**Tizen Native Application UI 가이드**

**-Tizen::UI namespace**

UI namespace는 GUI를 생성할 때 사용하며 UI와 연관된 이벤트를 다루기 위한 클래스와 인터페이스들을 포함하고 있다. 또한 4개의 Sub-Namespace를 포함하며 기능별로 분류해 두었다.

- Tizen::Ui::Controls : UI객체의 표시와 상호작용을 다룬다.

- Tizen::Ui::Scenes : Scene(화면)의 전환을 다룬다.

- Tizen::Ui::Animations : Animation기능을 다룬다.

- Tizen::Ui::Effects : Effect와 관련된 기능을 다룬다.

**1. Tizen::Ui namespace의 주요기능**

1. **Scene management**

사용자들은 scene management의 기능들을 통해서 scene간 앞, 뒤 그리고 특정한 scene으로 이동할 수 있는 어플리케이션을 개발할 수 있다.

(Scene은 Android의 Activity와 유사한 개념이다.)

전형적인 Tizen의 UI 어플리케이션은 여러 개의 panel을 가지는 form들로 구성된다. 이런 어플리케이션은 form의 life-cycle 그리고 form과 panel사이의 변환에서 발생하는 Overhead를 관리할 의무가 있다.

- Scene management의 기능.

1. Form의 Life-Cycle 관리기능을 향상시키고 Form또는 Panel사이의 Transition(이동)을 쉽게 만들어준다.
2. 특정 Scene으로 넘어가거나, 이전의 Scene으로 이동, 미리 정해진 시나리오대로 Scene의 이동을 돕는다.
3. 다양한 이동 효과를 추가할 수 있다.

- Tizen::UI::Scenes::SceneManager

UI이동을 위한 필수 Method를 제공하는 클래스이다. SceneManager는 UI이동의 기본단위로 Scene을 사용한다. Scene은 Form과 Panel로 화면에 표시된다.(Panel은 Scene에 꼭 포함되지 않아도 된다.)

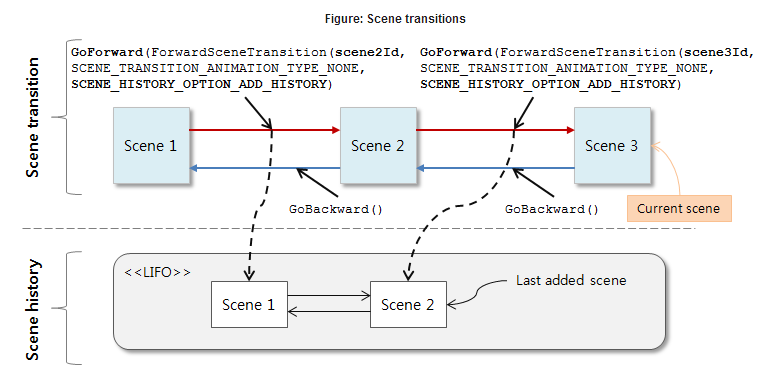
1. 사용자는 어플리케이션의 모든 Scene을 SceneManager클래스의 RegisterScene() 메소드를 통해 SceneManager에 등록해야한다.
2. 적절한 Form과Panel을 생성하기 위해 Tizen::Ui::Scenes::IFormFactory와 Tizen::Ui::Scenes::IPanelFactory클래스를 구현해야한다.

* Scene History와 Life-Cycle

사용자들은 Scene History를 통해서 Scene의 이동을 관리할 수 있고 Scene의 Life-Cycle을 history와 destroy 파라미터를 통해 적절히 변경할 수도 있다.

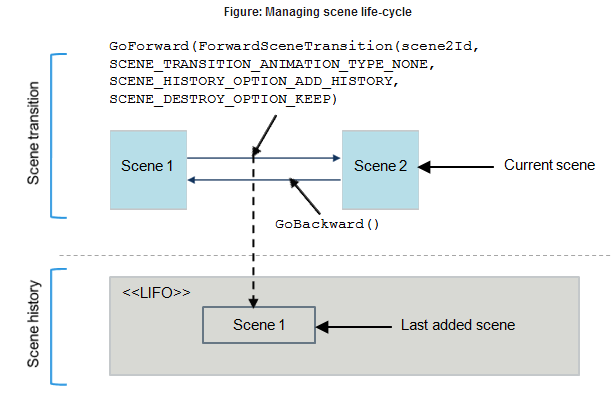
Scene이동의 4가지 경우.

1. 일반적인 Forward와 Backward 이동.

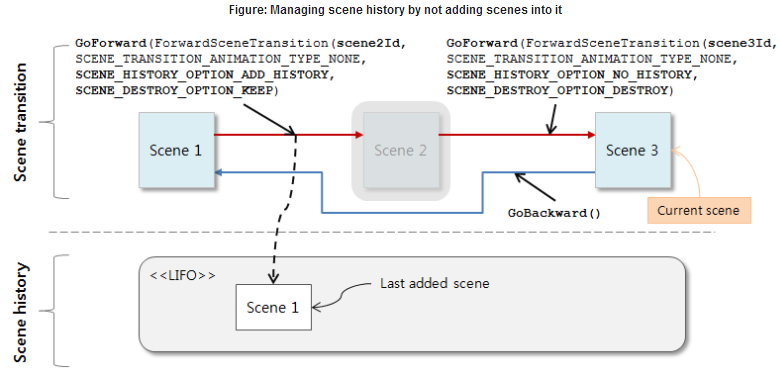


Scene1에서 Scene2로 이동할 때, Scene2에서 Scene3로 이동할 때 GoFoward() 메소드가 불려지며 파라미터를 통해 Scene History에 저장여부를 설정할 수 있다. 저장하게 되면 GoBackward() 메소드가 실행될 때 SceneHistory에서 저장된 Scene을 불러온다.

1. Transition Option을 사용한 Forward Transition.

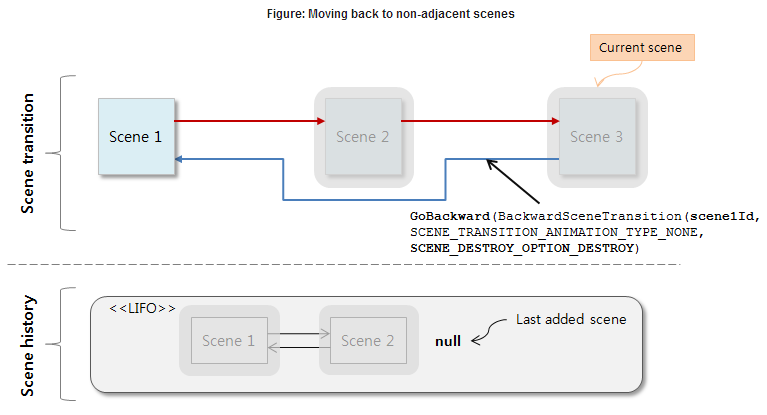


위 메커니즘은 SCENE\_DESTROY\_OPTION\_KEEP이라는 destroy 파라미터 옵션을 추가해서 보낸 경우이다. 이 때 Scene은 history에 추가되지만 Life-Cycle은 그대로 유지하여 Scene을 alive상태로 유지한다. 이렇게 alive상태를 유지하는 Scene은 나중에 다시 호출되었을 때 Creation요청없이 불려져 온다.



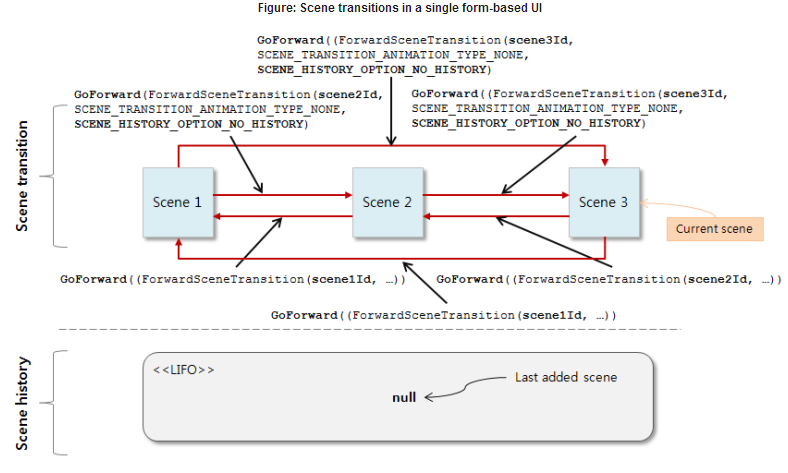
현재의 Scene이 Scene History에 추가되는 것을 원하지 않는다면 SCENE\_HISTORY\_OPTION\_NO\_HISTORY와 SCENE\_DESTROY\_OPTION\_DESTROY 파라미터 옵션을 사용하면 된다. 예를 들어 연락처 어플리케이션을 고려해 보자. 추가 버튼을 눌러서 연락처를 추가하는 화면이 불러져 왔다(Scene2). 이 후 연락처를 추가하게 되면 추가정보를 입력하는 화면이 나타난다(Scene3). 그 다음 사용자들이 하드웨어상의 Back key를 누르면 연락처 추가화면이 아닌 연락처목록(Scene1)이 불러져 온다. 이러한 흐름대로 라면 Scene2를 Scene History에 등록하는 것이 불필요하다.

1. Backward Transition Adjacent or Non-adjacent Scene(인접하였거나 인접하지 않은



Tizen 어플리케이션은 Scene History에서 바로 이전의 Scene과 그렇지 않은 Scene 둘 다로 이동할 수 있다. GoBackward() 메소드에 특정 Scene의 ID를 넣어주면 그 Scene으로 이동하고 그렇지 않으면 이전 단계에 저장되었던 Scene으로 이동한다. 특정 Scene으로 이동하는 경우 중간 단계에 있던 Scene들은 자동으로 뛰어 넘게되고 소멸된다(위 그림의 Scene2는 Skip & Destroy 된다).

1. Tab 스타일의 Panel Transition



Tab-style의 panel은 이전 Scene에 영향을 받지 않고 사용자의 행동에 따라 동작한다. 때문에 Scene은 이전의 Scene으로 돌아갈 필요가 없다. 따라서 Forward Transition을 할 때 SCENE\_HISTORY\_OPTION\_NO\_HISTORY라는 history 파라미터 옵션을 주어 Scene을 저장하지 않는 것이 효율적이다. 위 그림을 보면 GoForward() 메소드만 사용을 하며 특정 Scene의 ID를 파라미터로 넘겨주어 이동하게 된다.

1. **UI control management.**

사용자들은 UI controls(어플리케이션 Frame안에 표시되는 GUI객체)를 보고 상호작용 할 수 있다. UI Controls는 2가지로 분류가 된다.

1. Containers.

Container는 Controls와 다른 Containers를 포함할 수 있는 Control이다. 레이아웃과 화면 표시요소를 다루기 위한 객체이다. Containers는 그 안에 포함된 Controls와 부모-자식 관계를 이룬다.

Tizen::Ui::Container 클래스는 UI Controls를 구성하고 어플리케이션의 레이아웃을 설정하기 위해 사용된 UI객체에 해당한다. Container에 추가된 자식 Controls들은 그들이 쌓인 Stack 순서로 관리된다. Container는 또한 custom controls의 구현을 위해 사용된다.

* Life-Cycle

Container 객체는 파괴될 때 자동으로 그 자식 객체들을 제거한다.

따라서 사용자들은 Container가 파괴될 때 자식 객체들을 하나하나 삭제 시킬 필요가 없다. 하지만 자식객체를 Container에서 떼어내지 않고 삭제한다면 Container가 파괴될 때 자식객체의 삭제를 시도하고 ‘unhandled exception’을 유발한다.

* Container의 분류

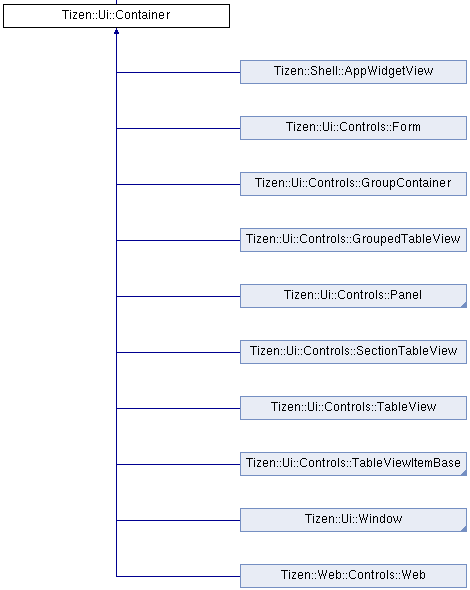
1. General Containers

Tizen::Ui::Controls::Frame, Tizen::Ui::Controls::Form, Tizen::Ui::Controls::Panel, Tizen::Ui::Controls::Popup.

1. Special Purpose Containers

특수 목적 Containers는 Tizen::Ui::Window 클래스로부터 상속받아 만들어 진다. Windows는 그들만의 Drawing frame buffer를 가지는 Container 클래스의 확장된 클래스이다. 따라서 Windows는 Form없이도 사용할 수 있다. Window를 상속받아 만들어진 Controls들은 특별한 목적을 가지고 설계되며 general Container의 역할을 할 수 없다.

* Container의 종류



1. Controls.

Controls는 UI 객체 그 자체이다. 예를 들면 List나 TextField 같은 것들 것 있다. 그리고 이것들은 Containers에 add될 수 있다. Controls는 사용자에게 상호작용을 제공하며 다른 Controls를 포함 할 수는 없다.

Tizen::Ui::Control 클래스는 모든 UI객체의 추상클래스이며 이 클래스는 windowing system아래 화면을 그리는 기능, 사용자의 input에 반응하기 위한 요소를 포함하고 있다. 또한 어떻게 input events를 받고 처리하는가를 결정하기도 한다.

* Control 공통 속성

1. Position & Size
2. Drawing logic
3. Events (Key, Focus, Touch, Gesture, Drag and Drop)

Events를 다루기 위해선 Listener를 정의해주어야 한다.

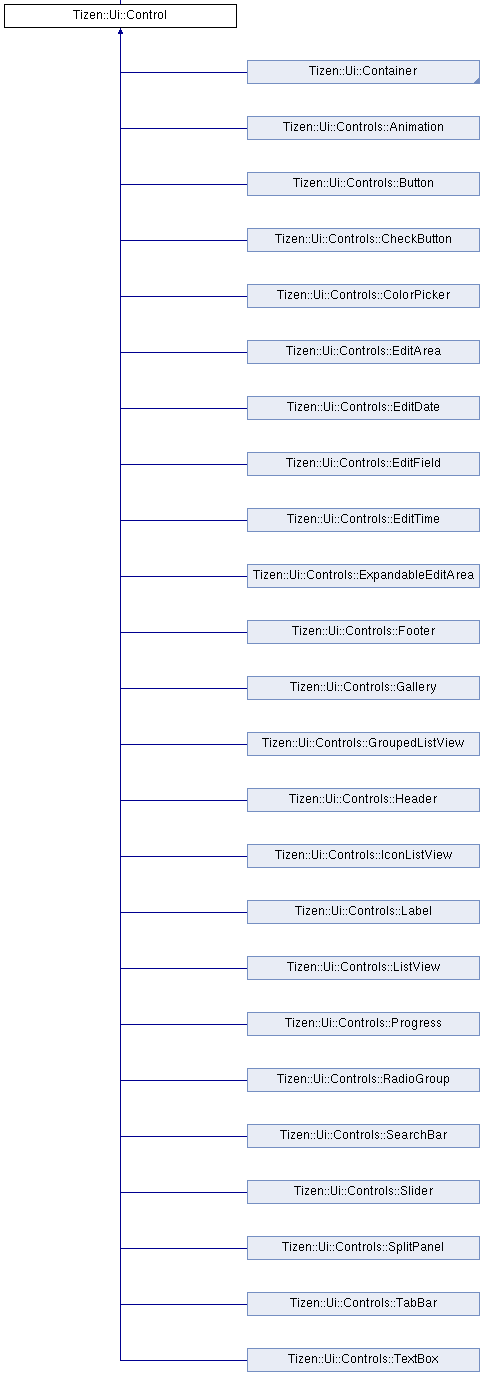
* Tizen에서의 UI Controls

UI Controls의 범위는 기본버튼에서 애니메이션 Controls같은 Advanced Controls까지 광범위하다. 모든 Controls들은 UI로 보여질 수 있다. Tizen에서는 볼 수 없는 Database, Data binding, Adapter Controls와 같은 것들을 UI Controls로 지원하지 않는다.

* UI Builder

Tizen의 UI Builder는 WYSIWYG(What You See Is What You Get) 형태의 Design Tool이다. 이 Tool은 거의 모든 UI Controls를 제공한다.

* Controls의 종류



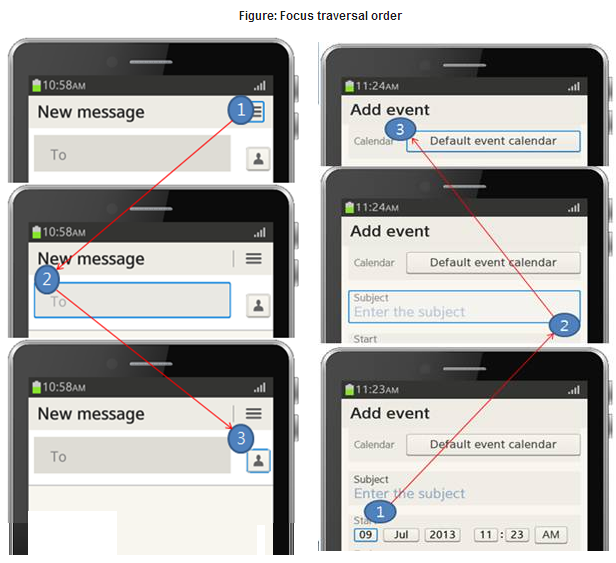
1. Focus UI

Focus UI는 Hardware 키보드나 블루투스 키보드를 사용해서 터치 행동과 유사하게 Controls들을 Controll할 수 있게 한다.

* Focus 이동 방법.

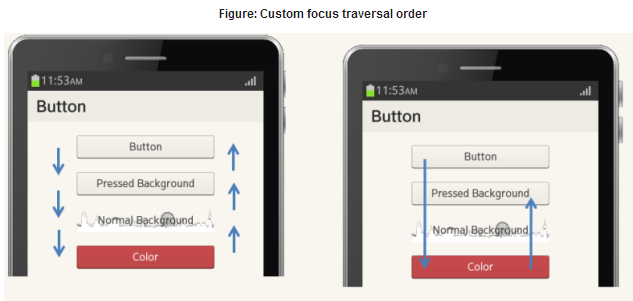
Tizen에서는 2가지 이동방법을 제공한다. Tab, Shift+Tab, Up, Down키를 통해 이전의 Control과 다음 Control을 돌아다닐 수 있다. 만약 Control이 내부적으로 item을 가지고 있다면(ListView, TableView) Left, Right키를 이용해 선택할 수 있다.

* Default Focus 이동경로

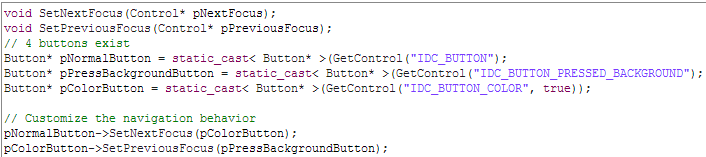


왼쪽 상단에서 오른쪽 하단을 향해 순서대로 전개된다.

* Custom Focus 이동경로



왼쪽과 같이 위에서 아래 아래서 위로 한 단계씩 이동하는 것이 Default로 설정 되어있고 Control 클래스의 메소드를 통해 이를 오른쪽과 같이 재정의 할 수 있다.



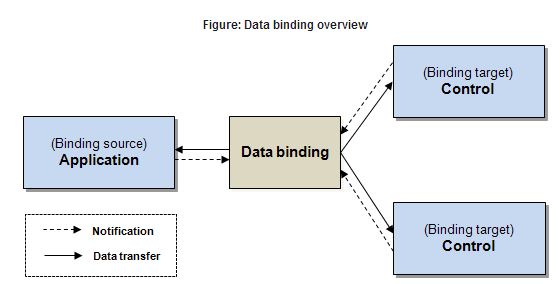
재정의 코드이다.

* Context Menu의 Focus UI

Context Menu는 Default로 Focus UI가 설정되어있지 않다. 따라서 SetFocusable(true) 메소드를 통해 설정을 해주어야 한다. 만약 Context Menu가 Focus를 가진다면 가려질 Window를 위해 OnDeactivated 이벤트 핸들러가 호출되므로 이를 고려해서 적용을 시켜야 한다.

1. Data Binding

Tizen은 UI Control의 속성들과 안전하고 간편하게 상호작용할 수 있는 어플리케이션을 위해 데이터 바인딩을 제공한다. 데이터 바인딩은 어플리케이션의 데이터를 UI Controls에 전달하거나 UI Controls의 속성을 어플리케이션에 설정하기 위해 Runtime(실행시간)에 사용할 수 있다. Data가 이동될 때 자동적으로 알맞게 변형된다.

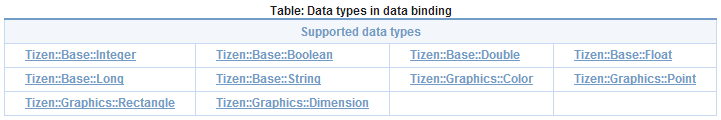


Tizen 개발에 있어서 어플리케이션 데이터와 UI Control 속성간에 데이터를 연결하는 가장 편한 방법은 UI Builder를 사용하는 것이다. 그러나 코드로 Data Binding을 구현하고 싶다면 Tizen::Ui::DataBindingContext 클래스를 사용하면 된다. 이 클래스로 사용자 정의 DataBinding을 구현할 수 도 있다.

* Binding 구성 요소

1. 특정한 Target Object(UI Control)
2. Binding Target(a predefined UI Control property)
3. Binding source 변수

* Binding에 지원되는 Data Type



* Data Flow Direction과 Triggers

Data가 Binding되자마자 data는 Binding Type에따라 다른 방향으로 전달 될 수 있다.

1. One-Way

Data가 Binding Source에서 Binding Target으로 전달된다. Binding Source에서 생기는 모든 변화는 Binding Target에 반영된다. 하지만 그 반대는 실행되지 않는다.

1. Two-Way

Data가 Binding Source와 Binding Target 양쪽으로 이동한다.

1. One-Way-To-Source

Data가 Binding Target에서 Binding Source로 거꾸로 전달된다.

* Trigger

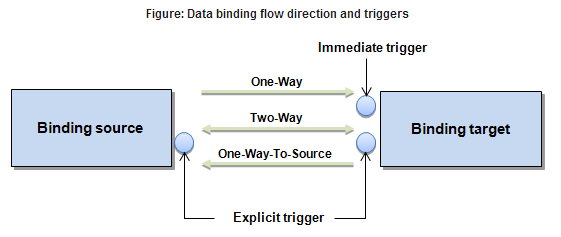
전달받은 Data를 바탕으로 어플리케이션 Data나 UI Control을 갱신하기 위해 Binding Target과 Binding Source 양쪽에 Trigger가 필요하다.

1. Immediate Trigger

이 Trigger는 Binding Target이 Source의 값이 변경되자마자 갱신되도록 한다. Binding Target만 사용할 수 있다.

1. Explicit Trigger

이 Trigger는 명시적인 요청이 있을 때만 Binding Target이나 Binding Source가 갱신되도록 한다. Binding Target/Source모두 사용 가능하다.



1. Event Handling.

사용자의 행동을 다루기 위해서는 UI event listener를 구현하고 그것을 Controls에 등록해야 한다. Tizen::Ui namespace에 정의된 UI event들은 모든 Controls에 공통이며 각 UI Control들은 자신의 필요에 따라 특별한 UI events를 정의한다.

* Tizen::Ui::Control, Tizen::Ui::Container, Tizen::Ui::Window 클래스에 정의된 UI event lists

1. Key events
2. Touch events
3. Drag and Drop events
4. Focus events
5. Touch mode change events
6. Window events

* Gesture Events

Gesture는 touch같은 input event와 유사하다. 하지만 gesture는 UI Control에 추가된 gesture detector로 다루어진다. Gesture events는 Tizen::Ui::TouchGestureDetector클래스로부터 상속받아 정의된다.

* Tizen Ui가 제공하는 gesture

1. Tap
2. Long press
3. Pan
4. Flick
5. Pinch
6. Rotation
7. Custom

* Propagating Events

이벤트의 전달을 위해서 Controls들은 부모-자식 관계를 반드시 추적해야 한다. 예를들어 2개의 Panel이 부분적으로 겹쳐져 있다면 Event는 아래의 Panel에 전파되지 못한다. 따라서 Overlap되는 Panel은 부모-자식관계를 가져야 하며 위쪽의 Panel은 반드시 visual element로 만들어 져야 된다는 것을 뜻한다. Visual Element로 만들어지면 touch나 key event들은 위쪽 Panel을 지나 전달될 것이다.

1. Layout.

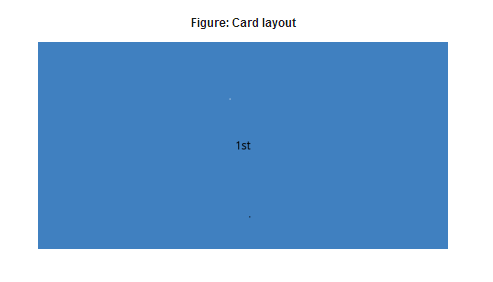
Tizen platform에서는 Container에 Layout을 추가하는 것이 가능하다.

기본적으로 자식 Control의 위치는 부모 Container의 왼쪽 상단을 기준으로 정해지지만 어떤 Layout을 추가하는가에 따라 기준점이 변경 될 수 있다.

* Tizen에서 제공하는 Layout

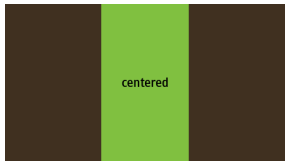
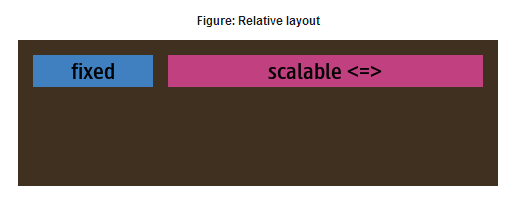
1. Card Layout

Card Layout은 각각의 Control을 부모의 Container Size에 맞추기 위해 사용한다.



1. Relative Layout

Relative Layout은 Container에 추가된 각각의 Control들이 서로서로 영향을 미치게 한다. 이 Layout이 사용되면 Controls간의 관계를 할당해 주어야 한다.



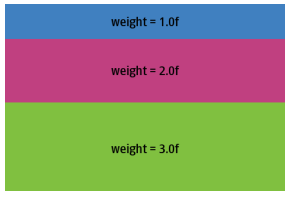
1. Horizontal Box Layout & Vertical Box Layout

Control들이 수평 또는 수직으로 정렬된다. 두 Layout에는 다음과 같은 속성들이 있다.

1. Weight

Layout에서 Control이 차지하고 남는 여분의 공간을 설정한다.

Default Weight값은 0.0이다.



1. Alignment

Layout에서 어떻게 Control을 정렬할 것인지 설정한다.

Vertical Layout에서는 left side, center, right side의 세가지 속성이 있고 Horizontal Layout에서는 top side, center, bottom side세가지 속성이 있다.

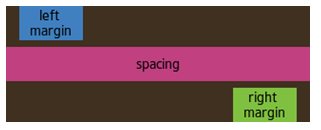


1. Spacing

현재 추가된 Control과 이전에 존재하던 Control사이의 공간을 설정한다. Default 값은 0이다.

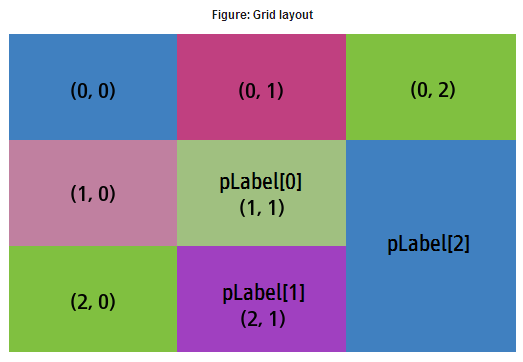
1. Margin

Control을 속성값을 바탕으로 정렬하고 적용하기 전에 Margin 속성으로 남겨둘 공간을 설정한다. Vertical Box Layout이라면 Left와 Right side, Horizontal Box Layout이라면 Top과 Bottom side의 Margin값을 설정 할 수 있다.



1. Grid Layout

Grid Layout은 각각의 Control들이 2차원 Grid에 각각 위치하는 Layout이다. 각 칸은 위치된 Control의 높이와 너비를 기준으로 크기가 정해진다.



Grid Layout에는 다음과 같은 속성들이 있다.

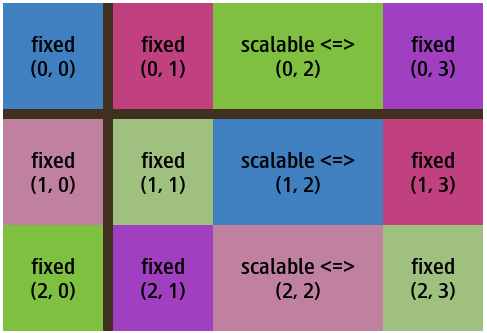
1. Stretch, Shrink, Collapse

행이나 열의 크기가 추가되는 Controls의 크기보다 크거나 작다면 Controls를 그에 맞춰 줄이거나 확장시킨다.

Default는 false로 되어있으므로 true로 활성화 시킨 후 사용할 수 있다.

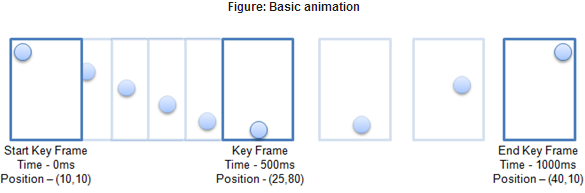
1. Alignment, Spacing, Margin

앞서 언급한 Vertical/Horizontal Box Layout과 같은 속성.



1. **Animation**

타이젠에서 Basic 애니메이션 Start value, Key value, End value 3가지 요소로 구성되어 있다. Key value는 interpolators에 의해 자동으로 계산되어지는 값이다.



Tizen::Ui::Animations namespace는 풍부한 UI개발을 위해 애니메이션과 연관된 많은 기능들을 제공한다. 이 Namespace는 두 개의 카테고리로 나눠진다.

* Animating UI controls

1. Animating UI controls

UI controls들의 속성을 애니메이션으로 보여주는 것을 말한다.

Start value와 End value에 같은 데이터 타입을 넣어주고 Interpolator를 선택하면 Tizen 플랫폼이 애니메이션을 만들어 준다.

데이터 타입에는

Tizen::Ui::Animations::IntegerAnimation,

Tizen::Ui::Animations::FloatAnimation,

Tizen::Ui::Animations::PointAnimation,

Tizen::Ui::Animations::DimensionAnimation,

Tizen::Ui::Animations::RectangleAnimation,

Tizen::Ui::Animations::RotateAnimation까지 총 6개가 제공된다.

Interpolator는 어떤 애니메이션을 만들지 선택할 수 있게 해준다.

Tizen에서는 6가지 Interpolator를 제공한다.

ANIMATION\_INTERPOLATOR\_LINEAR

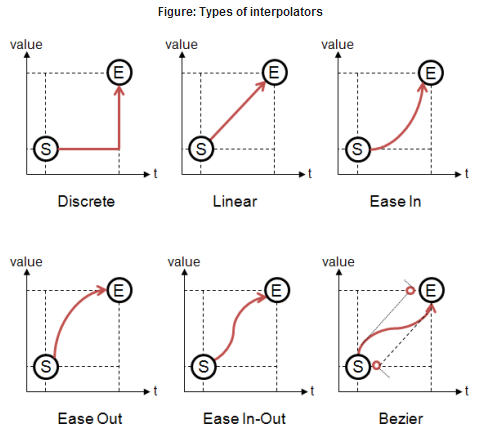
ANIMATION\_INTERPOLATOR\_DISCRETE

ANIMATION\_INTERPOLATOR\_EASE\_IN

ANIMATION\_INTERPOLATOR\_EASE\_OUT

ANIMATION\_INTERPOLATOR\_EASE\_IN\_OUT

ANIMATION\_INTERPOLATOR\_BEZIER



각 Interpolator들이 중간 값들을 계산하는 방법을 그림으로 나타낸 것이다.

아래 예제는 Start value, End value, Duration, Interpolator를 사용해서 애니메이션을 만들고 사용하는 예제이다.

1. Tizen::Ui::Animations::FloatAnimation클래스의 객체를 초기화.



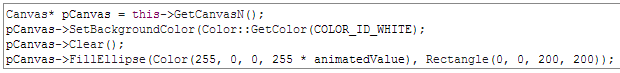
Start value : 0.0f, End value : 1.0f, Duration : 1000ms,

Interpolator : ANIMATION\_INTERPOLATOR\_LINEAR.

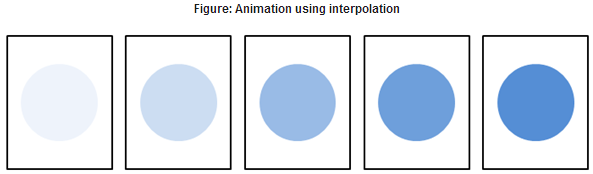
1. GetAnimatedValue() 메소드를 통해 500ms 후의 animated value를 가져온다.



1. 받아온 animated value를 alpha값으로 사용하는 ellipse를 그려준다.



1. 최종 결과



또한 애니메이션들은 UI Control의 다양한 타입의 속성에 적용할

수 있으며 미리 정의된 Animation Target으로 선택할 수 있다.

ANIMATION\_TARGET\_POSITION,

ANIMATION\_TARGET\_SIZE,

ANIMATION\_TARGET\_ALPHA,

ANIMATION\_TARGET\_ROTATION.

이렇게 만들어진 애니메이션들을 Tizen::Ui::Animations::AnimationGroup 클래스로 묶어 하나의 애니메이션처럼 사용할 수 있다.

AnimationGroup에는 ParallelAnimationGroup과 SequentialAnimationGroup이 있다. ParallelAnimationGroup은 AnimationGroup에 추가된 Animation들을 동시에 Animating하는 것이고 SequentialAnimationGroup은 추가한 Animation들에 순서를 줘서 Animating할 수 있게 하는 것이다.

아래 예제는 Animation Group을 사용하는 예제이다.

1. Tizen::Ui::Animations::FloatAnimation과 Tizen::Ui::Animations::DimensionAnimation클래스의 객체를 만들고 초기화 한다.



1. 앞서 만들었던 두 개의 Animation을 Group으로 만들기 위해 ParrellelAnimationGroup객체를 만든다.



1. 각각의 Animation을 Alpha와 Size에 적용 시킨다.



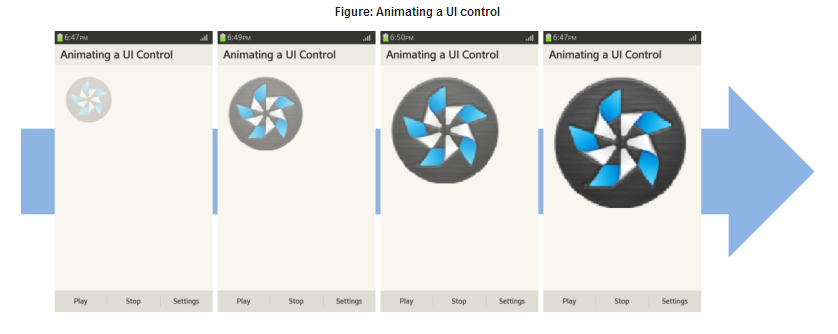
1. control객체의 show state의 변화를 감지하기 위한 트리거를 설치해 준다.



1. Animation을 화면에 그려준다.



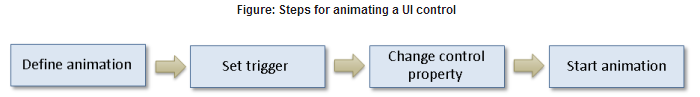
1. 최종 결과



애니메이션에서 Control의 size를 변화시키면 Control안에 포함된 내용들(text, image)은 같이 변화한다.

또한 부모의 객체에 애니메이션이 적용되어 있으면 자식 객체도 똑같이 애니메이션이 적용된다. 예외적으로 Size속성에 적용되어 있다면 자식객체에는 적용되지 않는다. 추가로 부모 Controls에 있는 ControlAnimator::StartUserAnimation()메소드는 자식 Control에 있는 같은 메소드를 포함하지 않는다.

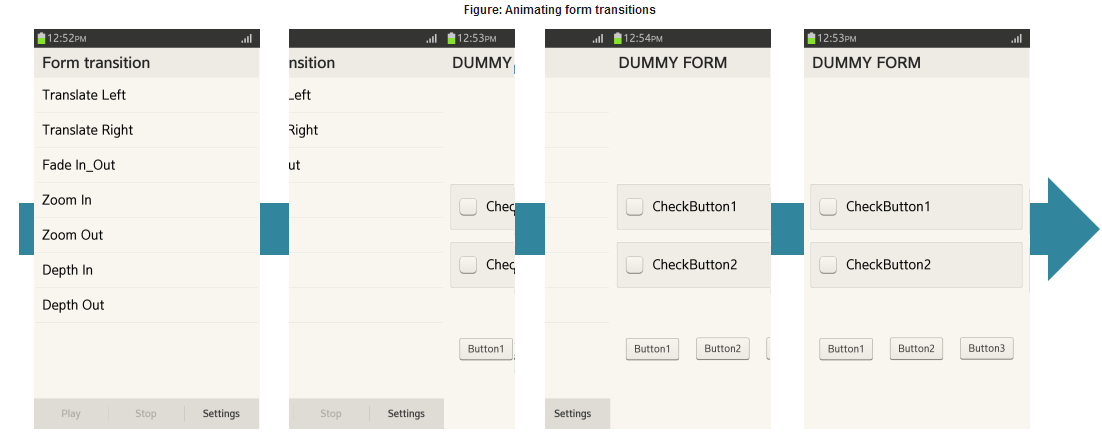
UI Control의 Animating 과정은 다음과 같다.



1. Animating a form transition

Form이동 애니메이션은 앞의 UI Control 애니메이션과 비슷하다.

Form이동 애니메이션을 위해 Tizen::Ui::Animations::FrameAnimator 클래스가 사용된다.



FrameAnimator클래스를 통해 아래의 8가지 이동을 설정할 수 있다.

Zoom in and out

Depth in and out

Fade in and out

Translate left and right

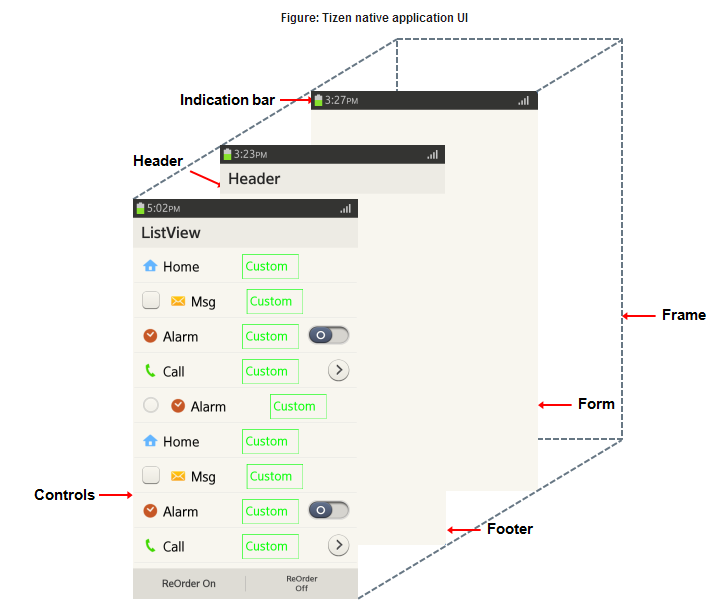
Triggering a Form Transition Animation

SetFormTransitionAnimation()메소드를 사용해 Form Transition Animation을 정의할 수 있다. Tizen::Ui::Animations::FrameAnimatorFormTransitionAnimation type과 duration, animation interpolator type을 파라미터로 받아 사용한다. Form transition에는 최대 1초의 duration이 사용될 수 있고 이처럼 설정을 하지 않을 경우 Fade-in-out, 1ms, linear의 디폴트 값이 적용된다.

* Animating a visual element object

1. VisualElement
2. VisualElementAnimation

**2. Tizen User Interface**

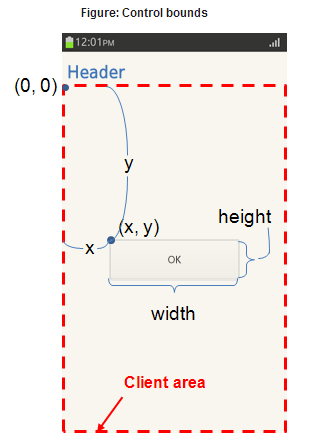


**<Tizen native application의 UI구조>**

Tizen native application UI는 일반적으로 최상위 window를 만드는 풀스크린Frame과 그 안에 Form으로 구성되어있다. 하나의 어플리케이션에는 적어도 하나의 Frame이 포함되어야만 한다. Form은 Indicator bar, header, footer, UI Controls들을 가질 수 있고 하나의 어플리케이션에는 여러 개의 Form이 있을 수 있다.

**3. Coordinate System**

Relative coordinate system에서 모든 Control객체의 위치는 부모 container의 왼쪽 상단으로부터 결정된다. 사용자들은 GetBoundsF(), GetPositionF(), GetSizeF(), GetWidthF(), GetHeightF() 메소드를 통해 모든 Control객체의 위치나 크기 정보를 얻을 수 있다.



**<Control객체의 Bounds>**

또한 사용자들은 Form의 Client area의 Bounds도 GetClientAreaBoundsF() 메소드를 통해 얻을 수 있다. Client area는 System Control객체(Indicator area, header, footer)들을 포함하지 않는다. 만약 System Control객체들이 translucent나 invisible(반투명, 투명)상태이면 Client area는 그 공간을 채우기 위해 확장된다.



**<다양한 조건에서 Client area의 Bounds>**

**4. Control Life-Cycle**

모든 Container와 Control객체들은 device의 heap영역에 만들어져야 한다. 어플리케이션이 종료될 때 Tizen platform은 Frame control객체와 하위객체들을 삭제한다. 그리고 어플리케이션에 할당 되어 있던 heap 메모리를 해제한다.

Tizen platform이 스스로 Control의 Life-cycle을 처리하지만 사용자들이 수정할 수도 있다.

* Control이 초기화 될 때 OnInitializing() event handler는 오버라이딩 될 수 있고 이 곳에서 사용자 정의 initialization을 할 수 있다.
* RemoveControl()과 RemoveAllControls() 메소드로 Controls객체들을 확실히 지울 수 있다. (Container에 등록되지 않은 Controls들은 이 메소드를통해 지우는 것이 좋다.)
* Control이 삭제될 때 OnTerminating() event handler를 오버라이딩해 사용자 정의 clean-up을 수행할 수 있다.

**5. Tizen::Ui namespace의 핵심 클래스 다이어그램**

