Event Handling

The direct-manipulation style of interaction used in GUIs relies heavily on drag-and-drop, but for a long time drag-and-drop was not supported directly by web-browsers.

Drag-and-drop could only be implemented using plug-ins and proprietary technologies (such as Flash).

The W3C's DOM Events Specification standardised a wide range of events within browsers, including mouse-movements, and most browsers fully support it.

Events recognied by web-browsers include:

- click
- mousedown
- mouseup
- mouseover
- mouseout
- etc.

If you want to use an event merely to trigger an action, you can do it by creating a suitable *event listener* in the HTML code, e.g.:

```
<button onclick='myFunction()'>Click Here</button>
```

Clicking the button will cause the function named `myFunction()` to be executed.

However, events can be used to do more than just trigger an action.

Each time an event occurs, an *event object* is created.

It contains various information about that event:

- Which mouse-button or key was pressed
- Where the cursor was when the event occurred (within the browser window or the entire screen)
- What element the cursor was over when the event occurred (a button, an image, the document, etc.)

Using the information contained in the event object, you can create web-pages that respond to a wide range of user actions.

For example, you can obtain the cursor coordinates and then move an image to the cursor position.

If you do this repeatedly, the image will follow the cursor: in this way you can create drag-and-drop functionality.

To use the event object, you must first set up an event listener in accordance with the W3C standard.

This is done using the method `addEventListener()`.

The `addEventListener()` method is used as follows:

```
element.addEventListener(event,action,capture)
```

where:

- `event` is the event to listen for, e.g., `click`
- `action` is the action to be taken or the name of the function to be called when the specified action is detected
- `capture` is a Boolean value which selects either 'capturing' or 'bubbling' of events.

The `capture` parameter is usually set to `false`. For example:

```
myImage.addEventListener('click', showEvt, false);
```

This code would attach an event listener to the element identified as `myImage`.

If the mouse-button is clicked whilst the mouse-pointer is over the image, the function `showEvt()` will be called.

Note that, when using `addEventListener()` to trigger a function, the function name is used WITHOUT parameter brackets.

When a function is called using `addEventListener()`, a *reference* to the corresponding event object is automatically passed to the function as a *parameter*.

You can access the event object by giving the parameter a name, and then using it in the body of the function. For example:

```
function showEvt(evt) {

  alert('Cursor x = ' + evt.clientX + ', y = ' + evt.clientY);
}
```

Once registered, event listeners can be removed using the method `removeEventListener()`. For example:

```
myImage.removeEventListener('click', showEvt, false);
```

Drag-and-Drop

Consider the code needed to allow an element to be dragged with the mouse.

The basic sequence of actions is as follows:

- Wait until the mouse button is depressed, (`mousedown`) whilst over the element, then...
- ...each time the mouse is moved (`mousemove`):
  - obtain the coordinates of its new position
  - use these coordinates to set the position of the element.

  This will cause the element to follow the mouse. Keep doing this until...
- ...the mouse button is released (`mouseup`).

However, if the mouse is moved too quickly or in certain directions, the cursor may move off the element.

If this happens:

- Mouse-movements won't be received by the element, so it will stop moving.
- If the mouse button is released the mouse-up event will not be received by the element, so releasing the mouse-button won't stop the dragging.

To avoid this, mouse-move and mouse-up events should be detected on (e.g.) the *document* rather than on the element that is to be dragged.

A practical sequence of actions is as follows:

1. Attach a `mousedown` event listener to the *element*
2. When a `mousedown` event is detected by the element, attach `mousemove` and `mouseup` event listeners to the *document*;
3. When a mouse-movement is detected, the (document) event listener should obtain the mouse coordinates and use them to set the position of the element.
4. When a mouse-up event is detected, the (document) event listener should:
    - Cancel the `mousemove` event listener so that the element no longer follows the mouse.
    - Cancel the `mouseup` event listener so that further `mouseup` events will not have any effect.

First, create a web-page that includes a suitable element, e.g., a `<div>`:

```
<!DOCTYPE html>
<html>
<head>
<script type ='text/javascript'>

</script>
</head>
<body>
<div id='div0'
     style='position:absolute; left:50px; top:50px;
            width:50px; height:50px; background-color:red'>
</div>
</body>
</html>
```

Note that, when declaring an element which is to be moved:

- its `position` attribute should be set to `absolute`
    - an element cannot be moved if its `position` attribute is set to `static` (which is the default)
- its initial position should be set using (e.g.) the `left` and `top` CSS properties
    - some browsers will not recognise new position settings if no initial position has been set.

Next, create the JavaScript functions which will allow the `<div>` to be dragged.

The first function should attach a `mousedown` event listener to the `<div>`:

```
var myDiv = null;

window.addEventListener('DOMContentLoaded', setupEvents, false);

function setupEvents() {

  myDiv = document.getElementById('div0');
  myDiv.addEventListener('mousedown', startDrag, false);
}
```

Note that:

- A *global variable*, `myDiv` is declared to hold a reference to the `<div>`.
- The `setupEvents()` function is called when the page is loaded.
    - This is achieved by attaching an event listener to the `window` object which calls the function when the `DOMContentLoaded` event occurs.

Now you need to create `startDrag()` and the other functions.

```
function startDrag() {

  document.addEventListener('mousemove', dragDiv, false);
  document.addEventListener('mouseup', stopDrag, false);
  return false;
}

function dragDiv(evt) {

  myDiv.style.left = parseInt(evt.clientX) + 'px';
  myDiv.style.top = parseInt(evt.clientY) + 'px';
  return false;
}

function stopDrag() {

  document.removeEventListener('mousemove', dragDiv, false);
  document.removeEventListener('mouseup', stopDrag, false);
}
```

This code works as follows:

- `startDrag()` does two things:
    - It attaches a `mousemove` event listener to the *document*.
      Any subsequent movement of the mouse over the document will cause the function `dragDiv()` to be executed.
    - It attaches a `mouseup` event listener to the *document*.
      Any subsequent release of the mouse-button over the document will cause the function `stopDrag()` to be executed.

  The function then returns `false`, ensuring that no further action is taken in response to this event.

- If the mouse is now moved, `dragDiv()` will be executed. This function:
    - Receives an event object as a parameter (`evt`).
    - Obtains the *x* and *y* coordinates of the cursor from the event object (`clientX` and `clientY`).
    - Uses these coordinates to set the `left` and `top` style attributes of the `<div>`, thus moving it to the new cursor position.
    - Returns `false`, ensuring that no further action is taken in response to this event.
- If the mouse-button is released, `stopDrag()` will be executed.
    - This function cancels both the `mousemove` and `mouseup` event listeners.
    - Subsequent mouse-movements and mouse-up events will be ignored.