TestPipe CQRS Walkthrough

We borrowed a lot of knowledge and code from:

* Greg Young
* Udi
* Eric Evans
* Diary.CQRS Project – CodeProject (actually much of my implementation matches this as it was simple and concise and easy to grasp. It also has a good article accompanying it so it made a perfect fit for my first iteration of a CQRS implementation).
* CQRS Journey Project – Microsoft
* AGRCQRS Project
* CQRS Mailing List
* CQRS and EventSourcing @ Stackoverflow
* …and many more.

# Command/Write Side

Let’s start our walkthrough with processing in the UI. In the end, isn’t it all about the user anyway.

UI

1. The User wants to add a new Application.
2. The User completes the new Application form on the Add Application page.
3. The User submits the new Application form.

Controller

1. The Controller receives the post from the page with the new Application data.
2. The Controller instantiates a new CreateApplicationCommand passing in the Application data as parameters (Id and Title).

Command

1. The CreateApplicationCommand takes the Application data passed in its contructor and sets its internal state.

Controller

1. The Controller uses the IOC Container to instantiate the CommandBus passing in the CommandHandlerFactory responsible for CreateApplicationCommands.
2. The Controller calls Sends on the CommandBus to deliver the CreateApplicationCommand for processing.
3. The Controller Redirects the page to the next Action.

CommandBus

1. In the CommandBus Send method the CommandBus gets the specific Handler for the CreateApplicationCommand from the CommandHandlerFactory.

CommandHandlerFactory

1. The CommandHandlerFactory uses the IOC Container and Reflection to get the proper instance of the CommandHandler for the CreateApplicationCommand (note: the IOC returns all instances of ICommandHandlers then Reflection is used to find the one instance that has an argument matching CreateApplicationCommand).
2. It instantiates the CommandHandler with the configured IEventStoreRepository for the runtime.

CommandBus

1. The CommandBus calls Execute on the CommandHandler it retrieved from the CommandHandlerFactory passing in the CreateApplicationCommand as an argument.

CommandHandler Instance

1. In the CommandHandler Execute method creates a new Application aggregate instance passing in the data retrieved from the Command’s state (Id and Title)

AggregateRoot Instance

1. The Application aggregate constructor creates a new ApplicationCreatedEvent passing in the parameter values it received from the CommandHandler.
2. Then the Application aggregate calls ApplyChange method it inherited from the AggregateRoot class passing in the AppliationCreatedEvent.

AggregateRoot

1. The ApplyChange method uses a dynamic instance of the AggregateRoot instance (note: dynamic just means the instance will be resolved at runtime).
2. A Converter class is used to convert the generic Event to the proper instance of the ApplicationCreatedEvent.

Converter

1. The Converter ChangeTo returns a dynamic that resolves to the specific Event type at runtime.

AggregateRoot

1. The EventHandler’s Handle method is called passing in the ApplicationCreatedEvent (note: EventHandlers are located on AggregateRoot instances that implement IHandle<T> interface with T matching the Event the AggregateRoot instance handles).

AggregateRoot Instance

1. The Handle method for the ApplicationCreateEvent sets the state of the Application aggregate.

AggregateRoot

1. The ApplicationCreatedEvent is placed on the AggregateRoot’s uncommitted change list.

CommandHandler Instance

1. The Application aggregate version is set to -1 (signifying its new).
2. Then the CommandHandler calls save on the Repository passing the aggregate and aggregate version as parameters.

Repository

1. The repository saves the events from the AggregateRoot’s uncommitted change list.
2. If the version of the aggregate isn’t -1 (new), it retrieves the aggregate from the EventStore by its ID and compares the returned aggregate ID to the ID of the current uncommitted event and if they are not equal it will throw a ConcurrencyException.
3. Next the repository calls Save on the EventStore.

EventStore

1. For each uncommitted event
   1. The EventStore increments the aggregate version.
   2. If the version matches the snapshot criteria, the aggregate is saved as a Memento.
2. For each uncommitted event
   1. The EventStore publishes the event on the EventBus.

This ends the walk through of the command/write side of the CQRS application.

## Query/Read Side

>UI Write Command

> CommandBus

> HandleCommand

> Create Event (versioned schema)

> Serialize EventSource

> Persist and Publish EventSource

> Deserialize EventSource

> HandleEvent (map EventSource to ViewData)

> Persist ViewData

> Query ViewData

> UI Read ViewData

Command creates an event

Event is serialized and saved

When replaying we need to deserialize and handle event

On read side we need to also deserialize and handle event

CreateApplicationCommandHandler : ICommandHandler<CreateApplicationCommand>

Create new Application(command.Id, command.Title) : AggregateRoot

Application.AggregateRoot.ApplyChange(new ApplicationCreatedEvent(Guid.NewGuid(), id, title))

AggregateRoot.ApplyChange(Event @event)

Application.Handle(dynamic source, string type)

Application state is set.

SqlEventStore.Save(aggregate, aggregate.Version)

Application.AggregateRoot.GetUncommitedEvents() – returns internal state for IEnumberable<Event> uncommitedEvents

ForEach @event in uncommittedEvents

Increment version number

Every 3rd version

Cast AggreateRoot to IOriginator

Use IOriginator to Application.GetMemento()

Create new ApplicationMemento(this.Id, this.Key, this.Title, this.Environments, this.Version) : BaseMemento

Set BaseMemento.Version

SaveMemento(BaseMento) – here we lost ApplicationMemento properties

Set @event version

ChangeToEventSource(@event) – translate to persisted schema from runtime schema

SaveEventSource(eventSource)

eventSources.Add(eventSource)

ForEach eventSource in eventSources

Publish(eventSource)