Session: 14

HTML5 Geolocation and APIs

Objectives

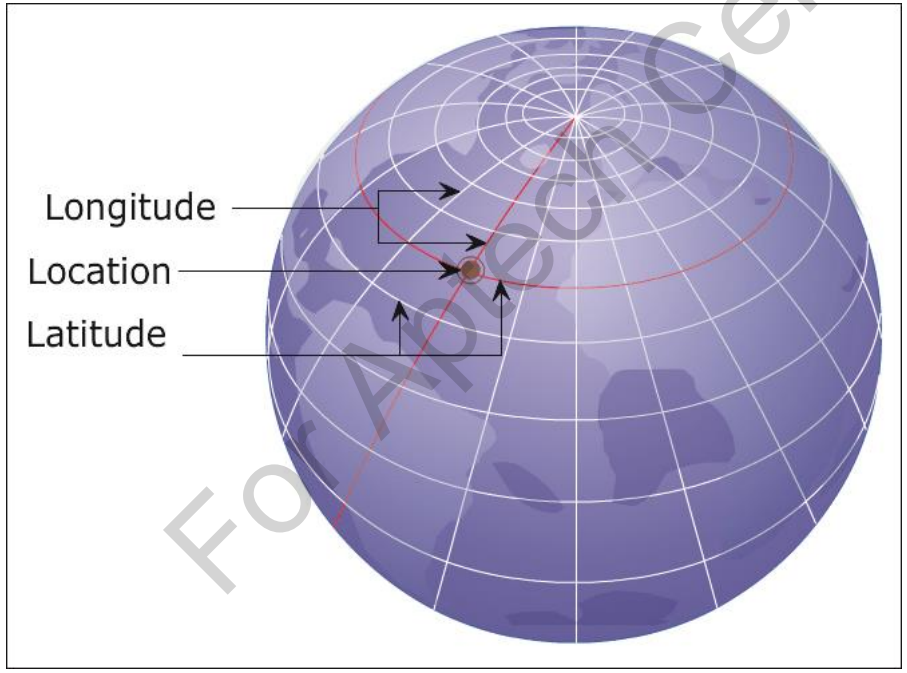
- Explain geolocation and its use in HTML5
- Explain the Google Maps API
- Explain the drag-and-drop operations in HTML5
- Explain the concept of Application Cache

Geolocation 1-2

Geolocation in computing terminology determines the current location of a user on the devices.

The location of the user is represented as a single point that comprises two components: latitude and longitude.

- Following figure shows the representation of latitude and longitude with respect to a location on the globe:



Geolocation 2-2

- Different sources through which devices can determine the information about the location are as follows:

Global Positioning System (GPS)

- GPS is a satellite navigation system that provides information about the location on any part of the globe.
- The GPS system is maintained by the government of the United States.

IP Address

- Location information can be derived from IP Address which is assigned to devices, such as desktops, printers, and so on connected on a network.

GSM/CDMA Cell IDs

- These are used by the cell phones.

WiFi and Bluetooth MAC address

- These are used by devices that have wireless network connection.

User Input

- It is a software tool which can be used on any device requesting for location information.
- The information retrieved by the tool is based on the data provided by the user. For example, a zip code.

Geolocation API 1-2

In HTML5, the Geolocation API is a specification by W3C for providing a consistent way to develop location-aware Web applications.

The Geolocation API provides a high-level interface to retrieve location information related to the hosting devices.

The interface hides the details, such as how the information is gathered or which methods were used to retrieve the information.

The object that holds implementation of the Geolocation API is the Geolocation object.

- This object is used in JavaScript to retrieve the geographic information about the devices programmatically.
- The browser processes the script and returns the location to the Geolocation API.
- The Geolocation API is supported on most of the modern browsers available on desktop and mobile phones.

Geolocation API 2-2

- Following table lists the browsers providing support for Geolocation API:

Browser	Version Support
Safari	5.0+
Chrome	5.0+
Firefox	3.5+
Internet Explorer	9.0+
Opera	10.6+
iOS (Mobile Safari)	3.2+
Android	2.0+

Implementing Geolocation Object 1-3

The `Geolocation` object is available as a new property of the `navigator` object.

The `navigator` object is a browser object that allows a user to retrieve information about the specific location.

- Following syntax shows how to create a `Geolocation` object in JavaScript:

Syntax:

```
var geolocation = window.navigator.geolocation;
```

where,

- `window`: Is the top level object in JavaScript object hierarchy.

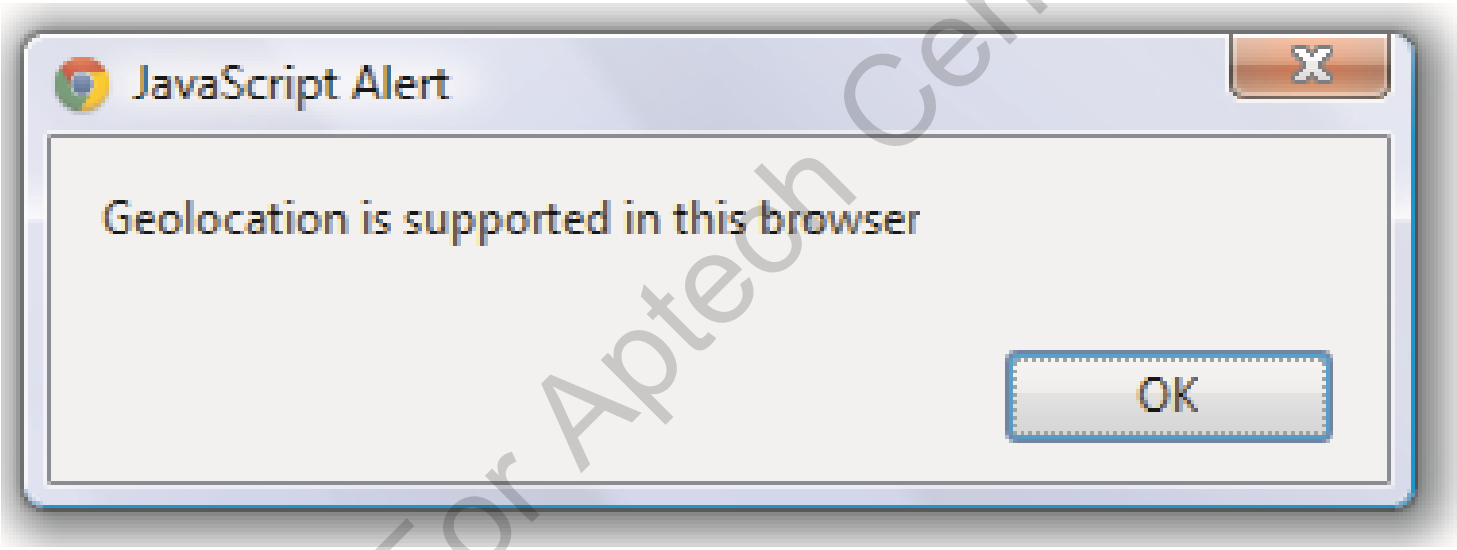
Implementing Geolocation Object 2-3

- **The Code Snippet demonstrates the script that tests the existence of `geolocation` object within a browser.**

```
<!DOCTYPE html>
<html>
  <head>
    <title> Testing Support for Geolocation in Browsers</title>
    <script>
     function display_location_enabled()
     {
      // Default message
      var str = "Geolocation is not supported in this browser";
      if (window.navigator.geolocation)
      {
          str = "Geolocation is supported in this browser";
      }
      alert (str);
     }
    </script>
  </head>
  <body>
       <input type="button" value="Geolocation Support"
         onClick="display_location_enabled()"></input>
  </body></html>
```

Implementing Geolocation Object 3-3

- In the code, the 'if' statement checks existence of the `geolocation` property in the browser.
- If browser provides implementation for the property, then an alert window displays the message '`Geolocation is supported in this browser`'.
- Otherwise, the default message is displayed.
- Following figure shows the existence of `geolocation` object in the Chrome browser:



Geolocation Methods

- The `geolocation` object provides three methods that can be used to determine the current position of the user.
- Following table lists the methods of the `geolocation` object:

Method	Description
`getCurrentPosition()`	Retrieves the current geographic location information of the user
`watchPosition()`	Retrieves the geographic information of the device at regular intervals
`clearWatch()`	Terminates the current watch process

Retrieve User Information 1-5

The current position of a user is retrieved using the method `getCurrentPosition(successCallback, errorCallback, options)`

This function accepts three parameters, out of which two are optional, `errorCallback` and `options`.

The first parameter, `successCallback` is the name of the function which is invoked after the position of a device is found successfully.

The second parameter, `errorCallback` is the name of the function which will be called, if an error occurs in retrieving the position.

The last parameter, options represents a `PositionOptions` object.

Retrieve User Information 2-5

- The Code Snippet demonstrates the script that will retrieve the current location of the user.

```
<!DOCTYPE html>
<html >
 <head>
  <title>Geolocation API</title>
  <script>
   function getLocation()
   {
       if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
       }
       else{
         alert ("Geolocation is not supported in this browser.");
       }
   }
   function showPosition(position)
   {
    alert('Latitude: ' + position.coords.latitude + '\n' +
          'Longitude: ' +  position.coords.longitude);
   }
   </script>
 </head>
```

Retrieve User Information 3-5

```
<body>
  <input type="button" value=" Display Location"
   onClick="getLocation()"/>
</body>
</html>
```

- In the code, the `getCurrentPostion()` function obtains the position which is passed as a parameter to the `showPosition()` function.
- The `showPosition()` function obtains the coordinates of a location through `position` object.
- The `position` object is defined in the Geolocation API and holds the current location of the device.
- It contains attribute named `coords` that retrieves the latitude and longitude of the location.
- The values retrieved for latitude and longitude are in decimal degrees.

Retrieve User Information 4-5

- Following table lists the attributes of `position` object:

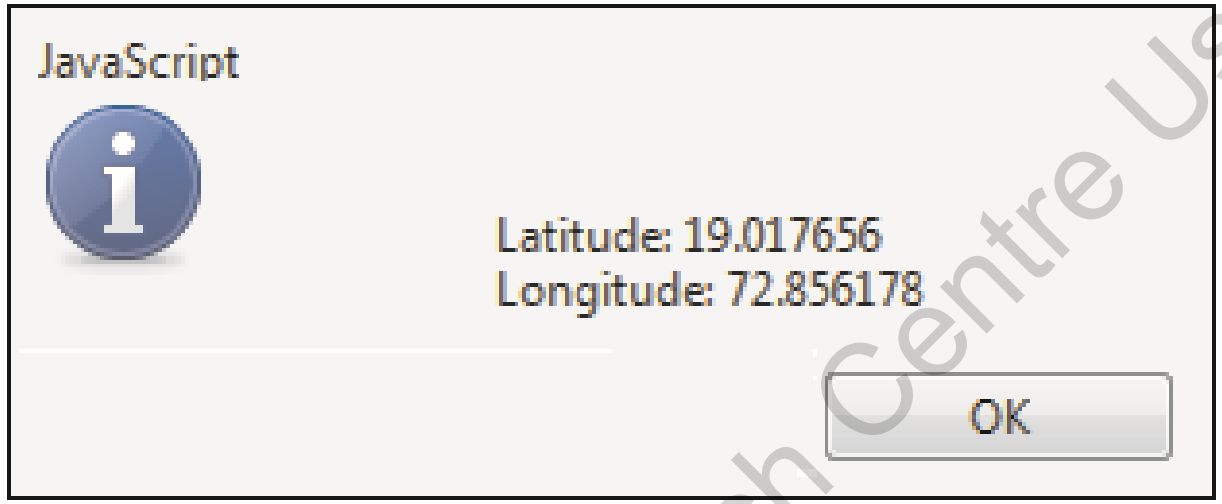
Attribute	Description
`coords`	An object of type Coordinates that provides different properties, such as latitude, longitude, altitude, accuracy, speed, and so on.
`timestamp`	An object of type DOMTimeStamp.

- Following figure shows the notifications for the Web page containing geolocation code:

- The browser seeks permission from the user to share their location information with the application.

Retrieve User Information 5-5

- Following figure shows a message displaying current location of the user, when the **Share My Location** button is clicked:



Handling Errors 1-4

- An application could fail in gathering geographic location information. In that case, the `geolocation` object calls an `errorCallback()` function.
- The `errorCallback()` function handles errors by obtaining a `PositionError` object from the API.

HTML

- The `PositionError` object holds information related to errors occurred while finding the geographic location of the user.
- Following table lists the properties of `PositionError` object:

Property	Description
`code`	Returns a numeric value for the type of error occurred.
`message`	Returns a detailed message describing the error encountered. The message can be used for debugging.

Handling Errors 2-4

- Following table lists different error codes returned by the `code` property of the `PositionError` object:

Code	Constant	Description
1	PERMISSION_DENIED	Application does not have permission to access Geolocation API.
2	POSITION_UNAVAILABLE	Position of the device could not be obtained.
3	TIMEOUT	Unable to retrieve location information within the specified interval.

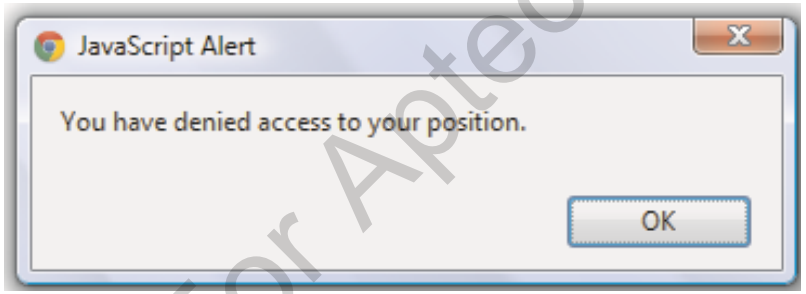
Handling Errors 3-4

The Code Snippet demonstrates the error handling routine for the geolocation code.

```
<!DOCTYPE html>
<html>
<head>
    <title>Handling Error</title>
<script>
   function getLocation()   {
     function showPosition(position)
     {
       alert('Latitude: ' + position.coords.latitude + '\n' +
       'Longitude: ' + position.coords.longitude);
   }
   function errorHandler(error) {
    switch (error.code) {
     case error.PERMISSION_DENIED:
     alert ('You have denied access to your position.');
     break;
     case error.POSITION_UNAVAILABLE:
       alert ('There was a problem getting your position.');
     break;
     case error.TIMEOUT:
       alert ('The application has timed out attempting to
              get your position.');
     break;
  }
}
</script> </head>
<body>
    <input type="button" value="Display Location"
          onClick="getLocation()"/>
</body>
</html>
```

Handling Errors 4-4

- In the code, if application fails to find the current location of the user, then the `errorHandler()` function is invoked.
- The function is passed as the second parameter in the `getCurrentPostion()` method and is used to handle the errors occurred in the application.
- It obtains the `error` object which is of type `PositionError` from the API and compares it with the error codes specified in the `switch-case` statement.
- Depending on the error occurred, the appropriate `case` statement is executed and an alert message is displayed to the user.
- Following figure shows the output displaying error message for geolocation application:



- The reason for the error is that the Chrome browser blocks the URL whose file path starts with file:///.

PositionOptions Object 1-3

- `PositionOptions` **object is an optional third parameter passed to the** `getCurrentPosition()` **method.**
- **This object defines properties that are optional and are used by an application while retrieving the geolocation information.**
- **Following table lists the attributes of** `PositionOptions` **object:**

Attribute	Description
`enableHighAccuracy`	Indicates that the application wants to receive the most accurate results for geolocation. The default value of the attribute is false.
`maximumAge`	Obtains the cached position object whose age is less than the specified maximumAge limit (in milliseconds). If age limit is set to 0, then the application must obtain a new position object.
`timeout`	Indicates the maximum time length (in milliseconds) for which the application can wait to obtain the position object.

PositionOptions Object 2-3

- **The Code Snippet demonstrates the script to set attributes of `PositionOptions` object.**

```
<script>
   var options = {
                    enableHighAccuracy: true,
                    maximumAge: 50000,
                    timeout: 60000
                  };
    function getLocation() {
      if (navigator.geolocation)   {
         navigator.geolocation.getCurrentPosition(showPosition,
         errorHandler, options);
      }
      else{
        alert ("Geolocation is not supported in this browser.");
      }
}
. . .
</script>
```

PositionOptions Object 3-3

- In the code, an object named options is set with attributes.
- The attribute, `maximumAge` enables the application to use a cached `position` object which is not older than 50 seconds.
- Also, the `timeout` limit is set to 60 seconds for an application, before notifying an error.
- The `options` set of values is passed as third parameter to the `getCurrentPosition()` method.

Google Maps API 1-5

Google Maps API is used to display locations on a map based on the values of their coordinates, latitude and longitude.

The Google Maps API must be configured in JavaScript, before it can be referenced further on the page.

It contains a `Map` object which is instantiated and displayed on a Web page.

- Following syntax shows the configuration of Google Maps API in JavaScript:

Syntax:

```
<script src="http://maps.google.com/maps/api/js?sensor=false">
</script>
```

where,

- `src`: Is the URL of Google Maps API.
- `sensor`: Parameter sent with the URL. It indicates whether application uses any sensor such as GPS system.

Google Maps API 2-5

- The Code Snippet demonstrates how to load and initialize the Google Maps API in the `<script>` tag.
- The code will execute after the page is loaded completely and will invoke a function in response to the `onload` event.

```
<!DOCTYPE html>
<html>
  <head>
  <title> Load and Initialize Google Maps </title>
    <style>
        html { height: 100% }
      body { height: 100%; width: 100%; margin: 10% }
      #map_canvas { height: 50%; width: 50% }
    </style>
<script
  src="http://maps.google.com/maps/api/js?sensor=false">
</script>
```

Google Maps API 3-5

```
function initialize()
 {
   // Loading Google Maps
    var num = new google.maps.LatLng(51.528663,-0.173171);
    var myOptions = {
                zoom: 16,
                center: num,
                 mapTypeId: google.maps.MapTypeId.HYBRID
        };
    var mymap = new google.maps.Map(document.getElementById("
                map_canvas"), myOptions);
    var marker = new google.maps.Marker({
                  position: num,
                  map: mymap,
                  title:"Lord's Cricket Ground, London!"
                });
  }
</script>
</head>
    <body onload="initialize()">
       <div id="map_canvas"></div>
    </body>
</html>
```

Google Maps API 4-5

- Following table lists some of the `myOptions` properties:

Property	Description
`zoom`	Sets the initial resolution at which map is displayed. A lower zoom value 0 represents a full map of the Earth. Similarly, a higher zoom value displays a map with high resolution.
`center`	Centers the map on a specific point by creating an object of type `LatLng` which holds the location coordinates.
`mapTypeId`	Sets an initial map type. The map types supported are: `ROADMAP` for normal, `SATELLITE` for photographic tiles, `HYBRID` for roads and city names, and `TERRAIN` for water features.

Google Maps API 5-5

- Following figure displays the object on the Web page that is centered on Lord's Cricket Ground in London:

Tracking User's Location 1-3

- The `Geolocation` object is used by the Google Maps API to display the geolocation information in the applications.
- The Code Snippet demonstrates the code that displays current location of a user on the map using `Geolocation` object.

```
<!DOCTYPE html>
<html lang="en">
  <head>
  <style>
        html, body {
        width: 100%;
        height: 100%;
        padding: 10%
    }
    #map_canvas {
        height: 50%;
        width: 50%;
    }
  </style>
<script src="http://maps.google.com/maps/api/js?sensor=false">
</script>
<script>
    // Check support for Geolocation in the browser
     if (navigator.geolocation) {
      // Locate position and invoke function
       navigator.geolocation.getCurrentPosition(displayPosition, errorFunction);
    }
    else {
       alert('Geolocation is not enabled in your browser');
    }
```

Tracking User's Location 2-3

```
// Creates the Map object
  var map = new
google.maps.Map(document.getElementById("
            map_canvas"), myOptions);
// Displays icon on the located position
  var marker = new google.maps.Marker({
              position: latlng,

              map: map,
            title:"User location"
            });
  }
// Error callback function
function errorFunction(pos) {
     alert('Error!');
   }
 </script> </head>
 <body>
     <div id="map_canvas"></div>
     <div id="user_location"></div>
 </body> </html>
```

```
// Success function
function displayPosition(position) {
   var my_lat = position.coords.latitude;
   var my_lng = position.coords.longitude;
   var div_info =
document.getElementById('user_location');
   div_info.innerHTML = '<h1> Latitude is :' +
my_lat + ' and Longitude is  + my_lng + '</h1>';
  // Load Google Maps
   var latlng = new google.maps.LatLng(my_lat,
    my_lng);
   var myOptions = {
      zoom: 2, //the initial resolution is set at
      // which map is displayed
    center: latlng,  //centers the map
    mapTypeId: google.maps.MapTypeId.ROADMAP //sets
      // the map type
   };
```

Tracking User's Location 3-3

- The code uses the `getCurrentPosition()` method and retrieves the current position of the user.
- Then, it passes the information to `displayPosition()` function, which retrieves the coordinates, latitude and longitude.
- The retrieved coordinates are set into the properties of the `Options` object named myOptions and initialize the `Map` object.
- Finally, the `Map` object is displayed along with the current position information in the `<div>` element.
- Following figure shows the output displaying the current location of the user on the Google Maps:

Latitude is :19.017656 and Longitude is 72.856178

Drag and Drop

HTML5 defines drag-and-drop operations that are based on events. Currently, drag-and-drop operations are supported by all major browsers.

The event-based mechanism allow the elements to be copied, reordered, or deleted on a Web page.

The drag-and-drop operation involves the use of a pointing device, such as mouse on a visual medium.

To perform the drag operation, a mousedown event is triggered followed by multiple mousemove events.

Similarly, the drop operation is performed when a user releases the mouse.

The benefit of drag-and-drop mechanism is that it has brought the drag-and-drop operations on the browser level.

This makes programming easier, thus eliminating the need of complex JavaScript code written in earlier HTML versions.

Drag Operation

- The steps required to make any element draggable on a Web page are as follows:

1. Set the `draggable` attribute of an element to be dragged.
2. Set an `ondragstart` event on the element which stores the data being dragged.
3. Store the data into the `DataTransfer` object.

- The Code Snippet shows how to set the `draggable` attribute of an image element.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Drag and Drop API</title>
  </head>
  <body>
    <div id="div" style="border: red 2px solid; height:125px;
         width:75px; padding: 10px">
     <img src="image.jpg" height="75" width="75" id="image1"
         draggable="true"/>
    </div>
  </body> </html>
```

Drag Events

- During various stages of the drag-and-drop operation, a number of events are fired.
- These events are mouse-based events.
- Following table lists various events triggered during the drag operation:

Event	Description
`dragstart`	Triggers when an element is started to be dragged by the user.
`drag`	Triggers when an element is being dragged using a mouse.
`dragleave`	Triggers when the drag and drop operation is completed.

DataTransfer Object 1-3

- The `dataTransfer` object reveals the **drag data store** that contains the dragged data in the drag-and-drop operation.
- It allows getting and setting of the data being dragged.
- In other words, the `dataTransfer` object holds the data during drag-and-drop operation.
- The `dataTransfer` Object enables to define two types of information.
- These are as follows:
  - The data type of the draggable element
  - The value of the data being stored in the data store

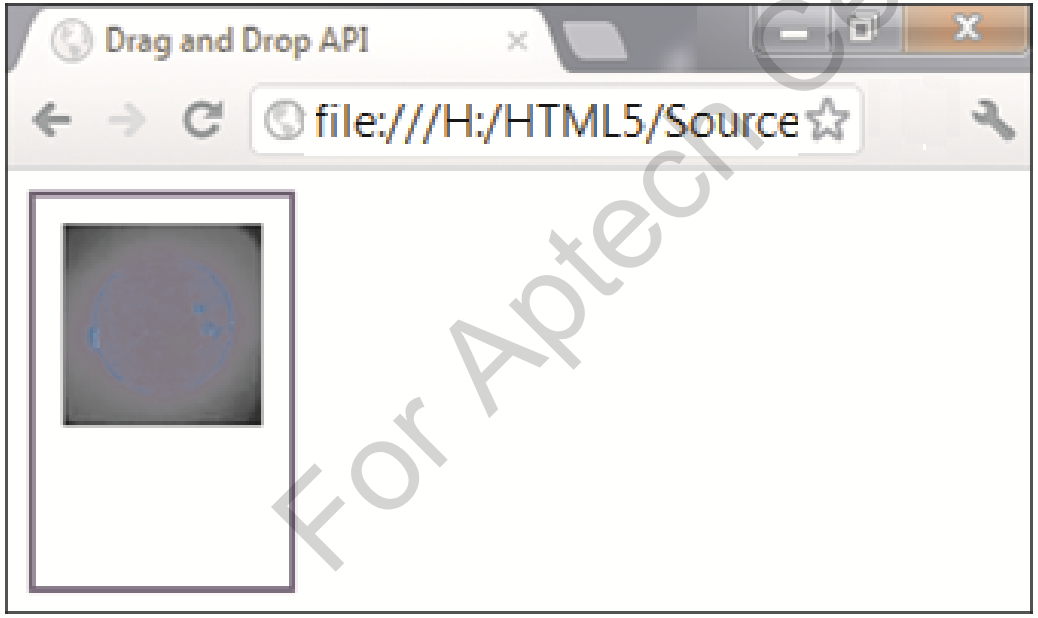
DataTransfer Object 2-3

- **The Code Snippet demonstrates how to associate an element with `dragstart` event to store the data being dragged.**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Drag and Drop API</title>
    <script>
        function drag_image(event)
        {
          event.dataTransfer.setData("image", event.target.id);
        }
   </script>  </head>
<body>
  <div id="div1" style="border: blue 2px solid; height:125px;
       width:75px; padding: 10px">
   <img src="image.jpg" height="75" width="75" id="image1"
       draggable="true" ondragstart="drag_image(event)"/>
  </div>
</body>
</html>
```

DataTransfer Object 3-3

- In the code, the `<img>` element has been set with an event listener for the `dragstart` event.
- When the image is dragged, then the `dragstart` event is fired and calls `drag_image()` function.
- The function uses the `dataTransfer` object to store the data during drag-and-drop operation.
- The string '`image`' represents the data type and `event.target.id` represents the value of `id` attribute of the draggable element.
- Following figure shows the output of the image element to be dragged:



Drop Operation

After the element has been set up for dragging, it can be dropped in some element on the Web page.

By default, elements on the page are not set up to receive dragged elements.

Thus, the behavior of element acting as a drop element must be changed.

This can be done by creating event listeners for the drop element.

The drop element is also referred to as target element.

Drop Events 1-4

- For any element to receive the drop operation, it must be associated with the drop events.
- Following table lists the events of the drop operation:

Event	Description
`dragenter`	Triggers when a draggable element is being dragged on the target element for the first time.
`dragleave`	Triggers when an element is dragged outside the target element.
`dragover`	Triggers when an element is dragged inside the target element.
`drop`	Triggers when an element is dropped in the target element.

Drop Events 2-4

The Code Snippet demonstrates how to set up event listeners to drop the image element on the target element.

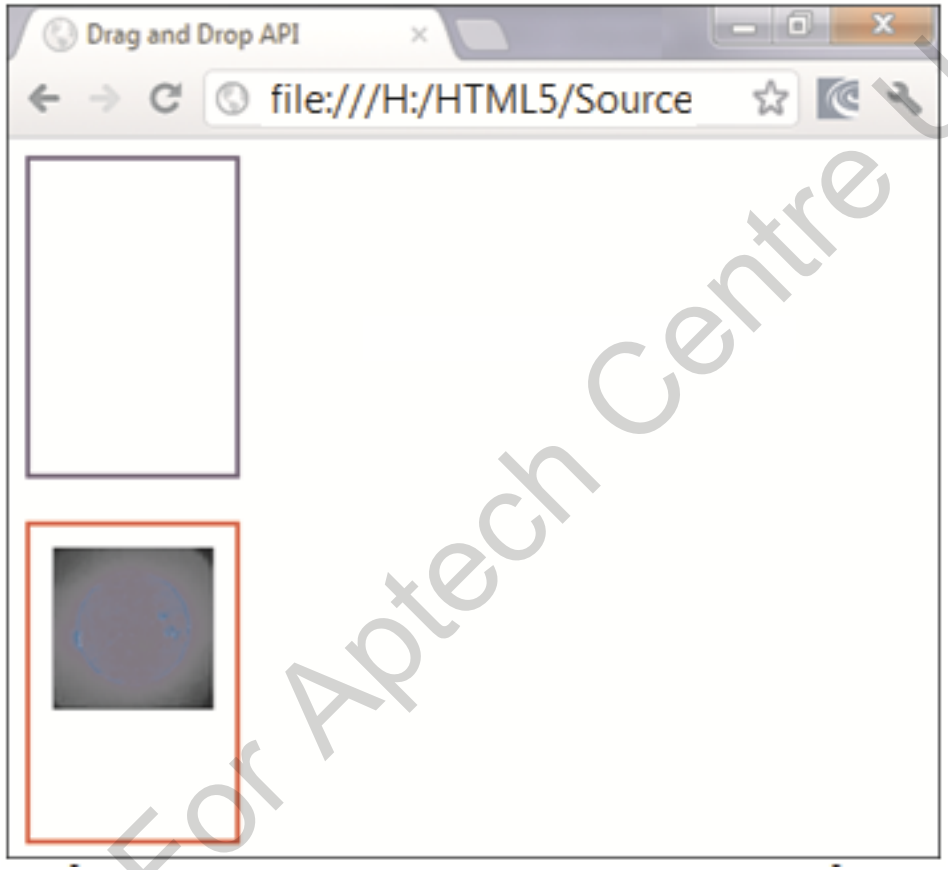
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Drag and Drop API</title>
  <script>
   function drag_image(event)    {
     event.dataTransfer.setData("image", event.target.id);
   }
   function allow_drop(event)
   {
     event.preventDefault();
   }
   function drop_image(event)    {
      var data=event.dataTransfer.getData("image");
    event.target.appendChild(document.getElementById(data));
   }
   </script> </head>
<body>
  <div id="div1" style="border: blue 2px solid; height:125px;
      width:75px; padding: 10px">
   <img src="image.jpg" height="75" width="75" id="image1"
       draggable="true" ondragstart="drag_image(event)"/>
  </div>
  <br/>
  <div id="div2" style="border: red 2px solid; height:125px;
       width:75px; padding: 10px" ondrop="drop_image(event)"
      ondragover="allow_drop(event)">
  </div>
  </body>
</html>
```

Drop Events 3-4

- In the code, the `<div>` element with id attribute, set as '`div2`', is associated with two event listeners namely, `ondragover` and `ondrop`.
- The `ondropover` calls the `allow_drop()` function which prevents the default behavior of the target element.
- By default, browsers do not support dropping of one elements on the other element.
- To prevent the default behavior, the statement, `event.preventDefault()` is invoked.
- Then, the `drop` event is fired on the target element.
- It calls the function `drop_image()` which uses `getData()` method to retrieve image that is set as '`image`'.
- Finally, it appends the dragged image as a element into the target element, `div2`.

Drop Events 4-4

- Following figure shows the output of the drop operation, after the image is dragged on the target element:



Offline Web Applications API

- HTML5 supports offline Web applications that allow a user to work with them without being online.
- The offline Web applications works by saving all the Web pages locally on the user's system.
- This concept is also known as **Application Cache**.
- The **Application Cache** enables all resources, such as HTML, JavaScript, images, and CSS pages of an Web application to be stored locally on the system.
- Following are the steps that can be taken to cache resources locally on the system:

1. Create a manifest file to define the resources that need to be saved.

2. Reference the manifest file in each Web page designed to use cached resources.

Creating a Manifest File 1-2

- The manifest file is a text file that defines the caching behavior for resources used by the Web page.
- The file should be saved with the `.manifest` extension.
- The Code Snippet demonstrates creation of a manifest file.

```
CACHE:
# Defines resources to be cached.
   check.js
   styles.css
   images/figure1.jpg

FALLBACK:
# Defines resources to be used if non-cached resources cannot be
# downloaded
  Other_images/ figure2.png

NETWORK:
# Defines resources that will not be cached.
  figure3.png
```

Creating a Manifest File 2-2

- Following are the sections defined in the `.manifest` file:

CACHE

- This section defines resources, such as `check.js`, `styles.css`, and `figure1.png` to be stored locally.

FALLBACK

- This section defines alternative resource to be used, when the actual resource is not available.

NETWORK

- This section specifies resources to be accessed when there is a network connection. Resources in this section are not cached.

Declaring a Manifest 1-3

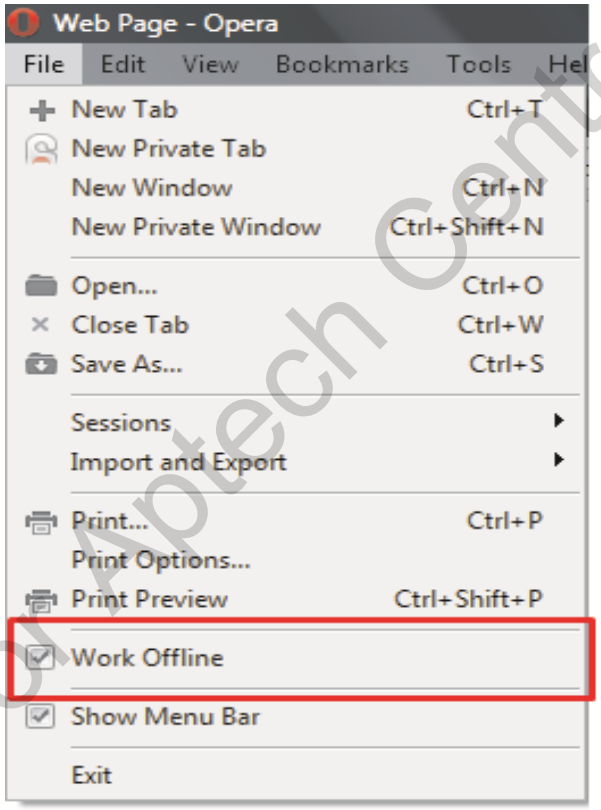
- To associate a manifest with a Web page, assign `.manifest` file to the attribute named `manifest` specified with the `html` element.
- The Code Snippet demonstrates how to add the `.manifest` file in an HTML document.

```
<!doctype html>
<html manifest="appcache.manifest">
 <head>
  <title> Web Page </title>
     <link rel="stylesheet" href="styles.css"/>
     <script type="text/javascript" src="check.js"></script>
 </head>
 <body>
  <input type="button" value="click Here..." onClick="display()"/>
  <img src="images/figure1.jpg" width="100" height="100"/>
 </body>
</html>
```

- In the code, the "`appcache.manifest`" is specified with the `<html>` tag.
- The interpretation of the manifest file is similar to any other file reference.
- The document uses a relative file path, as both the manifest file and HTML document are located in the same directory.
- By default, a Web page declaring manifest is cached automatically.

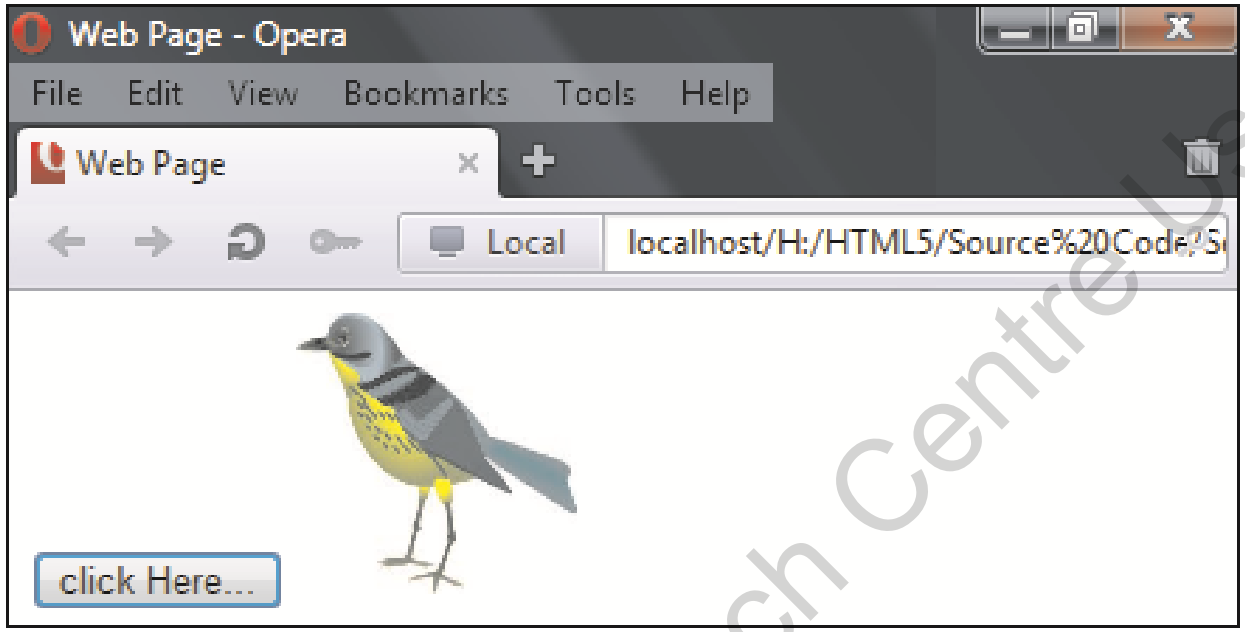
Declaring a Manifest 2-3

- The benefit of Application Cache is that it improves performance of a Web page by reducing the number of requests made to the Web server.
- The Web server hosts the Web application to be accessed on the network.
- Following figure shows how to enable the **Work Offline** mode in the Opera browser:



Declaring a Manifest 3-3

- Following figure shows the cached Web page in the Opera browser:



Summary

- Geolocation determines the current location of a user on devices.
- The location is represented as a single point on a map that comprises two components: latitude and longitude.
- The Geolocation API is a specification provided by the W3C which provides a consistent way to develop location-aware Web applications.
- Google Maps API is used to display the user's location on the map.
- The object of type Map is created in JavaScript, before it can be referenced in an HTML document.
- The drag-and-drop operations defines an event-based mechanism using which elements on a Web page can be copied, reordered, or deleted.
- HTML5 supports offline Web applications that allow a user to work with them without being online.