优蓝科技
ULANTEC

汽车总线测试编程 API 说明

版本：V1.2.7

天津优蓝科技有限公司

2022 年 6 月

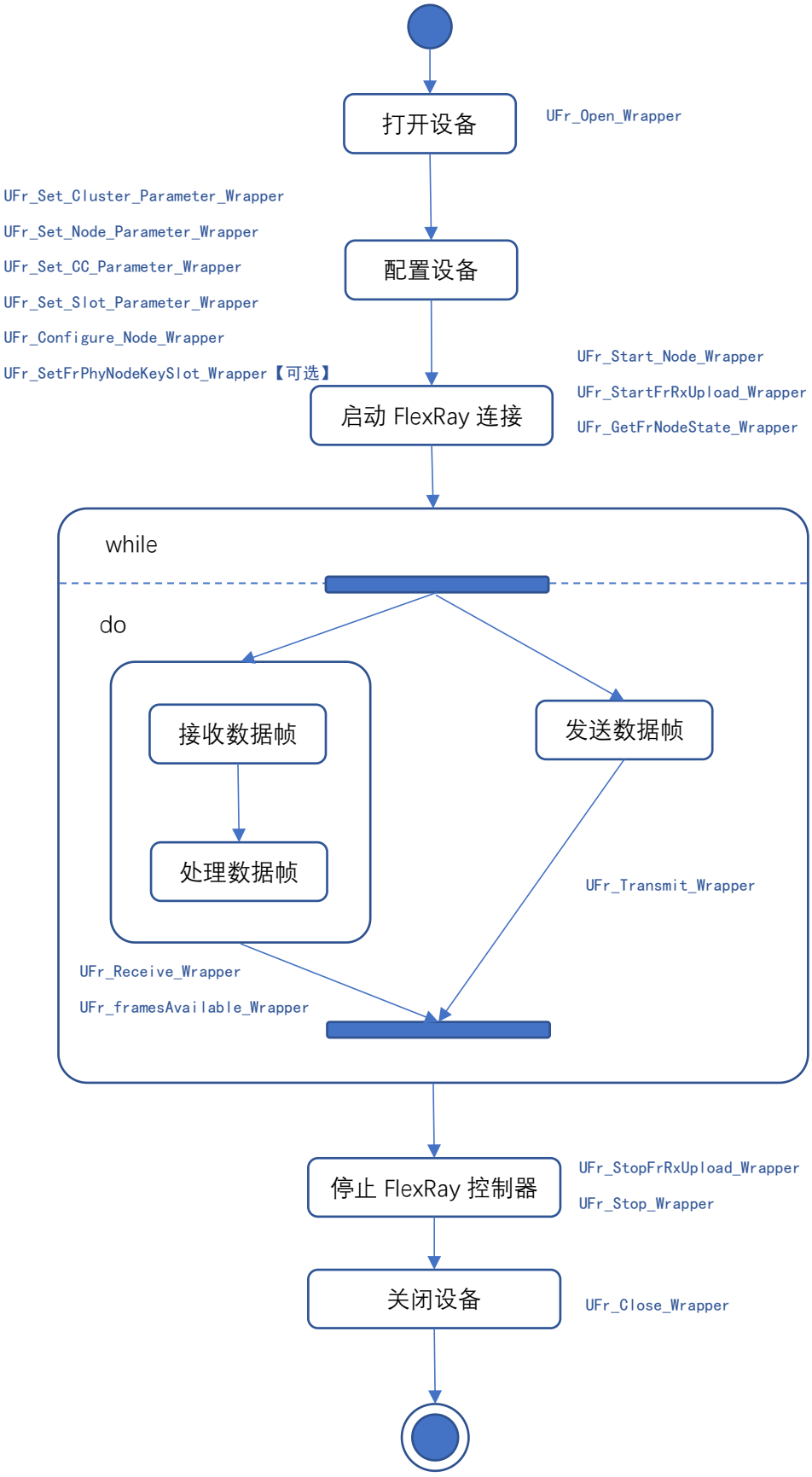
目录

1. FlexRay 编程 API 说明

本章介绍 FlexRay 网络模块编程相关的数据结构、枚举类型和函数说明，提供基于 QT 开发环境的测试例程作为编程参考。

1.1 函数基本调用流程

下图描述 FlexRay 编程函数的基本调用流程，具体函数使用方法参考后面的函数具体说明。


打开设备
UFr_Open_Wrapper
UFr_Set_Cluster_Parameter_Wrapper
UFr_Set_Node_Parameter_Wrapper
UFr_Set_CC_Parameter_Wrapper
UFr_Set_Slot_Parameter_Wrapper
UFr_Configure_Node_Wrapper
UFr_SetFrPhyNodeKeySlot_Wrapper【可选】
配置设备
UFr_Start_Node_Wrapper
UFr_StartFrRxUpload_Wrapper
UFr_GetFrNodeState_Wrapper
启动 FlexRay 连接
while
do
接收数据帧
处理数据帧
发送数据帧
UFr_Transmit_Wrapper
UFr_Receive_Wrapper
UFr_framesAvailable_Wrapper
停止 FlexRay 控制器
UFr_StopFrRxUpload_Wrapper
UFr_Stop_Wrapper
关闭设备
UFr_Close_Wrapper

1.2 初始化

1.2.1 打开设备

1.2.1.1 枚举定义

1.2.1.1.1 CMD_RETURN_STATUS

这个枚举类型定义了 API 函数库的函数返回值，适用于本文所列的所有函数返回值，包括 FlexRay、CAN 和 Fibex 操作函数，具体定义如下：

```
typedef enum
{
    CMD_SUCCESS = 0,
    CMD_FAIL_STATUS,
    CMD_FAIL_TIMEOUT,
    CMD_FAIL_SOCKET,
    CMD_FAIL_EXIT,
    CMD_FAIL_PARAMETER,
    CMD_FAIL_INSTANCENOEXIST,
    CMD_FAIL_OPENFIBEX,
    CMD_FAIL_PROCEDURE,
    CMD_FAIL_INVALIDFILE,
    CMD_FAIL_SET_NODE_ATTIBUTE,
    CMD_FAIL_SET_NODE_PARAMETER,
    CMD_FAIL_SET_CC_PARAMETER,
    CMD_FAIL_SIGNAL_NO_EXIST,
    CMD_FAIL_NO_SPACE,
    CMD_FAIL_UNKNOW_SIGNAL,
    CMD_FAIL_UNKNOW_FRAME,
} CMD_RETURN_STATUS;
```

成员说明：

CMD_SUCCESS

函数执行成功。

CMD_FAIL_STATUS

设备返回的状态失败，如果返回该值，说明命令已经发送给设备，但是设备返回执行结果是失败的。

CMD_FAIL_TIMEOUT

函数执行超时，如果返回该值，说明命令发送给设备，但是设备没有返回执行结果，可能原因包括：

1）网络物理连接有问题，检查网络物理连接是否正确；

2）网络访问有问题，检查主机 IP 地址和设备 IP 地址是否可以互联，可以用 ping 命令检查网络连接。

3）接收函数如果返回超时，表明当前缓冲区在指定的超时时间内没有消息到达。

CMD_FAIL_SOCKET

本地建立 socket 失败，命令采用 UDP 通讯方式，如果建立 socket 失败会产生这个错误。

CMD_FAIL_EXIT

程序退出时接收函数可能的返回值，由于接收函数可以采用阻塞接收方式，程序退出时提前退出接收函数的阻塞状态时返回这个值。

CMD_FAIL_PARAMETER

函数输入参数有误。

CMD_FAIL_INSTANCENOEXIST

实例号不存在，UFr_Open_Wrapper 应返回大于 0 的有效实例号。

CMD_FAIL_OPENFIBEX

打开 Fibex 数据库文件失败，可能由于文件格式有错误或 Fibex 版本不兼容。

CMD_FAIL_PROCEDURE

函数调用流程错误，需要首先调用 UFr_Open_Wrapper 函数并执行成功后再调用其他函数。

CMD_FAIL_INVALIDFILE

无效文件。

CMD_FAIL_SET_NODE_ATTIBUTE

设置节点属性失败。

CMD_FAIL_SET_NODE_PARAMETER

设置节点参数失败。

CMD_FAIL_SET_CC_PARAMETER

设置 CC 控制器参数失败。

CMD_FAIL_SIGNAL_NO_EXIST

加载的数据库文件中不存在该信号的定义。

CMD_FAIL_NO_SPACE

要编码信号的缓冲区空间不足，可能由于缓冲区长度定义错误，或者信号定义有误。

CMD_FAIL_UNKNOW_SIGNAL

加载的 fibex 文件没有该信号的定义。

CMD_FAIL_UNKNOW_FRAME

加载的 fibex 文件没有该帧的定义。

1.2.1.2 UFr_Open_Wrapper

这个函数用于打开 FlexRay 设备，应该在进行所有其他操作之前调用。

```
sint8 UFr_Open_Wrapper(char *FrIPaddr)
```

参数说明：

FrIPaddr

【IN】要打开设备的以太网地址。

返回值：

如果函数执行成功返回值是大于 0 的设备实例号，后面的其他函数操作都要将此实例号作为输入参数来标明打开设备的唯一实例。否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值的负值及说明。

备注：

例子：

```
char targetIP[] = "192.168.0.7";
//打开设备
instance = UFr_Open_Wrapper(targetIP);
if(instance <= 0)
{
    printf("Open flexray device fail!\r\n");
    return 0;
}
printf("Open flexray device OK[%s]!\r\n",targetIP);
```

1.3 配置

1.3.1 配置 clusters

1.3.1.1 结构体定义

1.3.1.1.1 FlexrayClusterParameter_t

这个结构体定义了 FlexRay 协议规范中有关全局 Cluster 参数的数据成员，具体定义如下：

```
typedef struct fcpt
{
    uint8_t gColdstartAttempts;
    uint8_t gdActionPointOffset;
    uint8_t gdCASRxLowMax;
    uint8_t gdDynamicSlotIdlePhase;
    uint8_t gdMinislot;
    uint8_t gdMinislotActionPointOffset;
    uint16_t gdStaticSlot;
    uint8_t gdSymbolWindow;
    uint8_t gdTSSTransmitter;
    uint8_t gdWakeupSymbolRxIdle;
    uint8_t gdWakeupSymbolRxLow;
```

```
    uint16_t gdWakeupSymbolRxWindow;
    uint8_t gdWakeupSymbolTxIdle;
    uint8_t gdWakeupSymbolTxLow;
    uint32  gListenNoise;
    uint8_t gNetworkManagementVectorLength;
    uint16_t gMacroPerCycle;
    uint8_t gMaxWithoutClockCorrectionFatal;
    uint8_t gMaxWithoutClockCorrectionPassive;
    uint16_t gNumberOfMinislots;
    uint16_t gNumberOfStaticSlots;
    uint16_t gOffsetCorrectionStart;
    uint8_t gPayloadLengthStatic;
    uint8_t gSyncNodeMax;
} FlexrayClusterParameter_t;
```

成员说明：

gColdstartAttempts

允许簇内节点通过启动调度同步尝试启动簇的最大次数，有效值为 2~31 次。

gdActionPointOffset

动作点偏离静态时隙或符号窗的起始点的宏节拍数量，有效值为 1~63MT。

gdCASRxLowMax

CAS 接收窗口上限，有效值为 67~99gdBit。

gdDynamicSlotIdlePhase

一个动态时隙内的空闲阶段持续时间，有效值为 0~2 个微时隙。

gdMinislot

微时隙的持续时间，有效值为 2~63MT。

gdMinislotActionPointOffset

微时隙动作点偏离微时隙起始点的宏节拍数量，有效值为 1~31MT。

gdStaticSlot

静态时隙的持续时间，有效值为 4~661MT。

gdSymbolWindow

符号窗的持续时间，有效值为 0~142MT。

gdTSSTransmitter

传输起始序列（TSS）的位数，有效值为 3~15gdBit。

gdWakeupSymbolRxIdle

节点对收到唤醒符号的“空闲”部分持续时间进行测试时所使用的位数。持续时间 等于(gdWakeupSymbolTxIdle-gdWakeupSymbolTxLow)/2 再减去一个安全部分，有效值为 14~59gdBit。

gdWakeupSymbolRxLow

节点对收到唤醒符号的 LOW 部分进行测试时所使用的位数。这是接收器为检测 LOW 部分必须接收到的零的位数。该持续时间等于 gdWakeupSymbolTxLow 减去一个安全部分，有效值为 11~59gdBit。

gdWakeupSymbolRxWindow

检测唤醒所使用的窗口的大小。唤醒检测需要一段 LOW 和空闲时间(从一个 WUS)，在这个大小的窗口内，能够完整检测一段 LOW 的时间(另一个 WUS)。这个持续时间等于 gdWakeupSymbolTxIdle+2×gdWakeupSymbolTxLow，再加上一个安全部分，有效值为 76~301gdBit。

gdWakeupSymbolTxIdle

节点发送唤醒符的“空闲”部分所使用的位数，持续时间等于 18μs，有效值为 45~180gdBit。

gdWakeupSymbolTxLow

节点发送唤醒符的“LOW”部分所使用的位数。持续时间等于 6μs，有效值为 15~60gdBit。

gListenNoise

在存在噪声的情况下，启动监听超时值和唤醒监听超时值的上限。它是节点参数 pdListenTimeout 的倍数，有效值为 2~16。

gNetworkManagementVectorLength

网络管理向量的长度，有效值为 0~12 字节。

gMacroPerCycle

一个通信周期的总宏节拍数量，有效值为 10~16000MT。

gMaxWithoutClockCorrectionFatal

定义缺失时钟修正项的连续偶/奇循环对数目，缺失时钟修正项将导致协议从正常有源状态或正常无源状态过渡到停止状态，有效值为 1~15 个偶/奇循环对。

gMaxWithoutClockCorrectionPassive

定义缺失时钟修正项的连续偶/奇循环对数目，缺失时钟修正项将导致协议从正常有源状态过渡到正常无源状态。需要注意的是 gMaxWithoutClockCorrectionPassive≤gMaxWithoutClockCorrectionFatal，有效值为 1~15 个偶/奇循环对。

gNumberOfMinislots

动态段的微时隙数，有效值为 0~7986 个。

gNumberOfStaticSlots

静态段的静态时隙数，有效值为 2~1023 个。

gOffsetCorrectionStart

NIT 内偏差修正相位的起点，表示周期开始后的第几个宏节拍开始修正，有效值 9~15999MT。

gPayloadLengthStatic

静态帧的有效负载长度，有效值为 0~127 个字。

gSyncNodeMax

可发送“同步帧指示位设置为 1”的帧的最大节点数量，有效值为 2~15 个。

1.3.1.2 UFr_Set_Cluster_Parameter_Wrapper

这个函数用于设置节点的全局 Cluster 参数，应该在 UFr_Open_Wrapper 打开设备成功后调用。

```
uint8 UFr_Set_Cluster_Parameter_Wrapper
(sint8 instance, uint8 controller_num, const FlexrayClusterParameter_t *fcp
)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】控制器号，用于表示此类设备的通道号，从 0 开始排序。

*fcp

【IN】总线参数定义结构体指针，用于配置总线参数，调用前需要预先定义该结构体变量并赋值，具体请参看结构体 FlexrayClusterParameter_t 的定义。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

FlexrayClusterParameter_t 定义的全局 Cluster 参数一般由 FlexRay 总线系统设计方提供，请根据设计方要求填写对应的数值，否则将影响 FlexRay 总线连接建立和正确通讯。

例子：

```
//定义 flexray 总线的 cluster 参数结构体
FlexrayClusterParameter_t mFlexrayClusterParameter
{
        10,   //uint8_t gColdstartAttempts;
        4,    //uint8_t gdActionPointOffset;
        91,   //uint8_t gdCASRxLowMax;
        1,    //uint8_t gdDynamicSlotIdlePhase;
        10,   //uint8_t gdMinislot;
        5,    //uint8_t gdMinislotActionPointOffset;
        119,  //uint16_t gdStaticSlot;
        0,    //uint8_t gdSymbolWindow;
        11,   //uint8_t gdTSSTransmitter;
        59,   //uint8_t gdWakeupSymbolRxIdle;
        50,   //uint8_t gdWakeupSymbolRxLow;
        301,  //uint16_t gdWakeupSymbolRxWindow;
        180,  //uint8_t gdWakeupSymbolTxIdle;
        60,   //uint8_t gdWakeupSymbolTxLow;
        2,    //uint32_t gListenNoise;
        0,    //uint8_t gNetworkManagementVectorLength;
        10000,//uint16_t gMacroPerCycle;
        8,    //uint8_t gMaxWithoutClockCorrectionFatal;
        6,    //uint8_t gMaxWithoutClockCorrectionPassive;
        1,    //uint16_t gNumberOfMinislots;
        80,   //uint16_t gNumberOfStaticSlots;
        9993,//uint16_t gOffsetCorrectionStart;
        10,   //uint8_t gPayloadLengthStatic;
        10    //uint8_t gSyncNodeMax;
};
```

```
//配置 cluster 参数
    if(UFr_Set_Cluster_Parameter(instance, 0, &mFlexrayClusterParameter) !=
CMD_SUCCESS)
    {
        printf("Set_Cluster_Parameter fail!\r\n");
        return 0;
    }
```

1.3.2 配置 node

1.3.2.1 结构体定义

1.3.2.1.1 FlexrayNodeParameter_t

这个结构体定义了 FlexRay 协议规范中有关节点参数的数据成员，具体定义如下：

```
typedef struct fnpt
{
    uint32  pdListenTimeout;
    uint8_t pMacroInitialOffsetA;
    uint8_t pMacroInitialOffsetB;
    uint8_t pPayloadLengthDynMax;
    uint8_t pAllowHaltDueToClock;
    uint8_t pAllowPassiveToActive;
    uint8_t pClusterDriftDamping;
    uint16_t pdAcceptedStartupRange;
    uint16_t pDelayCompensationA;
    uint16_t pDelayCompensationB;
    uint16_t pKeySlotId;
    uint8_t pKeySlotUsedForStartup;
    uint8_t pKeySlotUsedForSync;
    uint16_t pLatestTx;
    uint8_t pMicroInitialOffsetA;
    uint8_t pMicroInitialOffsetB;
    uint32  pMicroPerCycle;
    uint32  pdMaxDrift;
    uint8_t pMicroPerMacroNom;
    uint16_t pRateCorrectionOut;
    uint16_t pOffsetCorrectionOut;
    uint8_t pSingleSlotEnabled;
    uint8_t pWakeupChannel;
```

```
    uint8_t pWakeupPattern;
    uint16_t pDecodingCorrection;
    uint16_t keySlotHeaderCrc;
    uint8_t pExternOffsetCorrection;
    uint8_t pExternRateCorrection;
} FlexrayNodeParameter_t;
```

成员说明：

pdListenTimeout

启动监听超时和唤醒监听超时值，有效值为 1284~1283846μT。

pMacroInitialOffsetA

A 通道静态时隙边界和第二个时间基准点的下一个宏节拍边界之间的宏节拍数，有效值为 2~68MT。

pMacroInitialOffsetB

B 通道静态时隙边界和第二个时间基准点的下一个宏节拍边界之间的宏节拍数，有效值为 2~68MT。

pPayloadLengthDynMax

动态帧的最大有效负载长度，有效值为 0~127 个字。

pAllowHaltDueToClock

允许时钟同步错误过渡到停止状态标志位，有效值为 0 和 1，1 为允许。

pAllowPassiveToActive

在正常无源状态到正常有源状态的过渡之前，必须要有的有效时钟修正项的连续偶/奇循环对数量。设置为 0 表示不允许从正常无源状态过渡到正常有源状态，有效值为 0~31 个偶/奇循环对。

pClusterDriftDamping

本地用于速率修正的簇漂移阻尼系数，有效值为 0~20μT。

pdAcceptedStartupRange

集成过程中，启动帧所允许的、经扩展的测量时钟偏差范围，有效值为 0~1875μT。

pDelayCompensationA

用来补偿 A 通道接收延迟的值。它覆盖了高达 2.5μs 的假定传播延迟，有效值为 0~200μT。

pDelayCompensationB

用来补偿 B 通道接收延迟的值。它覆盖了高达 2.5μs 的假定传播延迟，有效值为 0~200μT。

pKeySlotId

用于发送启动帧、同步帧或指定单时隙帧的时隙标识符（ID），有效值为 1~1023 时隙。

pKeySlotUsedForStartup

指示关键时隙是否用于发送启动帧的标志，有效值为 0 和 1，若 *pKeySlotUsedForStartup* 设置为 1，则 *pKeySlotUsedForSync* 也必须设置为 1。

pKeySlotUsedForSync

指示关键时隙是否用于发送同步帧的标志，有效值为 0 和 1，若 pKeySlotUsedForStartup 设置为 1，则 pKeySlotUsedForSync 也必须设置为 1。

pLatestTx

在动态段最后可以开始帧发送的微时隙数，有效值为 0~7980。

pMicroInitialOffsetA

由 pMacroInitialOffset[CH]描述的最接近的宏节拍边界和第二时间基准点之间的微节拍数，有效值为 0~239μT。

pMicroInitialOffsetB

由 pMacroInitialOffset[CH]描述的最接近的宏节拍边界和第二时间基准点之间的微节拍数，有效值为 0~239μT。

pMicroPerCycle

一个通信周期的总微节拍数量。需要注意的是 pMicroPerCycle= gMacroPerCycle * pMicroPerMacroNom，有效值为 640~640000μT。

pdMaxDrift

两个节点以非同步时钟运行一个通信周期以上时，两者之间的最大漂移偏差，有效值为 2~1923μT。

pMicroPerMacroNom

一个宏节拍包含的微节拍数量，有效值为 40~240μT。

pRateCorrectionOut

允许最大速率修正值的大小，有效值为 2~1923μT。

pOffsetCorrectionOut

允许最大偏差修正值的大小，有效值为 2~1923μT。

pSingleSlotEnabled

启动后节点是否将进入单时隙模式标志位，有效值为 0 和 1。

pWakeupChannel

节点用于发送唤醒模式的通道，0 为 A 通道，1 为 B 通道。

pWakeupPattern

当节点进入唤醒发送状态时，形成唤醒模式的唤醒符号重复次数，有效值为 2~63 次。

pDecodingCorrection

接收器用于计算主时间基准点和次级时间基准点之间差异的值，有效值为 14~143μT。

keySlotHeaderCrc

关键帧头 CRC 校验和，应用本模块时填写 0 即可，实际校验和由设备软件自动计算。

pExternOffsetCorrection

进行主机请求的外部偏差修正时，NIT 加上或减去的微节拍数，有效值为 0~7μT。

pExternRateCorrection

进行主机请求的外部速率修正时，周期加上或减去的微节拍数，有效值为 0~7μT。

1.3.2.2 UFr_Set_Node_Parameter_Wrapper

这个函数用于设置 FlexRay 总线节点参数，应该在 UFr_Open_Wrapper 打开设备成功后调用。

```
uint8 UFr_Set_Node_Parameter_Wrapper
```

*(sint8 instance, uint8 controller_num, const FlexrayNodeParameter_t *fnp)*

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】控制器号，用于表示此类设备的通道号，从 0 开始排序。

*fnp

【IN】总线节点参数定义结构体指针，用于配置节点参数，调用前需要预先定义该结构体变量并赋值，具体请参看结构体 FlexrayNodeParameter_t 的定义。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
//初始化 flexray 总线的 node 参数
FlexrayNodeParameter_t mFlexrayNodeParameter
{
        401202,    //uint32_t pdListenTimeout;
        9,         //uint8_t pMacroInitialOffsetA;
        9,         //uint8_t pMacroInitialOffsetB;
        8,         //uint8_t pPayloadLengthDynMax;
        0,         //uint8_t pAllowHaltDueToClock;
        20,        //uint8_t pAllowPassiveToActive;
        2,         //uint8_t pClusterDriftDamping;
        212,       //uint16_t pdAcceptedStartupRange;
        1,         //uint16_t pDelayCompensationA;
        1,         //uint16_t pDelayCompensationB;
        31,        //uint16_t pKeySlotId;
        1,         //uint8_t pKeySlotUsedForStartup;
        1,         //uint8_t pKeySlotUsedForSync;
        21,        //uint16_t pLatestTx;
        23,        //uint8_t pMicroInitialOffsetA;
        23,        //uint8_t pMicroInitialOffsetB;
        400000,    //uint32_t pMicroPerCycle;
        601,       //uint32_t pdMaxDrift;
        40,        //uint8_t pMicroPerMacroNom;
        601,       //uint16_t pRateCorrectionOut;
        141,       //uint16_t pOffsetCorrectionOut;
        0,         //uint8_t pSingleSlotEnabled;
        1,         //uint8_t pWakeupChannel;
        44,        //uint8_t pWakeupPattern;
        56,        //uint16_t pDecodingCorrection;
```

```
        0,          //uint16_t keySlotHeaderCrc;
        0,          //uint8_t pExternOffsetCorrection;
        0           //uint8_t pExternRateCorrection;
};
//配置节点参数
    if(UFr_Set_Node_Parameter_Wrapper(instance, 0, &mFlexrayNodeParameter) !=
CMD_SUCCESS)
    {
        printf("UFr_Set_Node_Parameter fail!\r\n");
        return 0;
    }
```

1.3.3 配置 CC

1.3.3.1 枚举定义

1.3.3.1.1 Fr_Bitrate

这个枚举类型定义了总线速率支持的选项，FlexRay 总线标准支持 2.5M，5M，8M 和 10M 四种速率，定义如下：

```
    typedef enum
    {
        Bitrate_10M = 0,
        Bitrate_5M,
        Bitrate_2_5M,
        Bitrate_8M
    } Fr_Bitrate;
```

1.3.3.2 结构体定义

1.3.3.2.1 FlexrayCCParameter_t

这个结构体定义了通讯控制器参数的数据成员，具体定义如下：

```
    typedef struct fccpt
    {
        uint8_t freezeMode;
        uint8_t channelA;
        uint8_t channelB;
        uint8_t syncFrameFilter;
```

```
    uint8_t bitRate;
} FlexrayCCParameter_t;
```

成员说明：

freezeMode

当总线失败时是否进入冻结模式，有效值为 0 和 1，0 为关闭，1 为使能。

channelA

是否使用通道 A，0 为不使用，1 为使用。channelA 与 channelB 的组合生成了节点协议参数 pChannels 的定义。

channelB

是否使用通道 B，0 为不使用，1 为使用。

syncFrameFilter

是否启动同步帧过滤，0 为禁用，1 为使能，当前。

bitRate

总线通讯速率选择，0 对应 10Mbps，1 对应 5Mbps，2 对应 2.5Mbps，3 对应 8Mbps，可以使用枚举类型定义 Fr_Bitrate。

1.3.3.3 UFr_Set_CC_Parameter_Wrapper

这个函数用于设置 FlexRay 节点通讯控制器参数，应该在 UFr_Open_Wrapper 打开设备成功后调用。

```
uint8 UFr_Set_CC_Parameter_Wrapper
(sint8 instance, uint8 controller_num, const FlexrayCCParameter_t *fccp)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】控制器号，用于表示此类设备的通道号，从 0 开始排序。

* fccp

【IN】总线节点 CC 参数定义结构体指针，用于配置节点 CC 参数，调用前需要预先定义该结构体变量并赋值，具体请参看结构体 FlexrayCCParameter_t 的定义。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
//初始化 flexray 总线的控制器参数
FlexrayCCParameter_t mFlexrayCCParameter
{
        0,    //uint8_t freezeMode;
        1,    //uint8_t channelA;
```

```
        1,    //uint8_t channelB;
        0,    //uint8_t syncFrameFilter;
        UFR_BITRATE_2_5M    //uint8_t bitRate;

};
//配置 CC 参数
    if(Fr_Set_CC_Parameter(instance, 0, &mFlexrayCCParameter) != CMD_SUCCESS)
    {
        printf("UFr_Set_CC_Parameter fail!\r\n");
        return 0;
    }
```

1.3.4 配置 slots

1.3.4.1 结构体定义

1.3.4.1.1 FlexrayMsgBufferParameter_t

这个结构体定义了配置时隙参数的数据成员，具体定义如下：

```
typedef struct fspt
{
    MessageBufferID_t msgBufferID;
    uint16_t frameID;
    uint8_t PPI;
    uint8_t payloadLength;
    uint8_t isTx;
    uint8_t channelA;
    uint8_t channelB;
    uint8_t baseCycle;
    uint8_t repetitionCycle;
    uint8_t repeatTx;
} FlexrayMsgBufferParameter_t;
```

成员说明：

msgBufferID

该参数作为输入参数时表示用户自定义使用的 MessageBuffer ID 号，取值范围是 0~127，**注意：用户在自定义配置 MessageBuffer ID 时，应从 0 开始顺序使用，并且需要先配置静态时隙后再配置动态时隙，不能静态时隙与动态时隙穿插配置。**如果该参数为 0xFF，则表示实际使用的 MessageBuffer 由硬件设备决定，函数调用完成后，这个参数会返回实际使用的 MessageBuffer ID 号，这个 ID 号

可作为发送数据帧函数的输入参数。**推荐使用 0xFF 作为输入参数，由设备本身决定使用哪个 MessageBuffer。**

frameID

定义要发送/接收 FlexRay 帧的时隙号，配置的数值应在 Cluster 相关参数定义的范围内。

PPI

负载数据前导指示位:表明帧中有效负载的内容。1 表示有效负载为特殊内容；0 表示有效负载为一般内容。 通信循环的静态段和动态段都可以输出帧，但该指示位在这两种情况下的含义不同:在静态段发送的帧(静态帧),1 表示负载场包含网络管理向量，0 表示负载场不包含该向量（接收端的网络管理向量的值会在每个周期的结束时被更新，并且该值由本周期内所有时隙的网络管理向量部分做与运算后得出最后的数值，最大 12 个字节，实际使用时由全局 cluster 参数的 gNetworkManagementVectorLength 来定义具体字节数);在动态段发送的帧(动态帧)，1 表示负载场包含辅助性报文 ID，0 表示负载场不包含辅助性报文 ID。

payloadLength

动态时隙发送数据的负载长度，单位是字，有效值为 1~pPayloadLengthDynMax。静态数据帧的负载长度由总线参数中的 gPayloadLengthStatic 参数自动确定。

isTx

定制这个时隙是否为发送时隙，1 为发送时隙，0 为接收时隙。

channelA

是否使用通道 A，1 为使用，0 为不使用。

channelB

是否使用通道 B，1 为使用，0 为不使用。

baseCycle

发送或接收周期的起始周期数。应小于 repetitionCycle 的值。

repetitionCycle

重复周期数。与 *baseCycle* 配合使用。取值范围是 1，2，4，8，16，32，64。

repeatTx

该参数只对发送时隙有效，定义该时隙消息帧是否重复发送，1 为重复发送（即周期触发方式），0 为只发送一次（即事件触发方式）。

周期触发方式下，调用一次发送数据的函数后，控制器将在每个可以发送数据的周期（具体由 *baseCycle* 和 *repetitionCycle* 两个参数确定）上进行 FlexRay 帧发送。

事件触发方式下，调用一次发送数据的函数后，控制器只在将要到来的可以发送数据的周期上发送一次数据。

1.3.4.2 UFr_Set_Slot_Parameter_Wrapper

这个函数用于设置 FlexRay 节点的时隙参数，应该在 UFr_Open_Wrapper 打开设备成功后调用。

```
uint8 UFr_Set_Slot_Parameter_Wrapper
(sint8 instance, uint8 controller_num, FlexrayMsgBufferParameter_t
*msgBufferParameter)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】控制器号，用于表示此类设备的通道号，从 0 开始排序。

* *msgBufferParameter*

【IN and OUT】总线节点时隙参数定义结构体指针，用于配置节点时隙参数，调用前需要预先定义该结构体变量并赋值，具体请参看结构体 *FlexrayMsgBufferParameter_t* 的定义。其中的 msgBufferID 参数是作为输入参数表示用户自定义使用的 MessageBuffer ID 号，要求详见上面的 *FlexrayMsgBufferParameter_t* 结构体参数定义的 msgBufferID 部分。如果该参数为 0xFF，则表示实际使用的 MessageBuffer 由硬件设备决定，函数调用完成后，这个参数会返回实际使用的 MessageBuffer ID 号，这个 ID 号可作为发送数据帧函数的输入参数。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

配置时隙参数时可以根据实际项目需要多次调用 UFr_Set_Slot_Parameter 函数进行多个时隙的参数配置。

例子：

```
//初始化单个时隙参数
FlexrayMsgBufferParameter_t mFlexrayMsgBufferParameter
{
        0xff,//MessageBufferID_t msgBufferID;
        31, //uint16_t frameID;
        0,  //uint8_t PPI;
        10, //uint8_t payloadLength;
        1,  //uint8_t isTx;
        1,  //uint8_t channelA;
        1,  //uint8_t channelB;
        0,  //uint8_t baseCycle;   //周期偏移
        1,  //uint8_t repetitionCycle;  //重复周期数
        0,  //uint8_t repeatTx;

};
//配置时隙参数
    ret = UFr_Set_Slot_Parameter_Wrapper(instance, 0,
&mFlexrayMsgBufferParameter);
    if(ret != CMD_SUCCESS)
    {
        printf("UFr_Set_Slot_Parameter FAIL!\r\n");
        return 0;
    }
```

1.3.5 配置监控 FIFO 接收

1.3.5.1 结构体定义

1.3.5.1.1 FlerayFifoParameterFlat_t

这个结构体定义了配置 FIFO 参数的数据成员，具体定义如下：

```
typedef struct
{
    uint16_t messageIdAccVal;
    uint16_t messageIdAccMask;
    uint16_t frameIdRejVal;
    uint16_t frameIdRejMask;
    uint8_t rangeFilter1Enable;
    uint16_t rangeFilter1Upper;
    uint16_t rangeFilter1Lower;
    uint8_t rangeFilter1Mode;
    uint8_t rangeFilter2Enable;
    uint16_t rangeFilter2Upper;
    uint16_t rangeFilter2Lower;
    uint8_t rangeFilter2Mode;
    uint8_t rangeFilter3Enable;
    uint16_t rangeFilter3Upper;
    uint16_t rangeFilter3Lower;
    uint8_t rangeFilter3Mode;
    uint8_t rangeFilter4Enable;
    uint16_t rangeFilter4Upper;
    uint16_t rangeFilter4Lower;
    uint8_t rangeFilter4Mode;
} FlexrayFifoParameterFlat_t;
```

成员说明：

messageIdAccVal

接收消息 ID 接收过滤值，与 messageIdAccMask 组合使用实现带 PPI 指示的数据帧中第一个负载数据字的消息 ID 接收过滤。

messageIdAccMask

接收消息 ID 接收过滤掩码。

frameIdRejVal

帧 ID 拒绝接收值，与 FrameIdRejMask 组合使用实现 FlexRay 帧 ID 拒绝接收的过滤。

FrameIdRejMask

帧 ID 拒绝接收掩码。

rangeFilter[x]Enable

范围过滤是否使用，1：使能， 0：禁用。

rangeFilter[x]Upper

本组 FrameID 范围过滤的最大值；

rangeFilter[x]Lower

本组 FrameID 范围过滤的最小值；

rangeFilter[x]Mode

本组 FrameID 范围过滤模式，0：接收， 1： 拒绝。

1.3.5.2 UFr_Set_Fifo_Parameter_Wrapper

这个函数用于设置 FlexRay 节点的 fifo 参数，应该在 UFr_Open_Wrapper 打开设备成功后调用。

```
uint8  UFr_Set_Fifo_Parameter_Wrapper
( sint8 instance, uint8 controller_num, FlexrayFifoParameterFlat_t *fifo_param_flat)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】控制器号，用于表示此类设备的通道号，从 0 开始排序。

* *fifo_param_flat*

【IN】总线节点 fifo 参数定义结构体指针，用于配置节点 fifo 监控接收模式的参数，调用前需要预先定义该结构体变量并赋值，具体请参看结构体 FlexrayFifoParameterFlat_t 的定义。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
//初始化 fifo 配置参数

FlexrayFifoParameterFlat_t mParamSetFRFifo{
    0, 0, 0, 0x7FF, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
};
 //配置时隙参数
  ret = UFr_set_Fifo_parameter(instance, 0, &mParamSetFRFifo);
    if(ret != CMD_SUCCESS)
    {
        printf("UFr_set_Fifo_parameter FAIL!\r\n");
        return 0;
    }
```

```
printf("UFr_set_Fifo_parameter OK!\r\n");
```

1.3.6 配置 FlexRay 通道的 KeySlot【可选】

1.3.6.1 结构体定义

1.3.6.1.1 FRPhyNodeKeySlotFlat_t

这个结构体定义了配置节点的 KeySlot 参数的数据成员，具体定义如下：

```
typedef struct p_fr_keyslotcfg_t
{
    uint16_t keySlotId_1;
    uint8_t keySlotUsedForStartup_1;
    uint8_t keySlotUsedForSync_1;

    uint16_t keySlotId_2;
    uint8_t keySlotUsedForStartup_2;
    uint8_t keySlotUsedForSync_2;
}FRPhyNodeKeySlotFlat_t;
```

成员说明：

keySlotId_1

本节点配置的第一个 KeySlot 号，此值将覆盖掉函数 UFr_Set_Node_Parameter_Wrapper 执行时配置的 pKeySlotId 值。

keySlotUsedForStartup_1

第一个 KeySlotID 是否作为启动帧，1：是，0：否。此值将覆盖掉函数 UFr_Set_Node_Parameter_Wrapper 执行时配置的 pKeySlotUsedForStartup 值。

keySlotUsedForSync_1

第一个 KeySlotID 是否作为同步帧，1：是，0：否。此值将覆盖掉函数 UFr_Set_Node_Parameter_Wrapper 执行时配置的 pKeySlotUsedForSync 值。

keySlotId_2

本节点配置的第 2 个 KeySlot 号，**注意：这个 KeySlot 号不要与第一个 KeySlot 号重复，也不要与发送时隙有重复。**

keySlotUsedForStartup_2

第 2 个 KeySlotID 是否作为启动帧，1：是，0：否。

keySlotUsedForSync_2

第 2 个 KeySlotID 是否作为同步帧，1：是，0：否。

1.3.6.2 UFr_SetFrPhyNodeKeySlot_Wrapper

这个函数用于设置 FlexRay 节点的 KeySlot 参数，应该在启动节点之前调用。功能是在本节点上实现 2 个冷启动节点，即单个产品可以建立总线连接，使得 POC 状态进入 Normal Active 状态，这样就可以实现对非冷启动节点的一对一测试，如果不需要两个冷启动节点就不必调用本函数，因为 UFr_Set_Node_Parameter_Wrapper 函数执行时已经配置了第一个 KeySlot 的值。当前只针对 UL-FLEXRAY-LAN 和 UL-FLEXRAY-Recorder 两个型号产品有效，UBus-3240 产品本身具有 2 路 FlexRay，可以通过配置 2 路 FlexRay 的 KeySlot 实现上述功能。

```
uint8 UFr_SetFrPhyNodeKeySlot_Wrapper(sint8 instance, uint8 controller_num,
FRPhyNodeKeySlotFlat_t *pFRPhyNodeKeySlotFlat)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】控制器号，用于表示此类设备的通道号，从 0 开始排序。

* *pFRPhyNodeKeySlotFlat*

【IN】总线节点 keyslot 参数定义结构体指针，用于配置节点 keyslot 参数，调用前需要预先定义该结构体变量并赋值，具体请参看结构体 FRPhyNodeKeySlotFlat_t 的定义。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
//配置节点 keyslot，如果不执行 UFr_SetFrPhyNodeKeySlot_Wrapper 函数，设备将按照节点配置参数的 keyslot 值进行配置，默认不启动第二个 keyslot
    //如果需要设备本身能建立 FlexRay 总线连接，进入 normal active 状态时，可以配制 2 个 keyslot 都使能
    FRPhyNodeKeySlotFlat_t mFRPhyNodeKeySlotFlat;
    //第一个 keyslot 可以沿用节点参数的配置，也可以根据需要修改为其他值，注意不能与总线上其他节点存在相同的 keyslot 值，否则无法建立总线连接
    mFRPhyNodeKeySlotFlat.keySlotId_1 = mFlexrayNodeParameter.pKeySlotId;
    mFRPhyNodeKeySlotFlat.keySlotUsedForSync_1 = mFlexrayNodeParameter.pKeySlotUsedForSync;
    mFRPhyNodeKeySlotFlat.keySlotUsedForStartup_1 = mFlexrayNodeParameter.pKeySlotUsedForStartup;
    //第二个 keyslot 可以找一个总线上没有被使用的 keyslotID，本例采用最后一个静态时隙值，也可以根据需要修改为其他值
    //注意不能与总线上其他节点存在相同的 keyslot 值，否则无法建立总线连接
    //keySlotId_2 的值如果为 0，表示不启动第二个 keyslot 设置
    mFRPhyNodeKeySlotFlat.keySlotId_2 = mFlexrayClusterParameter.gNumberOfStaticSlots;
```

```
    mFRPhyNodeKeySlotFlat.keySlotUsedForSync_2 = 1;
    mFRPhyNodeKeySlotFlat.keySlotUsedForStartup_2 = 1;

    if(UFr_SetFrPhyNodeKeySlot_Wrapper(instance, 0, &mFRPhyNodeKeySlotFlat) !=
CMD_SUCCESS)
    {
        printf("UFr_SetFrPhyNodeKeySlot_Wrapper fail!\r\n");
        return 0;
    }
    printf("UFr_SetFrPhyNodeKeySlot_Wrapper OK!\r\n");
```

1.3.7 配置生效

1.3.7.1 UFr_Configure_Node_Wrapper

这个函数用于设置节点参数生效，应该在 UFr_Open 打开设备成功并进行了前面的 UFr_Set_Cluster_Parameter 、 UFr_Set_Node_Parameter 、 UFr_Set_CC_Parameter 、 和 UFr_Set_Slot_Parameter 配置后调用。

```
uint8 UFr_Configure_Node_Wrapper(sint8 instance, uint8 controller_num)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
//配置使能
    if(UFr_Configure_Node_Wrapper(instance, 0) != CMD_SUCCESS)
    {
        printf("UFr_Configure_Node fail!\r\n");
        return 0;
    }
```

1.3.8 唤醒操作

1.3.8.1 UFr_SendWakeUp_Wrapper

在配置节点操作完成后，节点进入 Ready 状态，此时可以调用这个函数向 FlexRay 总线发送 Wakeup pattern，用于唤醒总线上处于等待状态的节点，应该在 UFr_Configure_Node_Wrapper 配置后调用。调用这个函数后，节点进入 wakeup 状态，需要等待节点进入 Ready 状态后才能调用 UFr_Start_Node 函数启动节点，可通过调用 UFr_GetFrNodeState_Wrapper 函数获得节点的状态。其他处于 ready 状态的节点可以收到 wakeup pattern，并开始建立总线连接。

```
uint8 UFr_SendWakeUp (sint8 instance, uint8 controller_num)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
//发送唤醒指令
    if(UFr_SendWakeUp_Wrapper(instance, 0) != CMD_SUCCESS)
    {
        printf("UFr_SendWakeUp fail!\r\n");
        return 0;
    }
```

1.3.9 配置设备时间偏移【可选】

1.3.9.1 UBus_SetTimeOffset_Wrapper

设备默认从上电后自动从 0 开始计时，并在接收到数据帧时做时间戳标识，如果对时间戳的起始时间有要求，可以调用此函数设置时间偏移，例如可以将当前系统时间配置到设备中，这样设备上报数据帧的时间戳就可以跟系统时间同步。

```
uint8 UBus_SetTimeOffset_Wrapper(sint8 instance, uint64 Second)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

Second

【IN】要配置的时间偏移值，单位是秒。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

注意：

此函数属于设备级的配置函数，调用后所有端口的时间戳都将发生变化，包括 FlexRay 和 CAN 通道，所以建议：如果需要设定时间偏移，最好在程序进行总线通讯之前调用本函数配置时间，否则可能导致接收报文的时间戳信息不准确。

例子：

```
//配置设备的时间从 0 开始计时
    if(UBus_SetTimeOffset_Wrapper(instance, 0) != CMD_SUCCESS)
    {
        printf("UBus_SetTimeOffset_Wrapper fail!\r\n");
        return 0;
    }
```

1.3.10 FlexRay 工作模式及 MTS 使能【可选】

1.3.10.1 枚举定义

1.3.9.1.1 Fr_WORKMODE_type

这个枚举类型定义了几种 FlexRay 节点的工作模式，定义如下：

```
typedef enum
{
    FR_WORKDMODE_NORMAL = 0,
    FR_WORKDMODE_PLAYBACK
} Fr_WORKMODE_type;
```

成员说明：

FR_WORKDMODE_NORMAL

正常报文收发模式。

FR_WORKDMODE_PLAYBACK

回放数据模式。

1.3.10.2 UFr_Set_WorkMode_Wrapper

FlexRay 节点支持正常通讯和回放数据两种工作模式，设备上电后默认进入正常通讯模

式，如果需要进行回放数据操作，需要调用此函数设置节点的工作模式。

FlexRay 总线的符号窗口期可以发送 MTS 介质测试符，通过此函数可以实现该功能。

```
uint8 UFr_Set_WorkMode_Wrapper(sint8 instance, uint8 controller_num, uint8 workmode, uint8 MTSenable)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

workmode

【IN】工作模式配置项，值为枚举类型 Fr_WORKMODE_type 中的值。

MTSenable

【IN】是否使能 MTS 发送，1：使能，0：禁用。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

此函数为可选执行函数，用户可根据实际需要调用。

例子：

```
//设置工作模式
    if((ret = UFr_Set_WorkMode_Wrapper(instance, 0, FR_WORKDMODE_NORMAL, 0)) != CMD_SUCCESS)
    {
        printf("UFr_Set_WorkMode_Wrapper fail[%d]!\r\n",ret);
        return 0;
    }
    printf("UFr_Set_WorkMode_Wrapper OK! \r\n");
```

1.3.11 协议状态查看

1.3.11.1 UFr_GetFrPSRState_Wrapper

调用这个函数可以获得 FlexRay 控制器的当前协议状态。可读取到 4 个协议状态字，其中包含 wakeup 状态。

```
uint8 UFr_GetFrPSRState(sint8 instance, uint8 controller_num, uint16 *frPSRState)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

frPSRState

【OUT】读取到的 FlexRay 控制器协议状态，由 4 个 16 位无符号数据组成。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
//获得 wakeup 状态
uint16 PSRstatus[4];
UFr_GetFrPSRState_Wrapper(instance, 0, PSRstatus);
uint16 wakeupStatus = PSRstatus[0]&0x0007;     // Select only WAKEUPSTATUS field

switch(wakeupStatus)                                  // Test Protocol State
{
      case FrPSR0_WAKEUPSTATUS_UNDEFINED:
            printf("Wakeup Status undefined!\r\n");
      break;
      case FrPSR0_WAKEUPSTATUS_RECEIVED_HEADER:
            printf("Wakeup Status Received_Header!\r\n");
      break;
      case FrPSR0_WAKEUPSTATUS_RECEIVED_WUP:
            printf("Wakeup Status Receiver_WUP!\r\n");
      break;
      case FrPSR0_WAKEUPSTATUS_COLLISION_HEADER:
            printf("Wakeup Status Collision_Header!\r\n");
      break;
      case FrPSR0_WAKEUPSTATUS_COLLISION_WUP:
            printf("Wakeup Status Collision_WUP!\r\n");
      break;
      case FrPSR0_WAKEUPSTATUS_COLLISION_UNKNOWN:
            printf("Wakeup Status Collision_Unknow!\r\n");
      break;
      case FrPSR0_WAKEUPSTATUS_TRANSMITTED:
            printf("Wakeup Status Transmitted, send wakeup success!\r\n");
      break;
      default:
      break;
```

```
}
```

1.4 启动连接

1.4.1 启动节点

1.4.1.1 UFr_Start_Node_Wrapper

这个函数用于启动节点建立或加入 FlexRay 总线连接，应该在配置操作完 UFr_Open_Wrapper 成后调用。

```
uint8 UFr_Start_Node_Wrapper(sint8 instance, uint8 controller_num)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

该函数执行后返回成功只代表本节点完成初始化配置，进入总线连接状态，需要等到 POC 状态为 normal Active 后才能进行收发数据操作。

例子：

```
//启动节点，进入总线建立连接状态
    if(UFr_Start_Node_Wrapper(instance, 0) != CMD_SUCCESS)
    {
        printf("UFr_Start_Node fail!\r\n");
        return 0;
    }
```

1.4.2 启动接收数据上传

1.4.2.1 UFr_StartFrRxUpload_Wrapper

这个函数用于启动节点接收数据上传给主机的功能，只有在该函数执行成功后才能正确接收数据帧，否则设备端不上传收到的数据帧，该函数应该在启动节点执行成功后调用。

```
uint8 UFr_StartFrRxUpload_Wrapper (sint8 instance, uint8 controller_num)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

该函数的调用可以根据实际项目需求来定，如果实际项目中不需要处理接收的数据帧，可以不调用本函数，这样可以节省主机端 CPU 处理资源和内存资源。设备端默认不开启接收数据上传功能。

例子：

```
//已经建立总线连接，启动接收数据上传操作
    if(UFr_StartFrRxUpload_Wrapper(instance, 0) != CMD_SUCCESS)
    {
        printf("UFr_StartFrRxUpload fail!\r\n");
        return 0;
    }
```

1.4.3 获取节点 POC 状态

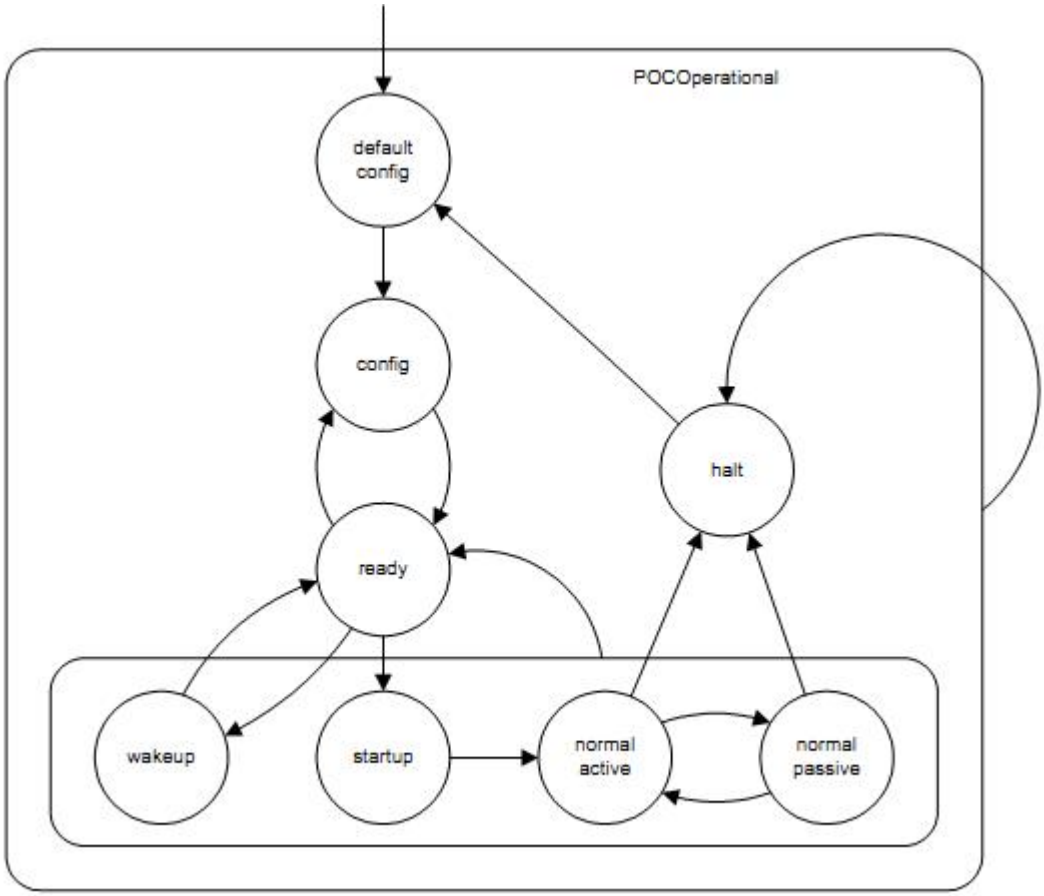
1.4.3.1 枚举定义

1.4.3.1.1 Fr_POC_state_type

这个枚举类型定义了节点的不同协议状态，定义如下：

```
typedef enum
{
    FR_POCSTATE_DEFAULT_CONFIG = 0,
    FR_POCSTATE_CONFIG,
    FR_POCSTATE_WAKEUP,
    FR_POCSTATE_READY,
    FR_POCSTATE_NORMAL_PASSIVE,
    FR_POCSTATE_NORMAL_ACTIVE,
    FR_POCSTATE_HALT,
    FR_POCSTATE_STARTUP,
} Fr_POC_state_type;
```

这个枚举类型定义了 FlexRay 协议规范中列出的全部 POC 状态，当节点进入 FR_POCSTATE_NORMAL_ACTIVE 状态后即可以进行正常的数据帧收发操作，各状态转换关系如下图：



1.4.3.2 UFr_GetFrNodeState_Wrapper

这个函数用于获取节点的 POC 状态和当前周期号，可在执行 UFr_Open 函数成功后的任何阶段调用。

```
uint8 UFr_GetFrNodeState_Wrapper(sint8 instance, uint8 controller_num, uint8 &frnodeState , uint8 &curCycle)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

frnodeState

【OUT】读取到的 FlexRay 控制器 POC 状态。

curCycle

【OUT】读取到的 FlexRay 总线当前周期号。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

该函数一般在启动 FlexRay 连接后调用来判断节点所处的 POC 状态，根据该状态判断总线连接是否正常。

例子：

```
uint8 frnodeState;
uint8 curCycle;

while(1)
    {
        //获得节点的状态
        UFr_GetFrNodeState_Wrapper(instance, 0, frnodeState, curCycle);
        switch(frnodeState)
        {
            case FR_POCSTATE_CONFIG:
                printf("FlexRay Node is in status [config]\r\n");
                break;
            case FR_POCSTATE_DEFAULT_CONFIG:
                printf("FlexRay Node is in status [defconfig]\r\n");
                break;
            case FR_POCSTATE_HALT:
                printf("FlexRay Node is in status [halt]\r\n");
                break;
            case FR_POCSTATE_NORMAL_ACTIVE:
                printf("FlexRay Node is in status [normal active]\r\n");
                break;
            case FR_POCSTATE_NORMAL_PASSIVE:
                printf("FlexRay Node is in status [normal passive]\r\n");
                break;
            case FR_POCSTATE_READY:
                printf("FlexRay Node is in status [ready]\r\n");
                break;
            case FR_POCSTATE_STARTUP:
                printf("FlexRay Node is in status [startup]\r\n");
                break;
            case FR_POCSTATE_WAKEUP:
                printf("FlexRay Node is in status [wakeup]\r\n");
                break;
            default:
```

```
                printf("FlexRay Node is in status [unknown]\r\n");
                break;
        }
        if(frnodeState == FR_POCSTATE_NORMAL_ACTIVE)
        {
          printf("FlexRay bus connect created! You can start recieve
upload!\r\n");
            break;
        }
        Sleep(1000);  //read once per second
    }
```

1.5 发送

1.5.1 UFr_Transmit_Wrapper

这个函数用于更新指定消息 Buffer 的发送数据，应该在节点进入 Normal Active 状态（即加入总线）后调用。

```
uint8 UFr_Transmit_Wrapper(sint8 instance,uint8 controller_num,uint8
messagebuffer_id,uint16 *buffer,uint8 wordlength)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

messagebuffer_id

【IN】指定发送数据的 messagebuffer ID 号，该参数应由 UFr_Set_Slot_Parameter 函数返回的 msgBufferID 作为输入，或者用户自己定义。

**buffer*

【IN】发送数据的缓冲区地址。

wordlength

【IN】发送数据的字长度，有效值为 0~127 个字。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

（1） 该函数功能是更新指定时隙的发送数据，数据需要等到时隙配置中指定的周期计数到达后才能实际发送到 FlexRay 总线上；

（2） 如果该时隙为静态时隙，那么字长度参数应为全局 cluster 参数中的

gPayloadLengthStatic 定义的长度值；

（3） 如果该时隙为动态时隙，那么字长度参数应小于或等于该时隙配置的负载长度；

例子：

```
uint8 PayloadLength = 10;
uint16 data[10]={0x1001,0x1002,0x1003,0x1004,0x1005,
                 0x1006,0x1007,0x1008,0x1009,0x100a};
//将接收到的 flexray 帧的数据部分通过时隙 1 发送出去
if(pCommLib->UFr_Transmit_Wrapper(instance, 0,时隙 1 使用的 messagebufferID,
data, PayloadLength) != CMD_SUCCESS)
            {
                printf("UFr_Transmit fail!\r\n");
            }
```

1.5.2 UFr_ReplaySend_Wrapper

这个函数打包回放 FlexRay 消息，1 次可以发送多包 FlexRay 消息，可回放静态时隙和动态时隙数据帧，应该在节点进入 Normal Active 状态（即加入总线）后调用。

	注意
	➢ 执行 UFr_ReplaySend_Wrapper 函数之前不需要调用 UFr_Set_Slot_Parameter_Wrapper 函数进行发送时隙消息缓冲区的配置，此函数可以根据填入的发送消息的时隙号和周期号自动在对应的时间区域进行发送。 ➢ 如需要保证回放 FlexRay 消息时的时间连续稳定，建议创建线程调用此函数不停发送，如果发送返回错误，表明设备缓冲区没有剩余空间了，需要等待 1 个周期左右时间，在设备发送出一批数据帧后再次尝试回放操作。

```
uint8 UFr_ReplaySend_Wrapper(sint8 instance, uint8 controller_num, uint8
numberOfSubPacket, uint8 *framebuffer, uint16 framebuffer_byteslength)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

numberOfSubPacket

【IN】指定本次发送缓冲区 framebuffer 包含多少帧 FlexRay 消息。

**framebuffer*

【IN】发送数据的缓冲区地址。

framebuffer_byteslength

【IN】发送缓冲区总字节长度。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
for(packetnum = 0; packetnum < numberOfsubPacket; packetnum++)
{
    //组织发送包的原始数据
    uFlexrayFrame_t mReplayFlexrayFrame;
    mReplayFlexrayFrame.ID = packetnum+1;
    mReplayFlexrayFrame.cycleOffset = 0;
    mReplayFlexrayFrame.PPI = 0;
    mReplayFlexrayFrame.CycleCount = lastcycle;
    mReplayFlexrayFrame.PayloadLength = 16;

    for(int i=0; i<mReplayFlexrayFrame.PayloadLength; i++)
    {
        mReplayFlexrayFrame.data[i] = (i << 8) + mReplayFlexrayFrame.ID ;
    }

    //将发送包数据填入回放帧缓冲区
    FlexrayReplayPacketHeader_t mFlexrayReplayPacketHeader;
    mFlexrayReplayPacketHeader.frameID = mReplayFlexrayFrame.ID;
    mFlexrayReplayPacketHeader.cycleCount = mReplayFlexrayFrame.CycleCount;
    mFlexrayReplayPacketHeader.cycleOffset = mReplayFlexrayFrame.cycleOffset;
    mFlexrayReplayPacketHeader.PPI = mReplayFlexrayFrame.PPI;
    mFlexrayReplayPacketHeader.payloadLength =
mReplayFlexrayFrame.PayloadLength;
    memcpy(ReplayFrameBuffer+bufferwrite_pos, &mFlexrayReplayPacketHeader,
sizeof(FlexrayReplayPacketHeader_t));
    bufferwrite_pos += sizeof(FlexrayReplayPacketHeader_t);
    memcpy(ReplayFrameBuffer+bufferwrite_pos, mReplayFlexrayFrame.data,
mFlexrayReplayPacketHeader.payloadLength*2);
    bufferwrite_pos += mFlexrayReplayPacketHeader.payloadLength*2;

}
        numberOfTotalPacket = numberOfsubPacket;
        //增加 1 包动态时隙帧，填入回放帧缓冲区
        FlexrayReplayPacketHeader_t mFlexrayReplayPacketHeaderdy;
        mFlexrayReplayPacketHeaderdy.frameID = 71;
```

```
        mFlexrayReplayPacketHeaderdy.cycleCount = lastcycle;
        mFlexrayReplayPacketHeaderdy.cycleOffset = 0;
        mFlexrayReplayPacketHeaderdy.PPI = 0;
        mFlexrayReplayPacketHeaderdy.payloadLength = 16;
        memcpy(ReplayFrameBuffer+bufferwrite_pos,
&mFlexrayReplayPacketHeaderdy, sizeof(FlexrayReplayPacketHeader_t));
        bufferwrite_pos += sizeof(FlexrayReplayPacketHeader_t);
        uint16 dydata[16];
        for(int i = 0; i < 16; i++)
        {
            dydata[i] = (i<<8) + 0x71;
        }
        memcpy(ReplayFrameBuffer+bufferwrite_pos, dydata,
mFlexrayReplayPacketHeaderdy.payloadLength*2);
        bufferwrite_pos += mFlexrayReplayPacketHeaderdy.payloadLength*2;

        numberOfTotalPacket = numberOfsubPacket+1;

        if(bufferwrite_pos > ReplayframebufferMaxbyteslength)
        {
            printf("[ERROR] Replay framebuffer bytes length[%d] can't big
than %d!",bufferwrite_pos,ReplayframebufferMaxbyteslength);
            return 0;
        }

//第一帧等待实际发送第一帧数据周期的前一个周期到来再开始回放数据
        if(isBeginning)
        {
            while(1)
            {
                 UFr_GetFrNodeState_Wrapper(instance,0,frnodeState,curCycle);
                if(curCycle  == 0)
                {
                    isBeginning = 0;
                    break;
                }

                //Sleep(1);
            }
        }
int trycount = 1000;最大尝试发送次数
while(trycount > 0)
{
    trycount--;
```

```
    //静态时隙+动态时隙，累计 numberOfTotalPacket 帧数据打包成一个以太网包发送

    if(UFr_ReplaySend_Wrapper(instance,0,numberOfTotalPacket,ReplayFrameBuffer,
    bufferwrite_pos) == CMD_SUCCESS)
    {

      printf("Replay[PacketCount=%d,bufflen=%d,cycle=%d]\r\n",numberOfTotalPack
      et,bufferwrite_pos,lastcycle);
                break;
      }else{
          printf(".");
     }
     Sleep(5);
}
```

1.6 接收

1.6.1 结构体定义

1.6.1.1 uFlexrayFrameHead_t

这个结构体定义了接收 FlexRay 消息帧的数据成员，具体定义如下：

```
typedef struct flexrayframethead
   {
       uint32 serial_number;
       uint32 timestamp_s;
       uint32 timestamp_us;
       uint8_t payloadPreambleIndicator;
       uint8_t nullFrameIndicator;
       uint8_t syncFrameIndicator;
       uint8_t startupFrameIndicator;
       uint16_t frameID;
       uint8_t  Channel;
       uint8_t cycleCount;
       uint8_t payloadLength;
       uint16_t headerCrc;
       uint16_t slotStatus;
       uint16 data[127];
```

```
} uFlexrayFrameHead_t;
```

成员说明：

serial_number

接收帧的序号，初始化设备后该序号从 0 开始计数。

timestamp_s

时间戳，从设备上电后开始的秒计时时间，与 *timestamp_us* 组合表示完整的时间戳。

timestamp_us

时间戳，从设备上电后开始的微秒计时时间，与 *timestamp_s* 组合表示完整的时间戳。

payloadPreambleIndicator

负载数据前导指示位:表明帧中有效负载的内容。1 表示有效负载为特殊内容；0 表示有效负载为一般内容。通信循环的静态段和动态段都可以输出帧，但该指示位在这两种情况下的含义不同:在静态段发送的帧(静态帧),1 表示负载场包含网络管理向量，0 表示负载场不包含该向量;在动态段发送的帧(动态帧)，1 表示负载场包含辅助性报文 ID，0 表示负载场不包含辅助性报文 ID。

nullFrameIndicator

空帧标志，如果为 1 表示负载数据有效，为 0 表示负载中没有有效数据。

syncFrameIndicator

同步帧标志，1 为同步帧。

startupFrameIndicator

启动帧标志，1 为启动帧。

frameID

接收 FlexRay 帧的时隙号，长度 11 位。

Channel

该帧数据接收的通道号，1 为 A 通道，2 为 B 通道，3 为 A、B 通道同时接收。

cycleCount

本消息接收时的周期计数值。

payloadLength

FlexRay 帧的负载数据长度。

headerCrc

帧头 CRC 值。

slotStatus

时隙状态值。16 位长度数据代表的意义如下表所示：

位	描述
15(VFB)	通道 B 有效帧标志，协议相关变量：vSS!ValidFrame
14(SYB)	通道 B 同步帧标志，协议相关变量：vRF!Header!SyFIndicator
13(NFB)	通道 B 空帧标志，协议相关变量：vRF!Header!NFIndicator
12(SUB)	通道 B 启动帧标志，协议相关变量：vRF!Header!SuFIndicator
11(SEB)	通道 B 语法错误标志，协议相关变量：vSS!SyntaxError
10(CEB)	通道 B 内容错误标志，协议相关变量：vSS!ContentError

9(BVB)	通道 B 越界标志，协议相关变量：vSS!BViolation
8(CH)	通道指示位： 0：第一个有效帧在 channel A 上接收，或者根本就没有有效帧 1：第一个有效帧在 channel B 上接收
7(VFA)	通道 A 有效帧标志，协议相关变量：vSS!ValidFrame
6(SYA)	通道 A 同步帧标志，协议相关变量：vRF!Header!SyFIndicator
5(NFA)	通道 A 空帧标志，协议相关变量：vRF!Header!NFIndicator
4(SUA)	通道 A 启动帧标志，协议相关变量：vRF!Header!SuFIndicator
3(SEA)	通道 A 语法错误标志，协议相关变量：vSS!SyntaxError
2(CEA)	通道 A 内容错误标志，协议相关变量：vSS!ContentError
1(BVA)	通道 A 越界标志，协议相关变量：vSS!BViolation
0	保留位，常为 0

1.6.2 UFr_Receive_Wrapper

这个函数用于接收 FlexRay 数据帧，应该在节点进入 Normal Active 状态（即加入总线）后调用。

```
uint8 UFr_Receive_Wrapper(sint8 instance, uFlexrayFrameHead_t *rcvFrFrameHead,
uint16 *rcvFrDatabuffer,    int timeout_ms)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

rcvFrFrameHead

【OUT】接收到的 flexray 帧的帧头部分存储结构体，具体定义参看结构体 *uFlexrayFrameHead_t* 的详细说明。

rcvFrDatabuffer

【OUT】接收到的 flexray 帧的数据部分，调用时需要提供存放接收数据的缓冲区首地址。

timeout_ms

【IN】指定接收等待超时时间，以毫秒为单位，如果缓冲区有数据则读取一帧数据后退出，如果缓冲区没有数据则进入等待状态，超时后退出。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
uFlexrayFrameHead_t  mFlexrayFrameHead;
uint16 rcvData[127];
    //接收数据并打印
    while(1)
    {
        if(UFr_Receive_Wrapper(instance,mFlexrayFrameHead,rcvData, 1000) ==
CMD_SUCCESS)
        {

printf("channel=%d, ID=%d, Cycle=%02d, PayloadLength=%d, slotstatus=%04X, startup=%d
, sync=%d, isNullFrame=%d, isPPI=%d, SE=%d, headCRC=%04x, TC=%d, data = ",

mFlexrayFrameHead.Channel, mFlexrayFrameHead.ID, mFlexrayFrameHead.CycleCount, mFl
exrayFrameHead.PayloadLength,
                    mFlexrayFrameHead.slotstatus, mFlexrayFrameHead.STARTUP,

mFlexrayFrameHead.SYNC, mFlexrayFrameHead.NF, mFlexrayFrameHead.PP, mFlexrayFrameH
ead.SyntaxError, mFlexrayFrameHead.headerCrc, mFlexrayFrameHead.TransmissionConfl
ict);
            for(int i=0; i<mFlexrayFrameHead.PayloadLength;i++)
            {
                printf("%04X ",*(uint16*)(rcvData+i));
            }
            printf("\r\n");
        }
    }
```

1.6.3 UFr_framesAvailable_Wrapper

这个函数用于获取动态库缓冲区包含的接收数据包个数，应该在节点进入 Normal Active 状态（即加入总线）后调用。通过查看缓冲区中剩余的数据帧数，可以判断上层应用软件接收数据的效率如何，如果剩余帧数比较高，并且一直再增加，说明上层应用软件处理接收数据速度较慢，容易造成丢失数据，应调整业务逻辑或处理方法以保证不丢失数据。**动态库中设定的最大缓冲区帧数为 100 万帧 FlexRay 消息。**

```
uint8 UFr_framesAvailable_Wrapper(sint8 instance)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

返回值：

函数返回当前接收缓冲区包含的接收数据包个数。

备注：

这个函数可以根据实际项目中业务逻辑需求决定是否调用。

例子：

```
Int count = UFr_framesAvailable_Wrapper(instance);
```

1.6.4 UFr_ClearRcvBuffer_Wrapper

这个函数用于清除 FlexRay 接收缓冲区的数据。

```
uint8 UFr_ClearRcvBuffer_Wrapper(sint8 instance)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

这个函数可以根据实际项目中业务逻辑需求决定是否调用。

例子：

```
Int count = UFr_ClearRcvBuffer_Wrapper(instance);
```

1.6.5 UFr_GetFrNWVector_Wrapper

这个函数用于获取动态库缓冲区包含的接收数据包个数，应该在节点进入 Normal Active 状态（即加入总线）后调用。通过查看缓冲区中剩余的数据帧数，可以判断上层应用软件接收数据的效率如何，如果剩余帧数比较高，并且一直再增加，说明上层应用软件处理接收数据速度较慢，容易造成丢失数据，应调整业务逻辑或处理方法以保证不丢失数据。**动态库中设定的最大缓冲区帧数为 100 万帧 FlexRay 消息。**

```
uint8 UFr_GetFrNWVector_Wrapper(sint8 instance, uint8 controller_num, uint8 *NWVectorLength, uint8 *NWVector);
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

NWVectorLength

【OUT】函数输出接收到的网络管理向量长度。

NWVector

【OUT】函数输出接收到的网络管理向量值，调用时应提供 12 个字节的缓冲区首地址给函数。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
uint8  NWVectorLength;
uint8  NWVector[12];
if(UFr_GetFrNWVector_Wrapper(instance, 0, &NWVectorLength, NWVector) != CMD_SUCCESS)
{
        printf("UFr_GetFrNWVector_Wrapper fail!\r\n");
        return 0;
}else
{
        printf("NWVectorLength = %d\r\n",NWVectorLength);
}
```

1.7 关闭

1.7.1 UFr_StopFrRxUpload_Wrapper

这个函数用于停止节点接收数据上传给主机的功能，如果主机端不再需要处理接收的数据帧，可以调用此函数关闭数据上传功能。

```
uint8 UFr_StopFrRxUpload_Wrapper(sint8 instance, uint8 controller_num)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

该函数的调用可以根据实际项目需求来定，如果实际项目中不需要处理接收的数据帧，可以调用此函数关闭上传功能（设备端默认关闭上传功能），这样可以节省主机端 CPU

处理资源和内存资源。设备端默认不开启接收数据上传功能。

例子：

```
//已经建立总线连接，启动接收数据上传操作
    if(UFr_ StopFrRxUpload_Wrapper (instance,0) != CMD_SUCCESS)
    {
        printf("UFr_ StopFrRxUpload fail!\r\n");
        return 0;
    }
```

1.7.2 UFr_Stop_Wrapper

这个函数用于停止节点的通讯功能，退出 FlexRay 总线连接，一般在退出 FlexRay 通讯程序之前调用。

```
uint8 UFr_Stop_Wrapper (sint8 instance, uint8 controller_num)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
if(UFr_Stop_Wrapper(instance,0) != CMD_SUCCESS)
{
    printf("UFr_Stop fail!\r\n");
}
```

1.7.3 UFr_Close_Wrapper

这个函数用于关闭打开的 FlexRay 设备。

```
uint8 UFr_Close_Wrapper(sint8 instance)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

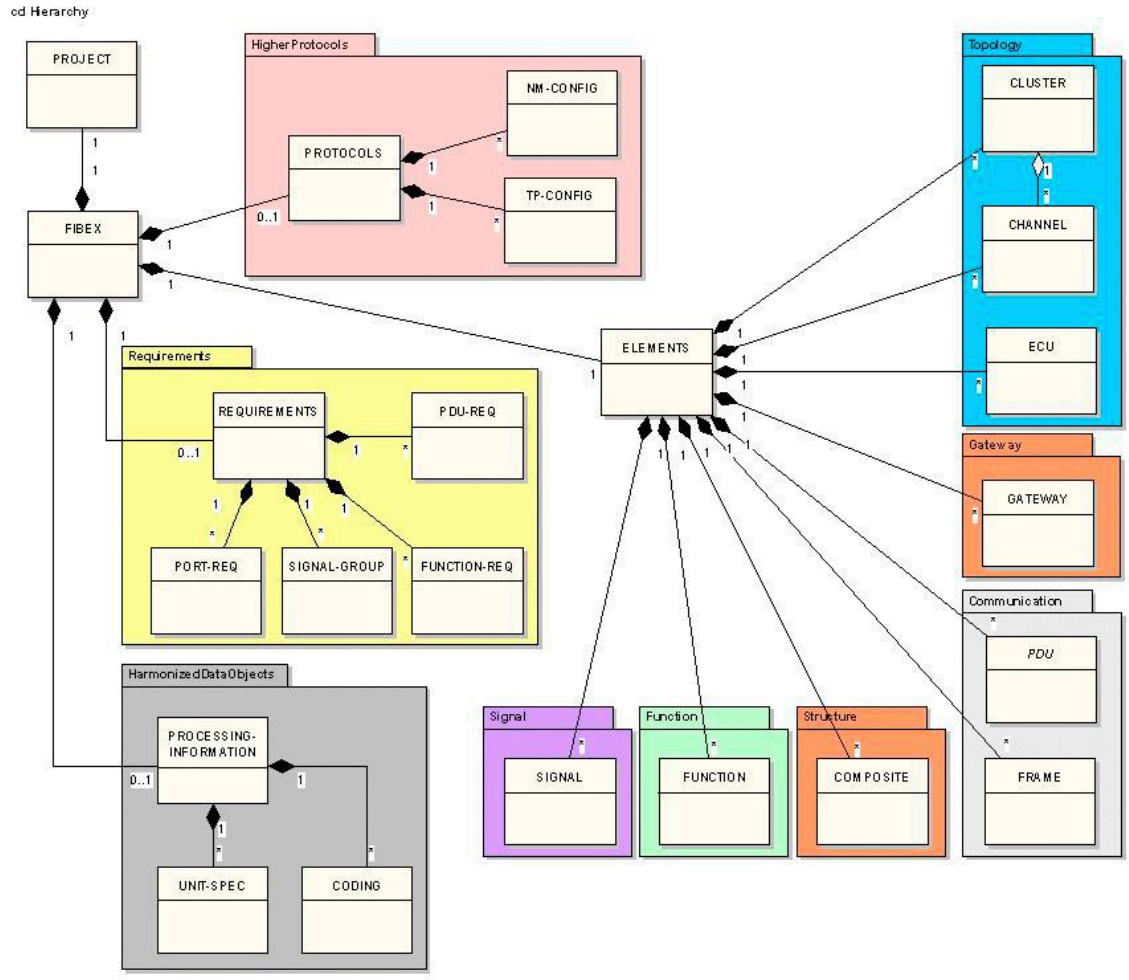
例子：

```
if(UFr_Close_Wrapper(instance) != CMD_SUCCESS)
{
    printf("UFr_Close fail!\r\n");
}
```

2. Fibex 数据库编程 API 说明

本章介绍 FlexRay 总线常用的 Fibex 数据库文件加载和解析的相关函数说明，提供基于 QT 开发环境的测试例程作为编程参考。

FIBEX 标准描述了一种支持多协议的格式，用于在面向消息的通信系统(例车载网络)中交换数据。FIBEX 数据库文件为 XML 格式定义，可用于导出车载网络数据库，以及在车辆网络开发期间导入不同类型的工具。目前，它支持 FlexRay、CAN、MOST 和 LIN 网络协议。从完整集群的设计到比特级别上的定义，都可以使用 FIBEX 格式来定义与车辆网络的描述相关的所有信息。



Fibex 数据库文件结构图

Fibex 文件可用于获得总线 Cluster 参数、节点参数、通讯控制器参数、帧信息和信号定义等内容，如果能获得被测产品厂家或总体单位提供的 XML 文件，可以通过调用优蓝科技提供的下列 API 接口，很方便的访问到需要的资源，不用去自行编程解析 XML 文件，减少编

程工作量，提高程序执行效率，快速上线产品。

2.1 打开数据库文件

2.1.1 UFr_Fibex_Open

这个函数用于打开 Fibex 数据库文件，执行此函数前需要成功执行 UFr_Open_Wrapper 函数，即成功访问硬件设备后才可以进行 Fibex 数据库文件相关操作。

后续所有与 Fibex 解析相关的操作都要基于成功调用本函数的基础上才能正确执行。

```
sint8 UFr_Fibex_Open(const char *FibexFileName)
```

参数说明：

FibexFileName

【IN】需要解析的 Fibex 文件绝对路径。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
sint8 result = UFr_Fibex_Open("E:\\Fibex\\Fibex-demo.xml");
```

2.2 获取总线参数

2.2.1 UFr_Fibex_GetParameter

这个函数用于从 Fibex 文件获取总线参数，需要用户输入 cluster 名称和 ECU 控制器名称字符串，函数如果返回正确，则可以使用函数输出的 outfcp，outfnp，outfccp 对应的结构体进行后面的初始化操作。

此函数应在 1.3 节的【配置】之前调用。

```
sint8 UFr_Fibex_GetParameter(const char *inClusterName, const char
*inControllerName, FlexrayClusterParameter_t *outfcp,
FlexrayNodeParameter_t *outfnp, FlexrayCCParameter_t *outfccp)
```

参数说明：

inClusterName

【IN】用户指定需要获取参数的 Cluster 名称。

inControllerName

【IN】用户指定需要获取参数的 ECU 控制器名称。

outfcp

【OUT】函数输出的总线 Cluster 参数结构体，调用时需要指定该结构体的指针，具体定义参看结构体 *FlexrayClusterParameter_t* 的详细说明。

outfnp

【OUT】函数输出的总线节点参数结构体，调用时需要指定该结构体的指针，具体定义参看结构体 *FlexrayNodeParameter_t* 的详细说明。

outfccp

【OUT】函数输出的总线控制器参数结构体，调用时需要指定该结构体的指针，具体定义参看结构体 *FlexrayCCParameter_t* 的详细说明。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
FlexrayClusterParameter_t mFlexrayClusterParameter;
FlexrayNodeParameter_t mFlexrayNodeParameter;
FlexrayCCParameter_t mFlexrayCCParameter;
//从 Fibex 文件获得总线参数
sint8 result = UFr_Fibex_GetParameter("BackboneFR","VGM",
&mFlexrayClusterParameter, &mFlexrayNodeParameter, &mFlexrayCCParameter);
if(result != CMD_SUCCESS)
{
      printf("UFr_Fibex_GetParameter fail(%d)!\r\n",result);
}else{
      printf("UFr_Fibex_GetParameter OK!\r\n");
}

//配置 cluster 参数

if(UFr_Set_Cluster_Parameter_Wrapper(instance,0,&mFlexrayClusterParameter) !=
CMD_SUCCESS)
{
     printf("Set_Cluster_Parameter fail!\r\n");
     return 0;
}
printf("Set_Cluster_Parameter OK!\r\n");
……
```

2.3 通过信号名获得帧名

2.3.1 UFr_Fibex_GetFrameName_BySignalName

函数通过信号名称查询该信号所在的帧名称，函数如果返回正确，则可以使用

outFrameName 进行后面时隙参数获得和配置时隙。

此函数可根据实际需要调用。

```
sint8 UFr_Fibex_GetFrameName_BySignalName(const char *signalname, char *outFrameName, bool *isFirstGet)
```

参数说明：

signalname

【IN】用户指定信号名称。

outFrameName

【OUT】函数执行后输出对应信号所在帧的名称。

isFirstGet

【OUT】表明该帧名是否为第一次获得，用户可根据该参数返回值来判断是否需要配置该帧对应的时隙参数，如果为 true 则可以进行配置，否则不需要再配置，因为之前该函数查询的某个信号已经在该帧中，不要多次配置相同帧的时隙参数。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
FlexrayClusterParameter_t mFlexrayClusterParameter;
 char FrameNameStr[100];
 bool isFirstGet;
 sint8 ret = UFr_Fibex_GetFrameName_BySignalName("RoadIncSignal", FrameNameStr, &isFirstGet);
    if(ret == CMD_SUCCESS)
    {
        printf("Frame Name is: %s\r\n",FrameNameStr);
    }
... ...
```

2.4 通过帧名称获得该帧的时隙配置

2.4.1 UFr_Fibex_GetSlotParameter

函数通过帧名称获得该帧的时隙配置信息，函数如果返回正确，则可以使用函数输出的 outmsgBufferParameter 结构体传给后面的 UFr_Set_Slot_Parameter_Wrapper 函数进行时隙参数配置。

```
sint8 UFr_Fibex_GetSlotParameter(const char *inFrameName, bool isTx, bool PPIflag, FlexrayMsgBufferParameter_t *outmsgBufferParameter)
```

参数说明：

inFrameName

【IN】用户指定帧名称。

isTx

【IN】用户指定该帧是否为当前节点的发送帧。

PPIflag

【IN】用户指定该帧负载前导指示标志。

outmsgBufferParameter

【OUT】函数输出参数，如果函数执行成功，该结构体存放该帧的时隙配置参数，可作为 UFr_Set_Slot_Parameter_Wrapper 函数的输入参数。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
FlexrayMsgBufferParameter_t gFlexrayMsgBufferParameter;
   ret = UFr_Fibex_GetSlotParameter("BcmBackBoneFr00", 1, 0,
&gFlexrayMsgBufferParameter);
   if(ret != CMD_SUCCESS)
   {
       printf("UFr_Fibex_GetSlotParameter FAIL(TX)[%d]!\r\n",ret);
       return 0;
   }
   //配置发送时隙参数
   ret =
UFr_Set_Slot_Parameter_Wrapper(instance,0,&gFlexrayMsgBufferParameter);
   if(ret != CMD_SUCCESS)
   {
       printf("UFr_Set_Slot_Parameter FAIL(TX)!\r\n");
        return 0;
   }
   printf("UFr_Set_Slot_Parameter ok[TX][used msgBufferID
= %d]!\r\n",gFlexrayMsgBufferParameter.msgBufferID);
... ...
```

2.5 通过帧名进行时隙配置

2.5.1 UFr_Fibex_Set_Slot_Parameter_Wrapper

函数通过帧名称进行对应时隙的参数设置，合并了 UFr_Fibex_GetSlotParameter 和 UFr_Set_Slot_Parameter_Wrapper 两个函数的操作，函数如果返回正确，当该帧为发送帧时，可以使用 UFr_Fibex_Transmit_Wrapper 函数进行该帧的数据发送。

```
uint8 UFr_Fibex_Set_Slot_Parameter_Wrapper(sint8 instance, uint8 controller_num,
const char *inFrameName, bool isTx, bool PPIflag)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

inFrameName

【IN】用户指定帧名称。

isTx

【IN】用户指定该帧是否为当前节点的发送帧。

PPIflag

【IN】用户指定该帧负载前导指示标志。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
ret = UFr_Fibex_Set_Slot_Parameter_Wrapper(instance, 0, "BcmVddmBackBoneFr00", true, false);
    if(ret != CMD_SUCCESS)
    {
        printf("UFr_Fibex_Set_Slot_Parameter_Wrapper FAIL(%d)!\r\n",ret);
        return 0;
    }else{
        printf("UFr_Fibex_Set_Slot_Parameter_Wrapper BcmVddmBackBoneFr00 ok!\r\n");
    }
... ...
```

2.6 通过帧名发送数据

2.6.1 UFr_Fibex_Transmit_Wrapper

函数通过帧名称进行该帧的数据发送操作。

```
uint8 UFr_Fibex_Transmit_Wrapper(sint8 instance, uint8 controller_num, const char *inFrameName, uint8 *buffer, uint8 bytelength)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UFr_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

controller_num

【IN】要配置的控制器号，FlexRay 通讯控制器的数值为 0。

inFrameName

【IN】用户指定帧名称。

**buffer*

【IN】发送数据的缓冲区地址。

bytelength

【IN】发送数据的字节长度，有效值为 0~254 个字节。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
if(UFr_Fibex_Transmit_Wrapper(instance, 0, "AsdmBackBoneFr10", (uint8*)data, payloadLength)!= CMD_SUCCESS)
    {
        printf("UFr_Transmit fail!\r\n");
    }else{
        printf("UFr_Transmit ok!\r\n");
    }
... ...
```

2.7 根据接收数据帧头获得帧名

2.7.1 UFr_Fibex_GetFrameName

通过接收到的 FlexRay 帧头信息，获得该帧的帧名称。此函数应在 UFr_Recieve 函数接收到有效数据后，将帧头信息传给本函数来获取该帧在 Fibex 文件中定义的帧名。

```
sint8 UFr_Fibex_GetFrameName(uFlexrayFrameHead_t *rcvFrFrameHead, char *outFrameName)
```

参数说明：

rcvFrFrameHead

【IN】接收数据帧的帧头结构体，详细定义参看结构体 *uFlexrayFrameHead_t* 的定义。

outFrameName

【OUT】函数输出的帧名。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
FlexrayFrameHead_t uFlexrayFrameHead;
char framename[100];
if(ret = UFr_Receive_Wrapper(instance, &uFlexrayFrameHead, rcvData, 1000))==
```

```
CMD_SUCCESS)
{
    ret = UFr_Fibex_GetFrameName(&uFlexrayFrameHead,  framename);
    if(ret == CMD_SUCCESS)
    {
        printf("FrameName=%s \r\n",framename);
    }
}
... ...
```

2.8 解码信号

2.8.1 UFr_Fibex_decodeSignal

通过接收到的 FlexRay 帧头信息、帧数据和指定的信号名称来解码该信号的值。此函数应在 UFr_Recieve 函数接收到有效数据后，将帧头和数据传给本函数来获取指定信号名的物理值。

```
sint8 UFr_Fibex_decodeSignal(uFlexrayFrameHead_t *rcvFrFrameHead, uint8
*rcvFrDatabuffer, const char *signalname, float *signalval)
```

参数说明：

rcvFrFrameHead

【IN】接收数据帧的帧头结构体，详细定义参看结构体 *uFlexrayFrameHead_t* 的定义。

rcvFrDatabuffer

【IN】接收数据帧的数据首地址。

signalname

【IN】指定要解码的信号名。

signalval

【OUT】函数执行成功后返回指定信号的物理值。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
char framename[100];
    ret = UFr_Fibex_GetFrameName(&uFlexrayFrameHead,  framename);
    if(ret == CMD_SUCCESS)
    {
        printf("FrameName=%s \r\n",framename);
    }
   //解析信号
```

```
    if(strcmp(framename, "BbmBcmBackBoneFr01") == 0)
      {
          ret = UFr_Fibex_decodeSignal(&uFlexrayFrameHead, (uint8*)rcvData,
"BrkPedlTrvlTar", &sigval1);
          if(ret == CMD_SUCCESS)
        {
              printf("signal: BrkPedlTrvlTar= %f\r\n",sigval1);
        }else{
              printf("signal: BrkPedlTrvlTar can't be decode[%d]!\r\n",ret);
        }
        ret = UFr_Fibex_decodeSignal(&uFlexrayFrameHead, (uint8*)rcvData,
"BrkPedlTrvlChks", &sigval2);
        if(ret == CMD_SUCCESS)
        {
            printf("signal: BrkPedlTrvlChks= %d\r\n",(uint32)sigval2);
        }else{
            printf("signal: BrkPedlTrvlChks can't be decode[%d]!\r\n",ret);
        }
    }

... ...
```

2.9 编码信号

2.9.1 UFr_Fibex_encodeSignal

编码输出帧中的指定信号物理值。函数执行成功后，需要调用 UFr_Fibex_Transmit_Wrapper 或 UFr_Transmit_Wrapper 函数将 DataBuffer 指定的数据发出。

```
sint8 UFr_Fibex_encodeSignal(const char *signalname, const float signalval, uint8
*DataBuffer, uint8 DataBufferBytelength)
```

参数说明：

signalname

【IN】指定要编码的信号名。

signalval

【IN】指定该信号的物理值。

DataBuffer

【IN&OUT】发送数据帧的数据首地址，编码后再将此数据发送出去。

DataBufferBytelength

【IN】发送数据帧的数据长度。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
    Float signalval = 1.1;
   ret = UFr_Fibex_encodeSignal("RoadInclnRoad", signalval, (uint8*)rcvData, 32);
   if(ret == CMD_SUCCESS)
   {
         printf("signal: RoadInclnRoadIncln encode ok!\r\n");
   }else{
         printf("signal: RoadInclnRoadIncln encode fail[%d]!\r\n",ret);
   }
  if(UFr_Fibex_Transmit_Wrapper(instance, 0, "BcmVddmBackBoneFr00",
         (uint8*)rcvData, payloadLength)!= CMD_SUCCESS)
  {
       printf("UFr_Transmit fail!\r\n");
  }else{
       printf("UFr_Transmit ok!\r\n");
  }
... ...
```

2.10 关闭数据库文件

2.10.1 UFr_Fibex_Close

这个函数用于关闭 Fibex 数据库文件，当程序不再使用 Fibex 文件的信息时可以执行此函数关闭 Fibex 文件释放占用的系统内存资源。

。

```
sint8 UFr_Fibex_Close(void)
```

参数说明：

无参数

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

例子：

```
UFr_Fibex_Close();
```

3. CAN/CANFD 编程 API 说明

本章介绍 CAN/CANFD 接口编程相关的数据结构、枚举类型和函数说明，提供基于 QT 开发环境的测试例程作为编程参考。

3.1 初始化

3.1.1 打开设备

3.1.1.1 UCAN_Open_Wrapper

这个函数用于打开 CAN/CANFD 设备通道，应该在进行所有其他操作之前调用。

```
sint8 UCAN_Open_Wrapper(char *FrIPaddr, uint8 nodeindex)
```

参数说明：

FrIPaddr

【IN】要打开设备的以太网地址。

nodeindex

【IN】要打开 CAN 设备的通道索引号，从 0 开始。

返回值：

如果函数执行成功返回值是大于 0 的设备实例号，后面的其他函数操作都要将此实例号作为输入参数来标明打开设备的唯一实例。否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值的负值及说明。

备注：

例子：

```
char targetIP[] = "192.168.0.7";
//打开设备
instance = UCAN_Open_Wrapper(targetIP, 0);
if(instance <= 0)
{
    printf("Open CAN channel fail!\r\n");
    return 0;
}
printf("Open CAN channel OK!\r\n");
```

3.2 配置

3.2.1 结构体定义

3.2.1.1 tCANParamstruct

这个结构体定义了 CAN 通道配置参数的数据成员，具体定义如下：

```
typedef struct CANFDParamstruct{
    uint32 stdFrameRxId;
    uint32 stdFrameRxIdMask;
    uint32 extFrameRxId;
    uint32 extFrameRxIdMask;
    uint8_t reserve;
    /*
     * The bitrate settings for standard frames or for the arbitration
phase of FD frames.
     */
    uint32 bitrate; /* The bitrate (bps) */
    /* Bit sampling point */
    /* User defined value (0 < bitSamplePoint < 1000), other value
enable the automatically calculation */
    uint32 bitSamplePoint;
    /* Bit time parameter for CAN or CAN FD arbitration phase */
    /* To use following parameters if 'bitrate' field in this structure
is set to 0 */
    uint16_t PRESDIV;    /* Prescaler Division factor setting */
    uint8_t PROPSEG;     /* Propagation Segment setting */
    uint8_t PSEG1;       /* Phase Segment 1 setting */
    uint8_t PSEG2;       /* Phase Segment 2 setting */
    uint8_t RJW;         /* ReSynchronization Jump Width setting */


    /*
     * The bitrate setting for the data phase of FD frames.
     */
    uint32 bitrateFD; /* The bitrate (bps) */
    /* Bit sampling point */
    /* User defined value (0 < bitSamplePoint < 1000), other value
enable the automatically calculation */
    uint32 bitSamplePointFD;
    /* Bit time parameter for CAN FD data phase */
    /* To use following parameters if 'bitrateFD' field in this
```

```
structure is set to 0 */
    uint16_t PRESDIV_FD; /* Prescaler Division factor setting */
    uint8_t PROPSEG_FD;  /* Propagation Segment setting */
    uint8_t PSEG1_FD;    /* Phase Segment 1 setting */
    uint8_t PSEG2_FD;    /* Phase Segment 2 setting */
    uint8_t RJW_FD;      /* ReSynchronization Jump Width setting */

    uint8_t autoRFR;    /* Automatic remote frame response */

    uint8_t disableTxFiFo;
}tCANFDParamstruct;
```

成员说明：

stdFrameRxId

定义标准帧的接收 ID,长度 11 位。

stdFrameRxIdMask

定义标准帧的接收掩码，0~0x7FF。

extFrameRxId

定义扩展帧的接收 ID，长度位 29 位。

extFrameRxIdMask

定义扩展帧的接收掩码，0~0x1FFFFFFF。

bitrate

定义为 CAN 速率，或 CANFD 的仲裁段速率，单位 bps。

注意：当 bitrate 为 0 时，由用户指以下参数定义 DAN 或 CAN FD 仲裁段速率。

BitSamplePoint

定义 CAN 总线的比特采样点，数值范围是 0~1000，对应千分比例值。其他值将使能自动计算功能。

PRESDIV

定义 CAN 总线预分频因数。

PROPSEG

定义 CAN 总线扩展段设置。

PSEG1

定义 CAN 总线阶段 1 设置。

PSEG2

定义 CAN 总线阶段 2 设置。

RJW

定义 CAN 总线重新同步跳转宽度设置。

bitrateFD

定义为 CANFD 数据段部分的速率，单位 bps。

注意：当 bitrateFD 为 0 时，由用户定义以下参数定义 CAN FD 数据段速率。

BitSamplePointFD

定义 CANFD 总线的比特采样点，数值范围是 0~1000，对应千分比例值。其他值将使能自动计算功能。

PRESDIV_FD

定义 CANFD 总线预分频因数。

PROPSEG_FD

定义 CANFD 总线扩展段设置。

PSEG1_FD

定义 CANFD 总线阶段 1 设置。

PSEG2_FD

定义 CANFD 总线阶段 2 设置。

RJW_FD

定义 CANFD 总线重新同步跳转宽度设置。

AutoRFR

定义自动远程帧响应功能是否使能，0：禁用，1：使能。

disableTxFiFo

定义发送缓冲区功能是否禁用，1：禁用，0：使能。

3.2.2 UCAN_Configure_Wrapper

这个函数用于设置 CAN 节点的参数，应该在 UCAN_Open_Wrapper 打开设备成功后调用。

```
uint8 UCAN_Configure_Wrapper(sint8 instance, tCANFDParamstruct
*ptCANParamstruct)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UCAN_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

ptCANParamstruct

【IN】总线参数定义结构体指针，用于配置总线参数，调用前需要预先定义该结构体变量并赋值，具体请参看结构体 tCANFDParamstruct 的定义。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
//定义 CAN 总线的参数结构体
tCANFDParamstruct mtCANParamstruct{
    0x1,//uint32 stdFrameRxId; /* Receive ID for standard frame */
    0x7FF,//uint32 stdFrameRxIdMask; /* Receive ID mask for standard frame */
    0x1,//uint32 extFrameRxId; /* Receive ID for extended frame */
    0x1fffffff,//uint32 extFrameRxIdMask; /* Receive ID mask for extended frame
*/
```

```
    0,//uint8_t reserve;     /*used for config MB max payload length, user can't
cfg this value,default to 64 bytes*/

    500000,//uint32 bitrate;   /* The bitrate used for standard frames or for
the arbitration phase of FD frames. */
    800,//uint32 bitSamplePoint;/* User defined value (0 < bitSamplePoint <
1000), other value enable the automatically calculation */
    /* Bit time parameter for CAN or CAN FD arbitration phase */
    0,//uint16_t PRESDIV;     /* Prescaler Division factor setting */
    0,//uint8_t PROPSEG;      /* Propagation Segment setting */
    0,//uint8_t PSEG1;        /* Phase Segment 1 setting */
    0,//uint8_t PSEG2;        /* Phase Segment 2 setting */
    0,//uint8_t RJW;          /* ReSynchronization Jump Width setting */

    /* Bit time parameter for CAN FD data phase */
    2000000,//uint32 bitrateFD; /* The bitrate used for the data phase of FD
frames. */
    /* User defined value (0 < bitSamplePoint < 1000), other value enable the
automatically calculation */
    800,//uint32 bitSamplePointFD;
    0,//uint16_t PRESDIV_FD; /* Prescaler Division factor setting */
    0,//uint8_t PROPSEG_FD;  /* Propagation Segment setting */
    0,//uint8_t PSEG1_FD;    /* Phase Segment 1 setting */
    0,//uint8_t PSEG2_FD;    /* Phase Segment 2 setting */
    0,//uint8_t RJW_FD;      /* ReSynchronization Jump Width setting */

    0,//uint8_t autoRFR;     /* Automatic remote frame response, the app will not
recieve RTR frame */
                             /* 要求双方都是 CAN frame，帧类型（扩展帧或标准帧）
相同，设置后所有外部远程帧都在设备
                                层自动回复了，app 层不再接收 RTR 帧*/
    1,//uint8_t disableTxFiFo; /*=1: disable tx FiFo, lost the frame when send
fail*/
                           /*=0: enable  tx FiFo, retry send the frame when send
fail*/
};

//配置 CAN 参数
    if(UCAN_Configure_Wrapper(instance, &mtCANParamstruct) != CMD_SUCCESS)
    {
        printf("UCAN_Configure_Wrapper fail!\r\n");
        return 0;
    }
    printf("UCAN_Configure_Wrapper OK!\r\n");
```

3.3 启动 CAN

3.3.1 UCAN_Start_Wrapper

这个函数用于启动 CAN 通道，应该在配置操作完成后调用，启动通道后即可进行发送和接收操作。

```
uint8 UCAN_Start_Wrapper(sint8 instance)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UCAN_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
//启动 CAN 通道
    if(UCAN_Start_Wrapper(instance) != CMD_SUCCESS)
    {
        printf("UCAN_Start_Wrapper fail!\r\n");
        return 0;
    }
    printf("UCAN_Start_Wrapper OK! \r\n");
```

3.4 发送

3.4.1 结构体定义

3.4.1.1 uCANTxFrameHead_t

本结构体定义了 CAN 发送报文的帧头部分结构体定义。

```
typedef struct CANTxFrameHead
{
    uint32 canID;
    uint8  protocolType; //0:CAN 2:CANFD  3:CANFD FAST
    uint8  RFResponseFlag;//Remote frame response flag
    uint8  isExtendedFrame;
```

```
    uint8  isRTRFrame;
    uint8  payloadLength;
} uCANTxFrameHead_t;
```

成员说明：

canID

定义发送帧的 ID 值。

protocolType

定义帧协议类型，0:CAN 2:CANFD 3:CANFD FAST。

RFResponseFlag

定义该帧为远程帧自动响应帧，如果在配置 CAN 通道时使能了远程帧自动响应功能，此处的标志如果为 1，那么当外部有本帧定义的 ID 远程帧到达时，设备将自动发送本帧数据，否则不发送本帧数据。

这个标志如果为 0 时，表明本帧为普通数据帧，正常发送。

isExtendedFrame

定义该帧是否为扩展帧，0：否，1：是。

isRTRFrame

定义该帧是否为远程帧，0：否，1：是。

payloadLength

定义该帧的数据长度。*有效数据长度为:0,1,2,3,4,5,6,7,8,12,16,20,24,32,48,64。*

3.4.2 UCAN_Transmit_Wrapper

这个函数用于发送 CAN 报文。

```
uint8 UCAN_Transmit_Wrapper(sint8 instance, uCANTxFrameHead_t
uCANTxFrameHead, uint8 *sendbuffer)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UCAN_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

uCANTxFrameHead

【IN】发送报文的帧头结构体，参考定义 *uCANTxFrameHead_t*。

Sendbuffer

【IN】发送报文的帧数据缓冲区首地址。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
//发送 1 帧数据
```

```
    uint8 sendbuffer[64];

    uCANTxFrameHead_t muCANTxFrameHead;
    muCANTxFrameHead.canID = 1;
    muCANTxFrameHead.RFResponseFlag = 0;
    muCANTxFrameHead.protocolType = CAN_TYPE;
    muCANTxFrameHead.isRTRFrame = 0;
    muCANTxFrameHead.isExtendedFrame = 0;
    muCANTxFrameHead.payloadLength = 8;//有效数据长度
值:{0,1,2,3,4,5,6,7,8,12,16,20,24,32,48,64};
    for(int i = 0; i < muCANTxFrameHead.payloadLength; i++)
    {
        sendbuffer[i] = i;
    }

        if((ret = UCAN_Transmit_Wrapper(instance, muCANTxFrameHead,
sendbuffer)) != CMD_SUCCESS)
        {
            printf("UCAN_Transmit_Wrapper fail[%d]!\r\n",ret);
        }else{
            printf("UCAN_Transmit_Wrapper ok!\r\n");
        }
```

3.5 接收

3.5.1 结构体定义

3.5.1.1 uCANFrameHead_t

本结构体定义了 CAN 接收报文的帧头部分结构体定义。

```
typedef struct CANFrameHead
{
    uint8  channelNo;
    uint32 serial_number;
    uint32 timestamp_s;
    uint32 timestamp_us;
    uint32 canID;
    uint8  protocolType; //0:CAN 2:CANFD  3:CANFD FAST
```

```
    uint8  isExtendedFrame;
    uint8  isRTRFrame;
    uint8  payloadLength;
} uCANFrameHead_t;
```

成员说明：

channelNo

定义接收该帧的通道号，从 0 开始。

serial_number

定义该帧的接收序号。

timestamp_s

定义该帧的时间戳，数值为秒部分。

timestamp_us

定义该帧的时间戳，数值为微秒部分。

canID

定义发送帧的 ID 值。

protocolType

定义帧协议类型，0:CAN 2:CANFD 3:CANFD FAST。

isExtendedFrame

定义该帧是否为扩展帧，0：否，1：是。

isRTRFrame

定义该帧是否为远程帧，0：否，1：是。

payloadLength

定义该帧的数据长度。*有效数据长度为:0,1,2,3,4,5,6,7,8,12,16,20,24,32,48,64。*

3.5.2 UCAN_Receive_Wrapper

这个函数用于接收 CAN 报文，采用阻塞接收方式，收到报文后立即返回，或者到达超时时间后返回。

```
uint8 UCAN_Receive_Wrapper(sint8 instance, uCANFrameHead_t
&outCANFrameHead, uint8 *databuffer, int timeout_ms)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UCAN_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

outCANFrameHead

【OUT】接收报文的帧头结构体，参考定义 uCANFrameHead_t。

databuffer

【OUT】接收报文的帧数据缓冲区首地址，调用前应预留足够的数据存储空间。

timeout_ms

【IN】接收报文的超时时间，以毫秒为单位。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
//接收数据
uCANFrameHead_t outCANFrameHead;
uint8 databuffer[64];

        if((ret = UCAN_Receive_Wrapper(instance, outCANFrameHead, databuffer,
2000))== CMD_SUCCESS)
        {
                 printf("ch=%d ",(int)outCANFrameHead.channelNo);
                printf("sn=%d ",(int)outCANFrameHead.serial_number);
                printf("time=%d.%06d ",(int)outCANFrameHead.timestamp_s ,
(int)outCANFrameHead.timestamp_us);
                printf("ID=%08x ",(uint32)outCANFrameHead.canID);
                printf("Length=%d ",outCANFrameHead.payloadLength);
                printf("RTR = %d ",outCANFrameHead.isRTRFrame);
                if(outCANFrameHead.protocolType == CAN_TYPE)
                {
                    printf("type=CAN ");
                }else if(outCANFrameHead.protocolType == CANFD_TYPE)
                {
                    printf("type=CANFD ");
                }else if(outCANFrameHead.protocolType == CANFDFAST_TYPE)
                {
                    printf("type=CANFD-FAST ");
                }

                printf("isExtendedFrame=%d
",outCANFrameHead.isExtendedFrame);

                printf("data: ");

                for(int i=0; i<outCANFrameHead.payloadLength;i++)
                {
                    printf("%02X ", databuffer[i]);
                }
                printf("\r\n");
```

```
    }else{
        //printf("UCAN_Receive_Wrapper return timeout[%d]\r\n",ret);
    }
```

3.5.3 UCAN_framesAvailable_Wrapper

这个函数用于获取动态库缓冲区包含的接收数据包个数，通过查看缓冲区中剩余的数据帧数，可以判断上层应用软件接收数据的效率如何，如果剩余帧数比较高，并且一直再增加，说明上层应用软件处理接收数据速度较慢，容易造成丢失数据，应调整业务逻辑或处理方法以保证不丢失数据。**动态库中设定的最大缓冲区帧数为 100 万帧消息。**

```
uint8 UCAN_framesAvailable_Wrapper(sint8 instance, sint32
*availableMsgCount)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UCAN_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

availableMsgCount

【OUT】函数执行成功后返回缓冲区中未被应用软件接收的报文数量。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

这个函数可以根据实际项目中业务逻辑需求决定是否调用。

例子：

```
sint32 availableMsgCount;
 UCAN_framesAvailable_Wrapper(instance, &availableMsgCount);
```

3.5.4 UCAN_ClearRcvBuffer_Wrapper

这个函数用于清除该通道接收缓冲区中的数据帧。

```
uint8 UCAN_ClearRcvBuffer_Wrapper(sint8 instance)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UCAN_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

这个函数可以根据实际项目中业务逻辑需求决定是否调用。

例子：

```
UCAN_ClearRcvBuffer_Wrapper(instance);
```

3.6 停止

3.6.1 UCAN_Stop_Wrapper

这个函数用于停止 CAN 通道，应该在程序退出前调用，停止通道后不能进行发送和接收操作。

```
uint8 UCAN_Stop_Wrapper(sint8 instance)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UCAN_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
//启动 CAN 通道
    if(UCAN_Stop_Wrapper(instance) != CMD_SUCCESS)
    {
        printf("UCAN_Stop_Wrapper fail!\r\n");
        return 0;
    }
    printf("UCAN_Stop_Wrapper OK! \r\n");
```

3.7 关闭

3.7.1 UCAN_Close_Wrapper

这个函数用于关闭 CAN 通道，应该在程序退出前调用。

```
uint8 UCAN_Close_Wrapper(sint8 instance)
```

参数说明：

instance

【IN】设备实例号，大于 0 为有效输入值，由函数 UCAN_Open_Wrapper 执行后的返回结果做为此函数的输入参数。

返回值：

如果函数执行成功返回值是 0，否则参看函数返回值枚举类型 CMD_RETURN_STATUS 定义的返回值及说明。

备注：

例子：

```
//启动 CAN 通道
    if(UCAN_Close_Wrapper(instance) != CMD_SUCCESS)
    {
        printf("UCAN_Close_Wrapper fail!\r\n");
        return 0;
    }
    printf("UCAN_Close_Wrapper OK! \r\n");
```

更新记录：

日期	软件版本	手册版本	更新内容
20220215	V1.0.0	V1.0.0	初版
20220227	V1.0.1	V1.0.1	接收消息结构体增加接收序号参数
20220420	V1.2.0	V1.2.0	更新时隙配置和发送数据接口
20220620	V1.2.1	V1.2.1	获得 POC 状态函数增加获得当前周期功能，增加数据回放和监控接收功能。
20220705	V1.2.2	V1.2.2	提供唤醒功能支持
20220815	V1.2.2	V1.2.4	修改部分勘误
20221227	V1.2.3	V1.2.5	更新部分参数说明
20231115	V1.2.6	V1.2.6	增加 Fibex 数据库处理函数和 CAN 接口函数，更新部分 FlexRay 函数。
20240309	V1.2.7	V1.2.7	增加 FlexRay 和 CAN 接口清除接收缓冲区函数