TIZEN

Tizen 原生应用程序开发指南

运用 70 个通俗易懂的举例对 Tizen 平台进行全面分析。

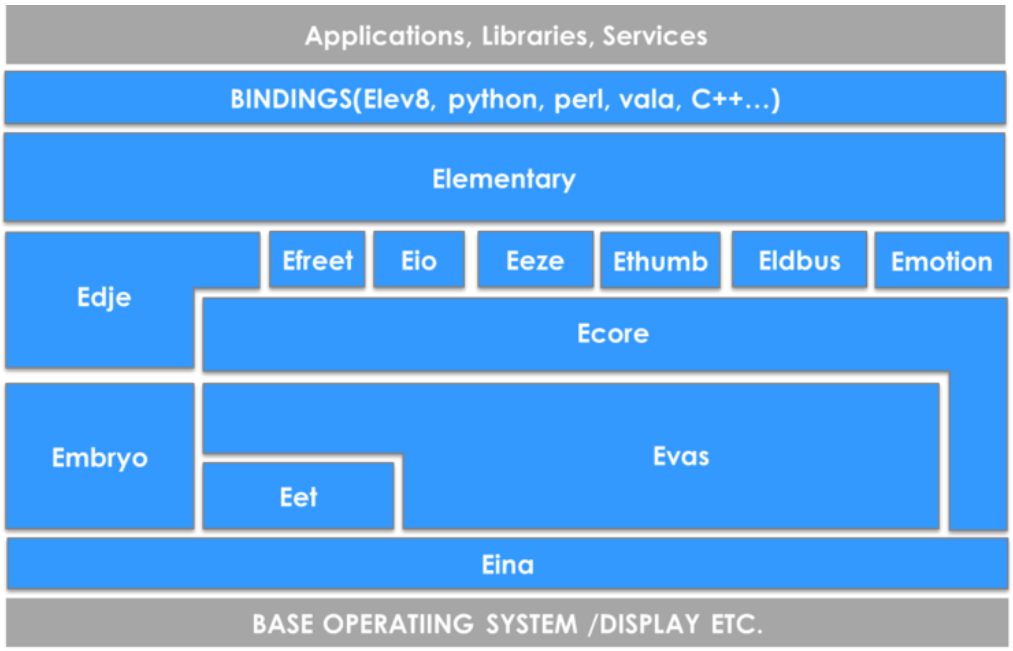
作者：Jung, Dong-Geun (Denis.Jung)

目录

1. 介绍 EFL

Tizen 原生 API 的核心部分包含 EFL，后者是一种集成了多个库的复杂库。

下图为 EFL 模块的示意图。



上级库关联着其下方的子库。例如，Elementary 依赖所有子库，而 Encore 仅关联 Evas、Eet 和 Eina。

1) Eina

Eina 是最基本的 EFL 元素之一，属于数据结构库。此库类似于 C++ 的 STL，可为用户提供多种有用功能，以供其通过安全、快速、轻松的方式实施复杂逻辑，其中包括排列、列表、哈希、树以及共享字符串。这些数据结构功能可令应用程序和 EFL 库上层级别中的所有 EFL 库有效实施必要的逻辑。

2) Eet

Eet 负责数据编码和解码工作。您可将任意数据结构或图像数据另存为压缩文件，或通过网络将其转移至其它计算机。您还可读取压缩文件并在以后解码这些文件。Eet 的这项压缩功能与 .Zip 文件的原则非常相似，通过此功能，可让您即时访问文件中任意位置的数据。若您使用加密功能，则能可靠地存储数据。

3) Evas

Evas 的作用就像画布。您可通过 Evas 来表达图像、文本和不同的形状，其中包括 Windows 中的方形、线条和多边形。所有输出均可得到具体表现。由于 Evas 提供这些对象作为 Evas_Object 类型，因此，您在编程期间可利用 Evas_Object 接口访问所有图形对象，并在屏幕上表达所需的图形对象。此外，该接口是提供给 Evas 使用的，所以，每个对象都能正确响应您输入的事件。

Evas 采用保留模式作为渲染方法，可在整个场景图内部管理各个对象，并自动优化和渲染要在屏幕上查看的对象。这样一来，应用程序开发人员就不会因为复杂的渲染机制而受阻。

基本上，Evas 可支持单个软件的所有渲染方法，同时还有助于通过后端支持来实现硬件加速。

4) Ecore

作为一种旨在为用户提供便利的基于系统的库，Ecore 提供与主循环、计时、事件、连接、IPC、线程、Windows 系统等内容有关函数。Ecore 在内部处理与各系统 API 相关的复杂使用设置和阶段。这一简化措施有助于用户在亲自执行各项函数时节省时间和精力。

EFL 应用程序基于 Ecore 提供的主循环工作。由于 Ecore 提供的函数是结合 Ecore 主循环的主逻辑进行处理的，因此如有必要，您应使用 Ecore 提供的函数而不是使用系统 API。

5) Edje

Edje 提供了用于构成复杂 GUI 的功能。Edje 将 EDC 用作脚本语言。使用EDC 脚本，您可在进行程序设计时将 GUI 与程序代码分离。EDC 通过 edje_cc 编译器转换成 EDJ 类型的二进制。程序可在运行时阶段读取此 EDJ 文件，以便将其绑定为 Evas_Object 来构建 GUI。由于 Edje 模块的存在，您可在应用程序中更改 GUI设计，而无需对其执行二次编译。

6) Embryo

Embryo 是一种字节码虚拟机，用于可在 EDC 文件内实施的小程序。在多数情况下，您可以另外使用 C 语言样式的 Embryo 脚本语言来执行简单的计算或函数，比如更改各个对象的状态。这种 Embryo 脚本通过 PAWN 编译器转换为 PAWN 二进制。PAWN 程序不需要依赖计算机环境；而是可以通过抽象机可执行程序 (AMX) 虚拟机解密后进行操作。此外，只需一次输出即可保证在不同系统环境中执行相同操作。

7) Emotion

Emotion 是视频和音频播放库。Emotion 使用其它视频播放插件（包括：Gstreamer、Xine 或 VLC）来播放视频。视频输出会与将要提供给用户的 Evas 对象同步。与此对应的是，用户可以播放视频，也可结合 GUI 编辑屏幕。

8) Elementary

Elementary 是一种小部件工具套件库，用于提供通用小部件，比如按钮、列表、标签和滑块。同时，此库还支持更改动态主题和扩展 GUI，以便针对“外观”和不同屏幕分辨率提供相应支持。

9）Efreet、Eio、Eeze、Ethumb 及 Eldbus

其它库包括：Efreet、Eio、Eeze、Ethumb 及 Eldbus。

Efreet 是为了让应用程序正常工作而设计的库，遵循与图标、桌面文件、菜单等内容相关的 Freedesktop.org 标准。

Eio 是面向异步 I/O 的库。Ethumb 提供的函数通过添加帧图像来生成缩略图图像。

Eeze 用于通过 udev 操控硬件设备。例如，在需要了解设备状态时，可利用 Eeze 确定设备状态，比如：CD-Rom 中是否插入了磁盘、CPU 温度，以及电池的剩余电量。最后，Eldbus 是环绕 dbus 库的包装程序，属于消息总线系统。它可实施一套 dbus 规范来执行 IPC。

10）语言绑定

总体而言，EFL 支持 C 语言，同时也涉及到使用 Elev8 (JavaScript)、Python、Pearl、C++ 和 Vala 的语言绑定项目。

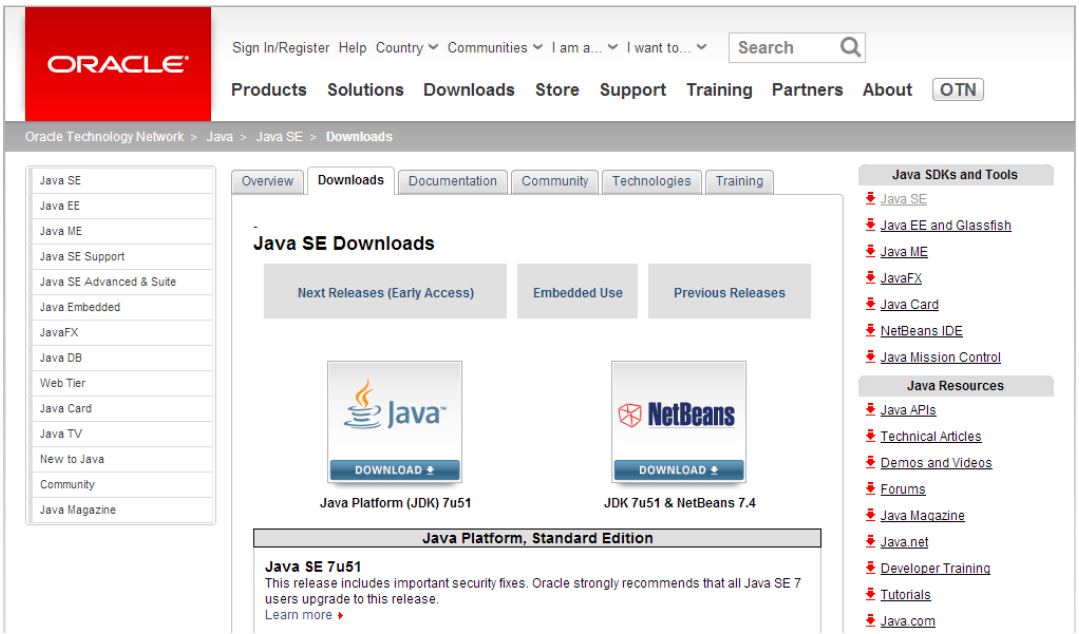
（参考：EFL Korea Community）

2. 建立 Tizen 开发环境

A. 下载 Java 开发工具包 (JDK)

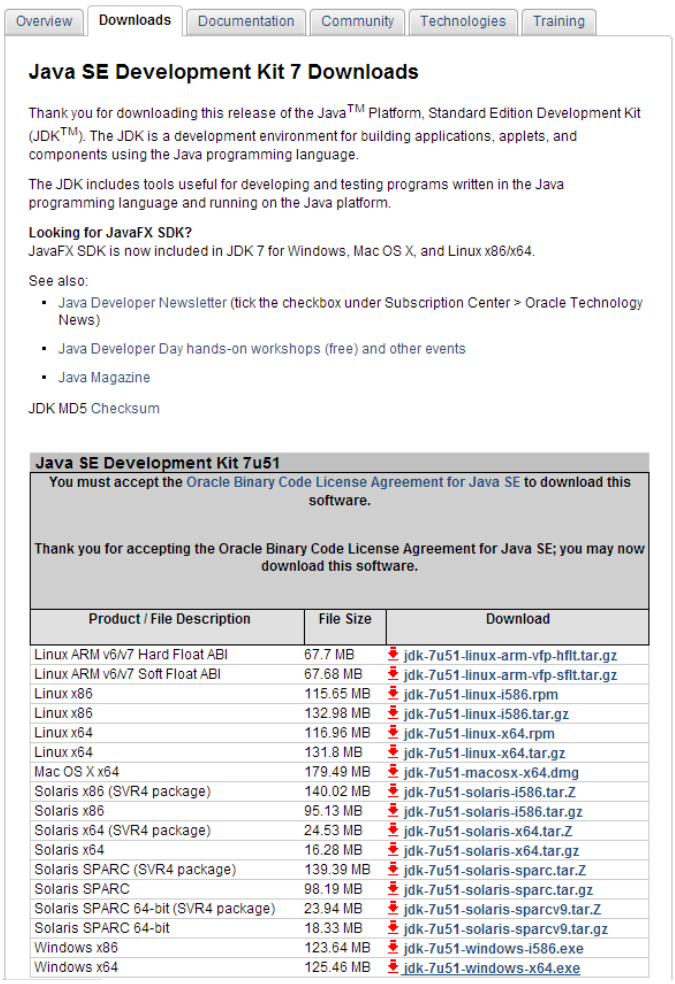
Tizen 使用 Java 在开发环境中支持所有类型的操作系统。因此，您的设备需要装有 JDK。如未安装 JDK，安装 Tizen SDK 时，您将看到错误消息，且无法完成安装。请按照以下步骤操作：

1) 打开 Web 浏览器，转至 http://www.oracle.com/technetwork/java/javase/downloads。



2) 按下 Java DOWNLOAD（或 JDK DOWNLOAD）按钮。当屏幕内容变化后，请选中“Accept License Agreement”复选框。

3）下载 Java SE Development Kit xux。从列表中选择您要使用的操作系统版本。如果您使用 32 位操作系统，请单击 Windows x86。如果您使用 64 位操作系统，请单击 Windows x64。



Overview | Downloads | Documentation | Community | Technologies | Training

Java SE Development Kit 7 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

Looking for JavaFX SDK?
JavaFX SDK is now included in JDK 7 for Windows, Mac OS X, and Linux x86/x64.

See also:

- Java Developer Newsletter (tick the checkbox under Subscription Center > Oracle Technology News)
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK MD5 Checksum

Java SE Development Kit 7u51

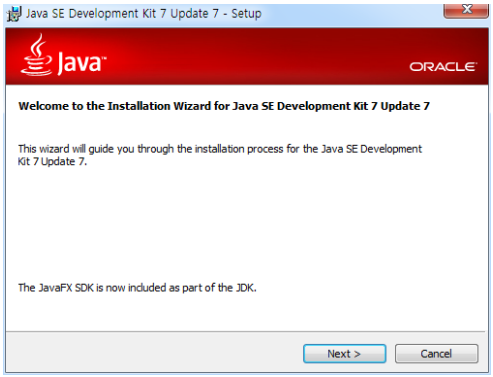
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

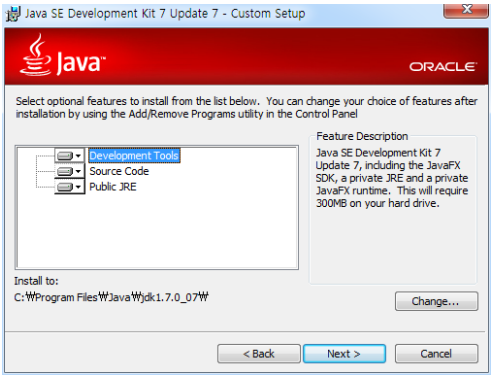
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	67.7 MB	jdk-7u51-linux-arm-vfp-hflt.tar.gz
Linux ARM v6/v7 Soft Float ABI	67.68 MB	jdk-7u51-linux-arm-vfp-sflt.tar.gz
Linux x86	115.65 MB	jdk-7u51-linux-i586.rpm
Linux x86	132.98 MB	jdk-7u51-linux-i586.tar.gz
Linux x64	116.96 MB	jdk-7u51-linux-x64.rpm
Linux x64	131.8 MB	jdk-7u51-linux-x64.tar.gz
Mac OS X x64	179.49 MB	jdk-7u51-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.02 MB	jdk-7u51-solaris-i586.tar.Z
Solaris x86	95.13 MB	jdk-7u51-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.53 MB	jdk-7u51-solaris-x64.tar.Z
Solaris x64	16.28 MB	jdk-7u51-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	139.39 MB	jdk-7u51-solaris-sparc.tar.Z
Solaris SPARC	98.19 MB	jdk-7u51-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.94 MB	jdk-7u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.33 MB	jdk-7u51-solaris-sparcv9.tar.gz
Windows x86	123.64 MB	jdk-7u51-windows-i586.exe
Windows x64	125.46 MB	jdk-7u51-windows-x64.exe

B. 安装 Java 开发工具包 (JDK)

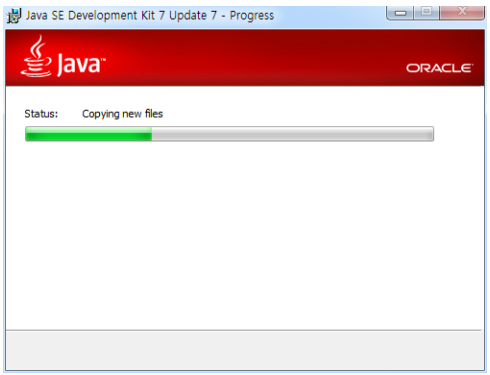
1) 双击您所下载的 EXE 文件。随后您将看到安装屏幕。单击 Next 按钮。

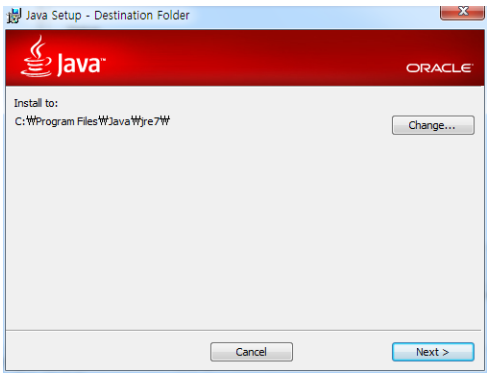




2) 此时出现的屏幕中会显示将供您选择用于安装的项目，请保持默认设置不变，单击 Next 按钮。

3) 一旦 JDK 开始安装，请等待安装流程完成。当您看到询问 JRE 安装路径的屏幕时，请单击 Next 按钮。若要指定非默认文件夹的其它安装路径（如，C:\Program Files\Java\jre7\），请单击 Change 更改路径。



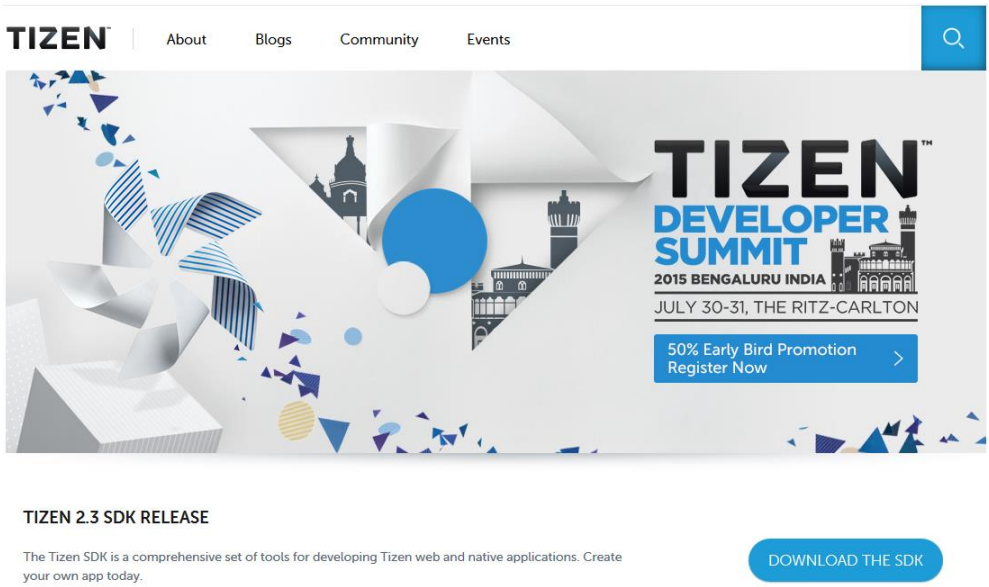


4) 一旦 JRE 开始安装，请等待安装流程完成。当您看到指示已完成安装的屏幕时，请单击 Close 按钮。

C. 下载 Tizen SDK

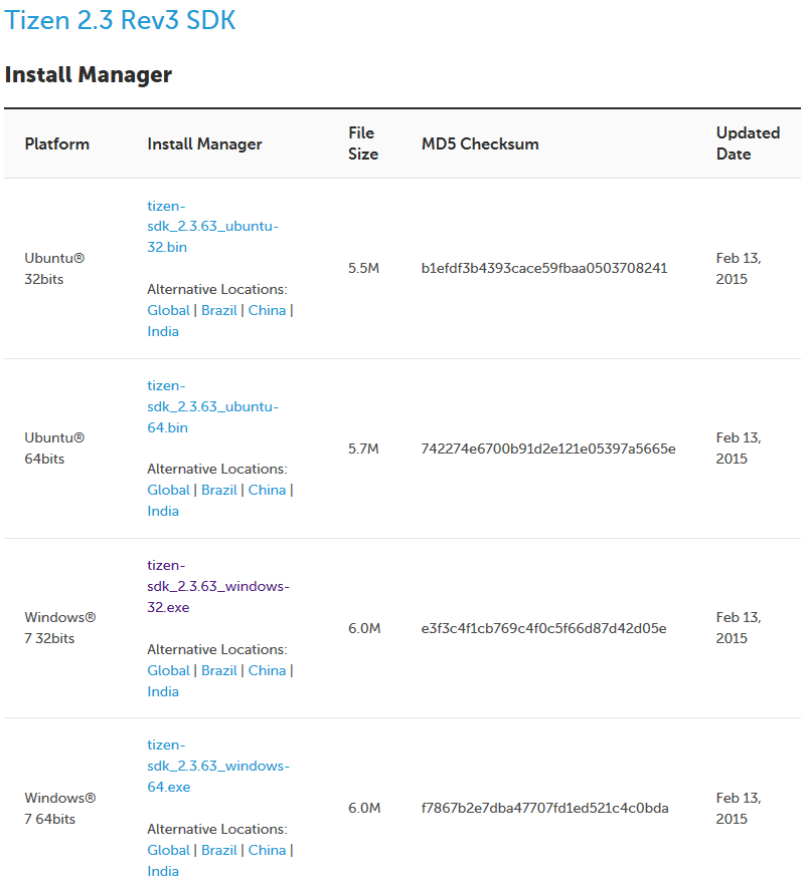
本节将为您介绍如何构建 Tizen 开发环境。首先，您需要下载 SDK 安装文件。

转至 https://www.tizen.org/。



在首先出现的屏幕上单击“DOWNLOAD THE SDK”按钮。随即您将进入 SDK 下载页面。

若您使用 32 位 Windows 版本，请在 Install Manager 中选择并下载 Windows XP/7 32bits。若您使用 64 位 Windows 版本，请选择并下载 Windows XP/7 64bits。Ubuntu 32 位/64 位操作系统和 Mac 操作系统也受到支持。



Tizen 2.3 Rev3 SDK

Install Manager

Platform	Install Manager	File Size	MD5 Checksum	Updated Date
Ubuntu® 32bits	tizen-sdk_2.3.63_ubuntu-32.bin Alternative Locations: Global | Brazil | China | India	5.5M	b1efdf3b4393cace59fbaa0503708241	Feb 13, 2015
Ubuntu® 64bits	tizen-sdk_2.3.63_ubuntu-64.bin Alternative Locations: Global | Brazil | China | India	5.7M	742274e6700b91d2e121e05397a5665e	Feb 13, 2015
Windows® 7 32bits	tizen-sdk_2.3.63_windows-32.exe Alternative Locations: Global | Brazil | China | India	6.0M	e3f3c4f1cb769c4f0c5f66d87d42d05e	Feb 13, 2015
Windows® 7 64bits	tizen-sdk_2.3.63_windows-64.exe Alternative Locations: Global | Brazil | China | India	6.0M	f7867b2e7dba47707fd1ed521c4c0bda	Feb 13, 2015

下载 Install Manager 文件。向下滚动查看，下载用于 SDK 映像的相同项目。下载 Install Manager 后，安装程序会在安装期间自动从 Internet 下载必要文件。您需先下载映像文件，以防安装期间出错。

SDK Image

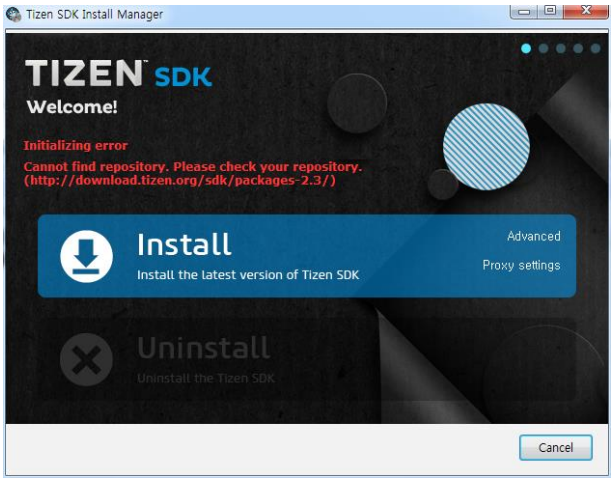
Platform	SDK Image	File Size	MD5 Checksum	Updated Date
Ubuntu® 32bits	tizen-sdk-image-TizenSDK_2.3.0_Rev3-ubuntu32.zip Alternative Locations: Global | Brazil | China | India	2.1G	b448ebb652cfb1fc7737e3d30a71ca39	July 1, 2015
Ubuntu® 64bits	tizen-sdk-image-TizenSDK_2.3.0_Rev3-ubuntu64.zip Alternative Locations: Global | Brazil | China | India	2.1G	ae0f7fa8a25aa7bd563f15e1718e11ac	July 1, 2015
Windows® 7 32bits	tizen-sdk-image-TizenSDK_2.3.0_Rev3-windows32.zip Alternative Locations: Global | Brazil | China | India	2.5G	360767f03103ef05ec1ac17f190d7f3e	July 1, 2015
Windows® 7 64bits	tizen-sdk-image-TizenSDK_2.3.0_Rev3-windows64.zip Alternative Locations: Global | Brazil | China | India	2.5G	0f16e4293215546dfa0072934abe1917	July 1, 2015

D. 安装 Tizen SDK

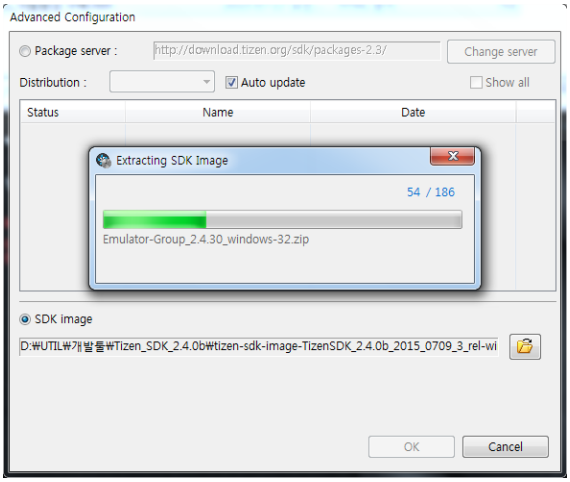
本节将为您介绍如何安装 Tizen SDK。所述说明基于 SDK 2.4.0b 版。但您也可轻松将这些说明应用到其它版本。另外，由于 Tizen SDK 安装简单，即便是新手也可安装，无需参考安装手册。

完成下载后，双击 EXE 文件（tizen_sdk_2.x.xx_window-32.exe）以执行此文件。如果您看到警告窗口，请忽略此窗口并单击 Yes。

出现安装屏幕后，您需要指定映像文件。单击右边的 Advanced 按钮。

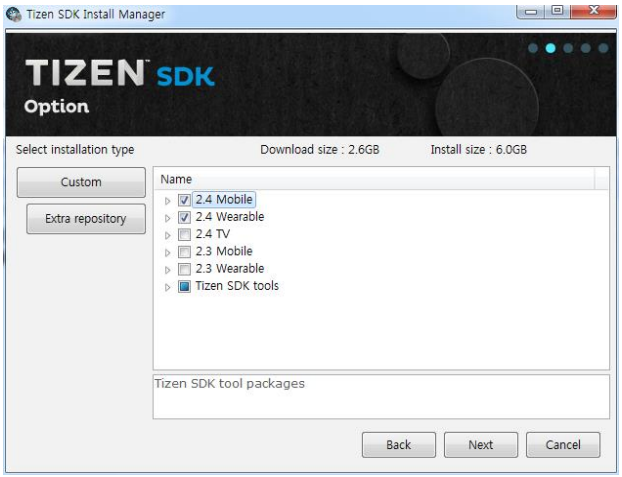


选中 SDK Image 旁的复选框，单击右边的 Open file 按钮。此时将打开一个弹出窗口供您从中打开文件。选中您从 Tizen 开发人员网站下载的映像文件 (tizen-sdk-image-TizenSDK_2.4.x_xxxx_xxxx_x_rel-windows-32.zip)。该 Zip 文件将解压。等待完成解压。

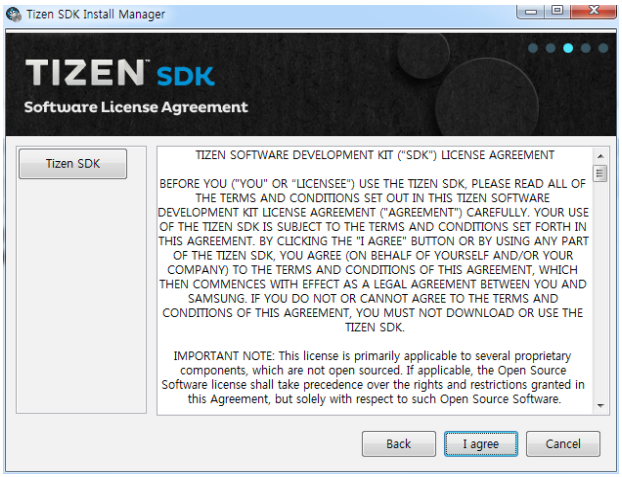


完成解压后后，单击 OK 按钮以关闭此窗口。单击主屏幕中的 Install 按钮，然后移至下一页。

出现 Select Install Type 屏幕后，选中 2.4 Mobile 和 2.4 Wearable，然后单击 Next 按钮。



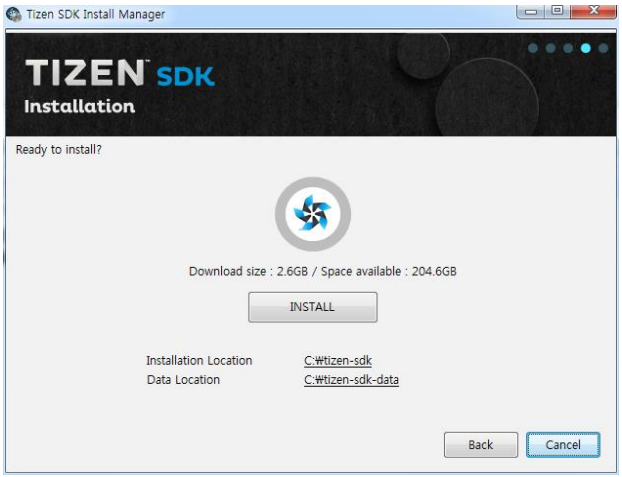
出现 Software License Agreement 屏幕后，单击 I agree 按钮。



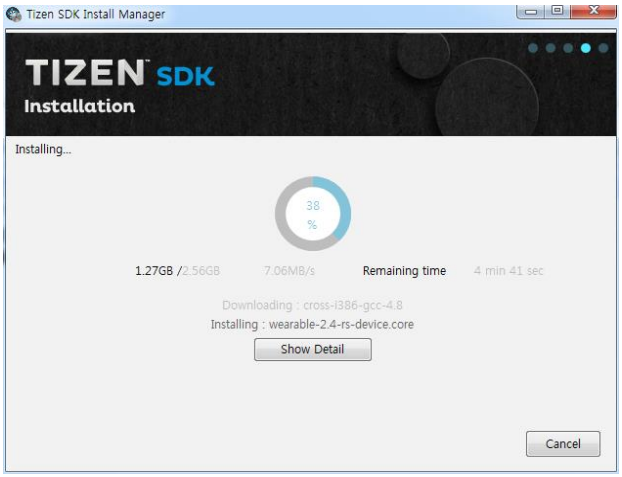
在 Ready to install 屏幕中，分别指定 SDK 的安装路径和 SDK 数据文件夹的位置。SDK 默认路径为 C:\tizen-sdk。如果您想更改路径，请单击地址字段并指定所需的文件夹。

默认数据文件夹路径为 C:\tizen-sdk-data。若您想采用简单的文件夹结构，可将路径编辑至 SDK 文件夹之下，比如 C:\tizen-sdk\data。

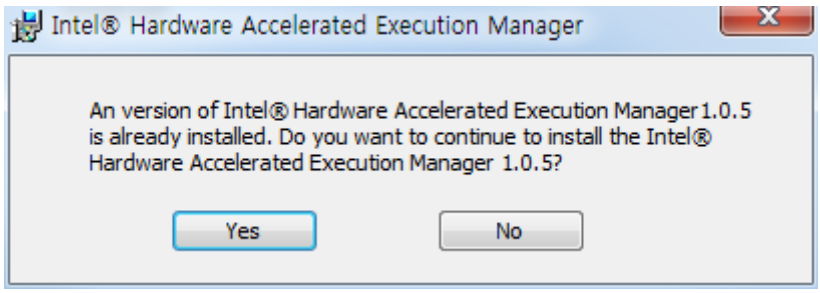
单击 Install 按钮开始安装。



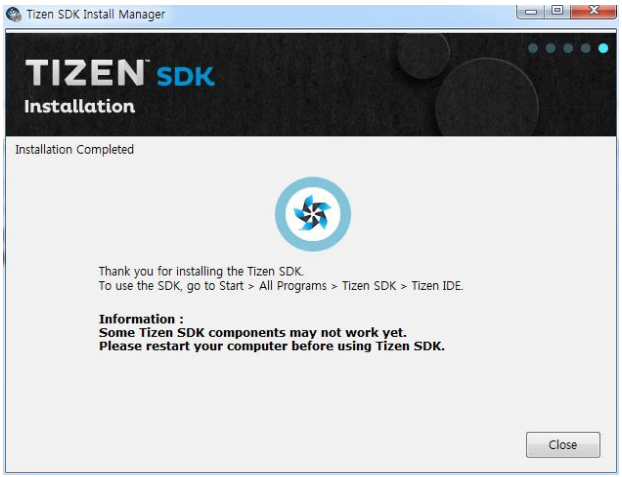
一旦 JRE 开始安装，请等待安装流程完成。



安装期间，您可能会看到弹出窗口，提示您需要更新硬件加速驱动程序。单击 Yes 并继续安装。



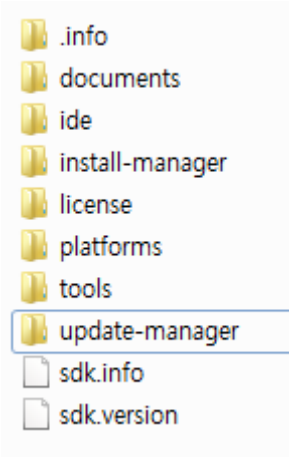
完成安装后，单击 Close 按钮。



3. 运行 Tizen 开发环境

A. 查看已安装的 SDK

本节将为您介绍已装 SDK 中要包含的数据类型。打开 SDK 文件夹后，您将看到文件夹列表，如下所示。如果采用默认设置，您的安装文件夹路径将为 C:\tizen-sdk。



我们一起来看看此列表中的一些重要文件夹。“\documents”文件夹中包含与 Tizen API 和开发有关的 PDF 手册。

“\platforms”文件夹中包含各种使用虚拟设备（模拟器）和 Tizen API 的示例。

原生应用程序示例均保存在“\platforms\tizen-x.x\mobile\samples\Template\Native”文件夹中。

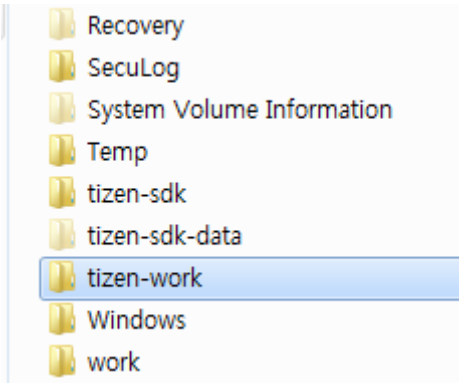
Web 应用程序示例均保存在“\tizen-sdk\platforms\tizen-x.x\mobile\samples\Template\Web”文件夹中。由于整个源项目都包含在内，因此，您可通过直接在 IDE 中加载、构建以及执行相关操作来查看结果。接下来要介绍的文件夹中包含对开发人员最有用的资源。

“\ide”文件夹中包含各种开发环境工具，可供开发人员创建 Tizen 应用程序。此文件夹提供的 IDE 可供您创建新项目、输入源代码、通过 UI 编辑器在屏幕上创建小部件、构建项目并通过模拟器将项目作为目标对象来执行。应用程序开发人员无需使用此文件夹中的内容。

“\tools”文件夹中包含的工具可供您在 Linux 环境中创建证书、构建以及执行源项目。

F. 执行 IDE 开发环境工具

下面，让我们一起来看看如何在实际开发环境中运行这些工具。首先，我们需要创建一个工作文件夹，然后再执行 IDE。在此例中，新文件夹创建在 C: 驱动器的根文件夹下，名为“tizen-work”。您可为您的文件夹选择任何路径和名称。



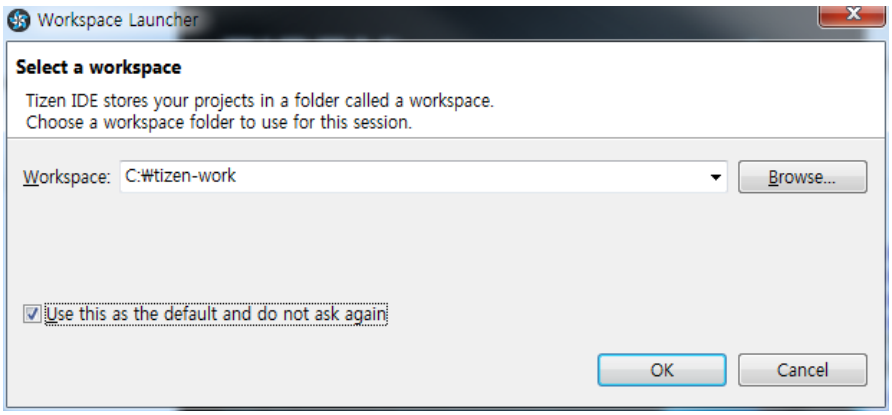
接下来将在所安装的 SDK 中执行 Tizen IDE。您可以采用 Windows 的“开始”菜单中的 [所有程序 > Tizen SDK > Tizen IDE] 路径。

[提示！]
Tizen 使用 Eclipse 作为其 IDE（开发工具）。这意味着 Tizen 与 Java 和 Android 共享相同的开发工具。

由于 Tizen 原生应用程序使用的是 C 语言而不是 Java，因此开发人员可以使用 Visual Studio。但在这种情况下，开发人员可能需要针对具体使用需求购买程序许可证。为避免此类不便，Eclipse 采用的是基本规格。开发人员免费使用此程序创建应用程序。

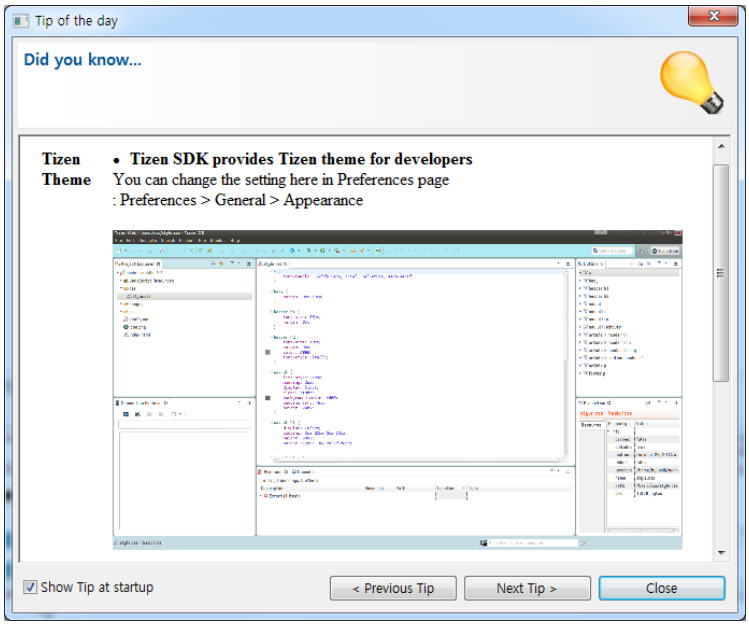
如果您熟悉 Visual Studio 和其它开发环境，您将很快习惯 Eclipse 环境。

初次执行 IDE 时，您将看到如下弹出窗口。此屏幕可供您指定用于保存源项目的工作文件夹。

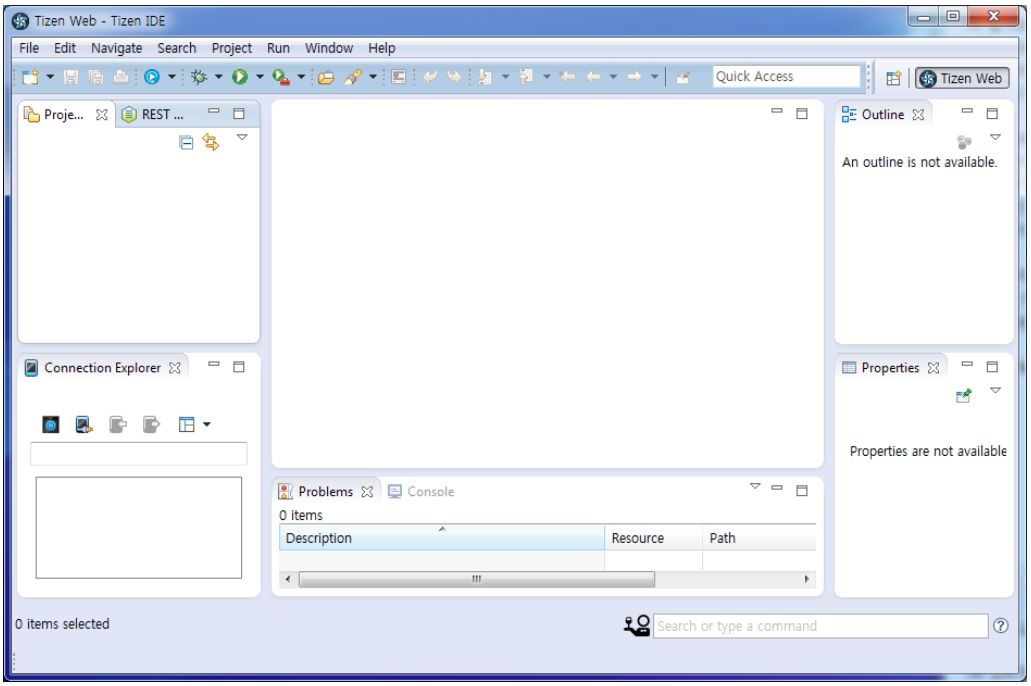


单击 Browse 按钮，指定将要用作工作空间的文件夹。在此情况下，系统会使用 C:\tizen-work 作为工作文件夹。若您已指定文件夹，请选中“Use this as the default and do not ask again”旁的复选框，然后单击 OK 按钮前进至下一步骤。建议您选中此消息对应的复选框，不然以后您每次执行 IDE 时都会看到此窗口。

执行 IDE 后，您将看到如下所示的屏幕。此屏幕将为您介绍使用提示。单击窗口右上角的 × 符号可关闭此窗口。



关闭屏幕后，您将看到如下所示的屏幕。如果您有使用 Java 或 Android 进行开发的经验，则会对此屏幕感到熟悉。



本节将为您简要介绍各个字段。主菜单和工具栏位于顶部。如果您不想使用工具栏，则可使用各种快捷键和快捷菜单。

您将在工具栏右侧看到一个标记为“Tizen Web”的方框。您可使用此框在 Web 应用程序和原生应用程序之间转换，以及在调试模式和执行模式之间转换。如果您是在调试模式下执行应用程序，界面中还会添加另一个按钮。因此，通过使用两个切换式按钮，您可在调试和执行模式之前根据需要进行切换。

您将在工具栏左侧看到一个名为“Project Explorer”的字段。该字段会向您显示保存在工作文件夹中的所有项目和保存在子文件中的所有字段（如，“Source”和“Resource”文件）。如果您在该字段中选择 UI 屏幕信息文件的源文件，屏幕中间将会显示对应详情。您还可修改项目属性，或从快捷菜单执行项目。

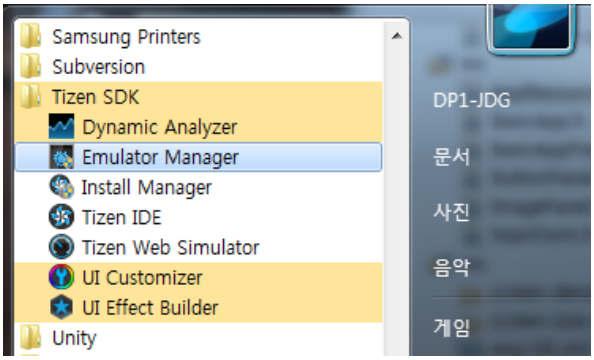
“Connection Explorer”字段中会显示一份可用设备列表，用于安装已开发的应用程序。执行模拟器时将会显示对应的模拟器。当您使用线缆连接电话时，该电话将被添加进来。

此时，您将在屏幕下方中部看到几个选项卡。其中最重要的选项卡如下。“Problems”选项卡用于显示您在构建项目时出现的 Log 消息、警告或错误消息。如果出现任何构建错误，您可参考此处显示的消息对错误进行纠正。

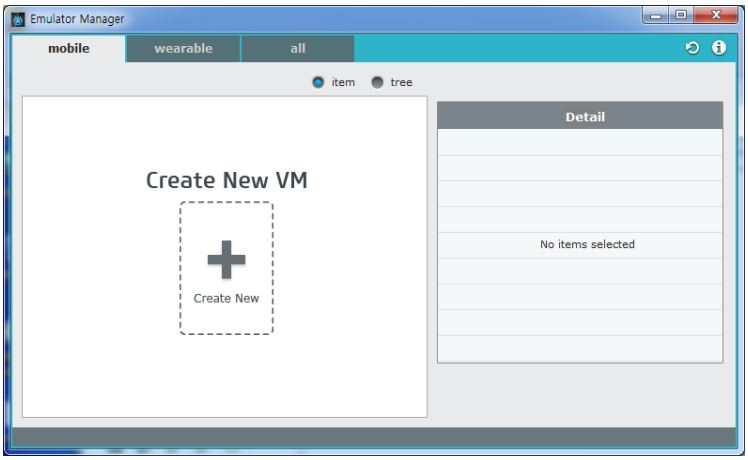
“Log”选项卡用于显示您在执行应用程序时出现的 Log 消息。通过应用 API 函数（比如源代码的 AppLog() 或 AppLogDebugTag()），您可在执行应用程序时查看进度状态。

G. 运行模拟器

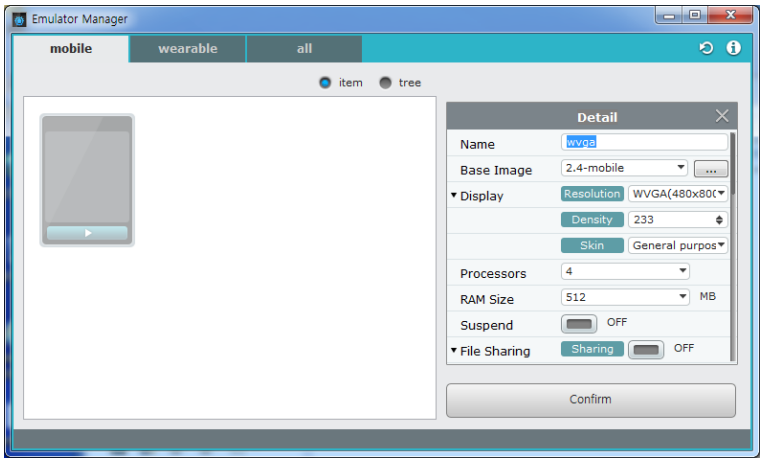
本节将为您介绍如何创建和运行模拟器。单击 [Windows“开始”按钮 > 所有程序 > Tizen SDK > Emulator Manager]。如果您看到警告窗口，请忽略此窗口并单击 Yes 按钮。



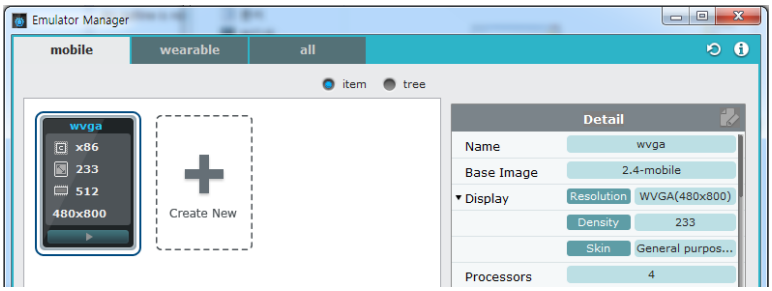
如果您是初次运行 Emulator Manager，则需创建模拟器。单击屏幕左侧“Create New VM”下方的“+”号。



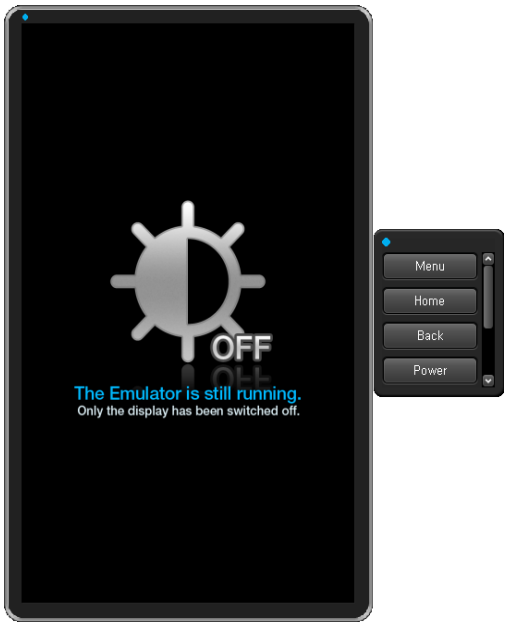
然后，您将看到右侧出现一个屏幕，供您指定模拟器的选项。将 wvga 指定为名称属性。您可以将剩余内容保留为默认属性。单击“Confirm”按钮创建新模拟器。



此时，您将在屏幕左侧看到名为“wvga”的新模拟器。单击新模拟器图标下方的箭头按钮以运行该模拟器。如果您想更改模拟器的选项，请单击 Reset 按钮。



当出现“Windows Security Warning”弹出窗口时，请单击 Unblock 按钮继续此流程。您可能会在模拟器右侧看到一个方形的小弹出窗口。此窗口中包含硬件按键。其中从上至下依次为 Menu、Home、Back、Power、Volume+ 和 Volume-。此弹出窗口配备最类似 Android 支持的硬件按键的功能。



如果设备转换至睡眠模式，您只需单击 Home 按钮即可重新打开屏幕。然后，请拖动屏幕进行解锁。

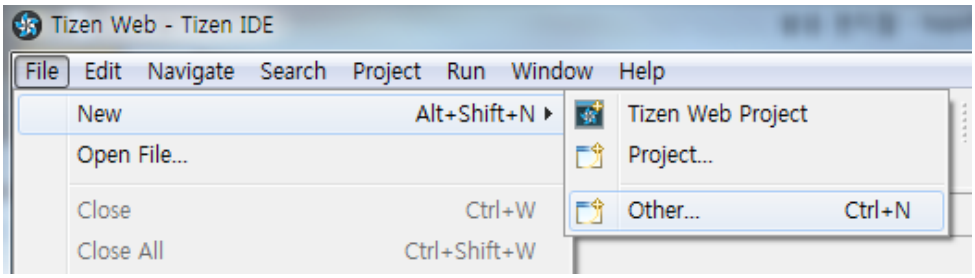
4. 执行示例

SDK 为您提供各种指南手册，但具体的实践示例可能更有用。这类示例将成为您今后处理新项目时最常参考的资源。您将了解如何将示例添加到 IDE以及如何使用模拟器运行这些示例。我们先来一起来看看一些重要示例。

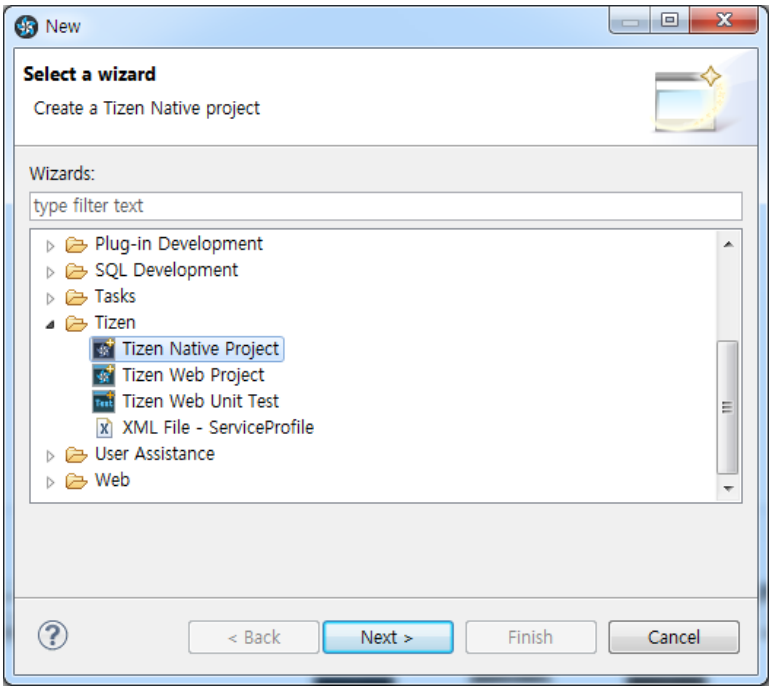
A. 向 IDE 添加样本

您现已准备好开发 Tizen 应用程序。让我们使用一个示例来实践此流程。

从主菜单选择 [File > New > Other ...]。

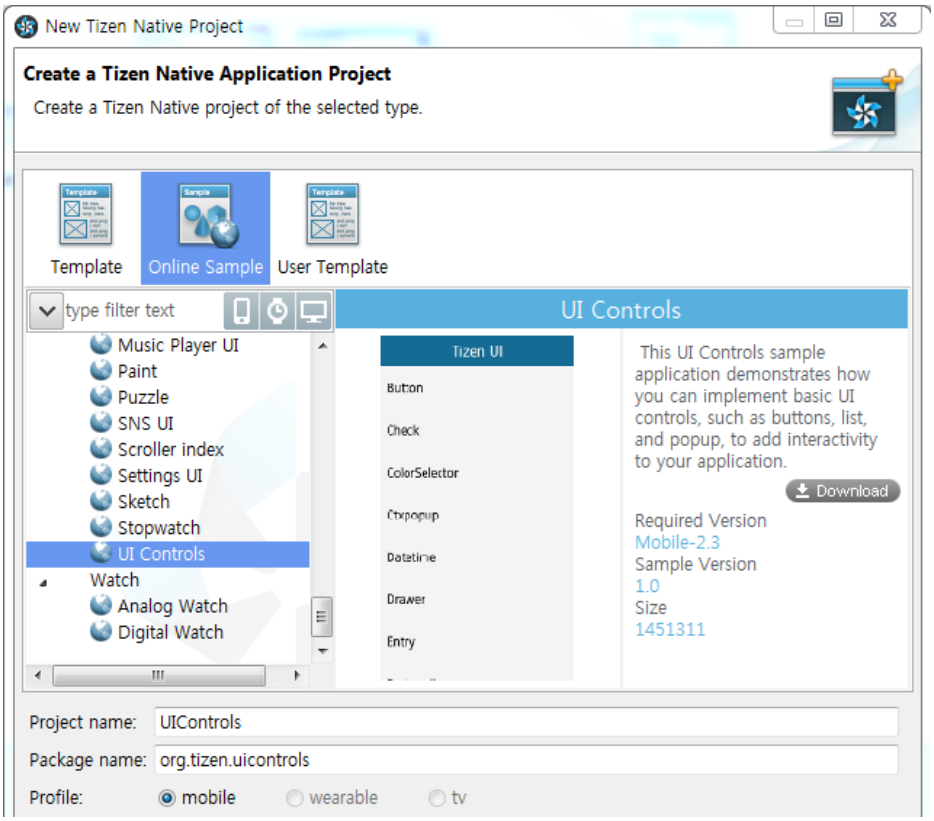


出现新弹出窗口后，您将看到一份树状结构的列表。打开 Tizen 文件夹，从各个项目中选择“Tizen Native Project”。然后单击 Next 按钮。

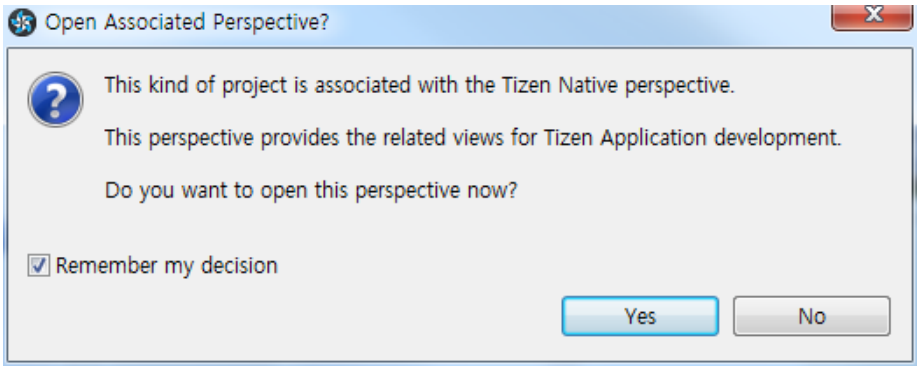


出现“Create a Tizen Native Application Project”弹出窗口后，单击 Template 按钮右侧的“Online Sample”按钮。此时您将看到一份原生应用程序示例列表。当您在左侧的列表中选择 [UI > UI Controls] 后，您将在右侧看到截屏图像。

此时，您需要为项目命名。此流程将导入一个现有示例，您采用与原始示例相同的名称即可。在“Project name”字段中键入“UIControls”，然后单击 Finish。



当您看到询问您是否要打开“Perspective”的弹出窗口时，请先选中“Remember my decision”复选框，然后单击 Yes。

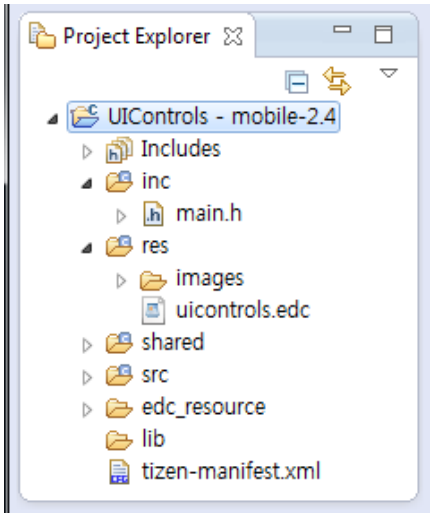


此弹出窗口将随即关闭，而对应示例则会添加到“Project Explorer”中。

此流程不会从 SDK 的“\platforms\”文件夹导入文件，而会导入您已复制到工作文件夹中的文件。当您打开您指定的工作文件夹 (C:\tizen-work) 后，您可以看到“UIControls”文件夹已被复制。

UIControls 示例可向您展示 Tizen 支持的所有类型小部件，以及如何使用 Container。

如果您单击已添加到“Project Explorer”中的“Basic App”左侧的“+”号，您将看到一份树状结构列表和一些文件夹。现在，让我们来看看每个文件夹下有哪些种类的文件。



“\src”文件夹中包含源文件 (.c)。“\inc”文件夹中包含头文件 (.h)。创建基础项目后，系统会自动生成该项目的主屏幕。

添加新屏幕后，您可以直接使用主屏幕的源文件。但我们建议您创建新的源文件，因为这样更便于针对其它项目进行管理和重复利用。

如果创建了库，您最好是使用新的头文件。

应用程序图标的图像都保存在“\shared\res”文件夹中。在您的智能手机中，您可通过点触图标列表中的某个特定图标来运行对应的应用程序。这种情况使用的是应用程序图标图像。

如果使用“tizen-manifest.xml”文件，您可以指定多种与应用程序有关的信息，包括应用程序 ID、SDK 版本、应用程序图标图像以及图标文本。

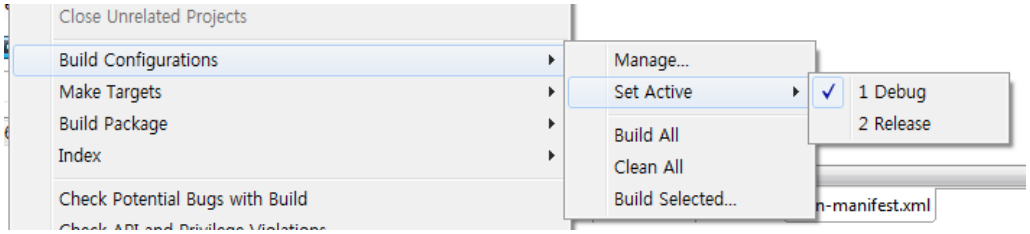
B. 构建样本

要使用模拟器运行源项目，您需要先构建 EXE 文件。Tizen 原生应用程序开发使用的是 C 语言，因此不会像在 Java 或 Android 环境中那样自动进行构建。同时，您还需要选择您所需的源项目构建模式。

构建模式分两种 — 调试模式和发布模式。在调试模式下，执行期间，您可在日志窗口中将所需信息显示为 Log 消息。此模式还可执行调试操作。

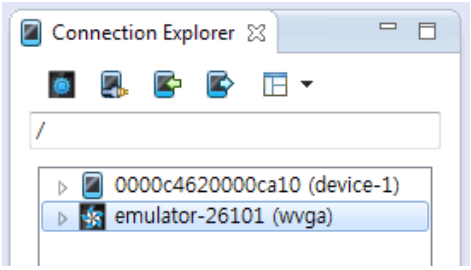
在发布模式下，您只可运行程序，不能查看 Log 消息。因此，在开发应用程序时，您应当使用调试模式，最终在卖方网站上注册时则需使用您在发布模式下构建的程序包。

默认设置为调试模式。如果您想更改为构建模式，请在“Project Explorer”中右键单击源项目，然后在 [Build Configurations > Set Active] 操作路径下选择您需要的项目。

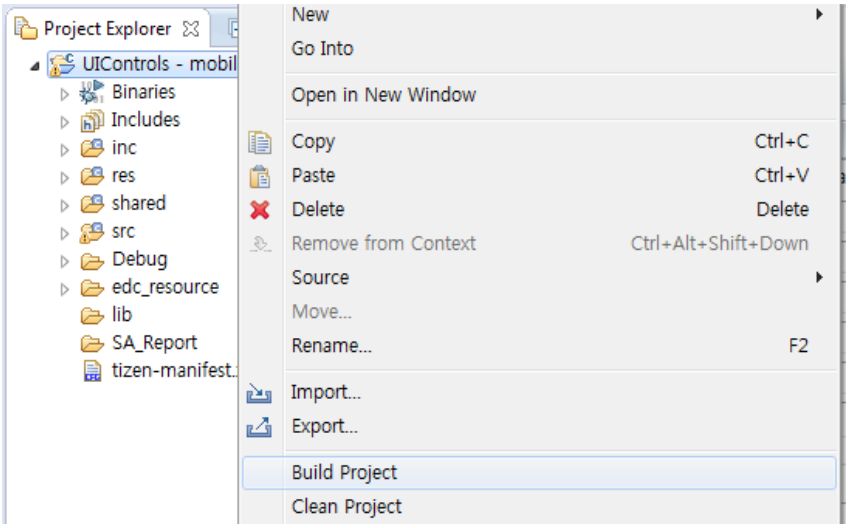


在 Eclipse 的左下方有一个“Connection Explorer”窗口。您将看到您刚刚创建的模拟器已经注册完毕。如果使用 USB 线缆连接某个终端，您还将看到增加的设备。

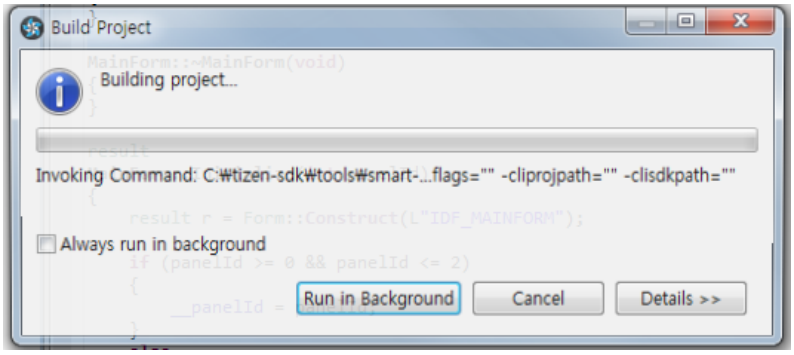
要运行模拟器，请在“Connection Explorer”窗口中选择模拟器，并构建您的项目。如果您想在终端上进行测试，您可以选择终端并构建您的项目。



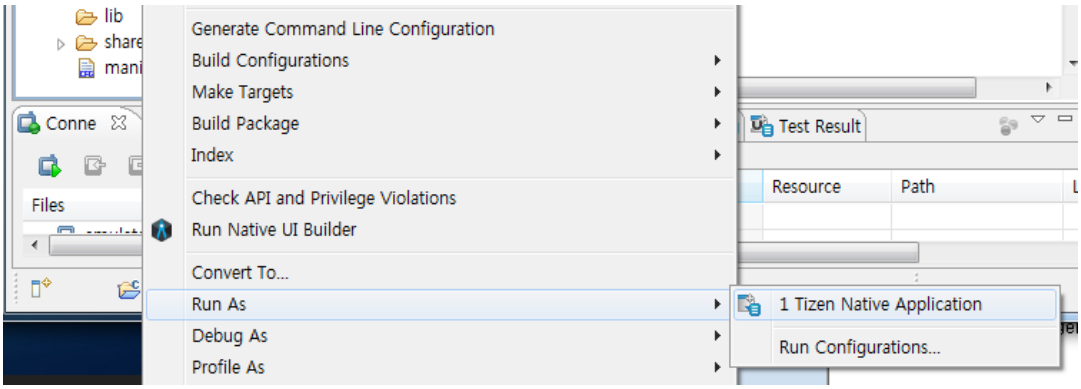
如果成功地指定了构建模式，则说明您已做好项目构建准备。在项目快捷菜单中选择 [Build Project] 以开始构建。



您将看到一个新的弹出窗口，其中会显示构建进展。请等待此窗口自动消失。如果弹出窗口消失并且“Problems”选项卡中没有显示任何错误，则表明您的项目构建成功。如果显示了任何错误消息，请纠正问题并重新构建。

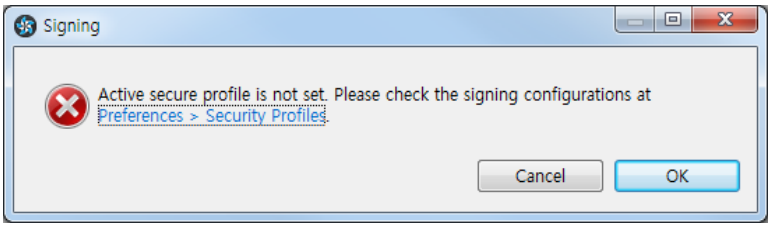


构建项目之后，即可运行项目。在源项目的快捷菜单中选择 [Run As > 1 Tizen Native Application]。

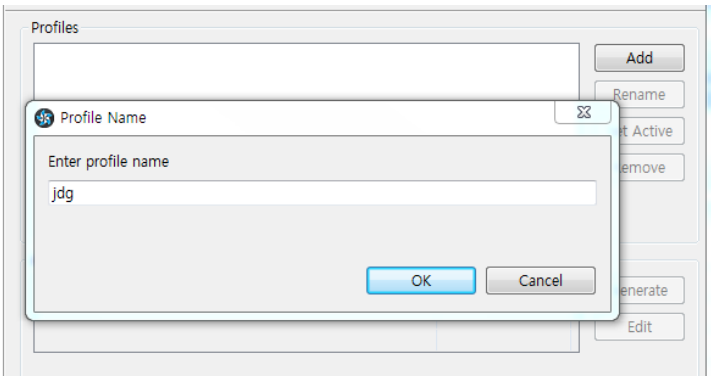


C. 创建证书

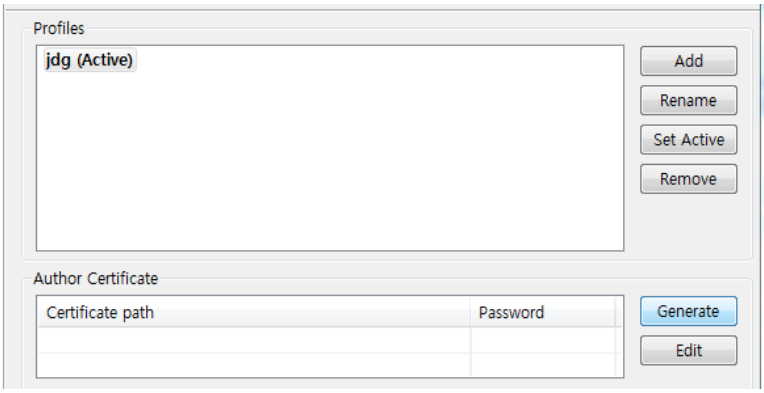
在此阶段，您可能会看到一个警告窗口，告知您 Secure Profile 不存在。自 Tizen SDK 2.1 起，您应按照以下操作步骤创建证书，以便在模拟器或终端中安装应用程序。



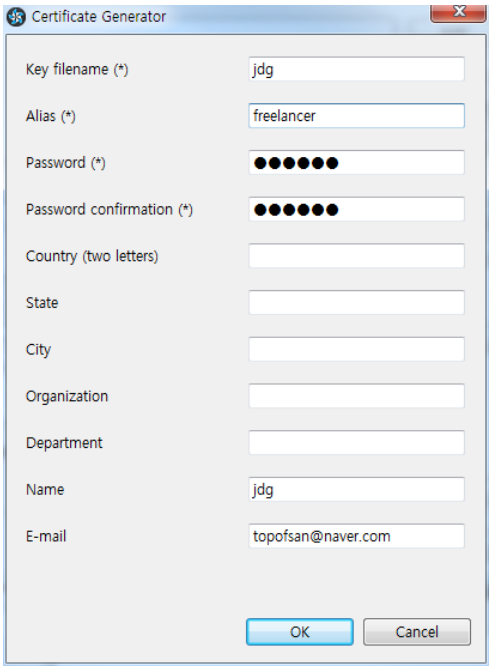
在弹出窗口中单击“Preferences > Security Profiles”位置关联的链接。随后将显示一个新的弹出窗口。单击右边的 Add 按钮。出现“Profile Name”弹出窗口后，输入您的配置文件名称，例如您的名称或贵公司的名称。单击 OK 按钮关闭弹出窗口。



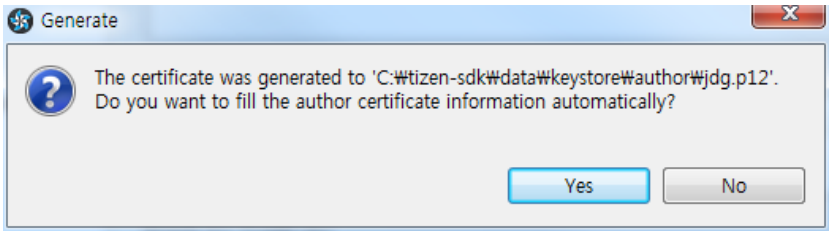
新项目将随即添加到 Profiles 中。此时，请输入详细信息。单击 Author Certification 右侧的 Generate 按钮。



出现 Certificate Generator 弹出窗口后，输入文件的名称作为 Key filename，输入附属机构的名称作为 Alias，并在 Password 和 Password confirmation 字段中输入密码。您可以使用 Profiles 的名称作为 Key filename 和 Alias。现在，您已输入使用时必需的所有信息。当您输完信息后，单击 OK 信息关闭弹出窗口。

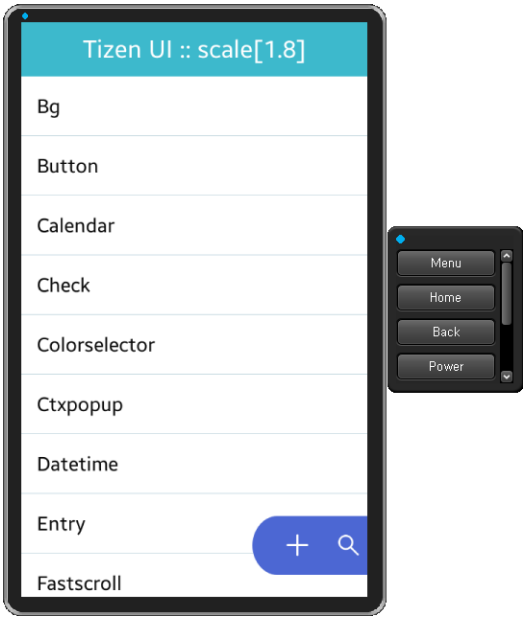


每当您打开此项目时，屏幕上都会显示一个弹出窗口询问您是否要使用此证书。此时您只需单击 Yes 按钮。



现在，请依次单击 Apply 和 OK 按钮，然后该弹出窗口将消失。当您单击“Signing”弹出窗口中的 OK 按钮后，源项目将安装在模拟器上。如果您以后需要编辑该证书，请在 Eclipse 的主菜单中选择 [Windows > Preferences]，并在出现“Preferences”弹出窗口后在窗口左侧树状结构的列表中选择 [Tizen SDK > Security Profiles]。

当应用程序成功安装到模拟器上之后，将会运行 UIControls。此示例全面展示了如何使用 UIControls Tizen 支持的小部件和容器。

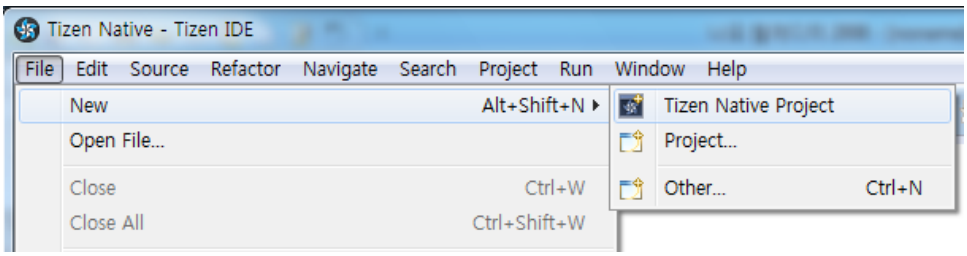


5. 创建 BasicUI 示例

在本节，我们将使用 BasicUI 方法和输出文本“Hello World”创建新的源项目。

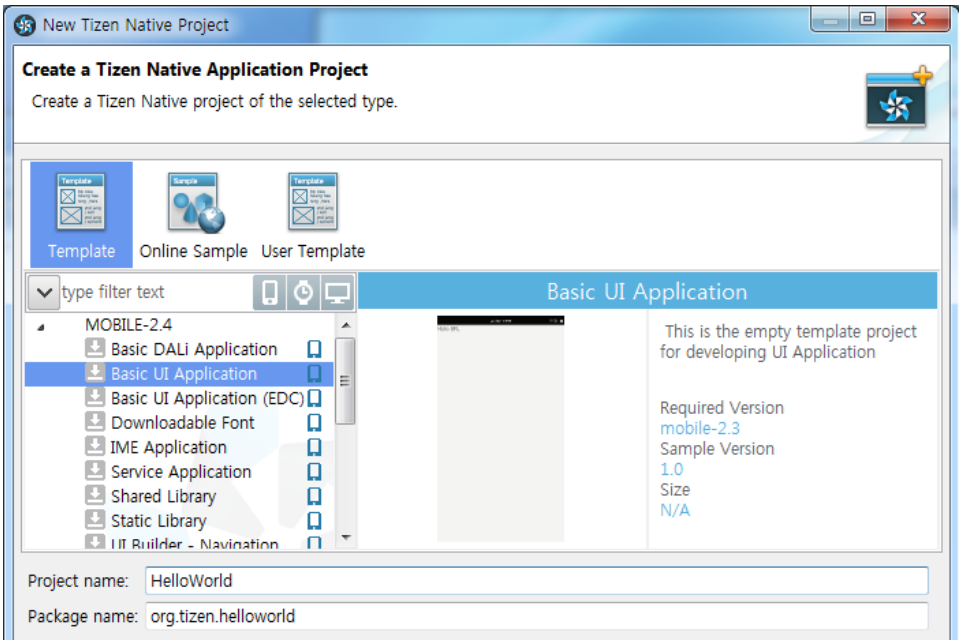
1）创建源项目

第一步，创建新的源项目。在 Eclipse 的主菜单中选择 [File > New > Tizen Native Project]。



出现源项目创建弹出窗口后，选择 [Template > MOBILE-2.x > Basic UI Application]。对于此处介绍的所有示例，您都可采用此方法。

在“Project name”字段中输入“HelloWorld”。



系统随即会自动填充“Package name”。如果您在开发将要注册应用程序商店的应用程序，您可使用该网站的地址作为“Package name”。例如，如果域名为“www.abc.com”，那么，您可输入“com.abc.helloworld”作为“Package name”。

此时，单击 Finish 按钮以创建新的源项目。

2）源项目的组件

让我们来看看源项目的基本组件。“\inc”文件夹中包含各种库。

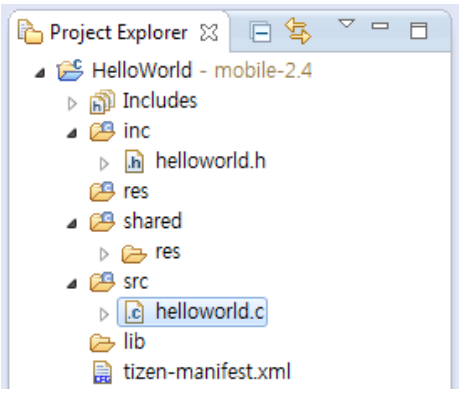
此文件夹中包含 C 语言 (.h) 中的头文件。通常，您可使用此文件夹定义各种库、函数的头文件以及全局变量。

“\res”文件夹通常用于保存包含图像和音频文件在内的资源文件。

“\src”文件夹中包 C 语言 (.c) 中的源文件。此文件夹主要用于定义函数的功能。大多数任务都是在此处执行的。

“\shared”文件夹中包含应用程序图标图像。当您将您的应用程序分发至应用程序商店后，您可以在此处保存您的应用程序图标。对于 Tizen Store，您应当使用圆形的应用程序图标。

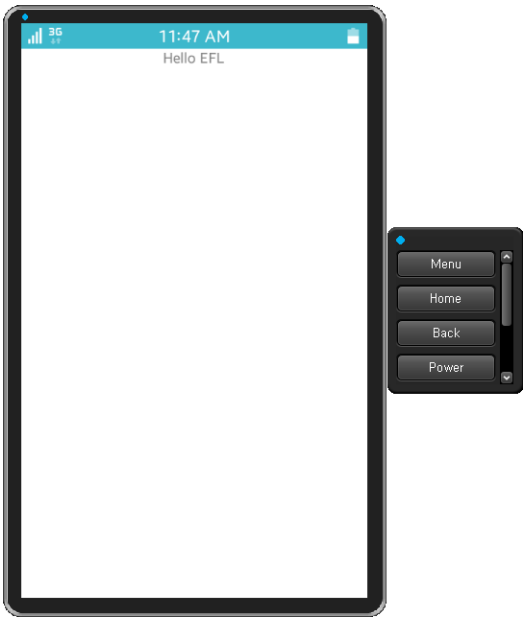
“tizen-manifest.xml”文件中包含各种应用程序信息（如，应用程序名称和版本）以及用户权限（特权）。基本上，该文件等同于 Android 系统中的 AndroidManifest.xml。



3）运行基本源项目

要运行源项目，请先右键单击“HelloWorld”项目。然后在快捷菜单中选择 [Build Project]。成功构建项目后，再次右键单击该项目，然后在快捷菜单中选择 [Run As > 1 Tizen Native Application]。

当该示例在模拟器上运行后，您将在界面顶部看到指示器，并在指示器下方看到应用程序屏幕。然后，您将看到“Hello EFL”文本。使用 Label 小部件时即会显示此文本。



4）基本源代码

现在，让我们更改一下 Label 小部件中显示的“Hello EFL”文本。要执行此操作，我们需要编辑源文件。打开“\src”文件夹，然后双击“helloworld.c”文件。此时您将在 Eclipse 主屏幕上看到文件内容，如下所示。

```
#include "helloworld.h"

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
} appdata_s;

static void
win_delete_request_cb(void *data, Evas_Object *obj, void *event_info)
{
        ui_app_exit();
}

static void
win_back_cb(void *data, Evas_Object *obj, void *event_info)
{
```

```
        appdata_s *ad = data;
        /* Let window go to hide state. */
        elm_win_lower(ad->win);
}
```

#include “helloworld.h”命令会引用“\inc”文件夹中的“helloworld.h”.通常，头文件的声明会确定要包含的库内容、全局变量以及函数的头文件，而源文件的定义则会用到函数内容。

“appdata_s”是一种用于保存应用程序中所用数据的结构。

“win_delete_request_cb()”是一种事件函数，会在请求删除应用程序时运行。此函数不是直接调用函数。

“win_back_cb()”是一种事件函数，会在单击 Back 按钮后运行。此函数不是直接调用函数。

接下来您将看到另一个函数“create_base_gui()”。此函数可创建由一个屏幕、多个容器以及一个小部件组成的窗口。

```
static void
create_base_gui(appdata_s *ad)
{
        /* Window */
        ad->win = elm_win_util_standard_add(PACKAGE, PACKAGE);
        elm_win_autodel_set(ad->win, EINA_TRUE);

        if (elm_win_wm_rotation_supported_get(ad->win)) {
                int rots[4] = { 0, 90, 180, 270 };
                elm_win_wm_rotation_available_rotations_set(ad->win, (const int
*)(&rots), 4);
        }

        evas_object_smart_callback_add(ad->win, "delete,request", win_delete_req
uest_cb, NULL);
        eext_object_event_callback_add(ad->win, EEXT_CALLBACK_BACK, win_back_cb,
 ad);

        /* Conformant */
```

```
        ad->conform = elm_conformant_add(ad->win);
        elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
        elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
        evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HIN
T_EXPAND);
        elm_win_resize_object_add(ad->win, ad->conform);
        evas_object_show(ad->conform);

        /* Label*/
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
        evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_
EXPAND);
        elm_object_content_set(ad->conform, ad->label);

        /* Show window after base gui is set up */
        evas_object_show(ad->win);
}
```

下文对一些主要代码行进行了解释。

elm_win_util_standard_add() 是一种用于创建窗口对象的 API。窗口是位于屏幕布局最上层的对象。每个应用程序配有单独的窗口。您可在窗口中放置一个小部件。但是，更常见的方法是添加容器，然后在容器之上添加小部件。

elm_win_wm_rotation_available_rotations_set() 是一种用于指定屏幕方向的 API。如果提供 0、90、180 和 270 四个角度用于阵列排布，则支持所有屏幕方向。

evas_object_smart_callback_add() 是一种用于指定小部件或容器这类智能对象的事件回调函数的 API。对于第一个参数，请提供发生事件的对象；对于第二个参数，请提供事件类型；对于第三个参数，请提供回调函数的名称；对于第四个对象，请提供用户数据。如果事件类型为“delete,request”，则表明应当删除对象。

eext_object_event_callback_add() 是一种用于指定对象的事件回调函数的 API。您可为智能对象和常规对象使用此 API。对于第一个参数，请提供发生事件的对象；对于第二个参数，请提供事件类型；对于第三个参数，请提供回调函数的名称；对于第四个对象，请提供用户数据。EEXT_CALLBACK_BACK 指示了 Back 按钮单击事件。

elm_conformant_add() 是一种用于创建 Conformant 的 API。Conformant 可在新元素（例如，键盘）添加到屏幕中后更改窗口大小。一个应用程序只能有一个 Conformant。应用程序也可以没有 Conformant。
但若要在屏幕顶部显示指示工具（状态栏），则需用到 Conformant。在存在 Conformant 的情况下，也可以不显示指示工具。

elm_win_indicator_mode_set() 是一种用于指定是否显示指示工具的 API。

elm_win_indicator_opacity_set() 是一种用于指定指示工具的透明度的 API。

evas_object_size_hint_weight_set() 是一种用于指定对象的大致尺寸的 API。以下是几个按顺序列出的参数：对象、水平尺寸提示以及垂直尺寸提示。EVAS_HINT_EXPAND 表示尽可能按照允许的空间最大尺寸来指定大小。

elm_win_resize_object_add() 是一种 API，用于在有不同对象被添加到窗口中时调整窗口对象的大小。

evas_object_show() 是一种用于显示对象的 API。创建对象后，对象的默认值为“Hide”。此函数可用于所有对象，无一例外。

elm_label_add() 是一种用于创建 Label 小部件的 API。Label 小部件可供您显示文本，以及通过使用 HTML 标记来更改文本属性，比如字体大小和颜色。

5) 更改 Label 中的文本

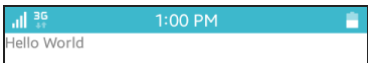
现在，我们将要更改屏幕上显示的“Hello EFL”文本。更改“elm_object_text_set()”函数，如下所示。

```
        ad->label = elm_label_add(ad->conform);
        //elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
        elm_object_text_set(ad->label, "Hello World");
        evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_
EXPAND);
```

elm_object_text_set() 是一种用于更改小部件的标题文本的 API。您可对 Button 小部件、Entry 小部件以及 Label 小部件使用此函数。您还可通过在创建文本时使用 HTML 标记来指定文本属性。

让我们再运行一次示例。当您再次运行示例时，请在主菜单中单击 [Run > Run]，或按下“Ctrl + F11”热键组合。

此示例将再次运行，屏幕上的文本则会变为“Hello World”。



6) 使用绝对坐标更改 Label 位置

Label 位于屏幕左侧。evas_object_size_hint_weight_set() 函数用于指定对象的相对大小，这是因为该函数的水平和垂直选项均被指定为“EVAS_HINT_EXPAND”。对于第一个参数，请输入要指定其属性的对象；对于第二个参数，请输入水平尺寸提示；对于第三个参数，请输出垂直尺寸提示。

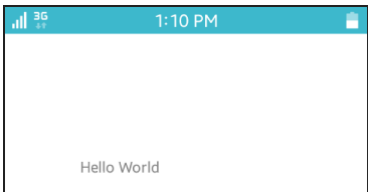
现在，让我们通过指定 Label 的绝对坐标来更改标签位置。更改代码，如下所示。

```
        elm_object_text_set(ad->label, "Hello World");
        //evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
        //elm_object_content_set(ad->conform, ad->label);
        evas_object_move(ad->label, 100, 200);
        evas_object_resize(ad->label, 400, 100);
        evas_object_show(ad->label);
```

evas_object_move() 是一种用于将对象大小指定为绝对值的函数。按照以下顺序输入参数值：要为其指定属性的对象、X 坐标位置以及 Y 坐标位置。

evas_object_resize() 是一种用于将对象位置指定为绝对值的函数。按照以下顺序输入参数值：要为其指定属性的对象、X 坐标（宽）以及 Y 坐标（高）。

再次运行应用程序，便会看到 Label 位置已改变。



7）使用 Box 容器支持多种分辨率

按上文所述来指定绝对坐标很方便。但这种方式却不适合为终端设备调整不同分辨率。现在我们将要学习如何使用 Box 容器来支持多种分辨率。

在 create_base_gui() 函数之上添加新函数。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
```

```
   /* show the frame */
   evas_object_show(frame);
}
```

my_box_pack() 是一种用于将小部件添加到 Box 容器中的函数。Box 是一种容器，可沿水平或垂直方向接连添加小部件或容器。其作用类似于 Android 系统中的 LinearLayout。此函数将在后续多个示例中得到运用。我们将在 Box 示例中详细介绍 Box 容器。

对于 my_box_pack() 函数的第一个参数，请输入 Box 容器的句柄。对于第二个参数，请输入将要添加到 Box 容器中的小部件或容器。
对于第三个参数，请输入小部件的水平尺寸提示。输入 EVAS_HINT_EXPAND 或 1.0 可指定小部件的最大水平尺寸；输入 0.0 则指定最小水平尺寸。对于 Bg 小部件，默认尺寸为 0。因此，如果指定 0.0，小部件将不会出现在屏幕上。
对于第四个参数，请输入小部件的垂直尺寸提示。
对于第五个参数，请输入小部件的水平位置。输入 0.0 将会靠左对齐小部件；输入 0.5 将会居中对齐小部件；输入 1.0 将会靠右对齐小部件；输入 EVAS_HINT_FILL 或 -1 则会使小部件覆盖整个水平区域。
对于第六个参数，请输入小部件的垂直位置。输入 0.0 将会靠上对齐小部件；输入 0.5 将会居中对齐小部件；输入 1.0 将会靠下对齐小部件；输入 EVAS_HINT_FILL 或 -1 则会使小部件覆盖整个垂直区域。

修改 create_base_gui() 函数底部的源代码，如下所示。

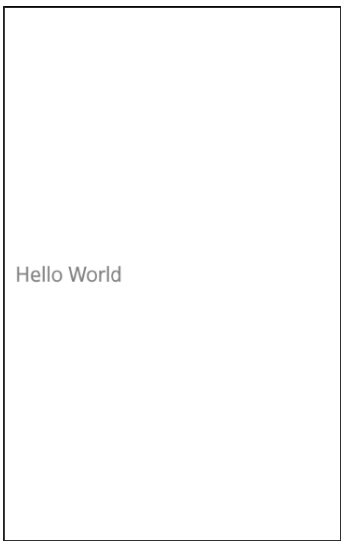
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
```

```
        evas_object_show(box);

        {

            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "Hello World");
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

使用 { } 符号是为了阐明上下窗口的包含关系。并无必要使用此符号。但是，由于使用此符号的可读性效果和不使用此符号的可读性效果之间相差甚远，因此，建议您养成使用此符号的习惯。

elm_box_add() 是一种用于创建 Box 容器的 API。

让我们再运行一次示例。效果变化不大。但此时该小部件在不同分辨率的终端屏幕上都显示在同一位置。



8）无效代码修改

由于在创建源项目时会默认添加一些源代码，通过删除一些不必要的源代码，可提升性能。让我们使用 EFL 开发人员 Carsten Haitzler 推荐的方法来修改一下源代码。

如下文所示，转至源文件顶部，删除 win_delete_request_cb() 函数并修改 win_back_cb() 函数。

```
/*static void
win_delete_request_cb(void *data, Evas_Object *obj, void *event_info)
{
        ui_app_exit();
}*/

static void
win_back_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        /* Let window go to hide state. */
        //elm_win_lower(ad->win);
        elm_win_iconified_set(ad->win, EINA_TRUE);
}
```

win_delete_request_cb() 是用在以下 create_base_gui() 函数代码中的回调函数。这是用于 PC 的事件，因此几乎不会发生在移动设备上。

Evas_object_smart_callback_add(ad->win, "delete,request", win_delete_request_cb, NULL);

如下文所示，转至 create_base_gui() 函数并修改函数的开头部分。

```
static void
create_base_gui(appdata_s *ad)
{
    /* set up policy to exit when last window is closed */
    elm_policy_set(ELM_POLICY_QUIT, ELM_POLICY_QUIT_LAST_WINDOW_CLOSED);
```

```
    /* Window */
    ad->win = elm_win_util_standard_add(PACKAGE, PACKAGE);
    elm_win_autodel_set(ad->win, EINA_TRUE);

    int rots[4] = { 0, 90, 180, 270 };
    elm_win_wm_rotation_available_rotations_set(ad->win, (const int *)(&rots),
4);

    eext_object_event_callback_add(ad->win, EEXT_CALLBACK_BACK, win_back_cb, a
d);

    /* child object - indent to how relationship */
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);
```

使用 elm_policy_set() 函数，在最后一个窗口关闭时关闭应用程序。

移动设备均支持 elm_win_wm_rotation_supported_get() 函数，而不受型号限制。因此，不需要考虑此函数，此函数已删除。

最后，让我们删除不必要的事件回调函数。转至源文件底部，删除以下四种回调函数：
- ui_app_orient_changed()
- ui_app_region_changed()
- ui_app_low_battery()
- ui_app_low_memory()

然后在源文件最下方的 main() 函数部分，删除用于将上述函数指定为回调函数的代码部分。

```
int
main(int argc, char *argv[])
{
    appdata_s ad = {0,};
```

```
    int ret = 0;
    ui_app_lifecycle_callback_s event_callback = {0,};
    app_event_handler_h handlers[5] = {NULL, };

    event_callback.create = app_create;
    event_callback.terminate = app_terminate;
    event_callback.pause = app_pause;
    event_callback.resume = app_resume;
    event_callback.app_control = app_control;

    //ui_app_add_event_handler(&handlers[APP_EVENT_LOW_BATTERY], APP_EVENT_LOW_B
ATTERY, ui_app_low_battery, &ad);
    //ui_app_add_event_handler(&handlers[APP_EVENT_LOW_MEMORY], APP_EVENT_LOW_ME
MORY, ui_app_low_memory, &ad);
    //ui_app_add_event_handler(&handlers[APP_EVENT_DEVICE_ORIENTATION_CHANGED],
APP_EVENT_DEVICE_ORIENTATION_CHANGED, ui_app_orient_changed, &ad);
    ui_app_add_event_handler(&handlers[APP_EVENT_LANGUAGE_CHANGED], APP_EVENT_LA
NGUAGE_CHANGED, ui_app_lang_changed, &ad);
    //ui_app_add_event_handler(&handlers[APP_EVENT_REGION_FORMAT_CHANGED], APP_E
VENT_REGION_FORMAT_CHANGED, ui_app_region_changed, &ad);
    //ui_app_remove_event_handler(handlers[APP_EVENT_LOW_MEMORY]);

    ret = ui_app_main(argc, argv, &event_callback, &ad);
    if (ret != APP_ERROR_NONE) {
        dlog_print(DLOG_ERROR, LOG_TAG, "app_main() is failed. err = %d", ret);
    }

    return ret;
}
```

9) 相关 API

appdata_s：一种用于保存应用程序信息的结构。

void create_base_gui(appdata_s *ad)：一种用于创建窗口以及构成屏幕的各种容器和小部件的函数。

void win_delete_request_cb()：一种事件函数，会在请求删除应用程序时执行。此函数不是直接调用函数。

void win_back_cb()：一种事件函数，会在单击 Back 按钮后执行。此函数不是直接调用函数。

Evas_Object *elm_win_util_standard_add(char *name, char *title)：一种用于创建窗口对象的函数。窗口是位于屏幕布局最上层的对象。每个应用程序配有单独的窗口。您可在窗口中放置一个小部件。但是，更常见的方法是添加容器，然后在容器之上添加小部件。

void elm_win_wm_rotation_available_rotations_set(Elm_Win *obj, const int *rotations, unsigned int count)：一种用于指定屏幕方向的 API。如果提供 0、90、180 和 270 四个角度用于阵列排布，则支持所有屏幕方向。

void evas_object_smart_callback_add(Evas_Object *obj, const char *event, Evas_Smart_Cb func, const void *data)：一种用于指定小部件或容器这类智能对象的事件回调函数的 API。对于第一个参数，请提供发生事件的对象；对于第二个参数，请提供事件类型；对于第三个参数，请提供回调函数的名称；对于第四个对象，请提供用户数据。如果事件类型为“delete, request”，则表明应当删除对象。

void eext_object_event_callback_add(Evas_Object *obj, Eext_Callback_Type type, Eext_Event_Cb func, void *data)：一种用于指定对象的事件回调函数的 API。您可为智能对象和常规对象使用此 API。对于第一个参数，请提供发生事件的对象；对于第二个参数，请提供事件类型；对于第三个参数，请提供回调函数的名称；对于第四个对象，请提供用户数据。EEXT_CALLBACK_BACK 指示了 Back 按钮单击事件。

Evas_Object *elm_conformant_add(Evas_Object *parent)：一种用于创建 Conformant 容器的函数。Conformant 可在键盘这类新元素添加到屏幕中后更改窗口大小。一个应用程序只能有一个 Conformant。应用程序也可以没有 Conformant。但若要在屏幕顶部显示指示工具（状态栏），则需用到 Conformant。在存在 Conformant 的情况下，也可以不显示指示工具。

void elm_win_indicator_mode_set(Elm_Win *obj, Elm_Win_Indicator_Mode mode)：一种用于指定是否显示指示工具的 API。

void elm_win_indicator_opacity_set(Elm_Win *obj, Elm_Win_Indicator_Opacity_Mode mode)：一种用于指定指示工具透明度的 API。

void evas_object_size_hint_weight_set(Evas_Object *obj, double x, double y)：一种用于指定对象的粗略尺寸的 API。以下是几个按顺序列出的参数：对象、水平尺寸提示以及垂直尺寸提示。EVAS_HINT_EXPAND 表示尽可能按照允许的空间最大尺寸来指定大小。

void elm_win_resize_object_add(Elm_Win *obj, Evas_Object *subobj)：一种 API，用于在有不同对象被添加到窗口中时调整窗口对象的大小。

Evas_Object *elm_label_add(Evas_Object *parent)：一种用于创建 Label 小部件的函数。Label 小部件可供您显示文本，以及通过使用 HTML 标记来更改文本属性，比如字体大小和颜色。

void evas_object_show(Evas_Object *obj)：一种用于在屏幕上显示对象的函数。创建对象后，对象的默认值为“Hide”。基本上您可以为所有对象使用 evas_object_show() 函数。

void elm_object_text_set(Evas_Object *obj, const char *text)：一种用于更改小部件标题文本的函数。您可对 Button 小部件、Entry 小部件以及 Label 小部件使用此函数。

void evas_object_move(Evas_Object *obj, Evas_Coord x, Evas_Coord y)：一种用于将对象大小指定为绝对值的 API。/ 参数：您要为其指定属性的对象、X 坐标位置以及 Y 坐标位置。

void evas_object_resize(Evas_Object *obj, Evas_Coord w, Evas_Coord h)：一种用于将对象位置指定为绝对值的 API。/ 参数：您要为其指定属性的对象、宽度以及 Y 坐标（高）。

6. 使用 Label 小部件

您需要使用 Label 小部件在屏幕上显示文本。Label 小部件可供您通过使用 HTML 标记来更改文本属性，如字体大小和颜色。

1) 将 Label 文本居中对齐

创建新的源项目，并将项目名称指定为“LabelEx”。要执行此操作，请在 Eclipse 的主菜单中，选择 [File > New > Tizen Native Project]，在弹出窗口出现后，选择 [Template > MOBILE-2.4 > Basic UI Application]。

创建源项目之后，打开 src 文件夹中的 labelex.c 文件，转至 create_base_gui() 函数，并如下文所示修改 Label 小部件的创建代码。

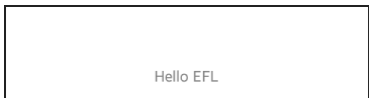
```
        /* Conformant */
        ad->conform = elm_conformant_add(ad->win);
        elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
        elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
        evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
        elm_win_resize_object_add(ad->win, ad->conform);
        evas_object_show(ad->conform);

        /* Label-1 */
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
        //evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
        //elm_object_content_set(ad->conform, ad->label);
        evas_object_move(ad->label, 120, 80);
        evas_object_resize(ad->label, 240, 80);
        evas_object_show(ad->label);
```

elm_object_text_set() 是一种用于指定小部件的标题文本的 API。输入与所需第二参数属性对应的 HTML 标记，以将这些属性应用至文本。<align=center> 将文本居中对齐。

evas_object_move() 是一种用于指定小部件的左上起始位置的 API。对于第二个参数，请输入所需 X 坐标。对于第三个参数，请输入所需 Y 坐标。

evas_object_resize() 是一种用于指定对象尺寸的 API。对于第二个参数，请输入所需宽度。对于第三个参数，请输入所需高度。

构建并运行源项目，您随后将看到“Hello EFL”文本显示在屏幕上。该文本将在水平方向上居中对齐。



2) 更改字体大小

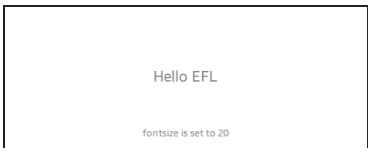
接下来我们将更改 Label 小部件标题文本的字体大小。在 create_base_gui() 函数末尾添加新代码。此代码可创建第二个 Label 小部件。

```
    /* Label-1 */
    ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
    //evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    //elm_object_content_set(ad->conform, ad->label);
    evas_object_move(ad->label, 120, 80);
    evas_object_resize(ad->label, 240, 80);
    evas_object_show(ad->label);

    /* Label-2 */
    Evas_Object *label = elm_label_add(ad->conform);
    elm_object_text_set(label, _("<font_size=20><align=center>fontsize is set t
o 20</align></font_size>"));
    evas_object_move(label, 120, 160);
    evas_object_resize(label, 240, 80);
    evas_object_show(label);

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

Evas_Object 是一种适用于屏幕对象（如小部件和容器）的通用变量。因此，无论是否将“Conformant”和“Label”声明为相同变量类型，都没有任何关系。

elm_object_text_set() 是一种用于更改小部件的标题文本的函数。您可对 Button 小部件、Entry 小部件以及 Label 小部件使用此函数。如果将 HTML 标记插入此函数，则会按照您在 Web 浏览器上看到样式显示文本。
<font_size=20> 是一种标记，可将字体大小指定为 20 像素。
<align=center> 是一种标记，可将水平对齐位置指定为居中。

再次运行示例，便会看到第二个 Label 小部件已添加进来，并会显示“font size is set to 20”文本。该文本显示的大小将小于第一个 Label 中的文本大小。已对水平对齐文本应用了居中对齐。



3) 更改字体颜色

接下来我们将更改 Label 小部件中的字体颜色。就像设置字体大小一样，您也可以使用 HTML 标记来更改字体颜色。在 create_base_gui() 函数末尾添加新代码。

```
    evas_object_show(label);

    /* Label-3 */
    label = elm_label_add(ad->conform);
    elm_object_text_set(label, _("<color=#FF4500FF><align=right>font color oran
ge red</align></color>"));
    evas_object_move(label, 50, 240);
    evas_object_resize(label, 380, 80);
    evas_object_show(label);

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

插到 elm_object_text_set() 函数中的 <color=#FF4500FF> 是一种标记，用于指定颜色。已采用 AARRGGBB 格式。透明度值达到最大值，等同于不透明效果；蓝色值为 45；绿色值为 0；红色值为最大值。输入 <color=#F40F> 会生成相同结果。

再次运行示例，便会看到第三个 Label 小部件已添加进来，并会显示橙色文本。

4）省略号

接下来，我们将学习当给定文本太长、超出 Label 右端边缘时如何显示省略号。在 create_base_gui() 函数末尾添加新代码。

```
    evas_object_show(label);

    /* Label-4 */
    label = elm_label_add(ad->conform);
    elm_object_text_set(label, _("<font_size=24>If the string length exceeds th
e width</font_size>"));
    evas_object_move(label, 72, 320);
    evas_object_resize(label, 560, 80);
    elm_label_ellipsis_set(label, EINA_TRUE);
    evas_object_show(label);

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

elm_label_ellipsis_set() 是一种用于向 Label 小部件应用省略号的 API。对于第一个参数，请提供要为其指定属性的 Label 小部件，对于第二个参数，请提供 true 或 false 值。EINA_TRUE 是一种布尔值，用于在 Tizen 中表示“true”。要指定“false”值，请输入 EINA_FALSE。

再次运行示例，便会看到屏幕上显示第四处文本，并在文本右端结尾处显示省略号。

5) 多行文本

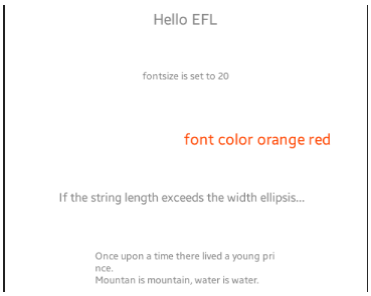
要在 Label 小部件中显示多行文本，请使用
 标记。在 create_base_gui() 函数末尾添加新代码。

```
    evas_object_show(label);

    /* Label-5 */
    label = elm_label_add(ad->conform);
    elm_label_line_wrap_set(label, EINA_TRUE);
    elm_object_text_set(label, _("<font_size=20><align=left>Once upon a time there lived a young prince.<br>Mountan is mountain, water is water. </align></font_size>"));
    evas_object_move(label, 120, 400);
    evas_object_resize(label, 240, 160);
    evas_object_show(label);

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

elm_label_line_wrap_set() 是一种用于为 Label 小部件设置自动换行的API。为第二个参数提供 EINA_TRUE 即可应用自动换行。

我们已创建第五个 Label 小部件并输入了长文本。我们使用
 标记将文本分成了两行。

再次运行示例，便会看到屏幕上显示第五处文本，并会看到文本在
 标记插入处换行。



6) 相关 API

Evas_Object：一种适用于屏幕对象（如小部件和容器）的通用变量。因此，无论是否将“Conformant”和“Label”等不同对象声明为相同变量类型，都没有任何关系。

void elm_object_text_set(Evas_Object *obj, char *text)：一种用于更改标题文本的 API。您可对 Button 小部件、Entry 小部件以及 Label 小部件使用此函数。如果将 HTML 标记插入此函数，则会按照您在 Web 浏览器上看到样式显示文本。

EINA_TRUE：一种布尔值，用于在 Tizen 中表示“true”。
EINA_FALSE：一种布尔值，用于在 Tizen 中表示“false”。

void elm_label_ellipsis_set(Evas_Object *obj, Eina_Bool ellipsis)：一种 API，用于在 Label 小部件标题文本超出文本行右端末尾位置时显示省略号。如果为第二个参数输入 EINA_TRUE，屏幕上将显示省略号；如果输入 EINA_FALSE，则会取消显示省略号。

7. 使用 Button 小部件

Button 小部件可接收用户输入的内容，最常用于各种小部件。Button 小部件可调用触摸事件，并能使用 EDJE 应用背景图像。

1）更改 Label 小部件中的文本

创建新的源项目，并将项目名称指定为“ButtonEx”。

创建源项目之后，打开 src 文件夹中的 buttonex.c 文件，并在 create_base_gui() 函数之上添加新函数。您添加的函数与 HelloWorld 示例中所用的函数相同。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
    /* create a frame we shall use as padding around the child widget */
    Evas_Object *frame = elm_frame_add(box);
    /* use the medium padding style. there is "pad_small", "pad_medium",
     * "pad_large" and "pad_huge" available as styles in addition to the
     * "default" frame style */
    elm_object_style_set(frame, "pad_medium");
    /* set the input weight/aling on the frame insted of the child */
    evas_object_size_hint_weight_set(frame, h_weight, v_weight);
    evas_object_size_hint_align_set(frame, h_align, v_align);
    {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
    }
    /* put the frame into the box instead of the child directly */
    elm_box_pack_end(box, frame);
    /* show the frame */
    evas_object_show(frame);
}
```

然后，转至 create_base_gui() 函数，并修改用来创建 Label 小部件的代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "<align=center>Press a Button</>");
            //evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVA
S_HINT_EXPAND);
            //elm_object_content_set(ad->conform, ad->label);
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

以下是按顺序列出的 my_box_pack() 函数参数：
- Box 容器
- 子窗口
- 水平尺寸提示：1.0 = 最大值。0.0 = 最小值。
- 垂直尺寸提示：1.0 = 最大值。0.0 = 最小值。
- 水平位置：0.0 = 左。0.5 = 中。1.0 = 右。-1 = 全屏。
- 垂直位置：0.0 = 上。0.5 = 中。1.0 = 下。-1 = 全屏。

运行示例。您将看到“Press Button”文本显示在屏幕上。

2) 创建 Button 小部件

接下来我们将在 create_base_gui() 函数末尾添加 Button 小部件的创建代码。

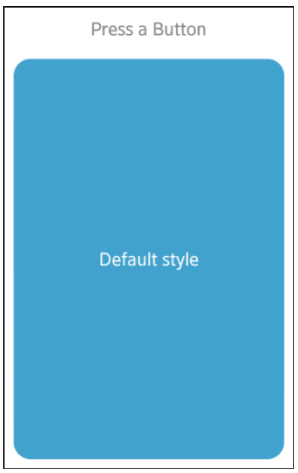
```
        {
        /* Label*/
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "<align=center>Press a Button</>");
        my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);

        /* Button-1 */
        Evas_Object *btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Default style");
        my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
    }
```

elm_button_add() 是一种用于创建 Button 小部件的 API。

使用 elm_object_text_set() 函数将标题文本指定为“Default style”。

其它函数已在之前示例中予以介绍。

构建并运行源项目，随后会看到 Button 小部件已添加进来，并且屏幕上会显示“Default style”文本。点击此按钮不会更改任何内容。这是因为我们尚未定义事件函数。



3) 定义 Button 事件函数

Tizen 中使用的 Enlightenment Foundation Libraries (EFL) 定义了回调样式的事件函数。如果您有 Web 编程经验，则会熟悉这种方法。现在让我们来定义 Button 事件函数。添加名为“btn_default_cb()”的函数。

需要注意的是，由于此函数将通过 create_base_gui() 调用，因此必须将 btn_default_cb() 置于 create_base_gui() 之前。在标题文件中声明各函数标题，即可让各函数之间相互调用，而不受次序限制。

```
static void
btn_default_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s* ad = data;
        elm_object_text_set(obj, "Button Pressed");
        elm_object_text_set(ad->label, "Button-1 Pressed");
}

static void
```

```
create_base_gui(appdata_s *ad)
{
```

btn_default_cb() 函数会接收三个参数。第一个参数是调用方发送的用户数据。在此例中，我们将使用应用程序数据 (appdata_s)。第二个参数是发生了事件的对象。在此例中，对象即为首个 Button。第三个参数是包含事件信息的结构。

elm_object_text_set() 是一种用于更改小部件的标题文本的 API。

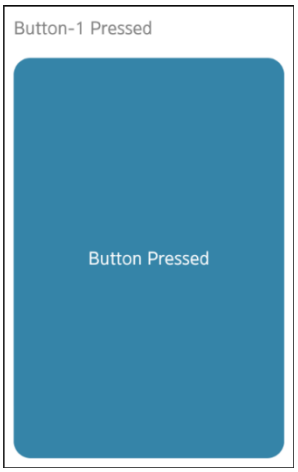
elm_object_text_set() 是一种用于更改小部件的标题文本的 API。

现在，我们需要转至 Button 创建代码部分，将上述函数指定为回调函数。转至 create_base_gui() 函数并添加一行代码。

```
        /* Button-1 */
        Evas_Object *btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Default style");
        evas_object_smart_callback_add(btn, "clicked", btn_default_cb, ad);
        my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
```

evas_object_smart_callback_add() 是一种函数，用于指定小部件或容器这类智能对象的事件回调函数。Evas 对象为屏幕上显示的所有对象。智能对象是除 Evas 提供的基本对象（Line、Rect、Polygon、Text、Image）以外的补充对象。对于第一个参数，请输入发生事件的对象。对于第二个参数，请输入事件类型。“clicked”指示的是单击事件。对于第三个参数，请输入回调函数的名称。对于第四个参数，请输入将要发送给回调函数的数据类型。在此例中，所需内容为应用程序数据。

再次运行示例，并单击 Button。Label 和 Button 中的文本将发生变化。



3）向 Button 应用图标：重新排序

一个 Button 中可以显示若干个图标图像。我们将学习如何应用各个图标图像。

在 create_base_gui() 函数末尾添加新代码。此代码可创建第二个 Button。

```
            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Default style");
            evas_object_smart_callback_add(btn, "clicked", btn_default_cb, ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            /* Button-2 */
            btn = elm_button_add(ad->conform);
            elm_object_style_set(btn, "icon_reorder");
            evas_object_smart_callback_add(btn, "clicked", btn_icon_reorder_cb,
ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
```

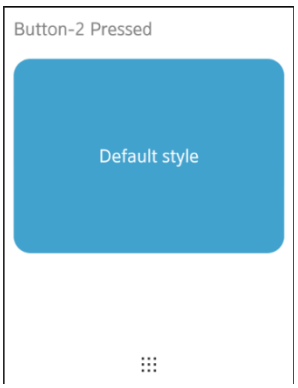
elm_object_style_set() 是一种用于指定小部件样式的函数。对于第一个参数，请输入将要应用某个样式的小部件。对于第二个参数，请输入样式类型。“icon_reorder”会显示重新排序图标。

我们已为 evas_object_smart_callback_add() 指定了回调函数 btn_icon_reorder_cb。此函数尚未实施。如下所示，在 create_base_gui() 函数之上添加新函数。

```
static void
btn_icon_reorder_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s* ad = data;
        elm_object_text_set(ad->label, "Button-2 Pressed");
}
```

我们为第二个 Button 定义了回调函数，并向 Label 小部件添加了文本变更代码。

再次运行示例，随后便会看到带有图标图像的另一个 Button 已添加进来。单击第二个 Button。Label 和第二个 Button 中的文本将发生变化。



4) 向 Button 应用图标：+ 和 -

现在我们将创建一个已应用加号图标和减号图标的 Button。在 create_base_gui() 函数末尾添加新代码。此代码可创建第三个 Button。

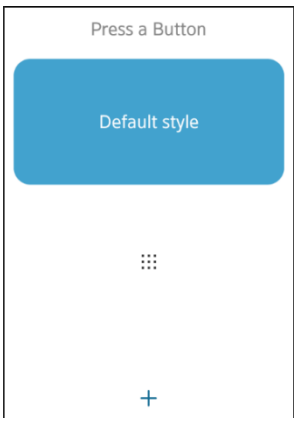
```
        /* Button-2 */
        btn = elm_button_add(ad->conform);
        elm_object_style_set(btn, "icon_reorder");
        evas_object_smart_callback_add(btn, "clicked", btn_icon_reorder_cb,
```

```
ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            /* Button-3 */
            btn = elm_button_add(ad->conform);
            elm_object_style_set(btn, "icon_expand_add");
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
```

为 elm_object_style_set() 函数的第二个参数输入“icon_expand_add”，随后将会显示一个加号图像。

回调函数的指定方法与前文所述的操作过程相同，此处不再赘述。

再次运行示例，随后便会看到带加号图标图像的第三个 Button 已添加进来。



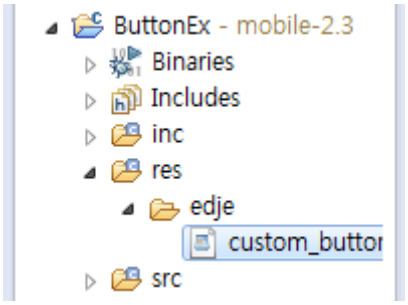
5）向 Button 应用背景图像：

要向 Button 小部件应用背景图像，您需要使用主题。EFL 使用名为 EDJE 的主题。从头创建主题文件很麻烦，因此我们将从附录中导入主题文件。

在源项目的 /res 文件夹下创建一个新文件夹，然后将该文件夹的名称指定为“edje”。要执行此操作，请右键单击 /res 文件夹，然后在快捷菜单中选择 [New > Folder]。弹出窗口出现后，请在文件夹名称字段中输入“edje”，然后单击“Finish”按钮。

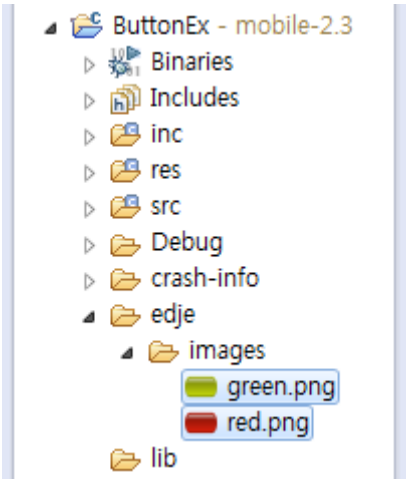
转至附录的 /etc/edje 文件夹，将 custom_button.edc 文件复制到新建的 /res/edje 文件夹。

可通过右键单击 /res 文件夹并在快捷菜单中选择 [New > Folder] 来创建新文件夹。要将文件复制到所需文件夹，请用鼠标将相关文件拖放至目标文件夹。



要向 Button 应用背景图像，需要用到图像文件。现在，我们将复制图像文件，以将其用作背景图像。在源项目的根文件夹下，创建一个新文件夹，然后将该文件夹的名称指定为“edje”。然后在 /edje 文件夹下创建新文件件，并将文件夹名称指定为“images”。

转至附录的 /image 文件夹，将 green.png 和 red.png 两个文件复制到新建的 /edje/images 文件夹中。



现在，我们必须添加源代码。在 create_base_gui() 函数末尾添加新代码。此代码会将 EDJE 文件注册为主题，并创建第四个 Button 小部件。

```
        /* Button-3 */
        btn = elm_button_add(ad->conform);
        elm_object_style_set(btn, "icon_expand_add");
        my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

        /* Theme */
        char edj_path[PATH_MAX] = "";
        app_get_resource("edje/custom_button.edj", edj_path, (int)PATH_MAX);
        elm_theme_extension_add(NULL, edj_path);

        /* Button-4 */
        btn = elm_button_add(ad->win);
        elm_object_style_set(btn, "customized");
        elm_object_text_set(btn, "Custom style");
        evas_object_smart_callback_add(btn, "clicked", btn_custom_cb, ad);
        my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
    }
```

app_get_resource() 是一种函数，用于查找 /res 文件夹的绝对路径、添加子文件夹路径并返回该路径。对于第一个参数，请输入子文件夹路径。在此例中，已经发送了“edje/custom_button.edj”，因此，完整路径为 ~/res/edje/custom_button.edj。此函数尚未创建。我们稍后会创建此函数。

elm_theme_extension_add() 是一种用于注册主题信息文件的函数。将 EDJE 文件的路径提供给此函数的第二个参数。

elm_object_style_set() 是一种用于将自定义主题应用至 Button 小部件的函数。我们已将自定义主题的名称指定为“customized”。该主题已在 custom_button.edc 文件中定义。

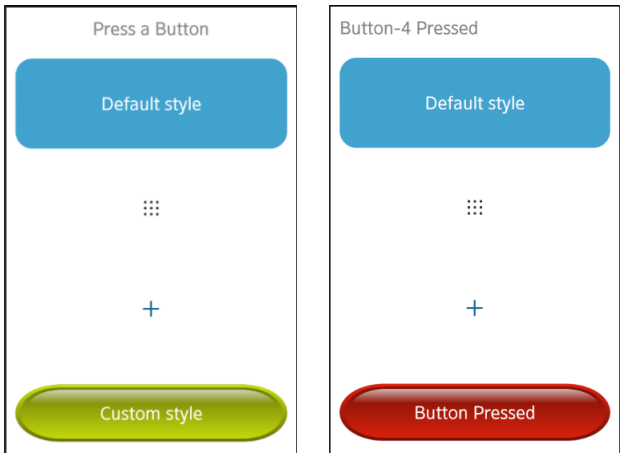
在 create_base_gui() 函数之上添加两个新函数。第一个函数是用于第五个 Button 的回调事件函数，第二个函数是负责返回 /res 文件夹绝对路径的函数。

```
static void
btn_custom_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s* ad = data;
        elm_object_text_set(obj, "Button Pressed");
        elm_object_text_set(ad->label, "Button-5 Pressed");
```

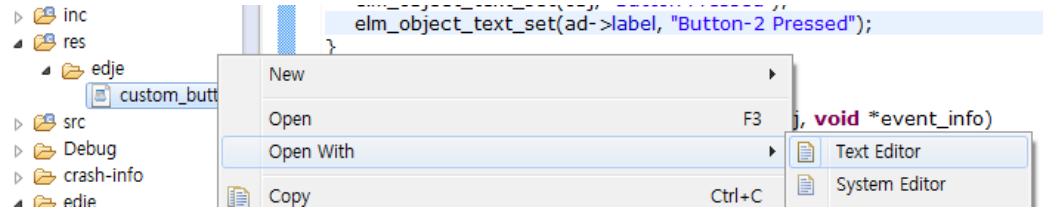
```
}

static void
app_get_resource(const char *res_file_in, char *res_path_out, int res_path_max)
{
        char *res_path = app_get_resource_path();
        if (res_path) {
                snprintf(res_path_out, res_path_max, "%s%s", res_path, res_file_
in);
                free(res_path);
        }
}
```

第二个函数“app_get_resource()”已调用“app_get_resource_path()”。此函数会查找 /res 文件夹的绝对路径，并返回该路径。

再次运行示例，随后便会看到应用绿色图像的第五个 Button 小部件已添加进来。点击 Button 可将其变为红色图像。取消点击操作则可将图像变回绿色，并改变标题文本。



有关 EDJE 文件要解释的元素还有很多，我们现在只解释了最重要的一些元素。在编辑器中打开 EDJE 文件。右键单击 /res/edje/custom_button.edc 文件，在快捷菜单中选择 [Open With > Text Editor]。



您可在文件顶部看到定义语句。在该语句中，ICON_NORMAL 指示了正常状态的背景图像文件名。如需将背景图像文件名更改为“btn_n.png”，请修改定义语句，如下所示：

```
#define ICON_NORMAL btn_n.png
```

和您猜到的一样，该语句下面的 ICON_PRESSED 指示了 Button 背景图像在按下状态的文件名。

```
#define ICON_NORMAL green.png
#define ICON_PRESSED red.png
#define BUTTON_MIN_WIDTH 142
#define BUTTON_MIN_HEIGHT 56
#define BUTTON_PADDING_LEFT_RIGHT 8
#define BUTTON_ICON_HEIGHT 46
#define BUTTON_ICON_WIDTH 46
#define BUTTON_TEXT_SIZE 30
```

再往下看，您将看到以下代码：

```
collections {
      base_scale: 1.8;
      group { name: "elm/button/base/customized";
            script {
                  public mouse_down = 0;
                  public multi_down = 0;
            }
```

对于“group”下的名称属性，您可指定主题名称。指定自定义主题名称，并将以下代码插入源代码。

```
elm_object_style_set(btn, "customized");
```

6）相关 API

Evas_Object *elm_button_add(Evas_Object *parent)：是一种用于创建 Button 小部件的 API。

void evas_object_smart_callback_add(Evas_Object *obj, const char *event, Evas_Smart_Cb func, const void *data)：是一种用于指定小部件或容器的回调事件函数的 API。/ 参数：发生事件的对象、事件类型（“clicked”指示了点击事件）、回调函数名称，以及将要发送至回调函数的数据。

Eina_Bool elm_object_style_set(Evas_Object *obj, const char *style)：一种用于指定小部件样式的 API。/ 参数：将要应用某个样式的小部件。指定样式类型。“icon_reorder”可指定重新排序图标样式；“icon_expand_add”可指定加号图标样式；“icon_expand_delete”可指定减号图标样式；“customized”可指定自定义样式。

void elm_theme_extension_add(Elm_Theme *th, const char *item)：一种用于注册主题信息文件的 API。/ 参数：主题路径和 EDJE 文件。

char *app_get_resource_path(void)：一种用于查找 /res 文件夹的绝对路径并返回该路径的 API。

8. 使用 Bg 小部件创建背景

有两种方式可用来显示小部件的背景色或背景图像。一种是使用 EDJE 来应用主题，另一种是使用 Bg 小部件。通过使用 BG 小部件，您可以轻松显示背景。

1) 创建颜色 Bg 小部件

创建新的源项目，并将项目名称指定为“BgEx”。

创建源项目之后，打开 src 文件夹中的 bgex.c 文件，并在 create_base_gui() 函数之上添加新函数。

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int x, int y, int w, int
h)
{
   evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
   evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
   elm_table_pack(table, child, x, y, w, h);
   evas_object_show(child);
}
```

my_table_pack() 是一种用于将小部件或容器添加到 Table 容器中的函数。Table 是一种容器，用于将屏幕划分为多个单元格，并将小部件放入所需单元格。通过使用 Table 小部件，您可使用多种显示器分辨率。

以下是按数序列出的 my_table_pack() 函数参数：
- Table 容器
- 子窗口
- 水平单元格号码
- 垂直单元格号码
- 水平单元格数量
- 垂直单元格数量

现在，我们将转至 create_base_gui() 函数并创建 Bg 小部件。“Conformant”和“Label”将被删除，因为他们不是本例所需的必要内容。

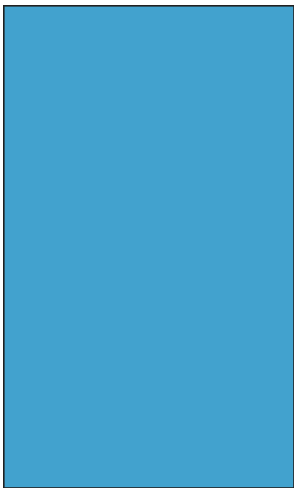
```
    /* Conformant */
    /*ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);*/

    /* Label*/
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "Hello EFL");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
    elm_object_content_set(ad->conform, ad->label);
    evas_object_show(ad->label);*/

    /* Table */
    Evas_Object *table = elm_table_add(ad->win);
    /* Make table homogenous - every cell will be the same size */
    elm_table_homogeneous_set(table, EINA_TRUE);
    /* Let the table child allocation area expand within in the box */
    evas_object_size_hint_weight_set(table, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_win_resize_object_add(ad->win, table);
    evas_object_show(table);

    {
        /* Bg-1 Color */
        Evas_Object *bg = elm_bg_add(ad->win);
        my_table_pack(table, bg, 0, 0, 2, 2);
        elm_bg_color_set(bg, 66, 162, 206);
        evas_object_show(bg);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

Elm_table_add() 是一种用于创建 Table 容器的 API。

elm_bg_add() 是一种用于创建 Bg 小部件的 API。

elm_bg_color_set() 是一种用于指定背景色和 Bg 小部件的 API。对于第一个参数，请输入将要应用属性的 Bg 小部件。对于第二到第四个参数，请输入颜色值。按照红绿蓝的顺序依次输入颜色值，值范围为：0 - 255。

运行示例，随后将看到屏幕编程蓝色。换言之，即已在整个屏幕范围内创建了 Bg 小部件。

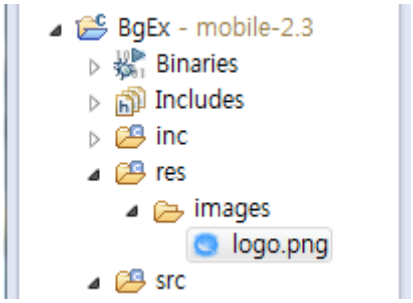


2) 向 Bg 小部件应用图像：原始大小

在本节中，我们将创建一个已应用背景图像的 Bg 小部件。要执行此操作，必须用到图像文件。

现在，我们将复制图像文件，以将其用作背景图像。在源项目的 /res 文件夹下创建一个新文件夹，然后将该文件夹的名称指定为“images”。右键单击 /res 文件夹，然后在快捷菜单中选择 [New > Folder]。弹出窗口出现后，请在文件夹名称字段中输入“images”，然后单击“Finish”按钮。

随后转至附录的 /Image 文件夹，将 logo.png 文件复制到新建的 /res/images 文件夹。要将文件复制到所需文件夹，请用鼠标将相关文件拖放至目标文件夹。



现在，我们将创建新的 Bg 小部件并指定背景图像。在 create_base_gui() 函数末尾添加新代码。

```
{
    /* Bg-1 Color */
    Evas_Object *bg = elm_bg_add(ad->win);
    my_table_pack(table, bg, 0, 0, 2, 2);
    elm_bg_color_set(bg, 66, 162, 206);
    evas_object_show(bg);

    /* Image path */
    char buf[PATH_MAX];
    app_get_resource("images/logo.png", buf, (int)PATH_MAX);

    /* Bg-2 Image Center */
    bg = elm_bg_add(ad->win);
    elm_bg_option_set(bg, ELM_BG_OPTION_CENTER);
    elm_bg_file_set(bg, buf, NULL);
    my_table_pack(table, bg, 2, 0, 2, 2);
}
```

app_get_resource() 是一种函数，用于查找 /res 文件夹的绝对路径、添加子文件夹路径，然后返回该路径。对于第一个参数，请输入子文件夹路径。在此例中，已经发送了“images/logo.png”，因此，完整路径为 ~/res/images/logo.png。此函数尚未创建。我们稍后会创建此函数。

elm_bg_option_set() 用于指定图像的显示样式。如果为第二个参数输入 ELM_BG_OPTION_CENTER，则会按图像原始大小在 Bg 小部件中心位置显示该图像。

elm_bg_file_set() 是一种用于指定 Bg 小部件的图像文件的函数。对于第一个参数，请指定将要应用属性的 Bg 小部件。对于第二个参数，请提供文件路径。

现在，我们将创建可返回 /res 文件夹绝对路径的函数。在 create_base_gui() 函数之上添加新代码。

```
static void
app_get_resource(const char *res_file_in, char *res_path_out, int res_path_max)
{
        char *res_path = app_get_resource_path();
        if (res_path) {
                snprintf(res_path_out, res_path_max, "%s%s", res_path, res_file_
in);
                free(res_path);
        }
}
```

由于该源代码与 ButtonEx 示例中使用的源代码相同，此处将不再详细介绍。

再次运行示例，随后便会看到有一个图像显示在屏幕右侧。右边的部分就是为第二个 Bg 指定的区域，图像则是按其原始大小居中显示的。

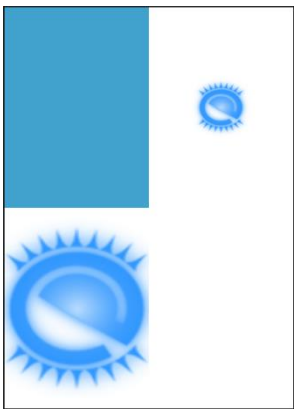
3）向 Bg 小部件应用图像：维持原有比例来调整大小

在本节，我们将学习如何在维持图像原有比例的同时让图像填充某个区域。在 create_base_gui() 函数末尾添加新代码。

```
        /* Bg-2 Image Center */
        bg = elm_bg_add(ad->win);
        elm_bg_option_set(bg, ELM_BG_OPTION_CENTER);
        elm_bg_file_set(bg, buf, NULL);
        my_table_pack(table, bg, 2, 0, 2, 2);

        /* Bg-3 Image Scale */
        bg = elm_bg_add(ad->win);
        elm_bg_option_set(bg, ELM_BG_OPTION_SCALE);
        elm_bg_file_set(bg, buf, NULL);
        my_table_pack(table, bg, 0, 2, 2, 2);
    }
```

如果为 elm_bg_option_set() 函数的第二个参数输入“ELM_BG_OPTION_SCALE”，即可在维持图像原有长宽比的同时让图像填充 Bg 区域。

再次运行示例，随后将看到一个大图像显示在屏幕左下角。



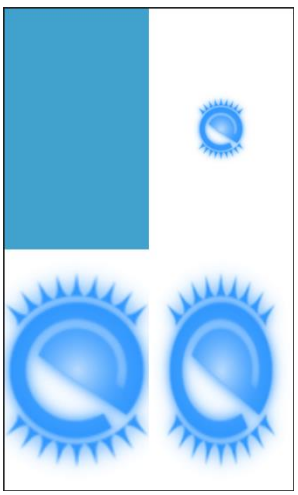
4）向 Bg 小部件应用图像：调整大小但不维持原有比例

在本节，我们将学习如何忽略图像原有比例的情况下让图像填充指定区域。在 create_base_gui() 函数末尾添加新代码。此代码可创建第四个 Bg 小部件。

```
        /* Bg-3 Image Scale */
        bg = elm_bg_add(ad->win);
        elm_bg_option_set(bg, ELM_BG_OPTION_SCALE);
        elm_bg_file_set(bg, buf, NULL);
        my_table_pack(table, bg, 0, 2, 2, 2);

        /* Bg-4 Image Stretch */
        bg = elm_bg_add(ad->win);
        elm_bg_option_set(bg, ELM_BG_OPTION_STRETCH);
        elm_bg_file_set(bg, buf, NULL);
        my_table_pack(table, bg, 2, 2, 2, 2);
    }
```

如果为 elm_bg_option_set() 函数的第二个参数输入“ELM_BG_OPTION_STRETCH”，即可让图像填充 Bg 区域。

再次运行示例，随后将看到一个大图像显示在屏幕右下角。您可以看到所示图像的高宽比与其原始长宽比并不相同。



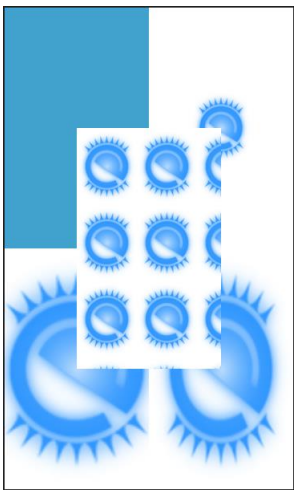
5）向 Bg 小部件应用图像：平铺样式

在本节，我们将学习如何采用平铺模式重复显示某个图像。在 create_base_gui() 函数末尾添加新代码。此代码可创建第五个 Bg 小部件。

```
        /* Bg-4 Image Stretch */
        bg = elm_bg_add(ad->win);
        elm_bg_option_set(bg, ELM_BG_OPTION_STRETCH);
        elm_bg_file_set(bg, buf, NULL);
        my_table_pack(table, bg, 2, 2, 2, 2);

        /* Bg-5 Image Tile */
        bg = elm_bg_add(ad->win);
        elm_bg_option_set(bg, ELM_BG_OPTION_TILE);
        elm_bg_file_set(bg, buf, NULL);
        my_table_pack(table, bg, 1, 1, 2, 2);
    }
```

如果为 elm_bg_option_set() 函数的第二个参数输入“ELM_BG_OPTION_TILE”，即可重复显示某个图像。

再次运行示例，随后将看到屏幕中央重复显示一个图像。



6）相关 API

Evas_Object *elm_bg_add(Evas_Object *parent)：一种用于创建 Bg 小部件的函数。

void elm_bg_color_set(Evas_Object *obj, int r, int g, int b)：一种用于指定 Bg 小部件的背景色的函数。/ 参数：将要应用属性的 Bg 小部件，对于第二到第四个参数，则为颜色值。按照红绿蓝的顺序依次输入颜色值，值范围为：0 - 255。

void elm_bg_option_set(Evas_Object *obj, Elm_Bg_Option option)：一种用于指定图像在 Bg 小部件中的显示样式的函数。/ 参数：第一个参数是 Bg 小部件，第二个参数是图像放置样式。样式类型如下：

- ELM_BG_OPTION_CENTER：按图像的原始大小在 Bg 区域的中心位置显示该图像。
- ELM_BG_OPTION_SCALE：在维持图像原始长宽比的同时让图像填充 Bg 区域。
- ELM_BG_OPTION_STRETCH：显示时让图像填充 Bg 区域。
- ELM_BG_OPTION_TILE：重复显示图像。

Eina_Bool elm_bg_file_set(Evas_Object *obj, const char *file, const char *group)：一种用于指定 Bg 小部件的图像文件的函数。/ 参数：Bg 小部件对象和文件路径。

9. 使用 Conformant 容器重新调整屏幕的大小

若要在屏幕的顶端（状态栏）显示一个指示器，您需要使用 Conformant 容器。即使已经存在一个 Conformant 容器，您也可以隐藏一个指示器。如果在显示一个新的面板（如键盘）时需要重新调整屏幕的大小，也将需要一个 Conformant 容器。我们将通过一个示例来学习如何使用 Conformant 容器。

1) 隐藏指示器

创建一个新的源项目，并将项目名称指定为“ConformantEx”。

创建源项目之后，打开 src 文件夹中的 bgex.c 文件，并在 create_base_gui() 函数之上添加一个新函数。之前在 HelloWorld 示例中使用了该函数。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
```

```
   /* show the frame */
   evas_object_show(frame);
}
```

添加一个 Button 小部件，以实施一个隐藏指示器的功能。在 create_base_gui() 函数的结尾添加新代码。此代码将创建一个 Box 容器，并添加一个 Button 小部件。

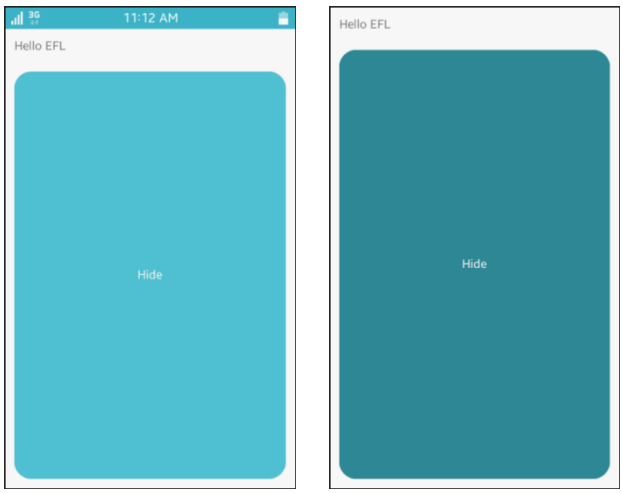
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "Hello EFL");
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);

            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Hide");
            evas_object_smart_callback_add(btn, "clicked", btn_hide_cb, ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

我们将实施一项功能以在点击新添加的 Button 时隐藏指示器。在 create_base_gui() 函数之上添加一个 Button 回调函数。

```
static void
btn_hide_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad   = (appdata_s*)data;
        elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_HIDE);
}
```

elm_win_indicator_mode_set() 是一个更改指示器模式的 API。将 Window 对象传递给第一个参数。一个应用程序只能有一个窗口。然后，将模式类型传递给第二个参数。传递 ELM_WIN_INDICATOR_HIDE 则会隐藏指示器。

构建并运行该示例，现在您将会在屏幕的顶端看到一个指示器。单击 Hide 按钮，然后该指示器将会消失。



当有指示器和无指示器时，Button 小部件的大小会有点区别。因为我们已指定 Label 小部件的高度作为最小值，所以当指示器不再存在时，Button 将会按指示器的高度进行垂直拉伸。

2) 显示指示器

现在我们将实施一项功能以显示之前已消失的指示器。在 create_base_gui() 函数的结尾添加新代码。此代码可创建第二个 Button。

```
            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Hide");
            evas_object_smart_callback_add(btn, "clicked", btn_hide_cb, ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            /* Button-2 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Show");
            evas_object_smart_callback_add(btn, "clicked", btn_show_cb, ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

然后，在 create_base_gui() 函数之上为第二个 Button 添加一个回调函数。

```
static void
btn_show_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad   = (appdata_s*)data;
        elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
}
```

再次运行该示例，然后单击 Hide 按钮。指示器消失。

现在，单击 Show 按钮。指示器再次出现。

3）创建 Entry 小部件

Entry 是一个编辑器小部件，用于接收用户输入的文本。由于文本是使用键盘输入的，因此需要更改屏幕的大小。为此，您需要在 Conformant 容器之上添加一个不同的容器（如 Box 或 Layout 容器），然后在新添加的容器上方添加一个 Entry。

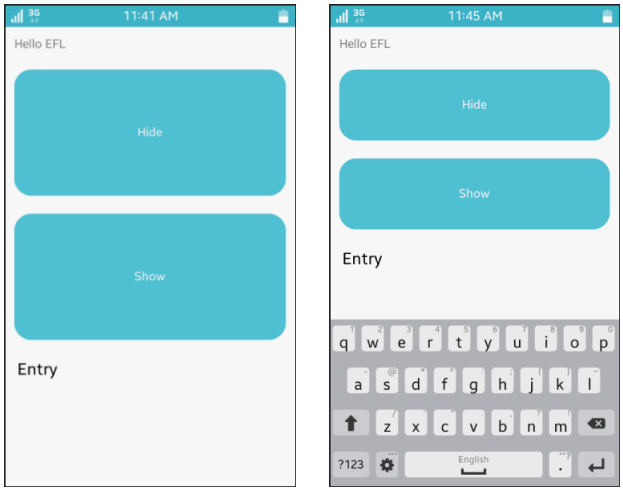
转至 create_base_gui() 函数，并在该函数的结尾添加新代码。此代码将创建一个 Entry 小部件。

```
        /* Button-2 */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Show");
        evas_object_smart_callback_add(btn, "clicked", btn_show_cb, ad);
        my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

        /* Entry */
        Evas_Object *entry = elm_entry_add(ad->conform);
        elm_object_text_set(entry, "Entry");
        my_box_pack(box, entry, 1.0, 1.0, -1.0, -1.0);
    }
```

elm_entry_add() 是一种用于创建 Entry 小部件的 API。我们已在 Conformant 之上创建了一个 Entry。

编译和运行该示例。在屏幕的顶端您将会看到一个 Label 小部件，在其下方则会看到文本“Entry”，这就是我们创建的 Entry 小部件。Entry 的背景颜色与屏幕背景的颜色相同，因此它们之间的边界不可见。稍后我们将会学习如何指定背景颜色。

点击文本“Entry”。将会显示一个键盘，而屏幕将会沿垂直方向缩短。现在，我们可以清楚地看到“Entry”。



4) 相关 API

Evas_Object *elm_entry_add(Evas_Object *parent)：一种用于创建 Entry 小部件的 API。

void elm_win_indicator_mode_set (Evas_Object *obj, Elm_Win_Indicator_Mode mode)：一种用于更改指示器模式的 API。/ 参数：Window 对象和模式类型。传递 ELM_WIN_INDICATOR_HIDE 则会隐藏指示器。

10. 使用 Entry 小部件

若要接收用户输入的文本字符串，您可以使用 Entry 小部件。现在我们将学习如何创建 Entry 小部件以及请求用户的输入内容。

1）创建 Entry 小部件

创建一个新的源项目，并将项目名称指定为“EntryEx”。

创建源项目之后，打开 src 文件夹中的 entryex.c 文件，并在 create_base_gui() 函数之上添加一个新函数。该函数将向一个 Table 容器添加一个小部件。

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int x, int y, int w, int
h)
{
    evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
    evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_table_pack(table, child, x, y, w, h);
    evas_object_show(child);
}
```

然后，转至 create_base_gui() 函数，并创建一个 Box 容器、一个 Table 容器和一个 Entry 小部件。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);
```

```
    {
        /* Box to put the table in so we can bottom-align the table
         * window will stretch all resize object content to win size */
        Evas_Object *box = elm_box_add(ad->conform);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        /* Table */
        Evas_Object *table = elm_table_add(ad->conform);
        /* Make table homogenous - every cell will be the same size */
        elm_table_homogeneous_set(table, EINA_TRUE);
        /* Set padding of 10 pixels multiplied by scale factor of UI */
        elm_table_padding_set(table, 20 * elm_config_scale_get(), 20 * elm_conf
ig_scale_get());
        /* Let the table child allocation area expand within in the box */
        evas_object_size_hint_weight_set(table, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
        /* Set table to fiill width but align to bottom of box */
        evas_object_size_hint_align_set(table, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_box_pack_end(box, table);
        evas_object_show(table);

        {
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
            my_table_pack(table, ad->label, 0, 0, 4, 1);

            /* Entry-1 */
            Evas_Object *entry = elm_entry_add(ad->conform);
            elm_entry_single_line_set(entry, EINA_TRUE);
            elm_entry_entry_insert(entry, "Entry-1");
            my_table_pack(table, entry, 0, 2, 4, 1);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
└──────────────────────────────────────────┘
```

Table 是一个可根据屏幕高宽比放置小部件的容器。我们之所以需要使用一个 Table 容器，是因为若要使用 Bg 作为一个 Entry 的背景，必须将两者

放置在同一空间中。我们使用了一个 Box 来指定小部件之间的间距。

elm_table_padding_set() 是一种用于指定边距的 API。

elm_entry_add() 是一种用于创建 Entry 小部件的 API。

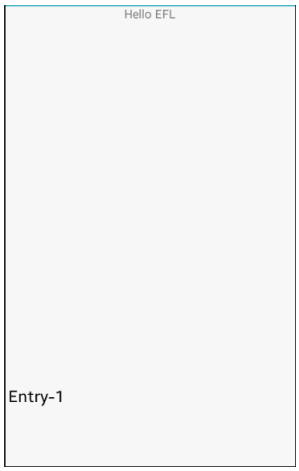
elm_entry_single_line_set() 是一种用于设置/取消设置多行输入的 API。若将 EINA_TRUE 传递给第二个参数，则设置为仅单行输入，若传递 EINA_FALSE，则设置为多行输入。

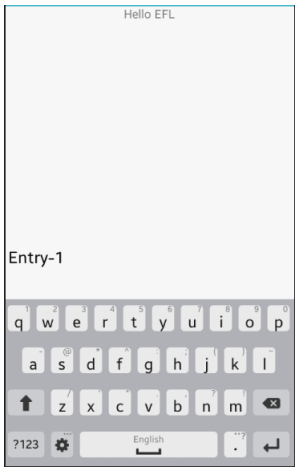
elm_entry_entry_insert() 是一种用于向 Entry 小部件添加标题文本的 API。换言之，它可以在现有标题文本的后面添加新文本。

构建并运行该源项目。您在屏幕底部看到的文本“Entry-1”就是我们创建的 Entry 小部件。

如果您单击该 Entry 小部件，将会显示一个键盘，以便您输入新文本。

您可以使用您的键盘在模拟器中输入文本。





2) 显示 Guide 文本

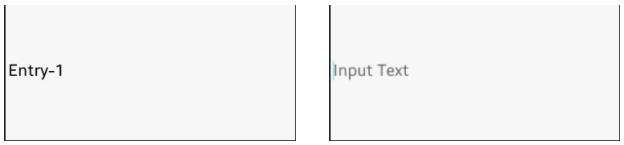
Guide 文本是解释编辑器作用的文本。如果输入字段是空字段，则会显示 Guide 文本，而且一旦您使用键盘输入文本，该 Guide 文本就会消失。智能手机的屏幕都较小，因此需要有效地利用空间。由于 Guide 文本可取代 Label 小部件，因此使用 Guide 文本有助于节省空间。

向 create_base_gui() 函数的 Entry-creating 代码添加新代码。

```
        /* Entry-1 */
        Evas_Object *entry = elm_entry_add(ad->conform);
        elm_entry_single_line_set(entry, EINA_TRUE);
        elm_entry_entry_insert(entry, "Entry-1");
        elm_object_part_text_set(entry, "elm.guide", "Input Text");
        my_table_pack(table, entry, 0, 2, 4, 1);
```

将“elm.guide”传递给 elm_object_part_text_set() 函数的第二个参数，就可以实现为一个 Entry 小部件指定 Guide 文本。将 Guide 文本的内容传递给第三个参数。

再次运行该示例，并删除 Entry 小部件的内容。将会显示文本“Input Text”。一旦您向此 Entry 小部件中输入文本，该 Guide 文本就会消失。



3) 使用 Bg 小部件创建背景

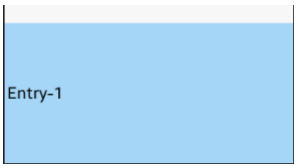
由于 Entry 小部件的背景颜色是白色，而屏幕背景的颜色也是白色，因此 Entry 小部件与屏幕之间的边界模糊不清。有两种方法可在 Entry 小部件中显示背景颜色。一种方法是使用 Bg 小部件；另一种方法是使用 EDJE。

现在我们将学习如何使用 Bg 小部件作为背景。在 create_base_gui() 函数的结尾添加新代码。此代码将创建一个 Bg 小部件，并为该 Bg 小部件指定与一个 Entry 小部件相同的坐标。请注意，必须先创建一个 Bg 小部件，然后再创建一个 Entry 小部件。

```
        /* Label*/
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
        my_table_pack(table, ad->label, 0, 0, 4, 1);

        /* Bg-1 */
        Evas_Object *bg = elm_bg_add(ad->conform);
        elm_bg_color_set(bg, 170, 220, 255);
        my_table_pack(table, bg, 0, 2, 4, 1);

        /* Entry-1 */
        Evas_Object *entry = elm_entry_add(ad->conform);
```

我们在创建 Entry 小部件之前先创建了一个 Bg 小部件，并且为这两个小部件指定了相同的坐标。这样做将使 Bg 小部件看起来就像是 Entry 小部件的背景。

再次运行该示例，现在您将会看到该 Entry 小部件的背景。



4) 导入在 Entry 小部件中输入的文本

我们现在将学习如何在一个 Button 被点击时请求在 Entry 小部件中输入的文本。若要在事件函数中使用一个 Entry 小部件，必须将其声明为一个全局变量或 AppData。

在源代码的顶端，您将会看到定义了一个 appdata_s 结构。这是一个存储应用程序中所用数据的结构。默认情况下声明了 Window、Conformant 和 Label。这里我们将添加一个 Entry。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *entry;
} appdata_s;
```

然后，返回至 create_base_gui() 函数，并在该函数的结尾添加新代码。此代码将创建一个 Button 小部件，并将一个 Entry 小部件存储到一个 AppData 结构中。

```
        /* Label*/
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
        my_table_pack(table, ad->label, 0, 0, 4, 1);

        /* Button-1 */
        Evas_Object *btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Get Text");
        evas_object_smart_callback_add(btn, "clicked", btn_clicked_cb, ad);
        my_table_pack(table, btn, 0, 1, 4, 1);

        /* Bg-1 */
        Evas_Object *bg = elm_bg_add(ad->conform);
        elm_bg_color_set(bg, 170, 220, 255);
        my_table_pack(table, bg, 0, 2, 4, 1);

        /* Entry-1 */
```

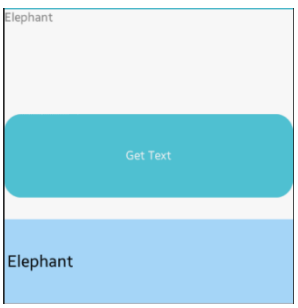
```
        Evas_Object *entry = elm_entry_add(ad->conform);
        elm_entry_single_line_set(entry, EINA_TRUE);
        elm_entry_entry_insert(entry, "Entry-1");
        elm_object_part_text_set(entry, "elm.guide", "Input Text");
        my_table_pack(table, entry, 0, 2, 4, 1);
        ad->entry = entry;
```

我们现在将创建一个 Button 回调函数。在 create_base_gui() 函数之上添加一个新函数。此代码将请求一个 Entry 小部件的文本，并将其显示在一个 Label 小部件中。

```
static void
btn_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s* ad = data;
        char* text = elm_entry_entry_get(ad->entry);
        elm_object_text_set(ad->label, text);
}
```

elm_entry_entry_get() 是一个请求 Entry 小部件的文本的函数。可以将该函数看作是 elm_entry_entry_insert() 的逆函数。

再次运行该示例，更改 Entry 小部件的文本，然后单击该 Button。在 Entry 中输入的文本将会显示在 Label 中。



5) 专用于输入密码的 Entry 小部件

当您输入密码时，必须将其文本显示为通配符，以免附近的人员看到密码。我们现在将创建一个专用于输入密码的 Entry 小部件。在 create_base_gui() 函数的结尾添加新代码。此代码将创建第二个 Bg 小部件和一个 Entry 小部件。

```
        /* Entry-1 */
        Evas_Object *entry = elm_entry_add(ad->conform);
        elm_entry_single_line_set(entry, EINA_TRUE);
        elm_entry_entry_insert(entry, "Entry-1");
        elm_object_part_text_set(entry, "elm.guide", "Input Text");
        my_table_pack(table, entry, 0, 2, 4, 1);
        ad->entry = entry;

        /* Bg-2 */
        bg = elm_bg_add(ad->conform);
        elm_bg_color_set(bg, 170, 220, 255);
        my_table_pack(table, bg, 0, 3, 4, 1);

        /* Entry-2 */
        entry = elm_entry_add(ad->conform);
        elm_entry_single_line_set(entry, EINA_TRUE);
        elm_entry_entry_insert(entry, "Entry-2");
        elm_entry_password_set(entry, EINA_TRUE);
        my_table_pack(table, entry, 0, 3, 4, 1);
    }
```

elm_entry_password_set() 是一个设置仅密码模式的 API。若将 EINA_TRUE 传递给第二个参数，将设置为仅密码模式，若传递 EINA_FALSE，则会关闭此模式。

再次运行该示例。当您在第二个 Entry 中输入文本时，该文本将显示为通配符，这样其他人就无法辨认出密码了。

6) 在 Entry 小部件中进行多行输入

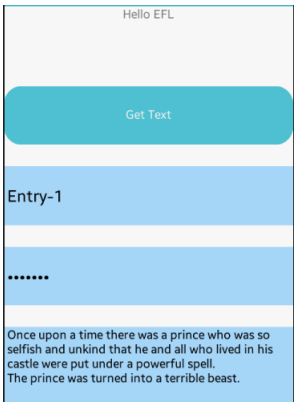
若要在 Entry 小部件中输入多行文本，您可以使用与 Label 小部件相同的方法。在 create_base_gui() 函数的结尾添加新代码。此代码将添加第三个 Bg 小部件和一个 Entry 小部件。

```
        /* Entry-2 */
        entry = elm_entry_add(ad->conform);
        elm_entry_single_line_set(entry, EINA_TRUE);
        elm_entry_entry_insert(entry, "Entry-2");
        elm_entry_password_set(entry, EINA_TRUE);
        my_table_pack(table, entry, 0, 3, 4, 1);

        /* Bg-3 */
        bg = elm_bg_add(ad->conform);
        elm_bg_color_set(bg, 170, 220, 255);
        my_table_pack(table, bg, 0, 4, 4, 2);

        /* Entry-3 */
        entry = elm_entry_add(ad->conform);
        elm_object_signal_emit(entry, "elm,state,scroll,enabled", "");
        elm_object_text_set(entry, "<font_size=30><align=left>Once upon a time
there was a prince who was so selfish and unkind that he and all who lived in h
is castle were put under a powerful spell.<br>The prince was turned into a terr
ible beast.</align></font_size>");
        my_table_pack(table, entry, 0, 4, 4, 2);
    }
```

elm_object_text_set() 是我们在更改 Label 和 Button 小部件的标题文本时所使用的一个 API。它也可以用于 Entry 小部件。但是，它必须与 elm_object_signal_emit() 函数一起使用。您需要将 HTML 标记传递给第二个参数。
<font_size=20> 是一种用于指定文本字体大小的标记。
<align=left> 是一种用于将水平对齐指定为左对齐的标记。

 是一个换行标记。

再次运行该示例。第三个 Entry 中显示了多行文本。请自己尝试用键盘输入文本。您可以看到字体大小发生了变化。使用 HTML 标记指定的属性只应用于输出，而不应用于输入。



7）相关 API

Evas_Object *elm_entry_add(Evas_Object *parent)：一种用于创建 Entry 小部件的 API。

void elm_entry_single_line_set(Evas_Object *obj, Eina_Bool single_line)：一种用于设置/取消设置多行的 API。/ 参数：Entry 对象以及是否以单行模式显示。若将 EINA_TRUE 传递给第二个参数，则设置为仅单行输入，若传递 EINA_FALSE，则设置为多行输入。

void elm_entry_entry_insert(Evas_Object *obj, const char *entry)：一种用于向 Entry 小部件添加标题文本的 API。换言之，它可以在现有标题文本的后面添加新文本。

void elm_object_part_text_set(Evas_Object *obj, const char *part, const char *text)：一种用于为 Entry 小部件指定 Guide 文本的 API。/ 参数：Entry 对象以及将应用此文本的区域。传递“elm.guide”则可以实现为 Entry 小部件指定 Guide 文本。将 Guide 文本的内容传递给第三个参数。

char *elm_entry_entry_get(Evas_Object *obj)：一种用于请求 Entry 小部件文本的 API。可以将该函数看作是 elm_entry_entry_insert() 的逆函数。

void elm_entry_password_set(Evas_Object *obj, Eina_Bool password)：一种用于设置仅密码模式的 API。若将 EINA_TRUE 传递给第二个参数，将设置为仅密码模式，若传递 EINA_FALSE，则会取消设置为该模式。

11. 使用 Check 小部件

若要实现在 On 和 Off 之间进行选择，您可以使用 Check 小部件。我们现在将学习如何创建一个 Check 小部件以及请求用户的事件。

1) 创建 Check 小部件

创建一个新的源项目，并将项目名称指定为“CheckEx”。

创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并向 appdata 结构添加新变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *check1;
        Evas_Object *check2;
        Evas_Object *check3;
        Evas_Object *check4;
} appdata_s;
```

我们添加了四个 Check 小部件变量。现在我们将使用该源代码创建一个 Check 小部件。在 create_base_gui() 函数之上添加一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
```

```
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

然后，转至 create_base_gui() 函数并添加新代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* Label*/
```

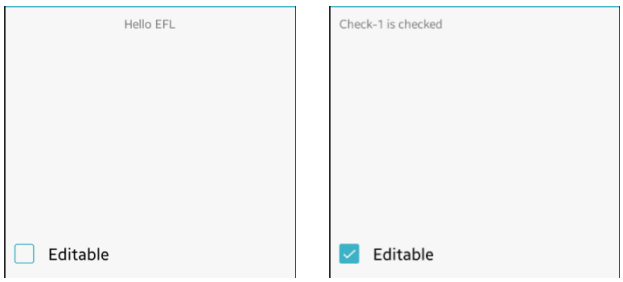
```
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
        my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);

        /* check 1 */
        ad->check1 = elm_check_add(ad->conform);
        elm_object_style_set(ad->check1,"popup");
        elm_object_text_set(ad->check1, "Editable");
        my_box_pack(box, ad->check1, 1.0, 1.0, -1.0, -1.0);
    }
}

/* Show window after base gui is set up */
evas_object_show(ad->win);
```

我们创建了一个 Box 小部件和一个 Check 小部件，并且还将这个 Check 小部件和一个 Label 小部件添加到了此 Box 小部件中。

elm_check_add(Evas_Object *parent) 是一种用于创建 Check 小部件的 API。

elm_object_text_set(obj, text) 是一种用于指定小部件标题文本的 API。

现在我们将构建并运行该示例。屏幕上将会显示一个 Check 小部件。点击正方形区域则会显示一个勾选标记。再次点击该区域则会使该勾选标记消失。



2) 更改 Check 小部件的符号

我们现在将要更改一个 Check 小部件的符号。为此，您需要更改其样式。向 create_base_gui() 函数添加新代码。

```
        /* check 1 */
        ad->check1 = elm_check_add(ad->conform);
        elm_object_style_set(ad->check1,"popup");
        elm_object_text_set(ad->check1, "Editable");
        my_box_pack(box, ad->check1, 1.0, 1.0, -1.0, -1.0);

        /* check 2 */
        ad->check2 = elm_check_add(ad->conform);
        elm_object_style_set(ad->check2, "favorite");
        elm_object_text_set(ad->check2, "Favorite");
        elm_check_state_set(ad->check2, EINA_TRUE);
        my_box_pack(box, ad->check2, 1.0, 1.0, -1.0, -1.0);

        /* check 3 */
        ad->check3 = elm_check_add(ad->conform);
        elm_object_style_set(ad->check3, "on&off");
        elm_object_text_set(ad->check3, "On / Off");
        elm_check_state_set(ad->check3, EINA_FALSE);
        my_box_pack(box, ad->check3, 1.0, 1.0, -1.0, -1.0);
    }
```

第二个和第三个 Check 小部件已创建好。

elm_object_style_set(Evas_Object *obj, const char *style) 是一种用于指定对象样式的 API。Check 小部件的样式类型如下：

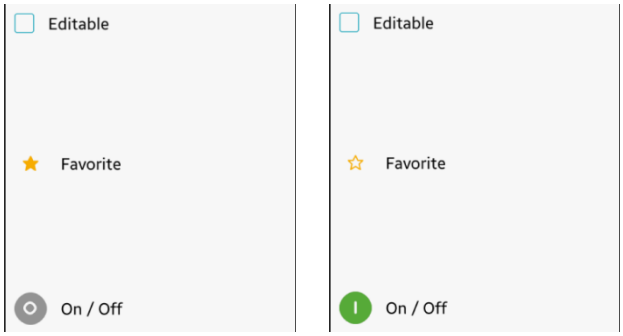
- favorite：星型符号
- on&off：开/关机符号

我们已将第二个 Check 小部件的样式指定为“favorite”。将会显示一个星型符号。

我们已将第三个 Check 小部件的样式指定为“on&off”。将会显示一个开/关机符号。

elm_check_state_set(Elm_Check *obj, Eina_Bool state) 是一种用于指定 Check 小部件 On/Off 状态的 API。若将 EINA_TRUE 传递给第二个参数，会将 Check 小部件的状态更改为 On，若传递 EINA_FALSE，则会将其状态更改为 Off。

再次运行该示例。第二个和第三个 Check 小部件已创建好，而且其符号现在已改变。



3) 为 Check 小部件请求一个 On/Off 事件

我们现在将要调用一个在用户点击一个 Check 小部件时发生的事件。为此，您需要为该 Check 小部件指定一个事件回调函数。向 create_base_gui() 函数添加新代码。

```
/* check 1 */
ad->check1 = elm_check_add(ad->conform);
elm_object_style_set(ad->check1,"popup");
elm_object_text_set(ad->check1, "Editable");
evas_object_smart_callback_add(ad->check1, "changed", check_changed_cb, ad);
my_box_pack(box, ad->check1, 1.0, 1.0, -1.0, -1.0);

/* check 2 */
ad->check2 = elm_check_add(ad->conform);
elm_object_style_set(ad->check2, "favorite");
elm_object_text_set(ad->check2, "Favorite");
elm_check_state_set(ad->check2, EINA_TRUE);
evas_object_smart_callback_add(ad->check2, "changed", check_changed_cb, ad);
my_box_pack(box, ad->check2, 1.0, 1.0, -1.0, -1.0);
```

```
    /* check 3 */
    ad->check3 = elm_check_add(ad->conform);
    elm_object_style_set(ad->check3, "on&off");
    elm_object_text_set(ad->check3, "On / Off");
    elm_check_state_set(ad->check3, EINA_FALSE);
    evas_object_smart_callback_add(ad->check3, "changed", check_changed_cb, ad);
    my_box_pack(box, ad->check3, 1.0, 1.0, -1.0, -1.0);
```

evas_object_smart_callback_add(Evas_Object *obj, char *event, Evas_Smart_Cb func, void *data) 是一种用于为智能对象（如一个小部件或容器）指定回调函数的 API。若将“changed”传递给第二个参数，则会在该 Check 小部件的状态发生改变时调用一个回调函数。

我们现在将创建一个回调函数。在 create_base_gui() 函数之上添加一个新函数。

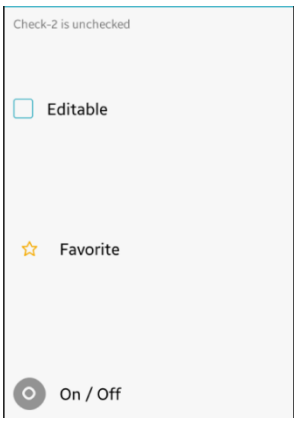
```
static void
check_changed_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        int check_num = 0;
        if( obj == ad->check1 )
                check_num = 1;
        else if( obj == ad->check2 )
                check_num = 2;
        else if( obj == ad->check3 )
                check_num = 3;
        else
                return;

        Eina_Bool state = elm_check_state_get(obj);
        char buf[64];

        sprintf(buf, "Check-%d is %s", check_num, state ? "checked" : "unchecked
");
        elm_object_text_set(ad->label, buf);
}
```

此代码将会计算出一个 Check 小部件已被用户点击的次数，调用该 Check 小部件的状态，然后在 Label 小部件中显示其状态。

Check 小部件状态更改事件函数的第一个参数将接收用户数据；第二个参数将接收发生事件的对象；而第三个参数将接收事件信息。

elm_check_state_get(const Elm_Check *obj) 是一种用于返回 Check 小部件 On/Off 状态的 API。该函数所起的作用与 elm_check_state_set() 函数的作用相反。

再次运行该示例。当您点击一个 Check 小部件时，该 Check 小部件的次数和状态将会显示在 Label 小部件中。



4) 启用/禁用 Check 小部件

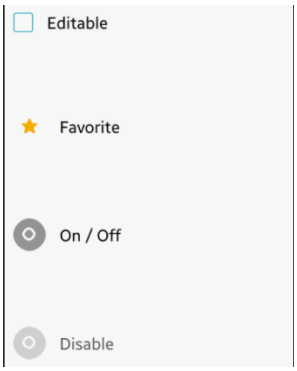
最后，我们将学习如何禁用 Check 小部件。向 create_base_gui() 函数添加用于创建第四个 Check 小部件的代码。

```
        /* check 3 */
        ad->check3 = elm_check_add(ad->conform);
        elm_object_style_set(ad->check3, "on&off");
        elm_object_text_set(ad->check3, "On / Off");
        elm_check_state_set(ad->check3, EINA_FALSE);
        evas_object_smart_callback_add(ad->check3, "changed", check_changed_cb,
ad);
        my_box_pack(box, ad->check3, 1.0, 1.0, -1.0, -1.0);

        /* check 4 */
        ad->check4 = elm_check_add(ad->conform);
        elm_object_style_set(ad->check4, "on&off");
```

```
        elm_object_text_set(ad->check4, "Disable");
        elm_object_disabled_set(ad->check4, EINA_TRUE);
        evas_object_smart_callback_add(ad->check4, "changed", check_changed_cb,
ad);
        my_box_pack(box, ad->check4, 1.0, 1.0, -1.0, -1.0);
    }
```

elm_object_disabled_set(Evas_Object *obj, Eina_Bool disabled) 是一种用于更改对象活动/非活动状态的 API。若将 EINA_TRUE 传递给第二个参数，将会禁用该对象，若传递 EINA_FALSE，则会启用该对象。

再次运行该示例。添加了第四个 Check 小部件。由于该 Check 小部件已被禁用，因此点击该小部件不会产生任何变化。



5) 相关 API

Evas_Object* elm_check_add(Evas_Object *parent) 是一种用于创建 Check 小部件的 API。

Eina_Bool elm_object_style_set(Evas_Object *obj, const char *style) 是一种用于指定对象样式的 API。Check 小部件的样式类型如下：

- popup：勾选标记
- favorite：星型符号
- on&off：开/关机符号

void elm_object_text_set(obj, text) 是一种用于指定小部件标题文本的 API。

void elm_check_state_set(Elm_Check *obj, Eina_Bool state) 是一种用于指定 Check 小部件 On/Off 状态的 API。若将 EINA_TRUE 传递给第二个参数，会将 Check 小部件的状态更改为 On，若传递 EINA_FALSE，则会将其状态更改为 Off。

void evas_object_smart_callback_add(Evas_Object *obj, char *event, Evas_Smart_Cb func, void *data) 是一种用于为智能对象（如一个小部件或容器）指定回调函数的 API。若将“changed”传递给第二个参数，则会在该 Check 小部件的状态发生改变时调用一个回调函数。

Eina_Bool elm_check_state_get(const Elm_Check *obj) 是一种用于返回 Check 小部件 On/Off 状态的 API。该函数所起的作用与 elm_check_state_set() 函数的作用相反。

void elm_object_disabled_set(Evas_Object *obj, Eina_Bool disabled) 是一种用于更改对象活动/非活动状态的 API。若将 EINA_TRUE 传递给第二个参数，将会禁用该对象，若传递 EINA_FALSE，则会启用该对象。

12. 使用 Radio 小部件

若要实现从多个菜单中选择一个菜单，您可以使用 Radio 小部件。我们现在将学习如何创建一个 Radio 小部件以及请求用户的事件。

1) 创建 Radio 小部件

创建一个新的源项目，并将项目名称指定为“RadioEx”。

创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在 create_base_gui() 函数之上添加一个新的函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
```

```
   evas_object_show(frame);
}
```

然后，转至 create_base_gui() 函数并添加新代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "Select Radio");
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);

            Evas_Object *radio, *radio_group;

            /* radio 1-1 */
            radio = elm_radio_add(ad->conform);
            elm_object_text_set(radio, "Cat");
            elm_radio_state_value_set(radio, 1);
            radio_group = radio;
            my_box_pack(box, radio, 1.0, 1.0, -1.0, -1.0);

            /* radio 1-2 */
            radio = elm_radio_add(ad->conform);
            elm_object_text_set(radio, "Dog");
            elm_radio_state_value_set(radio, 2);
            elm_radio_group_add(radio, radio_group);
```

```
            my_box_pack(box, radio, 1.0, 1.0, -1.0, -1.0);

            /* radio 1-3 */
            radio = elm_radio_add(ad->conform);
            elm_object_text_set(radio, "Hamster");
            elm_radio_state_value_set(radio, 3);
            elm_radio_group_add(radio, radio_group);
            my_box_pack(box, radio, 1.0, 1.0, -1.0, -1.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

创建了三个 Radio 小部件。

elm_radio_add(Evas_Object *parent) 是一种用于创建 Radio 小部件的 API。

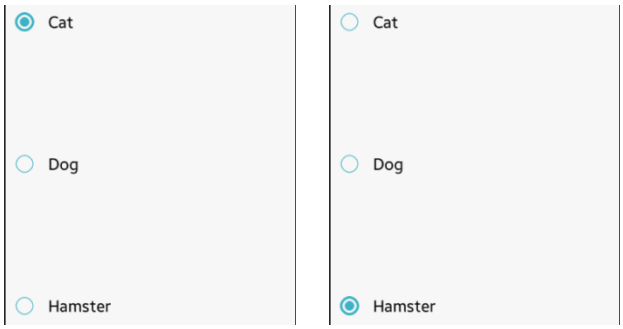
elm_radio_state_value_set(Elm_Radio *obj, int value) 是一种用于指定 Radio 小部件状态值的 API。在使用 Radio 小部件时，多个小部件将作为一组发挥作用。因此，必须为每个小部件指定一个单独的 ID 值。

radio_group 是 Radio 组变量。使用第一个 Radio 小部件作为一个 Radio 组。

elm_radio_group_add(Elm_Radio *obj, Evas_Object *group) 是一种用于向 Radio 组添加小部件的 API。将要添加的小部件传递给第一个参数。然后，将 Radio 组传递给第二个参数。

编译和运行该示例。屏幕上将会显示三个 Radio 小部件，如果您点击其中一个 Radio 小部件，将会显示一个勾选标记。

如果您选择另一个 Radio 小部件，勾选标记的位置将会随之移动。



2) 请求 Radio 小部件项目选择事件

在本小节中，我们将通过调用在用户点击 Radio 小部件时发生的事件，来查明哪个项目编号已被选中。为此，您需要为该 Radio 小部件指定一个事件回调函数。向 create_base_gui() 函数添加新代码。

```
        /* radio 1-1 */
        radio = elm_radio_add(ad->conform);
        elm_object_text_set(radio, "Cat");
        elm_radio_state_value_set(radio, 1);
        radio_group = radio;
        evas_object_smart_callback_add(radio, "changed", radio_animal_cb, ad);
        my_box_pack(box, radio, 1.0, 1.0, -1.0, -1.0);

        /* radio 1-2 */
        radio = elm_radio_add(ad->conform);
        elm_object_text_set(radio, "Dog");
        elm_radio_state_value_set(radio, 2);
        evas_object_smart_callback_add(radio, "changed", radio_animal_cb, ad);
        elm_radio_group_add(radio, radio_group);
        my_box_pack(box, radio, 1.0, 1.0, -1.0, -1.0);

        /* radio 1-3 */
        radio = elm_radio_add(ad->conform);
        elm_object_text_set(radio, "Hamster");
        elm_radio_state_value_set(radio, 3);
        evas_object_smart_callback_add(radio, "changed", radio_animal_cb, ad);
        elm_radio_group_add(radio, radio_group);
        my_box_pack(box, radio, 1.0, 1.0, -1.0, -1.0);
```

evas_object_smart_callback_add(Evas_Object *obj, char *event, Evas_Smart_Cb func, void *data) 是一种用于为智能对象（如一个小部件或容器）指定回调函数的 API。若将“changed”传递给第二个参数，则会在该 Radio 小部件的状态发生改变时调用一个回调函数。

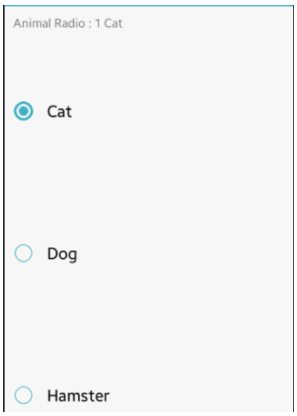
我们现在将创建一个回调函数。在 create_base_gui() 函数之上添加一个新函数。

```
static void
radio_animal_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        int value = 0;
        value = elm_radio_value_get(obj);
        char buf[64];
        sprintf(buf, "Animal Radio : %d", value);

        // 1st Radio Group
        switch( value ) {
        case 1 :
                sprintf(buf, "%s %s ", buf, "Cat");
                break;
        case 2 :
                sprintf(buf, "%s %s ", buf, "Dog");
                break;
        case 3 :
                sprintf(buf, "%s %s ", buf, "Hamster");
                break;
        }
        elm_object_text_set(ad->label, buf);
}
```

此代码将调用用户选中的 Radio 小部件的状态值，然后将此 Radio 小部件的状态值和动物名称显示在 Label 小部件中。

elm_radio_value_get(const Elm_Radio *obj) 是一种用于返回 Radio 小部件状态值的 API。它将返回当前选中的 Radio 小部件的状态值。

再次运行该示例。点击 Radio 小部件则会显示该 Radio 小部件的状态值和动物名称。



3) 第二个 Radio 组

在使用 Radio 小部件时，多个小部件将作为一组发挥作用。我们现在将学习添加三个新的 Radio 小部件并将它们分成两个 Radio 组。向 create_base_gui() 函数添加新代码。

```
        /* radio 1-3 */
        radio = elm_radio_add(ad->conform);
        elm_object_text_set(radio, "Hamster");
        elm_radio_state_value_set(radio, 3);
        evas_object_smart_callback_add(radio, "changed", radio_animal_cb, ad);
        elm_radio_group_add(radio, radio_group);
        my_box_pack(box, radio, 1.0, 1.0, -1.0, -1.0);

        /* radio 2-1 */
        radio = elm_radio_add(ad->conform);
        elm_object_text_set(radio, "Cookie");
        elm_radio_state_value_set(radio, 1);
        radio_group = radio;
        evas_object_smart_callback_add(radio, "changed", radio_dessert_cb, ad);
        my_box_pack(box, radio, 1.0, 1.0, -1.0, -1.0);

        /* radio 2-2 */
        radio = elm_radio_add(ad->conform);
        elm_object_text_set(radio, "Icecream");
        elm_radio_state_value_set(radio, 2);
        evas_object_smart_callback_add(radio, "changed", radio_dessert_cb, ad);
        elm_radio_group_add(radio, radio_group);
        my_box_pack(box, radio, 1.0, 1.0, -1.0, -1.0);
```

```
        /* radio 2-3 */
        radio = elm_radio_add(ad->conform);
        elm_object_text_set(radio, "Juice");
        elm_radio_state_value_set(radio, 3);
        evas_object_smart_callback_add(radio, "changed", radio_dessert_cb, ad);
        elm_radio_group_add(radio, radio_group);
        my_box_pack(box, radio, 1.0, 1.0, -1.0, -1.0);
    }
```

此代码将添加三个新的 Radio 小部件。我们已使用 elm_radio_state_value_set() 函数将这些小部件分别指定为 1、2 和 3。

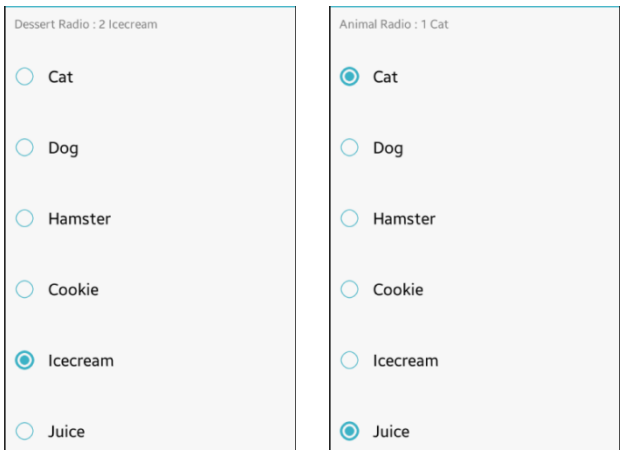
我们将第一个 Radio 小部件指定为一个 Radio 组，并使用 elm_radio_group_add() 函数向这个 Radio 组添加了第二个和第三个 Radio 小部件。

然后，我们将事件回调函数的名称指定为“radio_dessert_cb”。我们现在将创建一个回调函数。在 create_base_gui() 函数之上添加一个新函数。

```
static void
radio_dessert_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        int value = 0;
        value = elm_radio_value_get(obj);
        char buf[64];
        sprintf(buf, "Dessert Radio : %d", value);

        switch( value ) {
        case 1 :
                sprintf(buf, "%s %s ", buf, "Cookie");
                break;
        case 2 :
                sprintf(buf, "%s %s ", buf, "Icecream");
                break;
        case 3 :
                sprintf(buf, "%s %s ", buf, "Juice");
                break;
        }
        elm_object_text_set(ad->label, buf);
}
```

这个新函数的内容与 radio_animal_cb() 函数非常相似。

再次运行该示例，现在您将会看到总共显示了六个 Radio 小部件。点击 Radio 小部件将会显示第一组和第二组的勾选标记。



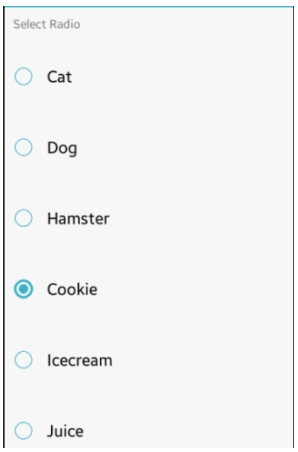
4) 使用源代码更改选定的 Radio 项目

我们现在将要实施一项功能以在执行该示例时自动选中第二组的第一个项目。在 create_base_gui() 函数的结尾添加新代码。

```
            /* radio 2-3 */
            radio = elm_radio_add(ad->conform);
            elm_object_text_set(radio, "Juice");
            elm_radio_state_value_set(radio, 3);
            evas_object_smart_callback_add(radio, "changed", radio_dessert_cb, a
d);
            elm_radio_group_add(radio, radio_group);
            my_box_pack(box, radio, 1.0, 1.0, -1.0, -1.0);

            /* Set selection to 2nd radio */
            elm_radio_value_set(radio_group, 1);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

elm_radio_value_set(Elm_Radio *obj, int value) 是一种用于为 Radio 组设置值的 API。与设定的值匹配的 Radio 小部件将被选中。

再次运行该示例。第二组的第一项被自动选中。



5) 相关 API

Evas_Object* elm_radio_add(Evas_Object *parent)：一种用于创建 Radio 小部件的 API。

void elm_radio_state_value_set(Elm_Radio *obj, int value)：一种用于指定 Radio 小部件状态值的 API。在使用 Radio 小部件时，多个小部件将作为一组发挥作用。因此，必须为每个小部件指定一个单独的 ID 值。

void elm_radio_group_add(Elm_Radio *obj, Evas_Object *group)：一种用于向 Radio 组添加小部件的 API。将要添加的小部件传递给第一个参数。然后，将 Radio 组传递给第二个参数。

void evas_object_smart_callback_add(Evas_Object *obj, char *event, Evas_Smart_Cb func, void *data)：一种用于为智能对象（如一个小部件或容器）指定回调函数的 API。若将“changed”传递给第二个参数，则会在该 Radio 小部件的状态发生改变时调用一个回调函数。

int elm_radio_value_get(const Elm_Radio *obj)：一种用于返回 Radio 小部件状态值的 API。它将返回当前选中的 Radio 小部件的状态值。

void elm_radio_value_set(Elm_Radio *obj, int value)：一种用于为 Radio 组设置值的 API。与设定的值匹配的 Radio 小部件将被选中。

13. 使用弹出窗口

若要向用户显示简短消息，您可以使用弹出窗口。您可以使弹出窗口在经过特定的一段时间之后关闭，还可以使弹出窗口接收用户的输入。

1）创建 Button 小部件

创建一个新的源项目，并将项目名称指定为“PopupEx”。

创建源项目之后，打开 src 文件夹中的 popupex.c 文件，并向 appdata_s 结构添加新变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *box;
        Evas_Object *popup;
        Evas_Object *entry;
        int popupNum;
} appdata_s;
```

“box”是一种用于按照顺序放置小部件的容器。

“popup”是 popup 小部件的句柄。它用于关闭弹出窗口或传输数据。

“entry”用于供用户在弹出窗口中输入文本。

“popupNum”保存当前弹出窗口的索引编号。

在 create_base_gui() 函数之上添加一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

然后，转至 create_base_gui() 函数并创建四个 Button 小部件。在该示例中，我们将创建四种不同类型的弹出窗口。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);
```

```
    {
        /* child object - indent to how relationship */
        /* A box to put things in vertically - default mode for box */
        ad->box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(ad->box, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
        elm_object_content_set(ad->conform, ad->box);
        evas_object_show(ad->box);

        {
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "Please click a button below");
            my_box_pack(ad->box, ad->label, 1.0, 0.0, 0.5, 0.0);

            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Popup Text");
            evas_object_smart_callback_add(btn, "clicked", make_popup_text, ad);
            my_box_pack(ad->box, btn, 1.0, 0.0, -1.0, -1.0);

            /* Button-2 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Popup 1 Button");
            evas_object_smart_callback_add(btn, "clicked", make_popup_text_1but
ton, ad);
            my_box_pack(ad->box, btn, 1.0, 0.0, -1.0, -1.0);

            /* Button-3 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Popup 3 Buttons");
            evas_object_smart_callback_add(btn, "clicked", make_popup_text_3but
ton, ad);
            my_box_pack(ad->box, btn, 1.0, 0.0, -1.0, -1.0);

            /* Button-4 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Popup Input Text");
            evas_object_smart_callback_add(btn, "clicked", make_popup_input_tex
t, ad);
            /* Note: this last button has weight 1 and align 0 so that the whol
e UI is
             * nicely and tightly packed at the top of the window.
             */
            my_box_pack(ad->box, btn, 1.0, 1.0, -1.0, 0.0);
        }
```

```
    }
    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

我们在一个 Conformant 之上创建了一个 Box，而且还在该 Box 之上添加了一个 Label 小部件和四个 Button 小部件。

为防止编译出错，必须为 Button 小部件创建回调函数。在 create_base_gui() 函数之上添加四个新函数。稍后我们将会定义这些函数的内容。

```
static void
make_popup_text(void *data, Evas_Object *obj, void *event_info)
{   }

static void
make_popup_text_1button(void *data, Evas_Object *obj, void *event_info)
{   }

static void
make_popup_text_3button(void *data, Evas_Object *obj, void *event_info)
{   }

static void
make_popup_input_text(void *data, Evas_Object *obj, void *event_info)
{   }
```

构建并运行该源项目，然后您可以看到一个 Label 小部件和几个 Button 小部件。我们将第一到第三个 Button 的区域高度指定为最小值。通过将 1.0 传递给 my_box_pack() 函数的第四个参数，将第四个 Button 的区域高度指定为最大值。此外，还通过将 0.0 传递给第六个参数，将每个 Button 的高度指定为最小值。这样做将会使小部件向顶端集中。

2) 创建文本弹出窗口

在本小节中，我们将创建最基本的弹出窗口，该窗口用于显示文本消息。向第一个 Button 的回调函数添加用于创建弹出窗口的代码。

```
static void
make_popup_text(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        ad->popup = elm_popup_add(ad->grid);
        elm_popup_align_set(ad->popup, ELM_NOTIFY_ALIGN_FILL, 1.0);
        evas_object_size_hint_weight_set(ad->popup, EVAS_HINT_EXPAND, EVAS_HINT
_EXPAND);
        elm_object_text_set(ad->popup, "Text popup - timeout of 3 sec is set.");

        evas_object_show(ad->popup);
        ad->popupNum = 1;
}
```

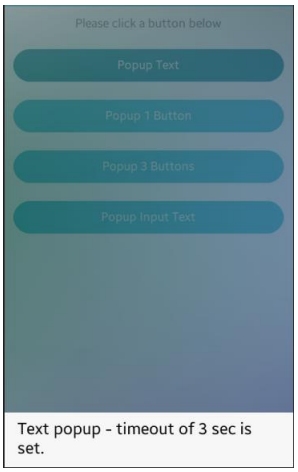
elm_popup_add() 是一种用于创建弹出窗口的 API。

elm_popup_align_set() 是一种用于指定弹出窗口位置的 API。为第二个参数输入该弹出窗口的水平位置。若输入 ELM_NOTIFY_ALIGN_FILL，则该弹出窗口将使用屏幕的整个水平区域。为第三个参数输入该弹出窗口的垂直位置。可接受的值范围为 0.0-1.0。

evas_object_size_hint_weight_set() 是一种用于指定小部件尺寸提示的 API。第二个参数指示宽度，而第三个参数指示高度。EVAS_HINT_EXPAND 是一种用于在指定区域内为小部件分配尽可能多空间的选项。

为 popupNum 指定“1”，以便记住它是第一个弹出窗口。

再次运行该示例，然后单击第一个 Button。在屏幕的底部将会出现一个弹出窗口，并显示一则文本消息。



3) 设置自动关闭计时器

您无法关闭弹出窗口，因为它们上面没有 Button。让我们来实施一项功能，以在经过特定的一段时间之后自动关闭一个弹出窗口。

向 make_popup_text() 函数添加新代码。

```
~
elm_object_text_set(ad->popup, "Text popup - timeout of 3 sec is set.");
elm_popup_timeout_set(ad->popup, 3.0);
evas_object_smart_callback_add(ad->popup, "timeout", popup_timeout, ad);
evas_object_show(ad->popup);
~
```

elm_popup_timeout_set() 是一种用于为弹出窗口设置计时器事件的 API。为第二个参数输入一个时间间隔。若输入 3.0，则会按 3 秒钟时间间隔发生计时器事件。

evas_object_smart_callback_add() 是一种用于指定接收事件的回调函数的 API。若为第二个参数输入“timeout”，则会在发生计时器事件时调用一个回调函数。对于第三个参数，请输入回调函数的名称。

我们现在将创建一个计时器事件函数。在 make_popup_text() 函数之上添加一个新函数。

这个新函数将删除一个弹出窗口，并在 Label 小部件中显示文本。

```
static void
popup_timeout(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        evas_object_del(obj);
        elm_object_text_set(ad->label, "Time out");
}
```

evas_object_del() 是一种用于删除对象的 API。在此例中，我们删除了计时器事件的主体。换言之，删除了一个弹出窗口。

再次运行该示例，然后单击第一个 Button。将会出现一个弹出窗口，不久之后该弹出窗口将会自动消失。

4) 当 Block 区域被点击时关闭弹出窗口

在本小节中，我们将实施一项功能，以在现有弹出窗口以外的区域被点击时关闭该弹出窗口。

在 make_popup_text() 函数的结尾添加新代码。

```
        evas_object_smart_callback_add(ad->popup, "timeout", popup_timeout, ad);
        evas_object_smart_callback_add(ad->popup, "block,clicked", popup_block_
clicked, ad);
        evas_object_show(ad->popup);
```

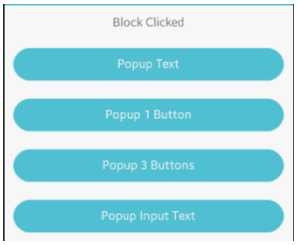
我们将该弹出窗口传递给 evas_object_smart_callback_add() 函数的第一个参数，并将“block,clicked”传递给第二个参数。这样做可以实现当弹出窗口以外的区域被点击时请求一个事件。

我们现在将创建一个请求该事件的函数。在 make_popup_text() 函数之上添加一个新函数。

这个新函数将删除一个弹出窗口，并在 Label 小部件中显示文本。

```
static void
popup_block_clicked(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        evas_object_del(obj);
        elm_object_text_set(ad->label, "Block Clicked");
}
```

再次运行该示例，单击第一个 Button，当出现一个弹出窗口时，单击该弹出窗口以外的区域。该弹出窗口将会消失，而且在 Label 小部件中将会显示文本“Block Clicked”。



5) 向弹出窗口添加一个 Button

在本小节中，我们将实施一项功能，以在第二个 Button 被点击时在显示的弹出窗口顶端创建一个 Button，并且在创建的这个 Button 被点击时关闭此弹出窗口。

向 make_popup_text_1button() 函数添加新代码。此代码将创建一个弹出窗口，并在该弹出窗口顶端添加一个 Button。

```
static void
make_popup_text_1button(void *data, Evas_Object *obj, void *event_info)
{
        Evas_Object *btn;
        appdata_s *ad = data;

        /* popup */
```

```
        ad->popup = elm_popup_add(ad->grid);
        elm_popup_align_set(ad->popup, ELM_NOTIFY_ALIGN_FILL, 1.0);
        evas_object_smart_callback_add(ad->popup, "block,clicked", popup_block_
clicked, ad);
        evas_object_size_hint_weight_set(ad->popup, EVAS_HINT_EXPAND, EVAS_HINT
_EXPAND);
        elm_object_text_set(ad->popup, "1Button popup");

        /* ok button */
        btn = elm_button_add(ad->popup);
        elm_object_text_set(btn, "OK");
        elm_object_part_content_set(ad->popup, "button1", btn);
        evas_object_smart_callback_add(btn, "clicked", popup_btn1_clicked, ad);

        evas_object_show(ad->popup);
        ad->popupNum = 2;
}
```

现在让我们实施一项功能，以便当弹出窗口顶端添加的 Button 被点击时在 Label 小部件中显示文本。在 make_popup_text_1button() 函数之上添加一个新函数。这个新函数是一个 OK 按钮事件函数。

```
static void
popup_btn1_clicked(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    const char *input;
    Eina_Strbuf *str;

    /* use eina_strbuf here for safe string allocation and formatting */
    input = elm_entry_entry_get(ad->entry);
    str = eina_strbuf_new();
    eina_strbuf_append_printf(str, "Input: '%s'", input);
    elm_object_text_set(ad->label, eina_strbuf_string_get(str));
    eina_strbuf_free(str);

    /* Destroy the popup AFTER reading from its child entry */
    evas_object_del(ad->popup);
    ad->popup = NULL;
}
```

eina_strbuf_new() 是一种用于创建 StrBuf 对象的 API。StrBuf 是一个实现轻松使用字符串的结构。

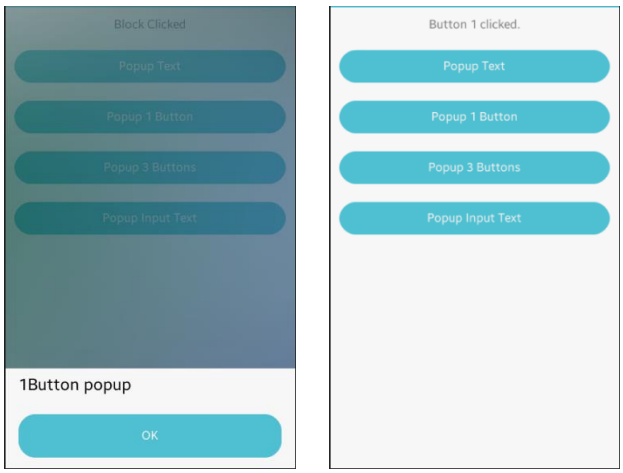
eina_strbuf_append_printf() 是一种用于向 StrBuf 添加新字符串的 API。

eina_strbuf_string_get() 是一种用于返回 StrBuf 中存储的字符串的 API。

eina_strbuf_free() 是一种用于删除 StrBuf 对象的 API。

再次运行该示例，然后单击第二个 Button。将会显示一个弹出窗口，您可以看到一则文本消息和一个 Button。

单击 OK 按钮将会使该弹出窗口消失，并且会更改 Label 小部件中的文本。



6) 向弹出窗口添加三个 Button

在本小节中，我们将向一个弹出窗口添加三个 Button。向 make_popup_text_3button() 函数添加新代码。此代码将创建一个弹出窗口，并在该弹出窗口顶端添加三个 Button。

复制和修改 make_popup_text_1button() 函数的内容，这样可节省一些键入操作。

```
static void
make_popup_text_3button(void *data, Evas_Object *obj, void *event_info)
```

```
{
        Evas_Object *btn;
        appdata_s *ad = data;

        /* popup */
        ad->popup = elm_popup_add(ad->grid);
        elm_popup_align_set(ad->popup, ELM_NOTIFY_ALIGN_FILL, 1.0);
        evas_object_smart_callback_add(ad->popup, "block,clicked", popup_block_
clicked, ad);
        evas_object_size_hint_weight_set(ad->popup, EVAS_HINT_EXPAND, EVAS_HINT
_EXPAND);
        elm_object_text_set(ad->popup, "3Button popup");

        /* ok button */
        btn = elm_button_add(ad->popup);
        elm_object_text_set(btn, "OK");
        elm_object_part_content_set(ad->popup, "button1", btn);
        evas_object_smart_callback_add(btn, "clicked", popup_btn1_clicked, ad);

        /* cancel button */
        btn = elm_button_add(ad->popup);
        elm_object_text_set(btn, "Cancel");
        elm_object_part_content_set(ad->popup, "button2", btn);
        evas_object_smart_callback_add(btn, "clicked", popup_btn2_clicked, ad);

        /* close button */
        btn = elm_button_add(ad->popup);
        elm_object_text_set(btn, "Close");
        elm_object_part_content_set(ad->popup, "button3", btn);
        evas_object_smart_callback_add(btn, "clicked", popup_btn3_clicked, ad);

        evas_object_show(ad->popup);
        ad->popupNum = 3;
}
```

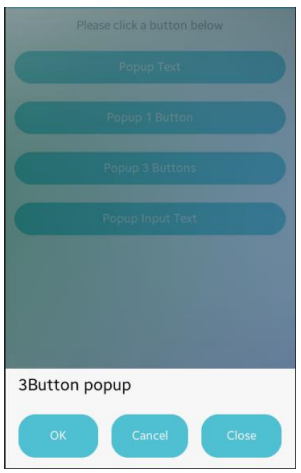
现在让我们实施一项功能，以便当弹出窗口顶端的一个 Button 被点击时在 Label 小部件中显示文本。在 make_popup_text_3button() 函数之上添加两个新函数。这些新函数是 Cancel 按钮和 Close 按钮的事件函数。

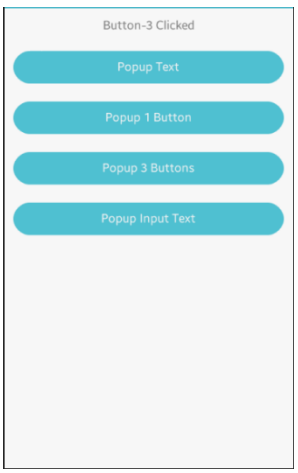
```
static void
popup_btn2_clicked(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
```

```
        evas_object_del(ad->popup);
        elm_object_text_set(ad->label, "Button-2 Clicked");
}

static void
popup_btn3_clicked(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        evas_object_del(ad->popup);
        elm_object_text_set(ad->label, "Button-3 Clicked");
}
```

再次运行该示例，然后点击第三个 Button。然后在弹出窗口的上面将会出现三个 Button。

点击其中任何一个 Button 都会使该弹出窗口消失，并更改 Label 中的文本。





7) 向弹出窗口添加 Entry 小部件

在本小节中，我们将实施一项功能，以向一个弹出窗口添加一个 entry 小部件，使用户能够在该弹出窗口中输入文本。向 make_popup_input_text() 函数添加新代码。此代码将创建一个弹出窗口，并在该弹出窗口顶端添加一个 Entry 和两个 Button。

复制和修改 make_popup_text_1button() 函数的内容，这样可节省一些键入操作。

```
static void
make_popup_input_text(void *data, Evas_Object *obj, void *event_info)
{
        Evas_Object *btn;
        appdata_s *ad = data;
        Evas_Object *entry;

        /* popup */
        ad->popup = elm_popup_add(ad->grid);
        elm_popup_align_set(ad->popup, ELM_NOTIFY_ALIGN_FILL, 1.0);
        evas_object_smart_callback_add(ad->popup, "block,clicked", popup_block_
clicked, ad);
        evas_object_size_hint_weight_set(ad->popup, EVAS_HINT_EXPAND, EVAS_HINT
_EXPAND);
        elm_object_part_text_set(ad->popup, "title,text", "Input Text");

        /* entry */
        entry = elm_entry_add(ad->popup);
        evas_object_size_hint_weight_set(entry, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
        evas_object_size_hint_align_set(entry, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_part_content_set(ad->popup, "elm.swallow.content" , entry);
        evas_object_show(entry);
        ad->entry = entry;

        /* OK button */
        btn = elm_button_add(ad->popup);
        elm_object_text_set(btn, "OK");
        elm_object_part_content_set(ad->popup, "button1", btn);
        evas_object_smart_callback_add(btn, "clicked", popup_btn1_clicked, ad);

        /* Cancel button */
        btn = elm_button_add(ad->popup);
```

```
        elm_object_text_set(btn, "Cancel");
        elm_object_part_content_set(ad->popup, "button2", btn);
        evas_object_smart_callback_add(btn, "clicked", popup_btn2_clicked, ad);

        evas_object_show(ad->popup);
        ad->popupNum = 4;
}
```

再次运行该示例，并单击第四个 Button，现在您将会看到已向该弹出窗口添加了一个 Entry。

现在让我们实施一项功能，以便当用户在一个 Entry 中输入文本并点击 OK 按钮时在 Label 中显示该文本。

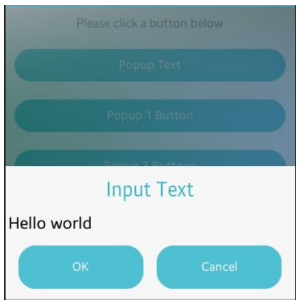
向 popup_btn1_clicked() 函数添加新代码。如果当前弹出窗口是第四个弹出窗口，此代码将会请求在该弹出窗口的 Entry 中输入标题文本，并将其显示在 Label 中。然后，此代码将会初始化 Entry 小部件。

```
static void
popup_btn1_clicked(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;

    if (ad->popupNum == 4) {
        const char *input;
```

```
        Eina_Strbuf *str;

        /* use eina_strbuf here for safe string allocation and formatting */
        input = elm_entry_entry_get(ad->entry);
        str = eina_strbuf_new();
        eina_strbuf_append_printf(str, "Input: '%s'", input);
        elm_object_text_set(ad->label, eina_strbuf_string_get(str));
        eina_strbuf_free(str);
    } else {
        elm_object_text_set(ad->label, "Button 1 clicked.");
    }

    /* Destroy the popup AFTER reading from its child entry */
    evas_object_del(ad->popup);
    ad->popup = NULL;

    /* Entry will be deleted when the popup is deleted (child widget) */
    ad->entry = NULL;
}
```

再次运行该示例，然后单击第四个 Button。当显示一个弹出窗口时，在 Entry 中输入文本，然后单击 OK 按钮。

该弹出窗口将会消失，而且这些文本将会显示在 Label 中。



8）相关 API

Evas_Object *elm_popup_add(Evas_Object *parent)：一种用于创建弹出窗口的 API。

void elm_popup_align_set(Evas_Object *obj, double horizontal, double vertical)：一种用于指定弹出窗口位置的 API。为第二个参数输入该弹出窗口的水平位置。若输入 ELM_NOTIFY_ALIGN_FILL，则该弹出窗口将使用屏幕的整个水平区域。为第三个参数输入该弹出窗口的垂直位置。可接受的值范围为 0.0-1.0。

void evas_object_size_hint_weight_set(Evas_Object *obj, double x, double y)：一种用于指定对象大致尺寸的 API。/ 参数：Window 对象、宽度提示和高度提示。EVAS_HINT_EXPAND 是一种用于尽可能扩大对象尺寸的选项。

void elm_popup_timeout_set(Evas_Object *obj, double timeout)：一种用于为弹出窗口指定计时器的 API。为第二个参数输入一个时间间隔。若输入 3.0，则会按 3 秒钟时间间隔发生计时器事件。

void evas_object_smart_callback_add(Evas_Object *obj, const char *event, Evas_Smart_Cb func, const void *data)：一种用于指定接收事件的回调函数的 API。若为第二个参数输入“timeout”，则会在发生计时器事件时调用一个回调函数。对于第三个参数，请输入回调函数的名称。

void evas_object_del(Evas_Object *obj)：一种用于删除对象的 API。

14. 使用 Slider 小部件

若要让用户输入值，您需要使用 Slider 小部件。在扫描音频或视频播放器中的播放列表时，该小部件也非常有用。

1) 创建 Slider 小部件

创建一个新的源项目，并将项目名称指定为“SliderEx”。

创建源项目之后，打开 src 文件夹中的 sliderex.c 文件，并在 create_base_gui() 函数之上添加一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
```

```
   evas_object_show(frame);
}
```

然后，转至 create_base_gui() 函数，并创建一个 Box 容器和一个 Slider 小部件。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "Please test the slider below");
            my_box_pack(box, ad->label, 1.0, 0.1, 0.5, 1.0);

            /* Slider-1 */
            Evas_Object *slider = elm_slider_add(ad->conform);
            elm_slider_min_max_set(slider, 0, 9);
            elm_slider_value_set(slider, 5);
            my_box_pack(box, slider, 1.0, 0.1, -1.0, 0.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

elm_slider_add() 是一种用于创建 Slider 小部件的 API。

elm_slider_min_max_set() 是一种用于指定 Slider 范围的 API。为第二个参数输入最小值。为第三个参数输入最大值。变量类型为“double”。

elm_slider_value_set() 是一种用于指定 Slider 的当前值的 API。

构建并运行该源项目。您可以看到一个 Slider 小部件。将轨迹条向左和向右滑动。

2）在 Slider 小部件中显示指示器

指示器是一项在拖动轨迹条时显示滑块当前值的功能。向用于创建 Slider 的代码添加新代码。

```
        /* Slider-1 */
        Evas_Object *slider = elm_slider_add(ad->conform);
        elm_slider_min_max_set(slider, 0, 9);
        elm_slider_value_set(slider, 5);
        elm_slider_indicator_show_set(slider, EINA_TRUE);
        elm_slider_indicator_format_set(slider, "%1.0f");
        my_box_pack(box, slider, 1.0, 0.1, -1.0, 0.0);
```

elm_slider_indicator_show_set() 是一种用于指定是否在 Slider 中显示指示器的 API。将 EINA_TRUE 传递给第二个参数，则会显示一个指示器。若传递 EINA_FALSE，则产生相反的结果。

elm_slider_indicator_format_set() 是一个为指示器中显示的文本指定格式的 API。

再次运行该示例，然后将轨迹条移动到各个位置。在轨迹条的上方将会显示数字。

3) Slider 跟踪事件

在本小节中，我们将实时请求一个在用户拖动轨迹条时发生的事件。当您部署一个视频播放器时，此功能将必不可少。

向用于创建 Slider 的代码添加新代码。

```
        /* Slider-1 */
        Evas_Object *slider = elm_slider_add(ad->conform);
        elm_slider_min_max_set(slider, 0, 9);
        elm_slider_value_set(slider, 5);
        elm_slider_indicator_show_set(slider, EINA_TRUE);
        elm_slider_indicator_format_set(slider, "%1.0f");
        evas_object_smart_callback_add(slider, "changed", slider_changed_cb, a
d);
        my_box_pack(box, slider, 1.0, 0.1, -1.0, 0.0);
```

若将“changed”传递给 evas_object_smart_callback_add() 函数的第二个参数，则可实现在滑块的值发生变化时请求一个事件。

现在，让我们定义一个事件函数。在 create_base_gui() 函数之上添加一个新函数。此代码将会请求 Slider 的当前值并将其显示在 Label 小部件中。

```
static void
slider_changed_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        char buf[64];

        double value = elm_slider_value_get(obj);
        sprintf(buf, "Slider : %d", (int)value);
        elm_object_text_set(ad->label, buf);
}
```

elm_slider_value_get() 是一个功能与 elm_slider_value_set() 相反的 API。它将返回 Slider 的当前值。变量类型为“double”。

再次运行该示例，然后将轨迹条移动到各个位置。该 Slider 的当前值将显示在 Label 小部件中。

4) 显示中心点

您可以在 Slider 的中央位置显示一个中心点。我们现在将向 create_base_gui() 函数添加用于创建一个新 Label 和 Slider 小部件的代码。

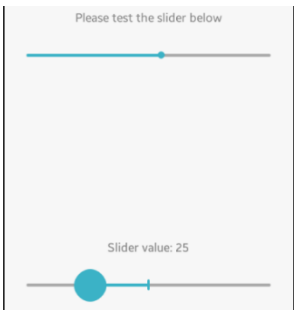
```
        /* Slider-1 */
        Evas_Object *slider = elm_slider_add(ad->conform);
        elm_slider_min_max_set(slider, 0, 9);
        elm_slider_value_set(slider, 5);
        elm_slider_indicator_show_set(slider, EINA_TRUE);
        elm_slider_indicator_format_set(slider, "%1.0f");
        evas_object_smart_callback_add(slider, "changed", slider_changed_cb, ad);
        my_box_pack(box, slider, 1.0, 0.1, -1.0, 0.0);

        /* Label-2 */
        ad->label2 = elm_label_add(ad->conform);
        elm_object_text_set(ad->label2, "Please test the slider below");
        my_box_pack(box, ad->label2, 1.0, 0.1, 0.5, 1.0);

        /* Slider-2 */
        slider = elm_slider_add(ad->conform);
        elm_slider_min_max_set(slider, 0, 99);
        elm_slider_value_set(slider, 30);
        elm_object_style_set(slider, "center_point");
        evas_object_smart_callback_add(slider, "changed", slider2_changed_cb, a
d);
        my_box_pack(box, slider, 1.0, 0.1, -1.0, 0.0);
    }
```

我们将该 Slider 小部件传递给 elm_object_style_set() 函数的第一个参数，并将“center_point”传递给第二个参数。这样做将会在 Slider 小部件的中央位置显示一个网格。

我们现在将为第二个 Slider 定义一个回调函数。在 create_base_gui() 函数之上添加一个新函数。

```
static void
slider2_changed_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    char buf[64];

    double value = elm_slider_value_get(obj);
    sprintf(buf, "Slider value: %d", (int)value);
    elm_object_text_set(ad->label2, buf);
}
```

再次运行该示例，您现在将会看到创建的第二个 Slider 小部件以及显示的中心点。拖动轨迹条，其值将会显示在第二个 Label 中。



5）相关 API

Evas_Object *elm_slider_add(Evas_Object *parent)：一种用于创建 Slider 小部件的 API。

void elm_slider_min_max_set(Evas_Object *obj, double min, double max)：一种用于指定 Slider 小部件范围的 API。参数：Slider 小部件对象、最小值和最大值。

void elm_slider_value_set(Evas_Object *obj, double val)：一种用于指定 Slider 小部件的当前值的 API。

void elm_slider_indicator_show_set(Evas_Object *obj, Eina_Bool show)：一种用于指定是否在 Slider 小部件中显示指示器的 API。将 EINA_TRUE 传递给第二个参数，则会显示一个指示器。若传递 EINA_FALSE，则产生相反的结果。

void elm_slider_indicator_format_set(Evas_Object *obj, const char *indicator)：一种用于指定指示器中显示的文本格式的 API。/ 参数：Slider 对象和文本格式。

double elm_slider_value_get(const Evas_Object *obj)：一种用于返回 Slider 的当前值的 API。变量类型为“double”。

Eina_Bool elm_object_style_set(Evas_Object *obj, const char *style)：一种用于指定对象样式的 API。将该 Slider 小部件传递给第一个参数，并将“center_point”传递给第二个参数，这样做将会在 Slider 小部件的中央位置显示一个网格。

15. 向 List 小部件添加文本项目。

若要在屏幕上显示一个包含多个文本项目的列表，您需要使用 List 小部件。List 小部件可上下滚动，还可以实现请求用户的选择事件。我们现在将通过一个示例来学习如何使用 List 小部件。

1) 创建一个 Text List 小部件

创建一个新的源项目，并将项目名称指定为“ListEx”。

创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在 create_base_gui() 函数之上添加一个新的函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
```

```
   /* show the frame */
   evas_object_show(frame);
}
```

然后，转至 create_base_gui() 函数，并创建一个 Box 容器和一个 List 小部件。接着，添加 10 个文本项目。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
            evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_
HINT_EXPAND);
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);

            /* List */
            const char *items[] = { "Seoul", "Tokyo", "Newyork", "Londeon", "Ba
ijing", "Kongga", "Moscuba", "Singgapol", "Pusan", "Hongkong" };
            Evas_Object *list = elm_list_add(ad->conform);

            for(int i=0; i < 10; i++)
                elm_list_item_append(list, items[i], NULL, NULL, NULL, (void*)
i);
            elm_list_go(list);
            my_box_pack(box, list, 1.0, 1.0, -1.0, -1.0);
```

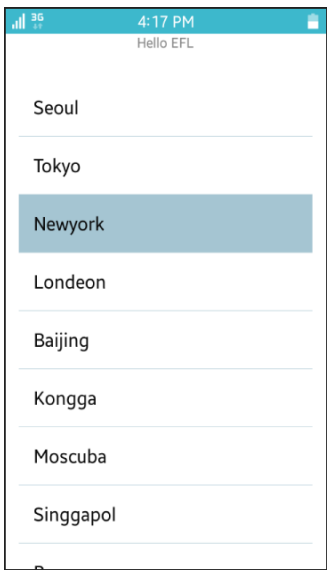
```
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

elm_list_add() 是一种用于添加新 List 小部件的 API。

elm_list_item_append() 是一种用于向 List 小部件添加项目的 API。第一个参数指明 List 小部件的对象，而第二个参数指明文本字符串。第三个参数指明左侧图标，而第四个参数指明右侧图标。以回调方式为第五个参数指定项目选择事件函数。然后，将用户数据传递给第六个参数。虽然通常都是传递“appdata”，但是为了标识每个项目的编号，我们传递了索引编号。

elm_list_go() 是一个通过刷新屏幕的操作在屏幕上反映出项目变化的 API。当添加了一个新项目时，或者当删除或修改了一个现有项目时，必须调用该函数，以便在屏幕上反映出变化。

运行该示例，您将会看到已创建了一个 List 小部件并显示了 10 个文本项目。您可以通过上下拖动鼠标来滚动浏览列表。如果您选中某一个项目，将会出现一个勾选标记。



2) 自动移除勾选标记

如果您选中某一个项目，将会出现一个勾选标记并一直保持显示。有时这项功能必不可少，但在其他时候，它是可有可无的。我们现在将学习如何禁用此功能。向 create_base_gui() 函数添加一行新代码。

```
            elm_list_go(list);
            evas_object_smart_callback_add(list, "selected", list_selected_cb,
NULL);
            my_box_pack(box, list, 1.0, 1.0, -1.0, -1.0);
        }
```

此代码指定了当用户选中 List 小部件项目时将会调用的一个回调函数。换言之，需要定义一个回调函数。

在 create_base_gui() 函数之上添加一个新函数。

```
static void
list_selected_cb(void *data, Evas_Object *obj, void *event_info)
{
        Elm_Object_Item *it = event_info;
        elm_list_item_selected_set(it, EINA_FALSE);
}
```

elm_list_item_selected_set() 是一种用于显示/移除 List 小部件项目勾选标记的 API。您需要将该项目对象传递给第一个参数。在此例中，将会传递“event_info”，它是传递自回调函数的第三个参数。若将 EINA_TRUE 传递给第二个参数，将会显示勾选标记，若传递 EINA_FALSE，则会移除勾选标记。

再次运行该示例，并选中一个 List 小部件项目。此时，将会显示一个勾选标记，然后该勾选标记将会消失。

3) 请求一个 List 小部件项目选择事件

在本小节中，我们将实施一项功能，以便当用户选中一个 List 小部件项目时会请求该项目的索引编号和文本，并将其显示在一个 Label 小部件中。

如下所示修改用于向一个 List 小部件添加项目的代码。

```
        for(int i=0; i < 10; i++)
                elm_list_item_append(list, items[i], NULL, NULL, list_item_clicked, (void*)i);
                //elm_list_item_append(list, items[i], NULL, NULL, NULL, (void*)i);
```

我们将项目选择事件回调函数指定为 list_item_clicked。我们现在将定义该函数。

在 create_base_gui() 函数之上添加一个新函数。

```
static void
list_item_clicked(void *data, Evas_Object *obj, void *event_info)
{
        int index = (int)data;
        Elm_Object_Item *it = event_info;
        const char *item_text = elm_object_item_text_get(it);

        char buf[PATH_MAX];
        sprintf(buf, "%d - %s", index, item_text);
        dlog_print(DLOG_INFO, "tag", "%s", buf);
}
```

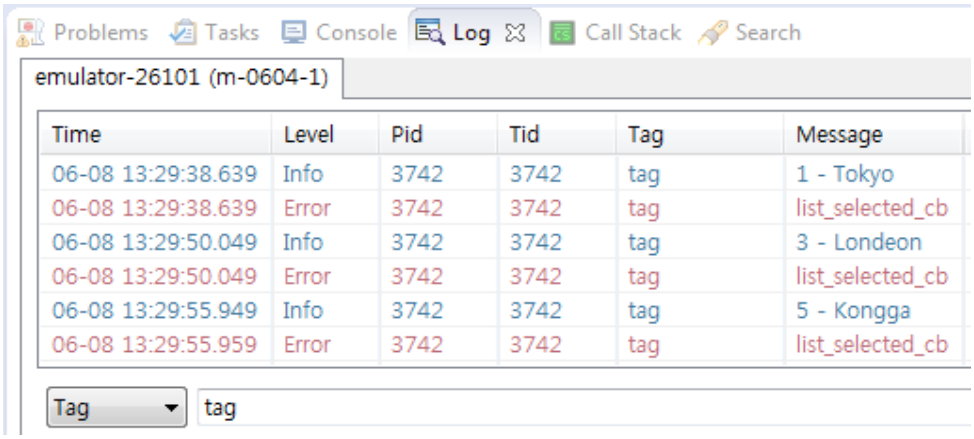
当用户在一个 List 小部件中选中某个项目时，将会调用此函数。第一个参数将接收该项目的索引编号。第二个参数将接收该 List 小部件的对象，而第三个参数将接收选定项目的对象。

elm_object_item_text_get() 是一种用于返回项目文本的 API。

紧随其后的代码用于将项目的索引编号和文本存储到字符串变量中，并将其显示在 Log 窗格内。

再次编译该示例，然后运行它。选中一个项目，现在您将会看到有关该项目的信息显示在 Log 窗格中。如果您在 Eclipse 底部未能看到 Log 窗格，请从菜单中选择 [Window > Show View > Other...]，然后从弹出窗口中选择 [Tizen > Log]。

若要查看 Log 消息，请从 Eclipse Log 窗格底部的组合框中选择 Tag，然后在右侧的 Edit 框中输入“tag”。



4) 在 Label 小部件中显示项目信息

我们已将选定项目的索引编号（而不是“appdata”）传递给项目选择事件函数。若要在 Label 小部件中显示项目的相关信息，您需要将 appdata 指定为全局变量。

在源文件的顶端添加一行新代码。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
} appdata_s;

appdata_s* m_ad = 0;
```

此代码将 appdata 声明为一个全局变量。

然后，如下所示在 create_base_gui() 函数的开头实施该全局变量。

```
static void
create_base_gui(appdata_s *ad)
{
        m_ad = ad;
```

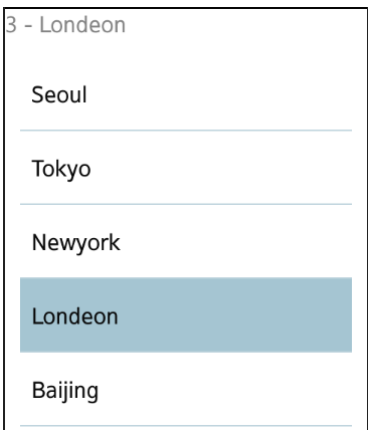
现在就可以在任何地方使用该 Label 小部件了。

返回至 list_item_clicked() 函数，并如下所示在该函数的结尾添加一行新代码。

```
        dlog_print(DLOG_INFO, "tag", "%s", buf);
        elm_object_text_set(m_ad->label, buf);
}
```

此代码将在 Label 小部件中显示有关所选 List 小部件项目的信息。

再次运行该示例，并选中以下 List 小部件项目。在 Label 中将会显示与选中的项目有关的信息。



5）相关 API

Evas_Object *elm_list_add(Evas_Object *parent)：一种用于添加 List 小部件的 API。/ 参数：父级对象。

Elm_Object_Item *elm_list_item_append(Evas_Object *obj, const char *label, Evas_Object *icon, Evas_Object *end, Evas_Smart_Cb func, const void *data)：一种用于向 List 小部件添加项目的 API。/ 参数：List 小部件、项目文本、左侧图标、右侧图标、项目选择事件函数的名称和用户数据。

void elm_list_go(Evas_Object *obj)：一种用于启动列表的 API。在屏幕上显示一个 List 小部件之前，必须先调用此函数。当添加了一个新项目时，或者当删除或修改了一个现有项目时，必须调用该函数，以便在屏幕上反映出变化。/ 参数：List 小部件。

void elm_list_item_selected_set(Elm_Object_Item *it, Eina_Bool selected)：一种用于显示/移除项目勾选标记的 API。/ 参数：List 小部件以及是否显示勾选标记。

const char *elm_object_item_text_get(const Elm_Object_Item *it)：一种用于返回项目文本的 API。/ 参数：项目的对象。

16. 在 GenList 小部件中显示图标

若要在 List 小部件中显示图标或双行文本，您可以使用 GenList 小部件。您还可以将列表分组。

在使用 GenList 小部件时，若要添加/删除项目以及添加图标等内容，都需要通过一个回调函数来处理，而且必须使用一个结构来存储项目数据。因此，GenList 小部件的使用比 List 小部件烦琐。让我们通过示例来详细了解如何使用 GenList 小部件。

1) 创建一个 GenList 小部件并显示文本

创建一个新的源项目，并将项目名称指定为“GenListEx”。

创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并定义新结构。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
} appdata_s;

typedef struct item_data
{
         int index;
         Elm_Object_Item *item;
} item_data_s;
```

item_data 是一种用于存储 GenList 小部件项目数据的结构。

在 create_base_gui() 函数之上添加一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
```

```
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

然后，转至 create_base_gui() 函数，并创建一个 Box 容器和一个 List 小部件。接着，添加 10 个文本项目。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
```

```
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);

            /* Genlist */
            Evas_Object *genlist = elm_genlist_add(ad->conform);
            my_box_pack(box, genlist, 1.0, 1.0, -1.0, -1.0);

            /* Create item class */
            Elm_Genlist_Item_Class *itc = elm_genlist_item_class_new();
            itc->item_style = "end_icon";
            itc->func.text_get = gl_text_get_cb;
            itc->func.del = gl_del_cb;

            /* Item add */
            for(int i=0; i < 10 ; i++)
            {
                item_data_s *id = calloc(sizeof(item_data_s), 1);
                id->index = i;
                id->item = elm_genlist_item_append(genlist, itc, id, NULL, ELM_
GENLIST_ITEM_NONE, NULL, id);
            }

            elm_genlist_item_class_free(itc);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

我们创建了一个 Box，然后创建了一个 GenList 并将其添加到该 Box 中。

elm_genlist_add() 是一种用于创建 GenList 小部件的 API。

elm_genlist_item_class_new() 是一种用于为 GenList 项目创建类的 API。

Elm_Genlist_Item_Class 是 GenList 项目的类。您将使用该类指定 GenList 样式和回调函数。样式类型如下：
- 将“end_icon”传递给 item_style 属性，则会在右端显示图标。
- 对于 func.text_get 属性，输入一种用于指定项目文本的回调函数。
- 对于 func.del 属性，输入一种用于删除项目的回调函数。

elm_genlist_item_append() 是一种用于向 GenList 小部件添加项目的 API。在调用此函数时，将会自动调用为 func.text_get 属性输入的回调函数。以下是按次序列出的参数：GenList 对象、项目类、父级项目、项目类型、项目选择事件回调函数的名称和用户数据。

elm_genlist_item_class_free() 是一种用于删除 GenList 项目的类的 API。

现在，我们定义一种用于指定 GenList 小部件项目的文本的回调函数。在 create_base_gui() 函数之上添加一个新函数。

```
static char*
gl_text_get_cb(void *data, Evas_Object *obj, const char *part)
{
        const char *items[] = { "Seoul", "Tokyo", "Newyork", "Londeon", "Baijing
", "Kongga", "Moscuba", "Singgapol", "Pusan", "Hongkong" };
        item_data_s *id = data;

        if (!strcmp(part, "elm.text")) {
                return strdup(items[id->index]);
        }

        return NULL;
}
```

此代码是一个回调函数，它将在一个 GenList 小部件项目中输入数据。用户数据将被传递给第一个参数。在此例中，传递的是项目数据结构。索引属性包含了项目的索引编号。第二个参数是项目的对象。传递给第三个参数的是该元素的类型。

一个 GenList 项目可拥有多个不同的元素。举例来说，如果这个值是“elm.text”，它表示该项目拥有文本元素。

strcmp (char *, char *) 是一种用于比较两个字符串的 API。如果结果证明两个字符串是相同的字符串，则返回值“0”。

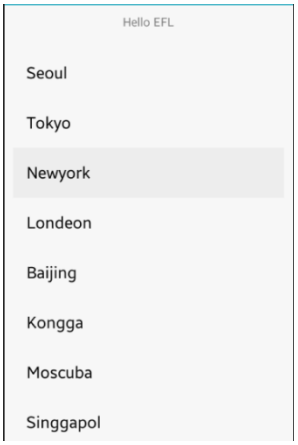
上述函数需要返回与指定的项目索引编号相匹配的文本，而使用 strdup() 函数的原因则是要创建一个新字符串。如果不这样做，将会终止该函数，同时将会使字符串数据消失。

现在，我们定义一个将会在删除项目时调用的回调函数。在 create_base_gui() 函数之上添加一个新函数。

```
static void
gl_del_cb(void *data, Evas_Object *obj)
{
        item_data_s *id = data;
        free(id);
}
```

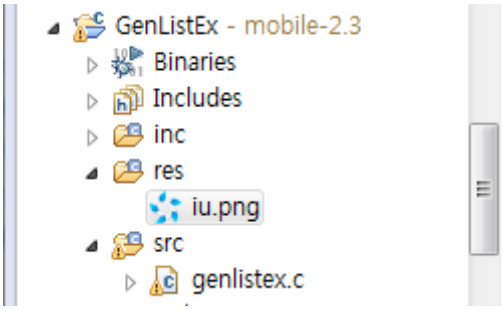
用户数据将被传递给第一个参数。这是一个项目数据结构，因而也是您需要删除的对象。

编译和运行该示例。创建了一个 GenList，并添加了 10 个文本项目。



2) 在 GenList 小部件中显示图标

在本小节中，我们将在一个项目的右侧显示一个图标图像。要执行此操作，必须用到图像文件。将附录的 /Image/iu.png 文件复制到源项目的 /res 文件夹中。



然后，创建一种用于向 GenList 项目添加图标图像的回调函数。如下所示，将以下三个函数添加到 create_base_gui() 函数中。

```
static void
app_get_resource(const char *res_file_in, char *res_path_out, int res_path_max)
{
        char *res_path = app_get_resource_path();
        if (res_path) {
                snprintf(res_path_out, res_path_max, "%s%s", res_path, res_file_
in);
                free(res_path);
        }
}

static Evas_Object*
create_image(Evas_Object *parent)
{
        char img_path[PATH_MAX] = { 0, };
        app_get_resource("iu.png", img_path, PATH_MAX);
        Evas_Object *img = elm_image_add(parent);
        elm_image_file_set(img, img_path, NULL);
        return img;
}
```

```
static Evas_Object*
gl_content_get_cb(void *data, Evas_Object *obj, const char *part)
{
        Evas_Object *content = create_image(obj);
        evas_object_size_hint_min_set(content, 50, 50);
        evas_object_size_hint_max_set(content, 50, 50);
        return content;
}
```

app_get_resource() 是一个向文件名称添加 res 文件夹路径并返回该文件的函数。

app_get_resource_path() 是一个返回 res 文件夹绝对路径的 API。

create_image() 是一种用于创建已应用图像文件的 Image 对象的函数。

elm_image_add() 是一种用于创建 Image 对象的 API。

elm_image_file_set() 是一个通过为 Image 对象指定图像文件路径来加载该图像的 API。

gl_content_get_cb() 是一种用于为 GenList 项目指定图标图像的回调函数。

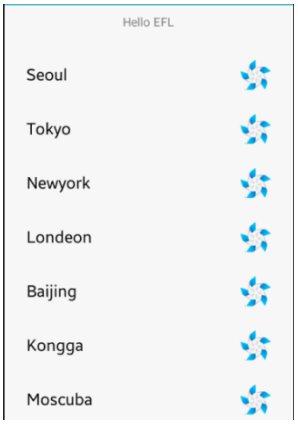
evas_object_size_hint_min_set() 是一种用于指定对象的最小尺寸提示的 API。在此例中，我们指定绝对值“50”作为提示。

evas_object_size_hint_max_set() 是一种用于指定对象的最大尺寸提示的 API。在此例中，我们指定绝对值“50”作为提示。

我们现在将您刚刚创建的图标指定回调函数分配给 GenList 项目。转至 create_base_gui() 函数并添加一行新代码。

```
        Elm_Genlist_Item_Class *itc = elm_genlist_item_class_new();
        itc->item_style = "end_icon";
        itc->func.text_get = gl_text_get_cb;
        itc->func.del = gl_del_cb;
        itc->func.content_get = gl_content_get_cb;
```

再次运行该示例，现在您将会看到在 GenList 项目的右侧显示了一个图标。



3) 请求选定项目的索引编号

在本小节中，我们将实施一项功能，以便当用户选中一个项目时将该项目的索引编号显示在一个 Label 小部件中。在源文件的顶端添加一个全局变量和一个函数。

```
typedef struct item_data
{
          int index;
          Elm_Object_Item *item;
} item_data_s;

appdata_s* m_ad = 0;

static void
list_item_clicked(void *data, Evas_Object *obj, void *event_info)
{
        item_data_s *id = data;
        char buf[PATH_MAX];
        sprintf(buf, "Item-%d", id->index);
        elm_object_text_set(m_ad->label, buf);
}
```

m_ad 是一种用于存储 appdata 的全局变量。

list_item_clicked() 是一种用于在 Label 小部件中显示指定项目的文本的项目选择事件回调函数。项目数据被传递给第一个参数。索引属性包含了项目的索引编号。我们要将该索引编号更改为一个字符串，并将该字符串显示在 Label 小部件中。

将 appdata 存储在全局变量中。在 create_base_gui() 函数的开头添加一行新代码。

```
create_base_gui(appdata_s *ad)
{
        m_ad = ad;
```

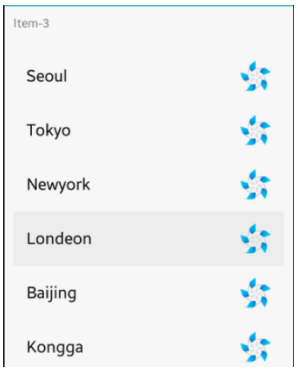
现在，我们只需定义一个回调函数。修改 create_base_gui() 函数的代码。

```
        for(int i=0; i < 10 ; i++)
        {
                item_data_s *id = calloc(sizeof(item_data_s), 1);
                id->index = i;
                id->item = elm_genlist_item_append(genlist, itc, id, NULL, ELM_
GENLIST_ITEM_NONE, list_item_clicked, id);
                //id->item = elm_genlist_item_append(genlist, itc, id, NULL, EL
M_GENLIST_ITEM_NONE, NULL, id);
        }
```

我们已将项目选择事件回调函数指定为“list_item_clicked”，并且

已将 GenList 单击事件回调函数指定为“gl_selected_cb”。

再次运行该示例。如果您选中一个项目，该项目的索引编号将显示在 Label 小部件中。



4）相关 API

Evas_Object *elm_genlist_add(Evas_Object *parent)：一种用于添加 GenList 小部件的 API。/ 参数：父级对象。

Elm_Genlist_Item_Class *elm_genlist_item_class_new(void)：一种用于创建项目类的 API。

Elm_Genlist_Item_Class：一个 GenList 项目的类。
    - item_style：项目的样式
- func.text_get：项目文本指定回调函数
- func.content_get：项目图标指定回调函数
- func.del：项目删除事件回调函数

Elm_Object_Item *elm_genlist_item_append(Evas_Object *obj, Elm_Genlist_Item_Class *itc, void *data, Elm_Object_Item *parent, Elm_Genlist_Item_Type type, Evas_Smart_Cb func, void *func_data)：一种用于向 GenList 添加项目的 API。在调用此函数时，将会自动调用为 func.text_get 属性输入的回调函数。/ 参数：GenList 对象、项目类、父级项目、项目类型（第五个参数）、项目选择事件回调函数的名称和用户数据。

void elm_genlist_item_class_free(Elm_Genlist_Item_Class *itc)：一种用于删除 GenList 项目类的 API。/ 参数：GenList 项目类的对象。

char *app_get_resource_path(void)：一个返回 res 文件夹绝对路径的 API。

sprintf(char*, char*, ...)：一种用于创建格式字符串的 API。

snprintf(char*, int, char*, ...)：一种用于创建指定长度的格式字符串的 API。

Evas_Object *elm_image_add(Evas_Object *parent)：一种用于创建 Image 对象的 API。/ 参数：父级对象。

Eina_Bool elm_image_file_set(Evas_Object *obj, char *file, char *group)：一种用于为 Image 对象指定图像文件的 API。/ 参数：Image 对象、图像文件的路径和图像组的名称（如果图像文件是一个 Edje 文件）。

void evas_object_size_hint_min_set(Evas_Object *obj, Evas_Coord w, Evas_Coord h)：一种用于指定对象的最小尺寸提示的 API。/ 参数：对象、宽度提示和高度提示。

void evas_object_size_hint_max_set(Evas_Object *obj, Evas_Coord w, Evas_Coord h)：一种用于指定对象的最大尺寸提示的 API。/ 参数：对象、宽度提示和高度提示。

strcmp(char*, char*)：一种用于比较两个字符串长度的 API。如果结果证明两个字符串的长度相同，则返回值“0”。

strdup(char*)：一种用于创建并返回另一个相同字符串的 API。

void elm_genlist_item_selected_set(Elm_Object_Item *it, Eina_Bool selected)：一种用于显示/移除项目勾选标记的 API。/ 参数：List 小部件以及是否显示勾选标记。

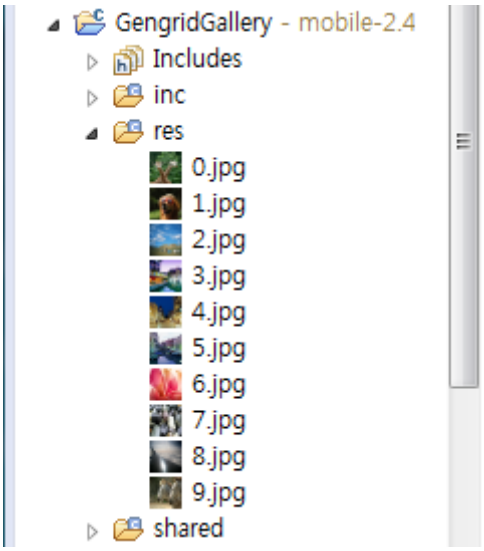
17. 创建复杂的 Gallery 小部件

我们现在将学习如何将两个或更多个小部件组合在一起，从而创建一个复杂的小部件。在该示例中，我们将使用一个 GenGrid 小部件和一个 Bg 小部件创建一个 Gallery 小部件。

1) 创建一个 GenGrid 小部件并添加一个图像

创建一个新的源项目，并将项目名称指定为“GengridGallery”。

在该示例中，我们将要创建一个显示图像的 Gallery 小部件。因此需要图像文件。从附录的 /Image 文件夹中复制 10 个图像文件（从 0.jpg 到 9.jpg）到源项目的 /res 文件夹中。



然后，打开 src 文件夹中的源文件 (~.c)，向 appdata 结构添加新变量，并定义新结构。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *gengrid;
```

```
        Evas_Object *bg;
} appdata_s;

typedef struct itemdata {
        int index;
        const char *path;
} itemdata_s;
```

我们向 AppData 结构添加了 GenGrid 和 Bg 变量。

itemdata 是一种用于存储 GenGrid 项目信息的结构。项目信息由索引编号和图像文件路径组成。

我们现在将创建一个 GenGrid 小部件并添加图像。完成此操作的方法与创建 GenList 小部件的方法相似。在 create_base_gui() 函数之上添加一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
```

```
   /* show the frame */
   evas_object_show(frame);
}
```

转至 create_base_gui() 函数并添加用于创建一个 Box 容器、一个 GenGrid 小部件和一个 Bg 小部件的新代码。该示例中将不使用 Label，因此删除它。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* Gengrid */
            ad->gengrid = create_gengrid(ad->conform);
            my_box_pack(box, ad->gengrid, 1.0, 1.0, -1.0, -1.0);

            /* Bg-1 Color */
            ad->bg = elm_bg_add(ad->conform);
            elm_bg_color_set(ad->bg, 66, 162, 206);
            my_box_pack(box, ad->bg, 1.0, 1.0, -1.0, -1.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

create_gengrid() 是一种用于创建并在之后返回一个 GenGrid 小部件的函数。过一会儿我们将创建这个小部件。

elm_bg_color_set() 是一种用于指定 Bg 小部件背景颜色的函数。该功能在本示例中不是必需的，但是我们仍然使用它来标识 Bg 小部件的位置。如果您想为其标注注释，请在完成本示例之后标注。

我们现在将定义一个用于创建 GenGrid 小部件的函数。在 create_base_gui() 函数之上添加一个新函数。

```
static Evas_Object*
create_gengrid(appdata_s *ad)
{
        Elm_Gengrid_Item_Class *gic;
        Evas_Object *gengrid;
        char buf[PATH_MAX];

        gengrid = elm_gengrid_add(ad->conform);
        elm_gengrid_item_size_set(gengrid, ELM_SCALE_SIZE(60), ELM_SCALE_SIZE(6
0));
        elm_gengrid_horizontal_set(gengrid, EINA_TRUE);

        gic = elm_gengrid_item_class_new();
        gic->func.content_get = gengrid_content_get_cb;

        for(int i = 0; i < 10; i++) {
                itemdata_s *id = calloc(sizeof(itemdata_s), 1);
                snprintf(buf, sizeof(buf), "%s%d.jpg", app_get_resource_path(),
i);
                id->index = i;
                id->path = strdup(buf);
                elm_gengrid_item_append(gengrid, gic, id, NULL, id);
        }

        return gengrid;
}
```

elm_gengrid_add() 是一种用于创建 GenGrid 小部件的 API。

elm_gengrid_item_size_set() 是一种用于指定 GenGrid 小部件的图标项目尺寸的 API。

elm_gengrid_horizontal_set() 是一种用于将滑动方向设置/取消设置为水平的 API。默认情况下滑动方向被设置为垂直。

elm_gengrid_item_class_new() 是一种用于为 GenGrid 项目创建项目类的 API。

Elm_Gengrid_Item_Class 是 GenList 小部件的项目类。类类型如下：
  - func.content_get：分配用于指定图标项目的回调函数的名称。
  - item_style：指定项目样式。默认设置为“default”。
  - func.text_get：为用于指定文本项目的回调函数分配名称。
  - func.del：为项目删除事件回调函数分配名称。可以使用该函数删除数据。

app_get_resource_path() 是一种用于返回 /res 文件夹绝对路径的 API。

elm_gengrid_item_append() 是一种用于向 GenGrid 添加新项目的 API。以下是按次序列出的参数：GenGrid 对象、项目类、项目数据结构、项目选择事件回调函数的名称和用户数据。

我们现在将定义一个向 GenGrid 项目添加图标图像的函数。在 create_gengrid() 函数之上添加一个新函数。一旦创建一个新项目，将会自动调用该函数。

```
static Evas_Object*
gengrid_content_get_cb(void *data, Evas_Object *obj, const char *part)
{
        itemdata_s *id = data;

        if (!strcmp(part, "elm.swallow.icon")) {
                Evas_Object *img = elm_image_add(obj);

                elm_image_file_set(img, id->path, NULL);
                elm_image_aspect_fixed_set(img, EINA_FALSE);
                evas_object_show(img);
                return img;
        }
        return NULL;
}
```

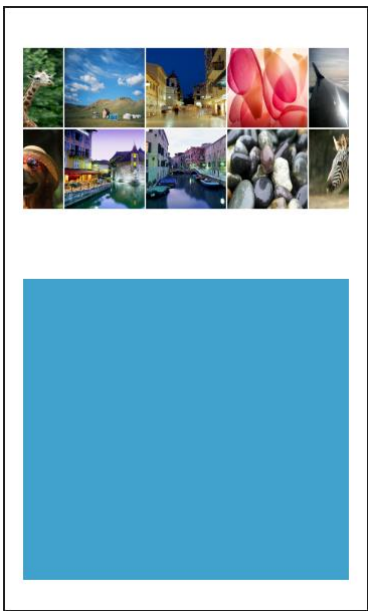
上述函数的第一个参数是用户数据，第二个参数是项目对象。元素的类型将被传递给第三个参数。
一个 GenGrid 小部件项目由多个元素组成，而且当元素类型为“elm.swallow.icon”时，您需要创建图标图像。

elm_image_add() 是一种用于创建 Image 对象的 API。

elm_image_file_set() 是一个通过为 Image 对象指定图像文件路径来加载该图像的 API。

elm_image_aspect_fixed_set() 是一种用于指定是否裁剪图像的 API。若将 EINA_TRUE 传递给第二个参数，则会缩小图像的尺寸，而不裁剪图像的任何区域。若传递 EINA_FALSE，则会裁剪掉图像的一部分区域，以使图像填满整个项目区域。默认设置为 EINA_TRUE。

编译和运行该示例。在屏幕的顶端显示了十个图标图像。向左和向右拖动屏幕。图像列表随之滚动。下面显示的正方形是 Bg 小部件。



2）在 Bg 小部件中显示选定的图标

在本小节中，我们将实施一项功能，以便当用户从 GenGrid 中选择一个图标时，将其图像显示在一个 Bg 小部件中。

在源文件的顶端添加一个全局变量。该全局变量用于存储 appdata。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *gengrid;
        Evas_Object *bg;
} appdata_s;

appdata_s* m_ad = 0;
```

然后，在 create_base_gui() 函数的开头初始化该全局变量。

```
static void
create_base_gui(appdata_s *ad)
{
        m_ad = ad;
```

我们现在将创建一个在用户选择 GenGrid 项目时执行的事件回调函数。如下所示修改底部 create_gengrid() 函数的代码。我们将项目选择事件回调函数指定为 gengrid_it_cb。

```
        for(int i = 0; i < 10; i++) {
                itemdata_s *id = calloc(sizeof(itemdata_s), 1);
                snprintf(buf, sizeof(buf), "%s%d.jpg", app_get_resource_path(),
i);
                id->index = i;
                id->path = strdup(buf);
                elm_gengrid_item_append(gengrid, gic, id, gengrid_it_cb, id);
                //elm_gengrid_item_append(gengrid, gic, id, NULL, id);
        }
```

```
        return gengrid;
}
```

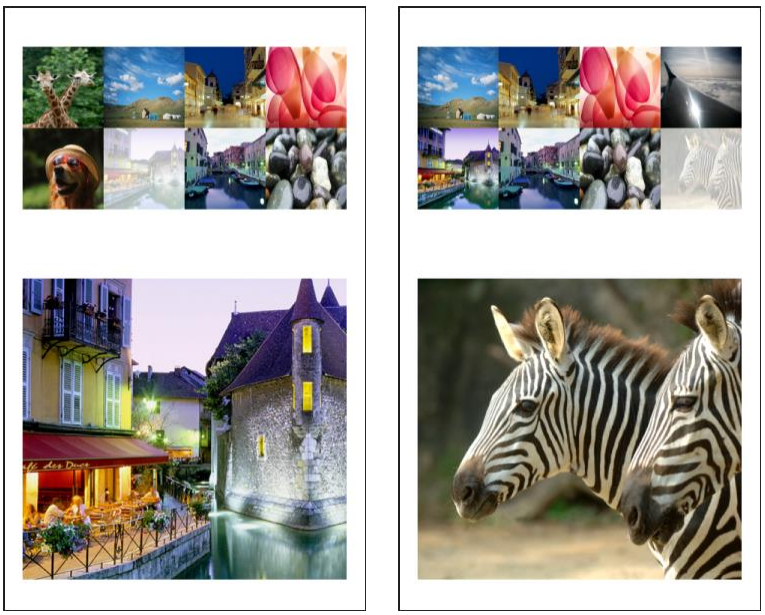
在 create_gengrid() 函数之上添加一个新函数。添加的函数是项目选择事件回调函数。

```
static void
gengrid_it_cb(void *data, Evas_Object *obj, void *event_info)
{
        itemdata_s *id = data;
        elm_bg_file_set(m_ad->bg, id->path, NULL);
}
```

该函数的第一个参数是项目数据结构。路径属性包含图像文件的路径。

elm_bg_file_set() 是一种用于为 Bg 小部件指定背景图像文件的 API。

再次运行该示例，并选择一个项目图标。匹配的图像将显示在 Bg 小部件中。



3）相关 API

Elm_Gengrid_Item_Class：一个 GenGrid 项目的类。类类型如下：
- func.content_get：分配用于指定图标项目的回调函数的名称。
- item_style：指定项目样式。默认设置为“default”。
- func.text_get：为用于指定文本项目的回调函数分配名称。
- func.del：为项目删除事件回调函数分配名称。可以使用该函数删除数据。

Evas_Object*elm_gengrid_add(Evas_Object *parent)：一种用于创建 GenGrid 小部件的 API。/ 参数：父级对象。

void elm_gengrid_item_size_set(Evas_Object *obj, Evas_Coord w, Evas_Coord h)：一种用于指定 GenGrid 的图标项目尺寸的 API。/ 参数：GenGrid 对象、宽度和高度。

void elm_gengrid_horizontal_set(Evas_Object *obj, Eina_Bool horizontal)：一种用于将 GenGrid 滑动方向设置/取消设置为水平的 API。默认情况下滑动方向被设置为垂直。/ 参数：GenGrid 对象和 EINA_TRUE 或 EINA_FALSE。

Elm_Gengrid_Item_Class *elm_gengrid_item_class_new(void)：一种用于创建 GenGrid 项目类的 API。

char *app_get_resource_path(void)：一个返回 /res 文件夹绝对路径的 API。

Elm_Object_Item *elm_gengrid_item_append(Evas_Object *obj, const Elm_Gengrid_Item_Class *gic, const void *data, Evas_Smart_Cb func, const void *func_data)：一种用于向 GenGrid 添加新项目的 API。/ 参数：GenGrid 对象、项目类、项目数据结构、项目选择事件回调函数的名称和用户数据。

Evas_Object *elm_image_add(Evas_Object *parent)：一种用于创建 Image 对象的 API。

Eina_Bool elm_image_file_set(Evas_Object *obj, const char *file, const char *group)：一个通过为 Image 对象指定图像文件路径来加载该图像的 API。

void elm_image_aspect_fixed_set(Evas_Object *obj, Eina_Bool fixed)：一种用于设置 Image 对象的图像裁剪属性的 API。若将 EINA_TRUE 传递给第二个参数，则会缩小图像的尺寸，而不裁剪图像的任何区域。若传递 EINA_FALSE，则会裁剪掉图像的一部分区域，以使图像填满整个项目区域。默认设置为 EINA_TRUE。

18. 使用 WebView 小部件创建一个简单 Web 浏览器

若要在屏幕上显示网页，您需要使用 WebView 小部件。若要创建 WebView 小部件，必须指定一个 Evas 对象作为其父级对象。Evas 是 EFL 中使用的画布。在该示例中，我们将使用 WebView 小部件创建一个简单 Web 浏览器示例。

1) 创建一个 WebView 小部件并显示网页

创建一个新的源项目，并将项目名称指定为“WebViewEx”。

创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在顶端添加新代码。此代码将声明一个库标题文件，并向 appdata 结构添加变量。

```
#include "webviewex.h"
#include <EWebKit.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *entry;
        Evas_Object *web_view;
} appdata_s;
```

在 create_base_gui() 函数之上添加一个新函数。该函数将向一个 Table 容器添加一个小部件。

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int x, int y, int w, int
h)
{
   evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
   evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
   elm_table_pack(table, child, x, y, w, h);
   evas_object_show(child);
}
```

然后，转至 create_base_gui() 函数并添加新代码。此代码将创建一个 Box 容器、一个 Table 容器、一个 Entry 小部件、三个 Button 小部件和一个 WebView 小部件。Conformant 和 Label 在本示例中都不是必需的，因此我们将对其标注注释。

```
    /* Conformant */
    /*ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);*/

    /* Label*/
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "Hello EFL");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
    elm_object_content_set(ad->conform, ad->label);
    evas_object_show(ad->label);*/

    {
        /* Box to put the table in so we can bottom-align the table
         * window will stretch all resize object content to win size */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_win_resize_object_add(ad->win, box);
        evas_object_show(box);

        /* Table */
        Evas_Object *table = elm_table_add(ad->win);
        /* Make table homogenous - every cell will be the same size */
        elm_table_homogeneous_set(table, EINA_TRUE);
        /* Set padding of 10 pixels multiplied by scale factor of UI */
        elm_table_padding_set(table, 5 * elm_config_scale_get(), 10 * elm_confi
g_scale_get());
        /* Let the table child allocation area expand within in the box */
        evas_object_size_hint_weight_set(table, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
        /* Set table to fiill width but align to bottom of box */
        evas_object_size_hint_align_set(table, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_box_pack_end(box, table);
```

```
    evas_object_show(table);

    {
        /* Entry */
        ad->entry = elm_entry_add(ad->win);
        elm_entry_scrollable_set(ad->entry, EINA_TRUE);
        eext_entry_selection_back_event_allow_set(ad->entry, EINA_TRUE);
        elm_object_text_set(ad->entry, "http://www.tizen.org");
        my_table_pack(table, ad->entry, 0, 0, 3, 1);

        /* Button-1 */
        Evas_Object *btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Prev");
        evas_object_smart_callback_add(btn, "clicked", btn_prev_cb, ad);
        my_table_pack(table, btn, 0, 1, 1, 1);

        /* Button-2 */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Go");
        evas_object_smart_callback_add(btn, "clicked", btn_go_cb, ad);
        my_table_pack(table, btn, 1, 1, 1, 1);

        /* Button-3 */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Next");
        evas_object_smart_callback_add(btn, "clicked", btn_next_cb, ad);
        my_table_pack(table, btn, 2, 1, 1, 1);

        /* WebView */
        Evas *evas = evas_object_evas_get(ad->win);
        ad->web_view = ewk_view_add(evas);
        ewk_view_url_set(ad->web_view, elm_object_text_get(ad->entry) );
        my_table_pack(table, ad->web_view, 0, 2, 3, 8);
    }
}

/* Show window after base gui is set up */
evas_object_show(ad->win);
```

我们为 Conformant 和 Label 标注了注释，并且创建了一个 Entry 小部件、三个 Button 小部件和一个 WebView 小部件。

若将 ELM_WIN_INDICATOR_SHOW 传递给 elm_win_indicator_mode_set() 函数的第二个参数，将会在屏幕的顶端显示一个指示器，若传递 ELM_WIN_INDICATOR_HIDE，则会隐藏该指示器。为了最大化利用空间，建议隐藏该指示器。

evas_object_evas_get() 是一种用于创建 Evas 对象的 API。Evas 是一张画布，您可以在上面绘制图像或形状。

ewk_view_add() 是一种用于创建 WebView 小部件的 API。若要创建 WebView 小部件，必须指定一个 Evas 对象作为其父级对象。

ewk_view_url_set() 是一种用于为 WebView 小部件指定 URL 路径的 API。为第一个参数指定 WebView 小部件对象；为第二个参数指定一个 URL 路径。我们传递了 Entry 小部件的标题文本：http://www.tizen.org 是 Tizen 开发人员支持网站。

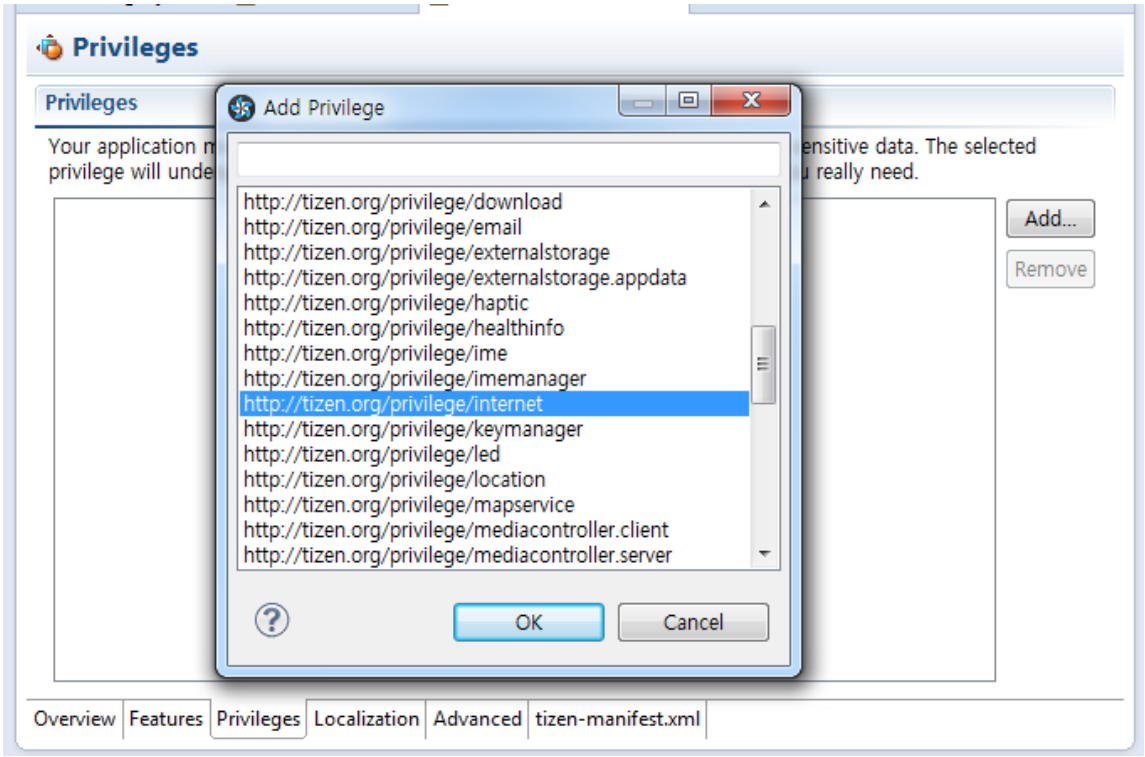
由于我们创建了三个 Button，因此需要三个回调函数。在 create_base_gui() 函数之上添加三个新函数。稍后我们将会定义这些函数的内容。

```
static void
btn_go_cb(void *data, Evas_Object *obj, void *event_info)
{
}

static void
btn_prev_cb(void *data, Evas_Object *obj, void *event_info)
{
}

static void
btn_next_cb(void *data, Evas_Object *obj, void *event_info)
{
}
```

如果您是在此状态下运行该示例，将不会显示网页。这是因为必须支持用户权限才能够使用网络。

在该源项目的 root 文件夹中双击 tizen-manifest.xml 文件以打开该文件。您将会在底部看到一些选项卡按钮。从这些按钮中选择 Privileges。

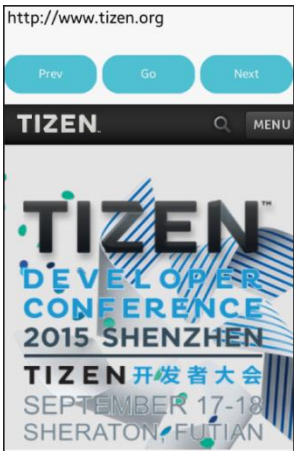
然后，单击 Add 按钮，当出现一个弹出窗口时，从列表中选择 http://tizen.org/privilege/internet，并单击 OK 按钮。



如果在底部的选项卡按钮中选择 tizen-manifest.xml，将会显示源代码。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.tizen.webviewex" version="1.0.0">
        <profile name="mobile"/>
        <ui-application appid="org.tizen.webviewex" exec="webviewex" multiple="false" nodisplay="false" taskmanage="true" type="capp">
                <label>webviewex</label>
                <icon>webviewex.png</icon>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/internet</privilege>
        </privileges>
</manifest>
```

现在，该示例将能够进行网络通信。编译和运行该示例。您将会看到 Tizen 开发人员支持网站。



2) 访问所需的网站

在本小节中，我们将实施一项功能，以便当用户在 Entry 小部件中输入一个 URL 地址并点击 Go 按钮时，将会让用户直接进入该 URL 地址。向 Go 按钮回调函数添加新代码。

```
static void
btn_go_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s* ad = data;
        ewk_view_url_set(ad->web_view, elm_object_text_get(ad->entry) );
}
```

用户数据被传递给该回调函数的第一个参数。

新添加的代码会将在 Entry 小部件中输入的文本指定为该 WebView 小部件的一个 URL 地址。

再次运行该示例。在 Entry 中输入您想要访问的网站地址，然后点击 Go 按钮。网页将显示在 WebView 中。

3) 转至上一页/下一页

在本小节中，我们将实施一项功能，以便当点击 Prev 按钮时让用户直接进入上一页面，而当点击 Next 按钮时让用户直接进入下一页面。

向 btn_prev_cb() 函数添加新代码。

```
static void
btn_prev_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s* ad = data;
        if( ewk_view_back_possible( ad->web_view ) == EINA_TRUE )
                ewk_view_back( ad->web_view );
}
```

ewk_view_back_possible() 是一种用于确定是否能够返回到上一屏幕的 API。

ewk_view_back() 是一个使应用程序移至上一屏幕的 API。

然后，向 btn_next_cb() 函数添加新代码。

```
static void
btn_next_cb(void *data, Evas_Object *obj, void *event_info)
```

```
{
        appdata_s* ad = data;
        if( ewk_view_forward_possible( ad->web_view ) == EINA_TRUE )
                ewk_view_forward( ad->web_view );
}
```

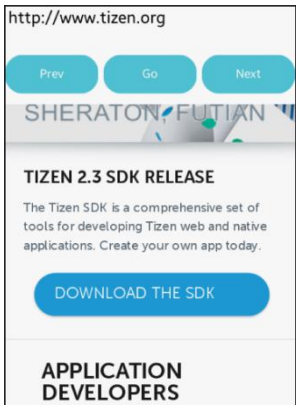
ewk_view_forward_possible() 是一种用于确定是否能够转到下一屏幕的 API。

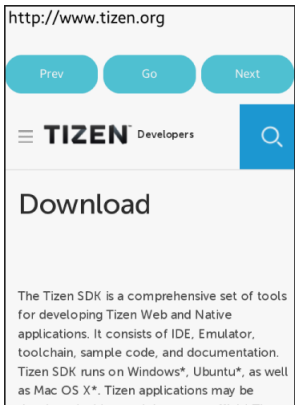
ewk_view_forward() 是一个使应用程序移至下一屏幕的 API。

再次运行该示例。向上滑动网页，将会显示链接“DOWNLOAD THE SDK”。单击该链接。如果未显示该链接，请单击屏幕上显示的其他任何链接。

当显示一个新页面时，单击 Prev 按钮。将会切换至原来的屏幕。

此时，单击 Next 按钮。将会再次显示 Download 屏幕。





5) 相关 API

Evas *evas_object_evas_get(const Evas_Object *obj)：一种用于创建 Evas 对象的 API。Evas 是一张画布，您可以在上面绘制图像或形状。

Evas_Object* ewk_view_add(Evas* e)：一种用于创建 WebView 小部件的 API。若要创建 WebView 小部件，必须指定一个 Evas 对象作为其父级对象。

Eina_Bool ewk_view_url_set(Evas_Object* o, const char* url)：一种用于为 WebView 小部件指定 URL 路径的 API。参数：WebView 小部件对象和 URL 路径。

Eina_Bool ewk_view_back_possible(Evas_Object* o)：一种用于确定是否能够返回到上一屏幕的 API。

Eina_Bool ewk_view_back(Evas_Object* o)：一个移至上一屏幕的 API。

Eina_Bool ewk_view_forward_possible(Evas_Object* o)：一种用于确定是否能够转到下一屏幕的 API。

Eina_Bool ewk_view_forward(Evas_Object* o)：一个移至下一屏幕的 API。

19. 利用 Layout 容器实施 Tab 屏幕

利用 Tab 屏幕，您可以在屏幕之间轻松地切换。在该示例中，我们将学习如何使用 Layout 和 Box 容器实施 Tab 屏幕。

1) 创建一个 Layout 容器并放置小部件

创建一个新的源项目，并将项目名称指定为“LayoutEx”。

创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在顶端添加新代码。此代码将向 appdata 结构添加变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *box1;
        Evas_Object *box2;
} appdata_s;
```

在该示例中，我们将实施 Tab 屏幕。box1 将成为第一个 Tab 屏幕，而 box 2 将成为第二个 Tab 屏幕。

在 create_base_gui() 函数之上添加一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
```

```
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

然后，返回至 create_base_gui() 函数，并创建一个 Box 容器、一个 Table 容器和 Button 小部件。为 Conformant 和 Label 标注注释。

```
    /* Conformant */
    /*ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);*/

    /* Label*/
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
    elm_object_content_set(ad->conform, ad->label);*/

    {
        /* Main Box */
        Evas_Object *box = create_box(ad->win);

        /* Table */
```

```
        Evas_Object *table = elm_table_add(ad->win);
        /* Make table homogenous - every cell will be the same size */
        elm_table_homogeneous_set(table, EINA_TRUE);
        /* Set padding of 10 pixels multiplied by scale factor of UI */
        elm_table_padding_set(table, 5 * elm_config_scale_get(), 5 * elm_config
_scale_get());
        /* Let the table child allocation area expand within in the box */
        evas_object_size_hint_weight_set(table, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
        /* Set table to fiill width but align to bottom of box */
        evas_object_size_hint_align_set(table, EVAS_HINT_FILL, 1.0);
        elm_box_pack_end(box, table);
        evas_object_show(table);

        {
            /* Tab Button-1 */
            Evas_Object *btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "Tab-1");
            evas_object_smart_callback_add(btn, "clicked", btn_tab1_cb, ad);
            my_table_pack(table, btn, 0, 5, 1, 1);

            /* Tab Button-2 */
            btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "Tab-2");
            evas_object_smart_callback_add(btn, "clicked", btn_tab2_cb, ad);
            my_table_pack(table, btn, 1, 5, 1, 1);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

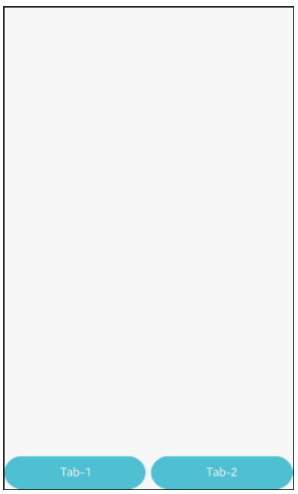
我们创建了一个 Table，并向该 Table 添加了两个 Button。我们使用了一个 Box 来指定小部件之间的间距。

我们现在将为这些 Button 创建回调函数。在 create_base_gui() 函数之上添加两个新函数。稍后我们将会定义这些函数的内容。

```
static void
btn_tab1_cb(void *data, Evas_Object *obj, void *event_info)
{
}
```

```
static void
btn_tab2_cb(void *data, Evas_Object *obj, void *event_info)
{
}
```

编译和运行该示例。您会在屏幕底部看到两个 Button。



2) 创建 Tab 屏幕

我们现在将创建两个 Tab 屏幕。向 create_base_gui() 函数添加新代码。此代码将创建两个 Layout 和两个 Box，并在其顶端添加小部件。

```
        /* Tab Button-2 */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Tab-2");
        evas_object_smart_callback_add(btn, "clicked", btn_tab2_cb, ad);
        my_table_pack(table, btn, 1, 5, 1, 1);

        /* Layout-1 */
        Evas_Object *layout1 = elm_layout_add(ad->win);
        elm_layout_theme_set(layout1, "layout", "drawer", "panel");
        my_table_pack(table, layout1, 0, 0, 2, 5);
```

```
        /* Box-1 */
        ad->box1 = create_box(layout1);
        elm_win_resize_object_add(ad->win, ad->box1);

        {
            /* Label */
            ad->label = elm_label_add(layout1);
            elm_object_text_set(ad->label, "Tab-1");
                my_box_pack(ad->box1, ad->label, 1.0, 0.0, -1.0, 0.5);
        }

        /* Layout-2 */
        Evas_Object *layout2 = elm_layout_add(ad->win);
        elm_layout_theme_set(layout2, "layout", "drawer", "panel");
        my_table_pack(table, layout2, 0, 0, 2, 5);

        /* Box-2 */
        ad->box2 = create_box(layout2);
        elm_win_resize_object_add(ad->win, ad->box2);
        evas_object_hide(ad->box2);

        {
            /* Button */
            btn = elm_button_add(layout2);
            elm_object_text_set(btn, "Tab-2");
            my_box_pack(ad->box2, btn, 1.0, 0.0, -1.0, 0.5);
        }
    }
}

/* Show window after base gui is set up */
evas_object_show(ad->win);
```

我们创建了两个 Layout，并在每个 Layout 顶端添加了一个 Box。这些 Layout 将作为 Tab 屏幕。我们向第一个 Tab 屏幕添加了一个 Label，并向第二个 Tab 屏幕添加了 Button。

elm_layout_add() 是一种用于创建 Layout 容器的 API。

elm_layout_theme_set() 是一个为 Layout 指定主题样式的 API。若要将一个 Window 置于 Layout 中，您需要将该 Layout 的主题样式指定为“panel”。为此，请传递以下参数值：

elm_layout_theme_set(layout, "layout", "drawer", "panel")

evas_object_hide() 是一种用于隐藏 Window 的 API。它所起的作用与 evas_object_show() 相反。

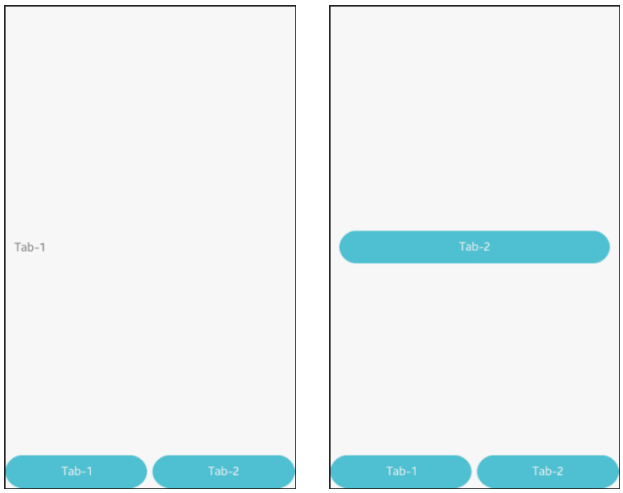
我们现在将实施一项功能，以便当点击屏幕底部的两个 Button 之一时显示匹配的 Tab 屏幕并隐藏另一个 Tab 屏幕。向 Button 回调函数添加新代码。

```
static void
btn_tab1_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        evas_object_show(ad->box1);
        evas_object_hide(ad->box2);
}

static void
btn_tab2_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        evas_object_hide(ad->box1);
        evas_object_show(ad->box2);
}
```

点击第一个 Button 将会显示第一个 Layout 并隐藏第二个 Layout。

点击第二个 Button 将会显示第二个 Layout 并隐藏第一个 Layout。

再次运行该示例。点击屏幕底部的一个 Button 将会显示匹配的 Tab 屏幕。



4）相关 API

Evas_Object *elm_layout_add(Evas_Object *parent)：一种用于创建 Layout 容器的 API。

Eina_Bool elm_layout_theme_set(Evas_Object *obj, const char *clas, const char *group, const char *style)：一个为 Layout 指定主题样式的 API。若要将一个 Window 置于 Layout 中，您需要将该 Layout 的主题样式指定为“panel”。

void evas_object_hide(Evas_Object *obj))：一种用于隐藏 Window 的 API。它所起的作用与 evas_object_show() 相反。

20. 使用 Naviframe 小部件实施页首和导航栏

若要实施 Header 和 Footer (Toolbar)，您可以使用 Naviframe 小部件。在该示例中，我们将学习如何在页首显示标题文本，在页脚显示一个导航栏。

1) 创建 Header

创建一个新的源项目，并将项目名称指定为“NaviframeEx”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在顶端添加新代码。此代码将向 appdata 结构添加变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *nf;
        Evas_Object *layout;
        Elm_Object_Item *frame_item;
        Evas_Object *toolbar;
        Elm_Object_Item *btn1;
        Elm_Object_Item *btn2;
        Elm_Object_Item *btn3;
} appdata_s;
```

nf 是一个 Naviframe 的对象，而 layout 是将被添加到一个 Naviframe 的主容器。

Elm_Object_Item 是一个 Naviframe 的项目结构。

btn1-btn3 是将被添加到一个导航栏的 Button。

在 create_base_gui() 函数之上添加一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

然后，转至 create_base_gui() 函数并添加新代码。此代码将创建一个 Naviframe、一个 Layout 和一个 Naviframe 项目。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
```

```
        /* Naviframe */
        ad->nf = elm_naviframe_add(ad->conform);
        elm_object_part_content_set(ad->conform, "elm.swallow.content", ad->nf);
        elm_object_content_set(ad->conform, ad->nf);

        /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->conform);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->nf, box);
        evas_object_show(box);

        {
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "Press Toolbar Button");
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.0);

            /* Header */
            ad->frame_item = elm_naviframe_item_push(ad->nf, "Naviframe Ex", NU
LL, NULL, box, NULL);

            /* Toolbar */
            ad->toolbar = toolbar_add(ad, ad->nf);
            elm_object_item_part_content_set(ad->frame_item, "toolbar", ad->too
lbar);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

我们在一个 Conformant 之上添加了一个 Naviframe，还在该 Naviframe 之上添加了一个 Box。

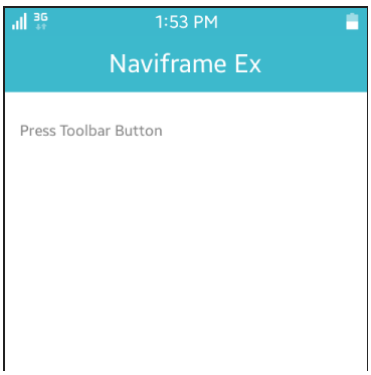
elm_naviframe_add() 是一种用于创建 Naviframe 对象的 API。

evas_object_size_hint_weight_set() 是一种用于指定对象尺寸提示的 API。以下是按次序列出的参数：对象、水平尺寸提示和垂直尺寸提示。EVAS_HINT_EXPAND 是一种用于指定尽可能大的对象尺寸的选项。

evas_object_size_hint_align_set() 将指定对象的对齐提示。以下是按次序列出的参数：对象、水平对齐提示和垂直对齐提示。EVAS_HINT_FILL 是一个使对象在指定区域内填满尽可能多空间的选项。

elm_naviframe_item_push() 是一种用于创建 Naviframe 项目的 API。以下是按次序列出的参数：Naviframe 对象、标题文本、用于移至上一项目的 Button 的对象、用于移至下一项目的 Button 的对象、内容和项目样式。

编译和运行该示例。标题文本将显示在 Header 中。



2) 在 Footer 中显示一个 Toolbar

在本小节中，我们将向一个 Naviframe 项目添加一个 Footer，并创建三个选项卡按钮。

在 create_base_gui() 函数之上添加一个新函数。将创建此函数，然后返回一个 Toolbar。

```
static Evas_Object *toolbar_add(appdata_s *ad, Evas_Object *parent)
{
        Evas_Object *toolbar = elm_toolbar_add(parent);
        evas_object_show(toolbar);

        ad->btn1 = elm_toolbar_item_append(toolbar, NULL, "Left", on_btn1_cb, a
d);
        ad->btn2 = elm_toolbar_item_append(toolbar, NULL, "Center", on_btn2_cb,
ad);
        ad->btn3 = elm_toolbar_item_append(toolbar, NULL, "Right", on_btn3_cb, ad);
```

```
        return toolbar;
}
```

elm_toolbar_add() 是一种用于创建 Toolbar 对象的 API。

elm_toolbar_item_append() 是一种用于向 Toolbar 添加项目的 API。以下是按次序列出的参数：Toolbar 对象、图标图像、标题文本、Button 单击事件回调函数的名称和用户数据。

现在，我们为这些选项卡按钮创建回调函数。在 toolbar_add() 函数之上添加三个新函数。当用户点击一个选项卡按钮时，此代码将更改一个 Label 小部件中的文本。

```
static void on_btn1_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = (appdata_s*)data;
        elm_object_text_set(ad->label, "Button-1 Pressed");
}

static void on_btn2_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = (appdata_s*)data;
        elm_object_text_set(ad->label, "Button-2 Pressed");
}

static void on_btn3_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = (appdata_s*)data;
        elm_object_text_set(ad->label, "Button-3 Pressed");
}
```

现在，我们需要调用 elm_toolbar_add() 函数来创建一个 Toolbar。在 create_base_gui() 函数的结尾添加新代码。

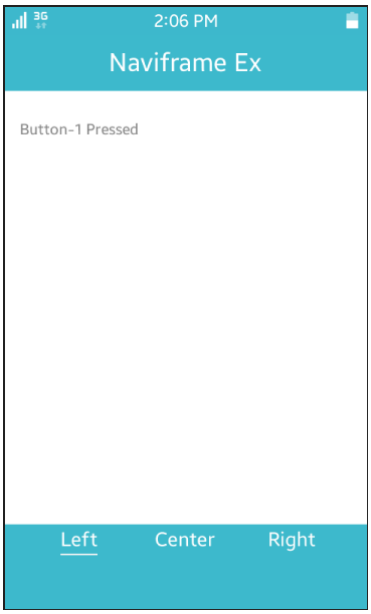
```
        /* Header */
        ad->frame_item = elm_naviframe_item_push(ad->nf, "Naviframe Ex", NULL, NULL, box, NULL);

        /* Toolbar */
```

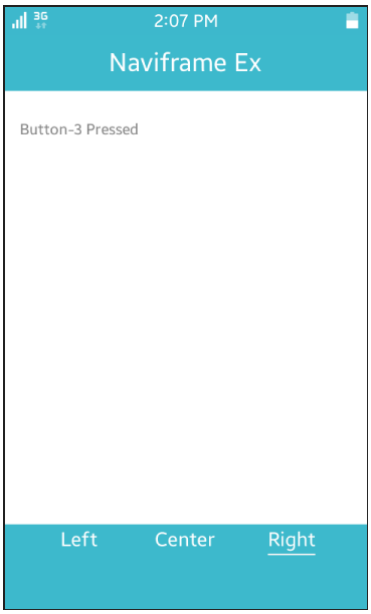
```
        ad->toolbar = toolbar_add(ad, ad->nf);
        elm_object_item_part_content_set(ad->frame_item, "toolbar", ad->toolba
r);
    }
}
```

我们通过调用 toolbar_add() 函数创建了一个 Toolbar。接着，将该 Toolbar 指定为 Naviframe 项目的内容。

elm_object_item_part_content_set() 是一种用于为对象项目指定内容的 API。以下是按次序列出的参数：对象项目、内容部分名称和内容对象。

再次运行该示例。您现在将会在屏幕底部看到一个 Footer，而且在该 Footer 中有三个 Button。点击一个 Button 将会更改 Label 小部件中的文本。在选中的 Button 下面将会显示一个勾选标记。





3) 更改 Toolbar 的属性

如果将文本置于顶端，屏幕的外观可能不美观。因此，我们会将文本移到中央位置，并移除勾选标记。

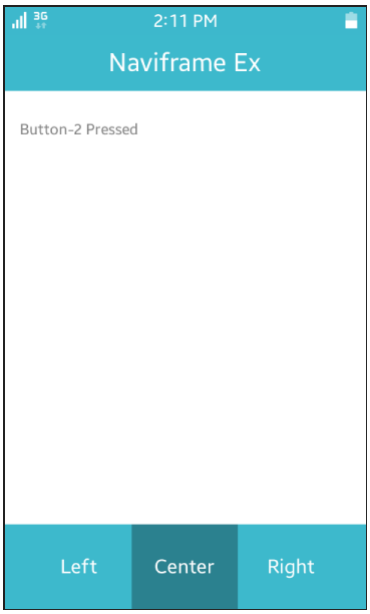
向 toolbar_add() 函数添加两行新代码。

```
static Evas_Object *toolbar_add(appdata_s *ad, Evas_Object *parent)
{
        Evas_Object *toolbar = elm_toolbar_add(parent);
        evas_object_show(toolbar);
        elm_toolbar_select_mode_set(toolbar, ELM_OBJECT_SELECT_MODE_NONE);
        elm_toolbar_transverse_expanded_set(toolbar, EINA_TRUE);

        ad->btn1 = elm_toolbar_item_append(toolbar, NULL, "Left", on_btn1_cb, a
d);
        ad->btn2 = elm_toolbar_item_append(toolbar, NULL, "Center", on_btn2_cb,
ad);
        ad->btn3 = elm_toolbar_item_append(toolbar, NULL, "Right", on_btn3_cb, a
d);
        return toolbar;
}
```

elm_toolbar_select_mode_set() 是一种用于指定 Toolbar 的选择模式的 API。将 ELM_OBJECT_SELECT_MODE_NONE 传递给第二个参数，则会移除勾选标记。默认设置为 ELM_OBJECT_SELECT_MODE_DEFAULT。

elm_toolbar_transverse_expanded_set() 是一种用于指定是否扩展 Toolbar 尺寸的 API。若向第二个参数输入 EINA_TRUE，将会使 Toolbar 填满 Footer 的整个区域，因而文本将显示在中央位置。

再次运行该示例。Toolbar 的文本将显示在中央位置，而且点击选项卡按钮将不会显示勾选标记。



4) 相关 API

Elm_Object_Item：一个对象项目或 Naviframe 项目的结构。

Elm_Object_Item *elm_naviframe_item_push(Evas_Object *obj, char *title_label, Evas_Object *prev_btn, Evas_Object *next_btn, Evas_Object *content, char *item_style)：一种用于创建 Naviframe 对象的 API。/ 参数：Naviframe 对象、标题文本、用于移至上一项目的 Button 的对象、用于移至下一项目的 Button 的对象、内容和项目样式。

void evas_object_size_hint_weight_set(Evas_Object *obj, double x, double y)：一种用于指定对象大致尺寸提示的 API。/ 参数：对象、水平尺寸提示和垂直尺寸提示。EVAS_HINT_EXPAND 是一种用于指定尽可能大的对象尺寸的选项。

void evas_object_size_hint_align_set(Evas_Object *obj, double x, double y)：一种用于指定对象对齐提示的 API。/ 参数：对象、水平对齐提示和垂直对齐提示。EVAS_HINT_FILL 是一个使对象在指定区域内填满尽可能多空间的选项。

Elm_Object_Item *elm_naviframe_item_push(Evas_Object *obj, char *title_label, Evas_Object *prev_btn, Evas_Object *next_btn, Evas_Object *content, char *item_style)：一种用于创建 Naviframe 项目的 API。/ 参数：Naviframe 对象、标题文本、用于移至上一项目的 Button 的对象、用于移至下一项目的 Button 的对象、内容和项目样式。

Evas_Object *elm_toolbar_add(Evas_Object *parent)：一种用于创建 Toolbar 的 API。

Elm_Object_Item *elm_toolbar_item_append(Evas_Object *obj, char *icon, char *label, Evas_Smart_Cb func, void *data)：一种用于向 Toolbar 添加项目的 API。/ 参数：Toolbar 对象、图标图像、标题文本、Button 单击事件回调函数的名称和用户数据。

void elm_object_item_part_content_set(Elm_Object_Item *it, char *part, Evas_Object *content)：一种用于为对象项目指定内容的 API。/ 参数：对象项目、内容部分名称和内容对象。

void elm_toolbar_select_mode_set(Evas_Object *obj, Elm_Object_Select_Mode mode)：一种用于指定 Toolbar 的选择模式的 API。/ 参数：Toolbar 对象和选择模式。默认的选择模式是 ELM_OBJECT_SELECT_MODE_DEFAULT。若传递 ELM_OBJECT_SELECT_MODE_NONE，将隐藏勾选标记。

void elm_toolbar_transverse_expanded_set(Evas_Object *obj, Eina_Bool transverse_expanded)：一种用于指定是否扩展 Toolbar 尺寸的 API。/ 参数：Toolbar 对象以及是否扩展 Toolbar 的尺寸。若输入 EINA_TRUE，将会使 Toolbar 填满 Footer 的整个区域，因而文本将显示在中央位置。

21. 使用 Box 容器依次放置小部件

为支持具有不同分辨率的各种终端，您可以在放置小部件时使用相对坐标。Table 容器允许您根据屏幕高宽比为小部件指定坐标。也可使用 Box 容器依次放置小部件或者使用相对坐标（左、右和中央；或者顶端、底部和中间位置）来放置小部件。此功能与 Android 系统的 LinearLayout 相似。在该示例中，我们将学习如何使用 Box 容器水平和垂直放置小部件。

1) 创建水平 Box

在该示例中，将创建三个 Box 容器和六个 Button 小部件。为使源代码变得简单，我们将先创建一个用于创建 Box 的函数和一个用于创建 Button 的函数。

创建一个新的源项目，并将项目名称指定为“BoxEx”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在 create_base_gui() 函数之上添加两个新函数。

```
static Evas_Object*
create_box(Evas_Object *parent)
{
        Evas_Object *box = elm_box_add(parent);
        evas_object_show(box);
        return box;
}

static Evas_Object *
create_button(Evas_Object *parent, char *text)
{
        Evas_Object *button = elm_button_add(parent);
        elm_object_text_set(button, text);
        evas_object_show(button);
        return button;
}
```

create_box() 是一种用于创建并在之后返回一个 Box 容器的函数。

elm_box_add() 是一种用于创建 Box 容器的 API。

create_button() 函数将创建一个 Button 小部件，为它指定标题文本，然后返回该小部件。

我们现在将使用刚刚创建的函数创建两个 Box 和三个 Button，并将它们水平放置。返回到 create_base_gui() 函数并添加新代码。该示例中将不使用 Label，因此删除它。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = create_box(ad->win);
        elm_box_padding_set(box, 10, 10);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);

        { /* child object - indent to how relationship */
            /* Create horizontal box */
            Evas_Object *horizontal_box = create_box(ad->win);
            elm_box_horizontal_set(horizontal_box, EINA_TRUE);
            elm_box_padding_set(horizontal_box, 10, 10);
            elm_box_pack_end(box, horizontal_box);

            { /* child object - indent to how relationship */
                Evas_Object *btn = create_button(horizontal_box, "Left");
                elm_box_pack_end(horizontal_box, btn);

                btn = create_button(horizontal_box, "Mid");
                evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, 0.0);
                evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0.0);
                elm_box_pack_end(horizontal_box, btn);

                btn = create_button(horizontal_box, "Right");
```

```
                elm_box_pack_end(horizontal_box, btn);
            }
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

elm_box_padding_set() 是一种用于指定 Box 容器边距的 API。以下是按次序列出的参数：容器对象、左边距和右边距、上边距和下边距。

evas_object_size_hint_weight_set() 是一个指定对象所占用空间大小提示的 API。以下是按次序列出的参数：对象、水平尺寸提示和垂直尺寸提示。EVAS_HINT_EXPAND 是一种用于指定尽可能大的对象尺寸的选项。

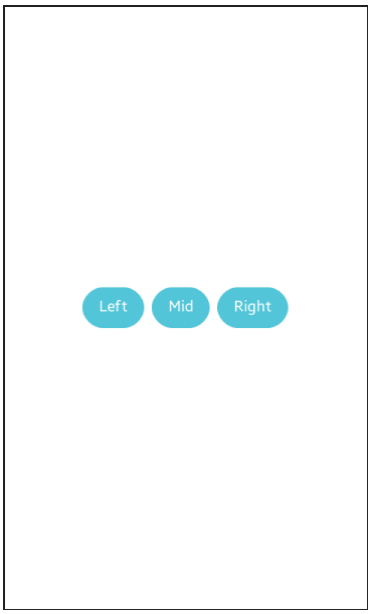
evas_object_size_hint_align_set() 是一种用于指定对象自身尺寸提示的选项。以下是按次序列出的参数：对象、水平尺寸提示和垂直尺寸提示。EVAS_HINT_FILL 是一种用于指定尽可能大的对象尺寸的选项。

elm_win_resize_object_add() 是一种用于将某个对象的尺寸指定为与另一对象尺寸相同的 API。我们将“win”传递给了第一个参数，将“main_box”传递给了第二个参数。这样做将会调整 main_box 的尺寸，以便它能够填满屏幕的整个区域。

elm_box_horizontal_set() 是一种用于指定 Box 对齐方向的 API。若将 EINA_TRUE 传递给第二个参数，则将对齐方式设置为水平；若传递 EINA_FALSE，则将对齐方式设置为垂直。默认设置为“垂直”。

elm_box_pack_end() 是一种用于向 Box 容器添加新对象的 API。如果对齐方式为水平，将会在屏幕右侧添加新对象。如果对齐方式是垂直，将会在屏幕底部添加新对象。

编译和运行该示例。三个 Button 将在屏幕中水平居中对齐。尽管已将第二个 Button 的尺寸指定为最大值，第二个 Button 还是占据了尽可能最小的空间。这是因为我们没有指定 Box 容器的宽度。



2) 最大化 Object 的尺寸

在本小节中，我们将学习如何将 Box 的宽度指定为最大值。向用于创建第二个 Button 的 create_base_gui() 函数代码添加新代码。

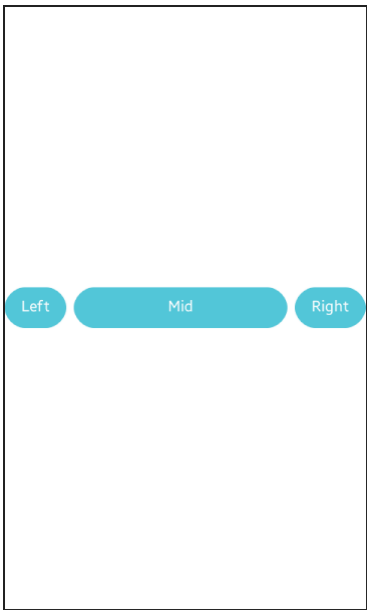
```
        /* Create horizontal box */
        Evas_Object *horizontal_box = create_box(ad->win);
        elm_box_horizontal_set(horizontal_box, EINA_TRUE);
        elm_box_padding_set(horizontal_box, 10, 10);
        evas_object_size_hint_weight_set(horizontal_box, EVAS_HINT_EXPAND, 0.0);
        evas_object_size_hint_align_set(horizontal_box, EVAS_HINT_FILL, 0.0);
        elm_box_pack_end(main_box, horizontal_box);
```

我们将 EVAS_HINT_EXPAND 传递给了 evas_object_size_hint_weight_set() 函数的第二个参数，以使该 Button 的宽度最大化。我们为第三个参数指定了 0.0，以使该 Button 的高度最小化。

然而，仅执行该操作并不会更改 Button 的宽度。evas_object_size_hint_weight_set() 是一种用于获取空间的函数；而 evas_object_size_hint_align_set() 是一种用于指定内容对齐方式的函数。

为 evas_object_size_hint_align_set() 函数的第二个参数输入一个水平对齐类型：0.0 表示左对齐；0.5 表示居中对齐；1.0 表示右对齐；传递 EVAS_HINT_FILL 则会将内容的宽度指定为最大值。
为第三个参数输入一个垂直对齐值类型。0.0 表示顶端对齐；0.5 表示居中对齐；1.0 表示底部对齐；传递 EVAS_HINT_FILL 则会将内容的高度指定为最大值。

再次运行该示例。您现在将会看到第二个 Button 的宽度扩大了。



3）更改 Box 的位置

由于第二个 Box 被放置在屏幕的中央位置，因此这些 Button 也将置于屏幕的中央位置。该 Box 容器会依次放置小部件。因此，向第一个 Box 添加一个新 Box，则会使第二个 Box 上移。在 create_base_gui() 函数的底部添加新代码。此代码将创建第三个 Box 并会将其添加到第一个 Box。

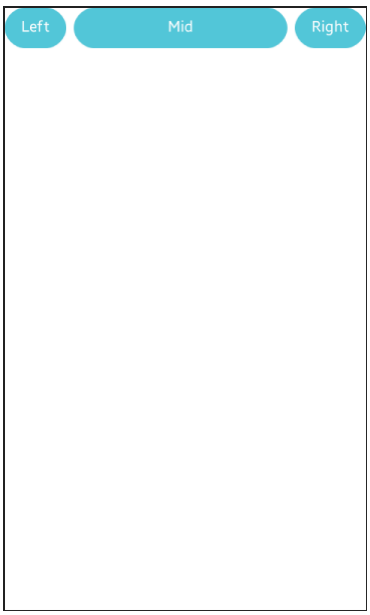
```
    btn = create_button(horizontal_box, "Right");
    elm_box_pack_end(horizontal_box, btn);
}

/* Create vertical box */
```

```
            Evas_Object *vertical_box = create_box(ad->win);
            elm_box_padding_set(vertical_box, 10, 10);
            // Set area size
            evas_object_size_hint_weight_set(vertical_box, EVAS_HINT_EXPAND, EV
AS_HINT_EXPAND);
            evas_object_size_hint_align_set(vertical_box, EVAS_HINT_FILL, EVAS_
HINT_FILL);
            elm_box_pack_end(box, vertical_box);
        }
    }
```

第一个 Box 的对齐方式为垂直，因此新添加的 Box 将置于第二个 Box 的下面。

我们将 EVAS_HINT_EXPAND 传递给 evas_object_size_hint_weight_set() 函数以获取了尽可能最大的空间，而且还将 EVAS_HINT_FILL 传递给 evas_object_size_hint_align_set() 函数以将内容的尺寸指定为最大值。

再次运行该示例，您现在将会看到 Button 已移至屏幕顶端。



4) 垂直对齐小部件

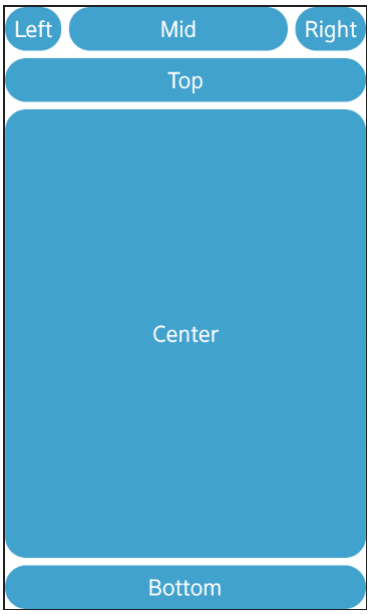
我们现在将向第三个 Box 添加三个 Button。由于没有指定 Button 的对齐方式，因此这些 Button 将垂直对齐。在 create_base_gui() 函数的底部添加新代码。

```
            /* Create vertical box */
            Evas_Object *vertical_box = create_box(ad->win);
            elm_box_padding_set(vertical_box, 10, 10);
            // Set area size
            evas_object_size_hint_weight_set(vertical_box, EVAS_HINT_EXPAND, EVA
S_HINT_EXPAND);
            evas_object_size_hint_align_set(vertical_box, EVAS_HINT_FILL, EVAS_H
INT_FILL);
            elm_box_pack_end(box, vertical_box);

            { /* child object - indent to how relationship */
                Evas_Object *btn = create_button(vertical_box, "Top");
                evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, 0.0);
                evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0.0);
                elm_box_pack_end(vertical_box, btn);

                btn = create_button(vertical_box, "Center");
                evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, EVAS_HI
NT_EXPAND);
                evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, EVAS_HINT_
FILL);
                elm_box_pack_end(vertical_box, btn);

                btn = create_button(vertical_box, "Bottom");
                evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, 0.0);
                evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0.0);
                elm_box_pack_end(vertical_box, btn);
            }
        }
    }
```

我们将第一个和第三个 Button 的宽度和高度指定为最小值。

我们将第二个 Button 的宽度和高度指定为最大值。

再次运行该示例。您现在将会看到，新添加的三个 Button 为垂直对齐。



5) 相关 API

Evas_Object *elm_box_add(Evas_Object *parent)：一种用于创建 Box 容器的 API。

void elm_box_padding_set(Evas_Object *obj, Evas_Coord horizontal, Evas_Coord vertical)：一种用于指定边距的 API。/ 参数：容器对象、左边距和右边距、上边距和下边距。

void evas_object_size_hint_weight_set(Evas_Object *obj, double x, double y)：一个指定对象所占用空间大小提示的 API。/ 参数：对象、水平尺寸提示和垂直尺寸提示。EVAS_HINT_EXPAND 是一种用于指定尽可能大的对象尺寸的选项。

void evas_object_size_hint_align_set(Evas_Object *obj, double x, double y)：一种用于指定对象尺寸提示的 API。/ 参数：对象、水平尺寸提示和垂直尺寸提示。EVAS_HINT_FILL 是一种用于指定尽可能大的对象尺寸的选项。

void elm_win_resize_object_add(Evas_Object *obj, Evas_Object *subobj)：一个将某个对象的尺寸指定为与另一对象尺寸相同的 API。/ 参数：Window 对象和其尺寸将被更改的对象。

void elm_box_horizontal_set(Evas_Object *obj, Eina_Bool horizontal)：一种用于指定 Box 方向的 API。/ 参数：对象和方向。若传递 EINA_TRUE，会将 Box 的方向设置为水平；若传递 EINA_FALSE，则将 Box 的方向设置为垂直。默认设置为“垂直”。

void elm_box_pack_end(Evas_Object *obj, Evas_Object *subobj)：一种用于向 Box 容器添加新对象的 API。如果对齐方式为水平，将会在屏幕右侧添加新对象。如果对齐方式是垂直，将会在屏幕底部添加新对象。/ 参数：Box 对象和内容对象。

22. 使用 Scroller 小部件创建子页面

当您开发一个商业应用程序时，需要创建多个页面。您可以使用以下方法来实施多个页面。

- 您可使用一个 Toolbar 和若干 Layout 来完成此操作，以便在点击某个选项卡按钮时可以显示/隐藏匹配的 Layout。
- 使用 Scroller 小部件移动 Layout。

在该示例中，我们将学习如何使用 Scroller 小部件在页面之间跳转。我们还将创建一个与第二个页面相关的源文件。

1) 主页面的 UI 任务

在本小节中，我们将通过使用 Naviframe 小部件在主页面上显示一个 Header，并向该主页面添加一个 Button。

创建新的源项目，并将项目名称指定为“Multipage”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并向 appdata 结构添加一个新变量。该变量用于显示标题的 Naviframe 对象。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *nf;
        Evas_Object *label;
} appdata_s;
```

在 create_base_gui() 函数之上添加一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
```

```
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

然后，转至 create_base_gui() 函数并按如下所示修改代码。此代码将创建一个 Naviframe、Box、Label、Button 和 Header。

```
    //eext_object_event_callback_add(ad->win, EEXT_CALLBACK_BACK, win_back_cb,
ad);

    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* Naviframe */
        ad->nf = elm_naviframe_add(ad->conform);
        elm_object_part_content_set(ad->conform, "elm.swallow.content", ad->nf);
```

```
        elm_object_content_set(ad->conform, ad->nf);

        Evas_Object *box = elm_box_add(ad->nf);
        elm_box_padding_set(box, 10 * elm_config_scale_get(), 10 * elm_config_s
cale_get());
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->nf, box);
        evas_object_show(box);

        {
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "Press Button");
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.0);

            /* Button */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Sub Window");
            my_box_pack(box, btn, 1.0, 0.0, -1.0, 0.0);

            /* Header */
            Elm_Object_Item *nf_it;
            nf_it = elm_naviframe_item_push(ad->nf, "Main Window", NULL, NULL,
box, NULL);
            eext_object_event_callback_add(ad->nf, EEXT_CALLBACK_BACK, eext_nav
iframe_back_cb, NULL);
            elm_naviframe_item_pop_cb_set(nf_it, naviframe_pop_cb, ad->win);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

以下代码将会在 Back 按钮被点击时终止该应用程序。问题在于，即使子页面上的 Back 按钮被点击时，也会终止该应用程序。因此，我们现在将给此代码标注注释。

```
eext_object_event_callback_add(ad->win, EEXT_CALLBACK_BACK, win_bac
k_cb, ad);
```

以下代码会在子页面上的 Back 按钮被点击时使该应用程序返回到主页面。

eext_object_event_callback_add(ad->nf, EEXT_CALLBACK_BACK, eext_naviframe_back_cb, NULL);

我们还需要这样一项功能，即在主页面上的 Back 按钮被点击时终止该应用程序。这是因为我们已经为用于调用 win_back_cb callback 函数的代码标注了注释。因此，我们将按如下所示添加代码。当主页面上的 Back 按钮被点击时，将会调用 naviframe_pop_cb 函数。

elm_naviframe_item_pop_cb_set(nf_it, naviframe_pop_cb, ad->win);

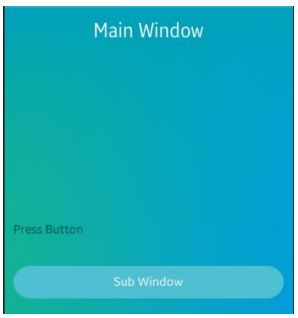
现在，让我们创建 naviframe_pop_cb 回调函数。在 create_base_gui() 函数之上添加一个新函数。

```
static Eina_Bool
naviframe_pop_cb(void *data, Elm_Object_Item *it)
{
        ui_app_exit();
        return EINA_FALSE;
}
```

当主页面上的 Back 按钮被点击时，将会调用上述函数。

ui_app_exit() 是一种用于终止应用程序的函数。

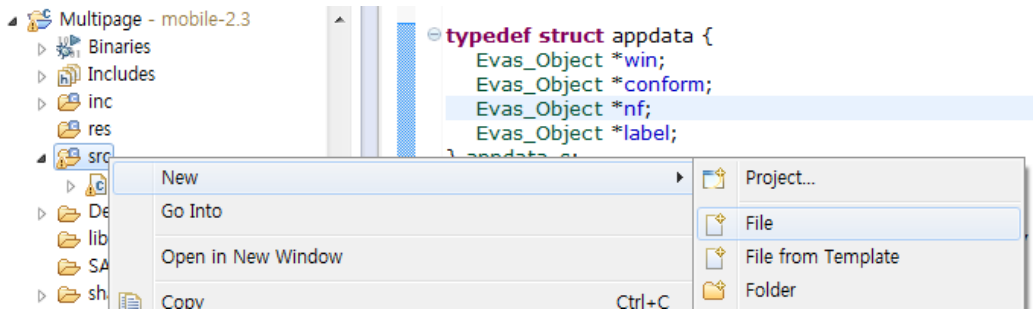
编译和运行该示例。您可以看到一个 Header、Label 和 Button。



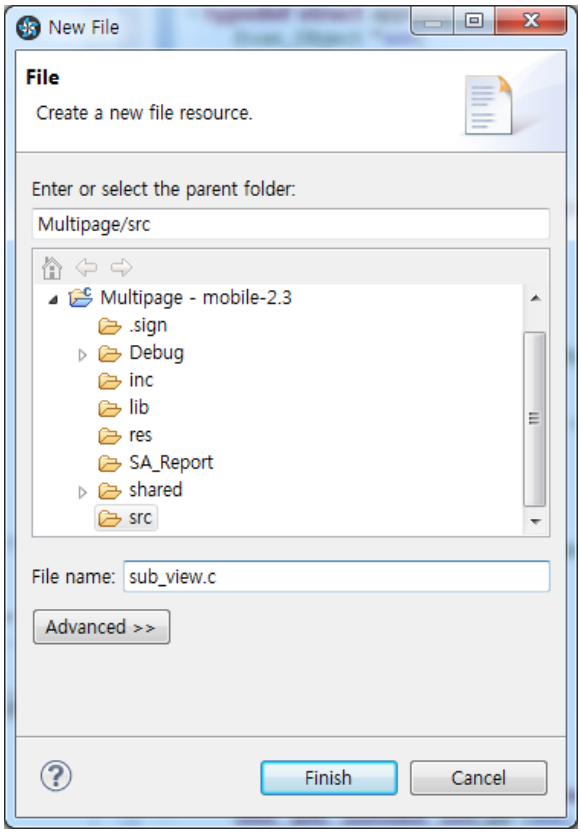
2）创建子页面

在本小节中，我们将实施一项功能，以便当 Button 被点击时显示子页面。您是否将创建子页面的代码添加到主源文件（multipage.c）中，这并不重要。不过，当代码变得越长，将越难以管理。此外，使用多个源文件还可以让您在以后开发另一个应用程序时更方便地使用这些功能。

现在我们将创建一个新的源文件。右键单击 /src 文件夹，然后在快捷方式菜单中选择 [New > File]。



弹出窗口出现后，在文件名称字段中输入“sub_view.c”，并单击 Finish 按钮。然后将会在 /src 文件夹中创建一个新的源文件。



双击新创建的源文件（sub_view.c），并在 Edit 模式下打开它。我们现在将添加用于创建一个子页面和图标 Button 的代码。

```
#include "multipage.h"

static Evas_Object*
create_button_view(Evas_Object *parent)
{
        Evas_Object *btn, *img, *box;

        box = elm_box_add(parent);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        evas_object_size_hint_align_set(box, EVAS_HINT_FILL, EVAS_HINT_FILL);

        /* icon_reorder style */
        btn = elm_button_add(box);
        elm_object_style_set(btn, "icon_reorder");
        evas_object_show(btn);
```

```
        elm_box_pack_end(box, btn);

        return box;
}

void
sub_view_cb(void *data, Evas_Object *obj, void *event_info)
{
        Evas_Object *scroller, *layout;
        Evas_Object *nf = data;

        scroller = elm_scroller_add(nf);
        layout = create_button_view(scroller);
        elm_object_content_set(scroller, layout);

        elm_naviframe_item_push(nf, "Sub Window", NULL, NULL, scroller, NULL);
}
```

由于需要将该子页面链接到第一个页面，因此我们将声明主标题文件 (multipage.h)。

create_button_view() 函数将创建一个 Box 容器并在该 Box 容器之上添加一个 Button 小部件。

sub_view_cb() 函数将通过创建一个 Scroller 和 Layout 来显示第二个页面。

elm_scroller_add() 是一种用于创建新 Scroller 的 API。

elm_object_content_set() 是一种用于指定容器内容的 API。如果将一个 Scroller 传递给第一个参数，并将一个 layout 传递给第二个参数，那么将会在屏幕上显示该 layout。

更改该 Header 的标题文本，并使用 elm_naviframe_item_push() 函数将此 Scroller 指定为 Naviframe 的内容。

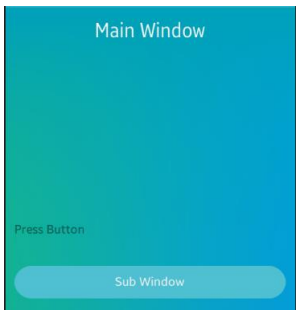
当第一个页面上的 Button 被点击时，需要调用 sub_view_cb() 函数。因此，我们必须在标题文件中声明一个函数。打开 /inc 文件夹，然后双击“multipage.h”文件。该文件打开时，在文件的末端声明子页面创建函数。

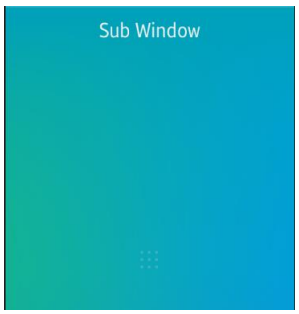
```
#if !defined(PACKAGE)
#define PACKAGE "org.tizen.multipage"
#endif

void sub_view_cb(void *data, Evas_Object *obj, void *event_info);

#endif /* __multipage_H__ */
```

现在可以在主页面上调用该函数了。转至 Multipage.c 文件，然后向 create_base_gui() 函数的 Button 创建代码添加一行新代码。此新代码将会在主页面上的 Button 被点击时调用子页面创建函数。

```
        /* Button */
        Evas_Object *btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Sub Window");
        evas_object_smart_callback_add(btn, "clicked", sub_view_cb, ad->nf);
        my_box_pack(box, btn, 1.0, 0.0, -1.0, 0.0);
```

再次运行该示例。当您点击屏幕上的 Button 时，将会显示子页面，而且您可以在屏幕中央看到一个图标 Button。若要返回到主页面，请点击 Back 按钮。





3) 通过点击一个 Button 跳转到主页面

在本小节中，我们将实施一项功能，以便当子页面上的 Button 被点击时使应用程序跳转到主页面。要做到这一点，我们需要调用 elm_naviframe_item_pop() 函数。

转至 sub_view.c 文件中的 sub_view_cb() 函数，然后修改其代码。将 Naviframe 传递给 create_button_view() 函数，

```
        scroller = elm_scroller_add(nf);
        layout = create_button_view(scroller, nf);
        //layout = create_button_view(scroller);
        elm_object_content_set(scroller, layout);

        elm_naviframe_item_push(nf, "Sub Window", NULL, NULL, scroller, NULL);
}
```

然后向 create_button_view() 函数添加新代码。

```
static Evas_Object*
create_button_view(Evas_Object *parent, Evas_Object *nf)
{
        Evas_Object *btn, *img, *box;

        box = elm_box_add(parent);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        evas_object_size_hint_align_set(box, EVAS_HINT_FILL, EVAS_HINT_FILL);

        /* icon_reorder style */
        btn = elm_button_add(box);
        elm_object_style_set(btn, "icon_reorder");
        evas_object_smart_callback_add(btn, "clicked", btn_back_cb, nf);
        evas_object_show(btn);
        ~
```

NaviFrame 对象将被传递给该函数的参数。我们已将 Button 回调函数指定为 btn_back_cb，并将 Naviframe 传递给了用户数据。

最后，在 create_button_view() 函数之上添加 Button 回调函数。

```
void
btn_back_cb(void *data, Evas_Object *obj, void *event_info)
{
        Evas_Object *nf = data;
        elm_naviframe_item_pop(nf);
}
```

elm_naviframe_item_pop() 是一种用于移到 Naviframe 堆栈中所积聚的页面列表顶端的命令。换言之，这是移至主页面的命令。

再次运行该示例，转至子页面，然后点击图标 Button。您将进入主页面。

若点击子页面上的 Back 按钮，您将直接进入主页面，而点击主页面上的 Back 按钮，则会使该应用程序屏幕消失。这并不表示该应用程序已被关闭；而是表示该应用程序已切换至后台模式。

4) 相关 API

eext_object_event_callback_add(ad->win, EEXT_CALLBACK_BACK, win_back_cb, ad)：用于在 Back 按钮被点击时终止应用程序的代码。为了在子页面上的 Back 按钮被点击时该应用程序不会终止，您必须对此代码标注注释。

eext_object_event_callback_add(ad->nf, EEXT_CALLBACK_BACK, eext_naviframe_back_cb, NULL)：此代码将会在子页面上的 Back 按钮被点击时使应用程序返回到主页面。

elm_naviframe_item_pop_cb_set(nf_it, naviframe_pop_cb, ad->win)：此代码将主页面上的 Back 按钮被点击时执行的回调函数指定为 naviframe_pop_cb。

ui_app_exit()：一种用于终止应用程序的 API。

Evas_Object *elm_naviframe_add(Evas_Object *parent)：一种用于创建新 Scroller 的 API。

void elm_object_content_set(Evas_Object *obj, Evas_Object *content)：一种用于为容器指定内容的 API。/ 参数：容器对象和内容对象。

Elm_Object_Item *elm_naviframe_item_push(Evas_Object *obj, const char *title_label, Evas_Object *prev_btn, Evas_Object *next_btn, Evas_Object *content, const char *item_style)：一种用于指定 Naviframe 的 Header 标题文本和内容的 API。/ 参数：Naviframe 对象、标题文本、用于移至上一项目的 Button 的对象、用于移至下一项目的 Button 的对象、内容和用户数据。

Evas_Object *elm_naviframe_item_pop(Evas_Object *obj)：一种用于移到 Naviframe 堆栈中所积聚的页面列表顶端的 API。/ 参数：Naviframe 对象。

23. 使用字符串

字符串转换或字符串搜索是一个在开发应用程序时常用到的功能。在该示例中，我们将学习如何使用用 C 语言编写的基本 API 查找字符串的位置以及将字符串转换为数字。

1) 复制字符串

创建新的源项目，并将项目名称指定为“StringEx”。创建源项目之后，通过以下方式添加新代码：打开 src 文件夹中的源文件 (~.c)，并转至 create_base_gui() 函数。

```
        /* Label*/
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "Hello EFL");
        evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_
EXPAND);
        elm_object_content_set(ad->conform, ad->label);
        elm_label_line_wrap_set(ad->label, EINA_TRUE);

        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        show_string_result(ad->label);
```

elm_label_line_wrap_set() 是一种用于为 Label 小部件指定自动换行的 API。

show_string_result() 是一种用于显示字符串转换结果的函数。现在，让我们开始编译该示例。

在 create_base_gui() 函数之上添加一个新函数。

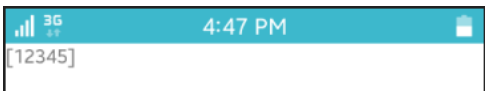
```
static void
show_string_result(Evas_Object *label)
{
```

```
        char buf[PATH_MAX], str1[100];
        char *str2;
        strcpy(str1, "12345");
        sprintf(buf, "[%s]", str1);

        elm_object_text_set(label, buf);
}
```

strcpy() 是一种用于复制字符串的 API。第一个参数指示用于存储字符串的内存地址；而第二个参数指示与原始字符串有关的数据。在此例中，我们将在一个名为“str1”的字符串变量中输入文本“12345”。

sprintf() 是一种用于创建特定格式类型的新字符串的 API。第一个参数指示用于存储字符串的内存地址，第二个参数指示格式类型，而第三个参数和后面的参数则指示要分配给此格式的数据。

编译和运行该示例。我们输出了以方括号括起来的原始字符串。



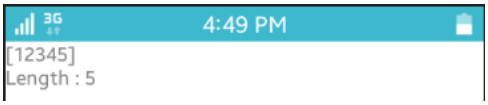
2) 请求字符串的长度

在本小节中，我们将请求存储在 str1 变量中的一个字符串的长度。在 show_string_result() 函数的结尾添加新代码。

```
      strcpy(str1, "12345");
      sprintf(buf, "[%s]", str1);

        int length = strlen(str1);
        sprintf(buf, "%s<br>Length : %d", buf, length);

        elm_object_text_set(label, buf);
```

strlen() 是一种用于请求并在之后返回字符串长度的 API。str1 是一个 char 数组，因此它拥有固定长度。在这种情况下，将会返回从字符串的开头到结束符号（\0）之前的长度。

再次运行该示例。将显示该字符串的长度。



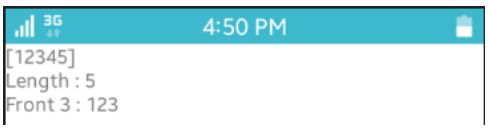
3）从字符串的开头处开始提取特定长度的字符

在本小节中，我们将从 str1 变量中提取前三个字符。在 show_string_result() 函数的结尾添加新代码。

```
        sprintf(buf, "%s<br>Length : %d", buf, length);

        str2 = eina_stringshare_add_length(str1, 3);
        sprintf(buf, "%s<br>Front 3 : %s", buf, str2);

        elm_object_text_set(label, buf);
```

eina_stringshare_add_length() 是一种用于从字符串的开头处开始提取特定长度字符的 API。第一个参数指示与原始字符串有关的数据，而第二个参数指示要提取的字符长度。将返回提取出来的字符串。

再次运行该示例。将从字符串“12345”的开头处提取字符“123”。



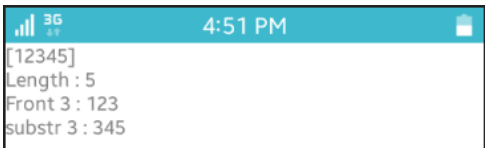
4）提取字符串的特定部分

在本小节中，我们将学习如何通过指定提取的起始点从字符串中提取特定长度的字符。在 show_string_result() 函数的结尾添加新代码。

```
        sprintf(buf, "%s<br>Front 3 : %s", buf, str2);

        str2 = eina_stringshare_add_length(str1 + 2, 3);
        sprintf(buf, "%s<br>substr 3 : %s", buf, str2);

        elm_object_text_set(label, buf);
```

我们将“str1 + 2”传递给了 eina_stringshare_add_length() 函数的第一个参数。这样做最终将传递字符“345”。

再次运行该示例。将从字符串“12345”的第三个字符位置开始提取三个字符。



5）将字符串转换为数值

若要将一个字符串转化为数值，您可以使用 atoi() 函数。在 show_string_result() 函数的结尾添加新代码。

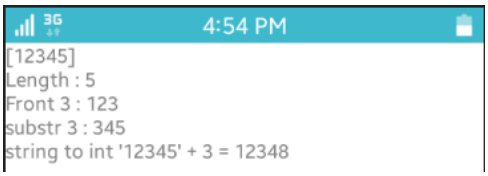
```
        sprintf(buf, "%s<br>substr 3 : %s", buf, str2);

        int i = atoi(str1);
        sprintf(buf, "%s<br>string to int '%s' + 3 =  %d", buf, str1, i + 3);

        elm_object_text_set(label, buf);
```

atoi() 是一个将字符串转换为 int 类型的 API。

atol() 是一个将字符串转换为 long 类型的 API。

atof 是一个将字符串转换为 float 类型的 API。

再次运行该示例。字符串“12345”将被转换成一个数字，屏幕上将会显示该数字加 3 之后的结果。



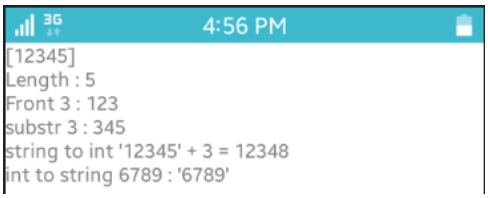
6) 将数字转换为字符串

若要将一个数字转化为字符串，您可以使用 sprintf() 函数。在 show_string_result() 函数的结尾添加新代码。

```
        sprintf(buf, "%s<br>string to int '%s' : %d", buf, str1, i);

        char str3[100];
        i = 6789;
        sprintf(str3, "%d", i);
        sprintf(buf, "%s<br>int to string %d : '%s'", buf, i, str3);

        elm_object_text_set(label, buf);
```

下面是在字符串类型格式声明中通常会使用的符号：
- %s：替换为字符串。
- %d：替换为一个数字，如 int 或 long 数字。
- %c：替换为一个字符。
- %f：替换为一个实数，如 float 或 double 数字。

再次运行该示例。数字“6789”将被转换为一个字符串。



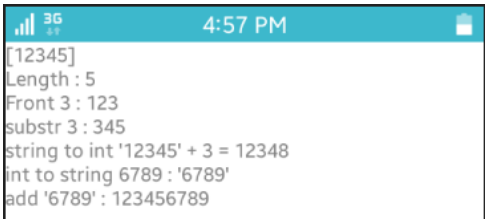
7）合并字符串

在本小节中，我们将学习如何将两个字符串合并成一个字符串。在 show_string_result() 函数的结尾添加新代码。

```
        sprintf(buf, "%s<br>int to string %d : '%s'", buf, i, str3);

        strcat(str1, str3);
        sprintf(buf, "%s<br>add '%s' : %s", buf, str3, str1);

        elm_object_text_set(label, buf);
```

strcat() 是一种用于合并两个字符串的函数。该函数会将第二个字符串附加到第一个参数的末尾。这并不是创建一个新字符串，而是将一个新字符串附加到现有字符串后面。

再次运行该示例。“6789”被附加到“12345”的后面，得到“123456789”。



8）查找字符串的位置

在本小节中，我们将学习如何查找字符串的位置。在 show_string_result() 函数的结尾添加新代码。

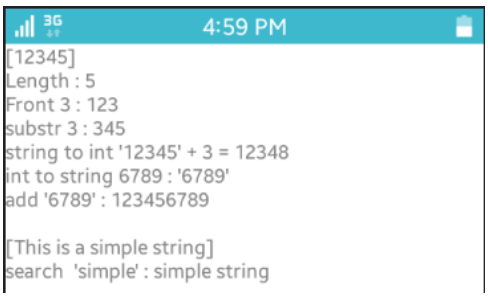
```
        sprintf(buf, "%s<br>add '%s' : %s", buf, str3, str1);

        str1[0] = '\0';
        str3[0] = '\0';

        strcpy(str1, "This is a simple string");
        sprintf(buf, "%s<br><br>[%s]", buf, str1);

        elm_object_text_set(label, buf);
        str2 = strstr( str1, "simple" );
        sprintf(buf, "%s<br>search   'simple' : %s", buf, str2);

        elm_object_text_set(label, buf);
```

“\0”是指示一个字符串结尾的结束符号。如果在一个 char 数组的第一个字符位置输入该符号，则会将该字符串初始化为空字符串。

strstr() 是一种用于查找字符串中特定字符串的位置的 API。该函数不会返回索引编号；它将会返回用于标识该字符串起始位置的指针。换言之，它将会返回一个字符串。

若要查找某个特定字符在一个字符串中的起始位置，您可以使用 strchr(char*, int) 函数。

再次运行该示例。将会显示跟在字符串“simple”的起始位置后面的字符串。



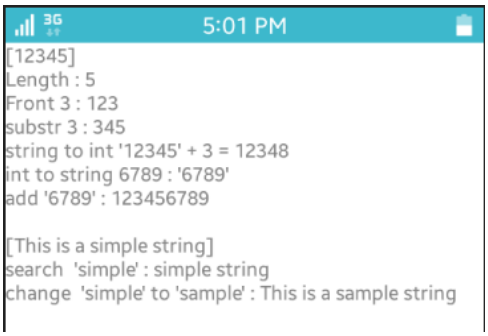
9) 复制字符串的特定字符段

在本小节中，我们将把字符串“simple”更改为“sample”。要做到这一点，我们需要使用 strncpy() 函数。在 show_string_result() 函数的结尾添加新代码。

```
        sprintf(buf, "%s<br>search   'simple' : %s", buf, str2);

        strncpy( str2, "sample", 6 );
        sprintf(buf, "%s<br>change   'simple' to 'sample' : %s", buf, str1);

        elm_object_text_set(label, buf);
```

strncpy() 是一种用于复制字符串的特定字符段的函数。str2 指示字符串“simple”的起始位置。因此，“simple”最后将被更改为“sample”。

再次运行该示例。您现在将会看到“simple”已被更改为“sample”。



10) 相关 API

char *strcpy (char *dest, char *src)：一种用于复制字符串的 API。/ 参数：用于存储字符串的内存地址和原始字符串的相关数据。

int sprintf (char *s, char *format, ...)：一种用于创建特定格式类型的新字符串的 API。/ 参数：用于存储字符串的内存地址、格式类型以及（从第三个参数开始）要分配给该格式的数据。

size_t strlen (char *s)：一种用于请求并在之后返回字符串长度的 API。在字符串类型格式声明中，%s 是一个替换为字符串的符号。%d 是一个替换为数字（如 int 或 long）的符号。%c 是一个替换为字符的符号。%f 是一个替换为实数（如 float 或 double）的符号。

Eina_Stringshare *eina_stringshare_add_length(const char *str, unsigned int slen)：一种用于从字符串的开头处开始提取特定长度字符的 API。/ 参数：原始字符串的相关数据和将要提取的长度。/ 返回：提取的字符串。

int atoi (char *nptr)：一个将字符串转换为 int 类型的 API。

long int atol (char *nptr)：一个将字符串转换为 long 类型的 API。

double atof (char *nptr)：一个将字符串转换为 float 类型的 API。

char *strcat (char *__dest, char *__src)：一种用于合并两个字符串的 API。该函数会将第二个字符串附加到第一个字符串的末尾。在这种情况下，将不会创建新的字符串，而是将新的字符串附加到现有字符串的后面。

char *strstr (char *haystack, char *needle)：一种用于查找字符串中特定字符串的位置的 API。该函数不会返回索引编号；它将会返回作为该字符串起始位置的指针。因此，它将会返回一个字符串。

char *strchr (char *s, int c)：一种用于在字符串中搜索某个特定字符的起始位置的 API。

char *strncpy (char *dest, char *src, size_t n)：一种用于复制字符串的特定字符段的 API。

24. 字符串结构“Eina_Strbuf”

有时候当您在使用用 C 语言编写的 API 处理字符串时会感觉走进了死胡同，对于这种情况，EFL 为您提供了一个名为“Eina_Strbuf”的字符串结构。在此例中，我们将学习如何删除字符串中的一部分内容、将字符串更改为另一字符串，以及将新字符串插入现有字符串的中间。

1) 在 Eina_Strbuf 中输入字符串

创建新的源项目，并将项目名称指定为“EinaStrbufEx”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，移至 create_base_gui() 函数并添加新代码。

```
        /* Label*/
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "Hello EFL");
        evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_
EXPAND);
        elm_object_content_set(ad->conform, ad->label);
        elm_label_line_wrap_set(ad->label, EINA_TRUE);

        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        show_eina_strbuf_result(ad->label);
```

elm_label_line_wrap_set() 是一种用于为 Label 小部件指定自动换行的 API。

show_eina_strbuf_result() 函数用于在屏幕上显示使用 Eina_Strbuf 的结果。现在，让我们开始编译该示例。

在 create_base_gui() 函数之上创建一个新函数。

```
static void
show_eina_strbuf_result(Evas_Object *label)
{
        Eina_Strbuf *strline, *strbuf;
        /* Create Eina_Strbuf */
        strbuf = eina_strbuf_new();
        /* Addend string */
        eina_strbuf_append(strbuf, "Hello Tizen");

        elm_object_text_set(label, eina_strbuf_string_get(strbuf));
        /* Free memory */
        eina_strbuf_free(strbuf);
}
```

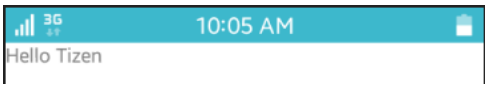
eina_strbuf_new() 是一种用于创建新 Eina_Strbuf 对象的 API。

eina_strbuf_append() 是一种用于向 Eina_Strbuf 添加字符串的 API。

eina_strbuf_string_get() 是一种用于请求存储在 Eina_Strbuf 中的字符串的 API。

eina_strbuf_free() 是一种用于删除存储在 Eina_Strbuf 中的数据的 API。

构建并运行该示例。



2) 按指定格式添加字符串

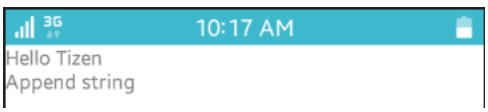
要按指定格式添加新字符串，您需要使用 eina_strbuf_append_printf() 函数。在 show_eina_strbuf_result() 函数中间添加新代码。

```
        eina_strbuf_append(strbuf, "Hello Tizen");

        /* Reset string */
        strline = eina_strbuf_new();
        eina_strbuf_append(strline, "Append string");
        eina_strbuf_append_printf(strbuf, "<br>%s", eina_strbuf_string_get(strl
ine) );

        elm_object_text_set(label, eina_strbuf_string_get(strbuf));
        /* Free memory */
        eina_strbuf_free(strline);
        eina_strbuf_free(strbuf);
```

我们使用 eina_strbuf_new() 函数创建了一个新的 Eina_Strbuf 对象，并使用 eina_strbuf_append() 函数输入了一个字符串。

eina_strbuf_append_printf() 是一种用于通过指定格式语句来添加新字符串的 API。第一个参数指示 Eina_Strbuf 对象，第二个参数指示格式类型，第三个参数和后续参数则指示将要分配给此格式的数据。

再次运行该示例。屏幕上分两行显示一个字符串。



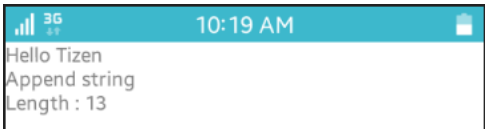
3) 请求字符串的长度

要请求存储在 Eina_Strbuf 中的某个字符串的长度，您需要使用 eina_strbuf_length_get() 函数。在 show_eina_strbuf_result() 函数的结尾添加新代码。

```
        eina_strbuf_append_printf(strbuf, "<br>%s", eina_strbuf_string_get(strline) );

        /* Length of string */
        eina_strbuf_append_printf(strbuf, "<br>Length : %d",
                        eina_strbuf_length_get(strline) );

        elm_object_text_set(label, eina_strbuf_string_get(strbuf));
```

eina_strbuf_length_get() 是一种用于返回存储在 Eina_Strbuf 中的字符串长度的 API。

再次运行该示例。屏幕上显示字符串“Append string”的长度：13。



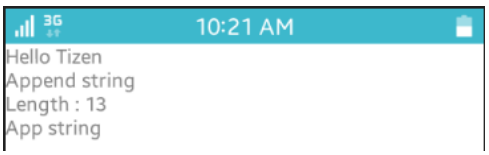
4) 删除字符串的一部分

要删除某个字符串中的一部分内容，您需要使用 eina_strbuf_remove() 函数。在 show_eina_strbuf_result() 函数的结尾添加新代码。

```
        eina_strbuf_append_printf(strbuf, "<br>Length : %d", eina_strbuf_length_
get(strline) );

        /* Remove part of string */
        eina_strbuf_remove(strline, 3, 6);
        eina_strbuf_append_printf(strbuf, "<br>%s", eina_strbuf_string_get(strl
ine) );

        elm_object_text_set(label, eina_strbuf_string_get(strbuf));
```

eina_strbuf_remove() 是一种用于仅删除字符串部分内容的 API。第一个参数指示 Eina_Strbuf 对象，第二个参数指示删除内容的起始位置，第三个参数指示删除内容的结束位置。例如，输入“6”则删除第六个字符之前的所有字符。

再次运行该示例。屏幕上将删除索引数字 3 到 5（第四个到第六个字符）。



5) 替换字符串

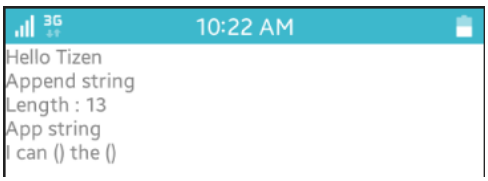
要将某个字符串替换为另一字符串，您需要使用 eina_strbuf_replace() 函数。在 show_eina_strbuf_result() 函数的结尾添加新代码。

```
        eina_strbuf_append_printf(strbuf, "<br>%s", eina_strbuf_string_get(strli
ne) );

        /* Replace string */
        eina_strbuf_reset(strline);
        eina_strbuf_append(strline, "I () () the ()");
        eina_strbuf_replace(strline, "()", "can", 1);
        eina_strbuf_append_printf(strbuf, "<br>%s", eina_strbuf_string_get(strl
ine) );

        elm_object_text_set(label, eina_strbuf_string_get(strbuf));
```

eina_strbuf_reset() 是一种用于重置存储在 Eina_Strbuf 中的字符串的 API。

eina_strbuf_replace() 函数用于将字符串更改为另一字符串。第一个参数指示 Eina_Strbuf 对象；第二个参数指示要更改的字符串；第三个参数指示要将现有字符串改成的目标字符串；第四个参数指示更改次数。

再次运行该示例。第一个“()”更改为“can”。



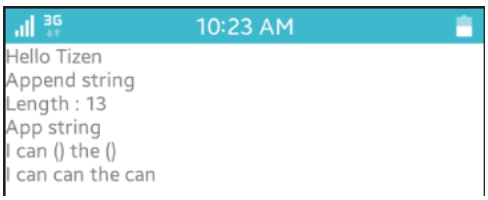
6) 替换所有相同字符串

在本小节中，我们将学习如何替换所有相同字符串。在 show_eina_strbuf_result() 函数的结尾添加新代码。

```
        eina_strbuf_append_printf(strbuf, "<br>%s", eina_strbuf_string_get(strline) );

        /* Replace all */
        eina_strbuf_replace_all(strline, "()", "can");
        eina_strbuf_append_printf(strbuf, "<br>%s", eina_strbuf_string_get(strline) );

        elm_object_text_set(label, eina_strbuf_string_get(strbuf));
```

eina_strbuf_replace_all() 是一种用于替换所有相同字符串的 API。第一个参数指示 Eina_Strbuf 对象；第二个参数指示要更改的字符串；第三个参数指示要将现有字符串改成的目标字符串。

再次运行该示例。所有“()”均更改为“can”。



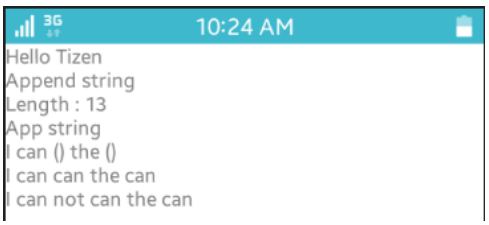
7）将字符串插入另一字符串的中间

在本小节中，我们将学习如何将字符串插入另一字符串的中间。在 show_eina_strbuf_result() 函数的结尾添加新代码。

```
        eina_strbuf_append_printf(strbuf, "<br>%s", eina_strbuf_string_get(strline) );

        /* Insert string */
        eina_strbuf_insert(strline, " not", 5);
        eina_strbuf_append_printf(strbuf, "<br>%s", eina_strbuf_string_get(strline) );

        elm_object_text_set(label, eina_strbuf_string_get(strbuf));
```

eina_strbuf_insert() 是一种用于将字符串插入另一字符串的特定位置的 API。第一个参数指示 Eina_Strbuf 对象，第二个参数指示要插入的字符串，第三个参数指示插入位置。

再次运行该示例。字符串“not”被插入到索引编号 5（第六个字符）的位置。



8）相关 API

Eina_Strbuf *eina_strbuf_new(void)：一种用于创建新 Eina_Strbuf 对象的 API。

Eina_Bool eina_strbuf_append(Eina_Strbuf *buf, char *str)：一种用于向 Eina_Strbuf 添加字符串的 API。

char *eina_strbuf_string_get(Eina_Strbuf *buf)：一种用于请求存储在 Eina_Strbuf 中的字符串的 API。

void eina_strbuf_free(Eina_Strbuf *buf)：一种用于删除存储在 Eina_Strbuf 中的数据的 API。

Eina_Bool eina_strbuf_append_printf(Eina_Strbuf *buf, char *fmt, ...)：一种用于通过指定格式语句来添加新字符串的 API。/ 参数：Eina_Strbuf 对象、格式类型，以及从第三个参数起均要分配给此格式的数据。

size_t eina_strbuf_length_get(Eina_Strbuf *buf)：一种用于返回存储在 Eina_Strbuf 中的字符串的 API。

Eina_Bool eina_strbuf_remove(Eina_Strbuf *buf, size_t start, size_t end)：一种用于仅删除字符串部分内容的 API。/ 参数：Eina_Strbuf 对象、删除内容的起始位置以及删除内容的结束位置。

void eina_strbuf_reset(Eina_Strbuf *buf)：一种用于重置存储在 Eina_Strbuf 中的字符串的 API。

Eina_Bool eina_strbuf_replace(Eina_Strbuf *buf, char *str, char *with, unsigned int n)：一种用于将字符串更改为另一字符串的 API。/ 参数：Eina_Strbuf 对象、要更改的字符串、要将现有字符串改成的目标字符串以及更改次数。

int eina_strbuf_replace_all(Eina_Strbuf *buf, char *str, char *with)：一种用于替换所有相同字符串的 API。/ 参数：Eina_Strbuf 对象、要更改的字符串以及要将现有字符串改成的目标字符串。

Eina_Bool eina_strbuf_insert(Eina_Strbuf *buf, char *str, size_t pos)：一种用于将某个字符串插入另一字符串的特定位置的 API。/ 参数：Eina_Strbuf 对象、要插入的字符串以及插入位置。

25. 数组结构 Eina_List

当您在开发应用程序时，有时候需要管理数组中的多个字符串或用户数据。EFL 提供一种名为“Eina_List”的数组结构。现在我们将通过示例学习如何使用此内容。

1) 创建 List 小部件

创建新的源项目，并将项目名称指定为“EinaListEx”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并添加新代码。将新变量添加到 appdata 结构，定义新结构，并将 10 个字符串存储到 char* 数组。

```
typedef struct appdata {
    Evas_Object *win;
    Evas_Object *conform;
    Evas_Object *label;
    Evas_Object *list;
    Evas_Object *button;
    Eina_List *data_list;
} appdata_s;

typedef struct {
    appdata_s  *ad;
    char       *data;
    int         id;
} itemdata_s;

/* List */
const char *items[] = {
        "Seoul", "Tokyo", "New York", "London", "Beijing",
        "Moscow", "Singapore", "Busan", "Hong Kong", "Paris",
        NULL
};
```

list 是 List 小部件的变量。

Button 是 Delete 按钮的变量。

data_list 是数组结构 Eina_List 的变量。

Itemdata_s 是 List 小部件的事件数据的结构。现在我们将在 ad 中输入应用程序数据，在 data 中输入用户选择的字符串，并在 id 中输入所选项目数。

Items[] 会存储 10 个将被添加到 List 小部件中的字符串。

转至 create_base_gui() 函数并添加新代码。此代码可创建 Frame、Table、Button 以及 List 小部件。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* A frame surrounding the whole UI, for outer padding */
        Evas_Object *frame;
        frame = elm_frame_add(ad->conform);
        evas_object_size_hint_weight_set(frame, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
        evas_object_size_hint_align_set(frame, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(ad->conform, frame);
        evas_object_show(frame);

        /* A table to layout our objects */
        Evas_Object *table;
        table = elm_table_add(frame);
        evas_object_size_hint_weight_set(table, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
        evas_object_size_hint_align_set(table, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(frame, table);
        evas_object_show(table);
        /* Set inner padding */
        elm_table_padding_set(table, 5 * elm_scale_get(), 5 * elm_scale_get());

        {
            /* Label */
```

```
            ad->label = elm_label_add(table);
            elm_object_text_set(ad->label, "Please select an item");
            evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, 0);
            evas_object_size_hint_align_set(ad->label, EVAS_HINT_FILL, 0.5);
            elm_table_pack(table, ad->label, 0, 0, 1, 1);
            evas_object_show(ad->label);

            /* Minus button */
            ad->button = elm_button_add(ad->conform);
            elm_object_text_set(ad->button, "Remove");
            //evas_object_smart_callback_add(ad->button, "clicked", btn_clicked
_cb, ad);
            evas_object_size_hint_weight_set(ad->button, EVAS_HINT_EXPAND, 0);
            evas_object_size_hint_align_set(ad->button, EVAS_HINT_FILL, 0.5);
            elm_table_pack(table, ad->button, 1, 0, 1, 1);
            evas_object_show(ad->button);

            /* List view */
            ad->list = elm_list_add(ad->conform);
            elm_list_mode_set(ad->list, ELM_LIST_COMPRESS);
            evas_object_size_hint_weight_set(ad->list, EVAS_HINT_EXPAND, EVAS_H
INT_EXPAND);
            evas_object_size_hint_align_set(ad->list, EVAS_HINT_FILL, EVAS_HINT
_FILL);
            elm_table_pack(table, ad->list, 0, 1, 2, 1);
        }
    }

    /* Let's add some elements to our lists */
    populate_list(ad);

    /* Go should be called before show for proper display */
    elm_list_go(ad->list);
    evas_object_show(ad->list);

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

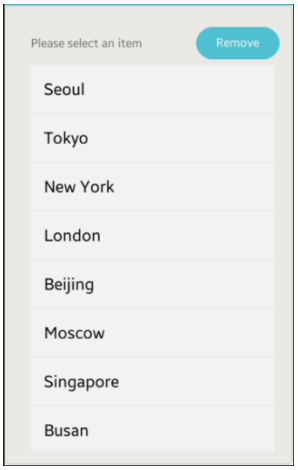
如果将 Table 添加到 Frame，则可指定外边距。

populate_list() 是一种用于将 10 个文本项目添加到 List 小部件的函数。现在我们将创建一个这样的函数。在 create_base_gui() 函数之上创建一个新函数。

```
static void
populate_list(appdata_s *ad)
{
    /* Now, let's create an Eina_List and add some data items to it */
    for (unsigned i = 0; items[i]; i++)
    {
        elm_list_item_append(ad->list, items[i], NULL, NULL, NULL, NULL);
    }

    elm_list_go(ad->list);
}
```

此代码可将 10 个文本项目添加到 List。请参考 ListEx 示例，了解对各个代码的解释。

构建并运行该示例。此时创建了一个 Label 小部件、一个 Button 小部件以及一个 List 小部件。同时，10 个文本项目也被添加至 List 小部件。



2) 在 Eina_List 中输入和输出数据

在本小节中，我们将学习如何在 Eina_List 中输入和输出字符串数据。按如下所示，修改 populate_list() 函数的代码：

此代码可将所有项目数据存储至 Itemdata_s 结构变量，将数据存储至 Eina_List，然后将存储在 Eina_List 中的字符串添加至 List 小部件。

eina_list_append(Eina_List*, void*) 是一种用于向 Eina_List 添加新项目的 API。由于此 API 会返回 Eina_List 的指针，因此必须将指针地址存储到 Eina_List 变量中。

eina_list_nth(Eina_List*, unsigned int) 是一种用于返回存储在特定位置的项目数据的 API。

eina_list_count(Eina_List*) 是一种用于返回存储在 Eina_List 中的项目数的 API。

再次运行该示例。此时会出现一个与前一个列表完全相同的列表。但此时，数据已经存储到一个全局变量中，因此可以在任意位置使用这些数据。

3) 在屏幕上显示关于所选项目的数据

在本小节中，我们将实施一项功能，以便在用户选择 List 小部件项目时在 Label 小部件中显示该项目的文本。按如下所示，修改 populate_list() 函数的代码：

```
populate_list(appdata_s *ad)
{
    for (unsigned i = 0; items[i]; i++)
    {
        itemdata_s *idata = calloc(1, sizeof(itemdata_s));
        idata->ad = ad;
        idata->data = strdup(items[i]);
        idata->id = i + 1;
        ad->data_list = eina_list_append(ad->data_list, idata);
        //eli = elm_list_item_append(ad->list, items[i], NULL, NULL, NULL, NULL);

        itemdata_s *itemdata = eina_list_nth(ad->data_list, i);
        //elm_list_item_append(ad->list, itemdata->data, NULL, NULL, NULL, NULL);
```

```
        Elm_List_Item *eli;
        eli = elm_list_item_append(ad->list, itemdata->data, NULL, NULL, list_i
tem_clicked_cb, idata);
    }

    elm_list_go(ad->list);
}
```

此代码可在项目被添加到 List 小部件时指定项目选择回调函数的名称，并传递项目数据。

我们已将项目选择回调函数的名称指定为 list_item_clicked。

现在，我们将定义此回调函数。在 populate_list() 函数之上添加一个新函数。

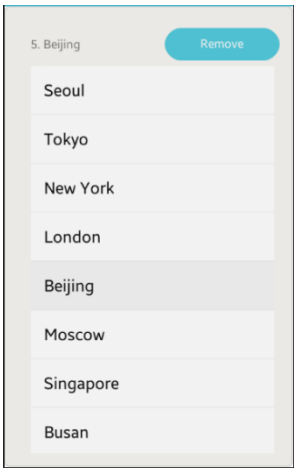
```
static void
list_item_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
    itemdata_s *idata = data;
    char buf[256];

    snprintf(buf, 256, "%d. %s", idata->id, idata->data);
    elm_object_text_set(idata->ad->label, buf);
}
```

list_item_clicked_cb() 是 List 项目选择事件函数。此函数将在 Label 中显示所选项目的文本，

并使用 sprintf() 函数将项目编号和文本转换为单个字符串。

同时，此函数还会使用 elm_object_text_set() 函数在 Label 小部件中显示项目数据。

再次运行该示例。选择一个 List 小部件项目，随后您将会看到在 Label 小部件中显示了该项目的项目编号和文本。



4) 删除数据

在本小节中，我们将实施一项功能，以便在用户点击 Button 时删除当前选定的项目。向 create_base_gui() 函数的用于创建 Button 的代码部分添加一行新代码。

```
        /* Minus button */
        ad->button = elm_button_add(ad->conform);
        elm_object_text_set(ad->button, "Remove");
        evas_object_smart_callback_add(ad->button, "clicked", btn_clicked_cb, a
d);
        evas_object_size_hint_weight_set(ad->button, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(ad->button, EVAS_HINT_FILL, 0.5);
        elm_table_pack(table, ad->button, 1, 0, 1, 1);
        evas_object_show(ad->button);
```

我们已将 Button 回调函数的名称指定为 btn_clicked_cb。

现在，我们需要定义此回调函数。在 create_base_gui() 函数之上创建一个新函数。

```
static void
btn_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
```

```
    appdata_s *ad = data;
    Elm_List_Item *it;

    if (!elm_list_items_get(ad->list))
    {
        elm_object_text_set(ad->button, "Remove");
        populate_list(ad);
        return;
    }

    it = elm_list_selected_item_get(ad->list);
    if (!it)
    {
        elm_object_text_set(ad->label, "No item selected");
        return;
    }

    /* Delete widget item, this will call item_del_cb */
    elm_object_item_del(it);

    /* If no more elements, offer to repopulate list */
    if (!ad->data_list)
    {
        elm_object_text_set(ad->button, "Populate");
        return;
    }
}
```

如果所选项目不存在于 List 小部件中，此代码会在 Button 小部件中显示“Remove”文本并退出此函数。

如果所选项目存在于 List 小部件中，此代码将删除该项目，并在 Button 小部件中显示“Populate”文本。

当删除 List 小部件的项目时，存储在数组结构中的该项目的数据必须随该项目一同被删除。在 populate_list() 函数的结尾添加一行新代码。

```
        itemdata_s *itemdata = eina_list_nth(ad->data_list, i);
        Elm_List_Item *eli;
        eli = elm_list_item_append(ad->list, itemdata->data, NULL, NULL, list_it
em_clicked_cb, idata);
        elm_object_item_del_cb_set(eli, item_del_cb);
```

```
    }

    elm_list_go(ad->list);
}
```

elm_object_item_del_cb_set() 是一种用于指定项目删除回调函数名称的 API。

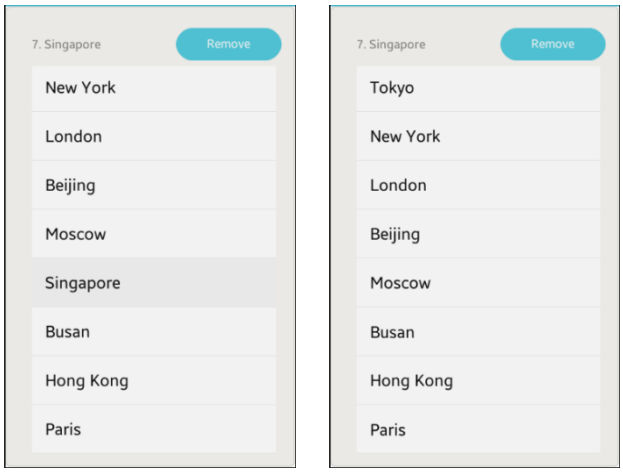
我们已将回调函数指定为 item_del_cb。在 elm_object_item_del_cb_set() 函数之上添加回调函数。

```
static void
item_del_cb(void *data, Evas_Object *obj, void *event_info)
{
    /* Those are the arguments */
    Elm_Widget_Item *it = event_info;
    itemdata_s *idata = data;
    (void) it;

    /* Remove list element from Eina_List */
    idata->ad->data_list = eina_list_remove(idata->ad->data_list, idata);
    free(idata->data);
    free(idata);
}
```

item_del_cb() 是 List 项目删除事件函数。此函数将删除存储在 Eina_List 中的项目数据。

eina_list_remove() 是一种用于删除存储在 Eina_List 中的项目的 API。

再次运行该示例。选择一个项目并点击 Button，该项目随即被删除。



5) 相关 API

Eina_List *eina_list_append(Eina_List *list, const void *data)：一种用于向 Eina_List 添加新项目的 API。由于此 API 会返回 Eina_List 的指针，因此必须将指针地址存储到 Eina_List 变量中。/ 参数：Eina_List 对象、项目数据对象。

unsigned int eina_list_count(const Eina_List *list)：一种用于返回存储在 Eina_List 中的项目数的 API。/ 参数：Eina_List 对象。

void *eina_list_nth(const Eina_List *list, unsigned int n)：一种用于返回存储在特定位置的项目数据的 API。/ 参数：Eina_List 对象、项目索引编号。

Eina_List *eina_list_remove(Eina_List *list, const void *data)：一种用于删除存储在 Eina_List 中的项目的 API。由于此 API 会返回 Eina_List 的指针，因此必须将指针地址存储到 Eina_List 变量中。/ 参数：Eina_List 对象、项目数据对象。

void elm_list_clear(Evas_Object *obj)：一种用于从 List 小部件删除列表中的所有项目的 API。

26. 使用 Timer

Timer 用于按照固定时间间隔发生事件。使用闹钟应用程序和动画效果时都需要用到 Timer。当您在开发应用程序时，可同时操作多个 Timer。现在我们将通过示例学习如何使用此内容。

1) 启动 Timer 事件

创建新的源项目，并将项目名称指定为“TimerEx”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并向 appdata 结构添加一个新变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Ecore_Timer *timer1;
        int timer_count;
} appdata_s;
```

Ecore_Timer 是 Timer 结构。

timer_count 是一种变量，用于存储 Timer 事件已发生的次数。

在 create_base_gui() 函数之上添加两个新函数。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child, double h_weight, double v_weig
ht,
                double h_align, double v_align)
{
    /* we use a frame for padding only */
    Evas_Object *frame = elm_frame_add(box);
    elm_object_style_set(frame, "pad_small");
    evas_object_size_hint_weight_set(frame, h_weight, v_weight);
    evas_object_size_hint_align_set(frame, h_align, v_align);
    {
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
```

```
ND);
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        evas_object_show(child);
        elm_object_content_set(frame, child);
    }
    elm_box_pack_end(box, frame);
    evas_object_show(frame);
}

static Evas_Object *
my_button_add(Evas_Object *parent, const char *text, Evas_Smart_Cb cb, void *cb_
data)
{
    Evas_Object *btn;

    btn = elm_button_add(parent);
    elm_object_text_set(btn, text);
    evas_object_smart_callback_add(btn, "clicked", cb, cb_data);

    return btn;
}
```

my_box_pack() 是一种用于将小部件添加到 Box 容器中的函数。

my_button_add() 是一种用于创建并返回 Button 小部件的函数。

返回到 create_base_gui() 函数并添加新代码。此代码创建两个 Box 和两个 Button。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        Evas_Object *btn, *box;

        /* Container: standard box */
        box = elm_box_add(ad->win);
```

```
        elm_box_horizontal_set(box, EINA_FALSE);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        evas_object_size_hint_align_set(box, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* Label */
            ad->label = elm_label_add(box);
            elm_object_text_set(ad->label, "No timer");
            my_box_pack(box, ad->label, EVAS_HINT_EXPAND, 0.0, 0.5, 0.0);

            /* Button-1 */
            btn = my_button_add(box, "Start", btn_start_cb, ad);
            my_box_pack(box, btn, EVAS_HINT_EXPAND, 0, EVAS_HINT_FILL, EVAS_HIN
T_FILL);

            /* Button-2 */
            btn = my_button_add(box, "Stop", btn_stop_cb, ad);
            my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVAS_HINT
_FILL, 0.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

第一个 Button 负责启动 Timer。现在我们将为 Button 创建回调函数。在 create_base_gui() 函数之上添加两个新函数。

```
static void
btn_start_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        ad->timer_count = 0;
        ad->timer1 = ecore_timer_add(1.0, timer1_cb, ad);
        elm_object_text_set(ad->label, "Timer started");
}

static void
btn_stop_cb(void *data, Evas_Object *obj, void *event_info)
{
```

```
}
```

点击第一个 Button 后将调用 btn_start_cb() 函数。此代码可将 Timer 事件发生次数的计数重置为“0”，并创建/启动新的 Timer。

ecore_timer_add(double, Ecore_Task_Cb, void*) 是一种用于创建并返回新 Timer 的 API。第一个参数指示时间间隔。时间单位为秒。例如，要使某个事件每隔 1.5 秒发生一次，您需要传递值“1.5”。第二个参数指示 Timer 事件回调函数的名称，第三个参数则指示用户数据。

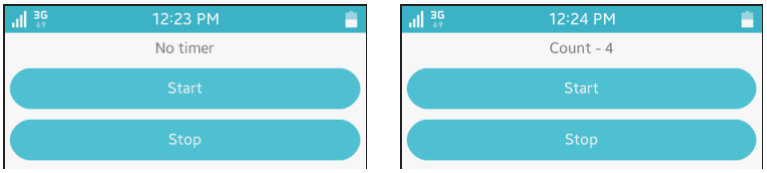
btn_stop_cb() 是用于第二个 Button 的事件函数。我们一会儿就要添加此功能。

现在，我们将创建 Timer 事件回调函数。在 btn_start_cb() 函数之上添加一个新函数。

```
static Eina_Bool
timer1_cb(void *data EINA_UNUSED)
{
        appdata_s *ad = data;
        ad->timer_count ++;
        char buf[100];
        sprintf(buf, "Count - %d", ad->timer_count);

        elm_object_text_set(ad->label, buf);
        return ECORE_CALLBACK_RENEW;
}
```

当发生 Timer 事件时，即会调用该新函数。此代码会增加 Timer 事件发生计数，并在屏幕上显示当前计数。

构建并运行该示例。点击 Start 按钮，随后您将在 Label 小部件中看到相关数字每秒增加一次。



2）停止 Timer

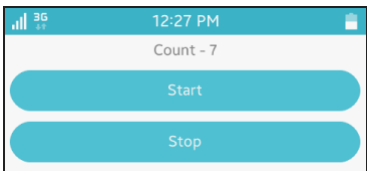
在本小节中，我们将实施一项功能，以便在用户点击 Stop 按钮后停止 Timer。按如下所示，在 btn_stop_cb() 函数之上添加代码：

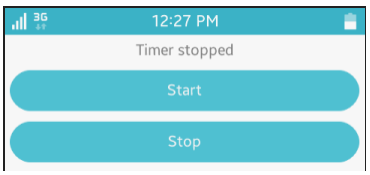
```
static void
btn_stop_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    ecore_timer_freeze(ad->timer1);
    ecore_timer_del(ad->timer1);
    elm_object_text_set(ad->label, "Timer stopped");
}
```

ecore_timer_freeze(Ecore_Timer*) 是一种用于暂停 Timer 事件的 API。要恢复 Timer 事件，您需要使用 ecore_timer_thaw() 函数。

ecore_timer_del(Ecore_Timer*) 是一种用于删除 Timer 对象的 API。

再次运行该示例。点击 Start 按钮，过一会儿之后再点击 Stop 按钮。随后，Timer 事件将停止。





3）相关 API

Ecore_Timer *ecore_timer_add(double in, Ecore_Task_Cb func, void *data)：一种用于创建并返回新 Timer 的 API。/ 参数：时间间隔（单位：秒）、Timer 事件回调函数名称以及用户数据。

void ecore_timer_freeze(Ecore_Timer *timer)：一种用于暂停 Timer 事件的 API。

void ecore_timer_thaw(Ecore_Timer *timer)：一种用于恢复 Timer 事件的 API。

void *ecore_timer_del(Ecore_Timer *timer)：一种用于删除 Timer 对象的 API。

27. 时间和日期

使用 i18n_ucalendar_h 可以请求当前日期和时间，同时还支持添加两种不同的时间。现在我们将通过示例学习如何使用此内容。

1）请求时区

创建新的源项目，并将项目名称指定为“DateTime”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并向 appdata 结构添加一个新变量。并将库头文件包含进来。

```
#include "datetime.h"
#include <utils_i18n.h>

typedef struct appdata {
    Evas_Object *win;
    Evas_Object *conform;
    Evas_Object *label1;
    Evas_Object *label2;
    Evas_Object *label3;
    Evas_Object *label4;
    Evas_Object *label5;
    Evas_Object *slide;
    Ecore_Timer *timer;
    char *tzid;
    i18n_ucalendar_h ucal;
} appdata_s;
```

我们将在 label1 中显示时区，在 label2 中显示当前日期和时间，在 label3 中显示 POSIX 时间，在 label4 中显示加时标题，在 label5 中显示加时结果。
滑块是 Slide 小部件，可将当前日期更改为执行加时操作后所生成的日期。

tzid 是一种字符串变量，用于存储时区。

i18n_ucalendar_h 是一种用于存储日期和时间信息的结构。

现在我们将在屏幕上创建八个 Label 小部件。要请求当前时间，您还需要请求时区设置。返回到 create_base_gui() 函数并添加新代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    /* Table */
    Evas_Object *table = elm_table_add(ad->win);
    elm_table_padding_set(table, 5 * elm_scale_get(), 5 * elm_scale_get());
    elm_object_content_set(ad->conform, table);
    evas_object_show(table);

    {
        Evas_Object *o;

        /* Timezone label */
        o = elm_label_add(table);
        elm_object_text_set(o, "Time Zone:");
        table_pack(table, o, 0, 0, 1, 1, 0.5, 1.0, 1.0, 1.0);

        ad->label1 = elm_label_add(table);
        table_pack(table, ad->label1, 1, 0, 1, 1, 0.5, 1.0, 0.0, 1.0);

        system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE, &
ad->tzid);
        elm_object_text_set(ad->label1, ad->tzid);

        /* Current time label */
        o = elm_label_add(table);
        elm_object_text_set(o, "Current Time:");
        table_pack(table, o, 0, 1, 1, 1, 0.5, 0.0, 1.0, 0.5);

        ad->label2 = elm_label_add(table);
        table_pack(table, ad->label2, 1, 1, 1, 1, 0.5, 0.0, 0.0, 0.5);

        /* Current time label */
        o = elm_label_add(table);
        elm_object_text_set(o, "Since Epoch:");
```

```
    table_pack(table, o, 0, 2, 1, 1, 0.5, 0.0, 1.0, 0.5);

    ad->label3 = elm_label_add(table);
    table_pack(table, ad->label3, 1, 2, 1, 1, 0.5, 0.0, 0.0, 0.5);

    /* Showcase datetime computation */
    ad->label4 = elm_label_add(table);
    elm_object_text_set(ad->label4, "40 days later:");
    table_pack(table, ad->label4, 0, 3, 1, 1, 0.5, 0.0, 1.0, 0.5);

    ad->label5 = elm_label_add(table);
    table_pack(table, ad->label5, 1, 3, 1, 1, 0.5, 0.0, 0.0, 0.5);
}

/* Show window after base gui is set up */
evas_object_show(ad->win);
```

我们已经创建了四个 Label 和两个 Button。

system_settings_get_value_string() 是一种用于请求系统配置信息的 API。若将 SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE 传递给第一个参数，则会将时区字符串返回给第二个参数。

构建并运行该示例。Settings 中指定的时区显示在 Label1 中。

2) 创建当前时间

当您创建 i18n_ucalendar_h 对象时，系统会自动存储当前时间。在 create_base_gui() 函数之上创建一个新函数。

```
static i18n_ucalendar_h
create_time(char *tzid)
{
    i18n_ucalendar_h ucal;
    i18n_uchar *_tzid = (i18n_uchar*)calloc(strlen(tzid) + 1, sizeof(i18n_uchar));
```

```
    // converts 'tzid' to unicode string
    i18n_ustring_copy_ua(_tzid, tzid);
    // gets length of '_tzid'
    int len = i18n_ustring_get_length(_tzid);
    // creates i18n_ucalendar_h
    int ret = i18n_ucalendar_create(_tzid, len, "en_US", I18N_UCALENDAR_TRADITIO
NAL, &ucal);
    if (ret != 0)
    {
        dlog_print(DLOG_ERROR, LOG_TAG, "i18n_ucalendar_create() failed with err
 = %d", ret);
        return NULL;
    }

    return ucal;
}
```

要创建 i18n_ucalendar_h 对象，您需要将时区转换为 i18n_uchar 格式。

i18n_ustring_copy_ua() 是一种用于将存储在 char 数组中的时区复制到 i18n_uchar 数组的 API。

i18n_ustring_get_length() 是一种用于返回 i18n_uchar 数组长度的 API。

i18n_ucalendar_create(i18n_uchar *zone_id, int32_t len, char *locale, i18n_ucalendar_type_e type, i18n_ucalendar_h *calendar) 是一种用于创建 i18n_ucalendar_h 对象的 API。对于第一个参数，返回的是时区；对于第二个参数，返回的是时区字符串长度；对于第三个参数，返回的是区域名称；对于第四个参数，返回的是 ucalendar 类型；对于第五个参数，返回的是 i18n_ucalendar_h 对象。

您可通过这种方式获取当前时间。现在，我们会将存储在 i18n_ucalendar_h 中的日期和时间转换为一个字符串，并在屏幕上显示该字符串。在 create_base_gui() 函数之上创建一个新函数。该函数用于接收 i18n_ucalendar_h 对象，并将对应的日期和时间转换为字符串。

```
static void
update(appdata_s *ad)
{
```

```
    int year, month, day, hour, minute, second;
    i18n_udate udate;
    char buf[256];
    int diff;

    /* Current time */
    i18n_ucalendar_get_now(&udate);
    i18n_ucalendar_set_milliseconds(ad->ucal, udate);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_YEAR, &year);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_MONTH, &month);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_DATE, &day);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_HOUR_OF_DAY, &hour);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_MINUTE, &minute);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_SECOND, &second);
    snprintf(buf, sizeof(buf), "%d/%02d/%02d %02d:%02d:%02d", year, month + 1, d
ay, hour, minute, second);
    elm_object_text_set(ad->label2, buf);
}
```

i18n_ucalendar_get(i18n_ucalendar_h, i18n_ucalendar_date_fields_e, int32_t) 是一种用于从 i18n_ucalendar_h 请求某类型数据的 API。若将 I18N_UCALENDAR_YEAR 传递给第二个参数，会使第三个参数返回年份值。

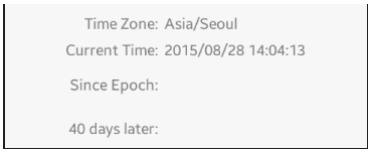
此代码用于按顺序请求年份、月份、日期、小时、分钟和秒钟的值。

通过使用 sprintf() 函数，此代码可将六种时间值转换为单个字符串 printf()，并返回该字符串。需要注意的是，需要为月份添加值“1”。月份值的范围为 0 - 11。

现在，我们将使用我们刚刚创建的两个函数在屏幕上显示当前时间。在 create_base_gui() 函数的结尾添加新代码。

```
    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    /* Prepare calendar object and 1-second timer */
    ad->ucal = create_time(ad->tzid);
    update(ad);
}
```

再次运行该示例。当前日期和时间显示在 Label2 中。



3) 数字手表

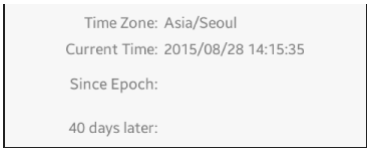
在本小节中，我们将实施一项功能，以便每秒更新一次显示在 Label2 中的日期和时间。换言之，通过使用 Timer，我们将实施一种功效等同于电子手表的功能。在 create_base_gui() 函数的结尾添加新代码。

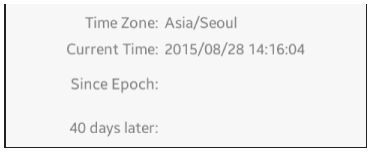
```
    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    /* Prepare calendar object and 1-second timer */
    ad->ucal = create_time(ad->tzid);
    ad->timer = ecore_timer_add(1.0, timer_cb, ad);
    update(ad);
}
```

此代码将每秒生成一次 Timer 事件。现在，我们需要创建 Timer 事件函数。在 create_base_gui() 函数之上创建一个新函数。

```
static Eina_Bool
timer_cb(void *data)
{
        appdata_s *ad = data;
        update(ad);
        return ECORE_CALLBACK_RENEW;
}
```

当发生 Timer 事件时，此代码将调用 update() 函数并更新时间和日期。再次运行该示例。时间每秒变化一次。





4) 计算 POSIX 时间

计算机上使用的时间单位为毫秒，如果您将公元 1 年到当前年份的时间总量转换为以毫秒计算的时间值，将会创建一个天文数字。因此必须采用新的计时机制，从计算机诞生之日起开始计时。1970 年 1 月 1 日 0:00:00 是最终敲定的开始日期。该计时机制被称为 POSIX 时间。通过使用 POSIX 时间，我们可以计算自应用程序启动后经过了多长时间。

向 update() 函数添加一行新代码。此代码指定 Button 回调函数的名称。

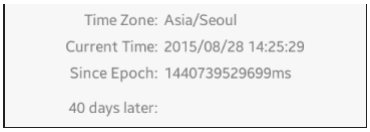
```
static void
update(appdata_s *ad)
{
    int year, month, day, hour, minute, second;
    i18n_udate udate;
    char buf[256];
    int diff;

    /* Current time */
    i18n_ucalendar_get_now(&udate);
    i18n_ucalendar_set_milliseconds(ad->ucal, udate);
    ~
    elm_object_text_set(ad->label2, buf);

    /* POSIX time since EPOCH */
    snprintf(buf, sizeof(buf), "%llums", (unsigned long long) udate);
    elm_object_text_set(ad->label3, buf);
}
```

i18n_udate 是 double 型的同类型数据。POSIX 时间的数值非常大，因此必须使用这种格式。

i18n_ucalendar_get_now(i18n_udate) 是一种用于将当前时间转换为以毫秒为单位的时间的 API。

i18n_ucalendar_set_milliseconds(i18n_ucalendar_h, i18n_udate) 是一种用于在 i18n_ucalendar_h 中指定新时间的 API。

再次运行此示例，随后您将看到 POSIX 时间显示在 label3 中。显示的时间单位为毫秒。



4) 计算时间

在本小节中，我们将学习如何向日期添加数字。首先，添加一个 Slider 小部件。在 create_base_gui() 函数的结尾添加新代码。

```
        ad->label5 = elm_label_add(table);
        table_pack(table, ad->label5, 1, 3, 1, 1, 0.5, 0.0, 0.0, 0.5);

        /* Spinner for more time difference */
        ad->slide = elm_slider_add(table);
        elm_slider_min_max_set(ad->slide, -365, 365);
        elm_slider_value_set(ad->slide, 40);
        table_pack(table, ad->slide, 0, 4, 2, 1, 1.0, 2.0, -1.0, 0.0);
        evas_object_smart_callback_add(ad->slide, "changed", spinner_cb, ad);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

我们已经创建了一个 Slider 小部件并为该小部件指定了 -365 到 +365 的数值范围。我们将值指定为 40。

现在我们将为 Slider 创建事件函数。在 create_base_gui() 函数之上创建一个新函数。当更改了 Slider 的值时，即会调用 update() 函数。

```
static void
spinner_cb(void *data, Evas_Object *obj, void *event_info)
{




appdata_s *ad = data;
        update(ad);
}
```

最后，在 update() 函数的结尾添加新代码。

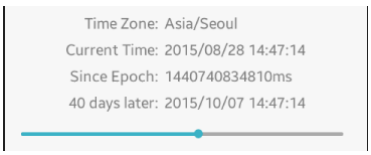
```
static void
update(appdata_s *ad)
{
    ~


    /* POSIX time since EPOCH */
    snprintf(buf, sizeof(buf), "%llums", (unsigned long long) udate);
    elm_object_text_set(ad->label3, buf);

    /* 40 days later (label) */
    diff = (int) elm_slider_value_get(ad->slide);
    if (diff >= 0)
        snprintf(buf, sizeof(buf), "%d day%s later:", diff, (diff > 1) ? "s" :
"");
    else if (diff < 0)
        snprintf(buf, sizeof(buf), "%d day%s earlier:", -diff, (diff < -1) ? "s
" : "");
    elm_object_text_set(ad->label4, buf);

    /* 40 days later (value) */
    i18n_ucalendar_add(ad->ucal, I18N_UCALENDAR_DATE, diff);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_YEAR, &year);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_MONTH, &month);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_DATE, &day);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_HOUR_OF_DAY, &hour);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_MINUTE, &minute);
    i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_SECOND, &second);
    snprintf(buf, sizeof(buf), "%d/%02d/%02d %02d:%02d:%02d", year, month + 1,
day, hour, minute, second);
    elm_object_text_set(ad->label5, buf);
```

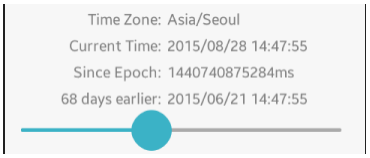
```
}
```

对正负值进行区分，此代码会在 label4 中显示给定的 Slider 值，在 label5 中显示当前日期与该值之和。

i18n_ucalendar_add() 是一种用于将数字添加到 i18n_ucalendar_h 对象的特定项目的函数。将 I8N_UCALENDAR_DATE 传递给第二个参数并将 40 传递给第三个参数后，即会产生一个比当前日期迟 40 天的日期。

再次运行该示例，然后点击第二个 Button。此时，比当前日期迟 40 天的日期会显示在 label4 中。通过 Slider 执行拖动操作可以更改用于计算的日期。





5) 相关 API

int system_settings_get_value_string(system_settings_key_e key, char **value)：一种用于请求系统配置信息的 API。若将 SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE 传递给第一个参数，则会将时区字符串返回给第二个参数。

i18n_uchar* i18n_ustring_copy_ua (i18n_uchar *dest, const char *src)：一种用于将存储在 char 数组中的时区复制到 i18n_uchar 数组的 API。

int32_t i18n_ustring_get_length (i18n_uchar *s)：一种用于返回 i18n_uchar 数组长度的 API。

int i18n_ucalendar_create (i18n_uchar *zone_id, int32_t len, char *locale, i18n_ucalendar_type_e type, i18n_ucalendar_h *calendar)：一种用于创建 i18n_ucalendar_h 对象的 API。/ 参数：时区、时区字符串长度、区域名称、ucalendar 类型以及 i18n_ucalendar_h 对象返回。

int i18n_ucalendar_get (i18n_ucalendar_h calendar, i18n_ucalendar_date_fields_e field, int32_t *val)：一种用于从 i18n_ucalendar_h 请求某类型数据的 API。/ 参数：i18n_ucalendar_h 对象、日期和时间字段，以及日期和时间值返回。

日期和时间字段类型：
- I18N_UCALENDAR_YEAR：年份
- I18N_UCALENDAR_MONTH：月份
- I18N_UCALENDAR_DATE：日期
- I18N_UCALENDAR_DAY_OF_WEEK：星期几
- I18N_UCALENDAR_AM_PM：上午或下午
- I18N_UCALENDAR_HOUR：小时
- I18N_UCALENDAR_MINUTE：分钟
- I18N_UCALENDAR_SECOND：秒钟
- I18N_UCALENDAR_MILLISECOND：毫秒

i18n_udate：double 型的同类型数据。用于存储 POSIX 时间。

int i18n_ucalendar_get_milliseconds(i18n_ucalendar_h calendar, i18n_udate *date)：一种用于将存储在 i18n_ucalendar_h 中的时间转换为 POSIX 时间的 API。显示的时间单位为毫秒。

int i18n_ucalendar_add (i18n_ucalendar_h calendar, i18n_ucalendar_date_fields_e field, int32_t amount)：一种用于将数字添加到 18n_ucalendar_h 对象的特定项目的 API。/ 参数：i18n_ucalendar_h 对象、日期和时间字段以及要添加的数字。

28. 日历示例

在此例中，我们将学习如何使用 i18n_ucalendar_h 创建日历。

1) 屏幕 UI 构成

创建新的源项目，并将项目名称指定为“CalendarEx”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，将新变量添加到 appdata 结构，同时还需添加库头文件。

```
#include "calendarex.h"
#include <utils_i18n.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *label_day[6][7];
        char *tzid;
        i18n_ucalendar_h ucal;
} appdata_s;
```

label_day 是用于显示日期的 Label 小部件数组。允许用于显示日期的空间为：横向最多 7 行，纵向最多 6 列。

在 tzid 中存储时区。

在 ucal 中存储当前日期和时间。

现在我们将添加 Label 小部件，以便在屏幕上显示日期。在 create_base_gui() 函数之上创建一个新函数。此函数将向 Table 添加一个小部件。

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int x, int y, int w, int
h)
{
```

```
   evas_object_size_hint_align_set(child, 0.5, 0.5);
   evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
   elm_table_pack(table, child, x, y, w, h);
   evas_object_show(child);
}
```

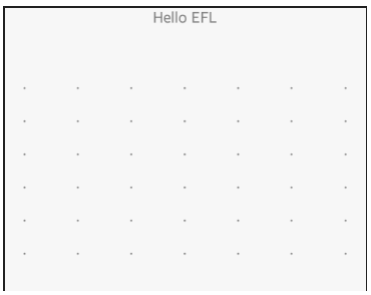
然后，在 create_base_gui() 函数的结尾添加新代码。此代码用于创建一个 Box 和一个 Table，并添加一个 Label。对用于创建 Conformant 的代码标注注释。

```
    /* Conformant */
    /*ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);*/

    /* Box to put the table in so we can bottom-align the table
     * window will stretch all resize object content to win size */
    Evas_Object *box = elm_box_add(ad->win);
    evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_win_resize_object_add(ad->win, box);
    evas_object_show(box);

    /* Table */
    Evas_Object *table = elm_table_add(ad->win);
    /* Make table homogenous - every cell will be the same size */
    elm_table_homogeneous_set(table, EINA_TRUE);
    /* Set padding of 10 pixels multiplied by scale factor of UI */
    elm_table_padding_set(table, 10 * elm_config_scale_get(), 10 * elm_config_s
cale_get());
    /* Let the table child allocation area expand within in the box */
    evas_object_size_hint_weight_set(table, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    /* Set table to fiill width but align to bottom of box */
    evas_object_size_hint_align_set(table, EVAS_HINT_FILL, 0.0);
    elm_box_pack_end(box, table);
    evas_object_show(table);

    {
        /* Label*/
        ad->label = elm_label_add(ad->win);
```

```
        elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
        my_table_pack(table, ad->label, 0, 0, 7, 1);

        for(int j=0; j < 6; j++)
        {
            for(int i=0; i < 7; i++)
            {
                ad->label_day[j][i] = elm_label_add(ad->win);
                elm_object_text_set(ad->label_day[j][i], ".");
                my_table_pack(table, ad->label_day[j][i], i, j + 2, 1, 1);
            }
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

通过使用两个 for 循环，我们创建了一个横向 7 行、纵向 6 列的 Label 小部件。

构建并运行该示例。“.”符号显示在新添加的 Label 中。



2) 显示当前日期和时间

在本小节中，我们将创建一个 i18n_ucalendar_h 对象，并在屏幕上显示当前日期和时间。在 create_base_gui() 函数之上添加两个新函数。

```
static i18n_ucalendar_h
create_time(char *tzid)
{
```

```
        i18n_ucalendar_h ucal;
        i18n_uchar *_tzid = (i18n_uchar*)calloc(strlen(tzid) + 1, sizeof(i18n_uc
har));
        i18n_ustring_copy_ua(_tzid, tzid);
        int len = i18n_ustring_get_length(_tzid);
        int ret = i18n_ucalendar_create(_tzid, len, "en_US", I18N_UCALENDAR_TRAD
ITIONAL, &ucal);
        return ucal;
}

static char* time2string(i18n_ucalendar_h ucal)
{
        int year, month, day, hour, minute, second;
        i18n_ucalendar_get(ucal, I18N_UCALENDAR_YEAR, &year);
        i18n_ucalendar_get(ucal, I18N_UCALENDAR_MONTH, &month);
        i18n_ucalendar_get(ucal, I18N_UCALENDAR_DATE, &day);
        i18n_ucalendar_get(ucal, I18N_UCALENDAR_HOUR, &hour);
        i18n_ucalendar_get(ucal, I18N_UCALENDAR_MINUTE, &minute);
        i18n_ucalendar_get(ucal, I18N_UCALENDAR_SECOND, &second);

        char *buf = malloc(100);
        sprintf(buf, "Now :%04d-%02d-%02d %02d:%02d:%02d", year, month + 1, day,
 hour, minute, second);
        return buf;
}
```

create_time() 是一种用于创建并返回 i18n_ucalendar_h 对象的函数，time2string() 是一种用于将存储在 i18n_ucalendar_h 中的日期转换为字符串并返回该字符串的函数。详情请参阅 DateTime 示例。

在本小节中，我们将使用上述函数，并在屏幕上显示当前日期和时间。在 create_base_gui() 函数的结尾添加新代码。

```
        evas_object_show(ad->win);

        system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE, &
ad->tzid);
        ad->ucal = create_time(ad->tzid);
        char *buf = time2string(ad->ucal);
        elm_object_text_set(ad->label, buf);
}
```

使用 system_settings_get_value_string() 函数请求时区。

使用 create_time() 函数创建 i18n_ucalendar_h 对象。

使用 time2string() 函数将存储在 i18n_ucalendar_h 中的日期和时间转换为单个字符串。

再次运行该示例。当前日期和时间显示在 Label 小部件中。

3) 计算日历日期

在本小节中，我们将找出当前月份的第一天是星期几并在 Label 数组中输入日期。在 create_base_gui() 函数之上创建一个新函数。

```
static void
draw_calendar(appdata_s *ad)
{
        int date, month, dow, days, is_leap;
        int max_day[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

        i18n_ucalendar_set(ad->ucal, I18N_UCALENDAR_DATE, 1);
        i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_MONTH, &month);
        i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_DAY_OF_WEEK, &dow);
        days = max_day[month];

        if( month == 1 )
        {
                i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_IS_LEAP_MONTH, &is_l
eap);
                if( is_leap == 1 )
                        days = 29;
        }

        int i=0, j=0;
        char buf[10];

        i = dow - 1;
        for(int d=1; d <= days; d++)
```

```
        {
                sprintf(buf, "%d", d);
                elm_object_text_set(ad->label_day[j][i], buf);

                i ++;
                if( i >= 7 )
                {
                        i = 0;
                        j ++;
                }
        }
}
```

max_day[] 是一种用于存储一月至十二月的最大天数的数组。

i18n_ucalendar_set() 是一种用于为 i18n_ucalendar_h 对象的特定项目指定新值的 API。将 I18N_UCALENDAR_DATE 传递给第二个参数并将 1 传递给第三个参数后，即会将当前日期更改为 1 号。

i18n_ucalendar_get() 是一种用于返回 i18n_ucalendar_h 对象中特定项目的值的 API。日期项目类型如下：

- 将 I18N_UCALENDAR_MONTH 传递给第二个参数，即会让第三个参数返回月份值。
- 将 I18N_UCALENDAR_DAY_OF_WEEK 传递给第二个参数，即会让第三个参数返回星期几。
- 将 I18N_UCALENDAR_IS_LEAP_MONTH 传递给第二个参数，即会让第三个参数返回当前月份是否为闰月。

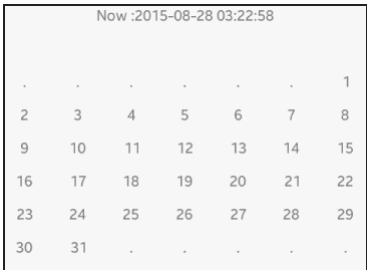
如果当前月份为闰月，最大天数将更改为 29 天。

跟在上述内容之后的代码用于分别将对应于某个月第一天到最后一天之间各天的数字转换为字符串，并在相关 Label 小部件中显示这些字符串。

我们需要在执行应用程序时调用此函数。在 create_base_gui() 函数的结尾添加一行新代码。

```
        elm_object_text_set(ad->label, buf);

        draw_calendar(ad);
}
```

再次运行该示例。屏幕上将显示与当天日期对应的日历。



4) 显示当前日期

在本小节中，我们将实施一项功能，以使用“[]”符号将对应于当天日期的数字括起来。向 draw_calendar() 函数添加新代码。

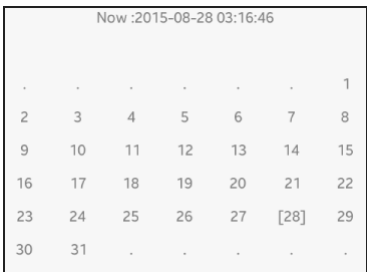
```
static void
draw_calendar(appdata_s *ad)
{
        int date, month, dow, days, is_leap;
        int max_day[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

        i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_DATE, &date);
        i18n_ucalendar_set(ad->ucal, I18N_UCALENDAR_DATE, 1);
        i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_MONTH, &month);
        i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_DAY_OF_WEEK, &dow);
        days = max_day[month];

        if( month == 1 )
        {
                i18n_ucalendar_get(ad->ucal, I18N_UCALENDAR_IS_LEAP_MONTH, &is_l
eap);
                if( is_leap == 1 )
                        days = 29;
```

```
        }

        int i=0, j=0;
        char buf[10];

        i = dow - 1;
        for(int d=1; d <= days; d++)
        {
                sprintf(buf, "%d", d);
                if( d == date )
                        sprintf(buf, "[%d]", d);
                elm_object_text_set(ad->label_day[j][i], buf);
                ~
```

使用 i18n_ucalendar_get() 函数请求当天日期，并将其存储在变量中。

当 for 循环中需要当天的日期时，系统会向对应值添加“[]”符号。

再次运行该示例。现在您将看到当天的日期状态有别于其它日期的状态。



5) Calendar 小部件

使用 Calendar 小部件可以轻松创建 Calendar。创建新示例并将其名称指定为 CalendarWidgetEx。

向 create_base_gui() 函数添加新代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
```

```
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        { /* child object - indent to how relationship */
            /* Label*/
            ad->label = elm_label_add(ad->win);
            elm_object_text_set(ad->label, "<align=center>Calendar</>");
            evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, 0.0);
            evas_object_size_hint_align_set(ad->label, EVAS_HINT_FILL, 0.5);
            elm_box_pack_end(box, ad->label);
            evas_object_show(ad->label);

            Evas_Object *cal = elm_calendar_add(ad->win);
            evas_object_size_hint_weight_set(cal, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
            evas_object_size_hint_align_set(cal, EVAS_HINT_FILL, 0.5);
            elm_box_pack_end(box, cal);
            evas_object_show(cal);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

elm_calendar_add() 是一种用于创建 Calendar 小部件的 API。

运行此示例，随后您将看到屏幕上显示 Calendar。使用向左/向右箭头可转至上一个/下一个月。

有关如何使用 Calendar 小部件的详情，请参阅 Help Contents 中的以下路径。

API References > Native Application > Mobile Native > Native API Reference > UI > EFL > Elementary > Elementary Widgets

6) 相关 API

int system_settings_get_value_string(system_settings_key_e key, char **value)：一种用于请求系统配置信息的 API。若将 SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE 传递给第一个参数，则会将时区字符串返回给第二个参数。

i18n_uchar* i18n_ustring_copy_ua (i18n_uchar *dest, const char *src)：一种用于将存储在 char 数组中的时区复制到 i18n_uchar 数组的 API。

int32_t i18n_ustring_get_length (i18n_uchar *s)：一种用于返回 i18n_uchar 数组长度的 API。

int i18n_ucalendar_create (i18n_uchar *zone_id, int32_t len, char *locale, i18n_ucalendar_type_e type, i18n_ucalendar_h *calendar)：一种用于创建 i18n_ucalendar_h 对象的 API。/ 参数：时区、时区字符串长度、区域名称、ucalendar 类型以及 i18n_ucalendar_h 对象返回。

int i18n_ucalendar_get (i18n_ucalendar_h calendar, i18n_ucalendar_date_fields_e field, int32_t *val)：一种用于从 i18n_ucalendar_h 请

求某类型数据的 API。/ 参数：i18n_ucalendar_h 对象、日期和时间字段，以及日期和时间值返回。

日期和时间字段类型：
- I18N_UCALENDAR_YEAR：年份
- I18N_UCALENDAR_MONTH：月份
- I18N_UCALENDAR_DATE：日期
- I18N_UCALENDAR_DAY_OF_WEEK：星期几
- I18N_UCALENDAR_AM_PM：上午或下午
- I18N_UCALENDAR_HOUR：小时
- I18N_UCALENDAR_MINUTE：分钟
- I18N_UCALENDAR_SECOND：秒钟
- I18N_UCALENDAR_MILLISECOND：毫秒

i18n_udate：double 型的同类型数据。用于存储 POSIX 时间。

int i18n_ucalendar_get_milliseconds(i18n_ucalendar_h calendar, i18n_udate *date)：一种用于将存储在 i18n_ucalendar_h 中的时间转换为 POSIX 时间的 API。显示的时间单位为毫秒。

int i18n_ucalendar_add (i18n_ucalendar_h calendar, i18n_ucalendar_date_fields_e field, int32_t amount)：一种用于将数字添加到 18n_ucalendar_h 对象的特定项目的 API。/ 参数：i18n_ucalendar_h 对象、日期和时间字段以及要添加的数字。

29. 请求鼠标触摸事件

要创建图像浏览器，您需要实施一项功能，以采用幻灯片形式显示图像，并且用户可在幻灯片中通过左右拖动屏幕来切换图像。此外，对于陶笛和钢琴应用程序等乐器应用程序，必须请求多次触摸信息。除了上述应用程序，大多数游戏和其它高品质应用程序也要求使用触摸事件。在此例中，我们将学习如何在用户触摸了 Window 时请求事件。

1) 请求容器触摸事件

创建新的源项目，并将项目名称指定为“MouseTouchEvent”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在 create_base_gui() 函数末尾添加新代码。

```
        /* Label*/
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "Hello EFL");
        evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_
EXPAND);
        elm_object_content_set(ad->conform, ad->label);
        evas_object_show(ad->label);

        /* Mouse Touch event callback */
        evas_object_event_callback_add( ad->conform, EVAS_CALLBACK_MOUSE_DOWN,
on_mouse_down , ad);
        evas_object_event_callback_add( ad->conform, EVAS_CALLBACK_MOUSE_MOVE,
on_mouse_move , ad);
        evas_object_event_callback_add( ad->conform, EVAS_CALLBACK_MOUSE_UP, on
_mouse_up , ad);

        /* Show window after base gui is set up */
        evas_object_show(ad->win);
```

evas_object_event_callback_add() 是一种用于为 evas 对象指定回调函数的 API。在此例中，Evas 对象为屏幕上显示的所有对象。因此，evas 对象包括基本对象（Line、Rect、Polygon、Text 及 Image）和智能对象（容器与小部件）。

对于第一个参数，请输入将要发生事件的对象。在此例中，我们已将对象指定为 Conformant。
对于第二个参数，请输入事件类型。EVAS_CALLBACK_MOUSE_DOWN 指示 Touch Down 事件。EVAS_CALLBACK_MOUSE_MOVE 指示 Touch Move 事件。EVAS_CALLBACK_MOUSE_UP 指示 Touch Cancel 事件。
对于第三个参数，请输入回调函数的名称。第四个参数指示用户数据。

现在，我们将实施一个回调函数，一旦用户触摸 Conformant 就会调用此函数。在 create_base_gui() 函数之上添加三个新函数。

```
static void
on_mouse_down(void *data, Evas *e, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        Evas_Event_Mouse_Down *ev = event_info;
        char buf[100];

        sprintf(buf, "Win Mouse down:%d,%d", ev->canvas.x, ev->canvas.y);
        elm_object_text_set(ad->label, buf);
}

static void
on_mouse_move(void *data, Evas *e, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        Evas_Event_Mouse_Move *ev = event_info;
        char buf[100];

        sprintf(buf, "Win Mouse move:%d,%d/%d,%d",
                        ev->prev.canvas.x, ev->prev.canvas.y, ev->cur.canvas.x,
ev->cur.canvas.y);
        elm_object_text_set(ad->label, buf);
}

static void
on_mouse_up(void *data, Evas *e, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        Evas_Event_Mouse_Up *ev = event_info;
        char buf[100];

        sprintf(buf, "Win Mouse up:%d,%d", ev->canvas.x, ev->canvas.y);
        elm_object_text_set(ad->label, buf);
```

```
}
└──────────────────────────────────────┘
```

on_mouse_down() 是在用户对 Conformant 执行了 Touch Down 操作时执行的回调函数。第一个参数接收用户数据，第二个参数接收发生了事件的对象。第三个参数接收包含 Touch 事件信息的 Evas_Event_Mouse_Down 对象。

在 Evas_Event_Mouse_Down 对象的属性中，canvas 属性包含 Touch 点坐标。

on_mouse_move() 是在用户对 Conformant 执行了 Touch Move 操作时执行的回调函数。第一个参数接收用户数据，第二个参数接收发生了事件的对象。第三个参数接收包含 Touch 事件信息的 Evas_Event_Mouse_Move 对象。

在 Evas_Event_Mouse_Move 对象的属性中，prev.canvas 属性包含之前的 Touch 点坐标。cur.canvas 包含当前的 Touch 点坐标。

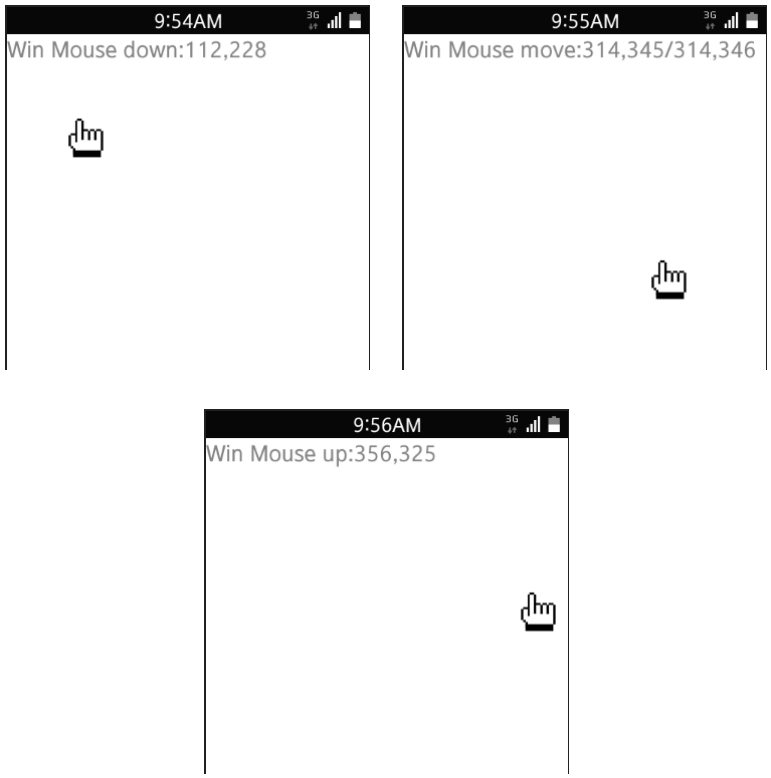
on_mouse_up() 是在用户对 Conformant 执行了 Touch Cancel 操作时执行的回调函数。第一个参数接收用户数据，第二个参数接收发生了事件的对象。第三个参数接收包含 Touch 事件信息的 Evas_Event_Mouse_Up 对象。

在 Evas_Event_Mouse_UP 对象的属性中，canvas 属性包含 Touch 点坐标。

构建并运行该示例。使用您的鼠标触摸屏幕。您将看到当前点坐标显示在 Label 小部件中。

使用您的鼠标拖动屏幕，您可看到之前的 Touch 点坐标和当前的 Touch 点坐标显示在 Label 小部件中。

将您的手指从鼠标上移开后，Label 小部件中会显示最近一次的点坐标。



3) 请求多次触摸事件

在本小节中，我们将在用户使用多个手指触摸屏幕时请求对应的各项触摸信息。向 create_base_gui() 函数添加新代码。

```
        /* Mouse Touch event callback */
        evas_object_event_callback_add( ad->conform, EVAS_CALLBACK_MOUSE_DOWN, o
n_mouse_down , ad);
        evas_object_event_callback_add( ad->conform, EVAS_CALLBACK_MOUSE_MOVE, o
n_mouse_move , ad);
        evas_object_event_callback_add( ad->conform, EVAS_CALLBACK_MOUSE_UP, on_
mouse_up , ad);

        /* Multi Touch event callback */
        evas_object_event_callback_add(ad->conform, EVAS_CALLBACK_MULTI_DOWN, m
ulti_down_cb, ad);
        evas_object_event_callback_add(ad->conform, EVAS_CALLBACK_MULTI_MOVE, m
ulti_move_cb, ad);
```

```
    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

EVAS_CALLBACK_MULTI_DOWN 指示 Multi Touch Down 事件。

EVAS_CALLBACK_MULTI_MOVE 指示 Multi Touch Move 事件。

现在，我们将创建 Multi Touch 事件函数。向 create_base_gui() 函数添加两个新函数。

```
static void
multi_down_cb(void *data, Evas *e, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        Evas_Event_Multi_Down *ev = (Evas_Event_Multi_Down*)event_info;
        char buf[100];

        sprintf(buf, "Multi down : %d - %d,%d", ev->device, ev->canvas.x, ev->ca
nvas.y);
        elm_object_text_set(ad->label, buf);
}

static void
multi_move_cb(void *data, Evas *e, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        Evas_Event_Multi_Move *ev = (Evas_Event_Multi_Move*)event_info;
        char buf[100];

        sprintf(buf, "Multi move : %d - %d,%d", ev->device, ev->cur.canvas.x, ev
->cur.canvas.y);
        elm_object_text_set(ad->label, buf);
}
```

multi_down_cb() 指示 Multi Touch Down 事件函数。未接收第一个点的事件。第三个参数接收包含 Touch 事件信息的 Evas_Event_Multi_Down 对象。

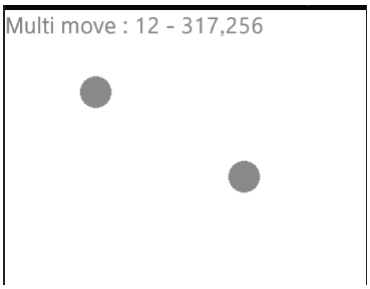
在 Evas_Event_Multi_Down 对象的属性中，canvas 属性包含 Touch 点坐

标。设备属性包含 Touch 点的 ID。当发生了 Move 事件或 Up 事件时，可通过 ID 来识别事件。

multi_move_cb() 指示 Multi Touch Move 事件函数。未接收第一个点的事件。第三个参数接收包含 Touch 事件信息的 Evas_Event_Multi_Move 对象。

在 Evas_Event_Multi_Move 对象的属性中，cur 属性包含 Touch 点坐标。设备属性包含 Touch 点的 ID。当发生了 Down 事件时，可通过 ID 来识别事件。

再次运行该示例。要在模拟器中测试 Multi Touch，请在按住键盘上的 Ctrl 键的同时左键单击屏幕上的两个点。然后，屏幕上会出现一个灰色圆圈。要测试 Multi Touch Move 事件，请拖动灰色圆圈。当您将手指从 Ctrl 键移开后，Multi Touch 标记会消失。



4）相关 API

void evas_object_event_callback_add(Evas_Object *obj, Evas_Callback_Type type, Evas_Object_Event_Cb func, void *data)：一种用于指定 evas 对象的回调函数的 API。Evas 对象是指屏幕上显示的所有对象。因此，evas 对象包括基本对象（Line、Rect、Polygon、Text 及 Image）和智能对象（容器与小部件）。/ 参数：发生事件的对象、事件类型、回调函数名称以及用户数据。事件类型如下：

- EVAS_CALLBACK_MOUSE_DOWN：Touch Down 事件
- EVAS_CALLBACK_MOUSE_MOVE：Touch Move 事件
- EVAS_CALLBACK_MOUSE_UP：Touch Cancel 事件
- EVAS_CALLBACK_MULTI_DOWN：Multi Touch Down 事件
- EVAS_CALLBACK_MULTI_MOVE：Multi Touch Move 事件

Evas_Event_Mouse_Down：Touch Down 事件信息结构。在各属性当中，canvas 属性包含 Touch 点坐标。

Evas_Event_Mouse_Move：Touch Move 事件信息结构。在各属性当中，prev.canvas 属性包含之前的 Touch 点坐标。cur.canvas 包含当前的 Touch 点坐标。

Evas_Event_Mouse_Up：Touch Cancel 事件信息结构。在各属性当中，canvas 属性包含 Touch 点坐标。

Evas_Event_Multi_Down：Multi Touch Down 事件信息结构。在各属性当中，canvas 属性包含 Touch 点坐标。设备属性包含 Touch 点的 ID。当发生了 Move 事件或 Up 事件时，可通过 ID 来识别事件。

Evas_Event_Multi_Move：Multi Touch Move 事件信息结构。在对象的各属性当中，cur 属性包含 Touch 点坐标。设备属性包含 Touch 点的 ID。当发生了 Down 事件时，可通过 ID 来识别事件。

30. 计算器示例

在此例中，我们将学习如何使用数学函数创建简单的计算器。

1) 屏幕 UI 构成

创建新的源项目，并将项目名称指定为“CalculatorEx”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，将新变量添加到 appdata 结构，同时还需要添加库头文件和 define 语句。

```
#include "calculatorex.h"
#include <math.h>

#define ID_BACK                 101
#define ID_CLEAR                102
#define ID_DOT                  103
#define ID_EQUAL                104
#define ID_PLUS                 111
#define ID_MINUS                112
#define ID_MULTIPLY             113
#define ID_DIVIDE               114
#define ID_X2                   121
#define ID_X3                   122
#define ID_SQRT                 123
#define ID_RECIPE               124

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *entry;
        float value;
        int calc_mode;
} appdata_s;
```

math.h 是数学函数库头文件。

在 define 语句中，定义 ID 以便区分各个 Button。

计算结果会显示在已添加到 appdata 结构的“entry”中。临时计算结果会存储在“value”中。

计算类型（+、-、*、/）会存储在 calc_mode 中。

此例中将创建很多 Button 小部件。为了减少源代码，我们现在将创建一种用于创建 Button 的函数。在 create_base_gui() 函数之上添加两个新函数。

```
static void
btn_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
}

static void
create_button(Evas_Object *parent, const char* text, int x, int y, int w, int h,
 void *data)
{
    Evas_Object *btn = elm_button_add(parent);
    elm_object_text_set(btn, text);
    evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, EVAS_HINT_FILL);
    elm_table_pack(parent, btn, x, y, w, h);
    evas_object_smart_callback_add(btn, "clicked", btn_clicked_cb, data);
    evas_object_show(btn);
}
```

btn_clicked_cb() 是 Button 回调函数。多个 Button 将调用同一回调函数。

create_button() 是一种用于接收与 Button 有关的信息并创建 Button 小部件的函数。

现在我们将创建一个 Box、一个 Table 和 22 个 Button。向 create_base_gui() 函数添加新代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
```

```
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* Box to put the table in so we can bottom-align the table
         * window will stretch all resize object content to win size */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_win_resize_object_add(ad->win, box);
        evas_object_show(box);

        /* Table */
        Evas_Object *table = elm_table_add(ad->win);
        /* Make table homogenous - every cell will be the same size */
        elm_table_homogeneous_set(table, EINA_TRUE);
        /* Set padding of 10 pixels multiplied by scale factor of UI */
        elm_table_padding_set(table, 10 * elm_config_scale_get(), 10 * elm_conf
ig_scale_get());
        /* Let the table child allocation area expand within in the box */
        evas_object_size_hint_weight_set(table, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
        /* Set table to fiill width but align to bottom of box */
        evas_object_size_hint_align_set(table, EVAS_HINT_FILL, 1.0);
        elm_object_content_set(ad->conform, table);
        evas_object_show(table);

        { /* child object - indent to how relationship */
            /* Entry */
            ad->entry = elm_entry_add(ad->win);
            elm_object_text_set(ad->entry, "0");
            evas_object_size_hint_weight_set(ad->entry, EVAS_HINT_EXPAND, EVAS_
HINT_EXPAND);
            evas_object_size_hint_align_set(ad->entry, EVAS_HINT_FILL, EVAS_HIN
T_FILL);
            elm_table_pack(table, ad->entry, 0, 0, 4, 1);
            evas_object_show(ad->entry);

            create_button(table, "Back", 0, 1, 2, 1, ID_BACK);
            create_button(table, "Clear", 2, 1, 2, 1, ID_CLEAR);

            create_button(table, "7", 0, 2, 1, 1, 7);
            create_button(table, "8", 1, 2, 1, 1, 8);
```

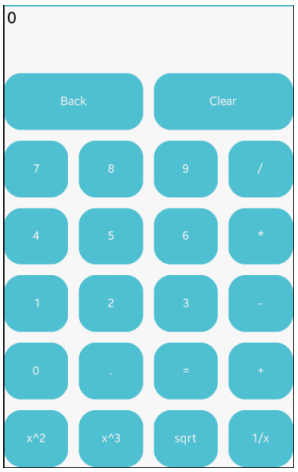
```
            create_button(table, "9", 2, 2, 1, 1, 9);
            create_button(table, "/", 3, 2, 1, 1, ID_DIVIDE);

            create_button(table, "4", 0, 3, 1, 1, 4);
            create_button(table, "5", 1, 3, 1, 1, 5);
            create_button(table, "6", 2, 3, 1, 1, 6);
            create_button(table, "*", 3, 3, 1, 1, ID_MULTIPLY);

            create_button(table, "1", 0, 4, 1, 1, 1);
            create_button(table, "2", 1, 4, 1, 1, 2);
            create_button(table, "3", 2, 4, 1, 1, 3);
            create_button(table, "-", 3, 4, 1, 1, ID_MINUS);

            create_button(table, "0", 0, 5, 1, 1, 0);
            create_button(table, ".", 1, 5, 1, 1, ID_DOT);
            create_button(table, "=", 2, 5, 1, 1, ID_EQUAL);
            create_button(table, "+", 3, 5, 1, 1, ID_PLUS);

            create_button(table, "x^2", 0, 6, 1, 1, ID_X2);
            create_button(table, "x^3", 1, 6, 1, 1, ID_X3);
            create_button(table, "sqrt", 2, 6, 1, 1, ID_SQRT);
            create_button(table, "1/x", 3, 6, 1, 1, ID_RECIPE);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

要根据屏幕高宽比放置小部件，我们创建了一个 Table。同时，为了指定小部件之间的距离，我们还创建了一个 Box。

为了显示计算结果，我们创建了一个 Entry 小部件。

以下是用于创建总计 22 个 Button 小部件的代码。对于数字 Button，系统会将相关数值作为用户数据传递给回调函数。对于其它 Button，系统会将定义了 define 语句的 ID 值作为用户数据进行传递。

构建并运行该示例。您可在屏幕顶端看到一个 Entry 小部件，并在 Entry 小部件下方看到 22 个 Button 小部件。



2）实施数字 Button 功能

在本小节中，我们将实施一项功能，以便在用户点击了 Button 0 到 Button 9 之间的某个数字 Button 时将相关数字添加到 Label 小部件。首先，将 appdata 声明为全局变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *nf;
        Elm_Object_Item *frame_item;
        float value;
        int calc_mode;
} appdata_s;

appdata_s* m_ad = 0;
```

然后，在 create_base_gui() 函数的开头初始化 appdata 变量。

```
static void
create_base_gui(appdata_s *ad)
{
        m_ad = ad;
```

现在我们将创建一个用于将 Entry 标题文本转换为数字的函数，同时还创建一个用于将文本添加到 Entry 小部件的函数。在 btn_clicked_cb() 函数之上添加两个新函数。

```
static float
get_entry_value()
{
    char* text = elm_object_text_get(m_ad->entry);
    float value = atof(text);
    return value;
}

static void
append_number_label(char str_new) {
    char buf[100];

    char* text = elm_object_text_get(m_ad->entry);
    float value = get_entry_value();
    if( value == 0.f )
        sprintf(buf, "%c", str_new);
    else
        sprintf(buf, "%s%c", text, str_new);

    elm_object_text_set(m_ad->entry, buf);
}
```

get_entry_value() 是一种用于将 Entry 标题文本转换为 float 类型并返回此内容的函数。

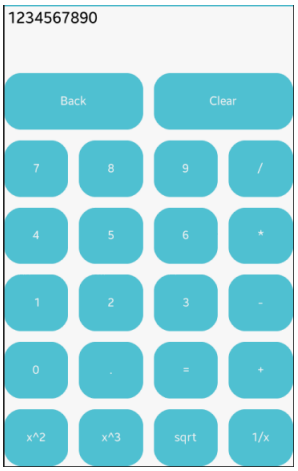
append_number_label() 是一种用于接收 char 变量并将此变量添加到 Entry 标题文本末尾的函数。

当用户点击某个数字 Button 时，需要调用此函数。向 btn_clicked_cb() 函数添加以下代码。

```
static void
btn_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
        char* text = NULL;
        int length = 0;
        float value = 0.f;
        int id = (int)data;

        if( id >= 0 && id <= 9 )
        {
                append_number_label('0' + id);
                return;
        }
}
```

如果用户数据是介于 0 到 9 之间的某个数字，则 Button 会被视为数字 Button，并且新文本将被添加到 Label 小部件的标题文本中。

再次运行该示例。点击某个数字 Button，相关值将被添加到 Label 小部件中。



3) 实施 Dot、Clear 和 Back 按钮功能

在本小节中，我们将实施“.”按钮、Back 按钮和 Clear 按钮功能。在 btn_clicked_cb() 函数的结尾添加新代码。

```
        if( id >= 0 && id <= 9 )
        {
                append_number_label('0' + id);
                return;
        }

        switch( id )
        {
        case ID_DOT :
                append_number_label('.');
                break;
        case ID_CLEAR :
                elm_object_text_set(m_ad->label, "0");
                break;
        case ID_BACK :
                text = elm_object_text_get(m_ad->label);
                length = strlen(text);
                if( length > 0 )
                        text = eina_stringshare_add_length(text, length - 1);
                if( strlen(text) < 1 )
                        text = "0";
                elm_object_text_set(m_ad->label, text);
                break;
        }
}
```

如果用户点击了“.”按钮，“.”符号将被添加到 Label 文本末尾处。

如果用户点击了 Clear 按钮，此代码将把 Label 文本改为“0”。

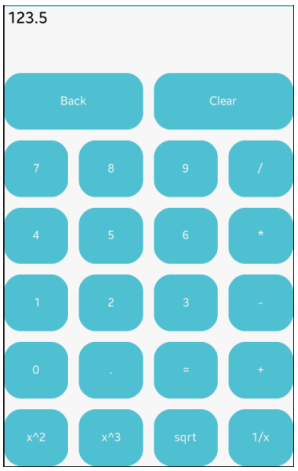
如果用户点击了 Back 按钮，此代码将删除 Label 文本的最后一个字符。

eina_stringshare_add_length(char*, unsigned int) 是一种用于从某个字符串起始处提取指定长度字符的 API。

再次运行该示例。点击键盘上的“.”按钮，“.”符号将被添加到 Label 文本的最右端。

点击 Back 按钮，即会删除 Label 文本最右端的一个字符。

点击 Clear 按钮，整个文本将变为“0”。



4）算术运算

在本小节中，我们将创建一个函数，此函数可在 Label 小部件中输入 float 类型的值，并在点击了“=”按钮时执行算术运算。在 btn_clicked_cb() 函数之上添加两个新函数。

```
static void
set_entry_value(float value)
{
    char buf[100];
    sprintf(buf, "%f", value);
    elm_object_text_set(m_ad->entry, buf);
}

static void
btn_equal_clicked()
{
    float value2 = get_entry_value();

    switch( m_ad->calc_mode )
    {
    case ID_PLUS :
        m_ad->value += value2;
        break;
    case ID_MINUS :
```

```
        m_ad->value -= value2;
        break;
    case ID_MULTIPLY :
        m_ad->value *= value2;
        break;
    case ID_DIVIDE :
        m_ad->value /= value2;
        break;
    }

    set_entry_value(m_ad->value);
}
```

set_label_value() 是一种用于接收实数、将该数字转换为字符串、并将此字符串输入 Label 小部件的函数。

btn_equal_clicked() 是一种用于根据算术运算类型使用两个数字执行计算的函数。

向 btn_clicked_cb() 函数添加新代码。

```
        ~
        case ID_BACK :
                text = elm_object_text_get(m_ad->label);
                length = strlen(text);
                if( length > 0 )
                        text = eina_stringshare_add_length(text, length - 1);
                if( strlen(text) < 1 )
                        text = "0";
                elm_object_text_set(m_ad->label, text);
                break;
        case ID_PLUS :
        case ID_MINUS :
        case ID_MULTIPLY :
        case ID_DIVIDE :
                m_ad->value = get_label_value();
                elm_object_text_set(m_ad->label, "0");
                m_ad->calc_mode = id;
                break;
        case ID_EQUAL :
                btn_equal_clicked();
                break;
```

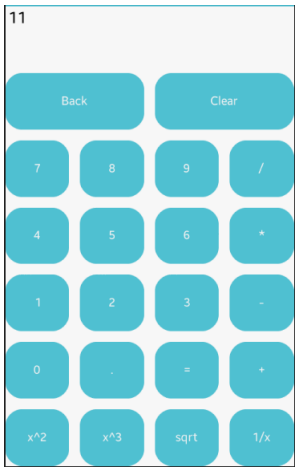
```
        }
}
```

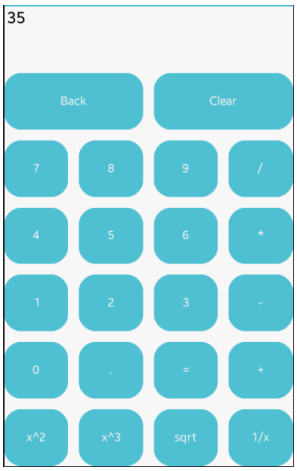
当用户点击 Arithmetic Operations 按钮时，此代码会将 Entry 文本转换为数字，并将其存储在一个全局变量中；

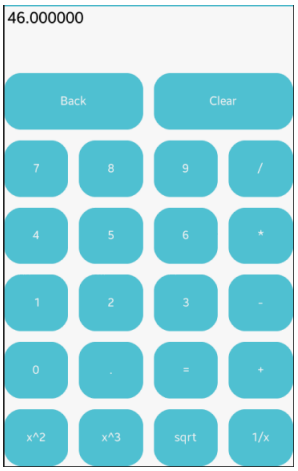
此代码会将 Entry 文本更改为“0”，然后将算术运算类型存储到一个全局变量中。

当用户点击“=”按钮时，此代码将调用 btn_equal_clicked() 函数。

再次运行该示例。然后，输入“11”并点击“+”按钮。数字“0”将显示在 Entry 小部件中。
然后，输入“35”并点击“=”按钮。运算结果将显示在 Entry 小部件中。
我们建议您还对减法、乘法和除法运算进行测试。







5）计算数字的平方和平方根

在本小节中，我们将使用数学函数计算数字的平方和平方根。在 btn_clicked_cb() 函数的结尾添加新代码。

```
        ~
        case ID_EQUAL :
                btn_equal_clicked();
                break;
        case ID_X2 :
                value = get_label_value();
                value = pow( value, 2 );
                set_label_value(value);
                break;
        case ID_X3 :
                value = get_label_value();
                value = pow( value, 3 );
                set_label_value(value);
                break;
        case ID_SQRT :
                value = get_label_value();
                value = sqrt( value );
                set_label_value(value);
                break;
        case ID_RECIPE :
                value = get_label_value();
                value = 1.f / value;
                set_label_value(value);
                break;
        }
}
```

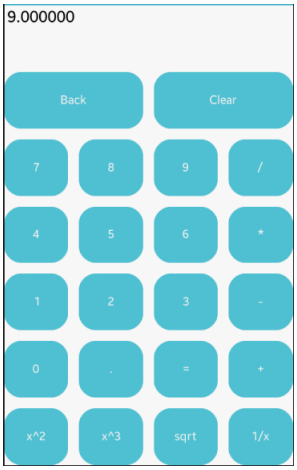
pow(double, double) 是一种用于计算数字平方值的数学函数。将 2 传递给第二个函数，则会返回数字的平方值，而传递 3，则会返回数字的立方值。

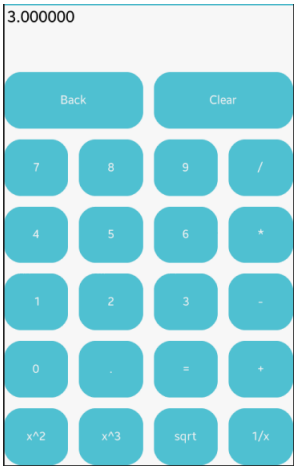
sqrt(double) 是一种用于计算数字的平方根值的数学函数。

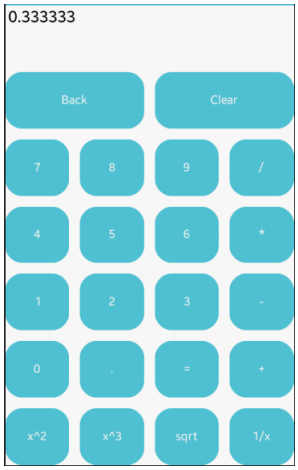
要请求数字的倒数，请用 1 除以该数字。

再次运行该示例。输入数字 3 并点击“x^2”按钮，随即将显示该数字的平方值。

此时点击“sqrt”按钮则会计算所得值的平方根，于是我们将得到原始的输入数值。
点击“1/x”按钮则会显示数字的倒数。







6）相关 API

double pow(double, double)：一种用于计算数字平方值的数学函数的 API。/ 参数：原始数字和乘方数。

double sqrt(double)：一种用于计算数字平方根的 API。/ 参数：原始数字。

31. 在画布上显示渐变色

应用程序开发新手都会问一个问题，那就是：

要在这个领域取得成功，必须具备哪些技能？

答案从来不曾变过，首先以及最重要的一点就是要具备出色的图形/渲染技能。

然后，他们还需具备良好的面向对象编程 (OOP) 的技能。

随着时间的推移，应用程序的开发变得越来越简单。过去，开发人员往往需要通过辛苦的编写代码来实施各项功能。但如今，市面上的各类平台能提供丰富的功能，开发人员只需调用并使用由平台提供的 API 即可。

在开发商用智能手机应用程序时，图形是最难的部分，仍需要开发人员付出巨大努力。如果有两个功能相同的应用程序，用户会选择图形效果更美观的那一个。因此，对于应用程序开发人员来说，最重要的素质就是能够制作出符合用户口味的、丰富多样且令人印象深刻的图形效果。

Evas 是 EFL 中提供的画布。Evas 上绘制的所有形状都是作为对象进行创建的。在此例中，我们将学习如何在此画布上绘制线条。

1) 创建画布，绘制渐变色

创建新的源项目，并将项目名称指定为“DrawGradiation”。创建源项目之后，打开 src 文件夹中的源文件 (*.c)，并在顶部添加新代码，如下所示：

```
typedef struct appdata {
    Evas_Object *win;
    Evas_Object *conform;
    //Evas_Object *label;
    Evas_Object *imgs[5];
} appdata_s;

static Evas_Object *
create_gradient_rect(appdata_s *ad)
```

```
{
   /* Generate gradient data on the fly */
   const int colors[2][4] = {
           /* red to blue */
           { 255, 0, 0, 255 }, { 0, 0, 255, 255 },
   };

   const int b_r = colors[0][0], b_g = colors[0][1], b_b = colors[0][2], b_a =
colors[0][3];
   const int e_r = colors[1][0], e_g = colors[1][1], e_b = colors[1][2], e_a =
colors[1][3];

   Evas_Object *img;
   unsigned int *data32;

   /* Create image object, set its image data size & type */
   Evas* canvas = evas_object_evas_get(ad->win);
   img = evas_object_image_filled_add(canvas);
   /* BGRA data */
   evas_object_image_colorspace_set(img, EVAS_COLORSPACE_ARGB8888);
   /* Size is 255x1 */
   evas_object_image_size_set(img, 255, 1);
   /* Mark image as having alpha */
   evas_object_image_alpha_set(img, EINA_TRUE);

   /* get a writable data pointer */
   data32 = evas_object_image_data_get(img, EINA_TRUE);

   for (unsigned x = 0; x < 255; x++)
   {
       int r, g, b, a;
       /* interpolate alpha */
       a = (b_a * (255 - x) + e_a * x) / (2 * 255);
       /* interpolate red */
       r = (b_r * b_a * (255 - x) + e_r * e_a * (x)) / (2 * 255 * 255);
       /* interpolate green */
       g = (b_g * b_a * (255 - x) + e_g * e_a * (x)) / (2 * 255 * 255);
       /* interpolate blue */
       b = (b_b * b_a * (255 - x) + e_b * e_a * (x)) / (2 * 255 * 255);
       /* write pixel value now */
       data32[x] = (a << 24) | (r << 16) | (g << 8) | b;
   }

   /* very important: set data back */
   evas_object_image_data_set(img, data32);
```

```
    evas_object_size_hint_weight_set(img, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    evas_object_size_hint_align_set(img, EVAS_HINT_FILL, EVAS_HINT_FILL);
    evas_object_show(img);
    return img;
}
```

要在屏幕上绘制图片，您需要使用名为 Evas 的画布。在本小节中，我们将创建五个 Image 对象。被添加到 appdata 结构中的 imgs[5] 是用于存储 Image 对象的数组变量。

create_gradient_rect() 是一种用于创建并返回颜色渐变的 Image 对象的函数。以下是对各项函数的描述：

colors[2][4] 是用于存储两种颜色的数组变量。其中一个是渐变色的开始颜色，另一个是渐变色的结束颜色。一个颜色是由四个数值共同构成的。这些数值依次列出后分别为 Red、Green、Blue 和 Alpha（半透明）。

evas_object_evas_get() 是一种用于创建 Evas 对象的 API。

evas_object_image_filled_add() 是一种用于创建 Image 对象的 API。

evas_object_image_colorspace_set() 是一种用于指定 Image 对象的颜色空间的 API。将 EVAS_COLORSPACE_ARGB8888 传递给第二个参数后，即可让一个像素包含四种不同数据（Red、Green、Blue 和 Alpha）。

evas_object_image_size_set() 是一种用于指定 Image 对象大小的 API。将 255 传递给第二个参数后，可使水平像素值变为 255。将 1 传递给第三个参数后，可使垂直像素值变为 1。

evas_object_image_alpha_set() 是一种用于指定是否将半透明应用于 Image 对象的 API。

evas_object_image_data_get() 是一种用于将 Image 对象的原始数据作为数组返回的 API。

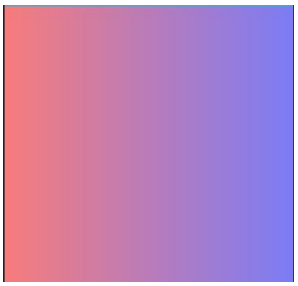
evas_object_image_data_set() 执行与 evas_object_image_data_get() 相反的功能。这是一种用于指定 Image 对象原始数据的 API。

现在我们将使用此函数在屏幕上创建并显示颜色渐变的 Image 对象。向 create_base_gui() 函数添加新代码。此示例中将不会使用 Label，请对此进行注释。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    Evas_Object *box = elm_box_add(ad->conform);
    elm_box_padding_set(box, ELM_SCALE_SIZE(10), ELM_SCALE_SIZE(10));
```

```
    elm_object_content_set(ad->conform, box);
    evas_object_show(box);

    {
        ad->imgs[0] = create_gradient_rect(ad);
        elm_box_pack_end(box, ad->imgs[0]);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

我们已创建一个 Box 对象和一个 Image 对象，并将其显示在屏幕上。运行示例。屏幕上显示一个左侧为红色、右侧为蓝色的颜色渐变方块。



2) 创建多个颜色渐变方块

在本小节中，我们将在上一小节中创建的 create_gradient_rect() 函数中实施一项功能，此功能可创建四种不同的颜色渐变方块。按如下所示修改代码：

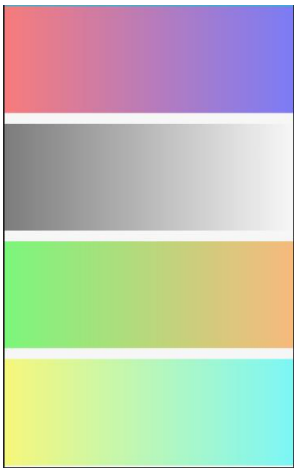
```
static Evas_Object *
create_gradient_rect(appdata_s *ad, unsigned i)
//create_gradient_rect(appdata_s *ad)
{
    /* Generate gradient data on the fly */
    const int colors[8][4] = {
        /* red to blue */
        { 255, 0, 0, 255 }, { 0, 0, 255, 255 },
        /* black to transparent */
        { 0, 0, 0, 255 }, { 0, 0, 0, 0 },
        /* green to orange */
```

```
        { 0, 255, 0, 255 }, { 255, 128, 0, 255 },
        /* yellow to cyan */
        { 255, 255, 0, 255 }, { 0, 255, 255, 255 }
    };
    /*const int colors[2][4] = {
         red to blue
        { 255, 0, 0, 255 }, { 0, 0, 255, 255 },
    };*/

    const int b_r = colors[i*2][0], b_g = colors[i*2][1], b_b = colors[i*2][2],
 b_a = colors[i*2][3];
    const int e_r = colors[i*2+1][0], e_g = colors[i*2+1][1], e_b = colors[i*2+
1][2], e_a = colors[i*2+1][3];

    Evas_Object *img;
    unsigned int *data32;
```

调用上述函数四次，将会生成四个不同的颜色渐变 Image。按如下所示，修改 create_base_gui() 函数的底部内容：

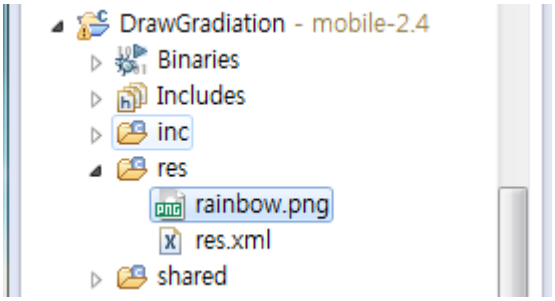
```
    {
        //ad->imgs[0] = create_gradient_rect(ad);
        //elm_box_pack_end(box, ad->imgs[0]);
        for (unsigned i = 0; i < 4; i++)
        {
            ad->imgs[i] = create_gradient_rect(ad, i);
            elm_box_pack_end(box, ad->imgs[i]);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

再次运行该示例。屏幕上显示四个颜色渐变方块。



3) 使用 Image 文件创建彩虹方块

在本小节中，我们将使用 Image 文件在屏幕上显示一个彩虹方块。将附录 / Image 文件夹中的 rainbow.png 文件复制到源项目的 /res 文件夹中。



现在，我们需要让源代码加载 Image 文件并将其分配给 Image 对象。转至源文件并在 create_base_gui() 函数之上添加一个新函数。

```
static Evas_Object *
create_rainbow_rect(appdata_s *ad)
{
    /* A much simpler method for gradients is to simply use an image from disk *
/
    Evas_Object *img;
    char path[PATH_MAX];

    /* Create image object, set its image data size & type */
```

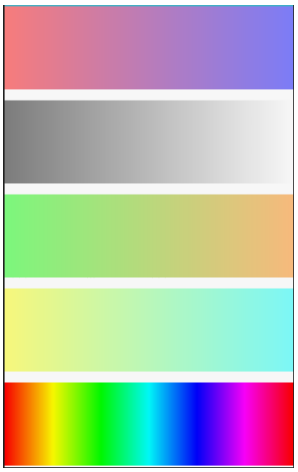
```
    Evas* canvas = evas_object_evas_get(ad->win);
    img = evas_object_image_filled_add(evas_object_evas_get(canvas));

    snprintf(path, sizeof(path), "%s/rainbow.png", app_get_resource_path());
    dlog_print(DLOG_ERROR, LOG_TAG, "path: '%s'", path);
    evas_object_image_file_set(img, path, NULL);

    evas_object_size_hint_weight_set(img, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    evas_object_size_hint_align_set(img, EVAS_HINT_FILL, EVAS_HINT_FILL);
    evas_object_show(img);
    return img;
}
```

evas_object_image_file_set() 是一种可通过指定 Image 对象的 Image 文件路径来加载该文件的 API。

现在我们将使用上述函数在屏幕上加载并显示 Image 文件。向 create_base_gui() 函数添加新代码。

```
    {
        for (unsigned i = 0; i < 4; i++)
        {
            ad->imgs[i] = create_gradient_rect(ad, i);
            elm_box_pack_end(box, ad->imgs[i]);
        }

        ad->imgs[5] = create_rainbow_rect(ad);
        elm_box_pack_end(box, ad->imgs[5]);
    }
```

再次运行该示例，随后将看到一个彩虹方块显示在屏幕底部。



4）相关 API

Evas *evas_object_evas_get(Evas_Object *obj)：一种用于创建 Evas 对象的 API。/ 参数：Window 对象。

evas_object_image_filled_add() 是一种用于创建 Image 对象的 API。

evas_object_image_colorspace_set() 是一种用于指定 Image 对象的颜色空间的 API。将 EVAS_COLORSPACE_ARGB8888 传递给第二个参数后，即可让一个像素包含四种不同数据（Red、Green、Blue 和 Alpha）。

evas_object_image_size_set() 是一种用于指定 Image 对象大小的 API。将 255 传递给第二个参数后，可使水平像素值变为 255。将 1 传递给第三个参数后，可使垂直像素值变为 1。

evas_object_image_alpha_set() 是一种用于指定是否将半透明应用于 Image 对象的 API。

evas_object_image_data_get() 是一种用于将 Image 对象的原始数据作为数组返回的 API。

evas_object_image_data_set() 执行与 evas_object_image_data_get() 相反的功能。这是一种用于指定 Image 对象原始数据的 API。

evas_object_image_file_set() 是一种可通过指定 Image 对象的 Image 文件路径来加载该文件的 API。

32. 在画布上显示方块

要在屏幕上绘制形状，您需要使用画布。Evas 是 EFL 中提供的画布。Evas 上绘制的所有形状都是作为对象进行创建的。在此例中，我们将学习如何在画布上绘制方块。

1) 创建画布，绘制方块

创建新的源项目，并将项目名称指定为“DrawRect”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并向 create_base_gui() 函数添加新代码。此示例中将不会使用 Label，请对此进行注释。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    /* Label*/
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
    elm_object_content_set(ad->conform, ad->label);*/

    { /* child object - indent to how relationship */
        /* A grid to stretch content within grid size */
        Evas_Object *grid = elm_grid_add(ad->win);
        evas_object_size_hint_weight_set(grid, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        elm_object_content_set(ad->conform, grid);
        evas_object_show(grid);

        {
            /* Canvas */
            Evas* canvas = evas_object_evas_get(ad->win);
```

```
            /* Rect-1 */
            Evas_Object *rect = evas_object_rectangle_add(canvas);
            evas_object_color_set(rect, 255, 0, 0, 192);
            evas_object_show(rect);
            elm_grid_pack(grid, rect, 4, 5, 52, 31);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

elm_grid_add() 是一种用于创建 Grid 容器的 API。Grid 容器支持根据屏幕高宽比在屏幕上放置对象。它与 Table 的区别在于：Table 是按单元格来划分区域，默认情况下将最大值指定为 100。

elm_grid_pack() 是一种用于在 Grid 容器中放置对象的 API。第一个参数指示 Grid；第二个参数指示此对象；第三个参数指示水平位置；第四个参数指示垂直位置；第五个参数指示宽度；第六个参数指示高度。将 4 传递给第三个参数后，即可将对象放置在水平方向上 4% 的位置。

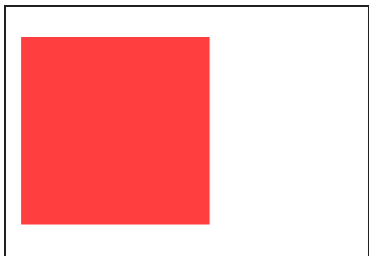
elm_grid_size_set(obj, w, h) 是一种用于指定 Grid 大小的 API。Grid 的默认大小为：宽 100，高 100。

evas_object_evas_get(Evas_Object *) 是一种用于创建 Evas 对象的 API。

evas_object_rectangle_add(Evas *) 是一种用于在画布上创建 Rectangle 对象的 API。

evas_object_color_set(Evas_Object *, int, int, int, int) 是一种用于指定形状颜色的 API。这些参数依次列出后分别为 Red、Green、Blue 和 semi-transparency。输入 255、0、0、192，则会创建一个半透明的红色方形。

构建并运行该示例。屏幕上显示一个粉红色方块。我们为方块指定了红色。但是，由于已向方块应用了半透明度，因此看起来呈粉红色。



2) 叠加两个半透明方块

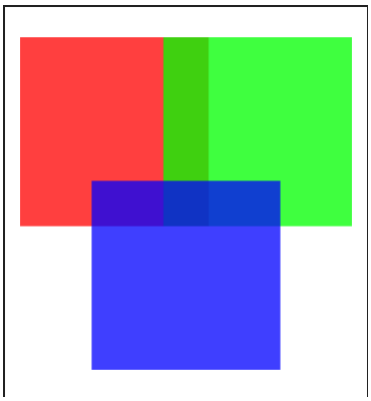
在本小节中，我们将了解当添加两个方块并将之叠加时颜色会发生什么变化。向 create_base_gui() 函数添加新代码。

```
            /* Rect-1 */
            Evas_Object *rect = evas_object_rectangle_add(canvas);
            evas_object_color_set(rect, 255, 0, 0, 192);
            evas_object_show(rect);
            elm_grid_pack(grid, rect, 4, 5, 52, 31);

            /* Rect-2 */
            rect = evas_object_rectangle_add(canvas);
            evas_object_color_set(rect, 0, 255, 0, 192);
            evas_object_show(rect);
            elm_grid_pack(grid, rect, 44, 5, 52, 31);

            /* Rect-3 */
            rect = evas_object_rectangle_add(canvas);
            evas_object_color_set(rect, 0, 0, 255, 192);
            evas_object_show(rect);
            elm_grid_pack(grid, rect, 24, 29, 52, 31);
        }
    }
```

我们将第二个方块的颜色指定为半透明绿色。

我们将第三个方块的颜色指定为半透明蓝色。

再次运行该示例。屏幕上显示出这三个方块。
由于应用了半透明度设置，因此叠加区域的颜色会变成对应的中间色。



3) 相关 API

Evas *evas_object_evas_get(Evas_Object *obj)：一种用于创建 Evas 对象的 API。/ 参数：Window 对象。

Evas_Object *evas_object_rectangle_add(Evas *e)：一种用于在画布上创建 Rectangle 对象的 API。

void evas_object_color_set(Evas_Object *obj, int r, int g, int b, int a)：一种用于指定形状颜色的 API。/ 参数：shape 对象、Red、Green、Blue 和 semi-transparency。允许的颜色值范围为 0 - 255。例如，如果您想创建 Yellow，请输入 255、255、0、255。

33. 在画布上显示多边形

要在屏幕上绘制形状，您需要使用画布。Evas 是 EFL 中提供的画布。Evas 上绘制的所有形状都是作为对象进行创建的。在此例中，我们将学习如何在画布上绘制多边形。

1) 创建画布，绘制三角形

创建新的源项目，并将项目名称指定为“DrawPolygon”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并向 create_base_gui() 函数添加新代码。此示例中将不会使用 Label，请对此进行注释。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    /* Label*/
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
    elm_object_content_set(ad->conform, ad->label);*/

    {
        /* Canvas */
        Evas* canvas = evas_object_evas_get(ad->win);

        /* Polygon - Triangle */
        Evas_Object *polygon = evas_object_polygon_add(canvas);
        evas_object_polygon_point_add(polygon, 20, 50);
        evas_object_polygon_point_add(polygon, 170, 150);
        evas_object_polygon_point_add(polygon, 20, 250);
        evas_object_color_set(polygon, 255, 200, 0, 255);
        evas_object_show(polygon);
    }
```

```
    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

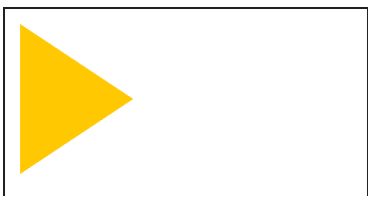
evas_object_evas_get(Evas_Object *) 是一种用于创建 Evas 对象的 API。

evas_object_polygon_add(Evas *) 是一种用于在画布上创建 Polygon 对象的 API。

evas_object_polygon_point_add(Evas_Object *, Evas_Coord, Evas_Coord) 是一种用于向 Polygon 对象添加点坐标的 API。Polygon 至少必须有三个点。第一个参数指示 Polygon 对象；第二个参数指示 X 坐标；第三个参数指示 Y 坐标。

evas_object_color_set(Evas_Object *, int, int, int, int) 是一种用于指定形状颜色的 API。这些参数依次列出后分别为 shape 对象、Red、Green、Blue 和 semi-transparency。

构建并运行该示例。屏幕上显示一个黄色三角形。



2) 显示 Pentagon

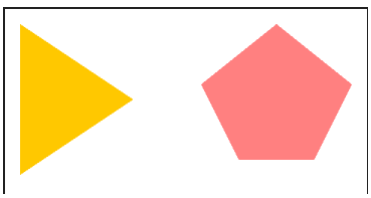
向 Polygon 对象添加四个点，即可创建一个方形，若添加五个点，则可创建一个多边形。向 create_base_gui() 函数添加新代码。

```
    /* Polygon - Triangle */
    Evas_Object *polygon = evas_object_polygon_add(canvas);
    evas_object_polygon_point_add(polygon, 20, 50);
    evas_object_polygon_point_add(polygon, 170, 150);
    evas_object_polygon_point_add(polygon, 20, 250);
```

```
        evas_object_color_set(polygon, 255, 200, 0, 255);
        evas_object_show(polygon);

        /* Polygon - Pentagon */
        polygon = evas_object_polygon_add(canvas);
        evas_object_polygon_point_add(polygon, 360, 50);
        evas_object_polygon_point_add(polygon, 460, 130);
        evas_object_polygon_point_add(polygon, 410, 230);
        evas_object_polygon_point_add(polygon, 310, 230);
        evas_object_polygon_point_add(polygon, 260, 130);
        evas_object_color_set(polygon, 255, 128, 128, 255);
        evas_object_show(polygon);
    }
```

我们创建了一个新的 Polygon 对象，并向其添加了五个点。

再次运行该示例，随后将看到一个粉红色多边形显示在屏幕上。



3) 使用 Polygon 绘制正多边形

正多边形是指夹角角度均相等且每条边的长度都相同的形状，比如正方形和正六边形。在本小节中，我们将创建一个用于创建正多边形的函数。由于我们将使用数学函数，因此需要在源文件之上包含库。

```
#include "drawpolygon.h"
#include <math.h>
```

在 create_base_gui() 函数之上创建一个新函数。此函数用于创建正多边形。

```
static Evas_Object*
crate_circle(Evas* canvas, int x, int y, int radius, int r, int g, int b, int a,
 int edge_count)
{
        int x1, y1, x2, y2;
        float angle=0.f;

        Evas_Object *polygon = evas_object_polygon_add(canvas);

        for(int i=0; i < edge_count; i++)
        {
                angle = (M_PI * 2) / (float)edge_count * i;
                x1 = sin(angle) * radius + x;
                y1 = cos(angle) * radius + y;
                evas_object_polygon_point_add(polygon, x1, y1);
        }
        evas_object_color_set(polygon, r, g, b, a);
        evas_object_show(polygon);
        return polygon;
}
```

M_PI 是一个包含 Pi 值的常量。

sin(double) 是一种用于计算正弦的 API。角度单位为 Pi。例如，要将正弦指定为 180 度，请传递 Pi，要将正弦指定为 90 度，请传递 Pi /2。

cos(double) 是一种用于计算余弦的 API。角度单位为 Pi。

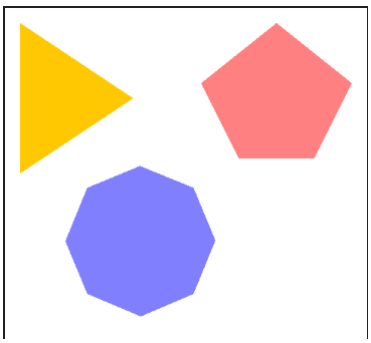
现在我们将使用上述函数创建正八边形。在 create_base_gui() 函数的结尾添加一行新代码。

```
        /* Polygon - Pentagon */
        polygon = evas_object_polygon_add(canvas);
        evas_object_polygon_point_add(polygon, 360, 50);
        evas_object_polygon_point_add(polygon, 460, 130);
        evas_object_polygon_point_add(polygon, 410, 230);
        evas_object_polygon_point_add(polygon, 310, 230);
        evas_object_polygon_point_add(polygon, 260, 130);
        evas_object_color_set(polygon, 255, 128, 128, 255);
        evas_object_show(polygon);

        /* Polygon - 10 */
```

```
    crate_circle(canvas, 180, 340, 100, 128, 128, 255, 255, 8);
}
```

create_circle() 函数的参数依次列出后分别为：Evas 对象、中心 x 坐标、中心 y 坐标、半径、Red、Green、Blue、semi-transparent 色以及点数。

再次运行该示例，随后将看到一个紫色正八边形显示在屏幕上。



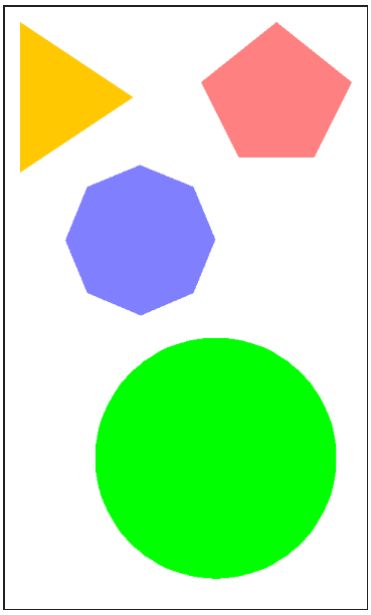
4）使用 Polygon 小部件绘制圆形

如果增加正多边形的点数，最终生成的形状看起来会像一个圆形。在 create_base_gui() 函数的结尾添加一行新代码。

```
    /* Polygon - 10 */
    crate_circle(canvas, 180, 340, 100, 128, 128, 255, 255, 8);

    /* Polygon - Circle */
    crate_circle(canvas, 280, 600, 160, 0, 255, 0, 255, 90);
}
```

我们向多边形添加了 90 个点。现在我们将了解这样增加点数会有什么样的结果。

再次运行该示例，随后将看到一个黄绿色的正 90 度多边形显示在屏幕上。由于此多边形的点数很多，因此看起来像是一个圆形。



5) 相关 API

Evas *evas_object_evas_get(Evas_Object *obj)：一种用于创建 Evas 对象的 API。/ 参数：Window 对象。

Evas_Object *evas_object_polygon_add(Evas *e)：一种用于在画布上创建 Polygon 对象的 API。

void evas_object_polygon_point_add(Evas_Object *obj, Evas_Coord x, Evas_Coord y)：一种用于向 Polygon 对象添加点坐标的 API。/ 参数：Polygon、x 坐标和 y 坐标。

void evas_object_color_set(Evas_Object *obj, int r, int g, int b, int a)：一种用于指定形状颜色的 API。/ 参数：shape 对象、Red、Green、Blue 和 semi-transparency。允许的颜色值范围为 0 - 255。例如，如果您想创建 Yellow，请输入 255、255、0、255。

34. 在画布上显示文本

要在屏幕上绘制形状，您需要使用画布。Evas 是 EFL 中提供的画布。Evas 上绘制的所有形状都是作为对象进行创建的。在此例中，我们将学习如何在画布上显示字符串。

1) 在画布上显示文本

创建新的源项目，并将项目名称指定为“CanvasTextColor”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在 create_base_gui() 函数之上添加一个文本对象创建函数。

```
// Create Text object
static Evas_Object *
create_text(Evas *canvas, Evas_Object *grid,
            Evas_Coord x, Evas_Coord y, Evas_Coord w, Evas_Coord h,
            const char *str, int font_size,
            int r, int g, int b, int a)
{
   Evas_Object *text = evas_object_text_add(canvas);
   evas_object_text_text_set(text, str);
   evas_object_text_font_set(text, "DejaVu", font_size);
   evas_object_color_set(text, r, g, b, a);
   elm_grid_pack(grid, text, x, y, w, h);
   evas_object_show(text);
}
```

evas_object_text_add(Evas *) 是一种用于在画布上创建文本对象的 API。

evas_object_text_text_set(Evas_Object *, char *) 是一种用于为文本对象指定字符串的 API。

evas_object_text_font_set(Evas_Object *, char *, Evas_Font_Size) 是一种用于为文本对象指定字体的 API。第一个参数指示文本对象；第二个参数指示字体类型；第三个参数指示字体大小。

现在，我们将使用刚刚创建的函数在画布上显示文本。向 create_base_gui() 函数添加新代码。此示例中将不会使用 Label，请对此进行注释。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    /* Label*/
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
    elm_object_content_set(ad->conform, ad->label);*/

    /* child object - indent to how relationship */
    /* A grid to stretch content within grid size */
    Evas_Object *grid = elm_grid_add(ad->win);
    elm_grid_size_set(grid, 480, 800);
    evas_object_size_hint_weight_set(grid, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_object_content_set(ad->conform, grid);
    evas_object_show(grid);

    { /* child object - indent to how relationship */
        Evas* canvas = evas_object_evas_get(ad->win);

        create_text(canvas, grid, 50, 100, 300, 100,
                "Hello World!", 60, 80, 80, 255, 255);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

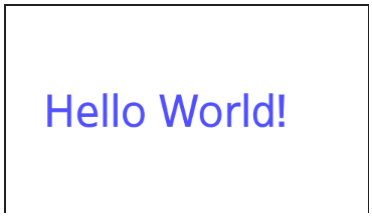
我们创建了一个 Grid 容器，用于将相对坐标分配给文本对象。

elm_grid_size_set() 是一种用于指定 Grid 大小的 API。Grid 的默认宽度为 100，默认高度也是 100。在上述代码中，我们已将 Grid 的大小更改为 480 x 800。这项新设置的定位精度比默认设置更好。

使用 evas_object_evas_get() 函数创建 Evas 对象。

使用 create_text() 函数创建文本对象。参数依次列出后分别为：画布对象、Grid、x 坐标、y 坐标、宽度、高度、文本字符串、字体大小、Red、Green、Blue 以及半透明颜色。

构建并运行该示例。屏幕上显示紫色的“Hello World”文本。



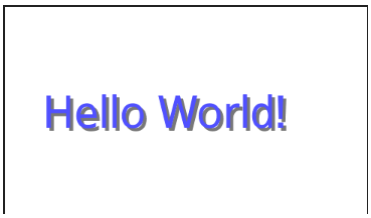
2) 向文本应用阴影

要对文本应用阴影效果，您需要在为文本分两次指定不同颜色和位置之后显示两次设置的文本。向 create_base_gui() 函数添加新代码。我们用与第一次设置的文本相同的内容和大小创建了第二个文本。

```
    { /* child object - indent to how relationship */
        Evas* canvas = evas_object_evas_get(ad->win);

        create_text(canvas, grid, 54, 104, 300, 100,
                "Hello World!", 60, 120, 120, 120, 255);

        create_text(canvas, grid, 50, 100, 300, 100,
                "Hello World!", 60, 80, 80, 255, 255);
    }
```

第二个文本将成为阴影，并将先于第一次设置的文本显示。同时，第二个文本的位置也会不同，其颜色为灰色。

再次运行该示例。阴影效果已经应用到文本。



3）相关 API

Evas *evas_object_evas_get(Evas_Object *obj)：一种用于创建 Evas 对象的 API。/ 参数：Window 对象。

Evas_Object *evas_object_text_add(Evas *e)：一种用于在画布上创建文本对象的 API。

void evas_object_text_text_set(Evas_Object *obj, const char *text)：一种用于为文本对象指定字符串的 API。

void evas_object_text_font_set(Evas_Object *obj, const char *font, Evas_Font_Size size)：一种用于为文本对象指定字体的 API。/ 参数：文本对象、字体类型以及字体大小。

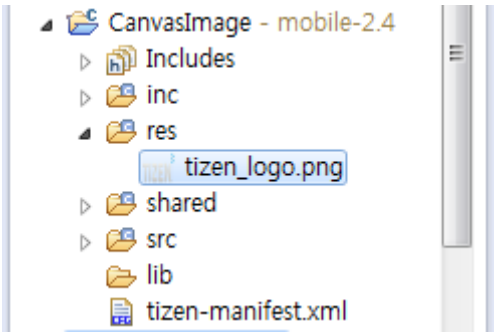
void evas_object_color_set(Evas_Object *obj, int r, int g, int b, int a)：一种用于指定形状颜色的 API。/ 参数：shape 对象、Red、Green、Blue 和 semi-transparency。允许的颜色值范围为 0 - 255。例如，如果您想创建 Yellow，请输入 255、255、0、255。

35. 在画布上显示图像

要在屏幕上绘制形状，您需要使用画布。Evas 是 EFL 中提供的画布。Evas 上绘制的所有形状都是作为对象进行创建的。在此例中，我们将学习如何在画布上显示图像。

1) 在画布上显示图像

创建新的源项目，并将项目名称指定为“CanvasImage”。在此例中，我们需要使用图像文件。将附录 /Image 文件夹中的 tizen_logo.png 文件复制到源项目的 /res 文件夹中。



要使用我们刚刚复制的图像文件，我们需要查找该图像文件的绝对路径。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并向 create_base_gui() 函数添加新函数。此函数将返回 /res 文件夹中存储的文件的绝对路径。

```
static void
app_get_resource(const char *res_file_in, char *res_path_out, int res_path_max)
{
	char *res_path = app_get_resource_path();
	if (res_path) {
		snprintf(res_path_out, res_path_max, "%s%s", res_path, res_file_
in);
		free(res_path);
	}
}
```

返回到 create_base_gui() 函数并添加新代码。此代码用于创建画布，并加载图像文件，以创建 Image 对象。对用于创建 Label 的代码标注注释。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    /* Label*/
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
    elm_object_content_set(ad->conform, ad->label);*/

    { /* child object - indent to how relationship */
        /* A grid to stretch content within grid size */
        Evas_Object *grid = elm_grid_add(ad->win);
        elm_grid_size_set(grid, 480, 800);
        evas_object_size_hint_weight_set(grid, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        elm_object_content_set(ad->conform, grid);
        evas_object_show(grid);

        {
            /* Canvas */
            Evas* canvas = evas_object_evas_get(ad->win);

            char img_path[PATH_MAX] = "";
            app_get_resource("tizen_logo.png", img_path, PATH_MAX);

            /* Image-1 */
            Evas_Object *img = evas_object_image_filled_add(canvas);
            evas_object_image_file_set(img, img_path, NULL);
            elm_grid_pack(grid, img, 40, 10, 400, 280);
            evas_object_show(img);
        }
    }

    /* Show window after base gui is set up */
```

```
    evas_object_show(ad->win);
```

我们创建了一个 Grid 容器，用于将相对坐标分配给 Image 对象。要将更精确的坐标分配给 Image 对象，我们使用 elm_grid_size_set() 函数将 Grid 的大小指定为 480x800。

evas_object_image_filled_add(Evas *) 是一种用于创建可令原始图像填满整个区域空间的 Image 对象的 API。

evas_object_image_file_set(Evas_Object *, char *, char *) 是一种用于将图像文件加载到 Image 对象的 API。

构建并运行该示例。屏幕上显示一个图像。此时您将看到原始图像呈垂直拉伸状态显示，水平方向比例正常。

2) 以平铺格式在画布上显示图像

在本小节中，我们将学习如何采用平铺格式在给定的区域内放置图像。在 create_base_gui() 函数的结尾添加新代码。此代码将创建第二个 Image 对象。

```
        /* Image-1 */
        Evas_Object *img = evas_object_image_filled_add(canvas);
        evas_object_image_file_set(img, img_path, NULL);
        elm_grid_pack(grid, img, 40, 10, 400, 280);
        evas_object_show(img);

        /* Image-2 */
        int w, h;
        img = evas_object_image_add(canvas);
```

```
            evas_object_image_file_set(img, img_path, NULL);
            evas_object_image_size_get(img, &w, &h);
            evas_object_image_fill_set(img, 110, 37, w, h);
            elm_grid_pack(grid, img, 40, 310, 400, 280);
            evas_object_show(img);
        }
    }
```

evas_object_image_add(Evas *) 是一种用于创建以平铺格式显示图像的 Image 对象的 API。需要指定图像的大小和起始位置。

evas_object_image_size_get(const Evas_Object *, int *, int *) 是一种用于返回原始图像大小的 API。第一个参数接收 Image 对象；第二个参数接收原始图像的宽度；第三个参数接收原始图像的高度。

evas_object_image_fill_set(Evas_Object *, Evas_Coord, Evas_Coord, Evas_Coord, Evas_Coord) 是一种用于为 Image 对象指定原始图像的显示位置和大小的 API。参数依次列出后分别为：Image 对象、x 坐标、y 坐标、宽度以及高度。

再次运行该示例。图像采用平铺格式放置。

3）以尽可能大的尺寸显示图像并保留原始比例

在本小节中，我们将学习如何在给定区域以尽可能大的尺寸显示图像。要在显示图像时保留其原始比例，您需要执行一项计算。在 create_base_gui() 函数之上创建一个新函数。此函数通过计算将要显示的图像大小来创建一个 Image 对象。

```
// Create IMAGE object. Source image's rate is no change
static Evas_Object *
create_image(Evas *canvas, Evas_Object *grid, const char *img_path, int x, int y,
 int w, int h)
{
   int source_w, source_h, new_x, new_y, new_w, new_h;
   float rate_h, rate_v, rate;

   // Create IMAGE object
   Evas_Object *img = evas_object_image_add(canvas);
   // Set source image file
   evas_object_image_file_set(img, img_path, NULL);
   // Get source image size
   evas_object_image_size_get(img, &source_w, &source_h);
   // Load failed - zero sized image
   if ((source_w == 0) || (source_h == 0))
     {
        evas_object_del(img);
        return NULL;
     }

   // Calculage Zoom rate
   rate_h = (float)w / (float)source_w;
   rate_v = (float)h / (float)source_h;
   rate = (rate_h < rate_v) ? rate_h : rate_v;

   // Calculate output image size
   new_w = source_w * rate;
   new_h = source_h * rate;
   evas_object_image_fill_set(img, 0, 0, new_w, new_h);

   // Calculate output Image position
   new_x = x + (w - new_w) / 2;
   new_y = y + (h - new_h) / 2;
   elm_grid_pack(grid, img, new_x, new_y, new_w, new_h);

   evas_object_show(img);
```

```
   return img;
}
```

现在，让我们通过调用刚刚创建的函数创建第三个 Image 对象。在 create_base_gui() 函数的结尾添加新代码。

```
            /* Image-2 */
            int w, h;
            img = evas_object_image_add(canvas);
            evas_object_image_file_set(img, img_path, NULL);
            evas_object_image_size_get(img, &w, &h);
            evas_object_image_fill_set(img, 110, 37, w, h);
            elm_grid_pack(grid, img, 40, 310, 400, 280);
            evas_object_show(img);

            /* Image-3 */
            create_image(canvas, grid, img_path, 40, 610, 400, 180);
        }
    }
```

create_image() 函数的参数依次列出后分别为：画布对象、图像文件路径、x 坐标、y 坐标、宽度以及高度。

再次运行该示例。第三个图像显示在屏幕底部，并保留了原始图像的原始比例。

4）相关 API

Evas *evas_object_evas_get(Evas_Object *obj)：一种用于创建 Evas 对象的 API。/ 参数：Window 对象。

Evas_Object *evas_object_image_filled_add(Evas *e)：一种用于创建可令原始图像填满整个区域空间的 Image 对象的 API。

void evas_object_image_file_set(Evas_Object *obj, const char *file, const char *key)：一种用于将图像文件加载到 Image 对象的 API。

Evas_Object *evas_object_image_add(Evas *e)：一种用于创建以平铺格式显示图像的 Image 对象的 API。需要指定图像的大小和起始位置。

void evas_object_image_size_get(const Evas_Object *obj, int *w, int *h)：一种用于返回原始图像大小的 API。第一个参数接收 Image 对象；第二个参数接收原始图像的宽度；第三个参数接收原始图像的高度。

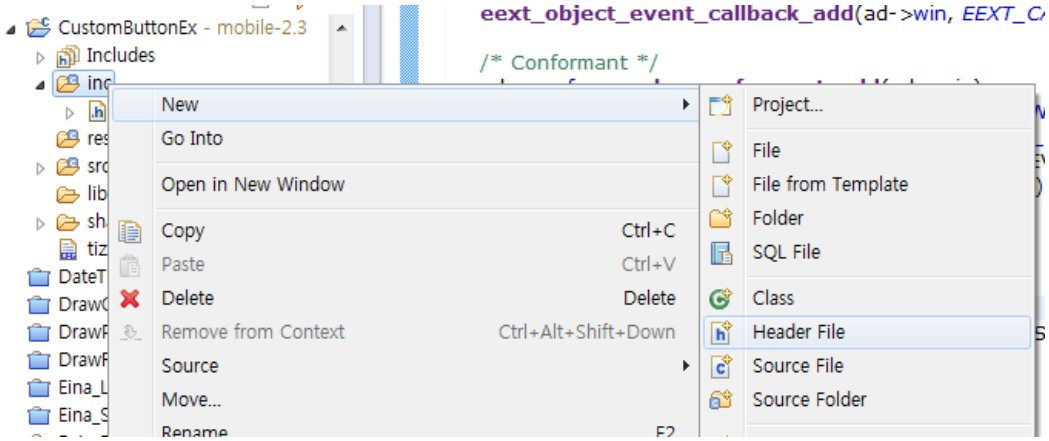
void evas_object_image_fill_set(Evas_Object *obj, Evas_Coord x, Evas_Coord y, Evas_Coord w, Evas_Coord h)：一种用于为 Image 对象指定原始图像的显示位置和大小的 API。/ 参数：Image 对象、x 坐标、y 坐标、宽度以及高度。

36. 创建自定义按钮

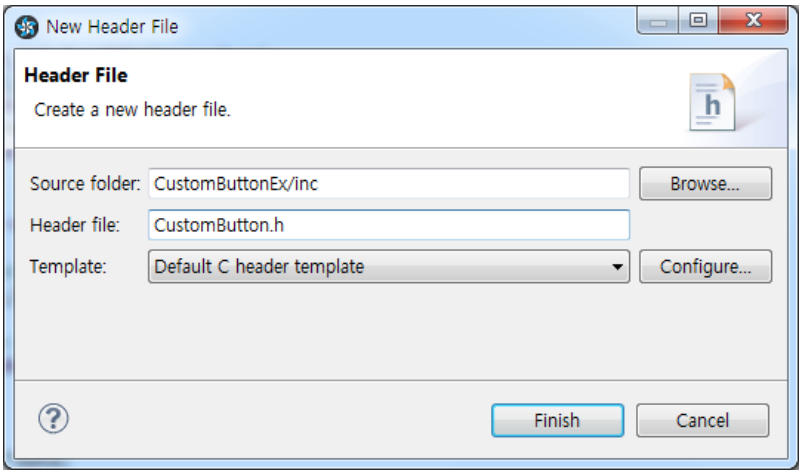
系统提供的基本小部件往往不足以满足用户的多样化需求。对于这类情况，您必需自己创建自定义的小部件。在此例中，我们将学习如何创建自定义的 Button 小部件。

1）在画布上显示文本

创建新的源项目，并将项目名称指定为“CustomButtonEx”。创建源项目之后，为自定义 Button 创建库文件：右键单击 /inc 文件夹，然后在快捷菜单中选择 [New > Header File]。



弹出窗口出现后，请在头文件字段中输入 CustomButton.h，然后点击 Finish 按钮。



现在我们将实施一项功能以在自定义小部件中显示背景方块。创建 /inc/CustomButton.h 文件后，请向其中添加源代码。此代码用于声明库头文件、定义数据结构，以及创建背景方块。

```
#ifndef CUSTOMBUTTON_H_
#define CUSTOMBUTTON_H_

#include <app.h>
#include <Elementary.h>
#include <system_settings.h>
#include <efl_extension.h>
#include <dlog.h>

typedef struct buttondata {
        Evas_Object *rect;
        Evas_Object *text;
} buttondata_s;

Evas_Object*
create_rect(Evas* canvas, Evas_Object* grid, int x, int y, int w, int h,
                int r, int g, int b, int a)
{
        Evas_Object *rect = evas_object_rectangle_add(canvas);
        evas_object_color_set(rect, r, g, b, a);
        elm_grid_pack(grid, rect, x, y, w, h);
        evas_object_show(rect);
        return rect;
}

buttondata_s*
```

```
create_button(Evas* canvas, Evas_Object* grid, Evas_Coord x, Evas_Coord y, Evas
_Coord w, Evas_Coord h,
                const char* str, Evas_Object_Event_Cb func)
{
        buttondata_s* bd = (buttondata_s*)malloc( sizeof( buttondata_s) );

        /* Rectangle */
        bd->rect = create_rect(canvas, grid, x, y, w, h, 128, 128, 255, 255);

        return bd;
}

#endif /* CUSTOMBUTTON_H_ */
```

buttondata 是自定义小部件中使用的一种数据结构。rect 指示某个区域中的小部件坐标。文本则用于存储标题文本字符串。

create_rect() 是一种用于在画布上创建 Rect 对象的函数。参数依次列出后分别为：画布对象、x 坐标、y 坐标、宽度、高度、Red、Green、Blue 以及半透明颜色。

evas_object_rectangle_add(Evas *) 是一种用于在画布上创建 Rectangle 对象的 API。

evas_object_color_set(Evas_Object *, int, int, int, int) 是一种用于指定形状颜色的 API。这些参数依次列出后分别为 Red、Green、Blue 和 semi-transparency。输入 255、0、0、192，则会创建一个半透明的红色方形。

create_button() 是一种用于创建自定义 Button 小部件的函数。我们需要做到能从外部调用此函数。参数依次列出后分别为：画布、x 坐标、y 坐标、宽度、高度、标题文本以及回调事件函数名称。

malloc(size_t) 是一种用于获取并返回指定内存大小的 API。

sizeof() 是一种用于请求并返回对象的内存大小的 API。

现在，让我们在主屏幕上创建一个自定义的小部件对象。打开 src 文件夹中的源文件 (~.c)，并添加新代码。

```
#include "custombuttonex.h"
#include "custombutton.h"

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
} appdata_s;

buttondata_s* m_bd1;
```

将自定义 Button 小部件文件包含进来，并将 buttondata 结构创建为全局变量。

然后，转至 create_base_gui() 函数并添加新代码。此代码将创建一个画布对象和一个自定义 Button 小部件。对用于创建 Label 的代码标注注释。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    /* Label*/
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
    elm_object_content_set(ad->conform, ad->label);*/

    { /* child object - indent to how relationship */
        /* A grid to stretch content within grid size */
        Evas_Object *grid = elm_grid_add(ad->win);
        elm_grid_size_set(grid, 480, 800);
        evas_object_size_hint_weight_set(grid, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        elm_object_content_set(ad->conform, grid);
```

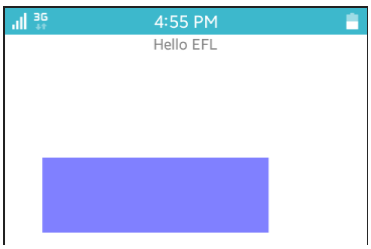
```
        evas_object_show(grid);

        {
            /* Canvas */
            Evas* canvas = evas_object_evas_get(ad->conform);

            /* Custom Button-1 */
            m_bd1 = create_button(canvas, grid, 50, 200, 300, 100, "Button-1",
NULL);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

我们创建了一个 Grid 容器，用于将相对坐标分配给对象。我们使用 elm_grid_size_set() 函数将 Grid 的大小更改成 480x800。

evas_object_evas_get(Evas_Object *) 是一种用于创建 Evas 对象的 API。

create_button() 函数用于实际创建我们刚刚在自定义 Button 小部件文件 (CustomButton.h) 中创建的自定义 Button。

构建并运行该示例。屏幕上显示一个天蓝色方块。这是自定义 Button 的背景方块。



2) 显示标题文本

在本小节中，我们将在方块背景上显示标题文本。为了显示文本，我们将使用 TextBlock。TextBlock 是一个画布对象，以便使用 HTML 标记指定文本属性，比如字体大小和颜色。返回至 CustomButton.h 文件并在 create_button() 函数之上添加新函数。

```
// Create TextBlock object
Evas_Object*
create_textblock(Evas* canvas, Evas_Object* grid, Evas_Coord x, Evas_Coord y, Evas_Coord w, Evas_Coord h, const char* str)
{
    Evas_Object *textblock = evas_object_textblock_add(canvas);
    elm_grid_pack(grid, textblock, x, y, w, h);

    Evas_Textblock_Style *st = evas_textblock_style_new();
    evas_textblock_style_set(st, "DEFAULT='font=Sans font_size=50 color=#eee wrap=mixed align=center'");
    evas_object_textblock_style_set(textblock, st);
    evas_textblock_style_free(st);
    evas_object_textblock_text_markup_set(textblock, str);
    evas_object_show(textblock);

    return textblock;
}
```

evas_object_textblock_add(Evas *) 是一种用于创建 TextBlock 对象的 API。

Evas_Textblock_Style 是一种用于存储 TextBlock 的属性信息的样式结构。

evas_textblock_style_new() 是一种用于创建 Evas_Textblock_Style 对象的 API。

evas_textblock_style_set(Evas_Textblock_Style *, char *) 是一种用于指定 Evas_Textblock_Style 的样式的 API。将 Evas_Textblock_Style 对象传递给第一个参数。对于第二个参数，请使用 HTML 标记指定文本属性。
font=Sans 是用于将字体指定为 Sans serif/Gothic 的标记。
font_size=50 是用于将字体大小指定为 50 的标记。

color=#eee 是用于将字体颜色指定为灰色的标记。
align=center 是用于指定水平居中对齐的标记。

evas_object_textblock_style_set(Evas_Object *, Evas_Textblock_Style *) 是一种用于指定 TextBlock 对象的样式的 API。将 TextBlock 对象传递给第一个参数，将 Evas_Textblock_Style 对象传递给第二个参数。

evas_textblock_style_free(Evas_Textblock_Style *) 是一种用于删除 Evas_Textblock_Style 对象的 API。

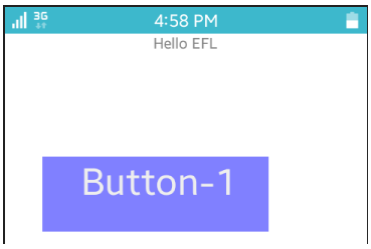
evas_object_textblock_text_markup_set(Evas_Object *, const char *) 是一种用于为 TextBlock 对象指定文本字符串的 API。

我们需要让 create_textblock() 函数来调用此函数。向 create_textblock() 函数添加新代码。

```
        bd->rect = create_rect(canvas, grid, x, y, w, h, 128, 128, 255, 255);

        /* Text */
        bd->text = create_textblock(canvas, grid, x, y, w, h, str);

        return bd;
```

create_textblock() 函数的参数依次列出后分别为：画布对象、x 坐标、y 坐标、宽度、高度以及文本字符串。

再次运行该示例。标题文本显示在方块区域。



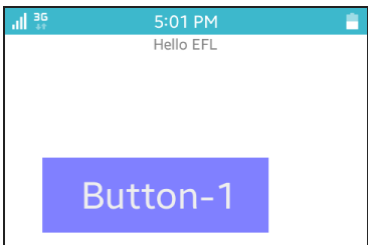
3）居中对齐标题文本

目前，标题文本显示在方块的顶端位置。接下来我们将把标题文本的位置改为居中。向 create_textblock() 函数添加新代码。

```
        Evas_Textblock_Style *st = evas_textblock_style_new();
        evas_textblock_style_set(st, "DEFAULT='font=Sans font_size=50 color=#eee
wrap=mixed align=center'");
        evas_object_textblock_style_set(textblock, st);
        evas_textblock_style_free(st);
        evas_object_textblock_valign_set(textblock, 0.5);
        evas_object_textblock_text_markup_set(textblock, str);
        evas_object_show(textblock);
```

evas_object_textblock_valign_set(Evas_Object *, double) 是一种指定垂直对齐 TextBlock 标题文本的 API。将 0 传递给第二个参数，会将文本靠上对齐，传递 1 则会将文本靠下对齐，传递 0.5 则会将文本居中对齐。

再次运行该示例。此次，标题文本居中显示。



4）请求 Button 单击事件

在本小节中，我们将实施一项功能，以便在点击自定义 Button 后更改背景色。要执行此操作，我们需要请求 Touch 事件。在 create_button() 函数的结尾添加新代码。

```
        bd->text = create_textblock(canvas, grid, x, y, w, h, str);

        evas_object_event_callback_add( bd->text, EVAS_CALLBACK_MOUSE_DOWN, on_
```

```
mouse_down, (void*)bd);
        evas_object_event_callback_add( bd->text, EVAS_CALLBACK_MOUSE_UP, on_mo
use_up, (void*)bd);

        return bd;
└──────────────────────────────────────────┘
```

此代码会在 TextBlock 中发生 Touch Down 事件时调用 on_mouse_down 函数。

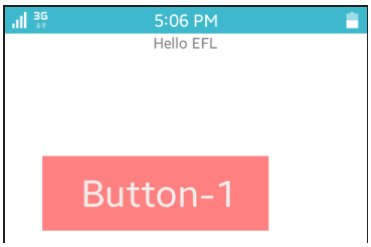
此代码会在 TextBlock 中发生 Touch Cancel 事件时调用 on_mouse_up 函数。

现在，我们将创建一个回调函数。在 create_button()) 函数之上添加两个新函数。

```
// Touch Down event callback
void
on_mouse_down(void *data, Evas *e, Evas_Object *button, void *event_info)
{
        buttondata_s* bd = (buttondata_s*)data;
        evas_object_color_set(bd->rect, 255, 128, 128, 255);
}

// Touch Up event callback
void
on_mouse_up(void *data, Evas *e, Evas_Object *button, void *event_info)
{
        buttondata_s* bd = (buttondata_s*)data;
        evas_object_color_set(bd->rect, 128, 128, 255, 255);
}
```

此代码会在用户对自定义 Button 执行 Touch Down 操作时将背景色更改为粉红色，会在用户对自定义 Button 执行 Touch Cancel 操作时将背景色更为天蓝色。

再次运行该示例。先点击、然后取消点击 Button。背景色会随之变化。



5) 在主屏幕上请求单击事件

要在主屏幕上请求 Button 单击事件，您需要允许在发生 Touch Cancel 事件时调用回调函数。在 create_button() 函数的结尾添加新代码。

```
        evas_object_event_callback_add( bd->text, EVAS_CALLBACK_MOUSE_DOWN, on_mouse_down, (void*)bd);
        evas_object_event_callback_add( bd->text, EVAS_CALLBACK_MOUSE_UP, on_mouse_up, (void*)bd);
        evas_object_event_callback_add( bd->text, EVAS_CALLBACK_MOUSE_UP, func, (void*)bd);

        return bd;
```

如果用户先点击然后取消点击 Button，主屏幕上会显示所传递的函数。

转至主源文件 (custombuttonex.c) 并修改 create_base_gui() 函数的代码。

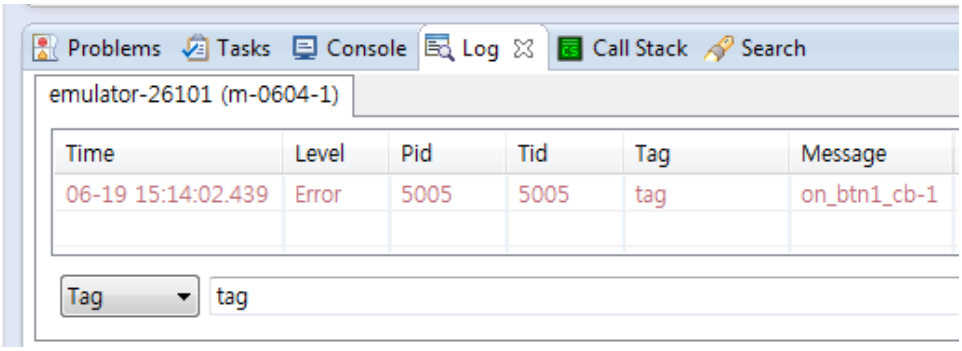
```
    {
        /* Canvas */
        Evas* canvas = evas_object_evas_get(ad->conform);

        /* Custom Button-1 */
        //m_bd1 = create_button(canvas, 50, 200, 300, 100, "Button-1", NULL);
        m_bd1 = create_button(canvas, grid, 50, 200, 300, 100, "Button-1", on_btn1_cb);
    }
```

我们已将自定义 Button 的回调函数名称指定为 on_btn1_cb。现在，我们需要添加回调函数。在 create_base_gui() 函数之上创建一个新函数。

```
static void
on_btn1_cb(void *data, Evas *e, Evas_Object *button, void *event_info)
{
        dlog_print(DLOG_ERROR, "tag", "on_btn1_cb-1");
}
```

此代码会在用户点击自定义 Button 时显示一则 Log 消息。

再次运行该示例。点击 Button，随后 Log 窗格中会显示“on_btn1_cb-1”消息。



6）更改标题文本

在本小节中，我们将实施一项功能，以便在点击新添加的第二个自定义 Button 时更改第一个 Button 的标题文本。要执行此操作，我们需要向自定义小部件文件 (CustomButton.h) 添加一个文本更改函数。函数的放置位置并不重要。但最好是尽量将从外部调用的函数置于最底层。这是因为可能会从此函数调用其它函数。

```
void set_button_text(buttondata_s* bd, const char* str)
{
        evas_object_textblock_text_markup_set(bd->text, str);
}

#endif /* CUSTOMBUTTON_H_ */
```

当在主屏幕上调用此函数时，相关 Button 的标题文本将会改变。转至主源文件 (custombuttonex.c)，并在 create_base_gui() 函数的结尾添加新代码。

```
    {
        /* Canvas */
```

```
        Evas* canvas = evas_object_evas_get(ad->conform);

        /* Custom Button-1 */
        m_bd1 = create_button(canvas, grid, 50, 200, 300, 100, "Button-1", on_bt
n1_cb);

        /* Custom Button-2 */
        create_button(canvas, grid, 50, 400, 300, 100, "Button-2", on_btn2_cb);
    }
```

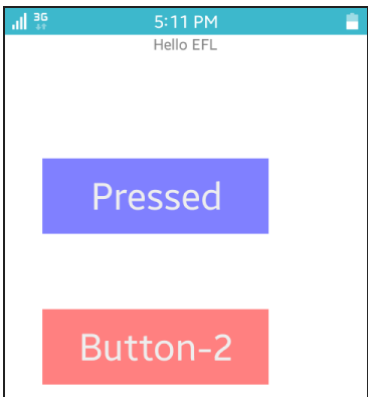
此代码可创建第二个自定义 Button、将标题文本指定为“Button-2”，并将回调函数指定为 on_btn2_cb。

最后，我们需要为第二个 Button 添加回调函数。在 create_base_gui() 函数之上创建一个新函数。

```
static void
on_btn2_cb(void *data, Evas *e, Evas_Object *button, void *event_info)
{
        set_button_text(m_bd1, "Pressed");
}
```

此代码可在用户点击第二个 Button 时将第一个 Button 的标题文本更改为“Pressed”。

再次运行该示例。点击第二个 Button，第一个 Button 的标题文本会改变。



7）相关 API

Evas_Object *evas_object_textblock_add(Evas *e)：一种用于创建 TextBlock 对象的 API。

Evas_Textblock_Style：一种存储 TextBlock 属性信息的样式结构。

Evas_Textblock_Style *evas_textblock_style_new()：一种用于创建 Evas_Textblock_Style 对象的 API。

void evas_textblock_style_set(Evas_Textblock_Style *ts, const char *text)：一种用于定义 Evas_Textblock_Style 样式的 API。/ 参数：Evas_Textblock_Style 对象和 HTML 标记。

void evas_object_textblock_style_set(Evas_Object *obj, Evas_Textblock_Style *ts)：一种用于指定 TextBlock 对象样式的 API。/ 参数：TextBlock 对象和 Evas_Textblock_Style 对象。

void evas_textblock_style_free(Evas_Textblock_Style *ts)：一种用于删除 Evas_Textblock_Style 对象的 API。

void evas_object_textblock_text_markup_set(Evas_Object *obj, const char *text)一种用于为 TextBlock 对象指定字符串的 API。

void evas_object_textblock_valign_set(Evas_Object *obj, double align)：一种用于指定垂直对齐 TextBlock 标题文本的 API。将 0 传递给第二个参数，会将文本靠上对齐，传递 1 则会将文本靠下对齐，传递 0.5 则会将文本居中对齐。

37. 使用 Animator

Animator 可用于按指定时间间隔更改屏幕上的对象。Animator 与 Timer 类似，但二者的区别在于 Animator 能生成多种画面效果。让我们通过示例来详细了解如何使用 GenList 小部件。

1) 创建动画

创建新的源项目，并将项目名称指定为“AnimatorEx”。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在源文件的顶端添加变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *rect1;
        Evas_Object *rect2;
} appdata_s;

Eina_Bool anim_continue = ECORE_CALLBACK_RENEW;
Ecore_Pos_Map pos_map = ECORE_POS_MAP_LINEAR;
```

rect1 和 rect2 是即将应用动画的方块对象。

anim_continue 是用于确定是否继续运行动画的布尔变量。

Ecore_Pos_Map 是用于指定动画样式的选项。我们将在 TimeLine 动画中使用此设置。

现在我们将创建第一个方块对象和一个 Animator 对象。向 create_base_gui() 函数添加新代码。为 Conformant 创建代码和 Label 创建代码加上注解。

```
    /* Conformant */
    /*ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
```

```
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);*/

    /* Label*/
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "Hello EFL");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
    elm_object_content_set(ad->conform, ad->label);
    evas_object_show(ad->label);*/

    /* Evas */
    Evas *evas = evas_object_evas_get(ad->win);

    /* Rect-1 */
    ad->rect1 = evas_object_rectangle_add(evas);
    evas_object_pass_events_set(ad->rect1, EINA_TRUE);
    evas_object_color_set(ad->rect1, 0, 0, 160, 160);
    evas_object_resize(ad->rect1, 50, 50);
    evas_object_show(ad->rect1);

    /* Animation-1 */
    Ecore_Animator *anim = ecore_animator_add(on_next_frame1, ad->rect1);
    ecore_animator_frametime_set(1. / 60);

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

创建 Evas 对象，以创建 Rectangle 对象。

ecore_animator_add(Ecore_Task_Cb, void *) 是一种用于创建 Animator 对象的 API。第一个参数指示帧事件回调函数，第二个参数则指示用户数据。它通常传递的是要为其应用动画的对象或 appdata。

ecore_animator_frametime_set(double) 是一种用于指定动画帧时间间隔的 API。例如，如果将间隔指定为 1/60，则每秒钟将发生 60 个帧事件。单位为秒。

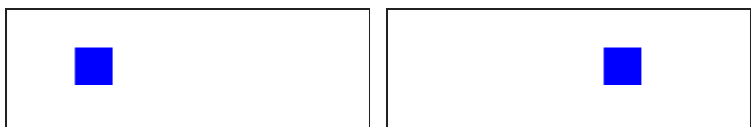
现在，我们将创建帧事件函数。在 create_base_gui() 函数之上创建一个新函数。

```
static Eina_Bool on_next_frame1(void *data)
{
        static int x = 0;
        if (x >= 350)
                x = 0;
        evas_object_move(data, x += 2, 50);
        return anim_continue;
}
```

evas_object_move() 是一种用于更改对象位置的 API。此函数以 2 为增量逐帧增加对象的 x 坐标。当 x 坐标超过 350 时，将重新从 0 开始。

如果从帧事件函数返回的值为 ECORE_CALLBACK_RENEW，则会继续运行动画。如果返回的值为 ECORE_CALLBACK_CANCEL，则会暂停运行动画。

构建并运行该示例。蓝色方块从左侧移至右侧。蓝色方块在移动一定距离之后，将从屏幕左侧重新开始移动。



2) 停止动画

要停止动画，帧事件函数需要返回 ECORE_CALLBACK_CANCEL。我们将实施一项功能以在点击 Button 时停止动画。在 create_base_gui() 函数之上创建一个新函数。该函数将向一个 Table 容器添加一个小部件。

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int x, int y, int w, int
h)
{
   evas_object_size_hint_align_set(child, EVAS_HINT_FILL, 0.5);
   evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
   elm_table_pack(table, child, x, y, w, h);
   evas_object_show(child);
}
```

然后，将新代码添加到 create_base_gui() 函数中。此代码将创建一个 Box 和一个 Table，并添加一个 Button。

```
    /* Rect-1 */
    ad->rect1 = evas_object_rectangle_add(evas);
    evas_object_pass_events_set(ad->rect1, EINA_TRUE);
    evas_object_color_set(ad->rect1, 0, 0, 160, 160);
    evas_object_resize(ad->rect1, 50, 50);
    evas_object_show(ad->rect1);

    {
        /* Box to put the table in so we can bottom-align the table
         * window will stretch all resize object content to win size */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_win_resize_object_add(ad->win, box);
        evas_object_show(box);

        /* Table */
        Evas_Object *table = elm_table_add(ad->win);
        /* Make table homogenous - every cell will be the same size */
        elm_table_homogeneous_set(table, EINA_TRUE);
        /* Set padding of 10 pixels multiplied by scale factor of UI */
        elm_table_padding_set(table, 20 * elm_config_scale_get(), 10 * elm_conf
ig_scale_get());
        /* Let the table child allocation area expand within in the box */
        evas_object_size_hint_weight_set(table, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
        /* Set table to fiill width but align to bottom of box */
        evas_object_size_hint_align_set(table, EVAS_HINT_FILL, 1.0);
        elm_box_pack_end(box, table);
        evas_object_show(table);

        {
            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "■");
            evas_object_smart_callback_add(btn, "clicked", btn_stop_cb, NULL);
            my_table_pack(table, btn, 0, 0, 2, 1);
        }
    }
```

```
    evas_object_raise(ad->rect1);

    /* Animation-1 */
    Ecore_Animator *anim = ecore_animator_add(on_next_frame1, ad->rect1);
```

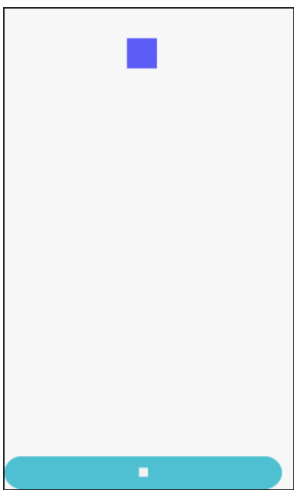
evas_object_raise() 是一种可将对象移至屏幕最顶端的 API，这样一来，该对象就不会被其它对象隐藏。

这样做才能在点击 Button 时，将 ECORE_CALLBACK_CANCEL 存储到 anim_continue 中。现在我们将在 create_base_gui() 函数之上为 Button 添加回调函数。

```
static void
btn_stop_cb(void *data, Evas_Object *obj, void *event_info)
{
        anim_continue = ECORE_CALLBACK_CANCEL;
}
```

这样做才能在点击 Button 后，使 on_next_frame1() 返回 ECORE_CALLBACK_CANCEL 的结果，从而停止动画。

再次运行该示例。点击 Button，方块停止移动。



3) 使用 Timer 小部件暂停/恢复动画

要暂停动画，您需要使用 ecore_animator_freeze() 函数。要恢复动画，您需要使用 ecore_animator_thaw() 函数。向 create_base_gui() 函数添加新代码。此代码将创建两个 Timer。

```
    /* Animation-1 */
    Ecore_Animator *anim = ecore_animator_add(on_next_frame1, ad->rect1);
    /* add 2 timers to go off every 6 seconds */
    ecore_timer_add(6, freeze_anim, anim);
    Ecore_Timer *timer = ecore_timer_add(6, thaw_anim, anim);
    /* delay the last timer by 3 seconds so the 2 timers are offset */
    ecore_timer_delay(timer, 3);
```

现已创建两个 Timer。第一个 Timer 用于暂停动画，第二个 Timer 用于恢复动画。

ecore_timer_delay() 是一种在经过一定时间后启动 Timer 的 API。对于第二个 Timer，将在 3 + 6 = 9 秒之后发生此事件，此后每 6 秒发生一次此事件。

我们现在将创建一个计时器事件函数。在 create_base_gui() 函数之上添加两个新函数。

```
static Eina_Bool freeze_anim(void *data)
{
        ecore_animator_freeze(data);
        // Animation stop Timer delete
        return ECORE_CALLBACK_CANCEL;
}

static Eina_Bool thaw_anim(void *data)
{
        ecore_animator_thaw(data);
        // Animation restart Timer delete
        return ECORE_CALLBACK_CANCEL;
}
```

freeze_anim() 函数将在应用程序启动之后 3 秒被调用。

ecore_animator_freeze(Ecore_Animator *) 是一种用于暂停动画的 API。

thaw_anim() 函数将在应用程序启动之后 6 秒被调用。

ecore_animator_thaw(Ecore_Animator *) 是一种用于恢复动画的 API。

再次运行此示例，您现在将会看到每 3 秒移动的动画会暂停和恢复一次。

4) TimeLine 动画

在本小节，我们将实施一个会播放一定时间量的动画。向 create_base_gui() 函数添加新代码。此代码将创建第二个 Rectangle 对象和一个 TimeLine 动画。

```
/* Rect-1 */
ad->rect1 = evas_object_rectangle_add(evas);
evas_object_pass_events_set(ad->rect1, EINA_TRUE);
evas_object_color_set(ad->rect1, 0, 0, 160, 160);
evas_object_resize(ad->rect1, 50, 50);
evas_object_show(ad->rect1);

/* Rect-2 */
ad->rect2 = evas_object_rectangle_add(evas);
evas_object_pass_events_set(ad->rect2, EINA_TRUE);
evas_object_color_set(ad->rect2, 0, 55, 0, 160);
evas_object_resize(ad->rect2, 50, 50);
evas_object_show(ad->rect2);

{
~
}

evas_object_raise(ad->rect1);
evas_object_raise(ad->rect2);

/* Animation-1 */
Ecore_Animator *anim = ecore_animator_add(on_next_frame1, ad->rect1);
/* add 2 timers to go off every 6 seconds */
```

```
    ecore_timer_add(6, freeze_anim, anim);
    Ecore_Timer *timer = ecore_timer_add(6, thaw_anim, anim);
    /* delay the last timer by 3 seconds so the 2 timers are offset */
    ecore_timer_delay(timer, 3);

    /* Animation-2 */
    ecore_animator_timeline_add(4, on_next_frame2, ad->rect2);

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

ecore_animator_timeline_add（double、Ecore_Timeline_Cb、void *）是一种用于创建 TimeLine 动画的 API。第一个参数指示播放时间，单位为秒。第二个参数是帧事件的函数名称，第三个为用户数据。

现在，我们将对第二个方块应用会改变方块位置、大小和颜色的动画。在 create_base_gui() 函数之上添加新代码。这是 TimeLine 动画帧事件函数。

```
static Eina_Bool on_next_frame2(void *data, double pos)
{
        double frame = ecore_animator_pos_map(pos, pos_map, 1.2, 15);
        evas_object_resize(data, 50 * (1 + frame * 2), 50 * (1 + frame * 2));
        evas_object_move(data, 200 * frame, 200 * frame + 100);
        evas_object_color_set(data, 255 * frame, 0, 255 * (1 - frame), 255);
        return ECORE_CALLBACK_RENEW;
}
```

ecore_animator_pos_map() 是一种会返回映射到当前动画位置之结果值的 API。返回值的范围在 0 至 1 之间。动画启动时的结果值为 0，然后此值会逐渐增加，直到在动画停止时达到 1。Timeline 动画事件函数的第二个参数必须传递给第一个参数。为第二个参数输入动画类型。样式类型如下：

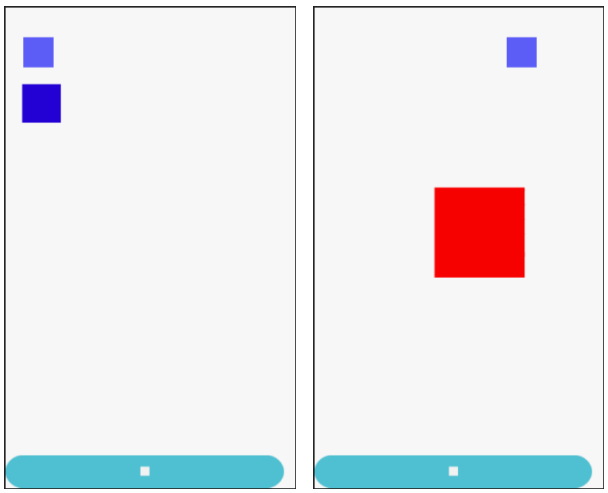
- ECORE_POS_MAP_LINEAR, /**< Linear 0.0 -> 1.0 */
- ECORE_POS_MAP_ACCELERATE, /**< Start slow then speed up */
- ECORE_POS_MAP_DECELERATE, /**< Start fast then slow down */
- ECORE_POS_MAP_SINUSOIDAL, /**< Start slow, speed up then slow down at the end */
- ECORE_POS_MAP_ACCELERATE_FACTOR, /**< Start slow then speed up, v1 being a power factor, @c 0.0 being linear, @c 1.0 being normal accelerate, @c 2.0 being much more pronounced accelerate (squared),

```
@c 3.0 being cubed, and so on */
-  ECORE_POS_MAP_DECELERATE_FACTOR, /**< Start fast then slow down, v1 being a power factor, @c 0.0 being linear, @c 1.0 being normal decelerate, @c 2.0 being much more pronounced decelerate (squared), @c 3.0 being cubed, and so on */
-  ECORE_POS_MAP_SINUSOIDAL_FACTOR, /**< Start slow, speed up then slow down at the end, v1 being a power factor, @c 0.0 being linear, @c 1.0 being normal sinusoidal, @c 2.0 being much more pronounced sinusoidal (squared), @c 3.0 being cubed, and so on */
-  ECORE_POS_MAP_DIVISOR_INTERP, /**< Start at gradient * v1, interpolated via power of v2 curve */
-  ECORE_POS_MAP_BOUNCE, /**< Start at @c 0.0 then "drop" like a ball bouncing to the ground at @c 1.0, and bounce v2 times, with decay factor of v1 */
-  ECORE_POS_MAP_SPRING /**< Start at @c 0.0 then "wobble" like a spring with rest position @c 1.0, and wobble v2 times, with decay factor of v1 */
```

为 ecore_animator_pos_map() 函数的第三个参数输入速度变化强度。为第四个参数输入速度变化节奏。

以下代码指定方块的大小、位置和颜色。

再次运行该示例，您现在将会看到当蓝色方块移至右下方时会变为红色并变大。



5) 为 TimeLine 动画应用加速

对我们刚创建的动画应用的样式是 ECORE_POS_MAP_LINEAR。这是一个保持动画按一定速度移动的选项。现在，我们将应用一个使动画首先慢速移动、然后逐渐增速的选项。向 create_base_gui() 函数添加新代码。此代码将创建一个新的 Button。

```
    {
        /* Button-1 */
        Evas_Object *btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "■");
        evas_object_smart_callback_add(btn, "clicked", btn_stop_cb, NULL);
        my_table_pack(table, btn, 0, 0, 2, 1);

        /* Button-2 */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Accelerate");
        evas_object_smart_callback_add(btn, "clicked", btn_accelerate_cb, ad->r
ect2);
        my_table_pack(table, btn, 0, 1, 1, 1);
    }
}
```

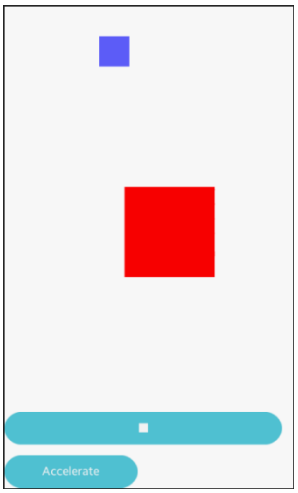
接下来，我们将为第二个 Button 创建回调函数。在 create_base_gui() 函数之上创建一个新函数。

```
static void
btn_accelerate_cb(void *data, Evas_Object *obj, void *event_info)
{
        pos_map = ECORE_POS_MAP_ACCELERATE;
        ecore_animator_timeline_add(4, on_next_frame2, data);
}
```

我们已经将动画样式更改为 ECORE_POS_MAP_ACCELERATE。这是一个加速选项。

我们已使用 ecore_animator_timeline_add() 函数创建了 Timeline 动画。

再次运行该示例，然后点击 Accelerate 按钮。这将启动动画。



6）其他动画样式

我们将为 Timeline 动画应用不同的样式。为 create_base_gui() 函数添加四个 Button。

```
        /* Button-2 */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Accelerate");
        evas_object_smart_callback_add(btn, "clicked", btn_accelerate_cb, ad->re
ct2);
        my_table_pack(table, btn, 0, 1, 1, 1);

        /* Button-3 */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Decelerate");
        evas_object_smart_callback_add(btn, "clicked", btn_decelerate_cb, ad->r
ect2);
        my_table_pack(table, btn, 1, 1, 1, 1);

        /* Button-4 */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Sinusoidal");
        evas_object_smart_callback_add(btn, "clicked", btn_sinusoidal_cb, ad->r
ect2);
        my_table_pack(table, btn, 0, 2, 1, 1);
```

```
        /* Button-5 */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Bounce");
        evas_object_smart_callback_add(btn, "clicked", btn_bounce_cb, ad->rect
2);
        my_table_pack(table, btn, 1, 2, 1, 1);

        /* Button-6 */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Spring");
        evas_object_smart_callback_add(btn, "clicked", btn_spring_cb, ad->rect
2);
        my_table_pack(table, btn, 0, 3, 1, 1);
    }
}
```

您还需要四个匹配的 Button 回调函数。在 create_base_gui() 函数之上添加四个新函数。

```
static void
btn_decelerate_cb(void *data, Evas_Object *obj, void *event_info)
{
        pos_map = ECORE_POS_MAP_DECELERATE;
        ecore_animator_timeline_add(4, on_next_frame2, data);
}

static void
btn_sinusoidal_cb(void *data, Evas_Object *obj, void *event_info)
{
        pos_map = ECORE_POS_MAP_SINUSOIDAL;
        ecore_animator_timeline_add(4, on_next_frame2, data);
}

static void
btn_bounce_cb(void *data, Evas_Object *obj, void *event_info)
{
        pos_map = ECORE_POS_MAP_BOUNCE;
        ecore_animator_timeline_add(4, on_next_frame2, data);
}

static void
btn_spring_cb(void *data, Evas_Object *obj, void *event_info)
{
```

```
        pos_map = ECORE_POS_MAP_SPRING;
        ecore_animator_timeline_add(4, on_next_frame2, data);
}
```

ECORE_POS_MAP_ACCELERATE 为 ECORE_POS_MAP_ACCELERATE 的相反属性。它是首先快速，然后逐渐减速。

ECORE_POS_MAP_SINUSOIDAL 先慢速启动，然后变快，最后再变慢。

ECORE_POS_MAP_BOUNCE 像弹力球一样震动。但是，它不会超过端点。

ECORE_POS_MAP_SPRING 像弹力球一样震动。它在弹离端点后缓慢停止。

再次运行该示例，并点击所添加的 Button，一次点击一个。

7）连续动画

我们将实施一项功能，以便在某种动画结束后自动启动第二种动画。您可以为其使用 Timer。在 create_base_gui() 函数上创建一个新 Button。

```
        /* Button-6 */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Spring");
```

```
            evas_object_smart_callback_add(btn, "clicked", btn_spring_cb, ad->re
ct2);
            my_table_pack(table, btn, 0, 3, 1, 1);

            /* Button-7 */
            btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "Twice");
            evas_object_smart_callback_add(btn, "clicked", btn_twice_cb, ad->re
ct2);
            my_table_pack(table, btn, 1, 3, 1, 1);
        }
    }
```

然后，在 create_base_gui() 函数之上添加两个新函数。

```
static Eina_Bool
start_second_anim(void *data)
{
        pos_map = ECORE_POS_MAP_SPRING;
        ecore_animator_timeline_add(4, on_next_frame2, data);
        return ECORE_CALLBACK_CANCEL;
}

static void
btn_twice_cb(void *data, Evas_Object *obj, void *event_info)
{
        pos_map = ECORE_POS_MAP_ACCELERATE;
        ecore_animator_timeline_add(4, on_next_frame2, data);
        ecore_timer_add(4, start_second_anim, data);
}
```

start_second_anim() 是一种用于启动第二个动画的函数。

btn_twice_cb() 是第 7 个 Button 的回调函数。启动加速动画，并使用 Timer 使第二个动画在 4 秒后自动开始。

再次运行该示例，并点击“Twice”按钮。加速动画开始并结束后，开始运行 Spring 动画。

8）相关 API

void ecore_animator_frametime_set(double frametime)：用于指定动画之间的时间帧。/ 参数 - 帧的时间间隔（单位：秒）。

Ecore_Animator *ecore_animator_add(Ecore_Task_Cb func, void *data)：创建 Animator 对象。/ 参数：帧事件的回调函数、用户数据。它通常传递的是要为其应用动画的对象或 appdata。

void ecore_animator_freeze(Ecore_Animator *animator)：暂停动画。

void ecore_animator_thaw(Ecore_Animator *animator)：重新启动动画。

Ecore_Animator *ecore_animator_timeline_add(double runtime, Ecore_Timeline_Cb func, void *data)：创建 Timeline 动画。/第一个参数为运行时间（单位：秒）。第二个参数是帧事件的函数名称，第三个为用户数据。

EAPI double ecore_animator_pos_map(double pos, Ecore_Pos_Map map, double v1, double v2)：返回映射至当前动画位置的结果值。返回值在 0 至 1 之间。动画启动时的结果值为 0，然后此值会逐渐增加，直到在动画停止时达到 1。只需将 Timeline 动画事件函数的第二个参数传递给第一个参数。为第二个参数指定 Ecore_Pos_Map。为第三个参数指定强度；为第四个参数指定速度变化速率。

Ecore_Pos_Map 的样式如下：
  - ECORE_POS_MAP_LINEAR, /**< Linear 0.0 -> 1.0 */
  - ECORE_POS_MAP_ACCELERATE, /**< Start slow then speed up */
  - ECORE_POS_MAP_DECELERATE, /**< Start fast then slow down */
  - ECORE_POS_MAP_SINUSOIDAL, /**< Start slow, speed up then slow down at the end */
  - ECORE_POS_MAP_ACCELERATE_FACTOR, /**< Start slow then speed up, v1 being a power factor, @c 0.0 being linear, @c 1.0 being normal accelerate, @c 2.0 being much more pronounced accelerate (squared), @c 3.0 being cubed, and so on */
  - ECORE_POS_MAP_DECELERATE_FACTOR, /**< Start fast then slow down, v1 being a power factor, @c 0.0 being linear, @c 1.0 being normal decelerate, @c 2.0 being much more pronounced decelerate (squared), @c 3.0 being cubed, and so on */
  - ECORE_POS_MAP_SINUSOIDAL_FACTOR, /**< Start slow, speed up then slow down at the end, v1 being a power factor, @c 0.0 being linear, @c 1.0 being normal sinusoidal, @c 2.0 being much more pronounced sinusoidal (squared), @c 3.0 being cubed, and so on */
  - ECORE_POS_MAP_DIVISOR_INTERP, /**< Start at gradient * v1, interpolated via power of v2 curve */
  - ECORE_POS_MAP_BOUNCE, /**< Start at @c 0.0 then "drop" like a ball bouncing to the ground at @c 1.0, and bounce v2 times, with decay factor of v1 */
  - ECORE_POS_MAP_SPRING /**< Start at @c 0.0 then "wobble" like a spring with rest position @c 1.0, and wobble v2 times, with decay factor of v1 */

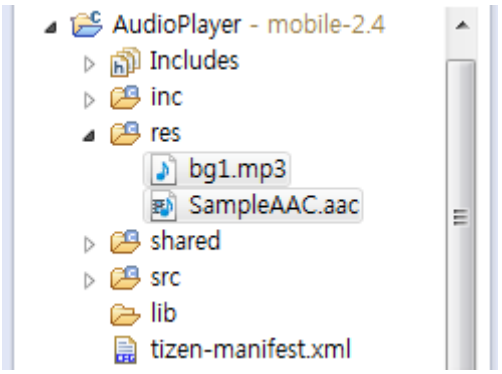
38. 播放音频

当 Sony 公司在 1979 年推出一款便携式音频设备时，市场调查的反馈结果是负面的。然而这并不能阻止 Sony 总裁 Orita 的前进步伐，正是由于他的坚持，随身听才得以问世。随身听很快成为一款畅销产品。在 21 世纪，Steve Jobs 发明了一款小型 MP3 播放器，并很快风靡全世界。然而随着智能手机的不断涌现，这些便携式音乐播放器逐渐湮没在历史长河中。

您可以使用 Player 播放音频和视频。在本小节中，我们将介绍如何播放音频文件。

1) 加载音频文件

创建一个新的源项目，并将项目名称指定为“AudioPlayer”。复制音频文件。将附录 /Audio 文件夹中的 2 个图像文件（bg1.mp3 和 SampleAAC.aac）复制到源项目的 /res 文件夹中。



现在，请输入源代码。打开 src 文件夹中的源文件 (~.c)，并在屏幕顶端添加库和变量。

```
#include "audioplayer.h"
#include <player.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
```

```
        Evas_Object *label;
        player_h player;
} appdata_s;

const char* file_name[] = { "SampleAAC.aac", "bg1.mp3" };
```

player_h 是一种用于播放音频、视频等媒体的 Player 结构。

file_name[] 是一种用于保存文件名称的字符串数组。

当运行此应用程序时，将会自动创建一个 Player 对象并加载第一个音频文件。在 create_base_gui() 函数的结尾添加新代码。

```
        evas_object_show(ad->win);

        // Create Player
        ad->player = create_player();

        // Load audio file to Player
        prepare_player(ad, 0);
```

create_player() 函数创建 Player，而 prepare_player() 加载音频文件。现在，让我们开始逐步进行编译。在 create_base_gui() 函数之上添加六个函数。

```
// Get player state
static player_state_e
get_player_state(player_h player)
{
        player_state_e state;
        player_get_state(player, &state);
        return state;
}

// Play completed event function
static void
on_player_completed(player_h* player)
{
        if(player)
                player_stop(player);
```

```
}

// Create Player
static player_h
create_player()
{
        player_h player;

        player_create(&player);
        player_set_completed_cb(player, on_player_completed, player);

        return player;
}

// Stop play
static void
stop_player(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;

        if( get_player_state(ad->player) == PLAYER_STATE_PLAYING || get_player_s
tate(ad->player) == PLAYER_STATE_PAUSED)
        {
                player_stop(ad->player);
        }
}

// Get full path of source file
static inline const char*
get_resource_path(const char *file_path)
{
        static char absolute_path[PATH_MAX] = {'\0'};

        static char *res_path_buff = NULL;
        if(res_path_buff == NULL)
                res_path_buff = app_get_resource_path();

        snprintf(absolute_path, PATH_MAX, "%s%s", res_path_buff, file_path);
        return absolute_path;
}

// Load file to Player
static void
prepare_player(appdata_s* ad, int index)
{
        // Stop play
```

```
        stop_player(ad, NULL, NULL);
        // Close file
        player_unprepare(ad->player);

        const char* file = file_name[index];
        // Get full path of source file
        const char *res_path = get_resource_path(file);

        // Load file
        player_set_uri(ad->player, res_path);
        // Prepare play
        int result = player_prepare(ad->player);
        dlog_print(DLOG_INFO, "tag", "File load : %d", result);
}
```

get_player_state() 函数返回 Player 的当前状态。

player_get_state(player_h, player_state_e) 是一种返回 Player 当前状态的 API。第一个参数是 Player 对象，第二个参数返回状态值。player_state_e 状态类型如下：

- PLAYER_STATE_NONE, /**< Player is not created */
- PLAYER_STATE_IDLE, /**< Player is created, but not prepared */
- PLAYER_STATE_READY, /**< Player is ready to play media */
- PLAYER_STATE_PLAYING, /**< Player is playing media */
- PLAYER_STATE_PAUSED, /**< Player is paused while playing media */

on_player_completed() 是一种在完成播放时调用的回调函数。

player_stop(player_h) 是一种用于停止播放的 API。

create_player() 函数创建 Player 对象。

player_create(player_h *) 是一种创建 Player 对象的 API。

player_set_completed_cb(player_h, player_completed_cb, void *) 是一种指定在完成播放时执行的回调函数的 API。第二个参数为回调函数的名称，而第三个参数为用户数据。

stop_player() 函数停止播放。

get_resource_path() 函数返回 /res 文件夹下所保存的所有文件的完整路径。

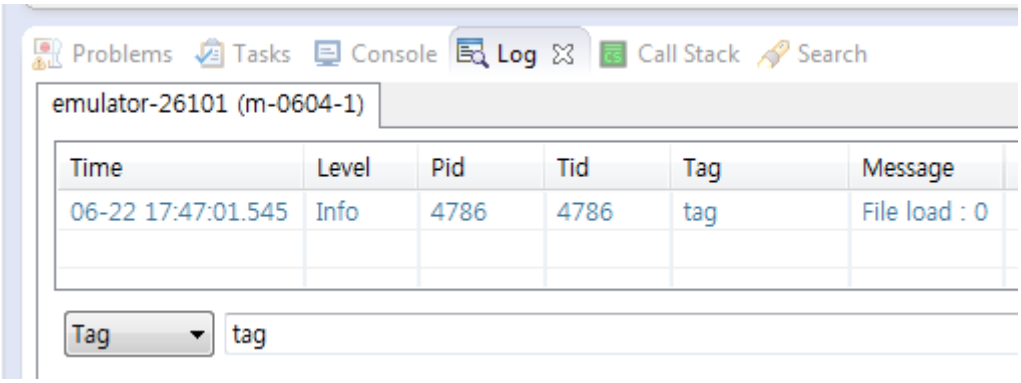
prepare_player() 函数加载音频文件。

player_unprepare(player_h) 是一种用于关闭您加载到 Player 的文件的 API。

player_set_uri(player_h, const char *) 是一种用于将文件加载到 Player 的 API。

player_prepare(player_h) 是一种用于使 Player 做好播放准备的 API。如已准备好播放，则返回 0。

如果您以此方式编译并开始运行，则您不会在屏幕上看到或听到任何内容。如果在 Log 窗口查看此消息，则会显示“File load:0”。



2）播放音频

我们将实施一项功能，以便在单击此 Button 时播放音频文件。在 create_base_gui() 函数之上创建一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
                      double h_weight, double v_weight, double h_align, double
 v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

向 create_base_gui() 函数添加新代码。此代码将创建一个 Box 容器，并添加一个 Button 小部件。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
```

```
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        Evas_Object * box, *btn;

        /* A box to put things in verticallly - default mode for box */
        box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        { /* child object - indent to how relationship */
            /* Label*/
            ad->label = elm_label_add(ad->win);
            elm_object_text_set(ad->label, "<align=center>Hello Tizen</>");
            /* expand horizontally but not vertically, and fill horiz,
             * align center vertically */
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.0);

            /* Play Button */
            btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "Play");
            evas_object_smart_callback_add(btn, "clicked", start_player, ad);
            my_box_pack(box, btn, 1.0, 0.0, -1.0, 0.0);
         }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

我们将为此 Button 创建一个回调函数。在 create_base_gui() 函数之上创建一个新函数。

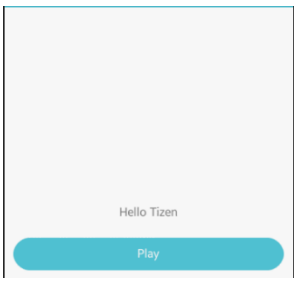
```
// Start play
static void start_player(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
```

```
        if( get_player_state(ad->player) != PLAYER_STATE_PLAYING)
                player_start(ad->player);
}
```

start_player() 函数用于开始播放音频。

player_start(player_h) 是一种用于开始播放音频的 API。

再次运行该示例，并点击 Play 按钮。这将播放音乐。



3) 暂停和停止

我们将添加 2 个 Button 以执行 Pause 和 Stop 功能。向 create_base_gui() 函数添加新代码。

```
            /* Play Button */
            btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "Play");
            evas_object_smart_callback_add(btn, "clicked", start_player, ad);
            my_box_pack(box, btn, 1.0, 0.0, -1.0, 0.0);

            /* Pause Button */
            btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "Pause");
            evas_object_smart_callback_add(btn, "clicked", pause_player, ad);
            my_box_pack(box, btn, 1.0, 0.0, -1.0, -1.0);

            /* Stop Button */
            btn = elm_button_add(ad->win);
```

```
            elm_object_text_set(btn, "Stop");
            evas_object_smart_callback_add(btn, "clicked", stop_player, ad);
            my_box_pack(box, btn, 1.0, 0.0, -1.0, -1.0);
        }
    }
```

我们已添加 2 个 Button。现在，我们将为这两个 Button 创建回调函数。在 create_base_gui() 函数之上添加新代码。

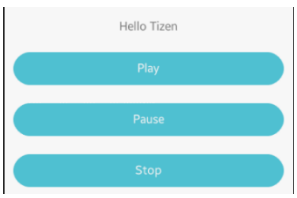
```
// Pause play
static void pause_player(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;

        if( get_player_state(ad->player) == PLAYER_STATE_PLAYING )
                player_pause(ad->player);
}
```

start_player() 函数暂停播放音频。

player_start(player_h) 是一种用于暂停播放音频的 API。

您不必为 Stop Button 另创建一个回调函数。这是因为您已具有 stop_player() 函数。

再次运行该示例，并点击 Play 按钮以播放音乐。单击 Pause 按钮可暂停，或单击 Play Again 按钮以重新开始。单击 Stop 按钮可停止，或单击 Play Again 按钮以从头开始播放。



4) 更改音频文件

我们将实施一项功能，以便通过添加 2 个 Button 来更改音频文件。向 create_base_gui() 函数添加新代码。

```
        /* Label*/
        ad->label = elm_label_add(ad->win);
        elm_object_text_set(ad->label, "<align=center>Hello Tizen</>");
        /* expand horizontally but not vertically, and fill horiz,
         * align center vertically */
        my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.0);

        /* File Load-1 Button */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "File-1");
        evas_object_smart_callback_add(btn, "clicked", btn_load_file1, ad);
        /* epand both horiz and vert, fill horiz and vert */
        my_box_pack(box, btn, 1.0, 0.0, -1.0, -1.0);

         /* File Load-2 Button */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "File-2");
        evas_object_smart_callback_add(btn, "clicked", btn_load_file2, ad);
        my_box_pack(box, btn, 1.0, 1.0, -1.0, 0.0);

        /* Play Button */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Play");
        evas_object_smart_callback_add(btn, "clicked", start_player, ad);
        my_box_pack(box, btn, 1.0, 0.0, -1.0, 0.0);
```

我们已创建 2 个 Button。最后，我们准备好为这些 Button 创建回调函数。在 create_base_gui() 函数之上添加新代码。

```
static void
btn_load_file1(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        // Load file to Player
        prepare_player(ad, 0);
}
```

```
static void
btn_load_file2(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        // Load file to Player
        prepare_player(ad, 1);
}
```

btn_load_file1() 函数加载第一个音频文件。如果将 0 传递给 prepare_player() 函数的第二个参数，则加载第一个音频文件。

btn_load_file2() 函数加载第一个音频文件。如果将 1 传递给 prepare_player() 函数的第二个参数，则加载第二个音频文件。

再次运行该示例，并点击 File-2 按钮，然后再点击 Play 按钮。您将听到不同的音乐。现在，请点击 File-1 按钮，然后再点击 Play 按钮。您将听到先前听过的音乐。

5) 相关 API

int player_get_state(player_h player, player_state_e *state)：一种返回 Player 当前状态的 API。第一个参数是 Player 对象，第二个参数返回状态值。player_state_e 状态类型如下：

- PLAYER_STATE_NONE, /**< Player is not created */
- PLAYER_STATE_IDLE, /**< Player is created, but not prepared */
- PLAYER_STATE_READY, /**< Player is ready to play media */
- PLAYER_STATE_PLAYING, /**< Player is playing media */
- PLAYER_STATE_PAUSED, /**< Player is paused while playing media */

int player_stop(player_h player)：一种用于停止播放的 API。

int player_create(player_h *player)：一种用于创建 Player 对象的 API。

int player_set_sound_type(player_h player, sound_type_e type)：一种用于指定声音类型的 API。要播放音频文件，只需指定 SOUND_TYPE_MEDIA。

int player_set_volume(player_h player, float left, float right)：一种指定扬声器音量的 API。音量值的范围为 0 至 1.0。/ 参数 - Player 对象、左侧音量、右侧音量。

int player_set_looping(player_h player, bool looping)：一种确定是否循环播放的 API。如果将 True 传递给第二个参数，则循环播放。传递 False，则只播放一次。

int player_set_completed_cb(player_h player, player_completed_cb callback, void *user_data)：一种指定在完成播放时执行的回调函数的 API。/ 参数 - Play 对象、回调函数名称、用户数据。

int player_unprepare(player_h player)：一种用于关闭您加载到 Player 的文件的 API。

int player_set_uri(player_h player, const char * uri)：一种用于将文件加载到 Player 的 API。

int player_prepare(player_h player)：一种用于使 Player 做好播放准备的 API。如已准备好播放，则返回 0。

int player_start(player_h player)：一种用于开始播放音频的 API。

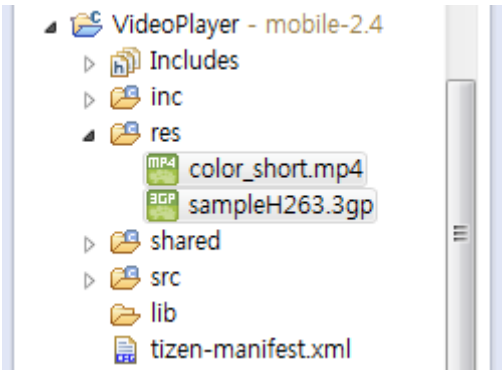
int player_pause(player_h player)：一种用于暂停播放音频的 API。

39. 播放视频

您可以使用 Player 播放音频和视频。两者的区别在于播放视频需要一个屏幕。解决方法就是创建一个与 Player 配合使用的 Image 对象。在本小节中，我们将讨论如何播放视频文件。

1) 创建屏幕

创建一个新的源项目，并将项目名称指定为“VideoPlayer”。复制视频文件。将附录 /Video 文件夹中的 2 个文件（color_short.mp4 和 sampleH263.3gp）复制到源项目的 /res 文件夹中。



与只需要一个 Player 和一个音频文件即可播放音频不同的是，您需要一个屏幕才能播放视频。您可通过以下方式编译屏幕：

- 创建 EVAS
- 创建一个 Image 对象
- 将该 Image 对象指定为 Player 的显示屏

现在，让我们开始实施。打开 src 文件夹中的源文件 (~.c)，并在屏幕顶端添加库和变量。

```
#include "videoplayer.h"
#include <player.h>

typedef struct appdata {
        Evas_Object *win;
```

```
        Evas_Object *conform;
        Evas_Object *label;
        player_h player;
        Evas_Object *video_rect;
} appdata_s;

const char* file_name[] = { "sampleH263.3gp", "color_short.mp4" };
```

player_h 是一种用于播放音频、视频等媒体的 Player 结构。video_rect 是一种在其中显示视频的 Image 对象。

file_name[] 是一种用于保存文件名称的字符串数组。

我们将创建用作屏幕的 Image 对象以及用作屏幕背景的 Bg 小部件。这是因为只有在黑色背景下视频才会达到预期的视觉效果。在 create_base_gui() 函数之上创建一个新函数。此函数将向 Table 添加一个小部件。

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int x, int y, int w, int
h)
{
   evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
   evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
   elm_table_pack(table, child, x, y, w, h);
   evas_object_show(child);
}
```

然后，将新代码添加到 create_base_gui() 函数中。对 Conformant 和 Label 标注注释。

```
    /* Conformant */
    /*ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);*/
```

```
    /* Label*/
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
    elm_object_content_set(ad->conform, ad->label);*/

    Evas_Object *bg, *btn;

    /* Background */
    bg = elm_bg_add(ad->win);
    elm_bg_color_set(bg, 0, 0, 0);
    elm_win_resize_object_add(ad->win, bg);
    evas_object_show(bg);

    {
        /* Box to put the table in so we can bottom-align the table
         * window will stretch all resize object content to win size */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_win_resize_object_add(ad->win, box);
        evas_object_show(box);

        /* Table */
        Evas_Object *table = elm_table_add(ad->win);
        /* Make table homogenous - every cell will be the same size */
        elm_table_homogeneous_set(table, EINA_TRUE);
        /* Set padding of 10 pixels multiplied by scale factor of UI */
        elm_table_padding_set(table, 10 * elm_config_scale_get(), 10 * elm_conf
ig_scale_get());
        /* Let the table child allocation area expand within in the box */
        evas_object_size_hint_weight_set(table, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
        /* Set table to fiill width but align to bottom of box */
        evas_object_size_hint_align_set(table, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_box_pack_end(box, table);
        evas_object_show(table);

        {
            /* Container: standard table */
            Evas_Object *tbl = elm_table_add(ad->win);
            evas_object_size_hint_weight_set(tbl, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
            evas_object_size_hint_align_set(tbl, EVAS_HINT_FILL, EVAS_HINT_FIL
L);
```

```
            elm_object_content_set(ad->conform, tbl);
            evas_object_show(tbl);

            /* The video object */
            ad->video_rect = video_rect_add(tbl);
            my_table_pack(table, ad->video_rect, 0, 0, 3, 3);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

为了将背景颜色更改为黑色，我们创建了一个 Bg 小部件。我们创建了一个 Table 容器以便根据屏幕高宽比放置此小部件，并使用了 Box 来设置小部件之间的宽度。

video_rect_add() 函数创建返回值和 Evas 图像对象。现在，我们开始逐步进行编译。在 create_base_gui() 函数之上添加两个新函数。

```
// Get full path of resource file
static inline const char *get_resource_path(const char *file_path)
{
        static char absolute_path[PATH_MAX] = {'\0'};
        static char *res_path_buff = NULL;
        if(res_path_buff == NULL)
        {
                res_path_buff = app_get_resource_path();
        }

        snprintf(absolute_path, PATH_MAX, "%s%s", res_path_buff, file_path);
        return absolute_path;
}

// Create Image for screen
static Evas_Object *
video_rect_add(Evas_Object *parent)
{
    Evas *evas = evas_object_evas_get(parent);
    Evas_Object *image = evas_object_image_filled_add(evas);
    evas_object_size_hint_weight_set(image, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    evas_object_size_hint_align_set(image, EVAS_HINT_FILL, EVAS_HINT_FILL);
    evas_object_show(image);
```

```
    return image;
}
```

get_resource_path() 函数返回 /res 文件夹下所保存的所有文件的完整路径。我们将在加载视频文件时用到此路径。

video_rect_add() 函数在屏幕区域中创建 Evas 和 Image 对象。

构建并运行该示例。整个屏幕显示为黑色。

2) 加载视频文件

当运行此应用程序时，将会自动创建一个 Player 对象并加载第一个视频文件。在 create_base_gui() 函数的结尾添加新代码。

```
    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    // Create Player
```

```
    ad->player = create_player();

    // Load audio file to Player
    prepare_player(ad, 0);
}
```

create_player() 函数创建 Player，而 prepare_player() 加载视频文件。现在，让我们开始逐步进行编译。在 create_base_gui() 函数之上添加五个函数。从此处开始，每一项操作与 AudioPlayer 示例非常相似。

```
// Get player state
static player_state_e
get_player_state(player_h player)
{
        player_state_e state;
        player_get_state(player, &state);
        return state;
}

// Play completed event function
static void
on_player_completed(player_h* player)
{
        if(player)
                player_stop(player);
}

// Create player
static player_h
create_player()
{
        player_h player;

        player_create(&player);
        player_set_sound_type(player, SOUND_TYPE_MEDIA);
        player_set_volume(player, 1.0, 1.0);
        player_set_looping(player, false);
        player_set_completed_cb(player, on_player_completed, player);

        return player;
}

// Stop play
```

```
static void
stop_player(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;

        if( get_player_state(ad->player) == PLAYER_STATE_PLAYING || get_player_s
tate(ad->player) == PLAYER_STATE_PAUSED)
                player_stop(ad->player);
}

// Load video file to Player
static void
prepare_player(appdata_s* ad, int index)
{
        player_stop(ad->player);
        // Close file
        player_unprepare(ad->player);

        const char* file = file_name[index];
        // Get full path of resource file
        const char *res_path = get_resource_path(file);

        // Load file
        player_set_uri(ad->player, res_path);
        player_set_display(ad->player, PLAYER_DISPLAY_TYPE_EVAS, GET_DISPLAY(ad->
video_rect));
        player_set_display_mode(ad->player, PLAYER_DISPLAY_MODE_FULL_SCREEN);
        // Prepare play
        int result = player_prepare(ad->player);
        dlog_print(DLOG_INFO, "tag", "File load : %d", result);
}
```

get_player_state() 函数返回 Player 的当前状态。

player_get_state(player_h, player_state_e) 是一种返回 Player 当前状态的 API。第一个参数是 Player 对象，第二个参数返回状态值。player_state_e 状态类型如下：

- PLAYER_STATE_NONE, /**< Player is not created */
- PLAYER_STATE_IDLE, /**< Player is created, but not prepared */
- PLAYER_STATE_READY, /**< Player is ready to play media */

- PLAYER_STATE_PLAYING, /**< Player is playing media */
- PLAYER_STATE_PAUSED, /**< Player is paused while playing media */

on_player_completed() 是一种在完成播放时调用的回调函数。

player_stop(player_h) 是一种用于停止播放的 API。

create_player() 函数创建 Player 对象。

player_create(player_h *) 是一种创建 Player 对象的 API。

player_set_sound_type(player_h, sound_type_e) 是一种指定声音类型的 API。要播放视频文件，只需指定 SOUND_TYPE_MEDIA。

player_set_volume(player_h, float, float) 是一种指定扬声器音量的 API。音量值的范围为 0 至 1.0。第二个参数为左侧音量，第三个参数为右侧音量。

player_set_looping(player_h, bool) 是一种确定是否循环播放的 API。如果将 True 传递给第二个参数，则循环播放。传递 False，则只播放一次。

player_set_completed_cb(player_h, player_completed_cb, void *) 是一种指定在完成播放时执行的回调函数的 API。第二个参数为回调函数的名称，而第三个参数为用户数据。

stop_player() 函数停止播放。

prepare_player() 函数加载视频文件。

player_unprepare(player_h) 是一种用于关闭您加载到 Player 的文件的 API。

player_set_uri(player_h, const char *) 是一种用于将文件加载到 Player 的 API。

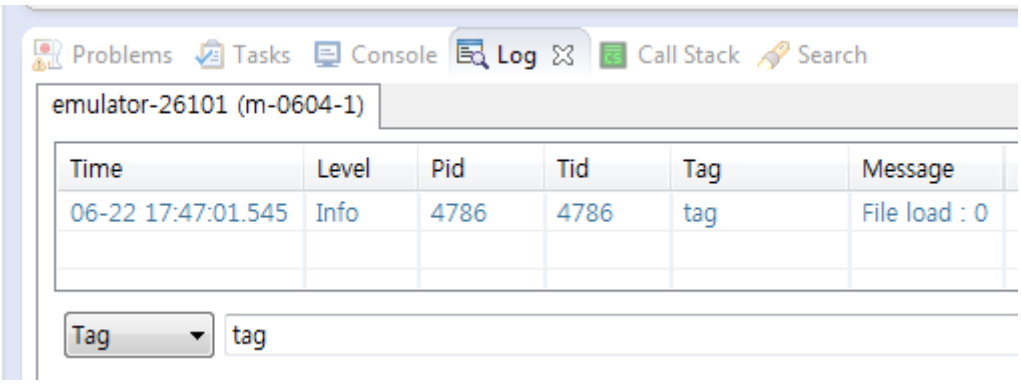
player_set_display(player_h, player_display_type_e, player_display_h) 函数为 Player 指定屏幕。第一个参数指定 Player 对象，第二个参数指

定屏幕类型。如果是基于画布的屏幕，则您可以指定 PLAYER_DISPLAY_TYPE_EVAS。第三个参数传递与屏幕对应的 player_display_h 对象。

GET_DISPLAY() 函数向对象请求 player_display_h。

player_prepare(player_h) 是一种用于使 Player 做好播放准备的 API。如已准备好播放，则返回 0。

如果您以此方式编译并开始运行，则您不会在屏幕上看到或听到任何内容。如果在 Log 窗口查看此消息，则会显示“File load:0”。



3）播放视频

我们将实施一项功能，以便在单击此 Button 时播放视频文件。在 create_base_gui() 函数的结尾添加新代码。

```
    /* Table */
    ~
    elm_box_pack_end(box, table);
    evas_object_show(table);

    {
        /* Play Button */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Play");
        evas_object_smart_callback_add(btn, "clicked", start_player, (void*)ad);
        my_table_pack(table, btn, 0, 3, 1, 1);
```

```
        /* Container: standard table */
        Evas_Object *tbl = elm_table_add(ad->win);
        evas_object_size_hint_weight_set(tbl, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        evas_object_size_hint_align_set(tbl, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(ad->conform, tbl);
        evas_object_show(tbl);
```

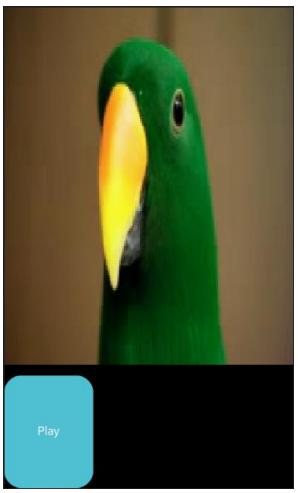
我们已添加一个用于创建 Button 的代码。现在，让我们为此 Button 创建一个回调函数。在 create_base_gui() 函数之上创建一个新函数。

```
// Start play
static void
start_player(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;

        if( get_player_state(ad->player) != PLAYER_STATE_PLAYING)
                player_start(ad->player);
}
```

start_player() 函数用于开始播放视频。

player_start(player_h) 是一种用于开始播放视频的 API。

再次运行该示例，并点击 Play 按钮。这将播放视频。



4) 暂停和停止

我们将添加 2 个 Button 以执行 Pause 和 Stop 功能。向 create_base_gui() 函数添加新代码。

```
        /* Play Button */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Play");
        evas_object_smart_callback_add(btn, "clicked", start_player, (void*)ad);
        my_table_pack(table, btn, 0, 3, 1, 1);

        /* Pause Button */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Pause");
        evas_object_smart_callback_add(btn, "clicked", pause_player, (void*)ad);
        my_table_pack(table, btn, 1, 3, 1, 1);

        /* Stop Button */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Stop");
        evas_object_smart_callback_add(btn, "clicked", stop_player, (void*)ad);
        my_table_pack(table, btn, 2, 3, 1, 1);

        /* Container: standard table */
        Evas_Object *tbl = elm_table_add(ad->win);
        evas_object_size_hint_weight_set(tbl, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        evas_object_size_hint_align_set(tbl, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(ad->conform, tbl);
        evas_object_show(tbl);
```

我们已添加 2 个 Button。现在，我们将为这两个 Button 创建回调函数。在 create_base_gui() 函数之上添加新代码。

```
// Pause play
static void pause_player(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;

        if( get_player_state(ad->player) == PLAYER_STATE_PLAYING )
                player_pause(ad->player);
```

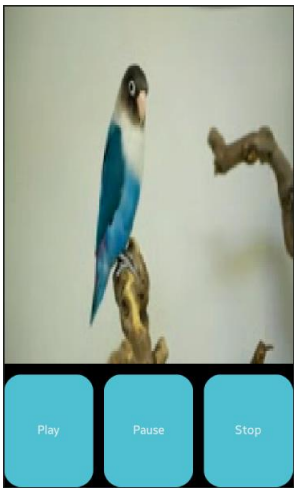
```
}
```

start_player() 函数暂停播放视频。

player_start(player_h) 是一种用于暂停播放视频的 API。

您不必为 Stop Button 另创建一个回调函数。这是因为您已具有 stop_player() 函数。

再次运行此示例，并点击 Play 按钮以播放视频。单击 Pause 按钮可暂停，或单击 Play Again 按钮以重新开始。单击 Stop 按钮可停止，或单击 Play Again 按钮以从头开始播放。



5) 更改视频文件

我们将实施一项功能，以便通过添加 2 个 Button 来更改视频文件。向 create_base_gui() 函数添加新代码。

```
        /* Stop Button */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "Stop");
        evas_object_smart_callback_add(btn, "clicked", stop_player, (void*)ad);
        my_table_pack(table, btn, 2, 3, 1, 1);
```

```
        /* File Load-1 Button */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "File-1");
        evas_object_smart_callback_add(btn, "clicked", btn_load_file1, (void*)a
d);
        my_table_pack(table, btn, 0, 4, 3, 1);

        /* File Load-2 Button */
        btn = elm_button_add(ad->win);
        elm_object_text_set(btn, "File-2");
        evas_object_smart_callback_add(btn, "clicked", btn_load_file2, (void*)a
d);
        my_table_pack(table, btn, 0, 5, 3, 1);

        /* Container: standard table */
        Evas_Object *tbl = elm_table_add(ad->win);
        evas_object_size_hint_weight_set(tbl, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        evas_object_size_hint_align_set(tbl, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(ad->conform, tbl);
        evas_object_show(tbl);
```

我们已创建 2 个 Button。最后，我们准备好为这些 Button 创建回调函数。在 create_base_gui() 函数之上添加新代码。

```
static void
btn_load_file1(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        // Load file to Player
        prepare_player(ad, 0);
}

static void
btn_load_file2(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        // Load file to Player
        prepare_player(ad, 1);
}
```

btn_load_file1() 函数加载第一个视频文件。如果将 0 传递给 prepare_player() 函数的第二个参数，则加载第一个视频文件。

btn_load_file2() 函数加载第一个视频文件。如果将 1 传递给 prepare_player() 函数的第二个参数，则加载第二个视频文件。

再次运行该示例，并点击 File-2 按钮，然后再点击 Play 按钮。您将看到不同的视频。现在，请点击 File-1 按钮，然后再点击 Play 按钮。您将会看到上次播放过的视频。

6）相关 API

int player_get_state(player_h player, player_state_e *state)：一种返回 Player 当前状态的 API。第一个参数是 Player 对象，第二个参数返回状态值。player_state_e 状态类型如下：

```
    - PLAYER_STATE_NONE,                    /**< Player is not created */
    - PLAYER_STATE_IDLE,                    /**< Player is created, but not prepared */
    - PLAYER_STATE_READY,                   /**< Player is ready to play media */
    - PLAYER_STATE_PLAYING,                 /**< Player is playing media */
    - PLAYER_STATE_PAUSED,                  /**< Player is paused whil
```

e playing media */

int player_stop(player_h player)：一种用于停止播放的 API。

int player_create(player_h *player)：一种用于创建 Player 对象的 API。

int player_set_sound_type(player_h player, sound_type_e type)：一种用于指定声音类型的 API。要播放视频文件，只需指定 SOUND_TYPE_MEDIA。

int player_set_volume(player_h player, float left, float right)：一种指定扬声器音量的 API。音量值的范围为 0 至 1.0。/ 参数 - Player 对象、左侧音量、右侧音量。

int player_set_looping(player_h player, bool looping)：一种确定是否循环播放的 API。如果将 True 传递给第二个参数，则循环播放。传递 False，则只播放一次。

int player_set_completed_cb(player_h player, player_completed_cb callback, void *user_data)：一种指定在完成播放时执行的回调函数的 API。/ 参数 - Play 对象、回调函数名称、用户数据。

int player_unprepare(player_h player)：一种用于关闭您加载到 Player 的文件的 API。

int player_set_uri(player_h player, const char * uri)：一种用于将文件加载到 Player 的 API。

int player_prepare(player_h player)：一种用于使 Player 做好播放准备的 API。如已准备好播放，则返回 0。

int player_start(player_h player)：一种用于开始播放视频的 API。

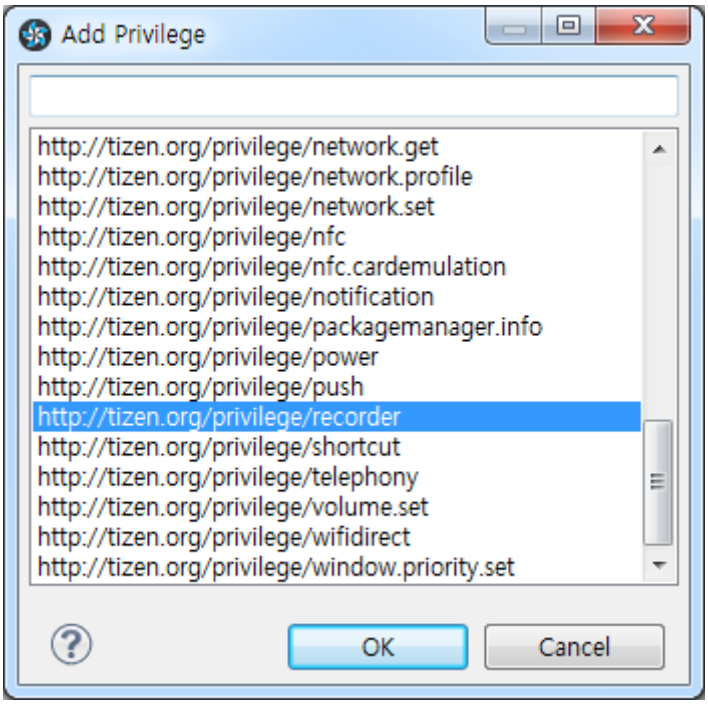
int player_pause(player_h player)：一种用于暂停播放视频的 API。

40. 录制音频

您可以使用智能手机随附的麦克风录音。您可以在音乐会上录音、录制您自己的声音以发送给别人（而不是发送文本消息），或者将其用作记事簿。让我们通过示例详细了解如何进行此操作。

1）注册权限

创建一个新的源项目，并将项目名称指定为“RecorderEx”。您需要拥有用户权限才能使用录音机。创建源项目之后，打开 tizen-manifest.xml 文件，在以下选项卡按钮中，单击 Privileges。然后，点击右上角的 Add 按钮。在弹出窗口上，从列表中选择 http://tizen.org/privilege/recorder，然后单击 OK 按钮以关闭窗口。



保存之后，点击底部选项卡按钮最右角的 tizen-manifest.xml 按钮，就能看到 xml 文件的源代码。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.recorderex" version="1.0.0">
        <profile name="mobile"/>
        <ui-application appid="org.example.recorderex" exec="recorderex" multiple="false" nodisplay="false" taskmanage="true" type="capp">
                <label>recorderex</label>
                <icon>recorderex.png</icon>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/recorder</privilege>
        </privileges>
</manifest>
```

2) 创建 Recorder

要创建 Recorder，您必须指定编解码器、文件类型和质量。打开 src 文件夹中的源文件 (~.c)，并在屏幕顶端添加库、变量和结构。

```
#include "recorderex.h"
#include <recorder.h>
#include <player.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;

        Evas_Object *btn_rec, *btn_recstop, *btn_play, *btn_playstop;

        player_h player;
        recorder_h recorder;
        recorder_audio_codec_e *codec_list;
        int codec_list_len;
        char file_path[PATH_MAX];
        recorder_audio_codec_e codec;
        recorder_file_format_e file_format;
        FILE *preproc_file;
} appdata_s;

typedef struct
{
```

```
        recorder_audio_codec_e *codec_list;
        int len;
} supported_encoder_data;
```

在本例中，我们将总共创建 4 个 Button 小部件。您已声明 4 个变量（btn_rec、btn_recstop、btn_play、btn_playstop）以更改每个 Button 的启用状态。

player_h 是一种我们在 AudioPlayer 和 VideoPlayer 示例中使用过的 Player 结构。

recorder_h 是一种 Recorder 结构。

recorder_audio_codec_e 是一种用于保存编解码器类型的枚举变量。

codec_list_len 是一种用于保存编解码器列表数目的变量。

file_path[] 是一种用于保存所录制文件的路径的变量。

recorder_file_format_e 是一种用于保存文件类型信息的枚举变量。

FILE 是一种用于文件输入/输出的文件数据流。

supported_encoder_data 是一种用于保存受支持编解码器列表的结构。

现在，我们将创建一个 Recorder 对象。在 create_base_gui() 函数的结尾添加新代码。

```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        // Create recorder
        _recorder_create(ad);
        ad->codec_list = audio_recorder_get_supported_encoder(ad->recorder, &ad->codec_list_len);
        ad->codec = ad->codec_list_len ? ad->codec_list[0] : RECORDER_AUDIO_CODEC_PCM;

        _codec_set(ad, codec);
```

```
}
```

_recorder_create() 函数创建 Recorder。

audio_recorder_get_supported_encoder() 函数请求受支持的编解码器列表并将其返回。

如果您拥有编解码器列表，则将第一个编解码器保存到名为 CODEC 的变量。如果列表不存在，请保存 PCM 编解码器。

_codec_set() 函数指定编解码器。

我们将输入创建 Recorder 所需的代码。在 create_base_gui() 函数之上添加八个新函数。

```
// Check is recording
static bool
_recorder_is_recording(appdata_s *ad)
{
        recorder_state_e state = RECORDER_STATE_NONE;
        recorder_get_state(ad->recorder, &state);
        return state == RECORDER_STATE_RECORDING;
}

// Stop recording
static void
record_stop(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    if (ad->recorder)
    {
        recorder_commit(ad->recorder);
        // Check is recording
        if (!_recorder_is_recording(ad))
        {
            recorder_unprepare(ad->recorder);
        }
        elm_object_disabled_set(ad->btn_play, EINA_FALSE);
        elm_object_disabled_set(ad->btn_rec, EINA_FALSE);
        elm_object_disabled_set(ad->btn_playstop, EINA_TRUE);
        elm_object_disabled_set(ad->btn_recstop, EINA_TRUE);
```

```
    }
}

// Maximum recording time event callback function
static void
_on_recording_limit_reached_cb(recorder_recording_limit_type_e type, void *user_
data)
{
        appdata_s *ad = user_data;
        if(type == RECORDER_RECORDING_LIMIT_TIME)
                // Stop recording
                record_stop(ad, NULL, NULL);
}

// Create recorder
static void
_recorder_create(appdata_s *ad)
{
        if(recorder_create_audiorecorder(&ad->recorder) == RECORDER_ERROR_NONE)
        {
                // Set maximum recording time event callback function
                recorder_set_recording_limit_reached_cb(ad->recorder, _on_record
ing_limit_reached_cb, ad);
                recorder_attr_set_audio_channel(ad->recorder, 1);
                recorder_attr_set_audio_device(ad->recorder, RECORDER_AUDIO_DEVI
CE_MIC);
                recorder_attr_set_time_limit(ad->recorder, 20);
        }
}

static bool
_recorder_supported_audio_encoder_cb(recorder_audio_codec_e codec, void *user_da
ta)
{
        bool result = false;
        supported_encoder_data *data = user_data;

        if(data && codec != RECORDER_AUDIO_CODEC_DISABLE)
        {
                data->codec_list = realloc(data->codec_list, sizeof(supported_en
coder_data) * (data->len + 1));
                data->codec_list[data->len] = codec;
                ++(data->len);
                result = true;
        }
```

```
        return result;
}

recorder_audio_codec_e*
audio_recorder_get_supported_encoder(recorder_h recorder, int *list_length)
{
        supported_encoder_data data = {0};
        data.codec_list = NULL;
        data.len = 0;

        int res = recorder_foreach_supported_audio_encoder(recorder, _recorder_s
upported_audio_encoder_cb, &data);

        if(res && list_length)
        {
                *list_length = data.len;
        }

        return data.codec_list;
}

const char*
get_file_format_by_codec(appdata_s* ad)
{
        switch(ad->codec)
        {
        case RECORDER_AUDIO_CODEC_AMR:
                ad->file_format = RECORDER_FILE_FORMAT_AMR;
                return "AMR";
                break;
        case RECORDER_AUDIO_CODEC_AAC:
                ad->file_format = RECORDER_FILE_FORMAT_MP4;
                return "MP4";
                break;
        case RECORDER_AUDIO_CODEC_VORBIS:
                ad->file_format = RECORDER_FILE_FORMAT_OGG;
                return "OGG";
                break;
        }

        ad->file_format = RECORDER_FILE_FORMAT_WAV;
        return "WAV";
}

static void
_codec_set(appdata_s *ad)
```

```
{
        char file_name[NAME_MAX] = {'\0'};
        const char *file_ext = get_file_format_by_codec(ad);

        char *data_path = app_get_data_path();
        snprintf(file_name, NAME_MAX, "record.%s", file_ext);
        snprintf(ad->file_path, PATH_MAX, "%s%s", data_path, file_name);
        free(data_path);
}
```

_recorder_is_recording() 函数检查录音的状态。

record_stop() 函数停止录音。

recorder_commit(recorder_h recorder) 是一种用于停止录音和保存数据的 API。

recorder_unprepare(recorder_h recorder) 是一种用于初始化 Recorder 的 API。

_on_recording_limit_reached_cb() 是一种在达到最大录音时间时调用的事件函数。它将会强制停止录音。

_recorder_create() 函数创建 Recorder。

recorder_create_audiorecorder() 是一种创建 Recorder 的 API。

recorder_set_recording_limit_reached_cb() 是一种用于达到最大录音时间的 API。

recorder_attr_set_audio_channel() 是一种指定音频通道数目的 API。在 Mono 中指定 1，在 Stereo 中指定 2。

recorder_attr_set_audio_device() 是一种指定录音设备的 API。在您的智能手机上，如果使用的是麦克风，则可以指定 RECORDER_AUDIO_DEVICE_MIC。

recorder_attr_set_time_limit() 是一种指定最大录音时间（单位：秒）的 API。

_recorder_supported_audio_encoder_cb() 是一种用于接收受支持的音频编解码器列表的回调函数。

audio_recorder_get_supported_encoder() 函数请求受支持的编解码器列表并将其返回。

recorder_foreach_supported_audio_encoder(recorder_h recorder, recorder_supported_audio_encoder_cb callback, void *user_data) 是一种请求受支持的编解码器列表的 API。它将列表数据传递给此回调函数，而不是立即请求列表。

get_file_format_by_codec() 函数返回文件格式或文件扩展名，具体取决于编解码器类型。

_codec_set() 函数指定编解码器。

3) 录音快速入门

我们将为屏幕添加 2 个 Button 以执行 Pause 和 Stop Recording 功能。向 create_base_gui() 函数添加新代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        Evas_Object *btn, *frame, *tbl;

        /* Frame for some outer padding */
        frame = elm_frame_add(ad->conform);
        elm_object_style_set(frame, "pad_medium");
        elm_object_content_set(ad->conform, frame);
        evas_object_show(frame);
```

```
        /* Table to pack our elements */
        tbl = elm_table_add(frame);
        elm_table_padding_set(tbl, 5 * elm_scale_get(), 5 * elm_scale_get());
        elm_object_content_set(frame, tbl);
        evas_object_show(tbl);

        {
            /* Just a label */
            ad->label = elm_label_add(tbl);
            elm_object_text_set(ad->label, "Audio recorder");
            evas_object_size_hint_align_set(ad->label, 0.5, 0.5);
            evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_
HINT_EXPAND);
            elm_table_pack(tbl, ad->label, 0, 0, 2, 1);
            evas_object_show(ad->label);

            /* Record Start Button */
            btn = elm_button_add(tbl);
            elm_object_text_set(btn, "Recording Start");
            evas_object_smart_callback_add(btn, "clicked", record_start, (void*)
ad);
            evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, 0);
            evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0.5);
            elm_table_pack(tbl, btn, 0, 1, 1, 1);
            evas_object_show(btn);
            ad->btn_rec = btn;

            /* Record Stop Button */
            btn = elm_button_add(tbl);
            elm_object_disabled_set(btn, EINA_TRUE);
            elm_object_text_set(btn, "Recording Stop");
            evas_object_smart_callback_add(btn, "clicked", record_stop, (void*)
ad);
            evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, 0);
            evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0.5);
            elm_table_pack(tbl, btn, 1, 1, 1, 1);
            evas_object_show(btn);
            ad->btn_recstop = btn;
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

我们创建了一个 Table 容器以便根据屏幕高宽比放置此小部件，并已使用 Frame 指定了外边界。

我们将实施一项开始录音的功能。在 create_base_gui() 函数之上添加两个新函数。

```
// Apply settings to recorder
static void _recorder_apply_settings(appdata_s *ad)
{
        if(ad->recorder)
        {
                // Set record file name
                recorder_set_filename(ad->recorder, ad->file_path);
                // Set record file format
                recorder_set_file_format(ad->recorder, ad->file_format);
                // Set record codec
                recorder_set_audio_encoder(ad->recorder, ad->codec);
        }
}

// Start recording
static void
record_start(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    if (ad->recorder)
    {
        // Apply settings to recorder
        _recorder_apply_settings(ad);
        recorder_prepare(ad->recorder);
        recorder_start(ad->recorder);
        elm_object_disabled_set(ad->btn_recstop, EINA_FALSE);
        elm_object_disabled_set(ad->btn_rec, EINA_TRUE);
        elm_object_disabled_set(ad->btn_play, EINA_TRUE);
        elm_object_disabled_set(ad->btn_playstop, EINA_TRUE);
    }
}
```

_recorder_apply_settings() 函数指定有关录音的信息。

recorder_set_filename() 是一种指定录音文件名称的 API。

recorder_set_file_format() 是一种指定录音格式的 API。

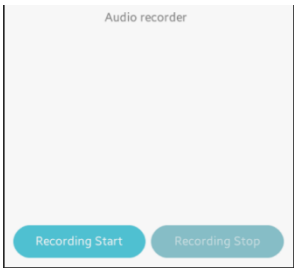
recorder_set_audio_encoder() 是一种指定编解码器的 API。

record_start() 函数开始录音。

recorder_prepare() 是一种做好录音准备的 API。

recorder_start() 是一种用于开始录音的 API。

编译示例之后，运行它并点击 Rec Start 按钮以开始录音。录音将在 20 秒之后自动结束。您还可以通过单击 Rec Stop 按钮手动停止录音。不幸的是，我们无法听到录制的音频文件。我们尚未实施音频播放功能。



4) 播放录音

我们将实施一项播放音频文件的功能。在 create_base_gui() 函数中添加一个用于创建 2 个新 Button 的代码。

```
            /* Record Stop Button */
            ~
            evas_object_show(btn);
            ad->btn_recstop = btn;

            /* Play Start Button */
            btn = elm_button_add(tbl);
            elm_object_disabled_set(btn, EINA_TRUE);
            elm_object_text_set(btn, "Play Start");
            evas_object_smart_callback_add(btn, "clicked", start_player, (void*)
ad);
```

```
            evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
            evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0.0);
            elm_table_pack(tbl, btn, 0, 2, 1, 1);
            evas_object_show(btn);
            ad->btn_play = btn;

            /* Play Stop Button */
            btn = elm_button_add(tbl);
            elm_object_disabled_set(btn, EINA_TRUE);
            elm_object_text_set(btn, "Play Stop");
            evas_object_smart_callback_add(btn, "clicked", stop_player, (void*)
ad);
            evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
            evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0.0);
            elm_table_pack(tbl, btn, 1, 2, 1, 1);
            evas_object_show(btn);
            ad->btn_playstop = btn;
        }
    }

    // create player
    ad->player = create_player();

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

create_player() 函数创建一个 Player。现在开始吧！在 create_base_gui() 函数之上添加五个新函数。这与 AudioPlayer 示例中的源代码基本相同。

```
// Get player state
static player_state_e get_player_state(player_h player)
{
        player_state_e state;
        player_get_state(player, &state);
        return state;
}

// Stop play
static void
stop_player(void *data, Evas_Object *obj, void *event_info)
```

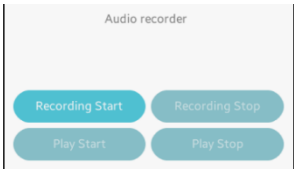
```
{
    appdata_s *ad = data;

    if( get_player_state(ad->player) == PLAYER_STATE_PLAYING || get_player_state
(ad->player) == PLAYER_STATE_PAUSED)
        player_stop(ad->player);

    elm_object_disabled_set(ad->btn_play, EINA_FALSE);
    elm_object_disabled_set(ad->btn_playstop, EINA_TRUE);
    elm_object_disabled_set(ad->btn_rec, EINA_FALSE);
    elm_object_disabled_set(ad->btn_recstop, EINA_TRUE);
}

// Load file to player
static void
prepare_player(appdata_s* ad)
{
        stop_player(ad, NULL, NULL);
        player_unprepare(ad->player);
        player_set_uri(ad->player, ad->file_path);
        player_prepare(ad->player);
}

// Start play
static void
start_player(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    prepare_player(ad);

    if (get_player_state(ad->player) != PLAYER_STATE_PLAYING)
    {
        player_start(ad->player);

        elm_object_disabled_set(ad->btn_rec, EINA_TRUE);
        elm_object_disabled_set(ad->btn_recstop, EINA_TRUE);
        elm_object_disabled_set(ad->btn_play, EINA_TRUE);
        elm_object_disabled_set(ad->btn_playstop, EINA_FALSE);
    }
}

// Create player
static player_h create_player()
{
        player_h player;
```

```
        player_create(&player);
        player_set_completed_cb(player, NULL, player);

        return player;
}
```

请参阅 AudioPlayer 示例，以了解更多详细信息。

运行示例。如果通过模拟器在台式机上测试，则必须连接麦克风。在手机或笔记本电脑上，则不需要麦克风即可进行测试。请按照以下步骤进行测试：

- 单击 Record Start 按钮以录音。
- 单击 Record Stop 按钮以停止录音。录音将在 20 秒之后自动停止。
- Play Start 按钮将播放录制的录音。
- 如果要在中途停止录音，请点击 Play Stop 按钮。



5) 相关 API

int preference_set_int(const char *key, int value)：一种将整数值保存到本机内存的 API。它将键值传递给第一个参数，并将数据传递给第二个参数。

int preference_set_boolean(const char *key, bool value)：是一种将布尔值保存至本机内存的 API。它将键值传递给第一个参数，并将数据传递给第二个参数。

int preference_get_boolean(const char *key, bool *value)：是一种向本机内存请求布尔类型数据的 API。当它将键值传递给第一个参数时，第二个参数将返回结果值。

int preference_get_int(const char *key, int *value)：是一种向本机内存请求整数类型数据的 API。当它将键值传递给第一个参数时，第二个参数将返回结果值。

int recorder_create_audiorecorder(recorder_h *recorder)：是一种创建 Recorder 的 API。

int recorder_set_recording_status_cb(recorder_h recorder, recorder_recording_status_cb callback, void *user_data)：是一种指定更改录音状态的事件函数的 API。

int recorder_set_recording_limit_reached_cb(recorder_h recorder, recorder_recording_limit_reached_cb callback, void *user_data)：是一种指定达到最大录音时间的事件函数的 API。

int recorder_attr_set_audio_channel(recorder_h recorder, int channel_count)：是一种指定音频通道数的 API。在 Mono 中指定 1，在 Stereo 中指定 2。

int recorder_attr_set_audio_device(recorder_h recorder, recorder_audio_device_e device)：是一种指定录音设备的 API。在您的智能手机上，如果使用的是麦克风，则可以指定 RECORDER_AUDIO_DEVICE_MIC。

int recorder_attr_set_time_limit(recorder_h recorder, int second)：是一种指定最大录音时间（单位：秒）的 API。

int recorder_set_filename(recorder_h recorder, const char *path)：是一种指定录音文件名称的 API。

int recorder_set_file_format(recorder_h recorder, recorder_file_format_e format) 是一种指定录音格式的 API。

int recorder_set_audio_encoder(recorder_h recorder, recorder_audio_codec_e codec)：是一种指定编解码器的 API。

int recorder_prepare(recorder_h recorder)：是一种做好录音准备的 API。

int recorder_start(recorder_h recorder)：是一种用于开始录音的 API。

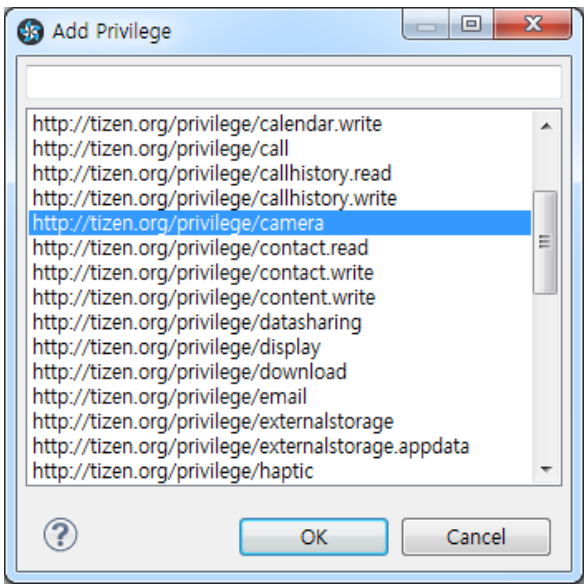
41. 摄像头截屏

随着智能手机的内置摄像头性能变得越来越快速稳定，数码相机开始过时了。事实上，智能手机的摄像头性能已取得显著进步，其效果毫不逊色于同级别的传统数码相机。如今，消费者在选择智能手机时越来越注重摄像头的性能。内置摄像头不仅可用于拍照，还可用于执行视频电话、扩增实境或条码扫描仪等其他各类操作。

在本小节，我们将使用手机上自带的摄像头在 Preview 模式下观看视频。此外，我们还将学习如何拍照并将其另存为文件。

1）注册权限

创建一个新的源项目，并将项目名称指定为“CameraEx”。您需要拥有用户权限才能使用摄像头。创建源项目之后，打开 tizen-manifest.xml 文件，在以下选项卡按钮中，单击 Privileges。然后，点击右上角的 Add 按钮。在弹出窗口上，从列表中选择 http://tizen.org/privilege/camera，然后单击 OK 按钮以关闭窗口。

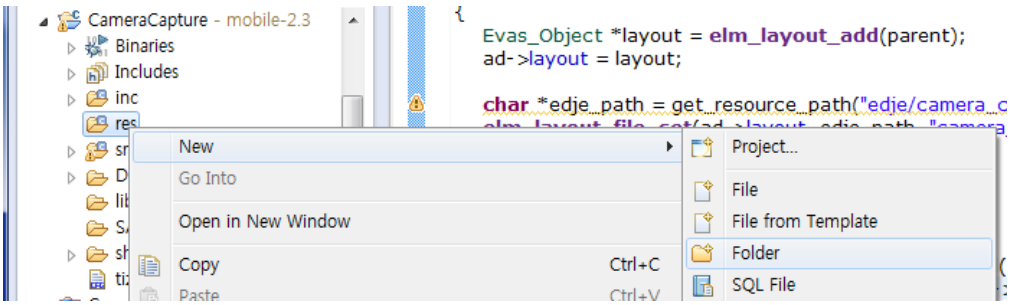


保存之后，点击底部选项卡按钮最右角的 tizen-manifest.xml 按钮，就能看到 xml 文件的源代码。

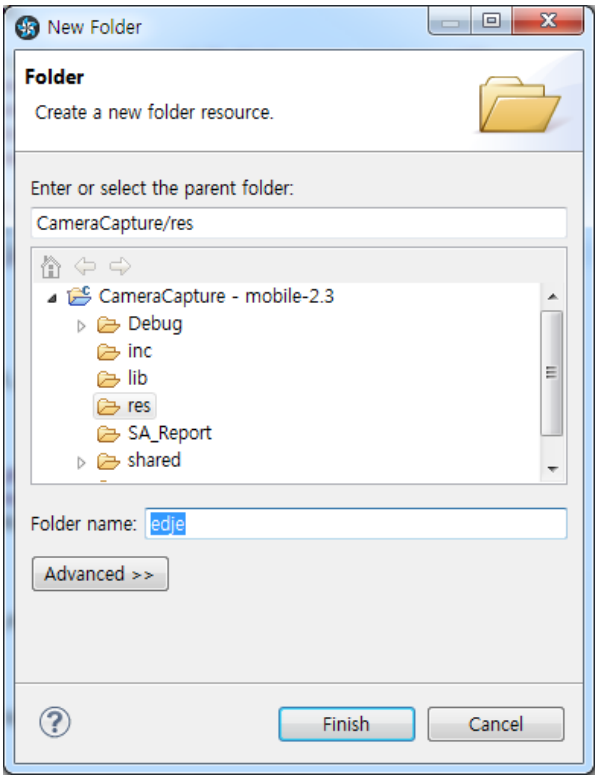
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.cameraex" version="1.0.0">
        <profile name="mobile"/>
        <ui-application appid="org.example.cameraex" exec="cameraex" multiple="false" nodisplay="false" taskmanage="true" type="capp">
                <label>cameraex</label>
                <icon>cameraex.png</icon>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/camera</privilege>
        </privileges>
</manifest>
```

您需要一个屏幕才能播放预览视频。现在，我们将复制 EDJE 文件。执行下列操作：

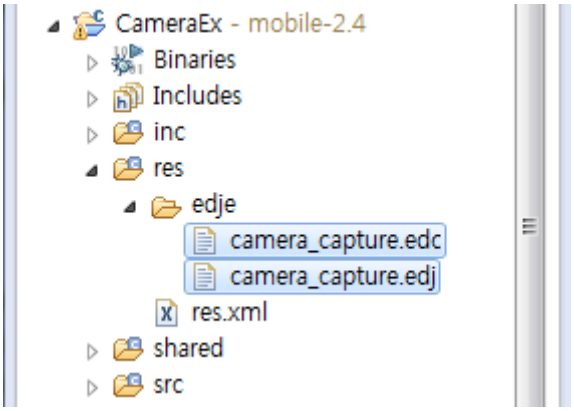
- 右键单击 /res 文件夹，在快捷菜单中选择 [New > Folder]。



- 弹出提示后，请在文件夹名称属性中输入“edje”。



- 将位于附录 /etc/edje 文件夹中的 camera_capture.edc 和 camera_capture.edj 文件复制到新文件夹。



2) 摄像头预览视频

打开 src 文件夹中的源文件 (~.c)，并在屏幕顶端添加库和变量。

```
#include "cameracapture.h"
#include <camera.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *layout;
        Evas_Object *camera_rect;
        Evas_Object *image;
        Evas_Object *box;

        camera_h camera;
        char *image_path;
} appdata_s;
```

预览视频与我们在 VideoPlayer 示例中使用的屏幕相似。Layout 为预览区域，其中，camera_rect 是用于显示预览视频的 Image 对象。

我们将在 Image 对象中显示拍摄的照片。

camera_h 是一种 Camera 结构。

image_path 是您在其中保存所拍摄照片的文件路径。

现在，我们将创建 Camera 对象并在屏幕上显示预览视频。在 create_base_gui() 函数之上创建一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
                      double h_weight, double v_weight, double h_align, double
 v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
```

```
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

然后，将新代码添加到 create_base_gui() 函数中。我们无需使用用于创建 Label 的代码，因此，我们可以为其添加注释。

```
static void
create_base_gui(appdata_s *ad)
{
    /* first say that we prefer acceleration via opengl - before we create any
windows */
    elm_config_accel_preference_set("opengl");

    ~


    eext_object_event_callback_add(ad->win, EEXT_CALLBACK_BACK, win_back_cb, ad);

    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    /* Label*/
```

```
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
    elm_object_content_set(ad->conform, ad->label);*/

    { /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        ad->box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(ad->box, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
        elm_object_content_set(ad->conform, ad->box);
        evas_object_show(ad->box);

        { /* child object - indent to how relationship */
            /* Create preview screen */
            Evas_Object *layout = _main_layout_add(ad, ad->win);
            my_box_pack(ad->box, layout, 0.9, 1.0, -1.0, -1.0);
         }
    }

    _create_camera(ad);
    // Start camera preview
    camera_start_preview(ad->camera);

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

要显示摄像头预览，必须使用 OpenGL 库。您只需将“opengl”传递给 elm_config_accel_preference_set() 函数，即可进行此操作。

_main_layout_add() 函数创建一个属于预览区域的 Layout 对象。

_create_camera() 函数创建一个 Camera 对象。

camera_start_preview() 是一种用于启动预览的 API。

我们将开始创建上面提到的函数。在 create_base_gui() 函数之上添加四个新函数。

```
static inline const char*
```

```
get_resource_path(const char *file_path)
{
    static char absolute_path[PATH_MAX] = "";
    static char *res_path_buff = NULL;
    if (res_path_buff == NULL)
        res_path_buff = app_get_resource_path();
    snprintf(absolute_path, sizeof(absolute_path), "%s%s", res_path_buff, file_p
ath);
    return absolute_path;
}

// Create preview screen
static Evas_Object*
_main_layout_add(appdata_s *ad, Evas_Object *parent)
{
    Evas_Object *layout = elm_layout_add(parent);
    ad->layout = layout;

    char *edje_path = get_resource_path("edje/camera_capture.edj");
    elm_layout_file_set(ad->layout, edje_path, "camera_capture");

    Evas *evas = evas_object_evas_get(parent);
    ad->camera_rect = evas_object_image_filled_add(evas);
    elm_object_part_content_set(layout, "render", ad->camera_rect);

    return layout;
}

static void _destroy_camera(appdata_s *ad)
{
        if(ad->camera)
        {
                camera_stop_preview(ad->camera);
                camera_destroy(ad->camera);
                ad->camera = NULL;
        }
}

static void
_create_camera(appdata_s *ad)
{
        if(ad->camera)
                _destroy_camera(ad);

        if(camera_create(CAMERA_DEVICE_CAMERA0, &ad->camera) == CAMERA_ERROR_NON
E)
```

```
        {
                camera_set_capture_format(ad->camera, CAMERA_PIXEL_FORMAT_JPEG);
                camera_set_display(ad->camera, CAMERA_DISPLAY_TYPE_EVAS, GET_DIS
PLAY(ad->camera_rect));
                camera_set_display_mode(ad->camera, CAMERA_DISPLAY_MODE_FULL);
                camera_set_display_rotation(ad->camera, CAMERA_ROTATION_270);
                camera_set_display_flip(ad->camera, CAMERA_FLIP_VERTICAL);
        }
        else
                ad->camera = NULL;
}
```

get_resource_path() 返回位于 /res 文件夹中的文件的绝对路径。

_main_layout_add() 函数创建一个属于预览区域的 Layout 对象。具体操作与 VideoPlayer 示例重叠。

_destroy_camera() 函数删除 Camera 对象。

camera_stop_preview(camera_h) 是一种用于停止摄像头预览的 API。

camera_destroy(camera_h) 是一种用于删除 Camera 对象的 API。

_create_camera() 函数创建和预览 Camera 对象。

camera_create(camera_device_e, camera_h *) 是一种用于创建 Camera 对象的 API。如果将 CAMERA_DEVICE_CAMERA0 传递给第一个参数，则会使用后置摄像头。如果传递的是 CAMERA_DEVICE_CAMERA1，则使用前置摄像头。所创建的 Camera 对象将返回至第二个参数。

camera_set_capture_format(camera_h, camera_pixel_format_e) 是一种指定手机图像格式的 API。图片尺寸一般较大，因此建议使用 JPEG 格式。

camera_set_display(camera_h, camera_display_type_e, camera_display_h) 是一种为 Camera 对象指定预览屏幕的 API。使用基于 Evas 的对象时，将 CAMERA_DISPLAY_TYPE_EVAS 传递给第二个参数。通过将 Image 对象传递给 GET_DISPLAY() 函数，将您请求的 camera_display_h 对象传递给第三个参数。

camera_set_display_mode(camera_h, camera_display_mode_e) 是一种指定

预览视频的扩展/折叠选项的 API。该选项的类型如下：

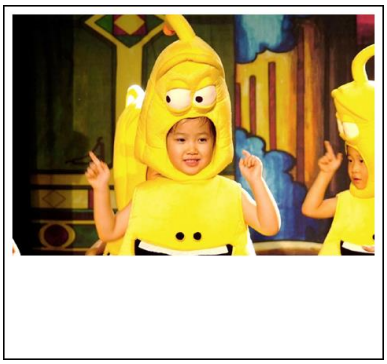
- CAMERA_DISPLAY_MODE_LETTER_BOX = 0, /**< Letter box */
- CAMERA_DISPLAY_MODE_ORIGIN_SIZE, /**< Origin size */
- CAMERA_DISPLAY_MODE_FULL, /**< Full screen */
- CAMERA_DISPLAY_MODE_CROPPED_FULL, /**< Cropped full screen */

camera_set_display_rotation(camera_h, camera_rotation_e) 是一种指定摄像头旋转角度的 API。

camera_set_display_flip(camera_h, camera_flip_e) 是一种用于指定垂直/水平方向的 API。该选项的类型如下：

- CAMERA_FLIP_NONE, /**< No Flip */
- CAMERA_FLIP_HORIZONTAL, /**< Horizontal flip */
- CAMERA_FLIP_VERTICAL, /**< Vertical flip */
- CAMERA_FLIP_BOTH /** Horizontal and vertical flip */

构建并运行一个示例。您必须在连接有智能手机或网络摄像头的 PC 上进行测试。预览视频将显示在屏幕顶部。



3) 固定屏幕方向

在水平位置转动屏幕时，预览也将随之改变。这是因为屏幕方向已从 Portrait 更改成 Landscape。现在，我们将屏幕方向固定。在 create_base_gui() 函数上方是指定屏幕方向类型的代码。执行如下更改：

```
        if (elm_win_wm_rotation_supported_get(ad->win)) {
                //int rots[4] = { 0, 90, 180, 270 };
                int rots[4] = { 0 };
                //elm_win_wm_rotation_available_rotations_set(ad->win, (const int *)(&rots), 4);
                elm_win_wm_rotation_available_rotations_set(ad->win, (const int *)(&rots), 1);
        }
```

0 表示与 180 相反的 Portrait Primary，而 180 表示 Portrait Secondary。90 表示 Landscape Primary，而 270 表示 Landscape Secondary。目前只支持 Portrait Primary 模式。

当您退出应用程序时，Camera 对象将会自动删除。在源文件下，向 app_terminate() 函数添加新代码。

```
static void
app_terminate(void *data)
{
        _destroy_camera(data);
}
```

app_terminate() 是一种在退出应用程序时执行的回调函数。您可在 main() 函数中更改此回调函数。

如果再次运行示例，并旋转屏幕，则预览的位置不再转变。

4) 随屏幕方向旋转预览

这次我们将演示随屏幕方向自动旋转摄像头的功能。在源文件下，修改用于 main() 函数和 ui_app_orient_changed() 的代码。

```
static void
ui_app_orient_changed(app_event_info_h event_info, void *user_data)
{
        appdata_s *ad = user_data;
        }

        return ret;
}
```

在 main() 函数中，指定用于将屏幕方向更改为 ui_app_orient_changed() 的事件函数。如果此代码不存在，则添加一个此代码。

然后，将新代码添加到 ui_app_orient_changed() 函数中。

app_event_get_device_orientation() 是一种用于从事件对象请求屏幕方向的 API。APP_DEVICE_ORIENTATION_0 或 APP_DEVICE_ORIENTATION_180 是 Portrait 模式；APP_DEVICE_ORIENTATION_90 或 APP_DEVICE_ORIENTATION_270 是 Landscape 模式。

camera_set_display_rotation() 是一种用于旋转摄像头的 API。

elm_win_rotation_with_resize_set() 是一种用于调整窗口大小以便与屏幕方向保持一致的 API。

5) 摄像头截屏

我们将实施一项功能，以将预览视屏截屏，并通过单击 Button 将其另存为图像文件。将 Button 创建代码添加到 create_base_gui() 函数。

```
        { /* child object - indent to how relationship */
            /* Create preview screen */
```

```
            Evas_Object *layout = _main_layout_add(ad, ad->win);
            my_box_pack(ad->box, layout, 0.9, 1.0, -1.0, -1.0);

            /* Capture button */
            Evas_Object *btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "#");
            evas_object_smart_callback_add(btn, "clicked", btn_capture_cb, ad);
            my_box_pack(ad->box, btn, 0.1, 0.0, -1.0, 0.5);
        }
    }
```

然后，在 create_base_gui() 函数之上添加五个新函数。

```
static inline char*
gen_data_path(const char *file_name)
{
    static char *absolute_path = NULL;
    char result[PATH_MAX] = "";
    if (absolute_path == NULL)
       absolute_path = app_get_data_path();
    snprintf(result, sizeof(result), "%s/%s", absolute_path, file_name);
    return strdup(result);
}

// Save image data to file
static char*
_save_file(appdata_s *ad, camera_image_data_s *image)
{
    char buf[PATH_MAX] = "";
    snprintf(buf, PATH_MAX, "camera_capture.jpg");
    char *file_name = gen_data_path(buf);

    FILE *f = fopen(file_name, "w");

    if(f)
    {
        fwrite(image->data, image->size, 1, f);
        fclose(f);
    }
    else
    {
        free(file_name);
        file_name = NULL;
```

```
    }
    return file_name;
}

static void
_on_camera_capture_cb(camera_image_data_s *image, camera_image_data_s *postview,
 camera_image_data_s *thumbnail, void *user_data)
{
    appdata_s *ad = user_data;
    free(ad->image_path);
}

static void _on_camera_capture_completed_cb(void *user_data)
{
        appdata_s *ad = user_data;
        camera_start_preview(ad->camera);
}

static void
btn_capture_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = (appdata_s *)data;
        camera_start_capture(ad->camera, _on_camera_capture_cb, _on_camera_captu
re_completed_cb, ad);
}
```

gen_data_path() 函数将会返回应用程序内的 /data 文件夹中的文件路径。您无法在 /res 文件夹中创建或修改文件。这就是为什么必须将文件另存到 /data 文件夹或公共文件夹中的原因。

app_get_data_path() 是一种可将绝对路径返回至 /data 文件夹的 API。

_save_file() 函数可将图像数据保存到数据文件中。

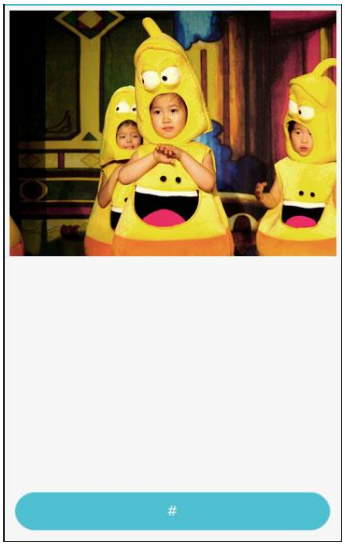
_on_camera_capture_cb() 是一种可接收摄像头截屏数据的回调函数。使用此函数保存数据。

_on_camera_capture_completed_cb() 是一种用于摄像头已完成截屏的事件回调函数。由于预览将在截屏后结束，因此，必须使用此函数重新启动预览。

btn_capture_cb() 是一种 Button 的回调函数。

camera_start_capture(camera_h, camera_capturing_cb, camera_capture_completed_cb, void *) 是一种用于启动摄像头截屏的 API。第一参数是 Camera 对象；第二个参数是数据传输函数；第三个参数是用于完成截屏的事件回调函数；第四个参数是用户数据。

再次运行此示例并点击 Button。如果听到摄像头发出咔嚓声，则意味着截屏成功。显示已保存文件的功能尚未实施。



5）在屏幕上显示已保存的图像文件

我们从在 Image 对象中显示已保存的图像文件开始。将生成 Image 对象的代码添加到 _main_layout_add() 函数。

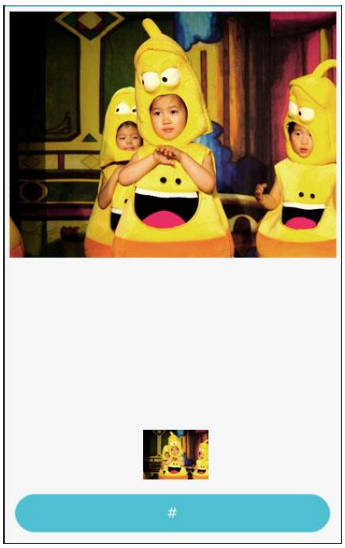
```
// Create preview screen
static Evas_Object*
_main_layout_add(appdata_s *ad, Evas_Object *parent)
{
    ~


    Evas *evas = evas_object_evas_get(parent);
    ad->camera_rect = evas_object_image_filled_add(evas);
    elm_object_part_content_set(layout, "render", ad->camera_rect);
```

```
    ad->image = elm_image_add(parent);
    elm_object_part_content_set(layout, "gallery", ad->image);

    return layout;
}
```

一旦摄像头截屏完成，便将图像文件加载到 Image 对象。将新代码添加到 _on_camera_capture_completed_cb() 函数。

```
static void
_on_camera_capture_cb(camera_image_data_s *image, camera_image_data_s *postview,
 camera_image_data_s *thumbnail, void *user_data)
{
    appdata_s *ad = user_data;
    free(ad->image_path);
    ad->image_path = _save_file(ad, image);
}
```

再次运行该示例，并点击 Button，以在 bg 小部件中显示截图。



6) 相关 API

int camera_start_preview(camera_h camera)：一种用于启动预览的 API。

int camera_stop_preview(camera_h camera)：一种用于停止预览的 API。

int camera_destroy(camera_h camera)：一种用于删除 Camera 对象的 API。

int camera_create(camera_device_e device, camera_h *camera)：一种用于创建 Camera 对象的 API。如果将 CAMERA_DEVICE_CAMERA0 传递给第一个参数，则会使用后置摄像头。如果传递的是 CAMERA_DEVICE_CAMERA1，则使用前置摄像头。所创建的 Camera 对象将返回至第二个参数。

int camera_set_capture_format(camera_h camera, camera_pixel_format_e format)：一种用于指定照片图像格式的 API。CAMERA_PIXEL_FORMAT_JPEG 将创建 JPEG 格式的图像。

int camera_set_display(camera_h camera, camera_display_type_e type, camera_display_h display)：一种用于为 Camera 对象指定预览屏幕的 API。使用基于 Evas 的对象时，将 CAMERA_DISPLAY_TYPE_EVAS 传递给第二个参数。通过将 Image 对象传递给 GET_DISPLAY() 函数，将您请求的 camera_display_h 对象传递给第三个参数。

int camera_set_display_mode(camera_h camera, camera_display_mode_e mode)：一种用于指定预览视频的展开/折叠选项的 API。该选项的类型如下：

- CAMERA_DISPLAY_MODE_LETTER_BOX = 0, /**< Letter box */
- CAMERA_DISPLAY_MODE_ORIGIN_SIZE, /**< Origin size */
- CAMERA_DISPLAY_MODE_FULL, /**< Full screen */
- CAMERA_DISPLAY_MODE_CROPPED_FULL, /**< Cropped full screen */

int camera_set_display_rotation(camera_h camera, camera_rotation_e rotation)：一种用于指定摄像头旋转角度的 API。

int camera_set_display_flip(camera_h camera, camera_flip_e flip)：一种用于指定垂直/水平方向的 API。该选项的类型如下：

- CAMERA_FLIP_NONE, /**< No Flip */
- CAMERA_FLIP_HORIZONTAL, /**< Horizontal flip */
- CAMERA_FLIP_VERTICAL, /**< Vertical flip */
- CAMERA_FLIP_BOTH /** Horizontal and vertical flip */

char *app_get_data_path(void)：一种可将绝对路径返回至 /data 文件夹的 API。

int camera_start_capture(camera_h camera, camera_capturing_cb capturing_cb, camera_capture_completed_cb completed_cb, void *user_data)：一种用于启动摄像头截屏的 API。/ parameter:Camera 对象、数据传输函数、用于完成截屏的事件回调函数、用户数据。

42. 系统信息

在应用程序开发期间，Help 屏幕会经常为您显示系统信息。若要开发将用到摄像头的应用程序，则必须查看您配备的是后置摄像头还是前置摄像头。要让摄像头与分辨率各异的设备兼容，您还必须请求显示器像素数。在本例中，我们将了解如何请求该系统信息。

1) 是否有后置摄像头

创建新的源项目，将项目名称指定为 SystemInfo。创建源项目之后，打开 src 文件夹中的源文件 (*.c)，并在顶端添加一个库头文件和变量。

```
#include "systeminfo.h"
#include <system_info.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label1;
        Evas_Object *label2;
        Evas_Object *label3;
        Evas_Object *label4;
        Evas_Object *label5;
        Evas_Object *label6;
} appdata_s;
```

我们共公布了 6 个 Label 小部件变量。现在，我们将在第一个 Label 中显示是否存在后置摄像头，在第二个 Label 中显示是否存在前置摄像头，在第三个 Label 中显示能否拨打电话，在第四个 Label 中显示水平像素数，在第五个 Label 中显示垂直像素数，在第六个 Label 中显示平台版本。在 create_base_gui() 函数之上创建三个新函数。

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int col, int row, int spanx, int spany,
      bool h_expand, bool v_expand, double h_align, double v_align)
```

```
{
    /* Create a frame around the child, for padding */
    Evas_Object *frame = elm_frame_add(table);
    elm_object_style_set(frame, "pad_small");

    evas_object_size_hint_weight_set(frame, h_expand ? EVAS_HINT_EXPAND : 0, v_e
xpand ? EVAS_HINT_EXPAND : 0);
    evas_object_size_hint_align_set(frame, h_align, v_align);

    /* place child in its box */
    {
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(frame, child);
        evas_object_show(child);
    }

    elm_table_pack(table, frame, col, row, spanx, spany);
    evas_object_show(frame);
}

static Evas_Object *
my_button_add(Evas_Object *parent, const char *text, Evas_Smart_Cb cb, void *cb_
data)
{
    Evas_Object *btn;

    btn = elm_button_add(parent);
    elm_object_text_set(btn, text);
    evas_object_smart_callback_add(btn, "clicked", cb, cb_data);

    return btn;
}

static Evas_Object *
my_label_add(Evas_Object *parent, const char *text)
{
    Evas_Object *btn;

    btn = elm_label_add(parent);
    elm_object_text_set(btn, text);

    return btn;
}
```

my_table_pack() 是将小部件添加到 Table 容器的函数。

my_button_add() 函数可创建 Button 小部件。

my_label_add() 函数可创建 Label 小部件。

将新代码添加到 create_base_gui() 函数中。此代码将创建 1 个 Frame、1 个 Table、1 个 Button 和 12 个 Label 小部件。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        Evas_Object *tbl, *btn, *frame, *o;

        /* Frame */
        frame = elm_frame_add(ad->win);
        elm_object_style_set(frame, "pad_medium");
        elm_object_content_set(ad->conform, frame);
        evas_object_show(frame);

        /* Container: standard table */
        tbl = elm_table_add(ad->win);
        /* Make this table homogeneous for nicer, fixed layout */
        elm_table_homogeneous_set(tbl, EINA_TRUE);
        elm_object_content_set(frame, tbl);
        evas_object_show(tbl);

        {
            /* Button */
            btn = my_button_add(tbl, "Load System Info", btn_clicked_cb, ad);
            my_table_pack(tbl, btn, 0, 0, 2, 1, EVAS_HINT_EXPAND, 0, EVAS_HINT_
FILL, EVAS_HINT_FILL);

            o = my_label_add(tbl, "Back Camera:");
            my_table_pack(tbl, o, 0, 1, 1, 1, EVAS_HINT_EXPAND, 0, 1.0, EVAS_HI
NT_FILL);
```

```
            ad->label1 = my_label_add(tbl, "");
            my_table_pack(tbl, ad->label1, 1, 1, 1, 1, EVAS_HINT_EXPAND, 0, 0.0,
 EVAS_HINT_FILL);

            o = my_label_add(tbl, "Front Camera:");
            my_table_pack(tbl, o, 0, 2, 1, 1, EVAS_HINT_EXPAND, 0, 1.0, EVAS_HI
NT_FILL);

            ad->label2 = my_label_add(tbl, "");
            my_table_pack(tbl, ad->label2, 1, 2, 1, 1, EVAS_HINT_EXPAND, 0, 0.0,
 EVAS_HINT_FILL);

            o = my_label_add(tbl, "Telephony:");
            my_table_pack(tbl, o, 0, 3, 1, 1, EVAS_HINT_EXPAND, 0, 1.0, EVAS_HI
NT_FILL);

            ad->label3 = my_label_add(tbl, "");
            my_table_pack(tbl, ad->label3, 1, 3, 1, 1, EVAS_HINT_EXPAND, 0, 0.0,
 EVAS_HINT_FILL);

            o = my_label_add(tbl, "Screen Width:");
            my_table_pack(tbl, o, 0, 4, 1, 1, EVAS_HINT_EXPAND, 0, 1.0, EVAS_HI
NT_FILL);

            ad->label4 = my_label_add(tbl, "");
            my_table_pack(tbl, ad->label4, 1, 4, 1, 1, EVAS_HINT_EXPAND, 0, 0.0,
 EVAS_HINT_FILL);

            o = my_label_add(tbl, "Screen Height:");
            my_table_pack(tbl, o, 0, 5, 1, 1, EVAS_HINT_EXPAND, 0, 1.0, EVAS_HI
NT_FILL);

            ad->label5 = my_label_add(tbl, "");
            my_table_pack(tbl, ad->label5, 1, 5, 1, 1, EVAS_HINT_EXPAND, 0, 0.0,
 EVAS_HINT_FILL);

            o = my_label_add(tbl, "Platform Version:");
            my_table_pack(tbl, o, 0, 6, 1, 1, EVAS_HINT_EXPAND, 0, 1.0, EVAS_HI
NT_FILL);

            ad->label6 = my_label_add(tbl, "");
            my_table_pack(tbl, ad->label6, 1, 6, 1, 1, EVAS_HINT_EXPAND, 0, 0.0,
 EVAS_HINT_FILL);
        }
    }
```

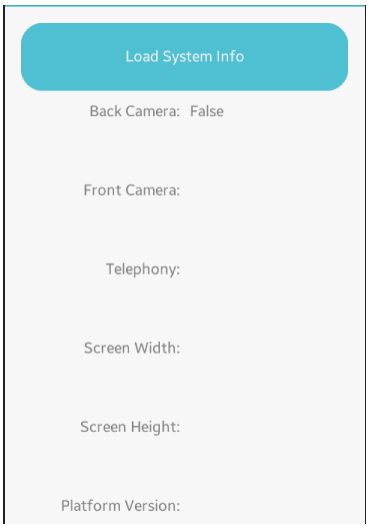
```
    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

我们现在将为这些 Button 创建回调函数。在 create_base_gui() 函数之上添加新代码。

```
static void
btn_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    char buf[100];
    char *sValue = NULL;
    bool bValue = false;
    int nValue = 0;
    int ret;

    ret = system_info_get_platform_bool("http://tizen.org/feature/camera.back",
&bValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        elm_object_text_set(ad->label1, bValue ? "True" : "False");
    }
}
```

system_info_get_platform_bool(char *, bool *) 是一种可请求系统信息的 API。它将会返回布尔格式的数据。第一个参数是键值。传递 “http://tizen.org/feature/camera.back” 将返回是否有后置摄像头。

构建并运行该示例。单击 Button 将更改第一个 Label 中的文本。从模拟器运行它将显示 False；从用户设备运行它将显示 True。



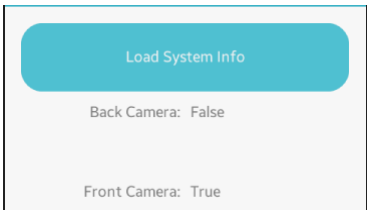
2) 是否有前置摄像头

我们将查明是否有前置摄像头，并在屏幕上显示它。在 btn_clicked_cb() 函数的结尾添加新代码。

```
    ret = system_info_get_platform_bool("http://tizen.org/feature/camera.back",
&bValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        elm_object_text_set(ad->label1, bValue ? "True" : "False");
    }

    ret = system_info_get_platform_bool("http://tizen.org/feature/camera.front",
 &bValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        elm_object_text_set(ad->label2, bValue ? "True" : "False");
    }
}
```

将 “http://tizen.org/feature/camera.front” 传递给第一个参数将返回是否有前置摄像头。

构建并运行该示例。单击 Button；您将在第二个 Label 中看到 True。



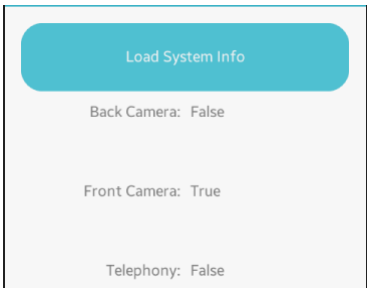
3) 是否有电话功能?

我们将查明是否有电话功能。在 btn_clicked_cb() 函数的结尾添加新代码。

```
    ret = system_info_get_platform_bool("http://tizen.org/feature/camera.front",
 &bValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        elm_object_text_set(ad->label2, bValue ? "True" : "False");
    }

    ret = system_info_get_platform_bool("http://tizen.org/feature/network.telep
hony", &bValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        elm_object_text_set(ad->label3, bValue ? "True" : "False");
    }
}
```

将 “http://tizen.org/feature/network.telephony” 传递给 system_info_get_platform_bool() 函数的第一个参数，将会返回是否具有电话功能。True 值不一定表示您可以使用电话或网络。它只是意味着硬件通信工具已准备就绪。如果没有 USIM 芯片或已在 Preferences 下禁用了网络功能，则无法使用通信功能。

构建并运行该示例。单击 Button；您将在第三个 Label 中看到 True。



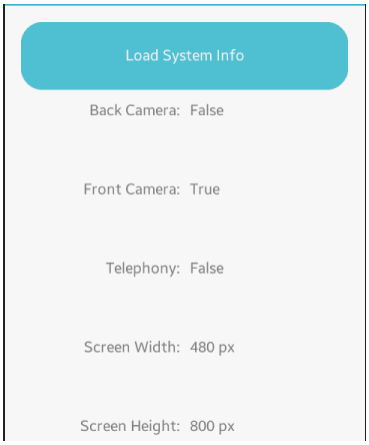
4) 显示器的像素数

现在，我们来请求显示器的像素数。在 btn_clicked_cb() 函数的结尾添加新代码。

```
    ret = system_info_get_platform_bool("http://tizen.org/feature/network.telephony", &bValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        elm_object_text_set(ad->label3, bValue ? "True" : "False");
    }

    ret = system_info_get_platform_int("tizen.org/feature/screen.width", &nValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        sprintf(buf, "%d px", nValue);
        elm_object_text_set(ad->label4, buf);
    }

    ret = system_info_get_platform_int("tizen.org/feature/screen.height", &nValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        sprintf(buf, "%d px", nValue);
        elm_object_text_set(ad->label5, buf);
    }
}
```

system_info_get_platform_int(char *, int *) 是一种可请求系统信息的 API。它将会返回整数格式的数据。第一个参数是键值。传递 “http://tizen.org/feature/screen.width” 将返回水平显示器像素数。传递“tizen.org/feature/screen.height”将返回显示器的垂直像素数。

构建并运行该示例。单击 Button；您将在第四个 Label 和第五个 Label 中看到数字。



5) 平台版本

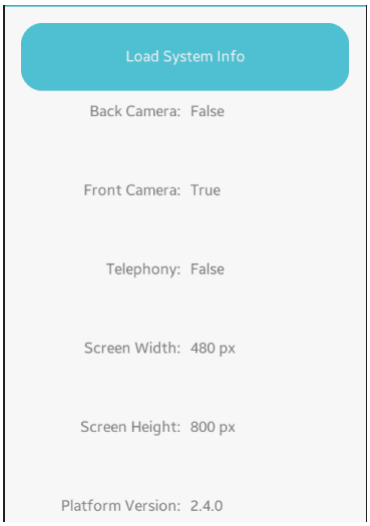
我们将请求平台版本。在 btn_clicked_cb() 函数的结尾添加新代码。

```
    ret = system_info_get_platform_int("tizen.org/feature/screen.height", &nValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        sprintf(buf, "%d px", nValue);
        elm_object_text_set(ad->label5, buf);
    }

    ret = system_info_get_platform_string("http://tizen.org/feature/platform.version", &sValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        elm_object_text_set(ad->label6, sValue);
        free(sValue);
    }
```

```
}
```

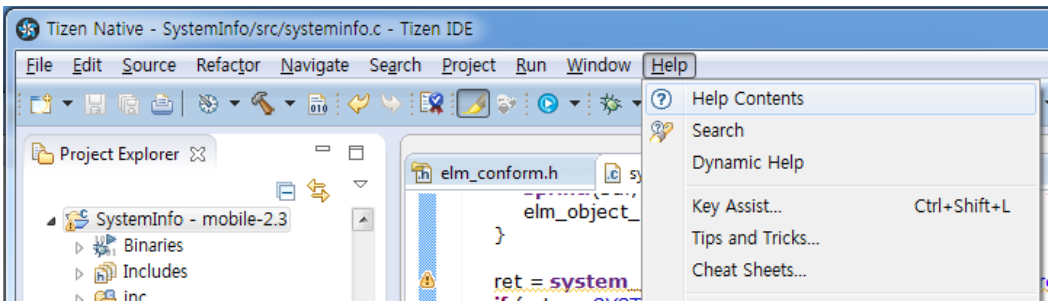
system_info_get_platform_string(char *, char **) 是一种可请求系统信息的 API。它将会返回字符串格式的数据。第一个参数是键值。传递 “http://tizen.org/feature/platform.version” 将返回平台版本。

构建并运行该示例。单击 Button；您将在第六个 Label 中看到平台版本。

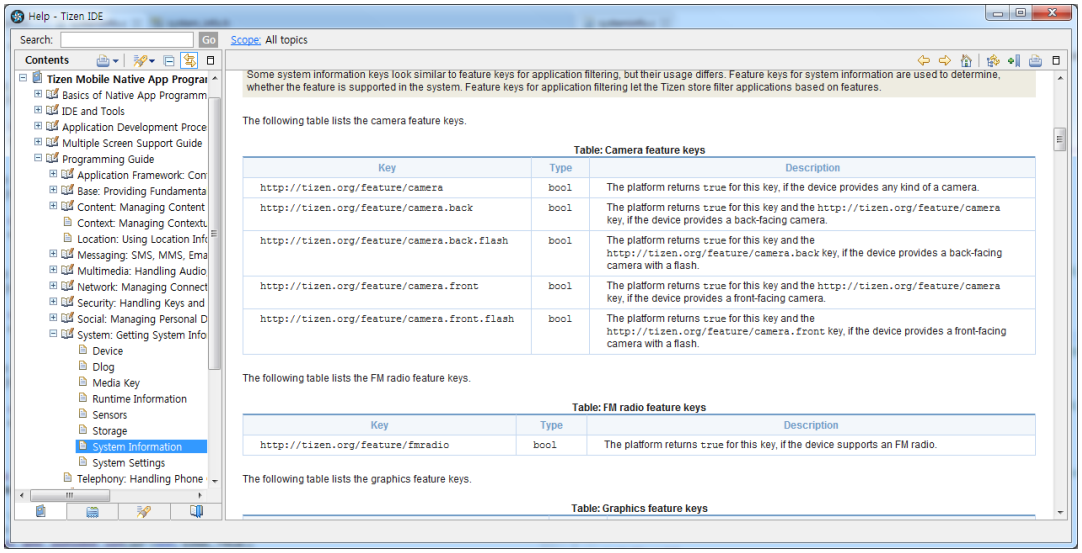


6) 相关 API

我们将在 Help Contents 列表中看到系统信息的类型、键值和返回格式。从主菜单中选择 [Help > Help Contents]。



在 Help Contents 运行时，从左侧的树状列表中选择 [Tizen Mobile Native App Programming > Programming Guide > System > System Information]。您将在屏幕右侧上看到关键值、返回类型和描述。



int system_info_get_platform_bool(const char *key, bool *value)：一种可请求系统信息的 API。它将会返回布尔格式的数据。

int system_info_get_platform_int(const char *key, int *value)：一种可请求系统信息的 API。它将会返回整数格式的数据。

int system_info_get_platform_string(const char *key, char **value)：一种可请求系统信息的 API。它将会返回字符串格式的数据。

43. 系统首选项

多语言支持要求您检查用户的语言设置。收到新消息时，如果正处于静音模式，您将需要用到振动提醒设置。在本例中，我们将了解如何请求有关系统首选项的信息。

1) 请求语言设置

创建新的源项目，将项目名称指定为 SystemSetting。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在源文件的顶端添加变量。

```
#include "systemsetting.h"

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label1;
        Evas_Object *label2;
        Evas_Object *label3;
        Evas_Object *label4;
} appdata_s;
```

我们共公布了 4 个 Label 小部件变量。我们将在第一个 Label 中标记语言设置，在第二个 Label 中标记是否处于静音模式，在第三个 Label 中标记时区，并在第四个 Label 中标记设备名称。

在 create_base_gui() 函数之上创建三个新函数。

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int col, int row, int span
x, int spany,
        bool h_expand, bool v_expand, double h_align, double v_align)
{
    /* Create a frame around the child, for padding */
    Evas_Object *frame = elm_frame_add(table);
    elm_object_style_set(frame, "pad_small");
```

```
    evas_object_size_hint_weight_set(frame, h_expand ? EVAS_HINT_EXPAND : 0, v_e
xpand ? EVAS_HINT_EXPAND : 0);
    evas_object_size_hint_align_set(frame, h_align, v_align);

    /* place child in its box */
    {
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(frame, child);
        evas_object_show(child);
    }

    elm_table_pack(table, frame, col, row, spanx, spany);
    evas_object_show(frame);
}

static Evas_Object *
my_button_add(Evas_Object *parent, const char *text, Evas_Smart_Cb cb, void *cb_
data)
{
    Evas_Object *btn;

    btn = elm_button_add(parent);
    elm_object_text_set(btn, text);
    evas_object_smart_callback_add(btn, "clicked", cb, cb_data);

    return btn;
}

static Evas_Object *
my_label_add(Evas_Object *parent, const char *text)
{
    Evas_Object *btn;

    btn = elm_label_add(parent);
    elm_object_text_set(btn, text);

    return btn;
}
```

my_table_pack() 是将小部件添加到 Table 容器的函数。

my_button_add() 函数可创建 Button 小部件。

my_label_add() 函数可创建 Label 小部件。

将新代码添加到 create_base_gui() 函数中。此代码将创建 1 个 Frame、1 个 Table、1 个 Button 小部件和 8 个 Label 小部件。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        Evas_Object *tbl, *btn, *frame, *o;

        /* Frame */
        frame = elm_frame_add(ad->win);
        elm_object_style_set(frame, "pad_medium");
        elm_object_content_set(ad->conform, frame);
        evas_object_show(frame);

        /* Container: standard table */
        tbl = elm_table_add(ad->win);
        /* Make this table homogeneous for nicer, fixed layout */
        elm_table_homogeneous_set(tbl, EINA_TRUE);
        elm_object_content_set(frame, tbl);
        evas_object_show(tbl);

        {
            /* Button */
            btn = my_button_add(tbl, "Load System Settings", btn_clicked_cb, a
d);
            my_table_pack(tbl, btn, 0, 0, 2, 1, EVAS_HINT_EXPAND, 0, EVAS_HINT_
FILL, EVAS_HINT_FILL);

            /* Fields */
            o = my_label_add(tbl, "Language:");
            my_table_pack(tbl, o, 0, 1, 1, 1, EVAS_HINT_EXPAND, 0, 1.0, EVAS_HI
NT_FILL);

            ad->label1 = my_label_add(tbl, "");
            my_table_pack(tbl, ad->label1, 1, 1, 1, 1, EVAS_HINT_EXPAND, 0, 0.0,
```

```
 EVAS_HINT_FILL);

            o = my_label_add(tbl, "Silent mode:");
            my_table_pack(tbl, o, 0, 2, 1, 1, EVAS_HINT_EXPAND, 0, 1.0, EVAS_HI
NT_FILL);

            ad->label2 = my_label_add(tbl, "");
            my_table_pack(tbl, ad->label2, 1, 2, 1, 1, EVAS_HINT_EXPAND, 0, 0.0,
 EVAS_HINT_FILL);

            o = my_label_add(tbl, "Time zone:");
            my_table_pack(tbl, o, 0, 3, 1, 1, EVAS_HINT_EXPAND, 0, 1.0, EVAS_HI
NT_FILL);

            ad->label3 = my_label_add(tbl, "");
            my_table_pack(tbl, ad->label3, 1, 3, 1, 1, EVAS_HINT_EXPAND, 0, 0.0,
 EVAS_HINT_FILL);

            o = my_label_add(tbl, "Device name:");
            my_table_pack(tbl, o, 0, 4, 1, 1, EVAS_HINT_EXPAND, 0, 1.0, EVAS_HI
NT_FILL);

            ad->label4 = my_label_add(tbl, "");
            my_table_pack(tbl, ad->label4, 1, 4, 1, 1, EVAS_HINT_EXPAND, 0, 0.0,
 EVAS_HINT_FILL);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

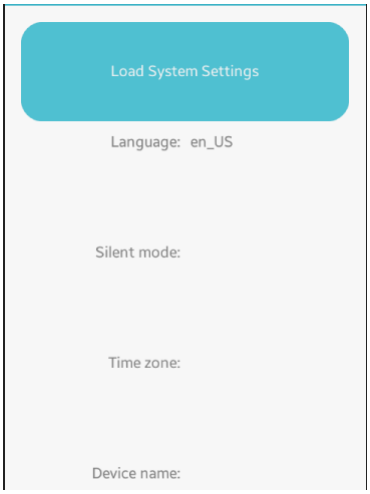
我们现在将为这些 Button 创建回调函数。在 create_base_gui() 函数之上添加新代码。

```
static void
btn_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    char buf[100];
    char *sValue = NULL;
    bool bValue;

    system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_LANGUAGE, &sValu
```

```
e);
    elm_object_text_set(ad->label1, sValue);
    free(sValue);
}
```

system_settings_get_value_string(system_settings_key_e, char **) 是一种用于请求系统首选项的 API。它将会返回字符串格式的数据。第一个参数是键值。传递 SYSTEM_SETTINGS_KEY_LOCALE_LANGUAGE 将返回用户指定的语言类型。

构建并运行该示例。单击 Button；您将在第一个 Label 中看到语言类型。



2) 请求静音模式

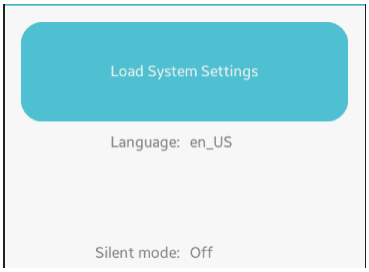
我们将在屏幕上标记它是否处于静音模式。在 btn_clicked_cb() 函数的结尾添加新代码。

```
    system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_LANGUAGE, &sValu
e);
    elm_object_text_set(ad->label1, sValue);
    free(sValue);

    system_settings_get_value_bool(SYSTEM_SETTINGS_KEY_SOUND_SILENT_MODE, &bVal
ue);
```

```
    elm_object_text_set(ad->label2, bValue ? "On" : "Off");
}
```

system_settings_get_value_bool(system_settings_key_e, bool *) 是一种用于请求系统首选项的 API。它将会返回布尔格式的数据。第一个参数是键值。传递 SYSTEM_SETTINGS_KEY_SOUND_SILENT_MODE 将返回是否处于静音模式。

构建并运行该示例。单击 Button；您将在第二个 Label 中看到是否处于静音模式。



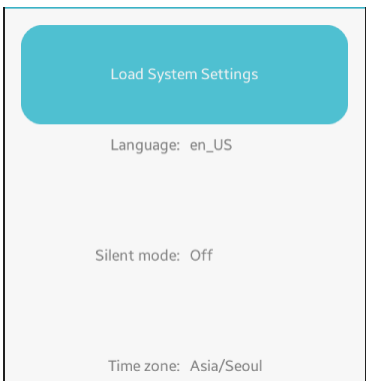
3) 请求时区

这次我们将请求时区。在 btn_clicked_cb() 函数的结尾添加新代码。

```
    system_settings_get_value_bool(SYSTEM_SETTINGS_KEY_SOUND_SILENT_MODE, &bValu
e);
    elm_object_text_set(ad->label2, bValue ? "On" : "Off");

    system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE, &sVal
ue);
    elm_object_text_set(ad->label3, sValue);
    free(sValue);
}
```

将 SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE 传递给 system_settings_get_value_string() 函数的第一个参数将返回字符串格式的时区。

构建并运行该示例。单击 Button；您将在第三个 Label 中看到用户设置的时区。



4) 请求设备名称

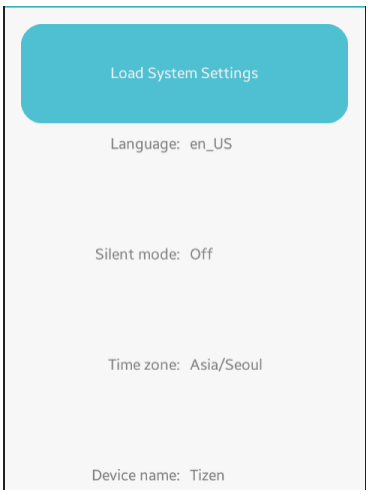
这次我们将请求设备名称。在 btn_clicked_cb() 函数的结尾添加新代码。

```
    system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE, &sValu
e);
```

```
    elm_object_text_set(ad->label3, sValue);
    free(sValue);

    system_settings_get_value_string(SYSTEM_SETTINGS_KEY_DEVICE_NAME, &sValue);
    elm_object_text_set(ad->label4, sValue);
    free(sValue);
}
```

将 SYSTEM_SETTINGS_KEY_DEVICE_NAME 传递给 system_settings_get_value_string() 函数的第一个参数将返回字符串格式的设备名称。

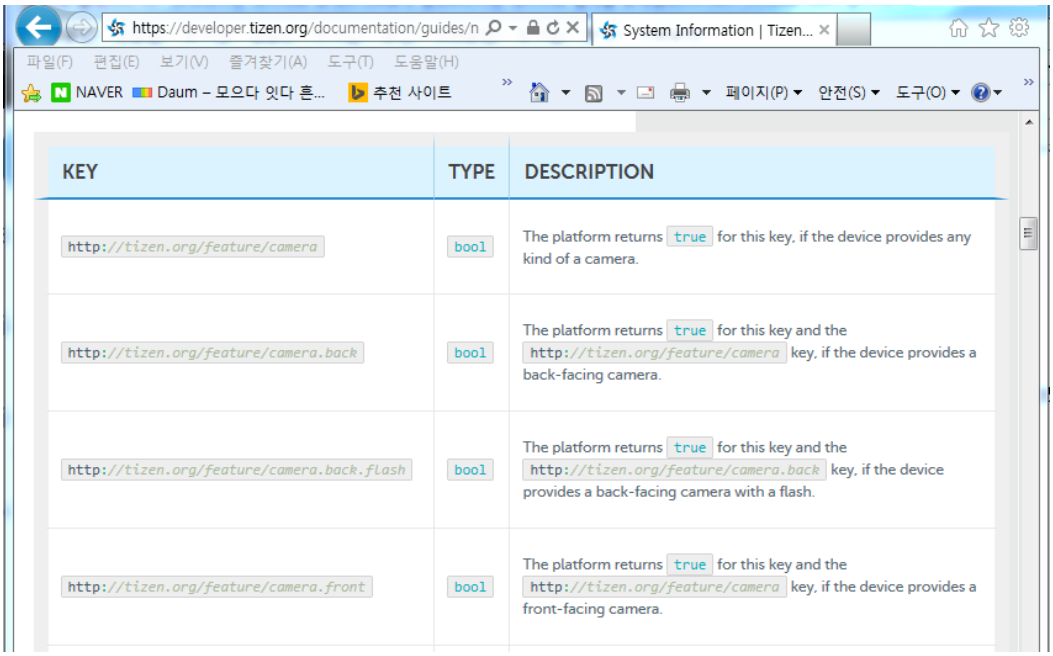
构建并运行该示例。单击 Button；您将在第四个 Label 中看到设备名称。



5) 相关 API

我们将在列表中看到系统首选项的类型、键值和返回格式。要查看用于 System Information 的键值列表，请打开 Web 浏览器，并转至以下地址：

https://developer.tizen.org/documentation/guides/native-application/system/system-information



int system_settings_get_value_int(system_settings_key_e key, int *value)：一种用于请求系统首选项的 API。它将会返回整数格式的数据。

int system_settings_get_value_bool(system_settings_key_e key, bool *value)：一种用于请求系统首选项的 API。它将会返回布尔格式的数据。

int system_settings_get_value_string(system_settings_key_e key, char **value)：一种用于请求系统首选项的 API。它将会返回字符串格式的数据。

44. 电池状态

在构建视频播放器应用程序时，如果电池电量降到 15% 或以下，则必须停止播放视频。否则，可能会错过重要来电。如果电池正在充电，即使剩余电量较低，仍可放心继续播放。我们将了解如何查看当前电池状态，并请求与电池有关的事件。

1) 请求电池状态

创建新的源项目，将项目名称指定为 BatteryInfo。创建源项目之后，打开 src 文件夹中的源文件 (*.c)，并在屏幕的顶部添加库。

```
#include "batteryinfo.h"
#include <device/battery.h>
#include <device/callback.h>
```

device/battery.h 是包含有关电池信息的库头文件。

device/callback.h 是用于与设备有关的事件回调的库头文件。

我们将实施一项功能，以在单击 Button 时于屏幕上显示电池电量和电池是否正在充电。在 create_base_gui() 函数之上创建一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
        double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
```

```
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

随后，它将在 create_base_gui() 函数末尾添加 Box 和 Button 创建代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        { /* child object - indent to how relationship */
            /* Label*/
            ad->label = elm_label_add(ad->win);
            elm_object_text_set(ad->label, "<align=center>Hello Tizen</>");
            /* expand horizontally but not vertically, and fill horiz,
             * align center vertically */
```

```
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.0);

            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "Default style");
            evas_object_smart_callback_add(btn, "clicked", show_battery_state,
ad);
            /* epand both horiz and vert, fill horiz and vert */
            my_box_pack(box, btn, 1.0, 1.0, -1.0, 0.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

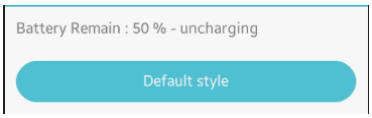
在 create_base_gui() 函数之上添加一个 Button 回调函数。

```
static void
show_battery_state(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        int result=0, percent=0;
        bool charging = false;
        device_battery_get_percent(&percent);
        device_battery_is_charging(&charging);
        char buf[100];
        sprintf(buf, "Battery Remain : %d %% - %s", percent, charging ? "chargin
g" : "uncharging");

        elm_object_text_set(ad->label, buf);
}
```

device_battery_get_percent(int *) 是一种用于返回百分比形式的电池电量的 API。

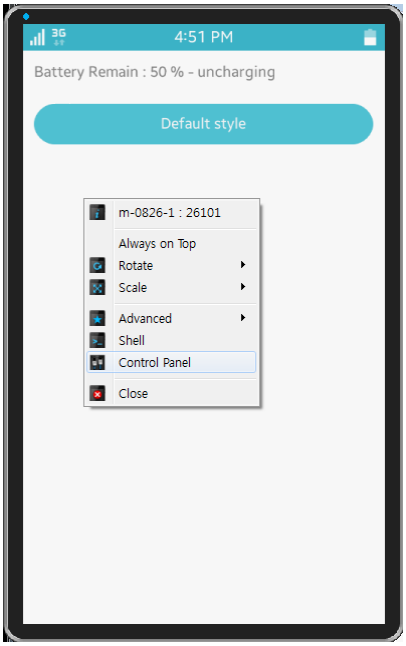
device_battery_is_charging(bool *) 是一种用于返回充电状态的 API。如果正在充电，将会返回 true；否则，将返回 false。

构建并运行该示例。单击 Button；您将在 Label 小部件中看到电池电量和充电状态。如果已从模拟器运行它，则可能会显示“50% remaining, uncharging”。



2）更改模拟器中的电池状态

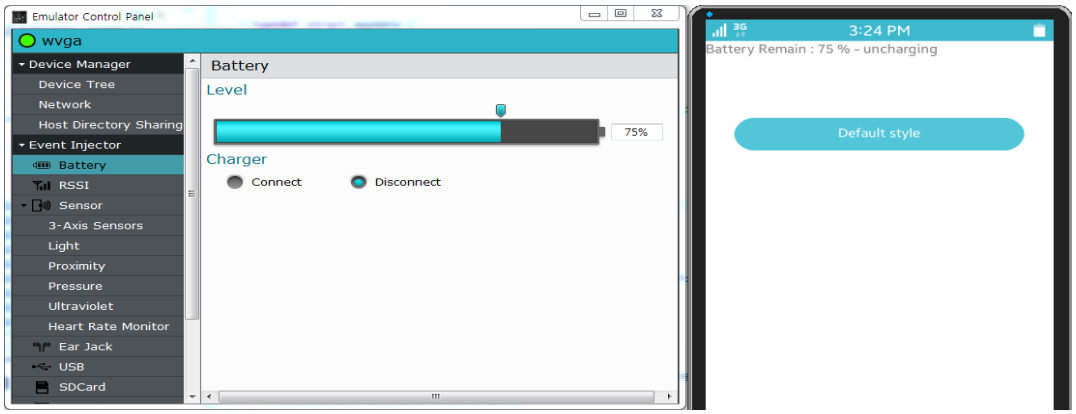
使用 Control Panel 更改模拟器中的电池状态。右键单击模拟器，在快捷菜单中选择 [Control Panel]。



在新的弹出窗口中，从左侧的树状列表中选择 [Event Injector > Battery]。

看到类似电池的滑块时，拖动滚动条以更改值，然后选择 Charger 下的 Connect 单选按钮。

随后，点击模拟器中的“Default style”按钮以显示全新信息。



3) 请求用于更改电池状态的事件

我们来实施一项功能，以便在连接充电线时自动请求事件。在 create_base_gui() 函数的结尾添加新代码。

```
        evas_object_show(ad->win);

        device_add_callback(DEVICE_CALLBACK_BATTERY_CHARGING, battery_charging_cb, ad);
}
```

device_add_callback(device_callback_e, device_changed_cb, void *) 是一种用于指定用于设备的事件回调函数的 API。将 DEVICE_CALLBACK_BATTERY_CHARGING 传递给第一个参数，即可请求用于连接充电器的事件。另一方面，通过传递 DEVICE_CALLBACK_BATTERY_LEVEL，即可请求用于更改电池电量的事件。

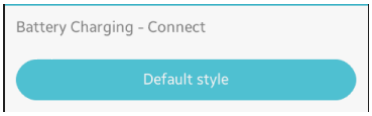
随后，在 create_base_gui() 函数之上创建一个回调函数。

```
static void battery_charging_cb(device_callback_e type, void *value, void *user_data)
{
        appdata_s *ad = user_data;
        char buf[100];
        sprintf(buf, "Battery Charging - %s", (int)value ? "Connect" : "Disconne
```

```
ct");
        elm_object_text_set(ad->label, buf);
}
```

如果已将 1 传递到充电器事件的第二个参数，则代表连接事件；0 代表中断事件。

再次运行该示例，并依次按几下 Control Panel 中 Charger 下的单选按钮。新消息将显示在 Label 小部件中。



4) 请求用于低电池电量的事件

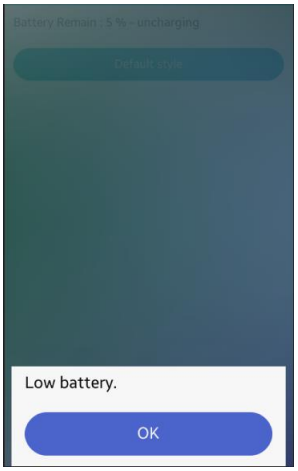
一旦电池电量降到 15% 以下，则必须切换到睡眠模式。源文件下有一个名为 ui_app_low_battery() 的函数。这是一种提醒您电池电量低的事件函数。将新代码添加到 ui_app_low_battery() 函数。

如果此函数不存在，或若要更改其名称，您可在 main() 函数中执行此操作。

```
static void
ui_app_low_battery(app_event_info_h event_info, void *user_data)
{
    show_battery_state(user_data, NULL, NULL);
}

int
main(int argc, char *argv[])
{
    appdata_s ad = {0,};
    int ret = 0;

    ui_app_lifecycle_callback_s event_callback = {0,};
    app_event_handler_h handlers[5] = {NULL, };

    event_callback.create = app_create;
    event_callback.terminate = app_terminate;
    event_callback.pause = app_pause;
    event_callback.resume = app_resume;
    event_callback.app_control = app_control;

    ui_app_add_event_handler(&handlers[APP_EVENT_LOW_BATTERY], APP_EVENT_LOW_BATTERY, ui_app_low_battery, &ad);
    ui_app_add_event_handler(&handlers[APP_EVENT_LANGUAGE_CHANGED], APP_EVENT_LANGUAGE_CHANGED, ui_app_lang_changed, &ad);

    ret = ui_app_main(argc, argv, &event_callback, &ad);
    if (ret != APP_ERROR_NONE) {
    dlog_print(DLOG_ERROR, LOG_TAG, "app_main() is failed. err = %d", ret);
    }

    return ret;
}
```

电池电量过低时，屏幕上会自动显示当前电池状态。

再次运行该示例，并检查 Control Panel 中的 Disconnect 框，然后将电池电量更改到低于 15%。警告弹出窗口将会出现，并且 Label 小部件中的消息将会更改。



5）相关 API

device/battery.h：包含有关电池信息的一种库头文件。

device/callback.h：用于与设备有关的事件回调的库头文件。

device_battery_get_percent(int *)：一种用于返回百分比形式的电池电量的 API。

int device_battery_get_percent(int *percent)：一种用于返回充电状态的 API。如果正在充电，将会返回 true；否则，将返回 false。

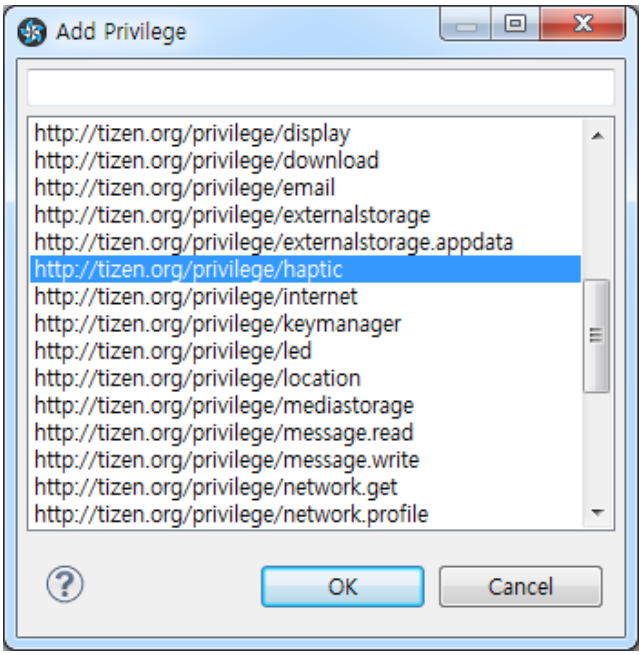
int device_add_callback(device_callback_e type, device_changed_cb callback, void *user_data)：一种用于指定用于设备的事件回调函数的 API。将 DEVICE_CALLBACK_BATTERY_CHARGING 传递给第一个参数，即可请求用于连接充电器的事件。另一方面，通过传递 DEVICE_CALLBACK_BATTERY_LEVEL，即可请求用于更改电池电量的事件。

45. 产生振动

如在静音模式下收到消息，应使用振动提示功能。振动提示有助于用户在专注于游戏的同时留意新消息。如将振动提示与 3D 图形相结合，将能得到 4D 体验。在本例中，我们将了解如何产生振动。

1）注册权限

创建新的源项目，将项目名称指定为 VibrateEx。您需要具备用户权限才能使用振动功能。创建源项目之后，打开 tizen-manifest.xml 文件，在以下选项卡按钮中，单击 Privileges。然后，点击右上角的 Add 按钮。在弹出窗口上，从列表中选择 http://tizen.org/privilege/haptic，然后点击 OK 按钮以关闭窗口。



保存之后，点击底部选项卡按钮最右角的 tizen-manifest.xml 按钮，就能看到 xml 文件的源代码。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.ex
ample.vibrateex" version="1.0.0">
        <profile name="mobile"/>
        <ui-application appid="org.example.vibrateex" exec="vibrateex" multiple=
"false" nodisplay="false" taskmanage="true" type="capp">
                <label>vibrateex</label>
                <icon>vibrateex.png</icon>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/haptic</privilege>
        </privileges>
</manifest>
```

2) 请求 Haptic 设备的数量

Haptic 是一种触摸驱动的界面。我们将请求计数，以找出您手机上安装了多少台 Haptic 设备。打开 src 文件夹中的源文件 (~.c)，并在顶部添加库头文件和变量。

```
#include "vibrateex.h"
#include <device/haptic.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        haptic_device_h handle;
        haptic_effect_h effect_handle;

        Ecore_Timer *timer1;
        int timer_count;
} appdata_s;
```

device/haptic.h 是一种用于控制 Haptic 设备的库。

haptic_device_h 是一种能够控制 Haptic 设备的句柄。

haptic_effect_h 是一种能够控制其中一种 Haptic 效果的句柄。

在 create_base_gui() 函数的结尾添加新代码。

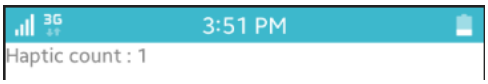
```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        haptic_count(ad);
}
```

haptic_count() 函数可显示 Haptic 设备的数量。在 create_base_gui() 函数之上创建此函数。

```
static void
haptic_count(appdata_s *ad)
{
        int error, num;
        error = device_haptic_get_count(&num);

        char buf[100];
        sprintf(buf, "Haptic count : %d", num);
        elm_object_text_set(ad->label, buf);
}
```

device_haptic_get_count(int *device_number) 是一种用于返回 Haptic 设备数量的 API。如果小于 1，则不支持振动功能。

构建示例，并安装到智能手机上。您无法在模拟器中测试振动功能。一旦执行示例，便会在 Label 小部件中显示 Haptic 设备的数量。



2) 产生振动功能

创建 Haptic 对象以产生振动。在 create_base_gui() 函数之上创建两个新函数。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}

static Evas_Object *
my_button_add(Evas_Object *parent, const char *text, Evas_Smart_Cb cb, void *cb_
data)
{
    Evas_Object *btn;

    btn = elm_button_add(parent);
    elm_object_text_set(btn, text);
    evas_object_smart_callback_add(btn, "clicked", cb, cb_data);
```

```
    return btn;
}
```

my_box_pack() 函数将向 Box 容器添加一个小部件。

my_button_add() 函数可创建 Button 小部件。

向 create_base_gui() 函数添加新代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    Evas_Object *btn, *box;

    /* Container: standard table */
    box = elm_box_add(ad->win);
    elm_box_homogeneous_set(box, EINA_TRUE);
    elm_box_horizontal_set(box, EINA_FALSE);
    evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    evas_object_size_hint_align_set(box, EVAS_HINT_FILL, EVAS_HINT_FILL);
    elm_object_content_set(ad->conform, box);
    evas_object_show(box);

    {
        /* Label*/
        ad->label = elm_label_add(box);
        my_box_pack(box, ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, 0.5, 0.
5);

        /* Buttons */
        btn = my_button_add(box, "Vibrate", btn_vibrate_cb, ad);
        my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVAS_HINT_FIL
L, EVAS_HINT_FILL);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

```
    /* Haptic */
    haptic_count(ad);
    device_haptic_open(0, &ad->handle);
}
```

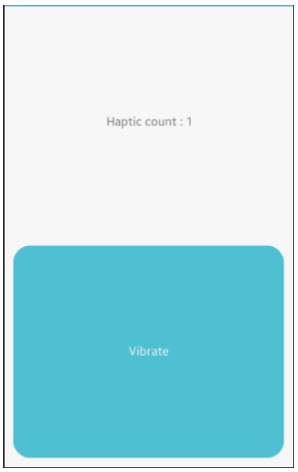
device_haptic_open(int, haptic_device_h *) 是一种用于创建 Haptic 对象的 API。将 Haptic 设备数量传递给第一个参数，则会让第二个参数返回 Haptic 对象。

我们将实施一项功能，以便在您点击 Button 时产生 5 秒的振动。在 create_base_gui() 函数之上创建一个 Button 回调函数。

```
static void
btn_vibrate_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        int error = device_haptic_vibrate(ad->handle, 5000, 100, &ad->effect_han
dle);
}
```

device_haptic_vibrate(haptic_device_h, int, int, haptic_effect_h *) 是一种可产生振动的 API。第一个参数可返回 Haptic 对象；第二个参数可返回持续时间（以微秒为单位）；第三个参数可返回强度 (0~100)；第四个参数可返回 Haptic 效应控制句柄。它用于强行停止振动。

再次运行此示例并点击 Button。振动会持续 5 秒钟。



3) 停止振动功能

我们将实施一项功能，以便通过添加第二个 Button 来强制停止振动功能。将 Button 创建代码添加到 create_base_gui() 函数。

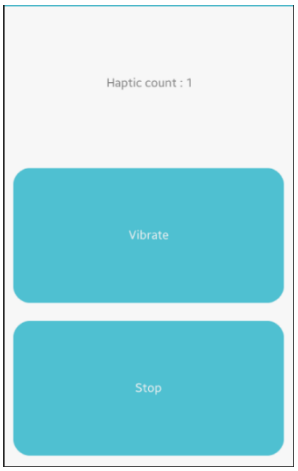
```
        /* Buttons */
        btn = my_button_add(box, "Vibrate", btn_vibrate_cb, ad);
        my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVAS_HINT_FILL,
 EVAS_HINT_FILL);

        btn = my_button_add(box, "Stop", btn_stop_cb, ad);
        my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVAS_HINT_FIL
L, EVAS_HINT_FILL);
    }
```

随后，在 create_base_gui() 函数之上创建一个 Button 回调函数。

```
static void
btn_stop_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        int error = device_haptic_stop(ad->handle, &ad->effect_handle);
}
```

device_haptic_stop(haptic_device_h, haptic_effect_h) 是一种用于停止 Haptic 功能的 API。它会将 Haptic 对象传递给第一个参数，并将效应控制句柄传递给第二个参数。

再次运行该示例，然后点击第一个 Button。一旦振动开始，便点击第二个 Button。它将停止振动。



4) 动态振动

我们将实施一项功能，以通过计时器打开/关闭振动功能。将第三个 Button 创建代码添加到 create_base_gui() 函数。

```
        btn = my_button_add(box, "Stop", btn_stop_cb, ad);
        my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVAS_HINT_FILL,
EVAS_HINT_FILL);

        btn = my_button_add(box, "Dynamic Vibrate", btn_dynamic_cb, ad);
        my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVAS_HINT_FIL
L, EVAS_HINT_FILL);
    }
```

最后，在 create_base_gui() 函数之上添加三个新函数。

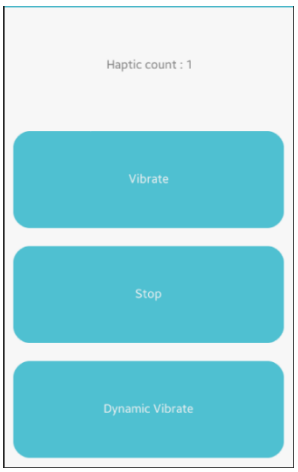
```
static void
dynamic_vibrate(appdata_s *ad)
{
    if( ad->effect_handle != NULL )
        device_haptic_stop(ad->handle, &ad->effect_handle);

    if( (ad->timer_count % 2) == 0 )
        device_haptic_vibrate(ad->handle, 500, 100, &ad->effect_handle);
}

static Eina_Bool
timer1_cb(void *data EINA_UNUSED)
{
    appdata_s *ad = data;
    ad->timer_count ++;
    dynamic_vibrate(ad);

    if(ad->timer_count > 5)
    {
        ecore_timer_del(ad->timer1);
        ad->timer1 = NULL;
    }
    return ECORE_CALLBACK_RENEW;
}

static void
btn_dynamic_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    ad->timer_count = 0;
    if (ad->timer1)
        ecore_timer_del(ad->timer1);
    ad->timer1 = ecore_timer_add(0.5, timer1_cb, ad);
    ad->effect_handle = NULL;

    dynamic_vibrate(ad);
}
```

系统将每 0.5 秒调用一次 dynamic_vibrate() 函数，以打开/关闭振动功能。

timer1_cb() 是一种每 0.5 秒调用一次的计时器事件函数。单击第三个 Button 将调用此函数 6 次。

btn_dynamic_cb() 函数可重置全局变量，并启动计时器。

再次运行该示例，然后点击第三个 Button。每 0.5 秒便会打开/关闭振动功能。



5) 相关 API

device_haptic_get_count(int *device_number)：一种用于返回 Haptic 设备数量的 API。如果小于 1，则不支持振动功能。

int device_haptic_open(int device_index, haptic_device_h *device_handle)：一种用于创建 Haptic 对象的 API。将 Haptic 设备数量传递给第一个参数，则会让第二个参数返回 Haptic 对象。

int device_haptic_vibrate(haptic_device_h device_handle, int duration, int feedback, haptic_effect_h *effect_handle)：一种用于产生振动的 API。第一个参数可返回 Haptic 对象；第二个参数可返回持续时间（以微秒为单位）；第三个参数可返回强度 (0~100)；第四个参数可返回 Haptic 效应控制句柄。它用于强行停止振动。

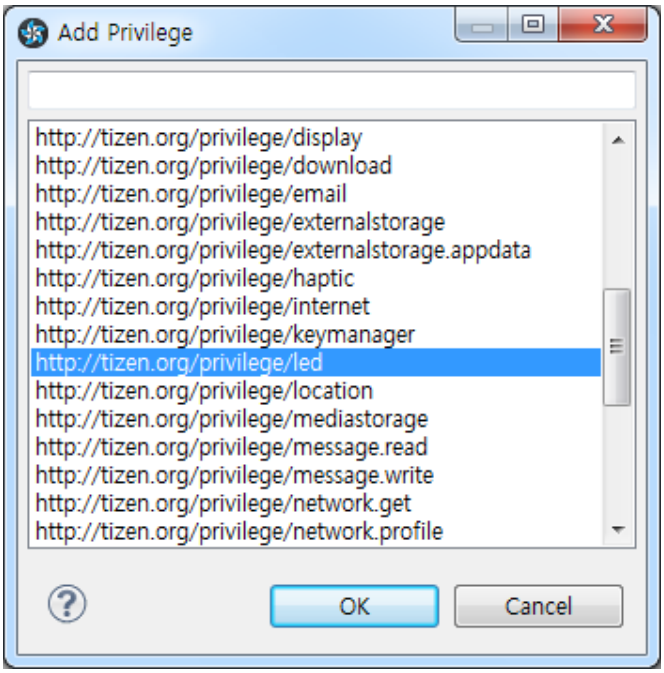
int device_haptic_stop(haptic_device_h device_handle, haptic_effect_h effect_handle)：一种用于停止 Haptic 功能的 API。它会将 Haptic 对象传递给第一个参数，并将效应控制句柄传递给第二个参数。

46. LED 闪光灯背光

在本例中，我们将了解如何打开/关闭用于摄像头闪光灯的 LED 背光。

1）注册权限

创建新的源项目，将项目名称指定为 LedFreshEx。您需要具备用户权限才能控制背光。创建源项目之后，打开 tizen-manifest.xml 文件，在以下选项卡按钮中，单击 Privileges。然后，点击右上角的 Add 按钮。在弹出窗口上，从列表中选择 http://tizen.org/privilege/led，然后点击 OK 按钮以关闭窗口。



保存之后，点击底部选项卡按钮最右角的 tizen-manifest.xml 按钮，就能看到 xml 文件的源代码。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.ledfreshex" version="1.0.0">
        <profile name="mobile"/>
```

```
        <ui-application appid="org.example.ledfreshex" exec="ledfreshex" multipl
e="false" nodisplay="false" taskmanage="true" type="capp">
                <label>ledfreshex</label>
                <icon>ledfreshex.png</icon>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/led</privilege>
        </privileges>
</manifest>
```

2) 请求 LED 的最大亮度

应在 LED 亮起时指定亮度。我们将请求最大亮度。打开 src 文件夹中的源文件 (~.c)，并在顶部添加库头文件和变量。

```
#include "ledfresh.h"
#include <device/led.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        int max;
} appdata_s;
```

device/led.h 是一种 LED 控制库。

max 是一种保存最大亮度值的变量。

在 create_base_gui() 函数的结尾添加新代码。

```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        get_max_brightness(ad);
}
```

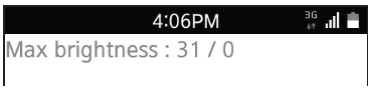
get_max_brightness 函数可请求 LED 背光的最大亮度。在 create_base_gui() 函数之上创建此函数。

```
static void
get_max_brightness(appdata_s *ad)
{
        int error = device_flash_get_max_brightness(&ad->max);
        int val = 0;
        error = device_flash_get_brightness(&val);

        char buf[PATH_MAX];
        sprintf(buf, "Max brightness : %d / %d", ad->max, val);
        elm_object_text_set(ad->label, buf);
}
```

device_flash_get_max_brightness(int *) 是一种用于请求 LED 背光最大亮度的 API。

device_flash_get_brightness(int *) 是一种用于请求 LED 背光当前亮度的 API。

运行示例。最大亮度和当前亮度都将显示在 Label 小部件中。



3) 打开/关闭 LED

我们将实施一项功能，以便通过添加 2 个 Button 来实施打开/关闭 LED 背光的功能。在 create_base_gui() 函数之上创建两个新函数。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
        double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
```

```
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}

static Evas_Object *
my_button_add(Evas_Object *parent, const char *text, Evas_Smart_Cb cb, void *cb_
data)
{
    Evas_Object *btn;

    btn = elm_button_add(parent);
    elm_object_text_set(btn, text);
    evas_object_smart_callback_add(btn, "clicked", cb, cb_data);

    return btn;
}
```

将小部件创建代码添加到 create_base_gui() 函数。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
```

```
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    Evas_Object *btn, *box;

    /* Container: standard table */
    box = elm_box_add(ad->win);
    elm_box_homogeneous_set(box, EINA_TRUE);
    elm_box_horizontal_set(box, EINA_FALSE);
    evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    evas_object_size_hint_align_set(box, EVAS_HINT_FILL, EVAS_HINT_FILL);
    elm_object_content_set(ad->conform, box);
    evas_object_show(box);

    {
        /* Label*/
        ad->label = elm_label_add(ad->conform);
        my_box_pack(box, ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, 0.5, 0.
5);

        /* Button-1 */
        btn = my_button_add(box, "LED On", btn_led_on_cb, ad);
        my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVAS_HINT_FIL
L, EVAS_HINT_FILL);

        /* Button-2 */
        btn = my_button_add(box, "LED Off", btn_led_off_cb, ad);
        my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVAS_HINT_FIL
L, EVAS_HINT_FILL);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

随后，在 create_base_gui() 函数之上创建一个 Button 回调函数。

```
static void
btn_led_on_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        device_flash_set_brightness(ad->max);
        device_led_play_custom(1000, 500, 0xFFFFFF00, LED_CUSTOM_DEFAULT);
```

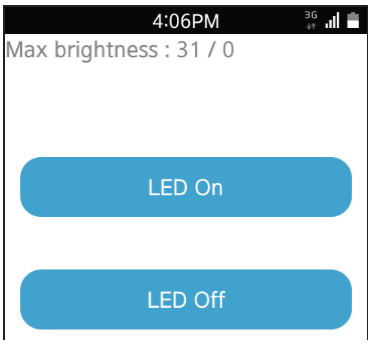
```
}

static void
btn_led_off_cb(void *data, Evas_Object *obj, void *event_info)
{
        device_led_stop_custom();
}
```

device_flash_set_brightness(int) 是一种用于指定 LED 亮度的 API。

device_led_play_custom(on, off, color, int) 是一种用于启动 LED On 的 API。

device_led_stop_custom(void) 是一种用于 LED Off 的 API。

再次运行该示例。“LED On”按钮将打开 LED；“LED Off”按钮会将其关闭。



4) LED Off 故障排除

视型号而定，LED 无法通过 device_led_stop_custom() 函数来关闭。在这种情况下，只需将亮度尽量降低至最低级别即可。如下所示修改 btn_led_off_cb() 函数。

```
static void
btn_led_off_cb(void *data, Evas_Object *obj, void *event_info)
{
        device_flash_set_brightness(0);
        device_led_stop_custom();
}
```

您可以使用 device_flash_set_brightness() 函数将亮度设置为 0，以将其强制关闭。

5）相关 API

device/led.h：一种 LED 控制库。

device_flash_get_max_brightness(int *)：一种用于请求 LED 背光最大亮度的 API。

device_flash_get_max_brightness(int *)：一种用于请求 LED 背光当前亮度的 API。

int device_flash_set_brightness(int brightness)：一种用于指定 LED 亮度的 API。

int device_led_play_custom(int on, int off, unsigned int color, unsigned int flags)：一种用于启动 LED On 的 API。

int device_led_stop_custom(void)：一种 LED Off API。

47. 用于旋转屏幕方向的事件

Orientation 指的是手机的旋转方向。Portrait 表示垂直；Landscape 表示水平。我们将了解如何检查手机的当前方向，并请求 Orientation 事件。

1）请求方向

创建新的源项目，将项目名称指定为 OrientationEvent。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在 create_base_gui() 函数末尾添加新代码。

```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        // Show now orientation
        show_orientation(ad, NULL, NULL);
}
```

show_orientation() 函数会请求当前屏幕方向，然后会将其输出到 Label 小部件。现在开始吧！在 create_base_gui() 函数之上创建一个新函数。

```
// Show now orientation
static void
show_orientation(appdata_s *ad, Evas_Object *obj, void *event_info)
{
        // Get orientation
        int result = elm_win_rotation_get(ad->win);

        switch( result )
        {
        case APP_DEVICE_ORIENTATION_0 :
                elm_object_text_set(ad->label, "Portrait-1");
                break;
        case       APP_DEVICE_ORIENTATION_90 :
                elm_object_text_set(ad->label, "Landscape-1");
                break;
        case APP_DEVICE_ORIENTATION_180 :
```

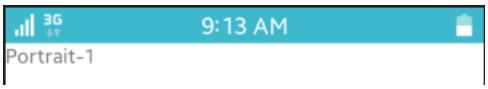
```
                elm_object_text_set(ad->label, "Portrait-2");
                break;
        case APP_DEVICE_ORIENTATION_270 :
                elm_object_text_set(ad->label, "Landscape-2");
                break;
        default :
                elm_object_text_set(ad->label, "Other Event");
                break;
        }
}
```

show_orientation() 函数会请求 Orientation，然后将其输出到 Label 小部件。

elm_win_rotation_get(const Evas_Object *) 是一种用于返回当前 Orientation 的 API。返回的类型如下：

- APP_DEVICE_ORIENTATION_0：Portrait First
- APP_DEVICE_ORIENTATION_90：Landscape First
- APP_DEVICE_ORIENTATION_180：Portrait Second
- APP_DEVICE_ORIENTATION_270：Landscape Second

构建并运行该示例。您将在 Label 小部件中看到“Portrait-1”文本。



2) 在模拟器中旋转屏幕

我们将实施一项功能，以便在单击 Button 时输出方向。在 create_base_gui() 函数之上创建一个新函数。

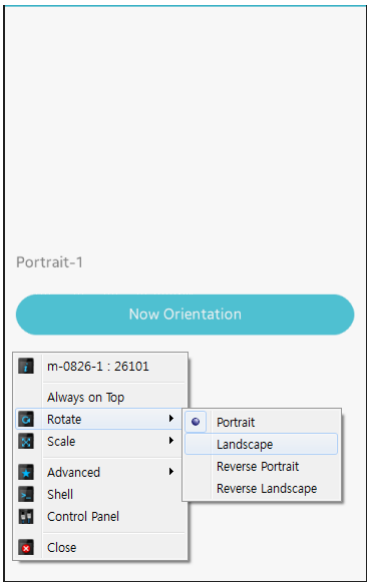
```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
                      double h_weight, double v_weight, double h_align, double
 v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

随后，将 Button 创建代码添加到 create_base_gui() 函数。

```
        /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
```

```
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        Evas_Object * box, *btn;

        /* A box to put things in verticallly - default mode for box */
        box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        { /* child object - indent to how relationship */
            /* Label*/
            ad->label = elm_label_add(ad->win);
            elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
            //evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVA
S_HINT_EXPAND);
            //elm_object_content_set(ad->conform, ad->label);
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.0);

            /* Button-1 */
            btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "Now Orientation");
            evas_object_smart_callback_add(btn, "clicked", show_orientation, (v
oid *)ad);
            my_box_pack(box, btn, 1.0, 0.0, -1.0, -1.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

我们将旋转模拟器。右键单击模拟器，在快捷菜单中选择 [Rotate > Landscape]。



在屏幕方向旋转时，点击 Button。这次您将看到“Landscape-2”文本。

将它更改回 Portrait，并点击 Button 以显示 Portrait-1。

3) 旋转 Orientation

我们将实施一项功能，以便在您点击 Button 时更改 Orientation。将第二个 Button 创建代码添加到 create_base_gui() 函数。

```
            /* Button-1 */
            btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "Now Orientation");
            evas_object_smart_callback_add(btn, "clicked", show_orientation, (vo
id *)ad);
            my_box_pack(box, btn, 1.0, 0.0, -1.0, -1.0);

            /* Button-2 */
            btn = elm_button_add(ad->win);
            elm_object_text_set(btn, "Orientation Change");
            evas_object_smart_callback_add(btn, "clicked", btn_orientation_chan
ge_cb, (void *)ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, 0.0);
        }
    }
```

然后，在 create_base_gui() 函数之上为新创建的 Button 创建一个回调函数。

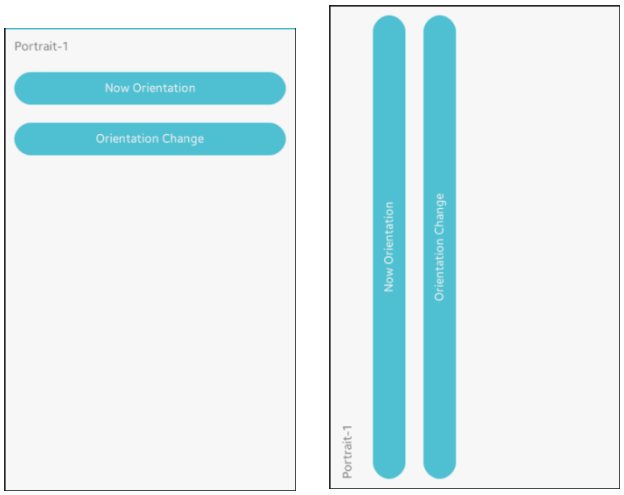
```
// 'Orientation Change' Button event function
static void
btn_orientation_change_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = (appdata_s*)data;
        // Get orientation
        int result = elm_win_rotation_get(ad->win);

        if( result == APP_DEVICE_ORIENTATION_0 || result == APP_DEVICE_ORIENTATI
ON_180 )
                elm_win_rotation_with_resize_set(ad->win, APP_DEVICE_ORIENTATION
_90);
        else
                elm_win_rotation_with_resize_set(ad->win, APP_DEVICE_ORIENTATION
_0);
}
```

此代码可识别当前 Orientation，并根据需要将其从 Portrait 更改成 Landscape，反之亦然。

elm_win_rotation_with_resize_set(Evas_Object *, int) 是一种用于更改 Orientation 的 API。该选项的类型如下：

- APP_DEVICE_ORIENTATION_0：Portrait First
- APP_DEVICE_ORIENTATION_90：Landscape First
- APP_DEVICE_ORIENTATION_180：Portrait Second
- APP_DEVICE_ORIENTATION_270：Landscape Second

再次运行该示例，然后点击第二个 Button。每次点击 Button 时，将依次显示 Landscape 模式和 Portrait 模式。



4) 请求用于更改方向的事件

我们来请求 Orientation 更改时发生的事件。在 create_base_gui() 函数的结尾添加新代码。

```
        // Show now orientation
        show_orientation(ad, NULL, NULL);

        // Set callback function of orientation change event
        evas_object_smart_callback_add(ad->win, "rotation,changed", win_rotatio
n_changed_cb, ad);
}
```

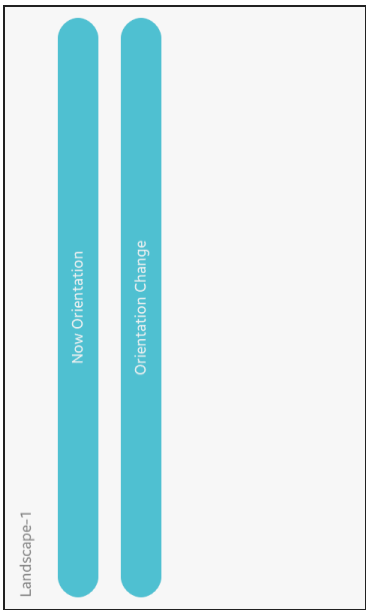
evas_object_smart_callback_add(Evas_Object *, char *, Evas_Smart_Cb, void *) 是一种 API，用于指定诸如 Layout 容器或 Button 小部件等智能对象的事件回调函数。您可以通过将 Win 传递给第一个参数，并为第二个参数指定“rotation, changed”，从而请求用于更改 Orientatioin 的事件。

我们已准备好创建用于更改 Orientation 的事件函数。在 create_base_gui() 函数之上创建一个新函数。

```
// Orientation changed event function
static void
win_rotation_changed_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = (appdata_s*)data;
        show_orientation(ad, NULL, NULL);
}
```

如果 Orientation 更改，则会调用上述函数。请求 Orientation 信息以便显示在屏幕上。

再次运行该示例，然后点击第二个 Button。如果屏幕方向更改，则 Landscape-1 会自动出现在 Label 小部件中。



5）固定方向

我们将锁定 Orientation，以便在您旋转手机时，方向保持不变。可允许的 Orientation 类型已在 create_base_gui() 函数的开始处定义。

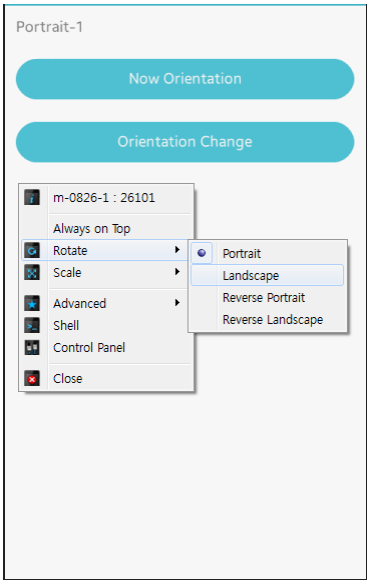
```
    int rots[4] = { 0, 90, 180, 270 };
    elm_win_wm_rotation_available_rotations_set(ad->win, (const int *)(&rots),
4);
```

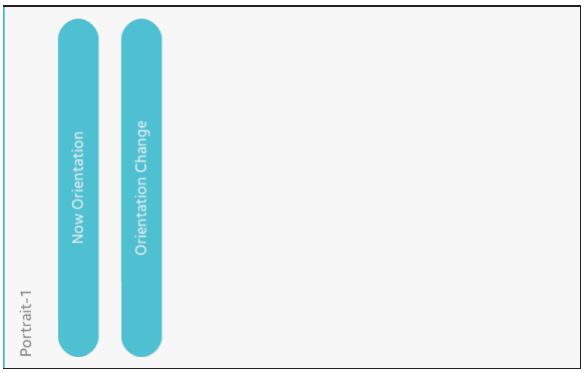
如下所示进行更改。

```
    int rots[1] = { 0 };
    elm_win_wm_rotation_available_rotations_set(ad->win, (const int *)(&rots),
1);
```

elm_win_wm_rotation_available_rotations_set(Evas_Object *, int *, unsigned int) 是一种用于指定 Orientation 所允许类型的 API。它会将保持角度的数组传递给第二个参数，并将数组中保存的数据计数传递给第三个参数。

再次运行该示例，右键单击并在快捷菜单中选择 [Rotate > Landscape]。模拟器旋转时，Orientation 模式将保持不变。





6) 相关 API

int elm_win_rotation_get(const Evas_Object *obj)：一种用于返回当前 Orientation 的 API。返回的类型如下：

- APP_DEVICE_ORIENTATION_0：Portrait First
- APP_DEVICE_ORIENTATION_90：Landscape First
- APP_DEVICE_ORIENTATION_180：Portrait Second
- APP_DEVICE_ORIENTATION_270：Landscape Second

void elm_win_rotation_with_resize_set(Evas_Object *, int)：一种用于更改 Orientation 的 API。该选项的类型如下：

- APP_DEVICE_ORIENTATION_0：Portrait First
- APP_DEVICE_ORIENTATION_90：Landscape First

- APP_DEVICE_ORIENTATION_180：Portrait Second
- APP_DEVICE_ORIENTATION_270：Landscape Second

void evas_object_smart_callback_add(Evas_Object *obj, const char *event, Evas_Smart_Cb func, const void *data)：一种用于指定诸如 Layout 容器或 Button 小部件等智能对象的事件回调函数的 API。您可以通过将 Win 传递给第一个参数，并为第二个参数指定“rotation, changed”，从而请求用于更改 Orientatioin 的事件。

void elm_win_wm_rotation_available_rotations_set(Evas_Object *obj, const int *rotations, unsigned int count)：一种用于指定 Orientation 所允许类型的 API。它会将保持角度的数组传递给第二个参数，并将数组中保存的数据计数传递给第三个参数。

48. 硬件键事件和调试模式

在构建音频/视频播放器或游戏/仪器应用程序时，可使用上/下键调节音量。此外，您还能够执行特定操作（如按硬件 Menu 菜单时显示菜单）。一般情况下，按硬件 Back 按钮可让您回到前一屏幕或退出应用程序。但要注意的是，您经常需要执行其它操作（如显示警告弹出窗口）。在本例中，我们将了解如何请求硬件键事件。

1) 请求硬件键事件

创建新的源项目，将项目名称指定为 HardwareKeyEvent。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在 create_base_gui() 函数末尾添加新代码。

```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        /* Hardware key event callback */
        evas_object_event_callback_add(ad->win, EVAS_CALLBACK_KEY_DOWN, on_keyd
own_cb, ad);
}
```

EVAS_CALLBACK_KEY_DOWN 代表硬件键按下事件的回调函数。

我们现在将创建一个回调函数。在 create_base_gui() 函数之上添加新代码。

```
static void
on_keydown_cb(void *data, Evas *evas, Evas_Object *o, void *event_info)
{
        appdata_s* ad = data;
        Evas_Event_Key_Down *ev = event_info;

        char *old_msg = elm_object_text_get(ad->label);
        char total_msg[PATH_MAX];
        char *key_value = strdup( ev->keyname );
```

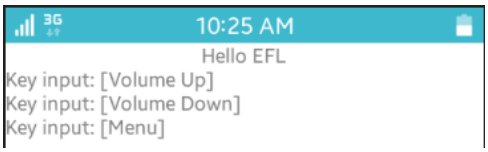
```
        if( strcmp(ev->keyname, "XF86Menu") ==0 ) {
                key_value = "Menu";
        }
        else if( strcmp(ev->keyname, "XF86Home") ==0 ) {
                key_value = "Home";
        }
        else if( strcmp(ev->keyname, "XF86Back") ==0 ) {
                key_value = "Back";
        }
        else if( strcmp(ev->keyname, "XF86PowerOff") ==0 ) {
                key_value = "Power";
        }
        else if( strcmp(ev->keyname, "XF86AudioRaiseVolume") ==0 ) {
                key_value = "Volume Up";
        }
        else if( strcmp(ev->keyname, "XF86AudioLowerVolume") ==0 ) {
                key_value = "Volume Down";
        }
        else {
                key_value = ev->keyname;
        }

        sprintf(total_msg, "%s<br/>Key input: [%s]", old_msg, key_value);
        elm_object_text_set(ad->label, total_msg);
}
```

on_keydown_cb() 是一种用于硬件按键的事件函数。第一个参数可接收用户数据；第二个参数可接收事件发生的对象；第三个参数可接收事件信息。

事件信息将以 Evas_Event_Key_Down 格式保存。键名属性将以字符串格式保存键值。键值的类型如下：

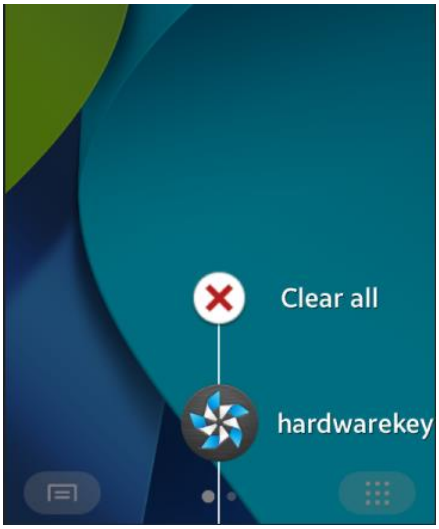
- XF86Menu：Menu 键
- XF86Home：Home 键
- XF86Back：Back 键
- XF86PowerOff：Power 键
- XF86AudioRaiseVolume：Volume Up 键
- XF86AudioLowerVolume：Volume Down 键

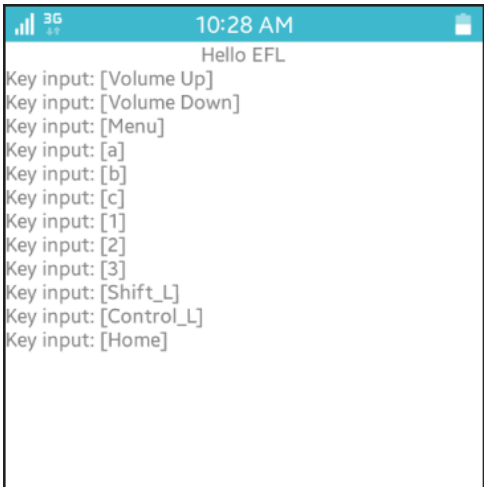
构建并运行该示例。按硬件按键中的 Volume up、Volume Down 和 Menu。键类型将在 Label 小部件中显示。



按键盘上的字母数字键加 Ctrl 和 Shift。每个键值都将显示。如果手机带有内置硬件键盘，您可以请求单个按键输入。

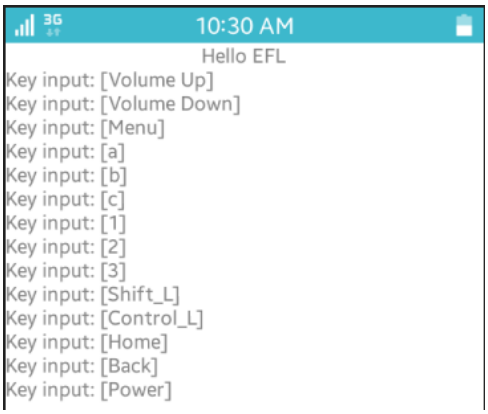
现在按 Home 键。应用程序将消失。再次按住 Home 键，可查看目前已运行的应用程序列表。从列表中选择 HardwareKeyEvent。您将再次看到该示例，并确认 Home 键已被接受。





现在按 Back 键。一旦应用程序消失，按住 Home 键，并从应用程序列表中选择 HardwareKeyEvent。

接着，按 Power 键。屏幕关闭后，按 Home 键，并在屏幕上拖动，以将其解锁。



2) 通过 Log 消息来检查

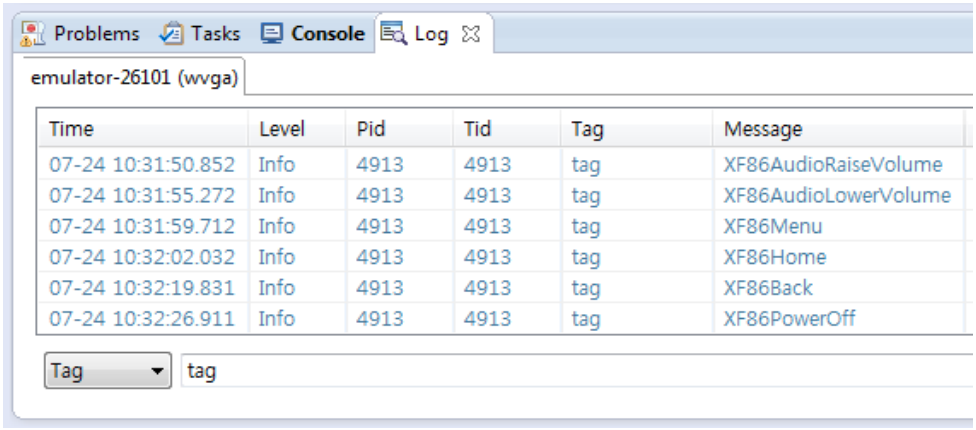
很难立即检查 Home 键、Back 键和 Power 键。使用 Log 消息能方便检查。向 on_keydown_cb() 函数添加一行新代码。

```
static void
on_keydown_cb(void *data, Evas *evas, Evas_Object *o, void *event_info)
{
        appdata_s* ad = data;
        Evas_Event_Key_Down *ev = event_info;
        dlog_print(DLOG_INFO, "tag", ev->keyname);

        char *old_msg = elm_object_text_get(ad->label);
        char total_msg[PATH_MAX];
        char *key_value = strdup( ev->keyname );
```

此代码会将键值输出到 Log 消息。

在面板 Select the Tag 下的组合框中选择 Log，然后在右侧的编辑框中输入“tag”。

再次运行该示例，并按 Hardware 键。按 Home 键、Back 键或 Power 键可促使屏幕消失，但消息保持不变。



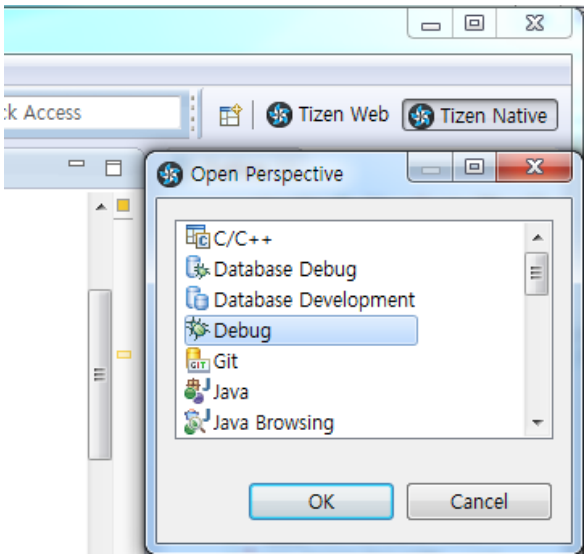
3) 调试模式

我们将学习如何在特定的源代码位置指定断点并实时检查变量的值。

在 on_keydown_cb() 函数的下一代码左侧，单击 Edit 窗口的边框。您将会看到一个淡蓝色圆圈。这是断点。
char *key_value = strdup(ev->keyname);

```
static void
on_keydown_cb(void *data, Evas *evas, Evas_Object *o, void *event_info)
{
    appdata_s* ad = data;
    Evas_Event_Key_Down *ev = event_info;
    dlog_print(DLOG_INFO, "tag", ev->keyname);

    char *old_msg = elm_object_text_get(ad->label);
    char total_msg[PATH_MAX];
    char *key_value = strdup( ev->keyname );

    if( strcmp(ev->keyname, "XF86Menu") ==0 ) {
        key_value = "Menu";
    }
```

现在我们将其改为调试模式。从 Eclipse 右上角的选项卡按钮中选择 Debug。如果没有看到名为 Debug 的选项卡按钮，则选择 Open Perspective。在显示的弹出窗口中，从列表选择 Debug。



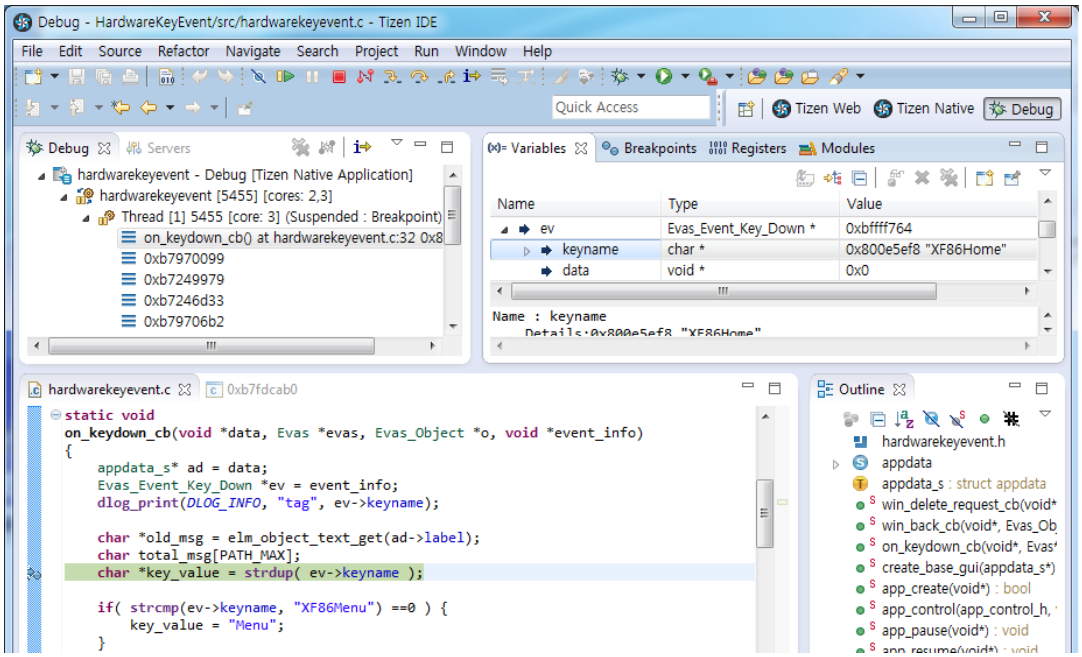
现在我们开始调试。通常使用 Ctrl + F11 键运行应用程序；但要进行调试，最好使用 F11 键。您也可以从主菜单中选择 [Run > Debug]。

如果应用程序在运行到 main() 函数时停止运行，请按 F8 键跳转到下一步。

如果应用程序仍在运行，请按 Menu 键。断点上会显示一个箭头，然后停止工作。

在 Eclipse 右上角的 Variables 面板中，您将会看到一个变量列表。在此处选择 ev 以打开树状列表。
将会在子属性中的键名称右侧显示一个键值。

要继续，请按 F8 键转到下一步。



可在调试模式中使用以下键盘快捷方式：

- F11：开始调试
- F8：下一步
- F5：进入
- F6：跨过
- F7：单步返回
- Ctrl + F2：停止调试

4）相关 API

您可以通过将 EVAS_CALLBACK_KEY_DOWN 传递给 void evas_object_event_callback_add(Evas_Object *obj, Evas_Callback_Type type, Evas_Object_Event_Cb func, void *data) 函数的第二个参数来请求一个硬件键按下事件。

此硬件键事件信息以 Evas_Event_Key_Down 格式保存。键名属性将以字符串格式保存键值。键值的类型如下：

- XF86Menu：Menu 键
- XF86Home：Home 键
- XF86Back：Back 键

- XF86PowerOff：Power 键
- XF86AudioRaiseVolume：Volume Up 键
- XF86AudioLowerVolume：Volume Down 键

int dlog_print(log_priority prio, const char *tag, const char *fmt, ...)：一种实时输出 Log 消息的 API。

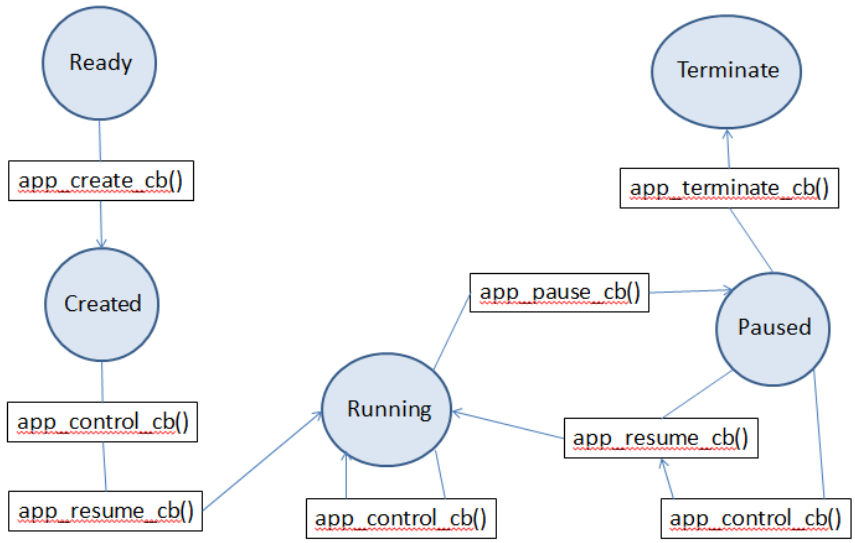
可在调试模式中使用以下键盘快捷方式：
- F11：开始调试
- F8：下一步
- F5：进入
- F6：跨过
- F7：单步返回
- Ctrl + F2：停止调试

49. 生命周期和调试模式

当初次运行此应用程序时，将会创建组成其 UI 的容器和小部件。此时，您可以轻松地将内存分配给变量。当应用程序关闭时，您需要删除内存。如需要播放背景音乐，请播放初次运行该应用程序时曾播放过的音乐。当应用程序终止运行时，您必须停止播放此音乐。当应用程序隐藏在后台时，屏幕上的动画应停止显示以降低 CPU 负荷。当从后台切换到前台时，必须恢复动画显示。我们将在本例中讨论生命周期。

1）生命周期事件

有关 Tizen 原生应用程序的生命周期，请参阅以下图表：



应用程序通常按以下顺序执行：app_create_cb() => app_control_cb() => app_resume_cb() 函数。

当退出此应用程序时，将调用 app_terminate_cb()。

当暂停应用程序（例如，切换到后台模式）时，将调用 app_pause_cb() 函数。

当恢复应用程序时，将调用 app_resume_cb() 函数。

如果另一应用程序发出了应用程序 Launch 请求，则调用 app_control_cb() 函数。

2) 生命周期的事件函数

创建新的源项目，将项目名称指定为 LifeCycle。创建源项目之后，打开 src 文件夹中的源文件 (*.c)，并在顶端添加新代码。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
} appdata_s;

static void
show_message(appdata_s* ad, const char* msg)
{
        dlog_print(DLOG_ERROR, "tag", msg);
        char *old_msg = elm_object_text_get(ad->label);
        char total_msg[PATH_MAX];
        sprintf(total_msg, "%s<br/>%s", old_msg, msg);
        elm_object_text_set(ad->label, total_msg);
}

static void
win_back_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        show_message(ad, "win_back_cb()");
        /* Let window go to hide state. */
    elm_win_iconified_set(ad->win, EINA_TRUE);
}
```

show_message() 会将一条新消息添加到 Label 小部件。我们将其置于顶部，这样便于从任何位置调用它。

win_back_cb() 是一个在单击 Back 按钮时运行的事件回调函数。为第一个参数传递 appdata 对象。

在源文件下面将会看到各个事件回调函数及其定义。为其中一个函数添加与生命周期关联的代码。

```
static bool
app_create(void *data)
{
        appdata_s *ad = data;
        create_base_gui(ad);
        show_message(ad, "app_create()");
        return true;
}

static void
app_control(app_control_h app_control, void *data)
{
        appdata_s *ad = data;
        show_message(ad, "app_control()");
}

static void
app_pause(void *data)
{
        appdata_s *ad = data;
        show_message(ad, "app_pause()");
}

static void
app_resume(void *data)
{
        appdata_s *ad = data;
        show_message(ad, "app_resume()");
}

static void
app_terminate(void *data)
{
        appdata_s *ad = data;
        show_message(ad, "app_terminate()");
}
```

app_create() 是一种在创建应用程序时执行的回调函数。您可以调用 create_base_gui() 函数，它会从此处创建一个 UI 对象。

app_control() 是一种在其他应用程序中发出应用程序 Launch 请求时运行的回调函数。

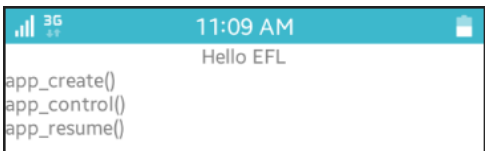
app_pause() 是一种在应用程序切换到后台模式时执行的回调函数。

app_resume() 是一种在应用程序从后台模式切换到前台模式时执行的回调函数。

app_terminate() 是一种在退出应用程序时执行的回调函数。

您可以在 main() 函数中指定这 5 个回调函数。通常您可以在 main() 函数中更改这些回调函数。

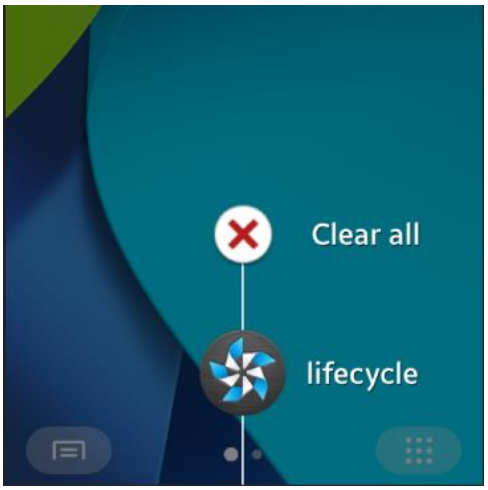
构建并运行该示例。当应用程序开始运行时，确认这 3 个附加函数已运行。其顺序是 app_create() => app_control() => app_resume()。

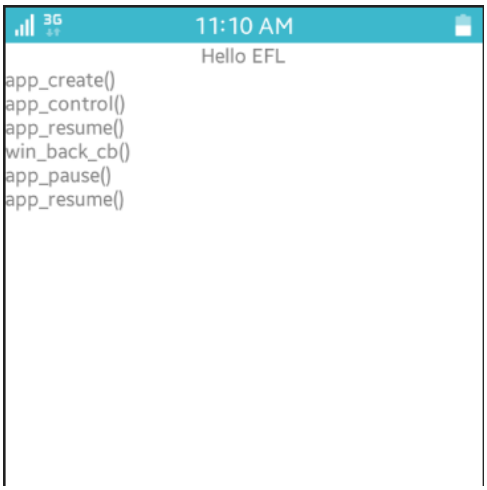


应用程序在运行时，单击 Back 按钮。现在在任何位置都找不到此应用程序，但这并不表明应用程序已关闭。它仍在后台模式下运行。

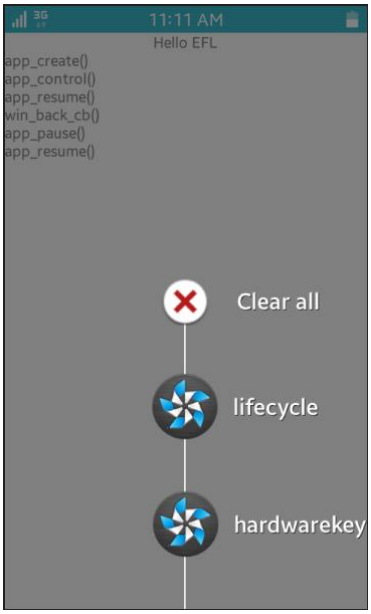
我们将再次调用此应用程序。单击并按住 Home 按钮，查看到目前为止您已运行的应用程序列表。从列表中选择 LifeCycle。

您将会再次看到此示例；确认这 3 个附加函数已运行。应用程序按以下顺序运行：win_back_cb() => app_pause() => app_resume()。Back 按钮用于运行 win_back_cb() 函数。当应用程序切换到后台模式时，将执行 app_resume() 函数。如果切换到前台模式，则执行 app_resume() 函数。



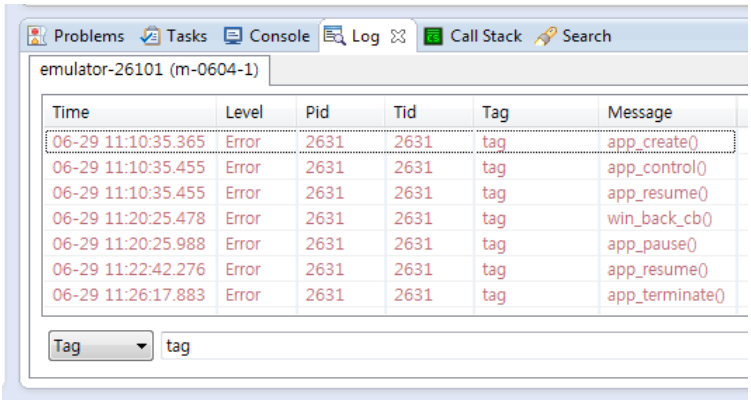


我们现在要退出应用程序。单击并按住 Home 按钮以查看应用程序列表。单击 Clear all 以退出所有应用程序。



应用程序已关闭。新消息的传递非常快，因而无法查看这些消息。幸运的是，我们有一个称为实时 Log 消息的项。在 Eclipse 底部选择 Log 面板，然后在组合框中选择 Tag。然后在右侧的编辑框中输入“tag”。

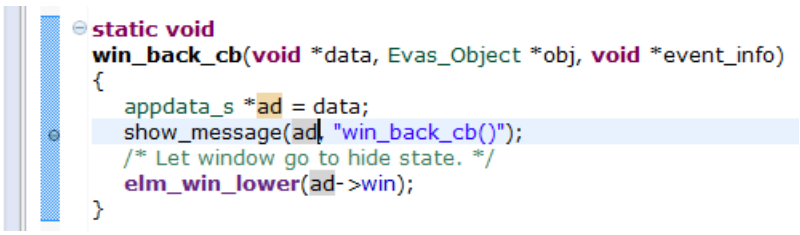
输出消息显示在 Label 小部件中。您将会看到 app_terminate() 已添加到底部。这意味着当您关闭应用程序时执行了 app_terminate() 函数。



3) 调试模式

我们将学习如何在特定的源代码位置指定断点并实时检查变量的值。

请转到 win_back_cb() 函数，点按 show_message() 调用代码左侧的 Edit 窗口边框。您将会看到一个淡蓝色圆圈。这是断点。



```
static void
win_back_cb(void *data, Evas_Object *obj, void *event_info)
{
   appdata_s *ad = data;
   show_message(ad, "win_back_cb()");
   /* Let window go to hide state. */
   elm_win_lower(ad->win);
}
```

同样，在以下 5 个函数中指定 show_message() 调用代码的断点：

- app_create()
- app_control()
- app_pause()
- app_resume()
- app_terminate()

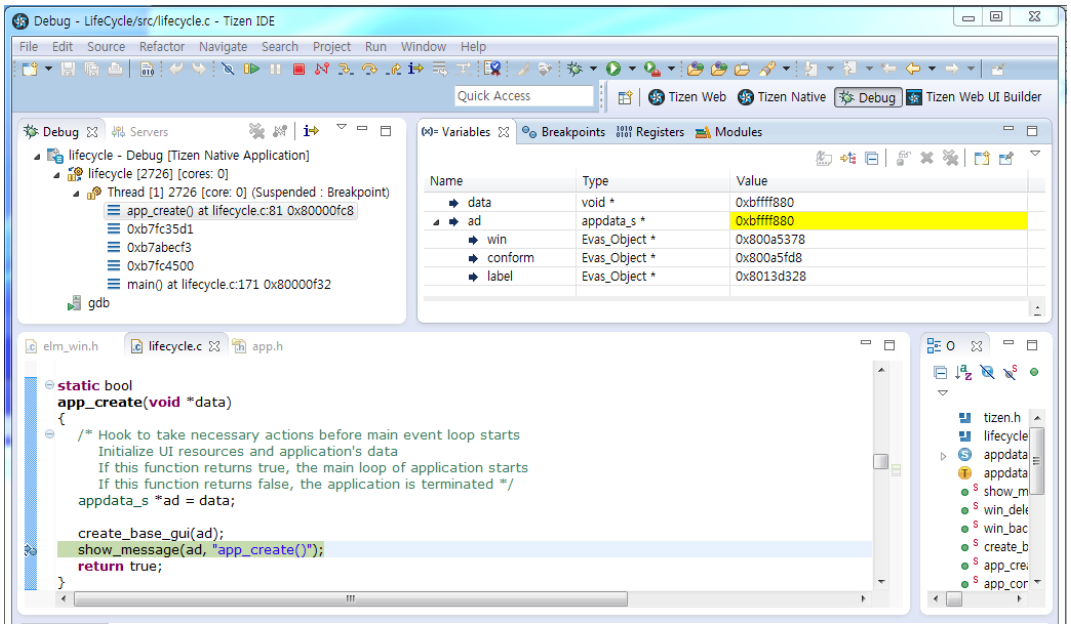
现在我们将其改为调试模式。从 Eclipse 右上角的选项卡按钮中选择 Debug。

现在我们开始调试。通常使用 Ctrl + F11 键运行应用程序；但要进行调试，最好使用 F11 键。

如果应用程序在运行到 main() 函数时停止运行，请按 F8 键跳转到下一步。

如果应用程序开始运行，请在 app_create() 函数停止。如果按 F8 键，它会传递断点，并在 app_control() 函数再次停止。

接下来，它会在 app_resume() 函数再次停止。如果再次按 F8 键，应用程序将会显示在屏幕中。



如果点按 Back 按钮，则会在 win_back_cb() 函数停止。如果按 F8 键，则会在 app_pause() 函数停止。

如果再次按 F8 键，应用程序将从屏幕中消失。

如果按住 Home 键，然后从应用程序列表中选择 LifeCycle，则将会在 app_resume() 函数停止。

按 F8 键会让应用程序重新显示在屏幕中。

可在调试模式中使用以下键盘快捷方式：

- F11：开始调试
- F8：下一步
- F5：进入

- F6：跨过
- F7：单步返回
- Ctrl + F2：停止调试

4）相关 API

void win_back_cb(void *data, Evas_Object *obj, void *event_info)：一种在用户单击 Back 按钮时执行的事件回调函数。为第一个参数传递 appdata 对象。

bool app_create(void *data)：一种在创建应用程序时执行的回调函数。您可以调用 create_base_gui() 函数，它会从此处创建一个 UI 对象。

void app_control()：一种在其他应用程序中发出应用程序 Launch 请求时运行的回调函数。

void app_pause()：一种在应用程序切换到后台模式时执行的回调函数。

void app_resume()：一种在应用程序从后台模式切换到前台模式时执行的回调函数。

void app_terminate()：一种在退出应用程序时执行的回调函数。

可在调试模式中使用以下键盘快捷方式：
- F11：开始调试
- F8：下一步
- F5：进入
- F6：跨过
- F7：单步返回
- Ctrl + F2：停止调试

50. 如何使用 Notify

当要向用户传递新消息时使用 Notify。您可以让消息在一小段时间后消失，甚至可以添加诸如 Button 之类的小部件。您还可以拦截用户的 UI 事件以及将它们显示在底部。

1) 为 Notify 指定 Timeout

创建新的源项目，将项目名称指定为 NotifyEx。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在 create_base_gui() 函数之上添加一个新的函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

然后，将新代码添加到 create_base_gui() 函数中。此代码将创建一个 Box 容器、一个 Button 和一个 Notify 对象。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);

            Evas_Object* notify = create_notify_top_timeout(box);

            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Top / Time out");
            evas_object_smart_callback_add(btn, "clicked", btn_click_cb, notif
y);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

create_notify_top_timeout() 函数会创建 Notify 并指定 Timeout。现在开始吧！在 create_base_gui() 函数之上创建一个新函数。

```
static Evas_Object*
create_notify_top_timeout(Evas_Object *parent)
{
        Evas_Object *notify;
        Evas_Object *box;
        Evas_Object *label;

        /* Create notify (top-aligned / hide automatically) */
        notify = elm_notify_add(parent);
        elm_notify_align_set(notify, 0.5, 0.0);
        elm_notify_timeout_set(notify, 3.0);

        /* Create box for stacking notify message */
        box = elm_box_add(notify);
        evas_object_show(box);

        /* Create label for notify message */
        label = elm_label_add(box);
        evas_object_size_hint_min_set(label, ELM_SCALE_SIZE(480), 0);
        elm_label_line_wrap_set(label, ELM_WRAP_WORD);
        elm_object_text_set(label, "<font align=center>This notification will hi
de automatically in 3 seconds  later.</font>");
        elm_box_pack_end(box, label);
        evas_object_show(label);

        elm_object_content_set(notify, box);
        return notify;
}

static void
btn_click_cb(void *data, Evas_Object *obj, void *event_info)
{
        Evas_Object *notify = data;
        evas_object_show(notify);
}
```

elm_notify_add(Evas_Object *) 是一种用于创建 Notify 对象的 API。

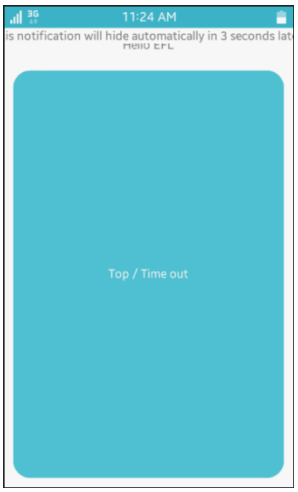
elm_notify_align_set(Evas_Object *, double, double) 是一种按屏幕高宽比指定 Notify 位置的 API。为第二个参数指定水平位置。它置于左侧为 0、中间为 0.5、右侧为 1 的位置。为第二个参数指定垂直位置。它置于上部为 0、中央为 0.5、下部为 1 的位置。

elm_notify_timeout_set(Evas_Object *, double) 是一种为 Notify 指定 Timeout 的 API。为第二个参数指定时间间隔，单位为秒。

下一个代码会在 Notify 中创建 Box 容器，在 Box 容器中创建 Label 小部件。

当用户单击 Button 时，btn_click_cb() 函数会在屏幕上显示 Notify。

构建并运行该示例。如果点按 Button，Notify 会显示 3 秒后消失。



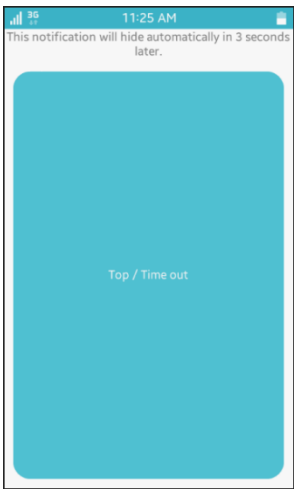
2) 调整 Notify 的大小

有时，Notify 中显示的文本字符串会被截短。我们将尝试通过更改应用程序的基本比例来解决此问题。将一行新代码添加到 app_create() 函数。

```
static bool
app_create(void *data)
{
        appdata_s *ad = data;
        elm_app_base_scale_set(1.8);
        create_base_gui(ad);

        return true;
}
```

elm_app_base_scale_set(double) 是一种更改应用程序基本比例的 API。默认比例为 1.0。当比例值增加时，Notify 区域变小。

再次运行此示例并点击 Button。此时，Notify 的所有文本均正确显示。



3) 为 Notify 添加 Button 小部件

我们将通过为 Notify 添加 Button 来实施一项功能，以便在您点按 Button（而不是 Timeout）时让 Notify 消失。向 create_base_gui() 函数添加新代码。

```
        /* Button-1 */
        Evas_Object *btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Top / Time out");
        evas_object_smart_callback_add(btn, "clicked", btn_click_cb, notify);
        my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

        notify = create_notify_top_manual(box);

        /* Button-2 */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Top / Manual");
        evas_object_smart_callback_add(btn, "clicked", btn_click_cb, notif
y);
        my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
```

```
        }
    }
```

此代码将创建第二个 Notify 和第二个 Button。在 create_base_gui() 函数之上添加两个新函数。

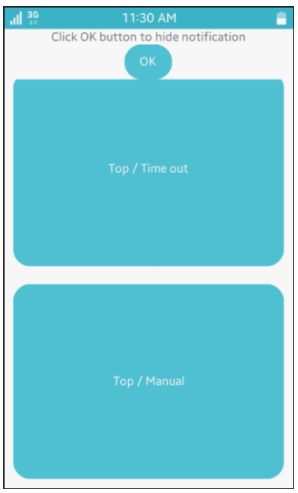
```
static void
btn_hide_notify_cb(void *data, Evas_Object *obj, void *event_info)
{
        Evas_Object *notify = data;
        evas_object_hide(notify);
}

static Evas_Object*
create_notify_top_manual(Evas_Object *parent)
{
        Evas_Object *notify;
        Evas_Object *box;
        Evas_Object *label;
        Evas_Object *btn;

        /* Create notify (top-aligned / hide manually) */
        notify = elm_notify_add(parent);
        elm_notify_align_set(notify, 0.5, 0.0);
        elm_notify_timeout_set(notify, 0.0);

        /* Create box for stacking notify message and button vertically */
        box = elm_box_add(notify);
        elm_box_horizontal_set(box, EINA_FALSE);
        evas_object_show(box);

        /* Create label for notify message */
        label = elm_label_add(box);
        evas_object_size_hint_min_set(label, ELM_SCALE_SIZE(480), 0);
        elm_label_line_wrap_set(label, ELM_WRAP_WORD);
        elm_object_text_set(label, "<font align=center>Click OK button to hide n
otification</center>");
        elm_box_pack_end(box, label);
        evas_object_show(label);

        /* Create button to hide notify */
        btn = elm_button_add(box);
        elm_object_text_set(btn, "OK");
```

```
        evas_object_size_hint_min_set(btn, ELM_SCALE_SIZE(80), ELM_SCALE_SIZE(5
8));
        elm_box_pack_end(box, btn);
        evas_object_show(btn);
        evas_object_smart_callback_add(btn, "clicked", btn_hide_notify_cb, notif
y);

        elm_object_content_set(notify, box);

        return notify;
}
```

btn_hide_notify_cb() 会隐藏 Notify。单击已添加到 Notify 的 Button，则会调用此函数。

create_notify_top_manual() 会创建第二个 Notify 对象。此代码会在 Notify 中创建 Box 容器，并在 Box 容器中创建 Label 小部件和 Button 小部件。

再次运行该示例，然后点击第二个 Button。将显示 Notify，您将会看到已添加 Label 和 Button。单击 Button 会导致 Notify 消失。



4) Notify 期间拦截事件

我们将实施一项功能以在显示 Notify 期间拦截用户事件。向 create_base_gui() 函数添加新代码。

```
            /* Button-2 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Top / Manual");
            evas_object_smart_callback_add(btn, "clicked", btn_click_cb, notify);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            notify = create_notify_top_block(box);

            /* Button-3 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Top / Block");
            evas_object_smart_callback_add(btn, "clicked", btn_click_cb, notif
y);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

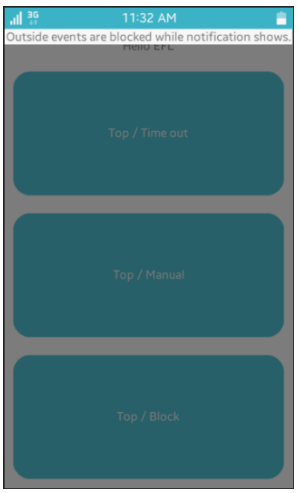
此代码将创建第三个 Notify 和第三个 Button。在 create_base_gui() 函数之上创建一个新函数。

```
static Evas_Object*
create_notify_top_block(Evas_Object *parent)
{
        Evas_Object *notify;
        Evas_Object *box;
        Evas_Object *label;

        /* Create notify (top-aligned / hide automatically / block outside event
s) */
        notify = elm_notify_add(parent);
        elm_notify_align_set(notify, 0.5, 0.0);
        elm_notify_timeout_set(notify, 3.0);
        elm_notify_allow_events_set(notify, EINA_FALSE);

        /* Create box for stacking notify message */
        box = elm_box_add(notify);
```

```
        evas_object_show(box);

        /* Create label for notify message */
        label = elm_label_add(box);
        evas_object_size_hint_min_set(label, ELM_SCALE_SIZE(480), 0);
        elm_label_line_wrap_set(label, ELM_WRAP_WORD);
        elm_object_text_set(label, "<font align=center>Outside events are blocke
d while notification shows.</center>");
        elm_box_pack_end(box, label);
        evas_object_show(label);

        elm_object_content_set(notify, box);
        return notify;
}
```

elm_notify_allow_events_set(Evas_Object *, Eina_Bool) 是一个确定在 Notify 处于 Show 状态时是否允许用户事件的函数。如果为第二个参数传递 EINA_FALSE，则会拦截用户事件。

再次运行该示例，然后点击第三个 Button。Notify 显示 3 秒后再次消失。当 Notify 处于 Show 状态时，单击其他 Button 不起作用。



5) 更改 Notify 的位置

我们将实施一项功能以在底部显示 Notify。向 create_base_gui() 函数添加新代码。

```
            /* Button-3 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Top / Block");
            evas_object_smart_callback_add(btn, "clicked", btn_click_cb, notify);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            notify = create_notify_bottom_timeout(box);

            /* Button-4 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Bottom / Timeout");
            evas_object_smart_callback_add(btn, "clicked", btn_click_cb, notif
y);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

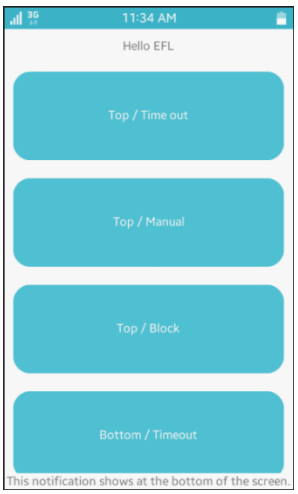
此代码将创建第四个 Notify 和第四个 Button。在 create_base_gui() 函数之上创建一个新函数。

```
static Evas_Object*
create_notify_bottom_timeout(Evas_Object *parent)
{
        Evas_Object *notify;
        Evas_Object *box;
        Evas_Object *label;

        /* Create notify (bottom-aligned / hide automatically) */
        notify = elm_notify_add(parent);
        elm_notify_align_set(notify, 0.5, 1.0);
        elm_notify_timeout_set(notify, 3.0);

        /* Create box for stacking notify message */
        box = elm_box_add(notify);
        evas_object_show(box);
```

```
        /* Create label for notify message */
        label = elm_label_add(box);
        evas_object_size_hint_min_set(label, ELM_SCALE_SIZE(480), 0);
        elm_label_line_wrap_set(label, ELM_WRAP_WORD);
        elm_object_text_set(label, "<font align=center>This notification shows a
t the bottom of the screen.</center>");
        elm_box_pack_end(box, label);
        evas_object_show(label);

        elm_object_content_set(notify, box);
        return notify;
}
```

elm_notify_align_set(Evas_Object *, double, double) 是一种按屏幕高宽比指定 Notify 位置的 API。为第二个参数指定水平位置。它置于左侧为 0、中间为 0.5、右侧为 1 的位置。为第二个参数指定垂直位置。它置于上部为 0、中央为 0.5、下部为 1 的位置。

再次运行该示例，然后点击第四个 Button。Notify 在底部短暂显示，然后消失。



6）相关 API

Evas_Object *elm_notify_add(Evas_Object *parent)：一种用于创建 Notify 对象的 API。

void elm_notify_align_set(Evas_Object *obj, double horizontal, double vertical)：一种按屏幕高宽比指定 Notify 位置的 API。为第二个参数指定水平位置。它置于左侧为 0、中间为 0.5、右侧为 1 的位置。为第二个参数指定垂直位置。它置于上部为 0、中央为 0.5、下部为 1 的位置。

void elm_notify_timeout_set(Evas_Object *obj, double timeout)：一种为 Notify 指定 Timeout 的 API。为第二个参数指定时间间隔，单位为秒。

void elm_notify_allow_events_set(Evas_Object *obj, Eina_Bool allow)：一个确定在 Notify 处于 Show 状态时是否允许用户事件的函数。如果为第二个参数传递 EINA_FALSE，则会拦截用户事件。

51. 如何使用加速传感器

您可以使用加速器传感器测量电话的晃动情况。您可以测量 X、Y 和 Z 轴的方向。此外，还可以选择是应用重力还是放弃重力，并分隔结果。要在模拟器中测试加速器传感器，请使用 Control Panel。

1) 确定是否支持加速器传感器

创建新的源项目，将项目名称指定为 SensorAcceleration。创建源项目之后，打开 src 文件夹中的源文件 (*.c)，并添加一个库头文件以及变量。

```
#include "sensoracceleration.h"
#include <sensor.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
        Evas_Object *label2;
} appdata_s;
```

sensor.h 是用于各传感器库的头文件。

我们将在 label0 中显示是否支持加速器传感器，在 label1 中显示当前的加速值，以及在 label2 中显示最大加速值。

在 create_base_gui() 函数之上创建一个新函数。该函数将向一个 Box 容器添加一个小部件。

在 create_base_gui() 函数之上创建两个新函数。

```
static void show_is_supported(appdata_s *ad)
{
        char buf[PATH_MAX];
        bool is_supported = false;
```

```
        sensor_is_supported(SENSOR_ACCELEROMETER, &is_supported);
        sprintf(buf, "Acceleration Sensor is %s", is_supported ? "support" : "no
t support");
        elm_object_text_set(ad->label0, buf);
}

static void
my_box_pack(Evas_Object *box, Evas_Object *child,
        double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

The show_is_supported() 函数将指出是否支持加速器传感器，然后在第一个 Label 小部件中显示结果。

sensor_is_supported(sensor_type_e, bool *) 是一种请求是否支持特定传感器的 API。将 SENSOR_ACCELEROMETER 传递给第一个参数，则会让第二个参数返回是否支持传感器。

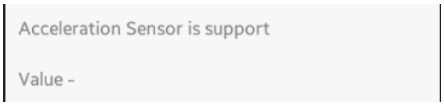
my_box_pack() 是一种用于将小部件添加到 Box 容器中的函数。

您应当做的就是在应用程序正在运行时调用 show_is_supported() 函数。在 create_base_gui() 函数末尾调用上述函数。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        Evas_Object * box, *btn;

        /* A box to put things in verticallly - default mode for box */
        box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        { /* child object - indent to how relationship */
            /* Label-0 */
            ad->label0 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label0, "Msg - ");
            my_box_pack(box, ad->label0, 1.0, 0.0, -1.0, 0.0);

            /* Label-1 */
            ad->label1 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label1, "Value - ");
            my_box_pack(box, ad->label1, 1.0, 0.0, -1.0, 0.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    show_is_supported(ad);
}
```

我们已添加两个 Label 小部件。另外，还调用了一个函数以确定是否支持此传感器。

构建并运行该示例。如果支持加速器传感器，则会看到消息“Acceleration Sensor is supported.”并非所有智能手机都支持此传感器。如果属于这种情况，请在模拟器中测试它。



2) 请求加速器传感器的事件

我们将实施一项功能，以在您摇晃手机时请求相应事件并在屏幕上显示加速值。在源文件顶部添加传感器结构以及一个全局变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
        Evas_Object *label2;
} appdata_s;

typedef struct _sensor_info
{
        sensor_h sensor;                /**< Sensor handle */
        sensor_listener_h sensor_listener;
} sensorinfo;

static sensorinfo sensor_info;
```

sensorinfo 是一种包含传感器对象和事件侦听器变量的结构。

sensor_info 是 sensorinfo 结构的全局变量。

要请求传感器事件，请启动侦听器。我们将使用传感器对象和事件侦听器来请求一个加速器传感器事件。在 create_base_gui() 函数之上创建两个新函数。

```
static void _new_sensor_value(sensor_h sensor, sensor_event_s *sensor_data, void
 *user_data)
{
        if( sensor_data->value_count < 3 )
                return;
        char buf[PATH_MAX];
        appdata_s *ad = (appdata_s*)user_data;

        sprintf(buf, "Value - X : %0.1f / Y : %0.1f / Z : %0.1f",
                        sensor_data->values[0], sensor_data->values[1], sensor_d
ata->values[2]);
        elm_object_text_set(ad->label1, buf);
}

static void
start_acceleration_sensor(appdata_s *ad)
{
        sensor_error_e err = SENSOR_ERROR_NONE;
        sensor_get_default_sensor(SENSOR_ACCELEROMETER, &sensor_info.sensor);
        err = sensor_create_listener(sensor_info.sensor, &sensor_info.sensor_lis
tener);
        sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sens
or_value, ad);
        sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() 是加速器传感器的事件回调函数。将新的传感器值输出到屏幕上。此传感器数据被传递给第二个参数。values[0] 包含 X 轴数据，values[1] 包含 y 轴数据，而 values[2] 包含 z 轴数据。

start_acceleration_sensor() 是一种启动加速器传感器并指定事件回调函数的回调函数。

sensor_get_default_sensor(sensor_type_e, sensor_h *) 是一种返回传感器对象的 API。将 SENSOR_ACCELEROMETER 传递给第一个参数，则会将加速器传感器对象返回给第二个参数。

sensor_create_listener(sensor_h, sensor_listener_h *) 是一种可创建事件侦听器的 API。将传感器对象传递给第一个参数，则会将侦听器对象返回给第二个参数。

sensor_listener_set_event_cb(sensor_listener_h, unsigned int, sensor_event_cb, void *) 是一种将回调函数指定给侦听器的 API。参数顺序如下：事件侦听器、时间间隔（单位：毫秒）、回调函数名称和用户数据。

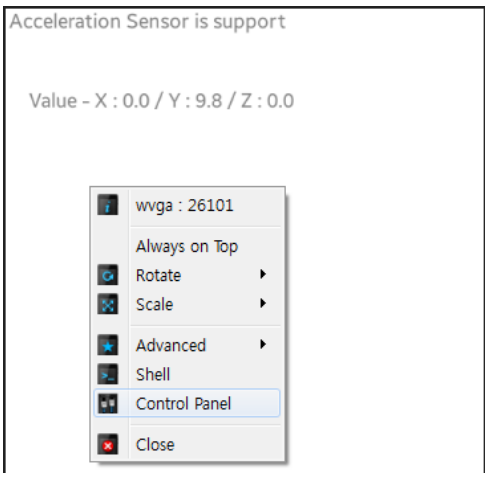
sensor_listener_start(sensor_listener_h) 是一种启动侦听器的 API。

当应用程序开始运行时将自动操作事件侦听器。在 create_base_gui() 函数末尾调用上述函数。

```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        show_is_supported(ad);
        start_acceleration_sensor(ad);
}
```

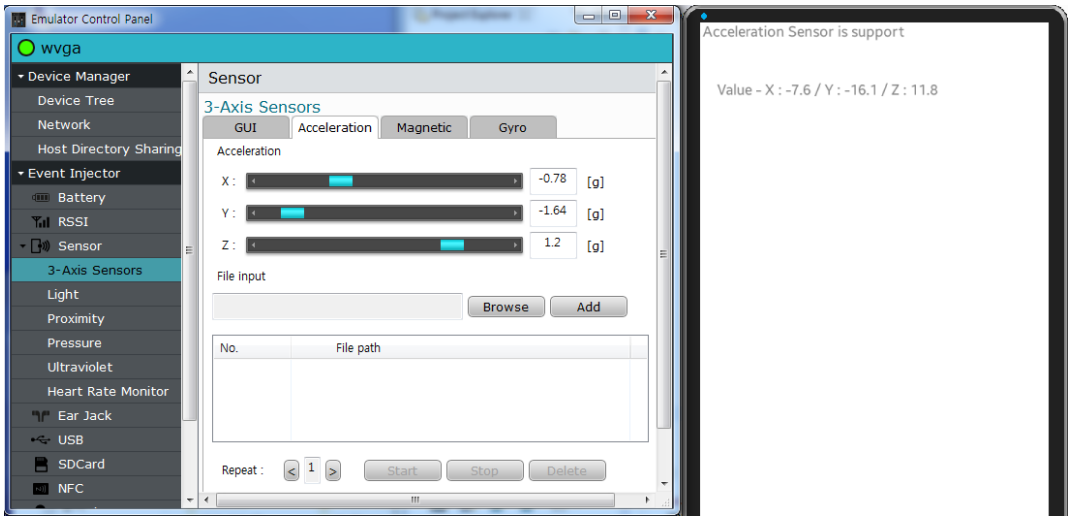
让我们再运行一次示例。要在智能手机上执行测试，只须摇晃手机。使用 Control Panel 在模拟器上执行测试。

右键单击模拟器，在快捷菜单中选择 Control Panel。



在 Control Panel 中，从左侧的树状列表中选择 [Event Injector > 3-Axis Sensors]，然后从屏幕右侧的选项卡按钮中选择 Acceleration。

拖动 3 个滑块，一次拖动一个。如果应用程序屏幕中的 X、Y 和 Z 值发生变化，则表明您已正确接收加速器数据。



3）请求最大加速值

如果您使用智能手机执行此测试，当您摇晃手机时，将难以看清这些字符。而当您停止摇晃以查看这些值时，向下方向为 9.8，其余值重置为 0。因此，我们需要一项功能来保存在手机上测试时得到的最大值。

在源文件顶部以数字格式声明一个数组变量，并将其重置为 0。此变量用于保存最大加速值。

```
typedef struct _sensor_info
{
        sensor_h sensor;                /**< Sensor handle */
        sensor_listener_h sensor_listener;
} sensorinfo;

static sensorinfo sensor_info;

float value[3] = {0.f, 0.f, 0.f};
```

向 create_base_gui() 函数添加新代码。

```
        /* Label-1 */
        ad->label1 = elm_label_add(ad->conform);
```

```
            elm_object_text_set(ad->label1, "Value - ");
            my_box_pack(box, ad->label1, 1.0, 0.0, -1.0, 0.0);

            /* Button */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Init Max Value");
            evas_object_smart_callback_add(btn, "clicked", btn_clicked, ad);
            my_box_pack(box, btn, 1.0, 0.0, -1.0, -1.0);

            /* Label-2 */
            ad->label2 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label2, "Max - ");
            my_box_pack(box, ad->label2, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

此代码会添加一个 Button 小部件和一个 Label 小部件。我们将实施一项功能以在第三个 Label 中显示加速器传感器的最大值，并在您点按 Button 时将最大值重置为 0。为 _new_sensor_value() 函数添加两个新函数和新代码。

```
static float get_absolute_max(float value1, float value2)
{
        float v1 = value1 > 0.f ? value1 : -value1;
        float v2 = value2 > 0.f ? value2 : -value2;
        float result = v1 > v2 ? v1 : v2;
        return result;
}

static void _new_sensor_value(sensor_h sensor, sensor_event_s *sensor_data, void
 *user_data)
{
        if( sensor_data->value_count < 3 )
                return;
        char buf[PATH_MAX];
        appdata_s *ad = (appdata_s*)user_data;

        sprintf(buf, "Value - X : %0.1f / Y : %0.1f / Z : %0.1f",
                        sensor_data->values[0], sensor_data->values[1], sensor_d
ata->values[2]);
        elm_object_text_set(ad->label1, buf);

        for(int i=0; i < 3; i++)
```

```
                value[i] = get_absolute_max(value[i], sensor_data->values[i]);

        sprintf(buf, "Max - X: %0.1f / Y: %0.1f / Z: %0.1f",
                        value[0], value[1], value[2]);
        elm_object_text_set(ad->label2, buf);
}

/* Button click event function */
static void
btn_clicked(void *data, Evas_Object *obj, void *event_info)
{
        for(int i=0; i < 3; i++)
                value[i] = 0.f;
}
```

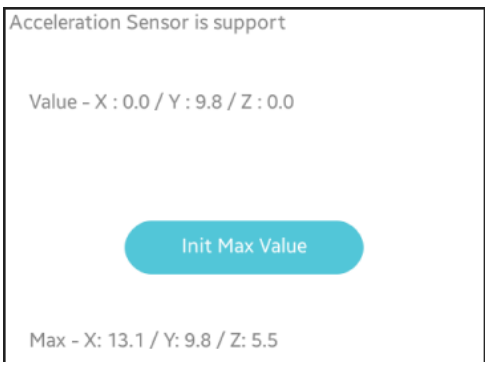
get_absolute_max(float, float) 函数通过将两个实数更改为绝对值来返回其中的较高值。

您添加到 _new_sensor_value() 函数的代码将在全局变量中保存 X、Y 和 Z 轴的最大值，并将其输出到 Label 小部件。

btn_clicked() 函数会在您点击 Button 时将保存在全局变量中的最大值重置为 0。

安装示例并摇晃手机。当停止摇晃时，第二个 Label 中的值被重置，第三个 Label 中的最大值仍保持不变。

您可以按住 Button 并再次摇晃手机以测量新值。



4) 请求不含重力的纯加速值

您可能注意到在结果值中，我们计算的加速值包含重力值。现在我们将获取纯加速值，不包含重力值。

按以下所示更改 start_acceleration_sensor() 函数的代码：

```
static void
start_acceleration_sensor(appdata_s *ad)
{
        sensor_error_e err = SENSOR_ERROR_NONE;
        //sensor_get_default_sensor(SENSOR_ACCELEROMETER, &sensor_info.sensor);
        sensor_get_default_sensor(SENSOR_LINEAR_ACCELERATION, &sensor_info.sens
or);
        err = sensor_create_listener(sensor_info.sensor, &sensor_info.sensor_lis
tener);
        sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sens
or_value, ad);
        sensor_listener_start(sensor_info.sensor_listener);
}
```

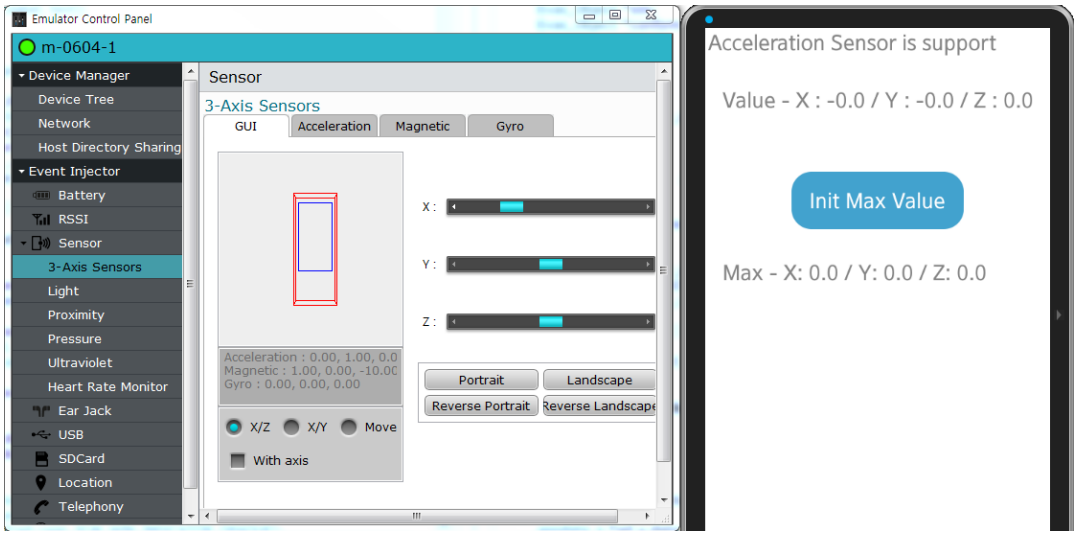
SENSOR_ACCELEROMETER 是一个传感器类型，它代表包含重力的加速器。

SENSOR_LINEAR_ACCELERATION 代表的是不含重力的加速器传感器。

安装示例并摇晃手机。现在您看到的就是纯加速值。

要在模拟器中执行此测试，请在 Control Panel 中选择 [Event Injector > 3-Axis Sensors]，从右侧的选项卡按钮中单击 GUI。

反复单击 Portrait 按钮和 Landscape 按钮。如果将传感器类型设置为 SENSOR_ACCELEROMETER，则 X、Y 和 Z 的总和将等于 9.8。如果设置为 SENSOR_LINEAR_ACCELERATION，则总和将为 0。



5) 相关 API

int sensor_is_supported(sensor_type_e type, bool *supported)：一种请求是否支持特定传感器的 API。将 SENSOR_ACCELEROMETER 传递给第一个参数，则会让第二个参数返回是否支持加速器传感器。

- SENSOR_ACCELEROMETER：包含重力的加速器传感器。

- SENSOR_LINEAR_ACCELERATION：不含重力的加速器传感器。

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor)：一种返回传感器对象的 API。将 SENSOR_ACCELEROMETER 传递给第一个参数，则会将加速器传感器对象返回给第二个参数。

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener)：一种创建事件侦听器的 API。将传感器对象传递给第一个参数，则会将侦听器对象返回给第二个参数。

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data)：一种向侦听器指定回调函数的 API。/ 参数：事件侦听器、时间间隔（单位：毫秒）、回调函数名称和用户数据。

int sensor_listener_start(sensor_listener_h listener)：一种启动侦听器的 API。

52. Gravity 传感器使用方法

用 Gravity 传感器测定设备方向。可对 X、Y、Z 轴方向进行测定。若想在模拟器中测试 Gravity 传感器，使用 Control Panel 即可。

1）判断是否支持 Gravity 传感器

创建新的源项目，将 Project name 命名为 SensorGravity。创建源项目之后，打开 src 文件夹内的源文件 （~.c)，添加库头文件和变量。

```
#include "sensorgravity.h"
#include <sensor.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
} appdata_s;
```

sensor.h 是各种传感器的库头文件。

在 label0 中显示是否支持 Gravity 传感器，在 label1 中显示当前的重力值。

在 create_base_gui() 上创建 2 个新函数。

```
static void show_is_supported(appdata_s *ad)
{
        char buf[PATH_MAX];
        bool is_supported = false;
        sensor_is_supported(SENSOR_GRAVITY, &is_supported);
        sprintf(buf, "Gravity Sensor is %s", is_supported ? "support" : "not sup
port");
        elm_object_text_set(ad->label0, buf);
}
```

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
        double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

show_is_supported() 是判断是否支持 Gravity 传感器后，将结果显示在第 1 个 Label 小部件上的函数。

sensor_is_supported(sensor_type_e, bool *) 是预先定义是否支持特定传感器的 API。在第 1 个参数上传入 SENSOR_GRAVITY，在第 2 个参数上显示是否支持传感器。

my_box_pack() 是在 Box 上添加小部件的函数。

show_is_supported() 函数在运行应用程序时调用即可。在 create_base_gui() 函数末尾调用上述函数。

```
   /* Conformant */
```

```
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        Evas_Object * box, *btn;

        /* A box to put things in verticallly - default mode for box */
        box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        { /* child object - indent to how relationship */
            /* Label-0 */
            ad->label0 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label0, "Msg - ");
            my_box_pack(box, ad->label0, 1.0, 0.0, -1.0, 0.0);

            /* Label-1 */
            ad->label1 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label1, "Value - ");
            my_box_pack(box, ad->label1, 1.0, 1.0, -1.0, 0.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);

show_is_supported(ad);
}
```

已添加 Box 容器和 2 个 Label 小部件。并且调用是否支持传感器的判断函数。

构建并运行示例。如果支持 Gravity 传感器将会显示“Gravity Sensor is support”字样。智能手机中可能会有不支持传感器的情况。在这种情况下，请在模拟器中进行测试。

2）定义 Gravity 传感器事件

下面来展示一下在设备转换方向时，定义相应事件并将重力数值显示在屏幕上的功能。在源文件上端添加传感器相关结构和全局变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
} appdata_s;

typedef struct _sensor_info
{
        sensor_h sensor;                /**< Sensor handle */
        sensor_listener_h sensor_listener;
} sensorinfo;

static sensorinfo sensor_info;
```

Sensorinfo 是包含传感器对象和事件侦听器变量的结构。

sensor_info 是 sensorinfo 结构的全局变量。

定义传感器事件就是调用侦听器。利用传感器对象和事件侦听器来定义 Gravity 传感器事件。在 create_base_gui() 函数上端创建 2 个新函数。

```
static void _new_sensor_value(sensor_h sensor, sensor_event_s *sensor_data, void
 *user_data)
{
        if( sensor_data->value_count < 3 )
                return;
        char buf[PATH_MAX];
        appdata_s *ad = (appdata_s*)user_data;
```

```
        sprintf(buf, "Gravity - X : %0.1f / Y : %0.1f / Z : %0.1f",
                        sensor_data->values[0], sensor_data->values[1], sensor_d
ata->values[2]);
        elm_object_text_set(ad->label1, buf);
}

static void
start_gravity_sensor(appdata_s *ad)
{
        sensor_error_e err = SENSOR_ERROR_NONE;
        sensor_get_default_sensor(SENSOR_GRAVITY, &sensor_info.sensor);
        err = sensor_create_listener(sensor_info.sensor, &sensor_info.sensor_lis
tener);
        sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sens
or_value, ad);
        sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() 是 Gravity 传感器的事件回调函数。新传感器值将显示在屏幕上。

在第 2 个参数上排列传入数据。在 values[0] 中保存 x 轴方向数据，在 values[1] 中保存 y 轴数据，在 values[2] 中保存 z 轴数据。

start_gravity_sensor() 是调用 Gravity 传感器以及定义事件回调函数的函数。

sensor_get_default_sensor(sensor_type_e, sensor_h *) 是返回传感器对象的 API。在第 1 个参数上传入 SENSOR_GRAVITY，第 2 个参数将会返回 Gravity 传感器对象。

sensor_create_listener(sensor_h, sensor_listener_h *) 是创建事件侦听器的 API。在第 1 个参数上传入传感器对象，第 2 个参数将会返回侦听器对象。

sensor_listener_set_event_cb(sensor_listener_h, unsigned int, sensor_event_cb, void *) 是在侦听器中定义回调函数的 API。参数依次为事件侦听器、时间间隔（单位为毫秒）、回调函数名称、用户数据。

sensor_listener_start(sensor_listener_h) 是启动侦听器的 API。

运行应用程序后自动启动事件侦听器。在 create_base_gui() 函数末尾调用上述函数。

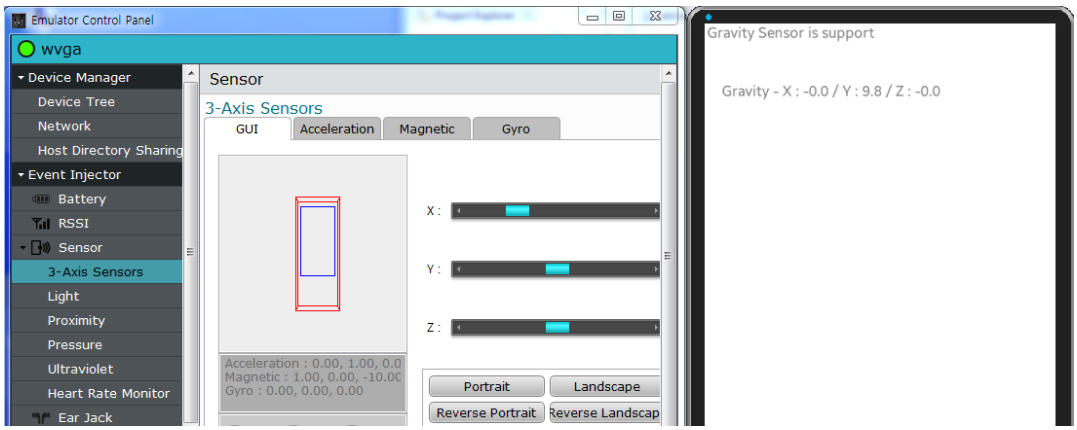
```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        show_is_supported(ad);
        start_gravity_sensor(ad);
}
```

让我们再运行一次示例。在智能手机上进行测试时，将手机旋转即可。在模拟器中测试时，使用 Control Panel 即可。

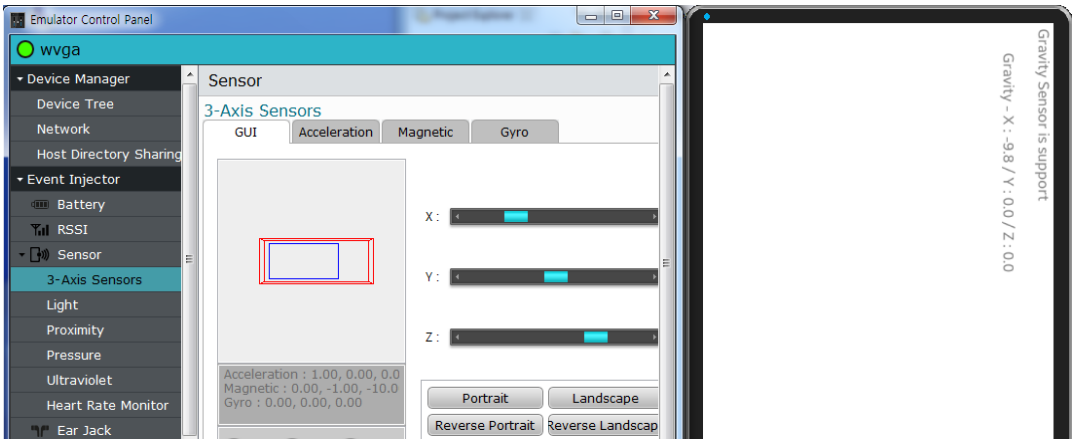
右键点击模拟器，在快捷菜单中选择 Control Panel。

出现 Control Panel 后，在左侧的树形目录中选择 [Event Injector > 3-Axis Sensors]，然后在屏幕右侧选项卡按键中选择 GUI。

按下 Control Panel 屏幕右侧的 Portrait 按键，应用程序屏幕的第 2 个 Label 小部件中显示 'X : 0.0 / Y : 9.8 / Z : 0.0'。



按下 Control Panel 屏幕右侧的 Landscape 按键，应用程序屏幕的第2 个 Label 小部件中显示 'X : 9.8 / Y : 0.0 / Z : 0.0'。



3）相关 API

int sensor_is_supported(sensor_type_e type, bool *supported)：定义是否支持特定传感器的 API。在第 1 个参数上传入 SENSOR_GRAVITY，在第 2 个参数上将会显示是否支持传感器。

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor)：返回传感器对象的 API。在第 1 个参数上传入 SENSOR_GRAVITY，第 2 个参数将会返回 Gravity 传感器对象。

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener)：创建事件侦听器的 API。在第 1 个参数上传入传感器对象，第 2 个参数将会返回侦听器对象。

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data)：定义侦听器回调函数的 API。/ 参数：事件侦听器、时间间隔（单位为毫秒）、回调函数名称、用户数据。

int sensor_listener_start(sensor_listener_h listener)：启动侦听器的 API。

53. Orientation 传感器使用方法

Orientation 传感器可以测量 3 个方向的转换。
- Azimuth 是指南针传感器。将手机平放在地面上时，指示北极的方向和角度差异。
- Pitch 是将手机直立时，指示 Z 轴方向的角度。主要在飞行类游戏或者汽车类游戏中发挥手柄作用。
- Roll 是在 Landscape 模式下将手机放平时，指示 Y 轴方向的角度。主要在飞行类游戏或者汽车类游戏中发挥控制速度的作用。

在 Portrait 模式下将手机直立时，水平方向为 X 轴，垂直方向为 Y轴。前后方向为 Z 轴。若想在模拟器中测试 Orientation 传感器，使用 Control Panel 即可。

Figure: Axis of the device

1）判断是否支持 Orientation 传感器

创建新的源项目，将 Project name 命名为 SensorOrientation。创建源项目之后，打开 src 文件夹内的源文件 （~.c），添加库头文件和变量。

```
#include "sensororientation.h"
#include <sensor.h>

typedef struct appdata {
```

```
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
} appdata_s;
```

sensor.h 是各种传感器的库头文件。

在 label0 显示是否支持 Orientation 传感器，在 label1 中显示当前 Orientation 值。

在 create_base_gui() 上创建 2 个新函数。

```
static void
show_is_supported(appdata_s *ad)
{
        char buf[PATH_MAX];
        bool is_supported = false;
        sensor_is_supported(SENSOR_ORIENTATION, &is_supported);
        sprintf(buf, "Orientation Sensor is %s", is_supported ? "support" : "not
 support");
        elm_object_text_set(ad->label0, buf);
}

static void
my_box_pack(Evas_Object *box, Evas_Object *child,
        double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
```

```
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

show_is_supported() 是判断是否支持 Orientation 传感器后，将结果显示在第 1 个 Label 小部件上的函数。

sensor_is_supported(sensor_type_e, bool *) 是预先定义是否支持特定传感器的 API。在第 1 个参数上传入 SENSOR_ORIENTATION，在第 2 个参数上将会显示是否支持传感器。

my_box_pack() 是在 Box 上添加小部件的函数。

show_is_supported() 函数在运行应用程序后调用即可。在 create_base_gui() 函数末尾调用上述函数。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        Evas_Object * box, *btn;

        /* A box to put things in verticallly - default mode for box */
        box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        { /* child object - indent to how relationship */
```

```
            /* Label-0 */
            ad->label0 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label0, "Msg - ");
            my_box_pack(box, ad->label0, 1.0, 0.0, -1.0, 0.0);

            /* Label-1 */
            ad->label1 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label1, "Orientation - ");
            my_box_pack(box, ad->label1, 1.0, 1.0, -1.0, 0.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    show_is_supported(ad);
}
```

创建 Box 容器和 2 个 Label 小部件。并且调用是否支持传感器的判断函数。

构建并运行示例。如果支持 Orientation 传感器将会显示“Orientation Sensor is support”字样。智能手机中可能会有不支持传感器的情况。在这种情况下，请在模拟器中进行测试。

2）定义 Orientation 传感器事件

下面来展示一下在设备变换方向时，定义相应事件并将数值显示在屏幕上的功能。在源文件上端添加传感器相关结构和全局变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
```

```
} appdata_s;

typedef struct _sensor_info
{
        sensor_h sensor;                /**< Sensor handle */
        sensor_listener_h sensor_listener;
} sensorinfo;

static sensorinfo sensor_info;
```

Sensorinfo 是包含传感器对象和事件侦听器变量的结构。

sensor_info 是 sensorinfo 结构的全局变量。

定义传感器事件始于侦听器。利用传感器对象和事件侦听器来定义 Orientation 传感器事件。在 create_base_gui() 函数上端创建 2 个新函数。

```
static void
_new_sensor_value(sensor_h sensor, sensor_event_s *sensor_data, void *user_data)
{
        if( sensor_data->value_count < 3 )
                return;
        char buf[PATH_MAX];
        appdata_s *ad = (appdata_s*)user_data;

        sprintf(buf, "Azimuth : %0.1f <br>Pitch : %0.1f <br>Roll : %0.1f",
                        sensor_data->values[0], sensor_data->values[1], sensor_d
ata->values[2]);
        elm_object_text_set(ad->label1, buf);
}

static void
start_orientation_sensor(appdata_s *ad)
{
        sensor_error_e err = SENSOR_ERROR_NONE;
        sensor_get_default_sensor(SENSOR_ORIENTATION, &sensor_info.sensor);
        err = sensor_create_listener(sensor_info.sensor, &sensor_info.sensor_lis
tener);
        sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sens
or_value, ad);
        sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() 是 Orientation 传感器的事件回调函数。新传感器值将显示在屏幕上。

在第 2 个参数上排列传入数据。在 values[0] 中保存 Azimuth 数据，在 values[1] 中保存 Pitch 数据，在 values[2] 中保存 Roll 数据。

start_orientation_sensor() 是调用 Orientation 传感器以及定义事件回调函数的函数。

sensor_get_default_sensor(sensor_type_e, sensor_h *) 是返回传感器对象的 API。在第 1 个参数上传入 SENSOR_ORIENTATION，第 2 个参数将会返回Orientation 传感器对象。

sensor_create_listener(sensor_h, sensor_listener_h *) 是创建事件侦听器的 API。在第 1 个参数上传入传感器对象，第 2 个参数将会返回侦听器对象。

sensor_listener_set_event_cb(sensor_listener_h, unsigned int, sensor_event_cb, void *) 是在侦听器中定义回调函数的 API. 参数依次为事件侦听器、时间间隔（单位为毫秒）、回调函数名称、用户数据。

sensor_listener_start(sensor_listener_h) 是启动侦听器的 API。

运行应用程序后将会自动启动事件侦听器。在 create_base_gui() 函数末尾调用上述函数。

```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        show_is_supported(ad);
        start_orientation_sensor(ad);
}
```

在这个状态下测试，旋转手机时将转换为 Landscape 模式。屏幕方向将采用 Portrait 模式固定。位于 create_base_gui() 函数上部的以下代码支持 4 个方向。

```
    int rots[4] = { 0, 90, 180, 270 };
    elm_win_wm_rotation_available_rotations_set(ad->win, (const int *)(&rots),
4);
```

将代码修改如下。

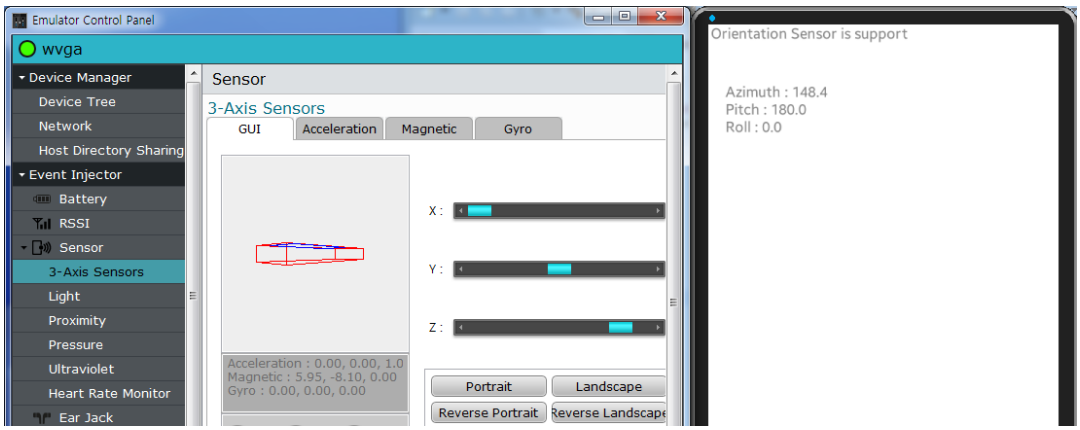
```
    int rots[1] = { 0 };
    elm_win_wm_rotation_available_rotations_set(ad->win, (const int *)(&rots),
1);
```

让我们再运行一次示例。在智能手机上进行测试时，将手机旋转即可。在模拟器中测试时，使用 Control Panel 即可。

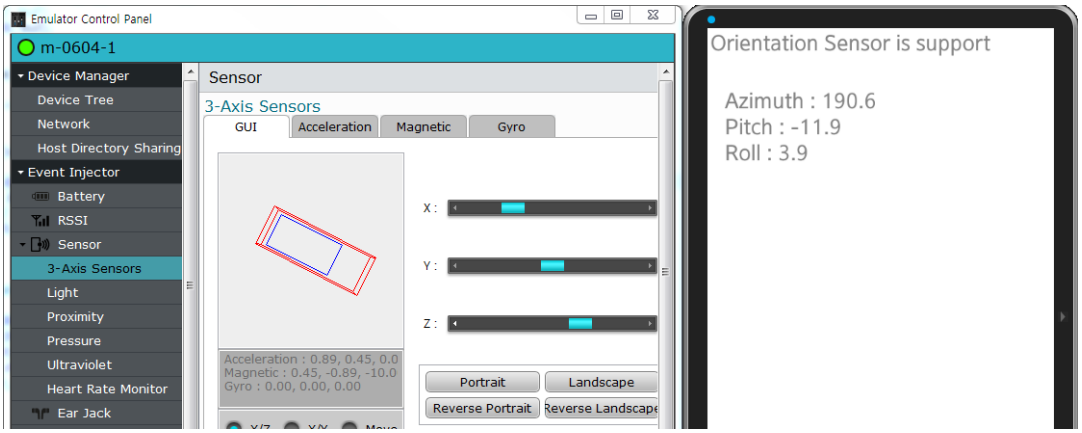
右键点击模拟器，在快捷菜单中选择 Control Panel。

出现 Control Panel 后，在左侧的树形目录中选择 [Event Injector > 3-Axis Sensors]，然后在屏幕右侧选项卡按键中选择 GUI。

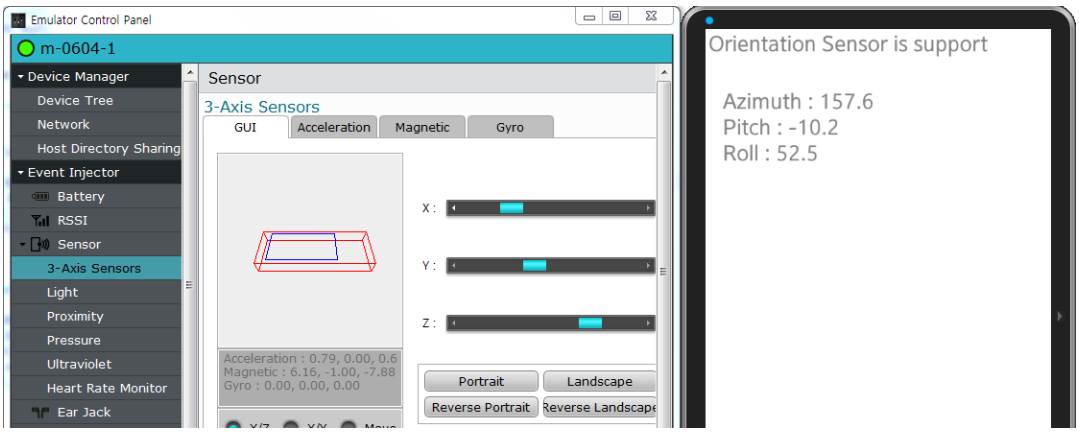
第 1 步首先测试 Azimuth。点击 Portrait 键，将 X 轴滑块移动至左部底端后，手机将平躺在地面上。在这个状态下，试着左右滑动 Z 轴滑块。应用程序屏幕上的 Azimuth 数值将在 0~360 范围内变动。



第 2 步测试 Pitch。点击 Portrait 键，在这个状态下，试着左右滑动 Z 轴滑块。应用程序屏幕上的 Azimuth 数值将在 -180~180 范围内变动。



第 3 步测试 Roll。点击 Landscape 键，在这个状态下，试着左右滑动 Z 轴滑块。应用程序屏幕上的 Roll 数值将在 -180~180 范围内变动。



3）相关 API

int sensor_is_supported(sensor_type_e type, bool *supported)：定义是否支持特定传感器的API。在第 1 个参数上传入 SENSOR_ORIENTATION，在第 2 个参数上将会显示是否支持 Orientation 传感器。

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor)：返回传感器对象的 API。在第 1 个参数上传入 SENSOR_ORIENTATION，第 2 个参数将会返回Orientation 传感器对象。

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener)：创建事件侦听器的 API。在第 1 个参数上传入传感器对象，第 2 个参数将会返回侦听器对象。

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data)：定义侦听器回调函数的 API。/ 参数：事件侦听器、时间间隔（单位为毫秒）、回调函数名称、用户数据。

int sensor_listener_start(sensor_listener_h listener)：启动侦听器的 API。

54. Magnetic 传感器使用方法

编写指南针应用程序或者测量周边磁场强度时可使用 Magnetic 传感器。可以测量 X、Y、Z 3 个轴的强度。在 Portrait 模式下将手机直立时，水平方向为 X 轴，垂直方向为 Y轴。前后方向为 Z 轴。若想在模拟器中测试 Magnetic 传感器，使用 Control Panel 即可。

1）判断是否支持 Magnetic 传感器

创建新的源项目，将 Project name 命名为 SensorMagnetic。创建源项目之后，打开 src 文件夹内的源文件 （~.c)，添加库头文件和变量。

```
#include "sensormagnetic.h"
#include <sensor.h>
#include <math.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
        Evas_Object *label2;
} appdata_s;
```

sensor.h 是各种传感器的库头文件。

math.h 是数学库头文件。

在 label0 显示是否支持 Magnetic 传感器，在 label1 中显示 3 个轴的磁场值。在 label2 中显示整体磁场值。

在 create_base_gui() 上创建 2 个新函数。

```
static void show_is_supported(appdata_s *ad)
{
        char buf[PATH_MAX];
```

```
        bool is_supported = false;
        sensor_is_supported(SENSOR_MAGNETIC, &is_supported);
        sprintf(buf, "Magnetic Sensor is %s", is_supported ? "support" : "not su
pport");
        elm_object_text_set(ad->label0, buf);
}

static void
my_box_pack(Evas_Object *box, Evas_Object *child,
        double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

show_is_supported() 是判断是否支持 Magnetic 传感器后，将结果显示在第1 个 Label 小部件上的函数。

sensor_is_supported(sensor_type_e, bool *) 是预先定义是否支持特定传感器的 API。在第 1 个参数上传入 SENSOR_MAGNETIC，在第 2 个参数上将会显示是否支持传感器。

my_box_pack() 是在 Box 上添加小部件的函数。

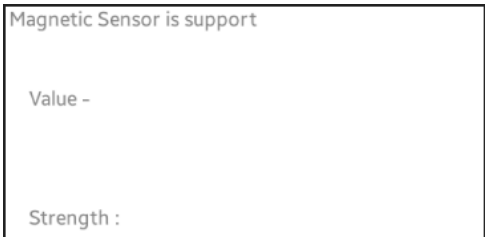
上述函数在运行应用程序时调用即可。在 create_base_gui() 函数末尾调用上述函数。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        Evas_Object * box, *btn;

        /* A box to put things in verticallly - default mode for box */
        box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        { /* child object - indent to how relationship */
            /* Label-0 */
            ad->label0 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label0, "Msg - ");
            my_box_pack(box, ad->label0, 1.0, 0.0, -1.0, 0.0);

            /* Label-1 */
            ad->label1 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label1, "Value - ");
            my_box_pack(box, ad->label1, 1.0, 0.0, -1.0, 0.0);

            /* Label-2 */
            ad->label2 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label2, "Strength : ");
            my_box_pack(box, ad->label2, 1.0, 1.0, -1.0, 0.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

```
show_is_supported(ad);
}
```

创建 3 个 Label 小部件。并且调用是否支持传感器的判断函数。

构建并运行示例。如果支持 Magnetic 传感器将会显示“Magnetic Sensor is support”字样。智能手机中可能会有不支持传感器的情况。在这种情况下，请在模拟器中进行测试。



2）定义 Magnetic 传感器事件

下面来展示一下在周边磁场改变时，定义相应事件并将数值显示在屏幕上的功能。在源文件上端添加传感器相关结构和全局变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
        Evas_Object *label2;
} appdata_s;

typedef struct _sensor_info
{
        sensor_h sensor;                /**< Sensor handle */
        sensor_listener_h sensor_listener;
} sensorinfo;

static sensorinfo sensor_info;
```

Sensorinfo 是包含传感器对象和事件侦听器变量的结构。

sensor_info 是 sensorinfo 结构的全局变量。

定义传感器事件始于侦听器。利用传感器对象和事件侦听器来定义 Magnetic 传感器事件。在 create_base_gui() 函数上端创建 2 个新函数。

```
static void
_new_sensor_value(sensor_h sensor, sensor_event_s *sensor_data, void *user_data)
{
        if( sensor_data->value_count < 3 )
                return;
        char buf[PATH_MAX];
        appdata_s *ad = (appdata_s*)user_data;

        sprintf(buf, "X : %0.1f / Y : %0.1f / Z : %0.1f",
                        sensor_data->values[0], sensor_data->values[1], sensor_d
ata->values[2]);
        elm_object_text_set(ad->label1, buf);
}

static void
start_magnetic_sensor(appdata_s *ad)
{
        sensor_error_e err = SENSOR_ERROR_NONE;
        sensor_get_default_sensor(SENSOR_MAGNETIC, &sensor_info.sensor);
        err = sensor_create_listener(sensor_info.sensor, &sensor_info.sensor_lis
tener);
        sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sens
or_value, ad);
        sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() 是 Magnetic 传感器的事件回调函数。新传感器值将显示在屏幕上。

在第 2 个参数上排列传入数据 。在 values[0] 中保存 x 轴方向数据，在 values[1] 中保存 y 轴数据，在 values[2] 中保存 z 轴数据。

start_magnetic_sensor() 是调用 Magnetic 传感器以及定义事件回调函数的函数。

sensor_get_default_sensor(sensor_type_e, sensor_h *) 是返回传感器对象的 API。在第 1 个参数上传入 SENSOR_MAGNETIC，第 2 个参数将会返回 Magnetic 传感器对象。

sensor_create_listener(sensor_h, sensor_listener_h *) 是创建事件侦听器的 API。在第 1 个参数上传入传感器对象，第 2 个参数将会返回侦听器对象。

sensor_listener_set_event_cb(sensor_listener_h, unsigned int, sensor_event_cb, void *) 是在侦听器中定义回调函数的 API. 参数依次为事件侦听器、时间间隔（单位为毫秒）、回调函数名称、用户数据。

sensor_listener_start(sensor_listener_h) 是启动侦听器的 API。

运行应用程序后将会自动启动事件侦听器。在 create_base_gui() 函数末尾调用上述函数。

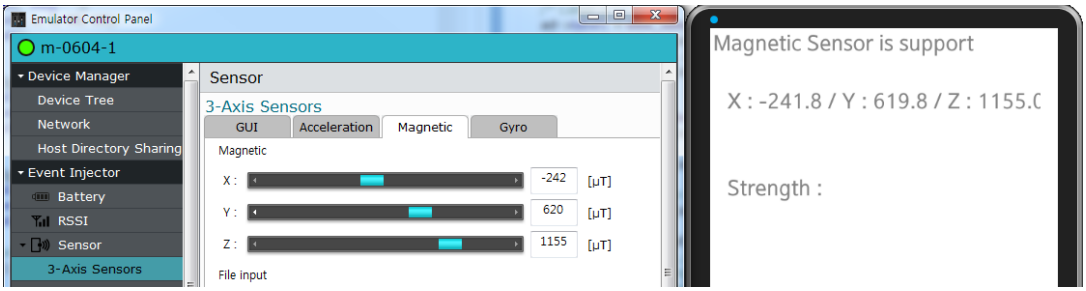
```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        show_is_supported(ad);
        start_magnetic_sensor(ad);
}
```

让我们再运行一次示例。在智能手机中测试时，靠近磁石或者将手机平放旋转即可。在模拟器中测试时，使用 Control Panel 即可。

右键点击模拟器，在快捷菜单中选择 Control Panel。

出现 Control Panel 后，在左侧的树形目录中选择 [Event Injector > 3-Axis Sensors]，然后在屏幕右侧选项卡按键中选择 Magnetic。

滑动 Control Panel 右侧屏幕的 3 个滑块，应用程序屏幕的第 2 个Label 小部件数值会发生相应变化。



3）定义整体磁场数值

下面了解一下在 X、Y、Z 轴上综合整体定义磁场数值的方法。若想求得磁场数值只要将 3 者的值平方和，然后求得平方根即可。在 _new_sensor_value() 函数上添加新函数，修改 _new_sensor_value() 函数的代码。

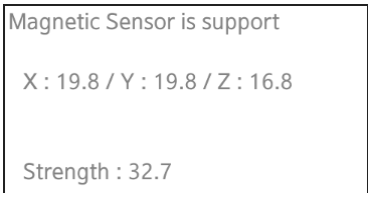
```
static float
_magnetic_strength_get(const float *values)
{
        float sum = 0.0;
        for(int i=0; i < 3; i++)
                sum += values[i] * values[i];

        return sqrt(sum);
}

static void
_new_sensor_value(sensor_h sensor, sensor_event_s *sensor_data, void *user_data)
{
        if( sensor_data->value_count < 3 )
                return;
        char buf[PATH_MAX];
        appdata_s *ad = (appdata_s*)user_data;

        sprintf(buf, "X : %0.1f / Y : %0.1f / Z : %0.1f",
                        sensor_data->values[0], sensor_data->values[1], sensor_d
ata->values[2]);
        elm_object_text_set(ad->label1, buf);

        float strength = _magnetic_strength_get(sensor_data->values);
        sprintf(buf, "Strength : %0.1f", strength);
        elm_object_text_set(ad->label2, buf);
```

```
}
```

_magnetic_strength_get(const float *) 是将排列中定义的 3 种数值求得平方和之后开平方根的函数。

sqrt(double) 是求平方根的数学 API。

让我们再运行一次示例。在第 3 个 Label 中显示整体磁场数值计算结果。



4）相关 API

int sensor_is_supported(sensor_type_e type, bool *supported)：定义是否支持特定传感器的 API。在第 1 个参数上传入 SSENSOR_MAGNETIC，在第 2 个参数上将会显示是否支持 Magnetic 传感器。

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor)：返回传感器对象的 API。在第 1 个参数上传入 SENSOR_MAGNETIC，第 2 个参数将会返回 Magnetic 传感器对象。

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener)：创建事件侦听器的 API。在第 1 个参数上传入传感器对象，第 2 个参数将会返回侦听器对象。

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data)：定义侦听器回调函数的 API。/ 参数：事件侦听器、时间间隔（单位为毫秒）、回调函数名称、用户数据。

int sensor_listener_start(sensor_listener_h listener)：启动侦听器的 API。

double sqrt(double)：求平方根的数学 API。

55. Proximity 传感器使用方法

来电时将手机贴近脸部，手机屏幕将会自动关闭。这就是 Proximity 传感器的作用机理。若想在模拟器中测试 Proximity 传感器，使用 Control Panel 即可。

1）判断是否支持 Proximity 传感器

创建新的源项目，将 Project name 命名为 SensorProximity。创建源项目之后，打开 src 文件夹内的源文件 （~.c），添加库头文件和变量。

```
#include "sensorproximity.h"
#include <sensor.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
} appdata_s;
```

sensor.h 是各种传感器的库头文件。

在 label0 中显示是否支持 Proximity 传感器，在 label1 中显示距离数值。

在 create_base_gui() 上创建 2 个新函数。

```
static void
show_is_supported(appdata_s *ad)
{
        char buf[PATH_MAX];
        bool is_supported = false;
        sensor_is_supported(SENSOR_PROXIMITY, &is_supported);
        sprintf(buf, "Proximity Sensor is %s", is_supported ? "support" : "not s
```

```
upport");
        elm_object_text_set(ad->label0, buf);
}

static void
my_box_pack(Evas_Object *box, Evas_Object *child,
        double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

show_is_supported() 是判断是否支持 Proximity 传感器后，将结果显示在第 1 个 Label 小部件上的函数。

sensor_is_supported(sensor_type_e, bool *) 是预先定义是否支持特定传感器的 API。在第 1 个参数上传入 SENSOR_PROXIMITY，在第 2 个参数上显示是否支持传感器。

my_box_pack() 是在 Box 上添加小部件的函数。

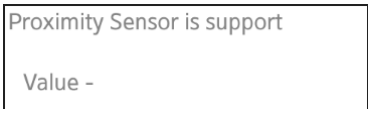
show_is_supported() 在运行应用程序时调用即可。在 create_base_gui()

函数末尾调用上述函数。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        Evas_Object * box, *btn;

        /* A box to put things in verticallly - default mode for box */
        box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        { /* child object - indent to how relationship */
            /* Label-0 */
            ad->label0 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label0, "Msg - ");
            my_box_pack(box, ad->label0, 1.0, 0.0, -1.0, 0.0);

            /* Label-1 */
            ad->label1 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label1, "Value - ");
            my_box_pack(box, ad->label1, 1.0, 1.0, -1.0, 0.0);

        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    show_is_supported(ad);
}
```

创建 Box 和 2 个 Label 小部件。并且已调用是否支持传感器的判断函数。

构建并运行示例。如果支持 Proximity 传感器将会显示“Proximity Sensor is support”字样。智能手机中可能会有不支持传感器的情况。在这种情况下，请在模拟器中进行测试。



2）定义 Proximity 传感器事件

下面来展示一下 Proximity 传感器感知物体后，定义相应事件并将数值显示在屏幕上的功能。在源文件上端添加传感器相关结构和全局变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
} appdata_s;

typedef struct _sensor_info
{
        sensor_h sensor;                /**< Sensor handle */
        sensor_listener_h sensor_listener;
} sensorinfo;

static sensorinfo sensor_info;
```

Sensorinfo 是包含传感器对象和事件侦听器变量的结构。

sensor_info 是 sensorinfo 结构的全局变量。

定义传感器事件就是调用侦听器。利用传感器对象和事件侦听器来定义 Proximity 传感器事件。在 create_base_gui() 函数上端创建 2 个新函数。

```
static void
_new_sensor_value(sensor_h sensor, sensor_event_s *sensor_data, void *user_data)
```

```
{
        if( sensor_data->value_count < 1 )
                return;
        char buf[PATH_MAX];
        appdata_s *ad = (appdata_s*)user_data;

        sprintf(buf, "Distance : %0.1f", sensor_data->values[0]);
        elm_object_text_set(ad->label1, buf);
}

static void
start_proximity_sensor(appdata_s *ad)
{
        sensor_error_e err = SENSOR_ERROR_NONE;
        sensor_get_default_sensor(SENSOR_PROXIMITY, &sensor_info.sensor);
        err = sensor_create_listener(sensor_info.sensor, &sensor_info.sensor_lis
tener);
        sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sens
or_value, ad);
        sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() 是 Proximity 传感器的事件回调函数。新传感器值将显示在屏幕上。

在第 2 个参数上传入传感器数据。在 values[0] 中保存距离数据。

start_proximity_sensor() 是调用 Proximity 传感器以及定义事件回调函数的函数。

sensor_get_default_sensor(sensor_type_e, sensor_h *) 是返回传感器对象的 API。在第 1 个参数上传入 SENSOR_PROXIMITY，第 2 个参数将会返回 Proximity 传感器对象。

sensor_create_listener(sensor_h, sensor_listener_h *) 是创建事件侦听器的 API。在第 1 个参数上传入传感器对象，第 2 个参数将会返回侦听器对象。

sensor_listener_set_event_cb(sensor_listener_h, unsigned int, sensor_event_cb, void *) 是在侦听器中定义回调函数的 API。参数依次为事件侦听器、时间间隔（单位为毫秒）、回调函数名称、用户数据。

sensor_listener_start(sensor_listener_h) 是启动侦听器的 API。

运行应用程序后将会自动启动事件侦听器。在 create_base_gui() 函数末尾调用上述函数。

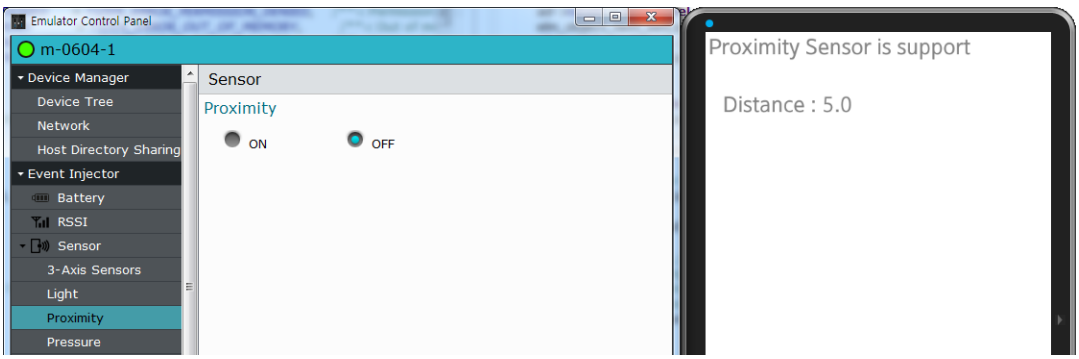
```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        show_is_supported(ad);
        start_proximity_sensor(ad);
}
```

让我们再运行一次示例。在智能手机上进行测试时，将手机贴近脸部即可。在模拟器中测试时，使用 Control Panel 即可。

右键点击模拟器，在快捷菜单中选择 Control Panel。

出现 Control Panel 后，在左侧的树形目录中选择 [Event Injector > Proximity]。

然后在 Control Panel 右侧屏幕中点击 ON 按键，在应用程序屏幕的第2个 Label 小部件中将会显示 0.0。点击 OFF 键在 Label 小部件中显示 5.0。



3）相关 API

int sensor_is_supported(sensor_type_e type, bool *supported)：定义是否支持特定传感器的 API。在第 1 个参数上传入 SENSOR_PROXIMITY，在第 2 个参数上显示是否支持 Proximity 传感器。

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor)：返回传感器对象的 API。在第 1 个参数上传入 SENSOR_PROXIMITY，第 2 个参数将会返回Proximity 传感器对象。

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener)：创建事件侦听器的 API。在第 1 个参数上传入传感器对象，第 2 个参数将会返回侦听器对象。

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data)：定义侦听器回调函数的 API。/ 参数：事件侦听器、时间间隔（单位为毫秒）、回调函数名称、用户数据。

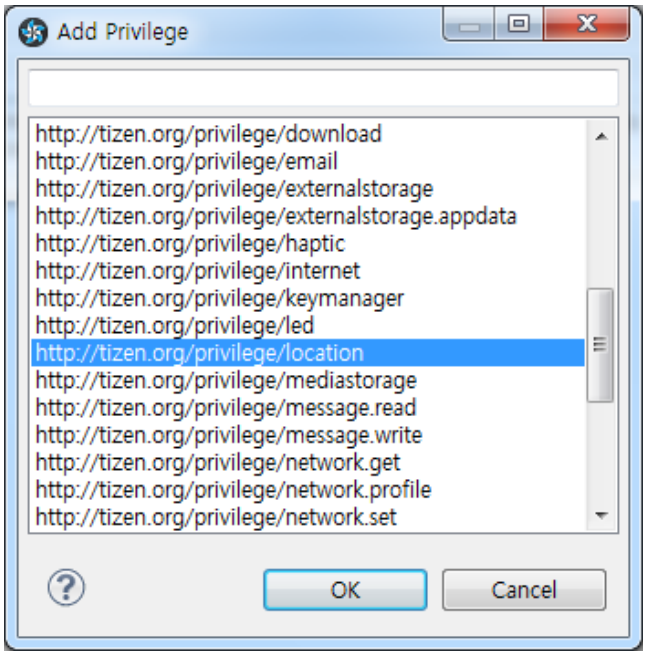
int sensor_listener_start(sensor_listener_h listener)：启动侦听器的 API。

56. GPS 传感器使用方法

若想实施地图或者导航仪应用就需要知道位置坐标。对于 Facebook、推特等 SNS 以及基于位置服务提供周边信息的应用程序来说，位置坐标必不可少。在本例中通过 Location Manager 来了解 GPS 传感器使用方法。

1）登录权限

创建新的 源项目，将 Project name 命名为 SensorGps。为使用 Location Manager，需拥有用户权限。创建源项目之后，打开 tizen-manifest.xml 文件，点击下方选项卡按键中的 Privileges。然后点击右侧上端的 Add 键。出现弹窗后，从目录中选择 http://tizen.org/privilege/location，点击 OK 键关闭弹窗。



保存后点击下方选项卡按键中，位于右侧末端的 tizen-manifest.xml 键，显示 xml 文件源代码。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.ex
```

```
ample.sensorgps" version="1.0.0">
        <profile name="mobile"/>
        <ui-application appid="org.example.sensorgps" exec="sensorgps" multiple=
"false" nodisplay="false" taskmanage="true" type="capp">
                <label>sensorgps</label>
                <icon>sensorgps.png</icon>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/location</privilege>
        </privileges>
</manifest>
```

2）检查 Location Manager 状态

下面来确认 Location Manager 是否处于可用状态。打开 src 文件夹内的源文件 （~.c），添加库头文件和变量。

```
#include "sensorgps.h"
#include <locations.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
        location_manager_h manager;
} appdata_s;
```

locations.h 是 Location Manager 的库头文件。

在 label0 中显示 Location Manager 状态，在 label1 显示位置信息。

location_manager_h 是 Location Manager 的结构。

在 create_base_gui() 上创建 2 个新函数。

```
static void
state_changed_cb(location_service_state_e state, void *user_data)
```

```
{
        appdata_s *ad = user_data;
        char buf[100];
        char *enable = (state == LOCATIONS_SERVICE_ENABLED) ? "Enable" : "Disabl
e";
        sprintf(buf, "State is %s", enable);
        elm_object_text_set(ad->label0, buf);
}

static void
show_state(appdata_s *ad)
{
        location_manager_create(LOCATIONS_METHOD_GPS, &ad->manager);
        location_manager_set_service_state_changed_cb(ad->manager, state_changed
_cb, ad);
        location_manager_start(ad->manager);
}

static void
my_box_pack(Evas_Object *box, Evas_Object *child,
        double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

state_changed_cb() 是变更 Location Manager 状态的事件回调函数。在第 1 个参数上传入状态值。状态种类如下所示。
- LOCATIONS_SERVICE_DISABLED：服务禁用状态
- LOCATIONS_SERVICE_ENABLED：服务可用状态

show_state() 是上报 Location Manager 状态变更事件的函数。

location_manager_create(location_method_e, location_manager_h*) 是创建 Location Manager 对象的 API。在第 1 个参数上传入 LOCATIONS_METHOD_GPS，第 2 个参数将会返回Location Manager 对象。位置信息收集种类如下所示。
- LOCATIONS_METHOD_GPS：使用 GPS
- LOCATIONS_METHOD_WPS：使用 WiFi
- LOCATIONS_METHOD_WPS：在 GPS 和 WiFi 中自动选择

location_manager_set_service_state_changed_cb(location_manager_h, location_service_state_changed_cb, void *) 是定义 Location Manager 状态变更事件函数名称的 API。参数依次为 Location Manager 对象、事件回调函数名称、用户数据。

location_manager_start(location_manager_h) 是启动 Location Manager 的 API。

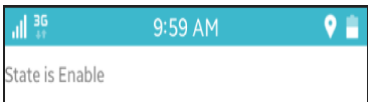
my_box_pack() 是在 Box 上添加小部件的函数。

运行应用程序后将会自动启动 Location Manager。在 create_base_gui() 函数中添加新代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        Evas_Object * box, *btn;

        /* A box to put things in verticallly - default mode for box */
        box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        { /* child object - indent to how relationship */
            /* Label-0 */
            ad->label0 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label0, "Hello EFL");
            my_box_pack(box, ad->label0, 1.0, 0.0, -1.0, 0.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    show_state(ad);
}
```

创建 Box 和 Label 小部件后，调用 Location Manager 状态变更事件上报函数。

构建并运行示例。如果在 Label 小部件中出现“State is Enable”字样，则 GPS 运转正常。



3）定义当前位置坐标

下面来展示一下通过点击 Button，定义当前经纬度坐标并将其显示在屏幕上的功能。在 create_base_gui() 函数中添加新代码。

```
        { /* child object - indent to how relationship */
            /* Label-0 */
            ad->label0 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label0, "Hello EFL");
            my_box_pack(box, ad->label0, 1.0, 0.0, -1.0, 0.0);

            /* Label-1 */
            ad->label1 = elm_label_add(ad->conform);
            elm_object_text_set(ad->label1, "Hello EFL");
            my_box_pack(box, ad->label1, 1.0, 0.0, -1.0, 0.0);

            /* Button */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Get Location");
            evas_object_smart_callback_add(btn, "clicked", btn_clicked_cb, ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, 0.0);
        }
    }
```

各添加 1 个 Label 小部件和 Button 小部件。

创建 Button 回调函数。在 create_base_gui() 函数上添加新代码。

```
static void
btn_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        double altitude, latitude, longitude, climb, direction, speed;
        double horizontal, vertical;location_accuracy_level_e level;time_t times
tamp;
```

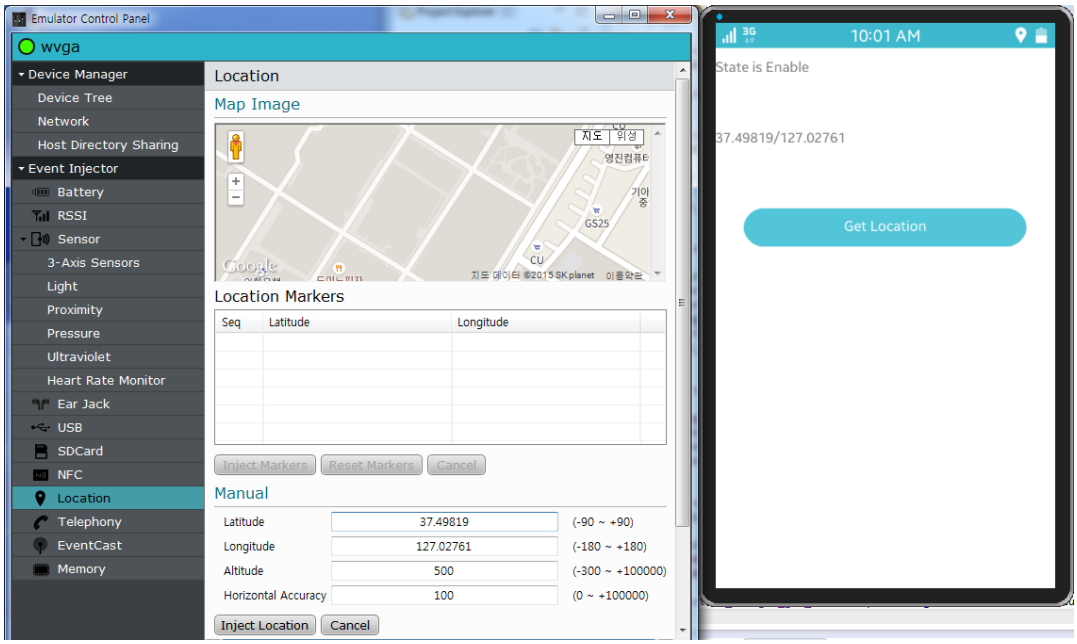
```
        location_manager_get_location(ad->manager, &altitude, &latitude, &longit
ude,
                &climb, &direction, &speed, &level, &horizontal, &vertical, &tim
estamp);
        char buf[100];
        sprintf(buf, "%0.5f/%0.5f", latitude, longitude);
        elm_object_text_set(ad->label1, buf);
}
```

location_manager_get_location(location_manager_h, double *, double *, double *, double *, double *, double *, location_accuracy_level_e *, double *, double*, time_t*) 是定义当前位置信息的 API。参数依次为 Location Manager 对象、高度、维度、经度、垂直移动速度、方向、水平移动速度、精确度、水平精确度（单位为米）、垂直精确度（单位为米）、时间。

然后是将维度和经度显示在第 2 个 Label 小部件上的代码。

让我们再运行一次示例。若想在模拟器中测试，使用 Control Panel 即可。右键点击模拟器，在快捷菜单中选择 Control Panel。

出现 Control Panel 后，在左侧的树形目录中选择 [Event Injector > Location]。然后在右侧 Latitude 界面中输入维度坐标 （ex 37.49819），在Longitude 中输入经度坐标 （ex 127.02761）。在 Altitude 中输入 500，在 Horizontal Accuracy 中输入 100 左右。然后点击 Inject Location 键。在应用程序屏幕点击 Button，在 Control Panel 中输入的经纬度坐标将显示在第 2 个 Label 小部件上。



4）定义位置移动事件

用户位置变更时，将自动在屏幕上显示新的经纬度坐标。在 create_base_gui() 函数末尾添加一行新代码。

```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        show_state(ad);
        location_manager_set_position_updated_cb(ad->manager, position_updated_cb, 2, ad);
}
```

location_manager_set_position_updated_cb(location_manager_h, location_position_updated_cb, int, void *) 是上报位置信息变更事件的 API。参数依次为 Location Manager 对象、事件回调函数名称、时间间隔、用户数据。

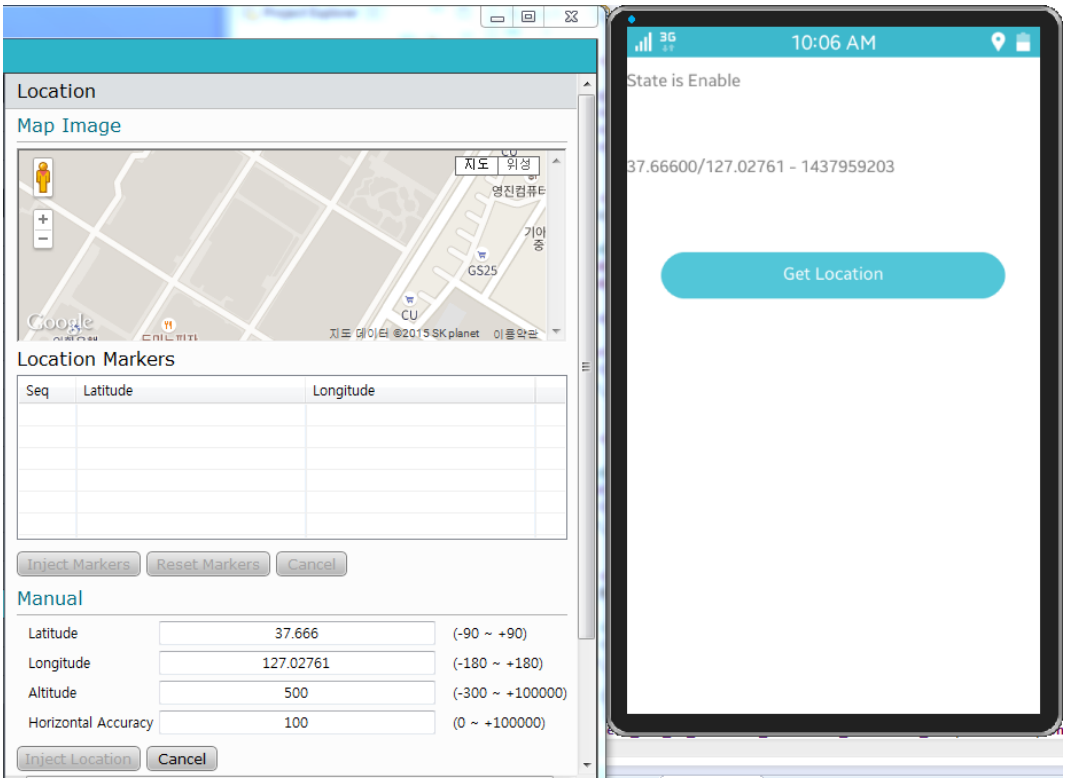
最后该创建回调函数了。在create_base_gui() 函数上添加新函数。

```
static void position_updated_cb(double latitude, double longitude, double altitude, time_t timestamp, void *user_data)
{
        appdata_s *ad = user_data;
        char buf[100];
        sprintf(buf, "%0.5f/%0.5f - %ld", latitude, longitude, timestamp);
        elm_object_text_set(ad->label1, buf);
}
```

position_updated_cb() 是接收新的位置信息时，调用 Location Manager 的回调函数。参数依次为维度、经度、高度、时间、用户数据。

函数内的经维度坐标和时间将会显示在屏幕上。

再次运行示例。在 Control Panel 中变更 Latitude(ex 37.666) 和 Longitude(ex 127.02761) 值，点击 Inject Location 键。 应用程序屏幕上经纬度坐标将自动变更。



5）相关 API

int location_manager_create(location_method_e method, location_manager_h* manager)：创建 Location Manager 对象的 API。在第 1 个参数上传入 LOCATIONS_METHOD_GPS，第 2 个参数将会返回Location Manager 对象。位置信息收集种类如下所示。

- LOCATIONS_METHOD_GPS：使用 GPS
- LOCATIONS_METHOD_WPS：使用 WiFi
- LOCATIONS_METHOD_WPS：在 GPS 和 WiFi 中自动选择

int location_manager_set_service_state_changed_cb(location_manager_h manager, location_service_state_changed_cb callback, void *user_data)：定义 Location Manager 状态，变更事件函数名称的 API。参数依次为 Location Manager 对象、事件回调函数名称、用户数据。

int location_manager_start(location_manager_h manager)：启动Location Manager 的 API。

int location_manager_get_location(location_manager_h manager, double *altitude, double *latitude, double *longitude, double *climb, double *direction, double *speed, location_accuracy_level_e *level, double *horizontal, double *vertical, time_t *timestamp)：定义当前位置信息的 API。参数依次为 Location Manager 对象、高度、维度、经度、垂直移动速度、方向、水平移动速度、精确度、水平精确度（单位为米）、垂直精确度（单位为米）、时刻。

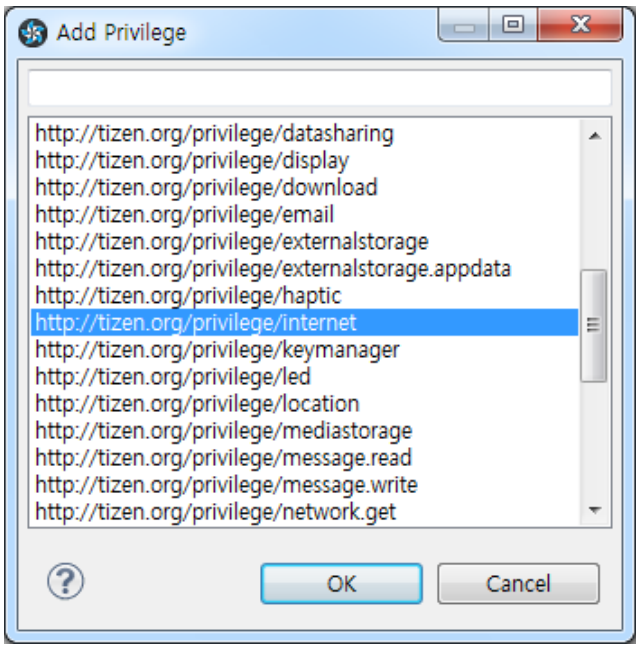
int location_manager_set_position_updated_cb(location_manager_h manager, location_position_updated_cb callback, int interval, void *user_data)：上报位置信息变更事件的 API。参数依次为 Location Manager 对象、事件回调函数名称、时间间隔、用户数据。

57. 谷歌地图库

如果要使用电子地图，使用谷歌地图即可。具体操作方法是在地图服务器中传入经纬度坐标和缩放等级值，然后将传入服务器的地图数据信息输入画布里即可。如果将示例创建硬编码，所需时间较长，所以只加载库文件进行简单展示。

1）登录权限

创建新的源项目，将 Project name 命名为 MapViewEx。为了与谷歌地图服务器通信，需拥有用户权限。创建源项目之后，打开 tizen-manifest.xml 文件，点击下方选项卡按键中的 Privileges。然后点击右侧上端的 Add 键出现弹窗后，从目录中选择 http://tizen.org/privilege/internet，点击 OK 键关闭弹窗。



保存后点击下方选项卡按键中，位于右侧末端的 tizen-manifest.xml键，显示 xml 文件源代码。

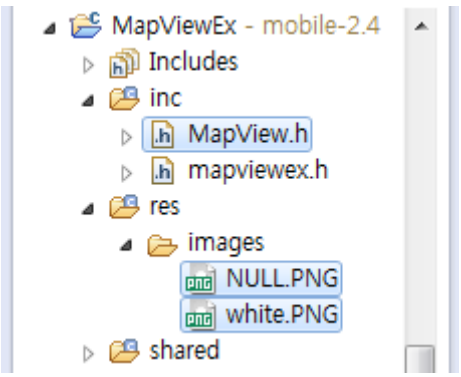
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.mapviewex" version="1.0.0">
        <profile name="mobile"/>
        <ui-application appid="org.example.mapviewex" exec="mapviewex" multiple="false" nodisplay="false" taskmanage="true" type="capp">
                <label>mapviewex</label>
                <icon>mapviewex.png</icon>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/internet</privilege>
        </privileges>
</manifest>
```

2）复制库文件

如果要硬编码再现所有电子地图功能，所需时间较长，所以只复制库文件来使用。将附录 /etc 文件夹中的 MapView.h 文件复制到源项目 /inc 文件夹里。

在地图源代码中可使用图像文件。在源项目 /res 文件夹内创建新文件夹，文件夹名称改为 images。然后，将附录 /Image 文件夹中的 2 个图像文件（NULL.PNG, white.PNG） 复制到刚刚创建的文件夹里。



3）创建 MapView 小部件

库和所需图像已复制完毕，所以用源代码编写 MapView 小部件。打开 /src 文件夹内的源文件 （~.c），添加库和经纬度坐标。

```
#include "mapviewex.h"
#include "MapView.h"

#define START_LATITUDE 40.779986
#define START_LONGITUDE -73.9615488

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
} appdata_s;
```

MapView.h 是刚刚从附录中复制的 MapView 库文件。

START_LATITUDE 和 START_LONGITUDE 是纽约曼哈顿中央公园的经纬度坐标。可以按个人喜好更换坐标。

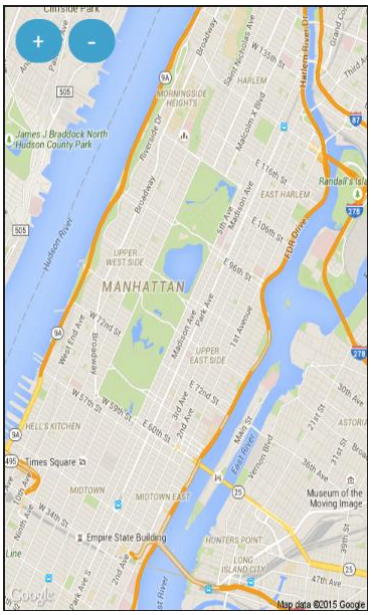
在 create_base_gui() 函数中添加新代码。将 Label 小部件按注释处理。

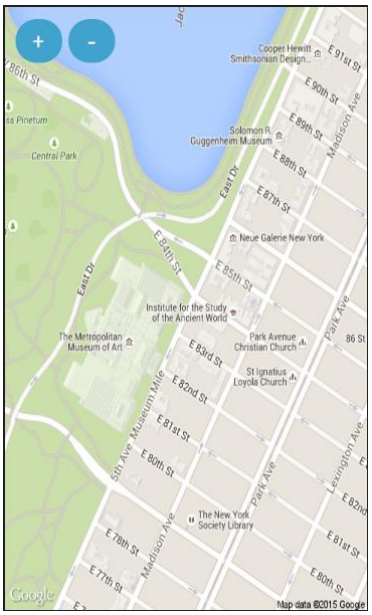
```
        /* Label*/
        /*ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "Hello EFL");
        evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT
_EXPAND);
        elm_object_content_set(ad->conform, ad->label);
        evas_object_show(ad->label);*/

        create_map(ad->conform, START_LATITUDE, START_LONGITUDE);

        /* Show window after base gui is set up */
        evas_object_show(ad->win);
```

create_map(Evas_Object *, double, double) 是创建 MapView 小部件的函数。参数依次为容器、维度坐标、经度坐标。

构建并运行示例。如果建模过程中发生错误，请打开 /inc/MapView.h 文件，将 #include <curl/curl.h> 上的代码改为以下形式。
#include <curl.h>

电子地图将正常显示，并带有 2 个按键。点击 + 键放大地图，点击 - 键缩小地图。滑动地图，地图将按滑动方向移动。





4）MapView 小部件源代码说明

下面来了解一下 MapView 库中所使用的源代码。打开从 /inc 文件夹中复制的 MapView.h 文件，移动至最下方，在create_map() 的末尾会出现如下所示的代码。

int mkdir (char *__path, __mode_t __mode)：创建新文件夹的API。

在 make_new_url() 函数中创建向谷歌地图服务器上报经纬度坐标地图数据的 URL 路径。创建的 URL 地址将保存在全局变量 place_api_url 或者 curr_url内。

arrange_start_main_page() 函数将会把经纬度坐标传入谷歌地图服务器。

map_dload_thread() 是下载地图数据的线程事件函数。

save_map_temp_file() 是将服务器接受到的地图数据，按文件形式保存的函数。

update_main_page() 是将地图数据保存于文件中并变更文件位置和大小的函数。

mouse_down_cb() 是用户点击下载地图文件时，处理事件的函数。

mouse_move_cb() 是用户点击移动地图文件时，移动地图文件的函数。

mouse_move_cb() 是用户点击解除时，处理事件的函数。

map_zoom_in() 是 + 键的回调函数。增加缩放等级，变更中心坐标，调用 arrange_start_main_page() 函数。

map_zoom_out() 是 - 键的回调函数。减小缩放等级，变更中心坐标，调用 arrange_start_main_page() 函数。

5）MapView.h 源代码

无法下载附录文件时，创建文本文件，将文件名改为 MapView.h。然后复制以下代码即可。

在 define 语法 START_URL 和 PLACE_API_URL 末尾 <Google Map Key> 部位输入自己持有的谷歌地图密钥即可。

```
/*
 * MapView.h
 *
 *   Created on: Apr 29, 2015
 *           Author: 김시찬 Samsung Electronics co
 */

#ifndef MAPVIEW_H_
#define MAPVIEW_H_

#include <app.h>
#include <Evas_GL.h>
#include <Evas_GL_GLES2_Helpers.h>
#include <Elementary.h>
#include <system_settings.h>
#include <efl_extension.h>
#include <dlog.h>
#include <curl/curl.h>
#include <math.h>
#include <dlog.h>
#include <locations.h>

#define TMP_DIR "/tmp/map_temp"
#define START_URL "http://maps.googleapis.com/maps/api/staticmap?language=korean
&zoom=99&center=unknown&size=480x800&key=<Google Map Key>"
#define PLACE_API_URL "https://maps.googleapis.com/maps/api/place/textsearch/xml?
query=where&sensor=true&key=<Google Map Key>"

#define MAX_TMP_FILENAME_LEN 256
#define MAX_URL_LEN 2048
#define MAP_TILE_X 3
#define MAP_TILE_Y 3
#define MAX_ZOOM_SCALE 19
#define MIN_ZOOM_SCALE 0
#define START_ZOOM_LEVEL 13
```

```
#define X_RESOLUTION 480
#define Y_RESOLUTION 800
#define ALL_TILE_USABLE_X 1000
#define ALL_TILE_USABLE_Y 1000
#define GEO_INFO_STR_NUM 20
#define MAP_DLOAD_Q_NUM 500
#define PLACE_SEARCHED_NUM 100
#define PLACE_INFO_MAX (100*1000)

typedef struct mapdata {
        float                       xangle;
        float                       yangle;
        Eina_Bool               mouse_down : 1;
        Eina_Bool               mouse_move : 1;
        Eina_Bool               mouse_move_update : 1;
        Eina_Bool               wait_for_update : 1;
        Ecore_Thread*       thread;
} mapdata_s;

typedef struct all_tile_info {
        double lati_of_center;
        double longti_of_center;
        char download_ok;
        char download_request;
        char file_name[MAX_TMP_FILENAME_LEN];
}all_tile_info_s;

typedef struct index_of_all_tile {
        int x;
        int y;
}index_of_all_tile_s;

typedef struct certer_tile_info {
        int x_info;
        int y_info;
}certer_tile_info_s;

typedef struct map_dload_queue {
        char url[MAX_URL_LEN];
        char filename[MAX_TMP_FILENAME_LEN];
        index_of_all_tile_s index;
}map_dload_queue_s;

typedef struct place_search_list {
        double str_lati;
        double str_longti;
```

```
} place_search_list_s;

typedef struct tile_info {
        int curr_x;
        int curr_y;
        char file_name[MAX_TMP_FILENAME_LEN];
}tile_info_s;

typedef struct MemoryStruct {
        char *memory;
        size_t size;
} MemoryStruct_s;

enum
{
        URL_CHANGED,
        URL_NOT_CHANGED,
}new_url_result;

enum
{
        ZOOM_CHANGED,
        CENTER_NAME_CHANGED,
        CENTER_GEOMETRY_CHANGED,
        ADD_NEW_MARKER,
        ADD_NEW_MULTI_MARKER,
        DELITE_MARKER,
        PLACE_INFO,
        MAX_REASON
}new_url_reason;

const tile_info_s tile_info_init_information[MAP_TILE_X][MAP_TILE_Y] = {
                {{-X_RESOLUTION,Y_RESOLUTION}, {-X_RESOLUTION,0}, {-X_RESOLUTION,
-Y_RESOLUTION}},
                {{0,Y_RESOLUTION}, {0,0}, {0,-Y_RESOLUTION}},
                {{X_RESOLUTION,Y_RESOLUTION}, {X_RESOLUTION,0}, {X_RESOLUTION,-Y
_RESOLUTION}}
                };
tile_info_s tile_info[MAP_TILE_X][MAP_TILE_Y];

//appdata_s *ad_g;
mapdata_s *m_md;
Evas *m_canvas;
Evas_Object* main_page[MAP_TILE_X][MAP_TILE_Y];
Evas_Object *m_btn1;
Evas_Object *m_btn2;
```

```
int map_dload_q_idx = 0;
char do_not_update_map = 0;
map_dload_queue_s map_dload_q[MAP_DLOAD_Q_NUM];
all_tile_info_s all_tiles[ALL_TILE_USABLE_X][ALL_TILE_USABLE_Y];
char place_api_url[MAX_URL_LEN] = PLACE_API_URL;
char curr_url[MAX_URL_LEN] = START_URL;
char search_str[MAX_REASON][20]={"zoom=", "center=", "center=", "&", "&", "&mark
ers=", "query="};
int place_searched_num;
place_search_list_s place_search_list[PLACE_SEARCHED_NUM];
location_manager_h   l_manager;
int location_initialized = 0;
double map_start_lati_of_center = 37.259606;
double map_start_longti_of_center = 127.045828;
double x_moved = 0;
double y_moved = 0;
Ecore_Timer * get_geometry_timer = NULL;
static char temp_place_info[PLACE_INFO_MAX];

Eina_Lock           set_info_mutex;
extern int map_dload_q_idx;
int current_zoom_level = START_ZOOM_LEVEL;
certer_tile_info_s current_center;
double curr_user_lati;
double curr_user_longti;
char m_icon_path[100];

void update_main_page(void);
Eina_Bool get_changed_location(void *data);

double next_lati_value(double curr_lati, int diff, int curr_zoom)
{
        double lat;

        int y1 = floor((1.0 - log(tan(curr_lati * ELM_PI / 180.0) +
                                                        (1.0 / cos(curr_lat
i * ELM_PI / 180.0)))
                                        / ELM_PI) / 2.0 * (Y_RESOLUTION*pow(2,
curr_zoom)));

        double n = ELM_PI - (2.0 * ELM_PI * (y1+2.5*diff) / (Y_RESOLUTION*pow(2,
 curr_zoom)));
        lat = 180.0 / ELM_PI *atan(0.5 * (exp(n) - exp(-n)));

        return lat;
```

```
}

double next_longti_value(double curr_longti, int diff, int curr_zoom)
{
        double lon;

        int x1 = floor((curr_longti + 180.0) / 360.0 * (X_RESOLUTION*pow(2, curr
_zoom)));

        lon = ((x1+1.875*diff) / ((double)X_RESOLUTION*pow(2, curr_zoom)) * 360.
0) - 180;

        return lon;
}

double latitude_of_polar(int polar)// polar=1 : Arctic, polar=-1 : Antarctic
{
        double n, y, lat;

        if(polar==1)
                y = 0;
        else
                y = Y_RESOLUTION;
        n = ELM_PI - (2.0 * ELM_PI * y / Y_RESOLUTION);
        lat = 180.0 / ELM_PI *atan(0.5 * (exp(n) - exp(-n)));
        return lat;
}

void delete_all_marker_from_url(char* url)
{
        char tmp_url_1[MAX_URL_LEN]={0,}, tmp_url_2[MAX_URL_LEN]={0,};
        char* next_pos = NULL;
        char* pos_and_oper = NULL;

        next_pos = strstr(url, "&markers=");

        while(next_pos!=NULL)
        {
                pos_and_oper = strstr((next_pos+1), "&");
                memset(tmp_url_1, 0, MAX_URL_LEN);
                memcpy(tmp_url_1, url, (next_pos-url));
                memset(tmp_url_2, 0, MAX_URL_LEN);
                memcpy(tmp_url_2, pos_and_oper, strlen(url)-(pos_and_oper-url));
                memset(url, 0, MAX_URL_LEN);
                snprintf(url, MAX_URL_LEN, "%s%s", tmp_url_1, tmp_url_2);
                next_pos = strstr(url, "&markers=");
```

```
        }
}

int make_new_url(int reason, int zoom, void* data, double x_changed, double y_ch
anged)
{
        char prev[MAX_URL_LEN]={0,}, new_url[MAX_URL_LEN]={0,}, tmp_str_changed
[MAX_URL_LEN]={0,}, tmp_str_changed_1[MAX_URL_LEN]={0,}, tmp_str_end[MAX_URL_LEN]
={0,};
        char *url_handle;
        char str_lati[GEO_INFO_STR_NUM]={0,}, str_longi[GEO_INFO_STR_NUM]={0,};
        char* pos = NULL;
        char* pos_and_oper = NULL;
        char* pos_comma = NULL;
        double     latitude, longitude;

        if(reason == PLACE_INFO)
        {
                url_handle = place_api_url;
        }
        else
        {
                url_handle = curr_url;
        }
        memcpy(prev, url_handle, strlen(url_handle));

        pos = strstr(prev, search_str[reason]);

        if(pos == NULL)
        {
                return URL_NOT_CHANGED;
        }

        pos_and_oper = strstr(pos, "&"); // to search first '&' from after "zoom
=..."

        if(pos_and_oper == NULL)
        {
                return URL_NOT_CHANGED;
        }

        memcpy(tmp_str_end, pos_and_oper, strlen(prev) - ((int)pos_and_oper-(int)
prev));

        switch (reason)
        {
```

```
                case ZOOM_CHANGED:
                {
                        if(zoom>0)
                        {
                                if(current_zoom_level<MAX_ZOOM_SCALE) current_zo
om_level++;
                                else return URL_NOT_CHANGED;
                        }
                        else if(zoom<0)
                        {
                                if(current_zoom_level>MIN_ZOOM_SCALE) current_zo
om_level--;
                                else return URL_NOT_CHANGED;
                        }
                        snprintf(tmp_str_changed, sizeof(tmp_str_changed), "zoom
=%d", current_zoom_level);
                }
                break;

                case CENTER_NAME_CHANGED:
                {
                        snprintf(tmp_str_changed, sizeof(tmp_str_changed), "cent
er=%s", (char*)data);
                }
                break;

                case CENTER_GEOMETRY_CHANGED:
                {
                        pos_comma = strstr(pos, ","); // to search first ',' fro
m after "center=..."

                        memcpy(str_lati, (char*)((int)pos+strlen("center=")), (i
nt)pos_comma - ((int)pos+strlen("center=")));
                        memcpy(str_longi, (char*)((int)pos_comma+1), ((int)pos_a
nd_oper-((int)pos_comma+1)) );
                        latitude = atof(str_lati);
                        longitude = atof(str_longi);

                        if(latitude>90) latitude=90;
                        if(latitude<-90) latitude=-90;

                        if(longitude>180) longitude=(-1)*(360-longitude);
                        if(longitude<-180) longitude=(longitude+360);

                        snprintf(tmp_str_changed, sizeof(tmp_str_changed), "cent
er=%f,%f", latitude, longitude);
```

```
                }
                break;

                case ADD_NEW_MARKER:
                {
                        char tmp_char = '%';
                        snprintf(tmp_str_changed, sizeof(tmp_str_changed), "&mar
kers=color:blue%c7Clabel:U%c7C%f,%f", tmp_char, tmp_char, y_changed, x_changed);
                }
                break;

                case ADD_NEW_MULTI_MARKER:
                {
                        char tmp_char = '%';
                        int i = place_searched_num;

                        while(i--)
                        {
                                snprintf(tmp_str_changed_1, sizeof(tmp_str_chang
ed_1), "%s&markers=color:red%c7Clabel:S%c7C%f,%f", tmp_str_changed, tmp_char, tm
p_char, place_search_list[i].str_lati, place_search_list[i].str_longti);
                                snprintf(tmp_str_changed, sizeof(tmp_str_change
d), "%s", tmp_str_changed_1);
                        }
                }
                break;

                case DELITE_MARKER:
                {
                        //do not process tmp_str_changed because we just need to
 set tmp_str_changed=""
                        delete_all_marker_from_url(tmp_str_end);// remove all "&
markers.." from tmp_str_end
                }
                break;

                case PLACE_INFO:
                {
                        snprintf(tmp_str_changed, sizeof(tmp_str_changed), "quer
y=%s", (char*)data);
                }
                break;

                default :
                        break;
        }
```

```
        memcpy(&new_url[0], prev, ((int)pos-(int)prev));
        memcpy(&new_url[((int)pos-(int)prev)], tmp_str_changed, strlen(tmp_str_c
hanged));
        memcpy(&new_url[((int)pos-(int)prev)+strlen(tmp_str_changed)], tmp_str_e
nd, strlen(tmp_str_end));

        memset(url_handle, 0, MAX_URL_LEN);
        memcpy(url_handle, new_url, strlen(new_url));

        return URL_CHANGED;
}

void request_map_download(int x_info, int y_info, double latitude, double longit
ude)
{
        char buf[100];

        if(all_tiles[x_info][y_info].download_request)
                return;

        all_tiles[x_info][y_info].download_request = 1;

        memset(all_tiles[x_info][y_info].file_name, 0, MAX_TMP_FILENAME_LEN);

        if(longitude>200 || longitude<-200 || latitude<latitude_of_polar(-1) ||
latitude>latitude_of_polar(1))
        {
                snprintf(all_tiles[x_info][y_info].file_name, sizeof(all_tiles[x
_info][y_info].file_name), "%s/white.PNG", m_icon_path);
                return;
        }

        all_tiles[x_info][y_info].lati_of_center = latitude;
        all_tiles[x_info][y_info].longti_of_center = longitude;
        snprintf(all_tiles[x_info][y_info].file_name, sizeof(all_tiles[x_info][y
_info].file_name), "%s/%d_%f,%f.PNG", TMP_DIR, current_zoom_level, latitude, lon
gitude);

        memset(buf, 0, 100);
        snprintf(buf, sizeof(buf), "%f,%f", latitude, longitude);

        make_new_url(CENTER_NAME_CHANGED, 0, buf, 0, 0);

        eina_lock_take(&set_info_mutex);
        memset(&map_dload_q[map_dload_q_idx].url[0], 0, MAX_URL_LEN);
```

```
        memcpy(&map_dload_q[map_dload_q_idx].url[0], curr_url, MAX_URL_LEN);
        memset(&map_dload_q[map_dload_q_idx].filename[0], 0, MAX_TMP_FILENAME_LE
N);
        memcpy(&map_dload_q[map_dload_q_idx].filename[0], all_tiles[x_info][y_in
fo].file_name, MAX_TMP_FILENAME_LEN);
        map_dload_q[map_dload_q_idx].index.x = x_info;
        map_dload_q[map_dload_q_idx].index.y = y_info;
        map_dload_q_idx++;
        eina_lock_release(&set_info_mutex);
}

int save_map_temp_file(char* url, char* file_name)
{
        CURL *curl_handle;
        FILE *currfile;
        int ret_val=0;

        curl_global_init(CURL_GLOBAL_ALL);

        /* init the curl session */
        curl_handle = curl_easy_init();

        /* set URL to get */
        curl_easy_setopt(curl_handle, CURLOPT_URL, url);

        /* no progress meter please */
        curl_easy_setopt(curl_handle, CURLOPT_FOLLOWLOCATION, 1L);

        /* open the files */
        currfile = fopen(file_name,"w");
        if (currfile == NULL) {
                curl_easy_cleanup(curl_handle);
                return -1;
        }

        /* we want the headers to this file handle */
        curl_easy_setopt(curl_handle,     CURLOPT_WRITEDATA, currfile);

        /*
         *       * Notice here that if you want the actual data sent anywhere el
se but
         *             * stdout, you should consider using the CURLOPT_WRITEDATA
 option.   */

        /* get it! */
        ret_val = curl_easy_perform(curl_handle);
```

```
        /* close the header file */
        fclose(currfile);

        /* cleanup curl stuff */
        curl_easy_cleanup(curl_handle);

        return ret_val;
}

int map_download(char* tmp_url, char* tmp_filename)
{
        if(save_map_temp_file(tmp_url, tmp_filename) != 0)
        {
                return 0;//fail
        }

        return 1;//success
}

void set_map_main_tile_info(int current_center_x, int current_center_y)
{
        for(int i=0; i<MAP_TILE_X ; i++)
        {
                for(int j=0; j<MAP_TILE_Y ; j++)
                {
                        memset(tile_info[i][j].file_name, 0, MAX_TMP_FILENAME_LE
N);

                        if(all_tiles[current_center_x-(1-i)][current_center_y-(1
-j)].download_ok)
                        {
                                memcpy(tile_info[i][j].file_name, all_tiles[curr
ent_center_x-(1-i)][current_center_y-(1-j)].file_name,
                                        strlen(all_tiles[current_center_x-(1-i)]
[current_center_y-(1-j)].file_name));
                        }
                        else
                        {
                                snprintf(tile_info[i][j].file_name, sizeof(tile_
info[i][j].file_name), "%s/NULL.PNG", m_icon_path);
                        }
                }
        }
        current_center.x_info = current_center_x;
        current_center.y_info = current_center_y;
```

```
}

void mouse_move_cb(void *data, Evas *e , Evas_Object *obj , void *event_info)
{
        Evas_Event_Mouse_Move *ev;
        ev = (Evas_Event_Mouse_Move *)event_info;
        //appdata_s *ad = data;
        mapdata_s *md = data;

        if(md->mouse_down == EINA_TRUE)
        {
                if((ev->cur.canvas.x != ev->prev.canvas.x) || (ev->cur.canvas.y !
= ev->prev.canvas.y))
                {
                        ecore_timer_del(get_geometry_timer);
                }

                // x point check
                if(ev->cur.canvas.x > ev->prev.canvas.x) //x++
                {
                        if(tile_info[1][1].curr_x > 0)
                        {
                                for(int j=0 ; j<MAP_TILE_Y ; j++)
                                {
                                        if(! all_tiles[current_center.x_info-2]
[current_center.y_info+(1-j)].download_request)
                                        {
                                                double lati_of_center = all_tile
s[current_center.x_info][current_center.y_info].lati_of_center;
                                                double longti_of_center = all_ti
les[current_center.x_info][current_center.y_info].longti_of_center;
                                                double longti_for_end_check;

                                                lati_of_center = next_lati_value
(lati_of_center, Y_RESOLUTION*(j-1), current_zoom_level);
                                                longti_of_center = next_longti_v
alue(longti_of_center, X_RESOLUTION*(-1), current_zoom_level);
                                                longti_of_center = next_longti_v
alue(longti_of_center, X_RESOLUTION*(-1), current_zoom_level);

                                                request_map_download(current_cen
ter.x_info-2, current_center.y_info+(1-j), lati_of_center, longti_of_center);
                                        }
                                }
                        }
```

```
                        for(int i=0; i<MAP_TILE_X ; i++)
                        {
                                for(int j=0; j<MAP_TILE_Y ; j++)
                                {
                                        tile_info[i][j].curr_x = tile_info[i][j].
curr_x+(ev->cur.canvas.x - ev->prev.canvas.x);
                                        evas_object_move(main_page[i][j], tile_i
nfo[i][j].curr_x, tile_info[i][j].curr_y);
                                }
                        }
                        x_moved = x_moved + (ev->cur.canvas.x - ev->prev.canvas.
x);

                        if(tile_info[1][1].curr_x > X_RESOLUTION)
                        {
                                set_map_main_tile_info(current_center.x_info-1,
current_center.y_info);
                                update_main_page();

                                for(int i=0; i<MAP_TILE_X ; i++)
                                {
                                        for(int j=0; j<MAP_TILE_Y ; j++)
                                        {
                                                evas_object_move(main_page[i][j],
 tile_info[i][j].curr_x-X_RESOLUTION, tile_info[i][j].curr_y);
                                                tile_info[i][j].curr_x = tile_in
fo[i][j].curr_x-X_RESOLUTION;
                                        }
                                }
                        }
                }
                else if(ev->cur.canvas.x < ev->prev.canvas.x) //x--
                {
                        if(tile_info[1][1].curr_x < 0)
                        {
                                for(int j=0 ; j<MAP_TILE_Y ; j++)
                                {
                                         if(! all_tiles[current_center.x_info+2]
[current_center.y_info+(1-j)].download_request)
                                        {
                                                double lati_of_center = all_tile
s[current_center.x_info][current_center.y_info].lati_of_center;
                                                double longti_of_center = all_ti
les[current_center.x_info][current_center.y_info].longti_of_center;
                                                double longti_for_end_check;
```

```
                                                lati_of_center = next_lati_value
(lati_of_center, Y_RESOLUTION*(j-1), current_zoom_level);
                                                longti_of_center = next_longti_v
alue(longti_of_center, X_RESOLUTION, current_zoom_level);
                                                longti_of_center = next_longti_v
alue(longti_of_center, X_RESOLUTION, current_zoom_level);

                                                request_map_download(current_cen
ter.x_info+2, current_center.y_info+(1-j), lati_of_center, longti_of_center);
                                        }
                                }
                        }

                        for(int i=0; i<MAP_TILE_X ; i++)
                        {
                                for(int j=0; j<MAP_TILE_Y ; j++)
                                {
                                        tile_info[i][j].curr_x = tile_info[i][j].
curr_x+(ev->cur.canvas.x - ev->prev.canvas.x);
                                        evas_object_move(main_page[i][j], tile_i
nfo[i][j].curr_x, tile_info[i][j].curr_y);
                                }
                        }
                        x_moved = x_moved + (ev->cur.canvas.x - ev->prev.canvas.
x);

                        if(tile_info[1][1].curr_x < -(X_RESOLUTION-80))
                        {
                                set_map_main_tile_info(current_center.x_info+1,
current_center.y_info);
                                update_main_page();

                                for(int i=0; i<MAP_TILE_X ; i++)
                                {
                                        for(int j=0; j<MAP_TILE_Y ; j++)
                                        {
                                                evas_object_move(main_page[i][j],
 tile_info[i][j].curr_x+X_RESOLUTION, tile_info[i][j].curr_y);
                                                tile_info[i][j].curr_x = tile_in
fo[i][j].curr_x+X_RESOLUTION;
                                        }
                                }
                        }
                }

                // y point check
```

```
                if(ev->cur.canvas.y > ev->prev.canvas.y) // y++
                {
                        if(tile_info[1][1].curr_y > 0)
                        {
                                for(int i=0 ; i<MAP_TILE_Y ; i++)
                                {
                                        if(! all_tiles[current_center.x_info-(1-
i)][current_center.y_info+2].download_request)
                                        {
                                                double lati_of_center = all_tile
s[current_center.x_info][current_center.y_info].lati_of_center;
                                                double longti_of_center = all_ti
les[current_center.x_info][current_center.y_info].longti_of_center;

                                                lati_of_center = next_lati_value
(lati_of_center, Y_RESOLUTION*(-1), current_zoom_level);
                                                lati_of_center = next_lati_value
(lati_of_center, Y_RESOLUTION*(-1), current_zoom_level);
                                                longti_of_center = next_longti_v
alue(longti_of_center, X_RESOLUTION*(i-1), current_zoom_level);

                                                request_map_download(current_cen
ter.x_info-(1-i), current_center.y_info+2, lati_of_center, longti_of_center);
                                        }
                                }
                        }

                        for(int i=0; i<MAP_TILE_X ; i++)
                        {
                                for(int j=0; j<MAP_TILE_Y ; j++)
                                {
                                        tile_info[i][j].curr_y = tile_info[i][j].
curr_y+(ev->cur.canvas.y - ev->prev.canvas.y);
                                        evas_object_move(main_page[i][j], tile_i
nfo[i][j].curr_x, tile_info[i][j].curr_y);
                                }
                        }
                        y_moved = y_moved + (ev->cur.canvas.y - ev->prev.canvas.
y);

                        if(tile_info[1][1].curr_y > Y_RESOLUTION)
                        {
                                set_map_main_tile_info(current_center.x_info, cu
rrent_center.y_info+1);
                                update_main_page();
```

```
                                for(int i=0; i<MAP_TILE_X ; i++)
                                {
                                        for(int j=0; j<MAP_TILE_Y ; j++)
                                        {
                                                evas_object_move(main_page[i][j],
 tile_info[i][j].curr_x, tile_info[i][j].curr_y-Y_RESOLUTION);
                                                tile_info[i][j].curr_y = tile_in
fo[i][j].curr_y-Y_RESOLUTION;
                                        }
                                }
                        }
                }
                else if(ev->cur.canvas.y < ev->prev.canvas.y) // y--
                {
                        if(tile_info[1][1].curr_y < 0)
                        {
                                for(int i=0 ; i<MAP_TILE_Y ; i++)
                                {
                                        if(! all_tiles[current_center.x_info-(1-
i)][current_center.y_info-2].download_request)
                                        {
                                                double lati_of_center = all_tile
s[current_center.x_info][current_center.y_info].lati_of_center;
                                                double longti_of_center = all_ti
les[current_center.x_info][current_center.y_info].longti_of_center;

                                                lati_of_center = next_lati_value
(lati_of_center, Y_RESOLUTION, current_zoom_level);
                                                lati_of_center = next_lati_value
(lati_of_center, Y_RESOLUTION, current_zoom_level);
                                                longti_of_center = next_longti_v
alue(longti_of_center, X_RESOLUTION*(i-1), current_zoom_level);

                                                request_map_download(current_cen
ter.x_info-(1-i), current_center.y_info-2, lati_of_center, longti_of_center);
                                        }
                                }
                        }

                        for(int i=0; i<MAP_TILE_X ; i++)
                        {
                                for(int j=0; j<MAP_TILE_Y ; j++)
                                {
                                        tile_info[i][j].curr_y = tile_info[i][j].
curr_y+(ev->cur.canvas.y - ev->prev.canvas.y);
                                        evas_object_move(main_page[i][j], tile_i
```

```
nfo[i][j].curr_x, tile_info[i][j].curr_y);
                                }
                        }
                        y_moved = y_moved + (ev->cur.canvas.y - ev->prev.canvas.
y);

                        if(tile_info[1][1].curr_y < -(Y_RESOLUTION-200))
                        {
                                set_map_main_tile_info(current_center.x_info, cu
rrent_center.y_info-1);
                                update_main_page();

                                for(int i=0; i<MAP_TILE_X ; i++)
                                {
                                        for(int j=0; j<MAP_TILE_Y ; j++)
                                        {
                                                evas_object_move(main_page[i][j],
 tile_info[i][j].curr_x, tile_info[i][j].curr_y+Y_RESOLUTION);
                                                tile_info[i][j].curr_y = tile_in
fo[i][j].curr_y+Y_RESOLUTION;
                                        }
                                }
                        }
                }
        }
}

void mouse_up_cb(void *data, Evas *e , Evas_Object *obj , void *event_info)
{
        Evas_Event_Mouse_Move *ev;
        ev = (Evas_Event_Mouse_Move *)event_info;

        //appdata_s *ad = data;
        mapdata_s *md = data;
        md->mouse_down = EINA_FALSE;
}

void mouse_down_cb(void *data, Evas *e , Evas_Object *obj , void *event_info)
{
        Evas_Event_Mouse_Move *ev;
        ev = (Evas_Event_Mouse_Move *)event_info;

        //appdata_s *ad = data;
        mapdata_s *md = data;
        md->mouse_down = EINA_TRUE;
}
```

```
void update_main_page(void)
{
        for(int i=0; i<MAP_TILE_X ; i++)
        {
                for(int j=0; j<MAP_TILE_Y ; j++)
                {
                        if(main_page[i][j] != NULL)
                        {
                                evas_object_del(main_page[i][j]);
                                main_page[i][j] = NULL;
                        }

                        main_page[i][j] = evas_object_image_filled_add(m_canvas);

                        evas_object_image_file_set(main_page[i][j], tile_info[i]
[j].file_name, NULL);
                        evas_object_move(main_page[i][j], tile_info[i][j].curr_x,
 tile_info[i][j].curr_y);
                        evas_object_resize(main_page[i][j], X_RESOLUTION, Y_RESO
LUTION);
                        evas_object_show(main_page[i][j]);
                        evas_object_raise (m_btn1);
                        evas_object_raise (m_btn2);

                        evas_object_event_callback_add(main_page[i][j], EVAS_CAL
LBACK_MOUSE_MOVE, mouse_move_cb, m_md);
                        evas_object_event_callback_add(main_page[i][j], EVAS_CAL
LBACK_MOUSE_DOWN, mouse_down_cb, m_md);
                        evas_object_event_callback_add(main_page[i][j], EVAS_CAL
LBACK_MOUSE_UP, mouse_up_cb, m_md);
                }
        }
}

void map_dload_thread(void *data, Ecore_Thread *thread)
{
        int dload_success=0;

      while (1)
      {
             while(map_dload_q_idx && !do_not_update_map)
             {
                    char dload_url[MAX_URL_LEN] = {0,};
                    char dload_filename[MAX_TMP_FILENAME_LEN] = {0,};
                    index_of_all_tile_s index;
```

```
                        eina_lock_take(&set_info_mutex);
                        map_dload_q_idx--;
                        memcpy(dload_url, &map_dload_q[map_dload_q_idx].url[0], MAX
_URL_LEN);
                        memcpy(dload_filename, &map_dload_q[map_dload_q_idx].filena
me[0], MAX_TMP_FILENAME_LEN);
                        index.x = map_dload_q[map_dload_q_idx].index.x;
                        index.y = map_dload_q[map_dload_q_idx].index.y;
                        eina_lock_release(&set_info_mutex);

                        dload_success = 0;
                        do
                        {
                                dload_success = map_download(dload_url, dload_filen
ame);
                        } while(dload_success==0);

                        all_tiles[index.x][index.y].download_ok = 1;
                        set_map_main_tile_info(current_center.x_info, current_cente
r.y_info);
                        update_main_page();

                        //ecore_thread_feedback(thread, &index);
                }
        }
}

void    thread_feedback(void *data, Ecore_Thread *thread, void *msg_data)
{
         //TODO
}

void    thread_end(void *data, Ecore_Thread *thread)
{
         //TODO
}

void    thread_cancel(void *data, Ecore_Thread *thread)
{
         //TODO
}

void loc_state_changed_cb(location_service_state_e state, void *user_data)
{
         if(state == LOCATIONS_SERVICE_ENABLED)
```

```
                location_initialized = 1;
}

int make_start_url(double lati, double longti)
{
        char buf[GEO_INFO_STR_NUM];
        map_start_lati_of_center = lati;
        map_start_longti_of_center = longti;

        if(make_new_url(ZOOM_CHANGED, 0, NULL, 0, 0) != URL_CHANGED)
                return URL_NOT_CHANGED;

        memset(buf, 0, GEO_INFO_STR_NUM);
        snprintf(buf, sizeof(buf), "%f,%f", map_start_lati_of_center, map_start_
longti_of_center);
        if(make_new_url(CENTER_NAME_CHANGED, 0, buf, 0, 0) != URL_CHANGED)
                return URL_NOT_CHANGED;

        return URL_CHANGED;
}

void arrange_start_main_page(void)
{
        map_dload_q_idx = 0;
        do_not_update_map = 1;

        x_moved = 0;
        y_moved = 0;

        current_center.x_info = ALL_TILE_USABLE_X/2;
        current_center.y_info = ALL_TILE_USABLE_Y/2;

        for(int i=0; i<ALL_TILE_USABLE_X ; i++)
        {
                for(int j=0; j<ALL_TILE_USABLE_Y ; j++)
                {
                        all_tiles[i][j].download_ok = 0;
                        all_tiles[i][j].download_request = 0;
                        memset(all_tiles[i][j].file_name, 0, MAX_TMP_FILENAME_LE
N);
                }
        }

        for(int i=0; i<MAP_TILE_X ; i++)
        {
                for(int j=0; j<MAP_TILE_Y ; j++)
```

```
                {
                        tile_info[i][j].curr_x = tile_info_init_information[i]
[j].curr_x;
                        tile_info[i][j].curr_y = tile_info_init_information[i]
[j].curr_y;

                       if(i==1 && j==1) continue;
                       request_map_download(current_center.x_info-(1-i), curren
t_center.y_info-(1-j),
                                            next_lati_value(map_start_lati_of_center,
 Y_RESOLUTION*(1-j), current_zoom_level),
                                            next_longti_value(map_start_longti_of_ce
nter, X_RESOLUTION*(i-1), current_zoom_level));
                }
        }
        //download ceter map first for visual effect
        request_map_download(current_center.x_info, current_center.y_info, map_s
tart_lati_of_center, map_start_longti_of_center);
        do_not_update_map = 0;

        set_map_main_tile_info(current_center.x_info, current_center.y_info);
}

void map_zoom_in()
{
        if(make_new_url(ZOOM_CHANGED, 1, NULL, 0, 0) == URL_CHANGED)
        {
                map_start_lati_of_center = next_lati_value(map_start_lati_of_cen
ter, y_moved*(-1), current_zoom_level-1);
                map_start_longti_of_center = next_longti_value(map_start_longti_
of_center, -x_moved, current_zoom_level-1);

                arrange_start_main_page();
        }
}

void map_zoom_out()
{
        if(make_new_url(ZOOM_CHANGED, -1, NULL, 0, 0) == URL_CHANGED)
        {
                map_start_lati_of_center = next_lati_value(map_start_lati_of_cen
ter, y_moved*(-1), current_zoom_level+1);
                map_start_longti_of_center = next_longti_value(map_start_longti_
of_center, -x_moved, current_zoom_level+1);

                arrange_start_main_page();
```

```
        }
}

static size_t
WriteMemoryCallback(void *contents, size_t size, size_t nmemb, void *userp)
{
        size_t realsize = size * nmemb;
        struct MemoryStruct *mem = (struct MemoryStruct *)userp;

        mem->memory = realloc(mem->memory, mem->size + realsize + 1);
        if(mem->memory == NULL) {
                /* out of memory! */
                printf("not enough memory (realloc returned NULL)\n");
                return 0;
        }

        memcpy(&(mem->memory[mem->size]), contents, realsize);
        mem->size += realsize;
        mem->memory[mem->size] = 0;

        return realsize;
}

int get_geometry_info_of_place(char* url, double* start_latitude, double* start_
longitude)
{
        CURL *curl_handle;
        MemoryStruct_s chunk;
        int ret_val=0;
        char* search_index = NULL;
        char* pos = NULL;
        char* pos_lati_start = NULL;
        char* pos_lati_end = NULL;
        char* pos_longti_start = NULL;
        char* pos_longti_end = NULL;
        char tmp_str_lati[GEO_INFO_STR_NUM]={0,}, tmp_str_longti[GEO_INFO_STR_NU
M]={0,};
        double tmp_lati_sum=0, tmp_longti_sum=0;

        chunk.memory = malloc(1);   /* will be grown as needed by the realloc ab
ove */
        chunk.size = 0;       /* no data at this point */

        curl_global_init(CURL_GLOBAL_ALL);

        /* init the curl session */
```

```
        curl_handle = curl_easy_init();

        /* set URL to get */
        curl_easy_setopt(curl_handle, CURLOPT_URL, url);
        curl_easy_setopt(curl_handle, CURLOPT_SSL_VERIFYPEER, 0L);

        /* no progress meter please */
        //curl_easy_setopt(curl_handle, CURLOPT_FOLLOWLOCATION, 1L);

        /* send all data to this function  */
        curl_easy_setopt(curl_handle, CURLOPT_WRITEFUNCTION, WriteMemoryCallbac
k);

        /* we want the headers to this file handle */
        curl_easy_setopt(curl_handle,     CURLOPT_WRITEDATA, (void *)&chunk);

        /* some servers don't like requests that are made without a user-agent
                 field, so we provide one */
        curl_easy_setopt(curl_handle, CURLOPT_USERAGENT, "libcurl-agent/1.0");

        /*
         *       * Notice here that if you want the actual data sent anywhere el
se but
         *             * stdout, you should consider using the CURLOPT_WRITEDATA
 option.   */

        /* get it! */
        ret_val = curl_easy_perform(curl_handle);

        memset(temp_place_info, 0, PLACE_INFO_MAX);
        memcpy(temp_place_info, chunk.memory, chunk.size);

        /* cleanup curl stuff */
        curl_easy_cleanup(curl_handle);

        if(chunk.memory)
               free(chunk.memory);

           /* we're done with libcurl, so clean it up */
        curl_global_cleanup();

        place_searched_num = 0;
        search_index = temp_place_info;
        while(1)
        {
                pos = strstr(search_index, "<location>");
```

```
                if(pos != NULL)// detected one more
                {
                        pos_lati_start = strstr((char*)pos, "<lat>");
                        pos_lati_end = strstr((char*)pos, "</lat>");
                        pos_longti_start = strstr((char*)pos, "<lng>");
                        pos_longti_end = strstr((char*)pos, "</lng>");

                        memset(tmp_str_lati, 0, GEO_INFO_STR_NUM);
                        memcpy(tmp_str_lati, (char*)(pos_lati_start+5), pos_lati
_end-(pos_lati_start+5));
                        place_search_list[place_searched_num].str_lati = atof(tm
p_str_lati);

                        memset(tmp_str_longti, 0, GEO_INFO_STR_NUM);
                        memcpy(tmp_str_longti, (char*)(pos_longti_start+5), pos_
longti_end-(pos_longti_start+5));
                        place_search_list[place_searched_num].str_longti = atof
(tmp_str_longti);

                        place_searched_num++;
                        search_index = pos+1;//just for next search
                }
                else
                {
                        break; //couldn't find anymore.
                }
        }

        for(int i=0 ; i<place_searched_num ; i++)
        {
                tmp_lati_sum = tmp_lati_sum + place_search_list[i].str_lati;
                tmp_longti_sum = tmp_longti_sum + place_search_list[i].str_longt
i;
        }

        *start_latitude = place_search_list[0].str_lati;
        *start_longitude = place_search_list[0].str_longti;

        return ret_val;
}

//handle key event for search
void key_down_cb(void *data, Evas *evas, Evas_Object *obj, void *event_info)
{
        const int MAX_CUR = 20;
        static char buf[50];
```

```
        static int cur = 0;
        double latitude, longitude;
        static char geometry_buf[50]={0,};

        Evas_Event_Key_Down *ev = event_info;
        Evas_Object *input = data;
        char tmp[50];

        if (cur == 0) snprintf(buf, sizeof(buf), "\0");

        if (!strcmp(ev->keyname, "Return"))
        {
                evas_object_text_text_set(input, "");
                cur = 0;
                for(int i =0 ; i<strlen(buf) ; i++)
                {
                        if(buf[i]==' ') buf[i]=','; // google api can't recogniz
e ' '
                }
                if(make_new_url(PLACE_INFO, 0, buf, 0, 0) == URL_CHANGED)//make
url for query of place info.
                {
                        if(!get_geometry_info_of_place(place_api_url, &latitude,
 &longitude)) // get place info data using query url and parse it.
                        {
                                memset(geometry_buf, 0, 50);
                                snprintf(geometry_buf, sizeof(geometry_buf), "%
f,%f", latitude, longitude);
                                make_new_url(DELITE_MARKER, 0, NULL, 0, 0); // d
elete all previous markers
                                if((make_new_url(CENTER_NAME_CHANGED, 0, geometr
y_buf, 0, 0) == URL_CHANGED)
                                && (make_new_url(ADD_NEW_MULTI_MARKER, 0, NULL,
0, 0) == URL_CHANGED))
                                {
                                        map_start_lati_of_center = latitude;
                                        map_start_longti_of_center = longitude;
                                        arrange_start_main_page();
                                        update_main_page();
                                }
                        }
                }
                return;
        }

        if (!strcmp(ev->keyname, "BackSpace"))
```

```
        {
                snprintf(tmp, strlen(buf), "%s", buf);
                evas_object_text_text_set(input, tmp);
                strcpy(buf, tmp);
                cur--;
                return;
        }

        if (cur >= MAX_CUR) return;

        if(!strcmp(ev->keyname, "space"))
        {
                snprintf(tmp, sizeof(tmp), "%s%s", buf, " "); // replace ' ' to
' '
                evas_object_text_text_set(input, tmp);
                cur++;
                strcpy(buf, tmp);
        }
        else if(!strcmp(ev->keyname, "comma"))
        {
                snprintf(tmp, sizeof(tmp), "%s%s", buf, ",");
                evas_object_text_text_set(input, tmp);
                cur++;
                strcpy(buf, tmp);
        }
        else if((!strcmp(ev->keyname, "Caps_Lock"))
                        || !strcmp(ev->keyname, "Shift_L")
                        || !strcmp(ev->keyname, "Num_Lock")
                        || !strcmp(ev->keyname, "Left")
                        || !strcmp(ev->keyname, "Right")
                        || !strcmp(ev->keyname, "Up")
                        || !strcmp(ev->keyname, "Down"))
        {
                // to be ignored...
        }
        else
        {
                snprintf(tmp, sizeof(tmp), "%s%s", buf, ev->keyname);
                evas_object_text_text_set(input, tmp);
                cur++;
                strcpy(buf, tmp);
        }
}

void go_to_current_geometry(void)
{
```

```
        double altitude, climb, direction, speed;
        double horizontal, vertical;location_accuracy_level_e level;time_t times
tamp;
        char buf[100];

        if(!location_initialized)
                return;

        location_manager_get_last_location(l_manager, &altitude, &curr_user_lati,
 &curr_user_longti,

                 &climb, &direction, &speed, &level, &horizontal, &vertical, &ti
mestamp);

        memset(buf, 0, 100);
        snprintf(buf, sizeof(buf), "%f,%f", curr_user_lati, curr_user_longti);
        if(make_new_url(CENTER_NAME_CHANGED, 0, buf, 0, 0) != URL_CHANGED)
                return;
        make_new_url(DELITE_MARKER, 0, NULL, 0, 0);
        if(make_new_url(ADD_NEW_MARKER, 0, buf, curr_user_longti, curr_user_lati)
 != URL_CHANGED)
                return;

        map_start_lati_of_center = curr_user_lati;
        map_start_longti_of_center = curr_user_longti;

        arrange_start_main_page();

        get_geometry_timer = ecore_timer_add(1, get_changed_location, NULL);
}

Eina_Bool get_changed_location(void *data)
{
        double altitude, latitude, longtitude, climb, direction, speed;
        double horizontal, vertical;location_accuracy_level_e level;time_t times
tamp;

        if(!location_initialized)
                return ECORE_CALLBACK_RENEW;

        location_manager_get_last_location(l_manager, &altitude, &latitude, &lon
gtitude,

                  &climb, &direction, &speed, &level, &horizontal, &vertical, &t
imestamp);
```

```
        if((latitude != curr_user_lati) || (longtitude != curr_user_longti))
        {
                ecore_timer_del(get_geometry_timer);
                go_to_current_geometry();
                return ECORE_CALLBACK_CANCEL;
        }
        else
        {
                return ECORE_CALLBACK_RENEW;
        }
}

//go to current user's point
void current_btn_cb(void *data, Evas *e , Evas_Object *obj , void *event_info)
{
        go_to_current_geometry();
}
static void
btn_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
        int btn_num = (int)data;
        //dlog_print(DLOG_ERROR, "tag", "clicked event on Button:%d", btn_num);

        switch( btn_num ) {
        case 1 :
                map_zoom_in();
                break;
        case 2 :
                map_zoom_out();
                break;
        }
}

void create_map(Evas_Object *win, double lati, double longti)
{
        m_md = (mapdata_s*)malloc( sizeof( mapdata_s) );

        /* Button-1 */
        m_btn1 = elm_button_add(win);
        elm_object_text_set(m_btn1, "+");
        evas_object_move(m_btn1, 20, 20);
        evas_object_resize(m_btn1, 50, 50);
        evas_object_smart_callback_add(m_btn1, "clicked", btn_clicked_cb, (void
*)1);
        evas_object_show(m_btn1);
```

```
        /* Button-2 */
        m_btn2 = elm_button_add(win);
        elm_object_text_set(m_btn2, "-");
        evas_object_move(m_btn2, 90, 20);
        evas_object_resize(m_btn2, 50, 50);
        evas_object_smart_callback_add(m_btn2, "clicked", btn_clicked_cb, (void
*)2);
        evas_object_show(m_btn2);

        /* Canvas */
        m_canvas = evas_object_evas_get(win);

        char *res_path = app_get_resource_path();
        if (res_path) {
                snprintf(m_icon_path, PATH_MAX, "%s%s", res_path, "images");
                free(res_path);
        }

        /* Thread */
        ecore_thread_feedback_run(map_dload_thread, thread_feedback, thread_end,
 thread_cancel, NULL, EINA_TRUE);
        eina_lock_new(&set_info_mutex);

        /* LocationManager */
        location_manager_create(LOCATIONS_METHOD_GPS, &l_manager);
        location_manager_set_service_state_changed_cb(l_manager, loc_state_chang
ed_cb, NULL) ;
        location_manager_start(l_manager);

        mkdir(TMP_DIR, 0755);

        if( make_start_url(lati, longti) != URL_CHANGED ) //let's make start url
 to load
                return;

        arrange_start_main_page();
}


#endif /* MAPVIEW_H_ */
```

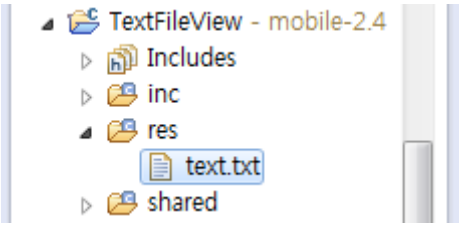
58. 读取和编写文本文件

在文件中读取并编写文本时使用 FILE 结构即可。位于 /res 文件夹的文件仅能读取，不能编写。位于 /data 文件夹的文件可读取并编写。

1）读取文本文件

下面来读取位于 /res 文件夹的文本文件展示在屏幕上。

创建新的源项目，将 Project name 命名为 TextFileView。创建源项目之后，将位于附录 /etc 文件夹内的 text.txt 文件复制到源项目 /res 文件夹里。



在 ext.txt 文件内存在如下所示的世界各国语言问候语。

```
Good morning <br/>
早上好 <br/>
Hyvää Huomenta <br/>
Bonjour <br/>
Guten Morgen <br/>
Jó reggelt kívánok <br/>
Buon giorno <br/>
おはようございます。 <br/>
안녕하세요 <br/>
Bună dimineaţa! <br/>
Buenos Días. <br/>
Günaydın <br/>
Xin chào <br/>
З д р а  в с т в у й т е <br/>
```

打开 src 文件夹内的源文件 （~.c），在 appdata 结构上添加变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        //Evas_Object *label;
        Evas_Object *entry;
} appdata_s;
```

添加 Entry 小部件变量。

在 create_base_gui() 上创建 2 个新函数。

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int col, int row, int span
x, int spany,
      double h_expand, double v_expand, double h_align, double v_align)
{
    /* Create a frame around the child, for padding */
    Evas_Object *frame = elm_frame_add(table);
    elm_object_style_set(frame, "pad_small");

    evas_object_size_hint_weight_set(frame, h_expand, v_expand);
    evas_object_size_hint_align_set(frame, h_align, v_align);

    /* place child in its box */
    {
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(frame, child);
        evas_object_show(child);
    }

    elm_table_pack(table, frame, col, row, spanx, spany);
    evas_object_show(frame);
}

static Evas_Object *
my_button_add(Evas_Object *parent, const char *text, Evas_Smart_Cb cb, void *cb_
data)
{
```

```
    Evas_Object *btn;

    btn = elm_button_add(parent);
    elm_object_text_set(btn, text);
    evas_object_smart_callback_add(btn, "clicked", cb, cb_data);

    return btn;
}
```

my_box_pack() 是在 Table 上添加小部件的函数。

my_button_add() 是创建 Button 小部件的函数。

然后移动至 create_base_gui() 函数，添加 Frame、Table、Button 和 Entry 小部件创建代码。将 Label 小部件创建代码按注释处理。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        Evas_Object *tbl, *btn, *frame;

        /* Frame */
        frame = elm_frame_add(ad->win);
        elm_object_style_set(frame, "pad_medium");
        elm_object_content_set(ad->conform, frame);
        evas_object_size_hint_weight_set(frame, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
        evas_object_size_hint_align_set(frame, EVAS_HINT_FILL, EVAS_HINT_FILL);
        evas_object_show(frame);

        /* Container: standard table */
        tbl = elm_table_add(ad->win);
        evas_object_size_hint_weight_set(tbl, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        evas_object_size_hint_align_set(tbl, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(frame, tbl);
```

```
        evas_object_show(tbl);

        {
            /* Button-1 */
            btn = my_button_add(ad->conform, "Read", btn_read_cb, ad);
            my_table_pack(tbl, btn, 0, 0, 1, 1, EVAS_HINT_EXPAND, 0.0, EVAS_HIN
T_FILL, EVAS_HINT_FILL);

            /* Entry */
            ad->entry = elm_entry_add(ad->conform);
            elm_entry_scrollable_set(ad->entry, EINA_TRUE);
            elm_object_signal_emit(ad->entry, "elm,state,scroll,enabled", "");
            elm_object_text_set(ad->entry, "Please press <b>Read</> button");
            my_table_pack(tbl, ad->entry, 0, 1, 2, 1, EVAS_HINT_EXPAND, EVAS_HI
NT_EXPAND, EVAS_HINT_FILL, EVAS_HINT_FILL);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

在 create_base_gui() 函数上添加 3 个新函数。

```
static void
app_get_resource(const char *res_file_in, char *res_path_out, int res_path_max)
{
        char *res_path = app_get_resource_path();
        if (res_path) {
                snprintf(res_path_out, res_path_max, "%s%s", res_path, res_file_
in);
                free(res_path);
        }
}

static char*
read_file(const char* filepath)
{
        FILE *fp = fopen(filepath, "r");
        if (fp == NULL)
                return NULL;
        fseek(fp, 0, SEEK_END);
        int bufsize = ftell(fp);
```

```
        rewind(fp);
        if (bufsize < 1)
                return NULL;

        char *buf = malloc(sizeof(char) * (bufsize));
        memset(buf, '\0', sizeof(buf));
        char str[200];

        while(fgets(str, 200, fp) != NULL) {
                dlog_print(DLOG_ERROR, "tag", "%s", str);
                sprintf(buf + strlen(buf), "%s", str);
        }
        fclose(fp);
        return buf;
}

static void
btn_read_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        char filepath[PATH_MAX] = { 0, };
        char *buf = NULL;
        app_get_resource("text.txt", filepath, PATH_MAX);
        buf = read_file(filepath);

        elm_object_text_set(ad->entry, buf);
}
```

app_get_resource() 是定义保存于文件夹内文件的绝对路径后返回的函数。

read_file() 是读取文本文件内容后返回的函数。

fopen (char *, char *) 是返回文件句柄的 API。在第 1 个参数上传入文件路径，在第 2 个参数上定义文件访问模式。“r”意味着只能读取，“w”意味着只能编写。

fseek (FILE *, int, int) 是将文件流移动至特定位置的 API。第 1 个参数是文件流，第 2 个是移动字节数，第 3 个是起始位置。SEEK_SET 意味着指向文件开头，SEEK_CUR 意味着指向当前位置，SEEK_END 意味着指向文件末尾。

ftell (FILE *) 是用字节数显示文件流当前位置的 API。位于末尾时显示文

件大小。

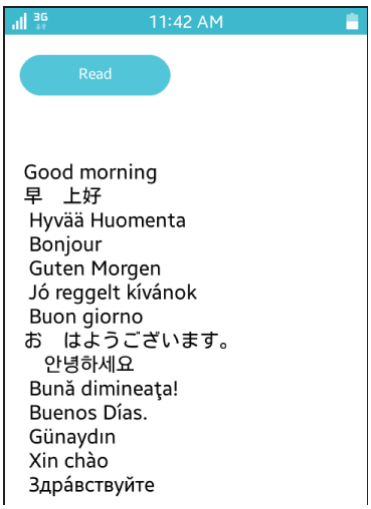
rewind (FILE *) 是退回文件流初始位置的 API。

fgets (char *, int, FILE *) 是读取文件文本数据的 API。在第 2 个参数上传入最大字符串长度，在第 3 个参数中传入文件流，第 1 个参数将会显示字符串数据。

fclose (FILE *) 是关闭文件流的 API。

btn_read_cb() 是 Button 回调函数。定义文本文件路径，读取相应文件的内容后，输入 Entry 小部件。

构建并运行示例。点击 Read 键，各国语言问候语将显示在 Entry 小部件上。



2）编写文本文件

下面来添加一个 Button，将修改后的文本以文件形式保存的功能。位于 /res 文件夹内的文件不允许编写。因此必须保存于 /data 文件夹内。在 create_base_gui() 函数中添加第 2 个 Button 创建代码。

```
        {
```

```
            /* Button-1 */
            btn = my_button_add(ad->conform, "Read", btn_read_cb, ad);
            my_table_pack(tbl, btn, 0, 0, 1, 1, EVAS_HINT_EXPAND, 0.0, EVAS_HINT
_FILL, EVAS_HINT_FILL);

            /* Button-2 */
            btn = my_button_add(ad->conform, "Write", btn_write_cb, ad);
            my_table_pack(tbl, btn, 1, 0, 1, 1, EVAS_HINT_EXPAND, 0.0, EVAS_HIN
T_FILL, EVAS_HINT_FILL);

            /* Entry */
            ad->entry = elm_entry_add(ad->conform);
            elm_entry_scrollable_set(ad->entry, EINA_TRUE);
            elm_object_signal_emit(ad->entry, "elm,state,scroll,enabled", "");
            elm_object_text_set(ad->entry, "Please press <b>Read</> button");
            my_table_pack(tbl, ad->entry, 0, 1, 2, 1, EVAS_HINT_EXPAND, EVAS_HIN
T_EXPAND, EVAS_HINT_FILL, EVAS_HINT_FILL);
        }
    }
```

为了将文本保存在文件上，在 create_base_gui() 函数上添加 3 个新函数。

```
static void
app_get_data(const char *res_file_in, char *res_path_out, int res_path_max)
{
        char *res_path = app_get_data_path();
        if (res_path) {
                snprintf(res_path_out, res_path_max, "%s%s", res_path, res_file_
in);
                free(res_path);
        }
}

static char*
write_file(const char* filepath, const char* buf)
{
        FILE *fp;
        fp = fopen(filepath, "w");
        fputs(buf, fp);
        fclose(fp);
}
```

```
static void
btn_write_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        char* buf = elm_entry_entry_get(ad->entry);

        char filepath[PATH_MAX] = { 0, };
        app_get_data("text.txt", filepath, PATH_MAX);
        write_file(filepath, buf);
}
```

app_get_data() 是返回 /data 文件夹内已存在文件绝对路径的函数。

app_get_data_path() 是返回 /data 文件夹绝对路径的 API。

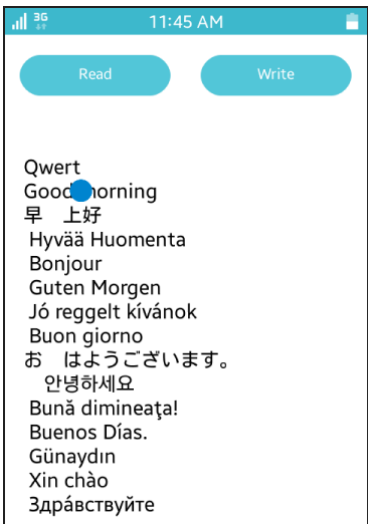
write_file() 是在文件内保存文本数据的函数。

fputs (char *, FILE *) 是在文件流上保存文本数据的 API。

btn_write_cb() 是第 2 个 Button 的回调函数。定义 /data 文件夹的绝对路径后，将输入 Entry 的文本数据保存于文件中。

让我们再运行一次示例。点击 Read 键加载文件，修改 Entry 内容后，点击 Write 键。

现在修改内容已全部保存于 /data 文件夹内。



3）读取 /data 文件夹内的文件

令人遗憾的是终止应用程序，重新运行后，文件内容并没有改变。这是因为虽然文件已经保存于 /data 文件夹内，但是从 /res 文件夹读取导入文件的缘故。下面将功能更改为在读取导入文件时首先从 /data 文件夹寻找；如果文件不存在，再从 /res 文件夹寻找。

将 btn_read_cb() 函数的内容修改为以下形式。这个函数的位置必须位于 app_get_data() 的下方。

```
static void
app_get_data(const char *res_file_in, char *res_path_out, int res_path_max)
{
        char *res_path = app_get_data_path();
        if (res_path) {
                snprintf(res_path_out, res_path_max, "%s%s", res_path, res_file_
in);
                free(res_path);
        }
}

static void
btn_read_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        char filepath[PATH_MAX] = { 0, };
        app_get_data("text.txt", filepath, PATH_MAX);
        // Read file in /data folder
        char *buf = NULL;
        buf = read_file(filepath);
        // If doesn't exist file in /data folder, read file in /res folder
        if( buf == NULL ) {
                app_get_resource("text.txt", filepath, PATH_MAX);
                buf = read_file(filepath);
        }

        elm_object_text_set(ad->entry, buf);
}
```

首先从 /data 文件夹读取文件。如果数据不存在，再从 /res 文件夹读取。

让我们再运行一次示例。点击 Read 键加载文件，修改 Entry 内容后，点击 Write 键保存。

终止应用程序后（长按 Home 键，出现应用程序目录后，点击 Clear all），重新运行。点击 Read 键，出现修改内容则更改完毕。

4）相关 API

FILE *fopen (char *, char *)：返回文件句柄的 API。在第 1 个参数上传入文件路径，在第 2 个参数上定义文件访问模式。“r”意味着只能读取，“w”意味着只能编写。

int fseek (FILE *__stream, long int __off, int __whence)：移动至文件流特定位置的 API。第 1 个参数是文件流，第 2 个是移动字节数，第 3 个是起始位置。SEEK_SET 意味着指向文件开头，SEEK_CUR 意味着指向当前位置，SEEK_END 意味着指向文件末尾。

long int ftell (FILE *__stream)：用字节数显示文件流当前位置的API。位于末尾时显示文件大小。

void rewind (FILE *__stream)：退回文件流初始位置的 API。

char *fgets (char *__s, int __n, FILE *__stream)：读取文件内文本数据的 API。在第 2 个参数上传入最大字符串长度，在第 3 个参数中传入文件流，第 1 个参数将会显示字符串数据。

int fclose (FILE *__stream)：关闭文件流的 API。

char *app_get_data_path(void)：返回 /data 文件夹绝对路径的API。

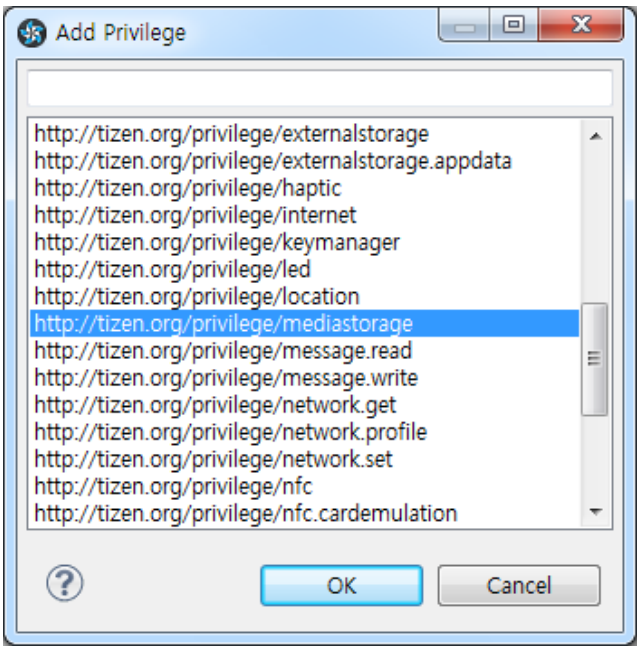
int fputs (char *__s, FILE *__stream)：在文件流上保存文本数据的 API。

59. 定义文件目录

Astro 等文件管理应用程序数年间一直处于必备应用程序的位置。不仅仅是文件管理应用程序，在图片浏览器和视频播放器中定义文件目录的功能必不可少。通过本例将展示将文件和文件夹目录添加到 List 小部件中，通过选择项目移动文件夹路径的简单文件管理应用程序制作过程。

1）登录权限

创建新的源项目，将 Project name 命名为 FileList。为了定义保存于内存中的文件目录，需拥有用户权限。创建源项目之后，打开 tizen-manifest.xml 文件，点击下方选项卡按键中的 Privileges。然后点击右侧上端的 Add 键出现弹窗后，从目录中选择http://tizen.org/privilege/mediastorage，点击 OK 键关闭弹窗。



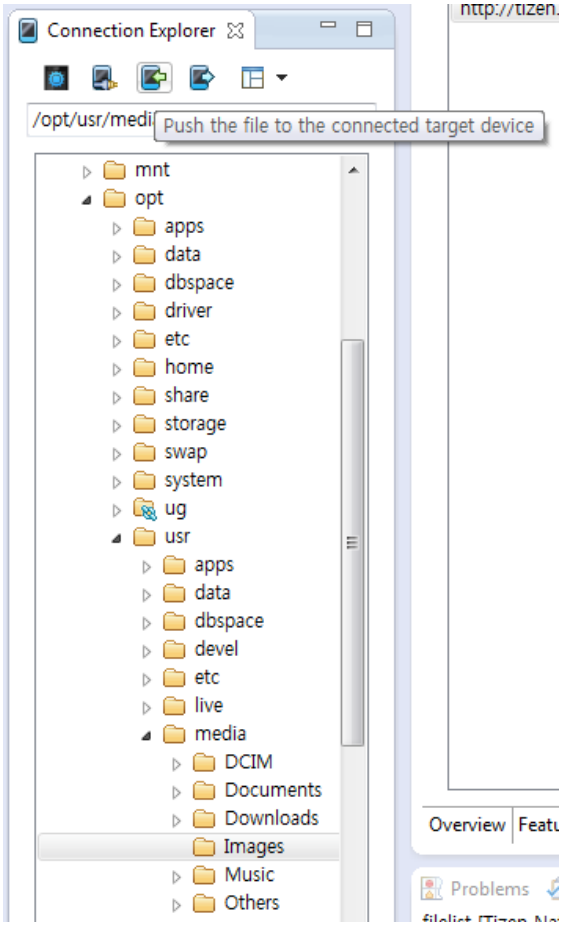
保存后点击下方选项卡按键中，位于右侧末端的 tizen-manifest.xml键，显示 xml 文件源代码。

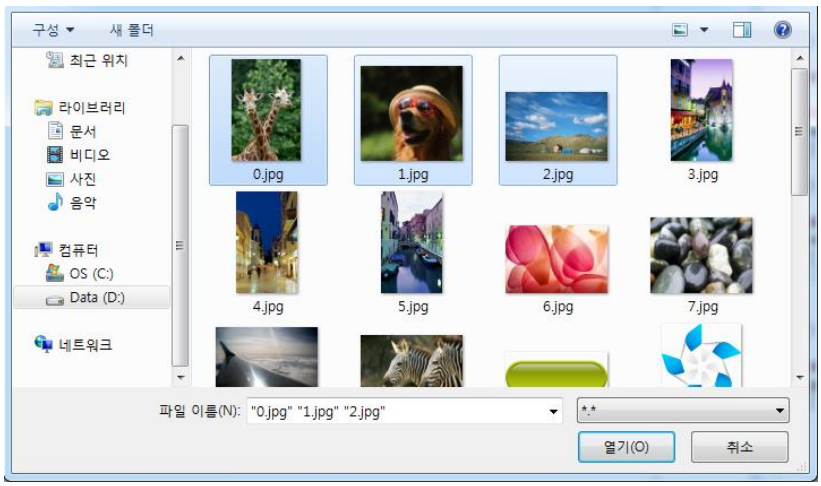
```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.ex
ample.filelist" version="1.0.0">
        <profile name="mobile" />
        <ui-application appid="org.example.filelist" exec="filelist" type="capp"
 multiple="false" taskmanage="true" nodisplay="false">
                <icon>filelist.png</icon>
                <label>filelist</label>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/mediastorage</privilege>
        </privileges>
</manifest>
```

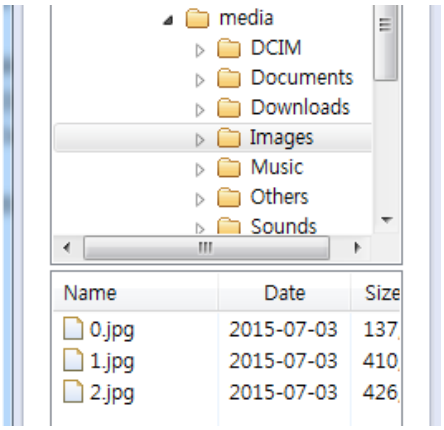
在默认情况下，模拟器在共享内存中以文件夹形式进行区分，但并不包含文件。为了进行测试，将几个图片文件复制到模拟器内存中。在 Eclipse 左侧下方的 Connection Explorer 中可以查看文件夹树形目录。保存图片的共享文件夹路径为 /opt/usr/media/Images。选择本文件夹，在上端工具栏中选择“Push the file to the connected target device”。

出现文件选择弹窗后，选择将要移动至附录 /Image 文件夹的 3 个图片文件（0.jpg, 1.jpg, 2.jpg），点击 OK 键。可以按个人喜好选择文件。





关闭弹窗将文件复制到 /Images 文件夹。



2）定义内存文件夹目录

下面来定义模拟器内存上的文件夹目录。打开位于 src 文件夹的源文件（~.c），添加 define 常量和变量。

```
#include "filelist.h"

#define FM_PHONE_FOLDER        "/opt/usr/media"
#define FM_MEMORY_FOLDER      "/opt/storage/sdcard"

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *list;
        char *current_path;
} appdata_s;
```

保存图片的共享文件夹路径为 /opt/usr/media/Images。
“/opt/storage/sdcard”是外置存储器的共享文件夹根路径。

List 是显示文件目录的 List 小部件变量。

current_path 是保存当前文件夹绝对路径的字符串变量。

在 create_base_gui() 函数末尾添加新代码。

```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        ad->current_path = calloc(PATH_MAX, sizeof(char));
        strcpy(ad->current_path, FM_PHONE_FOLDER );
        read_dir( ad );
}
```

在 current_path 上分配内存，复制内存根文件夹路径。

read_dir() 是特定文件夹内部定义文件目录的函数。下面就来创建读取功能。

在 create_base_gui() 上创建新函数。

```
static void
read_dir(appdata_s *ad)
{
        DIR *dir = opendir(ad->current_path);
        if( !dir )
                return;

        struct dirent *pDirent = NULL;
        char buf[100];

        while ((pDirent = readdir(dir)) != NULL)
        {
                if( pDirent->d_type == DT_DIR ) {
                        dlog_print(DLOG_INFO, "tag", "[Folder] %s", pDirent->d_n
ame);
                }
                else {
                        dlog_print(DLOG_INFO, "tag", "[File] %s", pDirent->d_nam
e);
                }
        }
        closedir(dir);
}
```

DIR 是控制文件夹的结构。可读取文件目录，删除、创建文件夹等。

opendir (char *) 是返回控制特定文件夹 DIR 对象的 API。

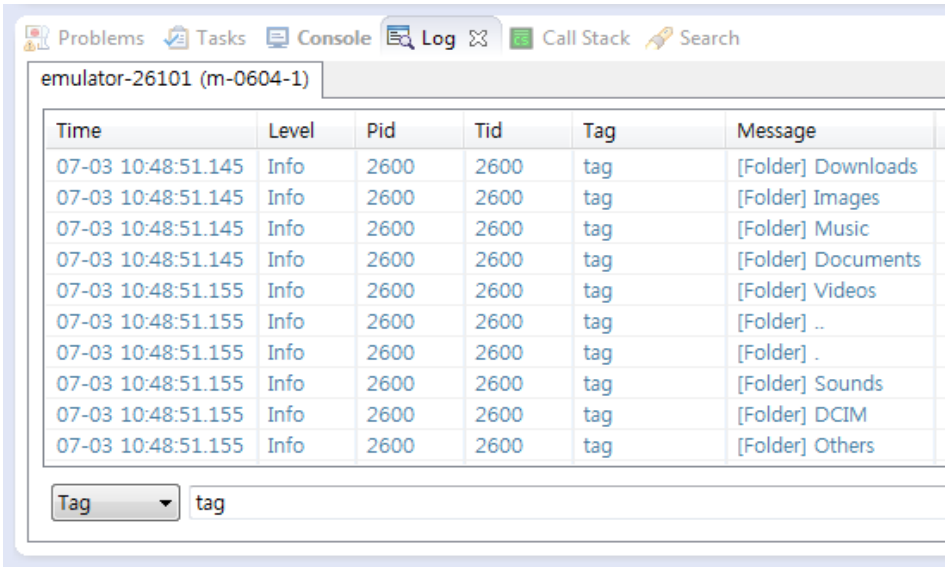
Dirent 是保存文件（或者文件夹）信息的结构。在属性中的 d_name 上已保存文件名称。在 d_type 中已保存形式。如果是 DT_DIR，就是文件夹，否则就是文件。

readdir (DIR *) 是读取特定文件夹内部文件目录，以 dirent 形式逐个返回的 API。到达文件末尾后，返回 NULL。

然后是区分文件和文件夹，显示 Log 信息的代码。

closedir (DIR *) 是关闭 DIR 的 API。

构建并运行示例。在 Log 面板下将呈现内存共用文件夹内的文件和文件夹目录。



3）在 List 小部件上添加文件目录

下面将文件目录添加到 List 小部件项目。在 create_base_gui() 函数中输入 Box 和 List 小部件创建代码。 将 Label 小部件按注释处理。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* Box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* List */
            ad->list = elm_list_add(ad->conform);
            elm_list_mode_set(ad->list, ELM_LIST_COMPRESS);
            evas_object_size_hint_weight_set(ad->list, EVAS_HINT_EXPAND, EVAS_H
INT_EXPAND);
            evas_object_size_hint_align_set(ad->list, EVAS_HINT_FILL, EVAS_HINT
_FILL);
            elm_box_pack_end(box, ad->list);
            evas_object_show(ad->list);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

将 read_dir() 函数修改为以下形式。

```
static void
read_dir(appdata_s *ad)
```

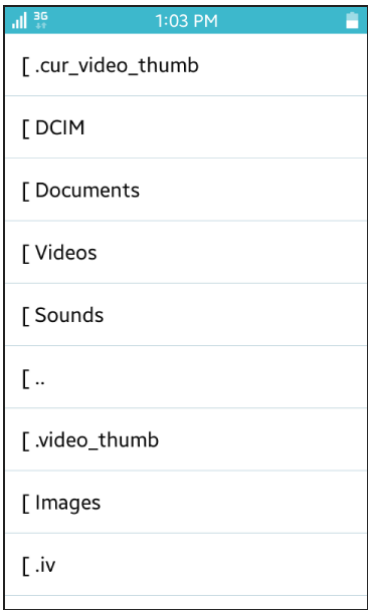
```
{
        DIR *dir = opendir(ad->current_path);
        if( !dir )
                return;

        struct dirent *pDirent = NULL;
        char buf[100];
        elm_list_clear(ad->list);

        while ((pDirent = readdir(dir)) != NULL)
        {
                if( pDirent->d_type == DT_DIR ) {
                        dlog_print(DLOG_INFO, "tag", "[Folder] %s", pDirent->d_n
ame);
                        sprintf(buf, "[ %s", pDirent->d_name);
                }
                else {
                        dlog_print(DLOG_INFO, "tag", "[File] %s", pDirent->d_nam
e);
                        sprintf(buf, "# %s", pDirent->d_name);
                }
                elm_list_item_append(ad->list, buf, NULL, NULL, NULL, ad);
        }
        closedir(dir);
}
```

elm_list_clear(Evas_Object *) 是删除所有 List 小部件项目的API。

为区分文件夹和文件，应用项目为文件夹时在名称前添加“[”符号，应用项目为文件时在名称前添加“#”符号。

elm_list_item_append(Evas_Object *, char *, Evas_Object *, Evas_Object *, Evas_Smart_Cb , void *)是在 List 小部件上添加新项目的 API。

让我们再运行一次示例。在 List 小部件上根文件夹内部的文件夹和文件目录已添加。“.”符号代表当前文件夹“..”符号代表上一级文件夹。



4）移动文件夹

下面来展示一下选择 List 小部件项目并移动至相应文件夹的功能。为此需要以下功能。

-选择 List 小部件项目的事件回调函数
-删除应用项目目录中的“.”符号和“..”符号
-当前文件夹不是根文件夹时，在首项添加“..”符号

在 read_dir() 函数中添加新代码。

```
static void
read_dir(appdata_s *ad)
{
        DIR *dir = opendir(ad->current_path);
        if( !dir )
                return;

        struct dirent *pDirent = NULL;
        char buf[100];
        elm_list_clear(ad->list);

        if( strcmp(ad->current_path, FM_PHONE_FOLDER) != 0 )
```

```
                elm_list_item_append(ad->list, "..", NULL, NULL, list_item_clic
ked, ad);

        while ((pDirent = readdir(dir)) != NULL)
        {
                if( strcmp(pDirent->d_name, ".") == 0 )
                        continue;
                if( strcmp(pDirent->d_name, "..") == 0 )
                        continue;

                if( pDirent->d_type == DT_DIR ) {
                        dlog_print(DLOG_INFO, "tag", "[Folder] %s", pDirent->d_n
ame);
                        sprintf(buf, "[ %s", pDirent->d_name);
                }
                else {
                        dlog_print(DLOG_INFO, "tag", "[File] %s", pDirent->d_nam
e);
                        sprintf(buf, "# %s", pDirent->d_name);
                }
                elm_list_item_append(ad->list, buf, NULL, NULL, list_item_click
ed, ad);
                //elm_list_item_append(ad->list, buf, NULL, NULL, NULL, ad);
        }
        closedir(dir);
}
```

当前文件夹为内存根文件夹时，在 List 小部件首项添加“..”符号。

应用项目名称为“.”符号和“..”时无视即可。

将项目选择事件回调函数命名为 list_item_clicked。下面就来创建此函数。在 read_dir() 上创建 3 个函数。

```
static char*
get_file_name(const char* item_text, bool *is_file)
{
        if( item_text[0] == '#' )
                *is_file = true;
        else
                *is_file = false;

        if( strcmp(item_text, "..") == 0 )
```

```
                return item_text;
        return item_text + 2;
}

static char*
get_new_path(char *current_path, const char *folder_name)
{
        if( strcmp(folder_name, "..") == 0)
        {
                int pos = strlen( current_path ) - strlen( strrchr( current_path,
 '/') );
                current_path[pos] = '\0';
        }
        else
                sprintf(current_path, "%s/%s", current_path, folder_name);
        return current_path;
}

static void
list_item_clicked(void *data, Evas_Object *obj, void *event_info)
{
        Elm_Object_Item *it = event_info;
        const char *item_text = elm_object_item_text_get(it);

        bool is_file;
        char *file_name = get_file_name(item_text, &is_file);
        if( is_file )
                return;

        appdata_s *ad = data;
        ad->current_path = get_new_path(ad->current_path, file_name);

        read_dir( ad );
}
```

get_file_name() 是接收 List 项目文本，去除符号并区分文件和文件夹的函数。

首字符为“#”时是文件，首字符为“[”时是文件夹。文本为“..”时代表上一级文件夹。

get_new_path() 是接收当前路径和新文件夹名称，定义整体文件夹路径的函数。

strrchr (char *, int) 是在字符串中搜索特定字符的 API。从后方开始搜索。因此返回至位于最后的字符指针。

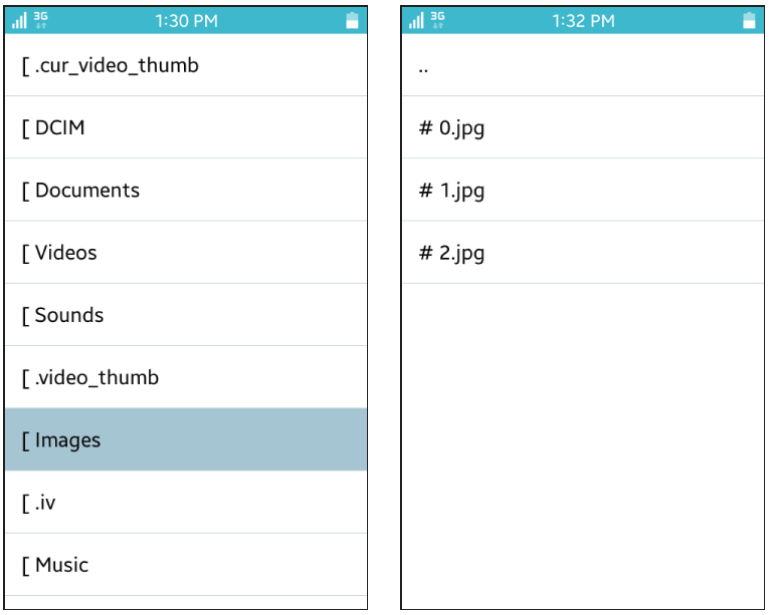
新文件夹名称为“..”时，代表上一级文件夹，因此在当前文件夹路径中删除最后文件夹的名称。否则就在当前文件夹路径后添加新文件夹名称，创建绝对路径。

list_item_clicked() 是 List 小部件项目选择事件函数。定义相应项目标题文本并区别应用项目名称后，创建新文件夹路径，将文件夹内的应用项目添加到 List 小部件上。

在此状态下创建会发生错误。这是因为 read_dir() 函数和 list_item_clicked() 函数相互调用的缘故。出现这种情况时，只需对函数的头进行声明即可。位置为源文件首端或者头文件。在本例中将在源文件首端进行声明。移动至源文件首端对 read_dir() 函数进行声明。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *list;
        char *current_path;
} appdata_s;

static void read_dir(appdata_s *ad);
```

重新运行示例，文件目录中“.”和“..”符号消失。选择 Images 项目后，出现之前复制的文件目录。在第 1 个项目上已添加“..”符号。选择后返回上一级文件夹。



5) ELM File Selector

使用 ELM File Selector 小部件可方便的对文件进行管理。创建新源项目，命名为 ElmFileSelectorEx。

创建源项目之后，打开 tizen-manifest.xml 文件，添加 Privilege。

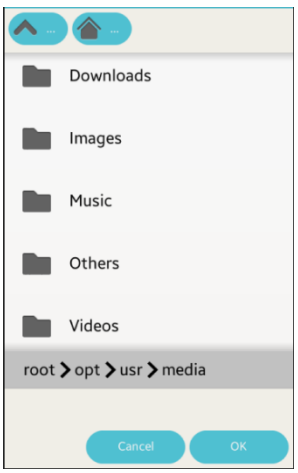
http://tizen.org/privilege/mediastorage

打开源文件（/src/elmfileselectorex.c），在 create_base_gui() 上添加新代码。创建 Box 和 FileSelector 的代码。将 Label 小部件创建代码删除。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);
```

```
    {
        /* Box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            Evas_Object *fs = elm_fileselector_add(ad->conform);
            evas_object_size_hint_weight_set(fs, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
            evas_object_size_hint_align_set(fs, EVAS_HINT_FILL, EVAS_HINT_FIL
L);
            elm_box_pack_end(box, fs);
            evas_object_show(fs);

            elm_fileselector_path_set(fs, FM_PHONE_FOLDER);
            //elm_fileselector_expandable_set(fs, EINA_TRUE);
            elm_fileselector_is_save_set(fs, EINA_FALSE);
            elm_fileselector_mode_set(fs, ELM_FILESELECTOR_LIST);
            elm_fileselector_folder_only_set(fs, EINA_FALSE);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

构建并运行示例。FileSelector 小部件上出现文件夹目录。



FileSelector 小部件的详细使用方法请参考以下链接。

https://docs.enlightenment.org/elementary/1.15.0/fileselector_example.html

https://docs.enlightenment.org/elementary/1.15.0/group__Fileselector.html

6）相关 API

DIR：控制文件夹的结构。可读取文件目录，删除、创建文件夹等。

DIR *opendir (char *__name)：返回控制特定文件夹 DIR 对象的API。

dirent：保存文件（或者文件夹）信息的结构。在属性中的 d_name 上保存文件名称。在 d_type 中已保存形式。如果是 DT_DIR 就是文件夹，否则就是文件。

struct dirent *readdir (DIR *__dirp)：读取特定文件夹内部文件目录，以 dirent 形式逐个返回的 API。 到达文件末尾后，返回 NULL。

int closedir (DIR *__dirp)：关闭 DIR 的 API。

void elm_list_clear(Evas_Object *obj)：删除所有 List 小部件项目的 API。

Elm_Object_Item *elm_list_item_append(Evas_Object *obj, const char *label, Evas_Object *icon, Evas_Object *end, Evas_Smart_Cb func, const void *data)：在 List 小部件上添加新项目的 API。

char *strrchr (char *__s, int __c)：在字符串中搜索特定字符的API。从后方开始搜索。因此返回至位于最后的字符指针。

60. Preference 使用方法

将应用程序的环境设置等信息保存在注册表中较为方便。Preference 是本地数据存储空间。非常适合保存环境设置信息。删除应用程序后，相应 Preference 也将一起被删除。

1）在 Preference 上保存字符串数据

创建新源项目，将 Project name 命名为 PreferenceEx。创建源项目之后，打开 src 文件夹内的源文件 （~.c），添加库头文件和变量。

```
#include "preferenceex.h"
#include <app_preference.h>
#include <stdlib.h>

typedef struct appdata {
    Evas_Object *win;
    Evas_Object *conform;
    Evas_Object *entry1;
    Evas_Object *spinner1;
} appdata_s;

const char *string_key = "string_key";
const char *integer_key = "integer_key";
```

app_preference.h 是使用 Preference 的库头文件。

stdlib.h 是字符串和数字形式转换的库头文件。

在 entry1 中输入字符串数据，在 spinner1 中输入数字数据。

下面将在屏幕中添加几种小部件。在 create_base_gui() 上创建 2 个新函数。

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int x, int y, int w, int
```

```
h)
{
    evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
    evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_table_pack(table, child, x, y, w, h);
    evas_object_show(child);
}

static Evas_Object *
my_button_add(Evas_Object *parent, const char *text, Evas_Smart_Cb cb, void *cb_
data)
{
    Evas_Object *btn;

    btn = elm_button_add(parent);
    elm_object_text_set(btn, text);
    evas_object_smart_callback_add(btn, "clicked", cb, cb_data);

    return btn;
}
```

my_box_pack() 是在 Table 上添加小部件的函数。

my_button_add() 是创建 Button 小部件的函数。

然后在 create_base_gui() 函数中添加新代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* Box to put the table in so we can bottom-align the table
         * window will stretch all resize object content to win size */
        Evas_Object *box = elm_box_add(ad->conform);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, 0.0);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);
```

```
        /* Table */
        Evas_Object *table = elm_table_add(ad->conform);
        /* Make table homogenous - every cell will be the same size */
        elm_table_homogeneous_set(table, EINA_TRUE);
        /* Set padding of 10 pixels multiplied by scale factor of UI */
        elm_table_padding_set(table, 10 * elm_config_scale_get(), 30 * elm_conf
ig_scale_get());
        /* Let the table child allocation area expand within in the box */
        evas_object_size_hint_weight_set(table, EVAS_HINT_EXPAND, 0.0);
        /* Set table to fiill width but align to bottom of box */
        evas_object_size_hint_align_set(table, EVAS_HINT_FILL, 0.0);
        elm_box_pack_end(box, table);
        evas_object_show(table);

        {
            /* Label-1 */
            Evas_Object *label = elm_label_add(ad->conform);
            elm_object_text_set(label, "Pet name:");
            my_table_pack(table, label, 0, 0, 1, 1);

            /* Bg-1 */
            Evas_Object *bg = elm_bg_add(ad->conform);
            elm_bg_color_set(bg, 210, 210, 210);
            my_table_pack(table, bg, 1, 0, 1, 1);

            /* Entry-1 */
            ad->entry1 = elm_entry_add(ad->conform);
            my_table_pack(table, ad->entry1, 1, 0, 1, 1);

            /* Label-2 */
            label = elm_label_add(ad->conform);
            elm_object_text_set(label, "Percentage:");
            my_table_pack(table, label, 0, 1, 1, 1);

            /* Spinner-1 */
            ad->spinner1 = elm_spinner_add(ad->conform);
            elm_spinner_editable_set(ad->spinner1, EINA_TRUE);
            elm_spinner_interval_set(ad->spinner1, 1);
            elm_spinner_min_max_set(ad->spinner1, 0, 100);
            elm_spinner_label_format_set(ad->spinner1, "%.0f");
            my_table_pack(table, ad->spinner1, 1, 1, 1, 1);

            Evas_Object *btn;

            /* Button-Save */
```

```
            btn = my_button_add(ad->conform, "Save", btn_save_cb, ad);
            my_table_pack(table, btn, 0, 3, 2, 1);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

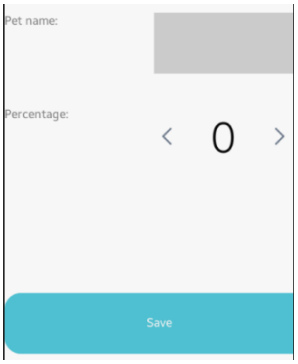
已添加 Box、Table、2 个Label、Bg、Entry、Spinner、1 个Button小部件。在第 1 个 Entry 上输入字符串，点击 Button 键将用户输入的字符串保存在 Preference 中。

在 create_base_gui() 上创建 Button 回调函数。

```
static void
btn_save_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    const char *string_value;
    int integer_value;

    string_value = elm_object_text_get(ad->entry1);
    preference_set_string(string_key, string_value);
}
```

preference_set_string(char *, char *) 是在 Preference 上保存字符串数据的 API。在第 1 个参数上传入 Key，在第 2 个参数上传入字符串数据。Key 必须在读写时保持一致。

构建并运行示例。在第 1 个 Entry 上输入字符串，点击 Save Button。所输入的字符串将会被保存。可惜的是尚不具备读取功能。下面就来创建读取功能。



2）读取字符串 Preference

添加第 2 个 Button，在 Preference 中创建数据读取功能。在 create_base_gui() 函数中添加新代码。

```
            Evas_Object *btn;

            /* Button-Load */
            btn = my_button_add(ad->conform, "Load", btn_read_cb, ad);
            my_table_pack(table, btn, 0, 2, 2, 1);

            /* Button-Save */
            btn = my_button_add(ad->conform, "Save", btn_save_cb, ad);
            my_table_pack(table, btn, 0, 3, 2, 1);
        }
    }
```

下面来创建新添加 Button 的回调函数。在 create_base_gui() 函数上添加新函数。

```
static void
btn_read_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    char *string_value = "";
    bool existing = false;
```

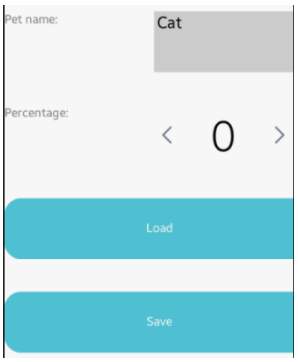
```
    if ((preference_is_existing(string_key, &existing) == 0) && existing)
    {
        preference_get_string(string_key, &string_value);
        elm_object_text_set(ad->entry1, string_value);
        free(string_value);
    }
}
```

在 Preference 中读取数据时，首先确认相应数据是否存在。

preference_is_existing(const char *, bool *) 是确认在Preference 中是否存在特定数据的 API。在第 1 个参数上传入 Key，在第 2 个参数上将会显示数据是否存在。

preference_get_string(char *, char **) 是在 Preference 中读取字符串数据的 API。在第 1 个参数上传入 Key，在第 2 个参数中返回字符串数据。

重新运行示例，在第 1 个 Entry 上输入字符串，点击 Save 键。然后在 Entry 上删除已输入的字符，点击 Read 键。之前输入的字符串将会显示在 Entry 上。



3） Preference数字读取与编写

下面就在 Preference 上保存数字后重新读取。首先在保存函数中添加以下新代码。

```
static void
btn_save_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    const char *string_value;
int integer_value;

    string_value = elm_object_text_get(ad->entry1);
    preference_set_string(string_key, string_value);

    integer_value = (int) elm_spinner_value_get(ad->spinner1);
    preference_set_int(integer_key, integer_value);
}
```

atoi (char *) 作为 Array to Int 的简写是将字符串转换为数字的 API。

preference_set_int(char *, int) 是在 Preference 中保存整数类型数据的 API。

在读取数据函数中也添加新代码。

```
static void
btn_read_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    char *string_value = "";
int integer_value;
    bool existing = false;

    if ((preference_is_existing(string_key, &existing) == 0) && existing)
    {
        preference_get_string(string_key, &string_value);
        elm_object_text_set(ad->entry1, string_value);
        free(string_value);
    }
```

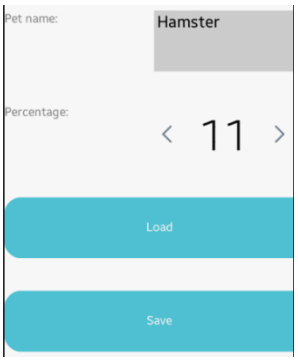
```
    if ((preference_is_existing(integer_key, &existing) == 0) && existing)
    {
        preference_get_int(integer_key, &integer_value);
        elm_spinner_value_set(ad->spinner1, (double) integer_value);
    }
}
```

preference_get_int(char *, int *) 是在 Preference 中读取整数类型数据的 API。

eina_convert_itoa(int, char *) 是将整数转换为字符串的 API。

让我们再运行一次示例。在第 1 个 Entry 上输入字符串，在第 2 个 Entry 上输入数字后，点击 Save 键。

变更输入数据后重新运行，点击 Read 键，保存在 Preference 中的数据将会重新显示。



4）相关 API

int preference_set_string(const char *key, const char *value)：在 Preference 中保存字符串数据的 API。在第 1 个参数上传入 Key，在第 2 个参数上传入字符串数据。Key 必须在读写时保持一致。

int preference_is_existing(const char *key, bool *existing)：确认 Preference 中是否存在特定数据的 API。在第 1 个参数上传入 Key，在第 2 个参数上将会显示数据是否存在数据。

int preference_get_string(const char *key, char **value)：在 Preference 中读取字符串数据的 API。在第 1 个参数上传入 Key，在第 2 个参数中将会返回字符串数据。

int atoi (char *__nptr)：作为 Array to Int 的简写是将字符串转换为数字的 API。

int preference_set_int(char *key, int value)：在 Preference 中保存整数型数据的 API。

int preference_get_int(char *key, int *value)：在 Preference 中读取整数型数据的 API。

int eina_convert_itoa(int n, char *s)：将整数转换为字符串的 API。

61. 用 SQLite 制作成绩表的示例

用户环境设置信息虽然可以使用 Preference 保存，但像日程管理、联系方式、成绩管理等系统性庞大的数据必须使用数据库。在大部分手机平台中提供系统性 SQLite 数据库，在 Tizen 中也支持 SQLite。SQLite 虽然相比甲骨文或 MS-SQL 最大数据容量偏小，但在移动设备中已足够使用。使用方法简单，并且因采用标准 SQL 查询语法，与其他系统中使用的数据库兼容性强。

一个数据库包含 1 个以上的 Table。将其想象成与 excel 工作表类似的 2 维表即可。横向排列有很多栏，填写由各字段集合组成的数据时，纵向累积单列记录。下面就利用 SQLite 创建简单成绩表示例。

1）创建数据库

创建新的源项目，将 Project name 命名为 SqliteEx。创建源项目之后，打开 src 文件夹内的源文件 （~.c），添加库头文件、变量和结构。

```
#include "sqliteex.h"
#include <sqlite3.h>
#include <stdlib.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object *entry1;
        Evas_Object *entry2;
        Evas_Object *entry3;
        Evas_Object *list;

        sqlite3 *db; // Database handle
        char *current_key;
} appdata_s;

typedef struct recdata {
        char key[10];
        char name[255];
```

```
        char english[10];
        char math[10];
} recdata_s;

appdata_s *m_ad;
```

sqlite3.h 是使用 SQLite 的库头文件。

stdlib.h 是字符串和数字形式转换的库头文件。

在 appdata 结构中添加 3 个 Entry。输入各自姓名、英语分数、数学分数。

在 list 小部件中显示数据库目录。

sqlite3 是数据库对象变量。

在 current_key 中保存当前选择的记录 Key 值。

recdata 是保存学生成绩数据的结构。1 个 recdata 对应一个记录。

为了随时随地访问 appdata，将 m_ad 用于定义全局变量。

在 /data 文件夹中创建数据库文件，创建成绩表。在 create_base_gui() 函数上创建 3 个新函数。

```
static int CreateTable(appdata_s *ad)
{
      char *ErrMsg;
      char *sql = "CREATE TABLE IF NOT EXISTS ReportCard(KEY INTEGER PRIMARY KEY,
 NAME TEXT NOT NULL, ENGLISH INT NOT NULL, MATH INT NOT NULL);";

      int ret = sqlite3_exec(ad->db, sql, NULL, 0, &ErrMsg);
      return ret;
}

static void
init_db(appdata_s *ad)
```

```
{
        sqlite3_shutdown();
        sqlite3_config(SQLITE_CONFIG_URI, 1);
        sqlite3_initialize();
        char * resource = app_get_data_path();
        int siz = strlen(resource) + 10;
        char * path = malloc(sizeof(char)*siz);
        strncat(path, resource, siz);
        strncat(path, "test.db", siz);

        sqlite3_open(path, &ad->db);
        free(path);

        CreateTable(ad);
}

static void
my_table_pack(Evas_Object *table, Evas_Object *child, int x, int y, int w, int h)
{
    evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
    evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_table_pack(table, child, x, y, w, h);
    evas_object_show(child);
}
```

CreateTable() 是利用 SQL 查询语法创建 Table 的函数。

查询语法中“CREATE TABLE”是创建 Table 的指令。

“IF NOT EXISTS”意味着 Table 不存在，需要创建。

将 Table 名称命名为 ReportCard。

“KEY INTEGER PRIMARY KEY”是添加 Key 栏的代码。采用数学形式，添加记录时，数字自动增加。

“NAME TEXT NOT NULL”是添加保存姓名栏的代码。采用文本形式，不得为空。

“ENGLISH INT NOT NULL”是添加保存英语分数栏的代码。采用数字形式，不得为空。

“MATH INT NOT NULL”是添加保存数学分数栏的代码。采用数字形式，不得为空。

sqlite3_exec(sqlite3*, char *, int (*callback), void *, char **) 是运行 SQL 查询语言的 API.参数依次为 SQLite 对象、查询语言、回调函数名称、用户数据、错误信息显示。

init_db() 是创建数据库文件的函数。

sqlite3_shutdown() 是关闭数据库的 API。

sqlite3_config(int, ...) 是定义数据库属性的 API。传入 SQLITE_CONFIG_URI 后，可以在数据库文件中保存数据。

sqlite3_initialize() 是初始化数据库的 API。

app_get_data_path() 是返回 /data 文件夹绝对路径的 API。在 /res 文件夹中不允许编写，所以在 /data 文件夹中创建数据库文件。

strncat (char *, char *, size_t) 是定义最大长度后在字符串后添加新字符串的 API。参数依次为原本字符串排列，将要添加的字符串数据，最大长度。

sqlite3_open(char *, sqlite3 **) 是打开数据库文件的 API。若数据库文件不存在则需创建。在第 1 个参数上传入文件路径，第 2 个参数将会返回数据库对象。

my_box_pack() 是在 Table 上添加小部件的函数。

运行应用程序后将自动打开数据库文件。在 create_base_gui() 函数末尾调用上述函数。

```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        init_db(ad);
}
```

2）添加新记录

下面来展示一下在 3 个 Entyr 小部件中输入姓名、英语成绩、数学成绩，点击 Button，在数据库中添加新记录的功能。在 create_base_gui() 函数中添加新代码。将 Label 小部件按注释处理。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        /* Box to put the table in so we can bottom-align the table
         * window will stretch all resize object content to win size */
        Evas_Object *box = elm_box_add(ad->conform);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, 0.0);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        /* Table */
        Evas_Object *table = elm_table_add(ad->conform);
        /* Make table homogenous - every cell will be the same size */
        elm_table_homogeneous_set(table, EINA_TRUE);
        /* Set padding of 10 pixels multiplied by scale factor of UI */
        elm_table_padding_set(table, 10 * elm_config_scale_get(), 10 * elm_conf
ig_scale_get());
        /* Let the table child allocation area expand within in the box */
        evas_object_size_hint_weight_set(table, EVAS_HINT_EXPAND, EVAS_HINT_EXP
AND);
        /* Set table to fiill width but align to bottom of box */
        evas_object_size_hint_align_set(table, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_box_pack_end(box, table);
        evas_object_show(table);

        {
            /* Bg-1 */
            Evas_Object *bg = elm_bg_add(ad->conform);
            elm_bg_color_set(bg, 210, 210, 210);
            my_table_pack(table, bg, 0, 0, 1, 1);
```

```
        /* Entry-1 */
        ad->entry1 = elm_entry_add(ad->conform);
        elm_object_part_text_set(ad->entry1, "elm.guide", "Name");
        my_table_pack(table, ad->entry1, 0, 0, 1, 1);

        /* Bg-2 */
        bg = elm_bg_add(ad->conform);
        elm_bg_color_set(bg, 210, 210, 210);
        my_table_pack(table, bg, 1, 0, 1, 1);

        /* Entry-2 */
        ad->entry2 = elm_entry_add(ad->conform);
        elm_object_part_text_set(ad->entry2, "elm.guide", "English");
        my_table_pack(table, ad->entry2, 1, 0, 1, 1);

        /* Bg-3 */
        bg = elm_bg_add(ad->conform);
        elm_bg_color_set(bg, 210, 210, 210);
        my_table_pack(table, bg, 2, 0, 1, 1);

        /* Entry-3 */
        ad->entry3 = elm_entry_add(ad->conform);
        elm_object_part_text_set(ad->entry3, "elm.guide", "Math");
        my_table_pack(table, ad->entry3, 2, 0, 1, 1);

        /* Button-Add */
        Evas_Object *btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Add");
        evas_object_smart_callback_add(btn, "clicked", btn_add_cb, ad);
        my_table_pack(table, btn, 0, 1, 1, 1);
    }
}

/* Show window after base gui is set up */
evas_object_show(ad->win);
```

创建 Box、Table、3 个 Bg 和 3 个 Entry 小部件。同时创建 1 个 Button 小部件。

点击 Button，展示在数据库中添加输入 Entry 的数据。在 create_base_gui() 函数上创建 2 个新函数。

```
static int
```

```
InsertRecord(appdata_s *ad, unsigned char *name, int english, int math)
{
        char sql[256];
        char *ErrMsg;
        snprintf(sql, 256, "INSERT INTO ReportCard VALUES(NULL,\'%s\',%d,%d);",
name, english, math);
        int ret = sqlite3_exec(ad->db, sql, NULL, 0, &ErrMsg);
        return ret;
}

static void
btn_add_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;

        char* s_name = elm_object_text_get(ad->entry1);
        char* s_english = elm_object_text_get(ad->entry2);
        int n_english = atoi (s_english);
        char* s_math = elm_object_text_get(ad->entry3);
        int n_math = atoi (s_math);

        InsertRecord(ad, s_name, n_english, n_math);
}
```

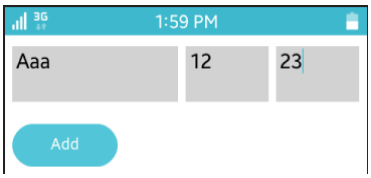
InsertRecord() 是利用 SQL 查询语法，添加新记录的函数。

在查询语法中“INSERT INTO ReportCard”是在名为 ReportCard 的 Table 中添加新记录的指令。

在 VALUES() 中传入记录数据。因为第 1 栏 Key 自动创建，在此省略。

btn_add_cb() 是用户点击 Button 时，定义 Entry 中所输入的数据并将其保存于数据库的函数。

构建并运行示例。在第 1 个 Entry 中输入姓名，在第 2 个 Entry 中输入英语分数，在第 3 个 Entry 中输入数学分数，点击 Add 键。在数据库中已添加新记录。令人遗憾的是无法确认。因为还没有创建读取数据库保存数据的功能。



3）读取数据库数据

下面来展示一下读取保存于数据库的数据并将其显示在屏幕上的功能。为此需要在 create_base_gui() 函数中添加 List 小部件创建代码。

```
static void
create_base_gui(appdata_s *ad)
{
        m_ad = ad;

        ~


            /* Button-Add */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Add");
            evas_object_smart_callback_add(btn, "clicked", btn_add_cb, ad);
            my_table_pack(table, btn, 0, 1, 1, 1);

            /* List */
            ad->list = elm_list_add(ad->conform);
            elm_list_mode_set(ad->list, ELM_LIST_COMPRESS);
            elm_list_go(ad->list);
            my_table_pack(table, ad->list, 0, 2, 3, 8);
        }
    }

        /* Show window after base gui is set up */
        evas_object_show(ad->win);
```

为了随时随地访问 appdata 对象，保存于全局变量中。添加 2 个新函数。本函数必须位于 init_db() 和 btn_add_cb() 之上。

```
static int db_read_cb(void *counter, int argc, char **argv, char **azColName)
{
        char buf[255];

        recdata_s* rd = malloc(sizeof(recdata_s));
        strcpy(rd->key, argv[0]);
        strcpy(rd->name, argv[1]);
        strcpy(rd->english, argv[2]);
        strcpy(rd->math, argv[3]);

        sprintf(buf, "%s / %s / %s / %s", argv[0], argv[1], argv[2], argv[3]);
        elm_list_item_append(m_ad->list, buf, NULL, NULL, NULL, (void*)rd);
        elm_list_go(m_ad->list);
        return 0;
}

static int read_db(appdata_s *ad)
{
        char *sql = "select * from ReportCard";
        int counter=0;
        char *ErrMsg;

        elm_list_clear(ad->list);
        int ret = sqlite3_exec(ad->db, sql, db_read_cb, &counter, &ErrMsg);
        return ret;
}
```

在数据库中读取多个记录时，单个记录信息将被传入回调函数。db_read_cb() 是接受 1 个记录并进行处理的回调函数。在第 3 个参数上排列传入数据。

创建 recdata_s 结构，保存各字段数据，将所有数据集合成一个字符串，在 List 小部件上添加新项目。

read_db() 是读取数据库上保存的所有数据并显示在 List 小部件上的函数。

“select * from ReportCard”查询语言表示返回 ReportCard Table 所保

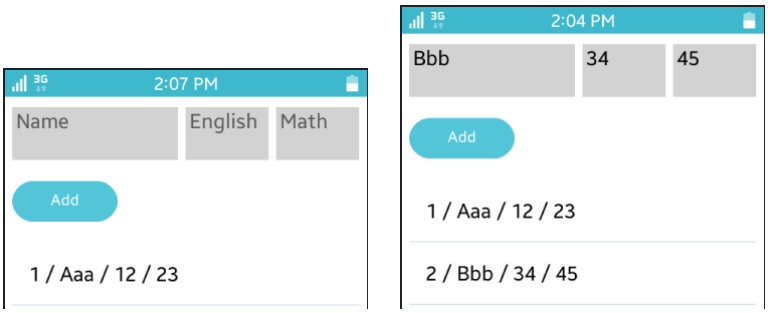
存的所有数据。

上述函数在运行应用程序后，点击 Button 调用即可。在 init_db() 和 btn_add_cb() 函数末尾添加新代码。

```
static void
init_db(appdata_s *ad)
{
        ~
        CreateTable(ad);
        read_db(ad);
}

static void
btn_add_cb(void *data, Evas_Object *obj, void *event_info)
{
        ~
        InsertRecord(ad, s_name, n_english, n_math);
        read_db(ad);
}
```

因此 read_db() 必须位于 init_db() 和 btn_add_cb() 之上。

让我们再运行一次示例。刚刚输入的数据已添加到 List 小部件上。输入新数据，点击 Add 键。在 List 小部件上添加新项目。



4）修改数据

下面来了解一下修改已保存数据的方法。首先创建用户选择 List 项目后，在 Entry 上显示项目数据的功能。在 db_read_cb() 函数中添加新项目的代码中定义回调函数名称。然后创建回调函数。

```
static void
list_item_clicked(void *data, Evas_Object *obj, void *event_info)
{
        recdata_s* rd = (recdata_s*)data;
        m_ad->current_key = rd->key;
        elm_object_text_set(m_ad->entry1, rd->name);
        elm_object_text_set(m_ad->entry2, rd->english);
        elm_object_text_set(m_ad->entry3, rd->math);
}

static int db_read_cb(void *counter, int argc, char **argv, char **azColName)
{
        char buf[255];

        recdata_s* rd = malloc(sizeof(recdata_s));
        strcpy(rd->key, argv[0]);
        strcpy(rd->name, argv[1]);
        strcpy(rd->english, argv[2]);
        strcpy(rd->math, argv[3]);

        sprintf(buf, "%s / %s / %s / %s", argv[0], argv[1], argv[2], argv[3]);
        elm_list_item_append(m_ad->list, buf, NULL, NULL, list_item_clicked, (v
```

```
oid*)rd);
        //elm_list_item_append(m_ad->list, buf, NULL, NULL, NULL, (void*)rd);
        elm_list_go(m_ad->list);
        return 0;
}
```

list_item_clicked() 是 List 小部件项目选择事件的回调函数。在全局变量中保存 Key 值，在 Entry 小部件中显示其他数据。

在 db_read_cb() 函数中 List 小部件项目添加代码已修改。定义项目选择回调函数名称。

然后在 create_base_gui() 函数中添加新 Button 创建代码。

```
            /* Button-Add */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Add");
            evas_object_smart_callback_add(btn, "clicked", btn_add_cb, ad);
            my_table_pack(table, btn, 0, 1, 1, 1);

            /* Button-Update */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Update");
            evas_object_smart_callback_add(btn, "clicked", btn_update_cb, ad);
            my_table_pack(table, btn, 1, 1, 1, 1);

            /* List */
            ad->list = elm_list_add(ad->conform);
            elm_list_mode_set(ad->list, ELM_LIST_COMPRESS);
            elm_list_go(ad->list);
            my_table_pack(table, ad->list, 0, 2, 3, 8);
```

点击 Button 展示修改当前所选项目数据的功能。在 create_base_gui() 上创建 2 个新函数。

```
static int
UpdateRecord(appdata_s *ad, unsigned char *name, unsigned char *english, unsigne
d char *math)
{
        char sql[256];
```

```
        char *ErrMsg;
        snprintf(sql, 256, "UPDATE ReportCard SET NAME=\'%s\', ENGLISH=\'%s\', M
ATH=\'%s\' WHERE KEY=\'%s\';",
                        name, english, math, ad->current_key);
        int ret = sqlite3_exec(ad->db, sql, NULL, 0, &ErrMsg);
        return ret;
}

static void
btn_update_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;

        char* s_name = elm_object_text_get(ad->entry1);
        char* s_english = elm_object_text_get(ad->entry2);
        char* s_math = elm_object_text_get(ad->entry3);

        UpdateRecord(ad, s_name, s_english, s_math);

        read_db(ad);
}
```

UpdateRecord() 是修改数据库保存数据的函数。

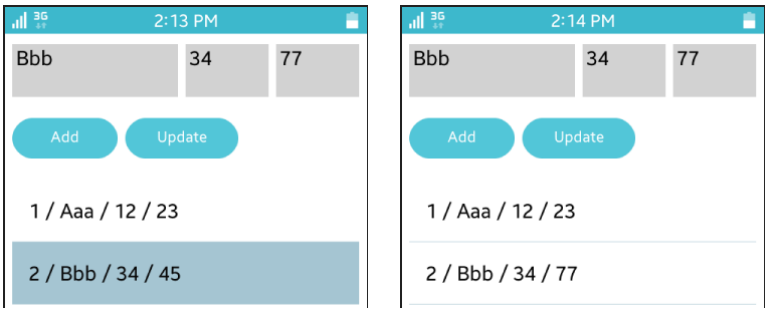
查询语言中“UPDATE ReportCard”是修改 ReportCard Table 保存数据的意思。

SET NAME=\'%s\’ 是在 NAME 栏保存字符串数据的意思。

WHERE KEY=\'%s\’ 是修改 KEY 值一致记录的意思。

btn_update_cb() 是将输入 Entry 的数据保存于当前所选记录的函数。

让我们再运行一次示例。在 List 项目中选择一项，在 Entry 中变更数据后，点击 Update键。List 小部件中的数据将被转换。保存于数据库当中。



5）删除记录

下面来展示一下添加第 3 个 Button，用户点击 Button 删除当前所选记录的功能。在 create_base_gui() 函数中添加 Button 创建代码。

```
        /* Button-Update */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Update");
        evas_object_smart_callback_add(btn, "clicked", btn_update_cb, ad);
        my_table_pack(table, btn, 1, 1, 1, 1);

        /* Button-Del */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Del");
        evas_object_smart_callback_add(btn, "clicked", btn_del_cb, ad);
        my_table_pack(table, btn, 2, 1, 1, 1);

        /* List */
        ad->list = elm_list_add(ad->conform);
        elm_list_mode_set(ad->list, ELM_LIST_COMPRESS);
        elm_list_go(ad->list);
        my_table_pack(table, ad->list, 0, 2, 3, 8);
```

然后在 create_base_gui() 上添加 2 个新函数。

```
static int
DelRecord(appdata_s *ad)
{
        char sql[256];
```

```
        char *ErrMsg;
        snprintf(sql, 256, "DELETE FROM ReportCard WHERE KEY=\'%s\';", ad->curre
nt_key);

        int ret = sqlite3_exec(ad->db, sql, NULL, 0, &ErrMsg);
        return ret;
}

static void
btn_del_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;

        DelRecord(ad);

        read_db(ad);
}
```

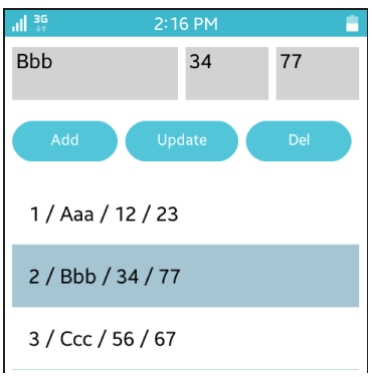
DelRecord() 是删除数据库保存记录的函数。

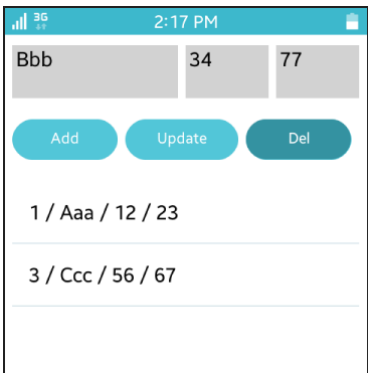
在查询语法中“DELETE FROM ReportCard”是删除 ReportCard Table 数据的意思。

WHERE KEY=\'%s\' 是删除 KEY 值一致记录的意思。

btn_del_cb() 是用户点击 Del 键时，删除当前所选 List 项目的函数。

让我们再运行一次示例。增添几个项目后，在 List 项目中选择一项，点击 Del 键相应项目消失。





6）相关 API

int sqlite3_exec(sqlite3*, char *, int (*callback), void *, char **)：运行 SQL 查询语言的 API。参数依次为 SQLite 对象、查询语言、回调函数名称、用户数据、错误信息显示。

int sqlite3_shutdown()：关闭数据库的 API。

int sqlite3_config(int, ...)：定义数据库属性的 API。传入 SQLITE_CONFIG_URI 后，可以在数据库文件中保存数据。

int sqlite3_initialize()：初始化数据库的 API.

char *app_get_data_path()：返回 /data 文件夹绝对路径的 API。在 /res 文件夹中不允许编写，因此在 /data 文件夹中创建数据库文件。

char *strncat (char *, char *, size_t)：定义最大长度在字符串后添加新字符串的 API。参数依次为原本字符串排列、将要添加的字符串数据、最大长度。

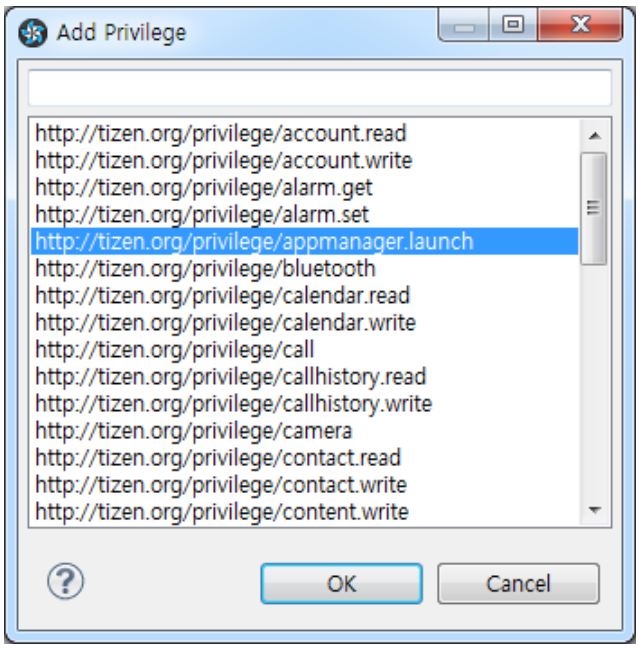
int sqlite3_open(char *, sqlite3 **)：打开数据库文件的 API。若数据库文件不存在则创建。在第 1 个参数上传入文件路径，第 2 个参数将会返回数据库对象。

62. 用 AppControl 调用外部应用程序

在需要开发的应用程序中通过硬编码实现所需图片浏览器或者照相机所有功能十分困难。Tizen 平台中默认安装的应用程序被称为 AppControl，在其他应用程序中可加载 AppControl。即使不是默认应用程序，只要知道安装包名称就可以加载。

1）登录权限

创建新的源项目，将 Project name 命名为 AppControlEx。为了运行外部应用程序，需拥有用户权限。创建源项目之后，打开 tizen-manifest.xml 文件，点击下方选项卡按键中的 Privileges。然后点击右侧上端的 Add 键。出现弹窗后，从目录中选择 http://tizen.org/privilege/appmanager.launch，点击 OK 键关闭弹窗。



重复相同过程添加以下 2 个用户权限。

- http://tizen.org/privilege/internet
- http://tizen.org/privilege/email

保存后点击下方选项卡按键中，位于右侧末端的 tizen-manifest.xml 键，显示 xml 文件源代码。

```
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.appcontrolex" version="1.0.0">
        <profile name="mobile"/>
        <ui-application appid="org.example.appcontrolex" exec="appcontrolex" multiple="false" nodisplay="false" taskmanage="true" type="capp">
                <label>appcontrolex</label>
                <icon>appcontrolex.png</icon>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/internet</privilege>
                <privilege>http://tizen.org/privilege/appmanager.launch</privilege>
                <privilege>http://tizen.org/privilege/email</privilege>
        </privileges>
</manifest>
```

2）加载示例

打开 src 文件夹内的源文件 （~.c），添加库头文件、变量和结构。

```
#include "appcontrolex.h"
#include <app.h>
#include <app_control.h>

#define FM_PHONE_FOLDER       "/opt/usr/media"

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
} appdata_s;
```

app.h 和 app_control.h 是使用 AppControl 的库头文件。

“/opt/usr/media”是共享文件夹根路径。

下面从目前为止以创建的示例中选择 HelloWorld 示例加载。在 create_base_gui() 函数中添加 Box、Button 创建代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    { /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        { /* child object - indent to how relationship */
            /* Label*/
            ad->label = elm_label_add(ad->win);
            elm_object_text_set(ad->label, "<align=center>Hello Tizen</>");
            evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);

            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Sample App");
            evas_object_smart_callback_add(btn, "clicked", btn_sample_app_cb, ad);
            /* epand both horiz and vert, fill horiz and vert */
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

在 create_base_gui() 上创建 2 个新函数。

```
static void
btn_sample_app_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;

        app_control_h app_control;
        app_control_create(&app_control);
        app_control_set_operation(app_control, APP_CONTROL_OPERATION_DEFAULT);
        app_control_set_app_id (app_control, "org.example.helloworld");

        if (app_control_send_launch_request(app_control, NULL, NULL) == APP_CONT
ROL_ERROR_NONE)
             dlog_print(DLOG_INFO, "tag", "Succeeded to launch a Helloworld app.
");
        else
             dlog_print(DLOG_INFO, "tag", "Failed to launch a calculator app.");

        app_control_destroy(app_control);
}

static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
```

```
    /* put the frame into the box instead of the child directly */
    elm_box_pack_end(box, frame);
    /* show the frame */
    evas_object_show(frame);
}
```

Btn_sample_app_cb() 是 Button 回调函数。

app_control_h 是 AppControl 结构。

app_control_create(app_control_h *) 是创建 AppControl 对象的 API。

app_control_set_operation(app_control_h, char *) 是定义 AppControl 角色的 API。加载外部应用程序时，传入 APP_CONTROL_OPERATION_DEFAULT 即可。

app_control_set_app_id(app_control_h, char *) 是在 AppControl 中定义将要加载示例安装包名称的 API。HelloWorld 示例的安装包名称是“org.example.helloworld”。

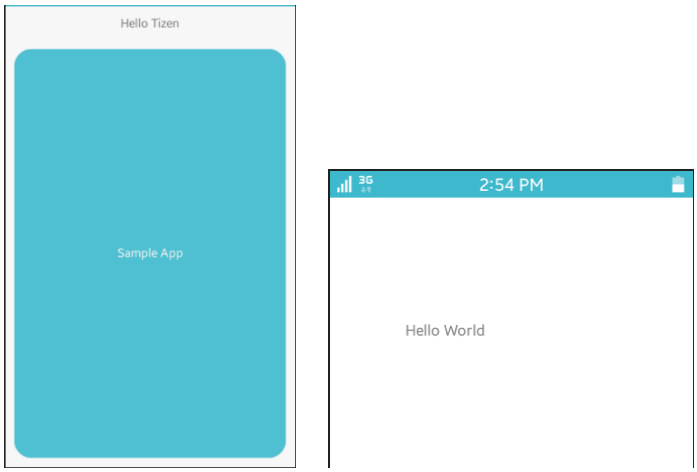
app_control_send_launch_request(app_control_h, app_control_reply_cb, void *) 是运行 AppControl 的 API。

app_control_destroy(app_control_h) 是删除 AppControl 对象的 API。

my_box_pack() 是在 Box 上添加小部件的函数。

运行示例前，首先确认是否已在模拟器中安装 HelloWorld 示例。如果未安装，用 HelloWorld 定义源项目安装在模拟器中。

在模拟器中安装好 HelloWorld 示例后，创建 AppControlEx 示例并运行。点击 Button 运行 HelloWorld 示例。



3）在其他应用程序中传入数据

下面在加载 HelloWorld 示例时传入数据。在 appcontrolex.c 文件的 btn_sample_app_cb() 函数中添加新代码。 ┌[appcontrolex.c]

```
┌──────────────────────────────────────────┐
static void
btn_sample_app_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;

        app_control_h app_control;
        app_control_create(&app_control);
        app_control_set_operation(app_control, APP_CONTROL_OPERATION_DEFAULT);
        app_control_add_extra_data(app_control, "pet", "dog");
        app_control_add_extra_data(app_control, "dessert", "juice");
        app_control_set_app_id (app_control, "org.example.helloworld");

        if (app_control_send_launch_request(app_control, NULL, NULL) == APP_CONT
ROL_ERROR_NONE)
             dlog_print(DLOG_INFO, "tag", "Succeeded to launch a Helloworld app.
");
        else
             dlog_print(DLOG_INFO, "tag", "Failed to launch a calculator app.");

        app_control_destroy(app_control);
}
└──────────────────────────────────────────┘
```

app_control_add_extra_data(app_control_h, char *, char *) 是在 AppControl 中添加数据的 API。参数依次为 AppControl 对象、Key 值、文本数据。

发送方已创建，下面来创建接受方。打开 HelloWorld 示例的源文件（/src/helloworld.c），在上方添加全局变量。 ┌[helloworld.c]

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
} appdata_s;

char recv_data[100];
```

按照 AppControl 添加数据个数调用事件函数。未保存全部数据定义全局变量。

源文件下方有一个名为 app_control() 的函数。这就是 AppControl 事件函数。添加新代码和新函数。

┌[helloworld.c]

```
bool _app_control_extra_data_cb(app_control_h app_control, const char *key, void *data)
{
        int ret;
        char *value;

        ret = app_control_get_extra_data(app_control, key, &value);
        strcat(recv_data, key);
        strcat(recv_data, ":");
        strcat(recv_data, value);
        strcat(recv_data, " / ");

        appdata_s *ad = data;
        elm_object_text_set(ad->label, recv_data);
        return true;
}

static void
```

```
app_control(app_control_h app_control, void *data)
{
        app_control_foreach_extra_data(app_control, _app_control_extra_data_cb,
 data);
}
```

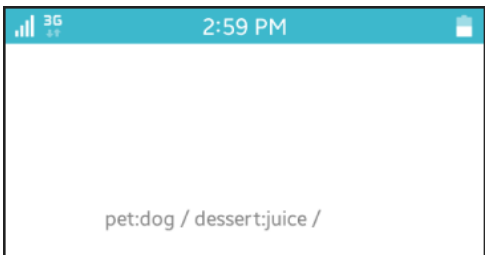
_app_control_extra_data_cb() 是接收单独数据的回调函数。因为已在 AppControlEx 示例中传入 2 个数据，本函数也调用 2 次。

app_control_get_extra_data(app_control_h, char *, char **) 是在 AppControl 中提取数据的 API。在第 2 个参数上传入 Key，第 3 个参数将会返回数据。

app_control() 是已接收 AppControl 对象时运行的回调函数。

app_control_foreach_extra_data(app_control_h, app_control_extra_data_cb, void *) 是定义保存于 AppControl 对象中的单独数据处理函数的 API。

首先在模拟器中安装好 HelloWorld 示例后，创建 AppControlEx 示例并运行。点击 Button 运行 HelloWorld 示例，传入 AppControlEx 示例的数据将显示在 Label 小部件上。



4）照相机 AppControl

下面来了解一下在默认应用程序中加载照相机应用程序的方法。移动至 AppControlEx 示例，在 create_base_gui() 函数末尾创建新 Button。 ┌[appcontrolex.c]

```
            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Sample App");
            evas_object_smart_callback_add(btn, "clicked", btn_sample_app_cb, ad);
            /* epand both horiz and vert, fill horiz and vert */
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            /* Button-2 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Camera");
            evas_object_smart_callback_add(btn, "clicked", btn_camera_cb, ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

创建第 2 个 Button 的回调函数。在 create_base_gui() 函数上添加新函数。

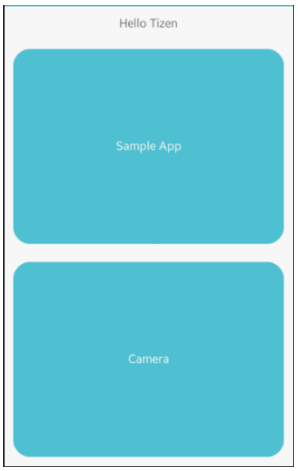
```
static void
btn_camera_cb(void *data, Evas_Object *obj, void *event_info)
{
        app_control_h app_control;

        app_control_create(&app_control);
        app_control_set_operation(app_control, APP_CONTROL_OPERATION_CREATE_CONTENT);
        app_control_set_mime(app_control, "image/jpg");
        if (app_control_send_launch_request(app_control, NULL, NULL) == APP_CONTROL_ERROR_NONE)
             dlog_print(DLOG_INFO, "tag", "Succeeded to launch camera app.");
        else
             dlog_print(DLOG_INFO, "tag", "Failed to launch camera app.");

        app_control_destroy(app_control);
```

```
}
```

app_control_set_operation(app_control_h, char *) 是定义 AppControl 角色的 API。因为照相后生成图片文件，传入 APP_CONTROL_OPERATION_CREATE_CONTENT 即可。

app_control_set_mime(app_control_h, char *) 是定义 MIME 形式的 API。照相机应用程序也可能有多种安装形式，所以不要定义特定应用程序，而需定义 MIME 形式。

重新运行示例，点击第 2 个 Button。运行照相机应用程序。在模拟器中无法运行时，在手机中测试即可。



5）E-Mail AppControl

下面来了解一下在默认应用程序中加载 E-Mail 应用程序的方法。在 create_base_gui() 函数末尾创建新 Button。

```
        /* Button-2 */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Camera");
        evas_object_smart_callback_add(btn, "clicked", btn_camera_cb, ad);
        my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
```

```
            /* Button-3 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "E-mail");
            evas_object_smart_callback_add(btn, "clicked", btn_email_cb, ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

创建第 3 个 Button 的回调函数。在 create_base_gui() 函数上添加新函数。

```
static void
btn_email_cb(void *data, Evas_Object *obj, void *event_info)
{
        app_control_h app_control;
        char *mail_address = "topofsan@naver.com";
        char *subject = "Tutorial message title";
        char *message = "Tutorial message content.";

        app_control_create(&app_control);
        app_control_set_operation(app_control, APP_CONTROL_OPERATION_COMPOSE);
        app_control_set_app_id(app_control, "email-composer-efl");
        app_control_add_extra_data(app_control, APP_CONTROL_DATA_TEXT, message);
        app_control_add_extra_data(app_control, APP_CONTROL_DATA_TO, mail_addres
s);
        app_control_add_extra_data(app_control, APP_CONTROL_DATA_SUBJECT, subjec
t);
        if (app_control_send_launch_request(app_control, NULL, NULL) == APP_CONT
ROL_ERROR_NONE)
             dlog_print(DLOG_INFO, "tag", "Succeeded to launch e-mail app.");
        else
             dlog_print(DLOG_INFO, "tag", "Failed to launch e-mail app.");

        app_control_destroy(app_control);
}
```

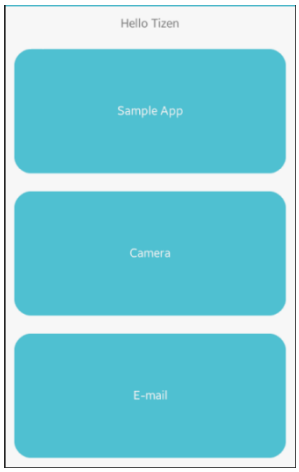
app_control_set_operation(app_control_h, char *) 是定义 AppControl 角色的 API。发送 E-Mail 传入 APP_CONTROL_OPERATION_COMPOSE 即可。

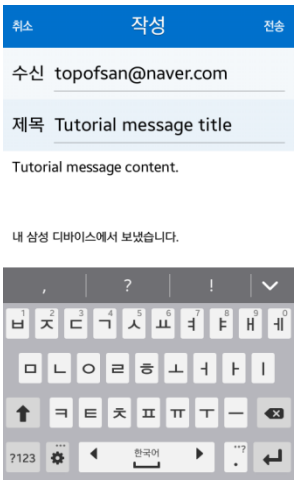
app_control_set_app_id(app_control_h, char *) 是在 AppControl 中定义将要加载示例安装包名称的 API。发送 E-Mail 是“email-composer-efl”。

app_control_add_extra_data(app_control_h, char *, char *) 是在 AppControl 中添加数据的 API。参数依次为 AppControl 对象、Key 值、文本数据。E-Mail 所使用的 Key，如下所示。

- APP_CONTROL_DATA_TEXT：邮件正文内容
- APP_CONTROL_DATA_TO：收件人地址
- APP_CONTROL_DATA_SUBJECT：邮件题目

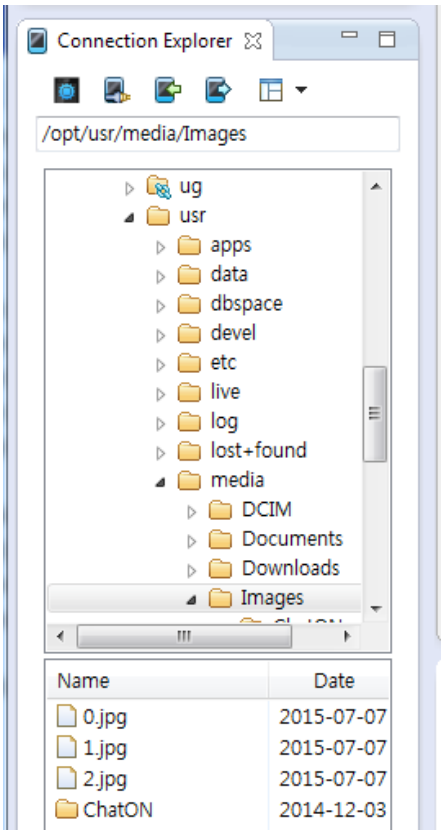
重新运行示例，点击第 3 个 Button。运行 E-Mail 应用程序。在模拟器中无法运行时，在手机中测试即可。





6）Image Viewer AppControl

下面来了解一下在默认应用程序中加载 Image Viewer 应用程序的方法。为了在 Image Viewer 应用程序中显示图片，需要图片文件。首先确认在模拟器 /opt/usr/media/Images 文件夹中是否存在 0.jpg 文件。如果文件不存在，从附录 /Image 文件夹中将 0.jpg 文件复制到终端设备上。在 Eclipse Connection Explorer 中拖放至 /opt/usr/media/Images 文件夹即可。



在 create_base_gui() 函数末尾创建新 Button。

```
        /* Button-3 */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "E-mail");
        evas_object_smart_callback_add(btn, "clicked", btn_email_cb, ad);
        my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

        /* Button-4 */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Image Viewer");
        evas_object_smart_callback_add(btn, "clicked", btn_image_viewer_cb,
ad);
        my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
    }
}
```

创建第 4 个 Button 的回调函数。在 create_base_gui() 函数上添加新函数。

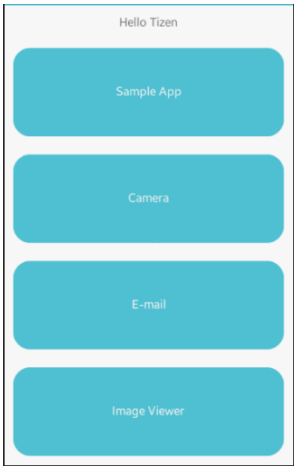
```
static void
btn_image_viewer_cb(void *data, Evas_Object *obj, void *event_info)
{
        app_control_h app_control;
        char buf[PATH_MAX];
        strcat(buf, FM_PHONE_FOLDER);
        strcat(buf, "/Images/0.jpg");

        app_control_create(&app_control);
        app_control_set_operation(app_control, APP_CONTROL_OPERATION_VIEW);
        app_control_set_uri(app_control, buf);
        app_control_set_mime(app_control, "image/*");

        if (app_control_send_launch_request(app_control, NULL, NULL) == APP_CONT
ROL_ERROR_NONE)
             dlog_print(DLOG_INFO, "tag", "Succeeded to launch Image Viewer app.
");
        else
             dlog_print(DLOG_INFO, "tag", "Failed to launch Image Viewer app.");

        app_control_destroy(app_control);
}
```

app_control_set_operation(app_control_h, char *) 是定义 AppControl 角色的 API。加载 Image Viewer 应用程序时，传入 APP_CONTROL_OPERATION_VIEW 即可。

app_control_set_uri(app_control_h, char *) 是定义应用项目路径的 API。定义图片的路径。

app_control_set_mime(app_control_h, char *) 是定义 MIME 形式的 API。Image Viewer 应用程序也可能有多种安装形式所以不要定义特定应用程序，必须定义 MIME 形式。

重新运行示例，点击第 4 个 Button。运行 Image View。



7）网页浏览器 AppControl

下面来了解一下在默认应用程序中加载网页浏览器应用程序的方法。在 create_base_gui() 函数末尾创建新 Button。

```
            /* Button-4 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Image Viewer");
            evas_object_smart_callback_add(btn, "clicked", btn_image_viewer_cb,
ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            /* Button-5 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Web browser");
            evas_object_smart_callback_add(btn, "clicked", btn_web_browser_cb,
ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

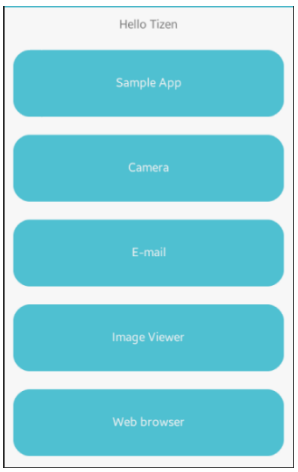
创建第 5 个 Button 的回调函数。在 create_base_gui() 函数上添加新函数。

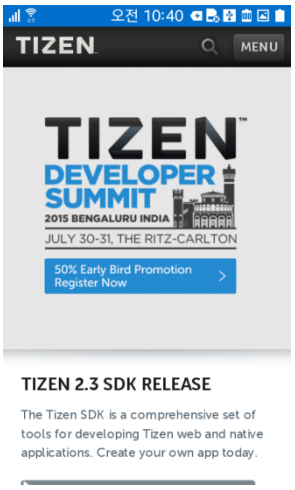
```
static void
btn_web_browser_cb(void *data, Evas_Object *obj, void *event_info)
{
        app_control_h app_control;

        app_control_create(&app_control);
        app_control_set_operation(app_control, APP_CONTROL_OPERATION_DEFAULT);
        app_control_set_app_id(app_control, "com.samsung.browser");
        app_control_set_uri(app_control, "www.tizen.org");
        if (app_control_send_launch_request(app_control, NULL, NULL) == APP_CONT
ROL_ERROR_NONE)
                dlog_print(DLOG_INFO, "tag", "Succeeded to launch browser app.");
        else
                dlog_print(DLOG_INFO, "tag", "Failed to launch browser app.");
        app_control_destroy(app_control);
}
```

app_control_set_app_id(app_control_h, char *) 是在 AppControl 中定义将要加载示例安装包名称的 API。默认网页浏览器应用程序的安装包名称是“com.samsung.browser”。

app_control_set_uri(app_control_h, char *) 是定义应用项目路径的 API。定义 Tizen 网页地址。

重新运行示例，点击第 5 个 Button。运行网页。





8）相关 API

int app_control_create(app_control_h *)：创建 AppControl 对象的 API。

int app_control_set_operation(app_control_h, char *)：定义 AppControl 角色的 API。加载外部应用程序时，传入 APP_CONTROL_OPERATION_DEFAULT 即可。

int app_control_set_app_id(app_control_h, char *)：在 AppControl 中定义将要加载示例安装包名称的 API。 HelloWorld 示例的安装包名称是“org.example.helloworld”。

int app_control_send_launch_request(app_control_h, app_control_reply_cb, void *)：运行 AppControl 的 API。

int app_control_destroy(app_control_h)：删除 AppControl对象的 API。

int app_control_add_extra_data(app_control_h, char *, char *)：在 AppControl 中添加数据的 API。参数依次为 AppControl 对象、Key 值、文本数据。

int app_control_get_extra_data(app_control_h, char *, char **)：在 AppControl 中提取数据的 API。在第 2 个参数上传入 Key，第 3 个参数将会返回数据。

int app_control_foreach_extra_data(app_control_h, app_control_extra_data_cb, void *)：定义保存于 AppControl 对象中的单独数据处理函数的 API。

int app_control_set_operation(app_control_h, char *)：定义 AppControl 角色的 API。因为照相后生成图片文件，传入 APP_CONTROL_OPERATION_CREATE_CONTENT 即可。

int app_control_set_mime(app_control_h, char *)：定义 MIME 形式的 API。

int app_control_set_operation(app_control_h, char *)：定义 AppControl 角色的 API。发送 E-Mail 传入 APP_CONTROL_OPERATION_COMPOSE 即

可。加载 Image Viewer 应用程序时，传入 APP_CONTROL_OPERATION_VIEW 即可。

int app_control_set_app_id(app_control_h, char *)：在 AppControl 中定义将要加载示例安装包名称的 API。发送电子邮件是“email-composer-efl”。默认网页浏览器应用程序的安装包名称是“com.samsung.browser”。

int app_control_add_extra_data(app_control_h, char *, char *)：在 AppControl 中添加数据的 API。参数依次为 AppControl 对象、Key 值、文本数据。E-Mail 所使用的 Key，如下所示。

- APP_CONTROL_DATA_TEXT：邮件正文内容
- APP_CONTROL_DATA_TO：收件人地址
- APP_CONTROL_DATA_SUBJECT：邮件题目

int app_control_set_uri(app_control_h, char *)：定义应用项目路径的 API。

63. 制作服务应用程序

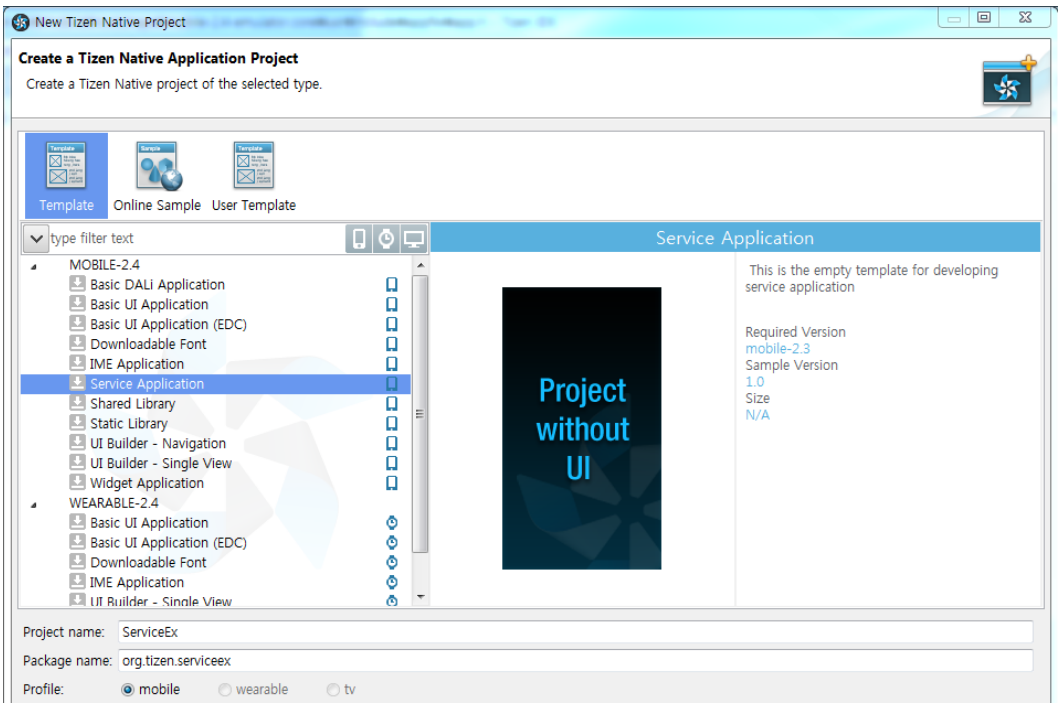
服务应用程序是没有用户直观 UI，在后台运行的应用程序。服务应用程序多以杀毒软件、防盗功能、通信功能等形式出现。

1）创建服务源项目

创建新源项目。选择 Eclipse 主菜单 [File > New Tizen Native Project]。

出现弹窗后，选择 [Template > MOBILE-2.x > Service Application]。

将 Project name 项命名为 ServiceEx，点击 Finish 键。



下面来展示一下运行服务应用程序 5 秒后，自动终止的功能。打开位于 src 文件夹的源文件 （~.c），添加 define 常量和变量。

```
#include <service_app.h>
#include "serviceex.h"
#include <ecore.h>

Ecore_Timer *timer1;
int timer_count = 0;
```

Ecore_Timer 是计时器结构。

在 timer_count 中保存计时器事件发生次数。

运行服务应用程序后，自动运行计时器。service_app_create() 是启动服务应用程序时的事件函数。service_app_terminate() 是服务应用程序终止时的事件函数。添加新代码。

```
bool service_app_create(void *data)
{
        dlog_print(DLOG_DEBUG, "tag", "%s", __func__);
        timer_count = 0;
        timer1 = ecore_timer_add(1.0, timer1_cb, NULL);
        return true;
}

void service_app_terminate(void *data)
{
        dlog_print(DLOG_DEBUG, "tag", "%s", __func__);
        return;
}
```

在运行和终止应用程序时，显示 Log 信息。运行应用程序时，一起运行计时器。

创建计时器事件回调函数。在 service_app_create() 函数上添加新函数。

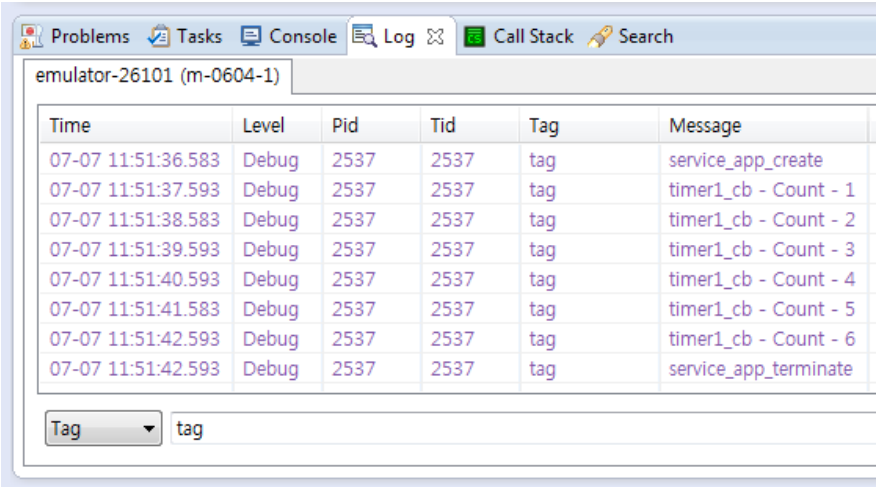
```
static Eina_Bool
timer1_cb(void *data EINA_UNUSED)
{
        timer_count ++;
        char buf[100];
```

```
        sprintf(buf, "Count - %d", timer_count);
        dlog_print(DLOG_DEBUG, "tag", "%s - %s", __func__, buf);

        if( timer_count > 5 )
                service_app_exit();
        return ECORE_CALLBACK_RENEW;
}
```

timer1_cb() 是计时器事件函数。1 秒运行一次，显示 Log 信息，5 秒后应用程序自动终止。

service_app_exit() 是终止服务应用程序的 API。

构建并运行示例。在模拟器中没有任何变化。这是因为服务应用程序没有直观的 UI 界面。那么就来确认一下 Log 信息。选择 Log 面板下组合框内的 Tag 选项，在右侧编辑框内输入 tag。显示应用程序起始信息与 6 次计数以及应用程序终止信息。



2）在外部应用程序中用服务应用程序传入事件

下面来展示一下在外部应用程序中传入事件，终止服务应用程序的功能。与在 AppControlEx 示例中向 HelloWorld 示例传入数据的方法一致。

在源文件中有一个名为 service_app_control() 的 AppControl 函数。在这个函数中添加新代码并在其上创建新函数。

```
bool _app_control_extra_data_cb(app_control_h app_control, const char *key, voi
d *data)
{
        int ret;
        char *value;

        ret = app_control_get_extra_data(app_control, key, &value);
        dlog_print(DLOG_DEBUG, "tag", "%s - %s : %s", __func__, key, value);
        if( strcmp(key, "dessert") == 0 && strcmp(value, "juice") == 0 )
        {
                dlog_print(DLOG_DEBUG, "tag", "Close message received");
                service_app_exit();
        }
        return true;
}

void service_app_control(app_control_h app_control, void *data)
{
        dlog_print(DLOG_DEBUG, "tag", "%s", __func__);
        app_control_foreach_extra_data(app_control, _app_control_extra_data_cb,
 NULL);
        return;
}
```

_app_control_extra_data_cb() 是接收单独数据的回调函数。因为已在 AppControlEx 示例中保存 2 个数据，本函数也调用 2 次。

app_control_get_extra_data(app_control_h, char *, char **) 是在 AppControl 中提取数据的 API。在第 2 个参数上传入 Key，第 3 个参数将会返回数据。

service_app_control() 是已接收 AppControl 对象时运行的回调函数。

app_control_foreach_extra_data(app_control_h, app_control_extra_data_cb, void *) 是定义保存于 AppControl 对象中的单独数据处理函数的 API。

将 timer1_cb() 函数的代码修改如下。因为经过 5 秒后服务应用程序自动终止，定义更长时间。

```
static Eina_Bool
timer1_cb(void *data EINA_UNUSED)
{
        timer_count ++;
        char buf[100];
        sprintf(buf, "Count - %d", timer_count);
        dlog_print(DLOG_DEBUG, "tag", "%s - %s", __func__, buf);

        //if( timer_count > 5 )
        if( timer_count > 50 )
                service_app_exit();
        return ECORE_CALLBACK_RENEW;
}
```

下面来展示一下在外部应用程序中调用应用程序的功能。打开此前创建的 AppControlEx 示例源文件。同时在 create_base_gui() 函数末尾添加 Button 创建代码。 ┌[appcontrolex.c]

```
            /* Button-5 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Web browser");
            evas_object_smart_callback_add(btn, "clicked", btn_web_browser_cb, a
d);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            /* Button-6 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Service close");
            evas_object_smart_callback_add(btn, "clicked", btn_service_close_cb,
 ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

在 create_base_gui() 上创建 Button 回调函数。

```
static void
btn_service_close_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;

        app_control_h app_control;
        app_control_create(&app_control);
        app_control_set_operation(app_control, APP_CONTROL_OPERATION_DEFAULT);
        app_control_add_extra_data(app_control, "pet", "dog");
        app_control_add_extra_data(app_control, "dessert", "juice");
        app_control_set_app_id (app_control, "org.example.serviceex");

        if (app_control_send_launch_request(app_control, NULL, NULL) == APP_CONT
ROL_ERROR_NONE)
             dlog_print(DLOG_INFO, "tag", "Succeeded to launch a Service app.");
        else
             dlog_print(DLOG_INFO, "tag", "Failed to launch a Service app.");

        app_control_destroy(app_control);
}
```

app_control_h 是 AppControl 结构。

app_control_create(app_control_h *) 是创建 AppControl 对象的 API。

app_control_set_operation(app_control_h, char *) 是定义 AppControl 角色的 API。加载外部应用程序时，传入 APP_CONTROL_OPERATION_DEFAULT 即可。

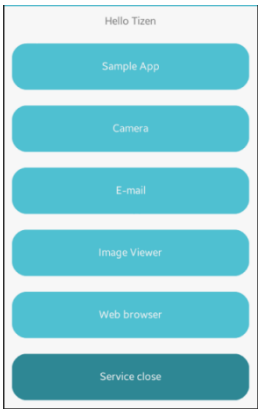
app_control_add_extra_data(app_control_h, char *, char *) 是在 AppControl 中添加数据的 API。参数依次为 AppControl 对象、Key 值、文本数据。

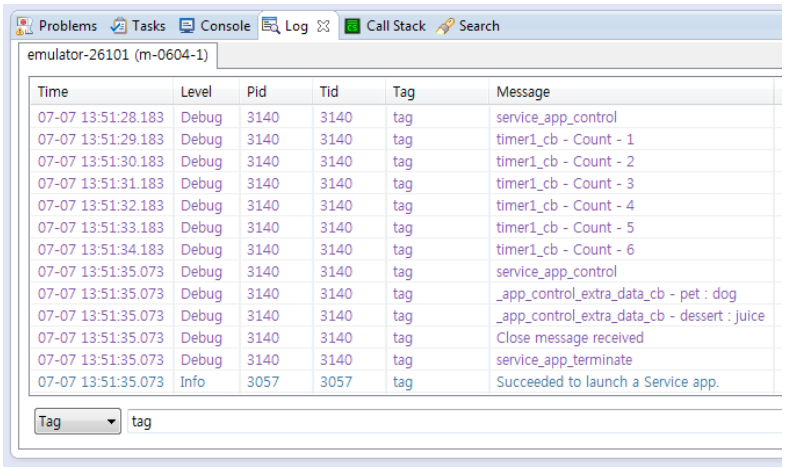
app_control_set_app_id(app_control_h, char *) 是在 AppControl 中定义将要加载示例安装包名称的 API。

app_control_send_launch_request(app_control_h, app_control_reply_cb, void *) 是运行 AppControl 的 API。

app_control_destroy(app_control_h) 是删除 AppControl 对象的 API。

首先运行服务应用程序后，运行 AppControlEx 示例。点击第 6 个 Button，在服务应用程序中终止 Log 信息。服务应用程序被终止。





3）相关 API

app_control_h：AppControl 结构。

int app_control_create(app_control_h *)：创建 AppControl 对象的 API。

int app_control_set_operation(app_control_h, char *)：定义 AppControl 角色的 API。加载外部应用程序时，传入 APP_CONTROL_OPERATION_DEFAULT 即可。

int app_control_add_extra_data(app_control_h, char *, char *)：在 AppControl 中添加数据的 API。参数依次为 AppControl 对象、Key 值、文本数据。

int app_control_set_app_id(app_control_h, char *)：在 AppControl 中定义将要加载示例安装包名称的 API。

int app_control_send_launch_request(app_control_h, app_control_repl

y_cb, void *)：运行 AppControl 的 API。

int app_control_destroy(app_control_h)：删除 AppControl 对象的 API。

int app_control_get_extra_data(app_control_h, char *, char **)：在 AppControl 中提取数据的 API。在第 2 个参数上传入 Key，第 3 个参数将会返回数据。

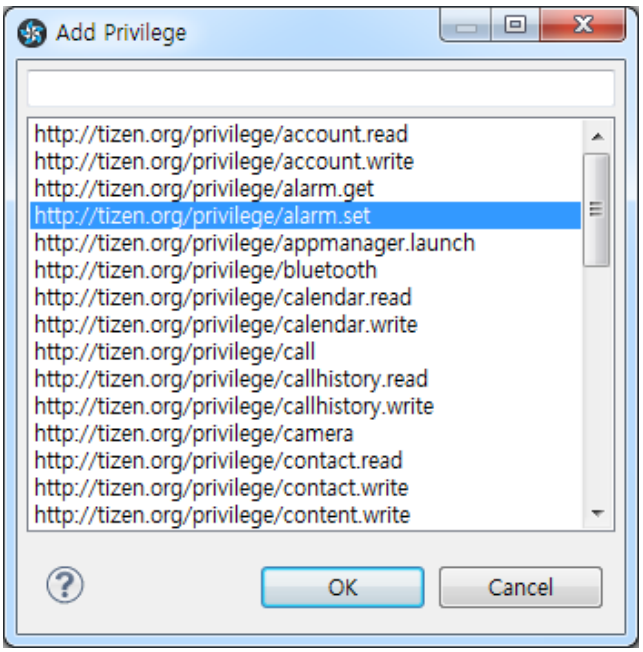
int app_control_foreach_extra_data(app_control_h, app_control_extra_data_cb, void *)：定义保存于 AppControl 对象中的单独数据处理函数的 API。

64. Alarm - 在定义时间内运行应用程序

开发闹铃等提醒应用程序时，即使应用程序处于终止状态，到达指定时间时必须能够自动运行应用程序。利用 Alarm 可以让应用程序在规定时间内运行。利用配有的计时器功能也可以在经过一定时间后运行应用程序。使用方法与 AppControl 类似。

1）登录权限

创建新的源项目，将 Project name 命名为Alarm。为了使用 Alarm，需拥有用户权限。创建源项目之后，打开 tizen-manifest.xml 文件，点击下方选项卡按键中的 Privileges。然后点击右侧上端的 Add 键。出现弹窗后，从目录中选择 http://tizen.org/privilege/alarm.set，点击 OK 键关闭弹窗。



保存后点击下方选项卡按键中，位于右侧末端的 tizen-manifest.xml 键，显示 xml 文件源代码。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.alarm" version="1.0.0">
        <profile name="mobile"/>
        <ui-application appid="org.example.alarm" exec="alarm" multiple="false" nodisplay="false" taskmanage="true" type="capp">
                <label>alarm</label>
                <icon>alarm.png</icon>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/alarm.set</privilege>
        </privileges>
</manifest>
```

2）启动 Timer Alarm

下面来展示一下点击 Button 3 秒后，运行 HelloWorld 示例的功能。如果在模拟器中未安装 HelloWorld 示例，首先安装 HelloWorld 示例。打开 src 文件夹内的源文件 （~.c），添加库头文件和变量。

```
#include "alarm.h"
#include <app_alarm.h>
#include <time.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        int timer_id;
        int date_id;
} appdata_s;
```

app_alarm.h 是使用 Alarm 的库头文件。

time.h 是使用与时间有关 API 的库头文件。

在 timer_id 中保存 Timer Alarm 的 ID。可利用它来终止 Alarm。

在 timer_id 中保存 Date Alarm 的 ID。可利用它来终止 Alarm。

在 create_base_gui() 函数上添加新函数。在 Box 上添加小部件的函数。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
            double h_weight, double v_weight, double h_align, double v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding style. there is "pad_small", "pad_medium",
    * "pad_large" and "pad_huge" available as styles in addition to the
    * "default" frame style */
   elm_object_style_set(frame, "pad_medium");
   /* set the input weight/aling on the frame insted of the child */
   evas_object_size_hint_weight_set(frame, h_weight, v_weight);
   evas_object_size_hint_align_set(frame, h_align, v_align);
     {
        /* tell the child that is packed into the frame to be able to expand */
        evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPA
ND);
        /* fill the expanded area (above) as opposaed to center in it */
        evas_object_size_hint_align_set(child, EVAS_HINT_FILL, EVAS_HINT_FILL);
        /* actually put the child in the frame and show it */
        evas_object_show(child);
        elm_object_content_set(frame, child);
     }
   /* put the frame into the box instead of the child directly */
   elm_box_pack_end(box, frame);
   /* show the frame */
   evas_object_show(frame);
}
```

create_base_gui() 函数末尾添加 Button 创建代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);
```

```
    {
        /* child object - indent to how relationship */
        /* A box to put things in verticallly - default mode for box */
        Evas_Object *box = elm_box_add(ad->win);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* child object - indent to how relationship */
            /* Label*/
            ad->label = elm_label_add(ad->conform);
            elm_object_text_set(ad->label, "<align=center>Hello Tizen</>");
            /* expand horizontally but not vertically, and fill horiz,
            * align center vertically */
            my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);

            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Start Timer Alarm");
            evas_object_smart_callback_add(btn, "clicked", btn_start_timer_cb,
ad);
            /* epand both horiz and vert, fill horiz and vert */
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

在 create_base_gui() 上创建 Button 回调函数。

```
static void
btn_start_timer_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        int ret;
        int DELAY = 3;
        int REMIND = 0;

        app_control_h app_control = NULL;
```

```
        ret = app_control_create(&app_control);
        ret = app_control_set_operation(app_control, APP_CONTROL_OPERATION_DEFAU
LT);
        ret = app_control_set_app_id (app_control, "org.example.helloworld");
        ret = alarm_schedule_after_delay(app_control, DELAY, REMIND, &ad->timer_
id);
        dlog_print(DLOG_DEBUG, "tag", "result = %d", ret);
        elm_object_text_set(ad->label, "Timer Alarm Started");
}
```

app_control_h 是 AppControl 结构。

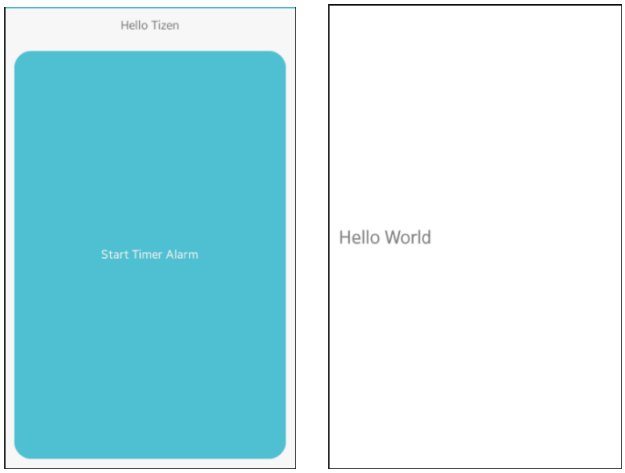
app_control_create(app_control_h *) 是创建 AppControl 对象的 API。

app_control_set_operation(app_control_h, char *) 是定义 AppControl 角色的 API。加载外部应用程序时，传入 APP_CONTROL_OPERATION_DEFAULT 即可。

app_control_set_app_id(app_control_h, char *) 是在 AppControl 中定义将要加载示例安装包名称的 API。HelloWorld 示例的安装包名称是“org.example.helloworld”。

alarm_schedule_after_delay(app_control_h, int, int, int *) 是经过一定时间后传入 AppControl 事件的 API。第 2 个参数是第 1 次运行的时间间隔，第 3 个参数是重新运行的时间间隔。定为 0 后，AppControl 事件仅运行一次。第 4 个参数中将会显示 Alarm 的 ID。用于终止 Alarm。

构建并运行示例。点击 Button，3 秒后运行 HelloWorld 示例。



3）终止 Timer Alarm

下面来展示一下在 Alarm 中添加 Button，终止 Alarm 的功能。在 create_base_gui() 函数末尾添加新代码。

```
            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Start Timer Alarm");
            evas_object_smart_callback_add(btn, "clicked", btn_start_timer_cb, ad);
            /* epand both horiz and vert, fill horiz and vert */
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            /* Button-2 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Stop Timer Alarm");
            evas_object_smart_callback_add(btn, "clicked", btn_stop_timer_cb, ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

同时在 create_base_gui() 上创建 Button 回调函数，修改 btn_start_timer_cb() 函数。

```
static void
btn_start_timer_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        int ret;
        int DELAY = 3;
        //int REMIND = 0;
        int REMIND = 8;

        app_control_h app_control = NULL;
        ret = app_control_create(&app_control);
        ret = app_control_set_operation(app_control, APP_CONTROL_OPERATION_DEFAU
LT);
        ret = app_control_set_app_id (app_control, "org.example.helloworld");
        ret = alarm_schedule_after_delay(app_control, DELAY, REMIND, &ad->timer_
id);
        dlog_print(DLOG_DEBUG, "tag", "result = %d", ret);
        elm_object_text_set(ad->label, "Timer Alarm Started");
}

static void
btn_stop_timer_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        alarm_cancel(ad->timer_id);
        elm_object_text_set(ad->label, "Timer Alarm Stopped");
}
```

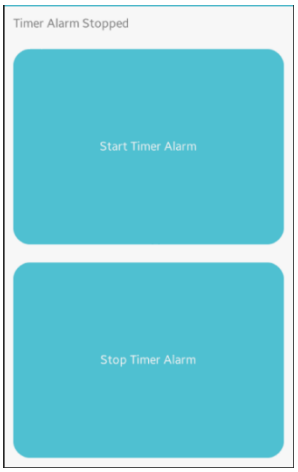
将 REMIND 变量的值改为 8，HelloWorld 示例每 8 秒重新运行。

btn_stop_timer_cb() 是用户点击 Button 时，终止 Timer Alarm 的函数。

alarm_cancel(int alarm_id) 是终止 Alarm 的 API.传入 Alarm 的 ID。

重新运行示例，点击第 1 个 Button。运行 HelloWorld 示例。HelloWorld 示例终止后，每 8 秒重新运行。点击第 2 个 Button，HelloWorld 示例将不再运行。Alarm 将被终止。

如果想要终止 Alarm，需要 ID，因此在 Alarm 开始后，必须点击第 2 个 Button，终止 Alarm。如果即使按下 Stop 键，HelloWorld 示例仍然运行，将模拟器删除，重新创建即可初始化。



3）Date Alarm

下面来展示一下定义特定时间，按个人意愿在所需时间内运行特定应用程序的功能。在 create_base_gui() 函数中添加 2 个 Button 创建代码。

```
            /* Button-2 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Stop Timer Alarm");
            evas_object_smart_callback_add(btn, "clicked", btn_stop_timer_cb, a
d);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            /* Button-3 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Start Date Alarm");
            evas_object_smart_callback_add(btn, "clicked", btn_start_date_cb, a
d);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            /* Button-4 */
            btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Stop Date Alarm");
            evas_object_smart_callback_add(btn, "clicked", btn_stop_date_cb, a
d);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

创建第 3 个、第 4 个 Button 的代码。然后在 create_base_gui() 上创建 Button 回调函数。

```
static void
btn_start_date_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        int ret;

        struct tm date;
        ret = alarm_get_current_time(&date);
        date.tm_sec+=4;

        app_control_h app_control = NULL;
        ret = app_control_create(&app_control);
        ret = app_control_set_operation(app_control, APP_CONTROL_OPERATION_DEFAU
LT);
        ret = app_control_set_app_id (app_control, PACKAGE);
        ret = alarm_schedule_at_date(app_control, &date, 0, &ad->date_id);
        elm_object_text_set(ad->label, "Date Alarm Started");
}

static void
btn_stop_date_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        alarm_cancel(ad->date_id);
        elm_object_text_set(ad->label, "Date Alarm Stopped");
}
```

btn_start_date_cb() 是调用 Date Alarm 的函数。

struct tm 是可以保存日期和时间的时间结构。

alarm_get_current_time(struct tm *) 是以 struct tm 形式显示当前时间的 API。

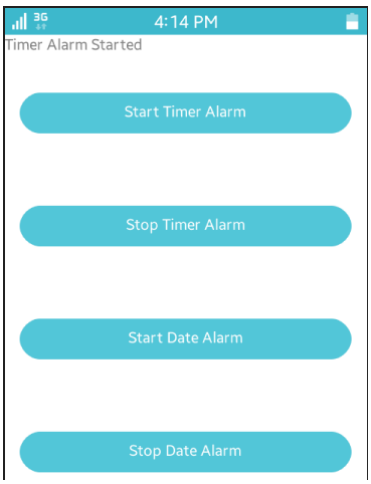
在 tm.tm_sec 中以秒为单位保存时间数据。将时间定义为从现在开始 4 秒后。

alarm_schedule_at_date(app_control_h, struct tm *, int, int *) 是在特定时间内运行 AppControl 事件的 API。在第 2 个参数中传入保存容器和时间的 struct tm 对象，在第 3 个参数内定义重新运行的时间间隔。传入 0 即为仅运行 1 次。第 4 个参数中将会显示 Alarm 的 ID。用于终止 Alarm。

btn_start_date_cb() 是终止 Date Alarm 的函数。

让我们再运行一次示例。点击第 3 个 Button，终止应用程序，稍后 Alarm 示例重新运行。

这一次点击第 3 个 Button 后，直接点击第 4 个 Button。终止应用程序后，即使时间流逝也没有任何变化。Date Alarm 已被终止。



4）相关 API

app_control_h：AppControl 结构。

int app_control_create(app_control_h *)：创建 AppControl 对象的 API。

int app_control_set_operation(app_control_h, char *)：定义 AppControl 角色的 API。加载外部应用程序时，传入 APP_CONTROL_OPERATION_DEFAULT 即可。

int app_control_set_app_id(app_control_h, char *)：在 AppControl 中定义将要加载示例安装包名称的 API。HelloWorld 示例的安装包名称是“org.example.helloworld”。

int alarm_schedule_after_delay(app_control_h, int, int, int *)：经过一定时间后传入 AppControl 事件的 API。第 2 个参数是第 1 次运行的时间间隔，第 3 个参数是重新运行的时间间隔。定为 0 后，AppControl 事件仅运行一次。第 4 个参数中将会显示 Alarm 的 ID。用于终止 Alarm。

int alarm_cancel(int alarm_id)：终止 Alarm 的 API。传入 Alarm 的 ID。

struct tm：可以保存日期和时间的时间结构。

int alarm_get_current_time(struct tm *)：以 struct tm 形式显示当前时间的 API。

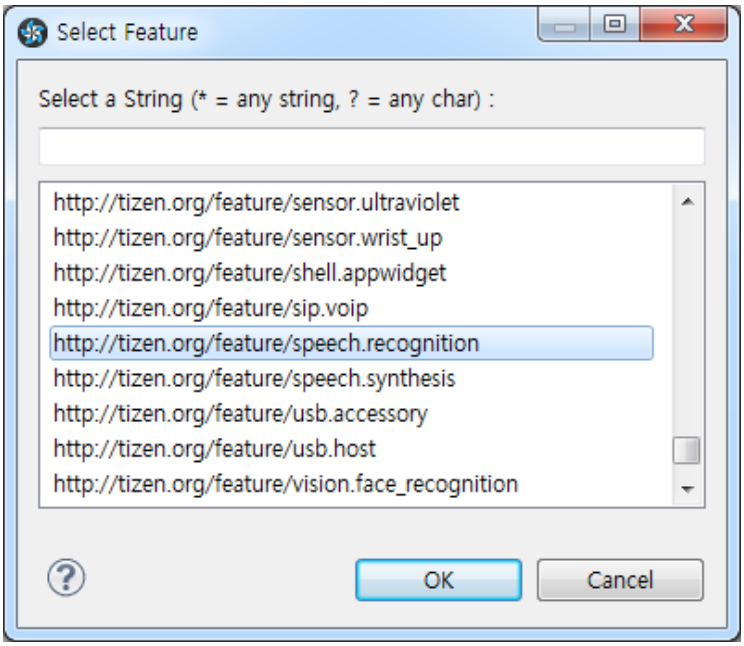
int alarm_schedule_at_date(app_control_h, struct tm *, int, int *)：在特定时间内运行 AppControl 事件的 API。在第 2 个参数中传入保存容器和时间的 struct tm 对象，在第 3 个参数内定义重新运行的时间间隔。传入 0 即为仅运行1次。第 4 个参数中将会显示 Alarm 的 ID。用于终止 Alarm。

65. TTS (Text to Speech)

用人声读取字符的功能叫做 TTS （Text to Speach），反之将人声识别为字符形式的功能叫做 STT （Speach to Text）。下面通过本示例来了解一下 TTS 的使用方法。

1）添加功能

创建新的源项目，将 Project name 命名为 TtsEx。为了使用 TTS 功能，需要登录 Feature。创建源项目之后，打开 tizen-manifest.xml 文件，点击下方选项卡按键中的 Features。然后点击右侧上端的 Add 键。出现弹窗后，从目录中选择 http://tizen.org/feature/speech.recognition，点击 OK 键关闭弹窗。



重复相同过程添加以下功能。

- http://tizen.org/feature/speech.synthesis

保存后点击下方选项卡按键中，位于右侧末端的 tizen-manifest.xml 键，显示 xml 文件源代码。

```
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.ttsex" version="1.0.0">
        <profile name="mobile"/>
        <ui-application appid="org.example.ttsex" exec="ttsex" multiple="false" nodisplay="false" taskmanage="true" type="capp">
                <label>ttsex</label>
                <icon>ttsex.png</icon>
        </ui-application>
        <feature name="http://tizen.org/feature/speech.recognition">true</feature>
        <feature name="http://tizen.org/feature/speech.synthesis">true</feature>
</manifest>
```

2）创建并删除 TTS

打开 src 文件夹内的源文件 （~.c），添加库头文件和变量。

```
#include "ttsex.h"
#include <tts.h>

typedef struct appdata {
    Evas_Object *win;
    Evas_Object *conform;
    Evas_Object *label;
    Evas_Object *entry;
    Evas_Object *button;
    tts_h tts;
} appdata_s;
```

tts.h 是使用 TTS 的库头文件。

在 entry 小部件中输入字符串。

点击 Button 小部件，开始播放字符串。

tts_h 是 TTS 结构。

下面来展示一下运行应用程序后，自动创建 TTS 对象，终止应用程序时，删除 TTS 对象的功能。在 create_base_gui() 上创建 5 个新函数。

```
static void
state_changed_cb(tts_h tts, tts_state_e previous, tts_state_e current, void* use
r_data)
{
    appdata_s *ad = user_data;

    switch (current)
    {
    case TTS_STATE_PLAYING:
        elm_object_text_set(ad->button, "Stop");
        break;
    case TTS_STATE_READY:
    default:
        elm_object_text_set(ad->button, "Play");
        break;
    }
}

static void
utterance_completed_cb(tts_h tts, int utt_id, void *user_data)
{
    appdata_s *ad = user_data;

    dlog_print(DLOG_INFO, LOG_TAG, "Utterance completed: %d", utt_id);
    elm_object_text_set(ad->button, "Stop (idle)");
}

static void
utterance_started_cb(tts_h tts, int utt_id, void *user_data)
{
    appdata_s *ad = user_data;

    dlog_print(DLOG_INFO, LOG_TAG, "Utterance started: %d", utt_id);
    elm_object_text_set(ad->button, "Stop (speaking)");
}

static tts_h
create_tts_handle(appdata_s *ad)
{
    tts_h tts;
    int ret = tts_create(&tts);
```

```
    if (TTS_ERROR_NONE != ret)
    {
        dlog_print(DLOG_INFO, "tag", "%s err = %d", __func__, ret);
    }
    else
    {
        tts_set_utterance_started_cb(tts, utterance_started_cb, ad);
        tts_set_utterance_completed_cb(tts, utterance_completed_cb, ad);
        tts_set_state_changed_cb(tts, state_changed_cb, ad);
        tts_prepare(tts);
    }

    return tts;
}

static void
destroy_tts_handle(tts_h tts)
{
    int ret = tts_destroy(tts); // tts is the TTS handle
    if (TTS_ERROR_NONE != ret)
    {
        dlog_print(DLOG_INFO, "tag", "%s err = %d", __func__, ret);
    }
}
```

state_changed_cb() 是变更 TTS 状态的事件函数。参数依次为 TTS 对象、先前状态值、当前状态值、用户数据。将状态值转换为字符串显示在屏幕上。

tts_state_e 是保存 TTS 状态信息的 INT 型变量。状态种类如下所示。

- TTS_STATE_CREATED：创建 TTS
- TTS_STATE_READY：准备完毕
- TTS_STATE_PLAYING：TTS 播放中
- TTS_STATE_PAUSED：暂停

utterance_completed_cb() 是结束播放 TTS 的事件函数。

utterance_started_cb() 是开始播放 TTS 的事件函数。

create_tts_handle() 是创建 TTS 对象返回的函数。

tts_create(tts_h*) 是创建 TTS 对象的 API。

tts_set_utterance_started_cb() 是定义开始播放 TTS 事件回调函数名称的 API。

tts_set_utterance_completed_cb() 是定义结束播放 TTS 事件回调函数名称的 API。

tts_set_state_changed_cb(tts_h, tts_state_changed_cb, void*) 是定义 TTS 状态变更事件回调函数名称的 API。参数依次为 TTS 对象、回调函数名称、用户数据。

tts_prepare(tts_h) 是转换 TTS 对象准备状态的 API。

destroy_tts_handle() 是删除 TTS 对象的函数。

tts_destroy(tts_h) 是删除 TTS 对象的 API。

上述函数在运行和终止应用程序时调用即可。

在 create_base_gui() 上创建 2 个新函数。

```
static void
btn_play_cb(void *data, Evas_Object *obj, void *event_info)
{
}

static void
my_box_pack(Evas_Object *box, Evas_Object *child, double h_weight, double v_weig
ht,
double h_align, double v_align)
{
        /* create a frame we shall use as padding around the child widget */
        Evas_Object *frame = elm_frame_add(box);
        /* use the medium padding style. there is "pad_small", "pad_medium",
         * "pad_large" and "pad_huge" available as styles in addition to the
```

btn_play_cb() 是 Button 事件函数。稍后会输入函数内容。

my_box_pack() 是在 Box 上添加小部件的函数。

在 create_base_gui() 函数中添加新代码。创建 Box、Entry、Button 的代码。同时调用创建 TTS 对象的函数。

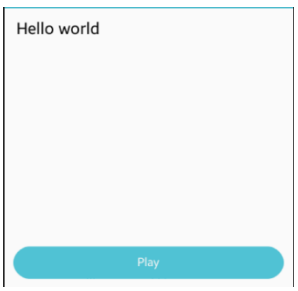
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    {
        Evas_Object *btn, *box;

        /* Container: standard box */
        box = elm_box_add(ad->win);
        elm_box_homogeneous_set(box, EINA_TRUE);
        elm_box_horizontal_set(box, EINA_FALSE);
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        evas_object_size_hint_align_set(box, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(ad->conform, box);
        evas_object_show(box);

        {
            /* Entry */
            ad->entry = elm_entry_add(box);
            elm_entry_single_line_set(ad->entry, EINA_FALSE);
            elm_entry_scrollable_set(ad->entry, EINA_TRUE);
            elm_object_text_set(ad->entry, "Hello world");
            my_box_pack(box, ad->entry, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVA
S_HINT_FILL, EVAS_HINT_FILL);

            /* Button-1 */
            btn = elm_button_add(box);
            elm_object_text_set(btn, "Play/Stop");
            evas_object_smart_callback_add(btn, "clicked", btn_play_cb, ad);
            my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVAS_HINT
_FILL, 0.0);
            ad->button = btn;
        }
    }
```

```
    /* Show window after base gui is set up */
    evas_object_show(ad->win);

ad->tts = create_tts_handle(ad);
}
```

在 app_terminate() 函数中也添加新代码。应用程序结束时将删除 TTS 对象。

```
static void
app_terminate(void *data)
{
        appdata_s *ad = data;
        destroy_tts_handle(ad->tts);
}
```

构建并运行示例。创建 TTS 对象，在 Ready 状态下，Button 捕获文本将转化为播放模式。现在就可以使用了。



3）将字符串转换为语音

下面来展示一下在 Entry 小部件上输入字符串，点击 Button 键，将字符串转换为语音播放的功能。

在 create_base_gui() 函数上添加 2 个新函数。同时在 Button 事件函数中输入内容。

```
static void
add_text(tts_h tts, appdata_s *ad)
{
    const char* text = "Good morning"; // Text for read
    const char* language = "en_US"; // Language
    int voice_type = TTS_VOICE_TYPE_FEMALE; // Voice type
    int speed = TTS_SPEED_AUTO;
    int utt_id; // Utterance ID for the requested text
    text = elm_object_text_get(ad->entry);

    int ret = tts_add_text(tts, text, language, voice_type, speed, &utt_id);
    if (TTS_ERROR_NONE != ret)
    {
        dlog_print(DLOG_INFO, "tag", "%s err = %d", __func__, ret);
    }
}

static int
get_state(tts_h* tts)
{
    tts_state_e current_state;
    int ret;
    ret = tts_get_state(*tts, &current_state);

    if (TTS_ERROR_NONE != ret)
    {
        dlog_print(DLOG_INFO, "tag", "%s state = %d", __func__, ret);
        return -1;
    }
    else
    {
        dlog_print(DLOG_INFO, "tag", "%s state = %d", __func__, current_state);
        return (int) current_state;
    }
}

static void
btn_play_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    int state = get_state(&ad->tts);
    if ((tts_state_e) state == TTS_STATE_READY || (tts_state_e) state == TTS_ST
ATE_PAUSED)
    {
        add_text(ad->tts, ad);
```

```
        int ret = tts_play(ad->tts);
        if (TTS_ERROR_NONE != ret)
        {
            dlog_print(DLOG_INFO, "tag", "%s err = %d", __func__, ret);
        }
    }
    else if ((tts_state_e) state == TTS_STATE_PLAYING)
    {
        int ret = tts_stop(ad->tts);
        if (TTS_ERROR_NONE != ret)
        {
            dlog_print(DLOG_INFO, "tag", "%s err = %d", __func__, ret);
        }
    }
}
```

add_text() 是在 TTS 对象中添加字符串的函数。

tts_add_text(tts_h, char*,char*, int, int, int*) 是在 TTS 对象中添加字符串的 API。参数依次为 TTS 对象、字符串、声音种类、速度。

声音种类如下所示。

- TTS_VOICE_TYPE_AUTO：声音种类自动
- TTS_VOICE_TYPE_MALE：男人声音
- TTS_VOICE_TYPE_FEMALE：女人声音
- TTS_VOICE_TYPE_FEMALE：儿童声音

get_state() 是显示 TTS 当前状态的函数。

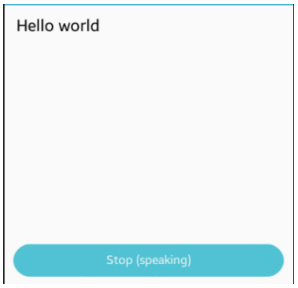
tts_get_state(tts_h tts, tts_state_e* state) 是返回 TTS 当前状态的 API。

btn_play_cb() 是点击 Button 时，启动 TTS 的回调函数。当前状态准备完毕或者暂停时，开始播放，在播放过程中停止。

tts_play(tts_h tts) 是开始播放 TTS 的 API。

tts_stop(tts_h tts) 是停止播放 TTS 的 API。

让我们再运行一次示例。点击 Button，开始播放 TTS。按一次停止播放，再按一次重新开始。



4）相关 API

int tts_create(tts_h*)：创建 TTS 对象的 API。

int tts_set_state_changed_cb(tts_h tts, tts_state_changed_cb callback, void* user_data)：定义 TTS 状态变更事件回调函数名称的 API。参数依次为 TTS 对象、回调函数名称、用户数据。

int tts_prepare(tts_h tts)：转换 TTS 对象准备状态的 API。

int tts_destroy(tts_h tts)：删除 TTS 对象的 API。

int tts_add_text(tts_h tts, const char* text, const char* language, int voice_type, int speed, int* utt_id)：在 TTS 对象中添加字符串的 API。参数依次为 TTS 对象、字符串、声音种类、速度。声音种类如下所示。

- TTS_VOICE_TYPE_AUTO：声音种类自动
- TTS_VOICE_TYPE_MALE：男子声音
- TTS_VOICE_TYPE_FEMALE：女人声音
- TTS_VOICE_TYPE_FEMALE：儿童声音

int tts_get_state(tts_h tts, tts_state_e* state)：转换 TTS 当前状态的 API。

int tts_play(tts_h tts)：开始播放 TTS 的 API。

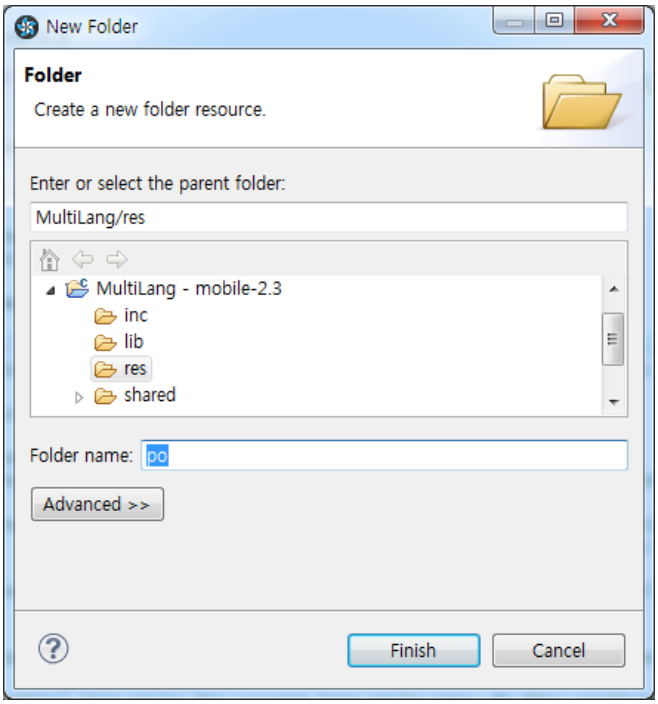
int tts_stop(tts_h tts)：停止播放 TTS 的 API。

66. 多国语言支持

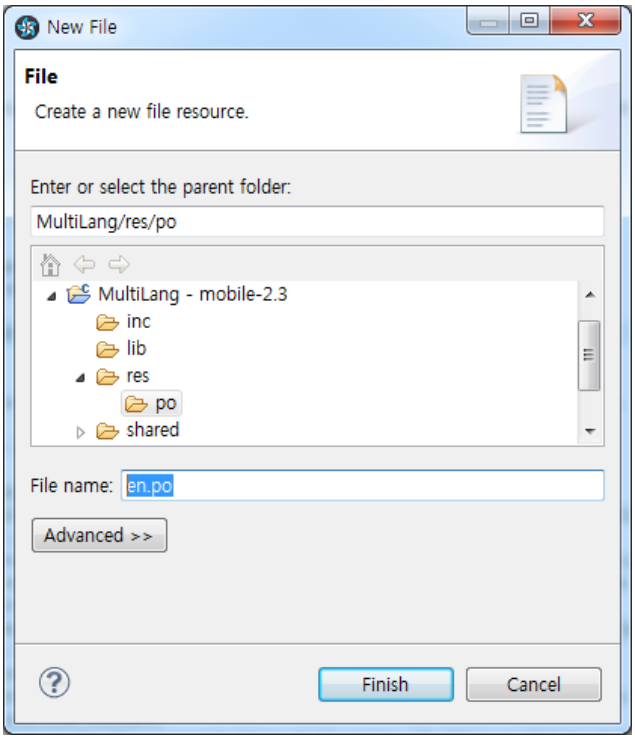
随着应用程序商店向开发人员开放，身处亚洲的开发人员也可以向欧洲消费者销售应用程序。但是为了支持各种语言，必须要制作不同版本的话，对开发者来说相当不便。将相同意思的各语言文本登录在源代码上，仅调用所需部分使用就可以万门解决这一问题。下面就来了解一下在 PO 文件中保存多国语言信息，从源代码中调用的方法。

1）在源代码中登录多国语言信息

创建新的源项目，将 Project name 命名为 MultiLang。创建源项目之后，登录多国语言信息。多国语言信息保存于 /res/po 文件夹内。右键点击 /res 文件夹，在快捷菜单中选择 [New > Folder]。出现弹窗后，在 Folder name 项目中输入 po，点击 Finish 键。

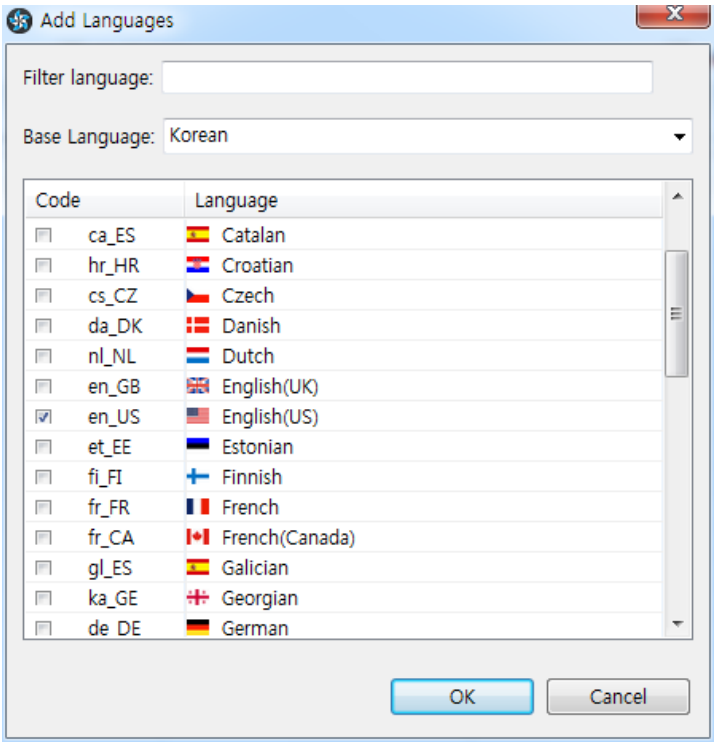


多国语言信息文件没有自动创建。下面将手动创建文件。右键点击 /res/po 文件夹，在快捷菜单中选择 [New > File]。出现弹窗后，在 Folder name 项目中输入 en.po，点击 Finish 键。

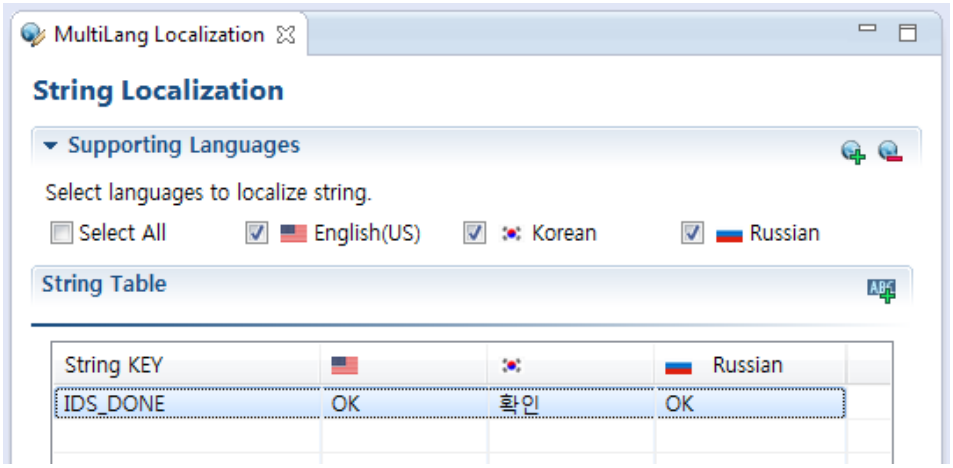


en.po 是保存英文相关信息的文件。因为大部分应用程序支持英语，可以说是必备的多国语言信息文件。随着支持语言数量的增加，po 文件的个数也随之增多。建好 po 后，在编辑器中显示内容。如果编辑器没有自动打开，双击 en.po 文件。

出现“Add Languages”弹窗后，从语言目录中检查几项（en_US (English)、ko_KR (Korean)、ru_RU (Russian)）。

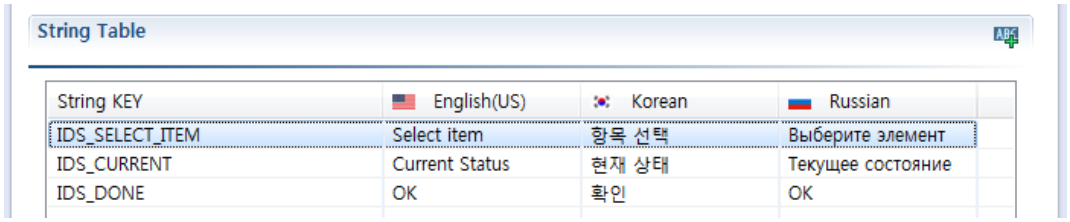


下面来添加一条信息。点击屏幕右侧的“Add String Key”键，在 MsgID 中输入 IDS_DONE。然后在 English 栏相应的字段中输入 OK。确认 Korean 字段，在 Russian 字段输入 OK。



下面来添加一条信息。点击“Add String Key”键，在新添项目中输入内容。(IDS_CURRENT、Current Status、当前状态、Текущее состояние)

再次点击“Add String Key”键，在新添项目中输入内容。(IDS_SELECT_ITEM、Select item、项目选择、Выберите элемент)



2）在源代码中定义多国语言信息

保存 po 文件后，打开 src 文件夹内的源文件 （~.c），在 appdata 结构上添加变量。

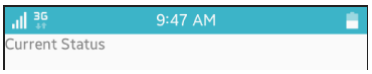
```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        Evas_Object* naviframe;
} appdata_s;
```

在 Label 小部件 IDS_CURRENT 中应用相应的多国语言。移动至 create_base_gui()，修改 Label 小部件创建代码。

```
    /* Label*/
    ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, i18n_get_text("IDS_CURRENT"));
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_object_content_set(ad->conform, ad->label);

        /* Show window after base gui is set up */
        evas_object_show(ad->win);
```

i18n_get_text(const char *message) 是在环境设置语言栏中返回相应多国语言信息的 API。

构建并运行示例。显示在 po 文件中登录的多国语言信息。



3）在 NaviFrame 中应用多国语言

在标题文本中应用多国语言时，将使用 i18n_get_text() 函数。如果想要在变更语言环境设置时自动应用，必须使用其他方法。在 create_base_gui() 函数中添加 NaviFrame、Box创建代码。

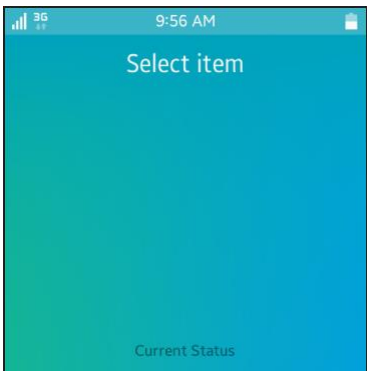
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    /* Naviframe */
    ad->naviframe = elm_naviframe_add(ad->conform);
    eext_object_event_callback_add(ad->naviframe, EEXT_CALLBACK_BACK, win_back_
cb, ad);
    elm_object_content_set(ad->conform, ad->naviframe);

    /* Box */
    Evas_Object *box = elm_box_add(ad->naviframe);
    elm_box_padding_set(box, 0, ELM_SCALE_SIZE(20));

    /* Push a view to naviframe */
    Elm_Object_Item *nf_it = elm_naviframe_item_push(ad->naviframe, "IDS_SELECT
_ITEM", NULL, NULL, box, NULL);
    /* Mark naviframe title as translatable text */
    elm_object_item_part_text_translatable_set(nf_it, NULL, EINA_TRUE);

    {
        /* Label*/
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, i18n_get_text("IDS_CURRENT"));
        //evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HI
```

```
NT_EXPAND);
        //elm_object_content_set(ad->conform, ad->label);
        evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, 0.0);
        evas_object_size_hint_align_set(ad->label, 0.5, 0.0);
        elm_box_pack_end(box, ad->label);
        evas_object_show(ad->label);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
```

创建 Naviframe 和 Box 容器，将题目文本定义为“IDS_SELECT_ITEM”。这样题目显示为“IDS_SELECT_ITEM”。所以有必要添加一些内容。

elm_object_item_part_text_translatable_set(it, part, translatable) 是在源代码中定义对象文本的 API。 如果第 3 个参数为 EINA_TRUE，则从源代码中提取文本。如果是 EINA_FALSE，则直接使用已输入的文本。

让我们再运行一次示例。显示标题，在题目中成功应用多国语言。



3）在小部件中应用多国语言

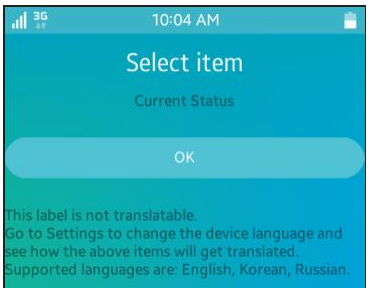
在 Label、Button、Entry 等小部件中应用多国语言另有办法。在 create_base_gui() 函数中添加 Button 创建代码。

```
{
    /* Label*/
    ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, i18n_get_text("IDS_CURRENT"));
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, 0.0);
    evas_object_size_hint_align_set(ad->label, 0.5, 0.0);
    elm_box_pack_end(box, ad->label);
    evas_object_show(ad->label);

    /* Button */
    Evas_Object* btn = elm_button_add(ad->conform);
    elm_object_translatable_text_set(btn, "IDS_DONE");
    evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, 0.0);
    evas_object_size_hint_align_set(btn, -1.0, 0.0);
    elm_box_pack_end(box, btn);
    evas_object_show(btn);

    /* Label*/
    Evas_Object *o = elm_label_add(ad->conform);
    elm_label_line_wrap_set(o, ELM_WRAP_WORD);
    elm_object_text_set(o, "This label is not translatable.<br/>"
        "Go to Settings to change the device language and see how the above
items will get translated.<br/>"
        "Supported languages are: English, Korean, Russian.");
    evas_object_size_hint_weight_set(o, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    evas_object_size_hint_align_set(o, -1.0, 0.0);
    elm_box_pack_end(box, o);
    evas_object_show(o);
}
```

elm_object_translatable_text_set(obj, text) 是在小部件中自动应用源代码登录文本的 API。用户在环境设置中变更语言选项时，小部件的文本也自动重新应用多国语言。

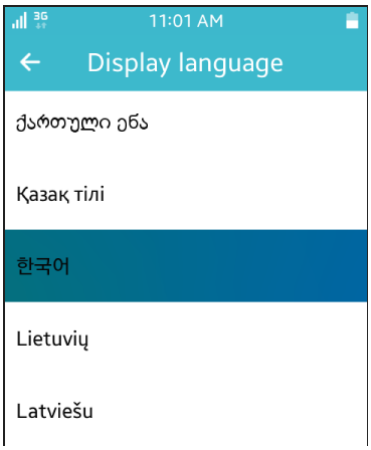
第 2 个 Label 小部件用于为用户显示文本字样，无特殊功能。

让我们再运行一次示例。在 Button 小部件上成功应用多国语言。

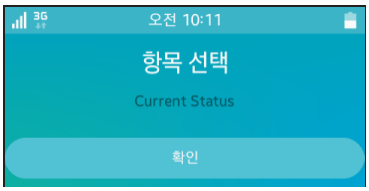


5）多国语言自动变更

模拟器的默认语言设置为 English (United States)。下面来变更为其他语言。在运行 MultiLang 应用程序状态下，点击 Back 键，移动至应用程序目录屏幕。点击 Settings 图标，在目录中选择“Language and input”。屏幕更换后，选择“Display language”。出现语言目录后，在韩国语、俄语中选择一种语言。



退出环境设置，重新运行 MultiLang 示例。长按 Home 键，出现目前为止已运行的应用程序目录，从中选择 MultiLang 即可。将示例转换为 Foreground 模式，NaviFrame 和 Button 自动转换为多国语言。在 Label 小部件上英语未发生变化。定义变更语言环境设置的事件，必须手动重新指定多国语言。



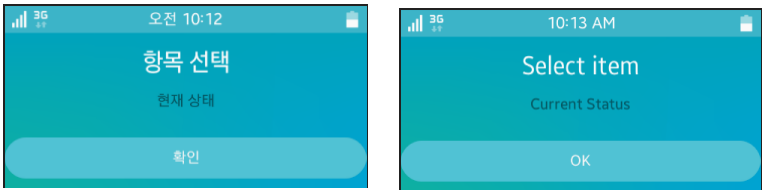
源文件下方有一个名为 ui_app_lang_changed() 的函数。在这个函数末尾添加新代码。

```
static void
ui_app_lang_changed(app_event_info_h event_info, void *user_data)
{
	/*APP_EVENT_LANGUAGE_CHANGED*/
	char *locale = NULL;
	system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_LANGUAGE, &l
ocale);
	elm_language_set(locale);
	free(locale);

	appdata_s* ad = user_data;
	elm_object_text_set(ad->label, i18n_get_text("IDS_CURRENT"));
}
```

ui_app_lang_changed() 是变更环境设置语言种类时运行的事件函数。如果想要变更函数名称，在 main() 函数中修改即可。

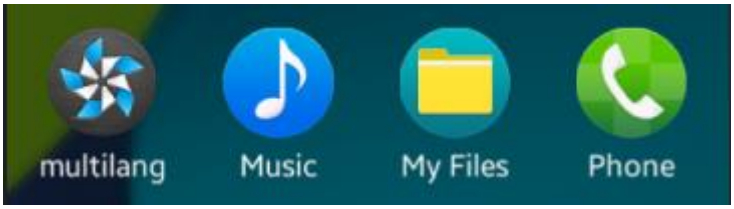
在 label 小控件上重新指定多国语言。这样用户在环境设置中变更语言种类时，小部件也将自动变更多国语言设置。

重新运行示例，在环境设置中将语言更改为 English (United States)。返回 MultiLang 示例，已自动应用多国语言。



6）变更 App 图标文本

点击 Back 键，移动至应用程序图标目录界面，呈现 MultiLang 示例的图标。Label 文本应该显示为 multilang。下面来了解一下应用多国语言的方法。



打开 tizen-manifest.xml 文件，点击下方选项卡按键中的 tizen-manifest.xml 按键，呈现源代码。其中有一条 <label>multilang</label> 代码，这就是应用程序图标的 Label 文本。添加新 XML 代码。

```
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.multilang" version="1.0.0">
        <profile name="mobile" />
        <ui-application appid="org.example.multilang" exec="multilang" type="capp" multiple="false" taskmanage="true" nodisplay="false">
                <icon>multilang.png</icon>
                <label>multilang</label>
                <label xml:lang="en-us">English</label>
                <label xml:lang="ko-kr">Korean</label>
                <label xml:lang="ru-ru">Russian</label>
```

```
        </ui-application>
</manifest>
```

在 label 标签中定义各语言文本后，应用多国语言。重新运行示例，在环境设置中将语言更改为韩国语或者俄语。返回应用程序图标目录屏幕，变更应用程序图标文本。

7）相关 API

char*　i18n_get_text(const char *message)：在环境设置语言栏中返回相应多国语言信息的 API。

void　elm_object_translatable_text_set(obj, text)：在小部件中自动应用源代码登录文本的 API。用户在环境设置中变更语言选项时，小部件的文本也自动重新应用多国语言。

void　elm_object_item_part_text_translatable_set(it, part, translatable)：在源代码中定义对象文本的 API。如果第 3 个参数为 EINA_TRUE，从源代码中提取文本。如果是 EINA_FALSE，则直接使用默认文本。

void　ui_app_lang_changed()：环境设置语言种类变更时运行的事件函数。如果想要变更函数名称，在 main() 函数中修改即可。

67. JSON 分析

JSON （Java Script Object Notation） 与 XML 一样同为传输数据时所使用的格式。从名字上即可得知它主要用于 Java 脚本，但比 XML 使用简便，目前广泛应用于网络通信。举例来说如果用 XML 语法和 JSON 语法编写相同内容，则 JSON 更为简便。因为它仅包含必需信息。下面通过本示例来了解一下分析 JSON 数据的方法。
XML 形式 - <name>Obama</name>50
JSON 形式 - [name:Obama, math:50]

1）定义 JSON 排列数据

创建新的源项目，将 Project name 命名为JsonParse。打开 src 文件夹内的源文件 （~.c），添加库头文件和变量。

```
#include "jsonparse.h"
#include <json-glib/json-glib.h>
```

json-glib/json-glib.h 是分析 JSON 的库头文件。

在 JSON 形式中在“[]”内保存排列。下面来了解一下从排列中逐个提取数据的方法。在 create_base_gui() 上创建新函数。

```
static void
parse_json(appdata_s *ad)
{
        JsonParser *parser = json_parser_new ();
        char buf[256];
        buf[0] = '\0';

        const char* data1 = "[11, 22, 33, 44, 55]";

        if( json_parser_load_from_data( parser, data1, strlen(data1), NULL))
        {
                JsonNode *root = json_parser_get_root (parser);
                JsonArray *temp_array = json_node_get_array ( root );
```

```
                for(int i=0; i < json_array_get_length(temp_array); i++ )
                        sprintf(buf, "%s - %d", buf, json_array_get_int_element
(temp_array, i));
        }

        elm_object_text_set(ad->label, buf);
}
```

JsonParser 是分析 JSON 数据的结构。

json_parser_new() 是创建 JsonParser 对象的 API。

在名为 data1 的字符串变量中保存 JSON 语法。用哟 5 个数字项的排列。

json_parser_load_from_data(JsonParser*, gchar*, gssize, GError**) 是在 JsonParser 对象中输入 JSON 语法的 API。参数依次为 JsonParser 对象、JSON 数据、数据长度、错误代码。

json_parser_get_root(JsonParser*) 是返回 JsonParser 起始位置的 API。返回形式为 JsonNode。

json_node_get_array(JsonNode*) 是以排列形式显示 Json 语法的 API。显示形式为 JsonArray。

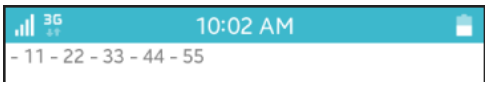
json_array_get_length(JsonArray*) 是显示 Json 排列项目个数的 API。显示形式为 guint。

json_array_get_int_element(JsonArray*, guint) 是以整数型显示 Json 特定项目排列的 API。显示形式为 gint64。

运行示例后，将自动运行上述函数。在 create_base_gui() 函数末尾添加一行新代码。

```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        parse_json(ad);
}
```

构建并运行示例。保存于 Json 排列的 5 个数据显示在 Label 小部件上。



2）用 Key 定义数据

Json 语法通常用于以下形式。name 为 Key，Obama 是数据。

{name:Obama, math:50}

下面来了解一下用 Key 值定义数据的方法。在 parse_json() 函数中添加新代码。

```
        ~
        const char* data2 = "{'time': '03:53:25 AM', 'millisec_epoch': 13621964
05309, 'date': '03-02-2013'}";
        if( json_parser_load_from_data( parser, data2, strlen(data2), NULL))
        {
                JsonNode *root = json_parser_get_root (parser);
                JsonObject *obj = json_node_get_object(root);
                char* time_data = json_object_get_string_member (obj, "time");
                long long epoch_data = json_object_get_int_member (obj, "millis
ec_epoch");
                sprintf(buf, "%s <br/><br/> time - %s <br/> epoch - %lld", buf,
 time_data, epoch_data);
        }

        elm_object_text_set(ad->label, buf);
}
```

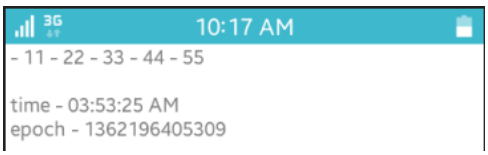
在名为 data2 的字符串变量中保存 Json 语法，利用 json_parser_load_from_data() 函数在 JsonParser 对象中输入 Json 语法。

json_node_get_object(JsonNode*) 是以 JsonObject 形式显示 Json 语法的 API。

json_object_get_string_member(JsonObject*, gchar*) 是在 JsonObject 中以字符串形式显示数据的 API。在第 2 个参数上传入 Key，将以 gchar* 形式显示。

json_object_get_int_member(JsonObject*, gchar*) 是在 JsonObject 中以数字形式显示数据的 API。在第 2 个参数上传入 Key，将以 gint64 形式显示。

让我们再运行一次示例。time 项目数据和 epoch 项目数据已显示在屏幕上。



3）多层次分析

以下 Json 语法中，在 coord 内包含有 Json 项目和 Json 排列。下面就来了解一下分析 Json 语法复杂结构的方法。

{'coord':{'lon':127.03, 'lat':37.5}, 'weather':[{'id':800, 'main':'Clear'}, {'id':887, 'main':'Cloudy'}]}

在 parse_json() 函数末尾添加新代码。

在 Json 项目内包含的 Json 项目中定义数据代码。

```
        ~
        const char* data3 = "{'coord':{'lon':127.03, 'lat':37.5},
                'weather':[{'id':800, 'main':'Clear'}, {'id':887, 'main':'Cloud
y'}]}";
        if( json_parser_load_from_data( parser, data3, strlen(data3), NULL))
        {
                JsonNode *root = json_parser_get_root (parser);
                JsonObject *obj = json_node_get_object(root);
                JsonNode *temp_node = json_object_get_member (obj, "coord");
                JsonObject *temp_object = json_node_get_object(temp_node);
```

```
            sprintf(buf, "%s <br/><br/> coord:lon - %0.2f", buf,
                    json_object_get_double_member (temp_object, "lon"));
        }

        elm_object_text_set(ad->label, buf);
```

在名为 data3 的字符串变量中保存 Json 语法，利用 json_parser_load_from_data() 函数在 JsonParser 对象中输入 Json 语法。

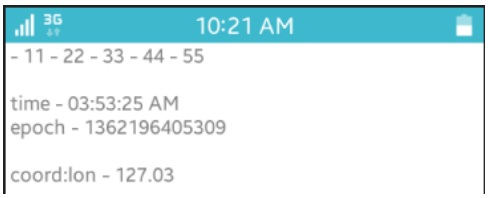
用 json_parser_get_root() 函数定义起始位置的 Json 节点，用 json_node_get_object() 函数以 JsonObject 形式显示 Json 语法。

json_object_get_member(JsonObject *, gchar*) 是返回特定 JsonObject 的 API。参数依次为 JsonParser、成员名称。返回形式为 JsonNode。

利用 json_node_get_object() 定义 JsonNode 中的 JsonObject。

json_object_get_double_member(JsonObject *, gchar*) 是在 JsonObject 中，以实数形式显示数据的 API。在第 2 个参数上传入 Key，将以 gdouble 形式显示。

让我们再运行一次示例。在 coord 组内显示 lon Key 相应数据。



4）分析阵列内组数据

下面来展示一下如何定义相同 Json 语法中，名为 weather 的阵列第 1 项 id 相应的值。在 parse_json() 函数中添加新代码。

```
        const char* data3 = "{'coord':{'lon':127.03, 'lat':37.5}, 'weather':[{'id':800, 'main':'Clear'},
                {'id':887, 'main':'Cloudy'}]}";
```

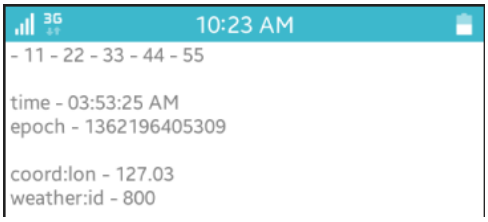
```
        if( json_parser_load_from_data( parser, data3, strlen(data3), NULL))
        {
                JsonNode *root = json_parser_get_root (parser);
                JsonObject *obj = json_node_get_object(root);
                JsonNode *temp_node = json_object_get_member (obj, "coord");
                JsonObject *temp_object = json_node_get_object(temp_node);
                sprintf(buf, "%s <br/><br/> coord:lon - %0.2f", buf,
                        json_object_get_double_member (temp_object, "lon"));

                temp_node = json_object_get_member ( obj,"weather" );
                JsonArray *temp_array = json_node_get_array ( temp_node );
                temp_node = json_array_get_element(temp_array, 0 );
                temp_object = json_node_get_object( temp_node );
                sprintf(buf, "%s <br/> weather:id - %d", buf, json_object_get_i
nt_member
                        (temp_object, "id"));
        }

        elm_object_text_set(ad->label, buf);
```

利用 json_object_get_member() 函数定义名为 weather 的阵列节点，利用 json_node_get_array() 函数以 JsonArray 形式显示节点。

利用 json_array_get_element() 函数定义第 1 项节点，利用 json_node_get_object() 函数以 JsonObject 形式显示节点。

最后利用 json_object_get_int_member() 函数以整数形式显示名为 id 的 Key中相应的数据。

让我们再运行一次示例。显示第 1 阵列项目 id 相应值。



5）相关 API

JsonParser *json_parser_new()：创建 JsonParser 对象的 API。

gboolean json_parser_load_from_data(JsonParser*, gchar*, gssize, GError**)：在 JsonParser 对象中输入 JSON 语法的 API。参数依次为 JsonParser 对象、JSON 数据、数据长度、错误代码。

JsonNode* json_parser_get_root(JsonParser*)：返回 JsonParser 起始位置的 API。返回形式为 JsonNode。

JsonArray* json_node_get_array(JsonNode*)：以排列形式显示 Json 语法的 API。显示形式为 JsonArray。

guint json_array_get_length(JsonArray*)：显示 Json 排列项目个数的 API。显示形式为 guint。

gint64 json_array_get_int_element(JsonArray*, guint)：以整数型显示 Json 特定项目排列的 API。显示形式为 gint64。

JsonObject* json_node_get_object(JsonNode*)：以 JsonObject 形式显示 Json 语法的 API。

gchar* json_object_get_string_member(JsonObject*, gchar*)：在 JsonObject 中以字符串形式显示数据的 API。在第 2 个参数上传入 Key，将以 gchar* 形式显示。

gint64 json_object_get_int_member(JsonObject*, gchar*)：在 JsonObject 中以数字形式显示数据的 API。在第 2 个参数上传入 Key，将以 gint64 形式显示。

JsonNode* json_object_get_member(JsonObject *, gchar*)：返回特定 JsonObject 的 API。参数依次为 JsonParser、成员名称。返回形式为 JsonNode。

68. XML 解析

当移动应用程序通过 HTTP 与服务器进行通信时，会传送 XML 格式的数据。XLM 是一种用于系统存储数据的文档协议，它不仅广泛用于通信领域，还用于存储应用程序开发时所需的数据（如屏幕布局和资源数据）。在本例中，我们将学习如何解析 XML 数据。

1) 请求 XML 数据

创建新的源项目，将项目名称指定为 XmlParse。打开 src 文件夹中的源文件 (~.c)，并添加库头文件与变量。

```
#include "xmlparse.h"
#include <libxml/HTMLparser.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label;
        bool value_begin;
        int value_type;
        char buffer[1024];
} appdata_s;
```

libxml/HTMLparser.h 是用于解析 XML 的库头文件。

value_begin 是保存数据开始与否的变量。

value_type 是保存数据类型的变量。

buffer[] 是保存已解析数据的字符串数组变量。

以下是一个简单 XML 语法示例。name 表示节点名称，Elsa 表示数据。我们现在来解析该语法。

<name>Elsa</name>95

在 create_base_gui() 函数之上添加一个新函数。该函数将向一个 Box 容器添加一个小部件。

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child, double h_weight, double v_weig
ht,
        double h_align, double v_align)
{
        /* create a frame we shall use as padding around the child widget */
```

然后，向 create_base_gui() 函数添加 Box-creating 和 Button-creatinh 代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    Evas_Object *box, *btn;

    /* Box */
    box = elm_box_add(ad->conform);
    elm_box_horizontal_set(box, EINA_FALSE);
    evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    evas_object_size_hint_align_set(box, EVAS_HINT_FILL, EVAS_HINT_FILL);
    elm_object_content_set(ad->conform, box);
    evas_object_show(box);

    {
        /* Label */
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "<align=center>Hello EFL</>");
        my_box_pack(box, ad->label, 1.0, 1.0, -1.0, -1.0);

        /* Button-1 */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Parse1");
        evas_object_smart_callback_add(btn, "clicked", btn_parse1_cb, ad);
        my_box_pack(box, btn, 1.0, 0.0, -1.0, 1.0);
```

```
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

在 create_base_gui() 函数之上添加两个新函数。

```
void
walkTree1(xmlNode * a_node, appdata_s *ad)
{
        xmlNode *cur_node = NULL;
        xmlChar *key = NULL;
        xmlChar *value = NULL;

        for (cur_node = a_node; cur_node; cur_node = cur_node->next)
        {
                if(!strcmp((const char*)cur_node->name, "name"))
                        ad->value_type = 1;
                if(!strcmp((const char*)cur_node->name, "math"))
                        ad->value_type = 2;

                if(!strcmp((const char*)cur_node->name, "text")) {
                        if( ad->value_type == 1 )
                        {
                                value = cur_node->content;
                                strcat(ad->buffer, "Name : ");
                                strcat(ad->buffer, (char*)value);
                                ad->value_type = 0;
                        }
                        else if( ad->value_type == 2 )
                        {
                                value = cur_node->content;
                                strcat(ad->buffer, " / Math : ");
                                strcat(ad->buffer, (char*)value);
                                strcat(ad->buffer, "<br/>");
                                ad->value_type = 0;
                        }
                }

                walkTree1(cur_node->children, ad);
        }
}
```

```
static void
btn_parse1_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        const char* buf = "<name>Elsa</name><math>95</math>";

        htmlParserCtxtPtr parser = htmlCreatePushParserCtxt(NULL, NULL, NULL, 0,
 NULL, 0);
        htmlCtxtUseOptions(parser, HTML_PARSE_NOBLANKS | HTML_PARSE_NOERROR | HT
ML_PARSE_NOWARNING | HTML_PARSE_NONET);
        htmlParseChunk(parser, buf, strlen(buf), 0);

        ad->buffer[0] = '\0';
        ad->value_type = 0;
        walkTree1(xmlDocGetRootElement(parser->myDoc), ad);
        elm_object_text_set(ad->label, ad->buffer);
}
```

walkTree1() 函数将从 xmlNode 对象中提取与“name”和“math”相对应的数据，并将数据添加到全局变量中。

从 for 循环中请求 XML 语法所包含的单个节点，当指定的单个节点的名称为“name”或“math”时，则请求该数据。为前进到下一个节点，该函数将调用自身。

xmlNode 是 XML 语法中指向特定点的结构变量。其属性如下所示：

- next：返回下一个节点的指针。
- name：节点名称。如果“text”已保存在此属性中，则意味着节点就是数据。在这种情况下，可从 content 属性中请求数据。
- content：节点数据。
- properties：节点属性数据。

btn_parse1_cb() 函数可创建 XML 解析器对象、输入 XML 语法，然后调用执行解析的函数。

htmlCreatePushParserCtxt(htmlSAXHandlerPtr sax, void *user_data, char *chunk, int size, char *filename, xmlCharEncoding enc) 是一种可创建 XML 解析器对象的 API。

htmlParserCtxtPtr 是一种 XML 解析器结构。在所有属性中，myDoc 以 xml

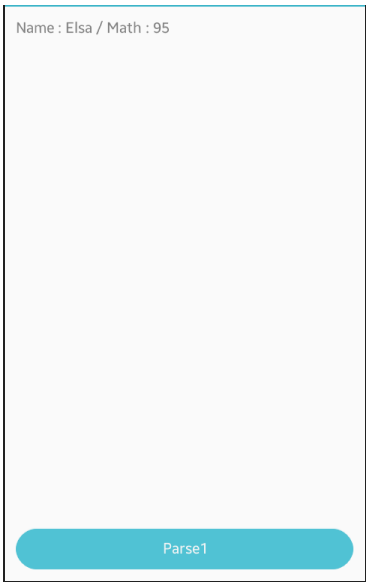
DocPtr 格式保存 XML 语法。

htmlCtxtUseOptions(htmlParserCtxtPtr ctxt, int options) 是一种可为 XML 解析器指定选项的 API。可重复指定此选项。

htmlParseChunk(htmlParserCtxtPtr ctxt, char *chunk, int size, int terminate) 是一种可在 XML 解析器中输入 XML 语法的 API。

xmlDocGetRootElement(xmlDocPtr doc) 是一种可返回 XML 语法开始节点的 API。

我们现在来构建并运行一个示例。点击 Button，name 节点的数据与 math 节点的数据将会显示在屏幕上。



2) 请求节点属性数据

如下所示，XML 可让您指定节点属性。在本小节中，我们将了解如何请求节点属性数据。

<student name="Aurora" math="27"></student>

在 create_base_gui() 函数末尾添加创建第二个 Button 的代码。

```
        /* Button-1 */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Parse1");
        evas_object_smart_callback_add(btn, "clicked", btn_parse1_cb, ad);
        my_box_pack(box, btn, 1.0, 0.0, -1.0, 1.0);

        /* Button-2 */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Parse2");
        evas_object_smart_callback_add(btn, "clicked", btn_parse2_cb, ad);
        my_box_pack(box, btn, 1.0, 0.0, -1.0, 1.0);
    }
```

在 create_base_gui() 函数之上添加两个新函数。

```
void
walkTree2(xmlDoc *doc, xmlNode * a_node, appdata_s *ad)
{
        xmlNode *cur_node = NULL;
        xmlAttr *cur_attr = NULL;
        xmlChar *key = NULL;

        for (cur_node = a_node; cur_node; cur_node = cur_node->next)
        {
                for (cur_attr = cur_node->properties; cur_attr; cur_attr = cur_a
ttr->next) {
                        key = xmlGetProp(cur_node, cur_attr->name);

                        if(!strcmp((const char*)cur_attr->name, "name"))
                        {
                                strcat(ad->buffer, "Name : ");
                                strcat(ad->buffer, key);
                        }
                        if(!strcmp((const char*)cur_attr->name, "math"))
                        {
                                strcat(ad->buffer, " / Math : ");
                                strcat(ad->buffer, key);
                                strcat(ad->buffer, "<br/>");
                        }

                        if(key!=NULL) { xmlFree(key); key=NULL; }
                }
```

```
                walkTree2(doc, cur_node->children, ad);
        }
}

static void
btn_parse2_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        const char* buf = "<?xml version=\"1.0\" encoding=\"utf-8\"?> <grade> <n
ame>M.I.T</name> <student name=\"Aurora\" math=\"27\"></student> <student name=\
"Piana\" math=\"88\"></student> <student name=\"Tangled\" math=\"77\"></student>
 </grade>";

        htmlParserCtxtPtr parser = htmlCreatePushParserCtxt(NULL, NULL, NULL, 0,
 NULL, 0);
        htmlCtxtUseOptions(parser, HTML_PARSE_NOBLANKS | HTML_PARSE_NOERROR | HT
ML_PARSE_NOWARNING | HTML_PARSE_NONET);

        htmlParseChunk(parser, buf, strlen(buf), 0);
        ad->buffer[0] = '\0';
        walkTree2(parser->myDoc, xmlDocGetRootElement(parser->myDoc), ad);
        elm_object_text_set(ad->label, ad->buffer);
}
```

walkTree2() 函数可请求 XML 节点的属性数据，并将数据显示在屏幕上。

从第一个 for 循环请求 XML 语法所包含的单个节点，从第二个 for 循环请求单个属性。然后，如果指定单个节点的名称为“name”或“math”，则请求数据。为前进到下一个节点，该函数将调用自身。

在 xmlNode 的属性中，properties 属性以 xmlAttr 格式保存节点属性。

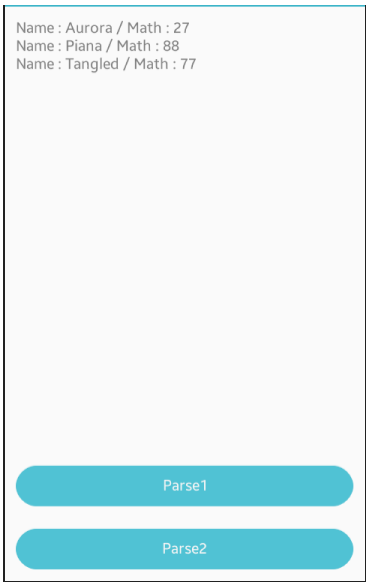
xmlGetProp(xmlNodePtr node, const xmlChar *name) 是一种可返回节点属性数据的 API。向第一个、第二个参数分别传递 XML 节点、属性名称。

xmlFree(xmlChar*) 是一种可删除 XML 数据的 API。

btn_parse2_cb() 函数可创建 XML 解析器对象、输入 XML 语法，然后调用运行属性数据解析的函数。

再次运行该示例，然后点击第二个 Button。节点属性数据将会显示在屏幕

上。



3) 多层次节点解析

以下 XML 语法中共有三个 <student> 节点，每个 <student> 节点内含有一个 <name> 节点和一个 <math> 节点。现在我们将要学习如何使用树状结构来访问节点。

```
<student><name>Obama</name><math>50</math></student>
<student><name>Psy</name><math>70</math></student>
<student><name>Yuna</name><math>65</math></student>
```

在 create_base_gui() 函数末尾添加创建第三个 Button 的代码。

```
        /* Button-2 */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Parse2");
        evas_object_smart_callback_add(btn, "clicked", btn_parse2_cb, ad);
        my_box_pack(box, btn, 1.0, 0.0, -1.0, 1.0);

        /* Button-3 */
        btn = elm_button_add(ad->conform);
```

```
        elm_object_text_set(btn, "Parse3");
        evas_object_smart_callback_add(btn, "clicked", btn_parse3_cb, ad);
        my_box_pack(box, btn, 1.0, 0.0, -1.0, 1.0);
    }
```

在 create_base_gui() 函数之上添加两个新函数。

```
void
walkTree3(xmlDoc *doc, xmlNode * a_node, appdata_s *ad)
{
        xmlNode *cur_node = NULL;
        xmlAttr *cur_attr = NULL;
        xmlChar *key = NULL;
        xmlChar *value = NULL;

        for (cur_node = a_node; cur_node; cur_node = cur_node->next)
        {
                if(!strcmp((const char*)cur_node->name, "student") )
                        ad->value_begin = true;

                if(!strcmp((const char*)cur_node->name, "name") && ad->value_beg
in)
                        ad->value_type = 1;
                if(!strcmp((const char*)cur_node->name, "math") && ad->value_beg
in)
                        ad->value_type = 2;

                if(!strcmp((const char*)cur_node->name, "text")) {
                        if( ad->value_type == 1 )
                        {
                                value = (char*)cur_node->content;
                                strcat(ad->buffer, "Name : ");
                                strcat(ad->buffer, (char*)value);
                                ad->value_type = 0;
                        }
                        else if( ad->value_type == 2 )
                        {
                                value = (char*)cur_node->content;
                                strcat(ad->buffer, " / Math : ");
                                strcat(ad->buffer, (char*)value);
                                strcat(ad->buffer, "<br/>");
                                ad->value_type = 0;
                        }
```

```
                }

                walkTree3(doc, cur_node->children, ad);
        }
}

static void
btn_parse3_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        const char* buf = "<?xml version=\"1.0\" encoding=\"utf-8\"?> <grade> <n
ame>M.I.T</name> <student><name>Obama</name><math>50</math></student> <student><
name>Psy</name><math>70</math></student> <student><name>Yuna</name><math>65</mat
h></student> </grade>";

        htmlParserCtxtPtr parser = htmlCreatePushParserCtxt(NULL, NULL, NULL, 0,
 NULL, 0);
        htmlCtxtUseOptions(parser, HTML_PARSE_NOBLANKS | HTML_PARSE_NOERROR | HT
ML_PARSE_NOWARNING | HTML_PARSE_NONET);
        htmlParseChunk(parser, buf, strlen(buf), 0);

        ad->value_begin = false;
        ad->buffer[0] = '\0';
        ad->value_type = 0;
        walkTree3(parser->myDoc, xmlDocGetRootElement(parser->myDoc), ad);

        elm_object_text_set(ad->label, ad->buffer);
}
```

walkTree3() 函数可在 XML 树状结构中请求第二个节点的数据，并将数据显示在屏幕上。

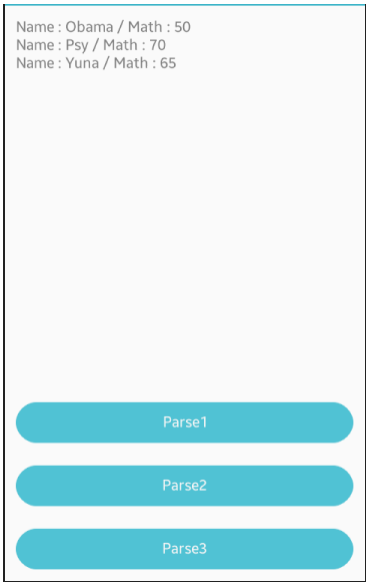
此函数将从 XML 语法中依次提取节点。如果节点名称为“student”，此函数会将该节点判定为第一个节点，并将全局变量“value_begin”设为 true。

如果节点名称为“name”或“math”，且全局变量“value_begin”为 true 时，此函数会将该节点判定为第二个节点，并会调用自身。

如果节点名称为“text”，则会将该节点判定数据，然后会读取其 content 属性值，并将该值添加到全局变量。为前进到下一个节点，该函数将调用自身。

btn_parse3_cb() 函数可创建 XML 解析器对象、输入 XML 语法，然后调用对第二个节点数据运行解析的函数。

再次运行该示例，然后点击第三个 Button。第二个节点数据将会显示在屏幕上。



4) 相关 API

htmlParserCtxtPtr htmlCreatePushParserCtxt(htmlSAXHandlerPtr sax, void *user_data, char *chunk, int size, char *filename, xmlCharEncoding enc)：一种创建 XML 解析器对象的 API。

htmlParserCtxtPtr：一种 XML 解析器结构。在所有属性中，myDoc 以 xmlDocPtr 格式保存 XML 语法。

int htmlCtxtUseOptions(htmlParserCtxtPtr ctxt, int options)：是一种可为 XML 解析器指定选项的 API。可重复指定此选项。

int htmlParseChunk(htmlParserCtxtPtr ctxt, char *chunk, int size, int terminate)：一种可在 XML 解析器中输入 XML 语法的 API。

xmlNodePtr xmlDocGetRootElement(xmlDocPtr doc)：一种可返回 XML 语法开始节点的 API。

xmlChar* xmlGetProp(xmlNodePtr node, const xmlChar *name)：一种可返回节点属性数据的 API。向第一个、第二个参数分别传递 XML 节点、属性名称。

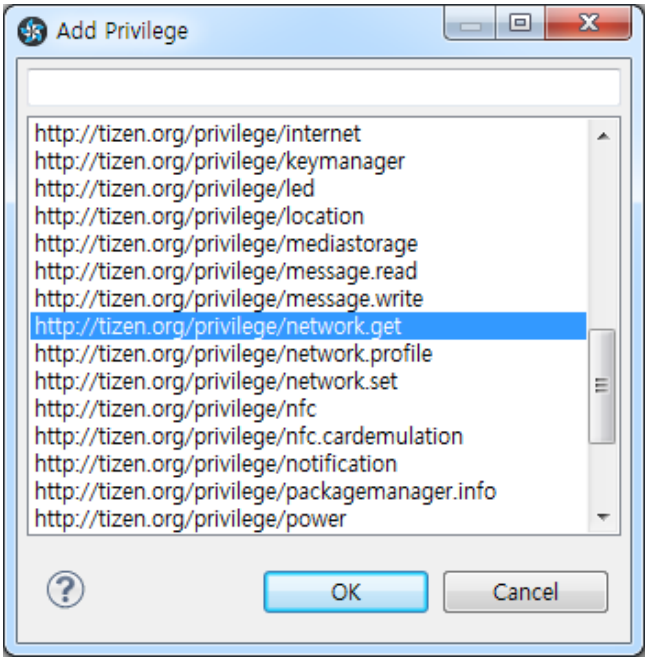
void xmlFree(xmlChar*)：一种可删除 XML 数据的 API。

69. 检查网络状态

在与服务器通信之前，建议您先检查一下网络状态。在本例中，我们将学习如何检查网络、移动通信、WiFi 通信的可用状态。

1）注册权限

创建新的源项目，将项目名称指定为 NetConnection。您需要具备用户权限才能检查通信状态。创建源项目之后，打开 tizen-manifest.xml 文件，在以下选项卡按钮中，点击 Privileges。然后，点击右上方的 Add 按钮。弹出窗口出现后，在列表中选择 http://tizen.org/privilege/network.get，然后点击 OK 按钮关闭窗口。



保存之后，在以下选项卡按钮中，点击右侧的 tizen-manifest.xml 按钮。随后，xml 文件的源代码将会出现。

```
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.netconnection" version="1.0.0">
        <profile name="mobile"/>
        <ui-application appid="org.example.netconnection" exec="netconnection" multiple="false" nodisplay="false" taskmanage="true" type="capp">
```

```
                <label>netconnection</label>
                <icon>netconnection.png</icon>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/network.get</privilege>
        </privileges>
</manifest>
```

2) 检查连接状态

在本小节中，我们将检查是否有网络连接。打开 src 文件夹中的源文件 (~.c)，并添加库头文件与变量。

```
#include "netconnection.h"
#include <net_connection.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label1;
        Evas_Object *label2;
        Evas_Object *label3;
        connection_h connection;
} appdata_s;
```

net_connection.h 是用于检查连接状态的库头文件。

label1 上将会显示连接状态，label2 上将显示移动连接状态，label3 上将显示 WiFi 连接状态。

connection_h 是一种通信信息结构。

在 create_base_gui() 函数之上添加一个新函数。此函数将会检查连接状态，并将结果显示在屏幕上。

```
static int
net_state(appdata_s *ad)
```

```
{
        int error_code;

        error_code = connection_create(&ad->connection);
        if (error_code != CONNECTION_ERROR_NONE) {
                dlog_print(DLOG_ERROR, "tag", "connection error");
             return error_code;
        }

        connection_type_e net_state;
        error_code = connection_get_type(ad->connection, &net_state);
        switch( net_state )
        {
        case CONNECTION_TYPE_DISCONNECTED       : /**< Disconnected */
                elm_object_text_set(ad->label1, "Net state Disconnected");
                break;
        case CONNECTION_TYPE_WIFI : /**< Wi-Fi type */
                elm_object_text_set(ad->label1, "Net state Wifi");
                break;
        case CONNECTION_TYPE_CELLULAR : /**< Cellular type */
                elm_object_text_set(ad->label1, "Net state Cellular");
                break;
        case CONNECTION_TYPE_ETHERNET : /**< Ethernet type */
                elm_object_text_set(ad->label1, "Net state Ethernet");
                break;
        case CONNECTION_TYPE_BT : /**< Bluetooth type */
                elm_object_text_set(ad->label1, "Net state BT");
                break;
        }
        return error_code;
}
```

connection_create(connection_h* connection) 是一种可创建 connection_h 对象的 API。

connection_get_type(connection_h connection, connection_type_e* type) 是一种可返回当前通信状态的 API。返回格式为 connection_type_e。connection_type_e 的类型如下：

- CONNECTION_TYPE_DISCONNECTED：通信中断
- CONNECTION_TYPE_WIFI：Wi-Fi 类型
- CONNECTION_TYPE_CELLULAR：移动通信类型
- CONNECTION_TYPE_ETHERNET：以太网类型
- CONNECTION_TYPE_BT：蓝牙类型

修改 create_base_gui() 函数的代码。此代码会创建 Frame、Box、Label 小部件并调用上述函数。

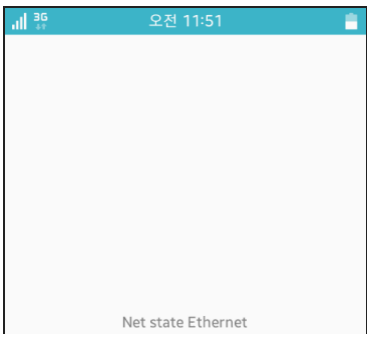
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    /* Frame for outer padding */
    Evas_Object *frame = elm_frame_add(ad->win);
    elm_object_style_set(frame, "pad_huge");
    elm_object_content_set(ad->conform, frame);
    evas_object_show(frame);

    /* Vertical box */
    Evas_Object *vbox = elm_box_add(ad->win);
    elm_box_padding_set(vbox, ELM_SCALE_SIZE(10), ELM_SCALE_SIZE(10));
    elm_object_content_set(frame, vbox);
    evas_object_show(vbox);

    {
        /* Label-1 */
        ad->label1 = elm_label_add(ad->conform);
        elm_object_text_set(ad->label1, "Net state");
        evas_object_size_hint_weight_set(ad->label1, EVAS_HINT_EXPAND, 0);
        elm_box_pack_end(vbox, ad->label1);
        evas_object_show(ad->label1);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    int error_code = net_state(ad);
}
```

此代码将在应用程序关闭后删除 connection_h 对象。向 app_terminate() 函数添加新代码。

```
static void
app_terminate(void *data)
{
        appdata_s *ad = data;
        connection_destroy(ad->connection);
}
```

connection_destroy(connection_h connection) 是一种可删除 connection_h 对象的 API。

构建并运行该示例。通信类型将显示在 Label 小部件上。



2) 请求移动通信状态

在本小节中，我们将请求移动通信（收费通信）的连接状态。在 create_base_gui() 函数之上添加一个新函数。

```
static void
cellular_state(appdata_s *ad)
{
        int error_code;
        connection_cellular_state_e state;
        error_code = connection_get_cellular_state(ad->connection, &state);

        switch( state )
        {
      case CONNECTION_CELLULAR_STATE_OUT_OF_SERVICE:
            elm_object_text_set(ad->label2, "Cell state Out of service");
```

```
            break;
      case CONNECTION_CELLULAR_STATE_FLIGHT_MODE:
            elm_object_text_set(ad->label2, "Cell state Flight mode");
            break;
      case CONNECTION_CELLULAR_STATE_ROAMING_OFF:
            elm_object_text_set(ad->label2, "Cell state Roaming off");
            break;
      case CONNECTION_CELLULAR_STATE_CALL_ONLY_AVAILABLE:
            elm_object_text_set(ad->label2, "Cell state Call only");
            break;
      case CONNECTION_CELLULAR_STATE_AVAILABLE:
            elm_object_text_set(ad->label2, "Cell state Available");
            break;
      case CONNECTION_CELLULAR_STATE_CONNECTED:
            elm_object_text_set(ad->label2, "Cell state Connected");
            break;
      default:
            elm_object_text_set(ad->label2, "Cell state Error");
            break;
        }
}
```

connection_get_cellular_state(connection_h connection, connection_cellular_state_e* state) 是一种可返回移动连接状态的 API。返回格式为 cconnection_cellular_state_e。connection_cellular_state_e 的类型如下：

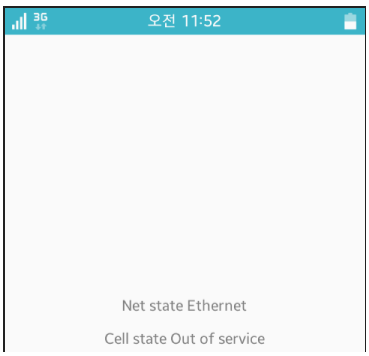
- CONNECTION_CELLULAR_STATE_OUT_OF_SERVICE：中断
- CONNECTION_CELLULAR_STATE_FLIGHT_MODE：飞行模式
- CONNECTION_CELLULAR_STATE_ROAMING_OFF：漫游关闭
- CONNECTION_CELLULAR_STATE_CALL_ONLY_AVAILABLE：仅限通话
- CONNECTION_CELLULAR_STATE_AVAILABLE：有可用连接，但尚未连接
- CONNECTION_CELLULAR_STATE_CONNECTED：已连接

接下来，我们将添加一个新的 Label 小部件，并将上述函数的结果显示在屏幕上。向 create_base_gui() 函数添加新代码。

```
      /* Label-1 */
      ad->label1 = elm_label_add(ad->conform);
      elm_object_text_set(ad->label1, "Net state");
      evas_object_size_hint_weight_set(ad->label1, EVAS_HINT_EXPAND, 0);
      elm_box_pack_end(vbox, ad->label1);
```

```
        evas_object_show(ad->label1);

        /* Label-2 */
        ad->label2 = elm_label_add(ad->conform);
        elm_object_text_set(ad->label2, "Cell state");
        evas_object_size_hint_weight_set(ad->label2, EVAS_HINT_EXPAND, 0);
        elm_box_pack_end(vbox, ad->label2);
        evas_object_show(ad->label2);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    int error_code = net_state(ad);
    if (error_code == CONNECTION_ERROR_NONE) {
        cellular_state(ad);
    }
}
```

再次运行该示例。移动连接状态将显示在第二个 Label 小部件中。



3) 请求 WiFi 状态

在本小节中，我们将请求 WiFi 连接状态。在 create_base_gui() 函数之上添加一个新函数。

```
static void
wifi_state(appdata_s *ad)
{
```

```
        connection_wifi_state_e wifi_state;
        connection_get_wifi_state(ad->connection, &wifi_state);
        switch (wifi_state)
        {
             case CONNECTION_WIFI_STATE_DEACTIVATED:
                   elm_object_text_set(ad->label3, "Wifi state Deactivated");
                   break;
             case CONNECTION_WIFI_STATE_DISCONNECTED:
                   elm_object_text_set(ad->label3, "Wifi state Disconnected");
                   break;
             case CONNECTION_WIFI_STATE_CONNECTED:
                   elm_object_text_set(ad->label3, "Wifi state Connected");
                   break;
             default:
                   dlog_print(DLOG_INFO, "tag", "Wifi error");
                   elm_object_text_set(ad->label3, "Wifi state Error");
                   break;
        }
}
```

connection_get_wifi_state(connection_h connection, connection_wifi_state_e* state) 是一种可返回 WiFi 连接状态的 API。返回格式为 connection_wifi_state_e。connection_wifi_state_e 的类型如下：

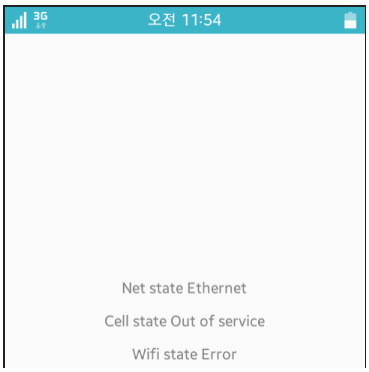
- CONNECTION_WIFI_STATE_DEACTIVATED：WiFi 未激活
- CONNECTION_WIFI_STATE_DISCONNECTED：WiFi 已中断
- CONNECTION_WIFI_STATE_CONNECTED：已连接 WiFi

接下来，我们将添加一个新的 Label 小部件，并将上述函数的结果显示在屏幕上。向 create_base_gui() 函数添加新代码。

```
        /* Label-2 */
        ad->label2 = elm_label_add(ad->conform);
        elm_object_text_set(ad->label2, "Cell state");
        evas_object_size_hint_weight_set(ad->label2, EVAS_HINT_EXPAND, 0);
        elm_box_pack_end(vbox, ad->label2);
        evas_object_show(ad->label2);

        /* Label-3 */
        ad->label3 = elm_label_add(ad->conform);
        elm_object_text_set(ad->label3, "Wifi state");
        evas_object_size_hint_weight_set(ad->label3, EVAS_HINT_EXPAND, 0);
        elm_box_pack_end(vbox, ad->label3);
```

```
        evas_object_show(ad->label3);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    int error_code = net_state(ad);
    if (error_code == CONNECTION_ERROR_NONE) {
        cellular_state(ad);
        wifi_state(ad);
    }
}
```

再次运行该示例。当前 WiFi 连接状态将会显示在第三个 Label 小部件中。



4) 相关 API

int connection_create(connection_h* connection) 是一种可创建 connection_h 对象的 API。

int connection_get_type(connection_h connection, connection_type_e* type) 是一种可返回当前通信状态的 API。返回格式为 connection_type_e。connection_type_e 的类型如下：

- CONNECTION_TYPE_DISCONNECTED：已中断
- CONNECTION_TYPE_WIFI：WiFi 类型
- CONNECTION_TYPE_CELLULAR：移动通信类型

- CONNECTION_TYPE_ETHERNET：以太网类型

- CONNECTION_TYPE_BT：蓝牙类型

int connection_destroy(connection_h connection) 是一种可删除 connection_h 对象的 API。

int connection_get_cellular_state(connection_h connection, connection_cellular_state_e* state) 是一种可返回通信状态的 API。返回格式为 cconnection_cellular_state_e。connection_cellular_state_e 的类型如下：

- CONNECTION_CELLULAR_STATE_OUT_OF_SERVICE：已中断

- CONNECTION_CELLULAR_STATE_FLIGHT_MODE：飞行模式

- CONNECTION_CELLULAR_STATE_ROAMING_OFF：漫游关闭

- CONNECTION_CELLULAR_STATE_CALL_ONLY_AVAILABLE ：仅限通话

- CONNECTION_CELLULAR_STATE_AVAILABLE：有可用连接，但尚未连接

- CONNECTION_CELLULAR_STATE_CONNECTED：已连接

int connection_get_wifi_state(connection_h connection, connection_wifi_state_e* state) 是一种可返回 WiFi 通信状态的 API。返回格式为 connection_wifi_state_e。connection_wifi_state_e 的类型如下：

- CONNECTION_WIFI_STATE_DEACTIVATED：WiFi 未激活

- CONNECTION_WIFI_STATE_DISCONNECTED：WiFi 已中断

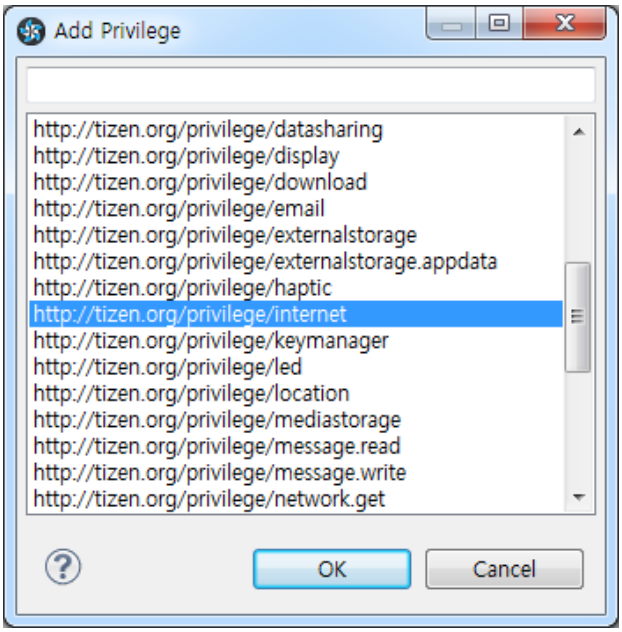
- CONNECTION_WIFI_STATE_CONNECTED：已连接 WiFi

70. HTTP 通信

HTTP 通信的使用方法简便，可同时传送大量数据。因此，在移动通信中亦被广泛采用。在本例中，我们将实施一项通过 HTTP 通信获取天气预报信息并从 Web 服务器下载图像的功能。

1）注册权限

创建新的源项目，将项目名称指定为 HttpRequest。您需要具备用户权限才能使用通信。创建源项目之后，打开 tizen-manifest.xml 文件，在以下选项卡按钮中，点击 Privileges。然后，点击右上方的 Add 按钮。弹出窗口出现后，在列表中选择 http://tizen.org/privilege/internet，然后点击 OK 按钮关闭窗口。



保存之后，在以下选项卡按钮中，点击右侧的 tizen-manifest.xml 按钮。随后，xml 文件的源代码将会出现。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.httprequest" version="1.0.0">
```

```
        <profile name="mobile"/>
        <ui-application appid="org.example.httprequest" exec="httprequest" multi
ple="false" nodisplay="false" taskmanage="true" type="capp">
                <label>httprequest</label>
                <icon>httprequest.png</icon>
        </ui-application>
        <privileges>
                <privilege>http://tizen.org/privilege/internet</privilege>
        </privileges>
</manifest>
```

2) 请求文本通信数据

在本小节中，我们将从 Web 服务器接收文本数据。打开 src 文件夹中的源文件 (~.c)，并添加库头、变量和结构。

```
#include "httprequest.h"
#include <curl/curl.h>

typedef struct MemoryStruct {
        char *memory;
        size_t size;
} memoryStruct;

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        //Evas_Object *label;
        Evas_Object *entry;
        Evas_Object *icon;
        memoryStruct ms;
} appdata_s;
```

对于 HTTP 通信，将使用 CURL 库。curl/curl.h 则是 CURL 库头文件。

memoryStruct 是一种存储通信结果数据的结构。

icon 是一种 Evas Image 对象。

我们现在将要实施一项功能，以连接服务器并接收来自服务器的文本数据。在 create_base_gui() 函数之上添加三个新函数。

```
static size_t
write_memory_cb(void *contents, size_t size, size_t nmemb, void *userp)
{
        size_t realsize = size * nmemb;
        memoryStruct *mem = (memoryStruct *)userp;

        mem->memory = realloc(mem->memory, mem->size + realsize + 1);
        if(mem->memory == NULL) {
                /* out of memory! */
                dlog_print(DLOG_INFO, "tag", "not enough memory (realloc returne
d NULL)");
                return 0;
        }

        memcpy(&(mem->memory[mem->size]), contents, realsize);
        mem->size += realsize;
        mem->memory[mem->size] = 0;
        return realsize;
}

void
get_http_data(const char* url, memoryStruct *data)
{
        CURL *curl_handle;
        CURLcode res;

        data->memory = malloc(1);   /* will be grown as needed by the realloc ab
ove */
        data->size = 0;       /* no data at this point */

        curl_global_init(CURL_GLOBAL_ALL);

        /* init the curl session */
        curl_handle = curl_easy_init();

        /* specify URL to get */
        curl_easy_setopt(curl_handle, CURLOPT_URL, url);

        /* send all data to this function   */
        curl_easy_setopt(curl_handle, CURLOPT_WRITEFUNCTION, write_memory_cb);
```

```
        /* we pass our 'chunk' struct to the callback function */
        curl_easy_setopt(curl_handle, CURLOPT_WRITEDATA, (void *)data);

        /* some servers don't like requests that are made without a user-agent
        field, so we provide one */
        curl_easy_setopt(curl_handle, CURLOPT_USERAGENT, "libcurl-agent/1.0");

        /* get it! */
        res = curl_easy_perform(curl_handle);

        /* cleanup curl stuff */
        curl_easy_cleanup(curl_handle);

        /* we're done with libcurl, so clean it up */
        curl_global_cleanup();
}

static void
btn_download_text(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        char url[100]={0,};

        sprintf(url, "http://api.openweathermap.org/data/2.5/weather?lat=37.498&
lon=127.027&units=metric");
        get_http_data(url, &ad->ms);

        elm_object_text_set(ad->label, ad->ms.memory);
        free( ad->ms.memory);
}
```

write_memory_cb() 是一种可接收服务器响应的事件函数。参数依次为 content、字节单位、内存大小、用户数据。

用第二个参数乘以第三个参数，即可算出总内存大小。字节单位大部分是 1。

下一个代码将通过 realloc() 函数并通过 memcpy() 函数复制数据，以将内存分配到 memoryStruct 结构的 memory 属性中。

随后，还会将内存大小添加到 memoryStruct 结构的 size 属性中，并用 0 来替换数据结尾处，以标记结束。

get_http_data() 函数可尝试与服务器进行通信。

curl_global_init(long flags) 是一种可初始化 CURL 库的 API。对于使用 CURL 的应用程序，必须在一开始就运行一次此 API。

curl_easy_init(void) 是一种可创建 CURL 对象的 API。

curl_easy_setopt(CURL *curl, CURLoption option, ...) 是一种可为 CURL 对象指定选项的 API。该选项的类型如下：
  - CURLOPT_URL：指定 URL 地址。
- CURLOPT_WRITEFUNCTION：指定用于接收通信结果的回调函数。
- CURLOPT_WRITEDATA：指定用户数据。
- CURLOPT_USERAGENT：指定用户代理。

curl_easy_perform(CURL *curl) 是一种可开始与服务器进行通信的 API。

curl_easy_cleanup(CURL *curl) 是一种可删除 CURL 数据的 API。

curl_global_cleanup(void) 是一种可删除 CURL 库全部数据的 API。一旦使用 CURL，则在关闭应用程序之前，必须至少调用一次此 API。

btn_download_text() 函数可在用户点击 Button 后接收来自服务器的文本数据，并将结果显示在 Label 小部件中。

现在，我们来请求纬度坐标为 37.498、经度坐标为 127.027 这个位置的天气信息。

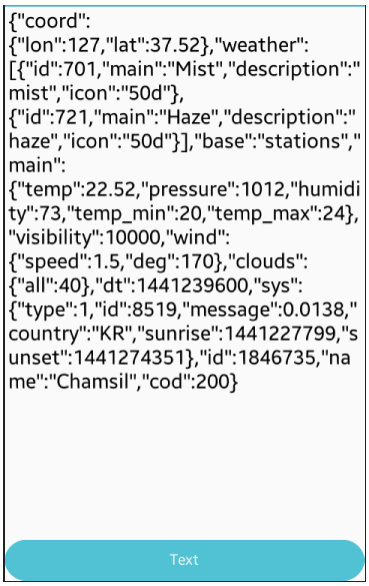
我们将创建一个 Button，用来接收来自天气预报服务器的数据。在 create_base_gui() 函数的结尾添加新代码。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);
```

```
    /* Vertical box */
    Evas_Object *vbox = elm_box_add(ad->win);
    elm_box_padding_set(vbox, ELM_SCALE_SIZE(10), ELM_SCALE_SIZE(10));
    elm_object_content_set(ad->conform, vbox);
    evas_object_show(vbox);

    {
        /* Entry */
        ad->entry = elm_entry_add(ad->conform);
        evas_object_size_hint_weight_set(ad->entry, EVAS_HINT_EXPAND, EVAS_HINT
_EXPAND);
        evas_object_size_hint_align_set(ad->entry, EVAS_HINT_FILL, EVAS_HINT_FI
LL);
        elm_box_pack_end(vbox, ad->entry);
        evas_object_show(ad->entry);

        /* Button-1 */
        Evas_Object *btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Text");
        evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0);
        evas_object_smart_callback_add(btn, "clicked", btn_download_text, ad);
        elm_box_pack_end(vbox, btn);
        evas_object_show(btn);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

我们已创建了 Box、Entry 和 Button 小部件。

我们现在来构建并运行一个示例。点击 Button，收到来自服务器中的响应后，Json 类型的天气信息将会显示在屏幕上。



如需有关如何从 Json 语法中提取个别数据的信息，请参阅 JsonParse 示例。

3) 下载图像

在本小节中，我们将实施一项功能，以从 Web 服务器下载图像，并将图像显示成 Image 对象。在 create_base_gui() 函数中添加 Image 对象创建代码和 Button 小部件创建代码。

```
    {
        /* Entry */
        ad->entry = elm_entry_add(ad->conform);
        evas_object_size_hint_weight_set(ad->entry, EVAS_HINT_EXPAND, EVAS_HINT_
EXPAND);
        evas_object_size_hint_align_set(ad->entry, EVAS_HINT_FILL, EVAS_HINT_FIL
L);
        elm_box_pack_end(vbox, ad->entry);
        evas_object_show(ad->entry);

        /* Image */
        ad->icon = elm_image_add(ad->conform);
        evas_object_size_hint_weight_set(ad->icon, EVAS_HINT_EXPAND, EVAS_HINT_
EXPAND);
        evas_object_size_hint_align_set(ad->icon, EVAS_HINT_FILL, EVAS_HINT_FIL
L);
```

```
        elm_box_pack_end(vbox, ad->icon);
        evas_object_show(ad->icon);

        /* Button-1 */
        Evas_Object *btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Text");
        evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0);
        evas_object_smart_callback_add(btn, "clicked", btn_download_text, ad);
        elm_box_pack_end(vbox, btn);
        evas_object_show(btn);

        /* Button-2 */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Image");
        evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0);
        evas_object_smart_callback_add(btn, "clicked", btn_download_image, ad);
        elm_box_pack_end(vbox, btn);
        evas_object_show(btn);
    }
```

我们创建了一个 Image 对象，并将其储存在 appdata 结构的 icon 变量中。接下来，我们将创建第二个 Button，并将回调函数的名称指定为 btn_download_image。

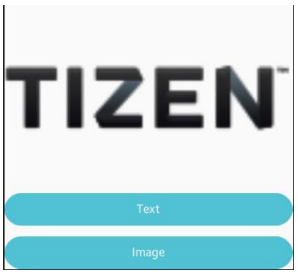
最后，我们需要创建一个 Button 回调函数。在 create_base_gui() 函数之上添加一个新函数。

```
static void
btn_download_image(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        char url[100]={0,};

        sprintf(url, "https://www.tizen.org/sites/all/themes/tizen_theme/logo.png");
        get_http_data(url, &ad->ms);

        // update icon image.
        if ( elm_image_memfile_set( ad->icon, ad->ms.memory , ad->ms.size, "png", 0) == EINA_FALSE)
```

```
        {
                dlog_print(DLOG_DEBUG, "tag", "%s : image setting error " , __fu
nc__);
        }
        free( ad->ms.memory);
}
```

elm_image_memfile_set(Evas_Object *obj, const void *img, size_t size, const char *format, const char *key) 是一种可在 Evas Image 对象中输入原始图像数据的 API。参数依次为 Evas Image 对象、图像数据、数据大小和图像类型。

构建示例并点击第二个 Button。从 Web 服务器下载的 Tizen 徽标图像将会显示在屏幕上。



4) 相关 API

CURLcode curl_global_init(long flags)：一种可初始化 CURL 库的 API。对于使用 CURL 的应用程序，必须在一开始就运行一次此 API。

CURL* curl_easy_init(void)：一种可创建 CURL 对象的 API。

CURLcode curl_easy_setopt(CURL *curl, CURLoption option, ...)：一种可为 CURL 对象指定选项的 API。该选项的类型如下：

- CURLOPT_URL：指定 URL 地址。
- CURLOPT_WRITEFUNCTION：指定用于接收通信结果的回调函数。
- CURLOPT_WRITEDATA：指定用户数据。
- CURLOPT_USERAGENT：指定用户代理。

CURLcode curl_easy_perform(CURL *curl)：一种可开始与服务器进行通信的 API。

void curl_easy_cleanup(CURL *curl)：一种可删除 CURL 数据的 API。

void curl_global_cleanup(void)：一种可删除 CURL 库全部数据的 API。一旦使用 CURL，则在关闭应用程序之前，必须至少调用一次此 API。

Eina_Bool elm_image_memfile_set(Evas_Object *obj, const void *img, size_t size, const char *format, const char *key)：一种可在 Evas Image 对象中输入原始图像数据的 API。参数依次为 Evas Image 对象、图像数据、数据大小和图像类型。

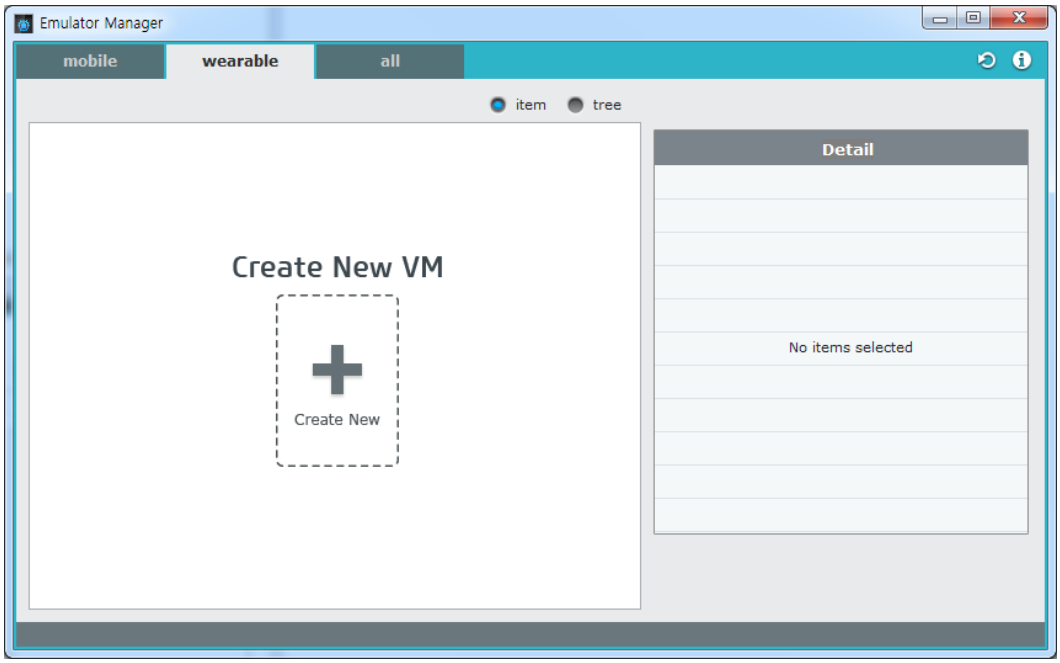
71. 创建穿戴式源项目

在该示例中，我们将学习如何使用创建穿戴式模拟器以及穿戴式源项目。

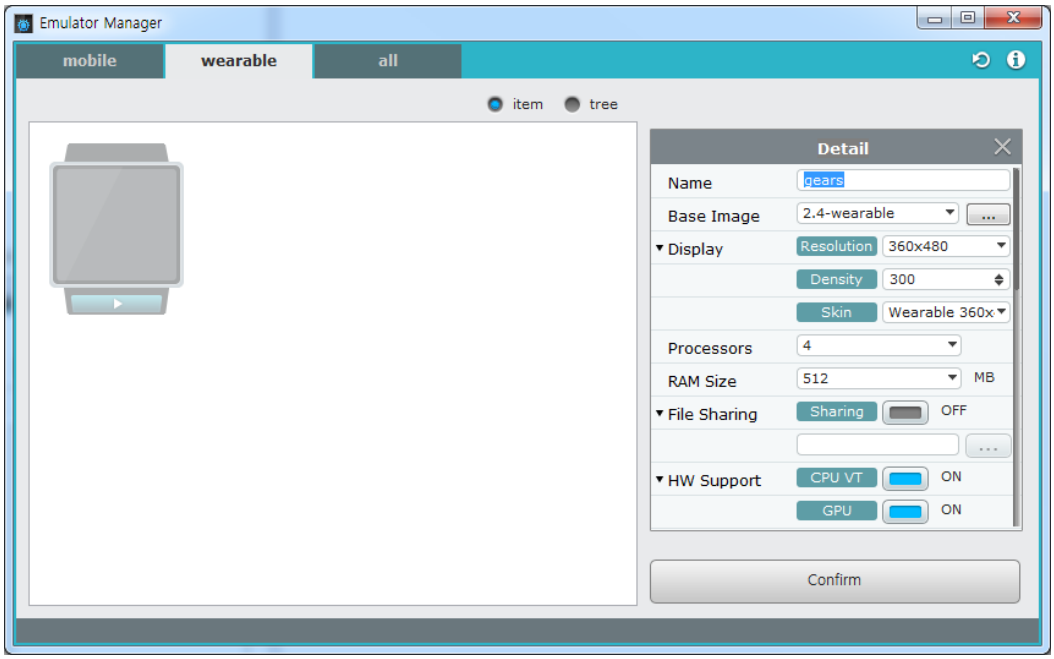
A. 运行穿戴式模拟器

在本节，我们将学习如何创建并运行模拟器。单击 [Windows“开始”按钮 > 所有程序 > Tizen SDK > Emulator Manager]。如弹出警告窗，忽略并点击 Yes 按钮。

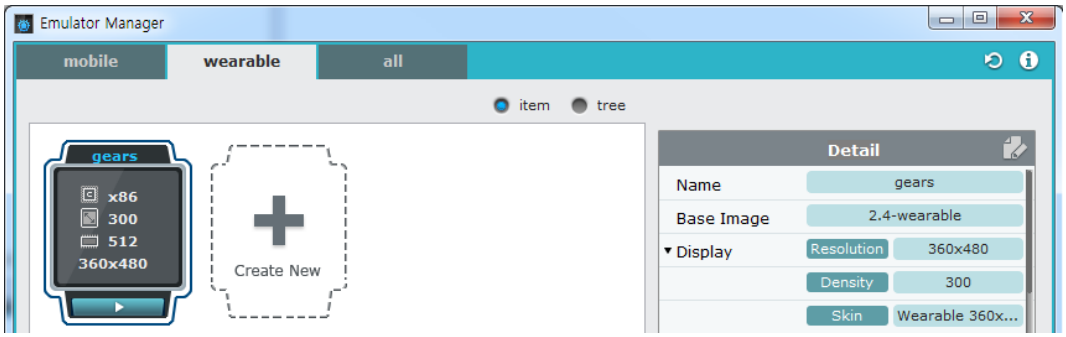
Emulator Manager 运行后，点击第二个选项卡按钮“穿戴式”。不存在穿戴式模拟器时，则需创建一个模拟器。单击左侧“Create New VM”下面的 + 号。



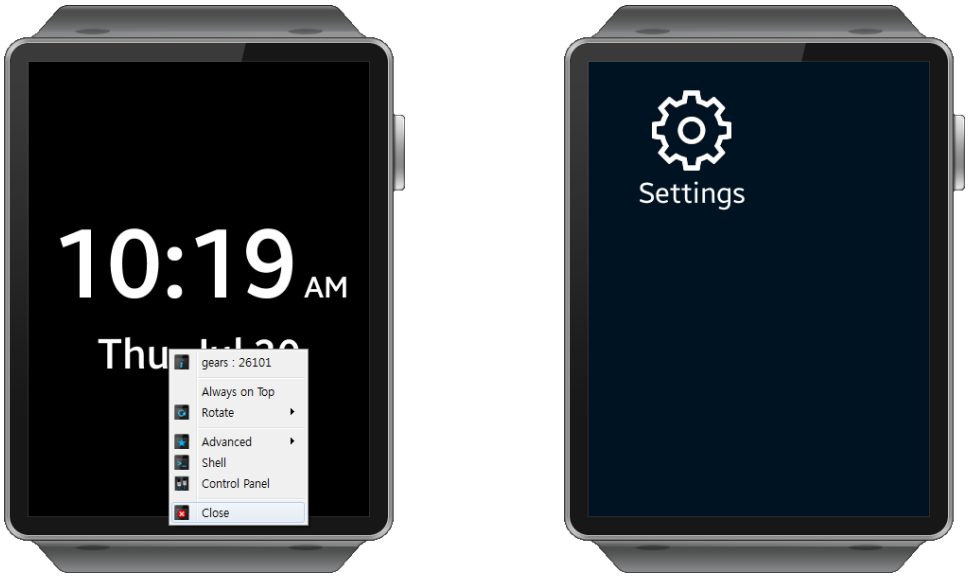
然后，您将看到右侧出现一个屏幕，供您指定模拟器的选项。将 gears 指定为名称属性。您可以将剩余内容保留为默认属性。单击 Confirm 按钮创建新模拟器。



现在，您会看到屏幕左侧已创建一个名为 gears 的新模拟器。单击新模拟器图标下方的箭头按钮以运行该模拟器。若要更改模拟器的选项，点击 Reset 按钮即可。



如果弹出‘Windows Security Warning’窗口，点击击 Unblock 按钮继续此流程。模拟器的右上侧有一个电源按钮。它具有 Power 和 Home 两种功能。按一次可打开/关闭屏幕，持续按下可终止模拟器。



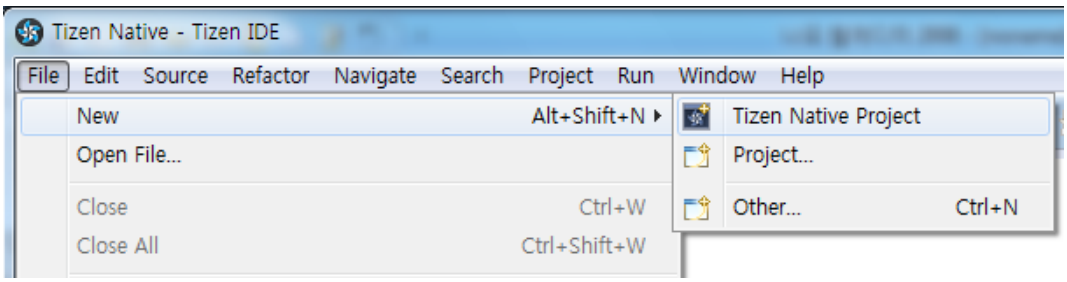
当设备切换到睡眠模式后，按下电源键或按鼠标右键然后在快捷菜单中选择 Close，也可终止模拟器。

向上轻弹可显示应用程序列表，向下轻弹可返回至上一屏幕。

B. 现在，我们将创建一个新的源项目，并会创建一个请求 Button 单击事件的示例。

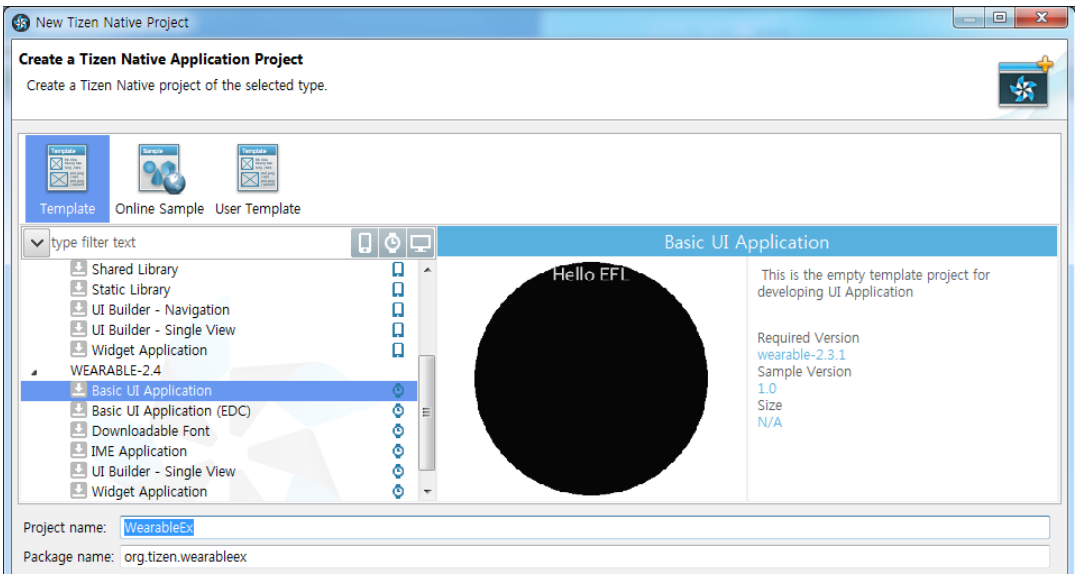
1）创建源项目

第一步是要创建一个新的源项目。在 Eclipse 的主菜单中选择 [File > New > Tizen Native Project]。



弹出源项目创建窗口时，选择 [Template > WEARABLE-2.x > Basic UI Application]。

在“Project name”字段中输入“WearableEx”。

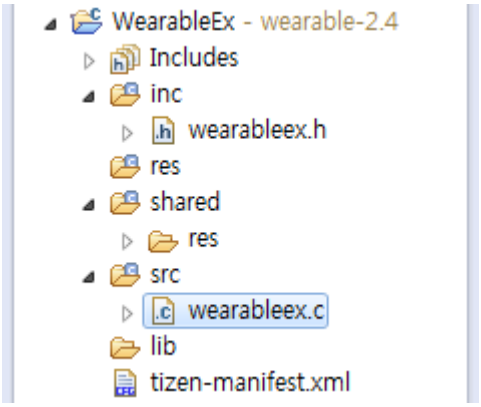


系统随即会自动填充“Package name”字段。现在，点击 Finish 按钮就会创建一个新的源项目。

2) 源项目的组件

穿戴式应用程序的基本源项目与移动应用程序的源项目非常类似。“\inc”文件夹中包含各种库。
此文件夹中包含采用 C 语言 (.h) 的头文件。通常，您可使用此文件夹定义各种库、函数的头文件以及全局变量。
“\res”文件夹通常用于保存包含图像和音频文件在内的资源文件。
“\src”文件夹中包含采用 C 语言 (.c) 的源文件。此文件夹主要用于定义函数的功能。大多数任务都是在此处执行的。
“\shared”文件夹中包含应用程序图标图像。在将您的应用程序分发至应用程序商店后，即可在此处保存您的应用程序图标。对于 Tizen Store，您应当使用圆形的应用程序图标。

“tizen-manifest.xml”文件储存着应用程序的各种信息（如应用程序的名称和版本）以及用户权利（权限）。。基本上，该文件等同于 Android 系统中的 AndroidManifest.xml。



3）运行基本源项目

现在，我们将运行首次创建时的源项目。右键单击 WearableEx 项目，然后在快捷菜单中选择 [Build Project]。成功项目构建后，右键单击该项目，然后在快捷菜单中选择 [Run As > 1 Tizen Native Application]。如果没有安装证书，可以使用与移动模拟器相同的方式安装证书。

在模拟器中运行该示例后，会在顶部看到“Hello EFL”的字样。使用 Label 小部件时即会显示此文本。

4) 更改 Label 中的文本

现在，让我们更改一下 Label 小部件中显示的“Hello EFL”文本。要执行此操作，我们需要编辑源文件。打开 src 文件夹，双击“wearableex.c”文件。现在，在 Eclipse 的主屏幕上，您将会看到如下所示的文件内容。此源代码与用于移动应用程序的源代码类似，因此请参阅本教材开头部分给出的具体说明。

现在，我们将更改屏幕上显示的“Hello EFL”文本。按如下所示修改 create_base_gui() 函数：

```
        ad->label = elm_label_add(ad->conform);
        //elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
        elm_object_text_set(ad->label, "Hello World");
        evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_
EXPAND);
        elm_object_content_set(ad->conform, ad->label);

        /* Show window after base gui is set up */
        evas_object_show(ad->win);
}
```

elm_object_text_set() 是一种用于更改小部件标题文本的 API。您可以将此函数用于 Button、Entry 以及 Label 小部件。您还能在创建文本时通过使用 HTML 标记来指定文本属性。

让我们再运行一次示例。当您再次运行示例时，请在主菜单中单击 [Run > Run]，或按下“Ctrl + F11”热键组合。此示例将再次运行，屏幕上的文本则会变为“Hello World”。

5) 添加 Button 小部件

在本小节中，我们将实施一项功能，以便在单击 Button 时更改 Label 的文本。方法与移动应用程序相同。向 create_base_gui() 函数添加新代码。

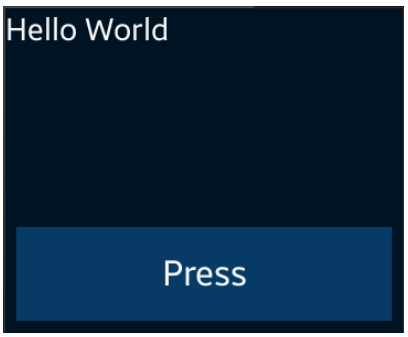
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    /* Box */
    Evas_Object *box = elm_box_add(ad->win);
    elm_box_padding_set(box, ELM_SCALE_SIZE(10), ELM_SCALE_SIZE(10));
    elm_object_content_set(ad->conform, box);
    evas_object_show(box);

    {
        /* Label*/
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "Hello World");
        evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT
_EXPAND);
        evas_object_size_hint_align_set(ad->label, EVAS_HINT_FILL, 0.0);
        elm_box_pack_end(box, ad->label);
        evas_object_show(ad->label);

        /* Button */
        Evas_Object *btn = elm_button_add(ad->conform);
```

```
        elm_object_text_set(btn, "Press");
        evas_object_smart_callback_add(btn, "clicked", btn_click_cb, ad);
        evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAN
D);
        evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0.0);
        elm_box_pack_end(box, btn);
        evas_object_show(btn);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

此代码将创建一个 Box 容器和一个 Button 小部件。我们已将单击事件回调函数的名称指定为“btn_click_cb”。现在将创建此函数。在 create_base_gui() 函数之上添加一个新函数。

```
static void
btn_click_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        elm_object_text_set(ad->label, "Button Pressed");
}
```

此代码会在单击 Button 时将 Label 小部件的文本更改为“Button Pressed”。再次运行此示例并点击 Button。Label 的文本发生改变。Basic UI 的使用方法与移动应用程序几乎相同。



72. 穿戴式设备的系统信息

在教材的开头部分，我们学习了如何请求移动系统信息。在本示例中，我们将编译与穿戴式应用程序相同的应用程序，并查看哪些部分是相同的，以及哪些部分是不相同的。

1) 是否存在后置摄像头

创建一个新的源项目，然后将类型指定为 Basic UI Application，并将项目名称指定为 wSystemInfo。创建源项目之后，打开 src 文件夹中的源文件 (~.c)，并在源文件顶端添加一个库头文件和变量。

```
#include "systeminfo.h"
#include <system_info.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label1;
        Evas_Object *label2;
        Evas_Object *label3;
        Evas_Object *label4;
        Evas_Object *label5;
} appdata_s;
```

我们共公布了五个 Label 小部件变量。在第一个 Label 中将显示是否存在后置摄像头；在第二个 Label 中显示是否可进行电话呼叫；在第三个 Label 中显示此显示器的水平像素数；在第四个 Label 中显示此显示器的垂直像素数；在第五个 Label 中则显示平台的版本。

将新代码添加到 create_base_gui() 函数中。此代码将创建一个 Button 小部件和五个 Label 小部件。

```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
```

```
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, ad->conform);
    evas_object_show(ad->conform);

    /* Box */
    Evas_Object *box = elm_box_add(ad->win);
    elm_box_padding_set(box, ELM_SCALE_SIZE(10), ELM_SCALE_SIZE(10));
    elm_object_content_set(ad->conform, box);
    evas_object_show(box);

    {
        /* Button */
        Evas_Object *btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "System Info");
        evas_object_smart_callback_add(btn, "clicked", btn_clicked_cb, ad);
        evas_object_size_hint_weight_set(btn, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, 0);
        elm_box_pack_end(box, btn);
        evas_object_show(btn);

        /* Label-1 */
        ad->label1 = elm_label_add(ad->conform);
        elm_object_text_set(ad->label1, "Back Camera :");
        evas_object_size_hint_weight_set(ad->label1, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(ad->label1, EVAS_HINT_FILL, 0);
        elm_box_pack_end(box, ad->label1);
        evas_object_show(ad->label1);

        /* Label-2 */
        ad->label2 = elm_label_add(ad->conform);
        elm_object_text_set(ad->label2, "Telephony :");
        evas_object_size_hint_weight_set(ad->label2, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(ad->label2, EVAS_HINT_FILL, 0);
        elm_box_pack_end(box, ad->label2);
        evas_object_show(ad->label2);

        /* Label-3 */
        ad->label3 = elm_label_add(ad->conform);
        elm_object_text_set(ad->label3, "Pixel Width :");
        evas_object_size_hint_weight_set(ad->label3, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(ad->label3, EVAS_HINT_FILL, 0);
        elm_box_pack_end(box, ad->label3);
        evas_object_show(ad->label3);
```

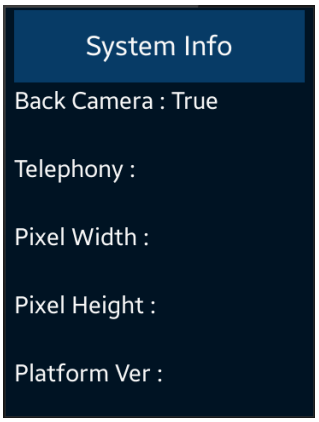
```
        /* Label-4 */
        ad->label4 = elm_label_add(ad->conform);
        elm_object_text_set(ad->label4, "Pixel Height :");
        evas_object_size_hint_weight_set(ad->label4, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(ad->label4, EVAS_HINT_FILL, 0);
        elm_box_pack_end(box, ad->label4);
        evas_object_show(ad->label4);

        /* Label-5 */
        ad->label5 = elm_label_add(ad->conform);
        elm_object_text_set(ad->label5, "Platform Ver :");
        evas_object_size_hint_weight_set(ad->label5, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(ad->label5, EVAS_HINT_FILL, 0);
        elm_box_pack_end(box, ad->label5);
        evas_object_show(ad->label5);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);
}
```

我们现在将创建一个 Button 回调函数。在 create_base_gui() 函数之上添加新代码。

```
static void
btn_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
        appdata_s *ad = data;
        char buf[100];
        char *sValue = NULL;
        bool bValue = false;
        int nValue = 0;
        int ret;

        ret = system_info_get_platform_bool("http://tizen.org/feature/camera.bac
k", &bValue);
        if (ret == SYSTEM_INFO_ERROR_NONE)
        {
                sprintf(buf, "Back Camera : %s", bValue ? "True" : "False");
                elm_object_text_set(ad->label1, buf);
        }
}
```

system_info_get_platform_bool(char *, bool *) 是一种可请求系统信息的 API。返回的数据类型是布尔型。第一个参数是键值，并将“http://tizen.org/feature/camera.back”传递给第一个参数，以返回是否存在后置摄像头。

构建并运行该示例。点击此 Button，您会看到第一个 Label 的文本发生了变化。在模拟器上点击 Button，将会显示“False”，而在用户设备上点击 Button，则会显示“True”。



2）是否有电话功能

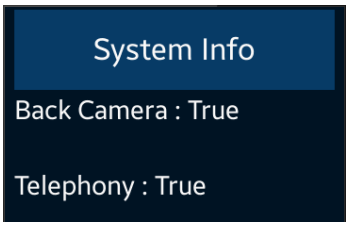
在本小节中，我们将检查是否具有电话功能。在 btn_clicked_cb() 函数的结尾添加新代码。

```
        if (ret == SYSTEM_INFO_ERROR_NONE)
        {
                sprintf(buf, "Back Camera : %s", bValue ? "True" : "False");
                elm_object_text_set(ad->label1, buf);
        }

        ret = system_info_get_platform_bool("http://tizen.org/feature/network.telephony", &nValue);
        if (ret == SYSTEM_INFO_ERROR_NONE)
        {
                sprintf(buf, "Telephony : %s", bValue ? "True" : "False");
                elm_object_text_set(ad->label2, buf);
        }
```

```
}
```

将“http://tizen.org/feature/network.telephony”传递给 system_info_get_platform_bool() 函数的第一个参数，将会返回是否具有电话功能。即使返回的值为 True，这并不表明您可以进行电话呼叫或可使用网络。这只是意味着设备配备了硬件通信功能。如果设备没有 USIM 芯片或在 Settings 中停用了网络功能，则无法使用通信功能。

构建并运行该示例。按下此 Button，然后您将会看到在第二个 Label 上显示了文本“True”。



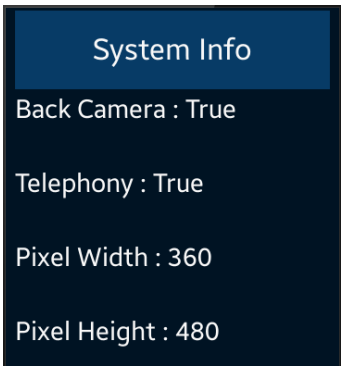
4) 显示器的像素数

在本小节中，我们将请求显示器的像素数。在 btn_clicked_cb() 函数的结尾添加新代码。

```
        if (ret == SYSTEM_INFO_ERROR_NONE)
        {
                sprintf(buf, "Telephony : %s", bValue ? "True" : "False");
                elm_object_text_set(ad->label2, buf);
        }

        ret = system_info_get_platform_int("tizen.org/feature/screen.width", &n
Value);
        if (ret == SYSTEM_INFO_ERROR_NONE)
        {
                sprintf(buf, "Pixel Width : %d", nValue);
                elm_object_text_set(ad->label3, buf);
        }

        ret = system_info_get_platform_int("tizen.org/feature/screen.height", &
nValue);
        if (ret == SYSTEM_INFO_ERROR_NONE)
        {
                sprintf(buf, "Pixel Height : %d", nValue);
                elm_object_text_set(ad->label4, buf);
        }
}
```

system_info_get_platform_int(char *, int *) 是一种可请求系统信息的 API。返回的数据类型为整数。第一个参数为键值，若传递“http://tizen.org/feature/screen.width”，则返回显示器的水平像素数。若传递“tizen.org/feature/screen.height”，则返回显示器的垂直像素数。

构建并运行该示例。按下此 Button，然后您将会看到在第三个和第四个 Label 上显示了相应的数字。



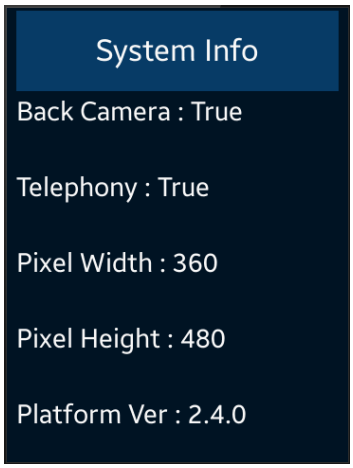
5) 平台版本

在本小节中，我们将请求平台的版本。在 btn_clicked_cb() 函数的结尾添加新代码。

```
        if (ret == SYSTEM_INFO_ERROR_NONE)
        {
                sprintf(buf, "Pixel Height : %d", nValue);
                elm_object_text_set(ad->label4, buf);
        }

        ret = system_info_get_platform_string("http://tizen.org/feature/platfor
m.version", &sValue);
        if (ret == SYSTEM_INFO_ERROR_NONE)
        {
                sprintf(buf, "Platform Ver : %s", sValue);
                elm_object_text_set(ad->label5, buf);
        }
}
```

system_info_get_platform_string(char *, char **) 是一种可请求系统信息的 API。返回的数据类型为字符串。第一个参数是键值，并传递“http://tizen.org/feature/platform.version”以返回平台的版本。

构建并运行该示例。按下此 Button，然后您将会看到在第五个 Label 上显示了平台的版本。



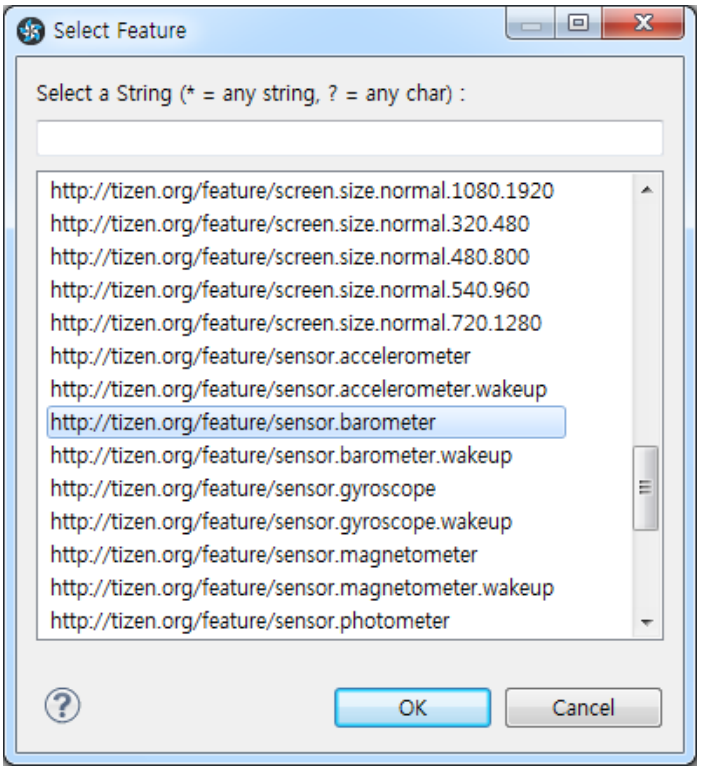
现在我们已了解到，用于移动应用程序的 API 也可用于穿戴式设备的应用程序。

73. 使用压力传感器

穿戴式设备中新增了一些智能手机不具备的功能，例如心率传感器、压力传感器等。在本例中，我们将学习如何使用压力传感器。

1）添加功能

创建一个新的源项目，并将项目名称指定为“wSensorPressure”。要使用压力传感器功能，您需要添加相关功能。创建源项目之后，打开 tizen-manifest.xml 文件，并在以下选项卡按钮中点击 Features。然后，点击右上方的 Add 按钮。弹出窗口出现后，在列表中选择 http://tizen.org/feature/sensor.barometer，然后点击 OK 按钮关闭窗口。



重复上述过程，添加以下功能。

- http://tizen.org/feature/sensor.barometer.wakeup

保存之后，在以下选项卡按钮中，点击右侧的 tizen-manifest.xml 按钮。

随后，xml 文件的源代码将会出现。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3.1" package="org.
example.wsensorpressure" version="1.0.0">
        <profile name="wearable"/>
        <ui-application appid="org.example.wsensorpressure" exec="wsensorpressur
e" multiple="false" nodisplay="false" taskmanage="true" type="capp">
                <label>wsensorpressure</label>
                <icon>wsensorpressure.png</icon>
        </ui-application>
        <feature name="http://tizen.org/feature/sensor.barometer">true</feature>
        <feature name="http://tizen.org/feature/sensor.barometer.wakeup">true</
feature>
</manifest>
```

2) 确定是否支持压力传感器

创建一个新的源项目，并将项目名称指定为“wSensorPressure”。创建源项目之后，打开 src 文件夹中的源文件 (*.c)，并添加库头文件与变量。

```
#include "wsensorpressure.h"
#include <sensor.h>

typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
} appdata_s;
```

sensor.h 是用于各传感器库的头文件。

我们将在 label0 中显示是否支持压力传感器，在 label1 中显示压力等级。

在 create_base_gui() 函数之上添加一个新函数。

```
```

```
static void show_is_supported(appdata_s *ad)
{
        char buf[PATH_MAX];
        bool is_supported = false;
        sensor_is_supported(SENSOR_PRESSURE, &is_supported);
        sprintf(buf, "Pressure Sensor is %s", is_supported ? "support" : "not su
pport");
        elm_object_text_set(ad->label0, buf);
}
```

show_is_supported() 函数在确定是否支持压力传感器后，将在第一个 Label 小部件中显示结果。

sensor_is_supported(sensor_type_e, bool *) 是一种请求是否支持特定传感器的 API。将 SENSOR_PRESSURE 传递给第一个参数，则会让第二个参数返回是否支持压力传感器。

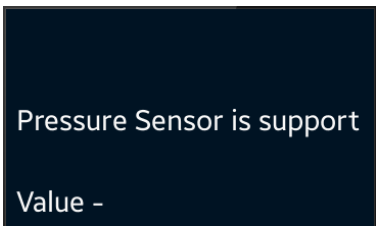
这样做才能在运行应用程序调用此函数。在 create_base_gui() 函数末尾调用以上函数。

```
        /* Conformant */
        ad->conform = elm_conformant_add(ad->win);
        elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
        elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
        evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HIN
T_EXPAND);
        elm_win_resize_object_add(ad->win, ad->conform);
        evas_object_show(ad->conform);

    /* Box */
    Evas_Object *box = elm_box_add(ad->win);
    elm_box_padding_set(box, ELM_SCALE_SIZE(10), ELM_SCALE_SIZE(10));
    elm_object_content_set(ad->conform, box);
    evas_object_show(box);

    {
        /* Label-0 */
        ad->label0 = elm_label_add(ad->conform);
        elm_label_line_wrap_set(ad->label0, EINA_TRUE);
        elm_object_text_set(ad->label0, "Msg - ");
        //evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HI
NT_EXPAND);
```

```
        //elm_object_content_set(ad->conform, ad->label);
        evas_object_size_hint_weight_set(ad->label0, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(ad->label0, EVAS_HINT_FILL, 0);
        elm_box_pack_end(box, ad->label0);
        evas_object_show(ad->label0);

        /* Label-1 */
        ad->label1 = elm_label_add(ad->conform);
        elm_label_line_wrap_set(ad->label1, EINA_TRUE);
        elm_object_text_set(ad->label1, "Value - ");
        evas_object_size_hint_weight_set(ad->label1, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(ad->label1, EVAS_HINT_FILL, 0);
        elm_box_pack_end(box, ad->label1);
        evas_object_show(ad->label1);
    }

    /* Show window after base gui is set up */
    evas_object_show(ad->win);

        show_is_supported(ad);
}
```

我们创建了一个 Box 容器和两个 Label 小部件。还调用了用于确定支持传感器的此函数。

构建并运行该示例。如果支持压力传感器，则会显示消息“Pressure Sensor is supported”。有些型号可能不支持此传感器。在这种情况下，您需要在模拟器上进行测试。



2) 请求压力传感器事件

在本小节中，我们将实施一项功能，以便在压力传感器检测到对象时请求相关事件，然后在屏幕上显示距离。在源文件顶端添加一个传感器相关结构和全局变量。

```
typedef struct appdata {
        Evas_Object *win;
        Evas_Object *conform;
        Evas_Object *label0;
        Evas_Object *label1;
} appdata_s;

typedef struct _sensor_info
{
        sensor_h sensor;                /**< Sensor handle */
        sensor_listener_h sensor_listener;
} sensorinfo;

static sensorinfo sensor_info;
```

sensorinfo 是一种包含传感器对象和事件侦听器变量的结构。

sensor_info 是 sensorinfo 结构的全局变量。

请求传感器事件则意味着要启动一个侦听器。现在我们将使用一个传感器对象和一个事件侦听器来请求一个压力传感器事件。在 create_base_gui() 函数之上添加两个新函数。

```
static void _new_sensor_value(sensor_h sensor, sensor_event_s *sensor_data, void
 *user_data)
{
        if( sensor_data->value_count < 1 )
                return;
        char buf[PATH_MAX];
        appdata_s *ad = (appdata_s*)user_data;

        sprintf(buf, "Pressure : %0.1f hPa", sensor_data->values[0]);
        elm_object_text_set(ad->label1, buf);
}

static void
start_pressure_sensor(appdata_s *ad)
{
        sensor_error_e err = SENSOR_ERROR_NONE;
        sensor_get_default_sensor(SENSOR_PRESSURE, &sensor_info.sensor);
        err = sensor_create_listener(sensor_info.sensor, &sensor_info.sensor_lis
```

```
tener);
        sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sens
or_value, ad);
        sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() 是压力传感器事件的回调函数。它在屏幕上显示新的传感器值。

将传感器数据传递给第二个参数。数字数据保存在 values[0] 中。

start_pressure_sensor() 函数将启动压力传感器并指定事件回调函数。

sensor_get_default_sensor(sensor_type_e, sensor_h *) 是一种可返回传感器对象的 API。将 SENSOR_PRESSURE 传递给第一个参数，则会让第二个参数返回压力传感器对象。

sensor_create_listener(sensor_h, sensor_listener_h *) 是一种可创建事件侦听器的 API。将传感器对象传递给第一个参数，则会让第二个参数返回侦听器对象。

sensor_listener_set_event_cb(sensor_listener_h, unsigned int, sensor_event_cb, void *) 是一种可为侦听器指定回调函数的 API。参数依次包括事件侦听器、时间间隔（单位：毫秒）、回调函数名称和用户数据。

sensor_listener_start(sensor_listener_h) 是一种启动侦听器的 API。

这样做才能在运行应用程序时自动运行事件侦听器。在 create_base_gui() 函数末尾调用以上函数。

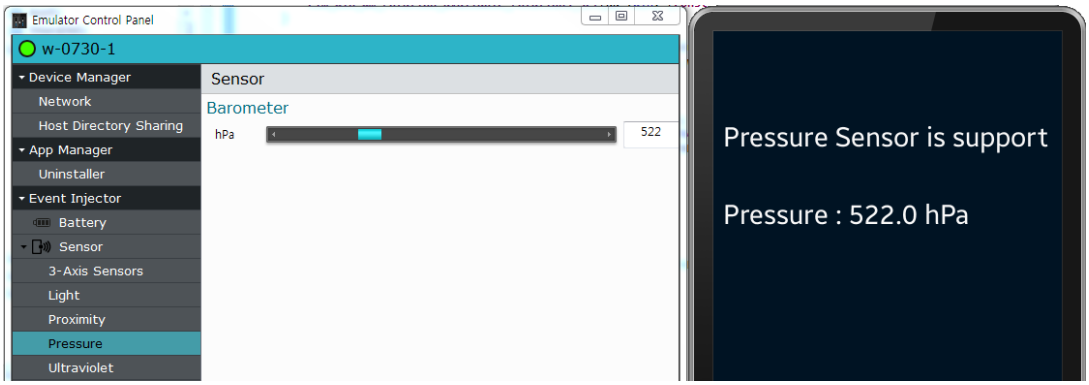
```
        /* Show window after base gui is set up */
        evas_object_show(ad->win);

        show_is_supported(ad);
        start_pressure_sensor(ad);
}
```

让我们再运行一次示例。在模拟器中测试此功能时，请使用 Control Panel。

右键单击模拟器，从快捷方式菜单中选择 Control Panel。

当 Control Panel 出现时，从左侧的树状列表中选择 [Event Injector > Pressure]。

在 Control Panel 右窗格中，向左右拖动触控条。如果模拟器第二个 Label 上的数字发生变化，则意味着该功能已成功实施。



3) 相关 API

int sensor_is_supported(sensor_type_e type, bool *supported)：一种请求是否支持特定传感器的 API。将 SENSOR_PRESSURE 传递给第一个参数，则会让第二个参数返回是否支持压力传感器。

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor)：一种返回传感器对象的 API。将 SENSOR_PRESSURE 传递给第一个参数，则会让第二个参数返回压力传感器对象。

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener)：一种创建事件侦听器的 API。将传感器对象传递给第一个参数，则会让第二个参数返回侦听器对象。

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data)：一种可为侦听器指定回调函数的 API。/ 参数：事件侦听器、时间间隔（单位：毫秒）、回调函数名称和用户数据。

int sensor_listener_start(sensor_listener_h listener)：一种启动侦听器的 API。