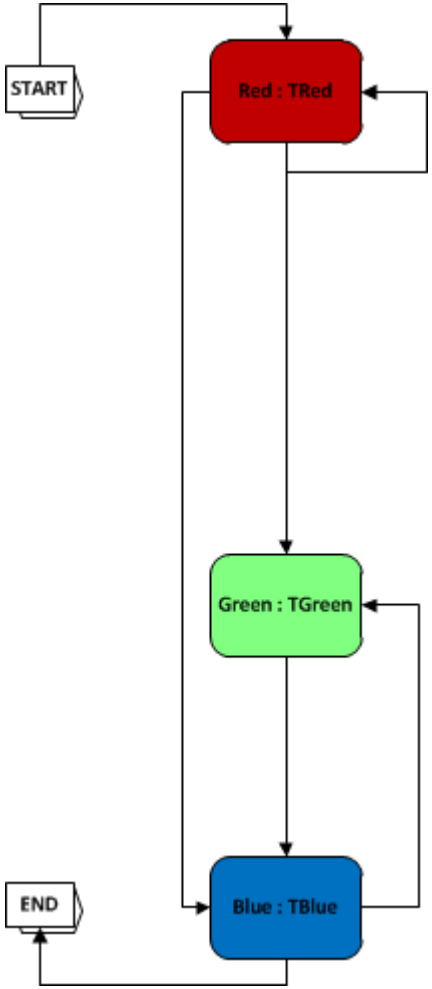
WorkflowManager<S,P>

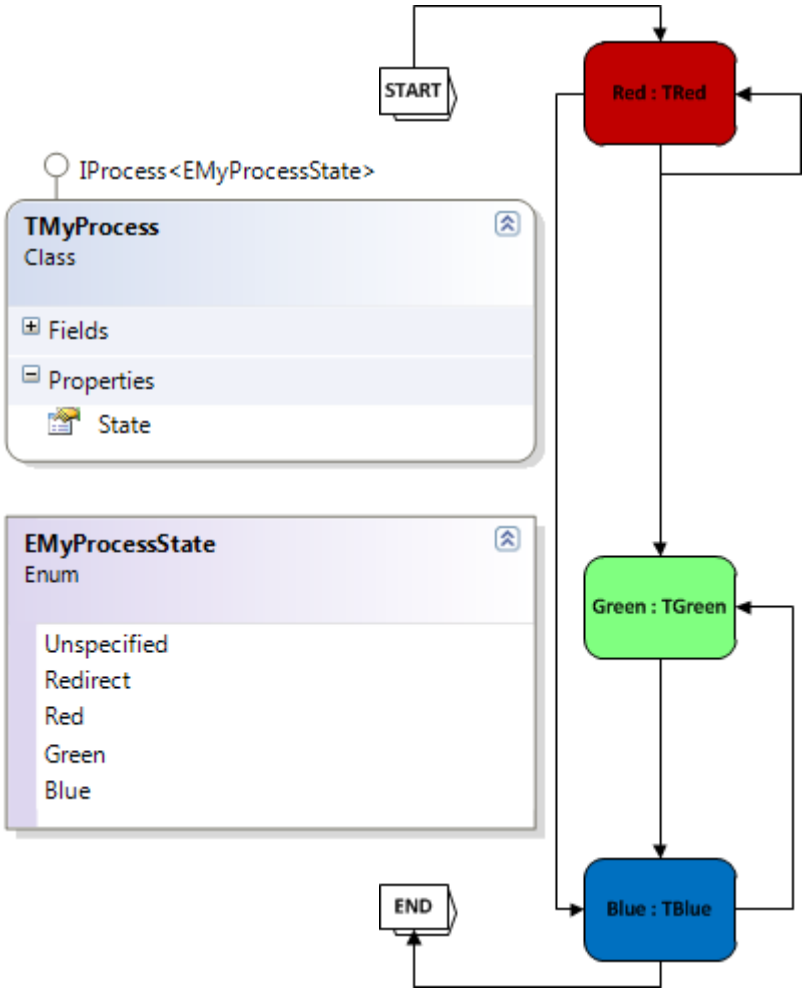
Implementing workflows in web applications

hasan.karahan81@gmail.com

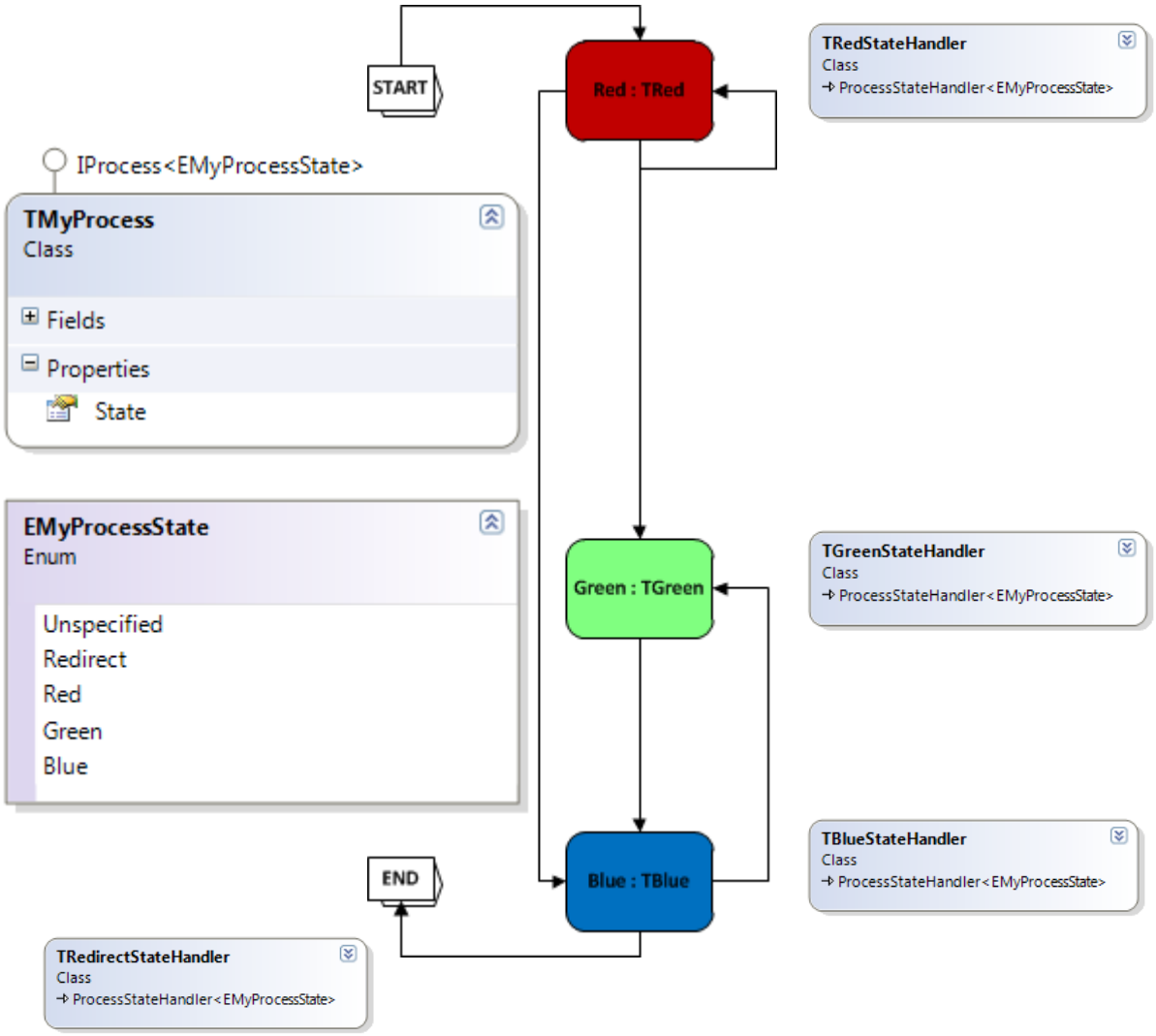
TMyWorkflow



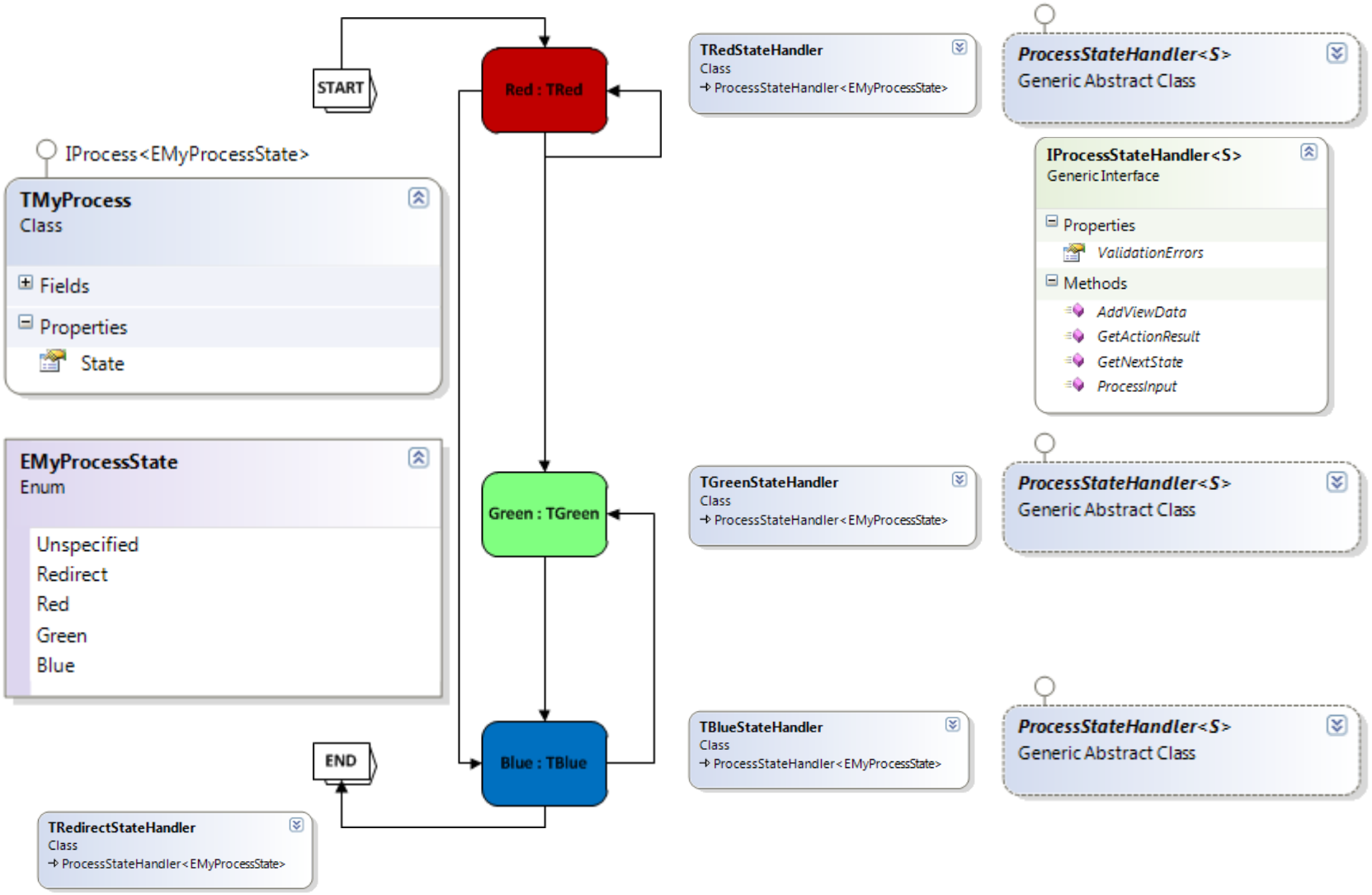
TMyWorkflow: IWorkflow<EMyWorkflowState>

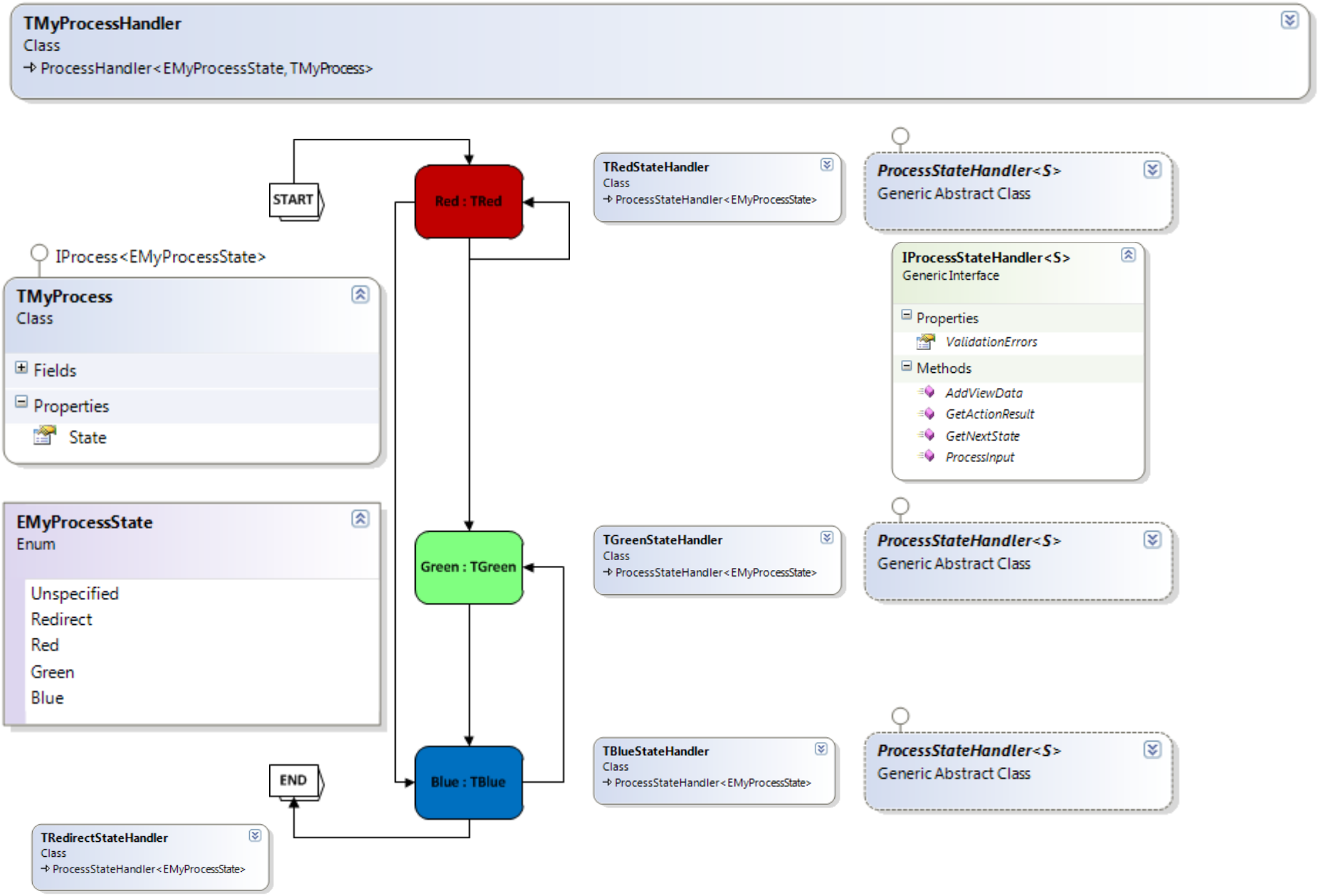


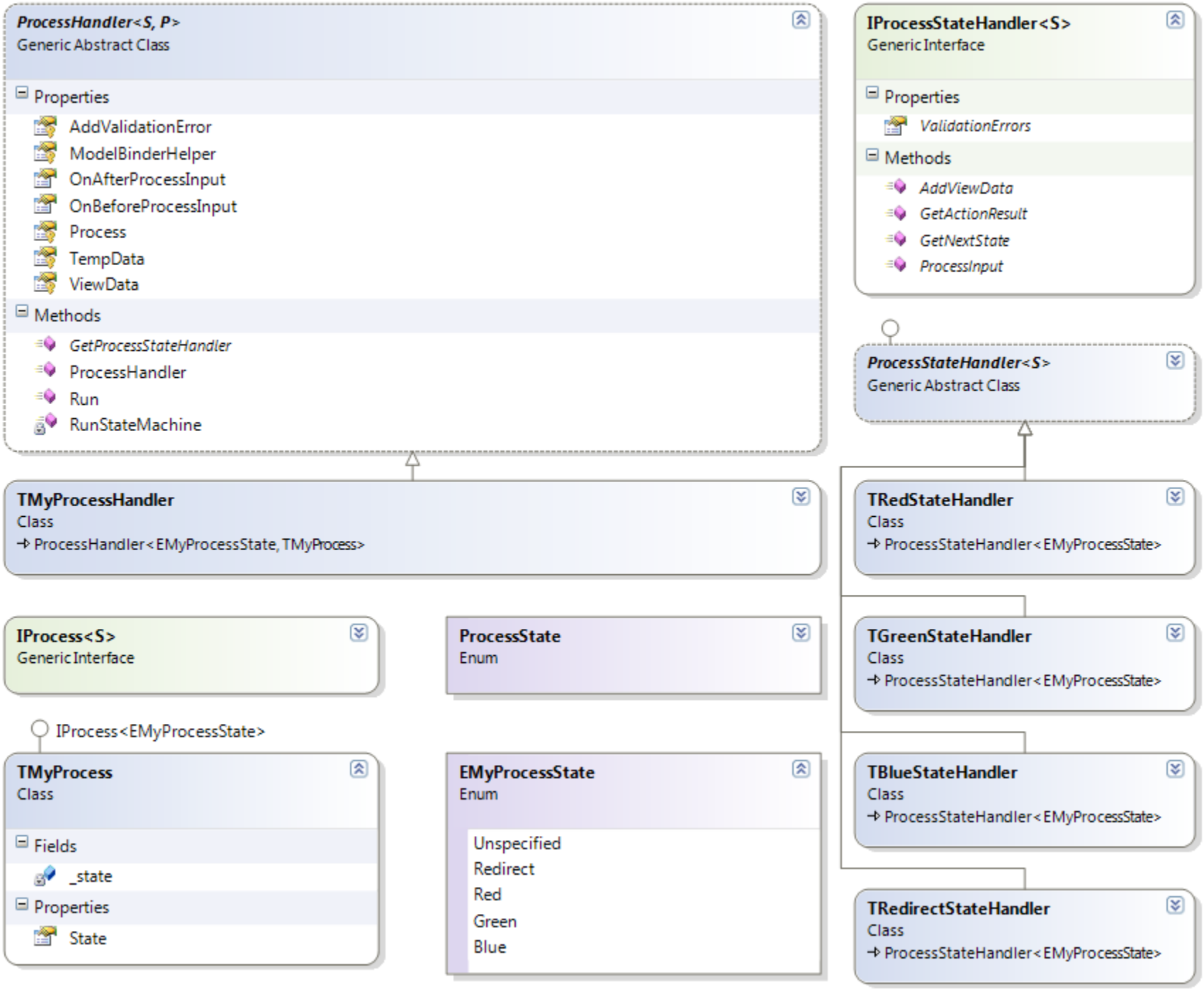
TMyWorkflow: IWorkflow<EMyWorkflowState>



TMyWorkflow: IWorkflow<EMyWorkflowState>



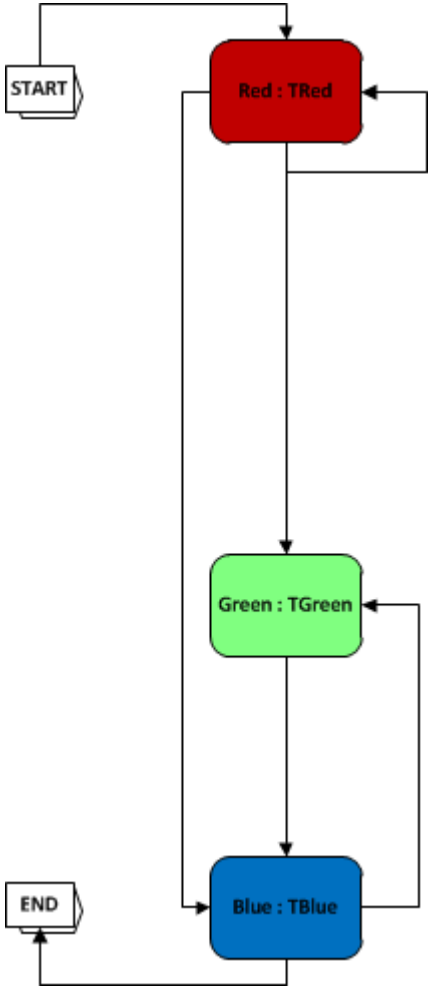

TMyProcessHandler
Class
ProcessHandler<EMyProcessState, TMyProcess>
START
Red : TRed
TRedStateHandler
Class
ProcessStateHandler<EMyProcessState>
ProcessStateHandler<S>
Generic Abstract Class
IProcess<EMyProcessState>
TMyProcess
Class
Fields
Properties
State
IProcessStateHandler<S>
Generic Interface
Properties
ValidationErrors
Methods
AddViewData
GetActionResult
GetNextState
ProcessInput
EMyProcessState
Enum
Unspecified
Redirect
Red
Green
Blue
Green : TGreen
TGreenStateHandler
Class
ProcessStateHandler<EMyProcessState>
ProcessStateHandler<S>
Generic Abstract Class
END
Blue : TBlue
TBlueStateHandler
Class
ProcessStateHandler<EMyProcessState>
ProcessStateHandler<S>
Generic Abstract Class
TRedirectStateHandler
Class
ProcessStateHandler<EMyProcessState>


ProcessHandler<S, P>
Generic Abstract Class
Properties
AddValidationError
ModelBinderHelper
OnAfterProcessInput
OnBeforeProcessInput
Process
TempData
ViewData
Methods
GetProcessStateHandler
ProcessHandler
Run
RunStateMachine
IProcessStateHandler<S>
Generic Interface
Properties
ValidationErrors
Methods
AddViewData
GetActionResult
GetNextState
ProcessInput
ProcessStateHandler<S>
Generic Abstract Class
TMyProcessHandler
Class
ProcessHandler<EMyProcessState, TMyProcess>
TRedStateHandler
Class
ProcessStateHandler<EMyProcessState>
IProcess<S>
Generic Interface
ProcessState
Enum
TGreenStateHandler
Class
ProcessStateHandler<EMyProcessState>
IProcess<EMyProcessState>
TMyProcess
Class
Fields
_state
Properties
State
EMyProcessState
Enum
Unspecified
Redirect
Red
Green
Blue
TBlueStateHandler
Class
ProcessStateHandler<EMyProcessState>
TRedirectStateHandler
Class
ProcessStateHandler<EMyProcessState>

MyController.Default

```
public ActionResult Default(SavedState savedState){

    var workflowManager = new TMyWorkflowManager(
        this.GetMyWorkflowFrom(savedState),
        this.ModelBinderHelper,
        this.AddValidationError,
        this.ViewData
    );

    var actionResult = workflowManager.Run(this.Request);
    if (actionResult is RedirectResult != true){
        this.SerializeMyWorkflow();
    } else {
        this.DeleteSavedState(savedStateId);
    }

    return actionResult;
}

private TMyWorkflow GetMyWorkflowFrom(SavedState savedState){..}
```

Pre- and Postconditions



Require & Ensure

Require & Ensure

```
public override void ScanWarehouseCompartmentManager.Require(){

    if (_workflow.State != StoringPalletWorkflowState.ScanWarehouseCompartment){
        throw new InvalidStateException(_workflow.State.ToString());
    }

    if (_workflow.StoringTour == null){
        throw new StorinTourIsNullException();
    }

    if (_workflow.WarehouseCompartment == null){
        ; //DONE!
    }
}

public override void ScanWarehouseCompartmentManager.Ensure(){

    if (_workflow.State != StoringPalletWorkflowState.ScanWarehouseCompartment){
        throw new InvalidStateException(_workflow.State.ToString());
    }

    if (_workflow.StoringTour == null){
        throw new StorinTourIsNullException();
    }

    if (_workflow.WarehouseCompartment == null){
        throw new WarehouseCompartmentIsNullException();
    }
}
```

MyController.Default

```
public ActionResult Default(SavedState savedState){

    var workflowManager = new TMyWorkflowManager(
        this.GetMyWorkflowFrom(savedState),
        this.ModelBinderHelper,
        this.AddValidationError,
        this.ViewData
    );

    var actionResult = workflowManager.Run(this.Request);
    if (actionResult is RedirectResult != true){
        this.SerializeMyWorkflow();
    } else {
        this.DeleteSavedState(savedStateId);
    }

    return actionResult;
}

private TMyWorkflow GetMyWorkflowFrom(SavedState savedState){..}
```

?