# BOOTPRINT Kit Sample Deployment and Development Manual

Horizon Robotics

2019.12

## Version History

| Version | Status | Date | Details |
|---|---|---|---|
| 1.0 | Release | 2019/12 | C： First Edition， Release |
| 1.1 | Release | 2020/1/6 | Change to adapt lastest deploy process |

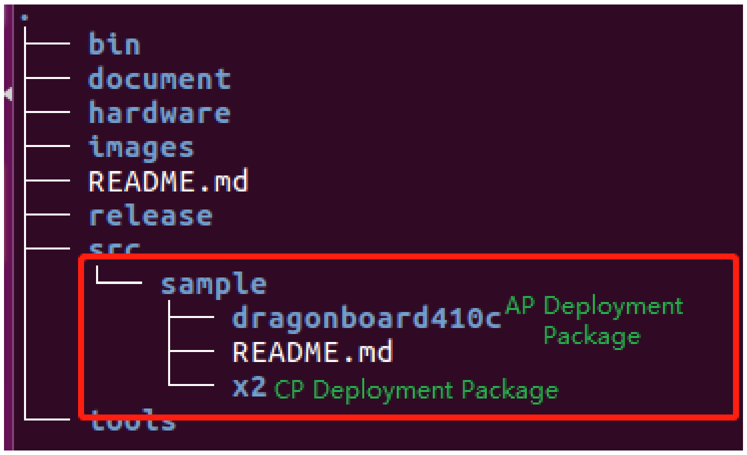
## Contents

# 1. Step by step deployment of BOOTPRINT Kit standard sample

## 1.1. Obtain Deployment Package

The deployment package can be obtained from github:
https://github.com/HorizonRobotics/bootprint_x2

Sample application is under the directory: /src/sample



```
.
├── bin
├── document
├── hardware
├── images
├── README.md
├── release
├── src
│   └── sample
│       ├── dragonboard410c
│       ├── README.md
│       └── x2
└── tools
```

## 1.2. CP Side Deployment

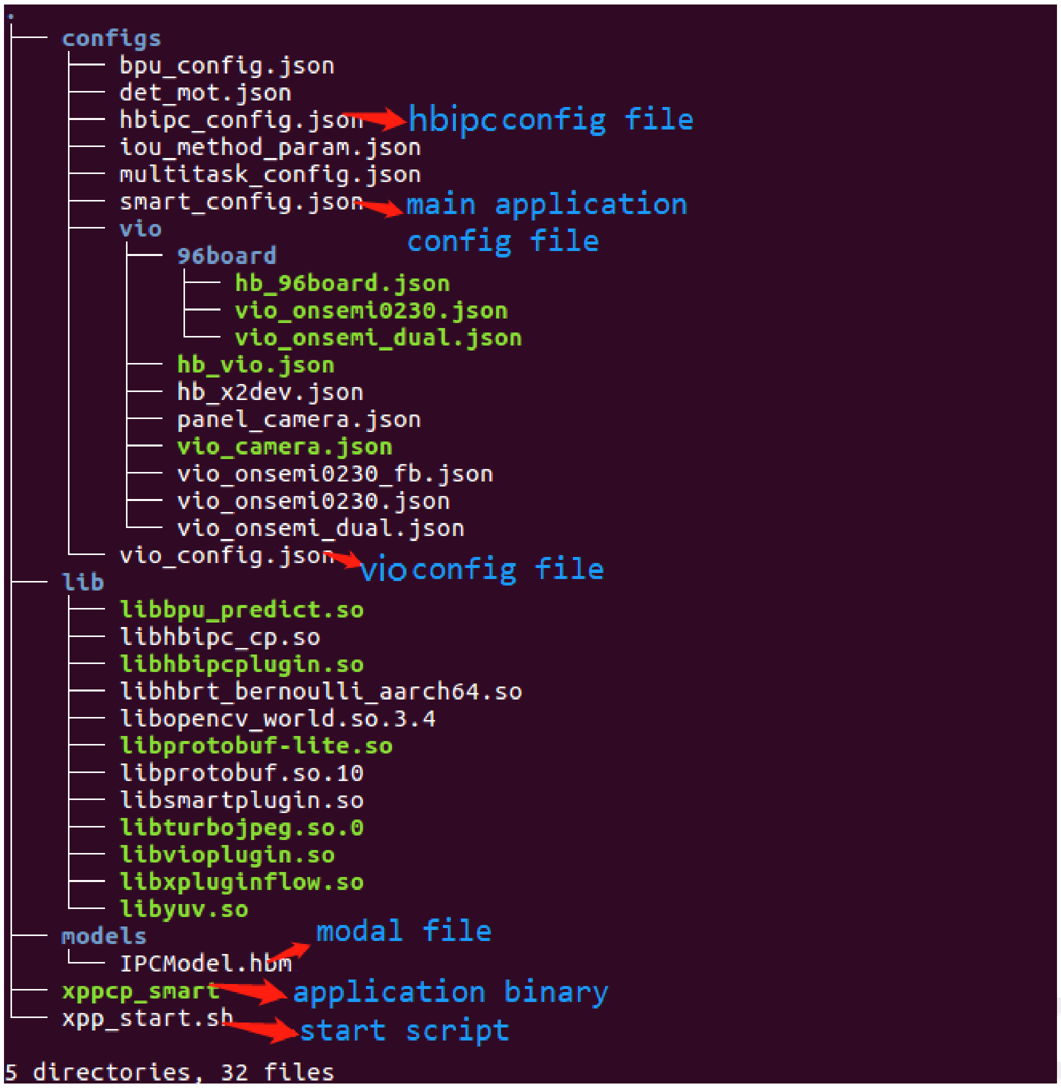
1) Get cp deploy package:

```
> cd src/sample/x2/xpp_cp/smartplugin/deploy/
> tar czf xppcp_smart.tgz xppcp_smart
```

2) Transfer the CP Side deployment package to any directory on CP and decompress. (Usually we transfer the package to directory: /userdata/)

```
> tar zxf xppcp_smart.tgz
```

The contents and directory structure of the deployment package is shown below:



```
.
├── configs
│   ├── bpu_config.json
│   ├── det_mot.json
│   ├── hbipc_config.json
│   ├── iou_method_param.json
│   ├── multitask_config.json
│   ├── smart_config.json
│   ├── vio
│   │   ├── 96board
│   │   │   ├── hb_96board.json
│   │   │   ├── vio_onsemi0230.json
│   │   │   └── vio_onsemi_dual.json
│   │   ├── hb_vio.json
│   │   ├── hb_x2dev.json
│   │   ├── panel_camera.json
│   │   ├── vio_camera.json
│   │   ├── vio_onsemi0230_fb.json
│   │   ├── vio_onsemi0230.json
│   │   └── vio_onsemi_dual.json
│   └── vio_config.json
├── lib
│   ├── libbpu_predict.so
│   ├── libhbipc_cp.so
│   ├── libhbipcplugin.so
│   ├── libhbrt_bernoulli_aarch64.so
│   ├── libopencv_world.so.3.4
│   ├── libprotobuf-lite.so
│   ├── libprotobuf.so.10
│   ├── libsmartplugin.so
│   ├── libturbojpeg.so.0
│   ├── libvioplugin.so
│   ├── libxpluginflow.so
│   └── libyuv.so
├── models
│   └── IPCModel.hbm
├── xppcp_smart
└── xpp_start.sh

5 directories, 32 files
```

**libvioplugin.so**: library for video processing module, vio module. The file named vio_config.json is the vio configuration file.

**libhbipcplugin.so**: module handling transfer between CP and AP. Currently handles the smart frame transfer. The file named hbipc_config.json is the configuration file of hbipc.

**libsmartplugin.so**: module handling intelligent processing. This module handles intelligent workflow. The file named det_mot.json is the configuration file of this module.

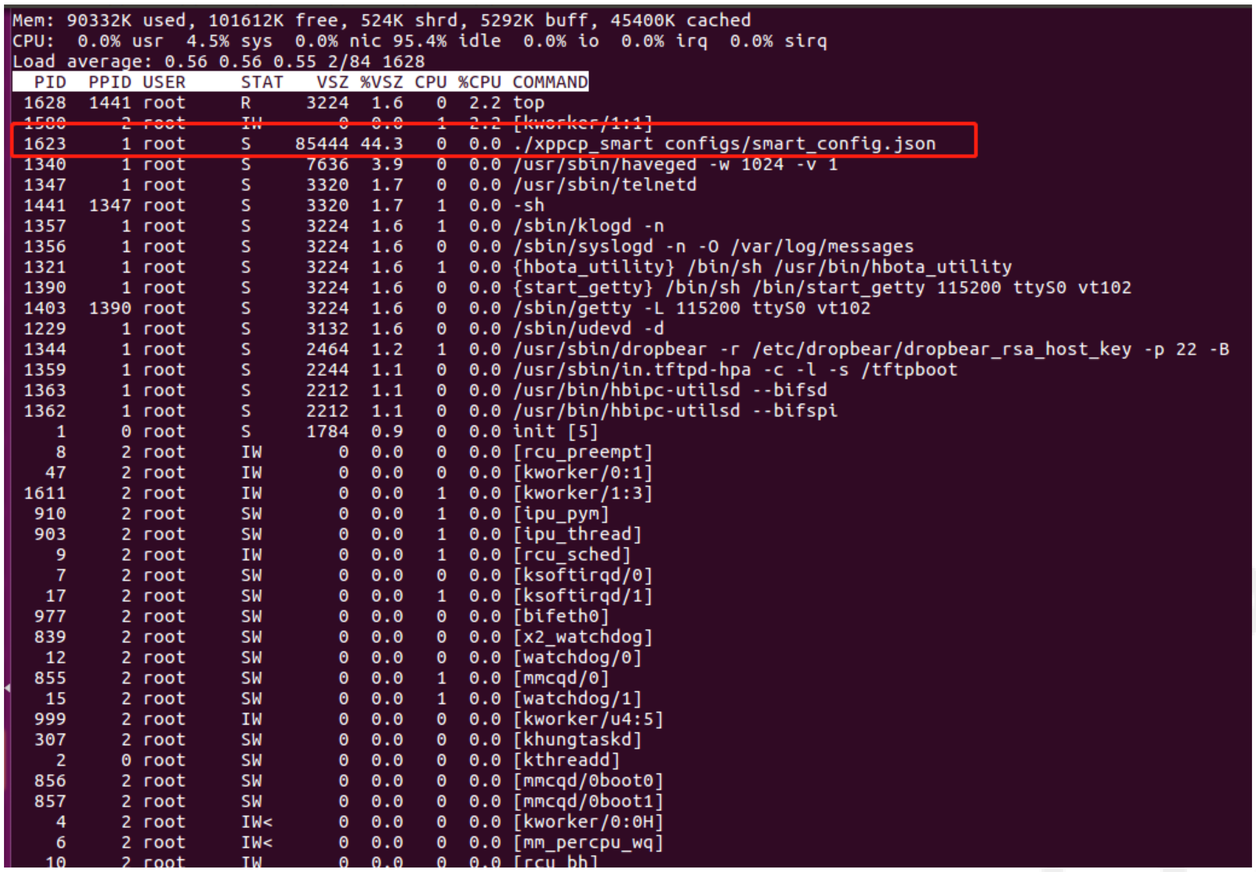
**xppcp_smart**: Main application binary.

3) Start CP side application:

Execute the following command to start CP side application:

```
sh xpp_start.sh
```

If the script launch successfully, **the CP application would block at hbipc waiting for AP to connect.** Use top to check the application started:



```
Mem: 90332K used, 101612K free, 524K shrd, 5292K buff, 45400K cached
CPU:  0.0% usr  4.5% sys  0.0% nic 95.4% idle  0.0% io  0.0% irq  0.0% sirq
Load average: 0.56 0.56 0.55 2/84 1628
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
 1628  1441 root     R     3224  1.6   0  2.2 top
 1588     2 root     IW       0  0.0   1  2.2 [kworker/1:1]
 1623     1 root     S    85444 44.3   0  0.0 ./xppcp_smart configs/smart_config.json
 1340     1 root     S     7636  3.9   0  0.0 /usr/sbin/haveged -w 1024 -v 1
 1347     1 root     S     3320  1.7   0  0.0 /usr/sbin/telnetd
 1441  1347 root     S     3320  1.7   1  0.0 -sh
 1357     1 root     S     3224  1.6   1  0.0 /sbin/klogd -n
 1356     1 root     S     3224  1.6   0  0.0 /sbin/syslogd -n -O /var/log/messages
 1321     1 root     S     3224  1.6   1  0.0 {hbota_utility} /bin/sh /usr/bin/hbota_utility
 1390     1 root     S     3224  1.6   0  0.0 {start_getty} /bin/sh /bin/start_getty 115200 ttyS0 vt102
 1403  1390 root     S     3224  1.6   0  0.0 /sbin/getty -L 115200 ttyS0 vt102
 1229     1 root     S     3132  1.6   0  0.0 /sbin/udevd -d
 1344     1 root     S     2464  1.2   1  0.0 /usr/sbin/dropbear -r /etc/dropbear/dropbear_rsa_host_key -p 22 -B
 1359     1 root     S     2244  1.1   0  0.0 /usr/sbin/in.tftpd-hpa -c -l -s /tftpboot
 1363     1 root     S     2212  1.1   0  0.0 /usr/bin/hbipc-utilsd --bifsd
 1362     1 root     S     2212  1.1   0  0.0 /usr/bin/hbipc-utilsd --bifspi
    1     0 root     S     1784  0.9   0  0.0 init [5]
    8     2 root     IW       0  0.0   0  0.0 [rcu_preempt]
   47     2 root     IW       0  0.0   0  0.0 [kworker/0:1]
 1611     2 root     IW       0  0.0   1  0.0 [kworker/1:3]
  910     2 root     SW       0  0.0   1  0.0 [ipu_pym]
  903     2 root     SW       0  0.0   1  0.0 [ipu_thread]
    9     2 root     IW       0  0.0   1  0.0 [rcu_sched]
    7     2 root     SW       0  0.0   0  0.0 [ksoftirqd/0]
   17     2 root     SW       0  0.0   1  0.0 [ksoftirqd/1]
  977     2 root     SW       0  0.0   0  0.0 [bifeth0]
  839     2 root     SW       0  0.0   0  0.0 [x2_watchdog]
   12     2 root     SW       0  0.0   0  0.0 [watchdog/0]
  855     2 root     SW       0  0.0   1  0.0 [mmcqd/0]
   15     2 root     SW       0  0.0   1  0.0 [watchdog/1]
  999     2 root     IW       0  0.0   0  0.0 [kworker/u4:5]
  307     2 root     SW       0  0.0   0  0.0 [khungtaskd]
    2     0 root     SW       0  0.0   0  0.0 [kthreadd]
  856     2 root     SW       0  0.0   0  0.0 [mmcqd/0boot0]
  857     2 root     SW       0  0.0   0  0.0 [mmcqd/0boot1]
    4     2 root     IW<      0  0.0   0  0.0 [kworker/0:0H]
    6     2 root     IW<      0  0.0   0  0.0 [mm_percpu_wq]
   10     2 root     IW       0  0.0   0  0.0 [rcu_bh]
```

The configuration file smart_config.json is shown below:

```
{
  "xroc_workflow_file": "configs/det_mot.json",
  "enable_profile": 0,
  "profile_log_path": "/userdata/log/profile.txt"
}
```

- xroc_workflow_file: specifies the intelligent workflow config file
- enable_profile: indicates whether online profile is enabled.
- profile_log_path: the path to the log output of online profile

Usage of xppcp_smart application:

```
Xppcp_smart  [-i/-d/-w/-f] xroc_config_file
```

- -i/-d/-w/-f indicates one of the log level in: info, debug, warning, fatal

## 1.3. AP Side Deployment

1) Loading AP Side drivers

Get drivers package:

```
>cd src/sample/dragonboard410c/
>tar czf ap_base.tgz ap_base
```

Transfer ap_base.tgz to any directory on AP side (DB410C). Decompress and execute the following command in the target directory:

```
>sudo ./410c_bif_depoly.sh
>sudo ./single_pix_stream.sh
```

2) AP client deploy

Get AP deploy package:

```
>cd src/sample/dragonboard410c/
>tar czf XppClientMini.tgz XppClientMini
```

Transfer XppClientMini.tgz to any directory on AP, decompress and execute the following command:

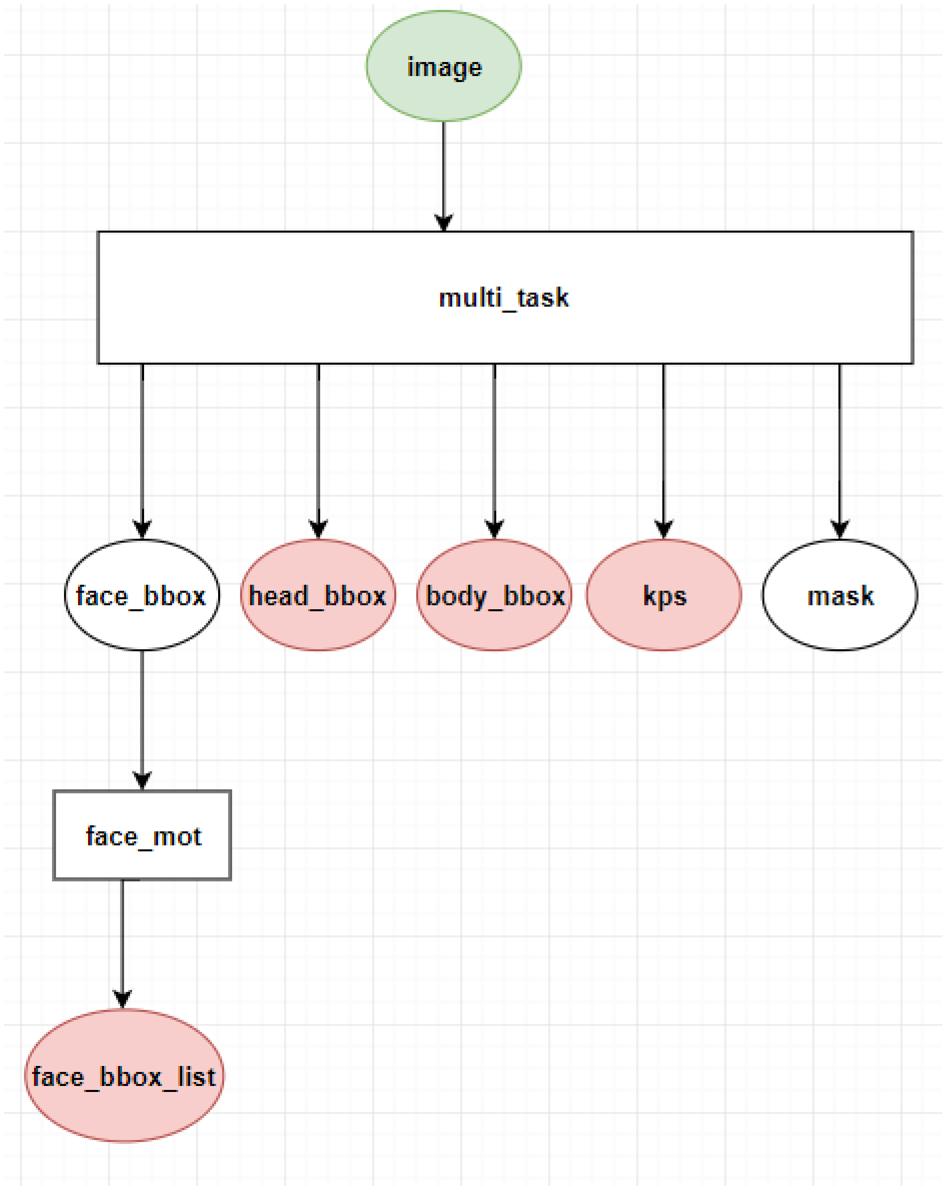
```
sudo ./start.sh
```

# 2. CP Side Intelligent Module Workflow Description

libsmartplugin.so is the intelligent processing module, one of the workflow processes intelligent frames. The configuration file of the workflow is det_mot.json:

```
{
  "inputs": ["image"],
  "outputs": [
    "image",
    "face_bbox_list",
    "head_box",
    "body_box",
    "kps"
  ],
  "workflow": [
    {
      "thread_count": 1,
      "method_type": "FasterRCNNMethod",
      "unique_name": "multi_task",
      "inputs": [
        "image"
      ],
      "outputs": [
        "face_box",
        "head_box",
        "body_box",
        "kps",
        "mask"
      ],
      "method_config_file": "multitask_config.json"
    },
    {
      "thread_count": 1,
      "method_type": "MOTMethod",
      "unique_name": "face_mot",
      "inputs": [
        "face_box"
      ],
      "outputs": [
        "face_bbox_list",
        "face_disappeared_track_id_list"
      ],
      "method_config_file": "iou_method_param.json"
    }
  ]
}
```

The configuration file above defined a workflow demonstrated below:



The default output includes:

- **face_bbox_list:** The ids tracking human face bound box list
- **head_bbox:** Human head detection bound box
- **body_bbox:** Human body detection bound box
- **kps:** Human skeleton key points

The four functions are the currently available. User can also modify the configuration file to turn off some of the functions. For example, if human head detection is not needed, delete head_bbox entries at the beginning of outputs of the workflow configuration file:

```
{
  "inputs": ["image"],
  "outputs": [
    "image",
    "face_bbox_list",
    "body_box",
    "kps"
  ],
  "workflow": [
…
…
…
  ]
}
```

# 3. Demo Application Development Manual

Demo application is open source. User can modify rendering methods based on AP deployment package.

Note: AP side application can be compiled directly on BOOTPRINT Kit AP (DB410C) using make command.

Overall, the demo application can be divided into two main modules:

1. Obtain source data, output synchronized video and intelligent frames;

2. Render video frames and overlap the intelligent frame on video frame.

| Standard API/Module | Description |
|---|---|
| Device data Input | Abstract data input API. Shield lower level device difference from user. |
| High Performance Render Engine | Render engine, simplify video stream and overlapping with intelligent data. |

## 3.1. Intelligent Video Standard Input Stream

Please refer to **InterfaceDeviceStream**, for detail description of input stream API. It serves to provide the following functions:

1. Device authentication, device control;
2. After the device is authenticated, obtain the input data from device.

Input stream utilize plugin standard design. While developing the input stream for a new device, the API implementation has to comply with the standard generating a dynamically linked library (DLL). The DLL will be used by demo application. Some local devices (such as BOOTPRINT Kit) does not need authentication for now. For remote devices (such as IPC), authentication is needed.

```
class InterfaceMsgContainer
{
public:
    virtual void recvVideoBlob(const PureVideoBlob *video, const SmartDataBlob *data)
= 0;
    virtual void recvCommnBlob(const SmartDataBlob *data) = 0;
};

class InterfaceDeviceStream
{
public:
    virtual void setReceiver(InterfaceMsgContainer *container) = 0;
};
```

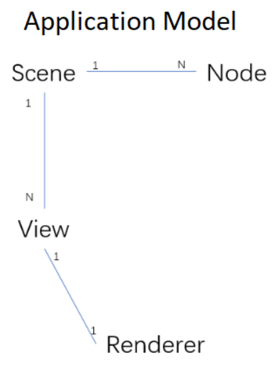
As described above, setReceiver registers stream receiver container. The container has two APIs, recvVideoBlob receives intelligent video (Synchronization is needed between video and intelligent data). The other API is recvCommnBlob which receives common data(intelligent data).

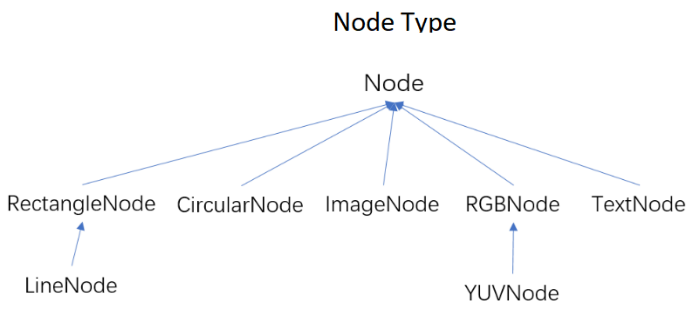
Usually, rendering utilizes recvVideoBlob API to receive video and intelligent data, completing the rendering of scenes. recvCommnBlob API primarily receives common user's functional data which needs no rendering in video, for example, identification results, snapshot optimization data etc.

## 3.2. High Performance Rendering Engine

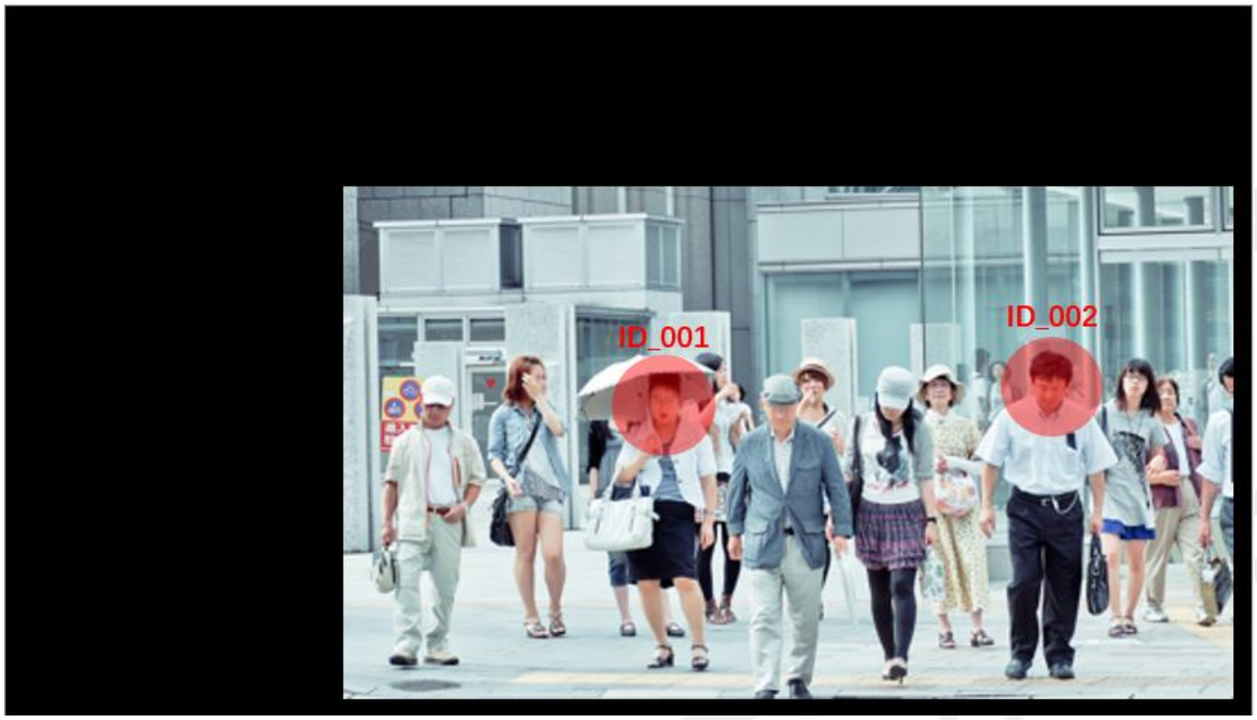
Video data usually come in yuv format. The intelligent data will be parsed into bounding boxes, images, tests and so on. All the above are regarded as primitive for image rendering. For BOOTPRINT Kit, intelligent video rendering usually composes of one video primitive and multiple intelligent primitive.

Rendering engine provides Scene-View program mode to complete drawing. User only need to add Node to Scene and configure the attributes of the node. The View will automatically display the results of rendering. Furthermore, Node support sub-nodes. The attributes of child nodes depends on the attributes of the parent nodes. For example, position attributes in the child node is relative to the position of the parent node.

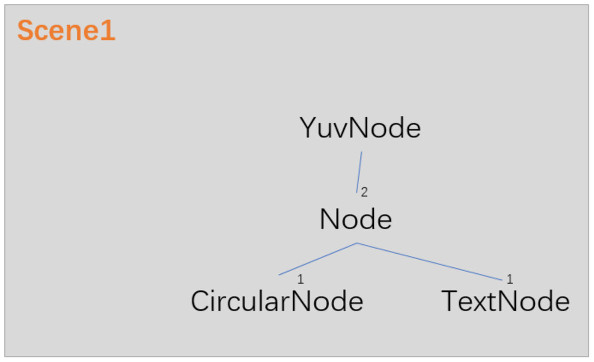


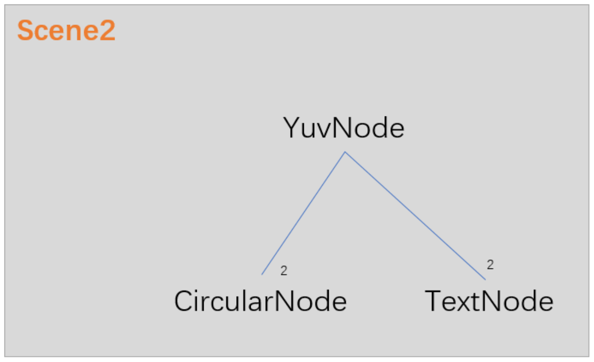


For example, below demonstrates the demo of one of the intelligent video player.



There are two possible structures satisfying the requirements supplied to Scene as demonstrated below:





About drawing engine, please refer to https://github.com/dolphin-run/sg-lite for more details.

## 3.3. BOOTPRINT Kit Player Implementation

The main structure of the player code is a Scene-View model plus a main function. Scene defines the object for rendering and View serves as the window display. Scene also need to load a data source plugin (here we use lib96boardstream.so) to obtain intelligent data and camera input from the CP of BOOTPRINT Kit.

### 3.3.1. Project Directory Structure:

```
----3rdparty
      |----glfw
      |----hbipc
      |----protobuf
      |----sglite
----plugin
      |----include
      |----filestream
      |----96boardstream
----XppClientMini
      |----pb
      |----makefile
```

3rdparty: includes 3rd party libraries:

- **glfw**: 3rd party library for drawing engine window
- **sglite**: Drawing engine, used for intelligent video rendering;
- **protobuf**: ver. 3.6.1, used for parsing intelligent frame. When intelligent frame is received, this is used to parse intelligent data. Protobuf is also used to implement input stream plugin.

- **hbipc**: Used to obtain intelligent data from CP. Used to implement BOOTPRINT Kit input stream plugin.

plugin: includes Input Stream Plugin:

- **include**: The standard API definition of plugin. When implementing plugin, define macro BUILDING_PLUGIN. Demo application will utilize this API when starting the plugin but macro BUILDING+PLUGIN does not need to be defined.
- **filestream**: Local record replay plugin, used to simulate devices for debugging. User can also store intelligent data in files and use this plugin to feed the intelligent data back.;
- **96boardstream**: BOOTPRINT Kit stream plugin. Utilizes hbipc and read_frame to obtain intelligent data and yuv video.

XppClientMini: includes Source Code:

- **pb**: The received intelligent frame pb message definition and C++ version parsing file. Used to parse pb messages. User must ensure CP side pb message definition matches with the definition here. If there is any modification, the corresponding C++ file must be generated.
- **Makefile**: Project compile Makefile
- **Other**: Other demo application source code

The implementation of demo application is described below:

### 3.3.2. Scene Implementation

Scene inherits SGScene, and implements InterfaceMsgContainer API: recvVideoBlob for obtaining intelligent video; recvCommnBlob for obtaining common data.

startStream starts streaming and loads data source plugin （lib96boardstream.so) to configure the receiver of data. (The receiver is the Scene itself since Scene implements InterfaceMsgContainer API). After the thread receives the notification from Scene to stop, it will kill the stream and unload the plugins.

```
class SmartScene : public SGScene, public InterfaceMsgContainer
```

After obtaining the video stream, the data processing can begin. Please refer to recvVideoBlob for the detailed process:

```
void SmartScene::recvVideoBlob(const PureVideoBlob * video, const SmartDataBlob *
data)
```

The input of recvVideoBlob is the video frame and intelligent frame. drawVideo will render the video. The code following is parsing the pb of intelligent frame and draw primitive according to field attributes. The Box and Points attribute of target is parsed to draw bounding box and human skeleton. The bounding box is generated according to different bounding box types.

Note: During video play, since video frame always exists (while intelligent frame can be empty), intelligent frame primitive can be overwhelming. Even each of the 24 frames in a single second can have various intelligent frame primitive. Clearing and redistributing nodes frequently is doomed to degrade performance. The Nodes in Scene is thus distributed ahead of time from node pool. Every time the node attributes are modified will reuse the nodes distributed.

### 3.3.3. View

The implementation of View depends on platform window system which is usually universal and requires no modification. The only caution is that, in View, there are parameters that can be controlled by camera, such as view.viewCamera()→setFillMode(true), which can control if view is showed in panoramic.

### 3.3.4. Demo Main Application

Main function is used to setup Scene and View; bind Scene to certain View and configure the data source plugin name. It will

then start streaming. Since View is the window application and depends on platform, main function implementation requires extra attention to initialization, exit, and loops and so on.