Passage du Cardinal 1  
1700 Fribourg  
Switzerland

Event Actors Documentation

|  |  |
| --- | --- |
| Document responsible: | Albert Adiyatullin |
| Last changed | 23.01.2018 |
| Description: | This document contains the documentation of the Event Actors framework. |

Table of Contents

[Description 2](#_Toc504486421)

[Classes 3](#_Toc504486422)

[Controller Actor 3](#_Toc504486423)

[View Actor 4](#_Toc504486424)

[Event 4](#_Toc504486425)

[Sending and handling of the events 4](#_Toc504486426)

[Sending the events from the Controller to the View 4](#_Toc504486427)

[Sending the events from the View to the Controller 4](#_Toc504486428)

[Checklist: How to register an event 5](#_Toc504486429)

[Event processed by the View 5](#_Toc504486430)

[Event processed by the Controller 5](#_Toc504486431)

# Description

The Event Actor framework provides a way to send messages between the actors at zero coupling between them. This means that instead of sending a specific message of every receiver using receiver’s enqueuer:



we send a message with a virtual “Do” method, which has a concrete implementation for every specific receiver.



To distinguish different types of messages, an Event formalism is used: for every message, a single Event class (common for all actors) is defined. Each actor can handle this Event in its own way, i.e. have a specific Event Handler for this Event.

In the hierarchy of the framework, there is one Controller and multiple Views. The Events can be sent only between the Controller and any of the Views. All the Views are nested actors of the Controller; the whole list of the Views, Events, and Event Handlers is recorded at the Controller’s side.

Each Event can be processed both by the Controller and the Views. Each particular View subscribes for the Events that it would like to receive from the Controller; the other Events will not be translated to this View.

Each View can have three flags:

* Receive only: the View cannot send any Events to the Controller
* Exclude from broadcast: the View cannot receive any Events from the Controller
* Paused: the View cannot neither send nor receive any Events

The flags can be changed during the run-time of the Actors.

# Classes

The Event Actors framework provides abstract classes for three entities: Controller Actor, View Actor, and Event.

## Controller Actor

This is an abstract parent class for concrete Controller actors.

It has dictionaries for all the Views, Events, and Event Handlers in the system, and governs their overall behaviour. It provides the whole functionality:

- to add/remove new Views and Events;

- to change View Settings;

- to generate Events for the Views / handle and broadcast Events from the Views.

## View Actor

This is an abstract parent class for concrete View actors.

Its includes an override method to register the events to the Controller, as well as two methods that allow to generate an event for the Controller only, or to generate event and let the Controller broadcast it to all the other Views.

## Event

This is a parent class for concrete Events.

This class supports methods for reading/writing the event name and serializing/deserializing. Currently the events are differentiated not by name, but by their IDs, which correspond to their locations in the computer file system.

Child classes for the events with the most common data types are stored in the folder “Event Data Types”.

# Sending and handling of the events

While an Event is being registered to the Controller, a Handler for this Event is also recorded. When the Event is sent, it is processed by the corresponding Handler. However, the way of sending and processing the events on the Controller and the View side is slightly different.

## Sending the events from the Controller to the View

As discussed in the Description, zero coupling implies that an Event can be processed by different Views in a different way. Therefore, the Event data is sent using the class “Abstract Event Message for View” that inherits from the standard Message class. Its “Send …” method is called by the Controller, and “Do” is implemented on the View side.

To sum up: on the View side, an Event is handled by the “Do” method of a child of “Abstract Event Message for View” class.

## Sending the events from the View to the Controller

The Event data sent from the View is handled by the controller (different behaviour for different Events), and also can be broadcasted to the other Views (same behaviour for all the Events). To facilitate the operation, the Event data is transmitted in two steps:

1. Event data is transmitted using a standard Message for the Controller method “Handle Event” or “Handle and Broadcast event”. A corresponding Event Handler (inheriting from the class “Abstract Event Handler for Controller”) is taken from the Controller dictionary, and Event data is processed by this Handler.
2. Event data is broadcasted to the other Views using the previously defined formalism.

To sum up: on the Controller side, an Event is handled by the “Handle Event” method of a child of “Abstract Event Handler for Controller” class, and broadcasted using the “Send…” method of “Abstract Event Message for View” class.

# Checklist: How to register an event

First of all, for every event, a new class inheriting from the Event class (or from any of its subclasses present in Event Data Types folder) should be created.

The event can be send both from Controller to View or back. It also can be handled in both the Controller and the Views. This is convenient when one View wants to broadcast the information to all the other Views.

## Event processed by the View

1. Create a handle method and a message for it.
   1. In the View, create a “Handle …” method, in which the event is handled.
   2. Create a message class for this method by inheriting from the “Abstract Event Message for View” class. Change the icon, otherwise you will not distinguish different messages during the event registration.
   3. In the message class, create a “Do” override method by copypasting any of the existing ones. It should call the “Handle …” method.
2. Let everyone know about the event
   1. Register this event in the “Register Events To Controller” override method of the View class. Provide the message class as the event handler.
   2. Register this event in the “Define Events” override method in the Controller class.
3. Call the event by using the Generate Event method of the Controller Actor class.

## Event processed by the Controller

1. Create a handle method and a message for it.
   1. In the Controller, create a “Handle …” method, to handle the event.
   2. Create a message class for this method by inheriting from the “Abstract Event Handler for the Controller” class. Change the icon, otherwise you will not distinguish different messages during the event registration.
   3. In the message class, create a “Handle Event” override method by copypasting any of the existing ones. It should call the “Handle …” method.
2. Let everyone know about the event
   1. Register this event in the “Define Events” override method in the Controller class. Send True to the “Source Has Handler” input of the “Add Event” method and provide the message class as a message handler.
3. Call the event by using the Generate Event/Generate and Broadcast Event method of the View Actor class.