EMBEDDED SYSTEMS PROGRAMMING 2014-15

UI Specification: Approaches

UIS: APPROACHES

- **Programmatic** approach: UI elements are created inside the application code

- **Declarative** approach: UI elements are listed in a data structure that is external to the code, albeit linked to it in some way.
  The data structure can be usually accessed with a visual editor

- A mixed approach is possible

PROGRAMMATIC APPROACH: PROS AND CONS

- ➕ Flexibility
- ➕ UI can be built at run time
- ➖ Not clear where/what to change to modify the UI
- ➖ Modifications imply recompilation
- ➖ Difficult to support multiple languages and/or multiple screen sizes

DECLARATIVE APPROACH: PROS AND CONS

- Better design: the presentation of the application is well separated from the code that controls its behavior
- Modifications concentrated in one point.
  And no need to recompile!
- Easy to support multiple languages and/or multiple screen sizes

Bottom line: **go declarative**

DECLARATIVE APPROACH: ANDROID

- UI data stored in **XML** files
- The XML vocabulary corresponds to the names of the `View` class/methods and its subclasses/methods

1. Write the XML code
2. The XML is compiled into a **resource**
3. Load the resource from your Java code

DECLARATIVE HELLOWITHBUTTON (1/4)

- Project file `res/layout/mylayout.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="horizontal">
    <Button android:id="@+id/bu"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/pressme" />
    <TextView android:id="@+id/tv"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="@string/pressplease" />
</LinearLayout>
```

DECLARATIVE HELLOWITHBUTTON (2/4)

- Project file `res/values/strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Hello World! (With Button)</string>
    <string name="pressme">Press me</string>
    <string name="pressplease">Press the button, please</string>
    <string name="goodjob">Good job!</string>
</resources>
```

DECLARATIVE HELLOWITHBUTTON (3/4)

- Source file `HelloWithButton.java` (1/2)

```
package it.unipd.dei.es1011.hellowithbuttond;


import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;


public class HelloWithButton extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        // Display the layout
        setContentView(R.layout.mylayout);

        // Get references to the TextView and the button.
        // Do it AFTER setContentView()! Before setContentView()
        // the objects have not been instantiated yet
        final TextView tv = (TextView)findViewById(R.id.tv);
        Button bu = (Button)findViewById(R.id.bu);
                                                                          ...
```

DECLARATIVE HELLOWITHBUTTON (4/4)

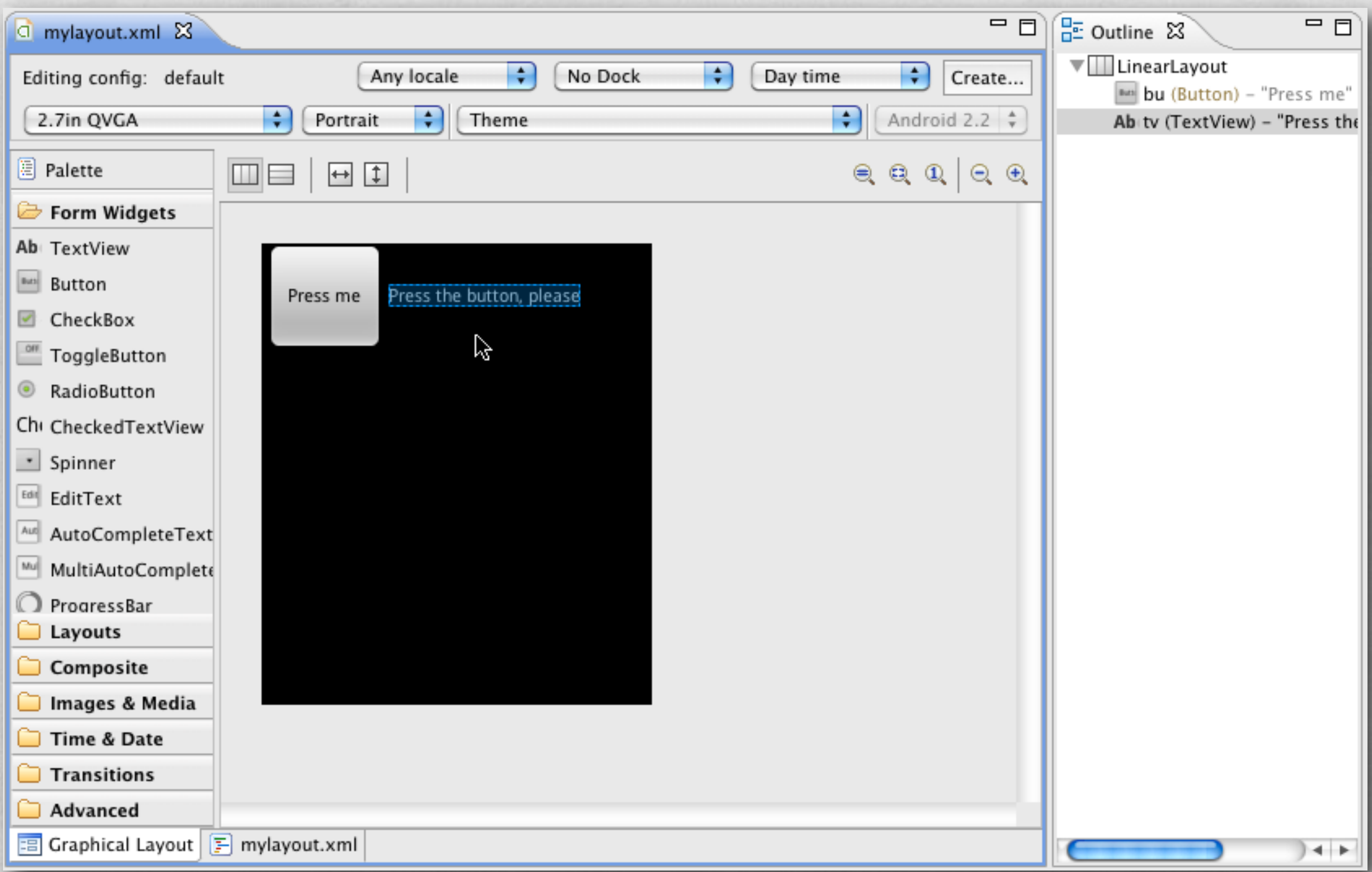
- Source file `HelloWithButton.java` (2/2)

```
...

        // Set the action to be performed when the button is pressed
        bu.setOnClickListener
        (
            new View.OnClickListener()
            {
                public void onClick(View v)
                {
                    // Perform action on click
                    tv.setText(getString(R.string.goodjob));
                }
            }
        );
    }
}
```

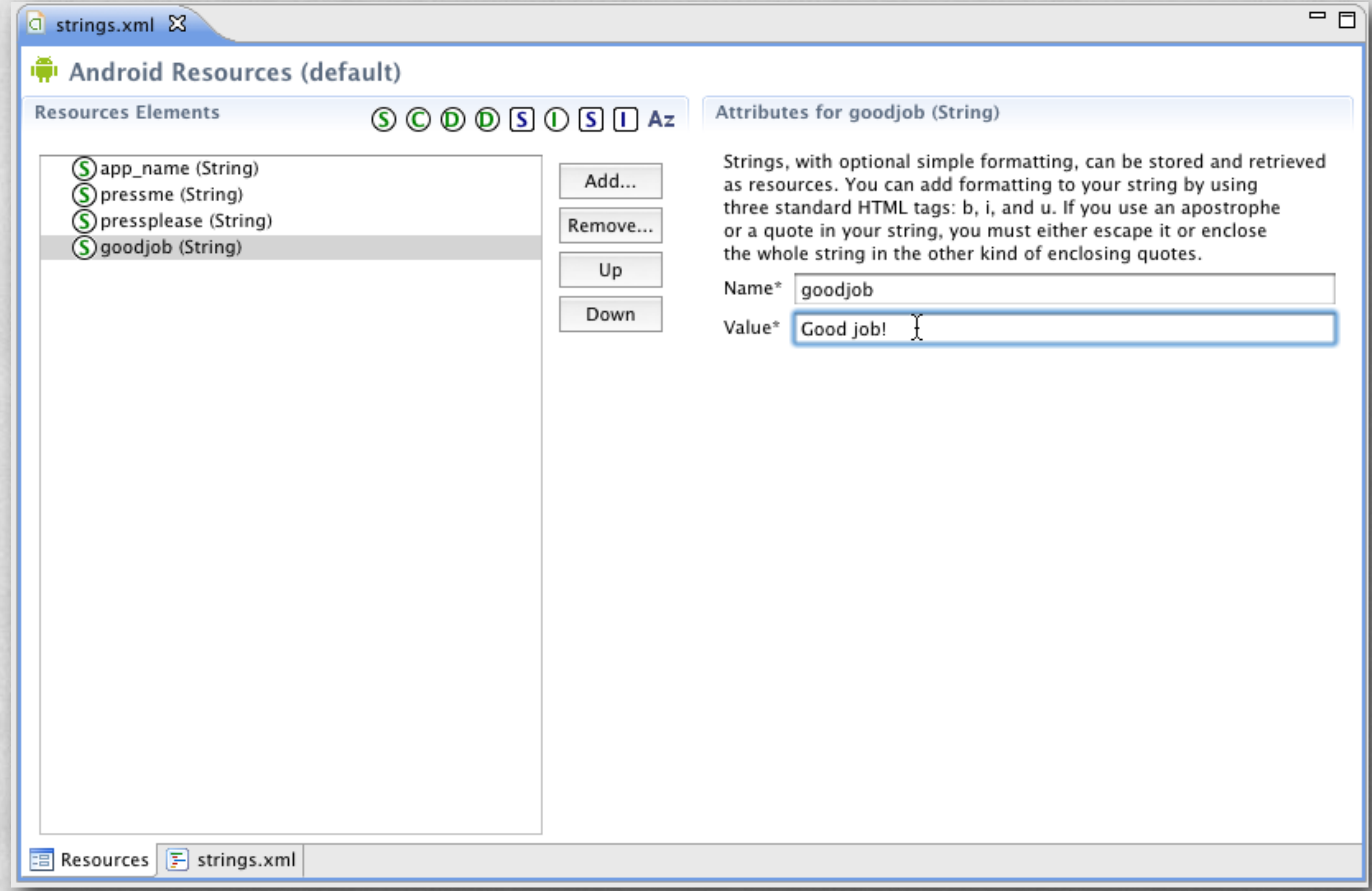
ANDROID: IDE SUPPORT (1/3)

- Visual editing of `mylayout.xml`



ANDROID: IDE SUPPORT (2/3)

- Visual editing of `strings.xml`



app_name (String)
pressme (String)
pressplease (String)
goodjob (String)

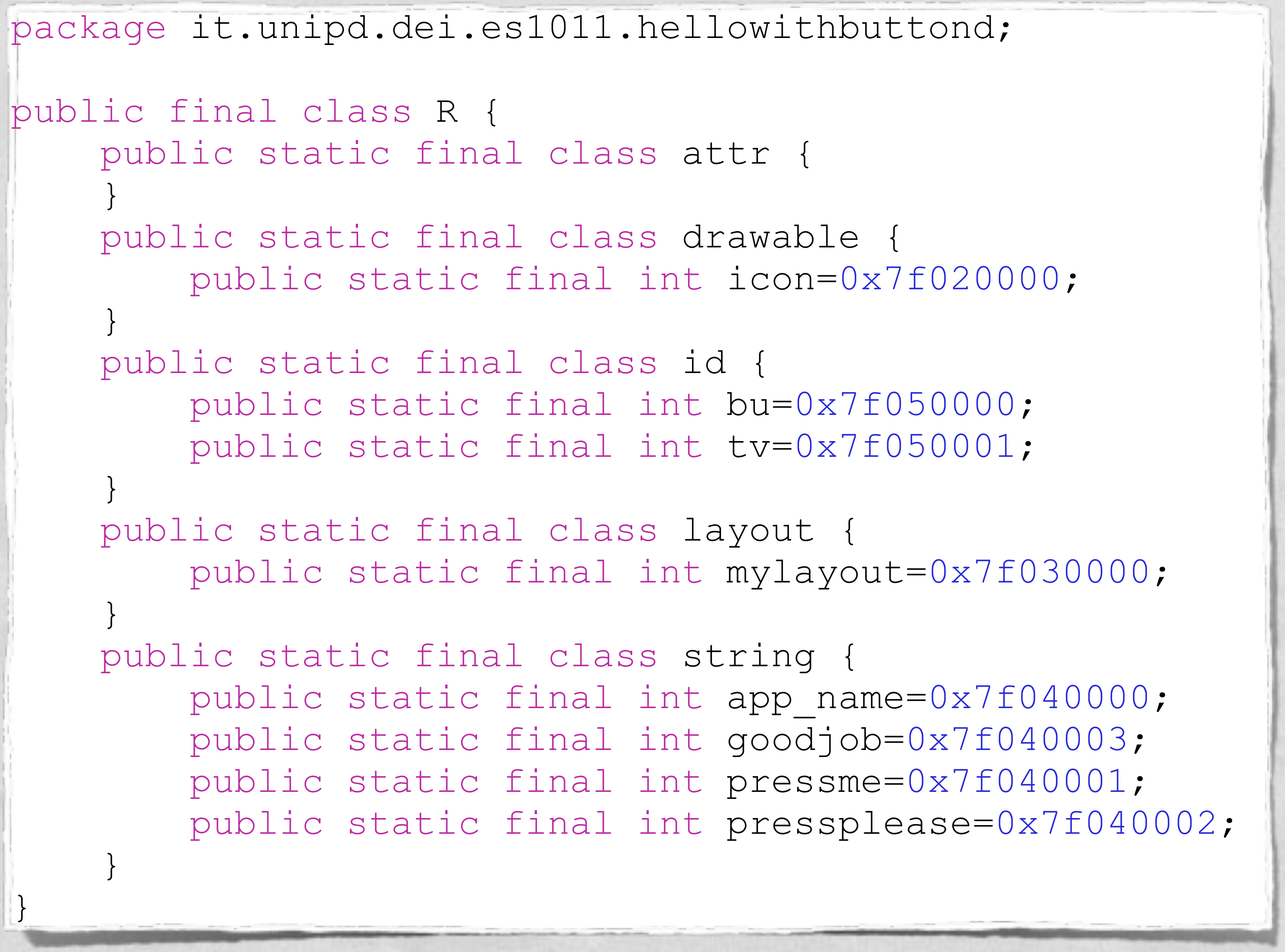
Attributes for goodjob (String)

Strings, with optional simple formatting, can be stored and retrieved as resources. You can add formatting to your string by using three standard HTML tags: b, i, and u. If you use an apostrophe or a quote in your string, you must either escape it or enclose the whole string in the other kind of enclosing quotes.

Name* goodjob
Value* Good job!

ANDROID: IDE SUPPORT (3/3)

- Autogenerated `R.java` source file



```java
package it.unipd.dei.es1011.hellowithbuttond;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int bu=0x7f050000;
        public static final int tv=0x7f050001;
    }
    public static final class layout {
        public static final int mylayout=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int goodjob=0x7f040003;
        public static final int pressme=0x7f040001;
        public static final int pressplease=0x7f040002;
    }
}
```

DECLARATIVE APPROACH: IOS

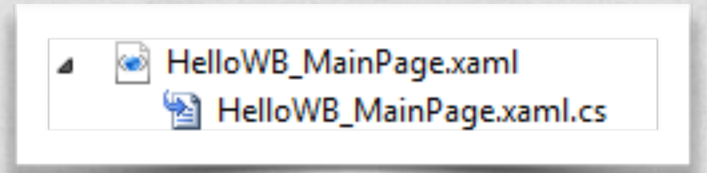
- UI data stored in **XIB** files or **Storyboard** files.
- Such files contain XML code but are not easy to read
- Objective-C code can refer to programmatically-defined UI objects via special instance variables called **outlets**.
  UI objects can invoke Objective-C code via special methods called **actions**

1. Create the UI with Interface Builder (inside Xcode)
2. Create outlets and actions in the source code
3. Establish the connections with Interface Builder

DECLARATIVE APPROACH: WINDOWS PHONE (1/2)

- Let us consider XAML applications

- Each page is associated with
  - one **.xaml file**
    that specifies the visual appearance of the page,
  - one **code-behind file** (written in C# or VB)
    that specifies the control logic of the page


HelloWB_MainPage.xaml
HelloWB_MainPage.xaml.cs

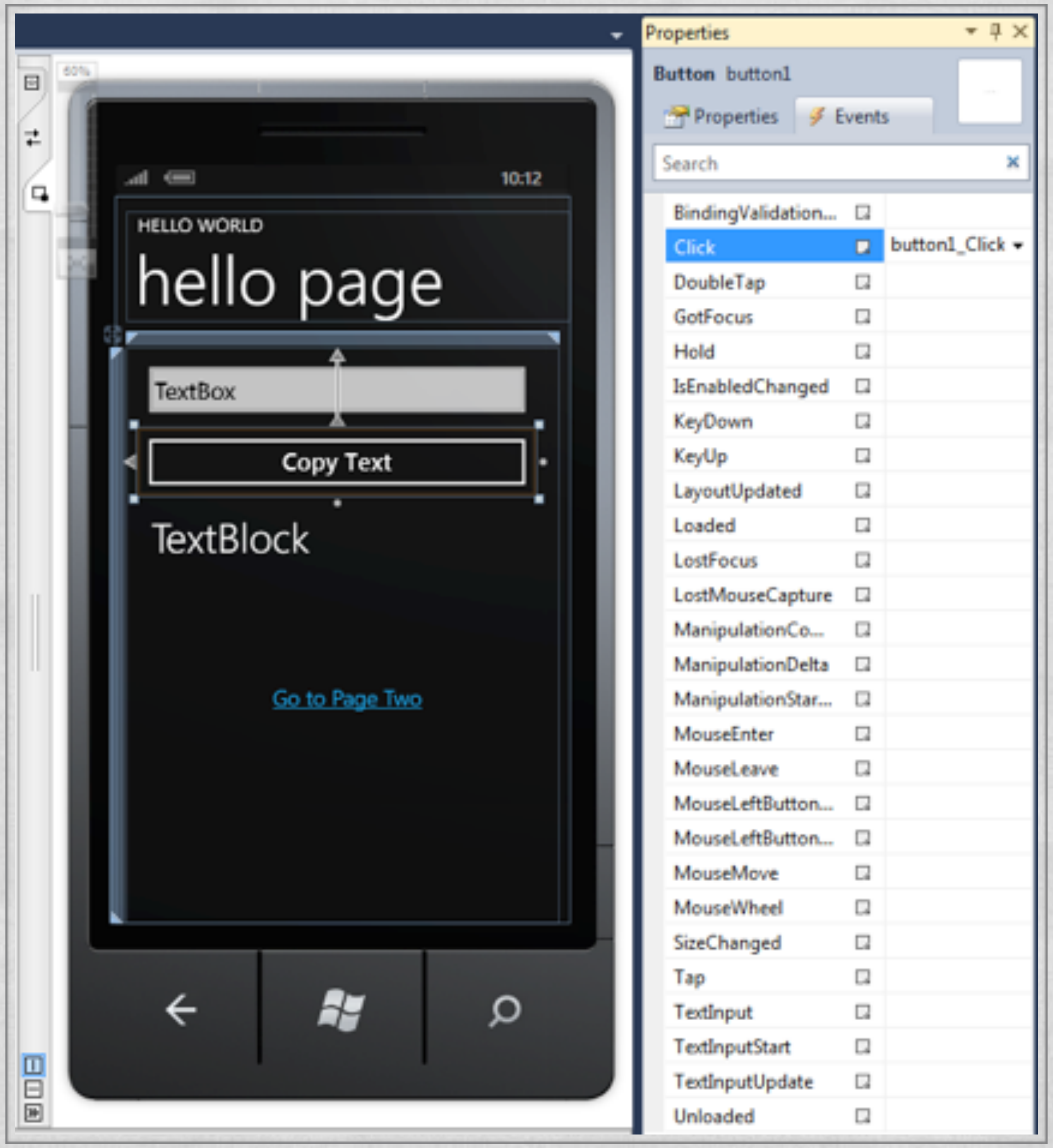
DECLARATIVE APPROACH: WINDOWS PHONE (2/2)

- Visual Studio allows to specify connections between UI events and methods in the code-behind file

```
namespace HelloWithButton
{
    public partial class MainPage : PhoneApplicationPage
    {
        ...

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            textBlock1.Text = "Good job!";
        }

        ...
}
```



Click: button1_Click