Hello World - Your First WebObjects Application

> This is deprecated information!

> This information is now located here

Create a New WebObjects Project

Make sure that Eclipse and WOLips are installed. Launch Eclipse, chosse *New->WebObjects Application* from the *File* menu.

Click 'Next'.
Type 'HelloWorld' as the 'Project Name' and click on the 'Set...' button to specify that you'd like to save your project to the desktop, or where ever you'd like to save your project. Note that earlier versions of WebObjects had problems building projects if their path had any spaces.

Click 'Next' lots more times until you come to a 'Finish' button, and click it (we'll cover all those other screens later).
If you did everything right, you'll end up with something that looks like this (depending on how you've configured Xcode).

Edit the 'Main' Component

You should now be looking at a brand-new WebObjects project. This is your 'blank canvas' if you're fond of analogies.
It may seem a little complicated if you're used to creating web sites that consist only of HTML files and images, but it isn't too much more complicated. The main files we're interested in are in the 'Web Components' group.

When you first create a WebObjects project, you will have just one 'Component', the 'Main' one.
A WebObjects component consists of a folder and a bunch of files, but at this stage we're really only interested in 2:

- Main.wo - which contains the interface
- Main.java - where we'll put the logic for the Main.wo page.

Double-click on the 'Main.wo' file to edit the interface in WebObjects Builder.

You should see something like the screen above. This is how a WebObjects component looks in WebObjects Builder. The top half of the window is a What-You-See-Is-What-You-Get (most of the time (WYSIWYG(MOTT)) HTML editor where you can design your pages. The bottom half of the window is used to connect interface elements to variables in the 'logic' section of your web application.
Add the text "Hello, what is your name?" by typing in the top half of the window.

Add some form elements to the page by selecting a 'WOForm', a 'Text Field' and a 'Submit Button' from the 'Forms' menu bar item.
Finally, add a place-holder String2 by selecting 'String' from the 'WebObject' menu bar item.
You now have a finished interface: a heading inviting visitors to your web application to enter their name, a text field where they can type their name, a button so they can submit the form and a place-holder for the string of characters that you want to display in response.

Add an Action and a Key

Now that you have an interface, you need to write the logic of your first WebObjects application. This application allows a user to type their name in to a form and displays a message in response. In order to achieve this, we're going to write a little Java code.
Each WebObjects component has a Java file that contains the logic for the component. Java is an object-oriented programming language so if you're not familiar with object-oriented programming the next section might introduce some new concepts. At this stage you don't need to know the gory details of object-oriented programming, but it might help to understand some basic terminology.
Each Java file contains a class which is like a recipe (it seems I like the analogies). The class contains both variables which are kind of like ingredients and methods which are like the steps in a recipe.
The main difference between a Java class and a recipe is that with Java you don't know what you're baking until you start, so whilst a recipe might say 'take 1 cup of flour and mix it', a Java class will say 'take 1 cup of stuff and mix it' and you get to decide exactly what stuff is when the application is running.
Anyway, enough background - let's get our hands dirty. In order for our application to behave in the way that we want we need to modify the 'Main' class associated with our page. We want to add two things: a variable (think ingredient) to keep track of what the user types in and a method (think step) outlining what we should do with the variable.
To make matters even more interesting, WebObjects builder uses its own terminology when talking about variables and methods. It refers to 'Keys' and 'Actions'. I'm going to do what WebObjects should and make things simpler by using the terminology that the rest of the world uses to refer to Java classes, so when WebObjects says 'Key' think 'Variable' and when you hear 'Action' think 'Method' (you'll get used to it).

So, to add a variable to keep track of our visitors name, click on the 'Interface' menu bar item and select 'Add Key...'.

In the property sheet that slides down, type visitorsName (the capitalisation is a convention when programming in Java) in the 'Name' field, 'String' in the 'Type' field and make sure 'An instance variable' is ticked before clicking 'Add'.

You should now see 'visitorsName' in the bottom-half of the window under 'Application' and 'Session'. This indicates that the Java class associated with this component has a variable called visitorsName that is available to 'bind' to interface components.
Repeat this step to add a 'message' variable to the 'Main' component.
Next, we need to add a method to the class that we will execute when visitors click on the submit button.
Click again on the 'Interface' menu item and this time choose 'Add Action...'.

Type 'sayHello' as the 'Action name' and click 'Add'.
Open up the Main.java file to see the variables and method that have been added to this file. It should now look like the code below:

```
public class Main extends WOComponent {
    public String visitorsName;
    public String message;

    public Main(WOContext context) {
        super(context);
    }

    public WOComponent sayHello() {
        return null;
    }
}
```

Connect the Interface to the Logic

Now you've created the interface and added a variable and method to the underlying Java code, all that we need to do is 'hook-up' the interface to the logic.
We need to specify that the 'sayHello' method should happen when the button is clicked and that the value for the 'visitorsName' variable should come from the text field. To do this we need to switch back to WebObjects Builder.

Click and drag from the space next to 'visitorsName' in the bottom half of the window to the text field and choose 'value' from the drop-down menu to indicate that you want the 'visitorsName' variable to get its value from the text field.
Repeat this step to bind the 'value' of the dynamic String to the 'message' variable and the 'action' of the 'Submit' button to the 'sayHello' method in the underlying Java class.

Once you've hooked up the interface to the logic, the only thing left to do is write a line (or two) of Java code to compose a custom message in response to the click of the button.

Edit the Java Code

Open the 'Main.java' file by double-clicking on it.
Fill out the details of the sayHello method like the example below. Basically, when this method is executed we want to construct a custom message based on the variable 'visitorsName' which will contain the name of the person we want to greet.

```
public class Main extends WOComponent {
    public String visitorsName;
    public String message;

    public Main(WOContext context) {
        super(context);
    }

    public WOComponent sayHello() {
        message = "Hi there, " + visitorsName + "!";
        return null;
    }
}
```

Congratulations - you've just written your first WebObjects application. Make sure you've saved both the Main.wo file in WebObjects Builder and the Main.java file in XCode and you're ready to compile your code and run the application. Click on the 'Build and Go' button in XCode to test your application.