

### Register and Create

```
// Register element 'my-element'
and return constructor
var elementConstr = Polymer({is:
'my-element'})
// Register element 'my-element'
with factory implementation
var elementConstr = Polymer(
{ is: 'my-element',
  factoryImpl: function (v1, v2)
  {
    this.v1 = v1;
    this.v2 = v2
  }});
// Create element
var myElement = new
elementConstr();
OR
var myElement =
document.createElement('my-element'
);
Create element with factory
implementation
var myElement = new
elementConstr(1, 2);
```

### Extend existing element

```
// Extend existing input element
var myElement = Polymer({
  is: 'my-input',
  extends: 'input'
});
```

### Data binding

<code>{{prop}}</code>	bind property 'prop', get child custom element notifications
<code>[[prop]]</code>	bind property 'prop', DO NOT get child custom element notifications
<code>{{prop::eventname}}</code>	bind property 'prop' when event 'eventname' occurs

### Data binding (cont)

<code>{{obj.prop}}</code>	bind property 'prop' of 'obj' object property
<code>{{!bool}}</code>	negate and bind boolean property 'bool'
<code>{{compute(a, b)}}</code>	execute 'compute(a, b)' and bind
<code>&lt;my-element selected\$="{{value}}"&gt;&lt;/my-element&gt;</code>	bind 'value' to attribute 'selected'
<code>&lt;my-element selected="{{value}}"&gt;&lt;/my-element&gt;</code>	bind 'value' to property 'selected'

### Behaviors

```
my-behavior.html
<script>
myBehavior = {
  listeners: {'click' :
'changeColor'},
  changeColor: function () {
    this.color = 'blue';
  }
}
</script>
my-element.html
<link rel="import" href="my-
behavior.html">
Polymer({
  is: 'my-element',
  behaviors: [myBehavior]
});
```

### Register observers

```
Polymer({
  is: 'my-element',
  properties: {
    color: String,
    arr: {
      type: Array,
```

### Register observers (cont)

```
      value: function
() {return [];}
    },
    obj: {
      type: Object
    }
  },
  observers: [
    'obs1(obj.prop2)',
    'obj2(color)',
    'obs3(obj.prop2, color)',
//Observe mupleple
    'obs4(obj.*) //Observe deep,
    'obs5(arr.splices) //Observe
array splices
  ]});
```

### Debouncing

```
Polymer({
  is: 'my-element',
  properties: {
    prop1: String,
    prop2: String
  },
  observers:
['doSomething(prop1, prop2)'],
  doSomething: function () {
    this.debounce('doSomething',
function () {
    console.log('called
once...');
  }, 300);
  }
});
```

When 'prop1' and 'prop2' are changed within 300ms time frame, 'doSomething' callback is executed once.



### Lifecycle Callbacks

created	when element is registered
ready	when element is registered and local DOM is ready
attached	when element is attached to DOM
detached	when element is removed from DOM
attributeChanged	when element attribute is changed

### Styling

:host	style host element
:host ::content	style distributed content
-- custom-property: value	create css property 'custom-property' and give it a 'value'
--mixin-name: {mixin contents}	create mixin 'mixin-name' with contents 'mixin contents'
color: var(-my-color, red)	apply custom property 'my color', set default to red
color: @apply(-mixin-name)	apply mixin 'mixin-name' to a property 'color'

### Import styles

```
style.html
<dom-module id="my-style">
  <template>
    <style>
      :host {
        --text-style: blue;
      }
    </style>
  </template>
</dom-module>
```

### Import styles (cont)

```
</style>
</template>
</dom-module>
my-element.html
<link rel="import"
href="style.html">
<dom-module id="my-element">
  <style include="my-style"></style>
</dom-module>
```

### Style from light DOM

```
<style is="custom-style">
  my-element {
    --text-style: red;
  }
</style>
```

### Dom repeat

```
<template is="dom-repeat" items=
{{table}} as="myTable">
<div>{{myTable.item}}</div>
</template>
...
Polymer ({ ...
  ready: function ()
{this.table = [{item: 'item1'},
{item: 'item2'}];}
...});
```

### Sort and Filter

```
<template is="dom-repeat" items=
{{table}} filler="myFilter"
sort="mySort">
<div>{{table.item}}</div>
</template>
...
Polymer ({ ...
  ready: function ()
{this.table = [{item: 'item1'},
{item: 'item2'}];},
```

### Sort and Filter (cont)

```
  myFilter: function (item)
{...},
  mySort: function (item)
{...}
...});
```

### Extending Behaviors

```
<link rel="import"
href="oldbehavior.html">
<script>
  NewBehaviorImpl = {
  }
  NewBehavior = [ OldBehavior,
NewBehaviorImpl ]
</script>
```

### Add event listeners

```
// Add event listeners as a element
prototype property
<dom-module id="my-element">
<template>
  <div
id="elementID1">Element 1</div>
  <div
id="elementID2">Element 1</div>
</template>
</dom-module>
{
  Polymer({
    is: 'my-element',
    listeners: {
      'click.elementID1' :
'handleClick1',
      'click.elementID2' :
'handleClick2'
      'click' : 'handleClick' //
Every click
});
// Adding event listeners on
elements
<element on-{event name} =
"{handler}"></element>
```

### Observe mutations

```

this._observer = Polymer.dom(this).
  observeNodes(function
    (info) {
      console.log ('Added
nodes: ' + info.addedNodes);
      console.log
('Removed nodes: '+
info.removedNodes);
    });
Polymer.dom(this).unobserveNodes
(this._observer)

```

### Property Definition

type type of a property (String, Array, Boolean, Number, Object)

value default value

readOnly set read only property

notify fire 'property-changed' event, when property is changed. Handy for nested data binding

reflectToAttribute update attribute when property is changed

observer execute provided Callback when property is changed

computed compute value based on other property values

### Example for *color* and *underline* properties:

```

properties: {
  color: String,
  underline: {
    type: Boolean,
    value: false,
    observer: 'applyDecoration',
    reflectToAttribute: true,
    notify: true
  }
}

```

### Property modifications

this.set('property', value) set this.property to 'value'

this.set('arr.1', value) set this.arr[1] to 'value'

this.set('obj.prop1', value) set this.obj.prop1 to 'value'

this.get('property') get value of this.property

this.get('obj.prop1') get value of this.obj.prop1

this.get('arr.1') get value of this.arr[1]

This API is required to observe properties of Array and Object types.

### Polymer API

Dom Polymer.dom(element).method manipulation

Shadow this.root

Host this

this.\$.myid local DOM element with id = 'myid'

this.\$\$('selector') "querySelector" in local DOM

contentElement.getDistributionNodes(-)

element.getDestinationInsertionPoints(-)

return <content> elements for light DOM 'element', where 'element' is distributed to local DOM

### Polymer API (cont)

contentElement.getContentChildren() return distributed elements to local DOM that are children of 'contentElement'

.{query | get}EffectiveChildren() return effective children for nesting when host element contains <content> tag

.getEffectiveChildNodes() same as above for nodes

### Conditional Template

```

<template is="dom-if" if="{user.isAdmin}">
  Only admins can see this.
</template>

```

### Utility Functions

toggleClass() toggle class

toggleAttribute() toggle attribute

attributeFollows() move attribute from one element to another

classFollows() move class from one element to another

async() call asynchronously

fire() dispatch event

cancelDebouncer() cancel given debouncer

flushDebouncer() call debouncer task and cancel debouncer

isDebouncerActive() return true if active, false otherwise

transform() transform element (css transform function)