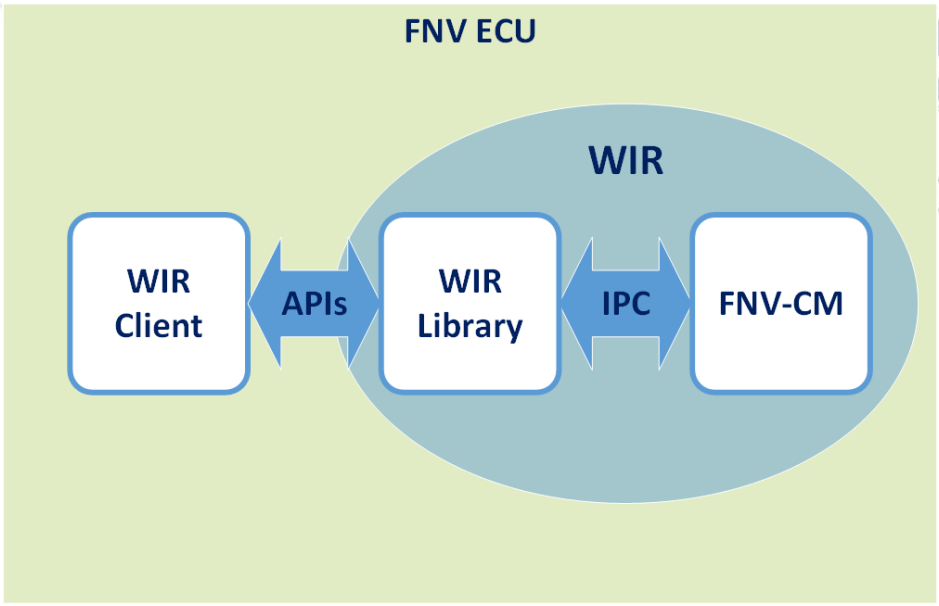
FNV2 Programmer Guide (11): Wireless Interface Router (WIR) APIs

Overview

Wireless Interface Router (WIR) can be envisioned as a middleware component that enables in-vehicle software applications (WIR Clients) to establish a cloud connection via any edge network interface available in the vehicle. A WIR Client can leverage this functionality by invoking the APIs provided by WIR. Application are required to be power aware and release the network interface on transition to low power state or power down of the module. The WIR framework consists of the following components:



The Wireless Interface Router APIs get built into a shared library: libwir.so

Pre-requisites

-libwir.so uses mq for IPC. Ensure mq is running on SYNC4 before using the Wireless Interface Router APIs.
-WIR Client should link against the following libraries in their Makefile to ensure full integration of the WIR library:

```
LIBS += -lwir
LIBS += -lmq
```

Implementation

The below classes and callback function details are provided in detail here: Wireless Interface Router APIs.

The following is a basic sequence of API calls that the WIR Client should implement.

1. To enable receipt of alerts, notifications, requests etc. from WIR, the WIR Client should derive a sub-class from the following class and override all the pure virtual callback functions defined under this class.

```
2. class ExampleClientCallback : public
   fnv::cm::WirelessInterfaceRouterCallbackInterface {
3. ...
   };
```

4. To enable sending of requests, alerts etc to the WIR, the WIR client should instantiate the following class and call the WIR APIs via this instance/object.

```
fnv::cm::WirelessInterfaceRouter wirInstance;
```

5. Subsequently, the WIR Client should be initialized and should register the callback interface class with the WIR using the initialize(...) call.


```
WIRRet_t initialize(const std::string& appId, const
WirelessInterfaceRouterCallbackInterface& cb)
```

appId should be an unique ID associated with the WIR Client, Application (Package) IDs

Ensure that the WirelessInterfaceRouterCallbackInterface instance (passed by reference to the initialize(...) call) is valid for the lifetime of the WIR API calls.

Example call:

```
WIRRet_t wirRetCode = wirInstance.initialize("exCl", exampleClientCb);
```

6. At this juncture, the WIR should be ready to receive requests and other API calls from the WIR Clients. A request to allocate network interface can be made using the following API call.

allocateNetworkInterface()

```
fnv::cm::WirIntent wirIntent =
(fnv::cm::WirIntent)(fnv::cm::WirSpecialIntent(fnv::cm::NET_IFACE_TCUCE
LL, fnv::cm::APN_FORD, fnv::cm::NET_IFACE_PRI_FG_2));
uint32_t allocId;
wirRetCode = wirInstance.allocateNetworkInterface(wirIntent,
                                                  allocId);
```

An unique allocId is returned to the WIR Client by WIR that would need to be held by the client and reference in subsequent API calls.

7. The status of the allocation request and the IP address of the allocated network interface will be communicated to the WIR Client using the following callback within the callback interface:

```
8. void networkInterfaceAllocationStatusCb(const uint32_t allocId,
                                           const
   fnv::cm::WirNetworkIfAllocation& wirNetIfAlloc)
```

9. Subsequently, the WIR Client may use the allocated network interface to set up a data transport and perform socket operations and data transactions. After the client is finished with the network interface, it may call the releaseNetworkInterface() API to release the allocation of network interface.

```
WIRRet_t releaseNetworkInterface(const uint32_t& allocId)
```

Threading model

The WIR library creates two threads from the WIR client's context.

- Receive thread - this thread would be blocked on receiving messages from FNV-CM. Once a message is received, the message would get enqueued onto an internal queue for further processing.
- Processing thread - this thread would process all the messages received from FNV-CM after dequeuing them from the internal queue. All the callbacks will be executed on this thread. The responses from the WIR client are also processed on this thread.

Sample Source Code

WIR API Indications/Callbacks Sample

sync_client_cb.hpp

```
#include <logger_api.hpp>
#include "WirIntent.hpp"
#include "WirIfCommon.hpp"
#include "WirNetworkIfAllocation.hpp"
```

```
namespace syncecu {
namespace fci {static Logger&
logger(*Logger::getInstance("WIRAPI.WirClient"));/**
 * Each client implements WirelessInterfaceRouterCallbackInterface to handle
 * notifications/alerts from Connection Manager. Instance of the
 * implemented class shall be used to register the callbacks with
 * WirelessInterfaceRouter.
 */
class SyncClientCallback: public
fnv::cm::WirelessInterfaceRouterCallbackInterface
{
public:    SyncClientCallback()
    {
        //@future
    }    ~SyncClientCallback()
    {
        //@future
    }    /**
     * @brief
     *    Callback to handle notification received from WIR on Network
Interface
     *    Allocation status.
     * @param   allocId
     *    Allocation ID corresponding to a particular Network Interface.
     *    WIR assigns this unique ID to the client the very first time the
     *    client requests a Network Interface.
     * @param   allocStatus
     *    Network Interface Allocation Status could be one of
     *    NET_IFACE_ALLOC_FAILURE or
     *    NET_IFACE_ALLOC_SUCCESS or
     *    NET_IFACE_ALLOC_INQUEUE or
     *    NET_IFACE_ALLOC_ILLEGAL
     * @param   ipAddr
     *    IP Address for the allocated Network Interface.
     * @param interfaceType
     *    The outgoing interface:
     *    NET_IFACE_TCUCELL
     *    NET_IFACE_TCUWIFI
     *    NET_IFACE_SYNCWIFI
     *    NET_IFACE_SYNCAPPL
     *    NET_IFACE_ERR
     * @return
     *    None
     */
    void networkInterfaceAllocationStatusCb(const uint32_t allocId,
                                            const
fnv::cm::WirNetworkIfAllocation& wirNetIfAlloc) override
    {
            // n/w is UP!
            logger.c("networkInterfaceUpCb: test/mock wlan interface
requested for is UP/READY!");
            logger.c("networkInterfaceUpCb: test/mock allocId: %u, ipAddr:
%s",
                    allocId,
                    wirNetIfAlloc.getIpAddr().c_str());
    }    /**
```

```
     * @brief
     *    Callback to handle notification Network Interface down
     *    status.
     * @param   allocId
     *    Allocation ID corresponding to a particular Network Interface.
     *    WIR assigns this unique ID to the client the very first time the
     *    client requests a Network Interface.
     * @return
     *    None
     */
    void networkInterfaceDownCb(const uint32_t allocId,
                                const fnv::cm::WirNetworkIfAllocation&
wirNetIfAlloc) override
    {
        if (allocId == 987) // test/mock wlanAllocId = 987
        {
            // the Wlan n/w is DOWN
            logger.c("Wlan interface requested for is DOWN!");
        }
    }    /**
     * @brief
     *    Callback to handle notification Network Interface Up
     *    status.
     * @param   allocId
     *    Allocation ID corresponding to a particular Network Interface.
     *    WIR assigns this unique ID to the client the very first time the
     *    client requests a Network Interface.
     * @param   ipAddr
     *    IP Address for the allocated Network Interface.
     * @return
     *    None
     */
    void networkInterfaceUpCb(const uint32_t allocId,
                              const fnv::cm::WirNetworkIfAllocation&
wirNetIfAlloc) override
    {
        // @future
    }    /**
     * @brief
     *    Callback to handle notification of network policy update.
     *    When network policy is changed from the SDN.
     * @param   permission
     *    A bit map of permissions, indicating which interfaces can be used
     */
    void networkPolicyUpdateCb(const uint16_t permission) override
    {
        // @future
    }    /**
     * @brief
     *    Callback for client to pause sending data on an interface.
     * @param   allocId
     *    Allocation ID corresponding to a particular Network Interface.
     *    WIR assigns this unique ID to the client the very first time the
     *    client requests a Network Interface.
     */
    void dataTransportPauseCb(const uint32_t allocId) override
    {
```

```
        // @future
    }    /**
     * @brief
     *    Callback for client to resume sending data on an interface.
     * @param   allocId
     *    Allocation ID corresponding to a particular Network Interface.
     *    WIR assigns this unique ID to the client the very first time the
     *    client requests a Network Interface.
     */
    void dataTransportResumeCb(const uint32_t allocId) override
    {
        // @future
    }    /**
     * @brief
     *    Callback for client to stop sending data on an interface.
     * @param   allocId
     *    Allocation ID corresponding to a particular Network Interface.
     *    WIR assigns this unique ID to the client the very first time the
     *    client requests a Network Interface.
     */
    void dataTransportStopCb(const uint32_t allocId) override
    {
        // @future
    }    /**
     * @brief
     *    WIR querys clients about activity over an allocated Network
     *    Interface. The client will call
reportActivityOverNetworkInterface()
     *    to report the status.
     * @param   allocId
     *    Allocation ID corresponding to a particular Network Interface.
     *    WIR assigns this unique ID to the client the very first time the
     *    client requests a Network Interface.
     */
    void requestActivityOverNetworkInterface(const uint32_t allocId, const
fnv::cm::DataTransportState_t& ifState) override
    {
        // @future
    }
};} // namespace fci
} // namespace syncecu
```


WIR Client Sample

sampleWirApp.cpp

```
#include <WirIf.hpp>

#include "sync_client_cb.hpp"
#include <iostream>
#include <thread>
#include <WirIfCommon.hpp>
#include <WirIntent.hpp>
#include <WirForegroundIntent.hpp>
#include <WirBackgroundBestEffortIntent.hpp>
#include <WirBackgroundGuaranteedIntent.hpp>
```

```
#include <WirOffPeakIntent.hpp>
#include <WirSpecialIntent.hpp>

int main(int argc,
         char* argv[])
{
    int clOpts;
    bool wrongArg = false;
    char* appId = NULL;
    std::string appIdStr = "ECGFCI";
    while ((clOpts = getopt(argc, argv, "a:")) != -1)
    {
        switch (clOpts)
        {
            case 'a':
                appId = optarg;
                break;
            default:
                wrongArg = true;
                std::cout << "Wrong argument!" << std::endl;
                std::cout << "Usage: sampleWirApp [-a appName]" << std::endl;
                std::cout << "-where: appName : name required for
initialization" << std::endl;
                break;
            }
    }
    if (wrongArg)
    {
        return EXIT_FAILURE;
    }
    if (!appId)
    {
        std::cout << "Empty argument for appName!" << std::endl;
        std::cout << "Defaulting to " << appIdStr << std::endl;
    }
    else
    {
        appIdStr = std::string(appId);
    }
    // Sublcass WirelessInterfaceRouterCallbackInterface
    syncecu::fci::SyncClientCallback syncClCbInst;

    // Instantiate WirelessInterfaceRouter
    fnv::cm::WirelessInterfaceRouter wirInstance;
    fnv::cm::WIRRet_t wirRetCode;
    //uint32_t expiry = 0; // 0 means infinite time

    // Initialize WIR client and pass reference to callback instance
    wirRetCode = wirInstance.initialize(appIdStr,
                                        syncClCbInst);
    std::cout << "Returned " << wirRetCode << " from initialize()" <<
std::endl;

    //
==========================================================================
    // To setup a foreground intent, which will go on the default cellular
interface
```

```
    // any of the following three options can be used
    //
================================================================================
    // fnv::cm::WirIntent wirIntent =
fnv::cm::WirForegroundIntent(fnv::cm::WIFI_PREF_NO,
fnv::cm::NET_IFACE_PRI_FG_2);
    // fnv::cm::WirIntent wirIntent =
fnv::cm::WirForegroundIntent(fnv::cm::WIFI_PREF_NO);
    // fnv::cm::WirIntent wirIntent = fnv::cm::WirForegroundIntent();

    //
================================================================================
    // To setup a WirBackgroundBestEffort intent, which will go on the
default wlan interface
    // any of the following three options can be used
    //
================================================================================
    // fnv::cm::WirIntent wirIntent =
fnv::cm::WirBackgroundBestEffortIntent();
    // fnv::cm::WirIntent wirIntent =
fnv::cm::WirBackgroundBestEffortIntent(expiry);
    // fnv::cm::WirIntent wirIntent =
fnv::cm::WirBackgroundBestEffortIntent(expiry, fnv::cm::NET_IFACE_PRI_BG_1);

    //
================================================================================
    // To setup a WirBackgroundGuaranteed intent, which will go on the
default wlan interface
    // any of the following 4 options can be used
    //
================================================================================
    // fnv::cm::WirIntent wirIntent =
fnv::cm::WirBackgroundGuaranteedIntent();
    // fnv::cm::WirIntent wirIntent =
fnv::cm::WirBackgroundGuaranteedIntent(expiry);
    // fnv::cm::WirIntent wirIntent =
fnv::cm::WirBackgroundGuaranteedIntent(expiry, fnv::cm::NET_IFACE_PRI_BG_1);
    // fnv::cm::WirIntent wirIntent =
fnv::cm::WirBackgroundGuaranteedIntent(expiry, fnv::cm::NET_IFACE_PRI_BG_1,
fnv::cm::OFFP_NO);

    //
================================================================================
    // To setup a WirOffPeakIntent intent, which will go over cellular
interface
    // during off peak hours any of the following 3 options can be used
    //
================================================================================
    // fnv::cm::WirIntent wirIntent = fnv::cm::WirOffPeakIntent();
    // fnv::cm::WirIntent wirIntent = fnv::cm::WirOffPeakIntent(expiry);
    // fnv::cm::WirIntent wirIntent = fnv::cm::WirOffPeakIntent(expiry,
fnv::cm::NET_IFACE_PRI_OP_1);

    //
================================================================================
    // To setup a WirSpecialIntent intent, which will go over cellular
interface
```

```
    // any of the following 3 options can be used
    //
================================================================================
    // fnv::cm::WirIntent wirIntent = fnv::cm::WirSpecialIntent();
    // fnv::cm::WirIntent wirIntent =
fnv::cm::WirSpecialIntent(fnv::cm::NET_IFACE_TCUCELL, fnv::cm::APN_FORD);
    fnv::cm::WirIntent wirIntent =
(fnv::cm::WirIntent)(fnv::cm::WirSpecialIntent(fnv::cm::NET_IFACE_TCUCELL,
fnv::cm::APN_FORD, fnv::cm::NET_IFACE_PRI_FG_2));
    std::cout << "IntentType: " << wirIntent.getIntentType() << std::endl;
    uint32_t allocId;
    std::cout << "Calling allocateNetworkInterface API" << std::endl;
    wirRetCode = wirInstance.allocateNetworkInterface(wirIntent,
                                                      allocId);
    std::cout << "return from allocateNetworkInterface API, ret is: "
              << wirRetCode
              << " allocId is: "
              << allocId
              << std::endl;
    std::cout << "Sleeping for 5 seconds before sending release" <<
std::endl;
    sleep(5);
    std::cout << "Calling releaseNetworkInterface API" << std::endl;
    wirRetCode = wirInstance.releaseNetworkInterface(allocId);
    std::cout << "return from releaseNetworkInterface API, ret is: "
              << wirRetCode
              << std::endl;
    if (wirRetCode == fnv::cm::WIR_SUCCESS)
        std::cout << "allocId: "
                  << allocId
                  << " released successfully"
                  << std::endl;
    return 0;
}
```