Kvaser Remote Device API

Generated by Doxygen 1.7.3

Fri Mar 1 2013 11:50:44

Contents

Chapter 1

Remote Device API

THIS IS A PRELIMINARY VERSION AND SUBJECT TO CHANGE

This is an API for remote devices.

Chapter 2

Support

For support, contact support@kvaser.com

Chapter 3

Device discovery

The following is an example of calls that can be used for device discovery.

Initialize/Unload library

- kvrInitializeLibrary()
- kvrUnloadLibrary()

Get the default addresses used for discovering devices

- kvrDiscoveryGetDefaultAddresses()

Discover all devices to the device list

- kvrDiscoveryStart()
- kvrDiscoveryGetResults()

Save devices you want to remember or use.

- kvrDiscoveryStoreDevices()

Get device status

- kvrDeviceGetServiceStatusText()

Chapter 4

Local Configuration

When the remote device is connected to the host it can be configured. A device can hold a number of different profiles. The number of profiles that the device supports can be found by using kvrConfigNoProfilesGet(). Each profile contains a complete configuration. To configure a specific profile open it with kvrConfigOpenEx(). To activate a specific profile use kvrConfigActiveProfileSet(). The active profile is the one the device will use.

The following is an example of calls that are used to configure a device.

Initialize/Unload library

- kvrInitializeLibrary()
- kvrUnloadLibrary()

Get the number of supported profiles.

- kvrConfigNoProfilesGet()

Get and set the active profile.

- kvrConfigActiveProfileGet()
- kvrConfigActiveProfileSet()

Open the device's configuration area. If protected, you need to enter a password as well.

- kvrConfigOpen()
- kvrConfigOpenEx()

If password is unknown, it is possible to first clear entire area including password

- kvrConfigClear()

Write new configuration and read the configuration from the device

- kvrConfigSet()
- kvrConfigGet()

Close the device's configuration area for writing

- kvrConfigClose()

Chapter 5

Network information

The following is an example of calls that are used for network maintenance.

Initialize library

- kvrInitializeLibrary()

Start scan for existing WLAN networks and get the result

- kvrWlanStartScan()
- kvrWlanGetScanResults()

Get device's IP settings

- kvrNetworkGetAddressInfo()

Get status information; WLAN connection state, RSSI, RTT, TX power etc.

- kvrNetworkGetConnectionStatus()

Start sending 'ping' and get latest RTT (and RSSI) values.

- kvrNetworkConnectionTest()
- kvrNetworkGetRssiRtt()

Chapter 6

Module Documentation

6.1 Local configuration

Typedefs

- typedef int32_t kvrConfigHandle

Functions

- kvrStatus kvrConfigOpen (int32_t can_channel_no, int32_t mode, const char *password, kvrConfigHandle *handle)
- kvrStatus kvrConfigActiveProfileSet (int32_t can_channel_no, int32_t profile_number)
- kvrStatus kvrConfigActiveProfileGet (int32_t can_channel_no, int32_t *profile_number)
- kvrStatus kvrConfigNoProfilesGet (int32_t can_channel_no, int32_t *no_profiles)
- kvrStatus kvrConfigOpenEx (int32_t can_channel_no, int32_t mode, const char *password, kvrConfigHandle *handle, uint32_t profile_no)
- void kvrConfigClose (kvrConfigHandle handle)
- kvrStatus kvrConfigVerifyXml (const char *xml_buffer, char *err_buffer, uint32_t err_buffer_size)
- kvrStatus kvrConfigSet (kvrConfigHandle handle, const char *xml_buffer)
- kvrStatus kvrConfigGet (kvrConfigHandle handle, char *xml_buffer, uint32_t xml_buffer_size)
- kvrStatus kvrConfigInfoGet (int32_t can_channel_no, int32_t profile_no, char *xml_buffer, uint32_t xml_buffer_size)
- kvrStatus kvrConfigClear (kvrConfigHandle handle)

kvrConfigMode_xxx

Configuration mode.

- #define kvrConfigMode_R 0
- #define kvrConfigMode_RW 1
- #define kvrConfigMode_ERASE 2

6.1.1 Define Documentation

6.1.1.1 #define kvrConfigMode_ERASE 2

Erase and write.

6.1.1.2 #define kvrConfigMode_R 0

Read only.

Examples:

kvrConfig.c, and kvrNetworkConnectionTest.c.

6.1.1.3 #define kvrConfigMode_RW 1

Read/write.

Examples:

kvrConfig.c.

6.1.2 Typedef Documentation

6.1.2.1 typedef int32_t kvrConfigHandle

A configuration handle. Created by calling kvrConfigOpen() or kvrConfigOpenEx().

6.1.3 Function Documentation

6.1.3.1 kvrStatus kvrConfigActiveProfileGet (int32_t *can_channel_no*, int32_t * *profile_number*)

Get active profile. See Local Configuration.

Parameters

in	*can_channel_no*	CAN channel number
out	*profile_number*	

Returns

kvrOK on success or any other kvrStatus on failure.

6.1.3.2 kvrStatus kvrConfigActiveProfileSet (int32_t *can_channel_no*, int32_t *profile_number*)

Set active profile. See Local Configuration.

Parameters

in	*can_channel_no*	CAN channel number
in	*profile_number*	

Returns

kvrOK on success or any other kvrStatus on failure.

Note

A pause of ~2 seconds is needed after kvrConfigActiveProfileSet(), to allow CANlib time to discard the device.

6.1.3.3 kvrStatus kvrConfigClear (kvrConfigHandle *handle*)

Clear the device configuration area. This will also clear any previously set device password. The handle must be opened kvrConfigMode_ERASE and closed with kvrConfigClose().

Parameters

in	*handle*	A configuration handle.

Returns

kvrOK on success or any other kvrStatus on failure.

6.1.3.4 void kvrConfigClose (kvrConfigHandle *handle*)

Closes the device's configuration area without programming it. Programming is done with kvrConfigSet().

Parameters

in	*handle*	A configuration handle

Note

When using kvrConfigMode_RW or kvrConfigMode_ERASE, a pause of ~2 seconds is needed after kvrConfigClose(), to allow CANlib time to discard the device.

Examples:

kvrConfig.c, and kvrNetworkConnectionTest.c.

6.1.3.5 kvrStatus kvrConfigGet (kvrConfigHandle *handle*, char * *xml_buffer*, uint32_t *xml_buffer_size*)

Reads the device configuration. On successful return, the buffer will contain a valid C string with the configuration in XML format. The handle must be opened kvrConfigMode_R or kvrConfigMode_RW and closed with kvrConfigClose().

Parameters

in	*handle*	A configuration handle.
out	*xml_buffer*	A pointer to the data buffer.
in	*xml_buffer_size*	The buffer size.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConfig.c.

6.1.3.6 kvrStatus kvrConfigInfoGet (int32_t *can_channel_no*, int32_t *profile_no*, char * *xml_buffer*, uint32_t *xml_buffer_size*)

Reads a simplified version of A device configuration PROFILE. On successful return, the buffer will contain a valid C string of this in XML format. Since the function takes a CAN channel rather than a kvrConfigHandle, it is not necessary to know the configuration password. Note that the partial XML data returned is not enough to reconfigure a device.

Parameters

in	*can_channel_no*	CAN channel number.
in	*profile_no*	Profile number
out	*xml_buffer*	A pointer to the data buffer.
in	*xml_buffer_size*	The buffer size.

Returns

kvrOK on success, kvrERR_BLANK when the profile is empty, or any other kvrStatus on failure.

6.1.3.7 kvrStatus kvrConfigNoProfilesGet (int32_t *can_channel_no*, int32_t * *no_profiles*)

Get the maximum number of profile(s) the device can store. See Local Configuration.

Parameters

in	*can_channel_no*	CAN channel number
out	*no_profiles*	

Returns

kvrOK on success or any other kvrStatus on failure.

6.1.3.8 kvrStatus kvrConfigOpen (int32_t *can_channel_no*, int32_t *mode*, const char * *password*, kvrConfigHandle * *handle*)

Open a configuration handle to the device. It should later be closed with kvrConfigClose().

This configuration handle is used both for changing the device configuration, reading status information, e.g. kvrNetworkGetConnectionStatus(), and issuing some other commands such as e.g. kvrNetworkConnectionTest() and kvrWlanStartScan().

To change the configuration, you need to open the configuration with kvrConfigMode_RW before calling kvrConfigSet().

To read the configuration, you can open the configuration with either kvrConfigMode_RW or kvrConfigMode_R before calling kvrConfigGet().

Setting a password is done through the configuration (with kvrConfigSet()). Resetting a password can be done by erasing the whole configuration with kvrConfigClear(), while first opening the configuration with kvrConfigMode_ERASE and supplying an empty password.

Before calling any other function, you must open the configuration with kvrConfigMode_R.

Parameters

in	*can_channel_no*	CAN channel number.
in	*mode*	Can be set to one of kvrConfigMode_xxx
in	*password*	The password as a C string. Use an empty string, i.e. "", if no password is required.
out	*handle*	A configuration handle

Returns

kvrOK on success or any other kvrStatus on failure.

Note

kvrConfigOpen() will operate on the active profile. See Local Configuration.
When using kvrConfigMode_RW or kvrConfigMode_ERASE, a pause of ~2 seconds is needed after kvrConfigClose(), to allow CANlib time to discard the device.

See also

kvrConfigOpenEx()

Examples:

kvrConfig.c, and kvrNetworkConnectionTest.c.

6.1.3.9 kvrStatus kvrConfigOpenEx (int32_t *can_channel_no*, int32_t *mode*, const char * *password*, kvrConfigHandle * *handle*, uint32_t *profile_no*)

Open a configuration handle to the device. It should later be closed with kvrConfigClose().

This configuration handle is used both for changing the device configuration, reading status information, e.g. kvrNetworkGetConnectionStatus(), and issuing some other commands such as e.g. kvrNetworkConnectionTest() and kvrWlanStartScan().

To change the configuration, you need to open the configuration with kvrConfigMode_RW before calling kvrConfigSet().

To read the configuration, you can open the configuration with either kvrConfigMode_RW or kvrConfigMode_R before calling kvrConfigGet().

Setting a password is done through the configuration (with kvrConfigSet()). Resetting a password can be done by erasing the whole configuration with kvrConfigClear(), while first opening the configuration with kvrConfigMode_ERASE and supplying an empty password.

Before calling any other function, you must open the configuration with kvrConfigMode_R.

The profile number is used to open a specific profile. See Local Configuration.

Parameters

in	*can_channel_no*	CAN channel number
in	*mode*	Can be set to one of kvrConfigMode_xxx
in	*password*	The password as a C string. Use an empty string, i.e. "", if no password is required.
out	*handle*	A configuration handle
in	*profile_no*	Profile number

Returns

kvrOK on success or any other kvrStatus on failure.

Note

When using kvrConfigMode_RW or kvrConfigMode_ERASE, a pause of ~2 seconds is needed after kvrConfigClose(), to allow CANlib time to discard the device.

See also

kvrConfigOpen()

6.1.3.10 kvrStatus kvrConfigSet (kvrConfigHandle *handle*, const char * *xml_buffer*)

Set the device configuration. The area is erased before it is programmed. The handle must be opened kvrConfigMode_RW and closed with kvrConfigClose() afterward. If the XML input creates any errors, kvrERR_PARAMETER will be returned.

Parameters

in	*handle*	A configuration handle.
in	*xml_buffer*	A pointer to a C string containing a valid XML config.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConfig.c.

6.1.3.11 kvrStatus kvrConfigVerifyXml (const char * *xml_buffer*, char * *err_buffer*, uint32_t *err_buffer_size*)

Verify that the xml buffer complies with both the DTD and internal restrictions. If the XML input creates any errors and err_buffer is to small to hold the resulting error message, kvrERR_PARAMETER will be returned.

Parameters

in	*xml_buffer*	A pointer to a C string containing an XML configuration.
out	*err_buffer*	A pointer to a buffer that will hold any error messages.
in	*err_buffer_size*	The buffer size. Maximum size needed is 2048 bytes.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConfig.c.

6.2 Network information

Data Structures

- struct kvrAddress
- struct kvrCipherInfoElement

Typedefs

- typedef int32_t kvrRssiHistory
- typedef uint32_t kvrRttHistory

Functions

- kvrStatus kvrNetworkConnectionTest (kvrConfigHandle handle, int32_t active)
- kvrStatus kvrNetworkGetRssiRtt (kvrConfigHandle handle, kvrRssiHistory *rssi, uint32_t rssi_size, uint32_t *rssi_count, kvrRttHistory *rtt, uint32_t rtt_size, uint32_t *rtt_count)
- kvrStatus kvrWlanStartScan (kvrConfigHandle handle, int32_t active, int32_t bss_type, int32_t domain)
- kvrStatus kvrWlanGetScanResults (kvrConfigHandle handle, int32_t *rssi, int32_t *channel, kvrAddress *mac, int32_t *bss_type, char *ssid, uint32_t *capability, uint32_t *type_wpa, kvrCipherInfoElement *wpa_info, kvrCipherInfoElement *rsn_info)
- kvrStatus kvrNetworkGetHostName (kvrConfigHandle handle, char *buffer, uint32_t buffer_size)
- kvrStatus kvrHostName (uint32_t ean_hi, uint32_t ean_lo, uint32_t ser_no, char *buffer, uint32_t buffer_size)
- kvrStatus kvrNetworkGetConnectionStatus (kvrConfigHandle handle, int32_t *state, int32_t *tx_rate, int32_t *rx_rate, int32_t *channel, int32_t *rssi, int32_t *tx_power)
- kvrStatus kvrNetworkGetAddressInfo (kvrConfigHandle handle, kvrAddress *address1, kvrAddress *address2, kvrAddress *netmask, kvrAddress *gateway, int32_t *dhcp)

kvrAddressType_xxx

Type of device address.

Note

Ports are currently not used.

- #define kvrAddressType_UNKNOWN 0
- #define kvrAddressType_IPV4 1
- #define kvrAddressType_IPV6 2
- #define kvrAddressType_IPV4_PORT 3
- #define kvrAddressType_MAC 4

kvrNetworkState_xxx

States for network connection.

- #define kvrNetworkState_UNKNOWN 0
- #define kvrNetworkState_INVALID 1
- #define kvrNetworkState_STARTUP 2
- #define kvrNetworkState_INITIALIZING 3
- #define kvrNetworkState_NOT_CONNECTED 4
- #define kvrNetworkState_CONNECTION_DELAY 5
- #define kvrNetworkState_CONNECTING 6
- #define kvrNetworkState_CONNECTED 7
- #define kvrNetworkState_AUTHENTICATING 8
- #define kvrNetworkState_AUTHENTICATION_FAILED 9
- #define kvrNetworkState_ONLINE 10
- #define kvrNetworkState_FAILED_MIC 11

kvrBss_xxx

Basic Service Set.

- #define kvrBss_INFRASTRUCTURE 0
- #define kvrBss_INDEPENDENT 1
- #define kvrBss_ANY 2

kvrRegulatoryDomain_xxx

Regulatory domain.

- #define kvrRegulatoryDomain_JAPAN_TELEC 0
- #define kvrRegulatoryDomain_EUROPE_ETSI 1
- #define kvrRegulatoryDomain_NORTH_AMERICA_FCC 2
- #define kvrRegulatoryDomain_WORLD 3
- #define kvrRegulatoryDomain_CHINA_MII 4

6.2.1 Define Documentation

6.2.1.1 #define kvrAddressType_IPV4 1

IP v.4 address.

Examples:

kvrConnect.c.

6.2.1.2 #define kvrAddressType_IPV4_PORT 3

IP v.4 address with tcp-port.

6.2.1.3 #define kvrAddressType_IPV6 2

IP v.6 address.

6.2.1.4 #define kvrAddressType_MAC 4

Ethernet MAC address.

6.2.1.5 #define kvrAddressType_UNKNOWN 0

Unknown (e.g., no reply from device).

6.2.1.6 #define kvrBss_ANY 2

Any.

Examples:

kvrConfig.c.

6.2.1.7 #define kvrBss_INDEPENDENT 1

Ad-hoc network.

6.2.1.8 #define kvrBss_INFRASTRUCTURE 0

Network with AP.

6.2.1.9 #define kvrNetworkState_AUTHENTICATING 8

EAPOL handshake ongoing.

6.2.1.10 #define kvrNetworkState_AUTHENTICATION_FAILED 9

Authentication have failed.

6.2.1.11 #define kvrNetworkState_CONNECTED 7

Network is reached.

6.2.1.12 #define kvrNetworkState_CONNECTING 6

Waiting for connections (ad-hoc).

6.2.1.13 #define kvrNetworkState_CONNECTION_DELAY 5

Delay during connection (ad-hoc).

6.2.1.14 #define kvrNetworkState_FAILED_MIC 11

MIC verification (EAPOL-key) failed.

6.2.1.15 #define kvrNetworkState_INITIALIZING 3

Started, waiting for initialization.

6.2.1.16 #define kvrNetworkState_INVALID 1

Network hardware has been disabled.

6.2.1.17 #define kvrNetworkState_NOT_CONNECTED 4

No connection (may auto-connect).

6.2.1.18 #define kvrNetworkState_ONLINE 10

Authentication completed.

6.2.1.19 #define kvrNetworkState_STARTUP 2

Configuring network hardware.

6.2.1.20 #define kvrNetworkState_UNKNOWN 0

Bad state, should never be reported.

6.2.1.21 #define kvrRegulatoryDomain_CHINA_MII 4

MII

6.2.1.22 #define kvrRegulatoryDomain_EUROPE_ETSI 1

ETSI

6.2.1.23 #define kvrRegulatoryDomain_JAPAN_TELEC 0

TELEC

6.2.1.24 #define kvrRegulatoryDomain_NORTH_AMERICA_FCC 2

FCC

6.2.1.25 #define kvrRegulatoryDomain_WORLD 3

WORLD

Examples:

kvrConfig.c.

6.2.2 Typedef Documentation

6.2.2.1 typedef int32_t kvrRssiHistory

Receive Signal Strength Indicator (RSSI).

6.2.2.2 typedef uint32_t kvrRttHistory

Round-trip delay time (RTT).

6.2.3 Function Documentation

6.2.3.1 kvrStatus kvrHostName (uint32_t *ean_hi*, uint32_t *ean_lo*, uint32_t *ser_no*, char * *buffer*, uint32_t *buffer_size*)

Read the generated Hostname.

Parameters

in	*ean_hi*	The device EAN_high number.
in	*ean_lo*	The device EAN_low number.

in	*ser_no*	The device serial number.
out	*buffer*	The device Hostname as a C string.
in	*buffer_size*	The device Hostname buffer size.

Returns

kvrOK on success or any other kvrStatus on failure.

6.2.3.2 kvrStatus kvrNetworkConnectionTest (kvrConfigHandle *handle*, int32_t *active*)

Activate or deactivate connection test. When actived, the device will connect and start pinging itself to measure RTT. Use kvrNetworkGetRssiRtt() (after a while) to get the latest values.

Parameters

in	*handle*	A configuration handle.
in	*active*	Activate or deactivate connection test.

Returns

kvrOK on success or any other kvrStatus on failure.

See also

kvrNetworkGetRssiRtt()

Examples:

kvrConfig.c, and kvrNetworkConnectionTest.c.

6.2.3.3 kvrStatus kvrNetworkGetAddressInfo (kvrConfigHandle *handle*, kvrAddress * *address1*, kvrAddress * *address2*, kvrAddress * *netmask*, kvrAddress * *gateway*, int32_t * *dhcp*)

Get information about the network address settings. For a WLAN connected device, address1, netmask and gateway are IP addresses and address2 is the MAC address.

Parameters

in	*handle*	A configuration handle.
out	*address1*	The first address associated with the device.
out	*address2*	The second address associated with the device.
out	*netmask*	The netmask for the device.
out	*gateway*	The gateway for the device.
out	*dhcp*	The device ueses Dynamic Host Configuration Protocol (DHCP).

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConfig.c.

6.2.3.4 kvrStatus kvrNetworkGetConnectionStatus (kvrConfigHandle *handle*, int32_t * *state*, int32_t * *tx_rate*, int32_t * *rx_rate*, int32_t * *channel*, int32_t * *rssi*, int32_t * *tx_power*)

Get connection status information.

Parameters

in	*handle*	A configuration handle.
out	*state*	Network connection state, see kvrNetworkState_xxx.
out	*tx_rate*	Transmit rate in kbit/s.
out	*rx_rate*	Receive rate in kbit/s.
out	*channel*	Channel.
out	*rssi*	Receive Signal Strength Indicator (RSSI).
out	*tx_power*	Transmit power level in dB.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConfig.c.

6.2.3.5 kvrStatus kvrNetworkGetHostName (kvrConfigHandle *handle*, char * *buffer*, uint32_t *buffer_size*)

Read the device Hostname.

Parameters

in	*handle*	A configuration handle
out	*buffer*	The device Hostname as a C string.
in	*buffer_size*	The device Hostname buffer size.

Returns

kvrOK on success or any other kvrStatus on failure.

6.2.3.6 kvrStatus kvrNetworkGetRssiRtt (kvrConfigHandle *handle*, kvrRssiHistory * *rssi*, uint32_t *rssi_size*, uint32_t * *rssi_count*, kvrRttHistory * *rtt*, uint32_t *rtt_size*, uint32_t * *rtt_count*)

Get a history of Receive Signal Strength Indicator (RSSI) and round-trip delay time (RTT) from the connection test.

Parameters

in	*handle*	A configuration handle.
out	*rssi*	Receive Signal Strength Indicator.
in	*rssi_size*	Number of entries in *rssi*.
out	*rssi_count*	Number of RSSI elements returned.
out	*rtt*	Round-trip delay time.
in	*rtt_size*	Number of entries in *rtt*.
out	*rtt_count*	Number of RTT elements returned.

Returns

kvrOK on success or any other kvrStatus on failure.

See also

kvrNetworkConnectionTest()

Examples:

kvrConfig.c, and kvrNetworkConnectionTest.c.

6.2.3.7 kvrStatus kvrWlanGetScanResults (kvrConfigHandle *handle*, int32_t * *rssi*, int32_t * *channel*, kvrAddress * *mac*, int32_t * *bss_type*, char * *ssid*, uint32_t * *capability*, uint32_t * *type_wpa*, kvrCipherInfoElement * *wpa_info*, kvrCipherInfoElement * *rsn_info*)

Get results from WLAN scan. Call kvrWlanGetScanResults() until it returns kvrERR_BLANK to mark that no more information is available from this scan.

To convert the security information into a more human readable form, use kvrWlanGetSecurityText().

Parameters

in	*handle*	A configuration handle.
out	*rssi*	Receive Signal Strength Indicator (RSSI).
out	*channel*	WLAN Channel.
out	*mac*	Media Access Control address.
out	*bss_type*	Basic Service Set (BSS) (see kvrBss_xxx).
out	*ssid*	Service Set Identifier(SSID) as a C string. Maximum length is 32 bytes.
out	*capability*	The advertised capabilities of the BSS.

out	*type_wpa*	Only type 1 (802.1X) is supported for connection.
out	*wpa_info*	Advertised information for WPA (see kvrCipherInfoElement).
out	*rsn_info*	Advertised information for RSN (see kvrCipherInfoElement).

Returns

kvrOK on success, kvrERR_NO_ANSWER when waiting for further scan results kvrERR_BLANK when no further scan results are available or any other kvrStatus on failure.

Examples:

kvrConfig.c.

6.2.3.8 kvrStatus kvrWlanStartScan (kvrConfigHandle *handle*, int32_t *active*, int32_t *bss_type*, int32_t *domain*)

Initiate a scan for existing WLAN networks. The result is acquired with consecutive calls to kvrWlanGetScanResults(). A new scan can not be initiated until all data has been retrieved from the previous one.

Note

The device should not be connected to a network when scanning. This includes the implicit connection done by kvrNetworkConnectionTest().

Parameters

in	*handle*	A configuration handle.
in	*active*	If set, performs an active scan.
in	*bss_type*	Basic service set (BSS) selection. kvrBss_xxx.
in	*domain*	Regulatory domain. See kvrRegulatoryDomain_xxx.

Returns

kvrOK on success, kvrERR_NO_ANSWER when previous scan is ongoing or any other kvrStatus on failure.

Examples:

kvrConfig.c.

6.3 Device discovery

Data Structures

- struct kvrDeviceInfo

  Holds information about a discovered device.

Typedefs

- typedef int32_t kvrDiscoveryHandle

Functions

- kvrStatus kvrDiscoveryGetDefaultAddresses (kvrAddress address_list[ ], uint32_t address_list_size, uint32_t *address_list_count, uint32_t address_type_flags)
- kvrStatus kvrDiscoveryOpen (kvrDiscoveryHandle *handle)
- kvrStatus kvrDiscoveryClose (kvrDiscoveryHandle handle)
- kvrStatus kvrDiscoverySetAddresses (kvrDiscoveryHandle handle, const kvrAddress address_list[ ], uint32_t address_list_size)
- kvrStatus kvrDiscoveryStart (kvrDiscoveryHandle handle, uint32_t delay_ms, uint32_t timeout_ms)
- kvrStatus kvrDiscoveryGetResults (kvrDiscoveryHandle handle, kvrDeviceInfo *device_info)
- kvrStatus kvrDiscoveryStoreDevices (const kvrDeviceInfo device_info_list[ ], uint32_t device_info_list_size)
- kvrStatus kvrDiscoveryClearDevicesAtExit (BOOL onoff)
- kvrStatus kvrDiscoverySetPassword (kvrDeviceInfo *device_info, const char *password)
- kvrStatus kvrDiscoverySetEncryptionKey (kvrDeviceInfo *device_info, const char *key)
- kvrStatus kvrDeviceGetServiceStatusText (const kvrDeviceInfo *device_info, char *buffer, uint32_t buffer_size)
- kvrStatus kvrDeviceGetServiceStatus (const kvrDeviceInfo *device_info, int32_t *state, int32_t *start_info)

kvrDeviceUsage_xxx

Remote device usage status.

- #define kvrDeviceUsage_UNKNOWN 0
- #define kvrDeviceUsage_FREE 1
- #define kvrDeviceUsage_REMOTE 2
- #define kvrDeviceUsage_USB 3
- #define kvrDeviceUsage_CONFIG 4

kvrAccessibility_xxx

Remote device accessability status.

- #define kvrAccessibility_UNKNOWN 0
- #define kvrAccessibility_PUBLIC 1
- #define kvrAccessibility_PROTECTED 2
- #define kvrAccessibility_PRIVATE 3

kvrRemoteState_xxx

State of connection to device.

- #define kvrRemoteState_VOID 0
- #define kvrRemoteState_AVAILABLE 1
- #define kvrRemoteState_DISCOVERED 2
- #define kvrRemoteState_STARTING 3
- #define kvrRemoteState_STARTED 4
- #define kvrRemoteState_CONNECTION_DOWN 5
- #define kvrRemoteState_CONNECTION_UP 6
- #define kvrRemoteState_REDISCOVER 7
- #define kvrRemoteState_UNWILLING 8
- #define kvrRemoteState_REDISCOVER_PENDING 9
- #define kvrRemoteState_CLOSING 10
- #define kvrRemoteState_REMOVE_ME 11
- #define kvrRemoteState_STANDBY 12
- #define kvrRemoteState_CONFIG_CHANGED 13
- #define kvrRemoteState_STOPPING 14
- #define kvrRemoteState_INSTALLING 15

kvrAvailability_xxx

Device avalability flags.

- #define kvrAvailability_NONE 0
- #define kvrAvailability_FOUND_BY_SCAN 1
- #define kvrAvailability_STORED 2

kvrAddressTypeFlag_xxx

Flags for setting what addresses that should be returned by kvrDiscoveryGetDefaultAddresses().

- #define kvrAddressTypeFlag_ALL 0xff
- #define kvrAddressTypeFlag_BROADCAST 0x01
- #define kvrAddressTypeFlag_STORED 0x02

kvrServiceState_xxx

Current service state

- #define kvrServiceState_VOID 0
- #define kvrServiceState_AVAILABLE 1
- #define kvrServiceState_DISCOVERED 2

- #define kvrServiceState_STARTING 3
- #define kvrServiceState_STARTED 4
- #define kvrServiceState_CONNECTION_DOWN 5
- #define kvrServiceState_CONNECTION_UP 6
- #define kvrServiceState_REDISCOVER 7
- #define kvrServiceState_UNWILLING 8
- #define kvrServiceState_REDISCOVER_PENDING 9
- #define kvrServiceState_CLOSING 10
- #define kvrServiceState_REMOVE_ME 11
- #define kvrServiceState_STANDBY 12
- #define kvrServiceState_CONFIG_CHANGED 13
- #define kvrServiceState_STOPPING 14
- #define kvrServiceState_INSTALLING 15

kvrStartInfo_xxx

Current start information

- #define kvrStartInfo_NONE 0
- #define kvrStartInfo_START_OK 1
- #define kvrStartInfo_ERR_IN_USE 2
- #define kvrStartInfo_ERR_PWD 3
- #define kvrStartInfo_ERR_NOTME 4
- #define kvrStartInfo_ERR_CONFIGURING 5
- #define kvrStartInfo_ERR_PARAM 6
- #define kvrStartInfo_ERR_ENCRYPTION_PWD 7

6.3.1 Define Documentation

6.3.1.1 #define kvrAccessibility_PRIVATE 3

Private (invisible, password needed to connect).

6.3.1.2 #define kvrAccessibility_PROTECTED 2

Protected (visible for all, password needed to connect).

6.3.1.3 #define kvrAccessibility_PUBLIC 1

Public (visible for all, no password required to connect).

6.3.1.4 #define kvrAccessibility_UNKNOWN 0

Unknown (e.g., no reply from device).

6.3.1.5 #define kvrAddressTypeFlag_ALL 0xff

All defined below

Examples:

kvrConnect.c.

6.3.1.6 #define kvrAddressTypeFlag_BROADCAST 0x01

Broadcast addresses

6.3.1.7 #define kvrAddressTypeFlag_STORED 0x02

Previously stored addresses

6.3.1.8 #define kvrAvailability_FOUND_BY_SCAN 1

Device was found by scan.

Examples:

kvrConnect.c.

6.3.1.9 #define kvrAvailability_NONE 0

Manually added.

6.3.1.10 #define kvrAvailability_STORED 2

Device was stored.

Examples:

kvrConnect.c.

6.3.1.11 #define kvrDeviceUsage_CONFIG 4

Device is being configured via USB.

Examples:

kvrConnect.c.

6.3.1.12 #define kvrDeviceUsage_FREE 1

Not in use.

Examples:

kvrConnect.c.

6.3.1.13 #define kvrDeviceUsage_REMOTE 2

Connected to a PC (as a remote device).

Examples:

kvrConnect.c.

6.3.1.14 #define kvrDeviceUsage_UNKNOWN 0

Unknown (e.g., no reply from device).

Examples:

kvrConnect.c.

6.3.1.15 #define kvrDeviceUsage_USB 3

Connected via USB cable.

Examples:

kvrConnect.c.

6.3.1.16 #define kvrRemoteState_AVAILABLE 1

Tries to ping known device.

6.3.1.17 #define kvrRemoteState_CLOSING 10

Will stop communication.

6.3.1.18 #define kvrRemoteState_CONFIG_CHANGED 13

Same as UNWILLING.

6.3.1.19 #define kvrRemoteState_CONNECTION_DOWN 5

Will try and restore connection.

6.3.1.20 #define kvrRemoteState_CONNECTION_UP 6

Device connected, heartbeat up.

6.3.1.21 #define kvrRemoteState_DISCOVERED 2

Currently not used.

6.3.1.22 #define kvrRemoteState_INSTALLING 15

Driver installation is in progress.

6.3.1.23 #define kvrRemoteState_REDISCOVER 7

Trying to talk to device.

6.3.1.24 #define kvrRemoteState_REDISCOVER_PENDING 9

Will do rediscover in a moment.

6.3.1.25 #define kvrRemoteState_REMOVE_ME 11

Device removed, it will be stopped.

6.3.1.26 #define kvrRemoteState_STANDBY 12

Known device, but unused.

6.3.1.27 #define kvrRemoteState_STARTED 4

Currently not used.

6.3.1.28 #define kvrRemoteState_STARTING 3

Initializes for new device.

6.3.1.29 #define kvrRemoteState_STOPPING 14

Tries to stop device.

6.3.1.30 #define kvrRemoteState_UNWILLING 8

Device turned down connection req.

6.3.1.31 #define kvrRemoteState_VOID 0

Marked as not in list.

6.3.1.32 #define kvrServiceState_AVAILABLE 1

Device available

6.3.1.33 #define kvrServiceState_CLOSING 10

Closing

6.3.1.34 #define kvrServiceState_CONFIG_CHANGED 13

Configuration has changed

6.3.1.35 #define kvrServiceState_CONNECTION_DOWN 5

Connection is currently down

6.3.1.36 #define kvrServiceState_CONNECTION_UP 6

Connection is corrently up

6.3.1.37 #define kvrServiceState_DISCOVERED 2

Device discovered

6.3.1.38 #define kvrServiceState_INSTALLING 15

Device is currently being installed

6.3.1.39 #define kvrServiceState_REDISCOVER 7

We've lost the device - rediscover it

6.3.1.40 #define kvrServiceState_REDISCOVER_PENDING 9

Rediscover is pending

6.3.1.41 #define kvrServiceState_REMOVE_ME 11

Remove me

6.3.1.42 #define kvrServiceState_STANDBY 12

Standbe

6.3.1.43 #define kvrServiceState_STARTED 4

Device is started

6.3.1.44 #define kvrServiceState_STARTING 3

Device is starting, other devices may inhibit this device from being started at the moment (e.g. by installing).

6.3.1.45 #define kvrServiceState_STOPPING 14

Stopping

6.3.1.46 #define kvrServiceState_UNWILLING 8

Unwilling, see sub state for reason

6.3.1.47 #define kvrServiceState_VOID 0

Void

6.3.1.48 #define kvrStartInfo_ERR_CONFIGURING 5

I'm being configured so won't start

6.3.1.49 #define kvrStartInfo_ERR_ENCRYPTION_PWD 7

Wrong encryption password.

6.3.1.50 #define kvrStartInfo_ERR_IN_USE 2

Already connected to someone else

6.3.1.51 #define kvrStartInfo_ERR_NOTME 4

This start is not for me

6.3.1.52 #define kvrStartInfo_ERR_PARAM 6

Invalid parameters in QRV (non matching versions)

6.3.1.53 #define kvrStartInfo_ERR_PWD 3

Wrong connection pwd

6.3.1.54 #define kvrStartInfo_NONE 0

No information available

6.3.1.55 #define kvrStartInfo_START_OK 1

Started OK

6.3.2 Typedef Documentation

6.3.2.1 typedef int32_t kvrDiscoveryHandle

Handle used for discovery.

6.3.3 Function Documentation

6.3.3.1 kvrStatus kvrDeviceGetServiceStatus (const kvrDeviceInfo * *device_info*, int32_t * *state*, int32_t * *start_info*)

Returns local connection status of the selected device.

Parameters

in	*device_info*	The device to request the status information from.
out	*state*	The service state as a kvrServiceState_xxx
out	*start_info*	The start information as a kvrStartInfo_xxx

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConnect.c.

6.3.3.2 kvrStatus kvrDeviceGetServiceStatusText (const kvrDeviceInfo * *device_info*, char * *buffer*, uint32_t *buffer_size*)

Returns local connection status of the selected device as ASCII text.

Parameters

in	*device_info*	The device to request the status information from.
out	*buffer*	The service status as a C string.
in	*buffer_size*	The service status buffer size.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConnect.c.

6.3.3.3 kvrStatus kvrDiscoveryClearDevicesAtExit (BOOL *onoff*)

Turn automatic clearing of the stored devices on/off.

Parameters

in	*onoff*	Turn auto-clear on/off. TRUE: Stored devices will be cleared automatically when the application exits, even if the application terminates abnormally. FALSE: Stored devices will be stored until removed.

Returns

kvrOK on success or any other kvrStatus on failure.

6.3.3.4 kvrStatus kvrDiscoveryClose (kvrDiscoveryHandle *handle*)

Closes the discovery handle opened with kvrDiscoveryOpen().

Parameters

in	*handle*	Discovery handle.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConnect.c.

6.3.3.5 kvrStatus kvrDiscoveryGetDefaultAddresses (kvrAddress *address_list[]*, uint32_t *address_list_size*, uint32_t * *address_list_count*, uint32_t *address_type_flags*)

Read out the list of default broadcast addresses. If *address_type_flags* is set to kvrAddressTypeFlag_ALL the returned list will contain all found addresses (both broadcast addresses and earlier stored addresses).

Parameters

out	*address_list*	An array of addresses.
in	*address_list_size*	Number of entries in *address_list*.
out	*address_list_count*	Number of addresses returned.
in	*address_type_flags*	Which kvrAddressTypeFlag_xxx types of addresses to return

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConnect.c.

6.3.3.6 kvrStatus kvrDiscoveryGetResults (kvrDiscoveryHandle *handle*, kvrDeviceInfo * *device_info*)

Call this after calling kvrDiscoveryStart(). The first call will return the first result, second call will return the second etc. Will return found devices until kvrERR_BLANK is returned.

Parameters

in	*handle*	Discovery handle.
out	*device_info*	Device info.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConnect.c.

6.3.3.7 kvrStatus kvrDiscoveryOpen (kvrDiscoveryHandle * *handle*)

Create a handle for device discovery. Used by for instance kvrDiscoveryStart(). Close with kvrDiscoveryClose().

Parameters

out	*handle*	Discovery handle.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConnect.c.

6.3.3.8 kvrStatus kvrDiscoverySetAddresses (kvrDiscoveryHandle *handle*, const kvrAddress *address_list[]*, uint32_t *address_list_size*)

Set a list of addresses to use for discovery (overwrites any existing addresses). Setting *address_list_size* with size = 0 will cause kvrDiscoveryStart() to only return stored devices (no network traffic).

Parameters

in	*handle*	Discovery handle.
in	*address_list*	An array of addresses.
in	*address_list_size*	Number of entries in *address_list*.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConnect.c.

6.3.3.9 kvrStatus kvrDiscoverySetEncryptionKey (kvrDeviceInfo * *device_info*, const char * *key*)

Sets the encryption key to use when encrypting communication.

Parameters

in	*device_info*	The device to set the password for.
in	*key*	The key as a C string.

Returns

kvrOK on success or any other kvrStatus on failure.

6.3.3.10 kvrStatus kvrDiscoverySetPassword (kvrDeviceInfo * *device_info*, const char * *password*)

Sets the accessibility password to use when connecting to a device.

Parameters

in	*device_info*	The device to set the password for.
in	*password*	The password as a C string.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConnect.c.

6.3.3.11 kvrStatus kvrDiscoveryStart (kvrDiscoveryHandle *handle*, uint32_t *delay_ms*, uint32_t *timeout_ms*)

Start discovering devices on the addresses previously specified with kvrDiscoverySetAddresses(). A delay of *delay_ms* ms is inferred between each device address request. After the last device address is probed, one more delay of *timeout_ms* is added before returning.

This means that the function will return in (about) <address_list_size> * *delay_ms* + *timeout_ms* ms

The results can be retrieved using kvrDiscoveryGetResults(). A new call to kvrDiscoveryStart() will discard any results not yet retrieved by kvrDiscoveryGetResults().

To decide if an address is a broadcast address, the ip address and subnet mask for all availible network cards are considered.

Beside returning the devices discovered by scan, it will also return any devices previously stored with kvrDiscoveryStoreDevices().

Note

A remote device with accessibility set to "private" will not reply to a broadcast scan.

Parameters

in	*handle*	Discovery handle.
in	*delay_ms*	Delay (in ms) in between sending discovery messages to addresses in the address list.
in	*timeout_ms*	Stop waiting for device discovery after timeout_ms milliseconds.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConnect.c.

6.3.3.12 kvrStatus kvrDiscoveryStoreDevices (const kvrDeviceInfo *device_info_list[]*, uint32_t *device_info_list_size*)

Store a list of devices that can be discovered later.

Parameters

in	*device_info_list*	A list of devices to remember.
in	*device_info_list_size*	The number of elements in *device_info_list*.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConnect.c.

6.4 Helper functions

Functions

- kvrStatus kvrGetErrorText (kvrStatus error, char *buffer, uint32_t buffer_size)
- kvrStatus kvrStringFromAddress (char *buffer, uint32_t buffer_size, const kvrAddress *address)
- kvrStatus kvrAddressFromString (int32_t address_type, kvrAddress *address, const char *address_string)
- kvrStatus kvrWlanGetSecurityText (char *security_string, uint32_t security_string_size, uint32_t capability, uint32_t type_wpa, const kvrCipherInfoElement *wpa_info, const kvrCipherInfoElement *rsn_info)
- kvrStatus kvrNetworkGenerateWepKeys (const char *pass_phrase, char *key64_1, char *key64_2, char *key64_3, char *key64_4, char *key128)
- kvrStatus kvrNetworkGenerateWpaKeys (const char *pass_phrase, const char *ssid, char *key)

6.4.1 Function Documentation

6.4.1.1 kvrStatus kvrAddressFromString (int32_t *address_type*, kvrAddress * *address*, const char * *address_string*)

Convert a C string into a kvrAddress.

Examples:

- **MAC address**
  address_string: "90:E6:BA:3C:32:12"
  type: kvrAddressType_MAC
- **IP v.4**
  address_string: "192.168.1.142"
  type: kvrAddressType_IPV4
- **IP v.4 with port**
  address_string: "192.168.1.142:8080"
  type: kvrAddressType_IPV4_PORT

Parameters

in	*address_type*	kvrAddressType_xxx to convert into.
out	*address*	Returned address.
in	*address_string*	C string to convert into a kvrAddress.

Examples:

kvrConnect.c.

6.4.1.2 kvrStatus kvrGetErrorText (kvrStatus *error*, char * *buffer*, uint32_t *buffer_size*)

Convert a kvrStatus errorcode to a text.

Parameters

in	*error*	The error code to convert.
out	*buffer*	Buffer to receive error text.
in	*buffer_size*	Buffer size in bytes.

Examples:

kvrConnect.c.

6.4.1.3 kvrStatus kvrNetworkGenerateWepKeys (const char * *pass_phrase*, char * *key64_1*, char * *key64_2*, char * *key64_3*, char * *key64_4*, char * *key128*)

Generates four 64-bit and one 128-bit WEP keys.

All generated keys are returned as ASCII hexadecimal C strings.

Parameters

in	*pass_phrase*	The pass phrase as a C string.
out	*key64_1*	Generated 64-bit WEP key 1 (10 + 1 bytes).
out	*key64_2*	Generated 64-bit WEP key 2 (10 + 1 bytes).
out	*key64_3*	Generated 64-bit WEP key 3 (10 + 1 bytes).
out	*key64_4*	Generated 64-bit WEP key 4 (10 + 1 bytes).
out	*key128*	Generated 128-bit WEP key (26 + 1 bytes).

Returns

kvrOK on success or any other kvrStatus on failure.

6.4.1.4 kvrStatus kvrNetworkGenerateWpaKeys (const char * *pass_phrase*, const char * *ssid*, char * *key*)

Generates a WPA key.

Parameters

in	*pass_phrase*	The pass phrase as a C string.
in	*ssid*	SSID as a C string. Maximum length is 32 bytes.
out	*key*	The WPA key, 256 bits as a an ASCII hexadecimal C string (64 + 1 bytes).

Returns

kvrOK on success or any other kvrStatus on failure.

6.4.1.5 kvrStatus kvrStringFromAddress (char * *buffer*, uint32_t *buffer_size*, const kvrAddress * *address*)

Convert a kvrAddress to a text string. The output format depends on the kvrAddressType_xxx. *buffer_size* must be the maximum lenth for each type. i.e. kvrAddressType_IPV4 must have lenght at least 16.

Parameters

out	*buffer*	The converted string as a C string.
in	*buffer_size*	Size of buffer.
in	*address*	The address to convert.

Examples:

kvrConnect.c.

6.4.1.6 kvrStatus kvrWlanGetSecurityText (char * *security_string*, uint32_t *security_string_size*, uint32_t *capability*, uint32_t *type_wpa*, const kvrCipherInfoElement * *wpa_info*, const kvrCipherInfoElement * *rsn_info*)

Returns a C string in human readable format from the security information gathered with kvrWlanGetScanResults()

Example output: "WPA2: G-TKIP (PSK) P1-AES P2-TKIP WPA: G-TKIP (PSK) P1-AES P2-TKIP"

The above example shows a network using Pre-Shared Key with both WPA2 and WPA enabled, for both modes, a group key using TKIP is needed, and in both modes you can choose between AES (CCMP) and TKIP as the cipher for the pairwise key.

The length of the generated string could theoretically be up to about 180 characters. If the length of the supplied security_string is too short, the result will be truncated and the function will return kvrERR_PARAMETER.

Parameters

out	*security_string*	A C string.
in	*security_string_size*	Max size of security_string.
in	*capability*	The advertised capabilities of the BSS.
in	*type_wpa*	Authentication suite type.
in	*wpa_info*	Advertised information for WPA.
in	*rsn_info*	Advertised information for RSN.

Returns

kvrOK on success or any other kvrStatus on failure.

Examples:

kvrConfig.c.

Chapter 7

Data Structure Documentation

7.1 kvrAddress Struct Reference

`#include <kvrlib.h>`

Data Fields

- uint32_t type
- uint8_t address [20]

7.1.1 Detailed Description

Device address.

Examples:

kvrConfig.c, and kvrConnect.c.

7.1.2 Field Documentation

7.1.2.1 uint8_t address[20]

IP or MAC address.

Examples:

kvrConfig.c.

7.1.2.2 uint32_t type

kvrAddressType_xxx.

The documentation for this struct was generated from the following file:

- kvrlib.h

7.2 kvrCipherInfoElement Struct Reference

`#include <kvrlib.h>`

Data Fields

- uint32_t version
- uint32_t capability
- uint32_t group_cipher
- uint32_t list_cipher_auth

7.2.1 Detailed Description

Capability of a WLAN AP. These are values from the standard 802.11 beacon.

To convert the security information into a more human readable form, use kvrWlanGetSecurityText().

Examples:

kvrConfig.c.

7.2.2 Field Documentation

7.2.2.1 uint32_t capability

Advertised capabilities. capability[5] Privacy flag 1: WEP 0: Open

7.2.2.2 uint32_t group_cipher

0xff: No WPA/RSN. 0x02: TKIP 0x04: CCMP

7.2.2.3 uint32_t list_cipher_auth

8 nybbles (In each nybble: pos 0: cipher where; 0x2: TKIP 0x4: CCMP pos 1: authentication where; 0xa: PSK 0x9: RADIUS Unused nybbles are set to 0xf.

7.2.2.4 uint32_t version

The documentation for this struct was generated from the following file:

- kvrlib.h

7.3 kvrDeviceInfo Struct Reference

Holds information about a discovered device.

`#include <kvrlib.h>`

Data Fields

- uint32_t struct_size
- uint32_t ean_hi
- uint32_t ean_lo
- uint32_t ser_no
- int32_t fw_major_ver
- int32_t fw_minor_ver
- int32_t fw_build_ver
- char name [256]
- char host_name [256]
- int32_t usage
- int32_t accessibility
- char accessibility_pwd [256]
- kvrAddress device_address
- kvrAddress client_address
- kvrAddress base_station_id
- int32_t request_connection
- int32_t availability
- char encryption_key [32]
- char reserved1 [256]
- char reserved2 [256]

7.3.1 Detailed Description

Holds information about a discovered device. The information in here is returned when a device is discovered. For a WLAN connected device, device_address and client_address are IP addresses, and base_station_id is the ethernet MAC address of the AP.

Depending on the "availability" flag, not all fields may be used. If "availability" includes the flag kvrAvailability_STORED the following fields are set: ean_hi, ean_lo, ser_no, device_address, request_connection, name and accessibility_pwd.

If the field "availability" includes the flag kvrAvailability_FOUND_BY_SCAN, the following fields are set: fw_major_ver, fw_minor_ver, fw_build_ver, usage, accessibility, host_name and client_address.

Examples:

kvrConnect.c.

7.3.2 Field Documentation

7.3.2.1 int32_t accessibility

kvrAccessibility_xxx.

7.3.2.2 char accessibility_pwd[256]

Accessibility password or empty.

Examples:

kvrConnect.c.

7.3.2.3 int32_t availability

The device availability. One or more kvrAvailability_xxx flags.

Examples:

kvrConnect.c.

7.3.2.4 kvrAddress base_station_id

Unique ID of base station, if any.

7.3.2.5 kvrAddress client_address

Address of connected client, if any.

Examples:

kvrConnect.c.

7.3.2.6 kvrAddress device_address

Address of remote device.

Examples:

kvrConnect.c.

7.3.2.7 uint32_t ean_hi

High part of EAN.

Examples:

kvrConnect.c.

7.3.2.8 uint32_t ean_lo

Low part of EAN.

Examples:

kvrConnect.c.

7.3.2.9 char encryption_key[32]

Encryption key.

7.3.2.10 int32_t fw_build_ver

Firmware build version.

Examples:

kvrConnect.c.

7.3.2.11 int32_t fw_major_ver

Major firmware version.

Examples:

kvrConnect.c.

7.3.2.12 int32_t fw_minor_ver

Minor firmware version.

Examples:

kvrConnect.c.

7.3.2.13 char host_name[256]

DNS hostname or empty.

Examples:

kvrConnect.c.

7.3.2.14 char name[256]

User-defined name.

Examples:

kvrConnect.c.

7.3.2.15 int32_t request_connection

Activate or deactivate a request for connection to a specified device. Activation means that the device will be connected to when it appears in the future.

7.3.2.16 char reserved1[256]

7.3.2.17 char reserved2[256]

7.3.2.18 uint32_t ser_no

Serial number.

Examples:

kvrConnect.c.

7.3.2.19 uint32_t struct_size

Size of struct, used for compatibility.

7.3.2.20 int32_t usage

kvrDeviceUsage_xxx.

Examples:

kvrConnect.c.

The documentation for this struct was generated from the following file:

- kvrlib.h

Chapter 8

File Documentation

8.1 kvrlib.h File Reference

```
#include <windows.h>
#include <kvaser_stdint.h>
```

Data Structures

- struct kvrAddress
- struct kvrCipherInfoElement
- struct kvrDeviceInfo

  Holds information about a discovered device.

Defines

kvrConfigMode_xxx

Configuration mode.

- #define kvrConfigMode_R 0
- #define kvrConfigMode_RW 1
- #define kvrConfigMode_ERASE 2

kvrAddressType_xxx

Type of device address.

Note

Ports are currently not used.

- #define kvrAddressType_UNKNOWN 0
- #define kvrAddressType_IPV4 1

- #define kvrAddressType_IPV6 2
- #define kvrAddressType_IPV4_PORT 3
- #define kvrAddressType_MAC 4

kvrNetworkState_xxx

States for network connection.

- #define kvrNetworkState_UNKNOWN 0
- #define kvrNetworkState_INVALID 1
- #define kvrNetworkState_STARTUP 2
- #define kvrNetworkState_INITIALIZING 3
- #define kvrNetworkState_NOT_CONNECTED 4
- #define kvrNetworkState_CONNECTION_DELAY 5
- #define kvrNetworkState_CONNECTING 6
- #define kvrNetworkState_CONNECTED 7
- #define kvrNetworkState_AUTHENTICATING 8
- #define kvrNetworkState_AUTHENTICATION_FAILED 9
- #define kvrNetworkState_ONLINE 10
- #define kvrNetworkState_FAILED_MIC 11

kvrBss_xxx

Basic Service Set.

- #define kvrBss_INFRASTRUCTURE 0
- #define kvrBss_INDEPENDENT 1
- #define kvrBss_ANY 2

kvrRegulatoryDomain_xxx

Regulatory domain.

- #define kvrRegulatoryDomain_JAPAN_TELEC 0
- #define kvrRegulatoryDomain_EUROPE_ETSI 1
- #define kvrRegulatoryDomain_NORTH_AMERICA_FCC 2
- #define kvrRegulatoryDomain_WORLD 3
- #define kvrRegulatoryDomain_CHINA_MII 4

kvrDeviceUsage_xxx

Remote device usage status.

- #define kvrDeviceUsage_UNKNOWN 0
- #define kvrDeviceUsage_FREE 1
- #define kvrDeviceUsage_REMOTE 2
- #define kvrDeviceUsage_USB 3
- #define kvrDeviceUsage_CONFIG 4

kvrAccessibility_xxx

Remote device accessability status.

- #define kvrAccessibility_UNKNOWN 0
- #define kvrAccessibility_PUBLIC 1
- #define kvrAccessibility_PROTECTED 2

- #define kvrAccessibility_PRIVATE 3

kvrRemoteState_xxx

State of connection to device.

- #define kvrRemoteState_VOID 0
- #define kvrRemoteState_AVAILABLE 1
- #define kvrRemoteState_DISCOVERED 2
- #define kvrRemoteState_STARTING 3
- #define kvrRemoteState_STARTED 4
- #define kvrRemoteState_CONNECTION_DOWN 5
- #define kvrRemoteState_CONNECTION_UP 6
- #define kvrRemoteState_REDISCOVER 7
- #define kvrRemoteState_UNWILLING 8
- #define kvrRemoteState_REDISCOVER_PENDING 9
- #define kvrRemoteState_CLOSING 10
- #define kvrRemoteState_REMOVE_ME 11
- #define kvrRemoteState_STANDBY 12
- #define kvrRemoteState_CONFIG_CHANGED 13
- #define kvrRemoteState_STOPPING 14
- #define kvrRemoteState_INSTALLING 15

kvrAvailability_xxx

Device avalability flags.

- #define kvrAvailability_NONE 0
- #define kvrAvailability_FOUND_BY_SCAN 1
- #define kvrAvailability_STORED 2

kvrAddressTypeFlag_xxx

Flags for setting what addresses that should be returned by kvrDiscoveryGetDefaultAddresses().

- #define kvrAddressTypeFlag_ALL 0xff
- #define kvrAddressTypeFlag_BROADCAST 0x01
- #define kvrAddressTypeFlag_STORED 0x02

kvrServiceState_xxx

Current service state

- #define kvrServiceState_VOID 0
- #define kvrServiceState_AVAILABLE 1
- #define kvrServiceState_DISCOVERED 2
- #define kvrServiceState_STARTING 3
- #define kvrServiceState_STARTED 4
- #define kvrServiceState_CONNECTION_DOWN 5
- #define kvrServiceState_CONNECTION_UP 6
- #define kvrServiceState_REDISCOVER 7
- #define kvrServiceState_UNWILLING 8
- #define kvrServiceState_REDISCOVER_PENDING 9
- #define kvrServiceState_CLOSING 10
- #define kvrServiceState_REMOVE_ME 11

- #define kvrServiceState_STANDBY 12
- #define kvrServiceState_CONFIG_CHANGED 13
- #define kvrServiceState_STOPPING 14
- #define kvrServiceState_INSTALLING 15

kvrStartInfo_xxx

Current start information

- #define kvrStartInfo_NONE 0
- #define kvrStartInfo_START_OK 1
- #define kvrStartInfo_ERR_IN_USE 2
- #define kvrStartInfo_ERR_PWD 3
- #define kvrStartInfo_ERR_NOTME 4
- #define kvrStartInfo_ERR_CONFIGURING 5
- #define kvrStartInfo_ERR_PARAM 6
- #define kvrStartInfo_ERR_ENCRYPTION_PWD 7

Typedefs

- typedef int32_t kvrConfigHandle
- typedef int32_t kvrRssiHistory
- typedef uint32_t kvrRttHistory
- typedef int32_t kvrDiscoveryHandle

Enumerations

- enum kvrStatus {
  kvrOK = 0,
  kvrERR_NOT_INITIALIZED = -1,
  kvrERR_GENERIC = -2,
  kvrERR_CHECKSUM = -3,
  kvrERR_PARAMETER = -4,
  kvrERR_PASSWORD = -5,
  kvrERR_BLANK = -6,
  kvrERR_NO_DEVICE = -7,
  kvrERR_NO_ANSWER = -8,
  kvrERR_NOT_IMPLEMENTED = -9,
  kvrERR_PERMISSION_DENIED = -10,
  kvrERR_OUT_OF_SPACE = -11,
  kvrERR_NO_SERVICE = -12,
  kvrERR_DUPLICATED_DEVICE = -13 }

Functions

- kvrStatus kvrConfigOpen (int32_t can_channel_no, int32_t mode, const char *password, kvrConfigHandle *handle)
- kvrStatus kvrConfigActiveProfileSet (int32_t can_channel_no, int32_t profile_number)
- kvrStatus kvrConfigActiveProfileGet (int32_t can_channel_no, int32_t *profile_number)
- kvrStatus kvrConfigNoProfilesGet (int32_t can_channel_no, int32_t *no_profiles)
- kvrStatus kvrConfigOpenEx (int32_t can_channel_no, int32_t mode, const char *password, kvrConfigHandle *handle, uint32_t profile_no)
- void kvrConfigClose (kvrConfigHandle handle)
- kvrStatus kvrConfigVerifyXml (const char *xml_buffer, char *err_buffer, uint32_t err_buffer_size)
- kvrStatus kvrConfigSet (kvrConfigHandle handle, const char *xml_buffer)
- kvrStatus kvrConfigGet (kvrConfigHandle handle, char *xml_buffer, uint32_t xml_buffer_size)
- kvrStatus kvrConfigInfoGet (int32_t can_channel_no, int32_t profile_no, char *xml_buffer, uint32_t xml_buffer_size)
- kvrStatus kvrConfigClear (kvrConfigHandle handle)
- kvrStatus kvrNetworkConnectionTest (kvrConfigHandle handle, int32_t active)
- kvrStatus kvrNetworkGetRssiRtt (kvrConfigHandle handle, kvrRssiHistory *rssi, uint32_t rssi_size, uint32_t *rssi_count, kvrRttHistory *rtt, uint32_t rtt_size, uint32_t *rtt_count)
- kvrStatus kvrWlanStartScan (kvrConfigHandle handle, int32_t active, int32_t bss_type, int32_t domain)
- kvrStatus kvrWlanGetScanResults (kvrConfigHandle handle, int32_t *rssi, int32_t *channel, kvrAddress *mac, int32_t *bss_type, char *ssid, uint32_t *capability, uint32_t *type_wpa, kvrCipherInfoElement *wpa_info, kvrCipherInfoElement *rsn_info)
- kvrStatus kvrNetworkGetHostName (kvrConfigHandle handle, char *buffer, uint32_t buffer_size)
- kvrStatus kvrHostName (uint32_t ean_hi, uint32_t ean_lo, uint32_t ser_no, char *buffer, uint32_t buffer_size)
- kvrStatus kvrNetworkGetConnectionStatus (kvrConfigHandle handle, int32_t *state, int32_t *tx_rate, int32_t *rx_rate, int32_t *channel, int32_t *rssi, int32_t *tx_power)
- kvrStatus kvrNetworkGetAddressInfo (kvrConfigHandle handle, kvrAddress *address1, kvrAddress *address2, kvrAddress *netmask, kvrAddress *gateway, int32_t *dhcp)
- kvrStatus kvrDiscoveryGetDefaultAddresses (kvrAddress address_list[ ], uint32_t address_list_size, uint32_t *address_list_count, uint32_t address_type_flags)
- kvrStatus kvrDiscoveryOpen (kvrDiscoveryHandle *handle)
- kvrStatus kvrDiscoveryClose (kvrDiscoveryHandle handle)
- kvrStatus kvrDiscoverySetAddresses (kvrDiscoveryHandle handle, const kvrAddress address_list[ ], uint32_t address_list_size)
- kvrStatus kvrDiscoveryStart (kvrDiscoveryHandle handle, uint32_t delay_ms, uint32_t timeout_ms)

- kvrStatus kvrDiscoveryGetResults (kvrDiscoveryHandle handle, kvrDeviceInfo *device_info)
- kvrStatus kvrDiscoveryStoreDevices (const kvrDeviceInfo device_info_list[ ], uint32_t device_info_list_size)
- kvrStatus kvrDiscoveryClearDevicesAtExit (BOOL onoff)
- kvrStatus kvrDiscoverySetPassword (kvrDeviceInfo *device_info, const char *password)
- kvrStatus kvrDiscoverySetEncryptionKey (kvrDeviceInfo *device_info, const char *key)
- kvrStatus kvrDeviceGetServiceStatusText (const kvrDeviceInfo *device_info, char *buffer, uint32_t buffer_size)
- kvrStatus kvrDeviceGetServiceStatus (const kvrDeviceInfo *device_info, int32_t *state, int32_t *start_info)
- kvrStatus kvrGetErrorText (kvrStatus error, char *buffer, uint32_t buffer_size)
- kvrStatus kvrStringFromAddress (char *buffer, uint32_t buffer_size, const kvrAddress *address)
- kvrStatus kvrAddressFromString (int32_t address_type, kvrAddress *address, const char *address_string)
- kvrStatus kvrWlanGetSecurityText (char *security_string, uint32_t security_string_size, uint32_t capability, uint32_t type_wpa, const kvrCipherInfoElement *wpa_info, const kvrCipherInfoElement *rsn_info)
- kvrStatus kvrNetworkGenerateWepKeys (const char *pass_phrase, char *key64_1, char *key64_2, char *key64_3, char *key64_4, char *key128)
- kvrStatus kvrNetworkGenerateWpaKeys (const char *pass_phrase, const char *ssid, char *key)
- void kvrInitializeLibrary (void)
- void kvrUnloadLibrary (void)
- kvrStatus kvrServiceQuery (int *status)
- kvrStatus kvrServiceStart (int *status)
- kvrStatus kvrServiceStop (int *status)

8.1.1 Detailed Description

8.1.2 LICENSE

WWW: http://www.kvaser.com

8.1.3 DESCRIPTION

THIS IS A PRELIMINARY VERSION AND SUBJECT TO CHANGE.

Proposed new remote device API.

Version

PRELIMINARY

Author

Kvaser AB

8.1.4 Enumeration Type Documentation

8.1.4.1 enum kvrStatus

Return type of kvrlib functions.

Enumerator:

kvrOK OK!

kvrERR_NOT_INITIALIZED kvrlib has not been initialized.

kvrERR_GENERIC Generic error.

kvrERR_CHECKSUM Checksum problem.

kvrERR_PARAMETER Error in supplied in parameters.

kvrERR_PASSWORD Supplied password was wrong.

kvrERR_BLANK List was not set or no more results.

kvrERR_NO_DEVICE Remote device is unreachable.

kvrERR_NO_ANSWER No answer arrived within given timeout.

kvrERR_NOT_IMPLEMENTED Function is not yet implemented.

kvrERR_PERMISSION_DENIED Permission denied.

kvrERR_OUT_OF_SPACE Out of space, eg. to many open handles, to small buffer.

kvrERR_NO_SERVICE The helper service is not running.

kvrERR_DUPLICATED_DEVICE There are duplicates in the device list.

8.1.5 Function Documentation

8.1.5.1 void kvrInitializeLibrary (void)

Initializes library stuff. Call this function before calling any other kvr function.

Examples:

kvrConfig.c, kvrConnect.c, and kvrNetworkConnectionTest.c.

8.1.5.2 kvrStatus kvrServiceQuery (int * *status*)

Queries the status of the helper service. The helper service is installed as a part of the driver package and is normally set to automatic start.

Note

This API call requires read access to the service.

Parameters

out	*status*	Win32 status code on failure.

Returns

kvrOK on success (meaning that the service is running) or any other kvrStatus on failure.

8.1.5.3 kvrStatus kvrServiceStart (int * *status*)

Starts the helper service. The helper service is installed as a part of the driver package and is normally set to automatic start.

Note

This API call requires control access to the service.

Parameters

out	*status*	Win32 status code on failure.

Returns

kvrOK on success (meaning that the service is started or already is running) or any other kvrStatus on failure.

8.1.5.4 kvrStatus kvrServiceStop (int * *status*)

Stops the helper service. The helper service is installed as a part of the driver package and is normally set to automatic start.

Note

This API call requires control access to the service.

Parameters

out	*status*	Win32 status code on failure.

Returns

kvrOK on success (meaning that the service is stopped or already is stopped) or any other kvrStatus on failure.

8.1.5.5 void kvrUnloadLibrary (void)

Unloads library stuff. Call this function after calling all other kvr functions.

Examples:

kvrConfig.c, and kvrConnect.c.

Chapter 9

Example Documentation

9.1 kvrConfig.c

```
/*
 * This examples shows how to configure a device
 */
#include <stdio.h>
#include "canlib.h"
#include "kvrlib.h"


//------------------------------------------------------------------------------
// List all connected devices
//------------------------------------------------------------------------------
void listDevices (void)
{
  int i;
  canStatus status = canOK;
  char name[100];
  int channel_count = 0;

  canGetNumberOfChannels(&channel_count);

  printf("First argument must be a channel!\n\n");
  printf("Channel\t Name\n");
  printf("-------------------------------------------------------\n");
  for (i = 0; (status == canOK) && (i < channel_count); i++) {
    name[0] = '\0';
    status = canGetChannelData(i, canCHANNELDATA_CHANNEL_NAME, name, sizeof(name)
      );
    printf("%d\t %s\n", i, name);
  }
}

//------------------------------------------------------------------------------
// Can we configure the device on this channel without password?
//------------------------------------------------------------------------------
int isPasswordFree (unsigned int channelNumber)
{
  kvrStatus       status;
  kvrConfigHandle handle;
```

```
  status = kvrConfigOpen(channelNumber, kvrConfigMode_R, "", &handle);
  if (status != kvrOK) {
    printf("Failed to open configuration with empty password on channel %d\n", ch
      annelNumber);
    return 0;
  } else {
    kvrConfigClose(handle);
    return 1;
  }
}

//--------------------------------------------------------------------------
// Wait until device appears (or timout occurs)
// Returns 0 if successful
//--------------------------------------------------------------------------
int waitForDevice(unsigned int ean_hi, unsigned int ean_lo, unsigned int serial,

                  int timeout_in_ms)
{
  unsigned long time_start;
  canStatus stat;

  printf("\nWaiting for device with EAN %08x%08x, and serial %d\n", ean_hi, ean_l
      o, serial);

  time_start = GetTickCount();
  Sleep(2000);

  do {
    int channel_count;
    DWORD tmp_serial[2];
    unsigned long tmp_ean[2];
    int channel;

    stat = canGetNumberOfChannels(&channel_count);
    if (stat != canOK) {
      printf("canGetNumberOfChannels() failed.\n");
      return -1;
    }

    for (channel=0; channel<channel_count; channel++) {
      stat = canGetChannelData(channel, canCHANNELDATA_CARD_UPC_NO, tmp_ean, size
      of(tmp_ean));
      if (stat != canOK) {
        printf("canGetChannelData(canCHANNELDATA_CARD_UPC_NO) failed.\n");
        return -2;
      }

      stat = canGetChannelData(channel, canCHANNELDATA_CARD_SERIAL_NO, tmp_serial
      , sizeof(tmp_serial));
      if (stat != canOK) {
        printf("canGetChannelData(canCHANNELDATA_CARD_SERIAL_NO) failed.\n");
        return -3;
      }

      //printf("%08x%08x %d\n", tmp_ean[1], tmp_ean[0], tmp_serial[0]);
      if (ean_hi == tmp_ean[1] && ean_lo == tmp_ean[0] && serial == tmp_serial[0]
      ) {
        printf("Found!\n\n");
        return 0; // Found!
      }
    }
```

```
    printf("Try again...\n");
    Sleep(500);
  } while (GetTickCount() < (time_start + timeout_in_ms));

  printf("Device did not appear within given timeout\n\n");
  return -4;
}

//--------------------------------------------------------------------------
// Can we configure the device on this channel (i.e. no-one else is using it)?
//--------------------------------------------------------------------------
int isAvailibleForConfig (unsigned int canlib_channel, const char *password)
{
  int             can_hnd;
  canStatus       stat;
  DWORD           bus_type;
  DWORD           serial[2];
  DWORD           tmp_serial[2];
  unsigned long   ean[2];
  unsigned long   tmp_ean[2];
  int             chan_no;
  int             tmp_chan;
  kvrConfigHandle cfg_hnd;

  can_hnd = canOpenChannel(canlib_channel, canOPEN_EXCLUSIVE);
  if (can_hnd < 0) {
    printf("Channel %d can not be opened exclusively.\n", canlib_channel);
    return 0;
  }


  stat = canIoCtl(can_hnd, canIOCTL_GET_BUS_TYPE, &bus_type, sizeof(bus_type));
  if (stat) {
    printf("ERROR: failed to get bustype %d\n", stat);
    canClose(can_hnd);
    return 0;
  }

  if (bus_type != kvBUSTYPE_GROUP_LOCAL) {
    printf("Channel is not local (bus type:%d).\n", bus_type);
    canClose(can_hnd);
    return 0;
  }
  canClose(can_hnd);


  //
  // check all channels on a given device
  //

  stat = canGetChannelData(canlib_channel, canCHANNELDATA_CARD_UPC_NO, ean, sizeo
      f(ean));
  stat = canGetChannelData(canlib_channel, canCHANNELDATA_CARD_SERIAL_NO, serial,
       sizeof(serial));
  stat = canGetChannelData(canlib_channel, canCHANNELDATA_CHAN_NO_ON_CARD, &chan_
      no, sizeof(chan_no));

  tmp_chan = canlib_channel - chan_no;
  while (1) {

    can_hnd = canOpenChannel(tmp_chan, canOPEN_EXCLUSIVE);
```

```
    if (can_hnd < 0) {
      printf("Channel %d (same device as channel %d) can not be opened exclusivel
      y.\n", tmp_chan, canlib_channel);
      return 0;
    }
    canClose(can_hnd);

    stat = canGetChannelData(++tmp_chan, canCHANNELDATA_CARD_UPC_NO, tmp_ean, siz
      eof(tmp_ean));
    stat = canGetChannelData(tmp_chan, canCHANNELDATA_CARD_SERIAL_NO, tmp_serial,
       sizeof(tmp_serial));
    if (tmp_ean[0] != ean[0] || tmp_ean[1] != ean[1] ||
        tmp_serial[0] != serial[0] || tmp_serial[1] != serial[1]) {
      break;
    }
  }

  canClose(can_hnd);

  if (isPasswordFree(canlib_channel)) {
    printf("No password is needed for configuring channel %d\n", canlib_channel);


    // This test will remove the device from Kvaser Hardware
    stat = kvrConfigOpen(canlib_channel, kvrConfigMode_RW, "", &cfg_hnd);
    if (stat != kvrOK) {
      printf("Failed to open configuration with empty password on channel %d\n",
      canlib_channel);
      return 0;
    } else {
      kvrConfigClose(cfg_hnd);
      // Wait for the device to reappear in Kvaser Hardware
      return 0 == waitForDevice(ean[1], ean[0], serial[0], 10000); // 10s timeout
      ;
    }
  } else {
    printf("Password is needed for configuring channel %d\n", canlib_channel);

    // This test will remove the device from Kvaser Hardware
    stat = kvrConfigOpen(canlib_channel, kvrConfigMode_RW, password, &cfg_hnd);
    if (stat != kvrOK) {
      printf("Failed to open configuration with supplied password '%s' on channel
       %d\n", password, canlib_channel);
      return 0;
    } else {
      kvrConfigClose(cfg_hnd);
      // Wait for the device to reappear in Kvaser Hardware
      return 0 == waitForDevice(ean[1], ean[0], serial[0], 10000); // 10s timeout
      ;
    }
  }

  return 1;
}

//------------------------------------------------------------------------------
// Scan for available networks and print some information about them
//------------------------------------------------------------------------------
kvrStatus doScanNetworks (kvrConfigHandle handle)
{
  kvrStatus status;
  kvrStatus stat;
```

```
  int32_t active   = 0;          // is a passive scan
  int32_t bss_type = kvrBss_ANY; // infrastructure and adhoc
  int32_t domain   = kvrRegulatoryDomain_WORLD;

  char ssid[40];
  char securityString[200];
  int32_t rssi;
  int32_t channel;
  kvrAddress mac;
  uint32_t capability;
  uint32_t type_wpa;
  kvrCipherInfoElement wpa_info;
  kvrCipherInfoElement rsn_info;

  status = kvrWlanStartScan(handle, active, bss_type, domain);
  if (status != kvrOK) {
    printf("Could not start scan (%d)\n", status);
    return status;
  }

  do {
    status = kvrWlanGetScanResults(handle, &rssi, &channel, &mac, &bss_type,
                                   ssid, &capability, &type_wpa,
                                   &wpa_info, &rsn_info );
    if (status == kvrOK) {
      printf("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\n");
      printf("SSID: %s\n", ssid);
      printf("RSSI: %d dBm\n", rssi);
      printf("WLAN Channel: %d\n", channel);
      printf("MAC address: %02X %02X %02X %02X %02X %02X\n",
             mac.address[0], mac.address[1], mac.address[2],
             mac.address[3], mac.address[4], mac.address[5] );
      printf("BSS type: %d\n", bss_type);
      printf("Capabilities: 0x%04x\n", capability);

      // Convert to string
      stat = kvrWlanGetSecurityText(securityString, sizeof(securityString),
                                    capability, type_wpa, &wpa_info, &rsn_info);
      printf("Security");
      if (stat == kvrERR_PARAMETER) {
        printf("(truncated)");
      }
      printf(": %s\n", securityString);
    }
  } while ((status == kvrOK) || (status == kvrERR_NO_ANSWER));

  // kvrERR_BLANK => no more networks => OK
  return (status == kvrERR_BLANK ? kvrOK : status);
}


//------------------------------------------------------------------------------
// Configure a device
//------------------------------------------------------------------------------
kvrStatus doConfigure (kvrConfigHandle handle)
{
  kvrStatus status;
  char new_xml_config[4096];
  char old_xml_config[4096];
  char xml_error[2048];
```

```
  // Save the old configuration
  status = kvrConfigGet(handle, old_xml_config, sizeof(old_xml_config));
  if (status != kvrOK) {
    printf("Could not read configuration from device (%d)\n", status);
    kvrConfigClose(handle);
    return status;
  }
  printf("Old configuration: %s\n", old_xml_config);

  // Adjust settings in XML file based on data from doScanNetworks()
  // ...
  memcpy(new_xml_config, old_xml_config, sizeof(old_xml_config));

  // Check that the new configuration is valid
  memset(xml_error, 0, sizeof(xml_error));
  status = kvrConfigVerifyXml(new_xml_config, xml_error, sizeof(xml_error));
  if (status != kvrOK) {
    printf("The XML configuration is not valid (%d):\n%s\n",
    status, xml_error);
    kvrConfigClose(handle);
    return status;
  }

  // Download new configuration
  status = kvrConfigSet(handle, old_xml_config);
  if (status != kvrOK) {
    printf("Could not write configuration to device (%d)\n", status);
    return status;
  }

  return status;
}


//------------------------------------------------------------------------------
// Try configuration
//------------------------------------------------------------------------------
kvrStatus doTryConfiguration (kvrConfigHandle handle, int seconds)
{
  kvrStatus status;

  kvrAddress address;
  kvrAddress mac;
  kvrAddress netmask;
  kvrAddress gateway;
  int32_t dhcp;

  int32_t state;
  int32_t tx_rate;
  int32_t rx_rate;
  int32_t channel;
  int32_t rssi_mean;
  int32_t tx_power;

  kvrRssiHistory rssi[14] = {0};
  kvrRttHistory rtt[14] = {0};
  int rtt_len = sizeof(rtt) / sizeof(kvrRttHistory);
  int rssi_len = sizeof(rssi) / sizeof(kvrRssiHistory);
  int rtt_actual;
  int rssi_actual;

  // connection test. 1 = activate ping
```

```
status = kvrNetworkConnectionTest(handle, 1);
if (status != kvrOK) {
  printf("Could not start ping(%d)\n", status);
  return status;
}

do {
  Sleep(1000);
  // Ask for RSSI and RTT so that we get updated
  // values when calling kvrNetworkGetConnectionStatus()
  status = kvrNetworkGetRssiRtt(handle, rssi, rssi_len, &rssi_actual,
                                rtt, rtt_len, &rtt_actual);
  if (status != kvrOK) {
    printf("Could not get RSSI / RTT (%d)\n", status);
    break;
  }

  status = kvrNetworkGetConnectionStatus(handle, &state, &tx_rate, &rx_rate,
                                         &channel, &rssi_mean, &tx_power);
  if (status != kvrOK) {
    printf("Could not get status (%d)\n", status);
    break;
  }

  printf("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\n");
  printf("Connection state: %d\n", state);
  printf("Transmit rate: %d kbit/s\n", tx_rate);
  printf("Receive rate: %d kbit/s\n", rx_rate);
  printf("Channel: %d\n", channel);
  printf("Receive Signal Strength Indicator: %d dBm\n", rssi_mean);
  printf("Transmit power level: %d dB\n", tx_power);
} while (--seconds > 0);

// connection test. 0 = deactivate ping
status = kvrNetworkConnectionTest(handle, 0);
if (status != kvrOK) {
  printf("Could not stop ping(%d)\n", status);
  return status;
}

status = kvrNetworkGetAddressInfo(handle, &address, &mac, &netmask, &gateway, &
    dhcp);
if (status != kvrOK) {
  printf("Could not get IP info(%d)\n", status);
  return status;
}

// Assume IP v.4, i.e. address/netmask/gateway.type is kvrAddressType_IPV4
printf("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\n");
printf("DHCP: %s\n", dhcp ? "ON" : "OFF");
printf("MAC address: %02X %02X %02X %02X %02X %02X\n",
       mac.address[0], mac.address[1], mac.address[2],
       mac.address[3], mac.address[4], mac.address[5] );
printf("IP Address: %d.%d.%d.%d\n", address.address[0], address.address[1],
  address.address[2], address.address[3]);
printf("Netmask: %d.%d.%d.%d\n", netmask.address[0], netmask.address[1],
  netmask.address[2], netmask.address[3]);
printf("Gateway: %d.%d.%d.%d\n", gateway.address[0], gateway.address[1],
  gateway.address[2], gateway.address[3]);


return status;
```

```
}


//------------------------------------------------------------------------------
// Scan for available networks, configure a local (USB) device, and use the new
// configuration to establish a connection to WLAN.
//------------------------------------------------------------------------------
int main (int argc, char *argv[])
{
  char *password = ""; //"Secret";

  kvrConfigHandle handle;
  kvrStatus status;
  int canlib_channel;

  DWORD serial[2];
  unsigned long ean[2];
  canStatus stat;

  // Initialize kvrlib
  kvrInitializeLibrary();
  canInitializeLibrary();

  switch (argc) {
  case 1:
    listDevices();
    return 0;
  case 2:
    canlib_channel = argv[1][0] - '0';
    break;
  default:
    listDevices();
    return -1;
  }


  stat = canGetChannelData(canlib_channel, canCHANNELDATA_CARD_UPC_NO, ean, sizeo
      f(ean));
  stat = canGetChannelData(canlib_channel, canCHANNELDATA_CARD_SERIAL_NO, serial,
       sizeof(serial));

  //------------------------------------------------------------------------------
  // Check configuration status

  // Can we configure the device?
  if (isAvailibleForConfig (canlib_channel, password)) {
    printf("Channel %d is availible for configuration\n", canlib_channel);
  } else {
    printf("Channel %d can not be opened for configuration.\n", canlib_channel);
    canUnloadLibrary();
    return -1;
  }
  canUnloadLibrary();

  //------------------------------------------------------------------------------
  // Start configuration - read only
  status = kvrConfigOpen(canlib_channel, kvrConfigMode_R, password, &handle);
  if (status != kvrOK) {
    printf("Could not start config (%d)\n", status);
    return status;
  }
```

```
  // List available networks. This information could be
  // helpful when creating the XML configuration.
  status = doScanNetworks(handle);
  if (status != kvrOK) {
    printf("Scan networks failed (%d)\n", status);
    kvrConfigClose(handle);
    return status;
  }

  kvrConfigClose(handle);

  //------------------------------------------------------------------------
  // Start configuration - read/write

  status = kvrConfigOpen(canlib_channel, kvrConfigMode_RW, password, &handle);
  if (status != kvrOK) {
    printf("Could not start config (%d)\n", status);
    return status;
  }

  // Configure the device by writing the new XML configuration
  status = doConfigure(handle);
  if (status != kvrOK) {
    printf("Could not write new configuration (%d)\n", status);
    kvrConfigClose(handle);
    return status;
  }

  // Done!
  kvrConfigClose(handle);

  // Wait for reboot
  if (waitForDevice(ean[1], ean[0], serial[0], 10000) != 0) { //10s timeout
    printf("waitForDevice() failed.\n");
    return -1;
  }

  //------------------------------------------------------------------------
  // Start configuration - read only
  status = kvrConfigOpen(canlib_channel, kvrConfigMode_R, password, &handle);
  if (status != kvrOK) {
    printf("Could not start config (%d)\n", status);
    return status;
  }

  // Test the new configuration for 5 s
  status = doTryConfiguration(handle, 5);
  if (status != kvrOK) {
    printf("doTryConfiguration failed (%d)\n", status);
    return status;
  }

  kvrConfigClose(handle);

  printf("\nDone!\n");

  kvrUnloadLibrary();
  return 0;
}
```

9.2 kvrConnect.c

```
/*
 * This examples shows how to find a device on your network.
 */

#include <stdio.h>
#include "canlib.h"
#include "kvrlib.h"

#ifndef MIN
# define MIN(a, b) (((a) < (b)) ? (a) : (b))
#endif
#ifndef MAX
# define MAX(a, b) (((a) > (b)) ? (a) : (b))
#endif


char password[64] = "Hello World!";


char *getUsage (int usage)
{
  static char *free    = "FREE";
  static char *remote  = "REMOTE";
  static char *usb     = "USB";
  static char *config  = "CONFIG";
  static char *unknown = "UNKNOWN";

  switch (usage) {
    case kvrDeviceUsage_FREE:   return free;
    case kvrDeviceUsage_REMOTE: return remote;
    case kvrDeviceUsage_USB:    return usb;
    case kvrDeviceUsage_CONFIG: return config;
    default:                    return unknown;
  }
}


int isUsedByMe (kvrDeviceInfo *di)
{
  int i;
  int channel_count;
  canStatus status;
  char ean[8];
  unsigned int ean_hi = 0;
  unsigned int ean_lo = 0;
  unsigned int serial = 0;

  canInitializeLibrary();

  status = canGetNumberOfChannels(&channel_count);
  if (status != canOK) {
    printf("ERROR: canGetNumberOfChannels failed %d\n", status);
    return -1;
  }

  for (i = 0; (status == canOK) && (i < channel_count); i++) {
    status = canGetChannelData(i, canCHANNELDATA_CARD_UPC_NO, ean, sizeof(ean));
    if (status != canOK) {
      printf("ERROR: canCHANNELDATA_CARD_UPC_NO failed: %d\n", status);
      return -1;
```

```
    }
    ean_hi  = (ean[7] << 24) & 0xFF000000;
    ean_hi += (ean[6] << 16) & 0x00FF0000;
    ean_hi += (ean[5] << 8)  & 0x0000FF00;
    ean_hi += ean[4]         & 0x000000FF;
    ean_lo  = (ean[3] << 24) & 0xFF000000;
    ean_lo += (ean[2] << 16) & 0x00FF0000;
    ean_lo += (ean[1] << 8)  & 0x0000FF00;
    ean_lo += ean[0]         & 0x000000FF;

    status = canGetChannelData(i, canCHANNELDATA_CARD_SERIAL_NO,
                               &serial, sizeof(serial));
    if (status != canOK) {
      printf("ERROR: canCHANNELDATA_CARD_SERIAL_NO failed: %d\n", status);
      return -1;
    }

    if ((di->ean_lo == ean_lo) && (di->ean_hi == ean_hi) &&
        (di->ser_no == serial))
    {
      return 1;
    }
  }

  return 0;
}


//--------------------------------------------------------------------------
// Dump information
//--------------------------------------------------------------------------
void dumpDeviceInfo (kvrDeviceInfo *di)
{
  char      service_text[256];
  char      buf[256];
  kvrStatus status;
  char      addr_buf[22];
  int       i;
  int32_t   service_state;
  int32_t   service_sub_state;

  printf("------------------------------------------------------------------------
      ---\n");
  printf("Device information\n");
  printf("EAN:          %x%x\n", di->ean_hi, di->ean_lo);
  printf("FW version:   %d.%d.%d\n",
         di->fw_major_ver, di->fw_minor_ver, di->fw_build_ver);
  printf("Serial:       %ld\n", di->ser_no);
  printf("Name:         %s\n", di->name);
  printf("Host name:    %s\n", di->host_name);
  printf("Password:     ");
  if (!di->accessibility_pwd[0]) {
    printf("None\n");
  } else {
    for(i = 0; i < MIN(32, di->accessibility_pwd[0]); i++) {
      if (i == 16) {
       printf("\n              ");
      }
      printf("%02x ", (uint8_t)di->accessibility_pwd[i + 3]);
    }
    printf(" (%02x%02x)\n",
           (uint8_t)di->accessibility_pwd[2], (uint8_t)di->accessibility_pwd[1]);
```

```
  }
  kvrStringFromAddress(addr_buf, sizeof(addr_buf), &di->device_address);
  printf("IP:          %s\n", addr_buf);
  kvrStringFromAddress(addr_buf, sizeof(addr_buf), &di->client_address);
  printf("Client IP:   %s\n", addr_buf);
  printf("Usage:       %s", getUsage(di->usage));
  if ((di->usage != kvrDeviceUsage_FREE) && (isUsedByMe(di) > 0)) {
    printf(" - Used by Me!\n");
  } else if (di->usage != kvrDeviceUsage_FREE && di->usage !=
      kvrDeviceUsage_UNKNOWN) {
    printf(" - Used by other!\n");
  } else {
    printf("\n");
  }
  printf("Alive:       %s\n",
         (di->availability & kvrAvailability_FOUND_BY_SCAN ? "Yes" : "No"));
  printf("Stored:      %s\n", (di->availability & kvrAvailability_STORED ? "Yes"
      : "No"));

  // Ask service for status service_text
  status = kvrDeviceGetServiceStatusText(di, service_text,
                                         sizeof(service_text));
  if (status != kvrOK) {
    kvrGetErrorText(status, buf, sizeof(buf));
    printf("Service:     FAILED - %s\n", buf);
  } else if (strncmp(service_text, "Service: ", strlen("Service: ")) == 0) {
    printf("Service:     %s\n", &service_text[strlen("Service: ")]);
  } else {
    printf("%s\n", service_text);
  }

  status = kvrDeviceGetServiceStatus(di, &service_state, &service_sub_state);
  if (status == kvrOK) {
    printf("service_state: %d.%d\n", service_state, service_sub_state);
  } else {
    printf("service_state: unknown\n");
  }
}


//------------------------------------------------------------------------------
// Broadcast for all devices of a specific EAN and add them to the device list
// Note that the device list could have some devices in it already.
//------------------------------------------------------------------------------
kvrStatus setupBroadcast (kvrDiscoveryHandle handle)
{
  char        buf[256];
  kvrStatus   status;
  kvrAddress  addr_list[10];
  uint32_t    no_addrs        = 0;
  uint32_t    i;

  status = kvrDiscoveryGetDefaultAddresses(addr_list,
                                           sizeof(addr_list)/sizeof(kvrAddress),
                                           &no_addrs,
                                           kvrAddressTypeFlag_ALL);

  if (status != kvrOK) {
    kvrGetErrorText(status, buf, sizeof(buf));
    printf("kvrDiscoveryGetDefaultAddresses() FAILED - %s\n", buf);
    return status;
```

```
  }

  if (no_addrs < sizeof(addr_list)/sizeof(kvrAddress)) {
    status = kvrAddressFromString(kvrAddressType_IPV4, &addr_list[no_addrs], "10.
      0.3.66");
    if (status != kvrOK) {
      printf("ERROR: kvrAddressFromString(%d, 10.0.3.1) failed\n",no_addrs);
    } else {
      no_addrs++;
    }
  } else {
    printf("NOTE: We don't have room for all devices in addr_list[%d].\n",
           sizeof(addr_list)/sizeof(kvrAddress));
  }

  for (i=0; i < no_addrs; i++) {
    status = kvrStringFromAddress(buf, sizeof(buf), &addr_list[i]);
    printf("Looking for device using: %s\n", buf);
  }

  status = kvrDiscoverySetAddresses(handle, addr_list, no_addrs);
  if (status != kvrOK) {
    kvrGetErrorText(status, buf, sizeof(buf));
    printf("kvrDiscoverySetAddresses() FAILED - %s\n", buf);
    return status;
  }

  return status;
}


//------------------------------------------------------------------------------
// Discover devices and add them to device list
//------------------------------------------------------------------------------
kvrStatus discoverDevices (kvrDiscoveryHandle handle)
{
  kvrStatus status;
  kvrDeviceInfo device_info[64];
  int devices;
  uint32_t delay_ms = 500;
  uint32_t timeout_ms = 300;
  char        buf[256];

  status = kvrDiscoveryStart(handle, delay_ms, timeout_ms);
  if (status != kvrOK) {
    kvrGetErrorText(status, buf, sizeof(buf));
    printf("kvrDiscoveryStart() FAILED - %s\n", buf);
    return status;
  }

  devices = 0;
  while (status == kvrOK) {
    status = kvrDiscoveryGetResults(handle, &device_info[devices]);
    if (status == kvrOK) {
      dumpDeviceInfo(&device_info[devices]);
      // Add some data and request store
      if (kvrDiscoverySetPassword(&device_info[devices], password) != kvrOK) {
        printf("Unable to set password: %s (%d)\n", password, strlen(password));
      }
      // Here we can decide to connect to the device
      //device_info[devices].request_connection = 1;
      devices++;
```

```
    } else {
      if (status != kvrERR_BLANK) {
        printf("kvrDiscoveryGetResults() failed %d\n", status );
      }
    }
  }

  status = kvrDiscoveryStoreDevices(device_info, devices);
  if (status != kvrOK) {
    kvrGetErrorText(status, buf, sizeof(buf));
    printf("Device store failed: %s\n", buf);
  }

  return kvrOK;
}



//--------------------------------------------------------------------------
// Setup a number of WLAN devices for future CANLIB use
//--------------------------------------------------------------------------
int main (int argc, char *argv[])
{
  kvrStatus status;
  kvrDiscoveryHandle handle;
  char        buf[256];

  if (argc > 1) {
    strcpy(password, argv[1]);
  }

  kvrInitializeLibrary();

  status = kvrDiscoveryOpen(&handle);
  if (status != kvrOK) {
    kvrGetErrorText(status, buf, sizeof(buf));
    printf("kvrDiscoveryOpen() FAILED - %s\n", buf);
    return status;
  }

  status = setupBroadcast(handle);
  if (status != kvrOK) {
    kvrGetErrorText(status, buf, sizeof(buf));
    printf("setupBroadcast() FAILED - %s\n", buf);
    return status;
  }
  status = discoverDevices(handle);
  if (status != kvrOK) {
    kvrGetErrorText(status, buf, sizeof(buf));
    printf("discoverDevices() FAILED - %s\n", buf);
    return status;
  }

  kvrDiscoveryClose(handle);
  kvrUnloadLibrary();

  return 0;
}
```

9.3 kvrNetworkConnectionTest.c

```
/*
 * This example shows how to get RSSI and RTT values for a connection.
 */

#include <stdio.h>
#include "canlib.h"
#include "kvrlib.h"


//------------------------------------------------------------------------------
// List all connected devices
//------------------------------------------------------------------------------
void listDevices (void)
{
  int i;
  canStatus status = canOK;
  char name[100];
  int channel_count = 0;

  canInitializeLibrary();
  canGetNumberOfChannels(&channel_count);

  printf("First argument must be a channel!\n\n");
  printf("Channel\t Name\n");
  printf("-------------------------------------------------------\n");
  for (i = 0; (status == canOK) && (i < channel_count); i++) {
    name[0] = '\0';
    status = canGetChannelData(i, canCHANNELDATA_CHANNEL_NAME, name, sizeof(name)
      );
    printf("%d\t %s\n", i, name);
  }
}


//------------------------------------------------------------------------------
// Get RSSI and RTT values
//------------------------------------------------------------------------------
void getRssiRtt (kvrConfigHandle handle, int *rssi_mean, int *rtt_mean)
{
  kvrStatus status;
  kvrRssiHistory rssi[14] = {0};
  kvrRttHistory rtt[14] = {0};
  int rtt_len = sizeof(rtt) / sizeof(kvrRttHistory);
  int rssi_len = sizeof(rssi) / sizeof(kvrRssiHistory);
  int rtt_actual;
  int rssi_actual;

  *rssi_mean = 0;
  *rtt_mean = 0;

  status = kvrNetworkGetRssiRtt(handle, rssi, rssi_len, &rssi_actual,
                                rtt, rtt_len, &rtt_actual);
  if (status == kvrOK) {
    int i;
    printf("RSSI (%d):", rssi_actual);
    for(i = 0; i < rssi_actual; i++) {
      *rssi_mean += rssi[i];
      printf(" %d", rssi[i]);
    }
    printf("\nRTT (%d):", rtt_actual);
```

```
    for(i = 0; i < rtt_actual; i++) {
      *rtt_mean += rtt[i];
      printf(" %d", rtt[i]);
    }

    printf("\n");

    if (rssi_actual) {
      *rssi_mean = *rssi_mean / rssi_actual;
    }

    if (rtt_actual) {
      *rtt_mean = *rtt_mean / rtt_actual;
    }
  }
}


//------------------------------------------------------------------------------
// kvrNetworkConnectionTest.exe <channel>
//------------------------------------------------------------------------------
int main (int argc, char *argv[])
{
  kvrConfigHandle config_handle;
  kvrStatus status;
  int canlib_channel = 0;
  int j;
  int rssi_mean, rtt_mean;
  kvrInitializeLibrary();

  switch (argc) {
  case 1:
    listDevices();
    return 0;
  case 2:
    canlib_channel = argv[1][0] - '0';
    break;
  default:
    listDevices();
    return -1;
  }

  printf("canlib channel = %d\n", canlib_channel);

  status = kvrConfigOpen(canlib_channel, kvrConfigMode_R, "", &config_handle);
  if (status != kvrOK) {
    printf("Could not start config (%d)\n", status);
    return status;
  }

  printf("config_handle = %d\n\n", config_handle);

  printf("kvrNetworkConnectionTest( on )\n\n");
  status = kvrNetworkConnectionTest(config_handle, 1); // Start sending pings.
  if (status != kvrOK) {
    printf("Could not start connection test (%d)\n", status);
    return status;
  }

  Sleep(2000); // Wait for the device to connect.

  for (j = 0; j < 10; j++) {
```

```
    Sleep(2000); // Wait for some pings to be sent.
    getRssiRtt(config_handle, &rssi_mean, &rtt_mean);
    printf("rssi_mean = %d\n", rssi_mean);
    printf("rtt_mean = %d\n\n", rtt_mean);
  }

  status = kvrNetworkConnectionTest(config_handle, 0); // Stop sending pings.
  printf("Done!\n");
  kvrConfigClose(config_handle);

  return 0;
}
```