OPEN-R SDK Training Course

Let’s make an AIBO program using C++!
Inter-Object Communication
"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
Now, the main topic: Programming with the OPEN-R SDK

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.
These objects contain components that can send data as well as receive data.

Inter-object communication in OPEN-R is established by the “subject” and the “observer” (which are included in the “object”).

Object A

Object B

Object C
Send data in Ready() function
Get data in Notify() function

Send data in Ready() that is included in the subject.
Invoke NotifyObservers() when sending data is finished.

Receive data in Notify() that is included in the observer.
Invoke AssertReady() when receiving data is finished.

Object A
SampleSubject

Object B
SampleObserver
Ready() means the partner said “I am ready to receive data”
Notify() means the partner said “I sent data to you”
Let me describe more about Ready() and Notify().
Ready() means the partner said “I am ready to receive data”
Notify() means the partner said “I sent data to you”

Let me describe more about Ready() and Notify().
Ready() means the partner said “I am ready to receive data”
Notify() means the partner said “I sent data to you”

Let me describe more about Ready() and Notify().

Ready() is executed. Here the data is sent to the partner. (Really, the data is put in a public area)

NotifyObservers() is executed. The system will now inform the partner.
Ready() means the partner said “I am ready to receive data”
Notify() means the partner said “I sent data to you”

Let me describe more about Ready() and Notify().

- **Ready()** is executed. Here the data is sent to the partner. (Really, the data is put in a public area)
- **NotifyObservers()** is executed. The system will now inform the partner.
- **Notify()** is executed. Here the partner can get the data and can process the data.
Ready() means the partner said “I am ready to receive data”
Notify() means the partner said “I sent data to you”
Let me describe more about Ready() and Notify().

Ready() is executed. Here the data is sent to the partner. (Really, the data is put in a public area)
I sent data
NotifyObservers() is executed. The system will now inform the partner.
Notify() is executed. Here the partner can get the data and can process the data.
I'm ready. Please send next data
AssertReady() is executed when the data processing is finished. AssertReady() means “I am ready to get next data”.
AssertReady() is executed when the data processing is finished. AssertReady() means “I am ready to get next data”.

NotifyObservers() is executed. The system will now inform the partner.
Ready() means the partner said “I am ready to receive data”
Notify() means the partner said “I sent data to you”
Let me describe more about Ready() and Notify().

First, someone says “send data”

Ready() is executed. Here the data is sent to the partner. (Really, the data is put in a public area)

NotifyObservers() is executed. The system will now inform the partner.

Notify() is executed. Here the partner can get the data and can process the data.

AssertReady() is executed when the data processing is finished. AssertReady() means “I am ready to get next data”.

Please .. send .. the … data
I sent data
Please .. send .. next .. data
I'm ready.
Please .. send .. next .. data

Ready() is executed. Here the data is sent to the partner. (Really, the data is put in a public area)
To send the data to the public area, invoke subject[]->SetData()
To receive the data from the public area, invoke event.Data(0)

To send the data:

```cpp
void SampleSubject::Ready(const OReadyEvent& event)
{
    char str[32];
    strcpy(str, "!!! Hello world !!!");
    subject[sbjSendString]->setData(str, sizeof(str));
    subject[sbjSendString]->NotifyObservers();
}
```

If there are multiple subjects in one object, an array index is required. If there is only one subject, `sbjSendString` should be replaced by ‘0’.
To send the data to the public area, invoke subject[ ]->SetData()
To receive the data from the public area, invoke event.Data(0)

To receive the data:

```cpp
void SampleObserver::Notify(const ONotifyEvent& event)
{
    const char* text = (const char *)event.Data(0);
    OSYSPRINT(("SampleObserver::Notify() %s\n", text));
    observer[event.ObsIndex()]->AssertReady();
}
```

Out to console

If there is only one observer, the array index should be ‘0’.
An analogy: a ‘telephone’ is needed to perform Inter-Object Communication

To communicate between different objects, a file named stub.cfg (our ‘telephone’) is needed. Each object needs its own stub.cfg file.

For sending data: stub.cfg

ObjectName: SampleSubject
NumOfOSubject: 1
NumOfOObserver: 1
Service: "SampleSubject.SendString.char.S", null, Ready()

For receiving data: stub.cfg

ObjectName: SampleObserver
NumOfOSubject: 1
NumOfOObserver: 1
Service: "SampleObserver.DummySubject.DoNotConnect.S", null, null
A dedicated direct line is needed to connect objects

To connect objects, a file named connect.cfg is needed. Each OPEN-R application requires one (and only one) connect.cfg file. This file is located in /OPEN-R/MW/CONF on the “Memory Stick”.

connect.cfg ← Direct line infrastructure
SampleSubject.SendString.char.S SampleObserver.ReceiveString.char.O

Unidirectional communication

SampleSubject.SendString.char.S → SampleObserver.ReceiveString.char.O
One object can be connected to multiple objects. We must create ‘telephones’ and ‘dedicated lines’ for each communication path to send and receive data.

[stub.cfg] In case of 3 sending, 2 receiving
ObjectName : ObjectA
NumOfOSubject : 3
NumOfOObserver : 2
Service : "ObjectA.SendString1.char.S", null, Ready1()
Service : "ObjectA.SendString2.char.S", null, Ready2()
Service : "ObjectA.SendString3.char.S", null, Ready3()
Service : "ObjectA.ReceiveString1.char.O", null, Notify1()
Service : "ObjectA.ReceiveString2.char.O", null, Notify2()

[connect.cfg]
ObjectA.SendString1.char.S ObjectB.ObserverA.char.O
ObjectA.SendString2.char.S ObjectC.ObserverB.char.O
ObjectD.SubjectAlpha.char.S ObjectA.ReceiveString1.char.O
ObjectE.SubjectBeta.char.S ObjectA.ReceiveString2.char.O
under this line, there are the other connection settings ...
We must create a list of objects in the application.

The objects list is called object.cfg. We must create this list to inform the system which objects will exist in the application. Each OPEN-R application requires this file. The object.cfg file should be placed in /OPEN-R/MW/CONF on the memory stick.

object.cfg

/MS/OPEN-R/MW/OBJS/POWERMON.BIN
/MS/OPEN-R/MW/OBJS/SUBJECT.BIN
/MS/OPEN-R/MW/OBJS/OBSERVER.BIN
This is V.S.P. (Very special pattern)
Copy and paste, and then edit the object names illustrated in red
SampleSubject.cc

```c++
#include <string.h>
#include <OPENR/OSyslog.h>
#include <OPENR/core_macro.h>
#include "SampleSubject.h"

SampleSubject::SampleSubject()
{
}

OStatus
SampleSubject::DoInit(const OSystemEvent& event)
{
    NEW_ALL_SUBJECT_AND_OBSERVER;
    REGISTER_ALL_ENTRY;
    SET_ALL_READY_AND_NOTIFY_ENTRY;
    return oSUCCESS;
}

OStatus
SampleSubject::DoStart(const OSystemEvent& event)
{
    ENABLE_ALL_SUBJECT;
    ASSERT_READY_TO_ALL_OBSERVER;
    return oSUCCESS;
}

OStatus
SampleSubject::DoStop(const OSystemEvent& event)
{
    DISABLE_ALL_SUBJECT;
    DEASSERT_READY_TO_ALL_OBSERVER;
    return oSUCCESS;
}

OStatus
SampleSubject::DoDestroy(const OSystemEvent& event)
{
    DELETE_ALL_SUBJECT_AND_OBSERVER;
    return oSUCCESS;
}

Here,
add Ready() or Notify() as needed
This is V.S.P. (Very special pattern)
Copy and paste, and then edit the object names illustrated in red
SampleSubject.h

```cpp
#include <OPENR/OObject.h>
#include <OPENR/OSubject.h>
#include <OPENR/OObserver.h>
#include "def.h"

class SampleSubject : public OObject {
public:
    SampleSubject();
    virtual ~SampleSubject() {};

    OSubject*   subject[numOfSubject];
    OObserver*  observer[numOfObserver];

    virtual OStatus DoInit   (const OSystemEvent& event);
    virtual OStatus DoStart  (const OSystemEvent& event);
    virtual OStatus DoStop   (const OSystemEvent& event);
    virtual OStatus DoDestroy(const OSystemEvent& event);

    void Ready(const OReadyEvent& event);
};
```

Here, add Ready() or Notify() as needed
This is V.S.P. (Very special pattern)
Copy and paste, and then edit the object names illustrated in red
OCF file

object sampleSubject 3072 16384 128 cache tlb user

Edit only the object name.
The other parameters should not be changed.
This is V.S.P. (Very special pattern)
Copy and paste, and then edit the object names illustrated in red

Makefile

```
PREFIX=/usr/local/OPEN_R_SDK
INSTALLDIR=../MS
CXX=$(PREFIX)/bin/mipsel-linux-g++
STRIP=$(PREFIX)/bin/mipsel-linux-strip
MKBIN=$(PREFIX)/OPEN_R/bin/mkbin
STUBGEN=$(PREFIX)/OPEN_R/bin/stubgen2
MKBINFLAGS=-p $(PREFIX)
LIBS=-lObjectComm -lOPENR
CXXFLAGS= -O2 -g -I. -I$(PREFIX)/OPEN_R/include/R4000 -I$(PREFIX)/OPEN_R/include

.PHONY: all install clean

all: sampleSubject.bin

%.o: %.cc

$(CXX) $(CXXFLAGS) -o $@ -c $^
SampleSubjectStub.cc: stub.cfg
$(STUBGEN) stub.cfg
sampleSubject.bin: SampleSubjectStub.o SampleSubject.o sampleSubject.ocf

$(MKBIN) $(MKBINFLAGS) -o $@ $(LIBS)
$(STRIP) $@

install: sampleSubject.bin

gzip -c sampleSubject.bin > $(INSTALLDIR)/OPEN-R/MW/OBJS/SUBJECT.BIN

clean:

rm -f *.o *.bin *.elf *.snap.cc
rm -f SampleSubjectStub.h SampleSubjectStub.cc def.h entry.h
rm -f $(INSTALLDIR)/OPEN-R/MW/OBJS/SUBJECT.BIN
```
Many objects interact in a substantial OPEN-R application. There can even be objects that don’t connect to any other objects.

This object does not connect to any other objects. It checks the power button, and when the button is pushed, it shuts down AIBO.