元件对存储器要求是基于以下系统测量的结果：

ARM7、 IAR Embedded Workbench V4.42A、 Thumb 模式、尺寸优化

32.2.1 GUI 元件的存储器要求

下表所示为 emWin 主要元件对存储器的要求。这些值在很大程度上取决于编译器选项、编译器版本和使用的 CPU。请注意，表中列出的值都是各个模块的基本函数的要求，还有几个可用的其他函数未在表中列出：

元件	ROM	RAM	描述
窗口管理器	+ 6.2 Kb	+ 2.5 Kb	“Hello world”应用在使用窗口管理器时的额外存储器要求。测量用的GUI_ALLOC_SIZE配置值为2000。
存储器设备	+ 4.7 Kb	+ 7 Kb	“Hello world”应用在使用存储器设备时的额外存储器要求。测量用的GUI_ALLOC_SIZE配置值为8000。
抗锯齿	+ 4.5 Kb	+ 2 * LCD_XSIZE	抗锯齿软件的额外存储器要求。
驱动	+ 2 - 8 Kbytes	20	驱动的存储器要求取决于所配置的驱动和是否使用数据缓存。使用数据缓存时，驱动要求更多的 RAM。有关详细信息，请参阅 “显示驱动”（第 813 页）。
多层	+ 2 - 8 Kbytes	-	如果使用多层或多显示配置，则每个额外层都需要额外的存储器，因为每个层都需要自己的驱动。
内核	5.2 Kb	80 字节	典型 “Hello world”应用在不使用额外软件项时的存储器要求。
内核 / JPEG	12 Kb	38 Kb	用于绘制 JPEG 文件的基本程序。
内核 / GIF	3.3 Kb	17 Kb	用于绘制 GIF 文件的基本程序。
内核 / Sprite	4.7 Kb	16 字节	用于绘制 sprite 和游标的程序。
内核 / 字体	（见描述）	-	有关 emWin 随附标准字体对 ROM 的详细要求，请参见 “字体”（第 169 页）。
小工具	4.5 Kb	-	这大致是小工具对 ROM 的基本要求，具体取决于小工具所使用的各内核函数。
小工具 / BUTTON	1 Kb	40 字节	*1
小工具 / CHECKBOX	1 Kb	52 字节	*1
小工具 / DROPDOWN	1.8 Kb	52 字节	*1
小工具 / EDIT	2.2 Kb	28 字节	*1
小工具 / FRAMEWIN	2.2 Kb	12 字节	*1
小工具 / GRAPH	2.9 Kb	48 字节	*1
小工具 / GRAPH_DATA_XY	0.7 Kb	-	*1
小工具 / GRAPH_DATA_YT	0.6 Kb	-	*1
小工具 / HEADER	2.8 Kb	32 字节	*1
小工具 / LISTBOX	3.7 Kb	56 字节	*1
小工具 / LISTVIEW	3.6 Kb	44 字节	*1
小工具 / MENU	5.7 Kb	52 字节	*1
小工具 / MULTIEDIT	7.1 Kb	16 字节	*1
小工具 / MULTIPAGE	3.9 Kb	32 字节	*1
小工具 / PROGBAR	1.3 Kb	20 字节	*1
小工具 / RADIOBUTTON	1.4 Kb	32 字节	*1
小工具 / SCROLLBAR	2 Kb	14 字节	*1
小工具 / SLIDER	1.3 Kb	16 字节	*1
小工具 / TEXT	0.4 Kb	16 字节	*1

*1. 表中所列小工具的存储器要求包括创建和绘制小工具需要的基本程序。根据具体的小工具，还有几种额外的可用函数未在表中列出。

32.2.2 堆栈要求

基本堆栈要求大约是 600 字节。如果使用窗口管理器，应再加 600 字节。如果有存储器设备，则建议再加 200 字节。需要注意的是，堆栈要求同样取决于应用、所用编译器和 CPU。

32.3 示例应用的存储器要求

本节将展示一些示例应用的要求。下表总结了相关存储器要求。若无特别说明，所列值的单位为字节:

示例	GUI 内核	字体	应用	启动代码	库	总计	GUI 内核	应用	堆栈	总计
	ROM						RAM			
Hello world	5.9 kB	1.8 kB	38	0.3 kB	0.1 kB	**8.1 kB**	62	-	272	**334**
窗口应用	43 kB	12.5 kB	2.7 kB	0.3 kB	1.5 kB	**60 kB**	5.2 kB	40	1.4 kB	**6.6 kB**

有关示例的详细情况，请参见以下各节。

第 33 章

支持

本章将介绍在发生任何问题的处理办法。问题可能发生在工具链上、硬件上，也可能是 GUI 函数的使用或性能方面，本章将介绍与 emWin 客户支持部门联系的方式。

33.1 工具链 （编译器、链接器）的问题

本节将介绍工具链可能出现的一些问题。内容涉及发生问题时的处理方式以及必要时与 emWin 客户支持部门的联系方式。

33.1.1 编译器故障

这是工具链 （编译器）的问题，不是 emWin 的问题。如果工具链中的任一工具发生故障，请与编译器支持人员联系：
“工具内部错误，请联系支持人员。”

33.1.2 编译器警告

emWin 的代码已在不同的目标系统和不同编译器下进行过测试。我们投入了大量时间来完善代码质量，并竭力避免出现编译器警告。但各种编译器在警告方面的敏感度是不一样的。因此，我们无法避免未知工具出现编译器警告。

不应看到的警告

这种警告是不应该出现的：
"Function has no prototype"（函数无原型）
"Incompatible pointer types"（指针类型不兼容）
"Variable used without having been initialized"（所用变量未初始化）
"Illegal redefinition of macro"（宏被非法重新定义）

可能会看到的警告

类似下列警告应忽略：
"Integer conversion, may loose significant bits"（整数转换，可能解除有效位）
"Statement not reached"（语句未达到）
"Descriptionless statements were deleted during op-timization"（无描述语句已在优化过程中删除）
"Condition is always true/false"（条件始终为真 / 假）
"Unreachable code"（代码达不到）

多数编译器提供了用于屏蔽选定警告的办法。

“参数未使用”警告

根据所使用的配置，有时并非函数的所有参数都会用到。为了避免出现这个问题相关的编译器警告，您可以在 GUIConf.h 文件中定义 GUI_USE_PARA 宏，如下例所示：

```
#define GUI_USE_PARA(para) para=para;
```

emWin 利用该宏来在必要时避免出现这类警告。

33.1.3 编译器错误

emWin 假设所用编译器兼容 ANSI C。编译器应至少支持下列标准之一：

- ISO/IEC/ANSI 9899:1990 (C90) with support for C++ style comments (//)
- ISO/IEC 9899:1999 (C99)
- ISO/IEC 14882:1998 (C++)

对函数指针调用语句中自变量数量的限制

但是，有些编译器并不100%兼容ANSI C，比如，有的对函数指针调用语句中的自变量数量有限制：

```
typedef int tFunc(int a, int b, int c, int d, int e,
                  int f, int g, int h, int i, int j);

static int _Func(int a, int b, int c, int d, int e,
                 int f, int g, int h, int i, int j) {
  return a + b + c + d + e + f + g + h;
}
```



```
static void _Test(void) {
  int Result;
  tFunc * pFunc;
  pFunc = _Func;
  Result = pFunc(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
}
```

如果上例无法编译，则只能使用 emWin 核心版本。 emWin 的额外工具，如窗口管理器、存储器件模块等，有时需要将最多 10 个参数传递给函数指针调用语句。 emWin 的核心内在一个函数指针调用语句中最多只需要2个参数。但是，如果您的编译器在一个函数指针调用语句中只支持一个自变量，则也可使用 emWin。如果这样，会有些函数不可用，如旋转文本或 UTF-8 编码等。有关在这种情况下如何配置 emWin 的详细情况，请查看 “高级配置”一章。

33.1.4 链接器问题

外部符号未定义

如果链接器显示错误消息 “Undefined external symbols...”（外部符号未定义 ……），则请核实下列文件是否已包含于项目中或库中：

- emWin 随附的所有资源文件
- 使用简单的总线接口时：Sample\LCD_X 文件夹下的硬件程序之一？有关详细信息，请参阅“配置”（第 905 页）。
- Sample\GUI_X 文件夹下的文件之一？有关详细信息，请参阅 “配置”（第 905 页）。

可执行文件太大

有些链接器只能仅链接项目使用的模块 / 函数。结果会导致可执行文件中存在大量未使用的代码。这种情况下，可以使用库。有关如何创建 emWin 库的详细信息，请参见 “入门指南”（第 31 页）。

33.2 硬件 / 驱动问题

如果您的工具都很正常，但显示器却不工作，可以尝试从以下方面解决问题。

堆栈尺寸太小？

确保配置了足够的堆栈。遗憾的是，我们无法评估配置和编译器到底需要使用多少堆栈。而且所需要的堆栈尺寸在很大程度上取决于应用。

显示器初始化错误？

请检查控制器初始化是否根据您的具体需求而调整过。

显示接口配置错误？

开始使用 emWin 时，而且显示器不显示时，您应该使用一个示波器来测量与显示器 / 控制器连接的引脚。如果有问题，请检查以下各项：

- 如果使用的是简单的总线接口：很可能是未正确配置硬件程序。如有可能，请使用仿真器按步运行这些程序。
- 如果使用的是全总线接口：很可能是未正确配置寄存器 / 存储器存取。

33.3 API 函数的问题

如果工具链和硬件都没有问题，但 API 函数表现异常，请根据 “联系支持”（第 927 页）描述的方法，编写一个小的示例。这样做有助于重现问题并快速解决问题。

33.4 性能问题

如果 emWin 出现任何性能问题，则须确定是软件的哪部分导致了问题。

是驱动导致问题吗？

为了确定问题根源，首先是编写一个小测试程序，用于执行一些测试代码，并测量出执行该代码所用的时间。应从上面描述过的 ProblemReport.c 文件着手。为了测量实际硬件驱动使用的时间，每份 emWin 都含有驱动 LCDNull.c。如果不需要向硬件输出，则可使用该驱动。为了激活驱动，要在 LCDConf.h 文件中定义 LCD_CONTROLLER 宏，如下所示：

```
#define LCD_CONTROLLER -2
```

实际驱动与 LCDNull 驱动所使用的时间之间的差异展现了实际硬件驱动耗费的执行时间。

驱动未优化？

如果使用实际驱动与使用 LCDNull 驱动之间存在显著差异，则问题的原因可能是驱动模式未优化。如果使用下列宏之一：LCD_MIRROR_X、LCD_MIRROR_Y、LCD_SWAP_XY 或 LCD_CACHE，则驱动可能未针对配置模式进行优化。这种情况下，请与我们的支持部门联系，我们应该可以优化代码。

显示控制器太慢？

同样请参见第 “显示驱动”（第 813 页）。如果使用慢速显示控制器 （如爱普生 SED1335），则该章可能回答了驱动为什么慢这个问题。

33.5 联系支持

如果需要与 emWin 支持部门联系，请发送以下信息：

- 可以把问题的详细描述以注释形式写在示例代码中。
- 配置文件 GUIConf.h。
- 配置文件 LCDConf.h。
- 可以在仿真中配置的示例源文件，无需任何其他文件，如下文所述。
- 如果工具链有任何问题，请同时发送编译器 / 链接器的错误消息。
- 如果硬件 / 驱动有任何问题，而且使用的是简单的总线接口，请同时发送包含配置在内的硬件程序。

问题报告

可基于以下文件创建问题报告。同时填写 CPU、所用工具链和问题描述。其位置是：Sample\Tutorial\ProblemReport.c:

```
/*********************************************************************
*                 SEGGER Microcontroller GmbH & Co. KG               *
*         Solutions for real time microcontroller applications       *
*                                                                    *
*                         emWin problem report                       *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File                :ProblemReport.c
CPU                 :
Compiler/Tool chain :
Problem description :
----------------------------------------------------------------------
*/

#include "GUI.h"
/*  Add further GUI header files here as required.*/


/*********************************************************************
*
*       Static code
*
**********************************************************************
*
* Please insert helper functions here if required.
*/

/*********************************************************************
*
*       MainTask
*/
void MainTask(void) {
  GUI_Init();
  /*
    To do:Insert the code here which demonstrates the problem.
  */
  while (1);   /* Make sure program does not terminate */
}
```

33.6 常见问题解答

Q: 我使用一种不同的 LCD 控制器。还能使用 emWin 吗？
A: 能。硬件存取由驱动模块实现，完全独立于 GUI 的其他部分。可以针对任何控制器 （存储器映射或总线驱动）轻松写出相应的驱动。请与我们联系。

Q: emWin 应使用什么 CPU？
A: emWin 可以使用任何 CPU（或 MPU），只要有 C 编译器即可。当然，其在 16/32 位 CPU 上的速度比 8 位 CPU 快。

Q: emWin 是否很灵活，可以完成我希望在应用中实现的功能？
A: emWin 很灵活，应该可以支持任何应用。如果由于某种原因，您认为情况并非如此，请与我们联系。信不信，我们同时提供源代码。

Q: emWin 是否采用多任务环境？
A: 是的，其采用多任务内核设计。

索引

C

D

E

F

G

H

J

K

L

M

N

P

Q

R

S

T

W

X

Y

Z