本说明针对 **Firmware v1.5.2**

1. 添加流程说明

(1) 在Firmware/msg下新建uORB的成员变量，eg: xxx.msg

(2) 在Firmware/msg/CMakeLists.txt中添加话题的xxx.msg ，作为cmake的编译索引。

2. 添加话题示例

(1) 这里，在Firmware/msg下新建`fantasy.msg`。内容为：

```
uint64 hehe1
uint64 hehe2
uint64 hehe3
uint64 hehe4

# TOPICS fantasy huihui jxf
```

> 关于数据类型的定义这里有两点需要说明：
>
> > ① 数据的类型是unit8/16/32/64（不是uint8*t*）、*bool*、*float32*、*float32[n]*（*n*为数组元素个数）。*uint8*在生成的头文件中表示为*uint8*t，float32在生成的头文件中表示为float。
> > ② 数据不需要用分号“；”隔开，换行即可

(2) Firmware/msg中的CMakeLists.txt中添加：`fantasy.msg`

```
  uavcan_parameter_value.msg
  ulog_stream.msg
  ulog_stream_ack.msg
  vehicle_attitude.msg
  vehicle_attitude_setpoint.msg
  vehicle_command_ack.msg
  vehicle_command.msg
  vehicle_control_mode.msg
  vehicle_force_setpoint.msg
  vehicle_global_position.msg
  vehicle_global_velocity_setpoint.msg
  vehicle_gps_position.msg
  vehicle_land_detected.msg
  vehicle_local_position.msg
  vehicle_local_position_setpoint.msg
  vehicle_rates_setpoint.msg
  vehicle_status.msg
  vision_position_estimate.msg
  vtol_vehicle_status.msg
  wind_estimate.msg
  vehicle_roi.msg
  mount_orientation.msg
  collision_report.msg
  low_stack.msg
  ca_trajectory.msg
+ fantasy.msg
  )
```

> 注意：
>
> **# TOPICS** `#` 号和`TOPICS`中间有一个空格。
>
> 一个消息定义就可以用于多个独立的主题。

3. 原理说明

`xxx.msg`为成员；
`# TOPICS`为话题的定义；

在编译的时候，通过**genmsg_cpp**自动生成一定结构的代码，再通过CMakeLists.txt进行编译，所以在编译一遍后，才能具体看到所定义的话题成员。

上面的`fantasy.msg`编译后生成的`/uORB/topics/fantasy.h`文件主要内容如下：

> 注意：
>
> > 每一个生成的C/C++结构体中，会多出一个uint64_t timestamp 字段。这个变量作为话题运行的时间戳，用于将消息记录到日志当中。
> >
> > 常有人说uORB文件夹下没有topics文件夹。确实Firmware/src/modules/uORB目录下是没有topics文件夹的。但是如果是FMUv2或FMUv3系列的飞控板，关注的是编译得到的目录
> > 对于Windows用户，Firmware/build_px4fmu-v2_default，这里面有你需要的东西。
> > 对于Ubuntu用户，在终端进行编译的话请查看~/src/Firmware/build_px4fmu-v2_default，使用Qt进行编译的请查看~/src/Firmware-Build。
> > 以上都是影子构建的表现。

4. 相关函数说明

主要是关于uORB函数的简单介绍，详细说明请点击传送门。

- 公告

```
orb_advert_t orb_advertise(const struct orb_metadata *meta, const void *data)
```

- 发布

```
int  orb_publish(const struct orb_metadata *meta, orb_advert_t handle, const void *data)
```

- 订阅

```
int  orb_subscribe(const struct orb_metadata *meta)
```

- 复制

```
int  orb_copy(const struct orb_metadata *meta, int handle, void *buffer)
```

参数说明：

orb_advert_t ：空指针 handle
const struct orb_metadata *meta ：话题ID
const void *data ：相关数据类型指针
话题之间的发布订阅依赖于handle进行相关性的传递，话题的ID和结构通过`# TOPICS fantasy`来定义；

> 注意：在公告和发布时用的是handle指针，订阅和复制用的是整形。

5. 发布订阅示例

示例在 px4_simple_app.c上进行测试

```c
/**
 * @file px4_simple_app.c
 * Minimal application example for PX4 autopilot
 *
 * @author Fantasy <fantasyjxf@gmail.com>
 */

#include <px4_config.h>
#include <px4_tasks.h>
#include <px4_posix.h>
#include <unistd.h>
#include <stdio.h>
#include <poll.h>
#include <string.h>
#include <math.h>

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

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

int px4_simple_app_main(int argc, char *argv[])
{
    PX4_INFO("Hello Fantasy!");

    /*定义话题结构*/
    struct fantasy_s test;
    /*初始化数据*/
    memset(&test, 0, sizeof(test));

    /*公告主题*/
    /*test_pub 为handle指针*/
    orb_advert_t test_pub = orb_advertise(ORB_ID(fantasy), &test);

    /*test数据赋值*/
     test.hehe1 = 2;
     test.hehe2 = 3;
     test.hehe3 = 3;

    /*发布测试数据*/
    orb_publish(ORB_ID(fantasy), test_pub, &test);

    /*订阅数据，在copy之前，必须要订阅*/
    /*test_sub_fd为handle*/
    int test_sub_fd = orb_subscribe(ORB_ID(fantasy));

    struct fantasy_s data_copy;
```

```
    /*copy数据*/
    orb_copy(ORB_ID(fantasy), test_sub_fd, &data_copy);

    /*打印*/
    PX4_WARN("[px4_simple_app] GanTA:\t%8.4f\t%8.4f\t%8.4f",
                        (double)data_copy.hehe1,
                        (double)data_copy.hehe2,
                        (double)data_copy.hehe3);

    return 0;
}
```

测试效果：

参考：

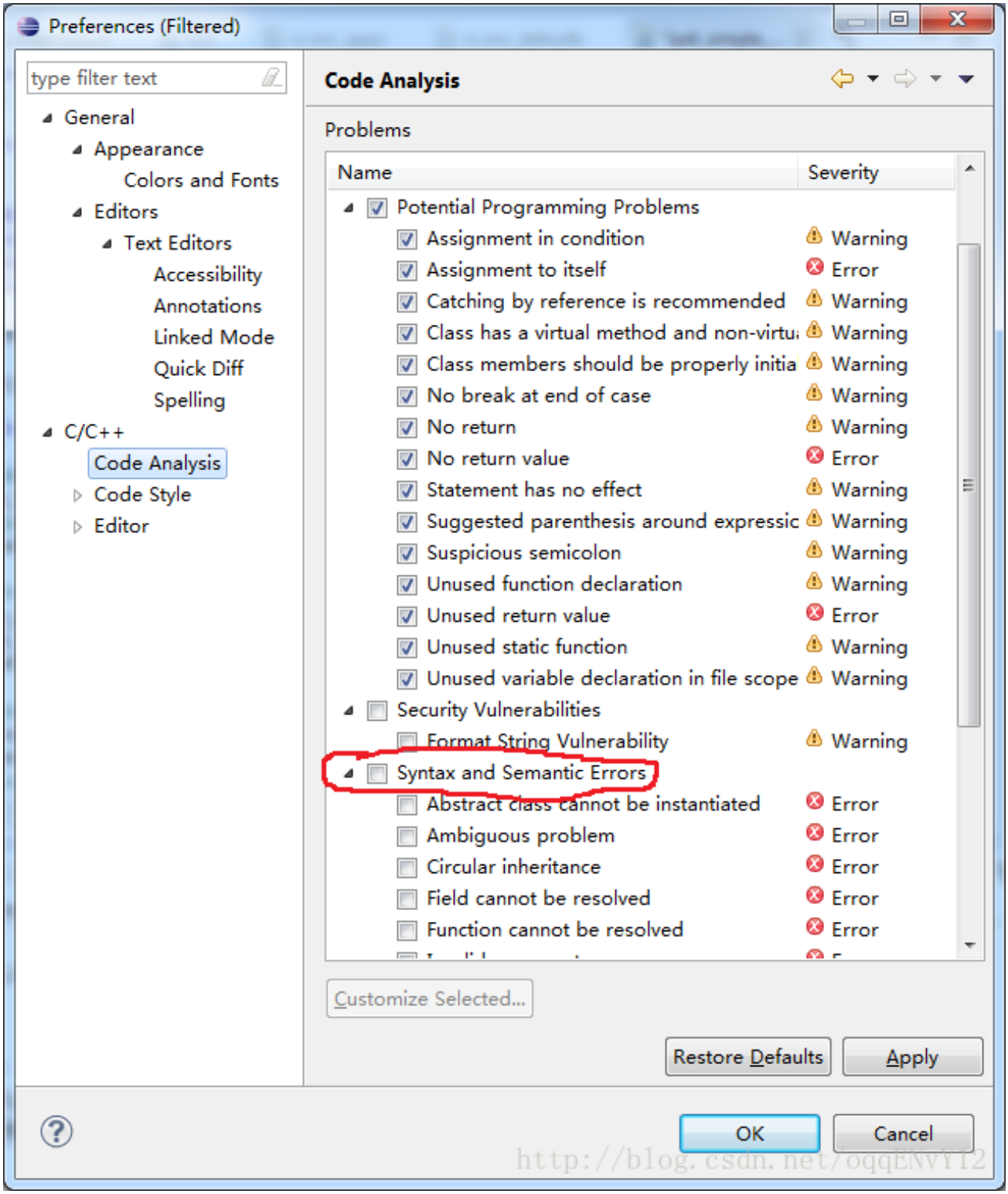
① 官网进程间通信开发指南

② 基于Firmware 1.2.0的uORB主题添加

③ PX4中文维基 - uORB消息机制

④ PX4中文维基 - 编写应用程序

最后赠送一个可能解决Eclipse环境下代码中到处都是奇怪警告、错误的方法



我反正是把这一栏取消勾选了，上面一栏可以留下的，代码问题分析。