# Development of Tizen Native Application


* This document is based on Tizen Studio

# Table of Contents

# Table of Contents

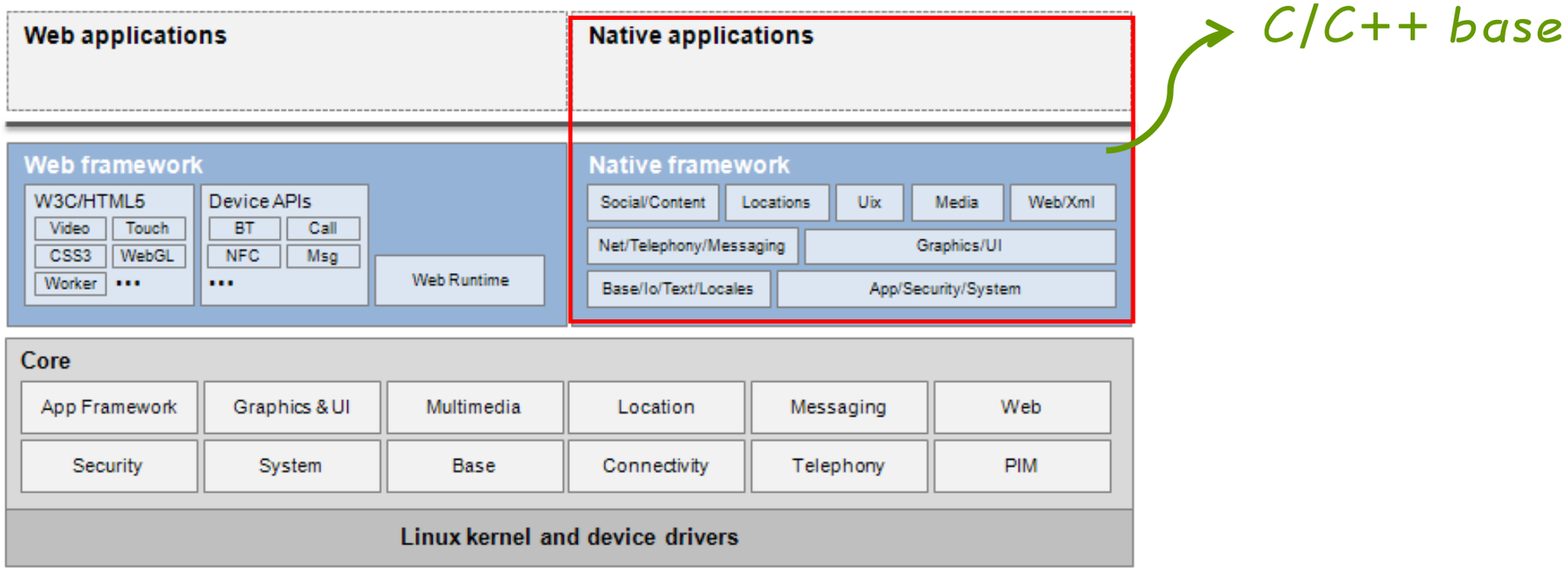
# Table of Contents

# Overview of Tizen Native Application

# Introduction



Native application is operated based on the Native Framework




| Benefits | Limitations |
| --- | --- |
| • Fast to drive<br>• Easy to control device<br>• High performance graphics | • Subordinated to the Platform<br>• High entry barrier(because of development language) |

# Implementation of Basic Mobile Application

# Tizen Native Application – Mobile Basic

To understand Native app, let's create a Basic UI project for mobile together.

# Tizen Native Application – Mobile Basic

We will proceed the implementation of the Mobile Basic UI app in 3 stages.

**Stage 1.**

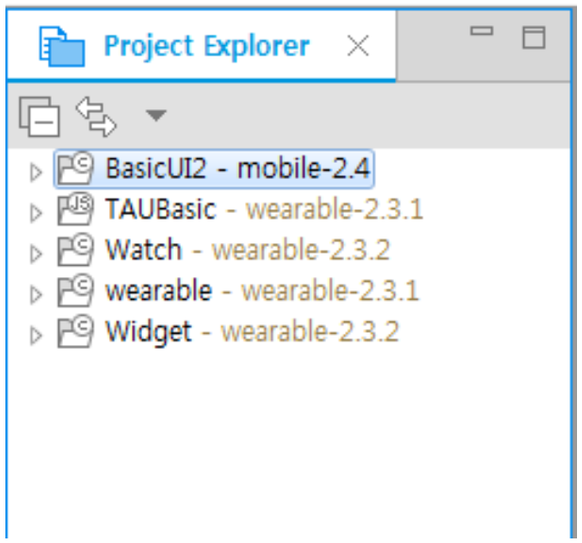
**Create Project in Tizen Studio**

**Stage 2.**

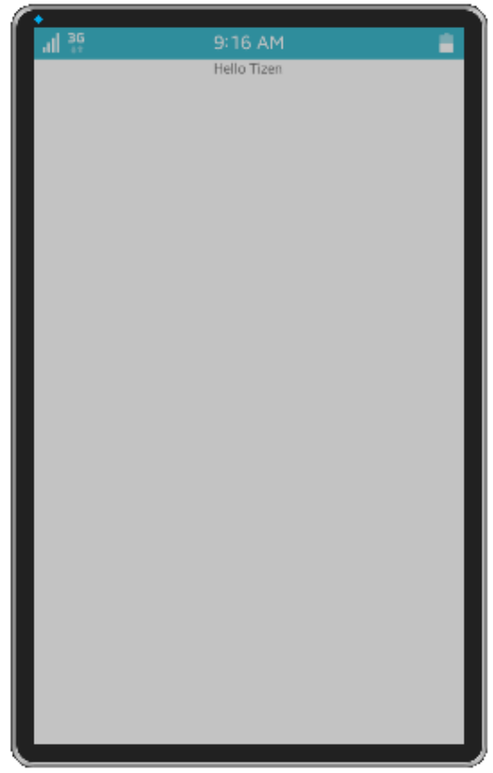
**Create Emulator for test**

**Stage 3.**

**Install and launch the project to the Emulator**



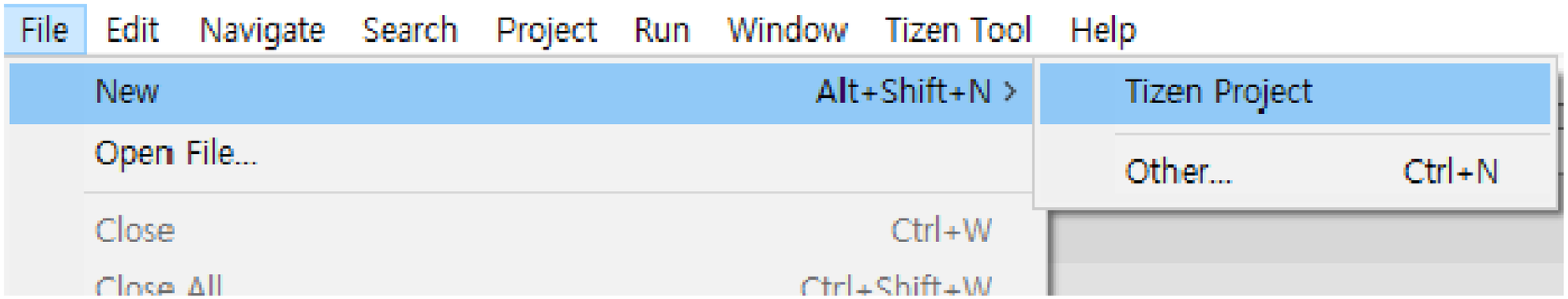




# Stage 1: Create Mobile Basic Project

Goal

Let’s create Project for Native UI Application with Tizen Studio.

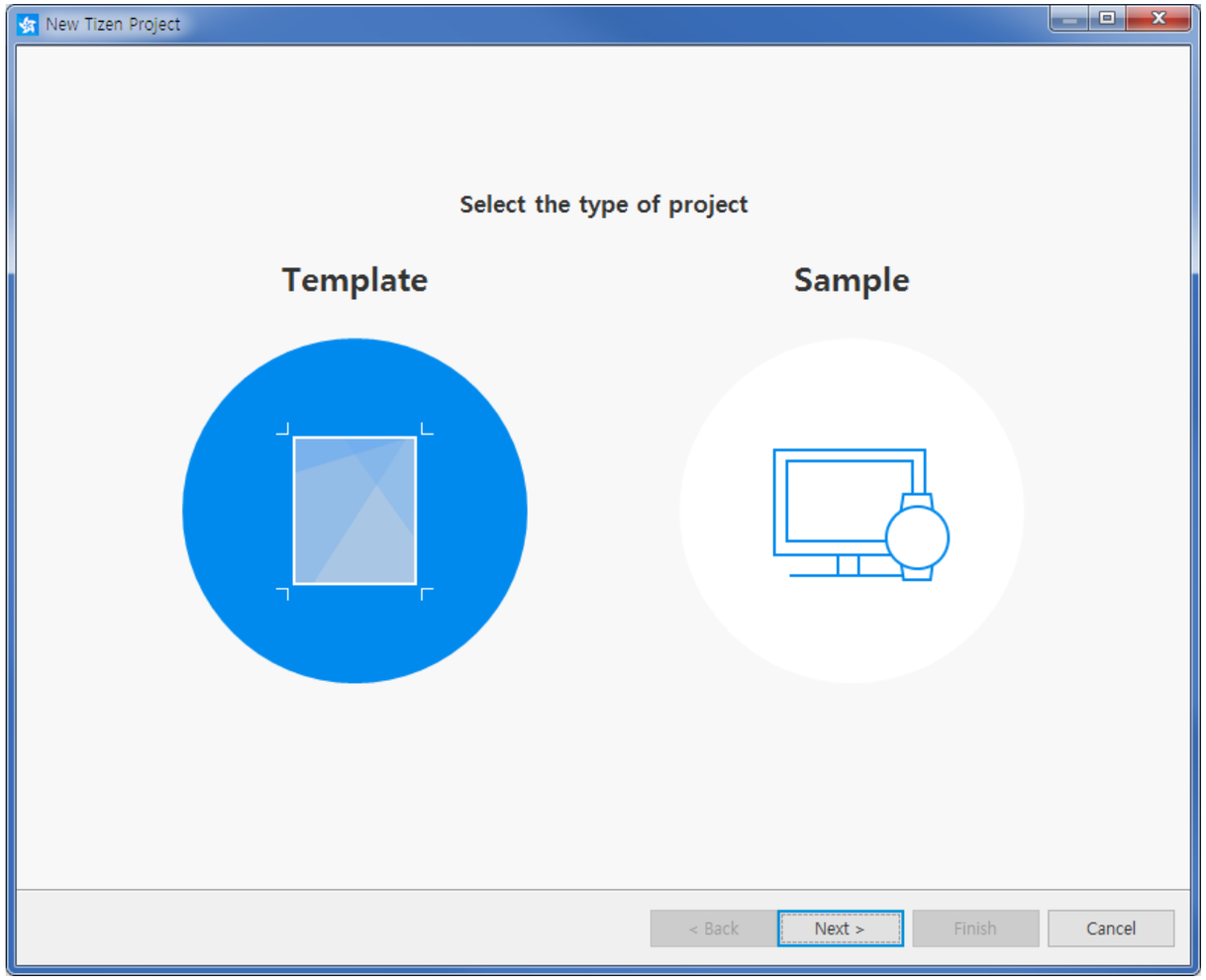


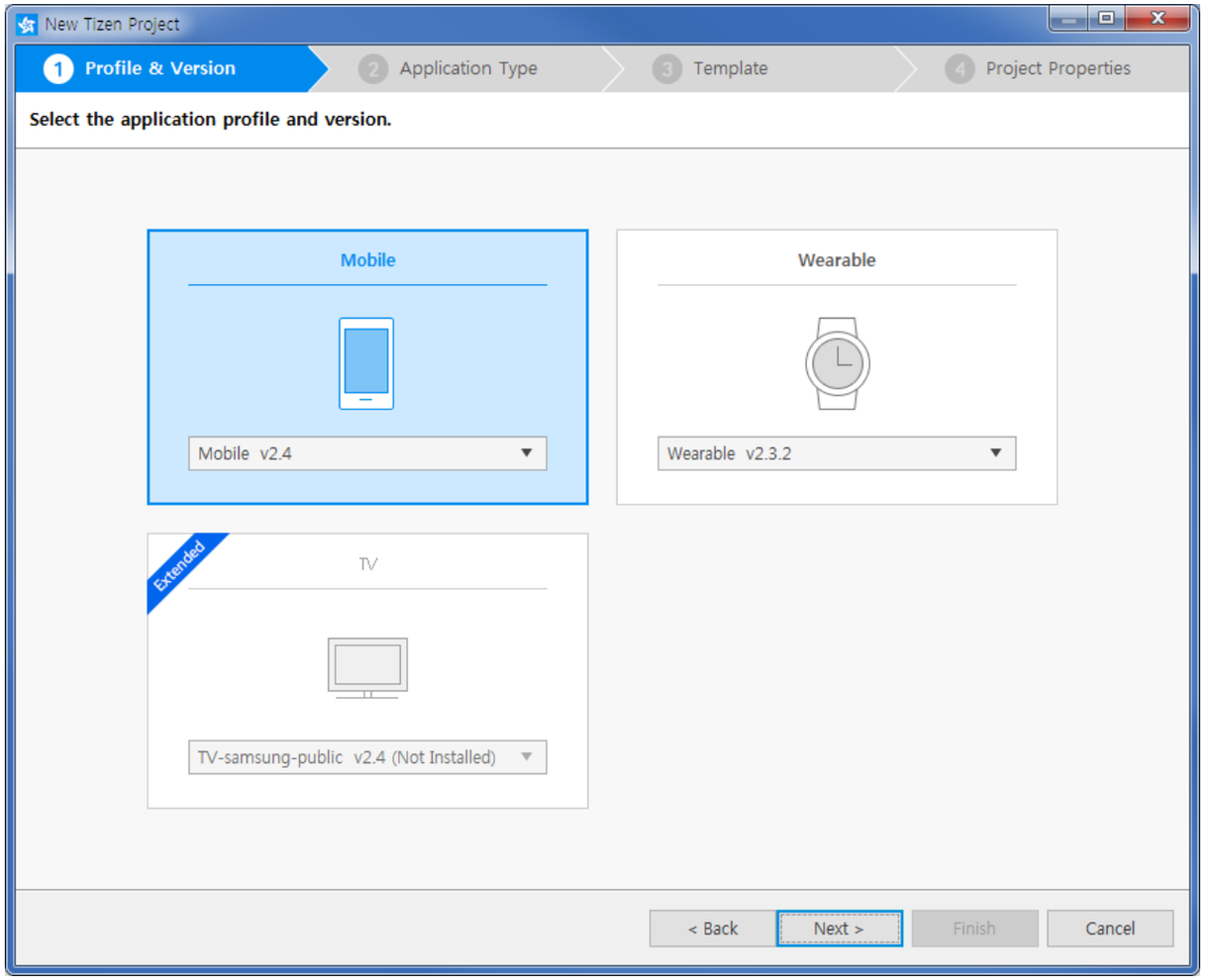
# Stage 1: Create Mobile Basic Project

Goal

Tizen Studio provide templates for various profiles i.e., Mobile, Wearable etc.

Choose Mobile and version 2.4.

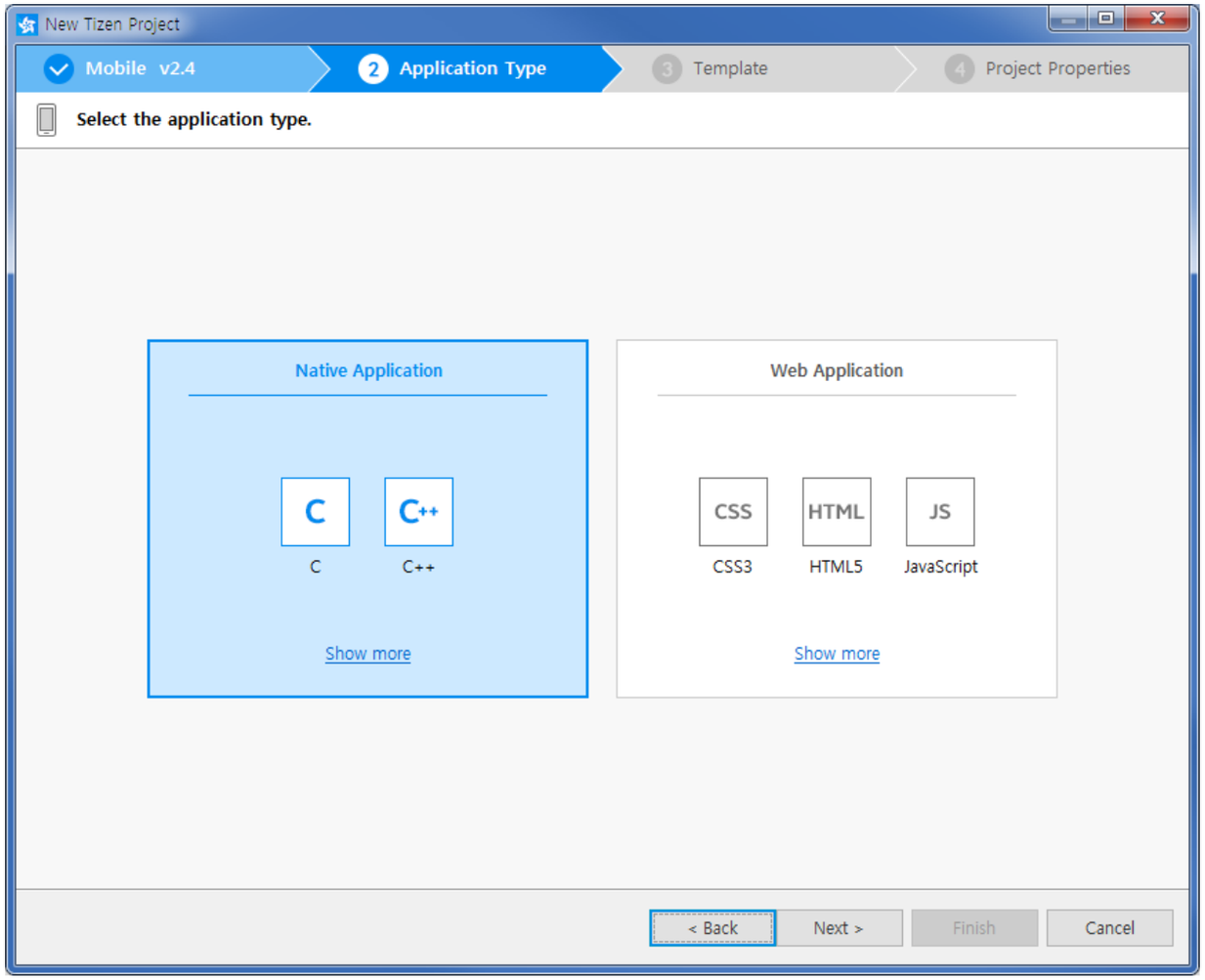


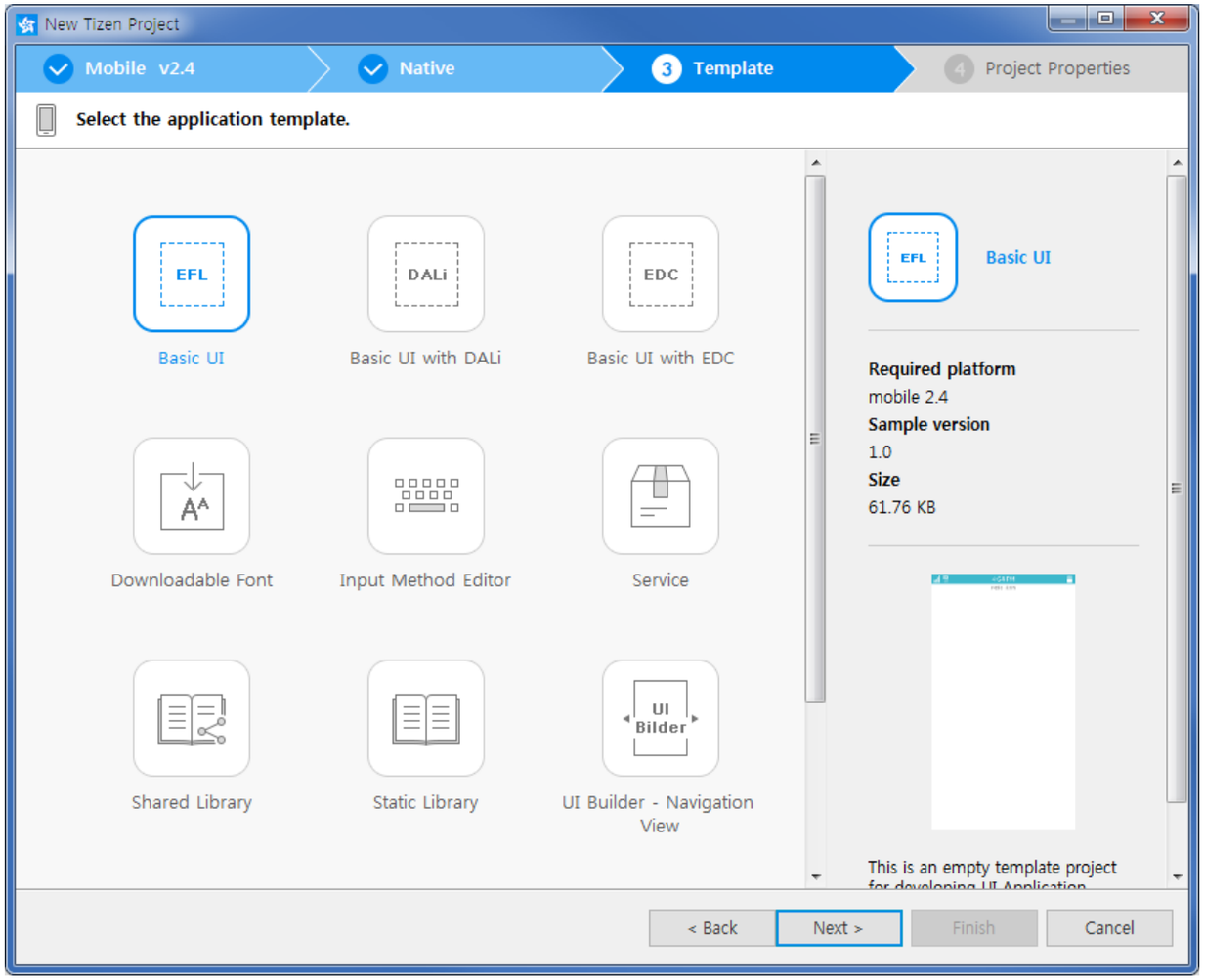



# Stage 1: Create Mobile Basic Project

Goal

Choose Native Application and Basic UI.

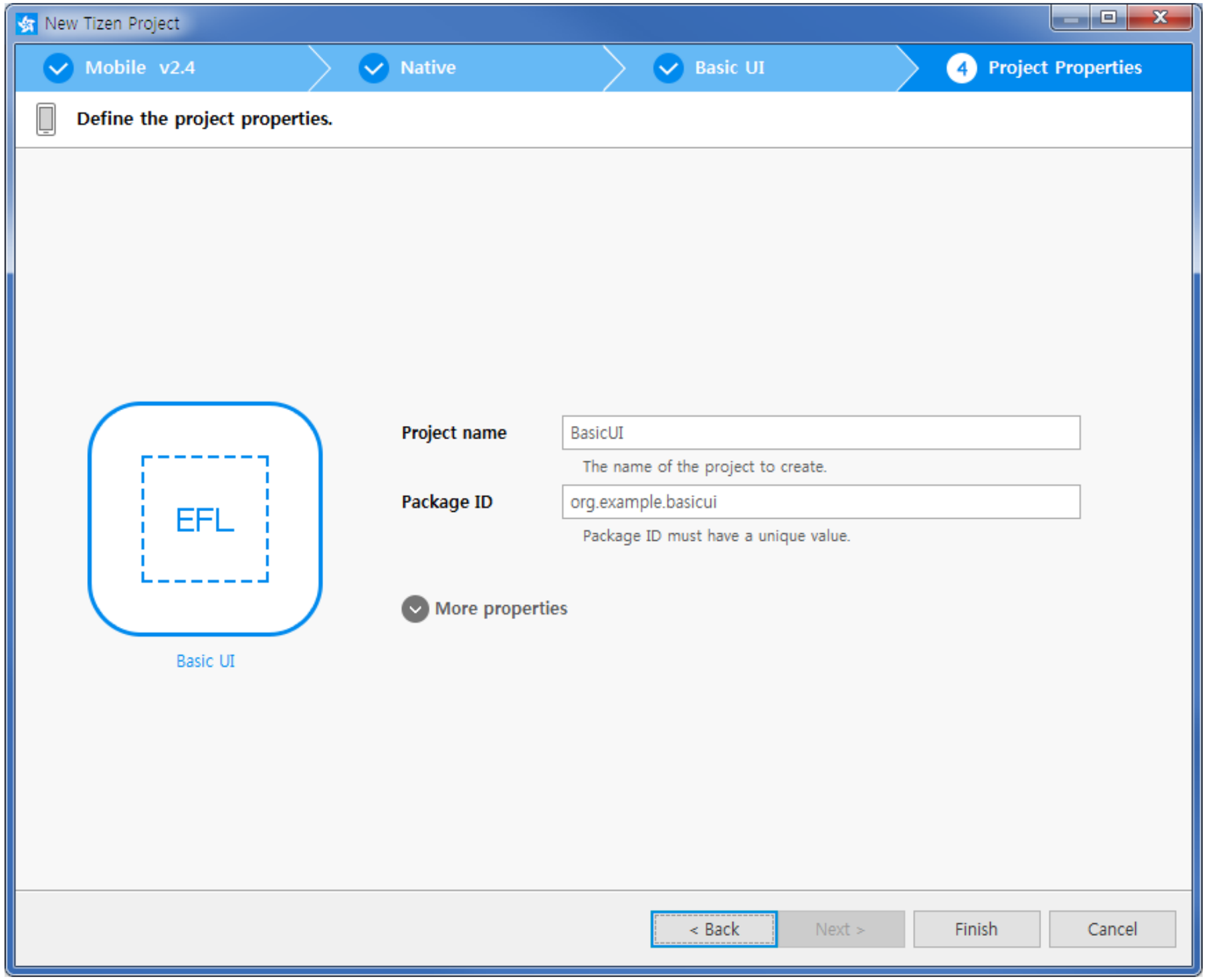


# Stage 1: Create Mobile Basic Project

Goal

Also, you can change the name of project, and this will affect to the app name



# Stage 1: Create Mobile Basic Project

Goal

Now, you can find your Project on the Project Explorer

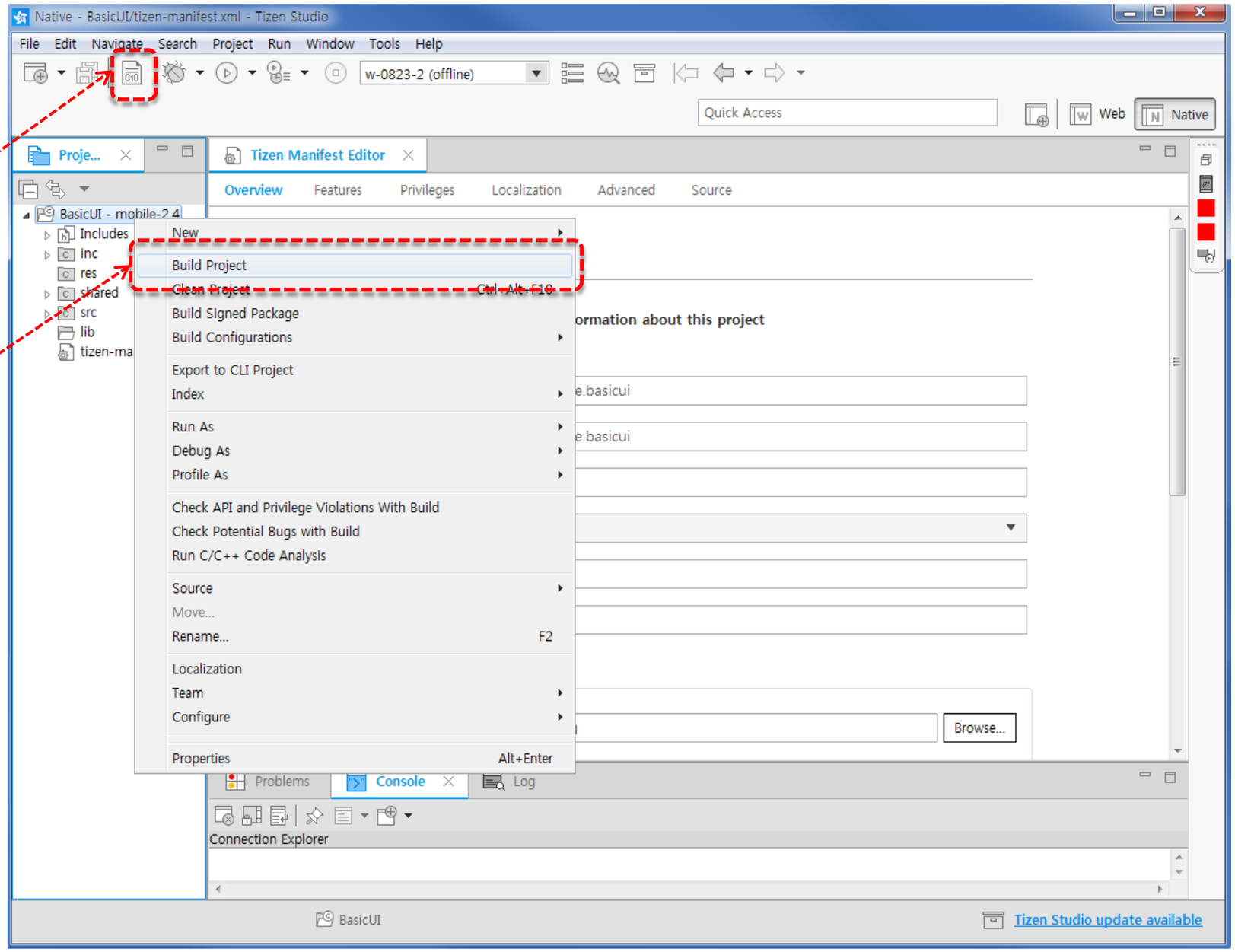
To build this project, Two methods are usually used

**Click the Project**

**Click Build Icon**

**Right Click on the Project**

**Build Project**



# Stage 2: Create Emulator for test

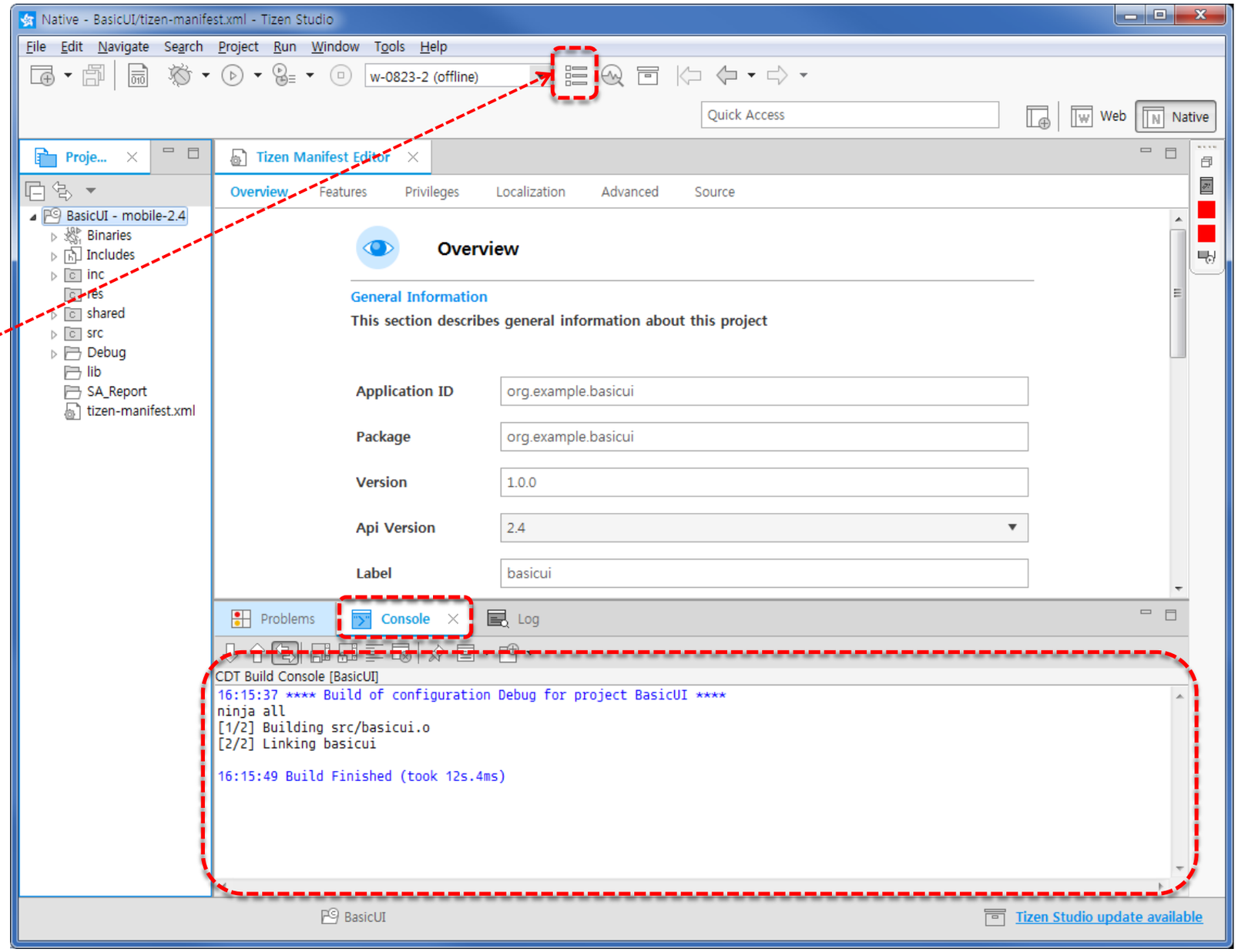
Goal

You can observe the progress of build through the Console page

In this page, also you can find error & warning messages

Create [Emulator] to test your project

**Click the Emulator Icon Here**



# Stage 2: Create Emulator for test

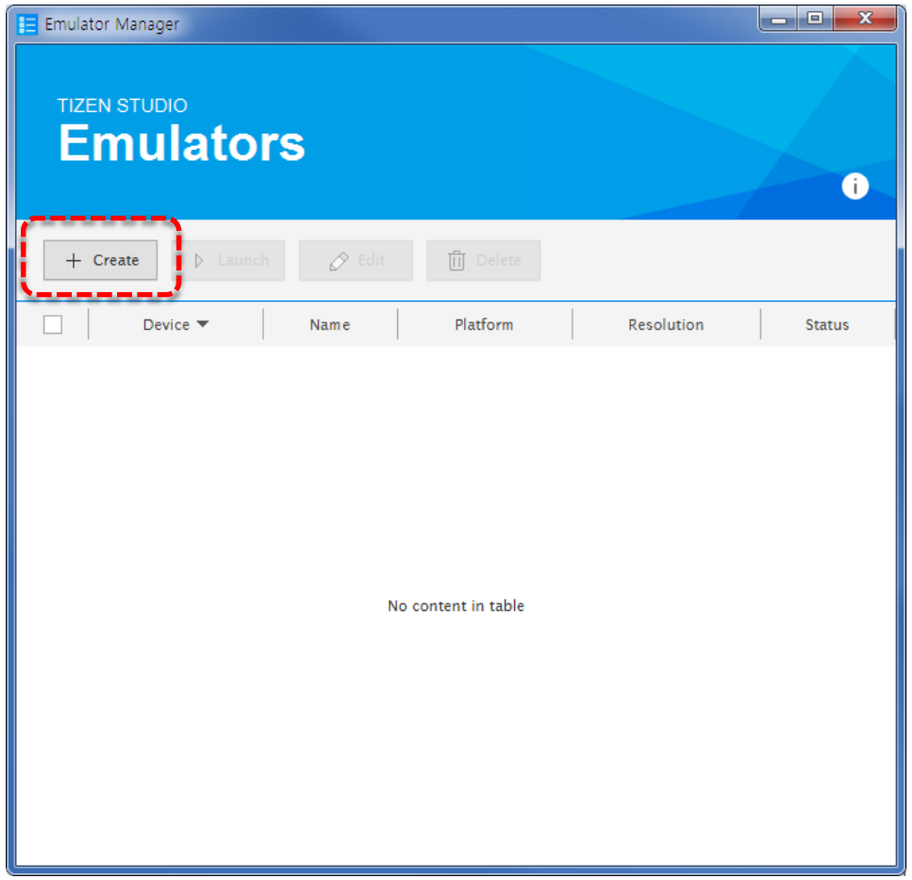
Goal

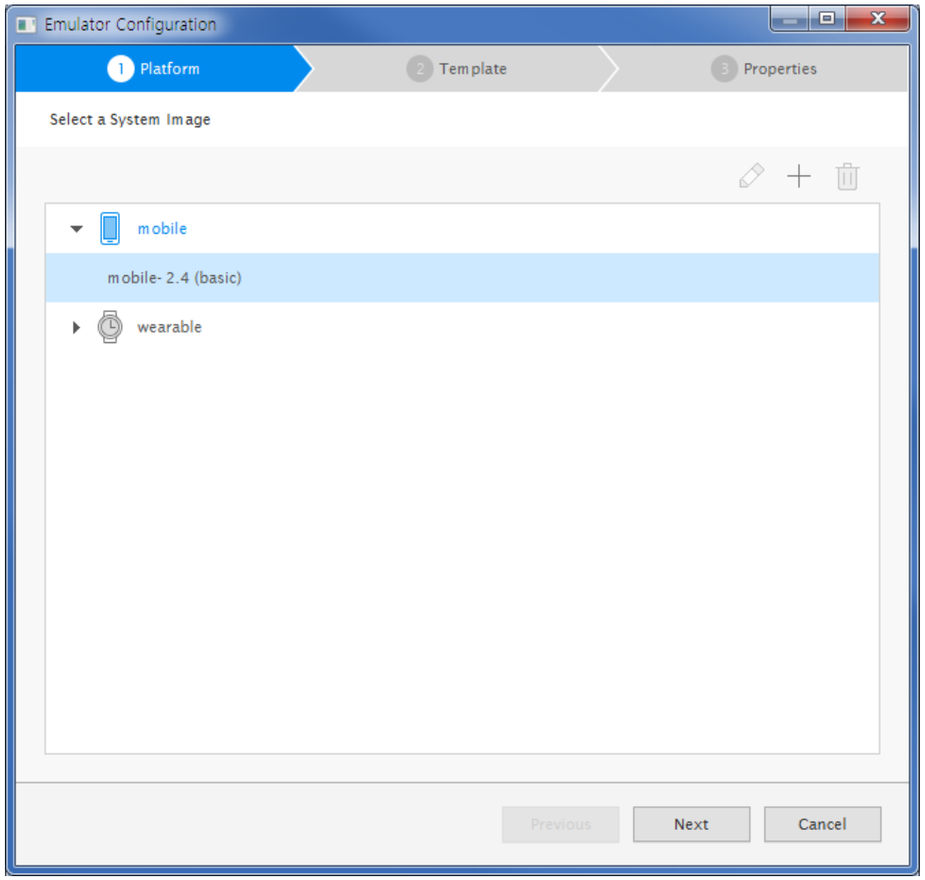
Tizen Studio provide Emulators for various profile(now, mobile and wearable)

For our Mobile Project, choose mobile profile

**Click**
**[+ Create]**

**Choose [mobile]**



# Stage 2: Create Emulator for test

Goal

Change the name of [Emulator] if you want

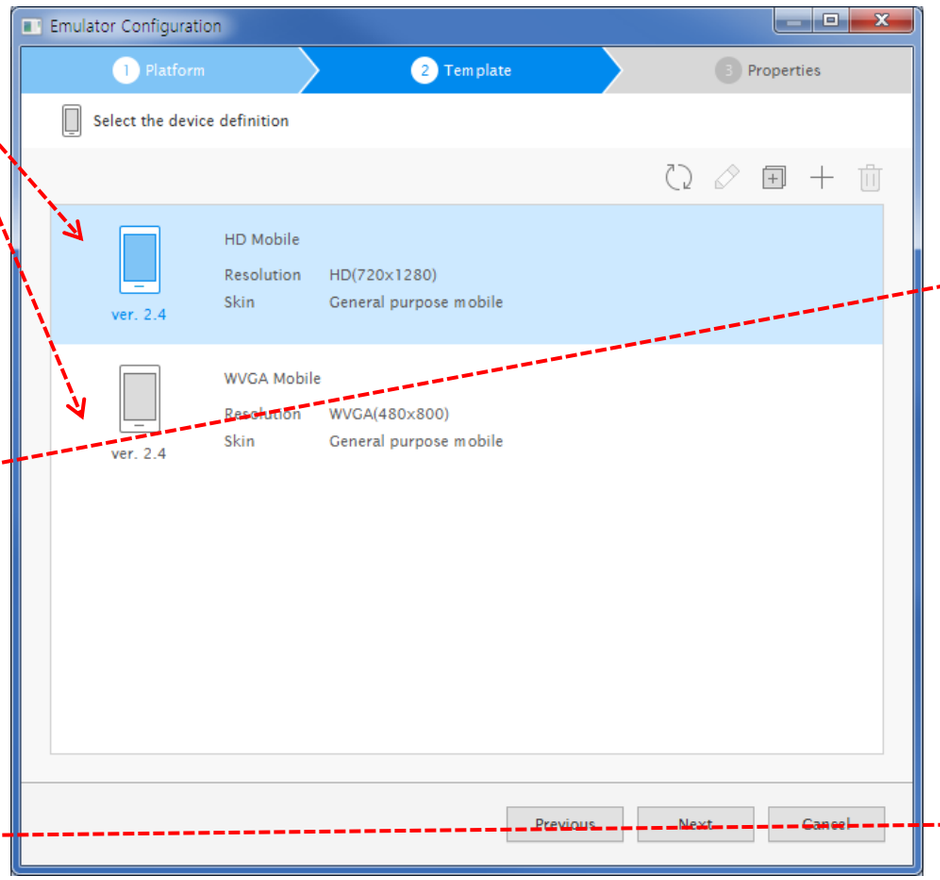
Another options are given as the Default for Mobile Emulator

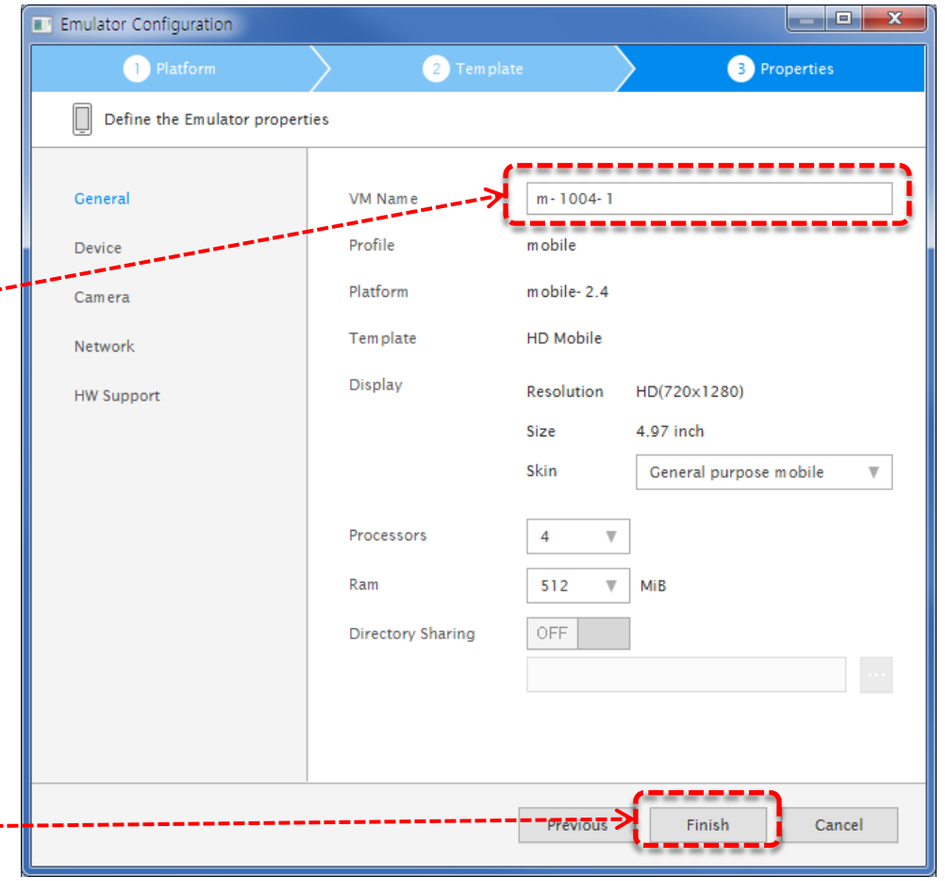
You can choose various screen size of [Emulator]

**Choose Resolution (HD or WVGA)**

**Check the name of Emulator**

**Click [Finish]**



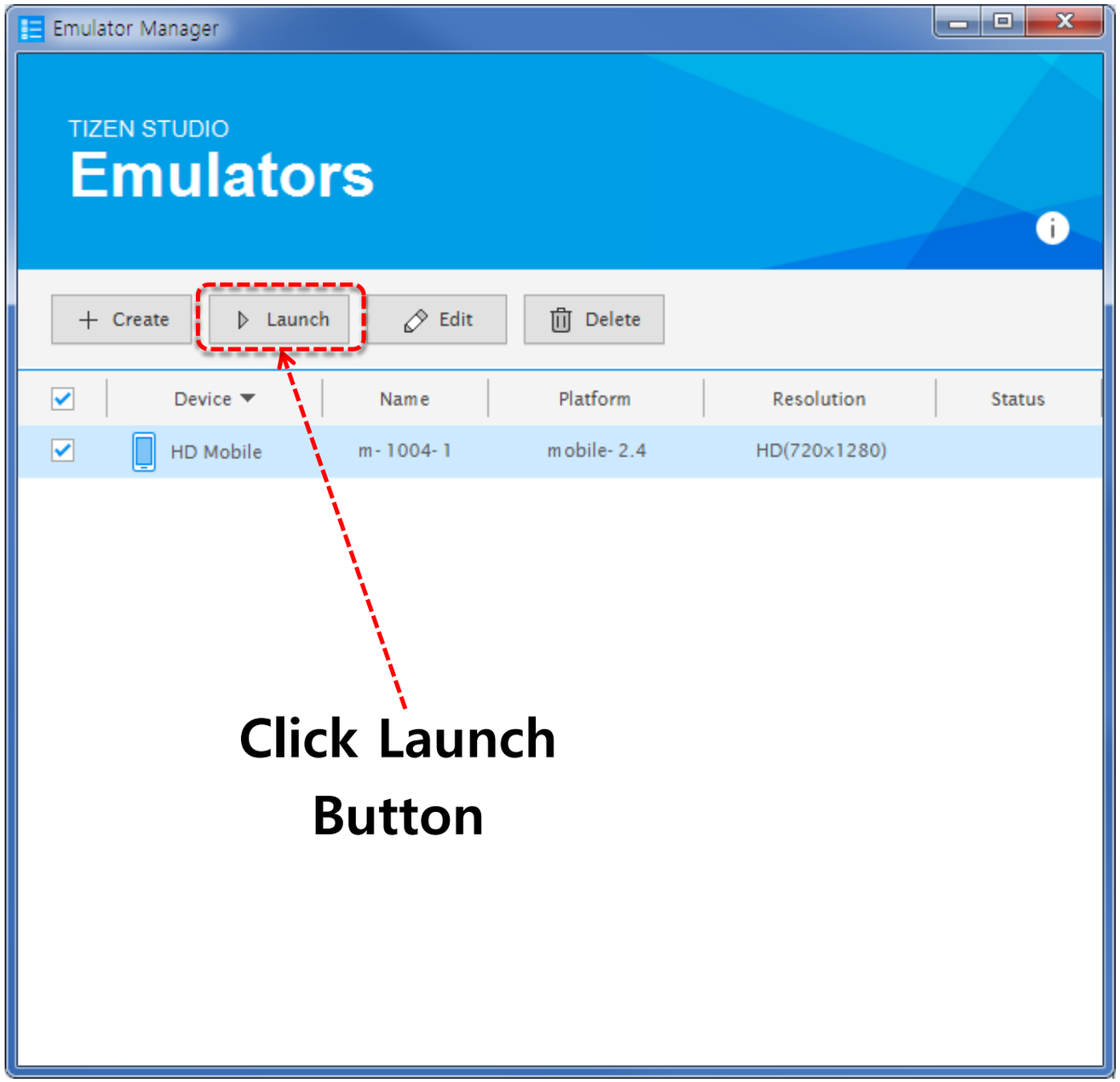



# Stage 2: Create Emulator for test

Goal

Click Launch Button to launch Emulator

You can find Default Mobile Emulator on the screen





**Now,**
**Let's launch your BasicUI project on the Emulator**

# Stage 3: Install & Launch the Project

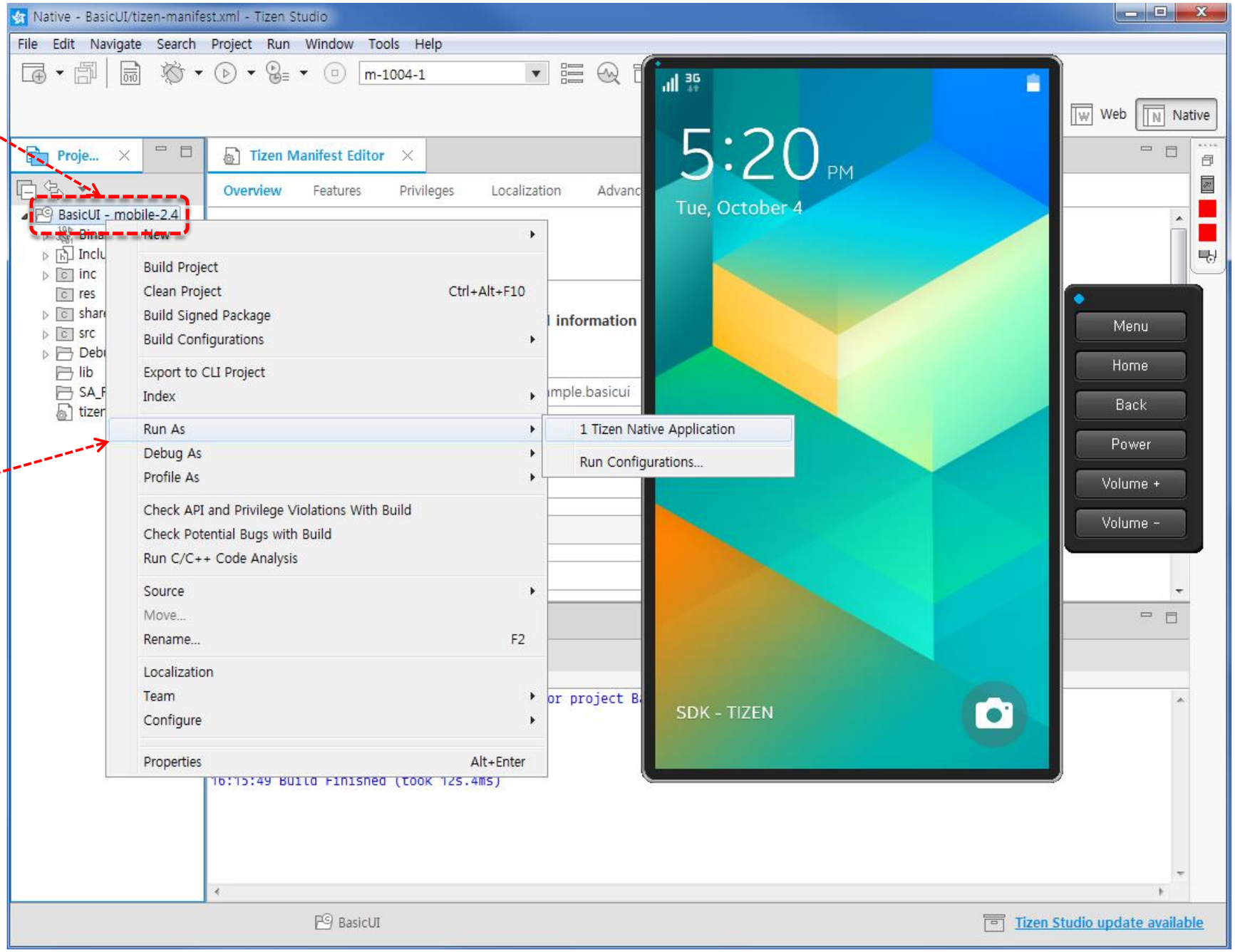
Goal

To install and launch your Project, just follow the sequence like below

**Right Click on BasicUI Project**

**Choose Run As**

**Choose Tizen Native Application**



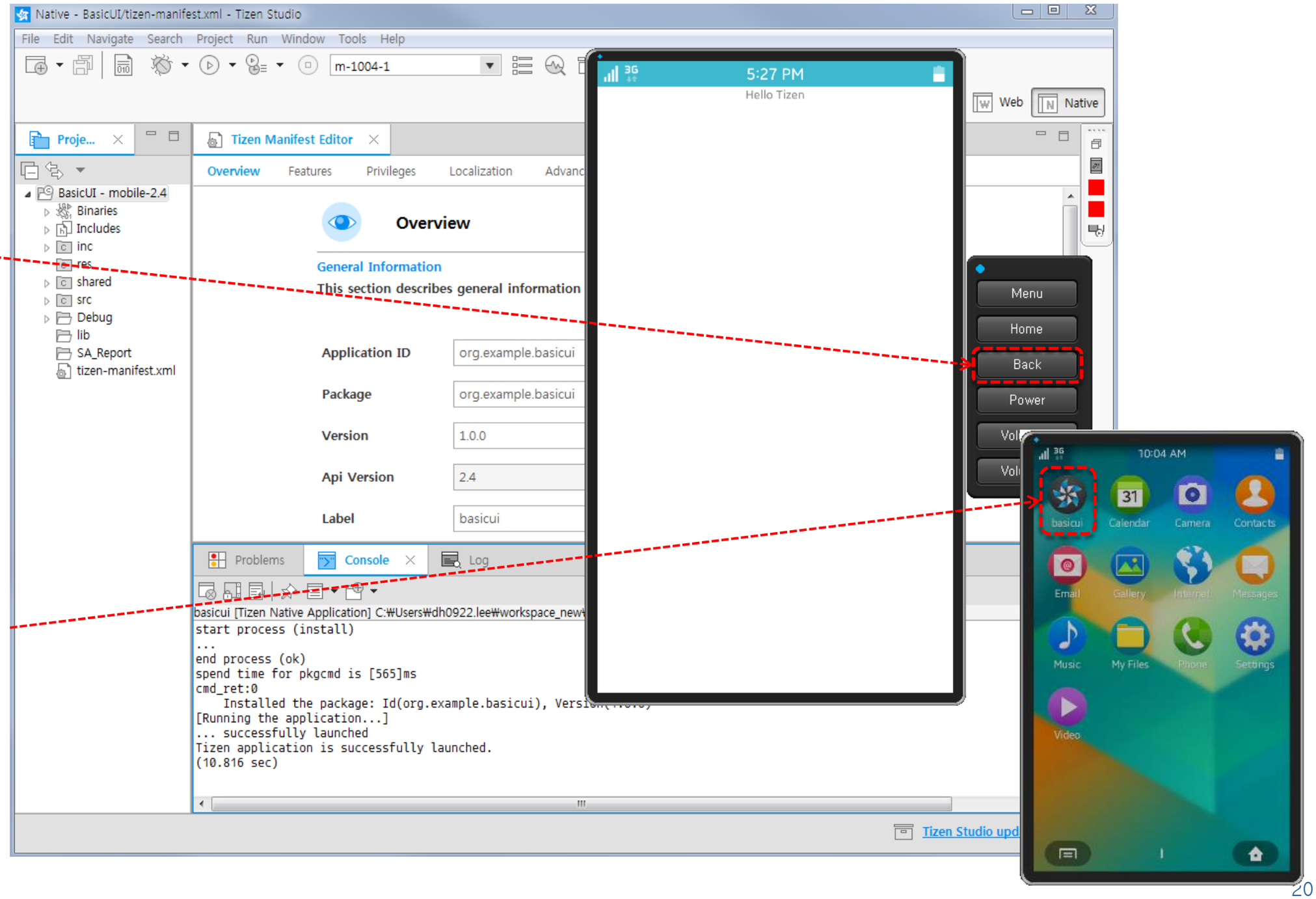
# Stage 3: Install & Launch the Project

Goal

When you swipe the screen (Lockscreen),
You can find ‘Hello Tizen’ on the white background
When you choose ‘Run As’, project will be installed and launched automatically

**Click Back button**

**Find Icon named ‘basicui’**



# Stage 3: Install & Launch the Project

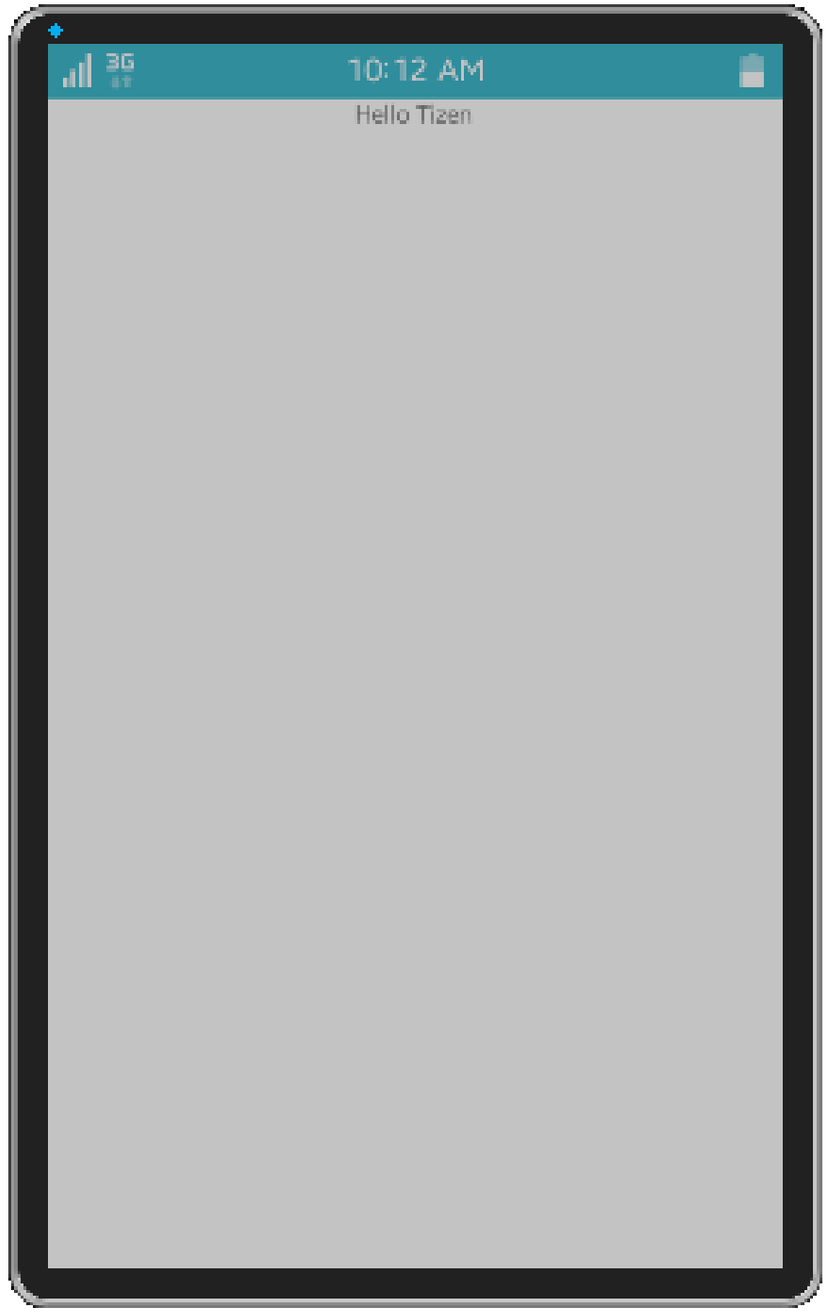
Goal

Good job !

You finished creating project, build and run of the Native UI Application.

It was very easy with Tizen Studio.

Click

# Implementation of Basic Wearable Application

# Tizen Native Application – Wearable Basic

Now, let's create a Basic UI project for wearable together.

# Tizen Native Application – Wearable Basic

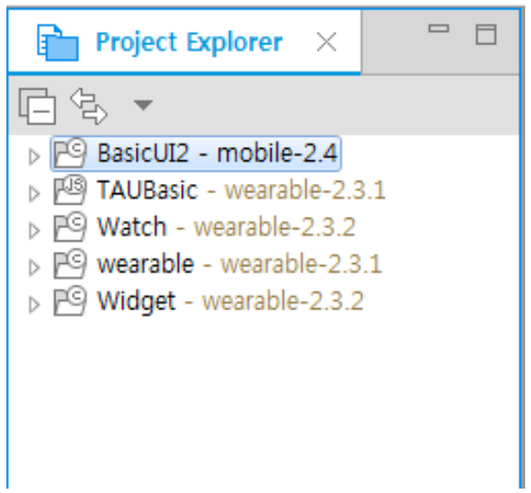
We will proceed the implementation of wearable basic UI app in 4 stages.

**Stage 1.**
**Create Project in Tizen Studio**

**Stage 2.**
**Create Emulator for test**

**Stage 3.**
**Install and Launch the Project**

**Stage 4.**
**Customize the Project**

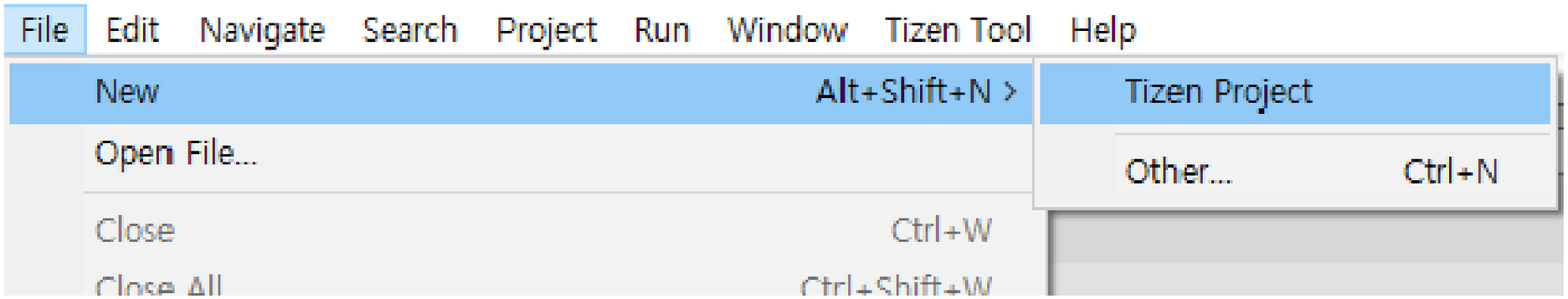





# Stage 1: Create Wearable Basic Project

Goal

Let's create Project for Wearable Native UI Application with Tizen Studio.

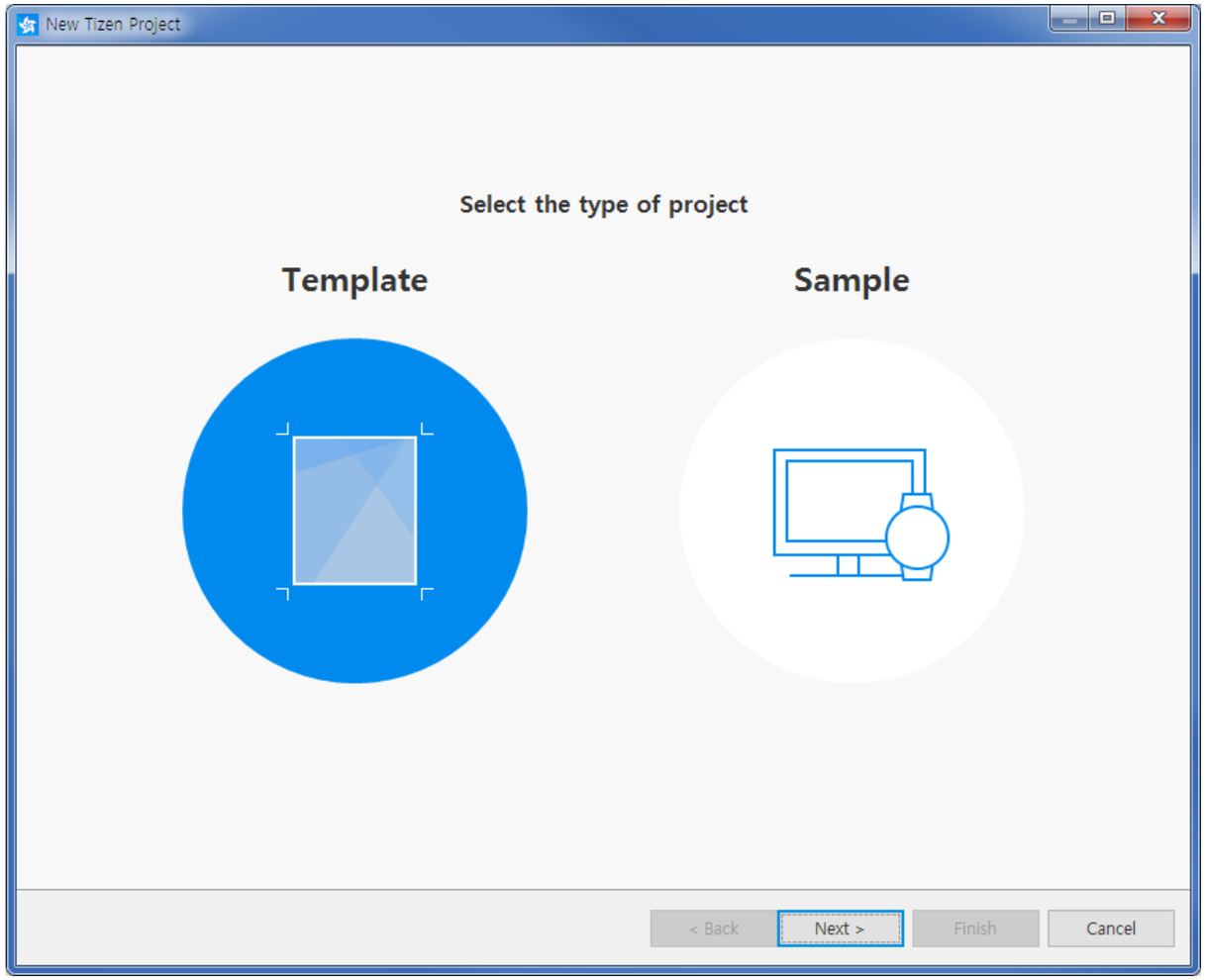


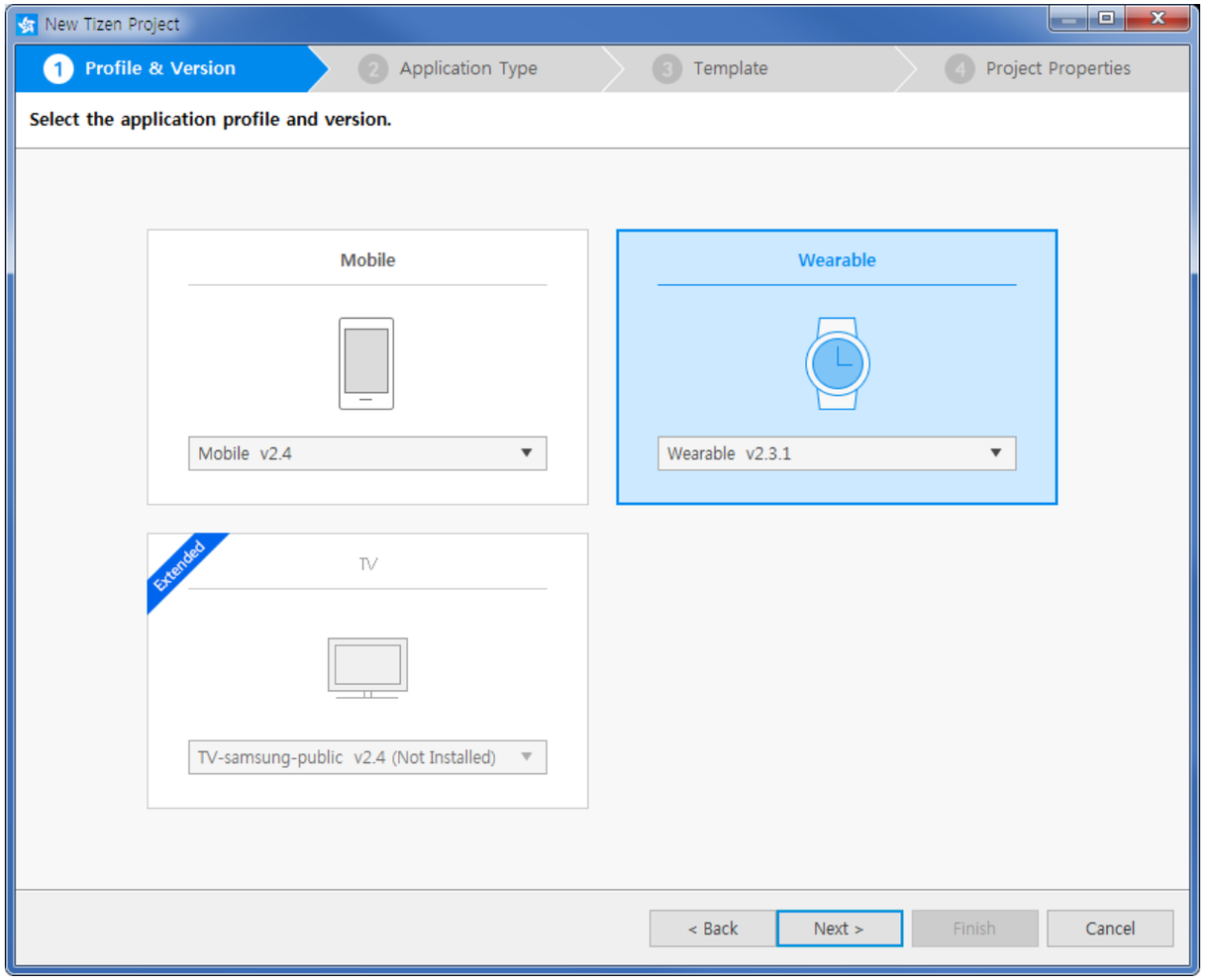
# Stage 1: Create Wearable Basic Project

Goal

Tizen Studio provide templates for various profiles i.e., Mobile, Wearable etc.

Choose Wearable and version 2.3.1.

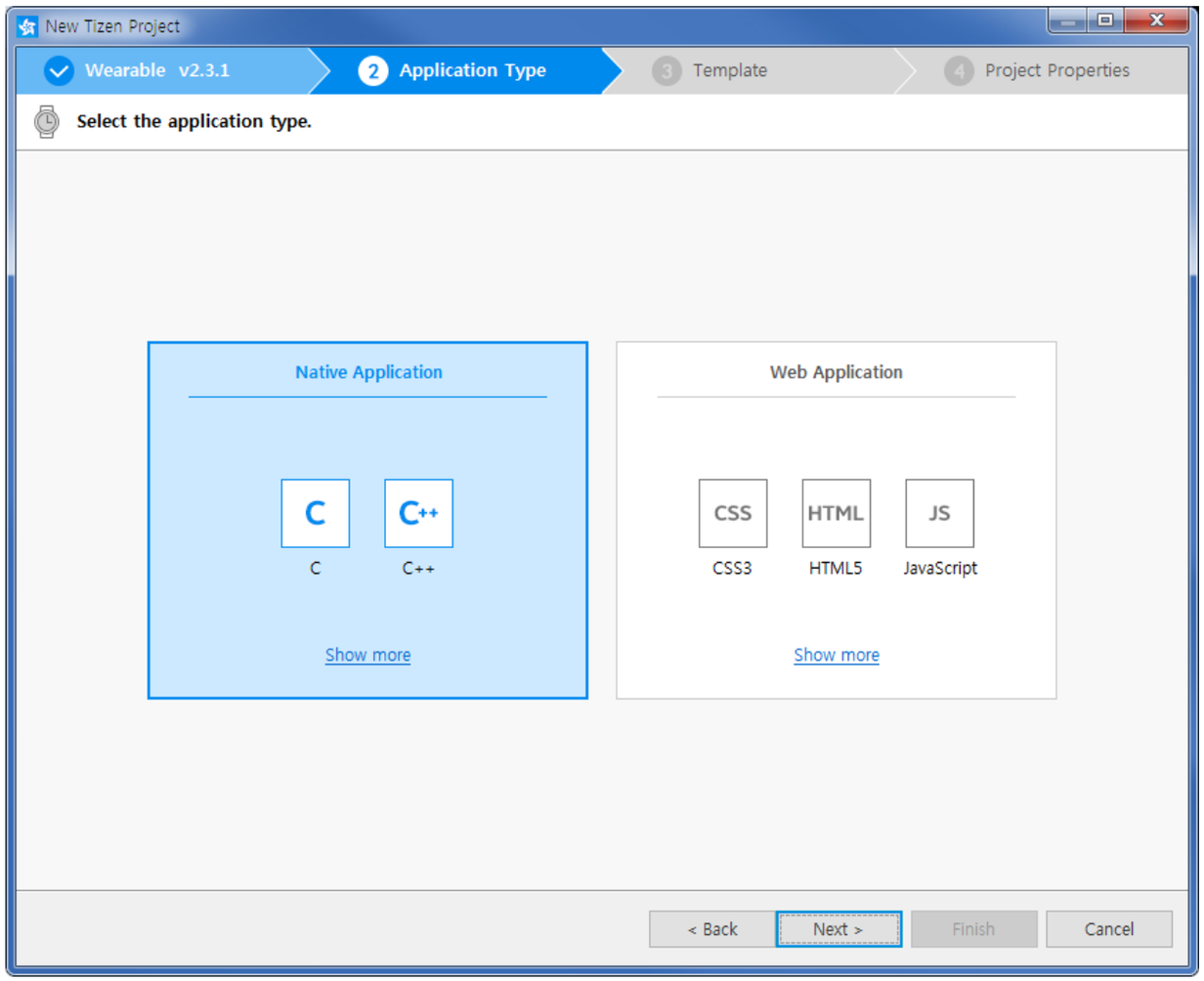


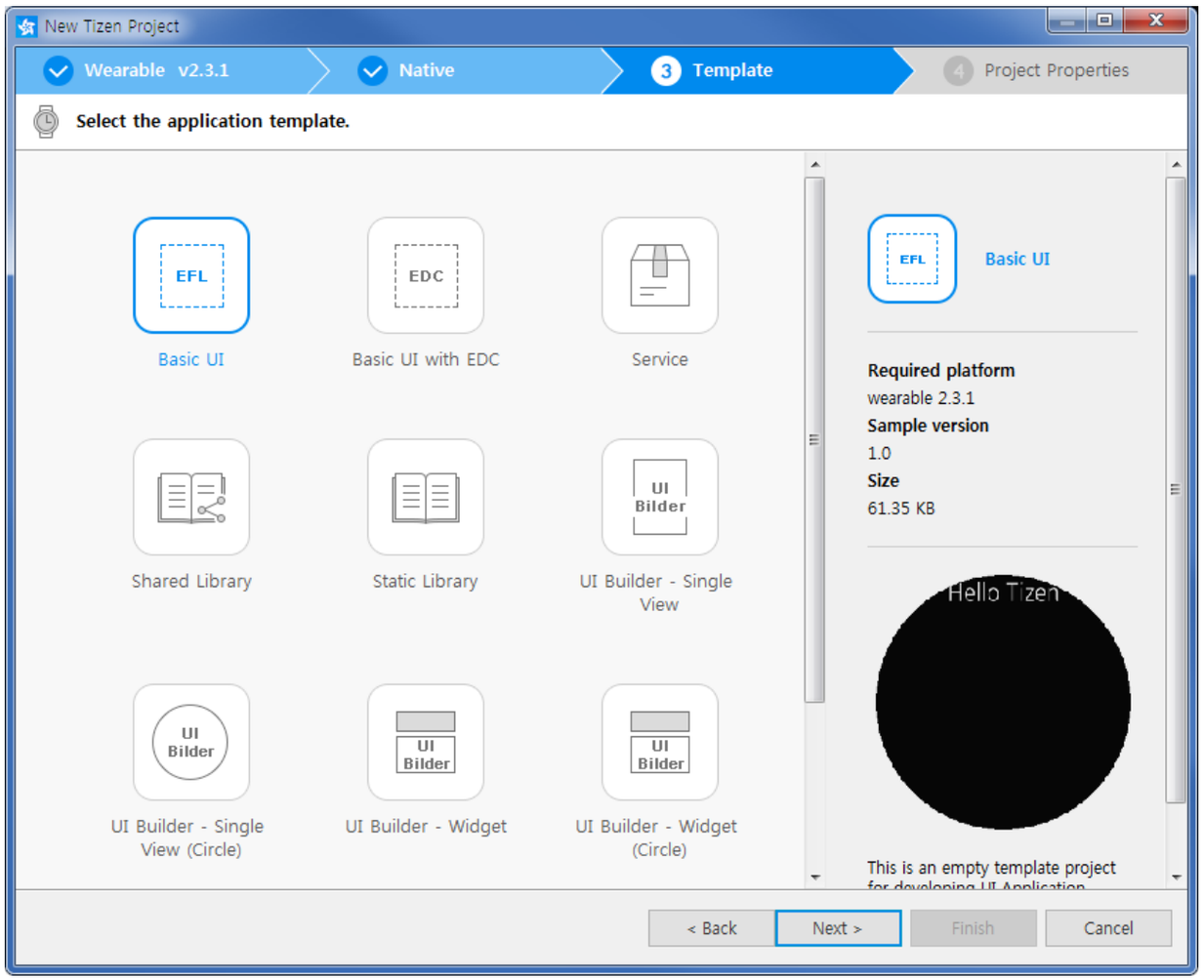



# Stage 1: Create Wearable Basic Project

Goal

## Choose Native Application and Basic UI.

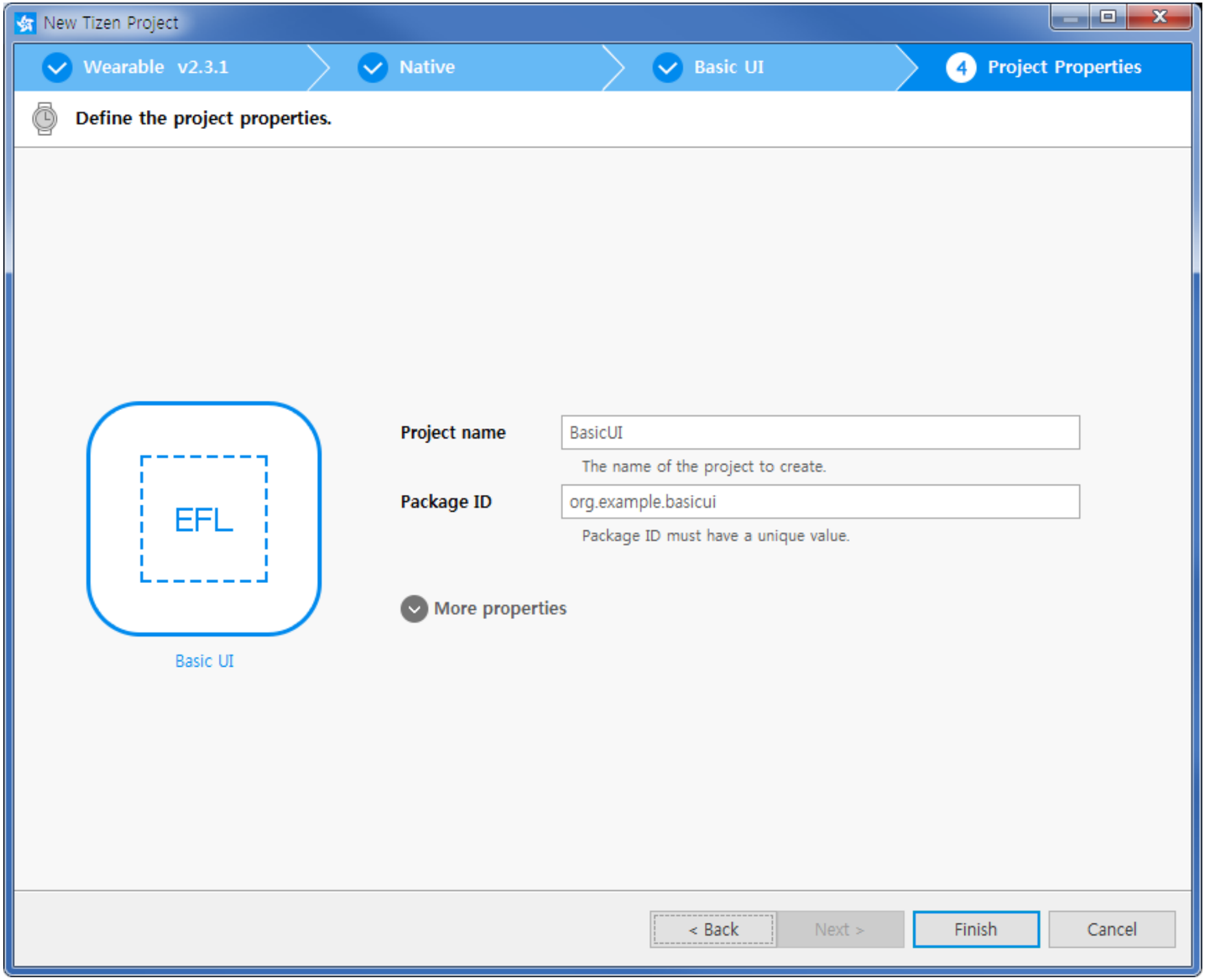


# Stage 1: Create Wearable Basic Project

Goal

Also, you can change the name of project, and this will affect to the app name



# Stage 1: Create Wearable Basic Project

Goal

Now, you can find your Project on the Project Explorer

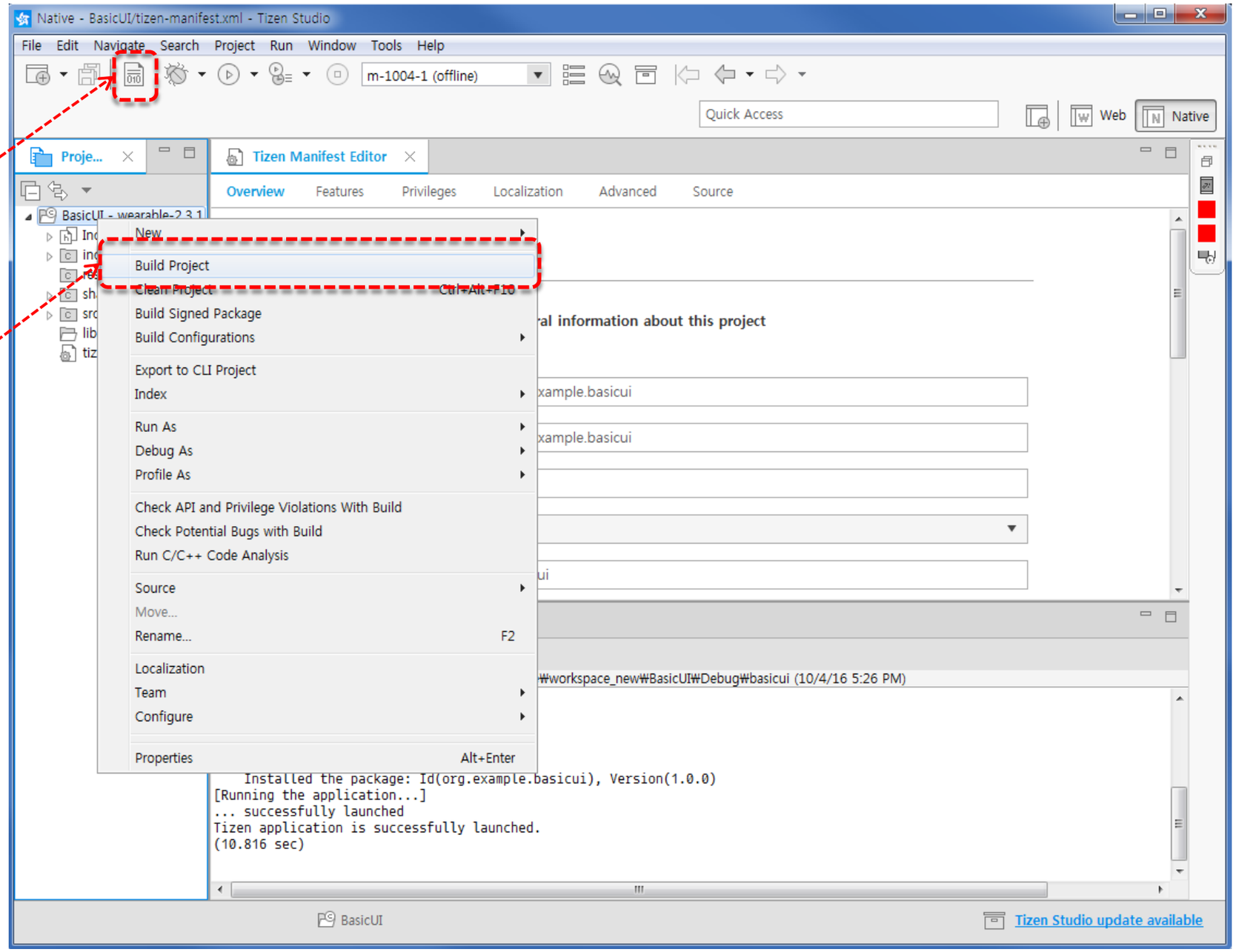
To build this project, Two methods are usually used

**Click the Project**

**Click Build Icon**

**Right Click on the Project**

**Build Project**

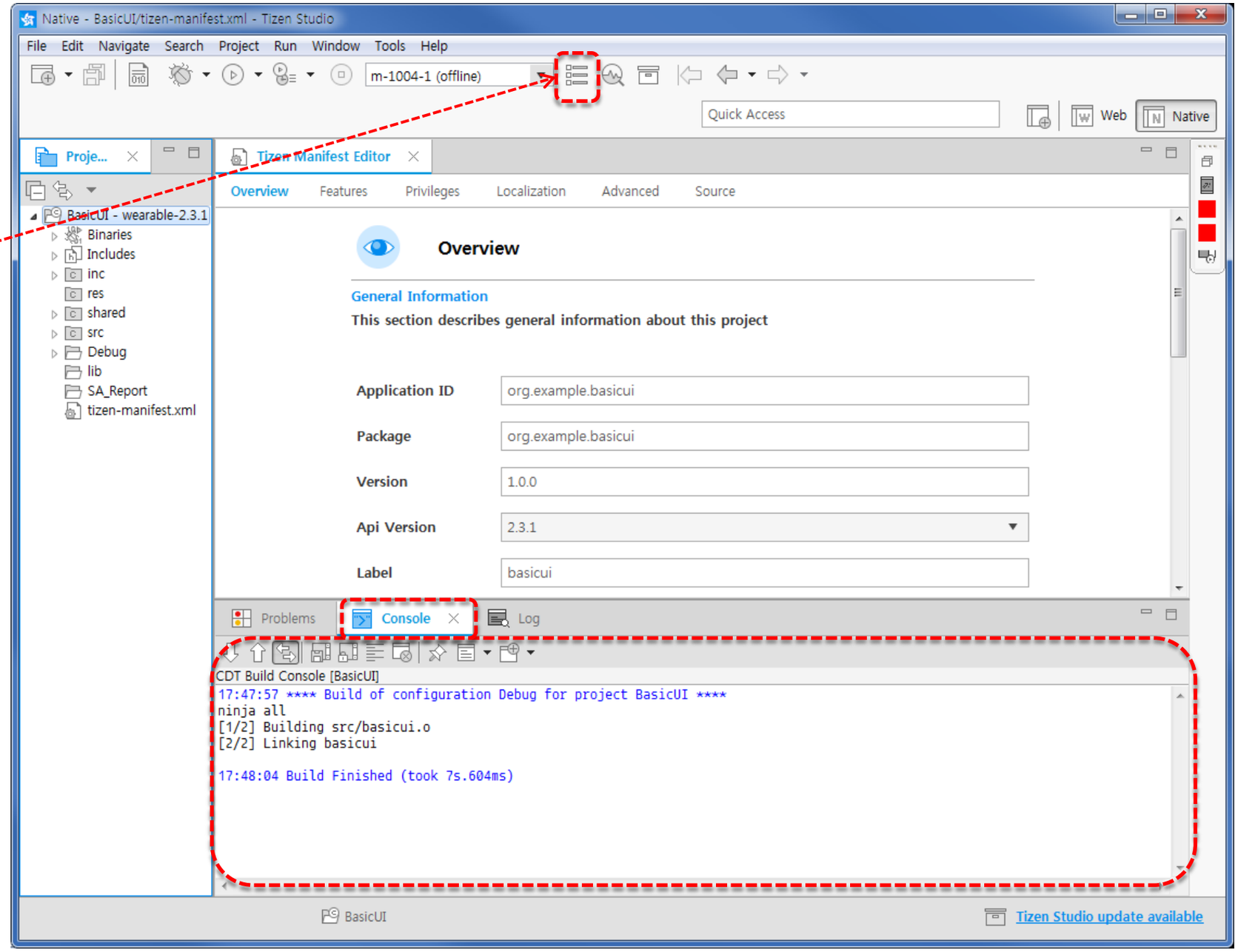


# Stage 2: Create Emulator for test

Goal

You can observe the progress of build through the Console page
In this page, also you can find error & warning messages
Create [Emulator] to test your project

**Click the Emulator Icon Here**



# Stage 2: Create Emulator for test

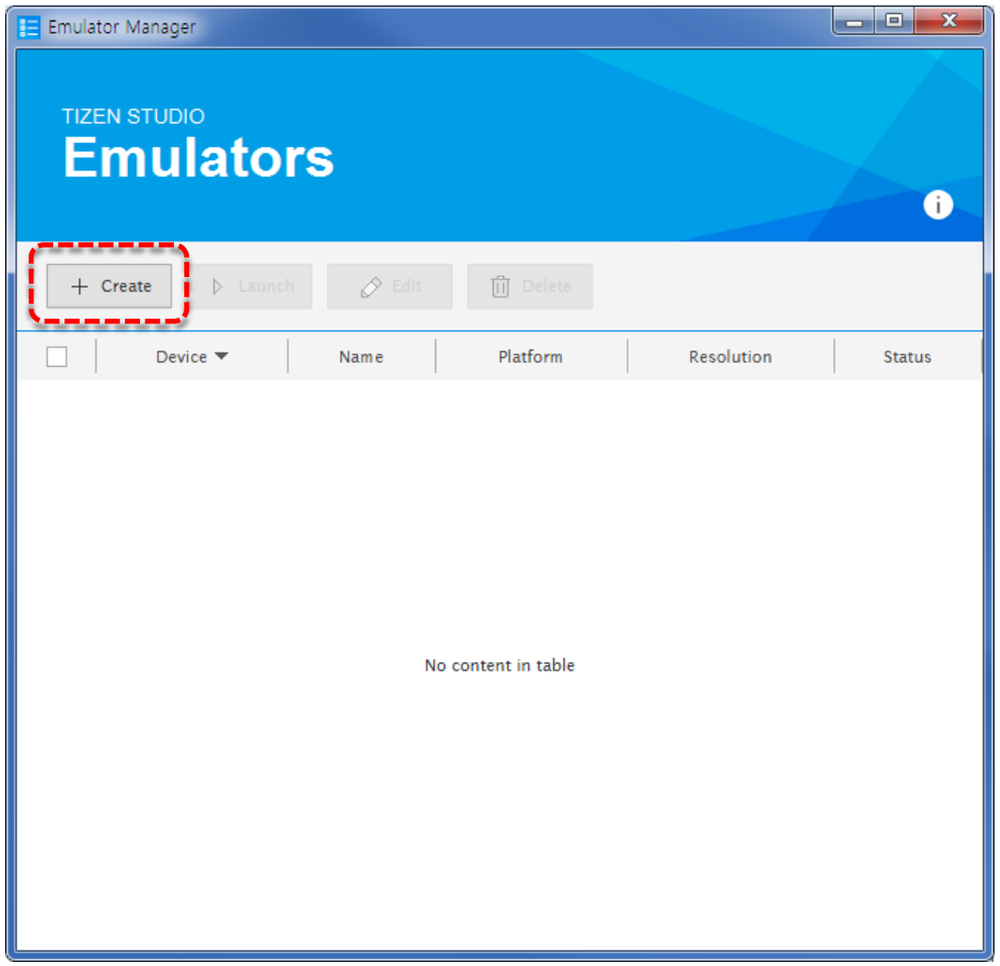
Goal

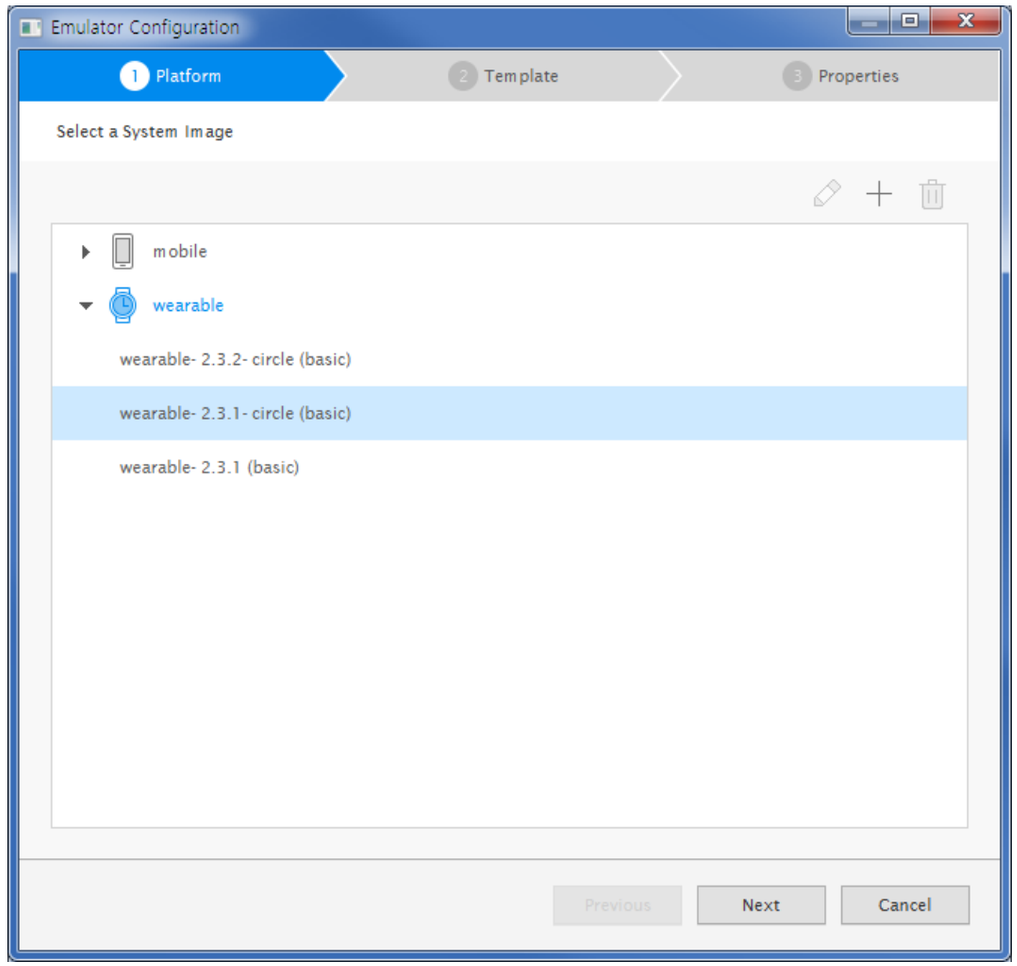
Tizen Studio provide Emulators for various profile(now, mobile and wearable)

For our Wearable Project, choose wearable category

**Click [+ Create]**

**Choose [wearable]**



# Stage 2: Create Emulator for test

Goal

Change the name of [Emulator] if you want

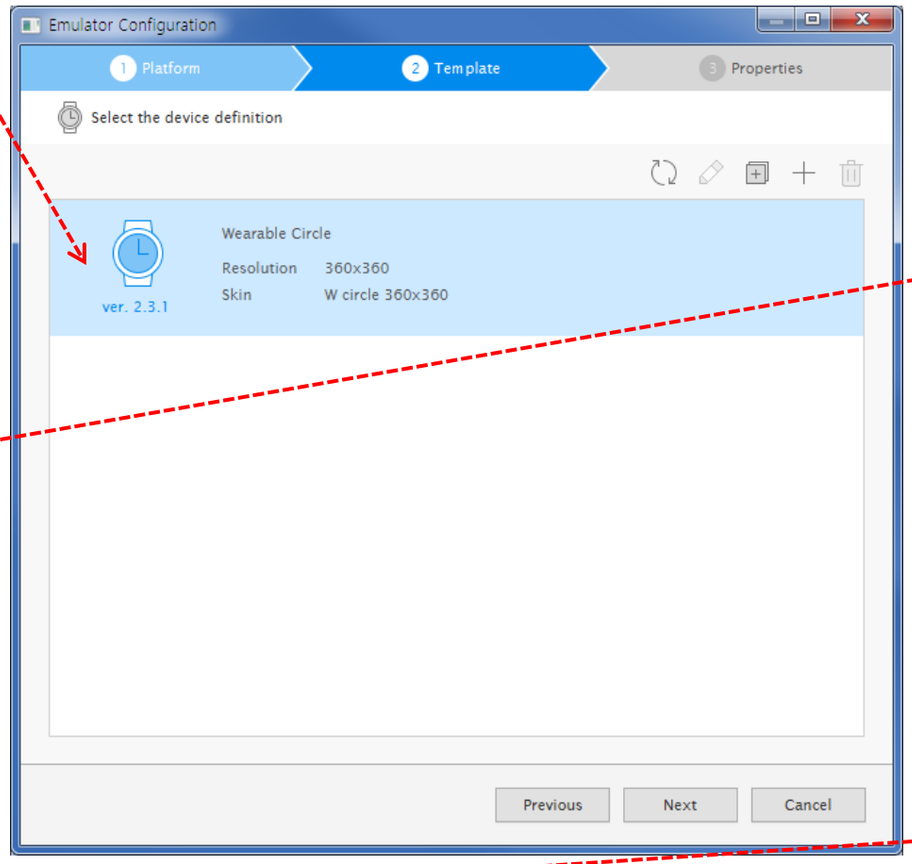
You also can choose Platform version

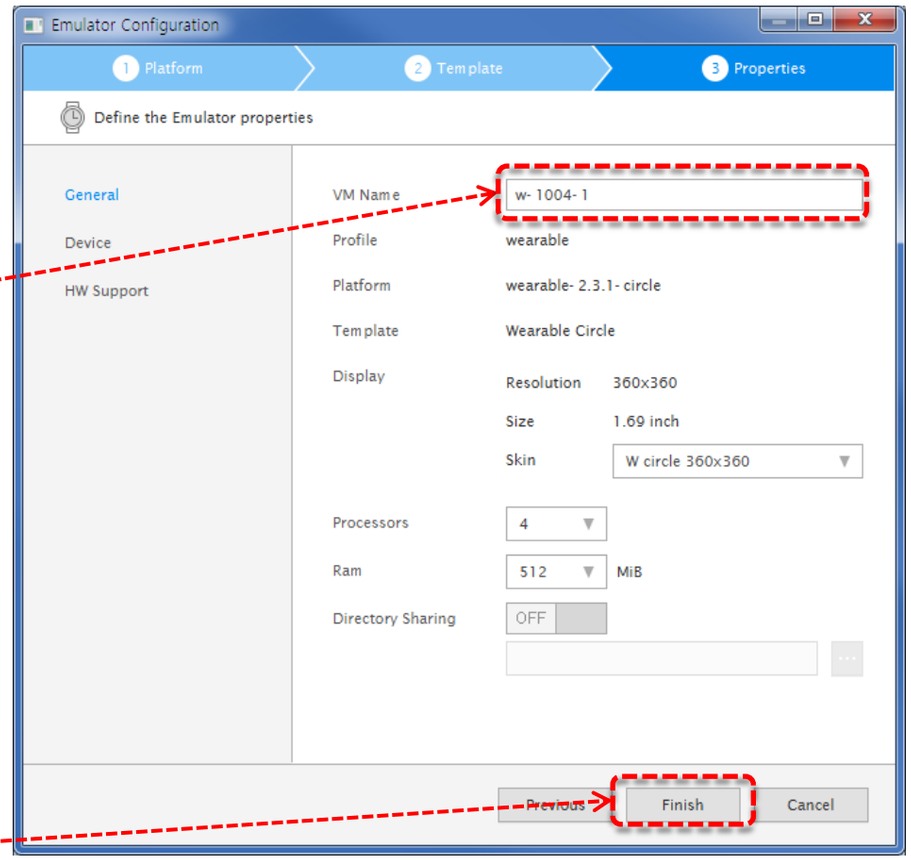
Each version provide different resolutions

**Choose Platform version (Circle is available on Tizen 2.3.1)**

**Check the name of Emulator**

**Click [Finish]**



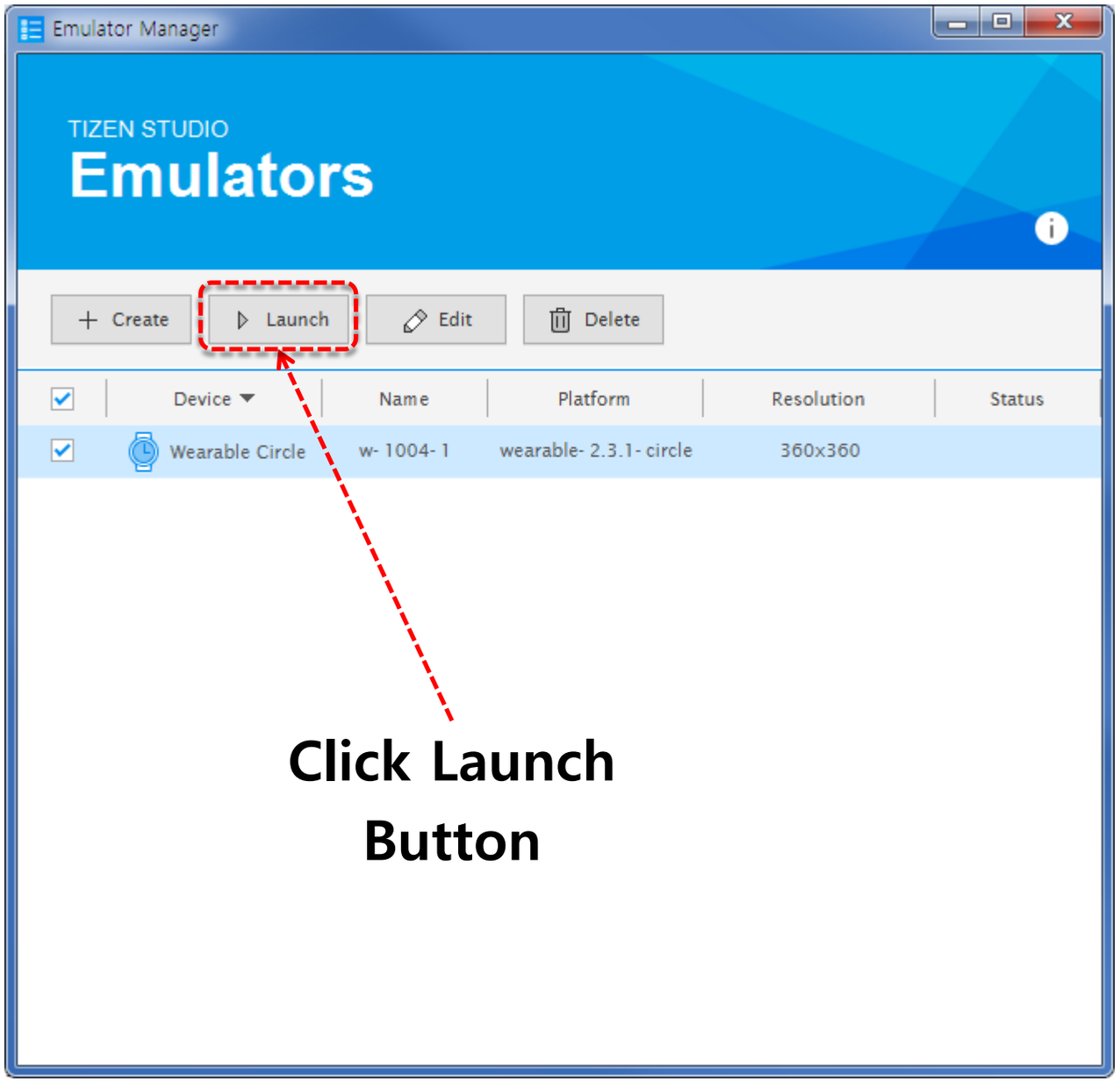



# Stage 2: Create Emulator for test

Goal

Click Play Button to launch Emulator

You can find Default Wearable(Circle) Emulator on the screen





**Now,**
**Let's launch your Wearable project on the Emulator**

# Stage 3: Install & Launch the Project

Goal

To Run(Launch) your Project, just follow the sequence like below
'Run' will install the project and launch the project automatically

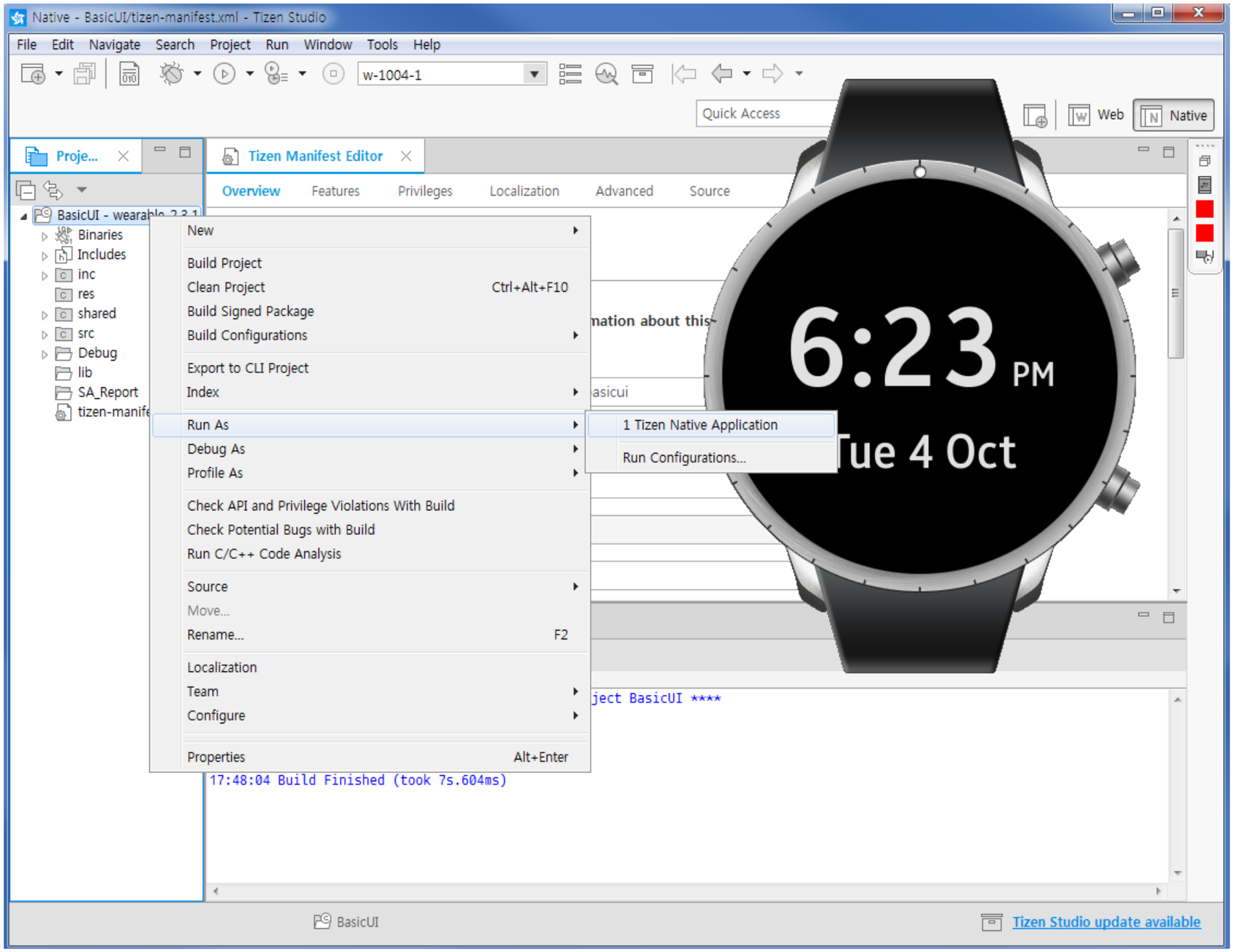
**Right Click on BasicUI Project**

↓

**Choose Run As**

↓

**Choose Tizen Native Application**



# Stage 3: Install & Launch the Project

Goal

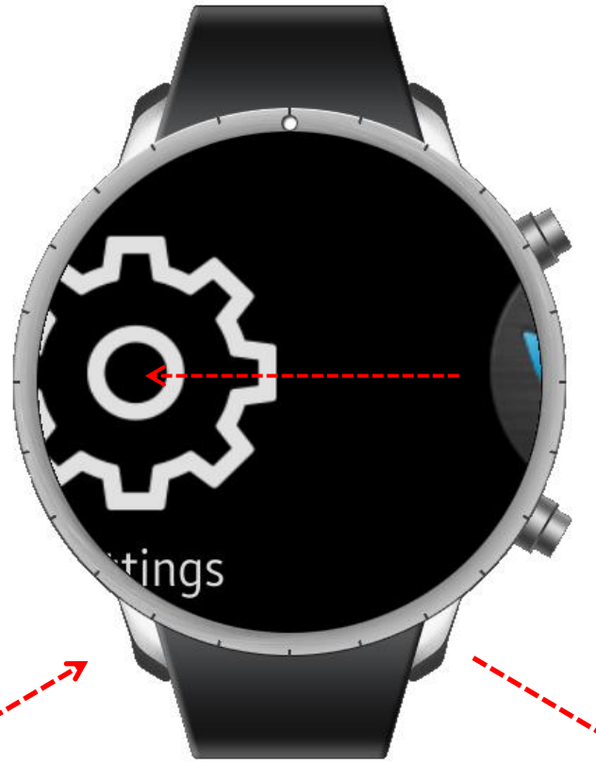
You can find ‘Hello Tizen’ on the black background

To find Icon of the project follow the sequence below


Back key

Home key

**Click Back or Home button**




**Scroll to the left**


**Swipe Up**


**Find ‘wearable’ Icon**

# Stage 4: Customize the Project

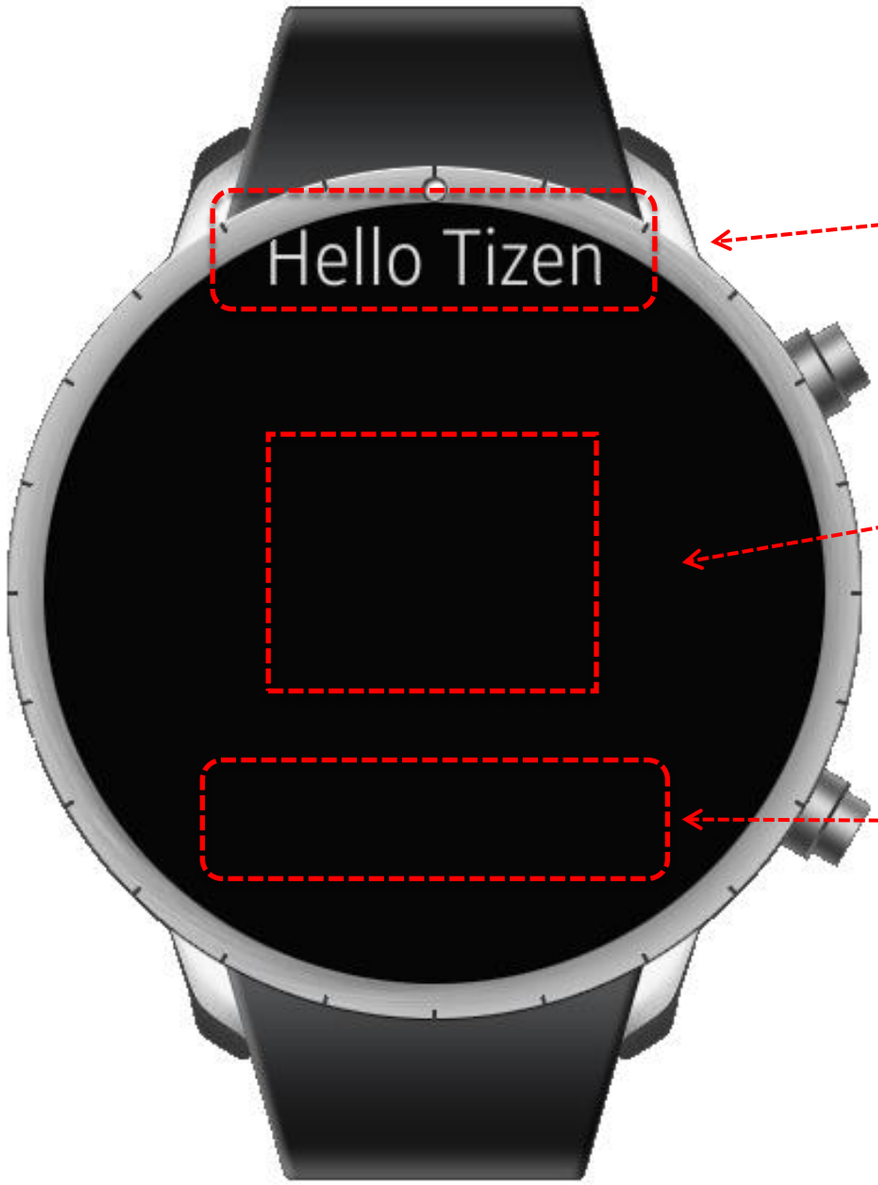
Goal

Good job !

You finished creating project, build and run of the Wearable Application.

It was very easy with Tizen Studio.

At this time, we make our own Wearable Application using this project.




**Change the ‘Hello Tizen’**

**Add rectangle**

**Add another Text**

**Look into the source file!!**

# Stage 4: Customize the Project

Goal

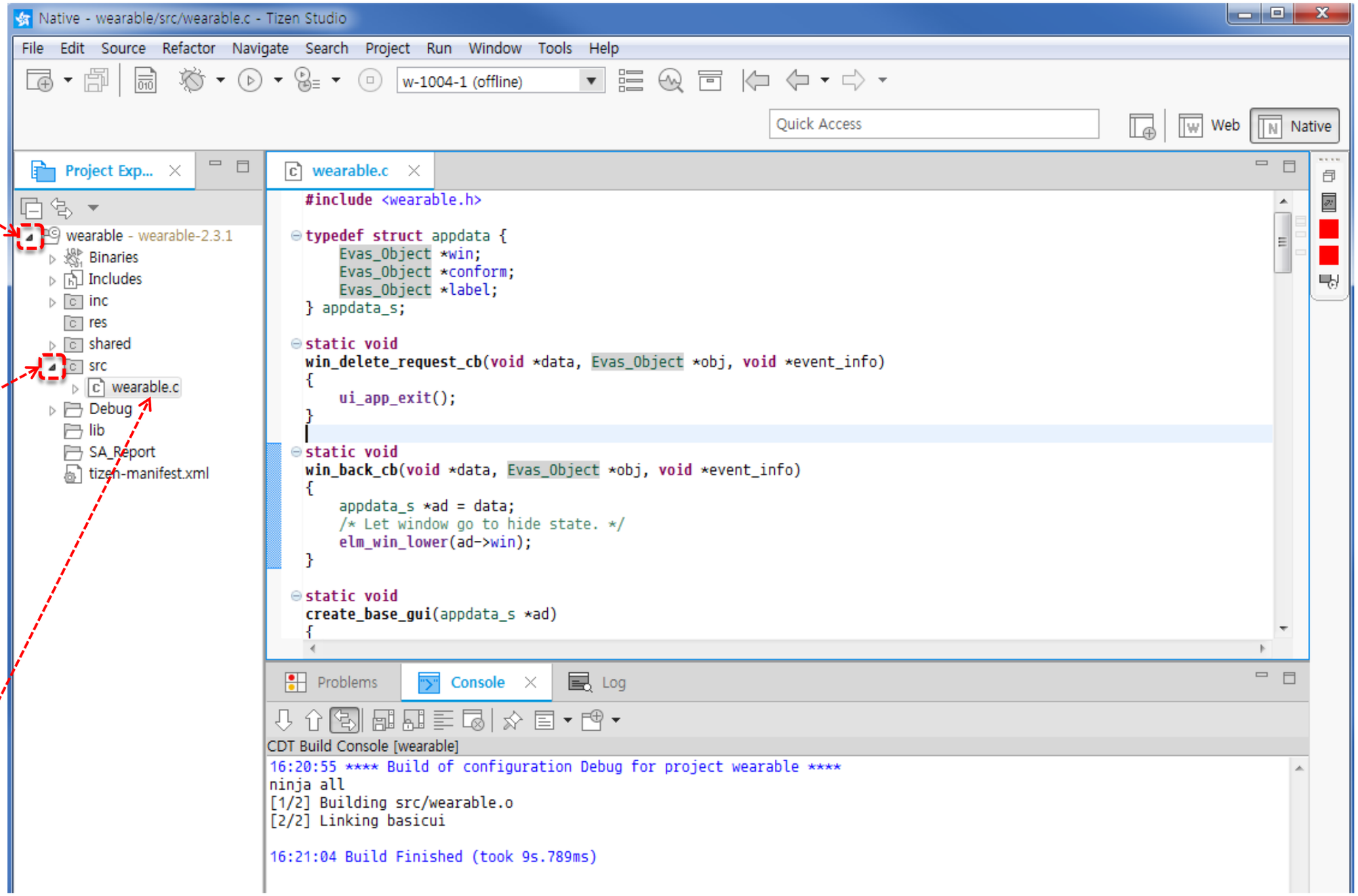
Double click on ‘wearable.c’ file

You can find source code on the right

**Click ‘Wearable’ arrow**

**Click ‘src’ arrow**

**Double Click**



# Stage 4: Customize the Project

Goal

Find ‘create_base_gui’ function in the ‘wearable.c’ file

*This function makes the view of the project like right side*

```c
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_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 Tizen</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);
}
```

# Stage 4: Customize the Project

Goal

The Objects(like window, conformant, label…) will be explained in next slide

Please just follow the instruction this time

```c
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_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 Tizen</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);
}
```

→ Recommend do not change (This form is standard)

→ Conformant is pre-formed layout (It has several empty parts to put the object like label, content, indicator etc)

→ This label is for ‘Hello Tizen’ text on the screen

# Stage 4: Customize the Project

Goal

Let's change the text 'Hello Tizen'

Add Rectangle like below

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

elm_object_text_set(ad->label, "<align=center>Good Luck</align>");

evas_object_size_hint_weight_set(ad->label, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);

elm_object_content_set(ad->conform, ad->label);

/* Rectangle */
Evas_Object *rect = evas_object_rectangle_add(evas_object_evas_get(ad->win));

evas_object_resize(rect, 100, 100);

evas_object_color_set(rect, 100, 0, 0, 100);

evas_object_move(rect, 130, 130);

evas_object_show(rect);
```

'Hello Tizen' > 'Good Luck'

RGBA value


**Some objects require certain parent.**
**'rectangle' object should be added to the 'Evas'**

# Stage 4: Customize the Project

Goal

Finally add another Text at the bottom

*Text object also require 'Evas' as a parent*

```
/* Text */
Evas_Object *text = evas_object_text_add(evas_object_evas_get(ad->win));

evas_object_text_text_set(text, "Fly High !!");

evas_object_text_font_set(text, "DejaVu", 30);

evas_object_color_set(text, 255, 255, 0, 255);

evas_object_move(text, 120, 300);

evas_object_show(text);
```

*Font style & size*


**Wearable default window size is 360x360.**
**There are some APIs for the certain object like 'evas_object_text_text_set ()'.**
**If you know what APIs are related to the object,**
**you can easily develop Tizen Native Application.**

# Deep Learning about Tizen Native UI Framework

# Understanding of Native UI Framework – EFL

Especially, Tizen Native Application is implemented by EFL

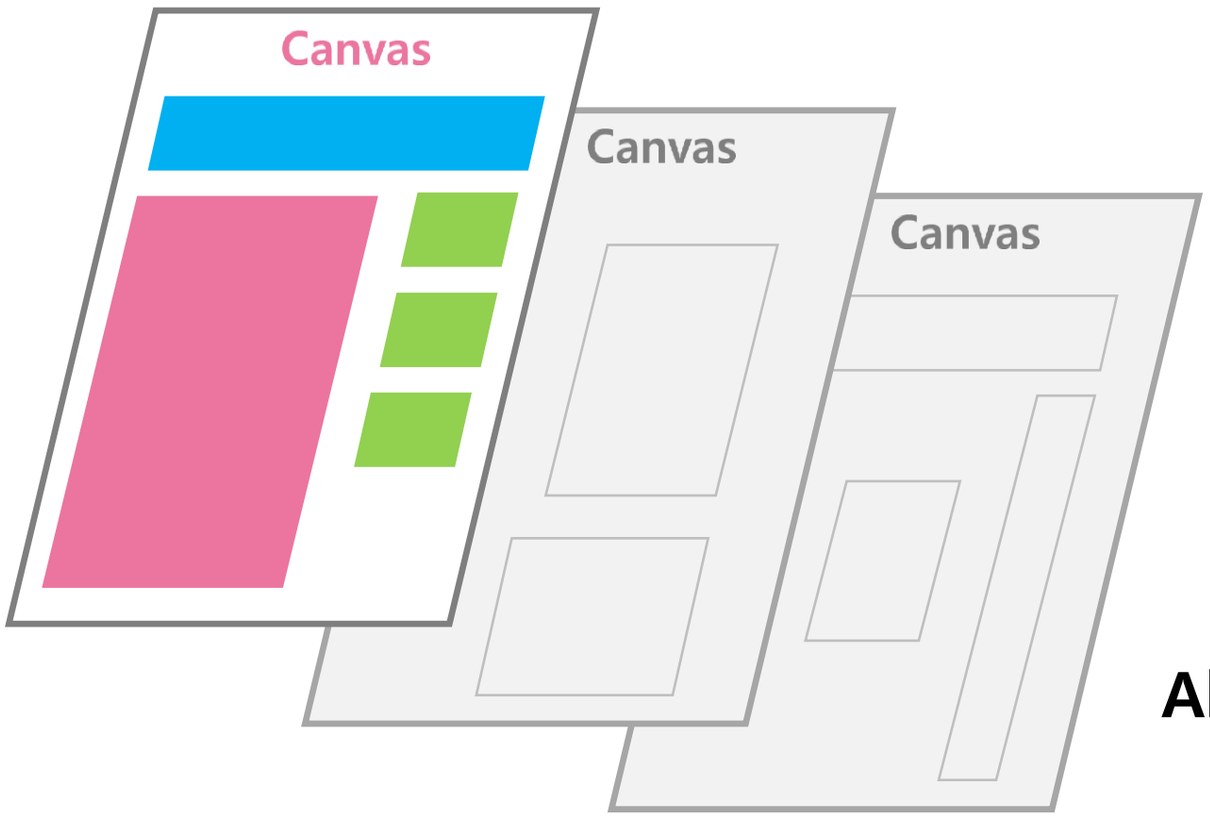
**The Objects you used before, like window, conformant, rectangle, and label are provided by EFL**


Then, What is the EFL ?

# Understanding of Native UI Framework – EFL

Tizen Native Development is like a drawing on the window

But to draw something, so many things are required and it is very complex




**To make drawing easy,**
**EFL provide Simple Method**

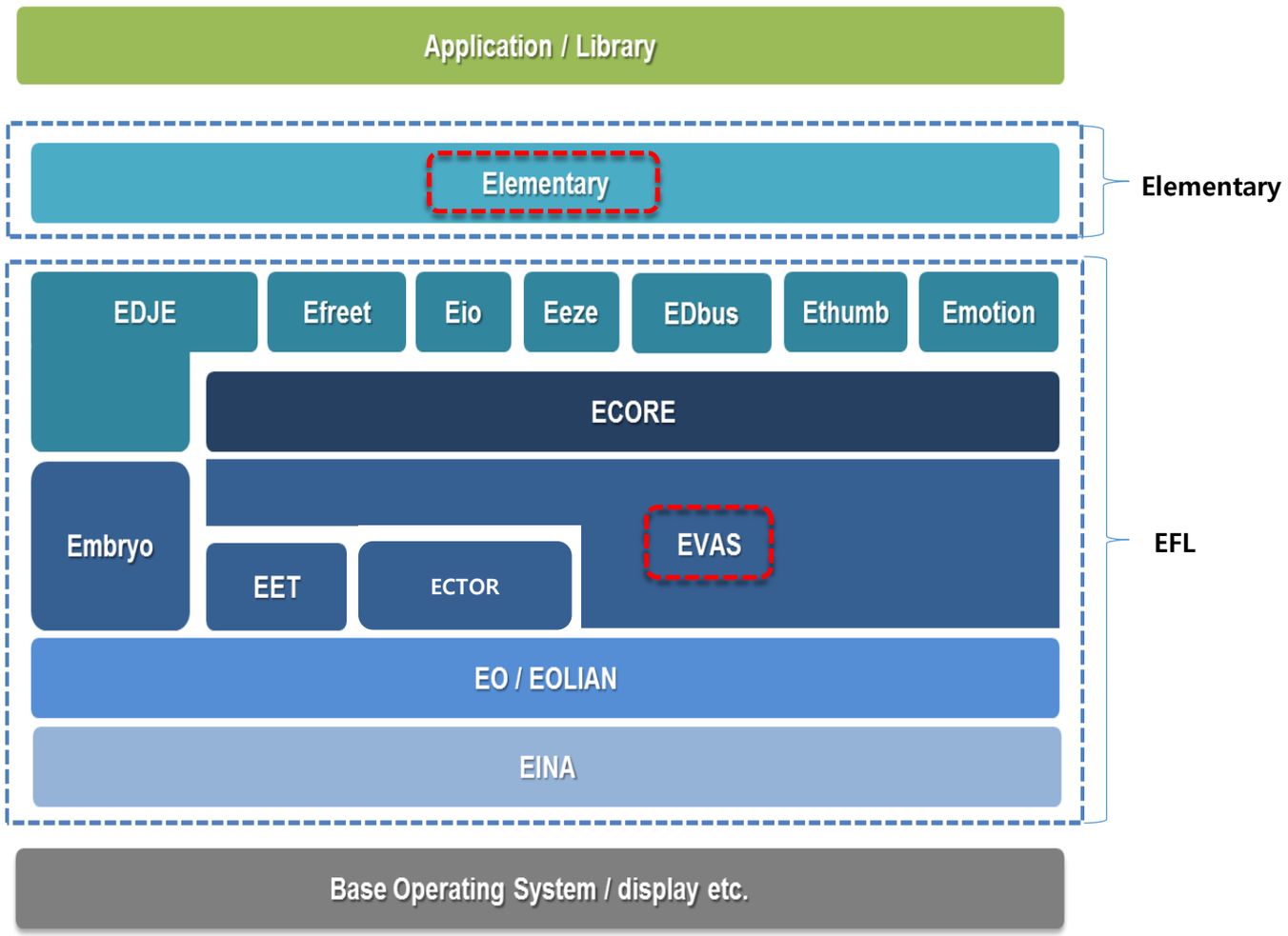
**So, EFL can be called as a**
**'The set of Graphical**
**User Interface Toolkit Library'**

**Also, it provides complete component like button, image and check box, makes development more visual and convenient**

# Understanding of Native UI Framework – EFL

EFL is made up of so many parts like below

With these parts, EFL offer many advantages for Tizen development




**In this Class, we'll look into two parts**

**Most used parts, EVAS and Elementary**

**EFL provide advantage of**

- **GUI**
- **Theme**
- **Animation**
- **User Input Event**
- **Various profile environment**
- **IPC/Socket Connection**
- **3D Graphic**
- **Video/Sound Output**

# Understanding of Native UI Framework – EFL



## EVAS is Canvas and Rendering Engine

**Rendering**
**based on Scene Graphic**

**Tracking all objects that are able to be displayed on the screen**

**Supervise screen output of the objects(Font, image loading, blending, scaling etc.)**

**Partial rendering: Only updated part be rendered and not visible part rendered though it exists on the screen**

evas_object_color_set

evas_object_text_font_set

evas_object_image_file_set

evas_object_scale_set

evas_object_resize

evas_object_move

evas_object_show

evas_object_hide

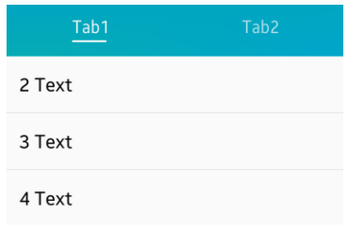
**Using these APIs, Control the Output of the screen**

# Understanding of Native UI Framework – EFL

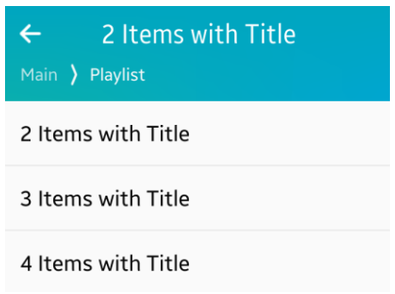
Goal

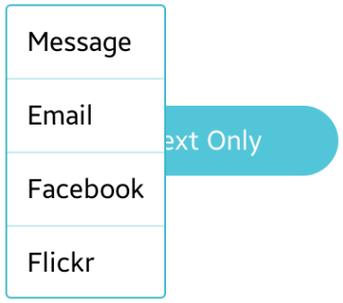
Elementary is more visual and kind

Components frequently used on development are provided as completed form



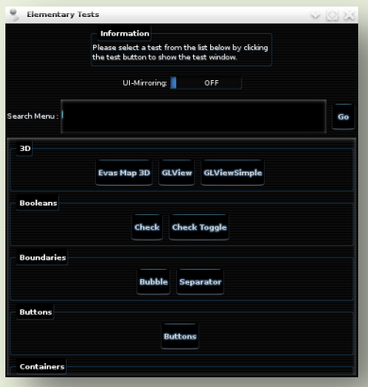



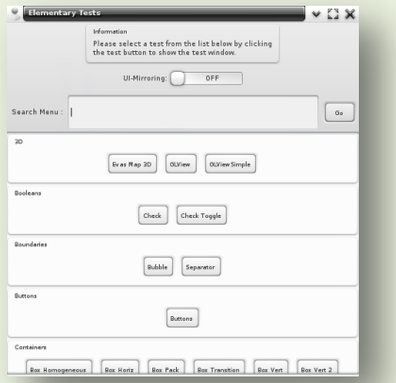




**Not only visual element(button, checkbox, image etc), but also non visual element(container: scroller, table, box etc) are provided**

**Screen flexibility: Ensure profer scalability according to the resolution**

**Theme: Using various Theme, different Look & Feel can be shown on the same component**



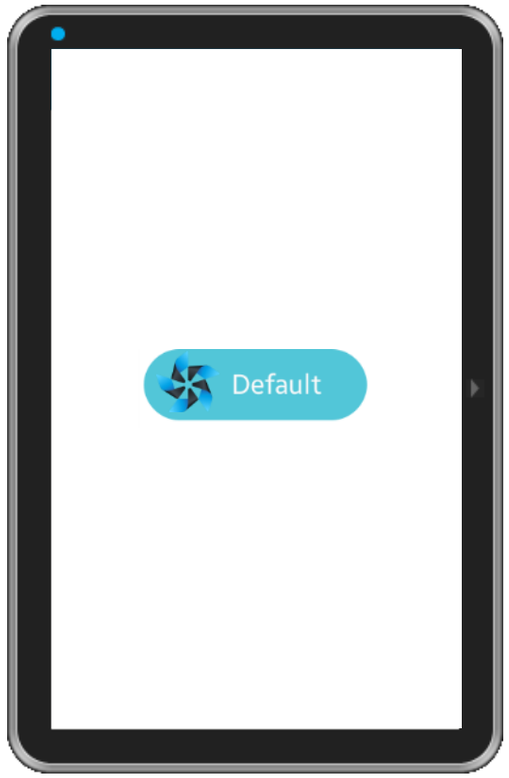



# Understanding of Native UI Framework – EFL

Elementary involves several parts of EFL like Evas, Edje, Ecore etc…
This means that Elementary do not provide only the shape of component but also operation, theme and scale etc

```
void create_base_gui()
{
   /* Window */
   Evas_Object *win = elm_win_util_standard_add(PACKAGE, PACKAGE);

   /* Button */
   Evas_Object *btn = elm_button_add(win);
   elm_object_text_set(btn, "Default");
   evas_object_smart_callback_add(btn, "clicked", btn_clicked_cb, NULL);
   evas_object_move(btn, 150, 300);
   evas_object_resize(btn, 400, 350);
   evas_object_show(btn);

   /* Image */
   Evas_Object *img = elm_image_add(btn);
   elm_image_file_set(img, "icon.png", NULL);
   elm_object_content_set(btn, img);

   evas_object_show(win);
}
```

Elementary APIs start with 'elm'




**Button is provided as set click event be available, familiar shape, be able to write text and icon and text position previously by Elementary**

# Understanding of Native UI Framework – EFL

So, using EFL is simple.
You just need to know what components are provided,
what API is related to them.

If you need more information, access to the below

- **Source in Tizen**
  - **https://review.tizen.org**
    - **EFL : platform/upstream/efl**
    - **Elementary : platform/upstream/elementary**

- **EFL Guides**
  - **https://developer.tizen.org/development/guides/native-application/user-interface/efl**

- **API reference**
  - **https://developer.tizen.org/dev-guide/latest/org.tizen.native.mobile.apireference/EFL.html (EFL)**
  - **https://developer.tizen.org/dev-guide/latest/org.tizen.native.mobile.apireference/Elementary.html (Elementary)**

# Understanding of Native UI Framework – Lifecycle

Goal

To develop your own Tizen Native Application, you need to know last one more
The Life Cycle of Tizen Native Application

**You can find ‘ui_app_lifecycle_callback’**
**in all of Native Application main source files**

**Don’t need to change this**

**Just know when these callbacks are called**

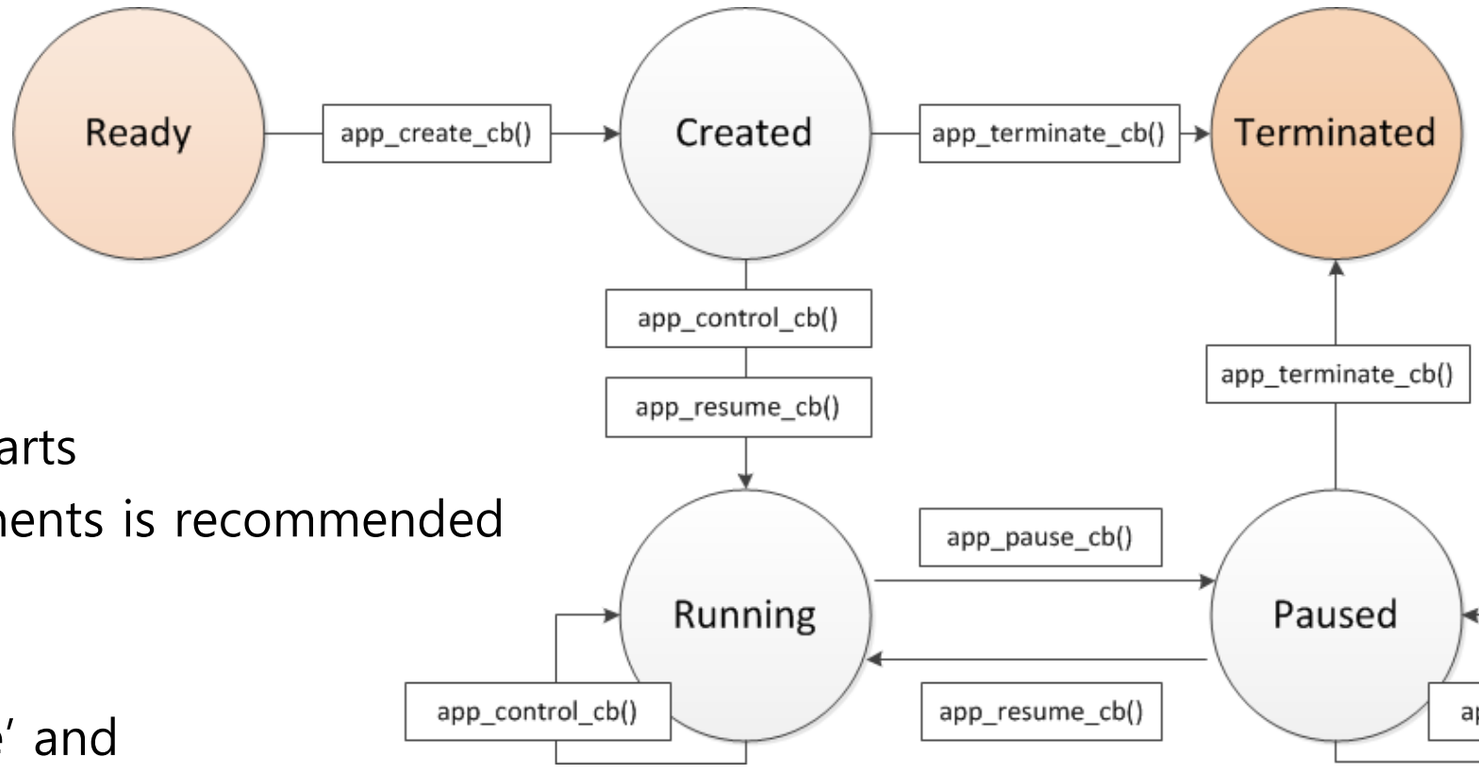
```
int
main(int argc, char *argv[])
{
        appdata_s ad = {0,};
        int ret = 0;

        ui_app_lifecycle_callback_s event_callback = {0,};

        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;

        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;
}
```

# Understanding of Native UI Frame Work – Lifecycle

There are five state of Native Application

These states are changed by Life Cycle Callback function like below

**Life Cycle**
**of Native Application**




**app_create:**

Called when the process starts

In this, Creating UI components is recommended

**app_control:**

Called after the 'app_create' and

when receive the launch request  on running state

from other process

**app_resume:**

Called when the window of the application is shown

**app_pause:**

Called when the window of the application is hide

**app_terminate:**

Called when the process of the application is terminating

and after the main loop quits

# Implementation of Watch Face Application

# Implementation – Watch Face UI Application

Let’s make a Watch Face UI Application
It is easy to develop anything you want if you are familiar with Tizen Studio
Follow up, and make your own Watch Face


**Tizen will provide wonderful experience on your development**

# Implementation – Watch Face UI Application

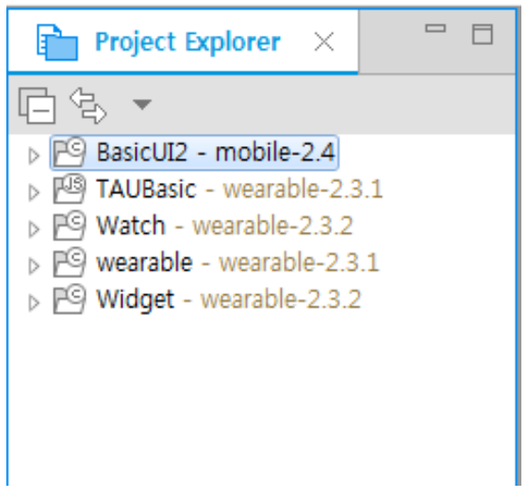
We will proceed the implementation of watch face UI app in 4 stages.

**Stage 1.**
**Create Project in Tizen Studio**

**Stage 2.**
**Create Emulator for test**

**Stage 3.**
**Create user interface layout**

**Stage 4.**
**Add operation to the watch layout**



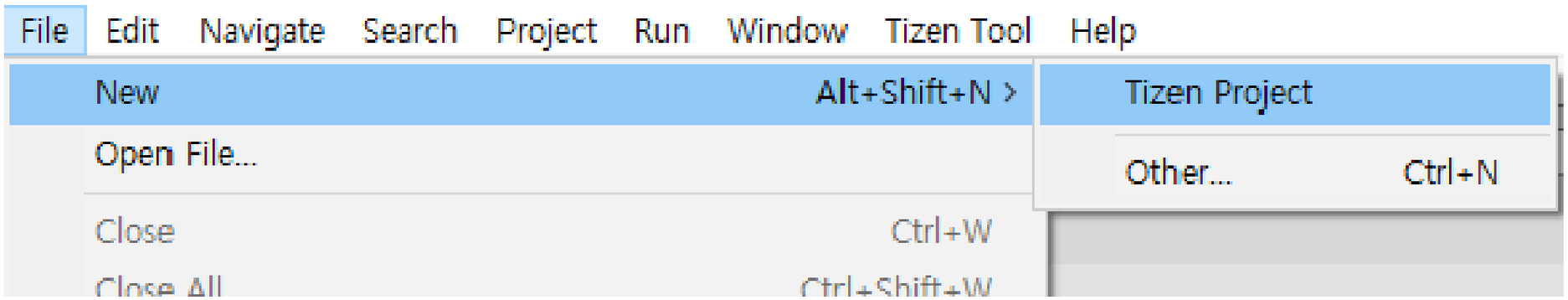



# Stage 1: Create Watch Project

To start Development,

Create New [Tizen Native Project] !

Tizen Studio provide some Templates for the easy start

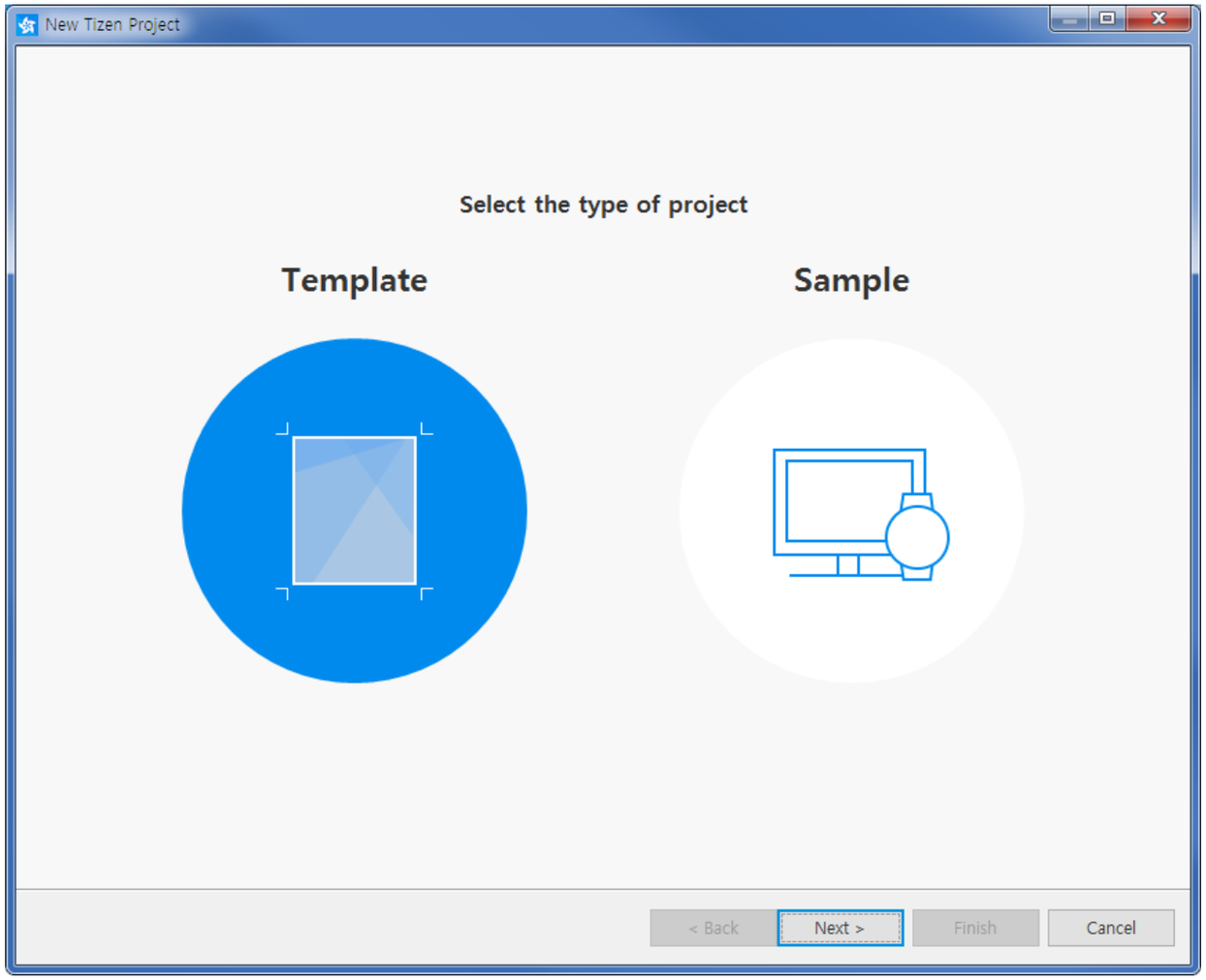


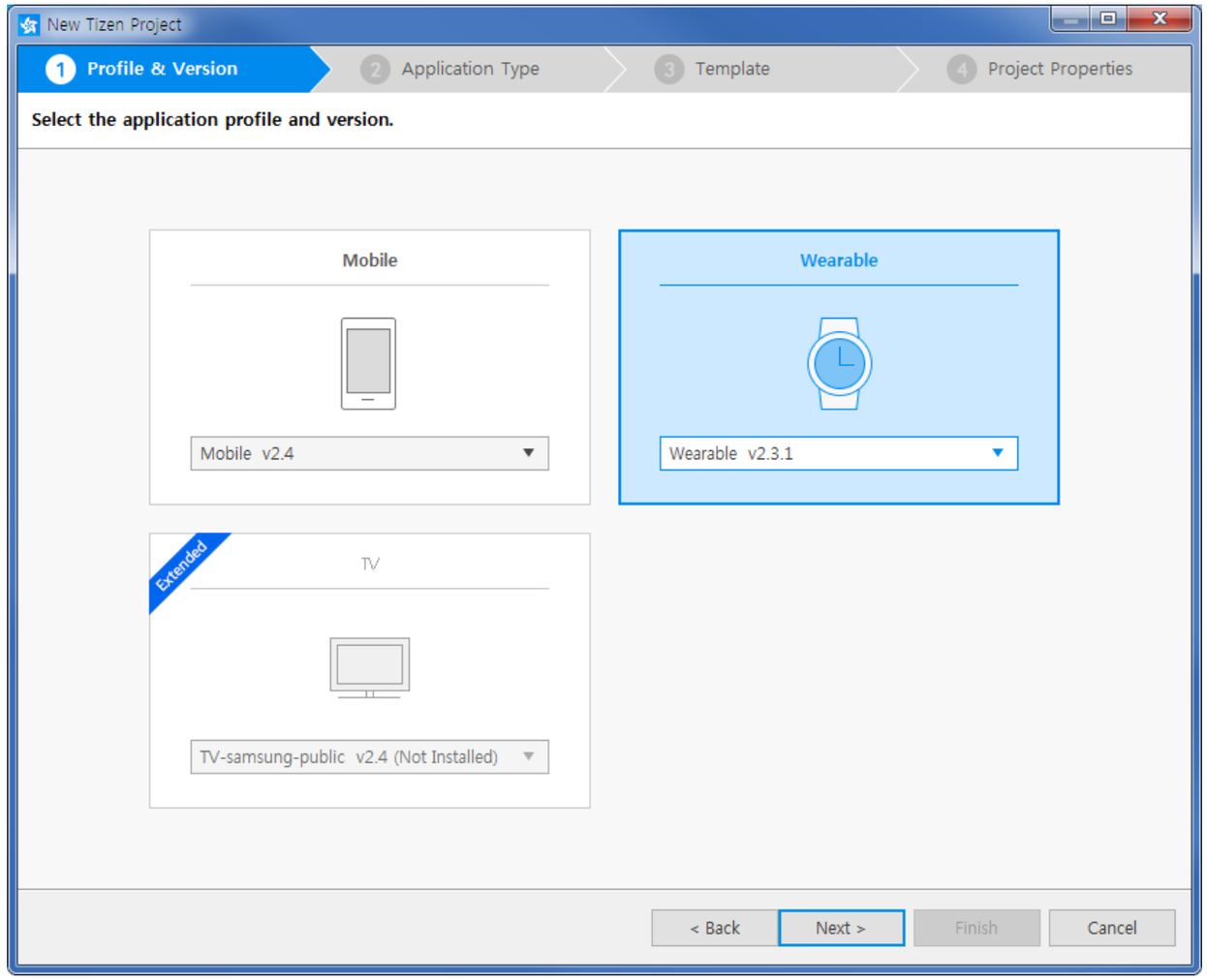
# Stage 1: Create Watch Project

Goal

Choose the Template most similar with what you want to develop

In this case, we'll choose Watch Template for Watch Face



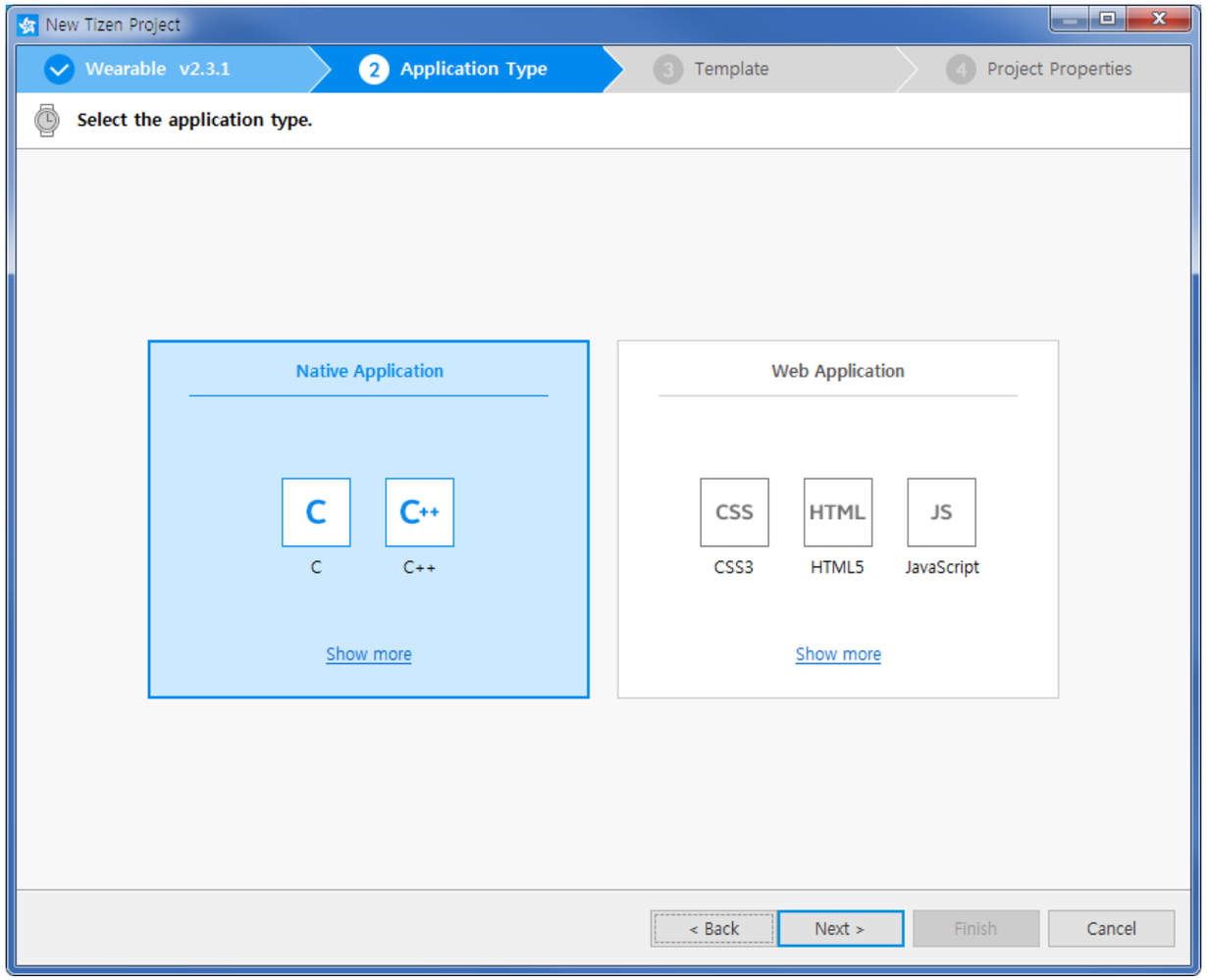



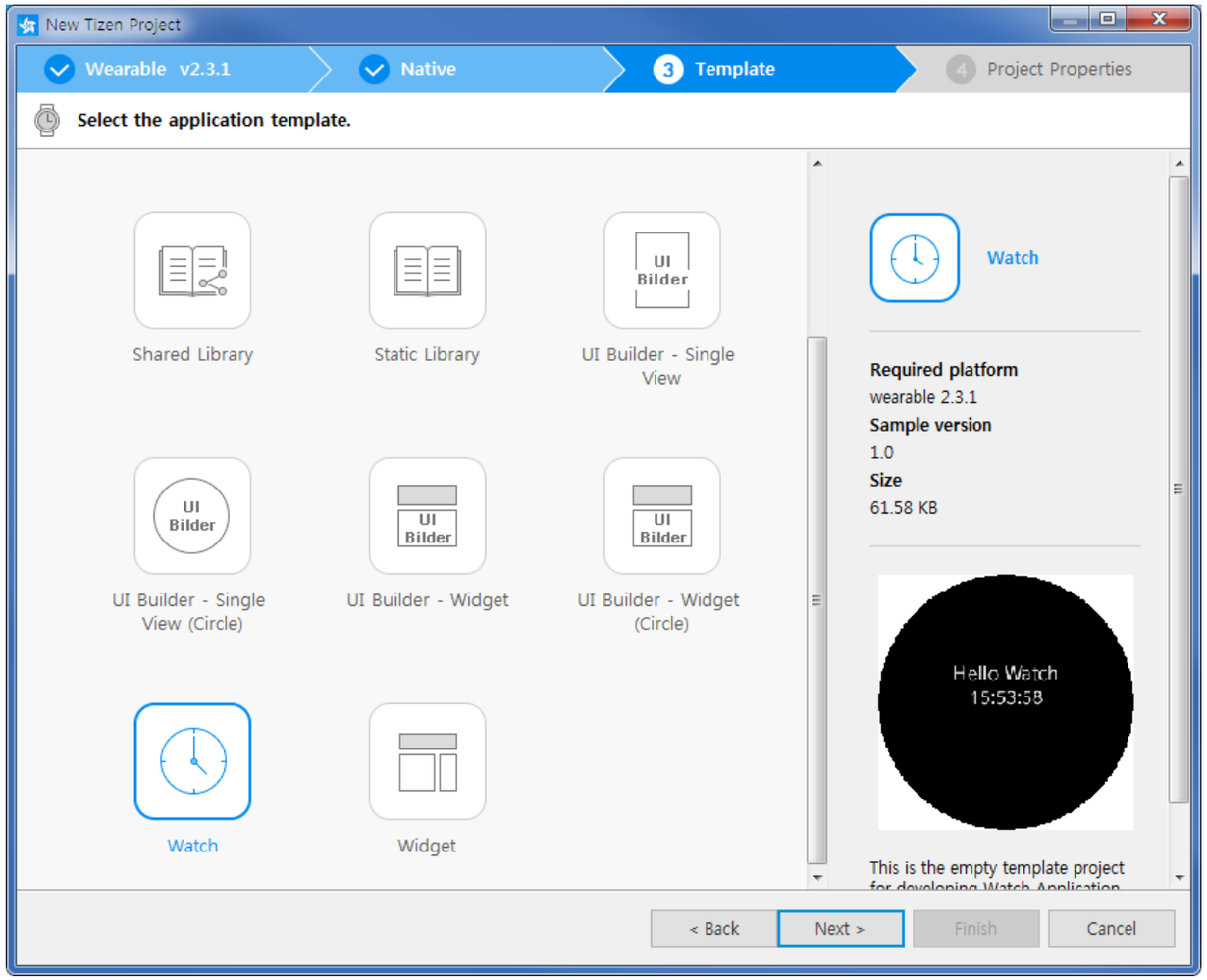
# Stage 1: Create Watch Project

Goal

Choose the Template most similar with what you want to develop

In this case, we'll choose Watch Template for Watch Face

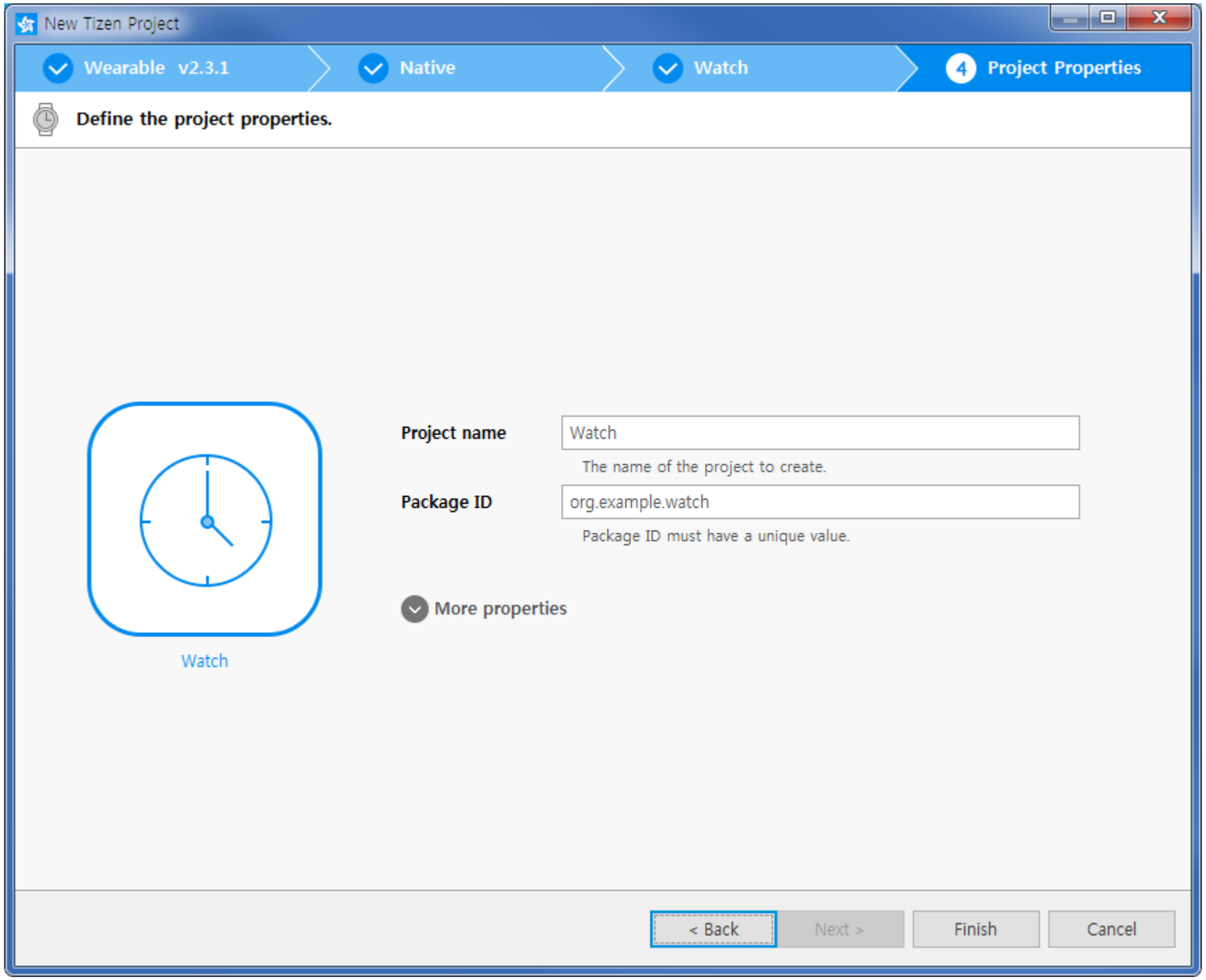


# Stage 1: Create Watch Project

Choose the Template most similar with what you want to develop

In this case, we'll choose Watch Template for Watch Face

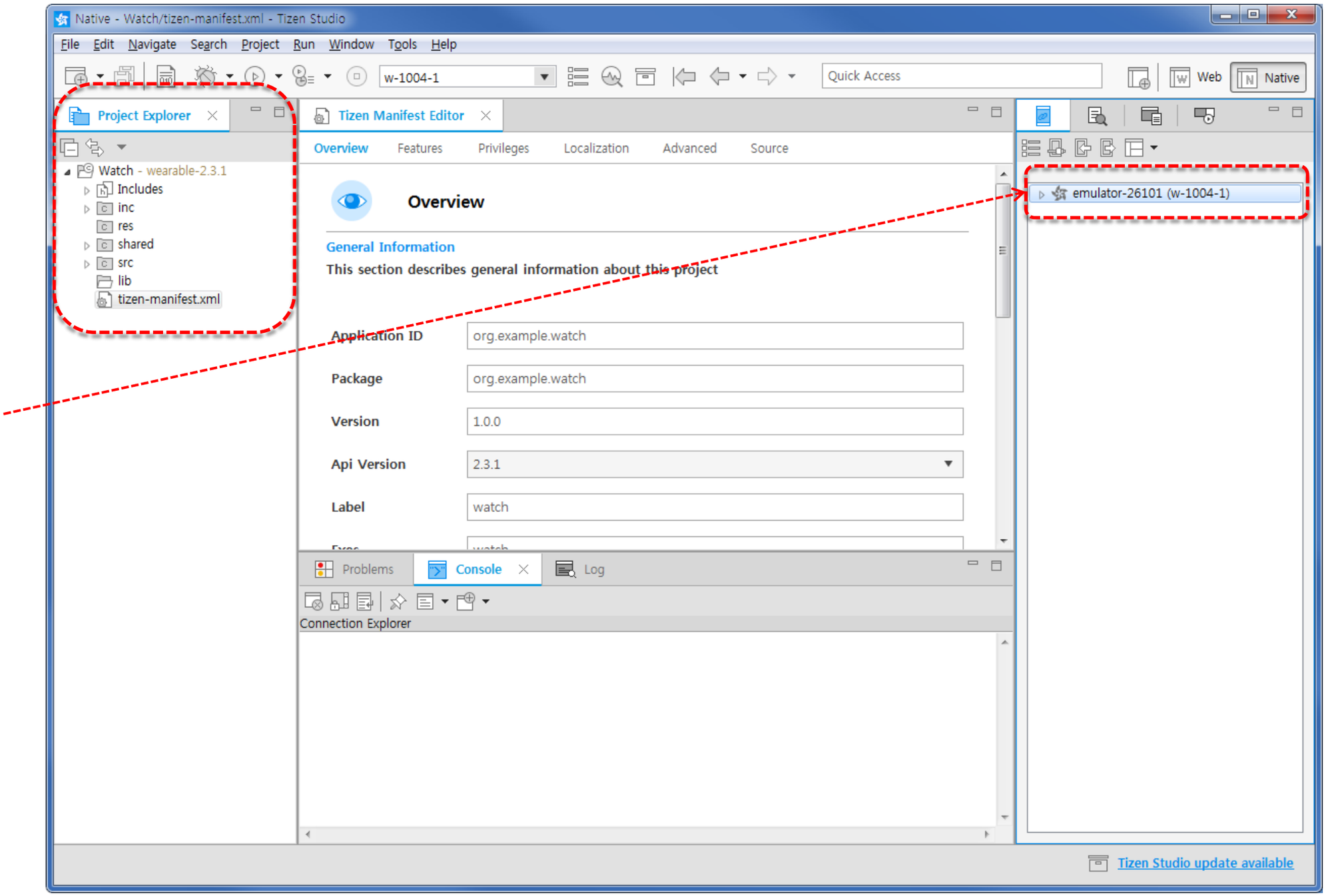


# Stage 1: Create Watch Project

Goal

Now, you can find your Project on the Project Explorer
SDK also provide [Emulator] for the test of your development
Let's launch [Emulator] from now

If you succeed to launch Emulator, you can find Emulator here

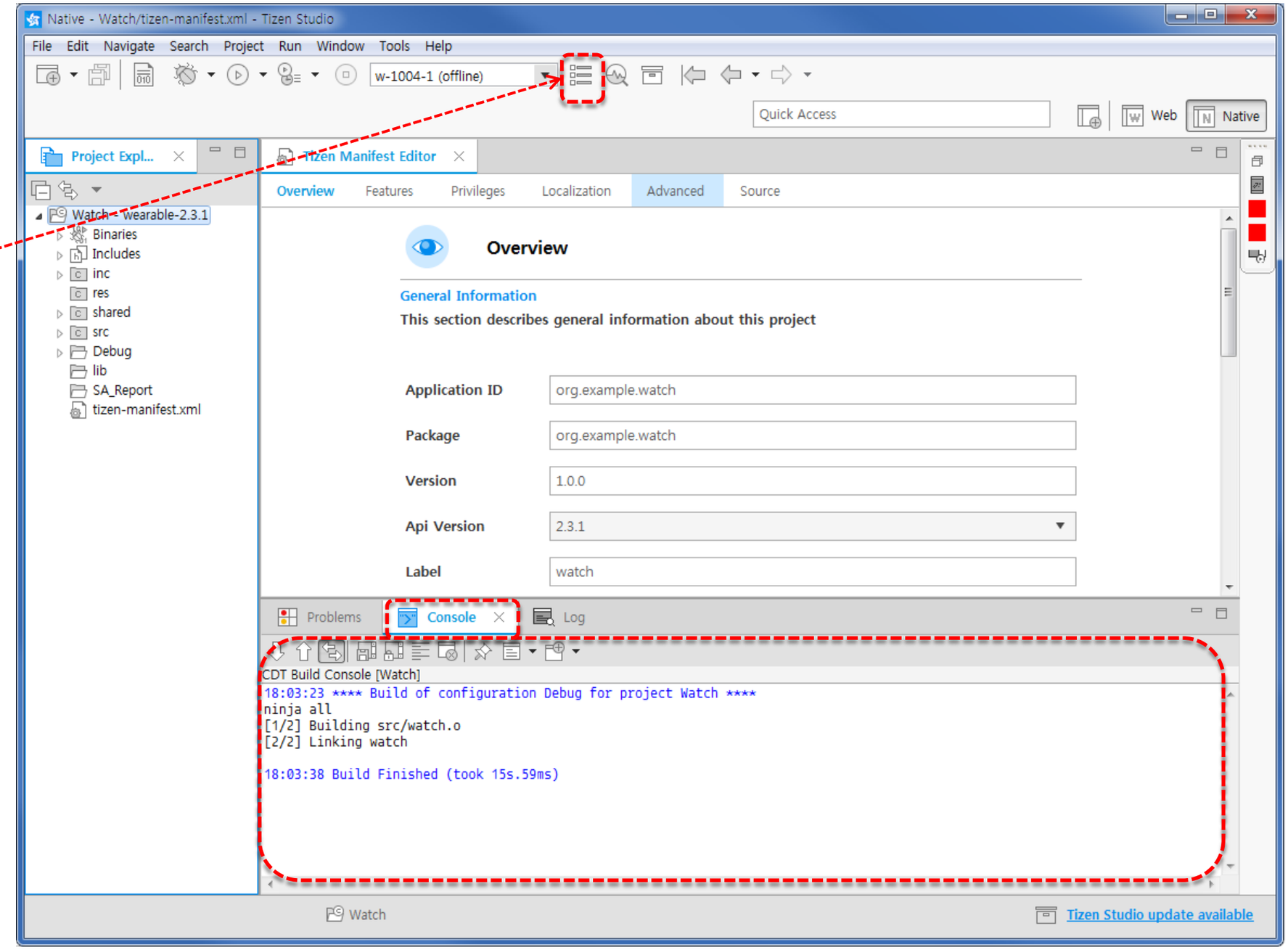


# Stage 2: Create Emulator for test

Goal

You can observe the progress of build through the Console page
In this page, also you can find error & warning messages
Create [Emulator] to test your project

**Click the Emulator Icon Here**



# Stage 2: Create Emulator for test

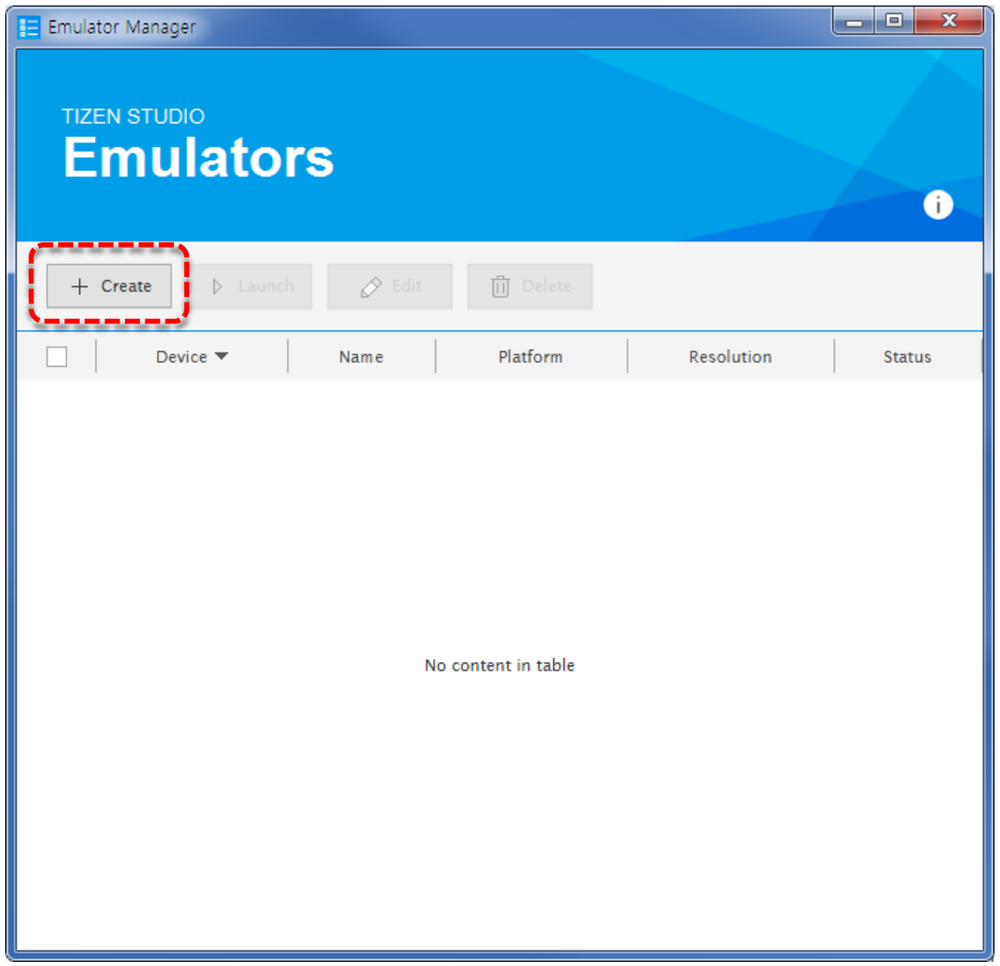
Goal

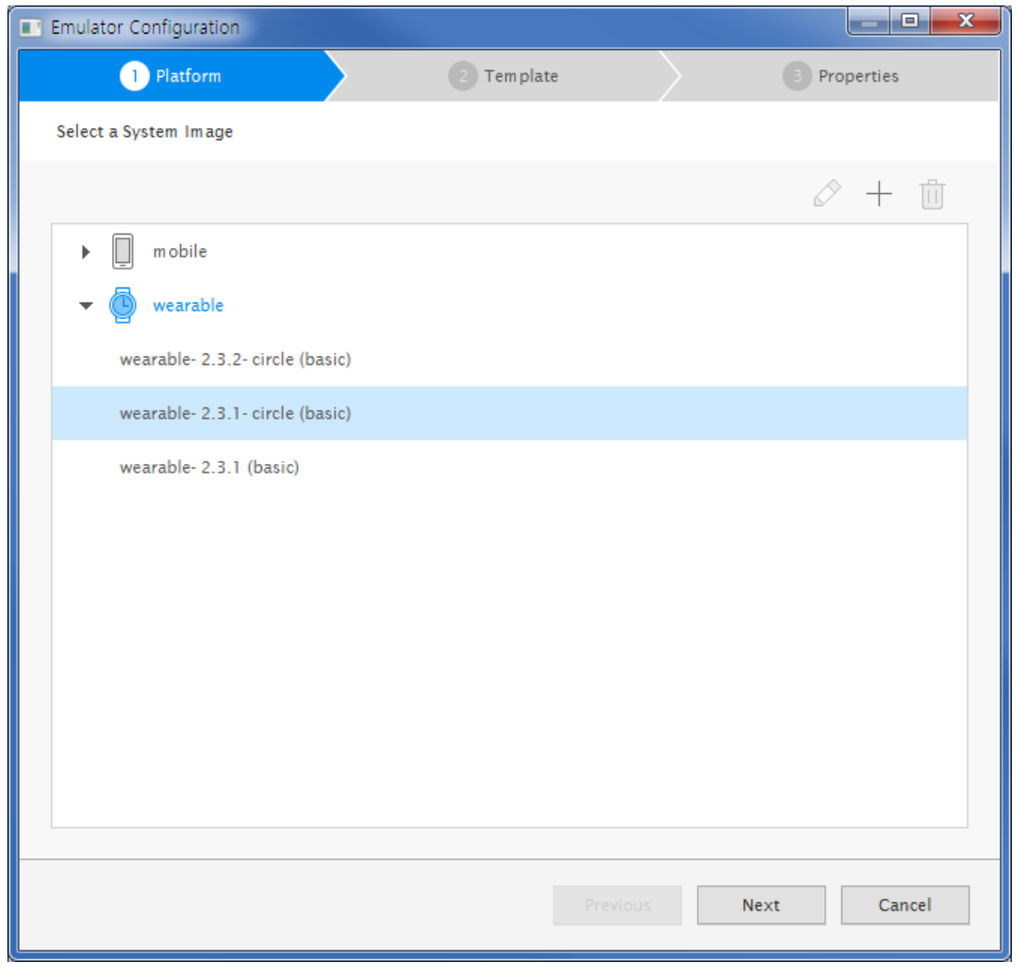
Tizen Studio provide Emulators for various profile(now, mobile and wearable)

For our Wearable Project, choose wearable category

**Click [+ Create]**

**Choose [wearable]**



# Stage 2: Create Emulator for test

Goal

Change the name of [Emulator] if you want

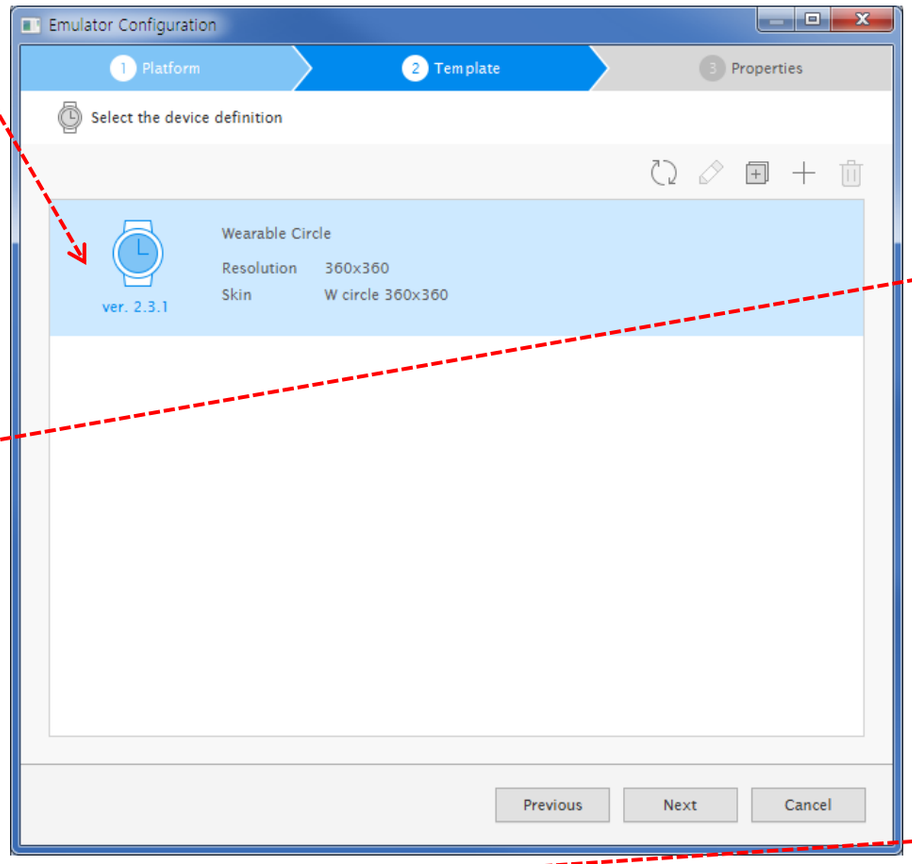
You also can choose Platform version

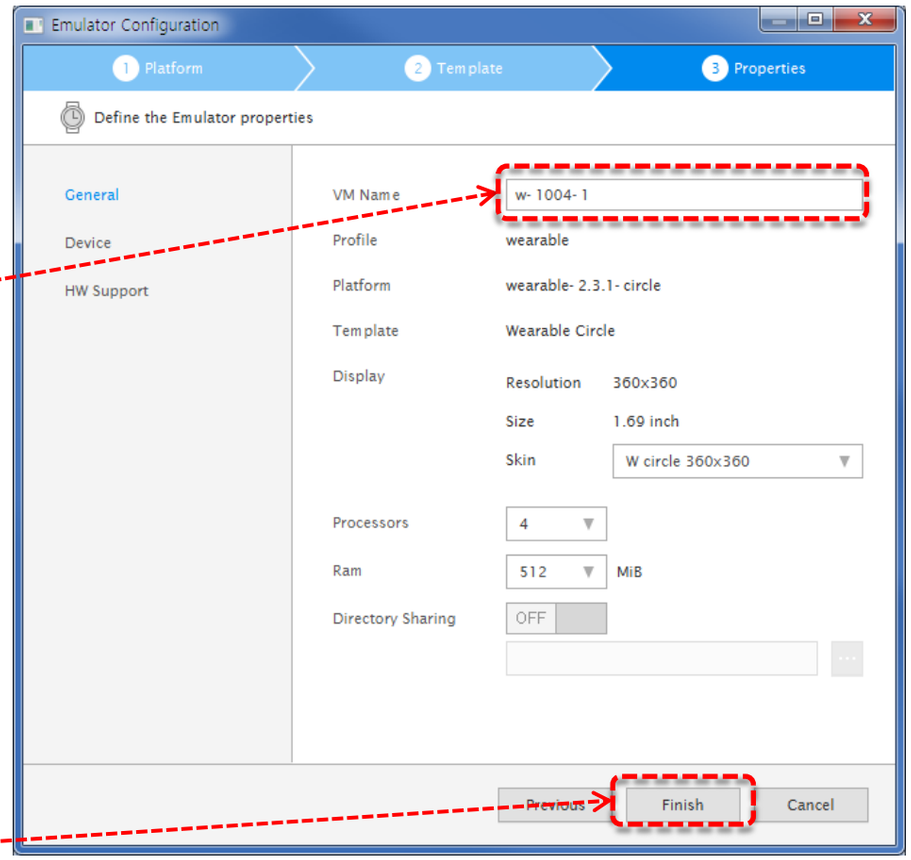
Each version provide different resolutions

**Choose Platform version (Circle is available on Tizen 2.3.1)**

**Check the name of Emulator**

**Click [Finish]**



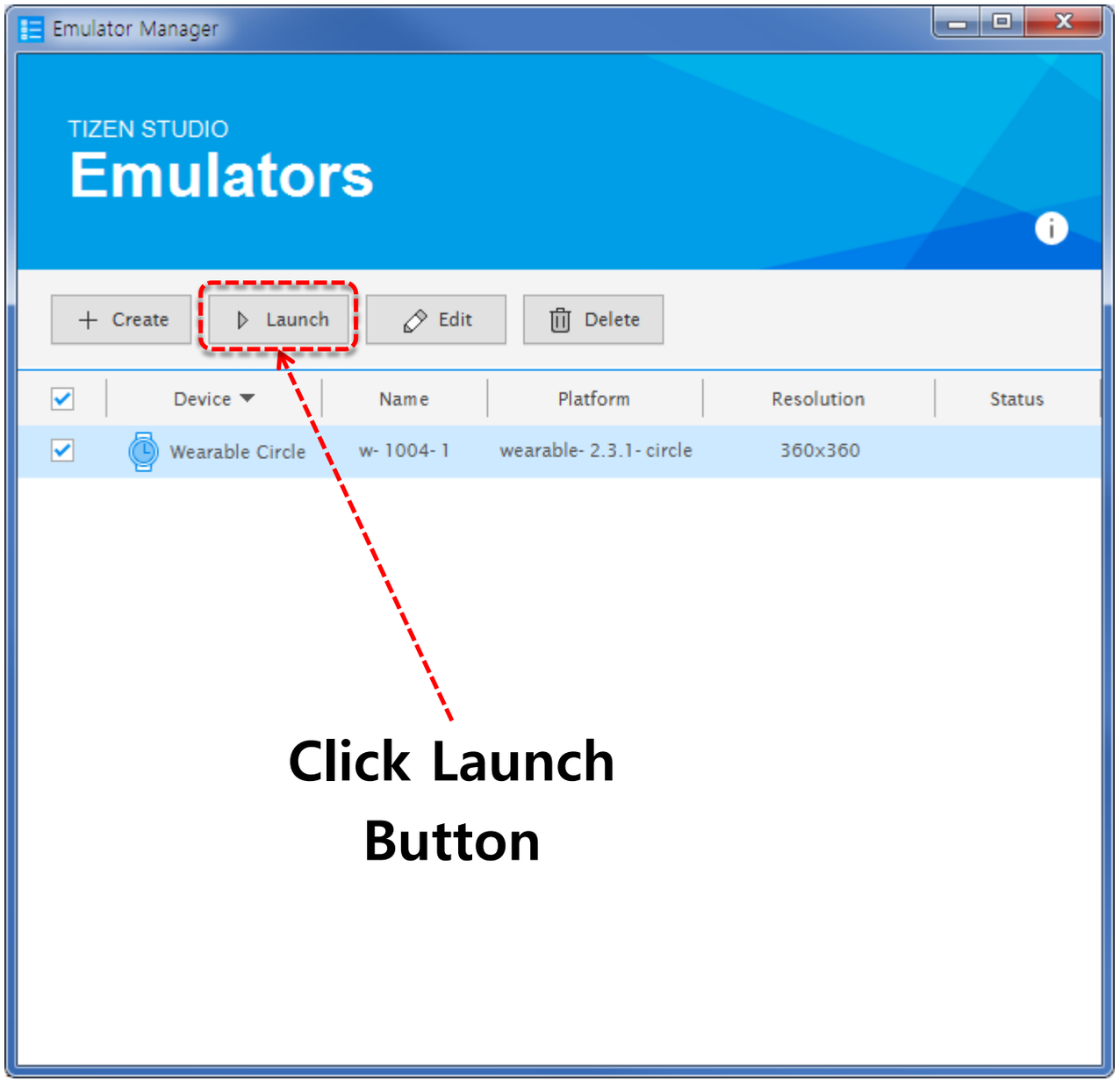



# Stage 2: Create Emulator for test

Goal

Click Play Button to launch Emulator

You can find Default Wearable(Circle) Emulator on the screen





**Now,**
**Let's launch your Wearable project on the Emulator**

# Stage 3: Install & Launch the Project

Goal

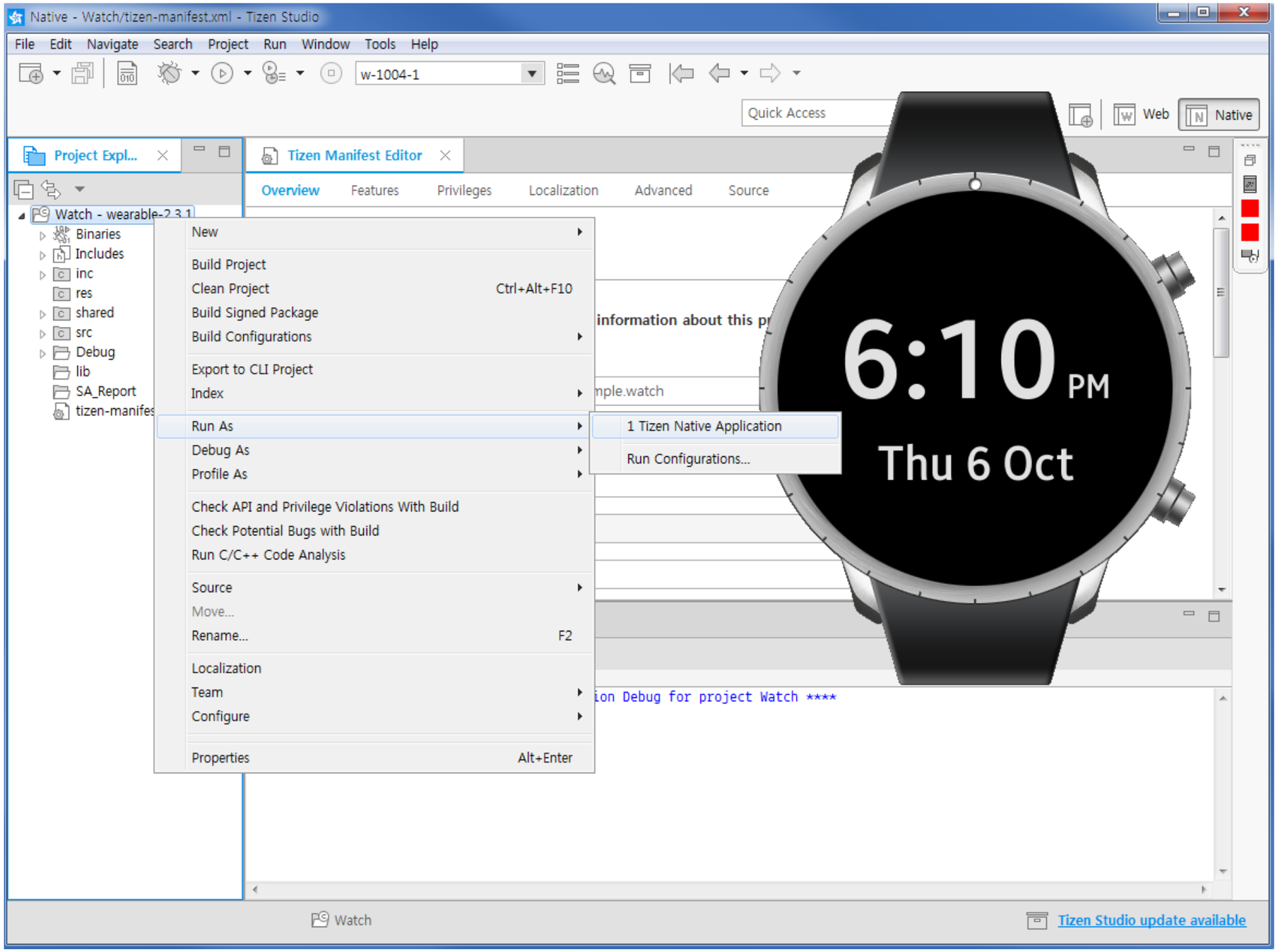
To Run(Launch) your Project, just follow the sequence like below

‘Run’ will install the project and launch the project automatically

**Right Click on Watch Project**

**Choose Run As**

**Choose Tizen Native Application**



# Stage 2: Create Emulator for test

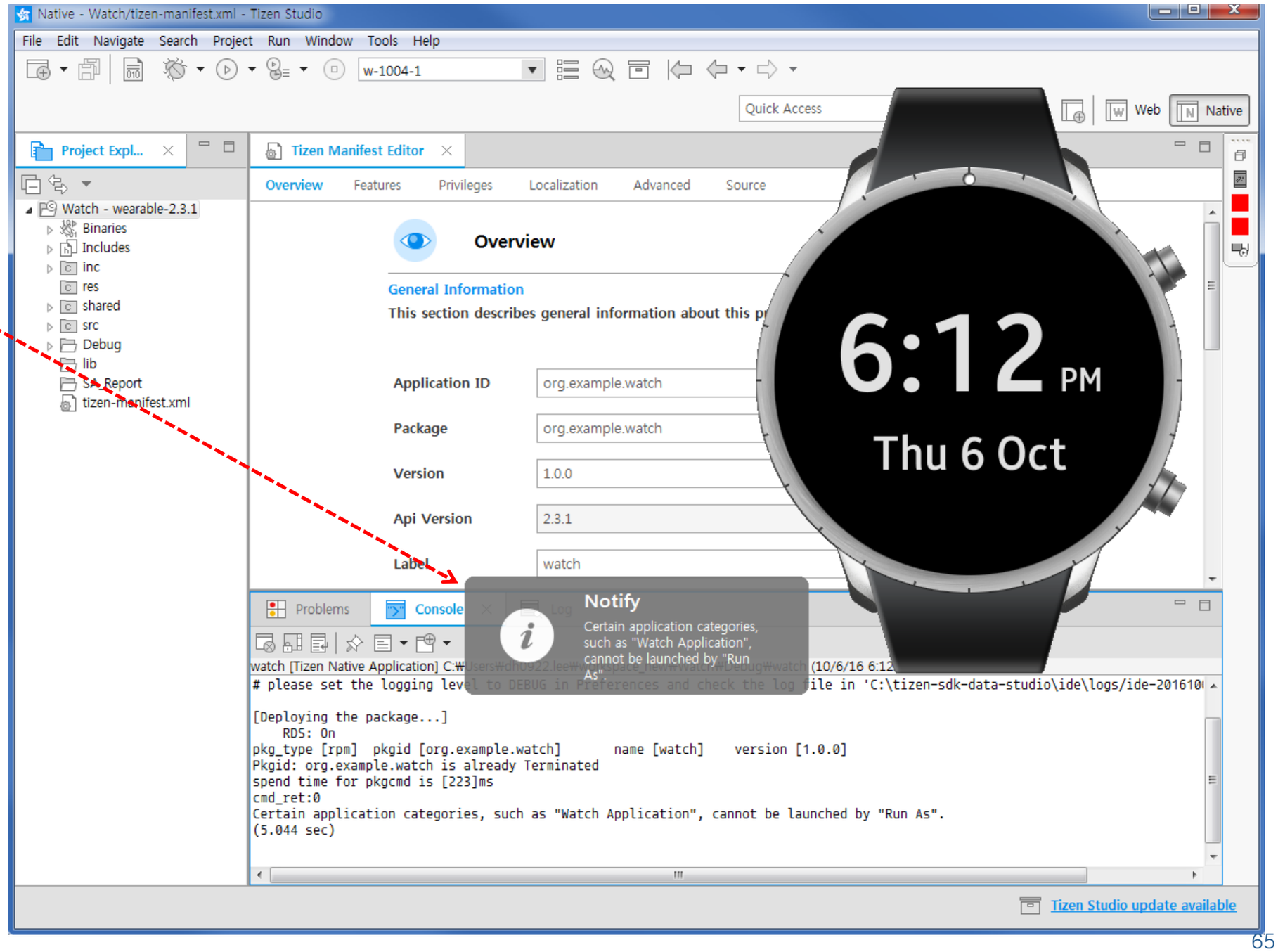
Goal

You can find that there's no change

Some Project like Watch, can't be applied to the Target(emulator) automatically

User must launch manually

Although you can't see your Watch, it is already setup, and you can find it in the 'setting menu'



# Stage 2: Create Emulator for test

Goal

Go to Setting and click Clock menu

You can find ‘Default Tizen Icon’ for your Watch project

After select your Watch, press Home button of the right bottom

**Slide Up** → **Click Settings** → **Click Clock**

**Click Last One** → **Clock is changed** → **Check Home**

# Stage 3: Create user interface layout

Goal

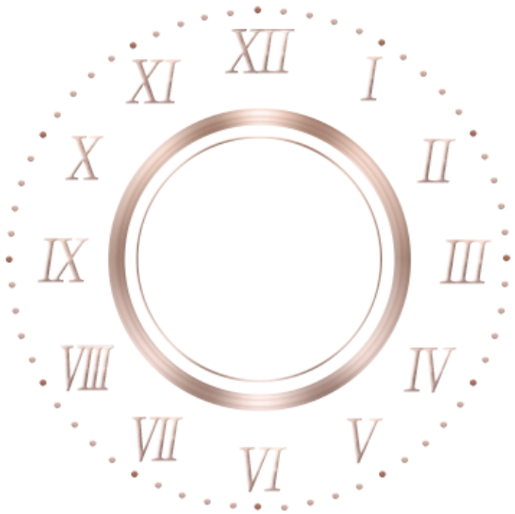
Change the given Watch to Watch Face like below

Let's analyze the Watch Face

Watch Face is made up of 9 Images


We'll change above watch to the below


**Divide into 9 parts**




Sun

# Stage 3: Create user interface layout

Goal

At first, to use Images, make folder for Image files

**Right Click on 'res'**

↓

**New**

↓

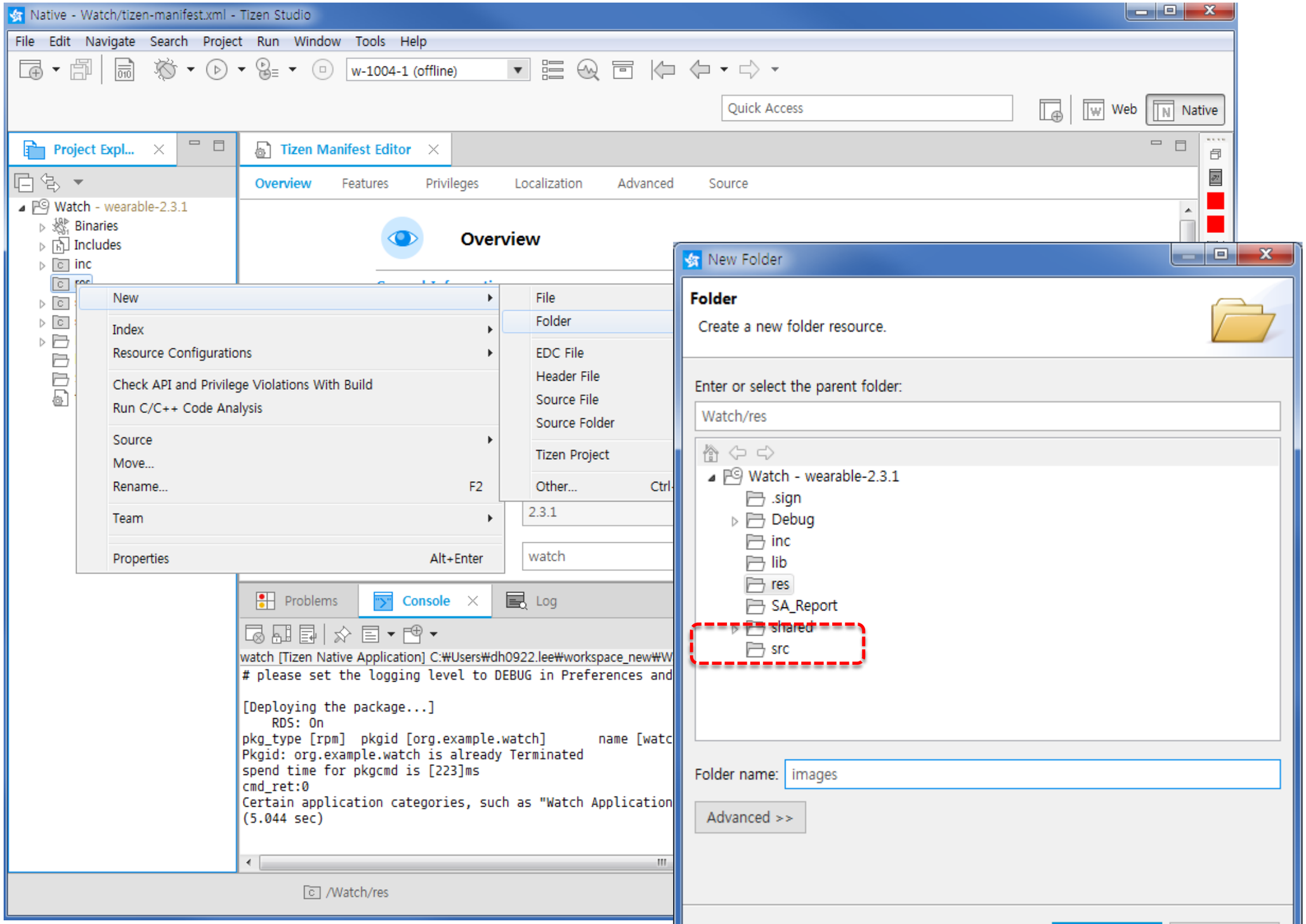
**Folder**

↓

**Write down Folder name [images]**

↓

**Finish**



# Stage 3: Create user interface layout

Goal

Copy Images and paste them to ‘image’ folder

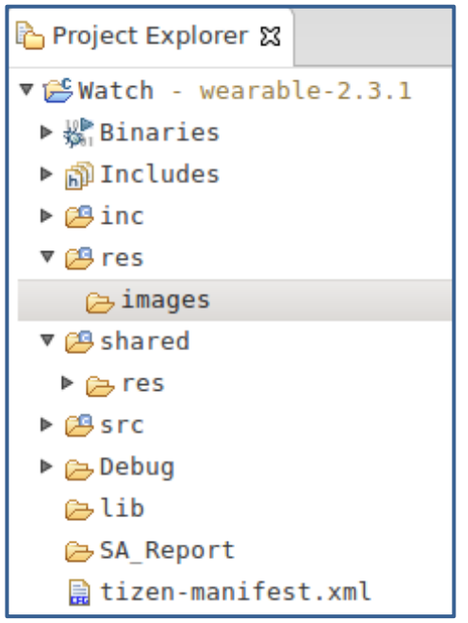
Folder:

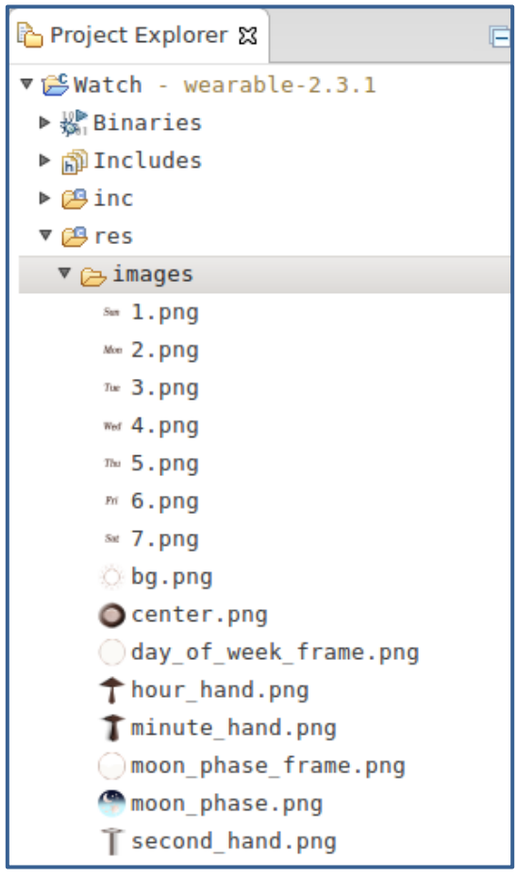
src: C file is located

inc: Header file is located

shared: Resources needed to be shared is located

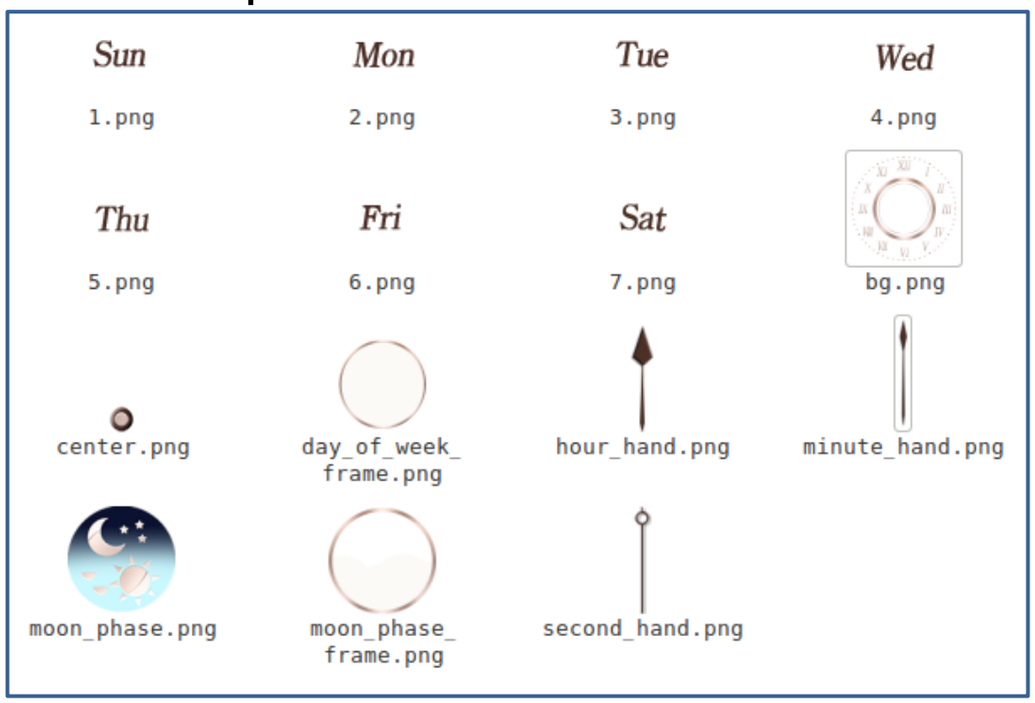
Paste




Copy

These images are given by instructor




**Now, let’s look into source file ‘Watch.c’**

# Stage 3: Create user interface layout

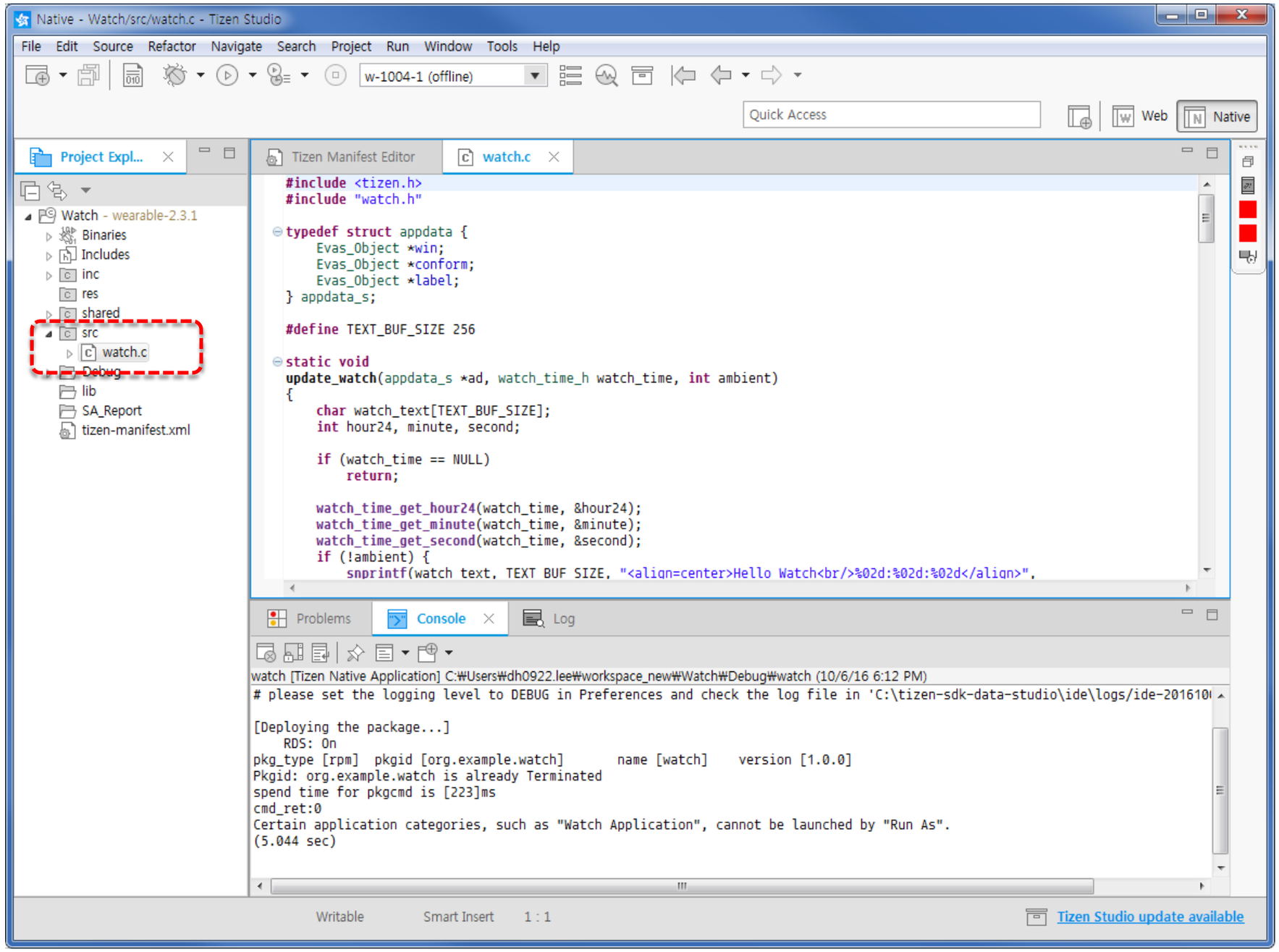
Goal

To make Watch Face , you should modify the ‘watch.c’ file
In this file, create each image part of Watch Face
Follow the given codes.

**Double click on watch.c file to open the file**

↓

**Find the code on the right side**



# Stage 3: Create user interface layout

Goal

Start with ‘create_base_gui’ function

This function create essential object window, conformant for your Watch

We also make each image of the Watch in this function

**Conformant is used normally like this way !!**

**Recommend do not change !!**


Don’t need this Delete !!

```
static void
create_base_gui(appdata_s *ad, int width, int height)
{
    int ret;
    watch_time_h watch_time = NULL;

    /* Window */
    ret = watch_app_get_elm_win(&ad->win);
    if (ret != APP_ERROR_NONE) {
        dlog_print(DLOG_ERROR, LOG_TAG, "failed to get window. err = %d", ret);
        return;
    }

    evas_object_resize(ad->win, width, height);

    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    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);
    evas_object_resize(ad->label, width, height / 3);
    evas_object_move(ad->label, 0, height / 3);
    evas_object_show(ad->label);

    ret = watch_time_get_current_time(&watch_time);
    if (ret != APP_ERROR_NONE)
        dlog_print(DLOG_ERROR, LOG_TAG, "failed to get current time. err = %d", ret);

    update_watch(ad, watch_time, 0);
    watch_time_delete(watch_time);

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

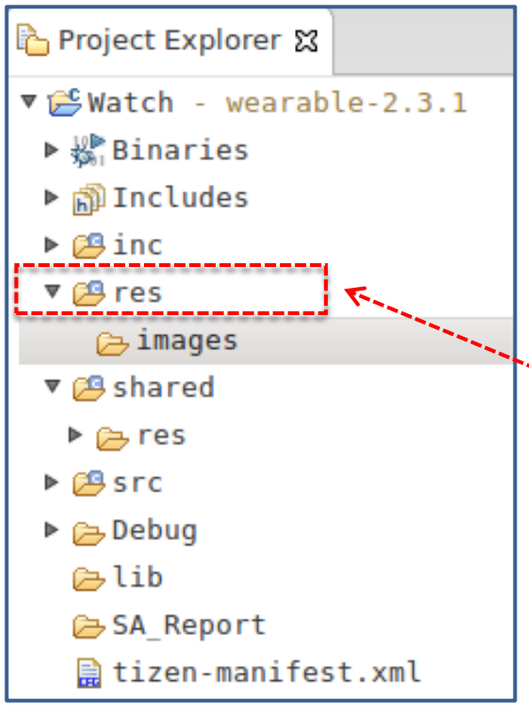
# Stage 3: Create user interface layout

Goal

As told, to we make Watch Face using Images saved in the ‘images’ folder
How can we use these image in the watch.c file?
The given function by EFL, ‘app_get_resource_path()’ get the path of ‘res’ folder

Get & Save the path of ‘res’ !!

‘resource_path’ indicates ‘res’ !!




```c
static void
create_base_gui(appdata_s *ad, int width, int height)
{
    int ret;
    watch_time_h watch_time = NULL;

    /* Window */
    ret = watch_app_get_elm_win(&ad->win);
    if (ret != APP_ERROR_NONE) {
        dlog_print(DLOG_ERROR, LOG_TAG, "failed to get window. err = %d", ret);
        return;
    }

    evas_object_resize(ad->win, width, height);

    /* Conformant */
    ad->conform = elm_conformant_add(ad->win);
    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);

    char *resource_path = NULL;
    resource_path = app_get_resource_path();

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

# Stage 3: Create user interface layout

Goal

First, create background image for Digital Watch

Follow up, below sequence

And let's study the each code

**Create empty object for background**

↓

**Get the path of background image file**

↓

**Set image file to the object**

↓

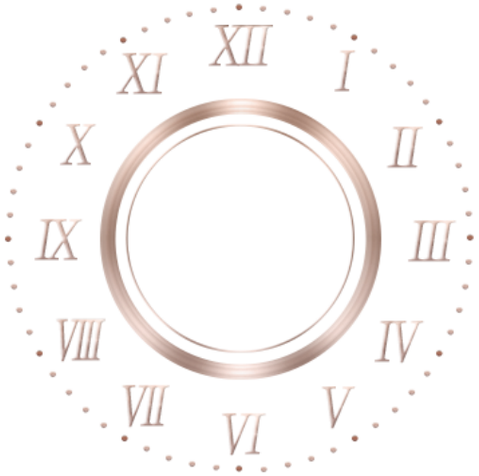
**Locate the object properly**

↓

**Set the size for the object**

↓

**Show the object**





```
char *resource_path = NULL;
resource_path = app_get_resource_path();

/* Background */
Evas_Object *bg = NULL;
char bg_path[1024];

snprintf(bg_path, sizeof(bg_path), "%s%s%s", resource_path, "images/", "bg.png");

bg = elm_bg_add(ad->win);
elm_bg_file_set(bg, bg_path, NULL);

evas_object_move(bg, 0, 0);
evas_object_resize(bg, 360, 360);
evas_object_show(bg);

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

# Stage 3: Create user interface layout

Goal

EFL offer each APIs for effective development

Also EFL offer intuitive APIs for understanding what this API is for

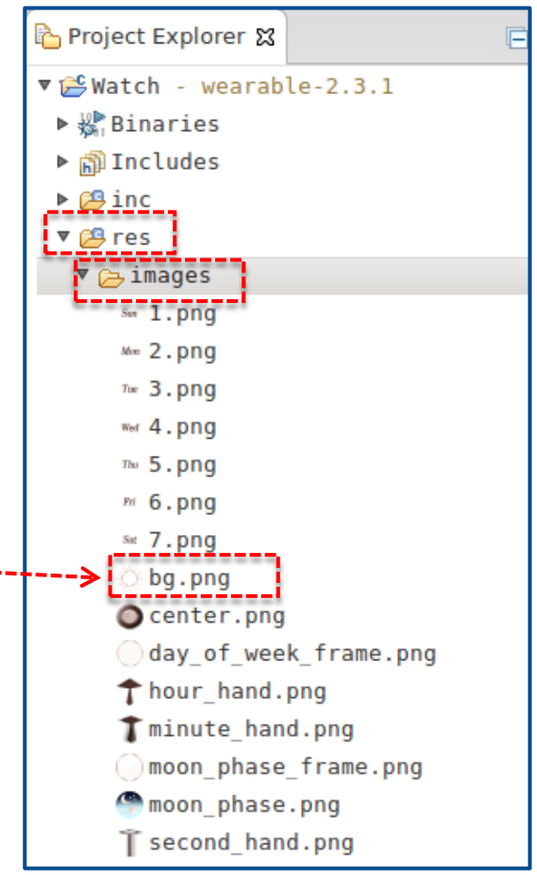
Let's match the APIs with purpose

## Create empty object for background

```
Evas_Object *bg = NULL;
bg = elm_bg_add(ad->win);
```

## Get the path of background image file

```
snprintf(bg_path, sizeof(bg_path), "%s%s%s", resource_path, "images/", "bg.png");
```




## Set image file to the object

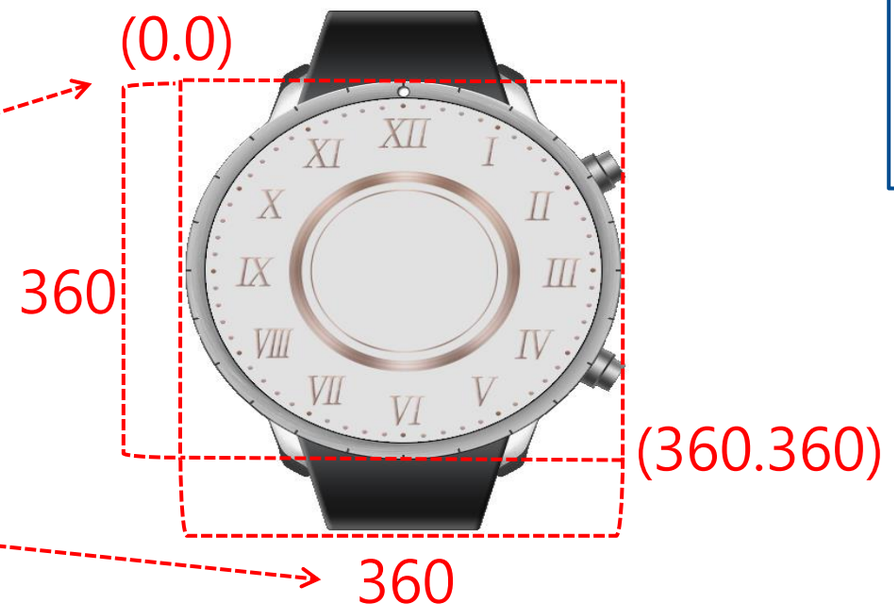
```
elm_bg_file_set(bg, bg_path, NULL);
```

## Locate the object properly

```
evas_object_move(bg, 0, 0);
```

## Set the size for the object

```
evas_object_resize(bg, 360, 360);
```




## Show the object

```
evas_object_show(bg);
```

# Stage 3: Create user interface layout

Goal

Second, create day image for Digital Watch

Day image is not used for background

For image object, EFL, offer ‘elm_image_xxx’ APIs

**Two image objects are needed to display the day**

**week_day =** Sun

**week_day_ frame =**

180 = 360/2 = middle of width & height

65 = distance between week_day and middle

55 = size of week_day & week_day_frame


```
/* Week Day */
Evas_Object *week_day_frame = NULL;
char week_day_frame_path[1024];

snprintf(week_day_frame_path, sizeof(week_day_frame_path), "%s%s%s", resource_path, "images/", "day_of_week_frame.png");

week_day_frame = elm_image_add(bg);
elm_image_file_set(week_day_frame, week_day_frame_path, NULL);

evas_object_move(week_day_frame, 180+65, 180-55/2);
evas_object_resize(week_day_frame, 55, 55);
evas_object_show(week_day_frame);
```

This coordinate is left top of the ‘week_day_frame’

```
Evas_Object *week_day = NULL;
char week_day_path[1024];

snprintf(week_day_path, sizeof(week_day_path), "%s%s%s", resource_path, "images/", "1.png");

week_day = elm_image_add(bg);
elm_image_file_set(week_day, week_day_path, NULL);

evas_object_move(week_day, 180+65, 180-55/2);
evas_object_resize(week_day, 55, 55);
evas_object_show(week_day);
```

# Stage 3: Create user interface layout

Goal

Third, create moon image for Watch Face
This is also image object like day
Just check size and position

**Two image objects are needed to display the moon**


**moon =** 

**moon_frame =** 

180 = 360/2 = middle of width & height

21 = distance between moon and middle

102 = size of moon & moon_frame

```
/* Moon */
Evas_Object *moon = NULL;
char moon_path[1024];

snprintf(moon_path, sizeof(moon_path), "%s%s%s", resource_path, "images/", "moon_phase.png");

moon = elm_image_add(bg);
elm_image_file_set(moon, moon_path, NULL);

evas_object_move(moon, 180-102/2, 180+21);
evas_object_resize(moon, 102, 102);
evas_object_show(moon);
```

This coordinate is left top of the 'moon'

```
Evas_Object *moon_frame = NULL;
char moon_frame_path[1024];

snprintf(moon_frame_path, sizeof(moon_frame_path), "%s%s%s", resource_path, "images/", "moon_phase_frame.png");

moon_frame = elm_image_add(bg);
elm_image_file_set(moon_frame, moon_frame_path, NULL);

evas_object_move(moon_frame, 180-102/2, 180+21);
evas_object_resize(moon_frame, 102, 102);
evas_object_show(moon_frame);
```

# Stage 5: Create user interface layout

Goal

Fourth, create center & hour hand of the clock

This is also image object like others

Just check size and position

**center =** 14x14

**hour_hand =**

180 = 360/2 = middle of width & height

14 = width & height of center

18 = width of hour_hand

88 = height of hour_hand

17 = distance between center and end of hour_hand

**Check cross part that each hand is overlapped**


```
/* Clock hands */
Evas_Object *center = NULL;
char center_path[1024];

snprintf(center_path, sizeof(center_path), "%s%s%s", resource_path, "images/", "center.png");

center = elm_image_add(bg);
elm_image_file_set(center, center_path, NULL);

evas_object_move(center, 180-14/2, 180-14/2);
evas_object_resize(center, 14, 14);
evas_object_show(center);
```

This coordinate is left top of the 'center'

```
Evas_Object *hour_hand = NULL;
char hour_hand_path[1024];

snprintf(hour_hand_path, sizeof(hour_hand_path), "%s%s%s", resource_path, "images/", "hour_hand.png");

hour_hand = elm_image_add(bg);
elm_image_file_set(hour_hand, hour_hand_path, NULL);

evas_object_move(hour_hand, 180-18/2, 180-88+17);
evas_object_resize(hour_hand, 18, 88);
evas_object_show(hour_hand);
```

This coordinate is left top of the 'hour_hand'

# Stage 3: Create user interface layout

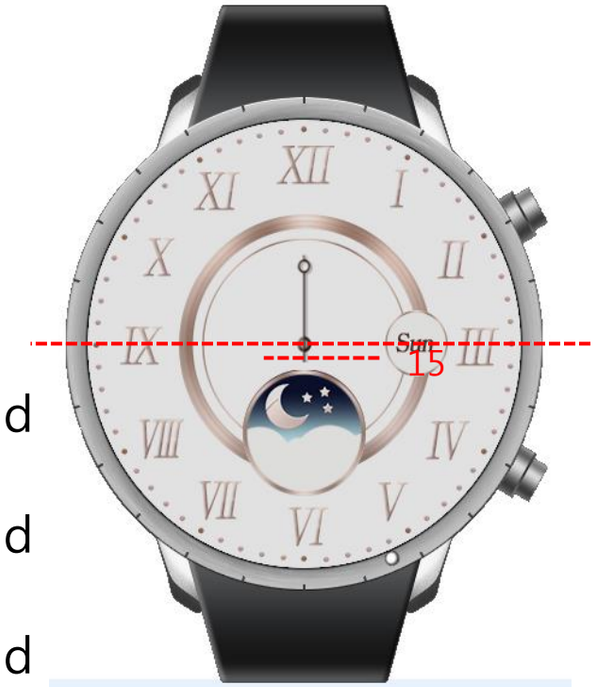
Goal

Fifth, create min & sec hands of the clock

This is also image object like others

Just check size and position

**sec_hand =**




15 = width of sec_hand

15 = distance between center and end of sec_hand

87 = height of sec_hand

**= min_hand**

12 = width of min_hand

20 = distance between center and end of min_hand

132 = height of min_hand


```
Evas_Object *min_hand = NULL;
char min_hand_path[1024];

snprintf(min_hand_path, sizeof(min_hand_path), "%s%s%s", resource_path, "images/", "minute_hand.png");

min_hand = elm_image_add(bg);
elm_image_file_set(min_hand, min_hand_path, NULL);

evas_object_move(min_hand, 180-12/2, 180-132+20);
evas_object_resize(min_hand, 12, 132);
evas_object_show(min_hand);
```

This coordinate is left top of the 'min_hand'

```
Evas_Object *sec_hand = NULL;
char sec_hand_path[1024];

snprintf(sec_hand_path, sizeof(sec_hand_path), "%s%s%s", resource_path, "images/", "second_hand.png");

sec_hand = elm_image_add(bg);
elm_image_file_set(sec_hand, sec_hand_path, NULL);

evas_object_move(sec_hand, 180-15/2, 180-87+15);
evas_object_resize(sec_hand, 15, 87);
evas_object_show(sec_hand);
```

This coordinate is left top of the 'sec_hand'

# Stage 3: Create user interface layout

Goal

Now, we finish the development of Watch Face UI

But, this watch looks like strange

Because hands of the clock are overlapped, and is not working

So, next we make this watch work properly

**Clock hands are not working**

**Moon is not working**

**Day is not working**


**Next, Let's make it work**

# Stage 4: Add operation to the watch

Goal

From now, we move the images we've already made
We put all our code into 'create_base_gui()' function before
At this time we make another function for moving the clock

Find 'app_create' function

↓

We'll make another function
'set_the_time' for moving the clock

↓

To access to images we've made at the another function, we should make these image objects global variables

↓

Put into global struct variable 'appdata' to control easily

```c
static bool
app_create(int width, int height, 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, width, height);

    set_the_time(ad);

    return true;
}
```

```c
typedef struct appdata {
    Evas_Object *win;
    Evas_Object *conform;
    Evas_Object *label;
    Evas_Object *week_day;
    Evas_Object *moon;
    Evas_Object *hour_hand;
    Evas_Object *min_hand;
    Evas_Object *sec_hand;
} appdata_s;
```

# Stage 4: Add operation to the watch

Goal

## Compare with regional variable code

week_day is already declared in structure ‘ad’

Evas_Obeject *week_day = NULL

↓

Removed

week_day

↓

ad->week_day

```c
/* Week Day */
Evas_Object *week_day_frame = NULL;
char week_day_frame_path[1024];

snprintf(week_day_frame_path, sizeof(week_day_frame_path), "%s%s%s", resource_path, "images/", "day_of_week_frame.png");

week_day_frame = elm_image_add(bg);
elm_image_file_set(week_day_frame, week_day_frame_path, NULL);

evas_object_move(week_day_frame, 180+65, 180-55/2);
evas_object_resize(week_day_frame, 55, 55);
evas_object_show(week_day_frame);

Evas_Object *week_day = NULL;
char week_day_path[1024];

snprintf(week_day_path, sizeof(week_day_path), "%s%s%s", resource_path, "images/", "1.png");

week_day = elm_image_add(bg);
elm_image_file_set(week_day, week_day_path, NULL);

evas_object_move(week_day, 180+65, 180-55/2);
evas_object_resize(week_day, 55, 55);
evas_object_show(week_day);
```

```c
/* Week Day */
Evas_Object *week_day_frame = NULL;
char week_day_frame_path[1024];

snprintf(week_day_frame_path, sizeof(week_day_frame_path), "%s%s%s", resource_path, "images/", "day_of_week_frame.png");

week_day_frame = elm_image_add(bg);
elm_image_file_set(week_day_frame, week_day_frame_path, NULL);

evas_object_move(week_day_frame, 180+65, 180-55/2);
evas_object_resize(week_day_frame, 55, 55);
evas_object_show(week_day_frame);

char week_day_path[1024];

snprintf(week_day_path, sizeof(week_day_path), "%s%s%s", resource_path, "images/", "1.png");

ad->week_day = elm_image_add(bg);
elm_image_file_set(ad->week_day, week_day_path, NULL);

evas_object_move(ad->week_day, 180+65, 180-55/2);
evas_object_resize(ad->week_day, 55, 55);
evas_object_show(ad->week_day);
```

# Stage 4: Add operation to the watch

Goal

## Compare with regional variable code

moon is already declared in structure ‘ad’

Evas_Obeject *moon = NULL

↓

Removed

```c
/* Moon */
Evas_Object *moon = NULL;
char moon_path[1024];

snprintf(moon_path, sizeof(moon_path), "%s%s%s", resource_path, "images/", "moon_phase.png");

moon = elm_image_add(bg);
elm_image_file_set(moon, moon_path, NULL);

evas_object_move(moon, 180-102/2, 180+21);
evas_object_resize(moon, 102, 102);
evas_object_show(moon);

Evas_Object *moon_frame = NULL;
char moon_frame_path[1024];

snprintf(moon_frame_path, sizeof(moon_frame_path), "%s%s%s", resource_path, "images/", "moon_phase_frame.png");

moon_frame = elm_image_add(bg);
elm_image_file_set(moon_frame, moon_frame_path, NULL);

evas_object_move(moon_frame, 180-102/2, 180+21);
evas_object_resize(moon_frame, 102, 102);
evas_object_show(moon_frame);
```

moon

↓

ad->moon

```c
/* Moon */
char moon_path[1024];

snprintf(moon_path, sizeof(moon_path), "%s%s%s", resource_path, "images/", "moon_phase.png");

ad->moon = elm_image_add(bg);
elm_image_file_set(ad->moon, moon_path, NULL);

evas_object_move(ad->moon, 180-102/2, 180+21);
evas_object_resize(ad->moon, 102, 102);
evas_object_show(ad->moon);

Evas_Object *moon_frame = NULL;
char moon_frame_path[1024];

snprintf(moon_frame_path, sizeof(moon_frame_path), "%s%s%s", resource_path, "images/", "moon_phase_frame.png");

moon_frame = elm_image_add(bg);
elm_image_file_set(moon_frame, moon_frame_path, NULL);

evas_object_move(moon_frame, 180-102/2, 180+21);
evas_object_resize(moon_frame, 102, 102);
evas_object_show(moon_frame);
```

# Stage 4: Add operation to the watch

Goal

## Compare with regional variable code

hour_hand is already declared in structure ‘ad’

Evas_Obeject *hour_hand = NULL

↓

Removed

hour_hand

↓

ad->hour_hand

```c
/* Clock hands */
Evas_Object *center = NULL;
char center_path[1024];

snprintf(center_path, sizeof(center_path), "%s%s%s", resource_path, "images/", "center.png");

center = elm_image_add(bg);
elm_image_file_set(center, center_path, NULL);

evas_object_move(center, 180-14/2, 180-14/2);
evas_object_resize(center, 14, 14);
evas_object_show(center);

Evas_Object *hour_hand = NULL;
char hour_hand_path[1024];

snprintf(hour_hand_path, sizeof(hour_hand_path), "%s%s%s", resource_path, "images/", "hour_hand.png");

hour_hand = elm_image_add(bg);
elm_image_file_set(hour_hand, hour_hand_path, NULL);

evas_object_move(hour_hand, 180-18/2, 180-88+17);
evas_object_resize(hour_hand, 18, 88);
evas_object_show(hour_hand);
```

```c
/* Clock hands */
Evas_Object *center = NULL;
char center_path[1024];

snprintf(center_path, sizeof(center_path), "%s%s%s", resource_path, "images/", "center.png");

center = elm_image_add(bg);
elm_image_file_set(center, center_path, NULL);

evas_object_move(center, 180-14/2, 180-14/2);
evas_object_resize(center, 14, 14);
evas_object_show(center);

char hour_hand_path[1024];

snprintf(hour_hand_path, sizeof(hour_hand_path), "%s%s%s", resource_path, "images/", "hour_hand.png");

ad->hour_hand = elm_image_add(bg);
elm_image_file_set(ad->hour_hand, hour_hand_path, NULL);

evas_object_move(ad->hour_hand, 180-18/2, 180-88+17);
evas_object_resize(ad->hour_hand, 18, 88);
evas_object_show(ad->hour_hand);
```

# Stage 4: Add operation to the watch

Goal

## Compare with regional variable code

Min_hand &
sec_hand are already
declared in structure
'ad'

Evas_Obeject
*min_hand = NULL
Evas_Obeject
*sec_hand = NULL

↓

Removed

min_hand,
sec_hand

↓

ad->min_hand,
ad->sec_hand

```c
Evas_Object *min_hand = NULL;
char min_hand_path[1024];

snprintf(min_hand_path, sizeof(min_hand_path), "%s%s%s", resource_path, "images/", "minute_hand.png");

min_hand = elm_image_add(bg);
elm_image_file_set(min_hand, min_hand_path, NULL);

evas_object_move(min_hand, 180-12/2, 180-132+20);
evas_object_resize(min_hand, 12, 132);
evas_object_show(min_hand);

Evas_Object *sec_hand = NULL;
char sec_hand_path[1024];

snprintf(sec_hand_path, sizeof(sec_hand_path), "%s%s%s", resource_path, "images/", "second_hand.png");

sec_hand = elm_image_add(bg);
elm_image_file_set(sec_hand, sec_hand_path, NULL);

evas_object_move(sec_hand, 180-15/2, 180-87+15);
evas_object_resize(sec_hand, 15, 87);
evas_object_show(sec_hand);
```

```c
char min_hand_path[1024];

snprintf(min_hand_path, sizeof(min_hand_path), "%s%s%s", resource_path, "images/", "minute_hand.png");

ad->min_hand = elm_image_add(bg);
elm_image_file_set(ad->min_hand, min_hand_path, NULL);

evas_object_move(ad->min_hand, 180-12/2, 180-132+20);
evas_object_resize(ad->min_hand, 12, 132);
evas_object_show(ad->min_hand);

char sec_hand_path[1024];

snprintf(sec_hand_path, sizeof(sec_hand_path), "%s%s%s", resource_path, "images/", "second_hand.png");

ad->sec_hand = elm_image_add(bg);
elm_image_file_set(ad->sec_hand, sec_hand_path, NULL);

evas_object_move(ad->sec_hand, 180-15/2, 180-87+15);
evas_object_resize(ad->sec_hand, 15, 87);
evas_object_show(ad->sec_hand);
```

# Stage 4: Add operation to the watch

Goal

Now, make ‘set_the_time’ function with Structure ‘ad’

**Make ‘set_the_time’ function over the ‘app_create’ function**

**Call ‘set_the_time’ function, after ‘create_base_gui’ function**

**Pass the ‘ad’ as a parameter**

```
static void
set_the_time(appdata_s *ad)
{

}

static bool
app_create(int width, int height, 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, width, height);

    set_the_time(ad);

    return true;
}
```

# Stage 4: Add operation to the watch

Goal

To move the clock according to the current time,
you should get the current time
Tizen provides APIs to get current time easily

```
watch_time_h watch_time = NULL;
```

This is pre-made handler for store of
several kinds of time information
Ex) hour, minute, second, day and 24hour

```
watch_time_get_current_time(&watch_time);
```

This function get current time information
and save it to the watch_time handler

```
watch_time_get_hour24(watch_time, &hour24);
watch_time_get_hour(watch_time, &hour);
watch_time_get_minute(watch_time, &minute);
watch_time_get_second(watch_time, &second);
watch_time_get_day_of_week(watch_time, &day);
```

```
static void
set_the_time(appdata_s *ad)
{
    watch_time_h watch_time = NULL;

    int day, hour, minute, second, hour24;

    watch_time_get_current_time(&watch_time);

    watch_time_get_hour24(watch_time, &hour24);
    watch_time_get_hour(watch_time, &hour);
    watch_time_get_minute(watch_time, &minute);
    watch_time_get_second(watch_time, &second);
    watch_time_get_day_of_week(watch_time, &day);
}
```

Get current hour type of 24hour Ex) 23

Get current hour type of 12hour Ex) 11

Get current hour type of minute Ex) 33

Get current hour type of second Ex) 58

Get current hour type of 24hour
Ex) 1->sun, 2->mon, 3->tue.....

# Stage 4: Add operation to the watch

Goal

First, we change the day according to current time

Use variable ‘day’ that has day information of the current

Make ‘update_the_day’ function over the ‘set_the_time’ function

```
static void
update_the_day(appdata_s *ad, int day)
{
    char *resource_path = NULL;
    char day_path[1024];

    resource_path = app_get_resource_path();

    snprintf(day_path, sizeof(day_path), "%s%s%d%s", resource_path, "images/", day, ".png");

    elm_image_file_set(ad->week_day, day_path, NULL);

}
```

Parameter ‘day’ determine what image will be used for the ‘ad->day’

Set that image file to the ‘ad->day’


Day is changed !!

```
static void
set_the_time(appdata_s *ad)
{

    watch_time_h watch_time = NULL;

    int day, hour, minute, second, hour24;

    watch_time_get_current_time(&watch_time);

    watch_time_get_hour24(watch_time, &hour24);
    watch_time_get_hour(watch_time, &hour);
    watch_time_get_minute(watch_time, &minute);
    watch_time_get_second(watch_time, &second);
    watch_time_get_day_of_week(watch_time, &day);

    update_the_day(ad, day);
}
```

# Stage 4: Add operation to the watch

Goal

Second, we move the moon according to current time

Tizen provide APIs for easy transformation of the object

Among them, Let's study 'evas_map_new' API

Make 'move_the_moon' function over the 'set_the_time' function

↓

Create four points

↓

This fill out four point with four cooridinates of 'ad->moon'

↓

Rotate four point of 'm' by degree, and center of the 'm' is a center axis

↓

This means that move 'ad->moon' to rotated position of 'm'

↓

Destroy the 'm' after rotation

```c
static void
move_the_moon(appdata_s *ad, int degree)
{
    Evas_Map *m = NULL;

    m = evas_map_new(4);
    evas_map_util_points_populate_from_object(m, ad->moon);
    evas_map_util_rotate(m, degree, 360/2, 180+21+102/2);
    evas_object_map_set(ad->moon, m);
    evas_object_map_enable_set(ad->moon, EINA_TRUE);
    evas_map_free(m);
}
```

```c
static void
set_the_time(appdata_s *ad)
{
    watch_time_h watch_time = NULL;

    int day, hour, minute, second, hour24;

    watch_time_get_current_time(&watch_time);

    watch_time_get_hour24(watch_time, &hour24);
    watch_time_get_hour(watch_time, &hour);
    watch_time_get_minute(watch_time, &minute);
    watch_time_get_second(watch_time, &second);
    watch_time_get_day_of_week(watch_time, &day);

    update_the_day(ad, day);

    move_the_moon(ad, hour24 * 15);
}
```

24*15 = 360


Moon image is rotated!!

# Stage 4: Add operation to the watch

Goal

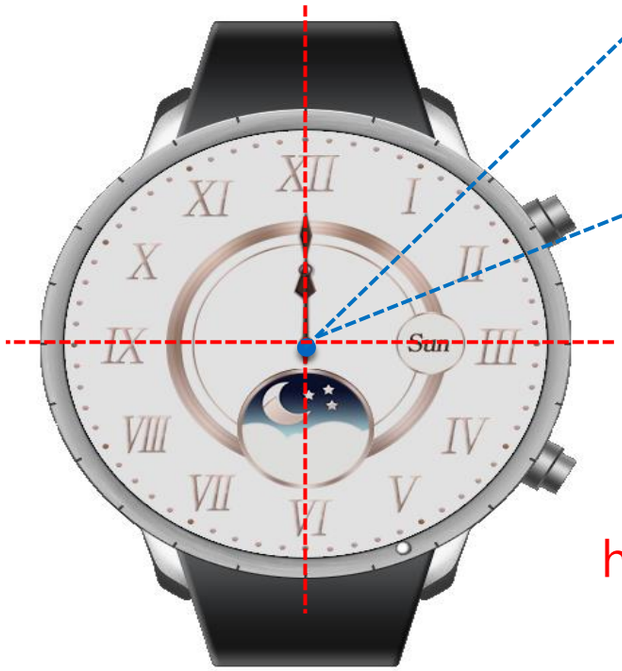
Third, we move the hour_hand according to current time
Use ‘evas_map_new’ API to rotate the hour_hand
Important thing is the ‘degree’

Make ‘move_the_hand’ function
over the ‘set_the_time’ function

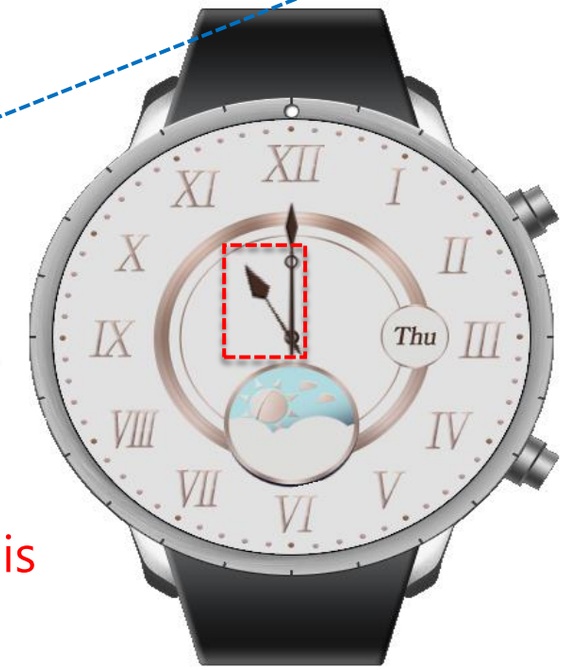
```
static void
move_the_hand(Evas_Object *hand, int degree)
{
    Evas_Map *m = NULL;

    m = evas_map_new(4);
    evas_map_util_points_populate_from_object(m, hand);
    evas_map_util_rotate(m, degree, 360/2, 360/2);
    evas_object_map_set(hand, m);
    evas_object_map_enable_set(hand, EINA_TRUE);
    evas_map_free(m);
}
```

Make just one function for all clock hands
Pass each hand as a parameter

Center axis for rotation is the center of the Watch




hour_hand is
rotated!!




12*30 = 360

```
move_the_hand(ad->hour_hand, hour * 30);
```

# Stage 4: Add operation to the watch

Goal

Fourth, we move the min_hand according to current time
Use ‘evas_map_new’ API to rotate the min_hand
Important thing is the ‘degree’

Call ‘move_the_hand’ function you’ve made before

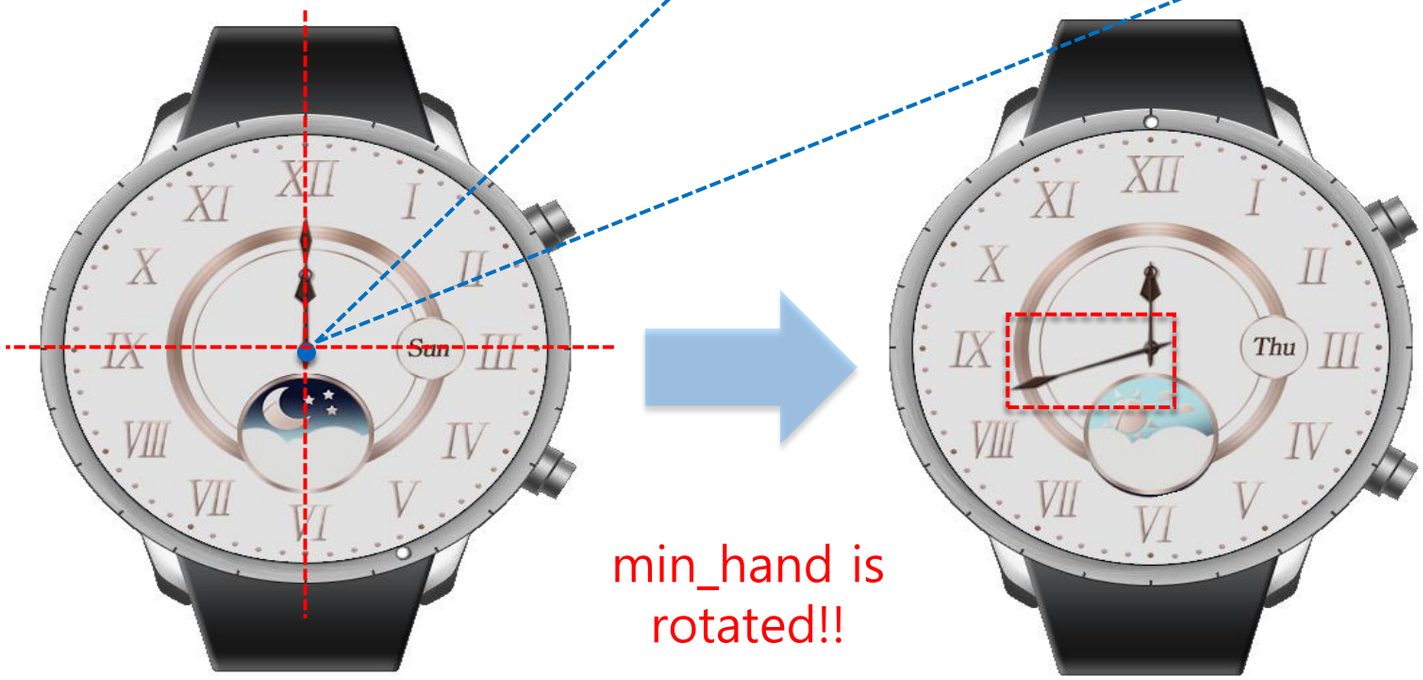
Pass ‘ad->min_hand’ as a parameter

Center axis for rotation is the center of the Watch

```
static void
move_the_hand(Evas_Object *hand, int degree)
{
    Evas_Map *m = NULL;

    m = evas_map_new(4);
    evas_map_util_points_populate_from_object(m, hand);
    evas_map_util_rotate(m, degree, 360/2, 360/2);
    evas_object_map_set(hand, m);
    evas_object_map_enable_set(hand, EINA_TRUE);
    evas_map_free(m);
}
```

60*6 = 360

```
move_the_hand(ad->min_hand, minute * 6);
```




min_hand is rotated!!

# Stage 4: Add operation to the watch

Goal

Fifth, we move the sec_hand according to current time
Use ‘evas_map_new’ API to rotate the sec_hand
Important thing is the ‘degree’

Call ‘move_the_hand’ function you’ve made before

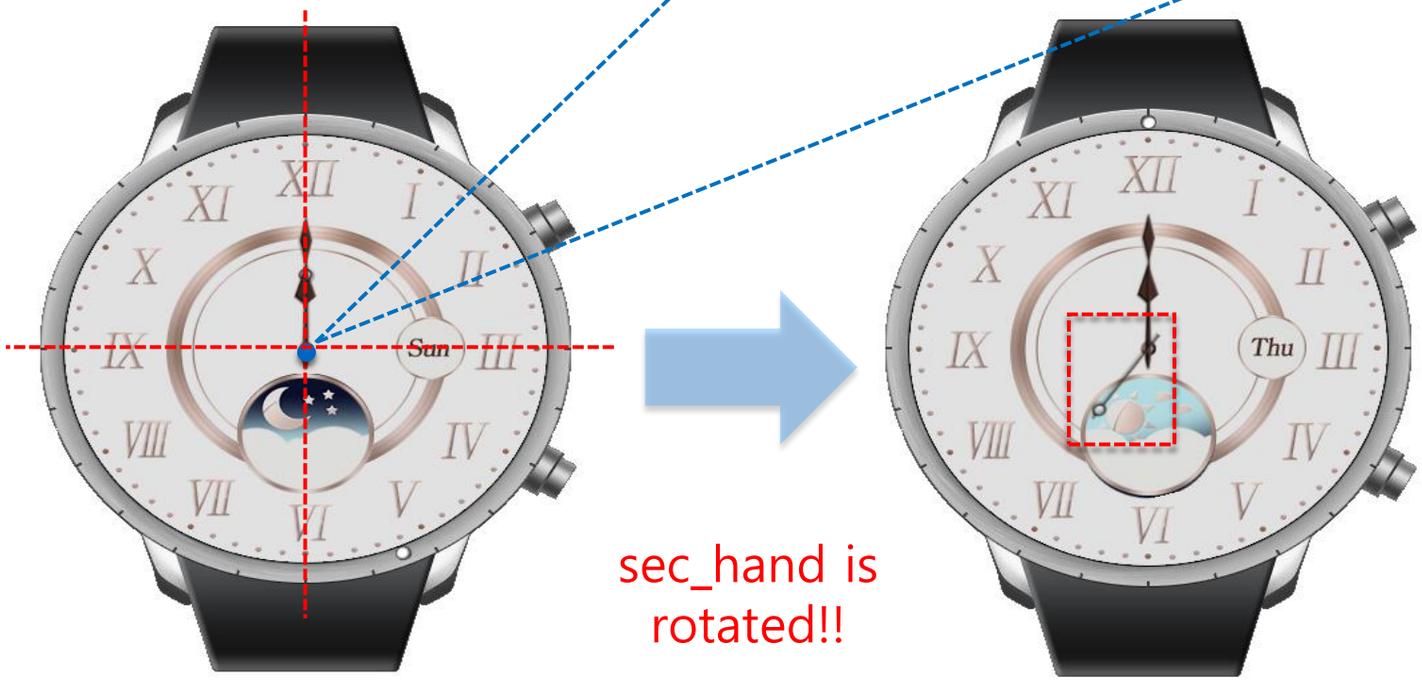
Pass ‘ad->sec_hand’ as a parameter

Center axis for rotation is the center of the Watch

```
static void
move_the_hand(Evas_Object *hand, int degree)
{
    Evas_Map *m = NULL;

    m = evas_map_new(4);
    evas_map_util_points_populate_from_object(m, hand);
    evas_map_util_rotate(m, degree, 360/2, 360/2);
    evas_object_map_set(hand, m);
    evas_object_map_enable_set(hand, EINA_TRUE);
    evas_map_free(m);
}
```

60*6 = 360

```
move_the_hand(ad->sec_hand, second * 6);
```




sec_hand is rotated!!

# Stage 4: Add operation to the watch

Goal

We make our Watch display the accurate time according to current time
But, there is one more thing we have to do
Let's look into the position of our function 'create_base_gui' & 'set_the_time'

**Our code for Watch is here**
**In 'app_create' function**

```c
static bool
app_create(int width, int height, 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, width, height);

    set_the_time(ad);

    return true;
}
```

**And then,**
**When this 'app_create' function will be called ?**

This is very important to update our Watch

**To answer this,**
**Let's understand lifecycle Of Tizen Native Application**

**How often this 'app_create' function will be called?**

# Stage 4: Add operation to the watch

Goal

In normal Applications of Tizen, there are Five Lifecycle Callback function

**'app_create' will be called just one time from launch to terminate**

**'app_control' will be called if application is already launched when the another application request to launch**

**'app_pause' will be called if window of the application is covered or hidden**

**'app_pause' will be called if window of the application is show**

**'app_pause' will be called when terminate the application**

```c
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 */
    return 1;
}

static void
app_control(app_control_h app_control, void *data)
{
    /* Handle the launch request. */
}

static void
app_pause(void *data)
{
    /* Take necessary actions when application becomes invisible. */
}

static void
app_resume(void *data)
{
    /* Take necessary actions when application becomes visible. */
}

static void
app_terminate(void *data)
{
    /* Release all resources. */
}
```

**Especially, some application like Watch has more lifecycle callback functions**

# Stage 4: Add operation to the watch

Goal

In normal Applications of Tizen, there are Five Lifecycle Callback function

**‘app_time_tick’**
**will be called every seconds**

**‘app_ambient_tick’**
**will be called every miutes &**
**when watch become ambient or not**

**‘app_pause’**
**will be called when**
**watch become ambient or not**

```c
static void
app_time_tick(watch_time_h watch_time, void *data)
{
    /* Called at each second while your app is visible. Update watch UI. */
}

static void
app_ambient_tick(watch_time_h watch_time, void *data)
{
    /* Called at each minute while the device is in ambient mode. Update watch UI. */
}

static void
app_ambient_changed(bool ambient_mode, void *data)
{
    /* Update your watch UI to conform to the ambient mode */
}
```

**Using these Lifecycle Callback functions,**
**we can update our watch every seconds**

# Stage 4: Add operation to the watch

Goal

We'll update our Watch every seconds

It's very easy to update Watch if you understand about Lifecycle

**Remove given code as default**
**(we don't need to use this)**

**Call 'set_the_time' function**
**In 'app_time_tick' function**

**Call the function you want to be called every minutes**

**Call the function you want to be called when application become ambient or not**

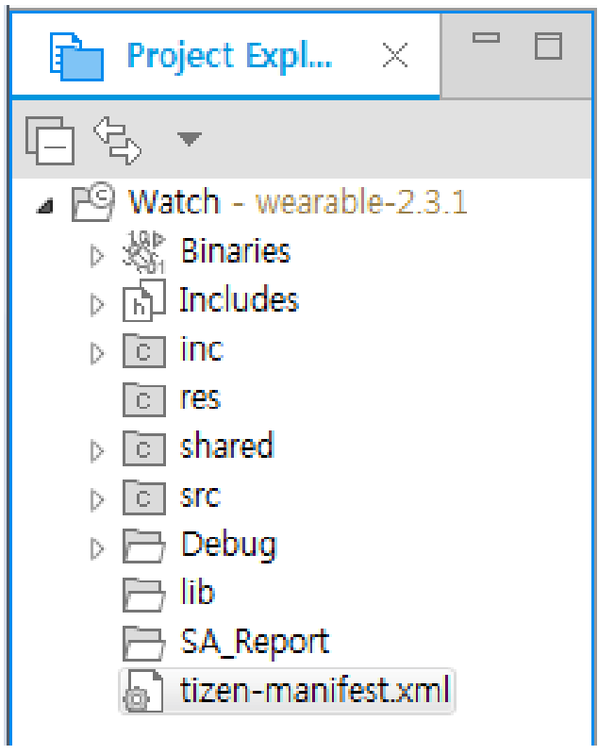
```c
static void
app_time_tick(watch_time_h watch_time, void *data)
{
    /* Called at each second while your app is visible. Update watch UI. */
    appdata_s *ad = data;
    update_watch(ad, watch_time, 0);
}

static void
app_ambient_tick(watch_time_h watch_time, void *data)
{
    /* Called at each minute while the device is in ambient mode. Update watch UI. */
    appdata_s *ad = data;
    update_watch(ad, watch_time, 1);
}

static void
app_ambient_changed(bool ambient_mode, void *data)
{
    /* Update your watch UI to conform to the ambient mode */
}
```

```c
static void
app_time_tick(watch_time_h watch_time, void *data)
{
    /* Called at each second while your app is visible. Update watch UI. */
    appdata_s *ad = data;
    set_the_time(ad);
}

static void
app_ambient_tick(watch_time_h watch_time, void *data)
{
    /* Called at each minute while the device is in ambient mode. Update watch UI. */
}

static void
app_ambient_changed(bool ambient_mode, void *data)
{
    /* Update your watch UI to conform to the ambient mode */
}
```

# Stage 4: Add operation to the watch

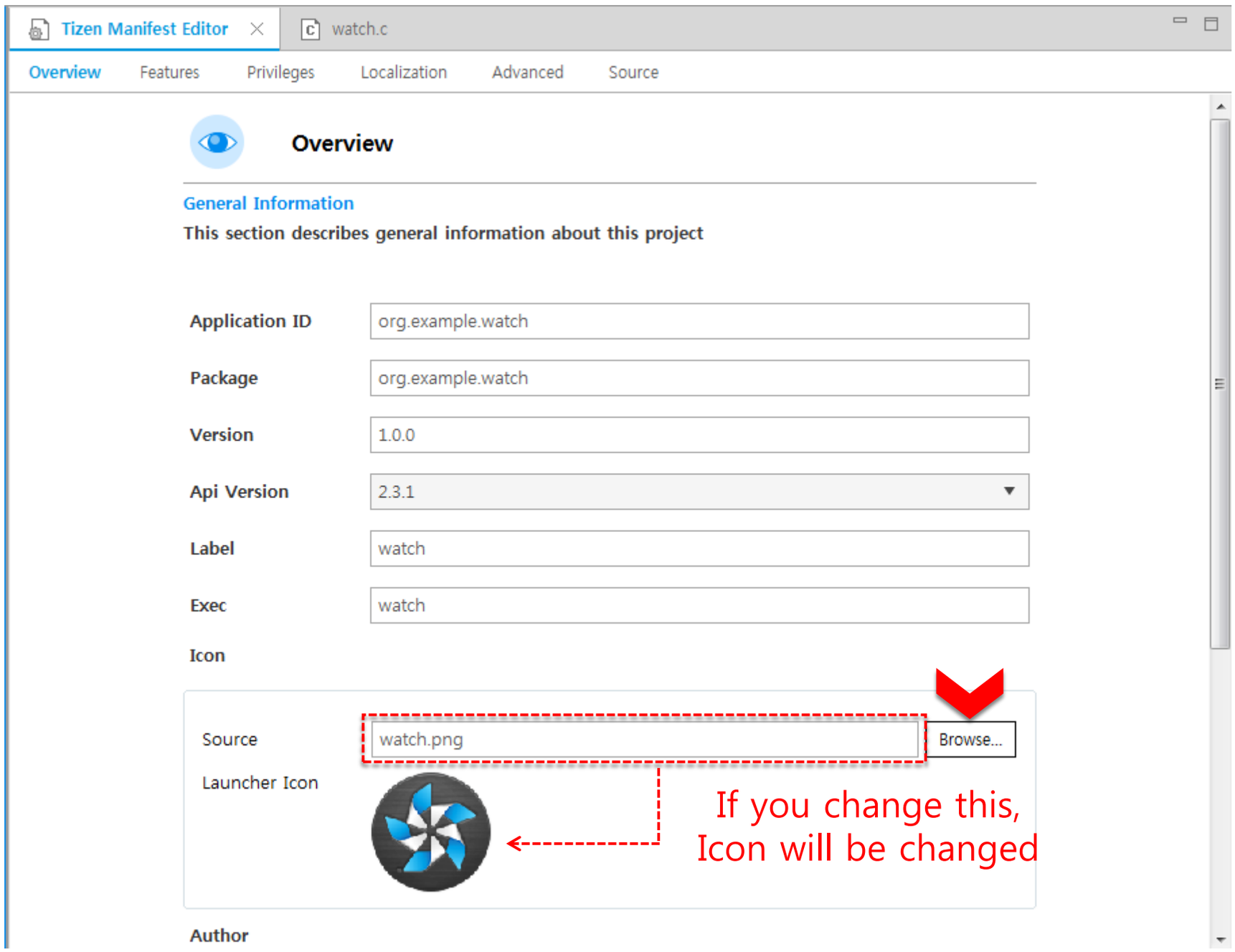
Goal

## In addition, you can change the Icon for your Watch




**Double click on ‘tizen-manifest.xml’**

**You can find Overview of the Watch Project on the right**



# Stage 4: Add operation to the watch

Goal

In addition, you can change the Icon for your Watch

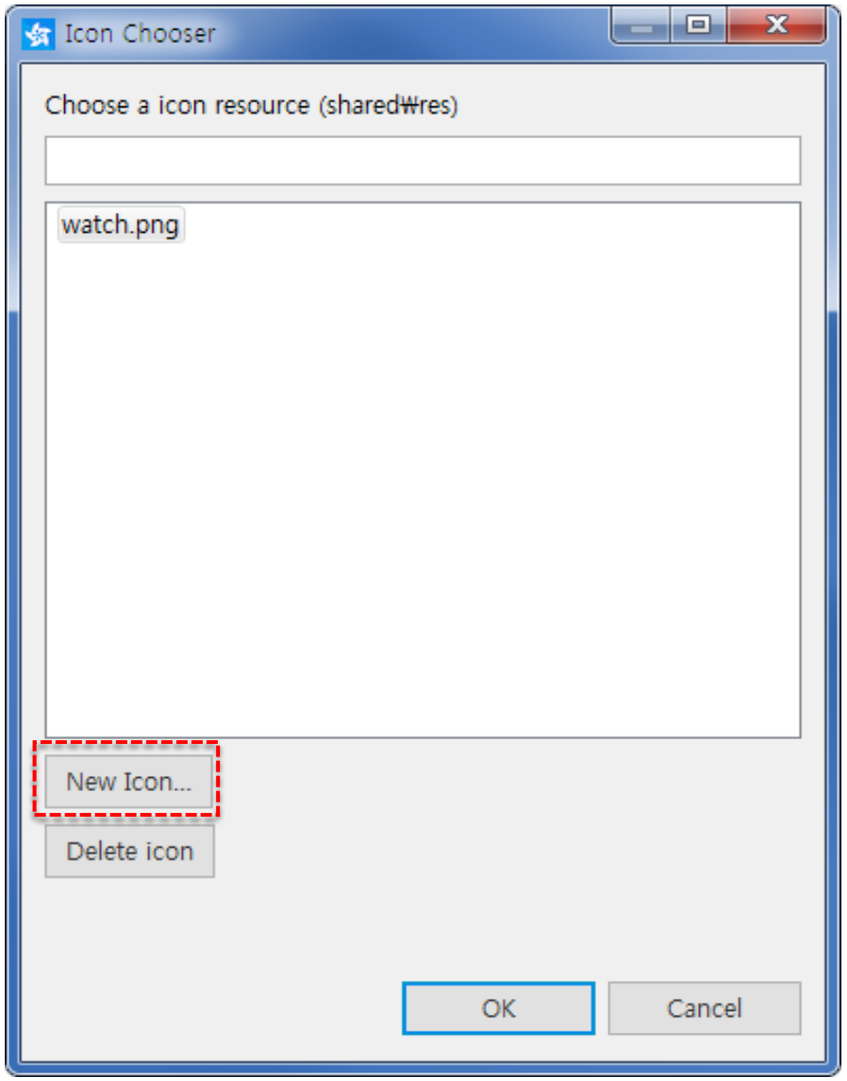
**New Icon**

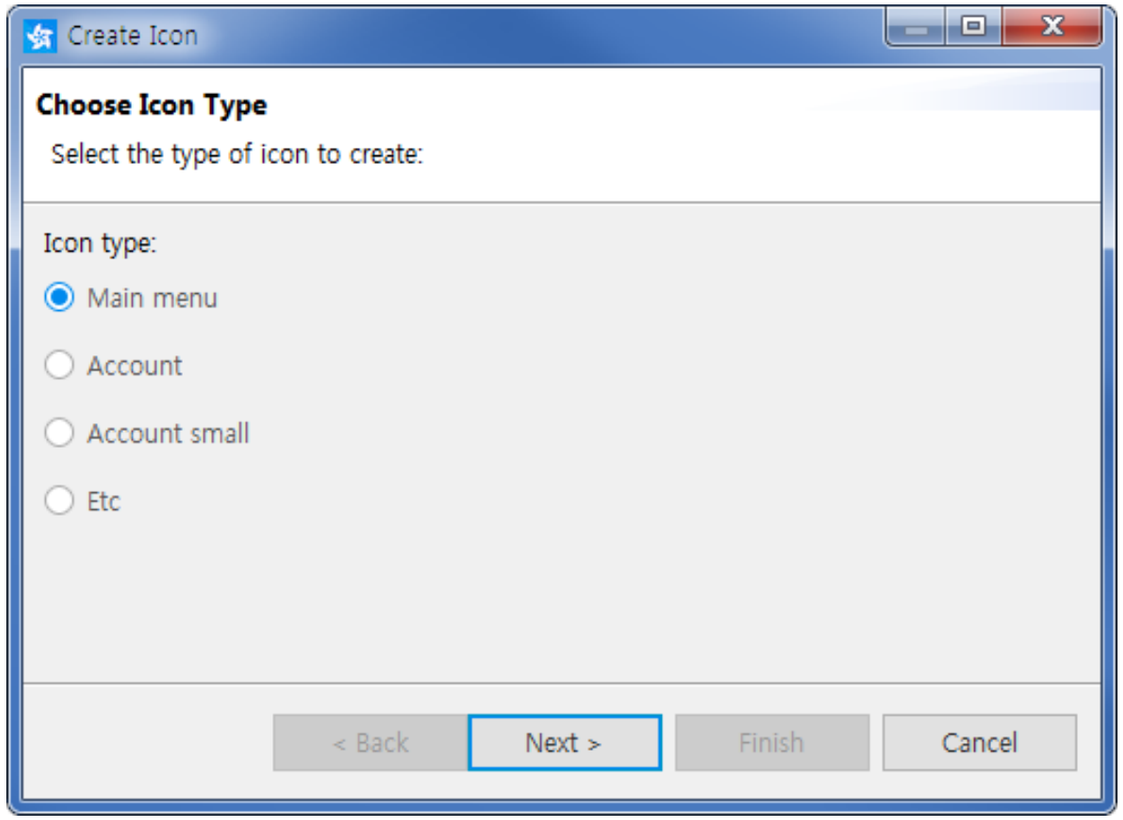
↓

**Browse…**

↓

**Select Image you want**



# Stage 4: Add operation to the watch

Goal

In addition, you can change the Icon for your Watch

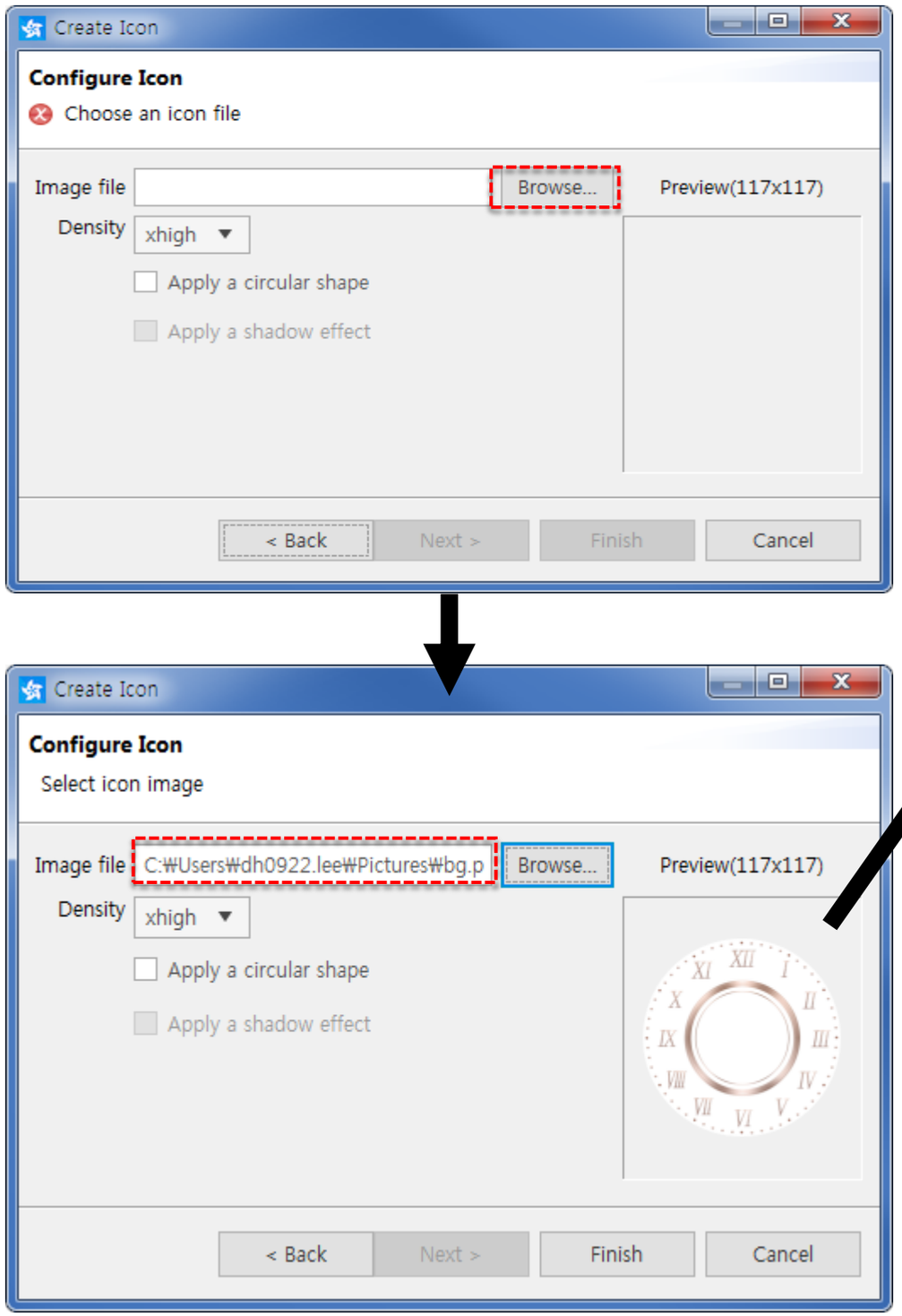
**New Icon**

↓

**Browse…**

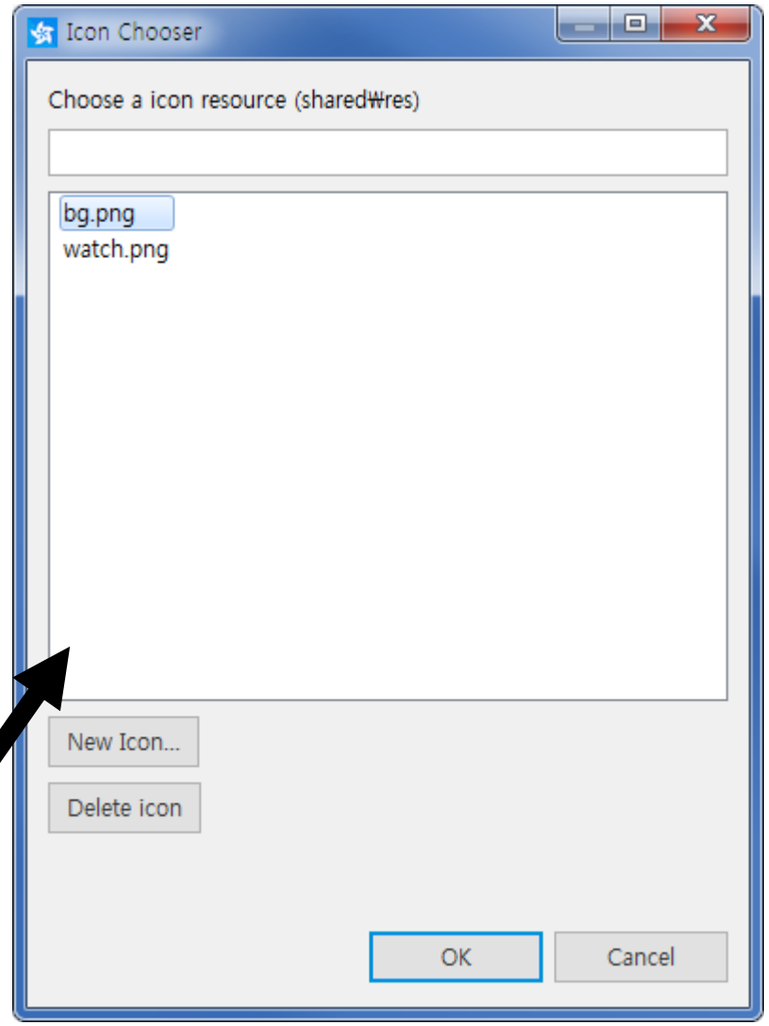
↓

**Select Image you want**










**You can find the image you selected on the list**

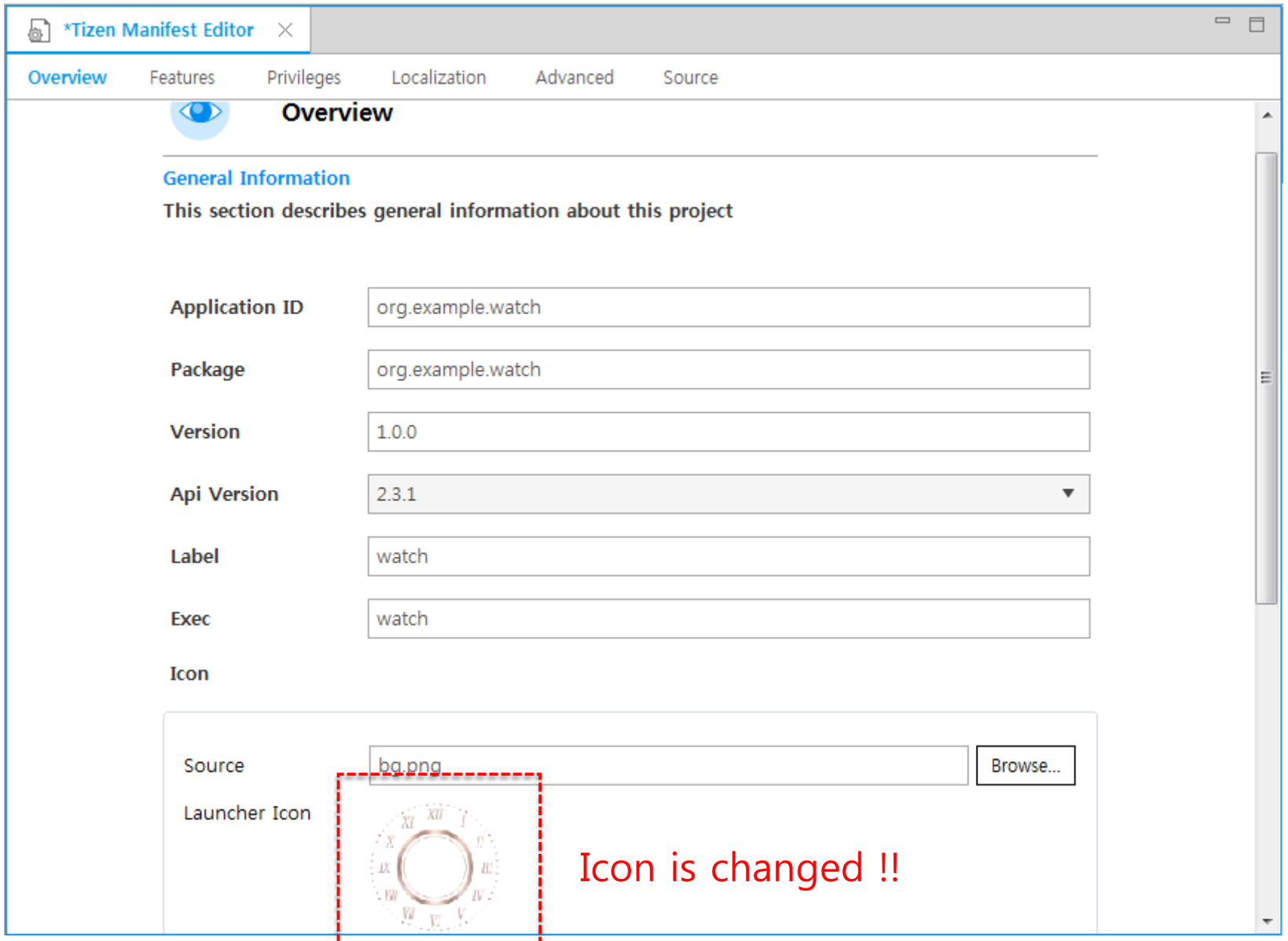
↓

**OK**

# Stage 4: Add operation to the watch

Goal

In addition, you can change the Icon for your Watch




**Also, in the settings menu you can find changed Icon**

# Stage 4: Add operation to the watch

Goal

Finally, remove the unused code

Tizen SDK show the warning in the source code like below

**Track the warning symbol and check the reason**

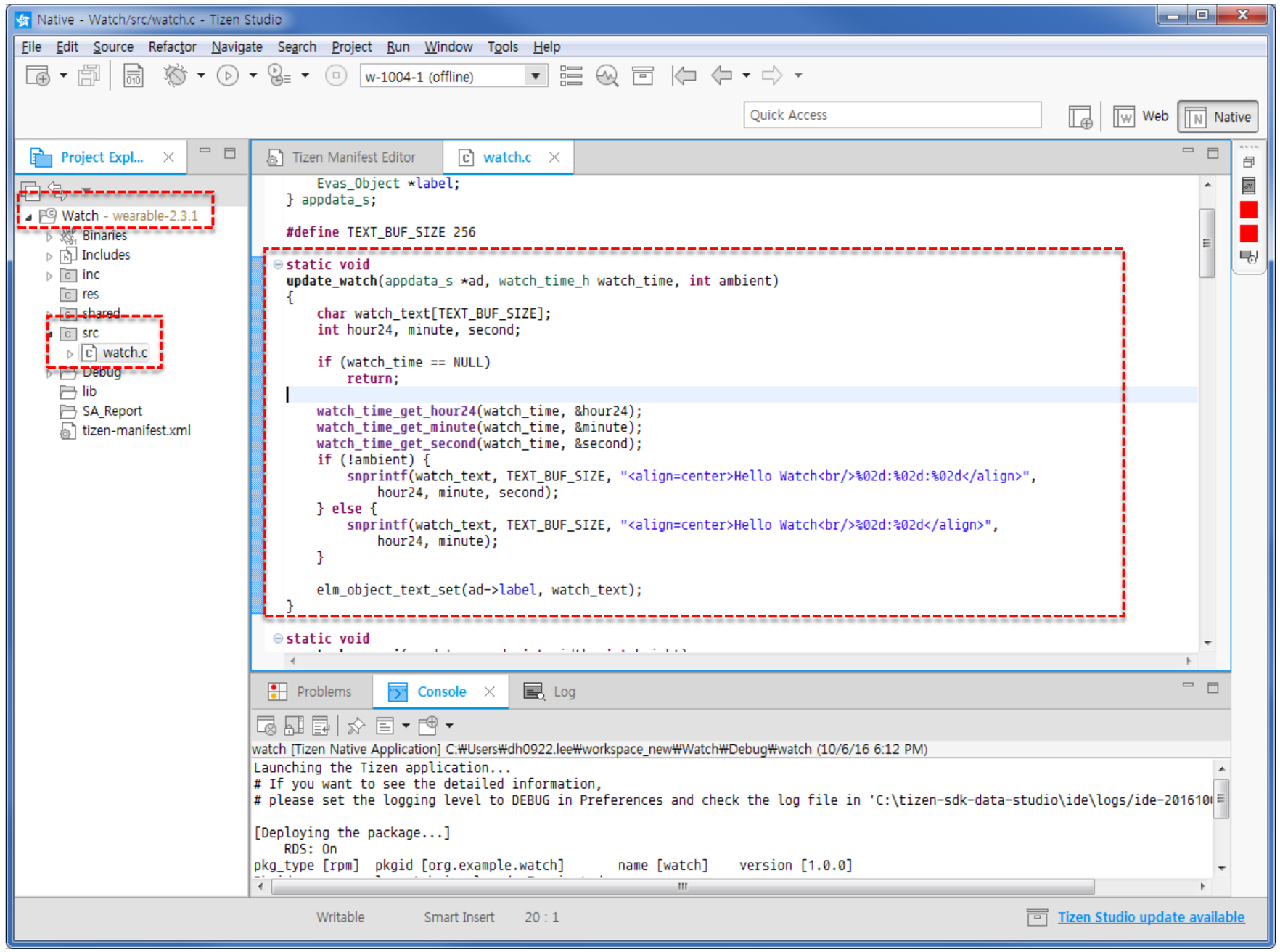
↓

**In this case, there's unused code !**

↓

**Remove &**

**Run the Watch project !**



# Stage 4: Add operation to the watch

Goal

Now, you get the Watch Face with Tizen

You can customize more, change image, display battery information and so on

**If you want to be more familiar with Tizen, visit here**

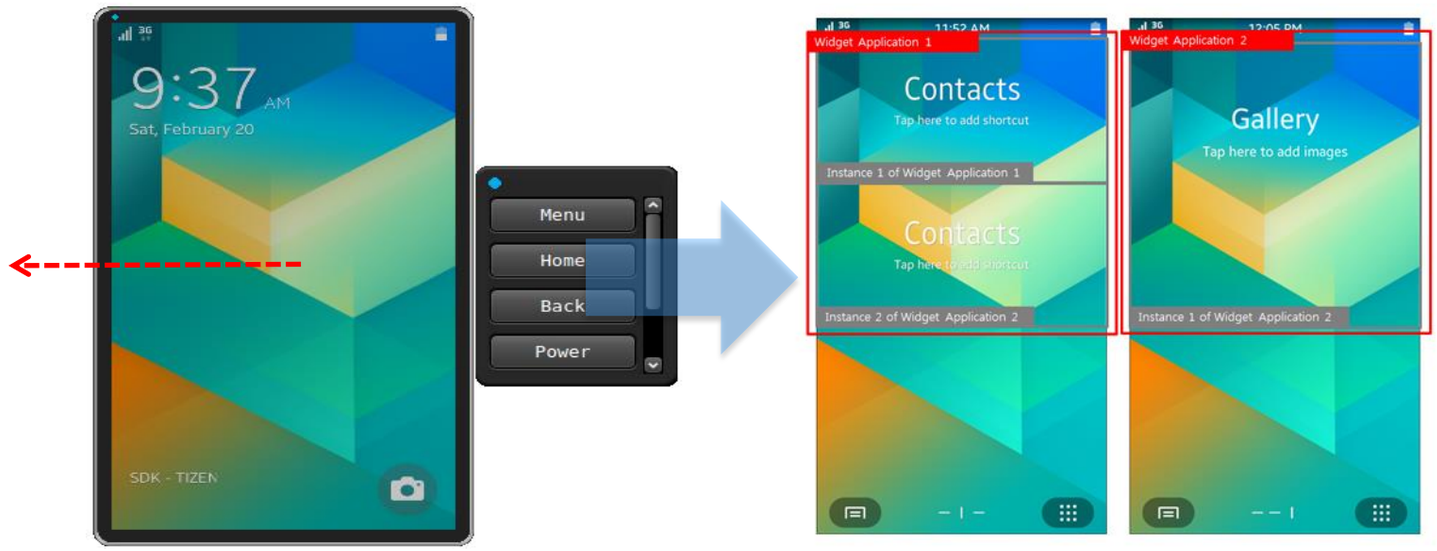
**https://developer.tizen.org/**

# Implementation of Widget Application

# Overview – Widget Application

There are two types of Native Application
One is UI Application you already experienced
The another is Widget Application you will experience from now

**Widget Applications**
**Can be found at the**
**Homescreen**




**Same widgets can be found**
**To show different information**

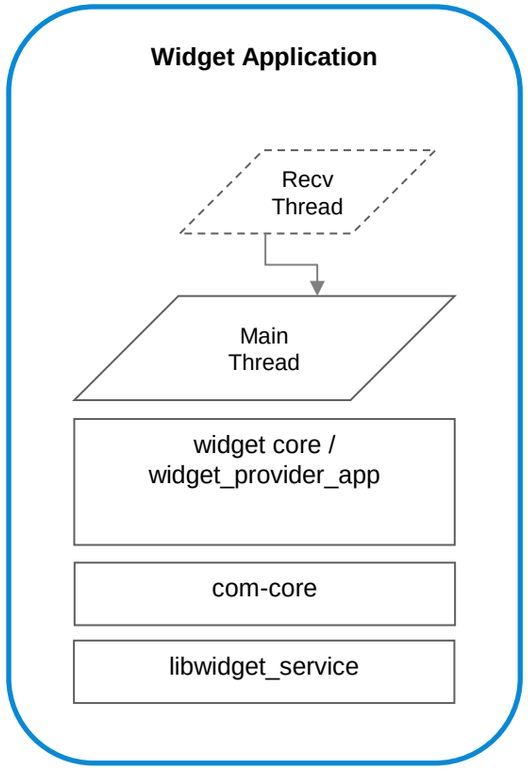
**Widget Application**
**can be connected with**
**UI Application**

# Implementation Plan

We will proceed the implementation of widget app in 2 stages.

## Stage 1.

**Understanding of Widget Application**




## Stage 2.

**How to develop Widget Application**

# Stage 1: Understanding of Widget Application

Goal

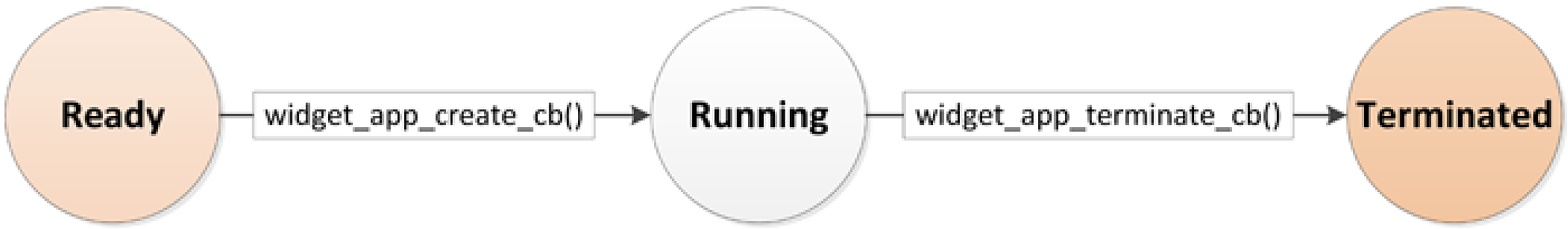
The crucial difference between UI App and Widget App is Life Cycle

Widget Application has one more step of life cycle for Instance

```c
int main(int argc, char *argv[])
{
    widget_app_lifecycle_callback_s ops = {0, };
    int ret;

    ops.create = widget_app_create;
    ops.terminate = widget_app_terminate;

    ret = widget_app_main(argc, argv, &ops, NULL);
    if (ret != WIDGET_ERROR_NONE) {
        dlog_print(DLOG_ERROR, LOG_TAG, "widget_app_main() is failed. err = %d", ret);
    }

    return ret;
}
```

**There are only two Life Cycle Callbacks**

**So, Application state is more simple than UI Application**




**But, there are one more step of life cycle,**

**Let's go to 'widget_app_create'**

# Stage 1: Understanding of Widget Application

Goal

Widget Application  can be made multiple same widget instances

Because of this, Widget Application should have Life Cycle for Instance

```
static widget_class_h widget_app_create(void *user_data)
{
    app_event_handler_h handlers[5] = {NULL, };

    widget_app_add_event_handler(&handlers[APP_EVENT_LANGUAGE_CHANGED],
            APP_EVENT_LANGUAGE_CHANGED, widget_app_lang_changed, user_data);
    widget_app_add_event_handler(&handlers[APP_EVENT_REGION_FORMAT_CHANGED],
            APP_EVENT_REGION_FORMAT_CHANGED, widget_app_region_changed,
            user_data);

    widget_instance_lifecycle_callback_s ops = {
            .create = widget_instance_create,
            .destroy = widget_instance_destroy,
            .pause = widget_instance_pause,
            .resume = widget_instance_resume,
            .update = widget_instance_update,
            .resize = widget_instance_resize,
    };

    return widget_app_class_create(ops, user_data);
}
```

**UI Application create UI component in ‘app_create’**

**Widget Application just create Class for widget Instance in ‘app_create’**

**UI for each Widget Instance be create in ‘widget_instance_create’**

**Widget Instance’s Life Cycle Callbacks & state**

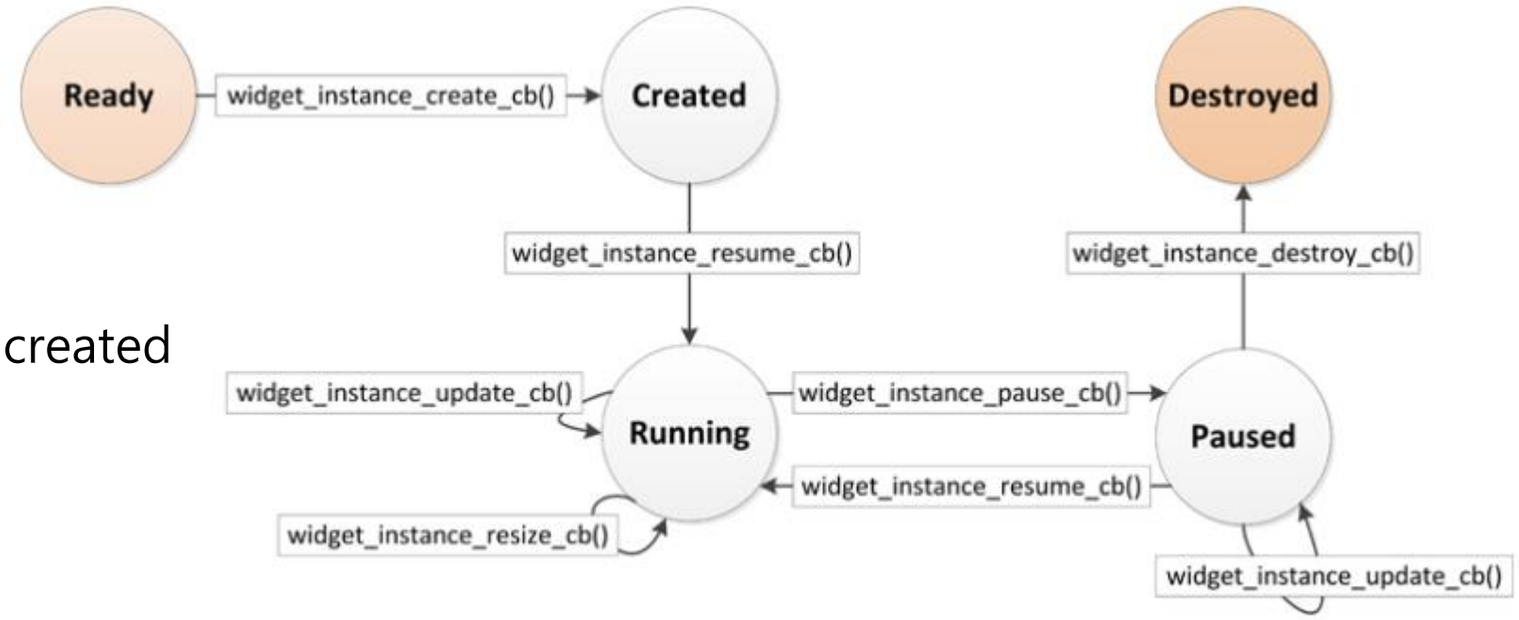
# Stage 1: Understanding of Widget Application

Goal

Widget Instance state is more similar with UI Application state

There are Five states and Six Life Cycle Callbacks

## Life Cycle of Widget Instance




**widget_instance_create:**

Called after the widget instance is created

**widget_instance_destroy:**

Called before the widget instance is destroyed

**widget_instance_pause:**

Called when widget invisible

**widget_instance_resume:**

Called when widget is visible

**widget_instance_resize:**

Called before widget size is changed

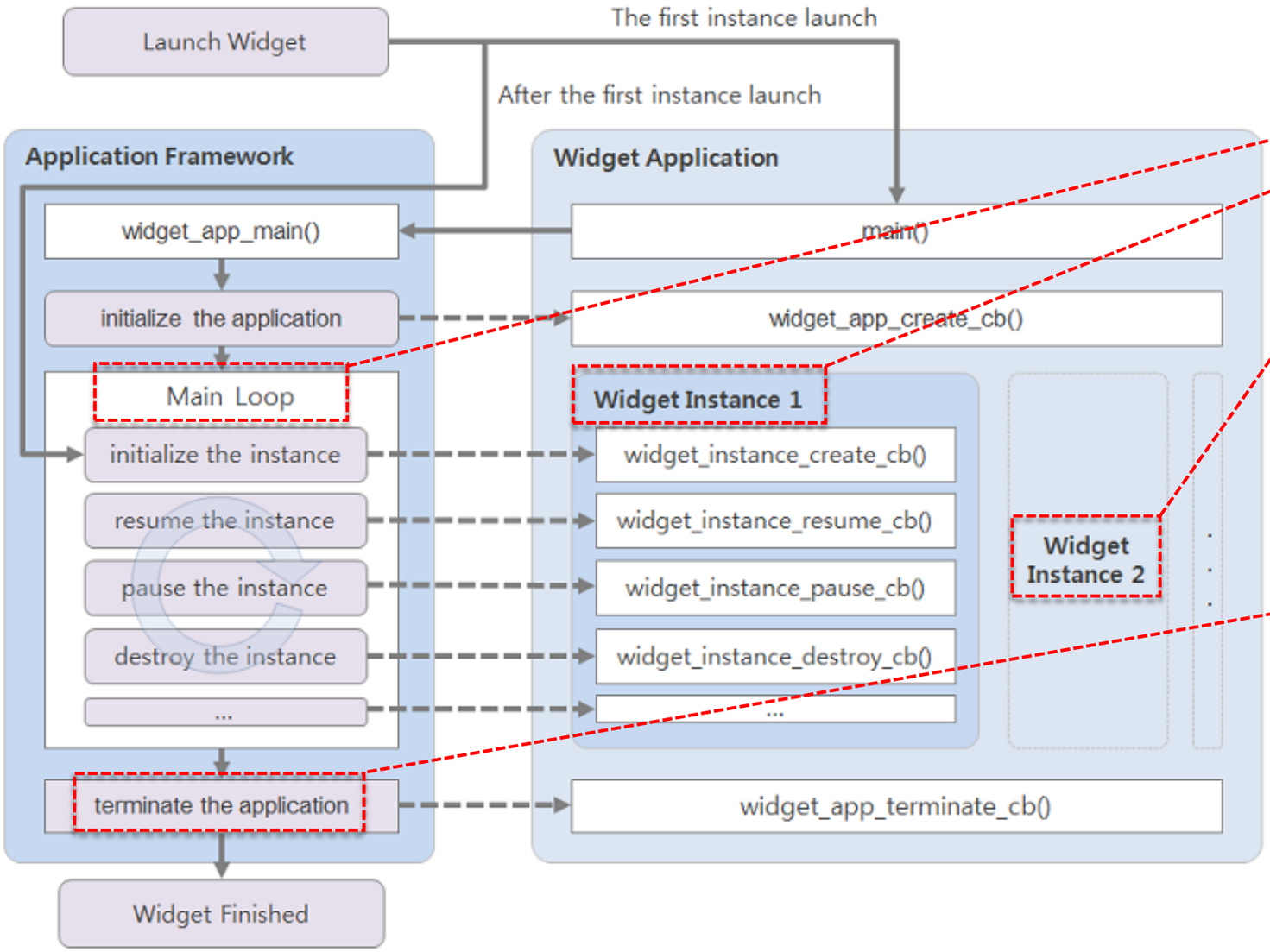
**widget_instance_update:**

Called when an event for updating widget is received

# Stage 1: Understanding of Widget Application

Goal

Multiple creating of Instance progress is like below
After Widget Application initialization at the beginning,
launch request goes to ‘widget_instance_create’ directly like below




**One process(Main Loop), Multiple Instances**

**Because of One Process, if Widget Application is terminated,
Every Instances are terminated**

# Stage 2: Development of Widget

Let’s make a Alarm Widget Application

Follow up, and make your own Alarm Widget

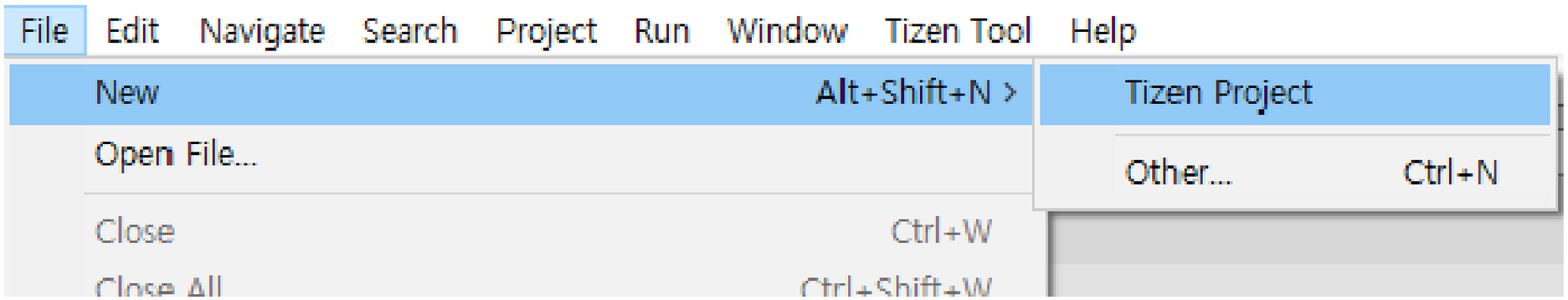

**Tizen will provide wonderful experience on your development**

# Stage 3: Development of Widget

To start Development,

Create New [Tizen Native Project] !

Tizen Studio provide some Templates for the easy start

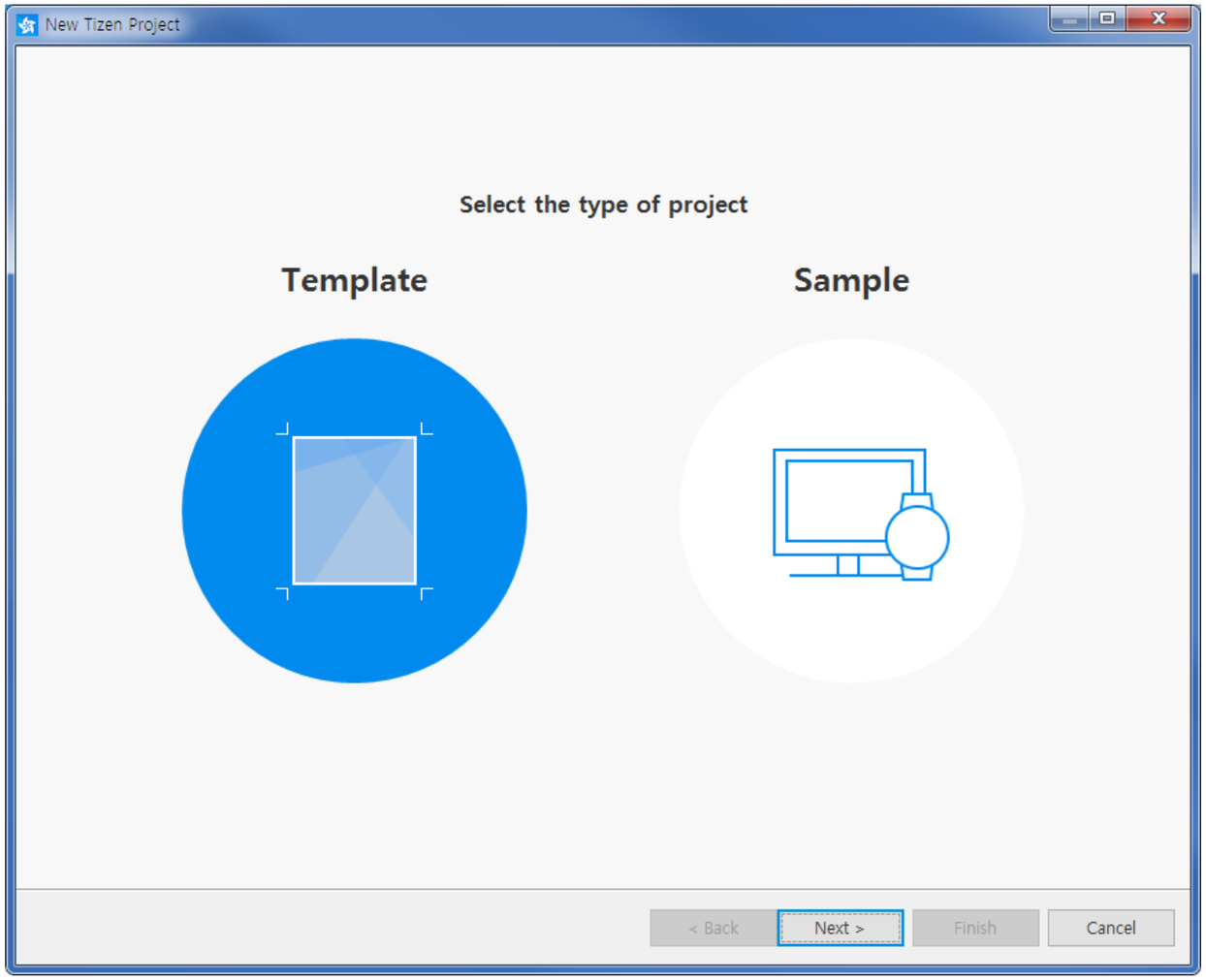


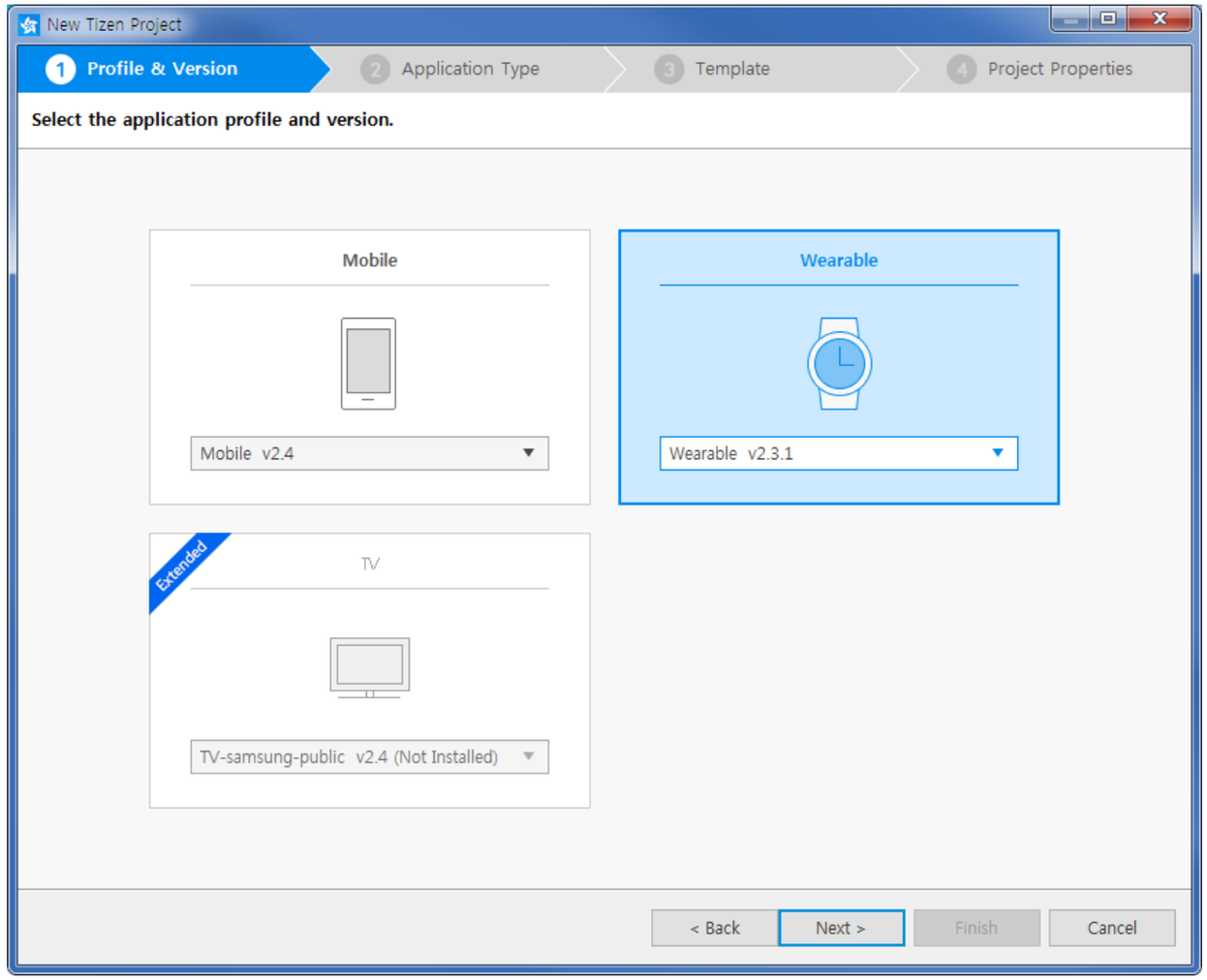
# Stage 3: Development of Widget

Goal

Choose the Template most similar with what you want to develop

In this case, we'll choose Widget



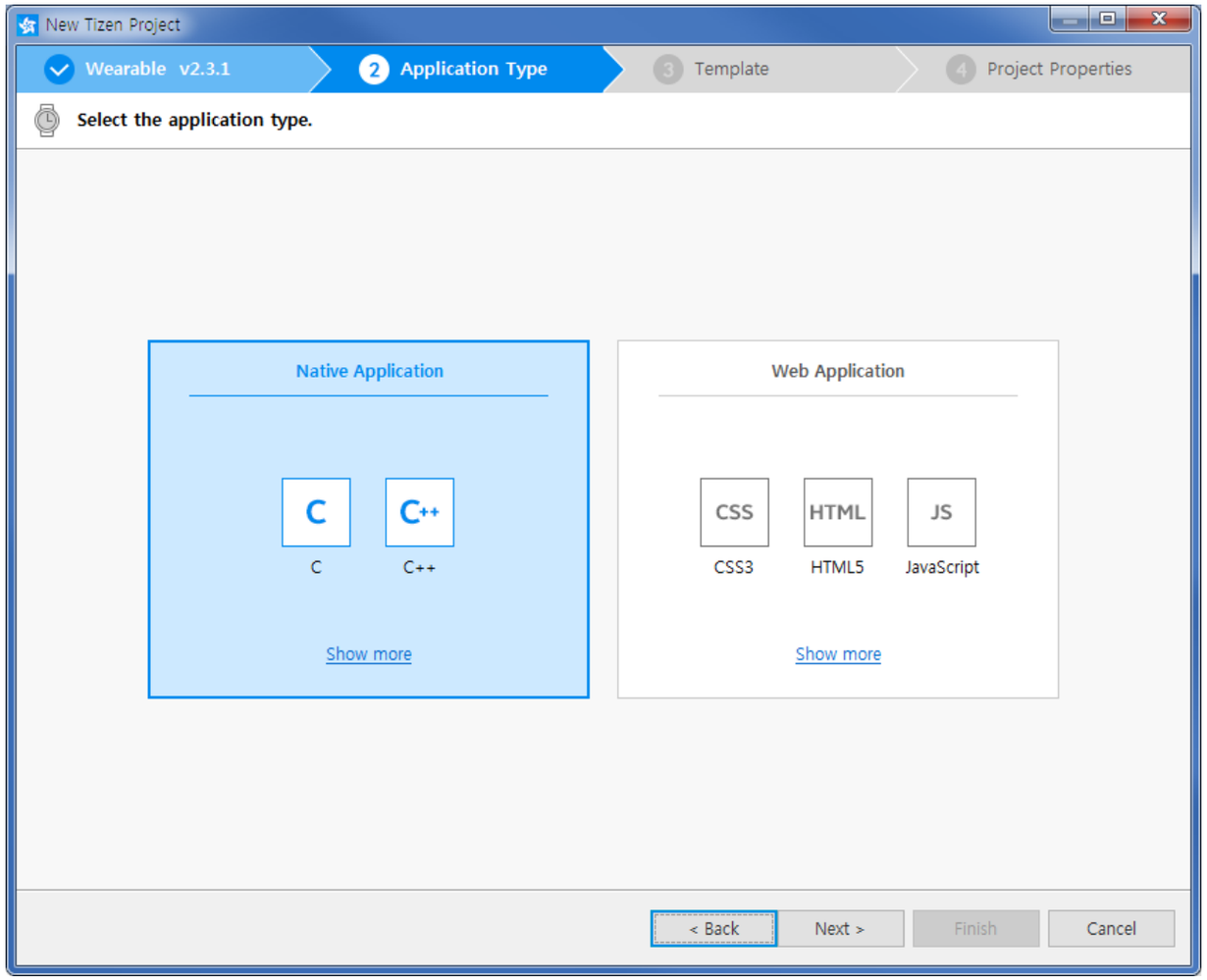



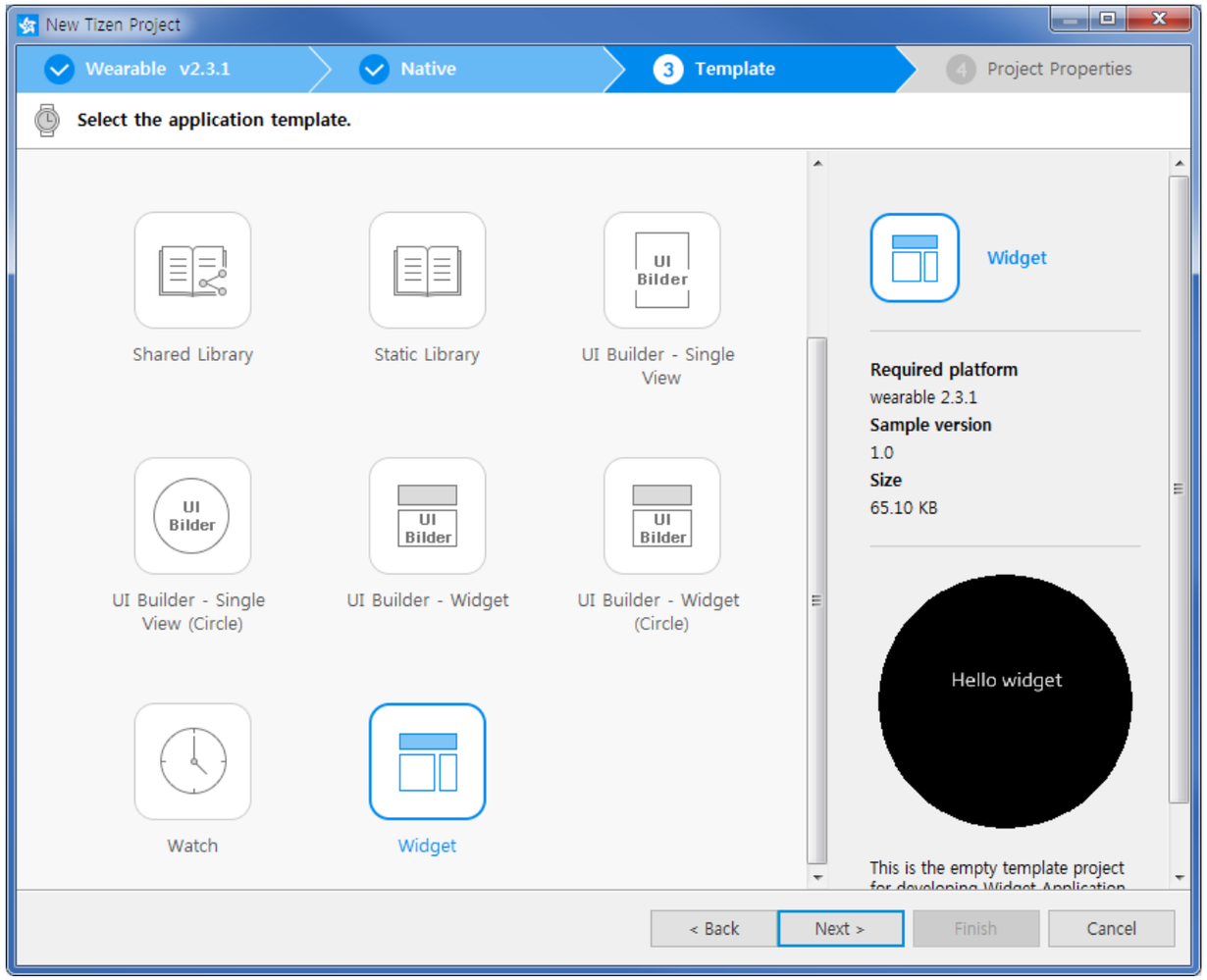
# Stage 3: Development of Widget

Goal

Choose the Template most similar with what you want to develop

In this case, we'll choose Widget

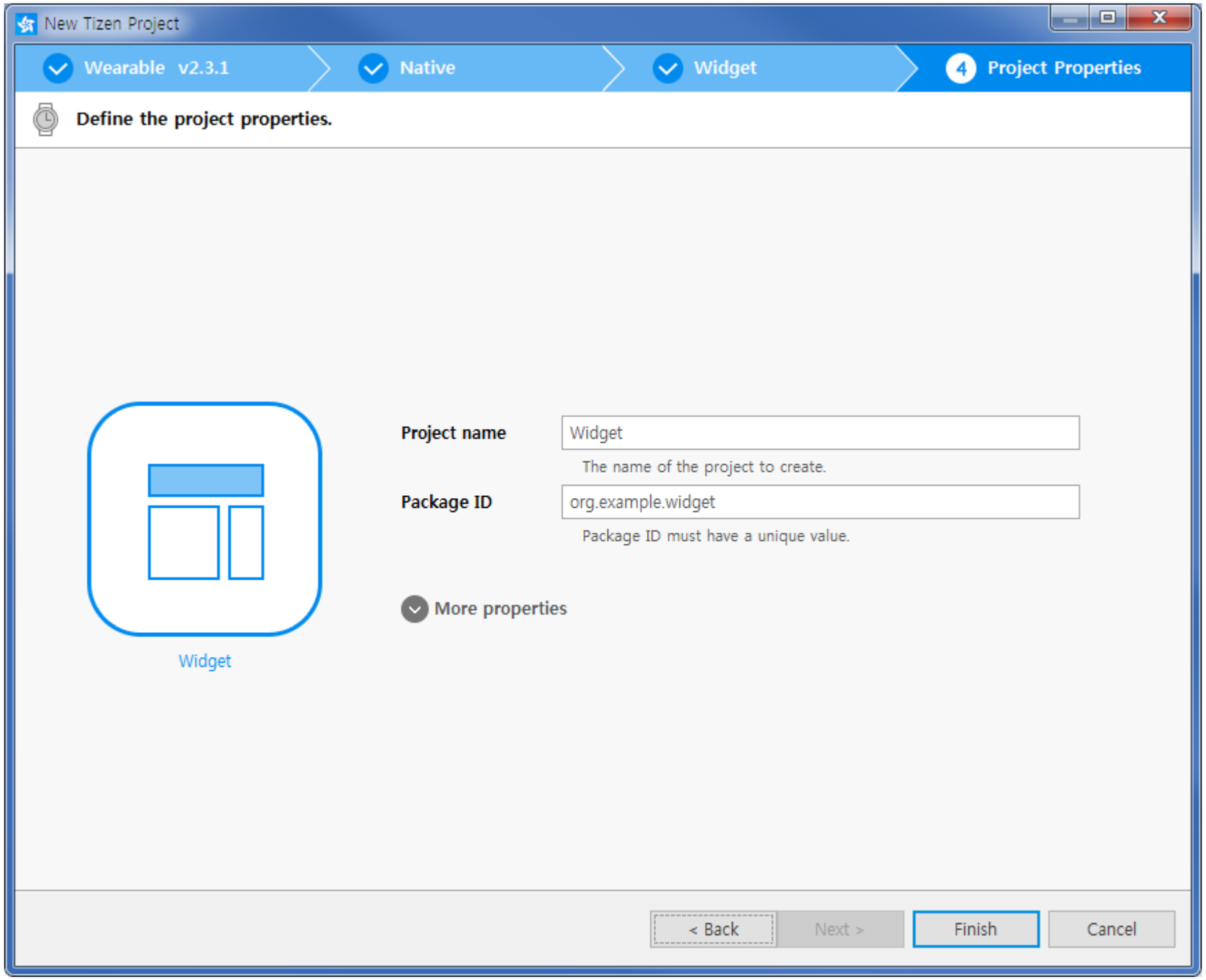


# Stage 3: Development of Widget

Choose the Template most similar with what you want to develop

In this case, we'll choose Widget



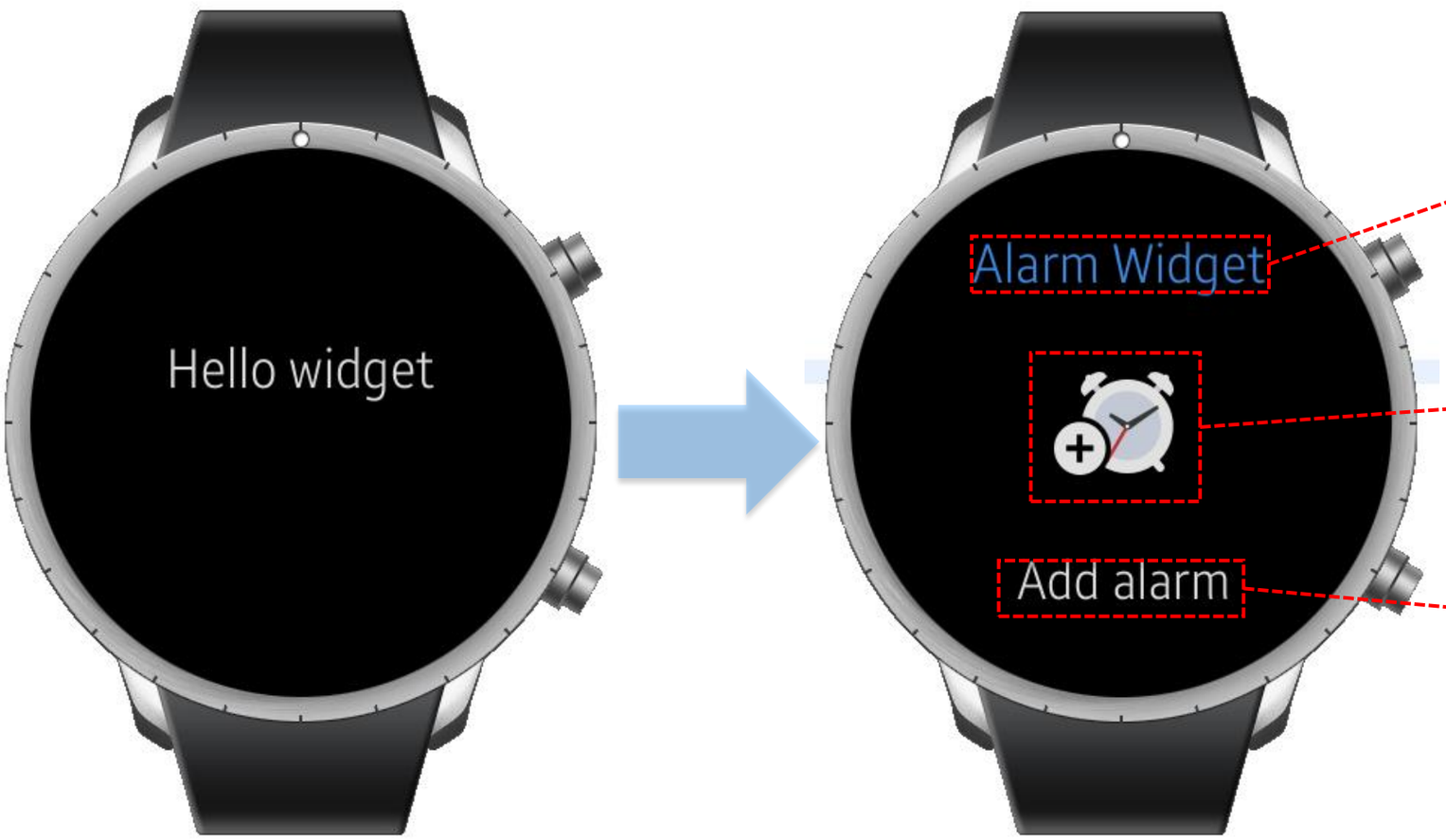
# Stage 4: Development of Widget

Goal

At first, analyze the Alarm Widget

In this class, we just create  Main View for state of no alarm like below

**Run & check Template**




**Main View consists of Three Parts**

**Text part for Title**

**Button part for click event with image**

**Text part for detail explanation text**

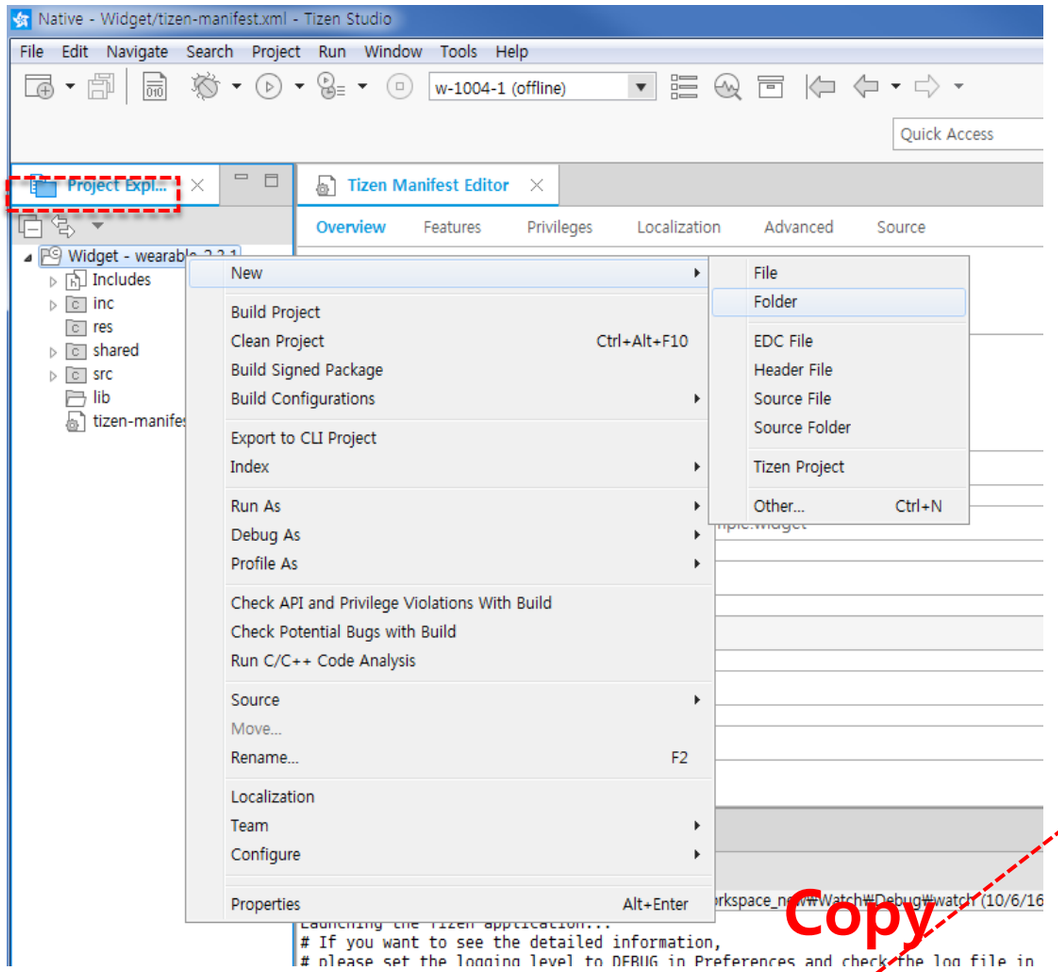
**So, we need two text objects and one button object**

# Stage 4: Development of Widget

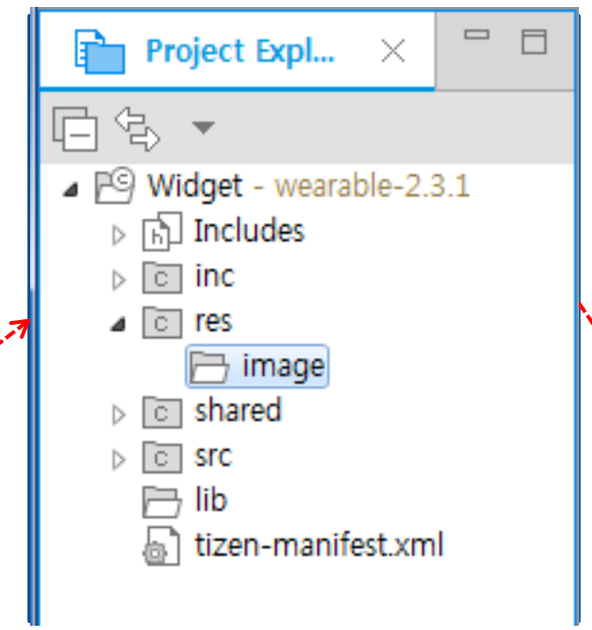
Goal

Check source files

Create ‘images’ folder under ‘res’ folder

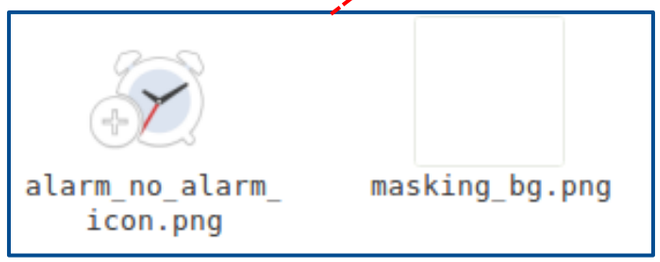


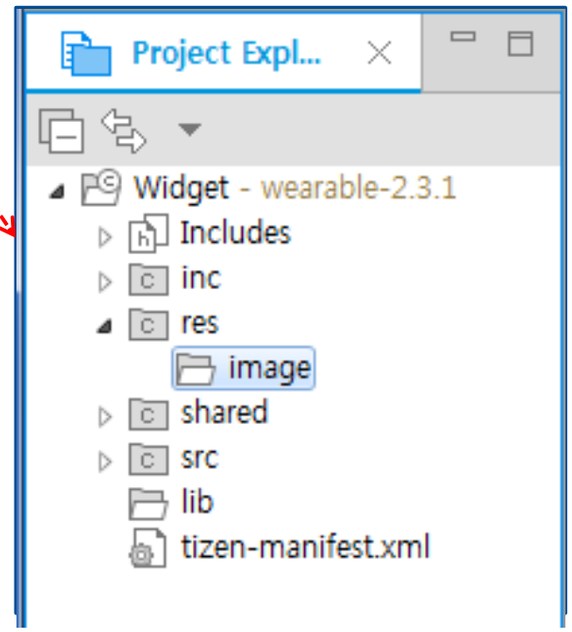

**This folder is for images**
**that we will use for button object**




Copy

Paste



# Stage 4: Development of Widget

Goal

It is recommend do not changed the Window and Conformant

**On the top of ‘widget.c’ file, ‘widget.h’ file is included This library provide many useful APIs**

**This window is dependent on the Homescreen**

**‘label’ is also for text**

**So, we need two objects for Button and one more text**

**This allow user to get data structure ‘wid’ at any functions with ‘context’**

```c
static int
widget_instance_create(widget_context_h context, bundle *content, int w, int h, void *user_data)
{
    widget_instance_data_s *wid = (widget_instance_data_s*) malloc(sizeof(widget_instance_data_s));
    int ret;

    if (content != NULL) {
        /* Recover the previous status with the bundle object. */

    }

    /* Window */
    ret = widget_app_get_elm_win(context, &wid->win);
    if (ret != WIDGET_ERROR_NONE) {
        dlog_print(DLOG_ERROR, LOG_TAG, "failed to get window. err = %d", ret);
        return WIDGET_ERROR_FAULT;
    }

    evas_object_resize(wid->win, w, h);

    /* Conformant */
    wid->conform = elm_conformant_add(wid->win);
    evas_object_size_hint_weight_set(wid->conform, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
    elm_win_resize_object_add(wid->win, wid->conform);
    evas_object_show(wid->conform);

    /* Label*/
    wid->label = elm_label_add(wid->conform);
    evas_object_resize(wid->label, w, h / 3);
    evas_object_move(wid->label, w / 4, h / 3);
    evas_object_show(wid->label);
    elm_object_text_set(wid->label, "Hello widget");

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

    widget_app_context_set_tag(context, wid);
    return WIDGET_ERROR_NONE;
}
```

**These values are decided by ‘tizen-manifest.xml’**

```
<support-size preview="preview.png">2x2</support-size>
```

# Stage 4: Development of Widget

Goal

Change some options of ‘label’ for Title

Add another ‘label’ for Detail Text

```
/* Label*/
wid->label = elm_label_add(wid->conform);

evas_object_resize(wid->label, 300, 80);

evas_object_move(wid->label, 80, 50);

evas_object_color_set(wid->label, 72, 156, 255, 255)

evas_object_show(wid->label);

elm_object_text_set(wid->label, "Alarm Widget");

/* Detail Text */
Evas_Object *detail_text = NULL;

detail_text = elm_label_add(wid->conform);

evas_object_resize(detail_text, 160, 80);

evas_object_move(detail_text, 110, 260);

evas_object_show(detail_text);

elm_object_text_set(detail_text, "<font_size=35>Add alarm</font_size>");
```


**Default text color is white**

**This way, can set font style, size, align etc**

# Stage 4: Development of Widget

Goal

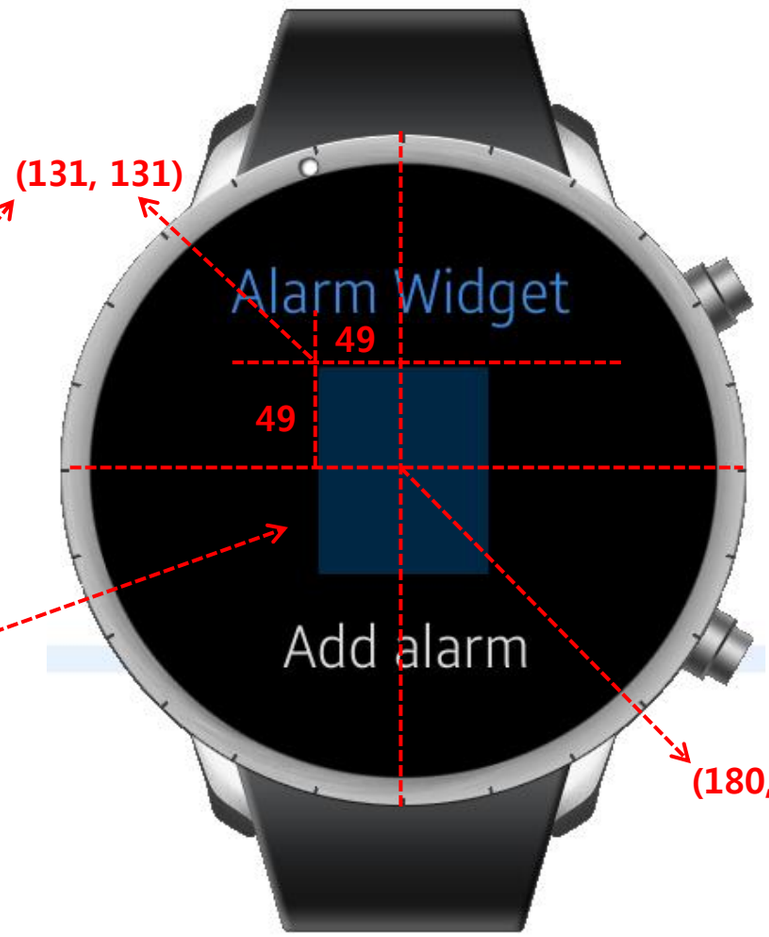
Add button for ‘click event’

When the button is clicked, we should do something

Button’s default style is rectangle and blue

```c
static void _add_alarm_cb(void *data, Evas_Object *obj, void *event_info)
{
    dlog_print(DLOG_INFO, LOG_TAG, "Add alarm is clicked");
}
```

```c
/* Add Alarm Button */
Evas_Object *button = NULL;

button = elm_button_add(wid->conform);

evas_object_resize(button, 98, 98);

evas_object_move(button, 180 - 98/2, 180 - 98/2);

evas_object_smart_callback_add(button, "clicked", _add_alarm_cb, NULL);

evas_object_show(button);
```




**Because, size of image for button is 98x98**

**Starting coordinate is left top,**
**To set to center, move to – (button size/2) from center**

**This function register callback function will be operated when button is clicked**

# Stage 4: Development of Widget

Goal

Add Image object to set button image

Get image path as same way that you already did before

```
/* Image for Button */
Evas_Object *img = NULL;
char *resource_path = NULL;
char image_path[1024];

resource_path = app_get_resource_path();

snprintf(image_path, sizeof(image_path), "%s%s", resource_path, "images/alarm_no_alarm_icon.png");

img = elm_image_add(button);

elm_image_file_set(img, image_path, NULL);

elm_object_content_set(button, img);
```

**Indicate 'res' folder**

**Indicate [res/images/alarm_no_alarm_icon.png] file**

**Set image file to image object**

**This function set image object to the button**


**But, is looks strange**

**Because default style of the button is blue and Image file we used is transparent**

**How can resolve this problem?**

# Stage 4: Development of Widget

Goal

Add button style

Check whether click event is working properly or not

**Add ‘transparent’ style to make button transparent**

```
button = elm_button_add(wid->conform);

elm_object_style_set(button, "transparent");
```


**How to check ‘click event’ ?**

```
static void _add_alarm_cb(void *data, Evas_Object *obj, void *event_info)
{
    dlog_print(DLOG_INFO, LOG_TAG, "Add alarm is clicked");
}
```

```
#ifdef  LOG_TAG
#undef  LOG_TAG
#endif
#define LOG_TAG "widget"
```

**DLOG_DEBUG : D**
**DLOG_WARN : W**
**DLOG_ERROR : E**

**In ‘inc/widget.h’ file, this log tag is declared**

Problems | Tasks | Console | Call Stack | Log

0 items

V D I W E F

emulator-26101 (alarm)

| Time | Level | Pid | Tid | Tag | Message |
|---|---|---|---|---|---|
| 02-22 20:28:32.800 | Info | 9893 | 9893 | widget | Add alarm is clicked |
| 02-22 20:28:39.480 | Info | 9893 | 9893 | widget | Add alarm is clicked |
| 02-22 20:28:53.490 | Info | 9893 | 9893 | widget | Add alarm is clicked |
| 02-22 20:29:00.690 | Info | 9893 | 9893 | widget | Add alarm is clicked |
| 02-22 20:29:02.730 | Info | 9893 | 9893 | widget | Add alarm is clicked |
| 02-22 20:29:03.710 | Info | 9893 | 9893 | widget | Add alarm is clicked |
| 02-22 20:29:19.520 | Info | 9893 | 9893 | widget | Add alarm is clicked |

Tag | widget

# Stage 4: Development of Widget

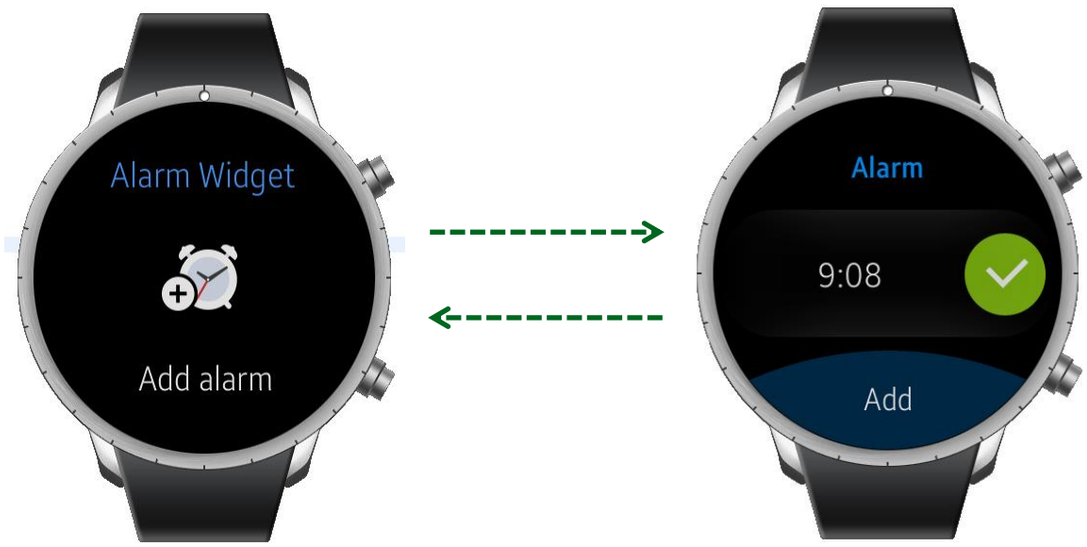
Good job !!

We finished to develop Alarm Widget Application

But there is one more thing !!

Widget Application can be connected with UI Application !!

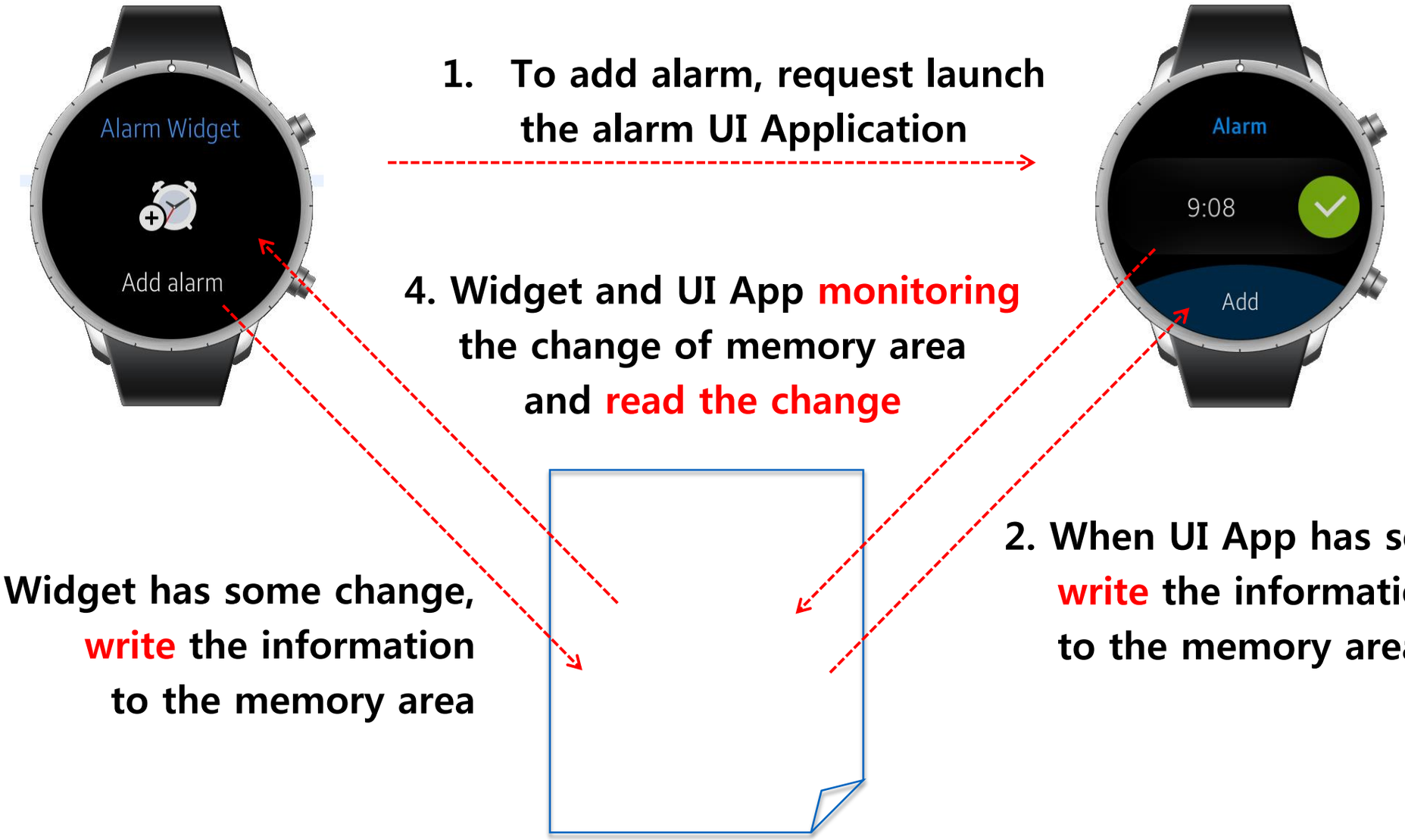

**Next class,**
**We will connect Alarm Widget with Alarm UI Application**



# Stage 5: Connection between Widget & UI

Goal

The most important thing to connect Widget with UI App is,
how to share the data between Widget and UI App
The mechanism for sharing the data is like below



# Stage 5: Connection between Widget & UI

Goal

For this mechanism, two APIs are required
The One is ‘app_control’ to request launch the UI Application
Another is ‘preference’ to write to and read the data from memory area

**app_control**

**This API is used when an application launch the another application**
**It also deliver the data when send request**

**preference**

**This API is used when save the data permanently**
**Data is saved as key-value pair**
**With this API, it is possible to recognize the change of the data**

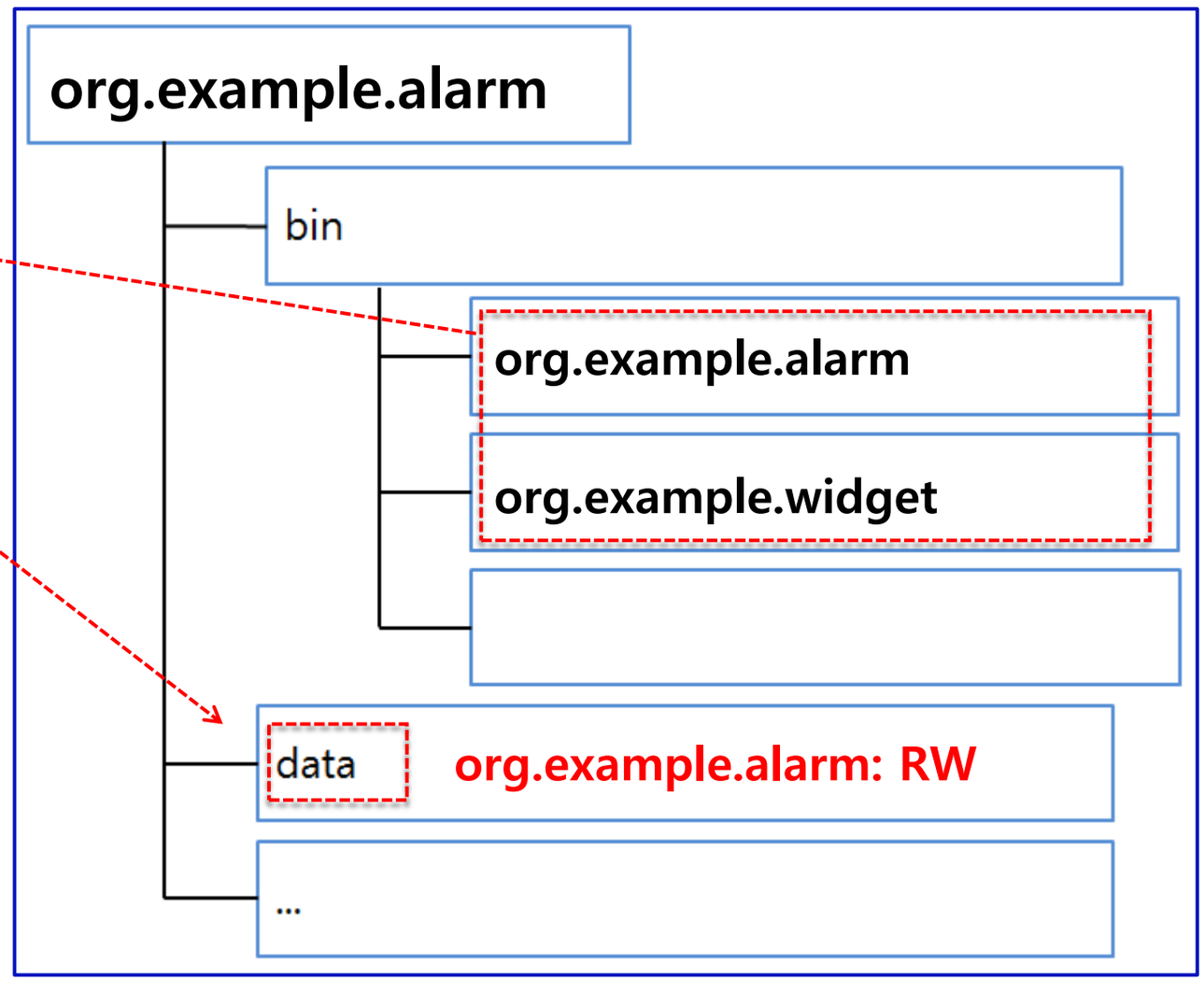
**And One more thing you should do before use these APIs**

# Stage 5: Connection between Widget & UI

Goal

To connect and to share the data,
Widget and UI App must be packaged as one Application
Because, the memory area where the data will be stored by ‘preference’ is located in one App’s data directory

**This data area is allowed to only packaged application like this**



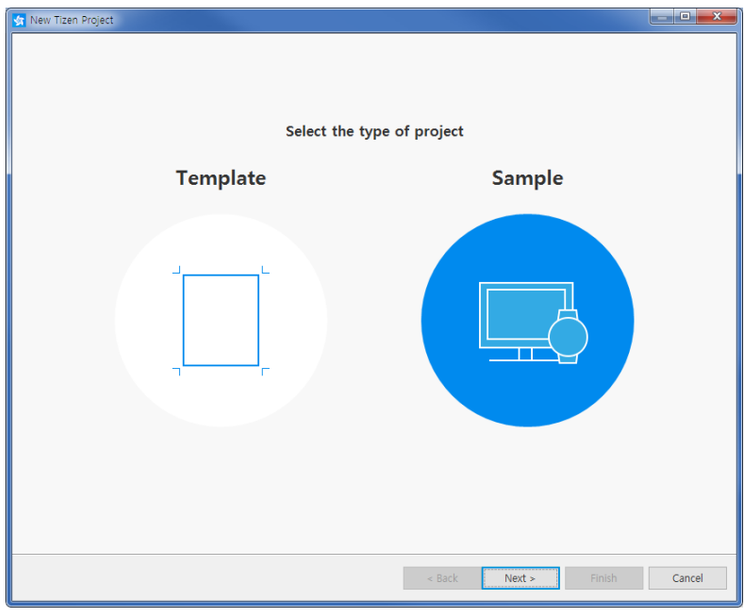

**So, we should package Widget and UI Application**

# Stage 5: Connection between Widget & UI

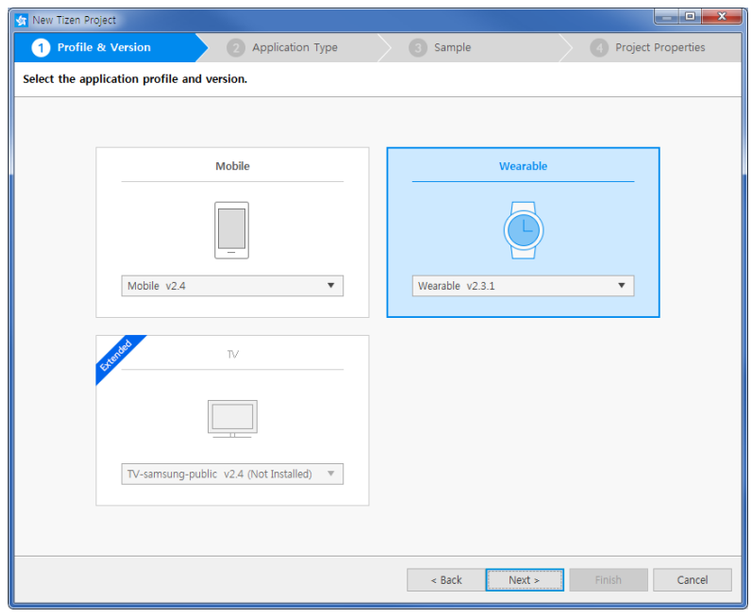
Goal

Import ‘Alarm’ UI Application given as a sample to connect with Widget Application. **File > New > Tizen Project.**

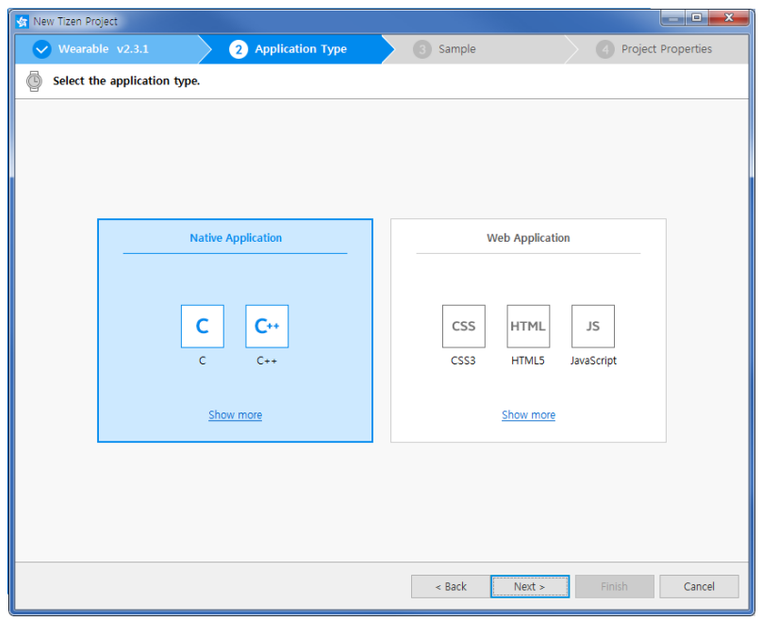
**Sample -> Next**



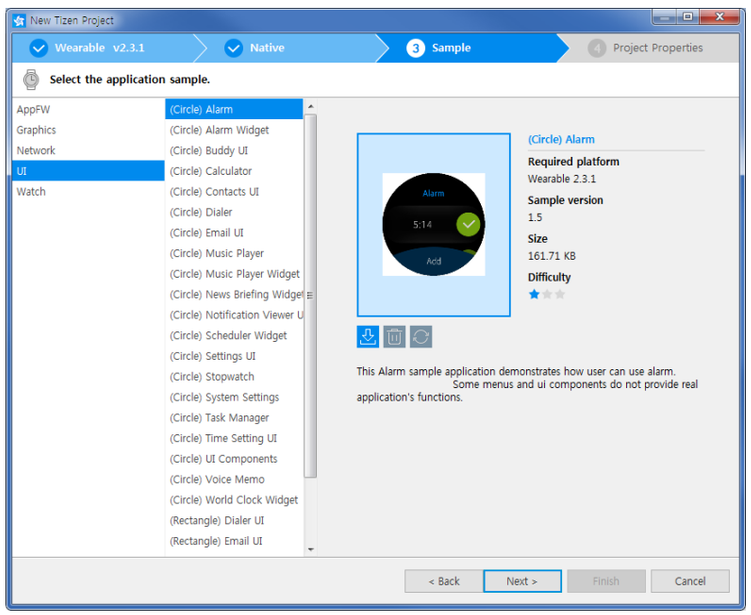

**Wearable v2.3.1 -> Next**



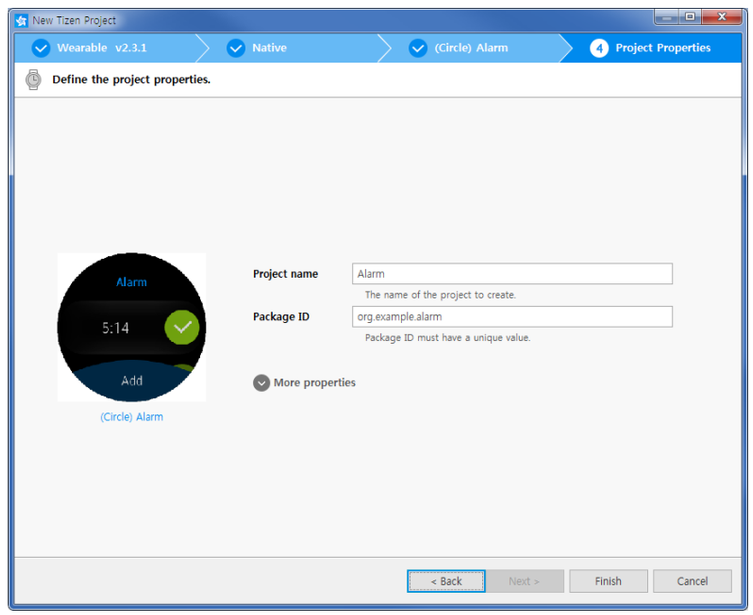

**Native Application -> Next**

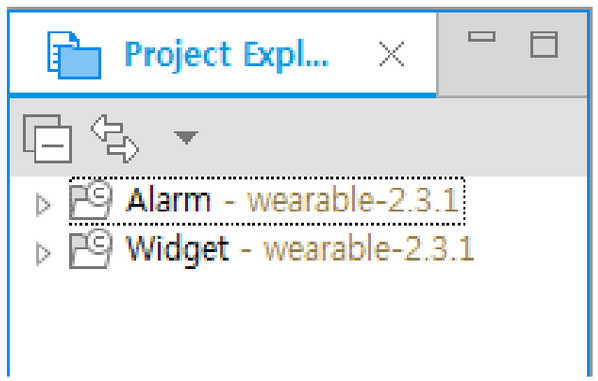



**UI -> (Circle) Alarm -> Next**




**Finish**



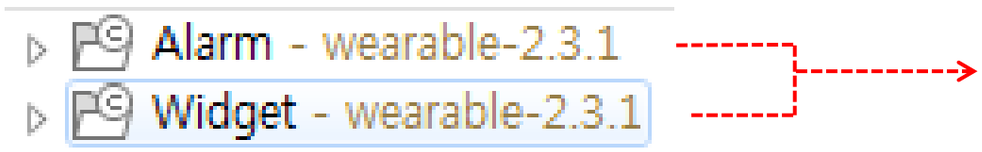

**You can find ‘Alarm’ on the ‘Project Explorer’ Now, let’s package these applications**

# Stage 5: Connection between Widget & UI

Goal

Check two projects in the ‘Project Explorer’
The One is ‘Alarm’ as a UI Application
Another is ‘Widget’ as a Widget Application

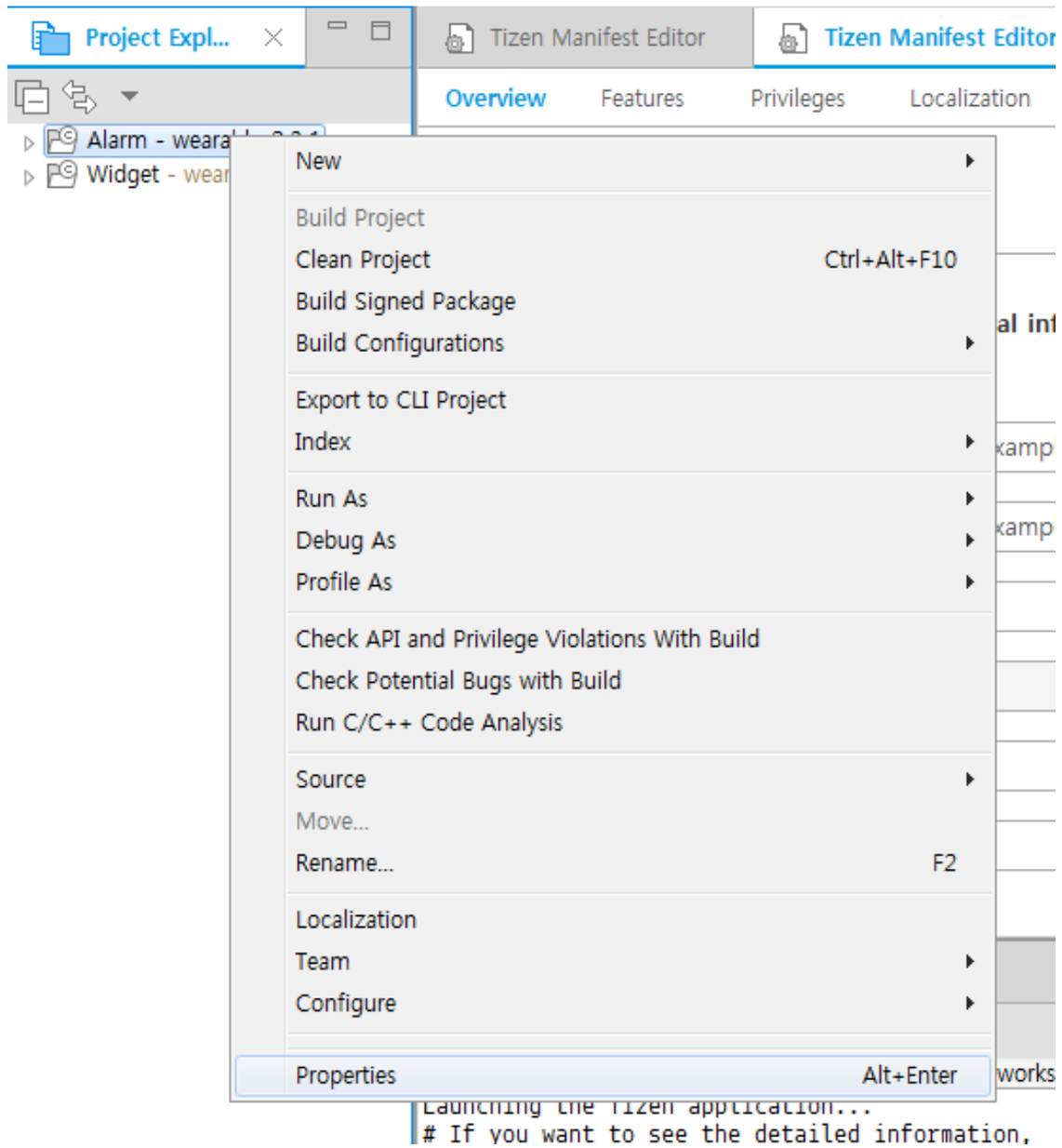
**To share the data using ‘preference’, package these two application**




**Alarm(right click)**

↓

**Properties**



# Stage 5: Connection between Widget & UI

Goal

Choose which Application will  be packaged

**Tizen SDK**

↓

**Package**

↓

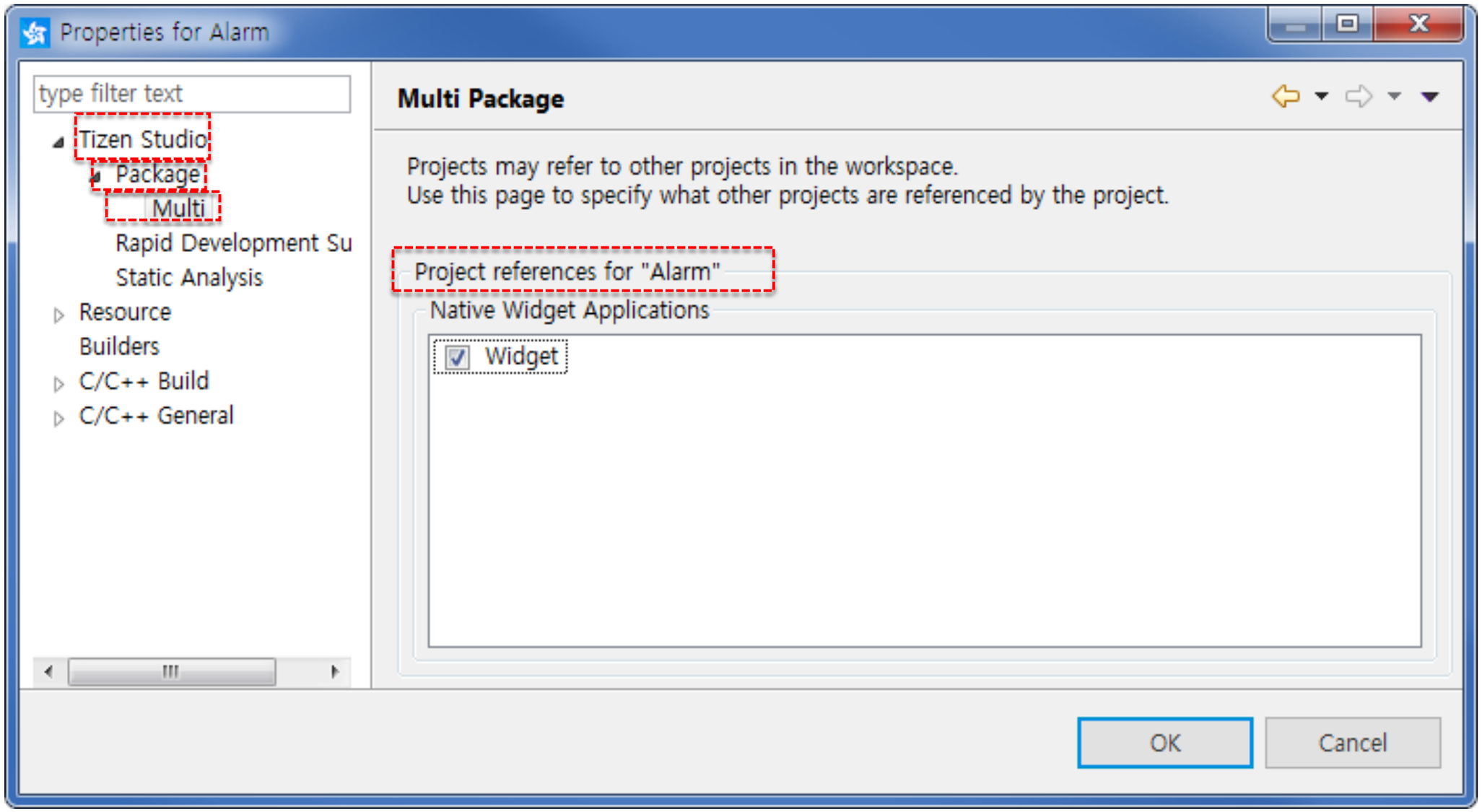
**Multi**

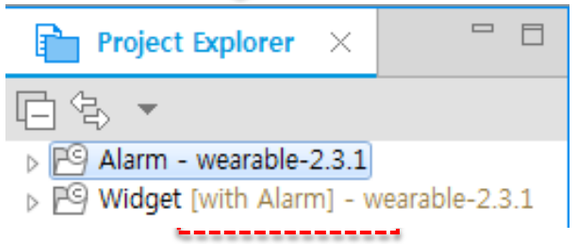
↓

**Check ‘Widget’**

↓

**OK**



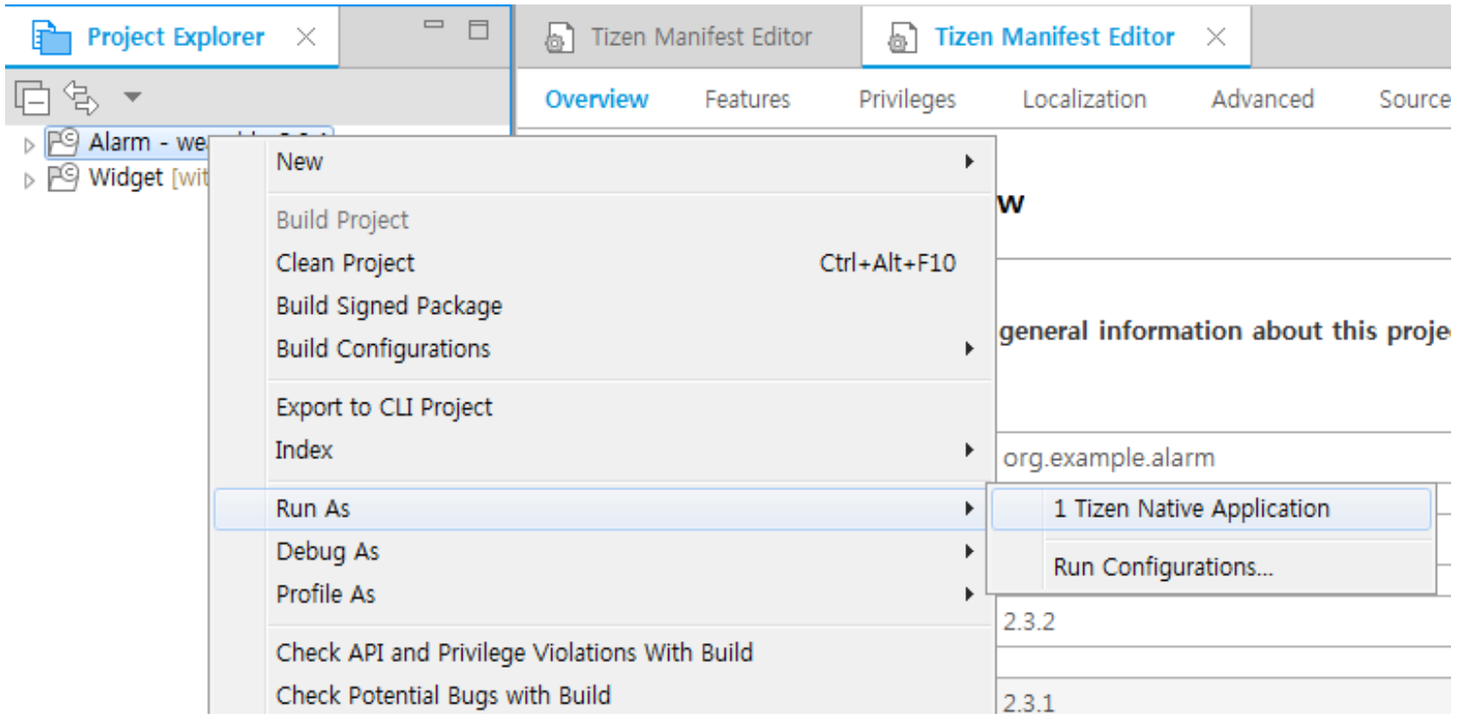




**After finished packaging, when you run the ‘Alarm’, the ‘Widget’ also be run**

# Stage 5: Connection between Widget & UI

Goal

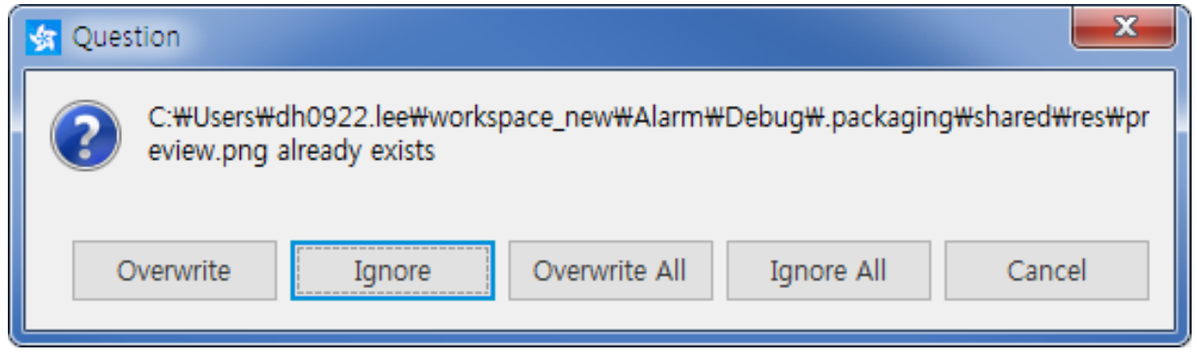
## Run ‘Alarm’ UI Application




**Right click on ‘Alarm’** → **Run As** → **Tizen Native Application**

**You can find this warning**

**Let’s check file directory of two Applications**



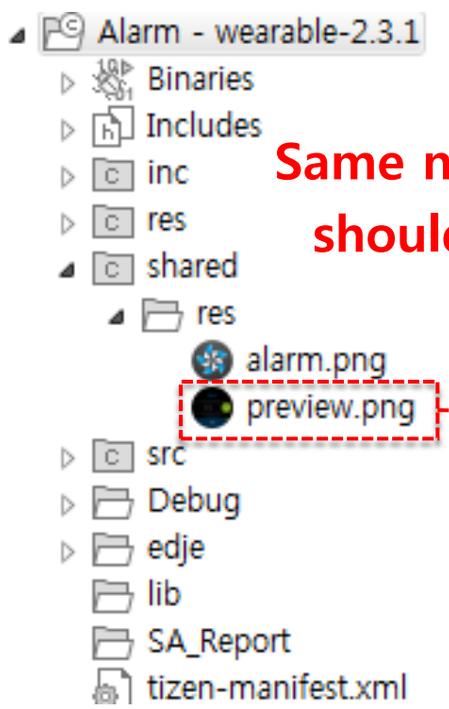

**Because after packaging, these two applications will use same data directory, there should not be same file name**

# Stage 5: Connection between Widget & UI

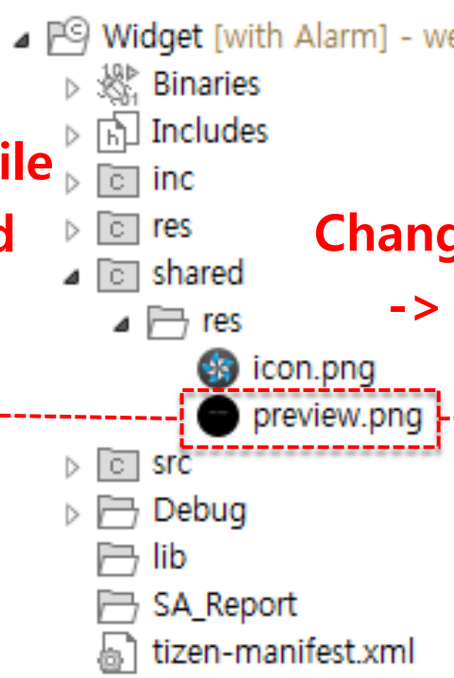
Goal

Find ‘res’ , ‘shared’ folder

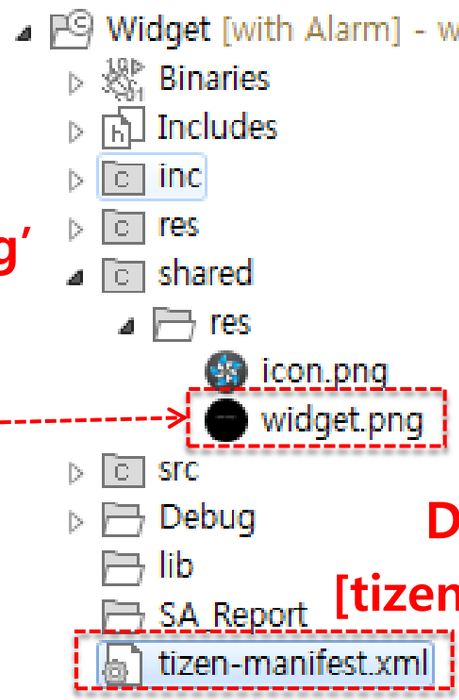
These two folders are the most critical folder



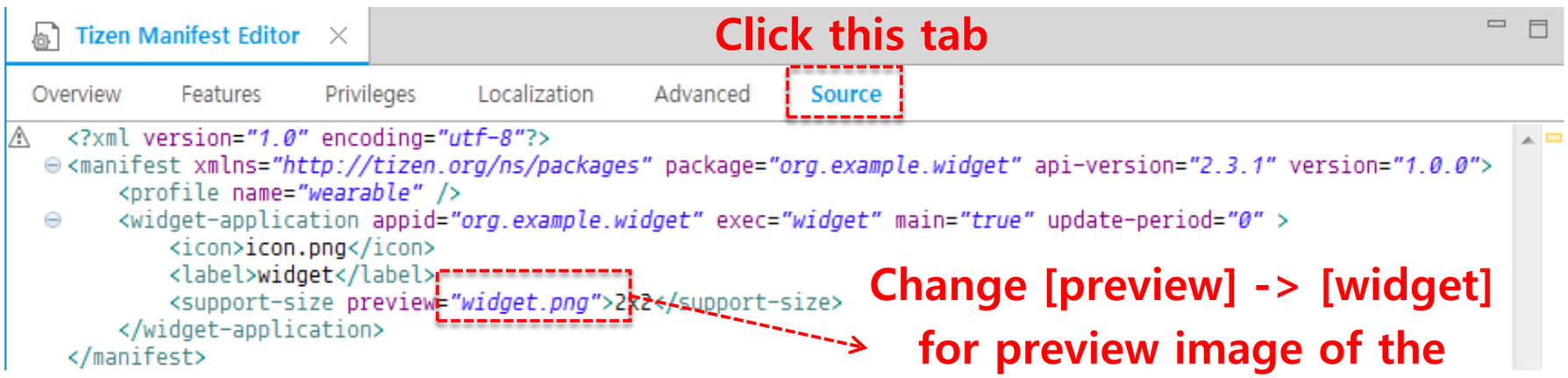

**Same name of the file should be changed**




**Change ‘preview.png’ -> ‘widget.png’**




**Double click [tizen-manifest.xml]**




**Click this tab**

**Change [preview] -> [widget] for preview image of the application**

**[tizen-manifest.xml] is where the properties of the application is listed {appid, pacakge name, Icon, image and size of preview and etc…}**

# Stage 5: Connection between Widget & UI

Goal

Run ‘Alarm’ again

**Find [Alarm] launched**


**Click back key**

**Swipe to the left**


**Click [+] button**

**Find [Widget]’s preview and click**


**Find [Widget] launched**

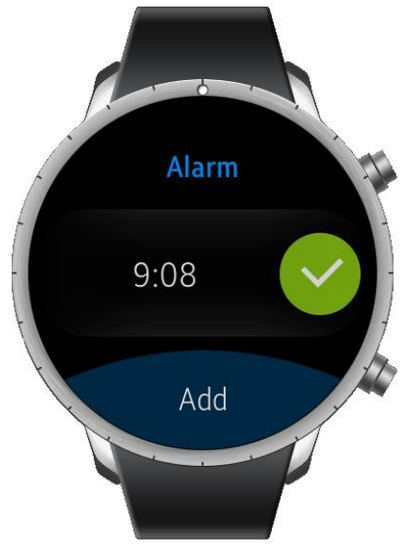

**Let’s check what scenario we will implement**

# Stage 5: Connection between Widget & UI

Goal

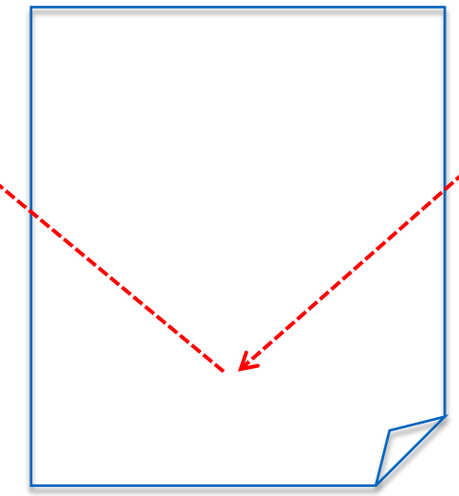
There are three scenario that are available below


**1. Launch ‘Alarm’ to set alarm**




**2. Write the information regarding with alarm to the data**

**3. Read the information regarding with alarm And draw [alarm view]**

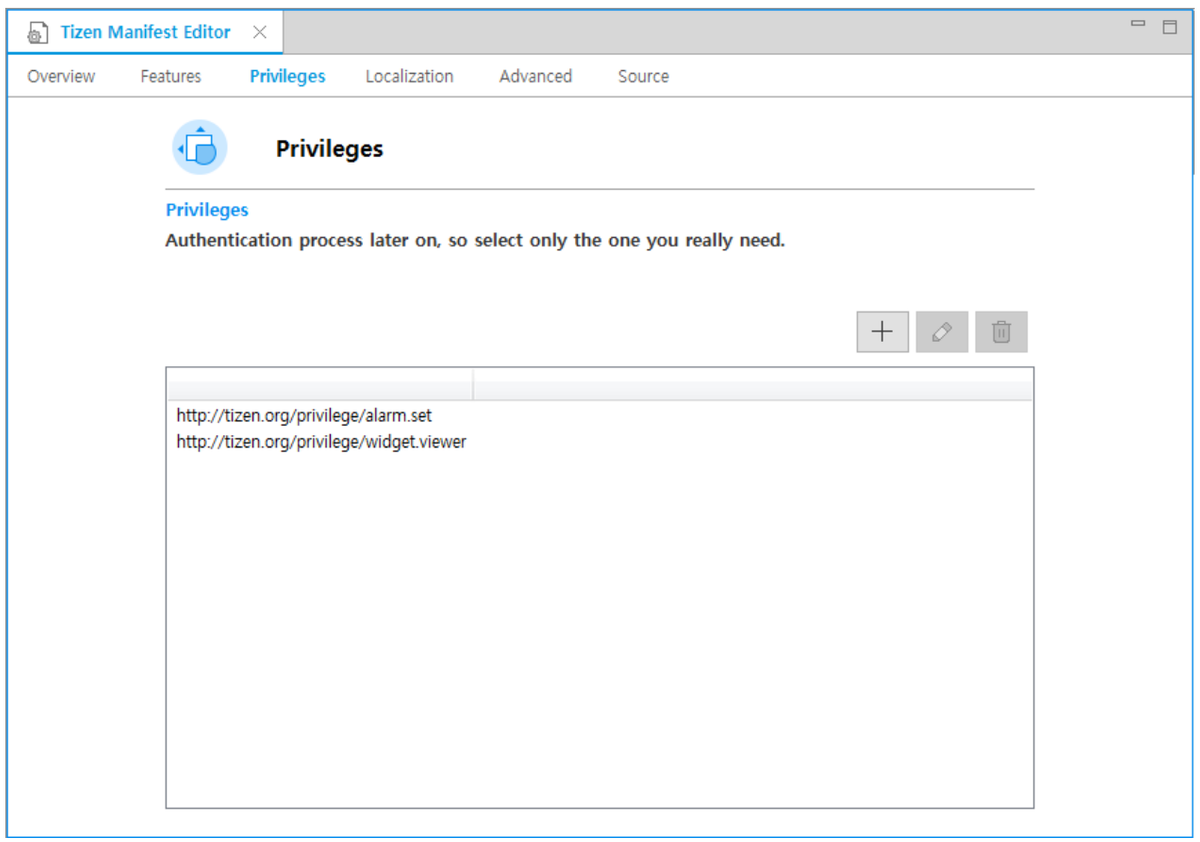


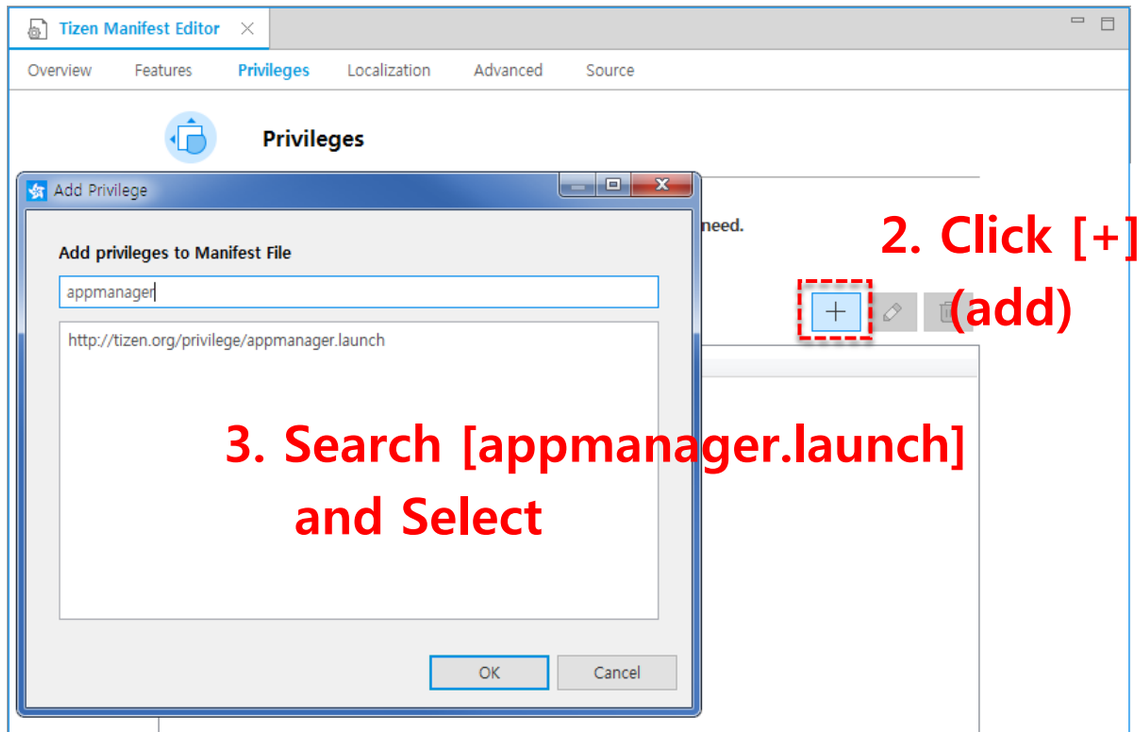
# Stage 5: Connection between Widget & UI

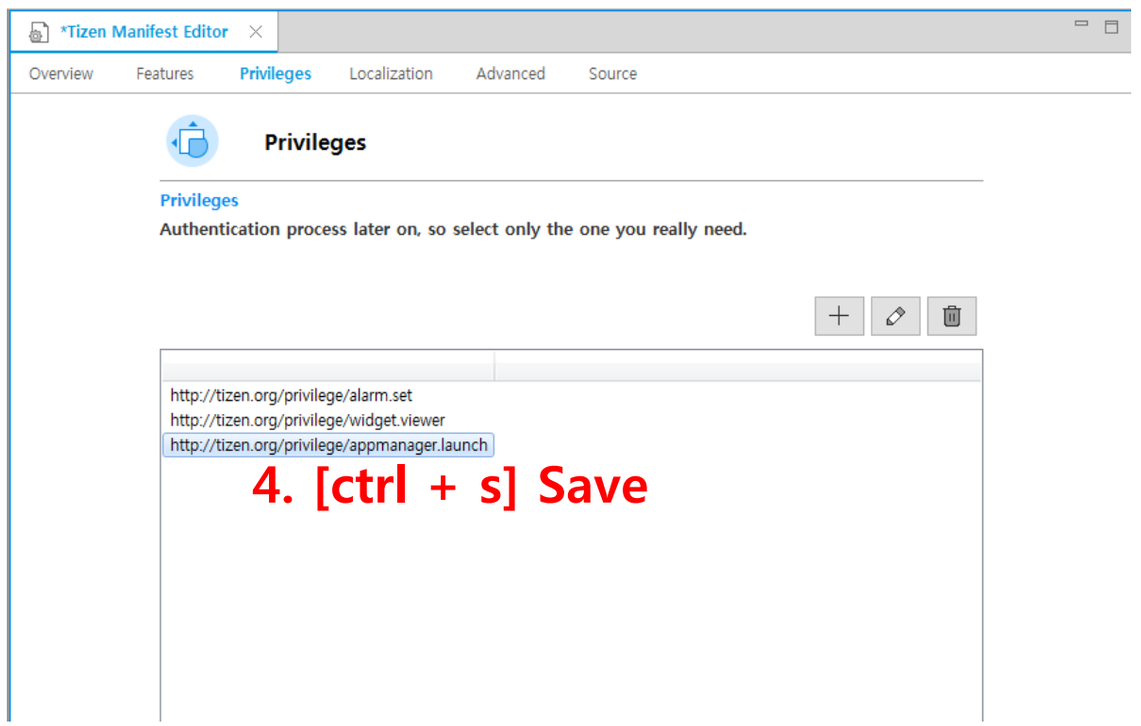
Goal

Launch [ Alarm ] UI Application using [ app_control ]
(Add privilege to use [ app_control ])

**At first, to use [app_control]**
**we need to add privilege**



# Stage 5: Connection between Widget & UI

Goal

Launch ‘Alarm’ UI Application using ‘app_control’
(When you click the [Alarm Image], [Alarm] UI Application should be launched)


**Find ‘_add_alarm_cb’ in the ‘widget_instance_create’ function you already set to the button**

```c
/* Add Alarm Button */
Evas_Object *button = NULL;
button = elm_button_add(wid->conform);
elm_object_style_set(button, "transparent");
evas_object_resize(button, 98, 98);
evas_object_move(button, 180 - 98/2, 180 - 98/2);
evas_object_smart_callback_add(button, "clicked", _add_alarm_cb, context);
evas_object_show(button);
```

**Pass ‘context’ instead of ‘NULL’**

```c
static void _add_alarm_cb(void *data, Evas_Object *obj, void *event_info)
{
    dlog_print(DLOG_INFO, LOG_TAG, "Add alarm is clicked");
}
```

**Add ‘app_control’**

```c
static void _add_alarm_cb(void *data, Evas_Object *obj, void *event_info)
{
    dlog_print(DLOG_INFO, LOG_TAG, "Add alarm is clicked");

    widget_instance_data_s *wid = NULL;
    app_control_h app_control;
    widget_context_h context = (widget_context_h)data;

    app_control_create(&app_control);

    app_control_set_operation(app_control, APP_CONTROL_OPERATION_DEFAULT);

    app_control_set_app_id(app_control, "org.example.alarm");

}
```

**This parameter means using ‘app_control’ to request launch(there are other parameters and ‘app_control’ can send other requests)**

**Set ‘app_id’ of the Application will be launched to the ‘app_control’**

# Stage 5: Connection between Widget & UI

Goal

Launch [Alarm] UI Application using [app_control]

(Set [instance_id] for distinguish the widget instance between multiple instances)

```
    app_control_set_app_id(app_control, "org.example.alarm");

    const char *instance_id = widget_app_get_id(context);

    app_control_add_extra_data(app_control, "widget_instance_id_for_app_control", instance_id);

    app_control_send_launch_request(app_control, NULL, NULL);

    app_control_destroy(app_control);
}
```

**Get [instance_id] to identify certain instance and use this ID as a [key] for saving the data**

**(Next, when we use 'preference' to save & monitoring & read the data, this 'key' is very important)**

**[app_control] also send the data using [key-value] pair**

**To inform [instance_id] to the UI App, save and send [instance_id] using [app_control]**

**[app_control] must be freed after sending the request**

**[Alarm] UI should received this request using [app_control]**

# Stage 5: Connection between Widget & UI

Goal

Launch [Alarm] UI Application using [app_control]
(Open [Aalrm/src/main.c] to check the code regarding launch request from [Widget])

**Find [app_control] function in the [main.c]**

**When the Application will receive the [app_control] signal, the [app_control] function will be operated first**

```
static void app_control(app_control_h app_control, void *user_data)
{
    char *operation = NULL;
    char *alarm_id = NULL;

    app_control_get_operation(app_control, &operation);

} else {

    ret = app_control_get_extra_data(app_control, INSTANCE_ID_FOR_APP_CONTROL, &s_info.instance_id);
    if (ret != APP_CONTROL_ERROR_NONE) {
        dlog_print(DLOG_ERROR, LOG_TAG, "Failed to app_control_get_extra_data(). Can't get extra data.");
        free(operation);
        return;
    }
    dlog_print(DLOG_INFO, LOG_TAG, "instance_id: %s", s_info.instance_id);

    data_initialize_widget_data();
```

**Get the information to check what operation will be operated**
**[Widget] send [APP_CONTROL_OPERATION_DEFAULT] and this mean that launch the Application**

**Get the data saved in [app_control] using [INSTANCE_ID_FOR_APP_CONTROL] key(this is defined as 'widget_instance_id_for_app_control' same with what we used in [Widget])**

# Stage 5: Connection between Widget & UI

Goal

Launch [Alarm] UI Application using [app_control]
(Run and check the operation)

**Find [Alarm] launched**


**Press back key**

**Click [+] button**

**Find [Widget]'s preview and click**


**Find [Widget] launched**


**Click alarm image**

**Find [Alarm UI App] launched**


**Let's Set alarm using [Alarm] UI Application**

# Stage 5: Connection between Widget & UI

Goal

Set alarm to the [Widget] using [preference]

(When alarm UI application set the alarm what  widget should display?)

**Find [Alarm] launched**


**Click alarm image**

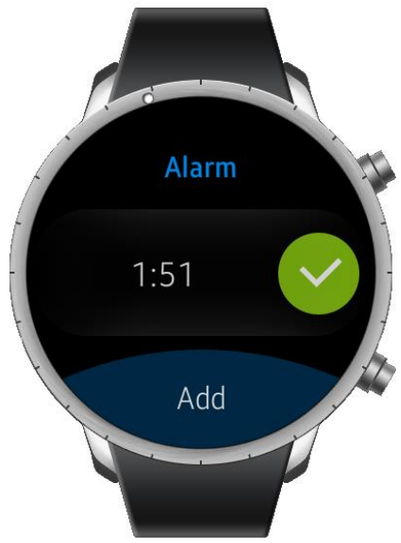
**Click number of [Hrs]**


**Rotate this Point to change the number**

**Click number of [Mins]**


**Click set button**


**Check the alarm**




**How can display this alarm information on the [Widget]?**

# Stage 5: Connection between Widget & UI

Goal

Set alarm to the [Widget] using [preference]
(Check [Alarm] code regarding [preference])

**Find [_alarm_set_time_for_widget] function in [main.c] file**
**When the button filled with clock image is clicked, this function be called**

```
static void _alarm_set_time_for_widget(struct genlist_item_data *gendata)
{
    struct tm *saved_time = NULL;
    char alarm_time[BUF_LEN] = {0, };
    char alarm_id_str[BUF_LEN] = {0, };
    int alarm_id = 0;

    /*
     * Let the widget knows what the time is for alarm.
     */
    saved_time = data_get_saved_time_from_gendata(gendata);
    if (saved_time->tm_hour > 12) {
        snprintf(alarm_time, sizeof(alarm_time), "%02d:%02d PM", saved_time->tm_hour - 12, saved_time->tm_min);
    } else {
        snprintf(alarm_time, sizeof(alarm_time), "%02d:%02d AM", saved_time->tm_hour, saved_time->tm_min);
    }

    alarm_id = data_get_alarm_id_from_gendata(gendata);
    snprintf(alarm_id_str, sizeof(alarm_id_str), "%d", alarm_id);

    data_add_widget_data_bundle_by_str("AlarmTime", alarm_time);
    data_add_widget_data_bundle_by_str("OnOff", "On");
    data_add_widget_data_bundle_by_str("SetAlarm", "Set");
    data_add_widget_data_bundle_by_str("AlarmId", alarm_id_str);
    dlog_print(DLOG_INFO, LOG_TAG, "Alarm Id is : %s", alarm_id_str);

    data_set_widget_alarm_to_preference(s_info.instance_id);
    data_set_instance_id_to_gendata(gendata, s_info.instance_id);

    preference_set_changed_cb(s_info.instance_id, _alarm_on_off_changed_cb, gendata);

    s_info.instance_id = NULL;
}
```

**Get the time information**

**Save the data to bundle as a [key-value] pair**

**Actually, we should know is this function**

**This [instance_id] is from [Widget] using [app_control]**

**Let's go to the [data_set_widget_alarm_to_preference] function**

# Stage 5: Connection between Widget & UI

Goal

Set alarm to the [Widget] using [preference]
(Set the data to the [preference])

```
void data_set_widget_alarm_to_preference(char *instance_id) {
    bundle_raw *r;
    int len;
    int ret = 0;

    if (s_info.widget_data_b == NULL) {
        dlog_print(DLOG_ERROR, LOG_TAG, "widget bundle is NULL");
        return;
    }

    bundle_encode(s_info.widget_data_b, &r, &len);

    ret = preference_set_int((const char *) r, len);
    if (ret != PREFERENCE_ERROR_NONE) {
        dlog_print(DLOG_ERROR, LOG_TAG, "Failed to set len");
    }

    preference_set_string(instance_id, (const char *) r);
    dlog_print(DLOG_INFO, LOG_TAG, "bundle string: %s, len: %d",
            (const char *) r, len);

}
```

**[s_info.widget_data_b] has many information formed [key-value] pair**

**[preference] only can save data that is formed [key-value] pair**

**But data type [bundle] is not formed [key-value] pair**

**So [bundle_encode] change bundle's type to (const char *) to use as a value of [key-value] pair**

**Set bundle has many information as a value to the key named [instance_id]**

**Through this function, any application can use same data directory with [Alarm] application and know the [instance_id], can get the bundle data that is set by [Alarm]**

# Stage 5: Connection between Widget & UI

Goal

Set alarm to the [Widget] using [preference]
(Monitoring data using [preference])

**First, add [app_preference.h] header file to use [preference] on the top of [widget.c] file**

```
#include <tizen.h>
#include <app_preference.h>
```

**To monitoring the key [instance_id]**

```
preference_set_string(instance_id, "Save data to this key");
```
Initialize the key [instance_id]

```
preference_set_changed_cb(instance_id, _alarm_changed_data_with_preference, context);
```

→ **This means that if the value of the key [instance_id] is changed [_alarm_changed_data_with_preference] function will be operated**

**In this function,
read and apply the
information to the [Widget]**

# Stage 5: Connection between Widget & UI

Goal

Set alarm to the [Widget] using [preference]
(Reading the data using [preference])

```
static void _alarm_changed_data_with_preference(const char *key, void *data)
{
    widget_context_h context = (widget_context_h)data;
    widget_instance_data_s *wid = NULL;
    char *bundle_string = NULL;
    char *alarm_time = NULL;
    int len;
    bundle *b = NULL;
    bundle_raw *r = NULL;

    dlog_print(DLOG_INFO, LOG_TAG, "[Alarm] changed the data key [instance_id]");

    widget_app_context_get_tag(context, (void**)&wid);

    preference_get_string(key, &bundle_string);
    preference_get_int(bundle_string, &len);
    r = (bundle_raw *)bundle_string;

    b = bundle_decode(r, len);

    bundle_get_str(b, "AlarmTime", &alarm_time);
    dlog_print(DLOG_INFO, LOG_TAG, "alarm time: %s", alarm_time);
}
```

**Get the string refer to the bundle saved data**

**Get the [len] that is another key to get bundle**

**Get bundle from [r] & [len]**

**Get alarm time from bundle using key [AlarmTime]**

**Check the log to find whether this function is called**

Problems | Tasks | Console | Call Stack | Log

emulator-26101 (w-0227-1)

| Time | Level | Pid | Tid | Tag | Message |
|---|---|---|---|---|---|
| 02-27 15:32:35.050 | Info | 3438 | 3438 | CAPI_WIDGET | widget_app.c: __check_status_for_cgroup(118) > enter foreground group |
| 02-27 15:32:36.000 | Info | 3438 | 3438 | widget | Add alarm is clicked |
| 02-27 15:32:36.030 | Info | 3438 | 3438 | CAPI_WIDGET | widget_app.c: __provider_pause_cb(277) > widget obj was paused |
| 02-27 15:32:36.040 | Info | 3438 | 3438 | CAPI_WIDGET | widget_app.c: __check_status_for_cgroup(128) > enter background group |
| 02-27 15:32:37.970 | Info | 3438 | 3438 | widget | [Alarm] changed the data key [instance_id] |
| 02-27 15:32:37.980 | Info | 3438 | 3438 | widget | alarm time: 03:32 PM |

Tag: widget

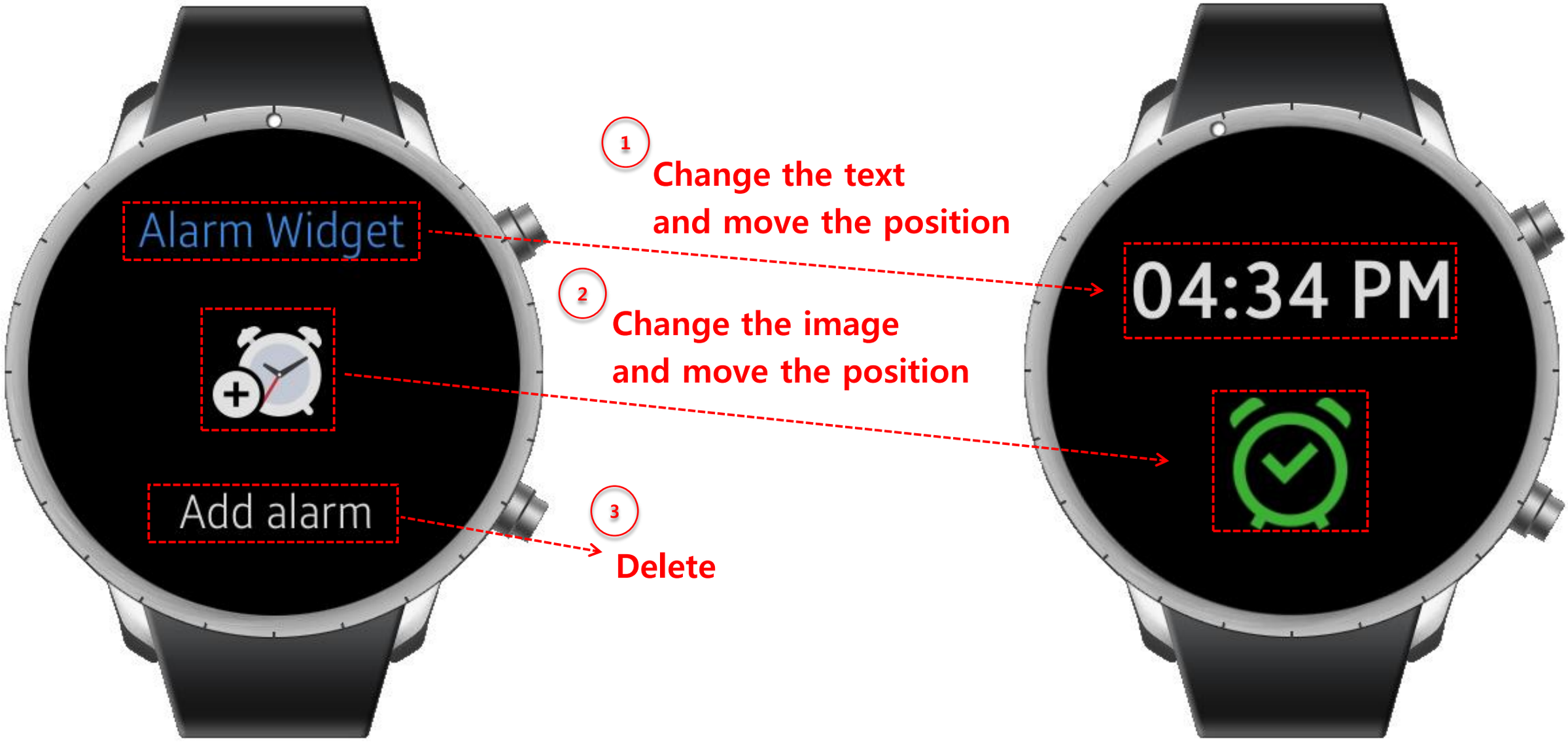
**Let's display alarm time on the [Widget]**

# Stage 5: Connection between Widget & UI

Goal

Set alarm to the [Widget] using [preference]
(Display the alarm time on the [Widget])

**To show the alarm time,**
**change the view like below**



# Stage 5: Connection between Widget & UI

Goal

Set alarm to the [Widget] using [preference]

(You should modify each local variables like button, detail_text and img declared in [widget_instance_create] function

```c
/* Detail Text */
detail_text = NULL;
detail_text = elm_label_add(wid->conform);
evas_object_resize(detail_text, 160, 80);
evas_object_move(detail_text, 110, 260);
evas_object_show(detail_text);
elm_object_text_set(detail_text, "<font_size=35>Add alarm</font_size>");

/* Add Alarm Button */
button = NULL;
button = elm_button_add(wid->conform);
elm_object_style_set(button, "transparent");
evas_object_resize(button, 98, 98);
evas_object_move(button, 180 - 98/2, 180 - 98/2);
evas_object_smart_callback_add(button, "clicked", _add_alarm_cb, context);
evas_object_show(button);

/* Image for Button */
img = NULL;
char *resource_path = NULL;
char image_path[1024];
resource_path = app_get_resource_path();
snprintf(image_path, sizeof(image_path), "%s%s", resource_path, "images/alarm_no_alarm_icon.png");
img = elm_image_add(button);
elm_image_file_set(img, image_path, NULL);
elm_object_content_set(button, img);
```

**Add these variables to the structure variable [widget_instance_data_s]**

```c
typedef struct widget_instance_data {
    Evas_Object *win;
    Evas_Object *conform;
    Evas_Object *label;
    Evas_Object *button;
    Evas_Object *detail_text;
    Evas_Object *img;
} widget_instance_data_s;
```

**This structure [wid] should be set to the [context]**

```c
widget_app_context_set_tag(context, wid);
```

**Get [wid] in the any function using [context]**

```c
widget_app_context_get_tag(context, (void**)&wid);
```

```c
/* Detail Text */
wid->detail_text = NULL;
wid->detail_text = elm_label_add(wid->conform);
evas_object_resize(wid->detail_text, 160, 80);
evas_object_move(wid->detail_text, 110, 260);
evas_object_show(wid->detail_text);
elm_object_text_set(wid->detail_text, "<font_size=35>Add alarm</font_size>");

/* Add Alarm Button */
wid->button = NULL;
wid->button = elm_button_add(wid->conform);
elm_object_style_set(wid->button, "transparent");
evas_object_resize(wid->button, 98, 98);
evas_object_move(wid->button, 180 - 98/2, 180 - 98/2);
evas_object_smart_callback_add(wid->button, "clicked", _add_alarm_cb, context);
evas_object_show(wid->button);

/* Image for Button */
wid->img = NULL;
char *resource_path = NULL;
char image_path[1024];
resource_path = app_get_resource_path();
snprintf(image_path, sizeof(image_path), "%s%s", resource_path, "images/alarm_no_alarm_icon.png");
wid->img = elm_image_add(wid->button);
elm_image_file_set(wid->img, image_path, NULL);
elm_object_content_set(wid->button, wid->img);
```

# Stage 5: Connection between Widget & UI

Goal

Set alarm to the [Widget] using [preference]
(Show the time using [wid->label] variable)

```
/* Change the text */
char time_text[1024];

evas_object_resize(wid->label, 300, 150);


evas_object_move(wid->label, 60, 80);


evas_object_color_set(wid->label, 255, 255, 255, 255);

snprintf(time_text, sizeof(time_text), "<font_size=60 font_weight=bold>%s</font_size>", alarm_time);

elm_object_text_set(wid->label, time_text);
```

**Set text color [white]**


**Apply text style like this way**
**To add text style command to the time text, use [snprintf]**

# Stage 5: Connection between Widget & UI

Goal

Check the Connection between Widget and UI Application

**Find [Alarm] launched**


**Click back key**

**Click [+] button**

**Find [Widget]'s preview and click**


**Find [Widget] launched**


**Click alarm image**

**Find [Alarm] launched**


**Click number of [Hrs]**

# Stage 5: Connection between Widget & UI

Goal

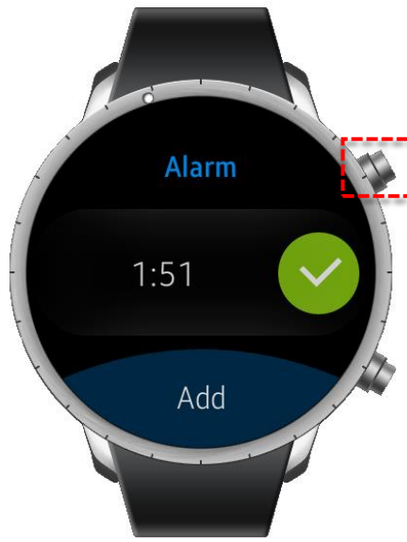
Check the Connection between Widget and UI Application

**Click number of [Mins]**


**Rotate this**
**Point to change**
**the number**

**Click set button**


**Check alarm**




**Click**
**Home button**

**Swipe to the left**


**Find alarm widget**


**If you want to be more familiar with Tizen, visit here**

**https://developer.tizen.org/**