SPT PackML Base

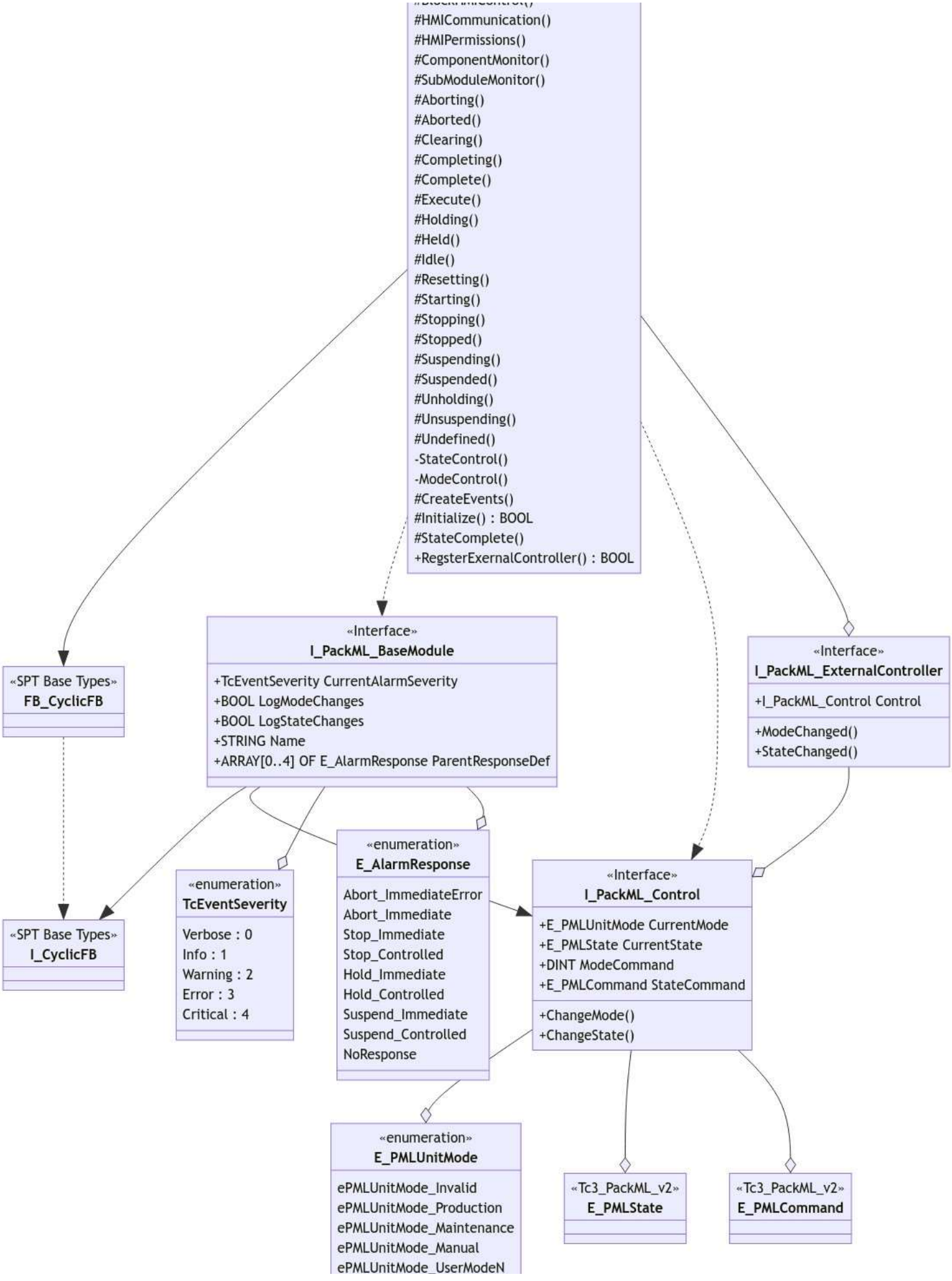
Last updated by | Nick Higgins | Sep 27, 2022 at 3:19 PM EDT

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


#HMICommunication()
#HMIPermissions()
#ComponentMonitor()
#SubModuleMonitor()
#Aborting()
#Aborted()
#Clearing()
#Completing()
#Complete()
#Execute()
#Holding()
#Held()
#Idle()
#Resetting()
#Starting()
#Stopping()
#Stopped()
#Suspending()
#Suspended()
#Unholding()
#Unsuspending()
#Undefined()
-StateControl()
-ModeControl()
#CreateEvents()
#Initialize() : BOOL
#StateComplete()
+RegsterExernalController() : BOOL
«SPT Base Types»
FB_CyclicFB
«Interface»
I_PackML_BaseModule
+TcEventSeverity CurrentAlarmSeverity
+BOOL LogModeChanges
+BOOL LogStateChanges
+STRING Name
+ARRAY[0..4] OF E_AlarmResponse ParentResponseDef
«Interface»
I_PackML_ExternalController
+I_PackML_Control Control
+ModeChanged()
+StateChanged()
«SPT Base Types»
I_CyclicFB
«enumeration»
TcEventSeverity
Verbose : 0
Info : 1
Warning : 2
Error : 3
Critical : 4
«enumeration»
E_AlarmResponse
Abort_ImmediateError
Abort_Immediate
Stop_Immediate
Stop_Controlled
Hold_Immediate
Hold_Controlled
Suspend_Immediate
Suspend_Controlled
NoResponse
«Interface»
I_PackML_Control
+E_PMLUnitMode CurrentMode
+E_PMLState CurrentState
+DINT ModeCommand
+E_PMLCommand StateCommand
+ChangeMode()
+ChangeState()
«enumeration»
E_PMLUnitMode
ePMLUnitMode_Invalid
ePMLUnitMode_Production
ePMLUnitMode_Maintenance
ePMLUnitMode_Manual
ePMLUnitMode_UserModeN
«Tc3_PackML_v2»
E_PMLState
«Tc3_PackML_v2»
E_PMLCommand

Interfaces

I_PackML_Control

Defines basic control and feedback mechanisms for control of PackML modules

Properties

Property	Type	Access	Description
CurrentMode	E_PMLUnitMode	RO	Current PackML mode
CurrentState	E_PMLState	RO	Current PackML state
ModeCommand	DINT	RW	Commanded PackML mode
StateCommand	E_PMLCommand	RW	Commanded PackML state

Notes

- You can request a mode/state change using the method call `ChangeState()` `ChangeMode()` or by setting the `StateCommand` `ModeCommand` properties--they both do the same thing

Methods

Method	Return Type	Access	Description
ChangeMode	null	PUBLIC	Request to enter a given mode
ChangeState	null	PUBLIC	Issue a specific PackML state command

I_PackML_BaseModule

(extends `I_CyclicFB, I_PackML_Control`)

Defines basic state & mode controls & status information required of all PackML modules (Machine Modules & Equipment Modules)

Properties

Property	Type	Access	Description
CurrentAlarmSeverity	TcEventSeverity	RO	Highest severity of any currently active event(s)
LogModeChanges	BOOL	RW	Enable/Disable logging of mode changes
LogStateChanges	BOOL	RW	Enable/Disable logging of PackML state changes
Name	STRING	RW	Name of this module
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;
```

I_PackML_ExternalController

(extends `I_CyclicFB, I_PackML_Control`)

Defines complete means of external control of a PackML module

Properties

Property	Type	Access	Description
Control	I_PackML_Control	WO	Pointer to control interface of module (see [Register](

Methods

Method	Return Type	Access	Description
ModeChanged	null	PUBLIC	Serves as a callback notification to external controllers that the module's mode has changed
StateChanged	null	PUBLIC	Serves as a callback notification to external controllers that the module's state has changed

Notes
See `FB_PackML_BaseModule.RegisterExternalController()` for more information.

Function Blocks

FB_PackML_BaseModule

(abstract, extends `FB_CyclicFB`, implements `I_PackML_BaseModule, I_PackML_Control`)

Methods

General

Method	Return Type	Access	Description
CreateEvents	null	PROTECTED	Initializes base component events
Initialize	BOOL	PROTECTED	Basic initialization routine
RegisterExternalController	BOOL	PUBLIC	Takes an instance of `I_PackML_ExternalController` and adds it to an internal collection of external controllers. External controllers are, for example, HMI or pushbutton aggregation function blocks.
StateComplete	null	PROTECTED	Signal to PackML state machine sequencer that this module can advance to the next state. This is normally called automatically by the state sequencer.

CreateEvents()
`METHOD PROTECTED CreateEvents`

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

Initialize()

```
METHOD PROTECTED Initialize
```

The base `Initialize()` method handles some basic tasks like setting pointers, counting the number of submodules/components, and most importantly ensuring any children are completely initialized before itself returning `TRUE`.

When creating an Equipment Module, override this method and implement your own routine. Many times it makes sense to use a state machine to make sure things are called in order and only as required. In fact, the base `Initialize()` does this using the predefined local `SequenceState` as the indexer. A second predefined local `DescendantSequenceState` can be used for indexing a state machine in your overridden `Initialize()`.

Example

```
METHOD PROTECTED FINAL Initialize : BOOL
VAR
    i : UDINT; //Generic iteration value
END_VAR

CASE DescendantSequenceState OF
    0:
        // Define the interfaces to the different subunits (Must take place before SUPER^.initialize call)
        FOR i := 1 TO SPT_PackMLBase.Parameters_PackML_Base.MAX_NO_OF_SUBMODULES DO
            ipSubModules[i] := ipSubModules_Init[i];
        END_FOR

        FOR i := 1 TO SPT_PackMLBase.Parameters_PackML_Base.MAX_NO_OF_COMPONENTS DO
            ipComponents[i] := ipComponents_Init[i];
        END_FOR

        LogModuleModeChanges    := FALSE;
        LogModuleStateChanges   := FALSE;
        DescendantSequenceState := DescendantSequenceState + 10;
    10:
        CustomModes(eMode                          := 4,
                    sName                          := 'My Custom Mode',
                    bDisableClearing               := FALSE,
                    bDisableStarting               := FALSE,
                    bDisableSuspended              := TRUE,
                    bDisableStopping               := FALSE,
                    bDisableAborting               := FALSE,
                    bDisableHolding                := TRUE,
                    bDisableHeld                   := TRUE,
                    bDisableUnholding              := TRUE,
                    bDisableSuspending             := TRUE,
                    bDisableUnsuspending           := TRUE,
                    bDisableResetting              := FALSE,
                    bDisableIdle                   := FALSE,
                    bDisableCompleting             := TRUE,
                    bDisableComplete               := TRUE,
                    bEnableUnitModeChangeStopped   := TRUE,
                    bEnableUnitModeChangeIdle      := TRUE,
                    bEnableUnitModeChangeSuspended := FALSE,
                    bEnableUnitModeChangeExecute   := TRUE,
                    bEnableUnitModeChangeAborted   := TRUE,
                    bEnableUnitModeChangeHeld      := FALSE,
                    bEnableUnitModeChangeComplete  := FALSE,
                    bError                         =>,
                    nErrorId                       =>);
    20:
        FOR i := 1 TO 4 DO
            ModeNames[i] := F_UnitModeToString(UDINT_TO_DINT(i));
        END_FOR

        DescendantSequenceState := DescendantSequenceState + 10;
    30:
        IF SUPER^.Initialize() THEN
            Initialize                 := TRUE;
        END_IF
END_CASE
```

RegisterExternalController()

```
METHOD FINAL RegisterExternalController : BOOL
VAR_INPUT
    Controller : I_PackML_ExternalController;
END_VAR
```

Alarm Handling

Method	Return Type	Access	Description
AbortImmediate	null	PROTECTED	Sequence to execute when a submodule or component faults. Ending PackML state will be `E_PMLState.ePMLState_Aborted`
AbortImmediateError	null	PROTECTED	Sequence to execute when a submodule or component faults. Ending PackML state will be `E_PMLState.ePMLState_Aborted` and an event will be raised
HoldControlled	null	PROTECTED	Sequence to execute when a submodule or component faults. Ending PackML state will be `E_PMLState.ePMLState_Held`
HoldImmediate	null	PROTECTED	Sequence to execute when a submodule or component faults. Ending PackML state will be `E_PMLState.ePMLState_Held`
StopControlled	null	PROTECTED	Sequence to execute when a submodule or component faults. Ending PackML state will be `E_PMLState.ePMLState_Stopped`
StopImmediate	null	PROTECTED	Sequence to execute when a submodule or component faults. Ending PackML state will be `E_PMLState.ePMLState_Stopped`
SuspendControlled	null	PROTECTED	Sequence to execute when a submodule or component faults. Ending PackML state will be `E_PMLState.ePMLState_Suspended`
SuspendImmediate	null	PROTECTED	Sequence to execute when a submodule or component faults. Ending PackML state will be `E_PMLState.ePMLState_Suspended`

Notes
The method which will be called on submodule or component fault depends on the severity of the event and the behavior defined by the subcomponent/component's `ParentResponseDefinitions` property.

AbortImmediateError()

```
METHOD PROTECTED AbortImmediateError
VAR_INPUT
    Name     : STRING;
    IsModule : BOOL;
END_VAR
```

`Name` will be included in the event text. If `IsModule` is `TRUE`, a SubModuleError is raised. If `FALSE` a ComponentError is raised.

HMI

Method	Return Type	Access	Description
AllowHMIControl	null	PROTECTED	Signal to this module that external functions via HMI should be allowed
BlockHMIControl	null	PROTECTED	Signal to this module that external functions via HMI should be blocked
HMICommunication	null	PROTECTED	Called cyclically; Handles HMI command requests
HMIPermissions	null	PROTECTED	Called cyclically; sets whether or not HMI commands are allowed based on current PackML mode

Notes

AllowHMIControl()

```
METHOD PROTECTED FINAL AllowHMIControl
VAR_INPUT
    ThisModuleOnly : BOOL; //! If TRUE, only the PackML module itself is controllable. If FALSE, all component
END_VAR
```

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_MyEquipmentModule_RT(CLK := (PackMLBaseModule_HMI.Status.HMIControlAvailable AND NOT HMICommandActi

//Signal to FB_PackMLBaseModule that a command is active
HMICommandActive_Descendant := PackMLBaseModule_HMI.Status.HMIControlAvailable AND (MyEquipmentModule_HMI.Comm

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

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

Monitoring

Method	Return Type	Access	Description
ComponentMonitor	null	PROTECTED	Called cyclically; checks components for faults and calls appropriate reaction method
SubModuleMonitor	null	PROTECTED	Called cyclically; checks submodules for faults and calls appropriate reaction method

Primary & Acting States

Method	Return Type	Access	Description
Aborted	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Aborted`
Complete	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Complete`
Execute	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Execute`
Held	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Held`
Idle	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Idle`
Stopped	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Stopped`
Suspended	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Suspended`
Undefined	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Undefined` . Should not normally be called.
Aborting	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Aborting`
Clearing	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Clearing`
Completing	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Completing`

Method	Return Type	Access	Description
Holding	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Holding`
Resetting	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Resetting`
Starting	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Starting`
Stopping	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Stopping`
Suspending	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Suspending`
Unholding	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Unholding`
Unsuspending	null	PROTECTED	Called cyclically according to PackML state; logic/sequence associated with PackML state `E_PMLState.ePMLState_Unsuspending`

Notes

The private method `StateControl` will automatically call the method that corresponds with the module's PackML state. These methods are typically overriden and replaced with customized machine logic.

Set `NoStateTasksToComplete` to `TRUE` to indicate to the state sequencer that there are things left to do before advancing states. Set `StateTasksComplete` when your machine logic is completed. Both of these variables are reinitialized when the state is advanced, so no additional management is necessary.

Call `SUPER^.` *State*() at the end of your method override so that the base method can take care of housekeeping.

Example

```
METHOD PROTECTED FINAL Clearing

CASE SequenceState OF
    0:
        NoStateTasksToComplete := FALSE;
        SequenceState          := SequenceState + 10;
    10:
        //Reset any alarms that get latched
        UnwindAlarms[E_Unwind.StartTimeout].Clear(0, 0);
        SequenceState := SequenceState + 10;
    20:
        // Enable here
        Axis.Enable();

        // Check enabled here
        IF Axis.Enabled THEN
            SequenceState := SequenceState + 10;
        END_IF
    30:
        StateTasksComplete := TRUE;
END_CASE

SUPER^.Clearing();
```