```c
#include <nuttx/config.h>
#include <unistd.h>
#include <stdio.h>
#include <poll.h>

#include <uORB/uORB.h>
#include <uORB/topics/sensor_combined.h>

__EXPORT int px4_simple_app_main(int argc, char *argv[]);

int px4_simple_app_main(int argc, char *argv[])
{
        printf("Hello Sky!\n");

        /* subscribe to sensor_combined topic */
        int sensor_sub_fd = orb_subscribe(ORB_ID(sensor_combined));

        /* one could wait for multiple topics with this technique, just using one here */
        struct pollfd fds[] = {
                { .fd = sensor_sub_fd,   .events = POLLIN },
                /* there could be more file descriptors here, in the form like:
                 * { .fd = other_sub_fd,   .events = POLLIN },
                 */
        };

        int error_counter = 0;

        while (true) {
                /* wait for sensor update of 1 file descriptor for 1000 ms (1 second) */
                int poll_ret = poll(fds, 1, 1000);

                /* handle the poll result */
                if (poll_ret == 0) {
                        /* this means none of our providers is giving us data */
                        printf("[px4_simple_app] Got no data within a second\n");
                } else if (poll_ret < 0) {
                        /* this is seriously bad - should be an emergency */
                        if (error_counter < 10 || error_counter % 50 == 0) {
                                /* use a counter to prevent flooding (and slowing us down) */
                                printf("[px4_simple_app] ERROR return value from poll(): %d\n"
                                        , poll_ret);
                        }
                        error_counter++;
                } else {

                        if (fds[0].revents & POLLIN) {
                                /* obtained data for the first file descriptor */
                                struct sensor_combined_s raw;
                                /* copy sensors raw data into local buffer */
                                orb_copy(ORB_ID(sensor_combined), sensor_sub_fd, &raw);
                                printf("[px4_simple_app] Accelerometer:\t%8.4f\t%8.4f\t%8.4f\n",
                                        (double)raw.accelerometer_m_s2[0],
                                        (double)raw.accelerometer_m_s2[1],
                                        (double)raw.accelerometer_m_s2[2]);
                        }
                        /* there could be more file descriptors here, in the form like:
                         * if (fds[1..n].revents & POLLIN) {}
                         */
                }
        }

        return 0;
}
```

编译并进行测试，首先打开 uorb 程序，这个程序可能已经启动了，但是再次启动它不会产生问题。

```
uorb start
```

启动传感器：

```
sh /etc/init.d/rc.sensors
```

启动自定义程序（注意此处与上面的不同，结尾添加了一个”&”符合）：

```
px4_simple_app &
```

二者的区别在于 uorb 和传感器程序都似乎 Daemon（守护程序，或称后天程序）。这使得即使我们忘了加“&”，我们依然可以启动、停止它们，而不会失去对 NuttShell 的控制。

自定义的程序现在就好快速输出一堆传感器信息：

```
[px4_simple_app] Accelerometer:   0.0483        0.0821        0.0332
[px4_simple_app] Accelerometer:   0.0486        0.0820        0.0336
[px4_simple_app] Accelerometer:   0.0487        0.0819        0.0327
[px4_simple_app] Accelerometer:   0.0482        0.0818        0.0323
[px4_simple_app] Accelerometer:   0.0482        0.0827        0.0331
[px4_simple_app] Accelerometer:   0.0489        0.0804        0.0328
```

因为它不是一个 daemon，我们没有办法去停止它。要么让它继续运行，要么输入以下命令重启飞控：

```
reboot
```

或者按飞控板上的复位按钮。本文后面的章节会讲如何将一个程序转换为一个 daemon。

步骤六：限定订阅频率

测试程序会将传感器信息以数据洪流般淹没窗口，事实上我们往往不需要全速率的数据，要跟上全速率的数据往往会将整个系统变慢。

将数据速率改为 1Hz 只需添加如下行：

```
orb_set_interval(sensor_sub_fd,1000);
```

对于我们的程序，在下面的这一行（大约第 55 行）下面添加该行即可：

```
/* subscribe to sensor_combined topic */

int sensor_sub_fd = orb_subscribe(ORB_ID(sensor_combined));
```

现在控制台就会以 1Hz 的速率输出传感器信息。

步骤七：发布信息

要使用计算后的输出，下一步就是发布结果。如果我们使用一个已知 mavlink 程序会将其传递到地面站软件的的 topic，我们甚至可以在地面站中查看结果。为达到此目的，让我们拦截 attitude（姿态）topic。

接口很简单：初始化将要发布的 topic 结构体，并通告它。

```
#include <uORB/topics/vehicle_attitude.h>
/* advertise attitude topic */
struct vehicle_attitude_s att;
memset(&att,0,sizeof(att));
int att_pub_fd = orb_advertise(ORB_ID(vehicle_attitude),&att);
```

在主循环中，当信息准备好后发布它：

```
orb_publish(ORB_ID(vehicle_attitude), att_pub_fd,&att);
```

修改后的完整程序如下：

```c
#include <stdio.h>
#include <poll.h>

#include <uORB/uORB.h>
#include <uORB/topics/sensor_combined.h>
#include <uORB/topics/vehicle_attitude.h>

__EXPORT int px4_simple_app_main(int argc, char *argv[]);

int px4_simple_app_main(int argc, char *argv[])
{
        printf("Hello Sky!\n");

        /* subscribe to sensor_combined topic */
        int sensor_sub_fd = orb_subscribe(ORB_ID(sensor_combined));
        orb_set_interval(sensor_sub_fd, 1000);

        /* advertise attitude topic */
        struct vehicle_attitude_s att;
        memset(&att, 0, sizeof(att));
        int att_pub_fd = orb_advertise(ORB_ID(vehicle_attitude), &att);

        /* one could wait for multiple topics with this technique, just using one here */
        struct pollfd fds[] = {
                { .fd = sensor_sub_fd,   .events = POLLIN },
                /* there could be more file descriptors here, in the form like:
                 * { .fd = other_sub_fd,   .events = POLLIN },
                 */
        };

        int error_counter = 0;

        while (true) {
                /* wait for sensor update of 1 file descriptor for 1000 ms (1 second) */
                int poll_ret = poll(fds, 1, 1000);

                /* handle the poll result */
                if (poll_ret == 0) {
                        /* this means none of our providers is giving us data */
                        printf("[px4_simple_app] Got no data within a second\n");
                } else if (poll_ret < 0) {
                        /* this is seriously bad - should be an emergency */
                        if (error_counter < 10 || error_counter % 50 == 0) {
                                /* use a counter to prevent flooding (and slowing us down) */
                                printf("[px4_simple_app] ERROR return value from poll(): %d\n"
                                        , poll_ret);
                        }
                        error_counter++;
                } else {

                        if (fds[0].revents & POLLIN) {
                                /* obtained data for the first file descriptor */
                                struct sensor_combined_s raw;
                                /* copy sensors raw data into local buffer */
                                orb_copy(ORB_ID(sensor_combined), sensor_sub_fd, &raw);
                                printf("[px4_simple_app] Accelerometer:\t%8.4f\t%8.4f\t%8.4f\n",
                                        (double)raw.accelerometer_m_s2[0],
                                        (double)raw.accelerometer_m_s2[1],
                                        (double)raw.accelerometer_m_s2[2]);

                                /* set att and publish this information for other apps */
                                att.roll = raw.accelerometer_m_s2[0];
                                att.pitch = raw.accelerometer_m_s2[1];
                                att.yaw = raw.accelerometer_m_s2[2];
                                orb_publish(ORB_ID(vehicle_attitude), att_pub_fd, &att);
                        }
                        /* there could be more file descriptors here, in the form like:
                         * if (fds[1..n].revents & POLLIN) {}
                         */
                }
        }

        return 0;
}
```

运行最终的程序，这次需要先启动一些别的程序：

```
uorb start
sh /etc/init.d/rc.sensors
mavlink start -d /dev/ttyS1 -b 115200
px4_simple_app &
```

如果你启动地面站软件 QGroundControl 或者 Mission Planner ,你可以实时观测传感器信息的曲线图（ Main Menu：Main Widget -> Realtime Plot ），这些信息正是你的程序发布的。

本章节包含了用户自定义 PX4 飞控程序的所有要点。注意飞控中所有 topic 的列表链接如下：

https://github.com/PX4/Firmware/tree/master/src/modules/uORB/topics

你也可以在工程中 “/src/modules/uORB/topics”目录下找到一系列头文件，这些头文件就包含了所有的 topic。

创建 **Daemon** 程序（后台程序）

参考链接：http://pixhawk.org/dev/px4_daemon_app

在 Unix 和其他多任务操作系统中，daemon(*/ˈdeɪmən/ or /ˈdiːmən/*)程序是指作为一个后台进程运行的计算机程序，而不是由用户直接控制的程序。daemon 概念的好处是它不需被用户或者 shell 发送到后台就能被启动，并且当它在运行时可以通过 shell 查询它的状态，它也可以被终止。

步骤一：创建一个小的普通程序

```
__EXPORT int px4_daemon_app_main(int argc,char*argv[]);
int px4_daemon_app_main(int argc,char*argv[])
{
        while(true){
```

```
                warnx("Hello Daemon!\n");
                sleep(1);
        }
        return0;
    }
```

这个程序的问题很明显：如果启动它时没有加“&”，它会阻塞 shell（Nuttx 不会阻塞，支持 Ctrl + Z /fg /bg ）。为了规避这个问题，下面的部分会将这个程序转换为一个 daemon。

步骤二：创建一个 Daemon 控制函数

主函数现在由一个 daemon 控制函数代替，主函数中的原来的部分现在有一个后台任务（task）/线程（thread）来代替。

```
#include <systemlib/systemlib.h>
..
__EXPORT int px4_daemon_app_main(int argc,char*argv[]);
..
int mavlink_thread_main(int argc,char*argv[]);
..
int mavlink_thread_main(int argc,char*argv[])
{
        while(true){
                warnx("Hello Daemon!\n");
                sleep(1);
                if(thread_should_exit)break;
        }

        return0;
}
..
int px4_daemon_app_main(int argc,char*argv[])
{
        if(argc <1)
                usage("missing command");

        if(!strcmp(argv[1],"start")){

                if(thread_running){
                        warnx("daemon already running\n");
```

```
                        /* this is not an error */
                        exit(0);
                }

                thread_should_exit =false;
                daemon_task = task_spawn_cmd("daemon",
                                             SCHED_RR,
                                             SCHED_PRIORITY_DEFAULT,
                                        4096,
                                             px4_daemon_thread_main,

        (argv)?(constchar**)&argv[2]:(constchar**)NULL);
                thread_running =true;
                exit(0);
        }

        usage("unrecognized command");
        exit(1);
    }
```

这个会启动一个拥有 4096 个字节的栈，并将非 daemon 特殊指令行选项传递给后台主函数。一个典型的调用如下：

```
px4_daemon_app start
```

上述代码并不报告一个状态，并且无法防止程序被多次启动。

步骤三：添加终止/状态命令和安全措施

如下程序添加了终止和状态命令以及安全措施：

```
    /**
 * @file px4_daemon_app.c
 * daemon application example for PX4 autopilot
 *
 * @author Example User <mail@example.com>
 */

#include <nuttx/config.h>
#include <nuttx/sched.h>
```