Wireless Interface Router APIs

WirelessInterfaceRouter Class

```
/**
 * Entry point for applications to request Wireless Interface Router (WIR)
 * for a network interface allocated for data transport and released when
 * no longer needed.
 *
 * Client accesses these APIs by creating an instance of
WirelessInterfaceRouter.
 */
class WirelessInterfaceRouter
{
public:
    WirelessInterfaceRouter();
    virtual ~WirelessInterfaceRouter();


    /**
     * @brief
     *    Initialize the WirelessInterfaceRouter with application ID and
     *    required callbacks
     * @param IN:  appId
     *    Application ID of the client
     * @param IN:  callback
     *    Pointer to a list of callbacks
     * @return
     *    WIR_SUCCESS if initialization is successful
     *    WIR_ERROR   if initialization fails
     */
    WIRRet_t initialize(const std::string& appId,
WirelessInterfaceRouterCallbackInterface& cb);


    /**
     * @brief
     *    Client requests WIR for a desired Network Interface to be
allocated.
     *    This is a blocking call.
     * @param IN:  intent
```

```
         *    Intent describing the criteria to be met by the network interface
         * @param OUT:  allocId
         *    Allocation ID assigned by WIR in response to this request.
         *    request.
         * @return
         *    WIR_SUCCESS if processed successfully
         *    WIR_ERROR   if any error
         */
        WIRRet_t allocateNetworkInterface(WirIntent& intent, uint32_t& allocId);


        /**
         * @brief
         *    Client queries WIR to provide status on its request for a
         *    network interface.
         *    This is a blocking call.
         * @param IN:  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.
         * @parm OUT:  WirNetworkIfAllocation
         *     The class contains the following parameters
         * allocStatus
         *    Indicates the status of Network Interface allocation.
         *    Status code can be one of the following enums:
         *    NET_IFACE_ALLOC_FAILURE
         *    NET_IFACE_ALLOC_SUCCESS
         *    NET_IFACE_ALLOC_INQUEUE
         *    NET_IFACE_ALLOC_ILLEGAL
         * ipAddr
         *    IP Address for the allocated Network Interface.
         * interfaceType
         *    Represents the type of allocated edge Network Interface.
         *    It can be one of the following enums.
         *    NET_IFACE_TCUCELL
         *    NET_IFACE_TCUWIFI
         *    NET_IFACE_SYNCWIFI
         *    NET_IFACE_SYNCAPPL
         * @return
         *    WIR_SUCCESS if the call is successfully processed
         *    WIR_ERROR   if any error
         */
        WIRRet_t getNetworkInterfaceAllocationStatus(const uint32_t& allocId,
                                                     WirNetworkIfAllocation&
    wirNetIfAlloc);


        /**
         * @brief
         *    Client queries WIR to get a list of network interfaces it is
         *    allowed to use.
         * @param OUT:  permission
         *    A bit map of permissions, indicating which interfaces can be used
         * @return
         *    WIR_SUCCESS if the call is successfully processed
         *    WIR_ERROR   if any error
         */
        WIRRet_t getNetworkPolicy(uint16_t& permission);


        /**
         * @brief
         *    Query availble network interfaces. Availble network interface
         *    table returned based on application permissions in the policy table
         * @param OUT:  ifaceTable
         *    Table of interfaces that are currently active
         * @return
         *    WIR_SUCCESS if the call is successfully processed
         *    WIR_ERROR   if any error
         */
         WIRRet_t getActiveNetworkInterfaces(InterfaceTable_t& ifaceTable);


        /**
         * @brief
         *    After opening a socket on an interface, applications will call
         *    setFnvSocketOpt to accommodate selection of MPTCP and other default
         *    FNV socket options as they become available.
         * @param IN:  sockFd
         *    socket file descriptor of the socket the application has created
         * @return
         *    WIR_SUCCESS if the call is successfully processed
         *    WIR_ERROR   if any error
         */
        WIRRet_t setFnvSocketOpt(const int& sockFd);


        /**
         * @brief
         *    Client's response to WIR's query about activity over an allocated
         *    Network Interface.
         *    This is a non-blocking call.
         * @param IN:  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 OUT:  ifState
         *    Denotes the state of data transport over the Network Interface.
         *    It can take one of the following enums:
         *    NET_DT_ACTIVE
         *    NET_DT_INACTIVE
         * @return
         *    WIR_SUCCESS if the call is successfully processed
         *    WIR_ERROR   if any error
         */
        WIRRet_t reportActivityOverNetworkInterface(const uint32_t& allocId,
                                                    const DataTransportState_t&
    ifState);


        /**
         * @brief
         *    Client requests WIR to release the Network Interface that got
    allocated for it
```

```
     *    This is a non-blocking call.
     * @param IN:  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
     *    WIR_SUCCESS if the call is successfully processed
     *    WIR_ERROR   if any error
     */
    WIRRet_t releaseNetworkInterface(const uint32_t& allocId);


    /**
     * @brief
     *    Close the WirelessInterfaceRouter with application ID
     * @param IN:  appId
     *    Application ID of the client
     * @return
     *    WIR_SUCCESS if close is successful
     *    WIR_ERROR   if close fails
     */
    WIRRet_t close(const std::string& appId);


private:
    // Reference to underlying ipc method used: mq
    WirImpl* m_pImpl;
};
```

WirelessInterfaceRouterCallbackInterface Class

```
/**
 * 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 WirelessInterfaceRouterCallbackInterface
{
public:
    /**
     * @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.
     * @parm    WirNetworkIfAllocation
     *     The class contains the following parameters
     * allocStatus
     *    Indicates the status of Network Interface allocation.
     *    Status code can be one of the following enums:
     *    NET_IFACE_ALLOC_FAILURE
     *    NET_IFACE_ALLOC_SUCCESS
     *    NET_IFACE_ALLOC_INQUEUE
     *    NET_IFACE_ALLOC_ILLEGAL
     * ipV4Addr
     *    IP Address for the allocated Network Interface.
     * interfaceType
     *    Represents the type of allocated edge Network Interface.
     *    It can be one of the following enums.
     *    NET_IFACE_TCUCELL
     *    NET_IFACE_TCUWIFI
     *    NET_IFACE_SYNCWIFI
     *    NET_IFACE_SYNCAPPL
     */
    virtual void networkInterfaceAllocationStatusCb(const uint32_t allocId,
                                                    const
WirNetworkIfAllocation& wirNetIfAlloc) = 0;

    /**
     * @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.
     * @parm    WirNetworkIfAllocation
     *     The class contains the following parameters
     * allocStatus
     *    Indicates the status of Network Interface allocation.
     *    Status code can be one of the following enums:
     *    NET_IFACE_ALLOC_FAILURE
     *    NET_IFACE_ALLOC_SUCCESS
     *    NET_IFACE_ALLOC_INQUEUE
     *    NET_IFACE_ALLOC_ILLEGAL
     */
    virtual void networkInterfaceDownCb(const uint32_t allocId,
                                        const WirNetworkIfAllocation&
wirNetIfAlloc) = 0;

    /**
     * @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.
     * @parm    WirNetworkIfAllocation
     *     The class contains the following parameters
     * allocStatus
     *    Indicates the status of Network Interface allocation.
     *    Status code can be one of the following enums:
```

```
     *    NET_IFACE_ALLOC_FAILURE
     *    NET_IFACE_ALLOC_SUCCESS
     *    NET_IFACE_ALLOC_INQUEUE
     *    NET_IFACE_ALLOC_ILLEGAL
     * ipV4Addr
     *    IP Address for the allocated Network Interface.
     * interfaceType
     *    Represents the type of allocated edge Network Interface.
     *    It can be one of the following enums.
     *    NET_IFACE_TCUCELL
     *    NET_IFACE_TCUWIFI
     *    NET_IFACE_SYNCWIFI
     */
    virtual void networkInterfaceUpCb(const uint32_t allocId,
                                      const WirNetworkIfAllocation&
wirNetIfAlloc) = 0;

    /**
     * @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
     */
    virtual void networkPolicyUpdateCb(const uint16_t permission) = 0;

    /**
     * @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.
     */
    virtual void dataTransportPauseCb(const uint32_t allocId) = 0;

    /**
     * @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.
     */
    virtual void dataTransportResumeCb(const uint32_t allocId) = 0;

    /**
     * @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.
     */
    virtual void dataTransportStopCb(const uint32_t allocId) = 0;

    /**
```

```
     * @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.
     */
    virtual void requestActivityOverNetworkInterface(const uint32_t allocId,
const DataTransportState_t& ifState) = 0;

    // dtor
    virtual ~WirelessInterfaceRouterCallbackInterface()
    {
        // Auto-generated destructor stub
    }
};
```

WIR Intents

WirForegroundIntent Class

```
/**
 * This class is used by the client to specify the intent and is
 * passed in as a parameter to allocateNetworkInterface call on
 * WirelessInterfaceRouter.
 */
class WirForegroundIntent : public WirIntent
{
public:
    WirForegroundIntent();
    WirForegroundIntent(WifiPreferredFlag_t wifiPref,
                        NetworkInterfacePriorityLevelForeground_t priority =
NET_IFACE_PRI_FG_2);
    ~WirForegroundIntent();
    WifiPreferredFlag_t getWifiPref() const;
    void setWifiPref(WifiPreferredFlag_t wifiPref);
    NetworkInterfacePriorityLevelForeground_t getPriority() const;
    void setPriority(NetworkInterfacePriorityLevelForeground_t priority);
protected:
    WifiPreferredFlag_t m_wifiPref;
    NetworkInterfacePriorityLevelForeground_t m_priority;
};
```

WirBackgroundBestEffortIntent Class

```
/**
 * This class is used by the client to specify the intent and is
 * passed in as a parameter to allocateNetworkInterface call on
 * WirelessInterfaceRouter.
 */
class WirBackgroundBestEffortIntent : public WirIntent
{
public:
    WirBackgroundBestEffortIntent();
    WirBackgroundBestEffortIntent(uint32_t expiry,
                                  NetworkInterfacePriorityLevelBackground_t
priority =
                                  NET_IFACE_PRI_BG_1);
    ~WirBackgroundBestEffortIntent();
    NetworkInterfacePriorityLevelBackground_t getPriority() const;
    void setPriority(NetworkInterfacePriorityLevelBackground_t priority);

protected:
    NetworkInterfacePriorityLevelBackground_t m_priority;
};
```

WirBackgroundGuaranteedIntent Class

```
/**
 * This class is used by the client to specify the intent and is
 * passed in as a parameter to allocateNetworkInterface call on
 * WirelessInterfaceRouter.
 */
class WirBackgroundGuaranteedIntent: public WirIntent
{
public:
    WirBackgroundGuaranteedIntent();
    WirBackgroundGuaranteedIntent(uint32_t expiry,
                                  NetworkInterfacePriorityLevelBackground_t
priority =
                                          NET_IFACE_PRI_BG_1,
                                  OffpeakFlag_t offpeak = OFFP_NO);
    ~WirBackgroundGuaranteedIntent();
    OffpeakFlag_t getOffpeak() const;
    void setOffpeak(OffpeakFlag_t offpeak);
    NetworkInterfacePriorityLevelBackground_t getPriority() const;
    void setPriority(NetworkInterfacePriorityLevelBackground_t priority);

protected:
    OffpeakFlag_t m_offpeak;
    NetworkInterfacePriorityLevelBackground_t m_priority;
};
```

WirSpecialIntent Class

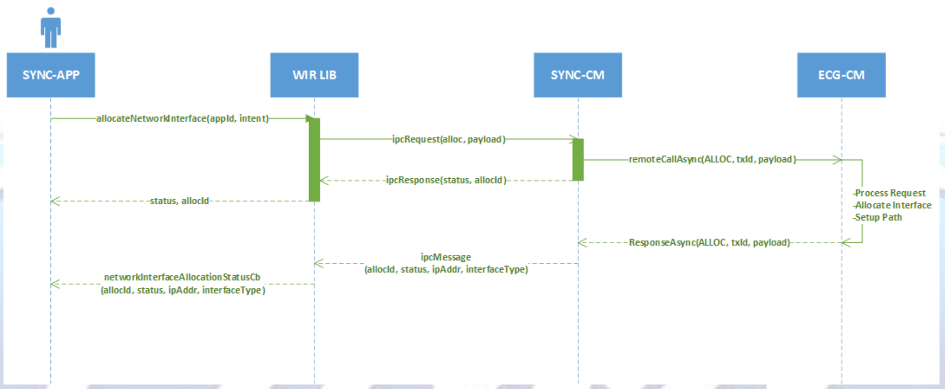
```
/**
 * This class is used by the client to specify the intent and is
 * passed in as a parameter to allocateNetworkInterface call on
 * WirelessInterfaceRouter.
 */
class WirSpecialIntent: public WirIntent
{
public:
    WirSpecialIntent();
    WirSpecialIntent(NetworkInterfaceType_t iface,
                     CellularApnType_t apn,
                     NetworkInterfacePriorityLevelForeground_t priority =
                             NET_IFACE_PRI_FG_2);
    ~WirSpecialIntent();
    CellularApnType_t getApn() const;
    void setApn(CellularApnType_t apn);
    NetworkInterfaceType_t getIface() const;
    void setIface(NetworkInterfaceType_t iface);
    const WlanProfile_t& getProfile() const;
    void setProfile(const WlanProfile_t& profile);
    NetworkInterfacePriorityLevelForeground_t getPriority() const;
    void setPriority(NetworkInterfacePriorityLevelForeground_t priority);

private:
    NetworkInterfaceType_t m_iface;   // Special Intent
    CellularApnType_t m_apn;      // Cellular special intent only, if
required
    WlanProfile_t m_profile;  // Wifi special intent only, if required
    NetworkInterfacePriorityLevelForeground_t m_priority;
};
```

WirOffPeakIntent Class

```
/**
 * This class is used by the client to specify the intent and is
 * passed in as a parameter to allocateNetworkInterface call on
 * WirelessInterfaceRouter.
 */
class WirOffPeakIntent : public WirIntent
{
public:
    WirOffPeakIntent();
    WirOffPeakIntent(uint32_t expiry, NetworkInterfacePriorityLevelOffPeak_t
priority = NET_IFACE_PRI_OP_1);
    ~WirOffPeakIntent();
    NetworkInterfacePriorityLevelOffPeak_t getPriority() const;
    void setPriority(NetworkInterfacePriorityLevelOffPeak_t priority);
protected:
    NetworkInterfacePriorityLevelOffPeak_t m_priority;
};
```

Figure 1 below depicts the flow of the API call and the various actors involved for requesting a network interface allocation.



SYNC-APP
WIR LIB
SYNC-CM
ECG-CM
allocateNetworkInterface(appId, intent)
ipcRequest(alloc, payload)
remoteCallAsync(ALLOC, txId, payload)
ipcResponse(status, allocId)
status, allocId
-Process Request
-Allocate Interface
-Setup Path
ResponseAsync(ALLOC, txId, payload)
ipcMessage (allocId, status, ipAddr, interfaceType)
networkInterfaceAllocationStatusCb (allocId, status, ipAddr, interfaceType)

Figure 1