GoUI 操作系统软件开发指南

GoUI Operating System Software User Guide

（适用于 PS-LCD, Applied to PS-LCD）

Version **1.23**

Greatal

北京大器智成技术有限公司 2013

Beijing Greatal Technologies Co. Ltd.

Change History

Version	Date	Content	Owner
V1.00	20100523	Initial	QuickDevcie
V1.01	20100608	Update Hyperterminal setting	QuickDevice
V1.02	20100727	Update CTP protocol	QuickDevice
V1.03	20100920	Support JavaScript	QuickDevice
V1.04	20100929	Rewrite module description	QuickDevice
V1.05	20110121	Update company info	Greatal
V1.06	20110817	Update Designer guide	Greatal
V1.07	20110909	Update Designer Property guide	Greatal
V1.08	20110910	Introduce Designer Chinese property	Greatal
V1.09	20110923	Support beep function	Greatal
V1.10	20111020	Introduce common widget properties	Greatal
V1.11	20111031	Introduce global.js and Timer widget	Greatal
V1.12	20111120	Introduce PS-LCD emulator	Greatal
V1.13	20111202	Introduce style sheet	Greatal
V1.14	20120103	Update Designer usage	Greatal
V1.15	20120224	Introduce UserDefine protocol	Greatal
V1.16	20120308	Introduce usrRxByte() function	Greatal
V1.17	20120313	Comparison between CTP & usrdefine	Greatal
V1.18	20120329	Add more FAQs	Greatal
V1.19	20120507	Add user define protocol example	Greatal
V1.20	20120611	Add additional functionality statement Add Canvas widget.	Greatal
V1.21	20120703	Introduce multi-line edit widget	Greatal
V1.22	20130115	Introduce system widgets	Greatal
V1.23	20130309	Introduce image button and sysHardKeypad widget	Greatal

术 语

PS-LCD: Programmable **S**mart **LCD** 缩写，可编程智能 LCD 模组

GoUI: Greatal Object-oriented User Interface 的缩写，大器智成面向对象的图形界面操作系统名称

HMI: Human Machine Interface 的缩写，人机界面

Cooky: 大器智成开发的 LCD 图形界面卡（HMI Processing Module）

HC: Host Controller 的缩写，外部控制器

CTP: Cooky Talk Protocol 的缩写，HC 与 PS-LCD 的通讯协议

Designer: Windows 下可视化组态编程软件，用于编辑和生成人机界面

SPF: Super Parser File 的缩写，Designer 的输出界面文件，通过 FLEX 工具下载到 PS-LCD，即可运行

UPF: Udisk Packed File 的缩写，Designer 的输出界面文件（类似 spf 文件），通过 U 盘下载到 PS-LCD，即可运行

Greatal

目　录

Greatal

Greatal

第一章 简　介

1.1 什么是 PS-LCD

可编程智能 LCD（即 Programmable Smart LCD，简称 PS-LCD）是一种包含 LCD 显示屏、LCD 控制器、触摸屏、人机界面处理系统和通讯接口于一体的智能显示模组，通过可选的通讯接口与外部控制单元（如：51 单片机、ARM、DSP、PC、PLC、总线设备等）连接，实现系统的人机交互界面。由于 PS-LCD 搭载了专用图形操作系统 **GoUI**，通过 PC 上的可视化组态编辑软件，用户无需专业图形编程知识，”0”代码即可轻松设计和生成系统所需图形界面。

1.2 基于 PS-LCD 的产品应用

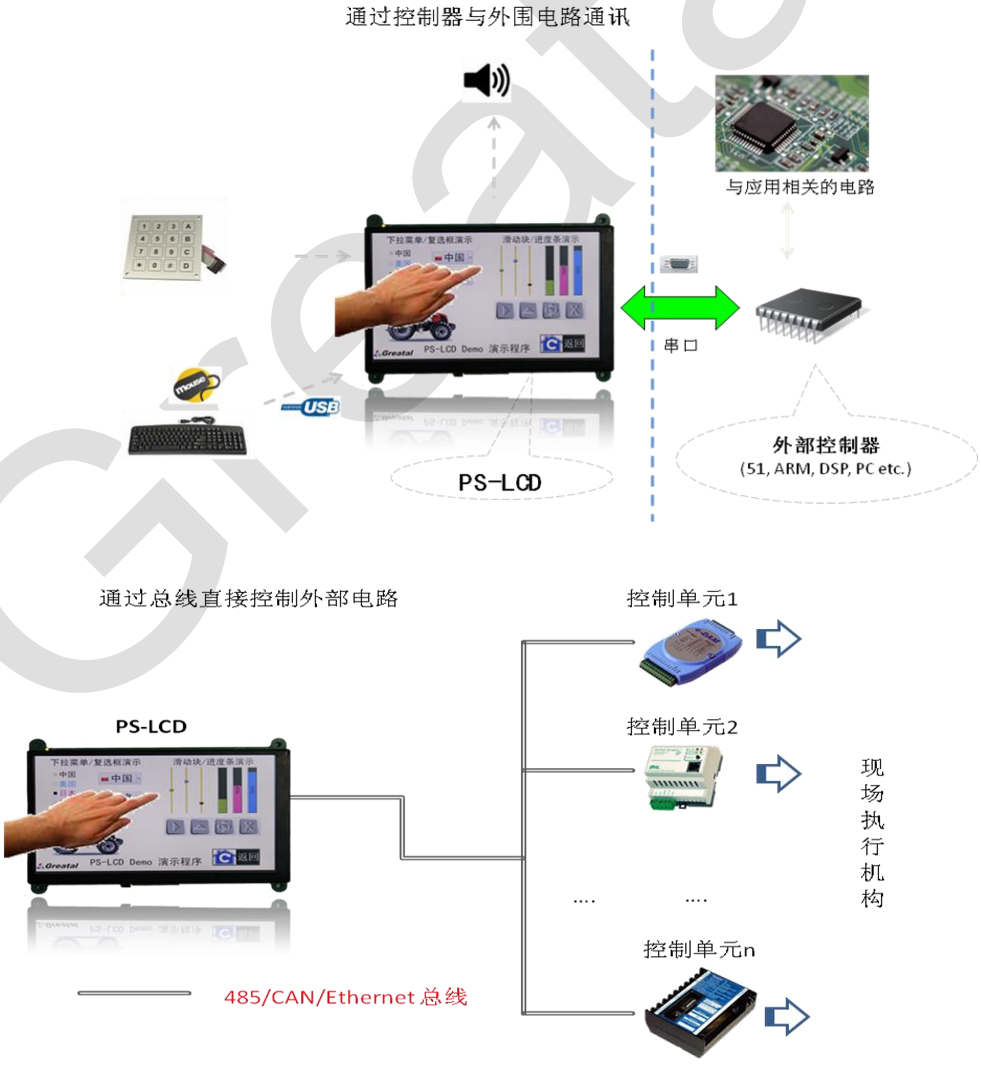


图 1-1 基于 PS-LCD 的产品应用示意图

典型的产品应用示意图见图 1-1，一般由两到三个部分组成：

- PS-LCD
- 外部控制器
- 外围应用电路

PS-LCD 硬件通讯接口简单，任何具有串口（或者 RS232、RS485、CAN 等总线接口）的控制单元都能轻松与之连接。

1.3 基于 PS-LCD 的产品软件

基于 PS-LCD 设计人机界面产品，无需编程，用鼠标即可设计出如下界面效果，如同制作 PPT 一样简单、直观！目前支持十多种常用控件（如文本框、按钮、进度条、仪表盘、曲线图等），可满足大部分应用需求。

图 1-2 PS-LCD 人机界面效果

与传统模式相比，有如下优势：

- **对控制单元性能要求低**，只要有通讯接口（如三线串口）的外部控制单元均可与 PS-LCD 连接。
- **开发难度低**，无需图形编程知识，用可视化组态式编辑软件，“所见即所得”、“0”代码设计和生成界面；通过 PS-LCD 软件模拟器，在没有硬件

情况下，可在 PC 上模拟实际界面运行效果，有效缩短界面开发周期；

- **占用控制单元资源少**，外部控制单元只需与 PS-LCD 交互应用相关数据即可，界面显示、触摸动作等行为均由 PS-LCD 上的 GoUI 操作系统按预设功能自动完成；
- **软件代码少**，外部控制单元只需三条指令或者函数与 PS-LCD 通信，快速集成界面；

Greatal

第二章 开发概述

2.1 开发步骤

通过大器智成专用可视化集成开发工具 Designer 轻松完成 PS-LCD 图形界面开发。该工具包含了开发所需的界面管理、配置、编辑、生成、模拟、下载等一系列软件功能，用户无需界面编程知识，即使无任何软件开发基础的用户也可快速完成产品界面开发，最大限度降低开发难度。

PS-LCD 界面开发过程可总结为如下三步：

1) 在 Windows 上用 Designer “所见即所得”、“0”代码生成界面；

2) 通过 USB 快速下载步骤一生成的文件 spf/upf 到 PS-LCD，然后重启 PS-LCD，界面即可运行；

3) 界面运行时，外部控制单元通过简单协议与界面通讯，快速集成产品。

图 2-1 基于 PS-LCD 的界面开发步骤

2.2 PS-LCD 工作模式

PS-LCD 可工作在两种模式，分别是：

- **下载模式（图 2-2）**

图 2-2 下载模式界面

在该模式下，采用大器智成专用软件工具 Flex 通过 USB 与 PS-LCD 连接，即可下载界面文件 spf，如图 2-3。

图 2-3 通过 Flex 下载界面文件到 PS-LCD 示意图

进入下载模式的方法：用专用下载线将 PS-LCD 与电脑相连，然后将 PS-LCD 复位或者重新上电，即进入下载模式。此时，PS-LCD 底板指示灯“两快一暗”闪烁，LCD 屏上出现下载图标。

- **界面模式**

选择下载软件 Flex 菜单“Tools”中“Start Cooky”可进入界面模式；或

者拔出 PS-LCD 上的下载线，复位或者重新上电，也可进入界面模式，主界面完全启动并显示大概需要 10 秒。此时，指示灯闪烁速度每秒一次。

在界面模式下，PS-LCD 主要完成：

1) 读取自定义 logo 图片并在启动过程中显示；

2) 待启动完成后，运行预先设计的人机界面；

3) 在人机界面运行过程中，外部控制单元可与人机界面通讯；

2.3 界面更新方式

将 Designer 生成的界面文件 spf 或者 upf 下载到 PS-LCD，即可完成界面更新。可选择 USB 下载或者 U 盘方式更新界面，二选一即可。

- **USB 下载更新（仅适用于 Windows XP）**

1) 进入下载模式；

2) 启动 Flex，点击 path 按钮，选择 spf 文件（扩展名为.spf），点击 Start 按钮，下载开始；

3) 待系统提示下载完成，点击 OK；

4) 重新启动 PS-LCD，进入界面模式，人机图形界面即开始运行。

- **U 盘更新（适用于 XP、Win7 及以上版本 Windows）：**

1) 确保 U 盘只有一个分区，而且在根目录下建立 goui 目录（必须小写）；

2) 将 Designer 生成的 upf 文件拷贝到 U 盘下的 goui 目录；

3) 当 PS-LCD 处于界面模式时，将 U 盘插入 USB 接口，大约 5 秒后，自动进入界面更新程序，更新完毕后，按提示拔掉 U 盘，PS-LCD 自动重启，新界面即运行，更新完毕。

2.4 指定启动界面 logo

可任意选择 24 bit 的位图图片（图片分辨率需要小于实际 LCD 分辨率）作为系统启动 logo，方法如下：

1) 进入下载模式；

2) 启动 Flex，点击 path 按钮，选择自定义的 logo 图片（扩展名为.bmp），点击 start 按钮，下载开始；

3) 待系统提示下载完成，点击 Ok；

4) 重新启动 PS-LCD，进入界面模式，即可看到步骤 2 下载的图片生效。

Greatal

第三章 设计界面

3.1 Designer 介绍

Designer 是一个 Windows 端的可视化界面集成开发环境，用于完成基于 GoUI 操作系统的智能显示产品界面开发。通过该工具，可设置运行参数、设计界面静态外观（如外观，字体，和尺寸等），定义动态行为，“所见即所得“、“0”代码快速完成最终产品的人机界面。甚至无需 PS-LCD 硬件，Designer 模拟器也可轻松完成界面效果验证。Designer 的功能区域划分如下：

图 3-1 Designer 功能区域示意图

菜单栏

文件->新建工程：新建一个项目工程；

文件->新建页面：在现有的项目工程中新建一个页面；

文件->打开 ：打开一个项目工程；

编辑 ：编辑界面时的拷贝、粘帖、删除等命令；

工具->界面配置 ：界面运行参数设定，选中即弹出如下对话框。

主页面——指定界面的主页面，界面启动后自动显示；

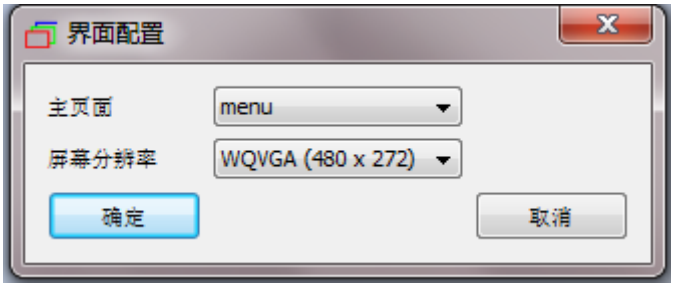


图 3-2 Designer 界面配置选项

屏幕分辨率——指定界面的分辨率，一般设定为与实际使用的 PS-LCD 分辨率一致；

工具->全局脚本：全局 JS 脚本，用于全局使用的变量和函数的定义。选中即弹出编辑器，可输入 JS 脚本，该脚本将在界面启动后最先运行一次，其中的所有定义变量和函数，在任何界面中可无条件访问。

工具->系统控件：设置与系统功能相关控件的属性。与用户控件不用，系统控件不支持拖拽操作，也不允许将用户控件拖放到系统控件设置界面。

图 3-3 系统控件设置界面

- **串口 0**——设置串口通讯类型，波特率，触发条件等参数；
- **界面管理**——设置桌面背景，调试模式，界面切换效果；
- **矩阵键盘**——监控外接矩阵键盘输入事件；

- **软键盘**——设置软键盘的输入方式，支持中英文和数字输入；
- **触摸屏**——设置触摸屏超时时间及相应动作；
- **背光**——设置 LCD 背光亮度；
- **蜂鸣器**——设置蜂鸣器动作时间及时间间隔；
- **环境变量**——存储用户数据，掉电后不丢失；

工具->生成界面： 生成界面文件（扩展名为 spf，该文件可直接下载到 PS-LCD 上运行）；

工具->下载界面： 启动 Flex 专用界面文件下载工具，可通过 USB 下载 spf 界面文件到 PS-LCD；

工具->模拟器： 启动 PS-LCD 软件模拟器，可模拟界面在最终产品上的实际运行效果和动态行为， 内嵌的串口模拟器可模拟实际串口收发数据；

选项->语言： 选择语言种类，目前支持简体中文和英文；

帮助->关于 Designer ：显示 Designer 版本信息；

帮助->检查更新 ：点击该选项，将访问大器智成软件发布页，查看最新版本软件信息；

图 3-4 PS-LCD 模拟器

工程管理区

浏览和管理工程页面文件。双击鼠标左键打开页面，右击鼠标，打开或者移除页面文件。

- **用户控件区**

该区列出了所有控件名称，通过按住鼠标左键拖拽某一个控件到页面编辑区，页面中即生成该控件，自由调整尺寸和外观，“所见即所得”。目前支持如下控件类型，可支持控件数还在不断增加中。

- 复选框——提供某一个选项供选择时使用；
- 下拉框——提供若干种选择项，每次只能选其中一种；
- 图片——显示特定图片，通过“源图片”属性来指定需要显示的图片名称，支持 bmp、jpg、png、tiff 和 gif 格式图片；
- 标签——显示文本信息；
- 进度条——指示实时进度状况；
- 文本按钮——操作按钮，可定义成自动重复或者单次有效；
- 图片按钮——外观为图片，但具有按钮功能，可定义按下和抬起图片；
- 滑动尺——通过滑动块位置来定义输入的数值；
- 单行文本框——编辑和显示文本，单行显示。在 PS-LCD 中运行界面时，点击该文本框，系统自动弹出软键盘，可直接输入数字、字母、符号等信息；
- 多行文本框——编辑和显示文本，可多行显示，显示内容超过控件尺寸，自动在右侧出现滚动条。在 PS-LCD 中运行界面时，点击该文本框，系统自动弹出软键盘，可直接输入数字、字母、符号等信息；
- 定时器——提供定时功能。当定时时间间隔到时，可使其执行一段脚本，从而实现动画和刷新界面等效果，该控件在界面运行时不可见；
- 仪表盘——用于模拟机械仪表，仪表的背景图片通过“源图片”属性指定。更换不同的背景可设计出不同的仪表界面；
- 段式数字——模拟数码管显示数字；
- 波形图——显示曲线波形，可最多支持 4 条波形同时显示；
- 刻度尺——带刻度显示的位置指示器；
- 画布——完成点、线、圆、矩形、多段线等基本图形的绘制；

- **页面编辑区**

页面的编辑区域，从控件区拖拽控件到该区域，然后用鼠标或键盘来完成控

件的位置布局，可进行拖拉、重复、恢复、移动、删除、重定义尺寸等操作。选中某个控件，然后通过控件属性区来设置控件的各项属性(如背景，颜色，大小，文字、风格、边框等)和定义事件动态行为。

对象管理区

包括当前页面编辑区所有控件对象名称列表，在这里可更改控件对象名称 ID，该 ID 作为控件的唯一标识，在 PS-LCD 与外部控制器通信，或者脚本编程中使用。双击第二列“<<<”，即可编辑该控件对象的动作脚本。当在页面编辑区选中某个控件时，对象管理区相应控件名称左侧出现蓝色指示。

控件属性区

列举出当前选中控件的所有属性，设置相关属性，可改变控件外观、风格和定义事件动作。可在界面运行时动态改变这些属性，在 Designer 中用鼠标指向蓝色属性名称，即会出现在线帮助。所有控件的属性分为四个大类：

- 外观——定义控件尺寸，字体，颜色等外观特性；
- 内容——定义文字内容，图标，风格类型等；
- 边框——某些控件有该类属性，定义边框的效果；
- 行为——定义控件的动态效果和动作，其中如下属性定义了界面在 PS-LCD 上运行时的动态行为：

动作脚本(Action): 点击该属性即可在弹出的脚本编辑框中输入类 C 脚本语言（JavaScript）。同时也可以在脚本编辑框的右上角设置脚本的触发条件，如下图所示。当界面在 PS-LCD 上运行时，满足设定的触发条件时，该脚本立即被执行，完成预设界面动作。

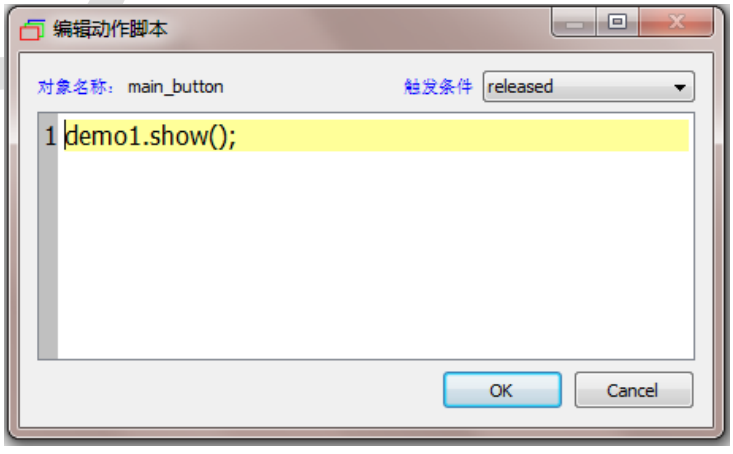


图 3-5 控件动作脚本编辑框

事件通知(Verbose)：定义是否自动向外部控制单元发送事件消息。如选中，当相应控件的动态属性发生变化，PS-LCD 自动通过通讯口向外部控制单元发送事件消息（如：当界面在 PS-LCD 上运行时，用户点击按钮控件，PS-LCD 通过串口发送如下消息到外部控制单元: E+xx.state=1, 其中 xx 为按钮控件名称）。**注意：*该属性只有选中 CTP 通讯协议时才有效*。**

快速显示(Fastshow): 页面(Form)控件特有属性。如选中该属性，界面在切换过程中始终保持在内存运行，可最大限度地提高切换到该界面的速度。否则，界面在切换时需从 flash 上读入内存再运行。

注意：如果过多页面设置该属性，有可能造成系统内存不足，系统运行速度缓慢。建议比较重要的页面设置该属性。

3.2 控件对象

PS-LCD 软件设计是完全基于对象概念，即所有界面元素和硬件功能都抽象为控件对象，通过各控件对象的属性和方法函数，实现系统界面和软件功能。控件分为系统控件和用户控件两种。

系统控件：将硬件相关的功能软件抽象为系统控件，如串口、触摸屏、背光、蜂鸣器等，一般在用户界面中不可见。系统控件由于与硬件一一对应，系统启动后自动创建，任何界面和控件可以无条件访问其属性和方法函数实现系统控制。用户只能对系统控件进行设置，而不允许增加、移动和删除。

用户控件：将不同的页面软件元素抽象为用户控件，如：文本框、按钮、图片、进度条等，一般在用户界面中可见。用户控件允许用户将控件拖拽到页面设计区创建、移动和删除，并且同一控件可以多次使用。系统启动时，根据用户的设计，动态加载和创建用户控件；访问其方法和属性时，必须确保控件所在的页面已经存在（当前页面或者有“**快速显示**”属性的页面）。

设计界面的静态外观过程主要就是利用现有用户控件如同搭积木一样构建不同功能页面。

3.3 动作脚本

Designer 不仅可设计界面静态外观、文字、风格类型等，而且也可通过控件

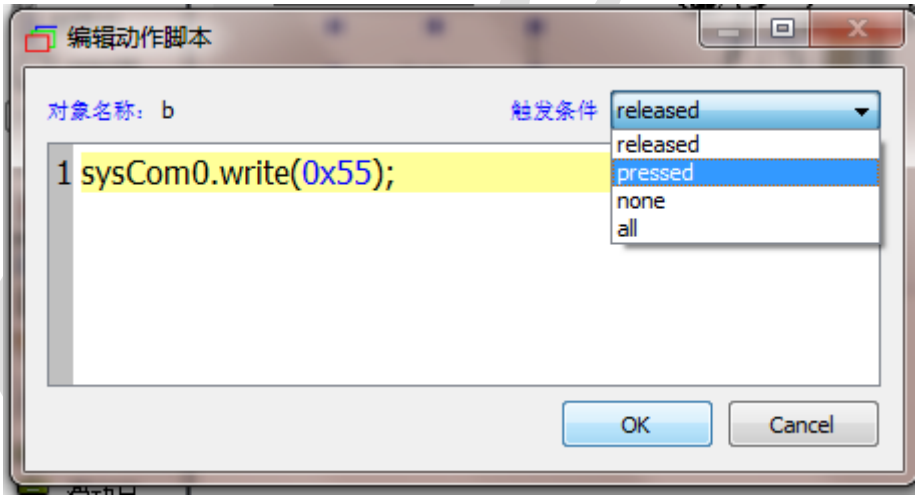
的“**动作脚本(action)**”属性实现界面运行时动态效果。实现方法是：在 Designer 中，选中控件，然后点击控件属性区“**动作脚本(action)**”或者点击对象管理区中的“<<<”，在弹出的编辑框输入动作脚本描述事件动作，选择脚本触发条件。

通过“**动作脚本**”程序，用户可以进行逻辑控制、运算、控件对象操作等来实现系统软件功能，大多数控件对象支持基于事件驱动动作脚本功能，即界面在运行过程中，满足一定控件事件条件后，无需外部干预，动作脚本自动触发执行，完成预设功能。这些事件条件类型我们称为**触发条件**。

设计界面动态效果其实就是基于触发条件设计不同控件动作脚本的过程。目前 PS-LCD 支持脚本类型为 JavaScript（语法类 C 语言，熟悉 C 语言的开发者很容易掌握）。

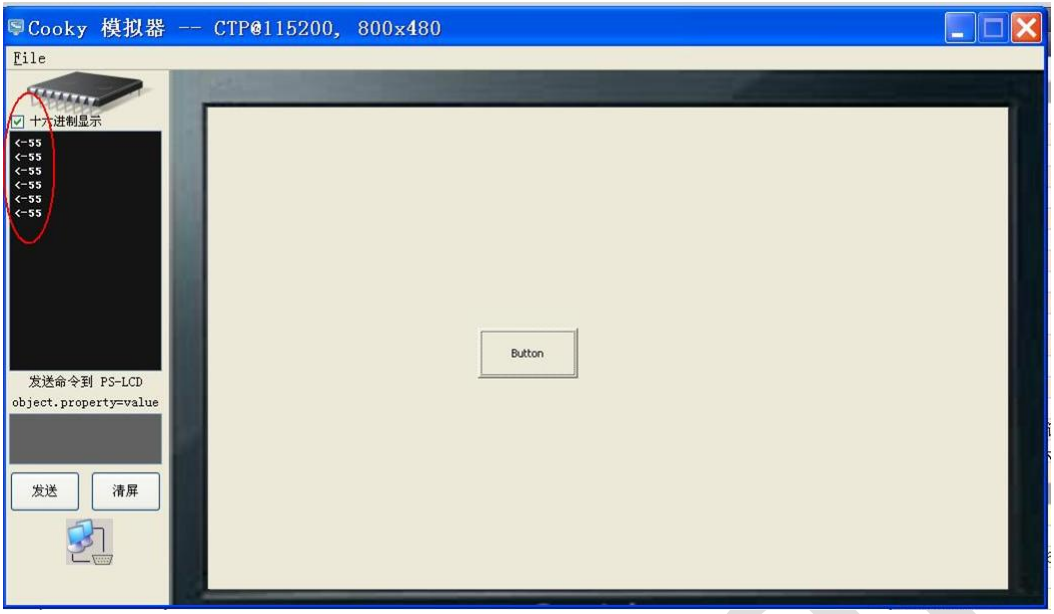
例如要实现如下界面功能：按下页面上某个按钮后向串口发送一个字节数据 0x55。在主界面上拖放一个按钮，命名为“b”，点击“**动作脚本**”，弹出动作脚本编辑框，在右上角的触发条件中选择 pressed 事件（即按下按钮时触发动作脚本）。在脚本编辑区输入如下代码：

```
sysCom0.write(0x55);
```



按钮控件的触发条件有 **released（抬起事件），pressed（按下事件），none（不触发），all（按下和抬起事件均触发）。**

运行 PS-LCD 模拟器，按下一次按钮，左边的模拟串口数据区会显示发送了一字节 16 进制数据 55。结果如下图。



注意：

- PS-LCD 仅支持 JavaScript 核心部分 ECMAScript（语法基本跟 C 一致）。ECMAScript 的详细说明参看如下连接的中的 ECMAScript 章节 http://www.w3school.com.cn/js/pro_js_syntax.asp；
- 在 PS-LCD 上调试 JavaScript 脚本非常简单直观。如果脚本有问题，界面在 PS-LCD 或者模拟器上运行时，屏幕上会自动弹出错误提示框（需打开“**系统控件**”的“**调试模式**”属性），根据提示的控件名称、错误类型、脚本行号等信息，用户可轻松找到脚本错误所在，一目了然。
- 控件属性、方法可直接在动作脚本代码中调用，具体用法参看“**软件速查表**”文档。

3.4 设计步骤

基本步骤如下：

1) 新建 Designer 工程，定义主页面名称和分辨率；

2) 生成主页面后，在系统控件中设置系统相关参数；

3) 在工程中“所见即所得”生成页面，设置背景、加入/设置控件、指定图片/动画、定义事件动作等；

4) 用 PS-LCD 模拟器验证界面效果和通讯过程。

5) 生成界面输出文件 spf/upf，然后将 spf 文件通过 PS-LCD 下载工具 Flex 下载到 PS-LCD 验证最终界面效果(或者将 upf 文件拷贝到用户自备的 U 盘 goui 目录下，PS-LCD 在界面模式时，将 U 盘插到 PS-LCD 上，5 秒后即开始更新界面，完成后拔掉 U 盘，系统自动重启，新界面运行)。

具体设计过程参看第五章。

Greatal

第四章 界面通讯

4.1 PS-LCD 通讯协议

PS-LCD 作为先进的智能人机界面产品，能通过通讯接口轻松灵活地与外部控制单元实现数据交互。目前，PS-LCD 支持两种通讯协议：**CTP**(Cooky Talking Protocol)协议和**用户自定义**（UserDefine）协议，**在运行时只能选择其中一种**，需在 Designer 中的“系统控件设置”界面的“**串口 0**“系统控件属性中指定。两种协议的比较见下表：

	CTP 协议	自定义协议
数据发送 (PS-LCD 传输数据到外部控制单元)	可选择两种方式中的一种: ▶ 在设计页面时，选中某个控件的行为属性中的”事件通知”，PS-LCD 在运行过程中，当控件事件产生时，自动发送数据消息，无需用户编写任何脚本；如：ID 为 xx 的按钮按下一次，PS-LCD 自动发送 xx.state=1 字符串消息； ▶ 不选择“事件通知”属性，与自定义协议发送方式一致	在设计界面时，在控件动作脚本中直接调用 **sysCom0.write()** 或者 **sysCom0.writeString()** 函数，实现在指定触发条件下发送任意自定义数据或者字符串。
数据接收 (外部控制单元传输数据到 PS-LCD)	外部控制单元发送有效的 JS 脚本程序字符串，PS-LCD 接收后自动执行。如：外部控制单元发送 xxx.text=”test”字符串（后必须加回车字符），名为 xxx 的控件文本内容自动刷新为“test”。任何有效的 JS 脚本程序都能通过通讯接口以字符串方式发送给 PS-LCD 立即执行。***用户无需编写接收处理程序***。	用户在设置好通讯协议类型后，在系统控件“串口 0”的动作脚本里调用 **sysCom0.read()** 函数读取有效串口数据。PS-LCD 根据设置好的阀值、帧头和帧尾等属性，自动接收串口数据放入缓冲区并按指定条件触发动作脚本执行完成数据读取。***用户需编写接收处理程序。***

注：CTP 方式和用户自定义方式具有相同的 write()和 writeString()方法。

4.2 CTP 协议

4.2.1 GUI 对象和属性

人机图形界面由数个控件组成，例如文本框，按钮，进度条等。把这些组成界面的控件实例称为 GUI 对象， 如名为“myedit”的文本框。

每个 GUI 对象有唯一的名字，便于识别。另外还有一些特征参数（例如颜色，大小等），通过改变其值，可以改变对象的行为和状态，这些特征参数称为属性。GUI 对象的属性分为：

- 静态属性（Static Property 或 Design-time Property）：在 Designer 设计阶段指定，运行时保持不变（例如按钮的风格和大小）。
- 动态属性 （Dynamic Property 或 Run-time Property）：既可以在 Designer 设计阶段指定，也可以在界面运行时改变其值（通过串口发送命令或者界面中嵌入的 JavaScript 脚本实现）。

4.2.2 定义

Designer 设计生成的界面，虽然可以在没有外部控制器参与的情况下能独立在 PS-LCD 上运行，但在很多应用场合，界面中的 GUI 对象需要与外部控制器交互数据，界面才能与实际应用集成在一起。这种数据交互包括：

- **GUI 对象 -> 外部控制器**

PS-LCD 自动检测控件对象 GUI 事件（例如按钮被点击），通过通讯接口发送事件消息告知外部控制器。注意：**PS-LCD 默认不发送任何事件消息给外部控制器，只有在 Designer 设计界面时，选中了相应控件的“事件通知（Verbose）”属性，并且在脚本编辑框中的触发事件中设置控件动作，才有此功能。**

- **外部控制器 -> GUI 对象**

外部控制器发送指令给 PS-LCD，控制/查询 GUI 对象状态（如文本框内容）。所有当前运行界面 GUI 对象的动态属性都能被外部控制器无条件控制和查询。

CTP 协议就是为了满足上面的通讯要求而产生，外部控制器通过 CTP 协议可简单直观地完成上述两种方向的数据交互，快速集成 Designer 所设计的界面。

定义：CTP 协议是实现外部控制器与 PS-LCD 界面中的 GUI 对象通讯的软件协议，类似操作系统的 Shell，能够互动式地解释和执行用户系统传输的命令。该协议命令集为纯 ASCII 码，只包含三种格式，简单直观，易于理解。外部控制器通过该协议与 PS-LCD 上的若干 GUI 对象通讯，能够直观快速地完成界面与实际应用的数据交互。通讯模型如图 4-1。

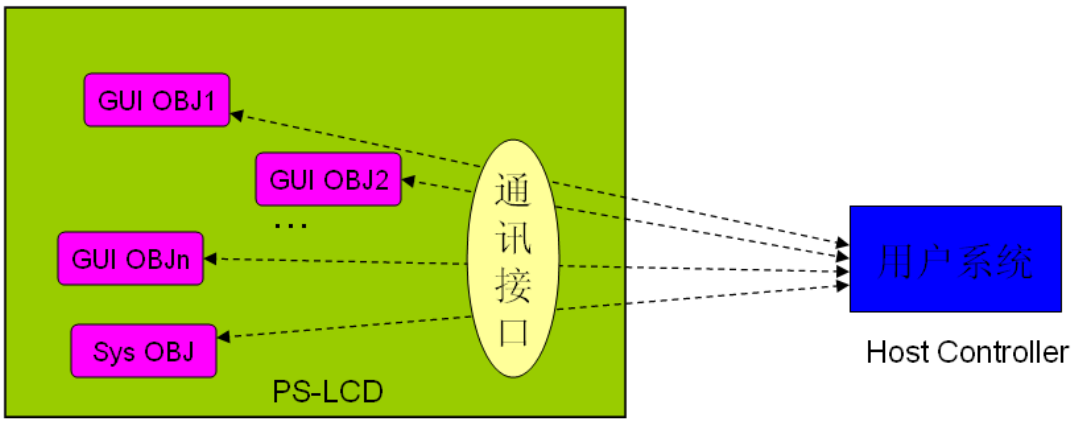


图 4-1 CTP 通讯协议模型

外部控制器通过 CTP 协议可实现：

- ☑ 控制 GUI 对象的动态属性和 PS-LCD 系统工作状态；
- ☑ 查询 GUI 对象的动态属性；
- ☑ 获得 GUI 对象的事件通知；

4.2.3 格式

PS-LCD 的 CTP 通讯协议完全为 ASCII 码（除中文字符外），包括三种类型：

- **控制(Control)**

控制 GUI 对象动态属性，格式：

HC → PS-LCD	[object name].[runtime property name] = xxx<CR>	如： e1.text= “abcd\r”
PS-LCD → HC	C+<CR> C-<CR> C?<CR> C!<CR>	成功 失败 命令格式错误 GUI 对象名字错误

- **查询(Query)**

查询 GUI 对象动态属性，格式：

HC → PS-LCD	[object name].[runtime property name] ? <CR>	如： e1.text ?\r
PS-LCD → HC	Q+[object].[runtime property] = xxx <CR> Q-<CR> Q?<CR> Q!<CR>	成功，如 Q+e1.text=”abcd”\r 失败 命令格式错误 GUI 对象名字错误

- 事件(Event)

GUI 对象的某个动态属性在运行时改变，而且该 GUI 对象的**事件通知（Verbose）**属性在 Designer 设计阶段已选中，满足指定的控件事件触发条件时，外部控制器将会收到 Event 类型的消息，格式如下：

PS-LCD → HC	E+[[object name].[runtime property name] = xxx <CR>	如：HC 收到 E+e1.text=”efgh” 表示 GUI 对象 e1 的内容被用户改为”efgh”，

注：text 为文本框动态属性，e1 为文本框的名字，”abcd”为需要显示的内容；<CR>为 ASCII 码’\r’(十六进制 0x0D)

4.2.4 外部控制器软件设计

对外部控制器来说，与界面相关的软件非常简单——**按三条 CTP 协议指令与 PS-LCD 通讯即可**。通常只需一个简单 Loop 循环，软件流程如图 4-2.从图中可看出，复杂的人机界面软件转变成了简单串行口通讯程序，而通讯协议简单直观，有效降低了开发难度，提高了系统稳定性，可大大缩短产品开发周期。

由于 PS-LCD 通讯协议为纯 ASCII 码格式，在没有外部控制器的情况下，将 PC 串口与 PS-LCD 连接，利用 Windows 的超级终端（程序->附件->通信->超级终端）或者串口调试助手可模拟外部控制器串口，轻松直观地跟踪代码。如果采用超级终端调试，相关设置如下：

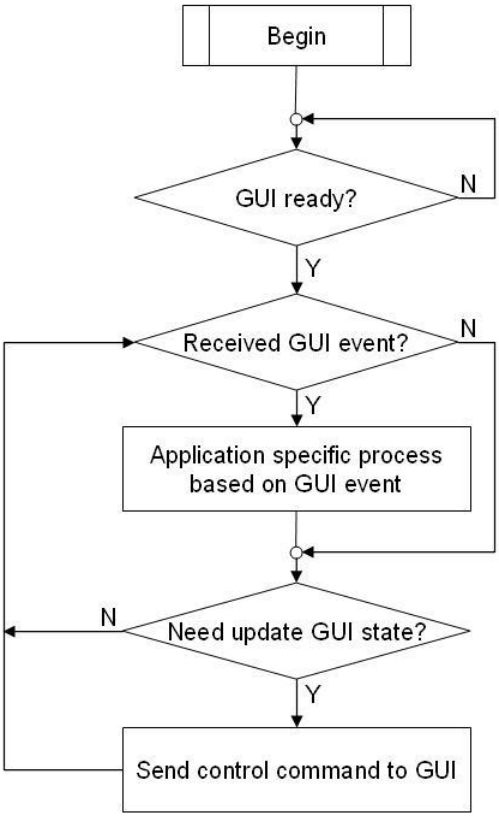
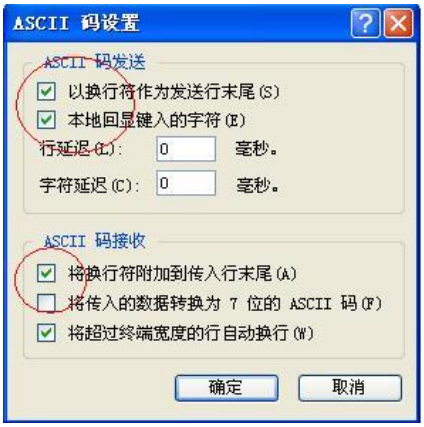


图 4-3 外部控制器软件流程图

1) 设置串口，红圈内的波特率须与 Designer 中设置一致，无硬件流控制.

2) 设置 ASCII 通信方式，红圈部分必须选中。



完成设置后，点“连接”按钮。然后上电启动 PS-LCD，待 PS-LCD 界面启动完毕后,在超级终端上将出现如下消息：

E+sys.state=ready. （表示 PS-LCD 已经就绪，可发送接收 CTP 消息）

如果点击界面上的某个按钮控件（假设按钮名字为 b），超级终端上显示：

E+b.state=1

如果我们输入如下命令(敲完后回车)：

e.text=“This is a test!”

那么你会看到 LCD 上的文本框（假设文本框的名字为 e）显示内容更新为”This is a test!”，同时超级终端上看到如下消息：

C+ （表示控制命令执行成功）

下面是一个外部控制器通过 CTP 通讯协议集成界面的简单例子：

- **要求：**用文本框实时显示电压值，按钮控制系统启动/停止
- **实现：**

硬件	外部控制器 + PS-LCD
软件	**PS-LCD side:** 1) 在 Designer 2) 中建立工程 demo，双击 main.ui，拖放一个文本框到界面编辑区，命名为 volt； 3) 在界面区拖放一个按钮，修改文本内容为“启动”，控件 ID 设为 start，不要勾选“事件通知”属性，点击属性的“动作脚本(action)”，在触发条件中选择“pressed”事件，在脚本编辑区

输入 **sysCom0.write（0x55）;**（**注意：**write()为系统控件 sysCom0 的方法函数，详细用法请参看“软件速查表”。），这样当界面运行时，按下 start 按钮，PS-LCD 会自动向外部控制单元发送一字节十六进制数 0x55。）

4) 点击菜单栏“**工具->生成界面**”，生成 demo.spf 文件；

5) 通过 Flex 下载 demo.spf 到 PS-LCD，并让 PS-LCD 进入界面模式，启动界面。

外部控制器（通常为 51 或者 ARM）软件：

1) 当收到字节值为 0x55，说明界面上按下了启动按钮，此时开始采集操作；

2) 采集电压值，假设此时为 11.22V，通过通讯接口发送字符串“volt.text=11.22”到 PS-LCD。

3) 定时重复步骤 2，可实现界面电压值的实时显示。

调试：*（用超级终端）*

假设界面并没有正常工作，可以打开超级终端（设置按上面的方法），先检查是否收到”E+sys.state=ready”消息来确认通信是否成功。在超级终端中输入 volt.text=”11.22”，查看 LCD 上是否有错误提示框弹出，然后再检查超级终端中的返回值（C+表示成功执行；C-表示执行出错）。总之，你可以快速直观地找到问题所在。

4.3 用户自定义协议

顾名思义，该协议由用户自定义实现，以便让 PS-LCD 能够灵活地与各种外部控制单元实现协议的无缝连接。理论上说，任何有公开详细通讯协议说明的外部控制单元(如：51/ARM/DSP 等外部控制器、PC 电脑、PLC、现场采集控制单元等)，都能与 PS-LCD 连接通讯，实现人机界面扩展。

在 Designer 设计界面时,选择“工具”→“系统控件”选项，弹出系统控件设置界面，选中“串口 0”系统控件，在属性栏中指定通讯协议为“UserDefine”，

同时可以设置通讯速率，门限值，帧头，帧尾等。这样 PS-LCD 即可工作在自定义协议模式下。

自定义协议下，系统控件 sysCom0 只提供接收和发送数据的方法函数接口，发送什么内容的数据，通过在任意控件中调用 sysCom0.write ()或者 sysCom0.writeString ()函数实现；接收到数据如何处理，由用户在 sysCom0 动作脚本中自定义实现。

以下是 **sysCom0** 系统该控件相关的函数接口说明：

write(char, …) —— 调用该函数，可发送任意内容数据，一个参数代表一个字节，每次调用最多支持 25 个字节。如：sysCom0.write（0x55, 0x66）将发送十六进制 0x55 和 0x66 两字节数据；sysCom0.write(xx.value) 将发送 ID 为 xx 的控件当前内容(自动转化为以字节数据)，假设 xx 为文本框 ID，当前内容为 16，那么串口发送一字节数据 0x10。

read(int) —— 该函数用于读取串口接收缓冲区数据。如：sysCom0.read(10)，从接收缓冲区读取 10 个字节数据并转换为整型，未读数据仍保留在接收缓冲区。当 x=0 时，读取接收缓冲区所有数据并清空接收缓冲区。

readableBytes( ) ——查询接收数据缓冲区未读出数据字节数。

clearReadBuffer( ) ——清空接收缓冲区数据。

关于 **sysCom0** 系统该控件详细用法，请参考“系统控件”章节。现举两个例子说明自定义通讯的实现过程。

4.3.1 无帧格式通讯

无帧格式的通讯，外部通讯单元与 PS-LCD 交互的数据信息量很小，一般只需要一个或几个字节即可完全表示完毕，在通讯过程中无需增加额外的控制字节，属于最简单的通讯过程。界面设计和通讯实现如下：

1) 新建工程，进入系统控件设置，选中系统控件“串口 0”，设置通讯协议为 UserDefine，设置接收阀值为 1 字节（假设外部控制单元只发送一字节数据给 PS-LCD 显示）。不要使能帧头和帧尾。

2) 用 Designer 完成界面控件布局；

3) 在需要发送数据控件动作脚本中设置触发条件，脚本中调用 sysCom0.write()实现数据发送；如在按钮的动作脚本中输入：sysCom0.write(0x55); 触发条件选择"pressed"。当该按钮在运行时按下一次，PS-LCD 将向通讯口发送一字节十六进制数据 55；

4) 打开系统控件设置界面，选择系统控件"串口 0",打开动作脚本编辑框，选择触发条件为"received"，输入脚本：

```
test.text = sysCom0.read(1);
```

上面函数将在接收缓冲区超过一字节时，自动被调用。从上面的实现可以看出，当 PS-LCD 接收到数据后，将把该字节读出，并显示在名字 ID 为 test 的控件文本中，假设该控件是一个文本框，那么外部控制单元每向 PS-LCD 发送一个字节，文本框内容自动刷新为该字节的数据显示；如：外部控制单元向 PS-LCD 发送一字节数据 0x10，这时可以看到界面的文本框内容自动刷新显示为 16；发送 0x11， 显示为 17，以此类推实现动态显示。

5) 启动模拟器，模拟通讯的发送接收过程，验证自定义的通讯协议，重复 2 和 3 步骤直到满意为止。

6) 生成 spf 文件，将初步调试好的界面下载到 PS-LCD 中，用实际的外部控制单元与 PS-LCD 通讯，验证最终设计结果。

关于以上界面和脚本的实现详情，请参考开发光盘中"**专题演示\ 4.3-无帧格式自定义通讯演示**"。关于自定义通讯协议函数说明，请参看"**软件速查表**"；

4.3.2 带帧格式通讯

上述步骤的通讯协议帧比较简单，主要为更好地说明自定义通讯协议的设计过程。在实际项目中，建议帧数据包中应有帧头、帧尾和校验字节位。当 PS-LCD 接收到超过帧字节数的数据后，触发执行动作脚本中的用户定义函数，确认帧头、帧尾和校验都无误后，再调用相关脚本刷新界面控件，这样可以最大限度地保证通讯可靠性。建议帧数据格式如下：

帧头字节	数据字节	校验字节	帧尾字节

假如设计一个界面实时显示电压(范围为0-220V)和电流值(范围为0-100A)，帧格式可按如下定义（类似工业领域中的 Modbus ASCII 通讯协议），共 9 个字节；

帧头	电压 高数据位	电压 低数据位	电流 高数据位	电流 低数据位	校验	帧尾

帧头字节：ASCII 码中的冒号（十六进制为 0x3A），1 字节

电压/电流 高数据位：对应值的十六进制数的高位 ASCII 码，1 字节

电压/电流 低数据位：对应值的十六进制数的低位 ASCII 码，1 字节

校验字节：所有有效数据字节（蓝色部分）相加和，共 2 字节

帧尾字节：ASCII 码回车(十六进制为 0x0D)和换行（十六进制为 0x0A）,共 2 字节

例如，一帧完整的字节序列如下：

0x3A 0x36 0x34 0x43 0x38 0x32 0x43 0x0D 0x0A

0x3A 是帧头 ASCII 码 冒号

0x36 0x34 代表电压值为十进制 100（100 的十六进制表示为 0x64，数字 6 的 ASCII 码为 0x36，数字 4 的 ASCII 码为 0x34）

0x43 0x38 代表电流值为十进制 200（200 的十六进制表示为 0xC8，字母 C 的 ASCII 码为 0x43，数字 8 的 ASCII 码为 0x38）

0x32 0x43 代表校验和十六进制 0x2C（100 + 200 = 300， 300 的十六进制表示为 0x12C，取低 8 位，值为 0x2C——数字 2 的 ASCII 码值 0x32，字母 C 的 ASCII 码值 0x43）

0x0D 0x0A 为帧尾标志，分别代表回车和换行。

如果外部控制单元按以上帧格式给 PS-LCD 发送数据，如何让界面实现正确解析和界面更新？设计步骤如下：

1) 用 Designer“所见即所得”，0 代码完成界面控件布局，拖如两个文本框，一个 ID 为 volt 用于显示电压，另外一个 ID 为 cur 用于显示电流；

2) 打开系统控件设置界面，选择串口 0：

在右侧属性设置里，选择帧头使能，设置帧头为 0x3A，阀值为 8，帧长

为 9，需减掉一个帧头字节，所以为 8，如下图所示。

点击动作脚本，选择触发条件为 **received**，用脚本编写实现接收函数：

```
var i, voltage, current, checksum;
var data = new Array(8);

if (sysCom0.readableBytes() >= 8)
{
    /* If detecting header tag, load frame bytes */
    data = sysCom0.read(8);

    /*convert data from ascii to int*/
    voltage = ascii2int(data[0]) * 16 + ascii2int(data[1]);
    current = ascii2int(data[2]) * 16 + ascii2int(data[3]);
    checksum = ascii2int(data[4]) * 16 + ascii2int(data[5]);
    /*verify checksum and tail tag*/
    if (checksum == Number((voltage + current) & 0xFF))
    {
        if (data[6] == 0x0d && data[7] == 0x0a)
        {
            /*refresh widgets now*/
            volt.text = voltage;
            cur.text = current;
            plot.value = voltage + "," + current;
        }
    }
}
```

上面函数将在 PS-LCD 自动检测到帧头 0x3A，而且接收缓冲区数据大于或者等于 8 字节时，执行动作脚本。该脚本首先是从缓冲区中读取 8 字节帧数据，

然后检查校验位和帧尾是否正确，如果没问题，刷新电压和电流的界面显示控件。外部控制器不断往 PS-LCD 发送固定格式的帧，电压和电流就实现了动态显示。

3) 运行模拟器检查是否运行正常，重复以上步骤直到满意为止；

4) 生成界面的 spf 文件，并下载到 PS-LCD；

5) 外部控制单元采集电压和电流值，通过上面定义的帧格式发送数据到 PS-LCD：

如果采集电压为 100V，电流为 200V， 发送如下数据：

0x3A 0x36 0x34 0x43 0x38 0x32 0x43 0x0D 0x0A (此时会看到 PS-LCD 上电压显示为 100， 电流显示为 200)

如果采集电压为 101V，电流为 201V， 发送如下数据：

0x3A 0x36 0x35 0x43 0x39 0x32 0x45 0x0D 0x0A (此时会看到 PS-LCD 上电压显示为 101， 电流显示为 201)

6) 需要实现 PS-LCD 按相同帧格式返回数据给外部控制器，调用 write()函数即可，如：

```
/* send frame data now,  assume that: format is same as received frame*/
sysCom0.write(0x3a, cmd1_h, cmd1_l, cmd2_h, cmd2_l, cs_h, cs_l, 0x0D, 0x0A);
```

其中 cmd1_h, cmd1_l, cmd2_h, cmd2_l, cs_h, cs_l 为脚本中用的临时变量。

关于以上的界面和脚本的实现详情，请参考开发光盘中“**专题演示\4.3-带帧格式自定义通讯演示**”。关于自定义通讯协议函数说明，请参看“**软件速查表**”；

Greatal

第五章 设计实例

本章以一个温控系统为例，详细说明基于 PS-LCD 的人机界面设计全过程(以 CTP 通讯协议为例)。

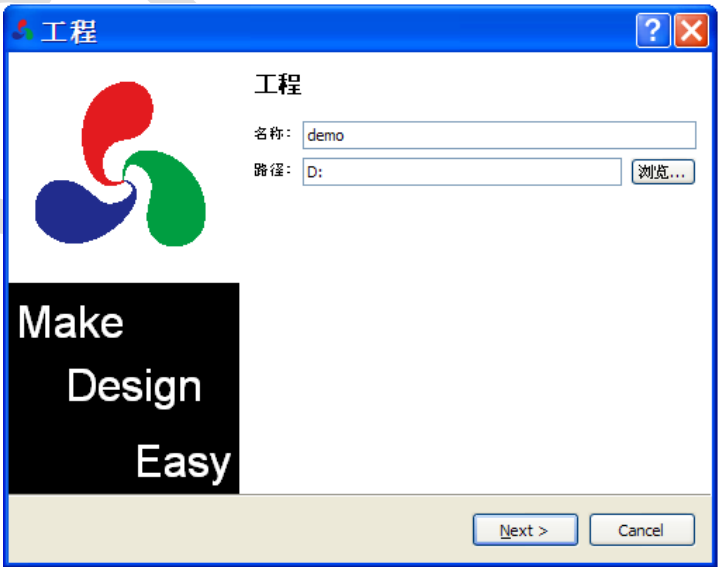
5.1 界面要求

- ☑ 用两个按钮来控制室内温度，按“+“， 温度上升，按”-”，温度下降，同时外部控制器控制相关电路调节对应温度;
- ☑ 温度需要数字显示出来;
- ☑ 当温度超过 30 度，显示“过热”动画；小于 5 度，显示“过冷”动画；介于 5-30 度之间，显示“正常”动画.

5.2 实现步骤

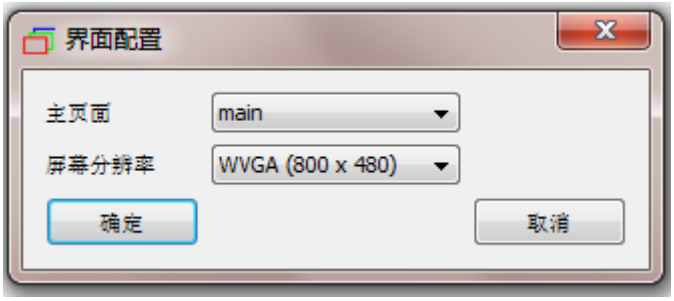
5.2.1 编辑界面

1) 打开可视化界面编辑工具 Designer，点击菜单栏中的文件->新建工程，输入新工程的名字，选择保存路径。我们这里新工程命名为“demo”，路径为 D 盘， 然后点“next”，然后点“finish”。

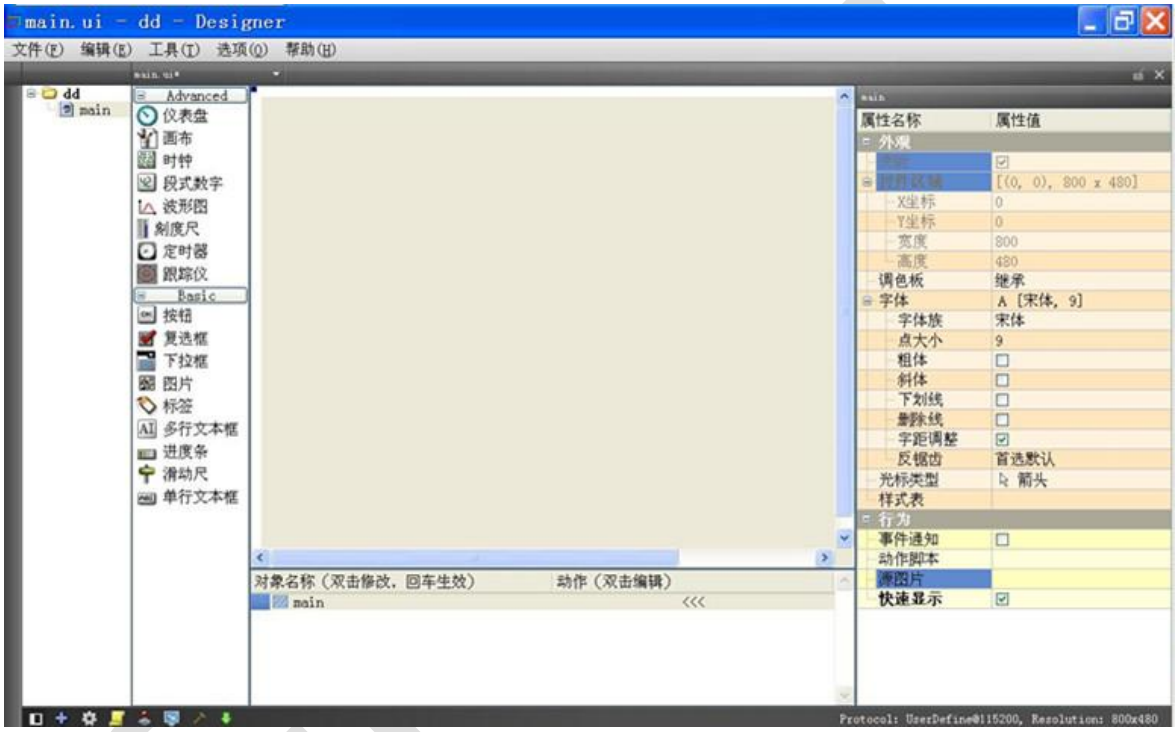


2) 当出现如下画面，采用默认主页面为 main，PS-LCD 分辨率为 WVGA(800 x 480)，

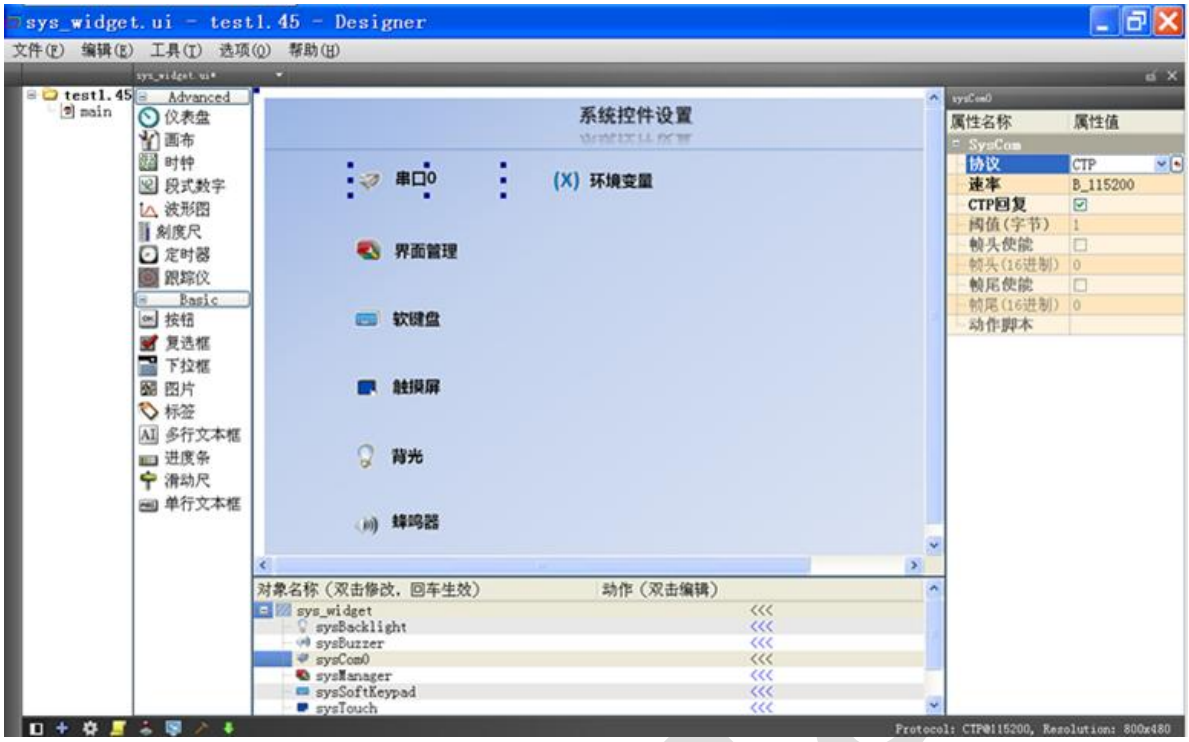
点确定。 工程建立完毕，可以开始编辑温控页面了。



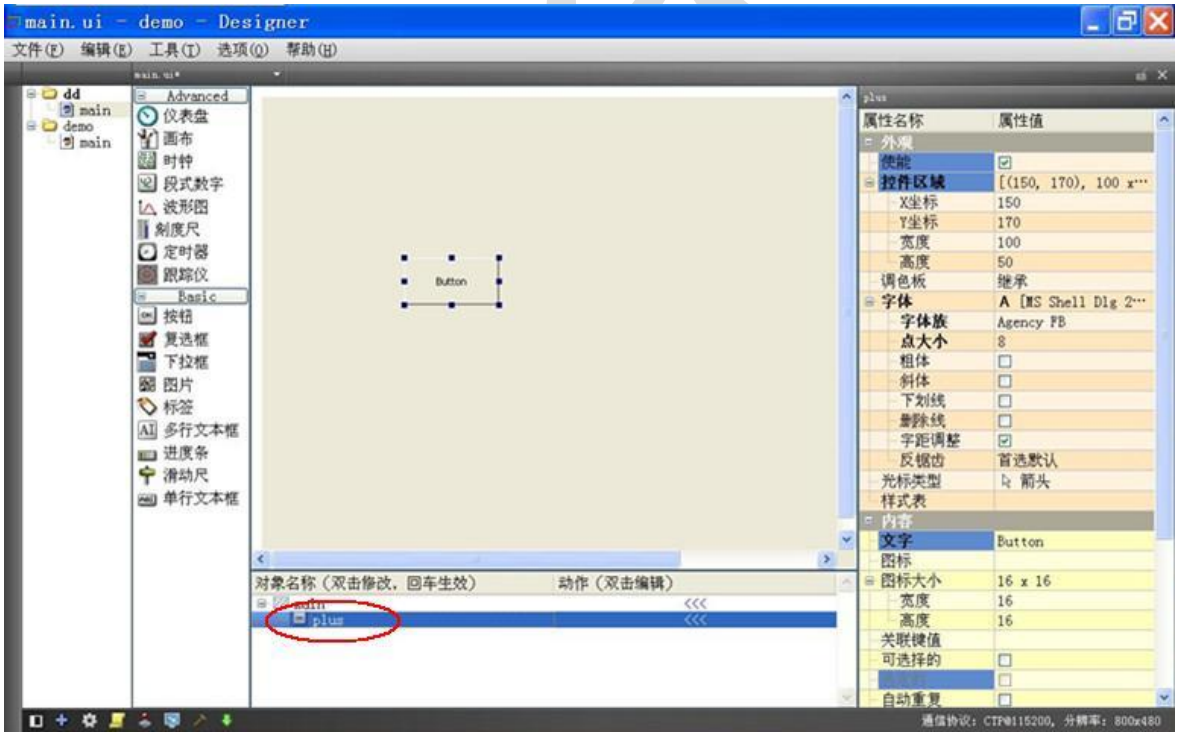
3) 当出现如下画面时，即可开始编辑 main 页面。



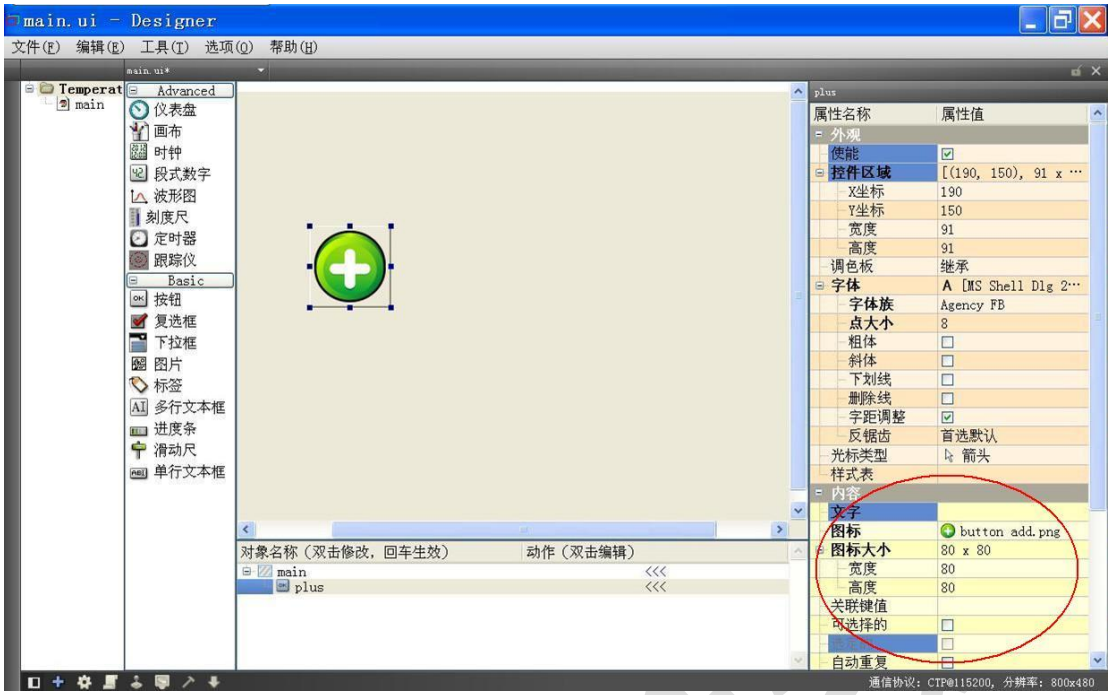
4) 首先选择工具-系统控件选项，出现如下画面，在右侧属性设置里选择协议 CTP，速率 115200。双击 main，返回设计页面。



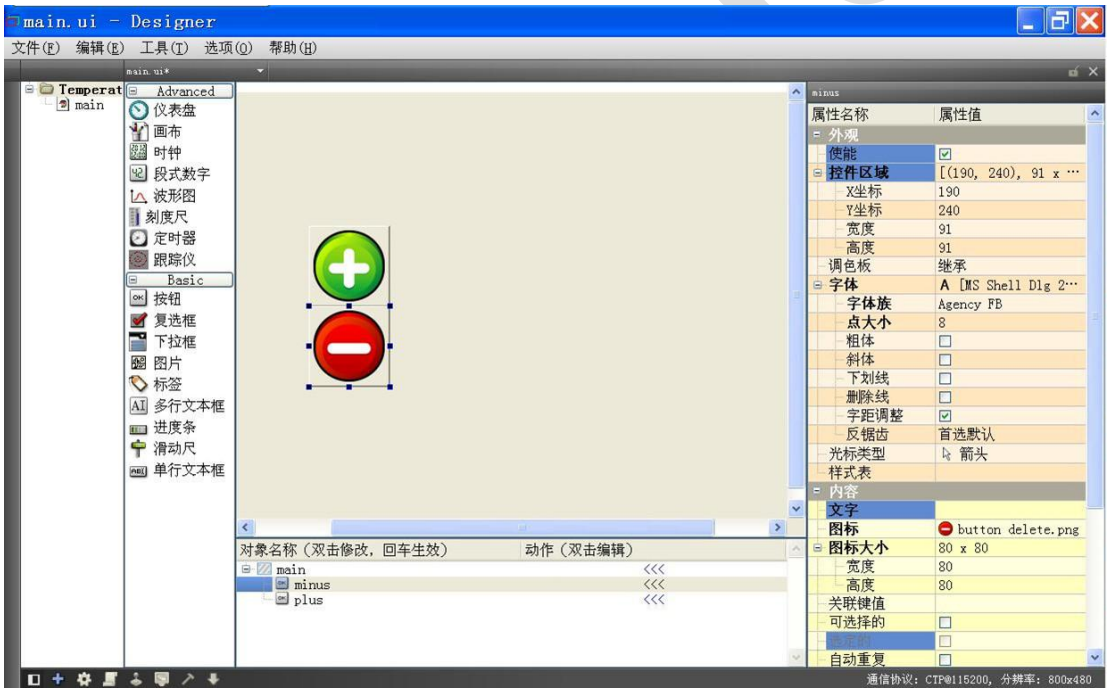
5) 用鼠标将按钮控件从控件区拖入页面编辑区，选中该按钮，然后在对象管理区中重命名为 plus。



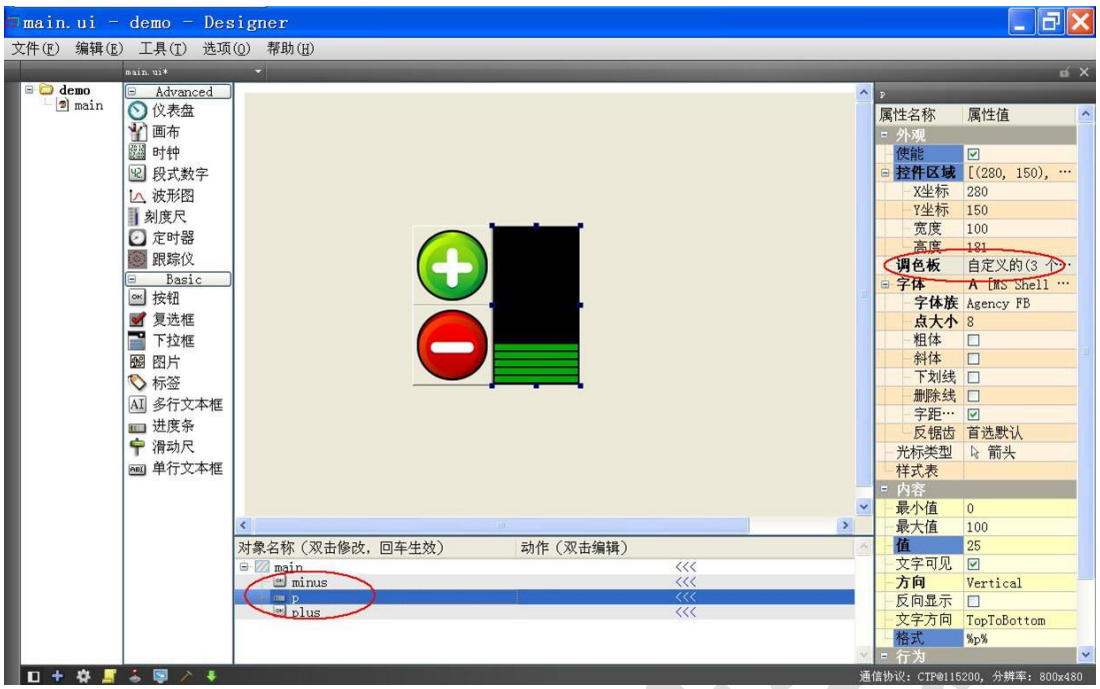
6) 仍然选中 plus 按钮，在属性区中点击图标(icon)属性上的按钮选择图标，(在此我们选择了一个加号的 png 格式图片)，并且在图标尺寸(iconSize)属性中将图标分辨率调整为 80x80，最后调整页面编辑区中的按钮大小与图标合适。



7) 按步骤 5 和 6 的方法，添加一个 minus 按钮,如下图。

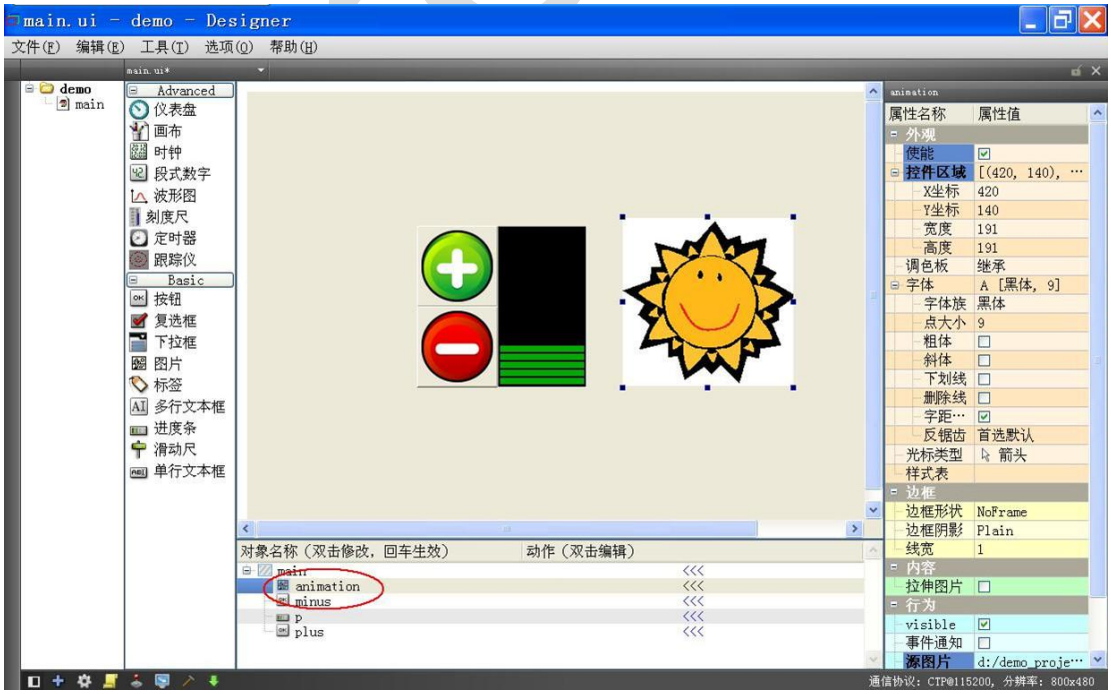


8) 从控件区拖一个进度条控件到页面编辑区，命名为 p,调整大小与按钮对齐。在属性区通过调色板（pallete）属性改变显示风格为黑底/绿进度指示，通过



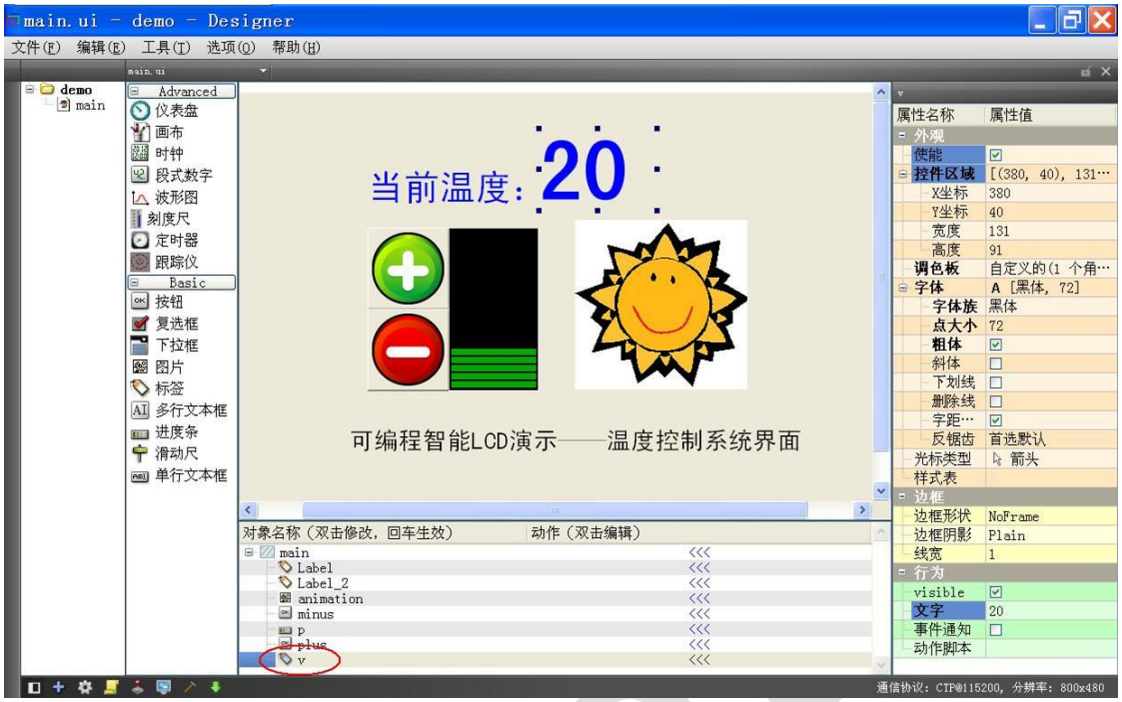
minimum/maximum/value 属性分别设置进度范围最小 0，最大 50，默认进度值为 20，通过方向(orientation)属性改变进度条方向为垂直 vertical。

9) 从控件区拖入图片控件，命名为 animation，选择“源图片(source) ”属性中的图片为“sunlog.gif”动画文件作为默认动画。然后将“cold.gif”和“hot.gif”拷贝到 demo 工程工作目录下的 picture 目录中，以便界面运行时改变动画效果。



10) 从控件区分别拖入三个标签控件，分别输入文字：PS-LCD 演示——温度控

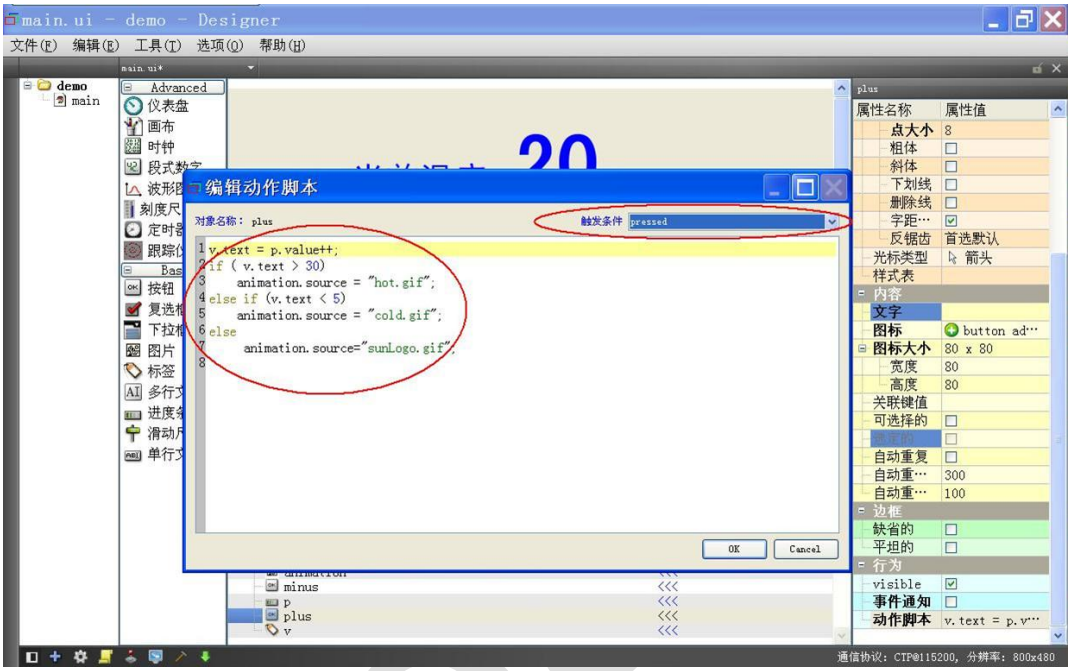
制系统页面、当前温度、20。再分别通过字体(font)属性调节字体类型，大小，颜色、位置等，最后将文本内容为 20 的标签命名为 v。



11) 选中进度条对象 p，然后选中属性区中的“事件通知（Verbose）”属性，以便进度条 p 的值发生变化时，PS-LCD 自动发送消息通知外部控制器。

12) 选中按钮 plus，点击属性区中的“动作脚本(action) ”属性按钮，进入脚本编辑界面，在“触发条件”中选择“pressed”，在输入框中输入如下 JavaScript 脚本程序：

```
v.text = p.value++;
if ( v.text > 30)
      animation.source = "hot.gif";
 else if (v.text < 5)
      animation.source = "cold.gif";
else
      animation.source="sunLogo.gif";
```

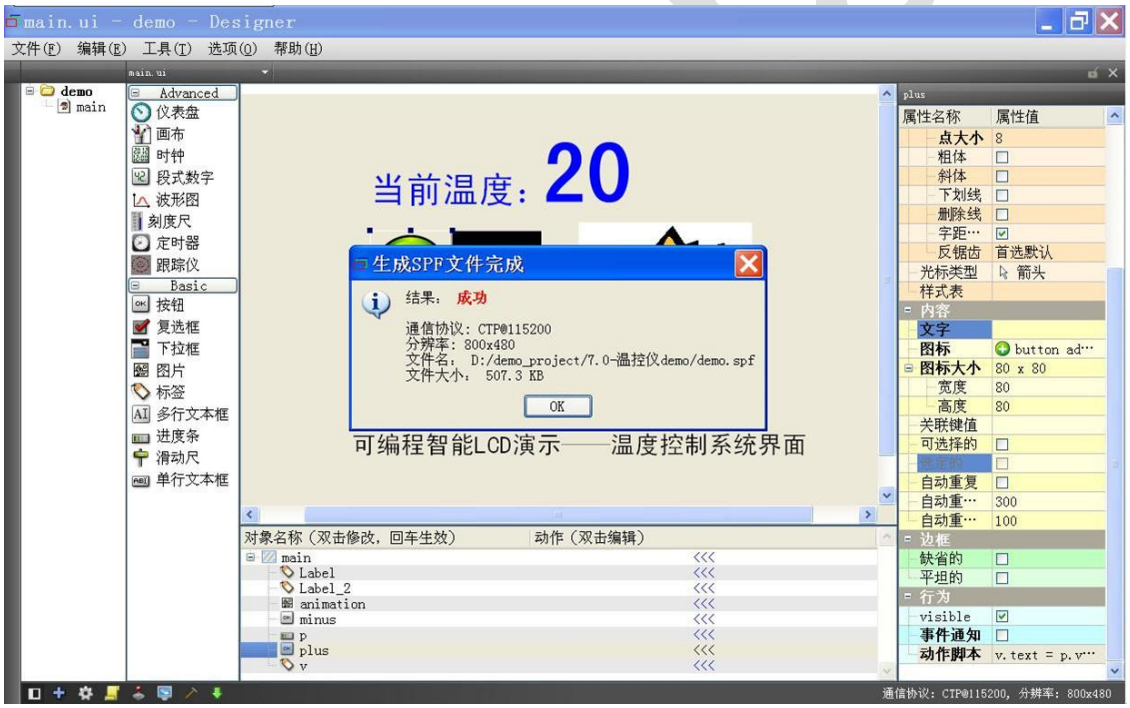


13) 同步骤 11，在按钮 minus 的“动作脚本(action) ”属性输入框中输入：

```
v.text = p.value--;
if ( v.text > 30)
      animation.source = "hot.gif";
 else if (v.text < 5)
      animation.source = "cold.gif";
else
      animation.source="sunLogo.gif";
```

14) 点击菜单栏“工具->模拟器”，启动 PS-LCD 软件模拟器，验证界面效果和动作是否与设计一致。

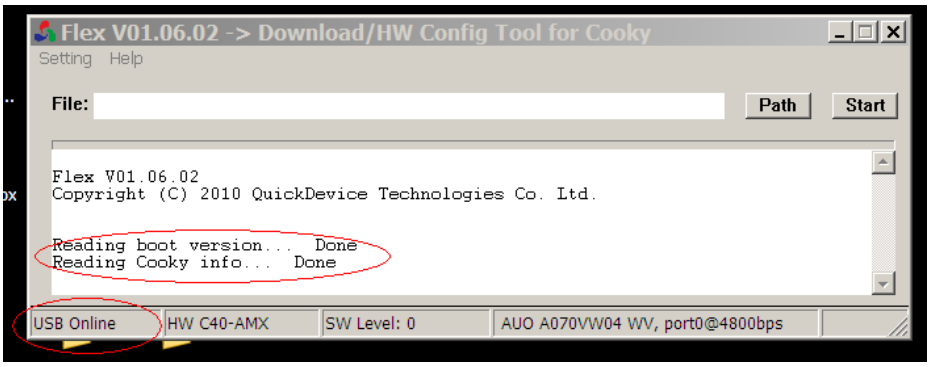
15) 上步确认无误后，点击菜单栏“工具->生成界面”选项生成 demo.spf 文件。



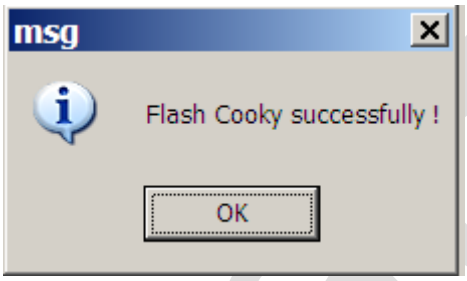
5.2.2 下载 SPF 文件

1) 用 USB 下载线连接 PS-LCD 与 PC，进入下载模式；

2) 启动 Flex 下载工具，这是会看到：USB online；



3) 点击 path 按钮，选择刚才生成的 demo.spf，点击 Start 开始下载，完成后提示下载成功：



4) 拔掉 USB 下载线，按复位按钮或者重新给 PS-LCD 上电进入界面模式，待启动完毕，上面设计的 demo 工程界面即出现在 LCD 上。

注意：FLEX 仅支持在 Windows XP 下使用，如果是采用 Win7 及以上版本 Windows 开发产品，请采用 U 盘下载方式更新界面。

5.2.3 外部控制器界面软件

外部控制器软件由三个文件组成：

- app.c (温控系统的 GUI 主程序)
- ctp.c (Greatal 提供的 CTP 协议源文件)
- hw.c (用户的串口驱动程序源文件)

ctp.c 由 Greatal 提供(在光盘中 **Cooky_Release\Designer_Demo\7.0-温控仪 demo\mcu-ctp-src.zip**)，*用户不需任何修改，直接与用户的应用程序编译连接即可*，这里不做介绍；hw.c 是跟特定的外部控制器 HC 硬件平台相关的串口驱动，只要提供三个 API 接口（hw_init()，hw_write()和 hw_read()）供 ctp.c 调用即可，用户自行完成。下面重点介绍 app.c 的程序，源代码见下一页。

```
/* PS-LCD based GUI demo codes
 * Copyright (C) 2011 Beijing Greatal Technologies Co. Ltd. */
#include "ctp.h"
#include <string.h>
void main()
{
    char name[16], prop[16], value[64];
    int degree = 25;

    /* Initialize ctp port */
    ctp_init();
    /* Infinite event loop*/
    while(1) {
        if (ctp_event(name, prop, value)) {
            /* no invalid event found, do something you'd like */
            continue;
        }
        if (!strcmp(name, "p") && !strcmp(prop, "value")) {
            /* if temperature setting get changed */
            degree = atoi(value);
            /* Call hardware control routine to set temperature */
            …..
        }
}
```

当设定温度变化时处理程序

代码分析如下：

- app.c 由一个无限循环的 event loop 组成，当调用 ctp_event（）函数时，程序通过外部控制器串口读取 GUI 的事件。如果没有 GUI 事件产生，该调用会立即返回（如果 hw.c 中的 hw_read()在没有有效串口数据时立即返回，适用于没有 OS 的系统）或者阻塞（如果 hw.c 中的 hw_read()在没有有效串口数据时阻塞， 适用于有 OS 的系统）。
- 如果 ctp_event（）的返回值非 0，说明有 GUI 事件产生，事件的内容存储在 name, prop 和 value（均为字符串），通过比较确定 GUI 的事件类型，然后调用相应的处理程序。
- 关于过热或者过冷的动画效果，在用 Designer 编辑界面时，直接在“动作脚本(action) ”属性中输入 JavaScript 脚本描述即可，无需外部控制器参与。
- 本例中用到了几个标准 C 语言 string 库里面的函数，它们是：strcmp

（），strcat()和 itoa()，具体用法请参考相关资料。

注意：app.c 源码中的蓝色字体为在 Designer 设计阶段定义的控件名字，红色字体为 ctp.c 提供的 API 函数，绿色字体为控件的动态属性名称（详情见“软件速查表”文档）。

5.2.4 实际运行结果

为测试效果，本例在如下软硬件环境编译运行：

- 硬件：7 寸 PS-LCD（LCD：AUO 的 7 寸 WVGA 800 x 480 TFT LCD）
- 外部控制器三线串口：采用 PC 机 RS232 串口模拟
- 编译环境：VC 下编译
- 调试环境：Windows 系统自带的“超级终端“

温控系统运行后，在实际 PS-LCD 上的界面效果如下：

图 5-1 温度正常时界面

图 5-2 按“+”键将温度调整到 31 度时界面

图 5-3 按“-”键将温度调整到 4 度时界面

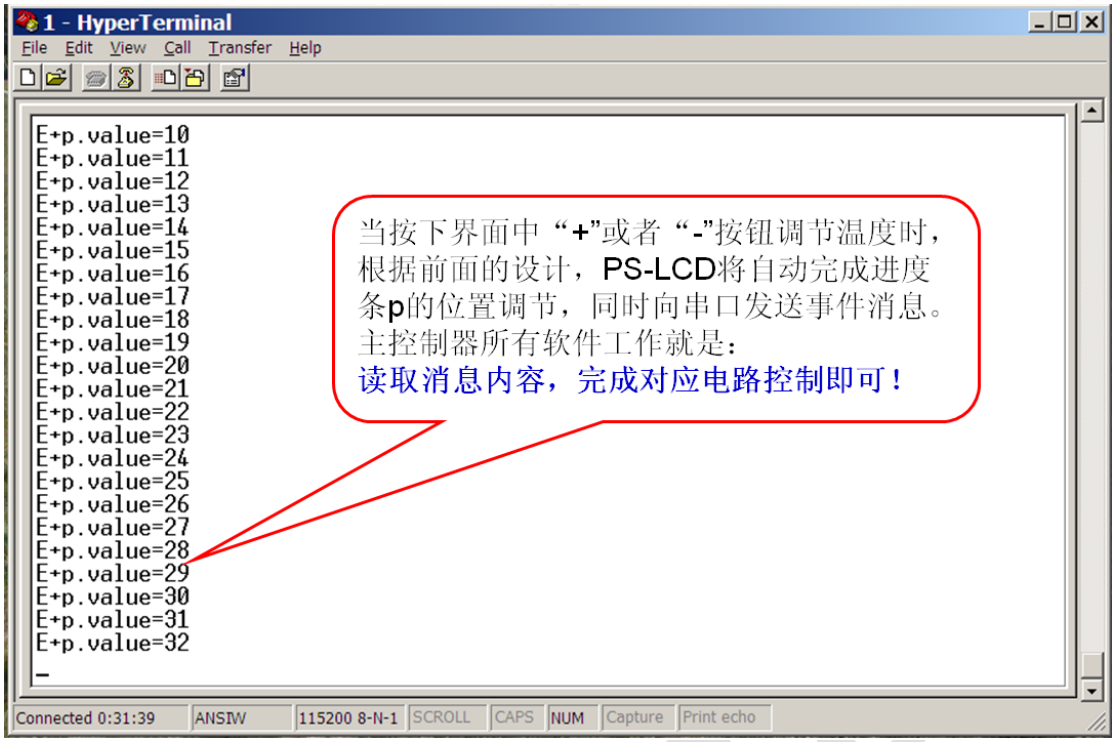


图 5-4 超级终端监控串口数据

5.3 总结

从上例可看出，为嵌入式系统增加人机界面，软件开发可以很简单：

- ☑ 调试一个串口驱动程序；
- ☑ 利用三个简单的 API（ctp_event（）， ctp_control（）， ctp_query（））根据实际需求写少量控制代码。

基于 CTP 通讯协议的人机界面程序为纯 C 代码，不依赖于任何库（除 C 语言的 string 库外），也不依赖于任何软件平台（有无操作系统均可，不需要任何 GUI 的函数库），可轻松嵌入到用户外部控制器 C 代码中。

总之，**任何外部控制单元（51/ARM/DSP，PLC 等），无论性能高低，只需一个串口（或者 485、CAN、Wifi 等），都能轻松快速与 PS-LCD 连接，用极少系统资源实现强大人机界面**。

注意：以上例子主要是为了说明 PS-LCD 的开发过程方便，打开了事件通知属性来演示。在实际项目中，由于外部控制器解析 PS-LCD 发送的事件通知字符串比较麻烦，不建议打开该属性，而是采用自定义发送函数来实现任意格式数据的发送。为准确了解开发和通讯过程，请参考光盘中的各演示工程。

第六章 系统控件

通过系统控件可实现 PS-LCD 的系统功能控制（如串口通讯、背光控制、蜂鸣器驱动、参数存储、功耗模式切换等人机界面系统常见功能）。为最大限度降低软件设计工作量，用户仅需在系统控件界面对相应参数属性进行设置，编写少量代码，即可快速实现这些功能。

图 6-1 系统控件设置界面

目前支持的系统控件有：

6.1 串口 0（sysCom0）

- 功能介绍：

  设置串口通讯协议类型，通讯速率，以及自定义协议的相关参数，可在动作脚本中读取串口数据和处理。**注意**：串口的通讯格式固定为：8N1，即 8 位数据，无奇偶校验位，1 位停止位，不可更改。

- 属性(Property)

  ctpReply：设置串口在 CTP 协议下是否返回应答标志(C+为发送正确，C-为发送错误)；

- 方法(Method)

  write(char …)：写串口发送缓冲区，发送数据；

writeString(string)：写串口发送缓冲区，发送字符串；

read(int i)：读取串口接收缓冲区，i 为读取的字符数，当 i=0 时，读取所有接收缓冲区中未读数据；

readableBytes()：读取接收缓冲区未读字节数；

clearReadBuff()：清空接收缓冲区；

- 事件(trigger Event)

received：当串口收到数据时产生；

sent：当串口发送数据时产生；

- 例子

sysCom0.ctpReply=1;

返回应答信号使能（只对 CTP 协议有效）

sysCom0.write(0x55);

写串口发送缓冲区，串口向外部发送一字节数据 0x55

sysCom0.write(0x55, 16, 'A');

写串口发送缓冲区，串口向外部发送三字节数据 0x55, 0x10, 0x41

sysCom0.writeString("12AB");

写串口发送缓冲区，串口向外部发送字符串"12AB"，即 4 字节数据 0x31, 0x32, 0x41, 0x4B

var a = new Array(2);

a = sysCom0.read(2);

读取串口接收缓冲区，返回 2 字节数据存储在数组 a 中，未读出的数据仍保留在接收缓冲区

var a = new Array(sysCom0.readableBytes());

a = sysCom0.read(0);

读取串口接收缓冲区所有数据，返回数据按字节存储在数组 a 中，读取完成后，接收缓冲区为空

var size = sysCom0.readableBytes();

读取接收缓冲区未读出数据字节数，每调用一次 read()，未读出数据字节数自动减少；每收到一个字节串口数据，未读出字节数自动加一

sysCom0.clearReadBuffer();

清空接收缓冲区数据

6.2 界面管理(sysManager)

- 功能介绍：

设置 PS-LCD 背景等参数。如果”**调试模式**”属性被选中，在界面运行时，若出现脚本错误，会弹出错误对话框，方便调试程序。如选中“**页面切换动画**”属性，在切换”**快速显示**”未选中页面时，显示加载页面进度动画。

- 属性(Property)

  debugMode：设置是否为调试模式；

  cursorVisibility：设置光标是否可见；

  formAnimation：设置界面切换时是否显示页面加载进度动画；

- 方法(Method)

  messageBox(string)：弹出消息框；

  execute(string)：执行特定系统命令；

- 例子

  sysManager.messageBox("Hello world");

  弹出消息框,内容为：Hello world

  sysManager.debugMode=1;

  打开调试功能，脚本出错立即提示

  sysManager.cursorVisibility=0;

  设置光标不可见

  sysManager.formAnimation=1;

  设在加载新界面时显示加载进度动画

  sysManager.execute("reboot");

  重新启动 PS-LCD

6.3 矩阵键盘(sysHardKeypad)

- 功能介绍：

  该系统控件负责监控外接的矩阵键盘事件，通过该控件可方便获取矩阵键盘的按键编码和事件（按下或者抬起），方便用户通过外接键盘控制界面行为。PS-LCD 支持最大 4x4 外接键盘，键盘位置与键值的对应关系为（C0-C3, R0-R3 指 Greatal 通讯板上的丝印标识管脚）：

	C0	C1	C2	C3
R0	0	4	8	12
R1	1	5	9	13
R2	2	6	10	14
R3	3	7	11	15

- 方法(Method)

code()：获取触发当前矩阵键盘事件的按键键值；

- 例子

```
If (sysHardKeypad.code() == 2 &&
   sysHardKeypad.currentTrigger() == “pressed”)
 …
```

当外接矩阵键盘的 C0-R2 处键按下时，执行相关动作，一般用于矩阵键盘的动作脚本中。

6.4 软键盘(sysSoftKeypad)

- 功能介绍：

软键盘是在系统软件模拟的键盘控件，方便用户通过触摸屏输入数据。软键盘的默认输入法可配置（letter、number、pinyin 三种可选）。界面中弹出的软键盘外观如下图：

图 6-2 字母/拼音输入软键盘外观

图 6-3 数字输入软键盘外观

在界面运行时，当点击文本框（多行或者单行文本框），软键盘自动弹出；输入完毕后，按 CANCEL 或者 OK 键，软键盘自动消失，同时文本框产生 editingCanceled 或者 editingFinished 事件。

- 属性(Property)

  defaultType：设置软键盘的默认输入法；

- 方法(Method)

  show()：显示软键盘；

  hide()：隐藏软键盘；

- 例子

  sysSoftKeypad.defaultType="letter";

  默认输入法为英文字母 （letter, number 和 pinyin 三种可选）

  sysSoftKeypad.show();

  打开软键盘输入

  sysSoftKeypad.hide();

  关闭软键盘输入 。

6.5 触摸屏(sysTouch)

- 功能介绍：

  设置空闲超时时间，可以设置在多长时间内没有触摸事件，屏幕自动进入待机模式或者再次触屏后唤醒进入正常模式，可在待机和唤醒情况下运行动作脚本。本触摸屏自带校准程序，如需要重新校准触摸，按住非控件区域 5 秒以上自动进入校准画面。

- 属性(Property)

  idleTimeout：触摸屏空闲超时时间；

  powerState：触摸屏的功耗模式，（支持两种模式：待机 suspend 和激活 active 模式）；

- 方法(Method)

  calibrate()：使触摸屏进入校屏模式；

- 事件(trigger Event)

  idleTimeout：当触摸屏空闲时间到时产生；

  wakenUp：当触摸屏再次激活时产生；

- 例子

  sysTouch.idleTimeout = 1000;

  设置触摸屏空闲超时为 1000ms, 1000ms 内没有触摸事件，屏幕自动进入待机模式

  sysTouch.powerState="suspend";

  设置触摸屏进入待机模式（支持两种模式：待机 suspend 和激活 active 模式）

  var l = sysTouch.powerState;

读取触摸屏的操作模式，返回字符串为"suspend"或者"active"

sysTouch.calibrate();

进入触摸屏校准画面。

6.6 背光(sysBacklight)

- 功能介绍：

  设置背光亮度。

- 属性(Property)

  brightness：设置背光亮度；

- 例子

  sysBacklight.brightness = 5;

  设置背光亮度为 5 （亮度范围为 0——10）

6.7 蜂鸣器(sysBuzzer)

- 功能介绍：

  用于控制蜂鸣器的发声，设置蜂鸣器发声时间和发声间隔。

- 属性(Property)

  interval：设置蜂鸣器的发声间隔时间；

  duration：设置蜂鸣器的发声持续时间

- 方法(Method)

  play(int i)：蜂鸣器开始发声，i 为发声次数，当 i=0 时，蜂鸣器持续发声，直到调用 stop 方式停止；

  stop()：停止蜂鸣器发声；

- 例子

  sysBuzzer.play(2);

  蜂鸣器响两声，每次时长和间隔由属性 duration 和 interval 设定

  sysBuzzer.play(0);

  蜂鸣器保持响，直到调用 sysBuzzer.stop()时停止，每次时长和间隔由属性 duration 和 interval 设定；

  sysBuzzer.stop();

  立即停止蜂鸣器发声

  sysBuzzer.interval = 1000;

  设定蜂鸣器发声时间隔 1000ms

  sysBuzzer.duration = 2000;

  设定蜂鸣器发声时每次时长 2000ms。

6.8 环境变量(sysVariable)

- 功能介绍：

  用于存储用户数据等，环境变量存储在 PS-LCD 内部 flash 中，掉电后内容不会丢失。对于 PS-LCD，最大可存储容量为 3M 字节。

- 方法(Method)

  write(string, string)：环境变量赋值；

  read(string)：读取环境变量；

  remove(string)：删除环境变量；

  save()：强制保存环境变量；

- 例子

  sysVariable.write("test", "123");

  把字符串 123 存到环境变量 test 中

  Var i = sysVariable.read("test");

  读取环境变量 test 的值到变量 i 中

  sysVariable.remove();

  清除所有环境变量

  sysVariable.remove("test");

  清除环境变量 test，其他变量保留

  sysVariable.save();

  强制保存所有环境变量。

注意：***除串口 0 系统控件外，其他系统控件功能在模拟器上无法模拟，请下载到 PS-LCD 硬件上验证。***

Greatal

第七章 用户控件

7.1 页面

- 功能介绍：

  页面是装载其他子用户控件的容器，负责管理子控件显示状态，分为普通页面和置顶页面(选中“置顶显示”属性)。通过在脚本中调 show()方法函数来实现页面切换。页面切换时，普通页面将自动消失或者转入后台运行，而置顶页面始终位于显示顶层，一般用来设计导航栏或者状态栏。

- 属性(Property)

  Source：指定背景图片名称，可支持 bmp, jpg, png, tiff, gif 等格式文件；

- 方法(Method)

  Show(string)：切换界面。对于选中快速显示页面(选中“快速显示”属性)，调用该方法会把页面从后台放到前台显示；否则，从 flash 中读取页面内容并显示。参数为字符串或者为空，用于指定切换效果，目前支持“**fade-in**”参数，表示渐变透明度切换到新页面。

- 事件(trigger Event)

  Loaded: 加载——页面从 flash 中读入并加载时产生；

  Raised: 提升——页面从后台切换到前台时产生，对于”快速显示”页面，系统启动时，首先产生一次 loaded 事件，以后再显示该页面，均产生 raised 事件。

- 例子

  xxx.source = “test.jpg”

  设定页面 ID 为 xxx 的背景图片为 test.jpg 文件图片(该文件必须位于 spf/picture/ 目录下)

  xxx.show();

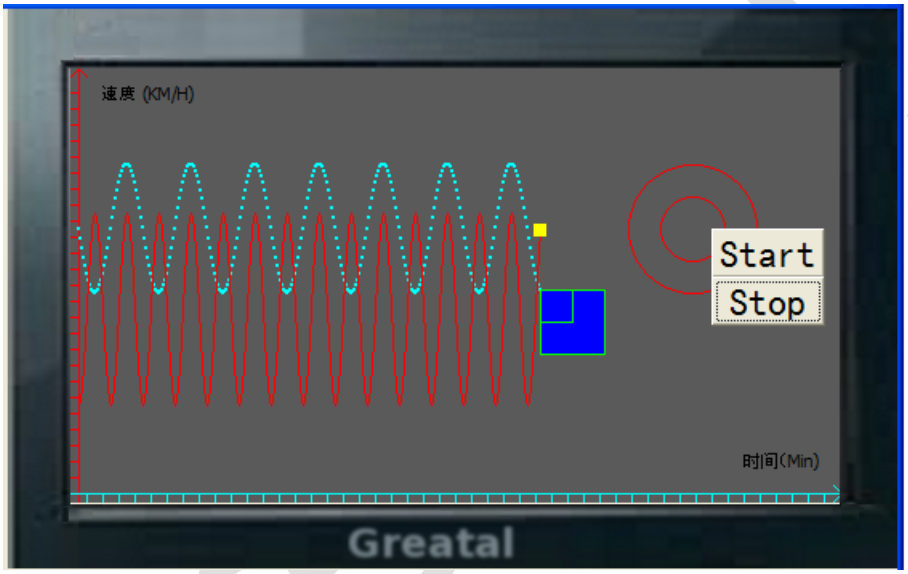
  切换至 ID 为 xxx 的页面

  xxx.show(“fade-in”);

采用渐变透明度效果切换至 ID 为 xxx 的页面(必须选中 xxx 页面的快速显示属性，才会出现切换效果)

7.2 画布

画布(Canvas)控件可以用来绘制基本的图形元素，如点、线、椭圆、多段线、矩形等。一般的使用方法是：所有的图形都可以在画布控件的“动作脚本”属性中先定义好，在画布控件创建时建立。界面运行过程中，通过脚本来更新已有的图形对象，从而实现图形变化。



7.2.1 建立图形

- 创建对象

所有的图形都是基于对象来操作的，不管是画点、线、圆、矩形或者其它图形，首先需要创建一个图形对象。如:

```
cc.createObject("mypoint", "point", 100,  100);
```

在画布坐标 (100, 100) 处创建一个点对象，其中"mypoint"是所创建的图形对象名称，用户可以自己定义；参数"point"表示当前创建的图形类型为点，必须严格按照下表中图形类型名称来定义；(100, 100) 是 X，Y 坐标值。(注意：假设 cc 是画布控件的对象名)

以下为支持的图形类型。 注意：方法名 createObject 区分大小写)

名称	图形	创建方法	参数说明
point	点	cc.createObject("mypoint", "point", 100, 100);	"mypoint"-对象名称，"point"-图形名称，(100，100)，点的坐标
line	线	cc.createObject("myline", "line", 20, 20, 80, 80);	"myline"-对象名称，"line"-图形名称，(20，20，80，80)，线的 x1, y1, x2, y2 坐标。
polyline	多段线	cc.createObject("mypoly", "polyline", 10, 10);	"mypoly"-对象名称，"polyline"-图形名称，(10，10)，点的坐标
ellipse	椭圆	cc.createObject("mycircle", "ellipse", 100, 100, 20, 20);	"mycircle"-对象名称，"ellipse"-图形名称，（100，100)，圆点坐标，（20，20）X，Y 方向的半径。如果相等则为圆，不等为椭圆。
rect	矩形	cc.createObject("myrect", "rect", 100, 100, 20, 20);	"myrect"-对象名称，"rect"-图形名称，(100，100)，左上角的坐标，(20, 20)，表示长和宽。

- 删除对象

当不再需要某个图形对象后，应该删除该对象，以便释放出更多的内存控件给系统使用。 mypoint 后，我们不能再对 mypoint 做任何操作，但是我们还可以再创建名为 "mypoint" 的对象，画布中的图形对象名称是唯一的，调用 createObject 不会创建重复的对象。

```
cc.deleteObject("mypoint");
```

7.2.2 设置图形

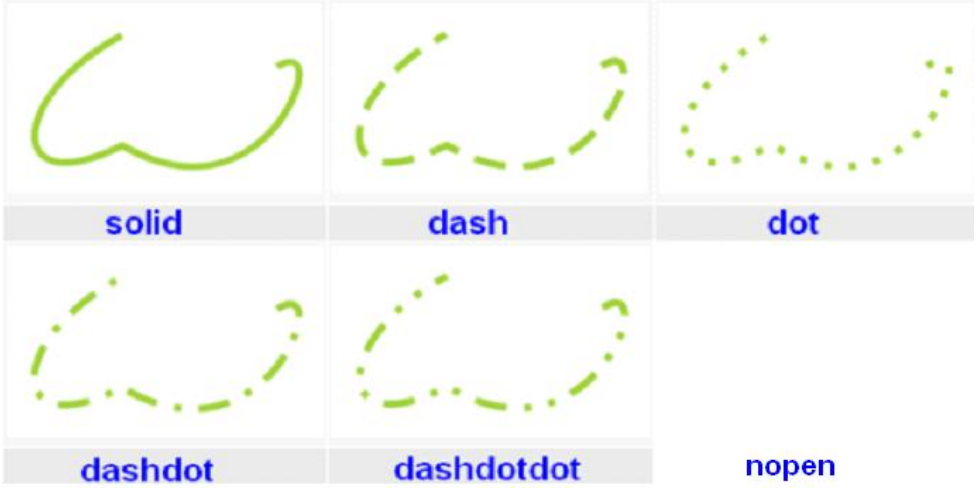
- 更改前景色和风格

可以调用下面的方法设置画笔属性：

```
cc.setObjectPalette("mypoint", "pen", "#FFFF00", "solid", 2);
```

参数 "mypoint"是图形对象名称，参数 "pen"表示当前选择的是画笔， 参数"#FFFF00"，表示画笔的颜色（格式必须是 "#" 加上颜色 RGB 的十六进制值，如红色"#FF0000"，黄色 "#00FF00"，蓝色 "#0000FF"，白色 "#FFFFFF"，黑色 "#000000")，参数 "solid" 表示画笔的风格,参数 2 表示画笔粗细为 2 个像素。

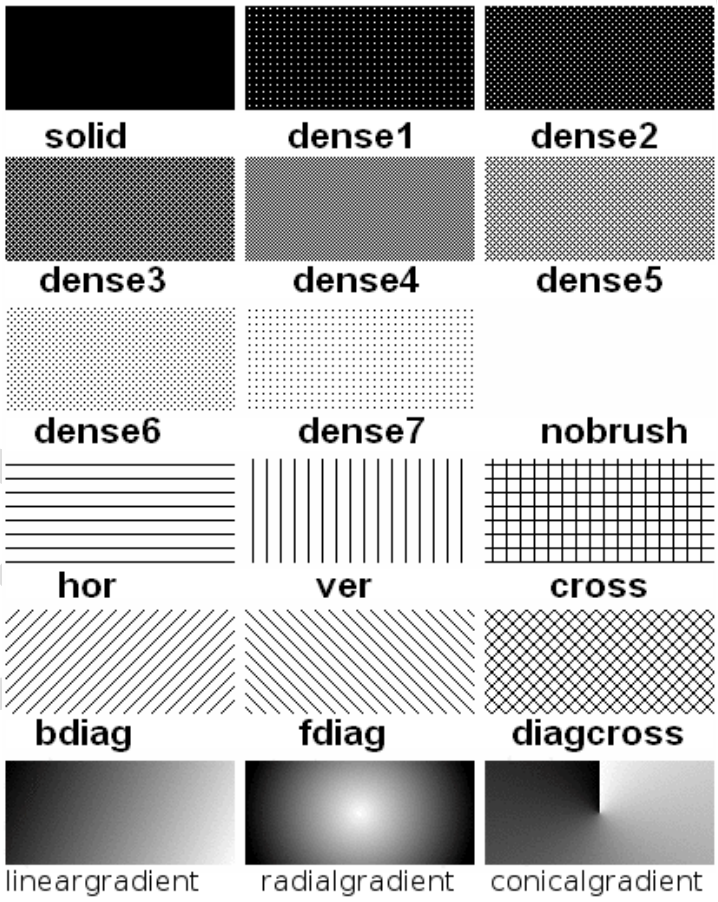
支持的画笔风格类型如下：



更改背景色和填充模式

可以调用下面方法设置画刷属性：

```
cc.setObjectPalette("mypoint", "brush", "#FFFF00", "dense1");
```



参数 "mypoint"是图形对象名称，参数 "brush"表示当前选择的是画刷， 参数"#FFFF00"，表示画刷的颜色（格式必须是 "#" 加上颜色 RGB 的十六进制值，如红色"#FF0000"， 黄色 "#00FF00"， 蓝色 "#0000FF"， 白色 "#FFFFFF"， 黑色 "#000000")，参数 "dense1" 表示画刷的风格。支持的画刷风格类型如上图。

注意：画刷不用设置粗细， 所以方法 setObjectPalette 后面不需要带粗细参数；画刷仅对矩形，圆等能填充的图形有效， 而像点， 线这样的图形不适用。

7.2.3 更改图形

- 扩展图形

画同一类型的图形，不用再创建新的对象，可以调用下面的方法来实现扩展：

```
cc.extendObject("mypoint", 180, 180);
```

参数 "mypoint"，是图形对象的名称，参数（180, 180），是新扩展的点对象的 X,Y 坐标。这样,我们的 mypoint 这个对象就对应了两个点图形（100, 100）和（180, 180），我们可以把它们看作是一个整体，我们可以同时对这两个图形做相应的操作（移动，缩放，等等）。

在执行扩展操作前，通过 setObjectExtention 函数设置操作基本属性，如：

```
cc.setObjectExtention("mypoint", "before");
```

设置名字为 mypoint 的对象的扩展方式为：前面扩展（扩展的绘图操作在原对象的左边进行）。

- 清除图形

清除完对象后，之前通过该图形对象画的图形将全部清空，如：

```
cc.clearObject("mypoint"); //清除 mypoint 对象中的所有图形
```

再清除完对象图形后，我们还可以再通过 cc.extendObject("mypoint", 180, 180)；这样的方法重新添加新的图形。

- 移动图形

移动一个图形对象，可以调用下面的方法来实现。支持两种方式的移动（相对和绝对移动），相对移动就是平移，每次移动指定的 X，Y 值；绝对移动，就是每次移动到指定的 X，Y 坐标位置。(缺省情况下是相对移动)

如果想使用绝对移动的方式，我们可以调用下面的方法。其中参数 "mypoint"， 是图形对象的名称，参数 "absolute" 表示绝对移动。

```
cc.setObjectMoving("mypoint", "absolute");
```

设置完移动的方式后，可通过脚本方法来实现图形的移动。参数 “mypoint” 是图形对象的名称，参数(1, 0) 表示每调用一次 moveObject 方法，图形对象

X 方向向右移动一个像素，Y 方向不变，也就是水平向右移动， 如果想让它水平向左移动，参数设置成（-1, 0）即可。如果前面设置的是绝对移动，那么参数(1,0)表示调用一次 moveObject 方法，将图形移动到坐标（1, 0）这个位置。

```
cc.moveObject("mypoint", 1, 0);
```

7.2.4 刷新图形

画布默认是画布全部区域刷新。其实，并不是每次图形操作都要画布上的全部区域重新绘制一遍，为提高画布控件的图形绘制的效率，在进行图形对象操作（创建、移动、更改等）前，根据需要，设定画布的刷新活动区域（该区域为矩形）。调用下面的方法来指定需要更新的画布区域：

```
cc.setActiveRegion(1, 2, 100, 200);
```

参数 1 和 2 分别是区域矩形的左上角 X 和 Y 坐标，100 是指定矩形区域的长，200 是指定区域的宽。

7.2.5 函数列表

方法名称	用途	使用方法	参数说明
createObject	创建一个图形对象，可以管理多个同一类型的图形（如多个点，对条线，等等）	cc.createObject("mypoint", "point", 100, 100);	"mypoint"-对象名称，"point"-图形名称，（100， 100），点的坐标
deleteObject	删除一个图形对象	cc.deleteObject("mypoint");	"mypoint"-对象名称
clearObject	清空一个图形对象，里面的图形全部清除	cc.clearObject("mypoint");	"mypoint"-对象名称
setObjectPalette	设置图形对象的调色板，包括画笔和画刷的颜色，风格等。	cc.setObjectPalette("mypoint", "pen", "#FFFF00", "solid", 2);	"mypoint"-对象名称，"pen"或"brush" - 画笔或画刷，"#FFFF00" - 画笔或

		cc.setObjectPalette("mypoint", "brush", "#FFFF00", "solid");	画刷颜色，"solid"-画笔或画刷风格，2 - 画笔的粗细。
setObjectExtention	设置图形对象的扩展方式（追加或插入），主要应用场合是，用 polyline 来画波形，控制波形的数据更新方向，左移或右移。	cc.setObjectExtention("mypoint", "after"); cc.setObjectExtention("mypoint", "before");	"mypoint"-对象名称，"after" - 表示扩展对象是以追加的方式，"before"- 表示扩展对象是以插入的方式。缺省是追加方式
extendObject	扩展图形对象，增加同一类型的图形到当前图形对象中来统一管理。	cc.extendObject("mypoint", 10, 20); cc.extendObject("myline", 10, 10, 80, 80);	"mypoint"-对象名称，(10, 20)-新扩展点的坐标，(10, 10, 80, 80) - 新扩展直线的 x1, y1, x2, y2 坐标。
setObjectMoving	设置图形对象移动的方式（相对或绝对）	cc.setObjectMoving("mypoint", "relative"); cc.setObjectMoving("mypoint", "absolute");	"mypoint"-对象名称，"relative"- 对象移动的方式是相对坐标，"absolute"- 移动的方式是绝对坐标。
moveObject	移动图形对象	cc.moveObject("mypoint", 1, 0); cc.moveObject("mypoint", -1, 0);	"mypoint"-对象名称，(1, 0) - 如果是相对方式移动，则表示往右水平移动一个像素；如果是绝对方式移动，则移动到坐标 (1, 0) 处。
setActiveRegion	设置画布的刷新区域	cc.setActiveRegion(5, 5, 100, 100);	不用指定图形对象，(5, 5, 100, 100）-表示矩形区域。

关于画布控件的具体用法，请参看光盘中“Designer_Demo\专题演示\4.3-画布控件演示”。

- 方法(Method)

  createObject：创建图形对象；

  deleteObject：删除图形对象；

  setObjectPalette：设置图形对象的绘图操纵调色板属性；

  setObjectExtention：设置图形对象的扩展方式；

  extendObject： 将已有的图形对象扩展出一个新的元素；

  setObjectMoving：设置图形对象的移动方式；

  moveObject：移动图形对象；

  clearObject：删除图形对象；

  setActiveRegion：设置图形对象的刷新区域；

- 例子

  cc.createObject("mypoint", "point", 100, 100);

  在坐标(100, 100)处创建一个名字为 mypoint 的点对象

  cc.deleteObject("mypoint");

  删除名字为 mypoint 的点对象

  cc.setObjectPalette("mypoint", "pen", "#FFFF00", "solid", 2);

  设置名字为 mypoint 的点对象的绘图操作调色板属性：颜色为 RGB 十六进制表示 0xFFFF00 的颜色，线条风格为 solid，线条宽度为 2 个像素

  cc.setObjectPalette("mypoint", "brush", "#FFFF00", "solid");

  设置名字为 mypoint 的点对象的填充操作调色板属性：颜色为 RGB 十六进制表示 0xFFFF00 的颜色，线条风格为 solid

  cc.setObjectExtention("mypoint", "after");

  设置名字为 mypoint 的对象的扩展方式为：后面扩展（扩展的绘图操作在原对象的右边进行）

  cc.setObjectExtention("mypoint", "before");

  设置名字为 mypoint 的对象的扩展方式为：前面扩展（扩展的绘图操作在原对象的左边进行）

  cc.extendObject("mypoint", 10, 20);

将对象 mypoint 扩展一个新元素（如果 mypoint 为一个点对象，则会在坐标(10，20)处增加一个点，这个点的位置按 setObjectExtention 的设置，此时的两个点同属 mypoint 对象）

cc.extendObject("myline", 10, 10, 80, 80);

将对象 myline 扩展一个新元素（如果 myline 为一个线对象，则会在坐标(10，10)到(80，80)间绘制一条直线，这条直线的位置按 setObjectExtention 的设置，此时的两条直线同属 myline 对象）

cc.setObjectMoving("mypoint", "relative");

设置 mypoint 对象的移动方式：相对移动 cc.setObjectMoving("mypoint", "absolute");设置 mypoint 对象的移动方式：按绝对坐标移动

cc.moveObject("mypoint", 1, 0);

将 mypoint 对象向右移动一个像素，向上不移动（假设 setObjectMoving 设置的移动方式为相对移动）

cc.moveObject("mypoint", -1, 0);

将 mypoint 对象向左移动一个像素，向上不移动（假设 setObjectMoving 设置的移动方式为相对移动）

cc.clearObject("mypoint");

清除名字为 mypoint 点对象内容

cc.setActiveRegion(5, 5, 100, 100);

设置画布 cc 的刷新区域为坐标(5，5)到(100，100)间的矩形区域,不在此区域的对象，执行 extend, move 操作，看不到屏幕有变化。设置合理的刷新区域能有效优化系统刷新速度。

7.3 图片

- 功能介绍：

  图片控件是用于显示各种图片的容器，支持的图片格式包括 bmp、jpg、png、tiff 和 gif 动画图片。如果在界面中显示一幅图片，只需设置图片控件的源图片属性即可。

- 属性(Property)

  source： 设置图片源，PS-LCD 默认查找当前工程的 spf/picture 目录，用户需要将要显示的图片拷贝到该目录下；

  scaledContents： 该属性将拉伸图片到适应控件尺寸；

  speed： 设置 gif 动画图片的播放速度；

- 方法(Method)

play(int)：播放 gif 图片动画指定次数，当指定次数为 0 时，将连续播放动画直到调用 stop()方法才停止；

pause(bool)：暂停/恢复 gif 动画播放；

stop()：停止播放 gif 动画；

frameNumber()：获得当前播放 gif 动画的帧号；

- 事件(trigger Event)

sourceChanged：当图片源发生改变时产生；

frameChanged：当播放的动画帧改变时产生；

loopDone：当动画播放完一个循环时产生；

- 例子

xxx.source='1.gif';

设置图片为 1.gif，并播放该图片动画；

xxx.source='2.jpg';

设置图片为 2.jpg，并显示该图片；

xxx.scaledContents=1;

将图片拉伸到适应控件尺寸；

xxx.speed=200;

设置动画图片的播放速度为原来的 200%；

xxx.play(0);

无限次播放 gif 动画，直到调用 stop()方法停止；

xxx.play(1);

播放 gif 动画一次循环，然后自动停止；

xxx.pause(1);

停止 gif 动画播放；

xxx.pause(0);

恢复 gif 动画播放；

var i=xxx.frameNumber();

获取当前播放的 gif 动画的帧号；

7.4 标签

- 功能介绍：

标签用于显示文字说明信息，并且支持编写脚本改变标签控件的显示文本以响应动态事件，标签文字的颜色可以通过属性进行设置。

- 属性(Property)

text：指定标签显示的内容；

- 方法(Method)

clear()：清除指定标签的文本内容；

- 事件(trigger Event)

textChanged：当标签的文本内容改变时产生

- 例子

xxx.text='Hello World!'

指定标签 xxx 的显示内容为 Hello World！

xxx.clear();

清除标签 xxx 的文本内容

7.5 文本按钮

- 功能介绍：

文本按钮可设置为普通按钮和开关按钮。当选中“可选择”属性时为开关按钮。系统默认为普通按钮。

- 属性(Property)

text：指定按钮上显示的文本内容；

flat：指定按钮形状，平坦或立体；

checked：按钮状态，按下或抬起，在开关按钮中可以根据按钮的不同状态执行不同的动作脚本；

- 方法(Method)

setCheckable(bool)：设置按钮是否为自锁定状态；

setChecked(bool)：设置按钮状态；

animateClick(int)：执行一次点击按钮操作，并在指定时间后自动恢复为抬起状态;

- 事件(trigger Event)

pressed：按下按钮时产生；

held：保持按下状态时产生（”自动重复”属性选中才会产生该事件）；

released：抬起按钮时产生；

all：按下和抬起按钮时都产生；

none：不产生任何事件；

- 例子

xxx.text='按钮'

设定按钮显示文字为‘按钮‘；

xxx.setChecked(1)

设置自锁按钮 xxx 为按下状态；

xxx.animateClick(100)

执行一次点击按钮 xxx 的操纵，100ms 后按钮状态自动恢复为抬起状态；

If (xxx.currentTrigger() ==”pressed”)

…

Else

…

如果按钮 xxx 的状态为按下时，执行相关动作（一般用在按钮动作脚本中）。

7.6 图片按钮

- 功能介绍：

  图片按钮在界面中外观就是一张图片，但能实现与文本按钮类似的功能，用户可自由定义图片按钮按下和抬起时的显示图片，以实现按下和抬起动态效果。

- 属性(Property)

  text：指定按钮上显示的文本内容；

  checked：按钮状态，按下或抬起，在开关按钮中可以根据按钮的不同状态执行不同的动作脚本；

- 方法(Method)

  setCheckable(bool)：设置按钮是否为自锁定状态；

  setChecked(bool)：设置按钮状态；

  animateClick(int)：执行一次点击按钮操作，并在指定时间后自动恢复为抬起状态;

- 事件(trigger Event)

  pressed：按下按钮时产生；

  held：保持按下状态时产生（”自动重复”属性选中才会产生该事件）；

  released：抬起按钮时产生；

  all：按下和抬起按钮时都产生；

  none：不产生任何事件；

- 例子

  xxx.text=’按钮’

  设定按钮显示文字为‘按钮‘；

  xxx.setChecked(1)

  设置自锁按钮 xxx 为按下状态；

  xxx.animateClick(100)

执行一次点击按钮 xxx 的操纵，100ms 后按钮状态自动恢复为抬起状态；

If (xxx.currentTrigger() ==”pressed”)

…

Else

…

如果按钮 xxx 的状态为按下时，执行相关动作（一般用在按钮动作脚本中）。

7.7 文本框

- 功能介绍：

  文本框是显示和输入文本的主要控件，几乎所有的输入动作都是利用文本框完成的。文本框分为单行文本框和多行文本框，当文字内容超过文本框尺寸时，多行文本框会自动出现滚动条。在界面运行时，点中非只读属性的文本框，系统自动弹出软键盘供用户输入数据，输入完毕，选中软键盘上的 CANCEL 或者 OK 键，软键盘自动消失。

- 属性(Property)

  text： 指定文本框的显示内容；

  frame： 用于设置文本框的边框形状

  cursorPositoin： 返回光标所在文本框的位置；

  readOnly： 设置文本框为只读；

- 方法(Method)

  clear()：清除文本框的内容；

  selectAll()：选择文本框中的所有内容；

  cut()：剪切文本框内容；

  copy()：拷贝文本框内容；

  paste()：粘贴文本框内容；

- 事件(trigger Event)

  textChanged： 当文本框的内容改变时产生；

- 例子

  xxx.text=’Hello123’

  指定文本框的内容为 Hello123；

  xxx.copy();

  复制文本框 xxx 中选定的文本内容；

  xxx.paste();

  粘贴复制的内容到文本框 xxx；

xxx.frame=1;

设置文本框有边框；

xxx.readOnly=1;

设置文本框为只读，点击该文本框时，不会自动弹出软键盘；

7.8 进度条

- 功能介绍：

  用于动态显示进度状况。

- 属性(Property)

  value： 设置进度条的当前位置；

  format： 设定进度条的文字显示方式，百分比或实际值；

- 方法(Method)

  reset()： 设置进度条及文字显示为零；

  setRange(min，max)： 设置进度条显示的位置范围；

- 事件(trigger Event)

  valueChanged： 当进度条的值改变时产生；

- 例子

  xxx.setRange(0,100);

  设置进度条的位置范围为 0-100；

  xxx.format(“%p”);

  设定进度条 xxx 的文字显示为百分比；

  xxx.format(“%v”);

  设定进度条 xxx 的文字显示为实际值；

7.9 定时器

- 功能介绍：

  提供定时功能。当定时时间间隔到时，可使其执行一段脚本，从而实现动画和刷新界面等效果，该控件在界面运行时不可见。

- 属性(Property)

  Interval： 设定定时器的时间间隔；

- 方法(Method)

  run(int)：启动定时器并执行指定次数定时，如果指定次数为 0，则无限次重复定时功能，直到调用 stop()方法停止；

  stop()：停止定时器；

- 事件(trigger Event)

timeout： 定时器到时时产生；

- 例子

xxx.interval=500;

设定定时器 xxx 的时间间隔为 500ms；

xxx.run(0);

启动定时器 xxx，自动重复定时和执行动作脚本，知道调用 xxx.stop()后停止；

xxx.run(2);

启动定时器 xxx，自动重复定时和执行动作脚本 2 次，或调用 xxx.stop()后停止；

xxx.stop();

停止定时器 xxx；

7.10 下拉菜单

- 功能介绍：

提供若干种选择项供使用者选择，每次只能选其中一种；选项在 designer 中编辑输入，方法是：双击下拉菜单控件，在弹出的界面中点击+增加选项；点击属性增加选项图标。在界面运行过程中也可以动态添加/删除选项。

- 属性(Property)

currentText： 设定下拉菜单的当前文本值；
currentIndex： 设定下拉菜单的当前索引值；
editable： 设定下拉菜单的文本内容为可编辑状态；
count： 返回下拉菜单中选项的总数；

- 方法(Method)

insertOption(int i, string s)：在索引值为 i 的位置插入选项 s，i 后面的其他选项顺次向后移动；
removeOption(int i)： 删除索引 i 处的选项，i 后面的选项顺次向前移动；

- 事件(trigger Event)

IndexChanged：当用户选择某个选项，下拉菜单的索引值发生改变时产生；

- 例子

xxx.currentIndex=2;

设定下拉菜单 xxx 的当前索引值为 2；

xxx.insertOption(2, "test");

在下拉菜单 xxx 索引 2 处增加 test 文本选项

xxx.removeOption(2);

删除下拉菜单 xxx 索引值为 2 的选项

7.11 复选框

- 功能介绍：

  如果用户选中该复选框，复选框动作脚本运行，可按用户需求编写脚本执行相应动作。另外，用户也可在其他控件的动作脚本中查询复选框对象的 checked 属性来查询复选框是否选中。

- 属性(Property)

  checked：复选框是选中状态；

- 事件(trigger Event)

  checked：复选框被选中是产生；

  unchecked：复选框未被选中时产生；

- 例子

  xxx.checked=1;

  选择复选框 xxx；

  xxx.checked=0;

  取消选择复选框 xxx；

7.12 滑动尺

- 功能介绍：

  当滑动块位置变化时，该控件会产生事件，可在动作脚本按需求编写相应动作。另外，用户也可在其他控件的动作脚本中设置或查询滑动块位置。

- 属性(Property)

  value：设定滑动块的当前位置；

  maximum：设定滑动块的最大位置值；

  minimum：设定滑动块的最小位置值；

- 事件(trigger Event)

  valueChanged：当滑动块的位置值改变时产生；

- 例子

  xxx.value=50;

  设定滑动块的当前位置为 50；

xxx.maximum=100;

设定滑动块位置的最大值为 100；

7.13 段式数字

- 功能介绍：

  模拟老式数码管样式显示数字

- 属性(Property)

  value： 设定段式数字的显示值；

- 事件(trigger Event)

  valueChanged： 当段式数字的值改变时产生；

- 例子

  xxx.value=100;

  设定段式数字的显示值为 100；

7.14 刻度尺

- 功能介绍：

  带刻度显示的位置指示器，刻度范围和风格可通过控件属性定义。

- 属性(Property)

  value： 设定刻度尺指针的刻度值；

- 事件(trigger Event)

  valueChanged： 当刻度尺的指针位置改变时产生；

- 例子

  xxx.value=10;

  设定刻度尺的指针到 10 刻度处；

7.15 仪表盘

- 功能介绍：

  可定制背景、指针风格的圆形进度指示器。该控件为用户设计各种仪表提供了极大灵活性：通过“源图片”属性来选择仪表盘的背景图片来定义仪表类型。仪表盘的大小可以通过拖动改变，指针的风格可以通过属性来定义粗细、长短和颜色，指针的转动角度也可以自定义。

- 属性(Property)

value：设定仪表盘显示的百分比；

- 事件(trigger Event)

valueChanged：当仪表盘的显示值改变时产生；

- 例子

xxx.value=10;

设定仪表盘指针到量程的 10%处；

7.16 波形

- 功能介绍：

波形控件用于实时显示波形曲线。

- 属性(Property)

value：设定波形曲线的 y 轴坐标，x 轴坐标自动加 1；

- 例子

x.value='10,20,30';

设定三条波形曲线的 y 轴坐标分别为 10，20，30，x 周坐标自动加 1；

7.17 日期时间

- 功能介绍：

用于显示当前的日期和时间，并有闹钟功能（在 PS-LCD 中，目前为软时钟，每次系统上电时，外部 CPU 需初始化该时钟为正确时间）。

- 属性(Property)

time：设定当前显示时间
data：设定当前显示日期
alarm：设定闹钟报警时间

- 事件(trigger Event)

alarmed：当闹钟时间到时产生；

- 例子

xxx.time="10:28:55";

设定当前时间是 10:28:55；

xxx.data="2013-01-11";

设定当前日期为 2013 年 1 月 11 日；

xxx.alarm="10:30:00";

设定闹钟在 10:30:00 执行动作脚本；

第八章 定制界面风格

通过前面章节的学习，我们已经能够生成、下载和运行自己设计的界面，具备了开发 PS-LCD 界面的基本概念和技能，完全可以胜任产品开发工作。但针对界面要求比较高场合，需更多动态效果，设计更加个性的界面样式，本章将详细阐述通过样式表(Style Sheet)来实现上述功能。

样式表（Style Sheet）是一段字符串文本，定义了如何显示各种控件的显示样式，能自如地同时改变一个或者多个控件的布局和外观。

这种改变，既可以在 Designer 设计阶段通过控件的“样式表(Style Sheet)”属性静态定义，也可以在界面运行时通过脚本改写该属性内容动态实现，为开发者提供了极大的灵活性。

8.1 样式表语法

样式表由两个主要的部分构成：**选择器 + 一条或多条声明**

```
selector {declaration1; declaration2; ... declarationN }
```

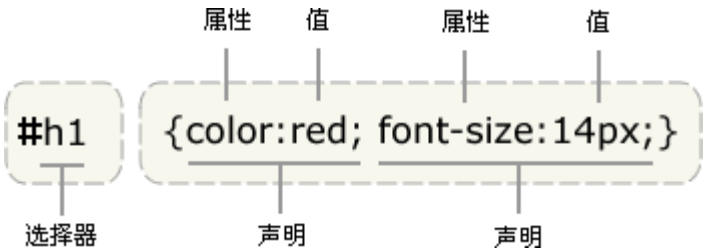
选择器通常是需要改变样式的控件名称（需在名字前加#），每条声明由一个属性和一个值组成。不同声明间被引号分开。属性（property）是您希望设置的样式属性，每个属性都有值。属性和值被冒号分开。

```
selector {property: value}
```

下面这行代码的作用是将名字为 h1 的控件内文字颜色定义为红色，同时将字体大小设置为 14 像素。在这个例子中，h1 是选择器，color 和 font-size 是属性，red 和 14px 是值。

```
#h1 {color: red; font-size:14px;}
```

下面的示意图为您展示了上面这段代码的结构：



提示：请使用花括号来包围声明。

用选择器可同时指定多个控件名字，用逗号将不同的控件名称分开，分享相同的声。。在下例中，名称为 h1 到 h6 所有控件字体颜色设置为绿色。

```
#h1,#h2,#h3,#h4,#h5,#h6 {color: green;}
```

样式表的风格，子控件从父控件那里继承。如在界面控件(form)样式表里面设置的风格，子控件（如：放在界面中的按钮控件）将继承其风格。

这里需要特别注意的是，如果针对明确的特定控件设置样式表，格式中的选择器和花括号可以忽略。如：同样完成设置 xxx 控件的样式，

在“动作脚本(action) ”中的写法如下：

```
xxx.styleSheet="color: green; background-color: blue;"
```

在 xxx 控件的“样式表(Style Sheet) ”属性中输入内容如下：（如果输入的样式表语法有问题，将无法点击样式表属性对话框中的 OK 按钮）

```
color: green; background-color: blue;
```

8.2 基本样式定义

8.2.1 颜色

color 设置前景颜色，一般为控件上文本颜色。下例设置 h1 控件前景色为红：

```
#h1 {color: red}
```

使用 background-color 属性为控件设置背景色。这个属性接受任何合法的颜色值。这条规则把元素的背景设置为灰色：

```
#p {background-color: gray;}
```

如果您希望背景色从元素中的文本向外少有延伸，只需增加一些内边距：

```
#p {background-color: gray; padding: 20px;}
```

selection-color 和 selection-background-color 分别对应选中区域的前景色和背景色，定义方法与上面类似。

8.2.2 文本

通过文本属性，您可以改变文本的颜色、转换、装饰文本等等.

- text-transform

处理文本的大小写。这个属性有 3 个值：none、uppercase、lowercase。默认值 none 对文本不做任何改动，将使用原有大小写。顾名思义，uppercase 和 lowercase 将文本转换为全大写和全小写字符。作为一个属性，text-transform 可能无关紧要，不过如果您突然决定把所有 h1 元素变为大写，这个属性就很有用。不必单独地修改所有 h1 元素的内容，只需使用 text-transform 为你完成这个修改：

```
#h1 {text-transform: uppercase}
```

使用 text-transform 有两方面的好处。首先，只需写一个简单的规则来完成这个修改，而无需修改 h1 元素本身。其次，如果您以后决定将所有大小写再切换为原来的大小写，可以更容易地完成修改。

- text-align

该属性定义文本的对齐方式，只对按钮和进度条控件有用，可以有如下值：

- top
- bottom
- left
- right
- center

- text-decoration

text-decoration 有 5 个值：none、underline、overline、line-through、blink。underline 会对元素加下划线，overline 的作用恰好相反，会在文本的顶端画一个上划线。值 line-through 则在文本中间画一个贯穿线，blink 会让文本闪烁。none 值会关闭原本应用到一个元素上的所有装饰。通常，无装饰的文本是默认外观。

8.2.3 图标

icon-size 属性设置图标大小，如下例将控件 h 的图标设为 16x16 像素大小：

```
#h {icon-size: 16px;}
```

8.2.4 字体

- font-style

font-style 属性最常用于规定斜体文本。该属性有两个值：normal - 文本正常显示、italic - 文本斜体显示。

- font-weight

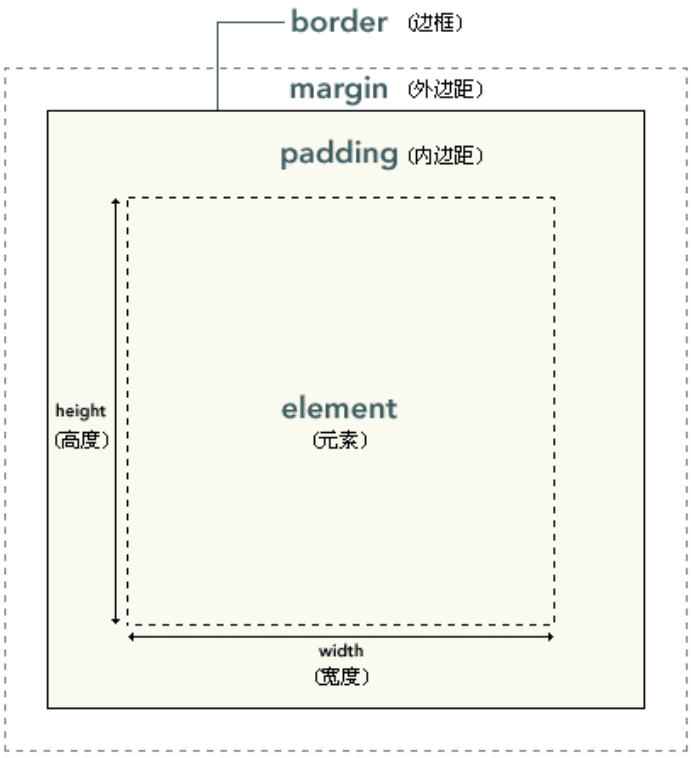
font-weight 属性设置文本的粗细。使用 bold 关键字可以将文本设置为粗体。关键字 100 ~ 900 为字体指定了 9 级加粗度。如果一个字体内置了这些加粗级别，那么这些数字就直接映射到预定义的级别，100 对应最细的字体变形，900 对应最粗的字体变形。数字 400 等价于 normal，而 700 等价于 bold。实例：

```
#p {font-weight:normal;}
```

- font-size

font-size 属性设置文本的大小，单位为像素。实例：#h1 {font-size:60px;}

8.3 框模型

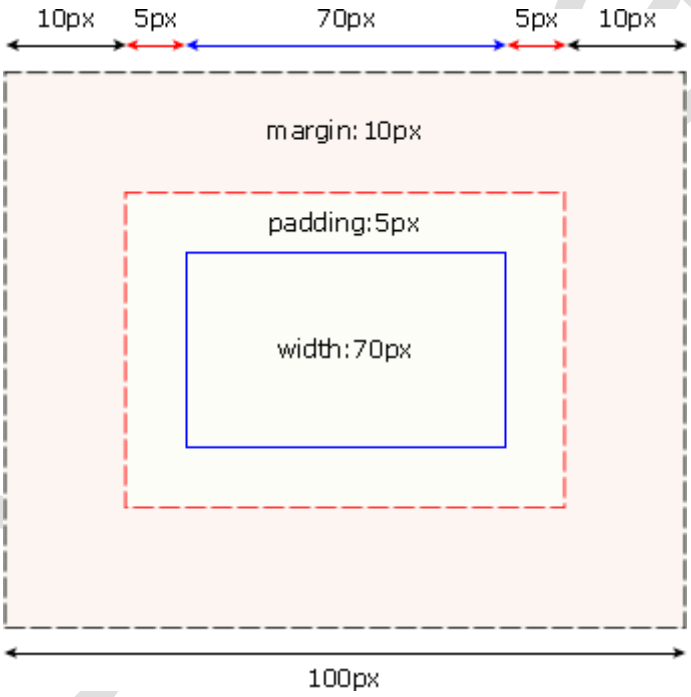


框模型(Box Model)规定了控件框处理内容、内边距、边框和外边距的方式。

控件框的最内部分是实际内容，直接包围内容的是内边距。内边距呈现了控件的背景。内边距的边缘是边框。边框以外是外边距，外边距默认是透明的，因此不会遮挡其后的任何元素。提示：背景应用于由内容和内边距组成的区域。内边距、边框和外边距都是可选的，默认值是零。内边距、边框和外边距可以应用于一个控件的所有边，也可以应用于单独的边。

在样式表中，width 和 height 指的是内容区域的宽度和高度。增加内边距、边框和外边距不会影响内容区域的尺寸，但是会增加元素框的总尺寸。假设框的每个边上有 10 个像素的外边距和 5 个像素的内边距。如果希望这个元素框达到 100 个像素，就需要将内容的宽度设置为 70 像素，见下图：

```
#box { width: 70px;   margin: 10px; padding: 5px;}
```



- element：元素。
- padding：内边距，也有资料将其翻译为填充。
- border：边框。
- margin：外边距，也有资料将其翻译为空白或空白边。

我们把 padding 和 margin 统一地称为内边距和外边距。边框内的空白是内边距，边框外的空白是外边距，很容易记吧。

8.3.1 内边距

padding 属性定义元素的内边距,值接受长度值或百分比值，但不允许使用负值。例如，如果您希望所有 h 控件的各边都有 10 像素的内边距，可这样：

```
#h {padding: 10px;}
```

还可以按照上、右、下、左的顺序分别设置各边的内边距，各边均可以使用不同的单位或百分比值：

```
#h {padding: 10px 10px 5px 2px;}
```

也通过使用下面四个单独的属性，分别设置上、右、下、左内边距：

- padding-top
- padding-right
- padding-bottom
- padding-left

下面的规则实现的效果与上面的简写规则是完全相同的：

```
#h { padding-top: 10px; padding-right: 10px; padding-bottom: 5px; padding-left: 2px; }
```

属　性	描　　述
padding	简写属性。作用是在一个声明中设置元素的所内边距属性。
padding-bottom	设置元素的下内边距。
padding-left	设置元素的左内边距。
padding-right	设置元素的右内边距。
padding-top	设置元素的上内边距。

8.3.2 边框

边框（border）是围绕控件内容和内边距的一条或多条线。border 属性允许你规定元素边框的样式、宽度和颜色。元素外边距内就是元素的的边框(border)。每个边框有 3 个方面：宽度、样式，以及颜色。

边框绘制在“元素的背景之上”。这很重要，因为有些边框是“间断的”（例如，点线边框或虚线框），元素的背景应当出现在边框的可见部分之间。边框的样式是边框最重要的一个方面，因为如果没有样式，将根本没有边框。

- border-style

border-style 属性定义了 10 个不同的样式:

值	描 述
none	定义无边框。
hidden	与 "none" 相同。不过应用于表时除外，对于表，hidden 用于解决边框冲突。
dotted	定义点状边框
dashed	定义虚线
solid	定义实线
double	定义双线。双线的宽度等于 border-width 的值。
groove	定义 3D 凹槽边框。其效果取决于 border-color 的值。
ridge	定义 3D 垄状边框。其效果取决于 border-color 的值。
inset	定义 3D 凹下效果边框。其效果取决于 border-color 的值。
outset	定义 3D 凸起效果边框。其效果取决于 border-color 的值。

例如，把一幅图片 i 的边框定义为 outset，使之看上去像是“凸起按钮”：

```
#i {border-style: outset;}
```

您可以为一个边框定义多个样式，例如：

```
#p {border-style: solid dotted dashed double;}
```

上面这条规则为名 p 的控件定义了四种边框样式：实线上边框、点线右边框、虚线下边框和一个双线左边框。我们又看到了这里的值采用了 top right bottom left 的顺序，讨论用多个值设置不同内边距时也见过这个顺序。如果您希望为

控件的某一个边设置边框样式，而不是设置所有 4 个边的边框样式，可以使用下面的单边边框样式属性：

- border-top-style
- border-right-style
- border-bottom-style
- border-left-style

因此这两种方法是等价的：

```
#p {border-style: solid solid solid none;}
#p {border-style: solid; border-left-style: none;}
```

注意：如果要使用第二种方法，必须把单边属性放在简写属性之后。因为如果把单边属性放在 border-style 之前，简写属性的值就会覆盖单边值 none。

border-width

border-width 属性为边框指定宽度。单位为像素，如 2px。所以，我们可以这样设置边框的宽度：

```
#p {border-style: solid; border-width: 5px;}
```

当然，也可以按照 top-right-bottom-left 的顺序设置元素的各边边框：

```
#p {border-style: solid; border-width: 15px 5px 15px 5px;}
```

您也可以通过下列属性分别设置边框各边的宽度：

- border-top-width
- border-right-width
- border-bottom-width
- border-left-width

因此，下面的规则与上面的例子是等价的：

```
#p {
    border-style: solid;
    border-top-width: 15px;
    border-right-width: 5px;
    border-bottom-width: 15px;
    border-left-width: 5px;
}
```

在前面的例子中，您已经看到，如果希望显示某种边框，就必须设置边框样式，比如 solid 或 outset。那么如果把 border-style 设置为 none 会出现什么情况：

```
#p {border-style: none; border-width: 50px;}
```

尽管边框的宽度是 50px，但是边框样式设置为 none。在这种情况下，不仅边框的样式没有了，其宽度也会变成 0。边框消失了，为什么呢？这是因为如果边框样式为 none，即边框根本不存在，那么边框就不可能有宽度，因此边框宽度自动设置为 0，而不论您原先定义的是什么。

记住这一点非常重要。事实上，忘记声明边框样式是一个常犯的错误。根据以下规则，所有 h1 元素都不会有任何边框，更不用说 20 像素宽了：

```
#h1 {border-width: 20px;}
```

由于 border-style 的默认值是 none，如果没有声明样式，就相当于 border-style: none。**因此，如果您希望边框出现，就必须声明一个边框样式。**

border-color

使用 border-color 属性设置边框颜色，它一次可以接受最多 4 个颜色值。可以使用任何类型的颜色值，例如可以是命名颜色，也可以是十六进制和 RGB 值：

```
#p {
  border-style: solid;
  border-color: blue rgb(25%,35%,45%) #909090 red;
  }
```

如果颜色值小于 4 个，值复制就会起作用。例如下面的规则声明了段落的上下边框是蓝色，左右边框是红色：

```
#p {
  border-style: solid;
  border-color: blue red;
  }
```

默认的边框颜色是控件本身的前景色。如果没有为边框声明颜色，它将与控件的文本颜色相同。另一方面，如果控件没有任何文本，假设它是一个表格，其中只包含图像，那么该表的边框颜色就是其父控件的文本颜色（因为 color 可以继承）。这个父控件很可能是界面(form)控件。还有一些单边边框颜色属性。它们的原理与单边样式和宽度属性相同：

- border-top-color
- border-right-color
- border-bottom-color
- border-left-color

要为 h1 元素指定实线黑色边框，而右边框为实线红色，可以这样指定：

```
#h1 {
  border-style: solid;
  border-color: black;
  border-right-color: red;
  }
```

我们刚才讲过，如果边框没有样式，就没有宽度。不过有些情况下您可能希望创建一个不可见的边框。样式表引入了边框颜色值 transparent。这个值用于创建有宽度的不可见边框。请看下面的例子：

```
#a {
  border-style: solid;
  border-width: 5px;
  border-color: transparent;
  }
```

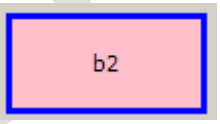
从某种意义上说，利用 transparent，使用边框就像是额外的内边距一样；此外还有一个好处，就是能在你需要的时候使其可见。这种透明边框相当于内边距，因为元素的背景会延伸到边框区域（如果有可见背景的话）。

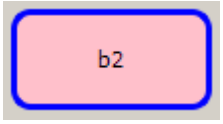
- border-radius

该属性用于设置控件的边框拐角弧度。默认的边框拐角弧度是 0，即直角，通过设置 border-radius 属性能定制出有弧度的控件，以按钮为例：

```
b2.styleSheet="border: 3px solid blue; border-radius: 0px";
b2.styleSheet="border: 3px solid blue; border-radius: 10px"
```

上面两行脚本执行完后的结果分别对应左右两幅效果图，从右图可看出 border-radius 属性发生了作用。





8.3.3 外边框

围绕在控件边框的空白区域是外边距(margin)。设置外边距会在元素外创建额外的“空白”。设置外边距的最简单的方法就是使用 margin 属性，值允许任何长度单位、百分数值甚至负值。可以设置为 auto。下面的声明在 h1 控件的各个边上设置了 2px 宽的空白：

```
#h1 {margin: 2px;}
```

下例为 h 控件的四个边分别定义了不同的外边距，单位是像素（px）：

```
#h1 {margin : 10px 0px 15px 5px;}
```

与内边距的设置相同，这些值的顺序是从上外边距（top）开始围着元素顺时针旋转的：top right bottom left。

margin 的默认值是 0，所以如果没有为 margin 声明一个值，就不会出现外边距。但是，在实际中，许多控件已经提供了预定的默认样式，外边距也不例外。当然，只要你特别作了声明，就会覆盖默认样式。

您可以使用下列任何一个属性来只设置相应的外边距，而不会直接影响所有其他外边距：

- margin-top
- margin-right
- margin-bottom
- margin-left

假设您希望把 p 元素的左外边距设置为 20px。不必使用 margin，而是可以采用以下方法：

```
#p {margin-left: 20px;}
```

另外，一个规则中可以使用多个这种单边属性，例如：

```
#h2 {
  margin-top: 20px;
  margin-right: 30px;
  margin-bottom: 30px;
  margin-left: 20px;
  }
```

当然，对于这种情况，使用 margin 更简介一些,下面语句效果跟上例一样。

```
#p {margin: 20px 30px 30px 20px;}
```

关于样式表的详细用法，可参考“**PS-LCD 软件速查表**”。

Greatal

附一 常见问题

界面设计常见问题

为什么 Designer 和 Flex 工具在 Win7 操作系统下运行不正常？

目前大器智成的 FLEX 下载工具仅在 Windows Xp 下安装上相关驱动后工作，Win7 及以上版本 Winodws 下无法正常工作。 采用 Win7 及以上版本 Winodws 的用户，建议采用 U 盘方式更新界面。

如何为界面选择字体？

Windows 中的矢量字库均能作为界面字体类型，提升界面视觉效果。选中 Designer 属性区中的字体属性，点击右边小按钮，在弹出的选择框中选择所需字体、字号和风格。

如何让界面风格与众不同？

Designer 默认的色彩比较单一呆板，通过调色板 Palette 来改变色彩设置，让你设计的图形界面色彩风格与众不同。如：在对象面板中选中某个按钮，然后在属性面板中选择“调色板”属性，点击“自定义”按钮，用户可以自己定义按钮的颜色特性。样式表是定制界面控件风格的一个好方法，可灵活实现各种界面样式，详情请参考第六章。

如何在界面中使用动画？

动画能让图形界面显得生动逼真，在 PS-LCD 上有两种方式实现动画效果：

- ✧ 直接使用 gif 格式图片。我们可以用 Windows 端专业的图片编辑工具如 photoshop 等生成所需 gif 图片，然后直接应用到图形界面中即可。由于 gif 仅支持 8bit 的图像， 最好使用在对动画图片颜色要求不高的场合。
- ✧ 使用 bmp，jpg，png，tiff 等格式的静态图片。例如：在界面运行时，利用定时器控件定时间隔执行脚本，如： xxx.value=1.png， xxx.value=2.png，…来实现动画。该种方式能显示 16/18bit 的高质量图片。

如果触摸屏定位不准，如何校准？

在界面运行时，按住界面的无触摸事件区域（如：图片控件区域，或者无控件区域），

并保持 5 秒以上，将自动进入校准程序。校准完毕后，校准数据自动保存在模组内，掉电不丢失。

? 如何在没有硬件键盘的情况下输入字母或者数字？

在界面运行时，点击单行或者多行文本框控件（该控件的“只读”属性必须不选中），输入软键盘自动弹出，即可进行输入操作。

? 如何将外接 4x4 键盘与界面关联？

在设计界面阶段，设置系统控件矩阵键盘(sysHardKeypad)的动作脚本来实现界面对按键事件的处理。例如：输入如下脚本，当按下外接矩阵键盘键值为 2 的按键，xxx 文本框控件的内容被设置为 123：

```
var c = sysHardKeypad.code();
if (c == 2 && sysHardKeypad.currentTrigger() == “pressed”)
    xxx.text = “pressed”;
```

? 如何定义供全局使用的函数？

在设计界面阶段，选中“工具->全局脚本”，即可打开 JS 脚本编辑器输入全局函数，该函数可供任何界面中的脚本调用，甚至可通过串口，外部控制器直接远程调用， 如外部控制器发送 func1(1,2) + 回车即可调用全局脚本中定义的 func1 函数。

? 我的界面很简单，但是为什么生成的 SPF 文件很大？

这极有可能是该界面选择的字体种类过多，请减少字体种类，建议一般产品界面字体少于 3 种，最好统一为一种字体，使界面简洁美观。

? 界面运行过程中，经常出现控件不存在的错误提示，如何解决？

先确认控件存在于哪个界面，然后用 Designer 打开工程界面，确保该界面的“快速显示”属性被选中。设置成快速显示的界面常驻内存，可确保界面被切换到后台后，脚本仍然可正常访问。

? 为什么在模拟器上调试好的界面，下载到 PS-LCD 运行，界面反应迟钝？

这可能是由于 PS-LCD 内存耗尽引起的。建议做如下检查和改进：

✧ 确认是否使用了远大于实际显示分辨率的图片，尤其 gif 动画图片。如：使用了一

个 2048x1024 分辨率的 gif 动画图片，但是实际界面显示使用的是 300x150 分辨率。这将造成极大的系统内存资源浪费，造成界面运行缓慢。建议所有图片的分辨率与实际显示的分辨率相等或者相近；

- ✧ 是否过多的使用定时器。例如：在界面中定义了 5 个定时器，每个事件间隔为 50ms，某个定时器动作脚本都输入大量的刷新控件操作或者运算，这样系统将大部分资源用于处理这些频繁的定时器事件，造成响应缓慢；
- ✧ 是否将大量的界面设置为“快速显示”。设置为“快速显示”的界面和其上的控件将永远不会从内存释放，大量这样的界面可能造成系统内存消耗完毕；
- ✧ 界面中是否使用了过多的字体种类。建议一个产品上只使用一到两种字体，字体种类过多，不仅耗费系统资源，而且界面显得繁琐。

通讯常见问题

? 到底在我的工程中该选择哪种协议？

如果 PS-LCD 与外部控制单元通讯数据量小，而且 PS-LCD 发送到控制单元的数据较少，可使用 CTP 通讯协议，三条指令完成交互，避免制定协议的麻烦。但是，如果交互过程比较多或者外部控制器有自己专门协议，推荐使用自定义通讯协议，可最大限度降低通讯的数据量，灵活实现界面动作；

? 当我点击界面中的按钮，外部控制器没有收到事件消息？

在 CTP 协议模式下，要让控件在状态变化时自动发送事件消息给外部控制器，必须在 Designer 生成界面时，选中该界面按钮的“事件通知”属性；如果已经选中了，仍无法收到消息，请检查串口波特率是否设置正确。

? 外部控制器如何确保发送的命令已经被 PS-LCD 正确执行？

在 CTP 通讯模式下，PS-LCD 在收到通讯命令后，会立刻执行，结束后会向外部控制器返回结果。如果外部控制器收到“C+”字符串，表示执行成功，可以发送下一条指令。“C-”表示执行失败，可尝试再次发送，或者检查错误原因。

? 在 CTP 协议模式下，如何取消 PS-LCD 命令执行结果的自动回复消息？

在 CTP 通讯模式下，PS-LCD 在收到命令并执行结束会向外部控制器返回执行结果，通过调用 **sysManager.ctpReply=0** 可以取消此消息的自动回复。

PS-LCD 能自定义通讯协议吗？

PS-LCD 支持两种协议通讯模式，选择 UserDefine 通讯模式，即可实现自定义协议，这样可灵活与各种外部设备连接。请参考开发光盘中“**专题演示\4.3-带帧格式自定义通讯演示**”。

如何在 CTP 协议模式下自定义串口消息？

如果要定义某个界面事件发生时（如按下按钮），发送特定二进制编码给外部控制器，可利用 **sysCom0.write()**函数自由定义，详情请参看“PS-LCD 软件速查表”中的用法。

在 CTP 通讯模式下，外部控制器更新界面控件比较容易，但是 PS-LCD 自动发送过来的界面事件字符串在外部控制器这边不好处理和识别，有什么改进办法吗？

可以取消控件的“事件通知”属性，直接在控件动作脚本中调用 **sysCom0.write()**函数实现自定义发送数据至主控，可简化外部控制器端的软件设计。

在 CTP 通讯模式下，如果外部控制器想一次更新不同界面控件内容，将发送大量的类似 xxx.yyy=zzz 的 CTP 命令，有办法简化吗？

可以通过远程调用方法简化。如：主控直接发送字符串 fresh(0x55, 0x66, 0xx77); 给 PS-LCD，一次刷新三个控件。PS-LCD 上用脚本实现 fresh 函数如下：

```
Function fresh(p1, p2, p3)
{
    Xxx.value = p1;
    Yyy.text = p2;
    Zzz.text=p3;
}
```

这实际相当于，主控可远程调用 PS-LCD 上的脚本函数，传递实时参数，与在动作脚本

中调用此函数实现同样的效果。

Greatal