

# PHP Basics: Working with Forms with GET and POST

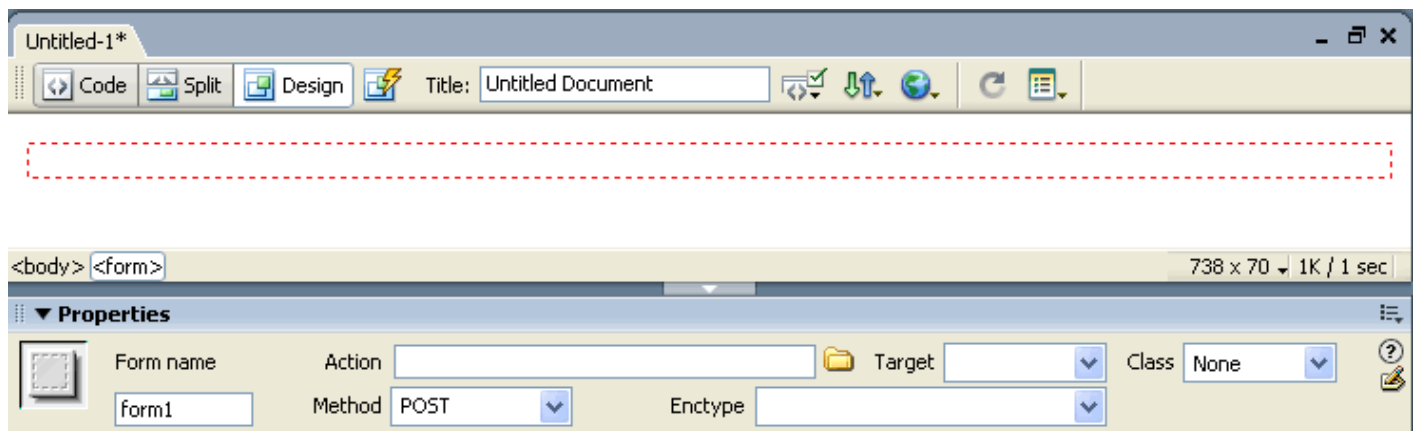
Forms are vital to almost every website, as they allow the user to interact with the site and to send data back to the web server. This data could be used in many different ways, for example it could be used so the user can enter custom search parameters to query a database, or it could be used for the user to enter their details to place an order on your site.

There are two methods which forms can use to submit data, either using the GET method or the POST method, and we'll be looking at both these methods in this tutorial. Before we start looking at reading the data submitted by a form, we'll start by having a quick look at creating an HTML form using Dreamweaver MX.

To start, open Dreamweaver MX, and then create a new PHP so that we have a blank page on which we can create our form. First, we'll add a form to the page and then we'll look at the various types of elements that can be added to the form.

## Adding a Form

To create a new form on the page, select **Insert > Form > Form** from the main Dreamweaver MX menu. Once you've done this, a new form will be added to the page, indicated by a red dotted line as shown in the figure below.



**Figure 1 – Empty Form & the Form Properties Panel**

When the form is selected, the Properties panel will change in Dreamweaver MX as shown in figure 1, which allows you to change the various properties for the form. These properties are explained below:

- **Form Name** – This is a unique name for the form, used to refer to the form when you want to change its properties through JavaScript or VBScript.
- **Action** – When a form is submitted, the data entered in the form is sent back to a certain page on the web server, and the page that the data is to be sent to is set here. This can be the same page that the form is on, or a different page on your web server.

- **Method** – The Form Method can be set to GET or POST. We'll be looking at more details about both of these methods later in this tutorial.
- **Target** – The options for the target specify which browser window is to be affected by the form submission, and these are exactly the same options as you would have for a normal HTML link.
- **Enctype** – The form Enctype specifies how the data is sent to the web server, and is used to change the way the data is sent to the server. For the purposes of this tutorial it can just be left blank, as these options are only needed in special circumstances, such as uploading files, which is beyond the scope of this tutorial.

Now that we've looked at creating an empty form, we're going to look at the various form elements that can be used on the form.

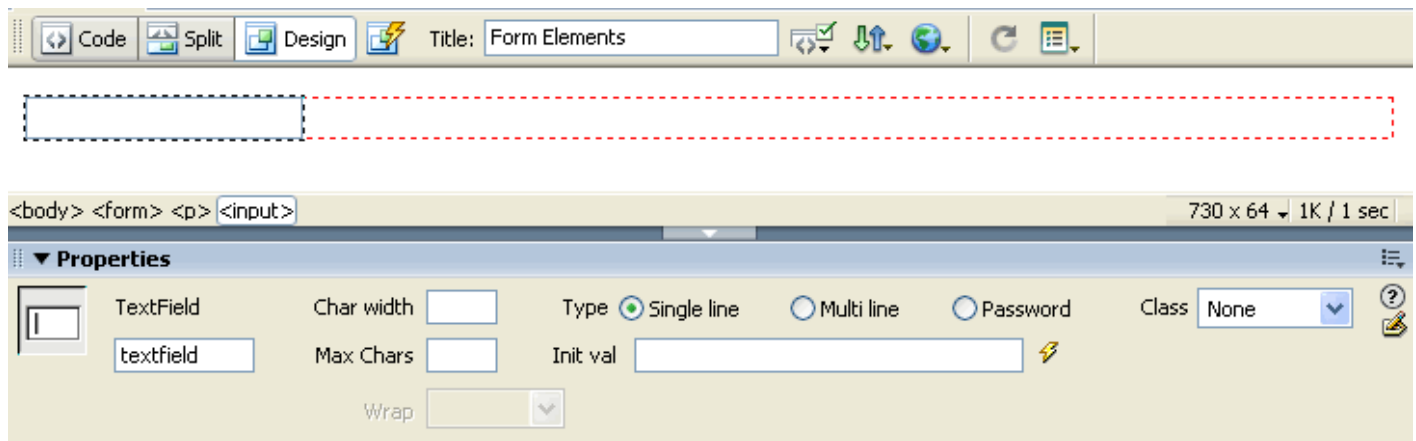
## HTML Form Elements

In this section, we're going to run through the different form elements that can be used to collect data from the user. For each element we'll look at the element itself and the various properties that can be set.

### Text Field

#### Insert > Form > Text Field

The first form element that we're going to look at is a text field, which as the name suggests, allows the user to enter text. By default this is a single line. A standard text field is shown in the image below, along with the Dreamweaver MX properties panel showing the options available.



**Figure 2 – Default text field and properties**

The available options for a text field are:

- **Char width** – This sets the length of the text field, i.e. how long it appears on screen, although this is better set using the width attribute of a CSS class.
- **Max Chars** – This sets the maximum number of characters that a user can enter in the text field. It's important to set this so that the user cannot enter more characters than your application expects. For

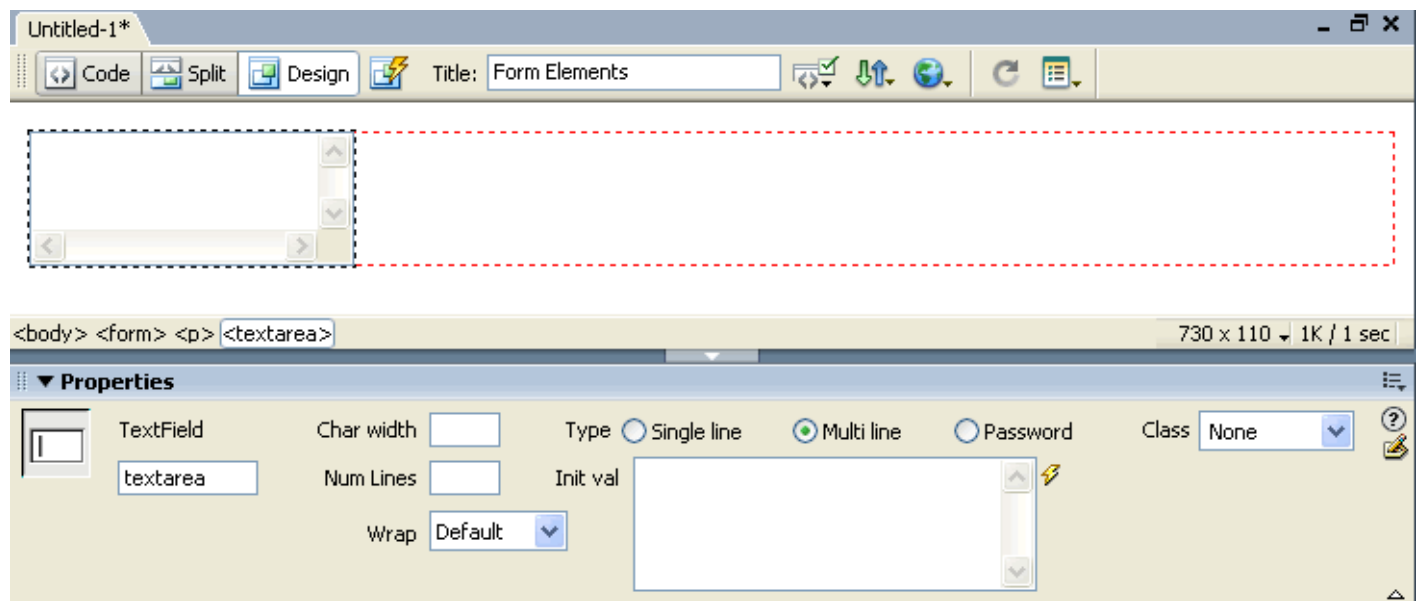
example, if the text entered into the field is being inserted into a database field which can accept a maximum of 20 characters, you should enter 20 in this field.

- **Type** – This sets how the text box behaves. For the default text box, shown above this is set to single line. Multi line creates a larger text field which allows more than one line of text to be entered, and we'll look at this separately. Password is as the name suggests is for password fields and allows the user to enter text as normal, but the text is hidden so that no one can see the text that has been entered.
- **Init val** – This option lets you set a value that is displayed in the text field when the form first loads.

## Text Area

*Insert > Form > Textarea*

A text area is almost identical to a text box, except that rather than accepting a single line of text, it can accept more than one line to allow the user to enter an address, for example. An example text area and its properties are shown in the image below.



**Figure 3 – Text Area and Properties**

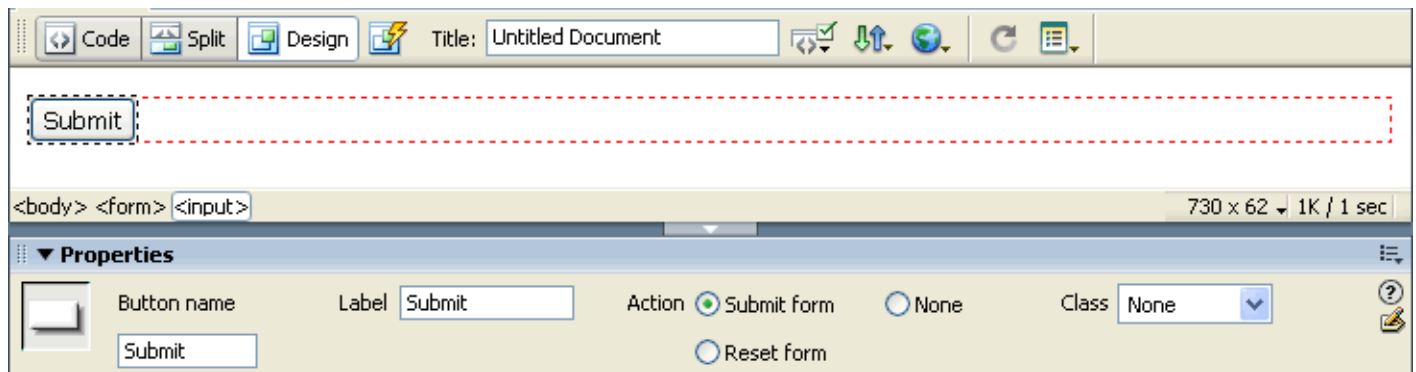
The only differences in the properties compared to a normal text field are:

- **Num Lines** – This setting controls the height of the text box, i.e. how many lines of text it can show at one time without scrolling.
- **Wrap** – The wrap setting defines how text is displayed inside the text area.

## Buttons

*Insert > Form > Button*

Buttons allow the user to perform certain actions, such as submitting the form, or resetting the form to its original state, clearing any information that was entered. An example button and its properties are shown below.



**Figure 4 – Example Button and properties**

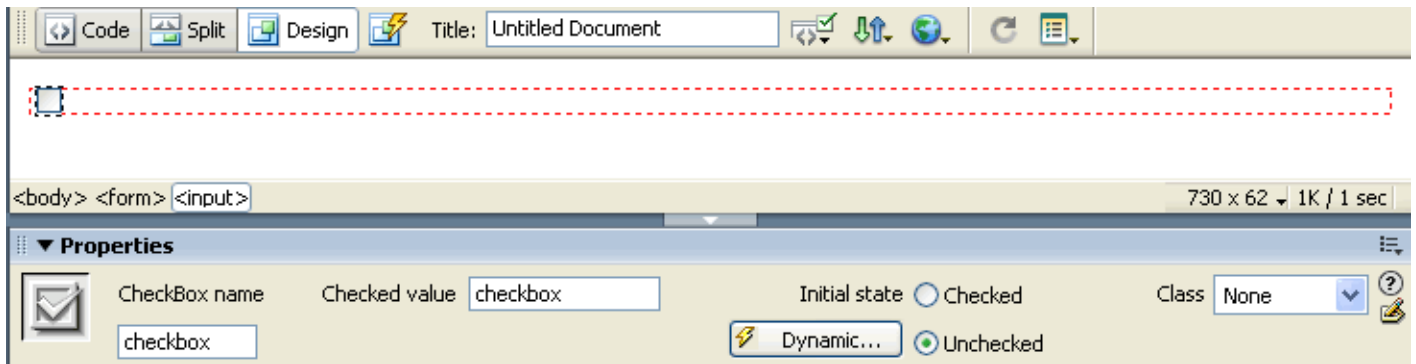
The available properties for a button are:

- **Button Name** – The name of the button
- **Label** – The text that will be displayed on the button
- **Action** – The action defines what will happen when the button is clicked. "Submit form" will submit the form when the button is clicked, "Reset form" will clear the form and reset it back to its default state. "None" will give the button no action, this should be used if you are attaching some JavaScript to the button.

## Check Box

*Insert > Form > Checkbox*

A check box allows the user to select an option, which can either be on or off. An example check box and its properties are shown in the image below.



**Figure 5 – Example Checkbox and properties**

A checkbox has the following properties:

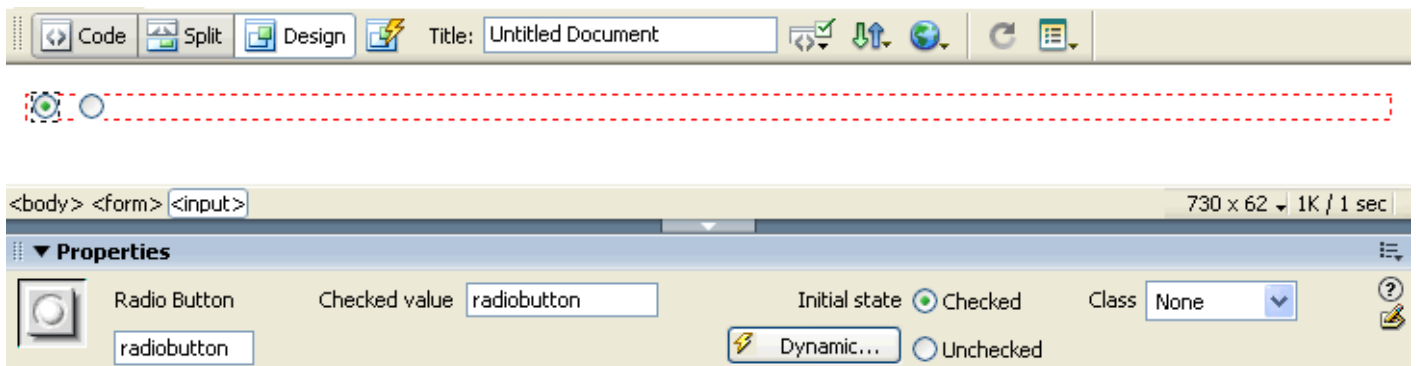
- **CheckBox name** – The name for the checkbox
- **Checked value** – The value returned by the checkbox if its selected
- **Initial State** – Whether the checkbox is set as selected or unselected when the form is initially loaded

## Radio Buttons

### Insert > Form > Radio Buttons

Radio buttons are similar to check boxes, in that they can only either be on or off. However, radio buttons can be set in groups so that only one button in a particular group can be selected, forcing the user to select a single option from a group.

An example of a group of radio buttons and their properties are shown below:



**Figure 6 – Example of Radio Buttons and their available options**

The properties for radio buttons are exactly the same as for check boxes, as shown below:

- **Radio Button Name** – This holds the name of the radio button. In order to create a group of radio buttons, where only option can be selected, all the radio buttons in the group should be given the same name.
- **Checked value** – This contains the value which will be returned by the radio button when it's selected. In a group of radio buttons, each radio button in the group should be assigned a different checked value.
- **Initial state** – This option sets whether a radio button will be displayed as checked or unchecked by default.

## List / Menu

### Insert > Form > List/Menu

A list/menu allows a user to select an option from a list. A menu allows the user to select from a dropdown menu, whereas a list shows all the options at once. An example menu and the relevant properties are shown below:

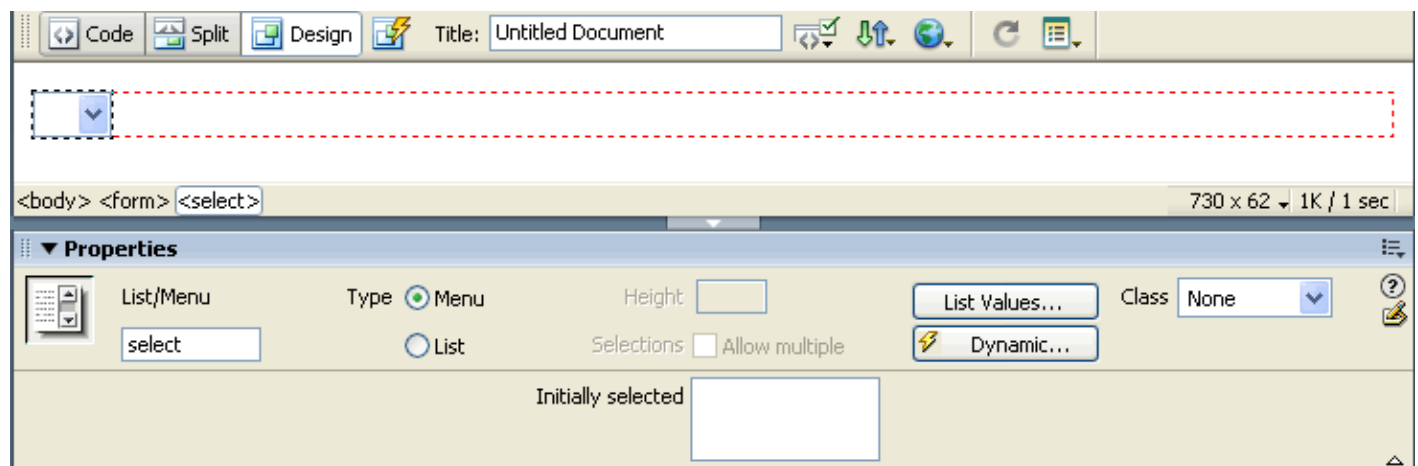


Figure 7 – Example Menu and properties

A List/Menu has the following properties:

- **List/Menu Name** – The name of the List/Menu
- **Type** – This sets how the options will be displayed. If "Menu" is selected the options will be displayed in the form of a dropdown menu. If "List" is selected, then the options will be displayed in the form of a list.
- **Initially selected** – This sets which option in the list will be selected when the form is first loaded.
- **List Values** – This button allows you to set the options and corresponding values that will be displayed in the list/menu. When the button is clicked, a new window opens as shown below.

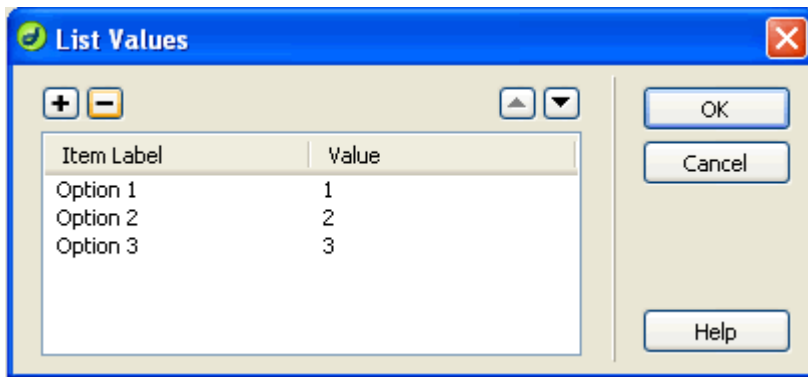


Figure 8 – The List values window

To add an option, click the + button, and then enter the label for the list option, and the value that will be returned when the option is selected. When you have entered all the options that will appear in the list, click OK.

## Hidden Field

### Insert > Form > Hidden Field

The final option that we're going to look at in this section is a hidden field. A hidden field as the name suggests cannot be seen on the form, but is used to hold extra information that is passed when the form is submitted.

An example of a hidden field and its corresponding properties are shown below:

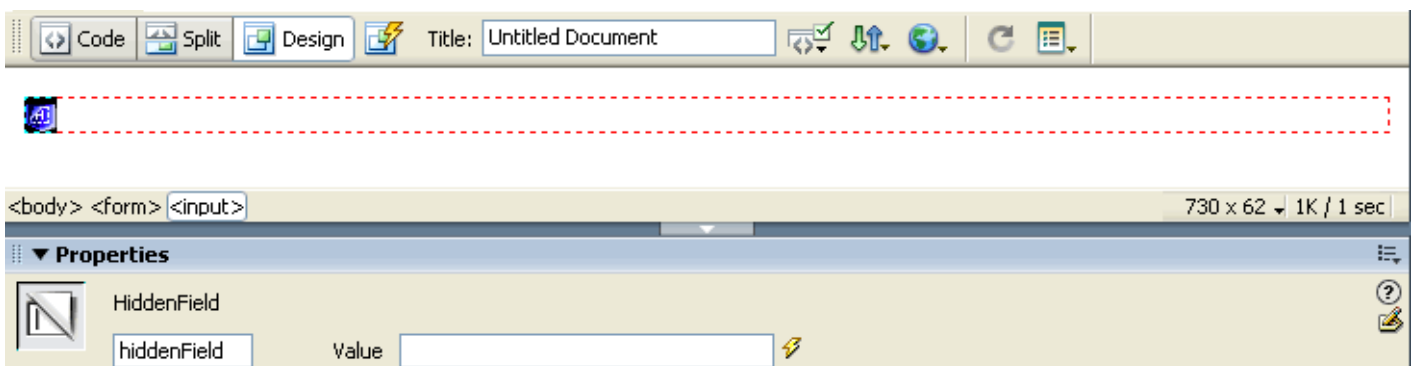


Figure 9 – Hidden Field and corresponding properties

A hidden field is very simple and has the following two properties:

- **Name** – The name of the hidden field
- **Value** – The value that will be contained in the hidden field

Although the hidden field cannot be seen in the browser, it can be seen if the user selects view source and looks at the HTML code that makes up the page. Therefore, it should never be used to hold confidential information.

Now that we have looked at the various HTML form options open to us, we will next move on and look at how we can use these options, and how to read back the entered data when a form has been submitted.

## Reading Data submitted by a Form

As mentioned at the start of this tutorial, there are two methods for reading data sent from a form, and we'll be taking a look at both of these methods now.

### Reading data with the GET method

The first method that we are going to look at is the GET method, where the data from the form is sent through the URL as shown in the example below:

```
http://www.mysite.com/mypage.php?firstname=gareth&surname=downes-powell
```

This method isn't often used for forms, as there a couple of limitations which are discussed below, but in some cases it may be all that's needed.

### Limitations of the GET Method

- **Amount of Data that can be Sent**

The first limitation of the GET method is that there is a limit to the amount of data that can be sent through a URL. As you can see from the example URL above, the data from the form is added onto the URL in the form of a query string.

However, there is a limit to the length of a URL and this varies between browsers. In Internet Explorer the maximum length of a URL including the actual URL path itself is 2048 characters, so this limits the amount of data that can be sent by the form.

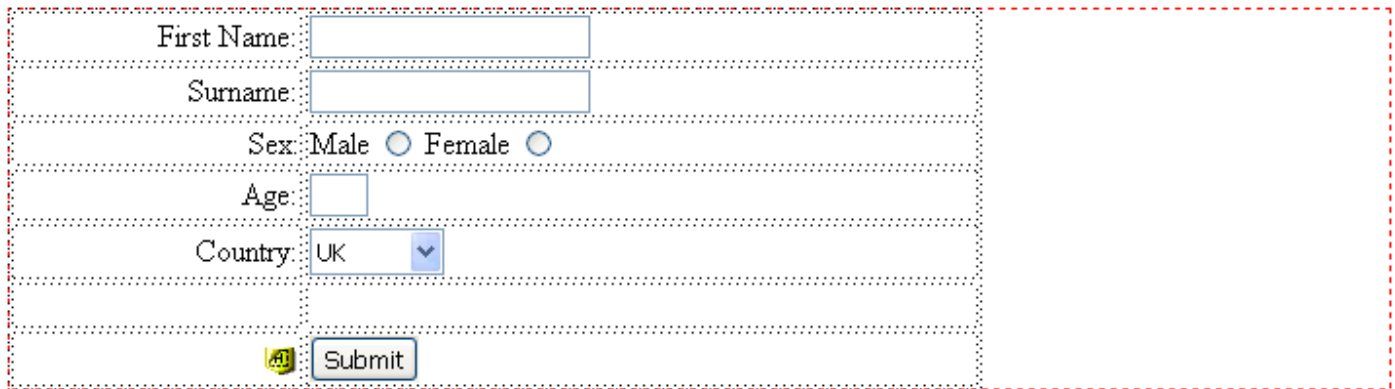
- **Security**

Another possible problem with the GET method is that because the data is sent through the URL, it means that the data can easily be changed by the user. Whilst this isn't a problem for most of the data as the user has entered it in the first place, however, there may be information that you would not want changed - such as the data entered in hidden fields.

If either of these limitations apply to you then its best to use the POST method for the form data which we'll look at later in this tutorial.

### Creating an Example Form

In order to have an example to work with, create the form shown below in Dreamweaver MX.



The image shows a web form with the following fields:

- First Name:
- Surname:
- Sex: Male  Female
- Age:
- Country: UK
- Submit:

**Figure 10 – Example Form**

Save the page as `getexample.php`. Next, create the form shown above, with the following details:

- **Form** – Action set to `getexample.php`, Method set to **GET**
- **First Name** – Text field – Name set to `firstname`
- **Surname** – Text field – Name set to `surname`
- **Sex** – Radio Buttons – Both gave the name set to `sex`. The first radio button has a checked value of `male`, and the second radio button has a checked value of `female`.
- **Age** – Text Field – Name set to `age`
- **Country** – List/Menu – Type set to Menu. Options set to UK, France, Spain for both Label and Value
- **Hidden Field** – Name set to `action`. Value set to `send`
- **Button** – Action set to `Submit Form`

Save the page, and next we'll add the PHP code to the page to read back the form data.

## Adding the PHP Code

Next, we're going to add the PHP code required to read back the form data. To do this we use a special PHP array called `$_GET[]`. This is an associative array containing the data for each form field, where each element's key is the name of the field on the form.

As a simple example of the `$_GET[]` array, switch to code view, and add the following code to your page underneath the final `</form>` tag.

```
<?php
if($_GET['action'] == "send"){
    echo "<pre>";
    var_dump($_GET);
    echo "</pre>";
}
?>
```

Here, we first check to see if the form has been submitted by checking for the presence of the hidden field, which is why we added it to the form. When the form is first loaded, the code is not run as the hidden field will not be present until after the form has been submitted. If the hidden field is present, then we use the `var_dump()` command to output the whole contents of the `$_GET` array.

Load the page in your browser, and then enter some values and submit the form. Example output is shown in figure 11 below. You can also see that the data is contained in the URL, as shown in the example below which corresponds to the output shown in figure 11.

```
http://localhost/getexample.php?firstname=Gareth&surname=Downes-  
Powell&sex=male&age=26&country=UK&action=send&Submit=Submit
```

Here you can see that each field of the form is present, along with the data that was entered in the form:

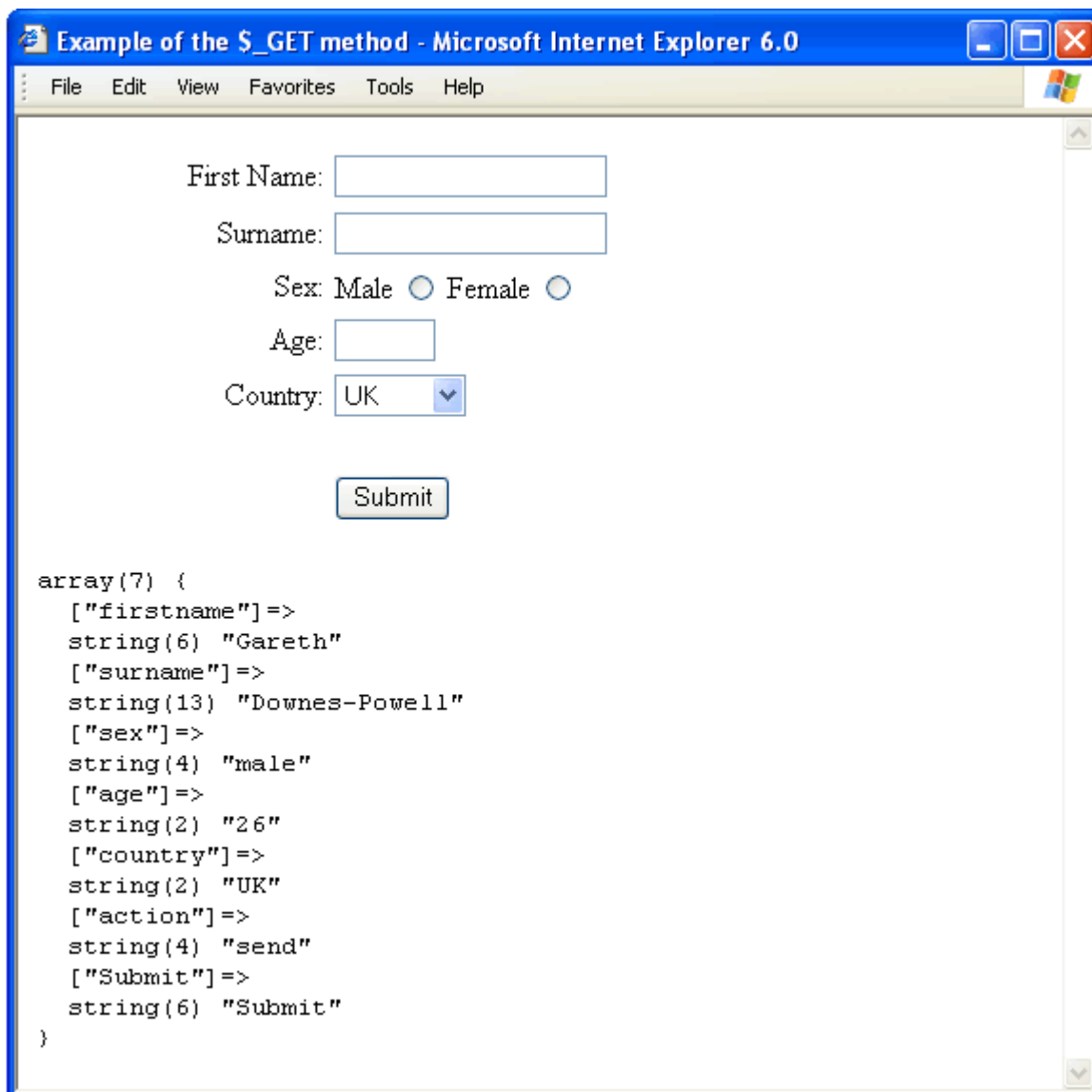


Figure 11 – Example output

To refer to individual values from the form, you use the following format:

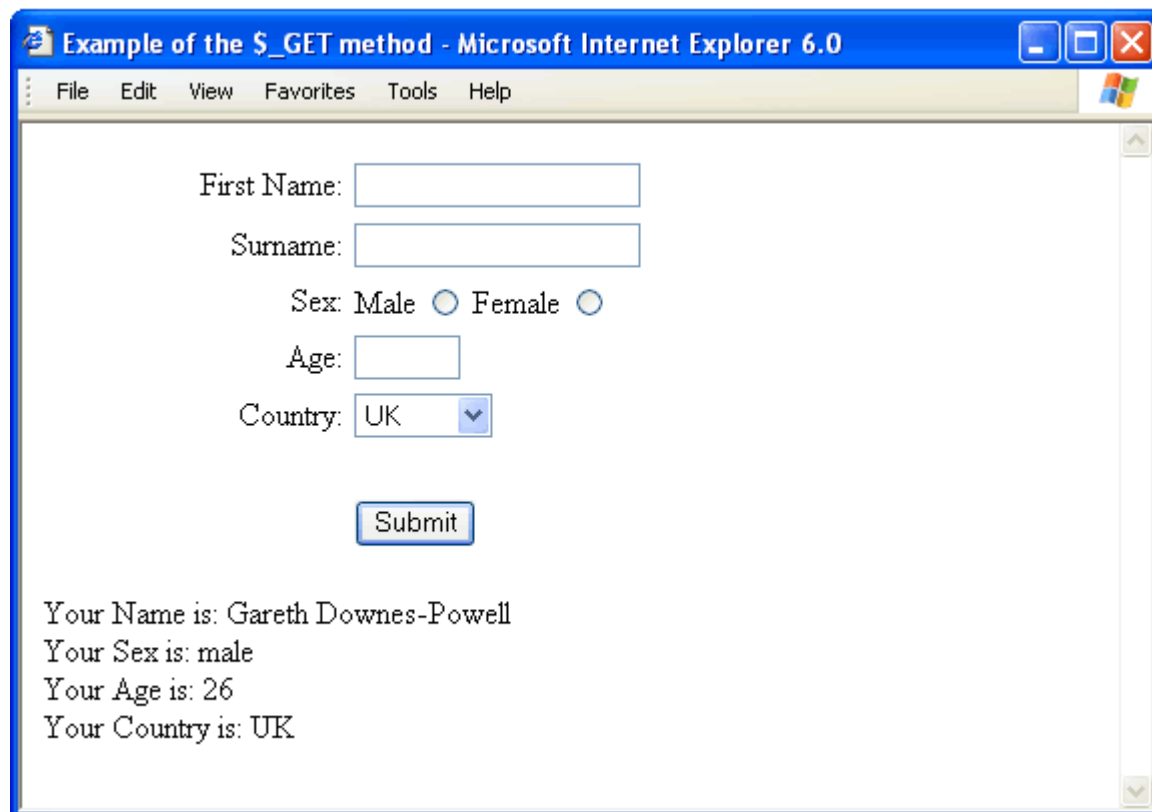
```
$_GET['fieldname'];
```

As an example, add the following code to the page in place of the previous PHP code.

```
<?php
if($_GET['action'] == "send"){
    echo "Your Name is: " . $_GET['firstname'] . " " . $_GET['surname'] . "<br>";
    echo "Your Sex is: " . $_GET['sex'] . "<br>";
    echo "Your Age is: " . $_GET['age'] . "<br>";
    echo "Your Country is: " . $_GET['country'] . "<br>";
}
?>
```

You can see that in the code above, we use the `$_GET[]` array, along with the relevant fieldname to extract the data that we require. Load the page in your browser again, and then fill in and submit the form.

An example of the new output is shown in figure 12 below:



The screenshot shows a web browser window titled "Example of the \$\_GET method - Microsoft Internet Explorer 6.0". The browser's address bar is empty. The page content includes a form with the following fields: "First Name:" (text input), "Surname:" (text input), "Sex: Male" (radio button), "Female" (radio button), "Age:" (text input), and "Country:" (dropdown menu with "UK" selected). Below the form is a "Submit" button. The output of the form is displayed below the submit button: "Your Name is: Gareth Downes-Powell", "Your Sex is: male", "Your Age is: 26", and "Your Country is: UK".

**Figure 12 – Example Output**

You can see that this output is much neater, and you can see how to use the form data using the `$_GET[]` array. In the final section of this tutorial, we're going to look at using the POST method.

## Reading Form Data with the POST method

The POST method has two main advantages over the GET method for working with form data.

The first is the amount of data that can be sent. Because the data is sent in a page header, and not through the URL, much more data can be sent, and so the form can request a lot more information from the user. The second advantage is that because the data is not sent through the URL, it's much harder for the user to modify the data contained in hidden variables for example, so this method is much more secure.

To access the data sent through the POST method, you use an almost identical method to the GET method, except that you use a different array called `$_POST`. Again, the fields are contained in this array with the fieldname as the element key.

We'll now change our simple form so that it uses the POST method instead of GET.

Open the page `getexample.php` from the previous section and then save the page as `postexample.php`.

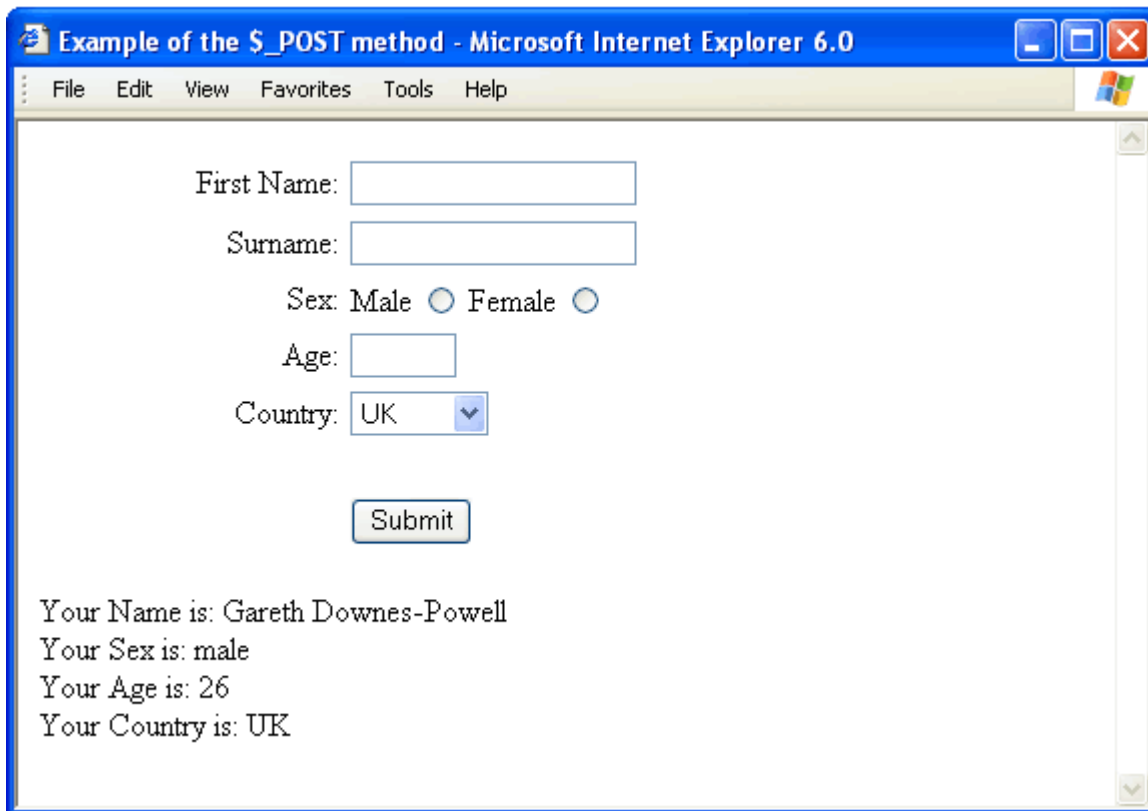
Next, select the form in Dreamweaver MX, and change the form method to **POST** in the properties panel. You'll also need to change the form action to point to the new name for this page i.e. `postexample.php`.

Switch into code view, and change the previous block of code so that it reads:

```
<?php
if($_POST['action'] == "send"){
    echo "Your Name is: " . $_POST['firstname'] . " " . $_POST['surname'] . "<br>";
    echo "Your Sex is: " . $_POST['sex'] . "<br>";
    echo "Your Age is: " . $_POST['age'] . "<br>";
    echo "Your Country is: " . $_POST['country'] . "<br>";
}
?>
```

You can see that the code is almost exactly the same, but this time we refer to the fields in the `$_POST[]` array.

Again, test the page in your browser, and you should see output similar to that shown in figure 13 below.



The screenshot shows a Microsoft Internet Explorer 6.0 window titled "Example of the \$\_POST method - Microsoft Internet Explorer 6.0". The browser's address bar is empty. The main content area displays a form with the following fields and values:

- First Name:
- Surname:
- Sex: Male  Female
- Age:
- Country:  (dropdown menu)

Below the form is a "Submit" button. The output of the form submission is displayed below the button:

```
Your Name is: Gareth Downes-Powell
Your Sex is: male
Your Age is: 26
Your Country is: UK
```

**Figure 13 – Example Output from the POST example**

You can see the output is the same, but this time if you look at the page URL you'll see that the URL is similar to:

```
http://localhost/postexample.php
```

And you'll notice that the form variables are no longer displayed in the URL.

## Summary

In this tutorial, we first looked at how to create an HTML form, along with the form elements that can be used, and the corresponding properties.

We then created a simple example form using a number of different form elements, and saw how to read the data back when it's sent using the GET method through the URL. We then altered the form so that is sent the data using the form POST method, using the `$_POST` array to read back the data.

Through this tutorial, we've looked at the basics needed to create a working HTML form. Although the forms have been fairly simple, as we progress through this series we will be working with more complex forms, and using the data submitted from the form to insert into a database, or to be sent through email for example.