# Proposal: Use properties that are Object.observeable as data binding interop between web components

Definition: Object.observeable properties: Properties that don't have getters/setters and are not native.

## Polyfill for native elements

* When Object.observe is available: use Object.getNotifier() that is triggered by an event listener (see <http://wiki.ecmascript.org/doku.php?id=harmony:observe_api_usage>)
* For older browsers, without Object.observe: create for every native property a corresponding property with prefix observable (e.g. observableValue) that is updated via an event listener and has special getters/setters to detect changes

## Why?

* simplification:
  + Assuming that data binding frameworks will use Object.observe under the hood already, relying on Object.observe would simplify the work that needs to be done in the frameworks
  + Object.observeable properties could already work out of the box to bind between Angular and Polymer components.
  + Angular would not need to have a fast path for Angular components that use Angular components (i.e. where it does not fire events), but could use the same way for observing properties on Angular components and on third party components.
* ? performance (not tested): Object.observe should be faster than firing events, especially when Angular is already using Object.observe.

## Observation: Angular and Polymer component authors do not need to care about events or Node.bind

### Angular components that use Angular components

|  |
| --- |
| **Usage in a template:**  <my-comp bind-name="childName"/>  **Component definition:**  @ComponentDirective({  selector: 'my-comp',  observe: {'internalName', 'nameChanged'}  bind: {'name', 'internalName'}  })  class MyComp {  constructor() {  this.internalName = 'someValue';  }  nameChanged(newValue, oldValue) {  ...  }  } |

### Polymer components that use Polymer components

|  |
| --- |
| **Usage in a template:**  <my-child-comp name="{{childName}}"/>  **Component definition:**  <polymer-element name="my-comp" attributes="name">  <script>  Polymer('my-comp', {  observe: {  'name': 'nameChanged'  },  ready: function() {  this.name = 'someValue';  },  nameChanged(newValue, oldValue) {  ...  }  });  </script>  </polymer-element> |

### Vanilla JS (with event notification on property changes):

|  |
| --- |
| **Usage:**  var container = document.createElement('my-comp');  container.addEventListener('name', function() {  ...  });  **Component definition:**  var MyCompProto = Object.create(HTMLElement.prototype);  Object.defineProperty(MyCompProto, 'name', {  get: () => { return this.\_name; },  set: (value) => {  this.\_name = value;  this.dispatchEvent(new CustomEvent('propertychange',  { properties: ['name']});  }  });  MyCompProto.createdCallback = function() {  this.name = 'someValue';  };  document.registerElement('my-comp', {prototype:MyCompProto}); |

### 

### Vanilla JS (with Object.observeable properties):

|  |
| --- |
| **Usage:**  var container = document.createElement('my-comp');  Object.observe(container, function(changes) {  ...  });  **Component definition:**  var MyCompProto = Object.create(HTMLElement.prototype);  MyCompProto.createdCallback = function() {  Object.observe(this, function(changes) {  ...  }  this.name = 'someValue';  };  document.registerElement('my-comp', {prototype:MyCompProto}); |

## Discussion

Pro

* See above for why

Con

* Requires a polyfill for native elements
  + but: using the events approach we also need a lookaside table to match native events to the properties on native events that changed.