Easton And Potomac Helix and RF&P Diamond Interlock Code

```
import jarray
import jmri
import time
import java
import javax.swing
from java.util import Random
class EP_Interlocks(jmri.jmrit.automat.AbstractAutomaton) :
        def init(self):
                print "EP Interlocks Program initializing"
                self.LT1 = turnouts.provideTurnout("1") #hopkins spring semaphore
                self.LT2= turnouts.provideTurnout("2") #furnace semaphore
                self.LT3 = turnouts.provideTurnout("3") #hood semaphore
                self.LT1.setCommandedState(THROWN) #hopkins spring semaphore set to RED
                self.waitMsec(1000)
                self.LT2.setCommandedState(THROWN) #furnace semaphore set to RED
                self.waitMsec(1000)
                self.LT3.setCommandedState(THROWN) #hood semaphore set to RED
                self.waitMsec(1000)
                self.LS1 = sensors.provideSensor("1") #hopkins spring request switch
                self.LS2 = sensors.provideSensor("2") #furnace request switch
                self.LS3 = sensors.provideSensor("3") #hood request switch
                self.LS4 = sensors.provideSensor("4") #helxi IRDOTs and toggle
                self.LS5 = sensors.provideSensor("5") #helix BD4
                self.LS6 = sensors.provideSensor("6") #diamond east request switch
                self.LS7 = sensors.provideSensor("7") #diamond west request switch
                self.LS8 = sensors.provideSensor("8") #diamond IRDOT
                self.LS999 = sensors.provideSensor("999") #stop program screen switch
                print "clearing Hopkins Spring request switch"
                self.LS1.setKnownState(INACTIVE) #clear hopkins spring request switch
                self.waitMsec(1000)
                print "clearing Furnace request switch"
                self.LS2.setKnownState(INACTIVE) #clear furnace request switch
                self.waitMsec(1000)
                print "clearing Hood request switch"
                self.LS3.setKnownState(INACTIVE) # clear hood request switch
                self.waitMsec(1000)
                print "clearing diamond east request switch"
                self.LS6.setKnownState(INACTIVE) #clear diamond east request switch
                self.waitMsec(1000)
                print "clearing diamond west request switch"
```

```
self.LS7.setKnownState(INACTIVE) #clear diamond west request switch
self.waitMsec(1000)
self.LS4.setKnownState(INACTIVE) #clear the helix irdot sensor (do not usde in production)
self.waitMsec(1000)
self.LS5.setKnownState(INACTIVE) #clear the helix sensor (do not use in production)
self.waitMsec(1000)
self.LS8.setKnownState(INACTIVE) #clear diamond irdot sensor (do not use in production)
self.waitMsec(1000)
self.LH257 = signals.getSignalHead("LH257") #hopkins spring LED
self.LH259 = signals.getSignalHead("LH259") #furnace LED
self.LH265 = signals.getSignalHead("LH265") #hood LED
self.LH275 = signals.getSignalHead("LH275") #diamond east LED
self.LH273 = signals.getSignalHead("LH273") #diamond west LED
self.LH281 = signals.getSignalHead("LH281") #rfp north signal
self.LH283 = signals.getSignalHead("LH283") #rfp south signal
self.LH289 = signals.getSignalHead("LH289") #ep east signal
self.LH291 = signals.getSignalHead("LH291") #ep west signal
print "clearing Hopkins Spring LED"
self.LH257.appearance = DARK #turn hopkins spring LED off
self.waitMsec(1000)
print "clearing Furnace LED"
self.LH259.appearance = DARK #turn furnace LED off
self.waitMsec(1000)
print "clearing Hood LED"
self.LH265.appearance = DARK #turn hood LED off
self.waitMsec(1000)
print "clearing diamond east LED"
self.LH275.appearance = DARK #turn diamond east LED off
self.waitMsec(1000)
print "clearing diamond west LED"
self.LH273.appearance = DARK #turn diamond west LED off
self.waitMsec(1000)
print "Setting RFP north to RED"
self.LH281.appearance = GREEN
self.waitMsec(1000)
self.LH281.appearance = RED #set rfp north signal to RED
self.waitMsec(1000)
print "Setting RFP south to RED"
self.LH283.appearance = GREEN
self.waitMsec(1000)
self.LH283.appearance = RED #set rfp south signal to RED
self.waitMsec(1000)
```

```
        print "Setting EP east to RED"
        self.LH289.appearance = GREEN
        self.waitMsec(1000)
        self.LH289.appearance = RED #set ep east signal to RED
        self.waitMsec(1000)
        print "Setting EP west to RED"
        self.LH291.appearance = GREEN
        self.waitMsec(1000)
        self.LH291.appearance = RED #set ep west signal to RED
        self.waitMsec(1000)
        self.AHS = 0 #hopkins spring access request type 1=add, 2=remove, preset to 0
        self.AF = 0 #furnace access request type 1=add, 2=remove, preset to 0
        self.AH = 0 #hood access request type 1=add, 2=remove, preset to 0
        self.N = 0
        self.Helix_Queue = [0,0,0]
        self.DE = time.clock()
        self.RN = Random()
        self.signal_aspects = {1:50, 2:75, 3:90, 4:100} #rfp signal aspects
        self.DA = 0
        self.cycle_range = {1:50, 2:70, 3:85, 4:95, 5:100} #random number ranges
        self.cycle_delay = {1:30, 2:60, 3:90, 4:120, 5:150} #time delay in seconds
        self.DT = 0
        #the following are previous state indicators for the manual pushbuttons on the layout
        self.LS1_prev = self.LS1.knownState
        self.LS2_prev = self.LS2.knownState
        self.LS3_prev = self.LS3.knownState
        self.LS6_prev = self.LS6.knownState
        self.LS7_prev = self.LS7.knownState
        #print "signal aspects = 50-RR, 75-GR, 90-RG, 100-GG"
        #print "signal range = 50, 70, 85, 95, 100"
        #print "signal delay = 30, 60, 90, 120, 150"
        print "clearing program stop switch"
        self.LS999.setKnownState(INACTIVE) #clear program stop switch
        self.waitMsec(1000)
        print "PROGRAM RUNNING"
        return

def handle(self):
        def check_button_state_change(self):
        #check all of the pushbuttons on the layout for a state change and rest previous if not equal
                if self.LS1_prev <> self.LS1.knownState:
                        self.LS1_prev = 0
```

```
        if self.LS2_prev <> self.LS2.knownState:
                self.LS2_prev = 0
        if self.LS3_prev <> self.LS3.knownState:
                self.LS3_prev = 0
        if self.LS6_prev<> self.LS6.knownState:
                self.LS6_prev = 0
        if self.LS7_prev <> self.LS7.knownState:
                self.LS7_prev = 0
        return
def helix(self):
        def check_helix_inputs(self):
                if self.LS1.knownState==ACTIVE \
                and self.LS1.knownState <> self.LS1_prev: #hopkins spring switch pressed
                        self.LS1_prev = self.LS1.knownState #copy current state to previous state
                        self.waitMsec(10)
                        if self.LT1.commandedState==CLOSED: #hopkins spring semaphore GREEN
                                self.LT1.setCommandedState(THROWN) #hopkins spring semaphore RED
                                self.waitMsec(1000)
                        else:
                                if self.LH257.appearance==DARK: #hopkins spring LED off, set request
                                        self.AHS = 1
                                elif self.LH257.appearance==RED: #hopkins spring LED on, remove request
                                        self.AHS = 2
                if self.LS2.knownState==ACTIVE \
                and self.LS2.knownState <> self.LS2_prev: #furnace switch pressed
                        self.LS2_prev = self.LS2.knownState #copy current state to previous state
                        self.waitMsec(10)
                        if self.LT2.commandedState==CLOSED: #furnace semaphore GREEN
                                self.LT2.setCommandedState(THROWN) #furnace semaphore RED
                                self.waitMsec(1000)
                        else:
                                if self.LH259.appearance==DARK: #furnace LED off, set request
                                        self.AF = 1
                                elif self.LH259.appearance==RED: #furnace LED on, remove request
                                        self.AF = 2
                if self.LS3.knownState==ACTIVE \
                and self.LS3.knownState <> self.LS3_prev: #hood switch pressed
                        self.LS3_prev = self.LS3.knownState #copy current state to previous state
                        self.waitMsec(10)
                        if self.LT3.commandedState==CLOSED: #hood semaphore GREEN
                                self.LT3.setCommandedState(THROWN) #hood semaphore RED
                                self.waitMsec(1000)
```

```
                else:
                        if self.LH265.appearance==DARK: #hood LED off, set request
                                self.AH = 1
                        elif self.LH265.appearance==RED: #hood LED on, remove request
                                self.AH = 2
        return
def update_queue(self):
        if self.AHS==1: #accress request from hopkins spring needs to be added
                self.Helix_Queue[self.N] = 1
                if self.N < 2:
                        self.N = self.N + 1
                self.AHS = 0
                self.LH257.appearance = RED
                self.waitMsec(10)
        if self.AHS==2: #access request from hopkins spring needs to be removed
                if self.Helix_Queue[0]==1:
                        self.Helix_Queue[0] = 0
                if self.Helix_Queue[1]==1:
                        self.Helix_Queue[1] = 0
                if self.Helix_Queue[2]==1:
                        self.Helix_Queue[2] = 0
                self.AHS = 0
                self.LH257.appearance = DARK
                self.waitMsec(10)
        #============================================================
        if self.AF==1: #accress request from furnace needs to be added
                self.Helix_Queue[self.N] = 2
                if self.N < 2:
                        self.N = self.N + 1
                self.AF = 0
                self.LH259.appearance = RED
                self.waitMsec(10)
        if self.AF==2: #access request from furnace needs to be removed
                if self.Helix_Queue[0]==2:
                        self.Helix_Queue[0] = 0
                if self.Helix_Queue[1]==2:
                        self.Helix_Queue[1] = 0
                if self.Helix_Queue[2]==2:
                        self.Helix_Queue[2] = 0
                self.AF = 0
                self.LH259.appearance = DARK
                self.waitMsec(10)
```

```
        #=============================================================
        if self.AH==1: #accress request from hood needs to be added
                self.Helix_Queue[self.N] = 3
                if self.N < 2:
                        self.N = self.N + 1
                self.AH = 0
                self.LH265.appearance = RED
                self.waitMsec(10)
        if self.AH==2: #access request from hood needs to be removed
                if self.Helix_Queue[0]==3:
                        self.Helix_Queue[0] = 0
                if self.Helix_Queue[1]==3:
                        self.Helix_Queue[1] = 0
                if self.Helix_Queue[2]==3:
                        self.Helix_Queue[2] = 0
                self.AH = 0
                self.LH265.appearance = DARK
                self.waitMsec(10)
        #=================================================================
        #compress empty queue entries
        if self.Helix_Queue[0]==0:
                self.Helix_Queue[0] = self.Helix_Queue[1]
                self.Helix_Queue[1] = self.Helix_Queue[2]
                self.Helix_Queue[2] = 0
        if self.Helix_Queue[1]==0:
                self.Helix_Queue[1] = self.Helix_Queue[2]
                self.Helix_Queue[2] = 0
        #=================================================================
        #set queue index to first non empty location or begining of queue
        if self.Helix_Queue[2]==0:
                self.N = 2
        if self.Helix_Queue[1]==0:
                self.N = 1
        if self.Helix_Queue[0]==0:
                self.N = 0
        return
def set_semaphores(self):
#check helix block detectors and if active set semaphore to red if green
        if self.LS4.knownState==ACTIVE \
        and self.LT1.commandedState==CLOSED:
                self.LT1.setCommandedState(THROWN)
                self.waitMsec(1000)
```

```
if self.LS4.knownState==ACTIVE \
and self.LT2.commandedState==CLOSED:
        self.LT2.setCommandedState(THROWN)
        self.waitMsec(1000)
if self.LS4.knownState==ACTIVE \
and self.LT3.commandedState==CLOSED:
        self.LT3.setCommandedState(THROWN)
        self.waitMsec(1000)
if self.LS5.knownState==ACTIVE \
and self.LT1.commandedState==CLOSED:
        self.LT1.setCommandedState(THROWN)
        self.waitMsec(1000)
if self.LS5.knownState==ACTIVE \
and self.LT2.commandedState==CLOSED:
        self.LT2.setCommandedState(THROWN)
        self.waitMsec(1000)
if self.LS5.knownState==ACTIVE \
and self.LT3.commandedState==CLOSED:
        self.LT3.setCommandedState(THROWN)
        self.waitMsec(1000)
if self.LS4.knownState==INACTIVE \
and self.LS5.knownState==INACTIVE:
#helix not occupied, now check to see if there are any Green semaphores
        if self.LT1.commandedState==THROWN \
        and self.LT2.commandedState==THROWN \
        and self.LT3.commandedState==THROWN:
        #all semaphores are RED process next request in queue
                if self.Helix_Queue[0]==1:
                        self.LT1.setCommandedState(CLOSED)
                        self.waitMsec(1000)
                        self.LT2.setCommandedState(THROWN)
                        self.waitMsec(1000)
                        self.LT3.setCommandedState(THROWN)
                        self.waitMsec(1000)
                        self.LH257.appearance = DARK
                        self.waitMsec(10)
                if self.Helix_Queue[0]==2:
                        self.LT1.setCommandedState(THROWN)
                        self.waitMsec(1000)
                        self.LT2.setCommandedState(CLOSED)
                        self.waitMsec(1000)
                        self.LT3.setCommandedState(THROWN)
```

```
                                        self.waitMsec(1000)
                                        self.LH259.appearance = DARK
                                        self.waitMsec(10)
                                if self.Helix_Queue[0]==3:
                                        self.LT1.setCommandedState(THROWN)
                                        self.waitMsec(1000)
                                        self.LT2.setCommandedState(THROWN)
                                        self.waitMsec(1000)
                                        self.LT3.setCommandedState(CLOSED)
                                        self.waitMsec(1000)
                                        self.LH265.appearance = DARK
                                        self.waitMsec(10)
                                self.Helix_Queue[0] = self.Helix_Queue[1]
                                self.Helix_Queue[1] = self.Helix_Queue[2]
                                self.Helix_Queue[2] = 0
                                #=================================================================
                                #set queue index to first non empty location or begining of queue
                                if self.Helix_Queue[2]==0:
                                        self.N = 2
                                if self.Helix_Queue[1]==0:
                                        self.N = 1
                                if self.Helix_Queue[0]==0:
                                        self.N = 0
                        return
                check_helix_inputs(self)
                update_queue(self)
                set_semaphores(self)
                return

        def diamond(self):
                def check_diamond_inputs(self):
                        if self.LS6.knownState==ACTIVE \
                        and self.LS6.knownState <> self.LS6_prev \
                        and self.LH291.appearance==RED \
                        and self.LH273.appearance==DARK:
                        #east access request switch pressed and no west pending request
                                self.LS6_prev = self.LS6.knownState #copy current state to previous state
                                self.waitMsec(10)
                                if self.LH275.appearance==DARK \
                                and self.LH289.appearance==RED:
                                #no east access request pending or granted, set request
                                        self.LH275.appearance = RED #turn on east request LED
```

```
                        self.waitMsec(10)
                else: #east request pending, remove request
                        self.LH289.appearance = RED #set east ep signal back to RED
                        self.waitMsec(10)
                        self.LH275.appearance = DARK #turn off east request LED
                        self.waitMsec(10)
        else:
                if self.LS7.knownState==ACTIVE \
                and self.LS7.knownState <> self.LS7_prev \
                and self.LH289.appearance==RED \
                and self.LH275.appearance==DARK:
                #west access request switch pressed and no east pending request
                        self.LS7_prev = self.LS7.knownState #copy current state to previous state
                        self.waitMsec(10)
                        if self.LH273.appearance==DARK \
                        and self.LH291.appearance==RED:
                        #no west access request pending or granted, set request
                                self.LH273.appearance = RED
                                self.waitMsec(10)
                        else: #west request pending, remove request
                                self.LH291.appearance = RED #set west ep signal back to RED
                                self.waitMsec(10)
                                self.LH273.appearance = DARK #turn off west request LED
                                self.waitMsec(10)
        return
def set_rfp_signals(self):
        def process_diamond_cycle(self):
                def generate_diamond_cycle(self):
                        self.DR = self.RN.nextInt(100)
                        print "signal RN = ", self.DR
                        if self.DR <= self.signal_aspects[1]:
                                self.DA = 1
                        if self.DR > self.signal_aspects[1] \
                        and self.DR <= self.signal_aspects[2]:
                                self.DA = 2
                        if self.DR > self.signal_aspects[2] \
                        and self.DR <= self.signal_aspects[3]:
                                self.DA = 3
                        if self.DR >= self.signal_aspects[3] \
                        and self.DR < self.signal_aspects[4]:
                                self.DA = 4
                        self.DR = self.RN.nextInt(100)
```

```
                print "delay RN = ", self.DR
                if self.DR <= self.cycle_range[1]:
                        self.DT = self.cycle_delay[1]
                if self.DR > self.cycle_range[1] \
                and self.DR <= self.cycle_range[2]:
                        self.DT = self.cycle_delay[2]
                if self.DR > self.cycle_range[2] \
                and self.DR <= self.cycle_range[3]:
                        self.DT = self.cycle_delay[3]
                if self.DR > self.cycle_range[3] \
                and self.DR <= self.cycle_range[4]:
                        self.DT = self.cycle_delay[4]
                if self.DR > self.cycle_range[4] \
                and self.DR <= self.cycle_range[5]:
                        self.DT = self.cycle_delay[5]
                self.LH281.appearance = RED #reset rfp signal to RED
                self.waitMsec(1000)
                self.LH283.appearance = RED #reset rfp signal to RED
                self.waitMsec(1000) #wait 1 second
                if self.DA==2 \
                or self.DA==4:
                        self.LH283.appearance = GREEN #rfp south
                        self.waitMsec(1000)
                if self.DA==3 \
                or self.DA==4:
                        self.LH281.appearance = GREEN #rfp north
                        self.waitMsec(1000)
                self.DE = time.clock() + self.DT
                return
        if time.clock()>=self.DE:
                generate_diamond_cycle(self)
        return
if self.LS8.knownState==ACTIVE \
or self.LH289.appearance==GREEN \
or self.LH291.appearance==GREEN:
#diamond occupied or access given to ep, set rfp signals to RED
        if self.LH281.appearance==GREEN:
                self.LH281.appearance = RED #rfp north signal
                self.waitMsec(1000)
        if self.LH283.appearance==GREEN:
                self.LH283.appearance = RED #rfp south signal
                self.waitMsec(1000)
```

```
        else:
                process_diamond_cycle(self)
        return
def set_ep_signals(self):
        if self.LS8.knownState==ACTIVE:
        #diamond occupied set all ep signals to RED and clear all requests
                if self.LH289.appearance==GREEN:
                        self.LH289.appearance = RED #set ep east signal RED
                        self.waitMsec(1000)
                if self.LH291.appearance==GREEN:
                        self.LH291.appearance = RED #set ep west signal RED
                        self.waitMsec(1000)
                if self.LH275.appearance==RED:
                        self.LH275.appearance = DARK #turn off ep east LED
                        self.waitMsec(1000)
                if self.LH273.appearance==RED:
                        self.LH273.appearance = DARK #turn off ep west LED
                        self.waitMsec(1000)
        if self.LH283.appearance==GREEN \
        or self.LH281.appearance==GREEN:
        #an rfp signal is GREEN, set ep signal to RED
                if self.LH289.appearance==GREEN:
                        self.LH289.appearance = RED #set ep east signal RED
                        self.waitMsec(1000)
                if self.LH291.appearance==GREEN:
                        self.LH291.appearance = RED #set ep west signal RED
                        self.waitMsec(1000)
        if self.LS8.knownState==INACTIVE \
        and self.LH283.appearance==RED \
        and self.LH281.appearance==RED:
        #all clear process ep diamond request
                if self.LH275.appearance==RED: #give east access to diamond
                        self.LH289.appearance = GREEN #ep east signal to GREEN
                        self.waitMsec(1000)
                        self.LH275.appearance = DARK #ep east LED off
                        self.waitMsec(1000)
                else:
                        if self.LH273.appearance==RED: #give west access to diamond
                                self.LH291.appearance = GREEN #ep west signal to GREEN
                                self.waitMsec(1000)
                                self.LH273.appearance = DARK #ep west LED off
                                self.waitMsec(1000)
```

```
                    return
                check_diamond_inputs(self)
                set_rfp_signals(self)
                set_ep_signals(self)
                return

        if self.LS999.knownState==INACTIVE:
                check_button_state_change(self)
                helix(self)
                diamond(self)
                return 1
        else:
                print "program terminating"
                return 0

EP_Interlocks().start()
```