

퓨즈를 활용한 모바일 앱 개발하기



퓨즈 튜토리얼: [시작편](#)

About FUSETOOLS,

퓨즈툴스(Fusetools)는 GPU 기반의 UX 렌더링 기술을 사용해 빠른 성능과 애니메이션을 구현하는 네이티브 기반의 크로스플랫폼 앱 개발 툴 제공업체입니다.

2011년 설립된 퓨즈툴스는 모바일 프로세서 기업 ARM사의 GPU 프로세서를 개발했던 멤버들이 모여 설립한 기업으로, 차세대 네이티브 앱 개발 툴의 새로운 패러다임을 제시합니다.

퓨즈의 GPU 멀티스레드 기술은 앱 개발 기간과 비용을 획기적으로 단축해 비즈니스를 성공적으로 이끌어 줍니다. 퓨즈는 UX 마크업과 자바스크립트 만을 사용해 만든 한 개의 소스 코드로, iOS 및 안드로이드 네이티브 앱을 만들어 줍니다. 수정사항을 다양한 디바이스에서 즉각적으로 확인할 수 있습니다.

퓨즈툴스는 2017년, 퓨즈 프로페셔널(Fuse Professional)을 출시하고 본격적으로 모바일 애플리케이션 개발 시장에서의 혁신적인 패러다임을 제시하고 있습니다.

오슬로(노르웨이)에 본사를 둔 퓨즈툴스는 현재 실리콘밸리(미국) 및 서울(한국)에 지사를 두고 있습니다.

퓨즈툴스 코리아
Fusetools Korea
www.fusetools.com/kr

© 2017 Fusetools Korea Inc.
All rights reserved
본 튜토리얼의 모든 내용은
저작권의 보호를 받으므로
무단 복제 및 배포를 금합니다

차례

01 퓨즈란 무엇인가?	5
01.1 모바일 앱의 개발 프로세스	6
01.2 퓨즈란 무엇인가?	12
01.3 퓨즈 개발 환경 설치하기	17
01.4 Hello, World! 퓨즈 앱을 만들기	33
02 하이킹 기록 앱, hikr 만들기	46
02.1 소개	47
02.2 프로젝트 구조	50
02.3 페이지 이해하기	53
02.4 화면 컴포넌트 이해하기	65
02.5 이벤트 처리 및 내비게이션 이해하기	70
03 퓨즈 UX 마크업 언어 배우기	79
03.1 UX 마크업 언어 소개	80
03.2 값의 종류	83
03.3 표현식(expression)	86
03.4 ux: 속성	88
03.5 기본 컨트롤	95

03.6 화면 레이아웃	106
03.7 화면 내비게이션	115
03.8 트리거와 애니메이션	118
04 자바스크립트와 데이터	121
04.1 자바스크립트의 역할	122
04.2 UX 화면과 데이터를 바인딩하기	124
04.3 이벤트 처리하기	126
04.4 Observable의 파워	128
05 퓨즈 스튜디오	131
05.1 소개	132
05.2 퓨즈 스튜디오의 화면 뷰와 설정	137
05.3 멀티 프리뷰 모드	162
06 퓨즈의 고급 기능 활용하기	169
06.1 Uno를 활용한 네이티브 코딩하기	170
06.2 iOS, 안드로이드 프로젝트에 퓨즈 컴포넌트 삽입하기	172

01

퓨즈란

무엇인가?

- 01.1 모바일 앱의 개발 프로세스
- 01.2 퓨즈란 무엇인가?
- 01.3 퓨즈 개발 환경 설치하기
- 01.4 Hello, World! 퓨즈 앱을 만들기

01.1

모바일 앱의 개발 프로세스

최초의 스마트폰인 아이폰이 출시되고 10년이 지난 지금, 스마트폰은 현대인의 필수품이 되었다. PC, 웹 애플리케이션 개발자는 모바일 애플리케이션(이하 앱) 비즈니스에서 비전을 보았다. 그렇지만 아이폰(iOS), 안드로이드(Android) 등 모바일 기기의 플랫폼이 다양해지면서 그들에 탑재될 앱을 만드는 것은 어려워졌다.

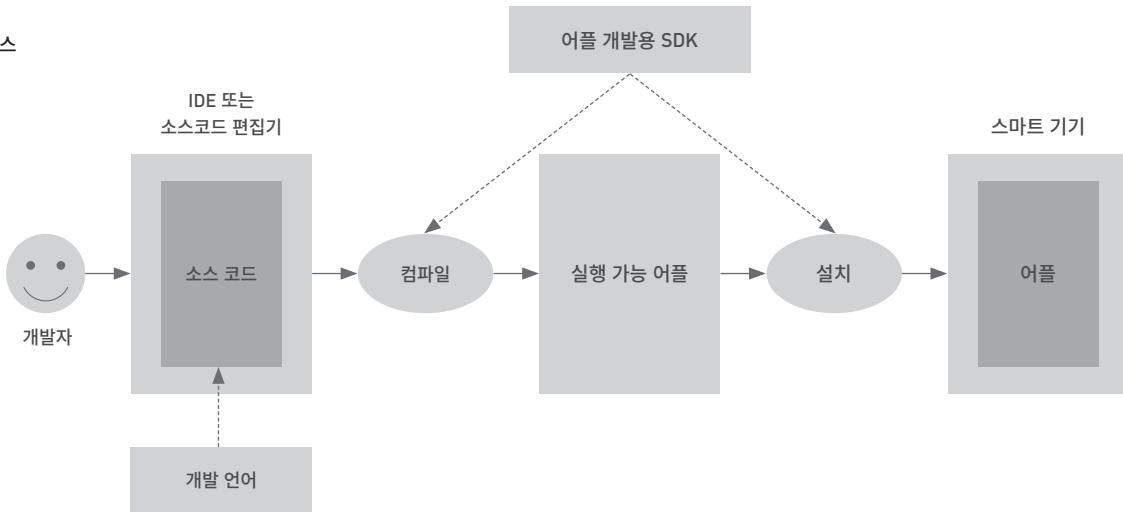
안드로이드 앱과 iOS 앱을 한번에 개발하기 위해서는 전자는 자바 언어, 후자는 Objective-C 또는 Swift 언어를 알아야 한다. 한 명의 개발자가 두 개 이상의 프로그래밍 언어를 익숙하게 사용하여 iOS 앱과 안드로이드 앱을 동시에 개발하는 것은 어려운 일이다. 따라서 앱 개발 회사의 상황에 따라 안드로이드 및 아이폰 개발자가 따로 있거나 아니면 한 파트의 개발자만 있고 나머지는 포기하거나 외주를 주기도 한다.

이런 상황에서 개발자가 아닌 학생 또는 직장인이 앱을, 그것도 안드로이드와 iOS 앱 모두를 개발한다는 것은 불가능에 가까웠다. 하지만 퓨즈(Fuse)가 등장하면서 네이티브 앱 개발의 패러다임이 달라졌다. 퓨즈는 개발자 뿐만 아니라 앱 개발을 배우고 싶은 일반인도 쉽게 안드로이드 및 iOS 네이티브 앱을 만들 수 있는 개발 툴이기 때문이다.

퓨즈를 사용하면 프로그래밍을 전혀 경험하지 않은 학생이나 초보 개발자도 자바, Objective-C 또는 Swift 등의 네이티브 언어로 개발한 것만큼의 고성능 모바일 앱을 다른 어떤 툴을 사용하는 것보다 쉽게 개발할 수 있다. 또한 하나의 프로젝트로 아이폰 및 안드로이드 앱을 동시에 만들 수 있다. 퓨즈의 막강한 화면 구현 언어인 UX 마크업 언어와 대중적인 프로그래밍 언어인 자바스크립트, 네이티브 코드 변환 언어인 Uno, 그리고 화면 설계 툴인 퓨즈 스튜디오(Fuse Studio) 덕분이다.

퓨즈의 특징과 장점을 이해하기 위해서는 스마트폰 앱 개발 프로세스를 이해하는 게 좋다.

스마트폰 앱 개발 프로세스

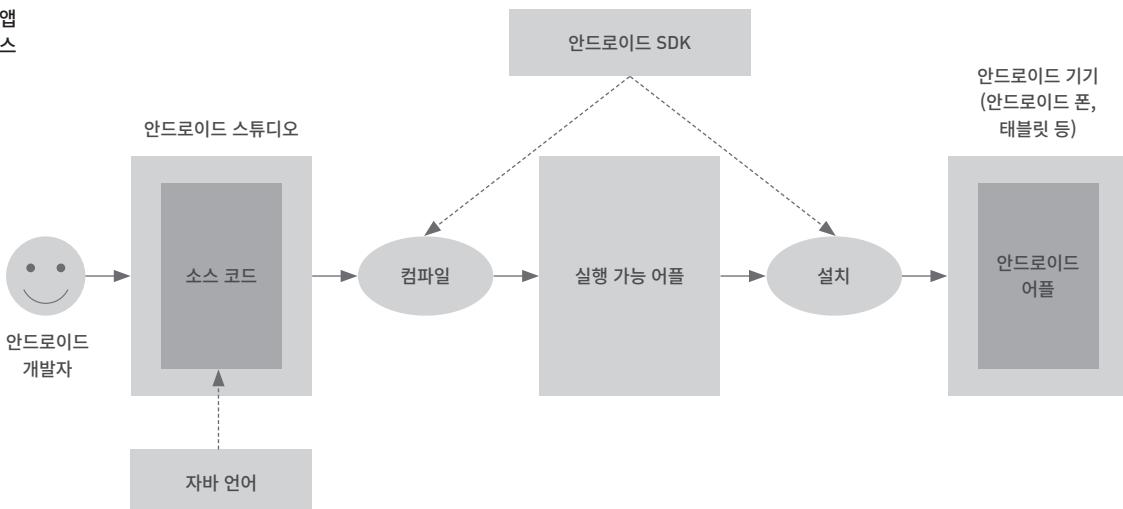


일반적으로 iOS와 안드로이드 앱 개발자는 해당 플랫폼의 개발 언어를 배우고 전용 IDE(Integrated Development Environment, 통합 개발 환경), 앱 개발용 SDK(Software Development Kit)를 컴퓨터에 설치하고 사용법을 익혀야 한다. IDE는 개발자가 소스코드를 손쉽게 편집하고 명령을 처리해주는 프로그램이고 앱 개발용 SDK는 소스코드를 컴파일하여 앱을 빌드하는 툴이다.

개발자는 IDE를 실행하여 소스코드를 작성하고 앱 개발용 SDK를 사용하여 컴파일, 빌드, 설치라는 과정을 거쳐 모바일 기기에 앱을 설치하고 테스트한다.

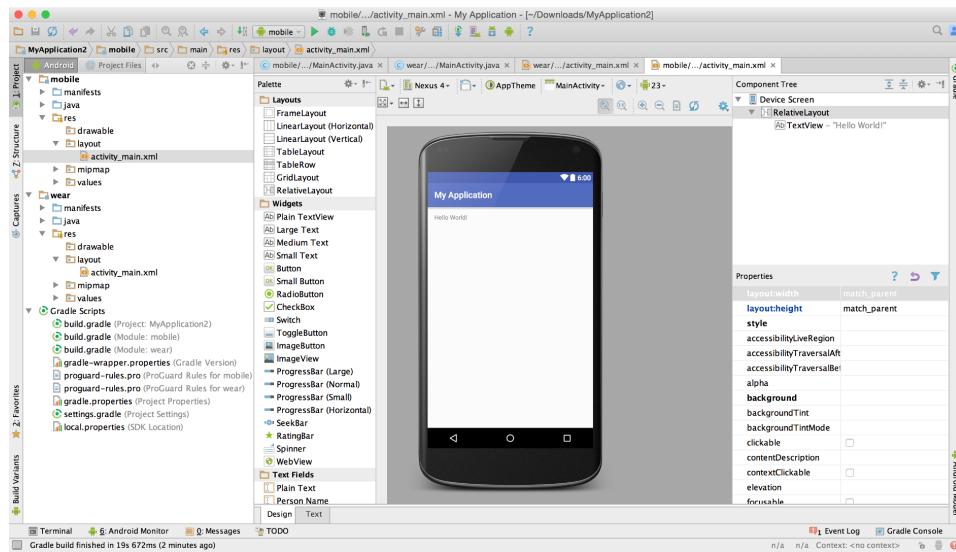
1. 안드로이드 어플 개발 프로세스

안드로이드 앱 개발 프로세스



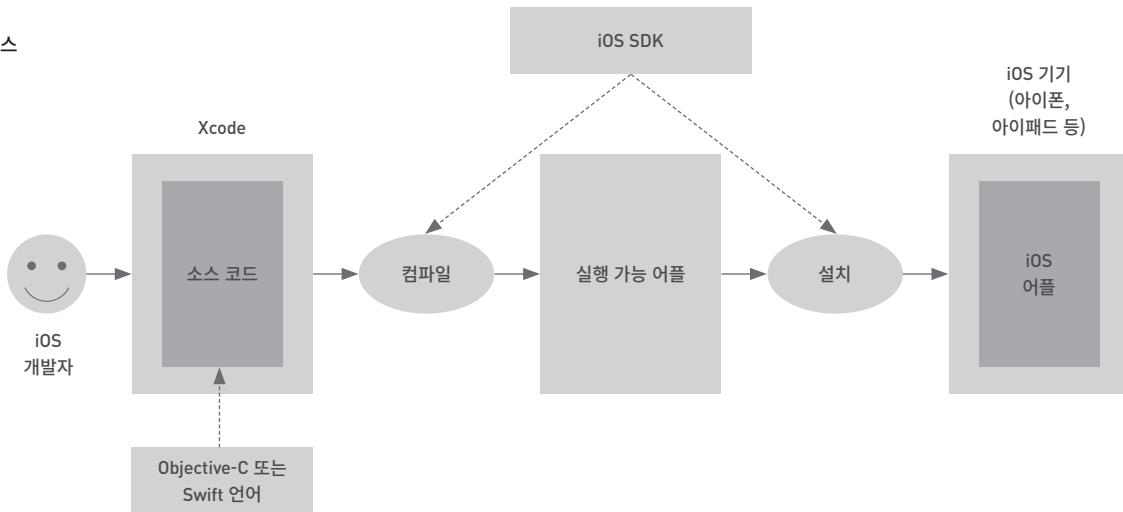
안드로이드 앱 개발자는 자바 언어를 배워야 하며 <https://developer.android.com/studio/index.html>에서 IDE인 안드로이드 스튜디오(Android Studio)와 앱 개발 SDK인 안드로이드 SDK를 다운로드 받아 설치해야 한다. 안드로이드 스튜디오를 실행하여 소스코드를 작성하고 안드로이드 SDK를 활용하여 컴파일 및 설치를 진행한다.

안드로이드 스튜디오 실행 화면



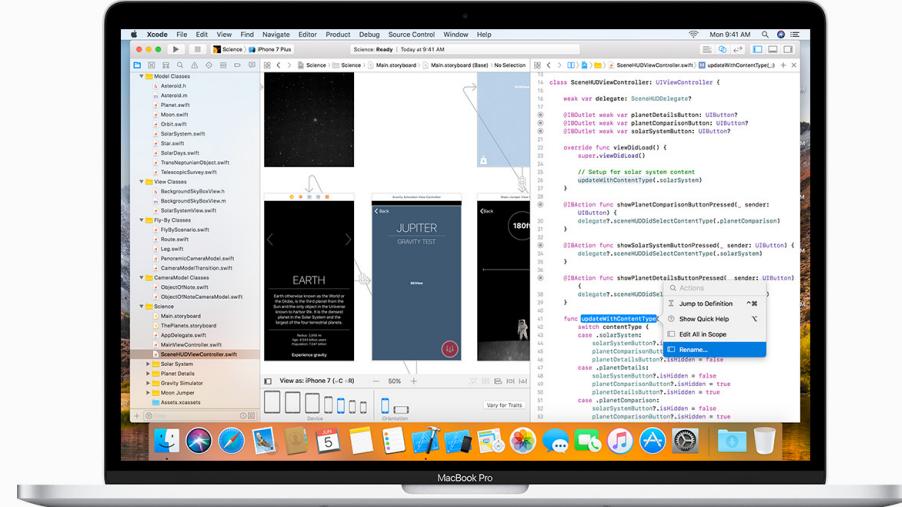
2. iOS 앱 개발 프로세스

iOS 앱 개발 프로세스



아이폰과 아이패드 등에서 동작하는 iOS 앱을 만드는 개발자는 Objective-C 또는 Swift 언어를 배워야 하며 <https://developer.apple.com/xcode>에서 IDE인 Xcode와 앱 개발 SDK인 iOS SDK를 다운로드 받아 설치해야 한다. Xcode 안에 iOS 최신 SDK가 포함되어 있다. Xcode를 실행한 후 소스코드를 작성하고 iOS SDK를 활용하여 컴파일 및 설치를 진행한다.

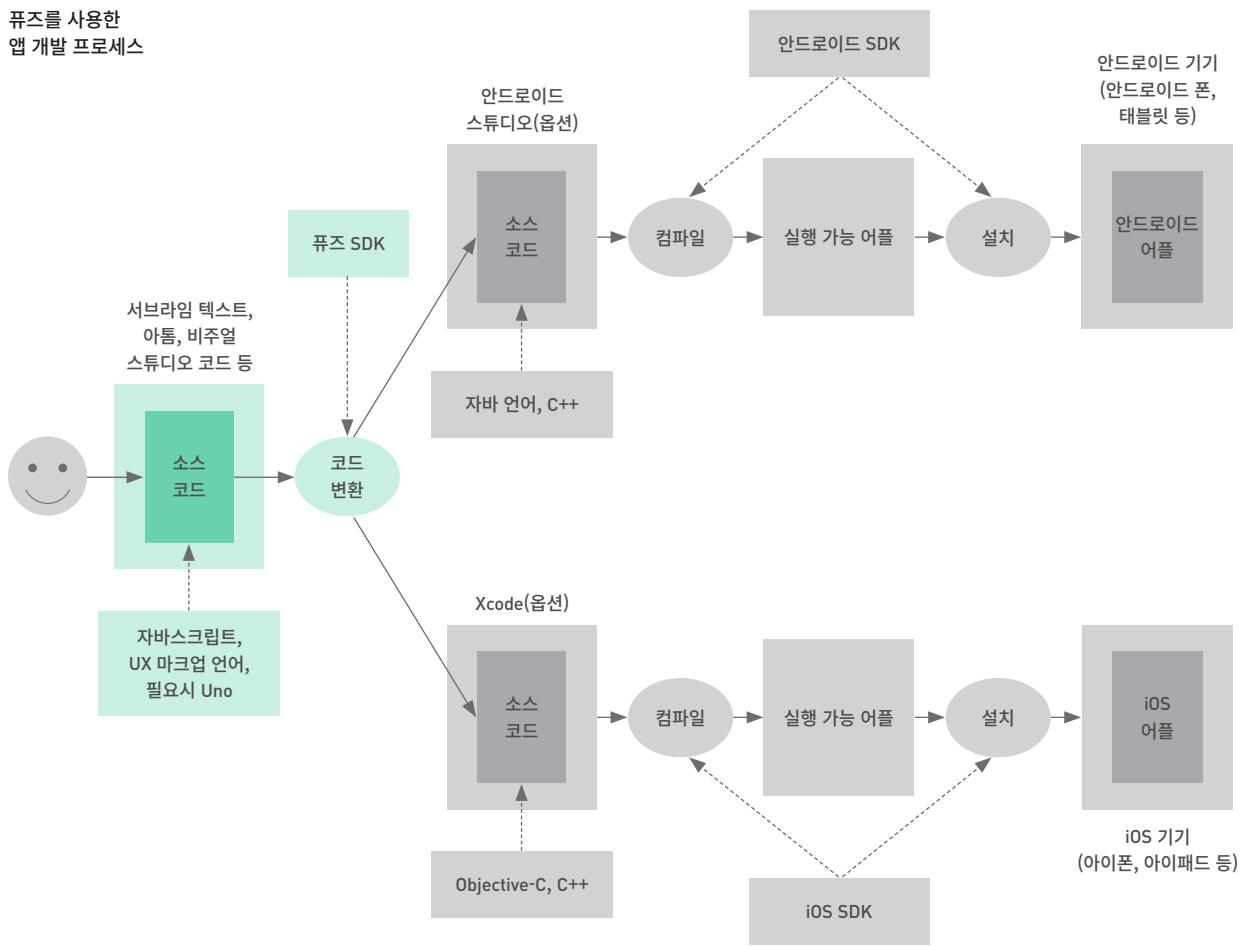
Xcode 실행화면 1



Xcode 실행화면 2



3. 퓨즈를 활용한 네이티브 기반의 크로스 플랫폼 앱 개발 프로세스



개발 SDK로 컴파일 및 빌드함으로써 앱을 만들 수 있다. 이와 같은 원리로 퓨즈를 사용하면 한 프로젝트에서 안드로이드 앱과 iOS 앱을 동시에 개발할 수 있게 되는 것이다.

퓨즈는 자바스크립트 및 UX 마크업 언어 등의 소스코드를 각 모바일 기기 플랫폼에서 지원하는 개발 언어의 소스코드로 변환시킴으로 해당 플랫폼의 개발 언어로 직접 개발한 것과 같은 효과를 제공한다. 이런 특성을 “네이티브(Native) 지원”이라고 한다. 또한 한 프로젝트에서 여러 모바일 플랫폼(안드로이드 및 iOS)의 앱을 만들 수 있으므로 다양한 플랫폼을 커버한다는 의미로 크로스 플랫폼(Cross Platform)이라 부른다.

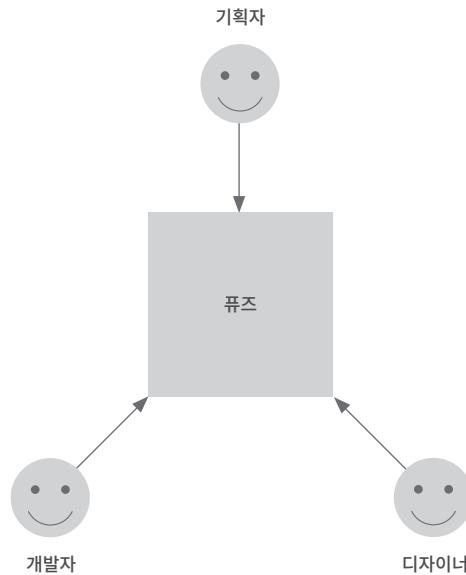
따라서 퓨즈는 “네이티브 기반의 크로스 플랫폼 앱 개발 툴”이라고 부를 수 있다.

01.2

퓨즈란 무엇인가?

퓨즈(Fuse)는 노르웨이에 본사를 둔 퓨즈툴스 사가 개발한 모바일 앱 개발 툴로 디자인, 프로토타이핑, 구현 기능이 통합되어 있어서 고품질의 모바일 앱을 개발할 수 있다. (출처: fusetools.com/docs) 이 말은 퓨즈의 사용자가 앱 디자인과 프로토타이핑을 한꺼번에, 아니면 일부만 활용할 수도 있다는 뜻이다.

퓨즈의 사용자



또한 네이티브 기반의 크로스 플랫폼 앱을 개발할 수 있는 툴이다. 따라서 안드로이드와, iOS 두 가지 앱을 한꺼번에 만들 수도 있고, 아니면 한 종류만 만들 수도 있다.

1. 구성 요소

퓨즈는 크게 3가지 기술 요소로 구성되어 있다.

1) UX 마크업 언어

모바일 앱의 화면을 정의하는 언어이다. XML을 따르는 선언 방식의 리액티브 (declarative-reactive) 언어이다. UX 마크업 언어로 작성된 파일은 확장자를 ux로 해야 한다.

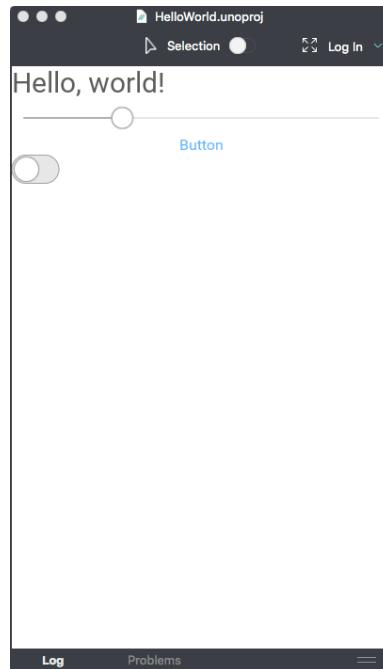
퓨즈는 UX 마크업 언어로 작성된 코드를 OpenGL을 사용하는 C++ 소스코드로 변환하여 빠른 성능의 모바일 화면을 만든다.

MainView.ux

```
<App>
  <StackPanel>
    <Text FontSize="30">Hello, world!</Text>
    <Slider />
    <Button Text="Button" />
    <Switch Alignment="Left" />
  </StackPanel>
</App>
```

위의 소스코드는 아래의 앱 화면을 상징한다.

MainView.ux이 나타내는 앱 화면



2) 자바스크립트 언어

앱의 화면이 아니라 기능을 구현할 때는 자바스크립트(JavaScript) 언어를 사용 한다. 자바스크립트 코드를 UX 마크업 언어 파일 내부에 포함시키거나 아니면 별도의 파일로 분리하여 작성한다.

자바스크립트가 포함된
MainView.ux

```
<App>
  <JavaScript>
    var Observable = require('FuseJS/Observable');
    var buttonText = Observable('Button');
    var clickCount = 0;

    function onClick() {
      clickCount += 1;
      buttonText.value = 'Clicks: ' + clickCount;
    }

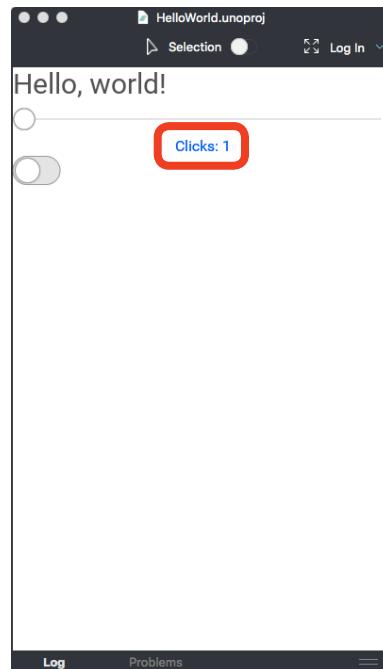
    module.exports = {
      buttonText: buttonText,
      onClick: onClick
    }
  </JavaScript>

  <StackPanel>
    <Text FontSize="30">Hello, world!</Text>
    <Slider />
    <Button Text="{buttonText}" Clicked="{onClick}" /> ②
    <Switch Alignment="Left" />
  </StackPanel>
</App>
```

위의 MainView.ux에서 버튼을 클릭했을 때, 버튼 이름이 “Clicks: 숫자”로 변경되는 기능을 자바스크립트로 구현했다. (① 참고) 버튼을 클릭했을 때 자바스크립트의 함수가 호출되도록 설정하기 위해 Button 컴포넌트에 Clicked 이벤트 핸들러를 추가했다. (② 참고)

버튼을 클릭하면 버튼 이름이 “Clicks: 1”로 변하고 계속 누르면 클릭 회수가 “Clicks: X”的 X 자리에 들어간다.

버튼을 클릭했을 때
버튼 이름이 변경되는 화면



3) Uno 언어

Uno 언어로 Example 클래스를 만들었는데 ①이 안드로이드 기기에서 동작하는 자바 언어, ②가 iOS 기기에서 동작하는 Objective-C다. 보통 자바스크립트로 구현할 수 없는 기능을 Uno 언어를 사용해서 네이티브 코드로 구현하는 경우가 많다.

Uno 예제

```
using Uno.Compiler.ExportTargetInterop;

class Example
{
    [Foreign(Language.Java)]
    public static extern(Android) void Log(string message)
    @{
        android.util.Log.d("ForeignCodeExample", message);
    }

    [Foreign(Language.ObjC)]
    public static extern(iOS) void Log(string message)
    @{
        NSLog(@"%@", message);
    }
}
```

The diagram shows two code blocks enclosed in brackets. The first bracket, labeled ①, encloses the Java code block. The second bracket, labeled ②, encloses the Objective-C code block.

2. 할 수 있는 일

퓨즈를 사용하여 할 수 있는 대표적인 일은 아래와 같다.

- 1 앱의 화면을 XML 방식의 UX 마크업 언어로 디자인한다.
- 2 퓨즈 스튜디오(Fuse Studio)를 사용하면 UX 마크업 언어로 작성된 ux 파일을 직접 수정하지 않고 편리하게 화면의 레이아웃과 스타일을 지정할 수 있다. (퓨즈 프로페셔널 라이센스에서만 가능)
- 3 모바일 앱의 동작 화면을 모바일 기기가 아니라 컴퓨터에서 바로 확인한다. (리얼타임 프리뷰 기능)
- 4 모바일 앱의 기능을 자바스크립트로 구현한다.
- 5 자바스크립트로 구현할 수 없는 기능은 자바, Objective-C, Swift 등의 네이티브 언어를 사용하여 구현한다. 이때 Uno 언어를 사용해 네이티브 언어로 작성된 코드를 패키징해야 한다.
- 6 앱을 안드로이드 및 iOS 기기에 직접 설치하여 실시간으로 동작 과정을 확인한다. (리얼타임 프리뷰 기능)
- 7 앱을 구글 플레이, 애플 앱스토어에 업로드 한 후 출시한다.
- 8 퓨즈로 개발한 컴포넌트를 안드로이드 및 iOS 프로젝트에 포함시킨다. (퓨즈 프로페셔널 라이센스에서만 가능)
- 9 미리 구현된 차트 컴포넌트를 사용하여 아름다운 차트를 만든다. (퓨즈 프로페셔널 라이센스에서만 가능)

3. 라이센스

퓨즈 사용자는 크게 3가지 라이센스 중 하나를 선택하여 사용할 수 있다. (출처: <https://www.fusetools.com/plans>) 본 책은 독자가 퓨즈 프로페셔널 라이센스를 갖고 있다는 가정 하에 집필되었다.

1 프리(Free)

퓨즈의 기본적인 기능을 제공한다. 누구나 무료로 사용할 수 있다.

2 프로페셔널(Professional)

프리 라이센스에서 제공하는 기능 외에 ▲퓨즈 스튜디오, ▲Xcode 및 안드로이드 스튜디오에 퓨즈 컴포넌트를 포함하는 기능, 차트와 같은 ▲프리미엄 컴포넌트를 제공한다. 비용을 지불해야 라이센스를 구매할 수 있다.

3 커스텀(Custom)

프로페셔널 라이센스에서 제공하는 기능 외에 1:1 다이렉트 기술 지원이 가능하며 공개되지 않은 최신 퓨즈 툴을 다운로드하여 사용할 수 있다. 퓨즈툴스 사와 별도로 비용을 협의해야 한다.

01.3

퓨즈 개발 환경 설치하기

1. 지원 플랫폼

퓨즈는 아래의 컴퓨터 운영체제에서 동작한다.

- 1 맥: macOS 10.10(요세미티) 버전 이상.
- 2 윈도우: 윈도우 7 이상, Open-GL 2.1 버전이 동작 가능한 GPU가 아니라면 오동작 가능성이 있음.

퓨즈를 사용해 앱을 만들면 아래의 모바일 플랫폼에서 동작한다.

- 1 Android : 4.1(젤리빈, API 레벨 16) 이상.
- 2 iOS : 8.0 버전 이상.

2. 소스코드 편집 툴 설치하기

퓨즈(=퓨즈 SDK)를 설치하기 전에 서브라임 텍스트 또는 아톰 중 하나를 택일하여 설치하는 것을 추천한다. 왜냐하면 퓨즈를 설치할 때 위의 툴이 존재하면 자동으로 UX 마크업 언어의 코드 하이라이트, 프리뷰 및 각종 명령 기능이 포함된 플러그인을 설치하기 때문이다. 필자는 속도가 빠른 서브라임 텍스트를 선호한다.

- 1 서브라임 텍스트 설치: <https://www.sublimetext.com/3>

위의 사이트에 접속하여 운영체제에 서브라임 텍스트 설치파일을 다운로드하고 설치한다.

Home Download Buy Blog Forum Support

Sublime Text 3

Download

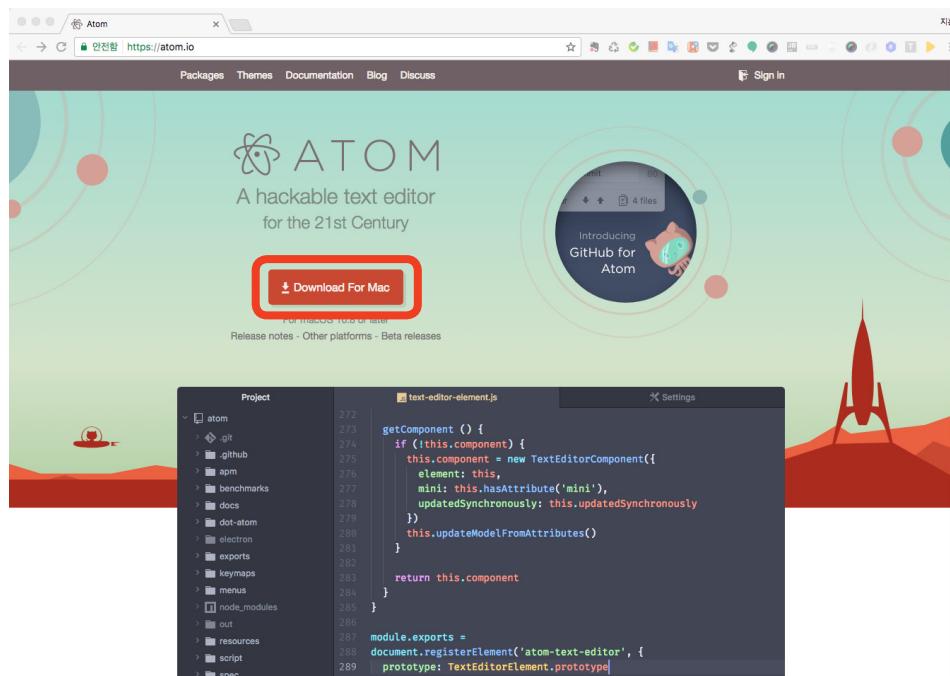
Sublime Text 3 is currently in beta. The latest build is 3126.

- OS X (10.7 or later is required)
- Windows - also available as a [portable version](#)
- Windows 64 bit - also available as a [portable version](#)
- Linux repos - also available as a [64 bit](#) or [32 bit tarball](#)

2 아톰 설치: <https://atom.io/>

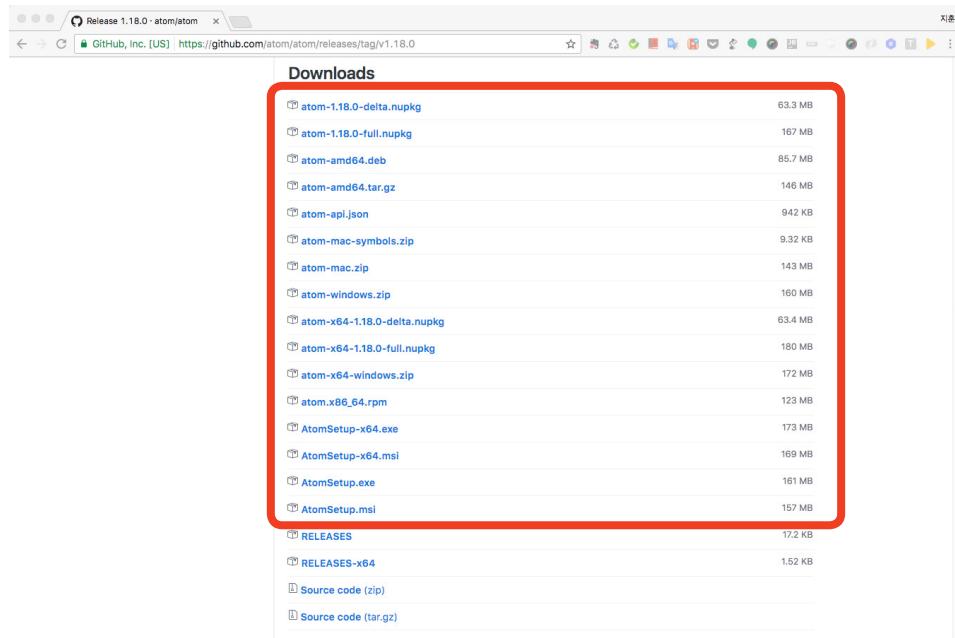
위의 사이트에 접속하여 운영체제에 맞는 아톰 설치파일을 다운로드하고 설치한다.

아톰 다운로드 페이지 1



아톰은 윈도우, 리눅스, 맥 등 여러 운영체제를 지원한다. 자신에게 필요한 운영체제 설치 파일을 아래 링크에서도 다운로드 할 수 있다. (예: <https://github.com/atom/atom/releases/tag/v1.18.0>)

아톰 다운로드 페이지 2

3 비주얼 스튜디오 코드: <https://code.visualstudio.com/>

퓨즈는 서브라임 텍스트와 아톰의 플러그인만을 공식적으로 지원하지만 퓨즈 커뮤니티 내 개발자가 만들어서 공개한 비주얼 스튜디오 코드용 퓨즈 플러그인도 있다.

비주얼 스튜디오 코드는 마이크로소프트 사에서 만든 소스코드 편집기이다. 윈도우, 맥, 리눅스에 설치할 수 있다. 코드 하이라이트, 오토 컴플리션, 디버깅 등의 핵심 기능이 빠르고 가볍게 동작하므로 인기가 많다. 주로 HTML, CSS, 자바스크립트 등의 소스코드를 편집할 때 사용되지만 2017년 7월 현재, 퓨즈 익스텐션(Fuse Extension)이란 플러그인(plugin)을 설치하면 퓨즈의 각종 소스코드를 하이라이팅, 오토컴플리션 기능을 사용하여 편집할 수 있다.

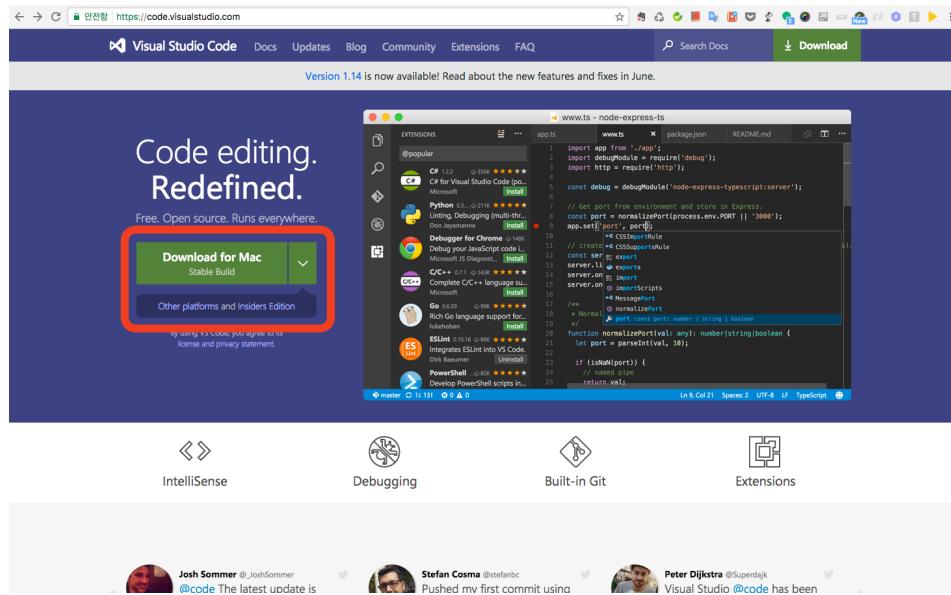
- 퓨즈 익스텐션 플러그인 소개 페이지: <https://marketplace.visualstudio.com/items?itemName=iGN97.fuse-vscode>

이 플러그인을 비주얼 스튜디오 코드에 설치하면 위에서 소개한 서브라임 텍스트 또는 아톰에서 제공하지 않는 오토 컴플리션 기능을 사용할 수 있으므로 더 편리하게 퓨즈 소스코드를 만들 수 있다.

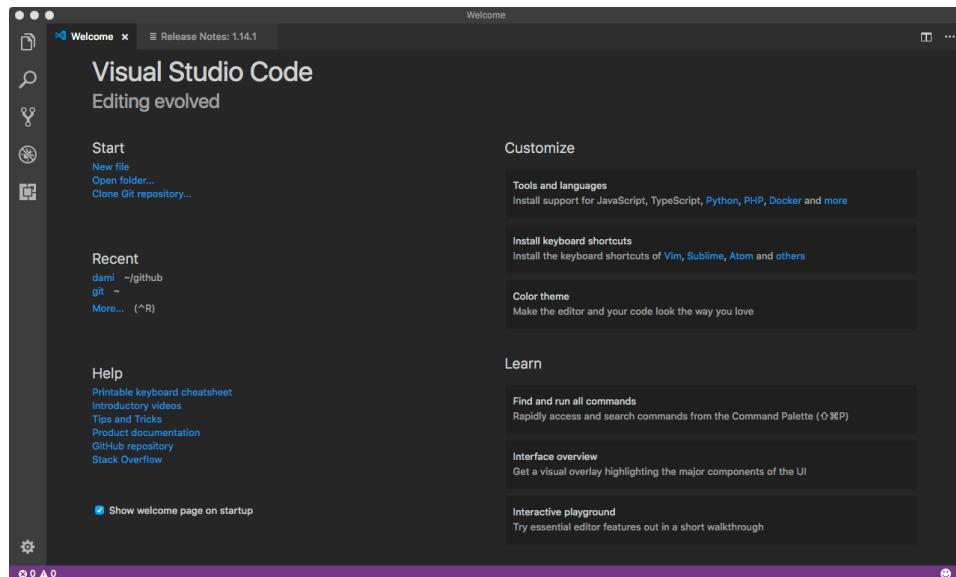
설치 방법은 다음과 같다.

- https://code.visualstudio.com에 접속하여 자신의 운영체제에 맞는 설치 파일을 다운로드하고 설치한다.

비주얼 스튜디오 코드 다운로드 페이지

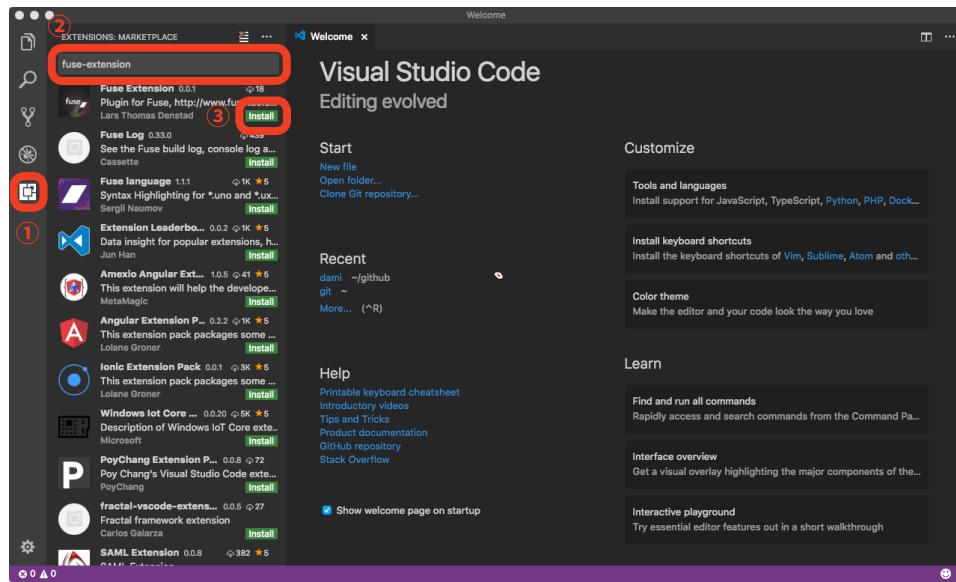


비주얼 스튜디오 코드 실행



- 퓨즈 익스텐션 플러그인을 아래의 순서대로 설치한다.
 - 1 단계: 비주얼 스튜디오 코드의 좌측 사이드 바 하단에 있는 플러그인 아이콘을 선택한다.
 - 2 단계: 플러그인 검색창에 fuse-extension을 입력한다.
 - 3 단계: 검색된 fuse-extension의 “install” 버튼을 선택하여 설치한다.

퓨즈 익스텐션 플러그인을 설치하기



만약 UX 마크업 언어로 소스코드를 작성할 때 오토 컴플리션 기능의 도움을 받고 싶다면 서브라임 텍스트, 아톰이 아니라 비주얼 스튜디오 코드와 관련 플러그인을 설치하기 바란다.

3. 퓨즈 SDK 설치하기

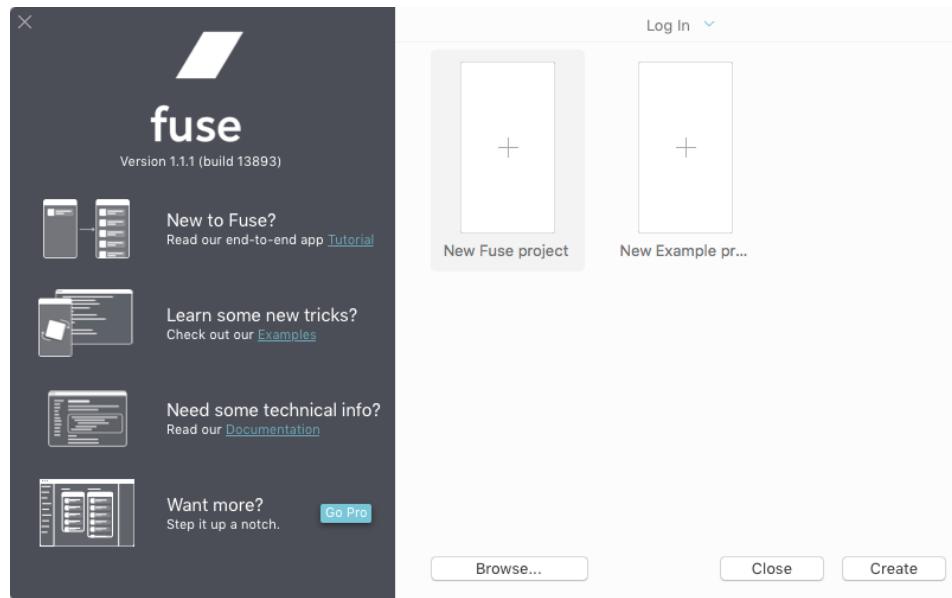
1) 맥 설치 방법

가. 퓨즈 설치하기

- <https://www.fusetools.com/downloads>에서 맥용 퓨즈 SDK를 다운로드하고 설치한다.
- 설치 후 명령 창에서 “fuse”를 실행했을 때 아래와 같은 실행화면이 나오면 잘 설치된 것이다.
- 안드로이드 스튜디오(<https://developer.android.com/studio/index.html>)와 관련 SDK를 다운로드하고 설치한다. 아니면 아래 명령어를 명령 프롬프트 또는 터미널에 입력하여 퓨즈에 필요한 안드로이드 SDK를 설치한다.

```
fuse install android
```

퓨즈 첫 실행화면

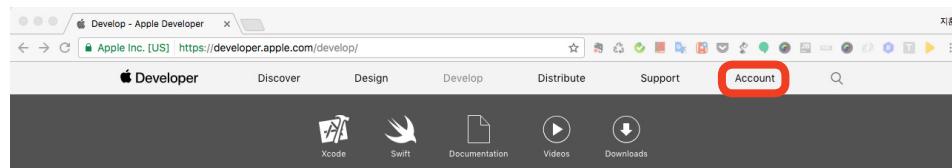


나. Xcode 설치 및 개발자 계정 가입하기

- 맥에서 iOS 개발 툴인 Xcode(<https://developer.apple.com/xcode>)를 설치한다.
- 무료로 애플 개발자 계정을 만든다.

<https://developer.apple.com/>의 “Account” 메뉴를 선택한 후 “Create Apple ID” 버튼을 눌러서 사용자 등록을 진행한다.

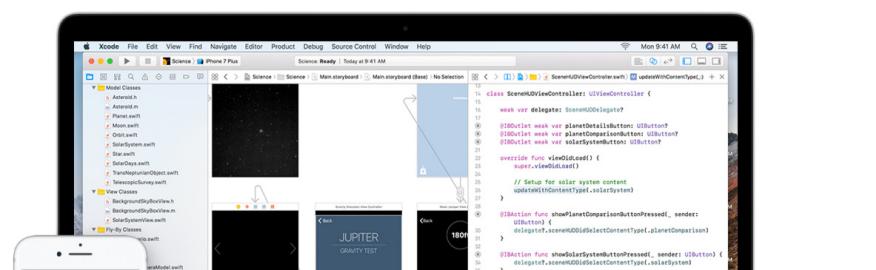
애플 개발자 계정 만들기 1



Develop

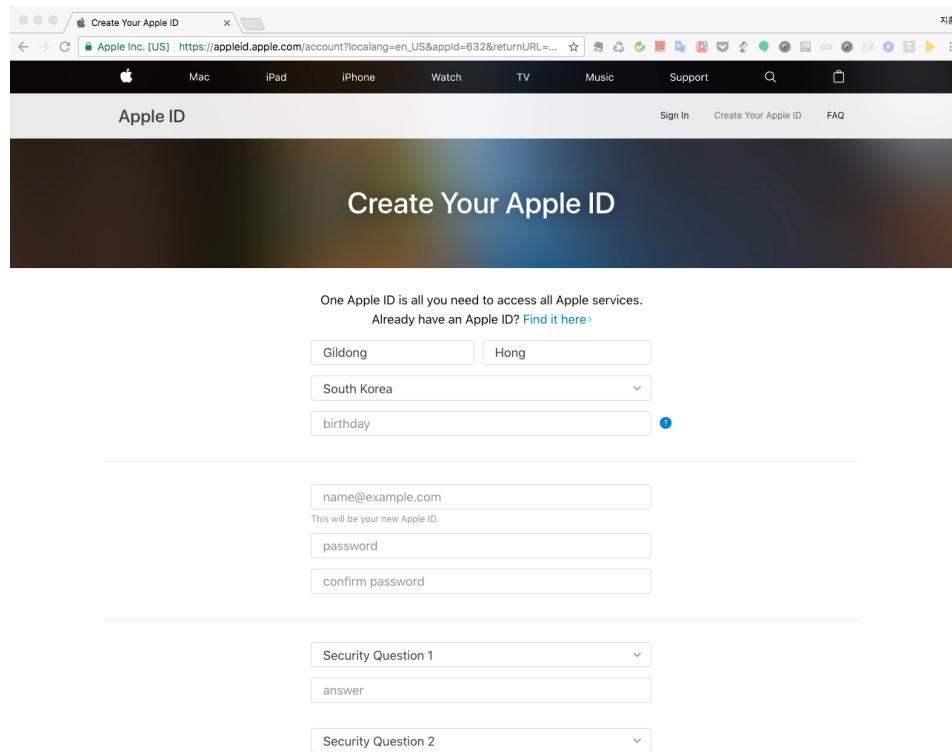
Bring Your Ideas to Life

With the power of Xcode, the ease of Swift, and the revolutionary features of cutting-edge Apple technologies, you have the freedom to create your most innovative apps ever.



- 개발자의 이메일 계정 및 암호, 관련 정보를 입력한다.

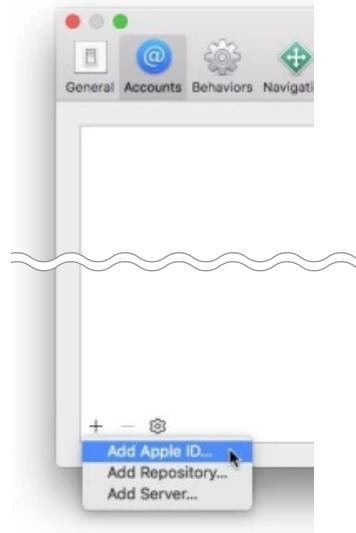
애플 개발자 계정 만들기 2



- Xcode의 설정 메뉴에서 애플 개발자 계정을 등록한다.

개발자 계정을 만든 후에는 Xcode 내에 계정 정보를 등록해야 iOS 기기에 앱을 설치할 수 있다. Xcode를 실행한 후 설정(Preferences) 메뉴를 선택하고 Account 항목으로 이동한 후, + 버튼을 눌러서 계정 정보를 입력한다.

Xcode에 애플 개발자 계정 등록하기



2) 윈도우 설치 방법

- <https://www.fusetools.com/downloads>에서 윈도우 용 퓨즈 SDK를 다운로드하고 설치한다.
- 안드로이드 스튜디오(<https://developer.android.com/studio/index.html>)와 관련 SDK를 다운로드하고 설치한다. 아니면 아래 명령어를 명령 프롬프트(윈도우) 또는 터미널(맥)에 입력해서 퓨즈에 필요한 안드로이드 SDK를 설치한다. 또한 안드로이드 기기와 USB로 연결할 때 필요한 USB 드라이버를 설치할 필요가 있다. (<https://developer.android.com/studio/run/oem-usb.html#Drivers> 참고)

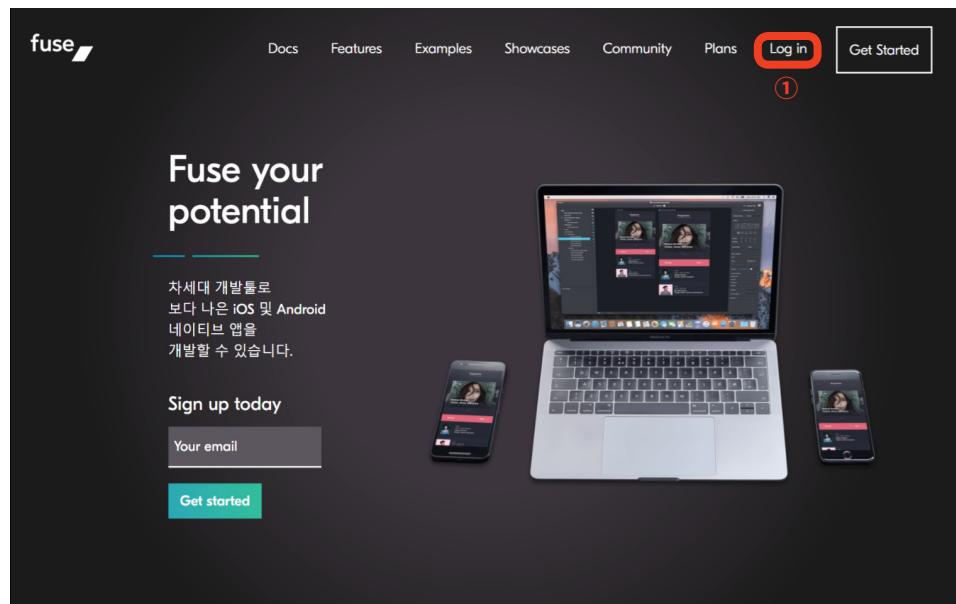
```
fuse install android
```

- 윈도우에서는 아이폰과 같은 iOS기기에 앱을 설치할 수 없으니 Xcode는 필요 없다. 오로지 안드로이드 앱만 개발할 수 있다.

3) 윈도우 설치 상세 프로세스

위의 1), 2)의 내용을 따라서 설치하기 힘든 사용자를 위해 윈도우 운영체제에서 퓨즈 SDK를 설치하는 상세 프로세스를 설명한다. 맥 운영체제도 비슷하다.

- 1 <https://www.fusetools.com/kr> 사이트에 접속 후 ①“Log In”을 클릭한다.



- 2 이미 등록한 fusetools 계정이 있다면 ①영역에 정보를 입력 한 후 Login을 진행하고 등록 된 계정이 없다면 ②”here”를 클릭하여 신규가입 페이지로 이동한다.

Welcome back.
Fill in your credentials in the form below to continue.

①

E-mail address:
Your e-mail address

Password:
Your password

Remember me on this computer

Log in

[Forgot your password?](#)

New user? Register [here](#). ②

- 2-1 [신규가입] 아이디로 사용 할 이메일을 ①입력란에 입력 한 후 ②“Sign up”을 클릭한다.

Register new account

Get a free Fuse account. You need an account to download Fuse Free, to start a trial or to buy Fuse Professional. We never share your email with anyone.

①

Email

Already have an account? [Log in.](#)

② **Sign up**

- 2-2 [신규가입] 등록한 이메일로 Fuse에서 아래와 같은 이메일이 발송된다. ① “go here”를 클릭하여 fusetools의 계정인증 페이지로 이동한다.

Welcome to the Fuse community!

To activate and log in to your account [go here](#). ①

We're so happy to have you with us, and look forward to see what you'll create with Fuse!

- Anders, Morten & the rest of the Fuse team

PS: we invite you to check out our [tutorial](#) to get started. You'll also find docs and examples there, and you can [log in](#) to update your profile or download Fuse (for OS X and Windows).

2-2 [신규가입] ①영역의 이름, 패스워드, 패스워드확인을 입력하고 라이선스 동의 항목을 선택 한 후 ②“Confirm and submit”을 클릭하여 가입을 완료 한다. (가입완료 후 자동 로그인 된다.)

Activate account

Complete the form below to finish creating your account and download Fuse.

Step 1: Verify e-mail

Email address:

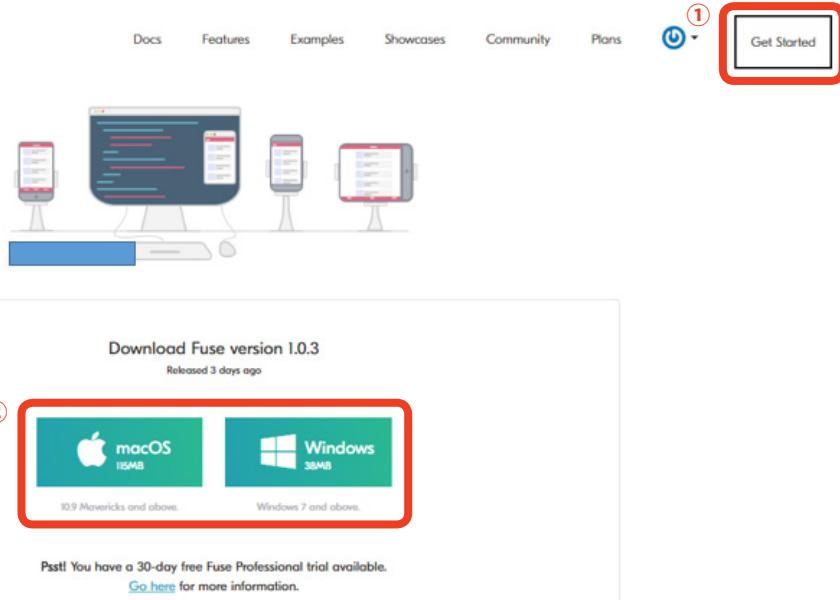
Step 2: Complete account details

① Name: Select a password:
Confirm your new password:

I accept the license agreement for Fuse

②

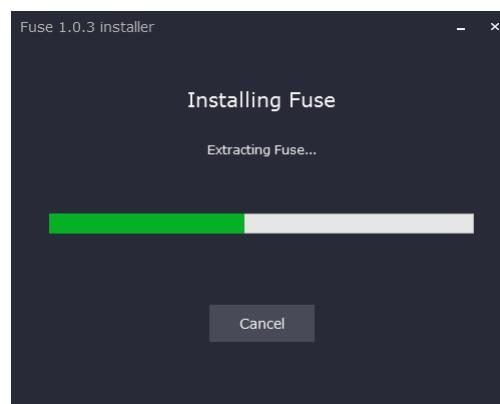
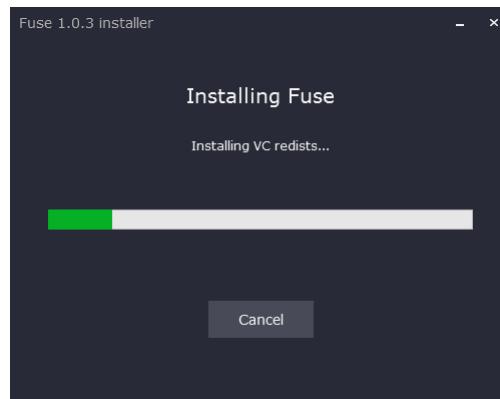
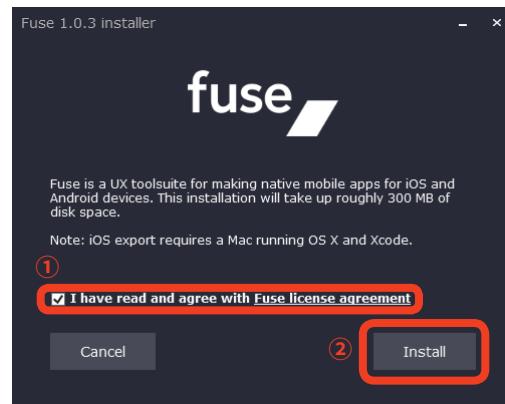
3 “Get Started” 클릭하여 Download 페이지로 이동한 후 운영체제에 맞는 ②Fuse 설치파일을 다운로드 한다. (이 책을 작성하는 기준으로 최신 버전은 1.0.3이다.)



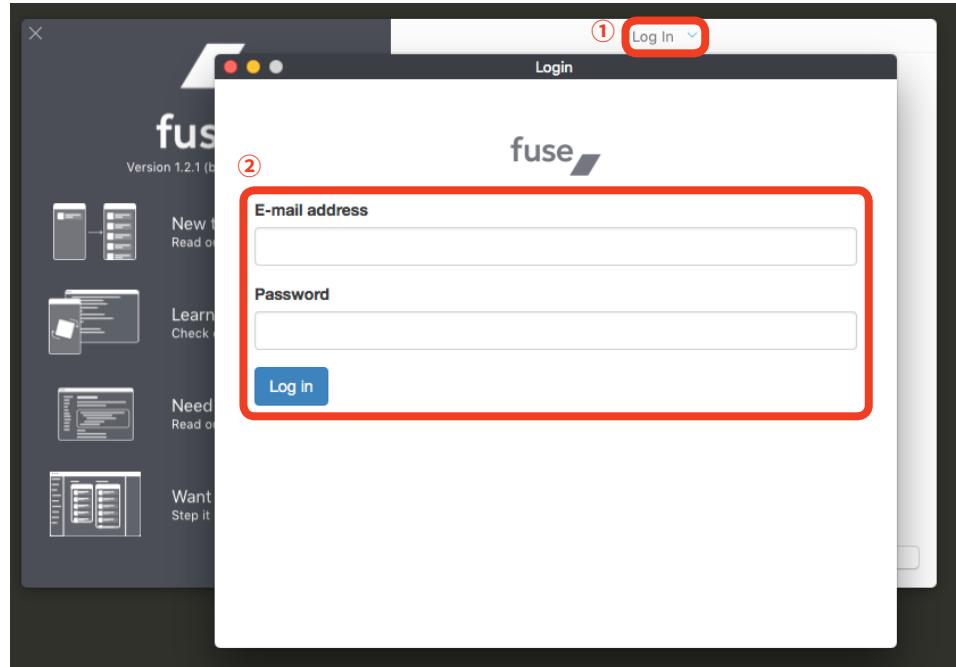
4 다운로드 한 설치파일을 실행한다.

이름	수정한 날짜	유형	크기
fuse_win_1_0_3_13739	2017-07-07 오후...	응용 프로그램	39,506KB

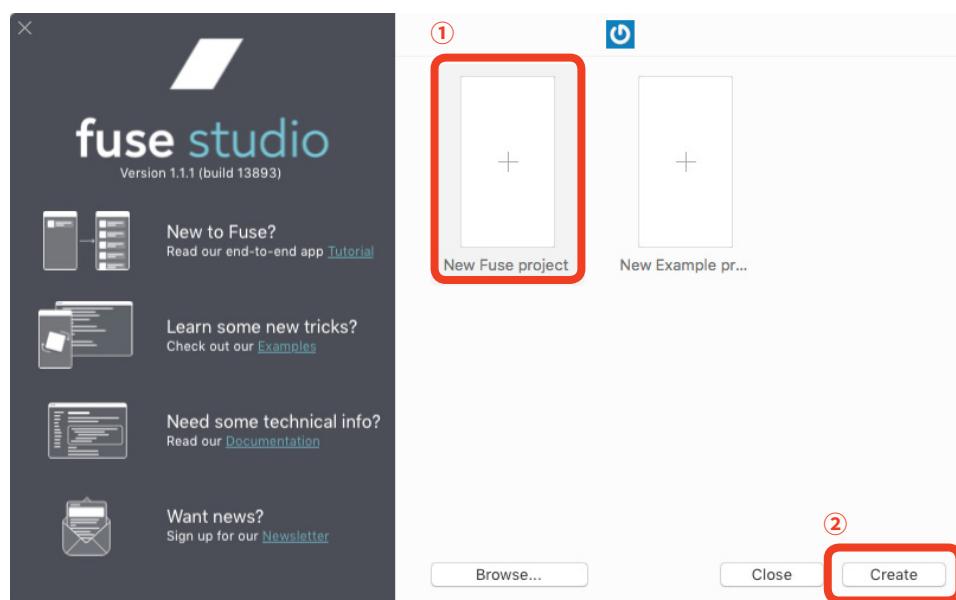
5 ①라이선스 동의를 선택 한 후 ②“Install”을 클릭하여 설치를 진행한다.

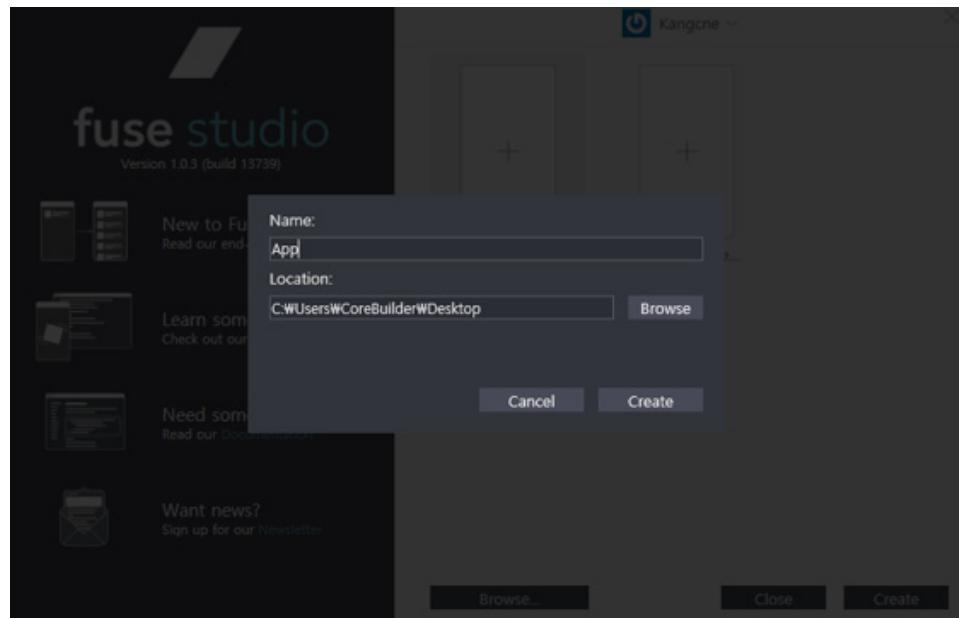


- 6 Fuse 설치가 완료 되면 자동으로 Fuse Studio가 실행된다. ①”Log In”을 클릭하면 ②로그인 창이 나타난다. ②영역에 Fusetools에 등록한 이메일과 패스워드를 입력한 후 로그인을 한다.



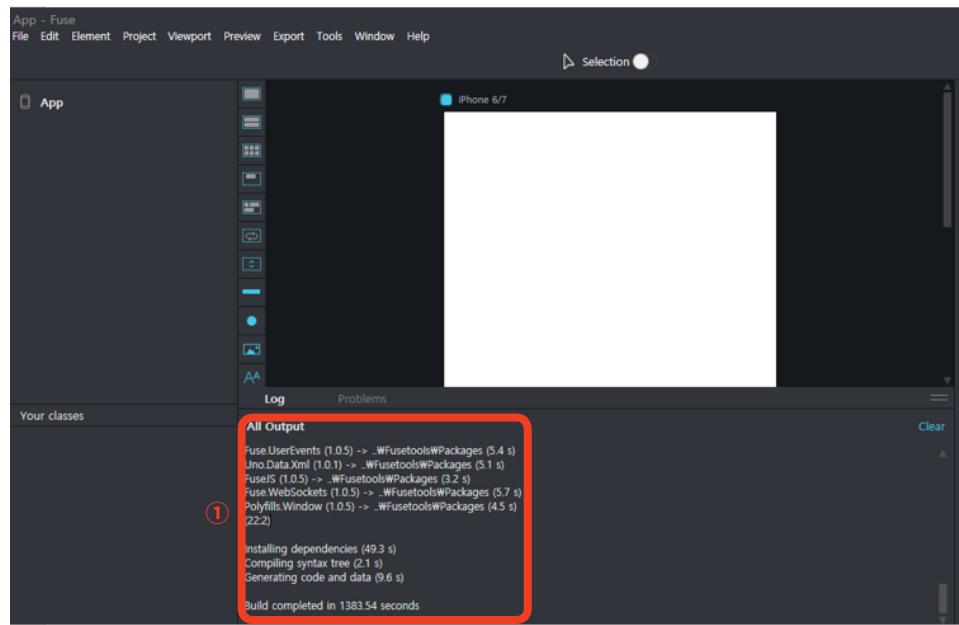
- 7 ①“New Fuse project”를 선택하고 ②“Create” 클릭하여 새 프로젝트를 생성한다.



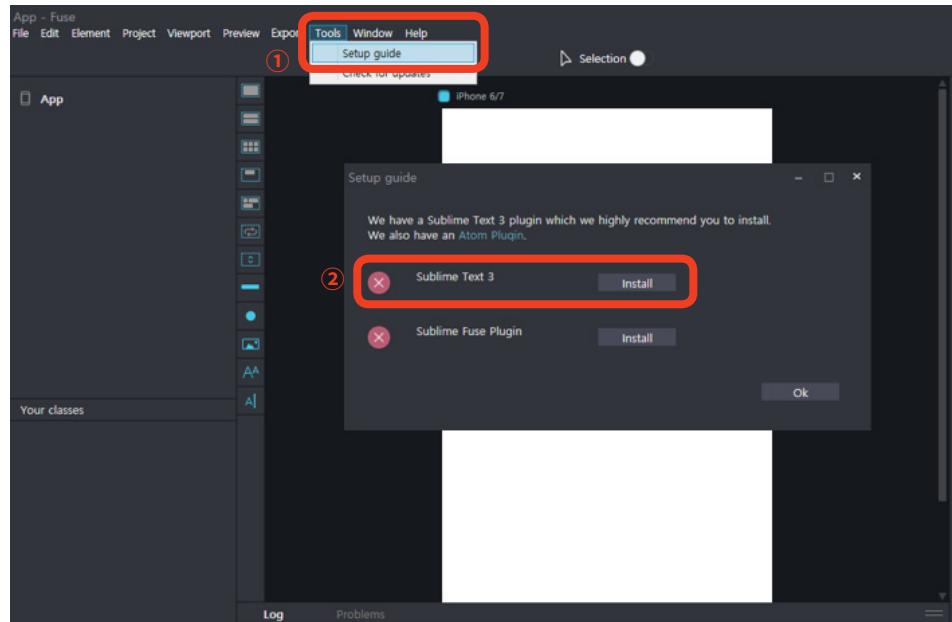


8 프로젝트가 생성 되고 ①과 같이 Fuse Package 들이 자동으로 설치된다.

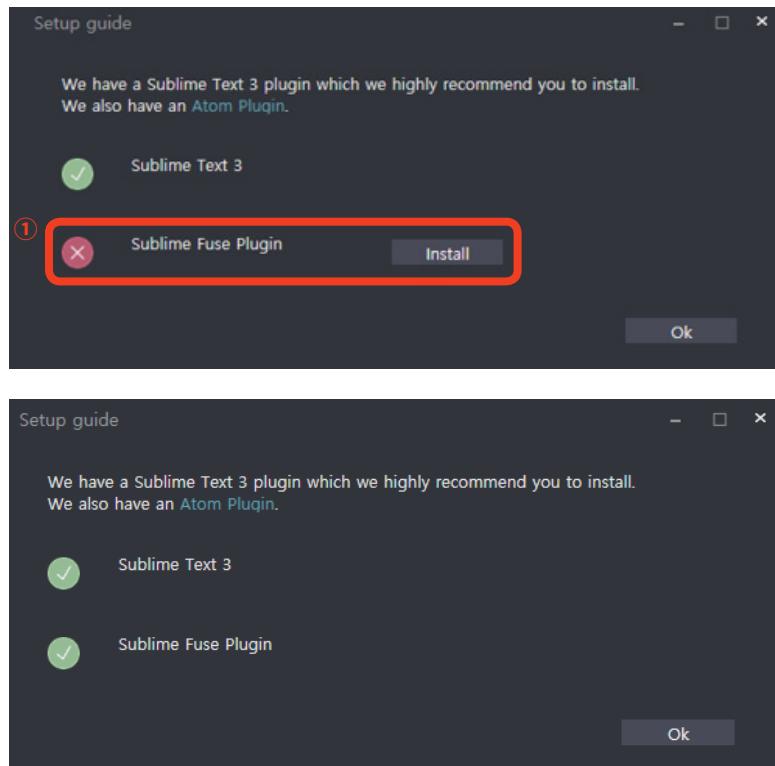
※ Fuse Package 설치는 PC사양 및 인터넷 속도에 따라 시간이 오래 걸릴 수 있으니 설치가 완료 될 때까지 기다려야 한다.



- 9 ① “Fuse Studio > Tools > Setup guide”를 클릭하여 서브라임 텍스트(Sublime Text)가 설치되지 않았다면 설치한다. (이미 설치 되었다면 다음 단계로 넘어간다)



- 10 ① Sublime Fuse Plugin 설치를 위해 “Install”을 클릭한다.



11 안드로이드 Preview 및 Export를 위해선 안드로이드 개발 환경이 필요하다.

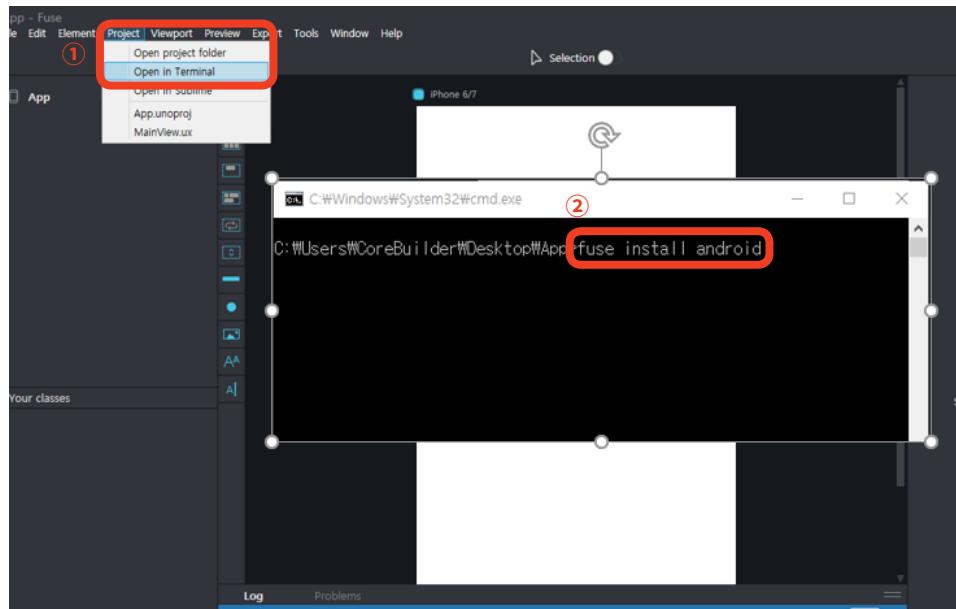
① “Fuse Studio > Project > Open in Terminal”을 클릭하여 명령 프롬프

트를 실행한다. (Mac OS는 Terminal 실행)

② “Fuse install android”를 입력하고 Enter를 입력한다.

※ 안드로이드 개발환경 설치는 PC사양 및 인터넷 속도에 따라 시간이 오

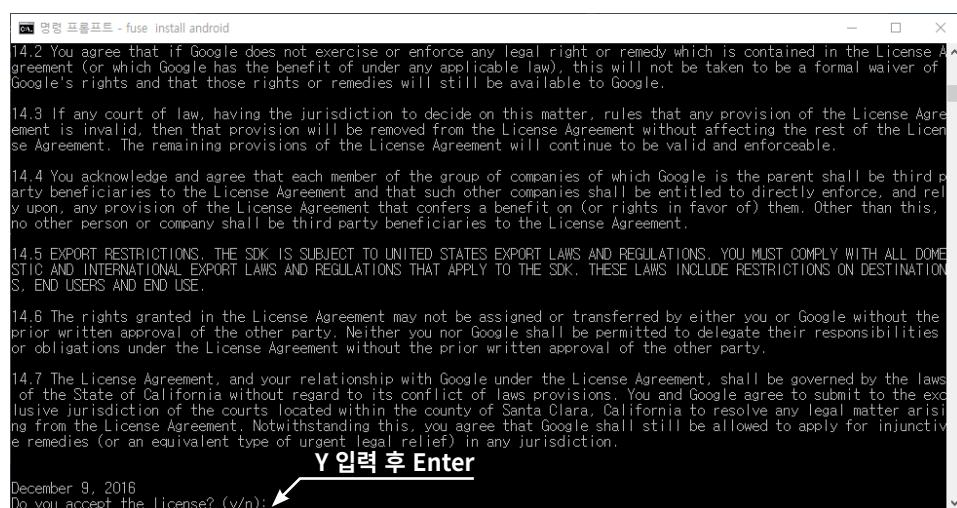
래 걸릴 수 있으니 설치가 완료 될 때까지 기다려야 한다.



```
C:\ 윤영 프롬프트 - fuse install android
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\ian>fuse install android
Fuse 1.0.9 (build 13739)
# Starting android installer
Java Development Kit wasn't found or an incompatible version of Java Development Kit found.
If you have an existing version of Java Development Kit, please specify its path here. If not, just press Enter to continue:
Ant wasn't found or an incompatible version of Ant found.
If you have an existing version of Ant, please specify its path here. If not, just press Enter to continue:
Android SDK wasn't found or an incompatible version of Android SDK found.
If you have an existing version of Android SDK, please specify its path here. If not, just press Enter to continue:
```

Three arrows point from the text "Enter 입력" to three separate lines of the command output, indicating where the user should press Enter to continue the process.



01.4

Hello, World! 퓨즈 앱을 만들기

2장에서 언급한 “Hello, World” 앱을 만드는 과정을 살펴보면서 퓨즈의 기본 사용법을 익혀 보자.

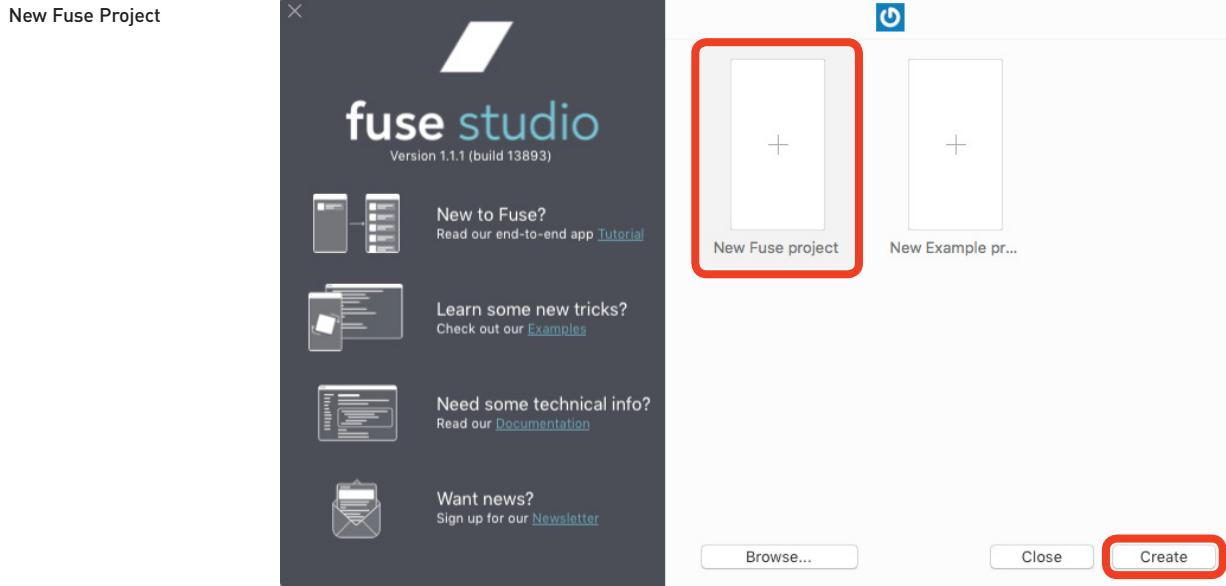
1. “Hello, World” 프로젝트 생성하기

퓨즈로 앱을 만들기 위해서는 처음에 프로젝트를 생성해야 한다. 모든 퓨즈 프로젝트는 아래와 같이 최소 2개의 파일이 있어야 한다.

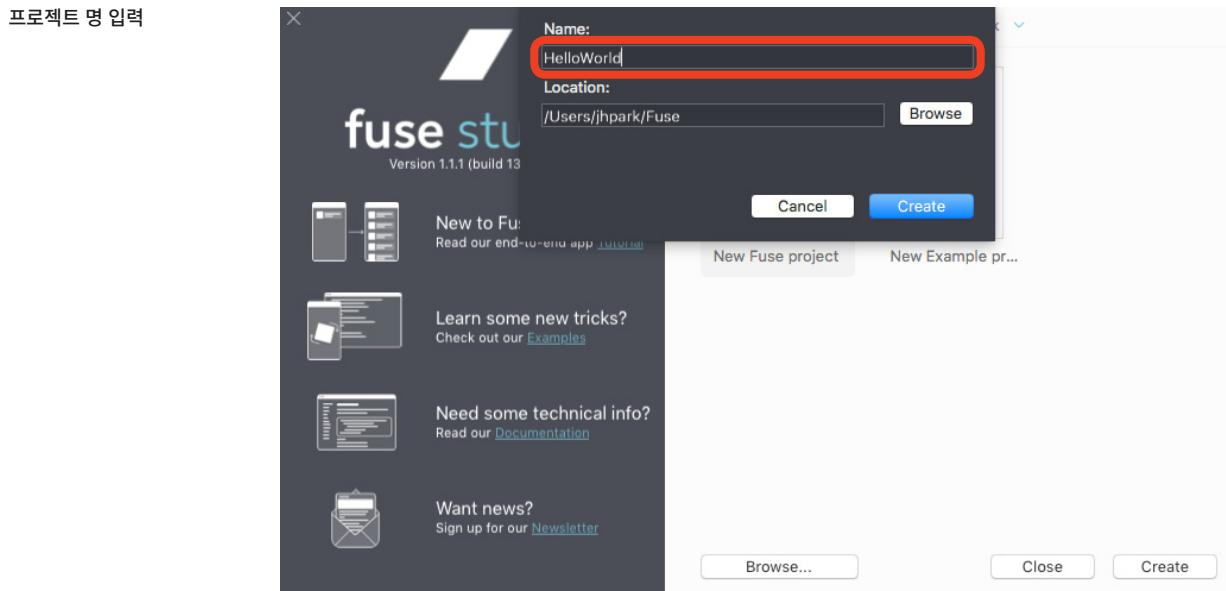
- [프로젝트이름].unoproj: 앱의 아이콘, 화면 방향 등 앱의 기본 정보를 포함
- MainView.ux: 앱의 메인 UX 파일. 화면에 표시되어야 하는 핵심 UI 컴포넌트와 레이아웃을 정의

퓨즈 프로젝트는 unoproj 파일과 MainView.ux가 있는 디렉토리를 의미한다. 퓨즈 프로젝트는 보통 퓨즈 실행 명령어를 사용해서 자동으로 만들지만 프로젝트 디렉토리를 직접 만들고 그 안에 unoproj, MainView.ux 파일을 수작업으로 만들어도 된다.

먼저 퓨즈 프로그램을 사용해서 HelloWorld 프로젝트를 만들어보자. 설치한 퓨즈 프로그램을 실행(명령 프롬프트/터미널에서 fuse를 타이핑 후 실행)한 후, 나타난 대시보드에서 “New Fuse Project”를 선택하고 “Create” 버튼을 누른다.

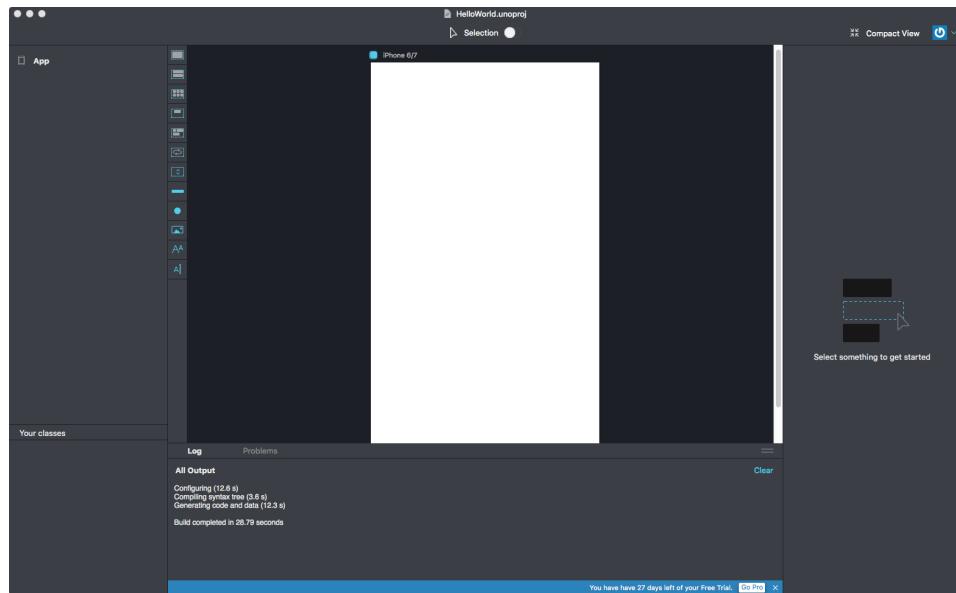


그 다음, 프로젝트를 생성할 디렉토리와 “HelloWorld”라는 프로젝트 이름을 입력하고 “Create” 버튼을 누른다.



그러면 아래와 같이 퓨즈 스튜디오가 실행이 되고 빈 화면이 나타난다. (독자는 퓨즈 프로페셔널 버전에 가입해서 퓨즈 스튜디오에 로그인했다고 가정한다.)

퓨즈 스튜디오



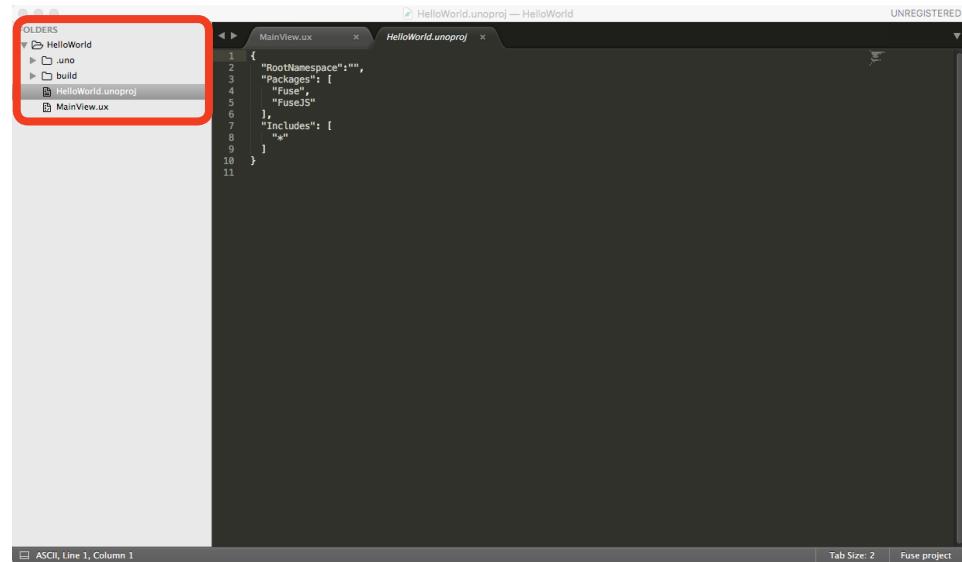
자동 생성된 HelloWorld 프로젝트 디렉토리에는 HelloWorld.unoproj 파일과 MainView.ux가 자동으로 만들어진다.

2. “Hello, World” 앱의 소스코드 만들기

앞에서 설명했던 “Hello, World” 앱의 코드를 구현해보자. 텍스트 편집기를 사용해서 UX 파일과 로직을 담당하는 자바스크립트 코드를 타이핑하자. 본 튜토리얼에서는 서브라임 텍스트를 사용한다.

퓨즈 스튜디오를 실행한 후, “Project > Open In Sublime”을 선택하면 바로 해당 프로젝트의 코드를 편집할 수 있게 설정된 서브라임 텍스트가 실행된다. 또는 직접 프로젝트 디렉토리를 드래그 앤 드롭 방식으로 서브라임 텍스트 안에 포함한다.

서브라임 텍스트에서
HelloWorld 프로젝트
디렉토리를 메인 디렉토리로
설정하기



MainView.ux 파일을 열면 처음에는 <App></App> 밖에 없다. 여기에 아래의 코드를 타이핑해서 입력한다.

MainView.ux

```
<App>  
  <JavaScript>  
    var Observable = require('FuseJS/Observable');  
    var buttonText = Observable('Button');  
    var clickCount = 0;  
  
    function onClick() {  
      clickCount += 1;  
      buttonText.value = 'Clicks: ' + clickCount;  
    }  
  
    module.exports = {  
      buttonText: buttonText,  
      onClick: onClick  
    }  
  </JavaScript>  
  
  <StackPanel>  
    <Text FontSize="30">Hello, world!</Text>  
    <Slider />  
    <Button Text="{buttonText}" Clicked="{onClick}" />  
    <Switch Alignment="Left" />  
  </StackPanel>  
</App>
```

HelloWorld 프로젝트의
MainView.ux

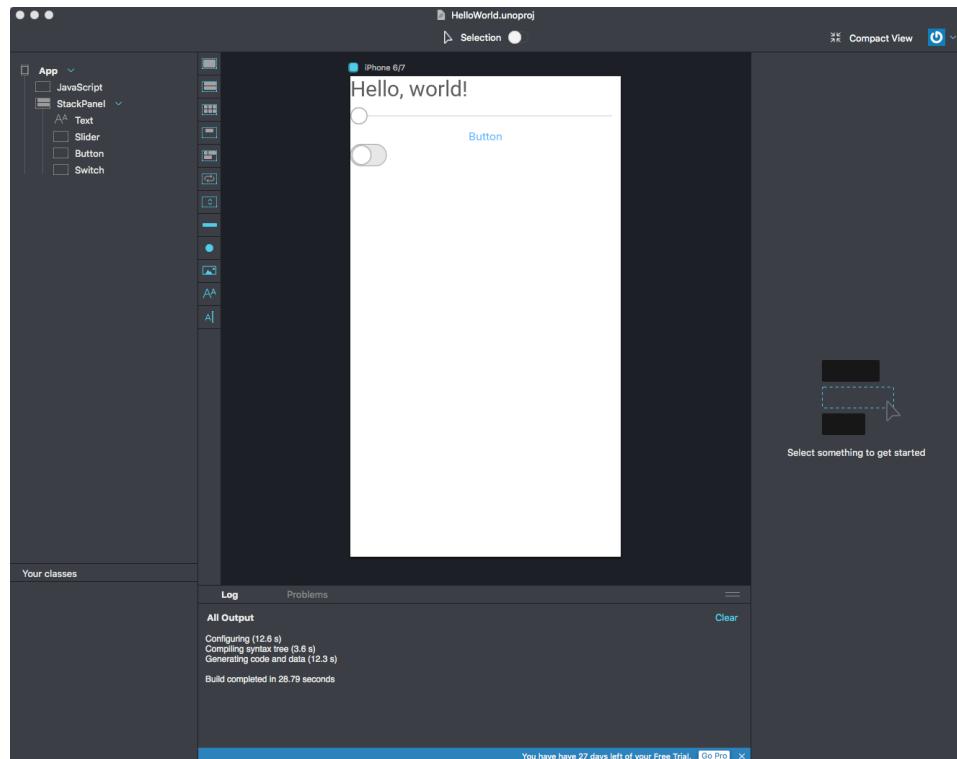
```
<App>
<JavaScript>
var Observable = require('FuseJS/Observable');
var buttonText = Observable("Button");
var clickCount = 0;

function onClick() {
    clickCount += 1;
    buttonText.value = 'Clicks: ' + clickCount;
}

module.exports = {
    buttonText: buttonText,
    onClick: onClick
}
</JavaScript>
<StackPanel>
<Text FontSize="30">Hello, world!</Text>
<Slider />
<Button Text="{buttonText}" Clicked="{onClick}" />
<Switch Alignment="Left" />
</StackPanel>
</App>
```

서브라임 텍스트에서 MainView.ux 파일을 열고 코드를 수정하고 저장하면 변경 내용이 퓨즈 스튜디오 화면에 바로 나오는 것을 알 수 있다.

퓨즈 스튜디오 내의
HelloWorld 앱 화면



퓨즈 스튜디오는 기본적으로 iPhone 6/7 해상도로 프리뷰 화면을 보여준다. 물론 프리뷰 화면의 해상도는 변경할 수 있다.

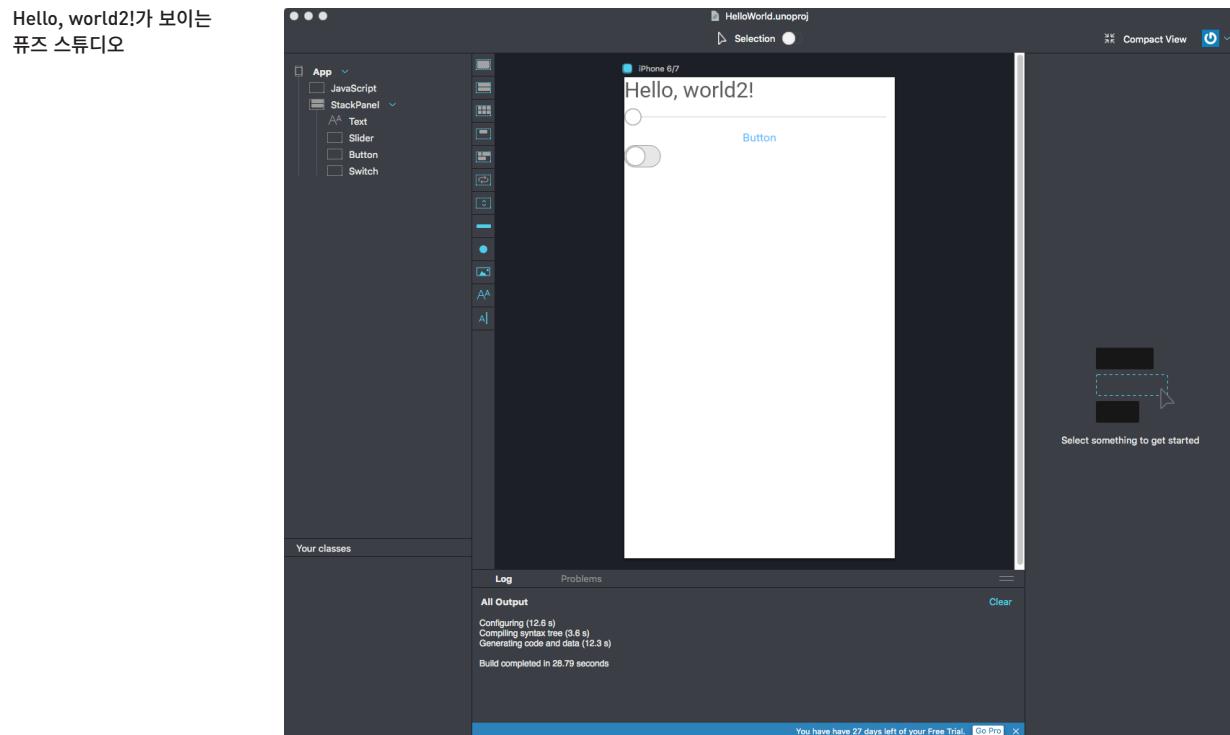
3. 앱을 컴퓨터에서 실시간으로 프리뷰하기

서브라임 텍스트에서 MainView.ux 파일을 열어 코드를 수정하고 저장하면 변경 내용이 퓨즈 스튜디오 화면에 바로 나오는 것을 알 수 있다. 이 기능을 프리뷰(Preview)라고 한다.

MainView.ux의 <Text FontSize="30">Hello, world!</Text>의 Hello, world!를 Hello, world2!로 변경한 후, 저장하면 퓨즈 스튜디오의 프리뷰 화면이 Hello, world2!로 즉시 변경된 것을 알 수 있다.

Hello, world!를
Hello, world2!로 변경하기

The screenshot shows the Sublime Text interface with the MainView.ux file open. The code has been modified from "Hello, world!" to "Hello, world2!". The file path is "MainView.ux — HelloWorld". The status bar at the bottom indicates "ASCII, Line 18, Column 41".



이 기능은 소스코드를 타이핑할 때 모바일 기기에서 동작할 때의 화면을 즉시 확인할 수 있어 개발을 빨리 할 수 있게 도와준다.

4. 앱을 모바일 기기에 설치하고 실시간으로 프리뷰하기

1) iOS 기기에 설치하기

앱을 모바일 기기에 설치한 후, 소스코드를 변경하면 모바일 기기의 앱이 즉시 변경되어 결과를 확인할 수도 있다.

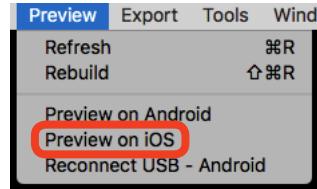
아래 명령을 사용하면 퓨즈 SDK는 HelloWorld 프로젝트를 iOS에 설치 가능한 앱으로 만들기 위해 C++, Objective-C 등으로 구성된 Xcode 프로젝트를 자동으로 만들어 준다.

가. 퓨즈 프로젝트의 iOS 프리뷰 컴파일 방법

iOS 프리뷰 컴파일 방법은 아래 3가지 중 편한 것을 선택하면 된다.

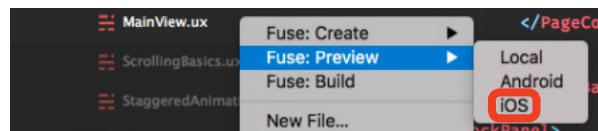
- 퓨즈 스튜디오 내의 “Preview > Preview on iOS” 메뉴를 선택한다.
-

퓨즈 스튜디오의
iOS 프리뷰 실행



- 서브라임 텍스트에서 오른쪽 마우스 버튼을 누르면 나오는 컨텍스트 메뉴를 사용해도 된다.
-

서브라임 텍스트의
iOS 프리뷰 실행



- 또는 명령 프롬프트나 터미널 프로그램을 실행하여 해당 프로젝트의 루트 디렉토리로 이동한 후 “fuse preview -t=iOS” 명령어를 실행한다.

서브라임 텍스트의
iOS 프리뷰 실행

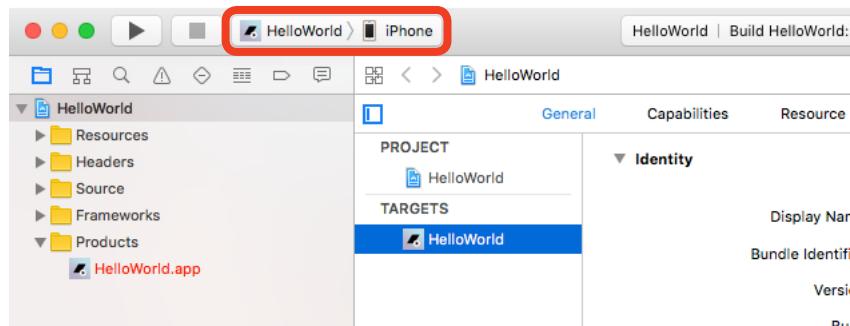

```

[[Fuse] HelloWorld $ ]
[[Fuse] HelloWorld $ ls -al
total 16
drwxr-xr-x  6 ian  staff  204 Aug 31 16:00 .
drwxr-xr-x  31 ian  staff  1054 Aug 31 16:00 ..
drwxr-xr-x  4 ian  staff  136 Aug 31 16:00 .uno
-rw-r--r--  1 ian  staff  101 Aug 31 16:00 HelloWorld.unoproj
-rw-r--r--  1 ian  staff  13 Aug 31 16:00 MainView.ux
drwxr-xr-x  4 ian  staff  136 Aug 31 16:01 build
[[Fuse] HelloWorld $ fuse preview -t=iOS
Fuse 1.2.0 (build 13961)
Configuring (0.7 s)
Compiling syntax tree (2.5 s)
Generating code and data]

```

나. Xcode로 앱을 iOS 기기에 설치하기

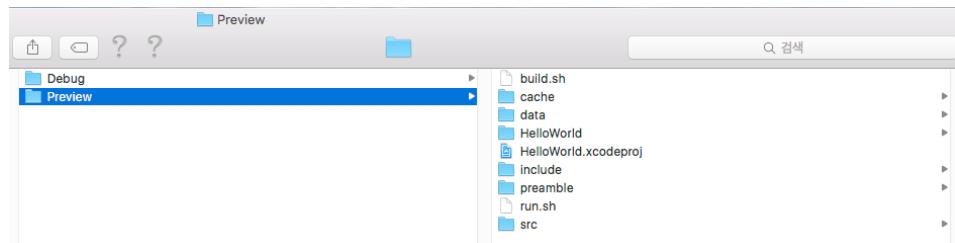
위의 방법이 성공하면 Xcode에서 인증서를 설정한 후, 설치 버튼을 눌러서 iOS 기기에 프리뷰 앱을 설치한다. 프리뷰 앱이 설치되면 서브라임 텍스트에서 코드를 변경하면 바로 앱에 반영되어 보이게 된다. (자세한 내용은 8부를 참고)

Xcode에서 프리뷰 앱을
설치하는 화면

다. 자동 생성된 Xcode 프로젝트 확인하기

HelloWorld/build/iOS/Preview 디렉토리가 위 명령을 통해 자동 생성된 Xcode 프로젝트의 루트 디렉토리이다. Xcode는 이 루트 디렉토리 내에 포함된 소스코드 프로젝트를 통합 관리한다.

자동 생성된
iOS 프리뷰 용
Xcode 프로젝트



2) 안드로이드 기기에 설치하기

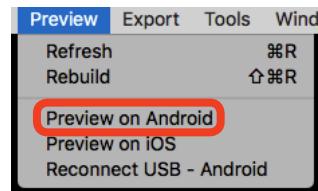
아래 명령을 사용하면 퓨즈 SDK는 HelloWorld 프로젝트를 안드로이드 기기에 설치 가능한 앱으로 만들기 위해 C++, Java 등으로 구성된 안드로이드 스튜디오용 프로젝트를 자동으로 만들어 준다.

가. 퓨즈 프로젝트의 안드로이드 프리뷰 컴파일 방법

안드로이드 프리뷰 컴파일 방법은 아래 방법 중 편한 것을 선택하면 된다.

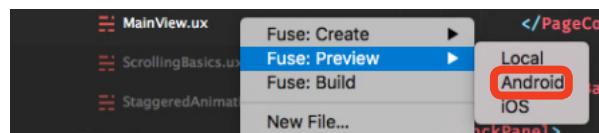
- 퓨즈 스튜디오 내의 “Preview > Preview on Android” 메뉴를 선택한다.

퓨즈 스튜디오의 안드로이드
프리뷰 실행



- 서브라임 텍스트의 컨텍스트 메뉴를 사용해서 설치한다.

서브라임 텍스트의
안드로이드 프리뷰 실행



5. 앱을 애플 앱스토어, 구글 플레이에 출시할 버전으로 만들어 모바일 기기에 설치하기

프리뷰 기능을 빼고 실제로 애플 앱스토어, 구글 플레이에서 배포할 앱을 만들어 모바일 기기에 설치해보자.

1) iOS 앱을 설치하기

프리뷰 모드 기능이 없는 순수 iOS 앱을 만들어서 테스트하고 싶으면 퓨즈 프로

젝트의 루트 디렉토리 안에서 아래의 명령을 실행한다. 소스코드를 변경하면 자동으로 앱에 반영하는 기능이 빠지고 일반 앱처럼 동작한다.

```
fuse build --target=iOS --run
```

실제로 애플 앱스토어에 출시할 앱을 만들 때 사용하는 명령어는 아래와 같다.

```
fuse build --target=iOS --configuration=Release
```

2) Android 앱을 설치하기

아래의 명령을 실행하면 순수 안드로이드 앱이 설치된다.

```
fuse build --target=Android --run
```

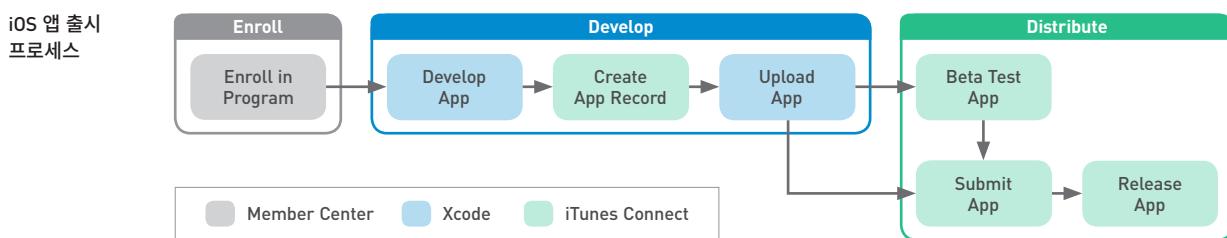
구글 플레이에 출시할 앱을 만들 때는 아래의 명령을 실행한다.

```
fuse build --target=Android --configuration=Release
```

6. 앱을 애플 앱스토어, 구글 플레이에 출시하기

1) iOS 어플을 출시하기

iOS 어플을 출시하기 위해서는 보통 아래의 프로세스를 따라야 한다. (출처: <https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html>)



가. 애플 개발자 프로그램 가입 (Enroll in Program)

<https://developer.apple.com/programs/>에 접속하여 애플 개발자 프로그램에 가입한다. 개인 또는 회사 자격으로 신청할 수 있다. 비용은 99달러이다.

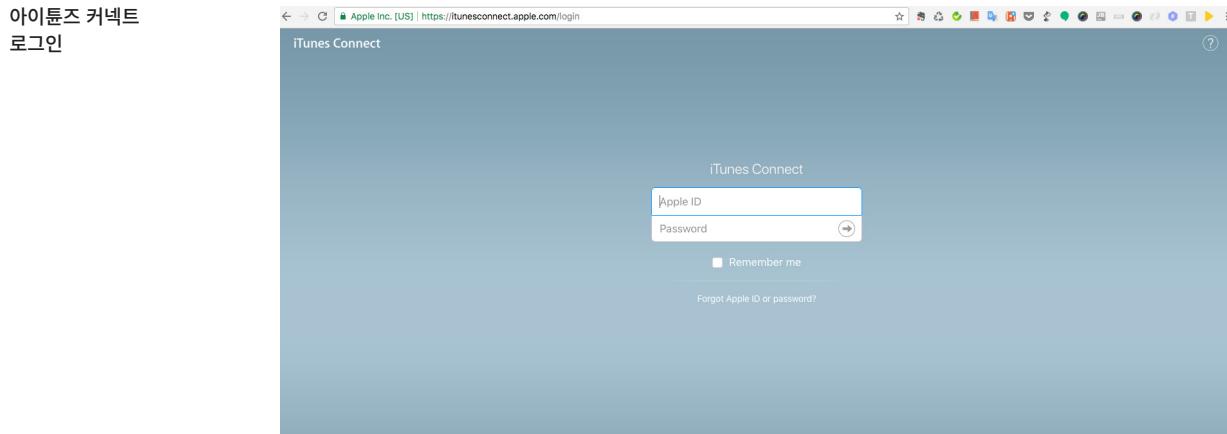
나. 어플을 개발하기 (Develop App)

일반적으로는 XCode를 사용하여 Objective-C 또는 Swift 언어로 어플을 개발

한다. 하지만 퓨즈로 개발할 때는 Objective-C, C++로 구성된 XCode 용 프로젝트가 자동으로 생성되기 때문에 본 단계는 필요 없다.

다. 아이튠즈 커넥트에 로그인 한 후 출시할 어플 정보를 등록하기

<https://itunesconnect.apple.com/login>에 접속한 후 애플 개발자 프로그램에 등록한 계정으로 로그인한다.



iTunes Store, App Store 및 iBooks Store의 콘텐츠를
손쉽게 관리하실 수 있습니다.

My Apps 메뉴를 클릭한 후 어플의 이름, 가격 등 각종 메타 정보를 입력한다.

The screenshot shows the 'App Information' section of the iTunes Connect 'App Store' tab. On the left, there's a sidebar with 'VERSION OR PLATFORM' and a '+' button. The main area has tabs for 'App Information' (which is selected), 'Pricing and Availability', and 'Localizable Information'. Under 'App Information', there are sections for 'General Information' (with fields for Bundle ID, Category, Primary Language, Secondary Language, License Agreement, and Rating), 'Localizable Information' (with Name and Privacy Policy URL fields), and 'General Information' (with fields for SKU, Your Bundle ID, Apple ID, and Category). A note at the top says 'This information is used for all platforms of this app. Any changes will be released with your next app version.'

라. 어플을 업로드하기(Upload App)

Xcode에서 빌드한 어플을 아이튠즈 커넥트에 업로드한다.

마. 어플을 심사 요청하기(Submit App)

애플의 앱스토어에 어플을 게재하려면 애플 측에 심사를 요청해야 한다. 애플의 직원이 심사요청 받은 어플을 테스트해서 문제가 없으면 출시를 허락한다. 보통 3일 ~ 1주일 정도의 시간이 걸린다.

바. 어플을 출시하기(Release App)

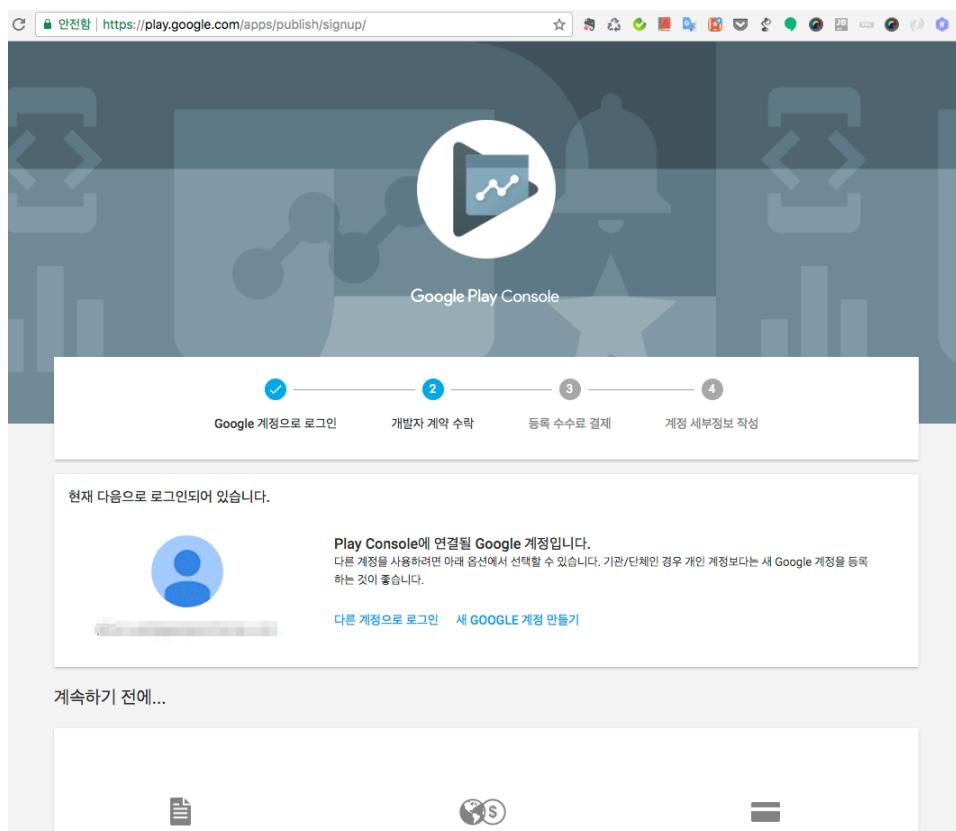
심사가 통과되면 사용자가 다운로드할 수 있도록 바로 릴리즈하거나 아니면 나중에 필요한 시점에 출시할 수 있다.

2) Android 어플을 출시하기

가. 구글 플레이 개발자 등록

<https://play.google.com/apps/publish/>에 접속하여 구글 플레이 개발자로 등록한다. 비용은 25달러이다.

구글 플레이
개발자 등록



나. 어플을 개발하기

일반적으로는 안드로이드 스튜디오를 사용하여 자바 언어로 어플을 개발한다. 하지만 퓨즈로 개발할 때는 자바, C++로 구성된 안드로이드 용 프로젝트가 자동으로 생성되기 때문에 본 단계는 필요 없다.

다. 구글 플레이 콘솔에 로그인 한 후 출시할 어플 정보를 등록하기

<https://play.google.com/apps/publish/>에 접속한 후 구글 플레이 개발자 프로그램에 등록한 계정으로 로그인한다. “모든 애플리케이션 > 애플리케이션 만들기” 메뉴를 클릭한 후 어플의 이름, 가격 등 각종 메타 정보를 입력한다.

라. 어플을 업로드하기

안드로이드 스튜디오 또는 안드로이드 SDK를 사용하여 빌드한 어플 APK 파일을 구글 플레이 콘솔에서 업로드한다.

마. 어플을 심사 요청하기

구글 플레이에 어플을 게재하려면 심사를 요청해야 한다. 보통 몇 시간, 최대 하루 이내에 심사가 완료된다.

바. 어플을 출시하기

심사가 통과되면 “출시 확인”을 클릭하여 사용자가 다운로드할 수 있도록 릴리즈한다. 또는 특정 시점에 출시되도록 예약할 수 있다.

02

하이킹 기록 앱, hikr 만들기

- 02.1 소개
- 02.2 프로젝트 구조
- 02.3 페이지 이해하기
- 02.4 화면 컴포넌트 이해하기
- 02.5 이벤트 처리 및 내비게이션 이해하기

02.1

소개

하이킹(도보 여행)을 기록하는 간단한 앱을 만들어보자. 퓨즈 프로젝트의 전반적인 개발 프로세스와 구조를 살펴보기에 좋다. 본 프로젝트는 퓨즈툴스의 공식 튜토리얼 예제(<https://www.fusetools.com/docs/tutorial/tutorial>)를 일부 수정(Backend.js 삭제)해서 만들어졌다.

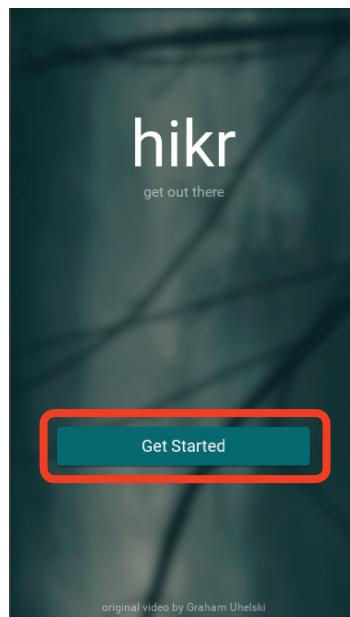
프로젝트를 단순화하기 위해 5개의 하이킹 기록이 이미 등록되어 있고, 각 기록의 상세 정보(하이킹 이름, 위치, 거리, 평점, 코멘트)를 수정하는 기능만을 구현한다. iOS, 안드로이드 모바일 기기에서 동작한다.

앱의 화면 기획은 아래와 같다.

가. 인트로 페이지

앱을 실행하면 “Get Started” 버튼이 보인다. 누르면 하이킹 리스트 페이지로 이동한다.

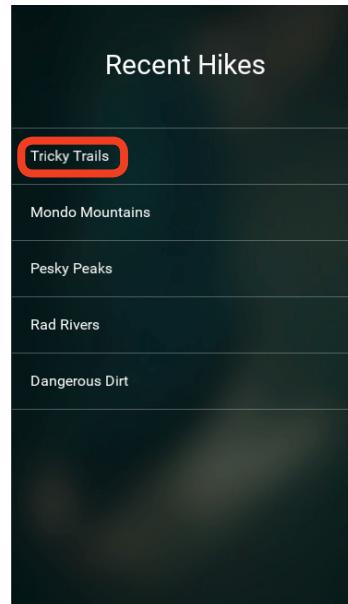
인트로 페이지



나. 하이킹 리스트 페이지

하이킹 리스트 페이지에는 총 5개의 하이킹 기록이 보이고, 선택하면 하이킹 정보를 수정할 수 있는 페이지로 이동한다.

하이킹 리스트 페이지



다. 하이킹 정보 수정 페이지

하이킹 이름(Name), 위치(Location), 거리(Distance), 평점(Rating), 코멘트(Comments)를 수정하고 저장 및 취소할 수 있다.

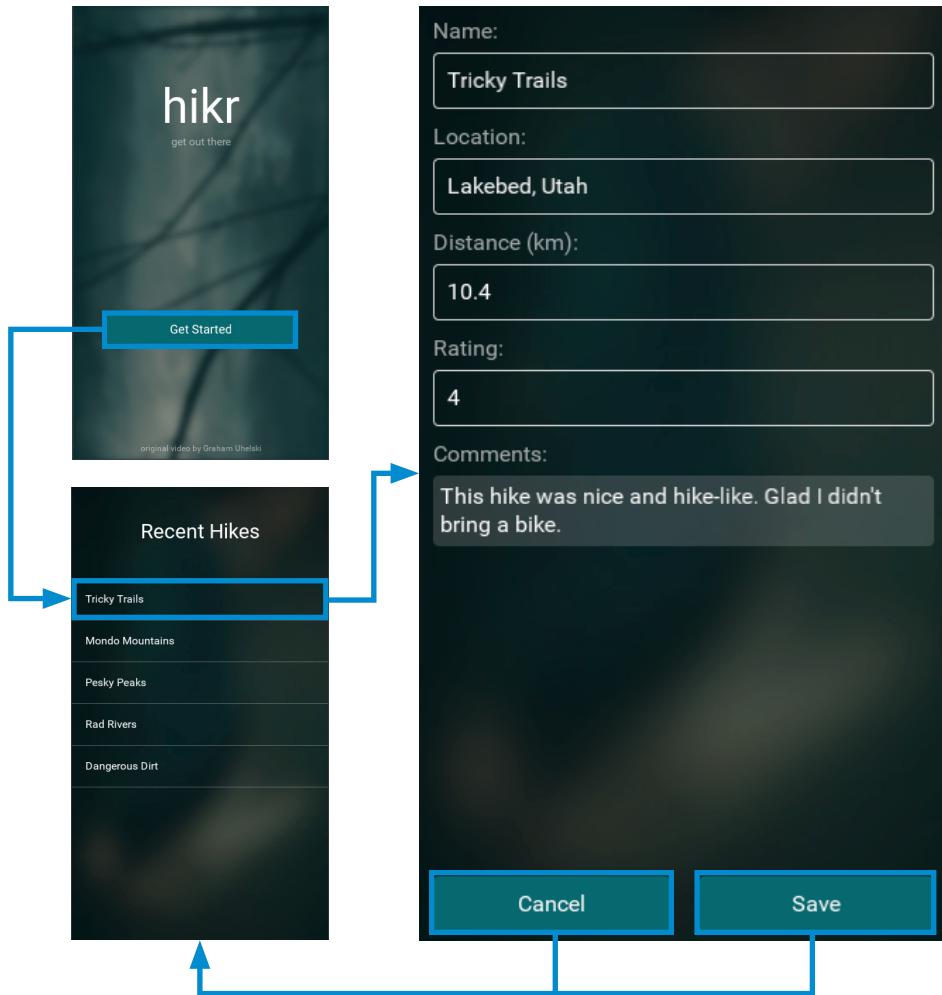
하이킹 정보 수정 페이지

Name:	Tricky Trails
Location:	Lakebed, Utah
Distance (km):	10.4
Rating:	4
Comments:	This hike was nice and hike-like. Glad I didn't bring a bike.

Cancel Save

각 페이지 간 화면 전환 관계도는 아래 그림과 같다.

hikr 화면 전환 관계도



02.2

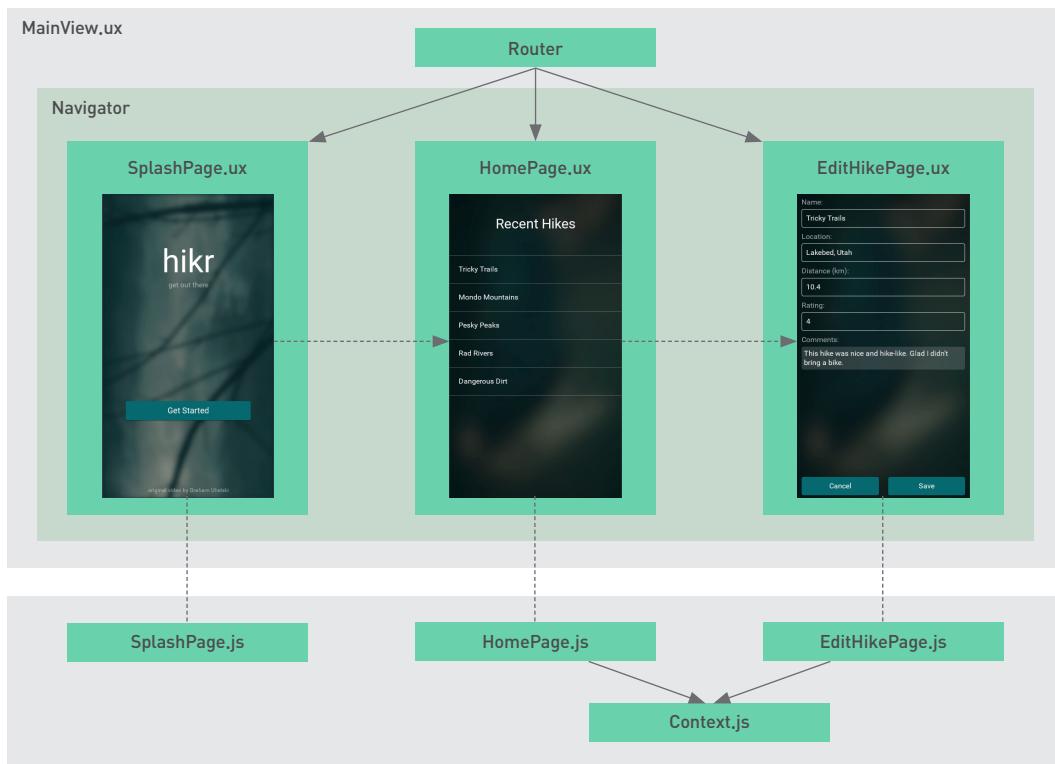
프로젝트 구조

본 프로젝트는 크게 4개의 디렉토리로 구성되어 있다.

- Pages: 앱의 각 화면을 나타내는 페이지와 관련 자바스크립트 파일
- Components: 본 프로젝트에 맞게 커스터마이징된 버튼, 페이지, 텍스트 등 의 컴포넌트
- Modules: 각 UX 컴포넌트 또는 각 페이지 별 자바스크립트 파일에서 공통적으로 사용되는 자바스크립트 모듈
- Assets: 이미지, 동영상 파일

hikr 앱의 시스템 아키텍처는 다음과 같다.

hikr 앱의 시스템 아키텍처



MainView.ux는 3개의 화면 페이지인 SplashPage, HomePage, EditHikePage를 Navigator 안에 포함하고 있다. 다른 페이지로 이동하기 위해서는 Router를 사용한다. 각 화면 페이지는 버튼 클릭 등의 이벤트를 처리하는 자바스크립트 파일인 SplashPage.js, HomePage.js, EditHikePage.js와 연결되어 있다. HomePage.js와 EditHikePage.js는 하이킹 정보를 읽거나 수정할 때 사용하는 Context.js 모듈을 공통적으로 사용한다.

프로젝트의 디렉토리 및 파일의 상세 정보는 아래와 같다.

- hikr 루트 디렉토리
 - MainView.ux: 메인 화면 UX 파일
 - hikr.unoproj: hikr 프로젝트 파일
 - Pages 디렉토리: 화면을 구성하는 각 페이지 ux와 관련 파일을 저장하는 디렉토리
 - SplashPage.ux: 앱 실행 후 처음 보여지는 페이지
 - SplashPage.js: SplashPage 화면의 “Get Started” 버튼을 눌렀을 때 동작하는 자바스크립트 코드 구현
 - HomePage.ux: 5개의 하이킹 기록 이름을 보여주는 리스트 페이지
 - HomePage.js: 하이킹 기록 이름을 눌렀을 때 동작하는 자바스크립트 코드 구현
 - EditHikePage.ux: 하이킹 기록의 이름, 위치, 거리, 평점, 코멘트 등을 수정할 수 있는 페이지
 - EditHikePage.js: 수정된 하이킹 기록을 저장하거나 취소하는 자바스크립트 코드 구현
 - Components 디렉토리: 화면을 구성하는 커스터마이징된 컴포넌트를 정의하는 디렉토리
 - hikr.Button.ux: SplashPage, EditHikePage에서 사용되는 특별한 버튼 컴포넌트
 - hikr.Page.ux: SplashPage, HomePage, EditHikePage.js에서 사용되는 배경 이미지가 있는 페이지 컴포넌트
 - hikr.Text.ux: 흰색 글자로 표시되는 텍스트 컴포넌트
 - hikr.TextBox.ux: 글자와 커서가 흰색인 텍스트박스 컴포넌트
 - hikr.TextView.ux: 글자와 커서가 흰색이며 배경색이 있는, 텍스트 입력이 가능한 텍스트뷰 컴포넌트

- Modules 디렉토리: 여러 자바스크립트 코드 파일에서 공통적으로 사용되는 자바스크립트 모듈을 저장하는 디렉토리
 - Context.js: 하이킹 기록 정보를 저장하고 있으며 새로운 값으로 수정하는 기능을 제공하는 자바스크립트 구현

02.3

페이지 이해하기

앱의 동작과정을 이해하기 위해 제일 먼저 분석해야 하는 파일은 MainView.ux이다. 이 곳에서 메인 화면의 내용이 결정된다.

1. MainView.ux 해설

MainView.ux

```
<App Background="#022328">
  <iOS.StatusBarConfig Style="Light" />
  <Android.StatusBarConfig Color="#022328" />

  <Router ux:Name="router" />

  <ClientPanel>
    <Navigator DefaultPath="splash">
      <SplashPage ux:Template="splash" router="router" />
      <HomePage ux:Template="home" router="router" />
      <EditHikePage ux:Template="editHike" router="router" />
    </Navigator>
  </ClientPanel>
</App>
```

```
<iOS.StatusBarConfig Style="Light" />
<Android.StatusBarConfig Color="#022328" />
```

iOS와 안드로이드에서 동작할 때 상태 바의 스타일과 색상을 정의하는 코드이다.

```
<Router ux:Name="router" />
```

SplashPage, HomePage, EditHikePage 등 3개의 페이지들이 내비게이션을 담당하는 Router 인스턴스를 생성해야 한다. 생성한 후 이름을 router로 명명하였다.

```
<ClientPanel>
  <Navigator DefaultPath="splash">
    <SplashPage ux:Template="splash" router="router" />
    <HomePage ux:Template="home" router="router" />
    <EditHikePage ux:Template="editHike" router="router" />
  </Navigator>
</ClientPanel>
```

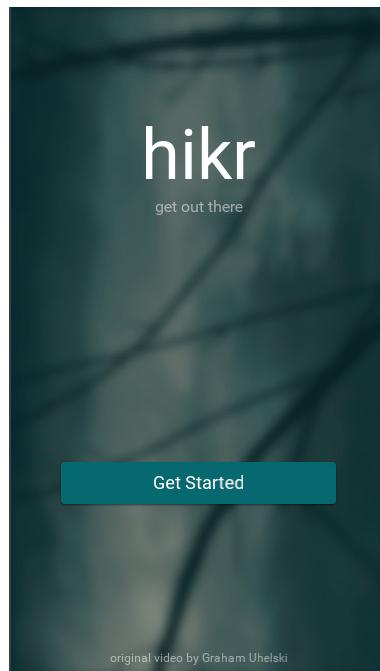
ClientPanel은 모바일 기기의 OS에서 제공하는 상, 하단의 바를 제외하는 공간

을 차지하는 Panel이다.

이 안에 Navigator가 존재하며 SplashPage, HomePage, EditHikePage를 하나의 Navigator 그룹에 묶는다. Navigator는 필요한 때에 각 페이지를 생성하고 라이프사이클을 관리한다. SplashPage, HomePage, EditHikePage는 각각 splash, home, editHike로 명명하고 각 페이지의 이동을 담당하는 Router의 이름은 router로 명명한다. 처음에 보여지는 기본 페이지는 “<Navigator DefaultPath=“splash”>”를 사용하여 SplashPage로 지정한다. SplashPage, HomePage, EditHikePage의 “ux:Template”的 의미는 각 페이지가 즉시 만들어지는 것이 아니라 추후 필요 시에 Navigator에 의해 동적으로 생성된다는 뜻이다.

2. SplashPage 해설

SplashPage.ux 화면



SplashPage.ux

```
<Page ux:Class="SplashPage">
  <Router ux:Dependency="router" />

  <JavaScript File="SplashPage.js" />

  <DockPanel ClipToBounds="true">
    <Video Layer="Background" File="../../Assets/nature.mp4" IsLooping="true"
          AutoPlay="true" StretchMode="UniformToFill" Opacity="0.5">
      <Blur Radius="4.75" />
    </Video>

    <hikr.Text Dock="Bottom" Margin="10" Opacity=".5"
              TextAlignment="Center" FontSize="12">
      original video by Graham Uhelski</hikr.Text>
```

```
<Grid RowCount="2">
    <StackPanel Alignment="VerticalCenter">
        <hikr.Text Alignment="HorizontalCenter" FontSize="70">
            hikr</hikr.Text>
        <hikr.Text Alignment="HorizontalCenter" Opacity=".5">
            get out there</hikr.Text>
    </StackPanel>

    <hikr.Button Text="Get Started" FontSize="18" Margin="50,0"
        Alignment="VerticalCenter" Clicked="{goToHomePage}" />
</Grid>
</DockPanel>
</Page>
```

앱을 실행하면 제일 먼저 보여지는 화면 페이지다.

```
<Page ux:Class="SplashPage">
```

내비게이터에 의해 전환되는 화면을 상징하는 페이지 컴포넌트를 사용하며, ux:Class로 SplashPage라는 클래스(Class)를 선언한다. 추후 MainView.ux 내의 Navigator에 의해 컴포넌트로 생성되어 화면에 모양이 표시된다.

```
<Router ux:Dependency="router" />
```

MainView.ux에서 만들어진 Router 객체(<Router ux:Name="router" />)를 받아서 연결한다. 이 Router 객체의 이름은 router이다. 하이킹 리스트 페이지인 HomePage로 이동할 때 SplashPage.js 안에서 사용된다.

```
<JavaScript File="SplashPage.js" />
```

SplashPage.js 파일 안의 자바스크립트를 실행한다. “Get Started” 버튼을 클릭했을 때 HomePage로 이동하는 자바스크립트 코드가 구현되어 있다.

```
<DockPanel ClipToBounds="true">
```

화면 레이아웃을 DockPanel로 지정한다. DockPanel은 화면 컴포넌트를 좌, 우, 위, 아래, 전체 등으로 배치하게 해준다.

```
<Video Layer="Background" File="../Assets/nature.mp4" IsLooping="true"
    AutoPlay="true" StretchMode="UniformToFill" Opacity="0.5">
    <Blur Radius="4.75" />
</Video>
```

화면의 배경은 nature.mp4 동영상이 실행되어 표시된다. 설정 값에 따라 무한 루프(IsLooping=true)로 실행되며 투명도는 반(Opacity="0.5") 정도, 흐릿하게 불러 처리 된다.

```
<hikr.Text Dock="Bottom" Margin="10" Opacity=".5" TextAlignment="Center"
    FontSize="12">original video by Graham Uhelski</hikr.Text>
```

hikr.Text 컴포넌트의 모양대로 화면 밑바닥(Dock="Bottom") 중앙(Text Alignment="Center")에 original video by Graham Uhelski를 표시한다.
(hikr.Text 컴포넌트는 추후 설명)

```
<Grid RowCount="2">
    <StackPanel Alignment="VerticalCenter">
        <hikr.Text Alignment="HorizontalCenter" FontSize="70">
            hikr</hikr.Text>
        <hikr.Text Alignment="HorizontalCenter" Opacity=".5">
            get out there</hikr.Text>
    </StackPanel>

    <hikr.Button Text="Get Started" FontSize="18" Margin="50,0"
        Alignment="VerticalCenter" Clicked="{goToHomePage}" />
</Grid>
```

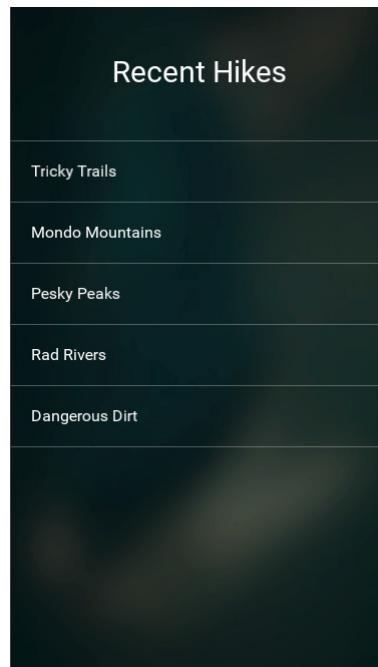
아래처럼 2개의 가로 행(Grid RowCount="2")을 만들고 각각 StackPanel, hikr.Button을 넣는다. StackPanel은 위에서 아래로 hikr.Text인 hikr, get out there를 중앙에 배치한다.

StackPanel
- hikr.Text : hikr
- hikr.Text : get out there

hikr.Button
- Get Started

3. HomePage 해설

HomePage.ux 화면



HomePage.ux

```
<hikr.Page ux:Class="HomePage">
<Router ux:Dependency="router" />

<JavaScript File="HomePage.js" />

<DockPanel>
    <hikr.Text FontSize="30" TextAlignment="Center" Dock="Top"
        Margin="0,50">Recent Hikes</hikr.Text>

    <ScrollView>
        <StackPanel>
            <Rectangle ux:Class="Separator" Height="1" Fill="#fff4" />

            <Each Items="{hikes}">
                <Separator />

                <Panel HitTestMode="LocalBoundsAndChildren" Clicked="{goToHike}">
                    <hikr.Text Value="{name}" Margin="20" />

                    <WhilePressed>
                        <Scale Factor=".95" Duration=".08" Easing="QuadraticOut" />
                    </WhilePressed>
                </Panel>
            </Each>

            <Separator />
        </StackPanel>
    </ScrollView>
</DockPanel>
</hikr.Page>
```

```
<hikr.Page ux:Class="HomePage">
```

위의 SplashPage.ux와 달리 컴포넌트 딕렉토리에 들어있는 커스터마이징된 hikr.Page 컴포넌트로 페이지를 선언한다. hikr.Page는 아래의 “3장. 화면 컴포넌트 이해하기”에 나와 있는 것처럼 배경 이미지가 있는 페이지이다. EditHikePage.ux에서도 재사용된다.

```
<Router ux:Dependency="router" />
```

MainView.ux에서 만들어진 Router 객체(`<Router ux:Name="router" />`)를 받아서 연결한다. 사용자가 화면의 하이킹 기록 이름을 클릭하면 하이킹 정보 수정 페이지인 EditHikePage로 이동할 때 HomePage.js 안에서 사용된다.

```
<JavaScript File="HomePage.js" />
```

HomePage.js 파일 안의 자바스크립트를 실행한다. 하이킹 이름을 클릭했을 때 실행되는 자바스크립트 코드가 구현되어 있다.

```
<DockPanel>
    <hikr.Text FontSize="30" TextAlignment="Center" Dock="Top"
        Margin="0,50">Recent Hikes</hikr.Text>
```

화면 레이아웃을 DockPanel로 지정한다. 그리고 Recent Hikes란 텍스트를 Top, 즉 최상단 중앙에 배치한다.

```
<ScrollView>
  <StackPanel>
```

Recent Hikes 텍스트 밑에 스크롤이 되는 뷰를 추가하고, 그 안에 위에서 아래로 컴포넌트를 배치할 수 있는 StackPanel을 만든다.

```
<Rectangle ux:Class="Separator" Height="1" Fill="#ffff4" />
```

각 하이킹 기록 이름의 사이에 들어갈 수평선 모양의 사각형 컴포넌트를 Separator 클래스로 선언한다. 클래스로 선언되었기 때문에 지금 당장 컴포넌트가 생성되지 않고, ux 파일 안에 <Separator ...>라고 명시된 지점에서 컴포넌트가 생성된다.

```
<Each Items="{hikes}">
  <Separator />
```

HomePage.js의 자바스크립트 코드가 실행되면 하이킹 기록 데이터는 {hikes} 배열 객체 안에 아래의 내용으로 총 5개가 저장된다.

hikes 배열 객체 리스트

배열 원소 위치 (index)	객체 속성 (propertys)	값 (value)
0	id	0
	name	Tricky Trails
	location	Lakebed, Utah
	distance	10.4
	rating	4
	comments	This hike was nice and hike-like. Glad I didn't bring a bike.
1	id	1
	name	Mondo Mountains
	location	Black Hills, South Dakota
	distance	20.86
	rating	3
	comments	Not the best, but would probably do again. Note to self: don't forget the sandwiches next time.
2	id	2
	name	Pesky Peaks
	location	Bergenhagen, Norway
	distance	8.2
	rating	5
	comments	Short but SO sweet!!

3	id	3
	name	Rad Rivers
	location	Moriyama, Japan
	distance	12.3
	rating	4
	comments	Took my time with this one. Great view!
4	id	4
	name	Dangerous Dirt
	location	Cactus, Arizona
	distance	19.34
	rating	2
	comments	Too long, too hot. Also that snakebite wasn't very fun.

<Each Items="“{hikes}”> 문을 사용해서 hikes 배열 안에 저장된 원소 각각의 하이킹 이름(name)을 화면에 표시할 것이다.

```

<Panel HitTestMode="LocalBoundsAndChildren" Clicked="“{goToHike}”>
    <hikr.Text Value="“{name}” Margin="20" />

    <WhilePressed>
        <Scale Factor=".95" Duration=".08" Easing="QuadraticOut" />
    </WhilePressed>
</Panel>

```

<hikr.Text Value="“{name}” Margin="20" />에 따라 hikes 배열 객체 내의 각 원소의 name 속성 값은 Panel 안의 hikr.Text 컴포넌트 모양으로 표시된다. (하단의 어두운 배경색 참조)

hikes 배열 객체 리스트

배열 원소 위치 (index)	객체 속성 (propert)	값(value)
0	id	0
	name	Tricky Trails
	location	Lakebed, Utah
	distance	10.4
	rating	4
	comments	This hike was nice and hike-like. Glad I didn't bring a bike.
1	id	1
	name	Mondo Mountains
	location	Black Hills, South Dakota

	distance	20.86
	rating	3
	comments	Not the best, but would probably do again. Note to self: don't forget the sandwiches next time.
2	id	2
	name	Pesky Peaks
	location	Bergenhagen, Norway
	distance	8.2
	rating	5
	comments	Short but SO sweet!!
3	id	3
	name	Rad Rivers
	location	Moriyama, Japan
	distance	12.3
	rating	4
	comments	Took my time with this one. Great view!
4	id	4
	name	Dangerous Dirt
	location	Cactus, Arizona
	distance	19.34
	rating	2
	comments	Too long, too hot. Also that snakebite wasn't very fun.

Clicked="“{goToHike}”를 지정하여 하이킹 이름이 클릭되었을 때 HomePage.js 안에 정의된 goToHike 함수가 호출된다. 이 함수는 EditHikePage로 페이지를 이동하는 기능을 실행한다. 또한 WhilePressed, 즉 사용자가 하이킹 이름을 클릭 또는 눌렀을 때 텍스트가 작게 축소되는 애니메이션 효과를 넣는다. 각각의 하이킹 이름은 Tricky Trails, Mondo Mountains, Pesky Peaks, Rad Rivers, Dangerous Dirt 등이다.

```
<Separator />
</StackPanel>
</ScrollView>
</DockPanel>
</hikr.Page>
```

제일 마지막 하이킹 이름 링크에 <Separator />를 한 개 생성한다.

4. EditHikePage 해설

EditHikePage.ux



EditHikePage.ux

```
<hikr.Page ux:Class="EditHikePage">
    <Router ux:Dependency="router" />

    <JavaScript File="EditHikePage.js" />

    <DockPanel>
        <Grid ColumnCount="2" Dock="Bottom">
            <hikr.Button Text="Cancel" Clicked="{cancel}" />
            <hikr.Button Text="Save" Clicked="{save}" />
        </Grid>

        <ScrollView>
            <StackPanel ItemSpacing="10" Padding="10">
                <hikr.Text ux:Class="TitleText" Opacity=".6" Margin="0,0,0,5" />

                <StackPanel>
                    <TitleText>Name:</TitleText>
                    <hikr.TextBox Value="{name}" />
                </StackPanel>

                <StackPanel>
                    <TitleText>Location:</TitleText>
                    <hikr.TextBox Value="{location}" />
                </StackPanel>

                <StackPanel>
                    <TitleText>Distance (km):</TitleText>
                    <hikr.TextBox Value="{distance}" InputHint="Decimal" />
                </StackPanel>

                <StackPanel>
                    <TitleText>Rating:</TitleText>
                    <hikr.TextBox Value="{rating}" InputHint="Integer" />
                </StackPanel>

                <StackPanel>
                    <TitleText>Comments:</TitleText>
                </StackPanel>
            </StackPanel>
        </ScrollView>
    </DockPanel>

```

```
<hikr.TextView Value="{comments}" TextWrapping="Wrap" />
</StackPanel>
</StackPanel>
</ScrollView>
</DockPanel>
</hikr.Page>
```

```
<hikr.Page ux:Class="EditHikePage">
```

HomePage.ux와 동일하게 컴포넌트 디렉토리에 저장된, 커스터마이징된 hikr.Page 컴포넌트로 페이지를 선언한다. 배경 이미지가 HomePage.ux와 동일하다.

```
<Router ux:Dependency="router" />
```

MainView.ux에서 만들어진 Router 객체(<Router ux:Name="router" />)를 받아서 연결한다. 사용자가 “Cancel”, “Save” 버튼을 클릭하여 하이킹 리스트 페이지인 HomePage로 이동할 때 HomePage.js 안에서 사용된다.

```
<JavaScript File="EditHikePage.js" />
```

EditHikePage.js 파일 안의 자바스크립트를 실행한다. “Save” 버튼을 누르면 수정된 하이킹 정보를 Context.js 내의 자바스크립트 객체에 저장하고 HomePage로 이동하는 코드가 구현되어 있다. “Cancel” 버튼을 누르면 정보를 업데이트하지 않고 단지 HomePage로만 이동한다..

```
<DockPanel>
<Grid ColumnCount="2" Dock="Bottom">
<hikr.Button Text="Cancel" Clicked="{cancel}" />
<hikr.Button Text="Save" Clicked="{save}" />
</Grid>
```

화면 레이아웃을 DockPanel로 지정한다. 그리고 Cancel, Save 텍스트의 hikr.Button을 Bottom, 즉 최하단에 배치한다. <Grid ColumnCount="2" ... > 이므로 한 행(row)에 2개의 컬럼(column)을 정의하고 순서대로 Cancel, Save 버튼을 배치한다. Cancel 버튼을 누르면 EditHikePage.js 내의 cancel 함수, Save 버튼을 누르면 save 함수가 호출된다.

```
<ScrollView>
<StackPanel ItemSpacing="10" Padding="10">
<hikr.Text ux:Class="TitleText" Opacity=".6" Margin="0,0,0,5" />
```

스크롤이 되는 뷰를 추가하고, 그 안에 위에서 아래로 컴포넌트를 배치할 수 있는 StackPanel을 만든다. 그리고 TitleText 컴포넌트 클래스를 정의해서 각 수정 항목의 제목으로 사용한다.

사용자가 HomePage에서 Mondo Mountains 항목을 선택했다고 가정하자. 그러면 해당 하이킹 정보가 담긴 객체가 EditHikePage로 넘어오게 되며 그것의 속성을 UX 파일 안에서 {name}, {location}, {distance}, {rating}, {comments} 등으로 접근할 수 있다.

사용자가 선택한
“Mondo Mountains”
하이킹 정보 객체의 내용

객체 속성(property)	값 (value)
id	0
name	Tricky Trails
location	Lakebed, Utah
distance	10.4
rating	4
comments	This hike was nice and hike-like. Glad I didn't bring a bike.

```
<StackPanel>
    <TitleText>Name:</TitleText>
        <hikr.TextBox Value="{name}" />
</StackPanel>
```

Name: 텍스트를 화면에 표시하고 HomePage에서 클릭했을 때 넘겨받은 hike 객체의 name 속성 값도 화면에 보여준다. 본 예에서는 Tricky Trails이다.

```
<StackPanel>
    <TitleText>Location:</TitleText>
        <hikr.TextBox Value="{location}" />
</StackPanel>
```

Location: 텍스트를 화면에 표시하고 HomePage에서 클릭했을 때 넘겨받은 hike 객체의 location 속성 값도 화면에 보여준다. 본 예에서는 Lakebed, Utah이다.

```
<StackPanel>
    <TitleText>Distance (km):</TitleText>
        <hikr.TextBox Value="{distance}" InputHint="Decimal" />
</StackPanel>
```

Distance: 텍스트를 화면에 표시하고 HomePage에서 클릭했을 때 넘겨받은 hike 객체의 distance 속성 값도 화면에 보여준다. 모바일 앱에서 hikr.TextBox를 선택하면 InputHint="Decimal"로 지정했기 때문에 숫자를 입력할 수 있는 키보드가 실행된다. 본 예에서는 10.4이다.

```
<StackPanel>
    <TitleText>Rating:</TitleText>
        <hikr.TextBox Value="{rating}" InputHint="Integer" />
</StackPanel>
```

Rating: 텍스트를 화면에 표시하고 HomePage에서 클릭했을 때 넘겨받은 hike 객체의 rating 속성 값도 화면에 보여준다. 본 예에서는 4이다.

```
<StackPanel>
    <TitleText>Comments:</TitleText>
    <hikr.TextView Value="{comments}" TextWrapping="Wrap" />
</StackPanel>
```

Comments: 텍스트를 화면에 표시하고 HomePage에서 클릭했을 때 넘겨받은 hike 객체의 comments 속성 값도 화면에 보여준다. 본 예에서는 “This hike was nice and hike-like. Glad I didn’t bring a bike.”이다.

02.4

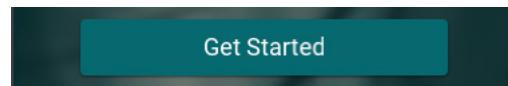
화면 컴포넌트 이해하기

hikr 프로젝트에서 커스터마이징되어 사용되는 컴포넌트를 살펴본다. 이들 모든 컴포넌트는 클래스(class)로 선언된다. 클래스로 선언된 컴포넌트는 선언 즉시 바로 생성되는 것이 아니라, 다른 컴포넌트에서 재사용될 때 생성된다는 사실을 기억해야 한다.

1. hikr.Button 컴포넌트

hikr.Button 컴포넌트는 위의 SplashPage의 “Get Started” 버튼, EditHike Page의 Cancel, Save 버튼을 생성할 때 사용하는 컴포넌트이다.

hikr에서 사용된 버튼



위 버튼은 배경색이 있는 사각형 모양이며 버튼을 클릭했을 때 작게 축소되는 애니메이션 효과가 공통적으로 있다. 이런 모양의 버튼을 직접 만들어보자

hikr.Button.ux

```
<Panel ux:Class="hikr.Button" Margin="10" Padding="10" FontSize="16">
    <string ux:Property="Text" />
    <float ux:Property="FontSize" />

    <Rectangle Layer="Background" Color="#125F63" CornerRadius="4">
        <DropShadow Angle="90" Distance="1" Spread="0.2" Size="2"
            Color="#00000060" />
    </Rectangle>

    <hikr.Text Value="{ReadProperty Text}" FontSize="{ReadProperty FontSize}"
        TextAlignment="Center" />
    <WhilePressed>
        <Scale Factor=".95" Duration=".08" Easing="QuadraticOut" />
    </WhilePressed>
</Panel>
```

```
<Panel ux:Class="hikr.Button" Margin="10" Padding="10" FontSize="16">
```

컴포넌트는 Panel 타입이며 클래스 이름을 hikr.Button으로 지정한다. 보통 컴포넌트의 클래스는 “{프로젝트 이름}.컴포넌트” 이름으로 명명한다. Margin, Padding, FontSize의 기본 값은 각각 10, 10, 16이다.

```
<string ux:Property="Text" />
<float ux:Property="FontSize" />
```

ux:Property를 사용하여 Text는 문자열(string), FontSize는 실수 값(float)으로 값을 넘겨 받아서 사용하겠다는 뜻이다. 이렇게 함으로써 hikr.Button을 생성할 때 버튼의 이름과 글자의 크기를 지정할 수 있게 된다.

```
<Rectangle Layer="Background" Color="#125F63" CornerRadius="4">
  <DropShadow Angle="90" Distance="1" Spread="0.2" Size="2"
    Color="#00000060" />
</Rectangle>
```

사각형을 만들어 패널의 배경으로 배치하고 색을 지정한다. 코너를 약간 둥글게 만든다. DropShadow를 사용해서 버튼 주위에 그림자 효과를 준다.

```
<hikr.Text Value="{ReadProperty Text}" FontSize="{ReadProperty FontSize}"
  TextAlignment="Center" />
```

패널 중앙에 hikr.Text 컴포넌트를 생성한다. hikr.Text는 아래에 나온 바대로 피 하얀색 Text이다. hikr.Text에 넘겨 받은 Text, FontSize를 ReadProperty, 즉 읽기만 가능하게 제한해서 지정하고 중앙에 배치한다.

```
<WhilePressed>
  <Scale Factor=".95" Duration=".08" Easing="QuadraticOut" />
</WhilePressed>
```

hikr.Button을 눌렀을 때 크기가 약간 축소(0.95)되는 애니메이션 효과를 삽입한다. hikr.Button은 컴포넌트 클래스이므로 <hikr.Button ...>와 같은 방식으로 컴포넌트를 생성해야 화면에 보이게 된다. SplashPage.ux, EditHikePage.ux에서 hikr.Button을 생성하는 코드를 살펴보자.

```
<hikr.Button Text="Get Started" FontSize="18" Margin="50,0"
  Alignment="VerticalCenter" Clicked="{goToHomePage}" />
```

SplashPage.ux에서 hikr.Button을 생성하는 예다. Text, FontSize의 값은 hikr.Button 컴포넌트에 전달되어 내부의 hikr.Text에서 사용된다. hikr.Button의 기본 Margin 값은 10이지만 Margin을 50,0으로 지정했으므로 새로운 Margin인 50,0으로 설정된다. Padding은 값을 지정하지 않았으므로 기본 값인 10이 적용된다.

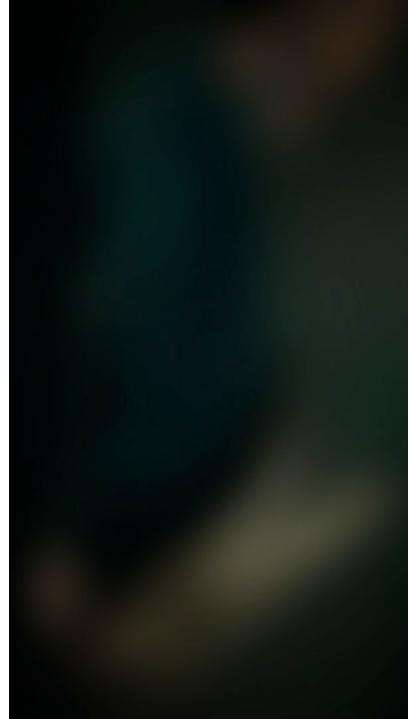
```
<hikr.Button Text="Cancel" Clicked="{cancel}" />
<hikr.Button Text="Save" Clicked="{save}" />
```

EditHikePage.ux에서 hikr.Button을 생성하는 예다. Text의 값은 Cancel, Save로 지정했지만 FontSize는 값이 없으므로 hikr.Button의 기본 FontSize인 16으로 할당된다. Margin, Padding도 기본값인 10으로 할당된다.

2. hikr.Page 컴포넌트

HomePage, EditHikePage는 공통적으로 아래 그림과 같은 배경 이미지가 표시된다. 따라서 hikr.Page라는 Page 컴포넌트를 정의하고 배경 이미지를 아래 그림으로 지정해서 공통적으로 사용하였다.

hikr 배경 이미지



hikr.Page.ux

```
<Page ux:Class="hikr.Page">
  <Image Layer="Background" File="../Assets/background.jpg"
    StretchMode="UniformToFill" Opacity=".7" />
</Page>
```

```
<Page ux:Class="hikr.Page">
```

컴포넌트는 Page 태입이며 클래스 이름을 hikr.Page로 지정한다.

```
<Image Layer="Background" File="../Assets/background.jpg"
  StretchMode="UniformToFill" Opacity=".7" />
```

페이지의 배경을 위의 hikr 배경 이미지인 background.jpg로 설정한다. 투명도를 0.7만큼 주고 페이지 전체를 꽉 채우게 이미지를 맞춘다.
HomePage.ux, EditHikePage.ux를 보면 제일 처음에 <hikr.Page ... >로 시작하므로 배경 이미지가 자동 설정된다.

3. hikr.Text 컴포넌트

hikr 앱에서는 흰색 텍스트를 많이 사용하므로 처음부터 흰색 텍스트를 보여주는 hikr.Text 컴포넌트를 정의한다. 이렇게 별도의 컴포넌트를 정의하면 나중에 텍스트 색이 흰색에서 파란색으로 변경 되도 이곳 안에서 Color를 수정하기만 하면 된다.

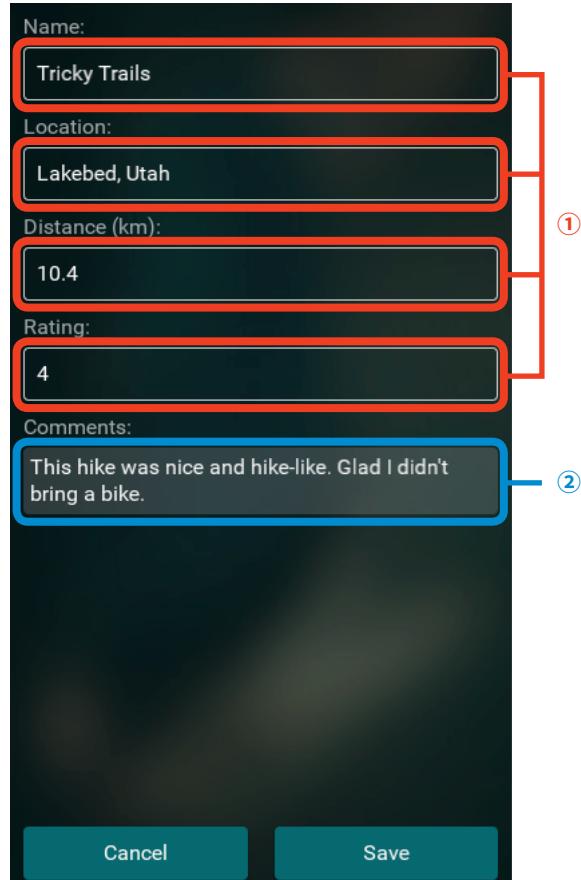
hikr.Text.ux

<Text ux:Class="hikr.Text" Color="White" />

4. hikr.TextBox, hikr.TextView 컴포넌트

EditHikePage에서 name, location, distance, rating, comment 등의 텍스트를 입력하고 보여지는 컴포넌트를 정의한다.

EditHikePage.ux



위 그림의 붉은색 사각형 부분(①)이 hikr.TextBox 컴포넌트이다. 텍스트의 색과 커서(Caret)가 흰색이며 패딩은 10,10,0,10을 적용한다.

hikr.TextBox.ux

```
<TextBox ux:Class="hikr.TextBox" TextColor="White" CaretColor="White"  
Padding="10,10,0,10" />
```

위 그림의 파란색 사각형 부분(②)이 hikr.TextView 컴포넌트이다. 텍스트의 색과 커서가 흰색이며 사각형 모양의 배경색을 표시하고 모서리를 둥글게 한다.

hikr.TextView.ux

```
<TextView ux:Class="hikr.TextView" TextColor="White" CaretColor="White"  
Padding="5">  
  <Rectangle Layer="Background" Color="#fff2" CornerRadius="4" />  
</TextView>
```

02.5

이벤트 처리 및 내비게이션 이해하기

이제까지 화면을 구성하는 컴포넌트와 레이아웃 중심으로 hikr 앱의 구조를 살펴보았다. 지금부터는 사용자가 각 페이지에서 하는 행위, 즉 앱의 기능을 중심으로 이해하기로 한다. 퓨즈 앱의 기능은 자바스크립트라는 언어로 구현한다. 자바스크립트 코드는 UX 파일 안에 포함되거나 외부의 파일로 독립적으로 구현될 수 있다.

1. SplashPage에서 “Get Started” 버튼을 누를 때의 이벤트 처리 및 내비게이션

```
<JavaScript File="SplashPage.js" />
```

SplashPage.ux 안에는 SplashPage.js 자바스크립트 파일이 지정되어 있다. 모바일 기기에서 SplashPage 화면이 표시될 때, 딱 한번 SplashPage.js의 코드가 실행된다.

```
SplashPage.js
function goToHomePage() {
    router.goto("home");
}

module.exports = {
    goToHomePage: goToHomePage
};
```

```
function goToHomePage() {
    router.goto("home");
}
```

goToHomePage() 자바스크립트 함수를 정의한다. 이 함수가 실행되면 router.goto("home")이 실행되는데 여기서 router는 SplashPage.ux 안에서 `<Router ux:Dependency="router" />`를 통해 넘겨 받은, 페이지를 전환할 수 있는 router 객체이다. goto 함수를 호출해서 home으로 명명된 HomePage로 이동한다.

```
module.exports = {
  goToHomePage: goToHomePage
};
```

SplashPage.ux에서 goToHomePage 함수를 호출할 수 있도록 설정하려면 반드시 module.exports 객체에 함수 이름을 등록해야 한다. goToHomePage (UX파일에서 호출하는 함수 이름) : goToHomePage(javascript 파일에 정의된 자바스크립트 함수 이름)으로 설정한다.

```
<hikr.Button Text="Get Started" FontSize="18" Margin="50,0"
Alignment="VerticalCenter" Clicked="{goToHomePage}" />
```

SplashPage.ux에서 “Get Started” 버튼을 누르면 Clicked에 설정된 goTo HomePage() 함수가 SplashPage.js의 module.exports에 등록되었는지 확인하고, 만약 있으면 실행된다. 현재 module.exports 안에 goToHomePage가 등록되어 있으므로 goToHomePage() 함수가 실행되어 HomePage로 이동한다.

2. HomePage에서 하이킹 기록 이름을 클릭했을 때의 이벤트 처리 및 내비게이션

```
<JavaScript File="HomePage.js" />
```

HomePage.ux 안에는 HomePage.js 자바스크립트 파일이 지정되어 있다. 모바일 기기에서 HomePage 화면이 표시될 때, 딱 한번 HomePage.js의 자바스크립트 코드가 실행된다.

```
HomePage.js
var Context = require("Modules/Context");

function goToHike(arg) {
  var hike = arg.data;
  router.push("editHike", hike);
}

module.exports = {
  hikes: Context.hikes,
  goToHike: goToHike
};
```

```
var Context = require("Modules/Context");
```

먼저 Modules 디렉토리에 있는 Context.js를 실행한 후 Context 객체를 생성한다. Context.js는 하이킹 기록 정보를 담고 있으며 값을 수정할 수 있는 자바스크립트 함수가 정의되어 있다.

Context.js를 자세히 살펴보자.

```
Context.js
var Observable = require("FuseJS/Observable");
var hikeList = [
```

```
{  
    id: 0,  
    name: "Tricky Trails",  
    location: "Lakebed, Utah",  
    distance: 10.4,  
    rating: 4,  
    comments: "This hike was nice and hike-like. Glad I didn't bring  
              a bike."  
},  
{  
    id: 1,  
    name: "Mondo Mountains",  
    location: "Black Hills, South Dakota",  
    distance: 20.86,  
    rating: 3,  
    comments: "Not the best, but would probably do again. Note to self:  
              don't forget the sandwiches next time."  
},  
{  
    id: 2,  
    name: "Pesky Peaks",  
    location: "Bergenhagen, Norway",  
    distance: 8.2,  
    rating: 5,  
    comments: "Short but SO sweet!!"  
},  
{  
    id: 3,  
    name: "Rad Rivers",  
    location: "Moriyama, Japan",  
    distance: 12.3,  
    rating: 4,  
    comments: "Took my time with this one. Great view!"  
},  
{  
    id: 4,  
    name: "Dangerous Dirt",  
    location: "Cactus, Arizona",  
    distance: 19.34,  
    rating: 2,  
    comments: "Too long, too hot. Also that snakebite wasn't very fun."  
};  
  
var hikes = Observable();  
hikes.replaceAll(hikeList);  
  
function updateHike(id, name, location, distance, rating, comments) {  
    for (var i = 0; i < hikes.length; i++) {  
        var hike = hikes.getAt(i);  
        if (hike.id == id) {  
            hike.name = name;  
            hike.location = location;  
            hike.distance = distance;  
            hike.rating = rating;  
            hike.comments = comments;  
            hikes.replaceAt(i, hike);  
            break;  
        }  
    }  
}  
  
module.exports = {  
    hikes: hikes,  
    updateHike: updateHike  
};
```

```
var Observable = require("FuseJS/Observable");
```

자바스크립트 객체 안의 특정 값이 UX 컴포넌트들에 연결되기 표시되기 위해서는 Observable 객체를 생성하고 그 안에 값을 넣어야 한다. Observable 객체를 만들기 위해서는 FuseJS/Observable 모듈을 임포트한다. Observable 객체의 값을 표시하는 컴포넌트는 Value 속성에 {"Observable 객체 이름"}을 지정해야 한다.

```
var hikeList = [
  {
    id: 0,
    name: "Tricky Trails",
    location: "Lakebed, Utah",
    distance: 10.4,
    rating: 4,
    comments: "This hike was nice and hike-like. Glad I didn't bring
               a bike."
  },
  {
    id: 1,
    name: "Mondo Mountains",
    location: "Black Hills, South Dakota",
    distance: 20.86,
    rating: 3,
    comments: "Not the best, but would probably do again. Note to self:
               don't forget the sandwiches next time."
  },
  {
    id: 2,
    name: "Pesky Peaks",
    location: "Bergenagen, Norway",
    distance: 8.2,
    rating: 5,
    comments: "Short but SO sweet!!!"
  },
  {
    id: 3,
    name: "Rad Rivers",
    location: "Moriyama, Japan",
    distance: 12.3,
    rating: 4,
    comments: "Took my time with this one. Great view!"
  },
  {
    id: 4,
    name: "Dangerous Dirt",
    location: "Cactus, Arizona",
    distance: 19.34,
    rating: 2,
    comments: "Too long, too hot. Also that snakebite wasn't very fun."
  }
];
```

먼저 5개의 하이킹 기록 정보들을 hikeList 배열 객체에 넣는다.

```
var hikes = Observable();
hikes.replaceAll(hikeList);
```

hikes라는 Observable 객체를 만들고, 그 안에 replaceAll 함수를 실행하여 hikeList 배열 객체를 넣는다.

```
function updateHike(id, name, location, distance, rating, comments) {
  for (var i = 0; i < hikes.length; i++) {
    var hike = hikes.getAt(i);
    if (hike.id == id) {
      hike.name = name;
      hike.location = location;
      hike.distance = distance;
      hike.rating = rating;
      hike.comments = comments;
      hikes.replaceAt(i, hike);
      break;
    }
  }
}
```

Observable 객체인 hikes 안에 updateHike() 함수 파라미터인 id 값과 일치되는 하이킹 기록 정보가 있으면 다른 파라미터인 name, location, distance, rating, comments 등으로 해당 기록 정보를 업데이트하는 updateHike 함수를 정의한다.

```
module.exports = {
  hikes: hikes,
  updateHike: updateHike
};
```

Context.js 외부에서 hikes 객체와 updateHike() 함수를 사용할 수 있도록 module.exports에 등록한다.

다시 HomePage.js를 살펴보자.

```
function goToHike(arg) {
  var hike = arg.data;
  router.push("editHike", hike);
}
```

HomePage 화면에서 5개의 하이킹 기록 이름 중 하나를 클릭하면 호출되는 goToHike() 함수를 정의한다. 사용자가 하이킹 이름을 클릭하면 해당 하이킹 기록 객체가 goToHike() 함수의 arg 파라미터 안에 담겨서 넘겨지며 내부에서는 arg.data로 접근할 수 있다. arg.data는 하이킹 기록 객체인데 hike 객체에 넣은 후, HomePage.ux에서 사용 가능한 router 객체의 push를 실행하여 editHike로 명명된 EditHikePage로 이동한다. 이때 hike를 push의 파라미터로 넘김으로써 EditHikePage에서 hike 객체 안에 담긴 id, name, location, distance, ratings, comments 등을 사용할 수 있게 된다.

router의 push 명령어는 현재 화면을 임시로 저장하고 다른 페이지로 이동한다. 이동 후에 이전 버튼을 누르거나 직전 페이지로 이동하는 코드가 실행되면 임시로 저장된 화면으로 되돌아갈 수 있다. 따라서 다른 페이지로 이동한 후, router의 goBack 함수나 백버튼 등으로 기존 페이지로 이동할 수 있게 된다.

```
module.exports = {
  hikes: Context.hikes,
  goToHike: goToHike
};
```

HomePage.js 안에서 hikes 객체, goToHike() 함수를 HomePage.ux 또는 다른 자바스크립트에서 사용할 수 있도록 module.exports에 등록한다. hikes 객체는 Context.hikes 명령어를 사용해서 위에서 설명한 Context 모듈에 등록된 hikes 객체와 연결된다.

```
<Each Items="{hikes}">
  <Separator />

  <Panel HitTestMode="LocalBoundsAndChildren" Clicked="{goToHike}">
    <hikr.Text Value="{name}" Margin="20" />
    <WhilePressed>
      <Scale Factor=".95" Duration=".08" Easing="QuadraticOut" />
    </WhilePressed>
  </Panel>
</Each>
```

다시 HomePage.ux를 살펴보자. `<Each Items="{hikes}">`에서 hikes는 module.exports에 등록된 hikes, 즉 Context.hikes 객체를 의미하며 하이킹 기록 정보 5개가 들어 있다. Each 문장이 hikes 배열 객체의 원소의 개수만큼, 즉 5번 반복되어 실행되는데 `<hikr.Text Value="{name}" Margin="20" />`와 같이 각 하이킹 기록 정보 객체의 name 값이 표시된다. 그리고 `<Panel HitTestMode ="LocalBoundsAndChildren" Clicked="{goToHike}">`의 Clicked 이벤트 핸들러로 module.export의 goToHike() 함수가 연결되어 있다. 따라서 하이킹 기록 이름을 클릭하면 goToHike() 함수가 실행되어 다른 페이지로 이동하는 것이다.

3. EditHikePage에서 Cancel, Save 버튼을 클릭했을 때의 이벤트 처리 및 내비게이션

```
<JavaScript File="EditHikePage.js" />
```

EditHikePage.ux 안에는 EditHikePage.js 자바스크립트 파일이 지정되어 있다. 모바일 기기에서 EditHikePage 화면이 표시될 때, 딱 한번 EditHikePage.js의 자바스크립트 코드가 실행된다.

```
var Context = require("Modules/Context");
var hike = this.Parameter;
var name = hike.map(function(x) { return x.name; });
var location = hike.map(function(x) { return x.location; });
var distance = hike.map(function(x) { return x.distance; });
var rating = hike.map(function(x) { return x.rating; });
var comments = hike.map(function(x) { return x.comments; });
```

```

function cancel() {
    // Refresh hike value to reset dependent Observables' values
    hike.value = hike.value;
    router.goBack();
}

function save() {
    Context.updateHike(hike.value.id, name.value, location.value,
    distance.value, rating.value, comments.value);
    router.goBack();
}

module.exports = {
    name: name,
    location: location,
    distance: distance,
    rating: rating,
    comments: comments,
    cancel: cancel,
    save: save
};

```

```
var Context = require("Modules/Context");
```

먼저 Modules 디렉토리에 있는 Context.js를 실행한 후 Context 객체를 생성한다. 사용자가 “Save” 버튼을 클릭하면 하이킹 기록을 수정해서 업데이트하는 updateHike() 함수가 Context.js 안에 구현되어 있기 때문이다. 그런데 이미 HomePage.js에서 require('Modules/Context') 명령을 실행하여 Context.js 가 먼저 실행되었으므로 EditHikePage.js에서는 새롭게 Context 객체를 만들지 않고 이미 만들어진 Context 모듈을 재사용한다.

```
var hike = this.Parameter;
```

HomePage.js에서 router.push("editHike", hike);가 실행되어 하이킹 기록 객체인 hike가 EditHikePage에 전달될 때 EditHikePage.js에서는 this.Parameter;로 받을 수 있다. 이렇게 받은 hike 객체가 아래와 같다고 가정하자.

사용자가 선택한
“Mondo Mountains”
하이킹 정보 객체의 내용

객체 속성(property)	값(value)
id	0
name	Tricky Trails
location	Lakebed, Utah
distance	10.4
rating	4
comments	This hike was nice and hike-like. Glad I didn't bring a bike.

```
var name = hike.map(function(x) { return x.name; });
var location = hike.map(function(x) { return x.location; });
var distance = hike.map(function(x) { return x.distance; });
var rating = hike.map(function(x) { return x.rating; });
var comments = hike.map(function(x) { return x.comments; });
```

Observable 객체인 hike의 원소 각각에 대해 map 안에 정의된 함수가 실행된다. map 함수가 실행되면 새로운 Observable 객체가 생성된다. 현재 hike 객체는 “Tricky Trails” 하이킹 기록 정보가 1개만 들어가 있으며, function(x)의 x에 파라미터 값으로 넘어간다. 따라서 hike.map(function(x) { return x.name; })는 “Tricky Trails” 값이 저장된 Observable 객체가 반환된다. location, distance, rating, comments는 각각 Lakebed, Utah, 10.4, 4, “This hike was nice and hike-like. Glad I didn’t bring a bike.”의 값이 들어간 Observable 객체가 할당된다.

```
function cancel() {
    // Refresh hike value to reset dependent Observables' values
    hike.value = hike.value;
    router.goBack();
}
```

“Cancel” 버튼을 클릭하면 호출되는 함수다. 화면에서 name, location 등의 값을 수정해도 Cancel 버튼을 누르면 초기값으로 돌아간다. “Save” 버튼을 눌러야 변경된 값이 유지된다.

hike.value = hike.value;는 매우 중요하다. 이 문장을 빼면 Cancel 버튼을 눌러서 HomePage로 갔다가 다시 동일한 하이킹 기록 이름을 선택하여 Edit HikePage로 가면 수정된 값이 취소되지 않고 그대로 있게 된다. 왜냐면 사용자가 수정한 값이 Observable 객체에 담기면 페이지가 전환되어도 바인딩 된 화면에 그대로 유지되기 때문이다. 따라서 화면이 전환되기 전에 입력 항목의 값이 원래 값으로 복원되기 위해서는 name, location, distance, comments, rating 등이 파생된 원래 Observable 객체인 hike를 기존 데이터 값으로 변경해야 한다. 그래서 hike.value = hike.value(이전 값); 명령을 실행하는 것이다. 그러면 자동으로 원래 값으로 화면 값이 변경된다.

router.goBack();이 실행되면 직전 페이지인 HomePage로 이동한다.

```
function save() {
    Context.updateHike(hike.value.id, name.value, location.value,
                      distance.value, rating.value, comments.value);
    router.goBack();
}
```

“Save” 버튼을 클릭하면 호출되는 함수다. 화면에서 name, location 등의 값을 수정했을 때 그 값을 다른 페이지로 이동해도 그대로 유지하는 역할을 한다. Context 객체의 updateHike() 함수를 호출해서 변경된 하이킹 기록 정보를

Context 객체 내의 Observable hikes 객체에 업데이트한다. 이렇게 업데이트된 Context 객체 내의 hikes는 HomePage, EditHikePage 내에서 값이 사용된다. `router.goBack();`이 실행되면 직전 페이지인 HomePage로 이동한다.

```
module.exports = {
  name: name,
  location: location,
  distance: distance,
  rating: rating,
  comments: comments,

  cancel: cancel,
  save: save
};
```

EditHikePage.js 안에서 name, location, distance, rating, comments 객체와 cancel, save 함수를 EditHikePage.ux 또는 다른 자바스크립트에서 사용할 수 있도록 module.exports에 등록한다.

```
<StackPanel>
  <TitleText>Name:</TitleText>
  <hikr.TextBox Value="{name}" />
</StackPanel>

<StackPanel>
  <TitleText>Location:</TitleText>
  <hikr.TextBox Value="{location}" />
</StackPanel>

<StackPanel>
  <TitleText>Distance (km):</TitleText>
  <hikr.TextBox Value="{distance}" InputHint="Decimal" />
</StackPanel>

<StackPanel>
  <TitleText>Rating:</TitleText>
  <hikr.TextBox Value="{rating}" InputHint="Integer" />
</StackPanel>

<StackPanel>
  <TitleText>Comments:</TitleText>
  <hikr.TextView Value="{comments}" TextWrapping="Wrap" />
</StackPanel>
</StackPanel>
```

다시 EditHikePage.ux를 살펴보자. `<hikr.TextBox Value="{name}" />`, `<hikr.TextBox Value="{location}" />`, `<hikr.TextBox Value="{distance}" InputHint="Decimal" />`, `<hikr.TextBox Value="{rating}" InputHint="Integer" />`, `<hikr.TextView Value="{comments}" TextWrapping="Wrap" />` 등을 통해 name, location, distance, rating, comments 등의 입력 항목에 사용자가 값을 넣으면 EditHikePage.js의 module.exports에 등록되어 있는 name, location, distance, rating, comments 등의 Observable 객체의 value 속성에 값이 저장된다. 이 값은 EditHikePage.js의 save() 함수 안에서 Context.updateHike() 함수를 호출할 때 사용된다.

03

퓨즈 UX

마크업 언어 배우기

03.1 UX 마크업 언어 소개

03.2 값의 종류

03.3 표현식(expression)

03.4 ux: 속성

03.5 기본 컨트롤

03.6 화면 레이아웃

03.7 화면 내비게이션

03.8 트리거와 애니메이션

03.1

UX 마크업 언어 소개

UX 마크업 언어(UX Markup Language)는 사용자 인터페이스와 레이아웃, 비주얼 효과, 애니메이션 등을 XML로 표현하는 언어이다. UX 마크업 언어는 퓨즈에서 가장 중요한 파트이다. 화면을 빠르게 만들 수 있기 때문이다. 이 언어로 작성된 파일은 ux 확장자로 저장된다.

1. 태그(Tag)와 요소(Element)

```
<App>
  <Rectangle Color="Blue" />
  <Circle Color="Red" />
</App>
```

<App>...</App>, <Rectangle Color="Blue" />, <Circle Color="Red" />처럼 시작 태그와 종료 태그까지의 내용을 UX 마크업 요소(element)라고 부른다. 위의 ux 코드는 총 3개의 요소가 있다.

태그는 요소의 일부분이며 시작 태그는 <태그 명>, 종료 태그는 </태그 명>으로 표시한다. 위의 <App>, </App>이 그 예다. 시작과 종료 태그 사이에 포함하는 것이 없다면 <Rectangle Color="Blue" />처럼 <태그 명 />으로 표시한다.

<App> 태그는 모바일 앱의 루트, 즉 최상단 태그이며 반드시 존재해야 한다.

```
<App>
  모바일 앱의 화면
</App>
```

2. 객체(Object) 또는 인스턴스(Instance)

퓨즈는 ux 파일의 요소를 클래스(class)의 객체(=인스턴스)로 생성한다. 위의 예제에서는 App, Rectangle, Circle이라는 화면 컴포넌트 클래스의 객체가 생성된다. 이를 화면 클래스는 퓨즈 라이브러리에 사전 정의 되어 있으며 unoproj

파일의 “Packages” 섹션에 “Fuse” 라이브러리를 추가해야만 사용할 수 있다.
“Packages” 섹션은 퓨즈에서 사용하는 라이브러리를 포함하는 곳이다.

```
"Packages": [  
    "Fuse",  
    "FuseJS"  
,
```

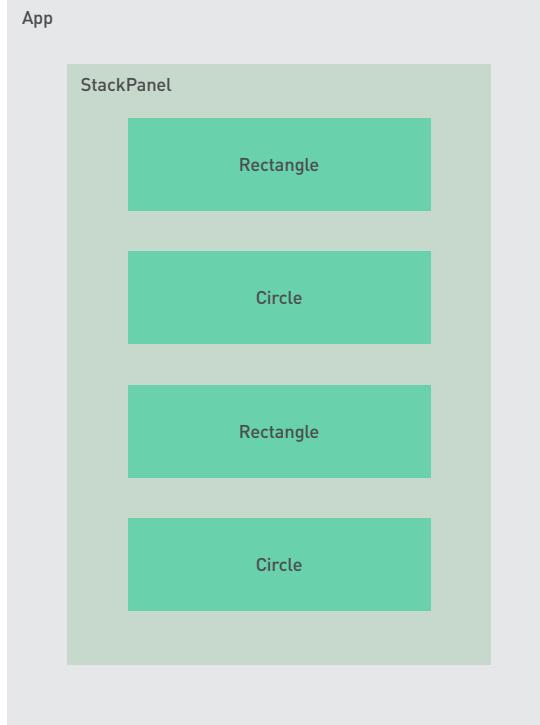
3. 포함 관계

하나의 요소는 다른 요소를 포함할 수 있다. 포함하는 요소를 부모(parent), 포함되는 요소를 자식(child)로 부른다.

```
<App>  
  <StackPanel>  
    <Rectangle Color="Blue" Margin="10" Width="100" Height="100" />  
    <Circle Color="Red" Margin="10" Width="100" Height="100" />  
    <Rectangle Color="Blue" Margin="10" Width="100" Height="100" />  
    <Circle Color="Red" Margin="10" Width="100" Height="100" />  
  </StackPanel>  
</App>
```

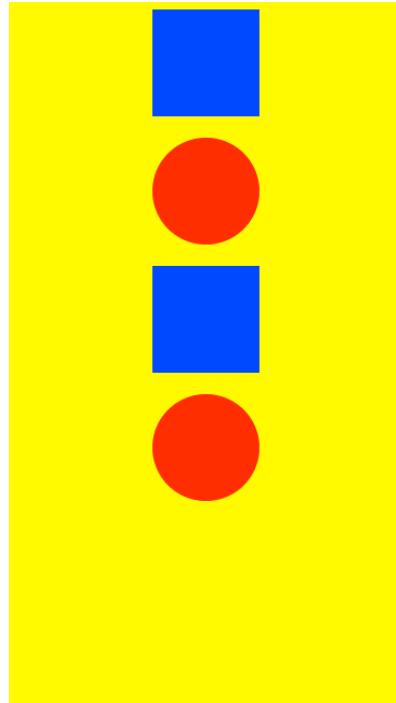
위의 코드는 아래와 같은 App > StackPanel > Rectangle, Circle, Rectangle, Circle 과 같은 포함관계가 형성되어 있다.

요소 간 포함 관계



실행 결과는 아래와 같다.

실행 결과



4. 속성(Property)

UX 마크업 요소는 여러 개의 속성(property)과 값을 옵션으로 가질 수 있다.

```
<태그이름 속성1="값" 속성2="값" ... 속성3="값"></태그이름>
```

```
<Rectangle Color="Blue" Margin="10" Width="100" Height="100" />
```

위의 코드에서 Rectangle은 사각형을 상징하는 태그이다. Color, Margin, Width, Height는 사각형의 속성이며 Color는 색, Margin은 다른 컴포넌트와의 간격, Width는 너비, Height는 높이를 의미한다. 해석하면 사각형 모양의 컴포넌트가 있는데 색은 파란색, 너비와 높이의 길이는 100 포인트(pt)이며 다른 컴포넌트와의 간격은 10 포인트이다.

03.2

값의 종류

UX 마크업 언어에서 속성에 할당되는 값의 형태를 살펴보자. 속성에 값을 할당할 때는 “ ” 기호로 값을 둘러싸야 한다.

1. 숫자

정수나 실수 값을 아래와 같은 형태로 표현할 수 있다.

1
1.3
13
.4 (0.4와 같은 표시)

예

- <Rectangle Width="10.5" Height="20" />

2. 단위

어떤 속성은 단위(unit) 값을 지원한다. 단위는 %(percent), pt(point), px (pixel) 등이 있다. 숫자 다음에 단위를 쓰면 된다. 단위 형태가 지정되지 않으면 pt가 적용된다.

예

- <Rectangle Width="50px" Height="20pt" /> : 너비 50 픽셀, 높이 20 포인트
- <Panel Width="50%" Height="100" /> : 너비는 부모 요소 너비의 50%, 높이는 100 포인트

3. 벡터

어떤 속성은 여러 개의 값을 넣어야 한다. 예를 들어 Margin 속성은 위, 아래, 왼쪽, 오른쪽 등의 4개 여백 값을 넣어야 한다.

예

- <Panel Margin="10, 20, 30, 40" /> : 왼쪽 여백 10, 위 여백 20, 오른쪽 여백 30, 아래 여백 40

여러 개의 값이 들어가야 하는 속성 값에 1개의 값을 넣으면 모든 여백을 동일한 값으로 자동 확장되어 설정된다.

예

- <Panel Margin="10" /> : <Panel Margin="10, 10, 10, 10" /> 과 같은 뜻

2개의 값이 “,”로 분리되어 있으면 좌우, 그리고 상하 여백 값으로 자동 확장되어 설정된다.

예

- <Panel Margin="10, 20" /> : <Panel Margin="10, 20, 10, 20" /> 과 같은 뜻

그리고 색상을 나타내는 속성인 Color는 각각 빛의 삼원색인 빨강(Red), 녹색(Green), 파랑(Blue)을 각각 0에서 1까지 “,”로 분리하여 비율 값을 설정한다. 이 때, 투명도에 해당하는 알파 값 1.0이 추가되어 설정된다.

예

- <Panel Color="1, 0, 1" /> : <Panel Color="1, 0, 1, 1" /> 과 같은 뜻. 알파 채널의 값이 1로 설정됨

어떤 속성은 여러 개의 값이 들어갈 때 각 값의 단위가 다를 수 있다. 예를 들어 Offset 이 그것이다. 아래는 Offset값을 X좌표를 기준으로 10%, Y좌표를 기준으로 20 pixel만큼 설정하는 속성을 Panel 태그에 지정한 것이다.

예

- <Panel Offset="10%, 20px" />

4. 색상

색상 속성은 “,”로 구분하는 벡터로도 사용 가능하나, 앞에 “#”을 붙인 후 빨강(Red), 녹색(Green), 파랑(Blue) 값을 각각 16진수로 0~255까지로 지정하는 16진수 표기 방식으로도 지정 가능하다. 16진수 표기 방식은 다음과 같은 조합을 사용하여 색의 값을 지정한다.

- 6자리 16진수 표기 방식 (#RRGGBB)
 - RR: 빨강에 대해 00에서 FF까지 지정 (0~255)
 - GG: 녹색에 대해 00에서 FF까지 지정 (0~255)
 - BB: 파랑에 대해 00에서 FF까지 지정 (0~255)
- 8자리 16진수 표기 방식 (#RRGGBBAA)
 - RR, GG, BB: 6자리 16진수 표기 방식과 동일
 - AA: 투명도에 대한 알파 값을 00에서 FF까지 지정 (0~255)

- 3자리 16진수 표기 방식 (#RGB)
 - 6자리 16진수 표기 방식을 간단히 줄인 것으로, 예를 들어 #112233은 #123으로 줄여서 표현이 가능하다 (#123456의 경우에는 3자리로 줄여 쓸 수 없다).
- 4자리 16진수 표기 방식 (#RGBA)
 - 8자리 16진수 표기 방식을 간단히 줄인 것으로, 예를 들어 #11223344는 #1234로 줄여서 표현이 가능하다 (#12345678의 경우에는 4자리로 줄여 쓸 수 없다).

알파 값을 지정하지 않는 표기 방식의 경우, 알파 값은 최대값으로 가정한다. 다음 예시와 같이 속성을 사용한다.

예

- <Rectangle Color="#18f"> (연한 파랑)
- <Rectangle Color="#fff"> (흰색)
- <Rectangle Color="#ff00ff77"> (투명도가 처리된 보라색)

5. 문자열

기본적인 문자열 값은 “ ”로 표시된다.

예

- <Text Value="Hello, World" />

만약 자바스크립트의 특정 변수의 값을 문자열 안에 표시하려면 해당 변수를 { }에 넣으면 된다.

예

- <Text Value="Hello, {username}!" /> : 자바스크립트 변수인 username의 값이 “Gildong”이라면 화면에 Hello, Gildong!이란 텍스트가 표시된다.

또한 계산 및 표현식을 “=”기호를 사용하여 문자열 안에 넣을 수 있다.

예

- <Text Value="Hello {= toLower({username}) } /> : 자바스크립트 변수인 username의 값이 “Gildong”이라면 toLower란 기본 제공 함수의 실행결과, 즉 Gildong이 소문자인 gildong으로 변환된 후 화면에 Hello, gildong!이란 텍스트가 표시된다.

03.3

표현식(expression)

퓨즈는 위에서 설명한 각종 값과 자바스크립트 변수 등을 연산자와 함수를 사용해서 결합한 후 특정 값을 계산하는 표현식을 제공한다.

1. 수식

표현식 안에서 +, -, *, / 등을 사용할 수 있다.

예

- <Panel Width="{foo} * 100% + 40%" /> : 자바스크립트 foo 변수의 숫자 값에 100을 곱하고 40을 더한 후 % 단위를 적용한다.
- <Slider Margin="{foo}" /> : 자바스크립트 foo 변수 값을 Margin으로 적용한다.

2. 벡터

여러 개의 값이 들어가는 속성에도 {}를 사용하여 자바스크립트 변수의 값을 할당할 수 있다.

예

- <Panel Margin="10, {spacing}, 10, {spacing} / 2" /> : 벡터 값을 표현할 때도 {}를 사용해서 자바스크립트 spacing 변수의 값을 할당할 수 있다.
- <Panel Margin="{spacing}" /> : <Panel Margin="{spacing}, {spacing}, {spacing}, {spacing}" />과 동일한 의미임.

3. 문자열과 Text 컴포넌트

Text 컴포넌트의 Value 속성 값을 지정하는 것은 <Text> </Text> 태그 사이에 값을 넣는 것과 같다. 단, Value 속성에 값을 넣을 때는 “ ”를 반드시 사용해야 한다.

예를 들어, <Text>Hello, {username}!</Text> 는 <Text Value="Hello, {username}" /> 와 의미가 같다.

4. 기본 제공 함수

표현식에 사용할 수 있는 기본 함수는 아래와 같다. 퓨즈의 버전이 달라질수록 지원하는 기본 함수가 추가되거나 삭제될 수 있기 때문에 주의가 필요하다. 전체 리스트는 <https://www.fusetools.com/docs/ux-markup/expressions>를 참고한다.

1) 레이아웃

- x(element): element의 부모 컴포넌트 대비 X 위치 값 반환 (가로 축)
- y(element): element의 부모 컴포넌트 대비 Y 위치 값 반환 (세로 축)
- width(element): element의 너비를 pt로 반환
- height(element): element의 높이를 pt로 반환

2) 문자열

- toLower(s): s의 문자를 소문자로 변환해서 반환
- toUpper(s): s의 문자를 대문자로 변환해서 반환

3) 수학

- min(a, b): a, b 중 작은 숫자를 반환
- max(a, b): a, b 중 큰 숫자를 반환
- abs(value): value의 절대 값을 반환

이외에도 20여가지의 수학 함수가 있다.

03.4

ux: 속성

요소의 속성 중에서 ux: 이 붙은 것은 특별한 역할을 한다.

1. ux:Name 속성

UX 요소의 이름을 설정할 때 사용한다. 이름을 설정해야 하는 경우는 특정 요소가 다른 요소에 의해 참조되어야 하는 때이다.

예

- <Panel ux:Name="panel1" /> : 이 Panel의 이름은 panel1으로 설정됨.
- <Rectangle LayoutMaster="panel1" /> : 이 Rectangle의 LayoutMaster는 위에서 panel1로 명명된 Panel 객체가 됨.
- <Rectangle Width="width(panel1) / 2"> : 위에서 panel1로 명명된 Panel의 너비/2 가 사각형의 너비가 됨.

2. ux:Class 속성

특정 요소를 컴포넌트 클래스(class) 타입으로 설정할 때 사용된다. 모든 요소는 인스턴스(instance), 클래스(class), 템플릿(template) 등의 타입으로 구분되는데 그 차이점을 아는 것이 중요하다.

<Text Color="White" Value="Hello."/>는 ux:Class, ux:Template 등이 지정되지 않았기 때문에 인스턴스 타입의 요소이다. ux 파일 안에 인스턴스 요소가 있으면 즉시 컴포넌트가 생성되고 화면에 표시된다. 따라서 본 예제 코드는 화면에 Color가 White, 즉 흰색의 “Hello.” 글자를 출력한다.

<Text ux:Class="WhiteText" Color="White" />는 ux:Class 가 지정되었으므로 클래스 타입의 요소이다. ux 파일 안에서 클래스 타입의 요소가 있으면 인스턴스 타입의 요소와 달리 즉시 컴포넌트가 생성되는 것이 아니라 새로운 종류의 컴포넌트를 준비만 하고 <WhiteText> 태그 이름으로 생성되기를 대기한다. 즉, ux 파일 안에서 <WhiteText Value="Hello."/>라고 하면 그때서야 <Text ux:Class="WhiteText" Color="White" />로 정의한 컴포넌트가 생

성되어 화면에 표시된다. Color가 White, 즉 흰색 글자인 “Hello.”가 표시된다. 클래스 타입의 요소는 새로운 종류의 컴포넌트를 정의하는 역할만 하고, 생성하지는 않는다. <Text ux:Class="WhiteText" Color="White" />는 WhiteText라는 이름의 새로운 컴포넌트를 정의했는데 그 컴포넌트는 사전 정의된 Text 컴포넌트의 모든 속성과 기능을 물려 받는다.

ux:Class의 값은 보통 {프로젝트 명}.{컴포넌트 이름}으로 명명한다. 만약 프로젝트 명이 abc라면 <Text ux:Class="abc.text" Color="White" />와 같이 설정하는 것이 관행이다. 한 ux 파일 안에서만 사용된다면 그 파일 안에서, 여러 ux 파일에서 사용된다면 별도 파일로 정의한다.

ux:Class는 모바일 앱 내에서 여러 곳에서 공통적으로 사용되는 화면 컴포넌트를 별도로 구분하여 재사용하기 위한 목적으로 사용된다. 따라서 매우 중요하다.

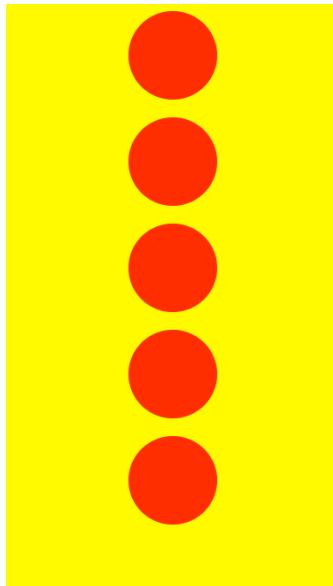
2부에서도 앱 내에서 자주 사용되는 컴포넌트를 hikr.Button, hikr.Page, hikr.Text, hikr.TextBox, hikr.TextView 등의 클래스 타입으로 정의해서 코드를 재사용하였다.

1) ux:Class 요소 없이 비슷한 여러 컴포넌트를 생성하는 경우

똑같은 모양의 원 5개를 만든다고 가정해보자. 보통 아래와 같이 할 것이다.

```
<App>
  <StackPanel>
    <Circle Color="Red" Width="100" Height="100" Margin="10" />
    <Circle Color="Red" Width="100" Height="100" Margin="10" />
  </StackPanel>
</App>
```

실행 화면



만약 각 원의 Width, Height가 모두 110으로 변경되어야 한다면 위의 Circle 요소 5개의 Width, Height를 110으로 바꿔야 한다. 번거로운 일이다.

2) ux:Class 요소를 기존 ux 파일 안에 정의하는 경우

```
<App>
  <StackPanel Background="Yellow">
    <Circle ux:Class="my.Circle" Color="Red" Width="100" Height="100"
      Margin="10" />

    <my.Circle />
    <my.Circle />
    <my.Circle />
    <my.Circle />
    <my.Circle />
  </StackPanel>
</App>
```

5개의 원이 공통적인 모양이므로 별도의 ux:Class로 my.Circle이란 이름의 클래스 요소를 정의한다.

```
<Circle ux:Class="my.Circle" Color="Red" Width="100" Height="100"
  Margin="10" />
```

각 원을 생성하기 위해서는 `<my.Circle />`을 입력한다. 만약 각 원의 Width, Height가 모두 110으로 변경되어야 한다면 `<Circle ux:Class="my.Circle" Color="Red" Width="100" Height="100" Margin="10" />`의 Width, Height만 110으로 바꾸면 된다.

3) ux:Class 클래스 요소를 독립 ux 파일에 저장하는 경우

위 1개 파일을 아래와 같이 2개 파일로 분리할 수 있다. 이렇게 하면 my.Circle 을 MainView.ux와 다른 ux 파일에서도 사용할 수 있게 된다.

MainView.ux

```
<App>
  <StackPanel Background="Yellow">
    <my.Circle />
    <my.Circle />
    <my.Circle />
    <my.Circle />
    <my.Circle />
  </StackPanel>
</App>
```

my.Circle.ux

```
<Circle ux:Class="my.Circle" Color="Red" Width="100" Height="100"
  Margin="10" />
```

3. ux:Dependency 속성

ux:Class가 포함된 컴포넌트 클래스를 정의할 때 의존 관계를 설정하는 용도로 사용된다. 주로 클래스 컴포넌트 외부에서 값을 전달 받을 때 이용한다.

```
<타입이름 ux:Dependency="의존관계 이름" />
```

아래에 설명할 ux:Property와 유사하지만 ux:Dependency는 아래의 특징이 있다.

- 의존 관계로 넘겨 받는 값을 변경할 수 없다.
- 디폴트 값을 받을 수 없다.
- 컴포넌트가 생성되어 초기화될 때만 값을 넘겨 받을 수 있다.

```
<App>
  <Router ux:Name="r" />
  <Panel ux:Name="p" />
  <MyBackButton router="r" panel="p" />
</App>
```

```
<Router ux:Name="r" />
<Panel ux:Name="p" />
```

r, p란 이름으로 Router, Panel 객체를 생성한다.

```
<MyBackButton router="r" panel="p" />
```

그리고 MyBackButton 컴포넌트를 생성하는데 이때 router, panel이란 속성 이름에 이미 생성한 r, p 값을 넣는다. router, panel이란 속성은 MyBackButton 컴포넌트 내부에서 ux:Dependency로 설정된 `<Router ux:Dependency="router" />`, `<Panel ux:Dependency="panel" />`의 Dependency 이름과 일치해야 한다.

MyBackButton.ux

```
<Panel ux:Class="MyBackButton">
  <Router ux:Dependency="router" />
  <Panel ux:Dependency="panel" />

  <JavaScript>
    function clicked() {
      router.goBack();
    }
  <JavaScript>

  <Clicked>
    <Callback Handler="{clicked}">
  </Clicked>

  <WhilePressed>
    <Change panel.Opacity="0.5" Duration="0.3" />
  </WhilePressed>
</Panel>
```

```
<Router ux:Dependency="router" />
<Panel ux:Dependency="panel" />
```

Router, Panel 태입의 객체를 MainView.ux에서 넘겨 받아 router.goBack(); <Change panel.Opacity="0.5" Duration="0.3" />처럼 사용할 수 있다.

4. ux:Property 속성

ux:Class가 포함된 컴포넌트 클래스를 정의할 때 새로운 속성을 추가하는 용도로 사용된다.

```
<태입이름 ux:Property="새로운 속성 이름" />
```

ux:Dependency와 달리 기본 값을 줄 수 있다.

```
<Panel ux:Class="MyComponent" NumberOfThings="13">
  <int ux:Property="NumberOfThings" />
  ...
</Panel>
```

<MyComponent /> 컴포넌트를 생성할 때 NumberOfThings에는 기본 값 13이 할당된다. 물론 <MyComponent NumberOfThings="100" />처럼 값을 지정할 수도 있다.

ux:Property로 설정된 값은 {Property 속성 이름}의 포맷으로 다른 컴포넌트의 속성 값으로 설정할 수 있다.

```
<Panel ux:Class="MyButton" BackgroundColor="#f00">
  <float4 ux:Property="BackgroundColor" />
  <Text>SUBMIT</Text>
  <Rectangle Layer="Background" Color="{Property BackgroundColor}"
    CornerRadius="10" />
</Panel>
```

MyButton 컴포넌트에는 BackgroundColor라는 속성이 추가되었는데, 이 속성의 값은 Color="{Property BackgroundColor}"로 인해 Rectangle 컴포넌트의 Color 값으로 설정된다.

ux:Property의 타입은 보통 아래와 같이 정한다.

- 정수: int
- 실수: double 또는 float
- 단위가 있는 숫자: Size
- 2개 값이 있는 벡터: float2

5. ux:Global 속성

ux:Global 속성은 글로벌 자원 이름과 값을 설정하고 다른 곳에서 해당 자원을 참조할 수 있게 해준다.

```
<타입이름 ux:Global="자원의 key" [ux:Value="값"] ... />
```

```
<App>
  <float4 ux:Global="MyApp.ThemeColor" ux:Value="Green" />
  <Panel Color="{Resource MyApp.ThemeColor}" />
</App>
```

```
<float4 ux:Global="MyApp.ThemeColor" ux:Value="Green" />
```

MyApp.ThemeColor란 이름으로 글로벌 자원 이름을 정의하고, 값은 Green 을 할당한다.

```
<Panel Color="{Resource MyApp.ThemeColor}" />
```

{Resource MyApp.ThemeColor}를 통해 MyApp.ThemeColor의 값인 Green 을 찾아서 Panel의 Color로 설정한다.

6. ux:Template 속성

ux:Template 속성의 의미는 주어진 요소가 템플릿(Template), 즉 바로 생성되지 않고 필요시 생성하라는 것이다. 주로 <Each>, <Navigator> 안에서 활용된다.

```
<타입이름 ux:Template="템플릿 이름" ... />
```

2부의 예제 중, MainView.ux를 살펴보자.

```
<App Background="#022328">
  <iOS.StatusBarConfig Style="Light" />
  <Android.StatusBarConfig Color="#022328" />

  <Router ux:Name="hikrRouter" />

  <ClientPanel>
    <Navigator DefaultPath="splash">
      <SplashPage ux:Template="splash" router="hikrRouter" />
      <HomePage ux:Template="home" router="hikrRouter" />
      <EditHikePage ux:Template="editHike" router="hikrRouter" />
    </Navigator>
  </ClientPanel>
</App>
```

```
<SplashPage ux:Template="splash" router="hikrRouter" />
<HomePage ux:Template="home" router="hikrRouter" />
<EditHikePage ux:Template="editHike" router="hikrRouter" />
```

Navigator 안에 있는 SplashPage, HomePage, EditHikePage 요소가 ux:Template 속성이 할당된 것을 알 수 있다. 즉, SplashPage, HomePage, EditHikePage 컴포넌트 전부를 생성하지 말고 Navigator가 필요시에 생성하라는 뜻이다. Navigator의 DefaultPath가 splash, 즉 SplashPage를 가리키므로 처음에는 SplashPage 컴포넌트가 생성되어 화면에 표시된다.

03.5

기본 컨트롤

퓨즈는 기본적인 컴포넌트를 조립해서 복잡한 화면 컴포넌트를 생성할 수 있는데 그들을 기본 컨트롤이라고 하며 Text, Rectangle, Circle, Image, Video 등이 있다.

1. Rectangle 컴포넌트

<https://www.fusetools.com/docs/fuse/controls/rectangle> 참조

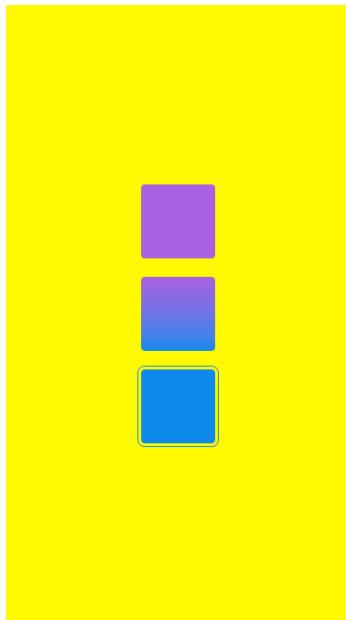
사각형을 표시한다. 기본적으로 아래와 같이 사용한다.

```
<Rectangle Color="Blue" Width="100" Height="100" />
```

사각형과 원 등 도형은 면(Fill)과 테두리(Stroke)의 색을 분리하여 지정할 수 있다. Color 속성은 Fill, 즉 면의 색을 의미한다. Width, Height 외에 다양한 속성이 존재한다.

```
<App Background="Yellow">
  <Grid Alignment="Center" Rows="100,100,100" Columns="100">
    <Rectangle Margin="10" CornerRadius="4">
      <SolidColor Color="#a542db" />
    </Rectangle>
    <Rectangle Margin="10" CornerRadius="4">
      <LinearGradient>
        <GradientStop Offset="0" Color="#a542db" />
        <GradientStop Offset="1" Color="#3579e6" />
      </LinearGradient>
    </Rectangle>
    <Rectangle Margin="10" CornerRadius="4">
      <Stroke Offset="4" Width="1" Color="#3579e6" />
      <SolidColor Color="#3579e6" />
    </Rectangle>
  </Grid>
</App>
```

Rectangle 화면



2. Circle 컴포넌트

<https://www.fusetools.com/docs/fuse/controls/circle> 참조

원을 표시한다. 기본적으로 아래와 같이 사용한다.

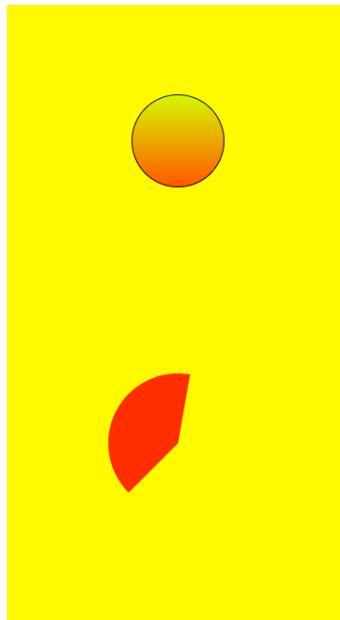
```
<Circle Width="100" Height="100" Color="#f00" />
```

원 안에 LinearGradient 요소를 넣어서 그라디에이션(gradation) 효과를 만들 수 있다. Stroke 요소를 쓰면 원의 두께를 설정하고 색을 지정할 수 있다. 또한 StartAngleDegrees, LengthAngleDegrees 등의 속성 값을 넣으면 시작 지점의 각도, 종료되는 각도를 계산하여 원 내부를 분할(slice) 할 수 있다.

```
<App Background="Yellow">
  <StackPanel>
    <Circle Width="100" Height="100" Margin="100" >
      <LinearGradient>
        <GradientStop Offset="0" Color="#cf0" />
        <GradientStop Offset="1" Color="#f40" />
      </LinearGradient>
      <Stroke Width="1">
        <SolidColor Color="#000" />
      </Stroke>
    </Circle>

    <Circle Width="150" Height="150" Color="#f00" StartAngleDegrees="135"
           LengthAngleDegrees="145" Margin="100"/>
  </StackPanel>
</App>
```

Circle 화면



3. Text 컴포넌트

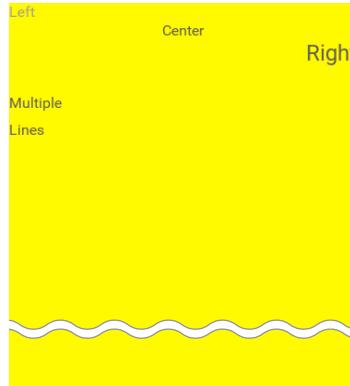
<https://www.fusetools.com/docs/fuse/controls/text> 참조

화면에 읽기 전용의 텍스트를 표시한다. 텍스트에 폰트를 지정할 수 있다.

```
<App Background="Yellow">
  <StackPanel>
    <Text Color="#999">Left</Text>
    <Text TextAlignment="Center">Center</Text>
    <Text FontSize="24" TextAlignment="Right">Right</Text>
    <Text LineSpacing="10">
      Multiple
      Lines
    </Text>
  </StackPanel>
</App>
```

텍스트의 색과 배치, 크기(pt), 줄 간격 등을 지정할 수 있다.

Text 1 실행화면



Font 요소를 써서 텍스트의 폰트를 지정할 수 있다. 먼저 ttf 파일 등 폰트 파일을 다운로드 해야 한다. Roboto-Medium.ttf, Roboto-Light.ttf 등이 폰트를 <http://www.dafont.com/roboto.font>에서 다운로드하고 해당 파일을 프로젝트 디렉토리에 넣었다고 가정한다.

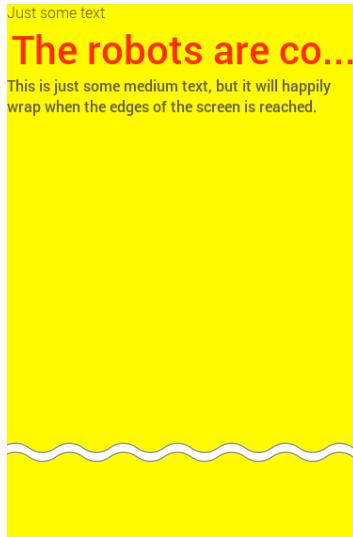
```
<App Background="Yellow">
  <Font File="Roboto-Medium.ttf" ux:Global="Medium" />
  <Font File="Roboto-Light.ttf" ux:Global="Light" />

  <Text ux:Class="Light" Font="Light" />
  <Text ux:Class="Medium" Font="Medium" TextWrapping="Wrap" />
  <Text ux:Class="Warning"
    Font="Medium"
    FontSize="42"
    TextAlignment="Center"
    Color="#f00" />

  <StackPanel>
    <Light>Just some text</Light>
    <Warning>The robots are coming!</Warning>
    <Medium>This is just some medium text, but it will happily wrap
      when the edges of the screen is reached.</Medium>
  </StackPanel>
</App>
```

TextWrapping 속성을 지정해야 텍스트가 길어도 다음 줄에 이어져서 나오게 된다. 지정하지 않으면 화면을 넘어가는 텍스트는 잘린다.

Text 2 실행화면



4. Image 컴포넌트

<https://www.fusetools.com/docs/fuse/controls/image> 참조

이미지를 표시한다. 프로젝트 내의 이미지 파일 또는 웹에 있는 이미지 URL을 지정하여 이미지를 표시할 수 있다.

```
<StackPanel>
  <Image File="some_file.png" />
  <Image Url="some_url" />
</StackPanel>
```

프로젝트 내의 이미지 파일을 보이게 하려면 unoproj 파일에 이미지 파일을 포함시켜야 한다. “이미지 파일:Bundle” 포맷으로 지정한다.

```
"Includes": [
  "*",
  "image.jpg:Bundle"
]
```

만약 모든 이미지를 포함하려면 “*”를 사용하면 된다.

```
"Includes": [
  "*.png:Bundle"
]
```

자바스크립트에서 이미지 파일 이름을 지정한 후, module.exports를 사용해서 Image 컴포넌트에 할당할 수 있다.

```
<JavaScript>
module.exports = {
  image: "image.jpg"
};
</JavaScript>
<Image File="{image}" />
```

5. Video 컴포넌트

<https://www.fusetools.com/docs/fuse/controls/video> 참조

비디오를 파일이나 웹 URL로 지정하여 플레이 할 수 있다. 지원하는 비디오 포맷은 모바일 운영체제에 따라 다르다.

중요한 속성은 아래와 같다.

- Volume: 0.0 ~ 1.0 사이. 기본은 1.0
- Duration: 초 단위로 비디오의 플레이 시간을 표시
- Position: 현재 비디오의 플레이 위치를 초 단위로 반환
- IsLooping: 비디오가 무한 루프로 재생되는 여부를 지정. 기본값은 false

nature.mp4란 비디오 파일이 프로젝트 안에 존재한다고 가정한다. Play, Pause 버튼을 누르면 재생하거나 멈춘다. <ProgressAnimation>을 사용하면 영상 재생의 정도를 표시할 수 있다.

```
<App>
  <DockPanel>
    <Video ux:Name="video" Dock="Fill" File="nature.mp4"
```

```
        IsLooping="true" StretchMode="UniformToFill">
    <ProgressAnimation>
        <Change progressBar.Width="100" />
    </ProgressAnimation>
</Video>
<Rectangle ux:Name="progressBar" Dock="Bottom" Fill="#f00"
    Width="0%" Height="10" />
<Grid Dock="Bottom" ColumnCount="2" RowCount="1">
    <Button Text="Play">
        <Clicked>
            <Resume Target="video" />
        </Clicked>
    </Button>
    <Button Text="Pause">
        <Clicked>
            <Pause Target="video" />
        </Clicked>
    </Button>
</Grid>
</DockPanel>
</App>
```

Video 실행 화면



6. Button 컴포넌트

<https://www.fusetools.com/docs/fuse/controls/button> 참조

퓨즈의 기본 버튼을 표시한다. 기본 모양은 파란색 텍스트가 들어가 있는 투명한 사각형이다.

```
<Button Text="Click me" />
```

겉으로 보기에는 단순 텍스트로 보일 수 있다. 따라서 버튼의 모양을 배경색이

있는 사각형으로 변경하려면 컴포넌트 클래스 타입으로 새롭게 버튼을 정의해야 한다.

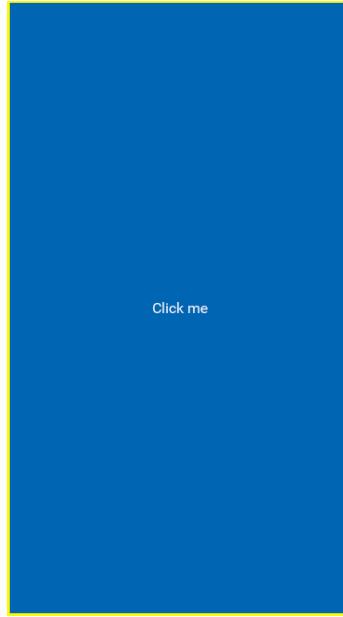
```
<App Background="Yellow">
  <Panel ux:Class="MyButton" HitTestMode="LocalBounds" Margin="4"
        Color="#25a">
    <string ux:Property="Text" />
    <Text Value="{ReadProperty Text}" Color="#fff" Alignment="Center"
          Margin="30,15" />

    <WhilePressed>
      <Change this.Color="#138" Duration="0.05" DurationBack=".2" />
    </WhilePressed>
  </Panel>

  <MyButton Text="Click me" />
</App>
```

Panel 타입인 MyButton 컴포넌트 클래스를 정의하여 새롭게 사각형 모양의 버튼을 정의하였다. Click Me라는 MyButton 컴포넌트를 생성하면 글자가 중앙에 배치되며 버튼을 눌렀을 때 색이 #138로 잠시 동안 변경된다.

Button 예제 화면



7. TextInput 컴포넌트

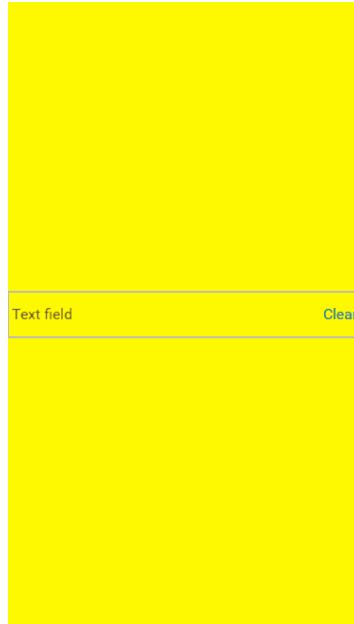
<https://www.fusetools.com/docs/fuse/controls/textinput> 참조

한 줄 정도의 텍스트를 입력할 수 있는 컴포넌트이다. 주로 사용자 이름, 비밀번호, 이메일, 검색어 등을 입력할 때 사용된다. 여러 줄의 텍스트를 입력할 때는 TextView를 사용하고, 모바일 앱의 운영체제에서 사용되는 텍스트 입력 창의 모습을 그대로 원하면 TextBox를 사용한다.

```
<App Background="Yellow">
  <Panel>
    <Button Alignment="CenterRight" Text="Clear" Margin="5">
      <Clicked>
        <Set text.Value="" />
      </Clicked>
    </Button>
    <TextInput ux:Name="text" PlaceholderText="Text field"
      PlaceholderColor="#ccc" Height="50" Padding="5" >
      <Rectangle Layer="Background">
        <Stroke Width="2" Brush="#BBB" />
      </Rectangle>
    </TextInput>
  </Panel>
</App>
```

텍스트 입력 창에 텍스트를 입력한 후, Clear 버튼을 누르면 `<Set text.Value="" />`에 따라 text가 가리키는 TextInput의 화면에 보여지는 텍스트와 Value가 빈 문자열로 채워지면서 지워진다.

TextInput 예제 화면 1



```
<App Background="Yellow">
  <!-- Subclassing TextInput -->
  <TextInput ux:Class="MyTextInput" FontSize="20"
    PlaceholderColor="#ccc" Padding="5">
    <Rectangle Layer="Background" CornerRadius="3">
      <Stroke Width="1" Color="#ccc" />
      <SolidColor Color="White" />
    </Rectangle>
  </TextInput>

  <!-- Example usage -->
  <StackPanel Margin="10" ItemSpacing="10">
    <MyTextInput PlaceholderText="Username" />
    <MyTextInput PlaceholderText="Password" IsPassword="true" />
    <MyTextInput PlaceholderText="Repeat password" IsPassword="true" />
    <MyTextInput />
  </StackPanel>
</App>
```

TextInput을 동일한 모양으로 여러 개를 만들 때는 TextInput 타입의 새로운 컴포넌트 클래스를 정의하는 것이 좋다. 폰트 크기가 20이고 모서리가 둥근 TextInput인 MyTextInput을 새롭게 정의하여 4개를 만들었다.

TextInput 예제 화면 2



8. TextView 컴포넌트

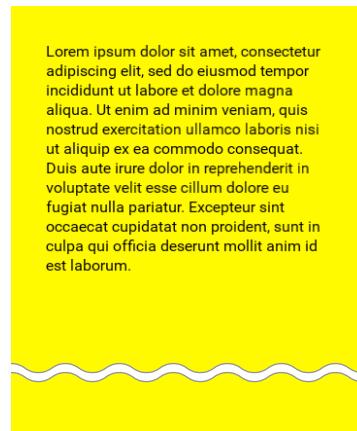
<https://www.fusetools.com/docs/fuse/controls/textview> 참조

여러 줄의 텍스트를 입력할 수 있는 컴포넌트이다. 많은 양의 텍스트를 보여주거나 편집할 때 사용한다.

```
<App Background="Yellow">
    <TextView ux:Class="TextViewer" TextWrapping="Wrap" Padding="20"
        Margin="20" TextColor="#000" CaretColor="#000" >
        Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
        eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
        ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
        aliquip ex ea commodo consequat. Duis aute irure dolor in
        reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
        pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
        culpa qui officia deserunt mollit anim id est laborum.
    </TextView>
    <TextViewer />
</App>
```

텍스트가 많아서 한 행에 다 못 보여주면 TextWrapping 속성 값이 Wrap이므로 다음 줄에 보여준다. TextColor는 검정색, CaretColor(커서의 색깔)도 검정색이다.

TextView 예제 화면



9. Slider 컴포넌트

<https://www.fusetools.com/docs/fuse/controls/slider> 참조

최소 값과 최대 값 사이의 값을 선택할 수 있는 슬라이더 컴포넌트이다. 모바일 기기의 운영체제에서 제공하는 슬라이드 모양을 표시하려면 <NativeView Host> 태그로 Slider를 포함해야 한다.

```
<App Background="Yellow">
  <StackPanel>
    <Slider Value="50" Minimum="0" Maximum="100" />
    <NativeViewHost>
      <Slider Value="0.25" Minimum="0" Maximum="1" />
    </NativeViewHost>
  </StackPanel>
</App>
```

Slider 예제 화면



10. Switch 컴포넌트

<https://www.fusetools.com/docs/fuse/controls/switch> 참조

On/Off를 표시하는 컴포넌트이다. 모바일 기기의 운영체제에서 제공하는 슬라이드 모양을 표시하려면 <NativeViewHost> 태그로 Switch를 포함해야 한다.

```
<App Background="Yellow">
  <StackPanel>
    <Switch ux:Name="sw" Margin="20">
      <WhileTrue Value="{ReadProperty sw.Value}">
        <DebugAction Message="Switch.Value = true" />
      </WhileTrue>
    </Switch>
  </StackPanel>
</App>
```

```
</Switch>
<NativeViewHost>
    <Switch />
</NativeViewHost>
</StackPanel>
</App>
```

Switch 값이 참이 되면 DebugAction이 실행되면서 퓨즈 스튜디오 콘솔에 “Switch.Value = true”란 메시지가 출력된다.

Switch 예제 화면



11. WebView 컴포넌트

<https://www.fusetools.com/docs/fuse/controls/webview> 참조

iOS와 Android 기기에서 웹 뷰를 실행하는 컴포넌트이다. 웹 URL 또는 HTML 파일을 지정하면 웹 브라우저에서 보여지는 것처럼 해당 파일을 표시한다. 웹 뷰는 iOS와 Android 기기에서 기본적으로 제공하는 웹 뷰 기능을 사용해야 하므로 <NativeViewHost> 태그 안에 <WebView>를 반드시 포함해야 한다.

```
<App Background="#333">
    <JavaScript>
        module.exports = {
            onPageLoaded : function(res) {
                console.log("WebView arrived at "+ JSON.parse(res.json).url);
            }
        };
    </JavaScript>
    <DockPanel>
        <StatusBarBackground Dock="Top"/>
        <NativeViewHost>
            <WebView Dock="Fill" Url="http://www.google.com">
                <PageLoaded>
                    <EvaluateJS Handler="{onPageLoaded}">
                        var result = {
                            url : document.location.href
                        };
                        return result;
                    </EvaluateJS>
                </PageLoaded>
            </WebView>
        </NativeViewHost>

        <BottomBarBackground Dock="Bottom" />
    </DockPanel>
</App>
```

WebView는 데스크톱 컴퓨터에서 프리뷰가 불가능하며 앱을 모바일 기기에 설치한 후 동작과정을 확인할 수 있다.

03.6

화면 레이아웃

레이아웃(layout)은 화면에 요소의 크기와 배치를 결정하는 방법이다. 앱은 다양한 비율과 해상도의 모바일 기기에서 동작하기 때문에 그들 모두에 잘 어울릴 수 있는 레이아웃을 설계해야 한다.

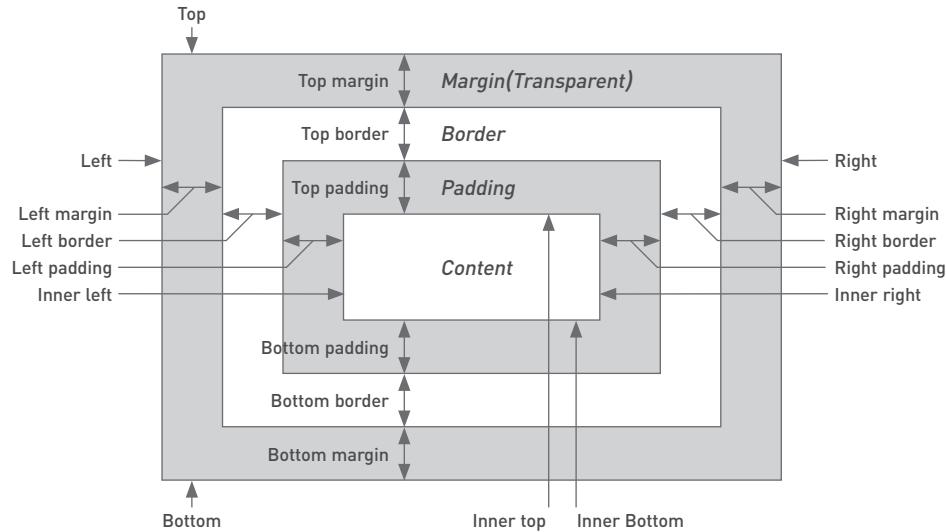
퓨즈는 기본적으로 Panel 컴포넌트에 고유의 레이아웃 규칙을 적용하여 컴포넌트들을 배치한다.

1. 기본 개념

1) 여백(Margin)과 패딩(Padding)

퓨즈의 화면 컴포넌트들은 여백과 패딩 값이 존재한다. 이는 웹의 CSS와 유사하다.

여백, 패딩 다이어그램



화면 요소는 내용을 보여주는 콘텐츠(Content) 영역, 요소의 경계영역인 경계선(Border) 영역과 콘텐츠 영역 사이의 공간인 패딩(Padding) 영역, 다른 화면 요소와의 간격인 여백(Margin)으로 구성된다.

화면 요소 간의 여백과 패딩



패딩과 여백은 왼쪽인 Left, Top, Right, Bottom 등 4개의 방향에 따라 다르게 설정할 수 있다.

```
<Panel Margin="10,20,30,40" />  
<Panel Padding="10,20,30,40" />
```

위 패널의 Margin, Padding은 각각 “10(left), 20(top), 30(right), 40(bottom)”의 값을 나타낸다.

```
<Panel Margin="10" /> <!-- 동일한 의미 : "10,10,10,10" -->  
<Panel Padding="10,20" /> <!-- 동일한 의미 : "10,20,10,20" -->
```

위의 예에서처럼 값을 축약해서 표시할 수도 있다.

2) 정렬(Alignment)

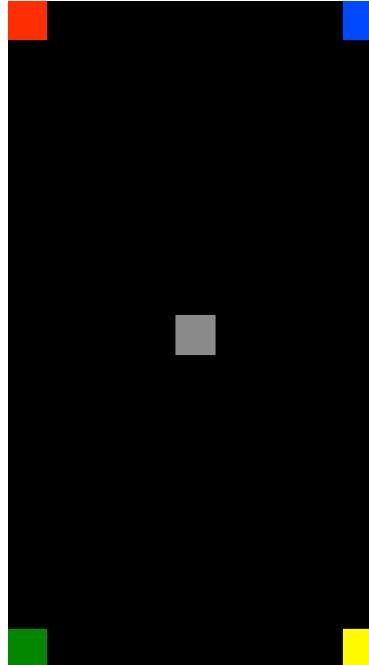
다른 컴포넌트를 포함할 수 있는 컴포넌트 안에서는 정렬 방법에 따라 포함되는 컴포넌트의 위치가 결정된다.

- 기본: 아무 값도 지정하지 않을 경우, 컴포넌트는 모든 방향으로 확장된다.
- Left: 왼쪽
- HorizontalCenter: 수평 중앙
- Right: 오른쪽
- Top: 상단
- VerticalCenter: 수직 중앙
- Bottom: 바닥

- TopLeft: 상단 왼쪽
- TopCenter: 상단 중앙
- TopRight: 상단 오른쪽
- CenterLeft: 중앙 왼쪽
- Center: 수평, 수직 중앙
- CenterRight: 중앙 오른쪽
- BottomLeft: 바닥 왼쪽
- BottomCenter: 바닥 중앙
- BottomRight: 바닥 오른쪽

```
<App Background="Black">
  <Panel>
    <Rectangle Alignment="TopLeft" Width="40" Height="40" Color="Red" />
    <Rectangle Alignment="TopRight" Width="40" Height="40" Color="Blue" />
    <Rectangle Alignment="Center" Width="40" Height="40" Color="Gray" />
    <Rectangle Alignment="BottomLeft" Width="40" Height="40" Color="Green" />
    <Rectangle Alignment="BottomRight" Width="40" Height="40" Color="Yellow" />
  </Panel>
</App>
```

정렬의 예



3) 너비(Width)와 높이(Height)

각 컴포넌트의 너비와 높이는 기본적으로 레이아웃 타입에 따라 자동으로 정해진다. 하지만 컴포넌트의 Width, Height 속성 값을 구체적으로 설정하면 상세하게 크기를 컨트롤 할 수 있다.

2. 레이아웃 타입

Dock, Stack, Grid 등 3개의 레이아웃 타입이 많이 사용된다.

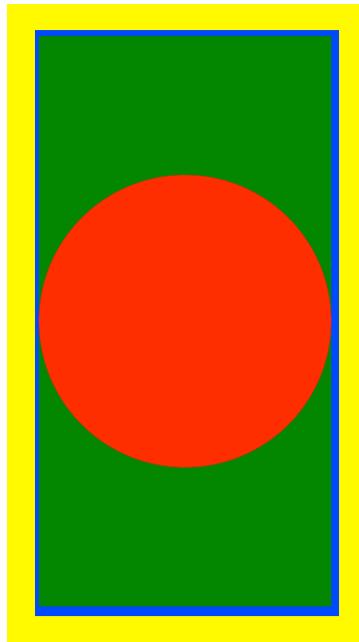
1) Panel

Panel은 다른 컴포넌트들을 포함하고 특정 레이아웃으로 그들을 배치하는 컴포넌트이다. Panel에 포함되는 컴포넌트는 레이아웃이 지정되지 않으면 기본적으로 Panel 전체 영역의 크기를 차지하고, 여러 개의 컴포넌트를 삽입하면 한 컴포넌트가 다른 컴포넌트 위에 오버레이 되어 보여진다.

```
<App Background="Yellow">
  <Panel Margin="30" Padding="4,6,8,10" Color="Blue">
    <Circle Color="Red" />
    <Rectangle Color="Green" />
  </Panel>
</App>
```

앱의 배경색은 노란색이다. Panel은 Margin이 30이므로 30pt만큼 앱의 경계선에서 떨어지고 Padding 만큼 내부에 포함하는 Circle, Rectangle이 떨어져서 보이게 된다. Circle, Rectangle이 차례대로 생성이 되는데 퓨즈는 처음 컴포넌트를 제일 상단에 보여준다. 따라서 Rectangle이 먼저 보이고 그 위에 Circle이 오버레이된다.

Panel 예제 화면



2) StackPanel

StackLayout이 적용된 Panel을 StackPanel이라 한다. 원래 StackLayout은 아래처럼 설정한다.

```
<Panel>
  <StackLayout />
  ... // 컴포넌트들
</Panel>
```

위의 코드를 단순화하려면 StackPanel을 사용하면 된다.

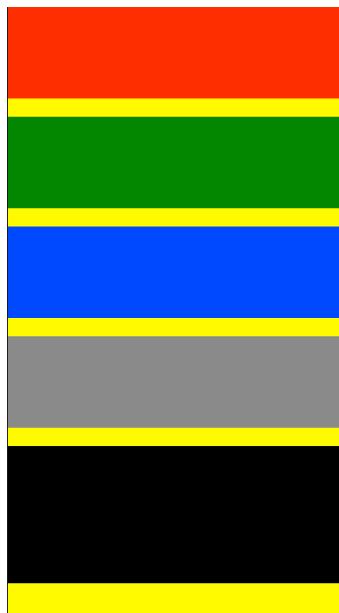
```
<StackPanel>
  ... // 컴포넌트들
</StackPanel>
```

StackPanel은 컴포넌트를 수평 또는 수직 방향으로 차례대로 배치한다. 기본 방향은 수직 방향(위에서 아래)지만 Orientation 속성 값을 조정하면 수평 방향으로 변경할 수 있다.

```
<App Background="Yellow">
  <StackPanel ItemSpacing="20">
    <Panel Height="100" Background="Red"/>
    <Panel Height="100" Background="Green"/>
    <Panel Height="100" Background="Blue"/>
    <Panel Height="100" Background="Gray"/>
    <Panel Height="150" Background="Black"/>
  </StackPanel>
</App>
```

StackPanel에 ItemSpacing 속성 값을 설정하면 포함되는 컴포넌트 사이의 간격을 pt로 벌릴 수 있다.

StackPanel 예제 화면



3) DockPanel

DockLayout이 적용된 Panel을 DockPanel이라 한다. 원래 DockLayout은 아래처럼 설정한다.

```
<Panel>
  <DockLayout />
  ...
  // 컴포넌트들
</Panel>
```

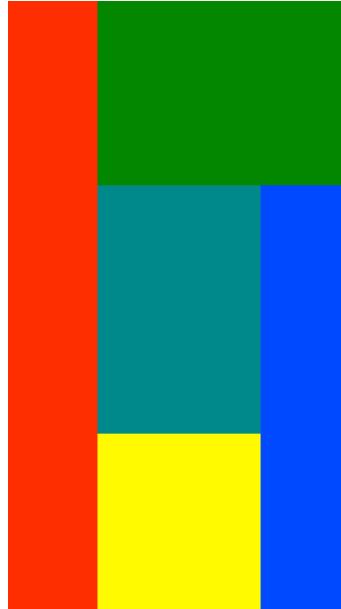
위의 코드를 단순화하려면 DockPanel을 사용하면 된다.

```
<DockPanel>
  ...
  // 컴포넌트들
</DockPanel>
```

DockPanel은 컴포넌트를 상, 하, 좌, 우 등 서로 다른 방향으로 배치한다. 포함되는 컴포넌트의 Dock 속성에 Top, Left, Right 등의 방향 값을 넣어서 배치한다.

```
<App Background="Yellow">
  <DockPanel>
    <Rectangle ux:Class="MyRectangle" MinWidth="100" MinHeight="200" />
    <MyRectangle Color="Red" Dock="Left"/>
    <MyRectangle Color="Green" Dock="Top"/>
    <MyRectangle Color="Blue" Dock="Right"/>
    <MyRectangle Color="Yellow" Dock="Bottom"/>
    <MyRectangle Color="Teal" />
  </DockPanel>
</App>
```

DockPanel 실행화면



먼저 빨간색 MyRectangle 컴포넌트를 Left 즉, 왼쪽에 배치한다. 그 다음 Green 색인 MyRectangle을 상단, Blue를 오른쪽, Yellow를 하단으로 하고 나머지 공간에는 마지막 Teal이 할당된다.

4) Grid

표처럼 동일한 크기의 행, 열로 이루어진 공간에 컴포넌트를 배치한다.

```
<Grid RowCount="4" ColumnCount="2"/>
```

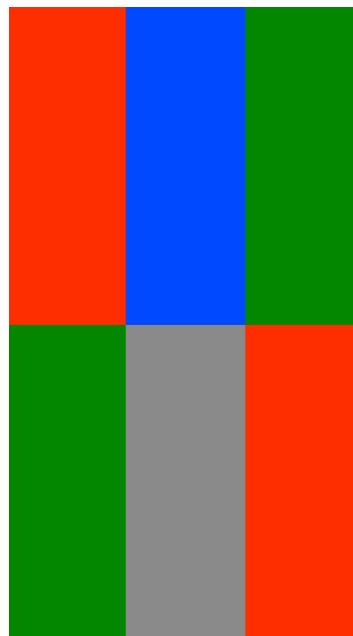
행과 열의 개수는 RowCount, ColumnCount 속성 값으로 지정한다.

```
<App Background="Yellow">
  <Grid RowCount="2" ColumnCount="3">
    <Rectangle Row="0" Column="0" Color="Red"/>
    <Rectangle Row="0" Column="1" Color="Blue"/>
    <Rectangle Row="0" Column="2" Color="Green"/>

    <Rectangle Row="1" Column="0" Color="Green"/>
    <Rectangle Row="1" Column="1" Color="Gray"/>
    <Rectangle Row="1" Column="2" Color="Red"/>
  </Grid>
</App>
```

Row가 0이면 첫 번째 행, 1이면 두 번째 행을 나타낸다. Column도 마찬가지다.

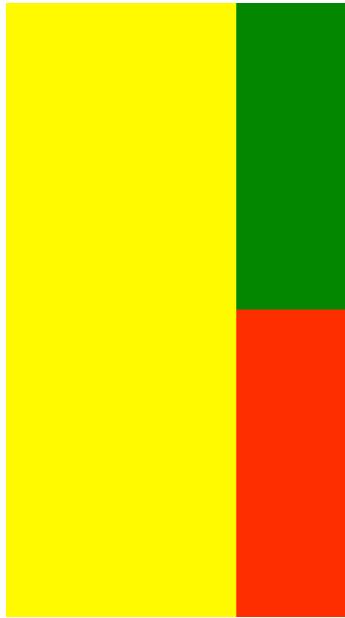
Grid 예제 화면 1



```
<App>
  <Grid RowCount="2" ColumnCount="3">
    <Rectangle ColumnSpan="2" RowSpan="2" Color="Yellow"/>
    <Rectangle Row="0" Column="2" Color="Green"/>
    <Rectangle Row="1" Column="2" Color="Red"/>
  </Grid>
</App>
```

ColumnSpan이 2이면 두 개의 컬럼, RowSpan이 2이면 두 개의 행을 통합하여 노란색으로 표시한다. 그리고 나머지 첫 번째 행의 세 번째 셀은 녹색, 두 번째 행의 세 번째 셀은 빨간색으로 나타낸다.

Grid 예제 화면 2



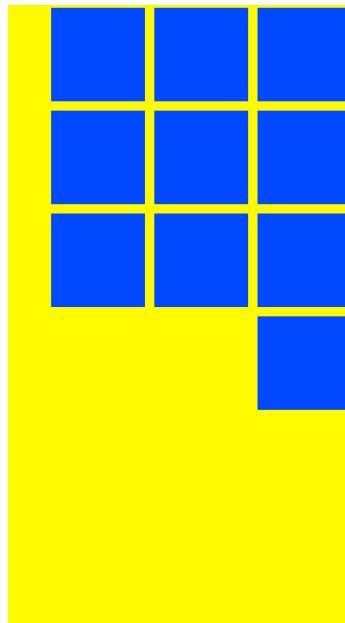
5) WrapPanel

WrapPanel은 특정 방향으로 컴포넌트를 배치하다가 공간이 부족하면 다음 공간에서 지정한 방향으로 할당하는 패널이다.

```
<App Background="Yellow">
    <WrapPanel FlowDirection="RightToLeft">
        <Each Count="10">
            <Rectangle Margin="5" Width="100" Height="100" Color="Blue"/>
        </Each>
    </WrapPanel>
</App>
```

WrapPanel의 컴포넌트 배치 방향이 RightToLeft, 즉 오른쪽에서 왼쪽이므로 포함하는 컴포넌트들을 오른쪽에서 왼쪽에 배치하다가 공간이 부족하면 아래 공간으로 배치한다. <Each Count="10">은 Rectangle을 10번 반복해서 생성한다는 뜻이다. 너비와 높이가 100pt인 사각형을 오른쪽에서 왼쪽으로 배치하고 공간이 부족하면 다음 줄에서 오른쪽으로 왼쪽으로 배치한다.

WrapPanel 예제 화면



03.7

화면 내비게이션

화면 전환을 하기 위해서는 Router, Navigator, PageControl 등 3개의 개념을 잘 이해해야 한다.

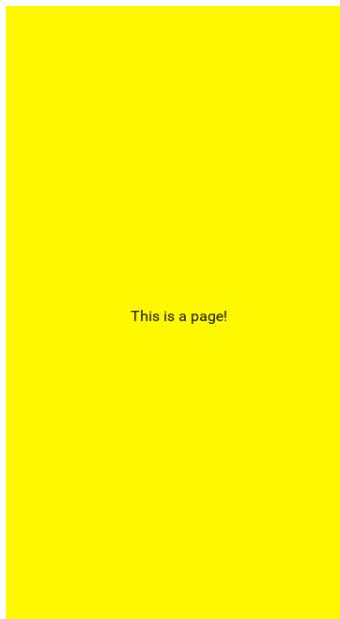
1. Page와 PageControl

내비게이션에 참여하는 각 화면을 페이지라고 하며 보통 Page 컴포넌트를 사용 한다.

```
<App Background="Yellow">
  <Page ux:Class="MyPage">
    <Text Alignment="Center">This is a page!</Text>
  </Page>

  <PageControl>
    <MyPage />
    <MyPage />
  </PageControl>
</App>
```

PageControl 예제 화면



Page 태입인 MyPage 컴포넌트를 생성하고 PageControl 안에 포함시키면 모바일 기기 화면을 좌우로 스와이프(Swipe)했을 때 페이지가 전환된다.

```
<App Background="Yellow">
  <JavaScript>
    module.exports = {
      gotoPage1: function() { router.goto("page1"); },
      gotoPage2: function() { router.goto("page2"); },
      gotoPage3: function() { router.goto("page3"); }
    };
  </JavaScript>

  <Router ux:Name="router" />

  <PageControl>
    <Panel ux:Name="page1" Color="#e74c3c" Clicked="{gotoPage2}" />
    <Panel ux:Name="page2" Color="#2ecc71" Clicked="{gotoPage3}" />
    <Panel ux:Name="page3" Color="#3498db" Clicked="{gotoPage1}" />
  </PageControl>
</App>
```

Page는 Router에 의해 이동될 수도 있다. 첫 번째 페이지인 page1을 클릭하면 gotoPage2() 자바스크립트 함수가 실행이 되어 page2로 이동한다. 두 번째 페이지인 page2를 클릭하면 gotoPage3() 자바스크립트 함수가 실행이 되어 page3으로 이동한다.

2. Router와 Navigator

Router는 PageControl, Navigator와 함께 사용이 되며 3가지 핵심 기능을 제공한다.

- goto(페이지 이름, [추가 데이터 객체]): 특정 페이지로 바로 이동한다. 그 페이지로 추가 데이터를 전달할 수도 있다.
- push(페이지 이름, [추가 데이터 객체]): 현재 페이지를 임시로 저장하고 특정 페이지로 이동한다. 그 페이지로 추가 데이터를 전달할 수도 있다.
- goBack(): push로 저장된, 직전의 임시 저장된 페이지로 이동한다.

Navigator는 템플릿(Template) 태입의 컴포넌트를 포함하며 필요시에 컴포넌트를 생성, 화면에 표시하고 특정 이벤트가 발생하면 Router를 사용하여 화면 페이지를 이동한다.

```
<App Background="Yellow">
  <JavaScript>
    module.exports = {
      gotoFirst: function() { router.goto("firstPage"); },
      gotoSecond: function() { router.goto("secondPage"); }
    };
  </JavaScript>

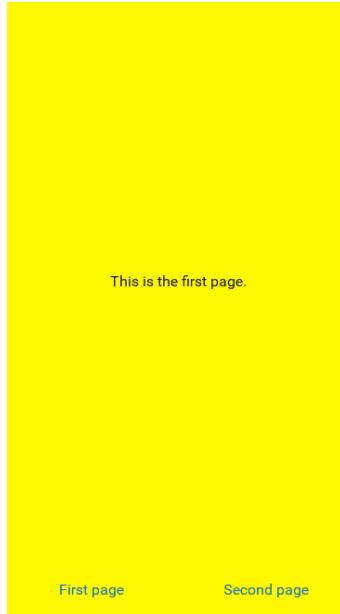
  <Router ux:Name="router" />
```

```
<DockPanel>
    <Navigator DefaultTemplate="firstPage">
        <Page ux:Template="firstPage">
            <Text Alignment="Center">This is the first page.</Text>
        </Page>
        <Page ux:Template="secondPage">
            <Text Alignment="Center">This is the second page.</Text>
        </Page>
    </Navigator>

    <Grid Dock="Bottom" Columns="1*, 1*">
        <Button Text="First page" Padding="20" Clicked="{gotoFirst}" />
        <Button Text="Second page" Padding="20" Clicked="{gotoSecond}" />
    </Grid>
</DockPanel>
</App>
```

Navigator는 firstPage, secondPage 등 2개의 Page 컴포넌트를 템플릿(Template) 타입으로 포함한다. 그리고 DefaultTemplate="firstPage"에 따라 first Page를 먼저 보여준다. 하단의 First page 버튼을 클릭하면 gotoFirst() 함수가 실행되어 firstPage로 이동한다. Second page를 클릭하면 gotoSecond() 함수가 실행되어 secondPage로 이동한다.

Router 예제 화면



03.8

트리거와 애니메이션

1. 트리거(Trigger)

트리거는 사용자가 앱에서 발생시키는 이벤트, 입력, 제스처, 컴포넌트의 상태 변화 등을 처리하는 객체이다. 보통 트리거는 애니메이션과 액션을 실행한다.

```
<App Background="Yellow">
  <Panel ux:Name="panel" Color="Blue">
    <Tapped>
      <Change panel.Color="Red" Delay="1" Duration="1"
            Easing="CubicOut" />
    </Tapped>
  </Panel>
</App>
```

파란 패널에 손을 탭(Tapped)하면 1초 후에 1초 동안 색이 빨간색으로 변경된다. 이렇게 순간적으로 발생한 이벤트를 처리하는 트리거를 펄스 트리거(Pulse Trigger)라고 부르며 Clicked, Tapped 등이 있다.

```
<App Background="Yellow">
  <Panel Color="Red" ux:Name="panel">
    <WhilePressed>
      <Scale Factor="0.9" Duration="5" Easing="BackOut" />
      <Change panel.Color="Blue" Delay="1" Duration="5"
            Easing="CubicOut" />
    </WhilePressed>
  </Panel>
</App>
```

반면에 이벤트가 계속 발생되고 있는 중에 처리하는 트리거도 있다. 위의 예제는 WhilePressed라는 트리거가 사용되는데, 화면을 계속 누르고 있는 동안 Scale, Change가 실행된다. 이런 트리거를 While 트리거라고 하는데 While Dragging, WhileFocused 등이 있다.

상세 트리거 리스트는 <https://www.fusetools.com/docs/fuse/triggers/trigger>에 있다.

2. 변형하기(Transform)

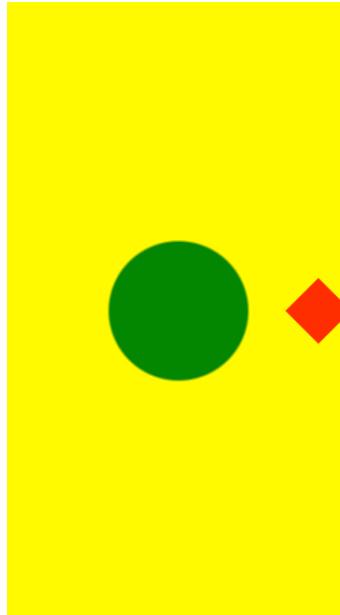
도형의 크기를 변경하거나 위치를 이동, 회전할 수 있다.

```
<App Background="Yellow">
  <Circle Color="Green" Width="50" Height="50">
    <Scaling Factor="3" />
  </Circle>

  <Rectangle Color="Red" Width="50" Height="50">
    <Translation X="150" />
    <Rotation Degrees="45" />
  </Rectangle>
</App>
```

Circle은 너비와 높이가 50이지만 Scaling의 Factor를 3으로 설정하면 3배가 커지게 된다. 따라서 너비와 높이가 150가 된다. Rectangle은 Translation의 X를 150, 즉 현재 위치에서 X축인 오른쪽으로 150만큼 이동한다. 또한 Rotation의 Degrees가 45이므로 사각형이 45도만큼 회전하였다.

변형하기 예제 화면



3. 애니메이션

트리거가 동작할 때 화면 요소가 어떻게 움직이게 될 지 설정하여 애니메이션 효과를 발생시킨다. 아래와 같은 타입을 활용한다.

- Change: 특정 속성을 다른 값으로 변경한다.
- Move: 위치를 이동한다.
- Rotate: 회전한다.
- Scale: 크기를 변경한다.

```
<App Background="Yellow">
  <Panel ux:Name="panel1" Color="Blue">
    <WhilePressed>
      <Change panel1.Color="#0f0" Duration="1" />
      <Move X="100" Delay="1" Duration="1" />
    </WhilePressed>
  </Panel>
</App>
```

패널을 WhilePressed, 즉 누르고 있으면 panel1, 즉 현재 패널의 색이 Duration 1초동안 0f0으로 변경되고, 그 다음 패널의 x축의 값이 100만큼 오른쪽으로 이동된다. 이때 Delay가 있으므로 1초 동안 기다렸다고 움직인다.

Duration, Delay 속성 값을 통해 애니메이션의 지속 시간 및 타이밍을 조절 한다.

04

자바스크립트와 데이터

- 04.1 자바스크립트의 역할
- 04.2 UX 화면과 데이터를 바인딩하기
- 04.3 이벤트 처리하기
- 04.4 Observable의 파워

04.1

자바스크립트의 역할

자바스크립트는 가장 인기 있는 프로그래밍 언어 중 하나다. 웹 브라우저에서 간단한 로직을 구현할 때 사용하며, 최근에는 서버에서도 Node.js 등을 사용해 구현한 로직을 직접 동작시키는 것 또한 가능해졌다. 퓨즈에서 로직을 처리하기 위해 사용하는 프로그래밍 언어는 자바스크립트다. 퓨즈에서 UX 마크업 언어 가 화면 구현을 담당한다면 자바스크립트는 이벤트가 발생했을 때 특정 기능을 실행하거나 UX 화면과 데이터를 연결하는 등의 로직을 책임진다.

```
<App>
  <JavaScript>
    module.exports.foo = "bar";
  </JavaScript>
  <Panel>
    <Text Value="{foo}" />
  </Panel>
</App>
```

자바스크립트 코드는 <JavaScript> 태그 안에 정의할 수 있다.

```
<App>
  <JavaScript File="Main.js"/>
  <Panel>
    <Text Value="{foo}" />
  </Panel>
</App>
```

또는 JavaScript 태그에 File 속성 값을 지정하면 별도의 파일에 자바스크립트 코드를 저장할 수 있다. 위 예에서는 Main.js가 그것이다.

자바스크립트 코드는 꼭 <App> 안에서만 정의할 필요는 없다. ux:Class, 즉 컴포넌트 클래스를 정의할 때도 정의할 수 있다.

```
<Panel ux:Class="MyComponent">
  <JavaScript>
    var Observable = require("FuseJS/Observable");
    var foo = Observable(10);
    module.exports = { foo }
  </JavaScript>

  <Text Value="{foo}" />
</Panel>
```

위의 코드는 App이 아니라 Panel 타입인 MyComponent 클래스를 정의할 때 자바스크립트 코드를 포함시킨 것이다. MyComponent 컴포넌트가 생성될 때 그 안에서 자바스크립트가 실행되고 데이터가 유지된다.

`console.log("메시지 내용")`을 자바스크립트 안에 포함하면 퓨즈 스튜디오의 콘솔에 메시지 내용이 출력된다. 디버깅할 때 활용한다.

퓨즈에서 사용되는 자바스크립트는 웹 브라우저에 사용되는 자바스크립트와 용도와 구현 방법이 거의 유사하나, 웹 브라우저의 특정 기능을 호출하는 자바스크립트 코드는 웹 브라우저에서만 동작하도록 되어 있기에 퓨즈에서 정상적으로 동작하지 않을 수도 있다는 점을 참고하였으면 한다.

04.2

UX 화면과 데이터를 바인딩하기

자바스크립트 안에서 정의한 데이터를 UX 화면 안에 표시되게 하는 방법은 간단하다. UX 마크업 언어 안에서 {표현식}을 사용하면 해당 표현식의 계산 값이 화면에 표시된다.

예

- <Text Value="{textKey}" /> : 자바스크립트 변수인 textKey 값이 할당되어 Text 컴포넌트가 표시됨

```
<App>
  <JavaScript>
    module.exports = {
      greeting: "Hello, world!"
    };
  </JavaScript>
  <Text Value="{greeting}" />
</App>
```

이때 주의할 점이 있다. 자바스크립트 변수의 값이 컴포넌트에 전달되어 표시되려면 반드시 자바스크립트 코드 안에 module.exports를 정의하고 그 안에 해당 변수를 등록해야 한다. “Hello, world!”란 텍스트를 포함하는 greeting이란 변수를 module.exports에 등록했으므로 <Text Value=”{greeting}” />의 {greeting}에 “Hello, world!” 값이 할당되어 표시된다. 만약 module.exports가 없다면 값이 컴포넌트에 전달되지 않는다.

```
<App Background="Yellow">
  <JavaScript>
    var mydata = ["1", "2", "3", "4", "5", "6"];
    module.exports = {
      data: mydata
    };
  </JavaScript>
  <StackPanel>
    <Each Items="{data}">
      <Text Value="{item}" />
    </Each>
  </StackPanel>
</App>
```

여러 개의 데이터를 화면에 연결할 때는 Each 태그를 사용한다. mydata는 6개의 원소를 포함하고 있는데, 화면에 각 원소의 값을 Text 컴포넌트로 표시하기 위해서 6개의 Text 컴포넌트를 직접 생성할 필요가 없다. <Each Items="“{배열 이름}”>을 사용하면 배열의 원소 개수만큼 반복하면서 컴포넌트를 생성한다. 동시에 각 원소의 값을 Text의 Value 속성 안에 할당한다.

Each 반복 예



```
App Background="Yellow">
<JavaScript>
  var mydata = [
    {name: "Hubert Cumberdale", age: 12},
    {name: "Marjory Stewart-Baxter", age: 43},
    {name: "Jeremy Fisher", age: 25}];

  module.exports = {
    data: mydata
  };
</JavaScript>
<StackPanel>
  <Each Items="{data}">
    <DockPanel>
      <Text Dock="Right" Value="{age}" />
      <Text Value="{name}" />
    </DockPanel>
  </Each>
</StackPanel>
</App>
```

물론 배열 원소의 값이 단순 값이 아닌 객체일 수도 있다. mydata 배열에는 name, age 속성이 있는 3개의 객체가 들어있다. <Each Items="“data”> 문장을 통해 각 원소 객체의 age, name 값이 화면에 출력된다.

Each 반복 예

Hubert Cumberdale	12
Marjory Stewart-Baxter	43
Jeremy Fisher	25

04.3

이벤트 처리하기

사용자의 이벤트를 처리하는 자바스크립트 함수를 정의할 수 있다. 이벤트가 발생하는 컴포넌트에 이벤트 핸들러를 등록해야 한다. 예를 들어, Button에 클릭 이벤트가 발생할 때 자바스크립트 함수가 실행되려면 Clicked 속성에 함수 이름을 지정해야 한다. 물론 컴포넌트에서 사용되는 자바스크립트 함수 또한 module.exports에 등록되어야 한다.

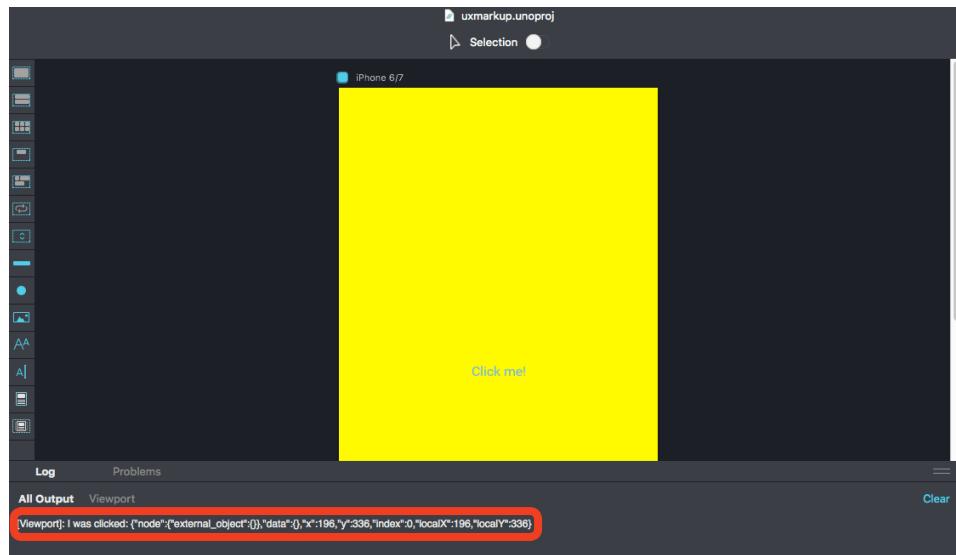
각 컴포넌트에서 처리 가능한 대표적인 이벤트 속성의 이름은 아래와 같다.

- Clicked: 클릭했을 때
- Tapped: 탭했을 때
- Gained: 입력 포커싱이 되었을 때
- KeyPressed: 키보드의 키가 눌러졌을 때
- KeyReleased: 키보드의 키에서 손가락을 놓았을 때

```
<App Background="Yellow">
  <JavaScript>
    module.exports = {
      clickHandler: function (args) {
        console.log("I was clicked: " + JSON.stringify(args));
      }
    };
  </JavaScript>
  <Button Clicked="{clickHandler}" Text="Click me!" />
</App>
```

Button의 Clicked 속성에 clickHandler 함수 이름을 지정했으며, 이 함수는 module.exports에 등록되어 있다. 따라서 Click me! 버튼을 클릭하면 clickHandler() 함수가 실행되면서 console.log를 통해 “I was clicked : “ 메시지가 콘솔에 출력된다. 함수의 파라미터 args에 넘겨지는 값은 이벤트 정보 (x, y 위치 등)를 포함하는 객체이다.

이벤트 핸들러 동작 화면



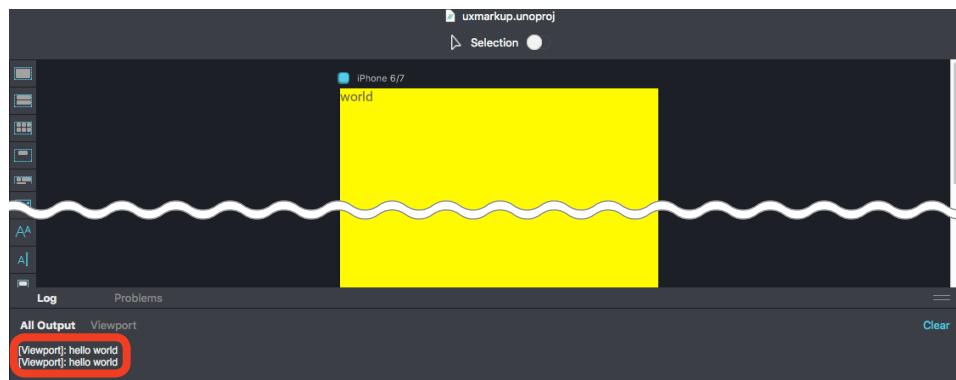
```
<App>
<JavaScript>
var hello = "world";

function writeHello(){
    console.log("hello " + hello);
}

module.exports = {
    hello : hello,
    writeHello : writeHello
};
</JavaScript>
<Panel>
    <Text Value="{hello}" Clicked="{!!writeHello}"/>
</Panel>
</App>
```

hello 변수의 값인 “world”가 Text 컴포넌트의 Value에 전달되어 표시된다. 또한 Clicked 이벤트 속성에 writeHello() 함수가 연결되었기 때문에 클릭하면 콘솔 화면에 “hello world”라는 문자열이 보인다.

실행 화면



04.4

Observable의 파워

자바스크립트 변수 값이 바뀌었을 때 그것과 연결된 모든 컴포넌트의 값이 변경되게 할 수 있다. 그렇게 하려면 변수를 Observable 타입으로 생성해야 한다.

```
var Observable = require("FuseJS/Observable");
```

위와 같이 require를 사용해서 Observable 모듈을 임포트 한 후, Observable() 함수를 호출해서 Observable 객체를 생성한다.

```
var singleValueObservable = Observable(true);
var listObservable = Observable([1,2,3,4]);

var singleValueObservable = Observable(10); //now has .value == 10
singleValueObservable.value = 20;           //now has .value == 20
var theValue = singleValueObservable.value; //theValue is now == 20
```

Observable 객체는 참/거짓, 리스트, 단순 값 등을 포함할 수 있으며, value 속성에 실제 값이 들어가게 된다.

일반 객체와 Observable 객체가 UX 화면에 연결된 것을 비교하여 차이점을 알아보자.

```
<App>
<JavaScript>
var hello = "world";

function writeHello(){
    hello = "world2";          □ ①
    console.log("hello " + hello);
}

module.exports = {
    hello : hello,
    writeHello : writeHello
};
</JavaScript>
<Panel>
    <Text Value="{hello}" Clicked="{writeHello}">
    </Panel>
</App>
```

hello를 표시하는 Text를 클릭했을 때 writeHello() 함수가 실행되면서 hello

변수 값을 world2로 변경한다. (① 코드 참조) 하지만 화면의 “world” 글자는 “world2”로 변경되지 않는다. 계속 “world”가 표시된다. 화면에 연결된 변수가 Observable 객체가 아니면 초기화 이후에 값이 바뀌어도 화면에는 영향이 가지 않는다.

```
<App>
<JavaScript>
var Observable = require("FuseJS/Observable");
var hello = Observable("world"); ②

function writeHello(){
    hello.value = "world2";
    console.log("hello " + hello);
}

module.exports = {
    hello : hello,
    writeHello : writeHello
};
</JavaScript>
<Panel>
    <Text Value="{hello}" Clicked="{writeHello}">
    </Panel>
</App>
```

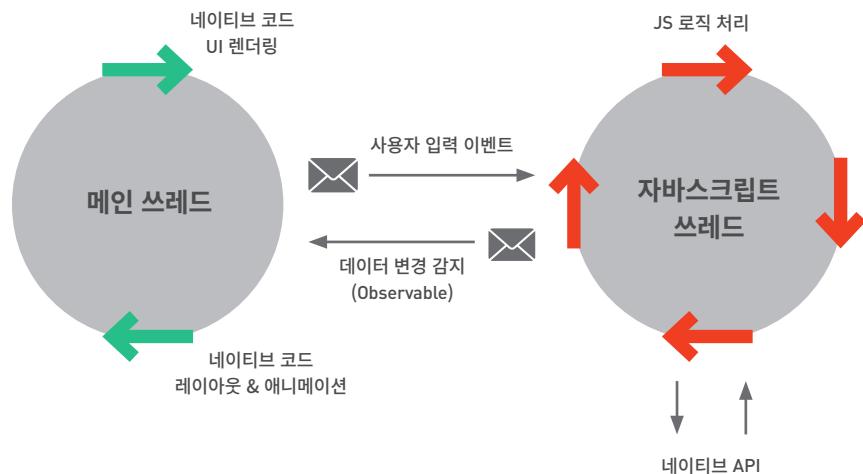
여기서는 hello를 Observable 객체로 만들었다. hello를 표시하는 Text를 클릭했을 때 writeHello() 함수가 실행되면서 hello.value 값이 “world2”로 변경되었다. (② 코드 참조) Observable 객체의 값을 변경할 때는 value 속성의 값을 바꿔야 한다. 그와 동시에 화면의 “world” 글자는 “world2”로 변경된다. 왜냐하면 Observable 객체가 화면에 연결되면 그 값이 변경되었을 때 바로 화면도 업데이트되기 때문이다.

따라서 Observable 객체는 화면에 연결된 자바스크립트 변수 값이 특정 상황에 변경되었을 때 화면이 자동으로 업데이트 되기를 원하는 경우에 사용한다.

참고

Observable을 사용하는 이유는 Fuse가 UI/UX 성능을 최대한으로 발휘하도록 설계한 아키텍처와 관련이 깊다. Fuse는 화면과 관련해 UI/UX를 담당하는 메인 쓰레드와 로직을 처리하는 자바스크립트 쓰레드가 동시에 상호 작용하여 네이티브 코드로 실행 되는 구조로 설계됐다. 사용자의 데이터 입력으로 인한 변경이나 자바스크립트에서의 데이터 변경 등 모든 데이터의 변경 상태를 감지하다 보면 아무래도 앱이 실행되었을 때, 성능이 낮아질 수밖에 없다. 이에 Fuse는 변경 상태 확인이 필요한 데이터에 대해서만 Observable을 사용하도록 권장하여, 메인 쓰레드와 자바스크립트 쓰레드가 각각 최상의 성능을 낼 수 있도록 구조가 설계되었다. 따라서

Observable이 강력하다고 해서 모든 데이터에 대해 Observable을 사용하기 보다는 데이터 변경을 필요로 하는 변수에 대해서만 Observable을 사용하는 것을 권장한다.



05

퓨즈 스튜디오

- 05.1 소개
- 05.2 퓨즈 스튜디오의 화면 뷰와 설정
- 05.3 멀티 프리뷰 모드

05.1

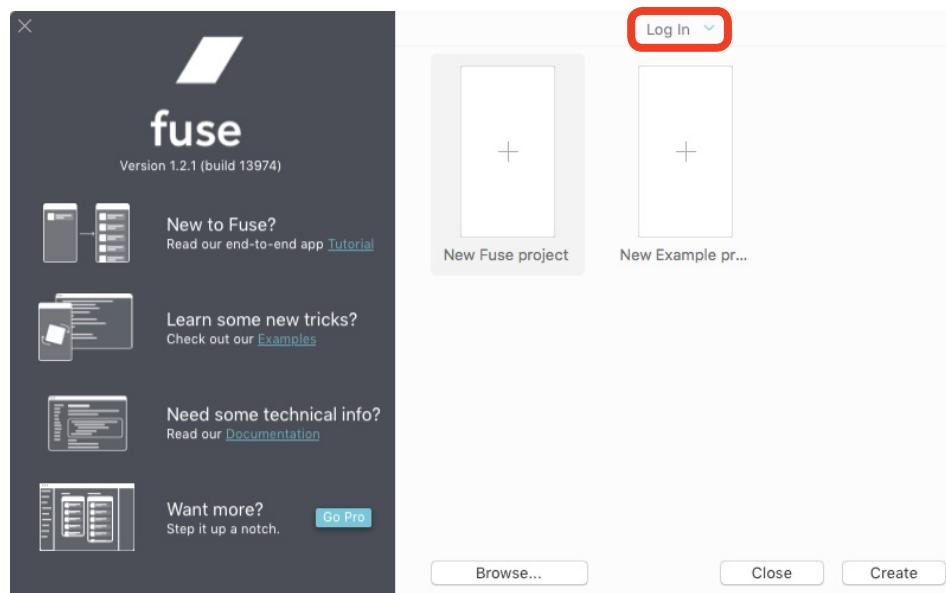
소개

퓨즈 스튜디오(Fuse Studio)는 프로페셔널 플랜에 가입한 사용자에게 제공되는 UI/UX 저작 툴이다. 화면의 레이아웃과 컴포넌트의 계층적 구조를 직관적으로 만들 수 있다.

1. 로그인

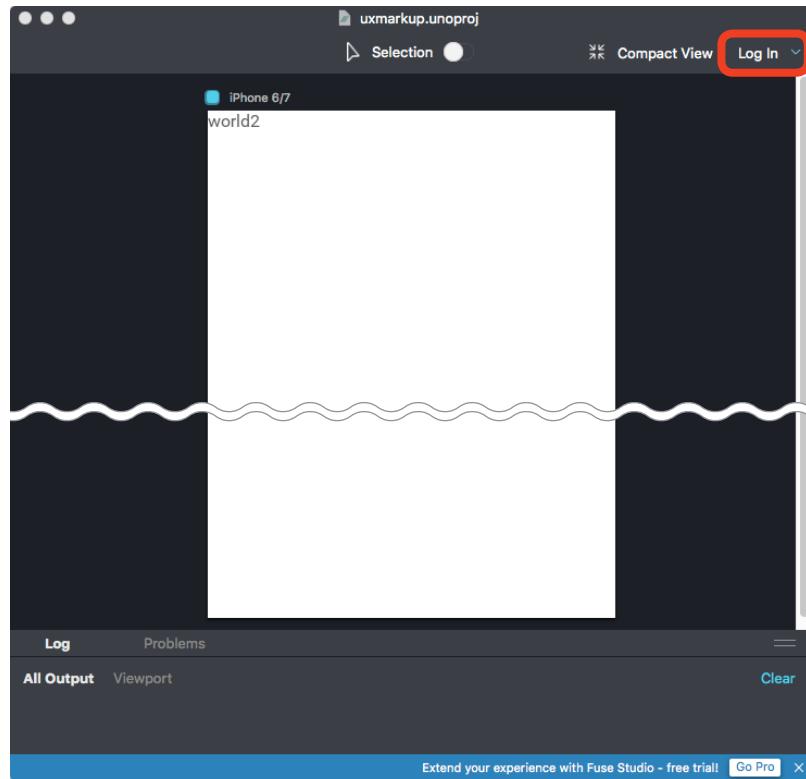
퓨즈 스튜디오를 실행하려면 퓨즈 대시보드에서 프로페셔널 가입 ID, PW를 입력한 후 로그인을 해야 한다.

대시보드에서 로그인하기



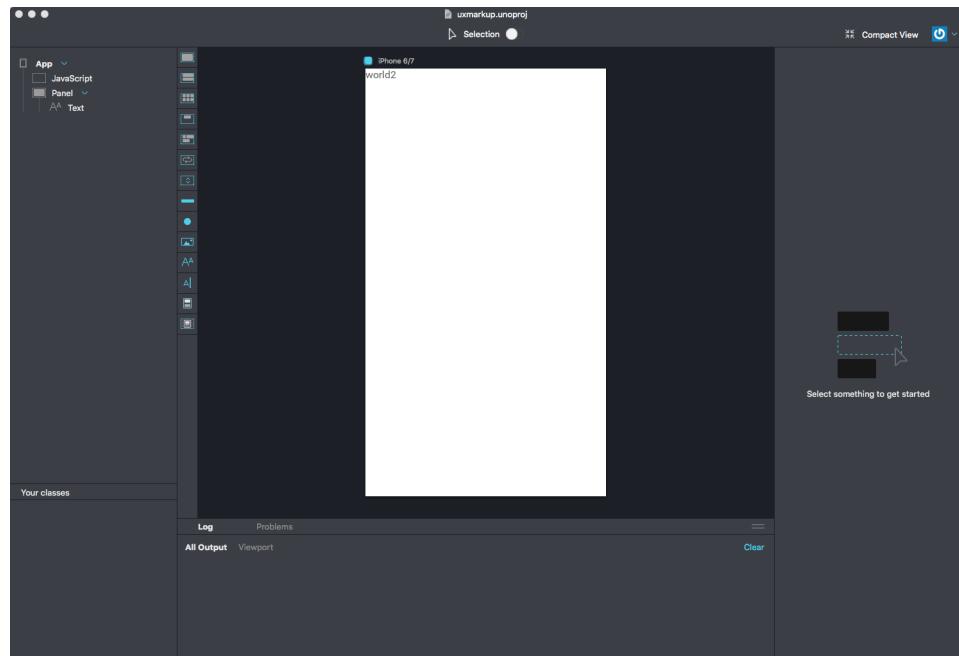
또는 프리뷰 모드에서 바로 로그인을 해도 된다.

프리뷰 모드에서
로그인하기



로그인 후 프리뷰 모드가 실행되면 자동으로 퓨즈 스튜디오가 동작한다.

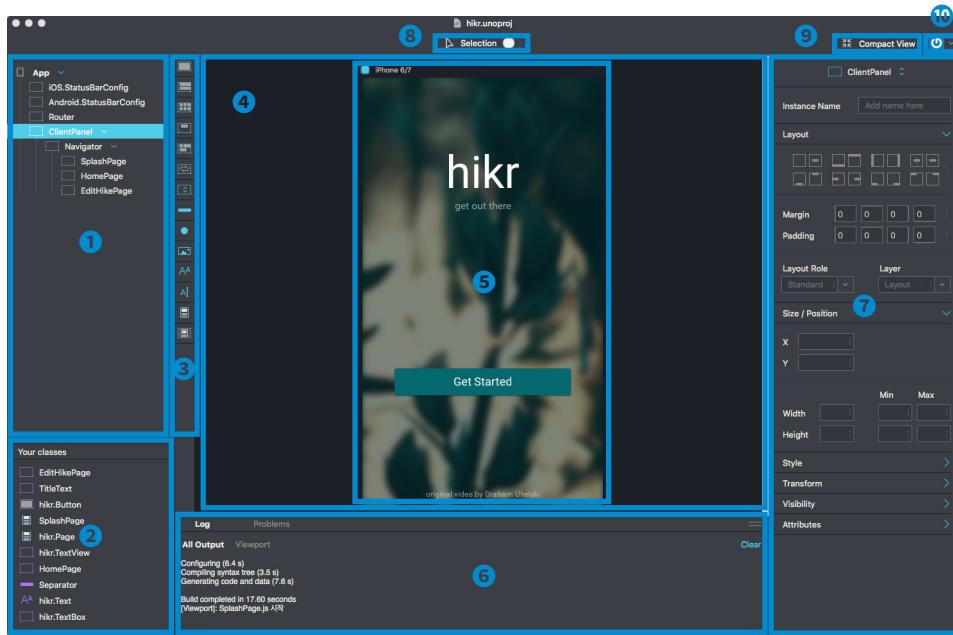
퓨즈 스튜디오 화면



2. UI 구조

퓨즈 스튜디오는 여러 개의 뷰로 구성되어 있다. 각 뷰의 설명은 아래와 같다.

퓨즈 스튜디오의 구조



- 1 UX 계층 뷰: 앱의 컴포넌트 및 요소의 구조를 계층적으로 보여준다.
- 2 클래스 뷰: 클래스로 정의한 컴포넌트의 리스트를 보여준다.
- 3 툴박스: 기본적인 퓨즈 컴포넌트의 리스트가 나열되어 있으며 컴포넌트를 선택하여 UX 계층 뷰에 삽입할 수 있다.
- 4 워크스페이스(Workspace): 앱의 프리뷰 화면, 즉 뷰포트를 보여주고 그 안에서 특정 컴포넌트를 선택할 수 있다. 여러 개 뷰포트를 동시에 보여줄 수 있다.
- 5 뷰포트(Viewport): 디바이스 별 프리뷰 화면 공간을 의미한다. 디폴트는 iPhone 6/7 이다.
- 6 메시지 뷰: 일반적인 로그와 문제 발생 시 에러 메시지 등을 보여준다.
- 7 인스펙터(Inspector): 컴포넌트의 메타 정보를 보여주고 속성 값을 수정할 수 있게 해준다. 속성 값이 수정되면 바로 워크스페이스의 뷰포트에 반영된다.
- 8 선택 모드(Selection mode) 스위치: 프리뷰를 보여주는 뷰포트는 기본적으로는 실행 모드이다. 앱의 동작 과정을 경험할 수 있다. 하지만 앱 내의 각 화면 별 컴포넌트를 선택하고 수정하려면 선택 모드로 변경해야 한다.
- 9 컴팩트 뷰(Compact View) 스위치: 퓨즈 스튜디오 내의 뷰포트와 메시지 뷰만 남기고 나머지를 보이지 않게 할 수 있다. 이렇게 동작하는 퓨즈 스튜디오를 컴팩트 모드로 동작한다고 이야기하며, “컴팩트 뷰” 스위치로 변경한다. 퓨즈 스튜디오의 컴포넌트 수정 기능을 사용하지 않고 주로 서브라임 텍

스트로만 코딩 할 경우에 이 모드를 선택하면 화면이 심플해지므로 간결하게 코딩 할 수 있다.

퓨즈 스튜디오는 크기와 프리뷰 화면의 선택 방법에 따른 2 종류의 작업 모드가 있다.

1) 퓨즈 스튜디오의 크기

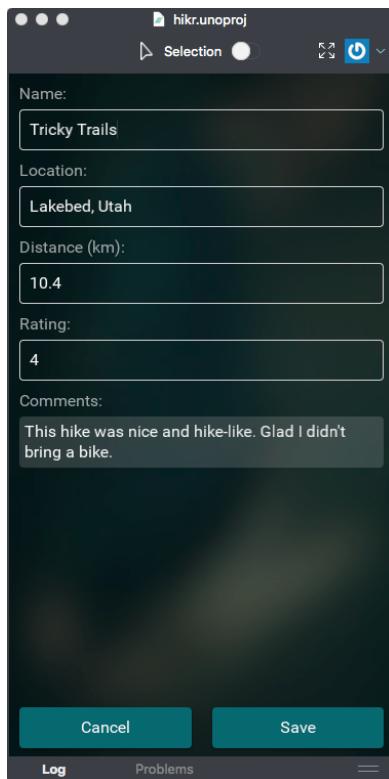
가. 표준 모드

위에서 본 퓨즈 스튜디오가 표준 모드로 동작하는 것이다. 퓨즈 스튜디오는 기본적으로 표준 모드로 실행되며 모든 화면 뷰와 설정 창이 보여진다.

나. 컴팩트 모드

컴팩트 뷰(Compact View) 스위치를 눌러서 뷰포트와 메시지 뷰만 남기고 아래 그림과 같이 나머지를 보이지 않게 할 수 있다.

컴팩트 모드 예



2) 프리뷰 화면의 선택 방법

프리뷰 화면을 마우스로 선택할 때 반응하는 방식에 따라 2개의 모드가 존재한다.

가. 실행 모드(Execution Mode)

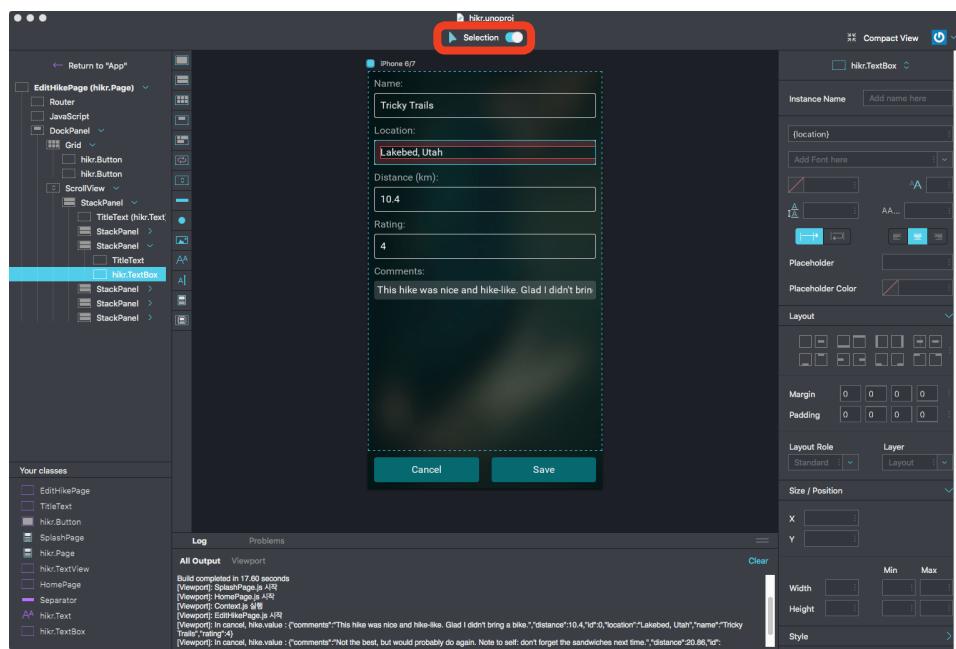
앱의 실행에 참여하는 것을 의미한다. 즉, 버튼을 누르면 해당 버튼에 연결된 기

능이 실행된다. 예를 들어, EditHikePage 화면의 “Cancel” 버튼을 누르면 하이킹 기록 리스트 페이지로 이동한다. 퓨즈 스튜디오는 기본적으로 이 모드로 동작한다.

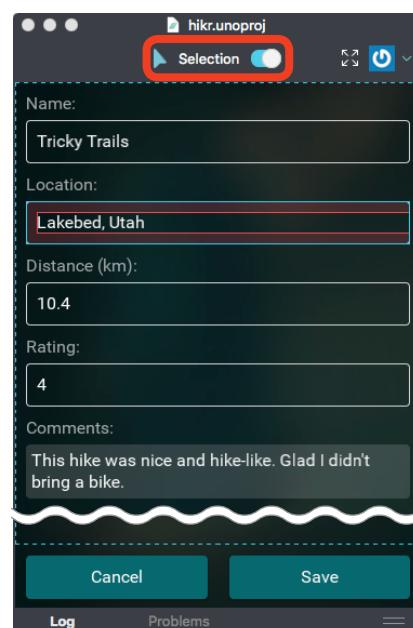
나. 선택 모드(Selection Mode)

프리뷰 화면에 있는 컴포넌트를 마우스로 선택해서 속성 값을 수정하려면 선택 모드로 변경해야 한다. 선택 모드 스위치를 눌러서 선택 모드로 변경하면 화면에 배치된 각 컴포넌트를 선택하여 속성 값을 확인하거나 변경할 수 있다. 컴포넌트를 선택하면 경계선과 마진, 패딩 등이 파란색과 빨간색으로 표시된다.

표준 모드에서의
선택 모드 버튼



컴팩트 모드에서의
선택 모드 버튼

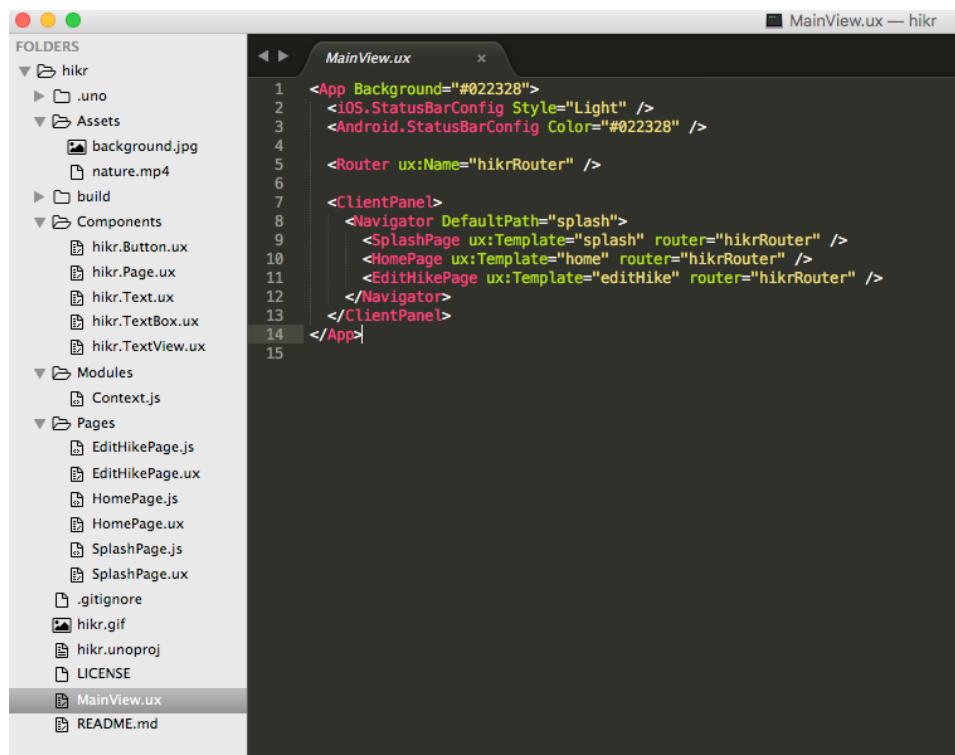


05.2

퓨즈 스튜디오의 화면 뷰와 설정

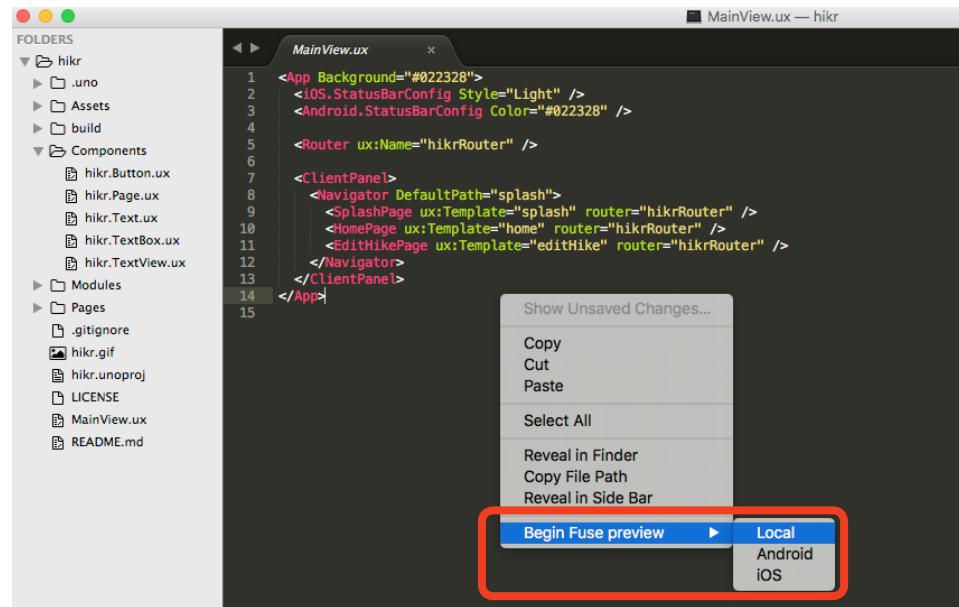
2부의 hikr 앱을 퓨즈 스튜디오에서 로딩하여 살펴보자. hikr 앱은 아래와 같은 파일 구조로 개발되었다.

hikr 프로젝트의 구조



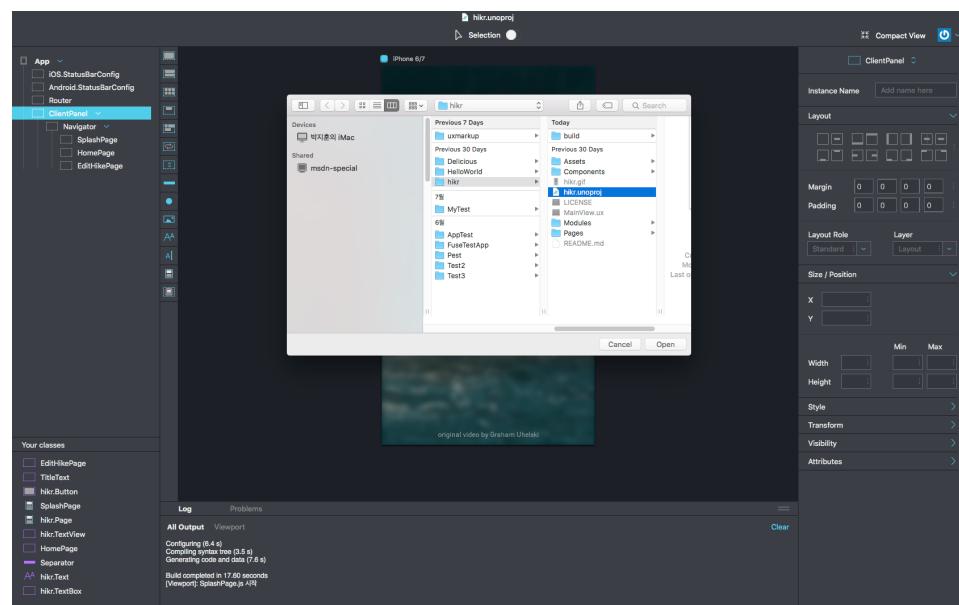
퓨즈 스튜디오를 실행하는 방법은 2가지다. 첫 번째 방법은 아래 그림과 같이 서브라임 텍스트에서 오른쪽 마우스 버튼을 눌러서 “Begin Fuse Preview > Local”을 선택한다. 그러면 퓨즈 스튜디오가 자동으로 실행된다.

서브라임 텍스트에서
퓨즈 프로젝트 실행



두 번째 방법은 아래 그림과 같이 퓨즈 스튜디오의 “File > Open” 메뉴를 실행해서 특정 디렉토리의 퓨즈 프로젝트를 직접 로딩하는 것이다. 퓨즈 프로젝트 폴더로 들어가서 그 안의 unoproj 파일을 선택하면 로딩된다.

퓨즈 프로젝트
직접 로딩하기



1. UX 계층 뷰

앱의 컴포넌트 구조를 계층적으로 보여준다. 또한 특정 컴포넌트를 선택하여 속성 값을 수정할 수 있다.

MainView.ux

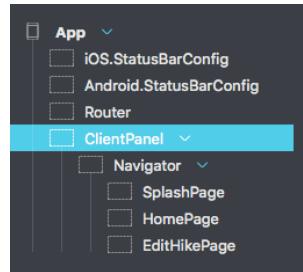
```
<App Background="#022328">
  <iOS.StatusBarConfig Style="Light" />
  <Android.StatusBarConfig Color="#022328" />

  <Router ux:Name="hikrRouter" />

  <ClientPanel>
    <Navigator DefaultPath="splash">
      <SplashPage ux:Template="splash" router="hikrRouter" />
      <HomePage ux:Template="home" router="hikrRouter" />
      <EditHikePage ux:Template="editHike" router="hikrRouter" />
    </Navigator>
  </ClientPanel>
</App>
```

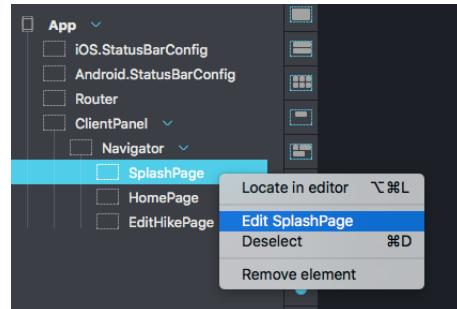
아래 그림과 같이 처음에는 MainView.ux의 구조가 보여진다.

App의 컴포넌트 계층 구조



SplashPage 컴포넌트를 선택하고 그 안의 내부 구조를 보고 싶으면 Splash Page를 마우스로 선택하고 오른쪽 마우스 버튼을 누르면 팝업 메뉴가 뜨는데 거기서 “Edit SplashPage”를 선택한다. (아래 그림 참조)

EditSplashPage 선택



아래의 SplashPage.ux에 대응하는, SplashPage에 포함된 컴포넌트들의 구조가 계층적으로 보여진다. SplashPage 안에는 Router, JavaScript, DockPanel이

있고, DockPanel에는 Video, hikr.Text, Grid가 있다. Grid에는 StackPanel, hikr.Button이 있고, StackPanel은 2개의 hikr.Button을 포함한다. Return to “App” 버튼을 누르면 이전 페이지로 돌아간다.

SplashPage.ux

```
<Page ux:Class="SplashPage">
    <Router ux:Dependency="router" />

    <JavaScript File="SplashPage.js" />

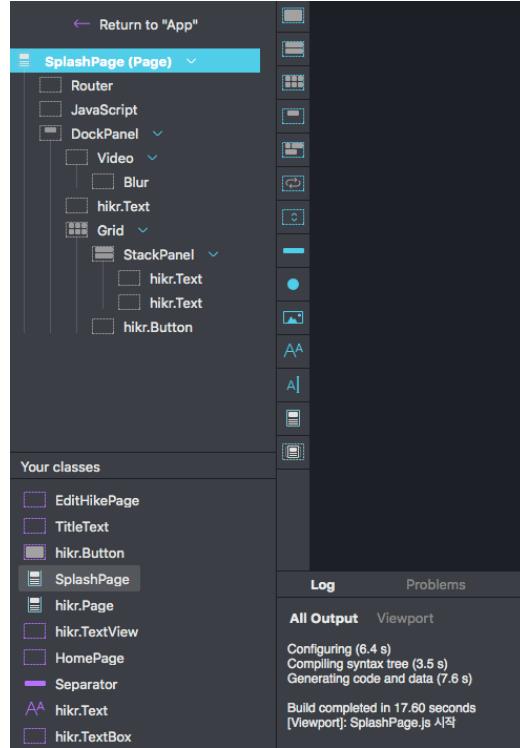
    <DockPanel ClipToBounds="true">
        <Video Layer="Background" File="../../Assets/nature.mp4"
            IsLooping="true" AutoPlay="true" StretchMode="UniformToFill"
            Opacity=".5">
            <Blur Radius="4.75" />
        </Video>

        <hikr.Text Dock="Bottom" Margin="10" Opacity=".5"
            TextAlignment="Center" FontSize="12">
            original video by Graham Uhelski</hikr.Text>

        <Grid RowCount="2">
            <StackPanel Alignment="VerticalCenter">
                <hikr.Text Alignment="HorizontalCenter" FontSize="70">
                    hikr</hikr.Text>
                <hikr.Text Alignment="HorizontalCenter" Opacity=".5">
                    get out there</hikr.Text>
            </StackPanel>

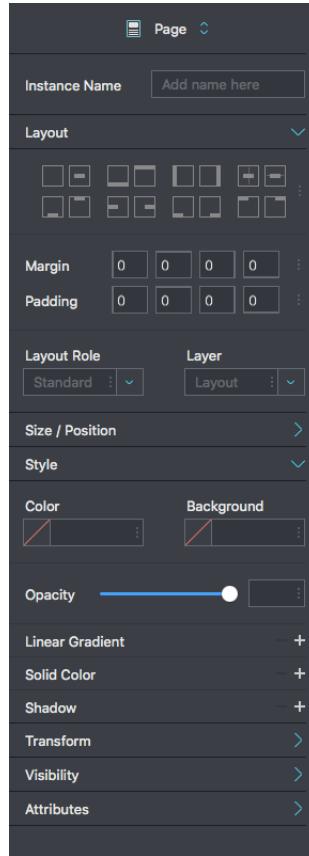
            <hikr.Button Text="Get Started" FontSize="18" Margin="50,0"
                Alignment="VerticalCenter" Clicked="{goToHomePage}" />
        </Grid>
    </DockPanel>
</Page>
```

SplashPage 페이지 선택



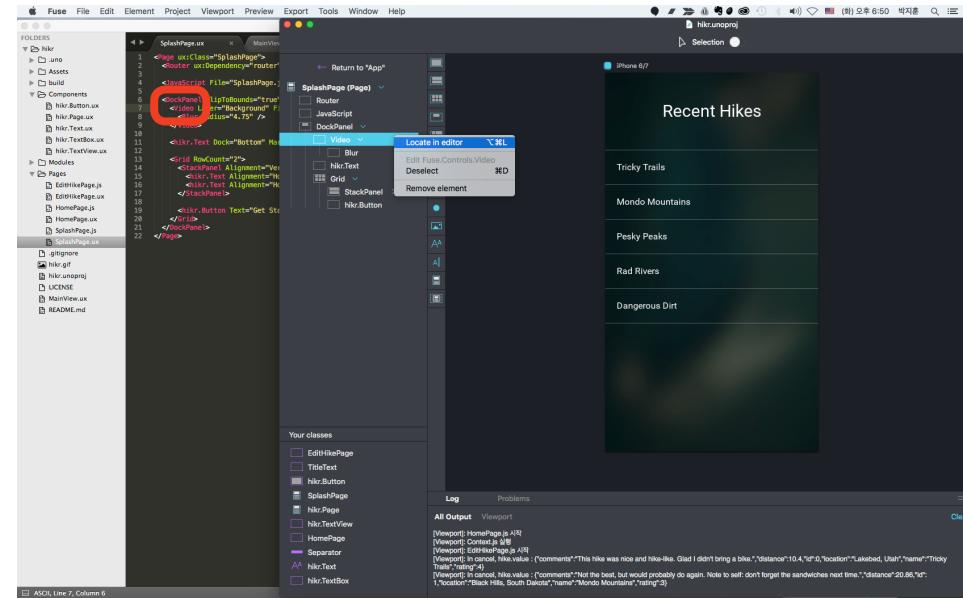
그와 동시에 퓨즈 스튜디오의 오른쪽 사이드 바의 인스펙터는 SplashPage의 속성 값을 수정할 수 있도록 변경된다. SplashPage는 Page 타입이므로 Page 타입이 설정된 것을 알 수 있다.

SplashPage 인스펙터



또는 특정 컴포넌트를 선택한 후, 소스코드의 해당 컴포넌트 위치로 바로 이동할 수 있다. 매우 편리한 기능이다. 먼저 소스코드로 이동하고자 하는 컴포넌트를 선택한 후 오른쪽 마우스 버튼을 눌러서 “Locate in editor”를 선택한다. 그러면 해당 컴포넌트의 소스코드로 바로 이동한다.

Locate In Editor 실행 결과



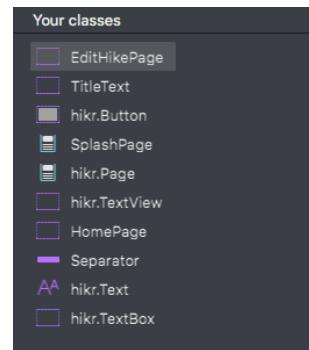
UX 계층 뷰의 SplashPage에서 Video를 선택하면 서브라임 텍스트에서 SplashPage.ux의 Video 태그로 커서가 이동하는 것을 알 수 있다.

컴포넌트를 삭제하고 싶을 때는 Remove 메뉴를 선택하면 된다.

2. 클래스 뷰

클래스로 정의한 컴포넌트의 전체 리스트를 보여준다. 컴포넌트를 마우스로 더블 클릭하면 해당 컴포넌트를 수정할 수 있도록 UX 계층 뷰와 인스펙터가 변경된다.

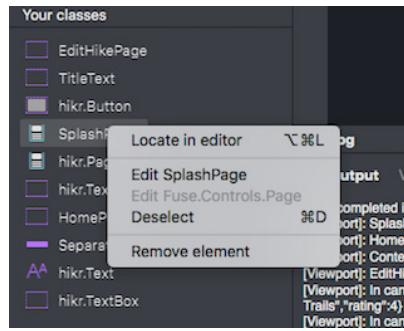
클래스 뷰



컴포넌트 클래스를 마우스로 선택한 후 오른쪽 마우스 버튼을 누르면 팝업 메뉴가 뜬다. (아래 그림 참조)

- Locate in editor: 해당 컴포넌트 클래스의 소스코드를 찾아 서브라임 텍스트에 보여준다.
- Edit SplashPage: 더블 클릭한 것과 같은 기능이다. 해당 컴포넌트 클래스를 수정할 수 있도록 UX 계층 뷰와 인스펙터를 변경한다.
- Remove element: 컴포넌트 클래스를 삭제한다.

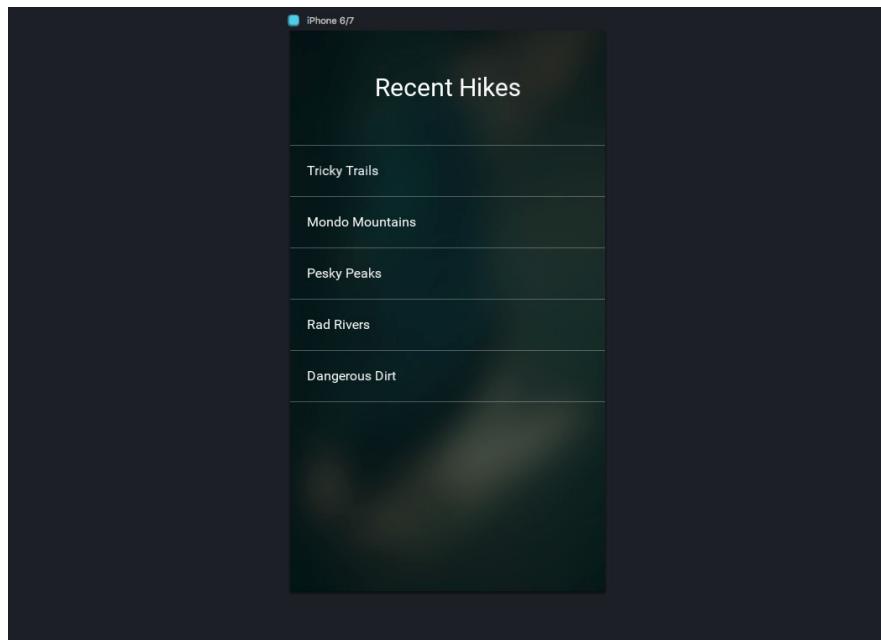
클래스의 뷰의 팝업 메뉴



3. 워크스페이스(Workspace)와 뷰포트(Viewport)

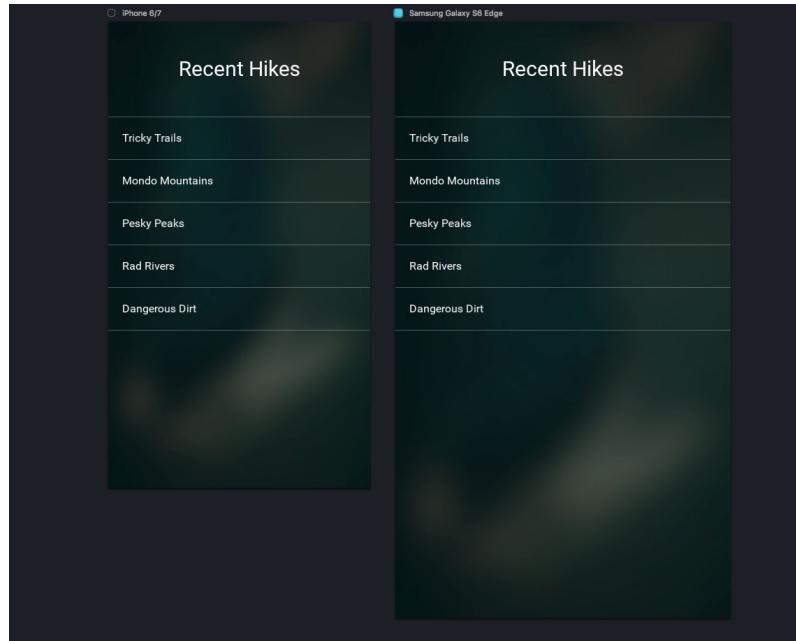
앱의 프리뷰 화면, 즉 뷰포트를 보여주고 앱을 실행한다. 선택 모드로 설정하면 특정 컴포넌트를 선택하고 속성 값을 변경할 수 있다. 여러 개 뷰포트를 동시에 보여줄 수 있다.

디폴트로 뷰포트는 iPhone 6/7 해상도로 설정된 상태이다.

1개의 뷰포트가 있는
워크스페이스

갤럭시 S6 엣지의 해상도를 가진 뷰포트를 추가하였다.

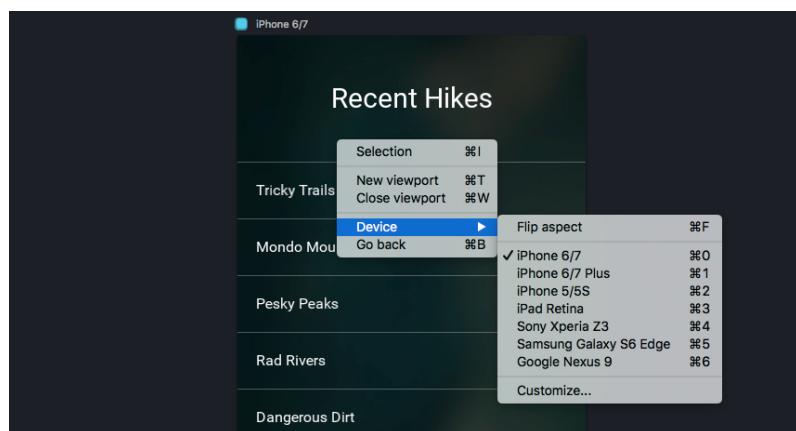
2개의 뷰포트가 있는
워크스페이스



뷰포트에 오른쪽 마우스 버튼을 누르면 팝업 메뉴가 뜬다. 팝업 메뉴의 기능은 아래와 같다.

- Selection: 실행 모드에서 선택 모드로 변경
- New Viewport: 새로운 뷰포트를 생성
- Close Viewport: 뷰포트를 닫기
- Device: 현재 뷰포트를 특정 디바이스의 해상도로 설정하기 (리스트에 없는 폰의 해상도는 Customize 메뉴를 사용해서 신규로 추가할 수 있음)
- Go back: 현재 페이지의 직전 페이지로 이동하기

워크스페이스 팝업 메뉴



워크스페이스를 사용하는 방식은 아래와 같다.

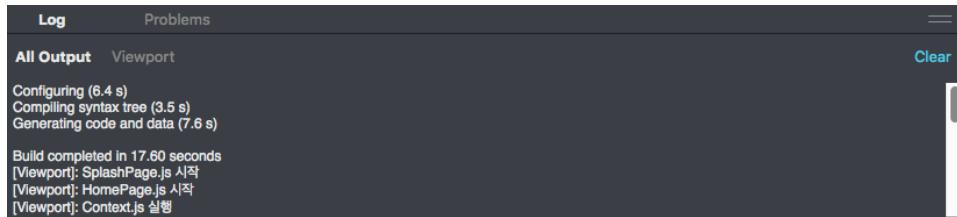
- 가. 앱의 동작 과정을 테스트하려면 “실행 모드”로 설정한 후, 마우스와 키보드로 화면에 이벤트를 발생하면서 앱을 사용한다.
- 나. 앱의 화면을 수정하려면 “선택 모드”로 설정한 후, 컴포넌트를 추가하거나 인스펙터로 속성 값을 변경한다.

4. 메시지 뷰

일반적인 로그와 문제 발생 시 에러 메시지 등을 보여준다.

“Log” 탭은 앱의 빌드 및 실행 메시지, 자바스크립트의 console.log()의 출력 메시지를 보여준다.

메시지 뷰



위의 메시지 뷰를 분석해보자.

Configuring (6.4 s)
Compiling syntax tree (3.5 s)
Generating code and data (7.6 s)

Build completed in 17.60 seconds

앱의 소스코드를 컴파일하고 빌드하는 것과 관련된 메시지이다. 빌드하는데 걸린 시간이 17.7초라는 것을 알 수 있다.

[Viewport]: SplashPage.js 시작
[Viewport]: HomePage.js 시작
[Viewport]: Context.js 실행

“[Viewport]: XXX”로 표시되는 텍스트는 앱 안에서 자바스크립트로 console.log()로 출력하는 메시지이다. 필자가 SplashPage.js, HomePage.js, Context.js 파일의 제일 첫 줄에 console.log("SplashPage.js 시작");, console.log("HomePage.js 시작");, console.log("Context.js 시작"); 등의 코드를 삽입했기 때문에 이렇게 메시지 뷰에 보이는 것이다.

ux 파일 또는 자바스크립트 파일에서 에러가 발생되면 그 메시지가 표시된다. 먼저 ux에서 문제가 발생한 경우를 살펴보자.

MainView.ux

```

<App Background="#022328">
  <iOS.StatusBarConfig Style="Light" />
  <Android.StatusBarConfig Color="#022328" />

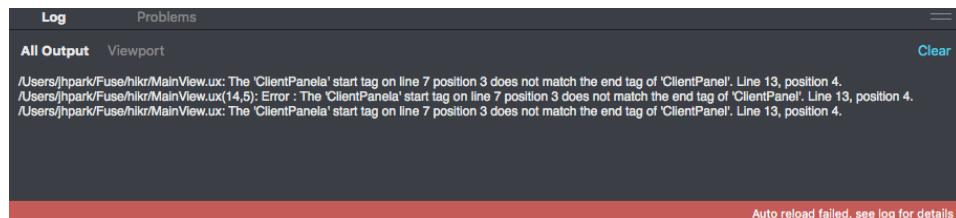
  <Router ux:Name="hikrRouter" />

  <ClientPanela> ①
    <Navigator DefaultPath="splash">
      <SplashPage ux:Template="splash" router="hikrRouter" />
      <HomePage ux:Template="home" router="hikrRouter" />
      <EditHikePage ux:Template="editHike" router="hikrRouter" />
    </Navigator>
  </ClientPanela>
</App>

```

서브라임 텍스트에서 MainView.ux 의 <ClientPanel> 태그를 <ClientPanela>로 변경(① 참조)하고 저장하면 메시지 뷰에 아래와 같은 에러 메시지가 표시된다.

ux 에러 메시지



```

/Users/jhpark/Fuse/hikr/MainView.ux: The 'ClientPanela' start tag on
line 7 position 3 does not match the end tag of 'ClientPanel'. Line 13,
position 4.
/Users/jhpark/Fuse/hikr/MainView.ux(14,5): Error : The 'ClientPanela'
start tag on line 7 position 3 does not match the end tag of
'ClientPanel'. Line 13, position 4.
/Users/jhpark/Fuse/hikr/MainView.ux: The 'ClientPanela' start tag on
line 7 position 3 does not match the end tag of 'ClientPanel'. Line 13,
position 4.

```

해석하면 MainView.ux 파일에 ClientPanela라는 시작 태그가 있는데 Client Panela라는 종료태그가 없다는 뜻이다. Line 13은 13번째 줄, position 4는 4번째 컬럼이란 뜻이다. ClientPanela를 다시 ClientPanel로 변경한 후 저장하면 에러 메시지가 사라진다.

하지만 자바스크립트에서 에러가 발생되면 바로 알기 힘들다. 자바스크립트 코드가 실행될 때 비로소 문제를 알 수 있다. 일부러 자바스크립트 코드를 틀리게 만들어보자.

SplashPage.js

```

console.log("SplashPage.js 시작");

function goToHomePage() {
  routerr.goto("home");
}

```

②

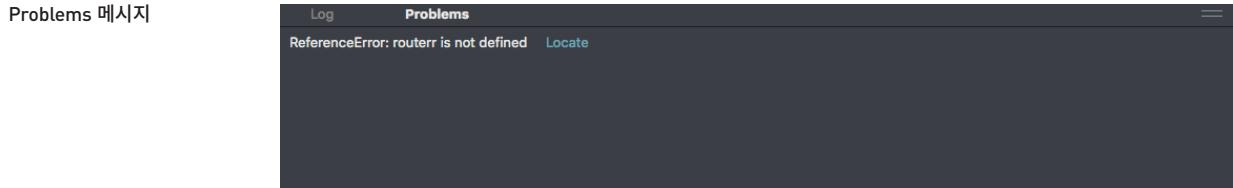
```
module.exports = {
  goToHomePage: goToHomePage
};
```

서브라임 텍스트에서 router.goto인데 routerr.goto로 변경(② 참조)하고 빌드하면 아래와 같은 에러 메시지가 출력된다.

```
Configuring (10.1 s)
Compiling syntax tree (3.7 s)
Generating code and data (33.5 s)

Build completed in 48.06 seconds
[Viewport]: SplashPage.js 시작
[Viewport]: Error: ReferenceError: routerr is not defined: Name:
ReferenceError: routerr is not defined
Error message: Uncaught ReferenceError: routerr is not defined
File name: Pages/SplashPage.js
Line number: 4
Source line:     routerr.goto("home");
JS stack trace: ReferenceError: routerr is not defined
    at goToHomePage (Pages/SplashPage.js:4:2)
    in Fuse.Reactive.FunctionMirror<Pages/SplashPage.js:4>
```

SplashPage.js의 4번째 줄에서 routerr 객체가 실제로 존재하지 않는다는 에러 메시지이다. “Problems” 탭을 선택하면 에러의 내용을 확인하고 발생한 소스 코드 위치로 이동할 수 있다.



Locate 버튼을 누르면 서브라임 텍스트에서 Splashpage.js로 이동한 후 에러 발생 위치인 4번째 줄로 커서가 이동된다.

routerr를 다시 router로 변경하고 저장하면 제대로 동작한다.

5. 툴박스

기본적인 퓨즈 컴포넌트들이 나열되어 있으며 컴포넌트를 선택하여 UX 계층 뷰에 삽입할 수 있다.

툴박스



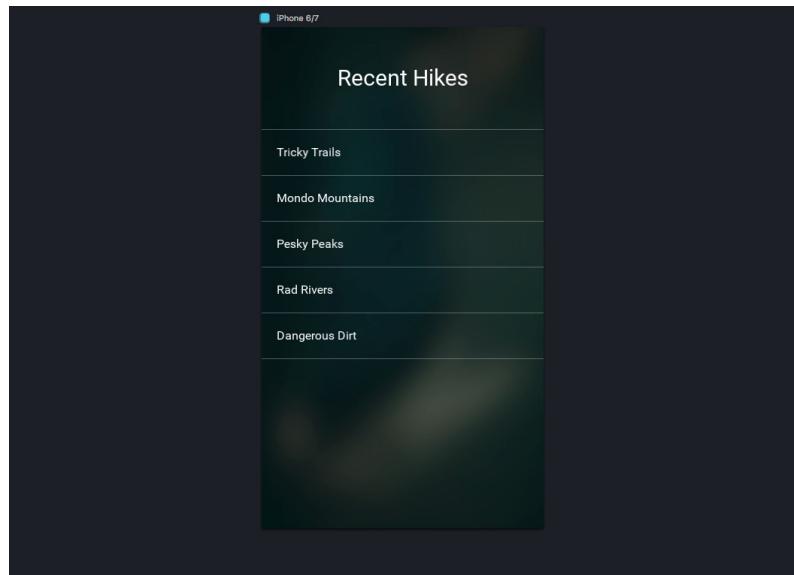
순서대로 아래와 같은 컴포넌트와 요소들이 나열되어 있다.

- 1 Panel
- 2 StackPanel
- 3 Grid
- 4 DockPanel
- 5 WrapPanel
- 6 Each
- 7 ScrollView
- 8 Rectangle
- 9 Circle
- 10 Image
- 11 Text
- 12 TextInput
- 13 Page
- 14 PageControl

컴포넌트를 마우스로 선택한 후 드래그(Drag)해서 UX 계층 뷰의 원하는 위치에 놓으면 컴포넌트가 소스코드에 자동으로 삽입된다.

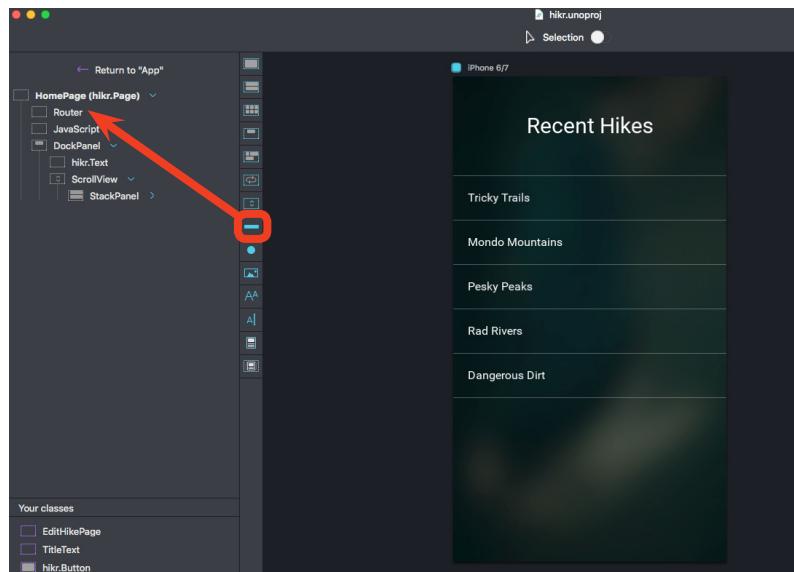
동작 과정을 살펴보기 위해 기존 HomePage.ux의 최상단에 노란색 사각형을 추가해보는 과정을 살펴보자.

HomePage.ux 화면



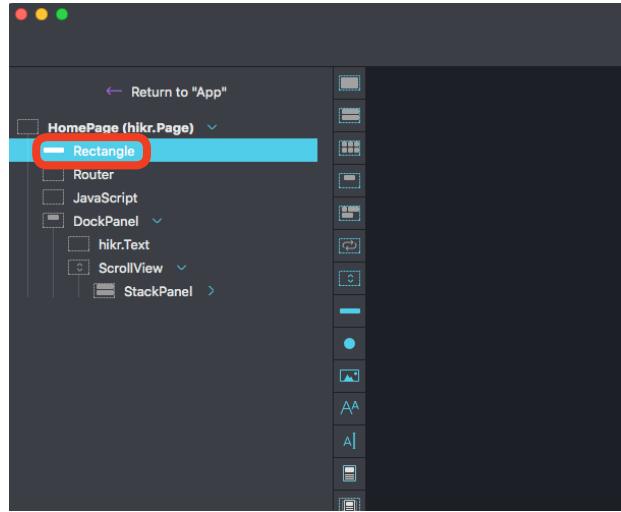
툴박스에서 Rectangle 컴포넌트를 마우스로 선택한 후, 드래그해서 UX 계층 뷰의 HomePage(hikr.Page) 바로 밑에 놓는다.

툴박스에서 Rectangle
컴포넌트 선택 및 드래그



삽입을 하면 아래 그림처럼 UX 계층 뷰에 Rectangle이 보이게 된다. 하지만 뷰 포트에는 사각형 모양이 보이지 않는다. 그 이유는 사각형의 너비와 높이, 색 등을 지정하지 않았기 때문이다.

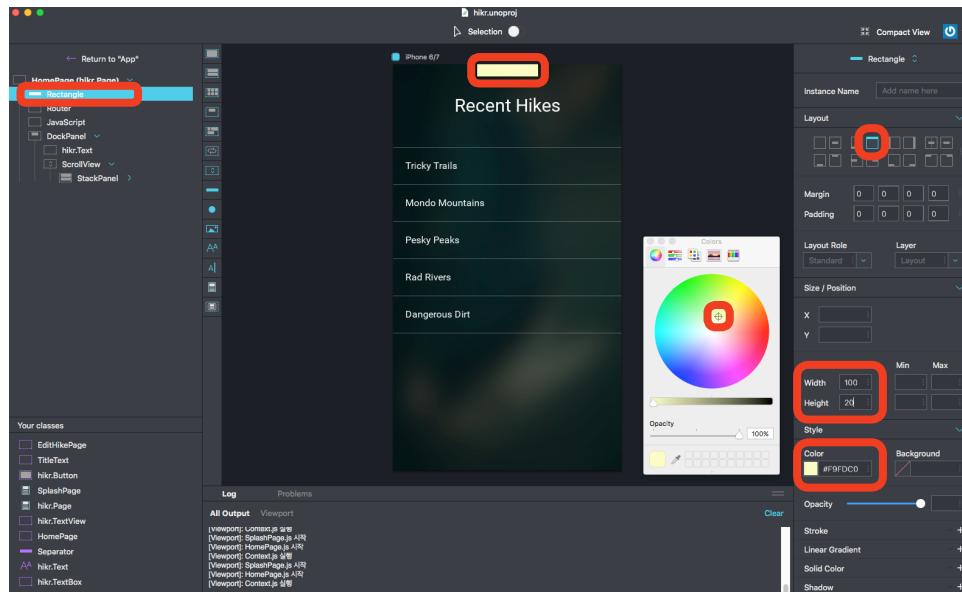
Rectangle 삽입 직후 화면



노란색 사각형을 최상단에 보이게 하기 위해 사각형의 너비와 높이, 위치, 색을 오른쪽 사이드 바의 인스펙터 뷰에서 설정한다. (아래 그림 참조)

- 최상단 위치: Layout 내의 상단 표시 사각형 아이콘 선택
- 너비와 높이: Size/Position 내의 Width, Height를 각 100, 20으로 설정
- 색: Color 메뉴의 원 팔레트를 실행, 노란색을 선택

Rectangle 삽입 후 속성 변경



이렇게 설정 값을 입력하면 위의 그림처럼 뷰포트의 상단에 노란색 사각형이 보이게 된다. 또한 HomePage.ux 파일에 `<Rectangle Width="100" Height="20" Color="#F9FDC0" Alignment="Top" />` 코드가 자동으로 삽입된다.

HomePage.ux

```
<hikr.Page ux:Class="HomePage">
    <Rectangle Width="100" Height="20" Color="#F9FDC0" Alignment="Top" /> ①
    <Router ux:Dependency="router" />
    <JavaScript File="HomePage.js" />

    <DockPanel>
        <hikr.Text FontSize="30" TextAlignment="Center" Dock="Top"
            Margin="0,50">Recent Hikes</hikr.Text>

        <ScrollView>
            <StackPanel>
                <Rectangle ux:Class="Separator" Height="1" Fill="#fff4" />

                <Each Items="{hikes}">
                    <Separator />

                    <Panel HitTestMode="LocalBoundsAndChildren"
                        Clicked="{goToHike}">
                        <hikr.Text Value="{name}" Margin="20" />

                        <WhilePressed>
                            <Scale Factor=".95" Duration=".08" Easing="QuadraticOut" />
                        </WhilePressed>
                    </Panel>
                </Each>

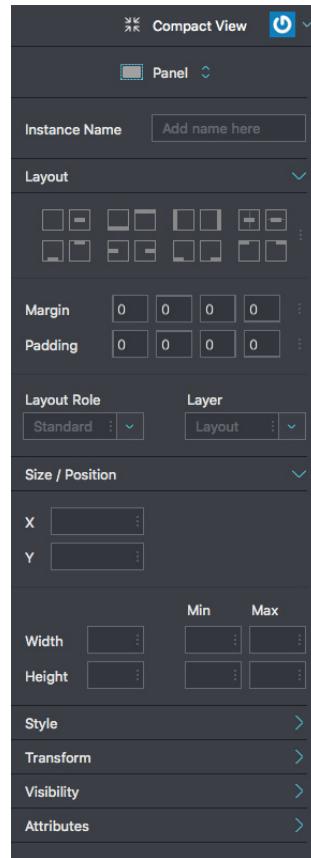
                <Separator />
            </StackPanel>
        </ScrollView>
    </DockPanel>
</hikr.Page>
```

서브라임 텍스트에서 Rectangle컴포넌트가 HomePage.ux의 첫 번째 코드에 자동으로 삽입되고 퓨즈 스튜디오에서 설정한 속성값이 입력된 것을 확인할 수 있다. (① 참조)

6. 인스펙터(Inspector)

퓨즈 스튜디오의 오른쪽 사이드 바에 있다. 컴포넌트의 메타 정보를 보여주고 속성 값을 수정할 수 있게 해준다. 속성 값이 수정되면 바로 워크스페이스의 뷰포트에 반영된다.

인스펙터 뷰



사용자가 UX 계층 뷰 또는 선택 모드 중일 때의 뷰포트에서 컴포넌트나 요소를 마우스로 선택하면 해당 이름과 속성값을 수정할 수 있도록 인스펙터의 모양이 변경된다.

예를 들어, hikr 앱의 SplashPage 화면에 있는 “Get Started” 버튼의 텍스트를 “Hello”로, 폰트 크기를 40으로 변경해보자.

SplashPage.ux

```
<Page ux:Class="SplashPage">
  <Router ux:Dependency="router" />

  <JavaScript File="SplashPage.js" />

  <DockPanel ClipToBounds="true">
    <Video Layer="Background" File="../../Assets/nature.mp4"
      IsLooping="true" AutoPlay="true" StretchMode="UniformToFill"
      Opacity="0.5">
      <Blur Radius="4.75" />
    </Video>

    <hikr.Text Dock="Bottom" Margin="10" Opacity=".5"
      TextAlignment="Center" FontSize="12">
      original video by Graham Uhelski</hikr.Text>

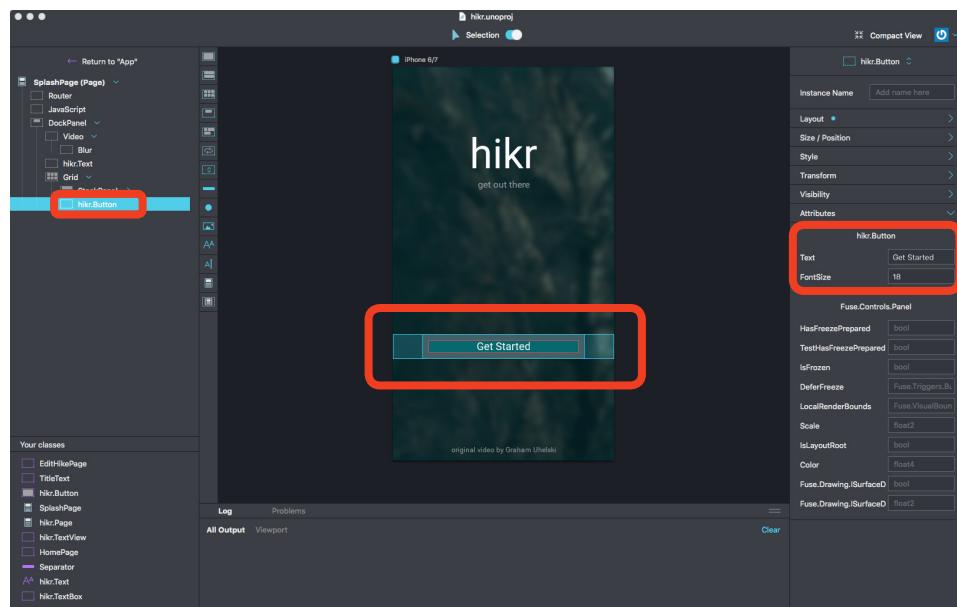
    <Grid RowCount="2">
      <StackPanel Alignment="VerticalCenter">
        <hikr.Text Alignment="HorizontalCenter" FontSize="70">
          hikr</hikr.Text>
      </StackPanel>
    </Grid>
  </DockPanel>
</Page>
```

```
<hikr.Text Alignment="HorizontalCenter" Opacity=".5">  
    get out there</hikr.Text>  
</StackPanel>  
  
<hikr.Button Text="Get Started" FontSize="18"  
    Margin="50,0" Alignment="VerticalCenter"  
    Clicked="{goToHomePage}" />  
</Grid>  
</DockPanel>  
</Page>
```

화면의 버튼은 `<hikr.Button Text="Get Started" FontSize="18" Margin="50,0" Alignment="VerticalCenter" Clicked="{goToHomePage}" />`로 생성된다.

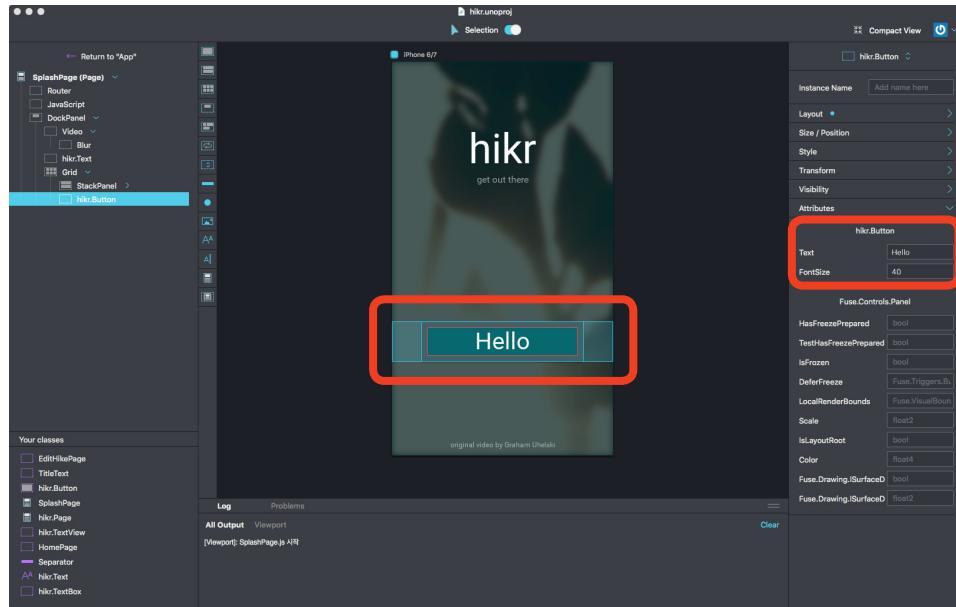
먼저 UX 계층 뷰에서 SplashPage로 이동한다. 그리고 버튼을 생성하는 hikr.Button 을 선택한다. 그러면 뷰포트에서 “Get Started” 버튼이 포커싱되고 내용, 패딩 등이 점선으로 표시된다. 그와 동시에 인스펙터는 hikr.Button 컨포넌트를 수정할 수 있도록 메뉴를 제공한다.

Get Started 버튼 선택



인스펙터의 Text, Font Size의 입력 값을 Hello, 40으로 설정하면 뷰포트의 버튼이 Hello로, 폰트크기가 40으로 바로 변경되는 것을 알 수 있다. (아래 그림 참조)

Get Started 버튼의 변경



그와 동시에 서브라임 텍스트의 `SplashPage.ux`의 `hikr.Button`의 코드도 `Text`, `FontSize`가 즉시 변경된다. (①, ② 참조)

변경된 `SplashPage.ux`

```

<Page ux:Class="SplashPage">
    <Router ux:Dependency="router" />

    <JavaScript File="SplashPage.js" />

    <DockPanel ClipToBounds="true">
        <Video Layer="Background" File="../../Assets/nature.mp4"
            IsLooping="true" AutoPlay="true" StretchMode="UniformToFill"
            Opacity="0.5">
            <Blur Radius="4.75" />
        </Video>

        <hikr.Text Dock="Bottom" Margin="10" Opacity=".5"
            TextAlignment="Center" FontSize="12">
            original video by Graham Uhelski</hikr.Text>

        <Grid RowCount="2">
            <StackPanel Alignment="VerticalCenter">
                <hikr.Text Alignment="HorizontalCenter" FontSize="70">
                    hikr</hikr.Text>
                <hikr.Text Alignment="HorizontalCenter" Opacity=".5">
                    get out there</hikr.Text>
            </StackPanel>
            <① hikr.Button FontSize="40" Margin="50,0"
                Alignment="VerticalCenter" Clicked="{goToHomePage}"
                Text="Hello" />
            <② />
        </Grid>
    </DockPanel>
</Page>
```

이렇듯 인스펙터를 사용하면 소스코드를 직접 수정하지 않고도 화면을 변경하고 바로 결과를 확인할 수 있으므로 빠르게 화면을 개발할 수 있다.

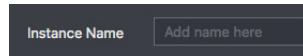
인스펙터는 선택한 컴포넌트의 종류에 따라 다르게 구성된다. 모든 화면 컴

포넌트에 대해 공통적인 부분과 컴포넌트 별로 다른 부분을 나눠서 익히는 것이 좋다.

각 화면 컴포넌트에 공통적인 인스펙터의 구성요소는 아래와 같다.

1) 공통 요소

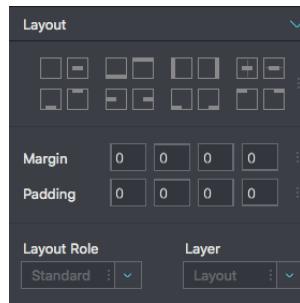
가. Instance Name



선택한 컴포넌트의 인스턴스 이름을 변경할 수 있다. 소스코드에서는 요소 내의 ux:Name 속성 값이 변경된다. 예를 들어, hello를 입력하면 <컴포넌트이름 ux:Name="hello" ... >로 변경된다.

나. Layout

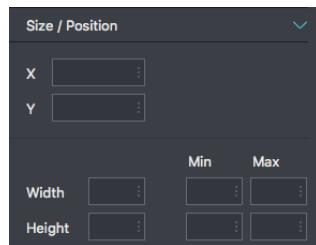
<https://www.fusetools.com/docs/fuse/elements/element> 참고



위, 아래, 좌, 우 등 컴포넌트의 위치를 결정할 수 있는 정렬 값을 설정할 수 있다. 또한 Margin, Padding, Layer 등을 설정할 수 있다.

다. Size/Position

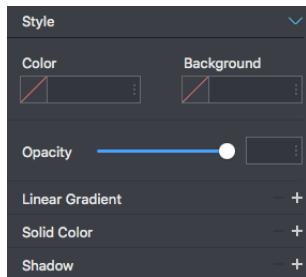
<https://www.fusetools.com/docs/fuse/elements/element> 참고



컴포넌트의 x, y 축과 너비와 높이 등을 설정할 수 있다.

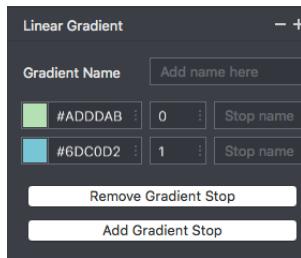
라. Style

<https://www.fusetools.com/docs/fuse/controls/shape>, <https://www.fusetools.com/docs/fuse/drawing/lineargradient>, <https://www.fusetools.com/docs/fuse/drawing/solidcolor>, <https://www.fusetools.com/docs/fuse/drawing/stroke>, <https://www.fusetools.com/docs/fuse/controls/shadow> 참고

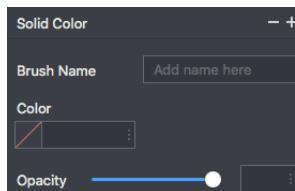


색과 배경색, 투명도 등을 설정할 수 있다. 또한 Linear Gradient, Solid Color, Shadow 등을 추가할 수 있다.

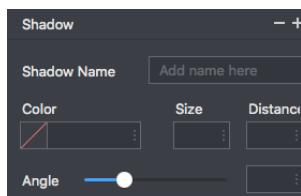
Linear Gradient의 오른쪽의 + 버튼을 누르면 그라데이션 효과를 넣을 수 있는 설정창이 보여진다. 2개의 색을 입력하여 그라데이션을 설정한다.



Solid Color의 오른쪽의 + 버튼을 누르면 솔리드 컬러를 설정할 수 있다.



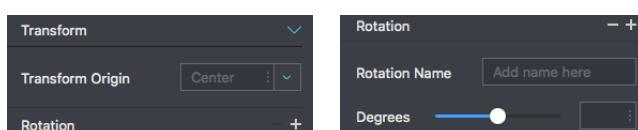
Shadow의 오른쪽의 + 버튼을 누르면 컴포넌트에 그림자 효과를 넣을 수 있다.



¶. Transform

<https://www.fusetools.com/docs/fuse/transform>, <https://www.fusetools.com/docs/fuse/rotation> 참고

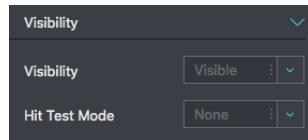
컴포넌트를 회전하는 기준 축을 설정하고, 각도를 줘서 회전할 수 있다.



비. Visibility

<https://www.fusetools.com/docs/fuse/elements/element> 참고

화면에서 보이지 않게 변경하거나 이벤트가 발생 가능한 컴포넌트의 영역을 설정할 수 있다.

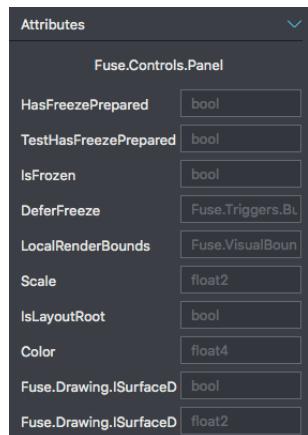


2) 컴포넌트 개별 요소

가. Panel

<https://www.fusetools.com/docs/fuse/controls/panel> 참고

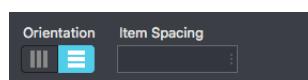
Panel은 아래의 속성 값을 설정할 수 있다.



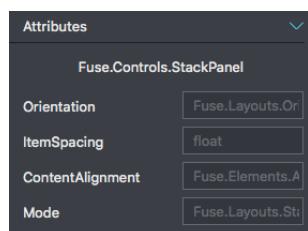
나. StackPanel

<https://www.fusetools.com/docs/fuse/controls/stackpanel> 참고

StackPanel의 컴포넌트 배치 방향(수직, 수평)과 간격을 설정할 수 있다.



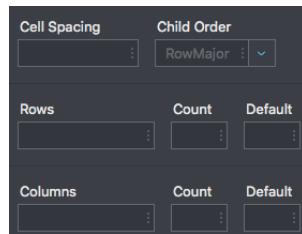
또한 아래와 같은 기타 속성값을 설정할 수 있다.



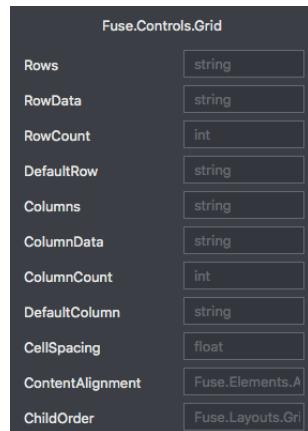
다. Grid

<https://www.fusetools.com/docs/fuse/controls/grid> 참고

행, 열의 숫자와 셀의 간격 등을 지정할 수 있다.



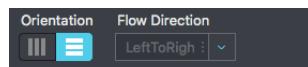
또한 아래와 같은 기타 속성값을 설정할 수 있다.



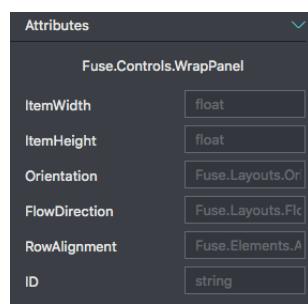
라. WrapPanel

<https://www.fusetools.com/docs/fuse/controls/wrappanel> 참고

컴포넌트 배치의 기본 방향과 순서의 방향을 지정할 수 있다.



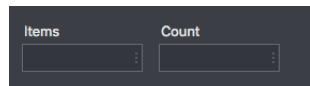
또한 아래와 같은 기타 속성값을 설정할 수 있다.



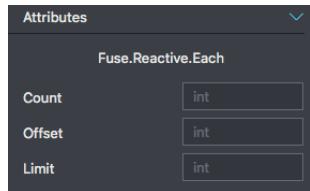
ㅌ. Each

<https://www.fusetools.com/docs/fuse/reactive/each> 참고

화면에 보여지는 컴포넌트는 아니지만 반복 설정을 할 때 배열 데이터의 이름과 숫자 등을 설정할 수 있다.



또한 아래와 같은 기타 속성값을 설정할 수 있다.



비. ScrollView

<https://www.fusetools.com/docs/fuse/controls/scrollview> 참고

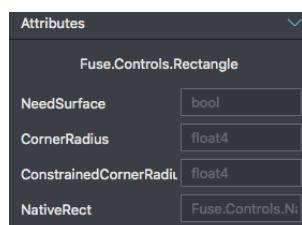
스크롤의 방향을 설정할 수 있다.



사. Rectangle

<https://www.fusetools.com/docs/fuse/controls/rectangle> 참고

아래와 같은 기타 속성값을 설정할 수 있다.



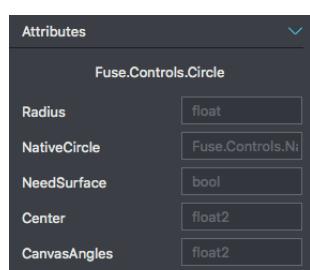
아. Circle

<https://www.fusetools.com/docs/fuse/controls/circle> 참고

원의 출발 각도, 종료 각도 등을 설정하여 슬라이스 할 수 있다.



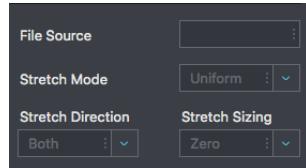
또한 아래와 같은 기타 속성값을 설정할 수 있다.



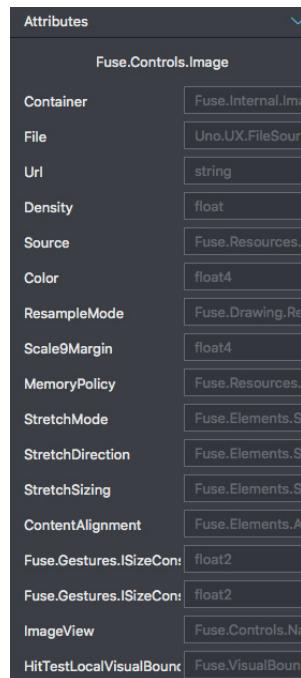
자. Image

<https://www.fusetools.com/docs/fuse/controls/image> 참고

이미지 파일의 경로, 화면에 보여지는 방식 등을 설정할 수 있다.



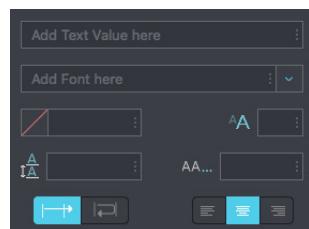
또한 아래와 같은 기타 속성값을 설정할 수 있다.



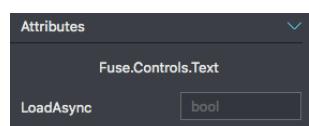
자. Text

<https://www.fusetools.com/docs/fuse/controls/text> 참고

텍스트 문자열과 폰트, 색, 정렬, 랩 여부 등을 설정할 수 있다.



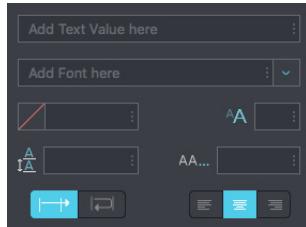
또한 아래와 같은 기타 속성값을 설정할 수 있다.



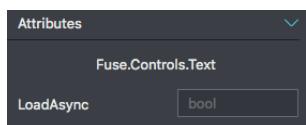
카. TextInput

<https://www.fusetools.com/docs/fuse/controls/textinput> 참고

텍스트 입력 가능한 라인의 Placeholder, 폰트, 색, 정렬, 랩 여부 등을 설정할 수 있다.



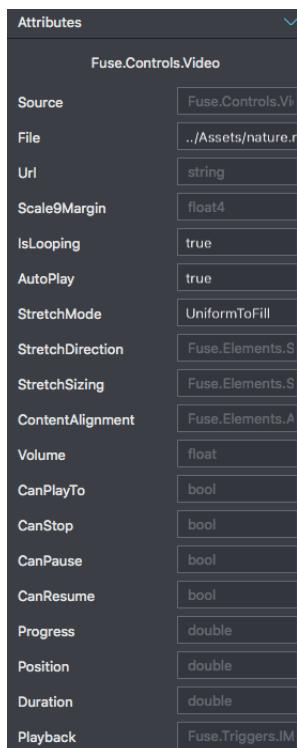
또한 아래와 같은 기타 속성값을 설정할 수 있다.



티. Video

<https://www.fusetools.com/docs/fuse/controls/video> 참고

비디오 파일의 위치, 반복 여부, 자동 플레이 등을 기타 속성 창에서 설정할 수 있다.



05.3

멀티 프리뷰 모드

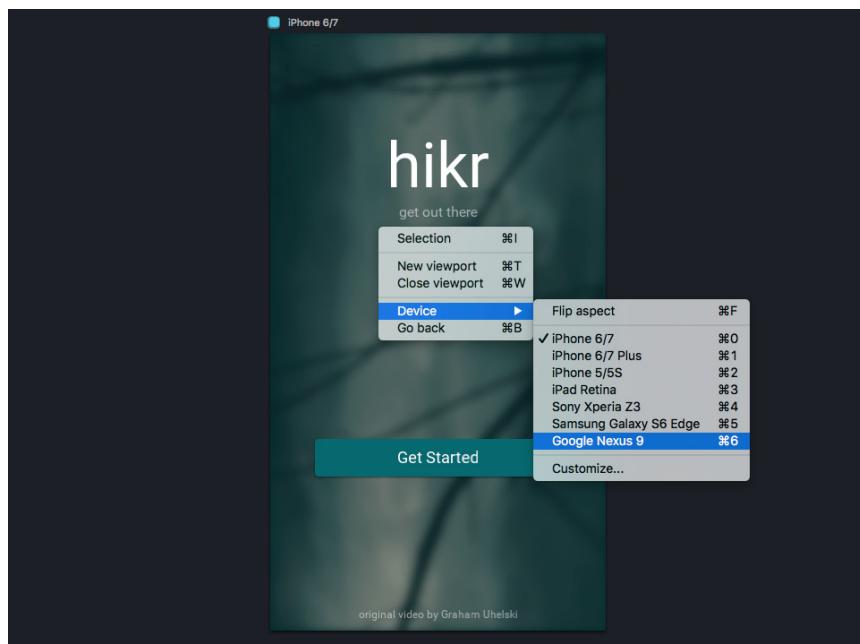
워크스페이스 내의 프리뷰 화면은 디폴트로 한 개가 실행된다. 만약 퓨즈 앱을 여러 기기의 다양한 해상도에서 실행되는 것을 목표로 개발한다면 그들의 화면에서 제대로 동작해야 하는지 확인해야 하는데 매번 뷰포트의 해상도를 변경하면서 실행 결과를 확인하는 것은 불편하다. 다행히 퓨즈 스튜디오에는 여러 개의 뷰포트를 동시에 보여주는 멀티 프리뷰 모드를 제공한다.

뷰포트 화면을 컨트롤하는 여러 방법을 살펴보자.

1. 디폴트 해상도인 iPhone 6/7에서 다른 기기의 해상도로 변경하기

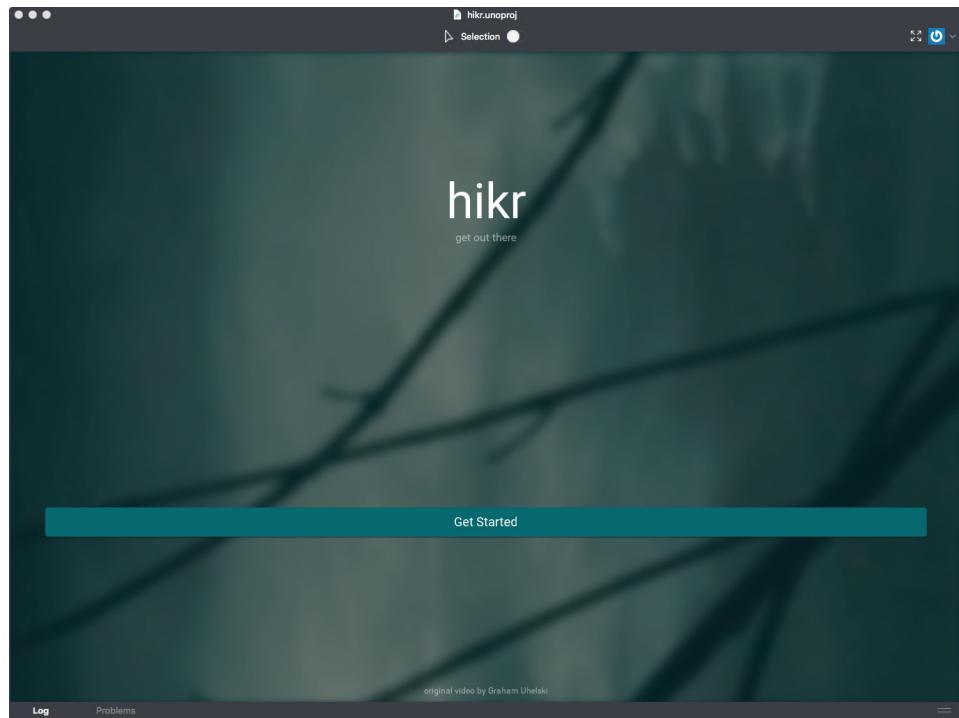
뷰포트 화면에 오른쪽 마우스 버튼을 클릭하고 Device 메뉴의 특정 기기를 선택한다.

뷰포트 변경



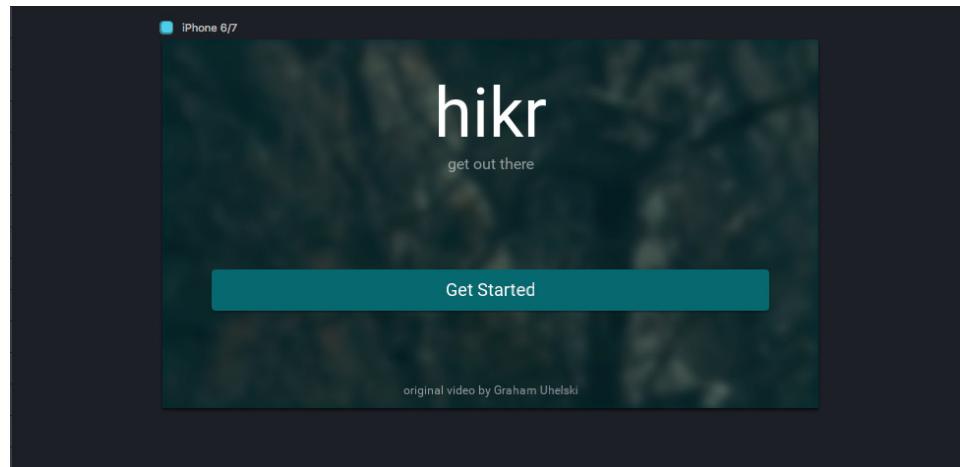
만약 구글 넥서스 9을 선택하면 기존 뷔포트는 사라지고 해당 기기의 뷔포트가 보이게 된다. (아래 그림 참조)

구글 넥서스 9의 뷔포트



오른쪽 마우스 버튼을 눌렀을 때의 팝업 메뉴 중 Devices > Flip aspect를 선택하면 디바이스를 가로로 눕힌 상태의 화면을 보여준다. 다시 반복하면 세로 상태의 기본 화면으로 돌아간다. (아래 그림 참조)

가로 화면 전환

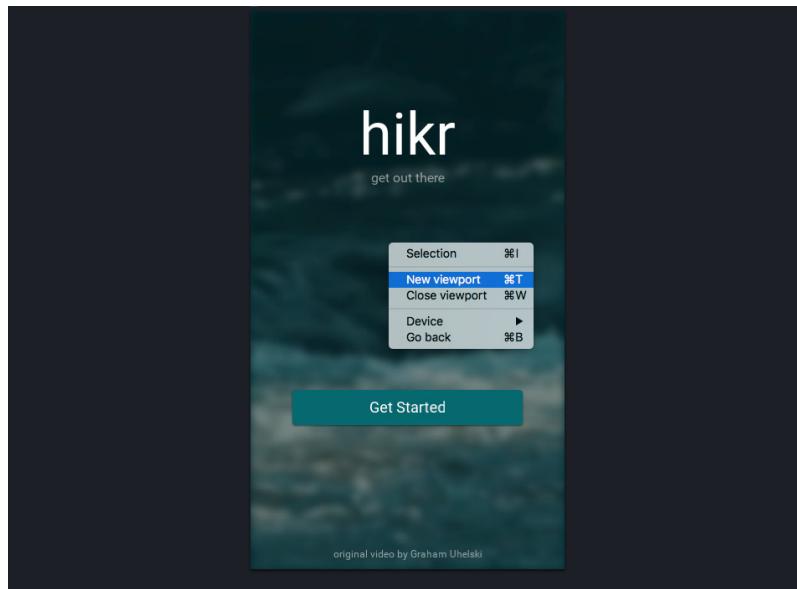


2. 새로운 뷰포트를 만들고 닫기

현재의 뷰포트 화면에서 오른쪽 마우스 버튼을 누른 후 팝업 메뉴에서 New viewport를 선택하면 동일한 모양의 뷰포트가 복사되어 생성된다. 이때 복사된 뷰포트에 오른쪽 마우스 버튼을 눌러 Devices 메뉴 중 특정 기기를 선택하면 해당 해상도로 변경된다.

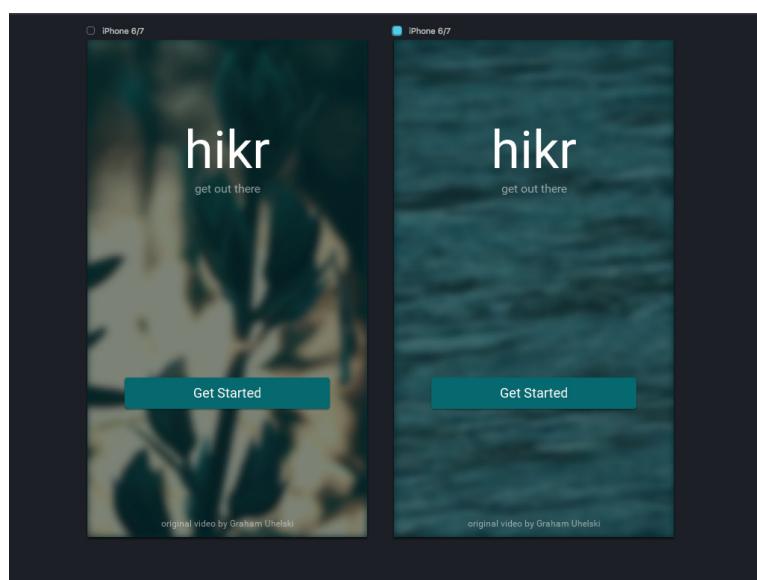
먼저 뷰포트 화면에서 New viewport 메뉴를 선택한다.

New Viewport 선택



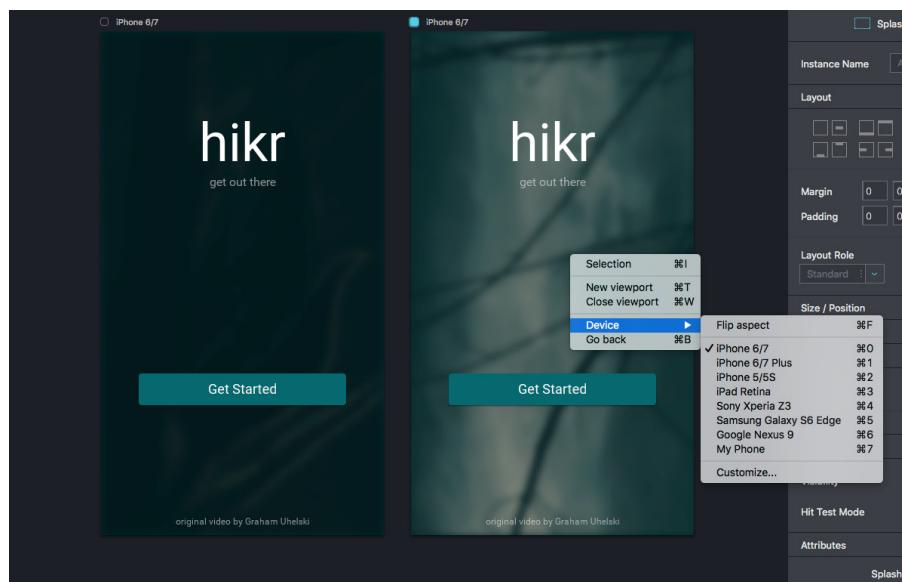
아래 그림에서 보다시피 동일한 모양의 뷰포트가 생성되었다.

New Viewport 선택



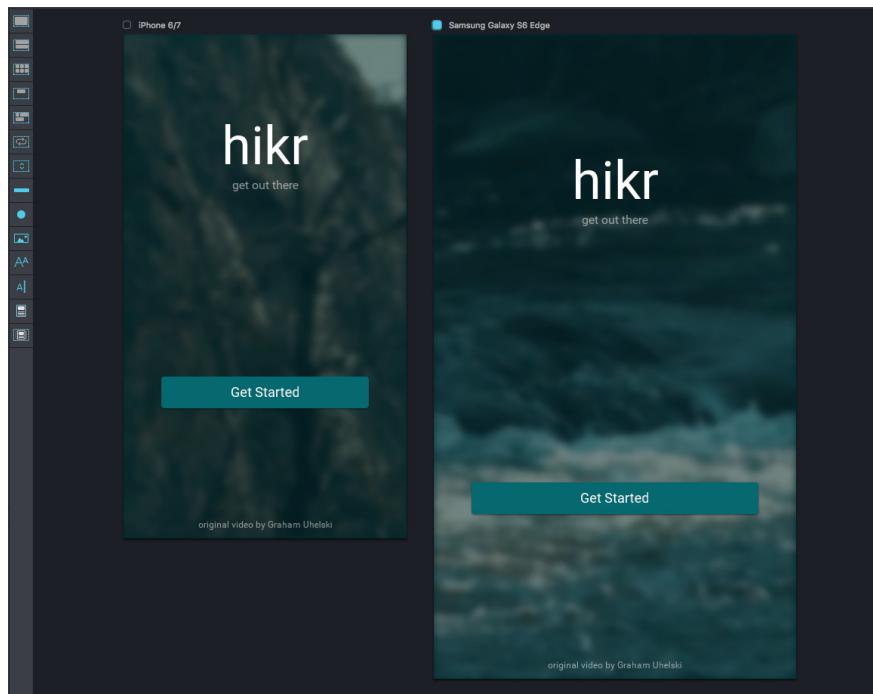
새로운 뷰포트의 해상도를 변경하기 위해 오른쪽 마우스 버튼을 누르고 Devices에서 원하는 기기를 선택한다.

새로운 기기 선택



Samsung Galaxy S6 Edge를 선택하면 아래와 같이 iPhone 6/7과 Samsung Galaxy S6 Edge 등 2개의 뷰포트를 볼 수 있다.

2개의 뷰포트



만약 특정 뷰포트가 필요 없어지면 오른쪽 마우스 버튼을 누른 후 Close view port를 선택한다.

3. 등록되지 않은 기기의 해상도 추가하기

뷰포트 화면에서 오른쪽 마우스 버튼을 누른 후 팝업 메뉴에서 Devices > Customize를 선택하면 지원하지 않는 해상도의 기기 정보를 등록할 수 있는 devices.json 파일이 생성되어 프로젝트 루트 디렉토리에 저장된다. 이 파일에서 새로운 기기 정보를 추가하면 뷰포트에서 그 기기의 해상도가 보이게 된다. devices.json은 기본적으로 아래의 기기 정보들이 등록되어 있다. IsDefault 값이 true면 해당 기기의 해상도가 뷰포트의 디폴트 값이 된다. 현재는 iPhone 6/7이 기본 해상도이다.

```
[  
  {  
    "Name" : "iPhone 6/7",  
    "Width" : 750,  
    "Height" : 1334,  
    "PixelsPerPoint": 2,  
    "PhysicalPixelsPerInch" : 326,  
    "IsDefault" : true  
  },  
  {  
    "Name" : "iPhone 6/7 Plus",  
    "Width" : 1242,  
    "Height" : 2208,  
    "PixelsPerPoint" : 3,  
    "PhysicalPixelsPerInch" : 401,  
    "PhysicalPixelsPerPixel" : 0.8695652173913  
  },  
  {  
    "Name" : "iPhone 5/5S",  
    "Width" : 640,  
    "Height" : 1136,  
    "PixelsPerPoint" : 2,  
    "PhysicalPixelsPerInch" : 326  
  },  
  {  
    "Name" : "iPad Retina",  
    "Width" : 2048,  
    "Height" : 1536,  
    "PixelsPerPoint" : 2,  
    "PhysicalPixelsPerInch" : 264,  
    "DefaultOrientation": "Landscape"  
  },  
  {  
    "Name" : "Sony Xperia Z3",  
    "Width" : 1080,  
    "Height" : 1920,  
    "PixelsPerPoint": 3,  
    "PhysicalPixelsPerInch": 423.64,  
    "DefaultOrientation": "Landscape"  
  },  
  {  
    "Name" : "Samsung Galaxy S6 Edge",  
    "Width" : 1440,  
    "Height" : 2560,  
    "PixelsPerPoint": 3,  
    "PhysicalPixelsPerInch": 577  
  },
```

```
{  
    "Name" : "Google Nexus 9",  
    "Width" : 2048,  
    "Height" : 1536,  
    "PixelsPerPoint": 1.5,  
    "PhysicalPixelsPerInch": 281,  
    "DefaultOrientation": "Landscape"  
},  
]
```

속성의 의미는 다음과 같다.

- Name: 화면에 보여지는 기기의 이름
- Width: 가로 해상도
- Height: 세로 해상도
- PixelsPerPoint: 포인트 당 픽셀수
- PhysicalPixelsPerInch: 인치당 픽셀수
- DefaultOrientation: 뷰포트의 기본 화면 방향

예를 들어 아래와 같은 디바이스의 정보를 추가한다고 가정하자.

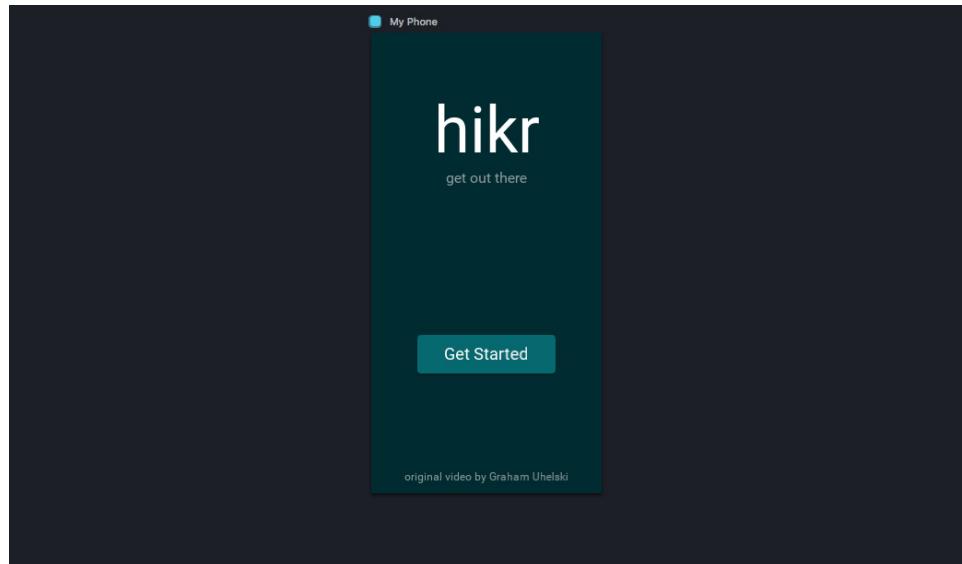
- Name: My Phone
- Width: 500
- Height: 1000
- PixelsPerPoint: 2
- PhysicalPixelsPerInch: 300

그러면 devices.json에 아래의 json 객체를 추가해야 한다.

```
{  
    "Name" : "My Phone",  
    "Width" : 500,  
    "Height" : 1000,  
    "PixelsPerPoint": 2,  
    "PhysicalPixelsPerInch": 300  
}
```

이제 퓨즈 스튜디오를 통해 “My Phone” 기기의 해상도를 지원하는 뷰포트를 만들 수 있다. Devices > My Phone이 추가된 것을 알 수 있으며 선택하면 아래 그림과 같은 뷰포트가 생성된다.

My Phone 뷔포트 화면



06

퓨즈의 고급 기능

활용하기

06.1 Uno를 활용한 네이티브 코딩하기

06.2 iOS, 안드로이드 프로젝트에 퓨즈 컴포넌트 삽입하기

아래 설명하는 퓨즈의 고급 기능은, 전문적인 개발을 위해
기존 네이티브 코드를 가져와 사용하는 방법 및 퓨즈에서 개발한 결과물을
기존 네이티브 개별 환경에서 사용하기 위한 기능이다.
본 책에서는 해당 기능을 간략하게만 설명하며,
보다 자세한 내용은 <https://www.fusetools.com/docs> 문서를 참고하도록 하자.

06.1

Uno를 활용한 네이티브 코딩하기

자바스크립트와 UX 마크업 언어만으로 모바일 플랫폼의 모든 기능을 100% 사용할 수 없다. 퓨즈에서 제공하는 자바스크립트 라이브러리와 컴포넌트로 해결할 수 없는, 모바일 플랫폼의 운영체제에서 제공하는 기능을 사용하기 위해서는 Uno라는 언어로 플랫폼의 해당 기능을 호출하는 네이티브 코드를 구현해야 한다.

Uno는 C#과 비슷한 문법의 언어이며 이 언어로 작성된 코드는 퓨즈 SDK에 의해 모바일 플랫폼에 대응되는 C++ 코드로 변환된다. 안드로이드와 iOS 등에서 동작하는 네이티브 코드인 자바나 Objective-C를 퓨즈 앱에 포함시키려면 Uno로 네이티브 코드를 패키징하는 작업 과정이 필요하다.

Example.uno

```
using Uno.Compiler.ExportTargetInterop;

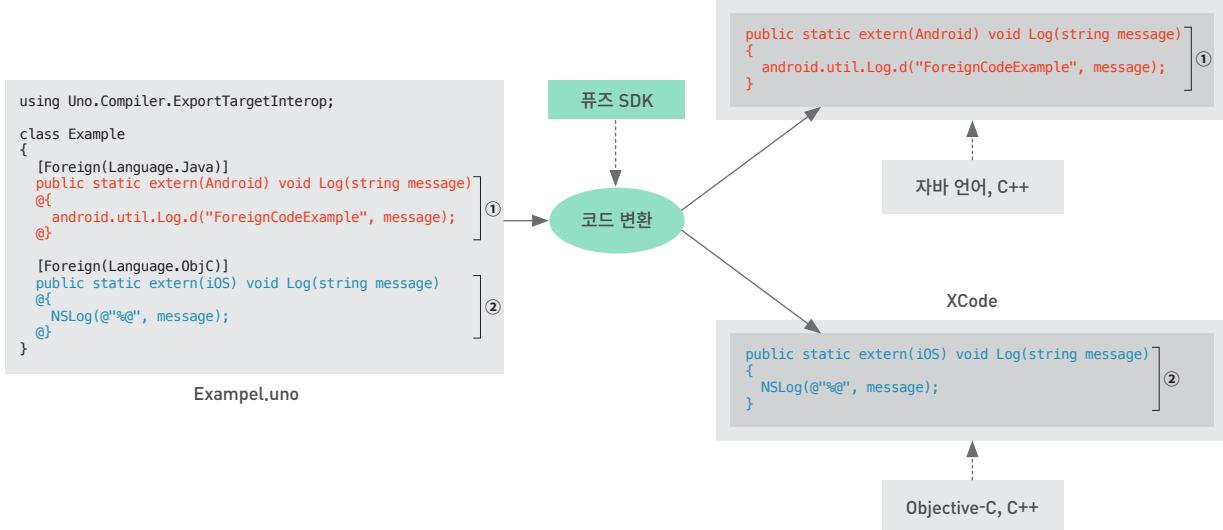
class Example
{
    [Foreign(Language.Java)]
    public static extern(Android) void Log(string message)
    @{
        android.util.Log.d("ForeignCodeExample", message);
    }

    [Foreign(Language.ObjC)]
    public static extern(iOS) void Log(string message)
    @{
        NSLog(@"%@", message);
    }
}
```

Uno 언어로 Example 클래스를 만들었는데 ①이 안드로이드 기기에서 동작하는 자바 언어이고, ②는 iOS 기기에서 동작하는 Objective-C이다. 둘 다 네이티브 코드다.

원리는 단순하다. 퓨즈 SDK가 uno파일을 네이티브 코드로 변환할 때 ([Foreign (Language.Java)])의 코드는 안드로이드용 소스코드, ([Foreign(Language.ObjC)])의 코드는 iOS용 소스코드로 변환된다. (하단 그림 참조)

Uno의 동작 과정



이렇게 코드가 변환되므로 Uno로 정의한 Log() 함수가 안드로이드와 iOS에서 네이티브 코드로 동작하는 것이다.

06.2

iOS, 안드로이드 프로젝트에 퓨즈 컴포넌트 삽입하기

퓨즈 프로페셔널 사용자는 화면 컴포넌트를 만들어서 iOS 및 안드로이드 프로젝트의 뷰(View)로 삽입할 수 있다. iOS와 안드로이드 프로젝트를 네이티브 방식, 즉 iOS SDK와 Android SDK 등을 사용하여 개발하는데 퓨즈에서 만든 특정 일부 화면을 포함하고 싶을 때 이 기술을 사용한다. 퓨즈는 화면을 빠르고 예쁘게 만들 수 있으므로 iOS 및 안드로이드 프로젝트에서 복잡한 화면을 만들어야 한다면 그 부분만 퓨즈로 만들어 포함하는 식으로 개발이 가능하다.

먼저 uno install Fuse.Views를 명령 창에서 실행한다. 그 다음 unoproj 설정에 아래의 내용을 추가한다.

```
"Packages": [  
    "Fuse",  
    "FuseJS",  
    "Fuse.Views"  
]
```

만약에 VideoView, StatsView라는 퓨즈 컴포넌트가 있는데 이것을 기존의 iOS와 안드로이드 프로젝트의 특정 화면으로 포함시키고 싶다고 가정해 보자.

```
<Panel ux:Class="VideoView">  
    <!-- ... -->  
</Panel>  
  
<Panel ux:Class="StatsView">  
    <!-- ... -->  
</Panel>
```

위의 VideoView, StatsView를 ExportedView 요소 안에 넣는다.

```
<App>  
    <ExportedViews>  
        <VideoView ux:Template="VideoView" />  
        <StatsView ux:Template="StatsView" />  
    </ExportedViews>  
</App>
```

다음에 아래와 같은 명령어를 실행하면 VideoView, StatsView가 해당 모바일

플랫폼의 라이브러리로 만들어진다.

uno build -t=ios -DLIBRARY (iOS용 라이브러리 빌드)

uno build -t=android -DLIBRARY (안드로이드 용 라이브러리 빌드)

이 라이브러리를 필요할 때 Xcode, 안드로이드 스튜디오 등에서 임포트하여 프로젝트에 포함해서 사용하면 된다.

퓨즈 컴포넌트를 모바일 플랫폼의 라이브러리로 만들어 사용하는 프로세스

