
SelfCheckout

**Report on Configuration
DefaultConfig**

PACKAGES

Default

EVENTS:

check
evActivate
evFinishTransaction
evIdle
evNext

GLOBALS:

Relations:

Screen

Composition of Display, Multiplicity of 1, Uni-directional

item

Composition of Food, Multiplicity of *, Uni-directional

Instantiated Relations:

itsFood

of Screen with item

itsDisplay

of item with Screen

CLASSES:

bagSensor

This is a scale that validates that the item scanned weighs the same as the item placed in the bag. If the weight is different, an error will display. This is just a method to check if a user is actually putting the right item into the bag. BagSensor also makes sure that a user doesn't remove the bag unless it is full until he has fully paid for the purchases.

Relations:

itsMyPicSerial

Association with myPicSerial, Multiplicity of 1, Bi-directional

Operations:

finishPayment

Overridden Properties

Subjects:

CPP_CG

Metaclasses:

Operation

Properties:

Kind: common

Inline: none

Primitive-operation , Public, Return type is void

isFull

Overridden Properties

Subjects:

CPP_CG

Metaclasses:

Operation

Properties:

Kind: common

Inline: none

Primitive-operation , Public, Return type is OMBoolean

Attributes:

maxWeight

Type of int, Public

weight

Type of int, Public

Display

This is the display screen for the computer. It will display what items have been put into the bag and what the current total is. This screen will tell you when it is ready for the next item or prompt you to make payment once you are done. You can add items, or delete items already there.

Overridden Properties

Subjects:

CG

Metaclasses:

Class

Properties:

Concurrency: active

Relations:

itsPayment

Association with payment, Multiplicity of 1, Bi-directional

itsFood

Association with Food, Multiplicity of 1, Bi-directional

Operations:

check

Event

evActivate

Event

evFinishTransaction

Event

evIdle

Event

evNext

Event

Attributes:

count

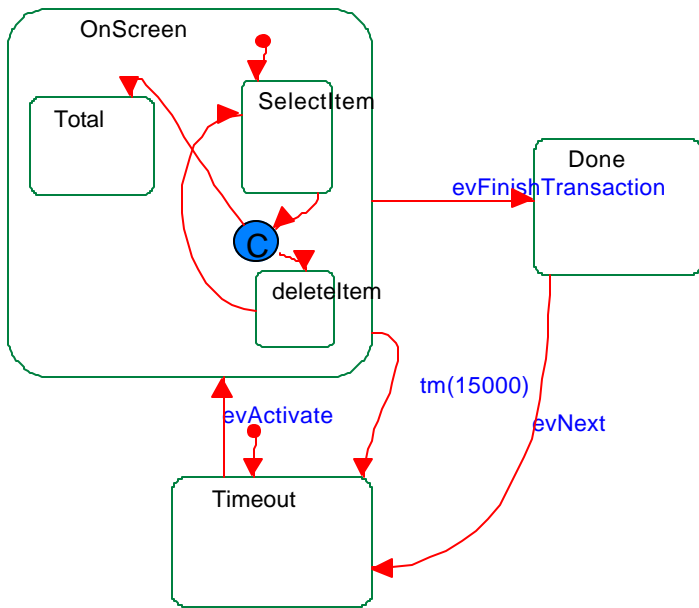
Type of int, Public, Initial Value: 0

nextItemReady

Ready or not for the next item

Type of OMBoolean, Public

Statechart



ROOT

Or-state

Substates:

Done

OnScreen

Timeout

Default Transition

Target:

Timeout

Done

Or-state

Out Transition

evNext

Target:

Timeout

OnScreen

Or-state

Substates:

deleteItem

SelectItem

Total

Out Transition

evFinishTransaction

Target:

Done

Out Transition

tm(15000)

Target:

Timeout

Default Transition

Target:

SelectItem

deleteItem

Or-state

Out Transition

Target:

SelectItem

SelectItem

Or-state

Out Transition

Condition Connector

Branches:

Target:

Total

Target:

deleteltem

Total

Or-state

Timeout

Or-state

Out Transition

evActivate

Target:

OnScreen

Food

Food is the item that the user is purchasing. There is a fixed price and weight associate with each item. A button on the picboard is connected to each item.

Relations:

itsMyPicSerial

Association with myPicSerial, Multiplicity of 1, Bi-directional

itsDisplay

Association with Display, Multiplicity of 1, Bi-directional

Attributes:

price

Type of long double, Public

quantity

Type of int, Public

weight

Type of int, Public

myPicSerial

This class is an interface for the user to select what items it is purchasing and add to the total amount owed. Each item is associated with a button which has a fixed name, price, and weight.

Relations:

itsFood

Association with Food, Multiplicity of 1, Bi-directional

itsPayment

Association with payment, Multiplicity of 1, Bi-directional

itsBagSensor

Association with bagSensor, Multiplicity of 1, Bi-directional

Superclasses:

PICserial

Public

Attributes:

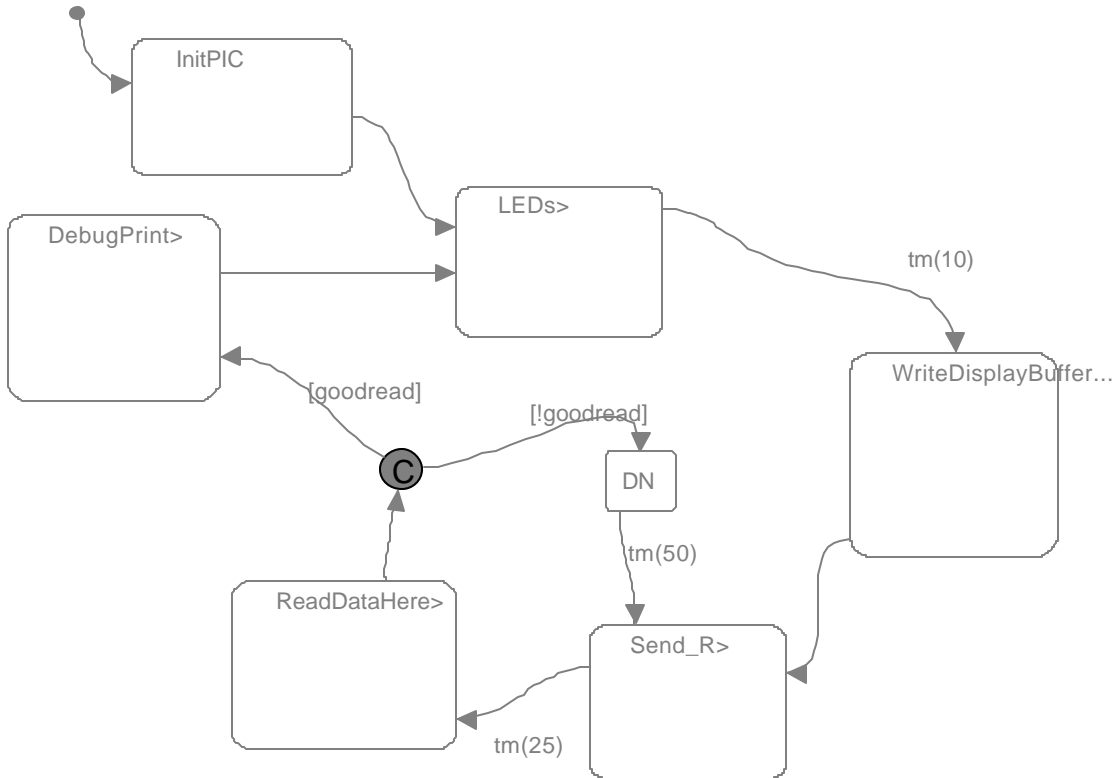
keyPressed

Activates the pic board 0-9, "." "enter" and "chs" buttons

Type of int, Public, Initial Value: int %s[] =

0x0002,0x4000,0x2000,0x1000,0x0400,0x0200,0x0100,0x0040,0x0020,0x0010,0x0800,0x0004,0x0001;

Statechart



ROOT

Or-state

Inherited

Substates:

DebugPrint

DN

InitPIC

LEDs

ReadDataHere

Send_R

WriteDisplayBuffer

Default Transition

Inherited

Target:

InitPIC

DebugPrint

Or-state

Inherited

EntryAction

```
//cout << " keys " << setbase(16) << setfill('0') << setw(8) << getAllKeyData() ;
```

```
//cout << " a0 " << (int)getAnalogValues(0) ;
```

```

//cout << " a1 " << (int)getAnalogValues(1);
//cout << " a2 " << (int)getAnalogValues(2);
//cout << endl;

//cout << setbase(16) << setfill('0') << setw(8) << getAllKeyData() << " " <<
(int)getAnalogValues(0) << " " << (int)getAnalogValues(1) << " " << (int)getAnalogValues(2)
<< endl;

//for (int i=0;i<5;i++)
//{ cout << " " << setbase(16) << (int) readBuffer[i]; }
//cout << endl;

```

Out Transition

Inherited

Target:
LEDs

DN

Or-state

Inherited

Out Transition

Inherited

tm(50)

Target:
Send_R

InitPIC

Or-state

Inherited

Out Transition

Inherited

Target:
LEDs

LEDs

Or-state

Inherited

EntryAction

writeLeds();

ExitAction
//leds++;

Out Transition

Inherited

tm(10)

Target:
WriteDisplayBuffer

ReadDataHere

Or-state

Inherited

EntryAction

goodread = ReadData();

Out Transition

Inherited

Condition Connector

Branches:

[goodread]

Target:

DebugPrint

[!goodread]

Target:

DN

Send_R

Or-state

Inherited

EntryAction

```
ControlChar[0] = 'R';
```

```
write(fd, ControlChar, 1); //write an 'R' to the PIC board to read the buffers.
```

Out Transition

Inherited

```
tm(25)
```

Target:

ReadDataHere

WriteDisplayBuffer

Nested Statechart

Or-state

Inherited

EntryAction

```
//for(int i=0;i<32;i++) Display[i]=leds;
```

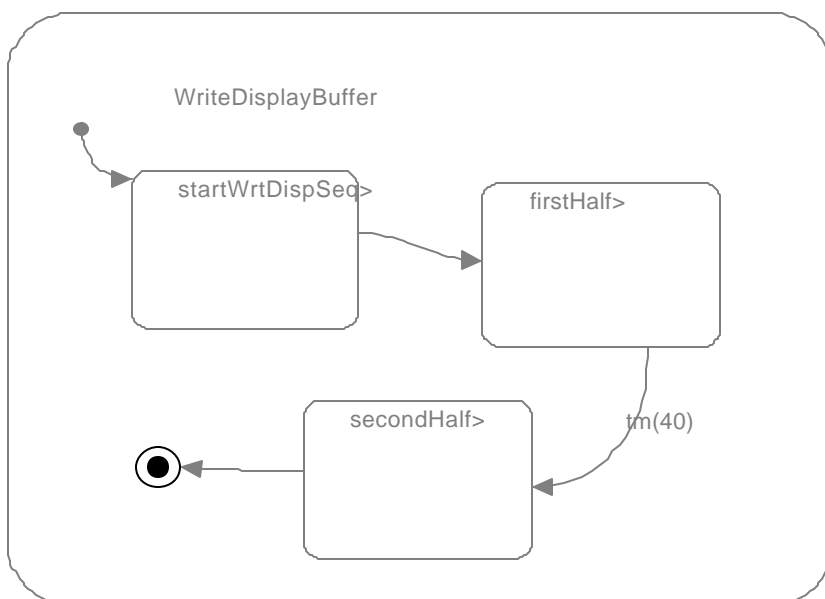
Out Transition

Inherited

Target:

Send_R

Statechart of WriteDisplayBuffer



ROOT

Or-state

Inherited

Substates:

WriteDisplayBuffer

WriteDisplayBuffer

Or-state

Inherited

Substates:

firstHalf

secondHalf

startWrtDispSeq

state_0

Default Transition

Inherited

Target:

startWrtDispSeq

firstHalf

Or-state

Inherited

EntryAction

```
write(fd, Display, 16); //write out 16 characters.
```

Out Transition

Inherited

```
tm(40)
```

Target:

secondHalf

secondHalf

Or-state

Inherited

EntryAction

```
write(fd, &Display[16], 16); //write out 16 characters.
```

Out Transition

Inherited

Target:

state_0

startWrtDispSeq

Or-state

Inherited

EntryAction

```
ControlChar[0] = 'B';
```

```
write(fd, ControlChar, 1); //write an 'B' to the PIC board to read the buffers.
```

Out Transition

Inherited

Target:

firstHalf

state_0

Local Termination State

payment

Allows a customer to have the option to pay by credit card or cash. If credit card is chosen, the user inputs the card number to verify that the number of digits match.

Basically that just checks the length of the numbers put in. The payment must match the amount owed.

Relations:

itsDisplay

Association with Display, Multiplicity of 1, Bi-directional

itsMyPicSerial

Association with myPicSerial, Multiplicity of 1, Bi-directional

Operations:

isAmountCorrect

Verifies if amount paid is equal to amount owed

Overridden Properties

Subjects:

CPP_CG

Metaclasses:

Operation

Properties:

Kind: common

Inline: none

Primitive-operation , Public, Return type is OMBoolean

isCardValid

Overridden Properties

Subjects:

CPP_CG

Metaclasses:

Operation

Properties:

Kind: common

Inline: none

Primitive-operation , Public, Return type is OMBoolean

Attributes:

Amount

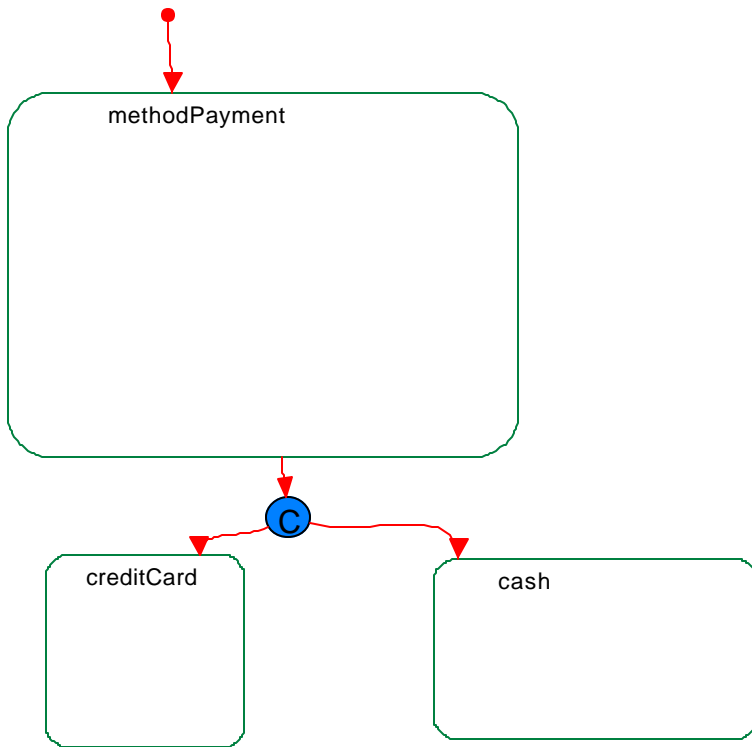
Type of int, Public

paymentMethod

how a customer wishes to pay. 1 = cash; 2 = credit card

Type of int, Public

Statechart



ROOT

Or-state

Substates:

cash

creditCard

methodPayment

Default Transition

Target:

methodPayment

cash

Or-state

creditCard

Or-state

methodPayment

Or-state

Out Transition

Condition Connector

Branches:

Target:

creditCard

Target:

cash

PICserial

Overridden Properties

Subjects:

CG

Metaclasses:

Class

Properties:

Concurrency: active

Subclasses:

myPicSerial

Operations:

getAllKeyData

*Gets status/value of the Keypad buttons.
Status is returned in the upper 2 bytes.
Values are returned in the bottom 2 bytes.
The parameter COSreset will*

Primitive-operation , Public, Return type is int

Args:

int COSreset = 0xFFFF

Body

```
union keydata {  
    long temp;  
    char t4[4];  
} KeyData;  
KeyData.t4[3]=getKeypadStatus(0,(char)((COSreset>>8)& 0xFF));  
KeyData.t4[2]=getKeypadStatus(1,(char)((COSreset)& 0xFF));  
KeyData.t4[1]=getKeypadValues(0);  
KeyData.t4[0]=getKeypadValues(1);  
  
return KeyData.temp;
```

getAnalogValues

Returns the value of each of the 3 analog values on the PIC board.

Overridden Properties

Subjects:

CPP_CG

Metaclasses:

Operation

Properties:

Kind: common

Inline: none

Primitive-operation , Public, Return type is 'unsigned char'

Args:

'int' i1

Constant

Body

```
i1= min(max(i1,0),2);  
return Analog[i1];
```

getKeypadStatus

Gets value of the Keypad character status.

Primitive-operation , Protected, Return type is 'unsigned char'

Args:

int i1

char COSreset

Which bits get reset when a status read is performed.

Body

```
i1= min(max(i1,0),1);
```

```

char temp = KeyStatus[i1];
KeyStatus[i1] &= (COSreset^0xFF);
return temp;

```

getKeypadValues

Gets value of the Keypad character.

Primitive-operation , Protected, Return type is 'unsigned char'

Args:

'int' i1

Body

```

i1= min(max(i1,0),1);
char temp = Keypad[i1];
Keypad[i1] &= 0x7f;
return temp;

```

PICserial

Constructor , Public

Args:

int ComPort = 1

This is the number of the communications port of this object.

Initializers

leds(0)

Body

```

if(ComPort==1)fd = open("/tyCo/0",O_RDWR,0644);
if(ComPort==2)fd = open("/tyCo/1",O_RDWR,0644);
io_status = ioctl(fd,FIOBAUDRATE,2400);
//cout << "ioctl status for " << fd << " = " << io_status << endl;
for(int i=0;i<32;i++) Display[i]=0;
Keypad[0]=0;
Keypad[1]=0;
KeyStatus[0]=0;
KeyStatus[1]=0;

```

ReadData

5 characters are read from the PIC board.

The first 2 are the Keypad and the last 3 are the analog pots.

A check is made to determine if a COS has occurred in any of the Keypad values.

TheKeypadStatus bits are set if there has been a change of state.

Primitive-operation , Protected, Return type is OMBoolean

Body

```

io_status = ioctl (fd, FIONREAD, (int)readBuffer);
if(readBuffer[0]== 5) {
    read(fd, readBuffer, 5); //read in 5 characters.

    KeyStatus[0] |= (readBuffer[0] ^ Keypad[0]);
    Keypad[0]=readBuffer[0];

    KeyStatus[1] |= (readBuffer[1] ^ Keypad[1]);
    Keypad[1]=readBuffer[1];

```

```

        Analog[0]= readBuffer[2];
        Analog[1]= readBuffer[3];
        Analog[2]= readBuffer[4];

        //GEN(evGoodReadPIC_Serial);
        return(true);
    }
    else
    {
        cout << "receive error in PIC_Serial" << endl;
        io_status = ioctl (fd, FIOFLUSH, 0);
        //GEN(evRetryPIC_SerialRead);
        return(false);
    }
}

```

setDisplayChar

Allows the user to set each element of the 32 character Display on the PIC board.

Primitive-operation , Public, Return type is 'void'

Args:

'int' i1

'char' p_Display

Body

```

i1= min(max(i1,0),31);
Display[i1] = p_Display;

```

setLeds

Allows the user to set the 8 led's on the PIC board.

Primitive-operation , Public, Return type is 'void'

Args:

'char' p_leds

Body

```

leds = p_leds;

```

writeLeds

Write the leds value to the PIC board.

Primitive-operation , Protected, Return type is void

Body

```

ControlChar[0] = 'L';

io_status = write(fd, ControlChar, 1); //write an 'R' to the PIC board to read the
buffers.

io_status = write(fd, (char*)&leds, 1); //write in 1 character.

//cout << "Led's = " << ControlChar[0] << (char) leds << leds << endl;

```

~PICserial

Shutdown this object;

Virtual, Destructor , Public

Body

```

close(fd);

```

Attributes:

Analog

This is the location of the three analog inputs.

Type of 'char %s[3]', Public

ControlChar

This character is set to the control for the PIC device.

'L' is used to set the value of the LEDs

- L#

'B' is used to set the 32 char board display

- Bcccccccccccccccccccccccccccccccccccc

'R' is used to initiate a transmission of analog and button values.

Type of 'char %s[1]', Public

Display

32 char display buffer on the PIC board

Type of 'char %s[32]', Public

fd

This is the file descriptor of the serial port returned by the open statement.

Type of int, Public

goodread

This is used to indicate the status of the read operation to the PIC board.

Type of OMBoolean, Public

io_status

contains the status of all i/o operations.

Type of int, Public

Keypad

There are 2 bytes of keypad

Type of 'char %s[2]', Public

KeyStatus

There are 2 bytes of keypad status

Type of 'char %s[2]', Public

leds

the led display outputs

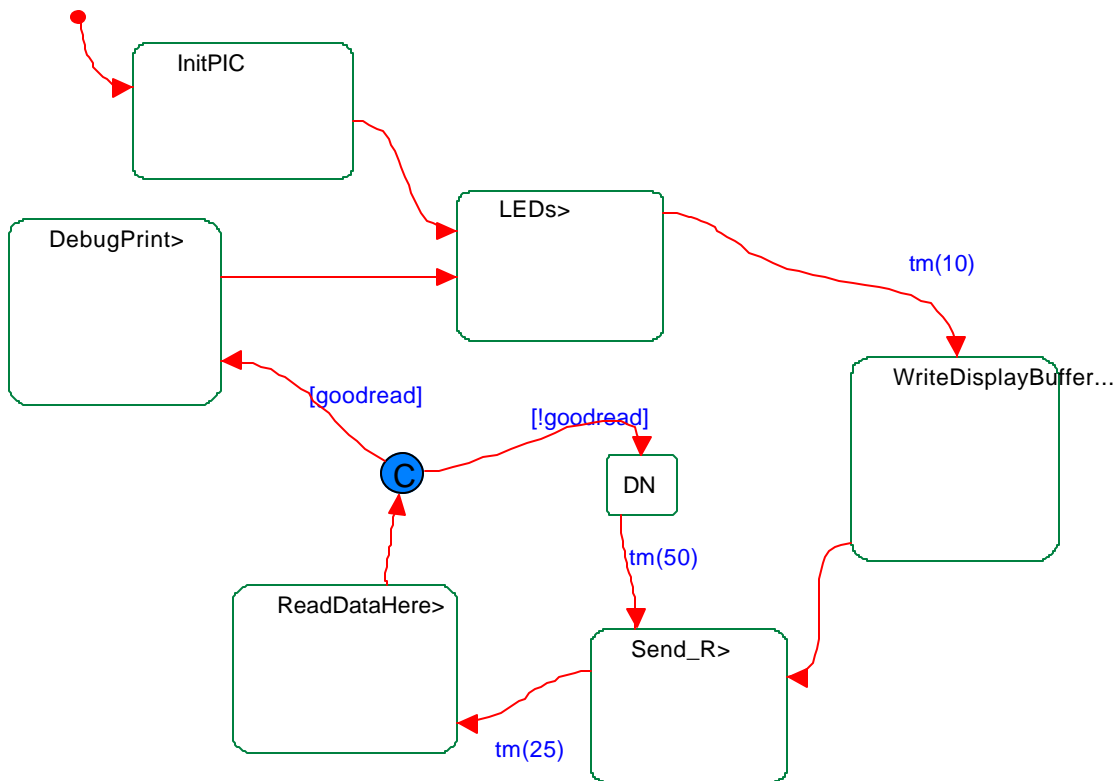
Type of char, Public

readBuffer

buffer to place the data read in by the ReadData operation.

Type of 'char %s[5]', Public

Statechart



ROOT

Or-state

Substates:

DebugPrint

DN

InitPIC

LEDs

ReadDataHere

Send_R

WriteDisplayBuffer

Default Transition

Target:

InitPIC

DebugPrint

Debug state to determine any possible difficulties with the PIC board.

Or-state

EntryAction

```

//cout << " keys " << setbase(16) << setfill('0') << setw(8) << getAllKeyData() ;
//cout << " a0 " << (int)getAnalogValues(0);
//cout << " a1 " << (int)getAnalogValues(1);
//cout << " a2 " << (int)getAnalogValues(2);
//cout << endl;

//cout << setbase(16) << setfill('0') << setw(8) << getAllKeyData() << " " <<
(int)getAnalogValues(0) << " " << (int)getAnalogValues(1) << " " << (int)getAnalogValues(2)
<< endl;

//for (int i=0;i<5;i++)
  
```

```
//{ cout << " " << setbase(16) << (int) readBuffer[i]; }  
//cout << endl;
```

Out Transition

Target:
LEDs

DN

Do nothing state used to create a small delay in the I/O stream.

Or-state

Out Transition

tm(50)

Target:
Send_R

InitPIC

A do-nothing state for future possible use.

Or-state

Out Transition

Target:
LEDs

LEDs

Write the leds attribute to the PIC board.

Or-state

EntryAction

writeLeds();

ExitAction

//leds++;

Out Transition

tm(10)

Target:
WriteDisplayBuffer

ReadDataHere

Read in the 5 characters that the PIC board returns in response to a request for data.

Or-state

EntryAction

goodread = ReadData();

Out Transition

Condition Connector

Branches:
[goodread]

Target:
DebugPrint
[!goodread]

Target:
DN

Send_R

Initiate the reading of the data from the PIC board by sending an 'R' to the board.

Or-state

EntryAction

```
ControlChar[0] = 'R';  
write(fd, ControlChar, 1); //write an 'R' to the PIC board to read the buffers.
```

Out Transition

tm(25)

Target:

ReadDataHere

WriteDisplayBuffer

initiate the execution of the output to the Display on the PIC board.

This state uses a sub_state_chart to accomplish this operation.

Nested Statechart

Or-state

EntryAction

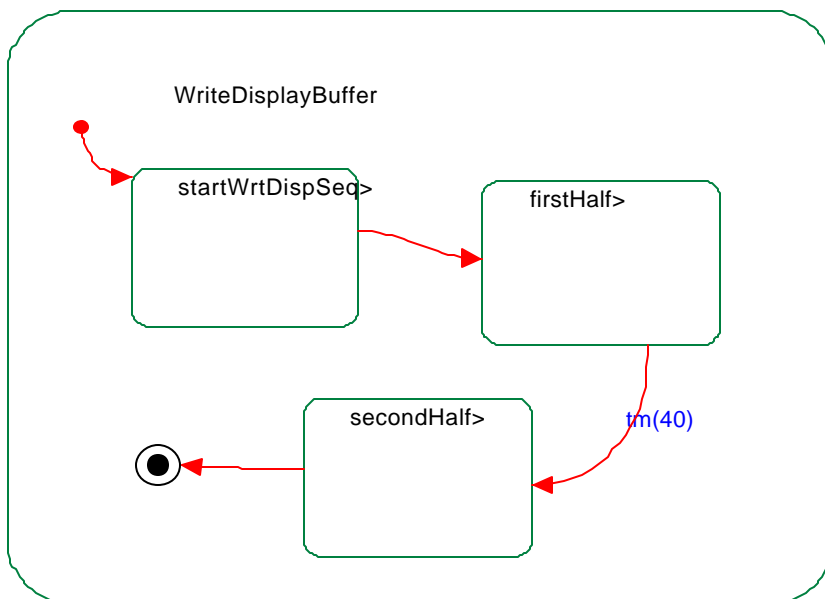
```
//for(int i=0;i<32;i++) Display[i]=leds;
```

Out Transition

Target:

Send_R

Statechart of WriteDisplayBuffer



ROOT

Or-state

Substates:

WriteDisplayBuffer

WriteDisplayBuffer

Or-state

Substates:

firstHalf

secondHalf

startWrtDispSeq

state_9

Default Transition

Target:
startWrtDispSeq

firstHalf

*The actual transmission of characters is broken down into 2 halves.
Send out the first 16 characters.*

Or-state

EntryAction

```
write(fd, Display, 16); //write out 16 characters.
```

Out Transition

```
tm(40)
```

Target:
secondHalf

secondHalf

After a short delay, send out the second 16 characters.

Or-state

EntryAction

```
write(fd, &Display[16], 16); //write out 16 characters.
```

Out Transition

Target:
state_9

startWrtDispSeq

Write the control character 'B' to the PIC board to initiate the sequence of characters to be displayed on the 32 element character display.

Or-state

EntryAction

```
ControlChar[0] = 'B';
```

```
write(fd, ControlChar, 1); //write an 'B' to the PIC board to read the buffers.
```

Out Transition

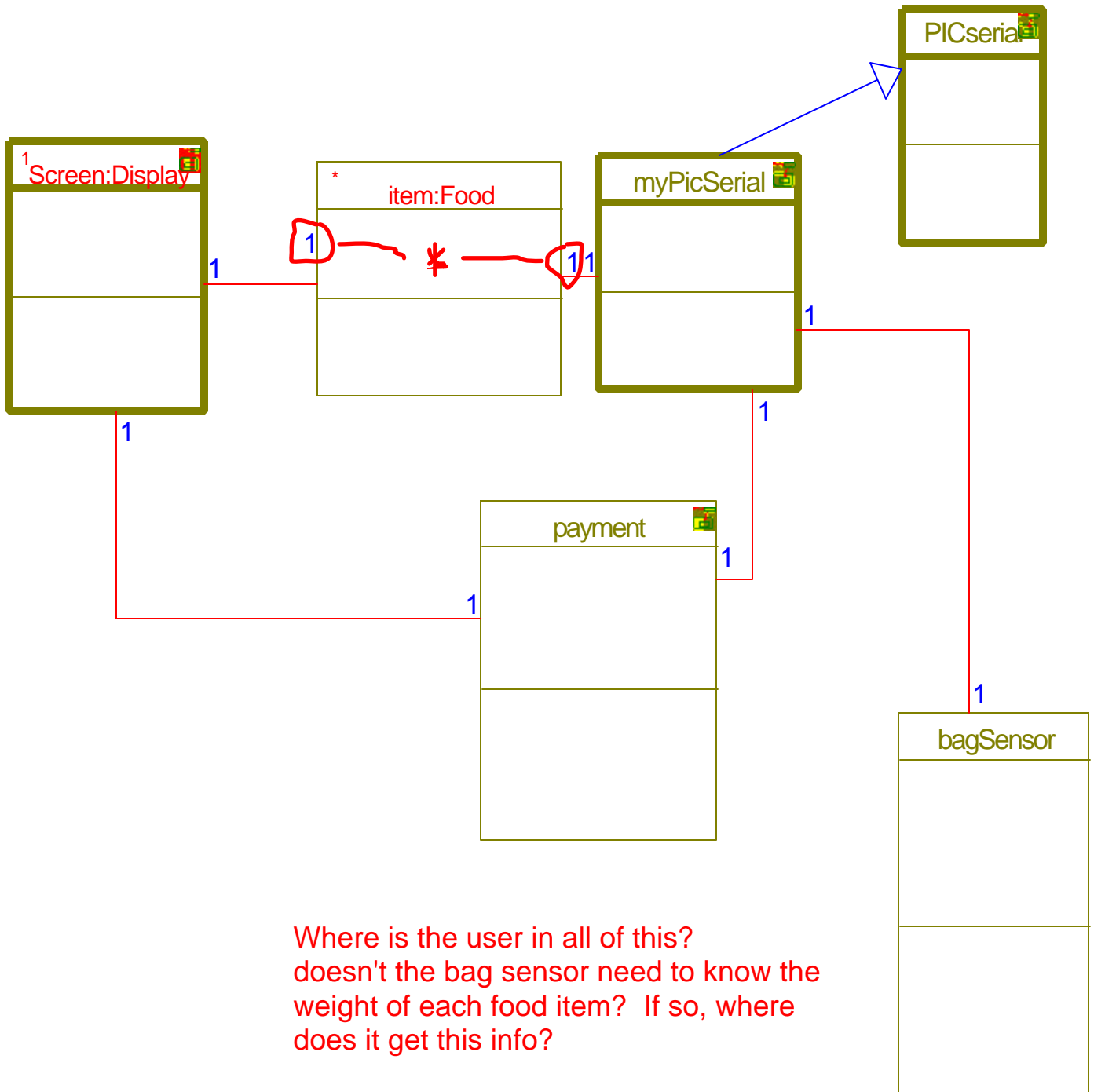
Target:
firstHalf

state_9

Local Termination State

OBJECT MODEL DIAGRAMS

Model1



Where is the user in all of this?
doesn't the bag sensor need to know the weight of each food item? If so, where does it get this info?

COMPONENTS

DefaultComponent

COMPONENT SETTINGS:

Build type: Executable

CONFIGURATIONS:

DefaultConfig

Scope type: Explicit

Instrumentation type: None

Time-model type: Real-time

Statechart generation type: Flat

FILES AND FOLDERS: