# Event System and Its Application

## Why there is an event system:

A lot of stuff happens in this volunteer system, when something happened you need to handle it.

Since you cannot know what need to be done for an event in advance, and this may change a lot, you cannot hard code the event handling code where the event is generated. Moreover that one handler can handle multiple events. And let the event handler to query the system state to trigger the handler would by very inefficient. So, to handle the events efficiently and flexibly we need an event system.

## Event System Design:

Event System is heavily relays on the **indexing of the database**. Proper indexing the database would significantly improve the performance.

The event system is composed by: manager, publishers, subscribers, and handler.

### Event

Event is something just happened. It is a data. It has the following field.

|  |  |  |
| --- | --- | --- |
| Event Type(string) | Event Value(object) | Event Sender(object) |
| The type of this event  (like point change, activity finished) | The value of the event  (Eg. For an point change event, the value is the changed point) | The sender of it (maybe a Guid of an user or an activity) |

### The manager

The manager is very simple. It is a singleton class under the event service namespace. It lets other to add subscribers and handlers to the event system or remove them from the event system.

### Publisher

The publisher is anything that uses the event service to publish an event.

When publishing an event you will need to provide the 3 fields list above.

### Subscriber

A subscriber is a record stored in the event database. When an event happened if fetches the record that matches the event and find the handler of this event.

The subscriber record has the following fields:

|  |  |  |  |
| --- | --- | --- | --- |
| Trigger Event types: List<string> | Trigger Senders: object | Subscriber name: string | Handler name: string |
| List of event types. Any event in this list will trigger the event subscriber (if the sender matched) | Only the sender match this filed will trigger this subscriber. You can leave this to null, so any sender will match | The name of the subscribers, this is unique in the subscriber database collection | Which handler has to handle this event |

When an event is fired the event service will find the matching subscriber record from the database. **The matching rules are**: 1. if the event type is contained in the subscriber event type list; 2. The sender is the same as the subscriber’s sender field. The matching will be done by the database’s **indexing function**, so although there may be millions of subscribers, to find one is very easy. Properly built index is very important.

### Handler

A handler handles the event. The subscriber will tell the handler the event and the subscriber name. It’s up to the handler to hander the event.

The handler interface is defined in the event service project. The implementation of the handler is really depends on the demand.

### The event handling flow chart

### 

## Notice System

Notice system is the simplest application of the event system.

When something happened, the notice handler will send a message to the designated user. Different event will trigger different subscribers that have notice system as the handler. The notice system handler will decide which message to send mostly on the subscriber name. The content of the message and the receiver may vary depend on the event value.

## Badge System

On the exact opposite, the badge system is very complex.

Since there are many kinds of badges and anyone can get any of them. Moreover, to grand a user with a badger, the condition is very complex. One badge may require different event. Like a badge may require you have gained 400 points and attend 20 activities.

**The badge processing flow chart is show below:**



### A badge Example:

The Badge B1 requires the following condition to be met:

|  |  |  |
| --- | --- | --- |
| Conditon ID | Trigger event type | Condition |
| C1 | Point change | Point>100 |
| C2 | Event finish | Finished event >20 |
| C3 | Finished Act X | Finished Act X |

So this badge will have an event subscriber:

|  |  |  |  |
| --- | --- | --- | --- |
| Trigger Event types: | Trigger Senders: | Subscriber name: | Handler name: |
| [Point change, Act finish, | AnyOne | B1 | BadgeHandler |

Let say, user U1 has finished Act X, this will generate 2 event in sequence: Act Finish, Point Change.

This will trigger the subscriber above 2 times.

For the first event, the subscriber will find the corresponding badge handler to handle this event. It first will call badge service to check if the user has got this badge already. If the user has this badge it will discard this event.

Inside the B1 handler it find out that this event met the condition C3, so it will user badge service to set C1 is met. In the badge service, it will first try to find a badge entity which is a partly or fully granted badge. If not it will creates badge entity like {B1:[C1,C2,C3]:U1} and mark the C3 as satisfied.

Then for the second event, let say the U1 has finished 17 activities, so this is the 18th. The same as above all the way in to B1 class, it discovered that it met no condition so nothing will happen.

Then for the point change event, let say U1 has 99 points, this activity have given him 10 points making it 109. In the B1, it met C1. It calls the badge service to satisfy the C1 condition. It found the badge entity and mark the C1 as satisfied.

Ok, let’s say U1 has finished and other activities (activity Y), it’s his 19th.

As said above, it will trigger 2 events, both will trigger the above subscriber (activity finished, point change).

This time the activity finished event is the same as above it met no condition. But the point change event met the C1 condition again, the B1 handler class will ask the badge service to satisfied C1. The badge service find the condition is already met so nothing will happen.

Until U1 finishes another activity, this time all the way into the B1 Class it finally met the condition C2. It ask the badge service to satisfied C2, and the service will return true to indicate thaht the badge has been granted.