

PyS60库 参考手册

1.4.1最终版

原版作者：诺基亚(NOKIA)
中译本作者：杜 俊(shagon)

发行日期：2007年 7月 4日
最后修订：2008年 2月 9日

原著版权所有：诺基亚公司 2004 - 2007

这是诺基亚公司制作的PyS60 1.4.1最终版(基于Python 2.2.2)。
相关内容已得到Apache License 2.0许可。相关资源包含在开源软件包中。

参阅：<http://www.apache.org/licenses/LICENSE-2.0>

<http://www.python.org/2.2.2/license.html>

中译本作者：杜 俊(shagon)

力求严谨，技术名词引用原文；英汉对照，排版严格遵照原版。

时间仓促、水平有限，欢迎广大编程爱好者不吝交流指正。

联系邮箱：shagon@163.com

最新修订请关注：<http://shagon.51.com>

置顶文章

更多信息请访问：<http://www.cnpda.com.cn>

<http://shagon.51.com>

前 言

PyS60是移植到Symbian Series 60平台的Python版本,全称是Python for Series 60。它极大简化了S60应用软件的开发,能够方便地调用Symbian C++ APIs。本参考手册针对PyS60 1.4.1最终版(基于Python 2.2.2)。

目录

1	绪 论	1
1.1	关于内容	1
1.2	关于读者	2
1.3	命名规则	2
2	API概述	3
2.1	Python标准库	3
2.2	PyS60功能扩展	3
2.3	第三方扩展	4
3	PyS60编程值得思考的一些问题	5
3.1	并发性问题	5
3.2	运行PyS60程序	5
3.3	标准输入/输出流	6
3.4	使用Unicode	6
3.5	日期与时间	6
3.6	线程操作的局限性	6
3.7	可缩放用户界面框架	7
3.8	异常处理	7
3.9	目前的局限和未来的发展	7
4	系统级的服务和信息	9
4.1	e32—Symbian系统级的服务模块	9
4.2	sysinfo—获取系统信息	11
5	用户界面及图象处理	13
5.1	appuifw—S60 GUI框架	13
5.2	graphics—图象处理	27
5.3	camera—拍照和录像	33
5.4	keycapture—按键捕获	36
5.5	topwindow—置顶窗体	37
5.6	gles—绑定OpenGL ES	38
5.7	glcanvas—OpenGL ES图形显示控件	45
5.8	sensor—访问手机传感器	46
6	声音和通信服务	49
6.1	audio—声音服务	49
6.2	telephone—电话服务	51
6.3	messaging—信息服务	52
6.4	inbox—信箱接口	53
6.5	location—GSM定位信息	54
6.6	positioning—位置信息的简化接口	55
7	数据管理	59

7.1	contacts—联系人服务	.59
7.2	calendar—日历服务	.64
7.3	calendar(EKA2)—日历服务	.69
7.4	e32db—Symbian数据库接口	.74
7.5	e32dbm—使用Symbian系统DBMS的DBM工具	.77
8	标准库支持及扩展	79
8.1	Python标准库支持	.79
8.2	标准库模块扩展	.80
9	扩展和内嵌	83
9.1	Python/C API 扩展	.83
9.2	Python for S60扩展	.84
10	术语	87
	BUG报告	91
	模块索引	93
	索引	95

绪 论

PyS60系列参考文档包含三部：

- 《Getting Started with Python for S60》[5]介绍如何安装PyS60，并尝试编写程序。
- 《PyS60库参考手册》详细介绍如何调用API及其他资源。
- 《Programming with Python for S60 Platform》[6]包含S60手机的编程范例及源码。

安装在手机上的PyS60平台包含以下部分：

- Python运行环境：
 - Python解释器
 - Python标准库及PyS60专用模块
 - S60用户界面框架
- Python脚本运行器：
 - 提供命令控制台，可运行脚本文件。
 - 安装及运行Python程序(针对S60 3rd之前的机型)：
 - * 识别py、pyc、pyd和pyo文件并安装。
 - * 识别PyS60应用软件并启动Python环境。

同时，也有适用于S60 C++ SDK的PyS60 SDK。

开发者可以在S60模拟器上运行Python程序，并用C++扩展Python模块(PYDs)。

诺基亚论坛[9]包含了丰富的资源和信息，欢迎访问和探讨。

1.1 关于内容

这本参考手册介绍了PyS60应用软件开发的知识，以及一些扩展开发的建议。

1.2 关于读者

这本手册面向S60平台应用程序的开发者。

读者需要对Python语言有一定熟悉(<http://www.python.org/>),并具有PyS60编程的基础。

参阅《Getting Started with Python for S60 Platform》[5]

1.3 命名规则

本手册中大部分命名沿用Symbian SDK的标准。

更多信息请参阅Symbian SDK文档[4]。

API概述

Python语言所有内建对象类型都能被S60平台支持，而众多库模块(library module)可以帮助开发者更加方便地进行设计。本章对这些API作一个概述。

2.1 Python标准库

为了节约手机的存储空间，PyS60并没有包含全部的Python标准库和可选库。事实上，这些没有被加入到PyS60的标准库，很多都能不作修改就应用到PyS60中。另外，PyS60 SDK中具有的一些模块文件，在手机上也没有。可用模块列表详见第八章。

在电脑上安装Python以后，库模块文件也会同时被安装。对于Windows系统，默认路径是‘\Python22\Lib’；对于Linux系统，默认路径是‘/usr/lib/python2.2’。Python可在官网下载：<http://www.python.org/>

参阅《The Python library modules' APIs》[1]

PyS60扩展了一些标准模块的功能，详见章节8.2.

2.2 PyS60功能扩展

可以通过C++扩展PyS60的功能。可以制作两类模块：1. 内建模块 2. 动态加载模块。

2.2.1 内建模块

PyS60有两个内建扩展模块：

- [e32](#) 模块被写入Symbian系统平台的Python解释器。它是Python解释器与Symbian系统平台之间的桥梁，用于调用Python标准库，以及使用Symbian平台的某些服务。
- [appuifw](#) 模块用于实现界面设计。它提供了可视化用户界面框架。

2.2.2 动态加载模块

以下模块可动态加载，用于调用S60平台的相关API。

- [graphics](#)：参阅章节5.2
- [e32db](#)：参阅章节7.4
- [messaging](#)：参阅章节6.3

- `inbox` : 参阅章节6.4
- `location` : 参阅章节6.5
- `sysinfo` : 参阅章节4.2
- `camera` : 参阅章节5.3
- `audio` : 参阅章节6.1
- `telephone` : 参阅章节6.2
- `calendar` : 参阅章节7.2
- `contacts` : 参阅章节7.1
- `keycapture` : 参阅章节5.4
- `topwindow` : 参阅章节5.5
- `gles` : 参阅章节5.6
- `glcanvas` : 参阅章节5.7

shagon注(本书中, 原版未提及而我认为有必要附加的内容都用这个颜色字体特别注明):

通常可以用三种方式加载模块, 下面分别做说明。

例如有一个模块叫做“shagon”, 该模块含有一个函数叫做“dujun()”。

第一种加载方式:

```
import shagon
shagon.dujun()
```

这种方式很好理解, `import`是保留字, 后面跟的是模块名称, 表示加载这个模块。调用模块内的成员时, 写上模块名, 跟一个点号, 再跟上成员名。

第二种加载方式:

```
import shagon as s
s.dujun()
```

上面一句的意思是加载shagon模块并以s命名它。这种方式的好处是方便记忆, 当一个模块名字比较难记时就取一个简单的代称, 使用时直接写这个代称, 而不用写原来的模块名。

第三种加载方式:

```
from shagon import *
dujun()
```

上面一句的意思是从shagon模块中加载所有成员。星号是通配符, 表示全部。同样也可以直接写成员名称, 比如“`from shagon import dujun`”, 表示从shagon模块中加载dujun()这个函数。

一般不建议使用这种加载方式!因为这种方式加载的类、函数等于直接定义在程序中的, 这就存在着严重隐患, 极易造成函数重名。举例来说, 如果一个函数名是“shagon.dujun()”, 那么几乎不可能造成重名, 而如果函数名仅仅是“dujun()”, 那么重名的可能性就非常大了!

2.3 第三方扩展

开发者可以自行编写模块来扩展Python。针对S60平台的Python/C API扩展详见章节9.1。更加深入的探讨详见章节9.2。相关范例请参阅资料[6]。

PyS60编程值得思考的一些问题

以下是精心挑选的典型问题，在开始PyS60编程之前，必须认真思考一下。

3.1 并发性问题

一般运行Python程序时，系统会构建用户界面（User Interface,以下简称UI）程序来启动Python环境。此时UI主线程就是Python解释器的初始化线程，默认为Python主线程。有一个例外的情况是，如果是用e32.start_server启动，则Python主线程将不再是UI线程。

为了产生新的线程，可以调用thread模块，但是这可能带来一些问题。其中最严重的后果是导致主UI线程堆栈过小。另外还可能出现的情况是，在UI激活的状态下，一些后台进程仍然保持运行。但这些进程无法直接操作UI，换句话说就是，无法调用appuifw模块。

因为Symbian系统上Python解释器销毁机制的限制，Python程序的设计有一个特殊的地方，主线程必须是最后一个活跃线程。（shagon注：即必须销毁全部其他线程之后再销毁主线程。）

一种称为“活动对象”（Active Object,以下简称AO）的机制，在Symbian系统上得到了广泛应用。它用于处理系统线程中相关联的、不具优先级的调度，以及基本API的调用。Python程序员可以明显地感受到，并发性问题的困扰主要出现在UI设计的时候。用AO激活UI响应，必须考虑程序逻辑结构的设计。同时需要考虑AO造成的并发事件的影响。调用e32.Ao_lock,或间接调用某些Python API,都能生成AO而将UI主通道阻塞，但此时与UI关联的响应仍然在起作用。

标准库中的thread.lock无法与UI主线程同步，因为它会抢占线程上下文环境，而令UI事件响应挂起。为了解决这个同步问题，特别设计了e32.Ao_lock。它的优点在于能令主线程进入等待，而又不影响UI响应。

PyS60极大地简化了程序员对于AO机制的应用，只需简单的设计就能处理同步问题。在e32模块中，特定的AO是作为可选项提供的。

3.2 运行PyS60程序

有两种方式运行PyS60程序。一是安装独立的PyS60平台的软件，在“功能表”中直接点图标运行。这种情况下，将产生独立的进程来执行Python程序。另一种方式是向手机直接存入或者安装脚本文件，然后启动

脚本运行器来运行该脚本文件。这种方法主要用于调试，或者编制新的脚本程序。更多信息请参阅：《Programming with Python for S60 Platform》[6]

值得一提的是，这个脚本运行器本身就是用Python编写的。它不仅能运行文件夹中的脚本文件，同时也允许用户以“命令控制台”的形式直接运行指令。或许你也发现了，既然运行器本身就是一个脚本程序，那么它就可能和其他脚本程序产生一些冲突，比如自定义函数重名。然而，换个角度来看这也是有益的，因为它会迫使程序员养成良好习惯，避免程序之间的冲突，在设计上多做考虑。

如何针对现有的运行器编写Python程序，请参阅《Programming with Python for S60 Platform》[6] 里面详细介绍了标准输出以及选项菜单等信息。

注 意：与旧版本不同的是，在最新的1.4.1版运行器中，对于appuifw.app.menu, appuifw.app.title, appuifw.app.exit_key_handler, appuifw.app.screen, appuifw.app.body, sys.stderr以及??等，改善了重建机制，无需应用程序自行保存和恢复这些配置。

3.3 标准输入/输出流

在sys模块中，依据C STDLIB的stdio流，定义了标准的Python输入/输出流。与此同时，在e32中也提供了用于Python实现的C STDLIB的输出流，但是仅适用于调试。其中，e32._stdo函数用于实现C STDLIB中stdout和stderr的路径指向。这样一来，就能捕获来自于Python解释器底层的错误信息和异常中断。

3.4 使用Unicode

在编码类库模块中，字符参数和返回类型都未作修改。S60平台支持Unicode编码，也支持无格式的字符。（shagon注：“无格式”是指没有明确指定编码）但是，返回类型只能是Unicode编码，同时UI的文字信息也要求是Unicode编码，否则可能出现乱码。

3.5 日期与时间

PyS60的时间格式类似于Unix，以格林尼治时间1970年1月1日0时0分0秒为原点，用浮点数保存时间值。（shagon注：这和Unix时钟一样。例如：1985年3月1日上午9点30分，在PyS60中表示成：478488600.0 即当前时间减去时间原点的时间间隔，以秒为单位。）

3.6 线程操作的局限性

PyS60调用标准thread模块来生成新线程，但是因为API本身特定的局限性，必须要注意一些问题。一般来说，线程之间无法共享资源。这就意味着，一个线程只能使用它自身分配的资源。

注 意：

- Symbian系统的STDLIB模块支持有一定的局限性(参阅S60 SDK文档[4])。
例如，STDLIB文件描述符无法在线程之间共享，因此也造成PyS60中无法线程共享。
- 在PyS60中，调用socket模块来处理套接字。

警告：如果错误地引用了某个线程中的对象，将有可能造成解释器崩溃！
错误代码一般为“KERN-EXEC 3”

3.7 可缩放用户界面框架

注意：仅适用于S60 2nd FP3 以上版本(shagon注:本书约定,“以上”包含当前数据。例如“1以上”意指“1,2,3,...”。以下不再特别说明。)

S60 2nd FP3 支持可缩放的UI框架。改良的API提供更强大的用户界面、图标装载、屏幕显示等设计。更多内容请参阅章节5.1.8和《Programming with Python for S60 Platform》[6]

3.8 异常处理

一旦程序出错，将被中断执行并返回错误代码。PyS60依据Symbian系统参数，返回错误代码。
如果函数没有返回类型，则返回None。(shagon注:即无任何显示信息，也不传递值)

3.9 目前的局限和未来的发展

对于Symbian系统，某些模块中的定义和真实的系统环境之间是有区别的。例如关于“当前工作目录”这个概念，两者就不一样。

目前PyS60并没有周期性搜集和回收资源的机制，因此必须考虑程序退出后的销毁作业。这一点也影响了涉及开放性资源的错误响应。

为了解决这个困扰，我们可以设法开发出一套周期性的资源回收机制，或者为Python解释器设计更好的销毁机制。

事实上，系统工程师已经在试验阶段成功地将GC模块应用到Symbian系统中。但是目前还没有正式应用，也没有向第三方开发者提供。

系统级的服务和信息

4.1 e32 — Symbian系统级的服务模块

e32模块提供了Symbian系统级的服务，可以实现UI以及标准Python库无法完成的功能。

4.1.1 模块函数

以下函数都定义在e32模块中(同时没有定义在任何类中)：

`ao_yield()`

使高优先级AO进入等待，让步于其他对象的调度。注意，可能造成UI运行在线程上下文环境中。

关于程序调度请参阅S60 SDK文档[4]。

(shaogn注:该函数一个常用的地方是配合键盘事件的响应。例如，要求用户按某个特定的按键之后才能执行后续操作。)

`ao_sleep(interval [, callback])`

延时`interval`秒，但不将其他调度挂起。如果设定了`callback`，则延时结束后会调用`callback`。

更多信息请参阅章节4.1.3的Ao_timer。

(shaogn注:中括号里面是可选项,下同。ao_sleep(延时秒数数值,回调函数(可选))

`ao_callgate(wrapped_callable)`

将`wrapped_callable`封装到返回的`callgate`对象中(可在任何线程中调用)。于是,调用`callgate`的同时也会自动在上下文环境中调用`wrapped_callable`,同时可传递自变量。

这是Symbian系统中典型的AO封装方法。

(shaogn注: ao_callgate(封装对象))

`drive_list()`

返回可用的驱动器列表。(shaogn注:一般有手机闪存盘c、内存缓冲盘d、存储卡e、固件盘z)

`file_copy(target_name, source_name)`

将源文件复制为目标文件。必须是完整路径。(shaogn注: file_copy(目标文件路径, 源文件路径))

`in_emulator()`

如果当前程序是在模拟器上运行则返回1，在手机上运行则返回0。

`set_home_time(time)`

设置手机时间。(详见章节3.5)

`pys60_version`

返回PyS60的版本及附加信息。

示 例：

```
>>> import e32
>>> e32.pys60_version
'1.2 final'
```

`pys60_version_info`

返回一个包含PyS60版本信息的五元组:(主版本号、副版本号、附加版本号、发布类型、序列代号)
除发布类型是字符串外,其他都是整型数据。如果发布类型不是“final”则意味着这是一个还在开发中的版本。例如:PyS60 1.2将返回 (1, 2, 0, 'final', 0)

s60_version_info

返回一个包含PyS60 SDK版本信息的二元组。其中:

- (1, 2) 代表 S60 1st
- (2, 0) 代表 S60 2nd
- (2, 6) 代表 S60 2nd FP2
- (2, 8) 代表 S60 2nd FP3
- (3, 0) 代表 S60 3rd

示 例:

```
>>> import e32
>>> e32.pys60_version
'1.2.0 final'
>>> e32.pys60_version_info
(1, 2, 0, 'final', 0)
>>> e32.s60_version_info
(2, 0)
>>>
```

is_ui_thread()

如果运行在UI线程上下文环境中则返回True,否则返回False。

start_exe(filename, command [,wait])

启动一个Symbian系统的可执行文件,并用其执行某些操作。如果设定了wait,则会激活同步监测用于捕获程序的退出状态。其中,正常退出返回0,异常退出返回2。

(shagon注:start_exe(完整的可执行文件路径(Unicode编码),命令参数,程序退出监测(可选))

start_server(filename)

以独立进程运行Python脚本,作为后台服务程序。需要注意的是,这种情况下无法调用appui fw模块。

(shagon注:start_server(完整的可执行文件路径(Unicode编码))

reset_inactivity()

重置设备空闲时间。与此同时,背景灯将被点亮。

(shagon注:设备空闲是指用户没有任何按键动作。这个函数可以用来保持背景灯长亮。)

inactivity()

返回用户上一次按键动作的时间。

4.1.2 Ao_lock 类

Ao_lock() 类

生成Ao_lock实例。此AO依赖于同步服务。它可以在主线程上运行,但不影响UI事件响应。

如果程序的某个线程处在Ao_lock中,那么它就不能关闭。而且,不得使用多个Ao_lock.wait,否则会引起AssertionError。

Ao_lock实例支持下列方法:

wait()

处在Ao_lock中的线程将被挂起,直到获得释放信号。需要注意,仅支持一个挂起,所以绝对不能递归调用。只有产生lock对象的线程才能调用wait,而进入wait后,其他AO仍处于激活状态,所以UI不会被冻结。然而这就面临一个问题,UI响应可能处在Ao_lock中的线程上下文环境中。这一点必须在设计逻辑结构时考虑。

signal()

发出信号，释放被挂起的线程。

4.1.3 Ao_timer 类

可以说，Ao_timer是对e32.ao_sleep的扩充。因为，在e32.ao_sleep产生的休眠没有结束的情况下用户强行退出程序，可能造成严重的后果。Ao_timer正是为解决这个问题而设计的，它产生的休眠即使没有结束，用户也可以安全地退出程序。

Ao_timer() 类

生成Ao_timer实例。此AO依赖于休眠服务。它可以在主线程上运行，但不影响UI事件响应。程序的某个线程处于Ao_timer休眠时，理论上不应该退出。每个Ao_timer实例只能产生一个休眠。

Ao_timer实例支持下列方法：

after(interval [,callback])

休眠interval秒，不影响程序调度。如果设定了callback，则在休眠结束后调用callback。

(shagon注:after(休眠的秒数数值，回调函数(可选)))

cancel()

将尚未结束的休眠取消。

4.2 sysinfo — 获取系统信息

sysinfo 模块用于获取S60手机的系统信息。

注意 ring_type() 不支持S60 1st 机型。

sysinfo 模块中含有下列函数：

active_profile()

获取当前可用的情景模式。包括：'general','silent','meeting','outdoor','pager','offline','drive','user <profile value>'。返回其中一个代表当前情景模式的字符串。

(shagon注:依次为“标准、静音、会议、户外、寻呼机、离线、驱动器、用户自设”等模式)

battery()

获取当前电池的电量信息。S60 2nd FP1(S60 2.1)之前的手机满格为7，S60 2nd FP1之后的手机满格为100。电量耗尽则为0。对于模拟器，这个数值永远是0。

注意：如果手机正在充电，那么就不能获得正确数值。

display_twips()

获取显示高宽值，以缇(twip)为单位。关于缇的定义参阅章节10。

display_pixels()

获取显示高宽值，以像素为单位。

free_drivespace()

获取驱动器可用空间信息。形如{'u'C':'100'}。注意，盘符后面跟着一个冒号(:)。

imei()

获取手机的IMEI码。对于模拟器，永远是'u'000000000000000'。

`max_ramdrive_size()`
 获取手机支持的内存上限。

`total_ram()`
 获取手机的内存总量。

`free_ram()`
 获取手机的可用内存量。

`total_rom()`
 获取手机的固件存储量。

`ring_type()`
 获取当前响铃类型。不支持S60 1st机型。包括 : 'normal', 'ascending', 'ring_once', 'beep', 'silent'. (shagon注:依次为 “ 连续响铃、渐强、响铃一次、蜂鸣、无声 ”)

`os_version()`
 获取手机的系统版本信息。用三元组表示:(主版本号,副版本号,附加版本号)
 数值范围定义如下¹:

- 主版本号 : 0 - 127
- 副版本号 : 0 - 99
- 附加版本号 : 0 - 32767

`signal_bars()`
 获取当前的网络信号强度。用0 - 7表示:0即无信号,7即信号最强。
 对于模拟器,这个数值永远是0。

`signal_dbm()`
 获取当前的网络信号强度,以dBm为单位。对SDK 2.8以上版本有效。
 对于模拟器,这个数值永远是0。

`sw_version()`
 获取软件的版本信息。形如: u'V 4.09.1 26-02-04 NHL-10 (c) NMP'
 对于模拟器,这个返回信息永远是 u'emulator'.

¹这些数据的描述依据S60 SDK文档[4]中的定义。

用户界面及图象处理

5.1 appuifw — S60 GUI 框架

appuifw 模块提供了S60 UI应用程序框架。图5.1展示了PyS60的UI布局。

注 意：此服务必须运行在主线程上下文环境，更准确地说是UI程序的初始化线程。

5.1.1 appuifw 模块基础

图5.2展示了S60程序的UI设计框架，而图5.3是几种可用的屏幕模式。

UI控件管理着所有的应用程序窗口，这是很显然的。

对于含有多个视图的程序，可以用导航面板的选项卡来切换视图。

对话框比普通UI控件享有更高优先级，这就是说，对话框往往都会置顶。

UI控件是由Python类型构建的，可用的类型有：

- Text
- Listbox
- Canvas

一旦构建了程序主体(*app.body*)，相应的UI控件就会随之出现在屏幕上。

Form 是一种万能的对话框类型。

Content_handler 用于实现UI程序之间的高层共性对话，以期简化MIME类型的操作内容。

下列函数定义了常用的对话框：

- note
- query
- multi_query
- selection_list
- multi_selection_list

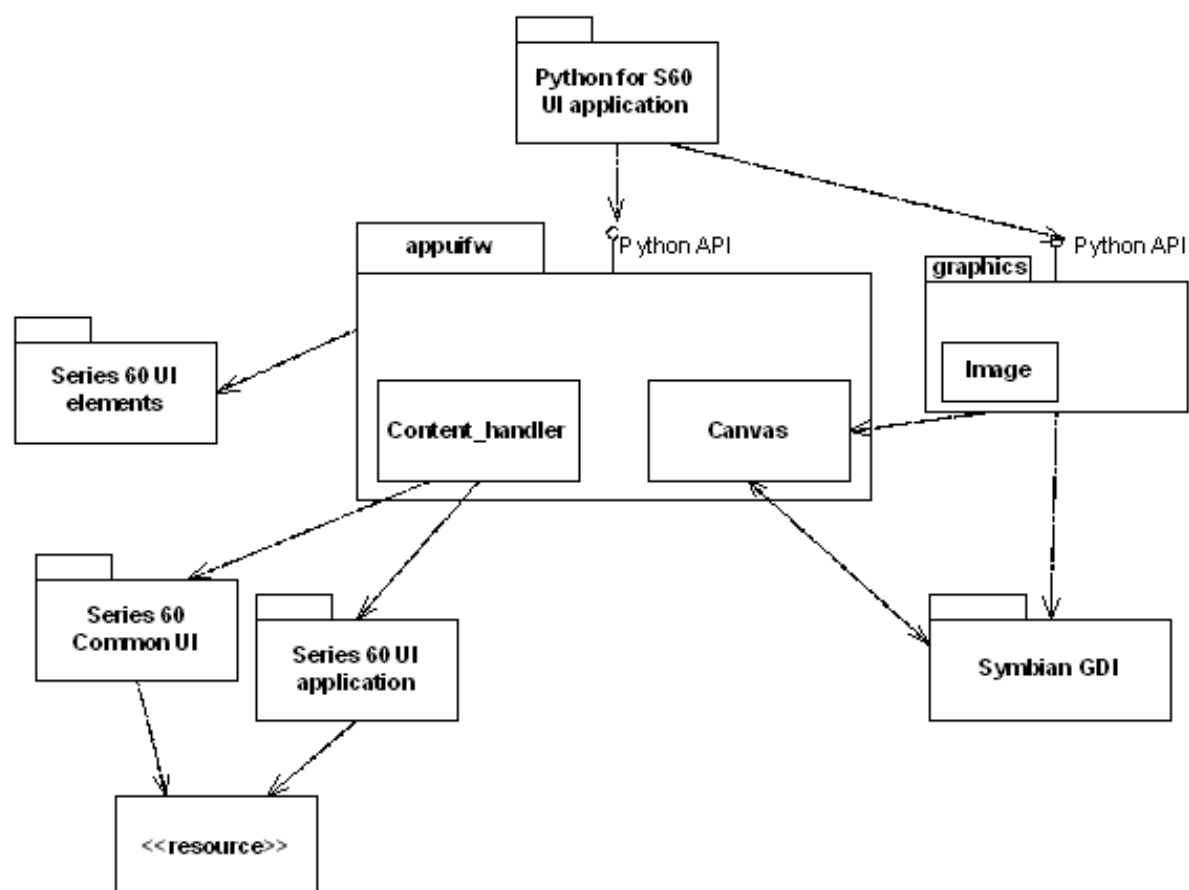


图5.1 PyS60的UI布局

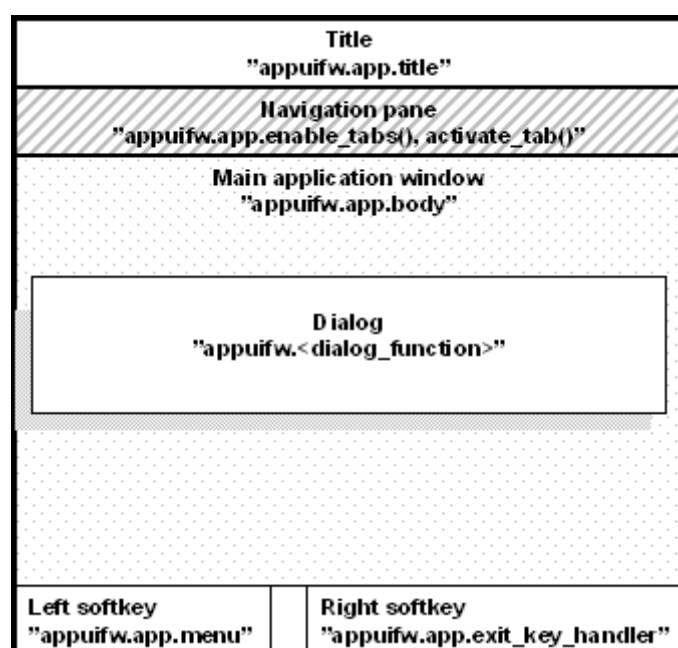


图5.2 PyS60的UI设计框架

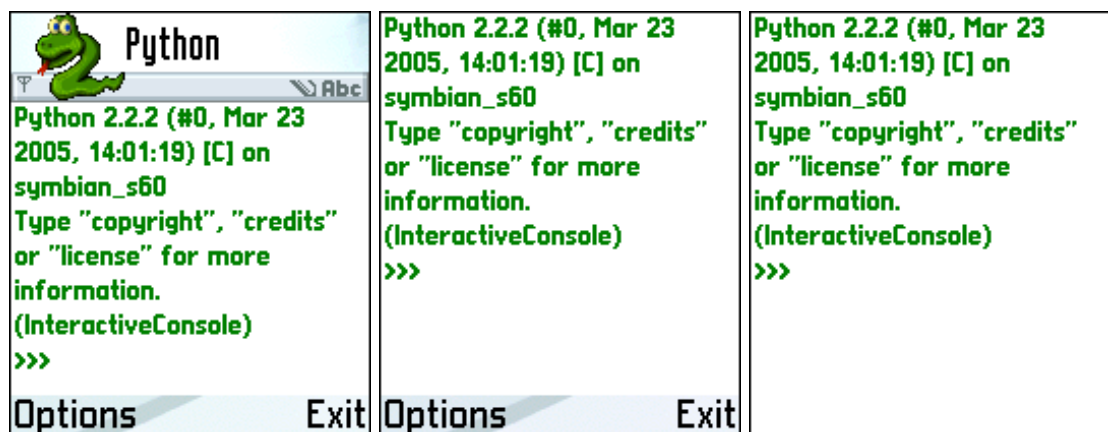


图5.3 UI屏幕模式.从左至右为:标准屏、大屏、全屏

- `popup_menu`

一旦执行相应函数，就会出现置顶的对话框，读取用户输入的信息以及确定或取消等操作。然而Form是特例，它以实例操作，所以只有被执行以后才能产生对话框。

5.1.2 软 键

软键响应默认为S60平台的初始设定。在没有对话框的情况下，按右软键退出，按左软键弹出选项菜单。PyS60中可以非常简单地设计菜单和右软键事件(详见章节5.1.4)。

同样地，软键事件对话框也可以自行设计，比如OK和Cancel(中文机型显示为"确定"和"取消")。但是如果要做独特的对话效果，那么最好用Form。

5.1.3 模块函数

以下函数都定义在`appui fw`模块中(同时没有定义在任何类中)：

`available_fonts()`

获取手机可用的字体列表(Unicode编码)。

`query(label, type[, initial_value])` ([shagon注](#): `query`(提示信息，输入类型，缺省值(可选))

生成一个单域对话框。`label`是需要显示的提示信息，`type`是要求输入的数据类型，有以下几种：

- 'text'
- 'code'
- 'number'
- 'date'
- 'time'
- 'query'
- 'float'

如果设定了`initial_value`，它将作为输入数据的缺省值([shagon注](#): 意思是如果用户不输入任何数据而直接按确定，那么就返回这个值)。它有如下定义：

- 对于文本输入框('text', 'code'), 缺省值是Unicode字符。
- 对于数值输入框('number'), 缺省值是数字。
- 对于日期输入框('date'), 缺省值是上个午夜与时间原点的间隔。

确认对话框('query')和时间输入框('time')都没有缺省值, 它们分别返回True/None和相应的时间数值。对于浮点数输入框('float')即便设定了默认值也无效。所有的对话框, 如果用户取消都返回None。

`multi_query(label_1, label_2)`

生成一个双域文本输入对话框(Unicode编码), 返回二元组(*label_1*, *label_2*)。如果取消则返回None。
(shagon注: `multi_query`(第一项提示信息, 第二项提示信息))

`note(text[, type[, global]])`

生成一个提示对话框。 *text* (Unicode编码) 为要显示的通知信息, *type* 是提示类型, 有 'error', 'info' 和 'conf' (shagon注: 依次为错误通知、信息通知、确认通知)。缺省为 'info'。

如果设定了一个非零整数作为 *global*, 将生成一个全局通知。意思是说即便是后台程序调用也会弹出提示对话框。 *type* 支持的对话框与标准对话框完全一样。

(shagon注: `note`(通知信息, 通知类型 (缺省为 'info'), 全局参数(可选)))

`popup_menu(list[, label])`

生成一个弹出菜单列表。支持一元组或多元组(必须是Unicode编码)。一元菜单直接显示全部项目, 多元菜单根据用户操作显示相应项目。返回选中的项目, 如果取消则返回None。

(shagon注: `popup_menu`(菜单表项, 菜单主标题(可选)))

`selection_list(choices[, search_field=0])`

生成一个选择列表。返回用户选中的项目, 如果取消则返回None。 *choices* 是列表项。 *search_field* 是查找面板参数, 缺省为0即禁用, 1为启用。

(shagon注: `selection_list`(列表表项, 查找面板参数(缺省为0即禁用))。当列表项目太多的时候才适合启用查找面板快速找到选项。但项目很少的情况下应该禁用, 否则反而影响用户视线。)

`multi_selection_list(choices[, style='checkbox', search_field=0])`

生成一个多选列表或可标记列表。返回用户选中的所有项目(一个元组), 如果取消则返回None。

choices 是列表项。 *style* 是列表类型, 有 'checkbox' 和 'checkmark', 分别生成多选列表和可标记列表。缺省为 'checkbox'。在多选列表中直接用导航键选中项目, 而在可标记列表中还需要笔形键的配合。 *search_field* 是查找面板参数, 缺省为0即禁用, 1为启用。列表样式见图5.4。

(shagon注: `multi_selection_list`(列表项, 列表类型(缺省为 'checkbox'), 查找面板(缺省为禁用))。当列表项目太多的时候才适合启用查找面板快速找到选项。但项目很少的情况下应该禁用, 否则反而影响用户视线。)

示 例 :

```
tuple = appuifw.multi_selection_list(L, style='checkmark', search_field=1)
```

5.1.4 Application 类型

事实上, 一旦调用了 `appuifw` 模块, 马上就会生成唯一的 `application` 实例, 不过实例名称被改为 `app`。

Application 类

Application 实例含有以下属性 :



图5.4 左边是多选列表，右边是可标记列表

body

设定程序窗体。目前支持Text, Listbox, Canvas和None。(shagon注:None就是使用缺省窗体)

exit_key_handler

设定右软键响应事件。不设定或设定为None都恢复为缺省设定。

(shagon注:缺省设定为"退出"、"取消"等操作)

menu

设定程序菜单。有两种类型：

- (title, callback) 普通菜单。
- (title, ((title, callback)[...])) 含有子菜单的多级菜单。

title(Unicode编码)是菜单项标题，callback是其响应事件。

菜单项的上限是30个。

示 例：

```
appuifw.app.menu = [(u"Item 1", item1),
                    (u"Submenu 1",
                     ((u"Subitem 1", subitem1),
                      (u"Subitem 2", subitem2))))]
```

screen

设定UI屏幕模式。可选模式有'normal', 'large'和'full'(见图5.3)。

(shagon注：依次为标准屏、大屏、全屏)

示 例：

```
appuifw.app.screen='normal' # (标准屏.显示标题面板和控制面板.)
appuifw.app.screen='large'  # (大屏.显示控制面板.)
appuifw.app.screen='full'   # (全屏.)
```

title

设定标题面板的标题。必须是Unicode编码。(shagon注：只有标准屏模式下才能显示)

focus

程序焦点。参数0表示丢失焦点，1表示恢复焦点。

当程序在前台与后台之间切换，或者进入/退出屏幕保护时，将涉及到焦点的问题。

(shagon注：可以这样理解，如果设定了焦点，那就可以在切换到前台时执行一种操作，切换到后台时执行另一种操作。一般地，按挂机键(红键)就是让程序在后台运行。)

示 例：

```
>>> import appuifw
>>> def cb(fg):
```

```

...     if(fg):
...         print "foreground"
...     else:
...         print "background"
...
>>> appuifw.app.focus=cb
>>> # 程序切换到后台将显示 :
>>> background
>>> # 程序切换到前台将显示 :
>>> foreground

```

注 意：如果焦点事件设定有误,将引起程序错误。例如在诺基亚6600上有可能引起无法停止的 `TypeError`。

`orientation`

设定程序风格(只适用于S60 3rd 机型)。可选项有: 'automatic', 'portrait', 'landscape'。
([shagon注](#):依次为'自动','肖像','风景')。缺省值是 'automatic'。

Application实例支持下列方法：

`activate_tab(index)`

激活选择卡视图。`index` 是选项卡组(从0开始)。

`full_name()`

返回当前在Python解释器里运行的程序名称(Unicode编码)。

`uid()`

返回当前在Python解释器里运行的程序UID(Unicode编码)。

`set_exit()`

在当前操作结束后退出程序。

`set_tabs(tab_texts[,callback=None])`

设定选项卡。`tab_texts` 是一列Unicode字符串。当用导航键在选项卡之间切换,可以变更视图,同时调用相应的 `callback`。如果 `tab_texts` 为空或只有一个元素,上述功能无效。

([shagon注](#):`set_tabs`(选项卡项目,回调函数(可选)))

`layout(layout_id)`

注 意：只适用于 S60 2nd FP3 以上机型。

返回 `layout_id` 项目的大小和位置(二元组)。位置以屏幕左上角为起始点。逻辑布局见图5.2。
`layout_id` 是定义在 `appui fw` 模块中的常量¹:

`EScreen`

屏 幕

`EApplicationWindow`

程序窗体 (占据整个屏幕的窗体)

`EStatusPane`

状态面板 (大部分程序都有状态面板)

`EMainPane`

主面板 (所有程序都有主面板)

`EControlPane`

控制面板

`ESignalPane`

信号面板 (显示信号强度)

`EContextPane`

上下文面板 (显示运行中的程序) ([shagon注](#):通常是显示程序的图标)

`ETitlePane`

标题面板 (显示程序名称或自定义标题)

¹这些数据的描述依据S60 SDK文档[4]中的定义。

EBatteryPane
电池面板（显示电池电量信息）

EUniversalIndicatorPane
通用指示器面板（引起用户的注意）（shagon注：比如屏幕右上角的那个电池图案）

ENaviPane
导航面板（显示应用程序状态、用户输入状态以及其他相关信息）

EFindPane
查找面板（用选择列表取代弹出列表）

EWallpaperPane
壁纸面板

EIndicatorPane
指示器面板（引起用户注意）

EAColumn
显示小图案和小字标题

EBColumn
显示大图标和大字标题

ECColumn
显示用户输入，与EDColumn配合。

EDColumn
显示附加图标，与ECColumn配合。

EStaconTop
landscape的状态面板与控制面板顶部布局

EStaconBottom
landscape的状态面板与控制面板底部布局

EStatusPaneBottom
landscape的状态面板底部布局

EControlPaneBottom
landscape的控制面板底部布局

EControlPaneTop
landscape的状态面板顶部布局

EStatusPaneTop
landscape的控制面板顶部布局

示 例：

```
>>> import appuifw
>>> appuifw.app.layout(appuifw.EMainPane)
((176, 144), (0, 44))
>>> # 诺基亚N70的主面板大小和位置
```

5.1.5 Form类型

Form具有非常大的设计自由度，更通俗点说，它是一个万能对话框。

用它设计出独特的复合对话框，甚至挣脱标准对话框的束缚，做出个性效果。

`Form(fields[, flags=0])` 类

生成Form实例。 *fields* 是域描述列表：`(label, type[, value])`

其中label是Unicode字符串，作为标题名称。type是'text'，'number'，'date'，'time'，'combo'，'float'中的一个，分别对应着：Unicode字符串、普通数字、Unix日期浮点数、Unix时间浮点数、([choice_label ...], index) 以及普通浮点数。

对于'float'，即便设定了缺省值，通常也不会显示在UI中。

Form通过各种属性值来配置对话框。包括域文字信息以及读取用户输入信息。

Form实例含有以下属性：

flags

属性标记。目前可用的标记有：

FFormEditModeOnly

设置Form为可编辑模式。

FFormViewModeOnly

设置Form为不可编辑模式。

FFormAutoLabelEdit

允许终端用户编辑Form的标题项。

FFormAutoFormEdit

允许终端用户增加或删除Form域。

注意，目前这项功能仍处于试验阶段，不支持所有SDK。

FFormDoubleSpaced

生成双行距布局。输入域占据两行，第一行显示项目标题，第二行用于数据处理。

menu

设置Form菜单项 (title, callback)。

其中title是Unicode字符串，用于显示菜单项名称，callback是事件响应。

save_hook

(shagon注：简单一句话，就是用来实现多级连锁对话框。)

生成一个用于接收自变量并返回布尔值的可请求对象，当用户试图保存当前对话框内容时就会调用这个对象。随之返回的布尔值决定了是否在UI中显示新的对话框，新对话框是依据前个对话框内容构建的。如果返回值是False，那么将恢复原对话框。

Form实例支持下列方法：

execute()

执行对话框的生成，使其在UI中可见。

insert(index, field_descriptor)

向Form中插入域描述符（先于index）。

pop()

从Form中移除域描述符并返回这个值。

length()

统计Form中的域描述符总数。

下标 f[i] 用于访问或修改Form f的第i个单元。不过上面已经提到 FFormAutoFormEdit 仍处于试验阶段，也就是对目前来说，要修改运行中的Form单元结果可能不那么理想。

5.1.6 Text 类型

Text是文字处理控件。目前可用的文字样式见图5.5.



图5.5 可用的Text类型样式

Text实例含有以下属性：

color

文本颜色。与graphics模块中的颜色定义相同。详见章节5.2。

focus

控件焦点(布尔值。True:恢复焦点,False:丢失焦点)。

通常编辑器控件需要用到导航栏(就像选项卡一样)，所以同时要使用导航面板控件。

font

文本字体。有两种方法来进行设定：

- 直接定义字体，形如u"Latin12"。如果手机不支持这个字体则定义无效。可以用appuifw.available_fonts来获取手机可用字体列表。

示 例：

```
t = appuifw.Text()
t.font = u"albi17b" # 设为17号粗斜体
t.font = u"LatinPlain12" # 设为拉丁12号常规字体
```

- 定义模式字体。

可选模式有：'annotation'，'title'，'legend'，'symbol'，'dense'，'normal'

(shagon注：依次为注解、标题、图注、记号、密集、常规等模式。)

设定当前字体示例：

```
t.font = "title" # 设为标题型字体
```

校对当前字体示例：

```
unicodeFont = t.font
```

字体属性值必须是Unicode字符串，而且代表手机可用的字体，否则无效。

(shagon注：事实上，第一种方法能够选择全部可用字体，而第二种仅能使用其中挑选出的样式，好处是便于记忆和书写。)

highlight_color

文本加亮颜色。与graphics模块中的颜色定义相同。详见章节5.2

style

文本样式。相关标志属性在appuifw模块中被定义。样式标志可以组合使用，通过“|”符号连接。

样式标志被划分为两类：一种定义文本，另一种定义加亮。

下面分别作说明。

文本样式标志：

STYLE_BOLD
粗体文字

STYLE_UNDERLINE
带下划线文字

STYLE_ITALIC
斜体文字

STYLE_STRIKETHROUGH
带删除线文字

加亮样式标志：

HIGHLIGHT_STANDARD
标准加亮（shagon注：在文字后面生成一个标准矩形加亮框）

HIGHLIGHT_ROUNDED
圆角加亮（shagon注：在文字后面生成一个圆角矩形加亮框）

HIGHLIGHT_SHADOW
阴影加亮（shagon注：生成阴影文字，即复制当前文字用加亮色填充，置于斜后方）

对于一串文字只允许使用一种加亮样式，无论这串文字使用了一种还是多种文字样式。

示 例：

```
t = appuifw.Text()

# 与以下文字样式及其组合设定类似的都有效：
t.style = appuifw.STYLE_BOLD
t.style = appuifw.STYLE_UNDERLINE
t.style = appuifw.STYLE_ITALIC
t.style = appuifw.STYLE_STRIKETHROUGH
t.style = (appuifw.STYLE_BOLD|
           appuifw.STYLE_ITALIC|
           appuifw.STYLE_UNDERLINE)

# 与以下加亮样式设定类似的都有效：
t.style = appuifw.HIGHLIGHT_STANDARD
t.style = appuifw.HIGHLIGHT_ROUNDED
t.style = appuifw.HIGHLIGHT_SHADOW

# 下面这个加亮样式组合是无效的，请勿尝试：
t.style = (appuifw.HIGHLIGHT_SHADOW|appuifw.HIGHLIGHT_ROUNDED)
```

Text实例支持下列方法：

`add(text)`

在光标处插入文字。（shagon注：add(Unicode字符串)）

`bind(event_code, callback)`

设置按键响应（按 *event_code* 对应键就会调用 *callback*）。

键位在key_codes模块中被定义。特别地，bind(*event_code*, None)用于清除已设键位的事件响应。

当然，事实上按键事件最终还是被UI控件所管理的。

`clear()`

清除所编辑的内容。

`delete([pos=0, length=len()])`

从光标 *pos* 处开始删除长度为 *length* 的文字。

`get_pos()`

返回当前光标位置。

`len()`

返回编辑中的文字长度。

```
get([pos=0, length=len()])
```

检索从光标 *pos* 处开始长度为 *length* 的文字。

```
set(text)
```

置入文字 *text* (Unicode)。

```
set_pos(cursor_pos)
```

将光标移动到 *cursor_pos* 位置。

5.1.7 Listbox 类型



图5.6 带图标的列表

Listbox实例是一个列表。正如在Symbian中常见的那样，它既可以是单行距布局，也可以是双行距布局。图5.6就是一个带图标的单行距列表。要使用这样的图标需要MBM或MIF格式文件，详见章节5.1.8。

Listbox(*list*, *callback*) 类

生成Listbox实例。 *list* 是表单项目， *callback* 是选项相应的事件响应。其中 *list* 可以有如下几种：

- 单行距列表。形如 [item1,item2]
- 双行距列表。形如 [(item1,item1description),(item2,item2description)]
(shagon注：(项目名称,项目描述))
- 带图形的单行距列表。形如 [(item1,icon1),(item2,icon2)]
(shagon注：(项目名称,图形描述))
- 带图形的双行距列表。形如 [(item1,item1description,icon1),
(item2,item2description,icon2)]
(shagon注：(项目名称,项目描述,图形描述))

示 例：生成一个带图标的单行距列表——

```
icon1 = appuifw.Icon(u"z:\\system\\data\\avkon.mbm", 28, 29)
icon2 = appuifw.Icon(u"z:\\system\\data\\avkon.mbm", 40, 41)
entries = [(u"Signal", icon1),
           (u"Battery", icon2)]
lb = appuifw.Listbox(entries, lbx_observe)
```

Listbox实例具有下列方法和特性：

`bind(event_code, callback)`

设置按键响应（按 *event_code* 对应键就会调用 *callback*）。

键位在 `key_codes` 模块中被定义。特别地，`bind(event_code, None)` 用于清除已设键位的事件响应。当然，事实上按键事件最终还是被 UI 控件所管理的。

`current()`

返回当前项目的索引值。

`set_list(list[, current])`

设定列表。其中 *list* 的定义与 `Listbox(list, callback)` 中一样。可选项 *current* 指定列表的焦点项目，其值是焦点项目的索引值。

`size`

获取列表大小(宽度,高度)。只支持 S60 3rd 以上机型。

`position`

获取列表位置(左上角坐标)。只支持 S60 3rd 以上机型。

5.1.8 Icon 类型

Icon 实例用于实现在列表中使用图标。详见章节 5.1.7。

Symbian 有一种特殊的压缩图片格式叫做 MBM，它包含了多副位图，通过编号调用所需图片。通常 MBM 文件还有一个后缀名为 *mbg* 的页眉文件，用于描述每个位图。比如 ‘*avkon.mbm*’ 就有 ‘*avkon.mbg*’（在 SDK 中能找到它们）。更多信息请参阅 SDK 文档[4]（“How to provide Icons”）。同时可以了解一下如何将位图集合起来转换成 MBM 文件，SDK 提供了这种工具。

S60 2nd FP3 引入了一种全新的格式，叫做 MIF (Multi-Image File，多图文件)。它与 MBM 类似，将多个文件压缩并保存为单一文件。注意，只有符合 SVG-T 标准的文件才能被压缩（SVG-T：Scalable Vector Graphics Tiny，可缩放矢量图形标准简化版），参阅[10]。

`Icon(filename, bitmap, bitmapMask)` 类

创建图标。*filename* 是完整路径，只能是 MBM 或 MIF 文件（仅 S60 2nd FP3 支持 MIF）。

bitmap 是位图，*bitmapMask* 是遮照，都取其索引值。

示 例：以下创建一个标准的信号符号图标

```
icon = appuifw.Icon(u"z:\\system\\data\\avkon.mbm", 28, 29)
```

5.1.9 Content_handler 类型

`Content_handler` 实例根据 MIME 类型处理数据内容。

`Content_handler([callback])` 类

生成 `Content_handler` 实例。如果同时设定了可选项 *callback*，那么当处理程序结束时将调用这个 *callback*。

`Content_handler` 实例支持下列方法：

`open(filename)`

在处理程序中打开一个已注册为 MIME 类型的文件。该程序将被嵌入到主叫线程中。

程序结束时将调用 *callback*（如果已设）。

`open_standalone(filename)`

在处理程序中打开一个已注册为MIME类型的文件。该程序将在独立的进程中运行。

注意，在这种情况下不会调用 *callback*（不论是否已设）。

5.1.10 Canvas 类型

Canvas是用于处理按键事件和屏幕绘制的UI控件。关于屏幕绘制详见章节5.2。

`Canvas([redraw_callback=None, event_callback=None, resize_callback=None])` 类

创建Canvas。可选项是按键事件响应描述，按下指定按键就会做出相应的事件响应。

注意：避免死循环！比如 *callback* 直接或间接指向了Canvas本身就可能造成无尽的环路调用。

redraw_callback 在需要重绘屏幕的时候就自动调用。这应该很好理解，比如用户临时运行其他软件然后又返回，或者菜单弹出以后又消失，这些情况都需要重绘屏幕。一般来说，可以根据左上角位置和右下角位置来确定重绘的区块范围，不过大多数情况下还是习惯于直接重绘整个Canvas。

event_callback 将在用户操作按键时被调用。按键事件一共有三种：EEventKeyDown, EEventKey, EEventKeyUp（*shagon*注：分别是按下、按住、放开）。当用户按下按键时将发生EEventKeyDown和EEventKey事件，而当用户放开按键时将发生EEventKeyUp事件。

event_callback 的参数是一个字典(关联数组)，它包含以下数据：

- 'type': EEventKeyDown, EEventKey 和 EEventKeyUp 中的一个。
- 'keycode': 按键的keycode。
- 'scancode': 按键的scancode。
- 'modifiers': 针对当前按键事件的操作描述。

键盘上的每个按键都有一个以上的scancode，与它对应的可能没有keycode，也可能有多个keycode。这是因为scancode对应的是手机的物理按键，而keycode是操作系统针对物理按键生成的特定编码。（*shagon*注：可以更简单地理解，对于同型号手机，物理按键是一样的，所以scancode是一样的。但是手机系统可能是大陆行货(简体中文)、港版(繁体中文)或者欧版(英文)等等，它们的字体集是不同的，所以keycode是不同的。）例如，用诺基亚无线键盘（SU-8W）按下A键，scancode是65，而keycode可能是65也可能是91（大写'A'的ASCII编码是65，小写'a'的ASCII编码是91），不论是否按住Shift键或已按下Caps Lock键。

在 `key_codes` 模块中定义了scancode和keycode。详见图5.7。

以下按键有特殊规定：

- 轻按编辑键(也称笔形键)将产生特殊的粘滞效果。（*shagon*注：编辑键通常用于组合按键，所以必须维持状态，即放开时不会立刻产生EEventKeyUp。但如果长按编辑键则无此效果。）
- 电源键和声控标签键无法进行检测。（*shagon*注：待机状态下长按右软键就启动声控标签）
- 每当按下右软键，就会调用 `appuifw.app.exit_key_handler`。（*shagon*注：不论你是否重新定义了右软键事件响应）



编号	Keycode	Scancode	shagon注
1.	EKeyLeftSoftkey	EScancodeLeftSoftkey	左软键
2.	EKeyYes	EScancodeYes	应答键
3.	EKeyMenu	EScancodeMenu	菜单键
4.	EKey0...9	EScancode0...9	数字键
5.	EKeyStar	EScancodeStar	星号键
6.	EKeyLeftArrow	EScancodeLeftArrow	导航左键
7.	EKeyUpArrow	EScancodeUpArrow	导航上键
8.	EKeySelect	EScancodeSelect	导航中键
9.	EKeyRightArrow	EScancodeRightArrow	导航右键
10.	EKeyDownArrow	EScancodeDownArrow	导航下键
11.	EKeyRightSoftkey	EScancodeRightSoftkey	右软键
12.	EKeyNo	EScancodeNo	挂机键
13.	EKeyBackspace	EScancodeBackspace	删除键
14.	EKeyEdit	EScancodeEdit	编辑键
15.	EKeyHash	EScancodeHash	井号键

图5.7 Keycode和Scancode的键位定义

其中，挂机键、菜单键、电源键以及声控标签键都有特殊功能，强制性定义，不得更改。

resize_callback 将在窗体大小改变时被调用。根据二元组变量(宽度值,高度值)来新建窗体。

Canvas实例含有以下属性：

size 一个二元组:(宽度值,高度值)。表示当前Canvas的大小。

Canvas实例提供了标准的绘图方法，详见章节5.2。

5.1.11 InfoPopup 类型

注 意：只适用于S60 3rd以上机型。

InfoPopup实例是一个用于消息提示的UI小控件。比如它可以产生置顶的帮助信息，告诉用户如何操作。当用户按下任意键或者到了指定时间，这条消息就会自行消失。

InfoPopup() 类

生成一个InfoPopup。

`show(text, [(x_coord, y_coord), time_shown, time_before, alignment])`

text 是显示在InfoPopup中的文字信息。可选项中, (*x_coord*, *y_coord*) 指定InfoPopup的显示位置, *time_shown* 指定显示时长(以毫秒计), *time_before* 指定弹出延时(以毫秒计), *alignment* 指定窗体调整方式。

缺省情况位置是(0, 0), *time_shown* 5秒, *time_before* 0秒, *alignment* 指定为 `appuifw.EHLeftVTop`

alignment 可用值如下, 它们都被定义在 `appuifw` 模块中²:

`EHLeftVTop`

水平方向左对齐, 垂直方向顶端对齐

`EHLeftVCenter`

水平方向左对齐, 垂直方向居中

`EHLeftVBottom`

水平方向左对齐, 垂直方向底端对齐

`EHCenterVTop`

水平方向居中, 垂直方向顶端对齐

`EHCenterVCenter`

水平方向居中, 垂直方向居中

`EHCenterVBottom`

水平方向居中, 垂直方向底端对齐

`EHRightVTop`

水平方向右对齐, 垂直方向顶端对齐

`EHRightVCenter`

水平方向右对齐, 垂直方向居中

`EHRightVBottom`

水平方向右对齐, 垂直方向底端对齐

`hide()`

立即隐藏弹出窗体。

示 例 :

```
>>> import appuifw
>>> i=appuifw.InfoPopup()
>>> i.show(u"Here is the tip.", (0, 0), 5000, 0, appuifw.EHRightVCenter)
>>>
```

5.2 graphics — 图象处理

`graphics` 模块提供了Symbian系统中与图象处理有关的操作, 包括打开、保存、旋转以及调整大小。

本模块同时支持前台和后台程序。但对于后者有一定限制, 因为后台进程无法访问UI (详见章节5.2.4)。

相关范例参见资料[6]。

注意, S60 1st 机型不支持本模块中的 `Image.open` 和 `Image.inspect`, 也不支持 `load`, `save`, `resize` 和 `transpose`。

²这些数据的描述依据S60 SDK文档[4]中的定义。

5.2.1 模块函数

以下函数都定义在 `graphics` 模块中(同时没有定义在任何类中)：

`screenshot()`

截获屏幕快照，并将图象保存为 `Image` 格式。

5.2.2 `Image`类静态方法

以下`Image`类静态方法都定义在`graphics`模块中：

`Image.new(size[, mode='RGB16'])`

根据给定大小和模式创建 `Image`对象。`size`是一个二元组，指定大小。`mode`用于指定色彩模式。

可用的色彩模式有：

- '1': 黑白 (1位)
- 'L': 256色 (8位)
- 'RGB12': 4096色 (12位)
- 'RGB16': 65536色 (16位)
- 'RGB': 16777216色 (24位)

`Image.open(filename)`

注意：不支持 S60 1st 机型

在 `Image` 对象中(RGB16模式)打开一张 JPEG 或 PNG 图片。

`filename` 必须是完整路径。

`Image.inspect(filename)`

注意：不支持 S60 1st 机型

检测图片文件。目前只能检测出图象大小(像素值)，返回一个字典。

`filename` 必须是完整路径。

5.2.3 `Image`对象

`Image`对象事实上就是内存中的位图。

注意一个问题 `resize`, `transpose`, `save` 和 `load` 都有一个可选的 `callback`。缺省情况下是同步的,也就是说,只有当操作结束时才返回。如果设定了 `callback`,那么就造成了异步的情况,只要参数可用且操作可行就立即返回。异步时通过后台监测执行情况,操作结束返回一个校验码。校验码为0表示操作成功,否则表示有错误发生。

有必要说明一点,即便是不完整的图片也是可以使用的,但是显然地,显示出来的是残缺的图象。

`Image`对象支持下列方法：

`resize(newsize[, callback=None, keepaspect=0])`

注意：不支持 S60 1st 机型

调整图象大小为 `newsize` , 如果 `keepaspect` 设定为1将维持高宽比,否则直接使用新的高度和宽度。

如果设定了 `callback` , 将以异步方式运行。

`transpose(direction[, callback=None])`

注 意：不支持 S60 1st 机型

旋转图象。 *direction* 是旋转方式，可用参数如下：

- `FLIP_LEFT_RIGHT`: 左右对调
- `FLIP_TOP_BOTTOM`: 上下对调
- `ROTATE_90`: 逆时针旋转90度
- `ROTATE_180`: 逆时针旋转180度
- `ROTATE_270`: 逆时针旋转270度

如果设定了 *callback*，将以异步方式运行。

`load(filename[, callback=None])`

注 意：不支持 S60 1st 机型

打开新的图片替换当前的 Image 对象。维持原先的色彩模式，并且图象大小要求相同。

filename 必须是完整路径。如果设定了 *callback*，将以异步方式运行。

`save(filename[, callback=None, format=None, quality=75, bpp=24, compression='default'])`

注 意：不支持 S60 1st 机型

保存图片为 JPEG 或 PNG 格式。设定 *format* 为 `'jpg'` 或 `'jpeg'` 将保存为 JPEG 图片，设定为 `'png'` 将保存为 PNG 图片。如果 *format* 未设定或者设定为 `None`，则默认保存为 JPEG 图片。

filename 是图片保存路径，必须是完整路径。

如果保存为 JPEG 图片，还需设定图片质量，*quality* 可设为 1 到 100 的某个数字。

如果保存为 PNG 图片，还需设定色深和压缩率，*bpp* 代表色深，*compression* 代表压缩率。

bpp 可用值有：

- 1: 黑白（1位）
- 8: 256色（8位）
- 24: 16777216色（24位）

compression 可用值有：

- `'best'`: 最佳压缩率，存储速度最慢
- `'fast'`: 普通压缩率，存储速度最快
- `'no'`: 不压缩，文件将非常大
- `'default'`: 默认，压缩率和存储速度达到最佳平衡

如果设定了 *callback*，将以异步方式运行。

`stop()`

停止进程中的异步操作。如果进程中不存在异步操作则无效。

Image 对象含有以下属性：

`size`

Image 的大小，以二元组保存。只读。

5.2.4 可绘对象的一般特性

就现在而言，所谓“可绘对象”其实只有 `appuifw` 模块的 `Canvas` 和 `graphics` 模块的 `Image` 两种。本章节重点讲解绘图的一般方法。

选项

大多数方法都支持标准的选项，包括：

- *outline*: 图形或文字的轮廓线颜色。缺省时无轮廓线。
- *fill*: 填充颜色。缺省时无填充。如果同时设定了 *pattern* ,那么将填充白色模式的区域。
- *width*: 图形轮廓线的线条粗细。
- *pattern*: 填充模式。必须是 `None` 或者黑白图(1位)。

坐标表示

可以用序列的形式表示坐标。坐标值可以是 `int`, `long` 或 `float` ([shagon注：整型、长整型、浮点型](#))。有效的坐标序列必须是一个非空序列，允许有两种形式：

- 交替出现 `x`、`y` 值,代表横坐标与纵坐标。这种情况下序列必须含有偶数个元素。
- 成对地出现 `x`、`y`,代表横坐标与纵坐标。

有效坐标序列示例：

- `(1, 221L, 3, 4, 5.85, -3)`: 序列中含有三个坐标
- `[(1,221L),(3,4),[5.12,6]]`: 序列中含有三个坐标
- `(1,5)`: 序列中含有一个坐标
- `[(1,5)]`: 序列中含有一个坐标
- `[[1,5]]`: 序列中含有一个坐标

无效坐标序列示例：

无效代码，请勿使用！

- `[]`: 序列为空
- `(1,2,3)`: 序列中含有奇数个元素
- `[(1,2),(3,4),None]`: 序列中含有无效元素
- `([1,2],3,4)`: 混合序列

颜色表示

所有方法都支持两种颜色表示形式：

- 三元组形式。包含三个0到255的整数，分别代表红、绿、蓝。
- 0xrrggbb形式。其中rr代表红色，gg代表绿色，bb代表蓝色，都是整数。

对于16位和12位色彩，将自动缩减色深；对于8位色彩，根据公式 $(2*r+5*g+b)/8$ 取整；对于黑白，根据公式 $(2*r+5*g+b)/1024$ 取0(黑)或1(白)。

有效颜色示例：

- 0xffff00: 嫩 黄
- 0x004000: 暗 绿
- (255,0,0): 鲜 红
- 0: 纯 黑
- 255: 深 蓝
- (128,128,128): 中 灰

无效颜色示例：

无效代码，请勿使用！

- (0,0.5,0.9): 不支持浮点型
- '#ff80c0': 不支持HTML格式
- (-1,0,1000): 数值越界
- (1,2): 缺少元素
- [128,128,192]: 无效形式

字体定义

可以通过下列方式定义字体：

- None, 使用默认字体
- 直接定义字体，形如 u'LatinBold19'
- 定义模式字体(即UI可用的模式字体，详见章节5.1.6)。
- 以二元组(或三元组)定义字体：
 - 第一个元素是字体名称（None即默认字体）
 - 第二个元素是字体大小（None即默认大小）
 - 第三个元素可选，设定样式(None即默认样式)

有以下样式标志：

- FONT_BOLD 粗 体

- FONT_ITALIC 斜 体
- FONT_SUBSCRIPT 下 标
- FONT_SUPERSCRIPT 上 标
- FONT_ANTIALIAS 平 滑
- FONT_NO_ANTIALIAS 不平滑

样式标志可以组合使用，通过“|”符号连接。比如 FONT_BOLD|FONT_ITALIC代表粗斜体。

注 意：平滑效果只适用于可缩放字体。

可以用`appuifw.available_fonts`来获取手机可用字体列表。

UI可用的模式字体有：

- 'normal' shagon注 常 规
- 'dense' 密 集
- 'title' 标 题
- 'symbol' 记 号
- 'legend' 图 注
- 'annotation' 注 解

对于后台进程，因为它们无法访问UI，所以不能使用模式字体，只能直接定义字体。

可绘对象的通用方法

`line(coordseq[, <options>])`

根据`coordseq`中的坐标点画线段。可选项 *options* 参阅前文“选项”的介绍。

(shagon注:如果给定两个以上的坐标点,将画出一条连续的折线。线段由前后两个坐标点确定)

`polygon(coordseq[, <options>])`

根据`coordseq`中的坐标点画线段,并连接首尾两个坐标点。可选项 *options* 参阅前文“选项”的介绍。
如果设定了 *fill* 或 *pattern*,将进行填充。

(shagon注:就是在line的基础上将首尾也连接起来,于是就形成了一个封闭的图形,可以填充颜色)

`rectangle(coordseq[, <options>])`

根据`coordseq`中的坐标点画矩形。每两个坐标点为一组,分别指定左上角顶点和右下角顶点。
注意,坐标点个数必须是偶数个。可选项 *options* 参阅前文“选项”的介绍。

`ellipse(coordseq[, <options>])`

根据`coordseq`中的坐标点画椭圆。每两个坐标点为一组,分别指定外切矩形左上角顶点和右下角顶点。
注意,坐标点个数必须是偶数个。可选项 *options* 参阅前文“选项”的介绍。

(shagon注:这里椭圆是根据外切矩形画的,而不是根据圆心和长短轴)

`pieslice(coordseq, start, end[, <options>])`

画一个扇形切片,该切片从`coordseq`中坐标点限定的椭圆中截取。椭圆的构建与`ellipse`中定义相同。
start 和 *end* 都是浮点数,分别指定切片的两条截线的方向角度。可选项 *options* 参阅前文“选项”的介绍。

(shagon注:这里以弧度制计算。0是水平向右, /2(约1.57)是垂直向上, (约3.14)是水平向左, - /2(约-1.57)是垂直向下。)

`arc(coordseq, start, end[, <options>])`

画一段弧线，该弧线从`coordseq`中坐标点限定的椭圆中截取。椭圆的构建与`ellipse`中定义相同。
`start`和`end`都是浮点数，分别指定弧线的两条截线的方向角度。可选项`options`参阅前文“选项”的介绍。

(shagon注：这里以弧度制计算。0是水平向右， $\pi/2$ (约1.57)是垂直向上， π (约3.14)是水平向左， $3\pi/2$ (约-1.57)是垂直向下。)

`point(coordseq[, <options>])`

根据`coordseq`中的坐标画点。可选项`options`参阅前文“选项”的介绍。如果`width`设定为大于1的数，那么将在所绘制的点外面加一层指定宽度的轮廓。注意，如果数值过大那么轮廓形状将可能不规则，要产生规则形状建议使用`ellipse`方法。

`clear([color=0xffffffff])`

将可绘对象的表面设置为指定颜色。缺省情况下为白色。

(shagon注：通常用这个方法清屏或设置背景色。)

`text(coordseq, text[fill=0, font=None])`

在`coordseq`中的坐标位置显示文字`text`。`fill`指定文字颜色(缺省为黑色)，`font`指定文字字体。

`measure_text(text[font=None, maxwidth=-1, maxadvance=-1])`

测量某字体文字的大小。在可选项中可以指定最大文字宽度和最大光标移动范围(均为像素值)。

返回一个三元组，内含：

- 文本框大小(左上横坐标,左上纵坐标,右下横坐标,右下纵坐标)
- 光标跨度(从文本起始处移动到末尾的距离)
- 最适宜文字个数(在`maxwidth`和`maxadvance`的限定下达到最佳显示效果的文本长度)

`blit(image[,target=(0,0), source=((0,0),image.size), mask=None, scale=0])`

复制源图象`image`到可绘对象。

可选项中，`target`和`source`分别指定目标图象和源图象的区域。可以只限定一个坐标点，即区域的左上角顶点；也可以限定两个坐标点，分别是区域的左上角顶点和右下角顶点。

`mask`用于设定遮照效果。缺省或设为`None`则无遮照。若设为黑白蒙板(1位)那么被黑色遮照的部分将透明。对于S60 2nd FP2 以上机型还允许使用灰度蒙板，那样就能产生部分透明的效果。

`scale`用于设定缩放效果。若设为0则不缩放，这意味着一旦目标区域比源区域小就会把多余的部分裁减掉；若设为非0的数则自动缩放，以适应大小。

注意一个问题，缩放图象非常耗时，如果要多次使用同一副图象，最好的办法是直接生成一个新的已经缩放到合适大小的`image`，以后就直接调用这个新的`image`。另外还有一点，`blit`方法产生的缩放是不抗锯齿的，可能造成变形。在这一点上用`resize`方法效果会更好。

5.3 camera — 拍照和录像

注 意：不支持S60 1st 机型

`camera`模块用于实现摄像头的基本功能，即拍摄照片和录制录像。

`camera`模块中的状态数据：

EOpenComplete

打开完毕 (shagon注:成功打开了一段视频并等待播放。)

ERecordComplete

拍摄完毕 (shagon注:调用stop_recording()不会立即出现这个状态,因为还有一个保存的过程。)

EPrepareComplete

准备就绪 (shagon注:摄像头初始化完成,等待拍摄。)

camera模块中的函数³:

cameras_available()

返回手机中可用的摄像头个数。

image_modes()

返回手机摄像头支持的图象格式。如:['RGB12','RGB','RGB16','JPEG_JFIF']

image_sizes()

返回手机摄像头支持的图象大小。如:[(640, 480), (160, 120)] (shagon注:分别是640×480像素和160×120像素。)

flash_modes()

返回手机支持的闪光模式。

max_zoom()

返回手机支持的最大数码变焦次数。(shagon注:例如返回“3”表示可变焦三次,即有四种焦距)

exposure_modes()

返回手机支持的曝光模式。如['auto','night'] (shagon注:分别为“自动”、“夜间模式”)

white_balance_modes()

返回手机支持的白平衡模式。如:['auto']

take_photo([mode, size, flash, zoom, exposure, white_balance, position])

拍摄照片,并以如下之一方式返回图象:

1. Image 格式(更多信息参阅章节5.2的graphics模块。)
2. 原始JPEG数据⁴。

可选项中涵盖了摄像头所有拍摄照片的基本设定。但具体是否支持必须结合真机来判断,可以用上面提到的相关函数测试。下面分别介绍各个项目,并列举可能的取值:

• *mode* 用于设定图象格式。默认值是'RGB16'。所有可能的取值如下:

- 'RGB12': 4096色(12位)
- 'RGB16': 65536色(16位)。默认值,一般手机都支持。
- 'RGB': 16777216色(24位)

对于 JPEG 数据还支持下列格式:

- 'JPEG_Exif': EXIF, 可交换图像文件格式
- 'JPEG_JFIF': JFIF, JPEG档案交换格式

• *size* 用于设定图象大小。默认值是(640, 480)。还有其他取值,例如诺基亚6670支持以下大小:(1152, 864), (640, 480), (320, 240), (160, 120)

• *flash* 用于设定闪光模式。默认是'none'。所有可能的取值如下:

- 'none'
- 无闪光。默认值,一般手机都支持。

³ 部分数据的描述依据S60 SDK文档[4]中的定义。

⁴ 更多信息请访问<http://en.wikipedia.org/wiki/JPEG>

- 'auto'
自动闪光
- 'forced'
强闪光
- 'fill_in'
柔和闪光
- 'red_eye_reduce'
消除红眼模式

- *zoom* 用于设定数码变焦次数。取值从0到 `max_zoom`。(shagon注：0即原始焦距；1则变焦一次，即第二种焦距；2则变焦二次，即第三种焦距；以下类推)

- *exposure* 用于设定曝光模式。注意，镜头光圈大小和快门速度都对曝光有影响。默认的曝光模式是 'auto'。所有可能的取值如下：

- 'auto'
自动曝光。默认值，一般手机都支持。
- 'night'
夜间模式（长曝光）
- 'backlight'
背光模式（明亮背景）
- 'center'
特写模式（特写镜头，周围环境模糊处理）

- *white_balance* 用于设定白平衡。物体颜色会因为投射光线而产生改变，在不同光线下拍摄出的照片会有不同的色温。(shagon注：事实上，人眼瞳孔能够自行修正光线的摄入量，但是数码摄像头当然做不到，所有才有必要设置这个所谓的“白平衡”，以此调节图片色温。)

默认的白平衡是 'auto'。所有可能的取值如下：

- 'auto' 色温值(shagon注)
自 动 约3000 - 7000K
- 'daylight'
日 光 约5200K
- 'cloudy'
阴 天 约6000K
- 'tungsten'
钨丝灯 约3200K
- 'fluorescent'
荧光灯 约4000K
- 'flash'
闪光灯 约6000K

- *position* 仅适用于具有多个摄像头的机型，用于指定使用哪个摄像头。以诺基亚6680为例，设为0则使用主摄像头(机身背面的大摄像头)，设为1则使用副摄像头(机身正面的小摄像头)。默认值是0。

注意，如果同时已经有其他程序在使用摄像头，将引起 `SymbianError: KErrInUse`；如果摄像头尚未准备就绪，将引起 `SymbianError: KErrNotReady`。

```
start_finder(callable[, backlight_on=1, size=main_pane_size])
```

启动取景器。找到目标后，将图象作为参数随 *callable* 调回。

可选项中，*backlight_on* 用于设定是否开启背景灯。1为开启，0为关闭。默认情况是开启背景灯。*size* 用于设定图象大小。形如 (176, 144) (shagon注：即176×144像素)。默认与主面板的大小相同(shagon注：诺基亚6670的主面板大小即176×144)。

示例代码：

```

>>> import appuifw
>>> import camera
>>> def cb(im):
...     appuifw.app.body.blit(im)
...
>>> import graphics
>>> appuifw.app.body=appuifw.Canvas()
>>> camera.start_finder(cb)
>>>

```

stop_finder()

停止取景器

release()

释放摄像头(只有释放之后,其他程序才能使用摄像头,否则会发生错误)

start_record(filename, callable)

开始录像。*filename* 是存储文件的完整路径。而 *callable* 将携带一个错误校验码被调回。校验码用来指示状态信息(参阅前面关于“状态数据”的描述)。

注意,开始录像之前必须先打开取景器。

stop_record()

停止录像

5.4 keycapture — 按键捕获

keycapture 模块提供了用于捕获按键事件的API。它依靠创建 KeyCapturer对象来监听按键。

一旦按键被按下,KeyCapturer 对象马上就会用回调方法报告按键事件。

目前keycapture 模块不支持捕获单独的“按下”或“放开”事件。

注 意:对于S60 3rd 机型,必须具有软件事件能力(capability SwEvent) 才能正常使用Keycapture。

(shagon注:Capability(能力)是S60第三版平台的系统安全机制中引入的重要概念。SwEvent(软件事件)能力授予应用程序捕获软件的按键以及指点笔事件的权利。)

5.4.1 模块常量

keycapture 模块定义了以下常量:

all_keys

定义在key_codes 模块中所有按键的列表。

5.4.2 KeyCapturer 对象

KeyCapturer 对象用回调方法报告所监听到的按键事件。回调函数将携带单个参数,指示被捕获的按键。

允许同时创建多个KeyCapturer 对象。

KeyCapturer 对象具有下列方法和特性:

keys

捕获按键列表。可读可写。

示 例:

```

keys = (key_codes.EkeyUpArrow,)
keys = keycapture.all_keys

```

forwarding

设定被捕获的按键事件是否转寄到其他程序中（1或0）。可读可写。

start()

开始捕获

stop()

停止捕获

last_key()

返回上一次捕获的按键

5.5 topwindow — 置顶窗体

topwindow 模块用于创建并管理置顶窗体(覆盖所有其他程序窗体，显示在屏幕最上方)。

可以向窗体插入图象，也可以设置背景色、可见度、拐角类型以及窗体阴影。

topwindow 依赖于 graphics 服务，事实上 TopWindow 对象所做的只是把 graphics Image 对象置顶。

TopWindow 对象提供两项服务：1. 显示/隐藏置顶窗体 2. 在窗体中显示图象。

不过可喜的是允许同时创建多个置顶窗体，并且窗体中允许插入多个图象。(shagon注:于是可以产生华丽的效果。。。)

5.5.1 TopWindow 对象

TopWindow() 类

生成 TopWindow 对象

TopWindow 对象具有下列方法和特性：

show()

显示窗体

hide()

隐藏窗体

add_image(*image*, *position*)

向窗体插入图象 *image*(graphics.Image)。 *position* 用于图象定位，可以只限定左上角顶点，也可以限定矩形框(两个坐标点, 左上角和右下角顶点)。前者图象保持原始大小, 后者调整到指定大小。

示 例：

```
add_image(image, (10,20))
add_image(image, (10,20,20,30))
```

remove_image(*image*[, *position*])

从窗体移除图象 *image*(graphics.Image)。如果未设定可选项 *position*，则移除窗体中的所有图象，否则移除指定的图象。可以只限定左上角顶点，也可以限定矩形框(两个坐标点, 左上角和右下角顶点)。对于后者，只有完全吻合矩形框的图象才会被移除。

示 例：

```
remove_image(image)
remove_image(image, (10,10))
remove_image(image, (10,10,20,20))
```

position

窗体定位(左上角顶点, 可读可写)

示 例:

```
position = (10, 20)
```

size

窗体大小(可读可写)

示 例:

```
size = (100, 200)
```

images

配置要插入到窗体中的图象。包含了一组 `graphics.Image` 对象以及它们的位置。对于位置的设定, 可以只限定左上角顶点, 也可以限定矩形框(左上角和右下角顶点)。前者图象保持原始大小, 后者自动调整到指定大小。参阅前面的 `add_image()` 和 `remove_image()` 方法。

示 例:

```
images = [(image1,(x1,y1)), (image2,(x1,y1,x2,y2)), (image3,(50,50,100,100))]
```

插入三个图象, 其中 `image1` 保持原始大小, 而 `image2`、`image3` 将调整为指定的矩形框大小。

shadow

窗体阴影(可读可写)。设为0即无阴影, 设为非0的数则生成相应厚度的阴影。

示 例: `shadow = 5`

corner_type

窗体拐角(可读可写)。可用的窗体拐角类型有:

- square
- corner1
- corner2
- corner3
- corner5

示 例: `corner_type = square` ([shagon注: 即标准的矩形窗体](#))

maximum_size

窗体最大值(二元组:(宽度,高度)。只读)

background_color

窗体背景色(可读可写)。形如 `0xaaabcc`。其中 `aa` 代表红色, `bb` 代表蓝色, `cc` 代表绿色, 均为十六进制。

示 例: `background_color = 0xffffffff` (设为白色)

visible

窗体可见度(可读可写)。只能设为1或0。1即可见, 0即不可见。参阅前面的 `show()` 和 `hide()` 方法。

5.6 gles — 绑定 OpenGL ES

gles 模块绑定了OpenGL ES 2D/3D graphics C API ([shagon注：平面图形及立体图形的应用程序接口](#))。OpenGL ES 的官方组织(Khronos Group)主页是<http://www.khronos.org>。目前来说,S60 Python仅支持OpenGL ES 1.0版(S60 2.6以上),但是相信1.1版(S60 3rd 以上适用)在不久的将来也会被支持,因此本书对两个版本都作介绍。

OpenGL ES API文档详见<http://www.khronos.org/opengles>

PyOpenGL桌面系统 OpenGL绑定详见<http://pyopengl.sourceforge.net>

OpenGL ES 图形支持需要另外一个独立的模块,glcanvas。更多信息参阅glcanvas 所在章节。

5.6.1 array 类型

为方便起见,gles 模块的array 以数值形式定义了GL类型。同时,也提供了一些简单的方法。

array(*type, dimension, sequence*)类

生成 array 对象。

type 指定类型,可以是 GL_FLOAT, GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT 或 GL_FIXED中的一个。

*dimension*指定数组元素的项目个数,换句话说也就是数组维度。它主要被用于某些需要获取输入数据元素大小的函数(比如要知道color含有三个还是四个成分)。注意 *dimension*不会影响数组的大小或索引。

*sequence*是格式化数组。如果*type* 指定为GL_FLOAT 那么允许含有浮点值和整型值,否则只能含整型值。这些数值都将通过C转换成相应的请求类型,一旦数值有误将产生无法预料的错误。

__len__()

返回array中的项目个数(与数组维度无关)。

__getitem__(*index*)

返回array中索引值为*index* 的项目(与数组维度无关)。

__setitem__(*index, value*)

设定array中索引号为*index* 的项目的*value* (值,与数组维度无关)。

5.6.2 错误处理

与PyOpenGL的错误处理类似,所有的GL错误都被报告成 gles.GLerror。Python运行相关代码时会自动检测GL错误状态,但并不是绑定了glGetError 请求。

5.6.3 OpenGL ES C API 差异

Python对于特定的OpenGL ES函数需要进行特定的处理,这是因为C API指针参量的问题。

Python支持不同的OpenGL ES版本。下面对OpenGL ES 1.0版和1.1版的相关函数分别做介绍。如果有函数未在此提及,请参阅官方网站的相关信息。

`glColorPointer(size, type, stride, sequence)`

sequence 必须是 `gles.array` 对象或其他Python序列对象。不过 `gles.array` 对象可以处理得更快。如果使用了 `gles.array` 对象，那么数据本身的类型和维度都被忽略，取而代之的是指定的 *size* 和 *type*。

`glColorPointerub(sequence)`

特殊的 `glColorPointer`，支持 `gles.array` 对象或其他Python对象。

其他参量包括：

- *size* 若是 `gles.array` 实例则是维度大小，否则是 *sequence* 的大小
- *type* `GL_UNSIGNED_BYTE`
- *stride* 0

`glColorPointerf(sequence)`

特殊的 `glColorPointer`，与 `glColorPointerub` 类似，唯一不同的是 *type* 指定为 `GL_FLOAT`。

`glColorPointerx(sequence)`

特殊的 `glColorPointer`，与 `glColorPointerub` 类似，唯一不同的是 *type* 指定为 `GL_FIXED`。

`glCompressedTexImage2D(target, level, internalformat, width, height, border, imageSize, data)`

data 必须是 `gles.array` 或Python字符串

`glCompressedTexSubImage2D(target, level, xoffset, yoffset, width, height, format, imageSize, data)`

data 必须是 `gles.array` 或Python字符串

`glDeleteTextures(sequence)`

sequence 必须是只含整型值的Python序列

`glDrawElements(mode, count, type, indices)`

indices 必须是 `gles.array` 或其他Python序列对象。不过 `gles.array` 可以处理得更快。

如果使用了 `gles.array` 对象，那么数据本身的类型被忽略，取而代之的是指定的 *type*。

`glDrawElementsub(mode, indices)`

特殊的 `glDrawElements`。*count* 指定为序列 *indices* 的大小，而 *type* 指定为 `GL_UNSIGNED_BYTE`。

`glDrawElementsus(mode, indices)`

特殊的 `glDrawElements`。*count* 指定为序列 *indices* 的大小，而 *type* 指定为 `GL_UNSIGNED_SHORT`。

`glFogv(pname, params)`

params 必须是只含浮点值的Python序列

`glFogxv(pname, params)`

params 必须是只含整型值的Python序列

`glGenTextures(n)`

返回值是一个Python元组

`glGetIntegerv(pname)`

返回值是一个Python元组

`glGetString(name)`

返回值是一个Python字符串

`glLightModelfv(pname, params)`

params 必须是只含浮点值的Python序列

`glLightModelxv(pname, params)`

params 必须是只含整型值的Python序列

`glLightfv(light, pname, params)`

params 必须是只含浮点值的Python序列

`glLightxv(light, pname, params)`

params 必须是只含整型值的Python序列

`glLoadMatrixf(m)`

m 必须是只含浮点值的Python序列。序列将自动被格式化(转换成相应类型)

`glLoadMatrixx(m)`

m 必须是只含整型值的Python序列。序列将自动被格式化(转换成相应类型)

`glMaterialfv(face, pname, params)`

params 必须是只含浮点值的Python序列

`glMaterialxv(face, pname, params)`

params 必须是只含整型值的Python序列

`glMultMatrixf(m)`

m 必须是只含浮点值的Python序列。序列将自动被格式化(转换成相应类型)

`glMultMatrixx(m)`

m 必须是只含整型值的Python序列。序列将自动被格式化(转换成相应类型)

`glNormalPointer(type, stride, sequence)`

sequence 必须是 `gles.array` 或其他Python序列对象。不过 `gles.array` 可以处理得更快。
如果使用了 `gles.array` 对象，那么数据本身的类型被忽略，取而代之的是指定的 *type*。

`glNormalPointerb(sequence)`

特殊的 `glNormalPointer`。 *type* 指定为 `GL_BYTE`，而 *stride* 指定为0。

`glNormalPointers(sequence)`

特殊的 `glNormalPointer`。 *type* 指定为 `GL_SHORT`，而 *stride* 指定为0。

`glNormalPointerf(sequence)`

特殊的 `glNormalPointer`。 *type* 指定为 `GL_FLOAT`，而 *stride* 指定为0。

`glNormalPointerx(sequence)`

特殊的 `glNormalPointer`。 *type* 指定为 `GL_FIXED`，而 *stride* 指定为0。

`glReadPixels(x, y, width, height, format, type)`

返回包含像素数据的Python字符串

`glTexCoordPointer(size, type, stride, sequence)`

sequence 必须是 `gles.array` 对象或其他Python序列对象。不过 `gles.array` 对象可以处理得更快。
如果使用了 `gles.array` 对象，那么数据本身的类型和维度都被忽略，取而代之的是指定的 *size* 和 *type*。

`glTexCoordPointerb(sequence)`

特殊的 `glTexCoordPointer`，支持 `gles.array` 对象或其他Python序列对象。

其他参量包括：

- *size* 若是 `gles.array` 实例则是维度大小，否则是 *sequence* 的大小
- *type* `GL_BYTE`
- *stride* 0

`glTexCoordPointers(sequence)`

特殊的 `glTexCoordPointer`，与 `glTexCoordPointerb` 类似，唯一不同的是 *type* 指定为 `GL_SHORT`。

`glTexCoordPointerf(sequence)`

特殊的`glTexCoordPointer`，与`glTexCoordPointerb`类似，唯一不同的是`type`指定为`GL_FLOAT`。

`glTexCoordPointerx(sequence)`

特殊的`glTexCoordPointer`，与`glTexCoordPointerb`类似，唯一不同的是`type`指定为`GL_FIXED`。

`glTexEnvfv(face, pname, params)`

`params`必须是只含浮点值的Python序列

`glTexEnvxv(face, pname, params)`

`params`必须是只含整型值的Python序列

`glTexImage2D(target, level, internalformat, width, height, border, format, type, pixels)`

`pixels`必须是Python字符串，或者`gles.array`对象，或者`graphics.Image`对象。

对于Python字符串，无需转换。

对于`gles.array`对象，其维度和类型都被忽略而直接使用数组中的原始数据。

对于`graphics.Image`对象，只能使用有限的几种`format`和`type`的组合，如表5.1所示。为了能达到最佳的显示效果，`graphics.Image`中的`CFbsBitmap`对象必须采用表中所列的显示模式，否则可能造成失真。

表5.1 有效的`format`和`type`组合及其对应的Symbian显示模式

<i>format</i>	<i>type</i>	对应的显示模式
<code>GL_LUMINANCE, GL_ALPHA</code>	<code>GL_UNSIGNED_BYTE</code>	EGray256
<code>GL_RGB</code>	<code>GL_UNSIGNED_BYTE</code>	EColor16M
<code>GL_RGB</code>	<code>GL_UNSIGNED_SHORT_5_6_5</code>	EColor64K

`glTexSubImage2D(target, level, xoffset, yoffset, width, height, format, type, pixels)`

对于`pixels`的要求与`glTexImage2D`相同。

`glVertexPointer(size, type, stride, sequence)`

`sequence`必须是`gles.array`对象或其他Python序列对象。不过`gles.array`对象可以处理得更快。如果使用了`gles.array`对象，那么数据本身的类型和维度都被忽略，取而代之的是指定的`size`和`type`。

`glVertexPointerb(sequence)`

特殊的`glVertexPointer`，支持`gles.array`对象或其他Python对象。

其他参量包括：

- `size` 若是`gles.array`实例则是维度大小，否则是`sequence`的大小
- `type` `GL_BYTE`
- `stride` 0

`glVertexPointers(sequence)`

特殊的`glVertexPointer`，与`glVertexPointerb`类似，唯一不同的是`type`指定为`GL_SHORT`。

`glVertexPointerf(sequence)`

特殊的`glVertexPointer`，与`glVertexPointerb`类似，唯一不同的是`type`指定为`GL_FLOAT`。

`glVertexPointerx(sequence)`

特殊的`glVertexPointer`，与`glVertexPointerb`类似，唯一不同的是`type`指定为`GL_FIXED`。

OpenGL ES 1.1

`glBufferData(target, size, data, usage)`

data 必须是 `gles.array` 对象。如果 *size* 设为 -1，则取内存中 *data* 的真实大小。

`glBufferDatab(target, data, usage)`

特殊的 `glBufferData`，*data* 可以是 `gles.array` 对象或其他 Python 序列对象。若使用 `gles.array` 对象，其在内存中的真实大小作为 *size*。若使用其他 Python 序列对象，其序列大小作为 *size*，而序列中的数据将被自动转化为 `GL_BYTE` 类型。

`glBufferDataub(target, data, usage)`

特殊的 `glBufferData`，与 `glBufferDatab` 类似，唯一不同的是数据转化为 `GL_UNSIGNED_BYTE` 类型。

`glBufferDatas(target, data, usage)`

特殊的 `glBufferData`，与 `glBufferDatab` 类似，唯一不同的是数据转化为 `GL_SHORT` 类型。

`glBufferDataus(target, data, usage)`

特殊的 `glBufferData`，与 `glBufferDatab` 类似，唯一不同的是数据转化为 `GL_UNSIGNED_SHORT` 类型。

`glBufferDataf(target, data, usage)`

特殊的 `glBufferData`，与 `glBufferDatab` 类似，唯一不同的是数据转化为 `GL_FLOAT` 类型。

`glBufferDatax(target, data, usage)`

特殊的 `glBufferData`，与 `glBufferDatab` 类似，唯一不同的是数据转化为 `GL_FIXED` 类型。

`glBufferSubData(target, size, data, usage)`

data 必须是 `gles.array` 对象。如果 *size* 设为 -1，则取内存中 *data* 的真实大小。

`glBufferSubDatab(target, data, usage)`

特殊的 `glBufferSubData`，*data* 可以是 `gles.array` 对象或其他 Python 序列对象。若使用 `gles.array` 对象，其在内存中的真实大小作为 *size*。若使用其他 Python 序列对象，其序列大小作为 *size*，而序列中的数据将被自动转化为 `GL_BYTE` 类型。

`glBufferSubDataub(target, data, usage)`

特殊的 `glBufferSubData`，与 `glBufferDatab` 类似，唯一不同的是数据转化为 `GL_UNSIGNED_BYTE` 类型。

`glBufferSubDatas(target, data, usage)`

特殊的 `glBufferSubData`，与 `glBufferDatab` 类似，唯一不同的是数据转化为 `GL_SHORT` 类型。

`glBufferSubDataus(target, data, usage)`

特殊的 `glBufferSubData`，与 `glBufferDatab` 类似，唯一不同的是数据转化为 `GL_UNSIGNED_SHORT` 类型。

`glBufferSubDataf(target, data, usage)`

特殊的 `glBufferSubData`，与 `glBufferDatab` 类似，唯一不同的是数据转化为 `GL_FLOAT` 类型。

`glBufferSubDatax(target, data, usage)`

特殊的 `glBufferSubData`，与 `glBufferDatab` 类似，唯一不同的是数据转化为 `GL_FIXED` 类型。

`glClipPlanef(plane, equation)`

equation 必须是只含浮点值的 Python 序列

`glClipPlanex(plane, equation)`

equation 必须是只含整型值的Python序列

`glDeleteBuffers(buffers)`
buffers 必须是只含整型值的Python序列

`glDrawTexsvOES(coords)`
coords 必须是只含整型值的Python序列

`glDrawTexivOES(coords)`
coords 必须是只含整型值的Python序列

`glDrawTexfvOES(coords)`
coords 必须是只含浮点值的Python序列

`glDrawTexfvOES(coords)`
coords 必须是只含整型值的Python序列

`glGenBuffers(n)`
 返回一个Python元组

`glGetBooleanv(pname)`
 返回一个Python元组

`glGetBufferParameteriv(target, pname)`
 返回一个整型值

`glGetClipPlaneiv(plane)`
 返回一个Python元组

`glGetClipPlaneiv(plane)`
 返回一个Python元组

`glGetFixedv(pname)`
 返回一个Python元组

`glGetFloatv(pname)`
 返回一个Python元组

`glGetLightfv(light, pname)`
 返回一个Python元组

`glGetLightxv(light, pname)`
 返回一个Python元组

`glGetMaterialfv(face, pname)`
 返回一个Python元组

`glGetMaterialxv(face, pname)`
 返回一个Python元组

`glGetTexEnvf(face, pname)`
 返回一个Python元组

`glGetTexEnvx(face, pname)`
 返回一个Python元组

`glGetTexParameterf(target, pname)`
 返回一个浮点值

`glGetTexParameterx(target, pname)`
 返回一个整型值

`glMatrixIndexPointerOES(size, type, stride, sequence)`
sequence 必须是 `gles.array` 对象或其他Python序列对象。不过 `gles.array` 对象可以处理得更快。
 如果使用了 `gles.array` 对象，那么数据本身的类型和维度都被忽略，取而代之的是指定的 *size* 和 *type*。

`glMatrixIndexPointerOESub(sequence)`
 特殊的 `glMatrixIndexPointerOES`，支持 `gles.array` 对象或其他Python对象。

其他参量包括：

- *size* 若是 `gles.array` 实例则是维度大小，否则是 *sequence* 的大小
- *type* `GL_UNSIGNED_BYTE`
- *stride* 0

`glPointParameterfv(pname, params)`

params 必须是只含浮点值的Python序列

`glPointParameterxv(pname, params)`

params 必须是只含整型值的Python序列

`glPointSizePointerOES(type, stride, sequence)`

sequence 必须是 `gles.array` 或其他Python序列对象。不过 `gles.array` 可以处理得更快。如果使用了 `gles.array` 对象，那么数据本身的类型被忽略，取而代之的是指定的 *type*。

`glPointSizePointerOESf(sequence)`

特殊的 `glPointSizePointerOES`。 *type* 指定为 `GL_FLOAT`，而 *stride* 指定为 0。

`glPointSizePointerOESx(target, data, usage)`

特殊的 `glPointSizePointerOES`。 *type* 指定为 `GL_FIXED`，而 *stride* 指定为 0。

`glWeightPointerOES(size, type, stride, sequence)`

sequence 必须是 `gles.array` 对象或其他Python序列对象。不过 `gles.array` 对象可以处理得更快。如果使用了 `gles.array` 对象，那么数据本身的类型和维度都被忽略，取而代之的是指定的 *size* 和 *type*。

`glWeightPointerOESf(sequence)`

特殊的 `glWeightPointerOES`，支持 `gles.array` 对象或其他Python对象。

其他参量包括：

- *size* 若是 `gles.array` 实例则是维度大小，否则是 *sequence* 的大小
- *type* `GL_FLOAT`
- *stride* 0

`glWeightPointerOESx(sequence)`

特殊的 `glWeightPointerOES`，与 `glWeightPointerOESf` 类似，唯一不同的是 *type* 指定为 `GL_FIXED`。

5.7 glcanvas — OpenGL ES 图形显示控件

`glcanvas` 模块提供了一个叫做 `GLCanvas` 的UI控件，用于显示OpenGL ES图形。`GLCanvas`的组成成分与 `appuifw.Canvas` 非常相似。

`GLCanvas` 使用EGL来显示OpenGL ES图象(EGL是标准API定义，详见www.khronos.org)，更准确地说是使用双缓冲的EGL窗体。更详细的信息请参阅相关技术文档。

`GLCanvas` 实例掌控OpenGL ES上下文对象，产生相应的显示。通常，当一个新的 `GLCanvas` 对象被创建时它就会被作为当前对象，直到它被销毁，或被另外的 `GLCanvas` 对象取代。因此，如果一个程序中含有多个 `GLCanvas` 对象的话，需要调用 `makeCurrent` 方法令所需的特定对象成为当前对象。

`GLCanvas(redraw_callback, [event_callback=None, resize_callback=None, attributes=None])` 类生成 `GLCanvas` 对象，即显示 OpenGL ES 图象的 UI 控件。

参数中的 `redraw_callback`, `event_callback` 和 `resize_callback` 与 `appuifw` 模块的 `Canvas` 有类似的定义。不同的是，它的 `redraw_callback` 可以被 `drawNow` 方法自动调用。

`attributes` 用于配置 EGL 相关属性，必须是一个 Python 字典，键名为 EGL 属性名称 (定义在 `glcanvas` 模块中)，值为对应的整型数字。如果没有特别设定 `attributes`，那么由 `CCoeControl` 掌控的窗体显示模式对应值就被指派为 `EGL_BUFFER_SIZE` 的值，而 `EGL_DEPTH_SIZE` 被设定为 16。`attributes` 中的属性配置将转寄给 `eglChooseConfig`，形成一份配置明细表，以及解释相关配置如何工作。

当一个新的 `GLCanvas` 对象被创建时，它会立即成为当前对象。所以如果刚创建的对象就是所需对象，就没有必要调用 `makeCurrent` 方法。

`bind(key_code, c)`
设置按键响应。`key_code` 必须是定义在 `key_codes` 模块中的键码。`c` 必须是可请求对象。

`drawNow()`
发出重绘屏幕请求，并调用 `eglSwapBuffers` 以显示 OpenGL ES 图象。

`makeCurrent()`
令所需 `GLCanvas` 对象立即成为当前对象。“当前”的意思是只有它能被用于 OpenGL ES 显示。更确切地，在 EGL 术语中“当前对象”掌控着 EGL 上下文和界面并负责传递给 `eglMakeCurrent`。使用 `makeCurrent` 方法可以将所需 `GLCanvas` 对象切换为当前对象，这对拥有多个对象的程序很有用。

5.8 sensor — 访问手机传感器

注 意：只支持 S60 3rd 以上机型

`sensor` 模块提供了访问手机物理传感器的功能。以下测试通过：

- 加速度传感器：触发手机三轴加速度事件 (shagon 注：三轴，3-axes，XYZ 轴)
- 轻触传感器：触发手机双击 (连续轻触两次) 事件
- 旋转传感器：触发手机方位事件

访问传感器的功能不只是传递触发事件，也可以过滤事件。例如，诺基亚 N95 支持 `RotEventFilter`，而诺基亚 5500 支持 `OrientationEventFilter`。它们都可以对手机方位改变而触发的事件进行过滤。

5.8.1 模块函数

`sensor` 模块提供下列函数：

`sensors()`

返回一个字典，包含了所有可用的传感器。形如：

```
{
    { 'sensor name 1': { 'id': sensor_id_1, 'category': category_id_1 } },
    { 'sensor name 2': { 'id': sensor_id_2, 'category': category_id_2 } },
}
```

```
...
}
```

`sensor_id_X` 和 `category_id_X` 都是整型值

5.8.2 常 量

以下都是在`OrientationEventFilter`类中使用的方位常量。当传感器感知到手机方位改变时就会以这些常量作为参数进行传递。方位常量是根据机身方位命名的，例如 `FRONT` 意味着手机正面向上放置，用户视线面对的是机身正面。

`orientation.TOP`

机身纵向正置(顶部向上)

`orientation.BOTTOM`

机身纵向倒置(底部向上)

`orientation.LEFT`

机身左侧向下

`orientation.RIGHT`

机身右侧向下

`orientation.FRONT`

机身横向正置(正面向上)

`orientation.BACK`

机身横向倒置(背面向上)

5.8.3 类

`sensor` 模块提供下列类：

`Sensor` 类

`Sensor`类可以访问物理传感器并过滤和传递事件。默认情况下不过滤，可以用`set_event_filter`方法开启过滤。例如`OrientationEventFilter`可以对加速度传感器触发事件进行过滤。

如果要针对同一个传感器进行多项不同的过滤，那就必须针对这个传感器创建多个`Sensor`对象。

`__init__(sensor_id, category_id)`

初始化 `Sensor` 对象。参数中`sensor_id`, `category_id` 必须是有效值，否则会引起 `connect` 方法异常。“有效”的意思是当它们传递给`__init__`时，这些值同样可以在字典中找到。

`connect(callback)`

连接传感器与回调函数。一个传感器只能有一个回调函数，所以如果已有回调函数那么就取而代之。另外，如果开启了过滤，那么就会先对传感器触发事件进行过滤。

当连接已被确认，将返回1，否则返回0。注 意：如果回调不存在或者无法实现，同样视为已被确认。

`disconnect()`

断开传感器与回调函数的连接。断开成功返回1，否则返回0。一旦断开成功，用`connect`方法连接起来的回调函数将不再接收任何事件。

`connected()`

检测传感器与回调函数的连接状态。已连接返回True，否则返回False。

`set_event_filter(event_filter)`

开启过滤。开启之后回调方法只能接收已过滤的事件。`event_filter`必须从EventFilter继承。如果已经确认了传感器与回调函数的连接，那么开启过滤后将重新确认。

EventFilter 类

EventFilter类提供了事件过滤接口。默认执行时不会过滤事件而直接传递，EventFilter派生类则可以对事件进行过滤，也就是决定哪些数据最终传递给回调函数。

`callback`

存储EventFilter的回调函数。当EventFilter对象与Sensor对象共同使用时，Sensor对象藉此设置相关变量。

`__init__()`

初始化EventFilter对象，并将callback初始化为None。

`__del__()`

销毁EventFilter对象。事实上也是调用cleanup进行扫尾工作。

`event(data)`

存储EventFilter事件。该方法需由派生类重载。

重载event方法可以将自身的data(数据或数据集)传递给回调函数。由重载实例调用无参的self.callback来传递事件。

`cleanup()`

回收资源。实际运行时无需特别调用该方法，由__del__自动调用。

OrientationEventFilter 类

从EventFilter类继承，仅适用于加速度传感器触发事件。即当手机机身方位发生改变时(比如开始竖直握在手里，后来正面朝上放到了桌上)引发事件。一旦用于Sensor对象，那么Sensor对象回调函数接收到的将不再是原始数据参数，而是经过转义的方位常量，以指示当前手机机身方位。如有需要，请查看sensor.py文件里面的OrientationCalibration相关变量。

`__init__()`

初始化OrientationEventFilter对象。

`event(sensor_val)`

过滤三轴加速度传感器触发事件(重载方法)。传递给回调函数的是经过转义的方位常量。

`cleanup()`

回收资源。实际运行时无需特别调用该方法，由__del__自动调用。

RotEventFilter 类

从EventFilter类继承。

当手机机身方位发生改变时(比如从机身左侧翻转到机身右侧)引发事件。

`event(sensor_val)`

重载方法。传递给回调函数的是经过转义的方位常量。

声音和通信服务

6.1 audio — 声音服务

audio 模块提供了录制、播放音频文件，以及访问文本朗读(TTS, text-to-speech)引擎的功能。其中，音频文件可以是任何手机本身支持的格式，通常有：WAV, AMR, MIDI, MP3, AAC以及Real文件。关于各种机型所支持的音频格式，详情请参阅诺基亚论坛网站[7]和S60平台网站[8]。

以下Sound类静态方法都定义在 audio 模块中：

Sound.open(filename)

打开指定文件，新建并初始化一个Sound对象。filename 必须是完整路径，形如 :u'c:\\foo.wav'。

audio 模块中的状态数据：

ENotReady

已创建Sound对象，但没有打开任何音频文件。

EOpen

已打开一个音频文件，但没有在播放或者录制。

EPlaying

正在播放一个音频文件。

ERecording

正在录制一个音频文件。

以下数据项用于支持连续重放功能：

KMdaRepeatForever

设置重放(参数times)

audio 模块中的方法：

say(text, prefix=audio.TTS_PREFIX)

将 text 传给文本朗读引擎。默认prefix是"(tts)"。

6.1.1 Sound 对象

注意：S60 1st机型不支持current_volume方法。

Sound类

Sound对象含有下列函数：

play([times=1, interval=0, callback=None])

播放音频文件。times 是重放次数(默认为1), interval 是重放延时(以微秒计)，缺省情况只

¹ 依据文件描述而动态加载相应的编码/解码器。

播放一遍。

可选项中, *callback* 用于设定回调函数, 该函数在音频开始播放以及结束播放(到达文件尾)时被调用。回调函数必须包含三个参数: 先前状态、当前状态、异常代码。这些参数都是定义在 *audio* 模块中的数据项。

需要注意的问题:

- 调用 `play(audio.KMdaRepeatForever)` 将不停地重放。
- 播放音频文件之后, 必须将其先停止再退出, 否则在Python脚本中将维持播放状态。
- 目前不支持同时播放多个音频文件。因此在播放新的文件之前, 必须先将旧的文件关闭。
- 在通话时播放音频, 对方会听到, 有些机型通话双方都会听到。
- 如果在录播放或者录制音频时调用 `play` 将引起 `RuntimeError` 异常。先调用 `stop` 可以避免这种情况。

`stop()`

停止播放和录制

`record()`

录制音频文件。如果已经存在, 则在文件尾追加。对于诺基亚手机, WAV是默认的录制格式。关于不同机型支持的各种格式, 详情请参阅诺基亚论坛网站[7]和S60平台网站[8]。

需要注意的问题:

- 通话时录制音频, 将会录下通话内容。
- 如果在录播放或者录制音频时调用 `record` 将引起 `RuntimeError` 异常。先调用 `stop` 可以避免这种情况。

`close()`

关闭文件

`state()`

返回Sound实例的当前状态。相关常量定义在*audio*模块中。

可能状态² 有:

- `ENotReady`
已创建Sound对象, 但没有打开任何音频文件。
- `EOpen`
已打开一个音频文件, 但没有在播放或者录制。
- `EPlaying`
正在播放一个音频文件。
- `ERecording`
正在录制一个音频文件。

`max_volume()`

返回手机支持的最大音量

`set_volume(volume)`

设定音量。若设为负数则自动调整为0, 即静音; 若超过`max_volume`则自动调整为`max_volume`。

`current_volume()`

返回当前音量

`duration()`

返回持续时间(微秒)

`set_position(microseconds)`

设定录放起始位置

²这些数据的描述依据S60 SDK文档[4]中的定义。

`current_position()`
返回当前录放位置

6.2 telephone — 电话服务

本模块提供支持电话服务的API。

显然,用户主动挂断电话时就会改变当前通话的状态;另外,模拟器中没有通话功能,所以该模块无效。

telephone 模块中含有下列函数:

`dial(number)`

呼叫 *number* 中设置的号码。例如: `'u'+358501234567'`。其中, + 是国际前缀, 358 是国家代码, 50 是网络代码(或称区域码), 1234567 是对方号码。

注意下列问题:

如果通话正在进行中而调用了 `dial`, 则会将该通话挂起并连接新通话。

如果多次调用 `dial`, 其中有一个呼叫成功的情况下, 其他呼叫不再连接。

`hang_up()`

将 `dial` 引发的呼叫挂断。

如果通话已经结束而调用 `hang_up`, 将引起 `SymbianError: KErrNotReady`

注意: 以下函数和数据项只支持 S60 3rd 以上机型

`incoming_call()`

监听来电。有新来电时马上调用 `answer` 来应答。

必须有 `incoming_call` 的激活, 否则 `answer` 无效。

`answer()`

应答来电。

`call_state(callable)`

通话状态改变时回调 *callable*。返回一个二元组, 前项是新状态, 后项是来电号码。

相关状态以常量形式定义在 telephone 模块中。

可用的数据项(状态信息)如下³:

`EStatusUnknown`

未知状态

`EStatusIdle`

空闲状态

`EStatusDialling`

呼叫状态

`EStatusRinging`

响铃状态

`EStatusAnswering`

应答状态

`EStatusConnecting`

正在连接状态

`EStatusConnected`

已连接状态

`EStatusReconnectPending`

临时性信号丢失, 可能引起重拨

³ 这些数据的描述依据 S60 SDK 文档[4]中的定义。

EStatusDisconnecting
断开连接状态
EStatusHold
挂起状态
EStatusTransferring
呼叫转移状态
EStatusTransferAlerting
呼叫转移正在进行远程通信

6.3 messaging — 信息服务

本模块提供支持信息服务的API。通常包含如下函数：

`sms_send(recipient, message, [encoding='7bit', callback=None])`

向号码为 *recipient* 的手机发送SMS短信, 内容为 *message*。

可选参数 *encoding* 指定文字编码, 例如: '7bit', '8bit', 'UCS2'。

可选参数 *callback* 以发送状态作为回调参量。相关状态信息定义在 `messaging` 模块中。

注意, 当先前的调用正在进行发送请求时, 调用第二个发送将引起 `RuntimeError`。

callback 缺省情况下, `sms_send` 函数在被调用之后挂起, 直到所发信息在队列中删除或发送失败⁴。

`mms_send(recipient, message, [attachment=None])`

注 意: 只支持S60 3rd 以上机型

向号码为 *recipient* 的手机发送MMS彩信, 内容为 *message*。

可选参数 *attachment* 是多媒体附件 (比如图片) 的完整路径。

以下数据项是 `messaging` 模块中可用的SMS发送状态信息：

ECreated

EMovedToOutBox

EScheduledForSend

ESent

SMS短信发送成功

EDeleted

SMS短信已从发件箱中删除

`sms_send` 操作将重新规划剩余可能的SMS短信发送

EScheduleFailed

ESendFailed

在SMS子系统多次尝试发送未果时返回这个失败状态信息。

`sms_send` 操作将重新规划剩余可能的SMS短信发送

ENoServiceCentre

在S60 3.x模拟器中返回的状态信息

在模拟器中它表明：`sms_send` 操作将重新规划剩余可能的SMS短信发送

EFatalServerError

在无网络连接时尝试发送SMS短信将引发这个状态信息。

通常会向用户显示“信息发送失败”警告。

⁴ 请注意, 此种挂起可能持续好几分钟, 所以一般建议设定可选参数 *callback*

在离线模式或无网络连接情况下尝试发送,信息将进入发送等待队列,直到连上网络之后开始发送⁵。
当前网络情况可用`sysinfo.active_profile()`和`sysinfo.signalBars()`查询。

以下代码就是`sms_send`状态信息处理示例：

```
>>> import messaging
>>>
>>> def cb(state):
...     if state==messaging.ESent:
...         print "**Message was sent**"
...     if state==messaging.ESendFailed:
...         print "**Something went wrong - Truly sorry for this**"
...
>>> messaging.sms_send("1234567", "Hello from PyS60!", '7bit', cb)
>>> **Message was sent** # 这就是回调函数的显示输出
```

6.4 inbox — 信箱接口

`inbox` 模块提供了收件箱、发件箱、已发件箱和草稿箱的API。
目前只支持SMS短信,并且只能捕获收件箱中的新信息。

`Inbox([folder_type])` 类

创建`Inbox` 对象

可选参数 *folder_type* 指定所访问的信箱。缺省为`inbox.EInbox`,即收件箱。

以下即为可用的信箱类型：

`EInbox`

收件箱

`EOutbox`

发件箱

`ESent`

已发件箱

`EDraft`

草稿箱

6.4.1 Inbox 对象

`Inbox` 模块中含有下列函数：

`sms_messages()`

获取信箱中的SMS短信ID列表

`content(sms_id)`

获取信箱中的SMS短信内容(指定ID)

`time(sms_id)`

获取信箱中的SMS发送时间(指定ID)

`address(sms_id)`

获取信箱中的SMS发送地址(指定ID)

⁵ 用户也可以手动删除待发信息(通过短信软件)。信息子系统将尝试发送最多约4次——如果最终失败则取消发送,此时用户将在状态面板看到失败警告。

```
delete(sms_id)
```

删除信箱中的SMS短信(指定ID)

```
unread(sms_id)
```

返回SMS短信(指定ID)的阅读状态(1:未读, 0:已读)

```
bind(callable)
```

创建用于捕获新信息的回调。当发现新的信息时 *callable* 将携带信息ID被调回。

注意:如果检查信息后立即删除,那么短信铃声和提示可能都不会出现。这个特性对于某些应用程序非常有用。

示 例:

```
>>> import inbox
>>> i=inbox.Inbox() #默认为inbox.ESent,即收件箱
>>> m=i.sms_messages()
>>> i.content(m[0])
u'foobar'
>>> i.time(m[0])
1130267365.03125
>>> i.address(m[0])
u'John Doe'
>>> i.delete(m[0])
>>>

>>> import inbox
>>> id=0
>>> def cb(id_cb):
...     global id
...     id=id_cb
...
>>> i=inbox.Inbox()
>>> i.bind(cb)
>>> # 向收件箱发送一条SMS短信. 'id' 被更新
>>> i.address(id)
u'John Doe'
>>> i.content(id)
u'print 1'
>>>
```

6.5 location — GSM 定位信息

location 模块提供了定位服务API。目前只支持一个函数:

注 意:需要 S60 3rd机型中ReadDeviceData,ReadUserData,Location等能力的支持。

```
gsm_location()
```

获取GSM定位信息,包括国家代码、网络代码、区域代码和蜂窝标识。定位区域通常包含多个基站,事实上它就是终端能在运营网络内移动的范围(无需告知其精确位置)。注册手机在运营网络的唯一标识码是由通信枢纽和跨国公司共同确立的(shagon注:原文是mcc和mnc,分别是Main Communication Center,通信枢纽; MultiNational Corporation,跨国公司)。

6.5.1 示 例

以下是通过location 模块来获取定位信息的示例:

```
>>> import location
>>> print location.gsm_location()
```

6.6 positioning — 位置信息的简化接口

positioning 模块提供了S60位置信息⁶的基本访问权限。包括由外接蓝牙GPS设备获取的信息和内置GPS接收器⁷获取的信息(S60 2nd FP2以上机型)。

本模块能够获取大量可访问定位装置的信息，包括位置、路线、精确度、卫星信息等等(依赖于具体的定位装置)，甚至还能获取其他细节方面的信息。

注 意：在S60 3rd 机型中需要 Location能力的支持

positioning 模块中的数据项：

POSITION_INTERVAL

position函数的回调响应间隔(以微秒计)。默认值是1,000,000微秒(等于1秒)。

positioning 模块中含有下列函数(返回值示例详见章节6.6.1)：

modules()

获取可用的positioning 模块信息

default_module()

获取默认模块ID

module_info(module_id)

获取指定模块的详细信息

select_module(module_id)

选择一个模块

set_requestors(requestors)

设定服务请求(至少一个)

position(course=0,satellites=0,callback=None, interval=positioning.POSITION_INTERVAL, partial=0)

缺省时返回一个位置信息字典。若设定了course和satellites,则同时返回路线和卫星信息(仅若可用)。

如果未设定callback,则位置信息不可用时该函数调用将一直被阻断。

如果给定callback函数,则立即返回调用,并以参数interval(以微秒计)作为调用时间间隔。callback函数以当前位置信息作为回调参数。

如果参数partial设为1,则函数返回的可能是未经过最终计算的信息。

返回字典和键值示例详见章节6.6.1

stop_position()

停止进行中的position 请求

⁶ 详细资料请参阅S60 API文档中的Location Acquisition API,它为多种不同技术应用提供了统一接口。

⁷ 关于GPS的更多信息,请访问http://en.wikipedia.org/wiki/Global_Positioning_System。

6.6.1 示 例

以下是如何运用positioning 模块来获取手机定位装置信息的示例(诺基亚N95测试通过)：

```
>>> positioning.modules()
[{'available': 0, 'id': 270526873, 'name': u'Bluetooth GPS'}, {'available': 1, 'id': 270526858, 'name': u'Integrated GPS'}, {'available': 1, 'id': 270559509, 'name': u'Network based'}]
>>> positioning.default_module()
270526858
>>> positioning.module_info(270526858)
{'available': 1, 'status': {'data_quality': 3, 'device_status': 7}, 'version': u'1.00(0)', 'name': u'Integrated GPS', 'position_quality': {'vertical_accuracy': 10.0, 'time_to_first_fix': 1000000L, 'cost': 1, 'time_to_next_fix': 1000000L, 'horizontal_accuracy': 10.0, 'power_consumption': 3}, 'technology': 1, 'id': 270526858, 'capabilities': 127, 'location': 1}
>>>
```

以下是如何使用positioning 模块的示例：

```
# 可用模块信息
print "***available modules***"
print positioning.modules()
print ""

# 默认模块ID
print "***default module***"
print positioning.default_module()
print ""

# 默认模块详细信息
print "***detailed module info***"
print positioning.module_info(positioning.default_module())
print ""

# 选择一个模块
positioning.select_module(positioning.default_module())

# 设定请求
# 至少设定一个请求,并且最后一个请求必须是服务申请
positioning.set_requestors([{"type": "service",
                             "format": "application",
                             "data": "test_app"}])

# 示例1.中断调用

# 定位
# 注意,第一次调用可能耗费较长时间(GPS技术原因)
print "***position info***"
print positioning.position()
print ""

# 再次定位
# 这个调用耗费较少时间
# 获取路线和卫星信息
print "***course and satelllites***"
print positioning.position(course=1,satelllites=1)
```

```

print ""

# 示例2.非中断调用

def cb(event):
    print "---"
    print event
    print "---"

print "***starts the position feed***"
print positioning.position(course=1,satellites=1,
                           callback=cb, interval=500000,
                           partial=0)

```

以上示例代码可能返回的字典数据如下：

```

{'satellites': {'horizontal_dop': 2.34999990463257, 'used_satellites': 5, 'vertical_dop': 2.29999995231628, 'time': 1187167353.0, 'satellites': 11, 'time_dop': 1.26999998092651}, 'position': {'latitude': 60.217033666473, 'altitude': 42.0, 'vertical_accuracy': 58.0, 'longitude': 24.878942093007, 'horizontal_accuracy': 47.531005859375}, 'course': {'speed': 0.0500000007450581, 'heading': 68.9599990844727, 'heading_accuracy': 359.989990234375, 'speed_accuracy': NaN}}

```

如需在模拟器运行以上代码，必须首先配置PSY：

(SimPSYConfigurator → Select Config File → <some config files> 或 Tools → Position).

数据管理

7.1 contacts — 联系人服务

contacts 模块提供了访问名片夹数据库的API。它以一种类似字典的 ContactDb 对象来描述数据库，包含了 Contact 对象，并以 ID(整型)进行索引。Contact 对象本身是一种类似列表的对象，它包含了 ContactField 对象并以字段索引值(整型)进行索引。ContactDb 对象的字典功能限制子集，事实上它只包含 `__iter__`、`__getitem__`、`__delitem__`、`__len__`、`keys`、`values` 和 `items`。

ContactDb 对象提供了对联系人数据库的实时访问。也就是说，其他程序所做的更改能够立即在 Python 程序中反映出来；另一方面，Python 程序所做的更改也能立即在其他程序中可见。当然，它也具有编辑锁定功能，以阻止其他程序对联系人进行更改。这和诺基亚手机自身的“名片夹”功能类似，用户在更改联系人信息时能够自动编辑锁定，这样其他程序就无法进行编辑，以免造成混乱。(shagon注:编辑锁定,也就是独占式编辑,即不允许多个程序同时对同一个信息进行读写)

7.1.1 模块函数

以下函数都定义在 Contact 模块中(同时没有定义在任何类中)：

```
open([filename[, mode]])
```

打开联系人数据库并返回 ContactDb 对象。*filename* 必须是完整路径，缺省时打开默认数据库。

如果缺省 *mode* 则指名数据库必须存在；如果 *mode* 是 'c' 则当指名数据库不存在时创建它；如果 *mode* 是 'n' 则新建一个空数据库，这将覆盖同名数据库。

警告：

可选参数 *filename* 和 *mode* 仅用于调试，事实上因为 S60 SDK 本身的原因，有时候这些参数是无效的。

7.1.2 ContactDb 对象

默认的联系数据库只有一个，不过使用 open 函数可以创建多个数据库。

ContactDb 类

ContactDb 对象支持下列方法：

```
add_contact()
```

增加一个联系人，并返回一个已编辑锁定的 Contact 对象。

通常一个联系人含有多项条目，对于不需要的条目可以直接忽略。

`find(searchterm)`

查找包含指定关键字的联系人，并以列表返回。

`import_vcards(vcards)`

导入指定的vCard ([shagon注:vCard是一种电子名片数据规范](#))

`export_vcards(ids)`

导出指定ID的vCard，以字符串返回

`keys()`

返回数据库中所有Contact对象的ID列表

`compact_required()`

校验是否需要压缩。返回一个表示真假的整型数。当大于32K的空间未被使用且这部分空间超过了数据库的50%，或者大于256K空间被闲置，则返回真。

`compact()`

将数据库压缩至最小

`__delitem__(id)`

删除指定联系人

`field_types()`

返回一个字典对象列表，包含所有可用条目类型信息。每个字典对象描述一个条目。其中最重要的键是'type'和'location'。'type'可以是类似'email_address'的字符串，而'location'是'none'，'home'，或'work'。另外一个重要的键是'storagetype'，它用于定义存储类型，通常是'text'，'datetime'，'item_id'，或'binary'。

注意，Contacts扩展模块不支持增加/读取/修改除'text'、'datetime'类型以外的条目。事实上，field_types返回的其他条目类型涉及高深的知识，本书不予讨论。

`groups`

返回联系人分组。只读。

7.1.3 Contact对象

Contact对象用于实时可视化访问数据库中的联系人。可用联系人ID(`contact[fieldid]`)或`find`方法来访问指定的联系人。

修改已被其他程序编辑锁定的联系人将引起ContactBusy异常。

Contact类

Contact对象含有以下属性：

`id`

当前Contact的唯一ID。只读。

`title`

当前Contact的标题。只读。

`is_group`

如果当前是联系人分组则返回1，如果是单一联系人则返回0。只读。

Contact类支持下列方法：

`begin()`

编辑锁定。这将阻止其他程序修改联系人。

锁定已被锁定的联系人将引起ContactBusy异常。

`commit()`

解除编辑锁定，并将更新存入数据库。

`rollback()`

解除编辑锁定，并丢弃更新，恢复到begin之前的状态。

`as_vcard()`

以 vCard 格式返回联系人。

`add_field(type [, value [, label=field_label]][, location=location_spec])`

增加联系人条目。如果联系人已被其他程序编辑锁定将引起 ContactBusy 异常。
参数 *type* 是字符串，代表可支持的条目类型。

S60 3rd 之前的版本支持以下条目类型：

- city
- company_name
- country
- date
- dtmf_string
- email_address
- extended_address
- fax_number
- first_name
- job_title
- last_name
- mobile_number
- note
- pager_number
- phone_number
- po_box
- postal_address
- postal_code
- state
- street_address
- url
- video_number
- wvid

以下条目类型可识别但无法创建：

- first_name_reading
- last_name_reading
- picture
- speed_dial
- thumbnail_image
- voicetag

S60 3rd 版本支持以下条目类型：

- city
- company_name
- country
- date
- dtmf_string
- email_address
- extended_address
- fax_number
- first_name
- job_title

- last_name
- mobile_number
- note
- pager_number
- phone_number
- po_box
- postal_address
- postal_code
- state
- street_address
- url
- video_number
- picture
- second_name
- voip
- sip_id
- personal_ringtone
- share_view
- prefix
- suffix
- push_to_talk
- locationid_indication

以下条目类型可识别但目前无法创建：

- first_name_reading
- last_name_reading
- speed_dial
- thumbnail_image
- voice_tag
- wvid

所有条目类型都以字符串传递，除了‘date’是浮点数(表示Unix时间)。
关于Unix时间详见章节3.5, 日期与时间。

field_label 是条目名称，缺省时使用默认名称。

location_spec 可以缺省，或是‘home’、‘work’其一。

注意，并不是任意的*type*与*location*组合都有效，这取决于具体的数据库。

```
find([type=field_type][, location=field_location])
```

查找当前联系人的指定条目。缺省时返回所有条目。

```
__delitem__(fieldindex)
```

从当前联系人中删除指定条目。

注意，这将改变所有条目的索引。同时因为ContactField对象以索引值对应条目，所以删除一个条目后，相应的原ContactField对象将指向其他条目。

7.1.4 ContactField对象

ContactField对象以索引值对应联系人条目，提供一些可供修改的属性。

如果它的父对象Contact未编辑锁定，其更新将被立即存入数据库；如果父对象已编辑锁定，那么必须在父对象调用commit函数后才能将更新存入数据库。

ContactField 类

ContactField对象含有以下属性：

label

用户可见的条目名称。可读可写。

value

条目值。可读可写。

type

条目类型。只读。

location

条目位置。只能是'none','work'或'home'。

schema

一个包含条目特性的字典，其所含内容与 ContactDb 的 field_types 方法返回值一致。

7.1.5 Groups 对象

Groups对象用于描述联系人分组，它是一种类似字典的对象(限制子集)。每个分组均以ID作为键值进行访问。

Groups对象返回一个类似于Group对象的列表,其值与指定键匹配。

支持通用方法：__iter__，__getitem__，__delitem__ and __len__。

Groups 类

Groups对象含有以下属性：

add_group([name])

创建新的联系人分组，并返回相应的Group对象，其名称是可选参数。

7.1.6 Group 对象

Group对象用于描述单一的联系人分组，它是一种类似列表的对象(限制子集)。Group可以列出属于当前分组的所有联系人ID。

Symbian系统中,分组同样是以联系人形式描述的,因此使用Contact对象也能访问分组,但这样就无法使用分组的独特功能。建议使用Groups和Group对象来访问分组。

支持通用方法：__iter__，__getitem__，__delitem__ and __len__。

Group类

Group对象含有以下属性：

id

当前Group对象的唯一ID。只读。

name

当前Group对象的名称。可读可写。

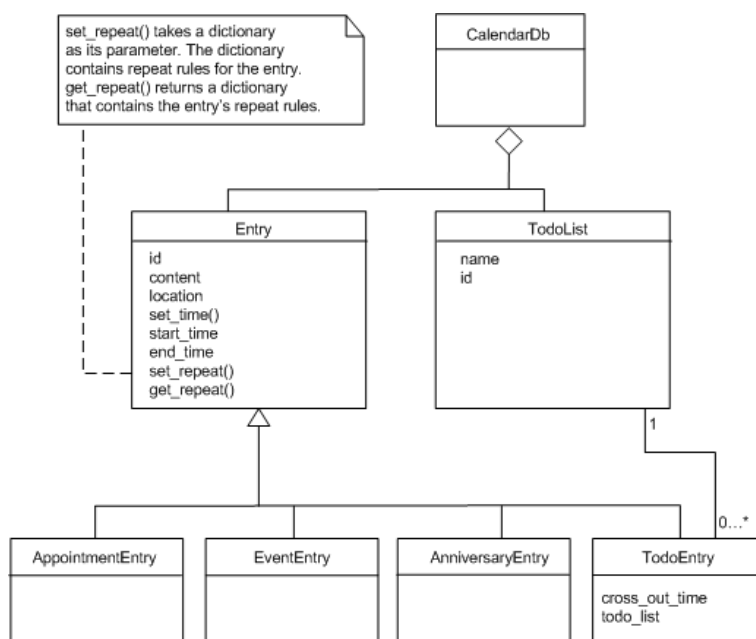


图7.1 :calendar 模块对象

7.2 calendar — 日历服务

calendar 模块提供了日历服务API。它以一种类似字典的CalendarDb 对象描述议程数据库, 包含Entry 对象, 并以ID进行索引。一共有四种Entry 对象, 分别是 :AppointmentEntry, EventEntry, AnniversaryEntry 和 TodoEntry。

CalendarDb 对象提供对数据库的实时访问。也就是说, 其他程序所做的更改能够立即在Python 程序中反映出来; 另一方面, Python 程序所做的更改也能立即在其他程序中可见。

待办事项表中的记录通过类似字典的 TodoListDict 和 TodoList 对象进行访问。

除个别外, 时间参数都使用Unix时间。关于Unix时间详见章节3.5, 日期与时间。

图7.1描绘了calendar 模块对象之间的关系。

7.2.1 模块函数

以下函数都定义在 calendar 模块中(同时没有定义在任何类中)：

`open([filename=None, mode=None])`

打开日历数据库并返回一个新的CalendarDb 对象。

filename 缺省时打开默认数据库。如果给定 *filename* 则必须是完整路径。

mode 可以是：

- None: 打开一个现有的数据库。
- 'c': 打开一个现有的数据库, 如果不存在则创建一个指名的数据库。
- 'n': 新建一个空数据库, 如果已存在同名数据库则将其覆盖。

7.2.2 CalendarDb 对象

日历项和待办事项都保存在日历数据库中。默认的数据库只有一个，但是可以用带参数'n'或'c'的open函数创建新的数据库。

CalendarDb类

CalendarDb 对象支持下列方法：

add_appointment()

创建并返回一个新的约会项AppointmentEntry。
调用Entry.commit后添加并保存至数据库。

add_event()

创建并返回一个新的备忘项EventEntry。
调用Entry.commit后添加并保存至数据库。

add_anniversary()

创建并返回一个新的纪念日项AnniversaryEntry。
调用Entry.commit后添加并保存至数据库。

add_todo()

创建并返回一个新的待办事项TodoEntry。
调用Entry.commit后添加并保存至数据库。

find_instances(start_date, end_date, search_str=u"[,appointments=0,events=0,anniversaries=0,todos=0]")

参数依次是起始日期、结束日期、搜索项和可选参数(指定记录类型)。在默认情况下搜索所有类型的记录。返回一个列表,包含搜索到的Entry实例。一个Entry实例就是包含ID和日期的字典。

一个记录可能含有多个实例,比如每周议程。然而所有返回实例都出现在同一天,也就是起迄日期之间的第一天。为了找到全部所需的实例就必须进行多次搜索。搜索字不需要全字匹配,包含部分文字即可。

monthly_instances(month, appointments=0, events=0, anniversaries=0, todos=0)

参数依次是 month(浮点数)和可选参数。

可选参数用于指定记录类型。返回一个列表,包含指定月份的所有实例。

daily_instances(day, appointments=0, events=0, anniversaries=0, todos=0)

参数依次是 day(浮点数)和可选参数。

可选参数用于指定记录类型。返回一个列表,包含指定日期的所有实例。

add_todo_list([name=None])

新建一个待办事项表,命名为 name。返回ID。

export_vcalendars((int,...))

以vCalendar格式导出记录。
参数是记录ID。

import_vcalendars(string)

导入vCalendar记录。返回记录ID。

todo_lists

包含类似字典的 TodoListDict 对象,借以访问数据库中的待办事项。

__delitem__(id)

从数据库中删除日历项,由参数 id 指定记录ID。

__getitem__(id)

返回Entry对象,由参数 id 指定记录ID。返回对象是 AppointmentEntry, EventEntry, AnniversaryEntry, TodoEntry 其中之一。

`compact()`

压缩数据库文件。返回值是一个整型数,非0则表示压缩成功。

7.2.3 Entry对象

Entry对象用于实时可视化访问数据库中的记录(根据ID)。若用`db.add_appointment`等方法创建一个新的记录,只有调用`commit`后才能保存至数据库。只要一个记录已被保存,那么`autocommit`模式将被默认开启,所有其他修改也将自动存入数据库。取消自动保存可使用`begin`方法。

注意,数据库中的记录无法被锁定,换句话说,即便使用了`begin`和`commit`方法,其他程序还是可以访问你正在使用的记录。当然,是针对记录本身而不是其对象。

Entry 类

Entry对象具有下列方法和特性:

`content`

设置或返回记录内容(Unicode文字)

`commit()`

将一个新记录添加至数据库。注意必须是`db.add_appointment`等方法创建的新记录,或者在调用`begin`方法之后才能调用`commit`。

`rollback()`

撤销上一个`commit`调用所做的修改。

`set_repeat(dictionary)`

设置记录的周期。*dictionary*是一个包含所有周期规则的字典。
关于周期规则详见章节7.3.4。

`get_repeat()`

返回周期规则字典。

`location`

设置或返回记录的位置信息。例如会议室信息。

`set_time(start[, end])`

设置记录的起讫时间(浮点值)。如果只给定一个时间,那么默认两者相同。对于备忘、纪念日和待办事项,时间值自动转化为日期值。

特别地, `TodoEntry` 可以用 `TodoEntry.set_time(None)` 方法设置为无限期。所谓无限期也就是移除起讫时间和所有周期规则。

`start_time`

设置记录的起始时间(浮点值)。不需设置起始时间则设为`None`。

`end_time`

设置记录的结束时间(浮点值)。不需设置结束时间则设为`None`。

`id`

记录ID

`last_modified`

当前记录上一次修改的时间(格林尼治时间)

alarm

设置记录的闹钟时间(浮点值)。不需设置或删除闹钟可将其设为None。

所有Entry类型都可以设置闹钟,但是只有Appointments和Anniversaries能够启用闹钟。这和诺基亚自身的“日历”程序类似,只有“会议”和“纪念日”能够启用闹钟。

另外注意,除了默认数据库中的记录闹钟之外,其他闹钟事实上也是无效的。

priority

记录优先级。可以是0到255的整型数。

对于诺基亚自身的“名片夹”和“日历”程序,1是高优先级,2是一般优先级,3是低优先级。

crossed_out

记录过期。为假表示记录未过期;为真表示记录已过期。

注意,待办事项必须有过期,相反其他记录类型不得有过期。

使用此方法,同时会将现场时间设为待办事项的过期时间。

参阅章节7.3.3, TodoEntry, cross_out_time.

replication

设置记录的复制权限。必须是'open', 'private'或'restricted'

(shagon注:分别是“开放”、“私有”、“受限”)

as_vcalendar()

以vCalendar格式返回记录。

AppointmentEntry

AppointmentEntry 类

AppointmentEntry 类从Entry类继承,本身不含附加方法。

EventEntry

EventEntry 类

EventEntry 类从Entry类继承,本身不含附加方法。

AnniversaryEntry

AnniversaryEntry 类

AnniversaryEntry 类从Entry类继承,本身不含附加方法。

TodoEntry

TodoEntry 类从Entry类继承,本身含有附加特性。

TodoEntry 类

TodoEntry 对象具有下列附加特性:

cross_out_time

记录的过期时间。必须是日期值(浮点数),None表示无过期。设定了过期时间就能使记录过期。

参阅章节7.3.3, Entry, crossed_out.

`todo_list`
当前记录所属的待办事项表ID。

`TodoListDict`

`TodoListDict` 是类似于字典的对象,用于访问待办事项表。

`TodoListDict` 类
`TodoListDict` 对象具有下列特性：

`default_list`
默认的待办事项表的ID

`TodoList`

`TodoList` 是类似于字典的对象,用于访问待办事项表。

`TodoList` 类
`TodoList` 对象具有下列特性：

`name`
待办事项表名称(Unicode)
`id`
待办事项表ID(整型数)

7.2.4 周期规则

周期规则规定了记录的循环情况。共有六种循环类型：

- `daily`: 每天
- `weekly`: 每周(比如周一、周三等)
- `monthly_by_dates`: 按日期每月(比如第15天、第17天等)
- `monthly_by_days`: 按星期每月(比如第四个周三、最后一个周一等)
- `yearly_by_date`: 按日期每年(比如12月24)
- `yearly_by_day`: 按星期每年(比如五月第三个周二)

周期规则可以有例外，也就是说即使制订了周期，也可以让一个记录处于循环之外。

周期的起始时间(浮点值)必须给定。结束日期可以是`None`,即无限循环。同时也可以规定循环间隔。例如对于`daily`, 1代表每天循环,2代表每两天循环；而对于`weekly`, [0,2]代表每周一和每周三循环。缺省时自动以起始日期作为循环基准。

如果需要修改周期规则可以调用`rep_data = entry.get_repeat()`, 首先改变`rep_data`字典, 然后调用`entry.set_repeat(rep_data)`。

如果需要撤销周期规则可以调用带假值参数的`entry.set_repeat`, 比如`entry.set_repeat(None)`。

周期规则范例如下页所示。

```

repeat = {"type": "daily", # 循环类型
          "exceptions": [exception_day, exception_day+2*24*60*60],
          # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
          "end": appt_start_date+30*24*60*60, # 循环结束
          "interval": 1}, # 循环间隔(例如:1等于每天,2等于每两天)

repeat = {"type": "weekly", # 循环类型
          "days": [0,1], # 按星期循环(周一、周二)
          "exceptions": [exception_day], # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
          "end": appt_start_date+30*24*60*60, # 循环结束
          "interval": 1}
          # 循环间隔(例如:1等于每周,2等于每两周)

repeat = {"type": "monthly_by_days", # 循环类型
          "days": [{ "week": 1, "day": 1 }, { "week": 4, "day": 0 }],
          # 每月第2个周二和第4个周一有周期事件
          "exceptions": [exception_day], # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
          "end": appt_start_date+30*24*60*60, # 循环结束
          "interval": 1}
          # 循环间隔(例如:1等于每月,2等于每两月)

repeat = {"type": "monthly_by_dates", # 循环类型
          "days": [0,15],
          # 每月第1天和第16天有周期事件
          "exceptions": [exception_day], # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
          "end": appt_start_date+30*24*60*60, # 循环结束
          "interval": 1}
          # 循环间隔(例如:1等于每月,2等于每两月)

repeat = {"type": "yearly_by_date", # 循环类型
          "exceptions": [exception_day], # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
          "end": appt_start_date+3*365*24*60*60, # 循环结束
          "interval": 1}
          # 循环间隔(例如:1等于每年,2等于每两年)

repeat = {"type": "yearly_by_day", # 循环类型
          "days": { "day": 1, "week": 1, "month": 1 },
          # 每年二月的第2个周二
          "exceptions": [exception_day], # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
          "end": appt_start_date+3*365*24*60*60, # 循环结束
          "interval": 1}
          # 循环间隔(例如:1等于每年,2等于每两年)

```

7.3 calendar(EKA2) — 日历服务

calendar 模块提供了日历服务API。它以一种类似字典的CalendarDb 对象描述议程数据库, 包含Entry 对象, 并以ID进行索引。一共有四种Entry对象, 分别是 :AppointmentEntry, EventEntry, AnniversaryEntry 和 TodoEntry。

CalendarDb 对象提供对数据库的实时访问。也就是说, 其他程序所做的更改能够立即在Python 程序中反映出来; 另一方面, Python程序所做的更改也能立即在其他程序中可见。

除个别外，时间参数都使用Unix时间。关于Unix时间详见章节3.5, 日期与时间。

7.3.1 模块函数

以下函数都定义在 `calendar` 模块中(同时没有定义在任何类中)：

`open([filename=None, mode=None])`

打开日历数据库并返回一个新的 `CalendarDb` 对象。

`filename` 缺省时打开默认数据库。如果给定 `filename` 则必须包含驱动器名、冒号和文件名, 但不是绝对路径。

`mode` 可以是：

- `None`: 打开一个现有的数据库。
- `'c'`: 打开一个现有的数据库, 如果不存在则创建一个指名的数据库。
- `'n'`: 新建一个空数据库, 如果已存在同名数据库则将其覆盖。

7.3.2 CalendarDb 对象

日历项和待办事项都保存在日历数据库中。默认的数据库只有一个，但是可以用带参数 `'n'` 或 `'c'` 的 `open` 函数创建新的数据库。

`CalendarDb` 类

`CalendarDb` 对象支持下列方法：

`add_appointment()`

创建并返回一个新的约会项 `AppointmentEntry`。
调用 `Entry.commit` 后添加并保存至数据库。

`add_event()`

创建并返回一个新的备忘项 `EventEntry`。
调用 `Entry.commit` 后添加并保存至数据库。

`add_anniversary()`

创建并返回一个新的纪念日项 `AnniversaryEntry`。
调用 `Entry.commit` 后添加并保存至数据库。

`add_todo()`

创建并返回一个新的待办事项 `TodoEntry`。
调用 `Entry.commit` 后添加并保存至数据库。

`add_reminder()`

创建并返回一个新的待办事项 `ReminderEntry`。
调用 `Entry.commit` 后添加并保存至数据库。

`find_instances(start_date, end_date, search_str=u"[,appointments=0,events=0,anniversaries=0,todos=0,reminder=0]")`

参数依次是起始日期、结束日期、搜索项和可选参数(指定记录类型)。在默认情况下搜索所有类型的记录。返回一个列表, 包含搜索到的 `Entry` 实例。一个 `Entry` 实例就是包含 ID 和日期的字典。
周期事件的记录含有多个实例, 比如每周事件。

`monthly_instances(month, appointments=0, events=0, anniversaries=0, todos=0, reminder=0)`

参数依次是 `month`(浮点数)和可选参数。
可选参数用于指定记录类型。返回一个列表, 包含指定月份的所有实例。

`daily_instances(day, appointments=0, events=0, anniversaries=0, todos=0)`

参数依次是 *day* (浮点数) 和可选参数。

可选参数用于指定记录类型。返回一个列表, 包含指定日期的所有实例。

`export_vcalendars((int,...))`

以vCalendar格式导出记录。

参数是记录ID。

`import_vcalendars(string)`

导入vCalendar记录。返回记录ID。

`__delitem__(id)`

从数据库中删除日历项, 由参数 *id* 指定记录ID。

`__getitem__(id)`

返回Entry对象, 由参数 *id* 指定记录ID。返回对象是 AppointmentEntry, EventEntry, AnniversaryEntry, TodoEntry 其中之一。

7.3.3 Entry对象

Entry对象用于实时可视化访问数据库中的记录(根据ID)。若用`db.add_appointment`等方法创建一个新的记录, 只有调用`commit`后才能保存至数据库。只要一个记录已被保存, 那么`autocommit`模式将被默认开启, 所有其他修改也将自动存入数据库。取消自动保存可使用`begin`方法。

注意, 数据库中的记录无法被锁定, 换句话说, 即便使用了`begin`和`commit`方法, 其他程序还是可以访问你正在使用的记录。当然, 是针对记录本身而不是其对象。

Entry 类

Entry 对象具有下列方法和特性:

`content`

设置或返回记录内容(Unicode文字)

`commit()`

将一个新记录添加至数据库。注意必须是`db.add_appointment`等方法创建的新记录, 或者在调用`begin`方法之后才能调用`commit`。

`rollback()`

撤销上一个`commit`调用所做的修改。

`set_repeat(dictionary)`

设置记录的周期。*dictionary*是一个包含所有周期规则的字典。
关于周期规则详见章节7.3.4。

`get_repeat()`

返回周期规则字典。

`location`

设置或返回记录的位置信息。例如会议室信息。

`set_time(start[, end])`

设置记录的起讫时间(浮点值)。如果只给定一个时间, 那么默认两者相同。对于备忘、纪念日
和待办事项, 时间值自动转化为日期值。

特别地, `TodoEntry` 可以用`TodoEntry.set_time(None)`方法设置为无限期。所谓无限期也就是
移除起讫时间和所有周期规则。

`start_time`
设置记录的起始时间(浮点值)。不需设置起始时间则设为None。

`end_time`
设置记录的结束时间(浮点值)。不需设置结束时间则设为None。

`id`
记录ID

`last_modified`
当前记录上一次修改的时间(格林尼治时间)

`originating`
一个整型数。表明当前记录是原始记录还是已被修改的记录。

`alarm`
设置记录的闹钟时间(浮点值)。不需设置或删除闹钟可将其设为None。
所有Entry类型都可以设置闹钟,但是只有Appointments和Anniversaries能够启用闹钟。这和诺基亚自身的“日历”程序类似,只有“会议”和“纪念日”能够启用闹钟。
另外注意,除了默认数据库中的记录闹钟之外,其他闹钟事实上也是无效的。

`priority`
记录优先级。可以从0到255的整型数。
对于诺基亚自身的“名片夹”和“日历”程序,1是高优先级,2是中优先级,3是低优先级。

`crossed_out`
记录过期。为假表示记录未过期;为真表示记录已过期。
注意,待办事项必须有过期,相反其他记录类型不得有过期。
使用此方法,同时会将现场时间设为待办事项的过期时间。
参阅章节7.3.3, TodoEntry, cross_out_time.

`replication`
设置记录的复制权限。必须是'open', 'private'或'restricted'
(shagon注:分别是“开放”、“私有”、“受限”)

`as_vcalendar()`
以vCalendar格式返回记录。

AppointmentEntry

AppointmentEntry 类

AppointmentEntry 类从Entry类继承, 本身不含附加方法。

EventEntry

EventEntry 类

EventEntry 类从Entry类继承, 本身不含附加方法。

AnniversaryEntry

AnniversaryEntry 类

AnniversaryEntry 类从Entry类继承, 本身不含附加方法。

ReminderEntry

ReminderEntry类

ReminderEntry类从Entry类继承，本身不含附加方法。

TodoEntry

TodoEntry类从Entry类继承，本身含有附加特性。

TodoEntry类

TodoEntry对象具有下列附加特性：

cross_out_time

记录的过期时间。必须是日期值(浮点数),None表示无过期。设定了过期时间就能使记录过期。
参阅章节7.3.3, Entry, crossed_out.

7.3.4 周期规则

周期规则规定了记录的循环情况。共有六种循环类型：

- daily: 每天
- weekly: 每周(比如周一、周三等)
- monthly_by_dates: 按日期每月(比如第15天、第17天等)
- monthly_by_days: 按星期每月(比如第四个周三、最后一个周一等)
- yearly_by_date: 按日期每年(比如12月24)
- yearly_by_day: 按星期每年(比如五月第三个周二)

周期规则可以有例外，也就是说即使制订了周期，也可以让一个记录处于循环之外。

周期的起讫时间(浮点值)必须给定。结束日期可以是None,即无限循环。同时也可以规定循环间隔。例如对于daily, 1代表每天循环,2代表每两天循环；而对于weekly, [0,2]代表每周一和每周三循环。缺省时自动以起始日期作为循环基准。

如果需要修改周期规则可以调用`rep_data = entry.get_repeat()`, 首先改变`rep_data`字典, 然后调用`entry.set_repeat(rep_data)`。

如果需要撤销周期规则可以调用带假值参数的`entry.set_repeat`, 比如`entry.set_repeat(None)`。

周期规则范例如下：

```
repeat = {"type": "daily", # 循环类型
          "exceptions": [exception_day, exception_day+2*24*60*60],
          # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
```

```

        "end": appt_start_date+30*24*60*60, # 循环结束
        "interval": 1}, # 循环间隔 (例如: 1等于每天, 2等于每两天)

repeat = {"type": "weekly", # 循环类型
          "days": [0, 1], # 按星期循环(周一、周二)
          "exceptions": [exception_day], # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
          "end": appt_start_date+30*24*60*60, # 循环结束
          "interval": 1}
          # 循环间隔 (例如: 1等于每周, 2等于每两周)

repeat = {"type": "monthly_by_days", # 循环类型
          "days": [{"week": 1, "day": 1}, {"week": 4, "day": 0}],
          # 每月第2个周二和第4个周一有周期事件
          "exceptions": [exception_day], # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
          "end": appt_start_date+30*24*60*60, # 循环结束
          "interval": 1}
          # 循环间隔 (例如: 1等于每月, 2等于每两月)

repeat = {"type": "monthly_by_dates", # 循环类型
          "days": [0, 15],
          # 每月第1天和第16天有周期事件
          "exceptions": [exception_day], # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
          "end": appt_start_date+30*24*60*60, # 循环结束
          "interval": 1}
          # 循环间隔 (例如: 1等于每月, 2等于每两月)

repeat = {"type": "yearly_by_date", # 循环类型
          "exceptions": [exception_day], # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
          "end": appt_start_date+3*365*24*60*60, # 循环结束
          "interval": 1}
          # 循环间隔 (例如: 1等于每年, 2等于每两年)

repeat = {"type": "yearly_by_day", # 循环类型
          "days": {"day": 1, "week": 1, "month": 1},
          # 每年二月的第2个周二
          "exceptions": [exception_day], # 这些日期无周期事件
          "start": appt_start_date, # 循环起始
          "end": appt_start_date+3*365*24*60*60, # 循环结束
          "interval": 1}
          # 循环间隔 (例如: 1等于每年, 2等于每两年)

```

7.4 e32db —Symbian 数据库接口

e32db 模块提供了访问关系数据库的API, 使用有限的SQL语法进行访问。

关于DBMS支持详见S60 SDK文档。关于本模块的使用范例详见[6]。

(shagon注: DBMS, Data Base Management System, 数据库管理系统)

e32db 模块支持下列函数:

`format_rawtime(timevalue)`

以当前系统日期/时间格式来格式化 *timevalue* (Symbian时间), 并返回一个Unicode字符串。

`format_time(timevalue)`

以当前系统日期/时间格式来格式化 *timevalue*, 并返回一个Unicode字符串。

7.4.1 Dbms对象

Dbms()类

创建一个Dbms对象。它支持基本的数据库操作。

Dbms对象支持下列方法：

begin()

开始操作数据库

close()

关闭数据库对象。即便是对于未打开的数据库对象这个方法也是安全的。

commit()

提交当前数据库操作

compact()

压缩数据库,收回数据库文件中未使用的空间

create(*dbname*)

删除一个数据库,*dbname*即其路径。

execute(*query*)

执行一次SQL查询。如果执行DDL指令(更新SQL图表)则返回0,如果执行DML指令(更新SQL数据)则返回一个数字代表行数。

open(*dbname*)

打开一个数据库,*dbname*必须是完整路径,形如u'c:\\foo.db'.

rollback()

回滚。即撤销当前操作,并恢复先前状态

7.4.2 DB_view对象

Db_view()类

创建一个Db_view对象。它用于生成SQL查询行集合,并且对其进行分析。

Db_view对象支持下列方法：

col(*column*)

返回列值。行集合的第一列索引号是1。如果当前类型不被支持则引起TypeError。
可支持的数据类型参阅表7.1

col_count()

返回行集合的列数

col_length(*column*)

返回列长。空列的列长为0;非空数字列和日期/时间列的列长为1;文本列的列长是字符串总长;二进制列的列长是字节数。

col_raw(*column*)

以二进制数据提取列值并以Python字符串返回。
行集合的第一列索引号是1。可支持的数据类型参阅表7.1

col_rawtime(*column*)

以长整型提取日期/时间列值作为原始Symbian时间值。
行集合的第一列索引号是1。可支持的数据类型参阅表7.1

col_type(*column*)

返回给定列的类型,以整型值表示。
此函数在调用col方法时使用。

`count_line()`
 返回行集合的有效行数
`first_line()`
 将光标移至行集合的第一行位置
`get_line()`
 获取当前行数据
`is_col_null(column)`
 校验当前列是否为空。空列可以像普通列一样访问。
 数字空列返回0或等价值,文本和二进制列返回长度0。
`next_line()`
 将光标移至行集合的下一行
`prepare(db, query)`
 为SQL选择指令做先期准备。*db*是Dbms对象,而*query*是待执行的SQL查询语句。

7.4.3 SQL与Python数据类型映射关系

从表7.1可以看到SQL与Python数据类型之间映射关系的一个概要。`col`函数可以提取除了LONG VARBINARY以外的所有值,并以合适的Python值返回。而`col_raw`函数可以提取除了LONG VARCHAR和LONG VARBINARY以外的所有列类型为二进制数据,并以Python字符串返回。

插入、更新、查找等操作对BINARY,VARBINARY,LONGVARBINARY等无效。
 BINARY和VARBINARY可以用`col`或`col_raw`函数读取。

SQL类型	Symbian 列类型 (基于DBMS C++ API)	Python类型	是否支持
BIT	EDbColBit	int 整型	是
TINYINT	EDbColInt8		
UNSIGNED TINYINT	EDbColUInt8		
SMALLINT	EDbColInt16		
UNSIGNED SMALLINT	EDbColUInt16		
INTEGER	EDbColInt32		
UNSIGNED INTEGER	EDbColUInt32		
COUNTER	EDbColUInt32 (具有TDbCol::EAutoIncrement 属性)		
BIGINT	EDbColInt64	long 长整型	
REAL	EDbColReal32	float 浮点型	
FLOAT	EDbColReal64		
DOUBLE			
DOUBLE PRECISION			
DATE			
TIME	Unicode		
TIMESTAMP			
CHAR(n)		EDbColText	
VARCHAR(n)			
LONG VARCHAR	EDbColLongText		
BINARY(n)	EDbColBinary	str 字符串	只读
VARBINARY(n)			
LONG VARBINARY	EDbColLongBinary	不可用	否

表7.1 SQL与Python数据类型映射关系

7.4.4 日期与时间处理

`col`和`format_time`函数都使用Unix时间格式,即按秒从格林尼治时间1970年1月1日零时零分零秒开始计时。不过数据库所使用的Symbian时间比Unix时间具有更高精确度和更大范围。Symbian特有的时间格式是64位数值,按微秒从公元0年1月1日零时零分零秒(以教皇格利高里为名)开始计时,并用负数表示公元前。由于这种格式和Unix时间互相转换可能产生一定误差,如果需要最佳精确度可以使用`col_rawtime`和`format_rawtime`函数。

SQL指令中日期与时间的具体含义依赖于当前系统的日期与时间格式。另外注意,年月日的排列顺序必须是当前系统设置的那一种,否则不予接受。可用`format_time`和`format_rawtime`函数将时间转换为当前系统可接受的时间格式。

7.5 e32dbm — 使用Symbian系统DBMS的DBM工具

`e32dbm`模块提供了DBM的API,使用Symbian RDBMS作为其存储后端。

该模块的API类似于`gdbm`模块,主要不同之处在于:

- 不支持`firstkey()` - `nextkey()`接口来指示键。可用`"for key in db"`语法或者`keys,keysiter`方法。
- 较`gdbm`支持更完整的字典功能。
- 通常用Unicode保存值,也就是即便DBM取到的是一般字符串也以Unicode字符串返回。

7.5.1 模块函数

`e32dbm`支持下列函数:

`open(dbname[,flags, mode])`

打开或创建数据库并返回一个`e32dbm`对象。`dbname`必须是完整路径,形如`u'c:\\foo.db'`。

`flag`可以是:

- `'r'`: 以只读方式打开一个已存在的数据库。这是默认模式。
- `'w'`: 以读写方式打开一个已存在的数据库。
- `'c'`: 以读写方式打开一个数据库,如果不存在则创建一个同名数据库。
- `'n'`: 以读写方式创建一个新的空数据库。

如果`flag`后再追加一个`'f'`则以`fast mode`(快速模式)打开数据库。在快速模式中,只有调用`sync`,`close`,`reorganize`,`clear`等其中一个函数才会对数据库进行更新和写操作。

事实上销毁机制本身会自动调用`close`,不过建议亲自使用`close`方法来关闭数据库以确保数据成功保存。关闭数据库之后将解除锁定,这样不必退出解释器就能重开或删除文件。

如果需要多次更新,那么强烈建议使用快速模式。这是因为插入和更新操作次数多时一起处理效率较高。特别是处理大量数据时,对于`e32db`来说每次只插入一个记录效率是非常低的。

7.5.2 e32dbm 对象

e32dbm 对象由 open 函数返回，它支持大多数的标准字典方法。

可支持的字典方法有：

- __getitem__
- __setitem__
- __delitem__
- has_key
- update
- __len__
- __iter__
- iterkeys
- iteritems
- intervalues
- get
- setdefault
- pop
- popitem
- clear

以上工作方式与一般字典是相应的。

另外，e32dbm 对象还支持下列方法：

close()

关闭数据库。在快速模式中将向磁盘提交所有未决更新。

试图关闭未打开的数据库将引起异常。

reorganize()

重组数据库。对于 e32db 数据库文件将自动调用 compact 方法进行压缩，收回文件中未使用的空间。建议在多次更新之后调用 reorganize()。

sync()

在快速模式中，向磁盘提交所有未决更新。

标准库支持及扩展

8.1 Python标准库支持

表8.1中给出了Python for S60标准库支持的概要。关于API描述请参阅[1]。

名 称	类 型	支 持	备 注
_testcapi	PYD	Y	
anydbm	PY	X	DBM API是依赖于PYD e32db的PY e32dbm的工具(详见章节7.5, e32dbm模块)
atexit	PY	X	
base64	PY	X	
bdb	PY	(X)	
binascii	内 建	X	
cmd	PY	(X)	
code	PY	X	
codecs	PY	X	
codeop	PY	X	
copy	PY	X	
copy_reg	PY	X	
cStringIO	内 建	X	
dis	PY	(X)	
errno	内 建	X	
exceptions	内 建	X	
__future__	PY	X	
httplib	PY	X	
imp	内 建	X	
keyword	PY	X	
linecache	PY	X	
marshal	内 建	X	
math	内 建	X	
md5 ¹	内 建	X	
mimertools	PY	X	
operator	内 建	X	
os, os.path	PY	X	封装内建e32posix。在章节3.9(目前的局限和未来的发展)有一定探讨。
pdb	PY	(X)	
quopri	PY	X	
名 称	类 型	支 持	备 注

¹源自RSA Data Security公司的MD5(Message-Digest)算法

random	PY	X	
re	PY	X	使用PY作为其引擎
repr	PY	X	
rfc822	PY	X	
select	PY	X	最小化执行:只支持来自套接字中的输入
socket	PY	X	需要PYD e32socket支持。包含章节8.2.2提到的扩展内容。其他讨论详见章节3.9,目前的局限和未来的发展。
sre	PY	X	封装内建_sre
string	PY	X	
StringIO	PY	X	
struct	内 建	X	
sys	内 建	X	
thread	内 建	X	包含章节8.2.1提到的扩展内容
threading	PY	(X)	
time	内 建	X	
traceback	PY	X	
types	PY	X	
urllib	PY	X	
urlparse(仅urlsplit)	PY	X	
uu	PY	X	
warnings	PY	X	
whichdb	PY	X	
xreadlines	内 建	X	
zipfile	PY	X	
zlib	PYD	X	

表8.1 库模块支持情况

表8.1中“类型”的意义分别为：

- PY —在Python中执行
- 内建 —内建的C/C++模块
- PYD—动态加载的C/C++模块

表8.1中“支持”的意义分别为：

- X —包含于PyS60
- (X)—不包含于PyS60,但可以同时在手机和SDK上正常运行
- Y —只包含于PyS60 SDK

8.2 标准库模块扩展

以下标准模块已得到扩展。

8.2.1 thread — 基于S60的标准thread模块扩展

以下函数已被添加到标准thread模块中：

`ao_waittid(thread_id)`

当由`thread_id`指定的线程结束时对其进行同步。该操作依赖于Symbian系统AO。

更多信息详见章节4.1.2, `Ao_lock`类型

8.2.2 socket — 基于S60的标准socket模块扩展

标准socket模块已添加蓝牙(Bluetooth,BT)功能。相关常量和函数定义如下：

注意：在1.0版本中函数`bt_advertise_service`, `bt_obex_receive`, 以及.....
`bt_rfcomm_get_available_server_channel`曾以`e32socket.socket`对象代替`socket`对象作为参数，
这是不恰当的, 现在已经更正。为了保留使用习惯, 旧方法仍然允许, 但可能在将来的版本中彻底
移除。

`AF_BT`

描述蓝牙地址簇

`BTPROTO_RFCOMM`

该常量描述蓝牙协议RFCOMM

`RFCOMM`

`OBEX`

蓝牙服务类。由`bt_advertise_service`支持。

`AUTH`

`ENCRYPT`

`AUTHOR`

蓝牙安全模式标记

`bt_advertise_service(name, socket, flag, class)`

设置蓝牙广播。`name`是服务端, 运行在由`socket`划定的本地频道上。

`flag`为真则开启广播, 为假则关闭广播。`class`是RFCOMM或OBEX。

`bt_discover([address])`

搜索未知的蓝牙设备或者已知蓝牙设备的RFCOMM服务。返回蓝牙字典和服务字典。在服务字典中, 键是服务名称, 值是通讯端口。

`bt_obex_discover([address])`

与`bt_discover`类似, 不同之处是搜索已经蓝牙设备的OBEX服务。

`bt_obex_send_file(address, channel, filename)`

将`filename`文件封装到一个OBEX对象并发送至远程端(`address, channel`)。

`bt_obex_receive(socket, filename)`

接受一个OBEX对象并保存其携带的`filename`文件。`socket`是一个限定的OBEX套接字。

`bt_rfcomm_get_available_server_channel(socket)`

为`socket`返回一个有效的RFCOMM服务。

`set_security(socket, mode)`

设置给定`socket`的安全级别。`mode`是整型标记, 采用二元或多操作数方式。操作数可以是：
`AUTH(authentication, 认证)`, `ENCRYPT`, `AUTHOR(authorization, 授权)`中的一个。

例如：`set_security(s, AUTH | AUTHOR)`。

注 意：在手机上使用蓝牙功能，很有必要设置安全级别。

注 意：SSL不被S60 1st 版本支持。SSL客户端证书不被支持。

以上函数的使用范例参阅Programming with Python for S60 Platform [6]

标准socket模块已添加设置默认入口点(Access Point,AP)功能。

相关常量和函数定义如下：

`select_access_point()`

从可用入口点列表中弹出选项并打开。返回选中的入口点ID。

`access_point(apid)`

创建入口点对象并返回

`set_default_access_point(apo)`

设置默认入口点。如果*apo*设置为None则清除默认入口点。

`access_points()`

列出可用的入口点ID和名称。

示 例1：

```
#从列表中选择入口点
apid = select_access_point()
apo = access_point(apid)
set_default_access_point(apo)

s = socket(AF_INET, SOCK_STREAM)
print apo.ip()
s.connect(('www.sourceforge.net',80))
s.send('GET /\r\n\r\n')
s.recv(100)
s.close()
apo.stop()
```

示 例2：

```
#入口点ID已知
apo = access_point(1)
set_default_access_point(apo)

s = socket(AF_INET, SOCK_STREAM)
s.connect(('www.sourceforge.net',80))
s.send('GET /\r\n\r\n')
s.recv(100)
s.close()
apo.stop()
```

示 例3：

```
#显示接口IP
#从列表中选择入口点
apid = select_access_point()
apo = access_point(apid)
apo.start()
#如果未定义静态IP地址,那么连接之后将由操作符给出IP地址
print apo.ip()
#连接断开后将释放动态IP地址
apo.stop()
```


扩展和内嵌

9.1 Python/C API 扩展

S60解释程序特有的API包括CSPyInterpreter类,Python/C API(参阅[3])和一些扩展API。

9.1.1 CSPyInterpreter 类

CSPyInterpreter 类提供了初始化解释器和运行脚本的接口。

它提供以下公用接口：

```
static CSPyInterpreter*
NewInterpreterL(TBool aCloseStdlib = ETrue,
               void(*aStdioInitFunc)(void*) = NULL,
               void* aStdioInitCookie = NULL);
TInt RunScript(int argc, char** argv);
void PrintError();
void (*iStdI)(char* buf, int n);
void (*iStdO)(const char* buf, int n);
```

构造器CSPyInterpreter::NewInterpreterL(可以调用其自身的aStdioInitFunc函数来初始化Symbian系统STDLIB的标准输入/输出描述符。它也可以随参数 aStdioInitCookie 被调用。CSPyInterpreter类可以被请求以STDLIB作为析构器。

RunScript 方法用于建立Python解释器上下文环境，并运行参数argv中指定的脚本程序。完成后即脱离解释器并返回校验码以指示运行成功或失败。

CSPyInterpreter::PrintError 方法用于标准错误输出，显示当前Python异常信息。

9.1.2 Python/C API 扩展

symbian_python_ext_util.h 定义内容

```
PyObject* SPyErr_SetFromSymbianOSErr(int error)
```

用Symbian系统枚举值error的符号名来设置Python异常类型PyExc_SymbianError并返回NULL。

如果error是特殊值KErrPython,那么就假定Python异常已被设定并返回NULL。

以下函数可以用于保存模块中的全局数据。它们分别被浅封装在PyDict_SetItem,PyDict_SetItemString,PyDict_GetItem,PyDict_GetItemString,PyDict_DelItem和PyDict_DelItemString中，使用方法相同。这些数据事实上被存放在一个特殊的全局字典中，并为解释器中所有模块和线程所共享。

```

int SPyAddGlobal(PyObject *key, PyObject *value)
int SPyAddGlobalString(char *key, PyObject *value)
PyObject* SPyGetGlobal(PyObject *key)
PyObject* SPyGetGlobalString(char *key)
void SPyRemoveGlobal(PyObject *key)
void SPyRemoveGlobalString(char *key)

```

python_globals.h 定义内容

```

PyThreadState* PYTHON_TLS->thread_state
    当前线程状态

```

线程状态和解释器锁存管理必须遵循这个定义。参阅[3]。

PyS60平台提供了查询相应线程状态的简易操作，以此扩展了Python/C API。

Python线程状态源自当前运行线程的上下文。

这可用于以C/C++代码PyEval_RestoreThread 来重建解释器上下文。

保存/重建解释器上下文：

```

Py_BEGIN_ALLOW_THREADS
/* 你的代码 */
Py_END_ALLOW_THREADS

```

重建/保存解释器上下文：

```

PyEval_RestoreThread(PYTHON_TLS->thread_state)
/* 你的代码 */
PyEval_SaveThread()

```

pythread.h 定义内容

```

int PyThread_AtExit(void(*)())
    这是标准 thread 模块C API的扩展, 可用于寄存线程特有的出口函数。
    对于主线程调用本函数和Py_AtExit效果相同。详见[1]。

```

9.2 Python for S60 扩展

编写Python扩展的基本规则和指导方针同样适用于PyS60环境。详见[2]。

有效Python/C API详见[3]; S60扩展示例详见[6]。

针对扩展模块，值得考虑的问题有：

- 数据结构。必须保证C/C++编写的扩展在Python解释器中可见并能有效执行。
- C/C++所描述的Python对象与扩展代码中对象类型之间的互相转换问题。
- 维持C/C++所描述的Python对象的引用计数。
- C/C++与Python之间异常消息的传递。
- 处理解释器线程状态和解释器锁存。

在使用S60环境下新的Python接口时,除了Python C扩展的一些常规注意事项外,以下原则也值得考虑:

- 尽量使用Python内建的接口。
- 基于上面一条,在设计新的接口时尽量使其与内建接口保持较少的转换。
- 将Symbian操作系统的异常/错误类型全部转换为Python异常类型。
- Unicode是GUI中使用的法定文字编码。在Symbian操作系统中使用Unicode字符串无需转换。
- 在调用解释器以外的服务时,如果耗时/阻塞较长,那么必须释放解释器锁存,并在调用结束后重新申请。
- 与其总使用Symbian系统顶层的浅封装工具,不如试想一下开发者在编写代码时真正所需的接口。比如说,要编写一个图形用户界面程序,开发者感兴趣的只是如何向用户显示信息并实现人际互动,而不会去关心GUI底层是如何实现工作的。
- C/C++编写的Python接口应该是最优化的,并且针对相关平台提供所有可能的接口。如果有必要,可以直接用Python编写更精致的封装模块。

一个扩展模块必须打包成动态加载库,并且安装到‘\system\libs’目录下,以‘module_name.pyd’命名。模块的初始化函数必须是序列为1的函数。模块只能通过文件名识别。作为PyS60的一个特性,可选模块终结器函数输出序号可能为2。

大型版本的PyMem_MALLOC和PyObject_NEW函数没有被引入,可以使用PyMem_Malloc, PyObject_New代替。

9.2.1 扩展服务

S60 Python平台是S60 UI应用程序框架与脚本语言简化UI扩展开发的阶梯。这个API在appuifw模块中执行。同时也有一些有效的通用扩展服务,详见章节9.1。

9.2.2 示例

下页代码片段演示了本章所涉及的一些讨论:

- Python数据类型转换;内建数据类型在扩展接口中的使用;Unicode字符串的使用(第8至12行)。
- 维持引用计数(第36行)。
- C/C++与Python之间异常消息的传递(第34行)。
- 调用解释器以外的服务时,释放解释器锁存(第29,31行)。
- 简化相应平台的提示信息API。

```

01 extern "C" PyObject *
02 note(PyObject* /*self*/, PyObject *args)
03 {
04     TInt error = KErrNone;
05     int l_tx, l_ty;
06     char *b_tx, *b_ty;
07
08     if (!PyArg_ParseTuple(args, "u#s#", &b_tx, &l_tx, &b_ty, &l_ty))
09         return NULL;
10
11     TPtrC8 stype((TUint8*)b_ty, l_ty);
12     TPtrC note_text((TUint16 *)b_tx, l_tx);
13     CAknResourceNoteDialog* dlg = NULL;
14
15     if (stype.Compare(KErrorNoteType) == 0)
16         dlg = new CAknErrorNote(ETrue);
17     else if (stype.Compare(KInfoNoteType) == 0)
18         dlg = new CAknInformationNote(ETrue);
19     else if (stype.Compare(KConfNoteType) == 0)
20         dlg = new CAknConfirmationNote(ETrue);
21     else {
22         PyErr_BadArgument();
23         return NULL;
24     }
25
26     if (dlg == NULL)
27         return PyErr_NoMemory();
28
29     Py_BEGIN_ALLOW_THREADS
30     TRAP(error, dlg->ExecuteLD(note_text));
31     Py_END_ALLOW_THREADS
32
33     if (error != KErrNone)
34         return SPyErr_SetFromSymbianOSError(error);
35     else {
36         Py_INCREF(Py_None);
37         return Py_None;
38     }
39 }

```

以下是本文档所用到的术语：

术 语	定 义
AAC(Adaptive Audio Coding)	ACC音质接近MP3,但使用更小的位速率。主要应用于压缩音乐。
Advertise (广播)	蓝牙广播服务,服务端向其他蓝牙设备告知其已正常工作。
AMR	Adaptive Multi-rate Codec(自适应多速率编码)文件格式
API	Application Programming Interface(应用程序接口)
Bluetooth(蓝牙)	蓝牙是一种应用于多个设备之间的低耗短距无线传输技术。
BPP(色深)	Bits Per Pixel(每个像素所占用位)
C STDLIB	Symbian系统下执行的C标准库
Dialog(对话框)	一个临时性的用户界面窗口,向用户提供信息和获取用户输入。
Discovery(搜寻)	搜寻附近的蓝牙设备,并获知它们所提供的服务。
DLL	Dynamic link library(动态链接库)
GSM (Global System for Mobile communication)	全球移动通信系统。采用分时通信技术,将数据数字化并压缩,偕同其他两个用户数据流(分别工作在各自的时间槽)在信道上传输。
GUI	Graphical User Interface(图形用户界面)
I/O	input/output(输入/输出)
IP	Internet Protocol(网际协议)
MBM (Multi BitMap)	Symbian系统独特的图像格式。SDK工具bmconv.exe可制作MBM文件。
MIDI (Musical Instrument Digital Interface)	电子乐器数字化接口。电子乐器和电脑之间信息传送的标准之一。
MIF (Multi Image File)	类似于MBM,包含SVG-T压缩文件。MifConv.exe可制作MIF文件。
MIME (Multi purpose Internet Mail Extensions)	MIME是早期Email协议的扩展方式之一,它允许在互联网上交换不同类型的数据文件。
MP3	音乐文件压缩的标准技术和格式,压缩率较大,音质较好。
OS	Operating System(操作系统)
Real Audio	Real Networks公司的一种音频格式
RDBMS	Relational database management system(关系型数据库管理系统)
SMS (Short Message System)(GSM网络)	GSM网络短信息服务。允许手机之间发送不超过160个字母(5位制式则是224个字母)的文字信息。

术 语	定 义
Softkey	软键。不对应于固定的函数,具体功能需查看屏幕显示,按提示操作。
SQL	Structured Query Language(结构化查询语言)
SVG, SVG-T; (Scalable Vector Graphics (-Tiny))	基于XML的可缩放矢量图形标准(简化标准),描绘二维图象和图形程序。
Twip(缇)	缇是屏幕显示系统的计量单位。1英寸等于1440缇,1厘米等于567缇。
UI	User Interface(用户界面)
UI control (UI 控件)	UI控件是GUI的组成部件,用于实现交互式操作。
WAV	录音文件格式之一,常用于多媒体程序。

参考资料

- [1] [Python] Library Reference 编者:G.van Rossum , F.L.Drake , Jr.
<http://www.python.org/doc>
- [2] Extending and Embedding [the Python Interpreter] 编者:G.van Rossum , F.L.Drake , Jr.
<http://www.python.org/doc>
- [3] Python/C API [Reference Manual] 编者:G.van Rossum , F.L.Drake , Jr.
<http://www.python.org/doc>
- [4] S60 SDK文档, <http://www.forum.nokia.com/>
- [5] Getting Started with Python for S60 Platform, <http://www.forum.nokia.com/>
- [6] Programming with Python for S60 Platform, <http://www.forum.nokia.com/>
- [7] 诺基亚论坛网站, 声音和视频部分(基于诺基亚手机)
<http://www.forum.nokia.com/audiovideo>
- [8] S60平台网站, 开发者部分(基于所有S60设备), <http://www.s60.com/>
- [9] PyS60编程爱好者版块, <http://discussion.forum.nokia.com/>
- [10] 可缩放矢量图形标准(Scalable Vector Graphics, SVG)1.1规范描述
<http://www.w3.org/TR/SVG/>

BUG报告

为了帮助PyS60不断改进，如果您在PyS60套件或本文档发现任何BUG，都欢迎报告给开发者。

首先您需要在SourceForge注册帐号，这样才能提交报告并可以让开发者与您取得进一步联系。

所有报告都通过PyS60 Bug Tracker提交(http://sourceforge.net/tracker/?group_id=154155)，相关信息将被转寄给开发者。

发送报告之前请您先确认网站是否已有类似问题的讨论。这不仅是为了节约开发者的宝贵时间，同时对您也是一个学习的机会。通常有些BUG在新版本中已被清除，或者其他编程爱好者已经找到可行的解决办法，当然您也可以尝试亲自来解决问题并与大家分享！

您可以通过页面上的搜索条来查阅数据库中已有的BUG。

如果您发现的BUG还未收录在数据库里面，请到Bug Tracker选择页面顶部的“Submit a Bug”提交您的报告。

报告表单含有多项条目，必填的只有“Summary”和“Details”。在Summary(摘要)中请用尽可能简短的文字作概述，少于十个单词为佳。在Details(详情)中请详细描述您所遇到的问题，包括您预期的结果和实践所得的结果。另外请说明您所使用的PyS60版本、扩展模块、硬件(手机真机或模拟器)、S60 SDK版本以及您的设备固件信息(在手机上输入*#0000#，将显示文字全部写明)。

如果希望报告存入相应版块，那么请在“Category”栏里填写BUG种类(比如“Documentation”，“Library”)。

每个报告都会被转寄给开发者，他们将尝试解决问题。如果有所进展，您会第一时间得到通知。

另外请参阅：

How to Report Bugs Effectively

(<http://www.mice.cs.ucl.ac.uk/multimedia/software/documentation/ReportingBugs.html>)

文章讲述如何填写有用的BUG报告。您将知道什么样的报告是有用的，以及它为什么有用。

Bug Writing Guidelines

(<http://www.mozilla.org/quality/bug-writing-guidelines.html>)

讲述如何填写优秀的BUG报告。

虽然部分内容是Mozilla自身实例，不过其中涉及的思想很值得借鉴。

shagon注 如果您在这部中译本发现翻译错误，请按如下方式反馈，感谢您的支持！

邮箱：shagon@163.com

网站：<http://shagon.51.com>

置顶文章

模块索引

appuifw, 13
audio, 49

calendar, 64, 69
camera, 33
contacts, 59

e32, 9
e32db, 74
e32dbm, 77

glcanvas, 45
gles, 38
graphics, 27

inbox, 53

keycapture, 36

location, 54

messaging, 52

positioning, 55

sensor, 46
socket, 81
sysinfo, 11

telephone, 51
thread, 81
topwindow, 37

索引

- __del__() (EventFilter 方法),48
- __delitem__() (CalendarDb 方法),65,71
- __delitem__() (ContactDb 方法),60
- __delitem__() (Contact 方法),62
- __getitem__() (CalendarDb 方法),65,71
- __getitem__() (array 方法),39
- __init__() (EventFilter 方法),48
- __init__() (OrientationEventFilter 方法),48
- __init__() (Sensor 方法),47
- __len__() (array 方法),39
- __setitem__() (array 方法),39
- access_point() (socket 模块),82
- access_points() (socket 模块),82
- activate_tab() (Application 方法),18
- active_profile() (sysinfo 模块),11
- add() (Text 方法),22
- add_anniversary() (CalendarDb 方法),65,70
- add_appointment() (CalendarDb 方法),65,70
- add_contact() (ContactDb 方法),59
- add_event() (CalendarDb 方法),65,70
- add_field() (Contact 方法),61
- add_group() (Groups 方法),63
- add_image() (TopWindow 方法),37
- add_reminder() (CalendarDb 方法),70
- add_todo() (CalendarDb 方法),65,70
- add_todo_list() (CalendarDb 方法),65
- address() (Inbox 方法),53
- AF_BT (socket 模块数据),81
- after() (Ao_timer 方法),11
- alarm (Entry 属性),67,72
- all_keys (keycapture 模块数据),36
- AnniversaryEntry (calendar 模块类),67,72
- answer() (telephone 模块),51
- ao_callgate() (e32 模块),9
- Ao_lock (e32 模块类),10
- ao_sleep() (e32 模块),9
- Ao_timer (e32 模块类),11
- ao_waittid() (thread 模块),81
- ao_yield() (e32 模块),9
- Application (appuifw 模块类),16
- AppointmentEntry (calendar 模块类),67,72
- appuifw (standard 模块),13
- arc() (方法),33
- array (gles 模块类),39
- as_vcalendar() (Entry 属性),67,72
- as_vcard() (Contact 方法),61
- audio (扩展模块),49
- AUTH (socket 模块数据),81
- AUTHOR (socket 模块数据),81
- available_fonts() (appuifw 模块),15
- background_color (TopWindow 属性),38
- battery() (sysinfo 模块),11
- begin()
 - Contact 方法,60
 - Dbms 方法,75
- bind()
 - GLCanvas 方法,46
 - Inbox 方法,54
 - Listbox 方法,24
 - Text 方法,22
- blit() (方法),33
- body (Application 方法),16
- bt_advertise_service() (socket 模块),81
- bt_discover() (socket 模块),81
- bt_obex_discover() (socket 模块),81
- bt_obex_receive() (socket 模块),81
- bt_obex_send_file() (socket 模块),81
- bt_rfcomm_get_available_server_channel()
 - (socket 模块),81
- BTPROTO_RFCOMM (socket 模块数据),81
- calendar (扩展模块),64,69
- CalendarDb (calendar 模块类),65,70
- call_state() (telephone 模块),51
- callback (EventFilter 属性),48
- camera (扩展模块),33
- cameras_available() (camera 模块),34
- cancel() (Ao_timer 方法),11
- Canvas (appuifw 模块类),25
- cleanup()
 - EventFilter 方法,48
 - OrientationEventFilter 方法,48
- clear()
 - 方法,33
 - Text 方法,22
- close()
 - Dbms 方法,75
 - e32dbm 方法,78
 - Sound 方法,50
- col() (Db_view 方法),75

col_count() (Db_view 方法),75
 col_length() (Db_view 方法),75
 col_raw() (Db_view 方法),75
 col_rawtime() (Db_view 方法),75
 col_type() (Db_view 方法),75
 color (Text 方法),21
 commit()
 Contact 方法,60
 Dbms 方法,75
 Entry 方法,66,71
 compact()
 CalendarDb 方法,66
 ContactDb 方法,60
 Dbms 方法,75
 compact_required() (ContactDb 方法),60
 connect() (Sensor 方法),47
 connected() (Sensor 方法),48
 Contact (contacts 模块类),60
 ContactDb (contacts 模块类),59
 ContactField (contacts 模块类),63
 contacts (扩展模块),59
 content() (Inbox 方法),53
 content (Entry 属性),66,71
 Content_handler (appuifw 模块类),24
 corner_type (TopWindow 属性),38
 count_line() (Db_view 方法),76
 create() (Dbms 方法),75
 cross_out_time (TodoEntry 属性),67,73
 crossed_out (Entry 属性),67,72
 current() (Listbox 方法),24
 current_position() (Sound 方法),51
 current_volume() (Sound 方法),50

 daily_instances() (CalendarDb 方法),65,71
 Db_view (e32db 模块类),75
 Dbms (e32db 模块类),75
 default_list (TodoListDict 属性),68
 default_module() (positioning 模块),55
 delete()
 Inbox 方法,54
 Text 方法,22
 dial() (telephone 模块),51
 disconnect() (Sensor 模块),47
 display_pixels() (sysinfo 模块),11
 display_twips() (sysinfo 模块),11
 drawNow() (GLCanvas 方法),46
 drive_list() (e32 模块),9
 duration() (Sound 方法),50

 e32 (扩展模块),9
 e32db (扩展模块),74
 e32dbm (模块),77
 EAColumn (appuifw 模块数据),19
 EApplicationWindow (appuifw 模块数据),18
 EBatteryPane (appuifw 模块数据),19
 EBColumn (appuifw 模块数据),19
 ECColumn (appuifw 模块数据),19
 EContextPane (appuifw 模块数据),18
 EControlPane (appuifw 模块数据),18
 EControlPaneBottom (appuifw 模块数据),19
 EControlPaneTop (appuifw 模块数据),19
 ECreated (messaging 模块数据),52
 EDColumn (appuifw 模块数据),19
 EDeleted (messaging 模块数据),52
 EDraft (inbox 模块数据),53
 EFatalServerError (messaging 模块数据),52
 EFindPane (appuifw 模块数据),19
 EHCenterVBottom (appuifw 模块数据),27
 EHCenterVCenter (appuifw 模块数据),27
 EHCenterVTop (appuifw 模块数据),27
 EHLeftVBottom (appuifw 模块数据),27
 EHLeftVCenter (appuifw 模块数据),27
 EHLeftVTop (appuifw 模块数据),27
 EHRightVBottom (appuifw 模块数据),27
 EHRightVCenter (appuifw 模块数据),27
 EHRightVTop (appuifw 模块数据),27
 EInbox (inbox 模块数据),53
 EIndicatorPane (appuifw 模块数据),19
 ellipse() (模块),32
 EMainPane (appuifw 模块数据),18
 EMovedToOutBox (messaging 模块数据),52
 ENaviPane (appuifw 模块数据),19
 ENCRYPT (socket 模块数据),81
 end_time (Entry 属性),67,72
 ENoServiceCentre (messaging 模块数据),52
 ENotReady (audio 模块数据),49
 Entry (calendar 模块类),66,71
 EOpen (audio 模块数据),49
 EOpenComplete (camera 模块数据),34
 EOutbox (inbox 模块数据),53
 EPlaying (audio 模块数据),49
 EPrepareComplete (camera 模块数据),34
 ERecordComplete (camera 模块数据),34
 ERecording (audio 模块数据),49
 EScheduledForSend (messaging 模块数据),52
 EScheduleFailed (messaging 模块数据),52
 EScreen (appuifw 模块数据),18
 ESendFailed (messaging 模块数据),52
 ESent
 inbox 模块数据,53
 messaging 模块数据,52
 ESignalPane (appuifw 模块数据),18
 EStaconBottom (appuifw 模块数据),19
 EStaconTop (appuifw 模块数据),19
 EStatusAnswering (telephone 模块数据),51
 EStatusConnected (telephone 模块数据),51
 EStatusConnecting (telephone 模块数据),51
 EStatusDialling (telephone 模块数据),51
 EStatusDisconnecting (telephone 模块数据),52
 EStatusHold (telephone 模块数据),52
 EStatusIdle (telephone 模块数据),51
 EStatusPane (appuifw 模块数据),18
 EStatusPaneBottom (appuifw 模块数据),19
 EStatusPaneTop (appuifw 模块数据),19
 EStatusReconnectPending (telephone 模块数据),51
 EStatusRinging (telephone 模块数据),51
 EStatusTransferAlerting (telephone 模块数据),52

EStatusTransferring (telephone 模块数据),52
 EStatusUnknown (telephone 模块数据),51
 ETitlePane (appuifw 模块数据),18
 EUniversalIndicatorPane (appuifw 模块数据),19
 event()
 EventFilter 方法,48
 OrientationEventFilter 方法,48
 EventEntry (calendar 模块类),67,72
 EventFilter (sensor 模块类),48
 EWallpaperPane (appuifw 模块数据),19
 execute()
 Dbms 方法,75
 Form 方法,20
 exit_key_handler (Application 属性),17
 export_vcalendars() (CalendarDb 方法),65,71

 export_vcards() (ContactDb 方法),60
 exposure_modes() (camera 模块),34

 FFormAutoFormEdit (appuifw 模块数据),20
 FFormAutoLabelEdit (appuifw 模块数据),20
 FFormDoubleSpaced (appuifw 模块数据),20
 FFormEditModeOnly (appuifw 模块数据),20
 FFormViewModeOnly (appuifw 模块数据),20
 field_types() (ContactDb 方法),60
 file_copy() (e32 模块),9
 find()
 Contact 方法,62
 ContactDb 方法,60
 find_instances() (CalendarDb 方法),65,70
 first_line() (Db_view 方法),76
 flags (Form 属性),20
 flash_modes() (camera 模块),34
 focus
 Application 属性,17
 Text 属性,21
 font (Text 属性),21
 Form (appuifw 模块类),19
 format_rawtime() (e32db 模块),74
 format_time() (e32db 模块),74
 forwarding (KeyCapturer 属性),37
 free_drivespace() (sysinfo 模块),11
 free_ram() (sysinfo 模块),12
 full_name() (Application 方法),18

 get() (Text 方法),23
 get_line() (Db_view 方法),76
 get_pos() (Text 方法),22
 get_repeat() (Entry 方法),66,71
 glBufferData() (gles 模块),43
 glBufferDatab() (gles 模块),43
 glBufferDataaf() (gles 模块),43
 glBufferDatas() (gles 模块),43
 glBufferDataub() (gles 模块),43
 glBufferDataus() (gles 模块),43
 glBufferDataax() (gles 模块),43
 glBufferSubData() (gles 模块),43
 glBufferSubDatab() (gles 模块),43
 glBufferSubDataaf() (gles 模块),43
 glBufferSubDatas() (gles 模块),43
 glBufferSubDataub() (gles 模块),43
 glBufferSubDataus() (gles 模块),43
 glBufferSubDataax() (gles 模块),43
 GLCanvas (glcanvas 模块类),43
 glcanvas (扩展模块),45
 glClipPlanef() (gles 模块),43
 glClipPlanex() (gles 模块),43
 glColorPointer() (gles 模块),40
 glColorPointerf() (gles 模块),40
 glColorPointerub() (gles 模块),40
 glColorPointerx() (gles 模块),40
 glCompressedTexImage2D() (gles 模块),40
 glCompressedTexSubImage2D() (gles 模块),40

 glDeleteBuffers() (gles 模块),44
 glDeleteTextures() (gles 模块),40
 glDrawElements() (gles 模块),40
 glDrawElementsub() (gles 模块),40
 glDrawElementsus() (gles 模块),40
 glDrawTexfvOES() (gles 模块),44
 glDrawTexivOES() (gles 模块),44
 glDrawTexsvOES() (gles 模块),44
 gles (扩展模块),38
 glFogv() (gles 模块),40
 glFogxv() (gles 模块),40
 glGenBuffers() (gles 模块),44
 glGenTextures() (gles 模块),40
 glGetBooleanv() (gles 模块),44
 glGetBufferParameteriv() (gles 模块),44
 glGetClipPlanef() (gles 模块),44
 glGetFixedv() (gles 模块),44
 glGetFloatv() (gles 模块),44
 glGetIntegerv() (gles 模块),40
 glGetLightfv() (gles 模块),44
 glGetLightxv() (gles 模块),44
 glGetMaterialfv() (gles 模块),44
 glGetMaterialxv() (gles 模块),44
 glGetUniformLocation() (gles 模块),40
 glGetTexEnvf() (gles 模块),44
 glGetTexEnvx() (gles 模块),44
 glGetTexParameterf() (gles 模块),44
 glGetTexParameterx() (gles 模块),44
 glLightfv() (gles 模块),41
 glLightModelfv() (gles 模块),40
 glLightModelxv() (gles 模块),40
 glLightxv() (gles 模块),41
 glLoadMatrixf() (gles 模块),41
 glLoadMatrixx() (gles 模块),41
 glMaterialfv() (gles 模块),41
 glMaterialxv() (gles 模块),41
 glMatrixIndexPointerOES() (gles 模块),44
 glMatrixIndexPointerOESub() (gles 模块),44

 glMultMatrixf() (gles 模块),41
 glMultMatrixx() (gles 模块),41
 glNormalPointer() (gles 模块),41
 glNormalPointerb() (gles 模块),41

glNormalPointerf() (gles模块),41
 glNormalPointers() (gles模块),41
 glNormalPointerx() (gles模块),41
 glPointParameterfv() (gles模块),45
 glPointParameterxv() (gles模块),45
 glPointSizePointerOES() (gles模块),45
 glPointSizePointerOESf() (gles模块),45
 glPointSizePointerOESx() (gles模块),45
 glReadPixels() (gles模块),41
 glTexCoordPointer() (gles模块),41
 glTexCoordPointerb() (gles模块),41
 glTexCoordPointerf() (gles模块),42
 glTexCoordPointers() (gles模块),41
 glTexCoordPointerx() (gles模块),42
 glTexEnvfv() (gles模块),42
 glTexEnvxv() (gles模块),42
 glTexImage2D() (gles模块),42
 glTexSubImage2D() (gles模块),42
 glVertexPointer() (gles模块),42
 glVertexPointerb() (gles模块),42
 glVertexPointerf() (gles模块),42
 glVertexPointers() (gles模块),42
 glVertexPointerx() (gles模块),42
 glWeightPointerOES() (gles模块),45
 glWeightPointerOESf() (gles模块),45
 glWeightPointerOESx() (gles模块),45
 graphics (扩展模块),27
 Group (contacts模块类),63
 Groups (contacts模块类),63
 groups (ContactDb 属性),60
 gsm_location() (location 模块),54

 hang_up() (telephone 模块),51
 hide()
 InfoPopup 方法,27
 TopWindow 方法,37
 highlight_color (Text 属性),21
 HIGHLIGHT_ROUNDED (appuifw 模块数据),22
 HIGHLIGHT_SHADOW (appuifw 模块数据),22
 HIGHLIGHT_STANDARD (appuifw 模块数据),22

 Icon (appuifw 模块类),24
 id
 Contact 属性,60
 Entry 属性,66,72
 Group 属性,63
 TodoList 属性,68
 Image.inspect() (graphics 模块),28
 Image.new() (graphics 模块),28
 Image.open() (graphics 模块),28
 image_modes() (camera 模块),34
 image_sizes() (camera 模块),34
 images (TopWindow 属性),38
 imei() (sysinfo 模块),11
 import_vcalendars() (CalendarDb 方法),65,71

 import_vcards() (ContactDb 方法),60
 in_emulator() (e32 模块),9
 inactivity() (e32 模块),10

 Inbox (inbox 模块类),53
 inbox (扩展模块),53
 incoming_call() (telephone 模块),51
 InfoPopup (appuifw 模块类),27
 insert() (Form 方法),20
 is_col_null() (Db_view 方法),76
 is_group (Contact 属性),60
 is_ui_thread() (e32 模块),10

 keycapture (扩展模块),36
 keys() (ContactDb 方法),60
 keys (KeyCatcher 属性),36
 KMdaRepeatForever (audio 模块数据),49

 label (ContactField 属性),63
 last_key() (KeyCatcher 方法),37
 last_modified (Entry 属性),66,72
 layout() (Application 方法),18
 len() (Text 方法),22
 length() (Form 方法),20
 line() (方法),32
 Listbox (appuifw 模块类),23
 load() (Image 方法),29
 location
 ContactField 属性,63
 Entry 属性,66,71
 扩展模块,54

 makeCurrent() (GLCanvas 方法),46
 max_ramdrive_size() (sysinfo 模块),12
 max_volume() (Sound 方法),50
 max_zoom() (camera 模块),34
 maximum_size (TopWindow 属性),38
 measure_text() (方法),33
 menu
 Application 属性,17
 Form 属性,20
 messaging (扩展模块),52
 mms_send() (messaging 模块),52
 module_info() (positioning 模块),55
 modules() (positioning 模块),55
 monthly_instances() (CalendarDb 方法),65,70

 multi_query() (appuifw 模块),16
 multi_selection_list() (appuifw 模块),16

 name
 Group 属性,63
 TodoList 属性,68
 next_line() (Db_view 方法),76
 note() (appuifw 模块),16

 OBEX (socket 模块),81
 open()
 Content_handler 方法,24
 Dbms 方法,75
 calendar 模块,64,70
 contacts 模块,59

e32dbm 模块,77
 open_standalone() (Content_handler 方法),25

 orientation (Application 属性),18
 orientation.BACK (属性),47
 orientation.BOTTOM (属性),47
 orientation.FRONT (属性),47
 orientation.LEFT (属性),47
 orientation.RIGHT (属性),47
 orientation.TOP (属性),47
 OrientationEventFilter (sensor 模块类),48
 originating (Entry 属性),72
 os_version() (sysinfo 模块),12

 pieslice() (方法),32
 play() (Sound 方法),49
 point() (方法),33
 polygon() (方法),32
 pop() (Form 方法),20
 popup_menu() (appuiw 模块),16
 position() (positioning 模块),55
 position
 Listbox 属性,24
 TopWindow 属性,38
 POSITION_INTERVAL (positioning 模块数据),55
 positioning (扩展模块),55
 prepare() (Db_view 方法),76
 priority (Entry 属性),67,72
 pys60_version (e32 模块数据),9
 pys60_version_info (e32 模块数据),9
 PYTHON_TLS->thread_state, 84
 PyThread_AtExit(), 84

 query() (appuiw 模块),15

 record() (Sound 方法),50
 rectangle() (方法),32
 release() (camera 模块),36
 ReminderEntry (calendar 模块类),73
 remove_image() (TopWindow 方法),37
 reorganize() (e32dbm 方法),78
 replication (Entry 属性),67,72
 reset_inactivity() (e32 模块),10
 resize() (Image 方法),28
 RFCOMM (socket 模块数据),81
 ring_type() (sysinfo 模块),12
 rollback()
 Contact 方法,61
 Dbms 方法,75
 Entry 方法,66,71
 RotEventFilter (sensor 模块类),48

 s60_version_info (e32 模块数据),10
 save() (Image 方法),29
 save_hook (Form 属性),20
 say() (audio 模块),49
 schema (ContactField 属性),63
 screen (Application 属性),17
 screenshot() (graphics 模块),28

 select_access_point() (socket 模块),82
 select_module() (positioning 模块),55
 selection_list() (appuiw 模块),16
 Sensor (sensor 模块类),47
 sensor (扩展模块),46
 sensors() (sensor 模块),46
 set() (Text 方法),23
 set_default_access_point() (socket 模块),82

 set_event_filter() (Sensor 方法),48
 set_exit() (Application 方法),18
 set_home_time() (e32 模块),9
 set_list() (Listbox 方法),24
 set_pos() (Text 方法),23
 set_position() (Sound 方法),50
 set_repeat() (Entry 方法),66,71
 set_requestors() (positioning 模块),55
 set_security() (socket 模块),81
 set_tabs() (Application 方法),18
 set_time() (Entry 方法),66,71
 set_volume() (Sound 方法),50
 shadow (TopWindow 属性),38
 show()
 InfoPopup 方法,27
 TopWindow 方法,37
 signal() (Ao_lock 方法),11
 signalBars() (sysinfo 模块),12
 signal_dbm() (sysinfo 模块),12
 size
 Canvas 属性,26
 Image 属性,29
 Listbox 属性,24
 TopWindow 属性,38
 sms_messages() (Inbox 方法),53
 sms_send() (messaging 模块),52
 socket (扩展模块),81
 Sound (audio 模块类),49
 Sound.open() (audio 模块),49
 SPyAddGlobal(), 84
 SPyAddGlobalString(), 84
 SPyErr_SetFromSymbianOSErr(), 83
 SPyGetGlobal(), 84
 SPyGetGlobalString(), 84
 SPyRemoveGlobal(), 84
 SPyRemoveGlobalString(), 84
 start() (KeyCapturer 方法),37
 start_exe() (e32 模块),10
 start_finder() (camera 模块),35
 start_record() (camera 模块),36
 start_server() (e32 模块),10
 start_time (Entry 属性),66,72
 state() (Sound 方法),50
 stop()
 Image 方法,29
 KeyCapturer 方法,37
 Sound 方法,50
 stop_finder() (camera 模块),36
 stop_position() (positioning 模块),55

stop_record() (camera 模块),36
style (Text属性),21
STYLE_BOLD (appuifw 模块数据),22
STYLE_ITALIC (appuifw 模块数据),22
STYLE_STRIKETHROUGH (appuifw 模块数据),22
STYLE_UNDERLINE (appuifw 模块数据),22
sw_version() (sysinfo 模块),12
sync() (e32dbm 方法),78
sysinfo (扩展模块),11

take_photo() (camera 模块),34
telephone (扩展模块),51
text() (扩展模块),81
thread (扩展模块),33
time() (Inbox方法),53
title
 Application属性,17
 Contact属性,60
todo_list (TodoEntry属性),68
todo_lists (CalendarDb属性),65
TodoEntry (calendar 模块类),67,73
TodoList (calendar 模块类),68
TodoListDict (calendar 模块类),68
TopWindow (topwindow 模块类),37
topwindow (扩展模块),37
total_ram() (sysinfo 模块),12
total_rom() (sysinfo 模块),12
transpose() (Image方法),29
type (ContactField属性),63

uid() (Application属性),18
unread() (Inbox方法),54

value (ContactField属性),63
visible (TopWindow属性),38

wait() (Ao_lock method), 10
white_balance_modes() (camera 模块),34