TIZEN

Guide to Developing Tizen Native Application

A complete analysis of the Tizen platform with 70 easily understandable examples.

Author: Jung, Dong-Geun (Denis.Jung)

Table Of Contents

Table Of Contents 2

1. Introducing EFL 6

2. Establishing a Tizen Development Environment 11

3. Running a Tizen Development Environment 22

4. Executing Examples 31

5. Creating a BasicUI example 42

6. Using a Label Widget 65

7. Using a Button Widget 73

8. Creating a Background with a Bg Widget 90

9. Resizing the Screen with a Conformant Container 102

10. Using an Entry Widget 110

11. Using a Check Widget 123

12. Using a Radio Widget 135

13. Using a Popup 148

14. Using a Slider Widget 168

15. Adding a Text Item to the List Widget. 177

16. Displaying an Icon in a GenList Widget 188

17. Creating a Complex Gallery Widget 202

18. Creating a Simple Web Browser Using the WebView Widget 215

19. Implementing a Tab Screen with a Layout Container 227

20. Implementing a Header and a Navigation Bar with a Naviframe Widget ... 236

21. Placing Widgets in Sequence with the Box Container 248

22. Creating a Sub Page with a Scroller Widget 260

23. Using Strings 273

24. The String Structure 'Eina_Strbuf' 285

25. Array Structure Eina_List 295

26. Using Timers 309

27. Time & Date 316

28. Calendar Example 331

29. Requesting a Mouse Touch Event 346

30. Calculator Example 356

31. Displaying a Gradient on the Canvas 373

32. Displaying a Square on the Canvas 384

33. Displaying Polygons on the Canvas 389

34. Displaying Text on the Canvas 397

35. Displaying an Image on the Canvas 402

36. Creating a Customized Button 412

37. Using an Animator 430

38. Playing Audio 452

39. Playing Video 468

40. Recording Audio 488

41. Camera Capture 509

42. System Information 532

43. System Preferences 546

44. Battery Status 557

45. Generating Vibration 567

46. LED Flash Backlight 580

47. Event for Rotating the Screen Orientation 589

48. Hardware Key Events and Debug Mode 601

49. Lifecycle and Debug Mode 611

50. Using a Notify 622

51. Using a Acceleration Sensor 638

52. Using a Gravity Sensor 653

53. Using an Orientation Sensor 662

54. Using a Magnetic Sensor 674

55. Using a Proximity Sensor 686

56. Using a GPS Sensor 695

57. Google Maps Library 708

58. Reading and Writing Text Files 753

59. Requesting a File List 767

60. Using Preference 786

61. Example of Making a Report Card with SQLite 797

62. Calling an External App Using AppControl 818

63. Developing a Service App 842

64. Alarm: Starting an APP at a Specified Time 852

65. Text to Speech (TTS) 867

66. Multi-Language Support 882

67. JSON Parsing 897

68. XML Parsing 907

69. Checking Network Status 923

70. HTTP Communication 936

71. Creating a Wearable Source Project 949

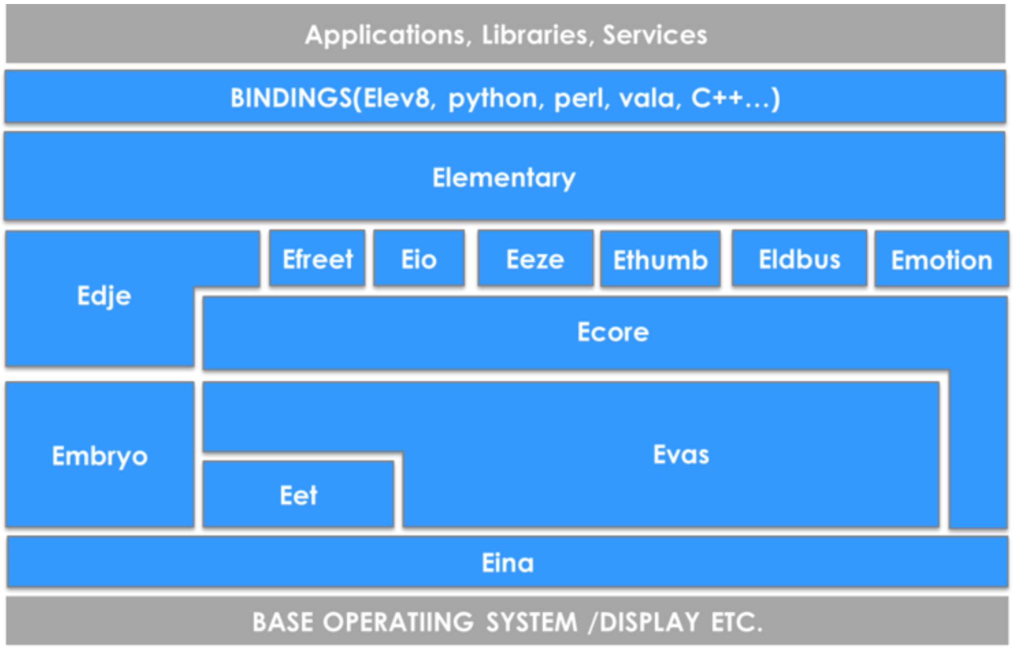
72. Wearable System Information 959

73. Using a Pressure Sensor 968

1. Introducing EFL

The core part of Tizen Native API consists of EFL, which is a complex library integrating multiple libraries.

The following picture is the diagram of an EFL block.


Applications, Libraries, Services
BINDINGS(Elev8, python, perl, vala, C++...)
Elementary
Edje
Efreet
Eio
Eeze
Ethumb
Eldbus
Emotion
Ecore
Embryo
Evas
Eet
Eina
BASE OPERATIING SYSTEM /DISPLAY ETC.

An upper library refers to its sub libraries. For instance, Elementary relies on all of the sub libraries, while Encore refers to Evas, Eet, and Eina.

1) Eina

Eina, one of the most basic elements of EFL, is a data structure library. Similar to STL of C++, it provides users with useful features to implement complex logics including alignment, list, hash, tree, and shared strings in an easy, fast, and safe way. These features of data structure allow applications, as well as all EFL libraries in the upper level of Eina library, to effectively practice necessary logics.

2) Eet

Eet performs the role of data encoding and decoding. You can save arbitrary data structures or image data as compressed files, or transfer them to a different machine via a network. You can also read the compressed files and decode them later. Using this compression feature of Eet which is very similar to the principle of .Zip files, you can instantly access data located in an arbitrary place from a file. If you use the encryption function, you can store data stably.

3) Evas

Evas plays the role of a canvas. Evas enables you to express images, texts, and different shapes including squares, lines, and polygons within a Window. All outputs are objectified. As Evas provides these objects as an Evas_Object type, you can access all graphic objects using the interface of Evas_Object during programming, and express desired graphic objects on the screen. In addition, the interface is provided for Evas so that each object properly responds to events that you input.

Evas adopts a retained-mode as a rendering method, which manages objects throughout the scene graph internally, and automatically optimizes and renders objects to be seen on the screen. Thus, app developers are not encumbered by a complicated rendering mechanism.

Basically, Evas supports the rendering methods of a software, but also helps the acceleration of hardware via back-end supports.

4) Ecore

As a system-based library supporting users' convenience, Ecore provides functions related to main loop, timing, event, connection, IPC, thread, Windows systems, and so on. Ecore internally processes complicated settings and stages of use related to the API of each system. This simplification helps users to save time and effort when they practice the functions of the system in person.

EFL applications work based on a main loop provided by Ecore. As the functions provided by Ecore are processed in connection with the main logic of Ecore's main loop, if necessary, you should use not the system API, but the functions provided by Ecore.

5) Edje

Edje provides functions to compose a complicated GUI. Edje uses EDC as a script language. Referring to the EDC script, you can separate the GUI from the program code to design a program. EDC is converted to the type of EDJ binary throughout the edje_cc compiler. The program can read this EDJ file during the run time to bind it as an Evas_Object to build a GUI. Due to this Edje feature, you can change the design of the GUI in an application without complying it again.

6) Embryo

Embryo, a virtual machine of bytecode, is used for a small program which can be implemented within an EDC file. In most cases, you can additionally use Embryo script language with C-language style to practice simple calculations or functions like changing the status of each object. This kind of Embryo script is converted to PAWN binary via the PAWN compiler. The PAWN program does not need to reply on the environments of a machine; instead, it can be deciphered via the virtual machine of Abstract Machine eXecutive (AMX) for operation. In addition, only one output guarantees the same operation in different system environments.

7) Emotion

Emotion is the library of video and audio playback. Emotion plays a video using other video play plug-ins including Gstreamer, Xine, or VLC. The video outputs are synchronized with Evas objects to be provided to a user. Accordingly, the user can play the video, and also easily compose the screen in connection with the GUI.

8) Elementary

Elementary is a widget toolkit library, providing widgets for general use such as buttons, lists, labels, and sliders. In addition, it provides the change of dynamic themes and the scalability of GUI to support 'Look & Feel' and various screen resolutions respectively.

9) Efreet, Eio, Eeze, Ethumb, and Eldbus

Additional libraries include Efreet, Eio, Eeze, Ethumb, and Eldbus.

Efreet is a library designed to make an application work in accordance with the standards of Freedesktop.org related to icons, desktop files, menus, and so on.

Eio is a library for asynchronous I/O. Ethumb provides functions to generate thumbnail images by adding the frame images.

Eeze is used to manipulate hardware devices through udev. For instance, related to the status of devices, Eeze can be used to determine the status of the devices including whether a CD-Rom has a disk inserted, the temperature of CPU, and the remaining power of the battery. Finally, Eldbus is a wrapper around the dbus library, which is a message bus system. It also implements a set of specifications of dbus to practice IPC.

10) Language Bindings

Basically, EFL supports C language, and is also involved in the language binding projects using Elev8 (JavaScript), Python, Pearl, C++, and Vala.

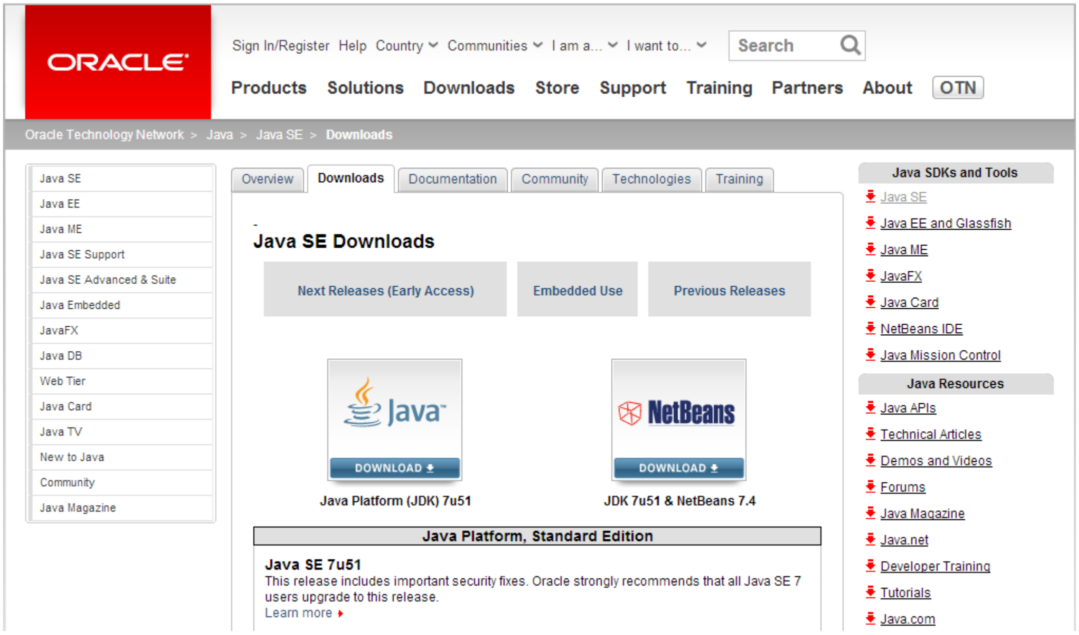
(Reference: EFL Korea Community)

2. Establishing a Tizen Development Environment

A. Download of Java Development Kit (JDK)

Tizen uses Java to support all types of OS in development environments. For that reason, your device needs to have JDK installed. If JDK is not installed, you will see an error message during the installation of Tizen SDK, and the installation will fail to complete. Follow the steps below:

1) Open a web browser, and go to http://www.oracle.com/technetwork/java/javase/downloads.



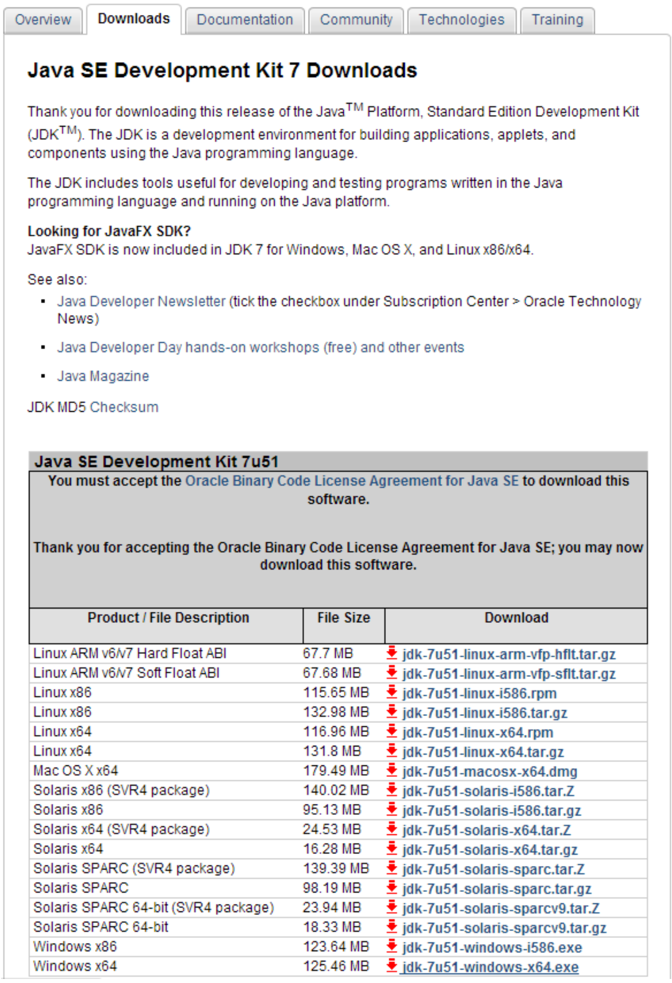
Java SE 7u51

This release includes important security fixes. Oracle strongly recommends that all Java SE 7 users upgrade to this release.

Learn more

2) Press the Java DOWNLOAD (or JDK DOWNLOAD) button. When the screen is changed, check the checkbox of 'Accept License Agreement.'

3) Download Java SE Development Kit xux. Simply select the OS version you are using from the list. If you use a 32-bit OS, click Windows x86. If you use a 64-bit OS, click Windows x64.



Java SE Development Kit 7 Downloads

Thank you for downloading this release of the Java^{TM} Platform, Standard Edition Development Kit (JDK^{TM}). 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

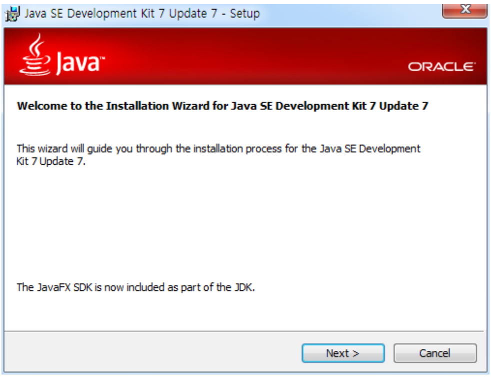
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

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. Installation of Java Development Kit (JDK)

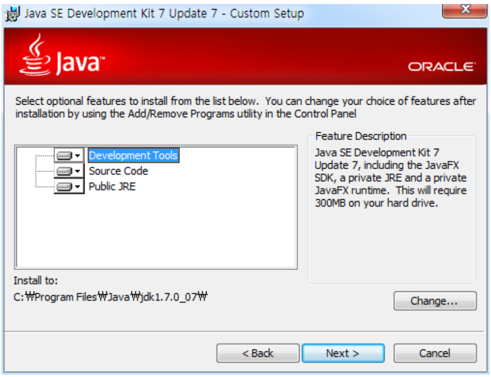
1) Double-click the EXE file downloaded. Then, you will see the installation screen. Click the Next button.


Java SE Development Kit 7 Update 7 - Setup

Welcome to the Installation Wizard for Java SE Development Kit 7 Update 7

This wizard will guide you through the installation process for the Java SE Development Kit 7 Update 7.

The JavaFX SDK is now included as part of the JDK.


Java SE Development Kit 7 Update 7 - Custom Setup

Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel

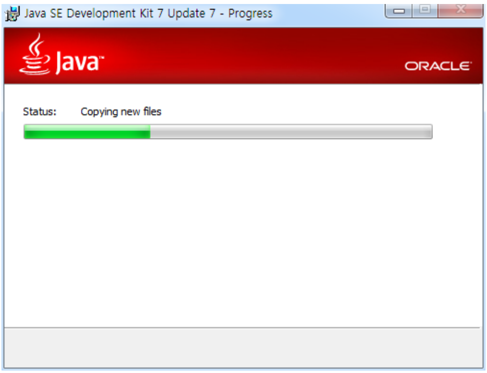
Development Tools
Source Code
Public JRE

Feature Description
Java SE Development Kit 7 Update 7, including the JavaFX SDK, a private JRE and a private JavaFX runtime. This will require 300MB on your hard drive.

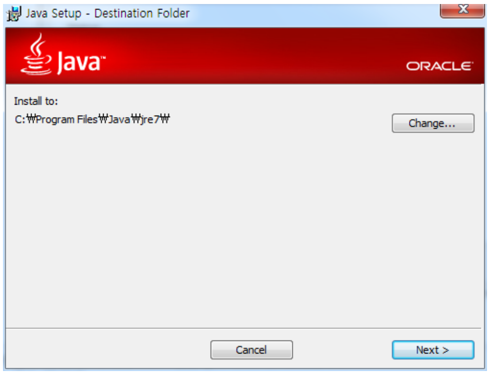
Install to:
C:₩Program Files₩Java₩jdk1.7.0_07₩

2) When the screen is displayed with items to be selected for installation, click the Next button with the default settings as is.

3) Once JDK starts the installation, wait until the installation finishes. When you see the screen asking for the installation path of JRE, click the Next button. If you want to designate the installation path somewhere other than the default folder (e.g. C:/Program Files/Java/jre7/), change the path by clicking Change.

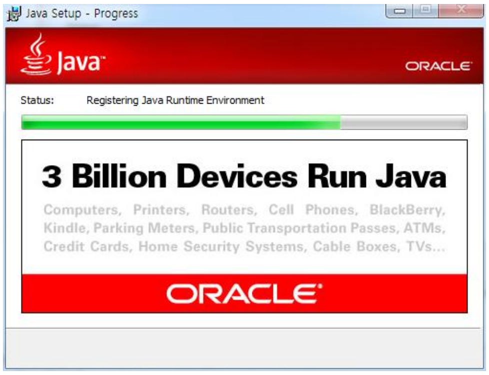

Java SE Development Kit 7 Update 7 - Progress

Status: Copying new files


Java Setup - Destination Folder

Install to:
C:₩Program Files₩Java₩jre7₩

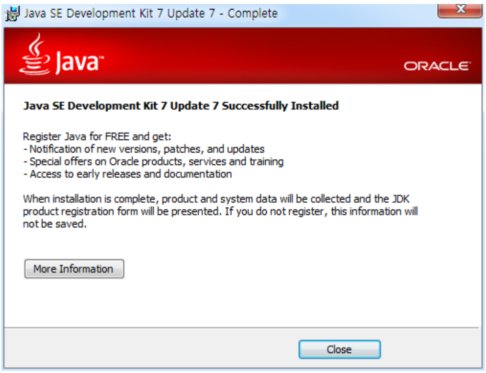
4) Once JRE starts the installation, wait until the installation finishes. When you see the screen indicating that the installation has been completed, click the Close button.



Status: Registering Java Runtime Environment

3 Billion Devices Run Java

Computers, Printers, Routers, Cell Phones, BlackBerry, Kindle, Parking Meters, Public Transportation Passes, ATMs, Credit Cards, Home Security Systems, Cable Boxes, TVs...



Java SE Development Kit 7 Update 7 Successfully Installed

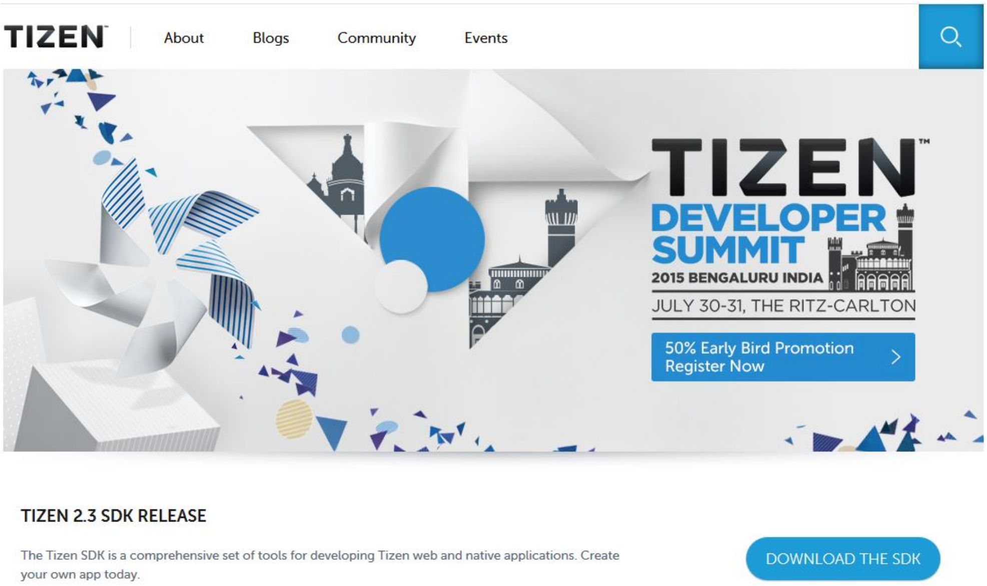
Register Java for FREE and get:
- Notification of new versions, patches, and updates
- Special offers on Oracle products, services and training
- Access to early releases and documentation

When installation is complete, product and system data will be collected and the JDK product registration form will be presented. If you do not register, this information will not be saved.

C. Download of Tizen SDK

This section will introduce you how to build Tizen development environments. First of all, you need to download the SDK installation file.

Go to https://www.tizen.org/.

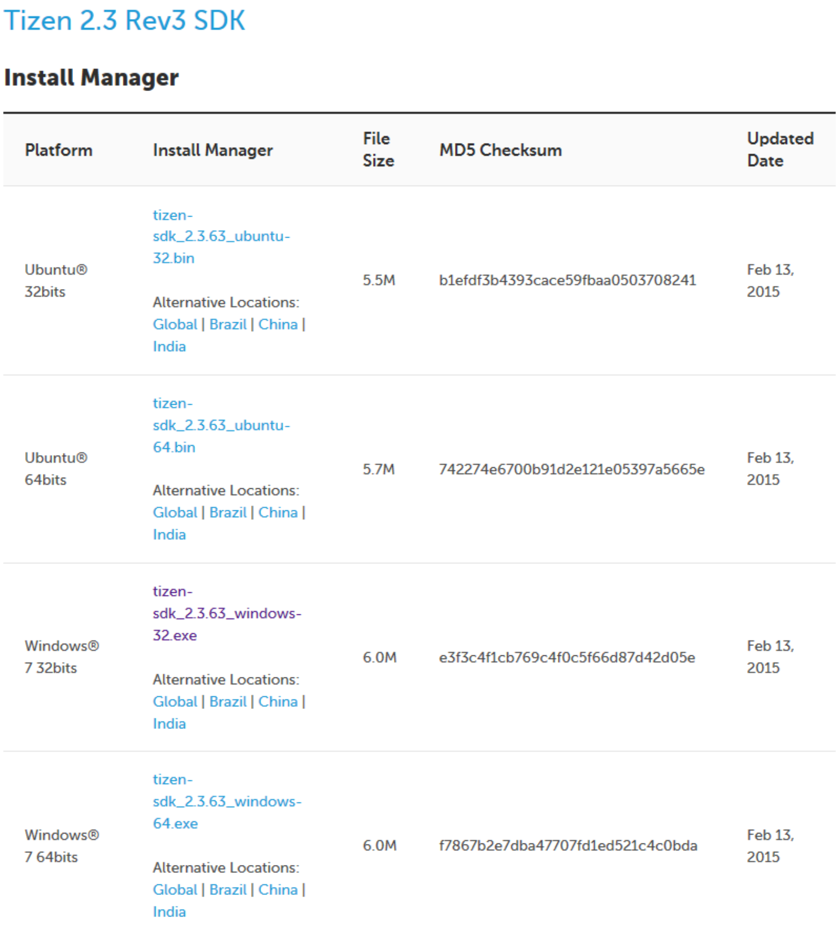


TIZEN 2.3 SDK RELEASE

The Tizen SDK is a comprehensive set of tools for developing Tizen web and native applications. Create your own app today.

Click the DOWNLOAD THE SDK button on the first screen. You will then move to the SDK download page.

If you use a 32-bit version of Windows, select and download Windows XP/7 32bits from Install Manager. If you use a 64-bit version of Windows, select and download Windows XP/7 64bits. Ubuntu 32-/64-bit and Mac OS are also supported.

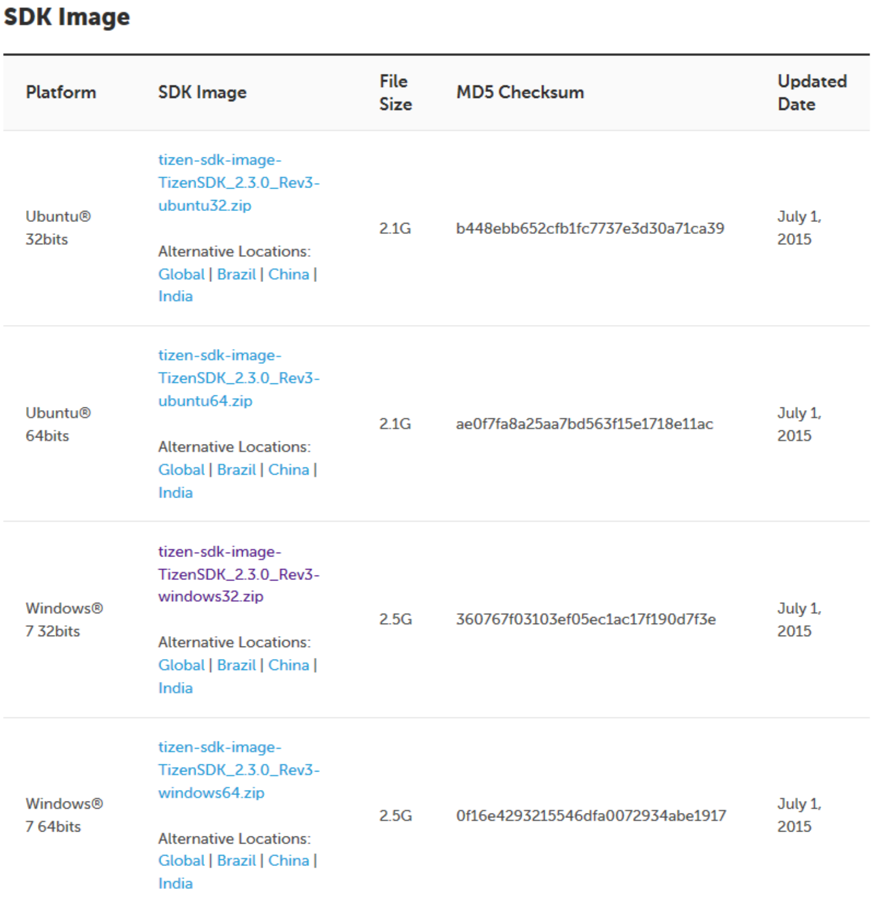


Tizen 2.3 Rev3 SDK

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

Download the Install Manager file. Scroll down and download the same items for SDK image. Once you download Install Manager, the necessary files will be automatically downloaded from the Internet during installation. You may need to download image files in advance to prevent any errors from occurring during installation.



SDK Image

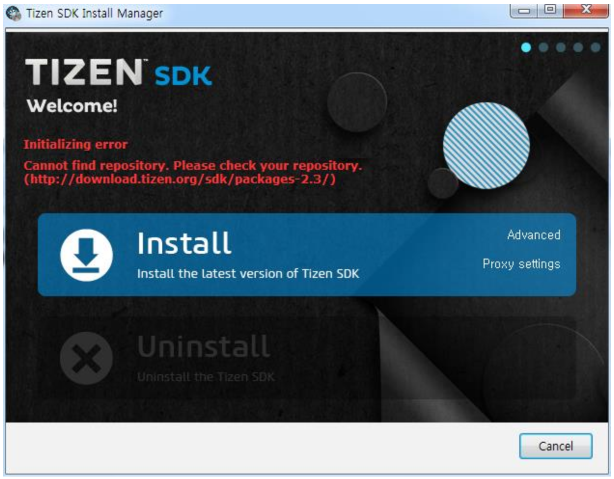
Platform	SDK Image	File Size	MD5 Checksum	Updated Date
Ubuntu® 32bits	tizen-sdk-image-TizenSDK_2.3.0_Rev3-ubuntu32.zip Alternative Locations: Global | Brazil | China | India	2.1G	b448ebb652cfb1fc7737e3d30a71ca39	July 1, 2015
Ubuntu® 64bits	tizen-sdk-image-TizenSDK_2.3.0_Rev3-ubuntu64.zip Alternative 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. Installation of Tizen SDK

This section will inform you how to install Tizen SDK. The instructions are based on SDK version 2.4.0b. However, you can easily apply them to other versions. Furthermore, the simple installation of Tizen SDK enables novices to install it without even referring to the manual.

When the download is completed, double-click the EXE file (tizen_sdk_2.x.xx_window-32.exe) to execute it. If you see a warning window, ignore it and click Yes.

When the installation screen appears, you need to designate an image file. Click the Advanced button on the right.



Tizen SDK Install Manager

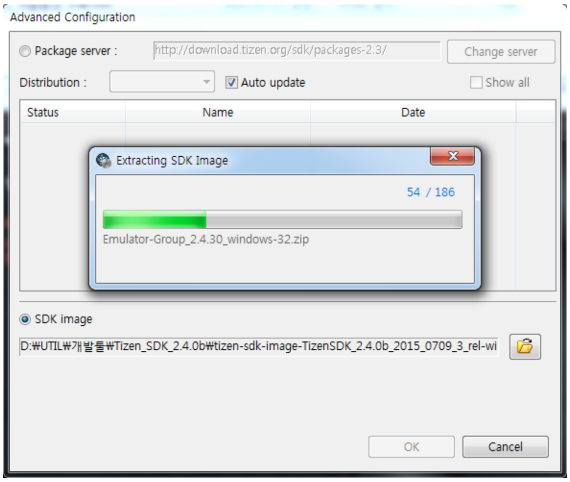
TIZEN SDK

Welcome!

Initializing error

Cannot find repository. Please check your repository.
(http://download.tizen.org/sdk/packages-2.3/)

Check the checkbox next to SDK Image, and click the Open file button on the right. This will open a popup window from where you can open files. Select the image file (tizen-sdk-image-TizenSDK_2.4.x_xxxx_xxxx_x_rel-windows-32.zip) you downloaded from the Tizen developer website. The Zip file will be decompressed. Wait until the decompression is completed.



Advanced Configuration

○ Package server : http://download.tizen.org/sdk/packages-2.3/

Distribution :

☑ Auto update

□ Show all

Status	Name	Date

Extracting SDK Image

54 / 186

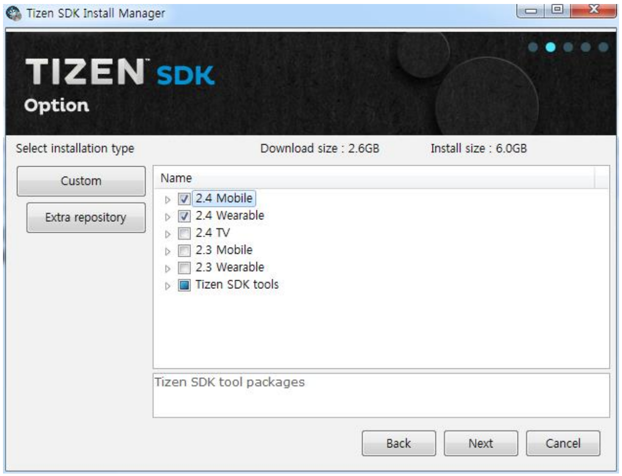
Emulator-Group_2.4.30_windows-32.zip

● SDK image

D:₩UTIL₩개발툴₩Tizen_SDK_2.4.0b₩tizen-sdk-image-TizenSDK_2.4.0b_2015_0709_3_rel-wi

When the decompression is completed, click the OK button to close the window. Click the Install button on the main screen, and move to the next page.

When the Select Install Type screen is displayed, check 2.4 Mobile and 2.4 Wearable, and click the Next button.

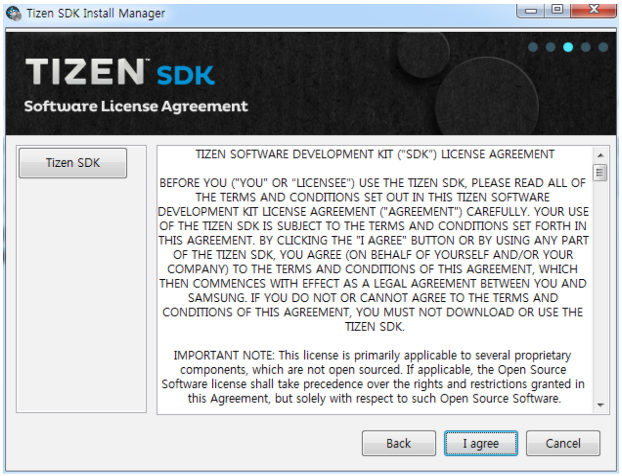


Select installation type   Download size : 2.6GB   Install size : 6.0GB

Name
- ☑ 2.4 Mobile
- ☑ 2.4 Wearable
- □ 2.4 TV
- □ 2.3 Mobile
- □ 2.3 Wearable
- ■ Tizen SDK tools

Tizen SDK tool packages

When the Software License Agreement screen is displayed, click the 'I agree' button.



TIZEN SOFTWARE DEVELOPMENT KIT ("SDK") LICENSE AGREEMENT

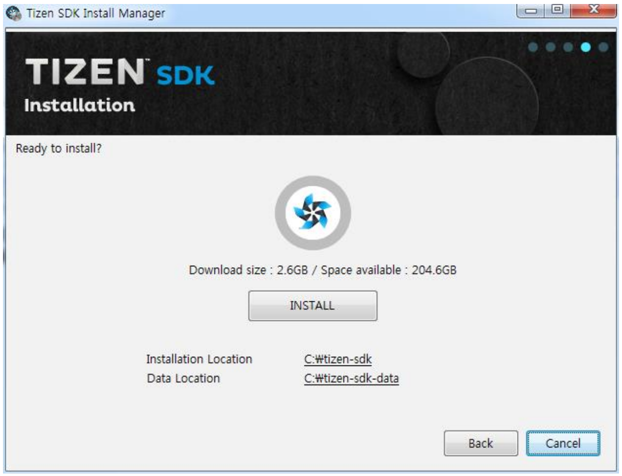
BEFORE YOU ("YOU" OR "LICENSEE") USE THE TIZEN SDK, PLEASE READ ALL OF THE TERMS AND CONDITIONS SET OUT IN THIS TIZEN SOFTWARE DEVELOPMENT KIT LICENSE AGREEMENT ("AGREEMENT") CAREFULLY. YOUR USE OF THE TIZEN SDK IS SUBJECT TO THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. BY CLICKING THE "I AGREE" BUTTON OR BY USING ANY PART OF THE TIZEN SDK, YOU AGREE (ON BEHALF OF YOURSELF AND/OR YOUR COMPANY) TO THE TERMS AND CONDITIONS OF THIS AGREEMENT, WHICH THEN COMMENCES WITH EFFECT AS A LEGAL AGREEMENT BETWEEN YOU AND SAMSUNG. IF YOU DO NOT OR CANNOT AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT, YOU MUST NOT DOWNLOAD OR USE THE TIZEN SDK.

IMPORTANT NOTE: This license is primarily applicable to several proprietary components, which are not open sourced. If applicable, the Open Source Software license shall take precedence over the rights and restrictions granted in this Agreement, but solely with respect to such Open Source Software.

In the Ready to install screen, designate the paths where SDK will be installed and where an SDK data folder will be located, respectively. The SDK default path is C:₩tizen-sdk. If you want to change the path, click the address field and designate your desired folder.

The default data folder path is C:₩tizen-sdk-data. If you prefer a simple folder structure, you can edit the path to the inside of the SDK folder, like C:₩tizen-sdk₩data.

Click the Install button to start the installation.

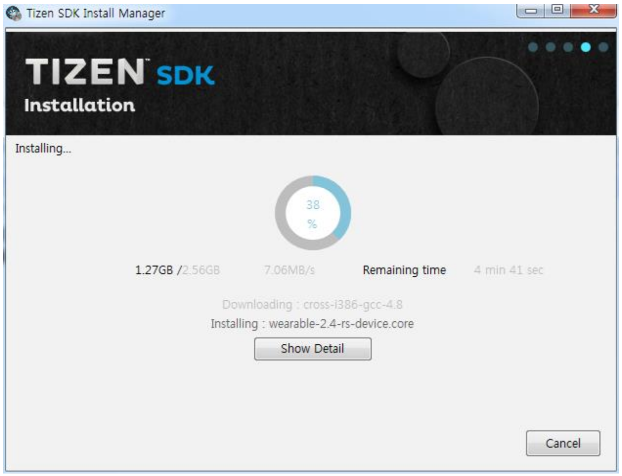


Ready to install?

Download size : 2.6GB / Space available : 204.6GB

Installation Location C:₩tizen-sdk
Data Location C:₩tizen-sdk-data

Once JRE starts installing, wait until the process finishes.



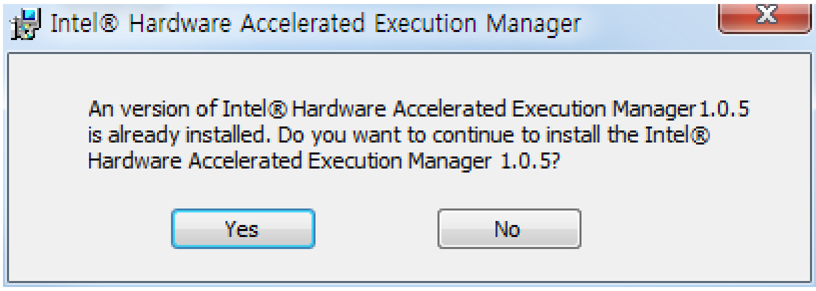
Installing...

38 %

1.27GB /2.56GB 7.06MB/s Remaining time 4 min 41 sec

Downloading : cross-i386-gcc-4.8
Installing : wearable-2.4-rs-device.core

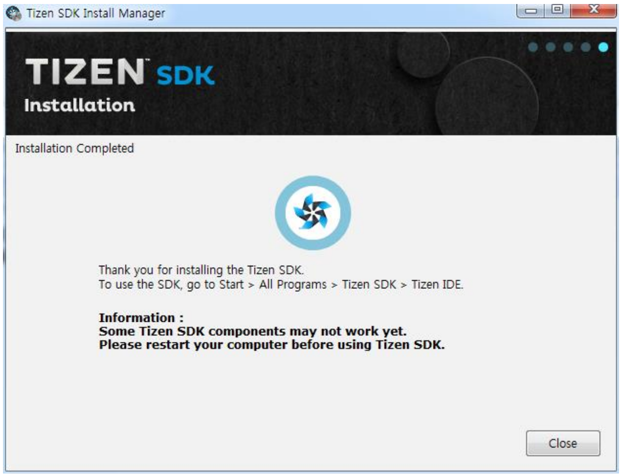
During installation, you may see a popup window informing you that you need to update the hardware acceleration driver. Click Yes and continue to install.



Intel® Hardware Accelerated Execution Manager

An version of Intel® Hardware Accelerated Execution Manager 1.0.5 is already installed. Do you want to continue to install the Intel® Hardware Accelerated Execution Manager 1.0.5?

When the installation is completed, click the Close button.



Installation Completed

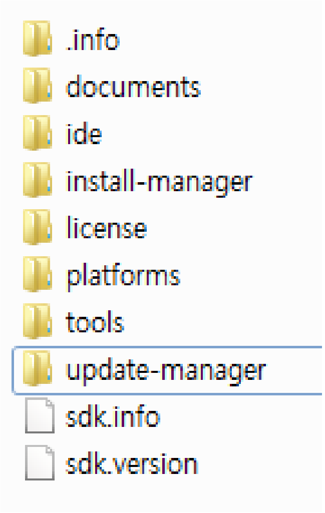
Thank you for installing the Tizen SDK.
To use the SDK, go to Start > All Programs > Tizen SDK > Tizen IDE.

Information :
Some Tizen SDK components may not work yet.
Please restart your computer before using Tizen SDK.

3. Running a Tizen Development Environment

A. Taking a Look at the Installed SDK

This section will introduce you to the kind of data that will be included in the installed SDK. When you open an SDK folder, you will see the folder list as below. If you followed the default settings, the path of your installation folder would be C:/tizen-sdk.



.info
documents
ide
install-manager
license
platforms
tools
update-manager
sdk.info
sdk.version

Let's take a look at some important folders from the list. The '₩documents' folder includes the PDF manuals related to Tizen API and development.

The '₩platforms' folder includes various examples using virtual device (emulator) and Tizen API.

The examples of native apps are saved in the folder '₩platforms₩tizen-x.x₩mobile₩samples₩Template₩Native.'

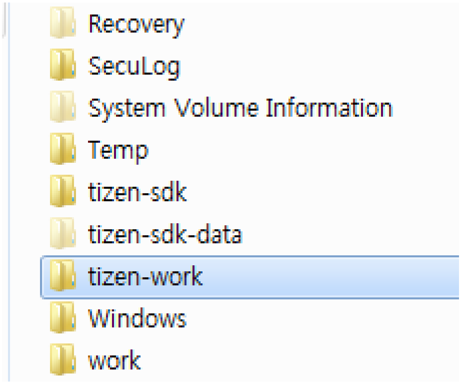
The examples of web apps are saved in the folder '₩tizen-sdk₩platforms₩tizen-x.x₩mobile₩samples₩Template₩Web.' As the whole source project is included, you can check the results by directly loading, building, and executing them in an IDE. The following folder has the most useful resources to developers.

The '₩ide' folder includes development environment tools which developers can use to create Tizen applications. This folder has an IDE where you can create a new project, enter source code, create widgets on the screen in UI editor and designate their properties, and build the project and execute it with emulator and as target. App developers do not need to use the contents of this folder.

The '₩tools' folder includes tools that you can use to create a certification, and build and execute source projects at Linux.

F. Execution of IDE-Development Environment Tools

Now, let's see how to run the tools in the actual development environments. First, we need to create a work folder before executing the IDE. In this example, a new folder is created in the C: drive root folder, and named 'tizen-work.' You can choose any paths and names for your folder.



Recovery
SecuLog
System Volume Information
Temp
tizen-sdk
tizen-sdk-data
tizen-work
Windows
work

The next step is to execute Tizen IDE in the installed SDK. You can simply follow the path [All programs > Tizen SDK > Tizen IDE] from the Start menu of Windows.

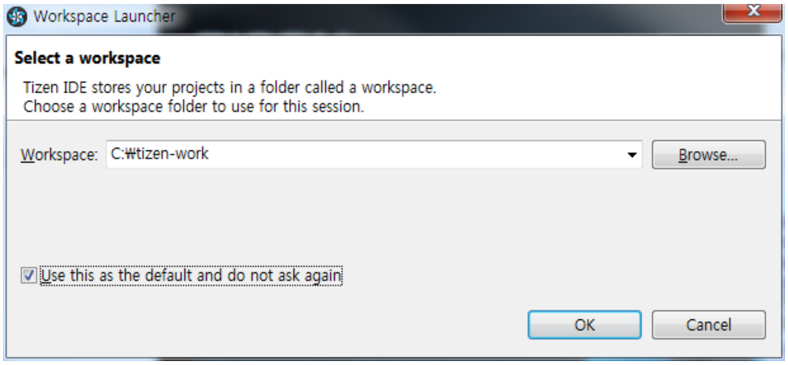
[Tip!]

Tizen uses Eclipse as its IDE (development tool). It means that Tizen shares the same development tool with Java and Android.

As the native apps of Tizen use C language, not Java, it is possible for developers to use Visual Studio. In this case, however, developers may need to purchase the license of the program for individual use. To avoid such inconvenience, Eclipse is provided as a basic specification. Developers can create their applications for free using this program.

If you are familiar with Visual Studio and other development environments, you will quickly become used to the environments of Eclipse.

When you initially execute IDE, you will see the popup window as shown below. This is the screen where you can designate a work folder in which to save source projects.



Select a workspace

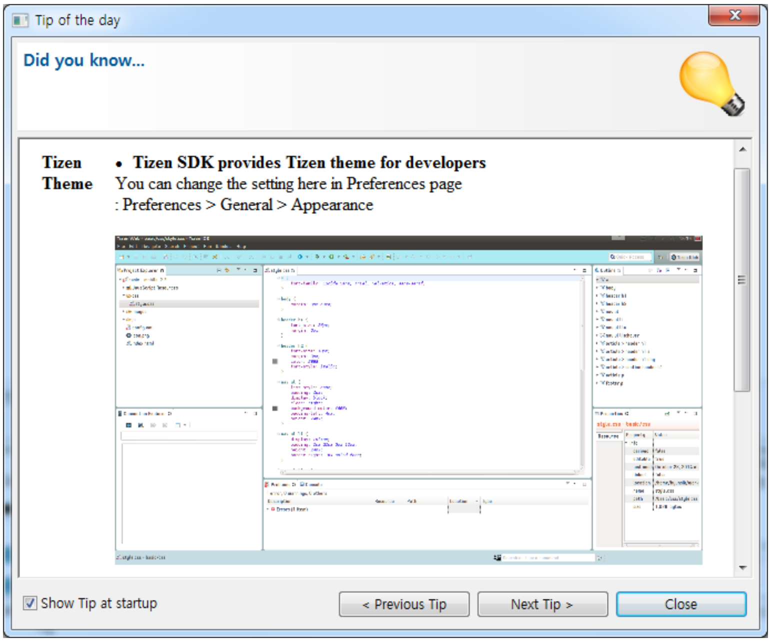
Tizen IDE stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:₩tizen-work

☑ Use this as the default and do not ask again

Click the Browse button, and designate a folder which will be used as a work space. In this case, C:₩tizen-work has been used as the work folder. If you have designated a folder, select the checkbox next to 'Use this as the default and do not ask again,' and click the OK button to go to the next step. It is recommended to check this message, otherwise you will see this window every time you execute IDE.

When IDE is executed, you will see a screen as shown below. This is the screen introducing tips for use. Click the × symbol in the top right corner of the window to close it.

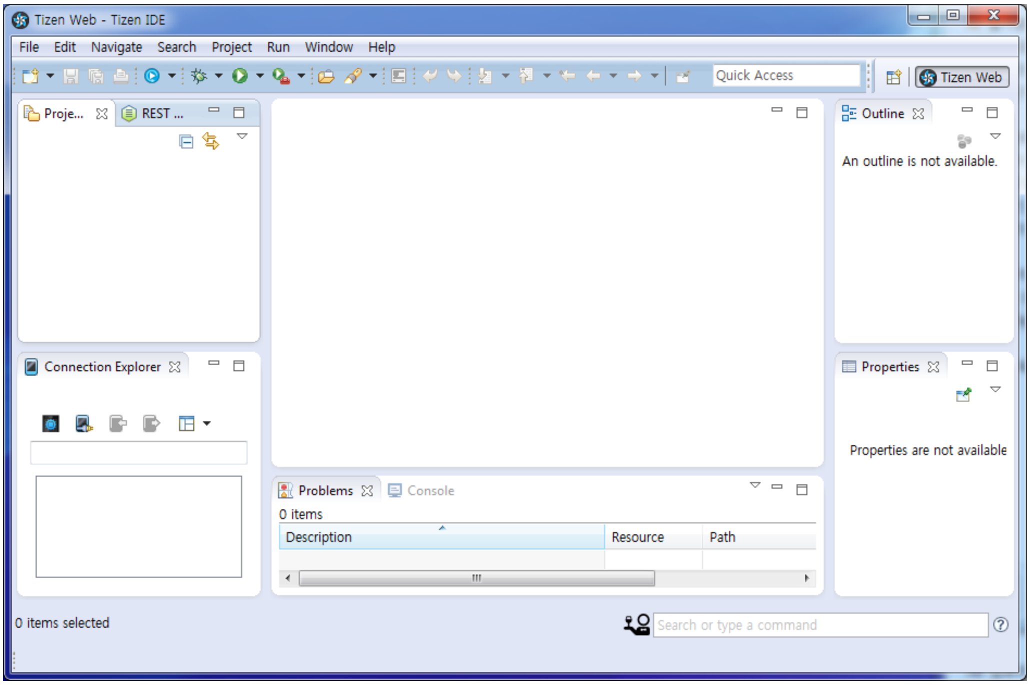


Tizen Theme

- **Tizen SDK provides Tizen theme for developers**
  You can change the setting here in Preferences page
  : Preferences > General > Appearance

☑ Show Tip at startup

When the screen is closed, you will a screen as shown below. If you have experience of developing with Java or Android, you may be familiar with this screen.



An outline is not available.

Properties are not available

0 items

Description	Resource	Path

0 items selected

This section will introduce you briefly to each field. The main menu and tool bar are at the top. Shortcut keys and menus are available if you do not wish to use the tool bar.

On the right side of the tool bar, you will see a square marked 'Tizen Web.' You can use this to convert between web apps and native apps, and between debug mode and execution mode. When you execute an application in debug mode, another button will be added. As a result, using two toggle-type buttons, you can move between debug and execution modes as you wish.

Below the left side of the tool bar, you will see a field called 'Project Explorer.' This field shows you the list of all the projects saved in the work folder and fields in the sub files (e.g. Source and resource files). When you select a source file of UI screen information file in this field, the detailed information is displayed in the middle of the screen. You can also modify the properties of a project or execute it from the shortcut menu.

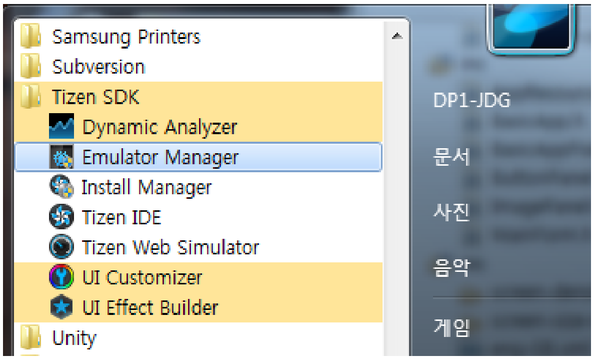
The field ‘Connection Explorer’ displays a list of devices available to install developed apps. Emulator will be displayed when it is executed. When you connect a phone with a cable, the phone will be added.

Now, you will see several tabs at the bottom of the screen in the center. The most important of these are as follows. 'Problems' displays log messages, warnings, or error messages that occur while you are building a project. When any build error occurs, you can correct it referring to the messages displayed here.

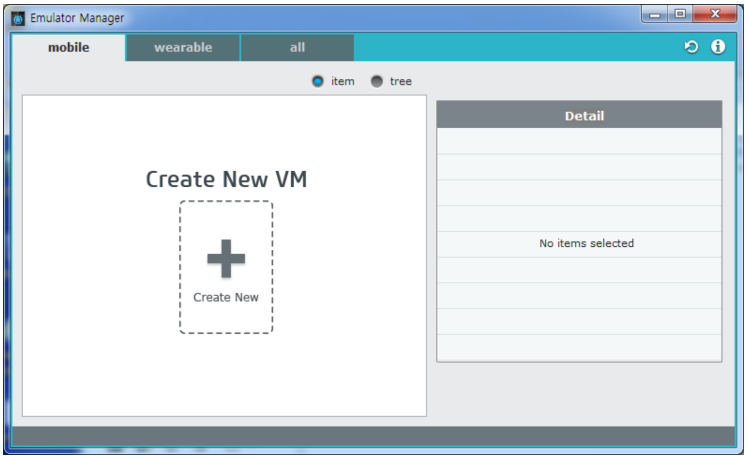
'Log' displays log messages occurring while you are executing an application. You can check the status of progress while executing an application by applying API functions such as AppLog() or AppLogDebugTag() for the source code.

G. Running an Emulator

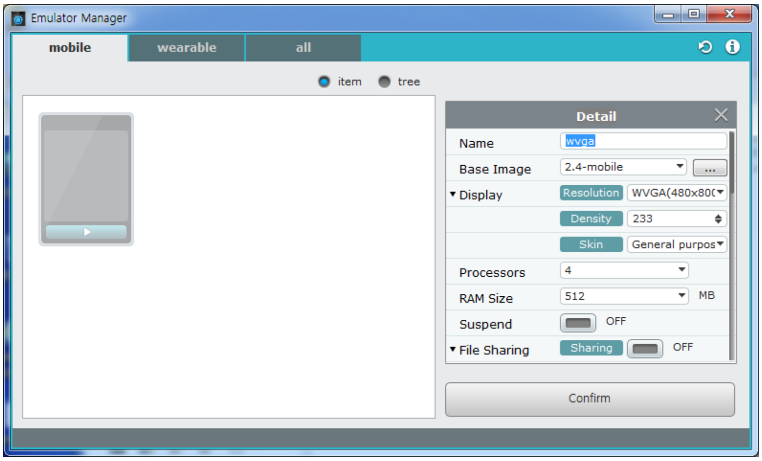
In this section, you will learn how to create and run an emulator. Click [Windows Start button > All Programs > Tizen SDK > Emulator Manager]. If you see a warning window, ignore it and click the Yes button.



If you initially run the Emulator Manager, you need to create an emulator. Click the '+' sign below 'Create New VM' on the left.

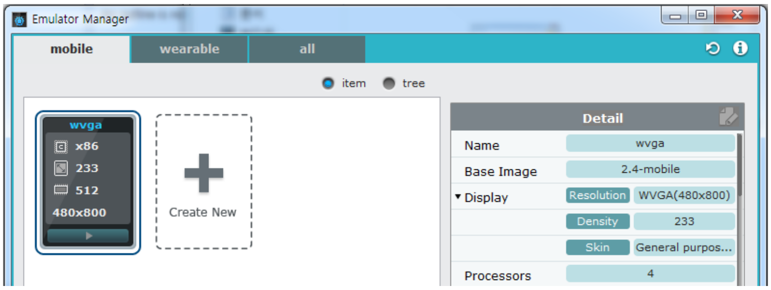


Then, you will see a screen where you can designate the options of the emulator on the right. Designate wvga as the name property. You can leave the rest as the default properties. Click the Confirm button to create a new emulator.



Name: wvga
Base Image: 2.4-mobile
Display: Resolution WVGA(480x80(
Density: 233
Skin: General purpos
Processors: 4
RAM Size: 512 MB
Suspend: OFF
File Sharing: Sharing OFF

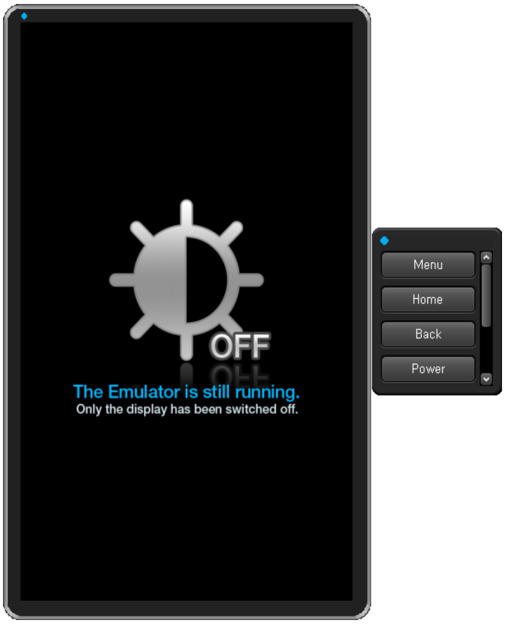
Now, you will have a new emulator named wvga on the left side of the screen. Click the arrow button below the new emulator icon to run the emulator. If you want to change the options of the emulator, click the Reset button.



wvga
x86
233
512
480x800

Name: wvga
Base Image: 2.4-mobile
Display: Resolution WVGA(480x800)
Density: 233
Skin: General purpos...
Processors: 4

When the 'Windows Security Warning' popup window appears, click the Unblock button to continue the process. You may see a little square popup window on the right side of the emulator. This window includes hardware keys. It has Menu, Home, Back, Power, Volume+, and Volume- from the top. It has mostly similar functions to the hardware keys supported by Android.

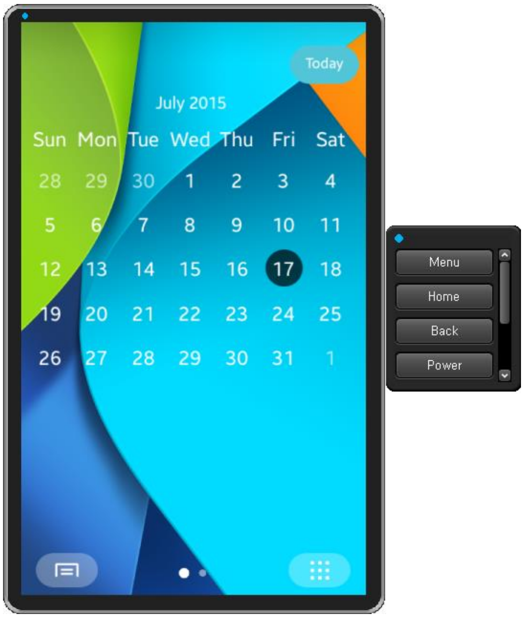


OFF

The Emulator is still running.

Only the display has been switched off.

When it is converted to sleep mode, simply click the Home button to turn on the screen again. Then, drag the screen to unlock.



July 2015

Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1

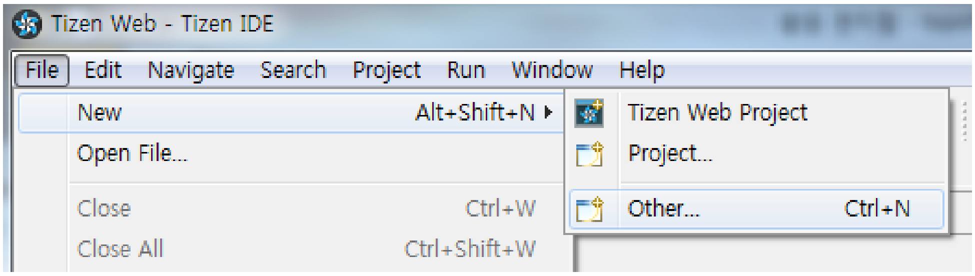
4. Executing Examples

SDK provides you with guide manuals, but examples for actual practice may be more helpful. Such examples will be the resources you would refer to the most frequently while you are working on your future projects. You will be shown how to add the examples to an IDE and run them using an emulator. Let's take a look at some important examples first.

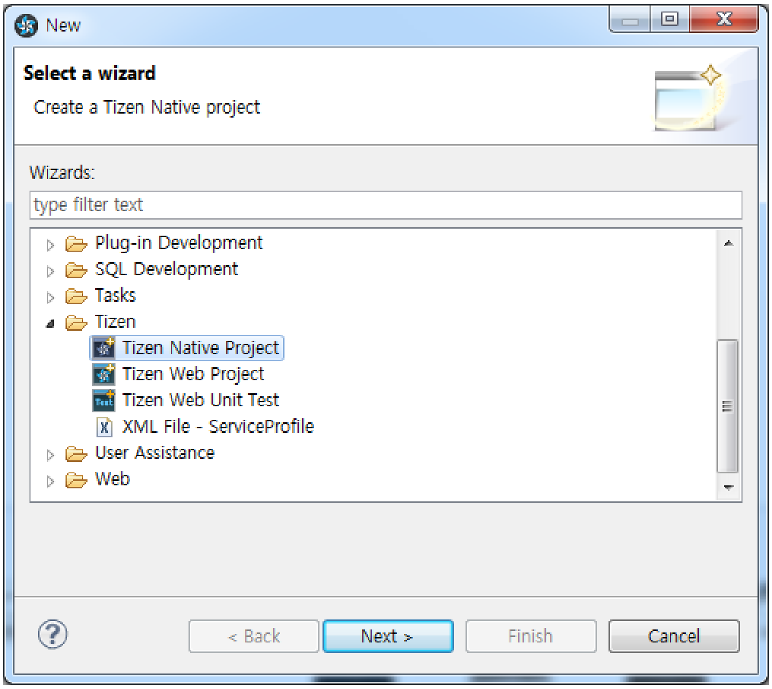
A. Adding a Sample to an IDE

You are now ready to develop Tizen applications. Let's practice the process using an example.

Select [File > New > Other ...] from the main menu.



When a new popup window appears, you will see a list in tree structure. Open the Tizen folder, and select 'Tizen Native Project' from among the items. Then, click the Next button.



New

Select a wizard

Create a Tizen Native project

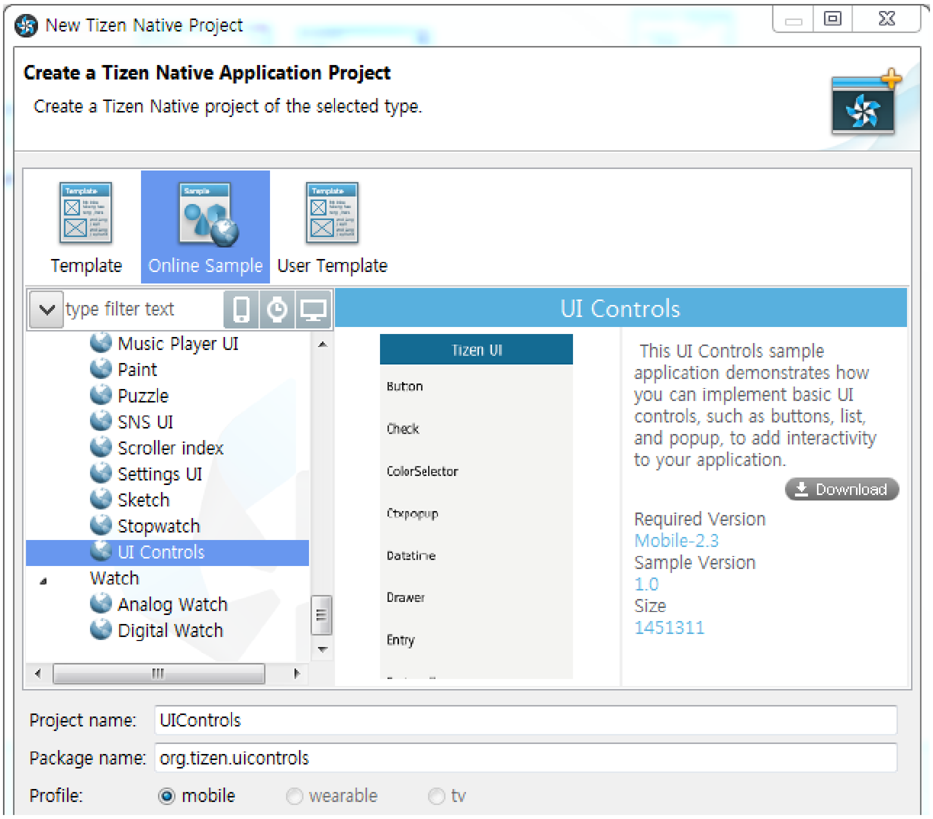
Wizards:

type filter text

- Plug-in Development
- SQL Development
- Tasks
- Tizen
  - Tizen Native Project
  - Tizen Web Project
  - Tizen Web Unit Test
  - XML File - ServiceProfile
- User Assistance
- Web

When the ‘Create a Tizen Native Application Project’ popup window appears, click the 'Online Sample' button on the right side of the Template button. Now, you will see the list of examples for native apps. When you select [UI > UI Controls] from the list on the left, you will see the screen capture image on the right.

At this stage, you need to name the project. This process imports one of the existing examples, so you can just give it the same name as the original example. Type 'UIControls' in the field 'Project name,' and click Finish.



New Tizen Native Project

Create a Tizen Native Application Project

Create a Tizen Native project of the selected type.

This UI Controls sample application demonstrates how you can implement basic UI controls, such as buttons, list, and popup, to add interactivity to your application.

Required Version: Mobile-2.3

Sample Version: 1.0

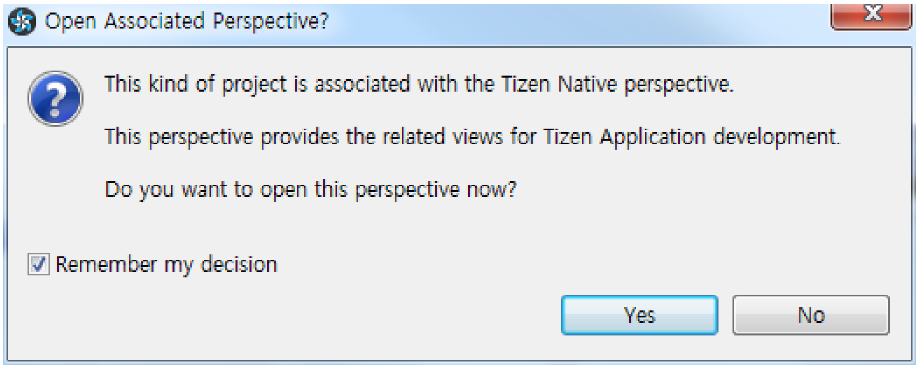
Size: 1451311

Project name: UIControls

Package name: org.tizen.uicontrols

Profile: ● mobile ○ wearable ○ tv

When you see a popup window asking you if you want to open ‘Perspective,’ click Yes after checking the 'Remember my decision' checkbox.



Open Associated Perspective?

This kind of project is associated with the Tizen Native perspective.

This perspective provides the related views for Tizen Application development.

Do you want to open this perspective now?

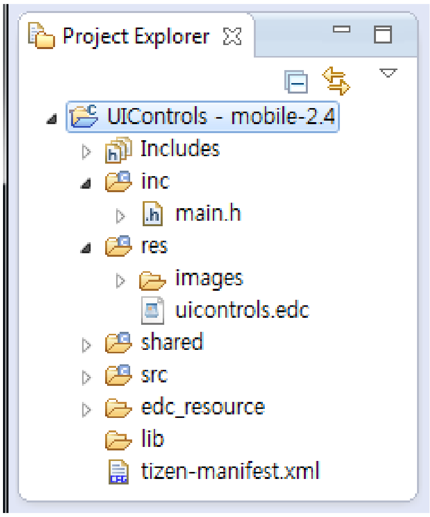
☑ Remember my decision

The popup window will close, and the example will be added to 'Project Explorer.'

This process is not to import files from the '/platforms' folder of SDK, but to import the files you have copied to your work folder. When you open the work folder (C:/tizen-work) that you designated, you can see the 'UIControls' folder is copied.

UIControls is an example that shows you all kinds of widgets supported by Tizen and how to use Container.

When you click the '+' sign on the left side of Basic App which has been added to 'Project Explorer,' you will see a list in tree structure and several folders. Now, let's see what kinds of files are in each folder.



The '/src' folder includes source files (.c). The '/inc' folder includes header files (.h). When a basic project is created, the main screen of it is automatically generated.

You can work in source files for the main screen when a new screen is added. However, it is recommended that you create new source files because it makes the management and reuse for other projects easier.

In the case of creating a library, it would be better for you to work with a new header file.

The images for app icons are saved in the '/shared/res' folder. From your smart phone, you can run an application by touching a certain icon from the icon list. The app icon images are used in this case.

Using the 'tizen-manifest.xml' file, you can designate various information related to an app, including application ID, the version of SDK, the image for the app icon, and Label text.

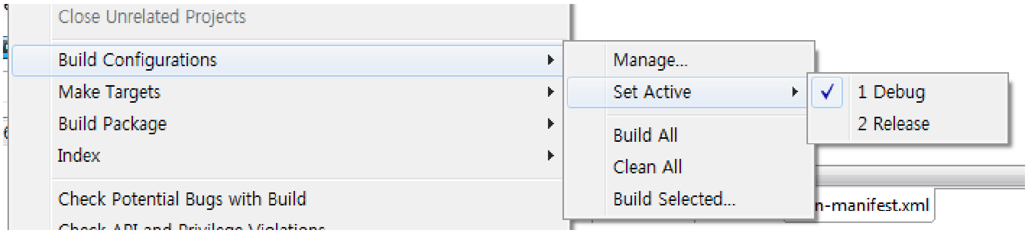
B. Building a Sample

To run a source project using the emulator, you need to build the EXE file first. The native app development of Tizen uses C language, so it is not built automatically like with Java or Android. Also, you need to choose in which mode you wish to build the source project.

Build mode has two types – debug and release modes. In debug mode, you can display desired information in the log window as log messages during execution. This mode can also perform debugging.

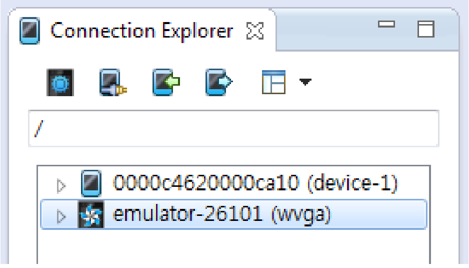
In release mode, you can only run the program, but cannot check log messages. For that reason, you should use debug mode while developing an application, and use the package built in release mode when you finally register it at the seller website.

The default setting is debug mode. If you want to change to build mode, right-click the source project at 'Project Explorer,' and choose items you want from [Build Configurations > Set Active].

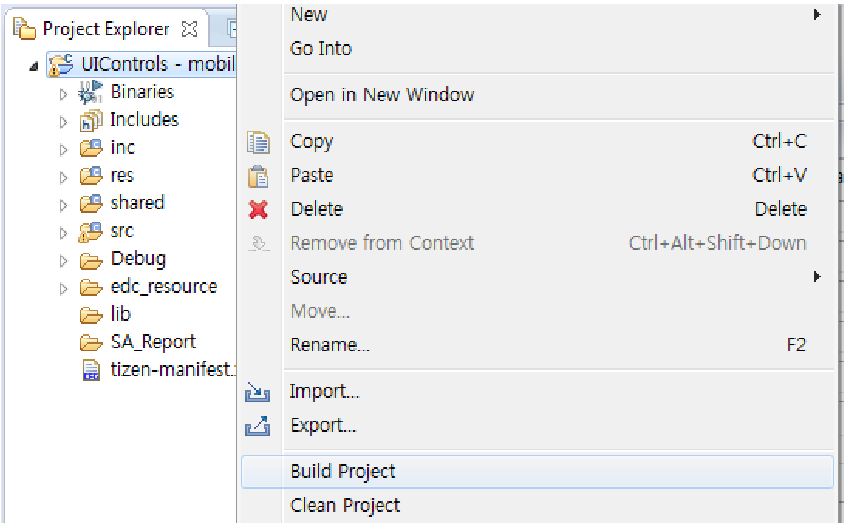


At the left bottom of Eclipse, there is the 'Connection Explorer' window. You will see the emulator you have just created has been registered. If you connect a terminal using a USB cable, you will also see the additional device.

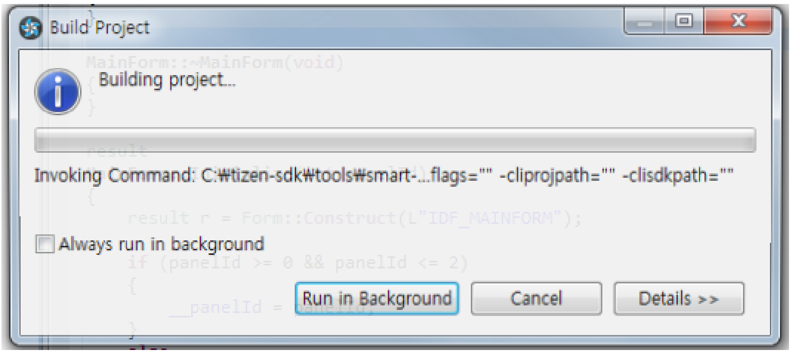
To run the emulator, select the emulator from the 'Connection Explorer' window, and build your project. If you want to test at a terminal, you can select the terminal and build your project.



If you successfully designated build mode, you are ready to build your project. Select [Build Project] from the project shortcut menu to start building.

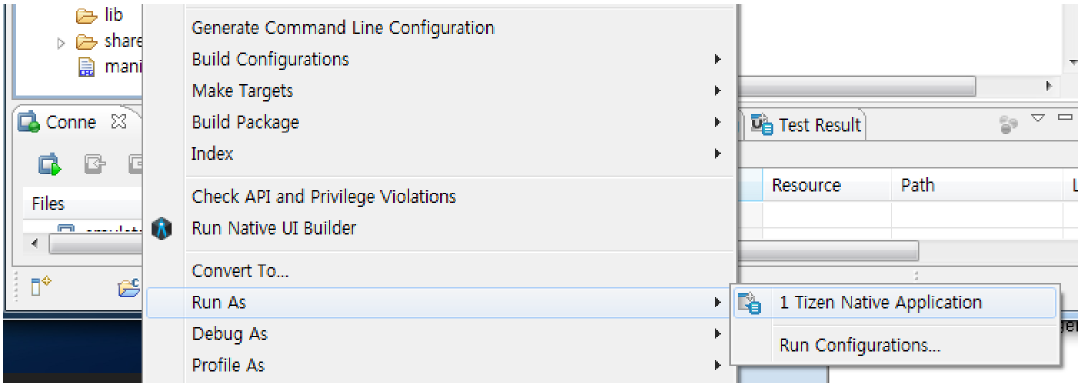


You will see a new popup window displaying the progress rate of the build. Wait until this window automatically disappears. If the popup window disappears, and if no errors are displayed on the 'Problems' tab, this means that your project has been successfully built. If any error messages are displayed, correct the issues and build it again.



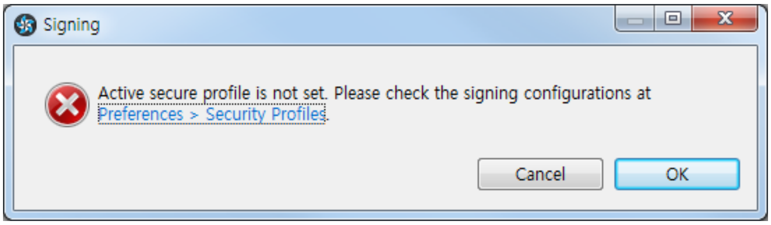
Invoking Command: C:₩tizen-sdk₩tools₩smart-...flags="" -cliprojpath="" -clisdkpath=""

Once the project is built, it is ready to be run. Select [Run As > 1 Tizen Native Application] from the shortcut menu of the source project.



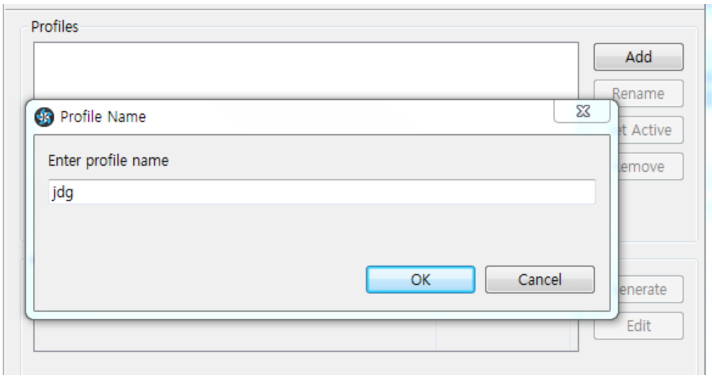
C. Creating a Certificate

At this stage, you may see a warning window indicating that Secure Profile does not exist. From Tizen SDK 2.1, you should create a certificate to install an app in an emulator or terminal by using the following procedure.



Active secure profile is not set. Please check the signing configurations at Preferences > Security Profiles.

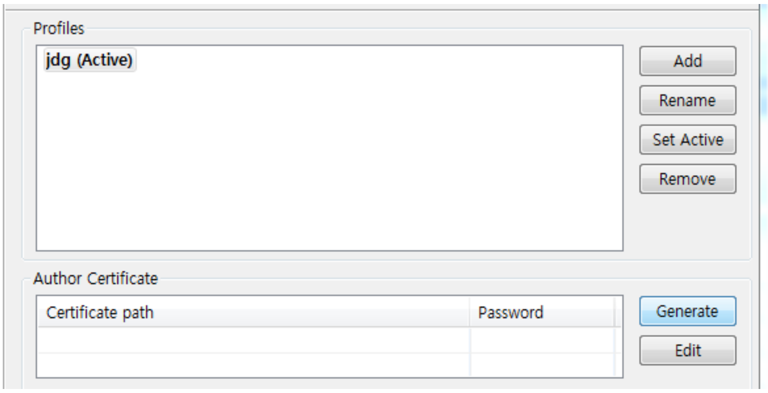
Click the link at Preferences > Security Profiles from the popup window. A new popup window will then be displayed. Click the Add button on the right. When the 'Profile Name' popup window appears, enter the name of your profile, for example your name or the name of your company. Click the OK button to close the popup window.



Profile Name

Enter profile name: jdg

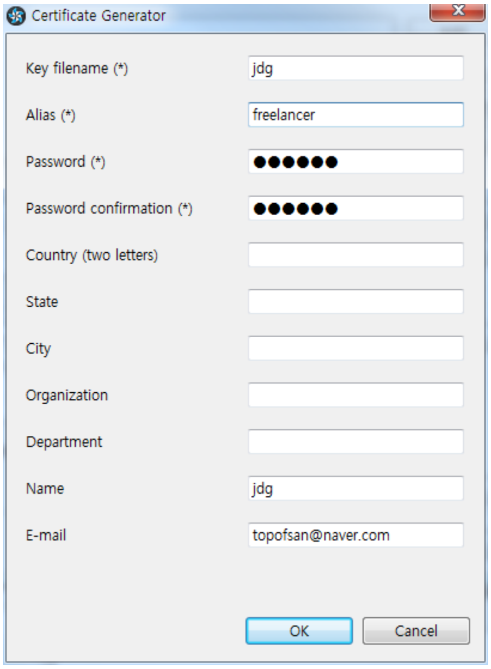
A new item is then added to Profiles. Now, it is time to enter detailed information. Click the Generate button on the right side of Author Certification.



Profiles: jdg (Active)

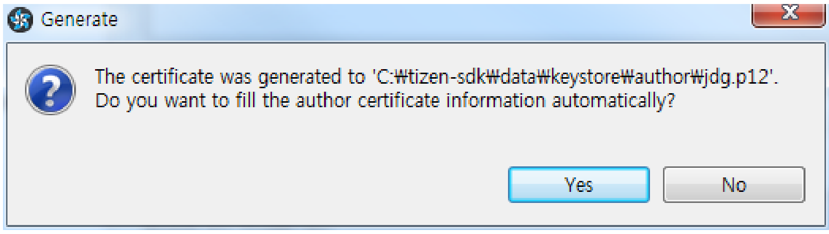
Certificate path	Password

When the 'Certificate Generator' popup window appears, enter the name of the file as the Key filename, the name of affiliated group as the Alias, and a password in the password and password confirmation fields. You can use the name of Profiles for Key filename and Alias. Now, you have entered all information required for use. Once you have finished entering the information, click the OK button to close the popup window.


Certificate Generator

Key filename (*)	jdg
Alias (*)	freelancer
Password (*)	●●●●●●
Password confirmation (*)	●●●●●●
Country (two letters)	
State	
City	
Organization	
Department	
Name	jdg
E-mail	topofsan@naver.com

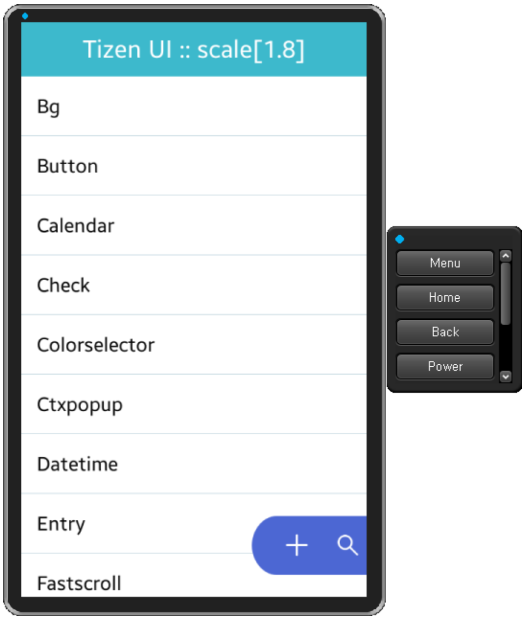
A popup window asking if you wish to use this certificate will appear automatically whenever you open this project. Simply click the Yes button.


Generate

The certificate was generated to 'C:₩tizen-sdk₩data₩keystore₩author₩jdg.p12'.
Do you want to fill the author certificate information automatically?

Now, click the Apply and OK buttons in order, then the popup window will disappear. When you click the OK button in the ‘Signing’ popup window, the source project will be installed on the emulator. If you need to edit the certificate later, select [Windows > Preferences] from the main menu of Eclipse, and select [Tizen SDK > Security Profiles] from the list in the tree structure on the left when the 'Preferences' popup window appears.

When the app is successfully installed on the emulator, UIControls will be run. This example comprehensively shows how to use widgets and containers supported by UIControls Tizen.

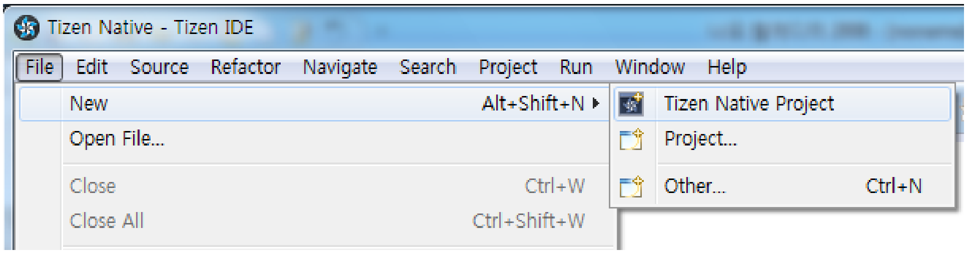


5. Creating a BasicUI example

In this section, we will make a new source project using the BasicUI method, and output text 'Hello World.'

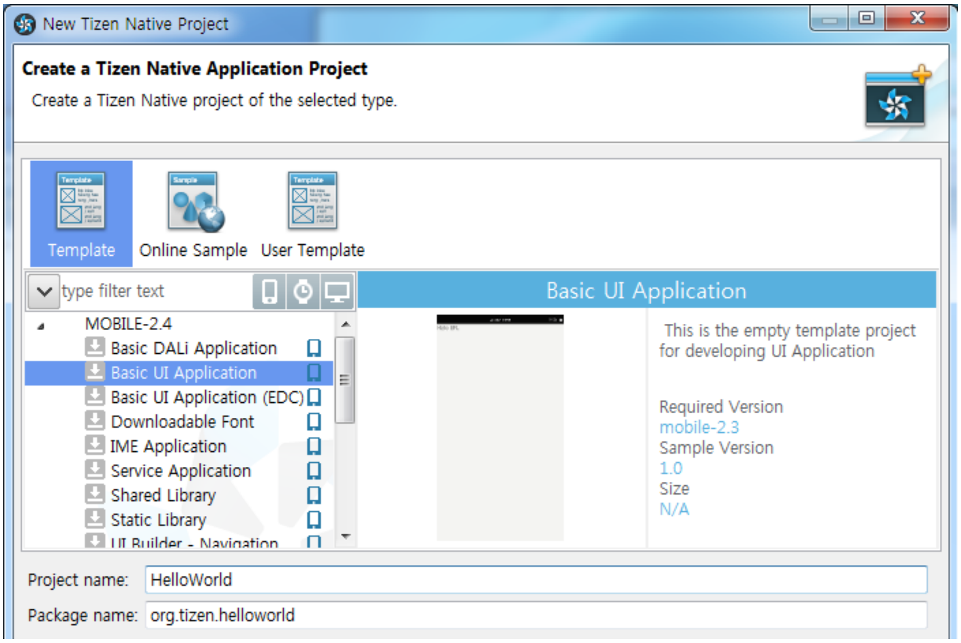
1) Creating a Source Project

The first step is to create a new source project. Select [File > New > Tizen Native Project] from the main menu of Eclipse.



When a popup window for creating a source project appears, select [Template > MOBILE-2.x > Basic UI Application]. For all examples introduced here, you can follow this method.

Enter 'HelloWorld' in the Project name field.



New Tizen Native Project

Create a Tizen Native Application Project

Create a Tizen Native project of the selected type.

This is the empty template project for developing UI Application

Required Version: mobile-2.3

Sample Version: 1.0

Size: N/A

Project name: HelloWorld

Package name: org.tizen.helloworld

The field Package name will be automatically filled. If you're developing an app which will be registered with app stores, you can use the website address for the Package name. For instance, if your domain name is 'www.abc.com,' you can enter 'com.abc.helloworld' as the Package name.

Now, click the Finish button to create a new source project.

2) Components of Source Project

Let's see some basic components of a source project. The 'inc' folder has libraries.

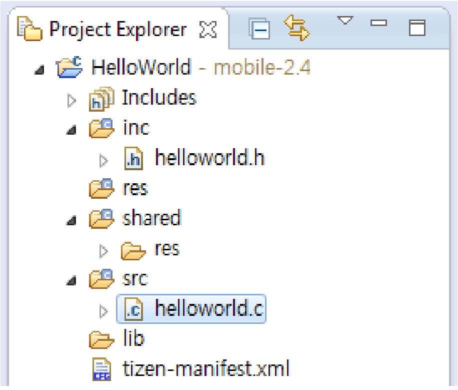
This folder includes header files in C language (.h). You can mostly use this folder to define libraries, function headers, or global variables.

The 'res' folder is usually used to save resource files including image or audio files.

The 'src' folder includes the source files in C language (.c). This folder is mainly used to define the features of functions. Most tasks are performed here.

The 'shared' folder includes the app icon images. When you distribute your apps to app stores, you can save your app icons here. For Tizen store, you should use app icons with a round shape.

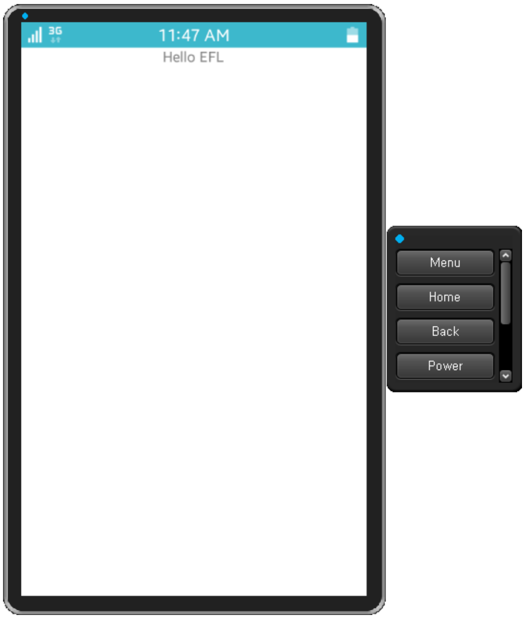
The 'tizen-manifest.xml' file includes various app information (e.g. the name and version of app) and user's rights (privilege). It is basically the same as AndroidManifest.xml in Android.



3) Running a Basic Source Project

In order to run the source project, first right-click the HelloWorld project. Then, select [Build Project] from the shortcut menu. When the project is successfully built, right-click the project again, and select [Run As > 1 Tizen Native Application] from the shortcut menu.

When the example runs on the emulator, you will see the indicator at the top, and the app screen below the indicator. Then, you will see the text, 'Hello EFL.' This text is displayed using Label Widget.



4) Basic Source Code

Now, let’s change the text, Hello EFL, displayed in Label Widget. To do this, we need to edit the source file. Open the ‘src’ folder, and double-click the 'helloworld.c' file. You will now see the content of the file as shown below on the main screen of Eclipse.

```
#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);
}
```

The command #include “helloworld.h” refers to ‘helloworld.h’ in the ‘inc' folder. In general, header files are declared with library include, global variables, and function headers, while source files are defined with the content of the function.

‘appdata_s’ is a structure to save data used in an app.

‘win_delete_request_cb()’ is an event function which is run when the request for app removal occurs. This function is not directly called.

‘win_back_cb()’ is an even function which is run when the Back button is clicked. This function is not directly called.

A little further down, you will see another function, 'create_base_gui().' This function creates windows consisting of a screen, various containers, and a widget.

```
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_request_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_HINT_EXPAN
D);
    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);
}
```

The following are explanations of some lines of the major code.

elm_win_util_standard_add() is an API that creates the Window object. The window is the topmost object in the screen layout. An app has a single window. You can place a widget in a window. However, the more common method is to add a container and then add a widget on top of the container.

elm_win_wm_rotation_available_rotations_set() is an API that specifies the orientation of the screen. If you pass the four angles, 0, 90, 180, and 270, as an array, all screen orientations are supported.

evas_object_smart_callback_add() is an API that specifies an event callback function for a smart object such as a widget or container. For the first parameter, pass the object where the event occurs; for the second parameter, the type of the event; for the third parameter, the name of the callback function; and for the fourth parameter, user data. If the type of the event is "delete,request", it means that the object shall be deleted.

eext_object_event_callback_add() is an API that specifies an event callback function for an object. You can use this API for both smart objects and general objects. For the first parameter, pass the object where the event occurs; for the second parameter, the type of the event; for the third parameter, the name of the callback function; and for the fourth parameter, user data. EEXT_CALLBACK_BACK indicates a Back button click event.

elm_conformant_add() is an API that creates a Conformant. A Conformant changes the size of the window when a new element (for example, a keypad) is added to the screen. An app must have no more than a single Conformant. It is also possible for an app to not have a Conformant.
A Conformant is necessary to display an indicator (status bar) at the top of the screen. It is also possible not to display an indicator when a Conformant exists.

elm_win_indicator_mode_set() is an API that specifies whether to display an indicator.

elm_win_indicator_opacity_set() is an API that specifies the opacity of the indicator.

evas_object_size_hint_weight_set() is an API that specifies the rough size of an object. The following are the parameters listed in order: the object, horizontal size hint, and vertical size hint. EVAS_HINT_EXPAND means specifying as large a size as the space allows.

elm_win_resize_object_add() is an API that resizes the Window object while adding a different object to it.

evas_object_show() is an API that displays an object. When an object is created, the default value of the object is Hide. You can use this function for all objects without exception.

elm_label_add() is an API that creates a Label widget. A Label widget lets you display text as well as change text properties such as the font size and color by using HTML tags.

5) Changing the Text in a Label

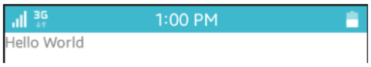
Now, we are going to change the text 'Hello EFL' displayed on the screen. Change the function ‘elm_object_text_set()’ as shown below.

```
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() is an API that changes the caption text of a widget. You can use this function for Button and Entry widgets as well as Label widgets. You can also specify text properties by using HTML tags when creating text.

Let's run the example again. When you run an example for the second time, click [Run > Run] in the main menu or press the 'Ctrl + F11' hotkey.

The example runs again, and the text on the screen changes to 'Hello World.'



Hello World

6) Changing the Position of a Label Using Absolute Coordinates

Labels are placed on the left side of the screen. The evas_object_size_hint_weight_set() function specifies the relative size of an object because the function's both horizontal and vertical options are specified as 'EVAS_HINT_EXPAND.' For the first parameter, enter the object whose properties you want to specify; for the second parameter, enter a horizontal size hint; and for the third parameter, a vertical size hint.

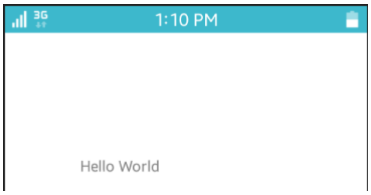
Now, let's change the position of the Label by specifying its absolute coordinates. Change the code as shown below.

```
    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() is a function that specifies the size of an object as an absolute value. Enter parameter values in the following order: the object whose properties you want to specify, the position of the X coordinate, and the position of the Y coordinate.

evas_object_resize() is a function that specifies the position of an object as an absolute value. Enter parameter values in the following order: the object whose properties you want to specify, the X coordinate (width), and the Y coordinate (height).

Run the app again, and you can see the position of the Label has changed.



1:10 PM

Hello World

7) Enabling Various Resolutions Using a Box Container

Specifying absolute coordinates as shown above is convenient. However, you cannot use the various resolutions of your terminal that way. We are now going to learn how to enable various resolutions by using a Box container.

Add a new function on top of the create_base_gui() function.

```
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_EXPAND);
          /* 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() is a function that adds a widget into a Box container. A Box is a container that successively adds widgets or containers in the horizontal or vertical orientation. It performs a similar role as Android's LinearLayout. This function will be used a lot in other examples later on. A detailed explanation of the Box container will be given in a Box example.

For the first parameter of the my_box_pack() function, enter the handle of the Box container. For the second parameter, enter the widget or container to be added to the Box container.
For the third parameter, enter a hint for the horizontal size of the widget. Entering EVAS_HINT_EXPAND or 1.0 specifies the horizontal size of the widget as large as possible; entering 0.0 specifies the size as small as possible. In the case of Bg widgets, the default size is 0. Therefore, if you specify 0.0, the widget will not appear on the screen.

For the fourth parameter, enter a hint for the vertical size of the widget.
For the fifth parameter, enter a horizontal position for the widget. Entering 0.0 left-aligns the widget; entering 0.5 center-aligns the widget; entering 1.0 right-aligns the widget; and entering EVAS_HINT_FILL or -1 lets the widget occupy the whole horizontal area.
For the 6th parameter, enter a vertical position for the widget. Entering 0.0 top-aligns the widget; entering 0.5 middle-aligns the widget; entering 1.0 bottom-aligns the widget; and entering EVAS_HINT_FILL or -1 lets the widget occupy the whole vertical area.

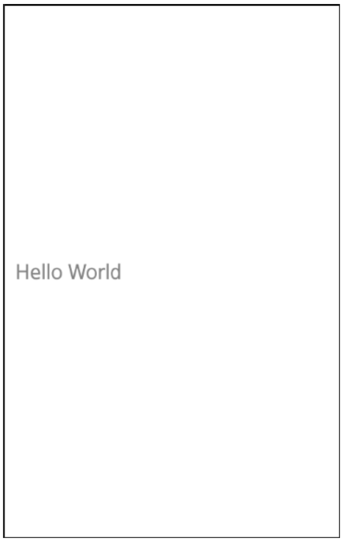
Modify the source code at the bottom of the create_base_gui() function as shown below.

```
    /* 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_EXPAN
D);
    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);
}
```

The symbol { } was used to clarify the inclusion relationship between the upper Window and the bottom Window. It is not necessary to use the symbol. However, because there is a huge difference in readability between when you use it and when you do not, we recommend you make a habit of using it.

elm_box_add() is an API that creates a Box container.

Let's run the example again. Little difference has been made. The widget, however, is now displayed in the same position on the screens of terminals with different resolutions.



8) Inefficient Code Modifications

By removing unnecessary parts from the source code that are added by default when you have created a source project, you can enhance performance. Let’s modify the source code using the method recommended by Carsten Haitzler, the developer of EFL.

Go to the top of the source file, delete the win_delete_request_cb() function, and modify the win_back_cb() function as shown below.

```
/*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() is the callback function used at the below code of the create_base_gui() function. This is an event used on PCs and therefore means little on mobile devices.

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

Go to the create_base_gui() function and modify the beginning of the function as shown below.

```
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, ad);

     /* 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_EXPAN
D);
     elm_win_resize_object_add(ad->win, ad->conform);
     evas_object_show(ad->conform);
```

Using the elm_policy_set() function, we made the app close when the last window closes.

In the case of mobiles devices, the elm_win_wm_rotation_supported_get() function is supported regardless of the model. Therefore, it does not require consideration and has been removed.

Lastly, let's remove unnecessary event callback functions. Go to the bottom of the source file and delete the following four callback functions:
- ui_app_orient_changed()
- ui_app_region_changed()
- ui_app_low_battery()
- ui_app_low_memory()

Then, from the main() function at the bottommost of the source file, delete the part that specifies the functions above as callback functions.

```
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_LOW_MEMORY],
APP_EVENT_LOW_MEMORY, 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_L
ANGUAGE_CHANGED, ui_app_lang_changed, &ad);
    //ui_app_add_event_handler(&handlers[APP_EVENT_REGION_FORMAT_CHANGED], APP_
EVENT_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) Related APIs

appdata_s: a structure that saves app information.

void create_base_gui(appdata_s *ad): a function that creates a window and various containers and widgets that compose the screen.

void win_delete_request_cb(): an event function that is executed when a request for deleting the app occurs. This function is not directly called.

void win_back_cb(): an even function that is executed when the Back button is clicked. This function is not directly called.

Evas_Object *elm_win_util_standard_add(char *name, char *title): a function that creates a Window object. The window is the topmost object in the screen layout. An app has a single window. You can place a widget in a window. However, the more common method is to add a container and then add a widget on top of the container.

void elm_win_wm_rotation_available_rotations_set(Elm_Win *obj, const int *rotations, unsigned int count): an API that specifies the orientation of the screen. If you pass the four angles, 0, 90, 180, and 270, as an array, all screen orientations are supported.

void evas_object_smart_callback_add(Evas_Object *obj, const char *event, Evas_Smart_Cb func, const void *data): an API that specifies an event callback function for a smart object such as a widget or container. For the first parameter, pass the object where the event occurs; for the second parameter, the type of the event; for the third parameter, the name of the callback function; and for the fourth parameter, user data. If the type of the event is "delete,request", it means that the object shall be deleted.

void eext_object_event_callback_add(Evas_Object *obj, Eext_Callback_Type type, Eext_Event_Cb func, void *data): an API that specifies an event callback function for an object. You can use this API for both smart objects and general objects. For the first parameter, pass the object where the event occurs; for the second parameter, the type of the event; for the third parameter, the name of the callback function; and for the fourth parameter, user data. EEXT_CALLBACK_BACK indicates a Back button click event.

Evas_Object *elm_conformant_add(Evas_Object *parent): a function that creates a Conformant container. A Conformant changes the size of the window when a new element such as a keypad is added to the screen. An app must have no more than a single Conformant. It is also possible for an app to not have a Conformant. A Conformant is necessary to display an indicator (status bar) at the top of the screen. It is also possible not to display an indicator when a Conformant exists.

void elm_win_indicator_mode_set(Elm_Win *obj, Elm_Win_Indicator_Mode mode): an API that specifies whether to display an indicator.

void elm_win_indicator_opacity_set(Elm_Win *obj, Elm_Win_Indicator_Opacity_Mode mode): an API that specifies the opacity of the indicator.

void evas_object_size_hint_weight_set(Evas_Object *obj, double x, double y): an API that specifies the rough size of an object. The following are the parameters listed in order: the object, horizontal size hint, and vertical size hint. EVAS_HINT_EXPAND means specifying as large a size as the space allows.

void elm_win_resize_object_add(Elm_Win *obj, Evas_Object *subobj): an API that resizes the window object while adding a different object to it.

Evas_Object *elm_label_add(Evas_Object *parent): a function that creates a Label widget. A Label widget lets you display text as well as change text properties such as the font size and color by using HTML tags.

void evas_object_show(Evas_Object *obj): a function that displays the object on the screen. When an object is created, the default value of the object is Hide. You can use the evas_object_show() function commonly for all objects.

void elm_object_text_set(Evas_Object *obj, const char *text): a function that changes the caption text of a widget. You can use this function for Button and Entry widgets as well as Label widgets.

void evas_object_move(Evas_Object *obj, Evas_Coord x, Evas_Coord y): an API that specifies the size of an object as absolute values. / parameters: the object whose properties you want to specify, the position of the X coordinate, and the position of the Y coordinate.

void evas_object_resize(Evas_Object *obj, Evas_Coord w, Evas_Coord h): an API that specifies the position of an object as absolute values. / parameters: the object whose properties you want to specify, width, the Y coordinate (height).

6. Using a Label Widget

You need to use a label widget to display text on the screen. A Label widget lets you change text properties such as the font size and color by using HTML tags.

1) Center-Aligning Label Text

Create a new source project and specify the project name as 'LabelEx.' To do so, select [File > New > Tizen Native Project] in the main menu of Eclipse and, when a popup window appears, select [Template > MOBILE-2.4 > Basic UI Application].

After a source project is created, open the labelex.c file in the src folder, go to the create_base_gui() function, and modify the Label widget-creating code as shown below.

```
        /* 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-1 */
        ad->label = elm_label_add(ad->conform);
```

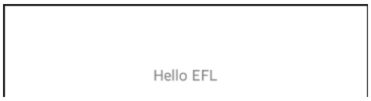
```
	elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
	//evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
	//elm_object_content_set(ad->conform, ad->label);
	evas_object_move(ad->label, 120, 80);
	evas_object_resize(ad->label, 240, 80);
	evas_object_show(ad->label);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

elm_object_text_set() is an API that specifies the caption text of a widget. Enter the HTML tag corresponding to your desired properties for the second parameter to apply the properties to the text. <align=center> center-aligns the text.

evas_object_move() is an API that specifies the top-left starting position of a widget. For the second parameter, enter the desired X coordinate. For the third parameter, enter the desired Y coordinate.

evas_object_resize() is an API that specifies the size of a widget. For the second parameter, enter the desired width. For the third parameter, enter the desired height.

Build and run the source project, and you will now see the text 'Hello EFL' displayed on the screen. The text is center-aligned in the horizontal orientation.



Hello EFL

2) Changing the Font Size

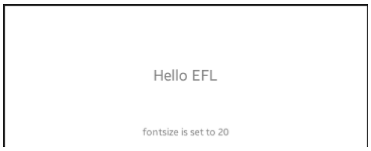
We are now going to change the font size of the caption text of a Label widget. Add a new code at the end of the create_base_gui() function. This is a code that creates the second Label widget.

```
    /* 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_EXP
AND);
    //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 to
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 is a common variable for objects placed on the screen such as widgets and containers. Therefore, it does not matter if you declare a Conformant and a Label as the same variable type.

elm_object_text_set() is a function that changes the caption text of a widget. You can use this function for Button and Entry widgets as well as Label widgets. Inserting HTML tags into this function displays the text in the same way as when seen in your web browser.
<font_size=20> is a tag that specifies the font size as 20 pixels.
<align=center> is a tag that specifies the position of the horizontal alignment as the center.

Run the example again, and you will now see the second Label widget added and the text 'fontsize is set to 20' displayed. The text is displayed smaller in size than the text in the first Label. Center alignment has been applied for the horizontal alignment of the text.



Hello EFL

fontsize is set to 20

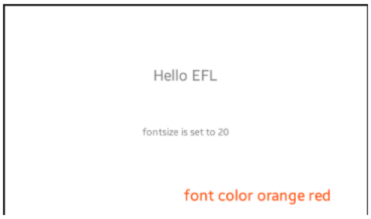
3) Changing the Font Color

We are now going to change the font color in a Label widget. As with the font size, use HTML tags to change the font color. Add a new code at the end of the create_base_gui() function.

```
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);
```

<color=#FF4500FF> inserted in the elm_object_text_set() function is a tag that specifies a color. The AARRGGBB format has been employed. The opacity is the maximum value, which equals opaque; the color blue is 45; the color green is 0; and the color red is the maximum value. Entering <color=#F40F> produces the same result.

Run the example again, and you will now see the third Label widget added and orange-colored text displayed.



Hello EFL

fontsize is set to 20

font color orange red

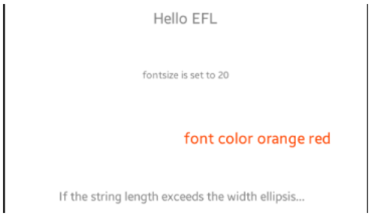
4) Ellipses

We are now going to learn how to display ellipses when a given text is too long and goes beyond the right end of the Label. Add a new code at the end of the create_base_gui() function.

```
    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 the
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() is an API that applies ellipses to a Label widget. For the first parameter, pass the Label widget whose properties you want to specify, and for the second parameter, pass true or false. EINA_TRUE is a Boolean value used in Tizen that indicates 'true.' To specify 'false,' enter EINA_FALSE.

Run the example again, and you will now see the fourth text displayed as well as ellipses displayed at the right end.



Hello EFL

fontsize is set to 20

font color orange red

If the string length exceeds the width ellipsis...

5) Multi-Line Text

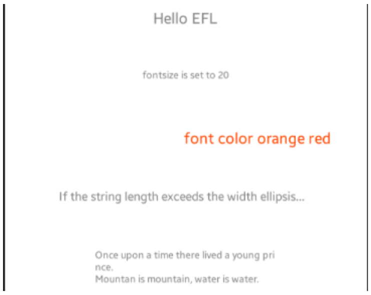
To display multiple lines of text in a Label widget, use the <br/> tag. Add a new code at the end of the create_base_gui() function.

```
    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() is an API that sets automatic line wrapping for a Label widget. Passing EINA_TRUE to the second parameter applies automatic line wrapping.

We created the fifth Label widget and entered long text. We broke the text into two lines using the <br/> tag.

Run the example again, and you will now see the fifth text displayed and the lines broken where the <br/> tags are placed.



Hello EFL

fontsize is set to 20

font color orange red

If the string length exceeds the width ellipsis...

Once upon a time there lived a young pri
nce.
Mountan is mountain, water is water.

6) Related APIs

Evas_Object: a common variable for objects placed on the screen such as widgets and containers. Therefore, it does not matter if you declare different objects such as a Conformant and a Label as the same variable type.

void elm_object_text_set(Evas_Object *obj, char *text): an API that changes the caption text. You can use this function for Button and Entry widgets as well as Label widgets. Inserting HTML tags into this function displays the text in the same way as when seen in your web browser.

EINA_TRUE: a Boolean value used in Tizen that indicates 'true.'
EINA_FALSE: a Boolean value used in Tizen that indicates 'false.'

void elm_label_ellipsis_set(Evas_Object *obj, Eina_Bool ellipsis): an API that displays ellipses when the caption text of a Label widget goes beyond the right end. Entering EINA_TRUE for the second parameter displays ellipses, and entering EINA_FALSE cancels the display of ellipses.

7. Using a Button Widget

Button widgets receive the user's input and are used most often among the widgets. Button widgets can call a touch event and apply a background image using EDJE.

1) Changing the Text in a Label Widget

Create a new source project and specify the project name as 'ButtonEx.'

After a source project is created, open the buttonex.c file in the src folder and add a new function on top of create_base_gui() function. The function you add is the same as the one used in the HelloWorld example.

```
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_EXPAND);
         /* 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);
}
```

Then, go to the create_base_gui() function and modify the code that creates a Label widget.

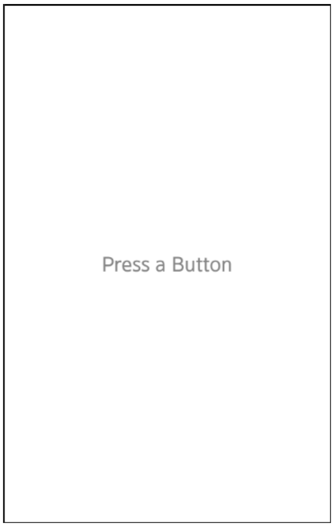
```
     /* Conformant */
     ad->conform = elm_conformant_add(ad->win);
     elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
     elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
     evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
 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

     {
          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>Press a Button</>");
            //evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EV
AS_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);
}
```

The following are the parameters of the my_box_pack() function listed in order:

- Box container
- Child window
- Horizontal size hint: 1.0 = maximum. 0.0 = minimum.
- Vertical size hint: 1.0 = maximum. 0.0 = minimum.
- Horizontal position: 0.0 = left. 0.5 = center. 1.0 = right. -1 = full.
- Vertical position: 0.0 = top. 0.5 = middle. 1.0 = bottom. -1 = full.

Run the example. You will now see the text 'Press Button' displayed on the screen.



Press a Button

2) Creating a Button Widget

We are now going to add a Button widget-creating code at the end of the create_base_gui() function.

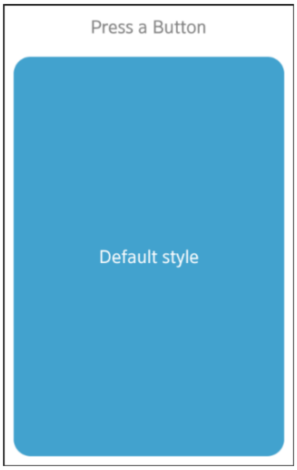
```
    {
        /* Label*/
        ad->label = elm_label_add(ad->conform);
        elm_object_text_set(ad->label, "<align=center>Press a Button</>");
        my_box_pack(box, ad->label, 1.0, 0.0, -1.0, 0.5);

        /* 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() is an API that creates a Button widget.

Specify the caption text as 'Default style' using the elm_object_text_set() function.

The other functions have already been covered in previous examples.

Build and run the source project, and you will see a Button widget added and the text 'Default style' displayed. Clicking the button does not produce any change. This is because we have not yet defined an event function.



3) Defining a Button Event Function

The Enlightenment Foundation Libraries (EFL) used in Tizen define event functions in the callback style. You may be familiar with this method if you have experience in web programming. Let's define a Button event function now. Add a function called 'btn_default_cb().'

One thing to note is that because this function is called by the create_base_gui(), btn_default_cb() must be placed before create_base_gui(). Declaring function headers in a header file lets functions call each other regardless of the order.

```
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)
{
```

The btn_default_cb() function receives three parameters. The first parameter is user data sent by the calling side. In this case, we will use app data (appdata_s). The second parameter is the object where an event has occurred. In this case, the object is the first Button. The third parameter is the structure that contains event information.

elm_object_text_set() is an API that changes the caption text of a widget.

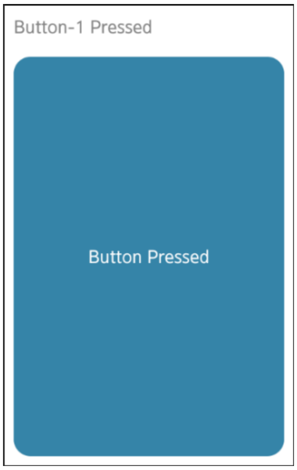
elm_object_text_set() is an API that changes the caption text of a widget.

Now, we need to go to the Button-creating code and specify the function above as a callback function. Go to the create_base_gui() function and add a line of code.

```
/* 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() is a function that specifies an event callback function for a smart object such as a widget or container. Evas objects are all objects displayed on the screen. Smart objects are supplementary objects other than basic objects (Line, Rect, Polygon, Text, Image) provided by Evas. For the first parameter, enter the object where an event occurs. For the second parameter, enter the type of the event. 'clicked' indicates a click event. For the third parameter, enter the name of the callback function. For the fourth parameter, enter the type of data that will be sent to the callback function. In this case, it is app data.

Run the example again and click the Button. The texts in the Label and Button will be changed.



3) Applying an Icon to a Button: Reorder

Several icon images can be displayed in a Button. We are going to learn how to apply each icon image.

Add a new code at the end of the create_base_gui() function. This is a code that creates the second 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);
    }
```

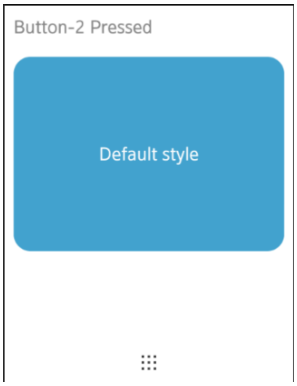
elm_object_style_set() is a function that specifies the style of a widget. For the first parameter, enter the widget to which a style will be applied. For the second parameter, enter the type of the style. ‘icon_reorder’ displays a reorder icon.

We specified the callback function btn_icon_reorder_cb for evas_object_smart_callback_add(). This function has not been implemented yet. As shown below, add a new function on top of the create_base_gui() function.

```
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");
}
```

We defined a callback function for the second Button, and we also added a text-changing code to the Label widget.

Run the example again, and you will now see that a second Button with an icon image has been added. Click the second Button. The text in the Label and the second Button will be changed.


Button-2 Pressed

4) Applying an Icon to a Button: + and -

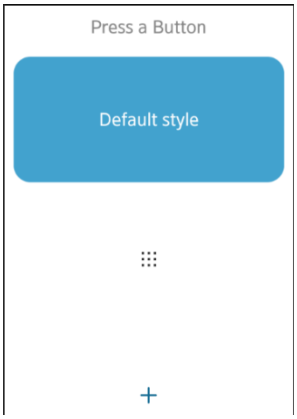
We are now going to create a Button to which a plus icon and a minus icon have been applied. Add a new code at the end of the create_base_gui() function. This is a code that creates the third Button.

```
        /* 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);
    }
```

Enter ‘icon_expand_add’ for the second parameter of the elm_object_style_set() function, and a plus icon image will then be displayed.

The way to specify a callback function is following the same procedure as before and therefore has been omitted here.

Run the example again, and you will now see that the third Button with a plus icon image has been added.



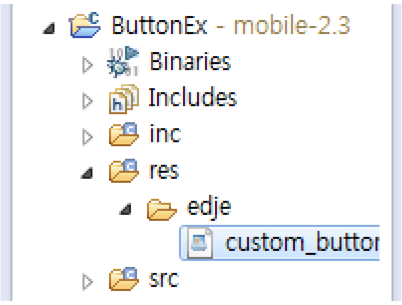
5) Applying a Background Image to a Button

To apply a background image to a Button widget, you need to use a theme. EFL uses a theme called EDJE. It is a hassle to create a theme file from scratch, so we are going to import a theme file from the appendix.

Create a new folder under the /res folder of the source project and specify the folder's name as 'edje.' To do so, right-click the /res folder and select [New > Folder] in the shortcut menu. When a popup window appears, enter 'edje' in the folder name field and click the Finish button.

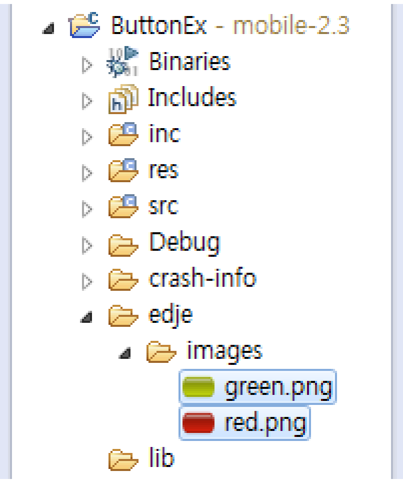
Go to the appendix's /etc/edje folder and copy the custom_button.edc file to the newly created /res/edje folder.

You can create a new folder by right-clicking the /res folder and selecting [New > Folder] in the shortcut menu. To copy a file to your desired folder, drag and drop the file to the folder using the mouse.


ButtonEx - mobile-2.3
Binaries
Includes
inc
res
edje
custom_buttor
src

To apply a background image to a Button, you need an image file. We are now going to copy an image file that will be used as a background image. Create a new folder under the root folder of the source project and specify the folder’s name as ‘edje.’ Then, create a new folder under the /edje folder and specify the folder’s name as ‘images.’

Go to the appendix's /image folder and copy the two files, green.png and red.png, to the newly created /edje/images folder.


ButtonEx - mobile-2.3
Binaries
Includes
inc
res
src
Debug
crash-info
edje
images
green.png
red.png
lib

Now, we must add source code. Add a new code at the end of the create_base_gui() function. This code registers the EDJE file as a theme and creates the fourth Button widget.

```
        /* 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() is a function that finds the absolute path of the /res folder, adds a subfolder path, and then returns the path. For the first parameter, enter the subfolder path. In this case, 'edje/custom_button.edj' has been sent, and therefore, the full path is ~/res/edje/custom_button.edj. This function has not been created yet. We will create this function shortly.

elm_theme_extension_add() is a function that registers a theme information file. Pass the path of the EDJE file to the second parameter of this function.

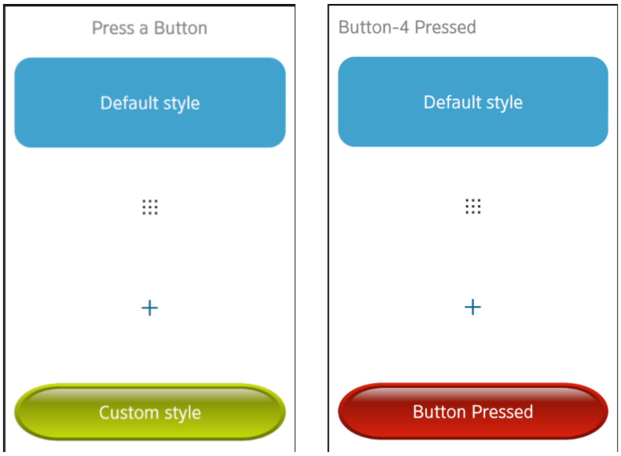
elm_object_style_set() is a function that applies a custom theme to a Button widget. We specified the name of the custom theme as 'customized.' This theme is defined in the custom_button.edc file.

Add two new functions on top of the create_base_gui() function. The first function is a callback event function for the fifth Button, and the second function is a function that returns the absolute path of the /res folder.

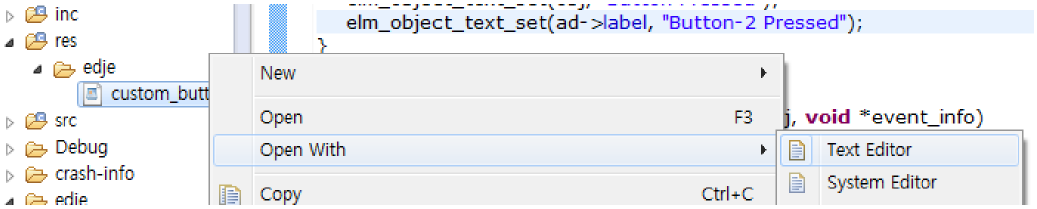
```
static void
btn_custom_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s* ad = data;
    elm_object_text_set(obj, "Button Pressed");
    elm_object_text_set(ad->label, "Button-5 Pressed");
}

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);
    }
}
```

The second function 'app_get_resource()' has called app_get_resource_path(). This function finds the absolute path of the /res folder and returns the path.

Run the example again, and you will now see the fifth Button widget with a green image applied has been added. Clicking the Button changes it to a red image. Canceling the click changes the image back to green and changes the caption text.



There are so many elements to explain regarding the EDJE file, so for now we are going to address only the most essential. Open the EDJE file in the editor. Right-click the /res/edje/custom_button.edc file and select [Open With > Text Editor] in the shortcut menu.



```
    elm_object_text_set(ad->label, "Button-2 Pressed");
}
j, void *event_info)
```

You can see the define statement at the top of the file. Within the statement, ICON_NORMAL indicates the file name of the background image in normal status. If you need to change the file name of the background image to 'btn_n.png,' modify the define statement as follows:

#define ICON_NORMAL btn_n.png

ICON_PRESSED below it, as you may have guessed, indicates the file name of the Button background image in pressed status.

```
#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
```

Further down, you will see the following code:

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

For the name property under ‘group,’ you can specify a name for the theme. Specify a customized theme name and insert the following into the source code.

elm_object_style_set(btn, "customized");

6) Related APIs

Evas_Object *elm_button_add(Evas_Object *parent): an API that creates a Button widget.

void evas_object_smart_callback_add(Evas_Object *obj, const char *event, Evas_Smart_Cb func, const void *data): an API that specifies a callback event function for a widget or container. / parameters: the object where an event occurs, the type of the event (‘clicked’ indicates a click event), the name of the callback function, and data that will be sent to the callback function.

Eina_Bool elm_object_style_set(Evas_Object *obj, const char *style): an API that specifies a style for a widget. / parameters: a widget to which a style will be applied. Specifies the type of the style. ‘icon_reorder’ indicates the reorder icon style; ‘icon_expand_add’ indicates the plus icon style; ‘icon_expand_delete’ indicates the minus icon style; and 'customized' indicates the customized style.

void elm_theme_extension_add(Elm_Theme *th, const char *item): an API that registers a theme information file. / parameters: the path of the theme and the EDJE file.

char *app_get_resource_path(void): an API that finds the absolute path of the /res folder and returns the path.

8. Creating a Background with a Bg Widget

There are two ways to display a background color or background image in a widget. One is to apply a theme using EDJE, and the other is to use a Bg widget. By using a BG widget, you can display a background with ease.

1) Creating a color Bg Widget

Create a new source project and specify the project name as 'BgEx.'

When the source project is created, open the bgex.c file in the src folder and add a new function on top of create_base_gui() function.

```
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() is a function that adds a widget or container to the Table container. Table is a container that divides the screen into multiple cells and places a widget in a desired cell. By using a Table widget, you can use various monitor resolutions.

The following are the parameters of the my_table_pack() function listed in order:

- Table container
- Sub-window
- Horizontal cell number
- Vertical cell number
- Number of horizontal cells
- Number of vertical cells

We are now going to go to the create_base_gui() function and create a Bg widget. Conformant and Label will be deleted because they are not necessary for this example.

```
    /* 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, "Hello EFL");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
    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);
```

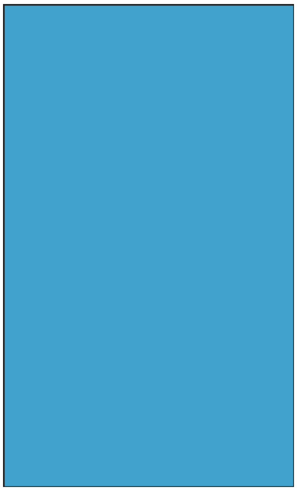
Elm_table_add() is an API that creates a Table container.

elm_bg_add() is an API that creates a Bg widget.

elm_bg_color_set() is an API that specifies a background color for a Bg widget. For the first parameter, enter the Bg widget to which properties will be applied. For the second to fourth parameters, enter a color value. Enter color values within a range of 0 to 255 for red, green, and blue, in this order.

Run the example, and you will see that the screen has changed to blue.

In other words, a Bg widget has been created in the whole area of the screen.

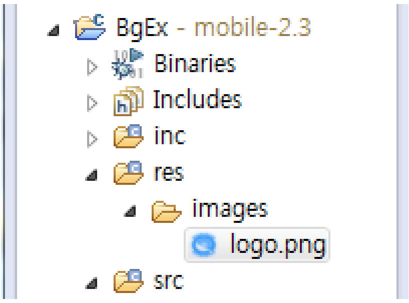


2) Applying an Image to a Bg Widget: Original Size

In this section, we are going to create a Bg widget to which a background image has been applied. To do so, an image file is necessary.

We are now going to copy an image file that will be used as a background image. Create a new folder under the /res folder of the source project and specify the folder's name as 'images.' Right-click the /res folder and select [New > Folder] in the shortcut menu. When a popup window appears, enter 'images' in the folder name field and click the Finish button.

Then, go to the appendix's /Image folder and copy the logo.png file to the newly created /res/images folder. To copy a file to your desired folder, drag and drop the file to the folder using the mouse.



BgEx - mobile-2.3
Binaries
Includes
inc
res
images
logo.png
src

We are now going to create a new Bg widget and specify a background image. Add a new code at the end of the create_base_gui() function.

```
{
    /* 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() is a function that finds the absolute path of the /res folder, adds the subfolder path, and then returns the path. For the first parameter, enter the subfolder path. In this case, 'images/logo.png' has been sent, and therefore, the full path is ~/res/images/logo.png. This function has not been created yet. We will create the function shortly.

elm_bg_option_set() specifies a style in which an image is displayed. Entering ELM_BG_OPTION_CENTER for the second parameter displays the image at the center of the Bg widget in its original size.

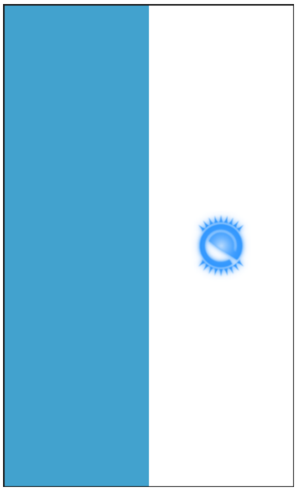
elm_bg_file_set() is a function that specifies an image file for a Bg widget. For the first parameter, specify the Bg widget to which properties will be applied. For the second parameter, pass the path of the file.

We are now going to create a function that returns the absolute path of the /res folder. Add a new code on top of the create_base_gui() function.

```
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);
    }
}
```

Because the source code is the same as the one used in the ButtonEx example, it will not be explained in detail.

Run the example again, and you will now see an image displayed on the right side of the screen. The right side is the area specified for the second Bg, and the image has been displayed at the center in its original size.



3) Applying an Image to a Bg Widget: Resizing While Maintaining the Original Proportion

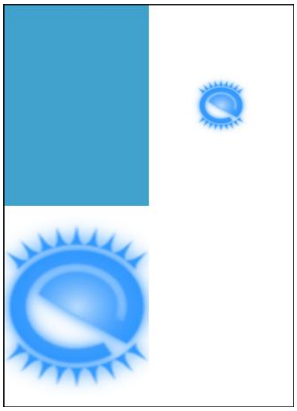
In this section, we are going to learn how to make an image fill an area while maintaining the original proportion of the image. Add a new code at the end of the create_base_gui() function.

```
/* 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);
    }
```

Entering 'ELM_BG_OPTION_SCALE' for the second parameter of the elm_bg_option_set() function makes an image fill the Bg area while maintaining the original aspect ratio of the image.

Run the example again, and you will now see a large image displayed in the bottom left corner of the screen.



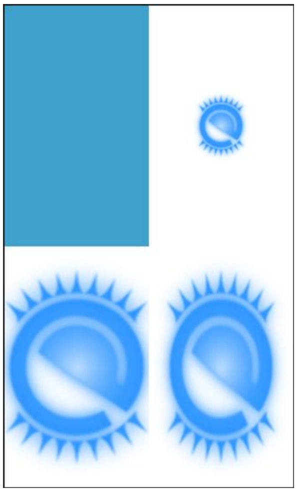
4) Applying an Image to a Bg Widget: Resizing Ignoring the Original Proportion

In this section, we are going to learn how to make an image fill an area ignoring the original proportion of the image. Add a new code at the end of the create_base_gui() function. This is a code that creates the fourth Bg widget.

```
        /* 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);
    }
```

Entering 'ELM_BG_OPTION_STRETCH' for the second parameter of the elm_bg_option_set() function makes an image fill the Bg area.

Run the example again, and you will now see a large image displayed in the bottom right corner of the screen. You can see that the aspect ratio of displayed image is different from the original.



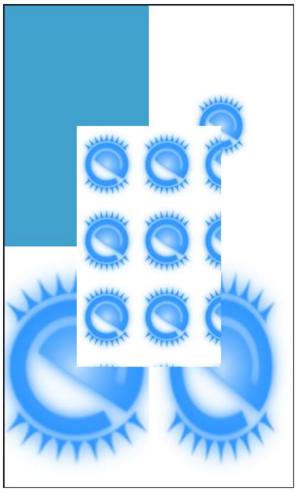
5) Applying an Image to a Bg Widget: Tile Style

In this section, we are going to learn how to repeatedly display an image in a tile-like pattern. Add a new code at the end of the create_base_gui() function. This is code that creates the fifth Bg widget.

```
        /* 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);
    }
```

Entering 'ELM_BG_OPTION_TILE' for the second parameter of the elm_bg_option_set() function displays an image repeatedly.

Run the example again, and you will now see that an image is repeatedly displayed at the center of the screen.



6) Related APIs

Evas_Object *elm_bg_add(Evas_Object *parent): a function that creates a Bg widget.

void elm_bg_color_set(Evas_Object *obj, int r, int g, int b): a function that specifies a background color for a Bg widget. / parameters: the Bg widget to which properties will be applied and, for the second to fourth parameters, color values. Enter color values within a range of 0 to 255 for red, green, and blue, in this order.

void elm_bg_option_set(Evas_Object *obj, Elm_Bg_Option option): a function that specifies a style in which an image will be displayed in a Bg widget. / parameters: The first parameter is the Bg widget object while the second is the image placement style. The style types are as follows:

- ELM_BG_OPTION_CENTER: displays the image at the center of the Bg area in its original size.

- ELM_BG_OPTION_SCALE: displays the image so that it fills the Bg area while maintaining its original aspect ratio.
- ELM_BG_OPTION_STRETCH: displays the image so that it fills the Bg area.
- ELM_BG_OPTION_TILE: displays the image repeatedly.

Eina_Bool elm_bg_file_set(Evas_Object *obj, const char *file, const char *group): a function that specifies an image file for a Bg widget. / parameters: the Bg widget object and the path of the file.

9. Resizing the Screen with a Conformant Container

To display an indicator at the top of the screen (status bar), you need to use a Conformant container. You can hide an indicator even if a Conformant container exists. A Conformant container is also necessary for resizing the screen when a new panel such as a keypad has appeared. We are going to learn how to use a Conformant container through an example.

1) Hiding the Indicator

Create a new source project and specify the project name as 'ConformantEx.'

After a source project is created, open the bgex.c file under the src folder and add a new function on top of the create_base_gui() function. This function was used previously in the HelloWorld example.

```
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_EXPAN
D);
          /* 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);
}
```

Add a Button widget to implement a feature that hides an indicator. Add new code at the end of the create_base_gui() function. This code creates a Box container and adds a Button widget.

```
     /* 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);
```

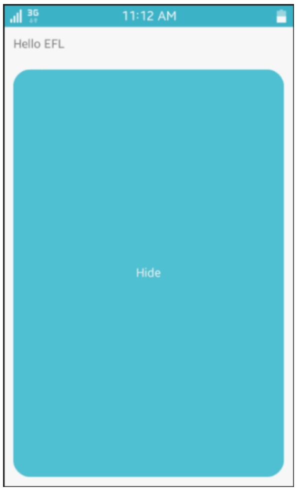
```
    evas_object_show(ad->conform);

    {
        /* 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);

        {
            /* 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);
```

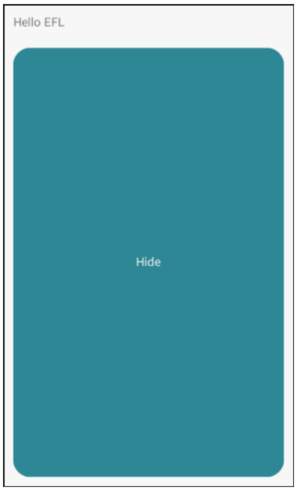
We are going to implement a feature that hides an indicator when a newly added Button is tapped. Add a Button callback function on top of the create_base_gui() function.

```
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() is an API that changes the mode of the indicator. Pass the Window object to the first parameter. An app can only have a single Window. Then, pass the mode type to the second parameter. Passing ELM_WIN_INDICATOR_HIDE hides the indicator.

Build and run the example, and you will now see an indicator at the top of the screen. Click the Hide button, and the indicator will disappear.





There is a difference in the size of Button widgets between when there is an indicator and when there is no indicator. Because we specified the height of the Label widget as minimum, the Button vertically stretches as much as the height of the indicator when the indicator is no longer present.

2) Displaying an Indicator

We are now going to implement a feature that displays an indicator that has previously disappeared. Add new code at the end of the create_base_gui() function. This is a code that creates the second Button.

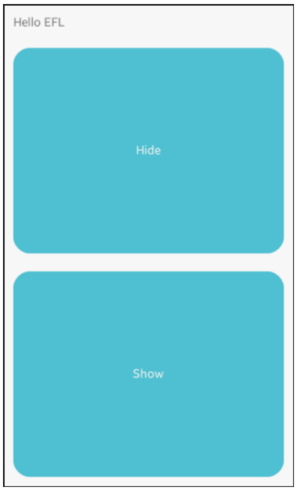
```
            /* Button-1 */
            Evas_Object *btn = elm_button_add(ad->conform);
            elm_object_text_set(btn, "Hide");
            evas_object_smart_callback_add(btn, "clicked", btn_hide_cb, ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

            /* 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);
        }
    }
```

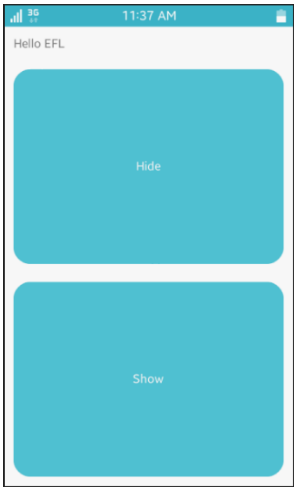
Then, add a callback function for the second Button on top of the create_base_gui() function.

```
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);
}
```

Run the example again, and click the Hide button. The indicator disappears.

Now, click the Show button. The indicator appears again.





3) Creating an Entry Widget

Entry is an editor widget that receives text from the user. Because text is entered by using a keypad, the size of the screen needs to be changed. To that end, you need to add a different container such as a Box or Layout on top of a Conformant container and then add an Entry over the newly added container.

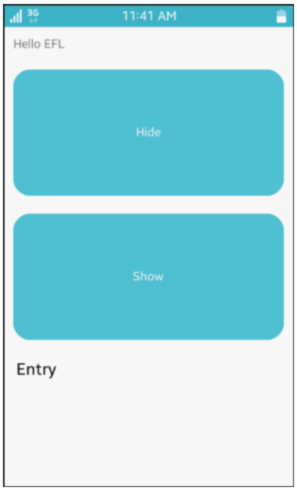
Go to the create_base_gui() function and add new code at the end of the function. This code creates an Entry widget.

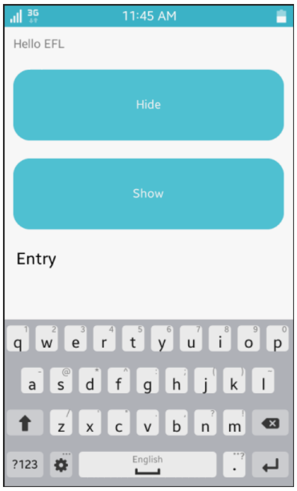
```
        /* Button-2 */
        btn = elm_button_add(ad->conform);
        elm_object_text_set(btn, "Show");
        evas_object_smart_callback_add(btn, "clicked", btn_show_cb, ad);
        my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);

        /* 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() is an API that creates an Entry widget. We have created an Entry on top of a Conformant.

Build and run the example. You will see a Label widget at the top of the screen, and below that, the text 'Entry' which is the Entry widget we created. The background color of the Entry is the same as the color of the screen background, so the boundary is not visible. We will learn how to specify a background color later on.

Tap the text 'Entry.' A keypad appears, and the screen shrinks vertically. Now, we can see 'Entry' clearly.





4) Related APIs

Evas_Object *elm_entry_add(Evas_Object *parent): an API that creates an Entry widget.

void elm_win_indicator_mode_set (Evas_Object *obj, Elm_Win_Indicator_Mode mode): an API that changes the mode of the indicator. / parameters: the Window object and the mode type. Passing ELM_WIN_INDICATOR_HIDE hides the indicator.

10. Using an Entry Widget

To receive text string input from the user, you can use an Entry widget. We are now going to learn how to create an Entry widget and request the content of the user's input.

1) Creating an Entry Widget

Create a new source project and specify the project name as 'EntryEx.'

After the source project is created, open the entryex.c file under the src folder and add a new function on top of the create_base_gui() function. This function adds a widget to a Table container.

```
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);
}
```

Then, go to the create_base_gui() function and create a Box container, a Table container, and an Entry widget.

```
    /* 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_EX
PAND);
        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_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_E
XPAND);
        /* 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 is a container that enables the placing of a widget based on the aspect ratio of the screen. We need to use a Table because in order to use a Bg as the background of an Entry, the two must be positioned in the same space. We used a Box to specify the distance between widgets.

elm_table_padding_set() is an API that specifies paddings.

elm_entry_add() is an API that creates an Entry widget.

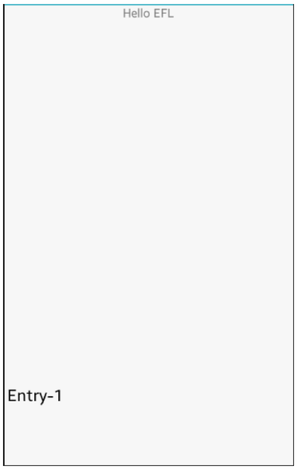
elm_entry_single_line_set() is an API that sets/unsets multi-line entry. Passing EINA_TRUE to the second parameter sets single-line entry only, while passing EINA_FALSE sets multi-line entry.

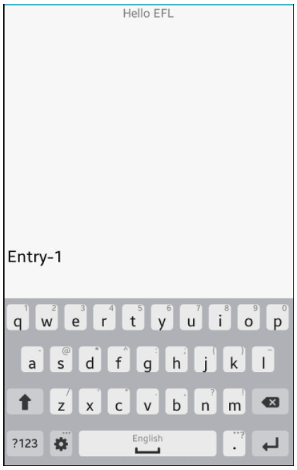
elm_entry_entry_insert() is an API that adds caption text to an Entry widget. In other words, it adds new text next to existing text.

Build and run the source project. The text 'Entry-1' you see at the bottom of the screen is the Entry widget we created.

If you click the Entry widget, a keypad appears so that you can enter new text.

You can use your keyboard for text entry in emulators.





2) Displaying Guide Text

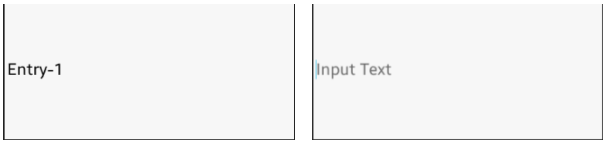
Guide text is text that explains the role of the editor. If an entry field is empty, Guide text appears, and, once you enter text using a keypad, the Guide text disappears. Smartphones have small screens, so it is necessary to use space efficiently. Because Guide text can replace Label widgets, using Guide text helps save space.

Add new code to the Entry-creating code of the create_base_gui() function.

```
/* 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);
```

Passing 'elm.guide' to the second parameter of the elm_object_part_text_set() function enables specifying Guide text for an Entry widget. Pass the content of the Guide text to the third parameter.

Run the example again and delete the content of the Entry widget. The text 'Input Text' appears. The Guide text disappears once you enter text into the Entry widget.



3) Creating a Background with the Bg Widget

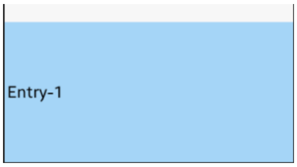
Because the background color of an Entry widget is white, and the color of the screen background is also white, the boundary between the Entry widget and the screen is unclear. There are two ways to display a background color in an Entry widget. One is to use a Bg widget and the other is to use EDJE.

We are now going to learn how to use a Bg widget as a background. Add new code at the end of the create_base_gui() function. This code creates a Bg widget and specifies the same coordinates as an Entry widget for the Bg widget. Please take note that a Bg widget must be created before an Entry widget.

```
    /* 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);
```

We created a Bg widget before an Entry widget and specified the same coordinates for both widgets. Doing so makes the Bg widget look like the background of the Entry widget.

Run the example again, and you will now see the background of the Entry widget.



4) Importing Text Entered in the Entry Widget

We are now going to learn how to request text entered in an Entry widget when a Button is tapped. To use an Entry widget in event functions, it is necessary to declare it as a global variable or AppData.

At the top of the source code, you will see an appdata_s structure defined. This is a structure to store data used in apps. By default, Window, Conformant, and Label are declared. We will add an Entry here.

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

Then, go back to the create_base_gui() function and add new code at the end of the function. This code creates a Button widget and stores an Entry widget in an AppData structure.

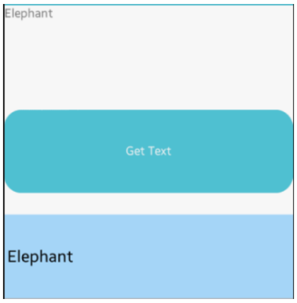
```
    /* Label*/
    ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "<align=center>Hello EFL</align>");
    my_table_pack(table, ad->label, 0, 0, 4, 1);

    /* 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;
```

We are now going to create a Button callback function. Add a new function on top of the create_base_gui() function. This code requests an Entry widget's text and displays it in a Label widget.

```
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() is a function that requests the text of an Entry widget. Think of the function as the reverse of elm_entry_entry_insert().

Run the example again, change the text of the Entry widget, and then click the Button. The text entered in the Entry is displayed in the Label.



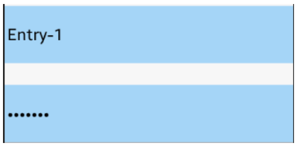
5) The Entry Widget Dedicated to Password Entry

When you enter a password, it is necessary to display the text as wildcard characters so that the password cannot be seen by people nearby. We are now going to create an Entry widget dedicated to password entry. Add new code at the end of the create_base_gui() function. This code creates a second Bg widget and an Entry widget.

```
        /* 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() is an API that sets password-only mode. Passing EINA_TRUE to the second parameter sets password-only mode, while passing EINA_FALSE turns off the mode.

Run the example again. When you enter text in the second Entry, the text is displayed as wildcard characters so that other people cannot identify the password.


Entry-1

•••••••

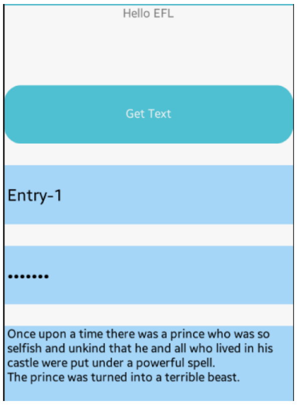
6) Entering Multiple Lines in the Entry Widget

To enter multiple lines of text in the Entry widget, you can use the same way as the Label widget. Add new code at the end of the create_base_gui() function. This code adds a third Bg widget and an Entry widget.

```
        /* 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 i
n his castle were put under a powerful spell.<br>The prince was turned into a
terrible beast.</align></font_size>");
        my_table_pack(table, entry, 0, 4, 4, 2);
    }
```

elm_object_text_set() is an API used when we changed the caption text of the Label and Button widgets. It can also be used for the Entry widget. However, it must be used together with the elm_object_signal_emit() function. You need to pass HTML tags to the second parameter.
<font_size=20> is a tag that specifies the font size of text.
<align=left> is a tag that specifies the horizontal alignment as left alignment.
<br> is a tag for line wrapping.

Run the example again. Multiple lines of text are displayed in the third Entry. Try entering text yourself using the keypad. You can see that the font size has changed. Properties specified with HTML tags only apply to output, not to input.



Entry-1

•••••••

Once upon a time there was a prince who was so selfish and unkind that he and all who lived in his castle were put under a powerful spell.
The prince was turned into a terrible beast.

7) Related APIs

Evas_Object *elm_entry_add(Evas_Object *parent): an API that creates an Entry widget.

void elm_entry_single_line_set(Evas_Object *obj, Eina_Bool single_line): an API that sets/unsets multi-line. / parameters: the Entry object and whether to display in single-line mode. Passing EINA_TRUE to the second parameter sets single-line entry only, while passing EINA_FALSE sets multi-line entry.

void elm_entry_entry_insert(Evas_Object *obj, const char *entry): an API that adds caption text to an Entry widget. In other words, it adds new text next to existing text.

void elm_object_part_text_set(Evas_Object *obj, const char *part, const char *text): an API that specifies Guide text for an Entry widget. / parameters: the Entry object and the area to which the text is applied. Passing 'elm.guide' enables specifying Guide text for an Entry widget. Pass the content of the Guide text to the third parameter.

char *elm_entry_entry_get(Evas_Object *obj): an API that requests the text of an Entry widget. Think of the function as the reverse of elm_entry_entry_insert().

void elm_entry_password_set(Evas_Object *obj, Eina_Bool password): an API that sets the password-only mode. Passing EINA_TRUE to the second parameter sets password-only mode while passing EINA_FALSE unsets the mode.

11. Using a Check Widget

To enable choosing between On and Off, you can use the Check widget. We are now going to learn how to create a Check widget and request the event of the user.

1) Creating a Check Widget

Create a new source project and specify the project name as 'CheckEx.'

After the source project is created, open the source file (~.c) under the src folder and add new variables to the appdata structure.

```
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;
```

We added four Check widget variables. We are now going to create a Check widget with the source code. Add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
        /* 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);
}
```

Then, go to the create_base_gui() function and add new code.

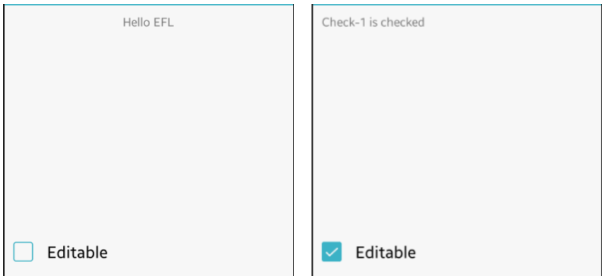
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win,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

We created a Box and a Check widget, and also added the Check widget and a Label widget to the Box.

elm_check_add(Evas_Object *parent) is an API that creates a Check widget.

elm_object_text_set(obj, text) is an API that specifies the caption text of a widget.

We are now going to build and run the example. A Check widget is displayed on the screen. Tapping the square area displays a checkmark. Tapping the area again makes the checkmark disappear.



Hello EFL

□ Editable

Check-1 is checked

☑ Editable

2) Changing the symbol of the Check Widget

We are now going to change the symbol of a Check widget. To do so, you need to change the style. Add new code to the create_base_gui() function.

```
        /* 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);
    }
```

The second and third Check widgets are created.

elm_object_style_set(Evas_Object *obj, const char *style) is an API that specifies the style of an object. The style types for the Check widget are as follows:

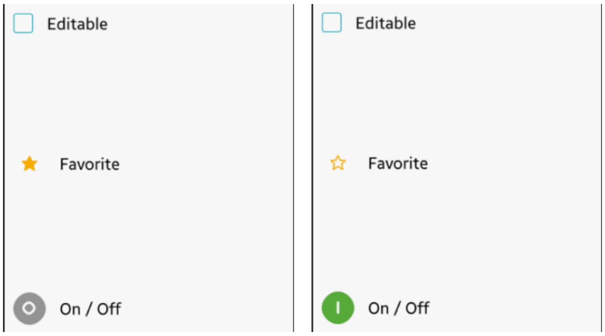
- favorite: asterisk symbol
- on&off: power on/off symbol

We specified the style of the second Check widget as "favorite." An asterisk symbol is displayed.

We specified the style of the third Check widget as "on&off." A power on/off symbol is displayed.

elm_check_state_set(Elm_Check *obj, Eina_Bool state) is an API that specifies the On/Off status of a Check widget. Passing EINA_TRUE to the second parameter changes the state of the Check widget to On while passing EINA_FALSE changes it to Off.

Run the example again. The second and third Check widgets are created, and the symbol is changed now.



3) Requesting an On/Off Event for the Check Widget

We are now going to call an event that occurs when the user taps a Check widget. To do so, you need to specify an event callback function for the Check widget. Add new code to the create_base_gui() function.

```
    /* 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, a
d);
    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, a
d);
    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, a
d);
    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) is an API that specifies a callback function for a smart object such as a widget or container. Passing 'changed' to the second parameter lets a callback function be called when the status of the Check widget changes.

We are now going to create a callback function. Add a new function on top of the create_base_gui() function.

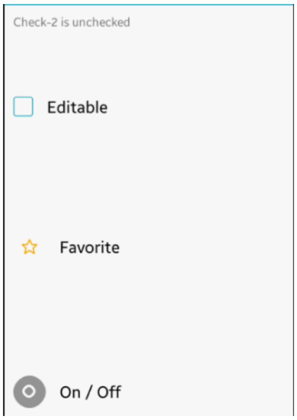
```
static void
check_changed_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    int check_num = 0;
    if( obj == ad->check1 )
        check_num = 1;
    else if( obj == ad->check2 )
        check_num = 2;
    else if( obj == ad->check3 )
        check_num = 3;
    else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

This code figures out what number a Check widget has been tapped by the user, calls the status of the Check widget, and then displays the status in the Label widget.

The first parameter of the Check widget status changing event function receives user data; the second parameter receives the object where an event has occurred; and the third parameter receives event information.

elm_check_state_get(const Elm_Check *obj) is an API that returns the On/Off status of a Check widget. The function performs a role that is the opposite of the role of the elm_check_state_set() function.

Run the example again. When you tap a Check widget, the number and status of the Check widget are displayed in the Label widget.



Check-2 is unchecked

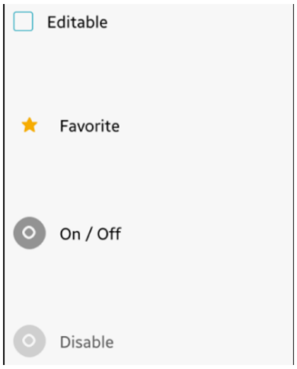
4) Enabling/Disabling the Check Widget

Lastly, we are going to learn how to disable a Check widget. Add code that creates a fourth Check widget to the create_base_gui() function.

```
          /* 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, a
d);
          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_c
b, 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) is an API that changes the active/inactive status of an object. Passing EINA_TRUE to the second parameter disables the object, while passing EINA_FALSE enables the object.

Run the example again. A fourth Check widget is added. Because the Check widget has been disabled, tapping it does not product any change.



5) Related APIs

Evas_Object* elm_check_add(Evas_Object *parent) is an API that creates a Check widget.

Eina_Bool elm_object_style_set(Evas_Object *obj, const char *style) is an API that specifies the style of an object. The style types for the Check widget are as follows:
 - "popup": checkmark
- favorite: asterisk symbol
- on&off : power on/off symbol

void elm_object_text_set(obj, text) is an API that specifies the caption text of a widget.

void elm_check_state_set(Elm_Check *obj, Eina_Bool state) is an API that specifies the On/Off status of a Check widget. Passing EINA_TRUE to the second parameter changes the state of the Check widget to On, while passing EINA_FALSE changes it to Off.

void evas_object_smart_callback_add(Evas_Object *obj, char *event, Evas_Smart_Cb func, void *data) is an API that specifies a callback function for a smart object such as a widget or container. Passing 'changed' to the second parameter lets a callback function be called when the status of the Check widget changes.

Eina_Bool elm_check_state_get(const Elm_Check *obj) is an API that returns the On/Off status of a Check widget. The function performs a role that is the opposite of the role of the elm_check_state_set() function.

void elm_object_disabled_set(Evas_Object *obj, Eina_Bool disabled) is an API that changes the active/inactive status of an object. Passing EINA_TRUE to the second parameter disables the object while passing EINA_FALSE enables the object.

12. Using a Radio Widget

To enable selecting one menu from multiple menus, you can use the Radio widget. We are now going to learn how to create a Radio widget and request the event of the user.

1) Creating a Radio Widget

Create a new source project and specify the project name as 'RadioEx.'

After the source project is created, open the source file (~.c) under the src folder and add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
        /* 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);
}
```

Then, go to the create_base_gui() function and add new code.

```
    /* 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_EX
PAND);
        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);
```

Three Radio widgets are created.

elm_radio_add(Evas_Object *parent) is an API that creates a Radio widget.

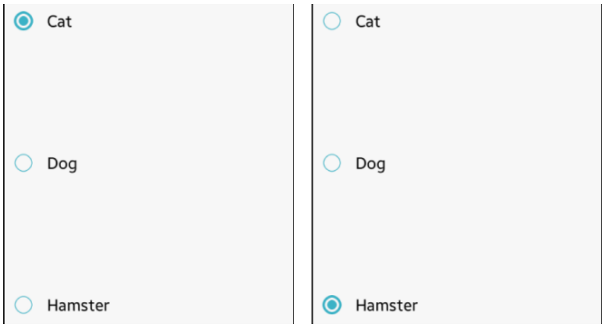
elm_radio_state_value_set(Elm_Radio *obj, int value) is an API that specifies a status value for a Radio widget. In the case of the Radio widget, several widgets function as a group. Therefore, it is necessary to specify a different ID value for each widget.

radio_group is the Radio group variable. Use the first Radio widget as a Radio group.

elm_radio_group_add(Elm_Radio *obj, Evas_Object *group) is an API that adds a widget to a Radio group. Pass the widget to be added to the first parameter. Then, pass the Radio group to the second parameter.

Build and run the example. Three Radio widgets are displayed on the screen, and if you tap a Radio widget, a checkmark is displayed.

If you select a different Radio widget, the position of the checkmark is moved.



2) Requesting a Radio Widget Item Selection Event

In this section, we are going to find out which item number has been selected by calling an event that occurs when the user taps a Radio widget. To do so, you need to specify an event callback function for the Radio widget. Add new code to the create_base_gui() function.

```
/* 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) is an API that specifies a callback function for a smart object such as a widget or container. Passing 'changed' to the second parameter lets a callback function be called when the status of the Radio widget changes.

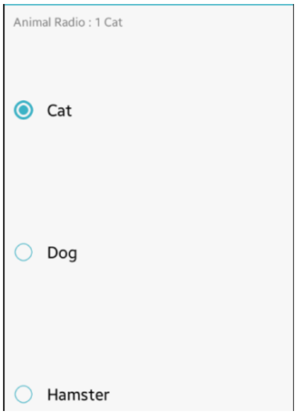
We are now going to create a callback function. Add a new function on top of the create_base_gui() function.

```
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);
}
```

This code calls the status value of the Radio widget selected by the user and then displays the status value and the animal name of the Radio widget in the Label widget.

elm_radio_value_get(const Elm_Radio *obj) is an API that returns the status value of a Radio widget. It returns the status value of the currently selected Radio widget.

Run the example again. Tapping the Radio widget displays the status value and animal name of the Radio widget.



Animal Radio : 1 Cat

3) A Second Radio Group

In the case of the Radio widget, several widgets function as a group. We are now going to learn add three new Radio widgets and divide them into two Radio groups. Add new code to the create_base_gui() function.

```
/* 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);
    }
```

This code adds three new Radio widgets. We specified the widgets as 1, 2, and 3, respectively using the elm_radio_state_value_set() function.

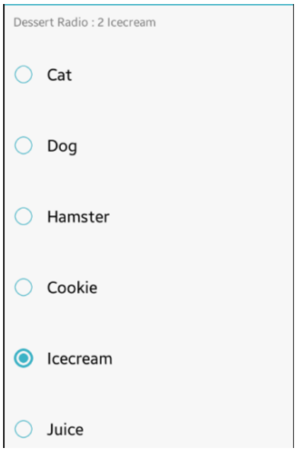
We specified the first Radio widget as a Radio group, and we also added the second and third Radio widgets to the Radio group using the elm_radio_group_add() function.

Then, we specified the name of the event callback function as 'radio_dessert_cb.' We are now going to create a callback function. Add a new function on top of the create_base_gui() function.

```
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);
}
```

The content of the new function is very similar to that of the radio_animal_cb() function.

Run the example again, and you will now see a total of six Radio widgets displayed. Tapping the Radio widget displays checkmarks for the first and second groups.



Dessert Radio : 2 Icecream

○ Cat

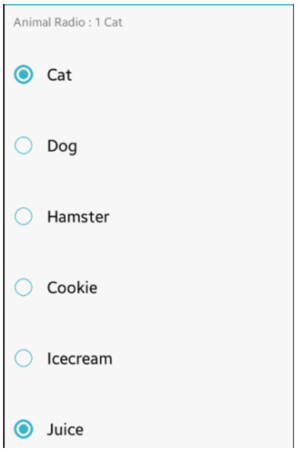
○ Dog

○ Hamster

○ Cookie

● Icecream

○ Juice



Animal Radio : 1 Cat

● Cat

○ Dog

○ Hamster

○ Cookie

○ Icecream

● Juice

4) Changing Selected Radio Items with Source Code

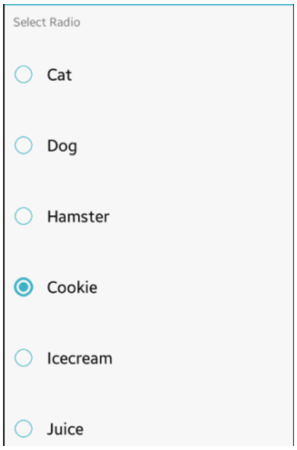
We are now going to implement a feature that automatically selects the first item of the second group when the example is executed. Add new code at the end of the create_base_gui() function.

```
        /* 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);

            /* 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) is an API that sets a value for a Radio group. The Radio widget that matches the set value will be selected.

Run the example again. The first item of the second group is selected automatically.



Select Radio

○ Cat

○ Dog

○ Hamster

● Cookie

○ Icecream

○ Juice

5) Related APIs

Evas_Object* elm_radio_add(Evas_Object *parent): an API that creates a Radio widget.

void elm_radio_state_value_set(Elm_Radio *obj, int value): an API that specifies a status value for a Radio widget. In the case of the Radio widget, several widgets function as a group. Therefore, it is necessary to specify a different ID value for each widget.

void elm_radio_group_add(Elm_Radio *obj, Evas_Object *group): an API that adds a widget to a Radio group. Pass the widget to be added to the first parameter. Then, pass the Radio group to the second parameter.

void evas_object_smart_callback_add(Evas_Object *obj, char *event, Evas_Smart_Cb func, void *data): an API that specifies a callback function for a smart object such as a widget or container. Passing 'changed' to the second parameter lets a callback function be called when the status of the Radio widget changes.

int elm_radio_value_get(const Elm_Radio *obj):an API that returns the status value of a Radio widget. It returns the status value of the currently selected Radio widget.

void elm_radio_value_set(Elm_Radio *obj, int value): an API that sets a value for a Radio group. The Radio widget that matches the set value will be selected.

13. Using a Popup

To display a simple message to the user, you can use a popup. You can make a popup close after a certain amount of time has passed, and you can also make a popup receive the user's input.

1) Creating a Button Widget

Create a new source project and specify the project name as 'PopupEx.'

After the source project is created, open the popupex.c file under the src folder and add new variables to the appdata_s structure.

```
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' is a container to place widgets in sequence.

'popup' is the handle of a popup widget. It is used to close a popup or to transmit data.

'entry' is used for the user to enter text in a popup window.

'popupNum' saves the index number of the current popup window.

Add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
        /* 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);
}
```

Then, go to the create_base_gui() function and create four Button widgets. In this example, we will create four different types of popups.

```
    /* 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_HIN
T_EXPAND);
        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_1butt
on, 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_3butt
on, 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_text,
 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);
        }
```

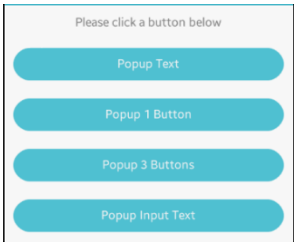
```
    }

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

We created a Box on top of a Conformant, and we also added one Label and four Button widgets on top of the Box.

To prevent build errors, it is necessary to create callback functions for Buttons. Add four new functions on top of the create_base_gui() function. We will define the content of the functions later.

```
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)
{ }
```

Build and run the source project, and then you can see a Label and Buttons. We specified the area heights of the first to third Buttons as minimum. We specified the area height of the fourth Button as maximum by passing 1.0 to the fourth parameter of the my_box_pack() function. In addition, we specified the height of each Button as minimum by passing 0.0 to the sixth parameter. Doing so concentrates the widgets at the top.



2) Creating a Text Popup

In this section, we are going to create the most basic popup that displays a text message. Add popup-creating code to the first Button's callback function.

```
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;
}
```

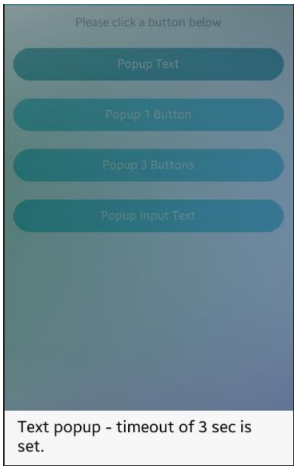
elm_popup_add() is an API that creates a popup.

elm_popup_align_set() is an API that specifies the position of a popup. For the second parameter, enter a horizontal position for the popup. Entering ELM_NOTIFY_ALIGN_FILL lets the popup use the whole horizontal area of the screen. For the third parameter, enter a vertical position for the popup. The acceptable range of values is 0.0-1.0.

evas_object_size_hint_weight_set() is an API that specifies a hint for the size of a widget. The second parameter indicates the width, while the third parameter indicates the height. EVAS_HINT_EXPAND is an option that allocates a widget as much space as possible in a given area.

Specifying '1' as popupNum is to remember that it is a first popup.

Run the example again and click the first Button. A popup appears at the bottom of the screen, and a text message is displayed.



Text popup - timeout of 3 sec is set.

3) Setting an Auto Shutdown Timer

You cannot close popups because there is no Button on them. Let's implement a feature that automatically closes a popup after a certain amount of time has passed.

Add new code to the make_popup_text() function.

```
~
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() is an API that sets a timer event for a popup. For the second parameter, enter a time interval. Entering 3.0 lets a timer event occur at 3 second intervals.

evas_object_smart_callback_add() is an API that specifies a callback function that receives events. Entering 'timeout' for the second parameter lets a callback function be called when a timer event occurs. For the third parameter, enter the name of the callback function.

We are now going to create a timer event function. Add a new function on top of the make_popup_text() function.

This new function deletes a popup and displays text in the Label widget.

```
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() is an API that deletes an object. In this case, we deleted the subject of the timer event. In other words, a popup is deleted.

Run the example again and click the first Button. A popup appears and then disappears automatically after a short while.

4) Closing a Popup When the Block Area is Touched

In this section, we are going to implement a feature that closes an existing popup when the area outside the popup is tapped.

Add new code at the end of the make_popup_text() function.

```
    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);
```

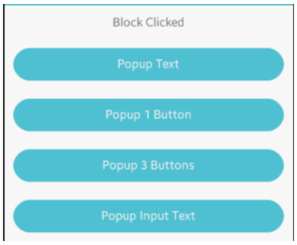
We passed the popup to the first parameter of the evas_object_smart_callback_add() function and 'block,clicked' to the second parameter. Doing so enables requesting an event when the area outside the popup is tapped.

We are now going to create a function that requests this event. Add a new function on top of the make_popup_text() function.

This new function deletes a popup and displays text in the Label widget.

```
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");
}
```

Run the example again, click the first Button, and when a popup appears, click the area outside the popup. The popup disappears, and the text ‘Block Clicked’ is displayed in the Label widget.



5) Adding a Button to a Popup

In this section, we are going to implement a feature that creates a Button on top of a popup that appears when the second Button is tapped and closes the popup when the created Button is tapped.

Add new code to the make_popup_text_1button() function. This code creates a popup and adds a Button on top of the popup.

```
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;
}
```

Now, let's implement a feature that displays text in the Label widget when a Button added on top of a popup is tapped. Add a new function on top of the make_popup_text_1button() function. This new function is an OK button event function.

```
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() is an API that creates a StrBuf object. StrBuf is a structure that makes it easy to use strings.

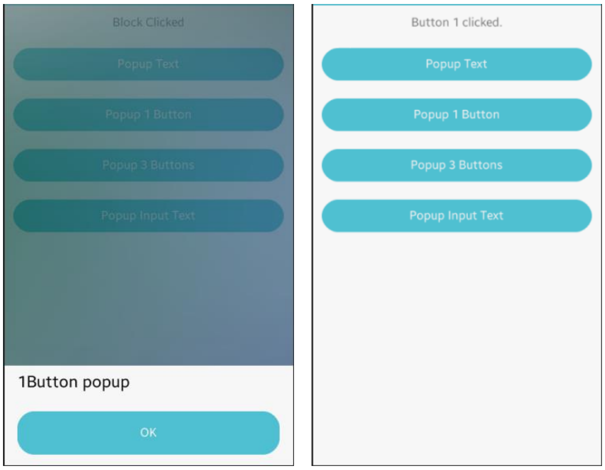
eina_strbuf_append_printf() is an API that adds a new string to a StrBuf.

eina_strbuf_string_get() is an API that returns the string stored in a StrBuf.

eina_strbuf_free() is an API that deletes a StrBuf object.

Run the example again and click the second Button. A popup appears, and you can see a text message and a Button.

Clicking the OK button makes the popup disappear and changes the text in the Label widget.



Block Clicked

1Button popup

Button 1 clicked.

6) Adding Three Buttons to a Popup

In this section, we are going to add three Buttons to a popup. Add new code to the make_popup_text_3button() function. This code creates a popup and adds three Buttons on top of the popup.

Copying and modifying the content of the make_popup_text_1button() function will spare you some typing.

```
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_clicke
d, ad);
   evas_object_size_hint_weight_set(ad->popup, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
   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);
```

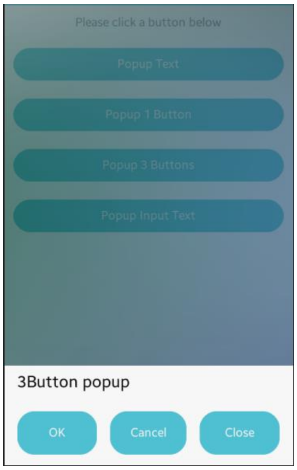
```
    evas_object_show(ad->popup);
    ad->popupNum = 3;
}
```

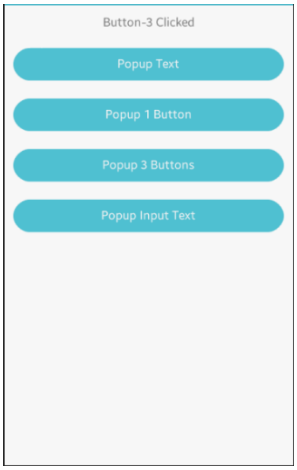
Now, let's implement a feature that displays text in the Label widget when a Button on top of a popup is tapped. Add two new functions on top of the make_popup_text_3button() function. These new functions are event functions for the Cancel button and the Close button.

```
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");
}
```

Run the example again and click the third Button. Three Buttons then appear over a popup.

Clicking any one of the Buttons makes the popup disappear and changes the text in the Label.





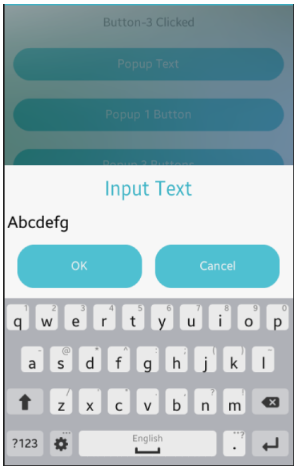
7) Adding an Entry Widget to a Popup

In this section, we are going to implement a feature that adds an entry to a popup so that the user can enter text in the popup. Add new code to the make_popup_input_text() function. This code creates a popup and adds one Entry and two Buttons on top of the popup.

Copying and modifying the content of the make_popup_text_1button() function will spare you some typing.

```
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_EXPAND);
    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;
}
```

Run the example again and click the fourth Button, and you will now see an Entry added to the popup.



Button-3 Clicked

Input Text

Abcdefg

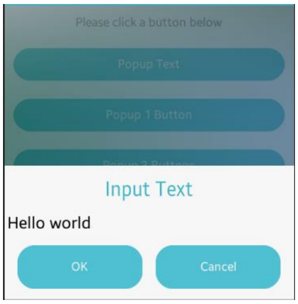
Now, let's implement a feature that, when the user enters text into an Entry and taps the OK button, displays the text in the Label.

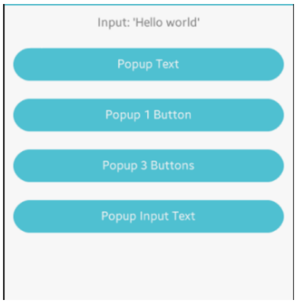
Add new code to the popup_btn1_clicked() function. If the current popup is a fourth popup, this code requests the caption text in the Entry of the popup and displays it in the Label. Then, this code initializes the Entry widget.

```
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;
}
```

Run the example again and click the fourth Button. When a popup appears, enter text in the Entry and click the OK button.

The popup disappears, and the text is displayed in the Label.





8) Related APIs

Evas_Object *elm_popup_add(Evas_Object *parent): an API that creates a popup.

void elm_popup_align_set(Evas_Object *obj, double horizontal, double vertical): an API that specifies the position of a popup. For the second parameter, enter a horizontal position for the popup. Entering ELM_NOTIFY_ALIGN_FILL lets the popup use the whole horizontal area of the screen. For the third parameter, enter a vertical position for the popup. The acceptable range of values is 0.0-1.0.

void evas_object_size_hint_weight_set(Evas_Object *obj, double x, double y): an API that specifies the rough size of an object. / parameters:the Window object, width hint, and height hint. EVAS_HINT_EXPAND is an option that expands the object as much as possible.

void elm_popup_timeout_set(Evas_Object *obj, double timeout): an API that specifies a timer for a popup. For the second parameter, enter a time interval. Entering 3.0 lets a timer event occur at 3 second intervals.

void evas_object_smart_callback_add(Evas_Object *obj, const char *event, Evas_Smart_Cb func, const void *data): an API that specifies a callback function that receives events. Entering 'timeout' for the second parameter lets a callback function be called when a timer event occurs. For the third parameter, enter the name of the callback function.

void evas_object_del(Evas_Object *obj): an API that deletes an object.

14. Using a Slider Widget

To let the user enter values, you need to use the Slider widget. This widget is also useful for scanning through the playlist in an audio or video player.

1) Creating a Slider Widget

Create a new source project and specify the project name as 'SliderEx.'

After the source project is created, open the sliderex.c file under the src folder and add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
        /* 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);
}
```

Then, go to the create_base_gui() function and create a Box container and a Slider widget.

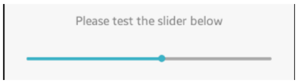
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win,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

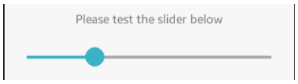
elm_slider_add() is an API that creates a Slider widget.

elm_slider_min_max_set() is an API that specifies the range of a Slider. For the second parameter, enter the minimum value. For the third parameter, enter the maximum value. The variable type is 'double.'

elm_slider_value_set() is an API that specifies the current value of a Slider.

Build and run the source project. You can see a Slider widget. Move the trackbar to the left and right.


Please test the slider below


Please test the slider below

2) Displaying an Indicator in the Slider Widget

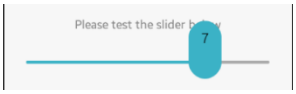
The indicator is a feature that displays the current value of the slider as the trackbar is dragged. Add new code to the code that creates a Slider.

```
/* 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() is an API that specifies whether to display an indicator in a Slider. Passing EINA_TRUE to the second parameter displays an indicator. Passing EINA_FALSE does the opposite.

elm_slider_indicator_format_set() is an API that specifies a format for text displayed in an indicator.

Run the example again and move the trackbar around. Numbers appear above the trackbar.


Please test the slider below

3) Slider Tracking Event

In this section, we are going to request in real time, an event that occurs when the user drags a trackbar. This feature is necessary when you implement a video player.

Add new code to the code that creates a Slider.

```
        /* 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);
```

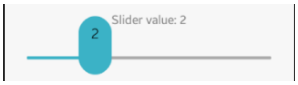
Passing 'changed' to the second parameter of the evas_object_smart_callback_add() function enables requesting an event when the value of the slider changes.

Now, let's define an event function. Add a new function on top of the create_base_gui() function. This code requests the current value of the Slider and displays it in the Label widget.

```
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() is an API that performs the opposite function of elm_slider_value_set(). It returns the current value of the Slider. The variable type is 'double.'

Run the example again and move the trackbar around. The current value of the Slider is displayed in the Label widget.



4) Displaying a Center Point

You can display a center point at the center of a Slider. We are now going to add code that creates a new Label and Slider to the create_base_gui() function.

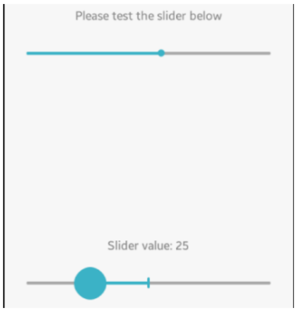
```
        /* Slider-1 */
        Evas_Object *slider = elm_slider_add(ad->conform);
        elm_slider_min_max_set(slider, 0, 9);
        elm_slider_value_set(slider, 5);
        elm_slider_indicator_show_set(slider, EINA_TRUE);
        elm_slider_indicator_format_set(slider, "%1.0f");
        evas_object_smart_callback_add(slider,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

We passed the Slider widget to the first parameter of the elm_object_style_set() function and 'center_point' to the second parameter. Doing so displays a grid at the center of the Slider widget.

We are now going to define a callback function for the second Slider. Add a new function on top of the create_base_gui() function.

```
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);
}
```

Run the example again, and you will now see a second Slider widget created and the center point displayed. Drag the trackbar, and values will be displayed in the second Label.



5) Related APIs

Evas_Object *elm_slider_add(Evas_Object *parent): an API that creates a Slider widget.

void elm_slider_min_max_set(Evas_Object *obj, double min, double max): an API that specifies the range of a Slider widget. Parameters: Slider widget object, minimum value, and maximum value.

void elm_slider_value_set(Evas_Object *obj, double val): an API that specifies the current value of a Slider widget.

void elm_slider_indicator_show_set(Evas_Object *obj, Eina_Bool show): an API that specifies whether to display an indicator in a Slider widget. Passing EINA_TRUE to the second parameter displays an indicator. Passing EINA_FALSE does the opposite.

void elm_slider_indicator_format_set(Evas_Object *obj, const char *indicator): an API that specifies the format of text displayed in an indicator. / parameters: Slider object and text format.

double elm_slider_value_get(const Evas_Object *obj): an API that returns the current value of a Slider. The variable type is 'double.'

Eina_Bool elm_object_style_set(Evas_Object *obj, const char *style): an API that specifies the style of an object. Passing the Slider widget to the first parameter and passing 'center_point' to the second parameter displays a grid at the center of the Slider widget.

15. Adding a Text Item to the List Widget.

To display a list of multiple text items on the screen, you need to use the List widget. The List widget can be scrolled up and down and also enables requesting the user's selection event. We are now going to learn how to use the List widget through an example.

1) Creating a Text List Widget

Create a new source project and specify the project name as 'ListEx.'

After the source project is created, open the source file (~.c) under the src folder and add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
         /* 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);
}
```

Then, go to the create_base_gui() function and create a Box container and a List widget. Next, add 10 text items.

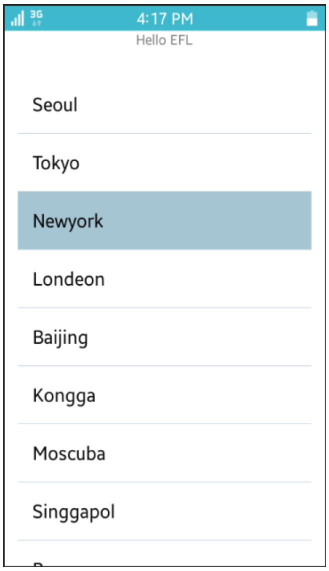
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win, 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_EX
PAND);
        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() is an API that adds a new List widget.

elm_list_item_append() is an API that adds items to a List widget. The first parameter indicates the object of the List widget, while the second parameter indicates the text string. The third parameter indicates the left icon, and the fourth parameter indicates the right icon. For the fifth parameter, specify the item selection event function in callback style. Then, pass user data to the sixth parameter. Although 'appdata' is passed generally, we passed the index numbers in order to identify which number the item is.

elm_list_go() is an API that reflects changes to items on the screen by refreshing the screen. It is necessary to call this function to reflect the change on the screen when a new item has been added or when an existing item has been deleted or modified.

Run the example, and you will see a List widget created and 10 text items displayed. You can scroll through a list by dragging your mouse up and down. If you select an item, a checkmark appears.



Seoul

Tokyo

Newyork

Londeon

Baijing

Kongga

Moscuba

Singgapol

2) Automatically Removing the Checkmark

If you select an item, a checkmark appears and stays displayed. Sometimes this feature is necessary, but other times it is not. We are now going to learn how to disable this feature. Add a new line of code to the create_base_gui() function.

```
        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);
    }
```

This code specifies a callback function that is called when the user selects a List widget item. In other words, a callback function needs to be defined.

Add a new function on top of the create_base_gui() function.

```
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() is an API that displays/removes the checkmark for List widget items. You need to pass the item object to the first parameter. In this case, pass 'event_info' that is passed from the third parameter of the callback function. Passing EINA_TRUE to the second parameter displays the checkmark while passing EINA_FALSE removes the checkmark.

Run the example again and select a List widget item. This time, a checkmark appears and then disappears.

3) Requesting a List Widget Item Selection Event

In this section, we are going to implement a feature that, when the user selects a List widget item, requests the index number and text of the item and displays them in a Label widget.

Modify the code that adds items to a List widget as shown below.

```
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);
```

We specified the item selection event callback function as list_item_clicked. We are now going to define this function.

Add a new function on top of the create_base_gui() function.

```
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);
}
```

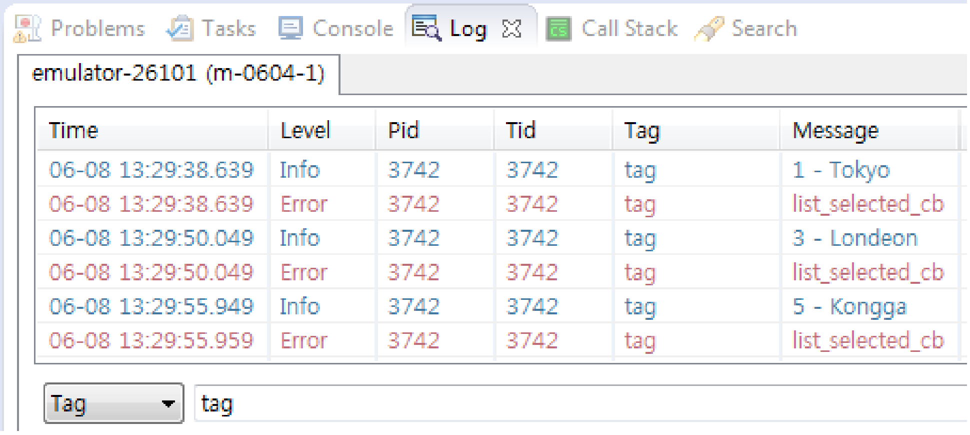
When the user selects an item in a List widget, this function is called. The first parameter receives the index number of the item. The second parameter receives the object of the List widget, and the third parameter receives the object of the selected item.

elm_object_item_text_get() is an API that returns the text of an item.

And after that is code that stores the index number and text of the item in string variables and displays them in the Log pane.

Build the example again and run it. Select an item, and you will now see information about the item displayed in the Log pane. If you cannot see the Log pane at the bottom of Eclipse, select [Window > Show View > Other...] in the menu and then select [Tizen > Log] in the popup window.

To view log messages, select Tag from the combo box at the bottom of the Eclipse Log pane and enter 'tag' in the Edit box to the right.



Time	Level	Pid	Tid	Tag	Message
06-08 13:29:38.639	Info	3742	3742	tag	1 - Tokyo
06-08 13:29:38.639	Error	3742	3742	tag	list_selected_cb
06-08 13:29:50.049	Info	3742	3742	tag	3 - Londeon
06-08 13:29:50.049	Error	3742	3742	tag	list_selected_cb
06-08 13:29:55.949	Info	3742	3742	tag	5 - Kongga
06-08 13:29:55.959	Error	3742	3742	tag	list_selected_cb

Tag: tag

4) Displaying Item Information in the Label Widget

Instead of 'appdata,' we passed the index number of the selected item to the item selection event function. To display information about the item in the Label widget, you need to specify appdata as a global variable.

Add a new line of code at the top of the source file.

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

appdata_s* m_ad = 0;
```

This code declares appdata as a global variable.

Then, implement the global variable at the beginning of the create_base_gui() function as shown below.

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

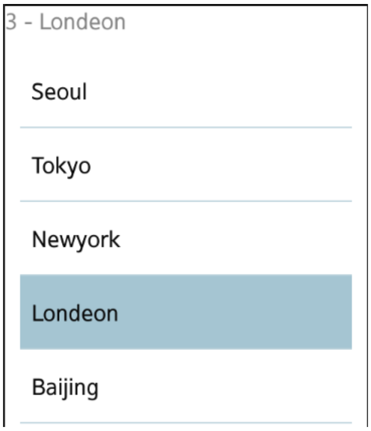
The Label widget can now be used anywhere.

Go back to the list_item_clicked() function and add a new line of code at the end of the function as shown below.

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

This code displays information about a selected List widget item in the Label widget.

Run the example again and select the following List widget item. Information about the selected item is displayed in the Label widget.



3 - Londeon

Seoul

Tokyo

Newyork

Londeon

Baijing

5) Related APIs

Evas_Object *elm_list_add(Evas_Object *parent): an API that adds a List widget. / parameters: parent object.

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): an API that adds items to a List widget. / parameters: List widget, item text, left icon, right icon, name of the item selection event function, and user data.

void elm_list_go(Evas_Object *obj): an API that starts a list. This function must be called first before displaying a List widget on the screen. It is necessary to call this function to reflect the change on the screen when a new item has been added or when an existing item has been deleted or modified. / parameters: List widget.

void elm_list_item_selected_set(Elm_Object_Item *it, Eina_Bool selected): an API that displays/removes the item checkmark. / parameters: List widget and whether to display the checkmark.

const char *elm_object_item_text_get(const Elm_Object_Item *it): an API that returns the text of an item. / parameters: object of the item.

16. Displaying an Icon in a GenList Widget

To display an icon or two-line text in a List widget, you can use the GenList widget. You can also divide lists into groups.

In the case of the GenList widget, adding/deleting an item and adding content such as an icon require processing by a callback function, and a structure must be used to store item data. Therefore, the GenList widget is more cumbersome to use than the List widget. Let's learn in detail how to use the GenList widget through an example.

1) Creating a GenList Widget & Displaying Text

Create a new source project and specify the project name as 'GenListEx.'

After the source project is created, open the source file (~.c) under the src folder and define a new structure.

```
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 is a structure that stores GenList widget item data.

Add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
        /* 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);
}
```

Next, go to the create_base_gui() function and create a Box container and a List widget. Then, add 10 text items.

```
    /* 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_EX
PAND);
        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_G
ENLIST_ITEM_NONE, NULL, id);
            }

            elm_genlist_item_class_free(itc);
        }
    }

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

We created a Box and then created a GenList and added it to the Box.

elm_genlist_add() is an API that creates a GenList widget.

elm_genlist_item_class_new() is an API that creates a class for a GenList item.

Elm_Genlist_Item_Class is the class of a GenList item. You specify a GenList style and callback function using this class. The style types are as follows:
- Passing "end_icon" to the item_style property displays the icon at the right end.

- For the func.text_get property, enter a callback function that specifies the text of the item.
- For the func.del property, enter a callback function that deletes the item.

elm_genlist_item_append() is an API that adds items to a GenList. When this function is called, the callback function entered for the func.text_get property is called automatically. The parameters listed in order are: GenList object, item class, parent item, item type, name of the item selection event callback function, and user data.

elm_genlist_item_class_free() is an API that deletes the class of a GenList item.

Now, let's define a callback function that specifies the text of a GenList widget item. Add a new function on top of the create_base_gui() function.

```
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;
}
```

This code is a callback function that enters data in a GenList widget item. User data is passed to the first parameter. In this case, the item data structure is passed. The index property contains the index number of the item. The second parameter is the object of the item. To the third parameter, the type of the element is passed.

One GenList item can have various elements. For example, if this value is "elm.text," it means the item has the text element.

strcmp (char *, char *) is an API that compares two strings. If the two strings turn out to be the same string, value '0' is returned.

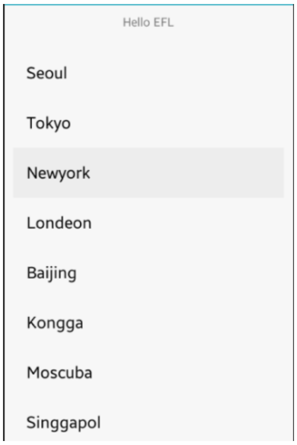
The function described above needs to return the text that matches a given item index number, and the reason for using the strdup() function is to create a new string. Not doing so terminates the function and at the same time makes string data disappear.

Now, let's define a callback function that is called when an item is deleted. Add a new function on top of the create_base_gui() function.

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

User data is passed to the first parameter. This is an item data structure, and as such is what you need to delete.

Build and run the example. A GenList is created, and 10 text items are added.


Hello EFL

Seoul

Tokyo

Newyork

Londeon

Baijing

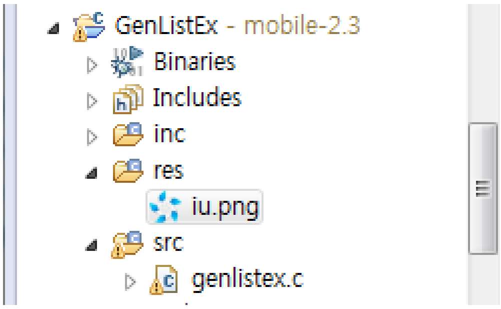
Kongga

Moscuba

Singgapol

2) Displaying an Icon in the GenList Widget

In this section, we are going to display an icon image to the right of an item. To do so, an image file is necessary. Copy the appendix's /Image/iu.png file to the /res folder of the source project.



Then, create a callback function that adds an icon image to a GenList item. As shown below, add the following three functions to the create_base_gui() function.

```
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() is a function that adds the path of the res folder to the name of a file and returns it.

app_get_resource_path() is an API that returns the absolute path of the res folder.

create_image() is a function that creates an Image object to which an image file has been applied.

elm_image_add() is an API that creates an Image object.

elm_image_file_set() is an API that, by specifying the path of an image file for an Image object, loads the image.

gl_content_get_cb() is a callback function that specifies an icon image for a GenList item.

evas_object_size_hint_min_set() is an API that specifies a hint for the minimum size of an object. In this case, we specified absolute value '50' as the hint.

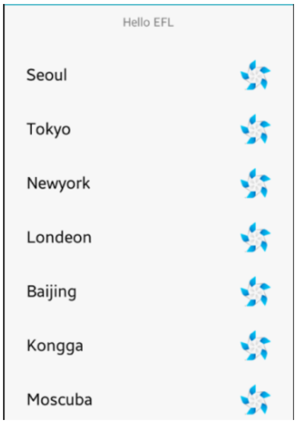
evas_object_size_hint_max_set() is an API that specifies a hint for the maximum size of an object. In this case, we specified absolute value '50' as the hint.

We are now going to assign the icon specifying callback function you just created to GenList items. Go to the create_base_gui() function and add a new line of code.

```
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;
```

Run the example again, and you will now see an icon displayed to the right of GenList items.



3) Requesting the Index Number of a Selected Item

In this section, we are going to implement a feature that, when the user selects an item, displays the index number of the item in a Label widget. Add one global variable and one function at the top of the source file.

```
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 is a global variable that stores appdata.

list_item_clicked() is an item selection event callback function that displays the text of a given item in a Label widget. Item data is passed to the first parameter. The index property contains the index number of the item. We are going to change this index number into a string and display the string in the Label widget.

Store appdata in the global variable. Add a new line of code at the beginning of the create_base_gui() function.

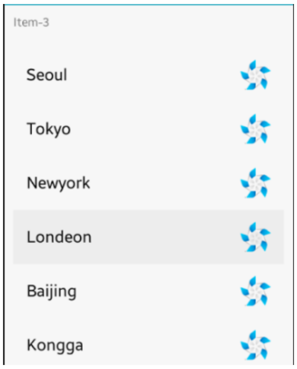
```
create_base_gui(appdata_s *ad)
{
   m_ad = ad;
```

Now, we only need to define a callback function. Modify the code of the create_base_gui() function.

```
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, ELM_GENLIST_ITEM_NONE, NULL, id);
}
```

We specified the item selection event callback function as 'list_item_clicked,' and specified the GenList click event callback function as 'gl_selected_cb.'

Run the example again. If you select an item, the index number of the item is displayed in the Label widget.



Item-3

Seoul

Tokyo

Newyork

Londeon

Baijing

Kongga

4) Related APIs

Evas_Object *elm_genlist_add(Evas_Object *parent): an API that adds a GenList widget. / parameters: parent object.

Elm_Genlist_Item_Class *elm_genlist_item_class_new(void): an API that creates an item class.

Elm_Genlist_Item_Class: the class of a GenList item.
- item_style: the style of the item
- func.text_get: item text specifying callback function
- func.content_get: item icon specifying callback function
- func.del: item deleting event callback function

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): an API that adds items to a GenList. When this function is called, the callback function entered for the func.text_get property is called automatically. / parameters: GenList object, item class, parent item, item type (fifth parameter), name of the item selection event callback function, and user data.

void elm_genlist_item_class_free(Elm_Genlist_Item_Class *itc): an API that deletes a GenList item class. / parameters: object of the GenList item class

char *app_get_resource_path(void): an API that returns the absolute path of the res folder.

sprintf(char*, char*, ...) : an API that creates a format string.

snprintf(char*, int, char*, ...) : an API that creates a format string with a specified length.

Evas_Object *elm_image_add(Evas_Object *parent): an API that creates an Image object. / parameters: parent object.

Eina_Bool elm_image_file_set(Evas_Object *obj, char *file, char *group): an API that specifies an image file for an Image object. / parameters: Image object, path of the image file, and the name of the image group (if the image file is an Edje file).

void evas_object_size_hint_min_set(Evas_Object *obj, Evas_Coord w, Evas_Coord h): an API that specifies a hint for the minimum size of an object. / parameters: object, width hint, and height hint.

void evas_object_size_hint_max_set(Evas_Object *obj, Evas_Coord w, Evas_Coord h): an API that specifies a hint for the maximum size of an object. / parameters: object, width hint, and height hint.

strcmp(char*, char*): an API that compares the lengths of two strings. If the two strings turn out to have the same length, value 0 is returned.

strdup(char*): an API that creates and returns another identical string.

void elm_genlist_item_selected_set(Elm_Object_Item *it, Eina_Bool selected): an API that displays/removes the item checkmark. / parameters: List widget and whether to display the checkmark.

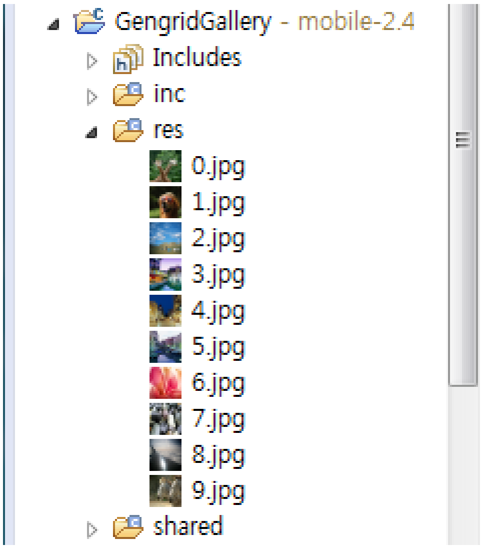
17. Creating a Complex Gallery Widget

We are now going to learn how to create a complex widget by combining two or more widgets. In this example, we will create a Gallery widget using a GenGrid widget and a Bg widget.

1) Creating a GenGrid Widget & Adding an Image

Create a new source project and specify the project name as 'GengridGallery.'

In this example, we are going to create a Gallery widget that displays images. Therefore, image files are needed Copy 10 image files, from 0.jpg through 9.jpg, under the appendix's /Image folder to the /res folder of the source project.



Then, open the source file (~.c) under the src folder, add new variables to the appdata structure, and then define the new structure.

```
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;
```

We added GenGrid and Bg variables to the AppData structure.

itemdata is a structure that stores GenGrid item information. Item information consists of index numbers and image file paths.

We are now going to create a GenGrid widget and add images. The method to do so is similar to the case of the GenList widget. Add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
        /* 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);
}
```

Go to the create_base_gui() function and add new code that creates a Box container, a GenGrid widget, and a Bg widget. Label will not be used in this example, so delete it.

```
    /* 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_EX
PAND);
        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() is a function that creates and then returns a GenGrid widget. We will create this widget in a short while.

elm_bg_color_set() is a function that specifies a background color for a Bg widget. This feature is unnecessary in this example, but we used it nonetheless to identify the position of the Bg widget. Annotate it after the completion of this example if you want to.

We are now going to define a function that creates a GenGrid widget. Add a new function on top of the create_base_gui() function.

```
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(60));
    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() is an API that creates a GenGrid widget.

elm_gengrid_item_size_set() is an API that specifies the size of the icon item of a GenGrid widget.

elm_gengrid_horizontal_set() is an API that sets/unsets the slide orientation to horizontal. The slide orientation is set to vertical by default.

elm_gengrid_item_class_new() is an API that creates an item class for a GenGrid item.

Elm_Gengrid_Item_Class is the item class of a GenList widget. The class types are as follows:
 - func.content_get: assign the name of the callback function that specifies the icon item.
 - item_style: specify the item style. The default setting is "default".
 - func.text_get: assign the name of the callback function that specifies the text item.
 - func.del: assign the name of the item deleting event callback function. Data can be deleted using this function.

app_get_resource_path() is an API that returns the absolute path of the /res folder.

elm_gengrid_item_append() is an API that adds new items to a GenGrid. The parameters listed in order are: GenGrid object, item class, item data structure, name of the item selection event callback function, and user data.

We are now going to define a function that adds icon images to a GenGrid item. Add a new function on top of the create_gengrid() function. This function is called automatically once a new item is created.

```
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;
}
```

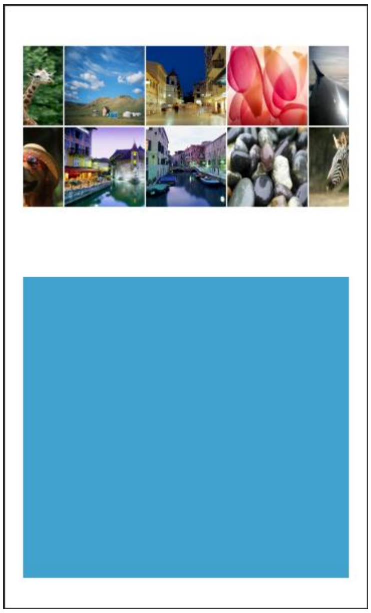
The first parameter of the function above is the user data, and the second parameter is the item object. The type of the element is passed to the third parameter.
A GenGrid widget item consists of multiple elements, and you need to create icon images when the element type is "elm.swallow.icon."

elm_image_add() is an API that creates an image object.

elm_image_file_set() is an API that, by specifying the path of an image file for an Image object, loads the image.

elm_image_aspect_fixed_set() is an API that specifies whether to crop the image. Passing EINA_TRUE to the second parameter lets the size of the image be reduced without any area cropped. Passing EINA_FALSE causes a cropped area in the image so that the image fills the whole item area. The default setting is EINA_TRUE.

Build and run the example. Ten icon images are displayed at the top of the screen. Drag the screen to the left and right. The image list is scrolled. The square shown below is the Bg widget.



2) Displaying a Selected Icon in the Bg Widget

In this section, we are going to implement a feature that, when the user selects an icon from a GenGrid, displays the image in a Bg widget.

Add a global variable at the top of the source file. This global variable stores 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;
```

Then, initialize the global variable at the beginning of the create_base_gui() function.

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

We are now going to create an event callback function for when the user selects a GenGrid item. Modify the code of the create_gengrid() function at the bottom as shown below. We specified the item selection event callback function as 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;
}
```

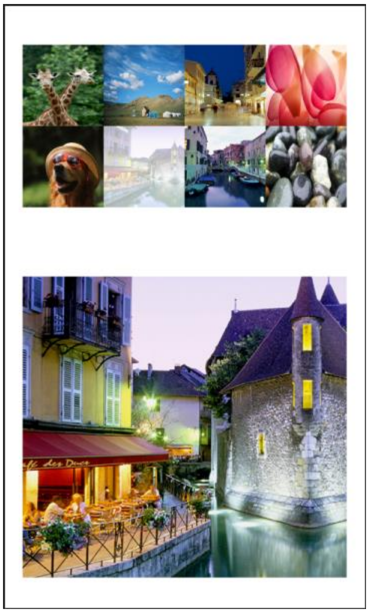
Add a new function on top of the create_gengrid() function. The function added is the item selection event callback function.

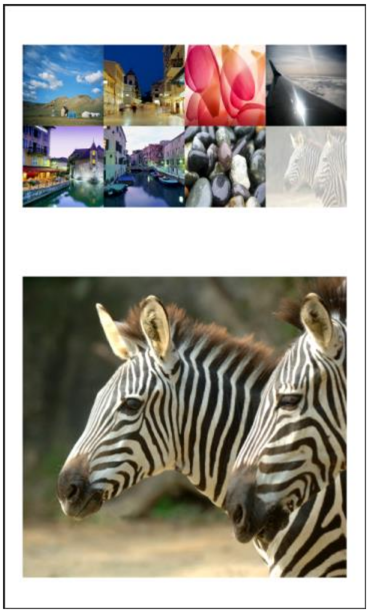
```
static void
gengrid_it_cb(void *data, Evas_Object *obj, void *event_info)
{
    itemdata_s *id = data;
    elm_bg_file_set(m_ad->bg, id->path, NULL);
}
```

The first parameter of the function is the item data structure. The path property contains the path of the image file.

elm_bg_file_set() is an API that specifies a background image file for a Bg widget.

Run the example again and select an item icon. The matching image is displayed in the Bg widget.





3) Related APIs

Elm_Gengrid_Item_Class: the class of a GenGrid item. The class types are as follows:
 - func.content_get: assign the name of the callback function that specifies the icon item.
 - item_style: specify the item style. The default setting is "default".
 - func.text_get: assign the name of the callback function that specifies the text item.
 - func.del: assign the name of the item deleting event callback function. Data can be deleted using this function.

Evas_Object*elm_gengrid_add(Evas_Object *parent): an API that creates a GenGrid widget. / parameters: parent object.

void elm_gengrid_item_size_set(Evas_Object *obj, Evas_Coord w, Evas_Coord h): an API that specifies the icon item size of a GenGrid. / parameters: GenGrid object, width, and height.

void elm_gengrid_horizontal_set(Evas_Object *obj, Eina_Bool horizontal): an API that sets/unsets the GenGrid slide orientation to horizontal. The slide orientation is set to vertical by default. / parameters: GenGrid object and EINA_TRUE or EINA_FALSE.

Elm_Gengrid_Item_Class *elm_gengrid_item_class_new(void): an API that creates a GenGrid item class.

char *app_get_resource_path(void): an API that returns the absolute path of the /res folder.

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): an API that adds new items to a GenGrid. / parameters: GenGrid object, item class, item data structure, name of the item selection event callback function, and user data.

Evas_Object *elm_image_add(Evas_Object *parent): an API that creates an image object.

Eina_Bool elm_image_file_set(Evas_Object *obj, const char *file, const char *group): an API that, by specifying the path of an image file for an Image object, loads the image.

void elm_image_aspect_fixed_set(Evas_Object *obj, Eina_Bool fixed): an API that sets the image cropping property of an Image object. Passing EINA_TRUE to the second parameter lets the size of the image be reduced without any area cropped. Passing EINA_FALSE causes a cropped area in the image so that the image can fill an image in the whole item area. The default setting is EINA_TRUE.

18. Creating a Simple Web Browser Using the WebView Widget

To display web pages on the screen, you need to use the WebView widget. For a WebView widget to be created, it is necessary to specify an Evas object as its parent. Evas is the canvas used in EFL. In this example, we are going to create a simple web browser example using the WebView widget.

1) Creating a WebView Widget & Displaying a Web Page

Create a new source project and specify the project name as 'WebViewEx.'

After the source project is created, open the source file (~.c) under the src folder and add new code at the top. This code declares a library header file and adds variables to the appdata structure.

```
#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;
```

Add a new function on top of the create_base_gui() function. This function adds a widget to a Table container.

```
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);
}
```

Then, go to the create_base_gui() function and add new code. This code creates one Box container, one Table container, one Entry widget, three Button widgets, and one WebView widget. Conformant and Label are not necessary in this example, so we will annotate them.

```
    /* 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, "Hello EFL");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_E
XPAND);
    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_EX
PAND);
        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_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_E
XPAND);
        /* 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);
```

We annotated Conformant and Label and created one Entry widget, three Button widgets, and one WebView widget.

Passing ELM_WIN_INDICATOR_SHOW to the second parameter of the elm_win_indicator_mode_set() function displays an indicator at the top of the screen, while passing ELM_WIN_INDICATOR_HIDE hides the indicator. Hiding the indicator is recommended for maximum use of space.

evas_object_evas_get() is an API that creates an Evas object. Evas is a canvas on which images or shapes can be drawn.

ewk_view_add() is an API that creates a WebView widget. For a WebView widget to be created, it is necessary to specify an Evas object as its parent.

ewk_view_url_set() is an API that specifies a URL path for a WebView widget. For the first parameter, specify the WebView widget object, and for the second parameter, specify a URL path. We passed the caption text of the Entry widget: http://www.tizen.org is the Tizen developer support website.

Because we created three Buttons, three callback functions are needed. Add three new functions on top of the create_base_gui() function. We will define the content of the functions later.

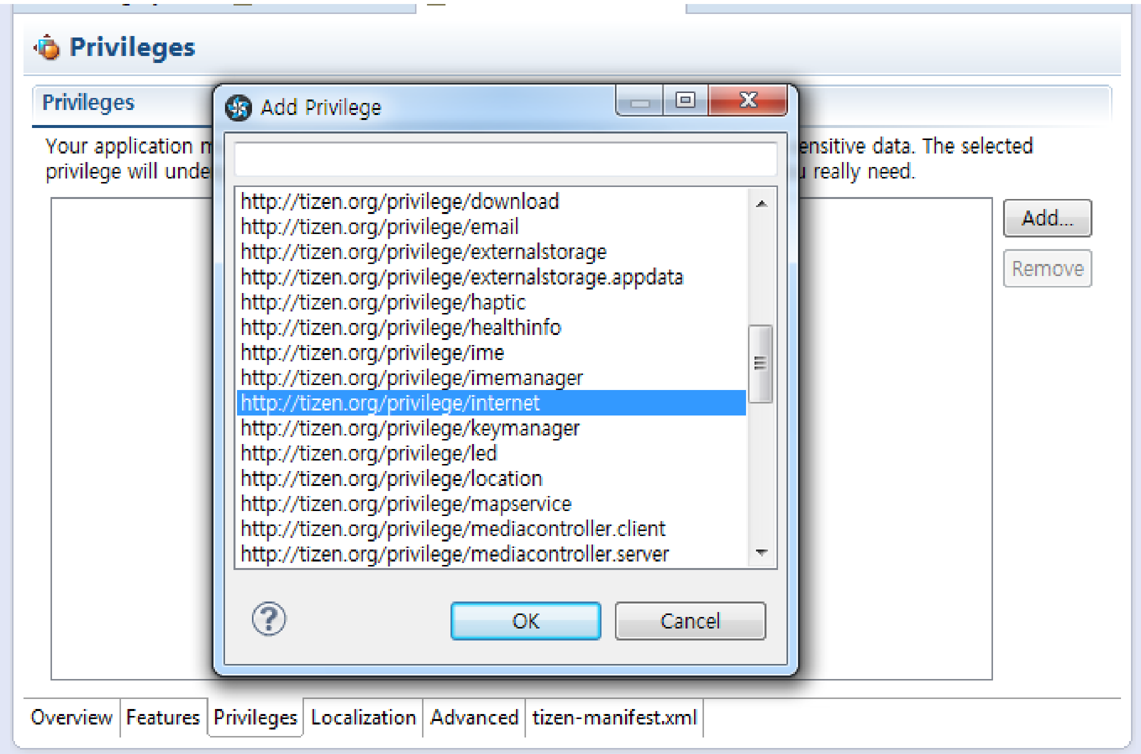
```
static void
btn_go_cb(void *data, Evas_Object *obj, void *event_info)
{
}
```

```
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)
{
}
```

If you run the example in this status, web pages will not be displayed. This is because it is necessary to support user privileges to enable the use of a network.

Open the tizen-manifest.xml file under the root folder of the source project by double-clicking the file. You can see a number of Tab buttons at the bottom. Select Privileges from among them.

Then, click the Add button and, when a popup window appears, select http://tizen.org/privilege/internet from the list and click the OK button.



Privileges

Privileges

Your application n... ensitive data. The selected privilege will unde... u really need.

Add Privilege

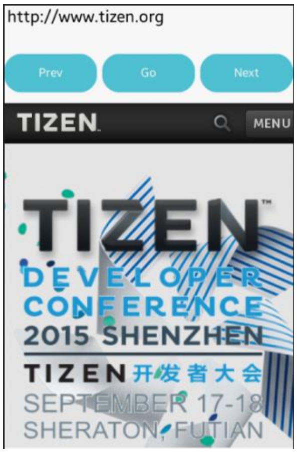
http://tizen.org/privilege/download
http://tizen.org/privilege/email
http://tizen.org/privilege/externalstorage
http://tizen.org/privilege/externalstorage.appdata
http://tizen.org/privilege/haptic
http://tizen.org/privilege/healthinfo
http://tizen.org/privilege/ime
http://tizen.org/privilege/imemanager
http://tizen.org/privilege/internet
http://tizen.org/privilege/keymanager
http://tizen.org/privilege/led
http://tizen.org/privilege/location
http://tizen.org/privilege/mapservice
http://tizen.org/privilege/mediacontroller.client
http://tizen.org/privilege/mediacontroller.server

Selecting tizen-manifest.xml from among the Tab buttons at the bottom displays source code.

```
<?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" no
display="false" taskmanage="true" type="capp">
        <label>webviewex</label>
        <icon>webviewex.png</icon>
    </ui-application>
    <privileges>
        <privilege>http://tizen.org/privilege/internet</privilege>
    </privileges>
```

```
</manifest>
```

Now, this example is capable of network communications. Build and run the example. You will see the Tizen developer support website.


http://www.tizen.org

2) Going to a Desired Website

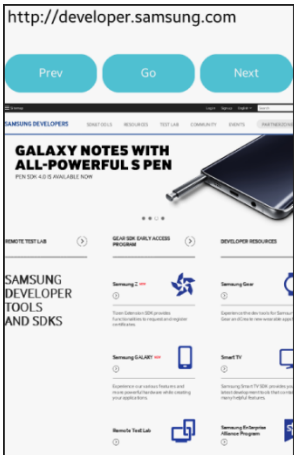
In this section, we are going to implement a feature that, when the user enters a URL address in an Entry widget and taps the Go button, directs the user to the URL address. Add new code to the Go button callback function.

```
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) );
}
```

User data is passed to the first parameter of the callback function.

The newly added code specifies the text entered in the Entry widget as a URL address for the WebView widget.

Run the example again. Enter the address of your desired website in the Entry and tap the Go button. The web page is displayed in the WebView.



http://developer.samsung.com

3) Going to Prev/Next Pages

In this section, we are going to implement a feature that directs the user to the previous page when the Prev button is tapped and directs the user to the next page when the Next button is tapped.

Add new code to the btn_prev_cb() function.

```
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() is an API that determines whether it is possible to go back to the previous screen.

ewk_view_back() is an API that moves the app to the previous screen.

Then, add new code to the btn_next_cb() function.

```
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 );
}
```

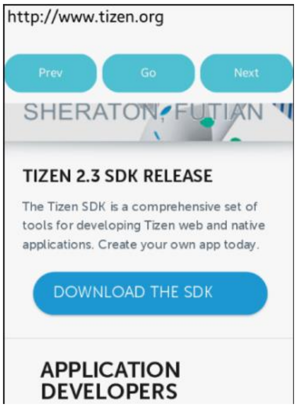
ewk_view_forward_possible() is an API that determines whether it is possible to go to the next screen.

ewk_view_forward() is an API that moves the app to the next screen.

Run the example again. Slide up the web page, and the link 'DOWNLOAD THE SDK' will appear. Click the link. If this link is not shown, click any other link shown on the screen.

When a new page is shown, click the Prev button. You are switched to the original screen.

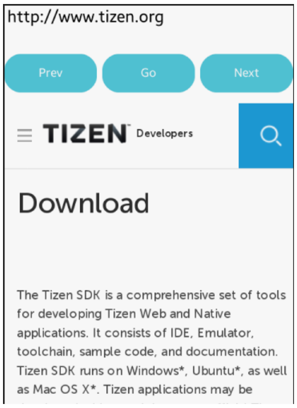
This time, click the Next Button. The Download screen appears again.



TIZEN 2.3 SDK RELEASE

The Tizen SDK is a comprehensive set of tools for developing Tizen web and native applications. Create your own app today.

APPLICATION DEVELOPERS



Download

The Tizen SDK is a comprehensive set of tools for developing Tizen Web and Native applications. It consists of IDE, Emulator, toolchain, sample code, and documentation. Tizen SDK runs on Windows*, Ubuntu*, as well as Mac OS X*. Tizen applications may be

5) Related APIs

Evas *evas_object_evas_get(const Evas_Object *obj): an API that creates an Evas object. Evas is a canvas on which images or shapes can be drawn.

Evas_Object* ewk_view_add(Evas* e): an API that creates a WebView widget. For a WebView widget to be created, it is necessary to specify an Evas object as its parent.

Eina_Bool ewk_view_url_set(Evas_Object* o, const char* url): an API that specifies a URL path for a WebView widget. parameters: WebView widget object and URL path.

Eina_Bool ewk_view_back_possible(Evas_Object* o): an API that determines whether it is possible to go back to the previous screen.

Eina_Bool ewk_view_back(Evas_Object* o): an API that moves to the previous screen.

Eina_Bool ewk_view_forward_possible(Evas_Object* o): is an API that determines whether it is possible to go to the next screen.

Eina_Bool ewk_view_forward(Evas_Object* o): an API that moves to the next screen.

19. Implementing a Tab Screen with a Layout Container

Using the Tab screen, you can switch between screens easily. In this example, we are going to learn how to implement a Tab screen using the Layout and Box containers.

1) Creating a Layout Container & Placing Widgets

Create a new source project and specify the project name as 'LayoutEx.'

After the source project is created, open the source file (~.c) under the src folder and add new code at the top. This code adds variables to the appdata structure.

```
typedef struct appdata {
    Evas_Object *win;
    Evas_Object *conform;
    Evas_Object *label;
    Evas_Object *box1;
    Evas_Object *box2;
} appdata_s;
```

In this example, we are going to implement the Tab screen. box1 becomes a first Tab screen, and box2 becomes a second Tab screen.

Add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
          /* 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);
}
```

Then, go back to the create_base_gui() function and create a Box container, a Table container, and Button widgets. Annotate Conformant and Label.

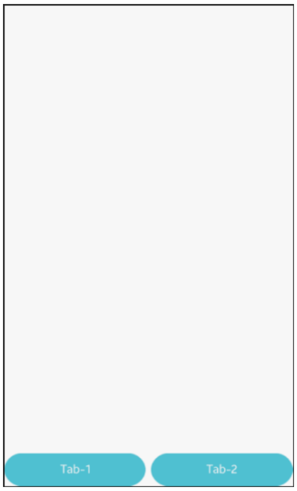
```
    /* Conformant */
    /*ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_win_resize_object_add(ad->win,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

    /* Label*/
    /*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);*/

    {
        /* 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_EXPAND);

        /* 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);
```

We created a Table and added two Buttons to the Table. We used a Box to specify the distance between widgets.

We are now going to create callback functions for the Buttons. Add two new functions on top of the create_base_gui() function. We will define the content of the functions later.

```
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)
{
}
```

Build and run the example. You can see two Buttons at the bottom of the screen.



2) Creating a Tab Screen

We are now going to create two Tab screens. Add new code to the create_base_gui() function. This code creates two Layouts and two Boxes and adds widgets on top of them.

```
/* 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);
```

We created two Layouts and added a Box on top of each Layout. The Layouts will act as Tab screens. We added a Label to the first Tab screen, and we added Buttons to the second Tab screen.

elm_layout_add() is an API that creates a Layout container.

elm_layout_theme_set() is an API that specifies a theme style for a Layout. To place a Window in a Layout, you need to specify the theme style of the Layout as 'panel.' To do so, pass the following parameter values:

elm_layout_theme_set(layout, "layout", "drawer", "panel") evas_object_hide() is an API that hides a Window. It performs the opposite function of evas_object_show().

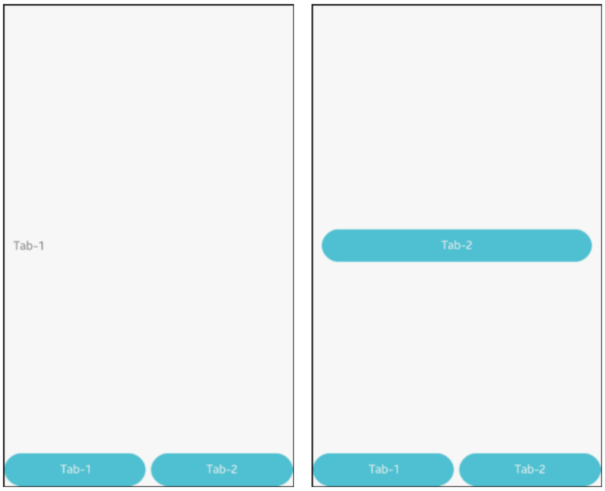
We are now going to implement a feature that, when one of the two Buttons at the bottom of the screen, displays the matching Tab screen and hides the other. Add new code to the Button callback function.

```
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);
}
```

Tapping the first Button displays the first Layout and hides the second Layout.

Tapping the second Button displays the second Layout and hides the first Layout.

Run the example again. Tapping a Button at the bottom of the screen displays the matching Tab screen.



4) Related APIs

Evas_Object *elm_layout_add(Evas_Object *parent): an API that creates a Layout container.

Eina_Bool elm_layout_theme_set(Evas_Object *obj, const char *clas, const char *group, const char *style): an API that specifies a theme style for a Layout. To place a Window in a Layout, you need to specify the theme style of the Layout as 'panel.'

void evas_object_hide(Evas_Object *obj): an API that hides a Window. It performs the opposite function of evas_object_show().

20. Implementing a Header and a Navigation Bar with a Naviframe Widget

To implement the Header and Footer (Toolbar), you can use the Naviframe widget. In this example, we are going to learn how to display title text in a header and display a navigation bar in a footer.

1) Creating a Header

Create a new source project and specify the project name as 'NaviframeEx.' After the source project is created, open the source file (~.c) under the src folder and add new code at the top. This code adds variables to the appdata structure.

```
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 is the object of a Naviframe, and layout is the main container that will be added to a Naviframe.

Elm_Object_Item is the item structure of a Naviframe.

btn1-btn3 are Buttons that will be added to a navigation bar.

Add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
        /* 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);
}
```

Then, go to the create_base_gui() function and add new code. This code creates a Naviframe, a Layout, and a Naviframe item.

```
    /* 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_EX
PAND);
        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", NULL, NULL, box, NULL);

            /* Toolbar */
            ad->toolbar = toolbar_add(ad, ad->nf);
            elm_object_item_part_content_set(ad->frame_item, "toolbar", ad->toolbar);
        }
    }

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

We added a Naviframe on top of a Conformant, and we added a Box on top of the Naviframe.

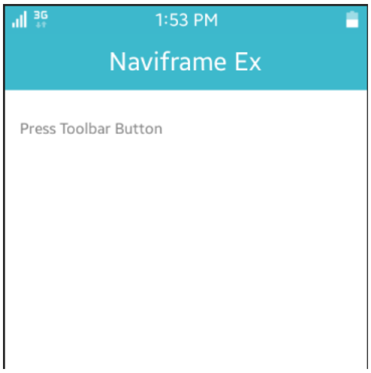
elm_naviframe_add() is an API that creates a Naviframe object.

evas_object_size_hint_weight_set() is an API that specifies a hint for the size of an object. The parameters listed in order are: object, horizontal size hint, and vertical size hint. EVAS_HINT_EXPAND is an option that specifies the size of an object as large as possible.

evas_object_size_hint_align_set() specifies a hint for the alignment of an object. The parameters listed in order are: object, horizontal alignment hint, and vertical alignment hint. EVAS_HINT_FILL is an option that lets an object fill as much space as possible in a given area.

elm_naviframe_item_push() is an API that creates a Naviframe item. The parameters listed in order are: Naviframe object, title text, the object of the Button for moving to the previous item, the object of the Button for moving to the next item, content, and item style.

Build and run the example. Title text is displayed in the Header.



2) Displaying a Toolbar in the Footer

In this section, we are going to add a Footer to a Naviframe item and create three Tab buttons.

Add a new function on top of the create_base_gui() function. This function creates and then returns a 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, ad);
    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() is an API that creates a Toolbar object.

elm_toolbar_item_append() is an API that adds items to a Toolbar. The parameters listed in order are: Toolbar object, icon image, caption text, name of the Button click event callback function, and user data.

Now, let's create callback functions for the Tab buttons. Add three new functions on top of the toolbar_add() function. This code changes the text in a Label widget when the user taps a Tab button.

```
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");
}
```

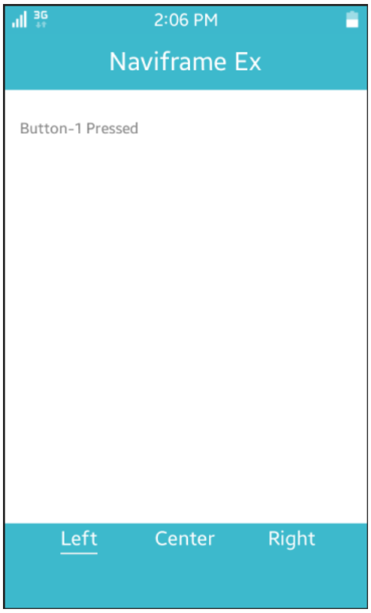
Now, we need to call the elm_toolbar_add() function to create a Toolbar. Add new code at the end of the create_base_gui() function.

```
            /* Header */
            ad->frame_item = elm_naviframe_item_push(ad->nf, "Naviframe Ex", NULL, N
ULL, box, NULL);

            /* Toolbar */
            ad->toolbar = toolbar_add(ad, ad->nf);
            elm_object_item_part_content_set(ad->frame_item, "toolbar", ad->toolbar);
      }
}
```

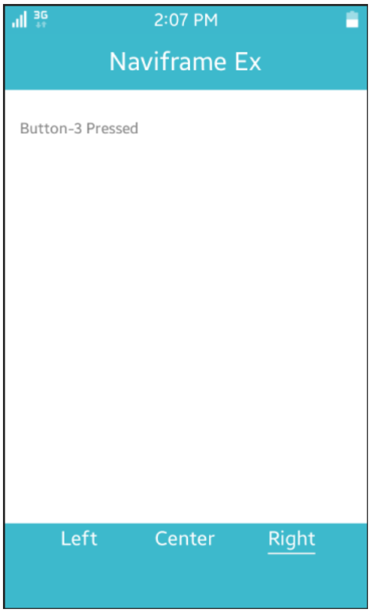
We created a Toolbar by calling the toolbar_add() function. Next, specify the Toolbar as the content of the Naviframe item.

elm_object_item_part_content_set() is an API that specifies content for an object item. The parameters listed in order are: object item, content part name, and content object.

Run the example again. You will now see a Footer at the bottom of the screen and three Buttons in the Footer. Clicking a Button changes the text in the Label widget. A checkmark appears below the selected Button.



Button-1 Pressed



Button-3 Pressed

3) Changing the Properties of the Toolbar

The appearance of the screen is not good with the text placed at the top. So, we are going to move text to the center and remove the checkmark.

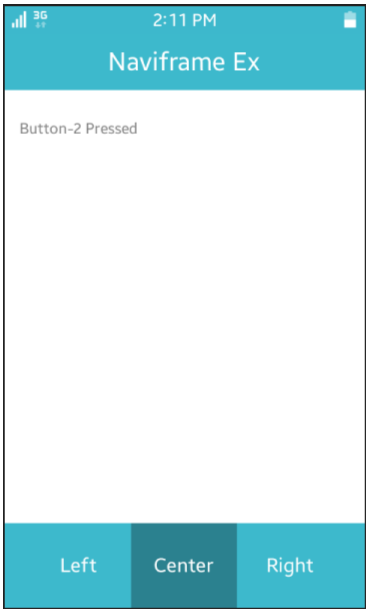
Add two new lines of code to the toolbar_add() function.

```
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, ad);
    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_select_mode_set() is an API that specifies the select mode of a Toolbar. Passing ELM_OBJECT_SELECT_MODE_NONE to the second parameter removes the checkmark. The default setting is ELM_OBJECT_SELECT_MODE_DEFAULT.

elm_toolbar_transverse_expanded_set() is an API that specifies whether to expand the size of a Toolbar. Entering EINA_TRUE to the second parameter lets the Toolbar fill the entire area of the Footer, and as a result, text is displayed at the center.

Run the example again. The text of the Toolbar is displayed at the center, and tapping a Tab button does not display the checkmark.



4) Related APIs

Elm_Object_Item: the structure of an object item or a Naviframe item.

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): an API that creates a Naviframe object. / parameters: Naviframe object, title text, the object of the Button for moving to the previous item, the object of the Button for moving to the next item, content, and item style.

void evas_object_size_hint_weight_set(Evas_Object *obj, double x, double y): an API that specifies a hint for the rough size of an object. /

parameters: object, horizontal size hint, and vertical size hint. EVAS_HINT_EXPAND is an option that specifies the size of an object to be as large as possible.

void evas_object_size_hint_align_set(Evas_Object *obj, double x, double y): an API that specifies a hint for the alignment of an object. / parameters: object, horizontal alignment hint, and vertical alignment hint. EVAS_HINT_FILL is an option that lets an object fill as much space as possible in a given area.

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): an API that creates a Naviframe item. / parameters: Naviframe object, title text, the object of the Button for moving to the previous item, the object of the Button for moving to the next item, content, and item style.

Evas_Object *elm_toolbar_add(Evas_Object *parent): an API that creates a Toolbar.

Elm_Object_Item *elm_toolbar_item_append(Evas_Object *obj, char *icon, char *label, Evas_Smart_Cb func, void *data): an API that adds items to a Toolbar. / parameters: Toolbar object, icon image, caption text, name of the Button click event callback function, and user data.

void elm_object_item_part_content_set(Elm_Object_Item *it, char *part, Evas_Object *content): an API that specifies content for an object item. / parameters: object item, content part name, and content object.

void elm_toolbar_select_mode_set(Evas_Object *obj, Elm_Object_Select_Mode mode): an API that specifies the select mode of a Toolbar. / parameters: Toolbar object and select mode. The default select mode is ELM_OBJECT_SELECT_MODE_DEFAULT. Passing ELM_OBJECT_SELECT_MODE_NONE hides the checkmark.

void elm_toolbar_transverse_expanded_set(Evas_Object *obj, Eina_Bool transverse_expanded): an API that specifies whether to expand the size of a Toolbar. / parameters: Toolbar object and whether to expand the size of the Toolbar. Entering EINA_TRUE lets the Toolbar fill the entire area of the Footer, and as a result, text is displayed at the center.

21. Placing Widgets in Sequence with the Box Container

To enable support of various terminals with different resolutions, you can use relative coordinates when placing widgets. The Table container lets you specify coordinates for a widget based on the aspect ratio. On the other hand, the Box container is used to place widgets in sequence or place widgets using relative coordinates (left, right, and center, or top, bottom, and middle). This feature is similar to Android's LinearLayout. In this example, we are going to learn how to place widgets horizontally and vertically using the Box container.

1) Creating a Horizontal Box

In this example, three Box containers and six Button widgets will be created. To make the source code simple, we are first going to create a function that creates Boxes and a function that creates Buttons.

Create a new source project and specify the project name as 'BoxEx.' After the source project is created, open the source file (~.c) under the src folder and add two new functions on top of the create_base_gui() function.

```
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() is a function that creates and then returns a Box container.

elm_box_add() is an API that creates a Box container.

create_button() is a function that creates a Button widget, specifies caption text to it, and then returns the widget.

We are now going to create two Boxes and three Buttons using the functions we just created and place them horizontally. Go back to the create_base_gui() function and add new code. Label will not be used in this example, so delete it.

```
     /* 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_EX
PAND);
        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() is an API that specifies the paddings of a Box container. The parameters listed in order are: container object, left & right paddings, and top & bottom paddings.

evas_object_size_hint_weight_set() is an API that specifies a hint for the size of the space an object occupies. The parameters listed in order are: object, horizontal size hint, and vertical size hint. EVAS_HINT_EXPAND is an option that specifies the size of an object as large as possible.

evas_object_size_hint_align_set() is an option that specifies a hint for the size of an object itself. The parameters listed in order are: object, horizontal size hint, and vertical size hint. EVAS_HINT_FILL is an option that specifies the size of an object as large as possible.

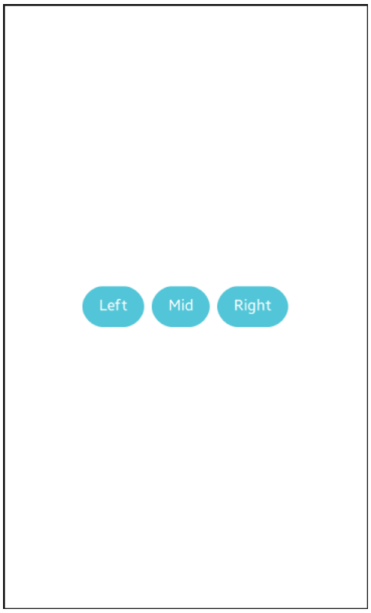
elm_win_resize_object_add() is an API that specifies the size of an object as the same size as another object. We passed 'win' to the first parameter and 'main_box' to the second parameter. Doing so resizes the main_box so that it fills the whole area of the screen.

elm_box_horizontal_set() is an API that specifies the alignment orientation of a Box. Passing EINA_TRUE to the second parameter sets the alignment to horizontal while passing EINA_FALSE sets the alignment to vertical. The default setting is vertical.

elm_box_pack_end() is an API that adds a new object to a Box container.

If the alignment is horizontal, a new object will be added at the right side of the screen. If the alignment is vertical, a new object will be added at the bottom of the screen.

Build and run the example. Three Buttons are horizontally aligned at the center of the screen. Despite having specified the size of the second Button as maximum, the second Button is occupying the smallest possible space. This is because we did not specify the width of the Box container.



2) Maximizing the Size of an Object

In this section, we are going to learn how to specify the width of a Box as maximum. Add new code to the create_base_gui() function's code that creates a second Button.

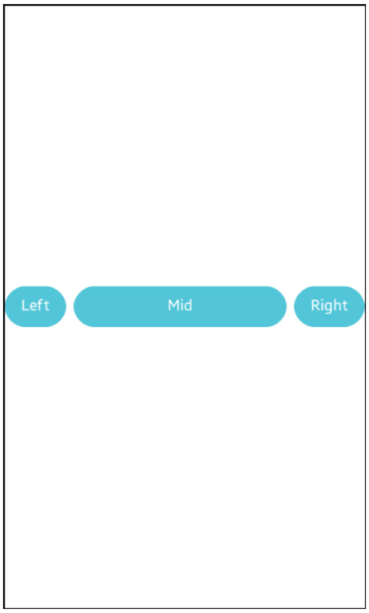
```
/* Create horizontal box */
Evas_Object *horizontal_box = create_box(ad->win);
elm_box_horizontal_set(horizontal_box, EINA_TRUE);
elm_box_padding_set(horizontal_box, 10, 10);
evas_object_size_hint_weight_set(horizontal_box, EVAS_HINT_EXPAND, 0.0);
evas_object_size_hint_align_set(horizontal_box, EVAS_HINT_FILL, 0.0);
elm_box_pack_end(main_box, horizontal_box);
```

We passed EVAS_HINT_EXPAND to the second parameter of the evas_object_size_hint_weight_set() function so that the width of the Button is maximized. We specified 0.0 for the third parameter so that the height of the Button is minimized.

However, this action alone will not change the width of the Button. evas_object_size_hint_weight_set() is a function that procures space, and evas_object_size_hint_align_set() is a function that specifies the alignment of content.

Enter a horizontal alignment type for the second parameter of the evas_object_size_hint_align_set() function: 0.0 indicates left alignment; 0.5 indicates center alignment; 1.0 indicates right alignment; and passing EVAS_HINT_FILL specifies the width of the content as maximum.
Enter a vertical alignment value type for the third parameter. 0.0 indicates top alignment; 0.5 indicates center alignment; 1.0 indicates bottom alignment; and passing EVAS_HINT_FILL specifies the height of the content as maximum.

Run the example again. You will now see the width of the second Button expanded.



3) Changing the Position of a Box

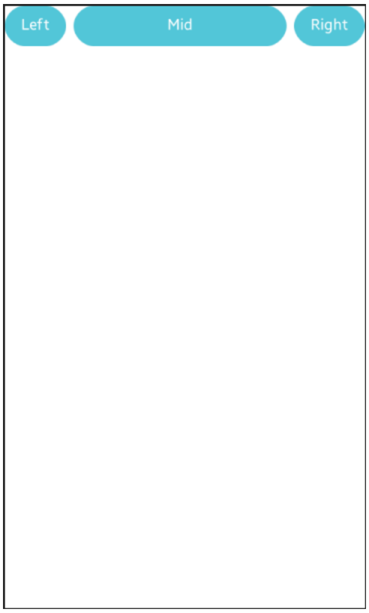
Because the second Box is placed at the center of the screen, the Buttons are also placed at the center of the screen. The Box container places widgets in sequence. Therefore, adding a new Box to the first Box moves the second Box upward. Add new code at the bottom of the create_base_gui() function. This code creates a third Box and adds it to the first 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_HI
NT_FILL);
            elm_box_pack_end(box, vertical_box);
        }
    }
```

The alignment of the first Box is vertical, and therefore the newly added Box will be placed below the second Box.

We procured the maximum possible space by passing EVAS_HINT_EXPAND to the evas_object_size_hint_weight_set() function, and we also specified the size of the content as maximum by passing EVAS_HINT_FILL to the evas_object_size_hint_align_set() function.

Run the example again, and you will now see the Buttons have moved to the top of the screen.



4) Aligning Widgets Vertically

We are now going to add three Buttons to the third Box. Because we did not specify the alignment of the Buttons, the Buttons will be aligned vertically. Add new code at the bottom of the create_base_gui() function.

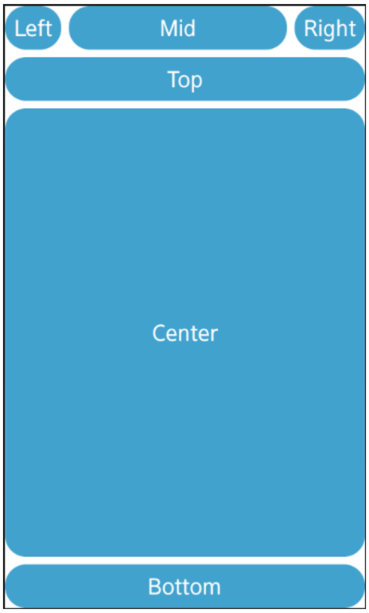
```
        /* Create vertical box */
```

```
            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, EVAS_
HINT_EXPAND);
            evas_object_size_hint_align_set(vertical_box, EVAS_HINT_FILL, EVAS_HINT_FI
LL);
            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_H
INT_EXPAND);
                evas_object_size_hint_align_set(btn, EVAS_HINT_FILL, EVAS_HINT_FI
LL);
                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);
            }
        }
    }
```

We specified the widths and heights of the first and third Buttons as minimum.

We specified the width and height of the second Button as maximum.

Run the example again. You will now see the newly added three Buttons aligned vertically.



5) Related APIs

Evas_Object *elm_box_add(Evas_Object *parent): an API that creates a Box container.

void elm_box_padding_set(Evas_Object *obj, Evas_Coord horizontal, Evas_Coord vertical): an API that specifies paddings. / parameters: container object, left & right paddings, and top & bottom paddings.

void evas_object_size_hint_weight_set(Evas_Object *obj, double x, double y): an API that specifies a hint for the size of the space an object occupies. / parameters: object, horizontal size hint, and vertical size hint. EVAS_HINT_EXPAND is an option that specifies the size of an object as large as possible.

void evas_object_size_hint_align_set(Evas_Object *obj, double x, double y): an API that specifies a hint for the size of an object. / parameters: object, horizontal size hint, and vertical size hint. EVAS_HINT_FILL is an option that specifies the size of an object as large as possible.

void elm_win_resize_object_add(Evas_Object *obj, Evas_Object *subobj): an API that specifies the size of an object as the same size as another object. / parameters: Window object and object whose size will be changed.

void elm_box_horizontal_set(Evas_Object *obj, Eina_Bool horizontal): an API that specifies the orientation of a Box. / parameters: object and orientation. Passing EINA_TRUE sets the orientation of the Box to horizontal while passing EINA_FALSE sets the orientation of the Box to vertical. The default setting is vertical.

void elm_box_pack_end(Evas_Object *obj, Evas_Object *subobj): an API that adds a new object to a Box container. If the alignment is horizontal, a new object will be added at the right side of the screen. If the alignment is vertical, a new object will be added at the bottom of the screen. / parameters: Box object and content object.

22. Creating a Sub Page with a Scroller Widget

When you develop a commercial app, you need to create multiple pages. You can use the following methods to implement multiple pages.
- With the use of a Toolbar and a number of layouts, making it so that the matching layout can be shown/hidden when a Tab button is tapped.
- Moving a layout with Scroller widget.

In this example, we are going to learn how to move between pages using the Scroller widget. We are also going to create a source file relevant to a second page.

1) UI Task for the Main Page

In this section, by using the Naviframe widget, we are going to display a header on the main page and add one Button to the main page.

Create a new source project and specify the project name as 'Multipage.' After the source project is created, open the source file (~.c) under the src folder and add a new variable to the appdata structure. This variable is for the Naviframe object that displays the title.

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

Add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAND);
        /* 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);
}
```

Then, go to the create_base_gui() function and modify the code as shown below. This code creates a Naviframe, Box, Label, Button, and 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_EXPAND);
    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_scale_get());
        evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
        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, NUL
L, box, NULL);
            eext_object_event_callback_add(ad->nf, EEXT_CALLBACK_BACK, eext_n
aviframe_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);
```

The code below terminates the app when the Back button is tapped. The problem is that the app also gets terminated even when the Back button is tapped on the sub page. For this reason, we will annotate this code for now.

```
eext_object_event_callback_add(ad->win, EEXT_CALLBACK_BACK,
win_back_cb, ad);
```

The code below moves the app back to the main page when the Back button is tapped on the sub page.

```
eext_object_event_callback_add(ad->nf, EEXT_CALLBACK_BACK,
eext_naviframe_back_cb, NULL);
```

We also need a feature that terminates the app when the Back button is tapped on the main page. This is because we annotated the code that calls the win_back_cb callback function. For this reason, we will add the code shown below. The naviframe_pop_cb function is called when the Back button is tapped on the main page.

```
elm_naviframe_item_pop_cb_set(nf_it, naviframe_pop_cb, ad->win);
```

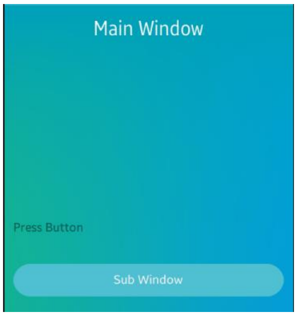
Now, let's create the naviframe_pop_cb callback function. Add a new function on top of the create_base_gui() function.

```
static Eina_Bool
naviframe_pop_cb(void *data, Elm_Object_Item *it)
{
    ui_app_exit();
    return EINA_FALSE;
}
```

The function above is called when the Back button is tapped on the main page.

ui_app_exit() is an function that terminates the app.

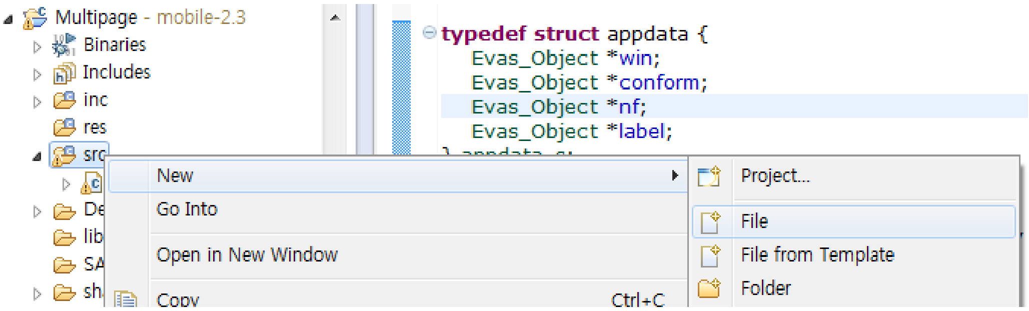
Build and run the example. You can see a Header, Label, and Button.



2) Creating a Sub Page

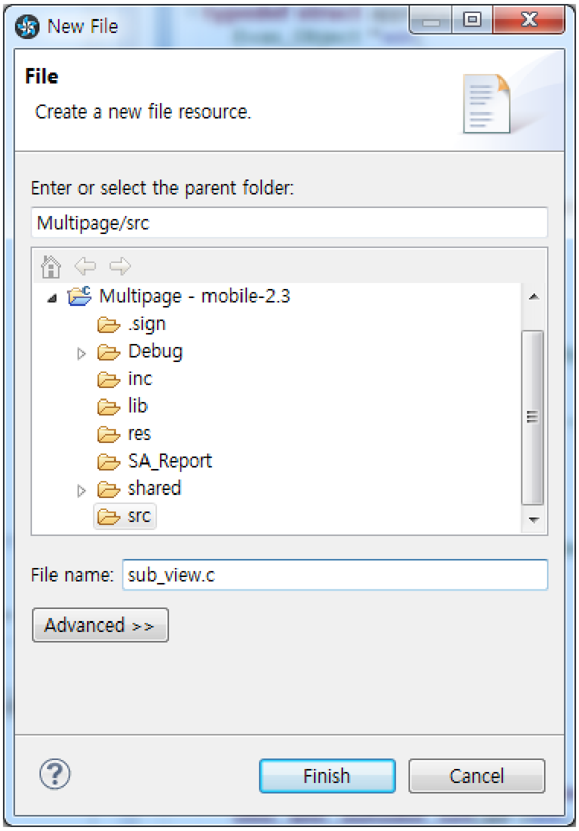
In this section, we are going to implement a feature that makes the sub page appear when a Button is tapped. It does not matter if you add sub page-creating code to the main source file (multipage.c). However, as the code gets longer, it becomes more difficult to manage. In addition, using multiple source files also makes it more convenient to use the features when you develop a different app later.

We are now going to create a new source file. Right-click the /src folder and select [New > File] in the shortcut menu.



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

When a popup window appears, enter 'sub_view.c' in the file name field and click the Finish button. A new source file will then be created under the /src folder.


New File

File

Create a new file resource.

Enter or select the parent folder:

Multipage/src

File name: sub_view.c

Double-click the newly created source file (sub_view.c) and open it in Edit mode. We are now going to add code that creates a sub page and icon Button.

```
#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_EXPAND);
    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);
}
```

Because the sub page needs to be linked to the first page, we will declare the main header file (multipage.h).

create_button_view() is a function that creates a Box container and adds a Button widget on top of the Box container.

sub_view_cb() is a function that displays a second page by creating a Scroller and Layout.

elm_scroller_add() is an API that creates a new Scroller.

elm_object_content_set() is an API that specifies the content of a container. Passing a Scroller to the first parameter and passing a layout to the second parameter displays the layout on the screen.

Change the title text of the Header and specify the Scroller as the content of the Naviframe by using the elm_naviframe_item_push() function.

The sub_view_cb() function needs to be called when the Button on the first page is tapped. To that end, we must declare a function in the header file. Open the /inc folder, and double-click the 'multipage.h' file. When the file is open, declare the sub page creating function at the bottom of the file.

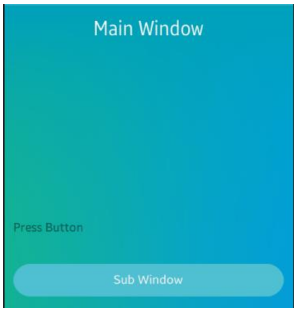
```
#if !defined(PACKAGE)
#define PACKAGE "org.tizen.multipage"
#endif

void sub_view_cb(void *data, Evas_Object *obj, void *event_info);

#endif /* __multipage_H__ */
```

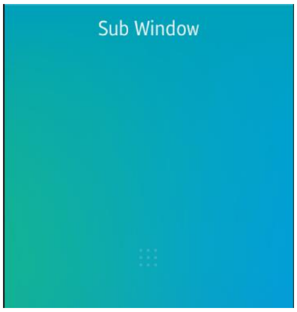
The function can now be called on the main page. Go to the Multipage.c file and add a new line of code to the Button-creating code of the create_base_gui() function. This new code calls the sub page-creating function when the Button on the main page is tapped.

```
            /* 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);
```

Run the example again. When you tap the Button on the screen, the sub page appears, and you can see an icon Button at the center of the screen. To go back to the main page, tap the Back button.





3) Moving to the Main Page with a Tap of a Button

In this section, we are going to implement a feature that moves the app to the main page when the Button on the sub page is tapped. To do so, we need to call the elm_naviframe_item_pop() function.

Go to the sub_view_cb() function of the sub_view.c file and modify the code. Pass the Naviframe to the create_button_view() function, and

```
 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);
```

```
}
```

then add new code to the create_button_view() function.

```
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_EXPAND);
    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);
    ~
```

The NaviFrame object is passed to the function's parameter. We specified the Button callback function as btn_back_cb and passed the Naviframe to user data.

Lastly, add the Button callback function on top of the create_button_view() function.

```
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() is the command for moving to the top of the list of pages accumulated in the stack of a Naviframe. In other words, it is the command for moving to the main page.

Run the example again, go to the sub page, and tap the icon Button. You are taken to the main page.

Tapping the Back button on the sub page directs you to the main page while tapping the Back button on the main page makes the app screen disappear. It does not mean that the app has been closed; it means the app has been switched to Background mode.

4) Related APIs

eext_object_event_callback_add(ad->win, EEXT_CALLBACK_BACK, win_back_cb, ad): code that terminates the app when the Back button is tapped. You must annotate this code in order for the app not to be terminated when the Back button is tapped on the sub page.

eext_object_event_callback_add(ad->nf, EEXT_CALLBACK_BACK, eext_naviframe_back_cb, NULL): code that moves the app back to the main page when the Back button is tapped on the sub page.

elm_naviframe_item_pop_cb_set(nf_it, naviframe_pop_cb, ad->win): code that specifies the callback function for when the Back button is tapped on the main page as naviframe_pop_cb.

ui_app_exit(): an API that terminates the app.

Evas_Object *elm_naviframe_add(Evas_Object *parent): an API that creates a new Scroller.

void elm_object_content_set(Evas_Object *obj, Evas_Object *content): an API that specifies content for a container. / parameters: container object and content object.

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): an API that specifies the Header title text and content of a Naviframe. / parameters: Naviframe object, title text, the object of the Button for moving to the previous item, the object of the Button for moving to the next item, content, and user data

Evas_Object *elm_naviframe_item_pop(Evas_Object *obj): an API for moving to the top of the list of pages accumulated in the stack of a Naviframe. / parameters: the Naviframe object.

23. Using Strings

String conversion or string search is a feature frequently used when developing an app. In this example, we are going to learn how to find the position of a string and convert strings into numbers by using basic APIs written in C.

1) Copying a String

Create a new source project and specify the project name as 'StringEx.' After the source project is created, add new code by opening the source file (~.c) under the src folder and going to the create_base_gui() function.

```
    /* 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() is an API that specifies automatic line wrapping for a Label widget.

show_string_result() is a function that displays the result of a string conversion. Now, let's start building the example.

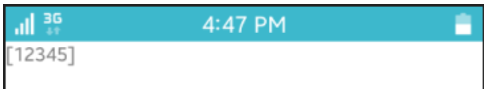
Add a new function on top of the create_base_gui() function.

```
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() is an API that copies a string. The first parameter indicates the address of the memory where the string gets stored, while the second parameter indicates data about the original string. In this case, we enter the text '12345' in a string variable called 'str1.'

sprintf() is an API that creates a new string of a specific format type. The first parameter indicates the address of the memory where the string get stored, the second parameter indicates the format type, and the third parameter and later parameters indicate data to be assigned to the format.

Build and run the example. We outputted the original string enclosed in square brackets.



```
[12345]
```

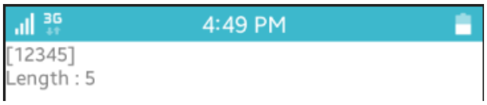
2) Requesting the Length of a String

In this section, we are going to request the length of a string stored in the str1 variable. Add new code at the end of the show_string_result() function.

```
 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() is an API that requests and then returns the length of a string. str1 is a char array, so it has a fixed length. In such cases, the length from the beginning of the string to before the End symbol (₩0) will be returned.

Run the example again. The length of the string is displayed.



```
[12345]
Length : 5
```

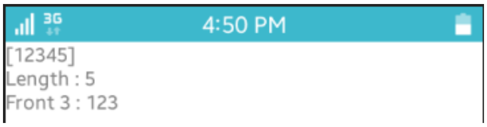
3) Extracting a Specific Length of Characters from the Beginning of a String

In this section, we are going to extract the first three characters from the str1 variable. Add new code at the end of the show_string_result() function.

```
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() is an API that extracts a specific length of characters from the beginning of a string. The first parameter indicates data about the original string, while the second parameter indicates the length to be extracted. The extracted string will be returned.

Run the example again. The characters '123' are extracted from the beginning of the string '12345.'



```
[12345]
Length : 5
Front 3 : 123
```

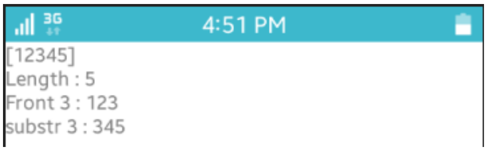
4) Extracting a Specific Part of a String

In this section, we are going to learn how to extract a certain length of characters from a string by specifying the starting point of extraction. Add new code at the end of the show_string_result() function.

```
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);
```

We passed 'str1 + 2' to the first parameter of the eina_stringshare_add_length() function. Doing so eventually passes the characters '345.'

Run the example again. Three characters are extracted from the string '12345,' beginning from the third character position.



```
[12345]
Length : 5
Front 3 : 123
substr 3 : 345
```

5) Converting a String into a Numeric Value

To convert a string into a number, you can use the atoi() function. Add new code at the end of the show_string_result() function.

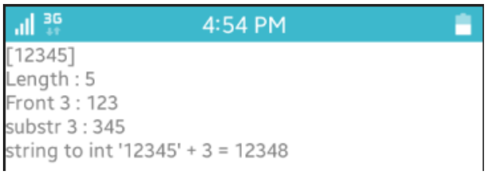
```
sprintf(buf, "%s<br>substr 3 : %s", buf, str2);

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() is an API that converts a string into the int type.

atol() is an API that converts a string into the long type.

atof is an API that coverts a string into the float type.

Run the example again. The string '12345' is converted into a number, and the result of adding 3 to the number is displayed on the screen.



```
[12345]
Length : 5
Front 3 : 123
substr 3 : 345
string to int '12345' + 3 = 12348
```

6) Converting a Number into a String

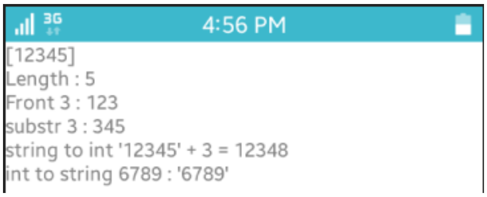
To convert a number into a string, you can use the sprintf() function. Add new code at the end of the show_string_result() function.

```
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);
```

The following are symbols commonly used in string-type format statements:

- %s: replace with a string.
- %d: replace with a number such as an int or a long.
- %c: replace with one character.
- %f: replace with a real number such as a float or a double.

Run the example again. The number '6789' is converted into a string.



```
[12345]
Length : 5
Front 3 : 123
substr 3 : 345
string to int '12345' + 3 = 12348
int to string 6789 : '6789'
```

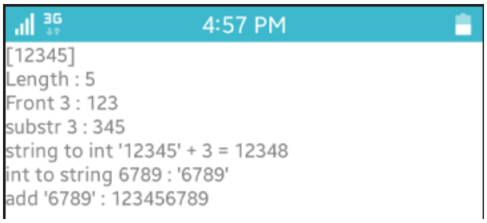
7) Combining Strings

In this section, we are going to learn how to combine two strings into one. Add new code at the end of the show_string_result() function.

```
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() is a function that combines two strings. This function appends a second string to the end of the first parameter. It is not that a new string is created but that a new string is appended to an existing string.

Run the example again. '6789' is added to '12345' to make '123456789.'



```
[12345]
Length : 5
Front 3 : 123
substr 3 : 345
string to int '12345' + 3 = 12348
int to string 6789 : '6789'
add '6789' : 123456789
```

8) Finding the Position of a String

In this section, we are going to learn how to find the position of a string. Add new code at the end of the show_string_result() function.

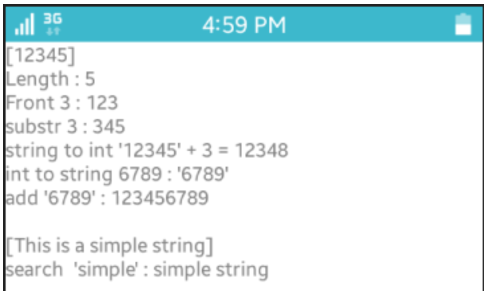
```
sprintf(buf, "%s<br>add '%s' : %s", buf, str3, str1);

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' is the End symbol that indicates the end of a string. Entering this symbol in the first character of a char array initializes the string as a blank.

strstr() is an API that finds the position of a certain string contained in a string. This function does not return the index number; it returns the pointer where the string starts. In other words, it returns a string.

To find the starting position of a certain char in a string, you can use the strchr(char*, int) function.

Run the example again. The string that comes after the starting position of the string 'simple' is displayed.



```
[12345]
Length : 5
Front 3 : 123
substr 3 : 345
string to int '12345' + 3 = 12348
int to string 6789 : '6789'
add '6789' : 123456789

[This is a simple string]
search  'simple' : simple string
```

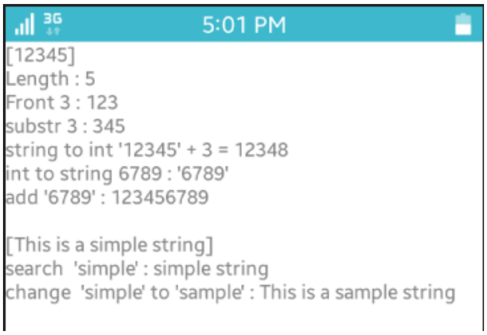
9) Copying a Specific Length of a String

In this section, we are going to change the string 'simple' into 'sample.' To do so, we need to use the strncpy() function. Add new code at the end of the show_string_result() function.

```
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() is a function that copies a specific length of a string. str2 indicates the position where the string 'simple' starts. Therefore, 'simple' is eventually changed into 'sample.'

Run the example again. You will now see 'simple' is changed into 'sample.'



```
[12345]
Length : 5
Front 3 : 123
substr 3 : 345
string to int '12345' + 3 = 12348
int to string 6789 : '6789'
add '6789' : 123456789

[This is a simple string]
search  'simple' : simple string
change  'simple' to 'sample' : This is a sample string
```

10) Related APIs

char *strcpy (char *dest, char *src): an API that copies a string. / parameters: address of the memory where the string gets stored and data about the original string.

int sprintf (char *s, char *format, ...) : an API that creates a new string of a specific format type. / parameters: the address of the memory where the string get stored, format type, and, from the third parameter on, data to be assigned to the format.

size_t strlen (char *s): an API that requests and then returns the length of a string. In string type format statements, %s is a symbol replaced by a string. %d is a symbol replaced by a number such as int or long. %c is a symbol replaced by a character. %f is a symbol replaced by a real number such as float or double.

Eina_Stringshare *eina_stringshare_add_length(const char *str, unsigned int slen): an API that extracts a specific length of characters from the beginning of a string. / parameters: data about the original string and the length to be extracted. / return: extracted string.

int atoi (char *nptr): an API that converts a string into the int type.

long int atol (char *nptr): an API that converts a string into the long type.

double atof (char *nptr): an API that converts a string into the float type.

char *strcat (char *__dest, char *__src): an API that combines two strings. This function appends a second string to the end of the first string. In this case, a new string is not created, but appended to an existing string.

char *strstr (char *haystack, char *needle): an API that finds the position of a certain string contained in a string. This function does not return the index number; it returns the pointer to where the string starts. Therefore, it returns a string.

char *strchr (char *s, int c): an API that searches the starting position of a certain character in a string.

char *strncpy (char *dest, char *src, size_t n): an API that copies a specific length of a string.

24. The String Structure 'Eina_Strbuf'

Sometimes you feel like you've hit a dead end while handling strings using APIs written in C. EFL provides a string structure called 'Eina_Strbuf.' In this example, we are going to learn how to delete part of a string, change a string into a different string, and insert a new string into the middle of an existing string.

1) Entering a String in Eina_Strbuf

Create a new source project and specify the project name as 'EinaStrbufEx.' After the source project is created, open the source file (~.c) under the src folder, move to the create_base_gui() function and add new code.

```
   /* 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_EXPAN
D);
   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() is an API that specifies automatic line wrapping for a Label widget.

show_eina_strbuf_result() is a function that displays the result of using Eina_Strbuf on the screen. Now, let's start building the example.

Create a new function on top of the create_base_gui() function.

```
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);
}
```

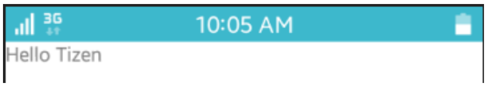
eina_strbuf_new() is an API that creates a new Eina_Strbuf object.

eina_strbuf_append() is an API that adds a string to Eina_Strbuf.

eina_strbuf_string_get() is an API that requests the string stored in Eina_Strbuf.

eina_strbuf_free() is an API that deletes the data stored in Eina_Strbuf.

Build and run the example.



Hello Tizen

2) Adding a String in a Specified Format

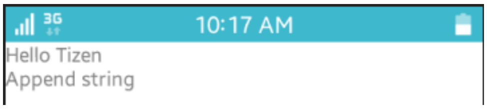
To add a new string in a specified format, you need to use the eina_strbuf_append_printf() function. Add new code in the middle of the show_eina_strbuf_result() function.

```
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(strline) );

elm_object_text_set(label, eina_strbuf_string_get(strbuf));
/* Free memory */
eina_strbuf_free(strline);
eina_strbuf_free(strbuf);
```

We created a new Eina_Strbuf object using the eina_strbuf_new() function and entered a string using the eina_strbuf_append() function.

eina_strbuf_append_printf() is an API that adds a new string by specifying a format statement. The first parameter indicates the Eina_Strbuf object, the second parameter indicates the format type, and the third parameter and later parameters indicate data to be assigned to the format.

Run the example again. Two lines of a string are displayed.



10:17 AM

Hello Tizen
Append string

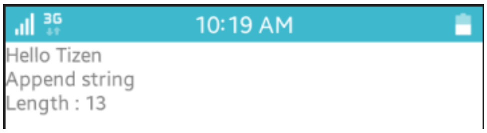
3) Requesting the Length of a String

To request the length of a string stored in Eina_Strbuf, you need to use the eina_strbuf_length_get() function. Add new code at the end of the show_eina_strbuf_result() function.

```
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() is an API that returns the length of a string stored in Eina_Strbuf.

Run the example again. The length of the string 'Append string,' 13, is displayed.



10:19 AM

Hello Tizen
Append string
Length : 13

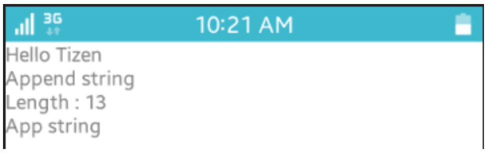
4) Deleting Part of a String

To delete part of a string, you need to use the eina_strbuf_remove() function. Add new code at the end of the show_eina_strbuf_result() function.

```
    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(strline) );

    elm_object_text_set(label, eina_strbuf_string_get(strbuf));
```

eina_strbuf_remove() is an API that removes only part of a string. The first parameter indicates the Eina_Strbuf object, the second parameter indicates the start position of removal, and the third parameter indicates the end position of removal. For example, entering ‘6’ deletes all characters up to the sixth.

Run the example again. Index numbers 3 to 5 (the fourth to sixth characters) are deleted.



```
Hello Tizen
Append string
Length : 13
App string
```

5) Replacing a String

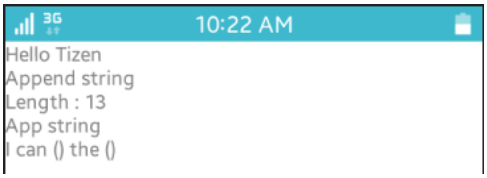
To replace a string with another string, you need to use the eina_strbuf_replace() function. Add new code at the end of the show_eina_strbuf_result() function.

```
eina_strbuf_append_printf(strbuf, "<br>%s", eina_strbuf_string_get(strline) );

/* 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(strline) );

elm_object_text_set(label, eina_strbuf_string_get(strbuf));
```

eina_strbuf_reset() is an API that resets the string stored in Eina_Strbuf.

eina_strbuf_replace() is a function that changes a string into a different string. The first parameter indicates the Eina_Strbuf object; the second parameter indicates the string to change; the third parameter indicates the string into which the existing string will be changed; and the fourth parameter indicates the number of times of change.

Run the example again. The first '()' is changed into 'can.'



```
Hello Tizen
Append string
Length : 13
App string
I can () the ()
```

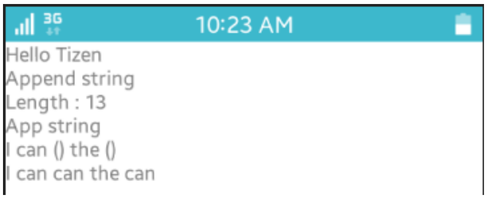
6) Replacing All Identical Strings

In this section, we are going to learn how to replace all identical strings. Add new code at the end of the show_eina_strbuf_result() function.

```
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() is an API that replaces all identical strings. The first parameter indicates the Eina_Strbuf object, the second parameter indicates the string to change, and the third parameter indicates the string into which the existing string will be changed.

Run the example again. All '()'s are changed into 'can.'



```
Hello Tizen
Append string
Length : 13
App string
I can () the ()
I can can the can
```

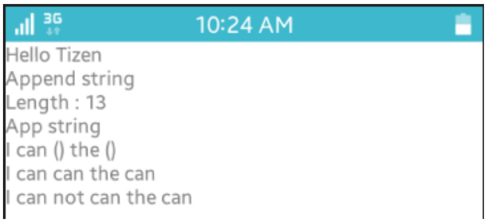
7) Inserting a String into the Middle of Another String

In this section, we are going to learn how to insert a string into the middle of another string. Add new code at the end of the show_eina_strbuf_result() function.

```
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() is an API that inserts a string into a specific position of another string. The first parameter indicates the Eina_Strbuf object, and the second parameter indicates the string to be inserted, and the third parameter indicates the insertion position.

Run the example again. The string 'not' is inserted into index number 5 (the sixth character).



```
Hello Tizen
Append string
Length : 13
App string
I can () the ()
I can can the can
I can not can the can
```

8) Related APIs

Eina_Strbuf *eina_strbuf_new(void): an API that creates a new Eina_Strbuf object.

Eina_Bool eina_strbuf_append(Eina_Strbuf *buf, char *str): an API that adds a string to Eina_Strbuf.

char *eina_strbuf_string_get(Eina_Strbuf *buf): an API that requests the string stored in Eina_Strbuf.

void eina_strbuf_free(Eina_Strbuf *buf): an API that deletes the data stored in Eina_Strbuf.

Eina_Bool eina_strbuf_append_printf(Eina_Strbuf *buf, char *fmt, ...) : an API that adds a new string by specifying a format statement. / parameters: Eina_Strbuf object, format type, and from the third parameter on, data to be assigned to the format.

size_t eina_strbuf_length_get(Eina_Strbuf *buf): an API that returns the string stored in Eina_Strbuf.

Eina_Bool eina_strbuf_remove(Eina_Strbuf *buf, size_t start, size_t end): an API that removes only part of a string. / parameters: Eina_Strbuf object, the start position of removal, and the end position of removal.

void eina_strbuf_reset(Eina_Strbuf *buf): an API that resets the string stored in Eina_Strbuf.

Eina_Bool eina_strbuf_replace(Eina_Strbuf *buf, char *str, char *with, unsigned int n): an API that changes a string into another string. / parameters: Eina_Strbuf object, the string to change, the string into which the existing string will be changed, and number of changed times.

int eina_strbuf_replace_all(Eina_Strbuf *buf, char *str, char *with): an API that replaces all identical strings. / parameters: Eina_Strbuf object, the string to change, and the string into which the existing string will be changed.

Eina_Bool eina_strbuf_insert(Eina_Strbuf *buf, char *str, size_t pos): an API that inserts a string into a particular position of another string. / parameters: Eina_Strbuf object, the string to be inserted, and insertion position.

25. Array Structure Eina_List

When you are developing apps, there are times when it is necessary to manage multiple strings or user data in an array. EFL provides an array structure called 'Eina_List.' We are now going to learn how to use it through an example.

1) Creating a List Widget

Create a new source project and specify the project name as 'EinaListEx.' After the source project is created, open the source file (~.c) under the src folder and add new code. Add new variables to the appdata structure, define the new structure, and store 10 strings in a char* array.

```
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 is the variable for the List widget.

button is the variable for the Delete button.

data_list is the variable for the array structure Eina_List.

Itemdata_s is a structure for the List Widget's event data. We are now going to enter app data in ad, enter the string selected by the user in data, and enter the number of the selected items in id.

Items[] stores 10 strings that will be added to the List widget.

Go to the create_base_gui() function and add new code. This code creates Frame, Table, Button, and List widgets.

```
    /* 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_E
XPAND);
        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_E
XPAND);
        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_HINT_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);
```

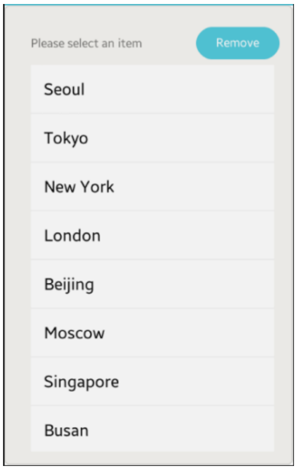
Adding a Table to a Frame makes it possible to specify an outer margin.

populate_list() is a function that adds 10 text items to a List widget. We are now going to create one. Create a new function on top of the create_base_gui() function.

```
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);
}
```

This code adds 10 text items to a List. Refer to the ListEx example for explanations of each code.

Build and run the example. One Label widget, one Button widget, and one List widget are created. And also, 10 text items are added to the List widget.



2) Inputting and Outputting Data in Eina_List

In this section, we are going to learn how to input and output string data in Eina_List. Modify the code of the populate_list() function as follows:

```
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++)
    {
        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_go(ad->list);
}
```

This code stores all item data in the Itemdata_s structure variable, stores the data in Eina_List, and then adds the string stored in Eina_List to a List widget.

eina_list_append(Eina_List*, void*) is an API that adds a new item to Eina_List. Because it returns Eina_List's pointer, it is necessary to store the address of the pointer in the Eina_List variable.

eina_list_nth(Eina_List*, unsigned int) is an API that returns item data stored in a particular position.

eina_list_count(Eina_List*) is an API that returns the number of items stored in Eina_List.

Run the example again. A list identical to the previous list appears. This time, however, data has been stored in a global variable, and therefore can be used anywhere.

3) Displaying Data about a Selected Item on the Screen

In this section, we are going to implement a feature that, when the user selects a List widget item, displays the text of the item in a Label widget. Modify the code of the populate_list() function as follows:

```
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_item_clicked_cb, idata);
    }

    elm_list_go(ad->list);
}
```

This code specifies the name of the item selection callback function for when an item is added to the List widget and passes the item data.

We specified the name of the item selection callback function as list_item_clicked.

We are now going to define the callback function. Add a new function on top of the populate_list() function.

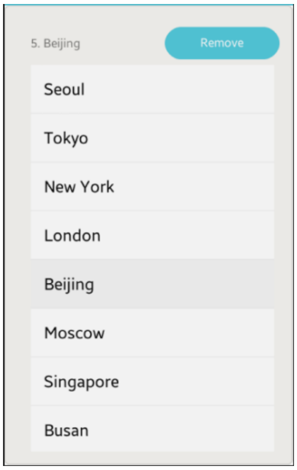
```
static void
list_item_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
    itemdata_s *idata = data;
    char buf[256];

    snprintf(buf, 256, "%d. %s", idata->id, idata->data);
    elm_object_text_set(idata->ad->label, buf);
}
```

list_item_clicked_cb() is the List item selection event function. It displays the text of a selected item in a Label.

It converts the item number and text into a single string using the sprintf() function.

It displays item data in the Label widget using the elm_object_text_set() function.

Run the example again. Select a List widget item, and you will now see its item number and text displayed in the Label widget.



4) Deleting Data

In this section, we are going to implement a feature that deletes the currently selected item when a Button is clicked. Add a new line to the Button-creating code of the create_base_gui() function.

```
/* 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);
```

We specified the name of the Button callback function as btn_clicked_cb.

Now, we need to define the callback function. Create a new function on top of the create_base_gui() function.

```
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;
    }
}
```

If the selected item does not exist in the List widget, this code displays the text “Remove” in the Button widget and exits the function.

If the selected item exists in the List widget, this code deletes the item and displays the text “Populate” in the Button widget.

When the List widget's item is deleted, the item's data stored in the array structure must be deleted together with the item. Add a new line of code at the end of the populate_list() function.

```
        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_item_cli
cked_cb, idata);
        elm_object_item_del_cb_set(eli, item_del_cb);
    }

    elm_list_go(ad->list);
}
```

elm_object_item_del_cb_set() is an API that specifies the name of an item deletion callback function.

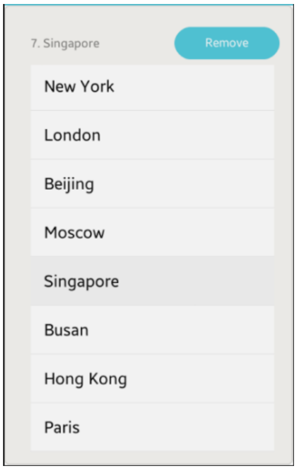
We specified the callback function as item_del_cb. Add the callback function on top of the elm_object_item_del_cb_set() function.

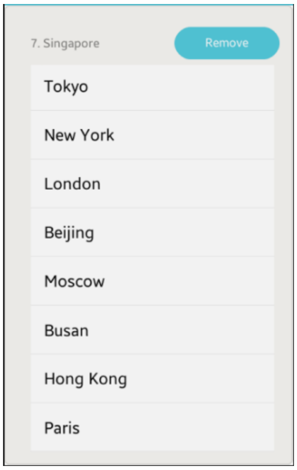
```
static void
item_del_cb(void *data, Evas_Object *obj, void *event_info)
{
    /* Those are the arguments */
    Elm_Widget_Item *it = event_info;
    itemdata_s *idata = data;
    (void) it;

    /* 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() is a List item deletion event function. It deletes item data stored in Eina_List.

eina_list_remove() is an API that deletes items stored in Eina_List.

Run the example again. Select an item and tap the Button, and the item will be deleted.





5) Related APIs

Eina_List *eina_list_append(Eina_List *list, const void *data): an API that adds a new item to Eina_List. Because it returns Eina_List's pointer, it is necessary to store the address of the pointer in the Eina_List variable. / parameters: Eina_List object, item data object.

unsigned int eina_list_count(const Eina_List *list): an API that returns the number of items stored in Eina_List. / parameters: Eina_List object.

void *eina_list_nth(const Eina_List *list, unsigned int n): an API that returns item data stored in a particular position. / parameters: Eina_List object, item index number.

Eina_List *eina_list_remove(Eina_List *list, const void *data): an API that deletes an item stored in Eina_List. Because it returns Eina_List's pointer, it is necessary to store the address of the pointer in the Eina_List variable. / parameters: Eina_List object, item data object.

void elm_list_clear(Evas_Object *obj): an API that deletes all the items in the list from a List widget.

26. Using Timers

A timer lets an event occur at regular time intervals. Timers are used necessarily for alarm apps and animation effects. When you develop an app, multiple timers are operated at the same time. We are now going to learn how to use it through an example.

1) Starting a Timer Event

Create a new source project and specify the project name as 'TimerEx.' After the source project is created, open the source file (~.c) under the src folder and add a new variable to the appdata structure.

```
typedef struct appdata {
    Evas_Object *win;
    Evas_Object *conform;
    Evas_Object *label;
    Ecore_Timer *timer1;
    int timer_count;
} appdata_s;
```

Ecore_Timer is the timer structure.

timer_count is a variable that stores the number of times a timer event has occurred.

Add two new functions on top of the create_base_gui() function.

```
static void
my_box_pack(Evas_Object *box, Evas_Object *child, double h_weight, double v_weight,
                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_EXPAND);
        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() is a function that adds a widget to a Box container.

my_button_add() is a function that creates and then returns a Button widget.

Go back to the create_base_gui() function and add new code. This code creates two Boxes and two Buttons.

```
    /* 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_EX
PAND);
        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
_HINT_FILL);

            /* Button-2 */
            btn = my_button_add(box, "Stop", btn_stop_cb, ad);
            my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EV
AS_HINT_FILL, 0.0);
        }
    }

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

The first Button is responsible for starting a timer. We are now going to create a callback function for the Button. Add two new functions on top of the create_base_gui() function.

```
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)
{
}
```

The btn_start_cb() function is called when the first Button is clicked. This code resets the count of timer event occurrences to '0' and creates/starts a new timer.

ecore_timer_add(double, Ecore_Task_Cb, void*) is an API that creates and then returns a new timer. The first parameter indicates the time interval. The unit of time is seconds. For example, to make an event occur every 1.5 seconds, you need to pass ‘1.5’. The second parameter indicates the name of the timer event callback function, while the third parameter indicates user data.

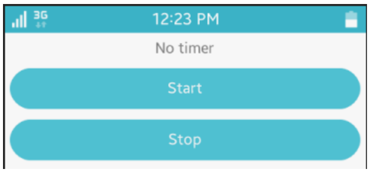
btn_stop_cb() is the event function for the second Button. We will add this feature in a short while.

We are now going to create a timer event callback function. Add a new function on top of the btn_start_cb() function.

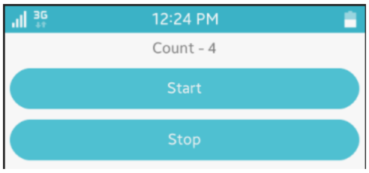
```
static Eina_Bool
timer1_cb(void *data EINA_UNUSED)
{
    appdata_s *ad = data;
    ad->timer_count ++;
    char buf[100];
    sprintf(buf, "Count - %d", ad->timer_count);

    elm_object_text_set(ad->label, buf);
    return ECORE_CALLBACK_RENEW;
}
```

When a timer event occurs, this new function is called. This code increases the count of timer event occurrences and displays the current count on the screen.

Build and run the example. Tap the Start button, and you will see the number in the Label widget increases once every second.



No timer



Count - 4

2) Stopping Timers

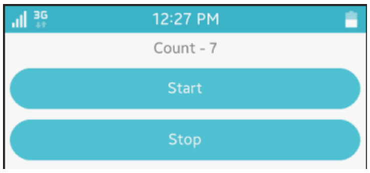
In this section, we are going to implement a feature that stops the timer when the Stop button is tapped. Add code on top of the btn_stop_cb() function as shown below:

```
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");
}
```

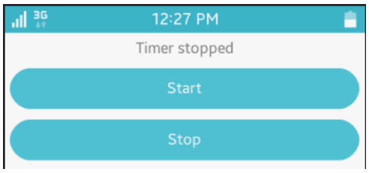
ecore_timer_freeze(Ecore_Timer*) is an API that pauses a timer event. To resume a timer event, you need to use the ecore_timer_thaw() function.

ecore_timer_del(Ecore_Timer*) is an API that deletes a timer object.

Run the example again. Tap the Start button, and then after a while, tap the Stop button. The timer event then stops.



Count - 7



Timer stopped

3) Related APIs

Ecore_Timer *ecore_timer_add(double in, Ecore_Task_Cb func, void *data): an API that creates and then returns a new timer. / parameters: time interval (unit: second), name of the timer event callback function, and user data.

void ecore_timer_freeze(Ecore_Timer *timer): an API that pauses a timer event.

void ecore_timer_thaw(Ecore_Timer *timer): an API that resumes a timer.

void *ecore_timer_del(Ecore_Timer *timer): an API that deletes a timer object.

27. Time & Date

Using i18n_ucalendar_h enables requesting the current date and time, and it also enables adding two different times. We are now going to learn how to use it through an example.

1) Requesting the Time Zone

Create a new source project and specify the project name as 'DateTime.' After the source project is created, open the source file (~.c) under the src folder and add a new variable to the appdata structure. Include the library header file as well.

```
#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;
```

We are going to display the time zone in label1, the current date and time in label2, the POSIX time in label3, the title of the time addition in label4, and the result of the time addition in label5.
slide is a Slide widget that changes the current date to the date resulting from an addition.

tzid is a string variable that stores the time zone.

i18n_ucalendar_h is a structure that stores date and time information.

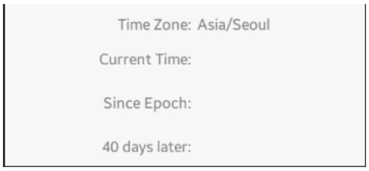
We are now going to create eight Label widgets on the screen. To request the current time, you also need to request the time zone setting. Go back to the create_base_gui() function and add new code.

```
    /* 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);

    /* 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_TIMEZON
E, &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);
```

We created four Labels and two Buttons.

system_settings_get_value_string() is an API that requests system configuration information. Passing SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE to the first parameter returns the time zone string to the second parameter.

Build and run the example. The time zone specified in Settings is displayed in Label1.



Time Zone: Asia/Seoul
Current Time:
Since Epoch:
40 days later:

2) Creating the Current Time

When you create an i18n_ucalendar_h object, the current time is automatically stored. Create a new function on top of the create_base_gui() function.

```
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_TRADITION
AL, &ucal);
    if (ret != 0)
    {
        dlog_print(DLOG_ERROR, LOG_TAG, "i18n_ucalendar_create() failed with err =
 %d", ret);
        return NULL;
    }

    return ucal;
}
```

To create an i18n_ucalendar_h object, you need to convert the time zone into the i18n_uchar format.

i18n_ustring_copy_ua() is an API that copies the time zone stored in a char array to an i18n_uchar array.

i18n_ustring_get_length() is an API that returns the length of an i18n_uchar array.

i18n_ucalendar_create(i18n_uchar *zone_id, int32_t len, char *locale, i18n_ucalendar_type_e type, i18n_ucalendar_h *calendar) is an API that creates an i18n_ucalendar_h object. For the first parameter, the time zone is returned; for the second parameter, the length of the time zone string; for the third parameter, the name of the region; for the fourth parameter, the ucalendar type; and for the fifth parameter, the i18n_ucalendar_h object.

You can obtain the current time this way. We are now going to convert the date and time stored in i18n_ucalendar_h into a string and display the string on the screen. Create a new function on top of the create_base_gui() function. This function receives an i18n_ucalendar_h object and converts its date and time into a string.

```
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) is an API that requests a type of data from i18n_ucalendar_h. Passing I18N_UCALENDAR_YEAR to the second parameter makes the third parameter return the year value.

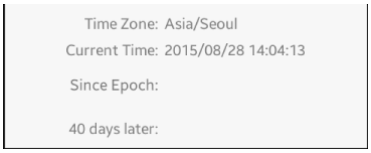
This code requests the year, month, date, hour, minute, and second values, in order.

By using the sprintf() function, this code turns the six types of time values into a single string printf() and returns the string. One thing to note is that in the case of month, value '1' needs to be added. The range of values for month is 0 to 11.

We are now going to display the current time on the screen using the two functions we just created. Add new code at the end of the create_base_gui() function.

```
    /* 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);
}
```

Run the example again. The current date and time are displayed in Label2.



Time Zone: Asia/Seoul

Current Time: 2015/08/28 14:04:13

Since Epoch:

40 days later:

3) Digital Watch

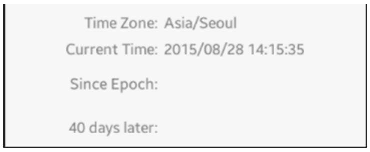
In this section, we are going to implement a feature that renews the date and time displayed in Label2 once every second. In other words, by using a timer, we will be implementing a feature that functions the same as a digital watch. Add new code at the end of the create_base_gui() function.

```
    /* 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);
}
```

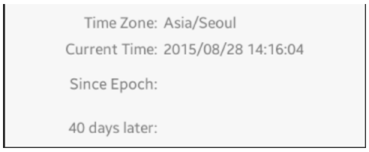
This code generates a timer event once every second. Now, we need to create a timer event function. Create a new function on top of the create_base_gui() function.

```
static Eina_Bool
timer_cb(void *data)
{
        appdata_s *ad = data;
        update(ad);
        return ECORE_CALLBACK_RENEW;
}
```

When a timer event occurs, this code calls the update() function and renews the time and date. Run the example again. The time changes once every second.



Time Zone: Asia/Seoul
Current Time: 2015/08/28 14:15:35
Since Epoch:
40 days later:



Time Zone: Asia/Seoul
Current Time: 2015/08/28 14:16:04
Since Epoch:
40 days later:

4) Calculating POSIX Time

The time unit used on computers is the millisecond, and if you convert the amount of time from 1 A.D. to the current year into a time value in milliseconds, an astronomical number will be created. Therefore, it became necessary to come up with a new time system that starts after the advent of the computer. 0:00:00 January 1, 1970 is the date that was eventually decided on as the starting date. This system is called POSIX time. With the use of POSIX time, it is possible to calculate how much time has passed since the app started.

Add a new line of code to the update() function. This code specifies the name of the Button callback function.

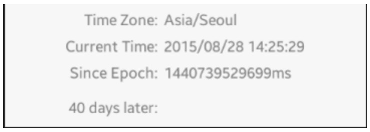
```
static void
update(appdata_s *ad)
{
    int year, month, day, hour, minute, second;
    i18n_udate udate;
    char buf[256];
    int diff;

    /* 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 is the same type of data as double. POSIX time has a very large numeric value, so it is necessary to use this type of format.

i18n_ucalendar_get_now(i18n_udate) is an API that converts the current time to a time in milliseconds.

i18n_ucalendar_set_milliseconds(i18n_ucalendar_h, i18n_udate) is an API that specifies a new time in i18n_ucalendar_h.

Run the example again, and you will now see POSIX time displayed in label3. The displayed unit of time is milliseconds.



```
Time Zone: Asia/Seoul
Current Time: 2015/08/28 14:25:29
Since Epoch: 1440739529699ms
40 days later:
```

4) Calculating Time

In this section, we are going to learn how to add numbers to the date. First, add one Slider widget. Add new code at the end of the create_base_gui() function.

```
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);
```

We created a Slider widget and specified a range of -365 to +365 for the widget. We specified the value as 40.

We are now going to create an event function for the Slider. Create a new function on top of the create_base_gui() function. When the value of the Slider is changed, the update() function is called.

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

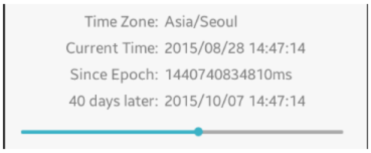
Lastly, add new code at the end of the update() function.

```
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);
}
```

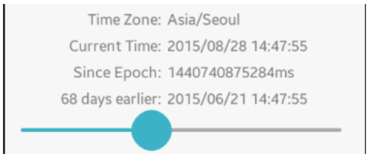
Distinguishing between negative and positive values, this code displays a given Slider value in label4 and displays the sum of the current date and the value in label5.

i18n_ucalendar_add() is a function that adds a number to a particular item of the i18n_ucalendar_h object. Passing 18N_UCALENDAR_DATE to the second parameter and passing 40 to the third parameter creates a date that is 40 days after the current date.

Run the example again and tap the second Button. The date that is 40 days after the current date is displayed in label4. Dragging through the Slider changes the date used for calculation.



Time Zone: Asia/Seoul
Current Time: 2015/08/28 14:47:14
Since Epoch: 1440740834810ms
40 days later: 2015/10/07 14:47:14



Time Zone: Asia/Seoul
Current Time: 2015/08/28 14:47:55
Since Epoch: 1440740875284ms
68 days earlier: 2015/06/21 14:47:55

5) Related APIs

int system_settings_get_value_string(system_settings_key_e key, char **value): an API that requests system configuration information. Passing SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE to the first parameter returns the time zone string to the second parameter.

i18n_uchar* i18n_ustring_copy_ua (i18n_uchar *dest, const char *src): an API that copies the time zone stored in the char array to the i18n_uchar array.

int32_t i18n_ustring_get_length (i18n_uchar *s): an API that returns the length of the i18n_uchar array.

int i18n_ucalendar_create (i18n_uchar *zone_id, int32_t len, char *locale, i18n_ucalendar_type_e type, i18n_ucalendar_h *calendar): an API that creates an i18n_ucalendar_h object. / parameters: time zone, length of the time zone string, region name, ucalendar type, and the i18n_ucalendar_h object return.

int i18n_ucalendar_get (i18n_ucalendar_h calendar, i18n_ucalendar_date_fields_e field, int32_t *val): an API that requests one type of data from i18n_ucalendar_h. / parameters: i18n_ucalendar_h object, date and time fields, and the date and time values return.

The date and time field types:
- I18N_UCALENDAR_YEAR: year
- I18N_UCALENDAR_MONTH: month
- I18N_UCALENDAR_DATE: date
- I18N_UCALENDAR_DAY_OF_WEEK: day of the week
- I18N_UCALENDAR_AM_PM: a.m. or p.m.
- I18N_UCALENDAR_HOUR: hours
- I18N_UCALENDAR_MINUTE: minutes
- I18N_UCALENDAR_SECOND: seconds
- I18N_UCALENDAR_MILLISECOND: milliseconds

i18n_udate: the same type of data as double. It is used for storing POSIX time.

int i18n_ucalendar_get_milliseconds(i18n_ucalendar_h calendar, i18n_udate *date): an API that the converts the time stored in i18n_ucalendar_h into POSIX time. The displayed unit of time is milliseconds.

int i18n_ucalendar_add (i18n_ucalendar_h calendar, i18n_ucalendar_date_fields_e field, int32_t amount): an API that adds a number to a particular item of the 18n_ucalendar_h object. / parameters: i18n_ucalendar_h object, date and time fields, and the numbers being added.

28. Calendar Example

In this example, we are going to learn how to create a calendar using i18n_ucalendar_h.

1) Screen UI Composition

Create a new source project and specify the project name as 'CalendarEx.' After the source project is created, open the source file (~.c) under the src folder and add a new variable to the appdata structure and also add a library header file.

```
#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 is a Label widget array that displays the date. The space allowed for displaying the date is up to 7 columns horizontally and up to 6 rows vertically.

In tzid, store the time zone.

In ucal, store the current date and time.

We are now going to add Label widgets for displaying the date to the screen. Create a new function on top of the create_base_gui() function. This function adds a widget to a Table.

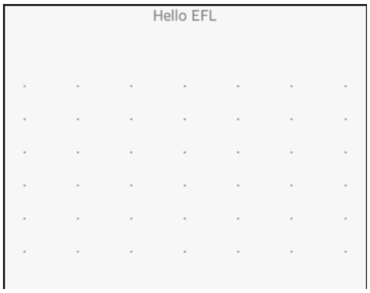
```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int x, int y, int w, int h)
{
    evas_object_size_hint_align_set(child, 0.5, 0.5);
    evas_object_size_hint_weight_set(child, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_table_pack(table, child, x, y, w, h);
    evas_object_show(child);
}
```

Then, add new code at the end of the create_base_gui() function. This code creates a Box and a Table and adds a Label. Annotate the Conformant-creating code.

```
    /* 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);*/
    /* 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_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_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);
```

By using two for-loops, we created a Label widget that is 6 columns vertically and 7 rows horizontally.

Build and run the example. The symbol '.' is displayed in the newly added Label.



2) Displaying the Current Date and Time

In this section, we are going to create an i18n_ucalendar_h object and displays the current date and time on the screen. Add two new functions on top of the create_base_gui() function.

```
static  i18n_ucalendar_h
```

```
create_time(char *tzid)
{
    i18n_ucalendar_h ucal;
    i18n_uchar *_tzid = (i18n_uchar*)calloc(strlen(tzid) + 1, sizeof(i18n_uchar));
    i18n_ustring_copy_ua(_tzid, tzid);
    int len = i18n_ustring_get_length(_tzid);
    int ret = i18n_ucalendar_create(_tzid, len, "en_US", I18N_UCALENDAR_TRADITIONA
L, &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, ho
ur, minute, second);
    return buf;
}
```

create_time() is a function that creates and then returns an i18n_ucalendar_h object, and time2string() is a function that converts the date stored in i18n_ucalendar_h to a string and returns the string. For detailed information, see the DateTime example.

In this section, we are going to use the function above and display the current date and time on the screen. Add new code at the end of the create_base_gui() function.

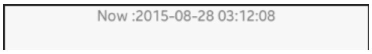
```
    evas_object_show(ad->win);

    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);
}
```

Request the time zone using the system_settings_get_value_string() function.

Create an i18n_ucalendar_h object using the create_time() function.

Convert the date and time stored in i18n_ucalendar_h into a single string using the time2string() function.

Run the example again. The current date and time are displayed in the Label widget.



Now :2015-08-28 03:12:08

3) Calculating Calendar Dates

In this section, we are going to find out what day of the week the current month's first day is and enter a date in the Label array. Create a new function on top of the create_base_gui() function.

```
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_leap);
        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[] is an array that stores a maximum number of days for January through December.

i18n_ucalendar_set() is an API that specifies a new value for a particular item of an i18n_ucalendar_h object. Passing I18N_UCALENDAR_DATE to the second parameter and passing 1 to the third parameter change the current date to the 1st.

i18n_ucalendar_get() is an API that returns the value of a particular item of an i18n_ucalendar_h object. The date item types are as follows:
- Passing I18N_UCALENDAR_MONTH to the second parameter makes the third parameter return the value of Month.
- Passing I18N_UCALENDAR_DAY_OF_WEEK to the second parameter makes the third parameter return the number of Day of Week.
- Passing I18N_UCALENDAR_IS_LEAP_MONTH to the second parameter makes the third parameter return whether or not the current month is a leap month.

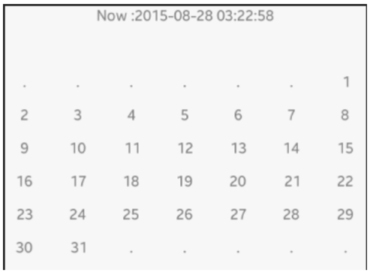
If the current month is a leap month, the max day is changed to 29 days.

The part after the above is code that converts the numbers that respectively correspond to the days between the first day and the last day of a month to strings and then display them in the relevant Label widget.

We need to let this function be called when the app is executed. Add a new line of code at the end of the create_base_gui() function.

```
    elm_object_text_set(ad->label, buf);

    draw_calendar(ad);
}
```

Run the example again. The calendar that corresponds to today's date is displayed.



```
Now :2015-08-28 03:22:58

.    .    .    .    .    .    1
2    3    4    5    6    7    8
9    10   11   12   13   14   15
16   17   18   19   20   21   22
23   24   25   26   27   28   29
30   31   .    .    .    .    .
```

4) Displaying the Current Date

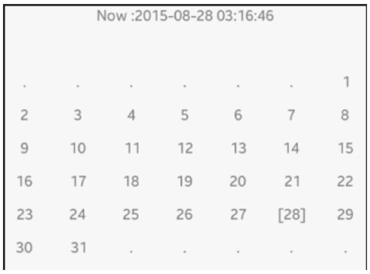
In this section, we are going to implement a feature that encloses the number corresponding to today's date in the '[]' symbol. Add new code to the draw_calendar() function.

```
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_leap);
        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);
        ~
```

Request today's date using the i18n_ucalendar_get() function and store it in a variable.

When it is a turn for today’s date in the for-loop, the symbol '[]' is added to the value.

Run the example again. You will now see today's date distinguished from the other dates.



```
              Now :2015-08-28 03:16:46

 .     .     .     .     .     .     1
 2     3     4     5     6     7     8
 9     10    11    12    13    14    15
 16    17    18    19    20    21    22
 23    24    25    26    27    [28]  29
 30    31    .     .     .     .     .
```

5) Calendar Widget

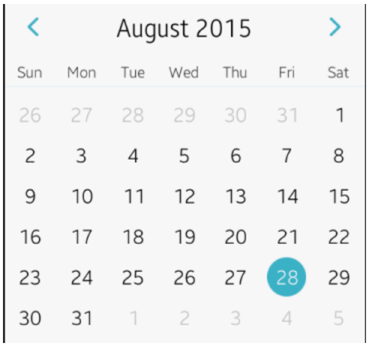
Using the Calendar widget makes it easy to create a Calendar. Create a new example and specify its name as CalendarWidgetEx.

Add new code to the create_base_gui() function.

```
    /* 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_EX
PAND);
        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_
EXPAND);
            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() is an API that creates a Calendar widget.

Run the example, and you will now see a Calendar displayed. Use the left/right arrow to go to the previous/next month.



< August 2015 >

Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

For detailed information on how to use the Calendar widget, refer to the following path in Help Contents.

API References > Native Application > Mobile Native > Native API Reference > UI > EFL > Elementary > Elementary Widgets

6) Related APIs

int system_settings_get_value_string(system_settings_key_e key, char **value): an API that requests system configuration information. Passing SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE to the first parameter returns the time zone string to the second parameter.

i18n_uchar* i18n_ustring_copy_ua (i18n_uchar *dest, const char *src): an API that copies the time zone stored in the char array to the i18n_uchar array.

int32_t i18n_ustring_get_length (i18n_uchar *s): an API that returns the length of the i18n_uchar array.

int i18n_ucalendar_create (i18n_uchar *zone_id, int32_t len, char *locale, i18n_ucalendar_type_e type, i18n_ucalendar_h *calendar): an API that creates an i18n_ucalendar_h object. / parameters: time zone, length of the time zone string, region name, ucalendar type, and the i18n_ucalendar_h object return.

int i18n_ucalendar_get (i18n_ucalendar_h calendar,
i18n_ucalendar_date_fields_e field, int32_t *val): an API that requests one type of data from i18n_ucalendar_h. / parameters: i18n_ucalendar_h object, date and time fields, and the date and time values return.

The date and time field types:
- I18N_UCALENDAR_YEAR: year
- I18N_UCALENDAR_MONTH: month
- I18N_UCALENDAR_DATE: date
- I18N_UCALENDAR_DAY_OF_WEEK: day of the week
- I18N_UCALENDAR_AM_PM: a.m. or p.m.
- I18N_UCALENDAR_HOUR: hours
- I18N_UCALENDAR_MINUTE: minutes
- I18N_UCALENDAR_SECOND: seconds
- I18N_UCALENDAR_MILLISECOND: milliseconds

i18n_udate: the same type of data as double. It is used for storing POSIX time.

int i18n_ucalendar_get_milliseconds(i18n_ucalendar_h calendar, i18n_udate *date): an API that the converts the time stored in i18n_ucalendar_h into POSIX time. The displayed unit of time is milliseconds.

int i18n_ucalendar_add (i18n_ucalendar_h calendar, i18n_ucalendar_date_fields_e field, int32_t amount) : an API that adds a number to a particular item of the 18n_ucalendar_h object. / parameters: i18n_ucalendar_h object, date and time fields, and the numbers being added.

29. Requesting a Mouse Touch Event

To create an image viewer, you need to implement a feature that displays images in a slideshow where the user can switch between the images by dragging the screen left and right. In addition, in the case of musical instrument apps such as ocarina and piano apps, it is necessary to request multi-touch information. Besides the apps mentioned above, the majority of games and quality other apps also use touch events. In this example, we are going to learn how to request an event when the user has touched a Window.

1) Requesting a Container Touch Event

Create a new source project and specify the project name as 'MouseTouchEvent.' After the source project is created, open the source file (~.c) under the src folder and add new code at the end of the create_base_gui() function.

```
    /* 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_EXPAN
D);
    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() is an API that specifies a callback function for an evas object. Evas objects in this case are all objects displayed on the screen. Therefore, evas objects include both basic objects (Line, Rect, Polygon, Text, and Image) and smart objects (containers and widgets).
For the first parameter, enter the object that an event will occur. In this case, we specified the object as Conformant.
For the second parameter, enter the event type.
EVAS_CALLBACK_MOUSE_DOWN indicates the Touch Down event.
EVAS_CALLBACK_MOUSE_MOVE indicates the Touch Move event.
EVAS_CALLBACK_MOUSE_UP indicates the Touch Cancel event.
For the third parameter, enter the name of the callback function. The fourth parameter indicates user data.

We are now going to implement a callback function that is called when the user touches a Conformant. Add three new functions on top of the create_base_gui() function.

```
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() is the callback function for when a Conformant is Touch Downed. The first parameter receives user data, and the second parameter receives the object where an event has occurred. The third parameter receives the Evas_Event_Mouse_Down object that contains Touch event information.

Among the properties of an Evas_Event_Mouse_Down object, the canvas property contains Touch point coordinates.

on_mouse_move() is the callback function for when a Conformant is Touch Moved. The first parameter receives user data, and the second parameter receives the object where an event has occurred. The third parameter receives the Evas_Event_Mouse_Move object that contains Touch event information.

Among the properties of an Evas_Event_Mouse_Move object, the prev.canvas property contains the previous Touch point coordinates. cur.canvas contains the current Touch point coordinates.

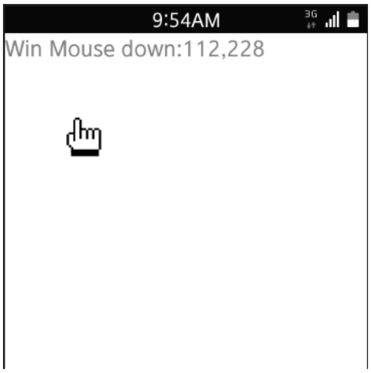
on_mouse_up() is the callback function for when a Conformant is Touch Canceled. The first parameter receives user data, while the second parameter receives the object where an event has occurred. The third parameter receives the Evas_Event_Mouse_Up object that contains Touch event information.

Among the properties of an Evas_Event_Mouse_UP object, the canvas property contains Touch point coordinates.

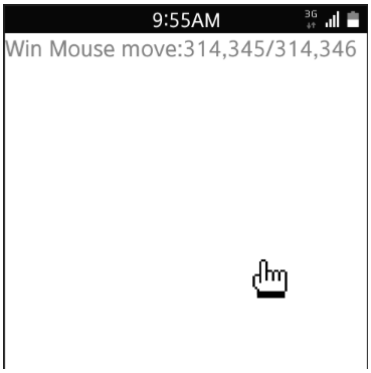
Build and run the example. Touch the screen using your mouse. You will see the current point coordinates displayed in the Label widget.

Drag the screen using your mouse, you can see the previous Touch point coordinates and the current Touch point coordinates displayed in the Label widget.

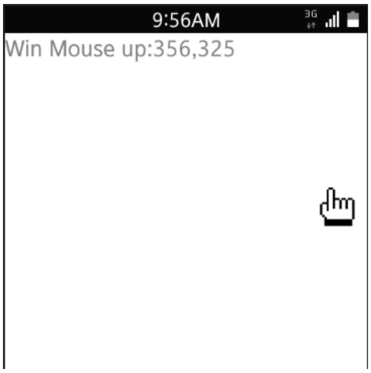
Removing your finger from your mouse displays the last point coordinates in the Label widget.



Win Mouse down:112,228



Win Mouse move:314,345/314,346



Win Mouse up:356,325

3) Requesting a Multi Touch Event

In this section, we are going to request each relevant piece of touch information when the user has touched the screen with multiple fingers. Add new code to the create_base_gui() function.

```
   /* 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_mo
use_up , ad);

   /* Multi Touch event callback */
   evas_object_event_callback_add(ad->conform, EVAS_CALLBACK_MULTI_DOWN,
multi_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 indicates the Multi Touch Down event.

EVAS_CALLBACK_MULTI_MOVE indicates the Multi Touch Move event.

We are now going to create a Multi Touch event function. Add two new functions to the create_base_gui() function.

```
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->canvas.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.canv
as.y);
    elm_object_text_set(ad->label, buf);
}
```

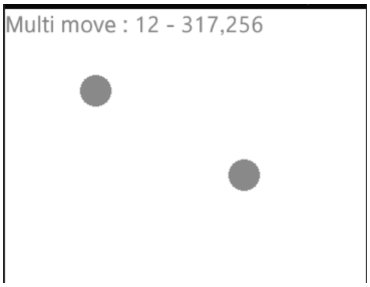
multi_down_cb() indicates the Multi Touch Down event function. The first point's event is not received. The third parameter receives the Evas_Event_Multi_Down object that contains Touch event information.

Among the properties of an Evas_Event_Multi_Down object, the canvas property contains Touch point coordinates. The device property contains the ID of the Touch point. When a Move event or an Up event has occurred, the event can be identified by the ID.

multi_move_cb() indicates the Multi Touch Move event function. The first point's event is not received. The third parameter receives the Evas_Event_Multi_Move object that contains Touch event information.

Among the properties of an Evas_Event_Multi_Move object, the cur property contains Touch point coordinates. The device property contains the ID of the Touch point. When a Down event has occurred, the event can be identified by the ID.

Run the example again. To test out Multi Touch in an emulator, left-click two spots on the screen while holding down the Ctrl key on your keyboard. Then, a gray circle appears. To test out a Multi Touch Move event, drag the gray circle. Removing your finger from the Ctrl key makes the Multi Touch marks disappear.



Multi move : 12 - 317,256

4) Related APIs

void evas_object_event_callback_add(Evas_Object *obj, Evas_Callback_Type type, Evas_Object_Event_Cb func, void *data): an API that specifies a callback function for an evas object. Evas objects refer to all objects displayed on the screen. Therefore, evas objects include both basic objects (Line, Rect, Polygon, Text, and Image) and smart objects (containers and widgets). / parameters: object where an event occurs, type of the event, name of the callback function, and user data. The event types are as follows:

- EVAS_CALLBACK_MOUSE_DOWN: Touch Down event
- EVAS_CALLBACK_MOUSE_MOVE: Touch Move event
- EVAS_CALLBACK_MOUSE_UP: Touch Cancel event
- EVAS_CALLBACK_MULTI_DOWN: Multi Touch Down event
- EVAS_CALLBACK_MULTI_MOVE: Multi Touch Move event

Evas_Event_Mouse_Down: Touch Down event information structure. Among the properties, the canvas property contains Touch point coordinates.

Evas_Event_Mouse_Move: Touch Move event information structure. Among the properties, the prev.canvas property contains the previous Touch point coordinates. cur.canvas contains the current Touch point coordinates.

Evas_Event_Mouse_Up: Touch Cancel event information structure. Among the properties, the canvas property contains Touch point coordinates.

Evas_Event_Multi_Down: Multi Touch Down event information structure. Among the properties, the canvas property contains Touch point coordinates. The device property contains the ID of the Touch point. When a Move event or an Up event has occurred, the event can be identified by the ID.

Evas_Event_Multi_Move: Multi Touch Move event information structure. Among the properties of the object, the cur property contains Touch point coordinates. The device property contains the ID of the Touch point. When a Down event has occurred, the event can be identified by the ID.

30. Calculator Example

In this example, we are going to learn how to create a simple calculator using mathematical functions.

1) Screen UI Composition

Create a new source project and specify the project name as 'CalculatorEx.' After the source project is created, open the source file (~.c) under the src folder, add a new variable to the appdata structure, and also add a library header file and define statements.

```
#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 is the mathematical functions library header file.

In define statements, define IDs for distinguishing between Buttons.

In 'entry' that has been added to the appdata structure, the result of the calculation is displayed. In 'value', a temporary result of the calculation is stored.

In calc_mode, the types of the calculation (+, - , *, /) are stored.

In this example, a large number of Button widgets will be created. To reduce the source code, we are now going to create a Button-creating function. Add two new functions on top of the create_base_gui() function.

```
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() is the Button callback function. The same callback function will be called by multiple Buttons.

create_button() is a function that receives Button-related information and creates a Button widget.

We are now going to create a Box, a Table, and 22 Buttons. Add new code to the create_base_gui() function.

```
    /* 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);

    {
        /* 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);
```

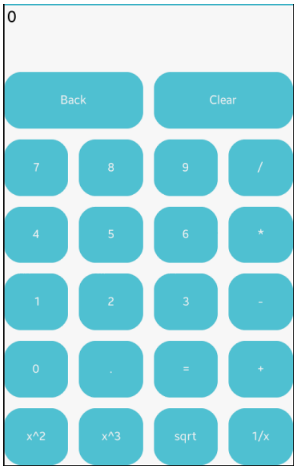
```
evas_object_size_hint_weight_set(box, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
        elm_win_resize_object_add(ad->win, box);
        evas_object_show(box);
        /* Table */
        Evas_Object *table = elm_table_add(ad->win);
        /* Make table homogenous - every cell will be the same size */
        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
_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_E
XPAND);
        /* 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_HINT
_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);
```

To place the widgets on the screen based on the aspect ratio, we created a Table. In addition, to specify the distance between the widgets, we created a Box.

To display calculation results, we created an Entry widget.

The following is code that creates a total of 22 Button widgets. In the case of number Buttons, the relevant numeric value is passed to the callback function as user data. In the case of other Buttons, the ID value defined the define statement is passed as user data.

Build and run the example. You can see an Entry widget at the top of the screen and 22 Button widgets below the Entry widget.



2) Implementing the Number Button Feature

In this section, we are going to implement a feature that, when a number Button between Button 0 and Button 9 is clicked, adds the relevant number to a Label widget. First, declare appdata as a global variable.

```
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;
```

Then, initialize the appdata variable at the beginning of the create_base_gui() function.

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

We are now going to create a function that converts the caption text of an Entry into a number and also create a function that adds text to an Entry widget. Add two new functions on top of the btn_clicked_cb() function.

```
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() is a function that coverts the caption text of an Entry into the float type and then returns it.

append_number_label() is a function that receives the char variable and adds it at the end of the caption text of an Entry.

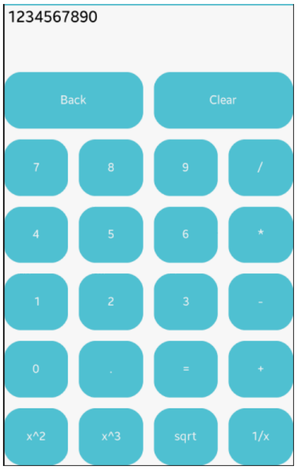
This function needs to be called when the user taps a number Button. Add the following code to the btn_clicked_cb() function.

```
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;
    }
}
```

If the user data is a number between 0 and 9, the Button is deemed a number Button, and new text is added to the caption text of a Label widget.

Run the example again. Click a number Button, and then the relevant value is added to the Label widget.



1234567890

3) Implementing the Dot, Clear, and Back Button Features

In this section, we are going to implement the ‘.’ button, Back button, and Clear button features. Add new code at the end of the btn_clicked_cb() function.

```
    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;
    }
}
```

If the user has clicked the '.' button, it adds the '.' symbol to the end of the Label text.

If the user has tapped the Clear button, this code changes the Label text to '0.'

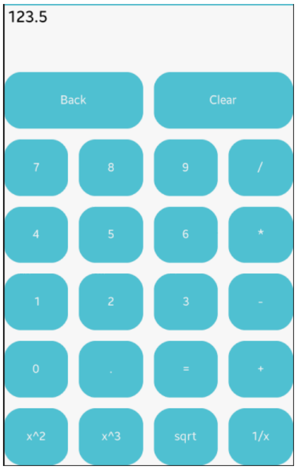
If the user has tapped the Back button, this code deletes the last character of the Label text.

eina_stringshare_add_length(char*, unsigned int) is an API that extracts a specified length of characters from the beginning of a string.

Run the example again. Tap the '.' button on the keypad, and the symbol '.' is added at the right end of the Label text.

Tap the Back button, and one character at the right end of the Label text is deleted.

Tap the Clear button, and the entire text is changed to '0.'



4) Arithmetic Operations

In this section, we are going to create a function that enters a float type value in a Label widget and performs an arithmetic operation when the '=' button is tapped. Add two new functions on top of the btn_clicked_cb() function.

```
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() is a function that receives a real number, converts it into a string, and then enters it in a Label widget.

btn_equal_clicked() is a function that performs computation using two numbers according to the type of the arithmetic operation.

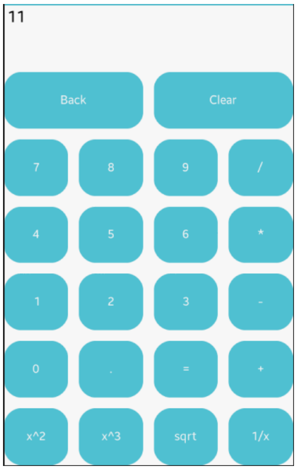
Add new code to the btn_clicked_cb() function.

```
    ~
    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;
    }
}
```

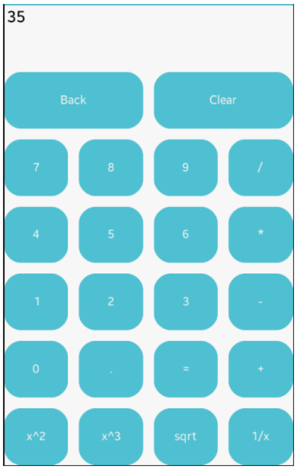
When the user taps the Arithmetic Operations button, this code converts Entry text into numbers and stores them in a global variable;
it changes Entry text to '0' and then stores the type of the arithmetic operation in a global variable.

When the user taps the ‘=’ button, this code calls the btn_equal_clicked() function.

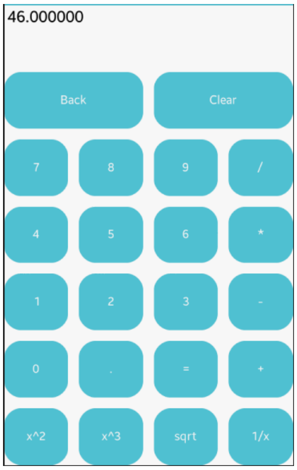
Run the example again. Then, enter '11' and tap the '+' Button. The number '0' is displayed in the Entry widget.
Then, enter ‘35’ and tap the ‘=’ button. The result of the operation is displayed in the Entry widget.
It is recommended that you test out subtraction, multiplication, and division as well.



11



35



46.000000

5) Calculating the Square and Square Root of a Number

In this section, we are going to calculate the square and square root of a number using mathematical functions. Add new code at the end of the btn_clicked_cb() function.

```
    ~
    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;
    }
}
```

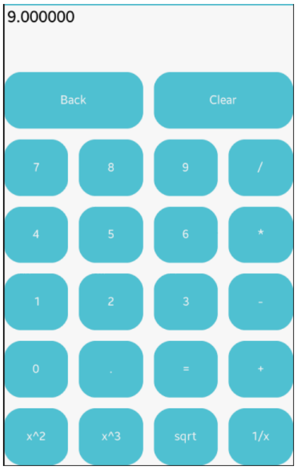
pow(double, double) is a mathematical function that calculates the square of a number. Passing 2 to the second parameter returns the square of the number, while passing 3 returns the cube of the number.

sqrt(double) is a mathematical function that calculates the square root of a number.

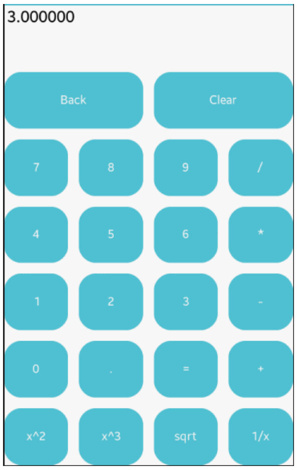
To request the inverse number of a number, divide 1 by the number.

Run the example again. Enter the number 3 and tap the ‘x^2’ button, and the square of the number is then displayed.

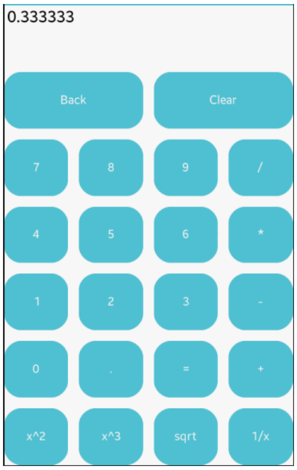
Tapping the ‘sqrt’ button in that status calculates the square root of the calculated value and in turn displays the original numerical value of the number.
Tapping the '1/x' Button displays the inverse number of the number.



9.000000



3.000000



0.333333

6) Related APIs

double pow(double, double): an API that is the mathematical function for calculating the square of a number. / parameters: original number and raise power.

double sqrt(double): an API that calculates the square root of a number. / parameters: original number.

31. Displaying a Gradient on the Canvas

It is common for newcomers to app development to ask

what skills are essential for success in the field.

The answer is invariably that they need good graphics/rendering skills first and foremost.

Second, good Object Oriented Programming (OOP) skills are required.

As time passes, app development is becoming increasingly easy. In the past, it was commonplace for developers to implement features themselves through hard coding. These days, however, platforms provide a variety of features, and developers only need to call and use APIs already provided by platforms.

Graphics is the most difficult part and still demands a lot of effort when developing a commercial smartphone app. When two given apps have the same features, the user chooses the one with better graphics. The most essential quality required of the app developer is the skill to implement diverse and impressive graphics that suit the user's taste.

Evas is the canvas provided in EFL. All shapes drawn on Evas get created as objects. In this example, we are going to learn how to draw a line on this canvas.

1) Creating a Canvas and Drawing a Gradient

Create a new source project and specify the project name as 'DrawGradiation.' After the source project is created, open the source file (~.c) under the src folder and add new code at the top as shown below:

```
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 = col
ors[0][3];
    const int e_r = colors[1][0], e_g = colors[1][1], e_b = colors[1][2], e_a = col
ors[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_EXPAN
D);
    evas_object_size_hint_align_set(img, EVAS_HINT_FILL, EVAS_HINT_FILL);
    evas_object_show(img);
    return img;
}
```

To draw a picture on the screen, you need to use a canvas called Evas. In this section, we are going to create five Image objects. imgs[5] added to the appdata structure is an array variable that stores an Image object.

create_gradient_rect() is a function that creates and then returns a gradated Image object. The following are descriptions of the functions:

colors[2][4] is an array variable that stores two kinds of colors. One is the gradient start color, and the other is the gradient end color. A color consists of a total of four numeric values. The numeric values listed in order are Red, Green, Blue, and Alpha (semi-transparent).

evas_object_evas_get() is an API that creates an Evas object.

evas_object_image_filled_add() is an API that creates an Image object.

evas_object_image_colorspace_set() is an API that specifies the color space of an Image object. Passing EVAS_COLORSPACE_ARGB8888 to the second parameter lets one pixel consist of four different data (Red, Green, Blue, and Alpha).

evas_object_image_size_set() is an API that specifies the size of an Image object. Passing 255 to the second parameter makes the number of horizontal pixels be 255. Passing 1 to the third parameter makes the number of vertical pixels be 1.

evas_object_image_alpha_set() is an API that specifies whether to apply semi-transparency to an Image object.

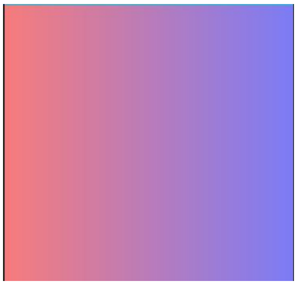
evas_object_image_data_get() is an API that returns the original data of an Image object as an array.

evas_object_image_data_set() performs the opposite function of evas_object_image_data_get(). It is an API that specifies original data for an Image object.

We are now going to create and display a gradated Image object on the screen using this function. Add new code to the create_base_gui() function. Label will not be used in this example, so annotate it.

```
    /* 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);
```

We created a Box object and an Image object and displayed them on the screen. Run the example. A gradated square whose left side is in red and whose right side is in blue is displayed on the screen.



2) Creating Various Gradated Squares

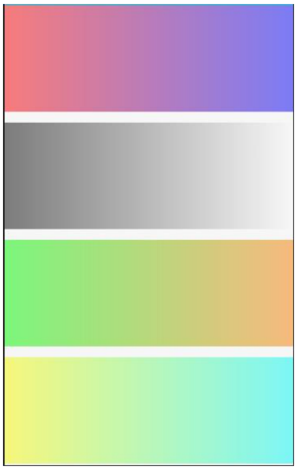
In this section, we are going to implement a feature in the create_gradient_rect() function we created in the previous section that creates four different kinds of gradated squares. Modify the code as follows:

```
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;
```

Calling the function above four times will create four different gradated Images. Modify the bottom of the create_base_gui() function as follows:

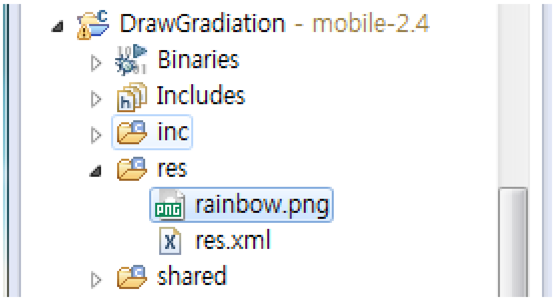
```
    {
        //ad->imgs[0] = create_gradient_rect(ad);
        //elm_box_pack_end(box, ad->imgs[0]);
        for (unsigned i = 0; i < 4; i++)
        {
            ad->imgs[i] = create_gradient_rect(ad, i);
            elm_box_pack_end(box, ad->imgs[i]);
        }
    }

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

Run the example again. Four gradated squares are displayed on the screen.



3) Creating a Rainbow Square Using an Image File

In this section, we are going to display a rainbow square on the screen using an Image file. Copy the rainbow.png file under the appendix's /Image folder to the /res folder of the source project.



Now, we need to let source code load the Image file and assign it to the Image object. Go to the source file and add a new function on top of the create_base_gui() function.

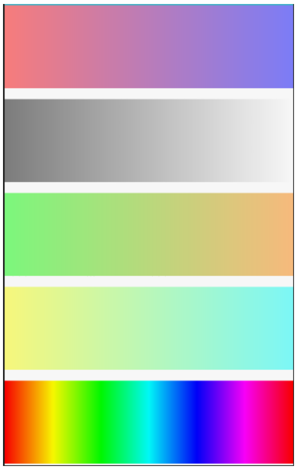
```
static Evas_Object *
create_rainbow_rect(appdata_s *ad)
{
    /* A much simpler method for gradients is to simply use an image from disk *
/
    Evas_Object *img;
    char path[PATH_MAX];

    /* 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() is an API that, by specifying the path of an image file for an Image object, loads the file.

We are now going to load and display the Image file on the screen using the function above. Add new code to the create_base_gui() function.

```
    {
        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]);
    }
```

Run the example again, and you will now see a rainbow square displayed at the bottom of the screen.



4) Related APIs

Evas *evas_object_evas_get(Evas_Object *obj): an API that creates an Evas object. / parameters: Window object.

evas_object_image_filled_add() is an API that creates an Image object.

evas_object_image_colorspace_set() is an API that specifies the color space of an Image object. Passing EVAS_COLORSPACE_ARGB8888 to the second parameter lets one pixel consist of four different data (Red, Green, Blue, and Alpha).

evas_object_image_size_set() is an API that specifies the size of an Image object. Passing 255 to the second parameter makes the number of horizontal pixels be 255. Passing 1 to the third parameter makes the number of vertical pixels be 1.

evas_object_image_alpha_set() is an API that specifies whether to apply semi-transparency to an Image object.

evas_object_image_data_get() is an API that returns the original data of an Image object as an array.

evas_object_image_data_set() performs the opposite function of evas_object_image_data_get(). It is an API that specifies original data for an Image object.

evas_object_image_file_set() is an API that, by specifying the path of an image file for an Image object, loads the file.

32. Displaying a Square on the Canvas

To draw a shape on the screen, you need to use a canvas. Evas is the canvas provided in EFL. All shapes drawn on Evas get created as objects. In this example, we are going to learn how to draw a square on the canvas.

1) Creating a Canvas and Drawing a Square

Create a new source project and specify the project name as 'DrawRect.' After the source project is created, open the source file (~.c) under the src folder and add new code to the create_base_gui() function. Label will not be used in this example, so annotate it.

```
    /* 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_E
XPAND);
    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_EX
PAND);
        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() is an API that creates a Grid container. Grid is a container that enables placing objects on the screen based on the aspect ratio. The difference with Table is that Table divides areas by cell and that the maximum value is specified as 100 by default.

elm_grid_pack() is an API that places objects in a Grid container. The first parameter indicates the Grid; the second parameter indicates the object; the third parameter indicates the horizontal position; the fourth parameter indicates the vertical position; the fifth parameter indicates the width; and the sixth parameter indicates the height. Passing 4 to the third parameter places the object in the position that is 4% in the horizontal direction.

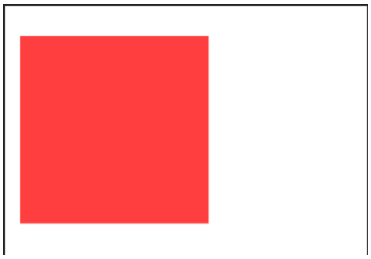
elm_grid_size_set(obj, w, h) is an API that specifies the size of a Grid. The default size of a Grid is 100 in width and 100 in height.

evas_object_evas_get(Evas_Object *) is an API that creates an Evas object.

evas_object_rectangle_add(Evas *) is an API that creates a Rectangle object on a canvas.

evas_object_color_set(Evas_Object *, int, int, int, int) is an API that specifies a color for a shape. The parameters listed in order are Red, Green, Blue, and semi-transparency. Entering 255, 0, 0, 192 creates a semi-transparent red.

Build and run the example. A pink-colored square is displayed on the screen. We specified red for the square. However, because semi-transparency has been applied to the square, it looks pink.



2) Overlaying Two Semi-Transparent Squares

In this section, we are going to see how the colors change when we add two squares and then overlay them. Add new code to the create_base_gui() function.

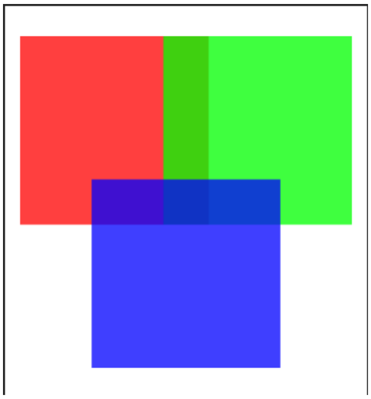
```
            /* Rect-1 */
            Evas_Object *rect = evas_object_rectangle_add(canvas);
            evas_object_color_set(rect, 255, 0, 0, 192);
            evas_object_show(rect);
            elm_grid_pack(grid, rect, 4, 5, 52, 31);

            /* 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);
        }
    }
```

We specified the color of the second square as a semi-transparent green.

We specified the color of the third square as a semi-transparent blue.

Run the example again. Three squares are displayed on the screen.

The colors of the overlaid areas change to their intermediate colors because semi-transparency has been applied.



3) Related APIs

Evas *evas_object_evas_get(Evas_Object *obj): an API that creates an Evas object. / parameters: Window object.

Evas_Object *evas_object_rectangle_add(Evas *e): an API that creates a Rectangle object on a canvas.

void evas_object_color_set(Evas_Object *obj, int r, int g, int b, int a): an API that specifies a color for a shape. / parameters: shape object, Red, Green, Blue, and semi-transparency. The allowed range of color values is 0 to 255. For example, if you want to create the Yellow, enter 255,255,0,255.

33. Displaying Polygons on the Canvas

To draw a shape on the screen, you need to use a canvas. Evas is the canvas provided in EFL. All shapes drawn on Evas get created as objects. In this example, we are going to learn how to draw polygons on the canvas.

1) Creating a Canvas & Drawing a Triangle

Create a new source project and specify the project name as 'DrawPolygon.' After the source project is created, open the source file (~.c) under the src folder and add new code to the create_base_gui() function. Label will not be used in this example, so annotate it.

```
    /* 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_E
XPAND);
    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);
```

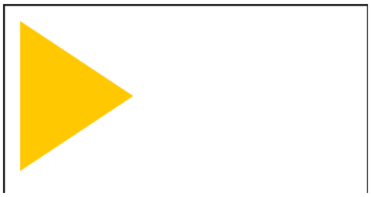
evas_object_evas_get(Evas_Object *) is an API that creates an Evas object.

evas_object_polygon_add(Evas *) is an API that creates a Polygon object on a canvas.

evas_object_polygon_point_add(Evas_Object *, Evas_Coord, Evas_Coord) is an API that adds point coordinates to a Polygon object. A Polygon must have at least three points. The first parameter indicates the Polygon object; the second parameter indicates the x-coordinate; and the third parameter indicates the y-coordinate.

evas_object_color_set(Evas_Object *, int, int, int, int) is an API that specifies a color for a shape. The parameters listed in order are the shape object, Red, Green, Blue, and semi-transparency.

Build and run the example. A yellow triangle is displayed on the screen.



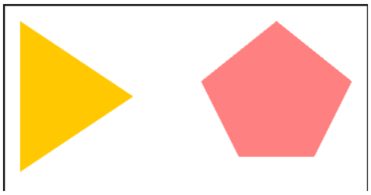
2) Displaying a Pentagon

Adding four points to a Polygon object creates a square while adding five points creates a pentagon. Add new code to the create_base_gui() function.

```
    /* 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);
}
```

We created a new Polygon object and added five points to it.

Run the example again, and you will now see a pink pentagon displayed on the screen.



3) Drawing a Regular Polygon Using a Polygon

Regular polygons are shapes all of whose angles are equal in measure and all of whose sides have the same length, such as squares and regular hexagons. In this section, we are going to create a function that creates a regular polygon. Because we will use mathematical functions, include the library at the top of the source file.

```
#include "drawpolygon.h"
#include <math.h>
```

Create a new function on top of the create_base_gui() function. This function creates a regular polygon.

```
static Evas_Object*
```

```
crate_circle(Evas* canvas, int x, int y, int radius, int r, int g, int b, int a, int edge_cou
nt)
{
    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 is a constant that contains the Pi value.

sin(double) is an API that calculates sine. The unit of angle is Pi. For example, to specify the sine as 180 degrees, pass Pi, and to specify the sine as 90 degrees, pass Pi /2.

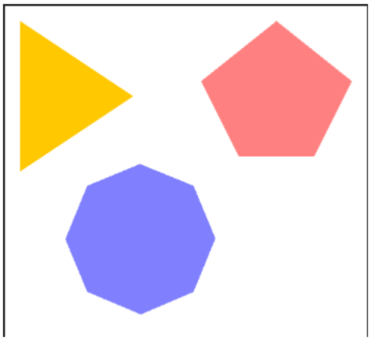
cos(double) is an API that calculates cosine. The unit of angle is Pi.

We are now going to create a regular octagon using the functions above. Add a new line of code at the end of the create_base_gui() function.

```
        /* 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);
    }
```

The create_circle() function's parameters listed in order are the Evas object, center x-coordinate, center y-coordinate, radius, Red, Green, Blue, semi-transparent color, and number of points.

Run the example again, and you will now see a purple regular octagon displayed on the screen.



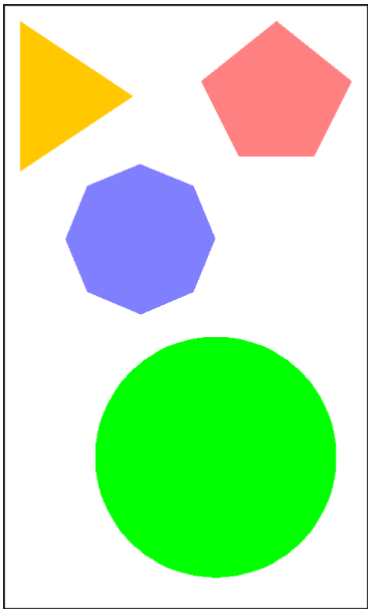
4) Drawing a Circle Using the Polygon Widget

If you increase the number of points of a regular polygon, it will eventually look like a circle. Add a new line of code at the end of the create_base_gui() function.

```
        /* 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);
    }
```

We added 90 points to a polygon. We are now going to see the result of this addition.

Run the example again, and you will now see a yellow-green regular 90-angle polygon displayed on the screen. Due to the polygon’s great number of points, it looks like a circle.



5) Related APIs

Evas *evas_object_evas_get(Evas_Object *obj): an API that creates an Evas object. / parameters: Window object.

Evas_Object *evas_object_polygon_add(Evas *e): is an API that creates a Polygon object on a canvas.

void evas_object_polygon_point_add(Evas_Object *obj, Evas_Coord x, Evas_Coord y): an API that adds point coordinates to a Polygon object. / parameters: Polygon, x-coordinate, and y-coordinate.

void evas_object_color_set(Evas_Object *obj, int r, int g, int b, int a): an API that specifies a color for a shape. / parameters: shape object, Red, Green, Blue, and semi-transparency. The allowed range of color values is 0 to 255. For example, if you want to create the Yellow, enter 255,255,0,255.

34. Displaying Text on the Canvas

To draw a shape on the screen, you need to use a canvas. Evas is the canvas provided in EFL. All shapes drawn on Evas get created as objects. In this example, we are going to learn how to display a string on a canvas.

1) Displaying Text on the Canvas

Create a new source project and specify the project name as 'CanvasTextColor.' After the source project is created, open the source file (~.c) under the src folder and add a text object creating function on top of the create_base_gui() function.

```
// 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 *) is an API that creates a text object on a canvas.

evas_object_text_text_set(Evas_Object *, char *) is an API that specifies a string for a text object.

evas_object_text_font_set(Evas_Object *, char *, Evas_Font_Size) is an API that specifies a font for a text object. The first parameter indicates the text object, the second parameter indicates the font type, and the third parameter indicates the font size.

We are now going to display text on the canvas using the function we just created. Add new code to the create_base_gui() function. Label will not be used in this example, so annotate it.

```
    /* 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_E
XPAND);
    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);
```

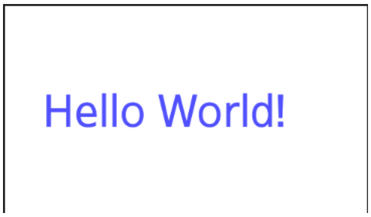
We created a Grid container to assign relative coordinates to the text object.

elm_grid_size_set() is an API that changes the size of a Grid. The default width of a Grid is 100, and the default height is 100 as well. We changed the size of a Grid to 480 x 800 in the code above. This new setting enables more precise placement than the default setting.

Create an Evas object using the evas_object_evas_get() function.

Create a text object using the create_text() function. The parameters listed in order are the canvas object, Grid, x-coordinate, y-coordinate, width, height, text string, font size, Red, Green, Blue, and semi-transparent color.

Build and run the example. The text 'Hello World' is displayed on the screen in purple.



Hello World!

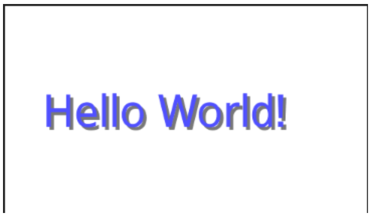
2) Applying a Shadow to Text

To apply a shadow effect to text, you need to display two texts after specifying a different color and a different position for each. Add new code to the create_base_gui() function. We created the second text with the same content and size as the first text.

```
    { /* 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);
    }
```

The second text will become the shadow and will be displayed before the first text. In addition, the position will be different, and its color is gray.

Run the example again. A shadow effect has been applied to text.



3) Related APIs

Evas *evas_object_evas_get(Evas_Object *obj): an API that creates an Evas object. / parameters: Window object.

Evas_Object *evas_object_text_add(Evas *e): an API that creates a text object on a canvas.

void evas_object_text_text_set(Evas_Object *obj, const char *text): an API that specifies a string for a text object.

void evas_object_text_font_set(Evas_Object *obj, const char *font, Evas_Font_Size size): an API that specifies a font or a text object. / parameters: text object, font type, and font size.

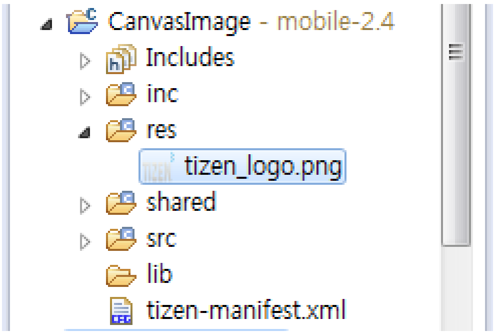
void evas_object_color_set(Evas_Object *obj, int r, int g, int b, int a): an API that specifies a color for a shape. / parameters: shape object, Red, Green, Blue, and semi-transparency. The allowed range of color values is 0 to 255. For example, if you want to create the Yellow, enter 255,255,0,255.

35. Displaying an Image on the Canvas

To draw a shape on the screen, you need to use a canvas. Evas is the canvas provided in EFL. All shapes drawn on Evas get created as objects. In this example, we are going to learn how to display an image on a canvas.

1) Displaying an Image on the Canvas

Create a new source project and specify the project name as 'CanvasImage.' In this example, we need to use an image file. Copy the tizen_logo.png file under the appendix's /Image folder to the /res folder of the source project.



To use the image file we just copied, we need to find the absolute path of the image file. After the source project is created, open the source file (~.c) under the src folder and add a new function to the create_base_gui() function. This function returns the absolute path of a file stored under the /res folder.

```
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);
    }
}
```

Go back to the create_base_gui() function and add new code. This code creates a canvas and loads an image file to create an Image object. Annotate the Label-creating code.

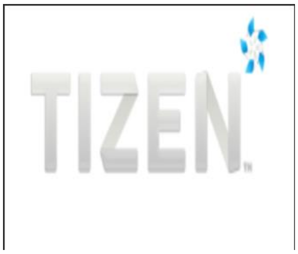
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win,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

        {
            /* 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);
```

We created a Grid container to assign relative coordinates to the Image object. To assign more precise coordinates to the image object, we specified the size of the Grid as 480x800 using the elm_grid_size_set() function.

evas_object_image_filled_add(Evas *) is an API that creates an Image object that lets an original image fill the entire space of an area.

evas_object_image_file_set(Evas_Object *, char *, char *) is an API that loads an image file into an Image object.

Build and run the example. An image is displayed on the screen. You will now see the original image is long horizontally but also has been stretched vertically.



2) Displaying an Image on the Canvas in Tile Format

In this section, we are going to learn how to place images in a given area in a tile format. Add new code at the end of the create_base_gui() function. This code creates a second Image object.

```
/* 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 */
```

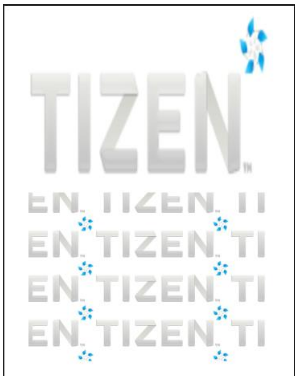
```
            int w, h;
            img = evas_object_image_add(canvas);
            evas_object_image_file_set(img, img_path, NULL);
            evas_object_image_size_get(img, &w, &h);
            evas_object_image_fill_set(img, 110, 37, w, h);
            elm_grid_pack(grid, img, 40, 310, 400, 280);
            evas_object_show(img);
        }
    }
```

evas_object_image_add(Evas *) is an API that creates an Image object that displays an image in a tile format. The size and start position of the image need to be specified.

evas_object_image_size_get(const Evas_Object *, int *, int *) is an API that returns the size of an original image. The first parameter receives the Image object; the second parameter receives the width of an original image; and the third parameter receives the height of an original image.

evas_object_image_fill_set(Evas_Object *, Evas_Coord, Evas_Coord, Evas_Coord, Evas_Coord) is an API that specifies the display position and size of an original image for an Image object. The parameters listed in order are the Image object, x-coordinate, y-coordinate, width, and height.

Run the example again. The image is placed in a tile format.



3) Displaying an Image as Large as Possible Retaining the Original Proportions

In this section, we are going to learn how to display an image as large as possible in a given area. To display an image retaining its original proportions, a calculation needs to be performed. Create a new function on top of the create_base_gui() function. This function creates an Image object by calculating the size of the image to be displayed.

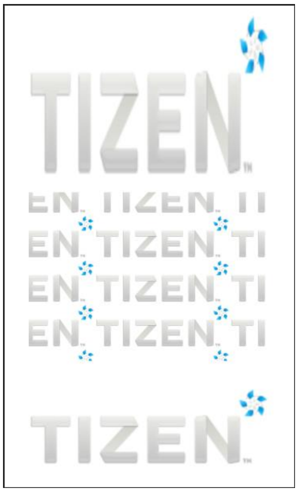
```
// Create IMAGE object. Source image's rate is no change
static Evas_Object *
create_image(Evas *canvas, Evas_Object *grid, const char *img_path, int x, int y, int
w, int h)
{
    int source_w, source_h, new_x, new_y, new_w, new_h;
    float rate_h, rate_v, rate;

    // 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;
}
```

Now, let's create a third Image object by calling the function we just created. Add new code at the end of the create_base_gui() function.

```
        /* 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);
    }
}
```

The parameters of the create_image() function listed in order are the canvas object, path of the image file, x-coordinate, y-coordinate, width, and height.

Run the example again. A third image is displayed at the bottom of the screen, and the original proportions of the original image have been retained.



4) Related APIs

Evas *evas_object_evas_get(Evas_Object *obj): an API that creates an Evas object. / parameters: Window object.

Evas_Object *evas_object_image_filled_add(Evas *e): an API that creates an Image object that lets an original image fill the entire space of an area.

void evas_object_image_file_set(Evas_Object *obj, const char *file, const char *key): an API that loads an image file into an Image object.

Evas_Object *evas_object_image_add(Evas *e): an API that creates an Image object that displays an image in a tile format. The size and start position of the image need to be specified.

void evas_object_image_size_get(const Evas_Object *obj, int *w, int *h): an API that returns the size of an original image. The first parameter receives the Image object; the second parameter receives the width of an original image; and the third parameter receives the height of an original image.

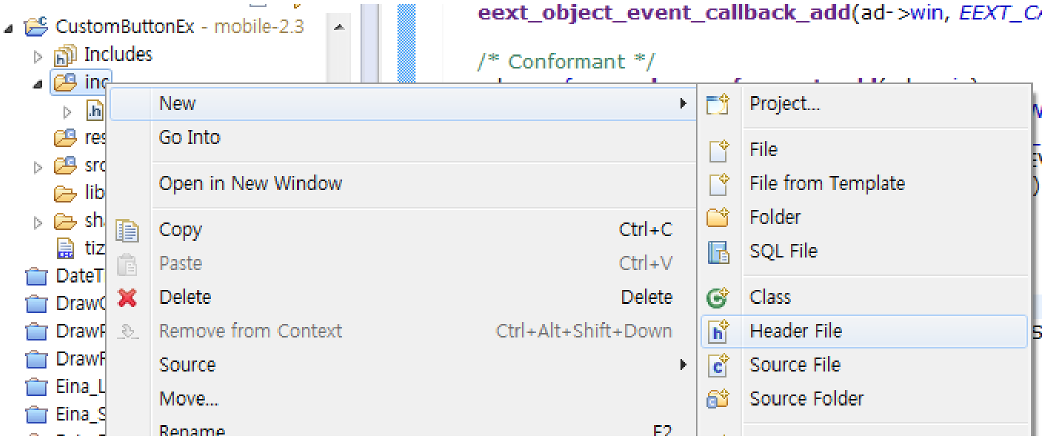
void evas_object_image_fill_set(Evas_Object *obj, Evas_Coord x, Evas_Coord y, Evas_Coord w, Evas_Coord h): an API that specifies the display position and size of an original image for an Image object. / parameters: Image object, x-coordinate, y-coordinate, width, and height.

36. Creating a Customized Button

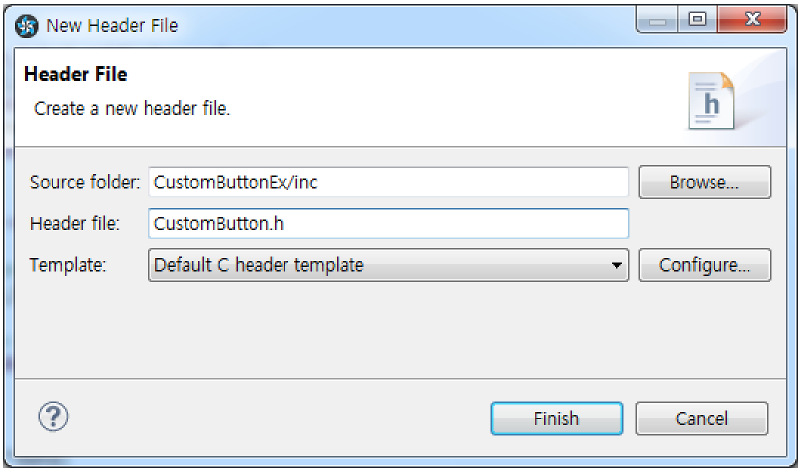
Basic widgets provided by a system are often simply not enough to satisfy users' various demands. In such cases, it is necessary to create customized widgets yourself. In this example, we are going to create a customized Button widget.

1) Displaying Text on the Canvas

Create a new source project and specify the project name as 'CustomButtonEx.' After the source project is created, create a library file for the customized Button: Right-click the /inc folder and select [New > Header File] in the shortcut menu.



When a popup window appears, enter CustomButton.h in the header file field and tap the Finish button.



Source folder: CustomButtonEx/inc

Header file: CustomButton.h

Template: Default C header template

We are now going to implement a feature that displays a background square in the customized widget. When the /inc/CustomButton.h file is created, add source code to it. This code declares a library header file, defines data structures, and creates a background square.

```
#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 is a data structure used in a customized widget. rect indicates the coordinates of a widget in an area. text stores the caption text string.

create_rect() is a function that creates a Rect object on a canvas. The parameters listed in order are the canvas object, x-coordinate, y-coordinate, width, height, Red, Green, Blue, and semi-transparent color.

evas_object_rectangle_add(Evas *) is an API that creates a Rectangle object on a canvas.

evas_object_color_set(Evas_Object *, int, int, int, int) is an API that specifies a color for a shape. The parameters listed in order are Red, Green, Blue, and semi-transparency. Entering 255, 0, 0, 192 creates a semi-transparent red.

create_button() is a function that creates a customized Button widget. We need to make this function able to be called from the outside. The parameters listed in order are the canvas, x-coordinate, y-coordinate, width, height, caption text, and name of the callback event function.

malloc(size_t) is an API that procures and then returns a specified size of memory.

sizeof() is an API that requests and then returns the memory size of an object.

Now, let's create a customized widget object on the main screen. Open the source file (~.c) under the src folder and add new code.

```
#include "custombuttonex.h"
#include "custombutton.h"

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

```
buttondata_s* m_bd1;
```

Include the customized Button widget file and create the buttondata structure as a global variable.

Then, go to the create_base_gui() function and add new code. This code creates a canvas object and a customized Button widget. Annotate the Label-creating code.

```
    /* 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_E
XPAND);
    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_EX
PAND);
```

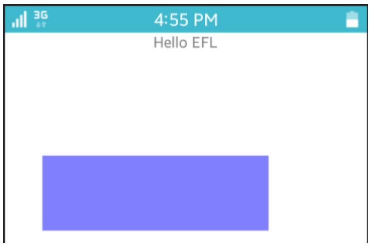
```
        elm_object_content_set(ad->conform, grid);
        evas_object_show(grid);

        {
            /* 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);
```

We created a Grid container to assign relative coordinates to the object. We changed the size of the Grid to 480x800 using the elm_grid_size_set() function.

evas_object_evas_get(Evas_Object *) is an API that creates an Evas object.

create_button() is a function that actually creates the customized Button we just created in the customized Button widget file (CustomButton.h).

Build and run the example. A sky blue square is displayed on the screen. This is the background square for the customized Button .



2) Displaying Caption Text

In this section, we are going to display caption text on a square background. To display text, we will use TextBlock. TextBlock is a canvas object that makes it possible to specify the properties of text such as font size and color using HTML tags. Go back to the CustomButton.h file and add a new function on top of the create_button() function.

```
// 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 *) is an API that creates a TextBlock object.

Evas_Textblock_Style is a style structure that stores the property information of TextBlock.

evas_textblock_style_new() is an API that creates an Evas_Textblock_Style object.

evas_textblock_style_set(Evas_Textblock_Style *, char *) is an API that specifies a style for Evas_Textblock_Style. Pass the Evas_Textblock_Style object to the first parameter. For the second parameter, specify text properties using HTML tags.
font=Sans is a tag that specifies the font as Sans serif/Gothic.
font_size=50 is a tag that specifies the font size as 50.
color=#eee is a tag that specifies the font color as gray.
align=center is a tag that specifies the horizontal alignment as center.

evas_object_textblock_style_set(Evas_Object *, Evas_Textblock_Style *) is an API that specifies a style for a TextBlock object. Pass the TextBlock object to the first parameter, and pass the Evas_Textblock_Style object to the second parameter.

evas_textblock_style_free(Evas_Textblock_Style *) is an API that deletes an Evas_Textblock_Style object.

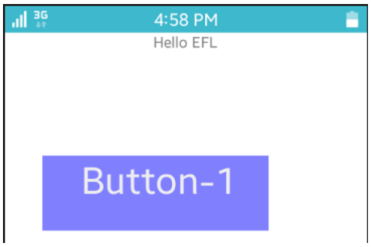
evas_object_textblock_text_markup_set(Evas_Object *, const char *) is an API that specifies a text string for a TextBlock object.

We need to make this function be called by the create_textblock() function. Add new code to the create_textblock() function.

```
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;
```

The parameters of the create_textblock() function listed in order are the canvas object, x-coordinate, y-coordinate, width, height, and text string.

Run the example again. Caption text is displayed in the square area.



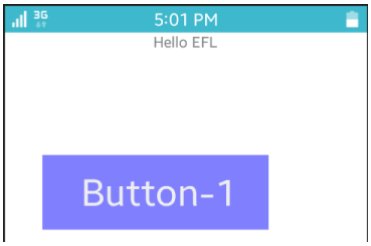
3) Center-Aligning Caption Text

Currently, the caption text is displayed at the top of the square. We are now going to change the position of the caption text to center. Add new code to the create_textblock() function.

```
    Evas_Textblock_Style *st = evas_textblock_style_new();
    evas_textblock_style_set(st, "DEFAULT='font=Sans font_size=50 color=#eee wrap=m
ixed 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) is an API that specifies the vertical alignment of TextBlock caption text. Passing 0 to the second parameter top-aligns the text, passing 1 bottom-aligns the text, and passing 0.5 center-aligns the text.

Run the example again. This time, the caption text is displayed in the center.



4) Requesting a Button Click Event

In this section, we are going to implement a feature that changes the background color when a customized Button is tapped. To do so, we need to request a Touch event. Add new code at the end of the create_button() function.

```
    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_mouse_up, (void*)bd);

    return bd;
```

This code calls the on_mouse_down function when a Touch Down event occurs in TextBlock.

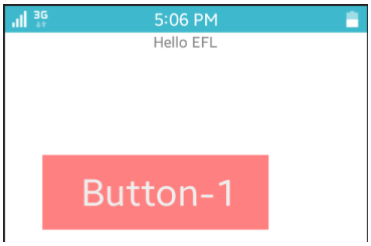
This code calls the on_mouse_up function when a Touch Cancel event occurs in TextBlock.

We are now going to create the callback functions. Add two new functions on top of the create_button() function.

```
// 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);
}
```

This code changes the background color to pink when the user Touch Downs a customized Button and changes the background color back to sky-blue when the user Touch Cancels the customized Button.

Run the example again. Tap and then untap the Button. The background color then changes.



```
}
```

5) Requesting a Click Event on the Main Screen

To request a Button click event on the main screen, you need to let a callback function be called when a Touch Cancel event occurs. Add new code at the end of the create_button() function.

```
   evas_object_event_callback_add( bd->text, EVAS_CALLBACK_MOUSE_DOWN, on_mo
use_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, (v
oid*)bd);

   return bd;
```

When the user taps and then untaps the Button, the function passed is displayed on the main screen.

Go to the main source file (custombuttonex.c) and modify the code of the create_base_gui() function.

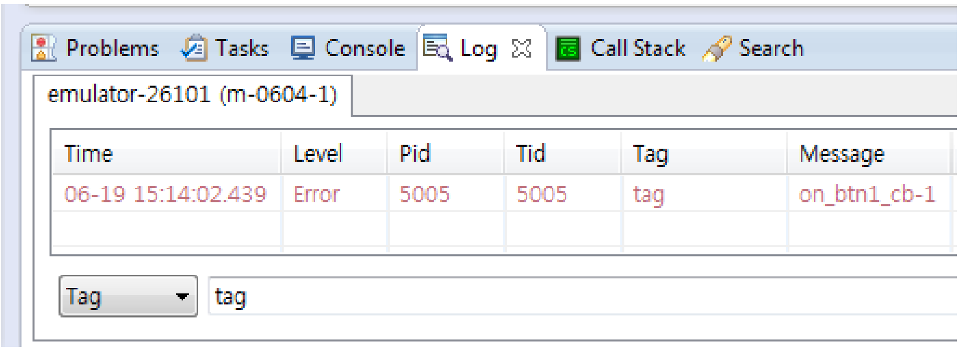
```
    {
        /* Canvas */
        Evas* canvas = evas_object_evas_get(ad->conform);

        /* 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_b
tn1_cb);
    }
```

We specified the name of the callback function for the customized Button as on_btn1_cb. Now, we need to add the callback function. Create a new function on top of the create_base_gui() function.

```
static void
on_btn1_cb(void *data, Evas *e, Evas_Object *button, void *event_info)
{
    dlog_print(DLOG_ERROR, "tag", "on_btn1_cb-1");
}
```

This code displays a log message when the user taps the customized Button.

Run the example again. Click the Button, and then the message 'on_btn1_cb-1' is displayed in the Log pane.



Time	Level	Pid	Tid	Tag	Message
06-19 15:14:02.439	Error	5005	5005	tag	on_btn1_cb-1

Tag ▾ tag

6) Changing Caption Text

In this section, we are going to implement a feature that changes the caption text of a first Button when a newly added second customized Button is tapped. To do so, we need to add a text-changing function to the custom widget file (CustomButton.h). Where the function gets placed does not really matter. However, it is advisable that you place a function called from the outside at the bottommost possible position. This is because it is possible another function may be called from this function.

```
void set_button_text(buttondata_s* bd, const char* str)
{
   evas_object_textblock_text_markup_set(bd->text, str);
}

#endif /* CUSTOMBUTTON_H_ */
```

When this function is called on the main screen, the caption text of the relevant Button is changed. Go to the main source file (custombuttonex.c) and add new code at the end of the create_base_gui() function.

```
    {
        /* 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_btn1_cb);
        /* Custom Button-2 */
        create_button(canvas, grid, 50, 400, 300, 100, "Button-2", on_btn2_cb);
    }
```

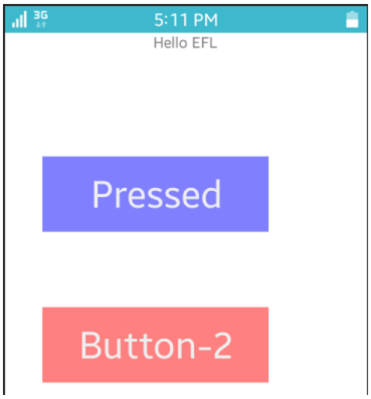
This code creates a second customized Button, specifies the caption text as "Button-2", and specifies the callback function as on_btn2_cb.

Lastly, we need to add the callback function for the second Button. Create a new function on top of the create_base_gui() function.

```
static void
on_btn2_cb(void *data, Evas *e, Evas_Object *button, void *event_info)
{
    set_button_text(m_bd1, "Pressed");
}
```

This code changes the caption text of the first Button to ‘Pressed’ when the user taps the second Button.

Run the example again. Tap the second Button, and the caption text of the first Button is changed.



7) Related APIs

Evas_Object *evas_object_textblock_add(Evas *e): an API that creates a TextBlock object.

Evas_Textblock_Style: a style structure that stores the property information of TextBlock.

Evas_Textblock_Style *evas_textblock_style_new(): an API that creates an Evas_Textblock_Style object.

void evas_textblock_style_set(Evas_Textblock_Style *ts, const char *text): an API that defines the style of Evas_Textblock_Style. / parameters: Evas_Textblock_Style object and HTML tags.

void evas_object_textblock_style_set(Evas_Object *obj, Evas_Textblock_Style *ts): an API that specifies a style for a TextBlock object. / parameters: TextBlock object and Evas_Textblock_Style object.

void evas_textblock_style_free(Evas_Textblock_Style *ts): an API that deletes an Evas_Textblock_Style object.

void evas_object_textblock_text_markup_set(Evas_Object *obj, const char *text): an API that specifies a string for a TextBlock object.

void evas_object_textblock_valign_set(Evas_Object *obj, double align): an API that specifies the vertical alignment of TextBlock caption text. Passing 0 to the second parameter top-aligns the text, passing 1 bottom-aligns the text, and passing 0.5 center-aligns the text.

37. Using an Animator

Using the Animator enables making a change to an object on the screen at certain intervals. The Animator is similar to the Timer, but its difference with the Timer is that it can produce various effects. Let's learn in detail how to use the GenList widget through an example.

1) Creating an Animation

Create a new source project and specify the project name as 'AnimatorEx.' After the source project is created, open the source file (~.c) under the src folder and add variables at the top of the source file.

```
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 and rect2 are square objects to which an animation will be applied.

anim_continue is a Boolean variable that determines whether to continue the animation.

Ecore_Pos_Map is an option that specifies the style of an animation. We will use it in a TimeLine animation.

We are now going to create a first square object and an Animator object. Add new code to the create_base_gui() function. Annotate the Conformant-creating code and the Label-creating code.

```
    /* 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*/
    /*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);*/

    /* 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);
```

Create an Evas object to create a Rectangle object.

ecore_animator_add(Ecore_Task_Cb, void *) is an API that creates an Animator object. The first parameter indicates the frame event callback function, and the second parameter indicates user data. It generally passes an object or appdata to which you will apply the animation.

ecore_animator_frametime_set(double) is an API that specifies a frame time interval for an animation. For example, if you specify the interval as 1/60, 60 frame events occur for one second. The unit is seconds.

We are now going to create a frame event function. Create a new function on top of the create_base_gui() function.

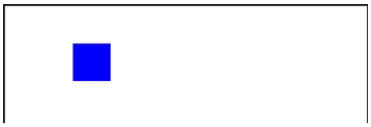
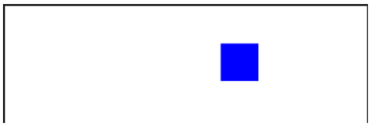
```
static Eina_Bool on_next_frame1(void *data)
{
   static int x = 0;
   if (x >= 350)
```

```
        x = 0;
    evas_object_move(data, x += 2, 50);
    return anim_continue;
}
```

evas_object_move() is an API that changes the position of an object. This function increases the x-coordinate of an object by 2 at every frame. When the x-coordinate becomes greater than 350, it starts back from 0.

If the value returned from the frame event function is ECORE_CALLBACK_RENEW, the animation continues. If the value ECORE_CALLBACK_CANCEL is returned, the animation pauses.

Build and run the example. A blue square moves from left to right. When the blue square has moved a certain distance, it restarts its movement from the left end of the screen.

2) Stopping an Animation

To stop an animation, the frame event function needs to return ECORE_CALLBACK_CANCEL. We are now going to implement a feature that stops an animation when a Button is tapped. Create a new function on top of the create_base_gui() function. This function adds a widget to a Table container.

```
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);
}
```

Then, add new code to the create_base_gui() function. This code creates a Box and a Table and adds a 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_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, 20 * elm_config_scale_get(), 10 * 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_E
XPAND);
        /* 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);
```

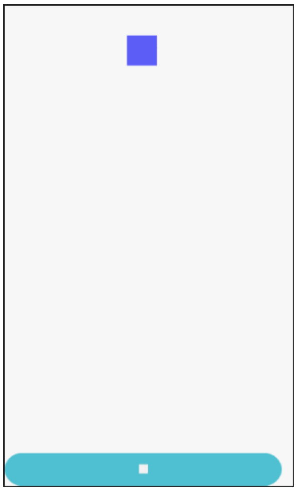
evas_object_raise() is an API that moves an object to the topmost level of the screen so that it is not hidden by other objects.

We need to make it so that when the Button is tapped, ECORE_CALLBACK_CANCEL gets stored in anim_continue. We are now going to add the callback function for the Button on top of the create_base_gui() function.

```
static void
btn_stop_cb(void *data, Evas_Object *obj, void *event_info)
{
    anim_continue = ECORE_CALLBACK_CANCEL;
}
```

We made it so that when the Button is tapped, the on_next_frame1() returns ECORE_CALLBACK_CANCEL, and as a result, the animation stops.

Run the example again. Tap the Button, and the square stops its movement.



3) Pausing/Resuming an Animation with the Timer Widget

To pause an animation, you need to use the ecore_animator_freeze() function. To resume an animation, you need to use the ecore_animator_thaw() function. Add new code to the create_base_gui() function. This code creates two Timers.

```
/* 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);
```

Two Timers have now been created. The first Timer is responsible for pausing an animation, while the second Timer is responsible for resuming an animation.

ecore_timer_delay() is an API that starts a Timer after a certain amount of time has passed. In the case of the second Timer, the event occurs after 3 + 6 = 9 seconds, and afterwards, an event occurs every 6 seconds.

We are now going to create a timer event function. Add two new functions on top of the create_base_gui() function.

```
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;
}
```

The freeze_anim() function will be called 3 seconds after the app starts.

ecore_animator_freeze(Ecore_Animator *) is an API that pauses an animation.

The thaw_anim() function will be called 6 seconds after the app starts.

ecore_animator_thaw(Ecore_Animator *) is an API that resumes an animation.

Run the example again, and you will now see a moving animation pause and resume every 3 seconds.

4) TimeLine Animation

In this section, we are going to implement an animation that plays for a certain amount of time. Add new code to the create_base_gui() function. This code creates a second Rectangle object and a TimeLine animation.

```
/* 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 *) is an API that creates a TimeLine animation. The first parameter indicates the play time, and the unit is seconds. The second parameter is the function name of the frame event, and the third is the user data.

We are now going to apply to the second square an animation that changes the position, size, and color of the square. Add new code on top of the create_base_gui() function. This is the TimeLine animation frame event function.

```
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() is an API that returns the resulting value that is mapped onto the position of the current animation. Returned values range between 0 and 1. The resulting value when the animation starts is 0, and it gradually increases and eventually reaches 1 when the animation stops. The second parameter of the Timeline animation event function must be passed to the first parameter. For the second parameter, enter the style of the animation. The style types are as follows:

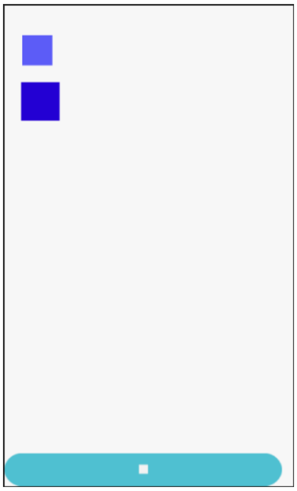
- ECORE_POS_MAP_LINEAR, /**< Linear 0.0 -> 1.0 */
- ECORE_POS_MAP_ACCELERATE, /**< Start slow then speed up */
- ECORE_POS_MAP_DECELERATE, /**< Start fast then slow down */
- ECORE_POS_MAP_SINUSOIDAL, /**< Start slow, speed up then slow down at the end */
- ECORE_POS_MAP_ACCELERATE_FACTOR, /**< Start 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 */

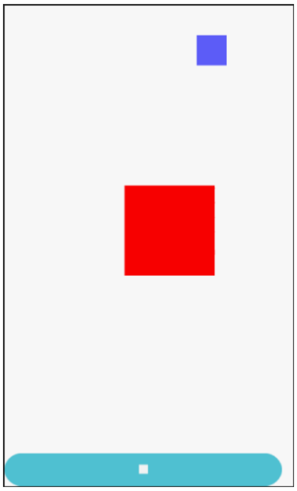
- ECORE_POS_MAP_DIVISOR_INTERP, /**< Start at gradient * v1, interpolated via power of v2 curve */
- ECORE_POS_MAP_BOUNCE, /**< Start at @c 0.0 then "drop" like a ball bouncing to the ground at @c 1.0, and bounce v2 times, with decay factor of v1 */
- ECORE_POS_MAP_SPRING /**< Start at @c 0.0 then "wobble" like a spring with rest position @c 1.0, and wobble v2 times, with decay factor of v1 */

For the third parameter of the ecore_animator_pos_map() function, enter the intensity of speed change. For the fourth parameter, enter the tempo of speed change.

The following code specifies the size, position, and color of a square.

Run the example again, and you will now see a blue square turn red and become larger as it moves to the bottom right.





5) Applying Acceleration to a TimeLine Animation

The style applied to the animation we just created is ECORE_POS_MAP_LINEAR. It is an option that keeps the animation moving at a certain speed. We are now going to apply an option that makes the animation move slowly at first and then gradually increase in speed. Add new code to the create_base_gui() function. This code creates a new 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->rect
2);
        my_table_pack(table, btn, 0, 1, 1, 1);
    }
}
```

Next, we will create a callback function for the second Button. Create a new function on top of the create_base_gui() function.

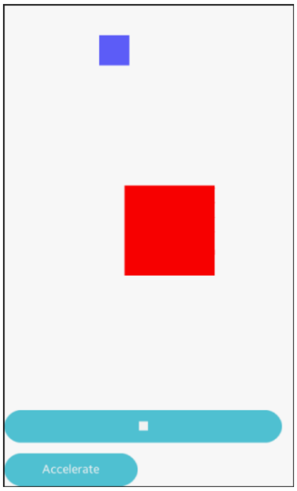
```
static void
```

```
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);
}
```

We have changed the animation style to ECORE_POS_MAP_ACCELERATE. This is the acceleration option.

We created a Timeline animation using the ecore_animator_timeline_add() function.

Run the example again and tap the Accelerate button. This starts the animation.



6) Other Animation Styles

We are going to apply different styles to the Timeline animation. Add four Buttons to the create_base_gui() function.

```
        /* 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->rect2);
        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->rect
2);
        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->rect
2);
        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->rect2);
        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->rect2);
        my_table_pack(table, btn, 0, 3, 1, 1);
    }
}
```

You need four matching Button callback functions as well. Add four new functions on top of the create_base_gui() function.

```
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);
}
```

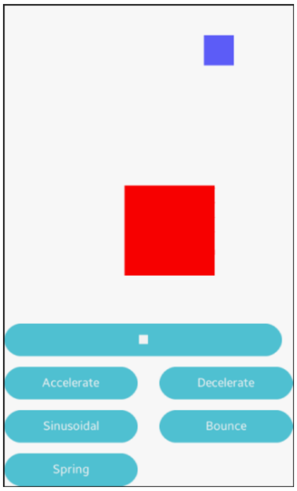
ECORE_POS_MAP_ACCELERATE is the opposite property of ECORE_POS_MAP_ACCELERATE. It creates movement that is fast at first, but gradually loses speed.

ECORE_POS_MAP_SINUSOIDAL creates movement that starts slow, becoming faster before becoming slow again at the end.

ECORE_POS_MAP_BOUNCE creates vibration similar to a bouncing ball. It does not exceed the ending point, however.

ECORE_POS_MAP_SPRING creates vibration like a bouncing ball. It slows to stop after bouncing off the ending point.

Run the example again and tap the Buttons you added, one at a time.



7) Continuous Animations

We will implement a feature that automatically starts the second animation after one type of animation ends, using the Timer. Create a new Button on the create_base_gui() function.

```
            /* 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->rect2);
            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->rect
2);
            my_table_pack(table, btn, 1, 3, 1, 1);
        }
    }
```

Then, add two new functions on top of the create_base_gui() function.

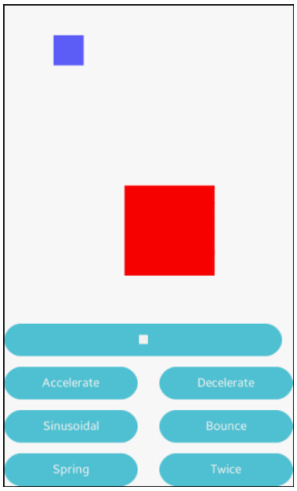
```
static Eina_Bool
start_second_anim(void *data)
{
   pos_map = ECORE_POS_MAP_SPRING;
   ecore_animator_timeline_add(4, on_next_frame2, data);
   return ECORE_CALLBACK_CANCEL;
}
```

```
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() is the function that starts the second animation.

btn_twice_cb() is the callback function of the seventh Button. Start the acceleration animation and use the Timer to enable the second animation to begin automatically 4 seconds later.

Run the example again and tap the 'Twice' button. The acceleration animation begins and ends, followed by the Spring animation.



8) Related APIs

void ecore_animator_frametime_set(double frametime): specifies the time frame between the animations. / parameters: time interval of the frame (in seconds).

Ecore_Animator *ecore_animator_add(Ecore_Task_Cb func, void *data): creates an Animator object. / parameters: callback function for the frame event, user data. It generally passes an object or appdata to which you will apply the animation.

void ecore_animator_freeze(Ecore_Animator *animator): pauses the animation.

void ecore_animator_thaw(Ecore_Animator *animator): restarts the animation.

Ecore_Animator *ecore_animator_timeline_add(double runtime, Ecore_Timeline_Cb func, void *data): creates a Timeline animation. / The first parameter is the run time in seconds. The second parameter is the function name of the frame event, and the third is the user data.

EAPI double ecore_animator_pos_map(double pos, Ecore_Pos_Map map, double v1, double v2): returns the resulting value mapped onto the current animation's position. The returned value is between 0 and 1. The resulting value when the animation starts is 0, and it gradually increases, eventually reaching 1 when the animation stops. Simply pass the second parameter of the Timeline animation event function to the first parameter. For the second parameter, specify Ecore_Pos_Map. For the third parameter, specify the intensity. For the fourth parameter, specify the rate of speed change.

The types of Ecore_Pos_Map are as follows:

- 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 */

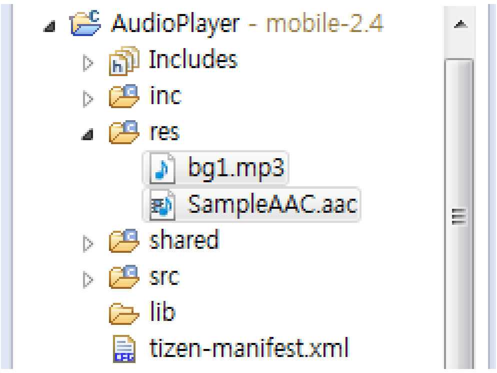
38. Playing Audio

When Sony came up with a portable audio device in 1979, the market research returned negative feedback. This did not stop Sony President Orita, however; his persistence gave birth to the Walkman. Soon, it became a bestseller. In the 21st century, a man named Steve Jobs created a small MP3 player and caused a worldwide sensation. With the endless flow of smartphones, however, stand-alone portable music players are facing obsoleteness.

Using a player, you can play audio and video. In this section, we will discuss how to play an audio file.

1) Loading an Audio File

Create a new source project and specify the project name as 'AudioPlayer.' Copy the audio file. Copy two files, bg1.mp3 and SampleAAC.aac, under the /Audio folder of the appendix to the /res folder of the source project.



Now, enter the source code. Open the source file (~.c) under the src folder and add libraries and variables at the top of the screen.

```
#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 is a Player structure that plays media such as audio and video.

file_name[] is an array of strings that save the file name.

As you run the app, a Player object will be automatically created and the first audio file loaded. Add new code at the end of the create_base_gui() function.

```
    evas_object_show(ad->win);

    // Create Player
    ad->player = create_player();

    // Load audio file to Player
    prepare_player(ad, 0);
```

The create_player() function creates the Player, while the prepare_player() function loads the audio file. Now, let us start building it step by step. Add six functions on top of the create_base_gui() function.

```
// 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_state(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);
}
```

The get_player_state() function returns the current status of the Player.

player_get_state(player_h, player_state_e) is an API that returns the current status of the Player. The first parameter is the Player object, while the second parameter returns the status value. The types of player_state_e status are:

- 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() is a callback function invoked upon completion of playback.

player_stop(player_h) is an API that stops the playback.

The create_player() function creates a Player object.

player_create(player_h *) is an API that creates a Player object.

player_set_completed_cb(player_h, player_completed_cb, void *) is an API that specifies the callback function upon completion of playback. The second parameter is the name of the callback function, whereas the third parameter is the user data.

The stop_player() function stops the playback.

The get_resource_path() function returns the full path to all the files saved under the /res folder.

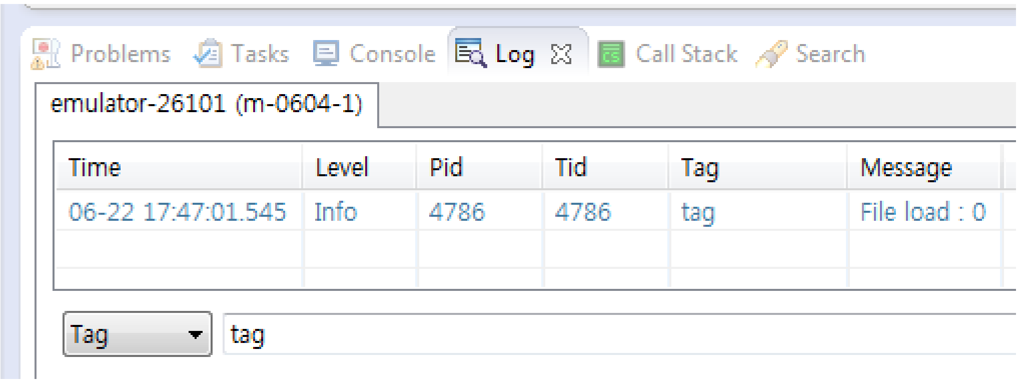
The prepare_player() function loads the audio file.

player_unprepare(player_h) is an API that closes the file you loaded to the Player.

player_set_uri(player_h, const char *) is an API that loads a file to the Player.

player_prepare(player_h) is an API that prepares for playback by the Player. When it is ready to play, it returns 0.

If you build and start running this way, you will see or hear nothing on the screen. If you check the message on the Log window, 'File load: 0' is displayed.



Time	Level	Pid	Tid	Tag	Message
06-22 17:47:01.545	Info	4786	4786	tag	File load : 0

Tag ▾ tag

2) Playing Audio

We will implement a feature that plays the audio file when the Button is tapped. Create a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
          /* 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);
}
```

Add new code to the create_base_gui() function. This code creates a Box and adds a 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);

     { /* 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_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</>");
            /* 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);
```

Let's create a callback function for the Button. Create a new function on top of the create_base_gui() function.

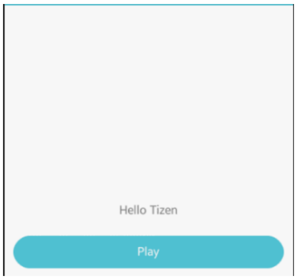
```
// Start play
static void start_player(void *data, Evas_Object *obj, void *event_info)
{
   appdata_s *ad = data;

   if( get_player_state(ad->player) != PLAYER_STATE_PLAYING)
```

```
        player_start(ad->player);
}
```

The start_player() function starts playing the audio.

player_start(player_h) is an API that starts playing the audio.

Run the example again and tap the Play button. The audio will now be played.



3) Pausing and Stopping

We will now add two Buttons to implement the Pause and Stop features. Add new code to the create_base_gui() function.

```
        /* 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);
        }
    }
```

We have added two Buttons. We are now going to create a callback function for the Buttons. Add new code on top of the create_base_gui() function.

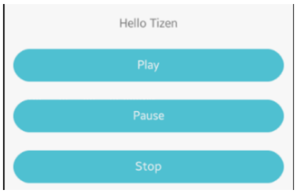
```
// Pause play
static void pause_player(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;

    if( get_player_state(ad->player) == PLAYER_STATE_PLAYING )
        player_pause(ad->player);
}
```

The start_player() function pauses the audio playback.

player_start(player_h) is an API that pauses the audio playback.

You do not need to create another callback function for the Stop button because you already have the stop_player() function in place.

Run the example again and tap the Play button to play the audio. Tap the Pause button to pause or the Play Again button to restart. Tap the Stop button to stop or the Play Again button to start playing from the beginning.



4) Changing the Audio File

We will implement a feature that changes the audio file by adding two Buttons. Add new code to the create_base_gui() function.

```
/* 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);
```

We have created two Buttons. Finally, we are ready to create a callback function for the Buttons. Add new code on top of the create_base_gui() function.

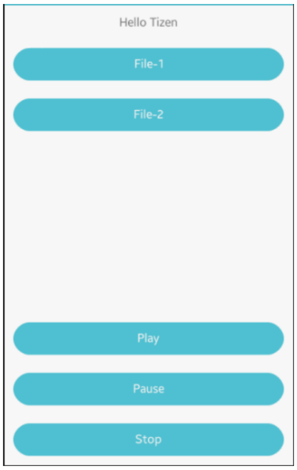
```
static void
btn_load_file1(void *data, Evas_Object *obj, void *event_info)
{
   appdata_s *ad = data;
   // Load file to Player
   prepare_player(ad, 0);
}

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);
}
```

The btn_load_file1() function loads the first audio file. If you pass 0 to the second parameter of the prepare_player() function, the first audio file will be loaded.

The btn_load_file2() function loads the first audio file. If you pass 1 to the second parameter of the prepare_player() function, the second audio file will be loaded.

Run the example again and tap the File-2 button followed by the Play button. You will hear different audio. Now, tap the File-1 button followed by the Play button. You will hear the audio you heard before.



5) Related APIs

int player_get_state(player_h player, player_state_e *state): an API that returns the current status of the Player. The first parameter is the Player object, while the second parameter returns the status value. The types of player_state_e status are:

- 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): an API that stops the playback.

int player_create(player_h *player): an API that creates a Player object.

int player_set_sound_type(player_h player, sound_type_e type): an API that specifies the sound type. To play the audio file, simply specify SOUND_TYPE_MEDIA.

int player_set_volume(player_h player, float left, float right): an API that specifies the speaker volume. The volume values range from 0 to 1.0. / parameters: Player object, left volume, right volume.

int player_set_looping(player_h player, bool looping): an API that determines whether the audio will loop or not. If you pass True to the second parameter, it is looped. False will play it only once.

int player_set_completed_cb(player_h player, player_completed_cb callback, void *user_data): an API that specifies the callback function upon completion of playback. / parameters: Player object, name of callback function, user data.

int player_unprepare(player_h player): an API that closes the file you loaded to the Player.

int player_set_uri(player_h player, const char * uri): an API that loads a file to the Player.

int player_prepare(player_h player): an API that prepares for playback by the Player. When it is ready to play, it returns 0.

int player_start(player_h player): an API that starts playing the audio.

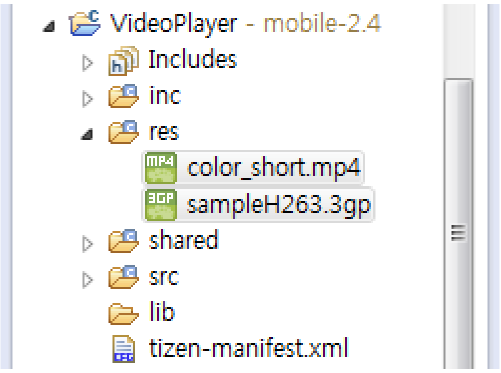
int player_pause(player_h player): an API that pauses the audio playback.

39. Playing Video

Using a player, you can play audio and video. The difference between the two is that playing video requires a screen. The solution is to create an image object to work with the Player. In this section, we will discuss how to play a video file.

1) Creating a Screen

Create a new source project and specify the project name as 'VideoPlayer.' Copy the video file. Copy two files, color_short.mp4 and sampleH263.3gp, under the /Video folder of the appendix to the /res folder of the source project.



```
VideoPlayer - mobile-2.4
  Includes
  inc
  res
    color_short.mp4
    sampleH263.3gp
  shared
  src
  lib
  tizen-manifest.xml
```

Unlike playing audio where you only need a Player and an audio file, to play a video file you need a screen. You can build a screen in the following ways:

- Create EVAS
- Create an image object
- Specify that image object as a display for the Player

First, open the source file (~.c) under the src folder and add libraries and variables at the top of the screen.

```
#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 is a Player structure that plays media such as audio and video. video_rect is an image object where video is displayed.

file_name[] is an array of strings that save the file name.

We are going to create an Image object to use as screen and a Bg widget to use as screen background. This is because the video will only look and feel the way it was intended on a black background. Create a new function on top of the create_base_gui() function. This function adds a widget to a Table.

```
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);
}
```

Then, add new code to the create_base_gui() function. Conformant and Label are annotated.

```
    /* 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_E
XPAND);
    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_EX
PAND);
        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
_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_E
XPAND);
        /* 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_
EXPAND);
            evas_object_size_hint_align_set(tbl, EVAS_HINT_FILL, EVAS_HINT_FILL);
            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);
```

To change the background color to black, we have created a Bg widget. We have created a Table container to place the widget based on the aspect ratio and used the Box to set the width between the widgets.

The video_rect_add() function creates returns as well as an Evas image object. Now, let's start building this function step by step. Add two new functions on top of the create_base_gui() function.

```
// 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;
}
```

The get_resource_path() function returns the full path to all the files saved under the /res folder. We will use this when loading the video file.

The video_rect_add() function creates Evas and Image objects in the screen area.

Build and run the example. The entire screen display is black.

2) Loading a video File

As you run the app, you will create a Player object automatically and load the first video file. Add new code at the end of the create_base_gui() function.

```
    /* 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);
}
```

The create_player() function creates the Player, while the prepare_player() function loads the video file. Now, let's start building it step by step. Add five functions on top of the create_base_gui() function. From here, everything appears very similar to the AudioPlayer example.

```
// 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_state(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);
}
```

The get_player_state() function returns the current status of the Player.

player_get_state(player_h, player_state_e) is an API that returns the current status of the Player. The first parameter is the Player object, and the second parameter returns the status value. The types of player_state_e status are:

- 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() is a callback function invoked upon completion of playback.

player_stop(player_h) is an API that stops the playback.

The create_player() function creates a Player object.

player_create(player_h *) is an API that creates a Player object.

player_set_sound_type(player_h, sound_type_e) is an API that specifies the sound type. To play the video file, simply specify SOUND_TYPE_MEDIA.

player_set_volume(player_h, float, float) is an API that specifies the speaker volume. The volume values range from 0 to 1.0. The second parameter is the left volume, while the third parameter is the right volume.

player_set_looping(player_h, bool) is an API that determines whether the video will loop or not. If you pass True to the second parameter, it is looped. False will play it only once.

player_set_completed_cb(player_h, player_completed_cb, void *) is an API that specifies the callback function upon completion of playback. The second parameter is the name of the callback function, while the third parameter is the user data.

The stop_player() function stops the playback.

The prepare_player() function loads the video file.

player_unprepare(player_h) is an API that closes the file you loaded to the Player.

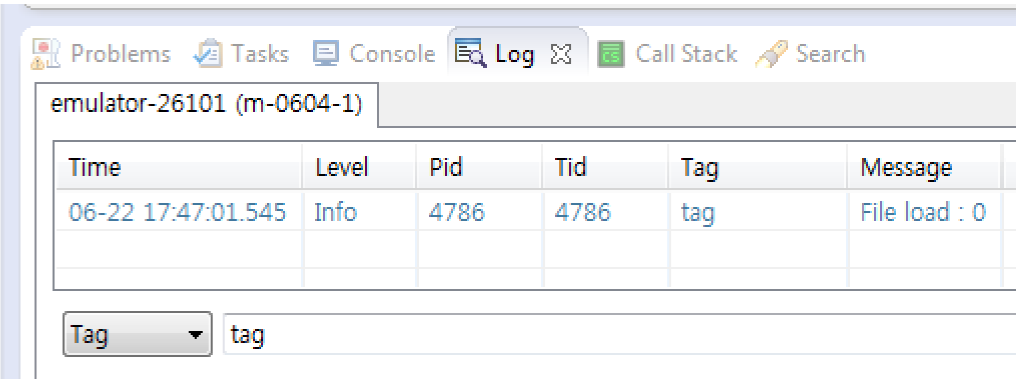
player_set_uri(player_h, const char *) is an API that loads a file to the Player.

The player_set_display(player_h, player_display_type_e, player_display_h) function specifies the screen to the Player. The first parameter specifies the Player object, while the second parameter specifies the screen type. If it is a canvas screen, specify PLAYER_DISPLAY_TYPE_EVAS. The third parameter passes the player_display_h object that corresponds to the screen.

The GET_DISPLAY() function requests player_display_h from the object.

player_prepare(player_h) is an API that prepares for playback by the Player. When it is ready to play, it returns 0.

If you build and start running this way, you will see or hear nothing on the screen. If you check the message on the Log window, ‘File load: 0’ is displayed.



Time	Level	Pid	Tid	Tag	Message
06-22 17:47:01.545	Info	4786	4786	tag	File load : 0

3) Playing Video

We will implement a feature that plays the video file when the Button is tapped. Add new code at the end of the create_base_gui() function.

```
/* 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_EXPAND);
        evas_object_size_hint_align_set(tbl, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(ad->conform, tbl);
        evas_object_show(tbl);
```

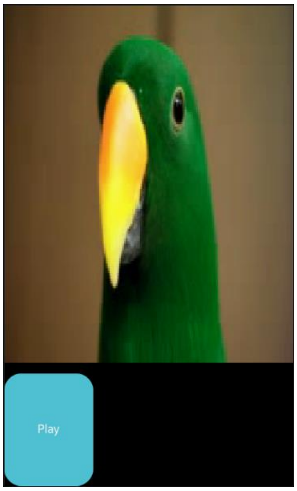
We have added a code that creates a Button. Now, we need to create a callback function for the Button. Create a new function on top of the create_base_gui() function.

```
// 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);
}
```

The start_player() function starts playing the video.

player_start(player_h) is an API that starts playing the video.

Run the example again and tap the Play button. This plays the video.



4) Pausing and Stopping

We will now add two Buttons to implement the Pause and Stop features. Add new code to the create_base_gui() function.

```
/* 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_EXPAND);
    evas_object_size_hint_align_set(tbl, EVAS_HINT_FILL, EVAS_HINT_FILL);
    elm_object_content_set(ad->conform, tbl);
    evas_object_show(tbl);
```

We have added two Buttons. We are now going to create a callback function for the Buttons. Add new code on top of the create_base_gui() function.

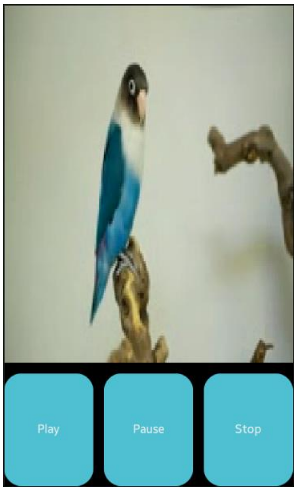
```
// Pause play
static void pause_player(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;

    if( get_player_state(ad->player) == PLAYER_STATE_PLAYING )
        player_pause(ad->player);
}
```

The start_player() function pauses the video playback.

player_start(player_h) is an API that pauses the video playback.

You do not need to create another callback function for the Stop button because you already have the stop_player() function in place.

Run the example again and tap the Play button to play the video. Tap the Pause button to pause or the Play Again button to restart. Tap the Stop button to stop or the Play Again button to start playing from the beginning.



5) Changing the Video File

We will implement a feature that changes the video file by adding two Buttons. Add new code to the create_base_gui() function.

```
/* 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*)ad);
        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*)ad);
        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_EXPAND);
        evas_object_size_hint_align_set(tbl, EVAS_HINT_FILL, EVAS_HINT_FILL);
        elm_object_content_set(ad->conform, tbl);
        evas_object_show(tbl);
```

We have created two Buttons. Finally, we are ready to create a callback function for the Buttons. Add new code on top of the create_base_gui() function.

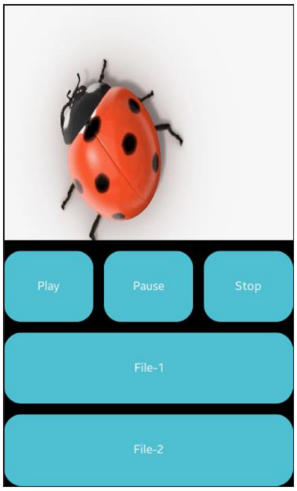
```
static void
btn_load_file1(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    // Load file to Player
    prepare_player(ad, 0);
}

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);
}
```

The btn_load_file1() function loads the first video file. If you pass 0 to the second parameter of the prepare_player() function, the first video file will be loaded.

The btn_load_file2() function loads the first video file. If you pass 1 to the second parameter of the prepare_player() function, the second video file will be loaded.

Run the example again and tap the File-2 button followed by the Play button. You see a different video. Now, tap the File-1 button followed by the Play button. You will see the same video you saw before.



6) Related APIs

int player_get_state(player_h player, player_state_e *state): an API that returns the current status of the Player. The first parameter is the Player object, while the second parameter returns the status value. The types of player_state_e status are:

- 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): an API that stops the playback.

int player_create(player_h *player): an API that creates a Player object.

int player_set_sound_type(player_h player, sound_type_e type): an API that specifies the sound type. To play the video file, simply specify SOUND_TYPE_MEDIA.

int player_set_volume(player_h player, float left, float right): an API that specifies the speaker volume. The volume values range from 0 to 1.0. / parameters: Player object, left volume, right volume.

int player_set_looping(player_h player, bool looping): an API that determines whether the audio will loop or not. If you pass True to the second parameter, it is looped. False will play it only once.

int player_set_completed_cb(player_h player, player_completed_cb callback, void *user_data): an API that specifies the callback function upon completion of playback. / parameters: Player object, name of callback function, user data.

int player_unprepare(player_h player): an API that closes the file you loaded to the Player.

int player_set_uri(player_h player, const char * uri): an API that loads a file to the Player.

int player_prepare(player_h player): an API that prepares for playback by the Player. When it is ready to play, it returns 0.

int player_start(player_h player): an API that starts playing the video.

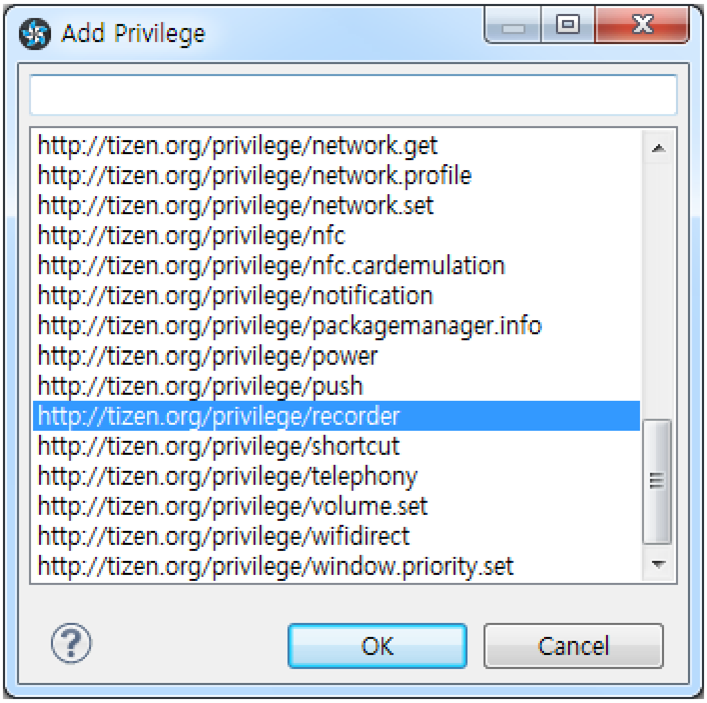
int player_pause(player_h player): an API that pauses the video playback.

40. Recording Audio

You can record sound using the microphone that comes with your smartphone. You can record music at a concert, record your own voice to send to someone instead of a text message, or use it as a memo pad. Let us find out in detail how to do this with an example.

1) Registering a Privilege

Create a new source project and specify the project name as 'RecorderEx.' You need to have the applicable user privilege to be able to use the recorder. After the source project is created, open the tizen-manifest.xml file and click 'Privileges' from the lower tab buttons. Then, click the Add button in the upper right corner. In the popup window, select http://tizen.org/privilege/recorder from the list and click the OK button to close the window.



http://tizen.org/privilege/network.get
http://tizen.org/privilege/network.profile
http://tizen.org/privilege/network.set
http://tizen.org/privilege/nfc
http://tizen.org/privilege/nfc.cardemulation
http://tizen.org/privilege/notification
http://tizen.org/privilege/packagemanager.info
http://tizen.org/privilege/power
http://tizen.org/privilege/push
http://tizen.org/privilege/recorder
http://tizen.org/privilege/shortcut
http://tizen.org/privilege/telephony
http://tizen.org/privilege/volume.set
http://tizen.org/privilege/wifidirect
http://tizen.org/privilege/window.priority.set

After saving, click the tizen-manifest.xml button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.exam
ple.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) Creating a Recorder

To create a Recorder, you must specify the Codec, file type, and quality. Open the source file (~.c) under the src folder and add libraries, variables, and structure at the top of the screen.

```
#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;
```

In this example, we are going to create a total of four Button widgets. You have declared four variables (btn_rec, btn_recstop, btn_play, btn_playstop) to change the Enable status of each Button.

player_h is a player structure that we used in the AudioPlayer and VideoPlayer examples.

recorder_h is a recorder structure.

recorder_audio_codec_e is an Enumeration that saves the types of codec.

codec_list_len is a variable that saves the number of codec lists.

file_path[] is a variable that saves the path to the recorded file.

recorder_file_format_e is an Enumeration that saves information on the file type.

FILE is a file data stream that you use for file input/output.

supported_encoder_data is a structure that saves the list of supported codecs.

We will now create a Recorder object. Add new code at the end of the create_base_gui() function.

```
    /* 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);
}
```

The _recorder_create() function creates a Recorder.

The audio_recorder_get_supported_encoder() function requests a list of supported codecs and returns it.

When you have the codec list in hand, save the first codec to a variable called codec. If the list does not exist, save the PCM codec instead.

The _codec_set() function specifies the codec.

Next, we need to enter the code required to create a Recorder. Add eight new functions on top of the create_base_gui() function.

```
// 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_recording_limit_reached_cb, ad);
        recorder_attr_set_audio_channel(ad->recorder, 1);
        recorder_attr_set_audio_device(ad->recorder, RECORDER_AUDIO_DEVICE_MIC);
        recorder_attr_set_time_limit(ad->recorder, 20);
    }
}
```

```
static bool
_recorder_supported_audio_encoder_cb(recorder_audio_codec_e codec, void *user_data)
{
    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_encoder_data) * (d
ata->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_supported
_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);
}
```

The _recorder_is_recording() function checks the status of recording.

The record_stop() function stops recording.

recorder_commit(recorder_h recorder) is an API that stops recording and saves the data.

recorder_unprepare(recorder_h recorder) is an API that initializes the recorder.

_on_recording_limit_reached_cb() is an event function that is invoked when the maximum recording time is reached. It forcibly stops the recording.

The _recorder_create() function creates a Recorder.

recorder_create_audiorecorder() is an API that creates a Recorder.

recorder_set_recording_limit_reached_cb() is an API that specifies the event function for reaching the maximum time for recording.

recorder_attr_set_audio_channel() is an API that specifies the number of audio channels. Specify 1 in Mono and 2 in Stereo.

recorder_attr_set_audio_device() is an API that specifies the recording device. On your smartphone, it is okay to specify RECORDER_AUDIO_DEVICE_MIC since you are using a microphone.

recorder_attr_set_time_limit() is an API that specifies the maximum time for recording (in seconds).

_recorder_supported_audio_encoder_cb() is a callback function that receives the list of supported audio codecs.

The audio_recorder_get_supported_encoder() function requests a list of supported codecs and returns it.

recorder_foreach_supported_audio_encoder(recorder_h recorder, recorder_supported_audio_encoder_cb callback, void *user_data) is an API that requests a list of supported codecs. It passes the list data to the callback function instead of requesting the list right away.

The get_file_format_by_codec() function returns either the file format or file extension depending on the codec type.

The _codec_set() function specifies the codec.

3) Getting Started with Recording

We will add two Buttons to the screen to implement the Pause and Stop Recording features. Add new code to the create_base_gui() function.

```
/* 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);
```

We have created a Table container in order to place the widget based on the aspect ratio and specified the outside margin using the Frame.

We will implement a feature that starts recording. Add two new functions on top of the create_base_gui() function.

```
// 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);
      }
}
```

The _recorder_apply_settings() function specifies information about the recording.

recorder_set_filename() is an API that specifies the file name of the recording.

recorder_set_file_format() is an API that specifies the recording format.

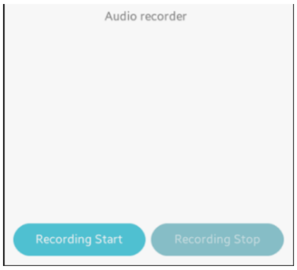
recorder_set_audio_encoder() is an API that specifies the codec.

The record_start() function starts recording.

recorder_prepare() is an API that prepares for recording.

recorder_start() is an API that starts recording.

After building the example, run it and tap the Rec Start button to start recording. Recording ends automatically after 20 seconds. You can also stop manually by tapping the Rec Stop button. Unfortunately, there is no way to hear the recorded audio file as we have not yet implemented the audio playback feature.



4) Playing the Recording

We will now implement a feature that plays the audio file. Add a code that creates two new Buttons to the create_base_gui() function.

```
        /* 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
_EXPAND);
        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
_EXPAND);
        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);
```

The create_player() function creates a Player. Next, add five new functions on top of the create_base_gui() function. You will see this is almost the same as the source code in the AudioPlayer example.

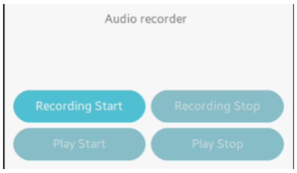
```
// Get player state
static player_state_e get_player_state(player_h player)
{
   player_state_e state;
   player_get_state(player, &state);
   return state;
}

// 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;
}
```

Please see the AudioPlayer example for more details.

Run the example. You need to connect a microphone if you are testing on your desktop with an emulator. On a phone or a laptop, a microphone is not required for testing. Try testing by following the steps below:

- Tap the Record Start button to record voice.
- Tap the Record Stop button to stop recording. It stops automatically after 20 seconds.
- The Play Start button will play the audio you have recorded.
- If you would like to stop playing during playback, tap the Play Stop button.



5) Related APIs

int preference_set_int(const char *key, int value): an API that saves an integer value in the local memory. It passes a key value to the first parameter and data to the second parameter.

int preference_set_boolean(const char *key, bool value): an API that saves a Boolean value in the local memory. It passes a key value to the first parameter and data to the second parameter.

int preference_get_boolean(const char *key, bool *value): an API that requests Boolean-type data from the local memory. As it passes a key value to the first parameter, the result value is returned from the second parameter.

int preference_get_int(const char *key, int *value): an API that requests integer-type data from the local memory. As it passes a key value to the first parameter, the result value is returned from the second parameter.

int recorder_create_audiorecorder(recorder_h *recorder): an API that creates a Recorder.

int recorder_set_recording_status_cb(recorder_h recorder, recorder_recording_status_cb callback, void *user_data): an API specifying an event function that changes the status of recording.

int recorder_set_recording_limit_reached_cb(recorder_h recorder, recorder_recording_limit_reached_cb callback, void *user_data): an API specifying an event function that reaches the maximum time for recording.

int recorder_attr_set_audio_channel(recorder_h recorder, int channel_count): an API that specifies the number of audio channels. Specify 1 in Mono and 2 in Stereo.

int recorder_attr_set_audio_device(recorder_h recorder, recorder_audio_device_e device): an API that specifies the recording device. On your smartphone, you can specify RECORDER_AUDIO_DEVICE_MIC since you are using a microphone.

int recorder_attr_set_time_limit(recorder_h recorder, int second): an API that specifies the maximum time for recording (in seconds).

int recorder_set_filename(recorder_h recorder, const char *path): an API that specifies the file name of the recording.

int recorder_set_file_format(recorder_h recorder, recorder_file_format_e format): an API that specifies the recording format.

int recorder_set_audio_encoder(recorder_h recorder, recorder_audio_codec_e codec): an API that specifies the codec.

int recorder_prepare(recorder_h recorder): an API that prepares for recording.

int recorder_start(recorder_h recorder): an API that starts recording.

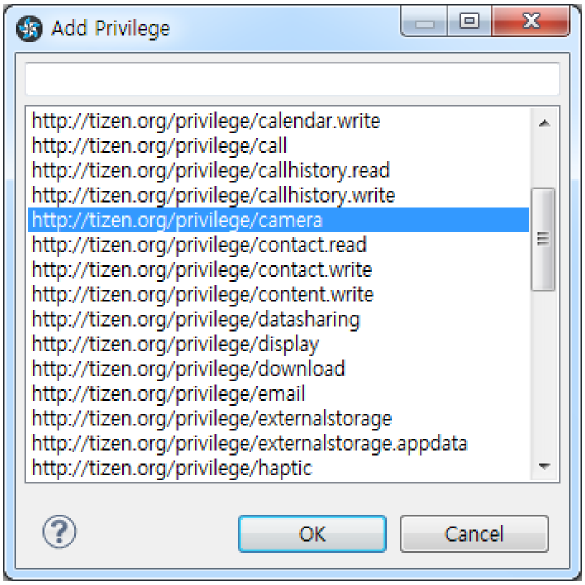
41. Camera Capture

Digital cameras are becoming increasingly obsolete as the performance of smartphones' built-in cameras makes rapid and steady progress. In fact, smartphone cameras have shown remarkable improvement comparable to the level of conventional digital cameras. Consumers nowadays place great emphasis on the camera quality when choosing a smartphone. A built-in camera is used not only for taking photos but also for all sorts of other purposes, including video calls, augmented reality, or as barcode scanners.

In this section, we are going to watch a video in Preview mode using the camera mounted on your phone. We will also learn how to take a picture and save it as a file.

1) Registering a Privilege

Create a new source project and specify the project name as 'CameraEx.' You need to have the applicable user privilege to be able to use the camera. After the source project is created, open the tizen-manifest.xml file and click Privileges from the lower tab buttons. Then, click the Add button in the upper right corner. In the popup window, select http://tizen.org/privilege/camera from the list and click the OK button to close the window.



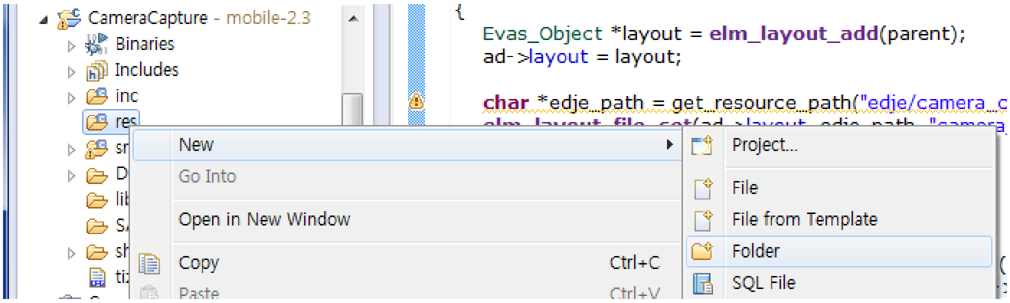
http://tizen.org/privilege/calendar.write
http://tizen.org/privilege/call
http://tizen.org/privilege/callhistory.read
http://tizen.org/privilege/callhistory.write
http://tizen.org/privilege/camera
http://tizen.org/privilege/contact.read
http://tizen.org/privilege/contact.write
http://tizen.org/privilege/content.write
http://tizen.org/privilege/datasharing
http://tizen.org/privilege/display
http://tizen.org/privilege/download
http://tizen.org/privilege/email
http://tizen.org/privilege/externalstorage
http://tizen.org/privilege/externalstorage.appdata
http://tizen.org/privilege/haptic

After saving, click the tizen-manifest.xml button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<?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>
```

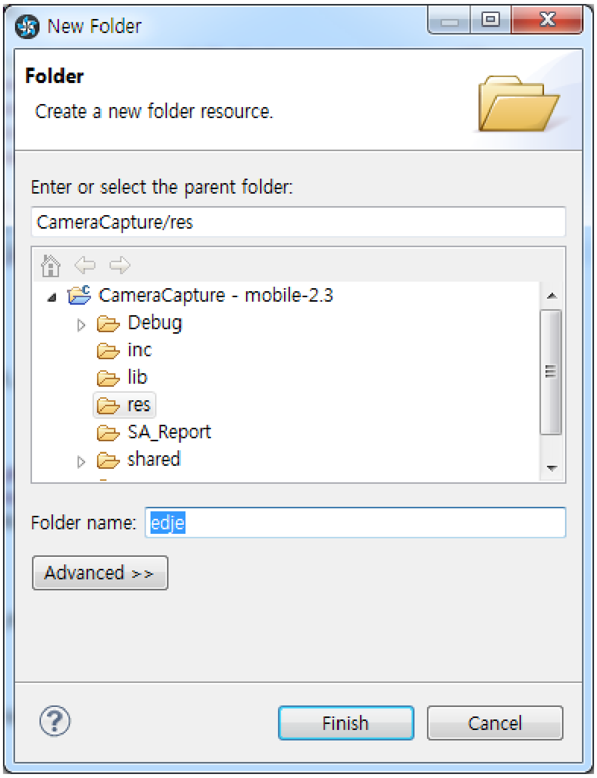
You need a screen to play a preview video. We are now going to copy the EDJE file. Do the following:

- Right-click the /res folder and select [New > Folder] in the shortcut menu.



```
{
    Evas_Object *layout = elm_layout_add(parent);
    ad->layout = layout;

    char *edje_path = get_resource_path("edje/camera_c
```

- When prompted, enter 'edje' in the Folder name field.



New Folder

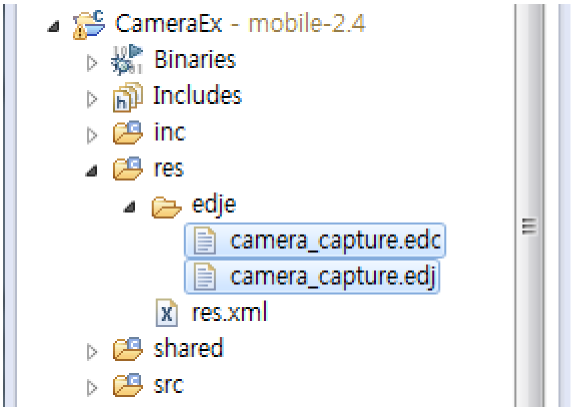
Folder

Create a new folder resource.

Enter or select the parent folder: CameraCapture/res

Folder name: edje

- Copy the camera_capture.edc and camera_capture.edj files located under the /etc/edje folder of the appendix to the new folder.



2) Camera Preview Video

Open the source file (~.c) under the src folder and add libraries and variables at the top of the screen.

```
#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;
```

The preview video resembles the screen we used in the VideoPlayer example. The layout is the preview area, while camera_rect is the Image object that displays the preview video.

We will display the picture we took in the image object.

camera_h is a camera structure.

image_path is a file path to which you save the picture you have taken.

We are now going to create a Camera object and display the preview video on the screen. Create a new function on top of the create_base_gui() function. This function adds a Widget to a Box container.

```
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_EXPAND);
        /* 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);
}
```

Then, add new code to the create_base_gui() function. The Label-creating code will not be used in this example, so annotate it.

```
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_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);*/

    { /* 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_EXPAND);
        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);
}
```

To display the camera preview, you must use the OpenGL library. You can do this by simply passing ‘opengl’ to the elm_config_accel_preference_set() function.

The _main_layout_add() function creates a Layout object that belongs to the preview area.

The _create_camera() function creates a Camera object.

camera_start_preview() is an API that starts a preview.

We will begin creating the functions we mentioned above. Add four new functions on top of the create_base_gui() function.

```
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_path);
    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_N
ONE)
    {
        camera_set_capture_format(ad->camera, CAMERA_PIXEL_FORMAT_JPEG);
        camera_set_display(ad->camera, CAMERA_DISPLAY_TYPE_EVAS, GET_DISPLAY(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;
}
```

The get_resource_path() returns the absolute path to the files located under the /res folder.

The _main_layout_add() function creates a Layout object that belongs to the preview area. The details overlap with the VideoPlayer example.

The _destroy_camera() function deletes a Camera object.

camera_stop_preview(camera_h) is an API that stops the camera preview.

camera_destroy(camera_h) is an API that deletes a Camera object.

The _create_camera() function creates and previews a Camera object.

camera_create(camera_device_e, camera_h *) is an API that creates a Camera object. If CAMERA_DEVICE_CAMERA0 is passed to the first parameter, use the rear camera. If CAMERA_DEVICE_CAMERA1 is passed, use the front camera. The Camera object that has been created is returned to the second parameter.

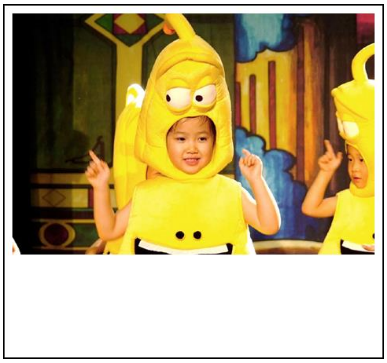
camera_set_capture_format(camera_h, camera_pixel_format_e) is an API that specifies the format of your photo image. Photos are generally large in size, so the JPEG format is recommended.
camera_set_display(camera_h, camera_display_type_e, camera_display_h) is an API that designates a preview screen to the Camera object. While using an Evas-based object, pass CAMERA_DISPLAY_TYPE_EVAS to the second parameter. Pass the camera_display_h object that you requested by passing the Image object to the GET_DISPLAY() function, to the third parameter.

camera_set_display_mode(camera_h, camera_display_mode_e) is an API that specifies the expand/collapse options of the preview video. The option types are as follows:
 - 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) is an API that specifies the rotation angle of the camera.

camera_set_display_flip(camera_h, camera_flip_e) is an API that specifies whether it is vertical or horizontal. The option types are as follows:
 - CAMERA_FLIP_NONE,   /**< No Flip */
 - CAMERA_FLIP_HORIZONTAL, /**< Horizontal flip */
 - CAMERA_FLIP_VERTICAL,   /**< Vertical flip */
 - CAMERA_FLIP_BOTH   /** Horizontal and vertical flip */

Build an example and try running it. It should be tested on a PC with a smartphone or a webcam attached to it. The preview video is displayed at the top of the screen.



3) Fixing the Screen Orientation

The preview shifts as you turn the screen in a horizontal position. This is because the screen orientation has changed from Portrait to Landscape. Let's now fix the screen orientation. Above the create_base_gui() function is a code that specifies the types of screen orientation. Make changes as follows:

```
   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’ means Portrait Primary as opposed to ‘180,’ which means Portrait Secondary. ‘90’ means Landscape Primary, whereas ‘270’ means Landscape Secondary. Currently, only the Portrait Primary mode is supported.

We will have the Camera object automatically deleted when you exit the app. Below the source file, add new code to the app_terminate() function.

```
static void
app_terminate(void *data)
{
    _destroy_camera(data);
}
```

app_terminate() is a callback function that is executed when you exit the app. You can change the callback function in the main() function.

If you run the example again and rotate the screen, the location of the preview no longer shifts.

4) Rotating the Preview with the Screen Orientation

This time, we are going to demonstrate a feature that rotates the camera automatically as the screen rotates its orientation. Modify the code for the main() function and ui_app_orient_changed() below the source file.

```
static void
ui_app_orient_changed(app_event_info_h event_info, void *user_data)
```

```
{
        appdata_s *ad = user_data;
        app_device_orientation_e screen_rot = 0;
        camera_rotation_e camera_rot = CAMERA_ROTATION_270;
        bool horizontal_box = false;
        /* Properly handle rotation */
        app_event_get_device_orientation(event_info, &screen_rot);
        switch (screen_rot)
        {
        case APP_DEVICE_ORIENTATION_0:
                camera_rot = CAMERA_ROTATION_270;
                break;
        case APP_DEVICE_ORIENTATION_90:
                camera_rot = CAMERA_ROTATION_180;
                horizontal_box = true;
                break;
        case APP_DEVICE_ORIENTATION_180:
                camera_rot = CAMERA_ROTATION_90;
                break;
        case APP_DEVICE_ORIENTATION_270:
                camera_rot = CAMERA_ROTATION_NONE;
                horizontal_box = true;
                break;
        }
        /* Set camera preview rotation */
        camera_set_display_rotation(ad->camera, camera_rot);
        /* Make sure to rotate the window itself */
        elm_win_rotation_with_resize_set(ad->win, screen_rot);
        /* Relayout elements in the window by chosing horizontal vs. vertical b
ox */
        elm_box_horizontal_set(ad->box, horizontal_box);
}

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_DEVICE_ORIENTATION_C
HANGED], APP_EVENT_DEVICE_ORIENTATION_CHANGED, ui_app_orient_changed,
&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;
}
```

In the main() function, specify the event function for changing the screen orientation to ui_app_orient_changed(). If this code does not exist, add one.

Then, add new code to the ui_app_orient_changed() function.

app_event_get_device_orientation() is an API that requests screen orientation from the event object. APP_DEVICE_ORIENTATION_0 or APP_DEVICE_ORIENTATION_180 is the Portrait mode; APP_DEVICE_ORIENTATION_90 or APP_DEVICE_ORIENTATION_270 is the Landscape mode.

camera_set_display_rotation() is an API that rotates the camera.

elm_win_rotation_with_resize_set() is an API that resizes the window to align with the screen orientation.

5) Camera Capture

We will now implement a feature that captures a preview video and save it as an image file by tapping a Button. Add a Button-creating code to the create_base_gui() function.

```
    { /* 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);
        }
    }
```

Then, add five new functions on top of the create_base_gui() function.

```
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_capture_co
mpleted_cb, ad);
}
```

The gen_data_path() function returns the path to a file under the /data folder inside the app. You cannot create or modify files in the /res folder. For this reason, the files must be saved to the /data folder or public folder.

app_get_data_path() is an API that returns the absolute path to the /data folder.

The _save_file() function saves the image data to a data file.

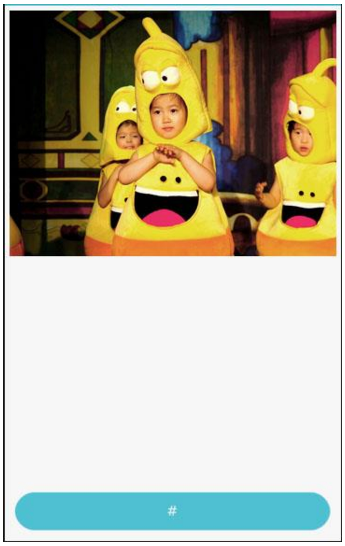
_on_camera_capture_cb() is a callback function that receives the data captured by the camera. Save the data with this function.

_on_camera_capture_completed_cb() is an event callback function for completed camera capture. The preview must be restarted with this function since the preview ends after capturing.

btn_capture_cb() is the callback function of the Button.

camera_start_capture(camera_h, camera_capturing_cb, camera_capture_completed_cb, void *) is an API that starts the camera capture. The first parameter is a Camera object; the second parameter is a data transfer function; the third parameter is an event callback function for completing the capture; and the fourth parameter is user data.

Run the example again and tap the Button. If you hear the camera clicking, it means you have captured successfully. The feature of displaying the saved file is not implemented yet.



5) Displaying the Saved Image File on the Screen

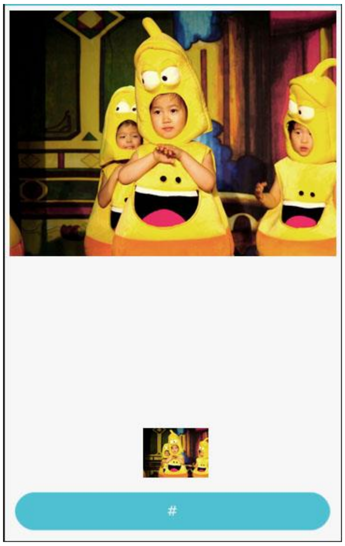
Now, we will add a feature that displays the saved image file in the Image object. Add a code that generates an Image object to the _main_layout_add() function.

```
// 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;
}
```

Load the image file to the Image object once the camera capture is done. Add new code to the _on_camera_capture_completed_cb() function.

```
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);
}
```

Run the example again and tap the Button to display the captured image in the Bg widget.



6) Related APIs

int camera_start_preview(camera_h camera): an API that starts the preview.

int camera_stop_preview(camera_h camera): an API that stops the preview.

int camera_destroy(camera_h camera): an API that deletes a Camera object.

int camera_create(camera_device_e device, camera_h *camera): an API that creates a Camera object. If CAMERA_DEVICE_CAMERA0 is passed to the first parameter, the rear camera is used. If CAMERA_DEVICE_CAMERA1 is passed, the front camera is used. The Camera object that has been created is returned to the second parameter.

int camera_set_capture_format(camera_h camera, camera_pixel_format_e format): an API that specifies the format of your photo image. CAMERA_PIXEL_FORMAT_JPEG will create an image in JPEG.

Int camera_set_display(camera_h camera, camera_display_type_e type, camera_display_h display): an API that designates a preview screen to the Camera object. While using an Evas-based object, pass CAMERA_DISPLAY_TYPE_EVAS to the second parameter. Pass the camera_display_h object that you requested by passing the Image object to the GET_DISPLAY() function, to the third parameter.

Int camera_set_display_mode(camera_h camera, camera_display_mode_e mode): an API that specifies the expand/collapse options of the preview video. The option types are as follows:

```
 - 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): an API that specifies the rotation angle of the camera.

int camera_set_display_flip(camera_h camera, camera_flip_e flip): an API that specifies whether it is vertical or horizontal. The option types are as follows:

- 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): an API that returns the absolute path to the /data folder.

int camera_start_capture(camera_h camera, camera_capturing_cb capturing_cb, camera_capture_completed_cb completed_cb, void *user_data): an API that starts camera capture. / parameters: Camera object, data transfer function, event callback function for completing the capture, and user data.

42. System Information

During app development, system information is often displayed on the Help screen for you. If you are developing an app that utilizes a camera, you need to check whether you have a rear camera or a front camera. To make it compatible with devices with different resolutions, you also have to request the number of monitor pixels. In this example, we will learn how to request the system information.

1) Whether or Not There is a Rear Camera

Create a new source project and specify the project name as 'SystemInfo.' After the source project is created, open the source file (~.c) under the src folder and add a library header file as well as variables at the top.

```
#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;
```

We have declared a total of six Label widget variables. We will now display whether or not the rear camera exists in the first Label, whether or not the front camera exists in the second Label, and whether or not you can make a call in the third Label. The number of horizontal pixels will be in the fourth Label, the number of vertical pixels in the fifth Label, and the platform version in the sixth Label. Create three new functions on top of the create_base_gui() function.

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int col, int row, int spanx, int s
pany,
        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_exp
and ? 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_EXPAN
D);
        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;
}
```

The my_table_pack() function adds a widget to the Table container.

The my_button_add() function creates a Button widget.

The my_label_add() function creates a Label widget.

Add new code within the create_base_gui() function. This code creates one Frame, one Table, and one Button, as well as twelve Label widgets.

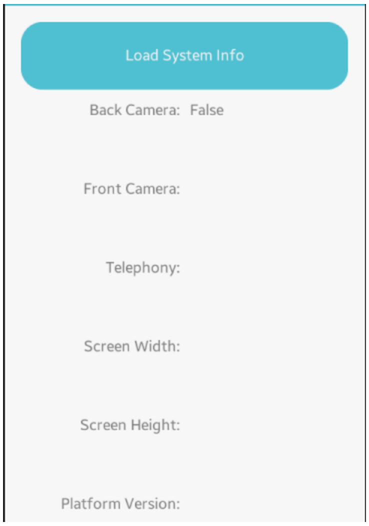
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win,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

We are now going to create callback functions for the Buttons. Add new code on top of the create_base_gui() function.

```
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", &bVal
ue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        elm_object_text_set(ad->label1, bValue ? "True" : "False");
    }
}
```

system_info_get_platform_bool(char *, bool *) is an API that requests the system information. It returns data in Boolean format. The first parameter is a key value. Passing "http://tizen.org/feature/camera.back" returns whether or not there is a rear camera.

Build and run the example. Tapping the Button will change the text in the first Label. Running it from an emulator displays False; running it from a user device displays True.



Back Camera: False

Front Camera:

Telephony:

Screen Width:

Screen Height:

Platform Version:

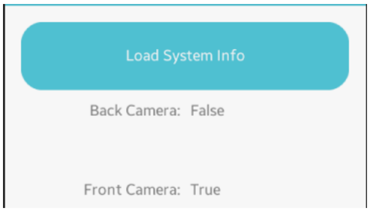
2) Whether or Not There is a Front Camera

We will now discover if we have a front camera and display it on the screen. Add new code at the end of the btn_clicked_cb() function.

```
    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");
    }
}
```

Passing “http://tizen.org/feature/camera.front” to the first parameter returns whether or not there is a front camera.

Build and run the example. Tap the Button; you will see True displayed in the second Label.



Back Camera: False

Front Camera: True

3) Are There Telephony Features?

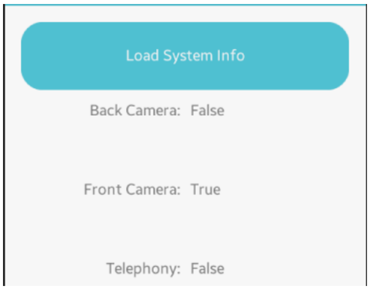
We will now discover if there are telephony features. Add new code at the end of the btn_clicked_cb() function.

```
    ret = system_info_get_platform_bool("http://tizen.org/feature/camera.front", &bVal
ue);
    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.telepho
ny", &bValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        elm_object_text_set(ad->label3, bValue ? "True" : "False");
    }
}
```

Passing “http://tizen.org/feature/network.telephony” to the first parameter of the system_info_get_platform_bool() function returns whether or not there are telephony features. A True value does not necessarily mean that you can use the telephony or network. It just means that you have a hardware communication tool in place. You will not be able to use the communication feature if you do not have a USIM chip or you have disabled the network feature under Preferences.

Build and run the example. Tap the Button; you will see True displayed in the third Label.

```
    {
```



Back Camera: False

Front Camera: True

Telephony: False

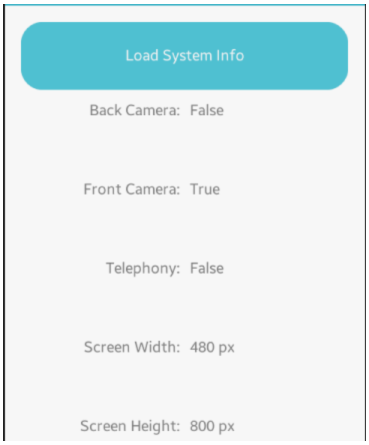
4) Number of Monitor Pixels

Let’s request the number of monitor pixels. Add new code at the end of the btn_clicked_cb() function.

```
    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 *) is an API that requests the system information. It returns data in integer format. The first parameter is a key value. Passing “http://tizen.org/feature/screen.width” returns the number of horizontal monitor pixels. Passing “tizen.org/feature/screen.height” returns the number of vertical monitor pixels.

Build and run the example. Tap the Button; you will see numbers displayed in the fourth Label and fifth Label.



Back Camera: False

Front Camera: True

Telephony: False

Screen Width: 480 px

Screen Height: 800 px

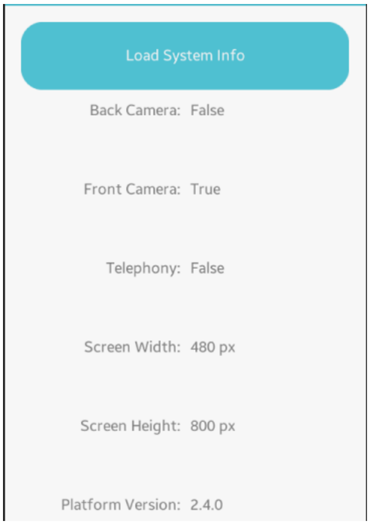
5) Platform Version

Let's request the platform version. Add new code at the end of the btn_clicked_cb() function.

```
    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.versio
n", &sValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        elm_object_text_set(ad->label6, sValue);
        free(sValue);
    }
}
```

system_info_get_platform_string(char *, char **) is an API that requests the system information. It returns data in string format. The first parameter is a key value. Passing “http://tizen.org/feature/platform.version” returns the platform version.

Build and run the example. Tap the Button; you will see the platform version displayed in the sixth Label.



Load System Info

Back Camera: False

Front Camera: True

Telephony: False

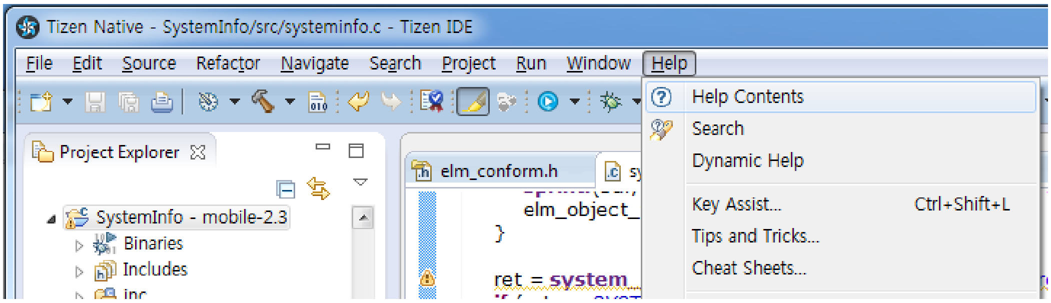
Screen Width: 480 px

Screen Height: 800 px

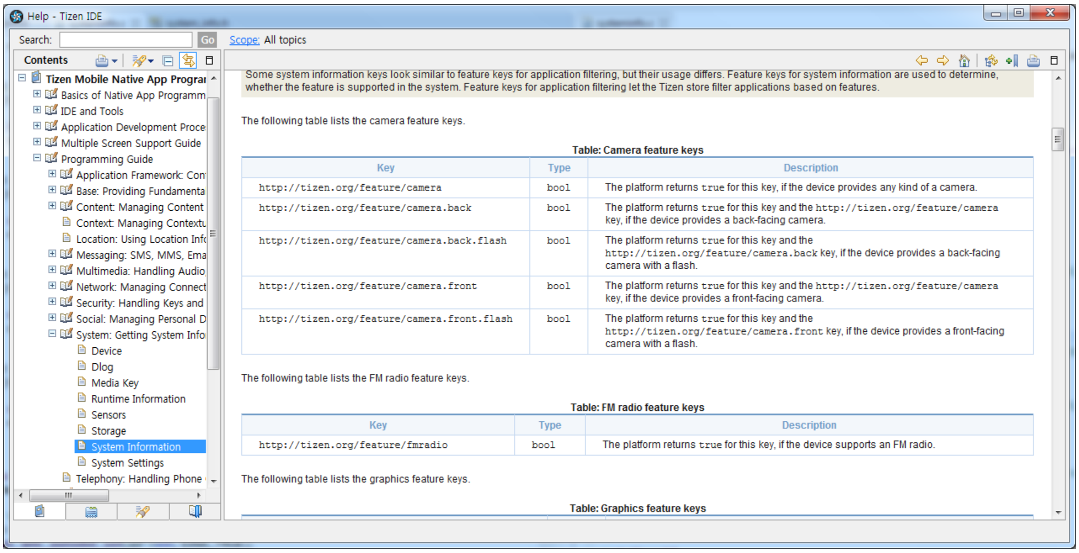
Platform Version: 2.4.0

6) Related APIs

To view the types of system information, key values, and return formats from the Help Contents list. Select [Help > Help Contents] from the main menu.



With the Help Contents running, choose [Tizen Mobile Native App Programming > Programming Guide > System > System Information] from the tree list on the left side. You will see the keys, return types, and description on the right side of the screen.


Help - Tizen IDE
Search:
Go
Scope: All topics
Contents
Tizen Mobile Native App Prograi
Basics of Native App Programm
IDE and Tools
Application Development Proce
Multiple Screen Support Guide
Programming Guide
Application Framework: Con
Base: Providing Fundamenta
Content: Managing Content
Context: Managing Contextu
Location: Using Location Inf
Messaging: SMS, MMS, Ema
Multimedia: Handling Audio
Network: Managing Connect
Security: Handling Keys and
Social: Managing Personal D
System: Getting System Info
Device
Dlog
Media Key
Runtime Information
Sensors
Storage
System Information
System Settings
Telephony: Handling Phone

Some system information keys look similar to feature keys for application filtering, but their usage differs. Feature keys for system information are used to determine, whether the feature is supported in the system. Feature keys for application filtering let the Tizen store filter applications based on features.

The following table lists the camera feature keys.

Table: Camera feature keys

Key	Type	Description
`http://tizen.org/feature/camera`	bool	The platform returns `true` for this key, if the device provides any kind of a camera.
`http://tizen.org/feature/camera.back`	bool	The platform returns `true` for this key and the `http://tizen.org/feature/camera` key, if the device provides a back-facing camera.
`http://tizen.org/feature/camera.back.flash`	bool	The platform returns `true` for this key and the `http://tizen.org/feature/camera.back` key, if the device provides a back-facing camera with a flash.
`http://tizen.org/feature/camera.front`	bool	The platform returns `true` for this key and the `http://tizen.org/feature/camera` key, if the device provides a front-facing camera.
`http://tizen.org/feature/camera.front.flash`	bool	The platform returns `true` for this key and the `http://tizen.org/feature/camera.front` key, if the device provides a front-facing camera with a flash.

The following table lists the FM radio feature keys.

Table: FM radio feature keys

Key	Type	Description
`http://tizen.org/feature/fmradio`	bool	The platform returns `true` for this key, if the device supports an FM radio.

The following table lists the graphics feature keys.

Table: Graphics feature keys

int system_info_get_platform_bool(const char *key, bool *value): an API that requests the system information. It returns data in Boolean format.

int system_info_get_platform_int(const char *key, int *value): an API that requests the system information. It returns data in integer format.

int system_info_get_platform_string(const char *key, char **value): an API that requests the system information. It returns data in string format.

43. System Preferences

Multilingual support requires you to check the language settings of the user. If you are in silent mode when a new message arrives, you would need a vibrating alert. In this example, we will learn how to request information about the system preferences.

1) Requesting Language Settings

Create a new source project and specify the project name as 'SystemSetting.' After the source project is created, open the source file (~.c) under the src folder and add variables at the top of the source file.

```
#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;
```

We have declared a total of four Label widget variables. We will mark language settings for the first Label as well as whether silent mode is enabled or not for the second Label, including the time zone for the third Label, and the device name for the fourth Label.

Create three new functions on top of the create_base_gui() function.

```
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_expand ? 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_EXPAND);
          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;
}
```

The my_table_pack() function adds a widget to the Table container.

The my_button_add() function creates a Button widget.

The my_label_add() function creates a Label widget.

Add new code within the create_base_gui() function. This code creates one Frame, one Table, and one Button widget, as well as eight Label widgets.

```
    /* 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);
```

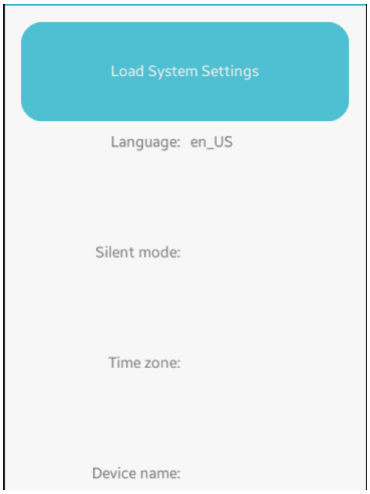
```
            o = my_label_add(tbl, "Silent mode:");
            my_table_pack(tbl, o, 0, 2, 1, 1, EVAS_HINT_EXPAND, 0, 1.0, EVAS_HI
NT_FILL);

            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);
```

We are now going to create callback functions for the Buttons. Add new code on top of the create_base_gui() function.

```
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, &sVa
lue);
    elm_object_text_set(ad->label1, sValue);
    free(sValue);
}
```

system_settings_get_value_string(system_settings_key_e, char **) is an API that requests the system preferences. It returns data in string format. The first parameter is a key value. Passing SYSTEM_SETTINGS_KEY_LOCALE_LANGUAGE returns the language type specified by the user.

Build and run the example. Tap the Button; you will see the language type displayed in the first Label.


Load System Settings

Language: en_US

Silent mode:

Time zone:

Device name:

2) Requesting the Silent Mode

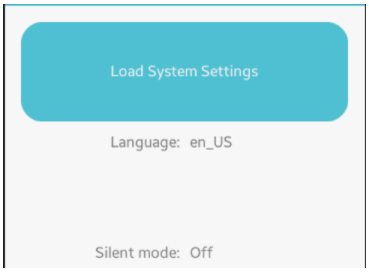
We will mark whether silent mode is enabled or not on the screen. Add new code at the end of the btn_clicked_cb() function.

```
    system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_LANGUAGE, &sValue);
    elm_object_text_set(ad->label1, sValue);
    free(sValue);

    system_settings_get_value_bool(SYSTEM_SETTINGS_KEY_SOUND_SILENT_MODE, &bValue);
    elm_object_text_set(ad->label2, bValue ? "On" : "Off");
}
```

system_settings_get_value_bool(system_settings_key_e, bool *) is an API that requests the system preferences. It returns data in Boolean format.

The first parameter is a key value. Passing SYSTEM_SETTINGS_KEY_SOUND_SILENT_MODE returns whether or not silent mode is enabled.

Build and run the example. Tap the Button; you will see whether silent mode is enabled or not in the second Label.



Language: en_US

Silent mode: Off

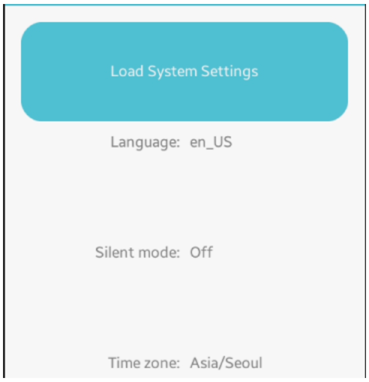
3) Requesting the Time Zone

We will request the time zone this time. Add new code at the end of the btn_clicked_cb() function.

```
    system_settings_get_value_bool(SYSTEM_SETTINGS_KEY_SOUND_SILENT_MODE, &b
Value);
    elm_object_text_set(ad->label2, bValue ? "On" : "Off");

    system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE,
&sValue);
    elm_object_text_set(ad->label3, sValue);
    free(sValue);
}
```

Passing SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE to the first parameter of the system_settings_get_value_string() function returns the time zone in a string.

Build and run the example. Tap the Button; you will see the time zone set by the user in the third Label.



Language: en_US

Silent mode: Off

Time zone: Asia/Seoul

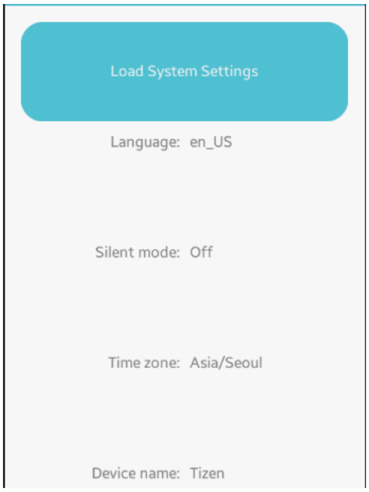
4) Requesting the Device Name

Now, we will request the device name. Add new code at the end of the btn_clicked_cb() function.

```
    system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_TIMEZONE, &sVal
ue);
    elm_object_text_set(ad->label3, sValue);
    free(sValue);

    system_settings_get_value_string(SYSTEM_SETTINGS_KEY_DEVICE_NAME, &sVal
ue);
    elm_object_text_set(ad->label4, sValue);
    free(sValue);
}
```

When you pass SYSTEM_SETTINGS_KEY_DEVICE_NAME to the first parameter of the system_settings_get_value_string() function, it returns the device name in a string.

Build and run the example. Tap the Button; you will see the device name displayed in the fourth Label.



Language: en_US

Silent mode: Off

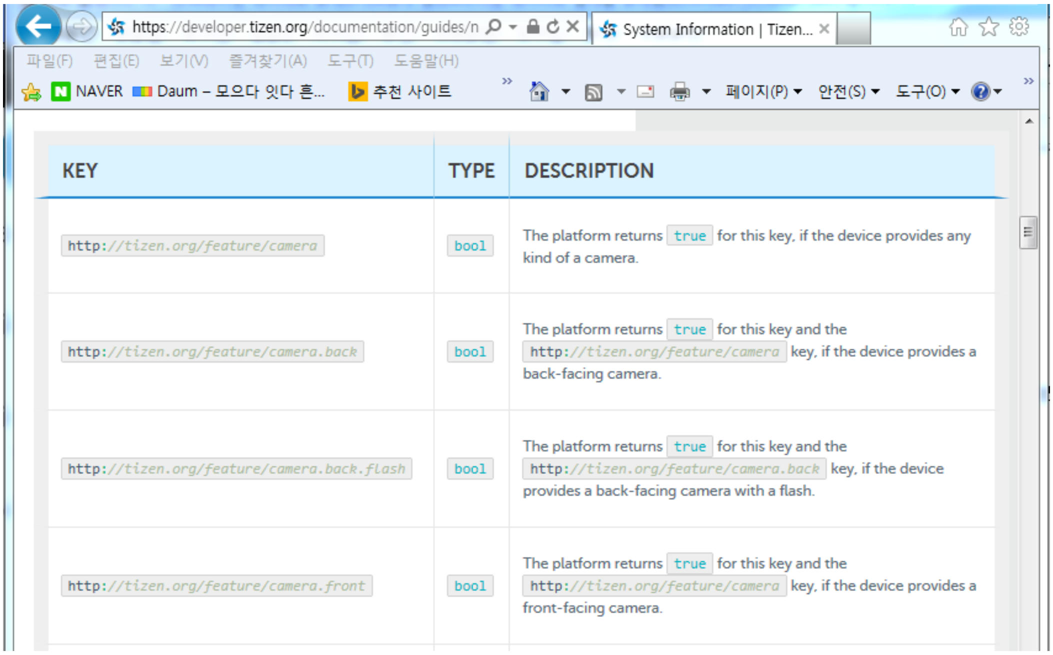
Time zone: Asia/Seoul

Device name: Tizen

5) Related APIs

Let’s take a look at the types of system preferences, key values, and return formats from the list. To see a list of key values for the System Information, open a web browser and go to the address below:

https://developer.tizen.org/documentation/guides/native-application/system/system-information


https://developer.tizen.org/documentation/guides/n
System Information | Tizen...
파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)
NAVER
Daum – 모으다 잇다 흔...
추천 사이트
페이지(P)
안전(S)
도구(O)

KEY	TYPE	DESCRIPTION
http://tizen.org/feature/camera	bool	The platform returns true for this key, if the device provides any kind of a camera.
http://tizen.org/feature/camera.back	bool	The platform returns true for this key and the http://tizen.org/feature/camera key, if the device provides a back-facing camera.
http://tizen.org/feature/camera.back.flash	bool	The platform returns true for this key and the http://tizen.org/feature/camera.back key, if the device provides a back-facing camera with a flash.
http://tizen.org/feature/camera.front	bool	The platform returns true for this key and the http://tizen.org/feature/camera key, if the device provides a front-facing camera.

int system_settings_get_value_int(system_settings_key_e key, int *value): an API that requests the system preferences. It returns data in integer format.

int system_settings_get_value_bool(system_settings_key_e key, bool *value): an API that requests the system preferences. It returns data in Boolean format.

int system_settings_get_value_string(system_settings_key_e key, char **value): an API that requests the system preferences. It returns data in string format.

44. Battery Status

Video playback must stop if the device’s battery level drops to 15% or lower. If not, you could miss an important call. While the device is charging, it is safe for playback to continue even if the battery level is low. We will now learn how to check the current battery status and request battery-related events.

1) Requesting the Battery Status

Create a new source project and specify the project name as ‘BatteryInfo.’ After the source project is created, open the source file (~.c) under the src folder and add a library at the top of the screen.

```
#include  "batteryinfo.h"
#include  <device/battery.h>
#include  <device/callback.h>
```

device/battery.h is a library header file containing information about the battery.

device/callback.h is a library header file for event callbacks related to devices.

We will implement a feature that displays the battery level as well as whether it is being charged or not on the screen when the Button is tapped. Create a new function on top of the create_base_gui() function. This function adds a Widget to a Box container.

```
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_EXPAND);
        /* 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);
}
```

Then, it adds the Box- and Button-creating code at the end of the create_base_gui() function.

```
    /* 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_EX
PAND);
        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);
```

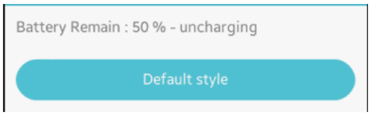
Add a Button callback function on top of the create_base_gui() function.

```
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 ? "charging" : "un
charging");

   elm_object_text_set(ad->label, buf);
}
```

device_battery_get_percent(int *) is an API that returns the battery level converted into percentage.

device_battery_is_charging(bool *) is an API that returns the charging state. It returns true while charging; otherwise, it returns false.

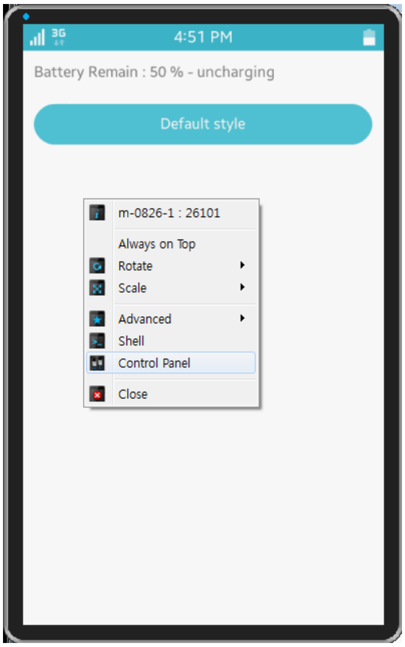
Build and run the example. Tap the Button; you will see the battery level and charging state in the Label widget. If you have run it from the emulator, it will probably display '50% remaining, uncharging.'



Battery Remain : 50 % - uncharging

2) Changing the Battery Status in the Emulator

Use the Control Panel to change the battery status in the emulator. Right-click the emulator and choose [Control Panel] from the shortcut menu.

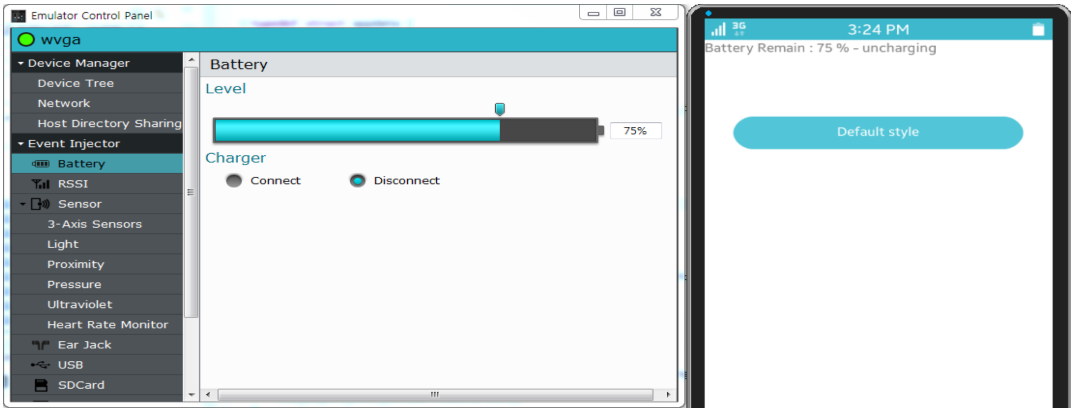


Battery Remain : 50 % - uncharging

In the new popup window, choose [Event Injector > Battery] from the tree list on the left side.

When you see a slider that looks like a battery, drag the bar to change the value, and then select the Connect Radio button under the Charger.

Then, tap the 'Default style' button in the emulator to show brand new information.



Level: 75%

Charger: ○ Connect ● Disconnect

Battery Remain : 75 % - uncharging

3) Requesting an Event for Changing the Battery Status

Let's implement a feature that automatically requests an event when you have connected the charging cable. Add new code at the end of the create_base_gui() function.

```
   evas_object_show(ad->win);

   device_add_callback(DEVICE_CALLBACK_BATTERY_CHARGING, battery_charging_cb, ad);
}
```

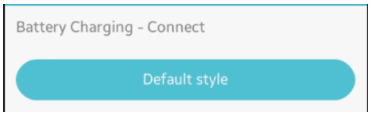
device_add_callback(device_callback_e, device_changed_cb, void *) is an API that specifies an event callback function for the device. You can request an event for connecting the charger by passing DEVICE_CALLBACK_BATTERY_CHARGING to the first parameter. Passing DEVICE_CALLBACK_BATTERY_LEVEL requests an event for changing the battery level.

Now, create a callback function on top of the create_base_gui() function.

```
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" : "Disconnect");
    elm_object_text_set(ad->label, buf);
}
```

If you have passed 1 to the second parameter of the charger event, it is a connection event; 0 represents a disconnection event.

Run the example again and tap the radio buttons several times in turn under the Charger in the Control Panel. A new message is displayed in the Label widget.



Battery Charging - Connect

4) Requesting the Event for Low Battery Levels

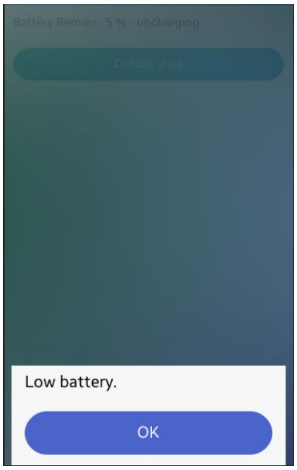
The device should switch to sleep mode once the battery level has dropped below 15%. There is a function named ui_app_low_battery() below the source file. This is an event function that alerts you of low battery level. Add new code to the ui_app_low_battery() function.

If this function does not exist, or if you would like to change the name, you may do so in the main() function.

```
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;
}
```

When the battery reaches a critical level, it automatically displays the current battery status on the screen.

Run the example again and check the Disconnect box in the Control Panel, and then change the battery level to less than 15%. A warning popup appears, and the message in the Label widget is changed.



Battery Remain : 5 % - uncharging

Low battery.

5) Related APIs

device/battery.h: a library header file that contains information about the battery.

device/callback.h: a library header file for event callbacks related to devices.

device_battery_get_percent(int *): an API that returns the battery level converted into percentage.

int device_battery_get_percent(int *percent): an API that returns the charging state. It returns true while charging; otherwise, it returns false.

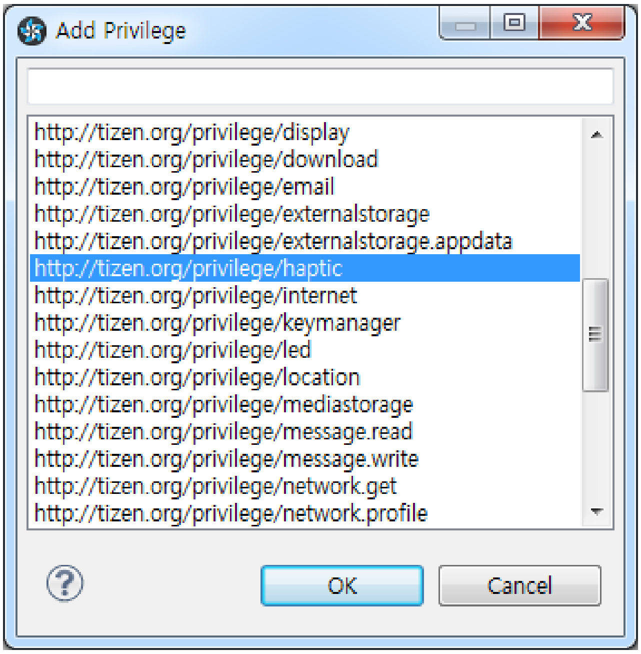
int device_add_callback(device_callback_e type, device_changed_cb callback, void *user_data): an API that specifies an event callback function for the device. You can request an event for connecting the charger by passing DEVICE_CALLBACK_BATTERY_CHARGING to the first parameter. Passing DEVICE_CALLBACK_BATTERY_LEVEL requests an event for changing the battery level.

45. Generating Vibration

If you receive a message in silent mode, it is helpful to be notified with a vibration. Vibration effects will help users stay immersed in the game but at the same time be aware they have new messages. You have yourself a 4D experience if you combine it with 3D graphics. In this example, we will learn how to generate vibration.

1) Registering a Privilege

Create a new source project and specify the project name as 'VibrateEx.' You need to have the applicable user privilege to be able to use the vibration feature. After the source project is created, open the tizen-manifest.xml file and click Privileges from the lower tab buttons. Then, click the Add button in the upper right corner. In the popup window, select http://tizen.org/privilege/haptic from the list and click the OK button to close the window.



http://tizen.org/privilege/display
http://tizen.org/privilege/download
http://tizen.org/privilege/email
http://tizen.org/privilege/externalstorage
http://tizen.org/privilege/externalstorage.appdata
http://tizen.org/privilege/haptic
http://tizen.org/privilege/internet
http://tizen.org/privilege/keymanager
http://tizen.org/privilege/led
http://tizen.org/privilege/location
http://tizen.org/privilege/mediastorage
http://tizen.org/privilege/message.read
http://tizen.org/privilege/message.write
http://tizen.org/privilege/network.get
http://tizen.org/privilege/network.profile

After saving, click the tizen-manifest.xml button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.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) Requesting the Number of Haptic Devices

Haptic is a touch-driven interface. We will request the count to find out how many Haptic devices are mounted to your phone. Open the source file (~.c) under the src folder and add a library header file as well as variables at the top.

```
#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 is a library for controlling the Haptic device.

haptic_device_h is a handle capable of controlling the Haptic device.

haptic_effect_h is a handle capable of controlling one of the Haptic effects.

Add new code at the end of the create_base_gui() function.

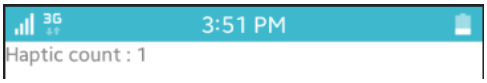
```
    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    haptic_count(ad);
}
```

The haptic_count() function displays the number of Haptic devices. Create this function on top of the create_base_gui() function.

```
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) is an API that returns the number of Haptic devices. If less than 1, the vibration feature will not be supported.

Build an example and install it on your smartphone. You cannot test the vibration feature in the emulator. Once the example is executed, the number of Haptic devices is displayed in the Label widget.



Haptic count : 1

2) Generating a Vibration Feature

Create a Haptic object to generate vibration. Create two new functions on top of the create_base_gui() function.

```
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_EXPAN
D);
        /* 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_dat
a)
{
    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;
}
```

The my_box_pack() function adds a widget to the Box container.

The my_button_add() function creates a Button widget.

Add new code to the create_base_gui() function.

```
    /* 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_FILL, 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);
}
```

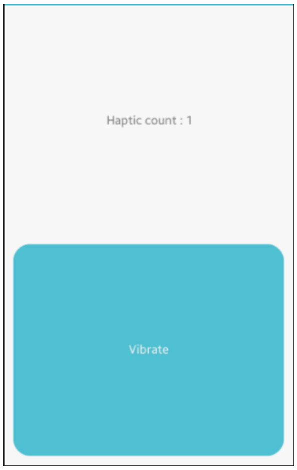
device_haptic_open(int, haptic_device_h *) is an API that creates a haptic object. Passing the Haptic device number to the first parameter makes the second parameter return a Haptic object in response.

We are going to implement a feature that generates vibration for 5 seconds when you tap the button. Create a Button callback function on top of the create_base_gui() function.

```
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_handle);
}
```

device_haptic_vibrate(haptic_device_h, int, int, haptic_effect_h *) is an API that generates vibration. The first parameter returns a Haptic object; the second parameter returns the duration (in milliseconds); the third parameter returns the intensity (0–100); and the fourth parameter returns a Haptic effect-controlling handle. It is used to stop the vibration forcibly.

Run the example again and tap the Button. The vibration lasts for 5 seconds.



3) Stopping the Vibration Feature

We will now implement a feature that forces the vibration to stop by adding a second Button. Add a Button-creating code to the create_base_gui() function.

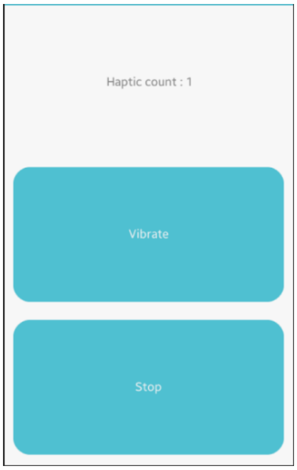
```
        /* Buttons */
        btn = my_button_add(box, "Vibrate", btn_vibrate_cb, ad);
        my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVAS_HINT_FILL, EVAS_HINT_FILL);
        btn = my_button_add(box, "Stop", btn_stop_cb, ad);
        my_box_pack(box, btn, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND, EVAS_HINT_FILL, EVAS_HINT_FILL);
    }
```

Then, create a Button callback function on top of the create_base_gui() function.

```
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) is an API that stops the Haptic features. It passes a Haptic object to the first parameter and an effect-controlling handle to the second.

Run the example again and tap the first Button. Once the vibration starts, tap the second Button. This will stop the vibration.



4) Dynamic Vibrate

We will implement a feature that switches the vibration on/off using a timer. Add the third Button-creating code to the create_base_gui() function.

```
        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_FILL, EVAS_HINT_FILL);
    }
```

Finally, add three new functions on top of the create_base_gui() function.

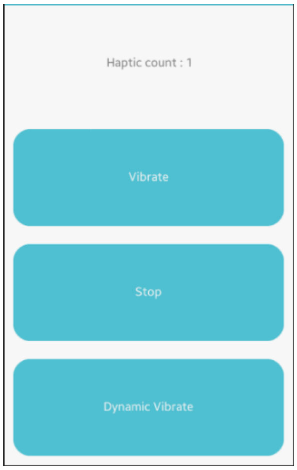
```
static void
dynamic_vibrate(appdata_s *ad)
{
    if( ad->effect_handle != NULL )
        device_haptic_stop(ad->handle, &ad->effect_handle);

    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);
}
```

The dynamic_vibrate() function is invoked every 0.5 seconds to switch the vibration on/off.

timer1_cb() is a timer event function invoked every 0.5 seconds. Tapping the third Button will invoke this function six times.

The btn_dynamic_cb() function resets the global variable and starts the timer.

Run the example again and tap the third Button. Vibration switches on/off every 0.5 seconds.



Haptic count : 1

5) Related APIs

int device_haptic_get_count(int *device_number): an API that returns the number of Haptic devices. If less than 1, the vibration feature will not be supported.

int device_haptic_open(int device_index, haptic_device_h *device_handle): an API that creates a haptic object. Passing the Haptic device number to the first parameter makes the second parameter return a Haptic object in response.

int device_haptic_vibrate(haptic_device_h device_handle, int duration, int feedback, haptic_effect_h *effect_handle): an API that generates a vibration. The first parameter returns a Haptic object; the second parameter returns the duration (in milliseconds); the third parameter returns the intensity (0–100); and the fourth parameter returns a Haptic effect-controlling handle. This is used to stop the vibration forcibly.

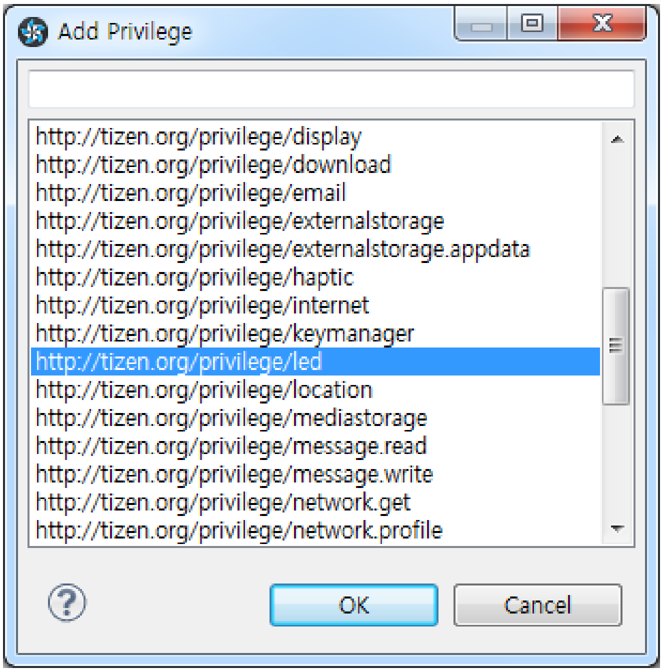
int device_haptic_stop(haptic_device_h device_handle, haptic_effect_h effect_handle): an API that stops the Haptic features. It passes a Haptic object to the first parameter and an effect-controlling handle to the second parameter.

46. LED Flash Backlight

In this example, we will learn how to turn on/off the LED backlight that is used for the camera flash.

1) Registering a Privilege

Create a new source project and specify the project name as ‘LedFreshEx.’ You need to have the applicable user privilege to be able to control the backlight. After the source project is created, open the tizen-manifest.xml file and click Privileges from the lower tab buttons. Then, click the Add button in the upper right corner. In the popup window, select http://tizen.org/privilege/led from the list and click the OK button to close the window.



http://tizen.org/privilege/display
http://tizen.org/privilege/download
http://tizen.org/privilege/email
http://tizen.org/privilege/externalstorage
http://tizen.org/privilege/externalstorage.appdata
http://tizen.org/privilege/haptic
http://tizen.org/privilege/internet
http://tizen.org/privilege/keymanager
http://tizen.org/privilege/led
http://tizen.org/privilege/location
http://tizen.org/privilege/mediastorage
http://tizen.org/privilege/message.read
http://tizen.org/privilege/message.write
http://tizen.org/privilege/network.get
http://tizen.org/privilege/network.profile

After saving, click the tizen-manifest.xml button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<?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" multiple="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) Requesting the Maximum Brightness of LED

The brightness should be specified when the LED is on. Let's request the maximum brightness. Open the source file (~.c) under the src folder and add a library header file as well as variables at the top.

```
#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 is a LED-controlling library.

max is a variable that saves the maximum brightness.

Add new code at the end of the create_base_gui() function.

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

    get_max_brightness(ad);
}
```

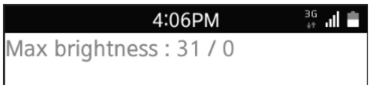
The get_max_brightness function requests the maximum brightness of the LED backlight. Create this function on top of the create_base_gui() function.

```
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 *) is an API that requests the maximum brightness of the LED backlight.

device_flash_get_brightness(int *) is an API that requests the current brightness of the LED backlight.

Run the example. Both maximum brightness and current brightness are displayed in the Label widget.



Max brightness : 31 / 0

3) LED On/Off

We will implement a feature that turns on/off the LED backlight by adding two Buttons. Create two new functions on top of the create_base_gui() function.

```
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_EXPAND);
         /* 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;
}
```

Add a widget-creating code to the create_base_gui() function.

```
    /* 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);

    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_FILL, 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_FILL, EVAS_HINT_FILL);
    }

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

Then, create a Button callback function on top of the create_base_gui() function.

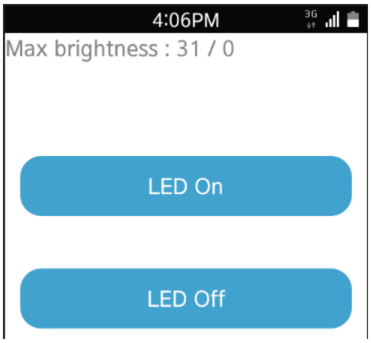
```
static void
btn_led_on_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    device_flash_set_brightness(ad->max);
    device_led_play_custom(1000, 500, 0xFFFFFF00, LED_CUSTOM_DEFAULT);
}

static void
btn_led_off_cb(void *data, Evas_Object *obj, void *event_info)
{
    device_led_stop_custom();
}
```

device_flash_set_brightness(int) is an API that specifies the brightness of the LED.

device_led_play_custom(on, off, color, int) is an API that starts LED On.

device_led_stop_custom(void) is an LED Off API.

Run the example again. The LED On button turns on the LED; the LED Off button turns it off.



4:06PM

Max brightness : 31 / 0

4) Troubleshooting LED Off

Depending on the model, LED may not go off with the device_led_stop_custom() function. In this case, simply decrease the brightness to the lowest level possible. Modify the btn_led_off_cb() function as shown below.

```
static void
btn_led_off_cb(void *data, Evas_Object *obj, void *event_info)
{
    device_flash_set_brightness(0);
    device_led_stop_custom();
}
```

You can forcibly turn it off using the device_flash_set_brightness() function to set the brightness to 0.

5) Related APIs

device/led.h: a LED-controlling library.

device_flash_get_max_brightness(int *): an API that requests the maximum brightness of the LED backlight.

int device_flash_get_brightness(int *brightness): an API that requests the current brightness of the LED backlight.

int device_flash_set_brightness(int brightness): an API that specifies the brightness of the LED.

int device_led_play_custom(int on, int off, unsigned int color, unsigned int flags): an API that starts LED On.

int device_led_stop_custom(void): an LED Off API.

47. Event for Rotating the Screen Orientation

Orientation refers to the direction in which your phone rotates. Portrait means vertical; Landscape means horizontal. We will find out how to check the current orientation of your phone and request the Orientation event.

1) Requesting the Orientation

Create a new source project and specify the project name as 'OrientationEvent.' After the source project is created, open the source file (~.c) under the src folder and add new code at the end of the create_base_gui() function.

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

    // Show now orientation
    show_orientation(ad, NULL, NULL);
}
```

The show_orientation() function requests the current screen orientation, and then outputs it to the Label widget. Next, create a new function on top of the create_base_gui() function.

```
// 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;
    }
}
```

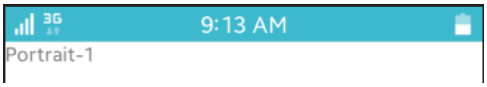
The show_orientation() function requests the Orientation, and then outputs it to the Label widget.

elm_win_rotation_get(const Evas_Object *) is an API that returns the current Orientation. The returned types are as follows:

- APP_DEVICE_ORIENTATION_0 : Portrait First
- APP_DEVICE_ORIENTATION_90 : Landscape First

- APP_DEVICE_ORIENTATION_180 : Portrait Second
- APP_DEVICE_ORIENTATION_270 : Landscape Second

Build and run the example. You will see the ‘Portrait-1’ text in the Label widget.



Portrait-1

2) Rotating the Screen in the Emulator

We will implement a feature that outputs the orientation when the Button is tapped. Create a new function on top of the create_base_gui() function.

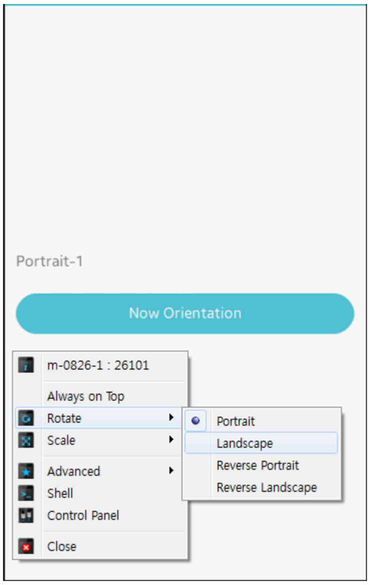
```
static void
my_box_pack(Evas_Object *box, Evas_Object *child,
                            double h_weight, double v_weight, double h_align, double
v_align)
{
   /* create a frame we shall use as padding around the child widget */
   Evas_Object *frame = elm_frame_add(box);
   /* use the medium padding 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_EXPAN
D);
        /* 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);
}
```

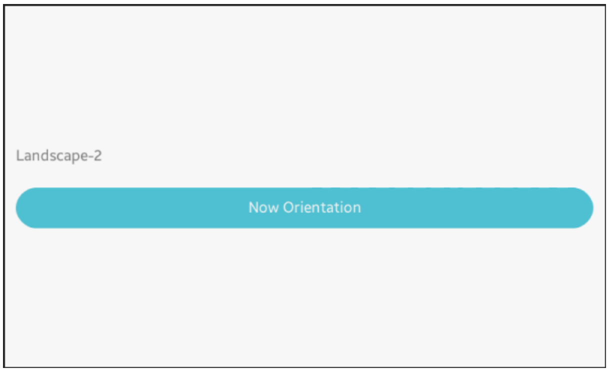
Then, add a Button-creating code to the create_base_gui() function.

```
        /* 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_EX
PAND);
```

```
        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, EVAS_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, (void *)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);
```

Let us now rotate the emulator. Right-click the emulator and select [Rotate > Landscape] from the shortcut menu.





As the screen orientation rotates, tap the Button. This time, you will see the ‘Landscape-2’ text.

Change it back to Portrait and tap the Button to display Portrait-1.

3) Rotating the Orientation

We are going to implement a feature that changes the Orientation when you tap the Button. Add the second Button-creating code to the create_base_gui() function.

```
                /* Button-1 */
                btn = elm_button_add(ad->win);
                elm_object_text_set(btn, "Now Orientation");
                evas_object_smart_callback_add(btn, "clicked", show_orientation, (void *)a
d);
                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_change
_cb, (void *)ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, 0.0);
        }
    }
```

Then, create a callback function of the newly created Button on top of the create_base_gui() function.

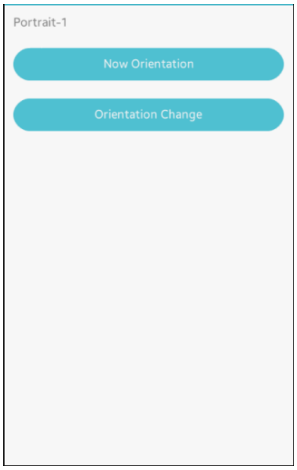
```
// 'Orientation Change' Button event function
static void
btn_orientation_change_cb(void *data, Evas_Object *obj, void *event_info)
{
   appdata_s *ad = (appdata_s*)data;
   // Get orientation
   int result = elm_win_rotation_get(ad->win);

   if( result == APP_DEVICE_ORIENTATION_0 || result == APP_DEVICE_ORIENTATION_
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);
}
```

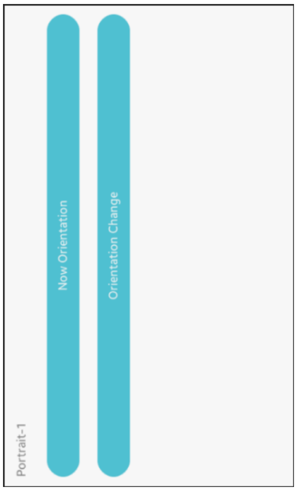
This code identifies the current Orientation and changes it as needed, from Portrait to Landscape and vice-versa.

elm_win_rotation_with_resize_set(Evas_Object *, int) is an API that changes the orientation. The option types are as follows:

- APP_DEVICE_ORIENTATION_0 : Portrait First
- APP_DEVICE_ORIENTATION_90 : Landscape First
- APP_DEVICE_ORIENTATION_180 : Portrait Second
- APP_DEVICE_ORIENTATION_270 : Landscape Second

Run the example again and tap the second Button. Each time you tap the Button, the Landscape mode and Portrait mode appear in turn.





4) Requesting an Event for Changing the Orientation

Let's make a request for the event when the Orientation changes. Add new code at the end of the create_base_gui() function.

```
// 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_rotation_changed_cb, ad);
}
```

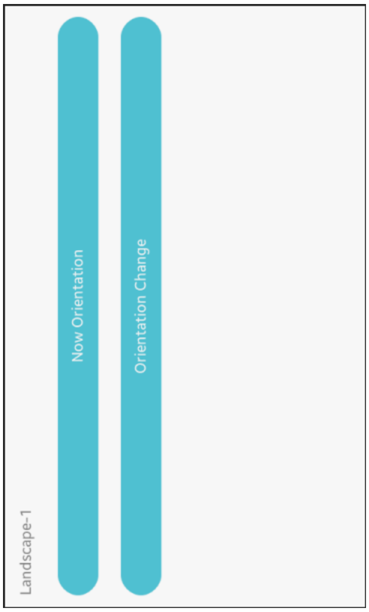
evas_object_smart_callback_add(Evas_Object *, char *, Evas_Smart_Cb, void *) is an API that specifies an event callback function for a smart object, such as a Layout container or Button widget. You can request the event for changing the Orientation by passing Win to the first parameter and specifying "rotation,changed" for the second parameter.

We are ready to create an event function for changing the Orientation. Create a new function on top of the create_base_gui() function.

```
// 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);
}
```

If the Orientation changes, the function above is invoked. Request the Orientation information to display it on the screen.

Run the example again and tap the second Button. If the screen orientation changes, Landscape-1 automatically appears in the Label widget.



5) Fixing the Orientation

We are going to lock the Orientation so that it stays in place when you rotate the phone. The allowable Orientation types are defined in the beginning of the create_base_gui() function.

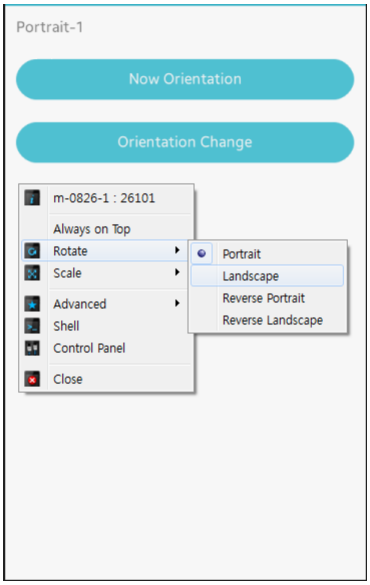
```
int rots[4] = { 0, 90, 180, 270 };
elm_win_wm_rotation_available_rotations_set(ad->win, (const int *)(&rots), 4);
```

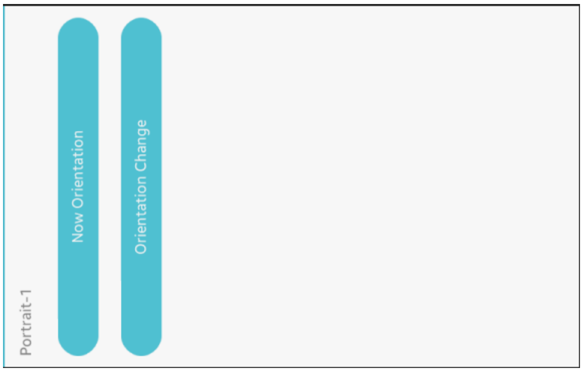
Change this as shown below.

```
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) is an API that specifies the allowable types of Orientation. It passes the array that holds the angle to the second parameter and the data count saved in the array to the third parameter.

Run the example again, and then right-click and select [Rotate > Landscape] from the shortcut menu. While the emulator rotates, the Orientation mode remains intact.





6) Related APIs

int elm_win_rotation_get(const Evas_Object *obj): an API that returns the current Orientation. The returned types are as follows:

- 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): an API that changes the Orientation. The option types are as follows:

- 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): an API that specifies an event callback function for a smart object, such as a Layout container or Button widget. You can request the event for changing the Orientation by passing Win to the first parameter and specifying "rotation,changed" for the second parameter.

void elm_win_wm_rotation_available_rotations_set(Evas_Object *obj, const int *rotations, unsigned int count): an API that specifies the allowable types of Orientation. It passes the array that holds the angle to the second parameter and the data count saved in the array to the third parameter.

48. Hardware Key Events and Debug Mode

You must program the use of the Up/Down keys for adjusting the volume when you build an audio/video player or a game/instrument app. In addition, you should be able to perform a specific action (for instance, displaying the menu when pressing the hardware Menu button). Typically, pressing the hardware Back button takes you back to the previous screen or exits the app. Note, however, that you need to perform other actions (for instance, displaying a warning popup) from time to time. In this example, we will learn how to request hardware key events.

1) Requesting a Hardware Key Event

Create a new source project and specify the project name as 'HardwareKeyEvent.' After the source project is created, open the source file (~.c) under the src folder and add new code at the end of the create_base_gui() function.

```
    /* 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 represents a callback function of the hardware key down event.

We are now going to create a callback function. Add new code on top of the create_base_gui() function.

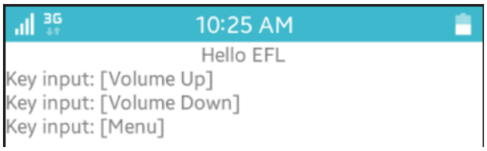
```
static void
on_keydown_cb(void *data, Evas *evas, Evas_Object *o, void *event_info)
{
    appdata_s* ad = data;
    Evas_Event_Key_Down *ev = event_info;

    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() is an event function for the hardware key. The first parameter receives user data; the second parameter receives the object where an event has occurred; and the third parameter receives the event information.

The event information is saved in Evas_Event_Key_Down format. The keyname property holds key values in string format. The types of key values are as follows:

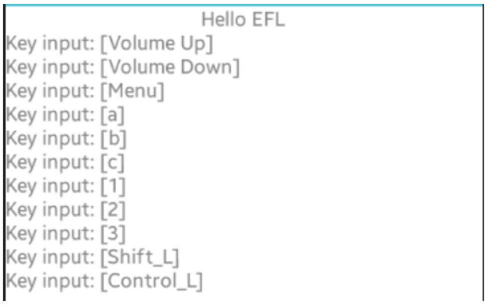
- XF86Menu: Menu key
- XF86Home: Home key
- XF86Back: Back key
- XF86PowerOff: Power key
- XF86AudioRaiseVolume: Volume Up key
- XF86AudioLowerVolume: Volume Down key

Build and run the example. Press the Volume up, Volume Down, and Menu hardware keys. The key types are displayed in the Label widget.



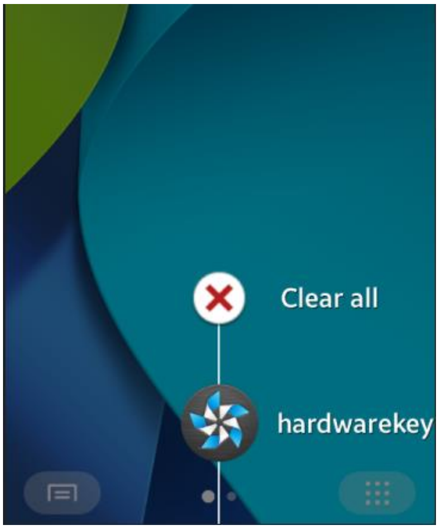
```
10:25 AM
Hello EFL
Key input: [Volume Up]
Key input: [Volume Down]
Key input: [Menu]
```

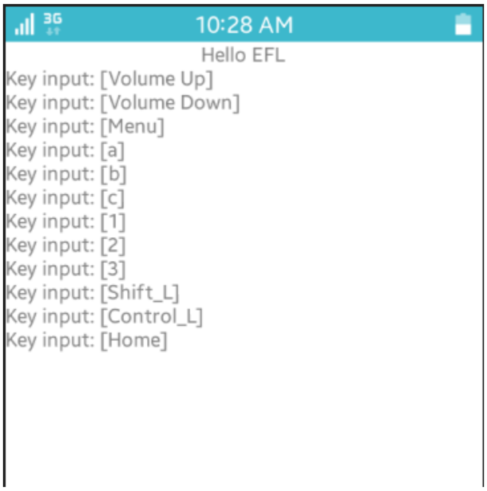
Press an alphanumeric key plus Ctrl+Shift on your keyboard. The value of each key is displayed. If your phone has a built-in hardware keypad, you may request individual key input.



```
Hello EFL
Key input: [Volume Up]
Key input: [Volume Down]
Key input: [Menu]
Key input: [a]
Key input: [b]
Key input: [c]
Key input: [1]
Key input: [2]
Key input: [3]
Key input: [Shift_L]
Key input: [Control_L]
```

Now, press the Home key. The app disappears. Press and hold the Home key again to see the list of apps you have run so far. Select HardwareKeyEvent from the list. You will see the example again and confirm that the Home key has been accepted.

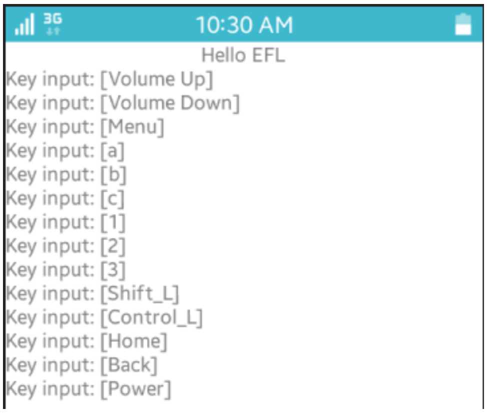




```
10:28 AM
Hello EFL
Key input: [Volume Up]
Key input: [Volume Down]
Key input: [Menu]
Key input: [a]
Key input: [b]
Key input: [c]
Key input: [1]
Key input: [2]
Key input: [3]
Key input: [Shift_L]
Key input: [Control_L]
Key input: [Home]
```

Now, press the Back key. Once the app has disappeared, press and hold the Home key and choose HardwareKeyEvent from the app list.

Next, press the Power key. When the screen turns Off, press the Home key and drag over the screen to unlock it.



10:30 AM

Hello EFL

```
Key input: [Volume Up]
Key input: [Volume Down]
Key input: [Menu]
Key input: [a]
Key input: [b]
Key input: [c]
Key input: [1]
Key input: [2]
Key input: [3]
Key input: [Shift_L]
Key input: [Control_L]
Key input: [Home]
Key input: [Back]
Key input: [Power]
```

2) Checking with the Log Message

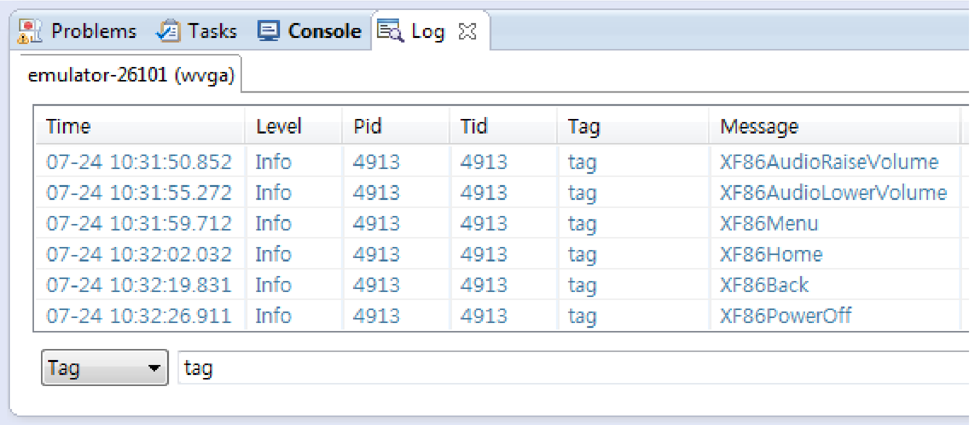
It is difficult to check the Home key, Back key, and Power key right away. We will use the Log message to facilitate this. Add a new line of code to the on_keydown_cb() function.

```
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  );
```

This code outputs the key value to a Log message.

Choose Log from the panels of Eclipse as below. Select the Tag in the combo box and enter 'tag' in the edit box on the right.

Run the example again and press the Hardware key. Pressing the Home key, Back key, or Power key may cause the screen to disappear, but the message remains in place.



Time	Level	Pid	Tid	Tag	Message
07-24 10:31:50.852	Info	4913	4913	tag	XF86AudioRaiseVolume
07-24 10:31:55.272	Info	4913	4913	tag	XF86AudioLowerVolume
07-24 10:31:59.712	Info	4913	4913	tag	XF86Menu
07-24 10:32:02.032	Info	4913	4913	tag	XF86Home
07-24 10:32:19.831	Info	4913	4913	tag	XF86Back
07-24 10:32:26.911	Info	4913	4913	tag	XF86PowerOff

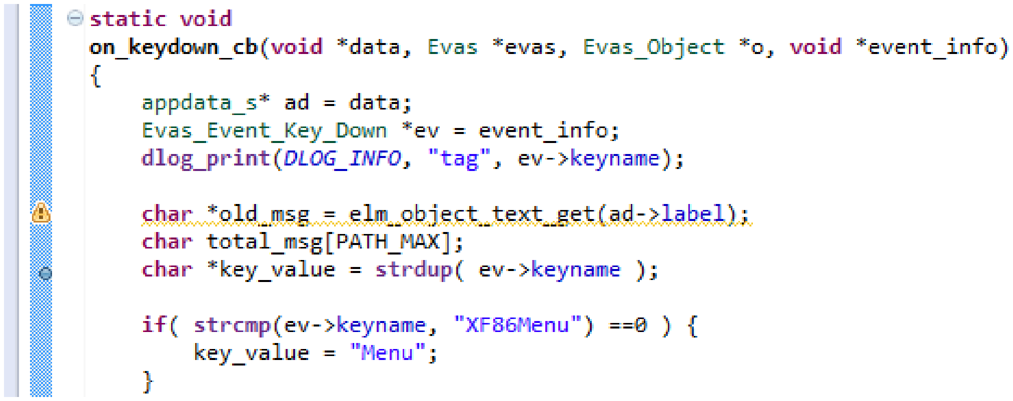
Tag: tag

3) Debug Mode

We will learn how to specify a break point at a specific source code location and check the values of the variables in real time.

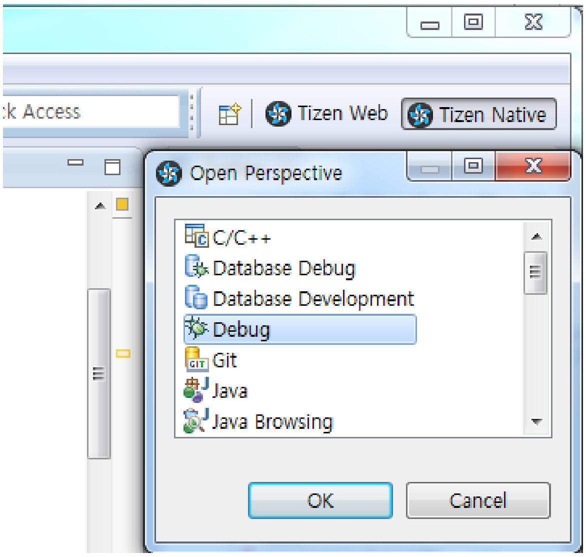
Click the border of the Edit window left of the next code of the on_keydown_cb() function. You will see a light blue circle. This is the break point.

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";
    }
```

Let's change this to debug mode. Choose Debug from the tab buttons in the upper right corner of Eclipse. If you do not see a Tab button called Debug, then choose Open Perspective. In the popup window that appears, choose Debug from the list.



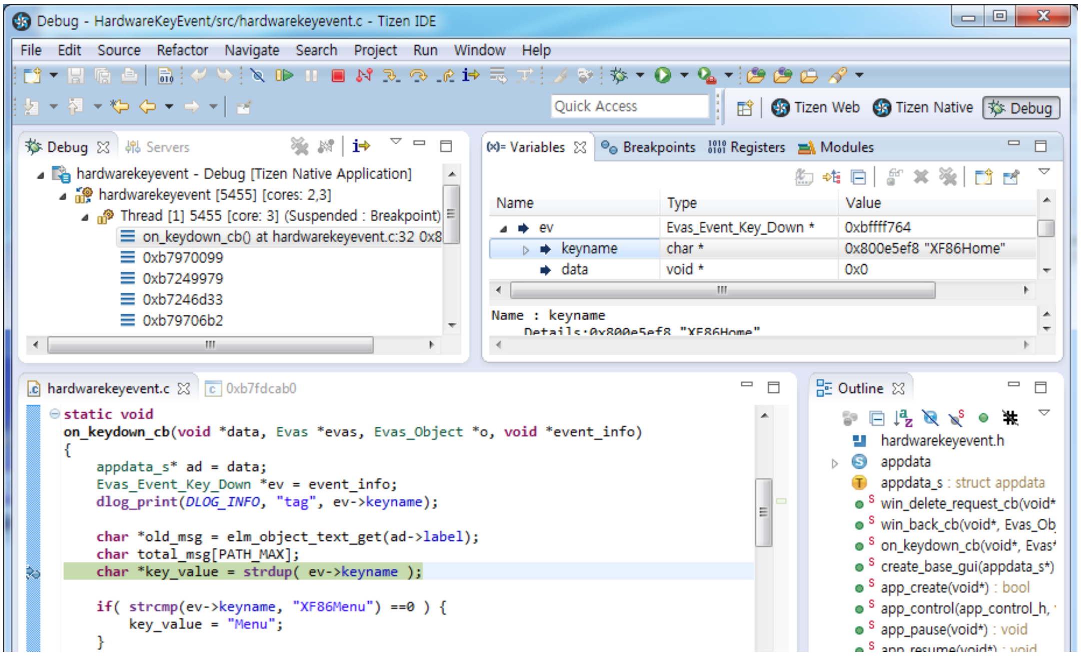
Let's start debugging now. Typically, Ctrl+F11 is used to run an app; for debugging, F11 is preferred. You may also choose [Run > Debug] from the main menu.

If the app stops running when it reaches the main() function, press F8 and skip to the next step.

If it is running, press the Menu key. An arrow is displayed at the break point, and it stops working.

In the Variables panel in the upper right corner of Eclipse, you will see a list of variables. Choose ev here to open a tree list.
A key value is displayed on the right side of the keyname in the sub properties.

To continue, press F8 and go to the next step.


Debug - HardwareKeyEvent/src/hardwarekeyevent.c - Tizen IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Tizen Web
Tizen Native
Debug
Debug
Servers
Variables
Breakpoints
Registers
Modules
hardwarekeyevent.c
0xb7fdcab0
Outline

```
hardwarekeyevent - Debug [Tizen Native Application]
  hardwarekeyevent [5455] [cores: 2,3]
    Thread [1] 5455 [core: 3] (Suspended : Breakpoint)
      on_keydown_cb() at hardwarekeyevent.c:32 0x8
      0xb7970099
      0xb7249979
      0xb7246d33
      0xb79706b2
```

Name	Type	Value
ev	Evas_Event_Key_Down *	0xbffff764
keyname	char *	0x800e5ef8 "XF86Home"
data	void *	0x0

```
Name : keyname
   Details:0x800e5ef8 "XF86Home"
```

```
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";
    }
```

```
hardwarekeyevent.h
appdata
appdata_s : struct appdata
win_delete_request_cb(void*
win_back_cb(void*, Evas_Ob
on_keydown_cb(void*, Evas*
create_base_gui(appdata_s*)
app_create(void*) : bool
app_control(app_control_h,
app_pause(void*) : void
app_resume(void*) : void
```

Use the following keyboard shortcuts in debug mode:

- F11: Start debugging
- F8: Next step
- F5: Step into
- F6: Step over
- F7: Step return
- Ctrl+F2: Stop debugging

4) Related APIs

You can request a hardware key down event by passing EVAS_CALLBACK_KEY_DOWN to the second parameter of the void evas_object_event_callback_add(Evas_Object *obj, Evas_Callback_Type type, Evas_Object_Event_Cb func, void *data) function.

The hardware key event information is saved in Evas_Event_Key_Down format. The keyname property holds key values in string format. The types of key values are as follows:

- XF86Menu: Menu key
- XF86Home: Home key
- XF86Back: Back key
- XF86PowerOff: Power key
- XF86AudioRaiseVolume: Volume Up key
- XF86AudioLowerVolume: Volume Down key

int dlog_print(log_priority prio, const char *tag, const char *fmt, ...) : an API that outputs Log messages in real time.

Use the following keyboard shortcuts in debug mode:

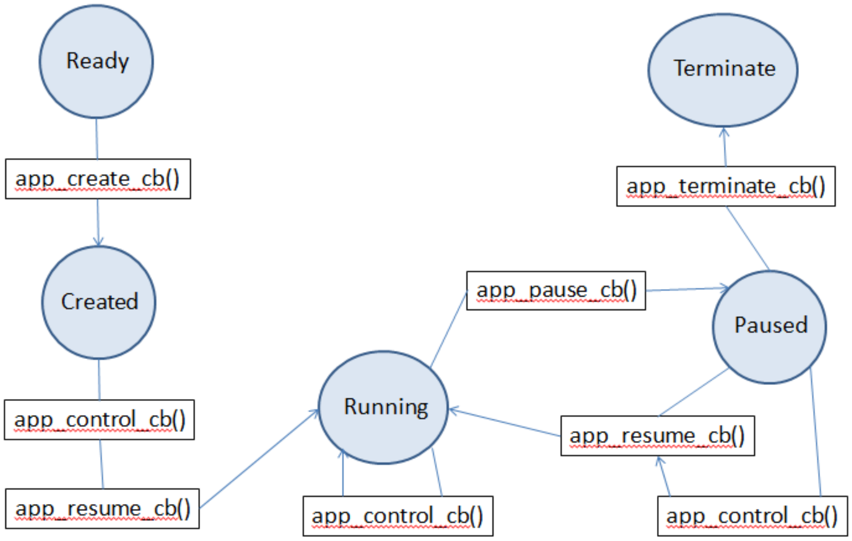
- F11: Start debugging
- F8: Next step
- F5: Step into
- F6: Step over
- F7: Step return
- Ctrl+F2: Stop debugging

49. Lifecycle and Debug Mode

When you run the app for the first time, it creates a container and a widget that make up the UI. At this point, you can easily allocate memory to the variables. When the app is shut down, you need to delete the memory. If you need to play background music, play the music that plays when you first run the app. This must be stopped when the app is terminated. When the app hides behind the background, the animation that appears on the screen should be stopped in order to reduce the CPU load. The animation must then resume when switching from background to foreground. We will look at the lifecycle in this example.

1) Events for the Lifecycle

For the lifecycle of Tizen native apps, refer to the following diagram:



The app execution generally follows this order: app_create_cb() => app_control_cb() => app_resume_cb() functions.

The app_terminate_cb() is invoked when you exit the app.

When the app is paused, such as when switching to background mode, the app_pause_cb() function is invoked.

When the app resumes, the app_resume_cb() function is invoked.

If another app makes a request for an app Launch, the app_control_cb() function in invoked.

2) Event Functions for the Lifecycle

Create a new source project and specify the project name as ‘LifeCycle.’ After the source project is created, open the source file (~.c) under the src folder and add new code at the top.

```
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);
}
```

The show_message() adds a new message to the Label widget. We have placed it at the top so it can be easily invoked from anywhere.

win_back_cb() is an even callback function that is run when the Back button is tapped. An appdata object is passed to the first parameter.

Underneath the source file, you will see various event callback functions and their definitions. To one of these functions, add a code that is associated with the lifecycle.

```
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() is a callback function executed when you create an app. Call the create_base_gui() function, which creates a UI object from here.

app_control() is a callback function that is run when there is an app Launch request in another app.

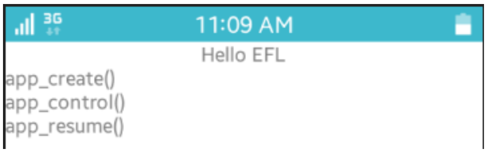
app_pause() is a callback function executed when the app switches to background mode.

app_resume() is a callback function executed when the app switches from background mode to foreground mode.

app_terminate() is a callback function that is executed when you exit the app.

You specify these five callback functions in the main() function. The main() function is where you normally make changes to the callback functions.

Build and run the example. Once the app starts running, confirm that three additional functions have been run. The order is app_create() => app_control() => app_resume().

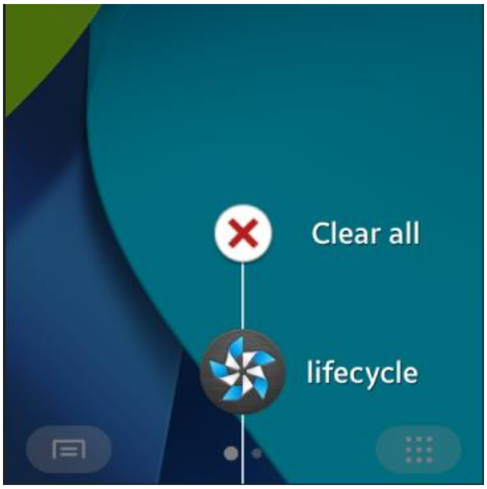


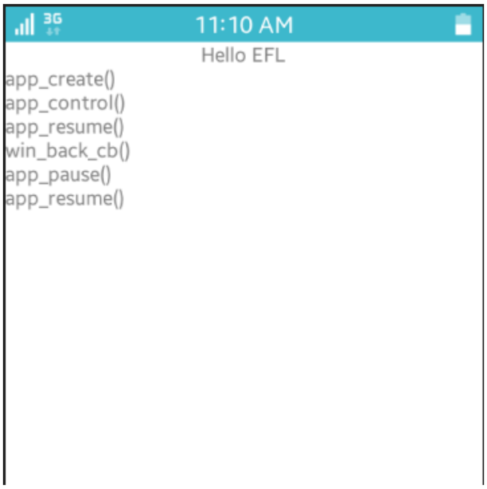
```
app_create()
app_control()
app_resume()
```

Tap the Back button if the app is running. The app is nowhere to be found, but that does not mean the app is down. It is still running in background mode.

We will invoke the app again. Press and hold the Home button to see a list of apps you have run up to this point. Choose LifeCycle from the list.

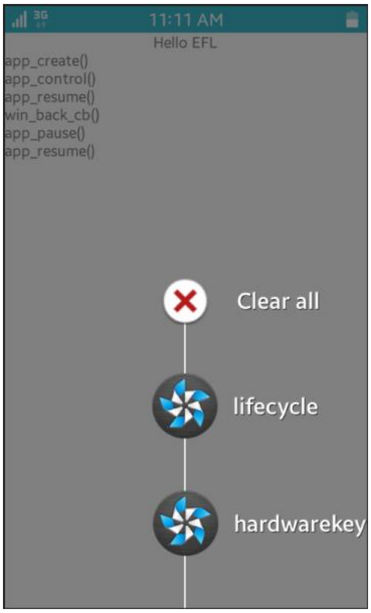
You will see the example again; confirm that three additional functions have been run. The app runs in the following order: win_back_cb() => app_pause() => app_resume(). The Back button will run the win_back_cb() function. When the app switches to background mode, the app_resume() function is executed. If it switches to foreground mode, the app_resume() function is executed.





```
app_create()
app_control()
app_resume()
win_back_cb()
app_pause()
app_resume()
```

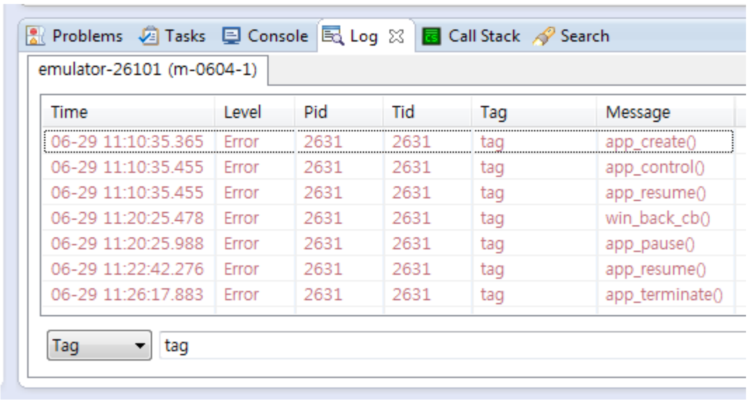
We are now going to exit the app. Press and hold the Home button to see a list of apps. Tap Clear all to exit all the apps.



```
app_create()
app_control()
app_resume()
win_back_cb()
app_pause()
app_resume()
```

The app has been shut down. New messages pass by in a flash, so there is no way of checking them. Luckily, we have something called real-time Log messages. Select the Log panel at the bottom of Eclipse and choose the Tag in the combo box. Then, enter 'tag' in the edit box on the right.

The output message is displayed in the Label widget. You will see that app_terminate() has been added to the bottom. This means that the app_terminate() function was executed when you shut down the app.



emulator-26101 (m-0604-1)

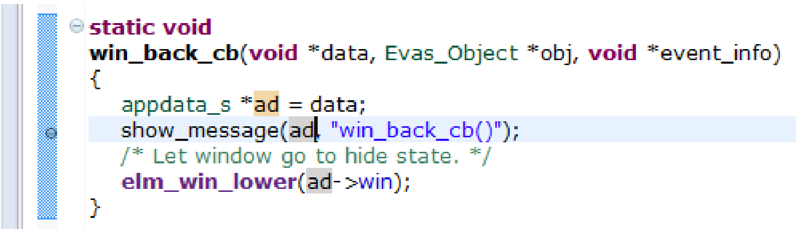
Time	Level	Pid	Tid	Tag	Message
06-29 11:10:35.365	Error	2631	2631	tag	app_create()
06-29 11:10:35.455	Error	2631	2631	tag	app_control()
06-29 11:10:35.455	Error	2631	2631	tag	app_resume()
06-29 11:20:25.478	Error	2631	2631	tag	win_back_cb()
06-29 11:20:25.988	Error	2631	2631	tag	app_pause()
06-29 11:22:42.276	Error	2631	2631	tag	app_resume()
06-29 11:26:17.883	Error	2631	2631	tag	app_terminate()

Tag ▾ tag

3) Debug Mode

We will learn how to specify a break point at a specific source code location and check the values of the variables in real time.

Go to the win_back_cb() function and tap the border of the Edit window on the left of the invoking code of show_message(). You will see a light blue circle. This is the break point.



```
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);
}
```

Similarly, specify a break point to the invoking code of show_message() in the following five functions:

- app_create()
- app_control()
- app_pause()
- app_resume()
- app_terminate()

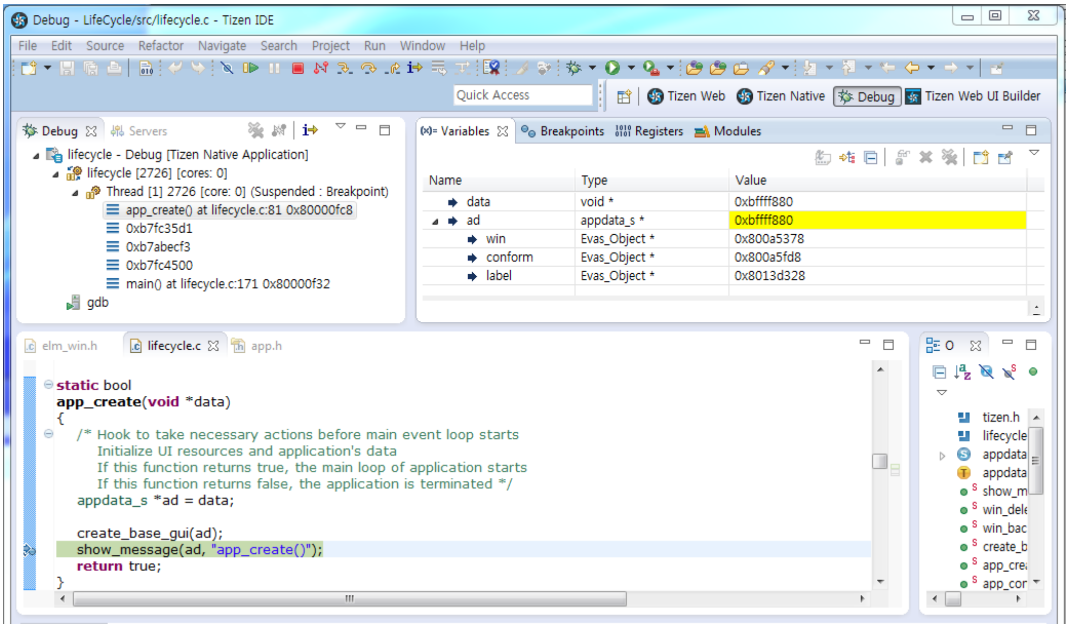
Let's change this to debug mode. Choose Debug from the tab buttons in the upper right corner of Eclipse.

Let's start debugging now. Typically, Ctrl+F11 is used to run an app; for debugging, F11 is preferred.

If the app stops running when it reaches the main() function, press F8 and skip to the next step.

If the app starts running, stop at the app_create() function. If you press F8, it will pass the break point and make another stop at the app_control() function.

Next, it stops at the app_resume() function one more time. If you press F8 again, the app will appear on the screen.



```
lifecycle - Debug [Tizen Native Application]
  lifecycle [2726] [cores: 0]
    Thread [1] 2726 [core: 0] (Suspended : Breakpoint)
      app_create() at lifecycle.c:81 0x80000fc8
      0xb7fc35d1
      0xb7abecf3
      0xb7fc4500
      main() at lifecycle.c:171 0x80000f32
  gdb
```

Name	Type	Value
data	void *	0xbffff880
ad	appdata_s *	0xbffff880
win	Evas_Object *	0x800a5378
conform	Evas_Object *	0x800a5fd8
label	Evas_Object *	0x8013d328

```
static bool
app_create(void *data)
{
    /* Hook to take necessary actions before main event loop starts
       Initialize UI resources and application's data
       If this function returns true, the main loop of application starts
       If this function returns false, the application is terminated */
    appdata_s *ad = data;

    create_base_gui(ad);
    show_message(ad, "app_create()");
    return true;
}
```

If you tap the Back button, it will stop at the win_back_cb() function. If you press F8, it will stop at the app_pause() function.

If you press F8 again, the app disappears from the screen.

If you press and hold the Home key, and then select LifeCycle from the app list, it will stop at the app_resume() function.

Pressing F8 will bring it back on the screen.

Use the following keyboard shortcuts in debug mode:
- F11: Start debugging
- F8: Next step
- F5: Step into
- F6: Step over
- F7: Step return
- Ctrl+F2: Stop debugging

4) Related APIs

void win_back_cb(void *data, Evas_Object *obj, void *event_info): an event callback function that is executed when the user taps the Back button. An appdata object is passed to the first parameter.

bool app_create(void *data): a callback function that is executed when you create the app. Call the create_base_gui() function, which creates a UI object from here.

void app_control(): a callback function that is run when there is an app Launch request in another app.

void app_pause(): a callback function that is executed when the app switches to background mode.

void app_resume(): a callback function that is executed when the app switches from background mode to foreground mode.

void app_terminate(): a callback function that is executed when you exit the app.

Use the following keyboard shortcuts in debug mode:

- F11: Start debugging
- F8: Next step
- F5: Step into
- F6: Step over
- F7: Step return
- Ctrl+F2: Stop debugging

50. Using a Notify

Use Notify when you want to pass a new message to the user. You can make the message disappear after a while or even add a widget such as a Button. You can also block the user's UI events and display them at the bottom.

1) Specifying a Timeout for a Notify

Create a new source project and specify the project name as 'NotifyEx.' After the source project is created, open the source file (~.c) under the src folder and add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
        /* 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);
}
```

Then, add new code to the create_base_gui() function. This code creates a Box container, a Button, and a Notify object.

```
    /* 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);

            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, notify);
            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);
```

The create_notify_top_timeout() function creates a Notify object and specifies the Timeout. Next, create a new function on top of the create_base_gui() function.

```
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 hide 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 *) is an API that creates a Notify object.

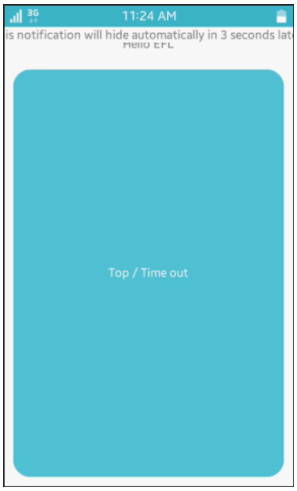
elm_notify_align_set(Evas_Object *, double, double) is an API that specifies the position of the Notify on the aspect ratio. For the second parameter, specify the horizontal position. It is placed on the left for 0, in the middle for 0.5, and on the right for 1. For the third parameter, specify the vertical position. It is placed at the top for 0, in the center for 0.5, and at the bottom for 1.

elm_notify_timeout_set(Evas_Object *, double) is an API that specifies a Timeout for the Notify. Specify an interval to the second parameter in seconds.

The next code creates a Box container in the Notify and a Label widget in the Box container.

The btn_click_cb() function displays the Notify on the screen when the user taps the Button.

Build and run the example. If you tap the Button, a Notify appears for 3 seconds, and then disappears.



is notification will hide automatically in 3 seconds lat

Hello EFL

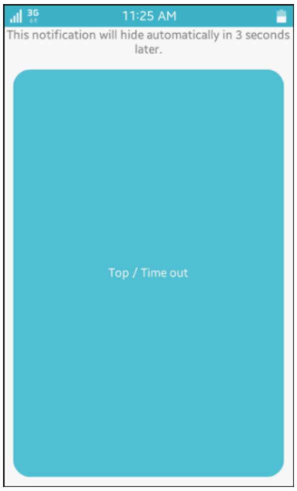
2) Resizing Notify

Sometimes, the text string displayed in a Notify is truncated. We will try to fix this by changing the App base scale. Add a new line of code to the app_create() function.

```
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) is an API that changes the App base scale. The default scale is 1.0. As the scale value grows, the Notify area becomes smaller.

Run the example again and tap the Button. This time, the text in the Notify appears correctly.



This notification will hide automatically in 3 seconds later.

3) Adding a Button Widget to Notify

We will implement, by adding a Button to the Notify, a feature that makes the Notify disappear when you tap the Button instead of when Timeout occurs. Add new code to the create_base_gui() function.

```
/* 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, notify);
my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
```

```
        }
    }
```

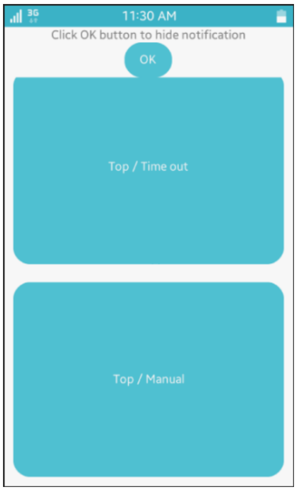
This is the code that creates the second Notify and the second Button. Add two new functions on top of the create_base_gui() function.

```
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 notification
</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(58));
    elm_box_pack_end(box, btn);
    evas_object_show(btn);
    evas_object_smart_callback_add(btn, "clicked", btn_hide_notify_cb, notify);

    elm_object_content_set(notify, box);

    return notify;
}
```

The btn_hide_notify_cb() hides Notify. Tapping the Button added to the Notify will invoke this function.

The create_notify_top_manual() creates the second Notify object. This code creates a Box container in the Notify and a Label widget and a Button widget in the Box container.

Run the example again and tap the second Button. The Notify is displayed, and you will see that a Label and a Button have been added. Tapping the Button will cause the Notify to disappear.



4) Notify for Blocking Events

We will implement a feature that blocks the user's events while a Notify is displayed. Add new code to the create_base_gui() function.

```
                /* 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, notify);
                my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
```

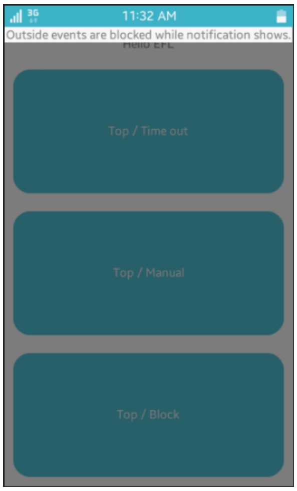
```
    }
```

This is the code that creates the third Notify and the third Button. Create a new function on top of the create_base_gui() function.

```
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 events) */
    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 blocked while n
otification 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) is a function that determines whether or not to allow user events when a Notify is in Show state. The user events will be blocked if you pass EINA_FALSE to the second parameter.

Run the example again and tap the third Button. A Notify appears for 3 seconds and disappears again. While the Notify is in Show state, tapping the other Buttons has no effect.


Outside events are blocked while notification shows.

5) Changing the Position of a Notify

We will implement a feature that displays a Notify at the bottom. Add new code to the create_base_gui() function.

```
            /* 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, notify);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

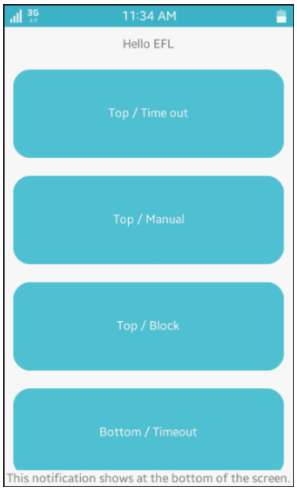
This is the code that creates the fourth Notify and the fourth Button. Create a new function on top of the create_base_gui() function.

```
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 at 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) is an API that specifies the position of the Notify on the aspect ratio. For the second parameter, specify the horizontal position. It is placed on the left for 0, in the middle for 0.5, and on the right for 1. For the third parameter, specify the vertical position. It is placed at the top for 0, in the center for 0.5, and at the bottom for 1.

Run the example again and tap the fourth Button. A Notify appears briefly at the bottom, and then disappears.



This notification shows at the bottom of the screen.

6) Related APIs

Evas_Object *elm_notify_add(Evas_Object *parent): an API that creates a Notify object.

void elm_notify_align_set(Evas_Object *obj, double horizontal, double vertical): an API that specifies the position of the Notify on the aspect ratio. For the second parameter, specify the horizontal position. It is placed on the left for 0, in the middle for 0.5, and on the right for 1. For the third parameter, specify the vertical position. It is placed at the top for 0, in the center for 0.5, and at the bottom for 1.

void elm_notify_timeout_set(Evas_Object *obj, double timeout): an API that specifies a Timeout for the Notify. Specify an interval to the second parameter in seconds.

void elm_notify_allow_events_set(Evas_Object *obj, Eina_Bool allow): a function that determines whether or not to allow user events when the Notify is in Show state. The user events will be blocked if you pass EINA_FALSE to the second parameter.

51. Using a Acceleration Sensor

You can measure the shaking of your phone with the accelerator sensor. You can measure the directions of the X-, Y-, and Z-axes. In addition, you can choose to apply gravity or discard it and separate those results. To test the accelerator sensor in the emulator, use the Control Panel.

1) Determining Whether the Accelerator Sensor Is Supported or Not

Create a new source project and specify the project name as ‘SensorAcceleration.’ After the source project is created, open the source file (~.c) under the src folder and add a library header file as well as variables.

```
#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 is a header file for various sensor libraries.

We are going to display whether or not the accelerator sensor is supported on label0, the current acceleration value on label1 and the maximum value of acceleration on label2.

Create a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

Create two new functions on top of the create_base_gui() function.

```
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" : "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_EXPAN
D);
        /* 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() function identifies whether the accelerator sensor is supported or not, and then displays the result in the first Label widget.

sensor_is_supported(sensor_type_e, bool *) is an API that requests whether or not a specific sensor is supported. Passing SENSOR_ACCELEROMETER to the first parameter makes the second parameter return whether or not the sensor is supported.

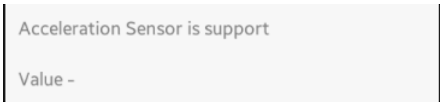
my_box_pack() is a function that adds a widget to a Box container.

All you have to do is invoke the show_is_supported() function while the app is running. Invoke the function above at the end of the create_base_gui() function.

```
    /* 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 */
        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_EXPAND);
        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);
}
```

We have added two Label widgets. Plus, we have invoked a function to determine whether or not the sensor is supported.

Build and run the example. If the accelerator sensor is supported, you will see the message 'Acceleration Sensor is supported.' Not all smartphones support the sensor. If this is the case, please test it in the emulator.



Acceleration Sensor is support

Value -

2) Requesting the Event for the Accelerator Sensor

We will implement a feature that requests the corresponding event as you shake the device and displays that acceleration value on the screen. Add a structure for the sensor and a global variable to the top of the source file.

```
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 is a structure that includes a sensor object and an event listener variable.

sensor_info is a global variable of the sensorinfo structure.

To request a sensor event, start the listener. We will request an accelerator sensor event using the sensor object and event listener. Create two new functions above the create_base_gui() function.

```
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_data->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_listener);
    sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sensor_value, ad);
    sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() is an event callback function for the accelerator sensor. Output a new sensor value to the screen. The sensor data are passed to the second parameter. values[0] contains x-axis data, values[1] contains y-axis data, and values[2] contains z-axis data.

start_acceleration_sensor() is a callback function that starts the accelerator sensor and specifies the event callback function.

sensor_get_default_sensor(sensor_type_e, sensor_h *) is an API that returns a sensor object. Passing SENSOR_ACCELEROMETER to the first parameter returns an accelerator sensor object to the second parameter.

sensor_create_listener(sensor_h, sensor_listener_h *) is an API that creates an event listener. Passing a sensor object to the first parameter returns a listener object to the second parameter.

sensor_listener_set_event_cb(sensor_listener_h, unsigned int, sensor_event_cb, void *) is an API that specifies a callback function to the listener. The parameters follow this order: event listener, interval (in milliseconds), callback function name, and user data.

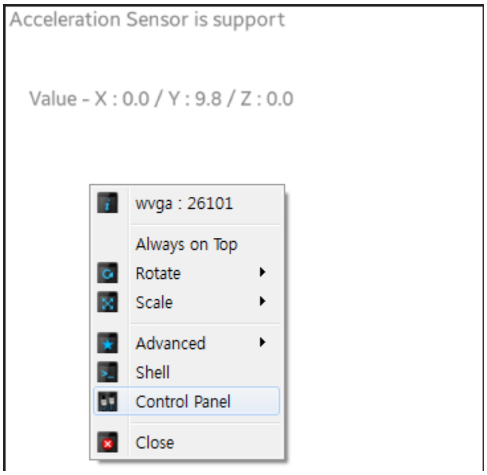
sensor_listener_start(sensor_listener_h) is an API that starts the listener.

We will operate the event listener automatically when the app starts running. Invoke the function above at the end of the create_base_gui() function.

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

    show_is_supported(ad);
    start_acceleration_sensor(ad);
}
```

Let's run the example again. To test on your smartphone, simply shake the device. Use the Control Panel for testing in the emulator.

Right-click the emulator and select the Control Panel from the shortcut menu.

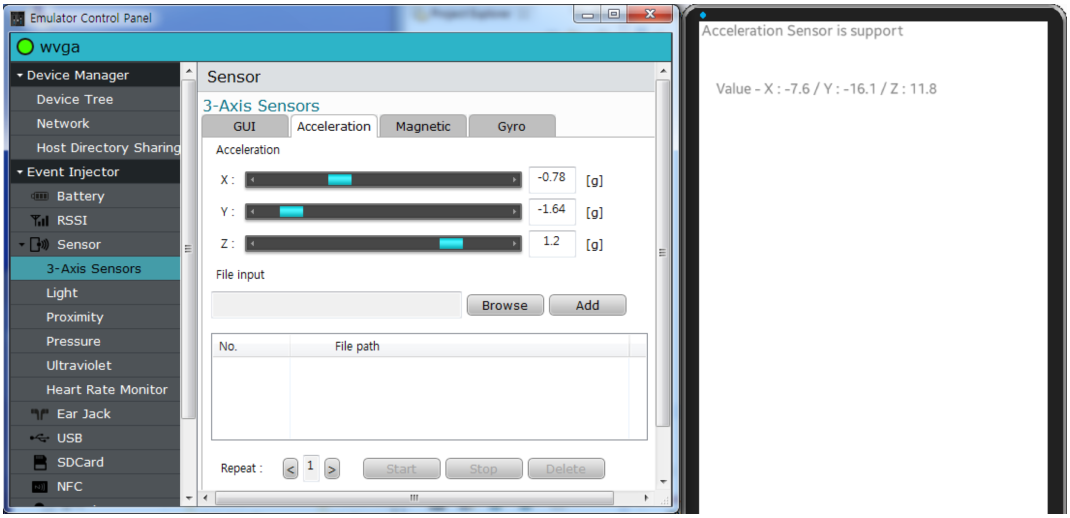


Acceleration Sensor is support

Value - X : 0.0 / Y : 9.8 / Z : 0.0

In the Control Panel, select [Event Injector > 3-Axis Sensors] from the tree list on the left, followed by Acceleration from the tab buttons on the right side of the screen.

Drag the three sliders one at a time. If the X, Y, and Z values change on the app screen, it means you have correctly received the accelerator data.



X : -0.78 [g]
Y : -1.64 [g]
Z : 1.2 [g]

Acceleration Sensor is support

Value - X : -7.6 / Y : -16.1 / Z : 11.8

3) Requesting the Maximum Acceleration Value

If you try testing it with your smartphone, the characters are hardly visible when you are shaking the phone. When you stop shaking to see the values, the downward direction is 9.8, and the rest is reset to 0. For that reason, we need a feature that saves the maximum value when testing on your phone.

Declare an array variable in number format at the top of the source file and reset it to 0. This variable saves the maximum acceleration value.

```
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};
```

Add new code to the create_base_gui() function.

```
            /* 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);
        }
    }
```

This code adds a Button widget and a Label widget. We will implement a feature that displays the maximum value of the accelerator sensor in the third Label and reset the maximum value to 0 when you tap the Button. Add two new functions and new code to the _new_sensor_value() function.

```
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 *use
r_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_data->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;
}
```

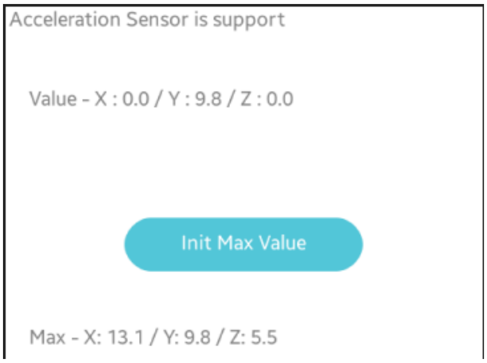
The get_absolute_max(float, float) function returns a higher value by changing two real numbers to absolute values.

The codes you added in the _new_sensor_value() function save the maximum values of X, Y, and Z axes in the global variable and output them to the Label widget.

The btn_clicked() function resets the maximum value saved in the global variable to 0 when you tap the Button.

Install the example and shake the phone. When you stop shaking, the value in the second Label is reset, but the maximum value remains intact in the third Label.

You can measure a new value by holding the Button and shaking again.


Acceleration Sensor is support

Value - X : 0.0 / Y : 9.8 / Z : 0.0

Max - X: 13.1 / Y: 9.8 / Z: 5.5

4) Requesting Pure Acceleration without Gravity

As you may have noticed in the result values, the acceleration we calculated includes the gravity. Now, let's obtain the acceleration value alone, excluding the gravity.

Make changes to the code of the start_acceleration_sensor() function as follows:

```
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.sensor);
    err = sensor_create_listener(sensor_info.sensor, &sensor_info.sensor_listener);
    sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sensor_value, ad);
    sensor_listener_start(sensor_info.sensor_listener);
}
```

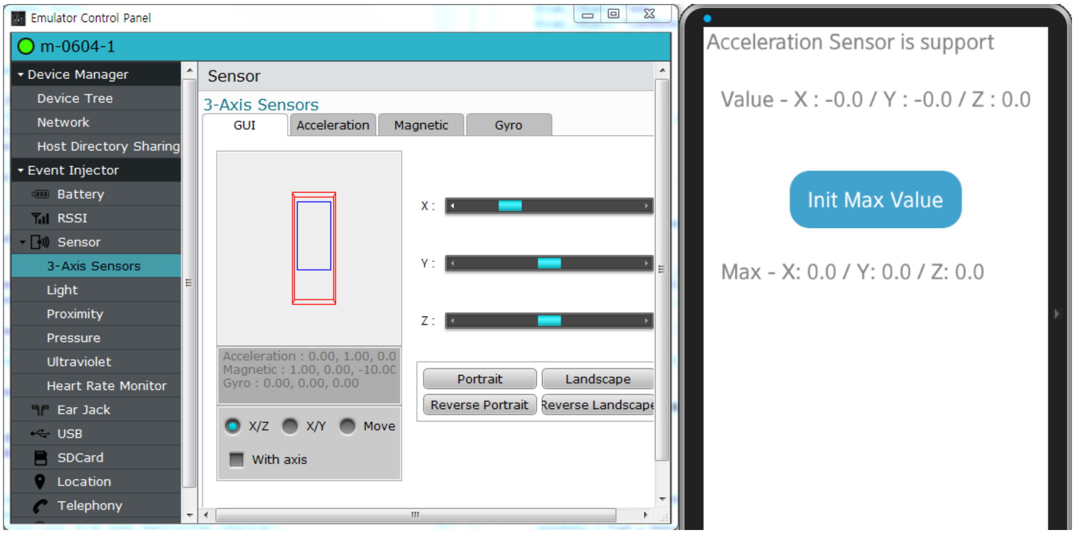
SENSOR_ACCELEROMETER is a sensor type representing the acceleration that includes the gravity.

SENSOR_LINEAR_ACCELERATION represents the accelerator sensor free of gravity.

Install the example and shake the phone. What you see now is the pure acceleration.

To test in the emulator, select [Event Injector > 3-Axis Sensors] in the Control Panel and click GUI from the tab buttons on the right.

Click the Portrait button and Landscape button back and forth. If you set the sensor type to SENSOR_ACCELEROMETER, the total of X, Y, and Z will be 9.8. With SENSOR_LINEAR_ACCELERATION, the total will be 0.



Acceleration : 0.00, 1.00, 0.0
Magnetic : 1.00, 0.00, -10.00
Gyro : 0.00, 0.00, 0.00

Acceleration Sensor is support

Value - X : -0.0 / Y : -0.0 / Z : 0.0

Max - X: 0.0 / Y: 0.0 / Z: 0.0

5) Related APIs

int sensor_is_supported(sensor_type_e type, bool *supported): an API that requests whether or not a specific sensor is supported. Passing SENSOR_ACCELEROMETER to the first parameter makes the second parameter return whether or not the accelerator sensor is supported.

- SENSOR_ACCELEROMETER: the accelerator sensor that includes the gravity.

- SENSOR_LINEAR_ACCELERATION: the accelerator sensor without the gravity.

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor): an API that returns a sensor object. Passing SENSOR_ACCELEROMETER to the first parameter returns an accelerator sensor object to the second parameter.

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener): an API that creates an event listener. Passing a sensor object to the first parameter returns a listener object to the second parameter.

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data): an API that specifies a callback function to the listener. / parameters: event listener, interval (in milliseconds), callback function name, and user data.

int sensor_listener_start(sensor_listener_h listener): an API that starts the listener.

52. Using a Gravity Sensor

The direction of the device can be estimated by the gravity sensor. The sensor helps to measure the directions of the X-, Y-, and Z-axes. Use the Control Panel to test the gravity sensor on the emulator.

1) Determining if the Gravity Sensor is Supported

Create a new source project and specify the project name as 'SensorGravity.' After the source project is created, open the source file (~.c) under the src folder and add library header files and variables.

```
#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 is a library header file for various sensors.

We are going to display whether the sensor is supported on label0, and the current gravity value on label1.

Create two functions on top of the create_base_gui() function.

```
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 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_EXPAN
D);
         /* 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() is a function that determines if the gravity sensor is supported and displays the result on the first Label widget.

sensor_is_supported(sensor_type_e, bool *) is an API that determines if a certain sensor is supported. Passing SENSOR_GRAVITY to the first parameter makes the second parameter return whether or not the sensor is supported.

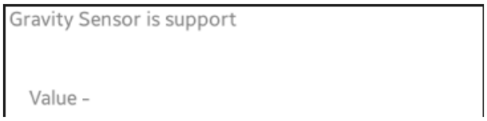
my_box_pack() is a function that adds a widget to a Box.

We need to call the show_is_supported() function when the app is run. Call the function above at the end of the create_base_gui() function.

```
    /* 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_EX
PAND);
        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);
}
```

We added a Box container and two Label widgets. Then, we called the function to determine if the sensor is supported.

Build and run the example. If the gravity sensor is supported, you will see the message 'Gravity Sensor is supported.' Some smartphones may not support the sensor. In such cases, you need to test it on the emulator.



Gravity Sensor is support

Value -

2) Requesting a Gravity Sensor Event

We will implement a feature that displays the gravity value on the screen by getting the event when the device direction changes. Add a sensor-related structure and global variables at the top of the source file.

```
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 is the structure including a sensor object and event listener variables.

sensor_info is a global variable of the sensorinfo structure.

Requesting a sensor event means to start the listener. We are now going to request a gravity sensor event by using the sensor object and event listener. Create two functions on top of the create_base_gui() function.

```
static void _new_sensor_value(sensor_h sensor, sensor_event_s *sensor_data, void *use
r_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_data->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_listener);
    sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sensor_value, ad);
    sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() is an event callback function for the gravity sensor. Output new sensor values on the screen.

The sensor data is passed to the second parameter as an array. The direction data of the X-, Y-, and Z-axes is saved in values[0], vales[1], and values[2], respectively.

start_gravity_sensor() is a function that starts the gravity sensor and specifies the event callback function.

sensor_get_default_sensor(sensor_type_e, sensor_h *) is an API that returns the sensor object. Passing SENSOR_GRAVITY to the first parameter returns the gravity sensor object to the second parameter.

sensor_create_listener(sensor_h, sensor_listener_h *) is an API that creates an event listener. Passing the sensor object to the first parameter returns the listener object to the second parameter.

sensor_listener_set_event_cb(sensor_listener_h, unsigned int,
sensor_event_cb, void *) is an API that specifies a callback function to the listener. The parameters listed in order are an event listener, time interval (in milliseconds), name of the callback function, and user data.

sensor_listener_start(sensor_listener_h) is an API that starts the listener.

We will make the event listener operate automatically when the app is run. Call the function above at the end of the create_base_gui() function.

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

show_is_supported(ad);
start_gravity_sensor(ad);
```

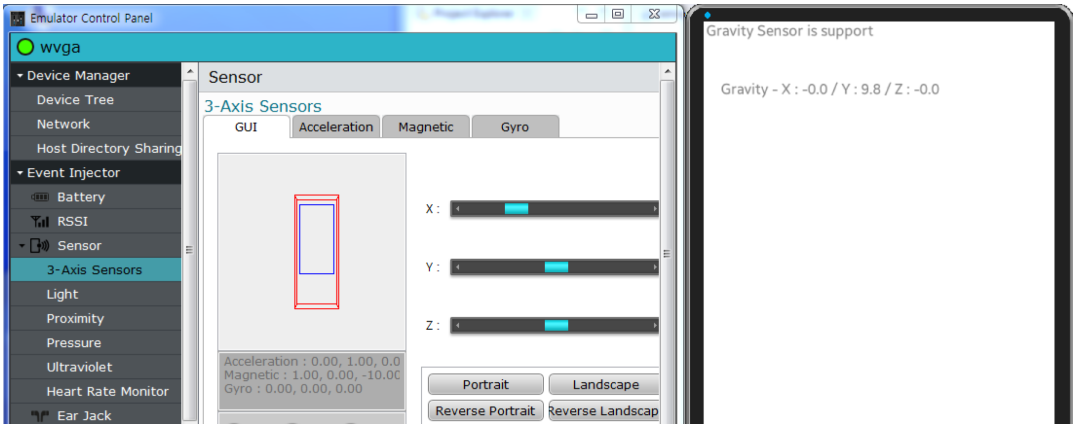
```
}
```

Run the example again. On a smartphone, simply rotate the phone for testing. On an emulator, use the Control Panel for testing.

Right-click the emulator and select Control Panel from the shortcut menu.

On the Control Panel, select [Event Injector > 3-Axis Sensors] from the list in tree structure on the left, and then click GUI from the tab buttons on the right side of the screen.

Clicking the Portrait button on the right side of the screen on the Control Panel displays 'X : 0.0 / Y : 9.8 / Z : 0.0' on the second Label widget of the app screen.

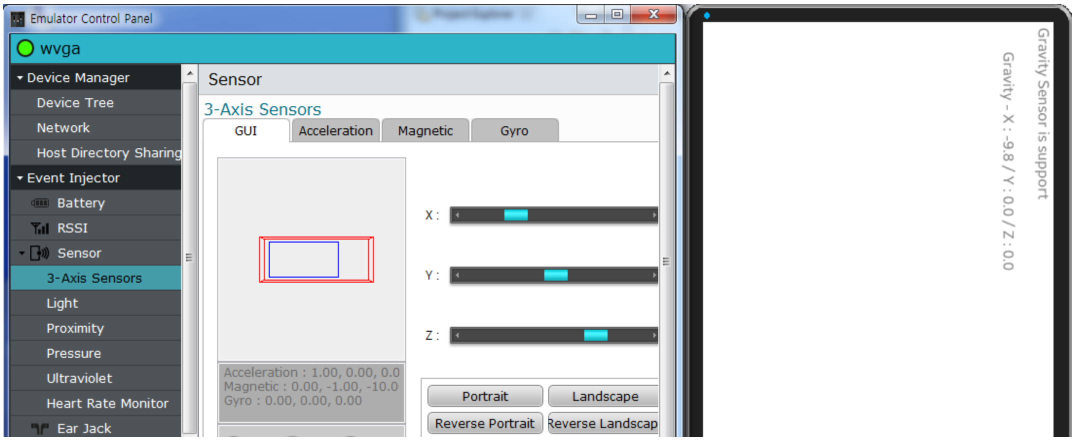


Acceleration : 0.00, 1.00, 0.0
Magnetic : 1.00, 0.00, -10.0C
Gyro : 0.00, 0.00, 0.00

Gravity Sensor is support

Gravity - X : -0.0 / Y : 9.8 / Z : -0.0

Clicking the Landscape button on the right side of the screen on the Control Panel displays 'X : 9.8 / Y : 0.0 / Z : 0.0' on the second Label widget of the app screen.



Acceleration : 1.00, 0.00, 0.0
Magnetic : 0.00, -1.00, -10.0
Gyro : 0.00, 0.00, 0.00

Gravity Sensor is support
Gravity - X : -9.8 / Y : 0.0 / Z : 0.0

3) Related APIs

int sensor_is_supported(sensor_type_e type, bool *supported): an API that determines if a certain sensor is supported. Passing SENSOR_GRAVITY to the first parameter makes the second parameter return whether or not the gravity sensor is supported.

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor): an API that returns the sensor object. Passing SENSOR_GRAVITY to the first parameter returns the gravity sensor object to the second parameter.

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener): an API that creates an event listener. Passing the sensor object to the first parameter returns the listener object to the second parameter.

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data): an API that specifies a callback function to the listener. / parameters: event listener, time interval (in milliseconds), name of the callback function, and user data

int sensor_listener_start(sensor_listener_h listener): an API that starts the listener.

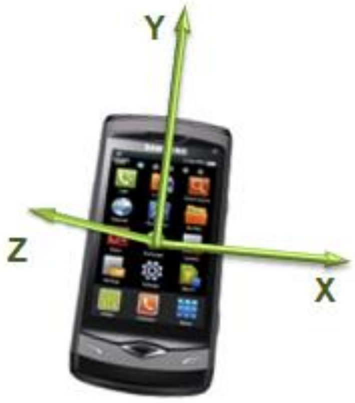
53. Using an Orientation Sensor

The orientation sensor can measure three types of directions.

- Azimuth is a compass sensor. This shows the difference of the angle from the Arctic when the phone is laid on the floor.
- Pitch shows the angle on the Z-axis when the phone is standing. It usually works as a handle in airplane or car games.
- Roll shows the angle on the Y-axis when the phone is laid in Landscape mode. It usually works to control speed in airplane or car games.

When the phone is in Portrait mode, the horizontal direction will be the X-axis, and the vertical direction will be the Y-axis. The forward and backward directions will be the Z-axis. Use the Control Panel to test the orientation sensor on the emulator.

Figure: Axis of the device



1) Determining if the Orientation Sensor is Supported

Create a new source project and specify the project name as 'SensorOrientation.' After the source project is created, open the source file (~.c) under the src folder and add library header files and variables.

```
#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 is a library header file for various sensors.

We are going to display if the orientation sensor is supported on label0, and the current gravity value on label1.

Create two functions on top of the create_base_gui() function.

```
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_EXPAN
D);
          /* 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() is a function that determines if the orientation sensor is supported and displays the result on the first Label widget.

sensor_is_supported(sensor_type_e, bool *) is an API that determines if a certain sensor is supported. Passing SENSOR_ORIENTATION to the first parameter makes the second parameter return whether or not the sensor is supported.

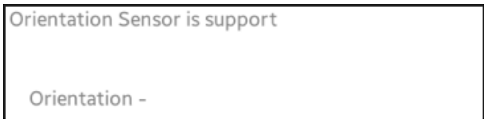
my_box_pack() is a function that adds a widget to a Box.

We need to call the show_is_supported() function when the app is run. Call the function above at the end of the create_base_gui() function.

```
    /* 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_EX
PAND);
        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);
}
```

We created a Box container and two Label widgets. Then, we called the function to determine if the sensor is supported.

Build and run the example. If the orientation sensor is supported, you will see the message 'Orientation Sensor is supported.' Some smartphones may not support the sensor. In such cases, you need to test it on the emulator.



Orientation Sensor is support

Orientation -

2) Requesting the Orientation Sensor Event

We will implement a feature that displays the value on the screen by getting the event when the device direction changes. Add a sensor-related structure and global variables at the top of the source file.

```
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 is the structure including a sensor object and event listener variables.

sensor_info is a global variable of the sensorinfo structure.

Requesting a sensor event starts the listener. We are now going to call an orientation sensor event by using the sensor object and event listener. Create two functions on top of the create_base_gui() function.

```
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_data->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_listener);
    sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sensor_value, ad);
    sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() is an event callback function for the orientation sensor. Output new sensor values on the screen.

The sensor data is passed to the second parameter as an array. The direction data of Azimuth, Pitch, and Roll data is saved in values[0], vales[1], and values[2], respectively.

start_orientation_sensor() is a function that starts the orientation sensor and specifies the event callback function.

sensor_get_default_sensor(sensor_type_e, sensor_h *) is an API that returns the sensor object. Passing SENSOR_ORIENTATION to the first parameter returns the orientation sensor object to the second parameter.

sensor_create_listener(sensor_h, sensor_listener_h *) is an API that creates an event listener. Passing the sensor object to the first parameter returns the listener object to the second parameter.

sensor_listener_set_event_cb(sensor_listener_h, unsigned int, sensor_event_cb, void *) is an API that specifies a callback function to the listener. The parameters listed in order are an event listener, time interval (in milliseconds), name of the callback function, and user data.

sensor_listener_start(sensor_listener_h) is an API that starts the listener.

We will make the event listener operate automatically when the app is run. Call the function above at the end of the create_base_gui() function.

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

    show_is_supported(ad);
    start_orientation_sensor(ad);
}
```

When you run the test in this status, the phone may change to Landscape mode when rotating the phone. Fix the screen direction to Portrait mode. At the top of the create_base_gui() function, make the code below support four directions.

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

Modify the code as follows.

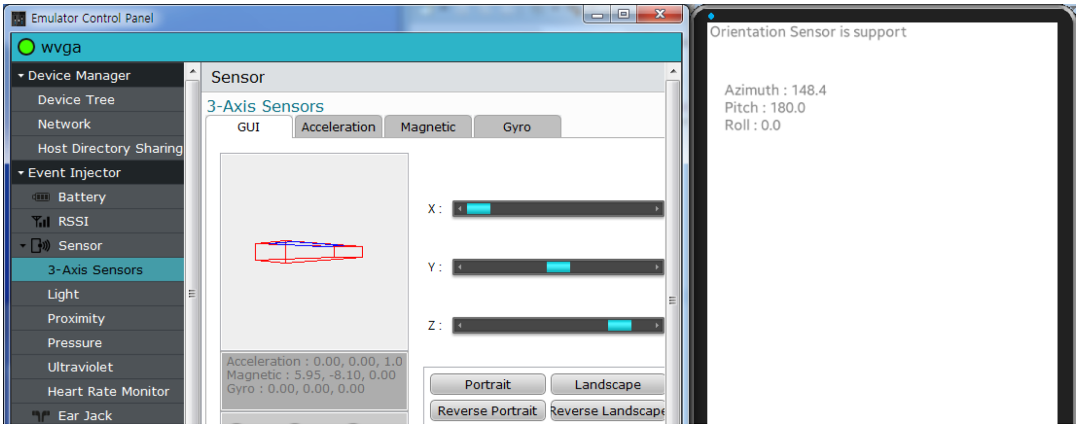
```
int rots[1] = { 0 };
elm_win_wm_rotation_available_rotations_set(ad->win, (const int *)(&rots), 1);
```

Run the example again. On a smartphone, simply rotate the phone for testing. On an emulator, use the Control Panel for testing.

Right-click the emulator and select Control Panel from the shortcut menu.

On the Control Panel, select [Event Injector > 3-Axis Sensors] from the list in tree structure on the left, and then click GUI from the tab buttons on the right side of the screen.

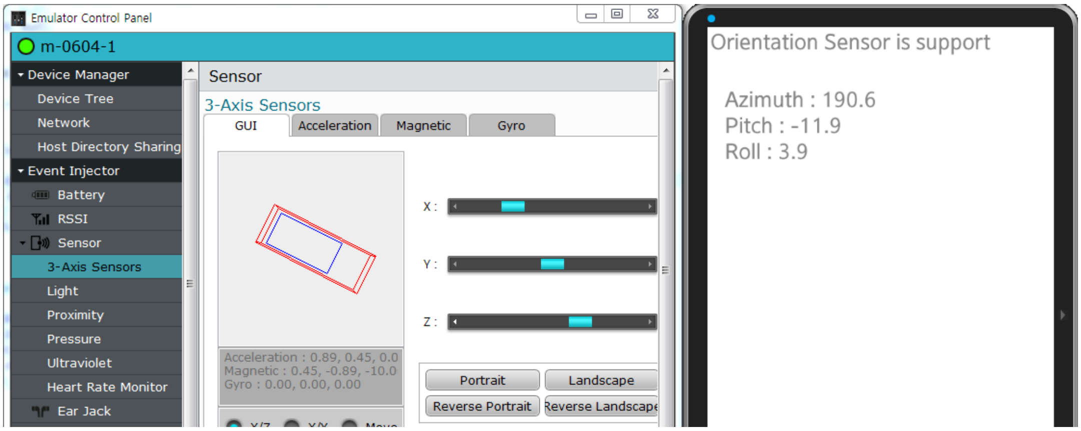
First, let's test Azimuth. Click the Portrait button, and move the slider of X-axis to the left end. Then, the phone will be laid flat. Now, drag the slider of Z-axis to the right and left. On the app screen, the values of Azimuth will change from 0 to 360.



```
Acceleration : 0.00, 0.00, 1.0
Magnetic : 5.95, -8.10, 0.00
Gyro : 0.00, 0.00, 0.00
```

```
Orientation Sensor is support

Azimuth : 148.4
Pitch : 180.0
Roll : 0.0
```

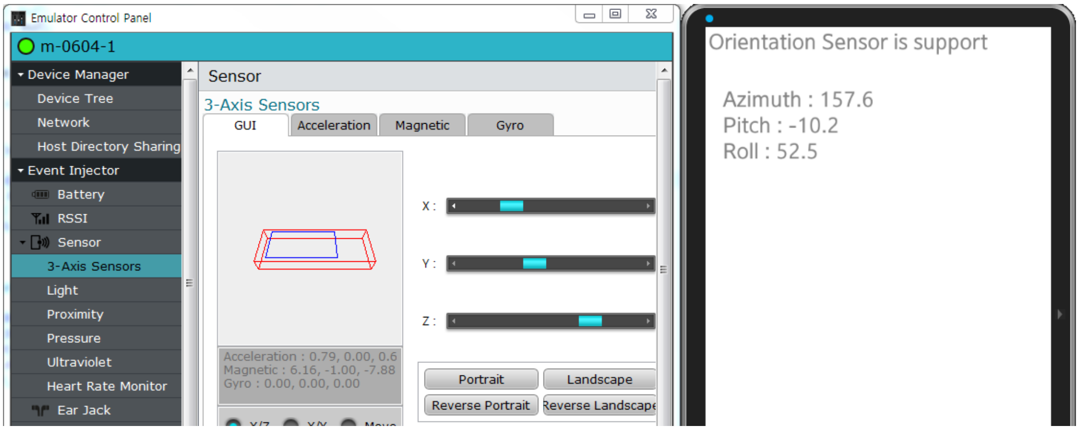
Second, let's test Pitch. Click the Portrait button, and drag the slider of Z-axis to the right and left. On the app screen, the values of Azimuth will change from -180 to 180.



```
Acceleration : 0.89, 0.45, 0.0
Magnetic : 0.45, -0.89, -10.0
Gyro : 0.00, 0.00, 0.00
```

```
Orientation Sensor is support

Azimuth : 190.6
Pitch : -11.9
Roll : 3.9
```

Third, let’s test Roll. Tap the Landscape button, and drag the slider of Y-axis to the right and left. On the app screen, the values of Roll will change from -180 to 180.



```
Acceleration : 0.79, 0.00, 0.6
Magnetic : 6.16, -1.00, -7.88
Gyro : 0.00, 0.00, 0.00
```

```
Orientation Sensor is support

Azimuth : 157.6
Pitch : -10.2
Roll : 52.5
```

3) Related APIs

int sensor_is_supported(sensor_type_e type, bool *supported): an API that determines if a certain sensor is supported. Passing SENSOR_ORIENTATION to the first parameter makes the second parameter return whether or not the orientation sensor is supported.

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor): an API that returns the sensor object. Passing SENSOR_ORIENTATION to the first parameter returns the orientation sensor object to the second parameter.

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener): an API that creates an event listener. Passing the sensor object to the first parameter returns the listener object to the second parameter.

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data): an API that specifies a callback function to the listener. / parameters: event listener, time interval (in milliseconds), name of the callback function, and user data.

int sensor_listener_start(sensor_listener_h listener): an API that starts the listener.

54. Using a Magnetic Sensor

You can use a magnetic sensor when developing a compass app or measuring the magnetic field strength. It can be measured from the strength of the X-, Y-, and Z-axes. When the phone is in Portrait mode, the horizontal direction will be the X-axis, and the vertical direction will be the Y-axis. The forward and backward directions will be the Z-axis. Use the Control Panel to test the magnetic sensor on the emulator.

1) Determining If the Magnetic Sensor is Supported

Create a new source project and specify the project name as 'SensorMagnetic.' After the source project is created, open the source file (~.c) under the src folder and add library header files and variables.

```
#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 is a library header file for various sensors.

math.h is a math library header file.

We are going to display if the magnetic sensor is supported on label0, and the magnetic values of the three axes on label1. We are going to display the value of the whole magnetic field on label2.

Create two functions on top of the create_base_gui() function.

```
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 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_EXPAN
D);
         /* 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() is a function that determines if the magnetic sensor is supported and displays the result on the first Label widget.

sensor_is_supported(sensor_type_e, bool *) is an API that determines if a certain sensor is supported. Passing SENSOR_MAGNETIC to the first parameter makes the second parameter return whether or not the sensor is supported.

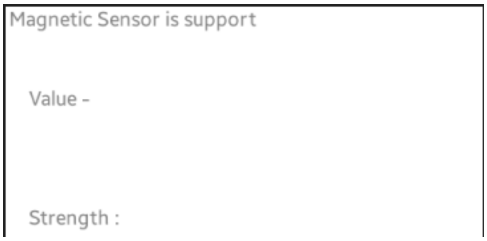
my_box_pack() is a function that adds a widget to a Box.

We need to call the function above when the app is run. Call the function above at the end of the create_base_gui() function.

```
    /* 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_EX
PAND);
        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);
}
```

Three Label widgets have been created. Then, the function is called to determine if the sensor is supported.

Build and run the example. If the magnetic sensor is supported, you will see the message 'Magnetic Sensor is supported.' Some smartphones may not support the sensor. In such cases, you need to test it on the emulator.



Magnetic Sensor is support

Value -

Strength :

2) Requesting the Magnetic Sensor Event

We will implement a feature that displays the value on the screen by requesting an event when the magnetic field around changes. Add a sensor-related structure and global variables at the top of the source file.

```
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 is the structure including a sensor object and event listener variables.

sensor_info is a global variable of the sensorinfo structure.

Requesting a sensor event starts the listener. We are now going to request a magnetic sensor event by using the sensor object and event listener. Create two functions on top of the create_base_gui() function.

```
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_data->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_listener);
    sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sensor_value, ad);
    sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() is an event callback function for the magnetic sensor. Output new sensor values on the screen.

The sensor data is passed to the second parameter as an array. The direction data of the X-, Y-, and Z-axes is saved in values[0], vales[1], and values[2], respectively.

start_magnetic_sensor() is a function that starts the magnetic sensor and specifies the event callback function.

sensor_get_default_sensor(sensor_type_e, sensor_h *) is an API that returns the sensor object. Passing SENSOR_MAGNETIC to the first parameter returns the magnetic sensor object to the second parameter.

sensor_create_listener(sensor_h, sensor_listener_h *) is an API that creates an event listener. Passing the sensor object to the first parameter returns the listener object to the second parameter.

sensor_listener_set_event_cb(sensor_listener_h, unsigned int, sensor_event_cb, void *) is an API that specifies a callback function to the listener. The parameters listed in order are an event listener, time interval (in milliseconds), name of the callback function, and user data.

sensor_listener_start(sensor_listener_h) is an API that starts the listener.

We will make the event listener operate automatically when the app is run. Call the function above at the end of the create_base_gui() function.

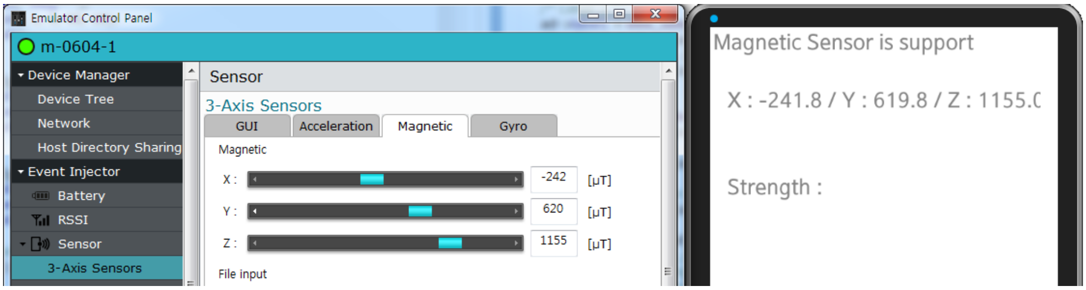
```
   /* Show window after base gui is set up */
   evas_object_show(ad->win);

   show_is_supported(ad);
   start_magnetic_sensor(ad);
}
```

Run the example again. On a smartphone, simply bring the phone near to a magnet, or rotate it while laying it on the floor for testing. On an emulator, use the Control Panel for testing.

Right-click the emulator and select Control Panel from the shortcut menu.

On the Control Panel, select [Event Injector > 3-Axis Sensors] from the list in tree structure on the left, and then click Magnetic from the tab buttons on the right side of the screen.

Drag the three sliders on the right screen of the Control Panel. The values of the second Label widget on the app screen will change.



X : -242 [μT]
Y : 620 [μT]
Z : 1155 [μT]

Magnetic Sensor is support

X : -241.8 / Y : 619.8 / Z : 1155.(

Strength :

3) Getting the Whole Magnetic Values

Now, we will learn how to get the whole magnetic values by integrating the values of the X-, Y-, and Z-axes. To get the whole magnetic values, add up the squares of the three values, and then the square roots. Add a new function on top of the _new_sensor_value() function, and modify the code of the _new_sensor_value() function.

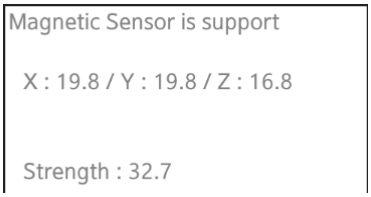
```
static float
```

```
_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_data->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 *) is a function that sums the squares of three values saved in the array, and returns the square roots of the result.

sqrt(double) is a math API to calculate a square root.

Run the example again. Now, the result of calculating the magnetic values will be displayed on the third Label.



Magnetic Sensor is support

X : 19.8 / Y : 19.8 / Z : 16.8

Strength : 32.7

4) Related APIs

int sensor_is_supported(sensor_type_e type, bool *supported): an API that determines if a certain sensor is supported. Passing SENSOR_MAGNETIC to the first parameter makes the second parameter return whether or not the magnetic sensor is supported.

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor): an API that returns the sensor object. Passing SENSOR_MAGNETIC to the first parameter returns the magnetic sensor object to the second parameter.

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener): an API that creates an event listener. Passing the sensor object to the first parameter returns the listener object to the second parameter.

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data): an API that specifies a callback function to the listener. / parameters: event listener, time interval (in milliseconds), name of the callback function, and user data.

int sensor_listener_start(sensor_listener_h listener): an API that starts the listener.

sqrt(double): A math API that calculates a square root.

55. Using a Proximity Sensor

When your phone rings and you bring the phone close to your face, the phone screen turns off. This feature is enabled by the proximity sensor. Use the Control Panel to test the proximity sensor on the emulator.

1) Determining if the Proximity Sensor is Supported

Create a new source project and specify the project name as 'SensorProximity.' After the source project is created, open the source file (~.c) under the src folder and add library header files and variables.

```
#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 is a library header file for various sensors.

We are going to display if the proximity sensor is supported on label0, and the distance value on label1.

Create two functions on top of the create_base_gui() function.

```
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 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_EXPAN
D);
           /* 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() is a function that determines if the proximity sensor is supported and displays the result on the first Label widget.

sensor_is_supported(sensor_type_e, bool *) is an API that determines if a certain sensor is supported. Passing SENSOR_PROXIMITY to the first parameter makes the second parameter return whether or not the sensor is supported.

my_box_pack() is a function that adds a widget to a Box container.

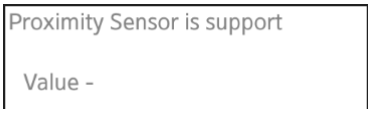
We need to call the show_is_supported() function when the app is run. Call the function above at the end of the create_base_gui() function.

```
    /* 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_EX
PAND);
        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);
}
```

We created a Box and two Label widgets. Then, we called the function to determine if the sensor is supported.

Build and run the example. If the proximity sensor is supported, you will see the message 'Proximity Sensor is supported.' Some smartphones may not support the sensor. In such cases, you need to test it on the emulator.



Proximity Sensor is support

Value -

2) Requesting a Proximity Sensor Event

We will implement a feature that displays the distance value on the screen by getting the event when the proximity sensor detects an object. Add a sensor-related structure and global variables at the top of the source file.

```
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 is the structure including a sensor object and event listener variables.

sensor_info is a global variable of the sensorinfo structure.

Requesting a sensor event starts the listener. We are now going to request a proximity sensor event by using the sensor object and event listener. Create two functions on top of the create_base_gui() function.

```
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_listener);
    sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sensor_value, ad);
    sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() is an event callback function for the proximity sensor. Output new sensor values on the screen.

The sensor data will be passed to the second parameter. The distance data is saved in values[0].

start_proximity_sensor() is a function that starts the proximity sensor and specifies the event callback function.

sensor_get_default_sensor(sensor_type_e, sensor_h *) is an API that returns the sensor object. Passing SENSOR_PROXIMITY to the first parameter returns the proximity sensor object to the second parameter.

sensor_create_listener(sensor_h, sensor_listener_h *) is an API that creates an event listener. Passing the sensor object to the first parameter returns the listener object to the second parameter.

sensor_listener_set_event_cb(sensor_listener_h, unsigned int, sensor_event_cb, void *) is an API that specifies a callback function to the listener. The parameters listed in order are an event listener, time interval (in milliseconds), name of the callback function, and user data.

sensor_listener_start(sensor_listener_h) is an API that starts the listener.

We will make the event listener operate automatically when the app is run. Call the function above at the end of the create_base_gui() function.

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

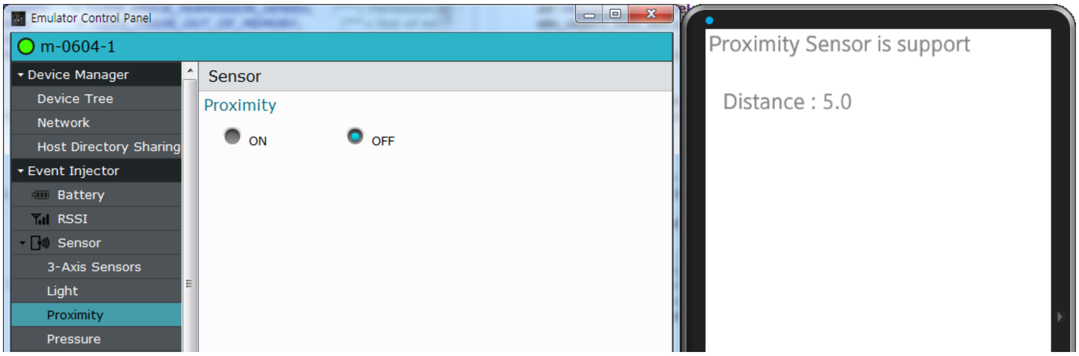
```
    show_is_supported(ad);
    start_proximity_sensor(ad);
}
```

Run the example again. On a smartphone, simply bring the phone close to your face for testing. On an emulator, use the Control Panel.

Right-click the emulator and select Control Panel from the shortcut menu.

On the Control Panel, select [Event Injector > Proximity] from the list in tree structure on the left.

Clicking the ON button on the right side of the screen on the Control Panel displays 0.0 on the second Label widget of the app screen. Clicking the OFF button displays 5.0 on the Label widget.



Proximity Sensor is support

Distance : 5.0

3) Related APIs

int sensor_is_supported(sensor_type_e type, bool *supported): an API that determines if a certain sensor is supported. Passing SENSOR_PROXIMITY to the first parameter makes the second parameter return whether or not the proximity sensor is supported.

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor): an API that returns the sensor object. Passing SENSOR_PROXIMITY to the first parameter returns the proximity sensor object to the second parameter.

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener): an API that creates an event listener. Passing the sensor object to the first parameter returns the listener object to the second parameter.

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data): an API that specifies a callback function to the listener. / parameters: event listener, time interval (in milliseconds), name of the callback function, and user data.

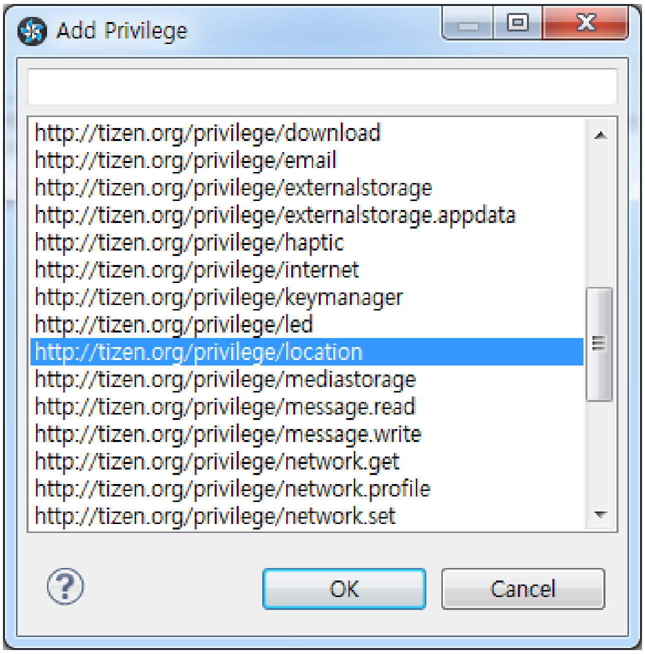
int sensor_listener_start(sensor_listener_h listener): an API that starts the listener.

56. Using a GPS Sensor

You need to know the geographic coordinates of locations to successfully use a map or navigation app. The coordinates are also essential for location-based services of SNS including Facebook and Twitter, as well as other informative apps. In this example, we will learn how to use a GPS sensor using Location Manager.

1) Registering a Privilege

Create a new source project and specify the project name as 'SensorGps.' You need to have the applicable user privileges to use Location Manager. After the source project is created, open the tizen-manifest.xml file, and click Privileges among the tab buttons below. Then, click the Add button in the upper right corner. When the popup window appears, select http://tizen.org/privilege/location from the list, and click the OK button to close the window.



http://tizen.org/privilege/download
http://tizen.org/privilege/email
http://tizen.org/privilege/externalstorage
http://tizen.org/privilege/externalstorage.appdata
http://tizen.org/privilege/haptic
http://tizen.org/privilege/internet
http://tizen.org/privilege/keymanager
http://tizen.org/privilege/led
http://tizen.org/privilege/location
http://tizen.org/privilege/mediastorage
http://tizen.org/privilege/message.read
http://tizen.org/privilege/message.write
http://tizen.org/privilege/network.get
http://tizen.org/privilege/network.profile
http://tizen.org/privilege/network.set

After saving, click the tizen-manifest.xml button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.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) Checking the Status of Location Manager

Now, we need to determine if Location Manager is available. Open the source file (~.c) under the src folder and add library header files and variables.

```
#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;
```

location.h is a library header file for Location Manager.

We are going to display the status on label0, and location information on label1.

location_manager_h is the Location Manager structure.

Create two functions on top of the create_base_gui() function.

```
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" : "Disable";
    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, a
d);
    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_EXPAN
D);
         /* 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() is an event callback function that changes the status of Location Manager. Now, the status values will be passed to the first parameter. The status types are as follows:

- LOCATIONS_SERVICE_DISABLED: Service is unavailable.
- LOCATIONS_SERVICE_ENABLED: Service is available.

show_state() is a function that requests an event to change the status.

location_manager_create(location_method_e, location_manager_h*) is an API that creates the Location Manager object. Passing LOCATIONS_METHOD_GPS to the first parameter makes the second parameter return the Location Manager object. The types of information collection for locations are as follows:

- LOCATIONS_METHOD_GPS : Uses GPS.
- LOCATIONS_METHOD_WPS : Uses Wi-Fi.
- LOCATIONS_METHOD_HYBRID : Automatically selects between GPS and Wi-Fi.

location_manager_set_service_state_changed_cb(location_manager_h, location_service_state_changed_cb, void *) is an API that specifies the name of the event function for Location Manager status change. The parameters listed in order are the Location Manager object, name of the event callback function, and user data.

location_manager_start(location_manager_h) is an API that operates Location Manager.

my_box_pack() is a function that adds a widget to a Box.

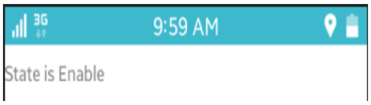
We will make the Location Manager operate automatically when the app is run. Add new code to the create_base_gui() function.

```
    /* 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_EX
PAND);
        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);
}
```

Create a Box and Label widget, and call the event request function to change the status of Location Manager.

Build and run the example. If the Label widget displays 'State is Enable,' the GPS is working normally.



State is Enable

3) Requesting Coordinates of the Current Location

We will implement a feature that displays the coordinates of the current location on the screen when you tap the button. Add new code to the create_base_gui() function.

```
        { /* 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);
      }
   }
```

We added one Label widget and one Button widget.

We are now going to create a button callback function. Add new code on top of the create_base_gui() function.

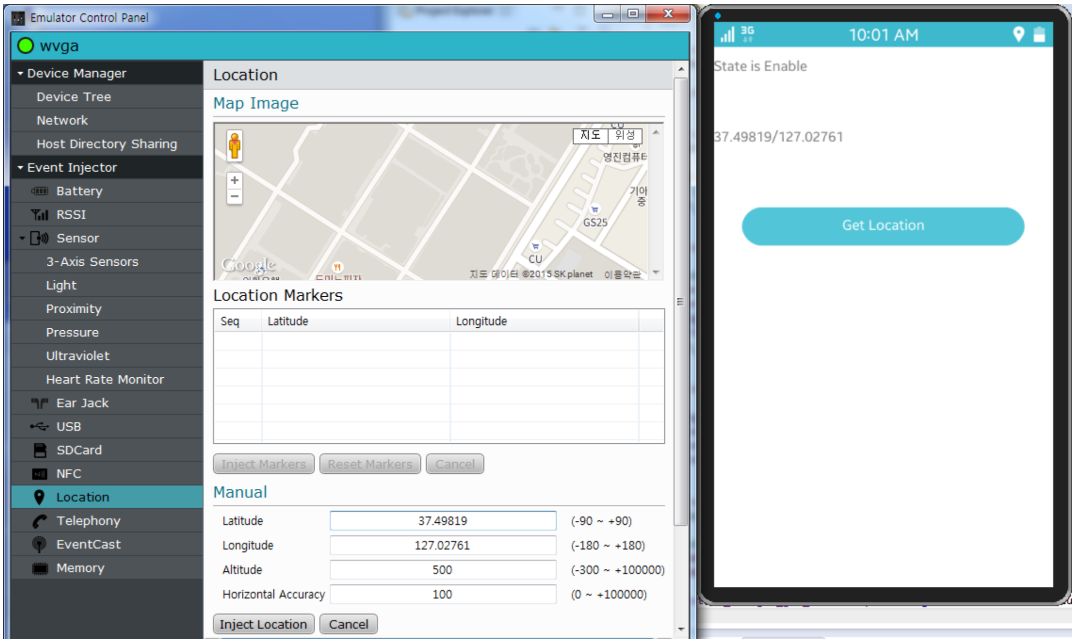
```
static void
btn_clicked_cb(void *data, Evas_Object *obj, void *event_info)
{
   appdata_s *ad = data;
   double altitude, latitude, longitude, climb, direction, speed;
   double horizontal, vertical;location_accuracy_level_e level;time_t timestamp;

   location_manager_get_location(ad->manager, &altitude, &latitude, &longitude,
                                 &climb, &direction, &speed, &level, &horizontal,
&vertical, &timestamp);
   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*) is an API that requests the information of the current location. The parameters listed in order are the object of Location Manager, altitude, latitude, longitude, vertical movement speed, direction, horizontal movement speed, accuracy, horizontal accuracy (in meters), vertical accuracy (in meters), and time.

The next code outputs the coordinates of latitude and longitude on the second Label widget.

Run the example again. Use the Control Panel to test it on the emulator. Right-click the emulator and select Control Panel from the shortcut menu.

On the Control Panel, select [Event Injector > Location] from the list in tree structure on the left. Then, input a latitude coordinate in Latitude on the right of the screen (e.g. 37.49819), and a longitude coordinate in Longitude (e.g. 127.02761). Input 500 for Altitude, and 100 for Horizontal Accuracy. Then, click the Inject Location button. When you tap the button on the app screen, the coordinates of latitude and longitude that you entered from the Control Panel will be displayed in the second Label widget.



Latitude 37.49819 (-90 ~ +90)
Longitude 127.02761 (-180 ~ +180)
Altitude 500 (-300 ~ +100000)
Horizontal Accuracy 100 (0 ~ +100000)

State is Enable

37.49819/127.02761

4) Requesting a Location Movement Event

We will show you how to make the new coordinates of latitude and longitude display automatically on the screen when the user's locations changes. Add a new line of code at the end of the create_base_gui() function.

```
    /* 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 *) is an API that requests an event to change the location information. The parameters listed in order are the Location Manager object, name of the event callback function, time interval, and user data.

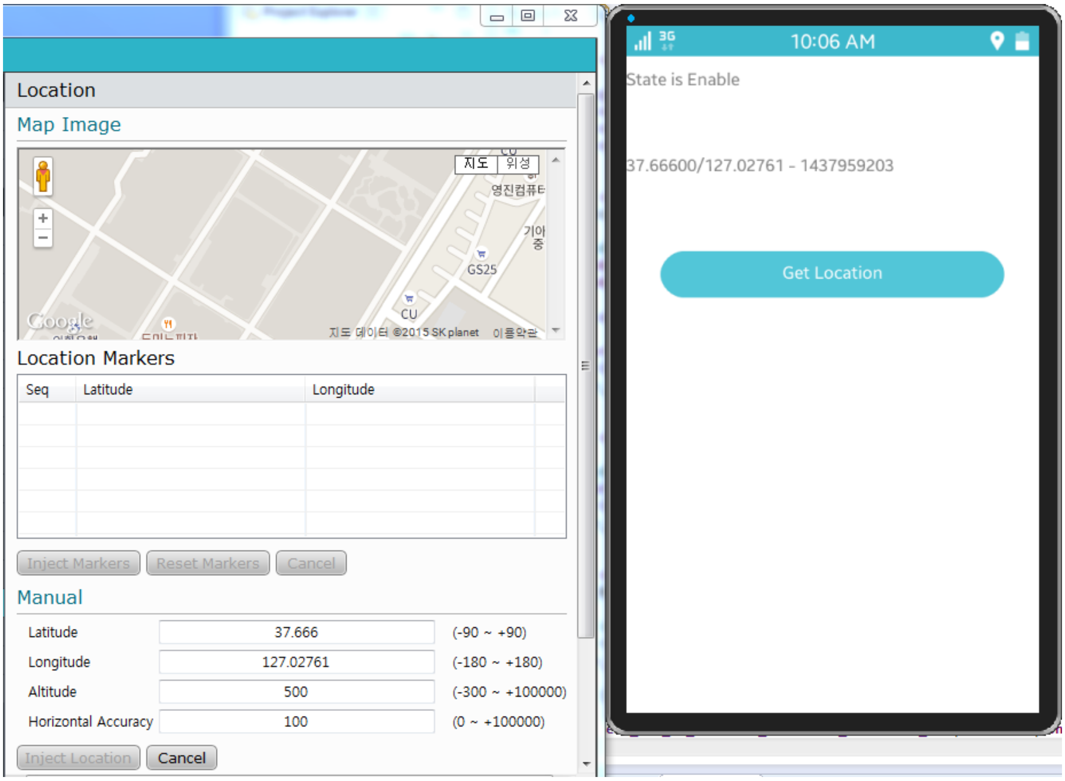
Finally, we are going to create a callback function. Add a new function on top of the create_base_gui() function.

```
static void position_updated_cb(double latitude, double longitude, double altitude, ti
me_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() is a callback function that is called when a new location information is received. The parameters listed in order are latitudes, longitudes, altitudes, time, and user data.

The coordinates of latitude and longitude, as well as the time are output on the screen within the function.

Run the example again. Change the values of latitude (e.g. 37.666) and longitude (e.g. 127.02761), and click the Inject Location button. The coordinates of latitude and longitude will automatically change on the app screen.



Location Markers

Seq	Latitude	Longitude

Manual

Latitude	37.666	(-90 ~ +90)
Longitude	127.02761	(-180 ~ +180)
Altitude	500	(-300 ~ +100000)
Horizontal Accuracy	100	(0 ~ +100000)

State is Enable

37.66600/127.02761 - 1437959203

5) Related APIs

int location_manager_create(location_method_e method, location_manager_h* manager): an API that creates the Location Manager object. Passing LOCATIONS_METHOD_GPS to the first parameter makes the second parameter return the Location Manager object. The types of information collection for locations are as follows:

- LOCATIONS_METHOD_GPS : Uses GPS.
- LOCATIONS_METHOD_WPS : Uses Wi-Fi.
- LOCATIONS_METHOD_HYBRID : Automatically selects between GPS and Wi-Fi.

int location_manager_set_service_state_changed_cb(location_manager_h manager, location_service_state_changed_cb callback, void *user_data): an API that specifies the name of the event function to change the status of Location Manager. The parameters listed in order are the Location Manager object, name of the event callback function, and user data.

int location_manager_start(location_manager_h manager): an API that operates Location Manager

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): an API that requests the information of the current location. The parameters listed in order are the object of Location Manager, altitude, latitude, longitude, vertical movement speed, direction, horizontal movement speed, accuracy, horizontal accuracy (in meters), vertical accuracy (in meters), and time.

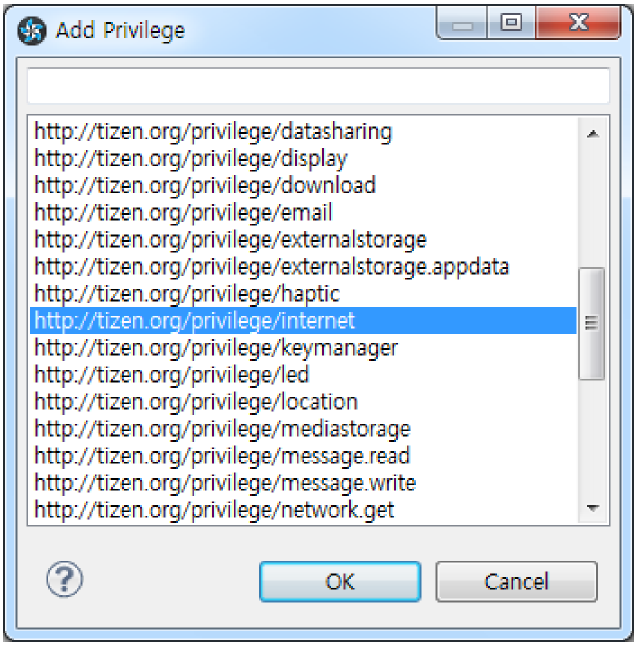
int location_manager_set_position_updated_cb(location_manager_h
manager, location_position_updated_cb callback, int interval, void
*user_data): an API that requests the event of location information
change. The parameters listed in order are the Location Manager object,
name of the event callback function, time interval, and user data.

57. Google Maps Library

You can use Google Maps for an electronic map. Pass the coordinates of latitude and longitude, as well as the values of the Zoom level to the Google Maps server, and then enter the map data received from the server to the canvas. Hard coding the example can be rather time-consuming, so we will call the library file to implement it in an easier way.

1) Registering a Privilege

Create a new source project and specify the project name as 'MapViewEx.' You need to have the applicable user privileges to communicate with the Google Maps server. After the source project is created, open the tizen-manifest.xml file, and click Privileges among the tab buttons below. Then, click the Add button in the upper right corner. When the popup window appears, select http://tizen.org/privilege/internet from the list, and click the OK button to close the window.



http://tizen.org/privilege/datasharing
http://tizen.org/privilege/display
http://tizen.org/privilege/download
http://tizen.org/privilege/email
http://tizen.org/privilege/externalstorage
http://tizen.org/privilege/externalstorage.appdata
http://tizen.org/privilege/haptic
http://tizen.org/privilege/internet
http://tizen.org/privilege/keymanager
http://tizen.org/privilege/led
http://tizen.org/privilege/location
http://tizen.org/privilege/mediastorage
http://tizen.org/privilege/message.read
http://tizen.org/privilege/message.write
http://tizen.org/privilege/network.get

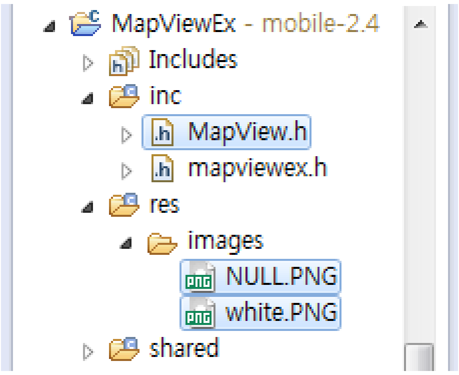
After saving, click the tizen-manifest.xml button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.exam
ple.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) Copying Library Files

Realizing the features of an electronic map through hard coding is rather time-consuming, so we will simply copy the library file for use. Copy the MapView.h file in the /etc folder of the appendix to the /inc folder of the source project.

Image files are also used in the map source code. Create a new folder under the /res folder of the source project and specify the folder's name as 'images.' Then, copy two image files (NULL.PNG and white.PNG) from the /image folder of the appendix to the folder that you have just created.



3) Creating a MapView Widget

We have copied all library and image files we need. We are now going to create a MapView widget using the source code. Open the source file (~.c) under the /src folder and add the library and the coordinates of latitude and longitude.

```
#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 is a MapView library file that we have just copied from the appendix.

START_LATITUDE and START_LONGITUDE are the coordinates of latitude and longitude for Central Park in Manhattan, New York City. You can choose any coordinates you wish.

Add new code to the create_base_gui() function. Annotate the Label widget.

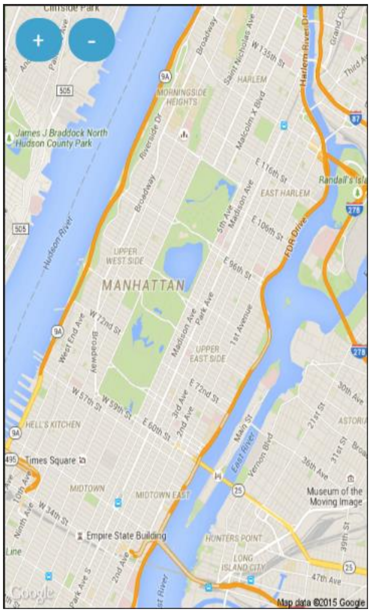
```
    /* Label*/
    /*ad->label = elm_label_add(ad->conform);
    elm_object_text_set(ad->label, "Hello EFL");
    evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_object_content_set(ad->conform, ad->label);
    evas_object_show(ad->label);*/

    create_map(ad->conform, START_LATITUDE, START_LONGITUDE);
```

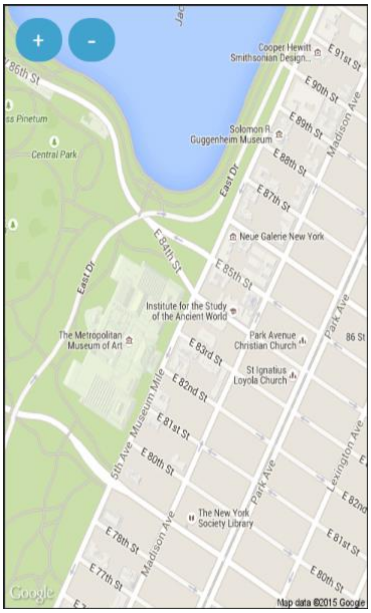
```
   /* Show window after base gui is set up */
   evas_object_show(ad->win);
```

create_map(Evas_Object *, double, double) is a function that creates a MapView widget. The parameters listed in order are a container, and the coordinates of latitude and longitude.

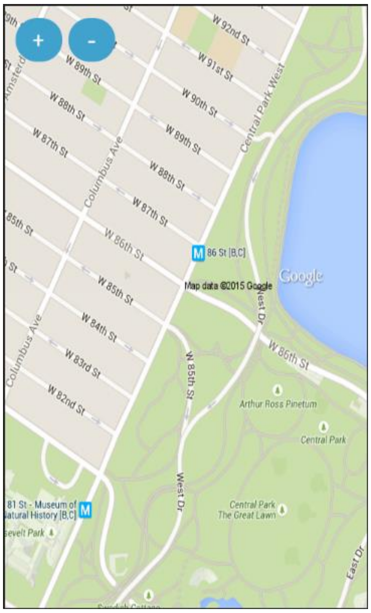
Build and run the example. If you have any errors while building the example, open the /inc/MapView.h file, and correct the #include <curl/curl.h> code as follows:
#include <curl.h>

The electronic map is displayed, and you will see two buttons. When you tap the '+' button, the map will be zoomed in, while tapping the '-' button will zoom out the map. You can also drag the map in any direction you want.





+
-
Cooper Hewitt Smithsonian Design...
Solomon R. Guggenheim Museum
Central Park
Neue Galerie New York
Institute for the Study of the Ancient World
The Metropolitan Museum of Art
Park Avenue Christian Church
St Ignatius Loyola Church
The New York Society Library
Museum Mile
Madison Ave
Park Ave
Lexington Ave
Map data ©2015 Google



4) Source Code of MapView Widget

Now, we will learn about the source code used in the MapView library. Open the MapView.h file that you copied to the /inc folder, and scroll to the bottom. At the end of create_map(), you will find the code as follows.

int mkdir(char *__path, __mode_t __mode): an API that creates a new folder.

make_new_url() is a function that creates a URL path that requests map data to the Google Maps server using the coordinates of latitude and longitude. The created URL address will be saved in global variables such as place_api_url or curr_url.

arrange_start_main_page() is a function that passes the coordinates of latitude and longitude to the Google Map server.

map_dload_thread() is an event function for the map data download thread.

save_map_temp_file() is a function that saves map data received from the server as a file.

update_main_page() is a function that saves map data into a tile, and changes the location and size of the tile.

mouse_down_cb() is a function that handles the event when the user Touch Downs the map tile.

mouse_move_cb() is a function that handles the event when the user Touch Moves the map tile.

mouse_up_cb() is a function that handles the event when the user Touch Cancels the map tile.

map_zoom_in() is a callback function for the '+' button. It increases the Zoom level, and changes the center coordinate. Then, it calls the arrange_start_main_page() function.

map_zoom_out() is a callback function for the '-' button. It decreases the Zoom level, and changes the center coordinate. Then, it calls the arrange_start_main_page() function.

5) Source Code of MapView.h

If you cannot download the appendix file, create a text file and name the file as 'MapView.h.' Then, copy the code below.

Among the define statement, for <Google Map Key> at the end of START_URL and PLACE_API_URL, enter the Google Maps key that you own.

```
/*
 * 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_RESO
LUTION}},
        {{0,Y_RESOLUTION},  {0,0},  {0,-Y_RESOLUTION}},
        {{X_RESOLUTION,Y_RESOLUTION},  {X_RESOLUTION,0},  {X_RESOLUTION,-Y_RESOLU
TION}}
        };
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=", "&", "&", "&markers=", "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_lati * 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_chan
ged)
{
    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_zoom_level++;
                else return URL_NOT_CHANGED;
            }
            else if(zoom<0)
            {
                if(current_zoom_level>MIN_ZOOM_SCALE) current_zoom_level--;
                else return URL_NOT_CHANGED;
            }
            snprintf(tmp_str_changed, sizeof(tmp_str_changed), "zoom=%d", current_zoo
m_level);
        }
        break;

        case CENTER_NAME_CHANGED:
        {
            snprintf(tmp_str_changed, sizeof(tmp_str_changed), "center=%s", (char*)data);
        }
        break;

        case CENTER_GEOMETRY_CHANGED:
        {
            pos_comma = strstr(pos, ","); // to search first ',' from after "center=..."

            memcpy(str_lati, (char*)((int)pos+strlen("center=")), (int)pos_comma - ((int)pos
+strlen("center=")));
            memcpy(str_longi, (char*)((int)pos_comma+1), ((int)pos_and_oper-((int)pos_co
mma+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), "center=%f,%f", latitude, l
ongitude);
        }
        break;

        case ADD_NEW_MARKER:
        {
            char tmp_char = '%';
            snprintf(tmp_str_changed, sizeof(tmp_str_changed), "&markers=color:blue%c7
Clabel: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_changed_1), "%s&markers=colo
r:red%c7Clabel:S%c7C%f,%f", tmp_str_changed, tmp_char, tmp_char, place_search_list[i].
str_lati, place_search_list[i].str_longti);
                snprintf(tmp_str_changed, sizeof(tmp_str_changed), "%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), "query=%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_changed));
    memcpy(&new_url[((int)pos-(int)prev)+strlen(tmp_str_changed)], tmp_str_end, 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 longitude)
{
    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, longitude);

    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_LEN);
    memcpy(&map_dload_q[map_dload_q_idx].filename[0], all_tiles[x_info][y_info].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 else 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_LEN);

            if(all_tiles[current_center_x-(1-i)][current_center_y-(1-j)].download_ok)
            {
                memcpy(tile_info[i][j].file_name, all_tiles[current_center_x-(1-i)][current_cente
r_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.PN
G", 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_tiles[current_center.x_info][current_center.y
_info].lati_of_center;
                        double longti_of_center = all_tiles[current_center.x_info][current_cent
er.y_info].longti_of_center;
                        double longti_for_end_check;

                        lati_of_center = next_lati_value(lati_of_center, Y_RESOLUTION*(j-1), c
urrent_zoom_level);
                        longti_of_center = next_longti_value(longti_of_center, X_RESOLUTION
*(-1), current_zoom_level);
                        longti_of_center = next_longti_value(longti_of_center, X_RESOLUTION
*(-1), current_zoom_level);

                        request_map_download(current_center.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.ca
nvas.x);
                    evas_object_move(main_page[i][j], tile_info[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_info[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_tiles[current_center.x_info][current_center.y
_info].lati_of_center;
                        double longti_of_center = all_tiles[current_center.x_info][current_cent
er.y_info].longti_of_center;
                        double longti_for_end_check;

                        lati_of_center = next_lati_value(lati_of_center, Y_RESOLUTION*(j-1), c
urrent_zoom_level);
                        longti_of_center = next_longti_value(longti_of_center, X_RESOLUTION,
 current_zoom_level);
                        longti_of_center = next_longti_value(longti_of_center, X_RESOLUTION,
 current_zoom_level);

                        request_map_download(current_center.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.ca
nvas.x);
                    evas_object_move(main_page[i][j], tile_info[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_RESOLUTIO
N, tile_info[i][j].curr_y);
                        tile_info[i][j].curr_x = tile_info[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_tiles[current_center.x_info][current_center.y
_info].lati_of_center;
                        double longti_of_center = all_tiles[current_center.x_info][current_cent
er.y_info].longti_of_center;

                        lati_of_center = next_lati_value(lati_of_center, Y_RESOLUTION*(-1), cu
rrent_zoom_level);
                        lati_of_center = next_lati_value(lati_of_center, Y_RESOLUTION*(-1), cu
rrent_zoom_level);
                        longti_of_center = next_longti_value(longti_of_center, X_RESOLUTION
*(i-1), current_zoom_level);

                        request_map_download(current_center.x_info-(1-i), current_center.y_inf
o+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_info[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, current_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_info[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_r
equest)
                    {
                        double lati_of_center = all_tiles[current_center.x_info][current_center.y
_info].lati_of_center;
                        double longti_of_center = all_tiles[current_center.x_info][current_cent
er.y_info].longti_of_center;

                        lati_of_center = next_lati_value(lati_of_center, Y_RESOLUTION, curren
t_zoom_level);
                        lati_of_center = next_lati_value(lati_of_center, Y_RESOLUTION, curren
t_zoom_level);
                        longti_of_center = next_longti_value(longti_of_center, X_RESOLUTION
*(i-1), current_zoom_level);

                        request_map_download(current_center.x_info-(1-i), current_center.y_inf
o-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.c
anvas.y);
                    evas_object_move(main_page[i][j], tile_info[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, current_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].cur
r_y+Y_RESOLUTION);
                        tile_info[i][j].curr_y = tile_info[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_RESOLUTION);
            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_CALLBACK_MOUSE_MOV
E, mouse_move_cb, m_md);
            evas_object_event_callback_add(main_page[i][j], EVAS_CALLBACK_MOUSE_DO
WN, mouse_down_cb, m_md);
            evas_object_event_callback_add(main_page[i][j], EVAS_CALLBACK_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].filename[0], MAX_T
MP_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_filename);
           } while(dload_success==0);

           all_tiles[index.x][index.y].download_ok = 1;
           set_map_main_tile_info(current_center.x_info, current_center.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_cente
r);
    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_LEN);
        }
    }

    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), current_center.y_info-(1-j),
                    next_lati_value(map_start_lati_of_center, Y_RESOLUTION*(1-j), current_zo
om_level),
                    next_longti_value(map_start_longti_of_center, X_RESOLUTION*(i-1), curre
nt_zoom_level));
        }
    }
    //download ceter map first for visual effect
    request_map_download(current_center.x_info, current_center.y_info, map_start_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_center, y_moved*(-1),
current_zoom_level-1);
        map_start_longti_of_center = next_longti_value(map_start_longti_of_center, -x_mo
ved, 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_center, 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_NUM]=
{0,};
    double tmp_lati_sum=0, tmp_longti_sum=0;

    chunk.memory = malloc(1);  /* will be grown as needed by the realloc above */
    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, WriteMemoryCallback);

    /* 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 else 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(tmp_str_lati);

            memset(tmp_str_longti, 0, GEO_INFO_STR_NUM);
            memcpy(tmp_str_longti, (char*)(pos_longti_start+5), pos_longti_end-(pos_longt
i_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_longti;
    }

    *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 recognize ' '
        }
        if(make_new_url(PLACE_INFO, 0, buf, 0, 0) == URL_CHANGED)//make url for qu
ery of place info.
        {
            if(!get_geometry_info_of_place(place_api_url, &latitude, &longitude)) // get pl
ace 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); // delete all previous mark
ers
                if((make_new_url(CENTER_NAME_CHANGED, 0, geometry_buf, 0, 0) == UR
L_CHANGED)
                 && (make_new_url(ADD_NEW_MULTI_MARKER, 0, NULL, 0, 0) == URL_C
HANGED))
                {
                    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 timestamp;
    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, &timestamp);

    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 timestamp;

    if(!location_initialized)
        return ECORE_CALLBACK_RENEW;

    location_manager_get_last_location(l_manager, &altitude, &latitude, &longtitude,
                                             &climb, &direction, &speed, &level, &horizontal,
 &vertical, &timestamp);

    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_changed_cb, N
ULL) ;
    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_ */
```

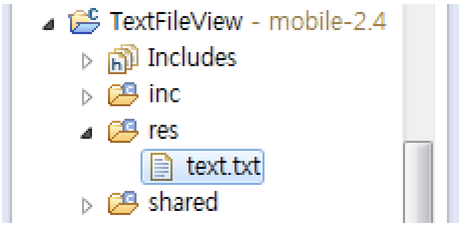
58. Reading and Writing Text Files

If you want to read and write text data in a file, use the FILE structure. You can read files in the /res folder, but cannot write them. The files in the /data folder are available both for reading and writing.

1) Reading Text Files

Now, we are going to read text files in the /res folder and display them on the screen.

Create a new source project and specify the project name as 'TextFileView.' After the source project is created, copy the text.txt file in the /etc folder of the appendix to the /res folder of the source project.



The ext.txt file includes greetings in different languages as follows:

```
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/>
Здра́вствуйте <br/>
```

Then, open the source file (~.c) under the src folder and add variables to the appdata structure,

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

Entry widget variables are now added.

Create two functions on top of the create_base_gui() function.

```
static void
my_table_pack(Evas_Object *table, Evas_Object *child, int col, int row, int spanx, 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_EXPAND);
        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_table_pack() is a function that adds a widget to a Table.

my_button_add() is a function that creates a Button widget.

Then, move to the create_base_gui() function, and add the widget-creating code for Frame, Table, Button, and Entry widgets. Annotate the Label widget-creating code.

```
    /* 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_E
XPAND);
        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_EXP
AND);
        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_HI
NT_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_
HINT_EXPAND, EVAS_HINT_FILL, EVAS_HINT_FILL);
        }
    }

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

Add three new functions on top of the create_base_gui() function.

```
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() is a function that requests the absolute path of a file saved in the /res folder, and returns it.

read_file() is a function that reads the content of a text file and returns it.

fopen(char *, char *) is an API that returns the handle of a file. Pass the file path to the first parameter, and specify the file access mode to the second parameter. "r" indicates read only, while "w" indicates write only.

fseek(FILE *, int, int) is an API that moves to a certain location of Filestream. The first, second, and third parameters are Filestream, the count of moved bytes, and starting point, respectively. SEEK_SET, SEEK_CUR, and SEEK_END indicate the start of a file, current location, and the end of the file, respectively.

ftell(FILE *) is an API that returns the current location of Filestream as the byte count. When it is located at the end, it will return the file size.

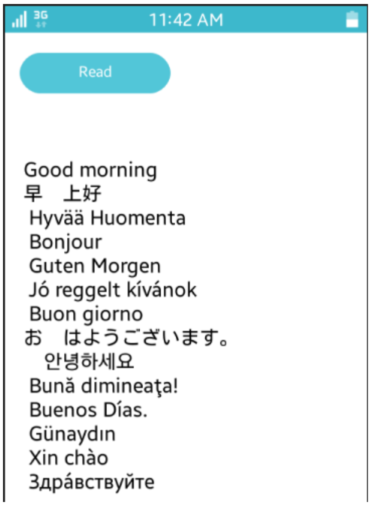
rewind(FILE *) is an API that turns back the location of Filestream to the original location.

fgets(char *, int, FILE *) is an API that reads text data from a file. Passing the maximum length of text to the second parameter and the Filestream to the third parameter makes the first parameter return the string data.

fclose(FILE *) is an API that closes the Filestream.

btn_read_cb() is a Button callback function. Request the text file path, read the content of the file, and enter it in the Entry widget.

Build and run the example. Tap the Read button, and the greetings in different languages will be output in the Entry widget.



```
Good morning
早　上好
 Hyvää Huomenta
 Bonjour
 Guten Morgen
 Jó reggelt kívánok
 Buon giorno
お　はようございます。
   안녕하세요
 Bună dimineaţa!
 Buenos Días.
 Günaydın
 Xin chào
 Здра́вствуйте
```

2) Writing a Text File

We will implement a feature that, by adding a Button, saves a modified text as a file. The /res folder does not allow writing a file. So, you should save it in the /data folder. Add the second Button-creating code to the create_base_gui() function.

```
        {
            /* 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_HINT_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_HINT_EXPAND, EVAS_HINT_FILL, EVAS_HINT_FILL);
        }
    }
```

To save text in a file, add three new functions on top of the create_base_gui() function.

```
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() is a function that returns the absolute path of the file existing in the /data folder.

app_get_data_path() is an API that returns the absolute path of the /data folder.

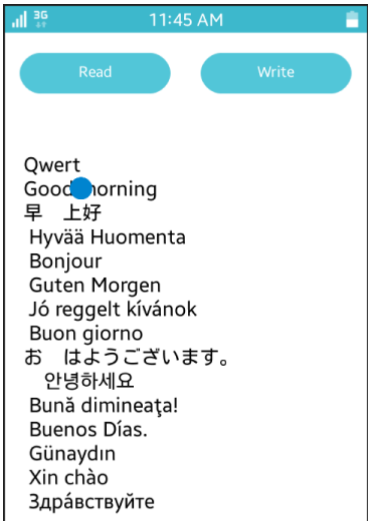
write_file() is a function that saves text data in a file.

fputs(char *, FILE *) is an API that saves text data in Filestream.

btn_write_cb() is a callback function for the second Button. Request the absolute path of the /data folder, and save the text data entered in the Entry in the file.

Run the example again. Tap the Read button to call the file, modify the content of the Entry, and tap the Write button.

Now, the modified content is saved in the /data folder.



Qwert
Good morning
早 上好
Hyvää Huomenta
Bonjour
Guten Morgen
Jó reggelt kívánok
Buon giorno
お はようございます。
안녕하세요
Bună dimineaţa!
Buenos Días.
Günaydın
Xin chào
Здрáвствуйте

3) Reading a File in the /data Folder

Unfortunately, the modified content will not be kept if you close the app and run it again. This is because the modified file is saved in the /data folder, but read from the /res folder. We are now going to change the feature so that when reading the file, the app searches for the file in the /data folder first, and when the file is not there, proceeds to search the /res folder.

Modify the content of the btn_read_cb() function as below. The function should be located lower than the app_get_data() function.

```
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);
}
```

The file is read from the /data folder first. If the data does not exist there, then it will be read from the /res folder.

Run the example again. Tap the Read button to call the file, modify the content of the Entry widget, and tap the Write button to save it.

Close the app (press and hold the Home button, and tap Clear All when the app list appears) and run it again. If you see the modified content when you tap the Read button, it is successfully done.

4) Related APIs

FILE *fopen(char *, char *): an API that returns the handle of a file. Pass the file path to the first parameter, and specify the file access mode to the second parameter. "r" indicates read only, while "w" indicates write only.

int fseek(FILE *__stream, long int __off, int __whence): an API that moves to a certain location of Filestream. The first, second, and third parameters are Filestream, the count of moved bytes, and starting point, respectively.

SEEK_SET, SEEK_CUR, and SEEK_END indicate the start of a file, current location, and the end of the file, respectively.

long int ftell(FILE *__stream): an API that returns the current location of the Filestream as the byte count. When it is located at the end, it will return the file size.

void rewind(FILE *__stream): an API that turns back the location of the Filestream to the original location.

char *fgets(char *__s, int __n, FILE *__stream): an API that reads text data from a file. Passing the maximum length of text to the second parameter and the Filestream to the third parameter makes the first parameter return the string data.

int fclose(FILE *__stream): an API to close the Filestream.

char *app_get_data_path(void): an API that returns the absolute path of the /data folder.

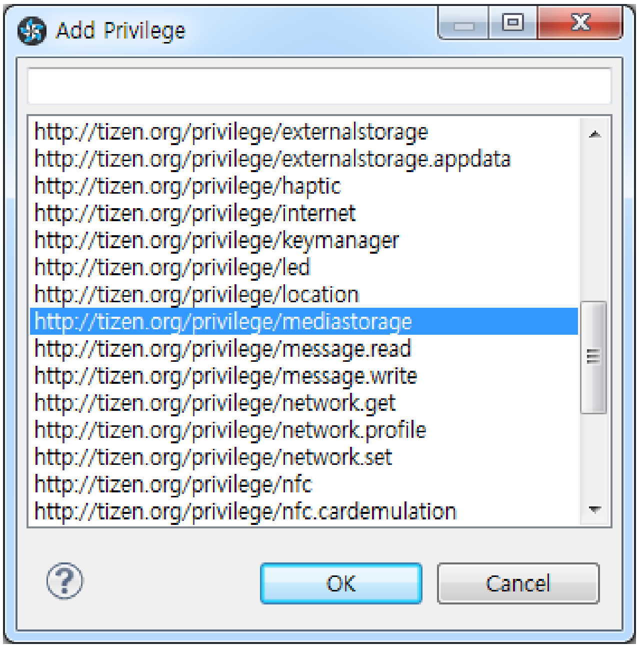
int fputs(char *__s, FILE *__stream): an API that saves text data in the Filestream.

59. Requesting a File List

Some file management apps like Astro have been used as an essential tool for file management for years. Requesting a file list is an essential feature for content-playing apps including an image viewer or audio player, as well as for file management apps. In this example, we are going to build a simple file management app where we can add files and folder lists to the List widget, select items, and move the folder path.

1) Registering a Privilege

Create a new source project and specify the project name as FileList. You need to have the applicable user privileges to request the file list saved in the memory. After the source project is created, open the tizen-manifest.xml file, and click Privileges among the tab buttons below. Then, click the Add button in the upper right corner. When the popup window appears, select http://tizen.org/privilege/mediastorage from the list, and click the OK button to close the window.



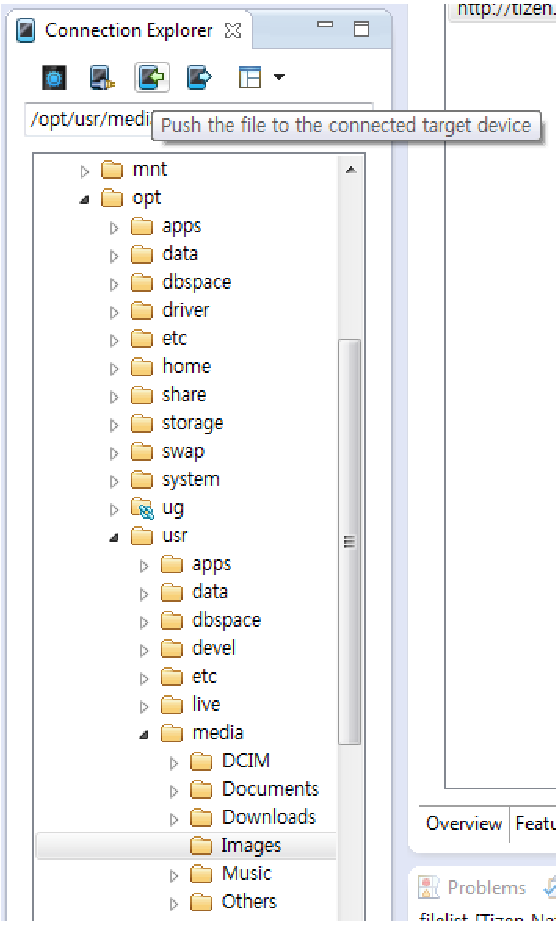
http://tizen.org/privilege/externalstorage
http://tizen.org/privilege/externalstorage.appdata
http://tizen.org/privilege/haptic
http://tizen.org/privilege/internet
http://tizen.org/privilege/keymanager
http://tizen.org/privilege/led
http://tizen.org/privilege/location
http://tizen.org/privilege/mediastorage
http://tizen.org/privilege/message.read
http://tizen.org/privilege/message.write
http://tizen.org/privilege/network.get
http://tizen.org/privilege/network.profile
http://tizen.org/privilege/network.set
http://tizen.org/privilege/nfc
http://tizen.org/privilege/nfc.cardemulation

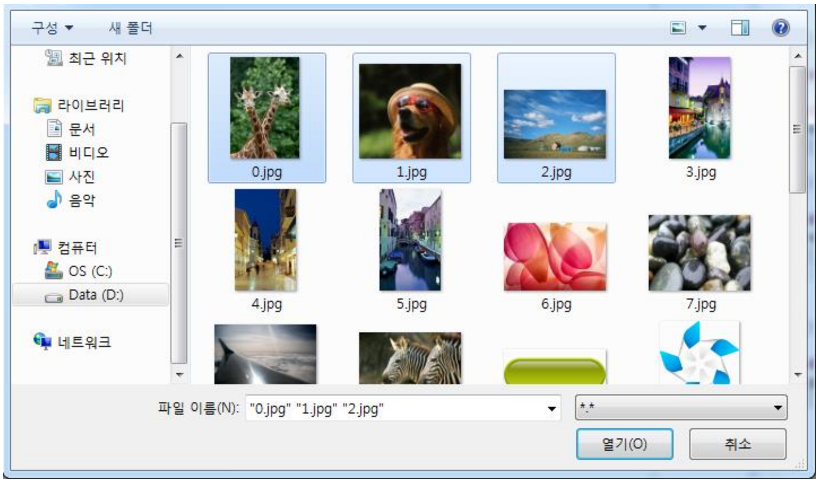
After saving, click the tizen-manifest.xml button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.example.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>
```

The shared memory of the emulator basically has divided folders, but it does not hold files. To test this feature, we are going to copy a few image files to the emulator memory. You can check the folder list in tree structure in Connection Explorer on the bottom left of Eclipse. The path of the shared folder to save images is /opt/usr/media/Images. Select this folder, and select 'Push the file to the connected target device' from the tool bar at the top.

When the popup window for file selection appears, go to the /Image folder of the appendix, select three image files (0.jpg, 1.jpg, and 2.jpg), and click the OK button. You can choose any files you want to use.

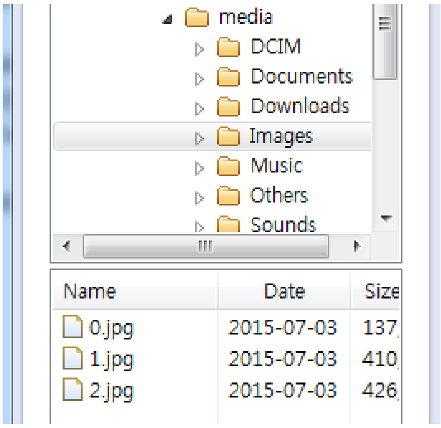




파일 이름(N): "0.jpg" "1.jpg" "2.jpg"

.

When the popup window closes, the selected files will be copied in the /Images folder.



Name	Date	Size
0.jpg	2015-07-03	137
1.jpg	2015-07-03	410
2.jpg	2015-07-03	426

2) Requesting an Internal Memory Folder List

In this section, we are going to request the internal memory folder list of the emulator. Open the source file (~.c) under the src folder and add defined variables and invariables.

```
#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;
```

"/opt/usr/media" is a root path for the shared folder of the internal memory.
"/opt/storage/sdcard" is a root path for the shared folder of the external memory.

list is a List widget variable to display the file list.

current_path is a string variable that saves the absolute path of the current folder.

Add new code at the end of the create_base_gui() function.

```
    /* 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 );
}
```

Allocate the memory to current_path, and copy the internal memory root folder path.

read_dir() is a function that requests the file list within a certain folder. Now, we are going to build the example.

Add a new function on top of the create_base_gui() function.

```
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_name);
        }
        else {
            dlog_print(DLOG_INFO, "tag", "[File] %s", pDirent->d_name);
        }
    }
    closedir(dir);
}
```

DIR is a structure that controls folders. It can read or delete a file list, and create a folder.

opendir(char *) is an API that returns a DIR object to control a certain folder.

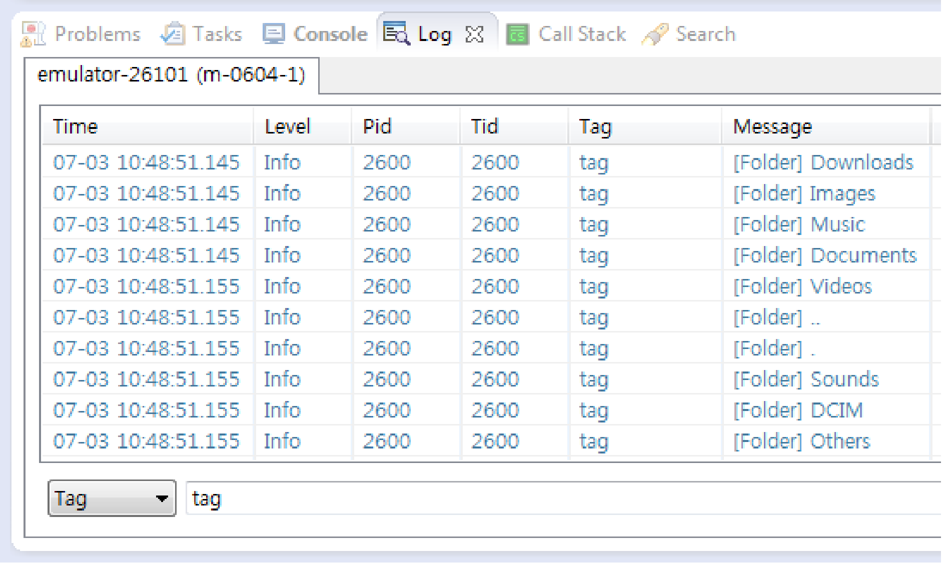
dirent is a structure to save a file (or folder) information. Among the properties, the file name is saved in d_name. The type of the file is saved in d_type. DT_DIR indicates a folder; otherwise, the object is a file.

readdir(DIR *) is an API that reads the file list from a certain folder and returns it as a type of dirent one by one. On reaching the end of the file list, NULL is returned.

Next, we are going to build code that distinguishes files and folders, and outputs the result as a Log message.

closedir(DIR *) is an API that closes DIR.

Build and run the example. The file and folder list existing in the shared folder of the internal memory appears on the Log panel.



```
Time                Level  Pid   Tid   Tag  Message
07-03 10:48:51.145  Info   2600  2600  tag  [Folder] Downloads
07-03 10:48:51.145  Info   2600  2600  tag  [Folder] Images
07-03 10:48:51.145  Info   2600  2600  tag  [Folder] Music
07-03 10:48:51.145  Info   2600  2600  tag  [Folder] Documents
07-03 10:48:51.155  Info   2600  2600  tag  [Folder] Videos
07-03 10:48:51.155  Info   2600  2600  tag  [Folder] ..
07-03 10:48:51.155  Info   2600  2600  tag  [Folder] .
07-03 10:48:51.155  Info   2600  2600  tag  [Folder] Sounds
07-03 10:48:51.155  Info   2600  2600  tag  [Folder] DCIM
07-03 10:48:51.155  Info   2600  2600  tag  [Folder] Others
```

3) Adding a File List to List Widget

Now, we will add the file list to the List widget as an item. Input Box- and List-creating code to the create_base_gui() function. Annotate the Label widget.

```
    /* 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);

    {
        /* 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);

        {
            /* 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_HINT_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);
```

Modify the read_dir() function as follows:

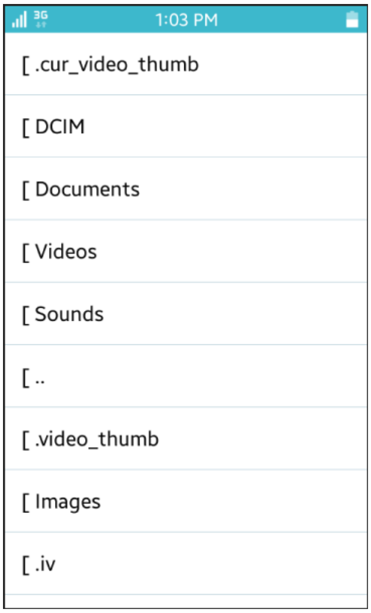
```
static void
read_dir(appdata_s *ad)
{
    DIR *dir = opendir(ad->current_path);
    if( !dir )
        return;

    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_name);
            sprintf(buf, "[ %s", pDirent->d_name);
        }
        else {
            dlog_print(DLOG_INFO, "tag", "[File] %s", pDirent->d_name);
            sprintf(buf, "# %s", pDirent->d_name);
        }
        elm_list_item_append(ad->list, buf, NULL, NULL, NULL, ad);
    }
    closedir(dir);
}
```

elm_list_clear(Evas_Object *) is an API that deletes all the items of the List widget.

To identify files from folders, the '[' symbol is added in front of the name if the content is a folder, while the '#' symbol is added in front of the name if the content is a file.

elm_list_item_append(Evas_Object *, char *, Evas_Object *, Evas_Object *, Evas_Smart_Cb , void *) is an API that adds a new item to the List widget.

Run the example again. The folder and file list within the root folder is added to the List widget. The '.' symbol indicates the current folder, while the '..' symbol indicates a top-level folder.



[.cur_video_thumb

[DCIM

[Documents

[Videos

[Sounds

[..

[.video_thumb

[Images

[.iv

4) Moving a Folder

We will implement a feature that when the user selects a List widget item, directs the user to the folder. To do so, the following features are necessary.

- Event callback function for selection of a List widget item
- Removing the '.' and '..' symbols from the content list
- If the current folder is not a root folder, the '..' symbol should be added in the first item.

Add new code to the read_dir() function.

```
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_clicked, 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_name);
            sprintf(buf, "[ %s", pDirent->d_name);
        }
        else {
            dlog_print(DLOG_INFO, "tag", "[File] %s", pDirent->d_name);
            sprintf(buf, "# %s", pDirent->d_name);
        }
        elm_list_item_append(ad->list, buf, NULL, NULL, list_item_clicked, ad);
        //elm_list_item_append(ad->list, buf, NULL, NULL, NULL, ad);
    }
    closedir(dir);
}
```

If the current folder is an internal memory root folder, add the symbol '..' as the first item of the List widget.

If the content name is '.' or '..,' it can simply be ignored.

Specify the name of the item selection event callback function as list_item_clicked. Now, we are going to build this function. Create three functions on top of the read_dir() function.

```
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() is a function that receives the text of a List item, removes the symbols, and identifies between a folder and a file.

If the first character is '#,' it is a file. If the first character is '[,' then it is a folder. If the text is '..,' it indicates a top-level folder.

get_new_path() is a function that requests the whole folder path by receiving the current path and the new folder name.

strrchr(char *, int) is an API that searches certain characters from a string. Start to search from the end. Return the pointer of the character at the very end.

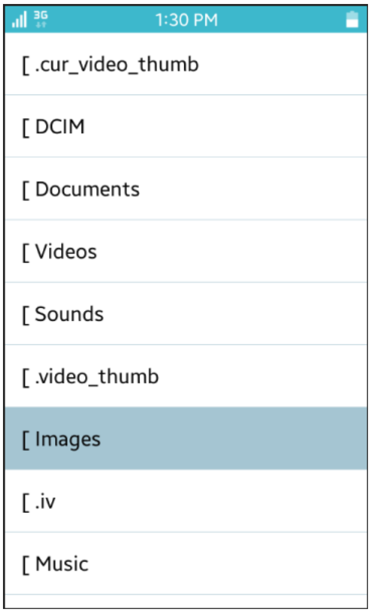
If the name of the new folder is '..,' it indicates that this is the top-level folder. In this case, you need to delete the name of the last folder in the current folder path. Otherwise, add a new folder name at the end of the current folder path to create an absolute path.

list_item_clicked() is an event function to select a List widget item. Request the caption text of the item, and separate the name of content. Then, create a new folder path, and add the content in the folder to the List widget.

However, if you build it as it is now, an error will occur because the read_dir() and list_item_clicked() functions call each other. In such cases, declare the header of the function. The location is either the top of the source file or the header file. In this example, we will declare it at the top of the source file. Go to the top of the source file, and declare the read_dir() function.

```
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);
```

Run the example again, and you will now see the '.' and '..' symbols are gone. When you select the Images item, the file list that you just copied will be displayed. The '..' symbol has been added to the first item. If you select it, it returns to the previous folder.



[.cur_video_thumb

[DCIM

[Documents

[Videos

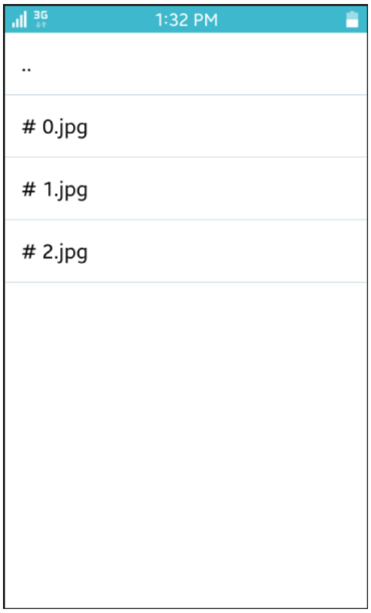
[Sounds

[.video_thumb

[Images

[.iv

[Music



..

0.jpg

1.jpg

2.jpg

5) ELM File Selector

You can easily implement a file manager by using the ELM File Selector widget. Create a new source project, and specify the name as ElmFileSelectorEx.

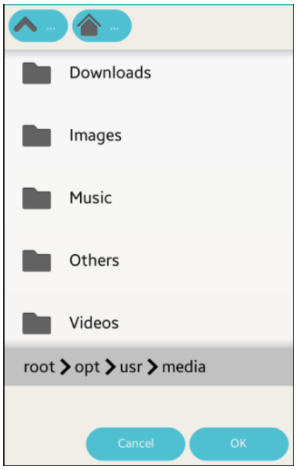
After the source project is created, open the tizen-manifest.xml file, and add the following privilege.

http://tizen.org/privilege/mediastorage

Open the source file (/src/elmfileselectorex.c), and add the new code to the create_base_gui() function. This code creates a Box and FileSelector widget. Delete the Label widget-creating code.

```
    /* 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);

    {
        /* 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);

        {
            Evas_Object *fs = elm_fileselector_add(ad->conform);
            evas_object_size_hint_weight_set(fs, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
            evas_object_size_hint_align_set(fs, EVAS_HINT_FILL, EVAS_HINT_FILL);
            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);
}
```

Build and run the example. The folder list is displayed in the FileSelector widget.



See the link below for further information on how to use the FileSelector widget.

https://docs.enlightenment.org/elementary/1.15.0/fileselector_example.html

https://docs.enlightenment.org/elementary/1.15.0/group__Fileselector.html

6) Related APIs

DIR: a structure that controls a folder. It can read or delete a file list, and create a folder.

DIR *opendir(char *__name): an API that returns a DIR object to control a specific folder.

dirent: a structure that saves file (or folder) information. Among the properties, the file name is saved in d_name. The type of the file is saved in d_type. DT_DIR indicates a folder; otherwise, the object is a file.

struct dirent *readdir(DIR *__dirp): an API that reads the file list within a certain folder and returns the files as a type of dirent one by one. Once reaching the end of the file list, NULL is returned.

int closedir(DIR *__dirp): an API that closes DIR.

void elm_list_clear(Evas_Object *obj): an API that deletes all the items of the List widget.

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): an API that adds new items to the List widget.

char *strrchr(char *__s, int __c): an API that searches certain characters from a string. It starts to search from the end, and returns the pointer of the character at the very end.

60. Using Preference

It is convenient if you save information such as the environment settings of an app in the registry. Preference is a storage space where you can save data locally. You can save the environment settings information here. When the app is deleted, the preference will also be deleted.

1) Saving String Data in Preference

Create a new source project and specify the project name as 'PreferenceEx.' After the source project is created, open the source file (~.c) under the src folder and add library header files and variables.

```
#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 is a library header file used for Preference.

stdlib.h is a library header file to change the type of strings and numbers.

We are going to input string data at entry1, and number data at spinner1.

Now, we are going to add some widgets on the screen. Create two functions on top of the create_base_gui() function.

```
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_dat
a)
{
     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_table_pack() is a function that adds a widget to a Table.

my_button_add() is a function that creates a Button widget.

Next, add new code to the create_base_gui() function.

```
    /* 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_config
_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);
}
```

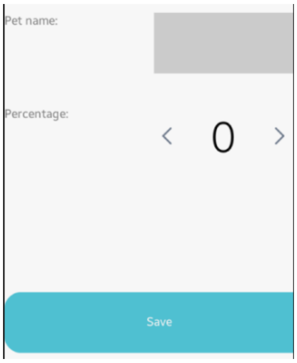
We have added one Box, one Table, two Label, one Bg, one Entry, one Spinner, and one Button widget. Enter a string in the first Entry, and tap the Button. The string will be saved in Preference.

Create a Button callback function on top of the create_base_gui() function.

```
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 *) is an API that saves string data in Preference. Pass the Key to the first parameter, and string data to the second parameter. Keys must be the same when performing reading and writing.

Build and run the example. Enter a string in the first Entry, and tap the Save button. The string will be saved. Unfortunately, we have not yet built the reading feature. Now, we are going to build the example.



2) Reading a String Preference

We will implement a feature that reads data in Preference by adding a second Button. Add new code to the create_base_gui() function.

```
            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);
        }
    }
```

Then, we are going to create a callback function for the added Button. Add a new function on top of the create_base_gui() function.

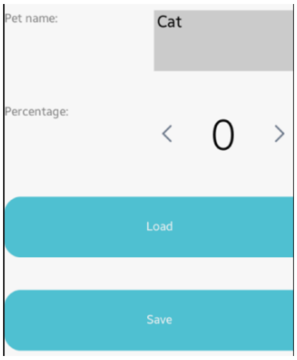
```
static void
btn_read_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    char *string_value = "";
    bool existing = false;

    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);
    }
}
```

When reading data in Preference, we first need to check if the data exists.

preference_is_existing(const char *, bool *) is an API that determines if certain data exists in Preference. Passing a Key to the first parameter makes the second parameter return whether or not the data exists.

preference_get_string(char *, char **) is an API that reads string data from Preference. Passing a Key to the first parameter makes the second parameter return the string data.

Run the example again, enter the string in the first Entry, and tap the Save button. Then, delete the characters entered in the Entry, and tap the Read button. The string that has been just entered will be displayed in the Entry.



Pet name: Cat

Percentage: 0

3) Reading and Writing Preference Numbers

Now, we are going to save numbers in Preference, and read them again. First, add new code as follows to the storage function.

```
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 *), which is the acronym of Array to Int, is an API that changes a string to numbers.

preference_set_int(char *, int) is an API that saves integer-type data in Preference.

Also, add new code to a function that reads data.

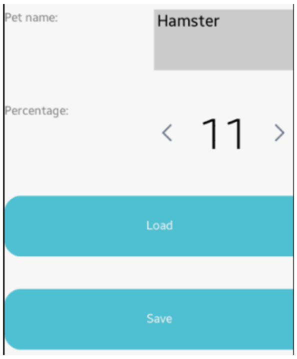
```
static void
btn_read_cb(void *data, Evas_Object *obj, void *event_info)
{
    appdata_s *ad = data;
    char *string_value = "";
    int integer_value;
    bool existing = false;

    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 *) is an API that reads integer-type data from Preference.

eina_convert_itoa(int, char *) is an API that converts integers to strings.

Run the example again. Enter a string in the first Entry, and enter numbers in the second Entry. Then, tap the Save button.

If you change the input data, run it again, and tap the Read button, the data saved in Preference will be displayed again.



Pet name: Hamster

Percentage: 11

4) Related APIs

int preference_set_string(const char *key, const char *value): an API that saves string data in Preference. Pass the Key to the first parameter, and string data to the second parameter. Keys must be the same when performing reading and writing.

int preference_is_existing(const char *key, bool *existing): an API that determines if certain data exists in Preference. Passing a Key to the first parameter makes the second parameter return whether or not the data exists.

int preference_get_string(const char *key, char **value): an API that reads string data from Preference. Passing a Key to the first parameter makes the second parameter return the string data.

int atoi(char *__nptr): the acronym of Array to Int; an API that changes a string to numbers.

int preference_set_int(char *key, int value): an API that saves integer-type data in Preference.

int preference_get_int(char *key, int *value): an API that reads integer-type data from Preference.

int eina_convert_itoa(int n, char *s): an API that converts integers to strings.

61. Example of Making a Report Card with SQLite

The information of user environment settings can be saved by using Preference. However, systematic and massive data like schedule management, contact information, and score management should be handled with a DB. Most mobile platforms provide SQLite as a DB. Tizen also supports SQLite. SQLite has relatively small maximum data capacity compared to Oracle or MS-SQL, but it is big enough for mobile devices. In addition, with easy operation and utilization of standard SQL query statements, SQLite has excellent compatibility with DBs which used to be used by other systems.

One DB includes one or more tables. A table is a two-dimensional table similar to an Excel spreadsheet. Multiple columns are placed horizontally on the table. Whenever data that consists of a collection of individual fields is filled, a new column of records will be added in the vertical direction. In this example, we are going to create a simple example of report card using SQLite.

1) Creating a DB

Create a new source project and specify the project name as SqliteEx. After the source project is created, open the source file (~.c) under the src folder, and add a library header file, variables, and a structure.

```
#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 is a library header file to use SQLite.

stdlib.h is a library header file to change the type of strings and numbers.

We added three Entries to the appdata structure. Now, we are going to enter a student name, and English and Math scores, respectively.

Then, we will output the DB list of the report card in the List widget.

sqlite3 is the object variable of the DB.

We are going to save the Key values of the currently selected records in current-key.

recdata is a structure to save the student's score data. One recdata means one record.

m_ad is declared as a global variable to make the appdata accessible anywhere.

Now, we are going to create a DB file in the /data folder, and then create a report card table. Create three functions on top of the create_base_gui() function.

```
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() is a function that creates a table by using an SQL query statement.

Among the query statement, 'CREATE TABLE' is a command to create a table.

'IF NOT EXISTS' is a command to create a table if there is no existing table.

In this example, the table name has been specified as ReportCard.

'KEY INTEGER PRIMARY KEY' is code to add a Key column. The available type is numbers. Whenever a record is added, the number will automatically increase.

'NAME TEXT NOT NULL' is code to add a column which saves a name. The available type is text, and it cannot be left blank.

'ENGLISH INT NOT NULL' is code to add a column which saves the English scores. The available type is numbers, and it cannot be left blank.

'MATH INT NOT NULL' is code to add a column which saves the Math scores. The available type is numbers, and it cannot be left blank.

sqlite3_exec(sqlite3*, char *, int(*callback), void *, char **) is an API that carries out an SQL query statement. The parameters listed in order are an SQLite object, query statement, name of the callback function, user data, and error message return.

init_db() is a function that creates a DB file.

sqlite3_shutdown() is an API that closes the DB.

sqlite3_config(int, ...) is an API that defines the properties of the DB. Passing SQLITE_CONFIG_URI saves data in the DB file.

sqlite3_initialize() is an API that initializes the DB.

app_get_data_path() is an API that returns the absolute path of the /data folder. The /res folder does not allow writing, so we will create a DB file in the /data folder.

strncat(char *, char *, size_t) is an API that adds a new string at the end of a string by defining the maximum length of the string. The parameters listed in order are original string arrays, string data to be added, and maximum length.

sqlite3_open(char *, sqlite3 **) is an API that opens the DB file. You need to create a DB file if it does not already exist. Passing the file path to the first parameter makes the second parameter return the DB object.

my_table_pack() is a function that adds a widget to a Table.

We are going to make the DB file automatically open when the app is run. Call the function above at the end of the create_base_gui() function.

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

    init_db(ad);
}
```

2) Adding New Records

We will implement a feature that adds new records in the DB when a student name, and English and Math scores are entered in three Entry widgets, and then a Button is tapped. Add new code to the create_base_gui() function. Annotate the Label widget.

```
    /* 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);

    {
        /* 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_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_EXPAND);
        /* 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);
```

We have created one Box, one Table, three Bg, and three Entry widgets. We also created a Button widget.

We will implement a feature that adds data that is entered in the Entry to the DB when you tap the button. Create two functions on top of the create_base_gui() function.

```
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);
}
```

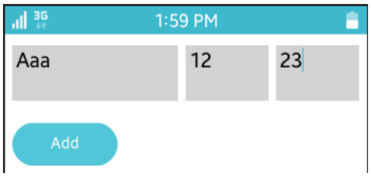
InsertRecord() is a function that adds new records by using an SQL query statement.

From the statement, 'INSERT INTO ReportCard' is a command to add new records to the table whose name is 'ReportCard.'

Pass the record data to VALUES(). We will not check the first column Key because it is created automatically.

When the user taps the Button, the btn_add_cb() function requests the data that is input in Entry, and saves it in the DB.

Build and run the example. Enter a student name in the first Entry, English scores in the second Entry, and Math scores in the third Entry. Then, tap the Add button. Now, the new records are added to the DB. Unfortunately, there is no way for us to check this ourselves. We have not yet implemented a feature to allow us to read data saved in the DB.



3) Reading the Data Saved in DB

We will implement a feature that outputs data saved in the DB and displays it on the screen. To do so, we need to add the List widget-creating code to the create_base_gui() function.

```
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);
```

appdata objects are saved in the global variables to be able to use them anywhere. Add two new functions. These functions should be placed on top of init_db() and btn_add_cb().

```
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;
}
```

When reading multiple records from the DB, individual record information will be passed to the callback function. db_read_cb() is a callback function that receives a record and handles it. The data is passed to the third parameter as an array.

Create the recdata_s structure, save data of each field, and add up all data to one string. Then, add the string to the List widget as a new item.

read_db() is a function that reads all data saved in the DB, and outputs it in the List widget.

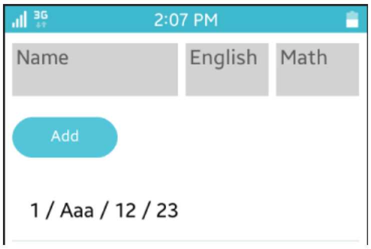
'select * from ReportCard' is a query statement that returns all data saved in the ReportCard table.

Call the function above when the app is run and the Button is tapped. Add new code at the end of the init_db() and btn_add_cb() functions.

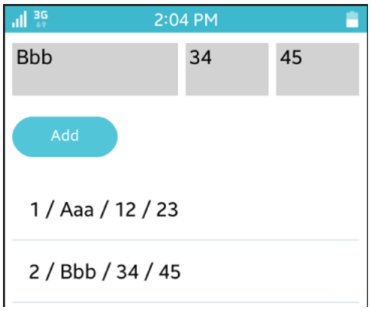
```
static void
init_db(appdata_s *ad)
{
    ~
    CreateTable(ad);
    read_db(ad);
}

static void
btn_add_cb(void *data, Evas_Object *obj, void *event_info)
{
    ~
    InsertRecord(ad, s_name, n_english, n_math);
    read_db(ad);
}
```

This is the reason why the read_db() function should be placed on top of the init_db() and btn_add_cb() functions.

Run the example again. The data that we just input has been added to the List widget. Input new data, and tap the Add button. You will now see a new item has been added to the List widget.



1 / Aaa / 12 / 23



Bbb 34 45

1 / Aaa / 12 / 23

2 / Bbb / 34 / 45

4) Modifying Data

We will now learn how to modify data that is already saved. When the user selects a List item, the data of the item can be output in the Entry. We will implement this feature now. In the db_read_cb() function, specify the name of a callback function to the code that adds a new item. Then, create the callback function.

```
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, (void*)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() is an event callback function to select a List widget item. Save the Key values in global variables, and output other data in the Entry widget.

The code for adding items in the List widget is modified in the db_read_cb() function. The name of the item selection callback function is specified.

Then, add new Button-creating code to the create_base_gui() function.

```
        /* 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);
```

We will now implement a feature that modifies the data of the currently selected item when you tap the button. Create two functions on top of the create_base_gui() function.

```
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₩', MAT
H=₩'%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() is a function that modifies data saved in the DB.

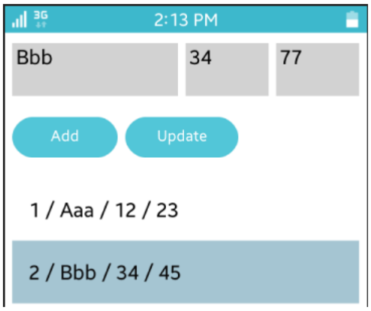
From the query statement, 'UPDATE ReportCard' modifies data saved in the ReportCard table.

SET NAME=₩'%s₩' saves string data in the NAME column.

WHERE KEY=₩'%s₩' modifies records whose Key values are the same.

btn_update_cb() is a function that saves data that is input in the Entry to the currently selected record.

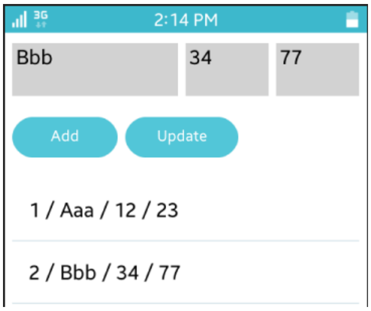
Run the example again. Select one of the List items, modify the data in the Entry, and tap the Update button. The data is now modified in the List widget and saved in the DB.



Bbb | 34 | 77

1 / Aaa / 12 / 23

2 / Bbb / 34 / 45



Bbb | 34 | 77

1 / Aaa / 12 / 23

2 / Bbb / 34 / 77

5) Deleting Records

We will implement a feature that deletes selected records by tapping a third Button. For this, we need to add a third Button. Add the Button-creating code in the create_base_gui() function.

```
/* 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);
```

Then, add two new functions on top of the create_base_gui() function.

```
static int
DelRecord(appdata_s *ad)
{
  char sql[256];
  char *ErrMsg;
  snprintf(sql, 256, "DELETE FROM ReportCard WHERE KEY=₩'%s₩';", ad->current_k
ey);

  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);
}
```

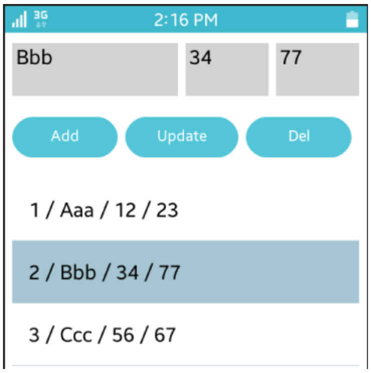
DelRecord() is a function that deletes the records saved in the DB.

From the query statement, 'DELETE FROM ReportCard' deletes data from the ReportCard table.

WHERE KEY=₩'%s₩' deletes records whose Key values are the same.

btn_del_cb() is a function that deletes the currently selected List item when the user taps the Del button.

Run the example again. Add some other items, select one of the List items, and tap the Del button. The selected item will be deleted.

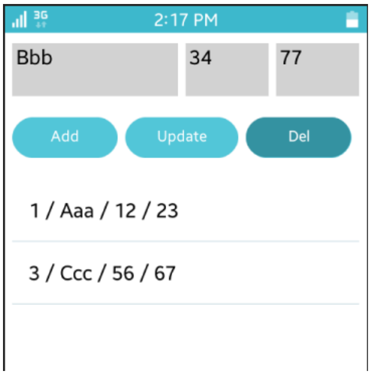


Bbb | 34 | 77

1 / Aaa / 12 / 23

2 / Bbb / 34 / 77

3 / Ccc / 56 / 67



Bbb | 34 | 77

1 / Aaa / 12 / 23

3 / Ccc / 56 / 67

6) Related APIs

int sqlite3_exec(sqlite3*, char *, int (*callback), void *, char **): an API that carries out an SQL query statement. The parameters listed in order are an SQLite object, query statement, name of the callback function, user data, and error message return.

int sqlite3_shutdown(): an API that closes the DB.

int sqlite3_config(int, ...): an API that defines the properties of the DB.

Passing SQLITE_CONFIG_URI saves data in the DB file.

int sqlite3_initialize(): an API that initializes the DB.

char *app_get_data_path(): an API that returns the absolute path of the /data folder. The /res folder does not allow writing, so we will create a DB file in the /data folder.

char *strncat(char *, char *, size_t): an API that adds a new string at the end of a string by defining the maximum length of the string. The parameters listed in order are original string arrays, string data to be added, and maximum length.

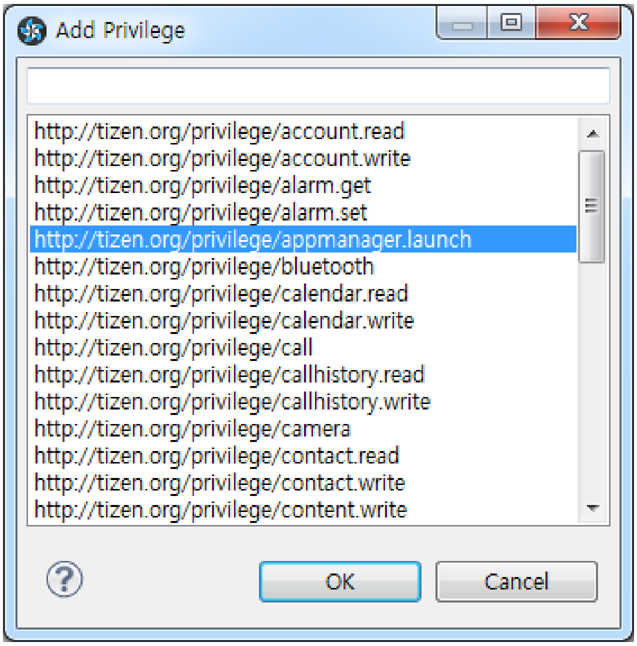
int sqlite3_open(char *, sqlite3 **): an API that opens the DB file. You need to create a DB file if it does not already exist. Passing the file path to the first parameter makes the second parameter return the DB object.

62. Calling an External App Using AppControl

When you develop an app, you may need to add features like an image viewer or camera. Realizing all the required features through hard coding would be rather difficult, but there simpler ways to achieve your aims. AppControl refers to apps installed on the Tizen platform as default. You can load this AppControl from other apps. Even though the apps are not default ones, you can load AppControl if you know the package name.

1) Registering a Privilege

Create a new source project and specify the project name as 'AppControlEx.' You need to have the applicable user privileges to run an external app. After the source project is created, open the tizen-manifest.xml file, and click Privileges among the tab buttons below. Then, click the Add button in the upper right corner. When the popup window appears, select http://tizen.org/privilege/appmanager.launch from the list, and click the OK button to close the window.



http://tizen.org/privilege/account.read
http://tizen.org/privilege/account.write
http://tizen.org/privilege/alarm.get
http://tizen.org/privilege/alarm.set
http://tizen.org/privilege/appmanager.launch
http://tizen.org/privilege/bluetooth
http://tizen.org/privilege/calendar.read
http://tizen.org/privilege/calendar.write
http://tizen.org/privilege/call
http://tizen.org/privilege/callhistory.read
http://tizen.org/privilege/callhistory.write
http://tizen.org/privilege/camera
http://tizen.org/privilege/contact.read
http://tizen.org/privilege/contact.write
http://tizen.org/privilege/content.write

Repeat the same process to add the following two user privileges.

- http://tizen.org/privilege/internet
- http://tizen.org/privilege/email

After saving, click the tizen-manifest.xml button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.exam
ple.appcontrolex" version="1.0.0">
    <profile name="mobile"/>
    <ui-application appid="org.example.appcontrolex" exec="appcontrolex" multiple="f
alse" 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) Loading the Example

Open the source file (~.c) under the src folder, and add library header files, variables, and a structure.

```
#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 and app_control.h are library header files to use AppControl.

“/opt/usr/media” is a root path for the shared folder.

Let's load the HelloWorld example from all the examples we have built so far. Add the Box- and Button-creating code to the create_base_gui() function.

```
    /* 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);
}
```

Create two functions on top of the create_base_gui() function.

```
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_CONTROL_E
RROR_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_EXPAN
D);
        /* 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() is a Button callback function.

app_control_h is an AppControl structure.

app_control_create(app_control_h *) is an API that creates an AppControl object.

app_control_set_operation(app_control_h, char *) is an API that specifies the role of AppControl. When loading an external app, simply pass APP_CONTROL_OPERATION_DEFAULT.

app_control_set_app_id(app_control_h, char *) is an API that specifies the package name of the example to be loaded to AppControl. The package name of the HelloWorld example is "org.example.helloworld."

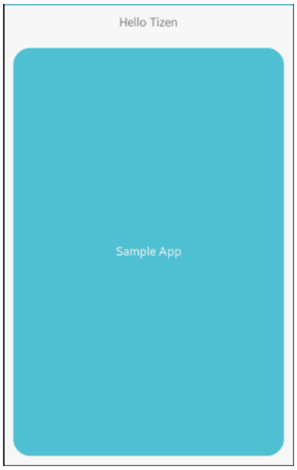
app_control_send_launch_request(app_control_h, app_control_reply_cb, void *) is an API that runs AppControl.

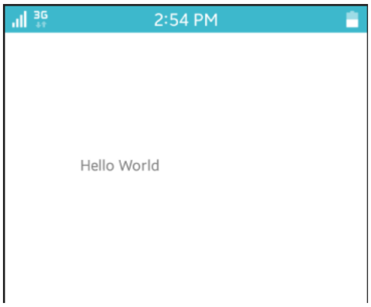
app_control_destroy(app_control_h) is an API that deletes the AppControl object.

my_box_pack() is a function that adds a widget to a Box.

Before running the example, check if the HelloWorld example is installed on the emulator. If it is not installed yet, create a source project with the name 'HellowWorld,' and install it on the emulator.

Once the HellowWorld example is installed on the emulator, build the AppControlEx example and run it. Tap the Button, and the HelloWorld example will run.





3) Passing Data to Other Apps

Now, we are going to pass data when loading the HelloWorld example. Add new code to the btn_sample_app_cb() function of the appcontrolex.c file.

```
┌[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_CONTROL_E
RROR_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 *) is an API that adds data to AppControl. The parameters listed in order are an AppControl object, Key values, and text data.

The sending part has been built. Now, we will implement the receiving part. Open the source file (/src/helloworld.c) of the HelloWorld example, and add a global variable at the top.

```
 ┌[helloworld.c]──────────────────────────┐
typedef struct appdata {
    Evas_Object *win;
    Evas_Object *conform;
    Evas_Object *label;
} appdata_s;

char recv_data[100];
 └──────────────────────────────────────┘
```

Event functions are called as many as the added number of data to AppControl. To save all the data, global variables have been declared.

Go to the bottom of the source file, you will see the app_control() function. This is the event function of AppControl. Now, add new code and functions.

[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);
}
```

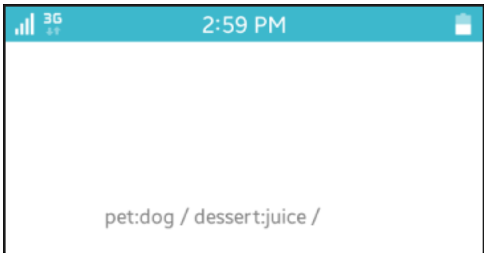
_app_control_extra_data_cb() is a callback function that receives individual data. As we passed two data in the AppControlEx example, this function will be called twice.

app_control_get_extra_data(app_control_h, char *, char **) is an API that extracts data from AppControl. Passing a Key to the second parameter makes the third parameter return the data.

app_control() is a callback function which is run when an AppControl object is received.

app_control_foreach_extra_data(app_control_h, app_control_extra_data_cb, void *) is an API that specifies a processing function for individual data saved in the AppControl object.

We will install the HelloWorld example on the emulator, and then run the AppControlEX example. When you tap the Button, the HelloWorld example will run, and the data passed from the AppControlEX example will be displayed in the Label widget.



pet:dog / dessert:juice /

4) Camera AppControl

Now, we will learn how to load the camera app from among the default apps. Go to the source file of the AppControlEX example, and create a new Button at the end of the create_base_gui() function.

[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);
        }
    }
```

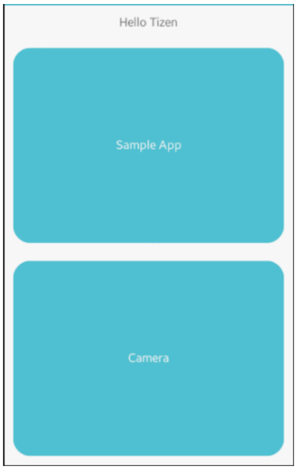
Now, we are going to create a callback function for the second Button. Add a new function on top of the create_base_gui() function.

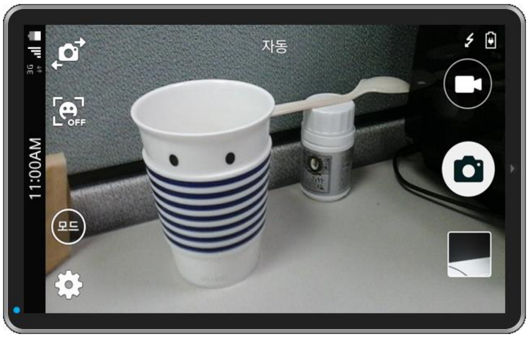
```
static void
btn_camera_cb(void *data, Evas_Object *obj, void *event_info)
{
    app_control_h app_control;

    app_control_create(&app_control);
    app_control_set_operation(app_control, APP_CONTROL_OPERATION_CREATE_CONTE
NT);
    app_control_set_mime(app_control, "image/jpg");
    if (app_control_send_launch_request(app_control, NULL, NULL) == APP_CONTROL_E
RROR_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 *) is an API that specifies the role of AppControl. When you take a picture, an image file will be created. So, you need to pass APP_CONTROL_OPERATION_CREATE_CONTENT.

app_control_set_mime(app_control_h, char *) is an API that specifies the MIME type. There might be different kinds of camera apps installed. That is why you should specify the MIME type rather than a certain app.

Run the example again and tap the second Button. Then, the camera app will be run. If the emulator cannot be run, you can test it from your phone.





5) Email AppControl

Now, we will learn how to load the email app from among the default apps. Create a new Button at the end of the create_base_gui() function.

```
            /* 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);
        }
    }
```

Then, we are going to create a callback function for the third Button. Add a new function on top of the create_base_gui() function.

```
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_address);
    app_control_add_extra_data(app_control, APP_CONTROL_DATA_SUBJECT, subject);
    if (app_control_send_launch_request(app_control, NULL, NULL) == APP_CONTROL_E
RROR_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);
}
```

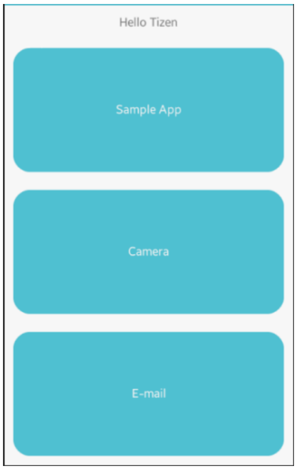
app_control_set_operation(app_control_h, char *) is an API that specifies the role of AppControl. When sending an email, pass APP_CONTROL_OPERATION_COMPOSE.

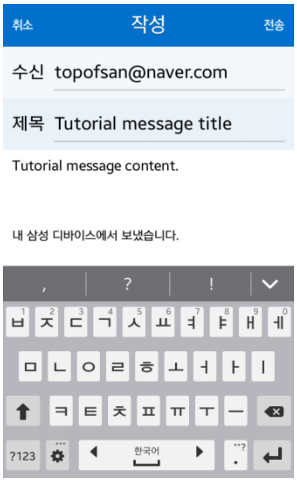
app_control_set_app_id(app_control_h, char *) is an API that specifies the package name of the example to be loaded to AppControl. Sending an email works with "email-composer-efl."

app_control_add_extra_data(app_control_h, char *, char *) is an API that adds data to AppControl. The parameters listed in order are an AppControl object, Key values, and text data. Keys that are used for the email are as follows:

- APP_CONTROL_DATA_TEXT: Body of the email
- APP_CONTROL_DATA_TO: The recipient's email address
- APP_CONTROL_DATA_SUBJECT: Subject (title) of the email

Run the example again and tap the third Button. The email app will run. If the emulator cannot be run, you can test it from your phone.





수신 topofsan@naver.com

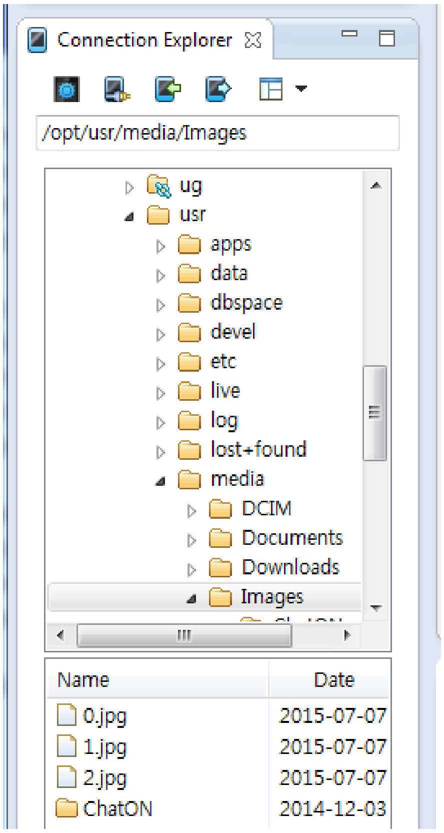
제목 Tutorial message title

Tutorial message content.

내 삼성 디바이스에서 보냈습니다.

6) Image Viewer AppControl

Now, we will learn how to load the Image Viewer app from among the default apps. You need an image file to display an image in the Image Viewer app. Check if the 0.jpg file exists in the /opt/usr/media/Images folder of the emulator. If the file does not exist, copy the 0.jpg file from the /image folder of the appendix to the user device. Now, simply drag the file from the Eclipse Connection Explorer to the /opt/usr/media/Images folder.



/opt/usr/media/Images

```
ug
usr
    apps
    data
    dbspace
    devel
    etc
    live
    log
    lost+found
    media
        DCIM
        Documents
        Downloads
        Images
```

Name	Date
0.jpg	2015-07-07
1.jpg	2015-07-07
2.jpg	2015-07-07
ChatON	2014-12-03

Create a new Button at the end of the create_base_gui() function.

```
            /* 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);
        }
    }
```

Next, we are going to create a callback function for the fourth Button. Add a new function on top of the create_base_gui() function.

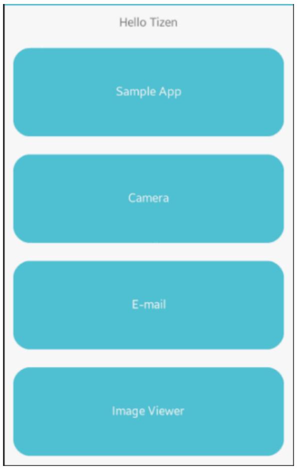
```
static void
btn_image_viewer_cb(void *data, Evas_Object *obj, void *event_info)
{
    app_control_h app_control;
    char buf[PATH_MAX];
    strcat(buf, FM_PHONE_FOLDER);
    strcat(buf, "/Images/0.jpg");

    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_CONTROL_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);
}
```

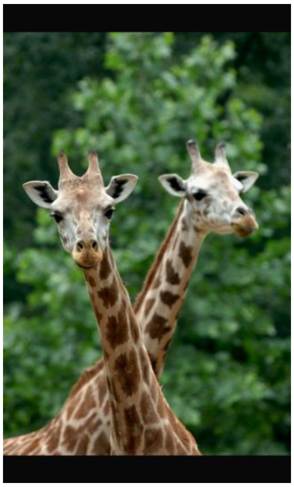
app_control_set_operation(app_control_h, char *) is an API that specifies the role of AppControl. When loading the Image Viewer app, simply pass APP_CONTROL_OPERATION_VIEW.

app_control_set_uri(app_control_h, char *) is an API that specifies the path of content. The path of the image file is specified.

app_control_set_mime(app_control_h, char *) is an API that specifies the MIME type. There might be different kinds of Image Viewer apps installed. That is why we should specify the MIME type rather than a specific app.

Run the example again and tap the fourth Button. Then, Image View will be run.


Hello Tizen
Sample App
Camera
E-mail
Image Viewer



7) Web Browser AppControl

Now, we will learn how to load the web browser app from among the default apps. Create a new Button at the end of the create_base_gui() function.

```
            /* 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, a
d);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

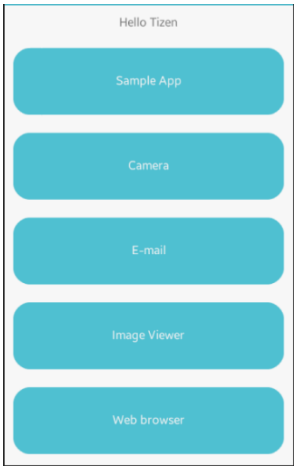
Next, we are going to create a callback function for the fifth Button. Add a new function on top of the create_base_gui() function.

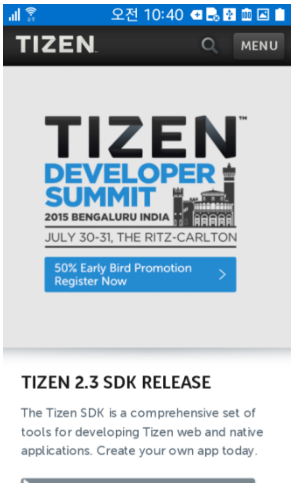
```
static void
btn_web_browser_cb(void *data, Evas_Object *obj, void *event_info)
{
    app_control_h app_control;

    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_CONTROL_E
RROR_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 *) is an API that specifies the package name of the example to be loaded to AppControl. The package name of the default web browser app is "com.samsung.browser."

app_control_set_uri(app_control_h, char *) is an API that specifies the path of content. In this case, we specified the Tizen website address.

Run the example again and tap the fifth Button. The web browser will run.





TIZEN 2.3 SDK RELEASE

The Tizen SDK is a comprehensive set of tools for developing Tizen web and native applications. Create your own app today.

8) Related APIs

int app_control_create(app_control_h *): creates an AppControl object.

int app_control_set_operation(app_control_h, char *): an API that specifies the role of AppControl. When loading an external app, simply pass APP_CONTROL_OPERATION_DEFAULT.

int app_control_set_app_id(app_control_h, char *): an API that specifies the package name of the example to be loaded to AppControl. The package name of the HelloWorld example is "org.example.helloworld."

int app_control_send_launch_request(app_control_h, app_control_reply_cb, void *): an API that runs AppControl.

int app_control_destroy(app_control_h): deletes the AppControl object.

int app_control_add_extra_data(app_control_h, char *, char *): an API that adds data to AppControl. The parameters listed in order are an AppControl object, Key values, and text data.

int app_control_get_extra_data(app_control_h, char *, char **): an API that extracts data from AppControl. Passing a Key to the second parameter makes the third parameter return the data.

int app_control_foreach_extra_data(app_control_h, app_control_extra_data_cb, void *): an API that specifies a processing function for individual data saved in the AppControl object.

int app_control_set_operation(app_control_h, char *): an API that specifies the role of AppControl. When you take a picture, an image file will be created. So, you need to pass APP_CONTROL_OPERATION_CREATE_CONTENT.

int app_control_set_mime(app_control_h, char *): an API that specifies the MIME type.

int app_control_set_operation(app_control_h, char *): an API that specifies the role of AppControl. When sending an email, pass APP_CONTROL_OPERATION_COMPOSE. When loading the Image Viewer app, pass APP_CONTROL_OPERATION_VIEW.

int app_control_set_app_id(app_control_h, char *): an API that specifies the package name of the example to be loaded to AppControl. Sending an email works with "email-composer-efl." The package name of the default web browser app is "com.samsung.browser."

int app_control_add_extra_data(app_control_h, char *, char *): an API that adds data to AppControl. The parameters listed in order are an AppControl object, Key values, and text data. Keys that are used for the email are as follows:

- APP_CONTROL_DATA_TEXT: Body of the email
- APP_CONTROL_DATA_TO: The recipient's email address
- APP_CONTROL_DATA_SUBJECT: Subject (title) of the email.

int app_control_set_uri(app_control_h, char *): an API that specifies the path of content.

63. Developing a Service App

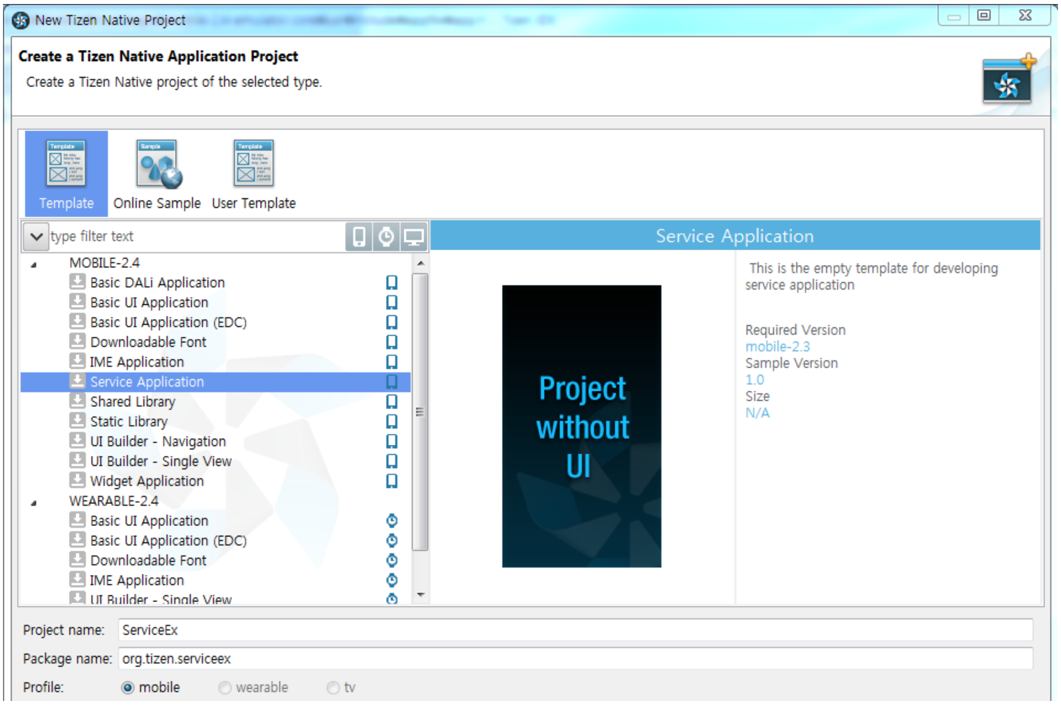
Service is an app running in the background without a UI visible to the user. Features such as antivirus, anti-theft, and communication features are often implemented using a Service app.

1) Creating a Source Project for Service

The first step is to create a new source project. Select [File > New Tizen Native Project] from the main menu of Eclipse.

When a popup window appears, select [Template > MOBILE-2.x > Service Application].

Enter 'ServiceEx' in the Project name field and click Finish.



New Tizen Native Project

Create a Tizen Native Application Project

Create a Tizen Native project of the selected type.

MOBILE-2.4
- Basic DALi Application
- Basic UI Application
- Basic UI Application (EDC)
- Downloadable Font
- IME Application
- Service Application
- Shared Library
- Static Library
- UI Builder - Navigation
- UI Builder - Single View
- Widget Application

WEARABLE-2.4
- Basic UI Application
- Basic UI Application (EDC)
- Downloadable Font
- IME Application
- UI Builder - Single View

Service Application

This is the empty template for developing service application

Required Version
mobile-2.3
Sample Version
1.0
Size
N/A

Project name: ServiceEx

Package name: org.tizen.serviceex

Profile: ● mobile ○ wearable ○ tv

We are now going to implement a feature that automatically terminates a Service app 5 seconds after it is launched. Open the source file (~.c) under the src folder and add a #define constant and variables.

```
#include <service_app.h>
#include "serviceex.h"
#include <ecore.h>

Ecore_Timer *timer1;
int timer_count = 0;
```

Ecore_Timer is the Timer structure.

Enter the count of the Timer event occurrences in timer_count.

We will make it so that a Timer is launched automatically when a Service app is launched. service_app_create() is an event function that is called when a Service app is launched. service_app_terminate() is an event function that is called when a Service app is launched. Add new code.

```
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;
}
```

This code displays a log message when an app is launched and terminated. This code also starts a Timer when an app is launched.

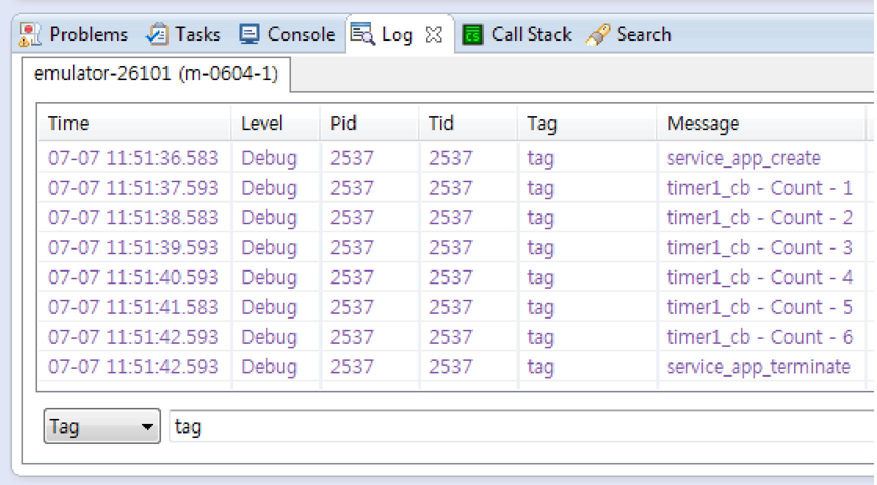
We are now going to create a callback function for the Timer event. Add a new function on top of the service_app_create() function.

```
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() is the Timer event function. Executed once every second, this function displays a log message and, 5 seconds later, automatically terminates the app.

service_app_exit() is an API that terminates a Service app.

Build and run the example. No changes occur on the emulator. This is because Service apps do not have a UI screen. We are now going to view the log message. To view log messages, select a Tag from the combo box at the bottom of the Eclipse Log pane and enter 'tag' in the Edit box to the right. An app start message, six counts, and an app termination message are displayed.



Time	Level	Pid	Tid	Tag	Message
07-07 11:51:36.583	Debug	2537	2537	tag	service_app_create
07-07 11:51:37.593	Debug	2537	2537	tag	timer1_cb - Count - 1
07-07 11:51:38.583	Debug	2537	2537	tag	timer1_cb - Count - 2
07-07 11:51:39.593	Debug	2537	2537	tag	timer1_cb - Count - 3
07-07 11:51:40.593	Debug	2537	2537	tag	timer1_cb - Count - 4
07-07 11:51:41.583	Debug	2537	2537	tag	timer1_cb - Count - 5
07-07 11:51:42.593	Debug	2537	2537	tag	timer1_cb - Count - 6
07-07 11:51:42.593	Debug	2537	2537	tag	service_app_terminate

2) Sending an Event from an External App to a Service App

In this section, we are going to implement a feature that terminates a Service app by letting an external app send an event. The process is the same as when passing data from the AppControlEx example to the HelloWorld example.

The service_app_control() function in the source file is the AppControl event function. Add new code to this function and add a new function on top of it.

```
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);
    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() is a callback function that receives individual data. If two pieces of data are saved in AppControl, this function is called twice.

app_control_get_extra_data(app_control_h, char *, char **) is an API that extracts data from AppControl. Passing a key to the second parameter makes the third parameter return data.

service_app_control() is a callback function which is run when an AppControl object is received.

app_control_foreach_extra_data(app_control_h, app_control_extra_data_cb, void *) is an API that specifies a function that processes individual data saved in an AppControl object.

Modify the code of the timer1_cb() function as follows. The Service app is automatically terminated after 5 seconds, and therefore it is necessary to specify a longer time.

```
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;
}
```

We are now going to implement a feature that makes an external app call a Service app. Open the source file we created previously in the AppControlEx example. Then, add Button-creating code at the end of the create_base_gui() function.

[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, ad);
            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, a
d);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

Add a Button callback function on top of the create_base_gui() function.

```
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_CONTROL_E
RROR_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 is an AppControl structure.

app_control_create(app_control_h *) is an API that creates an AppControl object.

app_control_set_operation(app_control_h, char *) is an API that specifies the role of AppControl. To load an external app, pass APP_CONTROL_OPERATION_DEFAULT.

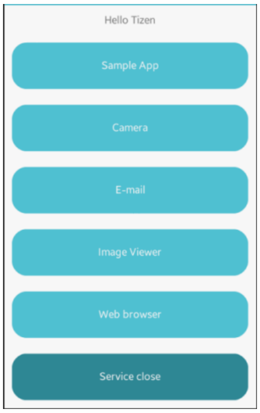
app_control_add_extra_data(app_control_h, char *, char *) is an API that adds data to AppControl. The parameters listed in order are an AppControl object, Key values, and text data.

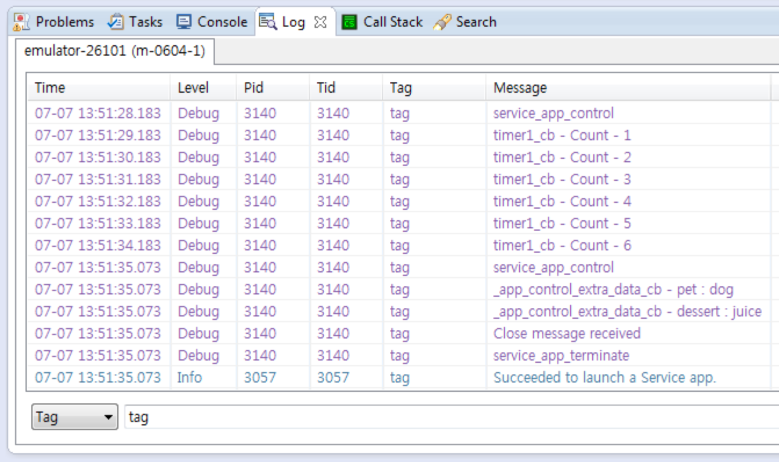
app_control_set_app_id(app_control_h, char *) is an API that specifies the package name of the example to be loaded to AppControl.

app_control_send_launch_request(app_control_h, app_control_reply_cb, void *) is an API that runs AppControl.

app_control_destroy(app_control_h) is an API that deletes an AppControl object.

Run the Service app and then run the AppControlEx example. Tap the sixth Button. The Service app will stop displaying log messages. In other words, the Service app has been terminated.





```
Time                Level  Pid   Tid   Tag  Message
07-07 13:51:28.183  Debug  3140  3140  tag  service_app_control
07-07 13:51:29.183  Debug  3140  3140  tag  timer1_cb - Count - 1
07-07 13:51:30.183  Debug  3140  3140  tag  timer1_cb - Count - 2
07-07 13:51:31.183  Debug  3140  3140  tag  timer1_cb - Count - 3
07-07 13:51:32.183  Debug  3140  3140  tag  timer1_cb - Count - 4
07-07 13:51:33.183  Debug  3140  3140  tag  timer1_cb - Count - 5
07-07 13:51:34.183  Debug  3140  3140  tag  timer1_cb - Count - 6
07-07 13:51:35.073  Debug  3140  3140  tag  service_app_control
07-07 13:51:35.073  Debug  3140  3140  tag  _app_control_extra_data_cb - pet : dog
07-07 13:51:35.073  Debug  3140  3140  tag  _app_control_extra_data_cb - dessert : juice
07-07 13:51:35.073  Debug  3140  3140  tag  Close message received
07-07 13:51:35.073  Debug  3140  3140  tag  service_app_terminate
07-07 13:51:35.073  Info   3057  3057  tag  Succeeded to launch a Service app.
```

3) Related APIs

app_control_h: the AppControl structure.

int app_control_create(app_control_h *): an API that creates an AppControl object.

int app_control_set_operation(app_control_h, char *): an API that specifies the role of AppControl. To load an external app, pass APP_CONTROL_OPERATION_DEFAULT.

int app_control_add_extra_data(app_control_h, char *, char *): an API that adds data to AppControl. The parameters listed in order are an AppControl object, Key values, and text data.

int app_control_set_app_id(app_control_h, char *): an API that specifies the package name of the example to be loaded to AppControl.

int app_control_send_launch_request(app_control_h, app_control_reply_cb, void *): an API that runs AppControl.

int app_control_destroy(app_control_h): an API that deletes an AppControl object.

int app_control_get_extra_data(app_control_h, char *, char **): an API that extracts data from AppControl. Passing a key to the second parameter makes the third parameter return data.

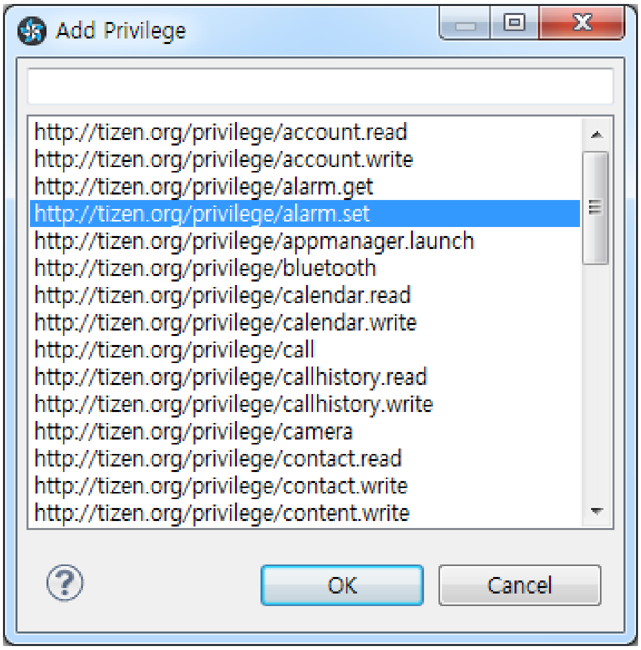
int app_control_foreach_extra_data(app_control_h, app_control_extra_data_cb, void *): an API that specifies a processing function for individual data saved in the AppControl object.

64. Alarm: Starting an APP at a Specified Time

When you develop an alarm app such as a wake-up call app, it is necessary to enable the app to be started automatically after a specified amount of time, even when it is closed. Using Alarm makes it possible to enable an app to be started at a specified time. It has a Timer feature, and therefore it can also enable an App to be started after a certain amount of time has passed. Alarm works in a similar way to AppControl.

1) Registering a Privilege

Create a new source project and specify the project name as 'Alarm.' You need to have the applicable user privileges to use the Alarm. After the source project is created, open the tizen-manifest.xml file, and click Privileges among the tab buttons below. Then, click the Add button in the upper right corner. When the popup window appears, select http://tizen.org/privilege/alarm.set from the list, and click the OK button to close the window.



http://tizen.org/privilege/account.read
http://tizen.org/privilege/account.write
http://tizen.org/privilege/alarm.get
http://tizen.org/privilege/alarm.set
http://tizen.org/privilege/appmanager.launch
http://tizen.org/privilege/bluetooth
http://tizen.org/privilege/calendar.read
http://tizen.org/privilege/calendar.write
http://tizen.org/privilege/call
http://tizen.org/privilege/callhistory.read
http://tizen.org/privilege/callhistory.write
http://tizen.org/privilege/camera
http://tizen.org/privilege/contact.read
http://tizen.org/privilege/contact.write
http://tizen.org/privilege/content.write

After saving, click the 'tizen-manifest.xml' button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3" package="org.exam
ple.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) Starting a Timer Alarm

In this section, we are going to implement a feature that runs the HelloWorld example 3 seconds after a Button is tapped. If the HelloWorld example has not been installed on your emulator, install the HelloWorld example first. Open the source file (~.c) under the src folder and add library header files and variables.

```
#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 is a library header file for using an Alarm.

time.h is a library header file for using time-related APIs.

In timer_id, we will enter the ID of the Timer Alarm. We can terminate the Alarm using this.

In date_id, we will enter the ID of the Date Alarm. We can terminate the Alarm using this.

Add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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_EXPAN
D);
          /* 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);
}
```

Add Button-creating code at the end of the create_base_gui() function.

```
    /* 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_EX
PAND);
        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);
}
```

Add a Button callback function on top of the create_base_gui() function.

```
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_DEFAULT);
   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 is an AppControl structure.

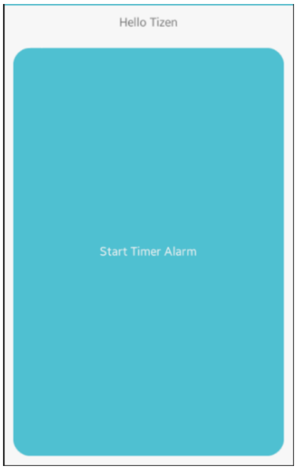
app_control_create(app_control_h *) is an API that creates an AppControl object.

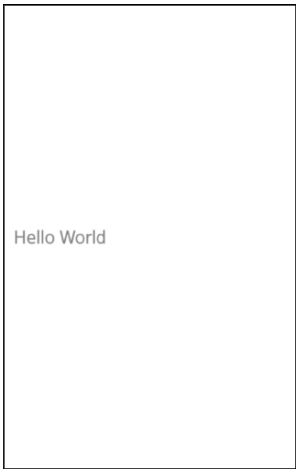
app_control_set_operation(app_control_h, char *) is an API that specifies the role of AppControl. To load an external app, pass APP_CONTROL_OPERATION_DEFAULT.

app_control_set_app_id(app_control_h, char *) is an API that specifies the package name of the example to be loaded to AppControl. The package name of the HelloWorld example is "org.example.helloworld."

alarm_schedule_after_delay(app_control_h, int, int, int *) is an API that passes an AppControl event after a certain amount of time has passed. The second parameter indicates the time interval for the first execution, and the third parameter indicates the time interval for re-executions. Entering 0 runs the AppControl event only once. The fourth parameter receives the ID of the Alarm, which is necessary for terminating the Alarm.

Build and run the example. Tap the Button, and the HelloWorld example will be run after 3 seconds.





3) Terminating a Timer Alarm

In this section, we are going to implement a feature that adds a Button to the Alarm example and terminates the Alarm. Add new code at the end of the create_base_gui() function.

```
            /* 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);
        }
    }
```

Then, create a Button callback function on top of the create_base_gui() function and modify the btn_start_timer_cb() function.

```
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_DEFAULT);
    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");
}
```

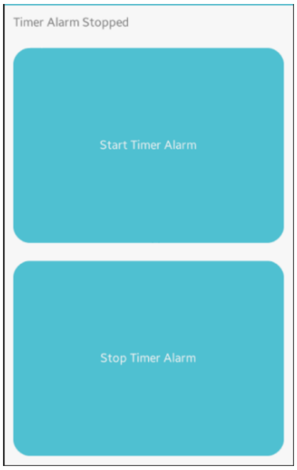
Changing the value of the REMIND variable to 8 runs the HelloWorld example every 8 seconds.

btn_stop_timer_cb() is a function that terminates the Timer Alarm when the user taps a Button.

alarm_cancel(int alarm_id) is an API that terminates an Alarm. It passes the ID of an Alarm.

Run the example again and tap the first Button. The HelloWorld example is run. Terminate the HelloWorld app; it is then run again every 8 seconds. Tap the second Button, and the HelloWorld example will no longer be run: the Alarm has been terminated.

To terminate an Alarm, its ID is required. Therefore, if you have started an Alarm, you need to terminate the Alarm by tapping the second Button. If the HelloWorld example keeps being run even after you tap the Stop button, delete your emulator and create it again. It will then be reset.


Timer Alarm Stopped

3) Date Alarm

We are now going to implement a feature that runs a certain app at a specified time. Add two Button-creating codes to the create_base_gui() function.

```
            /* 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);

            /* 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, 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, "Stop Date Alarm");
            evas_object_smart_callback_add(btn, "clicked", btn_stop_date_cb, ad);
            my_box_pack(box, btn, 1.0, 1.0, -1.0, -1.0);
        }
    }
```

This is code that creates a third Button and a fourth Button. Then, add a Button callback function on top of the create_base_gui() function.

```
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_DEFAULT);
    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() is a function that creates a Date Alarm.

struct tm is a time structure where the time and date are saved.

alarm_get_current_time(struct tm *) is an API that returns the current time in struct tm type.

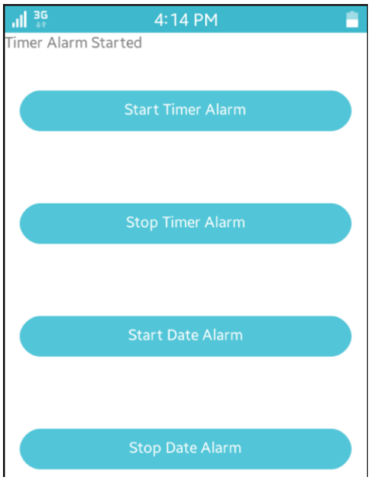
In tm.tm_sec, time is saved in seconds. We specified the execution time as 4 seconds after the current time.

alarm_schedule_at_date(app_control_h, struct tm *, int, int *) is an API that runs an AppControl event at a specified time. Pass the struct tm object that contains the time and date to the second parameter, and pass the re-execution interval to the third parameter. Passing 0 runs it only once. The fourth parameter receives the ID of the Alarm, which is necessary for terminating the Alarm.

btn_stop_date_cb() is a function that terminates a Date Alarm.

Run the example again. Tap the third Button and close the app, and the Alarm example is run again.

This time, tap the third Button and then the fourth Button immediately after. Then, close the app, and you will see no changes happen, which means the Date Alarm has been terminated.



Timer Alarm Started

4) Related APIs

app_control_h: the AppControl structure.

int app_control_create(app_control_h *): an API that creates an AppControl object.

int app_control_set_operation(app_control_h, char *): an API that specifies the role of AppControl. To load an external app, pass APP_CONTROL_OPERATION_DEFAULT.

int app_control_set_app_id(app_control_h, char *): an API that specifies the package name of the example to be loaded to AppControl. The package name of the HelloWorld example is "org.example.helloworld."

int alarm_schedule_after_delay(app_control_h, int, int, int *): an API that passes an AppControl after a certain amount of time has passed. The second parameter indicates the time interval for the first execution, while the third parameter indicates the time interval for re-executions. Entering 0 runs the AppControl event only once. The fourth parameter receives the ID of the Alarm, which is necessary for terminating the Alarm.

int alarm_cancel(int alarm_id): an API that terminates an Alarm. It passes the ID of an Alarm.

struct tm: a time structure where the time and date can be saved.

int alarm_get_current_time(struct tm *): an API that returns the current time in struct tm type.

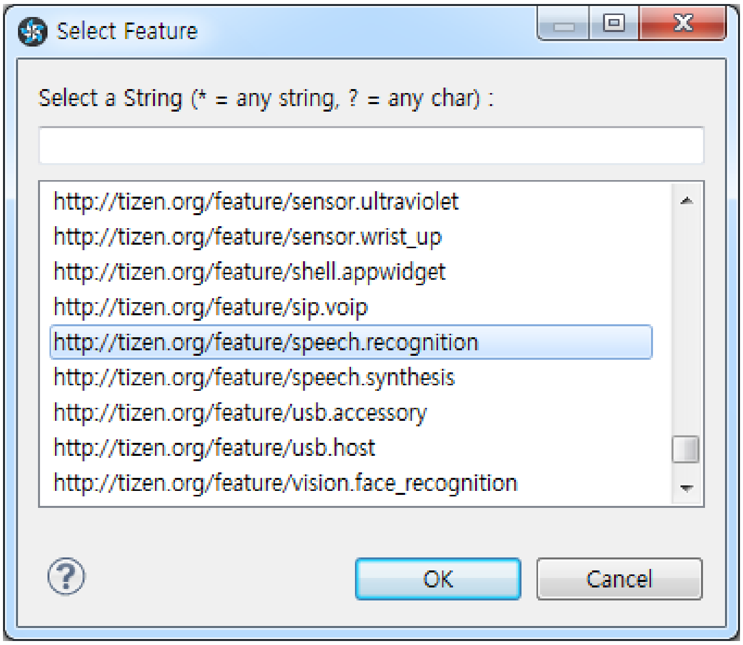
int alarm_schedule_at_date(app_control_h, struct tm *, int, int *): an API that runs an AppControl event at a specified time.Pass the struct tm object that contains the time and date to the second parameter, and pass the re-execution interval to the third parameter. Passing 0 runs it only once. The fourth parameter receives the ID of the Alarm, which is necessary for terminating the Alarm.

65. Text to Speech (TTS)

Text to Speech (TTS) is a feature that reads text aloud in a human voice. Speech to Text (STT), meanwhile, is a feature that recognizes a person's voice and reproduces it as text. In this example, we are going to learn how to use TTS.

1) Adding a Feature

Create a new source project and specify the project name as 'TtsEx.' To use TTS, you need to add relevant features. After the source project is created, open the tizen-manifest.xml file, and click Features among the tab buttons below. Then, click the Add button in the upper right corner. When a popup window appears, select http://tizen.org/feature/speech.recognition from the list, and click the OK button to close the window.



Select Feature

Select a String (* = any string, ? = any char) :

http://tizen.org/feature/sensor.ultraviolet
http://tizen.org/feature/sensor.wrist_up
http://tizen.org/feature/shell.appwidget
http://tizen.org/feature/sip.voip
http://tizen.org/feature/speech.recognition
http://tizen.org/feature/speech.synthesis
http://tizen.org/feature/usb.accessory
http://tizen.org/feature/usb.host
http://tizen.org/feature/vision.face_recognition

Add the following feature by repeating the procedure above.

- http://tizen.org/feature/speech.synthesis

After saving, click the tizen-manifest.xml button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<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) Creating & Deleting TTS

Open the source file (~.c) under the src folder and add library header files and variables.

```
#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 is a library header file for using TTS.

We are going to make the Entry widget receive a string input.

We are going to make it so that the string starts playing when the Button widget is tapped.

tts_h is the TTS structure.

We are now going to implement a feature that automatically creates a TTS object when the app is launched and deletes the TTS object when the app is closed. Add five new functions on top of the create_base_gui() function.

```
static void
state_changed_cb(tts_h tts, tts_state_e previous, tts_state_e current, void* user_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() is a TTS status change event function. The parameters listed in order are a TTS object, previous status value, current status value, and user data. This function converts the status value into a string and displays it on the screen.

tts_state_e is an INT variable that stores TTS status information. The status types are as follows:
- TTS_STATE_CREATED: TTS created.
- TTS_STATE_READY: TTS ready to play.
- TTS_STATE_PLAYING: TTS playing.
- TTS_STATE_PAUSED: TTS paused.

utterance_completed_cb() is a TTS playing completion event function.

utterance_started_cb() is a TTS playing start event function.

create_tts_handle() is a function that creates and then returns a TTS object.

tts_create(tts_h*) is an API that creates a TTS object.

tts_set_utterance_started_cb() is an API that specifies the name of the TTS playing start event callback function.

tts_set_utterance_completed_cb() is an API that specifies the name of the TTS playing completion event callback function.

tts_set_state_changed_cb(tts_h, tts_state_changed_cb, void*) is an API that specifies the name of the TTS status change event callback function. The parameters listed in order are a TTS object, name of the callback function, and user data.

tts_prepare(tts_h) is an API that changes the status of a TTS object to Ready.

destroy_tts_handle() is a function that deletes a TTS object.

tts_destroy(tts_h) is an API that deletes a TTS object.

The functions above need to be called when the app is launched and closed.

Add two new functions on top of the create_base_gui() function.

```
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_weight,
         double h_align, double v_align)
```

```
{
        /* create a frame we shall use as padding around the child widget */
evas_object_size_hint_align_set(frame, h_align, v_align);
{
evas_object_show(frame);
}
```

btn_play_cb() is the Button event function. We will define the content of this function later.

my_box_pack() is a function that adds a widget to a Box container.

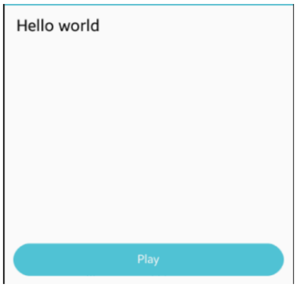
Add new code to the create_base_gui() function. This code creates a Box, an Entry, and a Button. Next, call the function that creates a TTS object.

```
    /* 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_EX
PAND);
        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_EXPAN
D, EVAS_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, EV
AS_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);
}
```

Add new code to the app_terminate() function. When the app is closed, the TTS object will also be deleted.

```
static void
app_terminate(void *data)
{
    appdata_s *ad = data;
    destroy_tts_handle(ad->tts);
}
```

Build and run the example. When a TTS object is created and its status becomes Ready, the caption text of the Button changes to 'Play.' Now, it is ready to play.



3) Converting a String to Voice

In this section, we are going to implement a feature that, when a string is entered in an Entry widget and a Button is tapped, outputs the string as voice.

Add two new functions on top of the create_base_gui() function. Then, fill out content in the Button event function.

```
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_STATE_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() is a function that adds a string to a TTS object.

tts_add_text(tts_h, char*,char*, int, int, int*) is an API that adds a string to a TTS object. The parameters listed in order are a TTS object, string, voice type, and speed.

The voice types are as follows:
- TTS_VOICE_TYPE_AUTO: auto voice type.
- TTS_VOICE_TYPE_MALE: male voice.
- TTS_VOICE_TYPE_FEMALE: female voice.
- TTS_VOICE_TYPE_CHILD: child's voice.

get_state() is a function that returns the current status of TTS.

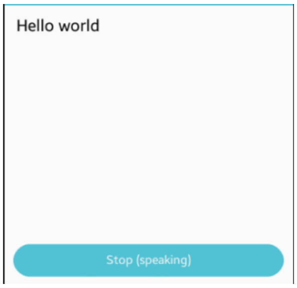
tts_get_state(tts_h tts, tts_state_e* state) is an API that returns the current status of TTS.

btn_play_cb() is a callback function that starts TTS when a Button is tapped. This function starts playing TTS when the current status of TTS is Ready or Paused, and stops TTS when it is being played.

tts_play(tts_h tts) is an API that starts the playback of TTS.

tts_stop(tts_h tts) is an API that stops the playback of TTS.

Run the example again. Tap the Button, and TTS will start playing. Tap the Button again, and TTS will stop playing. Tap the Button once again, and TTS will start playing again.


Hello world

4) Related APIs

int tts_create(tts_h*): an API that creates a TTS object.

int tts_set_state_changed_cb(tts_h tts, tts_state_changed_cb callback, void* user_data): an API that specifies the name of the TTS status change event callback function. The parameters listed in order are a TTS object, name of the callback function, and user data.

int tts_prepare(tts_h tts): an API that changes the status of a TTS object to Ready.

int tts_destroy(tts_h tts): an API that deletes a TTS object.

int tts_add_text(tts_h tts, const char* text, const char* language, int voice_type, int speed, int* utt_id): an API that adds a string to a TTS object. The parameters listed in order are a TTS object, string, voice type, and speed. The voice types are as follows:
- TTS_VOICE_TYPE_AUTO: auto voice type.
- TTS_VOICE_TYPE_MALE: male voice.
- TTS_VOICE_TYPE_FEMALE: female voice.
- TTS_VOICE_TYPE_CHILD: child's voice.

int tts_get_state(tts_h tts, tts_state_e* state): an API that returns the current status of TTS.

int tts_play(tts_h tts): an API that starts the playback of TTS.

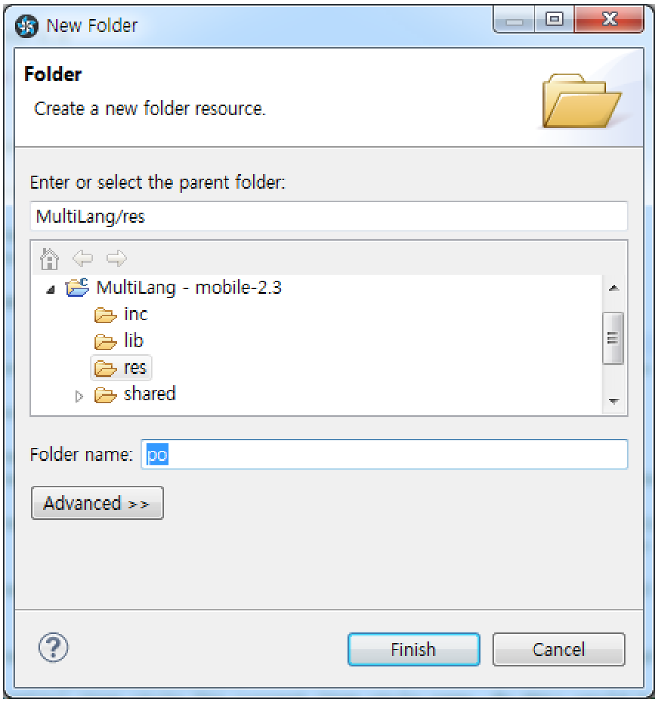
int tts_stop(tts_h tts): an API that stops the playback of TTS.

66. Multi-Language Support

With the opening of the App Store to individual developers, it has become possible for developers in Asia to sell their apps to European consumers. However, creating different language versions of an app for multi-language support can be very difficult and time-consuming. This problem is solved by adding language-specific texts that have the same meaning saved to Resources and calling the relevant text when necessary. In this example, we are going to learn how to store multi-language information in a PO file and call it from the source code.

1) Adding Multi-Language Information to Resources

Create a new source project and specify the project name as 'MultiLang.' Now that a source project has been created, we are now going to add multi-language information. Store multi-language information under the /res/po folder. Right-click the /res folder and select [New > Folder] in the shortcut menu. When a popup window appears, enter 'po' in the folder name field and click the Finish button.



New Folder

Folder

Create a new folder resource.

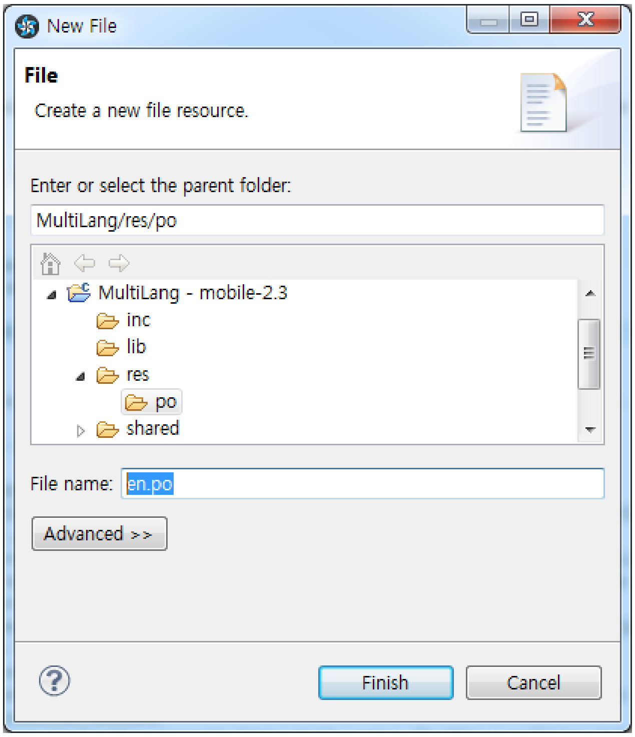
Enter or select the parent folder:

MultiLang/res

MultiLang - mobile-2.3
inc
lib
res
shared

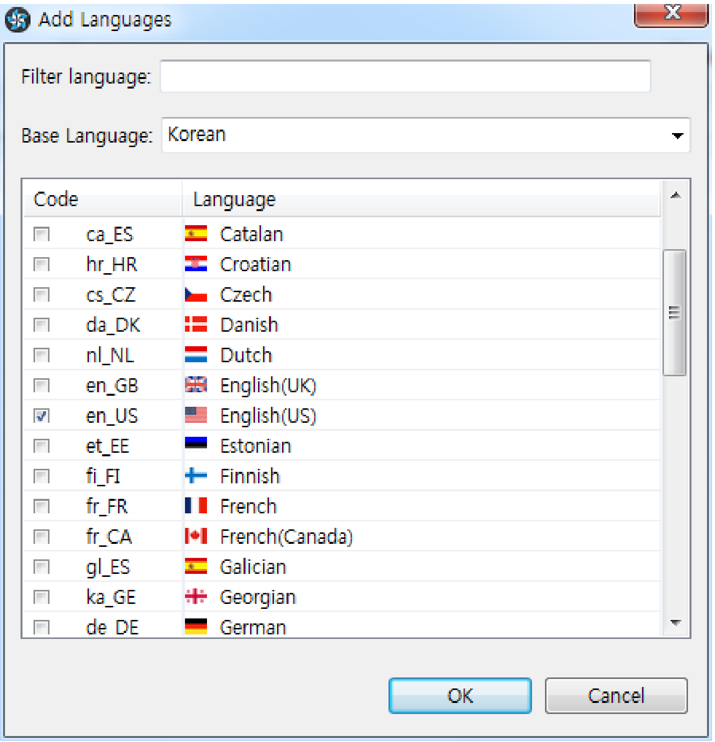
Folder name: po

A multi-language information file is not created automatically, making it necessary to create a multi-language information file manually. Right-click the /res/po folder and select [New > File] in the shortcut menu. When a popup window appears, enter 'en.po' in the file name field and click the Finish button.



en.po is a file that stores English-related information. Because most apps support English, this is an essential multi-language information file. As the number of supported languages increases, the number of po files also increases. When a po file is created, the content of the file is displayed in the Editor. If the Editor does not open automatically, double-click the en.po file.

When the Add Languages popup window appears, we will check several items (en_US (English), ko_KR (Korean), and ru_RU (Russian)) in the language list.

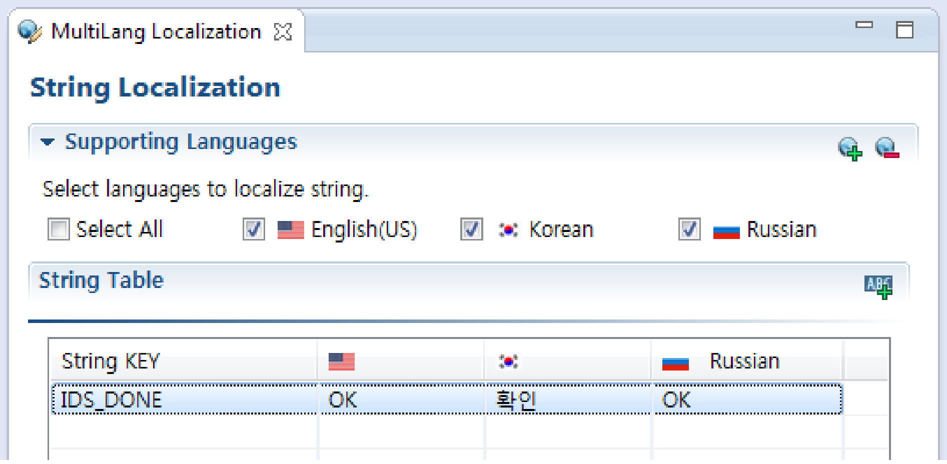


Filter language:

Base Language: Korean

Code	Language
□ ca_ES	Catalan
□ hr_HR	Croatian
□ cs_CZ	Czech
□ da_DK	Danish
□ nl_NL	Dutch
□ en_GB	English(UK)
☑ en_US	English(US)
□ et_EE	Estonian
□ fi_FI	Finnish
□ fr_FR	French
□ fr_CA	French(Canada)
□ gl_ES	Galician
□ ka_GE	Georgian
□ de_DE	German

We are now going to add a message. Click the 'Add String Key' button on the right of the screen and enter IDS_DONE for MsgID. Then, enter 'OK' in the English column field. Enter '확인' in the Korean field, and enter 'OK' in the Russian field.



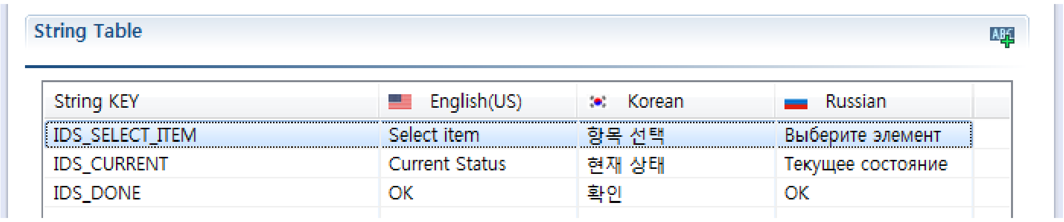
Select languages to localize string.

□ Select All ☑ English(US) ☑ Korean ☑ Russian

String KEY			Russian
IDS_DONE	OK	확인	OK

We are going to add one more message. Click the 'Add String Key' button and then enter the following text in the newly added item: IDS_CURRENT, Current Status, 현재 상태, and Текущее состояние.

Click the 'Add String Key' button once again and then enter the following text in the newly added item: IDS_SELECT_ITEM, Select item, 항목 선택, and Выберите элемент.



String Table

String KEY	English(US)	Korean	Russian
IDS_SELECT_ITEM	Select item	항목 선택	Выберите элемент
IDS_CURRENT	Current Status	현재 상태	Текущее состояние
IDS_DONE	OK	확인	OK

2) Requesting Multi-Language Information from Source Code

After the source project is created, open the source file (~.c) under the src folder and add a variable to the appdata structure.

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

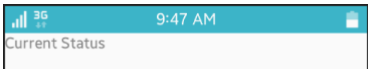
We are now going to apply the multi-language text relevant to IDS_CURRENT to the Label widget. Then, go to the create_base_gui()

function and modify the Label widget-creating code.

```
    /* 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_EXPA
ND);
    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) is an API that returns multi-language information which corresponds to the language in Settings.

Build and run the example. The multi-language information you added to the po file will be displayed on the screen.



Current Status

3) Applying Multi-Language to NaviFrame

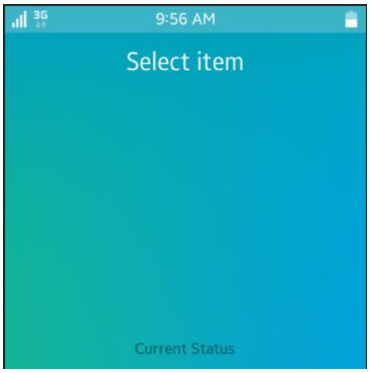
To apply multi-language to the title text of the header, you need to use the i18n_get_text() function. You need to use a different method to make the change be applied automatically when you have changed the language in Settings. Add NaviFrame- and Box-creating code to the create_base_gui() function.

```
    /* 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);

    /* 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_HINT_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);
```

We created a Naviframe and a Box container and specified the title text as "IDS_SELECT_ITEM." Doing so displays the title as "IDS_SELECT_ITEM." Therefore, something must be added.

elm_object_item_part_text_translatable_set(it, part, translatable) is an API that makes it so that the text of an object is requested from Resources. If the third parameter is EINA_TRUE, then the text is received from Resources. If it is EINA_FALSE, the text entered is used as it is.

Run the example again. A header will be displayed, with multi-languages applied to the title.



4) Applying Multi-Language to a Widget

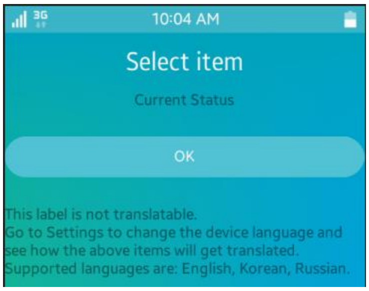
You need to use a different method to apply multi-language to widgets such as Label, Button, and Entry widgets. Add Button- and Label-creating code to the create_base_gui() function.

```
    {
        /* 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) is an API that automatically applies the text, which is registered as Resources in a widget. When the user changes the language in Settings, the new multi-language is automatically applied to a widget's text.

The second Label widget was used to display the text phrase to the user; it has no other particular functions.

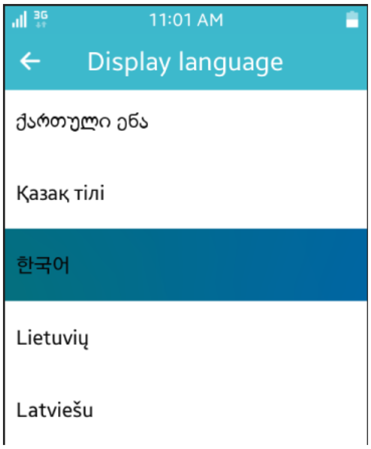
Run the example again. The new language has been applied to the Button widget.



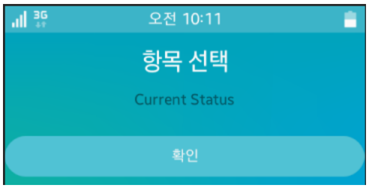
This label is not translatable.
Go to Settings to change the device language and see how the above items will get translated.
Supported languages are: English, Korean, Russian.

5) Changing the Language Automatically

The default language for the emulator is English (United States). We are going to change it to another language. Start the MultiLang app, tap the Back button, and then go to the App List screen. Tap the Settings icon and select 'Language and input' from the list. When the screen changes, select 'Display language.' When the Language list appears, select one language between Korean and Russian.



Exit the Settings screen and run the MultiLang example again. Press and hold the Home button, and a list of apps run so far will appear. Select MultiLang from among the apps. When the example changes to Foreground mode, the new language is automatically applied to the NaviFrame and Button. The language of the Label widget has not been changed from English. We need to manually set multi-language again by requesting an event that changes the language in Settings.



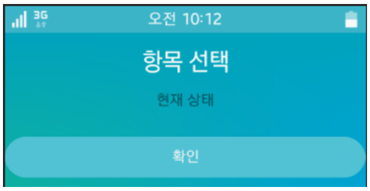
Go to the bottom of the source file, and there you will see a function named ui_app_lang_changed(). Add new code at the end of this function.

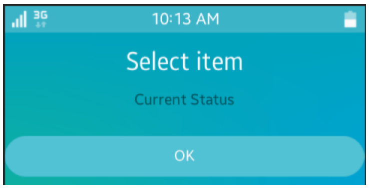
```
static void
ui_app_lang_changed(app_event_info_h event_info, void *user_data)
{
    /*APP_EVENT_LANGUAGE_CHANGED*/
    char *locale = NULL;
    system_settings_get_value_string(SYSTEM_SETTINGS_KEY_LOCALE_LANGUAGE, &local
e);
    elm_language_set(locale);
    free(locale);

    appdata_s* ad = user_data;
    elm_object_text_set(ad->label, i18n_get_text("IDS_CURRENT"));
}
```

ui_app_lang_changed() is an event function which is run when the language in Settings is changed. To change the name of the function, modify it from the main() function.

We have set multi-language again for the Label widget. Doing so makes multi-language automatically reflected in widgets when the user changes the language in Settings.

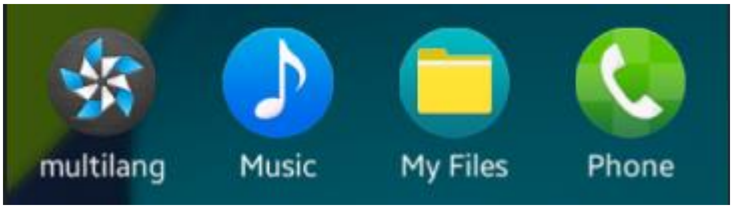
Run the example again and change the language in Settings to English (United States). Afterwards, go back to the MultiLang example, and you will now see the new language is automatically applied.





6) Changing Text of an App Icon

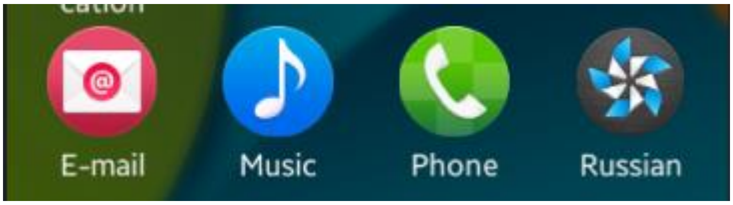
Go to the App icon list screen by tapping the Back button, and you will see the icon for the MultiLang example. The Label text is displayed as 'multilang.' We are going to learn how to apply multi-language to it.



Open the tizen-manifest.xml file and click the tizen-manifest.xml button among the tab buttons below, and you will see its source code. The code <label>multilang</label> displayed is that Label text of the app icon. Add new XML code.

```
<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>
```

Enter a text phrase for each language using the label tag to apply multi-language. Run the example again, and change the language in Settings to Korean or Russian. Then, go back to the App Icon List screen, and you will see the text of the app icon has changed to the chosen language.



7) Related APIs

char* i18n_get_text(const char *message): an API that returns multi-language information which corresponds to the language in Settings.

void elm_object_translatable_text_set(obj, text): an API that applies text registered in Resources to a widget. When the user changes the language in Settings, the new language is automatically applied to a widget's text.

void elm_object_item_part_text_translatable_set(it, part, translatable): an API that makes it so that the text of an object can be requested from Resources. If the third parameter is EINA_TRUE, then the text is received from Resources. If it is EINA_FALSE, the default text is used as it is.

void ui_app_lang_changed(): an event function which is run when the language in Settings is changed. To change the name of the function, modify it from the main() function.

67. JSON Parsing

Like XML, JSON (JavaScript Object Notation) is a protocol used for data transmission. As the name implies, it is commonly used in JavaScript, but because of its better ease of use than XML, it is now widely used in web based communications. For example, the following XML statement and JSON statement have the same content, but you can see that the JSON statement is much simpler. This is because JSON statements include only the necessary information. In this example, we are going to learn how to parse JSON data.
XML format - <name>Obama</name><math>50</math>
JSON format - [name:Obama, math:50]

1) Requesting JSON Array Data

Create a new source project and specify the project name as 'JsonParse.' Open the source file (~.c) under the src folder and add a library header file.

```
#include  "jsonparse.h"
#include  <json-glib/json-glib.h>
```

json-glib/json-glib.h is a library header file for JSON parsing.

JSON statements save arrays in the '[]' symbol. We are now going to extract data from an array one by one. Add a new function on top of the create_base_gui() function.

```
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 is a structure used for parsing JSON data.

json_parser_new() is an API that creates a JsonParser object.

We saved a JSON statement in a string variable named data1. This is an array that has five numeric items.

json_parser_load_from_data(JsonParser*, gchar*, gssize, GError**) is an API that enters a JSON statement in a JsonParser object. The parameters listed in order are a JsonParser object, JSON data, data length, and error code.

json_parser_get_root(JsonParser*) is an API that returns the start position of JsonParser. The return format is JsonNode.

json_node_get_array(JsonNode*) is an API that returns a Json statement as an array. The return format is JsonArray.

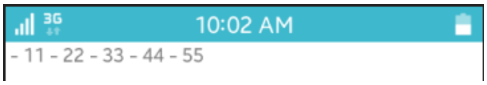
json_array_get_length(JsonArray*) is an API that returns the number of items in a Json array. The return format is guint.

json_array_get_int_element(JsonArray*, guint) is an API that converts a certain item of a Json array into an integer and returns it. The return format is gint64.

When the example is run, the function above is automatically run. Add a new line of code at the end of the create_base_gui() function.

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

   parse_json(ad);
}
```

We are now going to build and run the example. Five saved pieces of data in a Json array are displayed in the Label widget.



- 11 - 22 - 33 - 44 - 55

2) Requesting Data with a Key

Generally, Json statements are in the following form. name is the Key, and Obama is the data.

{name:Obama, math:50}

We are now going to learn how to request data using the value of a Key. Add new code to the parse_json() function.

```
     ~
   const char* data2 = "{'time': '03:53:25 AM', 'millisec_epoch': 1362196405309, '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, "millisec_epoch");
      sprintf(buf, "%s <br/><br/> time - %s <br/> epoch - %lld", buf, time_data, epoch_data);
   }

   elm_object_text_set(ad->label, buf);
}
```

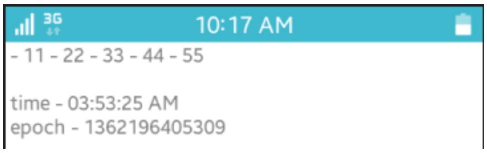
We saved a Json statement in a string variable named data2, and we also entered the Json statement in a JsonParser object using the json_parser_load_from_data() function.

json_node_get_object(JsonNode*) is an API that returns a Json statement in JsonObject form.

json_object_get_string_member(JsonObject*, gchar*) is an API that makes JsonObject return data in string format. Passing the Key to the second parameter sees data returned in gchar* format.

json_object_get_int_member(JsonObject*, gchar*) is an API that makes JsonObject return data in numerical form. Passing the Key to the second parameter sees data returned in gint64 format.

Run the example again. Data for the time item and data for the epoch item are displayed on the screen.



```
- 11 - 22 - 33 - 44 - 55

time - 03:53:25 AM
epoch - 1362196405309
```

3) Multi-Level Parsing

Looking at the following Json statement, you can see that coord includes a Json object and a Json array. In this section, we are going to learn how to parse complex Json statements like that.

{'coord':{'lon':127.03, 'lat':37.5}, 'weather':[{'id':800, 'main':'Clear'}, {'id':887, 'main':'Cloudy'}]}

Add new code at the end of the parse_json() function.

This code requests data from a Json object included in a Json statement.

```
~
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"));
}

elm_object_text_set(ad->label, buf);
```

We saved a Json statement in a string variable named data3, and we also entered the Json statement in a JsonParser object using the json_parser_load_from_data() function.

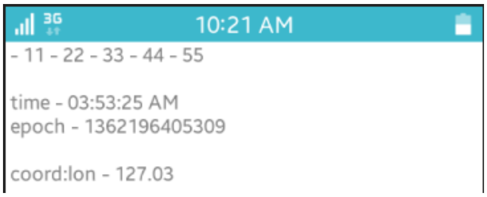
We requested the Json node at the start point using the json_parser_get_root() function, and we are now going to convert the Json statement into JsonObject form using the json_node_get_object() function.

json_object_get_member(JsonObject *, gchar*) is an API that returns a certain JsonObject. The parameters listed in order are a JsonParser and member name. The return format is JsonNode.

We requested JsonObject from JsonNode using the json_node_get_object() function.

json_object_get_double_member(JsonObject *, gchar*) is an API that makes JsonObject return data in the form of real numbers. Passing the Key to the second parameter sees data returned in gdouble format.

Run the example again. The screen displays the data corresponding to the lon Key in the coord group.



```
- 11 - 22 - 33 - 44 - 55

time - 03:53:25 AM
epoch - 1362196405309

coord:lon - 127.03
```

4) Parsing Data from an Inner Group of an Array

In this section, we are going to request the value that corresponds to 'id' in the first item of an array called 'weather' in the same Json statement. Add new code to the parse_json() function.

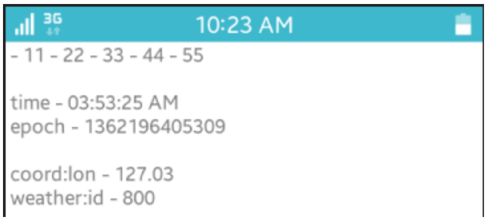
```
   const char* data3 = "{'coord':{'lon':127.03, 'lat':37.5}, 'weather':[{'id':800, 'main':'Clea
r'},
       {'id':887, 'main':'Cloudy'}]}";
   if( json_parser_load_from_data( parser, data3, strlen(data3), NULL))
   {
       JsonNode *root = json_parser_get_root (parser);
       JsonObject *obj = 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_int_member
           (temp_object, "id"));
   }

   elm_object_text_set(ad->label, buf);
```

Request an array node called 'weather' using the json_object_get_member() function and convert the array node into JsonArray form using the json_node_get_array() function.

Request the node for the first item using the json_array_get_element() function and convert it into JsonObject form using the json_node_get_object() function.

Lastly, convert the data corresponding to a Key named 'id' into an integer using the json_object_get_int_member() function.

Run the example again. The value corresponding to the 'id' of the first array item will be displayed on the screen.



```
- 11 - 22 - 33 - 44 - 55

time - 03:53:25 AM
epoch - 1362196405309

coord:lon - 127.03
weather:id - 800
```

5) Related APIs

JsonParser *json_parser_new(): an API that creates a JsonParser object.

gboolean json_parser_load_from_data(JsonParser*, gchar*, gssize, GError**): an API that enters a JSON statement in a JsonParser object. The parameters listed in order are a JsonParser object, JSON data, data length, and error code.

JsonNode* json_parser_get_root(JsonParser*): an API that returns the start position of JsonParser. The return format is JsonNode.

JsonArray* json_node_get_array(JsonNode*): an API that returns a Jason statement in array form. The return format is JsonArray.

guint json_array_get_length(JsonArray*): an API that returns the number of items in a Json array. The return format is guint.

gint64 json_array_get_int_element(JsonArray*, guint): an API that converts a certain item of a Json array into an integer and returns the integer. The return format is gint64.

JsonObject* json_node_get_object(JsonNode*): an API that returns a Jason statement in JsonObject form.

gchar* json_object_get_string_member(JsonObject*, gchar*): an API that makes JsonObject return data in a string form. Passing the Key to the second parameter sees data returned in gchar* format.

gint64 json_object_get_int_member(JsonObject*, gchar*): an API that makes JsonObject return data in numeric form. Passing the Key to the second parameter sees data returned in gint64 format.

JsonNode* json_object_get_member(JsonObject *, gchar*): an API that returns a certain JsonObject. The parameters listed in order are a JsonParser and member name. The return format is JsonNode.

68. XML Parsing

When a mobile app communicates with a service via HTTP, data is transmitted in XML form. XML is a document protocol for systemically storing data, and it is widely used not only for communications but also for storing data necessary for developing apps, such as screen layouts and resource data. In this example, we are going to learn how to parse XML data.

1) Requesting XML Data

Create a new source project and specify the project name as 'XmlParse.' Open the source file (~.c) under the src folder and add library header files and variables.

```
#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 is a library header file for XML parsing.

value_begin is a variable for storing information on whether to start data.

value_type is a variable for storing the data type.

buffer[] is a string array variable for storing parsed data.

The following is a simple XML statement. name indicates the name of the node, and Elsa indicates data. We are now going to parse this.

<name>Elsa</name><math>95</math>

Add a new function on top of the create_base_gui() function. This function adds a widget to a Box container.

```
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");

evas_object_show(frame);
}
```

Then, add Box- and Button-creating code to the create_base_gui() function.

```
    /* 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);

    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);
}
```

Add two new functions on top of the create_base_gui() function.

```
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 | HTML_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() is a function that extracts data corresponding to "name" and "math" in the xmlNode object and adds the data to a global variable.

Request the individual nodes included in the XML statement from the for-loop, and afterwards, if the name of a given individual node is 'name' or 'math,' request the data. In order to move on to the next node, the function calls itself.

xmlNode is a structure variable that points to a certain point in an XML statement. The properties are as follows:
 - next: returns the pointer of the next node.
 - name: the name of the node If 'text' is saved in this property, it means that the node is data. In such cases, data can be requested from the content property.
 - content: node data.
 - properties: node properties data.

btn_parse1_cb() is a function that creates an XML parser object, enters an XML statement, and then calls the function that runs parsing.

htmlCreatePushParserCtxt(htmlSAXHandlerPtr sax, void *user_data, char *chunk, int size, char *filename, xmlCharEncoding enc) is an API that creates an XML parser object.

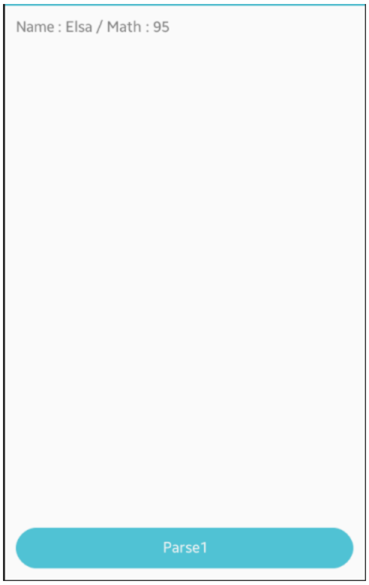
htmlParserCtxtPtr is the XML parser structure. Among the properties, myDoc contains an XML statement in xmlDocPtr form.

htmlCtxtUseOptions(htmlParserCtxtPtr ctxt, int options) is an API that specifies an option for an XML parser. This option can be specified redundantly.

htmlParseChunk(htmlParserCtxtPtr ctxt, char *chunk, int size, int terminate) is an API that enters an XML statement in a XML parser.

xmlDocGetRootElement(xmlDocPtr doc) is an API that returns the start node of an XML statement.

We are now going to build and run the example. Tap the Button, and the name node's data and the math node's data will be displayed on the screen.



Name : Elsa / Math : 95

2) Requesting Node Properties Data

As shown below, XML lets you specify properties for a node. In this section, we are going to learn how to request node properties data.

<student name="Aurora" math="27"></student>

Add code that creates a second Button at the end of the create_base_gui() function.

```
        /* 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);
    }
```

Add two new functions on top of the create_base_gui() function.

```
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_attr->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> <name>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 | HTML_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() is a function that requests the properties data of an XML node and displays it on the screen.

Request the individual nodes included in the XML statement from the first for-loop and request individual properties from the second for-loop. Then, if the name of a given individual node is 'name' or 'math,' request data. In order to move on to the next node, the function calls itself.

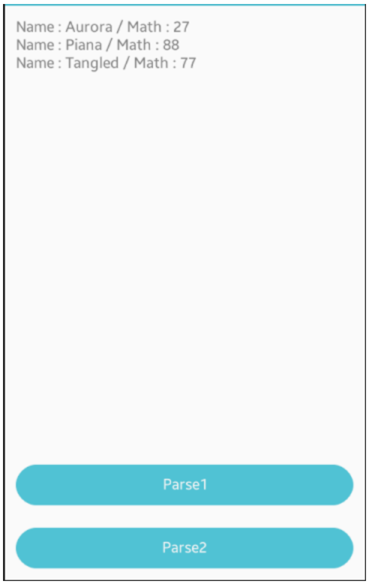
Among the properties of xmlNode, the 'properties' property contains the properties of the node in xmlAttr form.

xmlGetProp(xmlNodePtr node, const xmlChar *name) is an API that returns node properties data. Pass the XML node to the first parameter, and pass the name of the property to the second parameter.

xmlFree(xmlChar*) is an API that deletes XML data.

btn_parse2_cb() is a function that creates an XML parser object, enters an XML statement, and then calls the function that runs properties data parsing.

Run the example again and tap the second Button. Node properties data are displayed on the screen.



Name : Aurora / Math : 27
Name : Piana / Math : 88
Name : Tangled / Math : 77

3) Multi-Level Node Parsing

The following XML statement has a total of three <student> nodes, and each <student> includes a <name> node and a <math> node. We are now going to learn how to access nodes using a tree structure.

```
<student><name>Obama</name><math>50</math></student>
<student><name>Psy</name><math>70</math></student>
<student><name>Yuna</name><math>65</math></student>
```

Add code that creates a third Button at the end of the create_base_gui() function.

```
    /* 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);
    }
```

Add two new functions on top of the create_base_gui() function.

```
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_begin)
            ad->value_type = 1;
        if(!strcmp((const char*)cur_node->name, "math") && ad->value_begin)
            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> <name>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</math></student> </grade>";

    htmlParserCtxtPtr parser = htmlCreatePushParserCtxt(NULL, NULL, NULL, 0, NULL, 0);
    htmlCtxtUseOptions(parser, HTML_PARSE_NOBLANKS | HTML_PARSE_NOERROR | HTML_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() is a function that requests the data of the second node in an XML tree structure and displays it on the screen.

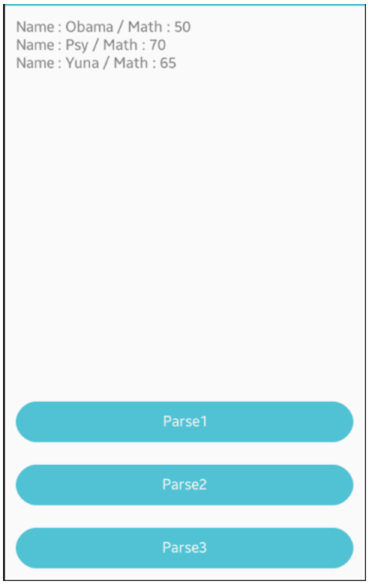
This function extracts nodes from an XML statement one by one. If the name of the node is 'student,' this function determines the node as the first node and sets the global variable 'value_begin' to true.

If the name of the node is 'name' or 'math,' and the global variable 'value_begin' is true, this function determines the node as the second node and calls itself.

If the name of the node is 'text,' it determines the node as data and then reads the value of its content property and adds the value to the global variable. In order to move on to the next node, the function calls itself.

btn_parse3_cb() is a function that creates an XML parser object, enters an XML statement, and then calls the function that runs parsing of the second node's data.

Run the example again and tap the third Button. The second node's data will be displayed on the screen.



Name : Obama / Math : 50
Name : Psy / Math : 70
Name : Yuna / Math : 65

4) Related APIs

htmlParserCtxtPtr htmlCreatePushParserCtxt(htmlSAXHandlerPtr sax, void *user_data, char *chunk, int size, char *filename, xmlCharEncoding enc): an API that creates an XML parser object.

htmlParserCtxtPtr: the XML parser structure. Among the properties, myDoc contains an XML statement in xmlDocPtr form.

int htmlCtxtUseOptions(htmlParserCtxtPtr ctxt, int options): an API that specifies an option for an XML parser. This option can be specified redundantly.

int htmlParseChunk(htmlParserCtxtPtr ctxt, char *chunk, int size, int terminate): an API that enters an XML statement in an XML parser.

xmlNodePtr xmlDocGetRootElement(xmlDocPtr doc): an API that returns the start node of an XML statement.

xmlChar* xmlGetProp(xmlNodePtr node, const xmlChar *name): an API that returns node properties data. Pass the XML node to the first parameter, and pass the name of the property to the second parameter.

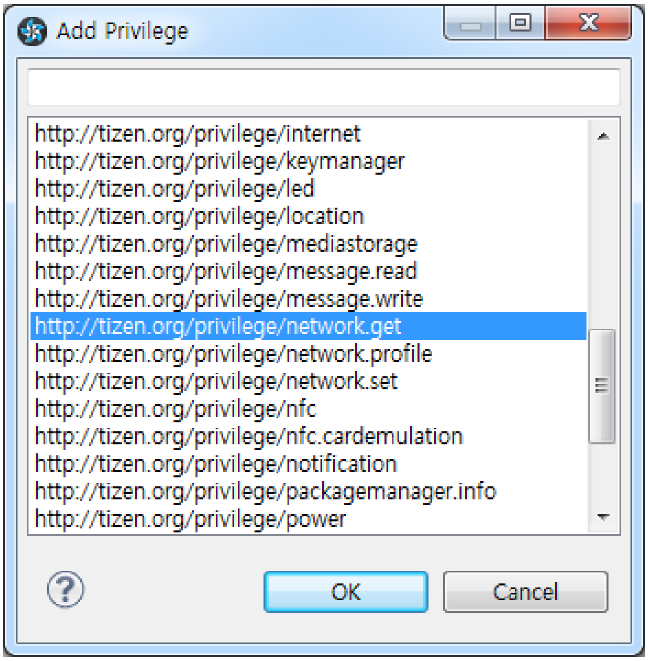
void xmlFree(xmlChar*): an API that deletes XML data.

69. Checking Network Status

It is advisable to check the status of your network before communicating with a server. In this example, we are going to learn how to check the availability of communication, mobile communication, and Wi-Fi communication.

1) Registering a Privilege

Create a new source project and specify the project name as 'NetConnection.' You need to have the applicable user privileges to check communication status. After the source project is created, open the tizen-manifest.xml file, and click Privileges among the tab buttons below. Then, click the Add button in the upper right corner. When a popup window appears, select http://tizen.org/privilege/network.get from the list, and click the OK button to close the window.



http://tizen.org/privilege/internet
http://tizen.org/privilege/keymanager
http://tizen.org/privilege/led
http://tizen.org/privilege/location
http://tizen.org/privilege/mediastorage
http://tizen.org/privilege/message.read
http://tizen.org/privilege/message.write
http://tizen.org/privilege/network.get
http://tizen.org/privilege/network.profile
http://tizen.org/privilege/network.set
http://tizen.org/privilege/nfc
http://tizen.org/privilege/nfc.cardemulation
http://tizen.org/privilege/notification
http://tizen.org/privilege/packagemanager.info
http://tizen.org/privilege/power

After saving, click the 'tizen-manifest.xml' button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<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) Checking Connection Status

In this section, we are going to check whether there is a network connection. Open the source file (~.c) under the src folder and add library header files and variables.

```
#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 is a library header file for checking connection status.

We will display connection status in label1, mobile connection status in label2, and Wi-Fi connection status in label3.

connection_h is the communication information structure.

Add a new function on top of the create_base_gui() function. This function checks connection status and displays it on the screen.

```
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) is an API that creates a connection_h object.

connection_get_type(connection_h connection, connection_type_e* type) is an API that returns current communication status. The return format is connection_type_e. The types of connection_type_e are as follows:
- CONNECTION_TYPE_DISCONNECTED: communication disconnected.
- CONNECTION_TYPE_WIFI: Wi-Fi type.
- CONNECTION_TYPE_CELLULAR: mobile communication type.
- CONNECTION_TYPE_ETHERNET: Ethernet type.
- CONNECTION_TYPE_BT: Bluetooth type.

Modify the code of the create_base_gui() function. This code creates Frame, Box, and Label widgets and calls the function above.

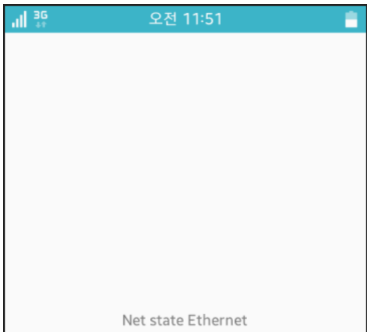
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win,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

This code deletes the connection_h object when the app is closed. Add new code to the app_terminate() function.

```
static void
app_terminate(void *data)
{
   appdata_s *ad = data;
   connection_destroy(ad->connection);
}
```

connection_destroy(connection_h connection) is an API that deletes a connection_h object.

Build and run the example. The type of communication is displayed in the Label widget.



Net state Ethernet

2) Requesting Mobile Connection Status

In this section, we are going to request the connection status of mobile communication, which is a charged communication. Add a new function on top of the create_base_gui() function.

```
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;
    }
}
```

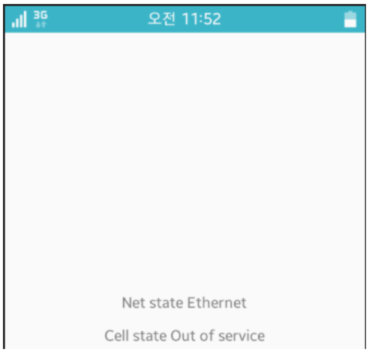
connection_get_cellular_state(connection_h connection, connection_cellular_state_e* state) is an API that returns mobile connection status. The return format is connection_cellular_state_e. The types of connection_cellular_state_e are as follows:

- CONNECTION_CELLULAR_STATE_OUT_OF_SERVICE: disconnected.
- CONNECTION_CELLULAR_STATE_FLIGHT_MODE: flight mode.
- CONNECTION_CELLULAR_STATE_ROAMING_OFF: roaming off.
- CONNECTION_CELLULAR_STATE_CALL_ONLY_AVAILABLE: only calls available.
- CONNECTION_CELLULAR_STATE_AVAILABLE: connection available but not yet connected.
- CONNECTION_CELLULAR_STATE_CONNECTED: connected.

We are now going to add a new Label widget and display the result of the function above on the screen. Add new code to the create_base_gui() function.

```
        /* 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);
    }
}
```

Run the example again. The mobile connection status will be displayed in the second Label widget.



Net state Ethernet

Cell state Out of service

3) Requesting WiFi Status

In this section, we are going to request WiFi connection status. Add a new function on top of the create_base_gui() function.

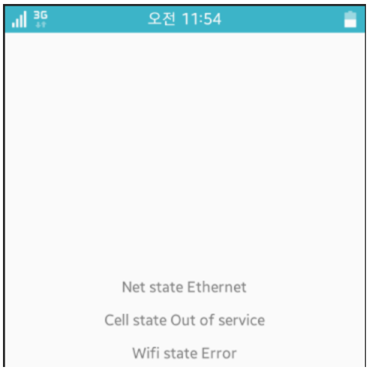
connection_get_wifi_state(connection_h connection, connection_wifi_state_e* state) is an API that returns the Wi-Fi connection status. The return format is connection_wifi_state_e. The types of connection_wifi_state_e are as follows:

- CONNECTION_WIFI_STATE_DEACTIVATED: Wi-Fi deactivated.
- CONNECTION_WIFI_STATE_DISCONNECTED: Wi-Fi disconnected.
- CONNECTION_WIFI_STATE_CONNECTED: Wi-Fi connected.

We are now going to add a new Label widget and display the result of the function above on the screen. Add new code to the create_base_gui() function.

```
        /* 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);
    }
}
```

Run the example again. The current Wi-Fi connection status will be displayed in the third Label widget.



Net state Ethernet

Cell state Out of service

Wifi state Error

4) Related APIs

int connection_create(connection_h* connection) is an API that creates a connection_h object.

int connection_get_type(connection_h connection, connection_type_e* type) is an API that returns the current communication status. The return format is connection_type_e. The types of connection_type_e are as follows:

- CONNECTION_TYPE_DISCONNECTED: disconnected.

- CONNECTION_TYPE_WIFI: Wi-Fi type

- CONNECTION_TYPE_CELLULAR: mobile communication type.

- CONNECTION_TYPE_ETHERNET: Ethernet type.

- CONNECTION_TYPE_BT: Bluetooth type.

int connection_destroy(connection_h connection) is an API that deletes a connection_h object.

int connection_get_cellular_state(connection_h connection, connection_cellular_state_e* state) is an API that returns mobile communication status. The return format is cconnection_cellular_state_e. The types of connection_cellular_state_e are as follows:

- CONNECTION_CELLULAR_STATE_OUT_OF_SERVICE: disconnected.

- CONNECTION_CELLULAR_STATE_FLIGHT_MODE: flight mode.

- CONNECTION_CELLULAR_STATE_ROAMING_OFF: roaming off.

- CONNECTION_CELLULAR_STATE_CALL_ONLY_AVAILABLE: only calls available.

- CONNECTION_CELLULAR_STATE_AVAILABLE: connection available but not yet connected.

- CONNECTION_CELLULAR_STATE_CONNECTED: connected.

int connection_get_wifi_state(connection_h connection, connection_wifi_state_e* state) is an API that returns the Wi-Fi communication status. The return format is connection_wifi_state_e. The types of connection_wifi_state_e are as follows:

- CONNECTION_WIFI_STATE_DEACTIVATED: Wi-Fi deactivated.

- CONNECTION_WIFI_STATE_DISCONNECTED: Wi-Fi disconnected.

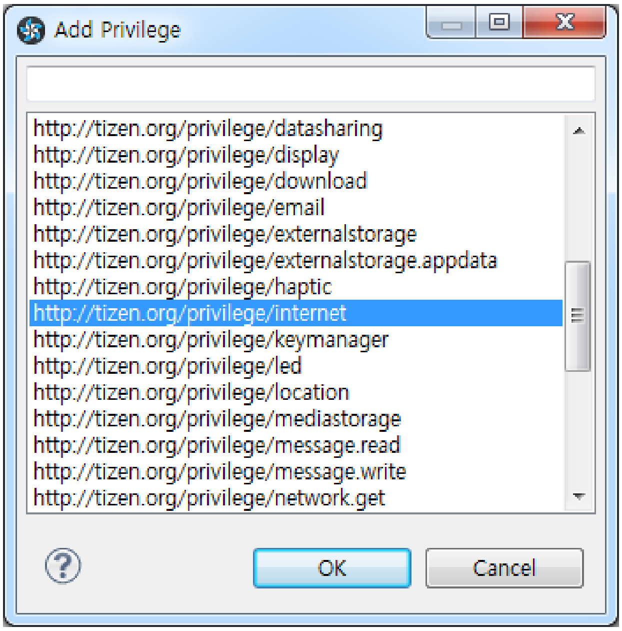
- CONNECTION_WIFI_STATE_CONNECTED: Wi-Fi connected.

70. HTTP Communication

HTTP communication is easy to use and capable of transferring large amounts of data at once. This property makes it attractive for use in mobile devices. In this example, we are going to implement a feature that requests weather information via HTTP communication and downloads images from a web server.

1) Registering a Privilege

Create a new source project and specify the project name as 'HttpRequest.' You need to have the applicable user privileges to use communication. After the source project is created, open the tizen-manifest.xml file, and click Privileges among the tab buttons below. Then, click the Add button in the upper right corner. When a popup window appears, select http://tizen.org/privilege/internet from the list, and click the OK button to close the window.



http://tizen.org/privilege/datasharing
http://tizen.org/privilege/display
http://tizen.org/privilege/download
http://tizen.org/privilege/email
http://tizen.org/privilege/externalstorage
http://tizen.org/privilege/externalstorage.appdata
http://tizen.org/privilege/haptic
http://tizen.org/privilege/internet
http://tizen.org/privilege/keymanager
http://tizen.org/privilege/led
http://tizen.org/privilege/location
http://tizen.org/privilege/mediastorage
http://tizen.org/privilege/message.read
http://tizen.org/privilege/message.write
http://tizen.org/privilege/network.get

After saving, click the 'tizen-manifest.xml' button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<?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" multiple="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) Requesting Text Communication Data

In this section, we are going to implement a feature to receive text data from a web server. Open the source file (~.c) under the src folder and add a library header, variables, and a structure.

```
#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;
```

For HTTP communication, the CURL library is used. curl/curl.h is a CURL library header file.

memoryStruct is a structure that stores communication result data.

icon is an Evas Image object.

We are now going to implement a feature that connects to a server and receives text data from it. Add three new functions on top of the create_base_gui() function.

```
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 returned 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 above */
   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=127.0
27&units=metric");
    get_http_data(url, &ad->ms);

    elm_object_text_set(ad->label, ad->ms.memory);
    free( ad->ms.memory);
}
```

write_memory_cb() is an event function that receives server responses. The parameters listed in order are the content, byte unit, memory size, and user data.

You can calculate the total memory size by multiplying the second parameter by the third parameter. The byte unit is mostly 1.

The next code assigns memory to the memory property of the memoryStruct structure using the realloc() function and copies data using the memcpy() function.

Then, it adds the size of memory to the size property of the memoryStruct structure and replaces the end of the data with 0 to mark the end.

get_http_data() is a function that attempts communication with the server.

curl_global_init(long flags) is an API that initializes the CURL library. For apps that use CURL, this API must be run once initially.

curl_easy_init(void) is an API that creates a CURL object.

curl_easy_setopt(CURL *curl, CURLoption option, ...) is an API that specifies an option for a CURL object. The option types are as follows:

- CURLOPT_URL: specifies a URL address.
- CURLOPT_WRITEFUNCTION: specifies a callback function that receives communication results.
- CURLOPT_WRITEDATA: specifies user data.
- CURLOPT_USERAGENT: specifies a user agent.

curl_easy_perform(CURL *curl) is an API that starts communication with a server.

curl_easy_cleanup(CURL *curl) is an API that deletes CURL data.

curl_global_cleanup(void) is an API that deletes the entire data of the CURL library. Once CURL is used, this API should be called at least once before closing the app.

btn_download_text() is a function that, when the user taps a Button, receives text data from a server and displays the result in a Label widgets.

Request weather information for a place with latitude coordinate 37.498 and longitude coordinate 127.027.

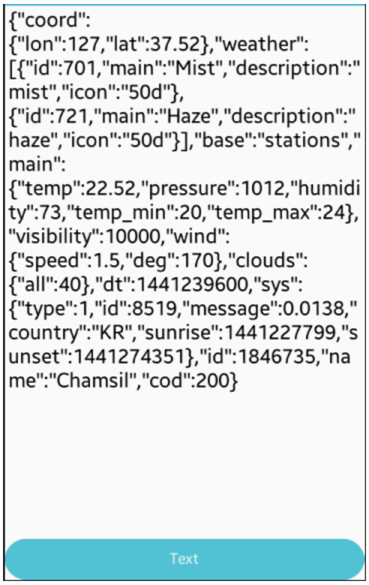
We are going to create a Button that receives data from the weather forecast server. Add new code at the end of the create_base_gui() function.

```
    /* 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_FILL);
        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);
}
```

We created Box, Entry, and Button widgets.

We are now going to build and run the example. Tap the Button and, when a response is received from the server, Json type weather information is displayed on the screen.



```
{"coord":
{"lon":127,"lat":37.52},"weather":
[{"id":701,"main":"Mist","description":"
mist","icon":"50d"},
{"id":721,"main":"Haze","description":"
haze","icon":"50d"}],"base":"stations","
main":
{"temp":22.52,"pressure":1012,"humidi
ty":73,"temp_min":20,"temp_max":24},
"visibility":10000,"wind":
{"speed":1.5,"deg":170},"clouds":
{"all":40},"dt":1441239600,"sys":
{"type":1,"id":8519,"message":0.0138,"
country":"KR","sunrise":1441227799,"s
unset":1441274351},"id":1846735,"na
me":"Chamsil","cod":200}
```

For information on how to extract individual pieces of data from a Json statement, see the JsonParse example.

3) Downloading Images

In this section, we are going to implement a feature that downloads images from a web server and displays them in an Image object. Add Image object- and Button widget-creating code to the create_base_gui() function.

```
    {
        /* 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_FILL);
        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_FILL);
        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);
    }
```

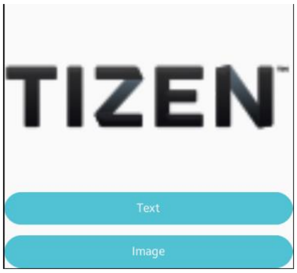
We created an Image object and saved it in the icon variable of the appdata structure. Then, we created a second Button and specified the name of the callback function as btn_download_image.

Lastly, we need to create a Button callback function. Add a new function on top of the create_base_gui() function.

```
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 " , __func__);
    }
    free( ad->ms.memory);
}
```

elm_image_memfile_set(Evas_Object *obj, const void *img, size_t size, const char *format, const char *key) is an API that enters original image data in an Evas Image object. The parameters listed in order are an Evas Image object, image data, data size, and image type.

Build the example and tap the second Button. The Tizen logo image downloaded from the web server is displayed on the screen.



4) Related APIs

CURLcode curl_global_init(long flags): an API that initializes the CURL library. For apps that use CURL, this API must be run once initially.

CURL* curl_easy_init(void): an API that creates a CURL object.

CURLcode curl_easy_setopt(CURL *curl, CURLoption option, ...): an API that specifies an option for a CURL object. The option types are as follows:
 - CURLOPT_URL: specifies a URL address.
- CURLOPT_WRITEFUNCTION: specifies a callback function that receives communication results.
- CURLOPT_WRITEDATA: specifies user data.
- CURLOPT_USERAGENT: specifies a user agent.

CURLcode curl_easy_perform(CURL *curl): an API that starts communication with a server.

void curl_easy_cleanup(CURL *curl): an API that deletes CURL data.

void curl_global_cleanup(void): an API that deletes the entire data of the CURL library. Once CURL is used, this API should be called at least once before closing the app.

Eina_Bool elm_image_memfile_set(Evas_Object *obj, const void *img, size_t size, const char *format, const char *key): an API that enters original image data in an Evas Image object. The parameters listed in order are an Evas Image object, image data, data size, and image type.

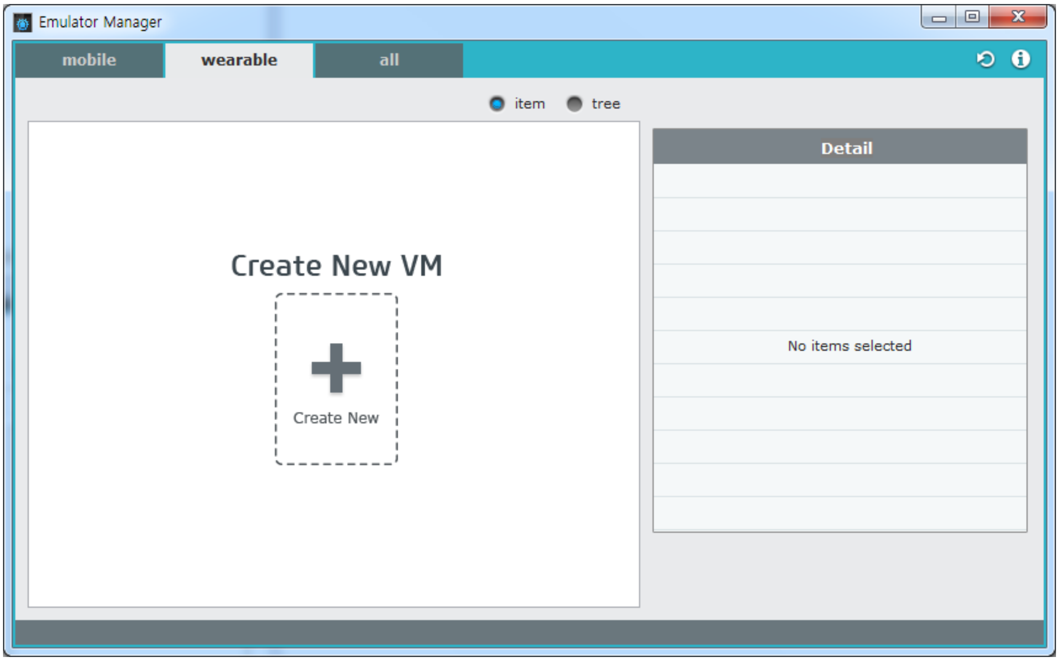
71. Creating a Wearable Source Project

In this example, we are going to learn how to create a wearable emulator and a wearable source project.

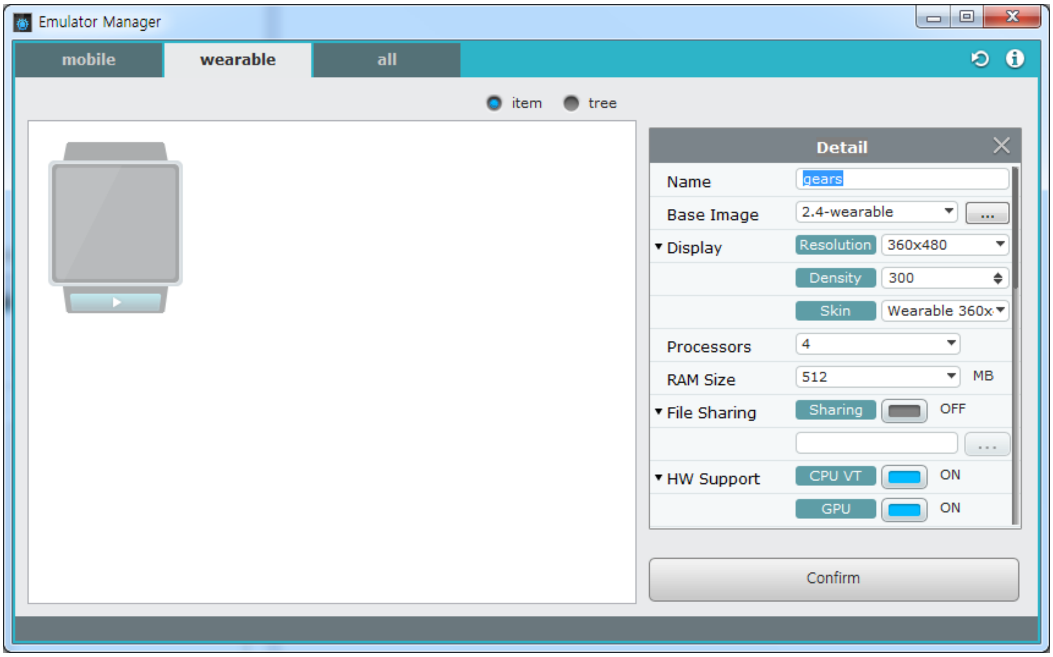
A. Running a Wearable Emulator

In this section, you will learn how to create and run an emulator. Click [Windows Start button > All Programs > Tizen SDK > Emulator Manager]. If you see a warning window, ignore it and click the Yes button.

When Emulator Manager is launched, click the second tab button 'wearable.' When there is no existing wearable emulator, an emulator needs to be created. Click the '+' sign below 'Create New VM' on the left.

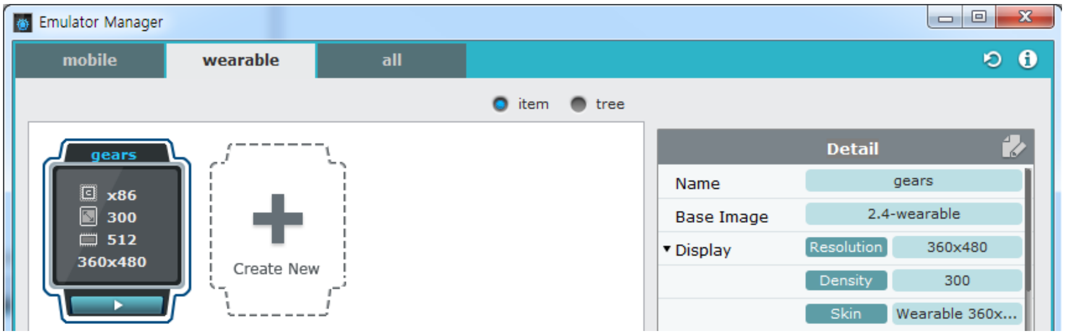


You will then see a screen where you can designate the options of the emulator on the right. Specify 'gears' as the name property. You can leave the rest as the default properties. Click the Confirm button to create a new emulator.



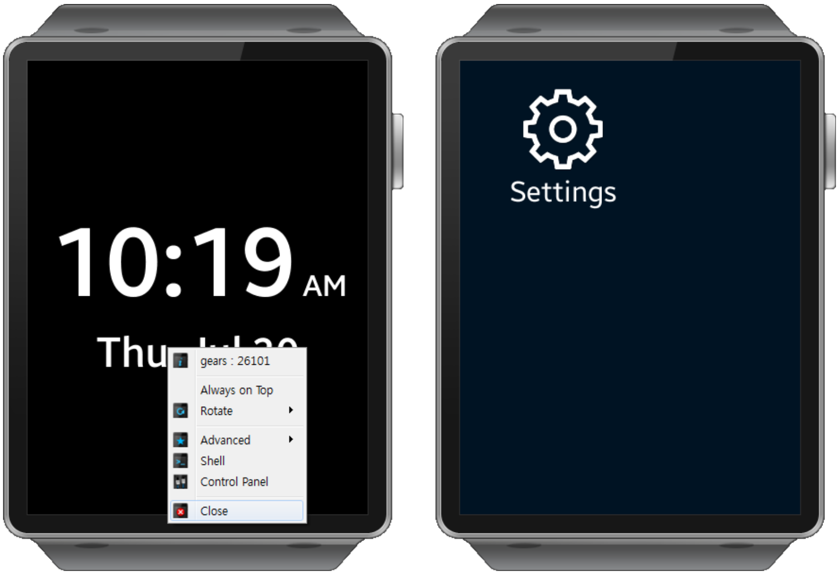
Name gears
Base Image 2.4-wearable
Display Resolution 360x480
Density 300
Skin Wearable 360x
Processors 4
RAM Size 512 MB
File Sharing Sharing OFF
HW Support CPU VT ON
GPU ON

You will now see a new emulator named 'gears' on the left side of the screen. Click the arrow button below the new emulator icon to run the emulator. If you want to change the options of the emulator, click the Reset button.



Name gears
Base Image 2.4-wearable
Display Resolution 360x480
Density 300
Skin Wearable 360x...

When the 'Windows Security Warning' popup window appears, tap the Unblock button to continue the process. There is a power button at the top right of the emulator, which performs two functions: Power and Home. Clicking it once turns on/off the screen, and clicking and holding it terminates the emulator.



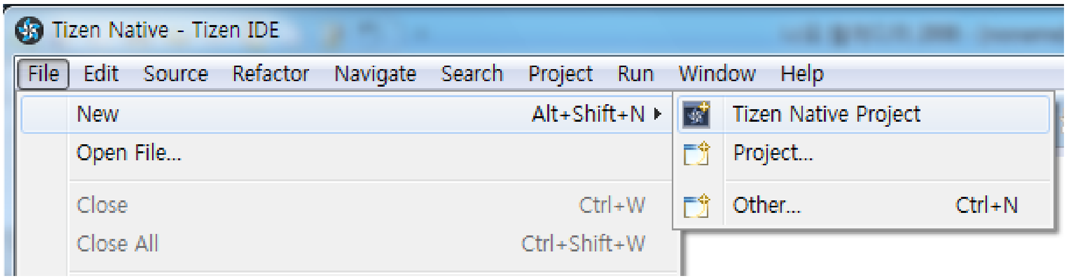
When it has been switched to Sleep mode, click the power button, or right-click the mouse and then select Close from the shortcut menu.

Flicking up displays a list of apps, and flicking down moves back to the previous screen.

B. We are now going to create a new source project and create an example that requests a Button click event.

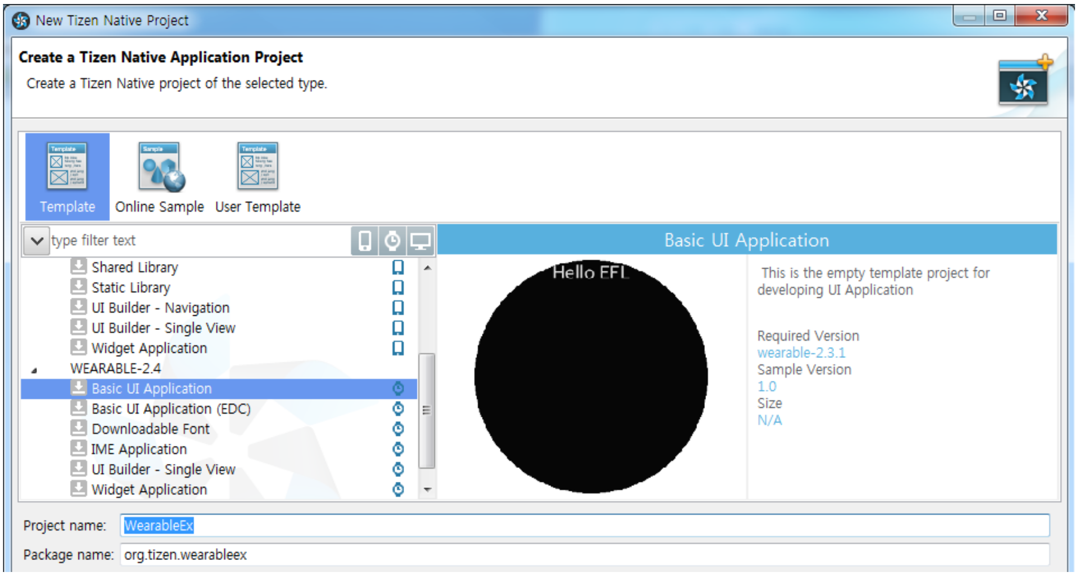
1) Creating a Source Project

The first step is to create a new source project. Select [File > New > Tizen Native Project] from the main menu of Eclipse.



When a popup window for creating a source project appears, select [Template > WEARABLE-2.x > Basic UI Application].

Enter 'WearableEx' in the Project name field.



Create a Tizen Native Application Project

Create a Tizen Native project of the selected type.

Basic UI Application

This is the empty template project for developing UI Application

Required Version
wearable-2.3.1
Sample Version
1.0
Size
N/A

Project name: WearableEx

Package name: org.tizen.wearableex

The field Package name will be automatically filled. Now, click the Finish button to create a new source project.

2) Components of Source Project

The basic source projects of wearable apps are very similar to those of mobile apps. The Includes folder contains libraries.

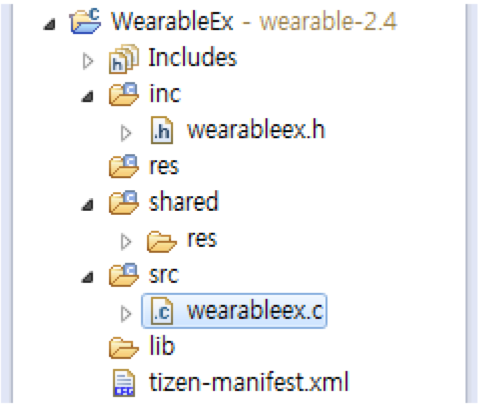
This folder includes header files in C language (.h). You can mostly use this folder to define libraries, function headers, or global variables.

The ‘res’ folder is usually used to save resource files including image or audio files.

The ‘src’ folder includes the source files in C language (.c). This folder is mainly used to define the features of functions. Most tasks are performed here.

The ‘shared’ folder includes the app icon images. When you distribute your apps to app stores, you can save your app icons here. For Tizen store, you should use app icons with a round shape.

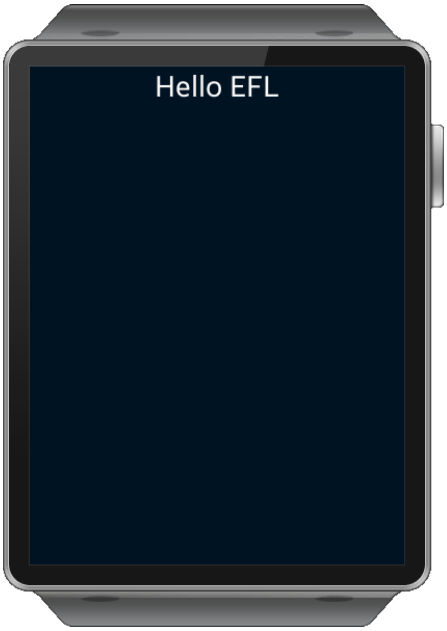
The ‘tizen-manifest.xml’ file includes a variety of app information (e.g. the name and version of app) and user’s rights (privileges). It is basically the same as AndroidManifest.xml in Android.



3) Running a Basic Source Project

We are going to run the source project the way it was first created. Right-click the WearableEx project and then select [Build Project] from the shortcut menu. When the project is successfully built, right-click the project and select [Run As > 1 Tizen Native Application] from the shortcut menu. If there is no certificate installed, install a certificate in the same way as mobile emulators.

When the example is run on the emulator, you will see the text 'Hello EFL' at the top. This text is displayed using Label Widget.



4) Changing the Text in a Label

Now, let’s change the text, 'Hello EFL,' displayed in the Label widget. To do this, we need to edit the source file. Open the src folder, and double-click the 'wearableex.c' file. You can now see the content of the file as shown below on the main screen of Eclipse. The source code is similar to that for mobile apps, so please see the explanation provided at the beginning of this guide.

We are now going to change the text 'Hello EFL' displayed on the screen. Modify the create_base_gui() function as shown below:

```
   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_EXPAN
D);
   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() is an API that changes the caption text of a widget. You can use this function for Button and Entry widgets as well as Label widgets. You can also specify text properties by using HTML tags when creating text.

Let's run the example again. When you run an example for the second time, click [Run > Run] in the main menu or press the 'Ctrl + F11' hotkey. The example runs again, and the text on the screen changes to 'Hello World.'

Hello World

5) Adding a Button Widget

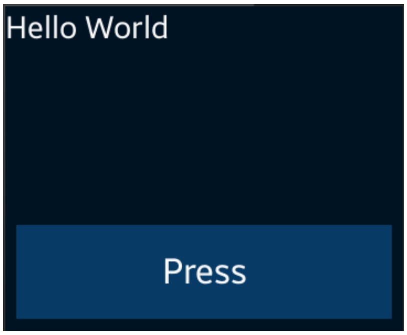
In this section, we are going to implement a feature that changes the text of a Label when a Button is clicked. The process is the same as for mobile apps. Add new code to the create_base_gui() function.

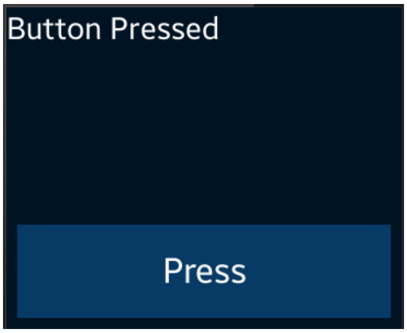
```
    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    elm_win_indicator_mode_set(ad->win, ELM_WIN_INDICATOR_SHOW);
    elm_win_indicator_opacity_set(ad->win, ELM_WIN_INDICATOR_OPAQUE);
    evas_object_size_hint_weight_set(ad->conform, EVAS_HINT_EXPAND, EVAS_HINT_EX
PAND);
    elm_win_resize_object_add(ad->win,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

This code creates a Box container and a Button widget. We specified the name of the click event callback function as 'btn_click_cb.' We are now going to create this function. Add a new function on top of the create_base_gui() function.

```
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");
}
```

This code changes the text of the Label widget to 'Button Pressed' when a Button is clicked. Run the example again and tap the Button. The text in the Label changes. You can see that using the Basic UI is a similar process to mobile apps.





72. Wearable System Information

At the beginning of this guide, we learned how to request mobile system information. In this example, we are going to build the same app as a wearable app and see in what ways they are different from each other and in others how they are the same.

1) Existence of a Rear Camera

Create a new source project and then specify the type as Basic UI Application and specify the project name as 'wSystemInfo.' After the source project is created, open the source file (~.c) under the src folder and add a library header file and variables at the top of the source file.

```
#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;
```

We declared a total of five Label widget variables. In the first Label, we are going to display whether or not a rear camera exits; in the second Label, availability of phone calls; in the third Label, the number of horizontal pixels on the monitor; in the fourth Label, the number of vertical pixels on the monitor; and in the fifth Label, the version of the platform.

Add new code in the create_base_gui() function. This code creates one Button widget and five Label widgets.

```
    /* 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);
```

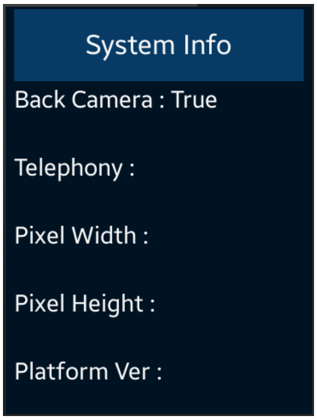
```
        elm_object_text_set(ad->label5, "Platform Ver :");
        evas_object_size_hint_weight_set(ad->label5, EVAS_HINT_EXPAND, 0);
        evas_object_size_hint_align_set(ad->label5, EVAS_HINT_FILL, 0);
        elm_box_pack_end(box, ad->label5);
        evas_object_show(ad->label5);
    }

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

We are now going to create a Button callback function. Add new code on top of the create_base_gui() function.

```
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", &bValu
e);
    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 *) is an API that requests system information. The type of returned data is Boolean. The first parameter is the key value, and passing 'http://tizen.org/feature/camera.back' to the first parameter returns whether or not a rear camera exists.

Build and run the example. Tap the Button, and you will see the text of the first Label has changed. Clicking the Button in an emulator displays 'False,' while tapping the Button in a user device displays 'True.'



System Info

Back Camera : True

Telephony :

Pixel Width :

Pixel Height :

Platform Ver :

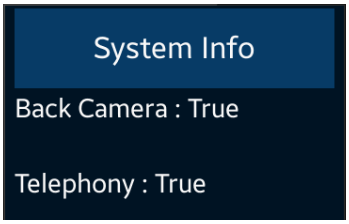
2) Existence of a Phone Feature

In this section, we are going to check whether or not a phone feature exists. Add new code at the end of the btn_clicked_cb() function.

```
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.telephon
y", &nValue);
    if (ret == SYSTEM_INFO_ERROR_NONE)
    {
        sprintf(buf, "Telephony : %s", bValue ? "True" : "False");
        elm_object_text_set(ad->label2, buf);
    }
}
```

Passing 'http://tizen.org/feature/network.telephony' to the first parameter of the system_info_get_platform_bool() function returns whether or not a phone feature exists. Even if the returned value is True, it does not mean that you can make phone calls or that you can use a network. This only means that the device is fitted with a hardware communication feature. You cannot use communication if the device does not have a USIM chip, or if the network feature has been deactivated in Settings.

Build and run the example. Tap the Button and you will see the text 'True' displayed in the second Label.



System Info

Back Camera : True

Telephony : True

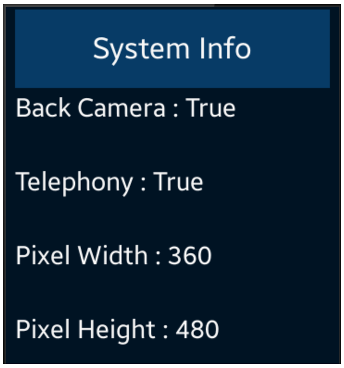
4) Number of Pixels on the Monitor

In this section, we are going to request the number of pixels of the monitor. Add new code at the end of the btn_clicked_cb() function.

```
    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", &nValue);
    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 *) is an API that requests system information. The type of returned data is an integer. The first parameter is the key value, and passing 'http://tizen.org/feature/screen.width' returns the number of horizontal pixels on the monitor. Passing 'tizen.org/feature/screen.height,' meanwhile, returns the number of vertical pixels on the monitor.

Build and run the example. Tap the Button, and you will see the values displayed in the third Label and fourth Label, respectively.



System Info

Back Camera : True

Telephony : True

Pixel Width : 360

Pixel Height : 480

5) Version of the Platform

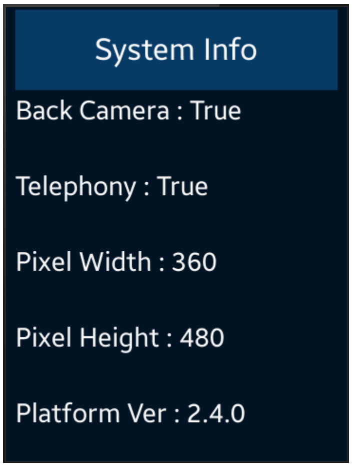
In this section, we are going to request the version of the platform. Add new code at the end of the btn_clicked_cb() function.

```
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/platform.versio
n", &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 **) is an API that requests system information. The type of returned data is string. The first parameter is the key value, and passing 'http://tizen.org/feature/platform.version' returns the version of the platform.

Build and run the example. Tap the Button, and you will see the version of the platform displayed in the fifth Label.



System Info

Back Camera : True

Telephony : True

Pixel Width : 360

Pixel Height : 480

Platform Ver : 2.4.0

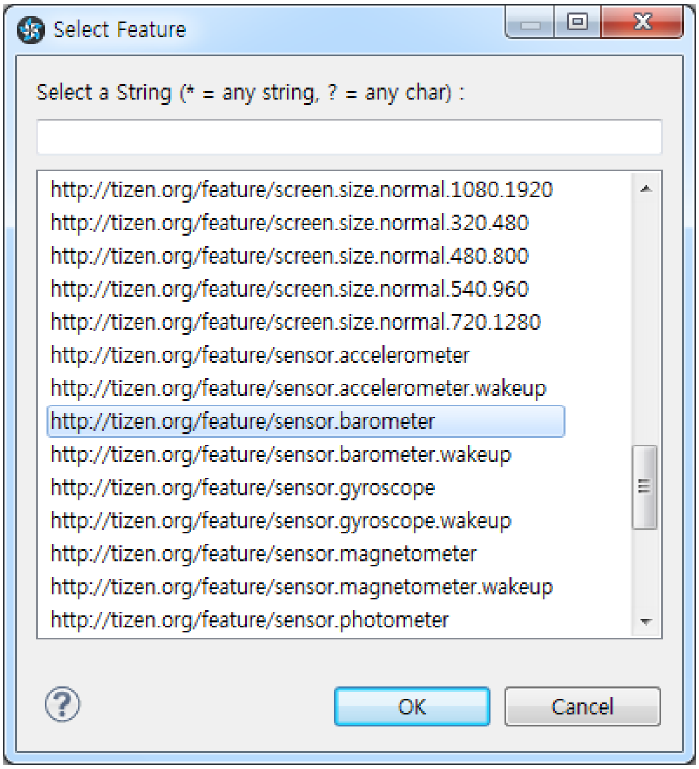
This illustrates that we can also use the APIs that we used for mobile apps for wearable apps.

73. Using a Pressure Sensor

Features that smartphones do not have, such as heart rate sensors and pressure sensors, are added to wearable devices. In this example, we are going to learn how to use the pressure sensor.

1) Adding a Feature

Create a new source project and specify the project name as 'wSensorPressure.' To use the pressure sensor feature, you need to add the relevant feature. After the source project is created, open the tizen-manifest.xml file, and click Features among the tab buttons below. Then, click the Add button in the upper right corner. When a popup window appears, select http://tizen.org/feature/sensor.barometer from the list, and click the OK button to close the window.



Select a String (* = any string, ? = any char) :

http://tizen.org/feature/screen.size.normal.1080.1920
http://tizen.org/feature/screen.size.normal.320.480
http://tizen.org/feature/screen.size.normal.480.800
http://tizen.org/feature/screen.size.normal.540.960
http://tizen.org/feature/screen.size.normal.720.1280
http://tizen.org/feature/sensor.accelerometer
http://tizen.org/feature/sensor.accelerometer.wakeup
http://tizen.org/feature/sensor.barometer
http://tizen.org/feature/sensor.barometer.wakeup
http://tizen.org/feature/sensor.gyroscope
http://tizen.org/feature/sensor.gyroscope.wakeup
http://tizen.org/feature/sensor.magnetometer
http://tizen.org/feature/sensor.magnetometer.wakeup
http://tizen.org/feature/sensor.photometer

Add the following feature by repeating the procedure above.
- http://tizen.org/feature/sensor.barometer.wakeup

After saving, click the 'tizen-manifest.xml' button in the far right corner from the tab buttons located at the bottom; you should be able to see the source code of the xml file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<manifest xmlns="http://tizen.org/ns/packages" api-version="2.3.1" package="org.exa
mple.wsensorpressure" version="1.0.0">
    <profile name="wearable"/>
    <ui-application appid="org.example.wsensorpressure" exec="wsensorpressure" multi
ple="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</feat
ure>
</manifest>
```

2) Determining Whether the Pressure Sensor is Supported

Create a new source project and specify the project name as 'wSensorPressure.' After the source project is created, open the source file (~.c) under the src folder and add a library header file and variables.

```
#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 is the header file for various sensor libraries.

In label0, we are going to display whether or not the pressure sensor is supported, and in label1, the level of pressure.

Add a new function on top of the create_base_gui() function.

```
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 support");
    elm_object_text_set(ad->label0, buf);
}
```

show_is_supported() is a function that, after determining whether the pressure sensor is supported, displays the result in the first Label widget.

sensor_is_supported(sensor_type_e, bool *) is an API that requests whether a certain sensor is supported. Passing SENSOR_PRESSURE to the first parameter makes the second parameter return whether or not the pressure sensor is supported.

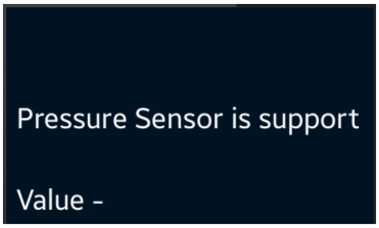
We need to make it so that this function is called when the app is run. Call the function above at the end of the create_base_gui() function.

```
    /* 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_EXP
AND);
    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_H
INT_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);
```

```
}
```

We created a Box container and two Label widgets. We also called the function that determines whether or not a sensor is supported.

Build and run the example. If the pressure sensor is supported, the message 'Pressure Sensor is supported' is displayed. Some models may not support the sensor. In such cases, you need to test it on the emulator.



Pressure Sensor is support

Value -

2) Requesting a Pressure Sensor Event

In this section, we are going to implement a feature that, when an object is detected by a pressure sensor, requests the relevant event and then displays the distance on the screen. Add a sensor-related structure and global variable at the top of the source file.

```
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 is a structure that includes the sensor object and event listener variable.

sensor_info is the global variable of the sensorinfo structure.

Requesting a sensor event means starting a listener. We are now going to request a pressure sensor event using a sensor object and an event listener. Add two new functions at the top of the create_base_gui() function.

```
static void _new_sensor_value(sensor_h sensor, sensor_event_s *sensor_data, void *use
r_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_listener);
    sensor_listener_set_event_cb(sensor_info.sensor_listener, 100, _new_sensor_value, ad);
    sensor_listener_start(sensor_info.sensor_listener);
}
```

_new_sensor_value() is the callback function for the pressure sensor event. It displays a new sensor value on the screen.

Sensor data is passed to the second parameter. Numeric data is saved in values[0].

start_pressure_sensor() is a function that starts the pressure sensor and specifies an event callback function.

sensor_get_default_sensor(sensor_type_e, sensor_h *) is an API that returns the sensor object. Passing SENSOR_PRESSURE to the first parameter makes the second parameter return the pressure sensor object.

sensor_create_listener(sensor_h, sensor_listener_h *) is an API that creates an event listener. Passing the sensor object to the first parameter makes the second parameter return the listener object.

sensor_listener_set_event_cb(sensor_listener_h, unsigned int, sensor_event_cb, void *) is an API that specifies a callback function for a listener. The parameters listed in order are an event listener, time interval (in milliseconds), name of the callback function, and user data.

sensor_listener_start(sensor_listener_h) is an API that starts a listener.

We are going to make it so that an event listener is automatically run when the app is run. Call the function above at the end of the create_base_gui() function.

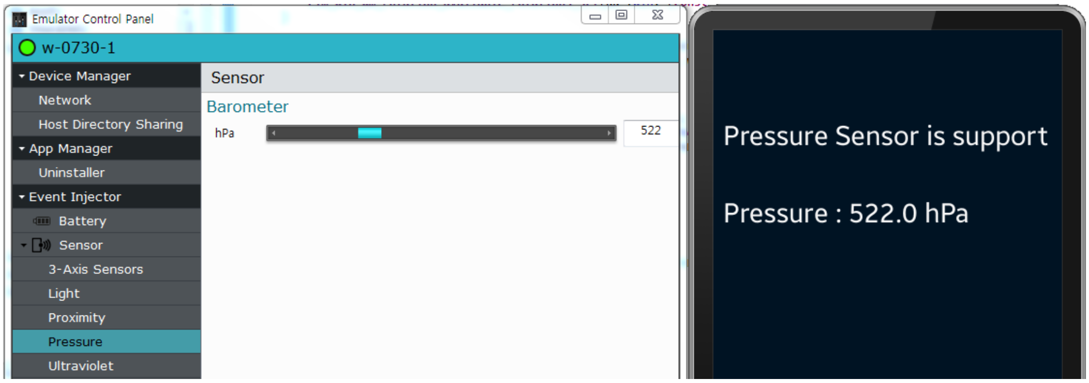
```
    /* Show window after base gui is set up */
    evas_object_show(ad->win);

    show_is_supported(ad);
    start_pressure_sensor(ad);
}
```

Let's run the example again. When testing the feature on the emulator, use the Control Panel.

Right-click the emulator and select Control Panel in the shortcut menu.

When the Control Panel appears, select the [Event Injector > Pressure] from the tree list to the left.

Drag the slide bar in the right pane of Control Panel left and right. If the figure on the emulator's second Label changes, it means that the feature has been implemented successfully.



Pressure Sensor is support

Pressure : 522.0 hPa

3) Related APIs

int sensor_is_supported(sensor_type_e type, bool *supported): an API that requests whether or not a certain sensor is supported. Passing SENSOR_PRESSURE to the first parameter makes the second parameter return whether or not the pressure sensor is supported.

int sensor_get_default_sensor(sensor_type_e type, sensor_h *sensor): an API that returns the sensor object. Passing SENSOR_PRESSURE to the first parameter makes the second parameter return the pressure sensor object.

int sensor_create_listener(sensor_h sensor, sensor_listener_h *listener): an API that creates an event listener. Passing the sensor object to the first parameter makes the second parameter return the listener object.

int sensor_listener_set_event_cb(sensor_listener_h listener, unsigned int interval_ms, sensor_event_cb callback, void *data): an API that specifies a callback function for a listener. / parameters: an event listener, time interval (in milliseconds), name of the callback function, and user data.

int sensor_listener_start(sensor_listener_h listener): an API that starts a listener.