OPEN-R SDK Training course

Let’s make an AIBO program using C++!
Communication with system objects
What is OPEN-R?

"OPEN-R" is the standard interface for the entertainment robot system that Sony is actively promoting. This interface greatly expands the capabilities of entertainment robots.

Here are a few of the characteristics of OPEN-R:

+ **Modularized hardware**
  - Change the robot's form by exchanging modules (e.g. you can change a leg module or the head module).
  - Each module is connected by a high speed serial bus featuring auto-detection of the robot's hardware configuration.

+ **Modularized software**
  - Software modules are called “objects”.
  - The programming model allows concurrently running objects to communicate with each other.
  - Connections between objects are defined in the connection description file.
  - It is relatively easy to replace objects.
  - Each object is loaded from the “Memory Stick”.

+ **Supports networking**
  - Supports IEEE802.11b Wireless LAN (via the PC card slot)
  - Supports TCP/IP network protocol
Let’s make AIBO’s eyes blink!
Please refer to the sample program: BlinkingLED

OPEN-R programming involves making “objects” and connecting them together via Inter-Object Communication. Each object is working independently, and is communicating with each other.

To make AIBO’s eye blink, you must make a request to the controller of AIBO’s eye (OVirtualRobotComm).

Light on the eye for 0.5 sec

Excuse me, would you please illuminate AIBO’s bottom right eye for 0.5 sec?

Okay!

OVirtualRobotComm
system object

User object
Establishing the Inter-Object Communication

+ Our User Object will be called BlinkingLED (BLINKLED.BIN)

+ The two objects we will use are POWERMON.BIN (for power issues), and BLINKLED.BIN

[OBJECT.CFG]
/MS/OPEN-R/MW/OBJS/POWERMON.BIN
/MS/OPEN-R/MW/OBJS/BLINKLED.BIN

+ We establish an inter-object communication (direct line) between BlinkingLED.Blink and OVirtualRobotComm.Effector

[CONNECT.CFG]

+ We configure stub.cfg (our ‘telephone’) for BlinkingLED it to have one sending connection (one subject).

[STUB.CFG]
ObjectName : BlinkingLED
NumOfOSubject : 1
NumOfOObserver : 1
Service : "BlinkingLED.Blink.OCommandVectorData.S", null, Ready()
Service : "BlinkingLED.DummyObserver.DoNotConnect.O", null, null
Our “Master Plan” for communicating with OVirtualRobotComm

1. Open each LED device, and get their primitive ID

2. Create commands to blink the LEDs.

3. #1 and #2 are established in the initialize section (DoInit).

4. Prepare a flag value to darken the LEDs when Dostop() is invoked.
1. Open each LED device, and get their primitive ID

static const char* const LED_LOCATOR[] = {
    "PRM:/r1/c1/c2/c3/l1-LED2:11",
    "PRM:/r1/c1/c2/c3/l2-LED2:12",
    "PRM:/r1/c1/c2/c3/l3-LED2:13",
    "PRM:/r1/c1/c2/c3/l4-LED2:14",
    "PRM:/r1/c1/c2/c3/l5-LED2:15",
    "PRM:/r1/c1/c2/c3/l6-LED2:16",
    "PRM:/r1/c1/c2/c3/l7-LED2:17",
};

OPrimitiveID ledID[NUM_LEDS];

void BlinkingLED::OpenPrimitives()
{
    for (int i = 0; i < NUM_LEDS; i++) {
        OStatus result = OPENR::OpenPrimitive(LED_LOCATOR[i], &ledID[i]);
    }
}

These are “Locators”. They look like file names. Each of them are a unique reference to a certain LED.

These are “Primitive IDs”; they behave like file handles.

Loop limit is the number of LEDs (7 for ERS-210)

This function gets the primitive IDs from specified locators.
2. Create commands to blink the LEDs.

```c
void BlinkingLED::NewCommandVectorData()
{
    OStatus result;
    MemoryRegionID cmdVecDataID;
    OCommandVectorData* cmdVecData;

    for (int i = 0; i < NUM_COMMAND_VECTOR; i++) {
        result = OPENR::NewCommandVectorData(NUM_LEDS,
                                              &cmdVecDataID, &cmdVecData);

        region[i] = new RCRegion(cmdVecData->vectorInfo.memRegionID,
                                  cmdVecData->vectorInfo.offset,
                                  (void*)cmdVecData,
                                  cmdVecData->vectorInfo.totalSize);

        cmdVecData->SetNumData(NUM_LEDS);
    }
}
```

2; make two sets of commands
7; Prepare number of LEDs
Make memory regions to insert commands
7 (number of LEDs) data areas are put in one region.
2. Create commands to blink the LEDs.

```c
for (int j = 0; j < NUM_LEDS; j++) {
    OCommandInfo* info = cmdVecData->GetInfo(j);
    info->Set(odataLED_COMMAND2, ledID[j], 1);
    OCommandData* data = cmdVecData->GetData(j);
    OLEDCommandValue2* val = (OLEDCommandValue2*)data->value;
    if (i % 2 == 0) { // There are two command sets
        // In the 1st command set, even LEDs are ON, odd LEDs are OFF
        val[0].led = (j % 2 == 0) ? oledON : oledOFF;
    } else {
        // In the 2nd command set, even LEDs are OFF, odd LEDs are ON
        val[0].led = (j % 2 == 0) ? oledOFF : oledON;
    }
    val[0].period = 64; // 8ms * 64 = 512ms
    // illuminate the LEDs for 0.5 sec
    OSYSDEBUG(("ledID[%d] %d¥n", j, val[0].led));
}
```
2. Create commands to blink the LEDs.

- **OCommandVectorData**
  - 2 command sets
  - CommandVecData

- **OCommandInfo**
  - 7 LEDs
  - info
  - In this ID of LED
  - Use 1 frame

- **OCommandData**
  - 7 LEDs
  - Data
  - Val[0]

- **OLEDCommandValue2**
  - .led
  - Blink LED
  - .period
  - For 0.5sec

16 frames (fixed) can be saved in this area.
3. #1 and #2 are established in the initialize section (DoInit).

```cpp
OStatus
BlinkingLED::DoInit(const OSystemEvent& event)
{
    OSYSDEBUG("BlinkingLED::DoInit()¥n");

    NEW_ALL_SUBJECT_AND_OBSERVER;
    REGISTER_ALL_ENTRY;
    SET_ALL_READY_AND_NOTIFY_ENTRY;

    V.S.P. (Very special pattern)

    OpenPrimitives();
    NewCommandVectorData();

    return oSUCCESS;
}
```
4. Prepare a flag value to darken the LEDs when Dostop() is invoked.

Constructor

BlinkingLED::BlinkingLED() :
    blinkingLEDState(BLS_IDLE)
{
    for (int i = 0; i < NUM_LEDS; i++) ledID[i] = oprimitiveID_UNDEF;
    for (int i = 0; i < NUM_COMMAND_VECTOR; i++)
        region[i] = 0;
}

OStatus
BlinkingLED::DoStop(const OSystemEvent& event)
{
    OSYSDEBUG("BlinkingLED::DoStop()\n");
    blinkingLEDState = BLS_IDLE;
    DISABLE_ALL_SUBJECT;
    DEASSERT_READY_TO_ALL_OBSERVER;
    return oSUCCESS;
}

void
BlinkingLED::Ready(const OReadyEvent& event)
{
    OSYSDEBUG("BlinkingLED::Ready()\n");
    BlinkLED();
    blinkingLEDState = BLS_START;
    ENABLE_ALL_SUBJECT;
    ASSERT_READY_TO_ALL_OBSERVER;

    return oSUCCESS;
}
Finally, This is a command to blink the LEDs.

```cpp
void BlinkingLED::BlinkLED()
{
    static int index = -1;

    if (index == -1) { // BlinkLED() is called first time.
        index = 0;
        subject[sbjBlink]->SetData(region[index]);
        index++;
    }

    subject[sbjBlink]->SetData(region[index]);
    subject[sbjBlink]->NotifyObservers();

    index++;
    index = index % NUM_COMMAND_VECTOR;
    OSYSDEBUG("index %d\n", index);
}
```

On the first execution, execute `SetData()` twice, and send two commands to blink LEDs. The second command will be “buffered” by the Observer.
This Blink LED program uses a “double buffer” approach to continuously process commands. How does the Observer know to buffer these commands?

**User object**

I light on the LED for 0.5 sec, but I don’t process next command yet.

**OVirtualRobotComm system object**

Blink LED for 0.5 sec

Excuse me, would you please light on the LED for 0.5 sec and then turn off for 0.5 sec.

**DoStart()**

SetData(region[0])

SetData(region[1])

BlinkingLED

**Ready()**

Next, light on the LED for 0.5 sec.

I turn off the LED for 0.5 sec, but I don’t process next command yet.

**Ready()**

Next, turn off the LED for 0.5 sec

SetData(region[0])

SetData(region[1])
Details of how OVirtualRobotComm.Effector system Observer works

Blink LED for 0.5sec

OVirtualRobotComm system object

SetData(region[0])
SetData(region[1])

Excuse me, would you please light on the LED for 0.5 sec and then turn off for 0.5 sec.

BlinkingLED
User object

I light on the LED for 0.5 sec, but I don’t process next command yet.

All data are cleared if you don’t make a special code.

Normally, in observers that are made by the user, when AssertReady() is invoked, all data areas are cleared, even if the data is not used.

I light on the LED for 0.5 sec, but I don’t process next command yet.
And additionally, the next command “turn off LED for 0.5 sec” has been forgotten forever.
Details of how OVirtualRobotComm.Effector system Observer works
This is a technique you can use in user objects where previous data is kept and processed later.

Blink LED for 0.5sec

OVirtualRobotComm system object

SetData(region[0])
SetData(region[1])

BlinkingLED user object

Excuse me, would you please light on the LED for 0.5 sec and then turn off for 0.5 sec.

Well, there are two data sections that are sent.

I will execute the 1st command.
And the 2nd command will be executed later.
Oh, ok, I will keep 2nd command.

I light on the LED for 0.5 sec, but I don’t process next command yet.
Please send next data.

ONotifyEvent event;
n = event.NumOfData();

The n is 2, so you know the data is 2.

RCRegion *region;
region = event.Data(0);
do something

Light on the LEDs according to the 1st data
region = event.Data(1);
region->AddReference();

Increment the reference counter of 2nd data

Observer[event.ObsIndex()]->AssertReady();

The 1st data is cleared
The 2nd data is kept and is moved to the top

This code exists in the system Observer object. You won’t find it in the BlinkingLED source code.
Details of how OVirtualRobotComm.Effector system Observer works
This is a technique you can use in user object that previous data is kept and proceed the data with a continuation.

Blink LED for 0.5 sec
Well, there are two data sections that are sent.
I will execute the 1st command. And the 2nd command will be executed later. Oh, ok, I will keep 2nd command.
I light on the LED for 0.5 sec, but I don’t process next command yet. Please send next data.

OVirtualRobotComm system object

SetData(region[0])
SetData(region[1])

BlinkingLED
user object

ONotifyEvent event;
n = event.NumOfData();
The n is 2, so you know the data is 2.
Here, check the reference count by using region->NumberOfReference(), and it will be more than two. You can decrease the reference count by using region->RemoveReference().

Excuse me, would you please light on the LED for 0.5 sec and then turn off for 0.5 sec.

RCRegion *region;
region = event.Data(0);
do something
region = event.Data(1);
region->AddReference();
Light on the LEDs according to the 1st data
Increment the reference counter of 2nd data
Observer[event.ObsIndex()]->AssertReady();
The 1st data is cleared
The 2nd data is kept and is moved to the top

Well, there are two data sections that are sent.
I will execute the 1st command. And the 2nd command will be executed later. Oh, ok, I will keep 2nd command.
I light on the LED for 0.5 sec, but I don’t process next command yet. Please send next data.

This code exists in the system Observer object. You won’t find it in the BlinkingLED source code.
Summary

+ 1. To establish inter-object communication, create the “telephone contractor list” (OBJECT.CFG), the “direct line list” (CONNECT.CFG), and the “telephone” (stub.cfg of BlinkingLED).
+ 2. To blink LEDs, make a request to the Effector observer in the OVirtualRobotComm system object.
+ 3. To make a request to the Effector observer, use the OCommandVectorData data type.
+ 4. To access LEDs, open 7 LEDs by using 
  
  OpenPrimitives();
+ 5. Make 2 memory areas that consist of OCommandVectorData data types, where we will insert the control data.
+ 6. Insert commands to blink 7 LEDs.
+ 7. Set which LEDs are on or off, and how long the LEDs are lit, in the 2 areas that are created in section 5
  
  NewCommandVectorData();
+ 8. Request OVirtualRobotComm object to light on or turn off the LEDs.
  
  BlinkLED();