Amelie API documentation

Document List

Disclaimer: This document can be subject to change without prior notice.

Contents

Amelie API

1 Document Info

1.1 History

Revision	Author	Issue Date	Comments
0.01	LKA	22-Feb-12	Initial Revision
0.02	LKA	15-May-12	Added API_CALIBRATE_LFRCO_REQ/CFM and API_PROD_TEST_CMD_IND/RES
0.03	TKP	27-Jun-12	Added MSC
0.04	LKA	26-Oct-12	Added API for accessing the following NVS parameters: • ApInfo • Static IPv4 Address incl. subnet and gateway address • Custom key • Custom ID
0.05	LKA	7-Nov-12	Added description of the CoLA interface.
0.06	LKA	19-Jun-13	Added API for storing DNS server address in the NVS.
0.07	LKA	5-Aug-13	Added generic API for NVS parameter access. Removed the API's used to set/get DNS server address and custom key/id.
0.08	LKA	16-Aug-13	Removed the old API mails for set/get of AP info and static IPv4 address info. This is now set with the generic API_SET_NVS_PARAMS_REQ.

1.2 References

1.3 Terms & Abbreviations

2 Introducing the Amelie Platform API

The API used to program Co-Located Applications (CoLA) on top of the Amelie Wi-Fi platform is described in this document. The API is, as outlined on Figure 1, composed of a function call based CoLA interface and a set of mail based API's used to access the functions implemented by the platform.

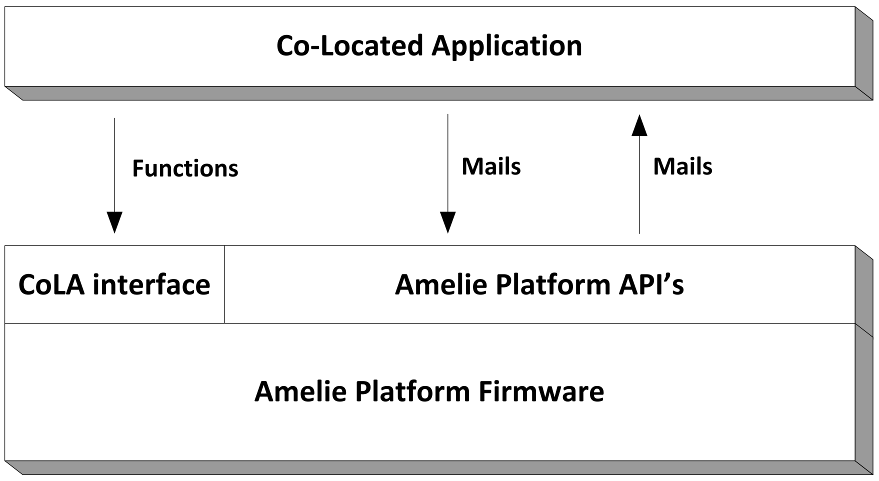


Figure 1 This figure illustrates how the CoLA is accessing the Amelie Platform.

The Amelie Platform API is composed of the following mail based API's:

- Amelie Management API
- Atheros Wi-Fi API
- Socket API
- IpConfig API
- DNS client API

The API's are all described in details in separate sections of this document. The Cola interface is described in the following.

2.1 The CoLA interface

The CoLA interface exports a subset of the RTX OS environment (RTX Core Interface) via a special data/control block stored at a fixed location in the FLASH. The CoLA interface offers functions to access the following modules inside the platform FW:

- OS – for mail and timer handling
- Heap – for dynamic allocation of memory
- NVS – for storage of data in a none volatile storage (FLASH)
- System – for performance counter and power management.
- RtxEai – for logging of comments to mail log and screen output.

All the functions available through the CoLA interface, except for the NVS functions, are prototyped in RtxCore.h (or file included by RtxCore.h). Please note that it is a subset of all the functions and global variables prototyped in RtxCore.h that is available to the CoLA. The first file included in implementation files using the CoLA interface must be either Cola.h or RtxCore.h. The prototypes of all the functions available through the CoLA interface are listed in the following with a brief description of the function.

2.1.1 OS

The CoLA interface allows access to the following RTX OS functions:

```
void RsShutdown(RsShutDownReasonType Reason, RsShutDownActionType Action)
```
This function is used to initiate a system shutdown or reset the system.

```
void RosTaskTerminated(RosTaskIdType TaskId)
```
This function must be called frim the CoLA task when the `TERMINATETASK` mail has been processed and the CoLA task is ready to stop.

```
RosMailType* RosMailAllocate(RosTaskIdType Src, RosTaskIdType Dst,
                             RosMailSizeType Size)
```
This function is used to allocate a mail packet. All mails allocated must be delivered using `RosMailDeliver()` before the CoLA task returns control to the OS in the platform.

```
void RosMailDeliver(RosMailType* Mail)
```
This function is used to deliver a mail packet allocated by `RosMailAllocate()`.

```
RosMailSizeType RosMailGetSize(const RosMailType* Mail)
```
This function can be used to get the size of a mail. This is typically needed if the application needs to store a copy of the mail for later processing.

```
RosTaskIdType RosMailGetSrc(const RosMailType* Mail)
```
This function can be used to obtain the task id of the task that has sent the mail.

```
RosTaskIdType RosMailGetDst(const RosMailType* Mail)
```
This function can be used to obtain the task id of the task that will receive the mail. This is used for debug only normally.

```
void RosMailSendP0(RosTaskIdType Src, RosTaskIdType Dst,
                   RosPrimitiveType Primitive)
```
This function can be used to allocate and send a mail with no parameters (primitive only).

```
void RosMailSendP1(RosTaskIdType Src, RosTaskIdType Dst,
                  RosPrimitiveType Primitive, rsuint8 P1)
```
This function can be used to send a mail with a single parameter of `rsuint8` type.

```
void RosMailSendP2(RosTaskIdType Src, RosTaskIdType Dst,
                   RosPrimitiveType Primitive, rsuint8 P1, rsuint8 P2)
```
This function can be used to send a mail with two parameters of `rsuint8` type.

```
void RosMailSendP3(RosTaskIdType Src, RosTaskIdType Dst,
                   RosPrimitiveType Primitive, rsuint8 P1, rsuint8 P2,
                   rsuint8 P3)
```
This function can be used to send a mail with three parameters of `rsuint8` type.

```
void RosMailSendCopy(RosTaskIdType Src, RosTaskIdType Dst,
                     const RosMailType* Mail, RosMailSizeType Size)
```
This function can be used to send a copy of a mail. Used to forward received mails typically.

```
void RosTimerStart(RosTimerIdType TimerId, RsTimerTickType Value,
                   const RosTimerConfigType* Config)
```
This function is used to start or restart a timer. Timeouts are signalled with a mail which is defined in the `Config` struct parsed to this function.

```
void RosTimerStop(RosTimerIdType TimerId)
```
This function is used to stop a timer.

2.1.2 Heap

The CoLA interface allows access to the following heap functions:

```
void* RcHeapAlloc(RcHeapSizeType Size)
```
This function allocates a block with the size specified on the system heap. The system will reset if the heap block cannot be allocated!

```
void* RcHeapAllocEx(RcHeapSizeType Size, RcHeapFlagsType Flags)
```
This function is identical to `RcHeapAlloc` with the only exception that it is possible with the `Flags` parameter to specify what the function should do if it fails to allocate the block on the heap. Setting `Flags = RHF_NULL_ON_FULL` can be used to get this function to return a `NULL` pointer if the block cannot be allocated.

```
void* RcHeapRealloc(void* Block, RcHeapSizeType Size)
```
This function can be used to reallocate a heap block with different size. The content of the original block is copied to the beginning of the new block allocated before a pointer to the new block is returned. The system will reset if the heap block cannot be allocated!

```
void* RcHeapReallocEx(void* Block, RcHeapSizeType Size, RcHeapFlagsType Flags)
```
This function is identical to `RcHeapRealloc` with the only exception that it is possible with the `Flags` parameter to specify what the function should do if it fails to allocate the block on the heap. Setting `Flags = RHF_NULL_ON_FULL` can be used to get this function to return a `NULL` pointer if the block cannot be allocated.

```
void RcHeapFree(void* Block)
```
This function is used to free a heap block.

2.1.3 NVS

The CoLA task has access to the system None Volatile Storage (NVS) through the following two functions:

```
void NvsRead(rsuint16 Address, rsuint16 Size, rsuint8* Data)
```
This function is used to read from the NVS.

```
void NvsWrite(rsuint16 Address, rsuint16 Size, rsuint8* Data)
```
This function is used to write data to the NVS.

IMPORTANT: The CoLA task has access to the entire NVS, but is should access the part allocated for the CoLA task only. The offset of the first byte allocated for the CoLA task can be read from the global variable `ColaNvsOffset` and the size of the NVS area allocated for the CoLA task can be read from `ColaNvsSize.` E.g. writing to a byte at offset 10 in the CoLA NVS should be done by: `NvsWrite(ColaNvsOffset+10, 1, &Data)` where Data is holding the data to write.

2.1.4 System

The following system calls are available through the CoLA interface:

```
RsStatusType SetIsr(rsuint8 Vector, ColaIsrType* Isr)
```
This function is used to register an interrupt handler from the CoLA task.

```
RsPerformanceTickType RsReadPerformanceCounter(void)
```
This function is used to read the performance counter which is a counter driven with the 28MHz system clock.

```
RsPerformanceTimeType
RsPerformanceTicks2Time(RsPerformanceTickType PerformanceTicks)
```
This function is used to convert performance counts to time.

```
RsPerformanceTickType RsTime2PerformanceTicks(RsPerformanceTimeType Time)
```
This function is used to convert from time to performance counts.

```
void ScCpuRequest(ScCpuUserIdType UserId, ScStateType State)
```
This function can be used by the CoLA task to request the high frequency clock being held active by preventing the platform from going in sleep mode. This is not needed normally as the platform FW makes sure that the system clock is running when the CoLA task is executed.

2.1.5 RtxEai

The CoLA interface offers support for generation of some debug info which is sent to a PC via the RTX Embedded Access Interface (RtxEai). The following log functions are available to the CoLA task:

```
void RtxEaiLogComment(rsuint8 TaskId, const char* format, ...)
void RtxEaiVaLogComment(rsuint8 TaskId, const char* format, va_list argptr)
```
These functions can be used to log a comment/print a text to the mail log in RSX.

```
void RtxEaiPrintf(rsuint8 TaskId, const char* format, ...)
void RtxEaiVaPrintf(rsuint8 TaskId, const char* format, va_list argptr)
```
These functions are used to print a test to the screen window in the RSX.

```
void RtxEaiClearScreen(rsuint8 TaskId)
```
This function can be used to clear the screen window in RSX.

```
void RtxEaiScreenGotoxy(rsuint8 TaskId, rsuint8 X, rsuint8 Y)
```
This function can be used to set the x,y pos in the screen window in RSX.

2.2 Accessing the Hardware

The platform FW implements and uses drivers for the Wi-Fi module and UART communication drivers for one of the USART's and one of the LEUART's. Furthermore the RTC is used to drive the timer system in the OS in the platform. The HW blocks used by the platform are not accessible to the CoLA, but a mechanism used by the CoLA to take over the UART drives is implemented.

The CoLA can access the internal HW block not used by the platform through the “em32lib” provided by EnergyMicro. Source code for the em32lib can be found in the 3Party folder where the SDK is installed. The em32lib is included as source in the build of the CoLA application.

2.3 Naming of the API mail primitives

The API mail naming scheme described in this section applies to all the API’s offered by the Amelie platform. Four types of primitives exist, Request (REQ), Indicate (IND), Response (RES) and Confirm (CFM). A “CFM” primitive only occurs as confirmation of an action initiated by a “REQ” primitive. A “RES” primitive can only follow an “IND” primitive. The direction of the primitives is shown in the figure below.

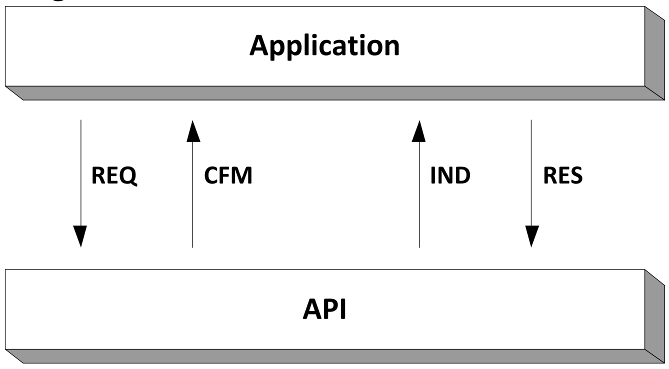


Figure 1 Illustration of the primitive direction used.

CFM primitives are normally sent to the application when the requested operation has been completed. The API will return a CFM primitive immediately in case a requested operation cannot be handled by the API. This will be indicated by the status parameter of the CFM.

In peer to peer (P2P) communication between e.g. a client and a server a sent “REQ” can be received as an “IND” in the other party and a sent “RES” will be received as a “CFM”.

3 Amelie Management API Specification

3.1 Amelie management

The API’s for general management are described in this section.

3.1.1 API_DEVICE_CONTROL_REQ

Description:	This mail is sent from the application to control a device (driver). It is intended for the application (usually a CoLA application) to disable a specific device driver (together with the interrupt used). The application can then takes over and use its own driver with a custom interrupt handler.
IntInPrimitive:	`API_DEVICE_CONTROL_REQ = 0x6F00`

Parameters:

Type	Name	Description
`ApiDeviceIdType`	`DeviceId`	Device ID.
`ApiDeviceControlType`	`Control`	Type of control.

3.1.2 API_DEVICE_CONTROL_CFM

Description: This mail is used to confirm API_DEVICE_CONTROL_REQ.
IntOutPrimitive: `API_DEVICE_CONTROL_CFM = 0x6F01`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	Status for the request. RSS_SUCCESS: Success. RSS_NOT_SUPPORTED: The device does not support the control requested. RSS_NOT_FOUND: No such device.
`ApiDeviceIdType`	`DeviceId`	Device ID.
`ApiDeviceControlType`	`Control`	Type of control.

3.1.3 API_SET_TIME_REQ

Description: This mail is used by the application to set the current system time.
IntInPrimitive: `API_SET_TIME_REQ = 0x6F02`
Parameters:

Type	Name	Description
`rsuint32`	`Time`	The relative time in seconds since 1970.

3.1.4 API_SET_TIME_CFM

Description: This mail is used to confirm API_SET_TIME_REQ.
IntOutPrimitive: `API_SET_TIME_CFM = 0x6F03`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

3.1.5 API_GET_TIME_REQ

Description: This mail is used by the application to request the current system time.
IntInPrimitive: `API_GET_TIME_REQ = 0x6F04`
Parameters: No parameters.

3.1.6 API_GET_TIME_CFM

Description: This mail is used to confirm API_SET_TIME_REQ.
IntOutPrimitive: `API_GET_TIME_CFM = 0x6F05`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint32`	`Time`	The relative time in seconds since 1970.
`rsuint32`	`TickCount`	Current system timer tick count in ms.

3.1.7 API_GET_PLATFORM_VERSION_REQ

Description: This mail is used by the application to request information about the version of the platform FW.
IntInPrimitive: `API_GET_PLATFORM_VERSION_REQ = 0x6F06`
Parameters: No parameters.

3.1.8 API_GET_PLATFORM_VERSION_CFM

Description: This mail is used to confirm API_SET_PLATFORM_VERSION_REQ.
IntOutPrimitive: `API_GET_PLATFORM_VERSION_CFM = 0x6F07`
Parameters:

Type	Name	Description
`rsuint16`	`Version`	Hex representation of the version. E.g. 0x0102 equals version 01.02.
`rsuint16`	`SubVersion`	Hex representation of sub version. Will be 0 normally. Only used if a branch of the platform FW is made.
`rsuint16`	`BuildNumber`	The build number is incremented each time the platform is released.
`rsuint8`	`LinkDate[5]`	BCD encoded link date. E.g. LinkDate = {12,03,24,10,45} equals 2012-03-24 10:45.
`rsuint8`	`LabelLength`	The length in bytes of the Label[] parameter.
`rsuint8`	`Label[1]`	The label used when the platform is released.

3.1.9 API_CALIBRATE_LFRCO_REQ

Description: This mail is used by the application to request a calibration of the LFRCO (internal 32kHz RC oscillator). The current HFCLK is used as reference. A repeating timer is started if Interval > 0 and a new LFRCO calibration is performed each time this time expires. This timer can be stopped by issuing a request with Interval = 0. A calibration of the LFRCO is performed at power on but the calibration timer is not started at this point.
IntInPrimitive: `API_CALIBRATE_LFRCO_REQ = 0x6F08`
Parameters:

Type	Name	Description
`rsuint16`	`Interval`	Calibration interval in secs.

3.1.10 API_CALIBRATE_LFRCO_CFM

Description: This mail is used to confirm API_CALIBRATE_LFRCO_REQ.
IntOutPrimitive: `API_CALIBRATE_LFRCO_CFM = 0x6F09`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

3.1.11 API_PROD_TEST_CMD_IND

Description: This mail is used to inform the application if a custom product test command has been received. The product test commands are received on the UART managed by the platform FW.
IntOutPrimitive: `API_PROD_TEST_CMD_IND = 0x6F0A`
Parameters:

Type	Name	Description
`rsuint8`	`Command`	The op-code of the command this is a response to.
`rsuint8`	`Length`	Length in bytes of Data[]
`rsuint8`	`Data[1]`	Command specific data.

3.1.12 API_PROD_TEST_CMD_RES

Description: This mail is used by the application to return a response to the product test command received previously by API_PROD_TEST_CMD_IND.

IntInPrimitive: `API_PROD_TEST_CMD_RES = 0x6F0B`

Parameters:

Type	Name	Description
`rsuint8`	`Command`	The op-code of the command this is a response to.
`RsStatusType`	`Status`	Test result/status.
`rsuint8`	`Length`	Length in bytes of Data[]
`rsuint8`	`Data[1]`	Command specific response data.

3.2 Amelie GPIO management

The GPIO interrupts are handled by the platform because the same two interrupts are used for multiple GPIO's, and hence share the same interrupt handlers. The platform handles the GPIO interrupts and notifies the application by mail if a GPIO interrupt has occurred. The application can use the GPIO functionality of the efm32lib to setup the GPIO's used.

3.2.1 API_GPIO_INTERRUPT_IND

Description: This mail is used inform the application when a GPIO interrupt has been detected.

IntOutPrimitive: `API_GPIO_INTERRUPT_IND = 0x6F20`

Parameters:

Type	Name	Description
`rsuint32`	`GpioInterrupt`	The GPIO interrupts detected.

3.2.2 API_ENABLE_STEP_UP_PORT_REQ

Description: This mail is used by the application to enable or disable the step up port.

IntInPrimitive: `API_ENABLE_STEP_UP_PORT_REQ = 0x6F21`

Parameters:

Type	Name	Description
`rsbool`	`Enable`	Enable/Disable true/false step up port.

3.2.3 API_ENABLE_STEP_UP_PORT_CFM

Description: This mail is used to confirm API_ENABLE_STEP_UP_PORT_REQ.

IntOutPrimitive: `API_ENABLE_STEP_UP_PORT_CFM = 0x6F22`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

3.3 Amelie NVS management

The API's defined in this section can be used by the application to access some parameters stored in the NVS.

3.3.1 API_SET_APINFO_REQ

Description: This mail is used by the application to store information about the AP the module is associated to.

IntInPrimitive: `API_SET_APINFO_REQ = 0x6F30`

Parameters:

Type	Name	Description
`rsuint8`	`SsidLength`	
`rsuint8`	`Ssid[API_SSID_LENGTH]`	
`rsuint8`	`SecurityType`	
`rsuint8`	`Ucipher`	
`rsuint8`	`Mcipher`	
`rsuint8`	`KeyIndex`	WEP key index
`rsuint8`	`KeyLength`	Max key length is 64
`rsuint8`	`Key[1]`	

3.3.2 API_SET_APINFO_CFM

Description: This mail is used to confirm API_SET_APINFO_REQ.

IntOutPrimitive: `API_SET_APINFO_CFM = 0x6F31`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

3.3.3 API_GET_APINFO_REQ

Description: This mail is used by the application to request the stored AP information.

IntInPrimitive: `API_GET_APINFO_REQ = 0x6F32`

Parameters: No parameters.

3.3.4 API_GET_APINFO_CFM

Description: This mail is used to confirm API_GET_APINFO_REQ.

IntOutPrimitive: `API_GET_APINFO_CFM = 0x6F33`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint8`	`SsidLength`	
`rsuint8`	`Ssid[API_SSID_LENGTH]`	
`rsuint8`	`SecurityType`	
`rsuint8`	`Ucipher`	
`rsuint8`	`Mcipher`	
`rsuint8`	`KeyIndex`	WEP key index
`rsuint8`	`KeyLength`	Max key length is 64
`rsuint8`	`Key[1]`	

3.3.5 API_SET_STATIC_IPV4_REQ

Description: This mail is used by the application to store static IPv4 address info.
IntInPrimitive: `API_SET_STATIC_IPV4_REQ = 0x6F34`
Parameters:

Type	Name	Description
`rsuint32`	`Address`	Set to 0xFFFFFFFF if DHCP is used.
`rsuint32`	`SubnetMask`	
`rsuint32`	`Gateway`	
`rsuint32`	`PrimDnsServer`	
`rsuint32`	`SecDnsServer`	

3.3.6 API_SET_STATIC_IPV4_CFM

Description: This mail is used to confirm API_SET_STATIC_IPV4_REQ.
IntOutPrimitive: `API_SET_STATIC_IPV4_CFM = 0x6F35`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

3.3.7 API_GET_STATIC_IPV4_REQ

Description: This mail is used by the application to read the IPv4 address information stored.
IntInPrimitive: `API_GET_STATIC_IPV4_REQ = 0x6F36`
Parameters: No parameters.

3.3.8 API_GET_STATIC_IPV4_CFM

Description: This mail is used to confirm API_GET_STATIC_IP_REQ.
IntOutPrimitive: `API_GET_STATIC_IPV4_CFM = 0x6F37`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint32`	`Address`	Set to 0xFFFFFFFF if DHCP is used.
`rsuint32`	`SubnetMask`	
`rsuint32`	`Gateway`	
`rsuint32`	`PrimDnsServer`	
`rsuint32`	`SecDnsServer`	

3.4 Type definitions

3.4.1 Includes

Description: Include of API configuration.
C-syntax:

```
#include <Api/ApiCfg.h>
```

3.4.2 API_SSID_LENGTH

Description: The max length of an ssid string.
C-syntax:

```
#define API_SSID_LENGTH 32
```

3.4.3 API_KEY_LENGTH

Description: The max length of an encryption key.
C-syntax:

```
#define API_KEY_LENGTH 64
```

3.5 ApiDeviceIdType

Description: This enum defines device IDs.

C-syntax:

```
typedef enum RSENUMTAG(ApiDeviceIdType)
{
   AD_NONE,                                    No device.
   AD_USART0,                                  USART 0.
   AD_USART1,                                  USART 1.
   AD_USART2,                                  USART 2.
   AD_LEUART0,                                 LEUART 0.
   AD_LEUART1,                                 LEUART 1.
   AD_MAX
} RSENUM8(ApiDeviceIdType);
```

3.5.1 ApiDeviceControlType

Description: This enum defines device controls.

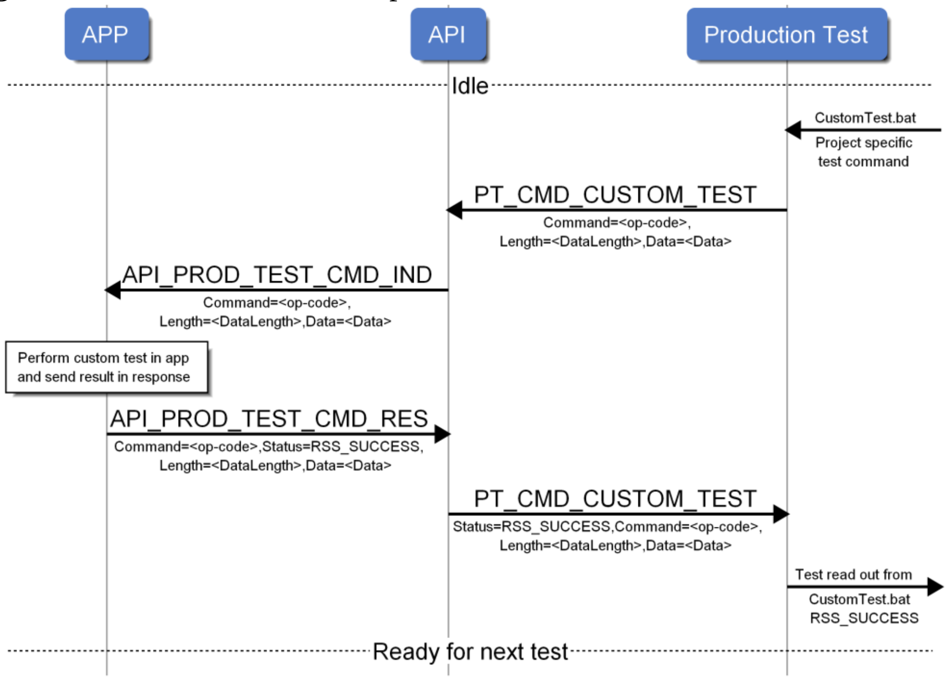
C-syntax:

```
typedef enum RSENUMTAG(ApiDeviceControlType)
{
   AC_NULL,                                    No control.
   AC_DISABLE,                                 Disable system device driver. A Co-located application may
                                               then use the hardware.
   AC_ENABLE,                                  Enable system device driver.
   AC_MAX
} RSENUM8(ApiDeviceControlType);
```

4 Sequence charts

4.1 Custom production test commands

The following MSC illustrates how custom production test commands are handled in the system.



APP
API
Production Test
Idle
CustomTest.bat
Project specific test command
PT_CMD_CUSTOM_TEST
Command=<op-code>, Length=<DataLength>,Data=<Data>
API_PROD_TEST_CMD_IND
Command=<op-code>, Length=<DataLength>,Data=<Data>
Perform custom test in app and send result in response
API_PROD_TEST_CMD_RES
Command=<op-code>,Status=RSS_SUCCESS, Length=<DataLength>,Data=<Data>
PT_CMD_CUSTOM_TEST
Status=RSS_SUCCESS,Command=<op-code>, Length=<DataLength>,Data=<Data>
Test read out from CustomTest.bat RSS_SUCCESS
Ready for next test

WEB Config API

5 Document Info

5.1 History

Revision	Author	Issue Date	Comments
0.1	CM	27-Aug-2001	Initial Revision

5.2 References

[1] A Document
[2]

5.3 Terms & Abbreviations

A Term	Explanation
ANABREV	Expanation

6 API specification

6.1 WEB config

6.1.1 API_WEB_CONFIG_INIT_REQ

Description: This mail is used by the application to initialize the WEB config server. The application must connect to an AP or set the module in SoftAP mode before the WEB config server is started.

IntInPrimitive: `API_WEB_CONFIG_INIT_REQ = 0x6FF0`

Parameters:

Type	Name	Description
`AwcInitDataType*`	`InitDataPtr`	Pointer to data structure holding initialization data for the WEB config server. The initialization data must be available as long as the WEB config server is running. Default configuration is used if a NULL pointer is passed.

6.1.2 API_WEB_CONFIG_INIT_CFM

Description: This mail is used to confirm API_WEB_CONFIG_INIT_REQ.

IntOutPrimitive: `API_WEB_CONFIG_INIT_CFM = 0x6FF1`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

6.1.3 API_WEB_CONFIG_ADD_PAGE_REQ

Description: This mail is used by the application to add a WEB page to the server.
IntInPrimitive: `API_WEB_CONFIG_ADD_PAGE_REQ = 0x6FF2`
Parameters:

Type	**Name**	**Description**
`rschar*`	`PathPtr`	The parth used to request the page
`rschar*`	`MenuTextPtr`	Optional menu text. A link to the page will be shown on the main page if a menu text is specified.
`AhResourceCbType`	`CallBack`	Pointer to the function that will generate the page when request by the client (WEB browser on remote client).

6.1.4 API_WEB_CONFIG_ADD_PAGE_CFM

Description: This mail is used to confirm API_WEB_CONFIG_ADD_PAGE_REQ.
IntOutPrimitive: `API_WEB_CONFIG_ADD_PAGE_CFM = 0x6FF3`
Parameters:

Type	**Name**	**Description**
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

6.1.5 API_WEB_CONFIG_TERMINATE_REQ

Description: This mail is used by the application to terminate the WEB config server.
IntInPrimitive: `API_WEB_CONFIG_TERMINATE_REQ = 0x6FF4`
Parameters: No parameters

6.1.6 API_WEB_CONFIG_TERMINATE_CFM

Description: This mail is used to confirm API_WEB_CONFIG_TERMINATE_REQ.
IntOutPrimitive: `API_WEB_CONFIG_TERMINATE_CFM = 0x6FF5`
Parameters:

Type	**Name**	**Description**
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

6.2 Type definitions

6.2.1 Includes

Description: Include of API configuration.
C-syntax:

```
#include <Api/ApiCfg.h>
```

Description: Include of WiFi API.
C-syntax:

```
#include <IOT/Dev/AtherosWifi/ApiAtherosWifi.h>
```

Description: Include of HTTP API.
C-syntax:

```
#include <IOT/App/Http/ApiHttp.h>
```

6.2.2 AwcInitDataType

Description: This struct is used to hold initialization data for the WEB config server.

C-syntax:

```
typedef struct AwcInitDataTypeTag
{
   rschar                  *TitlePtr;
   rschar                  *InfoTextPtr;
   rsuint32                ScanResultCount;
   ApiWifiScanResultType   *ScanResultPtr;
} AwcInitDataType;
```

rschar	*TitlePtr;	The title shown on the top of all pages. Default title is used if set to NULL.
rschar	*InfoTextPtr;	Optional info text shown below the title on the main page. Set to NULL if info text is not to be shown.
rsuint32	ScanResultCount;	Number of scan results stored in the memory block pointed to by ScanResultPtr;
ApiWifiScanResultType	*ScanResultPtr;	Pointer to scan result.

7 Sequence charts

TBD

Socket API

8 Document Info

8.1 History

Revision	Author	Issue Date	Comments
0.01	LKA	17-Jan-12	Initial Revision
0.02	LKA	23-Feb-12	Minor update.
0.03	TKP	22-Jun-12	Minor update.
0.04	LKA	28-Jun-12	Added: • `API_SOCKET_GET_TX_BUFFER_REQ/CFM` • `API_SOCKET_EXT_DATA_WRITE_REQ/CFM` • `API_SOCKET_EXT_DATA_READ_REQ/CFM`
0.05	LKA	9-Aug-12	Added instance parameter to API_SOCKET_CREATE_REQ/CFM.
0.06	LKA	8-Feb-13	Removed usage of RS_FAR_ADR

8.2 References

8.3 Terms & Abbreviations

9 The socket architecture

The purpose with the Socket API defined in this document is to make a generic API to be used by applications to access the TCP/UDP stack implemented in the lower layers of the platform. The platform will support multiple TCP/UDP stacks. The actual stack used is selected at configuration / compile time, and it is the main responsibility of the Socket API to make this decision transparent to the application.

The following two stacks are supported currently (two different platform releases):

1. The “Atherors host offload stack”, which is embedded in the AR4100 target firmware.
2. The “uIP stack”, which will be included in the platform implementation.

This is also illustrated on Figure 2where we have two different components implementing the Socket API. The first case is the ApiSocketHandler component, which is part of the code used to interface the AR4100 WiFi module. It maps the Socket API mails to the Atheros socket interface implemented in the AtherosWiFiDriver implementing the SPI protocol used to talk with the AR4100 module. In the second case the Socket API is implemented in the uIp wrapper component that interfaces the uIp stack implementation to the rest of the platform.

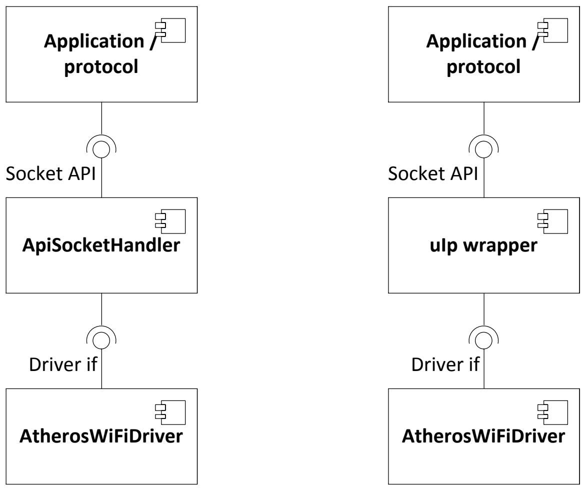


Figure 2 Illustration of two usage cases for the Socket API.

The Socket API is inspired by the Berkley Socket API, but with on big difference being that the Socket API is mail based. The Socket API offers primitives for socket creation, connection management on both client and server side, and data management. There is support for both stream (TCP) and datagram (UDP) based data transfer.

The data interface is designed to make it possible to transfer data from the application to the HW transmit buffer without having to copy the data more than once. I.e. the application passes a pointer to a buffer holding the data to be sent, and the lower layers will then copy directly from this buffer to e.g. the SPI driver or setup a DMA transfer. For received data the Socket API is designed to pass a pointer to the RX buffer in e.g. the WiFi driver to the application. Again this is done to make it possible to pass the data to the application without having to copy it to the application buffer. The drawback of this approach is that it is necessary for the application to inform the Socket API when it has finished processing the receive buffer. Failing to do so will cause the data flow to stop for all sockets.

10 API Specification

10.1 Socket management

The API's for socket management are described in this section.

10.1.1 API_SOCKET_CREATE_REQ

Description: This mail is used by the application to request the creation of a socket.
IntInPrimitive: `API_SOCKET_CREATE_REQ = 0x6100`
Parameters:

Type	Name	Description
`ApiSocketDomainType`	`Domain`	Specifies the protocol domain/family of the created socket.
`ApiSocketTypeType`	`Type`	Socket type
`ApiSocketProtocolType`	`Protocol`	Protocol used. 0 used to request the default protocol for the selected family/domain.
`rsuint32`	`Instance`	Instance can be used to match the cfm to the right application instance if multiple req's are sent simultaneously.

10.1.2 API_SOCKET_CREATE_CFM

Description: This mail is used to confirm API_SOCKET_CREATE_REQ
IntOutPrimitive: `API_SOCKET_CREATE_CFM = 0x6101`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`ApiSocketHandleType`	`Handle`	Handle to the socket created
`rsuint32`	`Instance`	Return the value of the Instance parameter of the req.

10.1.3 API_SOCKET_CLOSE_REQ

Description: This mail is used by the application to close a socket.
IntInPrimitive: `API_SOCKET_CLOSE_REQ = 0x6102`
Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket.

10.1.4 API_SOCKET_CLOSE_CFM

Description: This mail is used to confirm API_SOCKET_CLOSE_REQ
IntOutPrimitive: `API_SOCKET_CLOSE_CFM = 0x6103`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`ApiSocketHandleType`	`Handle`	Handle to the socket closed

10.1.5 API_SOCKET_CLOSE_IND

Description: This mail is used to inform the application when the remote device has closed the connection/socket.
IntOutPrimitive: `API_SOCKET_CLOSE_IND = 0x6104`
Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket.

10.1.6 API_SOCKET_CONNECT_REQ

Description: This mail is used by the client application to request a connection to a server.
IntInPrimitive: `API_SOCKET_CONNECT_REQ = 0x6105`
Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket.
`ApiSocketAddrType`	`ServerAddr`	The address of the remote socket to connect to.

10.1.7 API_SOCKET_CONNECT_CFM

Description: This mail is used to confirm API_SOCKET_CONNECT_REQ
IntOutPrimitive: `API_SOCKET_CONNECT_CFM = 0x6106`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`ApiSocketHandleType`	`Handle`	Handle to the socket.

10.1.8 API_SOCKET_BIND_REQ

Description: This mail is used by the server application to associate an address to the socket.
IntInPrimitive: `API_SOCKET_BIND_REQ = 0x6107`
Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket.
`ApiSocketAddrType`	`ServerAddr`	The address of the remote socket to connect to.

10.1.9 API_SOCKET_BIND_CFM

Description: This mail is used to confirm API_SOCKET_BIND_REQ
IntOutPrimitive: `API_SOCKET_BIND_CFM = 0x6108`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`ApiSocketHandleType`	`Handle`	Handle to the socket.

10.1.10 API_SOCKET_LISTEN_REQ

Description: This mail is used by the application to start listening for incoming connections to the socket specified.
IntInPrimitive: `API_SOCKET_LISTEN_REQ = 0x6109`
Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket.

10.1.11 API_SOCKET_LISTEN_CFM

Description: This mail is used to confirm API_SOCKET_LISTEN_REQ
IntOutPrimitive: `API_SOCKET_LISTEN_CFM = 0x610A`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`ApiSocketHandleType`	`Handle`	Handle to the socket.

10.1.12 API_SOCKET_CONNECT_IND

Description: This mail is used to inform the application when a client has requested a connection..
IntOutPrimitive: `API_SOCKET_CONNECT_IND = 0x610B`
Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket.

10.1.13 API_SOCKET_ACCEPT_REQ

Description: This mail is used by the server application to accept an incoming client connection.
IntInPrimitive: `API_SOCKET_ACCEPT_REQ = 0x610C`
Parameters:

Type	Name	Description
ApiSocketHandleType	Handle	Handle to the socket.

10.1.14 API_SOCKET_ACCEPT_CFM

Description: This mail is used to confirm API_SOCKET_ACCEPT_REQ
IntOutPrimitive: `API_SOCKET_ACCEPT_CFM = 0x610D`
Parameters:

Type	Name	Description
RsStatusType	Status	RSS_SUCCESS if the request succeeded.
ApiSocketHandleType	Handle	Handle to the server socket
ApiSocketHandleType	ClientHandle	Handle to new socket created for the connection to the client.
ApiSocketAddrType	ClientAddr	The address info (port, IP) of the client connected.

10.2 Data interface

The API's used for data transfer are described in this section.

10.2.1 API_SOCKET_SEND_REQ

Description: This mail is used by the application to send a data buffer via the stream based (e.g. TCP) socket specified. The application must not free/reuse the buffer to API_SOCKET_SEND_CFM has been received.
IntInPrimitive: `API_SOCKET_SEND_REQ = 0x6120`
Parameters:

Type	Name	Description
ApiSocketHandleType	Handle	Handle to the socket.
rsuint8*	BufferPtr	Pointer to buffer holding the data to be sent.
rsuint16	BufferLength	The size in byte of the buffer.
rsuint32	Flags	TBD. Set to 0.

10.2.2 API_SOCKET_SEND_TO_REQ

Description: This mail is used by the application to send a data gram (e.g. UDP) via the socket specified. The application must not free/reuse the buffer to API_SOCKET_SEND_CFM has been received.
IntInPrimitive: `API_SOCKET_SEND_TO_REQ = 0x6121`
Parameters:

Type	Name	Description
ApiSocketHandleType	Handle	Handle to the socket.
rsuint8*	BufferPtr	Pointer to buffer holding the data to be sent.
rsuint16	BufferLength	The size in byte of the buffer.
rsuint32	Flags	TBD. Set to 0.
ApiSocketAddrType	Addr	The address of the remote device to send to.

10.2.3 API_SOCKET_SEND_CFM

Description: This mail is used to confirm both the API_SOCKET_SEND_REQ and the API_SOCKET_SEND_TO_REQ.
IntOutPrimitive: `API_SOCKET_SEND_CFM = 0x6122`
Parameters:

Type	Name	Description
RsStatusType	Status	RSS_SUCCESS if the request succeeded.
ApiSocketHandleType	Handle	Handle to the socket closed
rsuint8*	BufferPtr	The buffer sent.

10.2.4 API_SOCKET_RECEIVE_IND

Description: This mail is used to inform the application when a data packet has been received on the socket specified

IntOutPrimitive: `API_SOCKET_RECEIVE_IND = 0x6123`

Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket closed
`rsuint8*`	`BufferPtr`	Pointer to buffer holding the packet received.
`rsuint16`	`BufferLength`	The length of the packet in the receive buffer.

10.2.5 API_SOCKET_RECEIVE_FROM_IND

Description: This mail is used to inform the application when an UDP data packet has been received on the socket specified

IntOutPrimitive: `API_SOCKET_RECEIVE_FROM_IND = 0x6124`

Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket closed
`rsuint8*`	`BufferPtr`	Pointer to buffer holding the packet received.
`rsuint16`	`BufferLength`	The length of the packet in the receive buffer.
`ApiSocketAddrType`	`Addr`	The address of the remote device.

10.2.6 API_SOCKET_FREE_BUFFER_REQ

Description: This mail is used by the application to free the buffer holding the packet just received. This mail *MUST* be sent from the application for each API_SOCKET_RECEIVE_IND and API_SOCKET_RECEIVE_FROM_IND. It is not possible to receive data from the remote device if the receive buffer is not freed!

IntInPrimitive: `API_SOCKET_FREE_BUFFER_REQ = 0x6125`

Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket.
`rsuint8*`	`BufferPtr`	Pointer to receive buffer to free.

10.2.7 API_SOCKET_GET_TX_BUFFER_REQ

Description: This mail is used by the application to allocate a TX buffer. The application can copy the TX data directly to the buffer indicated by the CFM. The application sends the buffer by usage of API_SOCKET_SEND_REQ or API_SOCKET_SEND_TO_REQ once all data of the TX packet has been copied/generated to the TX buffer allocated. Wait with allocation of the TX buffer to it is needed as the TX buffer is a shared resource.

IntInPrimitive: `API_SOCKET_GET_TX_BUFFER_REQ = 0x6127`

Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket.
`rsuint16`	`BufferLength`	The size of the TX buffer that the application wants to allocate.

10.2.8 API_SOCKET_GET_TX_BUFFER_CFM

Description: This mail is used to confirm API_SOCKET_GET_TX_BUFFER_REQ.

IntOutPrimitive: `API_SOCKET_GET_TX_BUFFER_CFM = 0x6128`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`ApiSocketHandleType`	`Handle`	Handle to the socket
`rsuint8*`	`BufferPtr`	Pointer to the TX buffer allocated.
`rsuint16`	`BufferLength`	The actual size of the TX buffer allocated

10.2.9 API_SOCKET_EXT_DATA_WRITE_REQ

Description: This mail can be used to write data to a TX buffer from an external MCU (via e.g. the UART) where from it is not possible to write to the TX buffer directly. The size of the data buffer of this mail is limited to 200 bytes due to internal mail buffer constrains in the WiFi module. I.e. multiple API_SOCKET_EXT_DATA_WRITE_REQ's must be sent if the size of the TX packet is more than 200 bytes. The TX buffer must first be allocated by usage of API_SOCKET_GET_TX_BUFFER_REQ.

IntInPrimitive: `API_SOCKET_EXT_DATA_WRITE_REQ = 0x6129`

Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket.
`rsuint8*`	`BufferPtr`	Pointer to the TX buffer.
`rsuint16`	`Offset`	Offset in the TX buffer where to the data is written
`rsuint16`	`DataLength`	The size of Data[]
`rsuint8`	`Data[1]`	The TX data. Max size is 200.

10.2.10 API_SOCKET_EXT_DATA_WRITE_CFM

Description: This mail is used to confirm API_SOCKET_EXT_DATA_WRITE_REQ.

IntOutPrimitive: `API_SOCKET_EXT_DATA_WRITE_CFM = 0x612A`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`ApiSocketHandleType`	`Handle`	Handle to the socket closed
`rsuint8*`	`BufferPtr`	Pointer to the TX buffer.
`rsuint16`	`Offset`	The Offset from the req.

10.2.11 API_SOCKET_EXT_DATA_READ_REQ

Description: This mail can be used to read data from a RX buffer from an external MCU (via e.g. the UART) where from it is not possible to read from the RX buffer directly. The size of the data buffer of the CFM mail is limited to 200 bytes due to internal mail buffer constrains in the WiFi module. I.e. multiple API_SOCKET_EXT_DATA_READ_REQ's must be sent if the size of the RX packet is more than 200 bytes. The RX buffer must be freed by usage of API_SOCKET_FREE_BUFFER_REQ once all data has been read.

IntInPrimitive: `API_SOCKET_EXT_DATA_READ_REQ = 0x612B`

Parameters:

Type	Name	Description
`ApiSocketHandleType`	`Handle`	Handle to the socket.
`rsuint8*`	`BufferPtr`	Pointer to the RX buffer.
`rsuint16`	`Offset`	Offset in the RX buffer where from data is read.
`rsuint16`	`ReqLength`	The number of bytes requested.

10.2.12 API_SOCKET_EXT_DATA_READ_CFM

Description: This mail is used to confirm API_SOCKET_EXT_DATA_READ_REQ.

IntOutPrimitive: `API_SOCKET_EXT_DATA_READ_CFM = 0x612C`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`ApiSocketHandleType`	`Handle`	Handle to the socket closed
`rsuint8*`	`BufferPtr`	Pointer to the RX buffer.
`rsuint16`	`Offset`	The Offset from the req.
`rsuint16`	`DataLength`	The size of Data[].
`rsuint8`	`Data[1]`	The RX data.

10.3 Type definitions

10.3.1 Includes

Description: Include of API configuration.
C-syntax:

```
#include <Api/ApiCfg.h>
```

10.3.2 ApiSocketHandleType

Description: This type is used to hold a handle to a socket.
C-syntax:

```
typedef rsuint32 ApiSocketHandleType;
```

10.3.3 ApiSocketPortNumberType

Description: This type is used to hold a port number.
C-syntax:

```
typedef rsuint32 ApiSocketPortNumberType;
```

10.3.4 ApiSocketDomainType

Description: This enum defines the protocol domain/family.
C-syntax:

```
typedef enum RSENUMTAG(ApiSocketDomainTypeTag)
{
   ASD_AF_INET                        = 0x00,   IPv4
   ASD_AF_INET6                       = 0x01,   IPv6
   ASD_MAX                                      Invalid
} RSENUM8(ApiSocketDomainType);
```

10.3.5 ApiSocketTypeType

Description: This enum defines the type of a socket.
C-syntax:

```
typedef enum RSENUMTAG(ApiSocketTypeTypeTag)
{
   AST_STREAM                         = 0x00,   TCP
   AST_DGRAM                          = 0x01,   UDP
   AST_MAX                                      Invalid
} RSENUM8(ApiSocketTypeType);
```

10.3.6 ApiSocketProtocolType

Description: This enum defines the protocols.
C-syntax:

```
typedef enum RSENUMTAG(ApiSocketProtocolTypeTag)
{
   ASP_DEFAULT                        = 0x00,   Use the default protocol.
   ASP_MAX                                      Invalid
} RSENUM8(ApiSocketProtocolType);
```

10.3.7 ApiSocketAddr4Type

Description: This struct is used to hold the IPv4 address.

C-syntax:

```
typedef struct ApiSocketAddr4Type
{
   rsuint32                         Addr;       Binary IPv4 address
} ApiSocketAddr4Type;
```

10.3.8 ApiSocketAddr6Type

Description: This struct is used to hold the IPv6 address.

C-syntax:

```
typedef struct ApiSocketAddr6Type
{
   rsuint8                         Addr[16];    IPv6 address.
   rsuint32                        FlowInfo;    IPv6 flow information.
   rsuint32                        ScopeId;     Set of interfaces for a scope.
} ApiSocketAddr6Type;
```

10.3.9 ApiSocketAddrUnionType

Description: This struct is used to hold either an IPv4 or an IPv6 address.

C-syntax:

```
typedef union ApiSocketAddrUnionType
{
   ApiSocketAddr4Type               V4;         IPv4 address
   ApiSocketAddr6Type               V6;         IPv4 address
} ApiSocketAddrUnionType;
```

10.3.10 ApiSocketAddrType

Description: This struct is used to hold information about the address associated with a socket.

C-syntax:

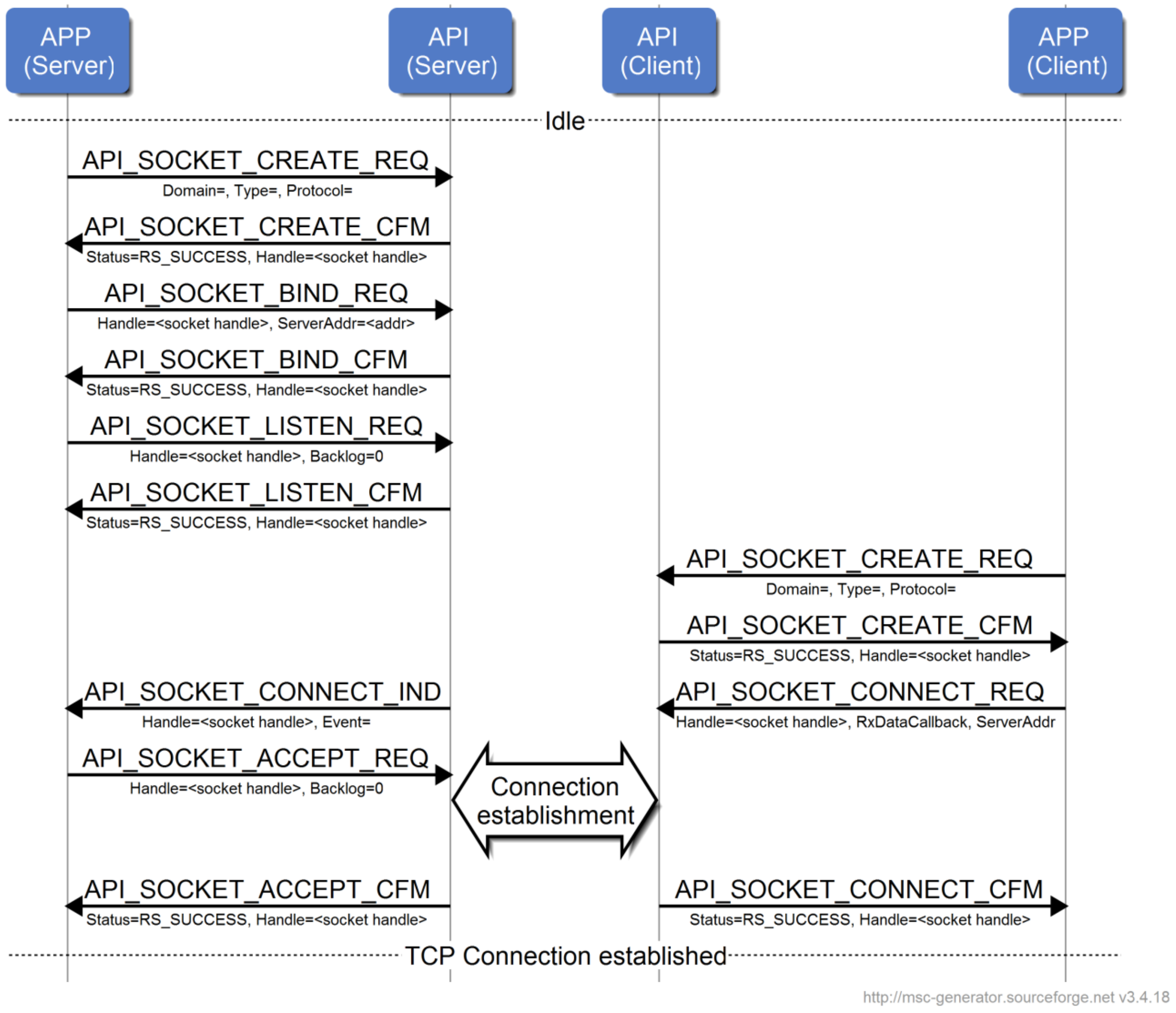
```
typedef struct ApiSocketAddrType
{
   ApiSocketDomainType              Domain;     IPv4 or IPv6
   ApiSocketPortNumberType          Port;       Remote port
   ApiSocketAddrUnionType           Ip;         IP v4/6 address
} ApiSocketAddrType;
```

11 Sequence charts

The usage of the API is described in more details by usage of MSC's in this section.

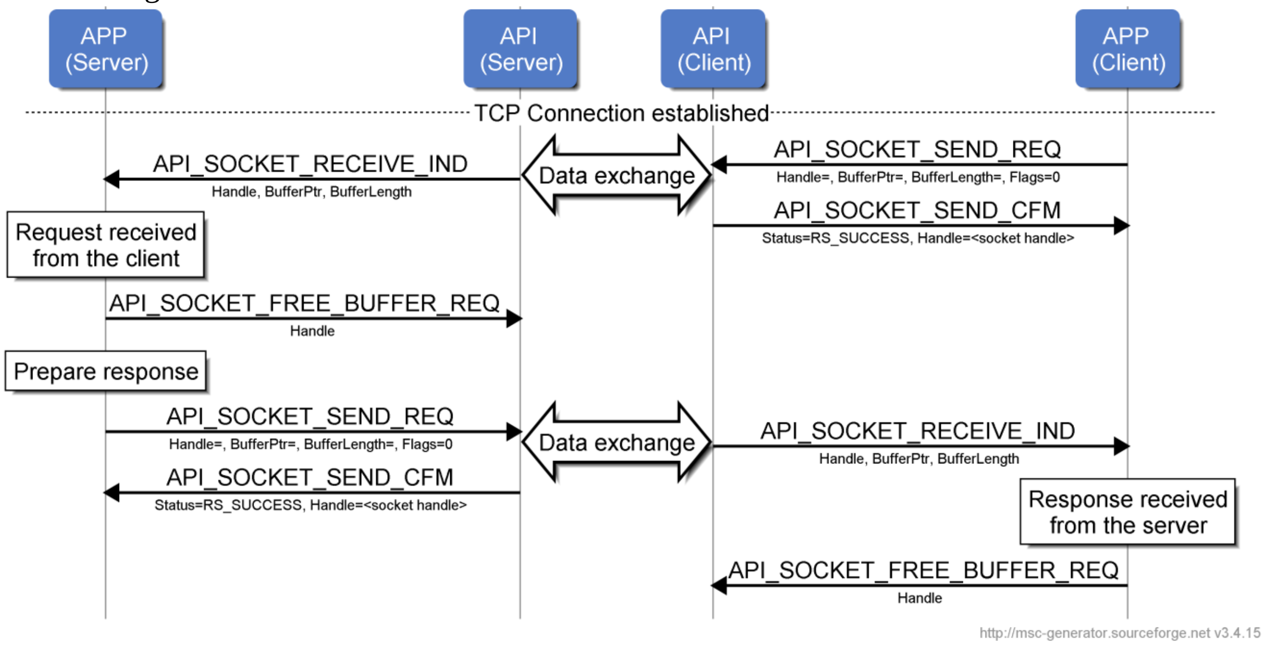
11.1 TCP connection establishment

The following MSC illustrates how the Socket API is used on both the client and the server to create sockets and establish a connection from the client to the server.


APP (Server)
API (Server)
API (Client)
APP (Client)
Idle
API_SOCKET_CREATE_REQ
Domain=, Type=, Protocol=
API_SOCKET_CREATE_CFM
Status=RS_SUCCESS, Handle=<socket handle>
API_SOCKET_BIND_REQ
Handle=<socket handle>, ServerAddr=<addr>
API_SOCKET_BIND_CFM
Status=RS_SUCCESS, Handle=<socket handle>
API_SOCKET_LISTEN_REQ
Handle=<socket handle>, Backlog=0
API_SOCKET_LISTEN_CFM
Status=RS_SUCCESS, Handle=<socket handle>
API_SOCKET_CREATE_REQ
Domain=, Type=, Protocol=
API_SOCKET_CREATE_CFM
Status=RS_SUCCESS, Handle=<socket handle>
API_SOCKET_CONNECT_IND
Handle=<socket handle>, Event=
API_SOCKET_CONNECT_REQ
Handle=<socket handle>, RxDataCallback, ServerAddr
API_SOCKET_ACCEPT_REQ
Handle=<socket handle>, Backlog=0
Connection establishment
API_SOCKET_ACCEPT_CFM
Status=RS_SUCCESS, Handle=<socket handle>
API_SOCKET_CONNECT_CFM
Status=RS_SUCCESS, Handle=<socket handle>
TCP Connection established
http://msc-generator.sourceforge.net v3.4.18

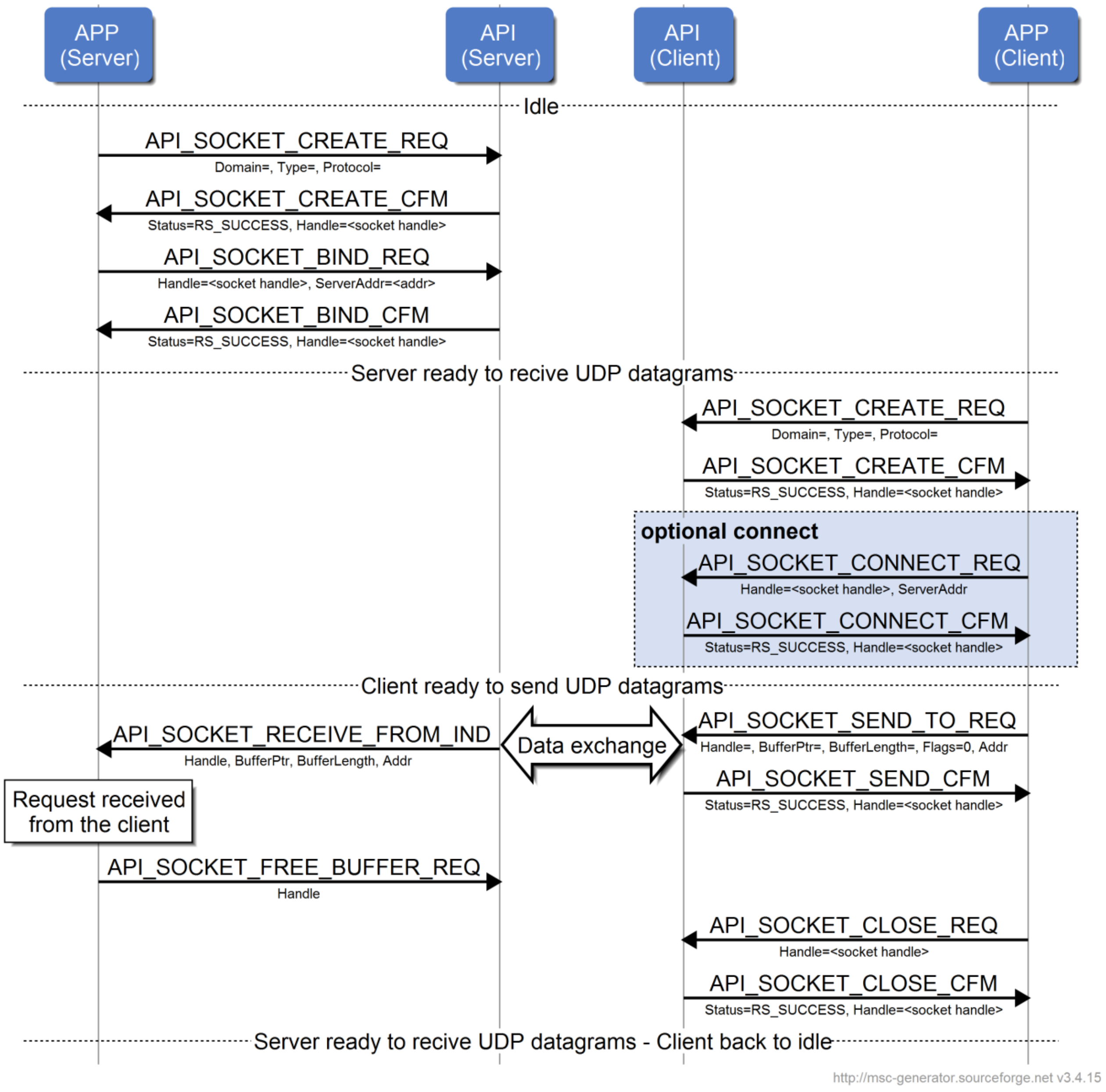
11.2 TCP data exchange

The following MSC illustrates how the Socket API is used to send and receive data.


APP (Server)
API (Server)
API (Client)
APP (Client)
TCP Connection established
API_SOCKET_SEND_REQ
Handle=, BufferPtr=, BufferLength=, Flags=0
Data exchange
API_SOCKET_RECEIVE_IND
Handle, BufferPtr, BufferLength
API_SOCKET_SEND_CFM
Status=RS_SUCCESS, Handle=<socket handle>
Request received from the client
API_SOCKET_FREE_BUFFER_REQ
Handle
Prepare response
API_SOCKET_SEND_REQ
Handle=, BufferPtr=, BufferLength=, Flags=0
Data exchange
API_SOCKET_RECEIVE_IND
Handle, BufferPtr, BufferLength
API_SOCKET_SEND_CFM
Status=RS_SUCCESS, Handle=<socket handle>
Response received from the server
API_SOCKET_FREE_BUFFER_REQ
Handle
http://msc-generator.sourceforge.net v3.4.15

11.3 UDP data exchange

The following MSC illustrates how the Socket API can be used to send a UDP datagram from a client to a server.


APP (Server)
API (Server)
API (Client)
APP (Client)
Idle
API_SOCKET_CREATE_REQ
Domain=, Type=, Protocol=
API_SOCKET_CREATE_CFM
Status=RS_SUCCESS, Handle=<socket handle>
API_SOCKET_BIND_REQ
Handle=<socket handle>, ServerAddr=<addr>
API_SOCKET_BIND_CFM
Status=RS_SUCCESS, Handle=<socket handle>
Server ready to recive UDP datagrams
API_SOCKET_CREATE_REQ
Domain=, Type=, Protocol=
API_SOCKET_CREATE_CFM
Status=RS_SUCCESS, Handle=<socket handle>
optional connect
API_SOCKET_CONNECT_REQ
Handle=<socket handle>, ServerAddr
API_SOCKET_CONNECT_CFM
Status=RS_SUCCESS, Handle=<socket handle>
Client ready to send UDP datagrams
API_SOCKET_RECEIVE_FROM_IND
Handle, BufferPtr, BufferLength, Addr
Data exchange
API_SOCKET_SEND_TO_REQ
Handle=, BufferPtr=, BufferLength=, Flags=0, Addr
API_SOCKET_SEND_CFM
Status=RS_SUCCESS, Handle=<socket handle>
Request received from the client
API_SOCKET_FREE_BUFFER_REQ
Handle
API_SOCKET_CLOSE_REQ
Handle=<socket handle>
API_SOCKET_CLOSE_CFM
Status=RS_SUCCESS, Handle=<socket handle>
Server ready to recive UDP datagrams - Client back to idle
http://msc-generator.sourceforge.net v3.4.15

12 Document Info

12.1 History

Revision	Author	Issue Date	Comments
0.01	LKA	17-Jan-12	Initial Revision.
0.02	LKA	23-Feb-12	General update.
0.03	LKA	4-Jun-12	Added IPv6 support.
0.04	TKP	26-Jun-12	Minor Update.
0.05	LKA	16-Aug-13	Added support for get of DNS server addr from DHCP sserver.

12.2 References

12.3 Terms & Abbreviations

13 API Specification

13.1 IP Address management

The API's for IP Address management are described in this section.

13.1.1 API_IPV4_CONFIG_REQ

Description:	This mail is used by the application to enable DHCP or set static IP.
IntInPrimitive:	`API_IPV4_CONFIG_REQ = 0x6180`

Parameters:

Type	Name	Description
`ApiIpConfigModeType`	`Mode`	Query, use DHCP, or set static IP
`ApiIpV4AddressType`	`Address`	The IP address used if DHCP is disabled.
`ApiIpV4AddressType`	`SubnetMask`	The subnet mask used if DHCP is disabled.
`ApiIpV4AddressType`	`Gateway`	The address of the default gateway if DHCP is disabled.

13.1.2 API_IPV4_CONFIG_CFM

Description:	This mail is used to confirm API_IPV4_CONFIG_REQ
IntOutPrimitive:	`API_IPV4_CONFIG_CFM = 0x6181`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`ApiIpV4AddressType`	`Address`	The IP address.
`ApiIpV4AddressType`	`SubnetMask`	The subnet mask.
`ApiIpV4AddressType`	`Gateway`	The address of the default gateway.
`ApiIpV4AddressType`	`PrimDnsServer`	The address of the primary DNS server.
`ApiIpV4AddressType`	`SecDnsServer`	The address of the secondary DNS server.

13.1.3 API_IPV6_QUERY_REQ

Description:	This mail is used by the application to obtain IPv6 address information.
IntInPrimitive:	`API_IPV6_QUERY_REQ = 0x6182`
Parameters:	No parameters

13.1.4 API_IPV6_QUERY_CFM

Description:	This mail is used to confirm API_IPV6_QUERY_REQ
IntOutPrimitive:	`API_IPV6_QUERY_CFM = 0x6183`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`ApiIpV6AddressType`	`GlobalAddress`	The IPv6 global address. All 0 values indicate that the address is invalid/unknown.
`ApiIpV6AddressType`	`LocalAddress`	The IPv6 local address. All 0 values indicate that the address is invalid/unknown.
`ApiIpV6AddressType`	`Gateway`	The IPv6 address of the default gateway. All 0 values indicate that the address is invalid/unknown.
`ApiIpV6AddressType`	`LinkAddrExtd`	The IPv6 Link Local address for Logo. All 0 values indicate that the address is invalid/unknown.
`rsuint32`	`LinkPrefix`	The IPv6 link prefix.
`rsuint32`	`GlobalPrefix`	The IPv6 global prefix.
`rsuint32`	`GatewayPrefix`	The IPv6 default gateway prefix.
`rsuint32`	`LinkAddrExtdPrefix`	The IPv6 link local prefix.

13.2 Ping

The Ping API can be used to ping a remote IP device.

13.2.1 API_IPV4_PING_REQ

Description: This mail is used by the application to ping a remote device.
IntInPrimitive: `API_IPV4_PING_REQ = 0x61A0`
Parameters:

Type	Name	Description
`ApiIpV4AddressType`	`Address`	The IP address of the device to ping.

13.2.2 API_IPV4_PING_CFM

Description: This mail is used to confirm API_IPV4_PING_REQ
IntOutPrimitive: `API_IPV4_PING_CFM = 0x61A1`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

13.2.3 API_IPV6_PING_REQ

Description: This mail is used by the application to ping a remote device.
IntInPrimitive: `API_IPV6_PING_REQ = 0x61A2`
Parameters:

Type	Name	Description
`ApiIpV6AddressType`	`Address`	The IP address of the device to ping.

13.2.4 API_IPV6_PING_CFM

Description: This mail is used to confirm API_IPV6_PING_REQ
IntOutPrimitive: `API_IPV6_PING_CFM = 0x61A3`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

13.3 Type definitions

13.3.1 Includes

Description: Include of API configuration.
C-syntax:

```
#include <Api/ApiCfg.h>
```

13.3.2 ApiIpConfigModeType

Description: This enum defines the different modes supported for the IP config command.
C-syntax:

```
typedef enum ApiIpConfigModeTypeTag
{
   AICM_QUERY                     = 0x00,

   AICM_USE_DHCP                  = 0x01,

   AICM_SET_STATIC_IP             = 0x02

} RSENUM8(ApiIpConfigModeType);
```

13.3.3 ApiIpV4AddrType

Description: This type is used to hold an IP V4 address.

C-syntax:

```
typedef rsuint32 ApiIpV4AddressType;
```

13.3.4 ApiIpV6AddrType

Description: This type is used to hold an IP V6 address.

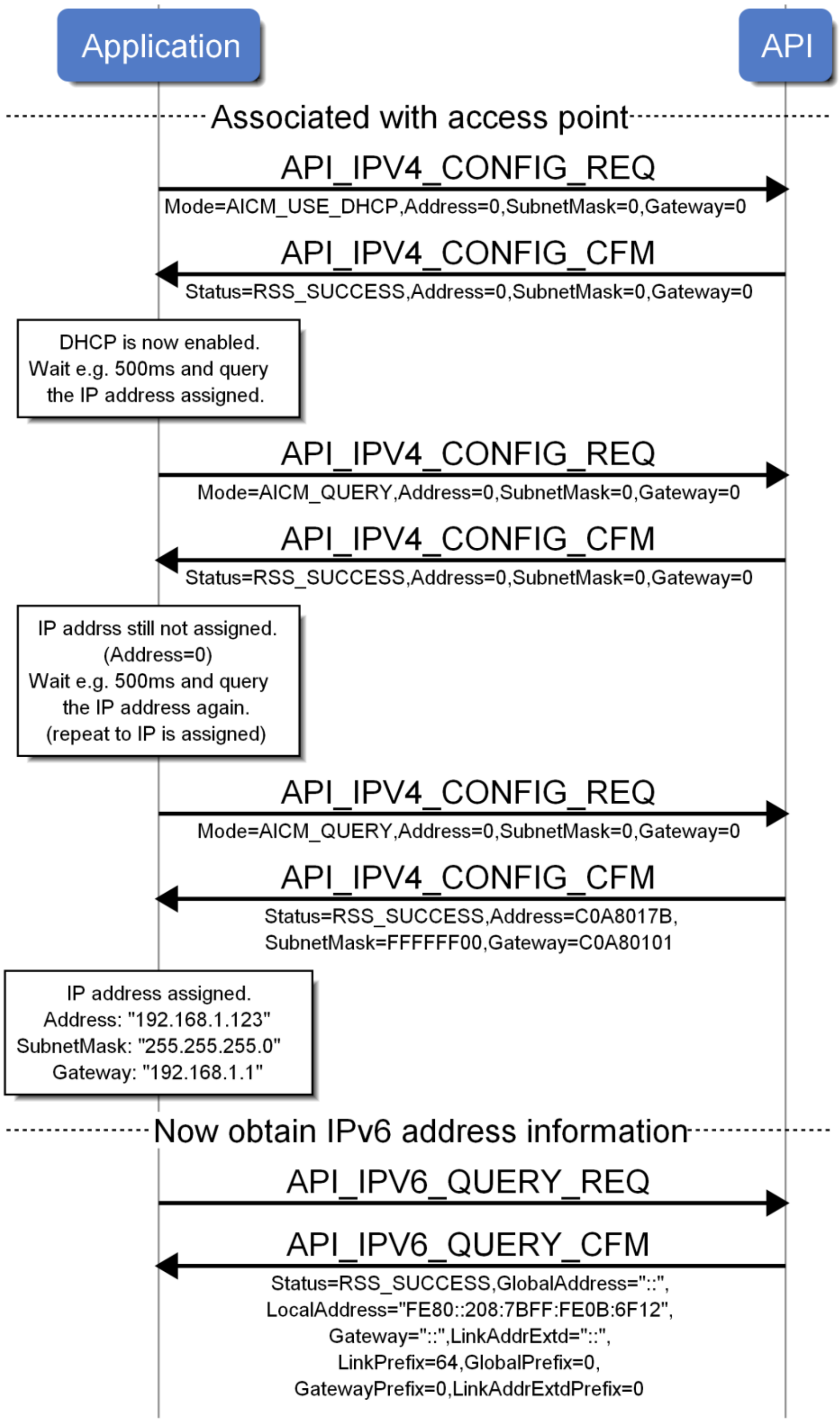
C-syntax:

```
typedef struct ApiIpV6AddressType
{
   rsuint8                        Addr[16];

} ApiIpV6AddressType;
```

14 Sequence charts

14.1 IPv4 DHCP config

The following MSC illustrates how the IpConfig API is used to enable DHCP and query the IP address assigned.


Application
API
Associated with access point
API_IPV4_CONFIG_REQ
Mode=AICM_USE_DHCP,Address=0,SubnetMask=0,Gateway=0
API_IPV4_CONFIG_CFM
Status=RSS_SUCCESS,Address=0,SubnetMask=0,Gateway=0
DHCP is now enabled. Wait e.g. 500ms and query the IP address assigned.
API_IPV4_CONFIG_REQ
Mode=AICM_QUERY,Address=0,SubnetMask=0,Gateway=0
API_IPV4_CONFIG_CFM
Status=RSS_SUCCESS,Address=0,SubnetMask=0,Gateway=0
IP addrss still not assigned. (Address=0) Wait e.g. 500ms and query the IP address again. (repeat to IP is assigned)
API_IPV4_CONFIG_REQ
Mode=AICM_QUERY,Address=0,SubnetMask=0,Gateway=0
API_IPV4_CONFIG_CFM
Status=RSS_SUCCESS,Address=C0A8017B, SubnetMask=FFFFFF00,Gateway=C0A80101
IP address assigned. Address: "192.168.1.123" SubnetMask: "255.255.255.0" Gateway: "192.168.1.1"
Now obtain IPv6 address information
API_IPV6_QUERY_REQ
API_IPV6_QUERY_CFM
Status=RSS_SUCCESS,GlobalAddress="::", LocalAddress="FE80::208:7BFF:FE0B:6F12", Gateway="::",LinkAddrExtd="::", LinkPrefix=64,GlobalPrefix=0, GatewayPrefix=0,LinkAddrExtdPrefix=0

15 Document Info

15.1 History

Revision	Author	Issue Date	Comments
0.01	LKA	20-Mar-12	Initial Revision.
0.02	TKP	8-May-12	Resolved some minor issues and IpV6Host flag to API_DNS_CLIENT_RESOLVE_REQ message.
0.03	TKP	1-Jun-12	Added Message Sequence Chart.
0.04	TKP	26-Jun-12	Minor update.
0.05	TKP	10-Aug-12	Added API_DNS_CLIENT_RESOLVE_CNAME_* messages.

15.2 References

15.3 Terms & Abbreviations

16 API Specification

16.1 DNS client interface

The API used for setting up DNS server addresses and resolving of host names to IP addresses are described in this section.

16.1.1 API_DNS_CLIENT_ADD_SERVER_REQ

Description: This mail is used by the application to add a DNS server. `IpV4` should be set to zero when adding an `IpV6` server address.

IntInPrimitive: `API_DNS_CLIENT_ADD_SERVER_REQ = 0x6220`

Parameters:

Type	Name	Description
`ApiIpV4AddressType`	`IpV4`	IPv4 address
`ApiIpV6AddressType`	`IpV6`	IPv6 address

16.1.2 API_DNS_CLIENT_ADD_SERVER_CFM

Description: This mail is used to confirm API_DNS_CLIENT_ADD_SERVER_REQ

IntOutPrimitive: `API_DNS_CLIENT_ADD_SERVER_CFM = 0x6221`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

16.1.3 API_DNS_CLIENT_RESOLVE_REQ

Description: This mail is used by the application to request the IP address resolution for the host with the host name specified.

IntInPrimitive: `API_DNS_CLIENT_RESOLVE_REQ = 0x6222`

Parameters:

Type	Name	Description
`rsbool`	`IpV6Host`	TRUE if host is IpV6.
`rsuint8`	`HostNameLength`	The length in bytes of the HostName.
`rsuint8`	`HostName[1]`	The host name to resolve.

16.1.4 API_DNS_CLIENT_RESOLVE_CFM

Description: This mail is used to confirm API_DNS_CLIENT_RESOLVE_REQ

IntOutPrimitive: `API_DNS_CLIENT_RESOLVE_CFM = 0x6223`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`ApiIpV4AddressType`	`IpV4`	IPv4 address
`ApiIpV6AddressType`	`IpV6`	IPv6 address
`rsuint8`	`HostNameLength`	The length in bytes of the HostName
`rsuint8`	`HostName[1]`	The host name to resolve.

16.1.5 API_DNS_CLIENT_RESOLVE_CNAME_REQ

Description: This mail is used by the application to request a canonical name resolution for the host with the host name specified.

IntInPrimitive: `API_DNS_CLIENT_RESOLVE_CNAME_REQ = 0x6224`

Parameters:

Type	Name	Description
`rsuint8`	`HostNameLength`	The length in bytes of the HostName.
`rsuint8`	`HostName[1]`	The host name to resolve.

16.1.6 API_DNS_CLIENT_RESOLVE_CNAME_CFM

Description: This mail is used to confirm API_DNS_CLIENT_RESOLVE_CNAME_REQ
IntOutPrimitive: `API_DNS_CLIENT_RESOLVE_CNAME_CFM = 0x6225`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint8`	`AliasNameLength`	The length in bytes of the AliasName
`rsuint8`	`AliasName[1]`	The alias name resolved.

16.2 Type definitions

16.2.1 Includes

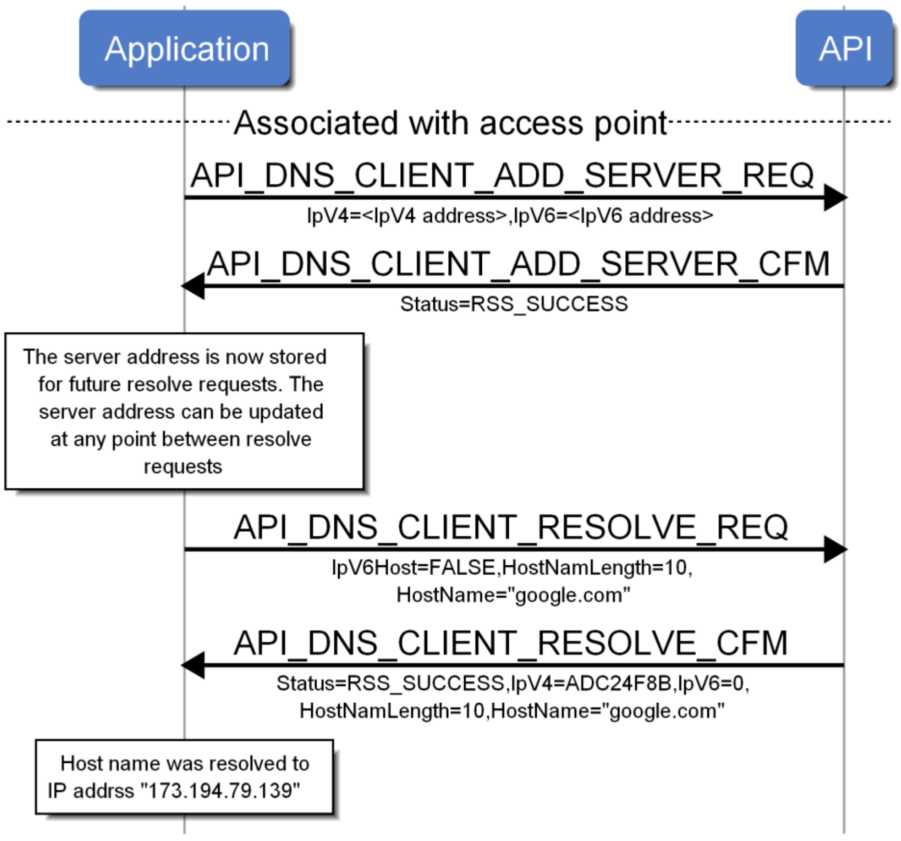
Description: Include of API configuration.
C-syntax:

```
#include <Iot/Net/Api/IpConfig/ApiIpConfig.h>
```

17 Sequence charts

17.1 DNS Client

The following MSC illustrates how to add a DNS server and then resolve a host name to an IpV4 address.


Application
API
Associated with access point
API_DNS_CLIENT_ADD_SERVER_REQ
IpV4=<IpV4 address>,IpV6=<IpV6 address>
API_DNS_CLIENT_ADD_SERVER_CFM
Status=RSS_SUCCESS
The server address is now stored for future resolve requests. The server address can be updated at any point between resolve requests
API_DNS_CLIENT_RESOLVE_REQ
IpV6Host=FALSE,HostNamLength=10, HostName="google.com"
API_DNS_CLIENT_RESOLVE_CFM
Status=RSS_SUCCESS,IpV4=ADC24F8B,IpV6=0, HostNamLength=10,HostName="google.com"
Host name was resolved to IP addrss "173.194.79.139"

HTTP API

18 Document Info

18.1 History

Revision	Author	Issue Date	Comments
0.01	LKA	4-Feb-13	Initial Revision.

18.2 References

18.3 Terms & Abbreviations

19 API Specification

19.1 HTTP server interface

The API used for HTTP server management.

19.1.1 API_HTTP_SERVER_INIT_REQ

Description:	This mail is used to initialize a HTTP server session that accepts connections for the port specified.
IntInPrimitive:	`API_HTTP_SERVER_INIT_REQ = 0x6240`

Parameters:

Type	**Name**	**Description**
`rsuint32`	`Port`	The port number used for this session.
`AhCallbackType`	`Callback`	Pointer to call back function used to pass HTTP request message data to the application when a HTTP request message is received from a HTTP client.

19.1.2 API_HTTP_SERVER_INIT_CFM

Description:	This mail is used to conform API_HTTP_SERVER_INIT_REQ.
IntOutPrimitive:	`API_HTTP_SERVER_INIT_CFM = 0x6241`

Parameters:

Type	**Name**	**Description**
`RsStatusType`	`Status`	Indicates whether the server init succeeded or not.
`rsuint32`	`Port`	The port number.
`rsuint32`	`Instance`	The instance id to be used in all further requests.

19.1.3 API_HTTP_SERVER_CONNECT_IND

Description:	This mail is sent to the application when a HTTP client connects to the server.
IntOutPrimitive:	`API_HTTP_SERVER_CONNECT_IND = 0x6244`

Parameters:

Type	**Name**	**Description**
`rsuint32`	`Instance`	The instance.
`rsuint32`	`Port`	The port number used.

19.1.4 API_HTTP_SERVER_SEND_RESPONSE_REQ

Description:	This mail is used to send a HTTP response message to a HTTP client.
IntInPrimitive:	`API_HTTP_SERVER_SEND_RESPONSE_REQ = 0x6246`

Parameters:

Type	**Name**	**Description**
`rsuint32`	`Instance`	The instance.
`rsuint16`	`StatusCode`	The HTTP status code of the HTTP response.
`rschar*`	`ReasonPtr`	Pointer to '\0' terminated reason phrase string.
`rsuint32`	`BodyDataLength`	
`rsuint8*`	`DataPtr`	
`AhAddHeaderCbType`	`AddHeaderCb`	
`AhAddBodyCbType`	`AddBodyCb`	

19.1.5 API_HTTP_SERVER_SEND_RESPONSE_CFM

Description: This mail is sent to the application when the entire HTTP response message has been retrieved by the client or if an error is detected.

IntOutPrimitive: `API_HTTP_SERVER_SEND_RESPONSE_CFM = 0x6247`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	Status
`rsuint32`	`Instance`	The instance.

19.1.6 API_HTTP_SERVER_ADD_RESOURCE_REQ

Description: This mail is used by the application to add a resource to the server. A resource can be deleted by sending this mail with the CallBack parameter set to NULL.

IntInPrimitive: `API_HTTP_SERVER_ADD_RESOURCE_REQ = 0x6248`

Parameters:

Type	Name	Description
`rsuint32`	`Instance`	The instance.
`rschar*`	`PathPtr`	Pinter to '\0' terminated string holding the path used to request the resource. The path string must be available as long as the server is running, as the server stores the pointer to the string only.
`AhResourceCbType`	`CallBack`	Pointer to the function that will generate/copy the resource data when request by the client (e.g. WEB browser on remote client). This function is called instead of the call back function specified in API_HTTP_SERVER_INIT when the client requests the resource identified by the PathPtr.

19.1.7 API_HTTP_SERVER_ADD_RESOURCE_CFM

Description: This mail is used to confirm API_HTTP_SERVER_ADD_RESOURCE_REQ.

IntOutPrimitive: `API_HTTP_SERVER_ADD_RESOURCE_CFM = 0x6249`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint32`	`Instance`	The instance.

19.2 HTTP client interface

The API used for HTTP client implementation.

19.2.1 API_HTTP_CLIENT_INIT_REQ

Description: This mail is used to initialize a HTTP client session and connect to the HTTP server specified.

IntInPrimitive: `API_HTTP_CLIENT_INIT_REQ = 0x6260`

Parameters:

Type	Name	Description
`rsuint32`	`Port`	The port number used for this session.
`rschar*`	`HostPtr`	Pointer to '\0' terminated sting holding the name of the host to connect to. The host name string must be available as long as the HTTP session is active.

19.2.2 API_HTTP_CLIENT_INIT_CFM

Description: This mail is used to conform API_HTTP_CLIENT_INIT_REQ.
IntOutPrimitive: `API_HTTP_CLIENT_INIT_CFM = 0x6261`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	Indicates whether the server init succeeded or not.
`rsuint32`	`Port`	The port number.
`rsuint32`	`Instance`	The instance id to be used in all further requests.

19.2.3 API_HTTP_CLIENT_SEND_REQ

Description: This mail is used to send a HTTP request message to a HTTP server and wait for the HTTP response message. A call back function is used to pass the headers and the data from the HTTP response back to the client application.
IntInPrimitive: `API_HTTP_CLIENT_SEND_REQ = 0x6264`
Parameters:

Type	Name	Description
`rsuint32`	`Instance`	The instance.
`AhCallbackType`	`Callback`	Pointer to call back function used to pass HTTP response message data to the application when the HTTP response message is received from the HTTP server.
`AhHttpMethodIdType`	`Method`	The HTTP method
`rschar*`	`UriPtr`	Pointer to '\0' terminated string holding the URI.
`rsuint32`	`BodyDataLength`	
`rsuint8*`	`DataPtr`	
`AhAddHeaderCbType`	`AddHeaderCb`	
`AhAddBodyCbType`	`AddBodyCb`	

19.2.4 API_HTTP_CLIENT_SEND_CFM

Description: This mail is sent to the application when the entire HTTP response message has been retrieved from the server or if an error is detected.
IntOutPrimitive: `API_HTTP_CLIENT_SEND_CFM = 0x6265`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	Status
`rsuint32`	`Instance`	The instance.
`rsuint16`	`HttpStatusCode`	The HTTP status code of the HTTP response.

19.3 Common HTTP client/server interface

The API used for HTTP client implementation.

19.3.1 API_HTTP_TERMINATE_REQ

Description: This mail is used to terminate the HTTP instance identified by the port specified.
IntInPrimitive: `API_HTTP_TERMINATE_REQ = 0x6270`
Parameters:

Type	Name	Description
`rsuint32`	`Instance`	The instance.

19.3.2 API_HTTP_TERMINATE_CFM

Description: This mail is sent to the application when the HTTP instance has been terminated.
IntOutPrimitive: `API_HTTP_TERMINATE_CFM = 0x6271`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	Status
`rsuint32`	`Instance`	The instance.

19.3.3 API_HTTP_TERMINATE_IND

Description: This mail is sent to the application when a HTTP client disconnects from the server.

IntOutPrimitive: `API_HTTP_TERMINATE_IND = 0x6272`

Parameters:

Type	Name	Description
`rsuint32`	`Instance`	The instance.

19.3.4 API_HTTP_RESUME_REQ

Description: This mail is used to resume the RX of a HTTP message that was suspended by returning TRUE from the data call back function (AhCallbackType).

IntInPrimitive: `API_HTTP_RESUME_REQ = 0x6273`

Parameters:

Type	Name	Description
`rsuint32`	`Instance`	The instance.

19.4 HTTP helper functions

19.4.1 HttpAddHeader

Description: This function is used to add a HTTP header to the HTTP message being built in the buffer specified by BufferPtr.

Function: `HttpAddHeader`

Return: `rsuint32` Number of bytes written to the buffer

Parameters:

Type	Name	Description
`rsuint8`	`*BufferPtr`	
`rsuint32`	`BufferLength`	
`rschar`	`*HeaderNamePtr`	
`rschar`	`*HeaderDataPtr`	

19.4.2 HttpSplitQueryString

Description: This function is used to split the query string of a HTTP request. Please note that this function modifies the query string by replacing the separator char '&' with a '\0' char and do %-decoding of the data strings of the query.

Function: `HttpSplitQueryString`

Return: `rsuint8` Actual number of query string elements stored.

Parameters:

Type	Name	Description
`rsuint8`	`Count`	Number of elements in the Query array.
`AhQueryDataType`	`Query[1]`	Pointer to array with ‘Count’ elements.
`rschar`	`*QueryStringPtr`	Pointer to the query string to split.

19.5 Type definitions

19.5.1 Includes

Description: Include of API configuration.

C-syntax:

```
#include <Api/ApiCfg.h>
```

19.5.2 AhHdrDataType

Description: This struct is used to hold HTTP header field id and value data ptr.

C-syntax:

```
typedef struct AhHdrDataTypeTag
{
   rsuint16                 HeaderLength;    The length of the header referenced by the HeaderPtr.
   rschar*                  HeaderPtr;       Pointer to block holding the HTTP header value/data.
   rsuint16                 DataLength;      The length of the data block referenced by the DataPtr.
   rschar*                  DataPtr;         Pointer to block holding the HTTP header value/data.
} AhHdrDataType;
```

19.5.3 AhDataType

Description: This struct is used to hold HTTP request message body data.

C-syntax:

```
typedef struct AhDataTypeTag
{
   rsuint32                 Offset;          The offset of this fragment in the complete body (file).
   rsuint32                 TotalLength;     The total length of the response data.
   rsuint32                 DataLength;      The length of the data block referenced by the DataPtr.
   rsuint8*                 DataPtr;         Pointer to block holding fragment of or complete HTTP
                                             body.
} AhDataType;
```

19.5.4 AhStringType

Description: This struct is used to hold HTTP path and query string data.

C-syntax:

```
typedef struct AhStringTypeTag
{
   rsuint16                 StringLength;    The length of the data block referenced by the DataPtr.
   rsuint8*                 StringPtr;       Pointer to string.
} AhStringType;
```

19.5.5 AhCbDataType

Description: This union is used to callback data.

C-syntax:

```
typedef union AhCbDataTypeTag
{
   AhHdrDataType            Hdr;             CbId = AH_CB_HDR
   AhDataType               Data;            CbId = AH_CB_DATA,
   AhStringType             String;          CbId = AH_CB_PATH or AH_CB_QUERY_STRING
} AhCbDataType;
```

19.5.6 AhCbIdType

Description: This enum identifies the type of data passed by the call-back function when HTTP request message is decoded.

C-syntax:

```
typedef enum RSENUMTAG(AhCbDataIdType)
{
   AH_CB_MESSAGE_BEGIN            = 0x00,    New HTTP message detected
   AH_CB_MESSAGE_END              = 0x01,    End of HTTP message detected
   AH_CB_PATH                     = 0x02,    Path received
   AH_CB_QUERY_STRING             = 0x03,    Query string received
   AH_CB_HDR                      = 0x04,    HTTP header field/data
   AH_CB_DATA                     = 0x05,    Body data
   AH_CB_INVALID                  = 0x06     Invalid
} RSENUM8(AhCbIdType);
```

19.5.7 AhHttpMethodIdType

Description: This enum identifies the HTTP methods supported.

C-syntax:

```
typedef enum RSENUMTAG(AhHttpMethodIdType)
{
   AHM_UNKNOWN                        = 0x00,    Will be used for AH_CB_MESSAGE_BEGIN.
   AHM_DELETE                         = 0x01,
   AHM_GET                            = 0x02,
   AHM_HEAD                           = 0x03,
   AHM_POST                           = 0x04,
   AHM_PUT                            = 0x05,
   AHM_INVALID                        = 0x06     Invalid
} RSENUM8(AhHttpMethodIdType);
```

19.5.8 AhCallbackType

Description: This type is used to hold a pointer to the call-back function used to pass HTTP request/response data to the application.

Function: `AhCallbackType`

Return: `rsbool` RX data suspend flag. Return TRUE if the application RX data buffer is full. This will stop the HTTP task from receiving more data from the remote HTTP server/client. Return FALSE if the application is able to handle more data.

Parameters:

Type	Name	Description
`AhCbIdType`	`CbId`	Call back id
`AhCbDataType`	`*DataPtr`	Data pointer.
`AhHttpMethodIdType`	`HttpMethod`	The HTTP method of the HTTP request.
`rsuint32`	`Instance`	The instance.

19.5.9 AhResourceCbType

Description: This type is used to hold a pointer to the callback function used to handle requests to a particular resource. The resource data (can either be generated or copied) is sent directly from this callback function by usage of the API_HTTP_SERVER_SEND_RESPONSE_REQ mail. The HTTP server implementation generates a HTTP response message with status:

- 404 if the client requests an unknown resource (this callback will not be called),
- 501 if this callback returns RSS_NOT_SUPPORTED
- 500 if this callback returns RSS_FAILED.

Function: `AhResourceCbType`

Return: `RsStatusType` This function should return RSS_SUCCESS if it generated the resource data requested successfully, RSS_NOT_SUPPORTED if the HTTP method of the request is unsupported, or RSS_FAILED if the function fails to generate the data requested.

Parameters:

Type	Name	Description
`AhHttpMethodIdType`	`HttpMethod`	The HTTP method of the HTTP request.
`rschar`	`*PathPtr`	Pointer to '\0' terminated string holding the path to the resource requested.
`rschar`	`*QueryPtr`	Pointer to '\0' terminated string holding optional query string. NULL if no query string is sent from the client.
`rsuint32`	`Instance`	The instance.

19.5.10 AhAddHeaderCbType

Description: This type is used to hold a pointer to the call-back function used to generate and add HTTP header fields to the HTTP message being send.

Function: `AhAddHeaderCbType`

Return: `rsuint32` Number of bytes added

Parameters:

Type	Name	Description
`rsuint8`	`*BufferPtr`	
`rsuint32`	`BufferLength`	
`rsuint8*`	`DataPtr`	Copy from REQ mail.
`rsuint32`	`Instance`	The instance.

19.5.11 AhAddBodyCbType

Description: This type is used to hold a pointer to the call-back function used to append body data to the HTTP message being sent.

Function: `AhAddBodyCbType`

Return: `rsuint32` Number of bytes added

Parameters:

Type	Name	Description
`rsuint8`	`*BufferPtr`	
`rsuint32`	`BufferLength`	
`rsuint32`	`Offset`	
`rsuint8*`	`DataPtr`	Copy from REQ mail.
`rsuint32`	`Instance`	The instance.

19.5.12 AhQueryDataType

Description: This struct is used to hold pointer to field name and value data of one query string element.

C-syntax:

```
typedef struct AhQueryDataTypeTag
{
   rschar*                    FieldPtr;
   rschar*                    ValuePtr;
} AhQueryDataType;
```

Member	Description
`rschar* FieldPtr;`	Pointer to '\0' terminated string holding the filed name of a query string element.
`rschar* ValuePtr;`	Pointer to '\0' terminated string holding the value of a query string element.

20 Sequence charts

TBD

Atheros WiFi API

21 Document Info

21.1 References

21.2 History

Revision	Author	Issue Date	Comments
0.01	LKA	9-Jan-12	Initial Revision.
0.02	LKA	23-Feb-12	General update.
0.03	LKA	23-Apr-12	Description of API_WIFI_SET_KEEP_ALIVE_INTERVAL_REQ is updated. ApiWifiScanResultType updated to reflect changes made in AR4100 SW R2 GA release. Added API_WIFI_GET_VERSION_REQ/CFM.
0.04	LKA	7-May-12	Added API for: • setting the regulatory domain • setting the physical WiFi mode (b, g, or n) • read out of RSSI level
0.05	TKP	26-Jun-12	Added MSCs.
0.06	TKP	28-Jun-12	Minor update
0.07	LKA	16-Aug-12	Changed API_WIFI_SCAN_CFM to include pointer to scan result instead of a copy of the scan result. Done to make it possible to increase the max size of the scan result. Added ActiveScanTime and PassiveScanTime parameters to API_WIFI_SCAN_CTRL_REQ.
0.08	LKA	27-Aug-12	Suspend time changed from rsuint16 to rsuint32. Added API: • for update of AR4100 firmware, and • for SoftAP control.
0.09	LKA	8-Oct-12	Added API_WIFI_GET_SCAN_RESULT_REQ
0.10	LKA	18-Aug-13	Changed ApiWifiScanResultType + added a bit more description of the scan result members.
0.11	LKA	14-Nov-13	Added API_WIFI_AP_SET_DHCP_POOL_REQ/CFM

21.3 Terms & Abbreviations

AP	Access point
API	Application Programming Interface
SSID	Service Set Identifier

22 API Specification

The Atheros WiFi management API is described in this section. The WiFi management API is mail based.

22.1 Power management

The API's described in this section are used to control the power / low power mode of the WiFi module.

22.1.1 API_WIFI_POWER_CTRL_REQ

Description: This mail is used to power on/off the WiFi module.

IntInPrimitive: `API_WIFI_POWER_CTRL_REQ = 0x6000`

Parameters:

Type	Name	Description
`rsbool`	`PowerOn`	TRUE: power on the WiFi module. FALSE: power off the WiFi module.

22.1.2 API_WIFI_POWER_CTRL_CFM

Description: This mail is used to confirm API_WIFI_POWER_CTRL_REQ. The confirm is sent when the WiFi module is ready to handle more commands when the module is powered on.

IntOutPrimitive: `API_WIFI_POWER_CTRL_CFM = 0x6001`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.1.3 API_WIFI_READY_IND

Description: This mail is used to inform the application when the WiFi module is ready after power on..

IntOutPrimitive: `API_WIFI_READY_IND = 0x6002`

Parameters: No parameters

22.1.4 API_WIFI_POWER_SAVE_CTRL_REQ

Description: This mail is used to control the low power mode of the WiFi module.

IntInPrimitive: `API_WIFI_POWER_SAVE_CTRL_REQ = 0x6003`

Parameters:

Type	Name	Description
`ApiWifiPowerModeType`	`PowerMode`	Enable/disable of power save mode. The remaining parameters are ignored if power save is disabled.
`rsuint16`	`IdlePeriod`	Time in ms the WiFi device remains awake after RX/TX before going to sleep mode.
`rsuint8`	`PsPollNumber`	The number of PowerSavePoll (PS-poll) messages the device should send before notifying the AP it is awake.
`ApiWifiDtimPolicyType`	`DtimPolicy`	DTIM policy.
`ApiWifiTxWakeupPolicyType`	`TxWakeupPolicy`	TX wakeup policy.
`rsuint8`	`NumTxToWakeup`	Number of uplink frames in a beacon interval to transition to awake.

22.1.5 API_WIFI_POWER_SAVE_CTRL_CFM

Description: This mail is used to confirm API_WIFI_POWER_SAVE_CTRL_REQ

IntOutPrimitive: `API_WIFI_POWER_SAVE_CTRL_CFM = 0x6004`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.1.6 API_WIFI_SUSPEND_ENABLE_REQ

Description: This mail is used to enable/disable support for suspend mode. Suspend support must be enabled before connection establishment. Suspend support is enabled by default on power on.

IntInPrimitive: `API_WIFI_SUSPEND_ENABLE_REQ = 0x6005`

Parameters:

Type	Name	Description
`rsbool`	`Enable`	TRUE: Enable suspend; FALSE disable suspend.

22.1.7 API_WIFI_SUSPEND_ENABLE_CFM

Description: This mail is used to confirm API_WIFI_SUSPEND_ENABLE_REQ

IntOutPrimitive: `API_WIFI_SUSPEND_ENABLE_CFM = 0x6006`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsbool`	`Enabled`	TRUE: Suspend is enabled; FALSE suspend is disabled.

22.1.8 API_WIFI_SUSPEND_REQ

Description: This mail is used to request the WiFi chip to suspend. Please note that the normal power save mode will be disabled when suspend is requested. Power save mode can be enabled again when API_WIFI_RESUME_IND is received if desired.

IntInPrimitive: `API_WIFI_SUSPEND_REQ = 0x6007`

Parameters:

Type	Name	Description
`rsuint32`	`SuspendTime`	The time in ms that the WiFi module should be suspended.

22.1.9 API_WIFI_SUSPEND_CFM

Description: This mail is used to confirm API_WIFI_SUSPEND_REQ

IntOutPrimitive: `API_WIFI_SUSPEND_CFM = 0x6008`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint32`	`SuspendTime`	The time in ms that the WiFi module wants to be suspended.

22.1.10 API_WIFI_RESUME_IND

Description: This mail is sent to the application when the suspended WiFi module has been resumed again and is ready to handle more requests.

IntOutPrimitive: `API_WIFI_RESUME_IND = 0x6009`

Parameters:

Type	Name	Description
`rsuint32`	`SuspendTime`	The time in ms that the WiFi module was suspended.

22.1.11 API_WIFI_RESUME_REQ

Description: This mail is used to reactivate the suspended WiFI module. This can be used if the application want to wake up the suspended WiFi module before the requested suspend time has elapsed.

IntInPrimitive: `API_WIFI_RESUME_REQ = 0x600A`

Parameters: No parameters

22.1.12 API_WIFI_RESUME_CFM

Description: This mail is used to confirm API_WIFI_RESUME_REQ

IntOutPrimitive: `API_WIFI_RESUME_CFM = 0x600B`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.1.13 API_WIFI_SCAN_CTRL_REQ

Description: This mail is used to enable/disable for ground (not connected to AP) and back ground scanning (connected to AP).

IntInPrimitive: `API_WIFI_SCAN_CTRL_REQ = 0x600C`

Parameters:

Type	Name	Description
`rsbool`	`ForGroundScan`	TRUE to enable for ground scanning and FALSE to disable.
`rsbool`	`BackGroundScan`	TRUE to enable back ground scanning and FALSE to disable.
`rsuint16`	`ActiveScanTime`	The time in ms that the device stays on a particular channel when active scanning. Time is specified in ms (10-65535) and the reset value is uses if set to 0.
`rsuint16`	`PassiveScanTime`	The time in ms that the device stays on a particular channel when passive scanning. Time is specified in ms (10-65535) and the reset value is uses if set to 0.

22.1.14 API_WIFI_SCAN_CTRL_CFM

Description: This mail is used to confirm API_WIFI_SCAN_CTRL_REQ

IntOutPrimitive: `API_WIFI_SCAN_CTRL_CFM = 0x600D`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.1.15 API_WIFI_SET_LISTEN_INTERVAL_REQ

Description: This mail is used to request a listen interval, which determines how often the AR4100 device should wake up and listen for traffic. The listen interval can be set by the TUs or by the number of beacons. The device may not be able to comply with the request (e.g., if the beacon interval is greater than the requested listen interval, the device sets the listen interval to the beacon interval).

IntInPrimitive: `API_WIFI_SET_LISTEN_INTERVAL_REQ = 0x600E`

Parameters:

Type	Name	Description
`rsuint16`	`ListenInterval`	The listen interval in K micro seconds (1024 u seconds) ranging from 100 to 1000.
`rsuint16`	`ListenBeacons`	Specifies the listen interval in beacons, ranging from 1 to 50.

22.1.16 API_WIFI_SET_LISTEN_INTERVAL_CFM

Description:	This mail is used to confirm API_WIFI_SET_LISTEN_INTERVAL_REQ
IntOutPrimitive:	`API_WIFI_SET_LISTEN_INTERVAL_CFM = 0x600F`
Parameters:	

Type	**Name**	**Description**
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.1.17 API_WIFI_SET_KEEP_ALIVE_INTERVAL_REQ

Description:	This mail is used to set a keep-alive interval. If there is no transmission or reception activity for the duration of the keep-alive interval, the STA must send a NULL data packet to the AP it is connected to.
IntInPrimitive:	`API_WIFI_SET_KEEP_ALIVE_INTERVAL_REQ = 0x6010`
Parameters:	

Type	**Name**	**Description**
`rsuint16`	`KeepAliveInterval`	The keep-alive interval in s, range 0-255s.

22.1.18 API_WIFI_SET_KEEP_ALIVE_INTERVAL_CFM

Description:	This mail is used to confirm API_WIFI_SET_KEEP_ALIVE_INTERVAL_REQ
IntOutPrimitive:	`API_WIFI_SET_KEEP_ALIVE_INTERVAL_CFM = 0x6011`
Parameters:	

Type	**Name**	**Description**
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.2 Connection management

The API's described in this section are used for setting up the connection to the AP (associate with the AP).

22.2.1 API_WIFI_SCAN_REQ

Description:	This mail is used to request the device to start scanning for access points.
IntInPrimitive:	`API_WIFI_SCAN_REQ = 0x6020`
Parameters:	No parameters

22.2.2 API_WIFI_SCAN_CFM

Description:	This mail is used to confirm API_WIFI_SCAN_REQ.
IntOutPrimitive:	`API_WIFI_SCAN_CFM = 0x6021`
Parameters:	

Type	**Name**	**Description**
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint16`	`ScanResultCount`	The number of AP found
`ApiWifiScanResultType*`	`ScanResult`	Pointer to memory block holding an array with the scan results.

22.2.3 API_WIFI_GET_SCAN_RESULT_REQ

Description:	This mail is used to request the device to return the scam result with the index specified. This can e.g. be used to read scan results when the API is executed on external host not having direct access to the memory where the scan result is stored.
IntInPrimitive:	`API_WIFI_GET_SCAN_RESULT_REQ = 0x6027`
Parameters:	

Type	**Name**	**Description**
`rsuint16`	`Index`	The index of the scan result to read. Valid index is 0 to ScanResultCount-1

22.2.4 API_WIFI_GET_SCAN_RESULT_CFM

Description: This mail is used to confirm API_WIFI_GET_SCAN_RESULT_REQ.
IntOutPrimitive: `API_WIFI_GET_SCAN_RESULT_CFM = 0x6028`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint16`	`Index`	The index of the scan result returned.
`ApiWifiScanResultType`	`ScanResult`	The scan results.

22.2.5 API_WIFI_SET_SSID_REQ

Description: This mail is used by the application to set the SSID of the AP to connect to. The SSID comes in to effect only when the application commits the changes by sending the API_WIFI_COMMIT_REQ.
IntInPrimitive: `API_WIFI_SET_SSID_REQ = 0x6022`
Parameters:

Type	Name	Description
`rsuint8`	`SsidLength`	The number of chars in the SSID.
`rsuint8`	`Ssid[1]`	The SSID.

22.2.6 API_WIFI_SET_SSID_CFM

Description: This mail is used to confirm API_WIFI_SET_SSID_REQ
IntOutPrimitive: `API_WIFI_SET_SSID_CFM = 0x6023`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.2.7 API_WIFI_COMMIT_REQ

Description: This mail is used by the application to commit to changes to SSID, security mode, etc. made previously.
IntInPrimitive: `API_WIFI_COMMIT_REQ = 0x6024`
Parameters: No parameters

22.2.8 API_WIFI_CONNECT_IND

Description: This mail is used to inform the application when connection to the AP has been established.
IntOutPrimitive: `API_WIFI_CONNECT_IND = 0x6025`
Parameters: No parameters

22.2.9 API_WIFI_DISCONNECT_IND

Description: This mail is used to inform the application when the connection to the AP has been released/lost.
IntOutPrimitive: `API_WIFI_DISCONNECT_IND = 0x6026`
Parameters: No parameters

22.3 Security management

The API’s described in this section are used to manage the security of the connection to the AP.

22.3.1 API_WIFI_WPS_REQ

Description: This mail is used to request a WPS connection to an AP.
IntInPrimitive: `API_WIFI_WPS_REQ = 0x6040`
Parameters:

Type	Name	Description
`ApiWifiWpsModeType`	`Mode`	Push button or PIN mode.
`rsuint8`	`Timeout`	Timeout in seconds (default value to use is 30).
`rsuint8`	`Pin[API_WIFI_PIN_LENGTH]`	The pin to be used in PIN mode.

22.3.2 API_WIFI_WPS_CFM

Description: This mail is used to confirm API_WIFI_WPS_REQ

IntOutPrimitive: `API_WIFI_WPS_CFM = 0x6041`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint8`	`SsidLength`	The number of chars in the SSID.
`rsuint8`	`Ssid[API_WIFI_SSID_LENGTH]`	The SSID.
`ApiWifiSecTypeType`	`SecType`	The type of security.
`ApiWifiCipherInfoType`	`Cipher`	WPA
`rsuint8`	`KeyIndex`	WEP
`rsuint8`	`KeyLength`	Length of Key[]
`rsuint8`	`Key[1]`	WEP key or WPA pass phrase.

22.3.3 API_WIFI_SET_NO_SECURITY_REQ

Description: This mail is used by the application to disable security. This mail must be sent before the commit req.

IntInPrimitive: `API_WIFI_SET_NO_SECURITY_REQ = 0x6042`

Parameters: No parameters

22.3.4 API_WIFI_SET_NO_SECURITY_CFM

Description: This mail is used to confirm API_WIFI_SET_NO_SECURITY_REQ

IntOutPrimitive: `API_WIFI_SET_NO_SECURITY_CFM = 0x6043`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.3.5 API_WIFI_SET_WPA_REQ

Description: This mail is used by the application to request usage of WPA or WPA2 security type. The SSID must be set before this mail is sent and this mail must be sent before the commit req.

IntInPrimitive: `API_WIFI_SET_WPA_REQ = 0x6044`

Parameters:

Type	Name	Description
`rsuint8`	`Version`	1 or 2 for WPA or WPA2.
`ApiWifiCipherInfoType`	`Cipher`	WPA chipper type.
`rsuint8`	`KeyLength`	Length of Key[].
`rsuint8`	`Key[1]`	8-63 char WPA pass phrase or 64 char PSK.

22.3.6 API_WIFI_SET_WPA_CFM

Description: This mail is used to confirm API_WIFI_SET_WPA_REQ

IntOutPrimitive: `API_WIFI_SET_WPA_CFM = 0x6045`

Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.3.7 API_WIFI_SET_WEP_REQ

Description: This mail is used by the application to request usage of WEP security type. The SSID must be set before this mail is sent and this mail must be sent before the commit req.

IntInPrimitive: `API_WIFI_SET_WEP_REQ = 0x6046`

Parameters:

Type	Name	Description
`rsuint8`	`KeyIndex`	WEP key index [1;4]
`rsuint8`	`KeyLength`	Length of Key[]. Can be either 5 or 13 (10 or 26 hex digits). All other values are not supported.
`rsuint8`	`Key[1]`	The WEP key. Only hex data is supported.

22.3.8 API_WIFI_SET_WEP_CFM

Description: This mail is used to confirm API_WIFI_SET_WEP_REQ
IntOutPrimitive: `API_WIFI_SET_WEP_CFM = 0x6047`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.3.9 API_WIFI_GET_SEC_TYPE_REQ

Description: This mail is used by the application to query the current security type used.
IntInPrimitive: `API_WIFI_GET_SEC_TYPE_REQ = 0x6048`
Parameters: No parameters

22.3.10 API_WIFI_GET_SEC_TYPE_CFM

Description: This mail is used to confirm API_WIFI_GET_SEC_TYPE_REQ
IntOutPrimitive: `API_WIFI_GET_SEC_TYPE_CFM = 0x6049`
Parameters:

Type	Name	Description
`ApiWifiSecTypeType`	`SecType`	The type of security.

22.4 WiFi management

The API's described in this section are used for general management of the WiFi module

22.4.1 API_WIFI_GET_SSID_REQ

Description: This mail is used by the application to get the current SSID used.
IntInPrimitive: `API_WIFI_GET_SSID_REQ = 0x6060`
Parameters: No parameters

22.4.2 API_WIFI_GET_SSID_CFM

Description: This mail is used to confirm API_WIFI_GET_SSID_REQ
IntOutPrimitive: `API_WIFI_GET_SSID_CFM = 0x6061`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint8`	`SsidLength`	The number of chars in the SSID.
`rsuint8`	`Ssid[1]`	The SSID.

22.4.3 API_WIFI_GET_MAC_ADDR_REQ

Description: This mail is used to request the MAC device address from the WiFi chip.
IntInPrimitive: `API_WIFI_GET_MAC_ADDR_REQ = 0x6062`
Parameters: No parameters

22.4.4 API_WIFI_GET_MAC_ADDR_CFM

Description: This mail is used to confirm API_WIFI_GET_MAC_ADDR_REQ.
IntOutPrimitive: `API_WIFI_GET_MAC_ADDR_CFM = 0x6063`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint8`	`MacAddr[6]`	The mac address stored in the WiFi chip.

22.4.5 API_WIFI_SET_TX_POWER_REQ

Description: This mail is used to set the TX power level of the WiFi module.
IntInPrimitive: `API_WIFI_SET_TX_POWER_REQ = 0x6064`
Parameters:

Type	Name	Description
`rsuint8`	`TxPower`	Tx power in dBm [0: max]. The max power is limited by the regularity domain and the physical limit of the chip (31.5 dBm).

22.4.6 API_WIFI_SET_TX_POWER_CFM

Description: This mail is used to confirm API_WIFI_SET_TX_POWER_REQ
IntOutPrimitive: `API_WIFI_SET_TX_POWER_CFM = 0x6065`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.4.7 API_WIFI_GET_TX_POWER_REQ

Description: This mail is used to get the TX power level of the WiFi module.
IntInPrimitive: `API_WIFI_GET_TX_POWER_REQ = 0x6066`
Parameters: No parameters

22.4.8 API_WIFI_GET_TX_POWER_CFM

Description: This mail is used to confirm API_WIFI_GET_TX_POWER_REQ
IntOutPrimitive: `API_WIFI_GET_TX_POWER_CFM = 0x6067`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint8`	`TxPower`	The TX power setting of the WiFi module.

22.4.9 API_WIFI_GET_VERSION_REQ

Description: This mail is used to request information about the SW and HW version of the WiFi module.
IntInPrimitive: `API_WIFI_GET_VERSION_REQ = 0x6068`
Parameters: No parameters

22.4.10 API_WIFI_GET_VERSION_CFM

Description: This mail is used to confirm API_WIFI_GET_VERSION_REQ
IntOutPrimitive: `API_WIFI_GET_VERSION_CFM = 0x6069`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint32`	`SwVersion`	WiFi software version.
`rsuint32`	`HwVersion`	WiFi chip version.
`rsuint32`	`HwType`	WiFi chip type.

22.4.11 API_WIFI_SET_PHY_MODE_REQ

Description: This mail is used to set the physical WiFi mode. This command should be used right after WiFi power on only and it must be before the first connect attempt.
IntInPrimitive: `API_WIFI_SET_PHY_MODE_REQ = 0x606A`
Parameters:

Type	Name	Description
`ApiWifiPhyModeType`	`Mode`	WiFi mode

22.4.12 API_WIFI_SET_PHY_MODE_CFM

Description: This mail is used to confirm API_WIFI_SET_PHY_MODE_REQ
IntOutPrimitive: `API_WIFI_SET_PHY_MODE_CFM = 0x606B`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.4.13 API_WIFI_GET_REG_DOMAIN_REQ

Description: This mail is used to get the current regulatory domain.
IntInPrimitive: `API_WIFI_GET_REG_DOMAIN_REQ = 0x606C`
Parameters: No parameters

22.4.14 API_WIFI_GET_REG_DOMAIN_CFM

Description: This mail is used to confirm API_WIFI_GET_REG_DOMAIN_REQ
IntOutPrimitive: `API_WIFI_GET_REG_DOMAIN_CFM = 0x606D`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint32`	`RegDomain`	The current regulatory domain.

22.4.15 API_WIFI_REG_DOMAIN_IND

Description: This mail is used to inform the application when the regulatory domain has changed.
IntOutPrimitive: `API_WIFI_REG_DOMAIN_IND = 0x606E`
Parameters:

Type	Name	Description
`rsuint32`	`RegDomain`	The current regulatory domain.

22.4.16 API_WIFI_GET_RSSI_REQ

Description: This mail is used to get the current RSSI.
IntInPrimitive: `API_WIFI_GET_RSSI_REQ = 0x606F`
Parameters: No parameters

22.4.17 API_WIFI_GET_RSSI_CFM

Description: This mail is used to confirm API_WIFI_GET_RSSI_REQ
IntOutPrimitive: `API_WIFI_GET_RSSI_CFM = 0x6070`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint8`	`Rssi`	The current rssi level.

22.5 SoftAP

22.5.1 API_WIFI_SET_MODE_REQ

Description: This mail is used to control whether the module acts as a station (default) or as an access point (AP).
IntInPrimitive: `API_WIFI_SET_MODE_REQ = 0x60A0`
Parameters:

Type	Name	Description
`ApiWifiModeType`	`Mode`	The new mode.

22.5.2 API_WIFI_SET_MODE_CFM

Description: This mail is used to confirm API_WIFI_AP_SET_MODE_REQ
IntOutPrimitive: `API_WIFI_SET_MODE_CFM = 0x60A1`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.5.3 API_WIFI_AP_SET_CHANNEL_REQ

Description: This mail is used to set the channel used when the module is AP.
IntInPrimitive: `API_WIFI_AP_SET_CHANNEL_REQ = 0x60A2`
Parameters:

Type	Name	Description
`rsuint16`	`Channel`	The channel number in frequency.

22.5.4 API_WIFI_AP_SET_CHANNEL_CFM

Description: This mail is used to confirm API_WIFI_AP_SET_CHANNEL_REQ
IntOutPrimitive: `API_WIFI_AP_SET_CHANNEL_CFM = 0x60A3`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.5.5 API_WIFI_AP_SET_HIDDEN_SSID_REQ

Description: This mail is used to control whether the SSID is broadcast or not when the module is operating as an AP.
IntInPrimitive: `API_WIFI_AP_SET_HIDDEN_SSID_REQ = 0x60A4`
Parameters:

Type	Name	Description
`rsbool`	`HiddenSsid`	Set hidden SSID to TRUE or FALSE.

22.5.6 API_WIFI_AP_SET_HIDDEN_SSID_CFM

Description: This mail is used to confirm API_WIFI_AP_SET_HIDDEN_SSID_REQ
IntOutPrimitive: `API_WIFI_AP_SET_HIDDEN_SSID_CFM = 0x60A5`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.5.7 API_WIFI_AP_SET_CONN_INACT_REQ

Description: This mail is used to set the inactivity period for the AP.
IntInPrimitive: `API_WIFI_AP_SET_CONN_INACT_REQ = 0x60A6`
Parameters:

Type	Name	Description
`rsuint32`	`Period`	Inactivity period in minutes. Default is 5 minutes.

22.5.8 API_WIFI_AP_SET_CONN_INACT_CFM

Description: This mail is used to confirm API_WIFI_AP_SET_CONN_INACT_REQ
IntOutPrimitive: `API_WIFI_AP_SET_CONN_INACT_CFM = 0x60A7`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.5.9 API_WIFI_AP_SET_COUNTRY_REQ

Description: This mail is used to set the country code.
IntInPrimitive: `API_WIFI_AP_SET_COUNTRY_REQ = 0x60A8`
Parameters:

Type	Name	Description
`rsuint8`	`CountryCode[3]`	"DB", "NA", "AL", "DZ", "AR", "AM", "AU", "AT", "AZ", "BH", "BY", "BE", "BZ", "BO", "BR", "BN","BG", "CA", "CL", "CN", "CO", "CR", "HR", "CY", "CZ", "DK", "DO", "EC", "EG", "SV", "EE", "FI","FR", "GE", "DE", "GR", "GT", "HN", "HK", "HU", "IS", "IN", "ID", "IR", "IE", "IL", "IT", "JP","JO", "KZ", "KP", "KR", "K2", "KW", "LV", "LB", "LI", "LT", "LU", "MO", "MK", "MY", "MX", "MC", "MA", "NL", "NZ", "NO", "OM", "PK", "PA", "PE", "PH", "PL", "PT", "PR", "QA", "RO", "RU", "SA", "SG", "SK", "SI", "ZA", "ES", "SE", "CH", "SY", "TW", "TH", "TT","TN", "TR", "UA", "AE", "GB", "US", "UY", "UZ", "VE", "VN", "YE", "ZW"

22.5.10 API_WIFI_AP_SET_COUNTRY_CFM

Description: This mail is used to confirm API_WIFI_AP_SET_COUNTRY_REQ
IntOutPrimitive: `API_WIFI_AP_SET_COUNTRY_CFM = 0x60A9`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.5.11 API_WIFI_AP_SET_BEACON_INTERVAL_REQ

Description: This mail is used to set the beacon interval for the AP.
IntInPrimitive: `API_WIFI_AP_SET_BEACON_INTERVAL_REQ = 0x60AA`
Parameters:

Type	Name	Description
`rsuint16`	`Interval`	Beacon interval in units of ms. Default is 100 ms.

22.5.12 API_WIFI_AP_SET_BEACON_INTERVAL_CFM

Description: This mail is used to confirm API_WIFI_AP_SET_BEACON_INTERVAL_REQ
IntOutPrimitive: `API_WIFI_AP_SET_BEACON_INTERVAL_CFM = 0x60AB`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.5.13 API_WIFI_AP_SET_DHCP_POOL_REQ

Description: This mail is used to set the pool/range of IP addresses that the DHCP server will assign client addresses from.
IntInPrimitive: `API_WIFI_AP_SET_DHCP_POOL_REQ = 0x60AC`
Parameters:

Type	Name	Description
`rsuint32`	`Start`	The first IP address.
`rsuint32`	`End`	The last IP address.
`rsuint32`	`LeaseTime`	The lease time.

22.5.14 API_WIFI_AP_SET_DHCP_POOL_CFM

Description: This mail is used to confirm API_WIFI_AP_SET_DHCP_POOL_REQ
IntOutPrimitive: `API_WIFI_AP_SET_DHCP_POOL_CFM = 0x60AD`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.6 Atheros Firmware Management

22.6.1 API_WIFI_ATH_FW_DOWNLOAD_INIT_REQ

Description: This mail is used to initiate an AR4100 firmware download. (This should not be used by normal applications.)
IntInPrimitive: `API_WIFI_ATH_FW_DOWNLOAD_INIT_REQ = 0x60F0`
Parameters: No parameters

22.6.2 API_WIFI_ATH_FW_DOWNLOAD_INIT_CFM

Description: This mail is used to confirm API_WIFI_ATH_FW_DOWNLOAD_INIT_REQ
IntOutPrimitive: `API_WIFI_ATH_FW_DOWNLOAD_INIT_CFM = 0x60F1`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.6.3 API_WIFI_ATH_FW_WRITE_REQ

Description: This mail is used to write a chunk of AR4100 firmware.
IntInPrimitive: `API_WIFI_ATH_FW_WRITE_REQ = 0x60F2`
Parameters:

Type	Name	Description
`rsuint32`	`Address`	
`rsuint16`	`Length`	
`rsuint8`	`Data[1]`	

22.6.4 API_WIFI_ATH_FW_WRITE_CFM

Description: This mail is used to confirm API_WIFI_ATH_FW_WRITE_REQ
IntOutPrimitive: `API_WIFI_ATH_FW_WRITE_CFM = 0x60F3`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.

22.6.5 API_WIFI_ATH_FW_EXECUTE_REQ

Description: This mail is used to activate the new AR4200 firmware.
IntInPrimitive: `API_WIFI_ATH_FW_EXECUTE_REQ = 0x60F4`
Parameters:

Type	Name	Description
`rsuint32`	`Address`	
`rsuint32`	`Option`	

22.6.6 API_WIFI_ATH_FW_EXECUTE_CFM

Description: This mail is used to confirm API_WIFI_ATH_FW_EXECUTE_REQ
IntOutPrimitive: `API_WIFI_ATH_FW_EXECUTE_CFM = 0x60F5`
Parameters:

Type	Name	Description
`RsStatusType`	`Status`	RSS_SUCCESS if the request succeeded.
`rsuint32`	`Result`	

22.7 Type definitions

22.7.1 Includes

Description: Include of API configuration.
C-syntax:

```
#include <Api/ApiCfg.h>
```

22.7.2 API_WIFI_SSID_LENGTH

Description: Defines the max length of a SSID.
C-syntax:

```
#define API_WIFI_SSID_LENGTH (32)
```

22.7.3 API_WIFI_BSSID_LENGTH

Description: Defines the max length of a BSSID.
C-syntax:

```
#define API_WIFI_BSSID_LENGTH (6)
```

22.7.4 API_WIFI_PIN_LENGTH

Description: Defines the max length of a PIN.
C-syntax:

```
#define API_WIFI_PIN_LENGTH (8)
```

22.7.5 API_WIFI_PSK_LENGTH

Description: Defines the length of the PMK.
C-syntax:

```
#define API_WIFI_PSK_LENGTH (64)
```

22.7.6 ApiWifiMacAddrType

Description: this type is used to hold a MAC address.
C-syntax:

```
typedef rsuint8 ApiWifiMacAddrType[6];
```

22.7.7 ApiWifiWpsModeType

Description: This enum defines the WPS modes supported.

C-syntax:

```
typedef enum ApiWifiWpsModeTypeTag
{
   AWWM_PIN                        = 0x01,

   AWWM_BUTTON_PRESS               = 0x02      Button press mode

} RSENUM8(ApiWifiWpsModeType);
```

22.7.8 ApiWifiSecTypeType

Description: This enum defines the security types supported.

C-syntax:

```
typedef enum ApiWifiSecTypeTypeTag
{
   AWST_NONE                       = 0x00,

   AWST_WEP                        = 0x01,

   AWST_WPA                        = 0x02,

   AWST_WPA2                       = 0x03

} RSENUM8(ApiWifiSecTypeType);
```

22.7.9 ApiWifiCipherTypeType

Description: This enum defines the cipher types supported.

C-syntax:

```
typedef enum ApiWifiCipherTypeTypeTag
{
   AWCT_WEP                        = 0x02,     WEP cipher

   AWCT_TKIP                       = 0x04,     Default for WPA

   AWCT_CCMP                       = 0x08      AES based encryption. Default for WPA2.

} RSENUM8(ApiWifiCipherTypeType);
```

22.7.10 ApiWifiCipherInfoType

Description: This struct is used to hold WPA cipher mode.

C-syntax:

```
typedef struct ApiWifiCipherInfoTypeTag
{
   ApiWifiCipherTypeType       Ucipher;    The cipher type used for unicast
   ApiWifiCipherTypeType       Mcipher;    The cipher type used for multicast
} ApiWifiCipherInfoType;
```

22.7.11 ApiWifiAuthTypeType

Description: This enum defines the WPA/WPA2 authentication types supported.

C-syntax:

```
typedef enum ApiWifiAuthTypeTypeTag
{
   AWAT_PSK                        = 0x01,     PSK

   AWAT_1X                         = 0x02      802.1x

} RSENUM8(ApiWifiAuthTypeType);
```

22.7.12 ApiWifiScanResultType

Description: This struct is used to hold scan result information.
C-syntax:

```
typedef struct ApiWifiScanResultType
{
   rsuint8                 Channel;
   rsuint8                 SsidLength;
   rsuint8                 Rssi;
   rsuint8                 SecurityEnabled;               Security enabled(1) or disabled(0)
   rsuint16                BeaconPeriod;
   rsuint8                 Preamble;
   rsuint8                 BssType;
   rsuint8                 Bssid[API_WIFI_BSSID_LENGTH];  The MAC address of the AP
   rsuint8                 Ssid[API_WIFI_SSID_LENGTH];    The SSID of the AP
   ApiWifiCipherTypeType   RsnCipher;                     WPA2 cipher mode
   ApiWifiAuthTypeType     RsnAuth;                       WPA2 authentication mode
   ApiWifiCipherTypeType   WpaCipher;                     WPA cipher mode
   ApiWifiAuthTypeType     WpaAuth;                       WPA authentication mode
} ApiWifiScanResultType;
```

22.7.13 ApiWifiPowerModeType

Description: This enum defines the power save modes supported.
C-syntax:

```
typedef enum ApiWifiPowerModeTypeTag
{
   AWPM_MAX_POWER              = 0x00,    Power save disabled

   AWPM_POWER_SAVE             = 0x01     Power save enabled

} RSENUM8(ApiWifiPowerModeType);
```

22.7.14 ApiWifiDtimPolicyType

Description: This enum defines the DTIM policy types supported.
C-syntax:

```
typedef enum ApiWifiDtimPolicyTypeTag
{
   AWDP_IGNORE_DTIM            = 0x01,    The WiFi device does not listen to any content after
                                          beacon (CAB) traffic.
   AWDP_NORMAL_DTIM            = 0x02,    DTIM period follows the listen interval.

   AWDP_STRICT_DTIM            = 0x03,    The WiFi device attempts to read all CABtraffic.

   AWDP_AUTO_DTIM              = 0x04     The WiFi device decides what to do.

} RSENUM8(ApiWifiDtimPolicyType);
```

22.7.15 ApiWifiTxWakeupPolicyType

Description: This enum defines the DTIM policy types supported.
C-syntax:

```
typedef enum ApiWifiTxWakeupPolicyTypeTag
{
   AWTP_TX_WAKEUP              = 0x01,    The WiFi device wakes up from sleep on TX.

   AWTP_TX_DONT_WAKEUP         = 0x02     The WiFi device does not wakeup on TX.

} RSENUM8(ApiWifiTxWakeupPolicyType);
```

22.7.16 ApiWifiPhyModeType

Description: This enum defines the physical WiFi modes supported.
C-syntax:

```
typedef enum ApiWifiPhyModeTypeTag
{
   AWPM_11B                          = 0x01,     802.11b mode

   AWPM_11G                          = 0x02,     802.11b and g mode

   AWPM_11N                          = 0x03,     802.11n mode

   AWPM_11G_ONLY                     = 0x04      802.11g only mode

} RSENUM8(ApiWifiPhyModeType);
```

22.7.17 ApiWifiModeType

Description: This enum defines the WiFi modes supported.
C-syntax:

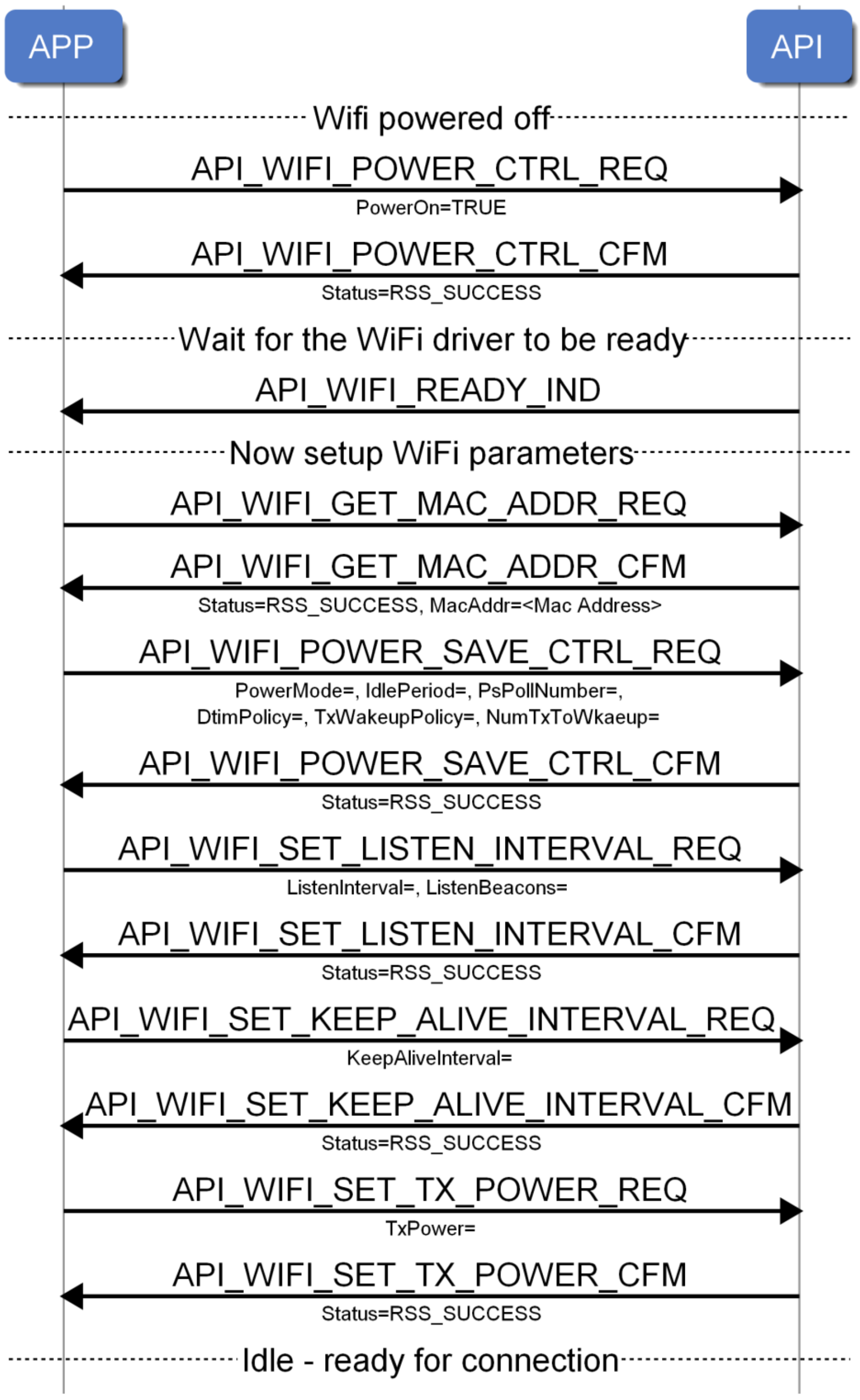
```
typedef enum ApiWifiModeTypeTag
{
   AWM_STATION                       = 0x00,     Normal WiFi mode.

   AWM_AP                            = 0x01      The module acts as an AP.

} RSENUM8(ApiWifiModeType);
```

23 Sequence charts

The usage of the API is described in more details by usage of MSC's in this section.

23.1 WiFi power on

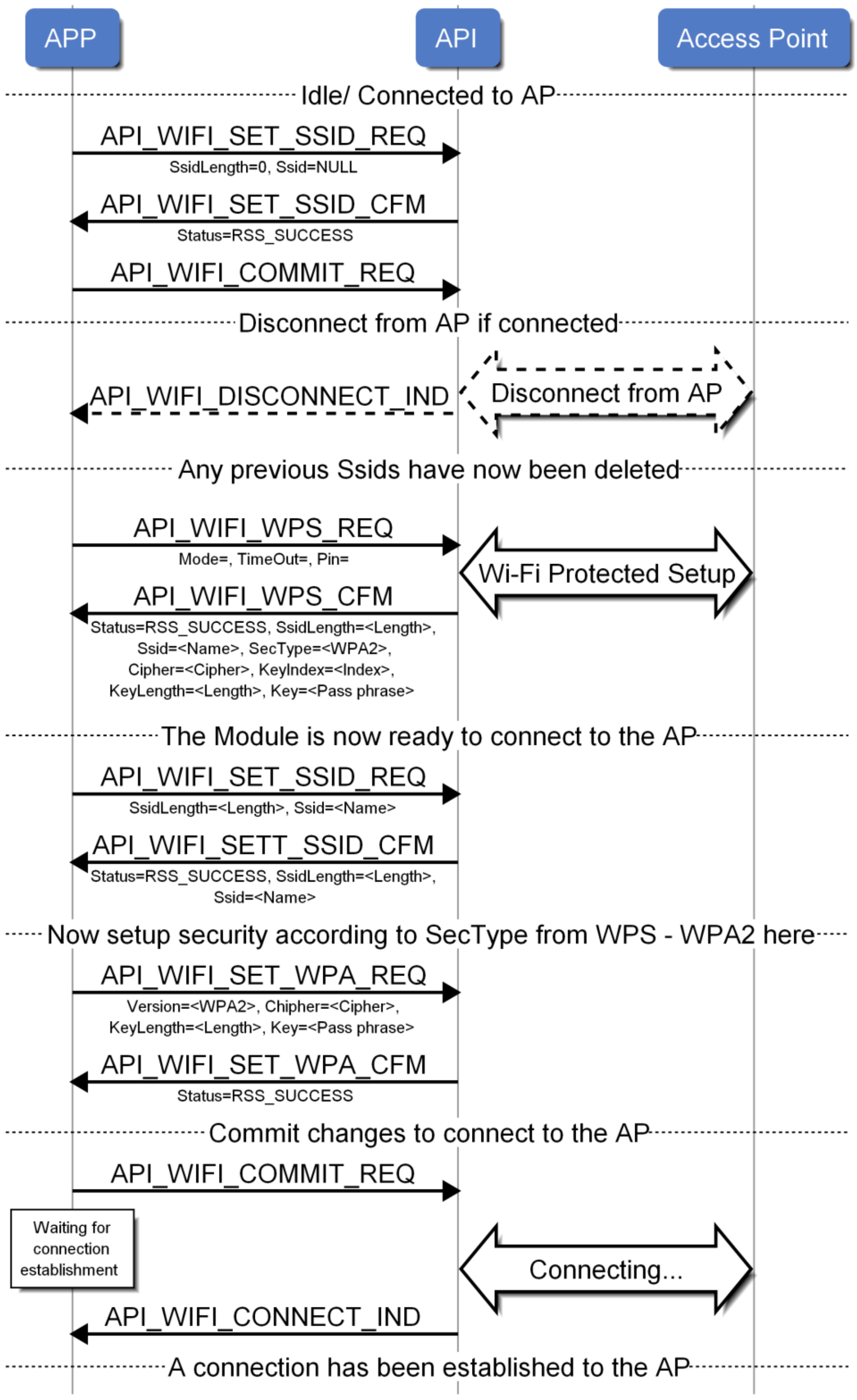
The following MSC illustrates how the Atheros WiFi API to power on and setup the WiFi module.



APP
API
Wifi powered off
API_WIFI_POWER_CTRL_REQ
PowerOn=TRUE
API_WIFI_POWER_CTRL_CFM
Status=RSS_SUCCESS
Wait for the WiFi driver to be ready
API_WIFI_READY_IND
Now setup WiFi parameters
API_WIFI_GET_MAC_ADDR_REQ
API_WIFI_GET_MAC_ADDR_CFM
Status=RSS_SUCCESS, MacAddr=<Mac Address>
API_WIFI_POWER_SAVE_CTRL_REQ
PowerMode=, IdlePeriod=, PsPollNumber=, DtimPolicy=, TxWakeupPolicy=, NumTxToWkaeup=
API_WIFI_POWER_SAVE_CTRL_CFM
Status=RSS_SUCCESS
API_WIFI_SET_LISTEN_INTERVAL_REQ
ListenInterval=, ListenBeacons=
API_WIFI_SET_LISTEN_INTERVAL_CFM
Status=RSS_SUCCESS
API_WIFI_SET_KEEP_ALIVE_INTERVAL_REQ
KeepAliveInterval=
API_WIFI_SET_KEEP_ALIVE_INTERVAL_CFM
Status=RSS_SUCCESS
API_WIFI_SET_TX_POWER_REQ
TxPower=
API_WIFI_SET_TX_POWER_CFM
Status=RSS_SUCCESS
Idle - ready for connection

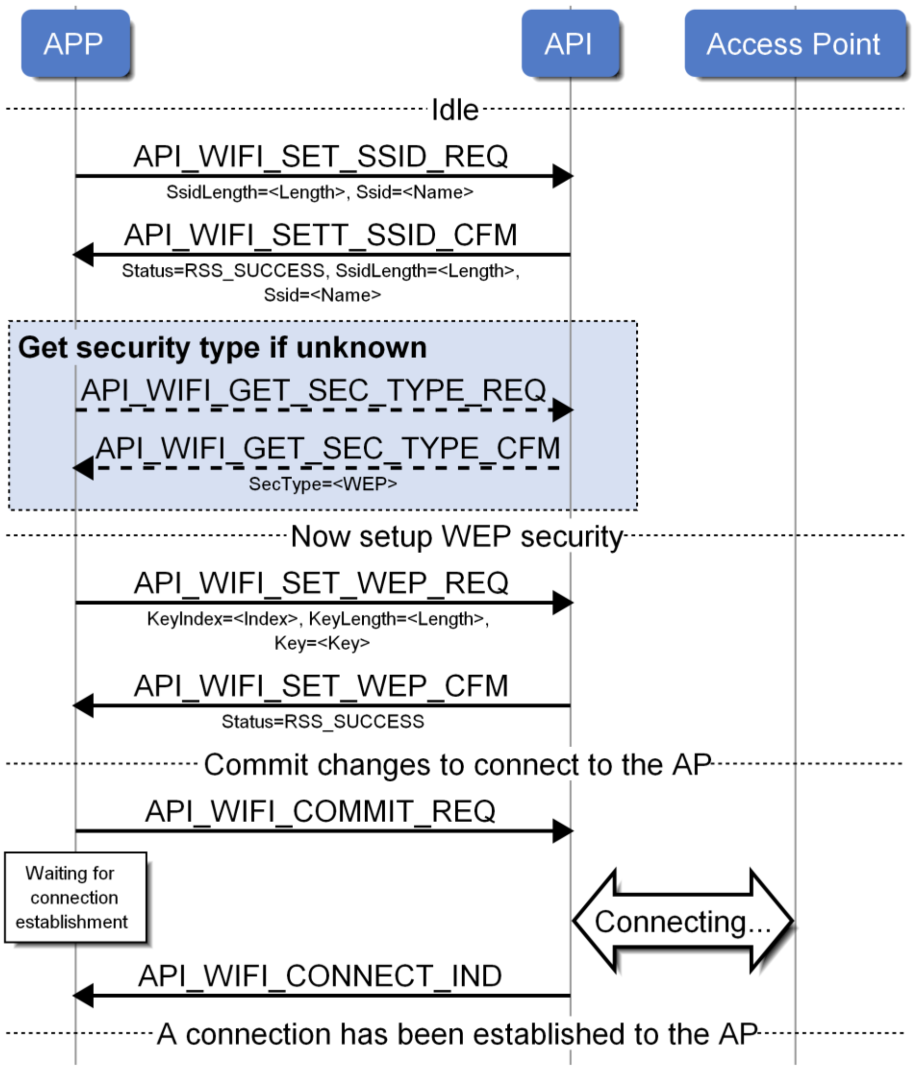
23.2 Wi-Fi Protected Setup and connect

The following MSC shows the relevant steps for initiating Wi-Fi Protected Setup and connection with WPA security mode.


APP
API
Access Point
Idle/ Connected to AP
API_WIFI_SET_SSID_REQ
SsidLength=0, Ssid=NULL
API_WIFI_SET_SSID_CFM
Status=RSS_SUCCESS
API_WIFI_COMMIT_REQ
Disconnect from AP if connected
API_WIFI_DISCONNECT_IND
Disconnect from AP
Any previous Ssids have now been deleted
API_WIFI_WPS_REQ
Mode=, TimeOut=, Pin=
Wi-Fi Protected Setup
API_WIFI_WPS_CFM
Status=RSS_SUCCESS, SsidLength=<Length>, Ssid=<Name>, SecType=<WPA2>, Cipher=<Cipher>, KeyIndex=<Index>, KeyLength=<Length>, Key=<Pass phrase>
The Module is now ready to connect to the AP
API_WIFI_SET_SSID_REQ
SsidLength=<Length>, Ssid=<Name>
API_WIFI_SETT_SSID_CFM
Status=RSS_SUCCESS, SsidLength=<Length>, Ssid=<Name>
Now setup security according to SecType from WPS - WPA2 here
API_WIFI_SET_WPA_REQ
Version=<WPA2>, Chipher=<Cipher>, KeyLength=<Length>, Key=<Pass phrase>
API_WIFI_SET_WPA_CFM
Status=RSS_SUCCESS
Commit changes to connect to the AP
API_WIFI_COMMIT_REQ
Waiting for connection establishment
Connecting...
API_WIFI_CONNECT_IND
A connection has been established to the AP

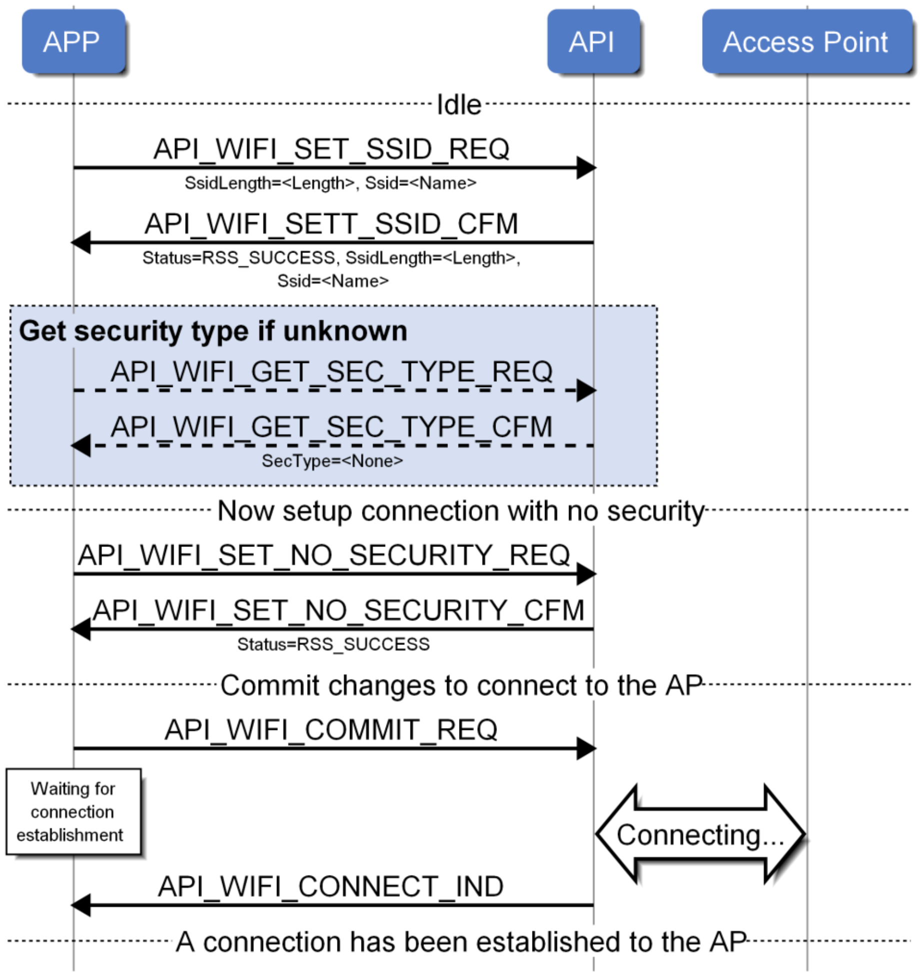
23.3 Wi-Fi connect with WEP

This MSC shows the steps for connecting to an AP with WEP security mode.


APP
API
Access Point
Idle
API_WIFI_SET_SSID_REQ
SsidLength=<Length>, Ssid=<Name>
API_WIFI_SETT_SSID_CFM
Status=RSS_SUCCESS, SsidLength=<Length>, Ssid=<Name>
Get security type if unknown
API_WIFI_GET_SEC_TYPE_REQ
API_WIFI_GET_SEC_TYPE_CFM
SecType=<WEP>
Now setup WEP security
API_WIFI_SET_WEP_REQ
KeyIndex=<Index>, KeyLength=<Length>, Key=<Key>
API_WIFI_SET_WEP_CFM
Status=RSS_SUCCESS
Commit changes to connect to the AP
API_WIFI_COMMIT_REQ
Waiting for connection establishment
Connecting...
API_WIFI_CONNECT_IND
A connection has been established to the AP

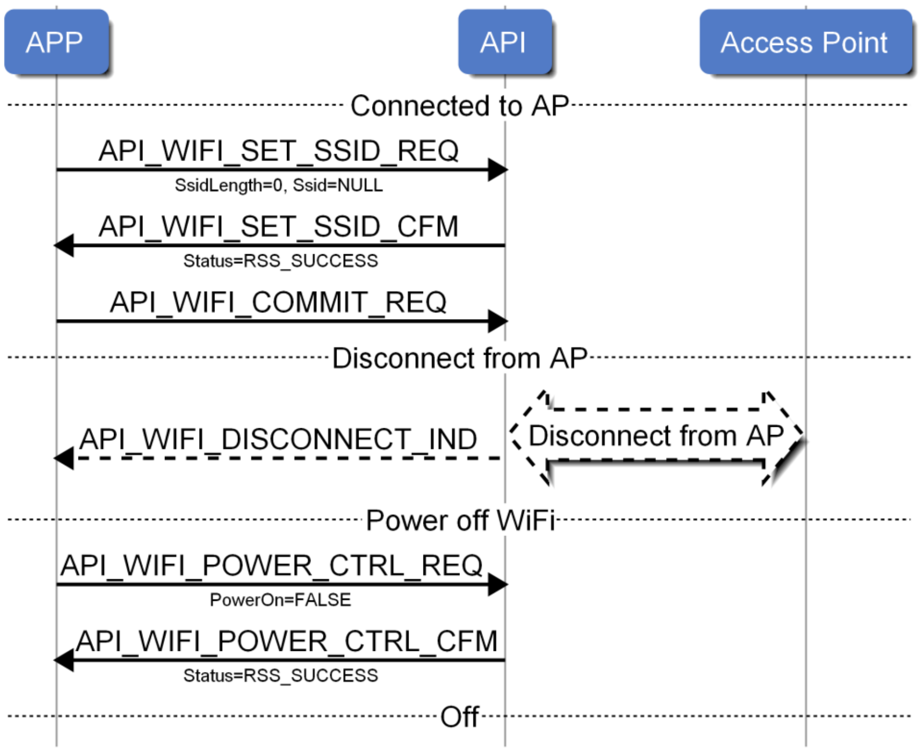
23.4 Wi-Fi connect with no security

This MSC shows the steps for connecting to an AP with no security.


APP
API
Access Point
Idle
API_WIFI_SET_SSID_REQ
SsidLength=<Length>, Ssid=<Name>
API_WIFI_SETT_SSID_CFM
Status=RSS_SUCCESS, SsidLength=<Length>, Ssid=<Name>
Get security type if unknown
API_WIFI_GET_SEC_TYPE_REQ
API_WIFI_GET_SEC_TYPE_CFM
SecType=<None>
Now setup connection with no security
API_WIFI_SET_NO_SECURITY_REQ
API_WIFI_SET_NO_SECURITY_CFM
Status=RSS_SUCCESS
Commit changes to connect to the AP
API_WIFI_COMMIT_REQ
Waiting for connection establishment
Connecting...
API_WIFI_CONNECT_IND
A connection has been established to the AP

23.5 Wi-Fi disconnect and power off

This MSC shows the steps for disconnecting from an AP and powering off of the Wi-Fi module.



For turnkey solutions, please contact RTX

	RTX A/S Stroemmen 6 DK-9400 Noerresundby Denmark Phone: +45 9632 2300 Fax: +45 9632 2310 Web: www.rtx.dk Email: technology@rtx.dk		