SPT Components Base

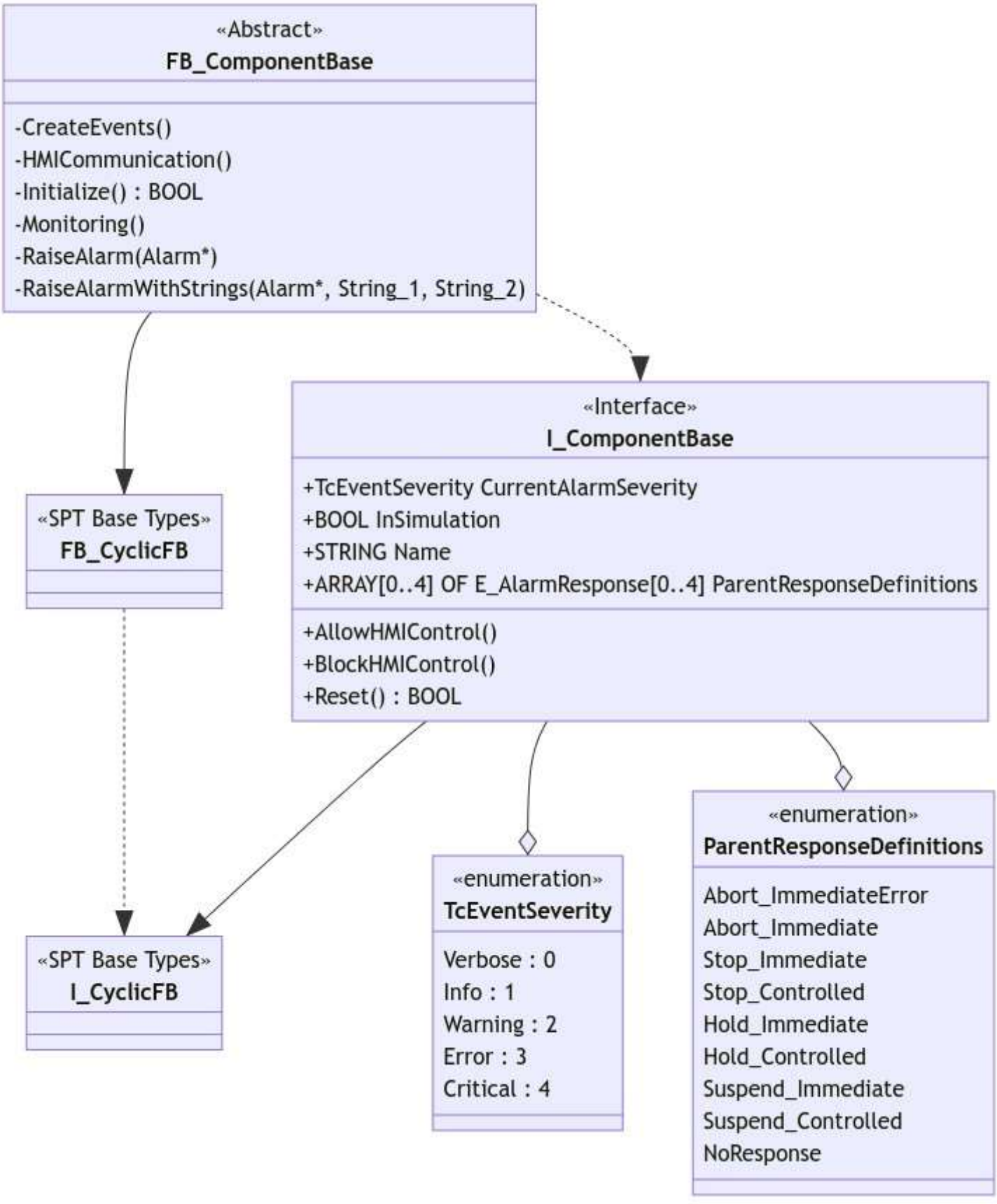
Last updated by | Nick Higgins | Sep 27, 2022 at 9:33 AM EDT

Contents

Overview

Basic structure and boilerplate code for components. Components can be used natively within PackML Equipment Modules or by themselves in a non-PackML based project.

Class Diagram


«Abstract»
FB_ComponentBase
-CreateEvents()
-HMICommunication()
-Initialize() : BOOL
-Monitoring()
-RaiseAlarm(Alarm*)
-RaiseAlarmWithStrings(Alarm*, String_1, String_2)
«Interface»
I_ComponentBase
+TcEventSeverity CurrentAlarmSeverity
+BOOL InSimulation
+STRING Name
+ARRAY[0..4] OF E_AlarmResponse[0..4] ParentResponseDefinitions
+AllowHMIControl()
+BlockHMIControl()
+Reset() : BOOL
«SPT Base Types»
FB_CyclicFB
«SPT Base Types»
I_CyclicFB
«enumeration»
TcEventSeverity
Verbose : 0
Info : 1
Warning : 2
Error : 3
Critical : 4
«enumeration»
ParentResponseDefinitions
Abort_ImmediateError
Abort_Immediate
Stop_Immediate
Stop_Controlled
Hold_Immediate
Hold_Controlled
Suspend_Immediate
Suspend_Controlled
NoResponse

Interfaces

I_ComponentBase

(extends `I_CyclicFB`)

Defines basic required functionality for component-level function blocks

Properties

Property	Type	Access	Description
CurrentAlarmSeverity	TcEventSeverity	RO	Highest severity of any currently active event(s)
InSimulation	BOOL	RW	Flag to indicate to component/parent that this component is running simulation code (defined per component)
Name	STRING	RW	Name of this component
ParentResponseDefinitions	ARRAY[0..4] OF E_AlarmResponse	RW	Defines how the parent of this component should react to each event severity

ParentResponseDefinitions

```
PROPERTY ParentResponseDefinitions : ARRAY[0..4] OF E_AlarmResponse
```

The expected response type of a component's parent can be specified. The actual response logic should be handled by the component's parent. The parent can observe `CurrentAlarmSeverity` and then decide what to do in response.

Example

```
ParentResponseDefinitions[TcEventSeverity.Verbose]  := E_AlarmResponse.NoResponse;
ParentResponseDefinitions[TcEventSeverity.Info]     := E_AlarmResponse.NoResponse;
ParentResponseDefinitions[TcEventSeverity.Warning]  := E_AlarmResponse.Suspend_Immediate;
ParentResponseDefinitions[TcEventSeverity.Error]    := E_AlarmResponse.Abort_ImmediateError;
ParentResponseDefinitions[TcEventSeverity.Critical] := E_AlarmResponse.Abort_ImmediateError;
```

Methods

Method	Return Type	Access	Description
AllowHMIControl	null	PUBLIC	Signal to this component that external functions via HMI should be allowed
BlockHMIControl	null	PUBLIC	Signal to this component that external functions via HMI should be blocked
Reset	BOOL	PUBLIC	Command this component to execute its fault reset routine

---

Function Blocks

FB_ComponentBase

(abstract, extends `FB_CyclicFB`, implements `I_ComponentBase`)

Contains property backers for all `I_ComponentBase` properties as well as basic housekeeping code for all component-level function blocks.

All `PROTECTED` methods can and usually should be overridden for component-specific functionality (HMI commands, event handling, etc.) and then called using `SUPER^.Method()`.

Methods

Method	Return Type	Access	Description
CreateEvents	null	PROTECTED	Initializes base component events
HMICommunication	null	PROTECTED	Handles HMI command requests
Initialize	BOOL	PROTECTED	Basic initialization routine
Monitoring	null	PROTECTED	Monitors component functions and fires events as required
RaiseAlarm	null	PROTECTED	Raises an event with no arguments
RaiseAlarmWithStrings	null	PROTECTED	Raises an event and also accepts string arguments for contextual information

CreateEvents()

`METHOD PROTECTED CreateEvents`

If a component has its own specific events defined, override this method to initialize them and then call `SUPER^.CreateEvents()` to initialize the base component events.

HMICommunication()

`METHOD PROTECTED HMICommunication`

Override and `SUPER^` call this method to implement component-specific HMI commands, status, configuration items. The base method uses a predefined local `HMICommandActive_Descendant` as an interlock to ensure only one command is fired at a time, whether it's a base command or a component-specific command. The base method also provides `HMICommandActive_Base` for you to use in your custom method extensions.

Example

```
SUPER^.HMICommunication();
HMICommand_MyComponent_RT(CLK := (ComponentBase_HMI.Status.HMIControlAvailable AND NOT HMICommandActive_Base)

//Signal to FB_ComponentBase that a command is active
HMICommandActive_Descendant := ComponentBase_HMI.Status.HMIControlAvailable AND (MyComponent_HMI.Command.MyCom

//Process Momentary HMI requests
IF HMICommand_MyComponent_RT.Q THEN
    IF MyComponent_HMI.Command.MyCommand THEN
        MyCommand();
    ELSIF MyComponent_HMI.Command.MyOtherCommand THEN
        MyOtherCommand();
    END_IF
END_IF

//Handle non-momentary commands such as jogging
IF ComponentBase_HMI.Status.HMIControlAvailable AND NOT HMICommand_MyComponent_RT.Q THEN
    IF MyComponent_HMI.Command.Jog THEN
        MyComponent.Jog();
    END_IF
END_IF

//Update HMI status info
MyComponent_HMI.Status.Red   := Red;
MyComponent_HMI.Status.Green := Green;
MyComponent_HMI.Status.Blue  := Blue;
```

Initialize()

`METHOD PROTECTED Initialize`

In cases where a custom component contains one or many other components within, override `Initialize()` and interlock the `SUPER^.Initalize()` call with the `InitComplete` property of your subcomponent(s). `CyclicLogic()` should also be overridden and the calls to subcomponents' `CyclicLogic()` placed within. Through a long and complicated call chain, this pattern will ensure subcomponent(s) have initialized completely before your custom component reports `InitComplete = TRUE`.

Example

```
METHOD PROTECTED FINAL Initialize : BOOL;

IF NOT (MySubcomponent1.InitComplete AND MySubcomponent2.InitComplete) THEN
    RETURN;
END_IF

IF SUPER^.Initialize() THEN
    Initialize := TRUE;
END_IF



METHOD PUBLIC FINAL CyclicLogic

MySubcomponent1.CyclicLogic();
MySubcomponent2.CyclicLogic();

//My custom component's code/state machine here

SUPER^.CyclicLogic();

_Busy  := MainState <> Idle AND MainState <> Error;
_Error := MainState = Error AND MainState <> Reset;
```

Monitoring()

```
METHOD PROTECTED Monitoring
```

Override this method to add event handling to your custom component.

Example

```
METHOD PROTECTED Monitoring

SUPER^.Monitoring();

//ErrorID 1 - MC_MoveVelocity
IF BasicAxis.MC_MoveVelocity.Error AND NOT BasicAxisAlarms[E_BasicAxis.MoveVelocityError].bRaised THEN
    RaiseAlarmWithStrings(Alarm := BasicAxisAlarms[E_BasicAxis.MoveVelocityError], UDINT_TO_STRING(BasicAxis.E
ELSIF NOT BasicAxis.MC_MoveVelocity.Error AND BasicAxisAlarms[E_BasicAxis.MoveVelocityError].bRaised THEN
    BasicAxisAlarms[E_BasicAxis.MoveVelocityError].Clear(0, 0);
END_IF

//ErrorID 2 - MC_MoveRelative
IF BasicAxis.MC_MoveRelative.Error AND NOT BasicAxisAlarms[E_BasicAxis.MoveRelativeError].bRaised THEN
    RaiseAlarmWithStrings(Alarm := BasicAxisAlarms[E_BasicAxis.MoveRelativeError], UDINT_TO_STRING(BasicAxis.M
ELSIF NOT BasicAxis.MC_MoveRelative.Error AND BasicAxisAlarms[E_BasicAxis.MoveRelativeError].bRaised THEN
    BasicAxisAlarms[E_BasicAxis.MoveRelativeError].Clear(0, 0);
END_IF

_CurrentAlarmSeverity := F_GetMaxSeverityRaised(Alarms := BasicAxisAlarms, CurrentSeverity := CurrentAlarmSeve
```

RaiseAlarm()

```
METHOD PROTECTED RaiseAlarm
VAR_IN_OUT
    Alarm : FB_TcAlarm;
END_VAR
```

Raises an alarm with no arguments

RaiseAlarmWithStrings()

```
METHOD PROTECTED RaiseAlarmWithStrings
VAR_IN_OUT
    Alarm : FB_TcAlarm;
END_VAR

VAR_INPUT
    String_1 : STRING;
    String_2 : STRING;
END_VAR
```

Raises an alarm with optional contextual information. In actuality the event being raised takes 3 string arguments, but the component's `Name` is automatically passed as the first argument.