Product Development

Vehicle Network Manager APIs

Version 1.0

Version Date: October 17, 2019

UNCONTROLLED COPY IF PRINTED

FORD CONFIDENTIAL

```
/**
 * @brief
 *    A Gateway-ECU (aka an Edge-ECU) hosts one or more wireless connectivity
 *    resources (Cellular or WiFi interfaces). Access request to a desired
 *    wireless interface can originate from applications resident on any
 *    Ethernet-Connected ECU called Host-ECU. Access is realized by creating
 *    a VLAN and by establishing appropriate routing-polices on both the
 *    Host and Gateway ECUs.
 *    This is a blocking call.
 * @param IN:  name
 *    A descriptive name given to the VLAN by the Caller (eg. vlan3)
 *    This name is used by both the caller and VNM for reference.
 * @param IN:  host_number
 *    The number assigned to the Host-ECU by the caller.
 *    Range: 1 to 14.
 * @param IN:  vlanid
 *    VLAN ID chosen by the caller.
 *    Range: 1 to 20.
 * @param IN:  gw_host_number
 *    The number assigned to the Gateway-ECU by the caller.
 *    Range: 1 to 14.
 * @param IN:  mtu
 *    Largest packet size (in octets) that can be transmitted
 *    over the edge wireless interface.
 *    Value: 1496 or smaller
 * @param IN:  svc_level
 *    Desired differentiated service level for the intended traffic.
 *    Range: 1 to 5 (1 = Expedited Forwarding
 *                   2 = Assured Forwarding (Maximize Reliability)
 *                   3 = Assured Forwarding (Minimize Delay)
 *                   4 = Best Effort
 *                   5 = Best Effort)
 * @param OUT:  ipaddr
 *    IP address the caller shall provide to its client to use as source
 *    address when establishing the intended connection.
 * @return
 *    VNMIPC_SUCCESS if processed successfully
 *    VNMIPC_ERROR   if any error
 */
VnmIpcRet_t ConfigVlanInterface(const std::string &name,
                                const std::string &host_number,
                                const std::string &vlanid,
                                const std::string &gw_host_number,
                                const std::string &mtu,
                                const std::string &svc_level,
                                std::string &ipaddr);
```

```
/**
 * @brief
 *    Caller requests VNM to remove a VLAN.
 *    This is a blocking call.
 * @param IN:  name
 *    Name of the VLAN to be removed.
 *    This is the same name given by the caller when the VLAN was
 *    created using ConfigVlanInterface().
 * @return
 *    VNMIPC_SUCCESS if processed successfully
 *    VNMIPC_ERROR   if any error
 */
VnmIpcRet_t FreeVlanInterface(const std::string &name);

/**
 * @brief
 *    Caller requests VNM to associate the VLAN interface and the desired
 *    edge wireless network interface in order to route the packets from
 *    one to the other.
 * @param IN:  name1
 *    Reference name of the VLAN.
 *    This is the same name given by the caller
 *    when the VLAN was created using ConfigVlanInterface.
 * @param IN:  name2
 *    For Cellular, this is the designated APN name.
 *    For WLAN STA, this is the physical interface name.
 * @param IN:  svclevel
 *    Desired differentiated service level for the intended traffic.
 *    Range: 1 to 5 (1 = Expedited Forwarding
 *                   2 = Assured Forwarding (Maximize Reliability)
 *                   3 = Assured Forwarding (Minimize Delay)
 *                   4 = Best Effort
 *                   5 = Best Effort)
 * @param OUT:  linkid
 *    ID referencing the linkage between the two
 *    interfaces.
 * @return
 *    VNMIPC_SUCCESS if initialization is successful
 *    VNMIPC_ERROR   if initialization fails
 */
VnmIpcRet_t LinkNetworks(const std::string &name1,
                         const std::string &name2,
                         const std::string &svclevel,
                         std::string &linkid);
```

```
/**
 * @brief
 *    Caller requests VNM to disassocicate the linkage established via
 *    LinkNetworks / LinkNetworksforWhs.
 *    This is a blocking call.
 * @param IN:  linkid
 *    This is the same linkid VNM returnd to the caller for
 *    LinkNetworks() / LinkNetworksforWhs().
 * @return
 *    VNMIPC_SUCCESS if processed successfully
 *    VNMIPC_ERROR   if any error
 */
VnmIpcRet_t UnlinkNetworks(const std::string &linkid);

/**
 * @brief
 *    BIND9 on ECG plays the role of a forwarder wherein it forwards DNS
 *    queries to external servers and forwards DNS responses to querying
 *    applications. For this purpose, it manages a dynamic table of DNS
 *    server entries called forwarders.
 *    Caller request VNM to add a forwarder entry for an upstream
 *    service/network.
 * @param IN:  service_name
 *    Physical or logical name of the wireless interface (eg. wlan1)
 * @param IN:  ip_address
 *    IP address of the DNS name server.
 * @param IN:  hints
 *    Forwarding Hints define the order in which the name servers are tried
 *    when a name is resolved.
 *    Hints shall be a union of one or more of the following bitmap values.
 *    bitmap values.
 *    FH_NONE - Primary DNS address
 *    FH_SECONDARY - Alternate DNS address
 *    FH_WIFI - Address is applicable to a WiFi network
 *    FH_LAN - Address is applicable to a private network
 * @return
 *    VNMIPC_SUCCESS if initialization is successful
 *    VNMIPC_ERROR   if initialization fails
 */
VnmIpcRet_t AddDnsForwarder(const std::string &service_name,
                            const std::string &ip_address,
                            uint32_t hints);
```

```
/**
 * @brief
 *    Caller request VNM to remove a DNS forwarder entry for a known
 *    upstream service/network.
 *    This is a blocking call.
 * @param IN:  service_name
 *    Physical or logical name of the wireless interface (eg. wlan1)
 * @return
 *    VNMIPC_SUCCESS if initialization is successful
 *    VNMIPC_ERROR   if initialization fails
 */
VnmIpcRet_t RemoveDnsForwarders(const std::string &service_name);


/**
 * @brief
 *    VNM stores contextual data, such as VLAN, routes, and rules that
 *    it creates when caller requests to establish a path. This needs
 *    to be cleared at certain times (eg. Caller start-up)
 *    Caller request VNM to clear the the contexts.
 *    This is a blocking call.
 * @return
 *    VNMIPC_SUCCESS if processed successfully
 *    VNMIPC_ERROR   if any error
 */
VnmIpcRet_t clearVnmContext(void);
```