

APCAD 应用开发指南

目录

第一部分 准备篇.....	6
第一章 APCAD 开发简介.....	6
1.1 APCAD 的简介.....	6
1.2 APCAD 的系统架构.....	7
1.3 APCAD 的应用层接口介绍.....	7
第二章 APCAD 开发环境搭建.....	8
2.1 APCAD 的开发准备工作.....	8
2.2 APCAD 的注册和登录.....	8
2.2.1 用户注册.....	8
2.2.2 用户登录:	9
2.2.3 用户购买非免费软件.....	10
2.3.4 购买完成。.....	11
2.3 APCAD 的安装和配置.....	11
2.4 创建第一个 APCAD 项目——HelloWorld.....	12
第二部分 基础篇.....	13
第三章 APCAD 程序设计基础.....	13
3.1 Lua 语言介绍.....	13
3.1.1 什么是 Lua?	13
3.1.2 Lua 编译运行.....	13
3.1.3 Lua 基本语法简介.....	13
3.2 IUP 介绍.....	14
3.2.1 什么是 IUP?	14
3.2.2 为什么使用 IUP?	14
3.2.3 如何学习 IUP?	14
3.3 IUP 常用控件简介.....	15
3.3.1 对话框 (DIALOGS)	15
3.3.2 文件选择对话框 (FILEDLG)	20
3.3.3 消息对话框 (MESSAGE)	22
3.3.4 提示对话框 (ALARM)	23
3.3.5 垂直排版容器 (VBOX)	24
3.3.6 水平排版容器 (HBOX)	25
3.3.7 文本编辑 (TEXT)	26
3.3.8 按钮 (BUTTON)	32
3.3.9 标签 (LABEL)	35
3.3.10 列表控件 (LIST)	38
3.3.11 矩阵控件 (MATRIX)	42
3.3.12 树形控件 (TREE).....	50
3.3.13 菜单控件 (MENUS)	57
3.3.14 单选控件 (RADIO)	59
3.3.15 多选控件 (TOGGLE).....	59
3.3.16 框架控件 (FRAME).....	61
3.3.17 分页控件 (TABS)	62

3.4 平台开发步骤.....	65
3.5 平台变量介绍.....	65
3.6 平台 SDK 介绍.....	68
第四章 APCAD 用户界面.....	69
4.1 用户界面简介.....	69
4.2 计算机外设消息相应.....	69
4.2.1 鼠标.....	69
4.2.2 键盘.....	70
4.3 平台界面应用.....	70
4.3.1 菜单.....	70
4.3.2 工具条.....	72
4.3.3 工作区.....	73
4.3.4 状态栏.....	75
4.3.5 命令行.....	75
4.3.6 系统托盘.....	75
4.3.7 背景渐变.....	75

第一部分 准备篇

第一章 APCAD 开发简介

- 1 APCAD 基本概念
- 2 APCAD 的开发环境和开发工具
- 3 APCAD 的下载和更新
- 4 APCAD 的应用程序框架

第二章 APCAD 开发环境搭建

- 1 APCAD 开发准备工作
- 2 开发包及其工具的安装和配置
- 3 创建第一个 APCAD 项目——HelloWorld

第二部分 基础篇

第三章 APCAD 程序设计基础

1.1 APCAD 三层架构体系

2.1 平台 SDK

3.1 LUA 语言介绍

4.1 IUP 语言介绍

第四章 APCAD 用户界面

- 1 用户界面简介
- 2 事件处理
 - 2.1 鼠标
 - 2.2 键盘
- 3 平台界面应用
 - 3.1 菜单
 - 3.2 工具条
 - 3.3 工作区
 - 3.4 状态栏
 - 3.5 命令行
- 4 常用控件应用
 - 4.1 对话框
 - 4.2 编辑
 - 4.3 文本框
 - 4.4 树形
 - 4.5 列表
 - 4.6 下拉列表
 - 4.7 单选项
 - 4.8 多选项
 - 4.9 按钮
 - 4.10 图片视图
 - 4.11 进度条

第五章 APCAD 三维显示

- 1 事件处理
 - 1.1 鼠标
 - 1.2 键盘
- 2 三维对象的创建

3 三维对象的编辑

第六章 APCAD 网络与通信

- 1 用户管理
- 2 传递消息
- 3 传输文件
- 4 传递模型

第七章 APCAD 应用模块管理

- 1 APP 创建
- 2 APP 上传
- 3 APP 下载
- 4 APP 安装
- 5 APP 卸载

第八章 APCAD 云端服务

- 1 用户登录管理
- 2 软件权限验证
- 3 云加密功能
- 4 移动端系统
- 5 数据云端存储

第九章 APCAD 平台工具

- 1 字符格式转换
- 2 ADO 数据接口
- 3 COM 接口
- 4 多线程实现
- 5 Dxf 格式输出

第三部分 实战篇

第九章 快速建模

第十章 楼梯设计

第十一章 管道设计

第十二章 机床设计

第四部分 高级篇

第十三章 扩展平台显示功能

第十四章 扩展平台网络功能

.....

第五部分 附录

第十五章 APCAD 平台帮助

第十六章 LUA 帮助

第十六章 IUP 帮助

第一部分 准备篇

第一章 APCAD 开发简介

在社会上存在许多基础的开发平台，从淘宝开发平台，到腾讯开发平台，到微信开发平台，这些都提供给大家方便的利用已有资源的方法，但在工业领域中都是各自为战，还没有产生过标准版的开发平台，我们现在推出的，就是工业三维中的开发平台。

1.1 APCAD 的简介

APCAD 是一个基于三维图形环境，具有网络基本功能，应用模块即用即加，支持各类云端服务的软件开发平台。它把一般三维开发的常用功能提取出来，做成通用接口，提高了三维开发的效率，让专业人士可以专注于自己专业。

APCAD 平台有如下的特点：

✧ 网络交互

现在是互联网的时代，但在工业设计领域中主流软件还是以单机版和局域网为主，比如 AutoCAD、Revit、Microstation 等，这样在资源共享和协同工作中就存在地域的问题。而我们的平台完全以互联网为核心，直接在网络中进行软件的开发和使用，提供里资源的利用率，解决了地域限制的问题，方便于整个社会和企业资源的互补和重复利用。

✧ 简单易学

我们平台的初级、中级用户都是使用 Lua 进行开发，Lua 是门小而全的解释性脚步语言，非常简单，容易上手，它跟我们平台的 C 接口进行配合使用，利用了各自的优点，达到了相互补充，相互促进的状态。

✧ 环境通用

用户只需要学习一门简单的开发语言，就可以完成以后的所有开发要求，用户可以利用他人开发的转换接口进行数据格式的导入导出，让不同软件进行无障碍的沟通联系。

✧ 互联互通

软件模块之间可以互相访问，可以组合他人的模块，比如有做好导入 dwg，导出 dxf，设计计算的，matlab 计算库。程序应用互相可以调用，是个逐步积累的过程，发现好的 app 可以直接替换。

✧ 忠诚度高

在平台上的用户数量可能比不过腾讯 qq 和微信，但用户的忠诚度是无比高的，因为我们的平台是工业用的，它可以带给大家真正提高生产的效率，所以用户黏着度是很高的。

为什么 APCAD 平台如此吸引用户的青睐，下面我们看看 APCAD 有哪些经典的功能吸引着我们：

1、简单易学通用的三维图形环境。

基于三维图形环境，支持三维实体的绘制，显示，编辑，包括基本的选择，旋转，平移，编辑属性等，提供基本的绘制函数，用户可以绘制自己需要的一切实体。

2、提供功能强大网络功能

具有网络基本功能，提供文件传输，给某个用户发送文件，文件夹的上传下载，断点续传等基本功能给用户发送消息，发送一段代码。

3、提供即用即加的应用模块

应用模块即用即加，能够定义平台的全部界面，包含菜单，工具条，程序名称等等，支持基本的对话框设计，以及对话框中使用的控件，整个都做成 app 模块的形式，可以随时替换。

4、提供安全可靠的云端服务

支持各类云端服务，提供云存储，云计算等功能，安全备份，权限管理等功能。

1.2 APCAD 的系统架构

这一节我们了解一下 APCAD 平台主要的系统架构。我们先看一下如下图 1-1，

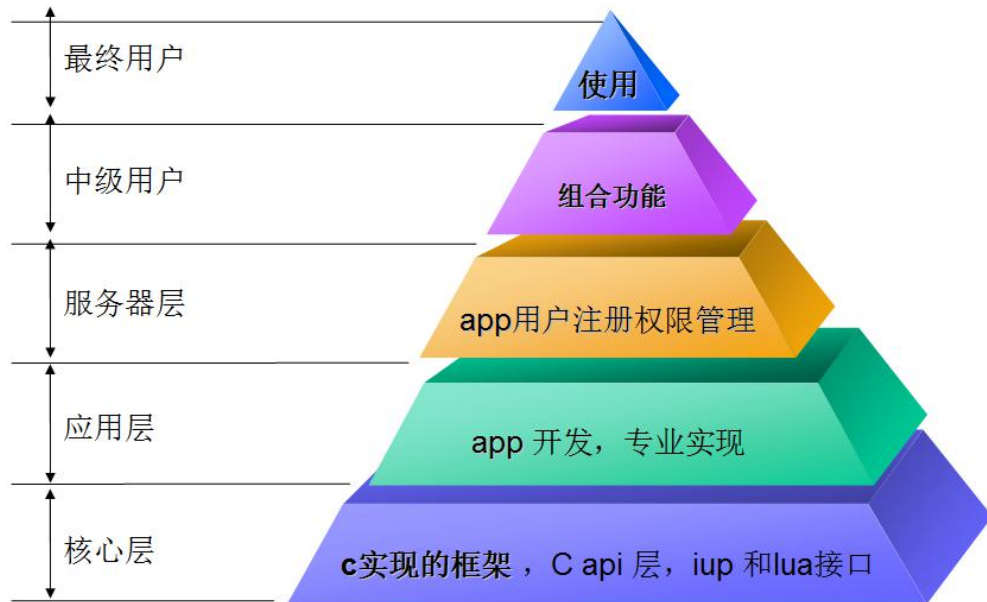


图 1-1

APCAD 平台主要分为三层，包含核心层，应用层，服务器层。下面对这三层进行简要的介绍。

1、核心层

核心层主要包含了利用 c 语言实现的程序框架，包含基本多窗口视图，菜单，工具条，命令行参数，工作区视图的操作，集成了 OpenGL、系统的键盘，鼠标的消息相应，提供了核心 C API 接口，包括 MD5 的处理，证书验证管理，com 接口，ZIP 压缩接口，ADO 数据库接口等主要功能；封装了 LUA 脚本语言和 IUP 界面设计模块，实现了基本的网络相关的操作接口。

2、应用层

应用层主要包含了用户进行开发 APP 的整个具体的过程，包含对 APP 的开发过程，打包上传的方法，对于 APP 的下载和使用，对于 APP 进行查找，搜索等。

3、服务器层

服务器层主要包含了用户进行使用者的管理，使用时间的权限管理，用户的注册、登录等网络。

1.3 APCAD 的应用层接口介绍

对于用户来说，应用层是跟我们交互最多的模块，也是主要接口和消息传递的主要层，下面我们对此层进行详细全面的介绍，让大家针对于 APCAD 有个整体的把握。

C 接口

LUA 接口

第二章 APCAD 开发环境搭建

本章主要介绍建立 APCAD 开发环境的过程，如何配置好 APCAD 的开发包和工具，以及怎么从网络中获得这些资源；其次介绍如何正确的安装和设置这些开发包，怎么使用和测试此环境；最后，我们利用 APCAD 创建第一个 APCAD 项目——HelloWorld，让用户从无到有，创建第一个完成独立的 APCAD 项目。

2.1 APCAD 的开发准备工作

配置 APCAD 之前，首先需要了解 APCAD 对于操作系统的要求，它可使用在 Window 7.0 版本以上的操作系统，对于计算机硬件的要求比较低，主流配置都是可以的。需要下载的软件以及版本如表 2-1 所示

软件名称	所用版本	下载地址
APCAD	1.0	www.apcad.com
LuaEditor	1.0	www.apcad.com
APCAD SDK	1.0	www.apcad.com

2.2 APCAD 的注册和登录



2.2.1 用户注册

1、登录网站 www.apcad.com，弹出如下界面，点击注册按钮：



The screenshot shows the APCAD website's login and registration interface. At the top, there are two buttons: 'login' with a house icon and 'register' with an upward arrow icon. The 'register' button is highlighted with a red rectangular box. Below these buttons are two input fields: '用户名称' (User Name) and '密码' (Password). At the bottom, there is a large black button with the text '确认' (Confirm).



2、点击注册（Register）按钮，弹出注册界面：

 login	 register
username:	<input type="text" value="rei"/>
passwd:	<input type="password" value="*****"/>
email:	<input type="text" value="apcad@163.com"/>
phone:	<input type="text" value="139XXXXXXXX"/>
<div>确认</div>	

3、输入本人信息，点击 即完成注册，回到登录界面进行登录即可。
APCAD 系统注册有问题时请联系我们。

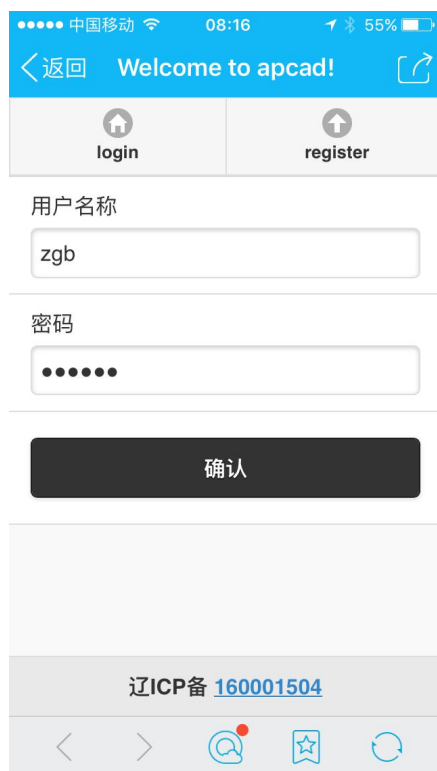
2.2.2 用户登录:

- 1、打开网址 www.apcad.com ，在电脑端和移动端都可。弹出如下界面：
电脑端：

 login	 register
用户名称	<input type="text" value="zgb"/>
密码	<input type="password" value="..."/>
<div>确认</div>	

辽ICP备 160001504

手机端：



- 2、输入用户名和密码进行登录：
- 3、登录后当软件是免费的或可试用的，在本人的界面中可看到这些软件。以手机端为例，如下所示：



2.2.3 用户购买非免费软件

如果软件是收费的，需要联系我们或者此软件的作者进行沟通和付费，平台也支持付费功能，

具体请联系我们。然后对方给予用户的使用的权限，用户可以在本人的界面下看到这些软件了。
平台付费系统如下图：



2.3.4 购买完成。

用户可以直接登录系统使用购买完的软件了

2.3 APCAD 的安装和配置

用户登录系统后，点击 APCAD 项，如图所示：



弹出如下界面，点击“下载”即可：

 home	 register
软件名称:apcad	
软件价格(人民币/每月):1000	
购买月数	<input type="text"/>
购买	
下载软件	

APCAD 软件下载完成后，在路径下直接运行“apcad.exe”即可。

2.4 创建第一个 APCAD 项目——HelloWorld

第二部分 基础篇

第三章 APCAD 程序设计基础

3.1 Lua 语言介绍

语言的优美，来自于使用者自己的感悟。Lua 的优雅，也只有使用后会明白。

3.1.1 什么是 Lua?

Lua 在葡萄牙语中意为‘月亮’，它是一门 **可扩展、可移植、简单、高效、功能强大** 的轻量级脚本语言。

- Lua 的扩展性能非常优越，因为从一开始就被设计为可扩展的，既可以用 Lua 代码来扩展，又可以用外部的 C 代码来扩展。（例如 APCAD 开发平台提供的基础 API）
- Lua 由标准 C 编写而成，几乎在所有操作系统和平台上都可以编译，运行。
- Lua 是一种简单、小巧的语言。它具有的概念不多，但每个概念都很有用。这样的简易性使 Lua 非常易于学习，同时还有利于减小 Lua 自身的大小。
- Lua 具有非常高效的实现。独立的评测结果显示 Lua 是脚本（解释型的）语言领域中最快的语言之一。
- Lua 还是一门功能非常强大的脚本语言。
 - ◆ Lua 同时支持 [面向过程](#) (procedure-oriented) 编程和 [函数式编程](#) (functional programming)；
 - ◆ 自动[内存管理](#)；
 - ◆ 只提供了一种通用类型的表（table），用它可以实现[数组](#)，哈希表，集合，对象；
 - ◆ 语言内置了模式匹配；
 - ◆ 闭包(closure)；
 - ◆ 函数也可以看做一个值；
 - ◆ 提供多线程（协同进程，并非操作系统所支持的线程）支持；
 - ◆ 通过闭包和 table 可以很方便地支持面向对象编程所需要的一些关键机制，比如[数据抽象](#)，[虚函数](#)，[继承](#)和[重载](#)等。

3.1.2 Lua 编译运行

APCAD 平台中内嵌了 Lua 语言解释器，我们可以直接在程序中使用 Lua 语言进行编写我们的软件，在此过程中主要是使用 Lua 语言进行开发。我们的第一个 APCAD 的 Lua 程序是从程序路径下“main.lua”开始的。

3.1.3 Lua 基本语法简介

想要学好 Lua，我们应该做到熟知 Lua 语言本身相关的语法规则，要了解它的词法规范、变量、所具有的值类型、各种表达式、语句、函数等等基础知识。下面仅列出部分规则仅供了解参考：（具体还请参考《Lua 程序设计第二版》）

- Lua 是区分大小写的。变量 A 与变量 a 是不同。
- C 语言中一条语句需要加上分号作为语句结束的标志，在 Lua 中是可选的。
- Lua 的标识符可以是任意字母、数字、下划线构成的字符串，但不能以数字开头。（应该避免使用以一个下划线开头并跟着一个或者多个大写字母（例如 ‘_VERSION’）的标识符，Lua 将这类标识符保留用作特殊用途）。
- Lua 的全局变量不需要声明。
- 删除一个变量仅需把他赋值为 nil 即可。例如 arg = nil（这样变量 arg 就被删除了）。
- Lua 的局部变量使用 local 关键字定义。
- Lua 的注释分单行注释和多行注释，单行注释使用 ‘--’，多行注释使用：“--[[”表示注释开始，并且一直延续到“]]”为止。

3.2 IUP 介绍

3.2.1 什么是 IUP?

IUP 是一个用于建立图形用户界面的多平台工具包。它提供了基于三种基础语言（C、Lua、Led）的简单 API，因此我们可以在 Lua 代码中调用它提供的 API 界面程序设计。

3.2.2 为什么使用 IUP?

- **跨平台**
IUP 的目标是允许程序的代码不需要经过任何修改就能在不同的系统上运行。支持的系统包括：GTK+, Motif and Windows
- **高性能**
IUP 的高性能源于其使用的是本地接口元素（可以理解为不自绘，使用系统自带的接口元素）。
- **很简单**
源自于简单的 API，用户能够快速的学习并使用。
- **够丰富**
IUP 目前提供了丰富的界面资源库，足以满足我们日常开发的需求。例如：菜单、按钮、文本编辑、树、列表等等。

3.2.3 如何学习 IUP?

学习任何一门语言或者工具，一定要懂得查阅相关资料，IUP 的在线文档地址是 <http://webserver2.tecgraf.puc-rio.br/iup/>。建议大家在学习 IUP 时做到边看边做，通过动手实验来查看控件中的各个属性对于该控件的影响。（在本文档中仅对一些概念进行简要说明，便于了解，详细的还请查阅 IUP 在线帮助文档）

在 IUP 中有三个概念需要明白：元素、属性、回调。

元素是应用程序中各种接口元素。

属性是被用于修改或者获得元素的属性。

回调是事件被触发时响应的函数。

下面让我们开始了解 IUP 中一些常用的控件以及相关的属性，同时我们会给出部分示例帮助您更好的学习 IUP。

3.3 IUP 常用控件简介

3.3.1 对话框 (DIALOGS)

在 IUP 中，对话框是一个特殊的元素，它是一个用来显示其他控件的元素。在本小节中将会介绍如何创建一个简单的对话框（IUP 中也提供了一些预定义好的对话框元素，我们会在下面的章节中对常用的几种进行单独的介绍）。

需要注意的是要创建自己的对话框，对话框中要包含的所有控件都必须在对话框创建之前完成创建。

创建一个对话框需要考虑：

- 1、对话框中包含的控件样式及属性：LABEL（标签控件）、TEXT(文本控件)、BUTTON（按钮控件）、LIST(列表控件)等等。
- 2、对话框的容器： iup.hbox{}（水平排版容器）、iup.vbox{}（垂直排版容器）等等；
- 3、对话框自身的属性：前景色、背景色、尺寸（像素尺寸还是字符尺寸）、标题、菜单等等。
- 4、对话框的弹出样式：模态对话框（对话框不关闭程序不继续执行），非模态对话框（对话框弹出后，程序依然运行）。

1) 创建

创建一个对话框的界面对象。

使用：

```
dlg= iup.dialog{}
```

-- 创建了一个对话框界面对象。{} 中可以设置对话框的属性，也可以设置对话框中要显示的其他控件或者容器。需要注意的是对话框控件中只能有一个控件或者容器作为参数，所以如果想要使对话框包含多个控件，那么必须使用容器。

简单示例：（显示 ‘Hello IUP’ 为例）

```
package.cpath = "?53.dll;" .. package.cpath
```

```
--检查所需要的库文件是否存在
```

```
--require 后的模块名替换‘?’，根据替换的结果在设置的目录下或者环境中查找文件是否存在。
```

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
lab = iup.label{
```

```
-- 对话框本身不能在其控件内部显示文本，只能借助于其他控件。这里使用 label 控件显示文本。
```

```
title = 'Hello IUP', -- label 控件的显示文本
```

```
expand = 'YES', -- 设置自动扩展
```

```
fontsize = 50, -- 设置文本字体的大小
```

```
alignment = 'ACENTER:ACENTER'
```

```
-- 设置文本相对于控件的位置，这里 ‘:’ 号前是水平方向 ‘:’ 号后是垂直方向；
```

```
}
```

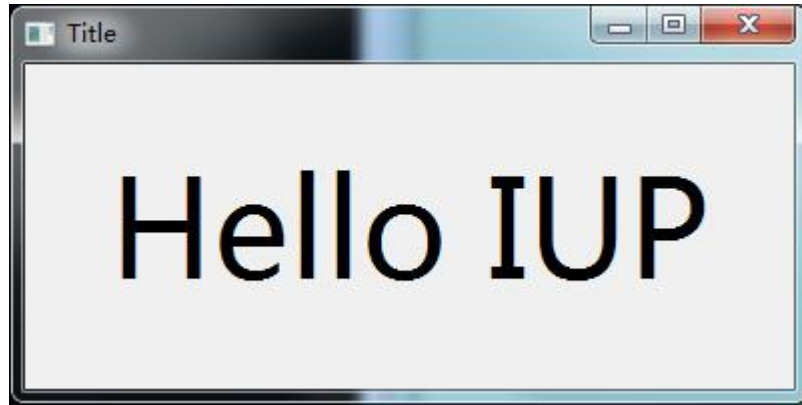
```
local dlg = iup.dialog{ --创建 dialog,
```

```
iup.vbox{ --纵向排版容器
```

```
lab; -- 将创建好的标签对象放入容器中。
```

```
};
    rastersize = '400x200'
}
dlg.title = 'Title' --设置对话框标题
dlg.map() --绘制对话框，绘制对话框时会将对对话框中的所有控件都进行绘制
dlg.popup() --弹出对话框。
```

显示效果：



2) 属性：TITLE （标题）

设置对话框的标题；默认值是 NULL。标题显示位置处于对话框的左上角；对话框的标题一般用于简要说明对话框的功能。

使用：

```
dlg.title = 'Dialog'
```

3) 属性：RASTERSIZE（大小）

对话框的大小（以像素为基本单位）。

使用：

```
dlg.rastersize = "400x200"--宽度设置为 400 高度设置为 200.
```

注意：

值样式为 "widthxheight"。width 和 height 分别对应水平和垂直方向；

值 'x200' 等同于 '0x200'，'100x' 等同于 '100x0'。需要注意的是宽或者高为 0 时不代表没有宽或者高，而是代表了自适应对话框中容器或者控件的大小。

4) 属性：DEFAULTENTER（默认回车键行为）

当焦点在其他界面控件上时，键盘上的回车键在被敲击后触发的默认按钮的行为。设置快捷键更方便。

使用：

```
dlg.defaultenter = button
```

--button 是一个已经创建出的按钮控件对象。具体用法请参考下面示例。

注意：

值仅接收 button 控件对象，其他控件没有意义。

控件自有的回车行为会屏蔽掉此项属性。比如，当焦点处于其他任何一个 button 控件时，敲击回车键都会执行焦点所在的 button 控件，而不是设置的默认 button 控件。又比如当前焦点处于一个多行的文本上时，回车的行为是换行而不是触发设置的 button。

5) 属性：DEFAULTESC（默认 Esc 键行为）

当焦点在其他控件时，键盘上的 Esc 键在敲击后触发的默认事件。

使用：

```
dlg.defaultesc= button2
```

--button2 是一个已经创建出的按钮控件对象。具体用法请参考下面示例：

注意:

使用方式与 defaultenter 相似。

6) 属性: MENU (菜单)

接收一个 iup.menu 创建的菜单控件对象作为对话框的菜单。

使用:

```
dlg.menu = menu
```

--值是创建好的 menu 控件对象。具体菜单请看下面的 menu 介绍。具体用法请参考下面示例;

7) 属性: RESIZE (可变大小)

允许交互 (交互可以理解为在对话框弹出后, 用户操作对话框) 的改变对话框的大小, 默认值是 "yes" (允许改变对话框大小);

使用:

```
dlg.resize = "no" --设置不允许交互改变对话框大小。
```

注意:

仅控件绘制之前设置有效, map (绘制) 后赋值无效。

如果设置的值为 'no', 对话框中的 maxbox 属性值会被设置为 'no' (不能放大)。

8) 属性: STARTFOCUS (初始焦点)

设置对话框弹出后, 获得焦点的控件 (可以理解为被鼠标选中的控件)。

使用:

```
dlg.startfocus = txt;
```

--txt 是 text 控件创建出的对象。设置获得焦点的控件是一个 text 控件。那么在对话框弹出后敲击键盘上的字符会直接显示在 text 控件中 (如果不是焦点所在的控件, 需要先鼠标点击确定焦点才能输入字符)。

注意:

如果没有设置这个属性, 对话框默认将第一个可以获得焦点的控件处于被选中状态。

在对话框显示之后获得焦点。

9) 属性: TOPMOST (顶层)

设置对话框总是放在其他所有对话框之前显示。默认值是 "no";

使用:

```
dlg.topmost = "yes" --设置当前 dlg 显示在其他对话框之前。
```

10) 回调: CLOSE_CB (对话框关闭)

设置对话框关闭 (或者 hide) 之前触发的回调函数; ih:close_cb()

使用方式:

在 dialog 创建时:

```
dlg = iup.dialog{close_cb = function () ...end}
```

或者在对话框之外:

```
dlg.close_cb = function () ...end
```

或者

```
function dlg:close_cb()
```

```
...
```

```
end
```

注意:

类似于 IUP_IGNORE 这种写法的值, 在 Lua 使用时要转换为 iup.IGNORE

如果设置回调的返回值是 iup.IGNORE (IGNORE 意思是忽略), 这个返回值会阻止对话框关闭。

11) 回调: K_ANY (键盘按键响应)

响应键盘上的按键处理。ih:k_any(number)

键盘上的每一个按键都有对应的数值，可以通过参数 number 来确定具体的按下的是什么键。

```
function dlg:k_any(num)
```

```
...
```

```
end
```

注意：

键盘上的任意键都对应着唯一的数字标识。

k_any 的参数 num（形参的名可自定义）会根据按键反馈给你一个数字。

12) 示例：

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
-----  
txt = iup.text{--使用 text 元素创建了一个可用于文本编辑的控件（详细请看 text）
```

```
    rastersize = '300x100',
```

```
    expand = 'yes',
```

```
    multiline = 'YES',
```

```
    wordwrap = 'yes'
```

```
}
```

```
    btn_ok_ = iup.button{rastersize = '100x',title = 'Ok',action = function ()  
print('Click ok button !') dlg:hide() end}
```

----使用 button 元素创建了一个可用于点击按钮的控件（详细请看 BUTTON）

```
    btn_cancel_ = iup.button{rastersize = '100x',title = 'Cancel',action = function  
() print('Click cancel button !') dlg:hide() end}
```

--（详细请看 BUTTON）

```
local function deal_open()
```

local filedlg = iup.filedlg{DIALOGTYPE = "OPEN"} -- filedlg 是 IUP 预定义好的文件选择对话框

```
filedlg:popup() -- 显示文件选择对话框的界面
```

```
if filedlg.value then -- 判断是否选择了文件
```

txt.append = '打开的文件全路径: ' .. filedlg.value -- 向文本编辑框中追加内容。这里是追加了选择文本的信息

```
end
```

```
end
```

```
menu_ = iup.menu{ -- 使用 menu 元素，创建了一个菜单。
```

iup.item{title = 'Open File',action = deal_open};--使用 item 元素，作为 menu 中的一项。点击'Open File'菜单时会调用 deal_open 函数。

```
iup.submenu{ -- 使用 submenu 创建一个子菜单控件。
```

```
    title = 'Open Submenu', --子菜单的名称
```

```
    iup.menu{
```

```

        iup.item{
            title = 'Submenu',
            action = function () iup.Message('Notice','你点击了 'Submenu' !
') end

            -- 点击'Submenu' 菜单时会调用 action 函数。
        };
    }
}

}

-----

dlg = iup.dialog{
    --使用 dialog 元素，创建一个自定义的对话框
    --注意对话框中只能有一个控件或者容器作为对话框的参数使用。比如下面使用的 VBox
元素。
    iup.vbox{
        --使用 vbox 元素用来垂直排版容器中包含的控件。
        txt; -- 使用创建好的文本控件。
        iup.hbox{btn_ok_, btn_cancel_}, --使用 hbox 元素用来水平排版容器中包含的
控件。

        alignment = 'ARIGHT'; --容器中控件的对齐方式是向右对齐。
        margin = '10x10'; --边距
    };
    rastersize = '400x300'; --对话框大小。
}

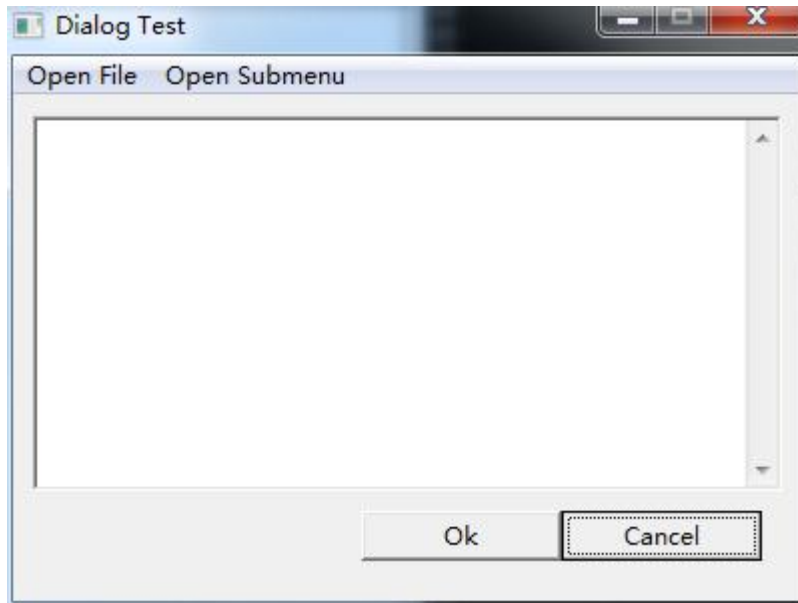
function dlg:close_cb() --close_cb 回调函数
    print('Close the dialog !')
    --return iup.IGNORE -- 如果没有注释掉此行，则点击对话框的 X 按钮，对话框被
阻止关闭。
end

function dlg:k_any(num) --k_any 回调函数
    print(num) -- 查看按下不同的键的结果
end

dlg.title = 'Dialog Test' --设置对话框标题,
dlg.DEFAULTENTER = btn_ok_ --设置默认 enter 键触发的按钮
dlg.DEFAULTESC = btn_cancel_ --设置默认 Esc 键触发的按钮
dlg.startfocus = btn_cancel_ --设置初始的焦点控件
dlg.menu = menu_ --为对话框添加菜单
dlg.resize = 'no' --设置对话框弹出后不允许交互时改变对话框大小。
dlg:map() --绘制对话框
dlg:popup() --弹出对话框。

```

13) 效果:



3.3.2 文件选择对话框（FILEDLG）

FILEDLG 是一个 IUP 预定义好的对话框元素，需要注意的是此对话框的属性设置必须在显示（popup）之前。

1) 创建

创建一个文件选择对话框元素。它是一个用来选择文件或者一个目录的预定义对话框。对话框想要显示只能通过 `iup.popup()` 显示。

使用:

```
filedlg = iup.filedlg{} -- {} 内可以设置 filedlg 的属性  
filedlg.popup() -- 将对话框弹出。  
local val = filedlg.value -- value 属性用来获取对话框的选择结果。
```

注意:

对话框显示必须是模态的。

对话框可以接收 '/' 作为路径分隔符，但是值的属性返回的字符串总是使用 '\\' 作为路径分隔符。

2) 属性：DIALOGTYPE（对话框类型）

FILEDLG 对话框的类型；可能的值有 "OPEN"、"SAVE"、"DIR"；默认值是 "OPEN"。

使用:

```
filedlg.DIALOGTYPE = "OPEN"
```

注意:

OPEN: 选择文件对话框。

SAVE: 保存文件对话框。

DIR: 路径选择对话框。

不同的类型，对话框的界面样式也是有区别的。

3) 属性：FILE（文件）

初始化显示在 '文件名' 对应区域的文件的名字（如下图）；

使用:

```
filedlg.file= 'E:/ProjectCode/DEV_IUP53/dlgTestFileDialog.lua'
```

效果：



注意：

如果 file 属性值是一个文件的全路径，控件的初始目录和 DIRECTORY 属性会被忽略。

4) 属性：FILTER（过滤）

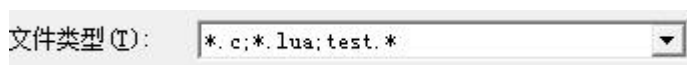
筛选出符合类型的文件。接收一个字符串，字符串中不能有空格，且每个类型之间用';'号隔开。

使用：

```
filedlg.filter = '*.c;*.lua;test.*'
```

--筛选出来显示的结果就是所有 c 文件、lua 文件和以 test 为名的文件。

效果：



5) 属性：MULTIPLEFILES（多选）

当 DIALOGTYPE 的属性值是“OPEN”时，允许用户选择多个文件。可选值是“yes”、“no”；默认值是“no”；

使用：

```
filedlg.MULTIPLEFILES = 'yes'
```

注意：

必须在同一个目录下多选。

只有在 DIALOGTYPE 的属性值是“OPEN”时设置有效。

6) 属性：TITLE（标题）

设置对话框标题显示的名称（对话框的名字）。

使用：

```
filedlg.title = 'FileDialog'
```

7) 属性：VALUE（选择的文件）

选择文件的名称（可以多个），其值是一个包含路径和文件名的字符串。如果没有选择文件则返回一个 nil。

使用：

```
local val = filedlg.value
```

-- 通常用一个变量接收 value 属性的值

```
print(val)
```

--（为了方便可以将只读的属性直接使用 Lua 提供的 print 函数打印查看结果）。

当选择一个文件时的打印结果：

```
E:\ProjectCode\DEV_IUP53\dlgTestFileDialog.lua
```

当选择多个文件时的打印结果：

```
E:\ProjectCode\DEV_IUP53|cd.d11|cdg1.d11|cdim.d11|cdlua53.d11|（注意：蓝色部分是路径文件夹，最后一个文件夹末尾没有'\', 文件使用'|'分隔）。
```

注意：

通常使用在 filedlg:popup() 之后。

该属性是只读的。

8) 示例:

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
filedlg = iup.filedlg{} --使用 filedlg 元素创建文件选择对话框界面对象。
```

```
filedlg.DIALOGTYPE = 'open'
```

--对话框的样式是打开，因为默认值是'open'，是可以省略的。

```
filedlg.title = 'FileDialog' --设置对话框的名字为'FileDialog'
```

```
filedlg.MULTIPLEFILES= 'yes' --设置可以多选文件
```

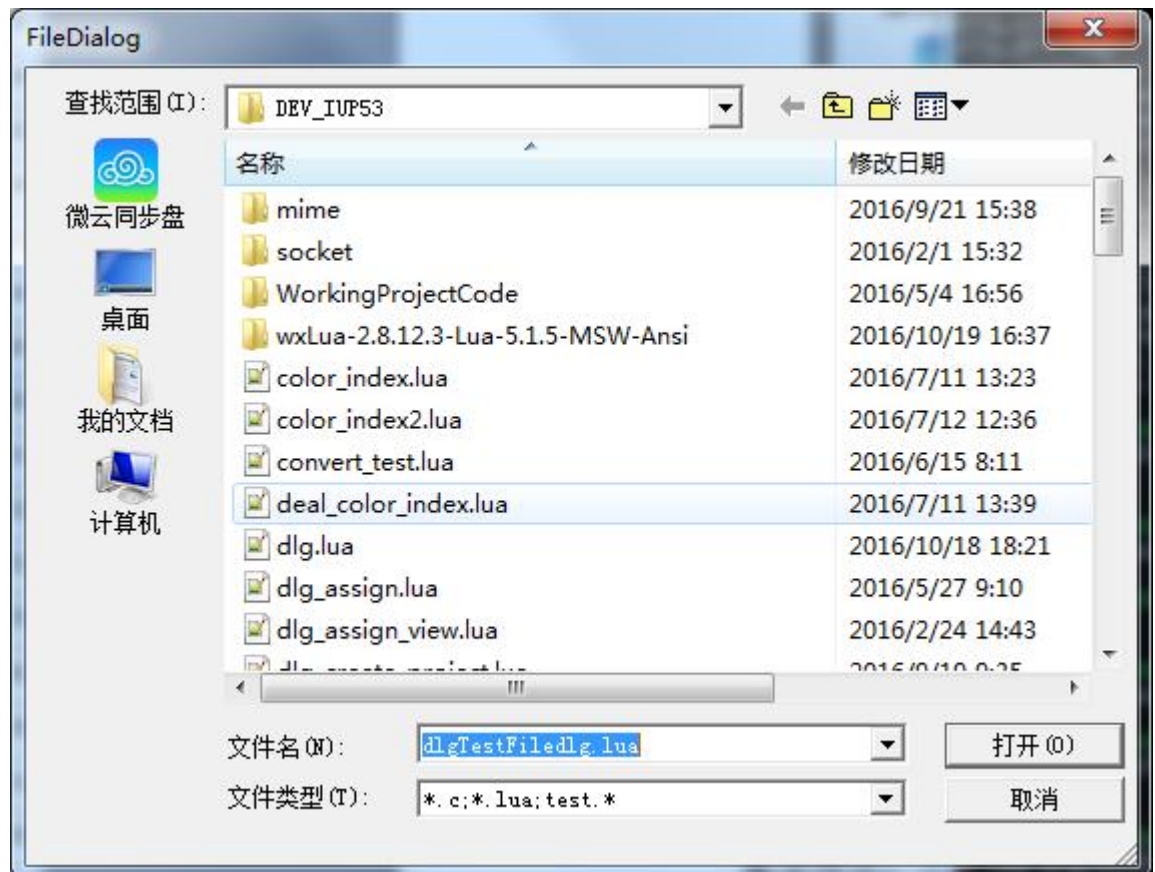
```
filedlg.file = 'E:/ProjectCode/DEV_IUP53/dlgTestFileDialog.lua'
```

--设置初始的文件名。

```
filedlg.FILTER = '*.c;*.lua;test.*' --过滤文件，只查找所有的C文件、Lua文件以及以test为名的所有不同类型的文件。
```

```
filedlg:popup() --显示对话框。
```

9) 效果:



3.3.3 消息对话框 (MESSAGE)

MESSAGE 是一个 IUP 预定义好的对话框元素,此控件不需要像 filedlg 那样需要 popup 才能显示,它在创建成功后运行到它时就会显示。

1) 创建

使用 `message` 元素创建一个消息对话框界面。通常用来提示用户。

使用：

```
iup.Message("title","This is message ! ")
-- 使用时 Message 的首字母 M 一定要大写，其他字母小写。
-- 第一个参数是消息框的标题。
-- 第二个参数是消息框的提示内容。
```

注意：

它是弹出时是模态的对话框。

参数过少，或者参数类型不正确会引发错误。参数的类型可以是数字或者字符串。

2) 示例

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
iup.Message('Message','This is a Message tips !') --消息提示。
```

3) 效果



3.3.4 提示对话框（ALARM）

ALARM 是一个 IUP 预定义好的对话框元素，此控件同样不需要像 `filedlg` 那样需要 `popup` 才能显示，它在创建成功后运行到它时就会显示。

1) 创建

使用 `alarm` 元素创建一个包含一段消息的模态对话框对象。与 MESSAGE 不同的是它可以有三个按钮。并且有返回值。根据返回值确定点击的按钮操作。

使用：

```
val = iup.Alarm('Title','Alarm Message','Button1','Button2','Button3')
-- 使用时 Alarm 的首字母 A 一定要大写，其他字母小写。
-- 第一个参数是消息框的标题。
-- 第二个参数是消息框的提示内容。
-- 后面的参数为按钮的标题（可选，最多 3 个，最少 1 个）。
```

--val 接收到的返回值如果是 1 代表第一个按钮被按下，依次类推。如果点击的是关闭按钮则返回的是 0。

注意：

它是弹出时是模态的对话框。

参数过少，或者参数类型不正确会引发错误。参数的类型可以是数字或者字符串。

此对话框中默认设置了"DEFAULTENTER" and "DEFAULTESC"属性, "DEFAULTENTER" 被设置在第一个按钮。"DEFAULTESC"被设置在最后一个按钮。

2) 示例

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?', 根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
local val = iup.Alarm('Message', 'This is a Alarm tips !', 'ok', 'cancel', 'help')
```

```
print(val) --根据返回的数字确定点击的按钮
```

3) 效果



3.3.5 垂直排版容器 (VBOX)

VBOX 是一个容器元素, 它可以在垂直方向上排版容器内的其他控件或者容器。

1) 创建

使用 vbox 元素创建一个空容器用来从上到下的排版它所包含的控件对象。

使用:

```
vbox = iup.vbox{}
```

-- {}中以数组的形式存放控件对象。也可以设置vbox的属性

2) 属性: ALIGNMENT (对齐)

设置容器中水平方向上对齐方式。可接受的值: "ALEFT", "ACENTER", "ARIGHT", 默认的值是"ALEFT".

使用:

```
vbox.alignment = "ARIGHT"
```

3) 属性: MARGIN (边距)

定义边缘间距。值的样式为: "widthxheight" (设置水平方向上和垂直方向上的控件与容器的边缘的距离, 以对齐设定的边为准)。默认值是'0x0' (没有边距)。

使用:

```
vbox.margin= "10x5"
```

--设置水平方向的间距是 10, 垂直方向的间距是 5。

4) 属性: GAP (间距)

定设容器内各个子控件在垂直方向上的间距。默认值是'0' (没有间距)。

使用:

```
vbox.gap= "10"
```

5) 示例:

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?', 根据替换的结果在设置的目录下或者环境中查找文件是否

在。

`require 'iuplua'` --加载需要的IUP基本函数库模块

`local dlg = iup.dialog{` --使用dialog元素创建对话框

`iup.vbox{` --使用vbox元素，垂直排版容器内的控件。

`iup.button{rastersize = '100x', title = 'Button1'};` -- 使用button元素创建了一个名为Button1 的按钮。

`iup.button{rastersize = '200x40', title = 'Button2'};`

`iup.button{rastersize = '300x100', title = 'Button3'};`

`alignment = 'ARIGHT';` --设置容器内控件对齐方式是右对齐。

`margin = '30x20';` --设置容器与控件的间距 水平方向是 30 ， 垂直方向 20

`gap = 10;` --设置容器中控件之间垂直方向上的间距。

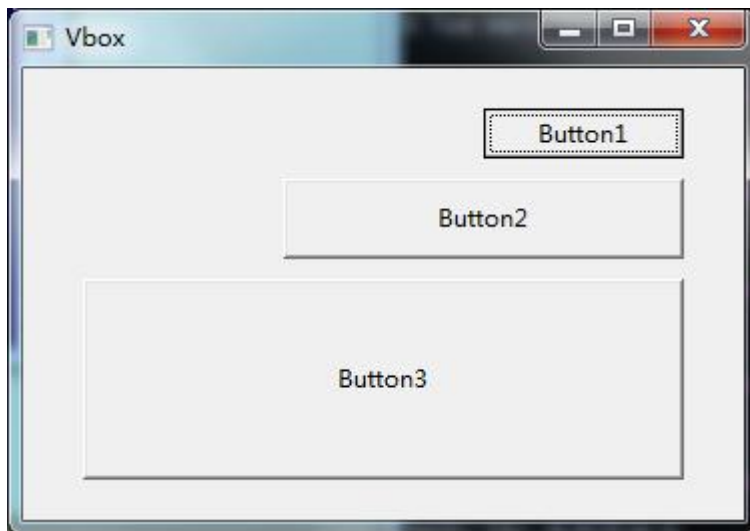
`};`

`title = 'Vbox';` --对话框的标题

`}`

`dlg:popup()` -- 显示对话框

6) 效果:



3.3.6 水平排版容器 (HBOX)

HBOX 是一个容器元素，它可以在水平方向上排版容器内的其他控件或者容器。

1) 创建

创建一个空容器用来从左到右的排版它所包含的控件对象。

使用:

`hbox = iup.hbox{`

`-- {}中以数组的形式存放控件对象。也可以设置hbox的属性`

注意:

此容器创建时可以没有对象在内，使用IupAppend or IupInsert进行填充。

使用方式与VBOX类似。不同的地方在于一个水平方向排版一个垂直方向排版。

2) 属性: ALIGNMENT (对齐)

设置容器中水平方向上对齐方式。可接受的值: "ATOP", "ACENTER", "ABOTTOM", 默认的值是"ATOP"。

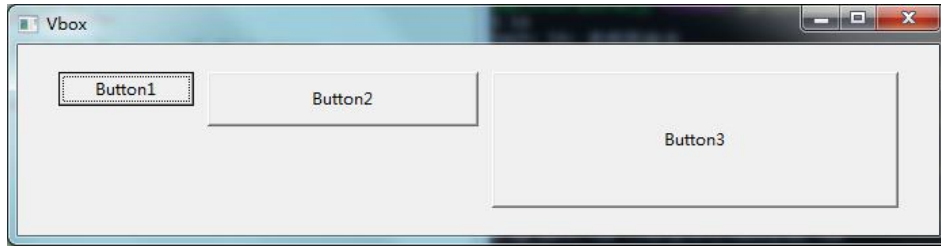
使用:

`vbox.alignment = "ABOTTOM"` --设置vbox的对齐样式为右对齐。

3) 示例:

将VBOX的示例中vbox改为hbox, 将alignment的值改为'ATOP' 试试。

4) 效果:



3.3.7 文本编辑 (TEXT)

1) 创建

TEXT元素可以创建一个可编辑的文本框。

使用:

```
txt= iup.text{}  
-- {} 中可以设置 text 元素的属性。
```

简单示例:

```
package.cpath = "?53.dll;" .. package.cpath  
--检查所需要的库文件是否存在  
--require 后的模块名替换'?', 根据替换的结果在设置的目录下或者环境中查找文件  
是否存在。  
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
txt= iup.text{value = 'hello',rastersize = '400x'}  
-- 使用 text 元素创建一个包含默认文本 'hello', 大小是'400x'的文本编辑框  
dlg = iup.dialog{txt, title = 'Text'}  
dlg:popup() --显示对话框
```

显示效果:



注意: 下面属性中的显示效果是对简单示例增加修改所得, 仅用来简单了解各个属性对控件的影响。

2) 属性: ALIGNMENT (文本对齐)

设置文本相对于文本编辑框的对齐方式。可接受的值: "ALEFT" (左对齐), "ARIGHT" (右对齐), "ACENTER" (中心), 默认的值是"ALEFT"。

使用:

```
txt.alignment= "acenter" -- 设置对齐方式是向右对齐。
```

显示效果:



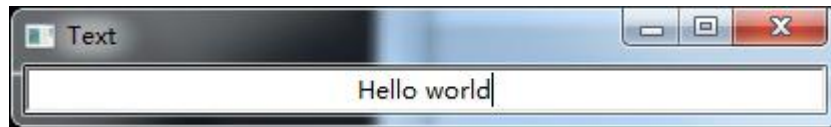
3) 属性: APPEND (追加内容)

在当前文本的末尾追加一段字符串;

使用:

```
txt.append = " world"
```

显示效果:



注意:

只写属性, map (绘制) 前赋值无效。

当 `multiline="yes"` 并且 `wordwrap='yes'` (这两个属性下文会有介绍) 时, 如果当前的文本编辑框中有内容, 则在追加文本之前, 一个换行符 '`\n`' 会被先插入, 其结果就是追加的文本会在新的行中显示。

4) 属性: BGCOLOR (背景色)

设置控件的背景色, 默认使用的是全局属性 `TXTBGCOLOR`。

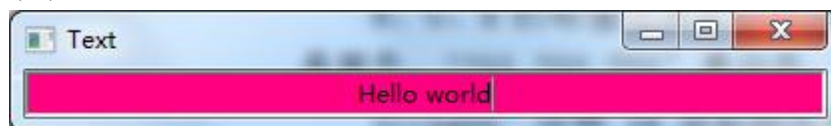
使用:

```
txt.bgcolor= '255 0 128'
```

```
-- (red=255 blue=128 green=0)
```

```
-- 值以空格隔开的字符串, 第一段数字代表红色, 第二段代表绿色, 第三段代表蓝色。
```

显示效果:



注意:

R、G、B 的取值范围都应该是从 0 ~ 255 之间的数 (包含 0 和 255); 比如: "0 0 0" 是黑色, "255 255 255" 是白色, "255 0 0" 是红色等等。

`#rrggbb` 这种 16 进制颜色表示法也可以使用。比如 `txt.bgcolor= "#FF0080"` 等同于 `txt.bgcolor='255 0 128';`

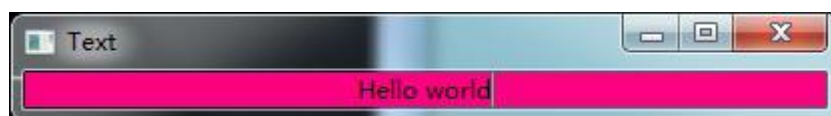
5) 属性: BORDER (边框)

在 text 控件四周设置边框。默认值是 "yes" (有边框);

使用:

```
txt.border = 'no'; --设置成没有边框
```

显示效果:



注意:

仅控件绘制之前设置有效, map (绘制) 后赋值无效

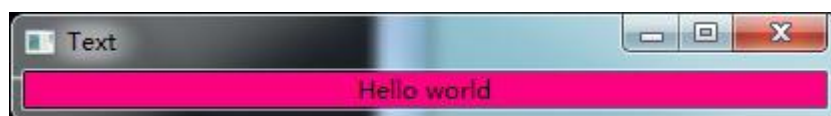
6) 属性: CANFOCUS (焦点)

控件是否允许成为关注点, 默认值是 'yes' (允许)。

使用:

```
txt.canfocus= 'no'
```

显示效果: (焦点没在文本编辑框上, 所以没有光标闪烁)。



注意:

仅控件绘制之前设置有效, map (绘制) 后赋值无效。

在 windows 下, text 控件在被鼠标点击时会自动获得焦点。

7) **属性: COUNT (字符数)**

返回文本编辑框中字符数目, 包括换行符;

使用:

```
local count = txt.count  
print(count) -- 用 print 函数输出查看结果。
```

注意:

只读, map(绘制后有效, 绘制前使用返回值为 nil)

没有任何文本返回值为 0;

如果返回值不是 0, 但是又没有文本显示时, 查看是否编辑框内包含多个空格字符。

8) **属性: FGColor (前景色)**

设置文本控件对象的前景色。通常它是相关联的文本的颜色。(需要注意的是并不是所有控件的前景色都是指文本)

使用:

`txt.fgcolor = "0 0 255"` --蓝色。也可以使用 16 进制的颜色属性值, 请看 [BGColor](#)。

显示效果:



9) **属性: FILTER (过滤)**

过滤用户输入的字符。值可以是 "LOWERCASE" (字符会被转换成小写字母显示), "UPPERCASE" (字符会被转换成大写字母显示) or "NUMBER" (只有数字允许输入显示)。

使用:

```
txt.filter= "number"
```

注意:

filter 值为 "LOWERCASE" 与 "UPPERCASE" 时, 标点符号或者中文字符等 (非 26 个英文字母) 的字符都会被正常显示, 不会发生任何改变。

10) **属性: MULTILINE (多行)**

允许文本多行编辑。默认值是 "no" (不允许)。

使用:

```
txt.multiline="yes"
```

注意:

仅控件绘制之前设置有效, map (绘制) 后赋值无效。

单行模式下一些字符是无效的。比如: "\t", "\r" and "\n"

11) **属性: PASSWORD (密码)**

用字符 '*' 来隐藏输入的字符。默认值是 "no"。

使用:

```
txt.password = "yes"
```

显示效果:



注意:

仅控件绘制之前设置有效, map (绘制) 后赋值无效。

12) **属性: READONLY (只读)**

设置文本编辑框控件为只读属性。默认值是 "no" (允许编辑)。

使用:

```
txt.readonly = "yes" -- 不允许编辑
```

注意:

控件设置为只读属性,则不在允许用户在界面操作中输入内容,但是通过 `value` 属性可以实现更改控件中的显示的内容。

13) 属性: SELECTEDTEXT (选择文本)

获得选中的文本内容。没有选择的文本返回 `nil` 值。

使用:

```
local val = txt.selectedtext  
print(val) -- 查看选中的文本内容
```

注意:

此属性为只读属性。

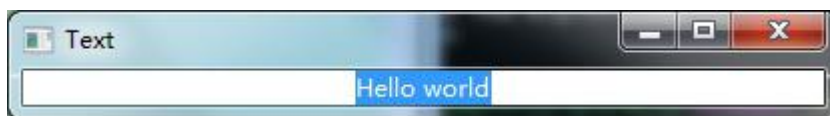
14) 属性: SELECTIONPOS (选中字符)

设置文本字符的选中效果。如下图。可接收的值有: "ALL" (所有)、"NONE" (不选)、"pos1:pos2" (按照字符处于控件中的位置选择。字符的位置从 0 开始)

使用:

```
txt.selectionpos = 'all'  
等同于  
txt.selectionpos = "0:".. #txt.value。
```

显示效果:



注意:

界面绘制之后设置有效(绘制之前无效);

此属性要设置在其他可以改变文本内容属性(比如 APPEND 属性)之后。

文本控件中的选择的属性还有 SELECTION 属性,感兴趣的可以看看。

15) 属性: VALUE (文本值)

设置或者获得控件中的文本内容。

使用:

```
txt.value = 'Hello' --设置文本中的内容  
print(txt.value)
```

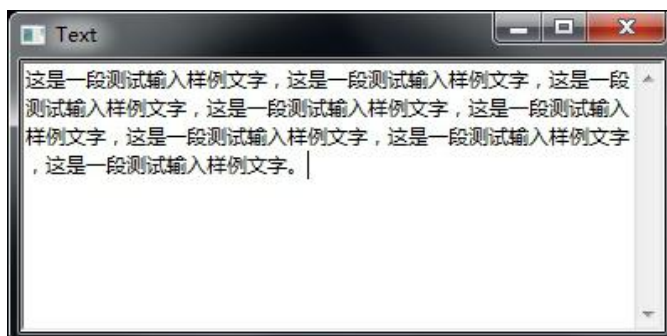
16) 属性: WORDWRAP (自动换行)

设置自动换行。默认值是 "no";

使用:

```
txt.wordwrap = 'yes'
```

显示效果: (自动换行效果)



注意：

仅控件绘制之前设置有效，map（绘制）后赋值无效。

只有 MULTILINE=' YES' 时，设置此属性才有意义；

水平滚动条将会被移除，只保留垂直方向的滚动条。

需要注意的是如果字符当中有换行符，那么换行的位置处于换行符的位置

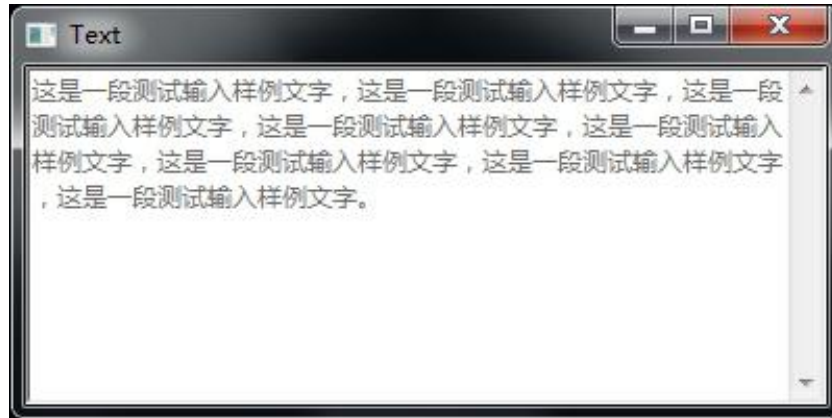
17) 属性：ACTIVE（激活）

控制控件激活的状态；默认值是“yes”（激活状态，是指在界面状态下用户可以操作控件）；

使用：

`txt.active= 'no'` -- 在界面状态下用户无法操作此控件

显示效果：



18) 属性：FONT（字体）

设置控件中文本的字体；

使用：

`txt.font = "Times, Bold 18"`

-- 逗号前是字体名，逗号后是字体的状态或者大小（状态可以有多个均以空格分隔）

注意：

字体可以整体设置也可以单独设置。

比如：

`txt.font = "Times, Bold Italic 18"`

等价于

`txt.fontsize = '18'`

`txt.fontstyle = 'Bold Italic'`

`txt.fontface = 'Times';`

显示效果：（以输出的 hello world 为例）



19) 属性：EXPAND（扩展）

允许文本控件自动扩展到容器剩余位置；默认值是“no”，对于容器的控件默认值是“yes”。

文本控件不是容器，所以它的默认值是“no”；

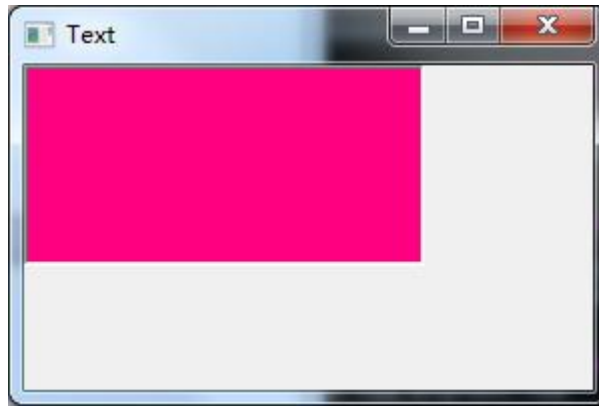
使用：

`txt.expand= "horizontal"`

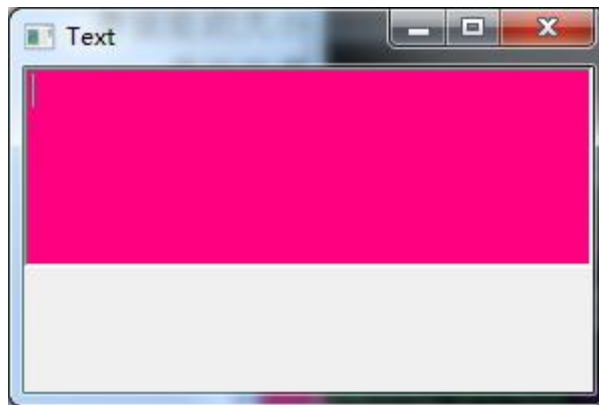
--向水平方向上扩展。

显示效果：

未设置 expand 时：（有颜色的是控件的大小）



expand= "horizontal" 时（控件在水平方向上填充了容器剩余空白位置）



注意：

expand 的值可以是："YES" (both directions)（上下左右都自动扩展），"HORIZONTAL"（水平方向扩展），"VERTICAL"（垂直方向扩展），"HORIZONTALFREE"（水平方向自由扩展），"VERTICALFREE"（垂直方向自由扩展）or "NO"（不扩展）。

20) 属性：RASTERSIZE（大小）

设置控件的大小（以像素为基本单位）。

使用：

```
txt.rastersize = "100x20" --设置控件的大小是宽度为100，高度为20；
```

21) 示例：

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
txt = iup.text{rastersize = '200x100';}
```

--创建一个文本编辑控件，同时设置该控件的大小是宽 200，高 100。

```
txt.BGCOLOR = '0 0 255' --控件的背景色设置成蓝色
```

```
txt.fgcolor = '255 0 0' --设置控件的前景色为红色。
```

```
txt.value = 'Hello' --设置默认的值 of 'hello'
```

```
txt.MULTILINE='YES' --设置该文本编辑框是多行的。
```

```
txt.wordwrap = 'yes' --设置自动换行
```

```
txt.fontstyle = "Strikeout" --设置字体的类型是删除线
```

```
txt.alignment = 'aleft' --设置文本对齐方式是向左对齐
```

```
txt.expand= "yes" --设置文本编辑框扩展填充容器空余位置。
```



```
txt.rastertime = '300x200'; --重新设定文本编辑框的大小。
```

```
local dlg = iup.dialog{ --创建对话框  
    txt; --文本控件对象  
    title = 'Text';--对话框标题  
    rastertime = '400x200'; --对话框大小  
}  
dlg:map() --绘制对话框
```

```
txt.append = ' world' --向文本编辑框中追加文本内容。
```

```
dlg:popup() --显示对话框
```

22) 效果:



3.3.8 按钮 (BUTTON)

1) 创建

使用button元素创建一个可点击的按钮控件。

使用:

```
btn = iup.button{  
-- {} 中可以设置 button 元素的属性。
```

简单示例:

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?', 根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的IUP基本函数库模块
```

```
btn= iup.button{ -- 使用button 元素创建了一个按钮控件。  
    rastertime = '200x',  
    title = 'Hello Button',  
    action = function () print('Click this button !') end  
}  
dlg = iup.dialog{btn, title = 'button'}  
dlg:popup() -- 创建对话框并显示
```

显示效果:



2) 属性: ALIGNMENT (对齐)

设置 button 中显示的文本相对于控件位置。默认值是: "ACENTER:ACENTER"。

使用:

```
btn.alignment = "ALEFT:ATOP"
```

--冒号左面为水平方向的相对位置, 这里是靠左; 右边是垂直方向的相对位置, 这里是顶部。

注意:

部分值也可以被接收。比如"ALEFT"或者":ATOP"; 另外的值则使用当前的对齐方式。

水平方向可接受的值:

"ALEFT" (左对齐), "ARIGHT" (右对齐), "ACENTER" (中心);

垂直方向可接受的值:

"ATOP" (上对齐), "ACENTER" (中心) and "ABOTTOM" (底部)。

3) 属性: FLAT (扁平化)

FLAT 属性会更改按钮的显示风格, 默认值是"no"; 当属性值被修改为"yes"时, 按钮的风格就会发生变化 (按钮的边框被隐藏起来, 看起来像一个 label 控件), 只有当鼠标浏览到按钮的区域时, 才会显示按钮的边框。

使用:

```
btn.flat = "yes";
```

注意:

仅控件绘制之前设置有效, map (绘制) 后赋值无效。

还可以看看 iup.flatbutton (预定义好的控件)。

4) 属性: TITLE (标题)

设置 button 的名字。

使用:

```
btn.title = "Hello Button" --设置按钮的显示文本为 'Hello Button'。
```

5) 属性: RASTERSIZE (大小)

按钮的大小 (以像素为单位)。

使用:

```
btn.rastersize = "100x"
```

--设置 button 宽度是 100, 高度值默认。具体请看 Dialog 中的 [RASTERSIZE](#)。

6) 回调: ACTION (行为)

通常是鼠标左键点击按钮触发的动作。

使用:

```
function btn:action()
```

```
...
```

```
end
```

7) 示例:

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换 '?', 根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
flat_btn = iup.button{title = 'Flat Button Style',rastersize = '200x100'}
```

--创建了一个 button 控件对象，包含了大小和文本标题。

```
flat_btn.flat = 'yes'
```

--设置 button 按钮是隐藏边框的，只有当鼠标移动到 button 范围时才显示边框。（边框总是存在的）。

```
normal_btn = iup.button{title = 'Normal Button Style',rastersize = '200x100'}
```

```
normal_btn.ALIGNMENT = "ALEFT:ATOP"
```

-- 对齐方式是:水平方向左对齐，垂直方向上对齐。

```
local dlg = iup.dialog{
```

```
    iup.vbox{
```

```
        flat_btn;
```

```
        normal_btn;
```

```
        gap = 10;
```

```
        margin = '10x10';
```

```
    };
```

```
    startfocus = normal_btn; -- 设置初始状态下，def_btn 代表的控件获得初始焦点
```

```
    title = 'Button';
```

```
}
```

```
function flat_btn:action() --鼠标左键点击按钮触发的操作。
```

```
    print('Flat Button is pressed ! '); --测试代码
```

```
end
```

```
function normal_btn:action() --鼠标左键点击按钮触发的操作
```

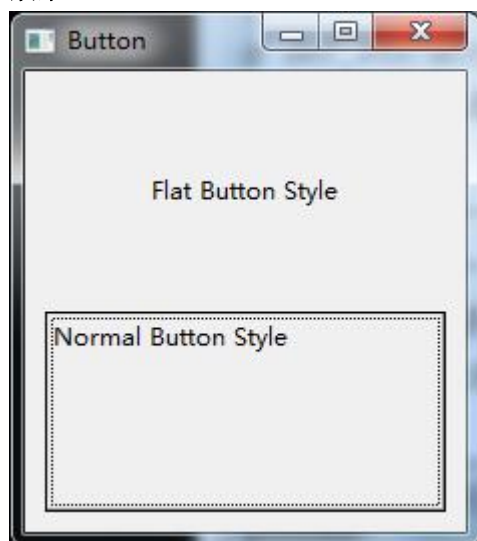
```
    print('Normal Button is pressed ! '); --测试代码
```

```
end
```

```
dlg:map() --绘制对话框
```

```
dlg:popup() --弹出对话框。
```

8) 效果:



3.3.9 标签 (LABEL)

1) 创建

使用 label 元素创建一个标签控件，标签可以显示一个分隔符、一个文本或者一个图像。
使用：

```
lab= iup.label {}  
-- {} 中可以设置 label 元素的属性。
```

简单示例：

```
package.cpath = "?53.dll;" .. package.cpath  
--检查所需要的库文件是否存在  
--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件  
是否存在。  
require 'iuplua' --加载需要的 IUP 基本函数库模块  
  
lab=iup.label{title    =   'Label',rastersize    =   '200x200',alignment    =  
'ACENTER:ACENTER'};  
--创建了一个 lable 对象，同时设置了其标题 title、大小 rastersize、以及对齐方式  
alignment;  
dlg = iup.dialog{lab, title = 'Label'}  
dlg:map()  
dlg:popup()
```

显示效果：



2) 属性: ALIGNMENT (对齐)

设置 label 中文本相对于 lable 控件的位置。默认值是: "ALEFT:ACENTER"

使用：

```
lab.alignment= "acenter:acenter" --冒号左面为水平对齐，右边是垂直方向对齐。
```

水平方向可能的值：

"ALEFT" (左对齐), "ARIGHT" (右对齐), "ACENTER" (中心)；

垂直方向可能的值：

"ATOP" (上对齐), "ACENTER" (中心) and "ABOTTOM" (底部对齐)。

注意：

部分值也可以被接收。比如"ALEFT"或者":ATOP"；另外的值则使用当前的对齐方式。

3) 属性: BGCOLOR (背景色)

在任何系统中 Label 的背景色属性都是被忽略的，它使用的是对话框中其容器的背景色，可以理解为 label 控件的背景色是透明的并且是更改无效的。通常改变 label 所在容器的背景色即可达到修改 label 背景色的目的。例如，使用 iup.frame 做容器,如下 dialog 创建示例：

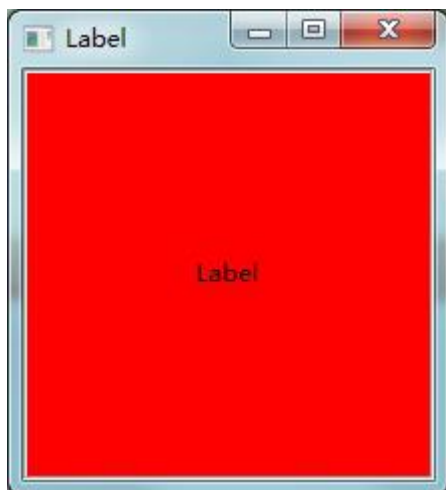
简单示例：

```
package.cpath = "?53.dll;" .. package.cpath
--检查所需要的库文件是否存在
--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件
是否存在。
require 'iuplua' --加载需要的 IUP 基本函数库模块

-----

lab= iup.label{title = 'Label',rastersize = '200x200',alignment =
'ACENTER:ACENTER'};
--创建了一个 label 对象，同时设置了其标题 title、大小 rastersize、以及对齐方式
alignment;
dlg = iup.dialog{ --创建 dialog
    iup.frame{ --使用 frame 控件做容器。
        lab; --label 控件对象
        bgcolor = '255 0 0'; --设置 frame 的颜色，这样在 frame 容器中的 label
控件的背景色也会跟着变化。
    };
    title = 'Label'; --对话框的标题
}
dlg:popup()
```

对话框显示效果如图：



4) 属性：FGCOLOR（前景色）

设置前景色，label 控件中设置的就是文本标题的颜色。

使用：

`lab.fgcolor = '0 0 255'` --蓝色。也可以使用 `'#rrgbbb'`（16 进制颜色表示法）。

5) 属性：SEPARATOR（分隔符）

设置将 label 控件做为分隔线使用。可能的值为： "HORIZONTAL"（水平方向的分隔线） or "VERTICAL"（垂直方向的分隔线）；

使用：

`lab.separator = "VERTICAL"` --设置 label 是一个垂直的分隔线。

注意：

仅控件绘制之前设置有效。

设置控件为分隔线，则标题等属性失效。

6) 属性：RASTERSIZE (大小)

设置将 label 控件的大小。

使用：

```
lab.rastersize= "100x"
```

--设置 label 控件的大小是水平方向上 100，垂直方向上默认(跟字体的大小也有关系)。

7) 示例：

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
lab_hello = iup.label{} --使用 label 元素创建一个 label 控件 (lab_hello 是一个接收了 label 控件的变量)。
```

```
lab_hello.rastersize = '300x'; -- 设置 lab_hello 的大小是水平方向上 300。
```

```
lab_hello.title = 'Hello Label'; --设置 label 控件的名称
```

```
lab_hello.fgcolor = '255 0 0'; --设置标题的前景色 (文本颜色)。
```

```
lab_sep = iup.label{separator = 'HORIZONTAL'}; --创建了一个水平方向上的分隔线 lab_sep
```

```
lab_text = iup.label{title = '上面的横线为水平方向上的分隔线 ! 当前的 label 演示如何简单的设置背景色 ! '}; --创建一个 label 对象 lab_text.
```

```
dlg = iup.dialog{
```

```
  iup.vbox{
```

```
    lab_hello;
```

```
    lab_sep;
```

```
    iup.frame{ --用 frame 作为 lab3 的容器，设置该容器的背景色，达到类似于修改标签背景色的效果。
```

```
      lab_text;
```

```
      bgcolor = '255 0 0';
```

```
    };
```

```
    gap = 10;
```

```
    margin = '10x10';
```

```
  };
```

```
  title = 'Label';
```

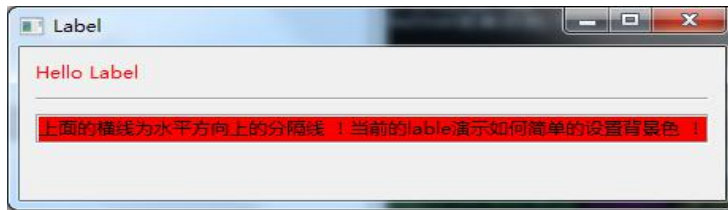
```
  rastersize = 'x150'
```

```
}
```

```
dlg:map()
```

```
dlg:popup()
```

8) 效果：



3.3.10 列表控件 (LIST)

1) 创建

使用 list 元素创建一个列表控件。依靠属性设置此控件可以包含四种不同风格的界面样式。

如下图这四种风格：(可编辑的下拉列表、不可编辑的下拉列表、带有编辑框的列表、没有编辑框的列表)



使用：

```
list= iup.list{}
```

简单示例：（下拉列表）

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
list = iup.list{ -- 使用 list 元素创建一个列表控件
```

```
dropdown = 'YES', -- 设置 list 是一个下拉列表控件
```

```
rastersize = '300x', -- 设置 list 大小是宽度 300 高度默认
```

```
'The First Value', -- list 列表中第一个值
```

```
'The Second Value',
```

```
'The Third Value',
```

```
'The Forth Value',
```

```
}
```

--dropdown 属性用来控制该列表是否为下拉列表。

```
dlg = iup.dialog{
```

```
list;
```

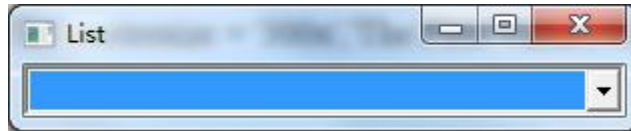
```
title = 'List';
```

```
}
```

```
dlg:popup() --弹出对话框。
```

显示效果：

弹出时:



展开时:



2) 属性: APPENDITEM (追加)

在控件中最后一条数据的后面插入一条数据。

使用:

```
list.appenditem = 'Append Item'
```

注意:

具有只写属性。绘制之前使用无效。

3) 属性: COUNT (列表数目)

返回列表中 item 的总数。通常在 map (绘制) 后使用。

使用:

```
print(list.count)
```

注意:

具有只读属性。

4) 属性: DROPDOWN (下拉列表)

设置控件样式是否为下拉列表控件。值可以是 "YES" or "NO"; 默认值是 'no' (不是下拉列表控件);

使用:

```
list.dropdown = 'yes' -- 修改控件的样式为下拉列表控件。
```

5) 属性: EDITBOX (编辑框)

为 list 添加文本编辑框; 值可以是 "YES" or "NO"; 默认值是 "NO" (不带编辑框或者不能编辑); 如果该属性的值为 "YES", 在 dropdown = 'no' 时, 列表控件会多出个文本编辑框, 在 dropdown = 'yes' 时, 可对下拉按钮旁边的文本显示区域进行文本编辑。

使用:

```
list.editbox = 'YES'
```

6) 属性: INSERTITEMid (插入值)

在被给的 id 位置前插入一个条数据。

使用:

```
list.insertitem1 = 'Insert First' -- 这里的 id 的值是 1, 意味着, 所赋的值将放在
```

列表第一位。

注意:

id 从 1 开始。如果 id 的值是 list.count+1 (列表总数+1) 意味着末尾追加效果等同于 appenditem 属性。

id 越界 (比如 id 的值为 list.count+2 或者小于等于 0 的数) 无效。绘制前设置无效。

具有只写属性。

7) 属性: MULTIPLE (多选)

允许可以同时选中多条数据。值可以是"YES" or "NO"; 默认值是"NO";

使用:

```
list.multiple = 'yes' --开启多选。
```

注意:

只有 list 的属性中 EDITBOX='NO' and DROPDOWN='NO'时, multiple = 'yes'才有效。

多选时, 请使用键盘上的 Ctrl 键加上鼠标点选。如果想快速选中并排多个可以使用 Shift 键加上鼠标操作。

8) 属性: REMOVEITEM (移除一项)

移除列表中不要的数据; 值起始于 1, 如果值是'all'或者是 nil, 会移除列表所有的项。

使用:

```
list.removeitem = 1 --移除列表中第一条数据。
```

注意:

在绘制之前设置无效。

9) 属性: VALUE (值)

list 的值取决于下拉和编辑框的组合。

当 editbox = 'YES' 时, list.value 获取到的就是编辑框中的值。

当 DROPDOWN='YES' or MULTIPLE='NO' 时, list.value 获取到的值是 id 号, id 从 1 开始的 (id 代表着数据在 list 控件中的位置)。

当 multiple = 'yes' 时, 其值用加减符号的字符串来表示, 加号代表选中, 减号则是代表没选中 (通过 '+' 号的位置取值)。

10) 属性: VISIBLEITEMS (dropdown 下显示 item 的数目)

list 的属性 DROPDOWN='YES' 时, 设置点击下拉按钮弹出的列表框中 item 可见数目。默认值是 5 (显示 5 个)。

使用:

```
list.visibleitems = 10 --设置可见 10 个。
```

11) 回调: ACTION (行为)

当 list 中的 item 的状态发生了变化触发的操作。

使用:

```
function list:action(str, item, state)  
-- 第一个参数是选项的文本  
-- 第二个参数选项的位置 (从 1 开始)  
-- 选中项的状态。1 是选中, 0 是取消选中  
print(str)  
print(item)  
print(state)  
-- 打印显示一下。  
end
```

12) 示例 1: (控件样式: 下拉带编辑框)

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?', 根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
list = iup.list{rastersize = '400x',}
```


--使用 *list* 元素创建一个水平方向上宽度为 400，垂直方向高度默认的 *list* 控件。

```
dlg = iup.dialog{  
    list;  
    title = 'List';  
}
```

list[1] = 'The First Value' -- *list* 控件中第一条数据

list[2] = 'The Second Value'

list[3] = 'The Third Value'

list[4] = 'The Forth Value'

-- *list* 的初值采用类似与数组的样式从 1 开始，中间不能有断层 *nil*，否则后面的值不会显示在控件中。

list.dropdown = 'yes' --设置 *list* 控件是一个下拉控件

list.editbox = 'YES' --设置控件可以编辑。

-- 可以将 *dropdown* 或者 *editbox* 属性的值修改试试效果。

list.VISIBLEITEMS = 3

--设置展开时显示区域内 *item* 个数，默认是 5 个，这里设置为 3 个，多出的需要拖动滚动条可见

list.value = *list*[1]

--为文本编辑框赋初值。（仅 *editbox* = 'YES' 试用）

dlg:map() --*map*，绘制对话框，绘制对话框时也会绘制对话框内所有的控件

list.APPENDITEM = 'Append Item'

--向控件中追加文本，在控件被绘制之后设置有效

list['INSERTITEM' .. (list.count+1)] = 'Insert Item'

--*count* 属性会获得控件中 *item* 的总数，利用 *insertitem* 达到追加效果。在控件被绘之后设置

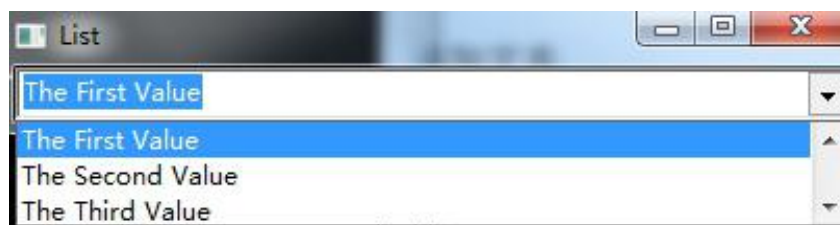
dlg:popup()

13) 效果

弹出时：



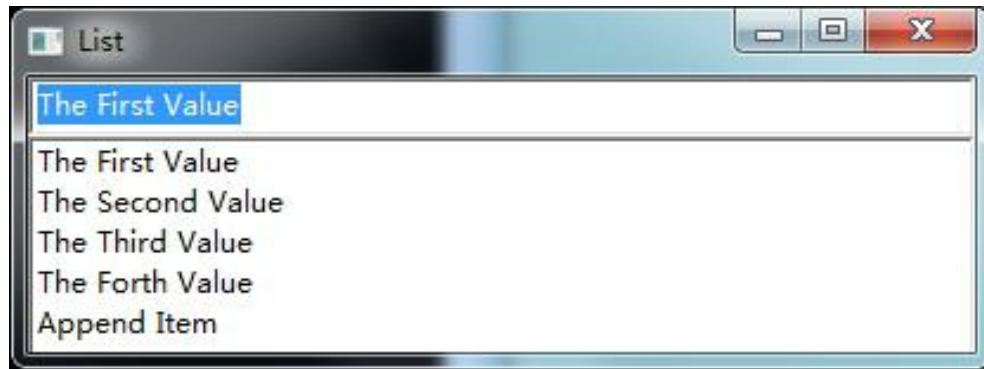
点击按钮展开时：（由于设置了只显示 3 个，后面的值只能通过鼠标滚轮滑动查看）。



14) 示例 2

将示例 1 中的 *list.dropdown* 的值设置为 'NO' 或者直接删掉这个属性。

15) 效果



3.3.11 矩阵控件 (MATRIX)

1) 创建

使用 `matrix` 元素创建一个矩阵类型的控件。在使用时需要 `require 'iupluacontrols'`。

使用：

```
mat= iup.matrix{}
```

简单示例：

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
require 'iupluacontrols' -- matrix 元素需要的库
```

```
mat = iup.matrix{} -- 使用 matrix 元素创建控件对象。
```

```
mat.numcol=2
```

```
mat.numlin=3
```

```
mat.numcol_visible=2
```

```
mat.numlin_visible=3
```

```
mat:setcell(0,0,"Inflation")
```

```
mat:setcell(1,0,"Medicine")
```

```
mat:setcell(2,0,"Food")
```

```
mat:setcell(0,1,"January 2000")
```

```
mat:setcell(0,2,"February 2000")
```

```
mat:setcell(1,1,"5.6")
```

```
mat:setcell(2,1,"2.2")
```

--setcell 的第一个参数代表 ‘行’，第二个参数代表 ‘列’，第三个参数用来设置该单元格的值

```
dlg = iup.dialog{iup.vbox{mat; margin="10x10"}}
```

```
dlg.title = 'Matrix';
```

```
dlg:popup()
```

显示效果：

Inflation	January 2000	February 2000
Medicine	5.6	
Food	2.2	

2) **属性：FOCUSCELL** (定义当前的单元格)

指定控件中当前的单元格。

接收的值样式为'*L:C*' (*L* 和 *C* 分别是代表行和列的两个数字);默认值是'*1:1*';

使用:

`mat.focuscell = '2:3'` -- 设置当前的单元格是第 2 行第 3 列的单元格。

注意:

如果该属性设置的值所代表的单元格不存在则, 属性设置无效。

3) **属性：HIDDEXTXTMARKS** (设置省略符号)

设置当时单元格中的文本大于单元格的空间时, 正常情况下会将文本裁剪到适合单元格大小。如果设置此属性为"*yes*", 则 '*...*' 符号会被放在可见区域的末尾, 用来表明还有文本不可见。

默认值是"*no*";

使用:

`mat.hiddentextmarks = 'yes'`

效果:

4) **属性：READONLY** (单元格只读属性)

禁用单元格的编辑功能。默认值是'*no*'(不禁用),

使用:

`mat.readonly = 'yes'`

-- 这样就禁止对单元格的编辑, 只能通过代码来操控矩阵中的单元格数据

5) **属性：VALUE** (设置当前单元格的值)

允许设置和返回当前单元格的值 (可以使用 *focuscell* 属性来指定当前单元格)。

使用:

`mat.value = 'new'` -- 设置当前单元格的值是'*new*'。

或者

`print(mat.value)` -- *print* 当前单元格的返回值。

6) **属性：CELLL:C** (获得 '*L:C*' 此单元格的值)

返回指定的单元格的值。

使用:

`print(mat['cell1:2'])` -- 打印显示第一行第二列的单元格的值。

注意:

该属性只读。

如果单元格不存在或者矩阵控件还没绘制时使用, 返回值是 *nil*。

7) **属性：ALIGNMENT** (文本对齐方式)

设置矩阵单元格中文本相对于单元格的位置 (水平方向上)。可接收的值: "*ALEFT*",

"ACENTER" or "ARIGHT".

使用:

`mat.alignment = "ARIGHT" -- 设置文本的对齐方式是右对齐。`

注意:

这里的 alignment 没有默认值。

8) **属性: ALIGNMENTC** (列中文本在水平方向上的对齐样式)

设置具体某一列中文本在水平方向上的对齐样式。可接收的值: "ALEFT", "ACENTER" or "ARIGHT".

使用:

`mat.alignment2 = "ARIGHT" -- 设置第 2 列的文本的对齐方式是右对齐。`

注意:

C 代表 column(列), 如果 C = 0 时, 默认值是 "ALEFT", C > 0 时的默认值是 "ACENTER"

在检查这个属性的默认值之前, 会先检查是否设置了 ALIGNMENT 属性。

9) **属性: ALIGNMENTLINO** (标题行的文本在水平方向上的对齐样式)

设置标题行 (第 0 行) 的文本对齐方式。可接收的值: "ALEFT", "ACENTER" or "ARIGHT". 默认值是: "ACENTER"

使用:

`mat.alignmentlin0 = 'ARIGHT' -- 设置右对齐`

10) **属性: LINEALIGNMENTL** (行中文本在垂直方向上的对齐样式)

设置指定行中文本在垂直方向上的对齐方式。可接收的值: "ATOP", "ACENTER" or "ABOTTOM". 默认值是: "ACENTER"

使用:

`mat.linealignment0 = 'atop' -- 设置第 0 行的文本在垂直方向上的对齐方式是上对齐。`

11) **属性: RESIZEMATRIX** (动态改变列宽)

矩阵的列宽可以使用鼠标拖拽从而发生变化。可能的值是 'yes' or 'no'; 默认的值是 "no"

使用:

`mat.resizematrix = "yes"`

12) **属性: WIDTHDEF** (设置默认列宽)

设置默认的列宽。

使用:

`mat.WIDTHDEF = 300 -- 设置默认的列宽是 300`

13) **属性: RASTERWIDTHn** (设置单元格宽度, n 代表第 n 列的)

单独设置每一列的列宽 (以像素为单位)。它的优先级比 WIDTHn 低。

使用:

`mat.rasterwidth0 = 20 -- 设置第 0 列的宽度是 20 (以像素为单位的值)`

`mat.rasterwidth1 = 100 -- 设置第 1 列的宽度是 100`

...

或者

`print(mat.rasterwidth1) -- 查看第一列的宽度 (以字符为单位的值 (实际大小))`

注意:

如果不单独设置列宽, 该列会采用默认的列宽。

14) **属性: HEIGHTDEF** (设置默认行高)

设置默认的行高。默认值是 8 ;

使用:

`mat.HEIGHTDEF = 20 -- 设置默认的行高是 20`

15) **属性: RASTERHEIGHTn** (设置单元格高度, n 代表第 n 行的)

单独设置每一行的行高 (以像素为单位)。它的优先级比 HEIGHTn 低。

使用:

mat.rasterheight0 = 20 -- 设置第 0 行的高度是 20 (以像素为单位的值)

注意:

如果不单独设置行高, 通常使用默认的行高 (HEIGHTDEF)。

16) **属性: ADDCOL** (添加列)

在指定的列的后面添加一个或者多个列。

使用:

mat.addcol = 1 -- 在第一列后插入一列。

或者添加多列:

mat.addcol = '1-4' -- 在第一列后面插入 4 列

注意:

如果在绘制之前设置此属性无效。

不能添加一个标题列 (第 0 列为标题列, 值设置为 0 时, 插入的列会成为第一列)。

如果对话框已经显示, 在操作此属性时要注意及时的调用 **redraw** 属性。比如 *mat.redraw = 'all'* 重绘一下矩阵控件。

17) **属性: ADDLIN** (添加行)

在指定的行的后面添加一个或者多个行。

使用:

mat.addlin = 4 -- 在第 4 行后插入一行

或者添加多行:

mat.addlin = '1-4' -- 在第一行后面插入 4 行

注意:

如果在绘制之前设置此属性无效。

不能添加一个标题行 (第 0 行为标题行, 值设置为 0 时, 插入的行成为首行)。

如果对话框已经显示, 在操作此属性时要注意及时的调用 **redraw** 属性。

18) **属性: DELCOL** (删除列)

删除指定的列, 一个或者多个。

使用:

mat.delcol = 4 -- 删除第 4 列

或者删除多列:

mat.delcol = '1-4' -- 删除从第 1 列到第 4 列 (包含 1,4)。

注意:

如果在绘制之前设置此属性无效。

不能删除标题列。

如果对话框已经显示, 在操作此属性时要注意及时的调用 **redraw** 属性。

19) **属性: DELLIN** (删除行)

删除指定的行, 一个或者多个。

使用:

mat.dellin = 4 -- 删除第 4 行

或者删除多行:

mat.dellin = '1-4' -- 删除从第 1 行到第 4 行 (包含 1,4)。

注意:

如果在绘制之前设置此属性无效。

不能删除标题列。

如果对话框已经显示，在操作此属性时要注意及时的调用 `redraw` 属性。

20) 属性: NUMCOL (设置列数)

定义在矩阵中的列数。多余的列不会显示。值必须是一个不小于 0 的整数。默认值是 0；
使用：

`mat.numcol = 4` -- 定义列数为 4 个，

或者：

`print(mat.numcol)` -- 获得当前矩阵的列数

注意：

列的数目并不包括标题栏（第 0 列）。

21) 属性: NUMCOL_VISIBLE (设置可见列数)

当控件的大小为自适应时，设置控件可见列的数目，列的数目不包括标题列，但是计算控件的自适应的大小时包括标题列。默认值是 4。（自适应大小是不设置控件的 `size` 或者 `rastersize` 属性）

使用：

`mat.numcol_visible = 2` -- 设置可见列数为 2 个

22) 属性: NUMLIN (设置行数)

定义在矩阵中的行数。多余的行不会显示。值必须是一个不小于 0 的整数。默认值是 0；
使用：

`mat.numlin = 4` -- 定义行数为 4 个，

或者：

`print(mat.numlin)` -- 获得当前矩阵的行数

注意：

行的数目并不包括标题栏（第 0 行）。

23) 属性: NUMLIN_VISIBLE (设置可见行数)

当控件的大小为自适应时，设置控件可见行的数目，行的数目不包括标题行，但是计算控件的自适应的大小时包括标题行。默认值是 3。

使用：

`mat.numlin_visible = 2` -- 设置可见行数为 2 个

24) 属性: MARKMODE (标记模式)

设置标记的模式。可接受的值为: "NO", "LIN", "COL", "LINCOL" or "CELL"; 默认的值: "NO" (选中时没有任何变化)；

'cell' -- 单元格模式。选中任意单元格，界面上都有反馈（背景色发生变化）

'lin' -- 行模式。选中行标题栏，整行被选中。

'col' -- 列模式。选中列标题栏，整列被选中。

'lincol' -- 行列模式。选中行或者列的任意标题栏，相对应的整行或者整列都被选中。

使用：

`mat.markmode = 'cell'` -- 设置标记模式为单元格模式。

25) 属性: MARKL:C (设置当前的 cell)

根据设定的标记模式来具体的标记某个单元格或者行或者列等，可接受的值是 "1" (标记) or "0" (不标记)。

使用：

`mat[markl:2] = '1'` -- 设值为 '1' 处于标记状态。

如果：

`mat.markmode = 'cell'` -- 产生的效果是第 1 行，第 2 列的单元格被选中。

`mat.markmode = 'lin'` -- 产生的效果是第 1 行的整行被选中。

`mat.markmode = 'col'` -- 产生的效果是第 2 列的整列被选中。

`mat.markmode = 'lincol'` -- 产生的效果是第 1 行的整行和第 2 列的整列被选中。

注意：

这仅是视觉上的被选中。因为 FOCUSCELL 属性所指向的值并不一定与 mark 设置的值相同。

不需要手动的 redraw。

26) 属性：REDRAW（设置重绘矩阵控件）

当矩阵发生了变化时，需要用户使用此属性来更新矩阵界面。

可接受的值是：

"ALL", 重绘整个矩阵。

"L%d", 重绘被给的单行。（"L3" 重绘第 3 行）

"L%d-%d", 重绘被给的多行。（"L2-4" 重绘第 2, 3 和 4 行）

"C%d", 重绘被给的单列。（"C3" 重绘第 3 列）

"C%d-%d", 重绘被给的多列。（"C2-4" 重绘第 2, 3 和 4 列）

通常情况下使用"ALL"会更方便一些。

注意：

该属性只接收值不返回值（只写）。

27) 属性：SHOW（显示不可见的位置）

当目标区域需要拖动滚动条才能看见的时候，设置此属性的值，让该目标区域可见。要滚动到某一行或者某一列使用值"L:*" or "*:C";

使用：

`mat.show = "5: *" --使滚动条滚动使第 5 行，出现在可见区域内。`

注意：

该属性只接收值不返回值（只写）。

对话框没弹出前设置无效。

28) 函数：SETCELL（单元格赋值）

设置单元格的值。

使用：

`mat.setcell(2,1,"5.6")`

--第 1 个参数代表行

-- 第 2 个参数代表列

-- 第 3 个参数是单元格的值

29) 函数：GETCELL（获取单元格值）

获得单元格中的值。

使用：

`print(mat.getcell(2,1))`

-- getcell 第一个参数是行

-- 第二个参数是列

30) 回调：ACTION_CB（键盘按键触发动作）

使用键盘触发的动作。

使用：

`function mat:action_cb(key, lin, col, number, value)`

`print('key= ' .. key)` -- 第一个参数以数字作为值代表所按下的键盘上的按键

`print('lin= ' .. lin)` -- 第二个参数代表所在的行

`print('col= ' .. col)` -- 第三个参数代表所在的列

`print('number= ' .. number)` -- 第四个参数代表编辑状态。1 代表处于编辑状态

```
print('value= ' .. value) -- 第五个参数单元格中的内容。
```

```
end
```

31) 回调：CLICK_CB（鼠标点击）

使用鼠标触发的操作。常见的有：鼠标左键单击、双击；鼠标右键单击。

使用：

```
function mat:click_cb(lin, col, str)
```

```
    print('lin= ' .. lin)      --第一个参数代表所在的行
```

```
    print('col= ' .. col)      --第二个参数代表所在的列
```

```
    print('str= ' .. str)      --第三个参数是一个字符串，通过字符串可以判断鼠标的操作。字符串中如果出现字符'D'代表有双击操作，字符 '1' 代表鼠标左键，2,3 则分别代表滚轮（按下而非滚动）和右键。
```

```
end
```

32) 示例

```
package.cpath = "?53.dll;" .. package.cpath
```

```
--检查所需要的库文件是否存在
```

```
--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件是否存在。
```

```
require 'iuplua' --加载需要的IUP 基本函数库模块
```

```
require 'iupluacontrols' --matrix 需要的库
```

```
-----  
mat = iup.matrix{} --使用 matrix 元素创建一个矩阵控件
```

```
mat.hiddentextmarks = 'yes' --设置当文本内容大于单元格时，则用...来代替被隐藏的部分
```

```
mat.numcol=6      --设计矩阵列数
```

```
mat.numlin=10     --设计矩阵的行数
```

```
mat.numcol_visible=3
```

```
--设置自适应状态下可见列数。为了防止6列所占的宽度过大，设计仅三列可见。可以拖动滚动条查看后面的列
```

```
mat.numlin_visible=4 --设置可见行数。
```

```
mat.rasterwidth0 = 50 --设置第0列标题列的宽度是100（以像素为单位）
```

```
mat.widthdef = 100 --设置默认的列宽为200
```

```
mat.heightdef = 20 --设置默认的行高为20
```

```
mat.rasterheight0 = 40 --设置第0行的高度为40（以像素为单位）
```

```
mat.alignment = 'ALEFT' --设置默认的文本对齐方式是右对齐（此属性无法控制标题）
```

```
mat.alignmentlin0 = 'ALEFT' --设置标题行的水平方向上的对齐方式为右对齐。
```

```
mat.linealignment0 = 'atop' --设置标题行的垂直方向上的对齐方式为顶部对齐
```

```
mat.alignment0 = 'ALEFT' --单独设置第一列在水平方向上的对齐方式是左对齐。
```

```
mat.readonly = 'yes' --设置单元格的文本不允许修改。
```

```
mat.markmode = 'lincol' --设置标记模式是'lincol'
```

```
mat:setcell(0,0,"Inflation") --设置第0行，第0列的单元格中显示的文本内容
```

```
mat:setcell(1,0,"Medicine")
```

```
mat:setcell(2,0,"Food")
```

```
mat:setcell(3,0,"Water")
```

```
mat:setcell(4,0,"Fruit")
```

```
mat:setcell(5,0,"Result")
```

```
mat:setcell(0,1,"January 2000")
```

```
mat:setcell(0,2,"February 2000")
```

```
mat:setcell(0,3,"March 2000")
```



```

mat:setcell(0,4,"March 2000")
mat:setcell(1,1,"5.6")
mat:setcell(2,1,"2.2")
mat:setcell(1,2,"这是文本超出单元格大小的测试用例。这是文本超出单元格大小的测试用例。
这是文本超出单元格大小的测试用例。")
--上面使用 setcell 的方式赋值。setcell 的第一个参数是行，第二个参数代表列，第三个参数
就是所要赋的值。
mat:focuscell = '2:2' -- 指定焦点单元格
mat.value = '当前焦点在这个单元格,\n 这是通过使用 value 这个属性赋的值!' -- 设置焦点
单元格的值,这里使用了换行符\n';
mat['mark1:2'] = '1' --设置第一行第2 列的标记状态。由于 markmode 的值是'lincol', 所以
第1 行和第2 列的所有单元格被标记。
dlg = iup.dialog{iup.vbox{mat; margin="10x10"}}
dlg.title = 'Matrix';
function mat:action_cb(key, lin, col, number, value) --处理键盘上的按键操作
    print('key= ' .. key)
    print('lin= ' .. lin)
    print('col= ' .. col)
    print('number= ' .. number)
    print('value= ' .. value)
    -- 操纵键盘时, 将参数打印出来以供参考学习。
end

function mat:click_cb(lin, col, str) --处理鼠标点击操作
    print('lin= ' .. lin)
    print('col= ' .. col)
    print('str= ' .. str)
    -- 操纵鼠标时, 将参数打印出来以供参考学习。
end

dlg:popup()

```

33) 效果



可以看到第 0 行和第 0 列是标题栏。

3.3.12 树形控件 (TREE)

1) 创建

使用 `tree` 元素创建一个树形控件。控件中任何一条数据都可以称之为节点，`tree` 中的节点有两种形式：branch（分支）和 leaf（叶子），分支节点是可以展开也可闭合的，叶子节点则不可以。`tree` 中的节点都有相应的 id 标识的（id 是从 0 开始的数字，当有只有一个根节点时，根节点的 id 是 0）

使用：

```
tree = iup.tree{} -- {} 中可以设置 tree 的属性.
```

注意：

每个节点都有代表它的 id 数字，数字从 0 开始，所以第一个节点的 id 是 0，最后一个节点的 id 是节点总数减 1。Tree 中通常使用 id 来处理很多操作。

简单示例：（仅有一个节点的示例）

```
package.cpath = "?53.dll;" .. package.cpath
```

```
--检查所需要的库文件是否存在
```

```
--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件是否存在。
```

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
-----  
tree = iup.tree{rastersize = '400x100'}
```

```
-- 使用 tree 元素创建一个树形控件。 {} 中可以设置 tree 的属性. 这里设置了控件的大小
```

```
tree.title0 = 'The root node is a branch node !'
```

```
--设置 tree 中跟节点的显示文本，根节点 id 是 0
```

```
dlg = iup.dialog{
```

```
    iup.vbox{
```

```
        tree;
```

```
        margin = '10x10';
```

```
    };
```

```
}
```

```
dlg.title = 'Sample Tree'; --对话框标题。
```

```
dlg:map()
```

```
-- 对话框被绘制时，对话框内所有的控件都会被绘制。
```

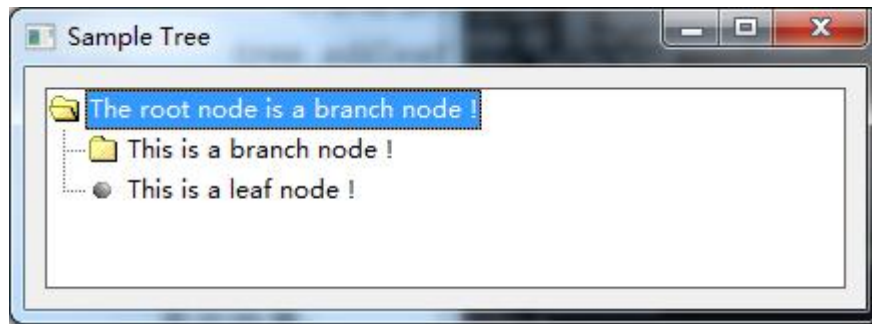
```
tree.addleaf = 'This is a leaf node !' -- 添加一个叶子节点
```

```
tree.addbranch = 'This is a branch node !' -- 添加一个分支节点。
```

```
-- 上面这两个属性要在控件被绘制后使用
```

```
dlg:popup()
```

显示效果：



2) **属性：TITLEid** （节点的标题）

设置指定节点的文本，id 是指定节点的标识

使用：

```
tree.title0 = 'The first node' --设置 tree 的第 0 个节点的标题.
```

3) **属性：ADDEXPANDED** （添加展开）

设置默认状态下是否展开所有的分支节点。可接受的值有“yes”和“no”，默认值是“yes”。

使用：

```
tree.addexpanded= "no" -- 一般都会设置成"no"不自动展开所有节点。
```

4) **属性：COUNT** （节点总数）

返回 tree 中节点的总数。没有节点返回值是 0.

使用：

```
print(tree.count)
-- 输出查看 tree 中节点的总数
```

注意：

具有只读属性。

5) **属性：SHOWDRAGDROP** （拖拽节点）

控件中可以拖拽节点。默认值是“no”；

使用：

```
tree.showdragdrop = 'yes' -- 启用拖拽效果
```

注意：

只有当属性 MARKMODE='SINGLE' 时有效。

只有在创建时设置才有效。

6) **属性：TOPITEM** （节点可见）

将当前不可见的节点（节点在分支内未被展开或者需要拖动滚动条才能看见的），将其显示出来（自动的展开分支节点或者滑动滚动条），属性的值是节点 id。

使用：

```
tree.topitem = 10 -- 将 id 是 10 的节点显示出来
```

注意：

具有只写属性。指定的 id 要存在否则没有意义。

7) **属性：CHILDCOUNTid** （子节点个数）

返回指定分支节点下的节点个数。

使用：

```
print(tree.CHILDCOUNT0) -- 查看 id 是 0 的分支下的节点个数。
```

注意：

具有只读属性。

它仅返回指定分支节点中子节点的个数（子节点的个数）。

8) **属性：COLORid** （颜色）

设置指定节点的文本颜色。如果没有指定节点，那么 tree 中具有焦点的节点文本会被改变颜色。

使用：

```
tree.color0 = '255 0 0' -- 设置 id 为 0 的节点文本颜色是红色。
```

注意：

值也可是 16 进制的颜色表示法比如 '#rrggbb'。

9) 属性：DEPTHid（深度）

返回指定节点所处的深度（从 0 开始，根节点的深度是 0）。

使用：

```
print(tree.depth0) -- 查看 id 是 0 的节点的深度
```

注意：

具有只读属性。

10) 属性：KINDid（类型）

返回指定节点的类型。节点存在返回值必然是“LEAF”和“BRANCH”二者之一。如果节点不存在返回 nil。

使用：

```
print(tree.kind0) -- 查看 id 是 0 的节点的类型
```

注意：

具有只读属性。

11) 属性：PARENTid（父节点的 id）

返回指定节点的父节点的 id。

使用：

```
print(tree.Parent1) -- 查看 id 是 1 的节点的父节点
```

注意：

具有只读属性，节点的 id 必须存在否则返回 nil。如果指定节点所处的深度是 0，那么值也是 nil。

12) 属性：STATEid（状态）

设置或者返回指定分支节点的状态（展开或者闭合）。设置时可接收的值是“EXPANDED”和“COLLAPSED”（可小写）。使用其返回值时，结果是“EXPANDED”和“COLLAPSED”之一。如果节点不存在返回值是 nil。

使用：

```
tree.state0 = 'collapsed' -- 设置分支 id 是 0 的节点状态为闭合状态
```

```
print(tree.state0) -- > "COLLAPSED" -- 获得 id 是 0 的节点的状态。
```

注意：

它仅会影响有子节点的分支节点。

如果指定节点是一个叶子节点时设置无效，返回是 nil。

13) 属性：TOTALCHILDCOUNTid（返回指定分支节点包含的所有节点个数）

获得指定分支节点的所有节点个数（不仅包含子节点，还包含子节点中分支节点下的所有节点）

使用：

```
print(tree.TOTALCHILDCOUNT0) -- 获得 id 是 0 的分支节点下所有节点的个数
```

注意：

具有只读属性。

14) 属性：USERDATAid（设置或者反馈指定节点的属性）

设置或者返回指定节点的属性（属性就是节点附着的值，通常使用表作为其附着的值，其他的类型的值也可以接收）

使用:

```
tree userdata0 = 'This is a string data !' -- 设置 id 是 0 的节点的属性是一个字符串
```

```
print(tree userdata0) -- > 'This is a string data !' -- 将 id 是 0 的节点附着的字符串值打印出来。
```

注意:

必须在绘制之后设置此值才有效。

如果设置节点附着的数据类型是 userdata 或者 table, 也可以使用 `iup.TreeSetUserId(tree, id, data)` 这种方式设置节点的数据。(如果指定的节点事先附着了其他类型的数据则会引发错误)。

也可以使用 `iup.TreeGetUserId(tree, 0)` 来获得指定节点附着的数据表。

15) 属性: IMAGEid (图标)

设置指定节点的图标样式。

使用:

```
tree.image0 = "c:\\image.bmp" -- 使用本地文件作为节点的图标。
```

注意:

1、具有只写的属性。

2、节点要存在否则无效。指定的文件如果不存在则图标会变成空白。

3、如果指定的节点是一个分支节点则图片仅用作该分支节点闭合时的样式。

4、值可以使用 IUP 自带的 imagelib 库中的图片 (需要加载图片库, 比如: `require "iupluaimglib"`), 也可以使用本地 bmp 格式的图片 (注意图片的像素尺寸, 尺寸的宽高一般在 16 ~ 18 之间)。

16) 属性: IMAGEEXPANDEDId (设置分支节点展开图标)

设置分支节点展开的图标。闭合图标由 imageid 属性设置。

使用:

```
tree.imageexpanded0 = "c:\\image_expand.bmp" -- 使用本地文件作为节点的图标。
```

注意:

需要注意的地方与 IMAGEid 一致。

17) 属性: IMAGELEAF (设置默认叶子节点图标)

设置叶子节点的默认图标。默认值: "IMGLEAF"; lua 自带的值 "IMGBLANK" and "IMGPAPER" 也可以使用。

使用:

```
tree.imageleaf = "c:\\image_leaf.bmp" -- 使用本地文件作为节点的图标。
```

注意:

如果有叶子节点使用了属性 imageid 来设置, 那么该节点的图标仍然是 imageid 设置的图片。

18) 属性: IMAGEBRANCHCOLLAPSED (设置默认的分支节点闭合图标)

设置默认的分支节点在闭合状态时的图标。默认值: "IMGCOLLAPSED";

使用:

```
tree.imagebranchcollapsed = "c:\\image_branch_collapsed.bmp"  
-- 使用本地文件作为节点的图标。
```

19) 属性: IMAGEBRANCHEXPANDED (设置默认的分支节点展开图标)

设置默认的分支节点在展开状态时的图标。默认值: "IMGEXPANDED"

使用:

```
tree.imagebranchexpanded = "c:\\image_branch_expanded.bmp"  
-- 使用本地文件作为节点的图标。
```

20) 属性: VALUE (值)

返回值是焦点所在节点 id (标识符)。在单选模式下 (MARKMODE="SINGLE") 时, 设置 value 的值也可以将节点选中。

使用:

```
print(tree.value) -- 显示当前选择节点
tree.value = 2 -- id 是 2 的节点被选中。
```

注意:

如果 tree 中有节点但是没有焦点返回值是 0;

如果 tree 中连节点都没有返回值是-1;

21) 属性: MARKEDid (指定节点选中状态)

指定节点的选中状态。id 是指定节点的标识符, 如果 id 为空或者不存在, 则焦点所在的节点会被用作参考节点。可以接收的值有 'yes' 和 'no', 默认值是 'no'

使用:

```
tree.marked2 = "yes" -- id 是 2 的节点被标记为选中。
```

22) 属性: MARKMODE (选择模式)

定义如何选择节点。可以是单选模式值为 'SINGLE', 也可以是多选值为 'MULTIPLE'; 默认是单选。

使用:

```
tree.MARKMODE = 'MULTIPLE' -- 启用多选。
```

23) 属性: ADDLEAFid (添加叶子节点)

在指定节点的后面添加一个叶子节点。

使用:

```
tree.addleaf0 = 'add a new leaf node'
-- 在 id 是 0 的节点后面添加一个叶子节点, 节点显示的标题就是 'add a new leaf node'
```

注意:

具有只写属性。

如果在绘制 (map) 前设置无效。

如果节点 id 是叶子节点, 所添加的节点的位置是处于该节点的下方 (兄弟位置)。如果节点 id 是分支节点, 所添加的节点的位置就称为该分支的第一个子节点。

如果添加节点的 id 值是-1, 则所添加的节点会成为第一个节点。

如果想要插入一个分支的下方 (做兄弟节点) 可以使用 INSERTLEAFid 属性设置。

24) 属性: ADDBRANCHid (添加分支节点)

添加一个分支节点在给定节点的后面。

使用:

```
tree.addbranch0 = 'add a new branch node'
```

注意:

注意的地方与 addleaf 类似。如果想要插入一个分支的下方 (做兄弟节点) 可以使用 INSERTBRANCHid 属性设置。

25) 属性: DELNODEid (删除节点)

删除指定的节点和它的子节点或者仅删除它的子节点。id 是指定节点的标识。可接受的值有:

"ALL": 删除所有的节点, id 的值被忽略。

"SELECTED": 删除指定的 id 节点和它的所有子节点。

"CHILDREN": 删除指定节点的所有子节点, 但是不包括节点本身。

"MARKED": 删除所有选中的节点 (包含子节点)。id 被忽略

使用:

```
tree.DELNODE0 = "CHILDREN" -- 设置删除 id 是 0 的节点下所有子节点。
```

注意:

绘制之前设置无效。具有只写属性。

26) 属性: EXPANDALL (设置所有分支节点的展开闭合状态)

控制所有分支节点的展开闭合状态, 可接受的值有 'yes' 和 'no'。

使用:

```
tree.expandall = 'yes'
-- 展开所有分支节点。
```

27) 回调: SELECTION_CB (选择节点)

当节点被选中或者释放时触发的函数。

使用:

```
function tree:selection_cb(id, status)
    -- id: tree 中节点的标识, status: 选中和未选中的状态值
    print('id = ' .. id)
    print('status = ' .. status)
end
```

注意:

使用时可以看到 status 为 0 时, id 值是上一个选中节点的 id (如果有)。Status 值为 1 时, id 是当前选择节点的 id。

28) 回调: RIGHTCLICK_CB (鼠标右键点击)

鼠标右键按下时触发的函数。

使用:

```
function tree:rightclick_cb (id) -- id: tree 中右键选中的节点标识
    print('id = ' .. id)
end
```

29) 回调: BUTTON_CB (处理鼠标消息)

当鼠标的任意键按下或者释放时触发的函数。tree 中没有直接给出双击节点触发事件函数, 所以可以使用 button_cb 来处理双击消息。

使用:

```
function tree:button_cb(button, pressed, x, y, status)
    print('button = ' .. button)
    print('pressed = ' .. pressed)
    print('x = ' .. x)
    print('y = ' .. y)
    print('status = ' .. status)
    -- 打印查看各个参数代表含义。
end
```

想要实现 tree 中双击消息示例:

```
function tree:button_cb(button, pressed, x, y, str)
    if string.find(str, "l") and string.find(str, "D") then
        -- 执行双击操作函数。
    end
end
```

-- 注意: 想要知道双击的是哪个节点, 还需要其他操作获得。

30) 回调: MAP_CB (绘制后的更新)

控件在绘制完成后的更新函数;

使用:

```
function tree:map_cb() ... end
或者使用
tree.map_cb = function () ... end
```

注意:

如果使用 tree 的 addbranch 和 addleaf 等属性形成 tree 中的数据, 那么必须在 tree 被绘制后才能使用。这时可以使用 map 方法, 将数据和处理数据的函数放入 map 方法中, 这样在控件绘制完成后, 自动调用 map 里的处理函数, 从而形成完整的带数据信息的 tree。

31) 示例

```
package.cpath = "?53.dll;" .. package.cpath
--检查所需要的库文件是否存在
--require 后的模块名替换?', 根据替换的结果在设置的目录下或者环境中查找文件是否在。
require 'iuplua' --加载需要的IUP 基本函数库模块

-----

tree = iup.tree{rastersize = '300x200'}
--使用 tree 元素创建一个树形控件, {} 中可以设置 tree 的属性.这里设置了 tree 的大小
tree.ADDEXPANDED = 'no' --设置不展看所有的分支节点
tree.title0 = 'Root Node' --设置 tree 的第 0 个节点的标题, 注意节点是从 0 开始的。
tree.showdragdrop = 'yes' --设置 可拖拽节点。
tree.color0 = '255 0 0' --设置 id 是 0 的节点颜色
tree.imageleaf = "IMGPAPER" --设置全局默认的叶子节点图标
tree.MARKMODE='SINGLE' --设置 选择模式为单选模式
dlg = iup.dialog{
    iup.vbox{ -- vbox 容器
        tree; --tree 变量
        margin = '10x10'; --间距
    };
}

function tree:selection_cb(id,status) --选择回调函数
    --id: tree 中节点的标识, status: 选中和未选中的状态值
    print('id = ' .. id)
    print('status = ' .. status)
    print('选择节点回调')
end

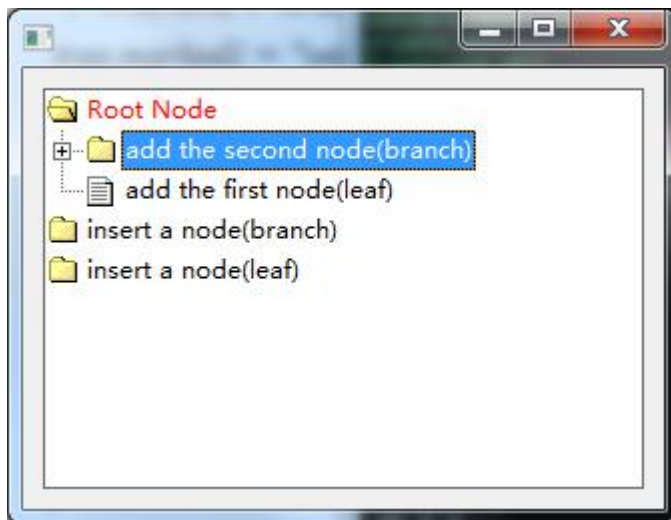
function tree:rightclick_cb (id) -- id: tree 中右键选中的节点标识
    print('id = ' .. id)
    print('右键点击操作')
end

function tree:button_cb(button,pressed,x,y,str)
    if string.find(str,"I") and string.find(str,"D") then
        --执行双击操作函数。
        print('在这里执行双击操作函数。')
    end
end
```


`dlg:map()`

```
tree.addleaf0 = 'add the first node(leaf)' -- 在 id 为 0 的分支节点下添加一个叶子节点。  
tree.USERDATA0 = 'This is a string data !' -- 设置 id 是 0 的节点的用户数据  
tree.addbranch0 = 'add the second node(branch)' -- 在 id 为 0 的分支节点内继续添加了一个分支节点。  
tree.addbranch1 = 'add a child node(branch)' -- 在 id 为 0 的分支节点内继续添加了一个分支节点  
tree.insertbranch0 = 'insert a node(leaf)' -- 在 id 为 0 的分支节点下插入了一个叶子节点  
tree.insertbranch0 = 'insert a node(branch)' -- 在 id 为 0 的分支节点下插入了一个分支节点  
-- tree.topitem = 2  
-- 设置节点 id 是 2 的节点可见，如下图中如过不设置此值。'add the second node(branch)' 这个节点将处于闭合状态。  
tree.marked1 = "yes" -- 将节点 1 标记为选中的状态。  
print(tree.USERDATA0) --> 'This is a string data !' -- 将 id 是 0 的节点的数据打印出来  
print(tree.count) --> 6 -- 将 tree 中所有节点的个数打印出来。  
print(tree.CHILDCOUNT0) --> 2 -- 将 id 是 0 的节点的子节点个数打印出来  
print(tree.depth2) --> 2 -- 将 id 是 2 的节点的深度打印出来  
print(tree.kind1) --> 'BRANCH' -- 将 id 是 1 的节点的类型打印出来  
print(tree.parent2) --> 1 -- 将 id 是 2 的节点的父节点 id 打印出来  
print(tree.state0) --> 'EXPANDED' -- 将 id 是 0 的节点的展开状态打印出来  
print(tree.TOTALCHILDCOUNT0) --> 3 -- 将 id 是 0 的节点的子节点个数打印出来  
print(tree.VALUE) --> 1 -- 从此值可以看到 tree 中当前焦点 id 是 1  
dlg:popup()
```

32) 效果



3.3.13 菜单控件 (MENUS)

通常使用菜单作为对话框的菜单或者作为某种控件的右键弹出菜单来使用。

想要建立一个完整的菜单控件可能会需要 `iup.item`、`iup.menu`、`iup.separator`、`iup.submenu` 这四种控件。这里将会对这些控件的属性方法进行一一介绍。

1) iup.item:

创建菜单中的 item。一个 item 代表着菜单中拥有具体执行操作的一项（仅 item 可自定义鼠标左键点击时触发的操作行为）

TITLE（标题属性）：

设置 item 的显示文本。

ACTION（鼠标点击的操作行为）：

设置点击 item 的操作行为。

2) iup.menu:

创建一个菜单控件对象。它只能接收 item、submenu、separator 这三种对象。

3) iup.separator:

创建菜单中的分隔线；在菜单中的两个 item 间显示一条线。

4) iup.submenu:

创建一个菜单项，用于点开时展开另一个菜单。

TITLE（标题属性）：

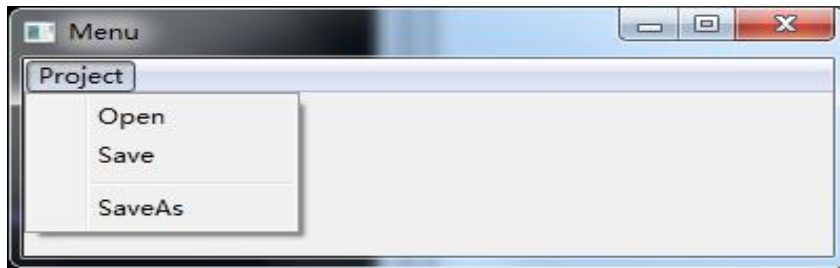
设置 submenu 的显示文本。

5) 示例：

```
package.cpath = "?53.dll;" .. package.cpath
--检查所需要的库文件是否存在
--require 后的模块名替换'?'，根据替换的结果查找文件是否在该路径下存在。
require 'iuplua' --加载需要的IUP 基本函数库模块
```

```
-----
Menu_ =
iup.menu{ -- 创建了一个 menu
    iup.submenu{ --创建了一个 submenu
        title = 'Project', -- submenu 的标题
        iup.menu{ -- submenu 中仍然要包含 menu
            iup.item{ --创建了一个 item
                title = 'Open', --item 的标题
                action =function () print('Deal Open Action !') end
                --action 点击此 item 执行的操作。
            };
            iup.item{title = 'Save',action =function () print('Deal Save Action !') end};
            iup.separator{}; -- 创建了一个分隔线
            iup.item{title = 'SaveAs',action =function () print('Deal SaveAs Action !') end};
        }
    }
};
dlg = iup.dialog{rastersize = '300x200'}
dlg.title = 'Menu';
dlg.menu = Menu_ -- Menu_ 被设置为对话框的菜单。
dlg:map()
dlg:popup()
```

6) 效果：



3.3.14 单选控件 (RADIO)

1) 创建

使用 radio 元素创建一个单选按钮。其实是创建一个空的容器用来将复选框分组并使他们处于互斥状态，即同一时刻只能选择一个。

使用：

```
radio = iup.radio{}
```

样式：



2) 属性:VALUE

设置或者获得当前选择的控件对象。

使用：

```
radio.value = tog
```

-- tog 是创建好的 toggle 控件对象（该控件要在 radio 容器的范围内）。

有关 radio 的示例请看下面的 **TOGGLE** 示例。

3.3.15 多选控件 (TOGGLE)

1) 创建

使用 toggle 元素创建可切换选择的控件。该控件有两种按钮状态：ON/OFF(选中/取消)。

使用：

```
tog = iup.toggle{}
```

样式：



2) 属性:TITLE (文本标题)

设置创建的 toggle 控件对象中显示的文本。

使用：

```
tog.title = 'tog'
```

3) 回调: ACTION (行为)

设置当 toggle 控件对象被鼠标左键点击复选框时触发的操作。

使用：

```
tog.action = function ()print('deal action !') end
```

4) 属性:RIGHTBUTTON (右侧复选框)

设置复选框放置在文本的右侧（默认是在左侧）。可接受的值有 'yes' 和 'no'，默认值是 'no'；

使用:

```
tog.rightbutton = 'yes' -- 使用该属性, 将选择框放置在文本右侧。
```

注意:

在绘制之前设置有效。

5) 属性:VALUE (值, 选择状态)

设置或者获得 toggle 的选择状态。值能是"ON", "OFF" or "TOGGLE";默认值是"OFF"。这个"TOGGLE"的值会反转当前的设置的状态。

使用:

```
tog.value = 'TOGGLE' -- 更改当前的选中状态。
```

或者:

```
print(tog.value) -- 查看一下当前的状态。
```

注意:

获得返回值时永远是大写的。设置值时可以忽略大小写的变化。

获取返回值时要在控件被绘制后(对话框被绘制之后)使用

6) 回调: VALUECHANGED_CB (状态改变)

当用户操作控件时, 控件的状态发生了变化时触发的操作。

使用:

```
function tog:valuechanged_cb()
    print(tog.value)
end
```

注意:

如果设置了回调函数 action, valuechanged_cb 的执行动作发生在 action 之后。

7) 示例

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换'?', 根据替换的结果在设置的目录下或者环境中查找文件是否存在。

```
require 'iuplua' --加载需要的 IUP 基本函数库模块
```

```
tog_open_ = iup.toggle{title = 'Open', action = function() print('Selected Open ')
end}
```

```
tog_save_ = iup.toggle{title = 'Save', action = function() print('Selected Save ')
end}
```

```
tog_new_ = iup.toggle{title = 'New', action = function() print('Selected New ') end}
-- 使用 toggle 元素分别创建三个复选控件。
```

```
radio = iup.radio{ --使用 radio 元素创建了单选控件容器。
```

```
    iup.hbox{ -- 该容器中可以使用 hbox, vbox 将复选控件排版。
```

```
        tog_new_,
```

```
        tog_open_,
```

```
        tog_save_;
```

```
        -- 将上面定义的三个复选框设置为彼此互斥的单选控件。
```

```
        gap = '20';
```

```
        margin = '10x10';
```

```
    };
```

```
}
```

```
radio.value = tog_open_ -- 设置单选控件初始值。
```

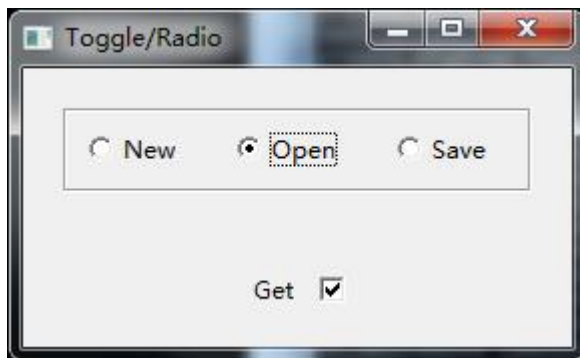
```

tog_get_ = iup.toggle{title = 'Get'}
tog_get_.action = function () print('deal action !') end
tog_get_.rightbutton = 'yes'    --设置控件的文本显示在复选框的左面。
tog_get_.value = 'ON'    --设置该控件初始的状态，默认值是'OFF'

dlg = iup.dialog{
    iup.vbox{
        iup.frame{radio}; -- 将radio 控件放置在 frame 控件下较为美观
        tog_get_;
        gap = '40';
        margin = '20x20';
        alignment = 'ACENTER';
    };
}
dlg.title = 'Toggle/Radio';
function tog_get_:valuechanged_cb()    -- 设置 tog_get_ 所代表的控件状态发生变化时
    处理函数。
    print(self.value)    -- 这里打印显示当前的状态。
end
dlg:map()
dlg:popup()

```

8) 效果



3.3.16 框架控件 (FRAME)

1) 创建

使用 frame 元素创建一个边框。

使用：

```
frm = iup.frame{}
```

2) 属性: BGCOLOR (背景色)

设置 frame 控件的背景色。

使用：

```
frm.bgcolor = '255 0 0';
```

-- 设置背景色为红色，也可以使用 16 进制的颜色表示法：'#rrggbb'；

注意：

设置 frame 的背景色时，一定要注意不能设置 title 属性。并且要在绘制之前设置。

3) 属性: TITLE (文本)

设置 frame 的标题文本; 默认在 frame 控件的左上角显示。

使用:

```
frm.title = 'Frame'
```

4) 属性: RASTERSIZE (大小)

设置 frame 控件的大小 (以像素为基本单位)。

使用:

```
frm.rastersize = '300x100'
```

5) 简单示例

```
package.cpath = "?53.dll;" .. package.cpath
```

--检查所需要的库文件是否存在

--require 后的模块名替换 '?', 根据替换的结果查找文件是否在该路径下存在。

require 'iuplua' --加载需要的 IUP 基本函数库模块

```
frm = iup.frame{}
```

```
frm.bgcolor = '255 0 0'; -- 虽然设置了 bgcolor 属性, 但是由于下面设置了 title 属性所以没有起到任何效果。
```

```
frm.title = 'Frame' -- 设置 frame 控件的标题文本。
```

```
frm.fgcolor = '255 0 0'; -- 设置前景色
```

```
frm.rastersize = '300x100' -- 设置 frame 控件的像素大小。
```

```
dlg = iup.dialog{
```

```
    iup.vbox{
```

```
        frm;
```

```
        margin = '10x10';
```

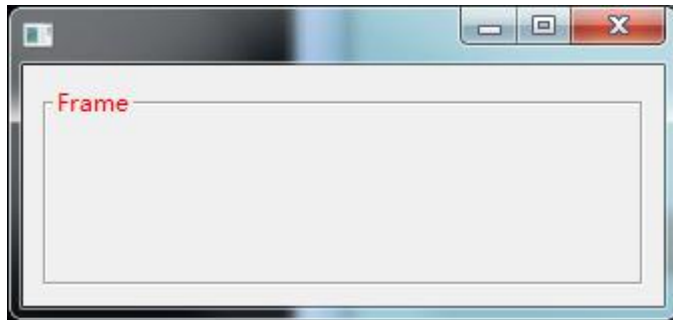
```
    };
```

```
}
```

```
dlg:map()
```

```
dlg:popup()
```

6) 效果



3.3.17 分页控件 (TABS)

1) 创建

使用 tabs 元素创建一个可以有多个标签页的控件。

使用:

```
tabs = iup.tabs{}
```

2) 属性: COUNT (数目)

获得标签页的个数。

使用:

```
print(tabs.count)
```

注意:

具有只读属性。

3) 属性: SHOWCLOSE (关闭按钮)

使每个标签页都有关闭按钮。默认值是 'no' ;

使用:

```
tabs.SHOWCLOSE = 'yes' -- 使用可关闭按钮
```

注意:

关闭的标签页实际上是被隐藏了。

4) 属性: TABTITLEn (标签名)

设置标签页的标题名。n 从 0 开始。

使用:

```
tabs.TABTITLE1 = 'Project' -- 第一个标签页的标题是'Project'。
```

注意:

也可以在标签的子控件对象中使用 TABTITLE 属性设置该标签的标题。

5) 属性: TABTYPE (类型)

设置标签页的类型(选项卡的位置)。可以是"TOP", "BOTTOM", "LEFT" or "RIGHT"; 默认值是"TOP";

使用:

```
tabs.TABTYPE = "BOTTOM" - 设置选项卡的位置在标签页的底部。
```

6) 属性: VALUE (设置当前页)

设置当前标签页。有多个标签页时, 可以使用程序控制标签页的切换。它接收的值是标签页中包含的子控件对象名称, 根据子控件的对象名来切换。

使用:

```
tabs.VALUE = vbox -- vbox 是某个标签中创建好的容器 vbox 对象。
```

注意:

切换标签的属性都需要在控件被绘制之后设置(通常在对话框弹出后使用)。

使用程序来控制标签页的切换还可以使用 **VALUE_HANDLE** 和 **VALUEPOS** 这两种属性。**VALUEPOS** 是根据标签页的位置来切换(标签页的位置从 0 开始)。**VALUE_HANDLE** 可以根据子控件对象句柄来切换。

7) 回调: TABCHANGE_CB (切换标签)

回调函数, 处理切换标签页时触发的事件。

使用:

```
function tabs:tabchange_cb(new_tab, old_tab)
    -- 第一个参数是切换后的标签页
    -- 第二个参数是切换前的标签页
    print(new_tab.tabtitle)
    print(old_tab.tabtitle)
    -- 打印一下标签页的标题看看
end
```

8) 示例

```
package.cpath = "?53.dll;" .. package.cpath
--检查所需要的库文件是否存在
```

--require 后的模块名替换'?'，根据替换的结果在设置的目录下或者环境中查找文件是否存在。

require 'iuplua' --加载需要的 IUP 基本函数库模块

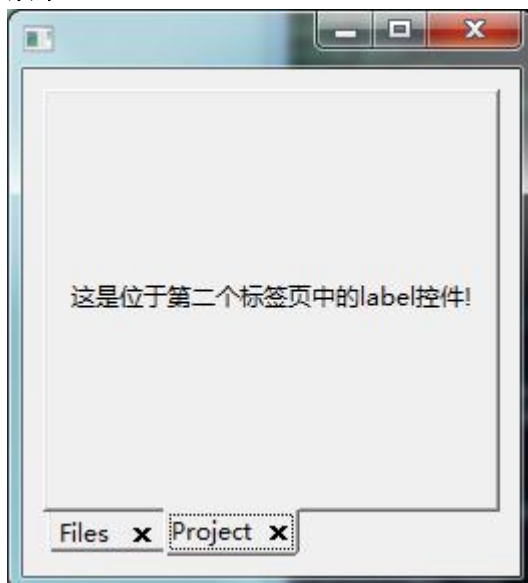
```
lab = iup.label{title = '这是位于第二个标签页中的 label 控件!', expand = 'yes'} vbox1
= iup.vbox{rastersize = '200x200', TABTITLE = 'Files'}; -- 在 vbox 中使用属性 TABTITLE
设置了标签页的标题
```

```
vbox2 = iup.vbox{rastersize = '200x200', lab};
tabs = iup.tabs{ -- 创建了一个管理标签页的控件对象
    vbox1; -- 将
    vbox2;
} -- 创建一个多页的控件对象
tabs.rastersize = '200x200'
tabs.SHOWCLOSE = 'yes' -- 使用可关闭按钮
tabs.TABTITLE1 = 'Project' -- 设置第 2 个标签页的
tabs.TABTYPE = "BOTTOM" -- 将选项卡放在标签页的底部
dlg = iup.dialog{
    iup.vbox{
        tabs;
        margin = '10x10';
    };
}
```

function tabs:tabchange_cb(new_tab, old_tab) -- 回调函数，当手动切换标签页时触发。

```
    print(new_tab.tabtitle)
    print(old_tab.tabtitle)
end
dlg:map()
tabs.VALUEPOS = 1 -- 设置对话框弹出时激活的标签页。
dlg:popup()
```

9) 效果



3.4 平台开发步骤

以上讲的都是 APCAD 开发的基础，有了这些基础知识，我们怎么样在 APCAD 下进行真正的 APP 开发呢？主要分为以下四步：

第一步：在平台路径下，找到入口文件“main.lua”，在此文件中添加用户的 Lua 代码，比如：

代码示例：

```
trace_out("Hello World! ")
```

第二步：在文件“main.lua”，实现系统必须的回调函数，包含以下函数，各自功能描述如下所示：

第三步：按照 Lua 的调用语法，配合平台提供的 API 函数，用户可以用任何符合规范的模式来进行程序的开发了。

第四步：把写好的程序打包成 APCAD 系统支持的格式，上传到服务器中，让用户下载试用。

3.5 平台变量介绍

APCAD 平台主变量是主要关键点，通过这些主变量可以控制平台的显示、操作等。下面介绍以下主要的变量：

1、scene

scene table 是平台的视图属性记录表结构。利用函数 get_scene_t 可以把该表取出来。它包含以下属性 offset、rotate、cen、scale、ortho、clip，记录了视图的偏移、旋转、中心、正交和所在切平面坐标系等属性。

2、frm

frm 是平台的框架句柄变量。

3、ID

ID 是平台的全局标识值。

4、luaaxis

luaaxis 是平台的坐标系变量。它有 base, x, y, z, beta 四个键值。base 表示坐标系的原心。x, y, z 表示坐标系的三个坐标轴向量。beta 表示坐标系的旋转的角度。

5、object

能够显示在平台的实体结构，该实体有以下的键值。

1) index : 是此实体的标识值。

2) localplacement : 是该实体所在的坐标系。该表是 luaaxis 变量。

3) hide : 表示该实体是否是隐藏的。1 表示隐藏，0 表示不隐藏。

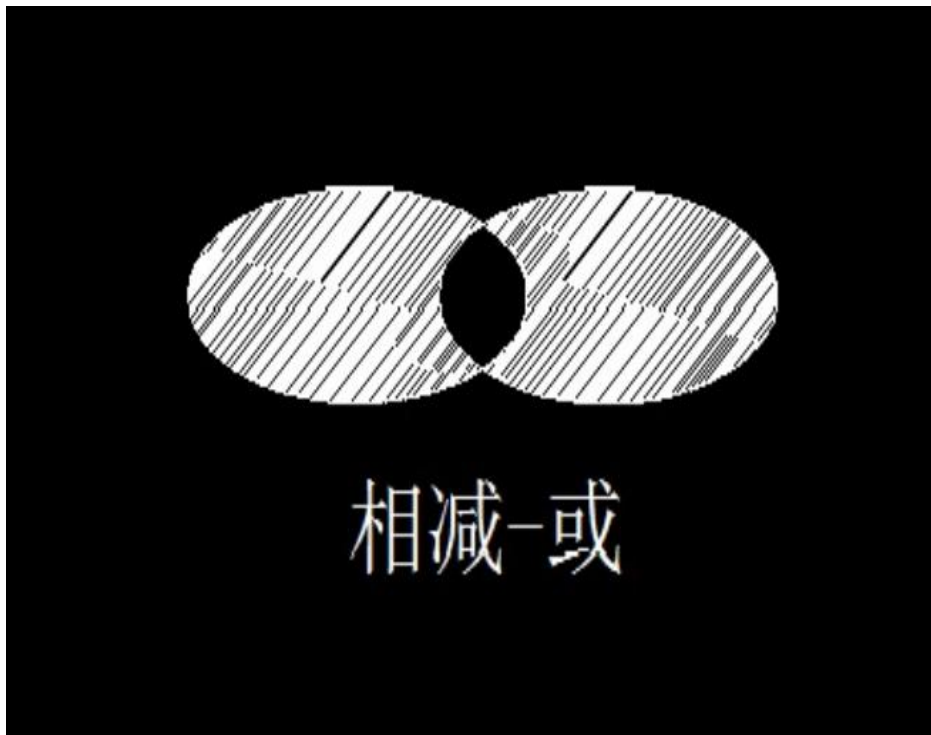
4) composed : 该表表示该实体是由其它两个实体计算得出来的。在绘制该实体时不需要计算 surfaces 中的值。键值 a 表示 a 实体的表。键值 b 表示 b 实体的表；键值 op 表示这两个实体是进行的操作，操作有交、并、或和拷贝。键 op =0 是拷贝，此时只有 a 有值，b 为 nil，该实体是通过 a 复制出的一个实体，如果 localplacement 有值，该实体就是实体 a 按照 localplacement 进行坐标转换得出的结果。键 op =1 是相交，是 a 和 b 相交得到的结果。键 op = 2 是取并集，是 a 和 b 相加得到的结果。键 op = 3 是或计算是 a 和 b 去掉相交的部分得到的结果。如图所示：



相乘-交



相加-并



5) surfaces : 该表描述该实体是由哪些具体的点线面组成。举例如下:

```
surfaces = {
  {points={{r, g, b, u, v, 0, 0, 0}, {r, g, b, u, v, 0, 100, 0}, {r, g, b, u, v, 100, 100, 0}, {r, g, b, u, v, 10, 10, 0}, {r, g, b, u, v, 90, 10, 0}, {r, g, b, u, v, 50, 90, 0}},
  lines={{1, 2}}, triangles={{1, 2, 3}}, quadrangles={{1, 2, 3, 4}},
  pts = {1, 2, 3, }, outer={1, 2, 3}, inners={{4, 5, 6}, {}},
  texts={{ptno=1, r, g, b, str}, {}, {}, },
  {},
},
```

- (1) points : 该表是描述点的集合。每点有 6 个键值: r, g, b, u, v, x, y, z。r, g, b 表示颜色对应的 RGB 值; u, v 表示贴图的位置; x, y, z 表示对应的坐标值。
- (2) lines : 该表是描述线集合。每段线起终点和终止点是上面点集合的 index 值。
- (3) triangles : 该表是描述三角形集合。每个三角形的顶点是上面点集合的 index 值。
- (4) quadrangles : 该表是描述四边形集合。每个四边形的顶点是上面点集合的 index 值。
- (5) pts : 当该实体就是个点时, 此项表示点的位置标示。每个点是上面点集合的 index 值。
- (6) outer : 该表是描述外轮廓点集合。每个图形的顶点是上面点集合的 index 值。
- (7) inners : 该表是描述内轮廓点集合。每个图形的顶点是上面点集合的 index 值。
- (8) texts: 该表是描述文本串集合。每个文本串有五个键值: ptno, r, g, b, str。ptno 表示点集合的 index 值; r, g, b 表示对应 RGB 颜色值大小; str 表示文本的内容

5、luapt

luapt 是平台提供的点类, 包含点的基本属性和对点的操作函数。

luapt 有三个键值: x, y, z。表示三维坐标的值。详细使用说明参见平台 API。

6、trace_out

trace_out 是把信息打印到测试窗口中, 便于用户检查程序错误。

3.6 平台 SDK 介绍

APCAD 平台 SDK 中包含很多内容，主要分为以下几大类：

1、三维绘图

三维绘图接口中包含了平台基本的三维显示接口，可以把实体通过面的形式进行显示，面通过线，线通过点进行数据的描述。同时平台实现了实体的交并或的基本操作，可以绘制出不规则的实体。

2、鼠标响应

此类接口是平台的鼠标回调函数，在函数中接收到鼠标的消息，包含基本的左键、右键的单击、双击消息，以及中键的滚动消息。用户可以在这些消息中实现具体的实际应用。

3、键盘响应

这类接口是平台的键盘回调函数，在函数中接收到键盘的消息，用户可对这些不同键值进行具体的处理。

4、系统的坐标系

描述了模型，在用户建模时需要明确的坐标系统，掌握坐标系统后在进行模型的旋转、编辑时会非常方便。

5、用户界面

用户界面包含了用户能够看见的所有界面接口，主要包含菜单、工具条、工作区、对话框、状态栏、命令行、系统托盘等。用户利用这些接口可实现完全自定义的软件界面，更人性化的界面人机交互。

6、字符转换接口

包含字符格式的各种转换方式，解决中文、日文、英文字体之间的转换问题。

7、多线程处理

在处理文件下载，或者非常费内存的情况下，用户可以用多线程来处理 这样的问题，提高计算机利用效率。

8、ADO 数据库接口

提供跟主流数据库交互的接口，用户利用这些接口可以访问数据库，进行数据的读取、存储和查询。

9、COM 接口

利用 COM 接口可以控制 Word、Excel 等软件，进行数据的导出、表格的定制、文档的生成非常方便。

10、生成 PDF 文件接口

包含字符格式的各种转换方式，解决中文、日文、英文字体之间的转换问题。

11、网络传输接口

利用网络接口，用户可以传输文件，发送消息，发送代码段，因为 Lua 是解释性语言，所以对方接到代码段后可以直接执行。

12、APP 管理接口

在用户完成自己模块以后，可以用这些命令来打包 APP，进行上传后续的 APP 管理。

13、DXF 导入导出接口

第四章 APCAD 用户界面

4.1 用户界面简介

4.2 计算机外设消息相应

4.2.1 鼠标

1、响应左键按下消息:

参数: scene:当前窗口句柄, flags:标志, x/y:屏幕坐标

代码示例:

```
function on_lbuttondown(scene, flags, x, y)
    trace_out("This is a function on_lbuttondown. \n");
end
```

2、响应左键抬起消息:

参数: scene:当前窗口句柄, flags:标志, x/y:屏幕坐标

代码示例:

```
function on_lbuttonup(scene, flags, x, y)
    trace_out("This is a function on_lbuttonup. \n");
end
```

3、响应左键双击消息:

参数: scene:当前窗口句柄, flags:标志, x/y:屏幕坐标

代码示例:

```
function on_lbuttondblclk(scene, flags, x, y)
    trace_out("This is a function on_lbuttondblclk. \n");
end
```

4、响应右键按下消息:

参数: scene:当前窗口句柄, flags:标志, x/y:屏幕坐标

代码示例:

```
function on_rbuttondown(scene, flags, x, y)
    trace_out("This is a function on_rbuttondown. \n");
end
```

5、响应右键抬起消息:

参数: scene:当前窗口句柄, flags:标志, x/y:屏幕坐标

代码示例:

```
function on_rbuttonup(scene, flags, x, y)
    trace_out("This is a function on_rbuttonup. \n");
end
```

6、响应右键双击消息:

参数: scene:当前窗口句柄, flags:标志, x/y:屏幕坐标

代码示例:

```
function on_rbuttondblclk(scene, flags, x, y)
    trace_out("This is a function on_rbuttondblclk. \n");
```

end

7、响应中键按下消息:

参数: scene:当前窗口句柄, flags:标志, x/y:屏幕坐标

代码示例:

```
function on_rbuttondblclk(scene, flags, x, y)  
trace_out("This is a function on_rbuttondblclk. \n");  
end
```

8、响应中键抬起消息:

参数: scene:当前窗口句柄, flags:标志, x/y:屏幕坐标

代码示例:

```
function on_mbuttonup(scene, flags, x, y)  
trace_out("This is a function on_mbuttonup. \n");  
end
```

9、响应中键双击消息:

参数: scene:当前窗口句柄, flags:标志, x/y:屏幕坐标

代码示例:

```
function on_mbuttondblclk(scene, flags, x, y)  
trace_out("This is a function on_mbuttondblclk. \n");  
end
```

10、响应鼠标移动消息:

参数: scene:当前窗口句柄, flags:标志, x/y:屏幕坐标

代码示例:

```
function on_mousemove(scene, flags, x, y)  
trace_out("This is a function on_mousemove. \n ");  
end
```

4.2.2 键盘

响应按键消息:

参数: scene:当前窗口句柄, key:键码。

代码示例:

```
function on_keydown(scene, key)  
trace_out("This is a function on_keydown. \n");  
end
```

4.3 平台界面应用

4.3.1 菜单

平台的菜单是可以全部换成用户自定义的菜单。主要包含以下主要内容。

1、菜单创建

用户可利用平台 API 函数 `add_menu` 生成菜单。函数原型如下:

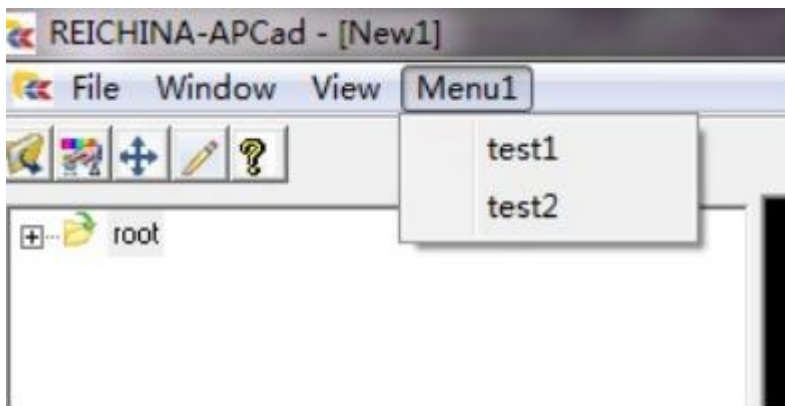
```
add_menu(frm, menu_items)
```

第一个参数 `frm` 是平台框架的句柄变量。

第二个参数是个表，该表有 name, nposition 和 items 三个键值。name 是个字符串，表示该主菜单的名称。nposition 可以设置新加入的 menu 位置，如果不设置，就是相当于 append。items 是表。它的每子项要求有两个键值：id 和 name。该子项的 id 是此菜单的标识值，name 是显示该子菜单的名称。例如：

```
add_menu(frm,
{
  name="menu1",
  items={
    {id=ID_1, name="test1"},
    {id=ID_2, name="test2"},
  }
});
```

结果如图所示：



当用户处理消息时，需要在文件中实现函数 on_command。函数原型如下：

```
on_command(id, scene)
```

函数第一个参数是被点击消息的 id 值，也就是上文中提到的 id 的值。第二个参数是平台的视图句柄变量。用户要获得上文中的消息，需要编辑如下代码：

```
function on_command(id, scene)
  if ia == ID_1 then dosomething() end
end
```

2、已有的菜单中添加子菜单

对于系统菜单，用户可以往菜单中添加需要的子菜单。

```
submenu = get_submenu(frm, 0);
```

利用函数 get_submenu 获取已经生成的菜单，0 表示第一个，依此类推。返回的变量是对应的该菜单，用户可以利用 insert_menu 实现菜单的插入到其中某个位置，示例如下：

```
submenu = get_submenu(frm, 0);
```

3、菜单分隔符的使用

在菜单之间添加分隔符号，加入这样的一个菜单项即可：

```
add_menu(frm, {{ia = IL + 1000, name = "test"},
{ia = 0, name = "null", flags = MF_SEPARATOR},
{ia = IL + 1001, name = "test1"}});
```

4、级联菜单的使用

添加菜单的子项的方法。级联子项的定义如下：

```
subitem_Inquire = sub_menu({ name = "Inquire", items = {{id=
```

```
ID_INQUIRD_USER, name="Users"}, {id=ID_INQUIRD_SPECIFICATION, name="Specifications"}}, }
```

主菜单的定义如下：

```
add_menu(frm, {name = "Tool", items = {{ia = subitem_Inquire, name = "Inquire", flags = MF_POPUP}}, }, );
```

5、获得主菜单的句柄

```
handle = get_mainmenu(frm);
```

6、删除和隐藏菜单的方法

remove_menu 是删除级联菜单中的某一个。del_menu 是删除整个菜单。

```
del_menu(frm, id, position);
```

```
remove_menu(frm, id, position);
```

4.3.2 工具条

用户可利用平台 API 函数 crt_toolbar 生成工具条。函数原型如下：

```
crt_toolbar(frm, toolbar_items)
```

第一个参数 frm 是平台框架的句柄变量。

第二个参数是个表，此表有以下的键值：

id：是此工具条按钮的标识值。

bmpname：该工具条的位图名称。用户可用编辑图片的软件进行编辑。

nbmps：该工具条的位图的数量。

dxButton：该工具条的位图起始点 x 值。

dyButton：该工具条的位图起始点 y 值

dxBitmap：该位图的长度。

dyBitmap：该位图的宽度。

buttons：是个表，它的每个子项有五个键值：

1. iBitmap：该按钮位图的位置个数。

2. idCommand：该按钮被单击时的消息 id 值。

3. iString：该按钮的提示内容。

4. fsState：该按钮的状态。TBSTATE_ENABLED 表示可用的。

5. fsStyle：该按钮的样式。BTNS_BUTTON 表示普通工具条按钮。BTNS_SEP 表示分割条。

当构造完此表，调用 crt_toolbar 函数就会在界面上显示该工具条。例如：

```
package.cpath = "./?53.dll;./?.dll";
```

```
local tool_ty1 = {  
    id = 1;  
    bmpname = "toolbar1.bmp",  
    nbmps = 5,  
    dxButton = 0,  
    dyButton = 0,  
    buttons = {  
        {  
            iBitmap = 0,  
            idCommand = ID_IMPORT_STAAD,  
            iString = "Import Staad",  
            fsState = TBSTATE_ENABLED,  
            fsStyle = BTNS_BUTTON,
```

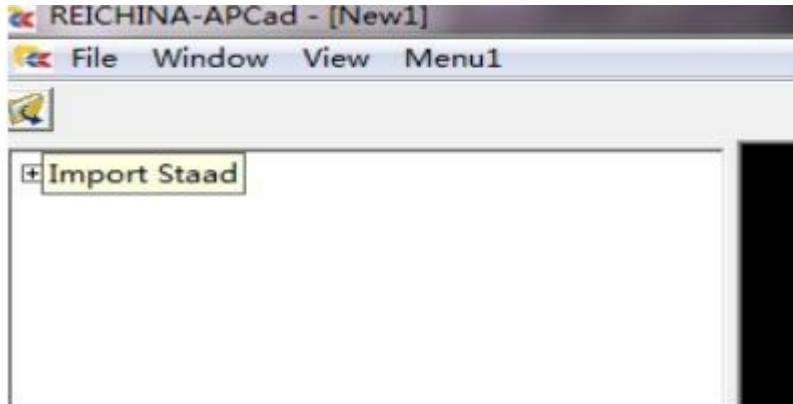


```

    },
};
}
crt_toolbar(frm, tool_ty1)

```

结果如图所示：



当用户处理该工具条的点击消息。同样是利用函数 `on_command`。与上文的菜单方法类似。

4.3.3 工作区

左面工作区实质上是使用平台提供的底层接口加上 IUP 的对话框制作而成。其中底层接口用到了如下函数：

将 IUP 对话框加入到平台工作区：

```
add_dlgtree(frm, hdlgtree)
```

-- `frm` 是平台框架变量，`hdlgtree` 是创建的 iup 对话框（对话框的设计样式请看下面的示例）。

控制工作区的显示或者不显示：

```
dlgtree_show(frm, true);
```

-- `frm` 平台框架变量，第二个参数接收的是布尔值 `true` or `false` 如果为 `true` 则代表显示工作区，为 `false` 则不显示工作区。

平台回调函数：

```
create_dlgtree()
```

-- 可以将创建工作区对话框的代码放入这个函数中供平台调用。

针对工作区的操作，比如右键菜单均是采用 IUP 中的控件操作完成，详细请查阅 IUP 在线帮助；

简单示例如下：

```

package.cpath = "?53.dll;" .. package.cpath
require 'iuplua'
local dlg
local tree = iup.tree{};
local function init_tree_attributes()
    tree.font = "COURIER_NORMAL_10"
    tree.markmode = "MULTIPLE"
    tree.addexpanded = "NO"
    tree.showrename = "YES"
    tree.expand="YES"
end

```

```

function init_tree_nodes()
    tree.name = "Figures"
    tree.addbranch = "3D"
    tree.addbranch = "2D"
    tree.addbranch1 = "parallelogram"
    tree.addleaf2 = "diamond"
    tree.addleaf2 = "square"
    tree.addbranch1 = "triangle"
    tree.addleaf2 = "scalenus"
    tree.addleaf2 = "isocoles"
    tree.value = "6"
end

function create_dlgtree()
    -- Creates text
    text = iup.multiline{expand = "YES"}
    -- Creates boxes
    vboxA = iup.vbox{text}
    init_tree_attributes()
    vboxB = iup.vbox{tree}

    -- Sets titles of the vbboxes
    vboxA.tabtitle = "AA"
    vboxB.tabtitle = "BB"

    -- Creates tabs
    tabs = iup.tabs{vboxA, vboxB; TABTYPE="BOTTOM"} -- 使用 tabs 元素创建标签页。
    dlg = iup.dialog{
        iup.vbox{tabs; margin="5x5"};
        BORDER="NO", -- 下面这些属性控制对话框的样式
        MAXBOX="NO",
        MINBOX="NO",
        MENUBOX="NO",
        CONTROL = "YES",
        size="220X0",
        shrink="YES"
    }
    iup.SetAttribute(dlg, "NATIVEPARENT", frm_hwnd)
    function dlg:show_cb(status)
        if status == 0 then
            hdlgtree = iup.GetAttributeData(dlg, "HWND")
            add_dlgtree(frm, hdlgtree)
        end
    end
    end
    dlg:show()
    init_tree_nodes()

```

end

4.3.4 状态栏

右下角的状态栏使用说明。例如：

```
statusbar_set_parts(frm, {150, 50, 150}) --设置状态栏的高和宽度。  
taskbar_addicon(frm)
```

4.3.5 命令行

函数 `send_gcad` 是向命令行中发送一个文本消息。例如：

```
send_gcad(frm, "from lua");
```

回调函数 `on_gcad_msg` 是从命令行中获得文本字符串。例如：

```
function on_gcad_msg(scene, str)  
dosomething();  
end;
```

4.3.6 系统托盘

当鼠标点击时显示文字的方法如下：

```
function on_lbuttondown()  
statusbar_set_text(frm, 1, "hello world!")  
taskbar_delicon(frm)  
end
```

该程序图标还相应以下函数：

```
function on_umnotify_lbuttondown()  
function on_umnotify_rbuttondown()  
function on_umnotify_lbuttondblclk()  
function on_umnotify_lbuttondown()  
托盘图标的闪烁和停止闪烁  
taskbar_flash(frm)  
taskbar_stopflash(frm)
```

4.3.7 背景渐变

接口函数如下：

```
scene_color(scene, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0)
```

变量 `scene` 后的值，前三个值是左下角的 RGB 值，第二组是右下角的 RGB 值，第三组是右上角的 RGB 值，第四组是左上角的 RGB 值。