FLEX Builder 2를 이용한 FLEX 어플리케이션 제작

홍성원 | 한국어도비시스템즈 기술영업부 차장

FLEX 어플리케이션을 제작하기 위해서는 먼저 FLEX의 컴포넌트 기반 개발 방법을 이해해야 한다. FLEX는 컴포넌트 기반 개발 방법을 지원하므로 개발자는 제공되는 FLEX 컴포넌트를 사용하여 어플리케이션을 개발하거나 컴포넌트를 조합하여 새로운 컴포넌트를 제작할 수 있다.

FLEX 어플리케이션을 제작할 때 가장 먼저 하는 작업은 컨테이너라는 컴포넌트를 사용하여 사용자 인터페이스를 정의하는 것이다. 컨테이너는 다양한 사용자 인터페이스 컨트롤이나 다른 컨테이너를 포함하는 스크린상의 사각형 영역으로 Box, Grid 컨테이너 등이 있다. 컨트롤은 Button이나 TextInput 등과 같이 사용자 인터페이스를 구성하는 요소를 말한다.

예를 들어보자. **<그림 1>**은 Box 컨테이너에 3개의 Button과 1개의 ComboBox를 추가한 예다. 왼쪽에 있는 화면은 HBox(Horizontal Box)라는 컨테이너를 사용한 예로, HBox 컨테이너를 사용하면 이 컨테이너 내에 포함된 사용자 인터페이스 컨트롤을 수평으로 배치한다. 오른쪽 화면은 VBox (Vertical Box) 컨테이너를 사용한 예로, 컨테이너 내의 컨트롤을 수직으로 배치한다.

<그림 1> Box 컨테이너에 3개의 Button과 1개의 ComboBox를 추가한 예

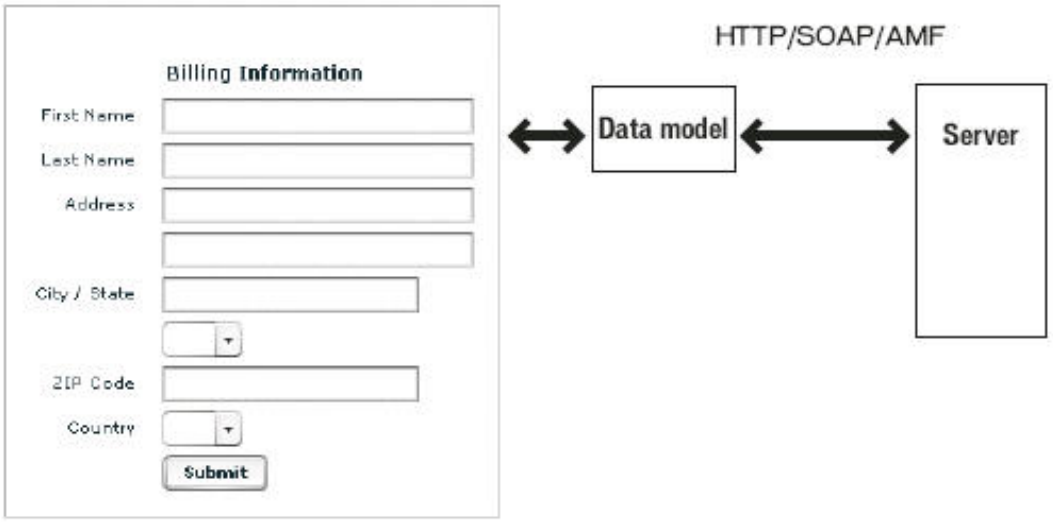
FLEX 어플리케이션에서는 다양한 컨테이너와 컨트롤을 사용하여 사용자 인터페이스를 구성하게 되고 이는 MVC(Model-View-Controller) 디자인 패턴에 의하면 View에 해당하는 요소이다. FLEX 어플리케이션에서 Model은 MXML이나 ActionScript 언어를 사용하여 정의할 수 있다. MXML은 사용자 정의 메쏘드가 필요하지 않은 단순한 데이터 모델을 정의할 때 사용되고 ActionScript는 사용자 정의 메쏘드를 가지거나 복잡한 데이터 모델을 정의할 때 사용된다.

<그림 2>는 데이터 모델을 사용하는 FLEX 어플리케이션 구조를 보여준다. FLEX에서는 이와 같이 데이터 모델을 정의한 후 데이터 바인딩이라는 개념을 사용해서 데이터를 사용자 인터페이스 컨트롤에 디스플레이 하게 된다.

데이터 바인딩을 사용하면 사용자 인터페이스에서 사용자가 입력하거나 선택한 데이터를 데이터 모델로 보내거나 데이터 모델의 데이터를 사용자 인터페이스에 디스플레이 할 수 있다. 또한 서버 측 데이터를 직접 FLEX 사용자 인터페이스 컨트롤에 바인딩 할 수도 있다. 웹 서비스의 결과 값을 정의된 데이터 모델로 바인딩하고, 이 값을 TextInput이나 Label 등의 사용자 인터페이스에 디스플레이 하는 것이 그 예이다.

또한 데이터 모델에서 FLEX가 제공하는 Validator 클래스를 사용하면 데이터에 대한 유효성 검사를 자동으로 수행할 수 있다. 예를 들어 우편번호를 입력 받는 화면에서 ZipCode validator를 사용하면 우편번호 형식의 데이터가 입력되었는지를 클라이언트 단에서 검사하고 적절한 오류 메시지를 디스플레이 할 수 있다.

이와 같이 FLEX 어플리케이션은 어플리케이션의 사용자 인터페이스, 데이터와 비즈니스 로직을 분리시킬 수 있는 MVC 디자인 패턴으로 구성할 수 있다.



<그림 2> 데이터 모델을 사용하는 FLEX 어플리케이션 구조

FLEX 어플리케이션의 작동 구조

그러면 간단한 FLEX 어플리케이션을 작성해보기 전에 FLEX 어플리케이션의 작동 구조를 알아보자. FLEX 어플리케이션은 FLEX 소스 코드를 컴파일 하면 생성되는 바이너리 형태의 swf 파일을 웹 서버에 올려놓고 사용자가 지정된 url을 호출하면 웹 서버가 FLEX swf을 서비스하는 구조로 동작하게 된다.

FLEX 어플리케이션을 서비스할 수 있는 웹 서버는 다음 세 가지 유형이 있다.

✧ **단순 웹 서버**

정적인 페이지 요청에 대해 HTML 페이지를 서비스하는 단순한 웹 서버를 말한다. IIS나 아파치 등의 그 예이며, 이 경우 개발자는 FLEX 어플리케이션을 미리 컴파일 한 후 FLEX 어플리케이션을 임베디드 하는 wrapper html 페이지를 작성해서 웹 서버에 올려 놓고 사용자는 wrapper html 페이지를 실행해서 FLEX 어플리케이션을 실행하게 된다.

✧ **웹 어플리케이션 서버**

JRun, Coldfusion 등과 같이 동적으로 FLEX 어플리케이션을 구동하는 페이지를 생성할 수 있는 서버를 말한다. 이 경우에도 역시 FLEX Data Service 2 기능을 사용하지 않는다면 FLEX 어플리케이션을 미리 컴파일 해서 생성된 swf 파일을 웹 어플리케이션 서버에 올려놓고 웹 어플리케이션 서버가 동적으로 FLEX 어플리케이션을 구동하는 페이지를 생성하도록 할 수 있다.

✧ **J2EE 어플리케이션 서버나 서블릿 컨테이너**

:FLEX 어플리케이션에서 FLEX Data Service 2 기능을 사용할 경우에는 JRun, Tomcat, 웹스피어, 웹로직 등의 J2EE 어플리케이션 서버나 서블릿 컨테이너가 필요하다. 참고로 FLEX Date Service 2를 설치할 때 JRun 4 개발자 버전이 같이 제공되는데 이 개발자 버전은 실제 운영에는 사용할 수 없고 개발시 테스트 용도로만 사용 가능하다.

그러면 FLEX 어플리케이션을 개발하는 단계를 살펴보도록 하자.

1. FLEX Builder 2 등의 통합 개발 환경을 사용한 MXML 파일(FLEX 소스코드) 작성
2. 필요한 컨테이너 추가
3. 컨테이너 내에 Button, TextInput 등의 필요한 사용자 인터페이스 컨트롤 추가
4. 데이터 모델 정의
5. 데이터 호출을 위한 웹 서비스나 HTTP 서비스, 원격 자바 객체 지정
6. 입력된 데이터에 대한 유효성 검사 루틴 추가
7. 컴포넌트를 확장하는 기능이 필요한 경우, ActionScript 코드 추가
8. 완성된 어플리케이션의 swf 파일 컴파일

FLEX 어플리케이션 개발시에는 MXML 언어와 ActionScript라는 두 가지 언어를 사용하게 된다. MXML은 XML 기반 언어로 FLEX에서 제공하는 다양한 사용자 인터페이스 컨트롤들을 사용하여 사용자 인터페이스를 정의하는데 사용된다.

ActionScript는 ECMA-262 Edition 4 기반의 스크립트 언어로 주로 클라이언트 단에서 처리되는 로직이나 함수를 정의하기 위해 사용된다.

FLEX에서 제공되는 컴포넌트는 ActionScript 클래스 라이브러리로 구성되어 있기 때문에 MXML 태그는 ActionScript 클래스나 ActionScript 클래스의 속성에 해당되게 된다. MXML로 작성한 어플리케이션을 컴파일 하면 사용한 MXML 태그에 해당하는 ActionScipt 객체를 포함하는 swf 파일이 생성되는 것이다.

예를 들어 MXML 태그를 아래와 같이 작성하면 label 속성값에 “Submit”라는 값을 갖는 Button 객체가 하나 생성되게 된다.

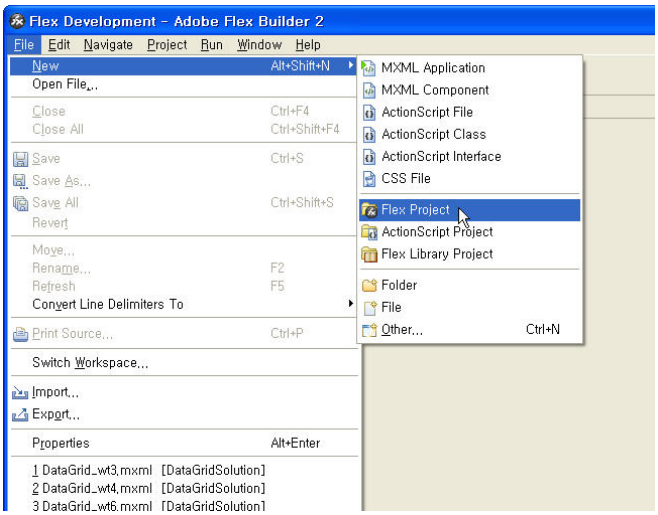
```
<mx:Button label=”Submit”/>
```

FLEX 어플리케이션 개발 예제

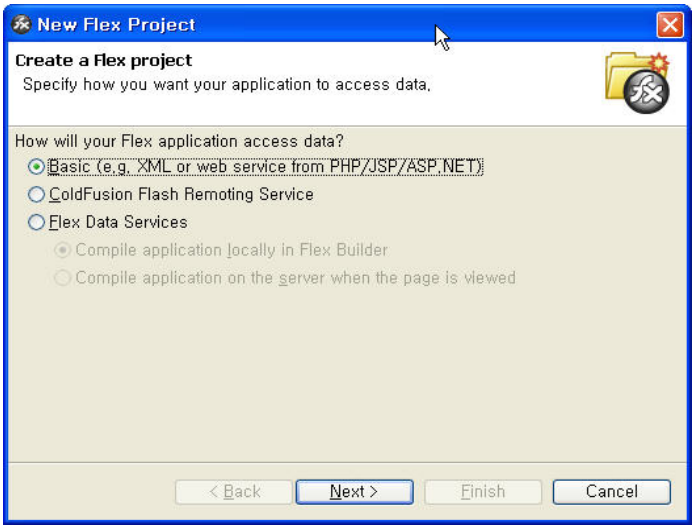
그러면 FLEX Builder 2를 사용해서 TextInput, TextArea, Button 컨트롤로 구성된 간단한 FLEX 어플리케이션을 하나 만들어 보도록 하자.

먼저 **<그림 3>**과 같이 File/New/FLEX Project를 메뉴를 선택하고 새로운 FLEX 프로젝트를 생성한다. FLEX Project는 하나의 FLEX 어플리케이션을 구성하기 위해 필요한 모든 파일의 집합으로, 프로젝트 유형은 **<그림 4>**와 같이 3가지 유형이 있다.

이번 실습에서는 FLEX Data service 2나 Coldfusion, Flash Remoting을 사용하지 않을 것이므로 첫 번째 “Basic” 옵션을 선택하고 프로젝트 이름을 “FirstDemo”라고 입력한 뒤 “Next” 버튼을 누른다.

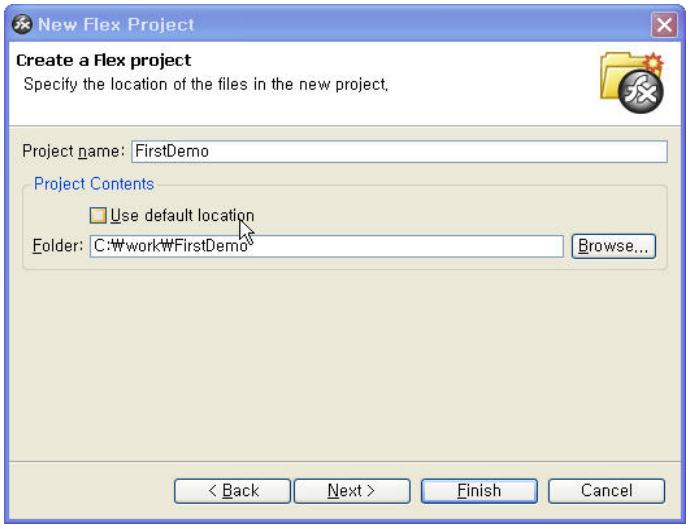


<그림 3> 새로운 FLEX 프로젝트 생성



<그림 4> 프로젝트 유형 선택

다음 화면에서는 Project name 항목에 "FristDemo"를 입력하고 folder는 "Browse" 버튼을 눌러 Project 파일을 저장할 디렉토리를 지정한다. **<그림 5>**



<그림 5> 프로젝트 이름 및 디렉토리 지정

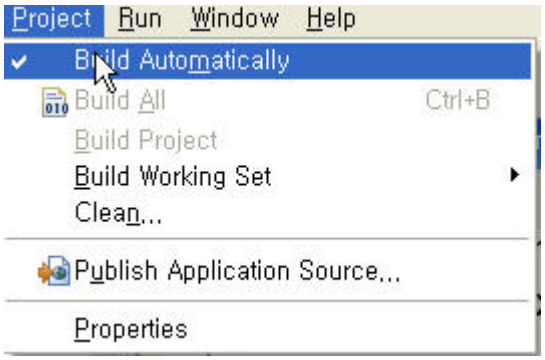
이렇게 Project가 생성되면 다음과 같은 서브 디렉토리와 파일이 생성되고 각각의 파일은 다음과 같은 용도로 활용된다.

✧ **bin :** 컴파일 된 FLEX 어플리케이션 파일(swf 파일)이 저장되는 디렉토리

- ✧ **html-template :** 생성된 swf를 임베드 할 수 있는 html wrapper 페이지가 저장되는 디렉토리
- ✧ **.settings :** 해당 프로젝트에 관련된 FLEX Builder에 대한 여러 가지 설정 파일이 저장되는 디렉토리
- ✧ **.actionSciprtProperties, .FLEXProperties, .project 파일 :** 프로젝트 설정 파일들

파일명을 따로 지정하지 않으면 Main application file은 프로젝트 이름과 동일하게 생성되므로 First Demo.mxml이라는 파일이 생성된다. FirstDemo.mxml 파일을 선택하고 FLEX Builder 2 상단의 Run 버튼을 누르면 FLEX 어플리케이션이 컴파일 된 후 실행된다.
<그림 8>과 같이 Project 메뉴에서 Build Automatically 옵션을 선택해놓으면 main application file이 수정되고 저장될 때마다 자동으로 FLEX Builder 2가 mxml 파일을 컴파일해서 업데이트된 swf 파일을 생성해준다.

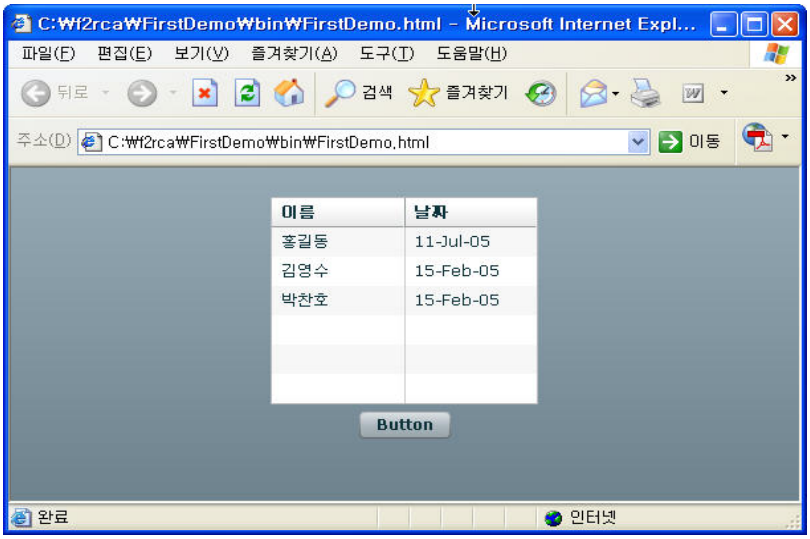
<그림 7> FLEX 어플리케이션 실행하기



<그림 8> Build Automatically 옵션이 선택된 상태

FLEX 어플리케이션 개발 과정 점검

간단한 FLEX 어플리케이션을 3페이지에서 설명한 FLEX 어플리케이션 개발 단계에 따라 작성해 보도록 하자. **<그림 9>**



<그림 9> 간단한 FLEX 어플리케이션

1. MXML 어플리케이션 파일은 프로젝트 생성 단계에서 이미 만들었고 파일 이름은 FirstDemo.mxml이다.

2. 필요한 컨테이너를 추가한다. **<그림 9>**에서 보면 사용자 인터페이스에서 사용된 컨트롤들이 세로로 배치되어 있으므로 VBox라는 컨테이너를 사용하면 된다. main application에서 사용되는 Application 컨테이너의 layout 속성을 아래와 같이 “vertical”로 지정하면 VBox 컨테이너와 동일하게 동작하므로 추가로 VBox 컨테이너를 삽입할 필요가 없다.

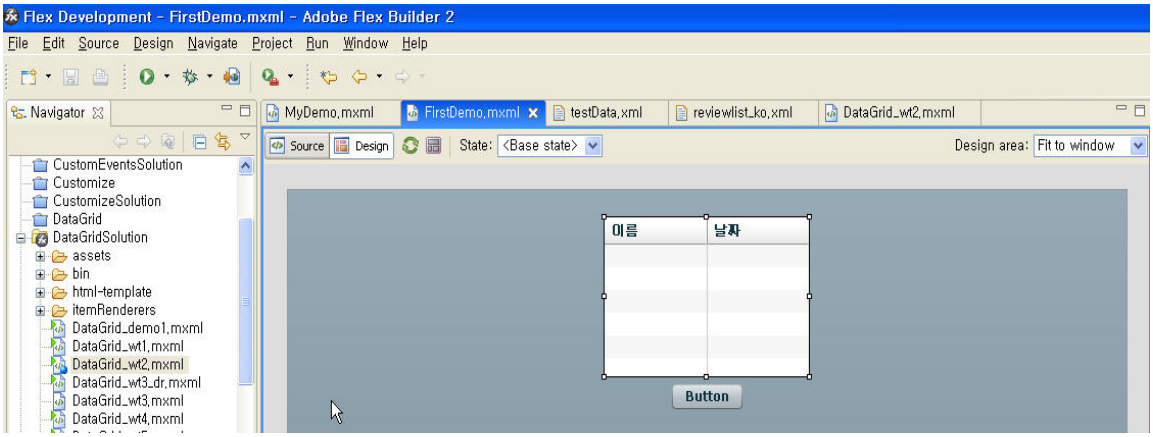
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical">
</mx:Application>
```

3. 다음으로 필요한 사용자 인터페이스 컨트롤을 구성하기 위해 DataGrid와 Button을 추가한다. FLEX에서 제공하는 컨트롤은 Editor에서 Source 보기 모드로 MXML 태그를 직접 입

력할 수도 있고, Design 모드로 전환한 후 왼쪽 하단 Component 패널에서 드래그해서 삽입할 수도 있다.

두 가지 방법 중 더 편리하다고 생각되는 옵션으로 **<그림 10>**과 같이 DataGrid와 Button을 삽입한다. 차후에 재사용하기 위해 DataGrid의 속성은 미리 지정해준다. dataProvider라는 속성은 DataGrid에 표시될 데이터를 지정하는 속성으로 reivewList라는 변수에 데이터 모델을 정의할 예정이므로 미리 아래와 같이 데이터 바인딩을 지정해 놓는다.

```
<mx:DataGrid dataProvider ="{reviewList}">
                    <mx:columns>
                    <mx:DataGridColumn   dataField="이름"   id="reviewerCol"
                              headerText="Reviewer"/>
                    <mx:DataGridColumn dataField="날짜"
                    headerText="Review Date" labelFunction="myDateFormat"/>
                    </mx:columns>
          </mx:DataGrid>
```



<그림 10> DataGrid와 Button을 삽입한 화면

4. 다음으로 어플리케이션에서 사용할 데이터 모델을 정의하기 위해 간단한 xml 파일을 하나 만들고 FristDemo 프로젝트 폴더에 testData.xml이라는 이름으로 저장한다.

```
<?xml version="1.0" encoding="utf-8"?>
    <list>
            <review>
                    <reviewer>홍길동 </reviewer>
                    <review_date>2005-07-11 10:00:38.0</review_date>
            </review>
            <review>
                    <reviewer>김영수</reviewer>
                    <review_date>2005-02-15 11:00:38.0</review_date>
            </review>
            <review>
                    <reviewer>박찬호</reviewer>
                    <review_date>2005-02-15 11:00:38.0</review_date>
            </review>
    </list>
```

5. 다음 단계는 데이터 모델을 액세스하기 위한 객체를 정의하는 단계로, 여기에서는 HTTPSerivce 객체를 이용할 예정이므로 다음과 같은 코드를 mx:Application 태그 바로 아래 3번째 줄에 삽입한다.

```
<mx:HTTPService id="simpletest"  url="testData.xml"
 result="reviewList=simpletest.lastResult.list.review"/>
```

여기서 HTTPService 태그의 result 속성은 http 서비스를 통해 데이터를 제대로 불러온 후 수행할 명령을 입력하는 부분인데, HTTPSerivice에서 가져온 결과값은 HTTPService의 id.lastReult라는 속성으로 액세스 가능하므로 reviewList라는 변수에 가져온 결과값 중에서 반복되는 reviewer와 review_date 데이터를 지정하기 위해 위와 같이 코드를 작성한다.

6. 이 어플리케이션에는 데이터 유효성 검사 루틴이 별도로 필요하지 않으므로 날짜 데이터를 “DD-MMM-YY" 형식으로 포맷하여 보여주는 함수를 추가해보자.
먼저 FLEX에서 제공하는 DateFormatter 클래스를 선언하고 formatString을 “DD-MMM-YY" 라는 형식으로 지정한다.

```
<mx:DateFormatter id="myDate" formatString="DD-MMM-YY"/>
```

그리고 위에서 선언된 DateFormatter를 이용하여 DataGrid의 날짜 데이터를 포맷할 때 사용할 함수를 <mx:Script> 태그 내에 삽입한다.

```
private function myDateFormat(item:Object,column:DataGridColumn):String
        {
return myDate.format(item[column.dataField]);
        }
```

7. HttpService를 수행한 결과를 저장하기 위한 reviewList라는 이름을 가지는 변수를 선언하고 변수 유형은 ArrayCollection으로 지정한다. 참고로 ArrayCollection은 FLEX 어플리케이션에서 반복적인 데이터 구조를 저장하기 위해 가장 많이 사용되는 데이터 유형이다.

```
<mx:Script>
        <![CDATA[
                        import mx.collections.ArrayCollection;
                        [Bindable]
                        private var reviewList:ArrayCollection;
        ]]>
</mx:Script>
```

마지막으로 어플리케이션이 실행될 때 HTTPService를 호출하기 위해 다음과 같이 어플리케이션 태그를 수정한다.

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="simpletest.send();" fontSize="12" >
```

8. Run 버튼을 눌러 FLEX 어플리케이션을 수행해본다.

이상으로 기본적인 FLEX Builder 2를 사용하여 FLEX 어플리케이션을 작성해보았다.
기본 개념만 이해하면 FLEX Component explorer 어플리케이션*이나 FLEX 2 Language reference 문서**에서 제공되는 다양한 FLEX 컨트롤의 API를 참조하면 다양한 형태의 FLEX 컨트롤을 사용

한 어플리케이션 개발이 충분히 가능하다.
다음 글에서는 FLEX에서 제공되는 Chart 컨트롤의 사용법을 알아보고 이를 활용하여 데이터를 시각적으로 보여주는 Chart 어플리케이션을 작성해보도록 하겠다.

* FLEX Component explorer 어플리케이션
http://examples.adobe.com/FLEX2/inproduct/sdk/explorer/explorer.html
** FLEX 2 Language reference 문서
http://livedocs.macromedia.com/FLEX/2/langref/index.html

<Tip> FLEX Builder 2에 대한 추가 정보는 Using FLEX Builder 2 문서 (http://download.macromedia.com/pub/documentation/en/FLEX/2/FLEX2_usingFLEXBuilder.pdf)나 FLEX Builder 2의 Help 메뉴에서 제공되는 도움말을 참고하면 된다.

<별첨 자료> 완성된 코드

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
    creationComplete="simpletest.send();" fontSize="12" >
        <mx:Script>
        <![CDATA[
                import mx.collections.ArrayCollection;
                [Bindable]
                private var reviewList:ArrayCollection;
                private
function myDateFormat(item:Object,column:DataGridColumn):String
                {
                        return myDate.format(item[column.dataField]);
                }
        ]]>
    </mx:Script>
    <mx:HTTPService id="simpletest"
        url="testData.xml"
        result="reviewList=simpletest.lastResult.list.review"/>
    <mx:DateFormatter id="myDate" formatString="DD-MMM-YY"/>
    <mx:DataGrid dataProvider ="{reviewList}">
        <mx:columns>
        <mx:DataGridColumn  dataField="reviewer"  id="reviewerCol"
                headerText="이름"/>
        <mx:DataGridColumn dataField="review_date"
        headerText="날짜" labelFunction="myDateFormat"/>
        </mx:columns>
    </mx:DataGrid>
    <mx:Button label="Button"/>
```