JavaScript and AJAX
2nd Edition

Tutorial 6

Working with the Event Model

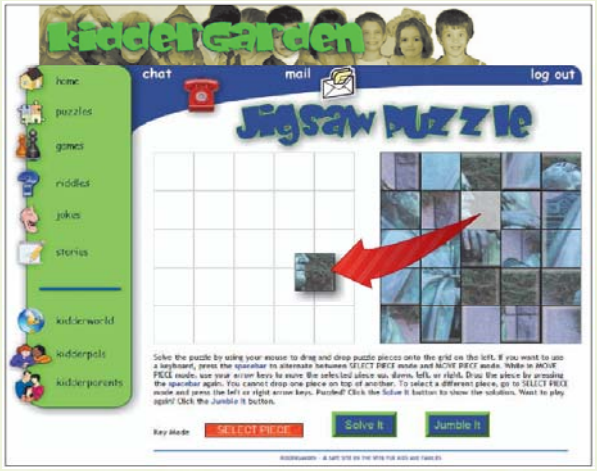
Objectives

- Compare the IE and W3C event models
- Study how events propagate under both event models
- Write a cross-browser function to capture and remove event handlers
- Study the properties of the event object
- Reference the event object under both event models

Objectives

- Retrieve information about the mouse pointer
- Work with the cursor style
- Capture keyboard events
- Halt the propagation of events under both event models
- Prevent the default action associated with an event

Introducing the Event Model



Introducing the Event Model

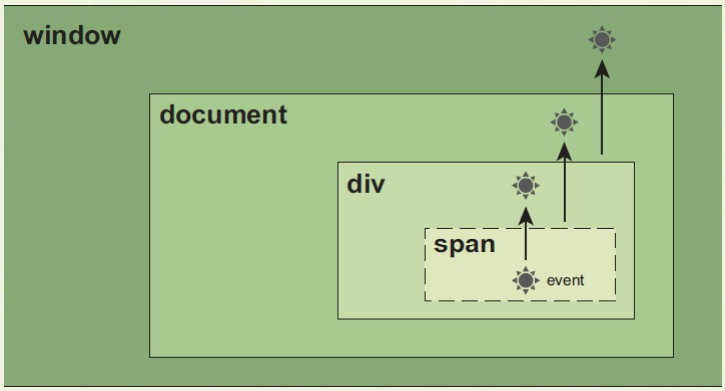
- An **anonymous function** is a function without a name
  - Needs to be run only once
- The **event model** describes how events interact with objects
  - **IE event model**
    - Supported by IE and Opera
  - **W3C event model**
    - Supported by other major browsers

Introducing the Event Model

- In Internet Explorer, **event bubbling** is when an event is initiated at the bottom of the object tree and rises to the top of the hierarchy
- In **event capturing**, events are initiated at the top of the object hierarchy and drop down the object tree to the lowest object
  - Not supported in the IE event model

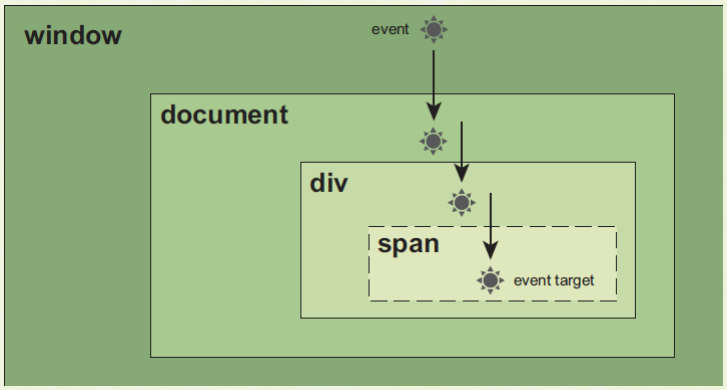
Introducing the Event Model

Event bubbling in the IE event model



Introducing the Event Model

Event capturing



Introducing the Event Model

- In the W3C event model, an event is split into three phases
  - A **capture phase** as the event moves down the object hierarchy
  - A **target phase** in which the event reaches the object from which the event originated
  - A **bubbling phase** in which the event moves back up the object hierarchy
- To run a function, you create an **event listener** that detects when a particular event has reached an object in the document

```
object.addEventListener(event, function, capture)
```


Introducing the Event Model

- Both event models allow you to remove event handlers from objects
  - The IE event model uses the detachEvent method

    ```
    object.detachEvent (onevent, function)
    ```
  - The W3C event model uses the removeEventListener method

    ```
    object.removeEventListener (event, function, capture)
    ```

Introducing the Event Model

- **IE Event Model**
  - To attach a function to an object, run:
    ```
    object.attachEvent(onevent, function);
    ```
    where object is the object receiving the event, onevent is the text string of the event handler, and function is the function that runs in response to the event. Multiple functions can be attached to the same event in the same object.
  - To detach a function, run the following:
    ```
    object.detachEvent(onevent, function);
    ```


Introducing the Event Model

- **W3C Event Model**
- To run a function when an event reaches an object, use
  ```
  object.addEventListener(event, function, capture);
  ```
  where object is the object receiving the event, event is the text string describing the event, function is the function to run in response to the event, and capture equals true if the event is moving down the document tree and false if the event is bubbling up the tree.
- To stop listening for an event, run the following:
  ```
  object.removeEventListener(event, function, capture);
  ```

Introducing the Event Object

- If the user has pressed a key on the keyboard, you may want to know which key was pressed
- This type of information is stored in an **event object**

Introducing the Event Object

- **The Internet Explorer Event Object**
  - In the Internet Explorer event model, the event object has the object reference:

    `window.event`
  - If you are dealing with events in the current browser window, you can drop the `window` reference
  - One of the more important properties is srcElement
  - The srcElement property is akin to the “this” keyword

Introducing the Event Object

Property	Description
event.button	Returns the number indicating which mouse button the user pressed (1 = left, 2 = right, 4 = middle)
event.cancelBubble	Set this property to true to cancel event bubbling; set it to false to continue event bubbling
event.fromElement	For mouseover and mouseout events, returns the object from which the pointer is moving
event.returnValue	Set this property to false to cancel the default action of the event; set it to true to retain the default action
event.srcElement	Returns the object in which the event was generated
event.toElement	For mouseover and mouseout events, returns the object to which the pointer is moving
event.type	Returns a text string indicating the type of event

Introducing the Event Object

- **The W3C Event Object**
  - In the W3C event model, the event object is inserted as a parameter of whatever function responds to the event
  - Give the event object any parameter name, but the standard practice is to name the parameter “e” or “evt”
  - For the DOM event model, the object that initiated an event is returned using the target property

Introducing the Event Object

Property	Description
evt.bubbles	Returns a Boolean value indicating whether *evt* can bubble
evt.button	Returns the number of the mouse button pressed by the user (0 = left, 1 = middle, 2 = right)
evt.cancelable	Returns a Boolean value indicating whether *evt* can have its default action canceled
evt.currentTarget	Returns the object that is currently handling the event
evt.eventPhase	Returns the phase in the propagation of *evt* (1 = capture, 2 = target, 3 = bubbling)
evt.relatedTarget	For mouseover events, returns the object that the mouse left when it moved over the target of the event; for mouseout events, returns the object that the mouse entered when leaving the target
evt.target	Returns the object that initiated the event
evt.timeStamp	Returns the date and time that the event was initiated
evt.type	Returns a text string indicating the event type

Introducing the Event Object

- Determining the Event Source

```
var grids = new Array();
var pieces = new Array();
var mousePiece = null;

function mouseGrab(e) {
   var evt = e || window.event;
   mousePiece = evt.target || evt.srcElement;

   alert("Event: " + evt.type + " on " + mousePiece.id);
}
```

Working with Mouse Events

Property	Returns	Event Model
evt.clientX *evt*.clientY	Returns the x and y coordinates of the event, *evt*, within the browser window	IE, W3C
evt.screenX *evt*.screenY	Returns the x and y coordinates of the event within the computer screen	IE, W3C
evt.offsetX *evt*.offsetY	Returns the x and y distances of the event from the object in which the event was initiated	IE
evt.x *evt*.y	Returns the x and y coordinates of the event relative to the element that initiated the event	IE
evt.pageX *evt*.pageY	Returns the x and y coordinates of the event within the document	W3C
evt.layerX *evt*.layerY	Returns the x and y coordinates of an event relative to its absolutely positioned parent element	W3C

Working with Mouse Events

- Keeping Dragged Items on Top

```
var diffX = null;
var diffY = null;
var maxZ = 1;

function mouseGrab(e) {
   var evt = e || window.event;
   mousePiece = evt.target || evt.srcElement;

   maxZ++;
   mousePiece.style.zIndex = maxZ; // place the piece above other objects

   var mouseX = evt.clientX; // x-coordinate of pointer
   var mouseY = evt.clientY; // y-coordinate of pointer
```

Formatting a Drag-and-Drop Action

- Mouse pointers can be defined using an object's style properties

```
object.style.cursor=cursorType;
```

- Can also define the pointer style in a CSS style declaration

```
cursor: cursorType
```


Formatting a Drag-and-Drop Action

Cursor	Stylo	Cursor	Stylo
+	crosshair	↕	n-resize
↖	default	↗	ne-resize
↖?	help	↔	e-resize
✥	move	↘	se-resize
☝	pointer	↕	s-resize
I	text	↙	sw-resize
◌	wait	↔	w-resize
	url(*uri*) where *url* is the URL of a file containing the cursor image	↖	nw-resize

Working with Keyboard Events

- Capturing a Keyboard Event
  - Three main keyboard events are available to work with
    - **keydown**: The user presses a key down
    - **keypress**: Follows immediately after the onkeydown event
    - **keyup**: The user releases a key

Working with Keyboard Events

- To run a command when the user presses down a key, use the onkeydown event handler
- To run a command when the user releases a key, use the onkeyup event handler
- To run a command when the user enters a character from the keyboard, use the onkeypress event handler
- To retrieve the code of the key pressed by the user during the keydown or keyup event, use the property:
  - **`evt.keyCode`**

  where *evt* is the event object.

Working with Keyboard Events

Property	Description	Event Model
evt.altKey	Returns a Boolean value indicating whether the Alt key was pressed during the event, where *evt* is the event object	IE, W3C
evt.ctrlKey	Returns a Boolean value indicating whether the Ctrl key was pressed	IE, W3C
evt.shiftKey	Returns a Boolean value indicating whether the Shift key was pressed	IE, W3C
evt.metaKey	Returns a Boolean value indicating whether any meta key was pressed	W3C
evt.keyCode	Returns a key code indicating which key was pressed during the keyup and keydown events	IE, W3C
evt.charCode	Returns the ASCII character code indicating which character was produced during the keypress event	W3C
evt.which	Returns the ASCII character code indicating which key was pressed during the keydown, keypress, and keyup events	W3C

Working with Keyboard Events

- Key code values

Key(s)	Key Code(s)	Key(s)	Key Code(s)
[0 – 9]	48 – 57	page up	33
[a – z]	65 – 90	page down	34
backspace	8	end	35
tab	9	home	36
enter	13	left arrow	37
shift	16	up arrow	38
ctrl	17	right arrow	39
alt	18	down arrow	40
pause/break	19	insert	45
caps lock	20	delete	46
esc	27	[f1 – f12]	112 – 123
space	32	num lock	144

Working with Keyboard Events

- Modifier Keys
  - Both event models use the following properties of the event object to determine the state of the Alt, Ctrl, and Shift keys

    ```
    evt.altKey;
    evt.ctrlKey;
    evt.shiftKey;
    ```

  - Each of these properties returns a Boolean value indicating whether the modifier key is being pressed
  - The W3C event model also supports the event object property

    ```
    evt.metaKey;
    ```


Controlling and Canceling Events

- JavaScript supports several methods for controlling and canceling events occurring within a browser
  - **`event.cancelBubble = value;`**
  - **`evt.stopPropagation()`**

```
function keyGrab(e) {
   var evt = e || window.event;

   if (evt.keyCode == 32) {toggleMode(); return false}
   else if (selectMode && evt.keyCode == 37) {selectPiece(-1); return false}
   else if (selectMode && evt.keyCode == 39) {selectPiece(1); return false}
   else if (!selectMode && evt.keyCode == 37) {keyMove(-8, 0); return false}
   else if (!selectMode && evt.keyCode == 38) {keyMove(0, -8); return false}
   else if (!selectMode && evt.keyCode == 39) {keyMove(8, 0); return false}
   else if (!selectMode && evt.keyCode == 40) {keyMove(0, 8); return false}

}
```

cancels the default action associated with the spacebar