

Event Aggregation

Events

- Many diverse components generate events
- These events may incur changes in model that need to be reflected in other components
- Unstructured co-ordination between components may yield unmanageable complexity



Publish Subscribe Messaging (Pub/Sub)



- Facilitate decoupled messaging between components
- Promote a more ordered evolution of dependencies

Aurelia Event Aggregator

A lightweight pub/sub messaging system for app-wide loosely coupled events.

- Message
- Broker (the EventAggregator)
- Publisher
- Subscriber
- Publish Method
- Subscribe Method
- Subscribe Once Method
- Channels

Message

- This is the 'Event' that can be transmitted through the EventAggregator
- It can be any class that meaningfully encapsulates information about the event.
- Often defined in a 'messages.js' module

```
export class ExampleEvent {  
    constructor(message) {  
        this.message = message;  
    }  
}
```

Broker (EventAggregator) + Publisher

- EventAggregator object imported from framework...
- ... and 'injected' into publisher object

```
import {inject} from 'aurelia-dependency-injection';
import {EventAggregator} from 'aurelia-event-aggregator';
import {ExampleEvent} from './example-event';

@Inject(EventAggregator)
export class ExamplePublisher {

  constructor(eventAggregator) {
    this.ea = eventAggregator;
  }

  ...
}
```

Broker (EventAggregator) + Subscriber

- EventAggregator object imported from framework...

```
import {inject} from 'aurelia-dependency-injection';
import {EventAggregator} from 'aurelia-event-aggregator';
import {ExampleEvent} from './example-event';

@Inject(EventAggregator)
export class ExampleSubscriber {

  constructor(eventAggregator) {
    this.ea = eventAggregator;
  }

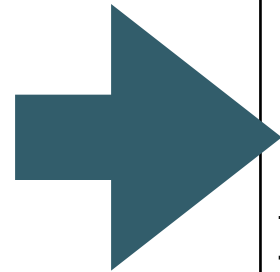
  ...
}
```

- ... and 'injected' into subscriber object

- Publisher & Subscriber share ExampleEvent class

Publish Method

- Create event object and dispatch to all subscribers



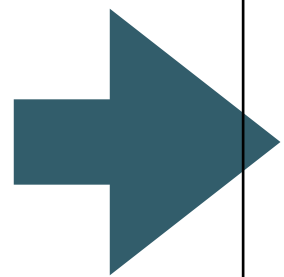
```
import {inject} from 'aurelia-dependency-injection';
import {EventAggregator} from 'aurelia-event-aggregator';
import {ExampleEvent} from './example-event';

@inject(EventAggregator)
export class ExamplePublisher {
  constructor(eventAggregator) {
    this.ea = eventAggregator;
  }

  publish() {
    this.ea.publish(new ExampleEvent('Some Event'));
  }
}
```


Subscribe Method

- Subscribe to events - callback triggered when event occurs



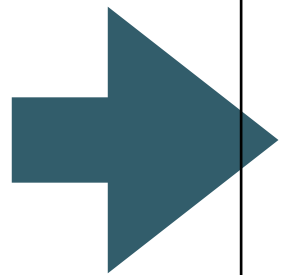
```
import {inject} from 'aurelia-dependency-injection';
import {EventAggregator} from 'aurelia-event-aggregator';
import {ExampleEvent} from './example-event';

@inject(EventAggregator)
export class ExampleSubscriber {
  constructor(eventAggregator) {
    this.ea = eventAggregator;
  }

  subscribe() {
    ea.subscribe(ExampleEvent, event => {
      console.log(event.message);
    });
  }
}
```

SubscribeOnce Method

- Subscribe for first occurrence of event only



```
import {inject} from 'aurelia-dependency-injection';
import {EventAggregator} from 'aurelia-event-aggregator';
import {ExampleEvent} from './example-event';

@Inject(EventAggregator)
export class ExampleSubscriber {
  constructor(eventAggregator) {
    this.ea = eventAggregator;
  }

  subscribe() {
    ea.subscribeOnce(ExampleEvent, event => {
      console.log(event.message);
    });
  }
}
```

Channels

- Alternative pub/sub mechanism - named channels

Publisher

```
import {inject} from 'aurelia-framework';
import {EventAggregator} from 'aurelia-event-aggregator';

@inject(EventAggregator)
export class APublisher {
  constructor(eventAggregator) {
    this.eventAggregator = eventAggregator;
  }

  publish(){
    var payload = {};
    this.eventAggregator.publish('channel name here', payload);
  }
}
```

Subscriber

```
import {inject} from 'aurelia-framework';
import {EventAggregator} from 'aurelia-event-aggregator';

@inject(EventAggregator)
export class ASubscriber {
  constructor(eventAggregator) {
    this.eventAggregator = eventAggregator;
  }

  subscribe() {
    this.eventAggregator.subscribe('channel name here', payload => {
      ...
    });
  }
}
```

Example: Donation Total

donation-service.js

- Publisher

messages.js

```
export class TotalUpdate {  
  constructor(total) {  
    this.total = total;  
  }  
}
```

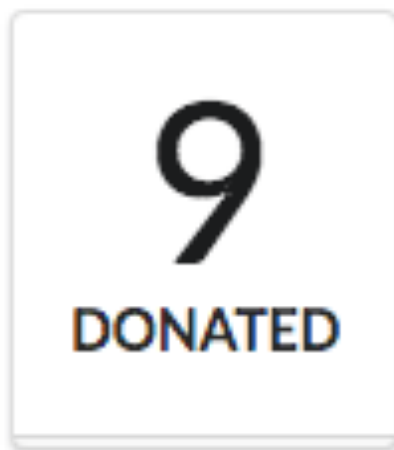
```
import {inject} from 'aurelia-framework';  
import {TotalUpdate} from './messages';  
import {EventAggregator} from 'aurelia-event-aggregator';  
  
@inject(Fixtures, EventAggregator)  
export default class DonationService {  
  
  donations = [];  
  total = 0;  
  ...  
  
  constructor(ea) {  
    ...  
    this.ea = ea;  
  }  
  
  donate(amount, method, candidate) {  
    const donation = {  
      amount: amount,  
      method: method,  
      candidate: candidate  
    };  
    this.donations.push(donation);  
    this.total = this.total + parseInt(amount, 10);  
  
    this.ea.publish(new TotalUpdate(this.total));  
  
  }  
  ...  
}
```

Example: Donation Total

- Subscriber

stats.html

```
<template>
  <section class="ui stacked statistic segment">
    <div class="value">
      ${total}
    </div>
    <div class="label">
      Donated
    </div>
  </section>
</template>
```



stats.js

```
import {inject} from 'aurelia-framework';
import {TotalUpdate} from '../services/messages';
import {EventAggregator} from 'aurelia-event-aggregator';

@inject(EventAggregator)
export class Stats {

  total = 0;

  constructor(ea) {

    ea.subscribe(TotalUpdate, msg => {
      this.total = msg.total;
    });

  }
}
```