miniStudio 新控件集（mGNCS）编程指南

版本 1.0 修订号 2

适用于 mGNCS Ver 1.0.x

北京飞漫软件技术有限公司

2010 年 5 月

目录

第一部分第一章 简介

新控件集介绍

在 miniStudio 的开发中，为实现可视化图形界面的设计，飞漫软件在 MiniGUI 现有接口基础上，开发了一套新的控件集。miniStudio 引入的新控件集是在原 MiniGUI 控件集基础上发展而来的，为与 MiniGUI 固有控件集（Intrinsic Control Set）区别，称为“新控件集（New Control Set，简称 mGNCS）”。

新控件集的特点如下：

- 采用面向对象编程思想，重新整理了控件之间的继承关系，用 C 语言实现了类似 C++ 类概念，对外提供 C 语言接口。这点上，miniStudio 引入的新控件集和 Gtk+ 的控件集类似。
- 对控件的接口和风格做了规范，使所有控件都能用统一的接口访问。
- 在 MiniGUI 3.0 外观渲染器的基础上，进一步扩展了外观渲染器的概念，新控件集中的每个控件都具有自己的外观渲染器，并按照控件的继承规则定义每个控件专有的渲染器接口，使其能够随着控件的变化而变化。
- 通过开发新控件集，我们同时解决了 MiniGUI 固有控件集在如下几个方面的不足：
  - 接口不统一，不规范；
  - 通过消息操作控件，代码维护困难，易于出错；
  - 固有控件的层次关系不明，重复代码较多，且不方便定制和扩展控件功能；
  - 固有控件的绘制效率较低，在开发板上运行时闪烁现象比较严重。

在新控件集中，存在如下几种类：

- 控件类。这种类用于表示各种控件（如主窗口、静态框、按钮、列表框等），也就是 MiniGUI 应用程序中具有可视化特征的类。需要注意的是，mGNCS 中所说的“控件”，其概念比 MiniGUI 所述“控件”有更广泛的外延，同时包括了主窗口等对象，而 MiniGUI 所指的控件，仅用于子窗口。在 mGNCS 中，所有控件类都派生自 mWidget 类。
- 非控件类。这种类，用于维护类层次结构以及控件所使用的数据等，包括如下几种：
  - 超类，它是 mGNCS 类层次结构中的最基础类，所有其他类由这个类派生而来，即 mObject 类。
  - 一般性类，这种类用来实现数据源、数据绑定等机制，也用来维护控件所使用的数据，如 mItem 类。
  - 不可见组件类，这种类用来表示界面中所使用的定时器等不可见的组件，和控件类（相当于可见组件类）一起，被统称为组件类。

一个控件由以下几个部分组成：

- 对象和类：mGNCS 使用一个对象结构体和类结构体来表示一个类。其中类结构体主要定义了一组函数指针，来模拟 C++ 类的虚函数表，以实现继承和多态能力。对象结构体则定义了一个类实例相关的数据结构。
- 控件类名：由一个唯一的字符串来标识一个控件类，对应的 C 语言宏使用 NCSCTRL_ 作为前缀。
- 控件风格：同 MiniGUI 固有控件集的风格概念。风格在新控件集中只被用于那些能够在控件初始化时确定，在控件的整个生命周期都不会变化的属性中使用。
- 控件属性：新控件集将控件可以通过 get/set 方法控制的特性提取出来，并为每一项设置一个整数标识符，通过类结构体的 setProperty 和 getProperty 方法来访问和控制一个控件的行为和状态。
- 控件事件：合并了 MiniGUI 中消息和通知码的概念，通过设置一个特定事件的回调函数，可以用来响应控件的事件。
- 外观渲染器：在 mGNCS 中，每个控件的绘制都由专门的外观渲染器实现。新控件集默认提供了 classic（经典）、fashion（流行）、skin（皮肤）、flat（平板）四种外观渲染器。控件可以选择任意一种有效的渲染器，通过为该渲染器设置特定的属性（如颜色、尺寸等）来调整控件的外观。为了方便描述，本指南中所述外观渲染器均简称为渲染器（renderer）；为了和控件自身的渲染器相区别，MiniGUI 3.0 定义的渲染器被称为“全局渲染器（global renderer）”，将新控件集引入的渲染器称为“控件渲染器（control renderer）”。这两类渲染器的作用有如下不同：
    - 全局渲染器作用于主窗口、MiniGUI 固有控件集以及新控件集的系统组件（如边框、标题栏、滚动条等非客户区元素）。
    - 控件渲染器仅作用于新控件集各个控件的客户区绘制。

另外，新控件集定义的不可见组件拥有和控件类似的接口，目前实现了定时器，在以后版本中计划实现的组件有特效切换组件、程序逻辑控制组件等等。

新控件集主要配合 miniStudio 使用，也可以作为 MiniGUI 3.0 的一个组件而直接使用，且可以和固有控件集中的控件混合使用。

本指南的内容组织

本指南被划分为如下附录：

- 第一部分：介绍 mGNCS 的基本概念及体系结构。
- 第二部分：面向应用开发者，讲述如何使用 mGNCS 开发应用程序，并详细描述主要的 API 接口。
- 第三部分：面向新控件集开发者，讲述如何给予 mGNCS 开发新的控件、渲染器等。
- 附录：本文档的编写规范以及其他快速参考式的内容等。

第一部分第二章 体系架构及主要特性

整体结构

如前所述，新控件集使用面向对象编程思想重新设计了 MiniGUI 的控件集，类似 MFC 和 Win32 的关系，也类似 Gtk+ 和 Xlib 的关系。图 p1c2-1 给出了 mGNCS 的基础类及其关系。

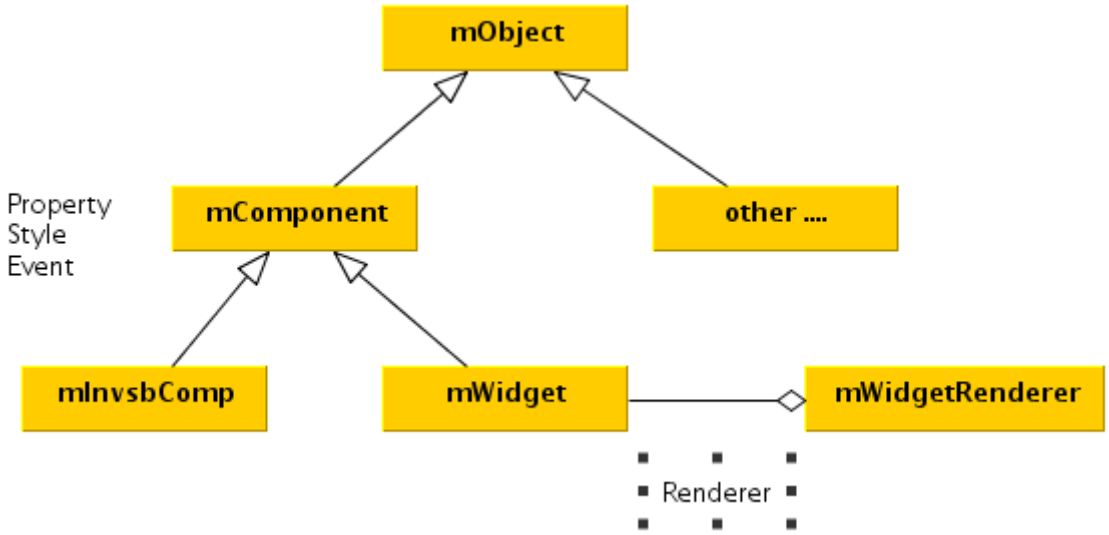


图 p1c2-1 mGNCS 中的基础类及其关系

如图所示，

- mObject 是整个 mGNCS 类关系中的最基础类（超类），所有对象从 mObject 派生而来。
- mComponent 是所有组件的基类；组件又分可见组件及不可见组件。可见组件就是我们常说的控件，而不可见组件的典型例子为定时器。mComponent 定义了组件类的基础组成部分，如风格、属性和事件等。
- mWidget 派生自 mComponent，是所有可见组件（即控件）的基类。
- mInvsbComp 派生自 mComponent，是所有不可见组件的基类。
- mWidget 通过聚合一个对应的渲染器（Renderer）接口，增加了渲染器这一新特性。

mGNCS 中，也存在若干直接派生自 mObject 的类，这些类属于通用性类，如用于管理列表型控件的列表项类等等。

总结而言，新控件集中存在如下几种类：

- 控件类。这种类用于表示各种控件（如主窗口、静态框、按钮、列表框等），也就是 MiniGUI 应用程序中具有可视化特征的类。需要注意的是，mGNCS 中所说的“控件”，其概念比 MiniGUI 所述“控

件”有更广泛的外延，同时包括了主窗口等对象，而 MiniGUI 所指的控件，仅用于子窗口。在 mGNCS 中，所有控件类都派生自 mWidget 类。

- 非控件类。这种类，用于维护类层次结构以及控件所使用的数据等，包括如下几种：
  - 超类，它是 mGNCS 类层次结构中的最基础类，所有其他类由这个类派生而来，即 mObject 类。
  - 一般性类，这种类用来实现数据源、数据绑定等机制，也用来维护控件所使用的数据，如 mItem 类。
  - 不可见组件类，这种类用来表示界面中所使用的定时器等不可见的组件，和控件类（相当于可见组件类）一起，被统称为组件类。

类层次图

下面给出了 mGNCS 目前主要的类及其层次关系：

- mObject
  - mItem
    - mListColumn
    - mItemManager
      - mListItem
  - mComponent
    - mInvsbComp
      - mTimer
    - mWidget
      - mStatic
        - mLEDLabel
        - mImage
        - mRect
        - mGroupBox
          - mButtonGroup
      - mButton
        - mCheckButton
          - mRadioButton
      - mPanel
        - mCombobox
        - mMainWnd
          - mDialogBox
      - mScrollWidget
        - mItemView
          - mListView
          - mIconView
          - mScrollView

        - mListbox
    - mContainer
        - mPage
- mProgressBar
- mPropSheet
- mSlider
    - mTrackbar
- mSpinner
    - mSpinBox
- mEdit
    - mSLEdit
    - mMLEdit
- mMonthCalendar

面向对象设计带来的好处

面向对象的设计给 mGNCS 带来了如下好处：

1. 简化编程，mGNCS 将常用的消息和通知处理抽象成了事件/处理器结构，从而大大简化了编程。
2. 规范化的控件风格、属性、事件、方法等，易于 miniStudio 等工具处理，从而实现界面设计的可视化。
3. 重构的控件类体系结构，对每个控件的功能做单一设计及处理，通过继承实现控件功能的扩展，从而简化了设计，提高了代码的可维护性以及可扩展性。

新控件集的其他主要特性

除了上述特性之外，mGNCS 还具有如下值得一提的特性：

外观渲染器

我们知道，MiniGUI 3.0 新增功能中最为重要的是外观渲染器（Look and Feel Renderer，LFRDR）。外观渲染器为用户提供了多种风格的主窗口和控件界面外观风格。应用程序在这几种风格的窗口界面之间进行切换非常容易，只要在创建窗口时传递不同的渲染器名称，就可以变换出不同风格的界面。

在 MiniGUI 3.0 外观渲染器的基础上，mGNCS 进一步扩展了外观渲染器的概念，新控件集中的每个控件都具有自己的外观渲染器，并按照控件的继承规则定义每个控件专有的渲染器接口，使其能够随着控件的变化而变化。 新控件集为每个控件默认提供了 classic（经典）、fashion（流行）、skin（皮肤）、flat（平板）等四种外观渲染器。控件可以选择任意一种有效的渲染器，通过为该渲染器设置特定的属性（如颜色、尺寸等）来调整控件的外观。为了和控件自身的渲染器相区别，MiniGUI 3.0 定义的渲染器被称为“全局渲

染器（global renderer）”，新控件集引入的渲染器称为“控件渲染器（control renderer）”。这两类渲染器的作用有如下不同：

- 全局渲染器作用于主窗口、MiniGUI 固有控件集以及新控件集的系统组件（如边框、标题栏、滚动条等非客户区元素）。
- 控件渲染器仅作用于新控件集各个控件的客户区绘制。

外观渲染器为控件外观的定制，提供了非常方便的实现方式。有关外观渲染器及相关接口的描述和使用，将在本指南第二部分第三章中做详细介绍。

事件监听机制

mGNCS 提供了一种类似 QT 的 signal 和 slot 的机制，能够把一个对象的事件链接到任意一个对象上。mGNCS 的事件监听链接机制提供了一种将事件发送者和事件关心者解耦的方式。它通过全局数据表，把发送者和接收者的关系对应起来。

在没有事件监听机制的情况下，我们通过编写事件的事件处理器来处理每个事件。有了事件监听机制，我们就可以在程序的任意地方建立事件和监听者之间的联系，从而解耦事件发生者和关心者之间的关系。

数据源及数据绑定

如果读者曾经使用 Visual Studio 开发过基于 MFC 的 C++ 应用程序的话，则一定记得这个工具提供一种机制，可以将对话框中的控件内容和给定的类成员变量绑定起来。在 VC 中，对话框控件的初始化值，可以从绑定的对话框成员变量中自动获得并设置好，而当对话框退出时，控件中的值又可以自动赋值给对应的对话框类成员变量。这就是 mGNCS 提供的数据源和数据绑定功能的思想来源。但是，mGNCS 提供的数据源和数据绑定功能更加强大。利用 mGNCS 的数据绑定功能，当 mGNCS 控件的值发生变化时，我们可以自动更新其他控件，或者将数据再次保存到期望的数据源中；通过 mGNCS 的数据源，我们可以定义不同格式的数据来源，如程序定义的字符串数组、文本文件、配置文件，甚至数据库等等，并使用这些数据自动填充到 mGNCS 控件中。

数据绑定和数据源是 mGNCS 提供给应用程序的两个非常重要的机制，这两个机制均有助于实现程序逻辑和它所处理的数据之间的分离，且便于类似 miniStudio 这样的可视化 GUI 设计工具来设计界面。

资源管理

mGNCS 中有一个相对独立的模块，即资源管理模块。这个模块主要配合 miniStudio 可视化设计工具使用。这个模块主要完成如下工作：

- 从 miniStudio 生成的资源包中装载界面描述数据，并生成对应的主窗口或对话框。
- 装载渲染器、图片、文本等其他界面相关的资源或者描述数据。

使用 miniStudio 开发工具的情况下，除了一些常用接口之外，应用程序基本上不需要直接调用资源管理模块的接口。有关资源管理模块接口的描述和使用，将在本指南第二部分第三章中做详细介绍。

第二部分第一章 快速入门

mGNCS 的 helloworld 程序

mGNCS 对 MiniGUI 的主窗口进行了包装，把它纳入到控件体系之内，所以，整个编程将变得非常的简单和易学。请看下面基于 mGNCS 的 helloworld 程序源代码：

清单 p2c1-1 helloworld.c

```
/*
** $Id$
**
** Listing P2C1.1
**
** helloworld.c: Sample program for mGNCS Programming Guide
**      The first mGNCS application.
**
** Copyright (C) 2009 Feynman Software.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// START_OF_INCS
#include <minigui/common.h>
#include <minigui/minigui.h>
```

```
#include <minigui/gdi.h>
#include <minigui/window.h>

#include <mgncs/mgncs.h>
// END_OF_INCS

// START_OF_HANDLERS
static void mymain_onPaint (mWidget *_this, HDC hdc, const CLIPRGN* inv)
{
        RECT rt;
        GetClientRect (_this->hwnd, &rt);
        DrawText (hdc, "Hello, world!", -1, &rt, DT_SINGLELINE|DT_CENTER|DT_VCENTER);
}

static BOOL mymain_onClose (mWidget* _this, int message)
{
        DestroyMainWindow (_this->hwnd);
        return TRUE;
}

static NCS_EVENT_HANDLER mymain_handlers [] = {
        {MSG_PAINT, mymain_onPaint},
        {MSG_CLOSE, mymain_onClose},
        {0, NULL}
```

```
};
// END_OF_HANDLERS

int MiniGUIMain (int argc, const char* argv[])
{
        MSG Msg;

        ncsInitialize ();

        mWidget* mymain = ncsCreateMainWindow (
        NCSCTRL_MAINWND, "Hello, world!",
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        1,
        0, 0, 300,200,
        HWND_DESKTOP,
        0, 0,
        NULL,
        NULL,
        mymain_handlers,
        0);

        // START_OF_MSGLOOP
        while (GetMessage (&Msg, mymain->hwnd)) {
```

```
            TranslateMessage (&Msg);
            DispatchMessage (&Msg);
        }
        // END_OF_MSGLOOP

        MainWindowThreadCleanup (mymain->hwnd);

        ncsUninitialize ();

        return 0;
}
```

该程序在屏幕上创建一个 300x200 的主窗口并显示“Hello, world!”：

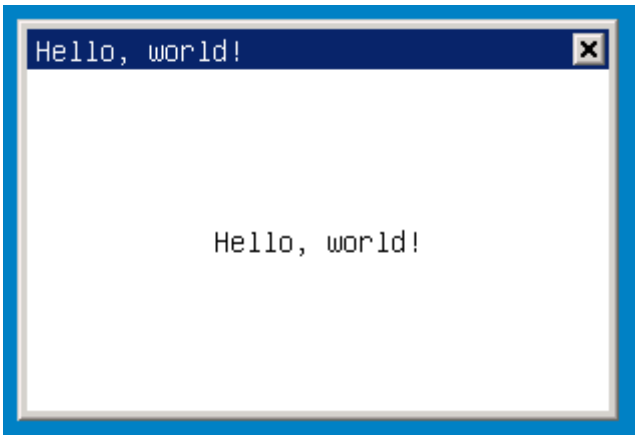


图 p1c1-1 helloworld 程序的输出

头文件

helloworld.c 除了要包含 MiniGUI 的四个头文件（<minigui/common.h>、<minigui/minigui.h>、<minigui/gdi.h> 和 <minigui/window.h>）外，还要包含自己特有的文件：

- <mgncs/mgncs.h>：mGNCS 头文件

所以，通常的情况下一个 mGNCS 程序的头文件为：

```
#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>

#include <mgncs/mgncs.h>
```

创建主窗口

和普通的 MiniGUI 程序一样，mGNCS 程序的入口点也是 MiniGUIMain ，但是，在使用 mGNCS 之前，必须先调用 ncsInitialize 来向 MiniGUI 注册 NCS 的控件：

```
ncsInitialize ();
```

之后，就可以通过调用 ncsCreateMainWindow 函数直接创建一个主窗口。该函数的参数比较多，但是创建默认窗口可以使用大量默认参数：

```
mWidget* mymain = ncsCreateMainWindow (
NCSCTRL_MAINWND,     // 主窗口的类名，NCS 控件类名宏以 NCSCTRL_ 开头
"Hello World!",      // 指定主窗口的标题文本
WS_CAPTION | WS_BORDER | WS_VISIBLE, // 指定窗口风格，创建一个有标题栏和边框的、且可见的窗口
WS_EX_NONE,          // 指定主窗口的扩展风格，用 0 或者 WS_EX_NONE 表示没有扩展风格
1,                   // 指定主窗口的标识值，任意 >0 的整数值
0, 0, 300, 200,      // 指定主窗口的位置和大小，依次是 x, y, width, height
HWND_DESKTOP,        // 指定主窗口的宿主，一般为 HWND_DESKTOP
0, 0,                // 主窗口的图标和菜单栏句柄，默认为空
NULL,                // 指定主窗口的初始属性信息，默认为空
NULL,                // 指定主窗口的初始渲染器信息，默认为空
```

```
mymain_handlers,     // 指定主窗口的事件处理器，见下节

0);                  // 指定主窗口的附加数据，这里为 0
```

上文中最重要的参数是事件处理器。

ncsCreateMainWindow 函数返回的是一个窗口对象指针（mWidget*），该对象被设置为控件基类的指针，以便用户使用。

mWidget 结构提供了一个 hwnd 数据成员，是和窗口对象关联的窗口句柄，用户可以直接使用该句柄调用对应的 MiniGUI API。

事件处理器

使窗口操作，就必须响应窗口的消息。NCS 提供了一个映射机制，将一个事件（消息或者通知码）和一个回调函数关联起来。不同事件的回调函数格式是不一样的，NCS 已经为每种事件事先定义好了接口，以方便使用。

多个映射可以放到一个 NCS_EVENT_HANDLER 数组中（该数组以 {0,NULL} 为结束），在创建窗口时传递给控件。

在该例子中，我们主要关注两个消息事件：

- MSG_PAINT
- MSG_CLOSE

这两个消息分别对应的事件处理器原型是：

```
/**

* \typedef void (*NCS_CB_ONPAINT)(mWidget*, HDC hdc, const PCLIPRGN clip_rgn);

* \brief the callback of event MSG_PAINT

*

* \param mWidget * sender pointer

* \param hdc the DC for painting

* \param clip_rgn invlidate region

*

*/
```

```
typedef void (*NCS_CB_ONPAINT)(mWidget*, HDC hdc, const PCLIPRGN clip_rgn);


/**
* \typedef  BOOL (*NCS_CB_ONCLOSE)(mWidget*);
* \brief the callback of event MSG_CLOSE
*
* \param mWidget* the sender pointer
*
* \return TRUE - about the message process, FALSE - continue default processing
*/
typedef BOOL (*NCS_CB_ONCLOSE)(mWidget*);
```

所以，我们定义这两个函数，并建立事件和事件处理器的映射数组：

```
// START_OF_HANDLERS
static void mymain_onPaint (mWidget *_this, HDC hdc, const CLIPRGN* inv)
{
        RECT rt;
        GetClientRect (_this->hwnd, &rt);
        DrawText (hdc, "Hello, world!", -1, &rt, DT_SINGLELINE|DT_CENTER|DT_VCENTER);
}


static BOOL mymain_onClose (mWidget* _this, int message)
{
        DestroyMainWindow (_this->hwnd);
        return TRUE;
```

```
}


static NCS_EVENT_HANDLER mymain_handlers [] = {
        {MSG_PAINT, mymain_onPaint},
        {MSG_CLOSE, mymain_onClose},
        {0, NULL}
};
// END_OF_HANDLERS
```

可以看到，同 MiniGUI 普通程序相比，mGNCS 避免了多层的 switch-case 嵌套语句的使用，让开发者免于记忆复杂的 WPARAM 和 LPARAM 含义和事件处理过程返回值的含义。

如果使用 miniStudio，上述代码可被自动生成，用户要做的，就是实现对应事件处理器（回调函数）。

消息循环

最后，要让程序运行起来，需要一个消息循环，同普通的 MiniGUI 编程一样：

```
// START_OF_MSGLOOP
while (GetMessage (&Msg, mymain->hwnd)) {
        TranslateMessage (&Msg);
        DispatchMessage (&Msg);
}
// END_OF_MSGLOOP
```

创建模态主窗口

我们知道，主窗口有模态和非模态之分，上面的示例程序建立了一个标准的非模态主窗口，如果我们希望创建模态主窗口，只需用 doModal 方创替代现有的消息循环即可，如下所示：

```
// START_OF_MINIGUIMAIN
int MiniGUIMain (int argc, const char* argv[])
```

```
{
        ncsInitialize ();

        mMainWnd* mymain = (mMainWnd*) ncsCreateMainWindow (
        NCSCTRL_MAINWND, "Hello, world!",
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        1,
        0, 0, 300, 200,
        HWND_DESKTOP,
        0, 0,
        NULL,
        NULL,
        mymain_handlers,
        0);

        _c(mymain)->doModal (mymain, TRUE);

        ncsUninitialize ();

        return 0;
}
// END_OF_MINIGUIMAIN
```

上述代码中，

- _c 宏的作用是取出 mymain 保存的类结构体（mMainWndClass）指针，因为 doModal 是 mMainWnd 类的一个虚函数，保存在虚函数表（mMainWndClass 的实例）中。
- doModal 方法第一个参数为 mMainWnd* self，和 C++ 中的 this 指针类似；第二个参数 BOOL autoDestroy，告诉 doModal 方法，是否自动删除对话框对象。当该参数 TRUE 时。调用完 doModal 后，mymain 就会被释放，不能使用了。

更进一步使用模板创建主窗口和控件

我们知道 MiniGUI 提供对话框模版的功能，使用模版，可以将控件和主窗口用数据结构的形式定义出来，从而帮助简化应用程序的编写。NCS 在 MiniGUI 基础上定义了包含更多信息的窗口模板结构，相对于 MiniGUI 的 DLGTEMPLATE 来说，要复杂一些，但更方便编程。需要注意的是，NCS 的窗口模版通常是由 miniStudio 根据用户设计的 UI 而自动生成的。更加复杂的窗口模版结构将帮助 NCS 更好地从 miniStudio 生成的资源文件中加载窗口。尽管模板不是针对开发者的，但我们也需要了解窗口模板的定义和使用方法。

清单 p2c1-3 中的代码展示了如何用模板创建一个如下图所示的对话框，并展示了如何使用属性和渲染器。

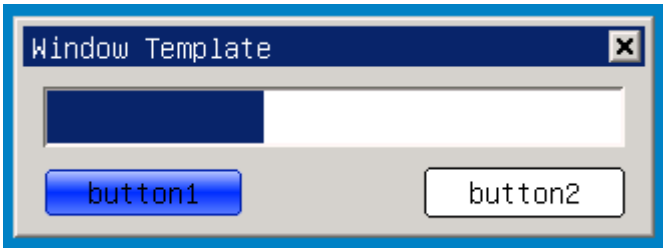


图 p1c1-2 使用窗口模版创建的对话框

清单 p2c1-3 wnd_template.c

```
/*
** $Id$
**
** Listing P2C1.3
**
** wnd_template.c: Sample program for mGNCS Programming Guide
**      Using window template.
**
** Copyright (C) 2009 Feynman Software.
```

```
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/control.h>

#include <mgncs/mgncs.h>

#define ID_BTN  101
#define ID_PROG 200

// START_OF_HANDLERS
static BOOL mymain_onCreate (mWidget* _this, DWORD add_data)
{
        SetTimer (_this->hwnd, 100, 20);
        return TRUE;
}
```

```
// START_OF_ONTIMER

static void mymain_onTimer (mWidget *_this, int id, DWORD count)

{

        static int pb_pos = 0;

        mProgressBar *pb = (mProgressBar*)ncsGetChildObj (_this->hwnd, ID_PROG);

        if (pb)

        _c(pb)->setProperty(pb, NCSP_PROG_CURPOS, pb_pos++);

}

// END_OF_ONTIMER


static void mymain_onClose (mWidget* _this, int message)

{

        DestroyMainWindow (_this->hwnd);

        PostQuitMessage (_this->hwnd);

}


static NCS_EVENT_HANDLER mymain_handlers[] = {

        {MSG_CREATE, mymain_onCreate},

        {MSG_CLOSE, mymain_onClose},

        {MSG_TIMER, mymain_onTimer},

        {0, NULL}

};

// END_OF_HANDLERS
```

```
// START_OF_RDRINFO
static NCS_RDR_ELEMENT btn_rdr_elements[] =
{
        { NCS_MODE_USEFLAT, 1},
        { -1, 0 }
};

static NCS_RDR_INFO btn1_rdr_info[] =
{
        {"fashion","fashion", btn_rdr_elements}
};

static NCS_RDR_INFO btn2_rdr_info[] =
{
        {"flat", "flat", NULL}
};
// END_OF_RDRINFO

// START_OF_PROPERTIES
static NCS_PROP_ENTRY progress_props [] = {
        {NCSP_PROG_MAXPOS, 100},
        {NCSP_PROG_MINPOS, 0 },
        {NCSP_PROG_LINESTEP, 1},
        {NCSP_PROG_CURPOS, 0 },
```

```
        { 0, 0 }
};
// END_OF_PROPERTIES

// START_OF_TEMPLATE
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_PROGRESSBAR,
                ID_PROG,
                10, 10, 290, 30,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
                "",
                progress_props,
                NULL,
                NULL, NULL, 0, 0
        },
        {
                NCSCTRL_BUTTON,
                ID_BTN,
                10, 50, 100, 25,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
                "button1",
```

```
                NULL,
                btn1_rdr_info,
                NULL, NULL, 0, 0
        },
        {
                NCSCTRL_BUTTON,
                ID_BTN,
                200, 50, 100, 25,
                WS_VISIBLE,
                WS_EX_NONE,
                "button2",
                NULL,
                btn2_rdr_info,
                NULL, NULL, 0, 0
        },

};

static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 320, 110,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
```

```
        "Window Template",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};
// END_OF_TEMPLATE

int MiniGUIMain (int argc, const char* argv[])
{
        ncsInitialize ();

        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);

        _c(mydlg)->doModal (mydlg, TRUE);

        ncsUninitialize ();

        return 0;
}
```

这个程序定义了含有三个控件的一个对话框：

- 上边的进度条，其最大、最小值是在初始化时设置；其当前位置会被定时刷新；
- 左边的按钮，使用了 Fashion 渲染器，以默认属性绘制；
- 右边的按钮，使用了 Flat 渲染器，以默认属性绘制。

模板

函数 ncsCreateMainWindowIndirect 根据模版创建对应的主窗口，在上面的程序中，这个模板是 mymain_templ。mymain_templ 的类型是 NCS_MAINWND_TEMPLATE，其中有个一个指针指向子控件的模板结构数组（NCS_WND_TEMPLAT*）。NCS_WND_TEMPLATE 和 NCS_MAINWND_TEMPLATE 的差别在于前者没有 HICON 和 HMENU 这种专门用于主窗口的数据成员。

mymain_templ 的声明如下：

```
static NCS_MAINWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,              // 窗口类名，这里取 NCSCTRL_DIALOGBOX
        1,                              // 窗口标示符
        0, 0, 320, 110,                 // 窗口位置和尺寸
        WS_CAPTION | WS_BORDER | WS_VISIBLE,    // 窗口风格
        WS_EX_NONE,                     // 窗口的扩展风格
        "Window Template",              // 窗口标题
        NULL,                           // 初始化属性
        NULL,                           // 渲染器信息
        mymain_handlers,                // 窗口事件处理器
        _ctrl_templ,                    // 子窗口模版结构数组指针
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),    // 子窗口个数
        0, 0, 0                         // 窗口附加数据、图标句柄及菜单栏句柄
};
```

需要注意的是：NCS_WND_TEMPLATE 也包含一个指向子控件模版结构数组的指针成员，这样可以形成多级的嵌套模版。

设置窗口属性

在这个示例中，我们通过 NCS_PROP_ENTRY 结构数组定义了进度条控件的初始属性值，这包括：

- 进度条的最大值；
- 进度条的最小值；
- 进度条的步进值；
- 进度条的初始位置。

具体代码如下：

```
static NCS_PROP_ENTRY progress_props [] = {
        {NCSP_PROG_MAXPOS, 100},
        {NCSP_PROG_MINPOS, 0 },
        {NCSP_PROG_LINESTEP, 1},
        {NCSP_PROG_CURPOS, 0 },
        { 0, 0 }
};
```

注意上述结构数组中的最后一个成员 {0, 0}，NCS 通过定义一个全零的结构来表示属性信息的结束。

在定义控件时，将上述属性结构数组的指针赋予对应的窗口模版属性成员即可，如下所示：

```
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_PROGRESSBAR,
                ID_PROG,
                10, 10, 290, 30,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
                "",
```

```
                progress_props,

                NULL,

                NULL, NULL, 0, 0

        },

        ...

};
```

如果使用 MiniGUI 固有控件集，要实现这种功能，必须在对话框的 MSG_INITDIALOG 消息中调用 SendMessage 来实现。而在新控件集中，只需要设置一个 NCS_PROP_ENTRY 数组，既可自动实现该功能。它不仅仅为了减少代码量，简化编程，更重要的是，提供了一个统一的接口。用户可以通过这种接口，把初始信息统一存放在外部文件中；NCS 的资源管理模块就可以完成类似的功能，且 miniStudio 即利用了这种便利性。

动态设置属性

属性的设置也可以在程序运行过程中动态设置。通过 mComponentClass（这是可见组件，即控件和不可见组件的基类）的 setProperty （被其子类继承并实现）就可实现。类似地，还有 getProperty 方法，用于获取给定属性的值。这两个方法的原型如下：

```
BOOL (*setProperty) (clss *self, int id, DWORD value);

DWORD (*getProperty) (clss *self, int id);
```

需要注意的是，所有的属性值都被定义成了 DWORD 类型，这个类型是个 32 位的值，可用来存储句柄、指针、整数值等等。

这整个示例程序中，我们创建了一个定时器来周期性地设置进度条的当前位置属性。在定时器的处理函数中做如下代码调用：

```


static void mymain_onTimer (mWidget *_this, int id, DWORD count)

{

        static int pb_pos = 0;

        mProgressBar *pb = (mProgressBar*)ncsGetChildObj (_this->hwnd, ID_PROG);

        if (pb)
```

```
        _c(pb)->setProperty(pb, NCSP_PROG_CURPOS, pb_pos++);
}
```

使用渲染器

渲染器是新控件集的一大特色。它的主要功能有

1. 在不更改控件的内部逻辑实现的基础上，对控件的外观做定制
2. 渲染器随控件的种类变化而变化，新增、删除一种控件不会对其他控件造成影响，这一点和老控件集的渲染器不同

作为控件的使用者，开发者不需要实现一个完整的渲染器，而只需要使用应有的渲染器。上面的例子使用了和窗口模版类似的方法来定义一个控件所使用的渲染器信息：

```
static NCS_RDR_ELEMENT btn_rdr_elements[] =
{
        { NCS_MODE_USEFLAT, 1},
        { -1, 0 }
};

static NCS_RDR_INFO btn1_rdr_info[] =
{
        {"fashion","fashion", btn_rdr_elements}
};

static NCS_RDR_INFO btn2_rdr_info[] =
{
        {"flat", "flat", NULL}
};
```

- btn_rdr_elements 定义了渲染器的属性数组。渲染器属性和控件的属性类似，其机制是完全一样的。它以 {-1,0} 为数组的结束，其中 NCS_MODE_USEFLAT 属性指出 Fashion 渲染器使用平坦风格（相对的，3D 风格下有明显的立体感）绘制一个对象。

- btn1_rdr_info 的 NCS_RDR_INFO，指出控件使用的渲染器类型及其属性列表。NCS_RDR_INFO 的定义如下：

```
/**
* A renderer element struct
*
* \note This struct is used by \ref NCS_RDR_INFO when create a NCS widget
*
* \sa NCS_RDR_INFO
*/
typedef struct _NCS_RDR_ELEMENT {
        /**
        * The renderer element id
        */
        int id;
        /**
        * The remderer element value
        */
        DWORD value;
}NCS_RDR_ELEMENT;


/**
* A struct used to include all the renderer info when create a widget
```

```
*
* \sa NCS_CREATE_INFO, NCS_MAIN_CREATE_INFO, NCS_WND_TEMPLATE,
*     NCS_MAINWND_TEMPLATE
*/
typedef struct _NCS_RDR_INFO {
        /**
        * The global renderer, the minigui 3.0 defined renderer
        */
        const char* glb_rdr;
        /**
        * The control renderer, defined by NCS
        */
        const char* ctl_rdr;
        /**
        * The Renderer element array, end by  { -1, 0}
        */
        NCS_RDR_ELEMENT* elements;
}NCS_RDR_INFO;
```

其中，

- glb_rdr 指全局渲染器，主要用于非客户区的绘制风格，这些区域由 MiniGUI 负责绘制。
- ctrl_rdr 指控件渲染器，则主要用于控件本身的绘制。

NCS_RDR_INFO 可以在模板中给出，从而可以直接创建出使用给定渲染器的控件来。

事件监听和连接

mGNCS 提供了一种类似 QT 的 signal 和 slot 的机制，能够把一个对象的事件链接到任意一个对象上。

基本原理

mGNCS 的事件监听链接机制提供了一种将事件发送者和事件关心者解耦的方式。它通过全局数据表，把发送者和接收者的关系对应起来。如下图所示：

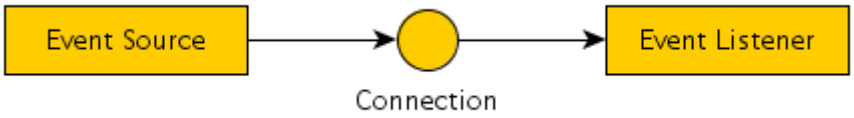


图 p1c1-3 事件监听和连接

当一个对象被删除时：

- 所有监听该对象的连接将被删除
- 该对象所监听的连接也将被删除

主要接口函数

下面给出了添加事件监听器的接口：

```
/**
* \typedef typedef BOOL (*NCS_CB_ONOBJEVENT) (mObject* self, mObject *sender,
int event_id, DWORD param);
* \brief The callback of connection
*
* \note For widgets, only support notification
*
* \param self  The listener pointer
* \param sender The sender of event
* \param event_id the id of event
* \param param the param of event
*
```

```
* \return TRUE - continue dispatch the event; FALSE - stop the event.
*/
typedef BOOL (*NCS_CB_ONOBJEVENT)(mObject* self, mObject *sender,
int event_id, DWORD param);

#define NCS_CB_ONPIECEEVENT NCS_CB_ONOBJEVENT

/**
* \fn BOOL ncsAddEventListener(mObject *sender, \
*                                          mObject* listener, \
*                                          NCS_CB_ONOBJEVENT event_proc, \
*                                          int event_id);
* \brief connect sender object to listener object
*
* \param sender The sender of event
* \param listener The listener of event
* \param event_proc The processing callback of event
* \param event_id The id of event
*
* \return TRUE - Sucessed, FALSE - Failed
*
* \sa ncsAddEventListener, ncsAddEventListeners, NCS_CB_ONOBJEVENT
*/
BOOL ncsAddEventListener(mObject *sender,
```

```
mObject* listener,
NCS_CB_ONOBJEVENT event_proc,
int event_id);

/**
* BOOL ncsAddEventListeners(mObject *sender,\
*                                          mObject* listener,\
*                                          NCS_CB_ONOBJEVENT event_proc,\
*                                          int* event_ids);
* \brief connect a group of events from sender to listener
*
* \param sender The sender of events
* \param listener The listener of events
* \param event_proc The processing callback of events
* \param event_ids The id array of events, end by 0
*
* \return TRUE - Sucessed, FALSE - Failed
*
* \sa ncsAddEventListener, ncsAddEventListeners, NCS_CB_ONOBJEVENT
*/
BOOL ncsAddEventListeners(mObject *sender,
mObject* listener,
NCS_CB_ONOBJEVENT event_proc,
int* event_ids);
```

其中，

- 回调 NCS_CB_ONOBJEVENT 定义了事件的处理函数，该回调能够同时获取发送者和接收者的对象指针，避免用户去查找
- ncsAddEventListeners 增加一个对像对另外一个对象的一组事件的监听
- ncsAddEventListener 增加一个对象对另外一个对象的单个事件的监听

下面给出了删除事件监听器的接口：

```
/**
* \fn BOOL ncsRemoveEventListener(mObject * listener);
* \brief remove the connections which are listened by the object
*
* \note this function is called when a listener is destroied
*
* \param listener the object pointer of listener
*
* \return TRUE - sucessed; FALSE - failed
*/
BOOL ncsRemoveEventListener(mObject * listener);


/**
* \fn BOOL ncsRemoveEventSource(mObject *sender);
* \brief remove the connections when a sender is destroied
*
* \note this function is called when a sender is destroied
*
* \param sender the sender of events
```

```
*

* \return TRUE - sucessed; FALSE - failed

*/

BOOL ncsRemoveEventSource(mObject *sender);
```

其中，

- 删除一个监听者或者事件发送者。因为只要监听者或者发送者有一个不存在，整个连接就没有存在的意义，都会被删除。
- 这两个函数由 mObject 自动调用，一般情况下不需要主动调用

下面给出了激发一个事件连接的接口：

```
/**

* \fn void ncsRaiseEvent (mObject *sender, int event_id, DWORD param);

* \biref Raise an event to listeners

*

* \note It will call the callback of \ref NCS_CB_ONOBJEVENT

*       added by \ref ncsAddEventListeners, or \ref ncsAddEventListener

*

* \param sender event sender

* \param event_id the id of event

* \param param the event param

*

* \sa NCS_CB_ONOBJEVENT, ncsAddEventListener, ncsAddEventListeners

*/

void ncsRaiseEvent(mObject *sender, int event_id, DWORD param);
```

该函数主要为内部使用，用于激发一个事件连接。该接口仅在用户希望手动强制引发事件，或者编写 NCS 控件时使用。

用法及示例

我们将在窗口模版示例程序的基础上，增加如下两个事件监听器：

- 单击 Stop 按钮时，将通过关闭定时器来停止进度条的自动更新；
- 单击 Exit 按钮时，将关闭整个对话框。

该程序创建的窗口效果如下图所示：

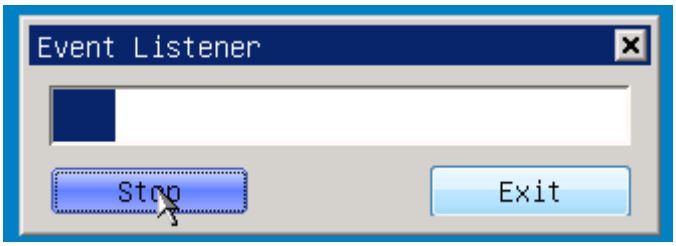


图 p1c1-4 监听事件示例

以下代码定义了两个监听器，分别用于 Stop 按钮和 Exit 按钮：

```
static BOOL mymain_onExit (mMainWnd *self, mButton *sender, int id, DWORD param)
{
        mymain_onClose ((mWidget*)self, MSG_CLOSE);
        return TRUE;
}

static BOOL mymain_onStop (mMainWnd *self, mButton *sender, int id, DWORD param)
{
        KillTimer (self->hwnd, 100);
        return TRUE;
}
```

这两个事件监听器在初始化窗口时，被连接到上述两个控件的单击事件上：

```
static BOOL mymain_onCreate (mWidget* self, DWORD addData)
{
        mButton *btn;

        btn = (mButton*)_c(self)->getChild (self, IDC_EXIT);
        ncsAddEventListener ((mObject*)btn, (mObject*)self,
        (NCS_CB_ONOBJEVENT)mymain_onExit, NCSN_WIDGET_CLICKED);

        btn = (mButton*)_c(self)->getChild (self, IDC_STOP);
        ncsAddEventListener ((mObject*)btn, (mObject*)self,
        (NCS_CB_ONOBJEVENT)mymain_onStop, NCSN_WIDGET_CLICKED);

        SetTimer (self->hwnd, 100, 20);
        return TRUE;
}

static void mymain_onTimer (mWidget *self, int id, DWORD count)
{
        static int pb_pos = 0;
        mProgressBar *pb = (mProgressBar*)ncsGetChildObj (self->hwnd, IDC_PROG);
        if (pb)
        _c(pb)->setProperty (pb, NCSP_PROG_CURPOS, pb_pos++);
}
```

```
static void mymain_onClose (mWidget* self, int message)
{
        DestroyMainWindow (self->hwnd);
        PostQuitMessage (self->hwnd);
}

static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate},
        {MSG_CLOSE, mymain_onClose},
        {MSG_TIMER, mymain_onTimer},
        {0, NULL}
};
```

显然，事件监听器机制为新控件集的编程带来了更多方便，也方便 miniStudio 工具实现可视化编程。

数据绑定和数据源

数据绑定和数据源是 mGNCS 提供给应用程序的两个非常重要的机制，这两个机制均有助于实现程序逻辑和它所处理的数据之间的分离，且便于类似 miniStudio 这样的可视化 GUI 设计工具来设计界面。本章将简单说明数据绑定和数据源的基本概念，后面的章节将详细讲述相关的接口及使用方法。

如果读者曾经使用 Visual Studio 开发过基于 MFC 的 C++ 应用程序的话，则一定记得这个工具提供一种机制，可以将对话框中的控件内容和给定的类成员变量绑定起来。在 VC 中，对话框控件的初始化值，可以从绑定的对话框成员变量中自动获得并设置好，而当对话框退出时，控件中的值又可以自动赋值给对应的对话框类成员变量。这就是 mGNCS 提供的数据源和数据绑定功能的思想来源。但是，mGNCS 提供的数据源和数据绑定功能更加强大。利用 mGNCS 的数据绑定功能，当 mGNCS 控件的值发生变化时，我们可以自动更新其他控件，或者将数据再次保存到期望的数据源中；通过 mGNCS 的数据源，我们可以定义不同格式的数据来源，如程序定义的字符串数组、文本文件、配置文件，甚至数据库等等，并使用这些数据自动填充到 mGNCS 控件中。

下边我们分别介绍 mGNCS 的数据绑定和数据源功能。

数据绑定

如果我们试图寻找图形用户界面和字符用户界面的相同点，我们会发现，它们都属于人机交互范畴，而人机交互的本质就是数据以人能够理解的方式和计算机能够理解的方式之间进行转换。注意以上的表述有三个关键词：

- 数据
- 人能够理解的方式
- 计算机能够理解的方式

我们知道，计算的本质是对数据的计算，数据是一切计算的核心；所谓方式，对计算机来讲就是数据的格式。人机界面通过显示特定格式的数据和人交互，而计算机在进行内部运算时，却希望以它能够理解的格式计算。当这两种格式不匹配时，我们该怎么办？其答案就是数据绑定。下图给出了 NCS 数据绑定的模型：

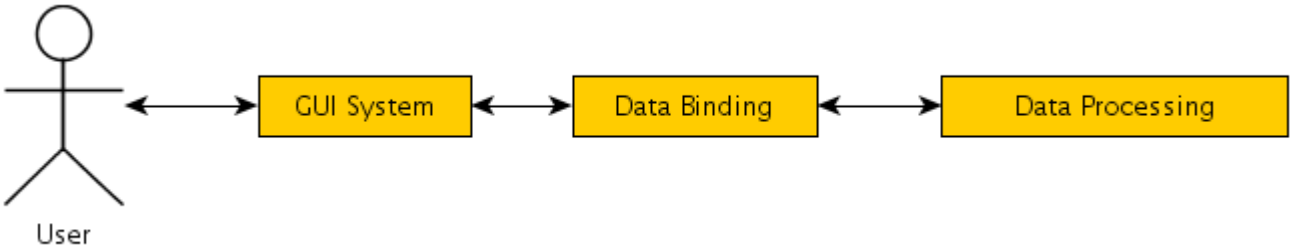


图 p2c1-5 mGNCS 的数据绑定模型

数据绑定在图形用户界面和内部逻辑之间传递数据，其优点是：

- 解耦图形用户界面和内部逻辑的处理，使开发人员更易于更换界面
- 规范化和模块化应用程序，提高程序的可扩展性
- 简化编程，把程序员从繁琐的 Get/Set 操作中解脱出来
- 统一接口，有利于 miniStudio 等工具进行可视化的操作，也有利于用户抽象

其缺点是需要额外的开销，不过，操作界面的代码本身也会带来开销，两者相抵，使用数据绑定是十分合算的。

清单 p2c1-5 给出了一个简单的数据绑定示例程序，这个程序的界面如下图所示：

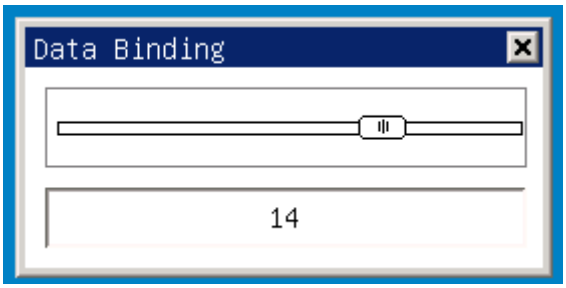


图 p2c1-6 数据绑定示例程序

该程序将编辑框中的内容和拖动条的位置绑定在了一起：

- 拖动拖动条，编辑框中的内容将自动改变，以反映当前的拖动条位置；
- 在编辑框中键入一个整数值（0~20），则拖动条的当前位置也将发生对应的变化。

清单 p1c2-5 data_binding.c

```
/*
** $Id$
**
** Listing P2C1.5
**
** data_binding.c: Sample program for mGNCS Programming Guide
**      Using data binding.
**
** Copyright (C) 2009 Feynman Software.
*/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#include <minigui/common.h>
```

```
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>

#include <mgncs/mgncs.h>

#define IDC_TRACKBAR     101
#define IDC_SLEDIT       102

// START_OF_BINDING
static BOOL mymain_onCreate (mWidget* self, DWORD add_data)
{
        mTrackbar * tb = (mTrackbar*)_c(self)->getChild (self, IDC_TRACKBAR);
        mSlEdit   * se = (mSlEdit*) _c(self)->getChild (self, IDC_SLEDIT);

        ncsConnectBindProps (NCS_CMPT_PROP (tb, NCSN_TRKBAR_CHANGED,
        NCSP_TRKBAR_CURPOS, NCS_BT_INT,
        NCS_PROP_FLAG_READ|NCS_PROP_FLAG_WRITE),
        NCS_CMPT_PROP (se, NCSN_EDIT_CHANGE,
        NCSP_WIDGET_TEXT, NCS_BT_STR,
        NCS_PROP_FLAG_READ|NCS_PROP_FLAG_WRITE),
        NCS_BPT_DBL);

        ncsAutoReflectObjectBindProps ((mObject*)se);
```

```
        return TRUE;
}
// END_OF_BINDING

static void mymain_onClose (mWidget* self, int message)
{
        DestroyMainWindow (self->hwnd);
        PostQuitMessage (self->hwnd);
}

static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate },
        {MSG_CLOSE, mymain_onClose },
        {0, NULL }
};

static NCS_PROP_ENTRY trackbar_props [] = {
        {NCSP_TRKBAR_MINPOS, 0},
        {NCSP_TRKBAR_MAXPOS, 20},
        {NCSP_TRKBAR_CURPOS, 10},
        {NCSP_TRKBAR_LINESTEP, 2},
        {NCSP_TRKBAR_PAGESTEP, 5},
        {0, 0}
};
```

```
static NCS_RDR_INFO trackbar_rdr_info[] =
{
        {"flat", "flat", NULL},
};

static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_TRACKBAR,
                IDC_TRACKBAR,
                10, 10, 240, 40,
                WS_BORDER | WS_VISIBLE | NCSS_TRKBAR_NOTICK | NCSS_NOTIFY,
                WS_EX_TRANSPARENT,
                "",
                trackbar_props,
                trackbar_rdr_info,
                NULL, NULL, 0, 0,
                MakeRGBA(255,0,0,255)
        },
        {
                NCSCTRL_SLEDIT,
                IDC_SLEDIT,
                10, 60, 240, 30,
                WS_BORDER | WS_VISIBLE | NCSS_EDIT_CENTER | NCSS_NOTIFY,
```

```
                WS_EX_NONE,
                "edit",
                NULL, NULL, NULL, NULL, 0, 0
        },
};

static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 268, 130,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Data Binding",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        MakeRGBA(255,255,255,255)
};

int MiniGUIMain(int argc, const char* argv[])
{
```

```
        mDialogBox* mydlg;

        if (argc > 1) {
                trackbar_rdr_info[0].glb_rdr = argv[1];
                trackbar_rdr_info[0].ctl_rdr = argv[1];
        }

        ncsInitialize ();

        mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);

        _c(mydlg)->doModal (mydlg, TRUE);

        MainWindowThreadCleanup (mydlg->hwnd);

        ncsUninitialize ();

        return 0;
}
```

完成上述的数据绑定，该程序仅仅在创建对话框时，做了如下所示的设置工作：

```
static BOOL mymain_onCreate (mWidget* self, DWORD add_data)
{
```

```
        mTrackbar * tb = (mTrackbar*)_c(self)->getChild (self, IDC_TRACKBAR);
        mSlEdit   * se = (mSlEdit*) _c(self)->getChild (self, IDC_SLEDIT);

        ncsConnectBindProps (NCS_CMPT_PROP (tb, NCSN_TRKBAR_CHANGED,
        NCSP_TRKBAR_CURPOS, NCS_BT_INT,
        NCS_PROP_FLAG_READ|NCS_PROP_FLAG_WRITE),
        NCS_CMPT_PROP (se, NCSN_EDIT_CHANGE,
        NCSP_WIDGET_TEXT, NCS_BT_STR,
        NCS_PROP_FLAG_READ|NCS_PROP_FLAG_WRITE),
        NCS_BPT_DBL);

        ncsAutoReflectObjectBindProps ((mObject*)se);
        return TRUE;
}
```

上面的代码段完成了两件事情：

- 调用 ncsConnectBindProps 函数，将拖动条的当前位置值属性（NCSP_TRKBAR_CURPOS）和编辑框的文本属性（NCSP_WIDGET_TEXT）绑定在了一起，并且当拖动条产生位置改变事件（NCSN_TRKBAR_CHANGED），或者编辑框产生内容改变事件（NCSN_EDIT_CHANGE）时触发绑定。
- 调用 ncsAutoReflectObjectBindProps 函数，使得编辑框可以自动响应数据绑定带来的内容改变。

假设没有数据绑定功能，我们就需要编写两个事件处理器分别响应两个控件的两个事件，而现在，一切都变得非常简单。

需要注意的是，拖动条的位置值属性是整数类型，而编辑框的内容属性是字符串类型，这两者之间的数据类型转换，将由 mGNCS 自动完成。

数据源

数据源就是数据的集合，通过抽象的数据源接口，我们可以为一些大型控件，如列表框、列表型等控件提供良好的数据交换机制。定义抽象的数据源接口之意义在于：

- 统一管理数据，是界面和数据分离
- 统一数据访问接口，便于程序的开发和维护，也便于 miniStudio 工具进行可视化处理

目前，mGNCS 实现了对如下几类数据源的支持：

- 应用程序定义的 C 语言数据，mGNCS 称为 Static Data Source（静态数据源）；
- 来自 MiniGUI 配置文件格式的文本数据；
- 来自类似 UNIX passwd 文件的以行为单位的文本域数据。

清单 p1c2-6 中的代码使用 C 程序定义的静态数据初始化了一个列表项控件，如下图所示：

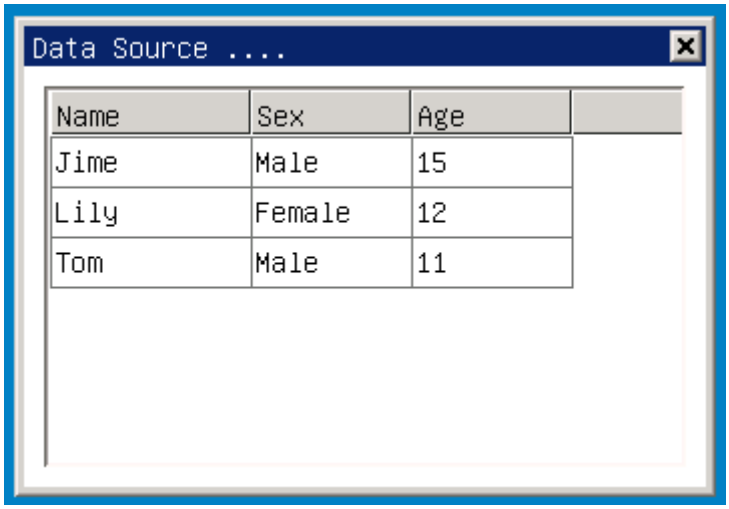


图 p2c1-7 数据源示例程序

清单 p1c2-6 data_source.c

```
/*
** $Id$
**
** Listing P2C1.6
**
** data_source.c: Sample program for mGNCS Programming Guide
```

```
**      Using static data source.
**
** Copyright (C) 2009 Feynman Software.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>

#include <mgncs/mgncs.h>

#define IDC_LISTVIEW 100

// START_OF_DEFINEDS
static const NCS_LISTV_CLMRD _header[] = {
        {"Name", "", 100, NCSF_LSTCLM_LEFTALIGN},
        {"Sex", "", 80, NCSF_LSTCLM_LEFTALIGN},
        {"Age", "", 80, NCSF_LSTCLM_LEFTALIGN}
};
```

```
static const char* _content[][3] = {
        {"Jime", "Male", "15"},
        {"Lily", "Female", "12"},
        {"Tom", "Male", "11"}
};
// END_OF_DEFINEDS

// START_OF_SETDATA
static BOOL mymain_onCreate (mWidget* self, DWORD add_data)
{
        mListView *lv = (mListView*)_c(self)->getChild (self, IDC_LISTVIEW);

        if (lv) {
                mRecordSet *rs;
                rs = _c(g_pStaticDS)->selectRecordSet (g_pStaticDS,
                "/listview/header", NCS_DS_SELECT_READ);
                _c(lv)->setSpecial (lv, NCSID_LISTV_HDR, (DWORD)rs, NULL);
                rs = _c(g_pStaticDS)->selectRecordSet (g_pStaticDS,
                "/listview/content", NCS_DS_SELECT_READ);
                _c(lv)->setSpecial (lv, NCS_CONTENT, (DWORD)rs, NULL);
        }

        return TRUE;
```

```
}

// END_OF_SETDATA


static void mymain_onClose (mWidget* self, int message)

{

        DestroyMainWindow (self->hwnd);

        PostQuitMessage (self->hwnd);

}


static NCS_EVENT_HANDLER mymain_handlers[] = {

        {MSG_CREATE, mymain_onCreate},

        {MSG_CLOSE, mymain_onClose},

        {0, NULL}

};


static NCS_WND_TEMPLATE _ctrl_templ[] = {

        {

                NCSCTRL_LISTVIEW,

                IDC_LISTVIEW,

                10, 10, 320, 190,

                WS_BORDER | WS_VISIBLE | NCSS_LISTV_SORT

                | NCSS_LISTV_LOOP,

                WS_EX_NONE,

                "",
```

```
                NULL, NULL, NULL, NULL, 0, 0
        }
};

static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_MAINWND,
        1,
        0, 0, 350, 240,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Data Source ....",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0, 0, 0,
};

int MiniGUIMain (int argc, const  char* argv[])
{
        mDialogBox* mydlg;

        ncsInitialize ();
```

```
        // START_OF_REGDS
        ncsRegisterStaticData ("/listview/header", (void*)_header, 3,
        sizeof(NCS_LISTV_CLMRD)/sizeof(DWORD), sizeof(DWORD),
        gListVColumnRecordTypes);
        ncsRegisterStaticData ("/listview/content", (void*)_content, 3,
        3, sizeof(char*), NULL);
        // END_OF_REGDS

        mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);

        _c(mydlg)->doModal (mydlg, TRUE);

        MainWindowThreadCleanup (mydlg->hwnd);

        ncsUninitialize ();

        return 0;
}
```

要实现上面的数据源功能，该程序完成了如下三个方面的工作：

第一步，定义数据源：

```
static const NCS_LISTV_CLMRD _header[] = {
```

```
        {"Name", "", 100, NCSF_LSTCLM_LEFTALIGN},
        {"Sex", "", 80, NCSF_LSTCLM_LEFTALIGN},
        {"Age", "", 80, NCSF_LSTCLM_LEFTALIGN}
};

static const char* _content[][3] = {
        {"Jime", "Male", "15"},
        {"Lily", "Female", "12"},
        {"Tom", "Male", "11"}
};
```

上述代码分别定义了列表型控件的表头信息以及内容数据。

第二步，注册数据源：

```
ncsRegisterStaticData ("/listview/header", (void*)_header, 3,
sizeof(NCS_LISTV_CLMRD)/sizeof(DWORD), sizeof(DWORD),
gListVColumnRecordTypes);
ncsRegisterStaticData ("/listview/content", (void*)_content, 3,
3, sizeof(char*), NULL);
```

上述代码将上述两种数据分别定义成了“listview/header”和“listview/content”，在需要引用这两种数据时，使用这里给出的名称即可。

第三步，使用数据源：

```
static BOOL mymain_onCreate (mWidget* self, DWORD add_data)
{
        mListView *lv = (mListView*)_c(self)->getChild (self, IDC_LISTVIEW);
```

```
        if (lv) {
                mRecordSet *rs;
                rs = _c(g_pStaticDS)->selectRecordSet (g_pStaticDS,
                "/listview/header", NCS_DS_SELECT_READ);
                _c(lv)->setSpecial (lv, NCSID_LISTV_HDR, (DWORD)rs, NULL);
                rs = _c(g_pStaticDS)->selectRecordSet (g_pStaticDS,
                "/listview/content", NCS_DS_SELECT_READ);
                _c(lv)->setSpecial (lv, NCS_CONTENT, (DWORD)rs, NULL);
        }

        return TRUE;
}
```

上述代码从数据源中获取对应的数据，然后调用列表型控件的 setSpecificData 方法进行设置。

小结

通过本章的学习，我们可以对 mGNCS 有一个整体的了解。相比 MiniGUI 的固有控件系统，mGNCS 重新构架了 MiniGUI 的控件体系，使之更加规范，且易于定制，另外，这种设计和针对 MiniGUI 应用开发的 miniStudio 集成开发环境是相适应的，两者之间是相辅相成的关系。

相比 MiniGUI 的固有控件体系，mGNCS 引入了面向对象的设计思想，大大提高了 MiniGUI 的可定制性和可扩展性，且便于可视化设计。当然，这对应用程序开发人员来讲带来了更长的学习路径，但相信通过本编程指南的讲解，任何已经掌握 MiniGUI 应用开发的开发人员都能够快速掌握 mGNCS。当然，设计和引入 mGNCS 的初衷，是为了开发更好的设计工具，即 miniStudio，而大部分功能，都可以在 miniStudio 中通过可视化设计来获得。毋庸置疑，配合 miniStudio 使用 mGNCS，将大大提高嵌入式图形界面应用程序的开发效率。

第二部分第二章 渲染器及资源管理

外观渲染器

渲染器的全称为“外观渲染器（Look and Feel Renderer）”，它在 MiniGUI 3.0 中发展出来的。

渲染器将窗口的管理逻辑和窗口的绘制分离，MiniGUI 只负责管理窗口，而由渲染器完成窗口非客户区的绘制。渲染器的概念，类似于皮肤或者主题的概念。不同的是：

1. MiniGUI 提供了专门的渲染器接口，由具体的渲染器实现
2. 渲染器由一组接口实现和一组属性定义组成
3. 渲染器可以应用到全局，对所有窗口起效; 也可以应用到一个专门的窗口实例上

MiniGUI 本身提供了一组渲染器接口，这些是基于窗口本身的，属于系统渲染器，或者是全局渲染器；NCS 为每个控件提供了渲染器接口，不同的控件接口不一样，属于控件渲染器。

渲染器公共属性

属性	类型	说明
RDR_3DBODY_FGCLR	DWORD(MakeRGBA)	3D 对象前景色
RDR_3DBODY_BGCLR	DWORD(MakeRGBA)	3D 对象背景色
RDR_CLIENT_FGCLR	DWORD(MakeRGBA	窗口客户区前景色
RDR_CLIENT_BGCLR	DWORD(MakeRGBA	窗口客户区背景色
RDR_SELECTED_FGCLR	DWORD(MakeRGBA)	被选择对象的前景色
RDR_SELECTED_BGCLR	DWORD(MakeRGBA)	被选择对象的背景色
RDR_SELECTED_LOSTFOCUS_CLR	DWORD(MakeRGBA)	失去焦点对象的颜色
RDR_DISABLE_FGCLR	DWORD(MakeRGBA)	不可用对象的前景色
RDR_DISABLE_BGCLR	DWORD(MakeRGBA)	不可用对象的背景色
RDR_HIGHLIGHT_FGCLR	DWORD(MakeRGBA)	高亮对象的前景色
RDR_HIGHLIGHT_BGCLR	DWORD(MakeRGBA)	高亮对象的背景色
RDR_METRICS_BORDER	int	边框的大小值
RDR_FONT	PLOGFONT	窗口使用的逻辑字体

RDR_BKIMAGE	PBITMAP	窗口背景图片
RDR_BKIMAGE_MODE	enum ImageDrawMode	窗口绘制模式

资源管理

在新控件集中，资源管理模块作为一个独立的模块，主要负责对资源包进行管理，其中包括资源包的装载、访问和卸载等功能。它在访问资源时是通过指定资源所在的资源包和资源的唯一标识符（ ID）等信息来获取所需要的内容 。

其中资源包是为了将各种资源有效地统一管理，统一资源与资源标识符 （ID）的映射而提出的概念。一个资源包内可以包含多种资源类型，如图片、字符串、UI 资源等等。资源包为资源的管理带来了诸多如下好处。

在了解资源管理前，需要了解下面两个基础概念：

资源包

资源的集合，任意种类和任意个数的资源集合，是资源替换的最小单位。

资源 ID

访问资源包内某一资源的唯一标识。

mGNCS 提供的资源管理模块，相比较 MiniGUI 固有控件集来讲，具有诸多的优势。如：

- 资源管理与应用代码的分离。
- 统一了各种资源的管理接口，访问资源更加便捷。
- 通过资源包的简单替换即可实现多语言，多风格，换肤等效果。
- 资源部署更加简单。
- 配合可视化工具 miniStudio，方便地生成包含各类资源描述的资源包。

mGNCS 的资源管理主要是围绕资源包来实现对资源的各种管理和访问的。如果要正确的访问某个资源，只需要知道其所在的资源包和对应的资源 ID。整体访问流程通常为：

1. 装载指定资源包。
2. 通过资源 ID 获取资源。
3. 使用资源，某些资源需要在使用后释放资源。
4. 卸载资源包。

当前资源包支持的资源类型有：

资源类型 ID	说明	备注
NCSRT_UI	UI 窗口资源	对主窗口和控件的描述信息。如窗口风格、大小、标题和属性等。
NCSRT_STRING	字符串资源	文件名、渲染器名和控件类名等固定的系统常量字符串。

NCSRT_TEXT	文本资源	UI 界面上用户指定的文本字符串及 locale 信息。
NCSRT_IMAGE	图片资源	各种 bmp, png 和 jpg 等图片信息。
NCSRT_RDR	渲染器资源	对指定的窗口渲染器的窗口元素外观属性的设置信息。
NCSRT_RDRSET	渲染器集资源	同类渲染器资源的集合。
NCSRT_BINARY	自定义资源	自定义格式的资源。

资源接口

访问资源包

资源包分为内建格式和外部文件 2 种，无论哪一种都应在系统启动时首先装载资源包，然后访问内部资源，最后系统退出后将资源包正确卸载。在访问非内建资源包时需要使用到如下接口：

```
HPACKAGE ncsLoadResPackageFromFile (const char* fileName);

#define ncsLoadResPackage  ncsLoadResPackageFromFile

void ncsUnloadResPackage (HPACKAGE package);
```

其中 ncsLoadResPackage 负责装载资源包， ncsUnloadResPackage 负责卸载资源包。二者需要成对使用。

如果需要装载的资源包是内建的内存格式，则需要首先通过获取内存资源包信息接口（ncsGetIncoreResPackInfo），将内存资源包具体信息返回，然后使用从内存装载资源包的接口（ncsLoadResPackageFromMem）装载资源包，最后使用上面提供的同一卸载接口（ncsUnloadResPackage）卸载资源包。其中 ncsGetIncoreResPackInfo 接口是由内建转换工具所提供，使用内建工具将指定资源包文件转换为内建资源时即已生成，应用无须关心其实现。

```
extern BOOL ncsGetIncoreResPackInfo(char **addr, int *size);

HPACKAGE ncsLoadResPackageFromMem (const void* mem, int size);
```

这是装载资源包文件的示例代码：

```


sprintf(f_package, "%s", "resmgr_demo.res");

SetResPath("./");
```

```
hPackage = ncsLoadResPackage (f_package);
if (hPackage == HPACKAGE_NULL) {
        printf ("load resource package:%s failure.\n", f_package);
        return 1;
}
```

这是卸载资源包文件的示例代码：

```
ncsUnloadResPackage(hPackage);
```

获取与设置 Locale

为了方便的进行 locale 的设置与获取，资源管理模块提供了下面两个接口：

```
const char* ncsSetDefaultLocale (char* language, char* country);
const char* ncsGetDefaultLocale(void);
```

其中设置接口仅对在其设置后通过资源包获取的资源有效，2 个参量分别代表代码语言和国家代码：

- language 语言代码用 2 个英文小写字母来表示。
- country 国家代码需要用 2 个或 3 个英文小写字母来表示。

如设置美式英文为默认 locale，可通过如下语句完成：

```
ncsSetDefaultLocale("en", "US");
```

获取接口返回当前使用的 locale 信息，该信息的格式为：language_country。

附：

ISO639 语言代码对照表 language

ISO3166 国家代码对照表 country

创建窗口 UI

在使用资源包的情况下，mGNCS 所提供的创建主窗口接口相对于 MiniGUI 的创建主窗口来说更加简单，对于主窗口的外观信息描述均已保存于资源包内，接口只需通过窗口资源 ID 即可将界面启动。其中：

- owner 参数指的是窗口的拖管窗口；
- hIcon, hMenu 默认为 0 即可；
- handlers 和 connects 是用户自定义的与该窗口资源 ID 相关联的事件侦听和连接；
- user_data 为附加数据，默认为 0 即可。

```
mMainWnd* ncsCreateMainWindowIndirectFromID (HPACKAGE package,

Uint32 wndId, HWND owner, HICON hIcon,

HMENU hMenu, NCS_EVENT_HANDLER_INFO* handlers,

NCS_EVENT_CONNECT_INFO *connects,

DWORD user_data);
```

对于通过窗口资源 ID 创建对话框的方法类似于主窗口，通过资源 ID 创建界面，参数含义也与上面创建主窗口的接口类似：

```
int ncsCreateModalDialogFromID (HPACKAGE package, Uint32 dlgId,

HWND owner, HICON hIcon, HMENU hMenu,

NCS_EVENT_HANDLER_INFO* handlers, NCS_EVENT_CONNECT_INFO *connects);
```

通过资源包创建 UI 主窗口的示例代码：

```


return ncsCreateMainWindowIndirectFromID(package,

ID_MAINWND1,

hParent,

h_icon,

h_menu,

mainwnd_Mainwnd1_handlers,

NULL,

user_data);
```

获取字符串

通过资源包获取字符串的方法很简单，只需要将资源 ID 传递给接口 ncsGetString，管理模块即会根据当前的 locale 信息，将默认 locale 的字符串返回给应用。

```
const char* ncsGetString (HPACKAGE package, Uint32 resId);
```

通过资源包获取字符串的示例代码：

```


SetDefaultWindowElementRenderer(ncsGetString(hPackage, NCSRM_SYSSTR_DEFRDR));
```

- 注：mGNCS 保留从 NCSRM_SYSSTR_BASEID 到 NCSRM_SYSSTR_MAXID 的 ID 值作为系统 ID，可以返回公共的定义，如默认渲染器：NCSRM_SYSSTR_DEFRDR

获取位图

资源管理模块提供了通过位图 ID 获取相应文件名的功能，便于上层应用在获取文件名后直接使用 MiniGUI 提供的位图装载相关接口操作位图：

```
const char* ncsGetImageFileName (HPACKAGE package, Uint32 resId);
```

除了使用 MiniGUI 相关接口访问位图外， mGNCS 还提供了直接通过资源 ID 获取位图对象的接口，上层应用使用完该接口提供的位图资源后，需通过相对应的释放接口释放位图对象。这对设置无关和设置相关位图都有效。

```
//设置相关位图接口

int ncsGetBitmap(HDC hdc, HPACKAGE package, Uint32 resId,  PBITMAP pBitmap);

void ncsReleaseBitmap (PBITMAP pBitmap);



//设置无关位图接口

int ncsGetMyBitmap(HPACKAGE package, Uint32 resId,

PMYBITMAP myBmp, RGB* pal);

void ncsReleaseMyBitmap (PMYBITMAP myBmp);
```

设置渲染器

通过资源包中所包含的渲染器 ID 可以对指定的窗口进行设置，使其具有该渲染器配置下的窗口元素外观属性。但该接口仅在当前窗口所使用的渲染器与新设置的渲染器同名，并且该窗口类是新渲染器所属控件类的子类时方可生效，否则将返回失败，设置不成功。

```
BOOL ncsSetWinRdr(HWND hWnd, HPACKAGE package, Uint32 rdrId);
```

另外还可以通过渲染器集合资源设置将某一渲染器集合作为默认系统渲染器配置。

```
BOOL ncsSetSysRdr(HPACKAGE package, Uint32 rdrSetId);
```

示例程序

本实例为用户演示了如何使用资源管理部分接口获取资源。该部分代码由 miniStudio 工具自动生成。

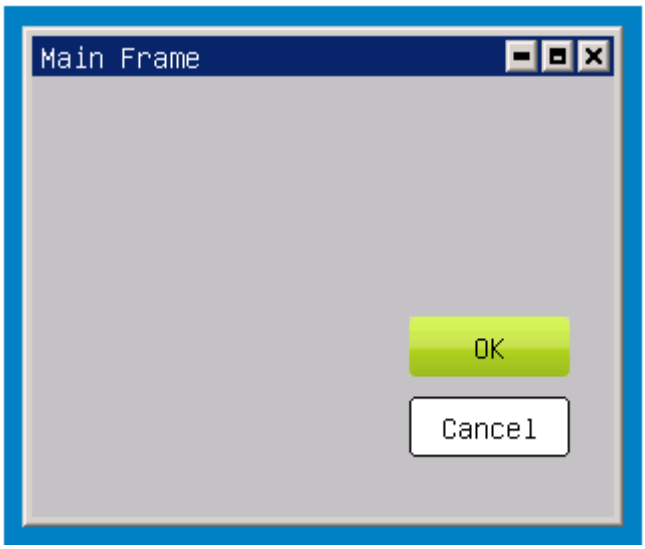


图 p2c2-1 resmgr 程序的输出

该程序在屏幕上创建一个包含 2 个按钮的 300x250 的对话框，其中主窗口是 classic 渲染器的外观效果，2 个按钮分别是 skin 和 flat 渲染器的外观效果。

清单 p2c2-2 resmgr_main.c

```
/*

** $Id: resmgr_main.c 534 2009-09-22 05:37:46Z xwyan $

**

** Listing P2C2.2
```

```
**
** resmgr_main.c: Sample program for mGNCS Programming Guide
**      The application entry of resource managerment.
**
** Copyright (C) 2009 Feynman Software.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>

#include <mgncs/mgncs.h>

#include "resource.h"
#include "ncs-windows.h"

HPACKAGE hPackage = HPACKAGE_NULL;

int MiniGUIMain(int argc, const char* argv[])
```

```
{
        #ifdef ntStartWindowEx
        MSG Msg;
        char f_package[MAX_PATH];
        mMainWnd *mWin;

        ncsInitialize();
        // START_OF_LOADRESPKG
        sprintf(f_package, "%s", "resmgr_demo.res");
        SetResPath("./");

        hPackage = ncsLoadResPackage (f_package);
        if (hPackage == HPACKAGE_NULL) {
                printf ("load resource package:%s failure.\n", f_package);
                return 1;
        }
        // END_OF_LOADRESPKG

        // START_OF_GETSTRING
        SetDefaultWindowElementRenderer(ncsGetString(hPackage, NCSRM_SYSSTR_DEFRDR));
        // END_OF_GETSTRING

        mWin = ntStartWindowEx(hPackage, HWND_DESKTOP, (HICON)0, (HMENU)0, (DWORD)0);
```

```
        while(GetMessage(&Msg, mWin->hwnd))
        {
                TranslateMessage(&Msg);
                DispatchMessage(&Msg);
        }

        MainWindowThreadCleanup(mWin->hwnd);
        // START_OF_UNLOADPKG
        ncsUnloadResPackage(hPackage);
        // END_OF_UNLOADPKG
        ncsUninitialize();
        #endif

        return 0;
}
```

清单 p2c2-3 resmgr.c

```
/*
** $Id: resmgr.c 534 2009-09-22 05:37:46Z xwyan $
**
** Listing P2C2.3
**
** resmgr.c: Sample program for mGNCS Programming Guide
**      Create main window by resource managerment.
**
```

```
** Copyright (C) 2009 Feynman Software.

*/


#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#include <minigui/common.h>

#include <minigui/minigui.h>

#include <minigui/gdi.h>

#include <minigui/window.h>


#include <mgncs/mgncs.h>


#include "resource.h"

#include "ncs-windows.h"


static NCS_EVENT_HANDLER_INFO mainwnd_Mainwnd1_handlers[] = {


        {-1, NULL}

};


NCS_WND_EXPORT mMainWnd* ntCreateMainwnd1Ex(HPACKAGE package, HWND hParent, HICON h_icon, HMENU
h_menu, DWORD user_data)

{
```

```
        // START_OF_UIWINDOW
        return ncsCreateMainWindowIndirectFromID(package,
        ID_MAINWND1,
        hParent,
        h_icon,
        h_menu,
        mainwnd_Mainwnd1_handlers,
        NULL,
        user_data);
        // END_OF_UIWINDOW
}
```

第二部分第三章 基础类介绍

mObject

mObject 介绍

mObject 是所有 mGNCS 的基础类，它：

- 封装了 mGNCS 继承和虚函数实现
- 定义了最基本的类成员结构

同时，定义了一组类操作函数，以控制类的行为

该类为基础类，不能直接使用

- 直接子类:
  - mComponent

mObject 操作函数

在 NCS 中，诸多关于类的操作都是以 mObject 指针为参数，这些有

- 类型判断操作: ncsInstanceOf, 相当于 java 的 instanceof 操作符，判断一个指针是否是指定的类的实例

```
/**
* \fn BOOL ncsInstanceOf(mObject* object, mObjectClass* clss);
* \brief check an object is the class instance or not,
*          same as \b instanceof operator in Java
*
* \param object - the pointer of object being to test
* \param clss - the pointer of class for test
```

```
*

* \return TRUE - object is instance of clss, FALSE - not

*/

BOOL ncsInstanceOf(mObject* object, mObjectClass* clss);
```

为方便操作，提供宏

```
#define INSTANCEOF(obj, clss)  \

ncsInstanceOf((mObject*)(obj), \

(mObjectClass*)(void*)(&Class(clss)))
```

使用举例

```


if(INSTANCEOF(obj, mWidget)) //判断对象 obj 是否是一个 mWidget 对象

.....
```

另外，为判断一个指针是否是 mObject 对象，可以使用 **ncsIsValidObj**

```
/**

* \fn static inline mObject*  ncsIsValidObj(mObject* obj);

* \brief Check a pointer is a valid mObject or not

*

* \param obj - the excpeted object pointer

*

* \return mObject * the pointer of obj or other NULL if obj is an invalid mObject pointer

*/

static inline mObject*  ncsIsValidObj(mObject* obj){

        return (INSTANCEOF(obj, mObject)(obj):NULL);

}
```

对应的宏是

```
/**
* \def CHECKOBJ
* \brief the wrapper of ncsIsValidObj
*
* \sa ncsIsValidObj
*/
#define CHECKOBJ(obj)  ncsIsValidObj((mObject*)obj)
```

- 类型转换操作 ：ncsSafeCast, 安全的类型转换，类似 C++的 dynamic_cast 操作符

```
/**
* \fn mObject* ncsSafeCast(mObject* obj, mObjectClass *clss);
* \brief safe type cast function, like the \b dynamic_cast operator in C++
*
* \param obj - the mObject pointer being casted
* \param clss - the target type to cast
*
* \return mObject * - the object pointer if cast safe, NULL otherwise
*
* \sa ncsInstanceOf
*/
mObject* ncsSafeCast(mObject* obj, mObjectClass *clss);
```

对应的宏是

```
/**
* \def SAFE_CAST
```

```
* \brief wrapper of ncsSafeCast, check the class type before cast.
*
* \note this macro is same as \b dynamic_cast in C++
*
* \sa ncsSafeCast
*/
#define SAFE_CAST(Type, obj)  \
TYPE_CAST(Type, ncsSafeCast((mObject*)obj,(mObjectClass*)(void*)(&(Class(Type)))))
```

使用举例

```
//将一个 obj 转为 mWidget 指针
mWidget * widget = SAFE_CAST(mWidget, obj);

//如果 obj 是一个 mWidget 类型或者 mWidget 子类类型，则转换成功，否则，widget == NULL
if(widget)
....
```

- 其他函数 ：newObject, deleteObject, TYPENAME 宏。

```
/**
* \fn mObject * newObject(mObjectClass *_class);
* \brief new a object instance, like \b new operator in C++
*
* \param _class - the class of the object
*
* \return the new pointer of object
```

```
*

*/

mObject * newObject(mObjectClass *_class);

/**

* \fn void deleteObject(mObject *obj);

* \brief delete a object intance, like \b delete operator in C++

*

* \param obj - the object want to delete

*

*/

void deleteObject(mObject *obj);



/**

* \def TYPENAME

* \brief Get the class name form a Object pointer

*/

#define TYPENAME(obj)  ((obj)(((obj)->_class)((obj)->_class)->typeName:"):")
```

创建一个对象，要用 newObject,因为它会调用对象的构造函数。

删除一个对象，要使用 deleteObject,因为他会调用对象的析构函数。

mComponent

mComponent 介绍

mComponent 提供了组件最基本的实现。组件是构成 mGNCS 程序最基本的原件

该类为基础类，不能直接使用

- 继承自: mObject
- 直接子类
  - mWidget
  - mInvsbComp

mComponent 方法

- setProperty

```
BOOL (*setProperty)(clss *_this, int id, DWORD value);
```

  -
  - 设置组件的属性
  - Return : TRUE -- 设置成功；FALSE--设置失败
  - Params
    - int id - 属性的 ID
    - DWORD value - 属性值

- getProperty

```
DWORD (*getProperty)(clss *_this, int id);
```

  -
  - 获取组件的属性
  - Return: 属性值或者 DWORD(-1)
  - Params
    - int id - 属性 ID

- setId

```
int (*setId)(clss *_this, int id);
```

  -
  - 设置组件的 Id
  - Return : 返回老的 Id
  - Params
    - int id - new id

- getId

```
int (*getId)(clss *_this);
```

  -

- 获取组件 Id
- Return : 组件 Id

- getReleated

```
mComponent* (*getReleated)(clss*_this, int releated);
```

-
- 取得和该组件相关的组件，例如父组件，子组件和兄弟组件
- Return : NULL 或者对应的组件对象指针
- Params:
  - int releated: 关系类型 :NCS_COMP_NEXT, NCS_COMP_PREV,NCS_COMP_PARENT,NCS_COMP_CHILDREN 之一

- setReleated:

```
mComponent* (*setReleated)(clss *_this, mComponent* comp,  int releated);
```

-
- 设置关联组件
- Return : 设置后的关联组件指针；如果没有设置成功，返回 NULL
- Params
  - mComponent *comp - 被设置的组件指针
  - int releated - 同 getReletaed

- getChild

```
mComponent* (*getChild)(clss* _this, int id);
```

-
- 获取 id 指定的子组件
- Return : NULL 或者对应的组件指针
- Params:
  - int id - 要获取的组件的 id

mComponent 操作函数

Component 支持一些通用的操作

- 事件安装和卸载函数

```
/**
```

```
* A struct of event-handler map
*
* \note only used for param
*
* \sa NCS_EVENT_HANDLER_NODE
*/
typedef struct _NCS_EVENT_HANDLER {
        /**
        * The event code
        */
        int message;
        /**
        * The event callback pointer
        */
        void *handler;
}NCS_EVENT_HANDLER;


/**
* \fn void * ncsSetComponentHandler(mComponent* comp, int message, void *handler);
* \brief set the component handler
*
* \param comp the compont to set
* \param message the event code
* \param handler the event callback pointer
```

```
*
* \return old event callback if it has been set
*
* \sa ncsSetComponentHandlers
*/
void * ncsSetComponentHandler(mComponent* comp, int message, void *handler);


/**
* \fn void ncsSetComponentHandlers(mComponent* comp, \
NCS_EVENT_HANDLER* handlers, \
int count);
* \brief set an array of event handlers
*
* \param comp - the component to set
* \param handlers - the array of \ref NCS_EVENT_HANDLER
* \param count - the count of array handlers.
*
* \note if count == -1, handlers must end by {-1, NULL};
*   anywhere, ncsSetComponentHandlers would stop
*   if it find an element of array handlers is equal {-1, NULL},
*   whether count is equal -1 or not
*
* \sa ncsSetComponentHandler
*/
```

```
void ncsSetComponentHandlers(mComponent* comp, \
NCS_EVENT_HANDLER* handlers, \
int count);


/**
* \fn void* ncsGetComponentHandler(mComponent* comp, int message);
* \brief get an event callback
*
* \param comp
* \param message - event code
*
* \return void * the handler of message, or NULL if not set
*/
void* ncsGetComponentHandler(mComponent* comp, int message);
```

- 注册相关函数

```
/**
* \fn BOOL ncsRegisterComponent(mComponentClass *compCls, \
DWORD dwStyle, \
DWORD dwExStyle, \
int idCursor, \
int idBkColor);
* \brief register a component class into MiniGUI, so that \ref ncsCreateWindow and
*     \ref ncsCreateWindow can find a \mComponentClass instance
*
```

```
* \param compCls the \ref mComponentClass to be registered
* \param dwStyle the default style
* \param dwExStyle the default extend style
* \param idCursor the default cursor
* \param idBkColor the default background color
*
* \return TRUE - success, FALSE - failed
*
* \sa ncsGetComponentClass, ncsCreateWindow,
*       ncsCreateWindowIndirect, ncsCreateMainWindow,
*       ncsCreateMainWindowIndirect
*/
BOOL ncsRegisterComponent(mComponentClass *compCls, \
DWORD dwStyle, \
DWORD dwExStyle, \
int idCursor, \
int idBkColor);


/**
* \fn mComponentClass * ncsGetComponentClass(const char* class_name, BOOL check);
* \brief Get a \ref mComponentClass instance from MiniGUI
*
* \note the class_name must be registered into MiniGUI by calling ncsRegisterComponent
*
```

```
* \param class_name the class name to find

* \param check check the class name with found mComponentClass instance,

*              to ensure that we found the right class

*

* \return the mComponentClass pointer if sucess, NULL otherwise

*/

mComponentClass * ncsGetComponentClass(const char* class_name, BOOL check);
```

mWidget

mWidget 介绍

mWidget 是所有控件的基础类

- 继承自: mComponent
- 直接子类
  - mStatic
  - mButton
  - mPanel
  - mScrollWidget
  - mProgressBar
  - mPropSheet
  - mSlider
  - mSpinbox

mWidget 风格

风格 ID	miniStudio 属性名	说明
NCSS_NOTIFY	Notify	决定控件是否产生 Notification 事件

mWidget 属性

属性名	miniStudio 属性	类型	RW	说明

	名			
NCSP_WIDGET_RDR	Renderer	const char*	W	设置控件当前渲染器
NCSP_WIDGET_TEXT	Text	const char*	W	设置当前控件的文本内容
NCSP_WIDGET_BKIMAGE	BkImage	PBITMAP	RW	设置或获取当前背景图片
NCSP_WIDGET_BKIMAGE_MODE	BkImageMode	ImageDrawMode	设置或者获取当前背景图片绘制模式	
NCSP_WIDGET_BKIMAGE_FILE	-	const char*	设置当前背景图片，自动从文件名加载	

mWidget 方法

无

mWidget 事件

- MSG_NCCREATE
  - 描述: 当窗口非客户区创建时。窗口的第一个消息，此时窗口尚未建成，以窗口句柄为参数的函数不能调用
  - 回调: void (* NCS_CB_ONNCCREATE)(mWidget *);
    - 返回值: 无
    - 参数
      - mWidget * 事件产生者对象指针

- MSG_CREATE
  - 描述: 当窗口创建时产生。
  - 回调: typedef BOOL(*) NCS_CB_ONCREATE (mWidget *, DWORD dwAddData)
    - 返回值: TRUE - 继续创建了；FALSE - 退出创建
    - 参数
      - mWidget *
      - DWORD dwAddData 附加数据

- 通知消息
  - 所有通知消息产生的事件都使用该回调

- void (* NCS_CB_NOTIFY)(mWidget *, int nc_code);
  - 返回值: 无
  - 参数
    - mWidget * 事件产生者对象指针
    - int nc_code : 事件通知码，用于区别不通的事件

注，所有通知事件的回调都是 NCS_CB_NOTIFY

mWidget 操作函数

- 创建控件的函数

```
/**
* A struct wrap the NCS_CREATE_INFO
*
* \note only allowed using in \ref ncsCreateMainWindow
* Don't use it directly
*
* \sa NCS_CREATE_INFO, ncsCreateMainWindow , ncsCreateMainWindowIndirect
*/
typedef struct _NCS_MAIN_CREATE_INFO {
        /**
        * The class name of a mMainWnd or its child class
        *
        * \note if className is NULL or an invalidate class name
        * \ref ncsCreateMainWindow and ncsCreateMainWindowIndirect
        *        use \ref CTRL_MINIMAINWND replaced
        *
        * \sa CTRL_MINIMAINWND
```

```
        */
        const char* className;
        /**
        * NCS_CREATE_INFO pointer
        */
        NCS_CREATE_INFO * create_info;
}NCS_MAIN_CREATE_INFO;


/**
* \fn mWidget* ncsCreateMainWindow (const char *class_name, const char *caption,
*                         DWORD style, DWORD ex_style, \
*                         int id, int x, int y, int w, int h, HWND host, \
*                         HICON hIcon, HMENU hMenu, NCS_PROP_ENTRY * props, \
*                         NCS_RDR_INFO * rdr_info, \
*                         NCS_EVENT_HANDLER * handlers, \
*                         DWORD add_data);
*
*        \brief create a NCS main window
*
*        \param class_name the class name of widget.
*               the class name must be register by \ref ncsRegisterComponent.
*               And must be \ref CTRL_MINIMAINWND or its dirved class.
*        \param caption the caption of the main window
*        \param style  the style of main window
```

```
*        \param ex_style  the extend style of main window
*        \param id the id of main window
*        \param x the x position of main window
*        \param y the y position of main window
*        \param w the width of main window
*        \param h the height of main window
*  \param host the handle of host window, can be NULL
*  \param hIcon the icon of main window
*  \param hMenu the menu bar handle
*  \param props the properties array pointer, end by {-1, 0} if it's not NULL
*  \param rdr_info the renderer info pointer
*  \param handlers the handlers of event array pointer,
*               end by {-1, NULL}, if it's not NULL
*  \param add_data the additional data send to callback \ref NCS_CB_ONCREATE
*               and \ref NCS_CB_ONINITDLG
*
*  \return mWidget* - a mWidget pointer, must be a mMainWnd instance
*
*  \sa ncsCreateWindow,ncsCreateWindowIndirect, nscCreateMainWindowIndirect,
*        NCS_PROP_ENTRY, NCS_RDR_INFO, NCS_EVENT_HANDLER, NCS_CB_ONCREATE,
*        mWidget, mInvisibleComponent , mMainWnd
*
*
*/
```

```
mWidget* ncsCreateMainWindow (const char *class_name, const char *caption,
DWORD style, DWORD ex_style, \
int id, int x, int y, int w, int h, HWND host, \
HICON hIcon, HMENU hMenu,
NCS_PROP_ENTRY * props, \
NCS_RDR_INFO * rdr_info, \
NCS_EVENT_HANDLER * handlers, \
DWORD add_data);


/**
* A struct include all the creating info, used by ncsCreateWindowIndirect
*
* \sa NCS_CREATE_INFO
*/
struct _NCS_WND_TEMPLATE{
        /**
        * The class name of mComponent, must be registered by \ref ncsRegisterComponent
        *
        * \note support \ref mInvisibleComponent class
        */
        const char*         class_name;
        /**
        * The id of commponet
        */
```

```
    int                 id;

    /**
    * The Location and Size of mWidget, ignored if class_name
    *  is a \ref mInvisibleComponent
    */
    int                 x, y, w, h;

    /**
    * The style of mWidget, ignored if class_name is a \ref mInvisibleComponent
    */
    DWORD               style;

    /**
    * The extend style of mWidget, ignored if class_name is a \ref mInvisibleComponent
    */
    DWORD               ex_style;

    /**
    * The caption of mWidget, ignored if class_name is a \ref mInvisibleComponent
    */
    const char*         caption;


    //same struct as NCS_CREATE_INFO
    /**
    * Same as NCS_CREATE_INFO
    *
    * \sa NCS_CREATE_INFO
```

```
        */
        NCS_PROP_ENTRY*     props;
        NCS_RDR_INFO*       rdr_info;
        NCS_EVENT_HANDLER*  handlers;
        NCS_WND_TEMPLATE*   ctrls;
        int                 count;
        DWORD               user_data;

        //FIXED ME Maybe I  should not put these two param here
        DWORD                           bk_color;
        PLOGFONT                    font;
};

//create control window indirect
/**
* \fn mWidget* ncsCreateWindowIndirect( const NCS_WND_TEMPLATE* tmpl, HWND hParent);
* \brief create a mComponent by \ref NCS_WND_TEMPLATE,
*
* \param tmpl the template pointer
* \param hParent the parent handle, if NCS_WND_TEMPLATE.class_name
*              is a \ref mInvisibleComponent, hParent must releated a mWidget object
*
* \return mWidget* - then pointer of object or NULL if failed
*
```

```
* \sa ncsCreateWindow, NCS_WND_TEMPLATE
*/
mWidget* ncsCreateWindowIndirect( const NCS_WND_TEMPLATE* tmpl, HWND hParent);

/**
* A struct include all the creating info for ncsCreateMainWindowIndirect,
*
* \note same as \ref ncsCreateMainWindow 's params
*
* \sa NCS_WND_TEMPLATE, ncsCreateMainWindow
*/
typedef struct _NCS_MAINWND_TEMPLATE{
        const char*         class_name;
        int                 id;
        int                 x, y, w, h;
        DWORD               style;
        DWORD               ex_style;
        const char*         caption;

        NCS_PROP_ENTRY*     props;
        NCS_RDR_INFO*       rdr_info;
        NCS_EVENT_HANDLER*  handlers;
        NCS_WND_TEMPLATE*   ctrls;
        int                 count;
```

```
        DWORD               user_data;


        //FIXED ME Maybe I  should not put these two param here
        DWORD                               bk_color;
        PLOGFONT                    font;


        HICON               hIcon;
        HMENU               hMenu;
}NCS_MAINWND_TEMPLATE;


/**
* \fn mWidget* ncsCreateMainWindowIndirect(const NCS_MAINWND_TEMPLATE* tmpl, HWND hHost);
* \biref create a main window from a template
*
* \param tmpl - the template of main window
* \param hHost - the host window handler of the main window
*
* \return mWidget * - the Instance of mMainWnd or NULL
*
* \sa ncsCreateMainWindow, NCS_MAINWND_TEMPLATE
*/
mWidget* ncsCreateMainWindowIndirect(const NCS_MAINWND_TEMPLATE* tmpl, HWND hHost);
```

- 对象指针和窗口句柄直接的关联操作

```
/**
```

```
* \fn static inline mWidget* ncsObjFromHandle(HWND hwnd);
* \brief Get a Object from window handle
*
* \param hwnd - the handle of window
*
* \return mWidget * the instance releated this handle, or NULL
*/
static inline mWidget* ncsObjFromHandle(HWND hwnd)
{
        if(IsWindow(hwnd))
        return (mWidget*)(GetWindowAdditionalData2(hwnd));
        return NULL;
}

/**
* \fn static inline mWidget* ncsGetChildObj(HWND hwnd, int id);
* \breif Get the child object pointer of window
*
* \param hwnd - the handle of window, which can be a normal
*                          MiniGUI window or a NCS window
* \param id - the child id. The child must be a NCS window
*
* \return mWidget *  the instance releated id, or NULL
*/
```

```
static inline mWidget* ncsGetChildObj(HWND hwnd, int id)
{
        return ncsObjFromHandle(GetDlgItem(hwnd,id));
}

/**
* \fn static inline mWidget* ncsGetParentObj(HWND hwnd)
* \brief Get the parent object pointer of window
*
* \param hwnd - the handle of child window, which can be a
*                          normal MiniGUI window or a NCS window
*
* \return mWidget * the instance of parent of hwnd
*/
static inline mWidget* ncsGetParentObj(HWND hwnd)
{
        return ncsObjFromHandle(GetParent(hwnd));
}
```

mWidget 示例

不能直接使用 mWidget

第二部分第四章 静态框系列控件类

静态框系列简介

静态框是用来在窗口的特定位置显示数字、文字和图片等信息，比如公司简介、公司产品商标等等，是最常见的控件之一。一般来说，静态框系列的控件行是不需要对用户的输入进行动态的响应，也就是说不需要接收任何输入（如键盘鼠标等），也就没有自己的事件。

控件中文名称	控件名称 （miniStudio 显示名）	用途	NCS 类名	控件窗口类 ID
静态框	Label	显示单行或多行文本	mStatic	NCSCTRL_STATIC
图片框	Image	显示图片	mImage	NCSCTRL_IMAGE
矩形框	Rectangle	绘制矩形	mRect	NCSCTRL_RECTANGLE
分组框	Group Box	可包含其他控件的区域控件	mGroupbox	NCSCTRL_GROUPBOX
按钮组	Button Group	管理 RadioButton，实现单选功能	mButtonGroup	NCSCTRL_BUTTONGROUP
发光二极管标签	LEDLabel	显示类 LED 字符	mLEDLabel	NCSCTRL_LEDLABEL
分隔栏	Horz/Vert Separator	分隔线，视觉上实现区域划分	mSeparator	NCSCTRL_SEPARATOR

静态框系列控件的继承关系如下：

- mStatic
  - mImage
  - mRect
  - mGroupbox
    - mButtonGroup
  - mLEDLabel
  - mSeparator

每种类型的静态框均继承父类的属性、事件、方法，因此在下面的介绍中将自动忽略继承的部分。

mStatic

- 功能：mStatic 是用来绘制包含文本，图片等内容的静态区域控件。
- 父类：mWidget
- 直接子类：
  - mImage
  - mRect
  - mGroupbox
  - mLEDLabel
  - mSeparator

mStatic 风格

无

mStatic 属性

属性名	类型	权限	属性说明	模式取值
NCSP_STATIC_ALIGN	int	RW	控件内容水平对齐模式	NCS_ALIGN_LEFT, NCS_ALIGN_RIGHT, NCS_ALIGN_CENTER
NCSP_STATIC_VALIGN	int	RW	控件内容垂直对齐模式	NCS_VALIGN_TOP, NCS_VALIGN_BOTTOM, NCS_VALIGN_CENTER
NCSP_STATIC_AUTOWRAP	int	RW	控件内容自动换行	1, 0

- NCSP_STATIC_ALIGN：设置控件内容为水平对齐模式，共有 3 种取值
  - NCS_ALIGN_LEFT：左对齐，水平对齐模式的默认取值 - 0
  - NCS_ALIGN_RIGHT：右对齐 - 1
  - NCS_ALIGN_CENTER：居中对齐 - 2
- NCSP_STATIC_VALIGN：设置控件内容为垂直对齐模式，共有 3 种取值
  - NCS_VALIGN_TOP：顶端对齐 - 0
  - NCS_VALIGN_BOTTOM：底端对齐，垂直对齐模式的默认取值 - 1
  - NCS_VALIGN_CENTER：居中对齐 - 2

- NCSP_STATIC_AUTOWRAP：设置静态框内容是否为自动换行模式，0 为单行模式即关闭自动换行模式，1 为自动换行模式

开发者可以通过以下方法设置 mStatic 的属性

```
//static 控件属性配置

static NCS_PROP_ENTRY static1_props [] = {

        { NCSP_STATIC_ALIGN, NCS_ALIGN_CENTER },     //对齐方式

        {0, 0}

};

...

static NCS_PROP_ENTRY static4_props [] = {

        { NCSP_STATIC_VALIGN, NCS_VALIGN_TOP },

        {0, 0}

};

...

static NCS_WND_TEMPLATE _ctrl_templ[] = {

        {

                NCSCTRL_STATIC,                          //控件名称

                IDC_STATIC1+0,

                10, 10, 160, 30,                          //控件位置, 大小

                WS_BORDER | WS_VISIBLE,

                WS_EX_NONE,

                "Default Text",                    //控件内文本内容

                NULL, //props,

                NULL, //rdr_info
```

```
            NULL, //handlers,
            NULL, //controls
            0,
            0 //add data
        },
        {
            NCSCTRL_STATIC,
            IDC_STATIC1+1,
            10, 50, 160, 30,
            WS_BORDER | WS_VISIBLE,
            WS_EX_NONE,
            "Center Text",
            static1_props, //props, 设置属性
            NULL, //rdr_info
            NULL, //handlers,
            NULL, //controls
            0,
            0 //add data
        },
        ...
```

mStatic 事件

目前静态框系列控件不响应任何事件。

mStatic 方法

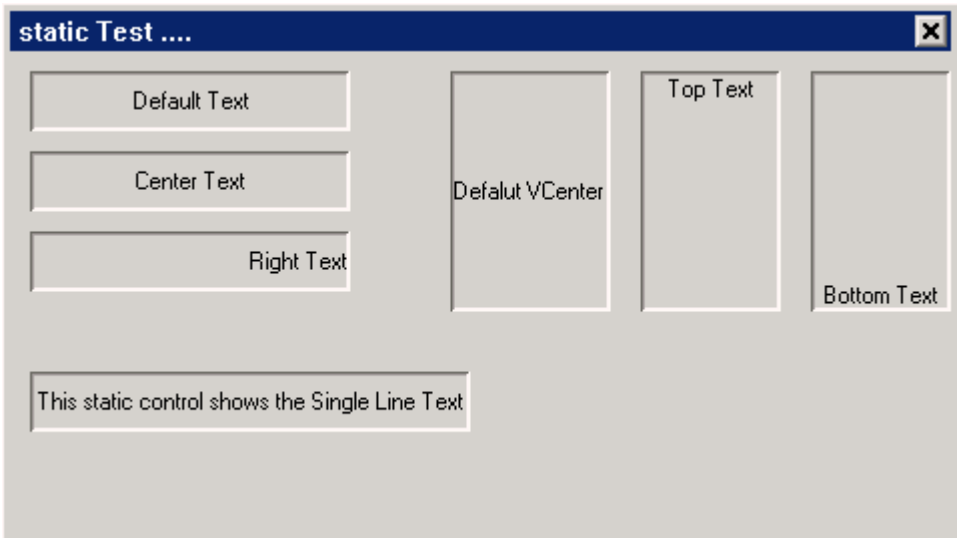
无

mStatic 渲染器

无

mStatic 实例

本实例为用户演示了如何创建不同模式的静态框。



清单 p2c4-1 static.c

```
/*
* ** $Id$
* **
* ** Listing P2C4.1
* **
* ** static.c: Sample program for mGNCS Programming Guide
* **      A mGNCS application for mStatic.
* **
```

```
* ** Copyright (C) 2009 Feynman Software.
* */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/control.h>
#include <mgncs/mgncs.h>

#define IDC_STATIC1 100
#define IDC_SATAICN 107

static BOOL mymain_onCreate(mWidget* self, DWORD add_data)
{
        //TODO : initialize
        return TRUE;
}

static void mymain_onClose(mWidget* self, int message)
```

```
{
        DestroyMainWindow(self->hwnd);
        PostQuitMessage(0);
}



//Propties for
static NCS_PROP_ENTRY static1_props [] = {
        { NCSP_STATIC_ALIGN, NCS_ALIGN_CENTER },
        {0, 0}
};
static NCS_PROP_ENTRY static2_props [] = {
        { NCSP_STATIC_ALIGN, NCS_ALIGN_RIGHT },
        {0, 0}
};
static NCS_PROP_ENTRY static4_props [] = {
        { NCSP_STATIC_VALIGN, NCS_VALIGN_TOP },
        {0, 0}
};


static NCS_PROP_ENTRY static5_props [] = {
        { NCSP_STATIC_VALIGN, NCS_VALIGN_BOTTOM },
        {0, 0}
};
```

```
static NCS_PROP_ENTRY static6_props [] = {
        { NCSP_STATIC_AUTOWRAP, 0 },
        {0, 0}
};



//Controls
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_STATIC ,
                IDC_STATIC1+0,
                10, 10, 160, 30,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
                "Default Text",
                NULL, //props,
                NULL, //rdr_info
                NULL, //handlers,
                NULL, //controls
                0,
                0 //add data
        },
        {
```

```
        NCSCTRL_STATIC ,
        IDC_STATIC1+1,
        10, 50, 160, 30,
        WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Center Text",
        static1_props, //props,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_STATIC ,
        IDC_STATIC1+2,
        10, 90, 160, 30,
        WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Right Text",
        static2_props, //props,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
```

```
            0,
            0 //add data
        },
        {
            NCSCTRL_STATIC ,
            IDC_STATIC1+3,
            220, 10, 80, 120,
            WS_BORDER | WS_VISIBLE,
            WS_EX_NONE,
            "Defalut VCenter ",
            NULL, //props,
            NULL, //rdr_info
            NULL, //handlers,
            NULL, //controls
            0,
            0 //add data
        },
        {
            NCSCTRL_STATIC ,
            IDC_STATIC1+4,
            315, 10, 70, 120,
            WS_BORDER | WS_VISIBLE,
            WS_EX_NONE,
            "Top Text",
```

```
            static4_props, //props,
            NULL, //rdr_info
            NULL, //handlers,
            NULL, //controls
            0,
            0 //add data
        },
        {
            NCSCTRL_STATIC ,
            IDC_STATIC1+5,
            400, 10, 70, 120,
            WS_BORDER | WS_VISIBLE,
            WS_EX_NONE,
            "Bottom Text",
            static5_props, //props,
            NULL, //rdr_info
            NULL, //handlers,
            NULL, //controls
            0,
            0 //add data
        },
        {
            NCSCTRL_STATIC ,
            IDC_STATIC1+6,
```

```
                10, 160, 220, 30,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
                "This static control shows the Single Line Text",
                static6_props, //props,
                NULL, //rdr_info
                NULL, //handlers,
                NULL, //controls
                0,
                0 //add data
        },

};


static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate},
        {MSG_CLOSE, mymain_onClose},
        {0, NULL}
};

//define the main window template
static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,
```

```
        1,
        0, 0, 480, 270,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "static Test ....",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};

int MiniGUIMain(int argc, const char* argv[])
{
        ncsInitialize();
        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);

        _c(mydlg)->doModal(mydlg, TRUE);


        MainWindowThreadCleanup(mydlg->hwnd);
```

```
        return 0;
}


#ifdef _MGRM_THREADS
#include <minigui/dti.c>
#endif
```

mImage

- 功能: 在一区域内加载显示图片控件
- 继承自: mStatic
- 直接子类: 无

mImage 属性

属性名	类型	权限	属性说明	取值
NCSP_IMAGE_IMAGE	PBITMAP	RW	image 控件内容图片 id，对应 mimage 中的 pbmp 图片文件指针	无
NCSP_IMAGE_IMAGEFILE	char*	RW	image 控件内容图片名字 id，对应 pbmp 图片的名字指针	无
NCSP_IMAGE_DRAWMODE	enum	RW	image 控件绘制模式 id，对应 mImageDrawMode	NCS_DM_NORMAL, NCS_DM_SCALED, NCS_DM_SCALED

NCSP_STATIC_ALIGN	int	RW	设置 image 控件内容水平对齐模式	NCS_ALIGN_LEFT, NCS_ALIGN_RIGHT, NCS_ALIGN_CENTER
NCSP_STATIC_VALIGN	int	RW	设置 image 控件内容垂直对齐模式	NCS_VALIGN_TOP, NCS_VALIGN_BOTTOM, NCS_VALIGN_CENTER

- NCSP_IMAGE_DRAWMODE：设置图片绘制模式，共有三种方式。
  - NCS_DM_NORMAL：按原始情况显示图片，绘制模式的默认取值 - 0
  - NCS_DM_SCALED： 拉伸，将图片拉伸成和以覆盖整个静态框 - 1
  - NCS_DM_TILED：平铺，将图片重复显示在整个静态框中 - 2
- NCSP_STATIC_ALIGN：参见 mStatic
- NCSP_STATIC_VALIGN：参见 mStatic

以下代码演示了如何设置绘制模式等属性

```
static BITMAP icon;

static BITMAP bitmap;



static void set_icon_info(mWidget* self, int id, PBITMAP pbmp, int align_id, int align)

{

        mImage *img;

        img = (mImage *)ncsGetChildObj(self->hwnd, id);



        if(img){

                _c(img)->setProperty(img, NCSP_IMAGE_IMAGE, (DWORD)pbmp);

                _c(img)->setProperty(img, align_id, align);

        }

}
```

```
static BOOL mymain_onCreate(mWidget* self, DWORD add_data)
{
        //TODO : initialize
        mImage *img;
        LoadBitmapFromFile(HDC_SCREEN, &bitmap, "image_test.jpg");

        LoadBitmapFromFile(HDC_SCREEN, &icon, "icon.png");

        set_icon_info(self, IDC_IMAGE1, &icon, NCSP_STATIC_ALIGN, NCS_ALIGN_LEFT);
        set_icon_info(self, IDC_IMAGE2, &icon, NCSP_STATIC_ALIGN, NCS_ALIGN_CENTER);
        set_icon_info(self, IDC_IMAGE3, &icon, NCSP_STATIC_ALIGN, NCS_ALIGN_RIGHT);
        set_icon_info(self, IDC_IMAGE4, &icon, NCSP_STATIC_VALIGN, NCS_VALIGN_TOP);
        set_icon_info(self, IDC_IMAGE5, &icon, NCSP_STATIC_VALIGN, NCS_VALIGN_CENTER);
        set_icon_info(self, IDC_IMAGE6, &icon, NCSP_STATIC_VALIGN, NCS_VALIGN_BOTTOM);

        img = (mImage *)ncsGetChildObj(self->hwnd, IDC_IMAGE7);
        if(img){
                _c(img)->setProperty(img, NCSP_IMAGE_IMAGE, (DWORD)&bitmap);
                _c(img)->setProperty(img, NCSP_IMAGE_DRAWMODE, NCS_DM_SCALED);
        }

        img = (mImage *)ncsGetChildObj(self->hwnd, IDC_IMAGE8);
        if(img){
                _c(img)->setProperty(img, NCSP_IMAGE_IMAGE, (DWORD)&bitmap);
```

```
                _c(img)->setProperty(img, NCSP_IMAGE_DRAWMODE, NCS_DM_TILED);
        }

        return TRUE;
}
```

mImage 事件

无

mImage 方法

无

mImage 实例

本实例为用户演示了如何显示并创建不同绘制模式的图片静态框

清单 p2c4-2 image.c

```
/*
* ** $Id$
* **
* ** Listing P2C4.2
* **
* ** image.c: Sample program for mGNCS Programming Guide
* **      A mGNCS application for mImage.
* **
* ** Copyright (C) 2009 Feynman Software.
* */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/control.h>
#include <mgncs/mgncs.h>

#define IDC_IMAGE1 100
#define IDC_IMAGE2 101
#define IDC_IMAGE3 102
#define IDC_IMAGE4 103
#define IDC_IMAGE5 104
#define IDC_IMAGE6 105
#define IDC_IMAGE7 106
#define IDC_IMAGE8 107

static BITMAP icon;
static BITMAP bitmap;
```

```
static void set_icon_info(mWidget* self, int id, PBITMAP pbmp, int align_id, int align)
{
        mImage *img;
        img = (mImage *)ncsGetChildObj(self->hwnd, id);

        if(img){
                _c(img)->setProperty(img, NCSP_IMAGE_IMAGE, (DWORD)pbmp);
                _c(img)->setProperty(img, align_id, align);
        }
}


static BOOL mymain_onCreate(mWidget* self, DWORD add_data)
{
        //TODO : initialize
        mImage *img;
        LoadBitmapFromFile(HDC_SCREEN, &bitmap, "image_test.jpg");

        LoadBitmapFromFile(HDC_SCREEN, &icon, "icon.png");

        set_icon_info(self, IDC_IMAGE1, &icon, NCSP_STATIC_ALIGN, NCS_ALIGN_LEFT);
        set_icon_info(self, IDC_IMAGE2, &icon, NCSP_STATIC_ALIGN, NCS_ALIGN_CENTER);
        set_icon_info(self, IDC_IMAGE3, &icon, NCSP_STATIC_ALIGN, NCS_ALIGN_RIGHT);
        set_icon_info(self, IDC_IMAGE4, &icon, NCSP_STATIC_VALIGN, NCS_VALIGN_TOP);
```

```
        set_icon_info(self, IDC_IMAGE5, &icon, NCSP_STATIC_VALIGN, NCS_VALIGN_CENTER);
        set_icon_info(self, IDC_IMAGE6, &icon, NCSP_STATIC_VALIGN, NCS_VALIGN_BOTTOM);

        img = (mImage *)ncsGetChildObj(self->hwnd, IDC_IMAGE7);
        if(img){
                _c(img)->setProperty(img, NCSP_IMAGE_IMAGE, (DWORD)&bitmap);
                _c(img)->setProperty(img, NCSP_IMAGE_DRAWMODE, NCS_DM_SCALED);
        }

        img = (mImage *)ncsGetChildObj(self->hwnd, IDC_IMAGE8);
        if(img){
                _c(img)->setProperty(img, NCSP_IMAGE_IMAGE, (DWORD)&bitmap);
                _c(img)->setProperty(img, NCSP_IMAGE_DRAWMODE, NCS_DM_TILED);
        }

        return TRUE;
}

static void mymain_onClose(mWidget* self, int message)
{
        DestroyMainWindow(self->hwnd);
        PostQuitMessage(0);
        UnloadBitmap(&bitmap);
}
```

```
//Propties for

//Controls
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_STATIC,
                0,
                10, 10, 100, 20,
                WS_VISIBLE,
                WS_EX_NONE,
                "image - left",
                NULL,
                NULL, //rdr_info
                NULL, //handlers,
                NULL, //controls
                0,
                0 //add data
        },
        {
                NCSCTRL_IMAGE,
                IDC_IMAGE1,
                10, 30, 100, 100,
```

```
            WS_BORDER | WS_VISIBLE,
            WS_EX_NONE,
            "",
            NULL,
            NULL, //rdr_info
            NULL, //handlers,
            NULL, //controls
            0,
            0 //add data
    },
    {
            NCSCTRL_STATIC,
            0,
            130, 10, 100, 20,
            WS_VISIBLE,
            WS_EX_NONE,
            "image - center",
            NULL,
            NULL, //rdr_info
            NULL, //handlers,
            NULL, //controls
            0,
            0 //add data
    },
```

```
    {
        NCSCTRL_IMAGE,
        IDC_IMAGE2,
        130, 30, 100, 100,
        WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "",
        NULL,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_STATIC,
        0,
        250, 10, 100, 20,
        WS_VISIBLE,
        WS_EX_NONE,
        "image - right",
        NULL,
        NULL, //rdr_info
        NULL, //handlers,
```

```
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_IMAGE,
        IDC_IMAGE3,
        250, 30, 100, 100,
        WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "",
        NULL,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_STATIC,
        0,
        10, 140, 100, 20,
        WS_VISIBLE,
        WS_EX_NONE,
```

```
        "image - top",
        NULL,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_IMAGE,
        IDC_IMAGE4,
        10, 160, 100, 100,
        WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "",
        NULL,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_STATIC,
```

```
        0,
        130, 140, 100, 20,
        WS_VISIBLE,
        WS_EX_NONE,
        "image - middle",
        NULL,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_IMAGE,
        IDC_IMAGE5,
        130, 160, 100, 100,
        WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "",
        NULL,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
```

```
        0 //add data
    },
    {
        NCSCTRL_STATIC,
        0,
        250, 140, 100, 20,
        WS_VISIBLE,
        WS_EX_NONE,
        "image - bottom",
        NULL,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_IMAGE,
        IDC_IMAGE6,
        250, 160, 100, 100,
        WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "",
        NULL,
```

```
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_STATIC,
        0,
        10, 270, 100, 20,
        WS_VISIBLE,
        WS_EX_NONE,
        "bitmap - scaled",
        NULL,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_IMAGE,
        IDC_IMAGE7,
        10, 290, 280, 150,
```

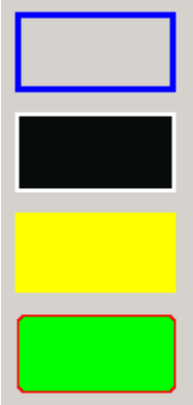
```
        WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "",
        NULL,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_STATIC,
        0,
        300, 270, 100, 20,
        WS_VISIBLE,
        WS_EX_NONE,
        "bitmap - tiled",
        NULL,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
```

```
        {
                NCSCTRL_IMAGE,
                IDC_IMAGE8,
                300, 290, 280, 150,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
                "",
                NULL,
                NULL, //rdr_info
                NULL, //handlers,
                NULL, //controls
                0,
                0 //add data
        },

};


static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate },
        {MSG_CLOSE, mymain_onClose },
        {0, NULL }
};
```

```
//define the main window template

static NCS_MNWND_TEMPLATE mymain_templ = {

        NCSCTRL_DIALOGBOX,

        1,

        0, 0, 600, 480,

        WS_CAPTION | WS_BORDER | WS_VISIBLE,

        WS_EX_NONE,

        "image Test ....",

        NULL,

        NULL,

        mymain_handlers,

        _ctrl_templ,

        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),

        0,

        0, 0,

};


int MiniGUIMain(int argc, const char* argv[])

{

        ncsInitialize();

        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect

        (&mymain_templ, HWND_DESKTOP);


        _c(mydlg)->doModal(mydlg, TRUE);
```

```
        MainWindowThreadCleanup(mydlg->hwnd);
        return 0;
}

#ifdef _MGRM_THREADS
#include <minigui/dti.c>
#endif
```

mRect



- 功能: 本控件为用户提供了绘制矩形的功能，可以通过设置属性的方法，方便快捷的绘制出具有填充色、圆角、边框等各种特色的矩形
- 继承自: mStatic
- 直接子类: 无

mRect 属性

属性名	类型	权限	说明
NCSP_RECT_BORDERSIZE	int	RW	rect 控件边框的粗细度，对应矩形控件的边框粗细，类型为 int
NCSP_RECT_BORDERCOLOR	dword	RW	rect 控件边框的颜色，对应矩形控件的边框颜色，类型为

			dword
NCSP_RECT_FILLCOLOR	dword	RW	rect 控件填充色，对应矩形控件的填充色，类型为 dword
NCSP_RECT_XRADIUS	int	RW	rect 控件圆角横向 x 半径，对应矩形控件的圆角半径，类型为 int
NCSP_RECT_YRADIUS	int	RW	rect 控件圆角竖向 y 半径，对应矩形控件的圆角半径，类型为 int
NCSP_RECT_FILLCLR_RED			
NCSP_RECT_FILLCLR_GREEN			
NCSP_RECT_FILLCLR_BLUE			
NCSP_RECT_FILLCLR_ALPHA			
NCSP_RECT_BRDCLR_RED			
NCSP_RECT_BRDCLR_GREEN			
NCSP_RECT_BRDCLR_BLUE			
NCSP_RECT_BRDCLR_ALPHA			

以下代码演示了如何设置矩形的属性

```
//Propties for

static NCS_PROP_ENTRY rect1_props [] = {

        {NCSP_RECTANGLE_BORDERSIZE, 3},

        {NCSP_RECTANGLE_BORDERCOLOR, 0xFFFF0000},

        {NCSP_RECTANGLE_FILLCOLOR, 0x00000000},

        {0, 0}

};


static NCS_PROP_ENTRY rect2_props [] = {

        {NCSP_RECTANGLE_BORDERSIZE, 2},

        {NCSP_RECTANGLE_BORDERCOLOR, 0xFFFFFFF},
```

```
        {NCSP_RECTANGLE_FILLCOLOR, 0xFF0F0F0F},
        {0, 0}
};

static NCS_PROP_ENTRY rect3_props [] = {
        {NCSP_RECTANGLE_BORDERSIZE, 0},
        {NCSP_RECTANGLE_BORDERCOLOR, 0xFF0C0000},
        {NCSP_RECTANGLE_FILLCOLOR, 0xFF00FFFF},
        {0, 0}
};

static NCS_PROP_ENTRY rect4_props [] = {
        {NCSP_RECTANGLE_BORDERSIZE, 5},
        {NCSP_RECTANGLE_BORDERCOLOR, 0xFF0000FF},
        {NCSP_RECTANGLE_FILLCOLOR, 0xFF00FF00},
        {NCSP_RECTANGLE_XRADIUS, 4},
        {NCSP_RECTANGLE_YRADIUS, 4},
        {0, 0}
};
//Controls
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        ...
        {
                NCSCTRL_RECTANGLE,
```

```
            ID_RECT1,
            110, 10, 80, 40,
            WS_VISIBLE,
            WS_EX_NONE,
            "",
            rect1_props, //props,
            NULL, //rdr_info
            NULL, //handlers,
            NULL, //controls
            0,
            0 //add data
        },
        ...
        {
            NCSCTRL_RECTANGLE,
            ID_RECT2,
            110, 60, 80, 40,
            WS_VISIBLE,
            WS_EX_NONE,
            "",
            rect2_props, //props,
            NULL, //rdr_info
            NULL, //handlers,
            NULL, //controls
```

```
        0,
        0 //add data
    },
    ...
    {
        NCSCTRL_RECTANGLE,
        ID_RECT3,
        110, 110, 80, 40,
        WS_VISIBLE,
        WS_EX_NONE,
        "",
        rect3_props, //props,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    ...
    {
        NCSCTRL_RECTANGLE,
        ID_RECT4,
        110, 160, 80, 40,
        WS_VISIBLE,
```

```
                WS_EX_NONE,
                "",
                rect4_props, //props,
                NULL, //rdr_info
                NULL, //handlers,
                NULL, //controls
                0,
                0 //add data
        },
};
```

mRect 事件

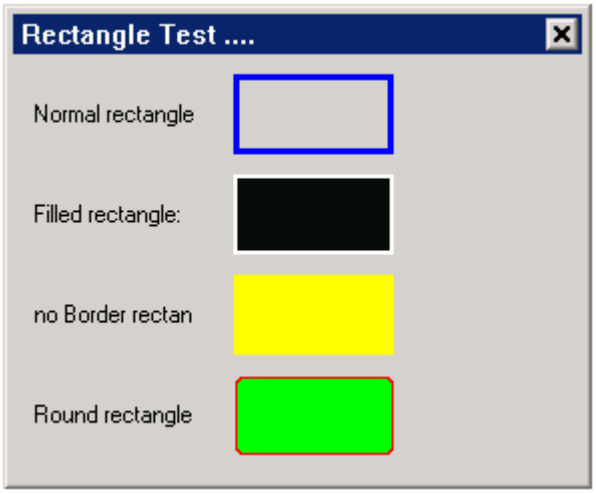
无

mRect 方法

无

mRect 实例

本实例为用户演示了如何绘制各种矩形



清单 p2c4-3 rectangle.c

```
/*
** $Id$
**
** Listing P2C4.3
**
** rectangle.c: Sample program for mGNCS Programming Guide
**      A mGNCS application for mRect.
**
** Copyright (C) 2009 Feynman Software.
**/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/control.h>
#include <mgncs/mgncs.h>

#define ID_RECT1  101
```

```
#define ID_RECT2  102

#define ID_RECT3  103

#define ID_RECT4  104


static BOOL mymain_onCreate(mWidget* self, DWORD add_data)

{

        //TODO : initialize

        return TRUE;

}


static void mymain_onClose(mWidget* self, int message)

{

        DestroyMainWindow(self->hwnd);

        PostQuitMessage(0);

}




//Propties for

static NCS_PROP_ENTRY rect1_props [] = {

        {NCSP_RECTANGLE_BORDERSIZE, 3},

        {NCSP_RECTANGLE_BORDERCOLOR, 0xFFFF0000},

        {NCSP_RECTANGLE_FILLCOLOR, 0x00000000},

        {0, 0}

};
```

```
static NCS_PROP_ENTRY rect2_props [] = {
        {NCSP_RECTANGLE_BORDERSIZE, 2},
        {NCSP_RECTANGLE_BORDERCOLOR, 0xFFFFFFFF},
        {NCSP_RECTANGLE_FILLCOLOR, 0xFF0F0F0F},
        {0, 0}
};

static NCS_PROP_ENTRY rect3_props [] = {
        {NCSP_RECTANGLE_BORDERSIZE, 0},
        {NCSP_RECTANGLE_BORDERCOLOR, 0xFF0C0000},
        {NCSP_RECTANGLE_FILLCOLOR, 0xFF00FFFF},
        {0, 0}
};

static NCS_PROP_ENTRY rect4_props [] = {
        {NCSP_RECTANGLE_BORDERSIZE, 5},
        {NCSP_RECTANGLE_BORDERCOLOR, 0xFF0000FF},
        {NCSP_RECTANGLE_FILLCOLOR, 0xFF00FF00},
        {NCSP_RECTANGLE_XRADIUS, 4},
        {NCSP_RECTANGLE_YRADIUS, 4},
        {0, 0}
};
```

```
//Controls
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_STATIC,
                0,
                10, 10, 80, 40,
                WS_VISIBLE, WS_EX_NONE,
                "Normal rectangle:",
                NULL, NULL, NULL, NULL,
                0, 0
        },
        {
                NCSCTRL_RECTANGLE,
                ID_RECT1,
                110, 10, 80, 40,
                WS_VISIBLE,
                WS_EX_NONE,
                "",
                rect1_props, //props,
                NULL, //rdr_info
                NULL, //handlers,
                NULL, //controls
                0,
                0 //add data
```

```
    },
    {
        NCSCTRL_STATIC,
        0,
        10, 60, 80, 40,
        WS_VISIBLE, WS_EX_NONE,
        "Filled rectangle:",
        NULL, NULL, NULL, NULL,
        0, 0
    },
    {
        NCSCTRL_RECTANGLE,
        ID_RECT2,
        110, 60, 80, 40,
        WS_VISIBLE,
        WS_EX_NONE,
        "",
        rect2_props, //props,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
```

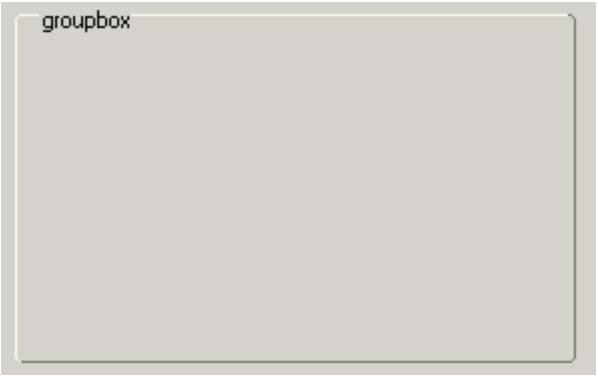
```
    {
        NCSCTRL_STATIC,
        0,
        10, 110, 80, 40,
        WS_VISIBLE, WS_EX_NONE,
        "no Border rectangle:",
        NULL, NULL, NULL, NULL,
        0, 0
    },
    {
        NCSCTRL_RECTANGLE,
        ID_RECT3,
        110, 110, 80, 40,
        WS_VISIBLE,
        WS_EX_NONE,
        "",
        rect3_props, //props,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
```

```
            NCSCTRL_STATIC,
            0,
            10, 160, 80, 40,
            WS_VISIBLE, WS_EX_NONE,
            "Round rectangle:",
            NULL, NULL, NULL, NULL,
            0, 0
        },
        {
            NCSCTRL_RECTANGLE,
            ID_RECT4,
            110, 160, 80, 40,
            WS_VISIBLE,
            WS_EX_NONE,
            "",
            rect4_props, //props,
            NULL, //rdr_info
            NULL, //handlers,
            NULL, //controls
            0,
            0 //add data
        },
};
```

```
static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate },
        {MSG_CLOSE, mymain_onClose },
        {0, NULL }
};

//define the main window template
static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 320, 320,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Rectangle Test ....",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};
```

```
int MiniGUIMain(int argc, const char* argv[])

{

        ncsInitialize();

        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect

        (&mymain_templ, HWND_DESKTOP);


        printf("NCSP_RECTANGLE_BORDERSIZE=%d\n",NCSP_RECTANGLE_BORDERSIZE);


        _c(mydlg)->doModal(mydlg, TRUE);




        MainWindowThreadCleanup(mydlg->hwnd);

        return 0;

}


#ifdef _MGRM_THREADS

#include <minigui/dti.c>

#endif
```

mGroupbox



- 功能: 分组框，本控件用于为其他控件提供可识别的分组。通常，使用分组框按功能细分窗体，在分组框中对所有选项分组能为用户提供逻辑化的可视提示。
- 继承自: mStatic
- 直接子类:
    - mButtongroup

mGroupbox 属性

无

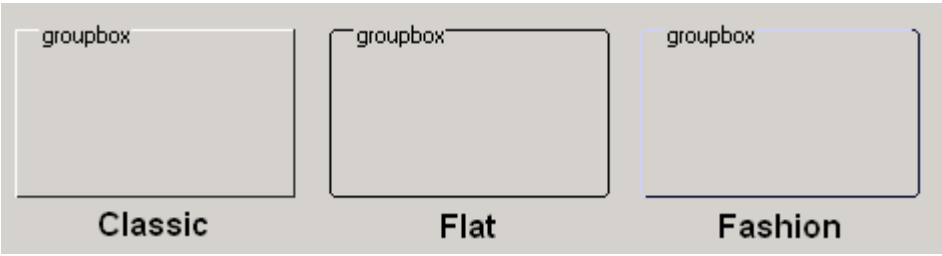
mGroupbox 事件

无

mGroupbox 方法

无

mGroupbox 渲染器

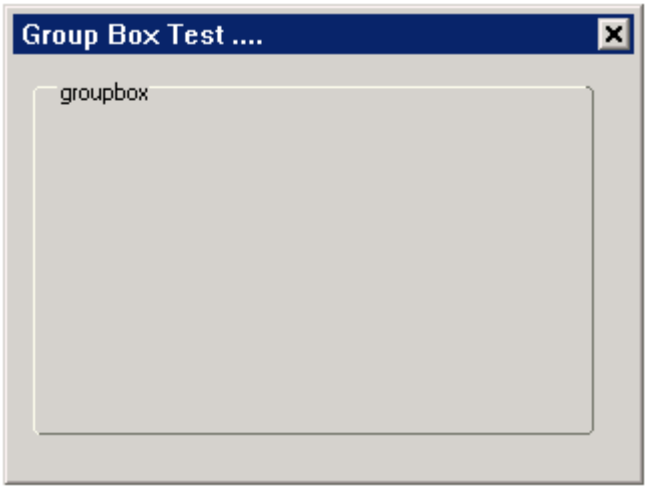


用户可以方便的为分组框设置如上图所示的 classic、flat、fashion 渲染器，具体方法见如下代码，

```
...
static NCS_RDR_INFO grp_rdr_info[] = {
        {"fashion", "fashion", NULL}
        //    {"flat", "flat", NULL}
        //    {"classic", "classic", NULL}
};

//Controls
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_GROUPBOX ,
                ID_GROUP,
                10, 10, 280, 180,
                WS_VISIBLE,
                WS_EX_NONE,
                "groupbox",
                NULL, //props,
                grp_rdr_info, //rdr_info
                NULL, //handlers,
                NULL, //controls
                0,
                0 //add data
        },
};
```

```
...
```

mGroupbox 实例



本实例为用户演示了如何生成带有渲染器的分组框

清单 p2c4-4 groupbox.c

```
/*
* ** $Id$
* **
* ** Listing P2C4.4
* **
* ** groupbox.c: Sample program for mGNCS Programming Guide
* **      A mGNCS application for mGroupbox.
* **
* ** Copyright (C) 2009 Feynman Software.
* */

#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>


#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/control.h>
#include <mgncs/mgncs.h>


#define ID_GROUP  101


static BOOL mymain_onCreate(mWidget* self, DWORD add_data)
{
        //TODO : initialize
        return TRUE;
}


static void mymain_onClose(mWidget* self, int message)
{
        DestroyMainWindow(self->hwnd);
        PostQuitMessage(0);
}


static NCS_RDR_INFO grp_rdr_info[] = {
```

```
        //    {"fashion", "fashion", NULL}
        //    {"flat", "flat", NULL}
        //    {"classic", "classic", NULL}
        {"skin", "skin", NULL}
};

//Controls
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_GROUPBOX ,
                ID_GROUP,
                //        10, 10, 280, 180,
                10, 10, 140, 90,
                WS_VISIBLE,
                WS_EX_NONE,
                "groupbox",
                NULL, //props,
                grp_rdr_info, //rdr_info
                NULL, //handlers,
                NULL, //controls
                0,
                0 //add data
        },
};
```

```
static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate },
        {MSG_CLOSE, mymain_onClose },
        {0, NULL }
};

//define the main window template
static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 320, 240,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Group Box Test ....",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};
```

```
int MiniGUIMain(int argc, const char* argv[])
{
        if(argc > 1)
        {
                grp_rdr_info[0].glb_rdr = argv[1];
                grp_rdr_info[0].ctl_rdr = argv[1];
        }
        ncsInitialize();
        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);

        _c(mydlg)->doModal(mydlg, TRUE);


        MainWindowThreadCleanup(mydlg->hwnd);
        return 0;
}

#ifdef _MGRM_THREADS
#include <minigui/dti.c>
#endif
```

mButtonGroup

- 功能: 本控件用于管理一组 RadioButton，使该组 RadioButton 能够互斥，从而实现单选功能。
- 继承自: mGroupbox
- 直接子类: 无

mButtonGroup 属性

属性名	类型	权限	属性说明
NCSP_BTNGRP_SELID	int	RW	当前选中的 radioButton ID 号
NCSP_BTNGRP_SELIDX	idx	RW	当前选中的 radioButton index 号
NCSP_BTNGRP_SELOBJ	mWidget*	RW	当前选中的 radioButton 的指针

mButtonGroup 方法

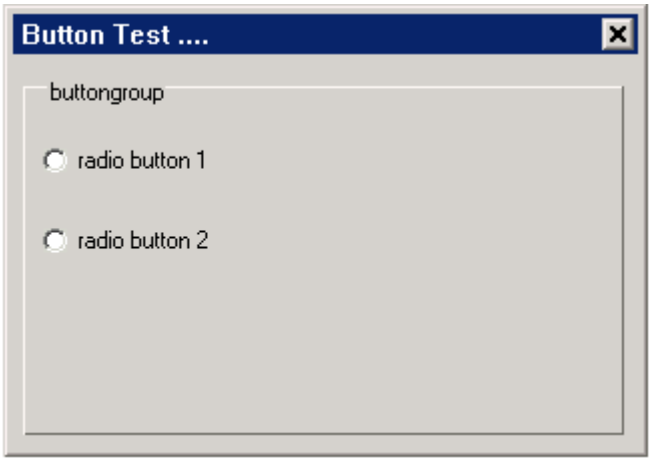
```
BOOL (*addButton)(clsName *group, mWidget *radio);
```

- 功能：添加一个 radioButton
- 返回值：BOOL

```
BOOL (*checkBtn)(clsName *group, mWidget *btn_to_check);
```

- 功能：选中一个 radioButton
- 返回值：BOOL

mButtonGroup 实例



请参见第五章按钮系列控件类中的 mRadioButton

mLEDLabel

- 功能: mLEDLabel 类是模仿 LED 数码管的静态框，用于显示 LED 字符，可用于工控等多种领域
- 继承自: mStatic
- 直接子类: 无

mLEDLabel 属性

属性名	类型	权限	属性说明
NCSP_LEDLBL_COLOR	DWORD	RW	设置 LED 字体的颜色，如红色为 0xFF0000FF
NCSP_LEDLBL_WIDTH	int	RW	设置 LED 字体的宽度
NCSP_LEDLBL_HEIGHT	int	RW	设置 LED 字体的高度

用户可以使用以下方法对属性进行设置：

```
//Propties for

static NCS_PROP_ENTRY static1_props [] = {
```

```
        { NCSP_LEDLBL_COLOR, 0xFF0000FF},
        { NCSP_LEDLBL_WIDTH, 10},
        { NCSP_LEDLBL_HEIGHT, 15},
        { NCSP_STATIC_ALIGN, NCS_ALIGN_RIGHT },
        { NCSP_STATIC_AUTOWRAP, 0 },
        {0, 0}
};

static NCS_PROP_ENTRY static2_props [] = {
        { NCSP_LEDLBL_COLOR, 0xFF0000FF},
        { NCSP_LEDLBL_WIDTH, 60},
        { NCSP_LEDLBL_HEIGHT, 90},
        { NCSP_STATIC_VALIGN, NCS_VALIGN_CENTER },
        { NCSP_STATIC_ALIGN, NCS_ALIGN_CENTER },
        {0, 0}
};

//Controls
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_LEDLABEL ,
                IDC_LEDLBL1+0,
                10, 10, 160, 50,
                WS_BORDER | WS_VISIBLE,
```

```
        WS_EX_NONE,
        "ABC 123",
        NULL, //props, 取默认属性
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_LEDLABEL ,
        IDC_LEDLBL1+2,
        10, 70, 160, 30,
        WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "2 4 6 8 0 ",
        static1_props, //props,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
```

```
                NCSCTRL_LEDLABEL ,
                IDC_LEDLBL1+5,
                180, 10, 100, 100,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
                "4",
                static2_props, //props,
                NULL, //rdr_info
                NULL, //handlers,
                NULL, //controls
                0,
                0 //add data
        },


};
```

mLEDLabel 风格

无

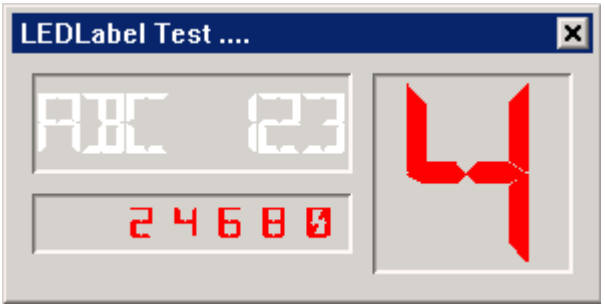
mLEDLabel 方法

无

mLEDLabel 渲染器

无

mLEDLabel 实例



本实例为用户演示了如何显示 LED 字符

```
/*
** $Id$
**
** Listing P2C4.6
**
** ledlabel.c: Sample program for mGNCS Programming Guide
**      A mGNCS application for mLEDLabel.
**
** Copyright (C) 2009 Feynman Software.
**/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#include <minigui/common.h>
#include <minigui/minigui.h>
```

```
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/control.h>
#include <mgncs/mgncs.h>

#define IDC_LEDLBL1 100
#define IDC_SATAICN 107

static BOOL mymain_onCreate(mWidget* self, DWORD add_data)
{
        //TODO : initialize
        return TRUE;
}

static void mymain_onClose(mWidget* self, int message)
{
        DestroyMainWindow(self->hwnd);
        PostQuitMessage(0);
}

//Propties for
static NCS_PROP_ENTRY static1_props [] = {
        { NCSP_LEDLBL_COLOR, 0xFF0000FF},
```

```
        { NCSP_LEDLBL_WIDTH, 10},
        { NCSP_LEDLBL_HEIGHT, 15},
        { NCSP_STATIC_ALIGN, NCS_ALIGN_RIGHT },
        { NCSP_STATIC_AUTOWRAP, 0 },
        {0, 0}
};

static NCS_PROP_ENTRY static2_props [] = {
        { NCSP_LEDLBL_COLOR, 0xFF0000FF},
        { NCSP_LEDLBL_WIDTH, 60},
        { NCSP_LEDLBL_HEIGHT, 90},
        { NCSP_STATIC_VALIGN, NCS_VALIGN_CENTER },
        { NCSP_STATIC_ALIGN, NCS_ALIGN_CENTER },
        {0, 0}
};

//Controls
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_LEDLABEL ,
                IDC_LEDLBL1+0,
                10, 10, 160, 50,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
```

```
        "ABC 123",
        NULL, //props,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_LEDLABEL ,
        IDC_LEDLBL1+2,
        10, 70, 160, 30,
        WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "2 4 6 8 0 ",
        static1_props, //props,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_LEDLABEL ,
```

```
                IDC_LEDLBL1+5,
                180, 10, 100, 100,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
                "4",
                static2_props, //props,
                NULL, //rdr_info
                NULL, //handlers,
                NULL, //controls
                0,
                0 //add data
        },

};


static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate},
        {MSG_CLOSE, mymain_onClose},
        {0, NULL}
};

//define the main window template
static NCS_MNWND_TEMPLATE mymain_templ = {
```

```
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 300, 150,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "LEDLabel Test ....",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};

int MiniGUIMain(int argc, const char* argv[])
{
        ncsInitialize();
        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);

        _c(mydlg)->doModal(mydlg, TRUE);
```

```
        MainWindowThreadCleanup(mydlg->hwnd);

        return 0;

}



#ifdef _MGRM_THREADS

#include <minigui/dti.c>

#endif
```

mSeparator

- 功能: 用于分隔各个项控件并进行区域划分的水平或垂直分隔线
- 继承自: mStatic
- 直接子类: 无

mSeparator 属性

无

mSeparator 风格

属性名	说明
NCSS_SPRTR_VERT	设置为竖向分割线，默认为横向

用户可使用以下方法对风格进行设置：

```
...

static NCS_WND_TEMPLATE _ctrl_templ[] = {

        ...,

        {

                NCSCTRL_SEPARATOR ,
```

```
            ID_GROUP,
            100, 20, 5, 200,
            WS_VISIBLE|NCSS_SPRTR_VERT,    //设置垂直风格
            WS_EX_NONE,
            "groupbox",
            NULL, //props,
            NULL, //rdr_info
            NULL, //handlers,
            NULL, //controls
            0,
            0 //add data
        },
};
```

mSeparator 事件

无

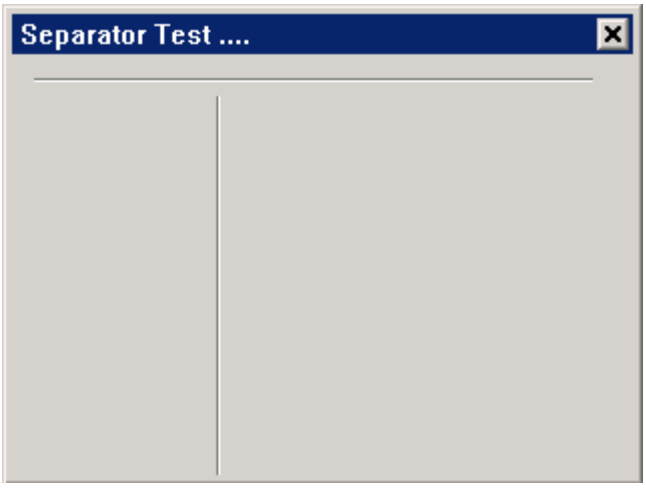
mSeparator 方法

无

mSeparator 渲染器

无

mSeparator 实例



本实例为用户演示了如何生成水平和垂直分隔线

```
/*
** $Id$
**
** Listing P2C4.7
**
** separator.c: Sample program for mGNCS Programming Guide
**      A mGNCS application for mSeparator.
**
** Copyright (C) 2009 Feynman Software.
**/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <minigui/common.h>

#include <minigui/minigui.h>

#include <minigui/gdi.h>

#include <minigui/window.h>

#include <minigui/control.h>

#include <mgncs/mgncs.h>


#define ID_GROUP  101


static BOOL mymain_onCreate(mWidget* self, DWORD add_data)

{

        //TODO : initialize

        return TRUE;

}


static void mymain_onClose(mWidget* self, int message)

{

        DestroyMainWindow(self->hwnd);

        PostQuitMessage(0);

}


//Controls

static NCS_WND_TEMPLATE _ctrl_templ[] = {

        {
```

```
        NCSCTRL_SEPARATOR ,
        ID_GROUP,
        10, 10, 280, 5,
        WS_VISIBLE,
        WS_EX_NONE,
        "groupbox",
        NULL, //props,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_SEPARATOR ,
        ID_GROUP,
        100, 20, 5, 200,
        WS_VISIBLE|NCSS_SPRTR_VERT,
        WS_EX_NONE,
        "groupbox",
        NULL, //props,
        NULL, //rdr_info
        NULL, //handlers,
        NULL, //controls
```

```
                0,
                0 //add data
        },
};



static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate },
        {MSG_CLOSE, mymain_onClose },
        {0, NULL }
};

//define the main window template
static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 320, 240,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Separator Test ....",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
```

```
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};

int MiniGUIMain(int argc, const char* argv[])
{
        ncsInitialize();
        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);

        _c(mydlg)->doModal(mydlg, TRUE);

        MainWindowThreadCleanup(mydlg->hwnd);
        return 0;
}

#ifdef _MGRM_THREADS
#include <minigui/dti.c>
#endif
```

第二部分第五章 按钮系列控件类

按钮系列控件简介

按钮作为用户界面中人机交互的不可或缺的控件，所有人机交互界面中都包括了按钮。在 mGNCS 的设计中，充分考虑了不同使用环境下的按钮使用情况，提供了不同的属性、接口等，方便用户组件化开发。mGNCS 中的按钮重构了 MiniGUI 3.0 中内置的按钮控件，并进行了较大的调整，增加了两个按钮类型，大大增强了按钮控件。mGNCS 中的按钮包括普通按钮、复选框、单选钮、菜单按钮、颜色选择按钮这 5 种类型。用户可以通过键盘或者鼠标来选择或者切换按钮的状态。用户的输入将使按钮产生通知消息，应用程序也可以向按钮发送消息以改变按钮的状态。 每种按钮类型对应一个类，里面包含了风格、属性、事件、方法、渲染器。mGNCS 中提供的控件，比通常 PC 上使用的控件功能更强大，提供了更多可配置信息，增加了渲染器编辑。

在本文档中，将对按钮类及其里面的这些属性等进行详细介绍，方便用户进一步了解按钮的细节，从而可以更灵活的使用按钮。

- 按钮相关的类层次关系
  - mWidget
    - mButton
      - mCheckButton
      - mRadioButton
      - mMenuButton (使用了 mPopMenuMgr)
      - mColorButton

- 控件创建方法
  - 自动创建：通过 miniStudio 中的界面设计器，拖拽对应的按钮控件，miniStudio 将自动创建控件，并提供可视化的控件配置，同时自动生成创建代码。
  - 手动生成：按照 mGNCS 控件创建过程，通过编程，传入对应控件窗口类 ID，生成控件，手动编程设置控件属性、事件处理。

按钮类渲染器

mGNCS 提供了 4 种类型的渲染器，定义了渲染器属性设置集合，4 种类型渲染器对应了 4 种渲染器属性设置集合。对于按钮类控件，4 种渲染器定义了按钮控件中那些渲染器属性可以设置，可以设置成什么效果，mGNCS 提供了默认的渲染器属性，当然，用户可以更改这些属性，从而可以设置不同的效果，也就是用户可以基于这 4 种类型渲染器，变更不同渲染器属性，创建无数渲染器实例。

mButton

控件窗口类

NCSCTRL_BUTTON

控件英文名

Push Button

简介

是按下方式的按钮。按钮上，简单的可以用文字表示，也可以用图片标示，还可以用图片和文字混合显示。

示意图

理解按钮的编程，首先要理解按钮的状态，在 mGNCS 中，按钮的状态分为四种，参见附录定义及注释：ButtonState

mButton 风格

按钮的风格包括两大类：

- 一个是定义按钮 CHECK 状态，也就是按下状态，通过属性中 Checkable/Autocheck/ThreeDCheck 进行设置；
- 另外一个是设置按钮面板是显示文本或者图片，或者图文并存，主要通过属性中的 Label Type 中的选项设置，默认为文本 Text，而图片和文本的布局则通过其他属性进行设置。

mButton 是所有 button 的基础类，它默认实现了 PushButton。其他的各种类型的 button 都是由它派生的。

继承自 mWidget 的风格

风格 ID	miniStudio 属性名	说明
NCSS_BUTTON_CHECKABLE	Checkable	设置按钮能否进行 CHECKED 状态转换，如果 FALSE，不进行 CHECK 状态转换，且 autocheck 和 ThreeDCheck 无效
NCSS_BUTTON_AUTOCHECK	Autocheck	在可进行 CHECKED 状态转换下，设置是否可以单击进行自动切换 CHECK 状态
NCSS_BUTTON_3DCHECK	ThreeDCheck	设定按钮 CHECK 的状态是三态(unchecked-halfchecked-checked)还是两态(unchecked-checked)切换
NCSS_BUTTON_IMAGE	LabelType ->Image	图片按钮风格， 按钮默认为 LabelType 中的 Label 文字模式

NCSS_BUTTON_IMAGELABEL	LabelType ->ImageLabel	水平排列图片文本按钮
NCSS_BUTTON_VERTIMAGELABEL	LabelType ->VertImageLabel	垂直排列图片文本按钮

mButton 属性

继承自 mWidget 的属性

属性 ID 名	miniStudio 属性名	类型	权限	说明
NCSP_BUTTON_ALIGN	Align	对齐枚举值	RW	按钮上的内容（图片或者文字）在水平方向按照设置的方式与按钮进行对齐
NCSP_BUTTON_VALIGN	VAlign	对齐枚举值	RW	按钮上的内容（图片或者文字）在垂直方向按照设置的方式与按钮进行对齐，对齐值为 top - 0, bottom - 1, center - 2
NCSP_BUTTON_WORDWRAP	Wordwrap	int	RW	按钮上文字自动换行属性，当文字内容单行长度大于控件宽度时，可以选择自动换行。TRUE：自动换行 FALSE：为单行，超过的部分不显示
NCSP_BUTTON_IMAGE	Image	PBITMAP	RW	图片，对应 button 属性 pbmp，即图片指针，必须在 NCSS_BUTTON_IMAGE 或 NCSS_BUTTON_IMAGELABEL 或 NCSS_BUTTON_VERTIMAGELABEL 被设置时有效
NCSP_BUTTON_CHECKSTATE	无	mButtonCheckState	RW	设置或获取 check 状态. 当 NCSS_BUTTON_CHECKABLE 设置时有效. 当 NCSS_BUTTON_3DCHECK 设置时 NCS_BUTTON_HALFCHECKED 被认为是有效值

NCSP_BUTTON_IMAGE_SIZE_PERCENT	ImageSizePercent	int	RW	在图文并存的风格下,设置图片在控件上的所占百分比(取值 15~85，表示从 15%到 85%)，剩下的为文本所占用
NCSP_BUTTON_GROUPID	GroupID	int	RW	在有 NCSS_BUTTON_CHECKABLE 风格时,可以设置一个 mButtonGroup 对象的 ID 给窗口. 它将自动获取对应的 mButtonGroup 对象，并设置
NCSP_BUTTON_GROUP	无	mButtonGroup *	RW	在有 NCSS_BUTTON_CHECKABLE 风格时，设置一个 mButtonGroup 对象指针,所有被加入到该 group 中的 button 都具有 Radio 风格的自动切换能力

mButton 事件

继承自 mWidget 的事件

事件通知码	说明	参数
NCSN_BUTTON_PUSHED	按键按下	
NCSN_BUTTON_STATE_CHANGED	状态变化后	新状态

注意,该控件继承了父类的 NCSN_WIDGET_CLICKED 事件。

mButton 渲染器

渲染器使用方法见外观渲染器

mButton Classic 渲染器

Classic 渲染器下的基本风格如下图：

mButton Classic 渲染器的绘制如下：非客户区的绘制请查阅 mWidget 的渲染器

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域
NCS_FGC_3DBODY	文本前景色	FgColor3DBody	DWORD(ARGB)		
NCS_BGC_3DBODY	背景颜色	BgColor3DBody	DWORD(ARGB)		
NCS_FGC_DISABLED_ITEM	窗口无效时文本前景色	TextDisableColor	DWORD(ARGB)		
NCS_BGC_DISABLED_ITEM	窗口无效时文本背景色	BgColorDisable	DWORD(ARGB)		

- 高亮效果：通过对背景色（NCS_BGC_3DBODY ）的亮度提高，作为高亮色
- check 效果：通过绘制内陷边框实现
- 3 态 Check 效果: 通过在内陷边框内绘制高亮色来实现半选中

示意图:

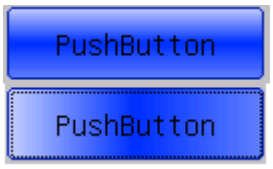
mButton Skin 渲染器

参阅 附录 B : Skin 渲染器使用的图片资源规范

mButton Fashion 渲染器

非客户区绘制请参阅 mWidget 的 Fashion 渲染器绘制

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域
NCS_FGC_3DBODY	文本前景色	FgColor3DBody	DWORD(ARGB)	同 Classic 渲染器	

NCS_FGC_DISABLED_ITEM	窗口无效时文本前景色	TextDisableColor	DWORD(ARGB)	同 Classic 渲染器	
NCS_BGC_3DBODY	背景颜色	BgColor3DBody	DWORD(ARGB)		
NCS_BGC_DISABLED_ITEM	窗口无效时文本背景色	BgColorDisable	DWORD(ARGB)		
NCS_MODE_BGC	渐变填充方式	GradientMode	int		Gradient Mode
NCS_METRICS_3DBODY_ROUNDX	窗口矩形圆角 X 半径	RoundX	int		0 ~ 窗口宽度的 1/2
NCS_METRICS_3DBODY_ROUNDY	窗口矩形圆角 Y 半径	RoundY	int		0 ~窗口高度的 1/2

- 渐变色：渐变色由底色（NCS_BGC_3DBODY 或 NCS_BGC_DISABLED_ITEM）给出不同的亮度因子，计算出两个目标色，分别从中心（底色）向两边或者上下（计算得出的加亮色）渐变
- 高亮色：由底色经过加亮后，渐变色再在此基础上计算获取
- check 状态的表示方法：由底色经过变暗处理后，再计算渐变色
- 3 态 check 的表示方法：除了在 check 状态方法同上，在半选状态，用窗口底色（NCS_BGC_WINDOW）来作为整个底色来绘制

示意图：

mButton Flat 渲染器

非客户区绘制请参阅 mWidget 的 Flat 渲染器绘制

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域

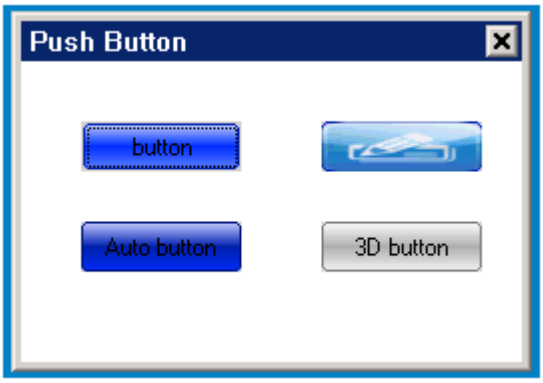
NCS_FGC_3DBODY	文本前景色	FgColor3DBody	DWORD(ARGB)	同 Classic 渲染器	
NCS_FGC_DISABLED_ITEM	窗口无效时文本前景色	TextDisableColor	DWORD(ARGB)	同 Classic 渲染器	
NCS_BGC_3DBODY	背景颜色	BgColor3DBody	DWORD(ARGB)	同 Classic 渲染器	
NCS_METRICS_3DBODY_ROUNDX	窗口矩形圆角 X 半径	RoundX	int		0 ~ 窗口宽度的 1/2
NCS_METRICS_3DBODY_ROUNDY	窗口矩形圆角 Y 半径	RoundY	int		0 ~窗口高度的 1/2

- 高亮：在控件四周绘制小的提示符号实现
- check 状态：在右-底编辑分别绘制线条，组成倒'L'型实现
- 3 态的 check 表示方法：在半选状态时，在左-上分别绘制线条，形成卧倒的'L'型

示意图:

mButton 编程示例

- 运行截图:



- 本程序使用了 Fashion 渲染器创建了 4 个不同风格的按钮，分别是普通按钮，图片按钮，Autocheck 风格按钮，3 态按钮。
- 普通按钮完整示例代码：button.c

清单 p2c5-1 button.c

- 定义普通按钮

```
{
        NCSCTRL_BUTTON,
        ID_BTN,
        30, 30, 80, 25,
        WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "button",
        NULL,          //props,
        btn_rdr_info, //rdr_info
        NULL,          //handlers,
        NULL,          //controls
        0,
        0              //add data
},
```

- 定义 Image 按钮，一般情况下都是在主窗口的 onCreate 事件中处理

```
{
        NCSCTRL_BUTTON,
        ID_BTN1,
        150, 30, 80, 25,
        WS_VISIBLE | NCSS_BUTTON_IMAGE,
        WS_EX_NONE,
```

```
        "Image",
        NULL,           //props,
        btn_rdr_info, //rdr_info
        NULL,           //handlers,
        NULL,           //controls
        0,
        0               //add data
},
```

- 加载并设置 Image

```
if(LoadBitmapFromFile(HDC_SCREEN, &bmp, "icon_button.png")!=0)
{
        printf("cannot load image file \"icon_button.png\"\n");
}

mButton *mb1 = (mButton*)ncsGetChildObj(self->hwnd, ID_BTN1);

if(mb1)
_c(mb1)->setProperty(mb1, NCSP_BUTTON_IMAGE, (DWORD)&bmp);
```

- 定义 Autocheck 风格的按钮

```
{
        NCSCTRL_BUTTON,
        ID_BTN1,
```

```
        30, 80, 80, 25,
        WS_VISIBLE  | NCSS_BUTTON_AUTOCHECK | NCSS_BUTTON_CHECKABLE,
        WS_EX_NONE,
        "Auto button",
        NULL,          //props,
        btn_rdr_info, //rdr_info
        NULL,          //handlers,
        NULL,          //controls
        0,
        0              //add data
},
```

- 定义 3 态 check 风格的按钮

```
{
        NCSCTRL_BUTTON,
        ID_BTN1,
        150, 80, 80, 25,
        WS_VISIBLE | NCSS_BUTTON_AUTOCHECK | NCSS_BUTTON_CHECKABLE \
        | NCSS_BUTTON_3DCHECK,
        WS_EX_NONE,
        "3D button",
        NULL,          //props,
        btn_rdr_info, //rdr_info
        NULL,          //handlers,
```

```
        NULL,           //controls

        0,

        0               //add data

},
```

mCheckButton

控件窗口类

NCSCTRL_CHECKBUTTON

控件英文名

CheckButton

简介

复选框按钮， 主要用于多选场合，常用的样式是一个方框，用户在方框中选中或不选中。

示意图

mCheckButton 属性

继承自 mButton 的属性。

mCheckButton 事件

继承自 mButton 的事件。

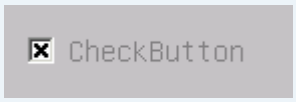
mCheckButton 渲染器

- 渲染器使用方法见外观渲染器

mCheckButton Classic 渲染器

- mCheckButton Classic 文本区域渲染器绘制如下: 非客户区的绘制请查阅 mWidget 的渲染器

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域

NCS_FGC_3DBODY	文本前景色	FgColor3DBody	DWORD(ARGB)	
NCS_FGC_DISABLED_ITEM	窗口无效时文本前景色	TextDisableColor	DWORD(ARGB)	

- 选择框 Check Box 的 Classic 渲染器是通过加载图片资源 来填充绘制的。（图片资源默认为 minigui-res, 默认安装在/usr/local/share/minigui/res/bmp/classic_check_button.bmp）
  - 图片的规格：图片由自左到右的 8 部分组成，每一部分都是一个正方形(13x13)，分别对应 checkbutton 的 8 种状态：
    - 0~3：未选中时的普通、高亮、按下、禁用状态
    - 4~7：选中时的普通、高亮、按下、禁用状态
  - 若图片大于绘制区域，会缩小图片进行绘制，小于或等于时采用图片实际大小绘制
  - 示例

mCheckButton Skin 渲染器

参阅 附录 B : Skin 渲染器使用的图片资源规范

mCheckButton Fashion 渲染器

- mCheckButton Fashion 文本区域渲染器绘制如下: 非客户区的绘制请查阅 mWidget 的渲染器

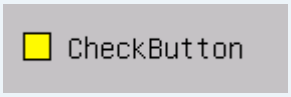
属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域
NCS_FGC_3DBODY	文本前景色	FgColor3DBody	DWORD(ARGB)		
NCS_FGC_DISABLED_ITEM	窗口无效时文本前景色	TextDisableColor	DWORD(ARGB)		

- 选择框 Check Box 的 Fashion 渲染器是通过加载图片资源 来填充绘制的。（注: 图片资源默认为 minigui-res, 默认安装在/usr/local/share/minigui/res/bmp/fashion_check_btn.bmp）
  - 图片的规格：图片由自上到下的 8 部分组成，每一部分都是一个正方形(13x13)，分别对应 checkbutton 的 8 种状态：
    - 0~3：未选中时的普通、高亮、按下、禁用状态

- 4~7：选中时的普通、高亮、按下、禁用状态
- 若图片大于绘制区域，会缩小图片进行绘制，小于或等于时采用图片实际大小绘制
- 示例

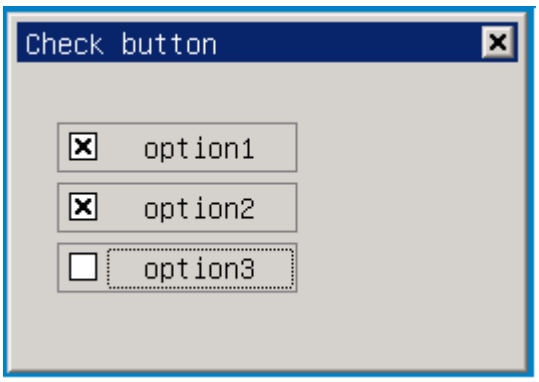
mCheckButton Flat 渲染器

- mCheckButton Flat 渲染器的文本区域和选择框绘制如下：非客户区的绘制请查阅 mWidget 的渲染器

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域
NCS_FGC_3DBODY	文本前景色	FgColor3DBody	DWORD(ARGB)		
NCS_BGC_3DBODY	背景色	BgColor3DBody	DWORD(ARGB)		

mCheckbutton 编程示例

- 运行截图:



- 复选框示例代码：checkbutton.c

```
#define ID_BTN  101
#define ID_BTN1 102
#define ID_BTN2 103
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_CHECKBUTTON,
                ID_BTN,
                20, 30, 120, 25,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
                "option1",
                NULL,          //props,
                btn_rdr_info, //rdr_info
                NULL,          //handlers,
                NULL,          //controls
                0,
                0              //add data
        },
        {
                NCSCTRL_CHECKBUTTON,
                ID_BTN1,
                20, 60, 120, 25,
                WS_BORDER | WS_VISIBLE | NCSS_BUTTON_AUTOCHECK,
                WS_EX_NONE,
```

```
            "option2",
            NULL,           //props,
            btn_rdr_info, //rdr_info
            NULL,           //handlers,
            NULL,           //controls
            0,
            0               //add data
        },
        {
            NCSCTRL_CHECKBUTTON,
            ID_BTN2,
            20, 90, 120, 25,
            WS_BORDER | WS_VISIBLE |NCSS_BUTTON_AUTOCHECK | NCSS_BUTTON_3DCHECK,
            WS_EX_NONE,
            "option3",
            NULL,           //props,
            btn_rdr_info, //rdr_info
            NULL,           //handlers,
            NULL,           //controls
            0,
            0               //add data
        }
};
```

```
static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 260, 180,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Check button",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};
```

mRadioButton

控件窗口类

NCSCTRL_RADIOBUTTON

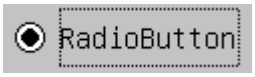
控件英文名

RadioButton

简介

单选按钮，和 mButtonGroup 配合使用。mButtonGroup 将多个单选按钮归纳为一组，组内的按钮只有一个处于选中状态，一个按钮被选中后，原选中按钮就会取消选中。 当单选按钮独立使用是，其行为和一个 mCheckbutton 类似。

示意图



mRadiobutton 属性

继承自 mButton 的属性。

mRadiobutton 事件

继承自 mButton 的事件。

mRadiobutton 渲染器

- 渲染器使用方法见外观渲染器

mRadiobutton Classic 渲染器

- mRadiobutton Classic 非客户区的绘制请查阅 mWidget 的渲染器

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域
NCS_FGC_3DBODY	文本前景色	FgColor3DBody	DWORD(ARGB)		
NCS_FGC_DISABLED_ITEM	窗口无效时文本前景色	TextDisableColor	DWORD(ARGB)		

- Radiobutton 选择框 Check Box 的 Classic 渲染器是通过加载图片资源 来填充绘制的。（图片资源默认为 minigui-res，默认安装在/usr/local/share/minigui/res/bmp/classic_radio_button.bmp）
  - 图片的规格：图片由自左到右的 8 部分组成，每一部分都是一个正方形(13x13)，分别对应 checkbutton 的 8 种状态：
    - 0~3：未选中时的普通、高亮、按下、禁用状态
    - 4~7：选中时的普通、高亮、按下、禁用状态
  - 若图片大于绘制区域，会缩小图片进行绘制，小于或等于时采用图片实际大小绘制
  - 示例

mRadiobutton Skin 渲染器

参阅 附录 B : Skin 渲染器使用的图片资源规范

mRadiobutton Fashion 渲染器

- mRadiobutton Fashion 非客户区的绘制请查阅 mWidget 的渲染器

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域
NCS_FGC_3DBODY	文本前景色	FgColor3DBody	DWORD(ARGB)		
NCS_FGC_DISABLED_ITEM	窗口无效时文本前景色	TextDisableColor	DWORD(ARGB)		

- Radiobutton 选择框 Check Box 的 Fashion 渲染器是通过加载图片资源 来填充绘制的。（图片资源默认为 minigui-res，默认安装在/usr/local/share/minigui/res/bmp/fashion_radio_btn.bmp）
  - 图片的规格：图片由自上到下的 8 部分组成，每一部分都是一个正方形(13x13)，分别对应 checkbutton 的 8 种状态：
    - 0~3：未选中时的普通、高亮、按下、禁用状态
    - 4~7：选中时的普通、高亮、按下、禁用状态
  - 若图片大于绘制区域，会缩小图片进行绘制，小于或等于时采用图片实际大小绘制
  - 示例

mRadiobutton Flat 渲染器

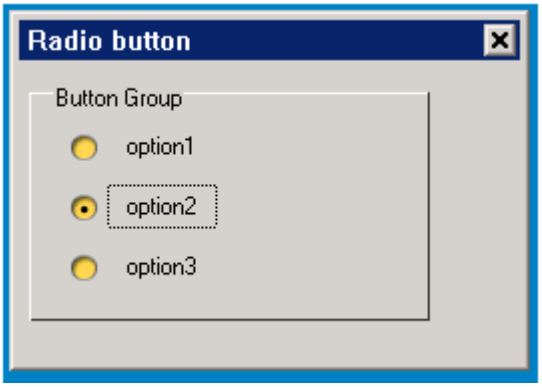
- mRadiobutton Flat 非客户区的绘制请查阅 mWidget 的渲染器

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域
NCS_FGC_3DBODY	文本前景色	FgColor3DBody	DWORD(ARGB)		

NCS_BGC_3DBODY	背景色	BgColor3DBody	DWORD(ARGB)	RadioButton

mRadiobutton 编程示例

- 运行截图:



- 单选钮示例代码：radiogroup.c

```
static NCS_PROP_ENTRY radioGroup_props [] = {
        {NCSP_BUTTON_GROUPID, 200},
        {0, 0}
};

static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_BUTTONGROUP ,
                ID_GROUP,
                5, 10, 200, 120,
                WS_VISIBLE|NCSS_NOTIFY,
                WS_EX_NONE,
```

```
        "Button Group",
        NULL,           //props,
        btn_rdr_info,   //rdr_info
        NULL,           //handlers,
        NULL,           //controls
        0,
        0               //add data
    },
    {
        NCSCTRL_RADIOBUTTON,
        ID_BTN1,
        20, 30, 80, 25,
        WS_VISIBLE,
        WS_EX_NONE,
        "option1",
        radioGroup_props, //props,
        btn_rdr_info,     //rdr_info
        NULL,             //handlers,
        NULL,             //controls
        0,
        0                 //add data
    },
    {
        NCSCTRL_RADIOBUTTON,
```

```
            ID_BTN2,
            20, 60, 80, 25,
            WS_VISIBLE,
            WS_EX_NONE,
            "option2",
            radioGroup_props, //props,
            btn_rdr_info,     //rdr_info
            NULL,             //handlers,
            NULL,             //controls
            0,
            0                 //add data
        },
        {
            NCSCTRL_RADIOBUTTON,
            ID_BTN2,
            20, 90, 80, 25,
            WS_VISIBLE,
            WS_EX_NONE,
            "option3",
            radioGroup_props, //props,
            btn_rdr_info,     //rdr_info
            NULL,             //handlers,
            NULL,             //controls
            0,
```

```
                0                   //add data
        },
};


static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 260, 180,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Radio button",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};
```

mMenuButton

控件窗口类

NCSCTRL_MENUBUTTON

控件英文名

MenuButton

简介

菜单按钮。点击该按钮，将弹出菜单，通过键盘或者鼠标，选择菜单项，选中的菜单项将在按钮上

显示。

示意图

mMenuButton 继承自 mPopMenuMgr 类。

mMenuButton 属性

继承自 mButton 的属性。

属性名	类型	权限	说明
NCSP_MNUBTN_POPMENU	mPopMenuMgr*	RW	设置 PopMenu 对象指针
NCSP_MNUBTN_CURITEM	int	RW	获取或设置当前选中 Menu Item 的 id

mMenuButton 事件

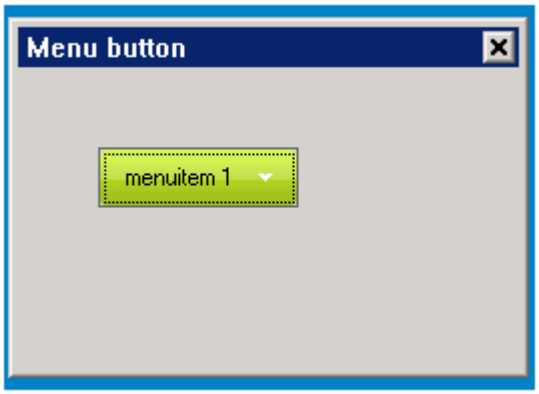
继承自 mButton 的事件。

事件通知码	说明	参数
NCSN_MNUBTN_ITEMCHANGED	当 MenuItem 改变时	新 Item 的 id 值

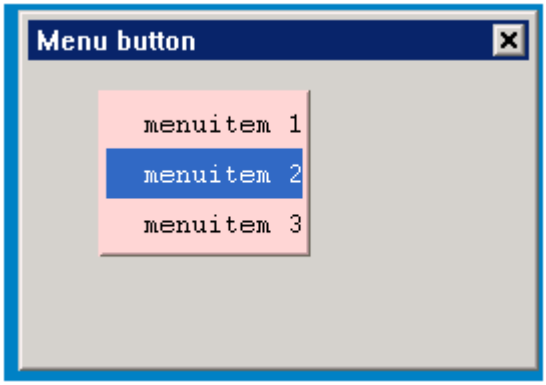
mMenuButton 编程示例

mMenuButton 的主要是通过创建 mPopMenuMgr 对象来实现的

- ：



按下后



- 菜单按钮示例代码：menubutton.c

```
/*
** $Id$
**
** Listing P2C5.4
**
** menubutton.c: Sample program for mGNCS Programming Guide
**      The first mGNCS application.
**
** Copyright (C) 2009 Feynman Software.
*/
```

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// START_OF_INCS

#include <minigui/common.h>

#include <minigui/minigui.h>

#include <minigui/gdi.h>

#include <minigui/window.h>


#include <mgncs/mgncs.h>

// END_OF_INCS


#define ID_BTN  101


// START_OF_HANDLERS

static void menubutton_onitemchanged(mMenuButton *self, int id, int nc,

DWORD add_data)

{

        char szText[100];

        sprintf(szText,"id=%d",id);

        MessageBox(self->hwnd, szText,"Menu ID",0);

}
```

```
static NCS_EVENT_HANDLER menubutton_handlers[] = {
        NCS_MAP_NOTIFY(NCSN_MNUBTN_ITEMCHANGED, menubutton_onitemchanged),
        {0, NULL}
};

static BOOL mymain_onCreate(mWidget* self, DWORD add_data)
{
        mPopMenuMgr * popmenu = NEW(mPopMenuMgr);

        _c(popmenu)->addItem(popmenu,0, "menuitem 1", NULL, 200, 0, NULL, 0);
        _c(popmenu)->addItem(popmenu,0, "menuitem 2", NULL, 201, 0, NULL, 0);
        _c(popmenu)->addItem(popmenu,0, "menuitem 3", NULL, 202, 0, NULL, 0);

        mButton *mb1 = (mButton*)ncsGetChildObj(self->hwnd, ID_BTN);
        if(mb1)
        {
                _c(mb1)->setProperty(mb1, NCSP_MNUBTN_POPMENU, (DWORD)popmenu);
        }

        return TRUE;
}

static void mymain_onClose(mWidget* self, int message)
{
```

```
        DestroyMainWindow(self->hwnd);
        PostQuitMessage(0);
}

static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate},
        {MSG_CLOSE, mymain_onClose},
        {0, NULL}
};
// END_OF_HANDLERS

// START_OF_RDRINFO
NCS_RDR_ELEMENT btn_rdr_elements[] =
{
        { NCS_MODE_USEFLAT, 1},
        { -1, 0 }
};
static NCS_RDR_INFO btn_rdr_info[] =
{
        {"skin", "skin", NULL},
};
// END_OF_RDRINFO

// START_OF_TEMPLATE
```

```
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_MENUBUTTON,
                ID_BTN,
                40, 40, 100, 30,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
                "menu button",
                NULL,                   //props,
                btn_rdr_info,           //rdr_info
                menubutton_handlers, //handlers,
                NULL,                   //controls
                0,
                0                       //add data
        },
};


static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 260, 180,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Menu button",
```

```
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};
// END_OF_TEMPLATE

int MiniGUIMain(int argc, const char* argv[])
{
        ncsInitialize();

        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);

        _c(mydlg)->doModal(mydlg, TRUE);

        MainWindowThreadCleanup(mydlg->hwnd);

        ncsUninitialize ();

        return 0;
```

```
}
```

mColorButton

控件窗口类

NCSCTRL_COLORBUTTON

控件英文名

ColorButton

简介

颜色选择按钮，该按钮用于选择颜色。点击该按钮时，弹出颜色选择框，用户选择颜色，确定后，选中的颜色在按钮面板上显示效果。

示意图

mColorButton 使用 mgutils 的 ColorSelectDialog 来选择颜色， 被选中的颜色显示在中间的框内。

mColorButton 属性

继承自 mButton 的属性。

属性名	类型	权限	说明
NCSP_CLRBTN_CURCOLOR	DWORD(ARGB)	RW	设置或者获取 Color 值

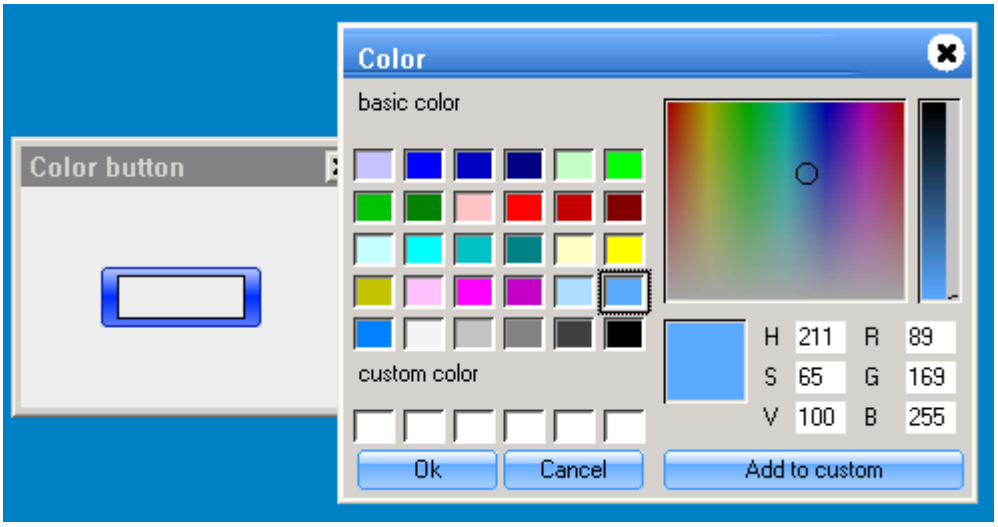
mColorButton 事件

继承自 mButton 的事件。

事件通知码	说明	参数
NCSN_CLRBTN_COLORCHANGED	当被用户选择后更新	DWORD(ARGB)

mColorButton 编程示例

- 运行截图



- 颜色按钮示例代码 ： colorbutton.c

- 定义处理函数

```
static BOOL mymain_oncolorchanged(mMainWnd* self, mColorButton *sender,
int id, DWORD param)
{
        SetWindowBkColor(self->hwnd, DWORD2PIXEL(HDC_SCREEN, param));
        InvalidateRect(self->hwnd, NULL, TRUE);

        return FALSE;
}

static BOOL mymain_onCreate(mWidget* self, DWORD add_data)
{
```

```
        mColorButton *btn = (mColorButton*)_c(self)->getChild(self, ID_BTN);
        if(btn)
        {
                _c(btn)->setProperty(btn, NCSP_CLRBTN_CURCOLOR, PIXEL2DWORD(HDC_SCREEN,
                GetWindowBkColor(self->hwnd)));
                ncsAddEventListener((mObject*)btn, (mObject*)self,
                (NCS_CB_ONPIECEEVENT)mymain_oncolorchanged,
                NCSN_CLRBTN_COLORCHANGED);
        }

        return TRUE;
}

static void mymain_onClose(mWidget* self, int message)
{
        DestroyMainWindow(self->hwnd);
        PostQuitMessage(0);
}

static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate},
        {MSG_CLOSE, mymain_onClose},
        {0, NULL}
};
```

- 定义窗口模板

```
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_COLORBUTTON,
                ID_BTN,
                40, 40, 80, 30,
                WS_VISIBLE|NCSS_NOTIFY,
                WS_EX_NONE,
                "button",
                NULL,          //props,
                btn_rdr_info, //rdr_info
                NULL,          //handlers,
                NULL,          //controls
                0,
                0              //add data
        }
};

static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_MAINWND,
        1,
        0, 0, 180, 140,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
```

```
        WS_EX_NONE,
        "Color button",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};
```

Appendix:A

GrandientMode

必须为以下值之一:

- MP_LINEAR_GRADIENT_MODE_HORIZONTAL: 水平渐变， 从中心向左右渐变
- MP_LINEAR_GRADIENT_MODE_VERTICAL: 垂直渐变，从中心向上下渐变

这两个值在 mGPlus 中定义（<mgplus/mgplus.h>）。虽然 mGPlus 也定义了其他渐变模式，但是在 NCS 中不被支持

ButtonState

以下四种状态之一:

- Normal: 正常状态。
- Hilight: 鼠标移动到按钮上，未点击时的状态，也称为高亮状态。
- Checked: 按钮处于被点击状态，也称为 checked 状态，这个状态下，又细分不同的状态，参见下面 ButtonCheckState 状态的定义。
- Disabled: 按钮处于无效状态。

ButtonCheckState

```
/* 按钮 CHECK 状态细分定义 */

enum mButtonCheckState{

        /* 表示没有按下情况 */

        NCS_BUTTON_UNCHECKED = 0,


        /* 在 3 态情况下，这个状态有效，可以理解为中间的过渡状态 */

        NCS_BUTTON_HALFCHECKED,


        /* 表示按下状态 */

        NCS_BUTTON_CHECKED

};
```

第二部分第六章 面板及其派生类

面板类控件简介

该类控件是容纳其它各类控件的容器类控件，一般为组合控件类和主窗口类。

面板及其派生类的类继承关系如下：

- mWidget
  - mPanel
    - mComboBox
    - mMainWnd
      - mDialogBox

mPanel

控件名称

NCSCTRL_PANEL

英文名

Panel

简要介绍

其他控件的容器，多用于对控件进行分组操作。通过将控件分组到 panel 控件中，可以方便地显示或隐藏一组控件。

示意图

Panel 必须存在于 MainWnd, DialogBox, 另一 Panel 控件或其它控件中。除可包含其他控件外，还可以包含文本和图片等内容。

mPanel 风格

继承自 mWidget 的风格

mPanel 属性

继承自 mWidget 的属性

mPanel 事件

继承自 mWidget 的事件

mPanel 方法

继承自 mWidget 的方法

该类没有新增方法。

mPanel 渲染器

继承自 mWidget 渲染器

mPanel 无新增加渲染器方法。

mPanel 实例

本实例为用户演示了如何使用 panel 对多个控件进行分组。

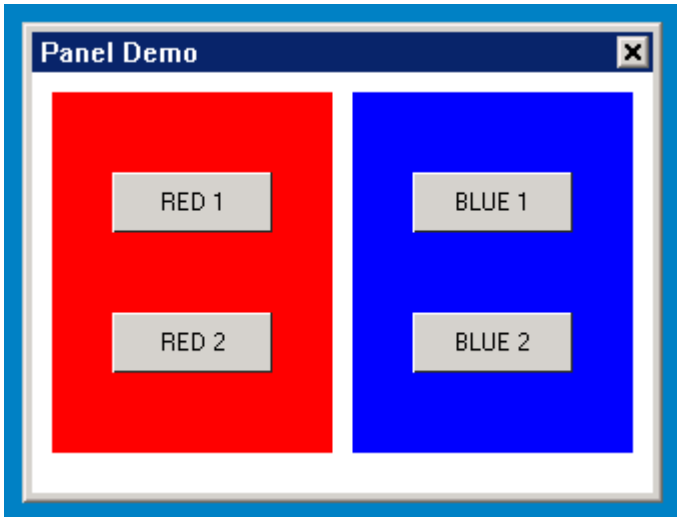


图 p2c6-1 panel 程序的输出

清单 p2c6-1 panel.c

```
/*
** $Id: panel.c 594 2009-10-10 05:59:06Z xwyan $
**
** Listing P2C6.1
**
** panel.c: Sample program for mGNCS Programming Guide
**      The demo application for Panel.
**
** Copyright (C) 2009 Feynman Software.
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// START_OF_INCS
```

```
#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>

#include <mgncs/mgncs.h>
// END_OF_INCS

#define     ID_PANEL_GROUP1     101
#define     ID_PANEL_GROUP2     102

#define     ID_RED1             201
#define     ID_RED2             202
#define     ID_BLUE1            301
#define     ID_BLUE2            302

// START_OF_REDGROUP
static NCS_WND_TEMPLATE _ctrl_group1[] = {
        {
                NCSCTRL_BUTTON,
                ID_RED1,
                30, 40, 80, 30,
                WS_VISIBLE,
                WS_EX_NONE,
```

```
                "RED 1",
                NULL,
                NULL,
                NULL,
                NULL,
                0,
        },
        {
                NCSCTRL_BUTTON,
                ID_RED2,
                30, 110, 80, 30,
                WS_VISIBLE,
                WS_EX_NONE,
                "RED 2",
                NULL,
                NULL,
                NULL,
                NULL,
                0,
        },
};
// END_OF_REDGROUP

// START_OF_BLUEGROUP
```

```
static NCS_WND_TEMPLATE _ctrl_group2[] = {
        {
                NCSCTRL_BUTTON,
                ID_BLUE1,
                30, 40, 80, 30,
                WS_VISIBLE,
                WS_EX_NONE,
                "BLUE 1",
                NULL,
                NULL,
                NULL,
                NULL,
                0,
        },
        {
                NCSCTRL_BUTTON,
                ID_BLUE2,
                30, 110, 80, 30,
                WS_VISIBLE,
                WS_EX_NONE,
                "BLUE 2",
                NULL,
                NULL,
                NULL,
```

```
                NULL,
                0,
        },
};
// END_OF_BLUEGROUP

// START_OF_PANEL
static NCS_WND_TEMPLATE panel_tmpl[] = {
        {
                NCSCTRL_PANEL,
                ID_PANEL_GROUP1,
                10, 10, 140, 180,
                WS_VISIBLE,
                WS_EX_NONE,
                "Red Group",
                NULL,
                NULL,
                NULL,
                _ctrl_group1,
                sizeof(_ctrl_group1)/sizeof(NCS_WND_TEMPLATE),
                0,
                0xFF0000FF,
        },
        {
```

```
                NCSCTRL_PANEL,
                ID_PANEL_GROUP2,
                160, 10, 140, 180,
                WS_VISIBLE,
                WS_EX_NONE,
                "Blue Group",
                NULL,
                NULL,
                NULL,
                _ctrl_group2,
                sizeof(_ctrl_group2)/sizeof(NCS_WND_TEMPLATE),
                0,
                0xFFFF0000,
        },
};
// END_OF_PANEL

// START_OF_HANDLERS
static BOOL mainwnd_onCreate(mWidget* self, DWORD add_data)
{
        _c(self)->addChildren(self, panel_tmpl, \
        sizeof(panel_tmpl)/sizeof(NCS_WND_TEMPLATE));
        return TRUE;
}
```

```
static void mainwnd_onClose(mWidget* self, int message)
{
        DestroyMainWindow(self->hwnd);
        PostQuitMessage(0);
}


static NCS_EVENT_HANDLER mainwnd_handlers[] = {
        {MSG_CREATE, mainwnd_onCreate},
        {MSG_CLOSE, mainwnd_onClose},
        {0, NULL}
};
// END_OF_HANDLERS

int MiniGUIMain(int argc, const char* argv[])
{
        MSG Msg;

        ncsInitialize ();

        mWidget* mainwnd = ncsCreateMainWindow (
        NCSCTRL_MAINWND, "Panel Demo",
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
```

```
        WS_EX_NONE,
        1,
        0, 0, 320,240,
        HWND_DESKTOP,
        0, 0,
        NULL,
        NULL,
        mainwnd_handlers,
        0);

        // START_OF_MSGLOOP
        while (GetMessage (&Msg, mainwnd->hwnd)) {
                TranslateMessage (&Msg);
                DispatchMessage (&Msg);
        }
        // END_OF_MSGLOOP

        MainWindowThreadCleanup (mainwnd->hwnd);
        ncsUninitialize ();

        return 0;
}
```

mCombobox

控件名称

NCSCTRL_COMBOBOX

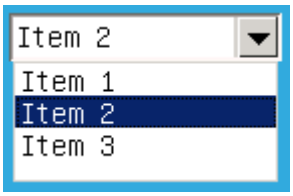
英文名

Combobox

简要介绍

融合了一个编辑框和一个列表框,用户可以直接在编辑框中键入文本,也可以从列表框列出的可选项中选择一个已有的条目.可以很好的完成用户的输入和选择功能。一般浏览器中的地址栏就是一个很好的应用。

示意图



mCombobox 风格

继承自 mPanel 的风格

风格名	miniStudio 属性名	说明
NCSS_CMBOX_SIMPLE	Type->Simple	简单组合框
NCSS_CMBOX_DROPDOWNLIST	Type->DropDownList	下拉组合框
NCSS_CMBOX_SORT	Sort->TRUE	字符串自动排序
NCSS_CMBOX_EDITNOBORDER	EditHasBorder ->FALSE	编辑框没有边框
NCSS_CMBOX_EDITBASELINE	EditHasBaseLine ->TRUE	编辑框显示带下划线字符
NCSS_CMBOX_READONLY	ReadOnly	编辑框显示的内容只读
NCSS_CMBOX_UPPERCASE	Case->Upper	编辑框显示内容全部转成大写字母显示
NCSS_CMBOX_LOWERCASE	Case->Lower	编辑框显示内容全部转成小写字母显示
NCSS_CMBOX_AUTOFOCUS	AutoFocus ->TRUE	控件获取焦点自动转给编辑框

mCombobox 属性

继承自 mPanel 的属性

属性名	miniStudio 属性名	类型	RW	说明
NCSP_COMB_DROPDOWNHEIGHT	DropDownHeight	int	RW	下拉列表的高度

NCSP_COMB_ITEMHEIGHT	ItemHeight	int	RW	列表项的高度
NCSP_COMB_ITEMCOUNT	ItemCount	int	RO	列表项数目
NCSP_COMB_TEXTLIMIT	TextLimit	int	RW	编辑框限定
NCSP_COMB_SELECT	--	int	RW	选择项索引

mCombobox 事件

继承自 mPanel 的事件

事件 ID	参数	说明
NCSN_CMBOX_SELCHANGE	--	选中项改变
NCSN_CMBOX_SETFOCUS	--	获得焦点
NCSN_CMBOX_KILLFOCUS	--	失去焦点
NCSN_CMBOX_EDITCHANGE	--	编辑框内容改变
NCSN_CMBOX_DROPDOWN	--	下拉列表弹出
NCSN_CMBOX_CLOSEUP	--	下拉列表关闭
NCSN_CMBOX_SELECTOK	--	下拉列表关闭时选择一项
NCSN_CMBOX_SELECTCANCEL	--	下拉列表关闭时没有选择

mCombobox 方法

继承自 mPanel 的方法

addItem

```
BOOL addItem(mCombobox *self, const char *item, DWORD addData);
```

- 参数：
  - item -- 添加选择项的内容
  - addData -- 该项的附加数据
- 说明：
  给 combobox 的下拉列表添加选择项
- 示例：

```
char *items = {
        "first item --- Chinese",
        "second item --- German",
        "third item -- English"
};
//向下拉列表添加三项
for (i = 0; i < 3; i++)
{
        _c(combo)->addItem(combo, items[i],  0);
}
```

removeItem

```
BOOL removeItem(mCombobox *self, int index);
```

- 参数：
  - index -- 要删除项的索引
- 说明：
  从 combobox 的下拉列表删除某个选择项
- 示例：

```
//删除掉下拉列表的第一项
_c(combo)->removeItem(combo, 0);
```

setItem

```
BOOL setItem(mCombobox *self, int index, const char *item);
```

- 参数：
  - index -- 要修改项的索引
- 说明：
  修改 combobox 的下拉列表某个选择项的内容
- 示例：

```
//修改下拉列表的第一项的内容为“new content”

_c(combo)->setItem(combo, 0, "new content");
```

getItem

```
const char* getItem(mCombobox *self, int index);
```

- 参数：
  - index -- 要获取项的索引
- 说明：
  获取 combobox 的下拉列表某个选择项的内容
- 示例：

```
const char *item_1 = _c(combo)->getItem(combo, 0);
```

setAddData

```
void* setAddData(mCombobox *self, int index, DWORD addData);
```

- 参数：
  - index -- 要设置附加数据的项的索引
  - addData -- 附加数据信息
- 说明：
  设置 combobox 的下拉列表某个选择项的附加数据
- 示例：

```
PBITMAP pbmp;

LoadBitmap (......);

_c(combo)->setAddData(combo, 0, (DWORD)pbmp);
```

getAddData

```
DWORD getAddData(mCombobox *self, int index);
```

- 参数：
  - index -- 要获取附加数据的项的索引

- 说明：
  获取 combobox 的下拉列表某个选择项的附加数据
- 示例：

```
DWORD add = _c(combo)->getAddData(combo, 0);
```

mCombobox 渲染器

继承自 mPanel 渲染器

mCombobox Classic 渲染器

属性名	miniStudio 属性名	类型	示意图	说明
NCS_BGC_3DBODY	ColorBg3DBody	DWORD(ARGB)		绘制下拉按钮的颜色
NCS_FGC_WINDOW	ColorFgWindow	DWORD(ARGB)		绘制下拉按钮的箭头的颜色

mCombobox Fashion 渲染器

属性名	miniStudio 属性名	类型	示意图	说明
NCS_BGC_3DBODY	ColorBg3DBody	DWORD(ARGB)		绘制下拉按钮的颜色
NCS_FGC_WINDOW	ColorFgWindow	DWORD(ARGB)		绘制下拉按钮的箭头的颜色
NCS_METRICS_3DBODY_ROUNDX	RoundX	int		下拉按钮的圆角 x 向半径
NCS_METRICS_3DBODY_ROUNDY	RoundY	int		下拉按钮的圆角 y 向半径
NCS_MODE_BGC	GradientMode	int		渐变效果的绘制模式（水平渐变 or 竖直渐变）

mCombobox Skin 渲染器

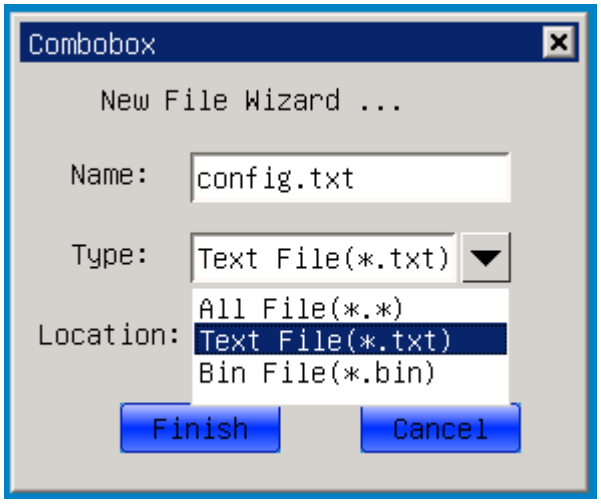
参阅 Skin 渲染器使用的图片资源规范(附录 B)中图片规范

mCombobox Flat 渲染器

属性名	miniStudio 属性名	类型	示意图	说明

NCS_BGC_3DBODY	ColorBg3DBody	DWORD(ARGB)	绘制下拉按钮的颜色
NCS_FGC_WINDOW	ColorFgWindow	DWORD(ARGB)	绘制下拉按钮的箭头的颜色

mCombobox 实例



清单 p2c6-1 combobox.c

```
/*
** $Id$
**
** Listing P2C1.4
**
** combobox.c: Sample program for mGNCS Programming Guide
**      Using Combobox.
**
** Copyright (C) 2009 Feynman Software.
*/

#include <stdio.h>
```

```
#include <stdlib.h>

#include <string.h>


#include <minigui/common.h>

#include <minigui/minigui.h>

#include <minigui/gdi.h>

#include <minigui/window.h>

#include <minigui/control.h>


#include <mgncs/mgncs.h>



#define ID_BTN  201

#define ID_NAME 202

#define ID_TYPE 203

#define ID_LOCA 204


static const char *file_type[] =

{

        "All File(*.*)",

        "Text File(*.txt)",

        "Bin File(*.bin)",

};
```

```
static BOOL mymain_onCreate (mWidget* _this, DWORD add_data)
{
        int i;

        // get combobox
        mCombobox *com = (mCombobox *)ncsGetChildObj(_this->hwnd, ID_TYPE);

        // add items
        for (i = 0; i < sizeof(file_type)/sizeof(file_type[0]); i++)
        {
                _c(com)->addItem(com, file_type[i], 0);
        }

        // set the selected item
        _c(com)->setProperty(com, NCSP_COMB_SELECT, 1);

        return TRUE;
}

static void mymain_onClose (mWidget* _this, int message)
{
        DestroyMainWindow (_this->hwnd);
        PostQuitMessage (_this->hwnd);
}
```

```
static void mymain_onPaint(mWidget *self, HDC hdc, const CLIPRGN* inv)
{
        SetBkMode (hdc, BM_TRANSPARENT);
        TextOut (hdc, 40, 10, "New File Wizard ...");
}

static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate},
        {MSG_CLOSE,  mymain_onClose},
        {MSG_PAINT,  mymain_onPaint},
        {0, NULL}
};

static void btn_onClicked(mWidget* _this, int id, int nc, HWND hCtrl)
{
        if(nc == NCSN_WIDGET_CLICKED)
        {
                PostMessage(GetParent(_this->hwnd), MSG_CLOSE, 0, 0);
        }
};

static NCS_EVENT_HANDLER btn_handlers[] =
{
```

```
        {NCS_NOTIFY_CODE(NCSN_WIDGET_CLICKED), btn_onClicked},
        {0, NULL}
};

static NCS_RDR_INFO btn_rdr_info[] =
{
        {"fashion","fashion", NULL}
};

//START_OF_INITIAL_PROPS
static NCS_PROP_ENTRY combo_props[] =
{
        { 0, 0 }
};
//END_OF_INITIAL_PROPS

//START_OF_TEMPLATE
static NCS_WND_TEMPLATE _ctrl_templ[] =
{
        {
                NCSCTRL_STATIC,
                0,
                10, 40, 70, 30,
                WS_VISIBLE,
```

```
            WS_EX_NONE,
            "Name:",
            NULL,
            NULL,
            NULL, NULL, 0, 0
        },
        {
            NCSCTRL_SLEDIT,
            ID_NAME,
            85, 45, 160, 25,
            WS_VISIBLE | WS_BORDER,
            WS_EX_NONE,
            "",
            combo_props,
            NULL,
            NULL, NULL, 0, 0
        },
        {
            NCSCTRL_STATIC,
            0,
            10, 80, 70, 30,
            WS_VISIBLE,
            WS_EX_NONE,
            "Type:",
```

```
            NULL,
            NULL,
            NULL, NULL, 0, 0
        },
        {
            NCSCTRL_COMBOBOX,
            ID_TYPE,
            85, 85, 160, 25,
            WS_VISIBLE | NCSS_CMBOX_DROPDOWNLIST,
            WS_EX_NONE,
            "",
            combo_props,
            NULL,
            NULL, NULL, 0, 0
        },
        {
            NCSCTRL_STATIC,
            0,
            10, 120, 70, 30,
            WS_VISIBLE,
            WS_EX_NONE,
            "Location:",
            NULL,
            NULL,
```

```
        NULL, NULL, 0, 0
    },
    {
        NCSCTRL_SLEDIT,
        ID_LOCA,
        85, 125, 160, 25,
        WS_VISIBLE | WS_BORDER,
        WS_EX_NONE,
        "",
        combo_props,
        NULL,
        NULL, NULL, 0, 0
    },
    {
        NCSCTRL_BUTTON,
        ID_BTN,
        50, 170, 80, 25,
        WS_VISIBLE | NCSS_NOTIFY,
        WS_EX_NONE,
        "Finish",
        NULL,
        btn_rdr_info,
        btn_handlers, NULL, 0, 0
    },
```

```
        {
                NCSCTRL_BUTTON,
                ID_BTN,
                170, 170, 80, 25,
                WS_VISIBLE | NCSS_NOTIFY,
                WS_EX_NONE,
                "Cancel",
                NULL,
                btn_rdr_info,
                btn_handlers, NULL, 0, 0
        },

};
//END_OF_TEMPLATE

static NCS_MNWND_TEMPLATE mymain_templ =
{
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 320, 240,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Combobox",
        NULL,
```

```
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};

int MiniGUIMain (int argc, const char* argv[])
{
        ncsInitialize ();

        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);

        _c(mydlg)->doModal (mydlg, TRUE);

        ncsUninitialize ();

        return 0;
}
```

第二部分第七章 容器及其派生类

容器类控件简介

该类控件是容纳其它各类控件的容器类控件，与面板类控件不同的是，该类控件提供默认的窗口滚动处理。

窗口及其派生类的类继承关系如下：

- mWidget
  - mScrollWidget
    - mContainer
      - mPage

mScrollWidget

控件名称

NCSCTRL_SCROLLWIDGET

英文名

ScrollWidget

简要介绍

所有包括滚动功能控件类的抽象基类。

示意图

抽象类,不能直接使用

mScrollWidget 风格

继承自 mWidget 的风格

mScrollWidget 属性

继承自 mWidget 的属性

属性 ID	miniStudio 名	类型	权限	说明
NCSP_SWGT_CONTWIDTH	-	unsigned int	RW	内容宽度
NCSP_SWGT_CONTHEIGHT	-	unsigned int	RW	内容高度
NCSP_SWGT_CONTX	ContentX	int	RW	内容在水平方向的偏移值

NCSP_SWGT_CONTY	ContentY	int	RW	内容在垂直方向的偏移值
NCSP_SWGT_HSTEPVALUE	HStepValue	unsigned int	RW	滚动内容时的水平步进值
NCSP_SWGT_VSTEPVALUE	VStepValue	unsigned int	RW	滚动内容时的垂直步进值
NCSP_SWGT_VISWIDTH	-	unsigned int	RW	可视区宽度
NCSP_SWGT_VISHEIGHT	-	unsigned int	RW	可视区高度
NCSP_SWGT_MARGINRECT	-	unsigned int	RW	边界矩形
NCSP_SWGT_DRAWMODE	ScrollBarMode	ncsSwgtDrawMode	RW	滚动条显示模式

mScrollWidget 事件

继承自 mWidget 的事件

mScrollWidget 方法

继承自 mWidget 的方法

ScrollWidget 的所有方法均是提供给子类使用的，请避免在具体的控件应用中使用该类接口。

坐标转换相关方法

在处理滚动的窗口内容时涉及到三类坐标：

- content：指定内容在全部内容中的坐标。
- viewport：指定内容在可视区域中的坐标。受内容滚动的影响。
- window：指定内容在窗口中的显示坐标。受 margin 区域影响。

其中 content 为显示内容的原始坐标，window 为在 MiniGUI 窗口中的坐标。若要将一个原始内容的 content 坐标转换为显示用的 window 坐标，二者间需要经过转换，即首先需要将 content 坐标转换到 viewport 坐标，就好比一共几页的内容，把某一页的内容作为当前内容显示的话，该页某一部分的坐标将从相对于整体内容的开始位置到相对于该页起始位置。在最终将内容显示到窗口上时，还需要进行一步 viewport 到 window 的转换，这是因为在窗口的客户区经常会添加一部分 margin，以调整显示效果。相反，将 MiniGUI 窗口中显示的内容位置转换为其在所有内容中的位置也需要相反的两步工作。

三类坐标的关系示意图：

三类坐标之间转换的相应接口为：

```
void contentToViewport(mScrollWidget *self, int *x, int *y);

void viewportToWindow(mScrollWidget *self, int *x, int *y);

void contentToWindow(mScrollWidget *self, int *x, int *y);



void windowToViewport(mScrollWidget *self, int *x, int *y);

void viewportToContent(mScrollWidget *self, int *x, int *y);

void windowToContent(mScrollWidget *self, int *x, int *y);
```

初始化相关方法

ScrollWidget 控件类默认将 margin 配置为全 0 值，当基于该控件类的子类控件需要与父类不同的 margin 配置时，可通过在构造函数中使用 initMargins 方法重新配置。

```
void initMargins(mScrollWidget *self, int l, int t, int r, int b);
```

如 iconview 控件类在 construct 方法中使用下面的代码初始化上下左右的 margin 值均为 5。

```
_c(self)->initMargins(self, 5, 5, 5, 5);
```

刷新相关方法

ScrollWidget 在自动调整滚动条区域和对显示区域相关的配置设置时需要对屏幕进行刷新，为了满足不同控件的处理需要，提供了两个回调方便上层控件处理，同时 ScrollWidget 也提供了默认处理实现：

```
void moveContent(mScrollWidget *self);

void refreshRect(mScrollWidget *self, const RECT *rc);
```

- moveContent 多用于滚动内容后的更新，如滚动窗口后的全屏刷新。
- refreshRect 用于刷新指定区域的内容。

获取信息方法

ScrollWidget 提供 isVisible 方法供子类在判断某一内容位置在当前显示区域是否可见，以让子类判断是否需要滚屏等操作。同时在子类进行内容绘制时，需要要内容绘制在指定的可视区，getVisRect 方法为获取允许绘制的最大范围提供了便利。

```
BOOL (*isVisible)(mScrollWidget *self, int x, int y);

void (*getVisRect)(mScrollWidget *self, RECT *rcVis);
```

- isVisible 用来判断指定内容位置当前是否在可视区域内。
- getVisRect 用于获取当前可视区域。

设置相关方法

ScrollWidget 及其派生类在窗口大小改变或内容改动时均需要对不同信息进行相应的设置，因此 ScrollWidget 提供了下面几个方法：

```
void resetViewPort(mScrollWidget *self, unsigned int visW,unsigned int visH);

void setScrollInfo(mScrollWidget *self, BOOL reDraw);

BOOL makePosVisible(mScrollWidget *self, int x, int y);
```

- resetViewport 用于重新调整可视区域的大小。如窗口大小改变时，需要使用该接口调整可视区。
- setScrollInfo 方法提供给子类控件使用，当可视区域或内容区域的大小变化时进行相应的滚动条信息设置。
- makePosVisible 用于自动将指定的内容位置显示在可视区内。

mScrollWidget 渲染器

继承自 mWidget 渲染器

mScrollWidget 无新增加渲染器方法。

mContainer

控件名称

NCSCTRL_CONTAINER

英文名

Container

简要介绍

具有滚动支持的容器类控件。

mContainer 风格

继承自 mScrollWidget 风格

mContainer 属性

继承自 mScrollWidget 属性

mContainer 事件

继承自 mScrollWidget 事件

mContainer 方法

继承自 mScrollWidget 方法

添加控件方法

Container 对 mGNCS 控件和 MiniGUI 固有控件都提供支持。如果需要向 Container 中添加一组 mGNCS 控件，可使用从 mWidget 类继承下来的 addChildren 方法。

```
BOOL addChildren(mContainer *self, NCS_WND_TEMPLATE* children, int count);


static NCS_PROP_ENTRY radioGroup_props [] = {
        {NCSP_BUTTON_GROUPID, IDC_RDOGROUP},
        {0, 0}
};


static NCS_WND_TEMPLATE ncsCtrlYourTaste[] = {
        {
                NCSCTRL_BUTTONGROUP,
                IDC_RDOGROUP,
```

```
            16, 10, 230, 160,
            WS_VISIBLE,
            WS_EX_TRANSPARENT,
            "optional snack",
            NULL,
            NULL,
            NULL,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_RADIOBUTTON,
            IDC_LAMIAN,
            36, 38, 200, 20,
            WS_VISIBLE | WS_TABSTOP | NCSS_BUTTON_AUTOCHECK,
            WS_EX_NONE,
            "northwest pulled noodle",
            radioGroup_props,
            NULL,
            NULL,
            NULL,
            0,
            0
```

```
    },
    {
        NCSCTRL_RADIOBUTTON,
        IDC_CHOUDOUFU,
        36, 64, 200, 20,
        WS_VISIBLE | WS_TABSTOP | NCSS_BUTTON_AUTOCHECK,
        WS_EX_NONE,
        "chang sha bad smelling bean curd",
        radioGroup_props,
        NULL,
        NULL,
        NULL,
        0,
        0
    },
    {
        NCSCTRL_RADIOBUTTON,
        IDC_JIANBING,
        36, 90, 200, 20,
        WS_VISIBLE | WS_TABSTOP | WS_DISABLED,
        WS_EX_NONE,
        "shan dong thini pancake",
        radioGroup_props,
        NULL,
```

```
            NULL,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_RADIOBUTTON,
            IDC_MAHUA,
            36, 116, 200, 20,
            WS_VISIBLE | WS_TABSTOP | NCSS_BUTTON_AUTOCHECK,
            WS_EX_NONE,
            "tianjin fired dough twist",
            radioGroup_props,
            NULL,
            NULL,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_RADIOBUTTON,
            IDC_SHUIJIAO,
            36, 142, 200, 20,
            WS_VISIBLE | WS_TABSTOP | NCSS_BUTTON_AUTOCHECK,
```

```
        WS_EX_NONE,
        "chengdu red oil boiled dumpling",
        radioGroup_props,
        NULL,
        NULL,
        NULL,
        0,
        0
    },
    {
        NCSCTRL_BUTTONGROUP,
        IDC_CKGROUP,
        250, 10, 100, 160,
        WS_VISIBLE,
        WS_EX_TRANSPARENT,
        "flavor",
        NULL,
        NULL,
        NULL,
        NULL,
        0,
        0
    },
    {
```

```
        NCSCTRL_CHECKBUTTON,
        IDC_XIAN,
        260, 38, 80, 20,
        WS_VISIBLE | NCSS_BUTTON_AUTOCHECK,
        WS_EX_NONE,
        "partial salty",
        //checkGroup_props,
        NULL,
        NULL,
        NULL,
        NULL,
        0,
        0
    },
    {
        NCSCTRL_CHECKBUTTON,
        IDC_LA,
        260, 64, 80, 20,
        WS_VISIBLE | NCSS_BUTTON_AUTOCHECK,
        WS_EX_NONE,
        "partial spicy",
        //checkGroup_props,
        NULL,
        NULL,
```

```
        NULL,
        NULL,
        0,
        0
    },
    {
        NCSCTRL_STATIC,
        IDC_PROMPT,
        16, 180, 360, 40,
        WS_VISIBLE,
        WS_EX_NONE,
        "northwest pulled noodle is competitive product in the wheaten food",
        NULL,
        NULL,
        NULL,
        NULL,
        0,
        0
    },
    {
        NCSCTRL_BUTTON,
        IDOK,
        70, 230, 70, 30,
        WS_VISIBLE | WS_TABSTOP | NCSS_NOTIFY,
```

```
            WS_EX_NONE,
            "Ok",
            NULL,
            NULL,
            NULL,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_BUTTON,
            IDCANCEL,
            200, 230, 70, 30,
            WS_VISIBLE | WS_TABSTOP | NCSS_NOTIFY,
            WS_EX_NONE,
            "Cancel",
            NULL,
            NULL,
            NULL,
            NULL,
            0,
            0
        },
};
```

向 Container 中添加一组 MiniGUI 固有控件，可使用 addIntrinsicControls 方法：

```
BOOL addIntrinsicControls(mContainer *self, const PCTRLDATA pCtrl, int nCount);


static CTRLDATA CtrlYourTaste[] =
{
        {
                "static",
                WS_VISIBLE | SS_GROUPBOX,
                16, 10, 230, 160,
                IDC_STATIC,
                "optional snack",
                0,
                WS_EX_TRANSPARENT
        },
        {
                "button",
                WS_VISIBLE | BS_AUTORADIOBUTTON | BS_CHECKED | WS_TABSTOP | WS_GROUP,
                36, 38, 200, 20,
                IDC_LAMIAN,
                "northwest pulled noodle",
                0
        },
        {
                "button",
```

```
            WS_VISIBLE | BS_AUTORADIOBUTTON,
            36, 64, 200, 20,
            IDC_CHOUDOUFU,
            "chang sha bad smelling bean curd",
            0
        },
        {
            "button",
            WS_VISIBLE | BS_AUTORADIOBUTTON | WS_DISABLED,
            36, 90, 200, 20,
            IDC_JIANBING,
            "shan dong thini pancake",
            0
        },
        {
            "button",
            WS_VISIBLE | BS_AUTORADIOBUTTON,
            36, 116, 200, 20,
            IDC_MAHUA,
            "tianjin fired dough twist",
            0
        },
        {
            "button",
```

```
        WS_VISIBLE | BS_AUTORADIOBUTTON,
        36, 142, 200, 20,
        IDC_SHUIJIAO,
        "chengdu red oil boiled dumpling",
        0
    },
    {
        "static",
        WS_VISIBLE | SS_GROUPBOX | WS_GROUP,
        250, 10, 100, 160,
        IDC_STATIC,
        "flavor",
        0,
        WS_EX_TRANSPARENT
    },
    {
        "button",
        WS_VISIBLE | BS_AUTOCHECKBOX,
        260, 38, 80, 20,
        IDC_XIAN,
        "partial salty",
        0
    },
    {
```

```
            "button",
            WS_VISIBLE | BS_AUTOCHECKBOX | BS_CHECKED,
            260, 64, 80, 20,
            IDC_LA,
            "partial spicy",
            0
        },
        {
            "static",
            WS_VISIBLE | SS_LEFT | WS_GROUP | WS_BORDER,
            16, 180, 360, 40,
            IDC_PROMPT,
            "northwest pulled noodle is competitive product in the wheaten food",
            0
        },
        {
            "button",
            WS_VISIBLE | BS_DEFPUSHBUTTON | WS_TABSTOP | WS_GROUP | WS_BORDER ,
            70, 230, 70, 30,
            IDOK,
            "OK",
            0
        },
        {
```

```
                "button",
                WS_VISIBLE | BS_PUSHBUTTON | WS_TABSTOP,
                200, 230, 70, 30,
                IDCANCEL,
                "Cancel",
                0
        },
};
```

焦点相关方法

在 Container 中经常需要设置和获取当前焦点控件，对此提供了下面 2 个方法：

```
HWND setFocus(mContainer *self, int id);

HWND getFocus(mContainer *self);
```

- setFocus 通过控件 ID 设置焦点控件。
- getFocus 获取当前焦点控件的句柄。

其它方法

Container 提供 adjustContent 方法允许应用通过其内容进行自身大小的调整功能，而不是通过 Container 控制控件显示范围。getPanel 方法可以获取当前 Container 内容的 hosting 窗口句柄。

```
HWND getPanel(mContainer *self);

void adjustContent(mContainer *self);
```

mContainer 渲染器

继承自 mScrollWidget 渲染器

mContainer 无新增加渲染器方法。

mContainer 实例

本实例为用户演示了如何使用 Container 实现在超出窗口大小的多个控件间跳转。

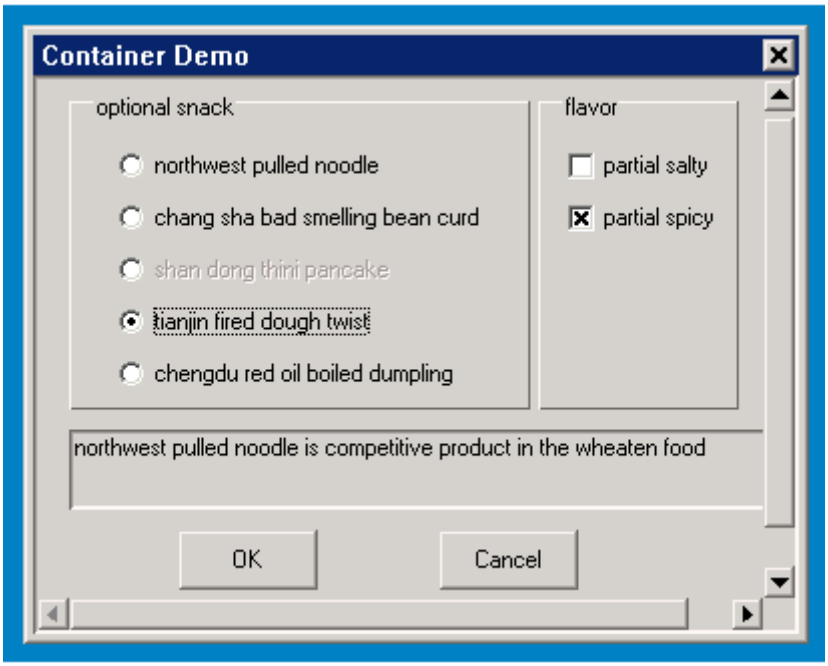


图 p2c6-1 Container 程序的输出

清单 p2c7-1 container.c

```
/*
** $Id: container.c 596 2009-10-10 08:49:44Z xwyan $
**
** Listing P2C7.1
**
** container.c: Sample program for mGNCS Programming Guide
**      The demo application for Container.
**
** Copyright (C) 2009 Feynman Software.
*/
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>

// START_OF_INCS
#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/control.h>

#include <mgncs/mgncs.h>
// END_OF_INCS

#define IDC_CONTAINER   100

#define IDC_RDOGROUP    151
#define IDC_CKGROUP     152

#define IDC_LAMIAN      101
#define IDC_CHOUDOUFU   102
#define IDC_JIANBING    103
#define IDC_MAHUA       104
#define IDC_SHUIJIAO    105
#define IDC_XIAN        110
```

```
#define IDC_LA          111
#define IDC_PROMPT      200


// START_OF_NCSCTRLS
static NCS_PROP_ENTRY radioGroup_props [] = {
        {NCSP_BUTTON_GROUPID, IDC_RDOGROUP},
        {0, 0}
};


static NCS_WND_TEMPLATE ncsCtrlYourTaste[] = {
        {
                NCSCTRL_BUTTONGROUP,
                IDC_RDOGROUP,
                16, 10, 230, 160,
                WS_VISIBLE,
                WS_EX_TRANSPARENT,
                "optional snack",
                NULL,
                NULL,
                NULL,
                NULL,
                0,
                0
        },
```

```
    {
        NCSCTRL_RADIOBUTTON,
        IDC_LAMIAN,
        36, 38, 200, 20,
        WS_VISIBLE | WS_TABSTOP | NCSS_BUTTON_AUTOCHECK,
        WS_EX_NONE,
        "northwest pulled noodle",
        radioGroup_props,
        NULL,
        NULL,
        NULL,
        0,
        0
    },
    {
        NCSCTRL_RADIOBUTTON,
        IDC_CHOUDOUFU,
        36, 64, 200, 20,
        WS_VISIBLE | WS_TABSTOP | NCSS_BUTTON_AUTOCHECK,
        WS_EX_NONE,
        "chang sha bad smelling bean curd",
        radioGroup_props,
        NULL,
        NULL,
```

```
        NULL,
        0,
        0
    },
    {
        NCSCTRL_RADIOBUTTON,
        IDC_JIANBING,
        36, 90, 200, 20,
        WS_VISIBLE | WS_TABSTOP | WS_DISABLED,
        WS_EX_NONE,
        "shan dong thini pancake",
        radioGroup_props,
        NULL,
        NULL,
        NULL,
        0,
        0
    },
    {
        NCSCTRL_RADIOBUTTON,
        IDC_MAHUA,
        36, 116, 200, 20,
        WS_VISIBLE | WS_TABSTOP | NCSS_BUTTON_AUTOCHECK,
        WS_EX_NONE,
```

```
            "tianjin fired dough twist",
            radioGroup_props,
            NULL,
            NULL,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_RADIOBUTTON,
            IDC_SHUIJIAO,
            36, 142, 200, 20,
            WS_VISIBLE | WS_TABSTOP | NCSS_BUTTON_AUTOCHECK,
            WS_EX_NONE,
            "chengdu red oil boiled dumpling",
            radioGroup_props,
            NULL,
            NULL,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_BUTTONGROUP,
```

```
            IDC_CKGROUP,
            250, 10, 100, 160,
            WS_VISIBLE,
            WS_EX_TRANSPARENT,
            "flavor",
            NULL,
            NULL,
            NULL,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_CHECKBUTTON,
            IDC_XIAN,
            260, 38, 80, 20,
            WS_VISIBLE | NCSS_BUTTON_AUTOCHECK,
            WS_EX_NONE,
            "partial salty",
            //checkGroup_props,
            NULL,
            NULL,
            NULL,
            NULL,
```

```
            0,
            0
        },
        {
            NCSCTRL_CHECKBUTTON,
            IDC_LA,
            260, 64, 80, 20,
            WS_VISIBLE | NCSS_BUTTON_AUTOCHECK,
            WS_EX_NONE,
            "partial spicy",
            //checkGroup_props,
            NULL,
            NULL,
            NULL,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_STATIC,
            IDC_PROMPT,
            16, 180, 360, 40,
            WS_VISIBLE,
            WS_EX_NONE,
```

```
        "northwest pulled noodle is competitive product in the wheaten food",
        NULL,
        NULL,
        NULL,
        NULL,
        0,
        0
    },
    {
        NCSCTRL_BUTTON,
        IDOK,
        70, 230, 70, 30,
        WS_VISIBLE | WS_TABSTOP | NCSS_NOTIFY,
        WS_EX_NONE,
        "Ok",
        NULL,
        NULL,
        NULL,
        NULL,
        0,
        0
    },
    {
        NCSCTRL_BUTTON,
```

```
                IDCANCEL,
                200, 230, 70, 30,
                WS_VISIBLE | WS_TABSTOP | NCSS_NOTIFY,
                WS_EX_NONE,
                "Cancel",
                NULL,
                NULL,
                NULL,
                NULL,
                0,
                0
        },
};
// END_OF_NCSCTRLS


// START_OF_INTRINSICCTRLS
static CTRLDATA CtrlYourTaste[] =
{
        {
                "static",
                WS_VISIBLE | SS_GROUPBOX,
                16, 10, 230, 160,
                IDC_STATIC,
                "optional snack",
```

```
        0,
        WS_EX_TRANSPARENT
    },
    {
        "button",
        WS_VISIBLE | BS_AUTORADIOBUTTON | BS_CHECKED | WS_TABSTOP | WS_GROUP,
        36, 38, 200, 20,
        IDC_LAMIAN,
        "northwest pulled noodle",
        0
    },
    {
        "button",
        WS_VISIBLE | BS_AUTORADIOBUTTON,
        36, 64, 200, 20,
        IDC_CHOUDOUFU,
        "chang sha bad smelling bean curd",
        0
    },
    {
        "button",
        WS_VISIBLE | BS_AUTORADIOBUTTON | WS_DISABLED,
        36, 90, 200, 20,
        IDC_JIANBING,
```

```
            "shan dong thini pancake",
            0
        },
        {
            "button",
            WS_VISIBLE | BS_AUTORADIOBUTTON,
            36, 116, 200, 20,
            IDC_MAHUA,
            "tianjin fired dough twist",
            0
        },
        {
            "button",
            WS_VISIBLE | BS_AUTORADIOBUTTON,
            36, 142, 200, 20,
            IDC_SHUIJIAO,
            "chengdu red oil boiled dumpling",
            0
        },
        {
            "static",
            WS_VISIBLE | SS_GROUPBOX | WS_GROUP,
            250, 10, 100, 160,
            IDC_STATIC,
```

```
        "flavor",
        0,
        WS_EX_TRANSPARENT
    },
    {
        "button",
        WS_VISIBLE | BS_AUTOCHECKBOX,
        260, 38, 80, 20,
        IDC_XIAN,
        "partial salty",
        0
    },
    {
        "button",
        WS_VISIBLE | BS_AUTOCHECKBOX | BS_CHECKED,
        260, 64, 80, 20,
        IDC_LA,
        "partial spicy",
        0
    },
    {
        "static",
        WS_VISIBLE | SS_LEFT | WS_GROUP | WS_BORDER,
        16, 180, 360, 40,
```

```
                IDC_PROMPT,
                "northwest pulled noodle is competitive product in the wheaten food",
                0
        },
        {
                "button",
                WS_VISIBLE | BS_DEFPUSHBUTTON | WS_TABSTOP | WS_GROUP | WS_BORDER ,
                70, 230, 70, 30,
                IDOK,
                "OK",
                0
        },
        {
                "button",
                WS_VISIBLE | BS_PUSHBUTTON | WS_TABSTOP,
                200, 230, 70, 30,
                IDCANCEL,
                "Cancel",
                0
        },
};
// END_OF_INTRINSICCTRLS

// START_OF_HANDLERS
```

```
static void dialog_onCSizeChanged(mWidget* self, int clientWidth, int clientHeight)
{
        HWND hContainer = GetDlgItem(self->hwnd, IDC_CONTAINER);
        if (hContainer != HWND_NULL || hContainer != HWND_INVALID)
        MoveWindow(hContainer, 0, 0, clientWidth, clientHeight, TRUE);
}

static BOOL container_onCommand(mWidget* self, int id, int nc, HWND hCtrl)
{
        if (id == IDOK || id == IDCANCEL) {

                if (nc == NCSN_BUTTON_PUSHED) {
                        //close dialog
                        HWND hParent = GetParent(self->hwnd);
                        SendNotifyMessage(hParent,
                        MSG_COMMAND, (WPARAM)MAKELONG(id, nc), (LPARAM)hCtrl);
                }

                return FALSE;
        }

        return FALSE;
}
```

```
static NCS_EVENT_HANDLER container_handlers[] = {
        {MSG_COMMAND, container_onCommand},
        {0, NULL}
};
// END_OF_HANDLERS

int MiniGUIMain(int argc, const char* argv[])
{
        ncsInitialize ();

        mDialogBox* dialog = (mDialogBox*)ncsCreateMainWindow (
        NCSCTRL_DIALOGBOX, "Container Demo",
        WS_CAPTION | WS_BORDER | WS_VISIBLE | NCSS_MNWND_MODE,
        WS_EX_NONE,
        1,
        0, 0, 400, 320,
        HWND_DESKTOP,
        0, 0,
        NULL,
        NULL,
        NULL,
        0);

        ncsSetComponentHandler((mComponent*)dialog,
```

```
MSG_CSIZECHANGED, dialog_onCSizeChanged);

mContainer *container =
(mContainer*)ncsCreateWindow (NCSCTRL_CONTAINER,
"",
WS_BORDER | WS_VISIBLE,
WS_EX_NONE,
IDC_CONTAINER,
0, 0, 300, 200, dialog->hwnd,
NULL, NULL, container_handlers, 0);

if (argc > 1 && strcmp(argv[1], "intrinsic") == 0) {
        _c(container)->addIntrinsicControls (container, CtrlYourTaste,
        sizeof(CtrlYourTaste)/sizeof(CTRLDATA));
}
else {
        _c(container)->addChildren(container, ncsCtrlYourTaste,
        sizeof(ncsCtrlYourTaste)/sizeof(NCS_WND_TEMPLATE));
}

_c(dialog)->doModal(dialog, TRUE);

MainWindowThreadCleanup (dialog->hwnd);
ncsUninitialize ();
```

```
        return 0;
}
```

mPage

控件名称

NCSCTRL_PAGE

英文名

Page

简要介绍

属性页控件，用于显示同类信息及窗口内容。

mPage 风格

继承自 mContainer 风格

mPage 属性

继承自 mContainer 属性

mPage 事件

继承自 mContainer 事件

mPage 方法

继承自 mContainer 方法

mPage 渲染器

继承自 mContainer 渲染器

mPage 无新增加渲染器方法。

mPage 实例

无

第二部分第八章 滑动控件类

滑动控件简介

滑动控件主要是对一个值的调节逻辑的可视化实现， 通过拖拽滑动控件，使用户对调节亮度、音量等场合，以及需要对某一范围的量值进行调节的操作更加直观， 免去了键盘输入的麻烦。

- 滑动控件类层次关系
  - mWidget
    - mSlider
      - mTrackBar
      - mScrollBar

- 控件创建方法
  - 自动创建：通过 miniStudio 中的界面设计器，拖拽对应的滑动控件，miniStudio 将自动创建控件，并提供可视化的控件配置，同时自动生成创建代码。
  - 手动生成：按照 mGNCS 控件创建过程，通过编程，传入对应控件窗口类 ID，生成控件，手动编程设置控件属性、事件处理。

mSlider

控件窗口类

NCSCTRL_SLIDER

控件英文名

Slider

简介

Slider 系列控件的基本类。

示意图

该控件为抽象控件， 不能直接使用

mSlider 风格

继承自 mWidget 的风格

风格名	miniStudio 属性名	说明
NCSS_SLD_HORIZONTAL	--	创建水平的 Slider 控件（默认）
NCSS_SLD_VERTICAL	--	创建竖直的 Slider 控件

mSlider 属性

继承自 mWidget 的属性

属性	miniStudio 属性名	类型	RW	说明
NCSP_SLD_MAXPOS	--	int	RW	设置 Slider 的滑动范围最大值 最值的设定一般在初始化的时候做
NCSP_SLD_MINPOS	--	int	RW	设置 Slider 的滑动范围最小值
NCSP_SLD_CURPOS	--	int	RW	设置滑动块的当前位置
NCSP_SLD_LINESTEP	--	int	RW	设置步长（方向键）
NCSP_SLD_PAGESTEP	--	int	RW	设置步长（pageUp/pageDown）

mSlider 事件

继承自 mWidget 的事件

事件 ID	参数	说明
NCSN_SLD_CHANGED	--	滑块位置变化
NCSN_SLD_REACHMAX	--	滑块到达最大值
NCSN_SLD_REACHMIN	--	滑块到达最小值

mSlider 方法

继承自 mWidget 的方法

mSlider 示例

该控件为抽象控件， 不能直接使用

mTrackBar

控件窗口类

NCSCTRL_TRACKBAR

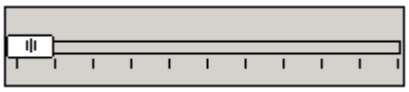
控件英文名

Trackbar

简介

滑块控件对范围内量值进行调节。

示意图



mTrackBar 风格

继承自 mSlider 的风格

风格	miniStudio 属性名	说明
NCSS_TRKBAR_HORIZONTAL	--	创建水平的 Trackbar 控件（默认）
NCSS_TRKBAR_VERTICAL	--	创建竖直的 Trackbar 控件
NCSS_TRKBAR_NOTICK	Ruler -> False	不显示刻度
	Ruler -> True	显示刻度

mTrackBar 属性

继承自 mSlider 的属性

属性名	miniStudio 属性名	类型	RW	说明	
NCSP_TRKBAR_MAXPOS	MaxPos	int	RW	设置 Trackbar 的滑动范围最大值	最值的设定一般在初始化的时候做
NCSP_TRKBAR_MINPOS	MinPos	int	RW	设置 Trackbar 的滑动范围最小值	
NCSP_TRKBAR_CURPOS	CurPos	int	RW	设置滑动块的当前位置	
NCSP_TRKBAR_LINESTEP	LineStep	int	RW	设置步长（方向键）	
NCSP_TRKBAR_PAGESTEP	PageStep	int	RW	设置步长（pageUp/pageDown）	

mTrackBar 事件

继承自 mSlider 的事件

事件 ID	参数	说明
NCSN_TRKBAR_CHANGED	--	滑块位置变化
NCSN_TRKBAR_REACHMAX	--	滑块到达最大值
NCSN_TRKBAR_REACHMIN	--	滑块到达最小值

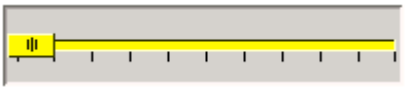
mTrackBar 方法

继承自 mSlider 的方法

mTrackBar 渲染器

mTrackBar Classic 渲染器

非客户区的绘制请查阅 mWidget 的渲染器

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域
NCS_BGC_3DBODY	滑块和滑轨颜色	ColorBg3DBody	DWORD(ARGB)		
NCS_BGC_DISABLED_ITEM	控件无效时滑块颜色	ColorBgDisable	DWORD(ARGB)		

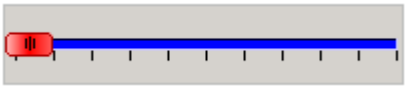
mTrackBar Skin 渲染器

参阅 附录 B : Skin 渲染器使用的图片资源规范

mTrackBar Fashion 渲染器

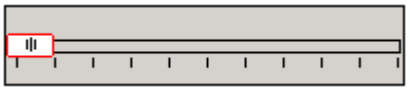
非客户区绘制请参阅 mWidget 的 Fashion 渲染器绘制

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域

NCS_BGC_3DBODY	滑块颜色	ColorBg3DBody	DWORD(ARGB)		
NCS_BGC_DISABLED_ITEM	控件无效时滑块颜色	ColorBgDisable	DWORD(ARGB)		
NCS_BGC_TRKBAR_SLIDER	滑轨颜色	SliderColor	DWORD(ARGB)		
NCS_METRICS_3DBODY_ROUNDX	滑块圆角 X 半径	ThumbRoundX	int		0 ~ 窗口宽度的 1/2
NCS_METRICS_3DBODY_ROUNDY	滑块圆角 Y 半径	ThumbRoundY	int		0 ~ 窗口高度的 1/2

mTrackBar Flat 渲染器

非客户区绘制请参阅 mWidget 的 Flat 渲染器绘制

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域
NCS_FGC_3DBODY	滑块边颜色	ColorFg3DBody	DWORD(ARGB)		
NCS_BGC_3DBODY	滑块颜色	ColorBg3DBody	DWORD(ARGB)	同 Classic 渲染器	

mTrackBar 示例

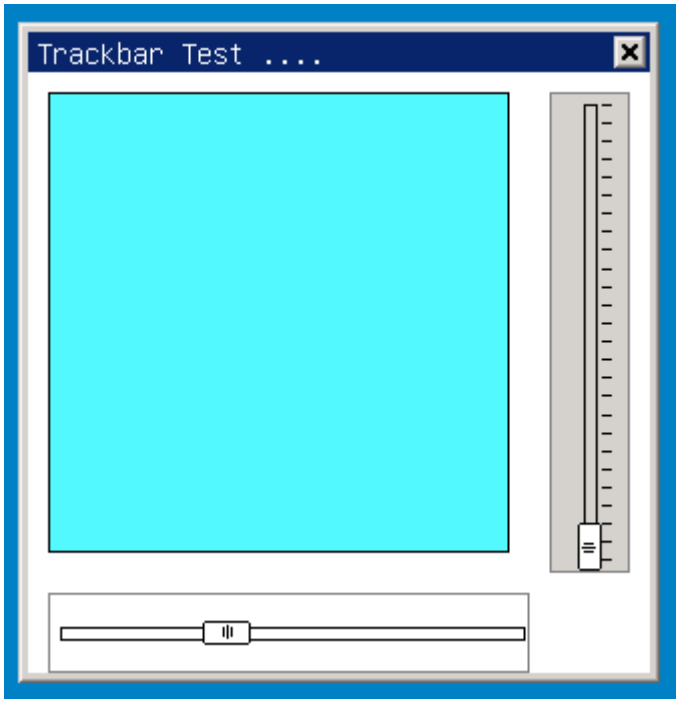


图 p2c8-1 trackbar 示例

清单 p2c8-1 trackbar.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#include <minigui/common.h>

#include <minigui/minigui.h>

#include <minigui/gdi.h>

#include <minigui/window.h>

#include <minigui/control.h>


#include <mgncs/mgncs.h>
```

```
#define ID_TRB1     101
#define ID_TRB2     102
#define ID_RECT           200

static BOOL mymain_onCreate(mWidget* self, DWORD add_data)
{
        return TRUE;
}

static void mymain_onClose(mWidget* self, int message)
{
        DestroyMainWindow(self->hwnd);
        PostQuitMessage(0);
}

static NCS_PROP_ENTRY trk_props [] = {
        {NCSP_TRKBAR_MINPOS, 0},
        {NCSP_TRKBAR_MAXPOS, 25},
        {NCSP_TRKBAR_CURPOS, 0},
        {NCSP_TRKBAR_LINESTEP, 5},
        {NCSP_TRKBAR_PAGESTEP, 5},
        {0, 0}
};
```

```
static void trackbar_notify(mTrackBar* self, int id, int code, DWORD add_data)
{
        mRectangle *rect = (mRectangle*)ncsGetChildObj(GetParent(self->hwnd), ID_RECT);

        if(rect)
        {
                DWORD fill_color = _c(rect)->getProperty(rect, NCSP_RECTANGLE_FILLCOLOR);
                int r = GetRValue(fill_color);
                int g = GetGValue(fill_color);
                int b = GetBValue(fill_color);
                int v = _c(self)->getProperty(self, NCSP_TRKBAR_CURPOS);
                switch(id)
                {
                        case ID_TRB1:
                        r = 10 * v;
                        break;
                        case ID_TRB2:
                        g = 10 * v;
                        break;
                }

                fill_color = MakeRGBA(r, g, b,255);
```

```
                const RECT rc = {10, 10, 230, 230};


                _c(rect)->setProperty(rect, NCSP_RECTANGLE_FILLCOLOR, fill_color);
                InvalidateRect(rect->hwnd, &rc, TRUE);
        }
}
static NCS_RDR_INFO track_rdr_info[] =
{
        {"flat", "flat", NULL},
        //{"skin", "skin", NULL},
        //{"classic", "classic", NULL},
        //{"fashion","fashion",NULL}
};



static NCS_EVENT_HANDLER trk1_handlers[] = {
        NCS_MAP_NOTIFY(NCSN_TRKBAR_CHANGED, trackbar_notify),
        {0, NULL}
};

static NCS_EVENT_HANDLER trk2_handlers[] = {
        NCS_MAP_NOTIFY(NCSN_TRKBAR_CHANGED, trackbar_notify),
        {0, NULL}
};
```

```
//Controls
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_RECTANGLE,
                ID_RECT,
                10, 10, 230, 230,
                WS_VISIBLE,
                WS_EX_NONE,
                "",
                NULL, //props,
                NULL,//btn2_rdr_info, //rdr_info
                NULL, //handlers,
                NULL, //controls
                0,
                0 //add data
        },

        {
                NCSCTRL_TRACKBAR,
                ID_TRB1,
                10, 260, 240, 40,
                WS_BORDER | WS_VISIBLE | NCSS_TRKBAR_NOTICK | NCSS_NOTIFY ,
```

```
        WS_EX_TRANSPARENT,
        "",
        trk_props, //props,
        track_rdr_info, //rdr_info
        trk1_handlers, //handlers,
        NULL, //controls
        0,
        0, //add data
        MakeRGBA(255,0,0,255)
    },
    {
        NCSCTRL_TRACKBAR,
        ID_TRB2,
        260, 10, 40, 240,
        WS_BORDER | WS_VISIBLE | NCSS_NOTIFY | NCSS_TRKBAR_VERTICAL,
        WS_EX_NONE,
        "",
        trk_props, //props,
        track_rdr_info, //rdr_info
        trk2_handlers, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
```

```
};



static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate },
        {MSG_CLOSE, mymain_onClose },
        {0, NULL }
};


//define the main window template
static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 320, 330,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Trackbar Test ....",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        MakeRGBA(255,255,255,255)
```

```
};


int MiniGUIMain(int argc, const char* argv[])
{
        if(argc>1)
        {
                track_rdr_info[0].glb_rdr = argv[1];
                track_rdr_info[0].ctl_rdr = argv[1];
        }



        ncsInitialize();
        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);


        _c(mydlg)->doModal(mydlg, TRUE);


        MainWindowThreadCleanup(mydlg->hwnd);
        return 0;
}
```

mScrollBar

控件窗口类

NCSCTRL_SCROLLBAR

控件英文名

Scrollbar

简介

可拖动滑块，常用于窗口内容显示调节。

示意图

mScrollBar 风格

继承自 mSlider 的风格

风格	miniStudio 属性名	说明
NCSS_SCRLBR_HORIZONTAL	--	创建水平的 scrollbar 控件（默认）
NCSS_SCRLBR_VERTICAL	--	创建竖直的 scrollbar 控件
NCSS_SCRLBR_ARROWS	HaveArrows	标记一个滚动条是否有箭头
NCSS_SCRLBR_LEFTDBLARROWS	DoubleArrows -> Left	标记一个滚动条是否有左双箭头 (和 NCSS_SCROLLBAR_RIGHTDBLARROWS 互斥)
NCSS_SCRLBR_RIGHTDBLARROWS	DoubleArrows -> Right	标记一个滚动条是否有右双箭头(和 NCSS_SCROLLBAR_LEFTDBLARROWS 互斥)
NCSS_SCRLBR_UPDBLARROWS	DoubleArrows -> Up	标记一个滚动条是否有上箭头(和 NCSS_SCROLLBAR_DOWNDBLAROWS 互斥)
NCSS_SCRLBR_DOWNDBLAROWS	DoubleArrows -> Down	标记一个滚动条是否有下箭头(和 NCSS_SCROLLBAR_DUPDBLAROWS 互斥)

mScrollBar 属性

继承自 mSlider 的属性

属性名	miniStudio 属性名	类型	RW	说明	
NCSP_SCRLBR_MAXPOS	MaxPos	int	RW	设置 scrollbar 的滑动范围最大值	最值的设定一般在初始化的时候做
NCSP_SCRLBR_MINPOS	MinPos	int	RW	设置 scrollbar 的滑动范围最小值	
NCSP_SCRLBR_CURPOS	CurPos	int	RW	设置滑动块的当前位置	
NCSP_SCRLBR_LINESTEP	LineStep	int	RW	设置步长（方向键）	

NCSP_SCRLBR_PAGESTEP	PageStep	int	RW	设置步长（pageUp/pageDown）

mScrollBar 事件

继承自 mSlider 的事件

事件 ID	参数	说明
NCSN_SCRLBR_CHANGED	--	滑块位置变化
NCSN_SCRLBR_REACHMAX	--	滑块到达最大值
NCSN_SCRLBR_REACHMIN	--	滑块到达最小值

mScrollBar 方法

继承自 mSlider 的方法

mScrollBar 渲染器

mScrollBar Classic 渲染器

非客户区的绘制请查阅 mWidget 的渲染器

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域
NCS_BGC_3DBODY	控件背景色	BgColor	DWORD(ARGB)		
NCS_BGC_DISABLED_ITEM	控件无效时滑块和箭头按钮颜色	ColorBgDisable	DWORD(ARGB)		
NCS_FGC_3DBODY	按钮上箭头颜色	ArrowColor	DWORD(ARGB)		
NCS_FGC_DISABLED_ITEM	按钮无效时箭头颜色	ArrowColorDisable	DWORD(ARGB)		

mScrollBar Skin 渲染器

参阅 附录 B : Skin 渲染器使用的图片资源规范

mScrollBar Fashion 渲染器

非客户区绘制请参阅 mWidget 的 Fashion 渲染器绘制

属性 ID	含义	miniStudio 属性名	值类型	生效区域示意图	值域
NCS_BGC_3DBODY	滑块和按钮颜色	ColorBg3DBody	DWORD(ARGB)		
NCS_BGC_DISABLED_ITEM	控件无效时滑块和箭头按钮颜色	ColorBgDisable	DWORD(ARGB)	同 Classic 渲染器	
NCS_BGC_WINDOW	滑轨颜色	ColorBgWindow	DWORD(ARGB)		
NCS_FGC_DISABLED_ITEM	按钮无效时箭头颜色	ArrowColorDisable	DWORD(ARGB)	同 Classic 渲染器	
NCS_METRICS_3DBODY_ROUNDX	滑块圆角 X 半径	RoundX	int		0 ~ 窗口宽度的 1/2
NCS_METRICS_3DBODY_ROUNDY	滑块圆角 Y 半径	RoundY	int		0 ~窗口高度的 1/2
NCS_MODE_BGC	渐变填充方式	GradientMode	int		GradientMode
					

mScrollBar Flat 渲染器

非客户区绘制请参阅 mWidget 的 Flat 渲染器绘制

属性 ID	含义	miniStudio 属性	值类型	生效区域示意图	值

		名			域
NCS_FGC_3DBODY	滑块和按钮边颜色	ColorFg3DBody	DWORD(ARGB)		
NCS_BGC_3DBODY	控件背景色	ColorBg3DBody	DWORD(ARGB)		

mScrollbar 示例

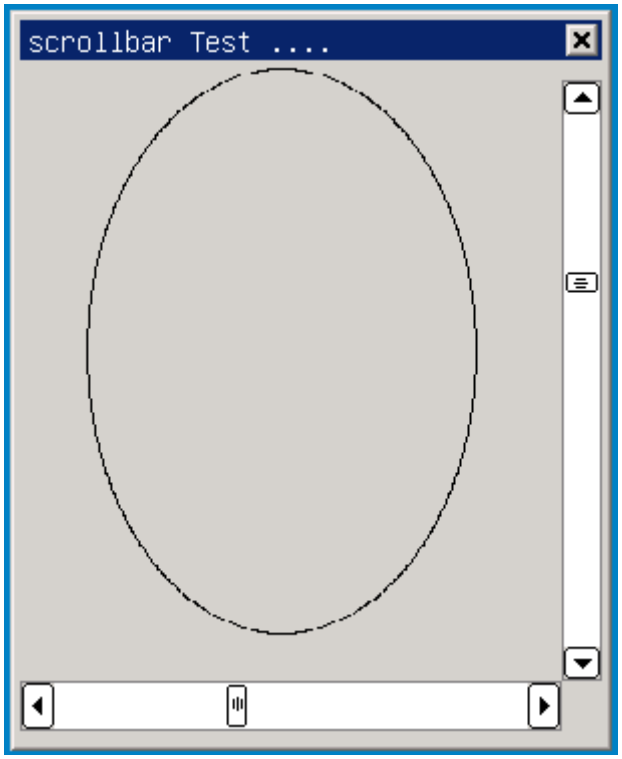


图 p2c8-2 scrollbar 示例

清单 p2c8-2 scrollbar.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>



#include <minigui/common.h>

#include <minigui/minigui.h>
```

```
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/control.h>

#include <mgncs/mgncs.h>

#define ID_SB1  100
#define ID_SB2  101

static RECT rcCircle = {0, 0, 400, 400};
static int radius_x = 0;
static int radius_y = 0;

static BOOL mymain_onCreate(mWidget* self, DWORD add_data)
{
        return TRUE;
}
static void mymain_onClose(mWidget* self, int message)
{
        DestroyMainWindow(self->hwnd);
        PostQuitMessage(0);
}

static void mymain_onPaint(mWidget* self, HDC hdc, const PCLIPRGN clip_rgn)
```

```
{
        ClipRectIntersect(hdc, &rcCircle);
        Ellipse(hdc, 130, 200, radius_x, radius_y);
}

//define the progress properites
static NCS_PROP_ENTRY scrollbar_props [] = {
        {NCSP_SCRLBR_MAXPOS, 255},
        {NCSP_SCRLBR_MINPOS, 0 },
        {NCSP_SCRLBR_LINESTEP, 5},
        {NCSP_SCRLBR_CURPOS, 10 },
        { 0, 0 }
};


static void scrollbar_notify(mScrollBar* self, int id, int code, DWORD add_data)
{
        HWND hWnd = GetParent(self->hwnd);

        if(id == ID_SB1)
        radius_x = _c(self)->getProperty(self, NCSP_SCRLBR_CURPOS);

        if(id == ID_SB2)
        radius_y = _c(self)->getProperty(self, NCSP_SCRLBR_CURPOS);
```

```
        InvalidateRect(hWnd, &rcCircle, TRUE);
}

static NCS_EVENT_HANDLER scrollbar_notifies[] = {
        NCS_MAP_NOTIFY(NCSN_SCRLBR_CHANGED, scrollbar_notify),
        {0, NULL}
};

static NCS_RDR_INFO sb_rdr_info[] =
{
        {"flat", "flat", NULL},
        //{"skin", "skin", NULL},
        //{"fashion","fashion", NULL}
};

//Controls
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_SCROLLBAR,
                ID_SB1,
                0, 440, 270, 25,
                WS_BORDER | NCSS_NOTIFY | WS_VISIBLE | NCSS_SCRLBR_ARROWS,
                WS_EX_NONE,
```

```
            "",
            scrollbar_props, //props,
            sb_rdr_info, //rdr_info
            scrollbar_notifies,
            NULL, //controls
            0,
            0 //add data
        },
        {
            NCSCTRL_SCROLLBAR,
            ID_SB2,
            270, 10, 20, 430,
            WS_BORDER | NCSS_NOTIFY | WS_VISIBLE \
            | NCSS_SCRLBR_ARROWS | NCSS_SCRLBR_VERTICAL,
            WS_EX_NONE,
            "",
            scrollbar_props, //props,
            sb_rdr_info, //rdr_info
            scrollbar_notifies,
            NULL, //controls
            0,
            0 //add data
        },
};
```

```
static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE, mymain_onCreate},
        {MSG_CLOSE, mymain_onClose},
        {MSG_PAINT, mymain_onPaint},
        {0, NULL}
};

//define the main window template
static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 300, 500,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "scrollbar Test ....",
        NULL,
        NULL,
        mymain_handlers,
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
```

```
};


int MiniGUIMain(int argc, const char* argv[])
{
        if(argc>1)
        {
                sb_rdr_info[0].glb_rdr = argv[1];
                sb_rdr_info[0].ctl_rdr = argv[1];
        }

        ncsInitialize();
        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);

        _c(mydlg)->doModal(mydlg, TRUE);

        MainWindowThreadCleanup(mydlg->hwnd);
        return 0;
}
```

第二部分第九章 旋钮类控件

旋钮类控件简介

旋钮类控件通常由一组箭头按钮（上下或者左右），通常表示一个范围，通过箭头控制，可以一步一步的改变值。

mGNCS 提供两种按钮，mSpinner 和 mSpinbox，他们的继承关系如下

- mWidget
  - mSpinner
    - mSpinbox

mSpinner

控件名称

NCSCTRL_SPINNER

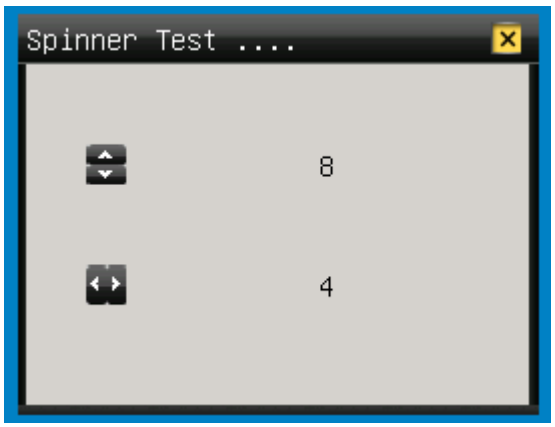
英文名

Spinner

简要介绍

由两个箭头按钮（上下或者左右）组成的能够控制在一个范围内变化的值的控件

示意图



mSpinner 风格

继承自 mWidget 的风格

风格 ID	miniStudio 名	说明

NCSS_SPNR_VERTICAL	Direction->Vert	垂直风格，上下箭头，当点击上下箭头的时候，会向目标窗口发送上下方向键的键盘消息
NCSS_SPNR_HORIZONTAL	Direction->Horz	水平风格，左右箭头,当点击左右箭头的时候，会向目标窗口发送左右方向键的键盘消息
NCSS_SPNR_AUTOLOOP	AutoLoop	当到达最大或者最小值时，自动跳转到最小或者最大值处

mSpinner 属性

继承自 mWidget 的属性

属性 ID	miniStudio 名	类型	权限	说明
NCSP_SPNR_MAXPOS	MaxPos	int	RW	变化范围的最大值
NCSP_SPNR_MINPOS	MinPos	int	RW	变化范围的最小值
NCSP_SPNR_CURPOS	CurPos	int	RW	当前值
NCSP_SPNR_LINESTEP	LineStep	int	RW	步进步长
NCSP_SPNR_TARGET	-	HWND	RW	目标窗口句柄

mSpinner 事件

继承自 mWidget 的事件

事件 ID	参数	说明
NCSN_SPNR_CHANGED	int 当前属性值	当前值变化
NCSN_SPNR_REACHMAX	int 当前属性值	当前值到达最大值
NCSN_SPNR_REACHMIN	int 当前属性值	当前值到达最小值

mSpinner 方法

继承自 mWidget 的方法

该类没有新增方法

mSpinner 渲染器

mSpinner classic 渲染器

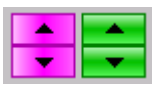
非客户区查看 mWidget 的 classic 渲染器

属性名	说明	miniStudio 属性名	类型	示意图
NCS_BGC_3DBODY	背景颜色	ColorBg3DBody	DWORD(ARGB)	
NCS_FGC_3DBODY	前景箭头的颜色	ColorFg3DBody	DWORD(ARGB)	
NCS_BGC_DISABLED_ITEM	无效时的背景颜色	ColorBgDisable	DWORD(ARGB)	
NCS_FGC_DISABLED_ITEM	无效时的箭头颜色	ColorFgDisable	DWORD(ARGB)	

mSpinner Skin 渲染器

参阅 附录 B : Skin 渲染器使用的图片资源规范

mSpinner Fashion 渲染器

属性名	说明	miniStudio 属性名	类型	示意图
NCS_FGC_3DBODY	按钮前景色	ColorFg3DBody	DWORD(ARGB)	同 Classic 渲染器
NCS_FGC_DISABLED_ITEM	窗口无效时按钮前景色	ColorFgDisable	DWORD(ARGB)	同 Classic 渲染器
NCS_BGC_3DBODY	背景颜色	ColorBg3DBody	DWORD(ARGB)	同 Classic 渲染器
NCS_BGC_DISABLED_ITEM	窗口无效时文本背景色	ColorBgDisable	DWORD(ARGB)	同 Classic 渲染器
NCS_MODE_BGC	渐变填充方式	GradientMode	GradientMode	

mSpinner Flat 渲染器

属性名	说明	miniStudio 属性名	类型	示意图
NCS_FGC_3DBODY	按钮前景色	ColorFg3DBody	DWORD(ARGB)	
NCS_BGC_3DBODY	背景颜色	ColorBg3DBody	DWORD(ARGB)	

mSpinner 示例

下面的示例（截图见 mSpinner 的示意图）演示了 Spinner 和一个 Static 控件关联。当 Spinner 的 pos 改变后，Static 会响应的变化。

- 主要涉及的属性有
  - NCSP_SPN_MAXPOS
  - NCSP_SPN_MINPOS
  - NCSP_SPN_CURPOS
- 主要涉及的事件:
  - NCSN_SPN_CHANGED

为了方便我们主要数据绑定的方法来实现

- 示例 spinner.c
- 在窗口的模板定义文件中设置 spinner 的属性

```
//Propties for
static NCS_PROP_ENTRY spinbox_props [] = {
        {NCSP_SPNR_MINPOS, MINVALUE},
        {NCSP_SPNR_MAXPOS, MAXVALUE},
        {NCSP_SPNR_CURPOS, CURVALUE},
        {NCSP_SPNR_LINESTEP, 1},
        {0, 0}
};
```

- 在主窗口的 MSG_CREATE 消息中，建立和 static 的连接
  - 获取窗口对象

```
mSpinner * spn1, *spn2;
mStatic  * show1, * show2;
spn1 = (mSpinner*)_c(self)->getChild(self, ID_SPINNER1);
spn2 = (mSpinner*)_c(self)->getChild(self, ID_SPINNER2);
show1 = (mStatic*)_c(self)->getChild(self, ID_SHOWSPINNER1);
show2 = (mStatic*)_c(self)->getChild(self, ID_SHOWSPINNER2);
```

  - 连接窗口属性

```
ncsConnectBindProps(
NCS_CMPT_PROP(spn1, NCSN_SPNR_CHANGED, NCSP_SPNR_CURPOS, NCS_BT_INT, NCS_PROP_FLAG_READ),
NCS_CMPT_PROP(show1, 0, NCSP_WIDGET_TEXT, NCS_BT_STR, NCS_PROP_FLAG_WRITE),
NCS_BPT_SIGNALE);

ncsConnectBindProps(
NCS_CMPT_PROP(spn2, NCSN_SPNR_CHANGED, NCSP_SPNR_CURPOS, NCS_BT_INT, NCS_PROP_FLAG_READ),
NCS_CMPT_PROP(show2, 0, NCSP_WIDGET_TEXT, NCS_BT_STR, NCS_PROP_FLAG_WRITE),
NCS_BPT_SIGNALE);
```

  - 更新当前的信息到 Static 中

```
ncsRaiseComponentBindProps((mComponent*)spn1, NCSN_SPNR_CHANGED);
```

```
ncsRaiseComponentBindProps((mComponent*)spn2, NCSN_SPNR_CHANGED);
```

mSpinbox

控件窗口类名

NCSCTRL_SPINBOX

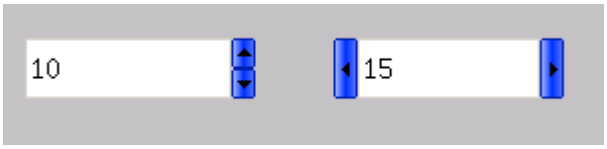
控件英文名

Spinbox

简要介绍

把箭头按钮和编辑框组合起来，通过箭头按钮控制编辑框内的内容

示意图



mSpinbox 风格

继承自 mSpinner 的风格

风格 ID	miniStudio 名	说明
NCSS_SPNBOX_NUMBER	ContentType ->Number	显示数值，可以通过相应的属性指定显示格式
NCSS_SPNBOX_STRING	ContentType ->String	显示字符串
NCSS_SPNBOX_SORT	Sort	字符串自动排序
NCSS_SPNBOX_EDITNOBORDER	EditBorder	编辑框没有边框
NCSS_SPNBOX_EDITBASELINE	EditBaseLine	编辑框显示带下划线字符
NCSS_SPNBOX_READONLY	ReadOnly	编辑框显示的内容只读
NCSS_SPNBOX_UPPERCASE	Case->Upper	编辑框显示内容全部转成大写字母显示
NCSS_SPNBOX_LOWERCASE	Case->Lower	编辑框显示内容全部转成小写字母显示
NCSS_SPNBOX_AUTOFOCUS	AutoFocus	控件获取焦点自动转给编辑框

mSpinbox 属性

继承自 mSpinner 的属性

mSpinbox 事件

继承自 mSpinner 的事件

mSpinbox 方法

继承自 mSpinner 的方法

以下的函数只有在 spinbox 包含 NCSS_SPNBOX_STRING 风格时有效

- addItem 增加一个新的项

```
BOOL (*addItem)(clsName *self, char *item);
```

  -
    - 增加一个字符串到 spinbox 中
    - 参数：
      - item : 添加的字符串
    - 返回值： 成功 - TRUE; 失败 - FALSE;

- remove Item

```
BOOL (*removeItem)(clsName *self, int index);
```

  -
    - 删除一个 item
    - 参数：
      - index 要删除 item 的索引
    - 返回值：成功 - TRUE; 失败 - FALSE;

- setItem

```
BOOL (*setItem)(clsName *self, int index, char *item);
```

  -
    - 设置一个 item 的字符串内容
    - 参数：
      - index 要设置的索引
      - item 新的 item 内容
    - 返回值：成功 - TRUE; 失败 - FALSE;

- getItem

```
char* (*getItem)(clsName *self, int index);
```

-
  - 获取 item 的字符串内容
  - 参数：
    - index item 的索引
  - 返回值：NULL - 失败；有效的字符串指针

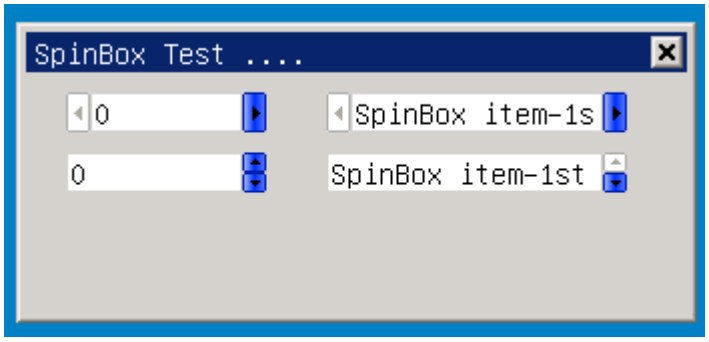
mSpinbox 渲染器

mSpinbox 渲染的效果由 mSpinner 和 mSlEdit 的效果组成，该类本身没有新增的属性

mSpinbox 示例

该示例分别说明了水平风格下和垂直风格下，数字型和列表型两种形态下的 spinbox。

示意图：



- 源代码 spinbox.c
- 主要代码如下:

```
#define ID_SPINBOX1      101
#define ID_SPINBOX2      102
#define ID_SPINBOX3      103
#define ID_SPINBOX4      104

static char * item [] =
```

```
{
        "SpinBox item-1st",
        "SpinBox item-2nd",
        "SpinBox item-3rd",
        "SpinBox item-4th"
};

static BOOL mymain_onCreate(mWidget* self, DWORD add_data)
{
        int i;
        mSpinBox *spinner3, *spinner4;

        spinner3 = (mSpinBox *)_c(self)->getChild(self, ID_SPINBOX3);
        spinner4 = (mSpinBox *)_c(self)->getChild(self, ID_SPINBOX4);

        for (i = 0; i < sizeof(item)/sizeof(char*); i++)     {         _c(spinner3)->addItem
(spinner3, item[i]);

                _c(spinner4)->addItem (spinner4, item[i]);

        }

        return TRUE;
}
```

```
//Propties for
static NCS_PROP_ENTRY spinner_props [] = {

        {NCSP_SPNBOX_MAXPOS, 12},
        {NCSP_SPNBOX_MINPOS, 0},
        {NCSP_SPNBOX_CURPOS, 0},
        {NCSP_SPNBOX_LINESTEP, 1},
        {0, 0}
};
```

第二部分第十章 进度条控件

进度条控件简介

进度条控件（ProgressBar）是 GUI 系统中不可或缺的的重要控件之一，可以直观的表示出某项的进度，在进行文件复制、软件安装、文件传输时经常用到。mGNCS 中的进度条控件基于 MiniGUI 3.0 中内置的进度条控件进行了增强和重构，增加了渲染器的设置，使其更加易用，更加绚丽。

- 进度条类层次关系
  - mWidget
    - mProgressBar

mProgressBar

控件窗口类

NCSCTRL_PROGRESSBAR

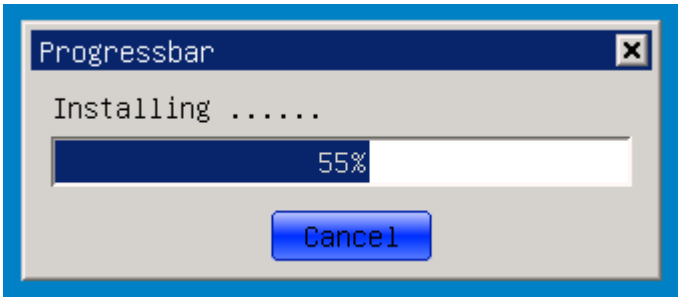
控件英文名

ProgressBar

简介

用于表示进度，在进行文件复制、软件安装、文件传输时经常用到。

示意图



mProgressBar 风格

继承自 mWidget 的风格

风格 ID	miniStudio 属性名	说明	
NCSS_PRGBAR_HORIZONTAL	--	miniStudio 中将这两种风格分成了两个控件来使用 分别对应为	Horz ProgressBar
NCSS_PRGBAR_VERTICAL	--		Vert ProgressBar

NCSS_PRGBAR_BLOCKS	BlockChunk ->TRUE	块状进度	
NCSS_PRGBAR_SMOOTH	BlockChunk ->FALSE	平滑进度，与上面的块状进度相对	
NCSS_PRGBAR_SHOWPERCENT	ShowPercent	显示当前进度值（百分比)	

mProgressBar 属性

继承自 mWidget 的属性

属性名	miniStudio 属性名	类型	RW	说明
NCSP_PROG_MAXPOS	MaxPos	int	RW	进度范围最大值
NCSP_PROG_MINPOS	MinPos	int	RW	进度范围最小值
NCSP_PROG_CURPOS	CurPos	int	RW	当前进度值
NCSP_PROG_LINESTEP	LineStep	int	RW	步长

mProgressBar 事件

继承自 mWidget 的事件

该控件类没有新增事件

mProgressBar 方法

继承自 mWidget 的方法

increase

```
int increase (mProgressBar *self, int delta);
```

- 参数：
  - self ： 控件对象指针
  - delta ： 增加幅度
- 说明 ： 用户可以使用改函数控制进度值的增加，delta 用来指定增加的量，一般用来控制非匀速的增长效果
- 示例 ：

```
_c(pb)->increase (pb, 5);     //进度值增加 5
```

stepIncrease

```
int stepIncrease (mProgressBar *_this);
```

- 参数：
  - self ：控件对象指针
- 说明：该函数可以步进增加进度值，每调用一次增加一个步长值，可以实现匀速的增长效果，步长值的设定通过相应的属性来完成
- 示例 ：

```
//设置 ProgressBar 的步进长度

_c(pb)->setProperty(pb, NCSP_PROG_LINESTEP, 5);

......

_c(pb)->stepIncrease (pb);
```

mProgressBar 渲染器

classic 渲染器

继承自 mWidget 的 classic 渲染器

属性名	miniStudio 属性名	类型	示意图	说明
NCS_BGC_HILIGHT_ITEM	ChunkColor	DWORD（ARGB）		进度条的 chunk 部分的颜色，本渲染器使用高亮 item 的颜色绘制

fashion 渲染器

继承自 mWidget 的 fashion 渲染器

属性名	miniStudio 属性名	类型	示意图	说明
NCS_BGC_PRGBAR_CHUNK	ChunkColor	DWORD（ARGB）		进度条的 chunk 部分的基础颜色，渲染器会根据这

	个颜色进行减淡、加深，来绘制渐变效果的进度条

flat 渲染器

继承自 mWidget 的 flat 渲染器

属性名	miniStudio 属性名	类型	示意图	说明
NCS_FGC_WINDOW	ChunkColor	DWORD（ARGB）		进度条的 chunk 部分的颜色，这里使用窗口前景色

skin 渲染器

参阅 Skin 渲染器使用的图片资源规范(附录 B)中 ProgressBar 图片规范

mProgressBar 编程示例

- ProgressBar 示例代码 ： progressbar.c
- ProgressBar 模板

```
static NCS_WND_TEMPLATE _ctrl_templ[] =
{
        {
                NCSCTRL_PROGRESSBAR,
                ID_PROG,
                10, 33, 290, 25,
                WS_BORDER | WS_VISIBLE | NCSS_PRGBAR_SHOWPERCENT,
                WS_EX_NONE,
                "",
                progress_props,
```

```
                NULL,
                NULL, NULL, 0, 0
        },
        {
                NCSCTRL_BUTTON,
                ID_BTN,
                120, 70, 80, 25,
                WS_VISIBLE | NCSS_NOTIFY,
                WS_EX_NONE,
                "Cancel",
                NULL,
                btn_rdr_info,
                btn_handlers, NULL, 0, 0
        },
};
```

- 初始化 ProgressBar 属性

```
static NCS_PROP_ENTRY progress_props[] =
{
        {NCSP_PROG_MAXPOS, 100},
        {NCSP_PROG_MINPOS, 0  },
        {NCSP_PROG_LINESTEP, 1},
        {NCSP_PROG_CURPOS, 0  },
        { 0, 0 }
```

```
};
```

- 在 Timer 消息中，改变 ProgressBar 的属性值

```
static int pb_pos = 0;

mProgressBar *pb = (mProgressBar*)ncsGetChildObj (_this->hwnd, ID_PROG);
if (pb)
{
        pb_pos++;
        _c(pb)->setProperty(pb, NCSP_PROG_CURPOS, pb_pos);

        if (pb_pos == _c(pb)->getProperty(pb, NCSP_PROG_MAXPOS))
        {
                DestroyMainWindow (_this->hwnd);
                PostQuitMessage (_this->hwnd);
        }
}
```

第二部分第十一章 属性表控件类

属性表控件类简介

属性表控件类由一个个的独立的属性页组成，每个属性页有一个凸舌，我们可以单击凸舌，在不同的属性页之间切换，这里的属性页就是可以容纳其他控件的页控件（Page 控件）。我们通常使用类似建立对话框的方法，即定义对话框模板的方法向属性表中添加属性页。

属性表控件类最常见的用途就是将不同类别的交互内容分门别类的放在同一对话框中，通过类别切换显示不同内容。这一方面节省了对话框空间，另一方面也使得交互界面更加容易使用。

窗口及其派生类的类继承关系如下：

- mWidget
  - mPropSheet

mPropSheet

控件名称

NCSCTRL_PROPSHEET

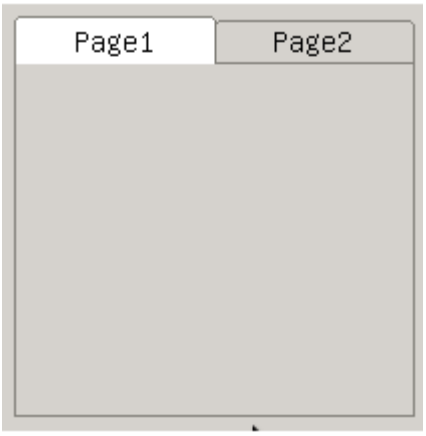
英文名

PropSheet

简要介绍

由多个独立的属性页组成，每个属性页有一个凸舌，通过凸舌进行属性页间的切换。

示意图



mPropSheet 风格

继承自 mWidget 的风格

风格名	miniStudio 属性名	说明
NCSS_PRPSHT_SIMPLE	style->Simple	控制属性页凸舌宽度的风格：所有的属性页凸舌具有相同的宽度。
NCSS_PRPSHT_COMPACTTAB	style->Compact	控制属性页凸舌宽度的风格：属性页凸舌的宽度取决于属性页标题文本的长度。
NCSS_PRPSHT_SCROLLABLE	style->Scrollable	控制属性页凸舌宽度的风格：属性页凸舌的宽度取决于属性页标题文本的长度，当属性页凸舌的数目过多时，将自动出现左右箭头用来调节当前可见的属性页凸舌。
NCSS_PRPSHT_TOP	TabPos ->Top	控制属性页凸舌在属性表中显示方向的风格：属性页凸舌显示在属性表的上方。
NCSS_PRPSHT_BOTTOM	TabPos ->Bottom	控制属性页凸舌在属性表中显示方向的风格：属性页凸舌显示在属性表的下方。

mPropSheet 属性

继承自 mWidget 的属性

属性 ID	miniStudio 名	类型	权限	说明
NCSP_PRPSHT_MINTABWIDTH	TabMinWidth	int	RW	凸舌最小宽度
NCSP_PRPSHT_TABMARGIN	TabMargin	int	RW	凸舌边界值，通常情况下，该值加上文字所占宽度为凸舌宽度
NCSP_PRPSHT_ACTIVEPAGE	-	mPage*	RW	当前活动页指针
NCSP_PRPSHT_ACTIVEPAGEIDX	ActivePageIndex	int	RW	当前活动页索引
NCSP_PRPSHT_FIRSTVIEWPAGE	-	mPage*	RO	当前第一个可见页指针
NCSP_PRPSHT_FIRSTVIEWPAGEIDX	-	int	RO	当前第一个可见页索引
NCSP_PRPSHT_PAGECOUNT	-	int	RO	当前属性页数

mPropSheet 事件

继承自 mWidget 的事件

事件 ID	参数	说明
NCSN_PRPSHT_ACTIVECHANGED	--	活动属性页已改变

mPropSheet 方法

继承自 mWidget 的方法

添加属性页

在创建了属性表控件后，可以通过 **addPage** 方法向属性表中添加属性页。该方法的 *dlgTemplate* 用来传递对话框模板， *handlers* 用来传递属性页的事件回调处理函数。函数原型如下：

```
mPage* addPage(mPropSheet *self, \

PDLGTEMPLATE dlgTemplate,  \

const NCS_EVENT_HANDLER* handlers);
```

如示例程序利用下面代码向属性表控件中添加了多个属性页：

```


PageSysInfo.controls = CtrlSysInfo;

PageSysInfo.caption = "Version Info";

PageSysInfo.dwAddData = PAGE_VERSION;

_c(propsheet)->addPage(propsheet, &PageSysInfo, mypage_handlers);



PageSysInfo.caption = "CPU Info";

PageSysInfo.dwAddData = PAGE_CPU;

_c(propsheet)->addPage(propsheet, &PageSysInfo, mypage_handlers);

```

```
PageSysInfo.caption = "MEM Info";
PageSysInfo.dwAddData = PAGE_MEMINFO;
_c(propsheet)->addPage(propsheet, &PageSysInfo, mypage_handlers);

PageSysInfo.caption = "Partition Info";
PageSysInfo.dwAddData = PAGE_PARTITION;
_c(propsheet)->addPage(propsheet, &PageSysInfo, mypage_handlers);

PageSysInfo.caption = "MiniGUI Info";
PageSysInfo.dwAddData = PAGE_MINIGUI;
_c(propsheet)->addPage(propsheet, &PageSysInfo, mypage_handlers);
```

其中各属性页的事件处理为：

```
static void mypage_onInitPage(mWidget* self, DWORD add_data)
{
        get_systeminfo ((mPage*)self);
}

static int mypage_onShowPage(mWidget* self, HWND hwnd, int show_cmd)
{
        return 1;
}

static int mypage_onSheetCmd(mWidget* self, DWORD wParam, DWORD lParam)
```

```
{
        if (wParam == IDC_REFRESH) {
                get_systeminfo ((mPage*)self);
        }

        return 0;
}

static NCS_EVENT_HANDLER mypage_handlers[] = {
        {MSG_INITPAGE, mypage_onInitPage},
        {MSG_SHOWPAGE, mypage_onShowPage},
        {MSG_SHEETCMD, mypage_onSheetCmd},
        {0 , NULL }
};
```

删除属性页

要删除某个属性页，只需调用属性表控件的 **removePage** 或 **removePageByIndex** 方法，需要注意的是在删除了一个属性页后会有可能改变其他属性页的索引值。其中 **removePage** 是通过属性页的类指针来删除指定页， **removePageByIndex** 通过属性页索引删除指定页。

函数原型如下：

```
BOOL removePageByIndex(mPropSheet *self, int pageIndex);

BOOL removePage(mPropSheet *self, mPage* page);
```

如要删除属性表中的第一个属性页可执行如下操作：

```
_c(propsheet)->removePageByIndex(propsheet, 0);
```

索引属性页

要获取指定索引的索引页类指针，需调用属性表控件的 **getPageByIndex** 方法；而要获取某个指定属性页的索引则只需调用属性表控件的 **getPageIndex** 方法。函数原型如下：

```
int getPageIndex(mPropSheet *self, mPage* page);

mPage* getPageByIndex(mPropSheet *self, int pageIndex);
```

如获取指定属性页的索引：

```
mPage *page;

... ...

_c(propsheet)->getPageIndex(propsheet, page);
```

或者是获取第一个属性页的类指针，再通过 page 控件的方法来操作属性页：

```
mPage *page = _c(propsheet)->getPageByIndex(propsheet, 0);

HWND hPanel = _c(page)->getPanel(page);

... ...
```

遍历属性页

在属性表控件中可以通过 **getNextPage** 和 **getPrevPage** 方法实现对所有属性页的遍历查找功能。其中 **getNextPage** 用于从指定属性页向后遍历属性页； **getPrevPage** 用于从指定属性页向前遍历属性页。

```
mPage* getNextPage(mPropSheet *self, mPage* page);

mPage* getPrevPage(mPropSheet *self, mPage* page);
```

如：

```
mPage *page = _c(propsheet)->getNextPage(propsheet, NULL);



while (page) {

        ... ...

        page = _c(propsheet)->getNextPage(propsheet, page);

}
```

广播消息

属性表控件可以通过 **broadCastMsg** 方法向所有的属性页广播消息。消息内容通过 *param1* 和 *param2* 来传送。当属性表控件接收到任意一个属性页对广播消息处理后返回的非零值后，属性表的返回值将为中断消息广播的属性页索引值加 1。通过此方法，可以实现无效输入处理等功能操作。

```
int broadCastMsg(mPropSheet *self, DWORD param1, DWORD param2);
```

mPropSheet 渲染器

继承自 mWidget 渲染器

mPropSheet 实例

本实例为用户演示了如何使用 propsheet 显示本机的一些系统信息，比如 CPU 类型、内存大小等等。

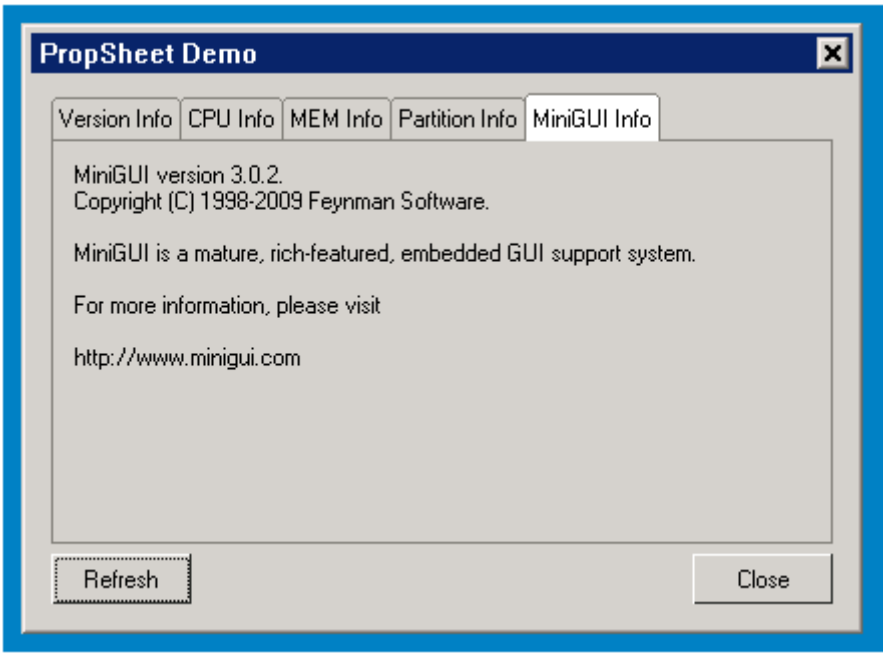


图 p2c11-1 propsheet 程序的输出

清单 p2c11-1 propsheet.c

```
/*

** $Id: propsheet.c 589 2009-10-09 06:19:44Z xwyan $

**

** Listing P2C11.1
```

```
**
** propsheet.c: Sample program for mGNCS Programming Guide
**      The demo application for PropSheet.
**
** Copyright (C) 2009 Feynman Software.
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// START_OF_INCS
#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/control.h>

#include <mgncs/mgncs.h>
// END_OF_INCS

#define PAGE_VERSION      1
#define PAGE_CPU          2
#define PAGE_MEMINFO      3
#define PAGE_PARTITION    4
```

```
#define PAGE_MINIGUI    5


#define BUF_LEN         10240
#define IDC_PROPSHEET     100
#define IDC_SYSINFO     101
#define IDC_REFRESH     102


static size_t read_sysinfo (const char* file, char* buff, size_t buf_len)
{
        size_t size;
        FILE   *fp = fopen (file, "r");

        if (fp == NULL) return 0;

        size = fread (buff, 1, buf_len, fp);

        fclose (fp);
        return size;
}


static void get_systeminfo (mPage* page)
{
        int     type;
        HWND    hwnd;
```

```
    char    buff [BUF_LEN + 1];
    size_t  size = 0;

    type = (int)GetWindowAdditionalData (page->hwnd);
    hwnd = GetDlgItem(_c(page)->getPanel(page), IDC_SYSINFO);
    buff [BUF_LEN] = 0;

    switch (type) {
        case PAGE_VERSION:
        size = read_sysinfo ("/proc/version", buff, BUF_LEN);
        buff [size] = 0;
        break;

        case PAGE_CPU:
        size = read_sysinfo ("/proc/cpuinfo", buff, BUF_LEN);
        buff [size] = 0;
        break;

        case PAGE_MEMINFO:
        size = read_sysinfo ("/proc/meminfo", buff, BUF_LEN);
        buff [size] = 0;
        break;

        case PAGE_PARTITION:
```

```
                size = read_sysinfo ("/proc/partitions", buff, BUF_LEN);
                buff [size] = 0;
                break;

                case PAGE_MINIGUI:
                size = snprintf (buff, BUF_LEN,
                "MiniGUI version %d.%d.%d.\n"
                "Copyright (C) 1998-2009 Feynman Software.\n\n"
                "MiniGUI is a mature, rich-featured, embedded "
                "GUI support system.\n\n"
                "For more information, please visit\n\n"
                "http://www.minigui.com\n",
                MINIGUI_MAJOR_VERSION, MINIGUI_MINOR_VERSION, MINIGUI_MICRO_VERSION);
                break;
        }

        if (size) {
                SetWindowText (hwnd, buff);
        }
        GetWindowText(hwnd, buff, BUF_LEN+1);
}

// START_OF_PAGEHANDLERS
static void mypage_onInitPage(mWidget* self, DWORD add_data)
```

```
{
        get_systeminfo ((mPage*)self);
}

static int mypage_onShowPage(mWidget* self, HWND hwnd, int show_cmd)
{
        return 1;
}

static int mypage_onSheetCmd(mWidget* self, DWORD wParam, DWORD lParam)
{
        if (wParam == IDC_REFRESH) {
                get_systeminfo ((mPage*)self);
        }

        return 0;
}

static NCS_EVENT_HANDLER mypage_handlers[] = {
        {MSG_INITPAGE, mypage_onInitPage},
        {MSG_SHOWPAGE, mypage_onShowPage},
        {MSG_SHEETCMD, mypage_onSheetCmd},
        {0 , NULL }
};
```

```
// END_OF_PAGEHANDLERS


static void btn_notify(mWidget *self, int id, int nc, DWORD add_data)
{
        mPropSheet *obj =
        (mPropSheet *)ncsGetChildObj(GetParent(self->hwnd), IDC_PROPSHEET);

        if (obj) {
                _c(obj)->broadCastMsg(obj, IDC_REFRESH, 0);
        }
}

static NCS_EVENT_HANDLER btn_handlers [] = {
        NCS_MAP_NOTIFY(NCSN_BUTTON_PUSHED, btn_notify),
        {0, NULL}
};

static NCS_RDR_INFO btn_rdr_info[] =
{
        {"classic","classic", NULL}
};

static NCS_WND_TEMPLATE _ctrl_tmpl[] = {
        {
```

```
            NCSCTRL_BUTTON,
            IDC_REFRESH,
            10, 240, 70, 25,
            WS_VISIBLE | WS_TABSTOP,
            WS_EX_NONE,
            "Refresh",
            NULL,
            btn_rdr_info,
            btn_handlers,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_BUTTON,
            IDCANCEL,
            330, 240, 70, 25,
            WS_VISIBLE | WS_TABSTOP,
            WS_EX_NONE,
            "Close",
            NULL,
            NULL,
            NULL,
            NULL,
```

```
                0,
                0
        },
};

static DLGTEMPLATE PageSysInfo =
{
        WS_BORDER | WS_CAPTION,
        WS_EX_NONE,
        0, 0, 0, 0,
        "",
        0, 0,
        1, NULL,
        0
};

static CTRLDATA CtrlSysInfo [] =
{
        {
                CTRL_STATIC,
                WS_VISIBLE | SS_LEFT,
                10, 10, 370, 180,
                IDC_SYSINFO,
                "test",
```

```
                0
        }
};

static NCS_RDR_INFO prop_rdr_info[] =
{
        {"classic", "classic", NULL},
};

static int init_propsheet (mDialogBox* self)
{
        // START_OF_CREATEPRPSHT
        mPropSheet *propsheet =
        (mPropSheet*) ncsCreateWindow (NCSCTRL_PROPSHEET,
        "", WS_VISIBLE | NCSS_PRPSHT_SCROLLABLE, WS_EX_NONE,
        IDC_PROPSHEET,
        10, 10, 390, 225, self->hwnd,
        NULL, prop_rdr_info, NULL, 0);
        // END_OF_CREATEPRPSHT

        if (!propsheet) {
                fprintf (stderr, "Error> Create propsheet failure.\n");
                return 1;
        }
```

```
// START_OF_ADDPAGES
PageSysInfo.controls = CtrlSysInfo;
PageSysInfo.caption = "Version Info";
PageSysInfo.dwAddData = PAGE_VERSION;
_c(propsheet)->addPage(propsheet, &PageSysInfo, mypage_handlers);

PageSysInfo.caption = "CPU Info";
PageSysInfo.dwAddData = PAGE_CPU;
_c(propsheet)->addPage(propsheet, &PageSysInfo, mypage_handlers);

PageSysInfo.caption = "MEM Info";
PageSysInfo.dwAddData = PAGE_MEMINFO;
_c(propsheet)->addPage(propsheet, &PageSysInfo, mypage_handlers);

PageSysInfo.caption = "Partition Info";
PageSysInfo.dwAddData = PAGE_PARTITION;
_c(propsheet)->addPage(propsheet, &PageSysInfo, mypage_handlers);

PageSysInfo.caption = "MiniGUI Info";
PageSysInfo.dwAddData = PAGE_MINIGUI;
_c(propsheet)->addPage(propsheet, &PageSysInfo, mypage_handlers);
// END_OF_ADDPAGES
```

```
        return 0;
}

static NCS_MNWND_TEMPLATE mymain_tmpl = {
        NCSCTRL_DIALOGBOX,
        1,
        0, 0, 420, 305,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "PropSheet Demo",
        NULL,
        NULL,
        NULL,
        _ctrl_tmpl,
        sizeof(_ctrl_tmpl)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};

int MiniGUIMain(int argc, const char* argv[])
{
        ncsInitialize();
        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_tmpl, HWND_DESKTOP);
```

```
        init_propsheet(mydlg);
        _c(mydlg)->doModal(mydlg, TRUE);

        MainWindowThreadCleanup(mydlg->hwnd);
        ncsUninitialize();
        return 0;
}
```

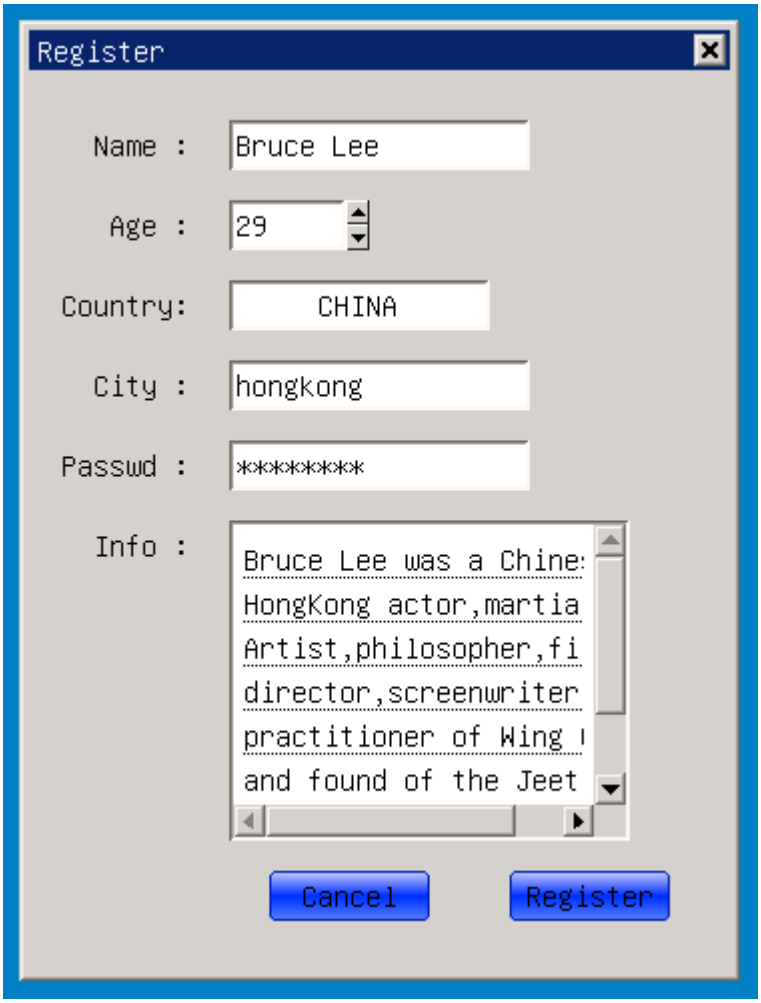
第二部分第十二章 编辑框系列控件

编辑框控件简介

编辑框控件（Edit）是 GUI 系统中不可或缺的的重要控件之一，主要用于接受用户的字符输入，实现用户交互和文本编辑的功能。实现上分为单行编辑框（Single Line Edit）和多行编辑框（Multiline Edit），单行编辑框用来接受用户内容相对简单的单行文本输入，与之相对，多行文本用来接受复杂的、大量的文本的输入。

- 编辑框类层次关系
    - mWidget
        - mScrollWidget
            - mItemView
                - mScrollView
                    - mEdit
                        - mSlEdit
                        - mMlEdit

- 示例图:



mEdit

控件窗口类

NCSCTRL_EDIT

控件英文名

Edit

简介

编辑框系列控件的基础类，是单行编辑框和多行编辑框的抽象父类，定义了两者共同的使用接口。

示意图

抽象类,不能直接使用

mEdit 风格

继承自 mScrollView 的风格

风格名	miniStudio 属性名	说明

NCSS_EDIT_LEFT	Align->Left	左对齐
NCSS_EDIT_CENTER	Align->Center	水平居中
NCSS_EDIT_RIGHT	Align->Right	右对齐
NCSS_EDIT_UPPERCASE	Case->Upper	输入内容自动转成大写
NCSS_EDIT_LOWERCASE	Case->Lower	输入内容自动转成小写
NCSS_EDIT_NOHIDESEL	HideSel ->FALSE	当编辑框失去焦点的时候，选中的内容依然保持选中状态
NCSS_EDIT_READONLY	ReadOnly ->TRUE	内容只读
NCSS_EDIT_BASELINE	BaseLine ->TRUE	内容带下划线显示

mEdit 属性

继承自 mScrollView 的属性

属性	miniStudio 属性名	类型	权限	说明
NCSP_EDIT_LIMITTEXT	MaxLength	int	RW	字符数目限定值
NCSP_EDIT_CARETPOS	--	int	RW	光标位置

mEdit 事件

继承自 mScrollView 的事件

事件 ID	参数	说明
NCSN_EDIT_CHANGE	--	内容变化
NCSN_EDIT_CONTCHANGED	--	当 edit 失去焦点时，内容发生变化了
NCSN_EDIT_UPDATE	--	内容被刷新，当通过 setText, resetContent,方法改变时，或改变属性时
NCSN_EDIT_SELCHANGED	--	选中部分改变
NCSN_EDIT_MAXTEXT	--	字符数量饱和
NCSN_EDIT_SETFOCUS	--	获得焦点
NCSN_EDIT_KILLFOCUS	--	失去焦点

mEdit 方法

继承自 mScrollView 的方法

setContent

```
void setContent(mEdit *self, const char* str, int start, int len)
```

- 参数：
  - str -- 显示在 edit 中的文字内容
  - start -- 显示开始位置(相对 str 的起始位置)，0 表示从头开始
  - len -- 显示字符数目，-1 表示一直到 str 的结束为止
- 说明：
  设置编辑框的显示内容，该方法会从 str 字符串中获取从第个字符位置开始的一共个字符，替换掉 edit 中现有的内容。
- 示例：

```
//edit 中会显示字符串"dddd Show Me"从第 6 个字符开始一直到结尾的字符串，这里就是“Show Me”

_c(edit)->setContent(edit, "dddd Show Me", 6, -1);
```

replaceText

```
void replaceText(mEdit *self, const char* str,

int start, int len, int replace_start, int replace_end)
```

- 参数：
  - str -- 用于替换的源字符串
  - start -- 用于替换的源文本相对于 str 的的偏移，0 表示从头开始
  - len -- 用到的源文本的长度，-1 表示从 start 开始，直到 str 结束
  - replace_start -- 替换开始的位置（相对于 edit 中的现有内容）
  - replace_end -- 替换结束的位置（相对于 edit 中的现有内容）
- 说明：
  字符串的替换，该方法会从 str 字符串中获取从第个字符位置开始的一共个字符，替换掉 edit 中现有的从到的内容。str 是要替换成的字符串，start 是相对 str 的起始位置，0 表示从头开始，len 是长度，-1 表示一直到 str 的结束为止，replace_start、replace_end 分别是要替换的位置的起点和终点，是相对于 edit 中现有的文本内容的位置偏移。
- 示例：

```
//edit 中会用字符串"dddd Show Me"从第 6 个字符开始一直到结尾的字符串（这里就是“Show Me”），

//来替换掉 edit 中现有文本的从第二个到第十个的字符

_c(edit)->replaceText(edit, "dddd Show Me", 6, -1, 2, 10);
```

insert

```
void insert(mEdit *self, const char* str, int start, int len, int at)
```

- 参数：
  - str -- 要插入的源字符串
  - start -- 使用到的源文本的起始位置，0 表示从头开始
  - len -- 使用到的源文本的长度，-1 表示从 start 开始，直到 str 结束
  - at -- 插入点的位置（相对于 edit 中的现有内容）,-1 表示结尾处
- 说明：
  字符串的插入，该方法会从 str 字符串中获取从第个字符位置开始的一共个字符，插入到 edit 中现有的内容的第个字符的位置。str 是要插入的字符串，start 是相对 str 的起始位置，0 表示从头开始，len 是长度，-1 表示一直到 str 的结束为止, at 参数是要插入的位置，是相对于 edit 中现有的文本内容的位置偏移。
- 示例：

```
//edit 中会用字符串"dddd Show Me"从第 6 个字符开始一直到结尾的字符串（这里就是“Show Me”），

//来插入到 edit 中现有文本的从第二个字符之后

_c(edit)->insert(edit, "dddd Show Me", 6, -1,  2);
```

append

```
void append(mEdit *self, const char* str, int start, int len)
```

- 参数：
  - str -- 要追加的源字符串
  - start -- 使用到的源文本的起始位置，0 表示从头开始
  - len -- 使用到的源文本的长度，-1 表示从 start 开始，直到 str 结束
- 说明：
  字符串的追加，该方法会从 str 字符串中获取从第个字符位置开始的一共个字符，追加到 edit 中现有的内容的后面。str 是要追加的字符串，start 是相对 str 的起始位置，0 表示从头开始，len 是长度，-1 表示一直到 str 的结束为止。
- 示例：

```
//edit 中会用字符串"dddd Show Me"从第 6 个字符开始一直到结尾的字符串（这里就是“Show Me”），

//来追加到 edit 中现有文本的末尾

_c(edit)->append(edit, "dddd Show Me", 6, -1);
```

getTextLength

```
int getTextLength(mEdit *self)
```

- 说明：
  获取 Edit 中字符串内容的长度
- 示例：

```
int text_len = _c(edit)->getTextLength(edit);
```

getContent

```
int getContent(mEdit *self, char *strbuff, int buf_len, int start, int end)
```

- 参数：
  - strbuff -- 获取到字符串的存放位置（应该提前分配好存放空间）
  - buff_len -- strbuff 的大小
  - start -- 获取内容的起始位置
  - end -- 获取内容的终止位置
- 说明：

获取 Edit 中字符串内容，从现有的内容中获取从到位置的内容写入到 strbuff 中，写入的最大数目限定为 buff_len。

- 示例：

```
char buff[128];

_c(edit)->getContent(edit, buff, 127, 0, -1);//取出 edit 中的全部内容，buff 最多存 127 个字符
```

setSel、getSel

```
int setSel(mEdit *self, int start, int end)

int getSel(mEdit *self, int *pstart, int *pend)
```

- 参数：
  - start \end -- 选中区域的起点、终点
  - pstart\pend -- 选中区域的起点、终点，用于函数返回
- 说明：
  设置、获取选中文本的区域，后两个参数分别对应选中区域的起点和终点。
- 示例：

```
_c(edit)->setSel(edit, 2, 10);//设置 edit 的第 2~10 个字符为选中状态



int ps,pe;

_c(edit)->getSel(edit, &ps, &pe);//获取选中区域
```

setMargin

```
void setMargin(mEdit *self, int left, int top, int right, int bottom)
```

- 参数：
  - left,top,right,bottom -- 上下左右的留白，这个参数不是矩形的概念，只是沿用了矩形的数据结构方便参数传递
- 说明：

设置编辑区的上下左右留白

- 示例：

```
//设置留白

_c(edit)- >setMargin(edit, 10,10,10,10);
```

copy、paste、cut

```
void copy(mEdit *self)

void cut(mEdit *self)

void paste(mEdit *self)

TextCopyPaste * setCopyPaste(mEdit *self, TextCopyPaste* cp)
```

剪切、复制、粘贴都是针对选中区域的操作，edit 中默认实现了一组使用 minigui 剪切板的操作集,用户还可

以通过 setCopyPaste 设置自己实现的操作集。

makevisible

```
BOOL makevisible(mEdit *self, int pos)
```

- 参数：
  - pos -- 需要可见的位置
- 说明：

控制编辑区的滚动，使位置的字符变为可见。

- 示例：

```
//是第 201 个字符处可见

_c(edit)- >makevisible(edit, 201);
```

mSlEdit

控件窗口类

NCSCTRL_SLEDIT

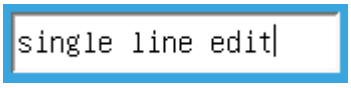
控件英文名

Single Line Edit 或者 SlEdit

简介

单行文本编辑框

示意图



mSlEdit 风格

继承自 mEdit 的风格

风格名	miniStudio 属性名	说明
NCSS_SLEDIT_PASSWORD	Password->TRUE	编辑框的内容以密码输入的方式屏蔽显示
NCSS_SLEDIT_AUTOSELECT	AutoSelect ->TRUE	自动选中风格，获得焦点后文本自动成选中状态

mSlEdit 属性

继承自 mEdit 的属性

属性	miniStudio 属性名	类型	权限	说明
NCSP_SLEDIT_TIPTEXT	ToolTip	char *	RW	提示信息字符串，当没有输入时，用来提示用户的信息
NCSP_SLEDIT_PWDCHAR	PasswordChar	char	RW	pass word 显示的字符，只有 NCSS_SLEDIT_PASSWORD 风格的有效，默认是'*'

mSlEdit 事件

继承自 mEdit 的事件

事件 ID	参数	说明
NCSN_SLEDIT_ENTER	--	捕获到 enter 键消息

mSlEdit 方法

继承自 mEdit 的方法

没有新引入的方法

mSlEdit 编程示例

- SlEdit 示例代码 ： edit.c
- 我们这样来定义 SlEdit 的使用模板

```
{
        NCSCTRL_STATIC,
        0,
        10, HSTART, 70, 25,
        WS_VISIBLE,
```

```
        WS_EX_NONE,
        "Name :",
        static_props,
        NULL,
        NULL, NULL, 0, 0
},
{   //左对齐
        NCSCTRL_SLEDIT,
        ID_NAME,
        100, HSTART, 150, 25,
        WS_BORDER | WS_VISIBLE | NCSS_EDIT_LEFT,
        WS_EX_NONE,
        "",
        NULL,
        NULL,
        NULL, NULL, 0, 0
},
{
        NCSCTRL_STATIC,
        0,
        10, HSTART + HSPACE, 70, 25,
        WS_VISIBLE,
        WS_EX_NONE,
        "Age :",
```

```
        static_props,
        NULL,
        NULL, NULL, 0, 0
},
{
        NCSCTRL_SPINBOX,
        ID_SPIN,
        100, HSTART + HSPACE, 70, 25,
        WS_VISIBLE | NCSS_SPNBOX_NUMBER | NCSS_SPNBOX_AUTOLOOP,
        WS_EX_NONE,
        "",
        spin_props,
        NULL,
        NULL, NULL, 0, 0
},
{
        NCSCTRL_STATIC,
        0,
        10, HSTART + 2 * HSPACE, 70, 25,
        WS_VISIBLE,
        WS_EX_NONE,
        "Country:",
        static_props,
        NULL,
```

```
        NULL, NULL, 0, 0
},
{   //居中对齐， 大写字母
        NCSCTRL_SLEDIT,
        ID_COUN,
        100, HSTART + 2 * HSPACE, 130, 25,
        WS_BORDER | WS_VISIBLE | NCSS_EDIT_CENTER | NCSS_EDIT_UPPERCASE,
        WS_EX_NONE,
        "",
        NULL,
        NULL,
        NULL, NULL, 0, 0
},
{
        NCSCTRL_STATIC,
        0,
        10, HSTART + 3 * HSPACE, 70, 25,
        WS_VISIBLE,
        WS_EX_NONE,
        "City :",
        static_props,
        NULL,
        NULL, NULL, 0, 0
},
```

```
{   //小写字母
        NCSCTRL_SLEDIT,
        ID_CITY,
        100, HSTART + 3 * HSPACE, 150, 25,
        WS_BORDER | WS_VISIBLE | NCSS_EDIT_LOWERCASE,
        WS_EX_NONE,
        "",
        NULL,
        NULL,
        NULL, NULL, 0, 0
},
{
        NCSCTRL_STATIC,
        0,
        10, HSTART + 4 * HSPACE, 70, 25,
        WS_VISIBLE,
        WS_EX_NONE,
        "Passwd :",
        static_props,
        NULL,
        NULL, NULL, 0, 0
},
{   //密码输入形式的 edit
        NCSCTRL_SLEDIT,
```

```
        ID_PSWD,
        100, HSTART + 4 * HSPACE, 150, 25,
        WS_BORDER | WS_VISIBLE | NCSS_SLEDIT_PASSWORD,
        WS_EX_NONE,
        "",
        NULL,
        NULL,
        NULL, NULL, 0, 0
},
```

mMlEdit

控件窗口类

NCSCTRL_MLEDIT

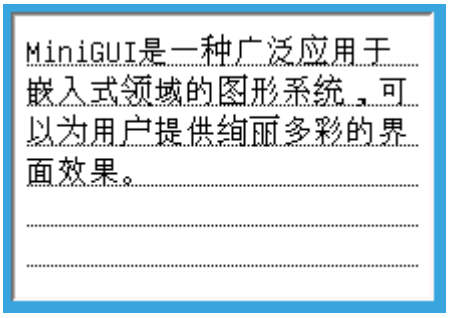
控件英文名

Multiline Edit 或者 MlEdit

简介

多行文本编辑框

示意图



mMlEdit 风格

继承自 mEdit 的风格

风格名	miniStudio 属性名	说明
NCSS_MLEDIT_AUTOWRAP	AutoWrap ->TRUE	自动换行

mMlEdit 属性

继承自 mEdit 的属性

属性	miniStudio 属性名	类型	权限	说明
NCSP_MLEDIT_LINECOUNT	--	int	RO	行数
NCSP_MLEDIT_LINEHEIGHT	LineHeight	int	RW	行高
NCSP_MLEDIT_LINEFEEDISPCHAR	--	char	WO	换行符改用该字符显示出来
NCSP_MLEDIT_LINESEP	LineSeperator	char	RW	换行符标记，默认是'\n'
NCSP_MLEDIT_CARETSHAPE	CaretShap	int	RW	光标形状 ED_CARETSHAPE_LINE 或者 ED_CARETSHAPE_BLOCK
NCSP_MLEDIT_NUMOFPARAGRAPHS	--	int	RO	段落的数目

mMlEdit 事件

继承自 mEdit 的事件

没有新引入的事件

mMlEdit 方法

继承自 mEdit 的方法

mMlEdit 编程示例

- MlEdit 示例代码 ： edit.c
- 我们这样来定义 MlEdit 的使用模板

```
{
        NCSCTRL_STATIC,
        0,
        10, HSTART + 5 * HSPACE, 70, 25,
```

```
        WS_VISIBLE,
        WS_EX_NONE,
        "Info :",
        static_props,
        NULL,
        NULL, NULL, 0, 0
},
{   //多行编辑框
        NCSCTRL_MLEDIT,
        ID_INFO,
        100, HSTART + 5 * HSPACE, 200, 160,
        WS_BORDER | WS_VISIBLE | WS_VSCROLL | NCSS_EDIT_BASELINE,
        WS_EX_NONE,
        "",
        NULL,
        NULL,
        NULL, NULL, 0, 0
},
```

第二部分第十三章 动画控件类

动画控件简介

动画控件通过对图片组和 gif 动态图片的绘制和显示，为用户提供了丰富的视觉体验。 mGNCS 中的动画控件源于 Minigui 3.0 的 animation 控件，在其基础上增加了对图片组播放和 gif 图片播放控制的支持。 同时， 用户可以通过设置图片播放间隔时间， 来控制播放速度。

- Animate 类层次关系
  - mStatic
    - mAnimate

- 控件创建方法
  - 自动创建：通过 miniStudio 中的界面设计器，拖拽对应的动画控件，在属性栏中选择需要加载的 GIF 图片或图片目录，miniStudio 将自动创建控件，并提供可视化的控件配置，同时自动生成创建代码。
  - 手动生成：按照 mGNCS 控件创建过程，通过编程，传入对应控件窗口类 ID，生成控件，手动编程设置控件属性、事件处理。

mAnimate

控件窗口类

NCSCTRL_ANIMATE

控件英文名

Animate

简介

动画控件。通过在控件上加载不同的图片处理对象， 来显示绘制动态图片。

示意图

mAnimate 风格

继承自 mStatic 的风格

风格 ID	miniStudio 属性名	说明
NCSS_ANMT_AUTOLOOP	Loop	设置动画播放是否为自动循环， 设置为 True 时，图片循环播放， 设置为 False 时，图片播放完毕则停止
NCSS_ANMT_SCALE	Scale->ImageScaled	设置图片播放面积为控件区域面积， 可以对图片进行扩大或缩减播放

NCSS_ANMT_AUTOFIT	Scale->AutoFit	设置控件区域自动匹配为图片面积
NCSS_ANMT_AUTOPLAY	Autoplay	设置图片是否为自动播放， True 时， 图片自动播放，False 时， 需要用户发送命令控制播放

mAnimate 属性

继承自 mStatic 的属性

属性 ID 名	miniStudio 属性名	类型	权限	说明
NCSP_ANMT_GIFFILE	GIFFile	String	RW	动画控件加载的 GIF 图片文件名
NCSP_ANMT_DIR	Dir	String	RW	动画控件加载的目录名， 控件加载目录内的所有图片，图片播放是按照所支持图片首字母 ascii 码顺序
NCSP_ANMT_INTERVAL	Interval	int	RW	播放图片时帧与帧之间的时间间隔

mAnimate 事件

继承自 mStatic 的事件

mAnimate 渲染器

非客户区的绘制请查阅 mStatic 的渲染器

mAnimate 方法

ncsAnimateStart

```
BOOL ncsAnimateStart(mAnimate* self);
```

- 参数：
  - self -- 需要操作的动画控件指针
- 说明：
  开始播放 self 对应的动画控件， 如果动画控件已经在播放， 则回到动画起始处开始播放。
- 示例：

```
//播放 anim 动画
```

```
mAnimate *anim = (mAnimate *)ncsGetChildObj(hwnd, IDC_ANI);

ncsAnimateStart(anim);
```

ncsAnimatePauseResume

```
BOOL ncsAnimatePauseResume(mAnimate* self);
```

- 参数：
  - self -- 需要操作的动画控件指针
- 说明：
  如果当前动画为暂停，此方法会继续播放动画， 如果动画为播放中， 此方法会暂停动画。
- 示例：

```
//暂停 anim 动画

mAnimate *anim = (mAnimate *)ncsGetChildObj(hwnd, IDC_ANI);

ncsAnimateStart(anim);

ncsAnimatePauseResume(anim);
```

ncsAnimateStop

```
BOOL ncsAnimateStop(mAnimate* self);
```

- 参数：
  - self -- 需要操作的动画控件指针
- 说明：
  停止动画， 并将动画重置回第一帧。
- 示例：

```
//暂停 anim 动画

mAnimate *anim = (mAnimate *)ncsGetChildObj(hwnd, IDC_ANI);

ncsAnimateStop(anim);
```

mAnimate 编程示例

- 运行截图:

- 本程序窗口区域上边的图片为 gif 动态图片， 下面是对程序所在目录中图片的幻灯片播放， 三个按钮 start, pause, stop 是对 gif 播放的控制。
- 普通按钮完整示例代码：animation.c

清单 p2c13-1 animation.c

- 设置加载图片

```
static NCS_PROP_ENTRY animate_props [] = {
        { NCSP_ANMT_GIFFILE, (DWORD)"tuzi1.gif" },
        { NCSP_ANMT_INTERVAL, 6 },
        {0, 0}
};

static NCS_PROP_ENTRY animate_props_ex [] = {
        { NCSP_ANMT_DIR, (DWORD)"." },
        { NCSP_ANMT_INTERVAL, 100 },
        {0, 0}
};
```

- 设置按键消息

```
static void btn_notify(mWidget *button, int id, int nc, DWORD add_data)
{
```

```
        mAnimate *anim = (mAnimate *)ncsGetChildObj(GetParent(button->hwnd), IDC_ANI);

        switch (id)
        {
                case IDC_BTN1 :
                ncsAnimateStart(anim);
                break;
                case IDC_BTN2 :
                ncsAnimatePauseResume(anim);
                break;
                case IDC_BTN3 :
                ncsAnimateStop(anim);
                break;
        }

}

static NCS_EVENT_HANDLER btn_handlers [] = {
        NCS_MAP_NOTIFY(NCSN_BUTTON_PUSHED, btn_notify),
        {0, NULL}
};
```

- 设置显示界面模板

```
static NCS_WND_TEMPLATE _ctrl_templ[] = {
```

```
    {
        NCSCTRL_ANIMATE,
        IDC_ANI,
        50, 50, 300, 300,
        WS_BORDER | WS_VISIBLE |NCSS_ANMT_AUTOFIT | NCSS_ANMT_AUTOLOOP | \
        NCSS_ANMT_AUTOPLAY,
        WS_EX_NONE,
        "test",
        animate_props, //props,
        animate_rdr_info,
        NULL, //handlers,
        NULL, //controls
        0,
        0 //add data
    },
    {
        NCSCTRL_ANIMATE,
        IDC_ANIM,
        0, 230, 300, 300,
        WS_BORDER | WS_VISIBLE | NCSS_ANMT_AUTOLOOP | NCSS_ANMT_AUTOFIT | \
        NCSS_ANMT_AUTOPLAY,,
        WS_EX_NONE,
        "test2",
        animate_props_ex, //props,
```

```
            animate_rdr_info,
            NULL, //handlers,
            NULL, //controls
            0,
            0 //add data
        },
        {
            NCSCTRL_BUTTON,
            IDC_BTN1,
            450, 100, 70, 30,
            WS_VISIBLE | NCSS_NOTIFY,
            WS_EX_NONE,
            "Start",
            NULL,
            btn_rdr_info,
            btn_handlers,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_BUTTON,
            IDC_BTN2,
            450, 200, 70, 30,
```

```
            WS_VISIBLE | NCSS_NOTIFY,
            WS_EX_NONE,
            "Pause",
            NULL,
            btn_rdr_info,
            btn_handlers,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_BUTTON,
            IDC_BTN3,
            450, 300, 70, 30,
            WS_VISIBLE | NCSS_NOTIFY,
            WS_EX_NONE,
            "Stop",
            NULL,
            btn_rdr_info,
            btn_handlers,
            NULL,
            0,
            0
        },
```

```
};
```

第二部分第十四章 其他高级控件类

其他高级控件类简介

该类控件主要是视图系列控件，多用于显示各种不同类型的列表项信息。

其他高级控件类的类继承关系如下：

- mWidget
  - mScrollWidget
    - mItemView
      - mListBox
      - mScrollView
      - mIconview
      - mListview

在高级控件类中将列表项作为一个对象来管理，不同控件使用到的列表项对象不同，各自的继承关系如下：

- mObject
  - mItem
    - mItemManager
      - mListItem
    - mListColumn

我们首先介绍各列表项对象，再具体介绍各控件类。

mItem

mItem 对象是所有列表项的基类，提供一系列基本的访问方法。直接使用该对象的有 mListBox, mScrollView 和 mIconView 等控件类。

mItem 状态

状态名	说明
NCSF_ITEM_NORMAL	条目状态：正常
NCSF_ITEM_SELECTED	条目状态：选中
NCSF_ITEM_DISABLED	条目状态：禁用

NCSF_ITEM_USEBITMAP	条目状态：包含位图，与 NCSF_ITEM_USEICON 互斥
NCSF_ITEM_USEICON	条目状态：包含图标，与 NCSF_ITEM_USEBITMAP 互斥

mItem 属性

属性 ID	类型	权限	说明
NCSP_ITEM_HEIGHT	int	RW	条目当前高度
NCSP_ITEM_FLAGS	DWORD	RW	条目状态

mItem 方法

继承自 mObject 的方法

一个条目可以包含多种内容信息，如文本、图片等等。提供的基础方法主要有：

```
BOOL isSelectedItem(mItem *self);

BOOL isEnabledItem(mItem *self);

void setItemEditor(mItem *self, hEditor editor);

hEditor getItemEditor(mItem *self);

void setItemAddData(mItem *self, DWORD addData);

DWORD getItemAddData(mItem *self);

void setItemImage(mItem *self, DWORD image);

DWORD getItemImage(mItem *self);

void setItemFlags(mItem *self, DWORD flags);

DWORD getItemFlags(mItem *self);

BOOL setItemHeight(mItem *self, int height);

int getItemHeight(mItem *self);

BOOL setItemString(mItem *self, const char* string);

char* getItemString(mItem *self);
```

- isSelectedItem：判断当前条目是否处于选中状态。
- isEnabledItem：判断当前条目是否处于使能状态。
- setItemString：设置条目文本。
- getItemString：获取条目当前文本。
- setItemHeight：设置条目高度。
- getItemHeight：获取条目高度。
- setItemFlags：设置条目状态。
- getItemFlags：获取条目状态。
- setItemImage：设置条目图标。
- getItemImage：获取条目图标。
- setItemAddData：设置条目附加数据。
- getItemAddData：获取条目附加数据。
- setItemEditor：设置条目编辑器。
- getItemEditor：获取条目编辑器。

mItemManager

通过链表管理 mItem 及其子类的基类管理器，提供对列表项的添加、删除和设置等方法。

mItemManager 状态

继承自 mItem 状态

状态名	说明
NCSF_ITMMNG_AUTOSORT	条目插入时自动排序
NCSF_ITMMNG_FROZEN	禁止或使能条目刷新功能

mItemManager 属性

继承自 mItem 属性

属性 ID	类型	权限	说明
NCSP_ITMMNG_ITEMCOUNT	int	RO	包含的条目数
NCSP_ITMMNG_FLAGS	DWORD	RW	条目状态
NCSP_ITMMNG_TOTALHEIGHT	int	RO	所有条目总高度

mItemManager 方法

继承自 mItem 方法

回调方法

mItemManager 类支持根据列表项或列表项字符串进行比较的 2 个回调方法。其中根据列表项比较的方法优先于根据列表项字符串比较的方法。

```
typedef int (*NCS_CB_CMPITEM) (mItemManager *manager, HITEM hItem1, HITEM hItem2);

typedef int (*NCS_CB_CMPSTR) (const char* s1, const char* s2, size_t n);

NCS_CB_CMPITEM setItemCmpFunc(mItemManager *self, NCS_CB_CMPITEM func);

NCS_CB_CMPSTR setStrCmpFunc(mItemManager *self, NCS_CB_CMPSTR func);

NCS_CB_CMPSTR getStrCmpFunc(mItemManager *self);
```

- setItemCmpFunc：用于设置根据列表项自身进行比较的回调方法，同时返回先前设置的方法。
- setStrCmpFunc：用于设置根据列表项字符串进行比较的回调方法，同时返回先前设置或默认的方法。
- getStrCmpFunc：用于获取当前使用的字符串比较方法。

排序方法

```
void setAutoSortItem(mItemManager *self, BOOL sort);

int sortItems(mItemManager *self, NCS_CB_CMPITEM pfn);
```

- setAutoSortItem：在添加所有列表项前设置或取消排序标记。当排序标记设置成功后，在其后添加的所有列表项将和已有的列表项进行相应比较后确定插入位置。
- sortItems：按照指定的列表项比较方法对所有列表项进行重新排序。

刷新方法

```
BOOL freeze(mItemManager *self, BOOL lock);

BOOL isFrozen(mItemManager *self);

int adjustItemsHeight(mItemManager *self, int diff);
```

- freeze：冻结或恢复对列表项的刷新。
- isFrozen：判断当前是否处于可刷新状态。
- adjustItemsHeight：调整列表项总高度变化值，在管理器处理可刷新状态时，该方法在调整完大小后会将变化立即反映到 UI 上，否则将不刷新 UI。

创建/删除/移动列表项

在对列表项进行插入时，首先确认是否是自动排序支持，如果是，则按照自动排序算法计算排序插入位置；否则根据前、后列表项或指定索引依次进行插入尝试，尝试成功则返回正确位置。

```
HITEM createItem(mItemManager *self, HITEM prev, HITEM next, int index, int *pos);

int insertItem(mItemManager *self, HITEM hItem, HITEM prev, HITEM next, int index, int *pos);

int moveItem(mItemManager *self, mItem *curItem, int count, mItem* prevItem);

int removeItem(mItemManager *self, HITEM hItem);

BOOL removeAll(mItemManager *self);
```

- createItem：创建一个新的列表项并插入到指定位置，同时通过最后的参量返回已插入的位置。
- insertItem：将已创建的列表项插入到管理器镶的指定位置，同时通过最后的参量返回已插入的位置。
- moveItem：将当前列表项及其后指定数目的列表项移动到某一列表项位置之后。
- removeItem：删除某一指定列表项。
- removeAll：删除所有列表项。

遍历列表项

```
list_t* getQueue(mItemManager *self);

HITEM getListEntry(list_t* entry);



HITEM getFirstItem(mItemManager *self);

HITEM getNext(mItemManager *self, HITEM hItem);

HITEM getPrev(mItemManager *self, HITEM hItem);
```

- getQueue：获取管理器的列表项列首。
- getListEntry：获取指定链表项的列表项指针。
- getFirstItem：获取管理器中的第一个列表项。

- getNext：获取指定列表项的后一个列表项。
- getPrev：获取指定列表项的前一个列表项。

获取列表项信息

```
HITEM getItem(mItemManager *self, int index);

int indexOf(mItemManager *self, HITEM hItem);



int inItem(mItemManager *self, int mouseX, int mouseY, HITEM *pRet, POINT *pt);

int getItemYPos(mItemManager *self, HITEM hItem);



int getTotalHeight(mItemManager *self);

int getItemCount(mItemManager *self);

int isEmpty(mItemManager *self);

BOOL getSelection(mItemManager *self, HITEM *pRet, int count);

int getSelectionCount(mItemManager *self);
```

- getItem：获取指定索引的列表项。
- indexOf：获取指定列表项的索引。
- inItem：获取指定鼠标位置下的列表项，并返回列表项的起始位置。
- getItemYPos：获取指定列表项的起始纵向坐标。
- getTotalHeight：获取列表项管理器的整体高度。
- getItemCount：获取列表项总数目。
- isEmpty：判断列表项是否为空。
- getSelectionCount：获取选中列表项数目。
- getSelection：获取指定数目的选中列表项信息。

设置/获取列表项状态

```
BOOL isEnabled(mItemManager *self, HITEM hItem);

BOOL enable(mItemManager *self, HITEM hItem, BOOL enable);
```

```
BOOL isSelected(mItemManager *self, HITEM hItem);
BOOL select(mItemManager *self, HITEM hItem, BOOL sel);
void selectAll(mItemManager *self);
void deselectAll(mItemManager *self);

HITEM hilight(mItemManager *self, HITEM hItem);
HITEM getHilight(mItemManager *self);
BOOL isHilight(mItemManager *self, HITEM hItem);



int setHeight(mItemManager *self, HITEM hItem, int height);
int getHeight(mItemManager *self, HITEM hItem);

void setAddData(mItemManager *self, HITEM hItem, DWORD addData);
DWORD getAddData(mItemManager *self, HITEM hItem);

void setImage(mItemManager *self, HITEM hItem, DWORD image);
DWORD getImage(mItemManager *self, HITEM hItem);

void setFlags(mItemManager *self, HITEM hItem, DWORD flags);
DWORD getFlags(mItemManager *self, HITEM hItem);

BOOL setText(mItemManager *self, HITEM hItem, const char* text);
```

```
const char* getText(mItemManager *self, HITEM hItem);

int getTextLen(mItemManager *self, HITEM hItem);
```

- isEnabled：判断指定列表项是否处于使能状态。
- enable：使能或禁止指定列表项。
- isSelected：判断指定列表项是否处于选择状态。
- select：选择或取消选择指定列表项。
- selectAll：选中所有列表项。
- deselectAll：取消所有列表项的选中状态。
- hilight：高亮选择指定的列表项。
- getHilight：获取当前高亮选择列表项。
- isHilight：判断指定列表项是否处于高亮状态。
- setHeight：设置列表项高度。
- getHeight：获取列表项高度。
- setAddData：设置列表项附加数据。
- getAddData：获取列表项附加数据。
- setImage：设置列表项位图信息。
- getImage：获取列表项位图信息。
- setFlags：设置列表项状态标志。
- getFlags：获取列表项状态标志。
- setText：设置列表项文本字符串。
- getText：获取列表项文本字符串。
- getTextLen：获取列表项文本字符串长度。

mListItem

mListItem 对象用于描述 mListView 控件的行对象。

mListItem 状态

继承自 mItemManger 状态

状态名	说明
NCSF_LSTITM_FOLD	折叠条目
NCSF_LSTITM_PRIVBKCOL	条目包含私有背景色
NCSF_LSTITM_PRIVBKCOL	条目包含私有前景色

mListItem 属性

继承自 mItemManager 属性

属性 ID	类型	权限	说明
NCSP_LSTITM_NRCHILD	int	RO	包含的子条目数
NCSP_LSTITM_DEPTH	int	RO	条目的深度
NCSP_LSTITM_RHEIGHT	int	RO	条目可见时的实际高度

mListItem 方法

继承自 mItemManager 方法

mListItem 对象提供了一系列操作方法：

```
void setBackground(mListItem *self, int index, int *color);

void setForeground(mListItem *self, int index, int *color);

int getBackground(mListItem *self, int index, int *color);

int getForeground(mListItem *self, int index, int *color);

BOOL addChild(mListItem *self, mListItem *child);

BOOL delChild(mListItem *self, mListItem *child);

BOOL setFold(mListItem *self, BOOL fold);

mListItem* getParent(mListItem *self);

int getChildCount(mListItem *self);

int getDepth(mListItem *self);

BOOL isFold(mListItem *self);
```

- setBackground：用于设置单元背景色。当 *color* 为空时对原设置进行清空处理；当 *index* 索引无效时，设置默认行背景色，否则设置指定单元背景色。
- getBackground：用于获取单元背景色。当 *index* 索引无效时，通过 *color* 返回默认行背景色，否则返回指定单元背景色。

- setForeground：用于设置单元前景色。当 *color* 为空时对原设置进行清空处理；当 *index* 索引无效时，设置默认行前景色，否则设置指定单元前景色。
- getForeground：用于获取单元前景色， 当 *index* 索引无效时，通过 *color* 返回默认行前景色，否则返回指定单元前景色。
- addChild：添加指定的子条目。
- delChild：删除指定的子条目。
- setFold：用于折叠或展开指定条目。
- getparent：用于获取指定条目的父条目。
- getChildCount：获取包含的子条目数目。
- getDepth：获取条目深度。
- isFold：判断条目是否处于折叠状态。

mListColumn

mListColumn 对象用于描述 mListView 控件的列对象。

mListColumn 状态

继承自 mItem 状态

状态名	说明
NCSF_LSTCLM_LEFTALIGN	文本对齐方式：左对齐
NCSF_LSTCLM_RIGHTALIGN	文本对齐方式：右对齐
NCSF_LSTCLM_CENTERALIGN	文本对齐方式：居中对齐

mListColumn 属性

继承自 mItem 属性

属性 ID	类型	权限	说明
NCSP_LSTCLM_POSX	int	RW	列起始位置横坐标
NCSP_LSTCLM_WIDTH	int	RW	列宽
NCSP_LSTCLM_SORTTYPE	ncsLstClmSortType	RW	列表项排序方式：值为升序、降序或不排序三者选其一
NCSP_LSTCLM_CMPFUNC	NCS_CB_LISTV_CMPCLM	RW	两列表项比较的回调方法

下面是属性中涉及的数据结构定义：

```
typedef enum

{

        NCSID_LSTCLM_NOTSORTED = 0,  //不排序

        NCSID_LSTCLM_HISORTED, //升序

        NCSID_LSTCLM_LOSORTED //降序

}ncsLstClmSortType;



typedef struct _NCS_LSTCLM_SORTDATA

{

        int     column;  //排序列索引

        int     losorted; //排序列排序方式

        mWidget *obj; //包含比较项的控件类

}NCS_LSTCLM_SORTDATA;



typedef int (*NCS_CB_LISTV_CMPCLM)(HITEM nItem1, HITEM nItem2, NCS_LSTCLM_SORTDATA *sortData);
```

mItemView

基类，不允许直接使用。

mItemView 风格

继承自 mScrollWidget 的风格

风格名	miniStudio 属性名	说明
NCSS_ITEMV_AUTOSORT	-	条目自动排序
NCSS_ITEMV_LOOP	-	条目可循环浏览

NCSS_ITEMV_SINGLE	-	条目单选支持，默认风格
NCSS_ITEMV_MULTIPLE	-	条目多选支持

mItemView 属性

继承自 mScrollWidget 的属性

属性 ID	miniStudio 名	类型	权限	说明
NCSP_ITEMV_DEFITEMHEIGHT	-	int	RW	条目默认高度
NCSP_ITEMV_ITEMCOUNT	-	int	RO	条目总数目

mItemView 事件

继承自 mScrollWidget 的事件

事件通知码	说明	参数
NCSN_ITEMV_CLICKED	鼠标点击事件	被点击的条目句柄
NCSN_ITEMV_SELCHANGING	正在改变选择的条目	处于高亮状态的条目句柄
NCSN_ITEMV_SELCHANGED	选择条目已改变	新的选择条目句柄
NCSN_ITEMV_ENTER	Enter 键被按下	-
NCSN_ITEMV_SETFOCUS	获取焦点	-
NCSN_ITEMV_KILLFOCUS	失去焦点	-

mItemView 方法

继承自 mScrollWidget 的方法

回调方法

mItemView 提供了初始化、绘制和销毁列表项 3 个回调方法，同时还有比较列表项的回调方法。

```
typedef int  (*NCS_CB_INITITEM)(mItemView *self, HITEM hItem);

typedef void (*NCS_CB_DSTRITEM)(mItemView *self, HITEM hItem);
```

```
typedef void (*NCS_CB_DRAWITEM)(mItemView *self, HITEM hItem, HDC hdc, RECT *rcDraw);


NCS_CB_DRAWITEM setItemDraw(mItemView *self, NCS_CB_DRAWITEM func);

NCS_CB_INITITEM setItemInit(mItemView *self, NCS_CB_INITITEM func);

NCS_CB_DSTRITEM setItemDestroy(mItemView *self, NCS_CB_DSTRITEM func);


NCS_CB_CMPITEM setItemCmpFunc(mItemView *self, NCS_CB_CMPITEM func);

NCS_CB_CMPSTR setStrCmpFunc(mItemView *self, NCS_CB_CMPSTR func);

NCS_CB_CMPSTR getStrCmpFunc(mItemView *self);
```

排序方法

```
void setAutoSortItem(mItemView *self, BOOL sort);

int sortItems(mItemView *self, NCS_CB_CMPITEM pfn);
```

- setAutoSortItem：在添加所有列表项前设置或取消排序标记。当排序标记设置成功后，在其后添加的所有列表项将和已有的列表项进行相应比较后确定插入位置。
- sortItems：按照指定的列表项比较方法对所有列表项进行重新排序。

刷新方法

```
void freeze(mItemView *self, BOOL lock);

BOOL isFrozen(mItemView *self);

int adjustItemsHeight(mItemView *self, int diff);
```

- freeze：冻结或恢复对列表项的刷新。
- isFrozen：判断当前是否处于可刷新状态。
- adjustItemsHeight：调整列表项总高度变化值，该方法在控件调整完列表项大小后会将变化立即反映到 UI 上，否则将不刷新 UI。

创建/删除/移动列表项

在对列表项进行插入时，首先确认是否是自动排序支持，如果是，则按照自动排序算法计算排序插入位置；否则根据前、后列表项或指定索引依次进行插入尝试，尝试成功则返回正确位置。

```
HWND createItem(mItemView *self, HITEM prev, HITEM next, int index,

int height, DWORD addData, int *pos, BOOL adjust);

int insertItem(mItemView *self, HITEM hItem, HITEM prev, HITEM next,

int index, int *pos);

int removeItem(mItemView *self, HITEM hItem);



BOOL removeAll(mItemView *self);
```

- createItem：创建一个新的列表项并插入到指定位置，同时通过最后的参量返回已插入的位置。
- insertItem：将已创建的列表项插入到管理器镶的指定位置，同时通过最后的参量返回已插入的位置。
- removeItem：删除某一指定列表项。
- removeAll：删除所有列表项。

遍历列表项

```
list_t* getQueue(mItemView *self);

HITEM getListEntry(mItemView *self, list_t* entry);

HITEM getFirstItem(mItemView *self);

HITEM getNext(mItemView *self, HITEM hItem);

HITEM getPrev(mItemView *self, HITEM hItem);
```

- getQueue：获取管理器的列表项列首。
- getListEntry：获取指定链表项的列表项指针。
- getFirstItem：获取管理器中的第一个列表项。
- getNext：获取指定列表项的后一个列表项。
- getPrev：获取指定列表项的前一个列表项。

获取列表项信息

```
HITEM getItem(mItemView *self, int index);
```

```
int indexOf(mItemView *self, HITEM hItem);


int inItem(mItemView *self, int mouseX, int mouseY, HITEM *pRet, POINT *pt);


int getTotalHeight(mItemView *self);

int getItemCount(mItemView *self);

int isEmpty(mItemView *self);

int getSelectionCount(mItemView *self);

BOOL getSelection(mItemView *self, HITEM *pRet, int count);
```

- getItem：获取指定索引的列表项。
- indexOf：获取指定列表项的索引。
- inItem：获取指定鼠标位置下的列表项，并返回列表项的起始位置。
- getTotalHeight：获取列表项管理器的整体高度。
- getItemCount：获取列表项总数目。
- isEmpty：判断列表项是否为空。
- getSelectionCount：获取选中列表项数目。
- getSelection：获取指定数目的选中列表项信息。

设置/获取列表项状态

```
BOOL isEnabled(mItemView *self, HITEM hItem);

BOOL enable(mItemView *self, HITEM hItem, BOOL enable);


BOOL isSelected(mItemView *self, HITEM hItem);

BOOL select(mItemView *self, HITEM hItem);

BOOL deselect(mItemView *self, HITEM hItem);


void selectAll(mItemView *self);
```

```
void deselectAll(mItemView *self);


int hilight(mItemView *self, HITEM hItem);

HITEM getHilight(mItemView *self);

BOOL isHilight(mItemView *self, HITEM hItem);


int setItemHeight(mItemView *self, HITEM hItem, int height);

int getItemHeight(mItemView *self, HITEM hItem);


DWORD getAddData(mItemView *self, HITEM hItem);

void setAddData(mItemView *self, HITEM hItem, DWORD addData);


void setImage(mItemView *self, HITEM hItem, DWORD image);

DWORD getImage(mItemView *self, HITEM hItem);


void setFlags(mItemView *self, HITEM hItem, DWORD flags);

DWORD getFlags(mItemView *self, HITEM hItem);


BOOL setText(mItemView *self, HITEM hItem, const char* text);

const char* getText(mItemView *self, HITEM hItem);

int getTextLen(mItemView *self, HITEM hItem);
```

- isEnabled：判断指定列表项是否处于使能状态。
- enable：使能或禁止指定列表项。
- isSelected：判断指定列表项是否处于选择状态。
- select：选择指定的列表项。
- deselect：取消选择指定的列表项。

- selectAll：选中所有列表项。
- deselectAll：取消所有列表项的选中状态。
- hilight：高亮选择指定的列表项。
- getHilight：获取当前高亮选择列表项。
- isHilight：判断指定列表项是否处于高亮状态。
- setItemHeight：设置列表项高度。
- getItemHeight：获取列表项高度。
- setAddData：设置列表项附加数据。
- getAddData：获取列表项附加数据。
- setImage：设置列表项位图信息。
- getImage：获取列表项位图信息。
- setFlags：设置列表项状态标志。
- getFlags：获取列表项状态标志。
- setText：设置列表项文本字符串。
- getText：获取列表项文本字符串。
- getTextLen：获取列表项文本字符串长度。

其它

```
int getFirstVisItem(mItemView *self);

void resetContent(mItemView *self);

int getRect(mItemView *self, HITEM hItem, RECT *rcItem, BOOL bConv);

int getCurSel(mItemView *self);

int setCurSel(mItemView *self, int newSel);

void refreshItem(mItemView *self, HITEM hItem, const RECT *rcInv);

BOOL showItemByIdx(mItemView *self, int index);

BOOL showItem(mItemView *self, HITEM hItem);
```

- getFirstVisItem：获取第一个可见的列表项。
- resetContent：删除所有列表项，并恢复所有信息配置到初始值。
- getRect：获取指定列表项的矩形区域，并通过最后参数决定是否将坐标转换为相对于屏幕的坐标。
- getCurSel：获取当前选中列表项索引。
- setCurSel：对通过索引设置指定列表项为选中项并将其显示出来，成功返回 0，失败返回-1。
- refreshItem：刷新指定列表项的指定区域，若指定区域为空，则刷新整个列表项。
- showItemByIdx：根据索引显示指定列表项。

- showItem：显示指定列表项。

mScrollView

控件窗口类

NCSCTRL_SCROLLVIEW

控件英文名

ScrollView

简介

用于显示和处理列表项，其中列表项的内容绘制完全是由应用程序自己确定。

示意图

mScrollView 风格

继承自 mItemView 风格

风格名	miniStudio 属性名	说明
NCSS_SCRLV_LOOP	-	条目可循环浏览
NCSS_SCRLV_SORT	-	条目自动排序

mScrollView 属性

继承自 mItemView 属性

mScrollView 事件

继承自 mItemView 事件

事件通知码	说明	参数
NCSN_SCRLV_CLICKED	鼠标点击事件	被点击的条目句柄
NCSN_SCRLV_SELCHANGING	正在改变选择的条目	处于高亮状态的条目句柄
NCSN_SCRLV_SELCHANGED	选择条目已改变	新的选择条目句柄

mScrollView 方法

继承自 mItemView 方法

mScrollView 继承自 mItemView，提供了控件自身的 addItem 方法，该方法通过列表项信息创建并插入列表项，同时返回插入位置给接口调用者。

```
typedef struct _NCS_SCRLV_ITEMINFO

{

        int     index;

        int     height;

        DWORD   addData;

}NCS_SCRLV_ITEMINFO;



HITEM addItem(mScrollView *self, NCS_SCRLV_ITEMINFO *info, int *pos);
```

在添加列表项内容前通过进行一些基础回调方法的设置等内容，如：

```


_c(scrlvObj)->freeze(scrlvObj, TRUE);

_c(scrlvObj)->setItemCmpFunc(scrlvObj, scrlv_cmp_item);

_c(scrlvObj)->setItemDraw(scrlvObj, scrlv_draw_item);



for (i = 0; i < TABLESIZE(people); i++) {

        info.height  = 32;

        info.index   = i;

        info.addData = (DWORD)people[i];

        _c(scrlvObj)->addItem(scrlvObj, &info, NULL);

}

_c(scrlvObj)->freeze(scrlvObj, FALSE);
```

mScrollView 实例

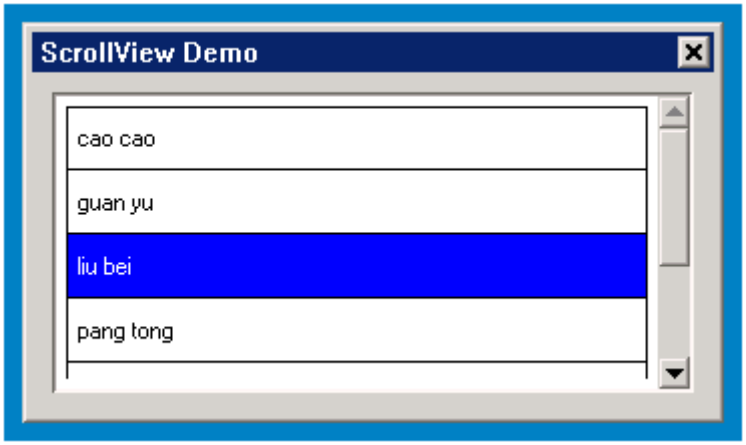


图 p2c6-1 scrollview 程序的输出

清单 p214-1 scrollview.c

```
/*
** $Id: scrollview.c 595 2009-10-10 08:19:47Z xwyan $
**
** Listing P2C14.1
**
** scrollview.c: Sample program for mGNCS Programming Guide
**      The demo application for ScrollView.
**
** Copyright (C) 2009 Feynman Software.
*/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
// START_OF_INCS
#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>

#include <mgncs/mgncs.h>
// END_OF_INCS

#define IDC_SCROLLVIEW  100

static const char *people[] =
{
        "cao cao",
        "sun quan",
        "liu bei",
        "zhu ge liang",
        "guan yu",
        "pang tong",
        "si ma yu",
};

static NCS_RDR_INFO rdr_info = {
        "classic","classic",NULL
```

```
};

// START_OF_HANDLERS
static void scrlv_notify (mWidget *self, int id, int nc, DWORD add_data)
{
        if (nc == NCSN_SCRLV_CLICKED)
        {
                if (self) {
                        const char* info;
                        mIconView *cls = (mIconView*)self;

                        info = (const char*)_c(cls)->getAddData(cls, (HITEM)add_data);

                        fprintf (stderr, "current item's data %s \n", info);
                }
        }
}

static NCS_EVENT_HANDLER scrlv_handlers[] = {
        NCS_MAP_NOTIFY(NCSN_SCRLV_CLICKED, scrlv_notify),
        {0, NULL }
};
// END_OF_HANDLERS
```

```
static NCS_WND_TEMPLATE _ctrl_tmpl[] = {
        {
                NCSCTRL_SCROLLVIEW,
                IDC_SCROLLVIEW,
                10, 10, 320, 150,
                WS_BORDER | WS_VISIBLE | NCSS_NOTIFY | NCSS_SCRLV_SORT,
                WS_EX_NONE,
                "",
                NULL,
                &rdr_info,
                scrlv_handlers,
                NULL,
                0,
                0
        },
};

static BOOL dialog_onKeyDown(mWidget* self,
int message, int code, DWORD key_status)
{
        if (message == MSG_KEYDOWN) {
                if (code == SCANCODE_REMOVE) {
                        mScrollView *scrlvObj;
                        int         curSel, count;
```

```
                        HITEM       delItem;

                        scrlvObj =
                        (mScrollView*)ncsObjFromHandle(GetDlgItem(self->hwnd,
IDC_SCROLLVIEW));
                        count = _c(scrlvObj)->getItemCount(scrlvObj);

                        if (scrlvObj) {
                                curSel = _c(scrlvObj)->getCurSel(scrlvObj);

                                if (curSel >= 0) {
                                        delItem = _c(scrlvObj)->getItem(scrlvObj, curSel);
                                        _c(scrlvObj)->removeItem(scrlvObj, delItem);
                                        if (curSel == count -1)
                                        curSel--;
                                        _c(scrlvObj)->setCurSel(scrlvObj, curSel);
                                }
                        }
                }
        }
        return FALSE;
}

static NCS_EVENT_HANDLER dialog_handlers[] = {
        {MSG_KEYDOWN, dialog_onKeyDown},
```

```
        {0, NULL }
};

static NCS_MNWND_TEMPLATE dialog_tmpl = {
        NCSCTRL_DIALOGBOX,
        7,
        0, 0, 350, 200,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "ScrollView Demo",
        NULL,
        &rdr_info,
        dialog_handlers,
        _ctrl_tmpl,
        sizeof(_ctrl_tmpl)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};

// START_OF_ITEMFUNCS
static int scrlv_cmp_item (mItemManager *manager, HITEM hItem1, HITEM hItem2)
{
        mScrollView *scrlvObj = (mScrollView*)manager->obj;
        const char *name1;
```

```
        const char *name2;

        if (scrlvObj) {
                name1 = (const char*)_c(scrlvObj)->getAddData(scrlvObj, hItem1);
                name2 = (const char*)_c(scrlvObj)->getAddData(scrlvObj, hItem2);
                return strcmp (name1, name2);
        }
        return 0;
}

static void scrlv_draw_item (mItemView *self, HITEM hItem, HDC hdc, RECT *rcDraw)
{
        const char  *name = (const char*)_c(self)->getAddData(self, hItem);
        gal_pixel   oldBrushClr = 0, oldTextClr = 0;
        BOOL        isHilite = FALSE;
        int         top;
        RECT        rcText;

        SetBkMode (hdc, BM_TRANSPARENT);

        top = rcDraw->top;
        if (_c(self)->indexOf(self, hItem) > 0) {
                top --;
        }
```

```
        if (_c(self)->isHilight(self, hItem)) {
                isHilite = TRUE;
                oldBrushClr = SetBrushColor (hdc, PIXEL_blue);
                FillBox (hdc, rcDraw->left + 1,
                top + 1, RECTWP(rcDraw) - 2, RECTHP(rcDraw) - 1);
                oldTextClr = SetTextColor (hdc, PIXEL_lightwhite);
        }

        Rectangle (hdc, rcDraw->left, top, rcDraw->right - 1, rcDraw->bottom - 1);

        CopyRect(&rcText, rcDraw);
        rcText.left += 5;
        DrawText(hdc, name, -1, &rcText, DT_VCENTER | DT_SINGLELINE);

        if (isHilite) {
                SetBrushColor (hdc, oldBrushClr);
                SetTextColor (hdc, oldTextClr);
        }
}
// END_OF_ITEMFUNCS

static BOOL scrlv_init(mDialogBox* self)
{
```

```
int     i;
HWND    scrlvWnd;
mScrollView *scrlvObj;
NCS_SCRLV_ITEMINFO info;

scrlvWnd = GetDlgItem (self->hwnd, IDC_SCROLLVIEW);
scrlvObj = (mScrollView*)ncsObjFromHandle(scrlvWnd);

if (!scrlvObj)
return FALSE;

// START_OF_ADDITEMS
_c(scrlvObj)->freeze(scrlvObj, TRUE);
_c(scrlvObj)->setItemCmpFunc(scrlvObj, scrlv_cmp_item);
_c(scrlvObj)->setItemDraw(scrlvObj, scrlv_draw_item);

for (i = 0; i < TABLESIZE(people); i++) {
        info.height  = 32;
        info.index   = i;
        info.addData = (DWORD)people[i];
        _c(scrlvObj)->addItem(scrlvObj, &info, NULL);
}
_c(scrlvObj)->freeze(scrlvObj, FALSE);
// END_OF_ADDITEMS
```

```
        return TRUE;
}

int MiniGUIMain(int argc, const char* argv[])
{
        ncsInitialize();
        mDialogBox* mydlg =
        (mDialogBox *)ncsCreateMainWindowIndirect (&dialog_tmpl, HWND_DESKTOP);

        scrlv_init(mydlg);
        _c(mydlg)->doModal(mydlg, TRUE);

        MainWindowThreadCleanup(mydlg->hwnd);
        ncsUninitialize();
        return 0;
}
```

mListBox

控件窗口类

NCSCTRL_LISTBOX

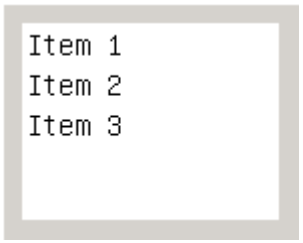
控件英文名

ListBox

简介

将用户提供的一系列可选项显示在可滚动的子窗口中，用户可通过键盘及鼠标操作来选中某一项或者多个项，选中的列表项通常高亮显示。列表框的最典型用法就是文件打开对话框。

示意图



mListBox 风格

继承自 mItemView 风格

风格名	miniStudio 属性名	说明
NCSS_LSTBOX_SINGLE	Multi->FALSE	单选列表项支持
NCSS_LSTBOX_MULTIPLE	Multi->TRUE	多选列表项支持
NCSS_LSTBOX_SORT	AutoSort	列表项排序支持
NCSS_LSTBOX_MOUSEFOLLOW	MouseFollow	列表项支持鼠标跟随
NCSS_LSTBOX_STRING	-	带有字符串的列表项
NCSS_LSTBOX_USEBITMAP	UseBitmap	列表项带有位图
NCSS_LSTBOX_CHECKBOX	Checkable	列表项包含 checkbox
NCSS_LSTBOX_AUTOCHECK	AutoCheck	列表项中的 checkbox 支持自动选择
NCSS_LSTBOX_AUTOCHECKBOX	-	同时包含 NCSS_LSTBOX_CHECKBOX 和 NCSS_LSTBOX_AUTOCHECK 2 种风格

mListBox 属性

继承自 mItemView 属性

属性 ID	miniStudio 名	类型	权限	说明
NCSP_LSTBOX_ITEMWIDTH	-	int	RO	列表项最大宽度
NCSP_LSTBOX_ITEMCOUNT	-	int	RO	列表项总数目
NCSP_LSTBOX_ITEMHEIGHT	-	int	RW	列表项高度
NCSP_LSTBOX_TOPITEM	-	int	RW	第一个可见列表项索引
NCSP_LSTBOX_HILIGHTEDITEM	-	int	RW	高亮列表项索引

mListBox 事件

继承自 mItemView 事件

事件通知码	说明	参数
NCSN_LSTBOX_CLICKED	鼠标点击事件	
NCSN_LSTBOX_SELCHANGED	选择条目已改变	新的选择条目句柄
NCSN_LSTBOX_ENTER	Enter 键被按下	-
NCSN_LSTBOX_SETFOCUS	获取焦点	-
NCSN_LSTBOX_KILLFOCUS	失去焦点	-
NCSN_LSTBOX_ERRSPACE	空间不足	-
NCSN_LSTBOX_DBCLK	双击列表项	-
NCSN_LSTBOX_SELCANCEL	取消选择条目	-
NCSN_LSTBOX_CLKCHKMARK	check mark 被点击	-

mListBox 方法

继承自 mItemView 方法

将字符串加入列表框

建立 listbox 控件之后，下一步是将字符串放入其中，这可以通过调用 addString 方法来完成，添加成功后该方法将返回列表项所在的索引值。字符串通常通过以 0 开始计数的索引数来引用，其中 0 对应于最顶上的条目。

```
int addString(mListBox *self, const char* string, DWORD addData);
```

也可以使用 insertString 指定一个索引值，将字符串插入到列表框中的指定位置，但在 NCSS_LISTBOX_SORT 风格下，会根据排序结果插入到相应位置，将忽略索引值设置：

```
int insertString(mListBox *self, const char* string, DWORD addData, int index);
```

如：

```


iteminfo.flag = NCSF_LSTBOX_CMBLANK;
```

```
iteminfo.image = 0;
for (i = 0; i < TABLESIZE(items); i++) {
        iteminfo.string = items[i];
        _c(lstboxObj)->addString (lstboxObj, &iteminfo);
}
```

如果添加的列表项除了包含字符串以外，还包含位图等信息，可通过 addItems 方法来完成。

```
typedef struct _NCS_LSTBOX_ITEMINFO
{
        char*   string;
        DWORD   flag;
        DWORD   image;
        DWORD addData;
}NCS_LSTBOX_ITEMINFO;

void addItems(mListBox *self, NCS_LSTBOX_ITEMINFO *info, int count);
```

如：

```
int count = 3;
mListItemInfo   lbii[count];

lbii[0].string = "test list";
lbii[0].flag = 0;
lbii[0].image = 0;
... ...
_c(listFile)->addItems(listFile, lbii, count);
```

删除列表框条目

列表框控件通过 delString 或 removeItemByIdx 方法可以删除指定索引值的列表项或通过 removeItem 方法删除指定列表项；同时也可以通过 resetContent 清空列表框中的所有内容。函数原型如下：

```
BOOL delString(mListBox *self, int index);

int removeItemByIdx(mListBox *self, int index);
```

如：

```


int sel     = _c(lstboxObj)->getCurSel(lstboxObj);

int count   = _c(lstboxObj)->getItemCount(lstboxObj);



if (sel >= 0) {

        _c(lstboxObj)->delString(lstboxObj, sel);



        if (sel == count -1)

        sel --;



        _c(lstboxObj)->setCurSel(lstboxObj, sel);

}
```

选择和获取条目

对于单项选择列表框和多项选择列表框在检索列表框条目的选中状态时需要使用不同的方法，下面先看单项选择列表框。选择条目除了通过鼠标和键盘操作外，还可以通过方法 setCurSel 来控制：

```
_c(listFile)->setCurSel(listFile, 1);
```

反之，可通过 getCurSel 来获取当前选定的索引项，如果没有选定项， 将返回 -1。

```
_c(listFile)->getCurSel(listFile);
```

另外也可以通过 *selectByIdx* 或 *deselectByIdx* 设置或取消选中状态。

```
int selectByIdx(mListBox *self, int index);

int deselectByIdx(mListBox *self, int index);
```

对于多项选择列表框来说，通过 setCurSel 和 getCurSel 方法只能用来设置和获取当前高亮项，无法获得所有具有选中状态的条目。但我们可以使用 setSel 来设定某特定条目的选择状态，而不影响其他项，其中 flag 的取值有三种含义：

- -1：时对指定列表项进行相反操作，即原状态未选中则选中它，否则取消选中状态。
- 0：取消对列表项的选中。
- 其它：选中列表项。

方法原型：

```
int setSel(mListBox *self, int newSel, int flag);
```

使用示例如：

```
_c(listFile)->setSel(listFile, 0, 1);
```

反之，我们可以用 isSelected 方法确定某特定条目的选择状态：

```
_c(listFile)->isSelected(listFile, 0);
```

另外对于多选列表框，可以通过 getSelectionCount 方法获取当前选中的条目个数，并通过 getSelection 获得所有被选中条目的索引值：

```
HITEM* selItems;

int selCount = _c(listFile)->getSelectionCount(listFile);

if (selCount == 0)

return;



selItem = alloca (sizeof(HITEM)*selCount);

_c(listFile)->getSelection(listFile, selItem, selCount);
```

查找含有字符串的条目

列表框通过 findString 方法提供了在指定范围内精确或模糊查找包含某一指定字符串的列表项的方法：

```
int findString(mListBox *self, int start, char* string, BOOL bExact);
```

下面的操作将从第 3 个列表项开始精确查找字符串为 test 的列表项。成功会返回查找到的列表项索引，否则返回 -1：

```
_c(listFile)->findString(listFile, 2, "test", TRUE);
```

设置和获取某条目的检查框的当前状态

```
_c(listFile)->getCheckMark(listFile, index);
```

返回由 index 指定索引处条目的检查框的状态。如果没有找到相应条目，将返回 -1。NCSF_LSTITEM_CMCHECKED 表示该条目的检查框处于选择状态。NCSF_LSTITEM_CMPARTCHECKED 表示该条目的检查框处于部分选择状态。NCSF_LSTITEM_CMBLANK 表示该条目的检查框处于未选择状态。

```
_c(listFile)->setCheckMark(listFile, index, (DWORD)status);
```

设置由 index 指定索引处条目的检查框的状态为 status 中指定的值。当没有找到 index 指定的条目时，返回 FALSE，成功返回 TRUE。

设置某列表框条目加粗显示状态

```
_c(listFile)->bold(listFile, index, TRUE);
```

该操作将指定索引项的内容进行加粗设置。

设置或获取某列表框条目选择状态

列表框通过 *isEnabled* 方法确定某特定条目是否处于禁止选中状态：

```
_c(listFile)->isEnabled(listFile, 0);
```

通过 *enableByIdx* 或 *enable* 方法可以支持条目选中或禁止条目选中。

```
_c(listFile)->enableByIdx(listFile, 0, TRUE);

or

_c(listFile)->enableByIdx(listFile, 0, FALSE);
```

设置字符串比较函数

列表框控件通过 *setStrCmpFunc* 方法来使用用户设置的排序方法排列列表项。

```
static int my_strcmp (const char* s1, const char* s2, size_t n)

{

        int i1 = atoi (s1);

        int i2 = atoi (s2);

        return (i1 - i2);

}

_c(listFile)->setStrCmpFunc(listFile, my_strcmp);
```

mListBox 实例

图 p2c6-1 listbox 程序的输出

清单 p2c14-3 listbox.c

```
/**

* $Id: listbox.c 595 2009-10-10 08:19:47Z xwyan $
```

```
*
* Listing P2C14.2
*
* listbox.c: Sample program for mGNCS Programming Guide
*      The demo application for ListBox.
*
* Copyright (C) 2009 Feynman Software.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// START_OF_INCS
#include <minigui/common.h>
#include <minigui/minigui.h>
#include <minigui/gdi.h>
#include <minigui/window.h>

#include <mgncs/mgncs.h>
// END_OF_INCS

#define IDC_LIST    100
#define IDC_DELETE  200
```

```
static char* items[] = {
        "Apples",
        "Apricots",
        "Bananas",
        "Grapefruit",
        "Kiwi",
        "Oranges",
        "Peaches",
        "The Longest String"
};

static void lstbox_init(mDialogBox *dialog)
{
        NCS_LSTBOX_ITEMINFO iteminfo;
        mListBox *lstboxObj;
        int i;

        lstboxObj = (mListBox *)ncsGetChildObj(dialog->hwnd, IDC_LIST);

        // START_OF_ADDITEMS
        iteminfo.flag = NCSF_LSTBOX_CMBLANK;
        iteminfo.image = 0;
        for (i = 0; i < TABLESIZE(items); i++) {
```

```
            iteminfo.string = items[i];
            _c(lstboxObj)->addString (lstboxObj, &iteminfo);
        }
        // END_OF_ADDITEMS
}

// START_OF_BTNHANDLERS
static void btn_notify(mWidget *self, int id, int nc, DWORD add_data)
{
        mListBox    *lstboxObj =
        (mListBox *)ncsGetChildObj(GetParent(self->hwnd), IDC_LIST);
        // START_OF_DELITEMS
        int sel     = _c(lstboxObj)->getCurSel(lstboxObj);
        int count   = _c(lstboxObj)->getItemCount(lstboxObj);

        if (sel >= 0) {
            _c(lstboxObj)->delString(lstboxObj, sel);

            if (sel == count -1)
            sel --;

            _c(lstboxObj)->setCurSel(lstboxObj, sel);
        }
        // END_OF_DELITEMS
```

```
}

static NCS_EVENT_HANDLER btn_handlers [] = {
        NCS_MAP_NOTIFY(NCSN_BUTTON_PUSHED, btn_notify),
        {0, NULL}
};
// END_OF_BTNHANDLERS
static NCS_WND_TEMPLATE _ctrl_tmpl[] = {
        {
                NCSCTRL_LISTBOX,
                IDC_LIST,
                20, 15, 170, 200,
                WS_BORDER | WS_VISIBLE | NCSS_NOTIFY,
                WS_EX_NONE,
                "",
                NULL,
                NULL,
                NULL,
                NULL,
                0,
                0
        },
        {
                NCSCTRL_BUTTON,
```

```
            IDC_DELETE,
            15, 230, 80, 30,
            WS_VISIBLE | WS_TABSTOP,
            WS_EX_NONE,
            "Delete",
            NULL,
            NULL,
            btn_handlers,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_BUTTON,
            IDCANCEL,
            115, 230, 80, 30,
            WS_VISIBLE | WS_TABSTOP,
            WS_EX_NONE,
            "Cancel",
            NULL,
            NULL,
            NULL,
            NULL,
            0,
```

```
                0
        },
};


static NCS_MNWND_TEMPLATE mainwnd_tmpl = {
        NCSCTRL_DIALOGBOX,
        1,
        100, 100, 220, 300,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "ListBox Demo",
        NULL,
        NULL,
        NULL,
        _ctrl_tmpl,
        sizeof(_ctrl_tmpl)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};

int MiniGUIMain(int argc, const char* argv[])
{
        ncsInitialize();
```

```
        mDialogBox* dialog =
        (mDialogBox *)ncsCreateMainWindowIndirect (&mainwnd_tmpl, HWND_DESKTOP);

        lstbox_init(dialog);
        _c(dialog)->doModal(dialog, TRUE);

        MainWindowThreadCleanup(dialog->hwnd);
        ncsUninitialize();
        return 0;
}
```

mIconView

控件窗口类

NCSCTRL_ICONVIEW

控件英文名

IconView

简介

将用户提供的一系列可选项以图标加标签文字的方式供其浏览，用户可通过键盘及鼠标操作来选中某一项或者多个项，选中的列表项通常高亮显示。标型控件的典型用法是作为桌面图标的容器和目录下文件的显示。

示意图

mIconView 风格

继承自 mItemView 风格

风格名	miniStudio 属性名	说明
NCSS_ICONV_LOOP	Loop	条目可循环浏览
NCSS_ICONV_SORT	AutoSort	条目自动排序

mIconView 属性

继承自 mItemView 属性

属性 ID	miniStudio 名	类型	权限	说明
NCSP_ICONV_DEFICONHEIGHT	-	int	RW	列表项高度
NCSP_ICONV_DEFICONWIDTH	-	int	RW	列表项宽度

mIconView 方法

继承自 mItemView 方法

iconview 控件通过 setIconSize 方法来初始化列表项的大小，并通过 addItem 方法根据列表项信息添加列表项。

```
typedef struct _NCS_ICONV_ITEMINFO
{
        int index;
        PBITMAP bmp;
        const char *label;
        DWORD addData;
}NCS_ICONV_ITEMINFO;

void setIconSize(mIconView *self, int width, int height);
HITEM addItem(mIconView *self, NCS_ICONV_ITEMINFO *info, int *pos);
```

添加列表项的示例代码如：

```
_c(iconvObj)->setIconSize(iconvObj, 90, 80);

for(i = 0; i < TABLESIZE(icon_demos); i++)
{
        pos = 0;
        memset (&info, 0, sizeof(NCS_ICONV_ITEMINFO));
        info.bmp = &icon_demos[i];
        info.index = TABLESIZE(icon_demos) * j + i;
        info.label = iconlabels[i];
        info.addData = (DWORD)iconlabels[i];
        _c(iconvObj)->addItem(iconvObj, &info, &pos);
}
_c(iconvObj)->setCurSel(iconvObj, 0);
```

mIconView 实例

图 p2c6-1 iconview 程序的输出

清单 p2c14-3 iconview.c

```
/**
* $Id: iconview.c 595 2009-10-10 08:19:47Z xwyan $
*
* Listing P2C14.3
*
* iconview.c: Sample program for mGNCS Programming Guide
*      The demo application for IconView.
*
* Copyright (C) 2009 Feynman Software.
*/
```

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// START_OF_INCS

#include <minigui/common.h>

#include <minigui/minigui.h>

#include <minigui/gdi.h>

#include <minigui/window.h>


#include <mgncs/mgncs.h>

// END_OF_INCS


#define IDC_ICONVIEW    100

#define IDC_ADD         600

#define IDC_DELETE      601


static BITMAP icon_demos [12];


static const char* iconfiles[12] =

{

        "./res/acroread.png",

        "./res/icons.png",

        "./res/looknfeel.png",
```

```
        "./res/package_games.png",
        "./res/tux.png",
        "./res/xemacs.png",
        "./res/gimp.png",
        "./res/kpilot.png",
        "./res/multimedia.png",
        "./res/realplayer.png",
        "./res/usb.png",
        "./res/xmms.png"
};

static const char *iconlabels[12] =
{
        "acroread",
        "icons",
        "looknfeel",
        "games",
        "tux",
        "xemacs",
        "gimp",
        "kpilot",
        "multimedia",
        "realplayer",
        "usb",
```

```
        "xmms"
};

static BOOL iconv_init(mDialogBox* self)
{
        NCS_ICONV_ITEMINFO info;
        static int i = 0, j = 0, pos = 0;
        mIconView *iconvObj;
        HWND iconvWnd;

        for(i = 0; i < TABLESIZE(icon_demos); i++)
        {
                LoadBitmap (HDC_SCREEN, &icon_demos[i], iconfiles[i]);
        }

        iconvWnd = GetDlgItem (self->hwnd, IDC_ICONVIEW);
        iconvObj = (mIconView*)ncsObjFromHandle(iconvWnd);

        if (!iconvObj)
        return FALSE;

        // START_OF_ADDITEMS
        _c(iconvObj)->setIconSize(iconvObj, 90, 80);
```

```
        for(i = 0; i < TABLESIZE(icon_demos); i++)
        {
                pos = 0;
                memset (&info, 0, sizeof(NCS_ICONV_ITEMINFO));
                info.bmp = &icon_demos[i];
                info.index = TABLESIZE(icon_demos) * j + i;
                info.label = iconlabels[i];
                info.addData = (DWORD)iconlabels[i];
                _c(iconvObj)->addItem(iconvObj, &info, &pos);
        }
        _c(iconvObj)->setCurSel(iconvObj, 0);
        // END_OF_ADDITEMS

        return TRUE;
}

// START_OF_WNDHANDLERS
static BOOL mainwnd_onKeyDown(mWidget* self,
int message, int code, DWORD key_status)
{
        if (message == MSG_KEYDOWN) {
                if (code == SCANCODE_REMOVE) {
                        mIconView *iconView;
                        int curSel, count;
```

```
                        HITEM delItem;

                        iconView =
                        (mIconView*)ncsObjFromHandle(GetDlgItem (self->hwnd, IDC_ICONVIEW));
                        count = _c(iconView)->getItemCount(iconView);

                        if (iconView) {
                                curSel = _c(iconView)->getCurSel(iconView);

                                if (curSel >= 0) {
                                        delItem = _c(iconView)->getItem(iconView, curSel);
                                        _c(iconView)->removeItem(iconView, delItem);
                                        if (curSel == count -1)
                                        curSel--;
                                        _c(iconView)->setCurSel(iconView, curSel);
                                }
                        }
                }
        }
        return FALSE;
}

static NCS_EVENT_HANDLER mainwnd_handlers[] = {
        {MSG_KEYDOWN, mainwnd_onKeyDown},
```

```
        {0, NULL }
};
// END_OF_WNDHANDLERS

// START_OF_ICONVHANDLERS
static void iconv_notify (mWidget *self, int id, int nc, DWORD add_data)
{
        if (nc == NCSN_ICONV_CLICKED)
        {
                if (self) {
                        int idx;
                        const char  *text;
                        mIconView   *iconvObj = (mIconView*)self;

                        idx = _c(iconvObj)->indexOf(iconvObj, (HITEM)add_data);
                        text = _c(iconvObj)->getText(iconvObj, (HITEM)add_data);
                        fprintf (stderr, "click icon[%d], text is %s \n", idx, text);
                }
        }
}

static NCS_EVENT_HANDLER iconv_handlers[] = {
        NCS_MAP_NOTIFY(NCSN_ICONV_CLICKED, iconv_notify),
        NCS_MAP_NOTIFY(NCSN_ICONV_SELCHANGED, iconv_notify),
```

```
        {0, NULL }
};
// END_OF_ICONVHANDLERS

// START_OF_BTNHANDLERS
static void btn_notify(mWidget *self, int id, int nc, DWORD add_data)
{
        mIconView *iconvObj =
        (mIconView *)ncsGetChildObj(GetParent(self->hwnd), IDC_ICONVIEW);

        if (!iconvObj)
        return;

        switch (id)
        {
                case IDC_ADD:
                {
                        char    buff[12];
                        int     count, pos = 0;
                        NCS_ICONV_ITEMINFO info;

                        count = _c(iconvObj)->getItemCount(iconvObj);
                        sprintf (buff, "icon%i", count+1);
```

```
            memset (&info, 0, sizeof(NCS_ICONV_ITEMINFO));
            info.bmp = &icon_demos[0];
            info.index = count;
            info.label = buff;
            info.addData = (DWORD)"icon";

            if (_c(iconvObj)->addItem(iconvObj, &info, &pos))
            _c(iconvObj)->setCurSel(iconvObj, pos);

            break;
        }

        case IDC_DELETE:
        {
            int     count, sel;
            char    *label = NULL;
            HITEM   hItem;

            sel     = _c(iconvObj)->getCurSel(iconvObj);
            count   = _c(iconvObj)->getItemCount(iconvObj);
            hItem   = _c(iconvObj)->getItem(iconvObj, sel);

            if (sel >= 0) {
                label = (char*)_c(iconvObj)->getAddData(iconvObj, hItem);
```

```
                                _c(iconvObj)->removeItem(iconvObj, hItem);

                                if (sel == count -1)
                                sel --;

                                _c(iconvObj)->setCurSel(iconvObj, sel);
                        }
                        break;
                }
        }
}

static NCS_EVENT_HANDLER btn_handlers [] = {
        NCS_MAP_NOTIFY(NCSN_BUTTON_PUSHED, btn_notify),
        {0, NULL}
};
// END_OF_BTNHANDLERS

static NCS_RDR_INFO iconv_rdr_info = {
        "classic", "classic", NULL
};

static NCS_WND_TEMPLATE _ctrl_tmpl[] = {
        {
```

```
        NCSCTRL_ICONVIEW,
        IDC_ICONVIEW,
        15, 10, 220, 250,
        WS_BORDER | WS_CHILD | WS_VISIBLE | NCSS_NOTIFY | NCSS_ICONV_LOOP,
        WS_EX_NONE,
        "",
        NULL,
        &iconv_rdr_info,
        iconv_handlers,
        NULL,
        0,
        0
    },
    {
        NCSCTRL_BUTTON,
        IDC_ADD,
        15, 280, 80, 30,
        WS_VISIBLE | NCSS_NOTIFY,
        WS_EX_NONE,
        "add",
        NULL,
        NULL,
        btn_handlers,
        NULL,
```

```
                0,
                0
        },
        {
                NCSCTRL_BUTTON,
                IDC_DELETE,
                155, 280, 80, 30,
                WS_VISIBLE | NCSS_NOTIFY,
                WS_EX_NONE,
                "delete",
                NULL,
                NULL,
                btn_handlers,
                NULL,
                0,
                0
        },
};

static NCS_MNWND_TEMPLATE mainwnd_tmpl = {
        NCSCTRL_DIALOGBOX,
        7,
        0, 0, 260, 350,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
```

```
        WS_EX_NONE,
        "IconView Demo",
        NULL,
        NULL,
        mainwnd_handlers,
        _ctrl_tmpl,
        sizeof(_ctrl_tmpl)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};

int MiniGUIMain(int argc, const char* argv[])
{
        ncsInitialize();
        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mainwnd_tmpl, HWND_DESKTOP);

        iconv_init(mydlg);
        _c(mydlg)->doModal(mydlg, TRUE);

        MainWindowThreadCleanup(mydlg->hwnd);
        ncsUninitialize();
        return 0;
}
```

mListView

控件窗口类

NCSCTRL_LISTVIEW

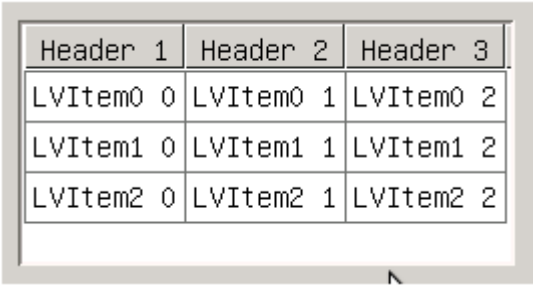
控件英文名

ListView

简介

以列表的方式显示一系列的数据项（列表项），不同列表项的相同类型子项以列的方式组织，列表型控件的表头（header）内容通常反映了列表项不同子项的意义。列表型控件经常用来作为文件浏览框，它可以在一个区域内显示包括文件名、文件类型、大小和修改日期在内的诸多文件属性。

示意图



mListView 风格

继承自 mItemView 风格

风格名	miniStudio 属性名	说明
NCSS_LISTV_NOTIFY	Notify	通知事件支持
NCSS_LISTV_LOOP	Loop	循环浏览支持
NCSS_LISTV_MULTIPLE	Multi->TRUE	多选支持
NCSS_LISTV_SINGLE		Multi->FALSE 单选支持
NCSS_LISTV_CHECKBOX	CheckBox	列表项包含 checkbox
NCSS_LISTV_AUTOCHECK	AutoCheck	列表项中的 checkbox 支持自动选择
NCSS_LISTV_AUTOCHECKBOX	-	同时包含 NCSS_LSTBOX_CHECKBOX 和 NCSS_LSTBOX_AUTOCHECK 2 种风格
NCSS_LISTV_TREE	Tree	支持树状列表
NCSS_LISTV_WITHICON	WithIcon	

NCSS_LISTV_SORT	Sort	排序支持

mListView 属性

继承自 mItemView 的属性

属性 ID	miniStudio 名	类型	权限	说明
NCSP_LISTV_DEFITEMHEIGHT	-	int	RW	列表项默认高度
NCSP_LISTV_ROWCOUNT	-	RO	int	列表项行数
NCSP_LISTV_HDRHEIGHT	HeadHeight	int	RW	列表头高度
NCSP_LISTV_HDRWIDTH	HeadWidth	int	RW	列表头总宽度
NCSP_LISTV_HDRVISIBLE	-	BOOL	RW	列表头是否可见
NCSP_LISTV_SORTCOLUMN	-	int	RW	排序列索引
NCSP_LISTV_GRIDLINEWIDTH	GridLineWidth	int	RW	网格宽度
NCSP_LISTV_GRIDLINECOLOR	GridLineColor	int	RW	网格颜色
NCSP_LISTV_COLCOUNT	-	int	RO	列表项列数

mListView 事件

继承自 mItemView 事件

事件通知码	说明	参数
NCSN_LISTV_CLICKED	鼠标点击事件	
NCSN_LISTV_SELCHANGED	选择条目已改变	新的选择项句柄
NCSN_LISTV_ITEMRDOWN	鼠标右键在列表项上按下	被点击的行索引
NCSN_LISTV_ITEMRUP	鼠标右键在列表项上弹起	被点击的行索引
NCSN_LISTV_HDRRDOWN	鼠标右键在表头上按下	被点击的列索引
NCSN_LISTV_HDRRUP	鼠标右键在列表项上弹起	被点击的列索引
NCSN_LISTV_ITEMDBCLK	双击列表项	-
NCSN_LISTV_FOLDITEM	树列表项折叠子节点	被点击的列表项句柄
NCSN_LISTV_UNFOLDITEM	树列表项打开子节点	被点击的列表项句柄

mListView 方法

继承自 mItemView 风格

列操作

在向该控件中添加列表项前，首先需要通过 addColumn 方法添加列：

```
for (i = 0; i < COL_NR; i++) {
        lstv_clminfo.index  = i;
        lstv_clminfo.text   = caption[i];
        lstv_clminfo.width  = 74;
        lstv_clminfo.pfnCmp = NULL;
        lstv_clminfo.flags  = NCSF_LSTCLM_CENTERALIGN | NCSF_LSTHDR_CENTERALIGN;
        _c(lstvObj)->addColumn(lstvObj, &lstv_clminfo);
}
```

其中 lstv_clminfo 是一个 NCS_LISTV_CLMINFO 结构，它包含了列表型控件的列信息。在添加列后，若需要设置或获取列的相关信息时，可通过如下方法来完成：

```
void setColumnWidth(mListView *self, int index, int width);
BOOL setHeadText(mListView *self, int col, const char* text);

mListColumn* getColumn(mListView *self, int index);
int getColumnIndex(mListView *self, mListColumn *column);
int getColumnWidth(mListView *self, int index);
int getColumnCount(mListView *self);
void showColumn(mListView *self, mListColumn *column);
```

此外，删除指定列可通过 delColumn 方法实现：

```
BOOL delColumn(mListView *self, int index);
```

列表项操作

列表型控件由许多纵向排列的列表项组成，每个列表项由列分为多个子项，列表项可以包含特定的应用程序定义的附加数据。应用程序可以通过相应的方法来添加、修改和设置、删除列表项或获取列表项的属性信息。

控件创建并添加列后，还没有列表项，此时需要通过 *addItem* 向其添加列表项：

```
NCS_LISTV_ITEMDATA subdata;
HITEM   hItem;

subdata.row = info->index;
subdata.col = 0;
subdata.text= classes[info->index];
subdata.textColor = 0;
subdata.flags = 0;
subdata.image = 0;

info->dataSize = 1;
info->data = &subdata;

hItem = _c(self)->addItem (self, info);
```

每个列表项包括一个或多个子项，子项的数目和列表型控件的列数相同。一个子项中包括字符串和图像，可以使用下列方法来获取和设置子项的信息：

```
void setBackground(mListView *self,int row,int col,int *color);
```

```
void setForeground(mListView *self,int row,int col,int *color);

int getBackground(mListView *self,int row,int col,int *color);

int getForeground(mListView *self,int row,int col,int *color);


BOOL setItemInfo(mListView *self, NCS_LISTV_ITEMDATA *info);

BOOL getItemInfo(mListView *self, NCS_LISTV_ITEMDATA *info);

const char* getItemText(mListView *self, int row, int col);

int getItemTextLen(mListView *self, int row, int col);

BOOL setItemText(mListView *self, int row, int col, const char* str);
```

查找列表项

findItem 用于在列表型控件中查找一个特定的列表项。如果查找成功的话，返回列表项句柄。

```
HITEM findItem(mListView *self, NCS_LISTV_FINDINFO *info);
```

比较和排序

除了继承自基类的排序功能支持外，该控件还可以指定某一列作为其排序的依照列，同时还可以设定排序类型是升序还是降序，或无序（不排列）。

```
void sort(mListView *self, NCS_CB_LISTV_CMPCLM func, int col, ncsLstClmSortType sort);

void setSortDirection(mListView *self, ncsLstClmSortType direction);

ncsLstClmSortType getSortDirection(mListView *self);

mListColumn* getSortColumn(mListView *self);

void setSortColumn(mListView *self, mListColumn* column);
```

回调方法

对列表头的绘制包括了背景色和内容绘制 2 部分，控件对此提供了回调方法供上层应用处理表头的绘制。方法：

```
void setCustomDrawHeader(mListView *self, NCS_CB_LISTV_CSTMHDROPS *func);
```

树型节点的操作

树型节点的操作包括获取相关节点和折叠一个节点。 相关方法有：

```
HITEM getRelatedItem(mListView *self,HITEM hItem,ncsListVIRType type);

HITEM getChildItem(mListView *self, HITEM parent, int index);

int getChildCount(mListView *self, HITEM hItem);

int foldItem(mListView *self, HITEM hItem, BOOL fold);
```

- getRelatedItem 用于获取一个节点的相关树型节点，如父节点，兄弟节点或第一个子节点。
  ncsListVIRType 指定相关节点和目标节点的关系，包括：
  - NCSID_LISTV_IR_PARENT：父节点
  - NCSID_LISTV_IR_FIRSTCHILD：第一个子节点
  - NCSID_LISTV_IR_LASTCHILD：最后一个节点
  - NCSID_LISTV_IR_NEXTSIBLING：下一个兄弟节点
  - NCSID_LISTV_IR_PREVSIBLING：上一个兄弟节点
- foldItem 用来折叠或者展开一个包含子节点的节点项。
- getChildItem：用于获取指定父节点下的子节点。
- getChildCount：用于获取指定节点的子节点数目。

mListView 实例

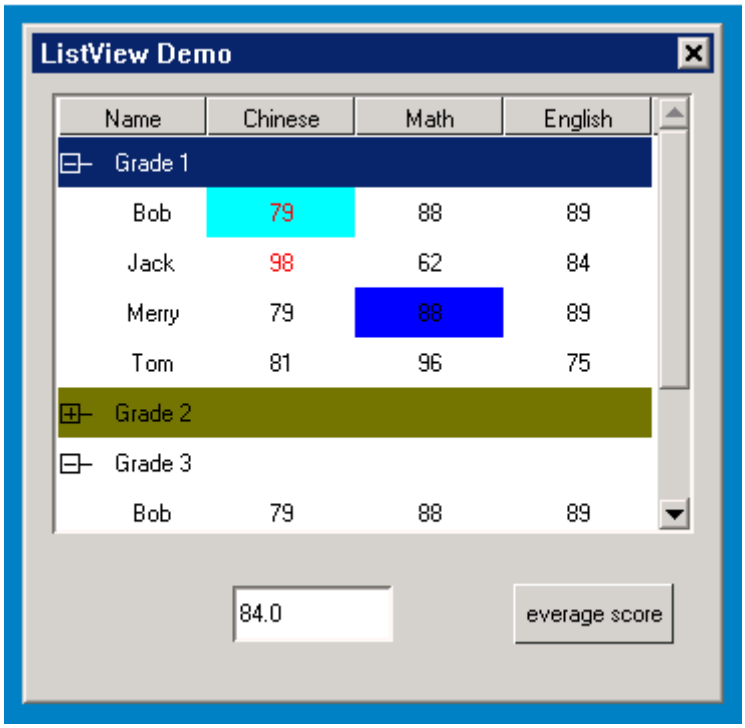


图 p2c6-1 listview 程序的输出

清单 p2c14-4 listview.c

```
/**
* $Id: listview.c 595 2009-10-10 08:19:47Z xwyan $
*
* Listing P2C14.4
*
* listview.c: Sample program for mGNCS Programming Guide
*      The demo application for ListView.
*
* Copyright (C) 2009 Feynman Software.
*/
```

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// START_OF_INCS

#include <minigui/common.h>

#include <minigui/minigui.h>

#include <minigui/gdi.h>

#include <minigui/window.h>


#include <mgncs/mgncs.h>

// END_OF_INCS


#define IDC_LISTVIEW    100

#define IDC_BTN1        101

#define IDC_SLEDIT      102


#define COL_NR       TABLESIZE(caption)

#define SCORE_NUM    TABLESIZE(scores)

#define CLASS_NUM    TABLESIZE(classes)

#define SUB_NUM      3


typedef struct _SCORE

{
```

```
        char *name;
        int scr[SUB_NUM];
} SCORE;
static char * caption [] =
{
        "Name", "Chinese", "Math", "English"
};

static char *classes [] =
{
        "Grade 1", "Grade 3", "Grade 2"
};

static SCORE scores[] =
{
        {"Tom",     {81, 96, 75}},
        {"Jack",    {98, 62, 84}},
        {"Merry",   {79, 88, 89}},
        {"Bob",     {79, 88, 89}},
};

static NCS_RDR_INFO rdr_info = {
        "classic","classic",NULL
};
```

```
static void btn_notify(mWidget *button, int id, int nc, DWORD add_data)
{
        mListView *lstvObj;
        mSlEdit *sleObj;
        HITEM   gradeItem, hItem;
        int     i, j, score;
        float   average = 0;
        char    buff[20];

        lstvObj = (mListView *)ncsGetChildObj(GetParent(button->hwnd), IDC_LISTVIEW);
        sleObj    = (mSlEdit *)ncsGetChildObj(GetParent(button->hwnd), IDC_SLEDIT);
        if (!lstvObj)
        return;



        gradeItem = _c(lstvObj)->getChildItem(lstvObj, 0, 0);
        for (i = 0; i < SCORE_NUM; i++) {
                hItem = _c(lstvObj)->getChildItem(lstvObj, gradeItem, i);

                for (j = 0; j < SUB_NUM; j++) {
                        sscanf(_c(lstvObj)->getItemText(lstvObj, hItem, 0, j+1), "%d", &score);
                        average += score;
                }
```

```
        }
        average = average / (SCORE_NUM * SUB_NUM);

        sprintf (buff, "%4.1f", average);

        _c(sleObj)->setContent(sleObj, buff, 0, strlen(buff));
}

static NCS_EVENT_HANDLER btn_handlers [] = {
        NCS_MAP_NOTIFY(NCSN_WIDGET_CLICKED, btn_notify),
        {0, NULL}
};

static NCS_WND_TEMPLATE _ctrl_tmpl[] = {
        {
                NCSCTRL_LISTVIEW,
                IDC_LISTVIEW,
                10, 10, 320, 220,
                WS_BORDER | WS_VISIBLE | NCSS_LISTV_SORT | NCSS_LISTV_LOOP,
                WS_EX_NONE,
                "score table",
                NULL,
                &rdr_info,
                NULL,
```

```
            NULL,
            0,
            0
        },
        {
            NCSCTRL_BUTTON,
            IDC_BTN1,
            240, 255, 80, 30,
            WS_VISIBLE | NCSS_NOTIFY,
            WS_EX_NONE,
            "everage score",
            NULL,
            NULL,
            btn_handlers,
            NULL,
            0,
            0
        },
        {
            NCSCTRL_SLEDIT,
            IDC_SLEDIT,
            100, 256, 80, 28,
            WS_BORDER | WS_VISIBLE,
            WS_EX_NONE,
```

```
                "",
                NULL,
                NULL,
                NULL,
                NULL,
                0,
                0
        },
};

static NCS_EVENT_HANDLER mainwnd_handlers[] = {
        {0, NULL}
};

static NCS_MNWND_TEMPLATE mainwnd_tmpl = {
        NCSCTRL_DIALOGBOX,
        7,
        0, 0, 350, 340,
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "ListView Demo",
        NULL,
        &rdr_info,
        mainwnd_handlers,
```

```
        _ctrl_tmpl,
        sizeof(_ctrl_tmpl)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};

static HITEM add_class_item (mListView *self, NCS_LISTV_ITEMINFO *info)
{
        // START_OF_ADDITEMS
        NCS_LISTV_ITEMDATA subdata;
        HITEM   hItem;

        subdata.row = info->index;
        subdata.col = 0;
        subdata.text= classes[info->index];
        subdata.textColor = 0;
        subdata.flags = 0;
        subdata.image = 0;

        info->dataSize = 1;
        info->data = &subdata;

        hItem = _c(self)->addItem (self, info);
        // END_OF_ADDITEMS
```

```
        return hItem;
}

static HITEM add_score_item (mListView *self, NCS_LISTV_ITEMINFO *info)
{
        char    buff[3][20];
        HITEM   hItem;
        int     i = info->index, j;

        // START_OF_ADDSUBITEMS
        NCS_LISTV_ITEMDATA subdata[4];

        for (j = 0; j < SCORE_NUM; j ++) {
                subdata[j].flags = 0;
                subdata[j].image = 0;
                subdata[j].row = info->index;
                subdata[j].col = j;
                if (j == 0) {
                        subdata[j].text = scores[i].name;
                        subdata[j].textColor = 0;
                }
                else {
                        sprintf (buff[j-1], "%d", scores[i].scr[j-1]);
```

```
                        subdata[j].text = buff[j-1];
                        if (scores[i].scr[j-1] > 90)
                        subdata[j].textColor = 0x0000FF;
                        else
                        subdata[j].textColor = 0;
                }
        }

        info->dataSize = SCORE_NUM;
        info->data = subdata;

        hItem = _c(self)->addItem (self, info);
        // END_OF_ADDSUBITEMS

        if (!hItem)
        return 0;

        return hItem;
}

static BOOL lstv_init(mDialogBox* self)
{
        int     i, j;
        int     color;
```

```
HITEM   hItem = 0, subItem;
HWND    lstvWnd = GetDlgItem (self->hwnd, IDC_LISTVIEW);
mListView *lstvObj;
NCS_LISTV_ITEMINFO  info;
NCS_LISTV_CLMINFO   lstv_clminfo;

lstvObj = (mListView*)ncsObjFromHandle(lstvWnd);

if (!lstvObj)
return FALSE;

_c(lstvObj)->freeze(lstvObj, TRUE);
//add column

// START_OF_ADDCLMS
for (i = 0; i < COL_NR; i++) {
        lstv_clminfo.index  = i;
        lstv_clminfo.text   = caption[i];
        lstv_clminfo.width  = 74;
        lstv_clminfo.pfnCmp = NULL;
        lstv_clminfo.flags  = NCSF_LSTCLM_CENTERALIGN | NCSF_LSTHDR_CENTERALIGN;
        _c(lstvObj)->addColumn(lstvObj, &lstv_clminfo);
}
// END_OF_ADDCLMS
```

```
info.height     = 25;
info.flags      = 0;
info.foldIcon   = 0;
info.unfoldIcon = 0;
for (i = 0; i < CLASS_NUM; i++) {
        info.parent = 0;
        info.index = i;
        hItem = add_class_item (lstvObj, &info);

        for (j = 0; j < SCORE_NUM; j++) {
                info.parent = hItem;
                info.index = j;
                subItem = add_score_item (lstvObj, &info);
        }

}

// START_OF_SETBGCLR
color = 0xFFFF00;
_c(lstvObj)->setBackground(lstvObj, 1, 1, &color);
color = 0xFF0000;
_c(lstvObj)->setBackground(lstvObj, 3, 2, &color);
color = 0x007777;
```

```
        _c(lstvObj)->setBackground(lstvObj, 5, -1, &color);
        // END_OF_SETBGCLR

        _c(lstvObj)->freeze(lstvObj, FALSE);

        return TRUE;
}

int MiniGUIMain(int argc, const char* argv[])
{
        ncsInitialize();
        mDialogBox* dialog =
        (mDialogBox *)ncsCreateMainWindowIndirect (&mainwnd_tmpl, HWND_DESKTOP);

        lstv_init(dialog);
        _c(dialog)->doModal(dialog, TRUE);

        MainWindowThreadCleanup(dialog->hwnd);
        ncsUninitialize();

        return 0;
}
```

第二部分第十五章 不可见控件

不可见控件简介

不可见控件是指那些不能在窗口上显示，但是却具有一定功能的组件。

这些组件是对一些功能模块的封装，保证它们能够像控件一样在 miniStudio 中被编辑

- mObject
  - mComponent
    - mInvsbComp

mInvsbComp

控件名称

无

英文名

Invisble Component

简要介绍

不可见组件的基类

示意图

基础类，不能直接使用

mInvsbComp 风格

继承自 mComponent 的风格

mInvsbComp 属性

继承自 mComponent 的属性

mInvsbComp 方法

mInvsbComp 提供了以下方法的实现：

- setId
- getId
- setReleated

- getReleated
- getChild

另外，为方便 mInvsbComp 的使用，提供了如下函数

- 创建不可见组件

```
/**
* \fn mInvsbComp * ncsCreateInvsbComp(const char* class_name, \
mComponent* parent, \
int id, NCS_PROP_ENTRY *props, \
NCS_EVENT_HANDLER * handlers, \
DWORD user_data);
* \brief create an Invisible Component
*
* \param class_name  the class name of Invisible Component
* \param parent the parent of creating Invisible Component
* \param id  the id of Invisible Component
* \param props the properties array of Invisible Component
* \param handlers the event handler array of Invisible Component
* \param user_data user data
*
* \return mInvsbComp * - the new created Invisible Component pointer, NULL or failed
*
* \sa NCS_INVSB_CREATE_INFO, ncsCreateInvsbCompIndirect
*/
mInvsbComp * ncsCreateInvsbComp(const char* class_name, \
```

```
mComponent* parent, \
int id, \
NCS_PROP_ENTRY *props, \
NCS_EVENT_HANDLER * handlers, \
DWORD user_data);

/**
* \fn mInvsbComp * ncsCreateInvsbCompIndirect(const char* class_name, \
NCS_INVSB_CREATE_INFO *create_info);
* \brief create an Invisible Component from creating info
*
* \param class_name the class name of Invisible Component
* \param create_info the creating information pointer
*
* \return mInvsbComp * - the Invisible Component pointer if success, NULL or failed
*
* \sa NCS_INVSB_CREATE_INFO, ncsCreateInvsbComp
*/
mInvsbComp * ncsCreateInvsbCompIndirect(const char* class_name, \
NCS_INVSB_CREATE_INFO *create_info);
```

其中，函数 ncsCreateInvsbCompIndirect 用到的 NCS_INVSB_CREATE_INFO 结构定义如下

```
/**
* A struct include Invisible Component Create info
```

```
*
* \sa ncsCreateInvsbCompIndirect
*/
typedef struct _NCS_INVSB_CREATE_INFO {
        /**
        * The parent Component
        */
        mComponent        * parent;
        /**
        * The id of component
        */
        int                 id;
        /**
        * The property of Component
        *
        * \sa NCS_PROP_ENTRY
        */
        NCS_PROP_ENTRY    * props;
        /**
        * The event handlers array
        *
        * \sa NCS_EVENT_HANDLER
        */
        NCS_EVENT_HANDLER * handlers;
```

```
        /**
        * Use defined data
        */
        DWORD               user_data;
}NCS_INVSB_CREATE_INFO;
```

- 注： **不鼓励直接使用该函数创建不可见组件** ，它们在手写代码中没有优势，优势在于，可以利用 miniStudio 提供的资源来加载。

故省略例子

mInvsbComp 事件

继承自 mComponent 的事件

mTimer

控件名称

NCSCTRL_TIMER

英文名

Timer

简要介绍

对 MiniGUI SetTimerEx 和 KillTimer 的封装

示意图

无示意图，在 miniStudio 中的图标是 

继承关系

- mObject
  - mComponent
    - mInvsbComp
      - mTimer

mTimer 风格

继承自 mInvsbComp 的风格

mTimer 属性

继承自 mInvsbComp 的属性

属性 ID	miniStudio 名	类型	权限	说明
NCSP_TIMER_INTERVAL	interval	DWORD	RW	设置 Timer 的时间间隔，以 10ms 为单位， 如果 Timer 正在运行，它会重启 Timer

mTimer 方法

继承自 mInvsbComp 的方法

- start

```
BOOL (*start)(clss *_this);
```

  - 启动 Timer
  - Return TRUE -- 启动成功, FALSE -- 启动失败

- stop

```
void (*stop)(clss *_this);
```

  - 停止正在运行的 Timer

- getParent

```
HWND (*getParent)(clss *_this);
```

  - 获取拥有 Timer 的窗口

mTimer 事件

继承自 mInvsbComp 的事件

事件通知码	说明	参数

MSG_TIMER	直接利用 MiniGUI 的定义	timer 走过的总时间数，即 MSG_TIMER 的 lParam 值

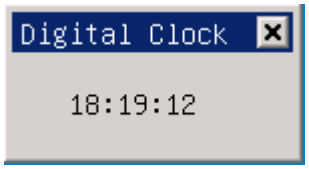
- 注：该事件的回调为：

```
BOOL (*NCS_CB_ONTIMER)(mTimer* timer, DWORD total_count);
```

  - Return : TRUE -- continue Timer, FALSE -- stop Timer
  - Params
    - DWORD total_count - Timer 启动以来总时间数

mTimer 示例

下面示例展示的是用 timer 显示一个数字钟表，运行效果图如下：



- 声明 Timer，使用和控件一样的结构

```
static NCS_WND_TEMPLATE _ctrl_templ[] = {
        {
                NCSCTRL_TIMER,
                100,
                10, 10, 0, 0,
                WS_BORDER | WS_VISIBLE,
                WS_EX_NONE,
                "",
                NULL, //props,
                NULL, //rdr_info
                NULL, //timer_props, //handlers,
                NULL, //controls
```

```
                0,
                0 //add data
        },
```

- 初始化 Timer，建立和一个 Static 控件的连接，并开始 Timer

```
static BOOL mymain_onCreate(mWidget* self, DWORD add_data)
{
        //TODO : initialize
        mTimer * timer = SAFE_CAST(mTimer,
        _c(self)->getChild(self, 100));
        if(timer)
        {
                ncsAddEventListener((mObject*)timer,
                (mObject*)ncsGetChildObj(self->hwnd, 101),
                (NCS_CB_ONPIECEEVENT)update_time,
                MSG_TIMER);
                _c(timer)->start(timer);
        }
        return TRUE;
}
```

- 当 MSG_TIMER 事件发生时，更新时间

```
static BOOL update_time(mStatic *listener,
mTimer* sender,
```

```
int id,
DWORD total_count)
{
        char szText[100];
        time_t tim;
        struct tm *ptm;
        static int old_count = 0;

        time(&tim);
        ptm = localtime(&tim);

        sprintf(szText,
        "%02d:%02d:%d",
        ptm->tm_hour,
        ptm->tm_min,
        ptm->tm_sec);
        old_count = total_count;

        SetWindowText(listener->hwnd, szText);
        InvalidateRect(listener->hwnd, NULL, TRUE);

        return FALSE;
}
```

完整的代码参考 timer.c

第二部分第十六章 其他相关类

本章介绍一些被 mGNCS 非控件类。它们是被设计出来提供给控件类使用。

mReferencedObj

该类封装一个自动计数，允许对象被传递时通过自动引用来控制该对象的生命周期。

该类主要用于一些非控件类上，例如 mToolItem, mPropMenuMgr 等类

- 继承关系
  - mObject
    - mRefrencedObj
- 直接子类
  - mPropMenuMgr
  - mToolItem
  - mToolImage

mReferencedObj 方法

- 引用相关的接口，有两个：

```
int addRef(mReferencedObj* self);

int release(mReferencedObj* self);
```

addRef 自动给计数加 1,并返回新的计数。 release 将自动给计数减 1,并返回新计数；如果计数为 0,自动调用 destroy 方法，删除该对象，包括该对象的内存。 **mReferenceObj 及其派生类需要调用 NEW 或者 NEWEX 宏来创建** ，因为 release 方法会调用 DELETE 宏来释放创建的对象。 派生类可以通过覆盖 release 方法，来改变对象的释放方式。

mReferencedObj 示例

该类为抽象类，不能直接使用

mPopMenuMgr

mPopMenuMgr 保存了 PropMenu 的信息，它能够：

- 随时创建一个 PopMenu 供使用
- 可以取得部分 MenuItem 的信息，填充到 MENUITEMINFO 结构中，提供给 MiniGUI 的相关函数使用

- 继承关系
    - mObject
        - mReferencedObj
            - mPopMenuMgr

mPopMenuMgr 方法

- 向 mPopMenuMgr 添加一个 MenuItem 的方法

```
BOOL addItem(mPropMenuMgr *self , \

int type, \

const char * str, \

PBITMAP bmp, \

int id, \

int state,  \

mPopMenuMgr *subMenu, \

DWORD add_data);
```

    - 参数
        - type - 菜单项类型，同 MiniGUI MENUITEMINFO 的定义
            - MTF_STRING
            - MTF_BITMAP
            - MTF_BMPSTRING
            - MTF_RADIOCHECK
            - MTF_MARKCHECK
            - MTF_SEPARATOR
        - str - Item 的 caption，在 type == MTF_STRING 或 MTF_BMPSTRING 有效
        - bmp - item 的位图，在 type == MTF_BITMAP 或 MTF_BMPSTRING 有效
        - id - item 的 id, **必须** 标记一个 id
        - state - item 状态，同 MiniGUI MENUITEMINFO 的定义
            - MFS_GRAYED
            - MFS_DISABLED

        - MFS_CHECKED
        - MFS_ENABLED
        - MFS_UNCHECKED
      - subMenu - 子菜单管理器
      - add_data - 用户附件数据
    - return : TRUE/FALSE

- 创建一个 PopMenu，返回该 Menu 的句柄

```
HMENU createMenu(mPopMenuMgr *self);
```

- 自动创建并弹出一个 PopMenu

```
void popMenu(mPopMenuMgr *self, mObject *owner);
```

    - params:
      - owner 指出 PopMenu 相关关联的对象， 它必须是一个 mWidget 或者其子类。*PopMenu 将发送 MSG_COMMAND 给 owner*
    - return : 无

- 添加一个分割条到 MenuItem 中去

```
BOOL addSeparator(mPopMenuMgr * self);
```

- 获取指定 MenuItem 的信息

```
BOOL getMenuItem(mPopMenuMgr * self, int idx, MENUITEMINFO *pmii, BOOL byCommand);
```

    - 将指定 MenuItem 的信息填充到 MENUITEMINFO 中去
    - params:
      - idx - menuitem 的索引或者 id
      - pmii - 输出参数
      - byCommand : TRUE - idx 为 menu item 的 id; FALSE - idx 为 menu item 的索引
    - return : TRUE / FALSE

mPopMenuMgr 示例

参见 mMenuButton 的示例

mToolImage

mToolImage 是为 mToolItem 提供图片的类。该类封装了各种类型的图片，以方便用户选用

- 继承关系
  - mObject
    - mReferencedObj
      - mToolImage

mToolImage 方法

mToolImage 的方法是开放的，下面提供的函数都是可以直接调用的

- 从一个图片对象创建新的 mToolImage 对象

```
mToolImage * ncsNewToolImage(PBITMAP pbmp, int cell_count, BOOL autoUnload, BOOL bVert);
```

  - params
    - pbmp : 图像的源
    - cell_count: pbmp 包含的小图像的个数
    - autoUnload : 自动调用 UnloadBitmap 删除该图像
    - bVert : 小图像的排列是否是垂直的
  - return : mToolImage 指针

- 从一个图片文件创建 mToolImage 对象

```
mToolImage * ncsNewToolImageFromFile(const char *fileName, \

int cell_count, \

BOOL autoUnload, \

BOOL bVert);
```

- 释放一个 mToolImage 对象

```
void ncsFreeToolImage(mToolImage *mti);
```

- 绘制 mToolImage 管理的指定位置的图片

```
BOOL ncsDrawToolImageCell(mToolImage *mti, HDC hdc, int idx, const RECT *prc);
```

  - params:
    - mti : mToolImage 指针
    - hdc : 目标 DC 句柄
    - idx : 小图像索引
    - prc: 目标矩形

- return TRUE/FALSE

对于用户来说，主要使用创建和删除相关的函数，绘制函数是由 ToolItem 使用的

mToolImage 示例

mToolItem

mToolItem 是 mToolbar 的 item 的基类。这个类以及它的派生类是不公开的。用户只需要通过对外提供的 API，自动创建即可。

mToolbar 也将自动管理 mToolItem 的删除

mToolItem 的类型

mToolItem 有很多子类，对外接口上，表现为各种类型，这些定义如下：

```
enum mToolItemType{
        NCS_UNKNOWNTOOLITEM = 0,
        NCS_PUSHTOOLITEM,
        NCS_MENUTOOLITEM,
        NCS_WIDGETTOOLITEM,
        NCS_SEPARATORTOOLITEM
};
```

可以通过以下函数来检测一个 ToolItem 的类型

- int ncsGetToolItemType(void *toolitem);
  - 获取 ToolItem 的类型

以下函数可以快速检测一个 item 的类型

- BOOL ncsIsPushToolItem(void *toolitem);
- BOOL ncsIsMenuToolItem(void *toolitem);
- BOOL ncsIsSeparatorToolItem(void *toolitem);
- BOOL ncsIsWidgetToolItem(void *toolitem);

mToolItem 创建和删除

- 创建一个 PushToolItem

```
void * ncsCreatePushToolItem(int id, mToolImage * img, const char * str, UINT flags);
```

  - params:
    - img : mToolImage 指针,可以为 NULL
    - str : 文字指针，可以为 NULL, 但，*img 和 str 中必须有一个不为 NULL*
    - flags: 定义 image 和 str 的关系
      - NCS_TOOLITEM_FLAG_TEXT_LEFT/NCS_TOOLITEM_FLAG_TEXT_UP, 文本在左或者上，默认为右或者下
      - NCS_TOOLITEM_FLAG_VERT, image 和 str 垂直排列，默认是水平排列
  - return : item 指针
- 创建一个 Menu Tool Item

```
void * ncsCreateMenuToolItem(int id, \

mToolImage * img, \

const char * str, \

UINT flags, \

mPopMenuMgr * menu);
```

  - 创建一个 MenuToolItem, 参数 menu 为一个 mPopMenuMgr, 其他参数同上

- 创建一个 Check Tool Item

```
void * ncsCreateCheckToolItem(int id, \

mToolImage * img, \

const char * str, \

UINT flags, \

int state);
```

  - 参数同 ncsCreatePushToolItem

- 创建一个 Radio Tool Item

```
void * ncsCreateRadioToolItem(int id, mToolImage * img, const char * str, UINT flags);
```

  - 参数同 ncsCreatePushToolItem。 **RadioToolItem 是自动分组的，从第一个或者上一个分割符开始到最后一个或者下个分割符结束直接的所有 RadioToolItem 是互斥的**

- 创建一个包含 mWidget 指针的 ToolItem

```
void * ncsCreateWidgetToolItem(mWidget * widget);
```

  - params
    - widget : mWidget 对象指针
  - return : item 指针

- 创建一个分割符

```
void * ncsCreateSeparatorItem(void);
```

  - return item 指针

其他 mToolItem 函数

mToolItem 还提供一些其他的函数，用于对 ToolItem 进行操作

- 获取或者设置 ToolItem 的 ID

```
int ncsToolItem_getId(void *self)



int ncsToolItem_setId(void *self, int id);
```

  - 如果不能设置或者不能获取，则返回-1

- 获取或者设置 CheckToolItem 的 Check 状态

```
BOOL ncsToolItem_setCheck(void *self, int check_state);

int ncsToolItem_getCheck(void *self);
```

  - 只针对 Check/RadioToolItem 起效。当对其他 ToolItem 调用 ncsToolItem_getCheck 时，将永远返回 unchecked 状态

- 对 MenuToolItem 弹出菜单

```
BOOL ncsToolItem_showMenu(void*self, mObject *owner);
```

- 仅针对 menu Tool item, owner 为一个 mWidgt*对象

mToolItem 示例

无

第三部分第一章 EVENT HANDLER 使用及实例

NCS Event Handler

NCS Event Handler 机制

- 作用：接收特定的消息，并在这些消息的回调函数中处理这些消息。
- 原理：创建好消息链表以后，在 NCS 消息循环之中，会调用相应函数，这些函数的作用是查找接收该消息的窗口，当找到接收该消息的窗口后，将会把这个消息发送到对应的窗口之中。
- 创建：在使用 NCS 创建窗口的时候，需要在创建窗口的函数、或者窗口模板中，填写窗口的 NCS_EVENT_HANDLER 的地址，并在这个数据结构中填写需要接受的消息，以及处理这些消息的回调函数。接下来 NCS 会将 NCS_EVENT_HANDLER 和一些窗口数据封装为 NCS_CREATE_INFO 这个结构，NCS_CREATE_INFO 会再次被封装，在 NCS 内部会将这些消息封装成链表。
- 销毁：在窗口销毁的时候，消息链将自行销毁。
- NCS_HANDLER 的数据结构

```
/**
* A struct of event-handler map
*
* \note only used for param
*
* \sa NCS_EVENT_HANDLER_NODE
*/
typedef struct _NCS_EVENT_HANDLER {
        /**
        * The event code
        */
```

```
        int message;

        /**

        * The event callback pointer

        */

        void *handler;

}NCS_EVENT_HANDLER;
```

不同的消息对应着不同的回调函数:

消息	回调函数指针	声明文件
MSG_NCCREATE	NCS_CB_ONNCCREATE	mwidget.h
MSG_CREATE	NCS_CB_ONCREATE	mwidget.h
MSG_INITDIALOG	NCS_CB_ONINITDLG	mwidget.h
MSG_ACTIVE MSG_FONTCHANGED MSG_KEYLONGPRESS MSG_KEYALWAYSPRESS MSG_DESTROY	NCS_CB_ONVOID	mwidget.h
MSG_SIZECHANGING	NCS_CB_ONSZCHGING	mwidget.h
MSG_SIZECHANGED	NCS_CB_ONSZCHGED	mwidget.h
MSG_CSIZECHANGED	NCS_CB_ONCSZCHGED	mwidget.h
MSG_FONTCHANGING	NCS_CB_ONFONTCHGING	mwidget.h
MSG_ERASEBKGND	NCS_CB_ONERASEBKGND	mwidget.h
MSG_PAINT	NCS_CB_ONPAINT	mwidget.h
MSG_CLOSE	NCS_CB_ONCLOSE	mwidget.h
MSG_KEYDOWN MSG_KEYUP MSG_CHAR MSG_SYSKEYDOWN MSG_SYSKEYUP MSG_SYSCHAR	NCS_CB_ONKEY	mwidget.h
MSG_LBUTTONDOWN MSG_LBUTTONUP MSG_LBUTTONDBCLK MSG_MOUSEMOVE MSG_RBUTTONDOWN MSG_RBUTTONUP MSG_RBUTTONDBCLK	NCS_CB_ONMOUSE	mwidget.h
MSG_NCLBUTTONDOWN MSG_NCLBUTTONUP MSG_NCLBUTTONDBCLK MSG_NCMOUSEMOVE MSG_NCRBUTTONDOWN MSG_NCRBUTTONUP	NCS_CB_ONNCMOUSE	mwidget.h

MSG_NCRBUTTONDBCLK		
MSG_HITTEST MSG_NCHITTEST	NCS_CB_ONHITTEST	mwidget.h
MSG_HSCROLL MSG_VSCROLL	NCS_CB_ONSCROLL	mwidget.h
MSG_COMMAND	NCS_CB_ONCMD	mwdiget.h
MSG_TIMER	NCS_CB_WIDGET_ONTIMER	mwidget.h
user define message	NCS_CB_ONMSG	mwidget.h
all notification event	NCS_CB_NOTIFY	mwidget.h

NCS Event HANDLER 设置

- 下面这四个函数都是 NCS 创建窗口的函数，创建的时候都要填写 NCS_EVENT_HANDLER 作为其中的一个参数。

ncsCreateWindow

```
/**

* \fn mWidget* ncsCreateWindow (const char *className, const char *caption, DWORD style, DWORD exStyle, \

*      int id, int x, int y, int w, int h, HWND parent, \

*      NCS_PROP_ENTRY * props, \

*      NCS_RDR_INFO * rdrInfo, \

*      NCS_EVENT_HANDLER * handlers, \

*      DWORD addData);

*

*  \brief create a NCS widget

*

*  \param className the class name of widget.

*         the class name must be register by \ref ncsRegisterComponent.
```

```
*         Any drived from \ref mComponent and registered component is validate
*  \param caption  the caption of the widget, ignored if class_name is a \ref mInvsbComp
*  \param style the style of widget, ignored if class_name is a \ref mInvsbComp
*  \param exStyle  the extend style of widget, ignored if class_name is a \ref mInvsbComp
*  \param id  the id of component
*  \param x  the x position of widget, ignored if class_name is a \ref mInvsbComp
*  \param y  the y position of widget, ignored if class_name is a \ref mInvsbComp
*  \param w  the width of widget ,ignored if class_name is a \ref mInvsbComp
*  \param h  the height of widget , ignored if class_name is a \ref mInvsbComp
*  \param parent  the handle of parent, if class_name is a \ref mInvsbComp, the \a parent must
*         releated a mWidget object
*   \param props  the properties array pointer, end by {-1, 0} if it's not NULL
*   \param rdrInfo the renderer info pointer
*   \param handlers the handlers of event array pointer, end by {-1, NULL}, if it's not NULL
*   \param addData the additional data send to callback \ref NCS_CB_ONCREATE
*
*   \return mWidget* - a mWidget pointer, if class_name is a \ref mInvsbComp, it's a
mInvisibleComponent* pointer
*
*   \sa ncsCreateMainWindow, nscCreateWindowIndirect, NCS_PROP_ENTRY, NCS_RDR_INFO,
*         NCS_EVENT_HANDLER, NCS_CB_ONCREATE, mWidget, mInvisibleComponent
*/
MGNCS_EXPORT mWidget* ncsCreateWindow (const char *className, const char *caption, DWORD style,
DWORD exStyle, \
int id, int x, int y, int w, int h, HWND parent, \
```

```
NCS_PROP_ENTRY * props, \
NCS_RDR_INFO * rdrInfo, \
NCS_EVENT_HANDLER * handlers, \
DWORD addData);
```

ncsCreateMainWindow

```
/**
* \fn mWidget* ncsCreateMainWindow (const char *className, const char *caption,
*          DWORD style, DWORD exStyle, \
*          int id, int x, int y, int w, int h, HWND host, \
*          HICON hIcon, HMENU hMenu, NCS_PROP_ENTRY * props, \
*          NCS_RDR_INFO * rdrInfo, \
*          NCS_EVENT_HANDLER * handlers, \
*          DWORD addData);
*
*  \brief create a NCS main window
*
*  \param className the class name of widget.
*         the class name must be register by \ref ncsRegisterComponent.
*         And must be \ref NCSCTRL_MAINWND or its dirved class.
*  \param caption the caption of the main window
*  \param style  the style of main window
*  \param exStyle  the extend style of main window
*  \param id the id of main window
```

```
*  \param x the x position of main window
*  \param y the y position of main window
*  \param w the width of main window
*  \param h the height of main window
*  \param host the handle of host window, can be NULL
*  \param hIcon the icon of main window
*  \param hMenu the menu bar handle
*  \param props the properties array pointer, end by {-1, 0} if it's not NULL
*  \param rdrInfo the renderer info pointer
*  \param handlers the handlers of event array pointer, end by {-1, NULL}, if it's not NULL
*  \param addData the additional data send to callback \ref NCS_CB_ONCREATE and \ref
NCS_CB_ONINITDLG
*
*  \return mWidget* - a mWidget pointer, must be a mMainWnd instance
*
*  \sa ncsCreateWindow,ncsCreateWindowIndirect, nscCreateMainWindowIndirect, NCS_PROP_ENTRY,
*             NCS_RDR_INFO, NCS_EVENT_HANDLER, NCS_CB_ONCREATE, mWidget,
*/
MGNCS_EXPORT mWidget* ncsCreateMainWindow (const char *className, const char *caption,
DWORD style, DWORD exStyle, \
int id, int x, int y, int w, int h, HWND host, \
HICON hIcon, HMENU hMenu,
NCS_PROP_ENTRY * props, \
NCS_RDR_INFO * rdrInfo, \
NCS_EVENT_HANDLER * handlers, \
```

```
DWORD addData);
```

ncsCreateWindowIndirect

```
//create control window indirect

/**

* \fn mWidget* ncsCreateWindowIndirect( const NCS_WND_TEMPLATE* tmpl, HWND hParent);

* \brief create a mComponent by \ref NCS_WND_TEMPLATE,

*

* \param tmpl the template pointer

* \param hParent the parent handle, if NCS_WND_TEMPLATE.class_name is a \ref mInvsbComp,

*        hParent must releated a mWidget object

*

* \return mWidget* - then pointer of object or NULL if failed

*

* \sa ncsCreateWindow, NCS_WND_TEMPLATE

*/

MGNCS_EXPORT mWidget* ncsCreateWindowIndirect( const NCS_WND_TEMPLATE* tmpl, HWND hParent);
```

ncsCreateMainWindowIndirect

```
/**

* \fn mWidget* ncsCreateMainWindowIndirect(const NCS_MNWND_TEMPLATE* tmpl, HWND hHost);

* \brief create a main window from a template

*

* \param tmpl - the template of main window

* \param hHost - the host window handler of the main window
```

```
*
* \return mWidget * - the Instance of mMainWnd or NULL
*
* \sa ncsCreateMainWindow, NCS_MNWND_TEMPLATE
*/
MGNCS_EXPORT mWidget* ncsCreateMainWindowIndirect(const NCS_MNWND_TEMPLATE* tmpl, HWND hHost);
```

NCS_MNWND_TEMPLATE

```
/**
* A struct include all the creating info for ncsCreateMainWindowIndirect,
*
* \note same as \ref ncsCreateMainWindow 's params
*
* \sa NCS_WND_TEMPLATE, ncsCreateMainWindow
*/
typedef struct _NCS_MNWND_TEMPLATE{
        const char*         class_name;
        int                 id;
        int                 x, y, w, h;
        DWORD               style;
        DWORD               ex_style;
        const char*         caption;

        NCS_PROP_ENTRY*     props;
```

```
        NCS_RDR_INFO*       rdr_info;
        NCS_EVENT_HANDLER*  handlers;
        NCS_WND_TEMPLATE*   ctrls;
        int                 count;
        DWORD               user_data;

        //FIXED ME Maybe I  should not put these two param here
        DWORD               bk_color;
        const char*         font_name;

        HICON               hIcon;
        HMENU               hMenu;
}NCS_MNWND_TEMPLATE;
```

NCS Event HANDLER 示例

```
static NCS_EVENT_HANDLER mymain_handlers[] = {
        {MSG_CREATE/*要接收的消息*/, mymain_onCreate/*接收消息的处理函数*/},
        {MSG_CLOSE/*要接收的消息*/, mymain_onClose/*接收消息的处理函数*/},
        {0, NULL}
};

static NCS_MNWND_TEMPLATE mymain_templ = {
        NCSCTRL_MAINWND,
        1,
        0, 0, 260, 180,
```

```
        WS_CAPTION | WS_BORDER | WS_VISIBLE,
        WS_EX_NONE,
        "Push Button",
        NULL,
        NULL,
        mymain_handlers,//窗口的消息 EVNET HANDLER 的地址。
        _ctrl_templ,
        sizeof(_ctrl_templ)/sizeof(NCS_WND_TEMPLATE),
        0,
        0, 0,
};

int MiniGUIMain(int argc, const char* argv[])
{
        ncsInitialize();
        mDialogBox* mydlg = (mDialogBox *)ncsCreateMainWindowIndirect
        (&mymain_templ, HWND_DESKTOP);
        _c(mydlg)->doModal(mydlg, TRUE);//在这里创建窗口
        MainWindowThreadCleanup(mydlg->hwnd);
        ncsUninitialize ();
        return 0;
}
```

按键过滤

按键过滤的作用

- 按键过滤的作用就是， 当一些 MiniGUI 窗口或控件，不需要接收某些字符的时候，使用字符过滤功能。可以将那些不需要的字符过滤掉。

按键处理回调函数类型定义

- MiniGUI 中所有的按键消息都使用这个回调函数，其中包括 MSG_KEYDOWN、MSG_KEYUP、MSG_CHAR、MSG_SYSKEYDOWN、MSG_SYSKEYUP、MSG_SYSCHAR。

```
/**
* \typedef BOOL (*NCS_CB_ONKEY)(mWidget*, int message, int code, DWORD keyStatus);
* \brief the callback of event \a MSG_KEYDOWN \a MSG_KEYUP \a MSG_CHAR \a MSG_SYSKEYDOWN
*        \a MSG_SYSKEYUP \a MSG_SYSCHAR
*
* \param mWidget *  the sender pointer
* \param message event code, distinguish the events
* \param code the scancode (\a MSG_KEYDOWN \a MSG_KEYUP \a MSG_SYSKEYDOWN \a MSG_SYSKEYUP)
*        or ascii code (\a MSG_CHAR \a MSG_SYSCHAR)
* \param keyStatus The shift key status when this message occurred
*
* \return TRUE - abort the message processing, FALSE - continue the default message processing
*/
typedef BOOL (*NCS_CB_ONKEY)(mWidget*, int message, int code, DWORD keyStatus);
```

按键过滤的原理

- 按键过滤的原理就是在接收 MSG_CHAR 窗口的回调函数中，阻止调用窗口默认处理函数。

- 在窗口的 MSG_CHAR 回调函数中，如果返回 FALSE，则该消息可以进入窗口默认处理函数，如果返回 TRUE，则直接返回，不再进入窗口默认处理函数。

按键过滤的使用方式

- 下面代码的功能是在接收到 MSG_CHAR 消息后将字母过滤掉

```
static BOOL Sledit1_onChar (mWidget* self, int message, int scancode, DWORD key_status)

{

        if ((scancode >= '0') && (scancode <= '9'))/*判断是否为数字*/

        return FALSE;/*是数字，窗口默认处理函数将对这些数据进行处理*/

        else

        return TRUE;/*非数字，窗口默认处理函数不再处理这些数据*/

}



static NCS_EVENT_HANDLER Sledit1_handlers [] = {

        {MSG_CHAR, Sledit1_onChar},/*此控件接收 MSG_CHAR 消息，Sledit1_onChar 为接到这个消息的回调函数*/

        {-1, NULL}

};
```

自定义消息

自定义消息的功能

- MiniGUI 提供的消息不能满足用户需求的时候，用户可以定义自己的消息。
- 为了使用户能够自定义消息,MiniGUI 定义了 MSG_USER 这个 宏,应用程序可如下定义自己的消息:

```
#define MSG_MYMESSAGE1 (MSG_USER + 1)

#define MSG_MYMESSAGE2 (MSG_USER + 2)
```

自定义消息回调函数指针

```
/**
* \typedef int (*NCS_CB_ONMSG)(mWidget*, int message, WPARAM wParam, LPARAM lParam);
* \brief the callback of a common message event
*
* \param mWidget * sender pointer
* \param message the event code
* \param wParam the wParam of message event
* \param lParam the lParam of message event
*
* \return int - this value will be as the finally return value of event, and none
*         default processing would be called
*/
typedef int (*NCS_CB_ONMSG)(mWidget*, int message, WPARAM wParam, LPARAM lParam);
```

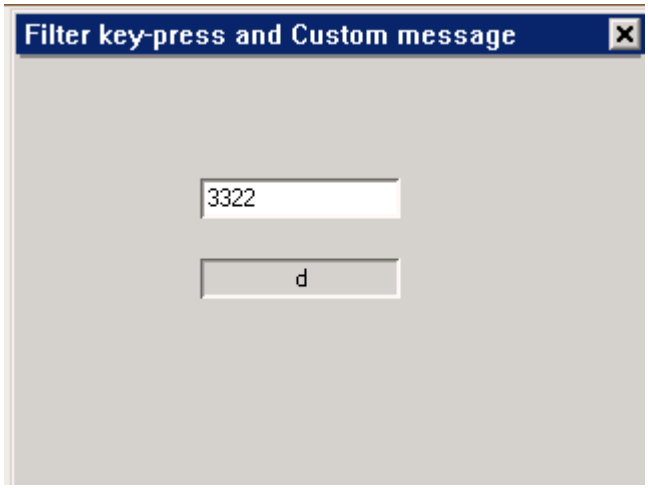
自定义消息的使用方式

- 在这个控件的消息句柄中添加了自定义消息，和接到这个消息后的回调函数。

```
static BOOL Sledit1_onChar (mWidget* self, int message, int scancode, DWORD key_status) ;

static NCS_EVENT_HANDLER Static1_handlers [] = {

        {MSG_MYMESSAGE1, Static1_onMyMessage},/*此控件接收“MSG_MYMESSAGE1”这个消息，
Static_onMyMessage 为这个消息的回调函数*/

        {-1,NULL}

};
```

按键过滤和自定义消息的结合实例

实例截图



运行效果说明

- 上面的单行编辑框为输入位置，其中滤去了字母按键，但当有字母按键的时候，会向 static 控件发送自定义消息，将 scancode 通过 wParam 参数发送给 static 控件，static 会将其显示出来。
- 在单行编辑框中进行按键过滤，将滤出的非数字发送到指定控件之中

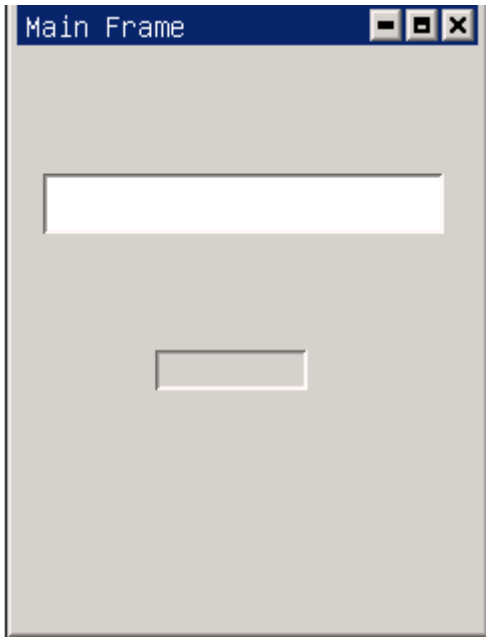
在 Studio 的详细制作说明

添加窗口

- 点击下面的按钮添加窗口，在 guibuilder 的左上。

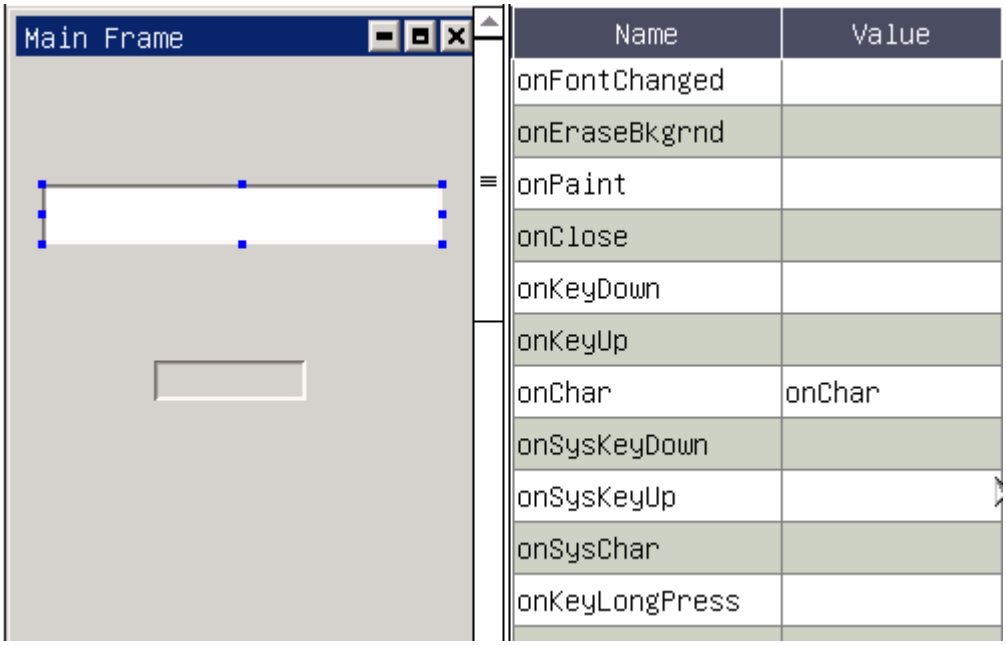
在窗口上添加控件

- 在窗口上建立 static 控件和 sledit 控件。
- 建立完的窗口如下图：



在右边 EVENT 标签拦下添加控件的接收消息，并完成回调函数

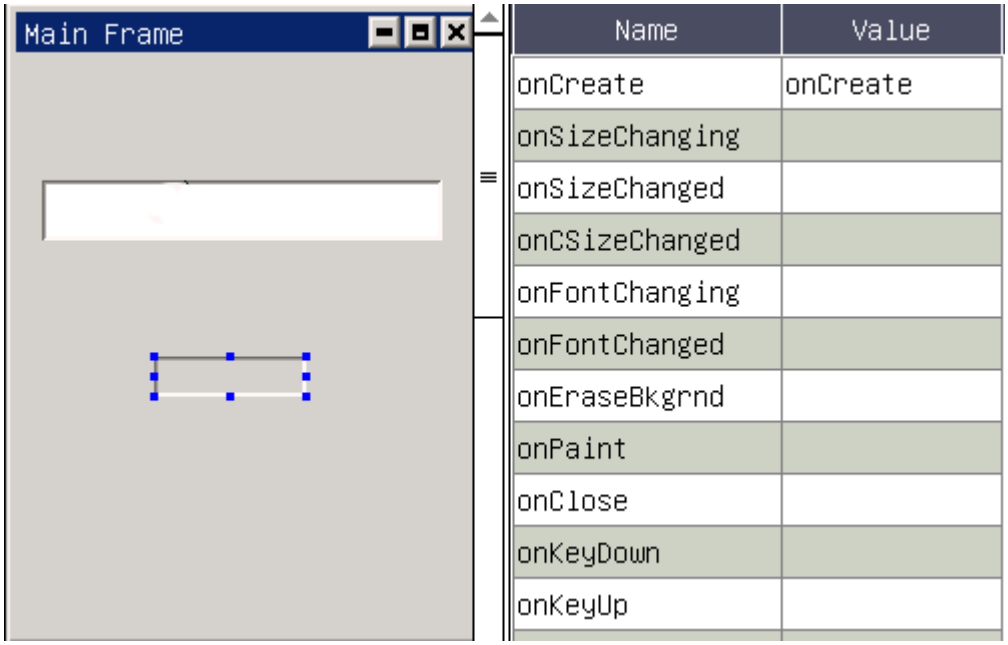
- 给 sledit 控件添加消息，并填写回调函数
- 在 sledit 控件的 EVENT 的 on_Char 事件中双击进入 on_Char 的回调函数
- 在这里 miniStudio 会生成默认的名字，也可以填写自己需要的名字
- 这样 miniStudio 会跳入 ECLIPSE 中对应的消息回调函数中。
- 添加回调函数的内容




```
//$func @1057925120 onChar -- Need by merge, don't modify
```

```
static BOOL Sledit1_onChar (mWidget* self, int message, int scancode, DWORD key_status)

{

        HWND hwnd;

        if ((scancode >= '0') && (scancode <= '9')){

                return FALSE;

        } else {

                hwnd = GetDlgItem (GetParent(self->hwnd), ID_STATIC1);

                SendMessage (hwnd, MSG_MYMESSAGE1, scancode, 0L);

                return TRUE;

        }

        //TODO:

        return FALSE; /*continue default message process*/

}
```

- 给 static 控件添加消息。



- 在 static 控件的 EVENT 的 on_Create 事件中双击进入 on_Create 的回调函数

- 在这里 miniStudio 会生成默认的名字，也可以填写自己需要的名字
- 这样 miniStudio 会跳入 ECLIPSE 中对应的消息回调函数中
- 在这里找到 static 的 EVENT_HANDLER 定义的位置，添加自定义事件和回到函数

```
//$handle @986179584 -- Need by merge, don't modify

static NCS_EVENT_HANDLER Static1_handlers [] = {

        {MSG_CREATE, Static1_onCreate},

        //$user -- TODO add your handlers hear

        {MSG_MYMESSAGE1, Static1_onMyMessage},/*接收自定义消息*/

        {-1,NULL}

};
```

- 填写回到函数 Static1_onMyMessage 的内容
- 当接到自定义消息后，对通过附加数据传达来的数据进行处理

```
static int Static1_onMyMessage (mMainWnd* self, int message, WPARAM wParam, LPARAM lParam)

{

        char buf[2];

        buf[1] = '\0';

        buf[0] = wParam;/*获得传递来的数据*/

        SetWindowText (self->hwnd, buf);/*将其显示*/

        return 0;

}
```

第三部分第二章 Listview 控件高级编程

mListView 高级编程

mListView 提供的很多高级编程方法，以丰富 mListView 控件的功能和适应能力。常用的高级编程方法有：

- Item 附加数据： 允许用户将自定义的数据和每个 item 相关联。
- 自定义绘制，包括自定义绘制 Header 和自定义绘制 Item。

下面对每种方法做详细介绍

Item 附加数据管理和使用

Item 附加数据的作用

mListView 中的 Item 主要包含了显示信息，如 Item 的文本、图标等。但是用户往往需要更多、更符合特定格式的数据，无法在 Item 中直接保存。所以，我们通过为 Item 添加附加数据（大多数情况下都是一个指针），来保存和特定 Item 关联的数据。

如何将附加数据与 Item 相关联

将附加数据与 Item 相关联有如下三种方式：
第一种，在 Item 创建的时候添加附加数据。

- 实例：

```
USER_DATA* user_data;

NCS_LISTV_ITEMINFO info;

.....

info.addData = (DWORD)user_data; //在这里添加其自身的附加数据。

.....

hItem = _c(self)->addItem (self, info);
```

第二种，在 Item 创建后，使用类 mItemView 中的 setAddData 函数添加附加数据。

```
void (*setAddData)(clsName*, HITEM hItem, DWORD addData);

/*

*  - void (*\b setAddData)(mItemView *self, \ref HITEM hItem, DWORD addData);\n

*    The function sets the additional data of the specified item.

*      \param hItem    The handle to the specified item.

*      \param addData  The new additional data.

*/
```

- 实例：

```
_c(mItemview)->setddData(itemview, hItem, addData);
```

第三种，在 Item 创建后把 HITEM 强制转换为(mItem*)类型，使用类 mItem 中的 setItemAddData 函数添加附加数据。

```
void (*setItemAddData)(clsName*, DWORD addData);

/*

*      - void (*\b setItemAddData)(mItem *self, DWORD addData);\n

*        The function is used to set the addtional data.

*          \param addData    The additional data.

*/
```

- 实例：

```
mItem*     item;

.....

item    = (mItem*)hItem;

_c(item)->setItemAddData(item, addData);
```

如何从 item 中获取附加数据

获取 item 中的附加数据有如下两种方式：

第一种，通过调用类 mItemView 中的函数 getAddData，获取附加数据。

```
DWORD (*getAddData)(clsName*, HITEM hItem);

/*

*  - DWORD (*\b getAddData)(mItemView *self, \ref HITEM hItem);\n

*    The function gets the additional data of the specified item.

*/
```

- 实例：

```
USER_DATA* adddata;

.....

adddata = (USER_DATA*)_c(itemview)->getAddData(itemview, hItem);
```

第二种，把 HITEM 强制转换为(mItem*)类型，通过调用类 mItem 中的 getItemAddData 函数获取附加数据。

```
DWORD (*getItemAddData)(clsName*);

/*

*      - DWORD (*\b getItemAddData)(mItem *self);\n

*        The function is used to get the additional data.

*          \return The additional data.

*/
```

- 实例：

```
USER_DATA* adddata;

.....

adddata = (USER_DATA*)_c(item)->getItemAddData(item);
```

如何释放 Item 中的附加数据

首先需要自定义 Item Destroy 函数，再通过调用类 mItemView 中的函数 setItemDestroy 进行回调函数的设置，这样在 Item 被销毁的时候，Item Destroy 函数就会被调用。用户可以在 Item Destroy 函数中释放附加

数据所占用的空间。

```
/**
* \var typedef void (*NCS_CB_DSTRITEM)(mItemView *self, HITEM hItem);
* \brief The callback of destroying item.
*/
typedef void (*NCS_CB_DSTRITEM)(mItemView *self, HITEM hItem);



NCS_CB_DSTRITEM (*setItemDestroy)(clsName*, NCS_CB_DSTRITEM func);
/*
*  - \ref NCS_CB_DSTRITEM (*\b setItemDestroy)(mItemView *self, \ref NCS_CB_DSTRITEM func);\n
*    Sets the user-defined function of destroying item.
*      \param func     The new funciton.
*      \return         The old function.
*/
```

- 实例：

```
/*设置 item Destroy 的回调函数*/
_c(listview)->setItemDestroy (listview, item_destroy);
```

自定义 Item 绘制

用户可以通过 Item 自绘制，在 Item 中绘制自己需要的图形和文字。当用户使用了 Item 自绘制后，Item 自身将不再绘制，而交给用户定义的 Item 绘制函数进行处理。使用类 mItemView 中的函数 setItemDraw 设置新自绘函数后，该函数会返回旧的绘制函数指针，用户可以保存下来，放在新绘制函数中使用，这样可以在原有绘制图像的基础上进行自绘制图像了。

- 自定义 Item 绘制回调函数介绍 ：

```
/**
```

```
* \var typedef void (*NCS_CB_DRAWITEM) (mItemView *self, HITEM hItem, HDC hdc, RECT *rcDraw);

* \brief The callback of drawing item.

*/

typedef void (*NCS_CB_DRAWITEM)(mItemView *self, HITEM hItem, HDC hdc, RECT *rcDraw);
```

-
  - self 是 mItemView 自身。
  - hItem 可以强制转化为 mItem。
  - hdc 是返回绘制区域的 HDC。
  - rcDraw 是的 Item 矩形区域。

- 自定义 Item 绘制函数设置是通过类 mItemView 中的函数 setItemDraw 进行设置的。

```
NCS_CB_DRAWITEM (*setItemDraw)(clsName*, NCS_CB_DRAWITEM func);

/*

*  - \ref NCS_CB_DRAWITEM (*\b setItemDraw)(mItemView *self, \ref NCS_CB_DRAWITEM func);\n

*    Sets the user-defined function of drawing item.

*      \param func     The new funciton.

*      \return         The old function.

*/
```

- 实例：

```
static NCS_CB_DRAWITEM old_itemdraw;

/*设置 Item 自绘制的回调函数，并将旧的绘制函数保存在 old_itemdraw 中*/

old_itemdraw = _c(itemview)->setItemDraw (itemview, new_itemdraw);
```

自定义 header 绘制

header 自绘制可以方便用户在 mListView 控件 header 中绘制自己所需要的图像或文字。当用户使用了 header 自绘制后，mListView 自身将不再绘制 header 部分，而交给用户定义的 header 绘制函数进行处理。

- 自定义 header 绘制回调函数介绍：

```
/** Type of drawing header background. */
typedef void (*NCS_CB_LISTV_DRAWHDRBK)(mListView *self, HITEM hHdr, HDC hdc, RECT *rcDraw);
/** Type of drawing header item. */
typedef void (*NCS_CB_LISTV_DRAWHDR)(mListView *self, HITEM hHdr, HDC hdc, RECT *rcDraw);

/** Contains function pointers for customized drawing */
typedef struct _NCS_CB_LISTV_CSTMHDROPS
{
        /** Colunm header background drawing function */
        NCS_CB_LISTV_DRAWHDRBK pfnDrawHdrBk;
        /** Colunm header item drawing function */
        NCS_CB_LISTV_DRAWHDR   pfnDrawHdrItem;
} NCS_CB_LISTV_CSTMHDROPS;
```

-
  - 这两个函数中的参数是相同的。
  - self 是 mListView 自身。
  - hHdr 可以强制转换为(mListColumn*)使用。
  - hdc 是返回的绘制区域的 HDC。
  - rcDraw 是返回的矩形区域。

- 自定义 header 绘制设置是通过调用类 mListView 中的函数 setCustomDrawHeader 进行回调函数设置的。

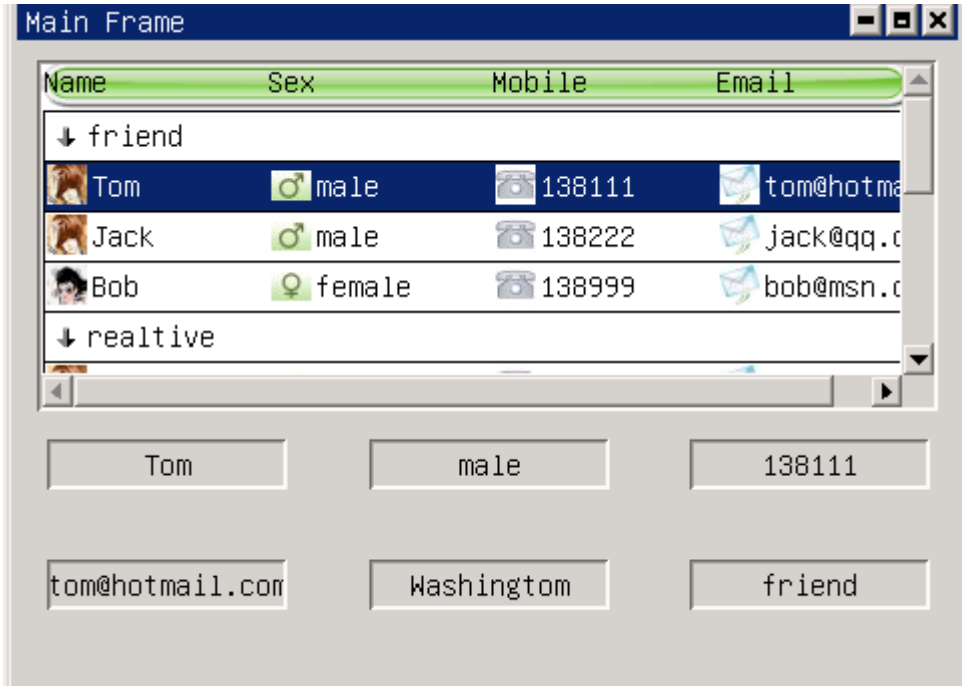
```
void (*setCustomDrawHeader)(clsName*, NCS_CB_LISTV_CSTMHDROPS *func);
/*
* - void (*\b setCustomDrawHeader)(mListView *self, \ref NCS_CB_LISTV_CSTMHDROPS *func); \n
*   Set the custom drawing header functions.
*/
```

- 实例：

```
/*自定义的 header 自绘制函数 mlistv_cstm_drawhkrbk 和 mlistv_cstm_drawhr*/

static NCS_CB_LISTV_CSTMHDROPS listv_cstm = {mlistv_cstm_drawhkrbk, mlistv_cstm_drawhr};

_c(listview)->setCustomDrawHeader (listview, &listv_cstm);//设置 header 自绘制回调函数。
```

mListView 实例

mListView 高级编程实例效果



上图可见，mListView 提供了比较丰富的功能，通过 mListView 高级编程，可以让 GUI 更加美观。
上图所提供的实例是一个通讯录实例，其功能是在 mListView 中分组显示联系人信息，当联系人被选中的时候，会在 mListView 下方的控件中显示联系人的全部信息。
通讯录中所使用的 mListView 高级编程部分:

- 使用了附加数据，附加数据里记录着每个联系人的信息，以及使用的图片资源。
- 使用了 head 自绘制功能，绘制 mListView 顶部的 header 部分，还有 header 上的文字。
- 使用了 Item 自绘制工能，为每个 Item 添加了黑线框。

实例中使用的自定的数据结构

```
typedef struct _user_data{
```

```
        char* name;

        char* sex;

        char* mobile;

        char* email;

        char* address;

        char* group;

}USER_DATA;
```

实例中使用的数据记录格式

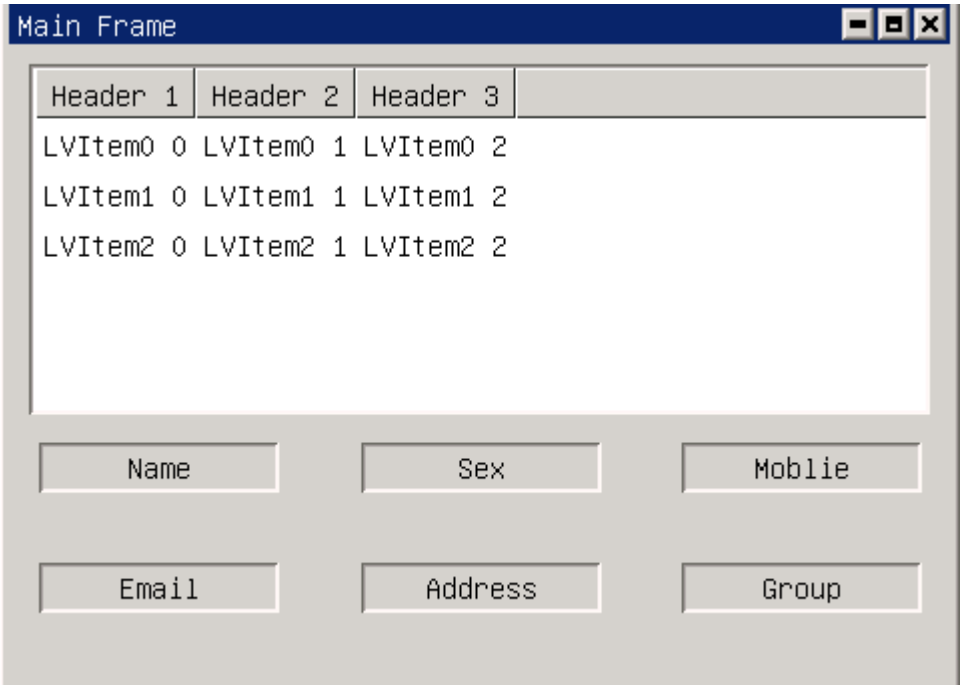
- 使用一个 TXT 文件作为 Item 的数据记录文件，这样就方便使用 MiniGUI 中的函数 GetValueFromEtcFile 获取这个文件中的数据了。
- 数据文件中的成员表示：

```
[group] #//数据库中的组

nr_child=3 #//数据库中有 3 个组

node0=friend #//第一个分组的名字是 friend

.....

[friend]#//friend 组

nr_child=3 #//有三个成员

node0=user_0 #//第一个成员为 user_0

.....

[user_0] #//成员 0

node0=Tom #//成员 0 的姓名

node1=male #//成员 0 的性别

node2=138111 #//成员 0 的电话

node3=tom@hotmail.com #//成员 0 的邮箱
```

```
node4=Washingtom # //成员 0 的地址

node5=friend #//成员 0 的分组

.....................
```

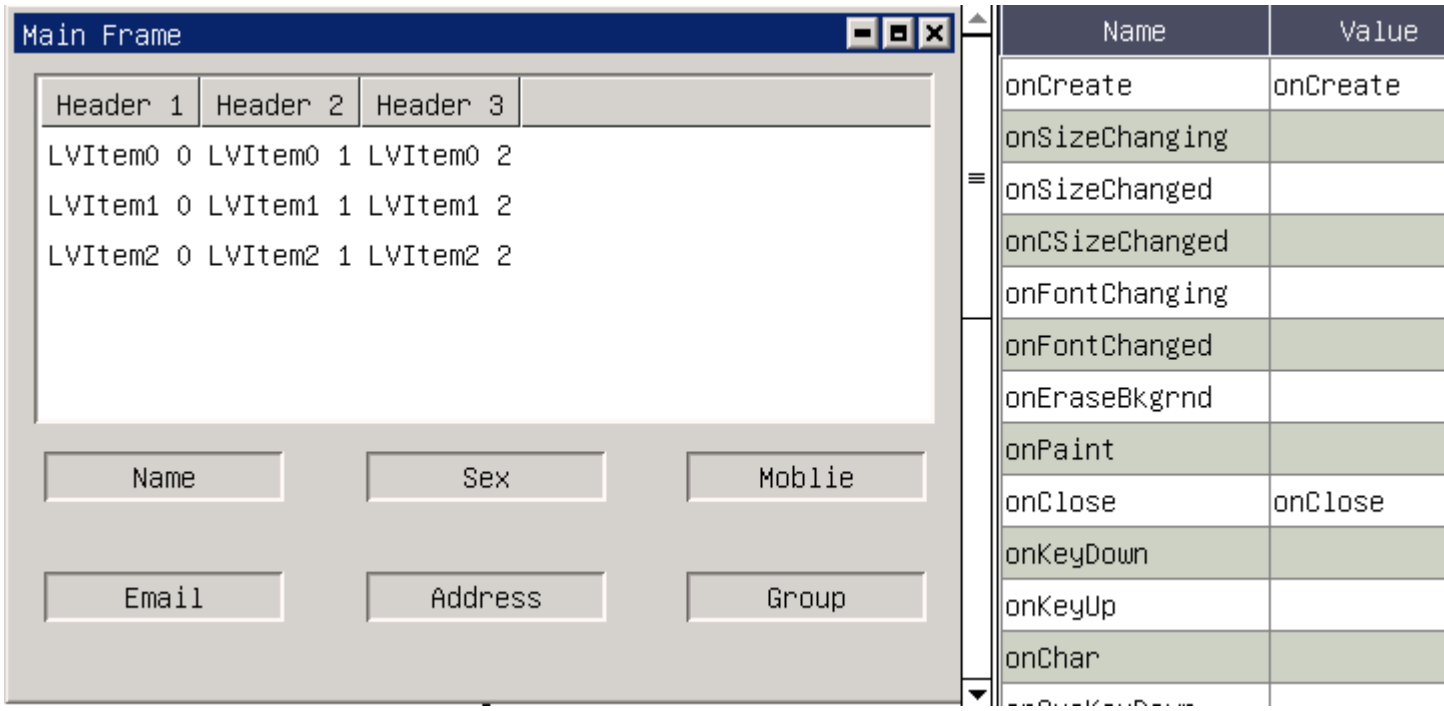
在窗口中创建所需控件并做布局

使用 guibulder 创建窗口，并在窗口上创建所需要的控件，实例中创建了一个 mListView 控件和六个 static 控件。



主窗口关联事件以及处理函数

为主窗口作事件关联，其中关联了 onCreate 和 onClose 事件，下面将介绍这两个事件的处理函数。



主窗口的 onCreate 处理函数

在主窗口的 onCreate 处理函数中，加载了所需的图片资源。调用了 initListView 函数，初始化了 mListView 控件，并为其添加联系人信息。

在函数 initListView 中对 Item 添加了附加数据，添加附加数据的方式是通过函数 get_adddata，在 get_adddata 函数中 malloc 了一些内存空间，用于存储 Item 的附加数据，这些空间将在函数 item_destroy 中释放。

函数 initListView 调用了类 mListView 中的函数 freeze。该函数用于控制 mListView 的更新，防止更新过程中的闪烁。

```
/*在 get_adddata 中创建附加数据*/

static USER_DATA* get_adddata (int n, const char* str)

{

        ..........

        USER_DATA* user_data;

        user_data = malloc (sizeof (USER_DATA));

        ..........

        user_data->name = malloc (sizeof (char)*len);

        ..............
```

```
        return user_data; //返回附加数据的指针。
}



static BOOL initListView (mWidget* self)
{
        ..........................
        _c(listview)->freeze (listview, TRUE);//进行加锁，避免添加数据时候的闪烁。
        /*添加组包括 frend，realtive，schollfellow*/
        ......................
        group_nr = get_nr_child ("group");
        for (i = 0; i < group_nr; i++){
                info.parent  = 0;
                info.index   = i;
                info.addData = 0;//添加附加数据，该附加数据的功能是区分是组，还是组中的成员。
                hItem        = add_group_item (listview, &info, i);

                get_part_text (i, "group", buf);
                user_nr = get_nr_child (buf);

                for (j = 0; j < user_nr; j++){
                        info.parent  = hItem;
                        info.index   = j;
                        /*添加附加数据，附加数据里记录这 Item 中的所有信息*/
```

```
                        info.addData = (DWORD)get_adddata (j, buf);
                        add_user_item (listview, &info);//添加 item 项。
                }
        }
        _c(listview)->freeze(listview, FALSE);
        ..................
}

static BOOL Mainwnd_onCreate (mWidget* self, DWORD dwAddData)
{

        if (hPackage){
                /*从资源包中加载图片*/
                ncsGetBitmap (HDC_SCREEN, hPackage, IDB_MALE_HEAD,   &bmp_male);
                ..................
        }
        initListView (self);//用于初始化 mListView，建立列，建立组填充内容。
        return TRUE;
}
```

主窗口的 onClose 的处理函数

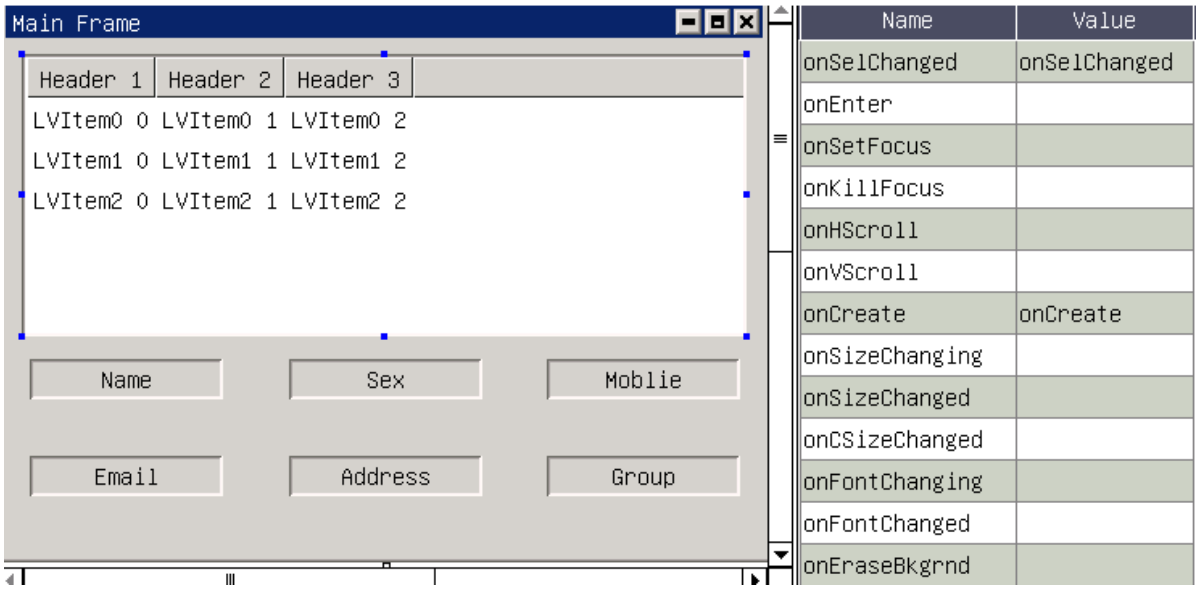
在主窗口的 onClose 处理函数中，释放掉所有的图片资源。

```
static BOOL Mainwnd_onClose (mWidget* self, int message)
{
        /*在窗口关闭的时候，将图片都卸载掉*/
```

```
        UnloadBitmap (&bmp_male);

        ..................



        return FALSE; /*continue default message process*/
}
```

mListView 关联事件以及处理函数

为 mListView 控件作事件关联，其中关联了 onCreate 和 onSelChanged 事件，下面将介绍这两个事件的处理函数。



控件 mListView 的 onCreate 处理函数

在控件 mListView 的 onCreate 处理函数中，将 item Destroy 的回调函数、header 自绘制回调函数、以及 Item 自绘制的回调函数进行关联。

```
static NCS_CB_DRAWITEM old_itemdraw;

//$func @2586458112 onCreate

static BOOL Listview_onCreate (mWidget* self, DWORD dwAddData)

{

        mListView *listview;
```

```
        listview = (mListView*)self;
        /*设置 item Destroy 的回调函数*/
        _c(listview)->setItemDestroy (listview, item_destroy);
        /*设置 CustomDrawHeader 的自绘制回调函数*/
        _c(listview)->setCustomDrawHeader (listview, &listv_cstm);
        /*设置 Item 自绘制的回调函数，并将旧的绘制函数保存在 old_itemdraw 中*/
        old_itemdraw = _c(listview)->setItemDraw (listview, new_itemdraw);
        //TODO:
        return TRUE;
}
```

自定义的 Item Destroy 函数

该函数将在 Item 销毁的时候被自动调用，用于释放存储 Item 附加数据的空间。

```
static void item_destroy (mItemView *self, HITEM hItem)
{
        USER_DATA* adddata;
        mListView* listview;

        listview = (mListView*)self;
        adddata  = (USER_DATA*)_c(listview)->getAddData (listview, hItem);
        if (adddata != 0){
                /*在 item 销毁的时候，释放掉这些内存*/
                free (adddata->name);
                free (adddata);
```

```
            ……
        }
}
```

自定义的 NCS_CB_LISTV_DRAWHDRBK 和 NCS_CB_LISTV_DRAWHDR

自定义的这个两个函数，分别用于绘制 mListView 上的 header 部分和在 header 上每列的文字。

```
static void mlistv_cstm_drawhkrbk (mListView *self, HITEM hHdr, HDC hdc, RECT *rcDraw)
{
        /*绘制 header 部分*/
        FillBoxWithBitmap (hdc, rcDraw->left, rcDraw->top, 435, 22, &bmp_banner );
}

static void mlistv_cstm_drawhr (mListView *self, HITEM hHdr, HDC hdc, RECT *rcDraw)
{
        mListColumn *listcolumn;
        SetBkMode (hdc, BM_TRANSPARENT);

        listcolumn = (mListColumn*)hHdr; //将 hHdr 强制转换为(mListColumn*)。
        /*使用 mListColumn 中的函数 getItemString 获得 Item 中的字符串信息*/
        TextOut (hdc, rcDraw->left, rcDraw->top, _c(listcolumn)->getItemString (listcolumn));
}

static NCS_CB_LISTV_CSTMHDROPS listv_cstm = {mlistv_cstm_drawhkrbk, mlistv_cstm_drawhr};
```

自定义的 Item 自绘制函数 new_itemdraw 实现

new_itemdraw 为 Item 的自绘制函数，其中 old_itemdraw 为原有的 Item 绘制函数，在这里使用的作用就是：首先进行原有绘制；然后再给每一行绘制一个黑框。

```
static void new_itemdraw (mItemView *self, HITEM hItem, HDC hdc, RECT *rcDraw)

{

        USER_DATA* adddata;

        /*首先使用默认绘制函数进行绘制，这样就可以在原有的基础上进行绘制了*/

        old_itemdraw (self, hItem, hdc, rcDraw);

        /*再为每项画一个矩形*/

        Rectangle (hdc, rcDraw->left, rcDraw->top, rcDraw->right, rcDraw->bottom);

}
```

控件 mListView 的 onSelChanged 处理函数

在控件 mListView 的 onSelChanged 处理函数中，当 mListView 被选中的时候，会得到高亮的 HITEM，并从其中获取附加数据，然后再显示到 mListView 下方的 static 控件中。

```
//$func @2586458112 onSelChanged -- Need by merge, don't modify

static void Listview_onSelChanged (mWidget* self, int id, int nc, DWORD add_data)

{

        ..............



        listview = (mListView*)self;

        switch (nc) {

                case NCSN_LISTV_SELCHANGED:

                hItem   = (HITEM)_c(listview)->getHilight(listview);

                /*获得当前 item 的附加数据*/

                adddata = (USER_DATA*)_c(listview)->getAddData(listview, hItem);
```

```
                if (adddata == 0){
                        clear_static (self); //如果为空则清空那些 static
                } else {
                        display (self, adddata); //如果带有附加数据则使用这些附加数据
                }
                break;
        }
}
```

第三部分第三章 自己动手开发 Button 控件

NCS 自定义控件的开发

NCS 具有很强的扩展性，用户可以根据自身需求，随心所欲的开发 NCS 自定义控件。
下面我们将讲解如何在 NCS 中自定义控件。

NCS 的继承机制

在 NCS 中自定义控件，就不得不介绍 NCS 的继承机制，因为它可以方便开发者利用现有类进行高效率的开发。
NCS 的框架是基于 OO 思想的，利用 C 语言的函数指针模拟 C++中的虚拟函数实现多态。
在继承上，实现 C 用宏来模拟继承，即保持子类结构的前半部分与父类结构在语义方面和语法方面都保持一致 。
一个控件由三部分组成：

- 1. 控件的属性集合
- 2. 控件的虚函数表
- 3. 控件渲染器的接口

例如 mWidget：
mWidget 是所有控件的基础类 ，继承自: mComponent 。
类 Wideget 介绍

自定义控件的功能需求

自定义一个 NCS button 控件，控件的外观使用两张图片实现（如上图），这两张图片分别代表按钮的按下和抬起。
当 button 在按下或抬起的时候分别更换不同的图片，表示控件自身处于的不同状态，同时会向父窗口发送通知事件。

选择那个类作为自定义控件的基类

作为新控件的开发，需要选择一个适合自身需求的基类，我们这里选择 Widget 作为基类。

开始自定义控件的编码工作

自定义控件的命名

对于自定义控件的命名，NCS 中有详细的规范。参见：新控件集的接口命名规范
在这里我们将新控件命名为"newbutton"。

```
#define NCSCTRL_NEWBUTTON    NCSCLASSNAME("newbutton")
```

自定义控件的定义

在这里定义，自定义控件中需要使用的变量和虚函数。从下面的定义中可以看到，Header、ClassHeader、RendererHeader 都继承自 Widget 类。

```
/*在这里定义了，两张图片的 PBITMAP，和 button 所处的状态标实*/

#define mNewButtonHeader(clsName) \

mWidgetHeader(clsName)        \

PBITMAP bmp_nor;              \

PBITMAP bmp_up;               \

int status;


#define mNewButtonClassHeader(clsName, parentClass)           \

mWidgetClassHeader(clsName, parentClass)


#define mNewButtonRendererHeader(clsName, parentClass)  \

mWidgetRendererHeader(clsName, parentClass)
```

```
MGNCS_EXPORT extern mNewButtonClass g_stmNewButtonCls;
```

自定义控件的 Property 和 Notify

定义自定义控件的属性、通知消息。因为该类的父类是 Widget 所以，要从*_WIDGET_MAX + 1 开始定义。

```
enum mNewbuttonProp {
        NCSP_NEWBTN_IMAGE_NORMAL = NCSP_WIDGET_MAX + 1,
        NCSP_NEWBTN_IMAGE_DOWN,
        NCSP_NEWBTN_MAX
};

enum mNewButtonNotify {
        NCSN_NEWBTN_PUSHED = NCSP_WIDGET_MAX + 1
};
```

自定义控件中虚函数的实现

```
/*控件中定义的枚举类型，表示 button 的两个状态*/
enum mNewButtonStatus {
        BTN_NORMAL    = 1,
        BTN_PUSHED
};
```

在头文件中定义完了，变量和虚函数，就需要在 C 文件中实现这些虚函数，并使用这些变量了。

```
/*类 mNewButton 的构造函数*/
static void mNewButton_construct(mNewButton *self, DWORD param)
{
        Class(mWidget).construct((mWidget*)self, param);
```

```
}

/*类 mNewButton 的 onLButtonDow 和 onLButtonUp 函数，其作用是接收鼠标事件，并通知父窗口。*/
static int mNewButton_onLButtonDown (mNewButton* self, int x, int y, DWORD keyFlags)
{
        SetCapture (self->hwnd);
        self->status = BTN_PUSHED;
        InvalidateRect(self->hwnd, NULL, TRUE);
        return 0;
}

static int mNewButton_onLButtonUp (mNewButton* self, int x, int y, DWORD keyFlags)
{
        if (GetCapture() == self->hwnd){
                self->status = BTN_NORMAL;
                InvalidateRect(self->hwnd, NULL, TRUE);
                ReleaseCapture ();
                ncsNotifyParent((mWidget*)self, NCSN_NEWBTN_PUSHED);
        }
}

/*类 mNewButton onPaint 的处理函数，其作用是根据 button 的不同状态，显示不同的图片*/
static void mNewButton_onPaint (mNewButton *self, HDC hdc, const PCLIPRGN pinv_clip)
{
```

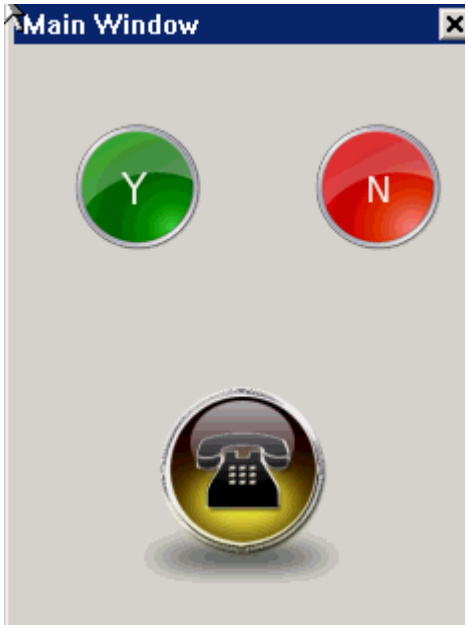
```
        RECT rect;
        GetWindowRect (self->hwnd, &rect);
        switch (self->status){
                case BTN_NORMAL:
                FillBoxWithBitmap (hdc, 0, 0, RECTW(rect),
                RECTH(rect), self->bmp_nor);
                break;
                case BTN_PUSHED:
                FillBoxWithBitmap (hdc, 0, 0, RECTW(rect),
                RECTH(rect), self->bmp_up);
                break;
        }
}

/*类 mNewButton setProperty 的处理函数，其作用是加载图片，并将其显示出来。*/
static BOOL mNewButton_setProperty (mNewButton *self, int id, DWORD value)
{
        char *str;
        if (id > NCSP_NEWBTN_MAX){
                return FALSE;
        }
        str = (char*)value;
        switch (id){
                case NCSP_NEWBTN_IMAGE_NORMAL:
```

```
                self->bmp_nor = LoadBitmapFromRes (HDC_SCREEN, str);
                if (self->bmp_nor == NULL){
                        printf ("Cant load %s\n ", str);
                        return 1;
                }
                break;

                case NCSP_NEWBTN_IMAGE_DOWN:
                self->bmp_up = LoadBitmapFromRes (HDC_SCREEN, str);
                if (self->bmp_up == NULL){
                        printf ("Cant load %s\n ", str);
                        return 1;
                }
                break;
        }
        InvalidateRect(self->hwnd, NULL, TRUE);
        return Class(mWidget).setProperty((mWidget*)self, id, value);
}

/*类 mNewButton 在销毁的时候使用的 destroy 函数，其作用是释放使用的图片资源*/
static void mNewButton_destroy(mNewButton *self)
{
        ReleaseRes (Str2Key (self->bmp_nor));
        ReleaseRes (Str2Key (self->bmp_up));
```

```
}

/*类 mNewButton onCreate 的处理函数，其作用是将 button 的 status 置为 BTN_NORMAL*/

static BOOL mNewButton_onCreate (mNewButton *self, LPARAM lParam)

{

        self->status = BTN_NORMAL;

}
```

自定义控件效果

下面就是自定义控件在窗口上的显示效果。
图中定义了三个 newbutton 控件。

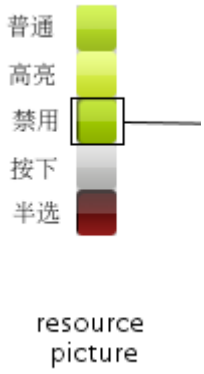


附录 A 新控件集 Skin 渲染器图片规格

前言

- **skin 的所有元素的类型都是文件名，该文件名对应的是加载到系统缓冲池中的图片(通过 LoadResource)。两者必须一致**
- skin renderer 实际不保存文件名，只保存由这些文件名生成的 RES_KEY,然后通过 GetResource 获取图片

- 由于很多控件有 normal、press、highlight、diable 等等状态的区分，对一个控件而言，每一个状态都保存成一幅图片，对资源的保存和图片加载的过程都是很大的浪费，所以 mgncs 中 skin 渲染器沿用 minigui 中的图片使用方式，就是将相关联的多张图合成一张，绘制时扣取不同的部分使用，例如，对 button 图片,我们设计成五个部分的组合图片，当我们要绘制某一个状态的 button 时，就取这一部分的子图来使用：



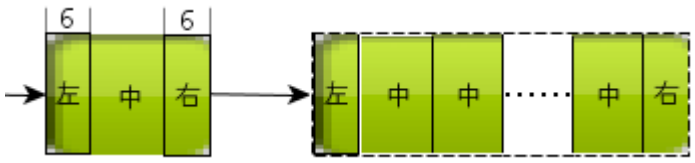
- 图片的绘制方式有两种:

  直接拉伸填充绘制

  这种方式不难理解，就是把资源图片按照绘制区域的大小以一定的比例进行放大来填充。

  分段填充绘制

  这种方式会将取到的图片（或者子图）按照上、中、下或者左、中、右的方式分成三段，然后把两端单独填充，中间部分则循环使用用中间部分来进行填充，我们依然以 button 为例，如图



  这种方式能较好得保留“圆角”等边界效果，使用这种方式填充的图片设计时要注意：

  1. 左中右、上中下 3 段的分界线每个图片是不同的，下面会详细说明

2. 中间部分由于要循环使用，注意过渡平滑一些

公用图片属性

属性名	类型	说明
NCS_IMAGE_ARROWS	文件名	箭头图片
NCS_IMAGE_ARROWSHELL	文件名	箭头按钮图片

- **图片的规格**
  - arrow 图片用于 skin 渲染器中箭头的绘制，由自上而下的 16 部分小图构成,每个小图都是一个正方形，内部的对应关系是
    - 0~3：向上的箭头的各个状态（0-普通、1-高亮、2-按下、3-禁止）
    - 4~7：向下的箭头的各个状态（4-普通、5-高亮、6-按下、7-禁止）
    - 8~11：向左的箭头的各个状态（8-普通、9-高亮、10-按下、11-禁止）
    - 12~15：向右的箭头的各个状态（12-普通、13-高亮、14-按下、15-禁止）
  - arrow_shell 图 spin 和 scroll 系列中，用于实现 arrow 按钮的效果，由自上而下的 4 部分小图构成,每个小图都是一个正方形，每一部分对应按钮的一种状态：
    - 0-普通
    - 1-高亮
    - 2-按下
    - 3-禁用
  - 另外，arrow 图一般是配合 arrow-shell 图使用的，使用时有叠加的处理，所以 arrow 周围的区域一般做成透明的。
  - 使用该图片采用直接拉伸填充，注意图片合理设计。

mButton

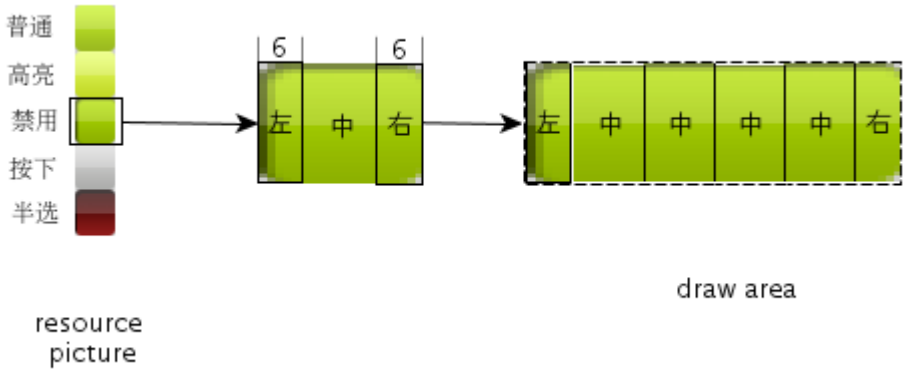
属性名	类型	说明
NCS_IMAGE_PUSHBUTTON	文件名	push button 的图片

图片的规格

- 用于 skin 渲染器中 pushbutton 的绘制效果，图片由自上而下的 5 部分组成，每一部分都是一个长方形的 button，分别对应 pushbutton 的五种状态：
  - 0 - 普通
  - 1 - 高亮
  - 2 - 按下
  - 3 - 禁用

- 4 - 三态按钮中的“半选”状态

- 绘制的时候，使用图片进行分段填充的，左右两端均为 6 个像素宽，中间一段宽度不限，注意图片的合理设计。



mCheckbutton

属性名	类型	说明
NCS_IMAGE_CHECKBUTTON	文件名	check button 的图片

图片的规格

- 用于 skin 渲染器的 checkbutton 的绘制渲染，图片由自上而下的 8 部分组成，每一部分都是一个长方形，分别对应 checkbutton 的 8 种状态：
  - 0~3:未选中时的普通、高亮、按下、禁用状态
  - 4~7:选中时的普通、高亮、按下、禁用状态
- 若图片大于绘制区域，会缩小图片进行绘制，否则采用直接用图片大小来填充，注意图片设计
- 示例

mRadiobutton

属性名	类型	说明

NCS_IMAGE_RADIOBUTTON	文件名	radio button 的图片

*图片的规格与 **checkbutton** 相同*

mListView

属性名	类型	说明
NCS_IMAGE_TREE	文件名	tree 中折叠/展开开关的图片
NCS_IMAGE_HEADER	文件名	表头部分的图片

图片的规格

- 用于实现树形控件中的折叠、展开按钮的渲染效果，图片由等份的上下两步分组成，上半部分是折叠的效果，下半部分是展开的效果
- 使用时，会直接以图片的大小来填充，不会拉伸，所以注意图片设计的尺寸，不要过大或过小
- 示例 

mPropSheet

属性名	类型	说明
NCS_IMAGE_TAB	文件名	PropSheet Tab 页的图片

图片的规格

- 用于渲染 propsheet 页的 tab 效果，由自上而下的四部分组成，mgncs 中使用到的只有第一、三两部分，分别表示：
  - 0 - inactive 页的 tab 效果
  - 2 - active 页的 tab 效果
- 分段绘制，左、中、右分别占 3、x、4 个像素，注意图片的设计
- 示例 

mProgressBar

属性名	类型	说明
NCS_IMAGE_PB_HCHUNK	文件名	水平 ProgressBar 的 chunk 图片
NCS_IMAGE_PB_VCHUNK	文件名	垂直 ProgressBar 的 chunk 图片
NCS_IMAGE_PB_HSLIDER	文件名	水平 ProgressBar 的 trackbar 图片
NCS_IMAGE_PB_VSLIDER	文件名	垂直 ProgressBar 的 trackbar 图片

图片的规格

- h_chunk
  - 用于水平进度条的进度显示，使用上 block 风格的 progressbar 分块拉伸填充，smooth 风格直接拉伸填充
  - 图片设计上建议不要水平渐变
  - 示例 
- v_chunk
  - 用于竖直进度条的进度显示，使用上 block 风格的 progressbar 分块拉伸填充，smooth 风格直接拉伸填充
  - 图片设计上建议不要竖直渐变
  - 示例 
- h_slider 未使用
- h_slider 未使用

mTrackbar

属性名	类型	说明
NCS_IMAGE_TB_HSLIDER	文件名	水平 Trackbar 的 Slider 图片
NCS_IMAGE_TB_VSLIDER	文件名	垂直 Trackbar 的 Slider 图片
NCS_IMAGE_TB_HTHUMB	文件名	水平 Trackbar 的 Trackbar 图片
NCS_IMAGE_TB_VTHUMB	文件名	垂直 Trackbar 的 Trackbar 图片

图片的规格

- h_slider
  - 用于水平 trackbar 的滑轨渲染，图片作为整体来使用，

  - 使用时采用分段填充处理，左右两段各为两个像素宽,中间部分循环使用，图片设计注意。
  - 示例
- v_slider
  - 用于竖直 trackbar 的滑轨渲染，图片作为整体来使用，
  - 使用时采用分段填充处理，上下两段各为两个像素宽,中间部分循环使用，图片设计注意。
  - 示例
- h_thumb
  - 用于水平 trackbar 的滑块渲染，图片有自上而下均分的四部分组成，分别对应滑块的普通、高亮、按下、禁用四种状态使用
  - 图片使用时直接拉伸填充，注意图片设计
  - 示例

- v_thumb
  - 用于竖直 trackbar 的滑块渲染，图片有自上而下均分的四部分组成，分别对应滑块的普通、高亮、按下、禁用四种状态使用
  - 图片使用时直接拉伸填充，注意图片设计
  - 示例

附录 B 公共结构和定义

AlignValues

- 水平对齐枚举值

```
enum enumNCSAlign{
        NCS_ALIGN_LEFT = 0,
        NCS_ALIGN_RIGHT,
        NCS_ALIGN_CENTER
};
```

- 垂直对齐枚举值

```
enum enumNCSVAlign{
        NCS_VALIGN_TOP = 0,
        NCS_VALIGN_BOTTOM,
        NCS_VALIGN_CENTER
};
```

ImageDrawModeValues

```
enum enumNCSImageDrawMode{
        /*普通方式绘制， 不拉伸也不平铺，默认情况下，居中（水平和垂直方向）显示图片。
        对于某些控件，如 Image，可以通过控件本身对齐设置来改变*/
        NCS_DM_NORMAL = 0,
        /* 拉伸或者缩写图片，以适应填充区域
        */
```

```
        NCS_DM_SCALED,
        /* 当图片的大小小于被填充区域时，平铺填满整个区域。平铺时，以行优先填充
        */
        NCS_DM_TILED
};
```

附录 C 新控件集的开发规范

术语及其含义

- MiniGUI 新控件集（MiniGUI New Control Set，mGNCS）。新控件集不仅仅实现了多种可定制、可扩展的控件类，同时也实现了资源管理功能，以及对其他常用功能的面向对象封装，如主窗口、对话框、菜单等。
- 风格（Style）。用于控制窗口或者控件基本外观和行为的标志。
    - 新控件集的控件风格，仅用于不可动态修改的控件属性，例如进度条是垂直的还是水平的，就可以用风格来实现。在各控件的实现中，应尽量减少风格的使用。
    - 除保留 WS_、WS_EX_ 等系统全局风格之外，MiniGUI 固有控件集定义的各控件风格不在新控件集中使用，新控件集中的控件风格将被重新定义。
    - 新控件集中，控件不再具有扩展风格。
- 属性（Property）。控件属性是一个新增的概念，一般用于那些可动态设置（如编辑框中的插入符位置），或动态改变（如列表框中的列表项数目）的控件特性。mGNCS 提供统一的接口（getProperty/setProperty）来获取或设置控件的属性。
- 渲染器（Renderer）。在 mGNCS 中，指控制主窗口或者控件外观的绘制方法集合，全称为“外观渲染器（Look and Feel Renderer）”，一般简称为“渲染器”。
    - 在新控件集中，每种可见的控件都会有自己的渲染器。为了和 MiniGUI 3.0 引入的渲染器相区别，我们将 MiniGUI 3.0 的渲染器称为“全局渲染器（global renderer）”，将新控件集引入的渲染器称为“控件渲染器（control renderer）”：
    - 全局渲染器作用于主窗口、老控件集以及新控件集的系统组件（如边框、标题栏、滚动条等非客户区元素）。
    - 控件渲染器仅作用于新控件集各个控件的客户区绘制。
- 窗口类名（Window Class Name）。为了和控件类的名称相区别，我们将 MiniGUI 用于子窗口的类名称为“窗口类名”。如静态框的控件类名为 `mStatic` ，而静态框的窗口类名为 `static_` 。
- 事件（Event）。在 MiniGUI 的传统编程模式中，存在消息（Message）和通知（Notification）两个概念，它们有各自的作用，但对应用程序开发者而言，它们均用于应用程序处理特定的事件。因此，从应用开发者看来，通知也好，消息也好，本质上都是一样的。因此，mGNCS 将消息和通知抽象为事件，从应用开发者角度看，每一个事件对应一个事件的处理器（Event Handler）。

新控件集的接口命名规范

新控件集的接口命名通常和类的名称有关，因此，在本文后面的规范描述中，类名的全小写方式、全大写方式、大小写混写方式、大写简写方式分别用 <classname>、<CLASSNAME>、<ClassName>、<CLSNM> 表示。

- 类名（指包含有方法，即函数指针的结构体）。新控件虽然采用 C 语言实现，但因为采用 OOP 思想，故而采用大小写混写的方式表示“类”的名称，并使用小写字母 m 作为前缀，如 mWidget ，以及对应的 mWidgetClass 。
- 一般性结构体。使用类似 MiniGUI 接口的全大写方式命名，并使用 NCS<CLSNM>_ 作为前缀。
- 函数及参数的命名规范。
    - 新控件集新增全局的一般性函数，使用 ncs 作为函数名前缀，之后是描述函数功能的动宾短语，并使用大小写混写的方法命名，如 ncsCreateWindow 。
    - 面向控件开发者的函数接口，第一个参数应该是 mWidget* 或者对应的子类类型指针。
    - 控件类的方法命名：动宾短语，单词连写，除第一个单词外，其他单词的第一个字母大写，如 void (*moveContent) (mWidget *) 。
    - 控件类中的消息回调函数的命名规范为： on<MessageName> ，如 void (*onSetFocus) (mWidget *) 。
    - 控件消息的参数顺序和约定：按照 MiniGUI 定义的消息规范，定义其顺序和名字。
    - 控件渲染器接口的命名规范为： <className>Renderer 。
    - 渲染器接口的内部函数参数约定：第一个是 mWidget* 或者对应的子类类型指针，第二个参数一般为 HDC 。
    - 在新控件集中，凡是和 C++ 成员函数等同的函数，表示本对象的指针变量命名为 self ，而本对象对应的类指针为 _class 。
    - 函数接口中的参数名称，均使用 Java JDK 的风格进行命名（第一个单词全小写，其后的单词大小写混写），不再使用匈牙利命名法。
    - 只读的指针型参数类型和返回值类型，必须使用 const 关键词做修饰。
- 窗口类名。为了不和老控件集的窗口类名冲突，新控件集使用下划线（_）作为窗口类名的后缀，如 static_ 。不使用前缀的原因，是为了在现有 MiniGUI 核心管理窗口类代码的算法下，尽量避免出现哈希冲突。窗口类名的 C 语言宏名称，则以 NCSCTRL_ 作为前缀，如 NCSCTRL_STATIC 。

需要注意的是，由于我们采用的是用 C 语言来实现类似 C++ 语言的面向规则特性，为了防止不必要的错误，我们需要把命名规范当做语法规范来遵守，这样的目的在于：

- 通过各种宏来简化编程。
- 便于通过宏固定编程模式，按照固定的编程模式，易读，易实现，不易出错。
- 编译文档的生成和检索。
- 最大限度减少开发者的学习时间。

类名称及其标识符的定义规则

类名称、对应的窗口类名及其简写形式由下表给出：

类名称	C 类名	窗口类名 Window classname	类名简写 CLSNM	备注
超类	mObject	-	OBJ	非控件类；NCS 类层次关系中的最基

				础类
列表项基类	mItem	-	ITEM	非控件类；用于表述列表项的基础类
列表项管理器	mItemManager	-	ITMMNG	非控件类；用于管理列表项集合的方法类
列信息类	mListColumn	-	LSTCLM	非控件类；描述列信息类
行列表项类	mListItem	-	LSTITM	非控件类；管理一行中的列表项集合
组件基类	mComponent	-	CMPT	非控件类
不可见组件基类	mInvsbComp	-	IVCMPT	非控件类
定时器	mTimer	-	TIMER	非控件类
控件基类	mWidget	widget_	WIDGET	
静态框	mStatic	static_	STATIC	
图片框	mImage	image_	IMAGE	
矩形框	mRect	rect_	RECT	
发光二极管标签	mLEDLabel	ledlabel_	LEDLBL	
分组框	mGroupBox	groupbox_	GRPBOX	
按钮组	mButtonGroup	buttongroup_	BTNGRP	
按钮	mButton	button_	BUTTON	
检查钮	mCheckButton	checkbutton_	CHKBTN	
单选钮	mRadioButton	radiobutton_	RDOBTN	
面板	mPanel	panel_	PANEL	
组合框	mCombobox	combobox_	CMBOX	
主窗口	mMainWnd	mainwnd_	MNWND	
对话框	mDialogBox	dialogbox_	DLGBOX	
可滚动基类	mScrollWidget	scrollwidget_	SWGT	
容器	mContainer	container_	CTNR	
属性页	mPage	page_	PAGE	

列表型基类	mItemView	itemview_	ITEMV
列表型	mListView	listview_	LISTV
图标型	mIconView	iconview_	ICONV
滚动型	mScrollView	scrollview_	SCRLV
列表框	mListBox	listbox_	LSTBOX
进度条	mProgressBar	progressbar_	PRGBAR
属性表	mPropSheet	propsheet_	PRPSHT
滑动器基类	mSlider	slider_	SLIDER
滑动条	mTackBar	trackbar_	TRKBAR
旋钮基类	mSpinner	spinner_	SPNR
旋钮框	mSpinBox	spinbox_	SPNBOX
分隔栏	mSeperator	seperator_	SPRTR
月历	mMonthCalendar	monthcalendar_	CDR
动画	mAnimation	animation_	ANMT
工具栏	mToolBar	toolbar_	TLBAR
滚动条	mScrollBar	scrollbar_	SCRLBR
编辑框基类	mEdit	edit_	EDIT
单行编辑框	mSLEdit	sledit_	SLEDIT
多行编辑框	mMLEdit	mledit_	MLEDIT

标识符的命名规则如下：

- 窗口类名标识符。为了不和老控件集的标识符名称冲突，新控件集使用 NCSCTRL_<CLASSNAME> 作为窗口类名标识符。
- 控件风格标识符。为了不和老控件集的风格名称冲突，新控件集使用 NCSS_<CLSNM>_ 作为风格标识符的前缀。
- 控件属性标识符。新控件集使用 NCSP_<CLSNM>_ 作为控件属性标识符的前缀。
- 控件通知码标识符。为了不和老控件集的通知码名称冲突，新控件集使用 NCSN_<CLSNM>_ 作为控件通知码标识符的前缀。
- 类特有数据标识符。新控件集使用 NCSSPEC_<CLSNM>_ 作为类特有数据的标识符前缀；如 mObject 类的特有数据，使用 NCSSPEC_OBJ_ 作为前缀。

- 类方法的参数及返回值所使用的标志、状态等。新控件集使用 NCSF_<CLSNM>_ 作为这种标识符的前缀。
- 函数接口及类方法的返回值标识符。新控件集使用 NCSE_<CLSNM>_ 作为这种标识符的前缀。
- 其他标识符。除上述标识符之外，其他类型的标识符，参照上述方法确定。

控件风格标识符的定义规则

在新控件集中，控件风格个数应该保持到最小，控件风格的取值范围为 0x00000000 ~ 0x0000FFFF，以免和 MiniGUI 定义的窗口风格混淆。

控件属性标识符的定义规则

控件属性标识符的规则：

1. 在直系继承关系内，属性标识符的值不能重复，即子类的属性标识符不能覆盖父类及其祖先的属性标识符。
2. 旁系继承关系内，属性标识符可以重复，即一个控件类的属性标识符可以和其兄弟控件类的属性标识符重复。

在技术上，利用 C 语言的枚举类型定义属性标识符，各控件的属性标识符前缀见上表。

1. 每个控件属性标识符枚举变量中，最后一项定义为 NCSP_<CLSNAM>_MAX 。
2. 每个控件属性标识符从其父类的最大属性标识符加一的值开始定义。
3. 利用枚举型能够自动加一的特性，顺序定义各个属性。

例如，

```
// mWidget

enum mWidgetProp {

        NCSP_WIDGET_RDR,

        NCSP_WIDGET_MAX

};



// mStatic

enum mStaticProp {
```

```
        NCSP_STATIC_HALIGN = NCSP_WIDGET_MAX + 1,

        NCSP_STATIC_VALIGN,

        NCSP_STATIC_MAX

};
```

控件通知码的定义规则

控件通知码的规则类似属性标识符的规则。但需要注意的是，mGNCS 中不能使用 0 作为通知码。

新控件集的函数库以及头文件

尽管 mGNCS 是在开发 miniStudio 的过程中引入的，但 mGNCS 也可作为 MiniGUI 3.0 上的一个组件而被单独使用。因此，我们将新控件集按照 MiniGUI 的一个组件来管理，因而定名为 mGNCS，函数库的名称为 libmgncs，当前版本为 1.0.6，头文件在系统头文件路径的 mgncs/ 目录下保存，主要的头文件名称为 mgncs.h。

当前的 mGNCS 定义了大量的通用控件，在不久的将来，我们还将在当前 mGNCS 的基础上形成针对特定领域的扩展控件集合，这时，将形成一个独立的函数库，其名称采用 libmgncs4* 的方式命名，头文件仍将保存在系统头文件路径的 mgncs/ 目录下。