GIVE WINGS TO YOUR IDEAS

Open AT IP Connectivity Development Guide

Revision: 002
Date: January 2004

wavecom

PLUG IN TO THE WIRELESS WORLD

This document is the sole and exclusive property of WAVECOM. Not to be distributed or divulged without prior written agreement.
Ce document est la propriété exclusive de WAVECOM. Il ne peut être communiqué ou divulgué à des tiers sans son autorisation préalable.
## Document History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Document history</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>04/11/03</td>
<td>First edition</td>
</tr>
<tr>
<td>002</td>
<td>05/01/04</td>
<td>Added UDP socket &amp; PING services. Updates in SMTP service to support the multi-recipient feature and the authentication feature. Added new error code ED_ERR_GPRS_SESSION_LOST for the responsecbk() of some services and ED_ERR_SERVICE_ALREADY_RUNNING for application callback (WMUX security)</td>
</tr>
</tbody>
</table>
Overview

This document describes the interfaces of the IP Connectivity library for the WAVECOM Open AT software.

It applies to the Open AT library eDLib-W302_V210.

New functionalities of this version are:

- UDP socket (see §2.7 : UDP SOCKET Service)
- PING (see §2.8 : PING Service)
- SMTP multi-recipients and authentication
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCUMENT HISTORY</td>
<td>2</td>
</tr>
<tr>
<td>OVERVIEW</td>
<td>3</td>
</tr>
<tr>
<td>CONTENTS</td>
<td>4</td>
</tr>
<tr>
<td>1 DESCRIPTION</td>
<td>8</td>
</tr>
<tr>
<td>2 OPEN AT IP CONNECTIVITY LIBRARY</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Required Header Files</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Initialization</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Dialing Service</td>
<td>10</td>
</tr>
<tr>
<td>2.3.1 Incoming Call Management</td>
<td>10</td>
</tr>
<tr>
<td>2.3.1.1 Required Header Files</td>
<td>10</td>
</tr>
<tr>
<td>2.3.1.2 The ed_answerSetupParams_t type</td>
<td>10</td>
</tr>
<tr>
<td>2.3.1.3 The ed_AnswerSetConfig function</td>
<td>14</td>
</tr>
<tr>
<td>2.3.1.4 The ed_AnswerGetConfig function</td>
<td>15</td>
</tr>
<tr>
<td>2.3.1.5 The ed_Accept function</td>
<td>15</td>
</tr>
<tr>
<td>2.3.2 Dial Up Management</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2.1 Required Header Files</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2.2 The ed_dialupSetupParams_t type</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2.3 The ed_DialupSetConfig function</td>
<td>19</td>
</tr>
<tr>
<td>2.3.2.4 The ed_DialupGetConfig function</td>
<td>19</td>
</tr>
<tr>
<td>2.3.2.5 The ed_DialupConnectionStart function</td>
<td>20</td>
</tr>
<tr>
<td>2.3.2.6 The ed_DialupConnectionStop function</td>
<td>22</td>
</tr>
<tr>
<td>2.3.2.7 The ed_DialupGetIPInformations function</td>
<td>23</td>
</tr>
<tr>
<td>2.3.2.8 The ed_DialupGetGeneralInformations function</td>
<td>24</td>
</tr>
<tr>
<td>2.3.3 PPP Management</td>
<td>25</td>
</tr>
<tr>
<td>2.3.3.1 Required Header Files</td>
<td>25</td>
</tr>
<tr>
<td>2.3.3.2 The ed_pppSetupParams_t type</td>
<td>25</td>
</tr>
<tr>
<td>2.3.3.3 The ed_PPPSetConfig function</td>
<td>27</td>
</tr>
<tr>
<td>2.3.3.4 The ed_PPPGetConfig function</td>
<td>28</td>
</tr>
<tr>
<td>2.3.4 DNS Management</td>
<td>28</td>
</tr>
<tr>
<td>2.3.4.1 Required Header Files</td>
<td>28</td>
</tr>
<tr>
<td>2.3.4.2 The ed_dnsSetupParams_t type</td>
<td>28</td>
</tr>
<tr>
<td>2.3.4.3 The ed_DNSSetConfig function</td>
<td>29</td>
</tr>
<tr>
<td>2.3.4.4 The ed_DNSGetConfig function</td>
<td>29</td>
</tr>
<tr>
<td>2.3.5 GPRS Management</td>
<td>30</td>
</tr>
<tr>
<td>2.3.5.1 Required Header Files</td>
<td>30</td>
</tr>
<tr>
<td>2.3.5.2 The ed_gprsSetupParams_t type</td>
<td>30</td>
</tr>
<tr>
<td>2.3.5.3 The ed_GPRSSetConfig function</td>
<td>31</td>
</tr>
<tr>
<td>2.3.5.4 The ed_GPRSGetConfig function</td>
<td>32</td>
</tr>
<tr>
<td>2.4 SMTP/POP3 e-mail Service</td>
<td>33</td>
</tr>
<tr>
<td>2.4.1 SMTP e-mail services</td>
<td>33</td>
</tr>
<tr>
<td>2.4.1.1 Required Header Files</td>
<td>33</td>
</tr>
</tbody>
</table>
2.5 FTP Service ................................................................. 49
  2.5.1 Required Header Files ................................................. 49
  2.5.2 The ed_FTPSetupParams_t type ................................... 49
  2.5.3 The ed_FTPSetConfig function .................................... 51
    2.5.3.1 Prototype .......................................................... 51
    2.5.3.2 Parameters ........................................................ 51
    2.5.3.3 Returned values ............................................... 51
  2.5.4 The ed_FTPGetConfig function ................................... 51
    2.5.4.1 Prototype .......................................................... 51
    2.5.4.2 Parameters ........................................................ 51
    2.5.4.3 Returned values ............................................... 51
  2.5.5 The ed_FTPGetFileParams_t type ................................. 52
  2.5.6 The ed_FTPGetFileSetConfig function .......................... 53
    2.5.6.1 Prototype .......................................................... 53
    2.5.6.2 Parameters ........................................................ 53
    2.5.6.3 Returned values ............................................... 53
  2.5.7 The ed_FTPGetFileGetConfig function .......................... 53
    2.5.7.1 Prototype .......................................................... 53
    2.5.7.2 Parameters ........................................................ 53
    2.5.7.3 Returned values ............................................... 53
  2.5.8 The ed_FTPPutFileParams_t type ................................. 54
  2.5.9 The ed_FTPPutFileSetConfig function .......................... 55
    2.5.9.1 Prototype .......................................................... 55
    2.5.9.2 Parameters ........................................................ 55
    2.5.9.3 Returned values ............................................... 55
  2.5.10 The ed_FTPPutFileGetConfig function ........................ 55
    2.5.10.1 Prototype .......................................................... 55
    2.5.10.2 Parameters ........................................................ 55
    2.5.10.3 Returned values ............................................... 55
  2.5.11 The ed_FTPGet function ............................................ 56
    2.5.11.1 Prototype .......................................................... 56
    2.5.11.2 Parameters ........................................................ 56
    2.5.11.3 Returned values ............................................... 57
  2.5.12 The ed_FTPPut function ............................................ 58
    2.5.12.1 Parameters ........................................................ 58
    2.5.12.2 Returned values ............................................... 59

2.6 TCP SOCKET Service .................................................. 60
2.6.1 Required Header Files ................................................................. 60
2.6.2 The ed_SocketSetupParams_t type ........................................ 60
2.6.3 The ed_SocketSetConfig function ............................................... 62
  2.6.3.1 Prototype ................................................................. 62
  2.6.3.2 Parameters .............................................................. 62
  2.6.3.3 Returned values ....................................................... 62
2.6.4 The ed_SocketGetConfig function ........................................... 63
  2.6.4.1 Prototype ............................................................... 63
  2.6.4.2 Parameters ............................................................ 63
  2.6.4.3 Returned values ...................................................... 63
2.6.5 The ed_SocketTCPStart function ............................................. 63
  2.6.5.1 Prototype ............................................................... 63
  2.6.5.2 Parameters ............................................................ 63
  2.6.5.3 Returned values ...................................................... 65
2.6.6 The ed_SocketTCPS Stop function ............................................ 65
  2.6.6.1 Prototype ............................................................... 65
  2.6.6.2 Parameters ............................................................ 65
  2.6.6.3 Returned values ...................................................... 66

2.7 UDP SOCKET Service .................................................................... 67
  2.7.1 Required Header Files ............................................................ 67
  2.7.2 The ed_UdpSocketSetupParams_t type .................................... 67
  2.7.3 The ed_UdpSocketSetConfig function ..................................... 69
    2.7.3.1 Prototype ............................................................ 69
    2.7.3.2 Parameters ........................................................... 69
    2.7.3.3 Returned values .................................................... 69
  2.7.4 The ed_UdpSocketGetConfig function ..................................... 70
    2.7.4.1 Prototype ............................................................ 70
    2.7.4.2 Parameters ........................................................... 70
    2.7.4.3 Returned values .................................................... 70
  2.7.5 The ed_SocketUDPStart function .......................................... 70
    2.7.5.1 Prototype ............................................................ 71
    2.7.5.2 Parameters ........................................................... 71
    2.7.5.3 Returned values .................................................... 72
  2.7.6 The ed_SocketUDPStop function ........................................... 73
    2.7.6.1 Prototype ............................................................ 73
    2.7.6.2 Parameters ........................................................... 73
    2.7.6.3 Returned values .................................................... 73

2.8 PING Service ............................................................................. 74
  2.8.1 Required Header Files ............................................................ 74
  2.8.2 The ed_PINGSetupParams_t type .......................................... 74
  2.8.3 The ed_PINGSetConfig function ............................................. 75
    2.8.3.1 Prototype ............................................................ 75
    2.8.3.2 Parameters ........................................................... 75
    2.8.3.3 Returned values .................................................... 75
  2.8.4 The ed_PINGGetConfig function ........................................... 76
    2.8.4.1 Prototype ............................................................ 76
    2.8.4.2 Parameters ........................................................... 76
    2.8.4.3 Returned values .................................................... 76
  2.8.5 The ed_PingEchoRequest function ........................................ 77
    2.8.5.1 Prototype ............................................................ 77
    2.8.5.2 Parameters ........................................................... 77
    2.8.5.3 Returned values .................................................... 77
  2.8.6 The ed_PingReset function .................................................. 78
    2.8.6.1 Prototype ............................................................ 78
2.8.6.2 Returned values ................................................................. 78

2.9 Data Flows Management ......................................................... 79
  2.9.1 Function eD_SendData ..................................................... 79
    2.9.1.1 Prototype .............................................................. 79
    2.9.1.2 Parameters ............................................................ 79
    2.9.1.3 Returned values ...................................................... 79

3 LIST OF ERROR CODES AND REPLY CODES ............................. 80

4 APPENDIX: OPEN AT RESOURCES .............................................. 84
  4.1 EEPROM objects ............................................................. 84
  4.2 Timers ............................................................................ 84
  4.3 RAM and ROM occupation ................................................. 84
1 Description

This library offers to the Open AT application developers a direct access to the following Internet protocols:

- POP3 (e-mail),
- SMTP (e-mail),
- FTP (file transfer protocol),
- TCP socket,
- DNS (domain name service)
- UDP Socket
- PING Service

OpenAT application has to be developed using Wavecom ADL environment and has to be linked with eDLib-30x_V210 library file.

NOTES:

- it is strongly recommended not to use the eDLib-30x_V210 library and the eDSofw-w302_V210 AT API layer at the same time.
- to use the IP connectivity library services, the GPRS or CSD connection must be done through the IP connectivity library dialing service.
2 Open AT IP connectivity library

The Open AT IP Connectivity library allows only one IP application (FTP, SMTP, POP3 or socket TCP) working at a time.

2.1 Required Header Files.

The interface header files shared by all IP connectivity services are:
   *ed_common.h*

2.2 Initialization

This function must be called in the *adl_main* of the client application.

This function initializes the IP connectivity library.
It loads IP connectivity parameters from EEPROM: these are default parameters or parameters saved in EEPROM.

Notes: If the client application wants to initialize PIN code using ADL function *adl_simSubscribe*, *ed_Init* must be called after *adl_simSubscribe*.

- **Prototype**
  ```c
  s8 ed_Init(void);
  ```

- **Parameters**
  None

- **Returned values**
  This function returns 0 on success, or a negative error value. Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_TCPIP_ALREADYUSED
  - ED_ERR_WM_UNABLE_SUBS_EVENT
  - ED_ERR_PARAM_READ_E2P

See §3 - List of error codes and reply codes for more information on error codes.
2.3 Dialing Service

IP connectivity library provides this service to handle GSM and GPRS dialing related events, and to setup dialing parameters.

2.3.1 Incoming Call Management

This set of parameters defines how the TCP/IP stack will behave when receiving an incoming call.

2.3.1.1 Required Header Files.

The interface header file for the Incoming Call Management service is:

```
ed_answer.h
```

2.3.1.2 The ed_answerSetupParams_t type

Structure containing the parameters about the GSM incoming call management.

```c
#define ANSWER_STRMAXSIZE 64

typedef struct {
    u8 AnswerMode;
    u8 CallbackTimer;
    ascii CallScreenNum[ANSWER_STRMAXSIZE+1];
    u8 RingCount;
    ed_ResponseCbk_f ResponseCbk;
}ed_answerSetupParams_t;
```
• **ANSWERMODE:**
  This parameter defines how the TCP/IP stack will behave when receiving an incoming call in accordance with the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (default)</td>
<td>(Ignore) ignores the incoming call. In this case, it is the responsibility of the application to accept/refuse the incoming call by issuing the <code>ed_Accept()</code> function (see §2.3.1.5).</td>
</tr>
<tr>
<td>1</td>
<td>(Automatic Answer) The TCP/IP stack goes off hook and accepts the incoming call. As described below, the calling number must match the one specified in the CallScreenNum parameter. The RingCount parameter should be &gt; 0.</td>
</tr>
<tr>
<td>2</td>
<td>(Static Callback) The TCP/IP stack ignores the incoming call and then automatically dials DIALN1 or DIALN2 (see §2.3.2.2) by issuing the <code>ed_DialupConnectionStart()</code> function (see §2.3.2.5). As described below, the calling number must match the one specified in the CallScreenNum parameter. The RingCount parameter should be &gt; 0. It also depends on the format of the caller phone number.</td>
</tr>
<tr>
<td>3</td>
<td>(Dynamic Callback) The TCP/IP stack ignores the incoming call and then automatically dials the calling number by issuing the <code>ed_DialupConnectionStart()</code> function (see §2.3.2.5). For this feature, the CallerID service is mandatory. As described below, the calling number must match the one specified in the CallScreenNum parameter. Ensure that the calling number is compatible with the local telephone network (extension issues).</td>
</tr>
</tbody>
</table>

**Note:** The AnswerMode parameter must be configured in accordance with ATS0 configuration for not interacting.

• **CALLBACKTIMER:**
  This parameter defines the number of seconds (between 2 and 255 inclusive) the TCP/IP stack will wait before an automatic callback operation occurs after receiving an incoming call. It only applies when the ANSWERMODE parameter is set to an automatic callback mode (value>1).

  This timer starts after the end of the ringing signal.

  By default this parameter is set to 2.

• **CALLSCREENNUM:**
  When receiving an incoming call, the caller identification (Caller ID) service allows the TCP/IP stack to identify the phone number of the remote caller. This information is helpful in preventing unauthorized callers to trigger actions on the TCP/IP stack.

  This parameter allows the user to filter the incoming calls when the ANSWERMODE parameter is set to an automatic mode (value>0). This filtering does not apply when the ANSWERMODE parameter is set to 0, in this case it is the host’s responsibility to accept or reject the incoming call.

  If an incoming phone number is unauthorized, the TCP/IP stack will ignore it.
This parameter is set in accordance with the following values:

- **0 (zero):** No remote caller authorized (default value)
- ***(all):** No filtering is applied on incoming calls. All the remote phone numbers are authorized. This value must be set when wanting to receive incoming calls while the Caller ID service is not available.

**Decimal phone number:** Only the phone number configured here before is authorized for incoming calls.

By default this parameter is set to 0.

**Note:** this parameter must be a NULL terminated string.

- **RINGCOUNT:**
  This parameter defines the number of rings (range [0 ; 15]) that will be waited before an automatic operation occurs when receiving an incoming call.

  This parameter only applies when the ANSWERMODE parameter is set to an automatic mode (value>0).

  If the ANSWERMODE parameter is used (value different from 0), the RINGCOUNT value should be >0 to use the feature.

  By default this parameter is set to 0.

  **Note:** The RINGCOUNT must be configured in accordance with ATS0 configuration for not interacting.

- **RESPONSECBK:**
  ResponseCbk is called by the IP connectivity library when incoming call is automatically accepted, when the IP session is opened or if the accept action fails.

  This callback function is defined as below:

  ```c
  typedef void (*ed_ResponseCbk_f) (s32 ResponseCode);
  ```
ResponseCode values:

- ED_INFO_CONNECT
  
  Modem speed negotiated between both side. It can be from the V32bis to V.90 depending on the software and also on the conditions. If the TCP/IP stack is configured for modem only operation (PPPMODE parameter, see §2.3.3.2) there is no more return codes.

- ED_OK_PPP
- ED_ERR_BUSY
- ED_ERR_NO_ANSWER
- ED_ERR_NO_CARRIER
- ED_ERR_PPP_ERROR
- ED_ERR_PPP_TIMEOUT
- ED_ERR_INVALID_EVENT
- ED_ERR_PHY_TIMEOUT
- ED_ERR_NETWORK_KO
- ED_ERR_WM_FLOW_OPEN
- ED_ERR_WM_PARSER_FLOW
- ED_ERR_CLI_INVALID

See §3 - List of error codes and reply codes for more information on returned codes.
2.3.1.3 The ed_AnswerSetConfig function

This function sets up some specific parameters about the incoming call management.

- **Prototype**
  
  ```c
  s8 ed_AnswerSetConfig(ed_answerSetupParams_t *Params);
  ```

- **Parameters**
  
  Params :
  Structure containing the parameters to set up. This structure is defined above.

- **Returned values**
  
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  
  ```
  • ED_ERR_NOT_INIT
  • ED_ERR_NO_TCPIP
  • ED_ERR_PARAM_BAD_VALUE
  • ED_ERR_PARAM_TOO_LONG
  • ED_ERR_PARAM_WRITE_E2P
  • ED_ERR_INVALID_ARG
  • ED_ERR_SERVICE_ALREADY_RUNNING
  ```

See §3 - List of error codes and reply codes for more information on error codes.
2.3.1.4 The ed_AnswerGetConfig function

This function gets up some specific parameters about the incoming call management.

- **Prototype**
  
  \[ s8 \text{ed\_AnswerGetConfig(ed\_answerSetupParams\_t *Params);} \]

- **Parameters**

  \[ \text{Params :} \]
  Structure containing the parameters set. This structure is defined above.

- **Returned values**

  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - ED_ERR_NOT_INIT
  - ED_ERR_NO_TCPIP
  - ED_ERR_INVALID_ARG

  See §3 - List of error codes and reply codes for more information on error codes.

2.3.1.5 The ed_Accept function

This command directs the TCP/IP stack to answer an incoming call.

When the incoming call is received, “RING” messages are sent to the customer application (if it did subscribe to the corresponding service). Depending on the value of the ANSWERMODE parameter (see §2.3.1.2) the TCP/IP stack may answer automatically or not.

If ANSWERMODE (see §2.3.1.2) is set to 0, it is the application that is responsible for answering the incoming call with this function.

- **Prototype**
  
  \[ s8 \text{ed\_Accept (ed\_ResponseCbk\_f ResponseCbk);} \]

- **Parameters**

  \[ \text{ResponseCbk :} \]
  ResponseCbk is called by the IP connectivity library when the IP session is opened or if the accept action fails.

  This callback function is defined as below:

  \[ \text{typedef void (* ed\_ResponseCbk\_f) (s32 ResponseCode);} \]
ResponseCode values :

- **ED_INFO_CONNECT**
  
  *Modem speed is negotiated between both sides. It can be from the V32bis to V.90 depending on the software and also on the conditions. If the TCP/IP stack is configured for modem only operation (PPPMODE parameter, see §2.3.3.2) there is no more return codes.*

- **ED_OK_PPP**
- **ED_ERR_BUSY**
- **ED_ERR_NO_ANSWER**
- **ED_ERR_NO_CARRIER**
- **ED_ERR_PPP_ERROR**
- **ED_ERR_PPP_TIMEOUT**
- **ED_ERR_INVALID_EVENT**
- **ED_ERR_PHY_TIMEOUT**
- **ED_ERR_NETWORK_KO**
- **ED_ERR_WM_FLOW_OPEN**
- **ED_ERR_WM_PARSER_FLOW**

- **Returned values**
  
  This function returns 0 on success, or a negative error value. Possible error values are :

  - **ED_ERR_NO_TCPIP**
  - **ED_ERR_NOT_INIT**
  - **ED_ERR_INVALID_ARG**
  - **ED_ERR_MODEM_RUNING**
  - **ED_ERR_INVALID_RING_TYPE**

  See §3 - List of error codes and reply codes for more information on returned values and error codes.

**NOTE** : Once the physical layer is up, the TCP/IP stack runs the applicable protocol as specified in the PPPMODE parameter (see §2.3.3.2).

The **ed.DialupConnectionStop()** function (see §2.3.2.6) allows to on hook and close the connection.
### 2.3.2 Dial Up Management

#### 2.3.2.1 Required Header Files

The interface header file for the Dial Up Management service is:

`ed_dialup.h`

#### 2.3.2.2 The `ed_dialupSetupParams_t` type

Structure containing the parameters about the GSM dial up management.

```c
#define DIALUP_STRMAXSIZE 64

typedef struct
{
  ascii DIALN1[DIALUP_STRMAXSIZE+1];
  ascii DIALN2[DIALUP_STRMAXSIZE+1];
  u8 DIALSELECT;
  ascii ISPUN[DIALUP_STRMAXSIZE+1];
  ascii ISPPW[DIALUP_STRMAXSIZE+1];
  u8 REDIALCOUNT;
  u8 REDIALDELAY;
  u8 PHYTIMEOUT;
} ed_dialupSetupParams_t;
```

- **DIALN1 and DIALN2**: Primary and secondary dial-up phone number to connect with the local ISP. Length depends on country.
  
  By default these parameters are set to empty strings.

- **DIALSELECT**: The value of this parameter determines the number called to establish an Internet connection. It configures the TCP/IP stack software to use the primary dial-up number or the secondary dial-up number.
  
  By default this parameter is set to 1.

- **ISPUN**: User name of the ISP account in ASCII text string up to 64 characters. When communication is initiated and once the physical (modem) connection has been established with the ISP, the TCP/IP stack software must provide the ISP with the user name associated with the account to be used.
  
  By default this parameter is set to empty string.
• **ISPPW:**
  Password for the ISP account in ASCII text string up to 64 characters.

  When communication is initiated and once the physical (modem) connection has been established with the ISP, the TCP/IP stack software must provide the ISP with the password associated with the account to be used.

  By default this parameter is set to empty string.

• **REDIALCOUNT:**
  Indicates how many unsuccessful connection (between 0 and 14, inclusive) attempts the TCP/IP stack software will try before terminating the connection attempt.

  If the value is set to 0, the TCP/IP stack software will not make any call retry.

  By default this parameter is set to 5.

• **REDIALDELAY:**
  This parameter controls how long (in seconds between 5 and 14 inclusive), if any, it will take between each call retry.

  If this parameter is configured to zero, the TCP/IP stack software will attempt another connection immediately after terminating the previous unsuccessful attempt.

  By default this parameter is set to 5.

• **PHYTIMEOUT:**
  This parameter is used by the TCP/IP stack software in order to terminate connections to the telephone line when a long period elapses without activity. “Without activity” is defined as a period (between 1 and 255 minutes) when no data is transferred between the Internet and the TCP/IP stack software. This timer prevents the telephone line from being tied up if for any reason some part of the system becomes stuck.

  By default this parameter is set to 15.
2.3.2.3 The ed_DialupSetConfig function

This function sets up some specific parameters about the dial up management.

• **Prototype**
  
  ```c
  s8 ed_DialupSetConfig(ed_dialupSetupParams_t *Params);
  ```

• **Parameters**
  
  *Params:*
  Structure containing the parameters to set up. This structure is defined above.

• **Returned values**
  
  This function returns 0 on success, or a negative error value. Possible error values are:
  
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_PARAM_BAD_VALUE
  - ED_ERR_PARAM_TOO_LONG
  - ED_ERR_PARAM_WRITE_E2P
  - ED_ERR_INVALID_ARG
  - ED_ERR_SERVICE_ALREADY_RUNNING

  See §3 - List of error codes and reply codes for more information on error codes.

2.3.2.4 The ed_DialupGetConfig function

This function gets up some specific parameters about the dial up management.

• **Prototype**
  
  ```c
  s8 ed_DialupGetConfig(ed_dialupSetupParams_t *Params);
  ```

• **Parameters**
  
  *Params:*
  Structure containing the parameters set. This structure is defined above.

• **Returned values**
  
  This function returns 0 on success, or a negative error value. Possible error values are:
  
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG

  See §3 - List of error codes and reply codes for more information on error codes.
2.3.2.5 The ed_DialupConnectionStart function

**Important note:** GPRS attachment must be performed before calling this function.

This function directs the TCP/IP stack to dial out and establish the connection.

Upon receiving this signal, the TCP/IP stack automatically initiates a complete connection session according to the `GPRSMODE` parameter (see §2.3.5.2), selecting the GSM or GPRS mode.

- In GSM mode, the TCP/IP stack will dial the number according to the Dial Option parameter (`DIALN1` or `DIALN2` depending on `DIALESELECT` (see §2.3.2.2)). If an error occurs, the TCP/IP stack automatically attempts according to the `REDIALCOUNT` parameter (see §2.3.2.2).

Once the physical layer is up, the TCP/IP stack runs the applicable protocol as specified in the `PPPMODE` parameter (see §2.3.3.2).
- In GPRS mode, the TCP/IP stack will establish a GPRS session with the APN using `APNUM`, `APNPW`, `CID` parameter (see §2.3.5.2). Once the GPRS link is up, the product is connected to the Internet.

### Prototype

```c
s8 ed_DialupConnectionStart (ed_ResponseCbk_f ResponseCbk);
```

### Parameters

**ResponseCbk**:

ResponseCbk is called by the IP connectivity library when the connection is established or if the connection fails.

This callback function is defined as below:

```c
typedef void (*ed_ResponseCbk_f) (s32 ResponseCode);
```

### ResponseCode values:

- **ED_INFO_CONNECT**
  
  *Modem speed negotiated between both side.* If the TCP/IP stack is configured for modem only operation (`PPPMODE` parameter, see §2.3.3.2) there is no more return codes.

- **ED_OK_PPP**

- **ED_ERR_BUSY**
  
  *A busy signal is detected on the remote site.* TCP/IP stack will wait `REDIALDELAY` seconds (see §2.3.2.2) and then dials again. This redialing will continue until success or until the number of call retries, defined in parameter `REDIALCOUNT` (see §2.3.2.2), has been reached.

- **ED_ERR_NO_ANSWER**
  
  *There is no response from the remote site.* TCP/IP stack will wait `REDIALDELAY` seconds (see §2.3.2.2) and then dials again. This redialing will continue until success or until the number of call retries,
defined in parameter `REDIALCOUNT` (see §2.3.2.2), has been reached.

- **ED_ERR_NO_CARRIER**
  The modem handshaking process with the remote host is interrupted or unsuccessful. TCP/IP stack will wait `REDIALDELAY` seconds (see §2.3.2.2) and then dials again. This re-dialing will continue until success or until the number of call retries, defined in parameter `REDIALCOUNT` (see §2.3.2.2), has been reached.

- **ED_ERR PPP_ERROR**
- **ED_ERR PPP_TIMEOUT**
- **ED_OK_GPRS_SESSION_SET**
- **ED_ERR INVALID EVENT**
- **ED_ERR PHY_TIMEOUT**
- **ED_ERR NETWORK KO**
- **ED_ERR SIM REMOVED**
- **ED_ERR WM FLOW OPEN**
- **ED_ERR WM_PARSER FLOW**
- **ED_ERR_WM_GPRS_OPEN**
- **ED_ERR GPRS ABORTED**
- **ED_ERR_WM_GPRS_IPCP**
- **ED_ERR_GPRS_SESSION_LOST**

- **Returned values**
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - **ED_ERR_NO_TCPIP**
  - **ED_ERR NOT_INIT**
  - **ED_ERR_PIN_NOT_ENTERED**
  - **ED_ERR_NETWORK_KO**
  - **ED_ERR INVALID_ARG**
  - **ED_ERR_NO_GPRS**
  - **ED_ERR MODEM_RUNNING**

See §3 - List of error codes and reply codes for more information on error codes.

**NOTE** : The `ed_DialupConnectionStop()` function (see §2.3.2.6) allows to on hook and close the connection.
2.3.2.6 The ed_DialupConnectionStop function

This function directs the TCP/IP stack to end a GPRS or GSM communication previously established with a START command.

- **Prototype**
  ```c
  s8 ed_DialupConnectionStop (ed_ResponseCbk_f ResponseCbk);
  ```

- **Parameters**
  
  ResponseCbk:

  ResponseCbk is called by the IP connectivity library when the connection is closed or if an error occurs.

  This callback function is defined as below:
  ```c
  typedef void (* ed_ResponseCbk_f) (s32 ResponseCode);
  ```

  ResponseCode values:
  - ED_OK_ON_HOOK
  - ED_ERR_ON_HOOK
  - ED_ERR_WM_FLOW_CLOSE
  - ED_ERR_NETWORK_KO
  - ED_ERR_SIM_REMOVED
  - ED_ERR_NO_CARRIER

- **Returned values**

  This function returns 0 on success, or a negative error value.

  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG
  - ED_ERR_PHY_NOT_ACTIVATED

  See §3 - List of error codes and reply codes for more information on error codes.
2.3.2.7 The ed_DialupGetIPInformations function

This function gets information about the activated IP connection.

- **Prototype**
  
  ```c
  s8 adl_DialupGetIpInformations (ed_dialupIPInfos_t * Infos);
  ```

- **Parameters**
  
  **Infos:**
  
  Structure containing the data of the activated IP connection using the following type:

  ```c
  typedef struct
  {
    u32 LocalIP; // Local IP address of the MS (only if is activated, else 0)
    u32 DistantIP; // Distant IP address (only if is activated, else 0)
    u32 Gateway; // Gateway IP address (only if is activated, else 0)
  } ed_dialupIPInfos_t;
  ```

- **Returned values**
  
  This function returns 0 on success, or a negative error value.

  Possible error values are:
  
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG
  - ED_ERR_PHY_NOT_ACTIVATED

  See §3 - List of error codes and reply codes for more information on error codes.
2.3.2.8  The ed_DialupGetGeneralInformations function

This function gets general information about the TCP/IP stack.

- **Prototype**
  
  ```c
  s8 adl_DialupGetGeneralInformations (ed_dialupGenInfos_t * Infos);
  ```

- **Parameters**

  **Infos:**
  Structure containing the data using the following type:

  ```c
typedef struct
  {
    ed_state_e STATE;       // current status of the IP connectivity library.
    ascii VERSION[100];    // current version of the IP connectivity library
  } ed_dialupGenInfos_t;
  ```

  **typedef enum**

  ```c
  {
    ED_INIT,
    ED_NO_SERVICE,
    ED_IDLE,
    ED_STARTING,
    ED_DIALING,
    ED_AUTHENTICATING,
    ED_CONNECTED,
    ED_DISCONNECTING
  } ed_state_e
  ```

- **Returned values**

  This function returns 0 on success, or a negative error value.

  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG

  See §3 - List of error codes and reply codes for more information on error codes.
2.3.3 PPP Management

2.3.3.1 Required Header Files
The interface header file for the PPP management service is:

    ed_ppp.h

2.3.3.2 The ed_pppSetupParams_t type
Structure containing the parameters about the PPP management.

#define PPP_AUTHSTRMAXSIZE 64
#define PPP_IPADDRSTRMAXSIZE 16

typedef struct
{
    ascii PPPMYIP[PPP_IPADDRSTRMAXSIZE+1];
    ascii PPPEERIP[PPP_IPADDRSTRMAXSIZE+1];
    ascii PPPSVRIP[PPP_AUTHSTRMAXSIZE+1];
    ascii PPSRVFPW[PPP_AUTHSTRMAXSIZE+1];
    u8 PPPMODE;
} ed_pppSetupParams_t;

- PPPMYIP:
  This parameter defines the IP address (32-bit number in dotted-decimal
  notation (i.e. xxx.xxx.xxx.xxx)) to be attributed to the TCP/IP stack when the
  PPP Server mode is running.

  By default this parameter is set to 0.0.0.0

Note:
When the TCP/IP stack behaves as a PPP server (according to the PPPMODE
parameter), it is in charge of the IP address attribution mechanism.

Once the PPP authentication successfully achieved, the remote PPP peer asks
the TCP/IP stack for an IP address. The related PPP layer, called IPCP,
suggests the peer an IP address previously stored in the TCP/IP stack
parameters. If the remote accepts this address, the IP link is then established.
• **PPPEERIP** :
  This parameter defines the IP address (32-bit number in dotted-decimal notation (i.e. xxx.xxx.xxx.xxx)) to be attributed to the remote PPP peer when the PPP Server mode is running.

  By default this parameter is set to 0.0.0.0

  Note:
  When the TCP/IP stack behaves as a PPP server (according to the PPPMODE parameter), it is in charge of the IP address attribution mechanism.
  Once the PPP authentication successfully achieved, the remote PPP peer asks the TCP/IP stack for an IP address. The related PPP layer, called IPCP, then suggests the peer an IP address previously stored in the TCP/IP stack parameters. If the remote accepts this address, the IP link is then established.

• **PPPSRVUN** :
  This parameter defines the login (Alpha-numeric ASCII text string up to 64 characters ) that must be specified by the remote PPP client.

  By default this parameter is set to empty string

  Note:
  When the TCP/IP stack behaves as a PPP server (according to the PPPMODE parameter), it checks the remote PPP client login/password before granting access.

• **PPPSRVPW** :
  This parameter defines the password (alpha-numeric ASCII text string up to 64 characters) that must be specified by the remote PPP client.

  By default this parameter is set to empty string

  Note:
  When the TCP/IP stack behaves as a PPP server (according to the PPPMODE parameter), it checks the remote PPP client login/password before granting access.
• **PPPMODE**:<br>This parameter selects the behavior that the TCP/IP stack will have after the physical layer successful launching. These are its possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(Modem) the TCP/IP stack behaves as a standard modem. The data is sent/received to/from a remote standard modem in a point-to-point connection. No IP/PPP stack is running.</td>
</tr>
<tr>
<td>1 (default)</td>
<td>(Standard PPP) the TCP/IP stack behaves as a PPP client for outgoing calls and as a PPP server for incoming calls.</td>
</tr>
<tr>
<td>2</td>
<td>(Reverse PPP) the TCP/IP stack behaves as a PPP server for outgoing calls and as a PPP client for incoming calls.</td>
</tr>
<tr>
<td>3</td>
<td>(PPP client only) the TCP/IP stack always behaves as a PPP client for both outgoing and incoming calls.</td>
</tr>
<tr>
<td>4</td>
<td>(PPP server only) the TCP/IP stack always behaves as a PPP server for both outgoing and incoming calls.</td>
</tr>
</tbody>
</table>

By default this parameter is set to 1.

### 2.3.3.3 The ed_PPPSetConfig function

This function sets up some specific parameters about the PPP management.

- **Prototype**
  
  ```c
  s8 ed_PPPSetConfig(ed_pppSetupParams_t  *Params);
  ```

- **Parameters**
  
  - **Params**
    - Structure containing the parameters to set up. This structure is defined above.

- **Returned values**
  
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG
  - ED_ERR_PARAM_BAD_VALUE
  - ED_ERR_PARAM_TOO_LONG
  - ED_ERR_PARAM_WRITE_E2P
  - ED_ERR_SERVICE_ALREADY_RUNNING

  See §3 - List of error codes and reply codes for more information on error codes.
2.3.3.4 The ed_PPPGetConfig function

This function gets up some specific parameters about the PPP management.

- **Prototype**
  
  ```c
  s8 ed_PPPGetConfig(ed_pppSetupParams_t *Params);
  ```

- **Parameters**
  
  *Params :
  
  Structure containing the parameters set. This structure is defined above.

- **Returned values**
  
  This function returns 0 on success, or a negative error value.
  
  Possible error values are:
  
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG

  See §3 - List of error codes and reply codes for more information on error codes.

2.3.4 DNS Management

2.3.4.1 Required Header Files

The interface header file for the DNS management service is:

`ed_dns.h`

2.3.4.2 The ed_dnsSetupParams_t type

Structure containing the parameters about the DNS.

```c
#define DNS_IPADDRSTRMAXSIZE 15

typedef struct
{
  ascii DnsServ1[DNS_IPADDRSTRMAXSIZE+1];
  ascii DnsServ2[DNS_IPADDRSTRMAXSIZE+1];
}ed_dnsSetupParams_t;
```

- **DnsServ1 and DnsServ2:**
  
  In order to translate the server names from literal format into IP addresses, the TCP/IP stack software implements the Domain Name System (DNS) protocol. The DNS Server IP address must be specified to the TCP/IP stack software.
  
  It is a 32-bits number in dotted-decimal notation (i.e. xxx.xxx.xxx.xxx).
  
  By default this parameter is set to 0.0.0.0.
2.3.4.3 The ed_DNSSetConfig function

This function sets up DNS parameters.

- **Prototype**
  ```c
  s8 ed_DNSSetConfig(ed_dnsSetupParams_t *Params);
  ```

- **Parameters**
  
  Params :
  Structure containing the DNS parameters to set up. This structure is defined above.

- **Returned values**
  
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG
  - ED_ERR_PARAM_BAD_VALUE
  - ED_ERR_PARAM_TOO_LONG
  - ED_ERR_PARAM_WRITE_E2P
  - ED_ERR_SERVICE_ALREADY_RUNNING

  See §3 - List of error codes and reply codes for more information on error codes.

2.3.4.4 The ed_DNSGetConfig function

This function gets up DNS parameters.

- **Prototype**
  ```c
  s8 ed_DNSGetConfig(ed_dnsSetupParams_t *Params);
  ```

- **Parameters**
  
  Params :
  Structure containing the parameters set. This structure is defined above.

- **Returned values**
  
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG

  See §3 - List of error codes and reply codes for more information on error codes.
2.3.5 GPRS Management

2.3.5.1 Required Header Files
The interface header file for the GPRS management service is:

```
#include "ed_gprs.h"
```

2.3.5.2 The ed_gprsSetupParams_t type
Structure containing the parameters about the GPRS management.

```
#define GPRS_AUTHSTRMAXSIZE 50
#define GPRS_STRMAXSIZE 100

typedef struct
{
    u8 Cid;
    u8 Mode;
    ascii ApnServ[GPRS_STRMAXSIZE+1];
    ascii ApnUn[GPRS_AUTHSTRMAXSIZE+1];
    ascii ApnPw[GPRS_AUTHSTRMAXSIZE+1];
} ed_gprsSetupParams_t;
```

- **Cid:**
  This parameter defines the PDP context identifier (between 1 and 4) which specifies a particular PDP context definition.

  By default this parameter is set to 1.

  **Note:**
  This parameter is local and may be used in other PDP context-related commands: the ed_GPRSSetConfig() function will overwrite the corresponding global PDP context definition.

- **Mode:**
  This parameter configures the activation of the Wavecom software for switching between GSM or GPRS as the following:

  - 0: WAVECOM software is configured for a GSM use
  - 1: WAVECOM software is configured for a GPRS use

  By default this parameter is set to GPRS use (Mode = 1).

- **ApnServ:**
  This parameter defines the Access Point Name parameter (alphanumeric ASCII text string up to 100 characters) coming from the GSM operator for providing GPRS access.

  By default this parameter is set to empty string
• **ApnUn** :
  This parameter defines Access Point Name Username (Alphanumeric ASCII text string up to 50 characters) parameter coming with the ApnPw from the GSM operator for providing GPRS access.

  By default this parameter is set to empty string

• **ApnPw** :
  This parameter defines the Access Point Name Password parameter (Alphanumeric ASCII text string up to 50 characters) used with the ApnUn from the GSM operator for providing GPRS access.

  By default this parameter is set to empty string

### 2.3.5.3 The ed_GPRSSetConfig function

This function sets up some specific parameters about the GPRS management.

- **Prototype**
  ```c
  s8 ed_GPRSSetConfig(ed_gprsSetupParams_t *Params);
  ```

- **Parameters**
  *Params :
  Structure containing the parameters to set up. This structure is defined above.

- **Returned values**
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG
  - ED_ERR_PARAM_BAD_VALUE
  - ED_ERR_PARAM_TOO_LONG
  - ED_ERR_PARAM_WRITE_E2P
  - ED_ERR_MODEM_RUNNING
  - ED_ERR_SERVICE_ALREADY_RUNNING

  See §3 - List of error codes and reply codes for more information on error codes.
2.3.5.4 The ed_GPRSGetConfig function

This function gets up some specific parameters about the GPRS management.

- **Prototype**
  
  ```c
  s8 ed_GPRSGetConfig(ed_gprsSetupParams_t *Params);
  ```

- **Parameters**

  Params :
  Structure containing the parameters set. This structure is defined above.

- **Returned values**

  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG

  See §3 - List of error codes and reply codes for more information on error codes.
2.4 SMTP/POP3 e-mail Service

IP connectivity library provides this service to handle e-mail related events, and to setup e-mail parameters.

2.4.1 SMTP e-mail services.

The SMTP service handles the sending of e-mail to a remote SMTP server.

In addition to the regular recipient (REC) field (the "To:" field), the carbon copy (CCREC) field (the "Cc:" field) is also supported. Each of these fields (REC, CCREC) may hold up to ten e-mail addresses. Each of these addresses for each field is identified by an index (between 1 and 10).

The SMTP service also offers a preset e-mail feature. It is possible to define up to three pre-recorded e-mails (by pre-setting, for each of them, the body, recipients (up to 10 e), carbon-copy (up to 10 e), the subject).

The preset e-mails are identified by their id (1, 2, 3). The id #1 e-mail settings (except the body) are also used to send non pre-recorded e-mails.

For compatibility reasons, only the first REC and CCREC address (identified by the index 1 in the list of 10 possible address) can be set with ed_EmailSetConfig(). ed_EmailAddrSet() should be used to set the 9 other REC and CCREC addresses.

2.4.1.1 Required Header Files.

The interface header file for the SMTP service is:

`ed_smtp.h`

2.4.1.2 The ed_EmailSetupParams_t type

Structure containing the parameters about the E-mail messages management.

```c
#define SMTP_STRMAXSIZE 120

typedef struct
{
  u8     Id;
  ascii  CcRec[SMTP_STRMAXSIZE+1];
  ascii  Rec[SMTP_STRMAXSIZE+1];
  ascii  Subj[SMTP_STRMAXSIZE+1];
  ascii  Body[SMTP_STRMAXSIZE+1];
} ed_EmailSetupParams_t;
```

- **Id**:
  
  The ID of the parameter to setup (from 1 to 3).
• **CcRec** :
  This parameter contains the e-mail address (up to 120 characters) of the additional recipient with index 1. For a given email id, up to 10 E-mail addresses may be recorded in the ‘Cc:’ field.
  There are two ways to access this field:
  - by using the ed_EmailSetConfig() and ed_EmailGetConfig() functions for the address with index 1 only,
  - or by using the ed_EmailAddrSet() and ed_EmailAddrGet() functions.

  By default this parameter is set to an empty string

  **NOTE:**
  For a given n, the “CCRECn” parameter is directly associated with the “RECn” parameter.
  The address must be provided in literal format (for instance dev12345678@wavecom.com).

• **Rec** :
  This parameter contains the e-mail address (text string up to 120 characters) of the recipient with index 1. For a given email id, up to 10 e-mail addresses may be recorded in the ‘To:’ field. This e-mail address will appear first in the header of the e-mail sent by the TCP/IP stack software in the field 'To: '.
  There are two ways of accessing this field for read or write:
  1. by using the ed_EmailSetConfig() and ed_EmailGetConfig() functions for the address with index #1 only,
  2. or by using the ed_EmailAddrSet() and ed_EmailAddrGet() functions.

  By default this parameter is set to an empty string

  **NOTE:**
  The address must be provided in literal format (for instance dev12345678@wavecom.com).

• **Subj** :
  This parameter contains pre-defined one-line subject (text string of up to 120 characters) that will be used by the TCP/IP stack to compose the e-mail header.

  By default this parameter is set to empty string

• **Body** :
  These parameters store pre-defined message bodies. They allow the host application to send pre-defined e-mail combinations.

  By default this parameter is set to empty string.
2.4.1.3 The ed_EmailSetConfig function
This function sets up some specific parameters about the e-mail messages management.

- **Prototype**
  s8 ed_EmailSetConfig(ed_EmailSetupParams_t *Params);

- **Parameters**
  Structure containing the parameter to set up. This structure is defined above.

- **Returned values**
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_PARAM_BAD_VALUE
  - ED_ERR_PARAM_TOO_LONG
  - ED_ERR_PARAM_WRITE_E2P
  - ED_ERR_INVALID_ARG
  - ED_ERR_SERVICE_ALREADY_RUNNING

See §3 - List of error codes and reply codes for more information on error codes.

2.4.1.4 The ed_EmailGetConfig function
This function gets up some specific parameters about the e-mail messages management.

- **Prototype**
  s8 ed_EmailGetConfig(ed_EmailSetupParams_t *Params);

- **Parameters**
  Structure containing the parameter to get. This structure is defined above.

- **Returned values**
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG
  - ED_ERR_PARAM_BAD_VALUE

See §3 - List of error codes and reply codes for more information on error codes.
2.4.1.5 The ed_EmailAddrParams_t type.

Structure holding an e-mail address, used to set or get one of the recipient and Carbon-copy recipient e-mail addresses (multi-recipient feature).

typedef struct
{
    u8    Id;
    u8     MailAddrIdx;
    ascii    MailAddr[SMTP_STRMAXSIZE+1];
} ed_EmailAddrParams_t;

- **Id**: The ID of the parameter to set up (from 1 to 3).

- **MailAddrIdx**: The e-mail address rank. For a given Id, 10 e-mail addresses maximum may be recorded to the ‘To:’ field and 10 e-mail addresses maximum may be recorded to the ‘Cc:’ field of the current e-mail to send.

  RANGE: [1, 10].

- **MailAddr**: The e-mail address (text string up to 120 characters) associated with a given Id and rank.

  For a given Id, by default, the recipient and the Carbon-copy recipient are empty.

  NOTE: Each address must be provided in literal format (for instance dev12345678@wavecom.com).

2.4.1.6 The ed_EmailAddrSet function.

For a given Id, this function sets the ‘To:’ or the ‘Cc:’ recipient according to the boolean bRecCcRecSwitch value (TRUE to set ‘Cc’ field, FALSE to set ‘To:’ field).

- **Prototype**
  
s8 ed_EmailAddrSet(ed_EmailAddrParams_t *Params, bool bRecCcRecSwitch);

- **Parameters**
  
  **Params**: Structure holding the Id, the e-mail address rank and the e-mail address (which may be an empty string). This structure is defined above.

  **bRecCcRecSwitch**: Switch defining the type of e-mail address being set. If FALSE, the e-mail address set is one out of the 10 e-mail addresses that go in the ‘To:’ field of the e-mail sent. If TRUE, the e-mail address set is one out of the 10 e-mail addresses that go in the ‘Cc:’ field of the e-mail to be sent.

- **Returned values**
This function returns 0 on success, or a negative error value.
Possible error values are:
- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_PARAM_BAD_VALUE
- ED_ERR_PARAM_BAD_MAILIDX
- ED_ERR_PARAM_TOO_LONG
- ED_ERR_PARAM_WRITE_E2P
- ED_ERR_INVALID_ARG
- ED_ERR_SERVICE_ALREADY_RUNNING

See §3 - List of error codes and reply codes for more information on error codes.

2.4.1.7 The ed_EmailAddrGet function.
- For a given Id, this function gets the ‘To:’ or the ‘Cc:’ recipient according to
the boolean bRecCcRecSwitch value (TRUE to get the ‘Cc’ field, FALSE to get
the ‘To:’ field).

- **Prototype**
  
s8 ed_EmailAddrGet(ed_EmailAddrParams_t *Params, bool bRecCcRecSwitch);

- **Parameters**
  
  Params:
  IN : Structure holding the Id, the e-mail address rank. This structure is
defined above.
  OUT : Structure holding the Id, the e-mail address rank and the e-mail
address (which may be an empty string).

  bRecCcRecSwitch:
  Switch defining the type of the e-mail address to get. If FALSE, the e-mail
address to get is one out of the 10 e-mail addresses that go in the ‘To:’
field of the e-mail sent. If TRUE, the e-mail address to get is one out of the
10 e-mail addresses that go in the ‘Cc:’ field of the e-mail to be sent.

- **Returned values**
  
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG
  - ED_ERR_PARAM_BAD_VALUE
  - ED_ERR_PARAM_BAD_MAILIDX

See §3 - List of error codes and reply codes for more information on error codes.
2.4.1.8 The ed_SMTPSetupParams_t type
Structure containing the parameters about the SMTP e-mail management.

```
#define SMTP_AUTHSTRMAXSIZE 64
#define SMTP_STRMAXSIZE 120

typedef struct
{
    u16    SmtpPort;
    ascii    SmtpServ[SMTP_STRMAXSIZE+1];
    ascii    Domain[SMTP_STRMAXSIZE+1];
    ascii    SmtpUn[SMTP_AUTHSTRMAXSIZE+1];
    ascii    SmtpPw[SMTP_AUTHSTRMAXSIZE+1];
    ascii    SenderName[SMTP_STRMAXSIZE+1];
    ascii    SenderAddr[SMTP_STRMAXSIZE+1];
} ed_SMTPSetupParams_t;
```

- **SmtpPort**: This parameter contains the port of the SMTP server used for the e-mail sending. It should only be changed upon request of your network administrator as it applies to network infrastructure including firewalls, proxy or specific TCP port translation.
  
  **SYNTAX**: From 1 to 5 numeric digits (0 to 9).
  
  By default, it is set to 25.
  
  **Notes**:
  - numbers above 65,535 are illegal as the port identification fields are 16 bits long in the TCP header.
  - port number 0 is illegal.

- **SmtpServ**: This parameter contains the address of the SMTP server that is to be used to send e-mail messages. In most cases, the local ISP maintains the SMTP server.
  
  **SYNTAX**: 32-bit number in dotted-decimal notation (i.e. xxx.xxx.xxx.xxx) or alphanumeric ASCII text string up to 120 characters if DNS is available.
  
  By default it is set to empty string

- **Domain**: When sending an e-mail message, the TCP/IP stack software must provide the SMTP server with the domain name of the sender. In some cases, this domain name may be different from the domain name included in the sender's e-mail address.
  
  This parameter contains the different domain name (text string up to 120 characters).
  
  By default it is set to empty string
- **SmtpUn:**
  To send e-mail messages, some SMTP servers use an authentication process. In these cases, the TCP/IP stack software will provide the SMTP user name (associated with a SMTP password) for the e-mail sending process. This parameter holds the SMTP user name (ASCII text string up to 64 characters). If this parameter is an empty string, the authentication mode is inactive. If this parameter and the SmtpPw parameter are both not empty, the authentication mode is active.

  **Note:** the IP Connectivity library only supports the LOGIN authentication mechanism.

  By default this parameter is set to empty string

- **SmtpPw:**
  This parameter holds the SMTP password (ASCII text string up to 64 characters) associated to SmtpUn parameter (see above description).

  By default this parameter is set to empty string

- **SenderName:**
  This parameter appears in the field: 'From: ' of the header of the e-mail sent by the TCP/IP stack software. It is the sender’s literal name (up to 120 characters). It is different from the SenderAddr parameter, which is the sender’s e-mail address.

  By default this parameter is set to empty string

  **Note:**
  The address must be provided in literal format (for instance “machine 245”).

- **SenderAddr:**
  This parameter contains the e-mail identification of the hardware platform itself or the optional attached equipment.

  By default this parameter is set to empty string

  **Note:**
  This e-mail address will appear in the header of the e-mail sent by the TCP/IP stack software, in the field "From: "
2.4.1.9 The ed_SMTPSetConfig function
This function sets up some specific parameters about the SMTP e-mail management.

- **Prototype**
  
  ```c
  s8 ed_SMTPSetConfig(ed_SMTPSetupParams_t *Params);
  ```

- **Parameters**
  Structure containing the parameter to set up. This structure is defined above.

- **Returned values**
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_PARAM_BAD_VALUE
  - ED_ERR_PARAM_TOO_LONG
  - ED_ERR_PARAM_WRITE_E2P
  - ED_ERR_INVALID_ARG
  - ED_ERR_SERVICE_ALREADY_RUNNING
  See §3 - List of error codes and reply codes for more information on error codes.

2.4.1.10 The ed_SMTPGetConfig function
This function gets up some specific parameters about the SMTP e-mail management.

- **Prototype**
  
  ```c
  s8 ed_SMTPGetConfig(ed_SMTPSetupParams_t *Params);
  ```

- **Parameters**
  Structure containing the parameter to get. This structure is defined above.

- **Returned values**
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG
  See §3 - List of error codes and reply codes for more information on error codes.
2.4.1.11 The ed_SendMail function

This function sends one of the 3 pre-defined e-mail messages to the SMTP server.

Once an IP link is established, the attached host can direct the TCP/IP stack to send an e-mail message at any time (except when the TCP/IP stack software is already in a process using TCP resources).

The header of this e-mail is built using the REC1/2/3, CCREC1/2/3 and SUBJ1/2/3 parameters while the body is filled in the BODY1/2/3 parameter.

This command is similar to a "send e-mail" operation issued by a standard messaging client on a PC.

- **Prototype**
  
  s8 ed_SendMail (u8 Id, ed_ResponseCbk_f ResponseCbk);

- **Parameters**
  
  Id:
  The number of the pre-defined e-mail to send (from 1 to 3).

  ResponseCbk:
  ResponseCbk is called by the IP Connectivity library after the e-mail has been successfully sent or when an error occurred.

  This callback function is defined as below:

  ```
  typedef void (* ed_ResponseCbk_f) (s32 ResponseCode);
  ```

  ResponseCode values:

  - ED_OK_MAIL_SENT:
  - ED_ERR_DISTANT_DNS:
  - ED_ERR_STACK_INTERNAL:
  - ED_ERR_DISTANT_NO_RESP:
  - ED_ERR_DISTANT_OPEN:
  - ED_ERR_DISTANT_CCREC_ADDR:
  - ED_ERR_DISTANT_REC_ADDR:
  - ED_ERR_DISTANT_SENDER_ADDR:
  - ED_ERR_DISTANT_BODY:
  - ED_ERR_DISTANT_SEND:
  - ED_ERR_DISTANT_CLOSE:
  - ED_ERR_DISTANT_TCP_CLOSED:
  - ED_ERR_DISTANT_TCP_CLOSED_BY_PEER:
  - ED_ERR_DISTANT_USERNAME:
  - ED_ERR_DISTANT_PASSWORD:
• ED_ERR_NETWORK_KO:
  
  GSM network is lost, SMTP e-mail sending is aborted

• ED_ERR_GPRS_SESSION_LOST:
  
  GPRS session is lost, SMTP e-mail sending is aborted.

• Returned values
  
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  • ED_ERR_NO_TCPIP
  • ED_ERR_NOT_INIT
  • ED_ERR_INVALID_ARG
  • ED_ERR_STACK_BAD_CONFIG:
  • ED_ERR_PHY_NOT_ACTIVATED:
  • ED_ERR_STACK_BUSY:

  See §3 - List of error codes and reply codes for more information on error codes.

2.4.1.12 The ed_PutMail function

This function allows the attached application to send an e-mail message containing body text passed to the TCP/IP stack directly with the ed_SendData() function (c.f. section 2.9.1).

Once an IP link is established, the attached host can send an e-mail message at any time (if the TCP/IP stack software is not already in a process using TCP resources).

The header of this e-mail is built using the Rec1 (see § 2.4.1.6), CcRec1 (see § 2.4.1.6) and Subj1 (see § Erreur ! Source du renvoi introuvable.) parameters.

This function is similar to a "send e-mail" operation issued by a standard messaging client on a PC.

• Prototype
  
  s8 ed_PutMail (ed_ResponseCbk_f ResponseCbk, ed_DataRequest_f DataRequest);

• Parameters
  
  ResponseCbk:
  
  ResponseCbk is called by the IP Connectivity library when the SMTP session is opened or when an error occurs.
  
  This callback function is defined as below:

  typedef void (* ed_ResponseCbk_f) (s32 ResponseCode);
ResponseCode values:
  o ED_OK_MAIL_SENT
    The mail has been successfully sent. The PutMail session is over.
  o ED_INFO_WAITING_FOR_DATA
    The TCP/IP stack is ready to send data to SMTP server. TCP/IP stack immediately transfers all the data to the SMTP server (using the DataRequest callback).
  o ED_ERR_Distant_DNS
    The address of the SMTP server has not been resolved by the secondary DNS server. TCP/IP stack is not able to reach the primary and secondary DNS servers or a wrong SMTP server address has been filled in.

Possible error codes are:
  • ED_ERR_STACK_INTERNAL
  • ED_ERR_Distant_NO_RESP
  • ED_ERR_Distant_OPEN
  • ED_ERR_Distant_CCREC_ADD
  • ED_ERR_Distant_REC_ADDR
  • ED_ERR_Distant_SENDER_ADDR
  • ED_ERR_Distant_BODY
  • ED_ERR_Distant_SEND
  • ED_ERR_Distant_CLOSE
  • ED_ERR_Distant_TCP_CLOSED
  • ED_ERR_Distant_TCP_CLOSED_BYPEER
  • ED_ERR_Distant_USERNAME
  • ED_ERR_Distant_PASSWORD
  • ED_ERR_NETWORK_KO
  • ED_ERR_GPRS_SESSION_LOST

DataRequest:
DataRequest is called by the IP Connectivity library to inform the client that it can send new data. It is called as soon as the SMTP session is opened, to start the data transfer.

This callback function is defined as follows:

```c
typedef void (* ed_DataRequest_f) (u16 MaxLen);
```

where `MaxLen` is the maximum number of bytes that can be sent (using the `ed_SendData()` function (c.f. section 2.9.1 below)).
### Note

it is not necessary to call `ed_SendData()` as soon as the DataRequest callback is triggered. The `ed_SendData()` call may occur later.

- **Returned values**
  This function returns 0 on success, or a negative error value.
  Possible error values are:
  - `ED_ERR_NO_TCPIP`
  - `ED_ERR_NOT_INIT`
  - `ED_ERR_INVALID_ARG`
  - `ED_ERR_STACK_BAD_CONFIG`
  - `ED_ERR_PHY_NOT_ACTIVATED`
  - `ED_ERR_STACK_BUSY`

See §3 - List of error codes and reply codes for more information on error codes.

### 2.4.2 POP3 service

The POP3 service handles the retrieving of e-mail from a remote POP3 server.

#### 2.4.2.1 Required Header Files

The interface header file for the POP3 service is:

`ed_pop3.h`

#### 2.4.2.2 The `ed_POP3SetupParams_t` type

Structure containing the parameters about the POP3 e-mail management.

```c
#define POP3_AUTHSTRMAXSIZE 64
#define POP3_STRMAXSIZE 120

typedef struct
{
    u16    Pop3Port;
    bool   bPop3HeaderMode;
    ascii   Pop3Serv[POP3_STRMAXSIZE+1];
    ascii   Pop3Un[POP3_AUTHSTRMAXSIZE+1];
    ascii   Pop3Pw[POP3_AUTHSTRMAXSIZE+1];
} ed_POP3SetupParams_t;
```

- **bPop3HeaderMode**:
  This parameter configures the TCP/IP stack to send or not the POP3 header to the application. The POP3 header contains the From, Cc and Subject fields.
  - 0: the e-mail header will not be sent to the application while retrieving
  - 1: the e-mail header will be sent to the application while retrieving

By default, this parameter is set to 1.
• **Pop3Port**:
  This parameter contains the POP3 server port used for e-mail retrieving.
  By default, this parameter is set to 110.

  **Syntax:**
  5 numeric digits. Note that numbers above 65 535 are illegal as the port identification fields are 16 bits long in the TCP header. Note that port number 0 is illegal too.

  **Note:**
  This parameter should be changed only upon request of your network administrator. It applies to network infrastructures including Firewalls, Proxy or specific TCP port translation.

• **Pop3Serv**:
  This parameter contains the address of the POP3 server that is to be used. The POP3 server must be the one where the specified e-mail account is hosted (which is not necessarily maintained by the local ISP).
  By default this parameter is set to empty string

  **Syntax:**
  32-bit number in dotted-decimal notation (i.e. xxx.xxx.xxx.xxx) or alphanumeric ASCII text string up to 120 characters if DNS is available.

• **Pop3Pw**:
  This parameter contains the password (ASCII text string up to 64 characters) for POP3 account. To retrieve e-mail messages sent to a specified e-mail address, the TCP/IP stack software must know the POP3 password that has been set for that e-mail account.
  By default this parameter is set to empty string

• **Pop3Un**:
  This parameter contains the user name (ASCII text string up to 64 characters) for POP3 account. To retrieve e-mail messages sent to a specified e-mail address, the TCP/IP stack software must know the POP3 user name that has been set for that e-mail account.
  By default this parameter is set to empty string
2.4.2.3 The ed_POP3SetConfig function
This function sets up some specific parameters about the POP3 e-mail management.

- **Prototype**
  
  s8 ed_POP3SetConfig(ed_POP3SetupParams_t *Params);

- **Parameters**
  Structure containing the parameter to set up. This structure is defined above.

- **Returned values**
  This function returns 0 on success, or a negative error value. Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG
  - ED_ERR_PARAM_BAD_VALUE
  - ED_ERR_PARAM_TOO_LONG
  - ED_ERR_PARAM_WRITE_E2P
  - ED_ERR_SERVICE_ALREADY_RUNNING

See §3 - List of error codes and reply codes for more information on error codes.

2.4.2.4 The ed_POP3GetConfig function
This function gets up some specific parameters about the POP3 e-mail management.

- **Prototype**
  
  s8 ed_POP3GetConfig(ed_POP3SetupParams_t *Params);

- **Parameters**
  Structure containing the parameter to get. This structure is defined above.

- **Returned values**
  This function returns 0 on success, or a negative error value. Possible error values are:
  - ED_ERR_NO_TCPIP
  - ED_ERR_NOT_INIT
  - ED_ERR_INVALID_ARG

See §3 - List of error codes and reply codes for more information on error codes.
2.4.2.5 The ed_GetMail function

This function allows the IP connectivity library user to initiate the retrieval of the first mail on the POP3 account specified in the POP3 settings.

Once an IP link is established, the IP connectivity library user can retrieve an e-mail message at any time (except when the TCP/IP stack software is already in a process using TCP resources).

This function is similar to a "check e-mail box" feature issued by a standard messaging client on a PC.

Note that the e-mail is deleted from the POP3 server after retrieval.

- **Prototype**
  
s8 ed_GetMail (ed_ResponseCbk_f ResponseCbk, ed_DataHandler_f DataHnd);

- **Parameters**
  
  ResponseCbk :
  
  ResponseCbk is called by the IP connectivity library when the POP3 connection is opened or when an error occurred.

  This callback function is defined as below :
  
  ```
  typedef void (* ed_ResponseCbk_f) (s32 ResponseCode);
  ```

  ResponseCode values :
  
  - **ED_INFO_MAIL**
    
    This response is issued when at least one e-mail message is located in the specified POP3 mailbox. It is also an indication that the e-mail reception begins. Depending on the bPop3HeaderMode parameter (see §2.4.2.2), the TCP/IP stack sends the e-mail header to the data handler DataHnd.
  
  - **ED_INFO_NOMAIL**
  
  - **ED_OK_POP3**
  
  - **ED_ERR_DISTANT_DNS**
  
  - **ED_ERR_STACK_INTERNAL**
  
  - **ED_ERR_DISTANT_USERNAME**
  
  - **ED_ERR_DISTANT_PASSWORD**
  
  - **ED_ERR_DISTANT_NO_RESP**
  
  - **ED_ERR_DISTANT_OPEN**
  
  - **ED_ERR_DISTANT_CLOSE**
  
  - **ED_ERR_DISTANT_TCP_CLOSED**
  
  - **ED_ERR_DISTANT_TCP_CLOSED_BYPEER**
  
  - **ED_ERR_DISTANT_DATA_RETR**
  
  - **ED_ERR_DISTANT_EMAIL_RETR**
  
  - **ED_ERR_DISTANT_EMAIL_HDR_RETR**
  
  - **ED_ERR_DISTANT_DEL**
DataHnd:
DataHnd is called by the IP connectivity library to send the POP3 received data.
At the end of the e-mail retrieval, IP connectivity library calls pfDataHnd with Data pointer set to NULL and DataLen set to 0.
This callback function is defined as:

\[
\text{typedef u16 (* edDataHandler_f)} (\text{u16 DataLen, u8 * Data});
\]

where:
- DataLen : number of available Bytes in u8 *Data.
- Data : Data pointer; released by IP connectivity library.
- RETURN: Number of bytes “consumed” by the callback. The data left will be handed over at the next call, with possible new data.

**Returned values**
This function returns 0 on success, or a negative error value.
Possible error values are:
- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG
- ED_ERR_STACK_BAD_CONFIG
- ED_ERR_PHY_NOT_ACTIVATED
- ED_ERR_STACK_BUSY

See §3 - List of error codes and reply codes for more information on error codes.
2.5 FTP Service

IP connectivity library provides this service to handle FTP related events and setup FTP parameters.

2.5.1 Required Header Files.
The interface header file for the FTP service is:

```c
ed_ftp.h
```

2.5.2 The ed_FTPSetupParams_t type
Structure containing the parameters about the FTP management.

```c
#define FTP_AUTHSTRMAXSIZE  64
#define FTP_STRMAXSIZE  120

typedef struct
{
    u16      FtpPort;
    ascii    FtpType;
    ascii    FtpServ[FTP_STRMAXSIZE+1];
    ascii    FtpUn[FTP_AUTHSTRMAXSIZE+1];
    ascii    FtpPw[FTP_AUTHSTRMAXSIZE+1];
}ed_FTPSetupParams_t;
```

- **FtpPort**: This parameter contains the control port of the FTP server used for file transfer.
  
  By default, this parameter is set to 21.

  Syntax:

  ```
  numeric 1 to 5 digits. Note that numbers above 65535 are illegal as the port identification fields are 16 bits long in the TCP header. Note that 0 is illegal too.
  ```

  Note:

  This parameter should be changed only upon request of your network administrator. It applies for network infrastructure including Firewalls, Proxy or specific TCP port translation.
• **FtpType:**
  This parameter contains the type of data to be transferred within the FTP session.

  Syntax:
  - A: for FTP ASCII sessions
  - I: for FTP Binary sessions

  By default, this parameter is set to I.

  Note:
  This parameter must be set before transferring files from a specified FTP server.

• **FtpServ:**
  This parameter contains the address of the FTP server that is to be used to download files.

  Syntax:
  32-bit number in dotted-decimal notation (i.e. xxx.xxx.xxx.xxx) or alphanumeric ASCII text string up to 120 characters if DNS is available.

  By default this parameter is set to empty string.

• **FtpUn:**
  This parameter contain the FTP user name of the FTP session (ASCII text string up to 64 characters ).

  By default this parameter is set to empty string.

• **FtpPw:**
  This parameter contain the FTP password of the FTP session (ASCII text string up to 64 characters ).

  By default this parameter is set to empty string.
2.5.3 The ed_FTPSetConfig function
This function sets up some specific parameters about the FTP management.

2.5.3.1 Prototype
s8 ed_FTPSetConfig(ed_FTPSetupParams_t *Params);

2.5.3.2 Parameters
Structure containing the parameter to set up. This structure is defined above.

2.5.3.3 Returned values
This function returns 0 on success, or a negative error value. Possible error values are:
- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG
- ED_ERR_PARAM_BAD_VALUE
- ED_ERR_PARAM_TOO_LONG
- ED_ERR_PARAM_WRITE_E2P

See §3 - List of error codes and reply codes for more information on error codes.

2.5.4 The ed_FTPGetConfig function
This function gets up some specific parameters about the FTP management.

2.5.4.1 Prototype
s8 ed_FTPGetConfig(ed_FTPSetupParams_t *Params);

2.5.4.2 Parameters
Structure containing the parameter to get. This structure is defined above.

2.5.4.3 Returned values
This function returns 0 on success, or a negative error value. Possible error values are:
- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG

See §3 - List of error codes and reply codes for more information on error codes.
2.5.5 The ed_FTPGetFileParams_t type
Structure containing the parameters about the FTP management to get a file from an FTP server.

typedef struct
{
    ascii    FtpGetFilename[FTP_STRMAXSIZE+1];
    ascii    FtpGetPath[FTP_STRMAXSIZE+1];
}ed_FTPGetFileParams_t;

- **FtpGetFilename**: This parameter contains the name (ASCII text string up to 120 characters) of the file to download from the FTP server.

  By default this parameter is set to empty string.

- **FtpGetPath**: This parameter contains the path (ASCII text string up to 120 characters) of the relevant file to download from the FTP server. For example it could be "/wavecom".

  By default this parameter is set to "." (for local directory).

Note:
Depending on the FTP server, the value . can be used for getting a file from the root directory of the FTP server.
2.5.6 The ed_FTPGetFileSetConfig function

This function sets up some specific parameters about the FTP download.

2.5.6.1 Prototype

```c
s8 ed_FTPGetFileSetConfig(ed_FTPGetFileParams_t *Params);
```

2.5.6.2 Parameters

Structure containing the parameter to set up. This structure is defined above.

2.5.6.3 Returned values

This function returns 0 on success, or a negative error value. Possible error values are:
- `ED_ERR_NO_TCPIP`
- `ED_ERR_NOT_INIT`
- `ED_ERR_INVALID_ARG`
- `ED_ERR_PARAM_BAD_VALUE`
- `ED_ERR_PARAM_TOO_LONG`
- `ED_ERR_PARAM_WRITE_E2P`

See §3 - List of error codes and reply codes for more information on error codes.

2.5.7 The ed_FTPGetFileGetConfig function

This function gets some specific parameters about the FTP download.

2.5.7.1 Prototype

```c
s8 ed_FTPGetFileGetConfig(ed_FTPGetFileParams_t *Params);
```

2.5.7.2 Parameters

Structure containing the parameter to get. This structure is defined above.

2.5.7.3 Returned values

This function returns 0 on success, or a negative error value. Possible error values are:
- `ED_ERR_NO_TCPIP`
- `ED_ERR_NOT_INIT`
- `ED_ERR_INVALID_ARG`

See §3 - List of error codes and reply codes for more information on error codes.
2.5.8 The ed_FTPPutFileParams_t type
Structure containing the parameters about the FTP management to put a file on an FTP server.

typedef struct
{
    ascii    FtpPutFilename[FTP_STRMAXSIZE+1];
    ascii    FtpPutPath[FTP_STRMAXSIZE+1];
}ed_FTPPutFileParams_t;

- **FtpPutFilename**:
  This parameter contains the name (ASCII text string up to 120 characters) of the file to upload to the FTP server.

  By default this parameter is set to empty string

- **FtpPutPath**:
  This parameter contains the path (ASCII text string up to 120 characters) of the relevant file to upload to the FTP server. For example, it could be “/wavecom”.

  By default this parameter is set to "."

Note:
Depending on the FTP server, the value . can be used for getting a file from the root directory of the FTP server.
2.5.9 The ed_FTPPutFileSetConfig function

This function sets up some specific parameters about the FTP upload.

2.5.9.1 Prototype

s8 ed_FTPPutFileSetConfig(ed_FTPPutFileParams_t *Params);

2.5.9.2 Parameters

Structure containing the parameter to set up. This structure is defined above.

2.5.9.3 Returned values

This function returns 0 on success, or a negative error value. Possible error values are:

- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG
- ED_ERR_PARAM_BAD_VALUE
- ED_ERR_PARAM_TOO_LONG
- ED_ERR_PARAM_WRITE_E2P

See §3 - List of error codes and reply codes for more information on error codes.

2.5.10 The ed_FTPPutFileGetConfig function

This function gets up some specific parameters about the FTP upload.

2.5.10.1 Prototype

s8 ed_FTPPutFileGetConfig(ed_FTPPutFileParams_t *Params);

2.5.10.2 Parameters

Structure containing the parameter to get. This structure is defined above.

2.5.10.3 Returned values

This function returns 0 on success, or a negative error value. Possible error values are:

- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG

See §3 - List of error codes and reply codes for more information on error codes.
### 2.5.11 The ed_FTPGet function

This function allows the TCP/IP stack to connect to the specified FTP server and to retrieve the specified file from this server. Once the operation has completed, the TCP/IP stack closes the FTP connection.

Once an IP link established, the attached application can retrieve a file from a FTP server at any time (except when the TCP/IP stack software is already in a process using TCP resources).

This command is similar to a GET operation (with an automatic connect/disconnect) issued by a standard FTP client on a PC. The TCP/IP stack handles the global FTP get process by itself.

#### 2.5.11.1 Prototype

```c
s8 ed_FTPGet (ed_ResponseCbk_f ResponseCbk, ed_DataHandler_f DataHnd);
```

#### 2.5.11.2 Parameters

**ResponseCbk**:

ResponseCbk is called by the IP connectivity library once the FTP connection is opened or once the file is fully retrieved or when an error occurs.

This callback function is defined as below:

```c
typedef void (* ed_ResponseCbk_f) (s32 ResponseCode);
```

**ResponseCode values**:

- ED_OK_FILE_TRANSFERED
- ED_INFO_DATA_BEGIN
- ED_ERR_DISTANT_DNS
- ED_ERR_STACK_INTERNAL
- ED_ERR_DISTANT_USERNAME
- ED_ERR_DISTANT_PASSWORD
- ED_ERR_DISTANT_NO_RESP
- ED_ERR_DISTANT_OPEN
- ED_ERR_DISTANT_CD
- ED_ERR_DISTANT_CLOSE
- ED_ERR_DISTANT_TCP_CLOSED
- ED_ERR_DISTANT_TCP_CLOSED_BYPEER
- ED_ERR_DISTANT_DATA_RETR
- ED_ERR_NETWORK_KO
- ED_ERR_GPRS_SESSION_LOST
DataHnd:

DataHnd is called by the IP connectivity library to send the FTP retrieve file.

At the end of FTP received datas, IP connectivity library calls pfDataHnd with Data pointer set to NULL and DataLen set to 0.

This callback function is defined as below:

```c
typedef u16 (* ed_DataHandler_f ) (u16 DataLen, u8 * Data);
```

where:

- DataLen: number of available Bytes in u8 *Data.
- Data: Data pointer; freed by IP connectivity library.
- RETURN: Number of bytes “consumed” by the callback. The data left will be handed over at the next call, maybe with new data.

### 2.5.11.3 Returned values

This function returns 0 on success, or a negative error value.

Possible error values are:

- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG
- ED_ERR_STACK_BAD_CONFIG
- ED_ERR_PHY_NOT_ACTIVATED
- ED_ERR_STACK_BUSY

See §3 - List of error codes and reply codes for more information on error codes.
2.5.12 The ed_FTPPut function

This function allows the TCP/IP stack to connect to the specified FTP server and to upload the data using the ed_SendData() function (see §2.9.1) to the specified file on this server. Once the operation completed, the TCP/IP stack closes the FTP connection.

Once an IP link is established, the attached application can send a file to a FTP server at any time (except when the TCP/IP stack software is already in a process using TCP resources).

This command is similar to a PUT operation (with an automatic connect/disconnect) issued by a standard FTP client on a PC. The TCP/IP stack handles the global FTP put process by itself.

- **Prototype**
  
  ```c
  s8 ed_FTPPut (ed_ResponseCbk_f ResponseCbk, ed_DataRequest_f DataRequest);
  ```

2.5.12.1 Parameters

**ResponseCbk** :

ResponseCbk is called by the IP connectivity library when the FTP session is opened or once the file is fully sent or when an error occurs.

This callback function is defined as below:

```c
typedef void (* ed_ResponseCbk_f) (s32 ResponseCode);
```

**ResponseCode values** :

- ED_OK_FILE_TRANSFERED
- ED_INFO_WAITING_FOR_DATA
- ED_ERR_DISTANT_DNS
- ED_ERR_STACK_INTERNAL
- ED_ERR_DISTANT_NO_RESP
- ED_ERR_DISTANT_OPEN
- ED_ERR_DISTANT_CD
- ED_ERR_DISTANT_SEND
- ED_ERR_DISTANT_CLOSE
- ED_ERR_DISTANT_TCP_CLOSED
- ED_ERR_DISTANT_TCP_CLOSED_BYPEER
- ED_ERR_DISTANT_USERNAME
- ED_ERR_DISTANT_PASSWORD
- ED_ERR_NETWORK_KO
- ED_ERR_GPRS_SESSION_LOST
DataRequest:

DataRequest is called by the IP connectivity library to inform the client that it can send new data.

This function is called when TCP session is opened to start data transfer.

This callback function is defined as below:

```c
typedef void (* ed_DataRequest_f ) (u16 MaxLen);
```

where MaxLen is the maximum number of bytes that can be sent now (using the `ed_SendData()` function).

### 2.5.12.2 Returned values

This function returns 0 on success, or a negative error value. Possible error values are:

- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG
- ED_ERR_STACK_BAD_CONFIG
- ED_ERR_PHY_NOT_ACTIVATED
- ED_ERR_STACK_BUSY

See §3 - List of error codes and reply codes for more information on error codes.
2.6 TCP SOCKET Service

IP connectivity library provides this service to handle TCP Socket related events, and to setup TCP Socket parameters.

2.6.1 Required Header Files.
The interface header file for the TCP SOCKET service is:

\[ ed\_socket.h \]

2.6.2 The ed_SocketSetupParams_t type
Structure containing the parameters about the TCP Socket management.

\#define SOCK\_STRMAXSIZE 120

typedef struct
{
    u16    TcpPort;
    u16    TcpTxDelay;
    ascii    TcpServ[SOCK\_STRMAXSIZE+1];
} ed\_SocketSetupParams\_t;

• **TcpPort** :
  This parameter holds:
  - the remote peer TCP port used for a “client” TCP session (i.e. the IP connectivity library opens a remote listening port)
  - the local listening TCP port for a “server” TCP session (i.e. the IP connectivity library opens a local port in listen mode and waits for a remote peer to connect).

  By default, this parameter is set to 0.

  **Syntax:**
  1 to 5 numeric digits. Note that numbers above 65 535 are illegal as the port identification fields are 16 bits long in the TCP header. Note that port number 0 is illegal too.

• **TcpTxDelay** :
  This parameter holds the delay (expressed in milliseconds) before the socket flushes the characters to transmit to the remote entity.

  **Explanation:** the IP connectivity library user provides data to transmit to the remote by using the \texttt{ed\_SendData()} function (see the pfDataRequest field of the \texttt{ed\_SocketTCPStart()} function for more information on the full mechanism). Suppose the IP connectivity library user calls \texttt{ed\_SendData()} and
provides to it one byte at every call. Internally, the IP connectivity library will not immediately send the byte (it would congest the internet network): it will “wait” until it has a reasonable amount of bytes to transmit. However, if the IP connectivity library user did not give data for TcpTxDelay ms., the IP connectivity library will transmit these bytes, even if the number of bytes is low.

By default, this parameter is set to 100 ms.

Syntax:

[0, 32760] in 20 ms steps.

- **TcpServ**:
  For a “client” TCP session (i.e. the IP connectivity library opens a remote listening port), this parameter contains the address of the remote TCP peer that is to be used. The “255.255.255.255” special IP address is forbidden. For a “server” TCP session (i.e. the IP connectivity library opens a local port in listen mode and waits for a remote peer to connect), this parameter is used to apply filtering of incoming TCP requests from a remote destination IP address. Only requests from the configured IP address will be allowed to connect to the Wavecom module. In case of no Filtering the TcpServ addr must be set to the “255.255.255.255” value. “0.0.0.0” is an invalid IP address for both “client” and “server” configuration.

Syntax:

32-bit number in dotted-decimal notation (i.e. xxx.xxx.xxx.xxx) or alphanumeric ASCII text string up to SOCK_STRMAXSIZE characters if DNS is integrated.

By default, this parameter is set to empty string.
2.6.3 The ed_SocketSetConfig function
This function sets up some specific parameters about the TCP Socket management.

2.6.3.1 Prototype
s8 ed_SocketSetConfig(ed_SocketSetupParams_t *Params);

2.6.3.2 Parameters
Params :
Structure containing the parameter to set up. This structure is defined above.

2.6.3.3 Returned values
This function returns 0 on success, or a negative error value. Possible error values are:
- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG
- ED_ERR_PARAM_BAD_VALUE
- ED_ERR_PARAM_TOO_LONG
- ED_ERR_PARAM_WRITE_E2P

See §3 - List of error codes and reply codes for more information on error codes.
2.6.4 The ed_SocketGetConfig function
This function gets up some specific parameters about the Socket management.

2.6.4.1 Prototype

\[\text{s8 ed_SocketGetConfig(ed_SocketSetupParams_t *Params);}\]

2.6.4.2 Parameters

Params:
Structure containing the parameter to get. This structure is defined above.

2.6.4.3 Returned values
This function returns 0 on success, or a negative error value.
Possible error values are:
- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG

See §3 - List of error codes and reply codes for more information on error codes.

2.6.5 The ed_SocketTCPStart function
This function is used to open a remote TCP socket or to start a listening TCP socket, if the device is connected.

Once an IP link is established, the attached application can open a listening TCP socket at any time (except when the TCP/IP stack software is already in a process using TCP resources).

If this function was used to start a listening TCP socket (listen==1), the TCP connection will be active upon reception of a TCP connection request sent by a remote allowed TCP peer (TcpServ) on the appropriate TCP port (TcpPort).

2.6.5.1 Prototype

\[\text{s8 ed_SocketTCPStart(u8 Listen} \]
\[\quad\text{ed_ResponseCbk_f pfResponseCbk,} \]
\[\quad\text{ed_DataHandler_f pfDataHnd,} \]
\[\quad\text{ed_DataRequest_f pfDataRequest);}\]

2.6.5.2 Parameters

Listen:
set to 1 if a listening TCP socket is to be started
Else IP connectivity library tries to open a TCP connection to the specified TCP server.
pfResponseCbk:

pfResponseCbk is called by the IP connectivity library when the Socket TCP connection is opened or when an error occurred.

This callback function is defined as below:

\[
\text{typedef void (* ed\_ResponseCbk\_f) (s32 ResponseCode);}\
\]

ResponseCode values:

- ED_INFO_LISTEN_SET
- ED_OK_TCP_CLOSED
- ED_INFO_WAITING_FOR_DATA
- ED_ERR_DISTANT_DNS
- ED_ERR_DISTANT_OPEN
- ED_ERR_STACK_INTERNAL
- ED_ERR_LISTEN_STOP
- ED_ERR_DISTANT_NO_RESP
- ED_ERR_DISTANT_SEND
- ED_ERR_DISTANT_CLOSE
- ED_ERR_DISTANT_TCP_CLOSED
- ED_ERR_DISTANT_TCP_CLOSED_BY_PEER
- ED_ERR_NETWORK_KO
- ED_ERR_GPRS_SESSION_LOST

pfDataHnd:

pfDataHnd is called by the IP connectivity library to send data received from the remote to the application.

When the socket is closed, IP connectivity library calls pfDataHnd with Data pointer set to NULL and DataLen set to 0.

This callback function is defined as below:

\[
\text{typedef u16 (* ed\_DataHandler\_f ) (u16 DataLen, u8 * Data);}\
\]

where:

- DataLen : number of available Bytes in u8 *Data.
- Data : Data pointer; freed by IP connectivity library.
- RETURN: Number of bytes “consumed” by the callback. The data left will be handed over at the next call, maybe with new data.
pfDataRequest:
pfDataRequest is called by the IP connectivity library to inform the application that it can send new data.
This function is called by the IP connectivity library as soon as the TCP session is opened to initiate the data transfer.
This callback function is defined as below:

```c
typedef void (* ed_DataRequest_f) (u16 MaxLen);
```
where MaxLen is the maximum number of bytes that can be sent now (using the `ed_SendData()` function).

### 2.6.5.3 Returned values

This function returns 0 on success, or a negative error value.
Possible error values are:
- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG
- ED_ERR_STACK_BAD_CONFIG
- ED_ERR_PHY_NOT_ACTIVATED
- ED_ERR_STACK_BUSY

See §3 - List of error codes and reply codes for more information on error codes.

### 2.6.6 The ed_SocketLTCPStop function

This function allows to close a TCP socket in listening mode. It should have been previously launched by the ed_SocketTCPStart command with listen parameter set to 1).

#### 2.6.6.1 Prototype

```c
s8 ed_SocketLTCPStop (ed_ResponseCbk_f pfResponseCbk);
```

#### 2.6.6.2 Parameters

pfResponseCbk:
pfResponseCbk is called by the IP connectivity library when the Socket TCP connection is closed or when an error occurred.
This callback function is defined as below:

```c
typedef void (* ed_ ResponseCbk_f) (s32 ResponseCode);
```

ResponseCode value:
- ED_OK_TCP_CLOSED
2.6.6.3 Returned values

This function returns 0 on success, or a negative error value. Possible error values are:

- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG
- ED_ERR_NO_LISTEN

See §3 - List of error codes and reply codes for more information on error codes.
2.7 UDP SOCKET Service

The IP Connectivity library provides this service to handle UDP Socket related events, and to setup UDP Socket parameters.

2.7.1 Required Header Files.

The interface header file for the UDP SOCKET service is:

```
ed_udpsocket.h```

2.7.2 The ed_UdpSocketSetupParams_t type

Structure containing the parameters about the UDP Socket management.

```
#define UDPSOCK_STRMAXSIZE 120

typedef struct {
    u16    UdpPort;
    u16    UdpTxDelay;
    ascii   UdpServ[UDPSOCK_STRMAXSIZE+1];
}ed_UdpSocketSetupParams_t;
```

- **UdpPort**
  - This parameter holds:
  - the remote peer UDP port used for a “client” UDP session (i.e. the IP Connectivity library user may be the first to transmit to the UdpServ / UdpPort address)
  ```
  OR
  ```
  - the local listening UDP port for a “server” UDP session (i.e. the IP Connectivity library opens a local port in listen mode and waits for a remote peer, filtered by the UdpServ parameter, to send a first datagram).

By default, this parameter is set to 0.

SYNTAX:

1 to 5 numeric digits (between 0 and 9). Numbers above 65,535 are illegal as the port identification fields are 16 bits long in the TCP header. Port number 0 is also illegal.

- **UdpTxDelay**
  - This parameter holds the delay (expressed in ms) before the socket flushes the characters to transmit to the remote entity.

  The IP Connectivity library user provides data to transmit to the remote by using the `ed_SendData()` function (see the pfDataRequest field of the
ed_SocketUDPStart() function for information on the full mechanism). Suppose the IP Connectivity library user calls ed_SendData() and provides it with data Byte by byte. Internally, the IP Connectivity library will not immediately send the byte since this would congest the internet network. It will “wait” until it has a “reasonable” amount of Bytes to transmit. However, if the IP Connectivity library user does not give more data for UdpTxDelay ms, the IP Connectivity library will transmit data already provided.

By default, this parameter is set to 100 ms.

SYNTAX:
[0, 32760] in 20 ms steps.

- **UdpServ**
  For a “client” UDP session (the IP Connectivity library user may be the first to transmit to the UdpServ/UdpPort address), this parameter contains the address of the remote UDP peer that should be used. The “255.255.255.255” special IP address is forbidden.
  For a “server” UDP session (the IP Connectivity library opens a local port in listen mode and waits for a remote peer to send a first datagram), this parameter is used to apply filtering of incoming UDP datagrams from a remote destination IP address. Only datagrams from the configured IP address will be allowed to connect to the Wavecom module. In case of no filtering the UdpServ parameter must be set to the “255.255.255.255” value. “0.0.0.0” is an invalid IP address for both “client” and “server” configuration.

SYNTAX:
32-bit number in dotted-decimal notation (i.e. xxx.xxx.xxx.xxx)
or alphanumeric ASCII text string up to UDPSOCK_STRMAXSIZE characters if DNS is integrated.

By default, this parameter is set to empty string.
2.7.3 The ed_UdpSocketSetConfig function

This function sets up some specific parameters about the UDP Socket management.

2.7.3.1 Prototype

```c
s8 ed_UdpSocketSetConfig(ed_UdpSocketSetupParams_t *Params);
```

2.7.3.2 Parameters

Params :
Structure containing the parameter to set up. This structure is defined above.

2.7.3.3 Returned values

This function returns 0 on success, or a negative error value. Possible error values are:
- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_PARAM_BAD_VALUE
- ED_ERR_PARAM_TOO_LONG
- ED_ERR_PARAM_WRITE_E2P
- ED_ERR_INVALID_ARG
- ED_ERR_SERVICE_ALREADY_RUNNING

See §3 - List of error codes and reply codes for more information on error codes.
2.7.4 The ed_UdpSocketGetConfig function
This function gets up some specific parameters about the UDP Socket management.

2.7.4.1 Prototype

s8 ed_UdpSocketGetConfig(ed_UdpSocketSetupParams_t *Params);

2.7.4.2 Parameters

Params :
Structure containing the parameter to get. This structure is defined above.

2.7.4.3 Returned values

This function returns 0 on success, or a negative error value. Possible error values are :
- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG

See §3 - List of error codes and reply codes for more information on error codes.

2.7.5 The ed_SocketUDPStart function

This function is used to create a UDP session in active mode or in listen mode, if the device is connected.

Once an IP link is established, the attached application can initiate a UDP session at any time (except if one of the IP Connectivity library service is already in use).

If this function is used to start a listening UDP socket (Listen parameter equal to 1), the UDP session will be active upon reception of a first datagram sent by a remote allowed UDP peer (UdpServ used as IP filter) on the appropriate local UDP port (UdpPort).

If this function is used to start an active UDP session(also called client session), the UDP session is effective as soon as the IP Connectivity library replies that the UDP session was successfully started.

Note that an effective UDP session (datagrams have already been exchanged) can only be stopped by the IP Connectivity library user (using the ed_SendData() function).

Note that a UDP session that is in listen mode can be stopped using the ed_SocketLUDPStop() function.
2.7.5.1 Prototype

```c
s8 ed_SocketUDPStart ( u8 Listen,
    ed_ResponseCbk_f pfResponseCbk,
    ed_DataHandler_f pfDataHnd,
    ed_DataRequest_f pfDataRequest);
```

2.7.5.2 Parameters

Listen:
- Set to 1 if a listening UDP socket is to be started. `UdpPort` is the local port number and it must not be 0(zero). `UdpServ` is used as an IP address filter. The UDP session will be effective upon reception of the first datagram.
- Else if an active UDP session is to be started. `UdpPort` is the remote port number and it must not be 0(zero). `UdpServ` is the remote peer IP address.

`pfResponseCbk`:  
`pfResponseCbk` is called by the IP Connectivity library when the Socket UDP session is started or when an error occurred. 
This callback function is defined as below:

```c
typedef void (* ed_ResponseCbk_f) (s32 ResponseCode);
```

ResponseCode values:
- `ED_INFO_LISTEN_SET`
- `ED_OK_UDP_CLOSED`
- `ED_INFO_WAITING_FOR_DATA`
- `ED_ERR_DISTANT_DNS`
- `ED_ERR_STACK_INTERNAL`
- `ED_ERR_LISTEN_STOP`
- `ED_ERR_NETWORK_KO`
- `ED_ERR_GPRS_SESSION_LOST`

`pfDataHnd`:
`pfDataHnd` is called by the IP Connectivity library to forward data received from the remote, to the application.

When the socket is closed, IP Connectivity library calls `pfDataHnd` with Data pointer set to NULL and DataLen set to 0.
This callback function is defined as below:

```c
typedef u16 (* ed_DataHandler_f ) (u16 DataLen, u8 * Data);
```

where:

- **Data**: Data pointer; freed by IP Connectivity library.
- **DataLen**: number of available bytes in **Data**.
- **RETURN**: Number of bytes “consumed” by the callback. Data left will be handed over at the next call, possibly with additional data.

**pfDataRequest**:

pfDataRequest is called by the IP Connectivity library to inform the application that it can send new data. It is called as soon as the UDP session is effective, to initiate the data transfer.

This callback function is defined as below:

```c
typedef void (* ed_DataRequest_f) (u16 MaxLen);
```

where **MaxLen** is the maximum number of bytes that can be sent now (using the `ed_SendData()` function (c.f. section 2.9.1 below)).

### 2.7.5.3 Returned values

This function returns 0 on success, or a negative error value.

Possible error values are:

- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG
- ED_ERR_STACK_BAD_CONFIG
- ED_ERR_PHY_NOT_ACTIVATED
- ED_ERR_STACK_BUSY

See §3 - List of error codes and reply codes for more information on error codes.
2.7.6 The ed_SocketLUDPStop function
This function allows to close a UDP socket in listening mode (previously launched by the ed_SocketUDPStart() command).

2.7.6.1 Prototype

\[
s8 \text{ ed\_SocketLUDPStop (ed\_ResponseCbk\_f pfResponseCbk)};
\]

2.7.6.2 Parameters

pfResponseCbk :

pfResponseCbk is called by the IP Connectivity library when the Socket UDP is closed or when an error occurred.

This callback function is defined as below:

\[
\text{typedef void (* ed\_ResponseCbk\_f) (s32 ResponseCode)};
\]

ResponseCode value:

- ED_OK_UDP_CLOSED

2.7.6.3 Returned values

This function returns 0 on success, or a negative error value.

Possible error values are:

- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG
- ED_ERR_NO_LISTEN

See §3 - List of error codes and reply codes for more information on error codes.
2.8 PING Service

The IP Connectivity library provides this service to issue PING echo requests, manage the corresponding echo replies, and setup PING parameters.

2.8.1 Required Header Files.
The interface header file for the PING service is:

    ed_ping.h

2.8.2 The ed_PINGSetupParams_t type
Structure containing the parameters about the PING management.

#define PING_STRMAXSIZE 120

typedef struct
{
    u8    PingDelay;
    ascii    PingRemote[PING_STRMAXSIZE+1];
}ed_PINGSetupParams_t;

• PingDelay :
  This parameter holds the maximum delay, in seconds, for which the ping task will wait for an echo reply. If this delay expires, the echo is considered as not answered and the ping task exits.

  SYNTAX :
  Integer in the range [0, 255]. Default value is 1.

• PingRemote :
  This parameter holds the address of the host to ping.

  SYNTAX :
  32-bit number in dotted-decimal notation (i.e. xxx.xxx.xxx.xxx)
  or alphanumeric ASCII text string up to PING_STRMAXSIZE characters if DNS is integrated.

  By default, this parameter is set to an empty string. Note that “0.0.0.0” is an invalid IP address.
2.8.3 The ed_PINGSetConfig function

This function sets up some specific parameters about the PING management.

2.8.3.1 Prototype

s8 ed_PINGSetConfig(ed_PINGSetupParams_t *Params);

2.8.3.2 Parameters

Params :
Structure containing the parameter to set up. This structure is defined above.

2.8.3.3 Returned values

This function returns 0 on success, or a negative error value. Possible error values are:
- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_PARAM_BAD_VALUE
- ED_ERR_PARAM_TOO_LONG
- ED_ERR_PARAM_WRITE_E2P
- ED_ERR_INVALID_ARG
- ED_ERR_SERVICE_ALREADY_RUNNING

See §3 - List of error codes and reply codes for more information on error codes.
2.8.4 The ed_PINGGetConfig function

This function gets the parameters of the PING management.

2.8.4.1 Prototype

s8 ed_PINGGetConfig(ed_PINGSetupParams_t *Params);

2.8.4.2 Parameters

Params :
Structure containing the parameter to get. This structure is defined above.

2.8.4.3 Returned values

This function returns 0 on success, or a negative error value. Possible error values are:
- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT
- ED_ERR_INVALID_ARG

See §3 - List of error codes and reply codes for more information on error codes.
2.8.5 The ed_PingEchoRequest function

If the device is connected, this function issues one echo request to the PingRemote host and start a task waiting for the echo reply. The echo request is made of 40 ICMP bytes (32 data bytes + 8 ICMP header bytes).

The ÍNG service returns ED_OK_ECHO_REPLY and exits if the echo reply is received within PingDelay seconds. The service returns ED_ERR_NO_REPLY and exits if PingDelay expired or if the ed_PingReset() function was called.

Note that the PING task can be stopped using the ed_PingReset() function.

2.8.5.1 Prototype

`s8 ed_PingEchoRequest (ed_ResponseCbk_f pfResponseCbk);`

2.8.5.2 Parameters

`pfResponseCbk :`

`pfResponseCbk` is called by the IP Connectivity library when a reply is received, or if the delay expired, or when an error occurred.

This callback function is defined as below:

```c
typedef void (* ed_ResponseCbk_f) (s32 ResponseCode);
```

ResponseCode values :

- **ED_INFO_ECHO_REQUEST**
- **ED_OK_ECHO_REPLY**
- **ED_ERR_DISTANT_DNS**
- **ED_ERR_STACK_INTERNAL**
- **ED_ERR_NO_REPLY**
- **ED_ERR_NETWORK_KO**
- **ED_ERR_GPRS_SESSION_LOST**

2.8.5.3 Returned values

This function returns 0 on success, or a negative error value. Possible error values are :

- **ED_ERR_NO_TCPIP**
- **ED_ERR_NOT_INIT**
- **ED_ERR_INVALID_ARG**
- **ED_ERR_STACK_BAD_CONFIG**
- **ED_ERR_PHY_NOT_ACTIVATED**
- **ED_ERR_STACK_BUSY**

See §3 - List of error codes and reply codes for more information on error codes.
2.8.6 The ed_PingReset function

This is a two purposes function:

1. It resets the ICMP seqnum header field to 0. In a context of a PING application, it is relevant to call this function before issuing the first call to ed_PingEchoRequest(). Everytime a Ping echo request is sent, the ICMP seqnum (sequence number) is incremented.

2. It stops the PING task if it is active.

2.8.6.1 Prototype

```
s8 ed_PingReset(void);
```

2.8.6.2 Returned values

This function returns 0 on success, or a negative error value. Possible error values are:

- ED_ERR_NO_TCPIP
- ED_ERR_NOT_INIT

See §3 - List of error codes and reply codes for more information on error codes.
2.9 Data Flows Management

IP connectivity library provides this service to handle the transmission data flows.

2.9.1 Function eD_SendData

This function is used to send data with the TCP/IP stack.

2.9.1.1 Prototype

\[ \text{s8 } \text{ed}_\text{SendData}(\text{u8 } \ast \text{Data}, \text{u16 DataLen}, \text{bool bEnd}); \]

2.9.1.2 Parameters

- \text{u8 } \ast \text{Data}
  
  Pointer on the data to send.
  
  Note: Data pointer must be freed by the application

- \text{u16 DataLen}
  
  Size of the data to send. It must be less or equal to \text{MaxLen} parameter given with the previous \text{ed_DataRequest_f} callback call.

- \text{bool bEnd}
  
  When set to TRUE, this flag is used to indicate to the IP connectivity library that it receives the last block of data to send for the current session. Once this boolean indicator is set, the IP connectivity library will not request data anymore.

2.9.1.3 Returned values

This function returns 0 on success, or a negative error value. Possible error values are:

- \text{ED_ERR_NO_TCPIP}
- \text{ED_ERR_NOT_INIT}
- \text{ED_ERR_INVALID_ARG}
- \text{ED_ERR_INVALID_CALL}
- \text{ED_ERR_INVALID_SIZE}

See §3 - List of error codes and reply codes for more information on error codes.
# 3 List of error codes and reply codes

<table>
<thead>
<tr>
<th>Codes</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ED_ERR_NO_TCPIP</td>
<td>-1</td>
<td>TCP/IP feature is not activated in the WISMO product.</td>
</tr>
<tr>
<td>ED_ERR_TCPIP_ALREADYUSED</td>
<td>-2</td>
<td>Another application is already using IP connectivity library</td>
</tr>
<tr>
<td>ED_ERR_PARAM_WRITE_E2P</td>
<td>-3</td>
<td>Parameter writing into E2P failed</td>
</tr>
<tr>
<td>ED_ERR_PARAM_READ_E2P</td>
<td>-4</td>
<td>Parameter reading from E2P failed</td>
</tr>
<tr>
<td>ED_ERR_PARAM_BAD_VALUE</td>
<td>-5</td>
<td>Parameter value is invalid</td>
</tr>
<tr>
<td>ED_ERR_PARAM_TOO_LONG</td>
<td>-6</td>
<td>Parameter string is too long</td>
</tr>
<tr>
<td>ED_ERR_INVALID_ARG</td>
<td>-7</td>
<td>Argument is invalid (or NULL pointer)</td>
</tr>
<tr>
<td>ED_ERR_MODEM_RUNNING</td>
<td>-8</td>
<td>WISMO has already a physical connection active. IP connectivity library can not be used</td>
</tr>
<tr>
<td>ED_ERR_INVALID_RING_TYPE</td>
<td>-9</td>
<td>RING is a Voice ring. Voice incoming calls are not accepted by IP connectivity library</td>
</tr>
<tr>
<td>ED_ERR_PIN_NOT_ENTERED</td>
<td>-10</td>
<td>PIN code has not been entered by user</td>
</tr>
<tr>
<td>ED_ERR_PHY_NOT_ACTIVATED</td>
<td>-11</td>
<td>The IP connectivity library network connection is not active</td>
</tr>
<tr>
<td>ED_ERR_STACK_BAD_CONFIG</td>
<td>-12</td>
<td>IP connectivity library application has an invalid configuration</td>
</tr>
<tr>
<td>ED_ERR_STACK_BUSY</td>
<td>-13</td>
<td>Another IP connectivity library application is already running</td>
</tr>
<tr>
<td>ED_ERR_NETWORK_KO</td>
<td>-14</td>
<td>WISMO module is not attached to the network. Network was lost before the requested end of connection.</td>
</tr>
<tr>
<td>ED_ERR_SIM_REMOVED</td>
<td>-15</td>
<td>SIM card is removed before the requested end of connection.</td>
</tr>
<tr>
<td>ED_ERR_ON_HOOK</td>
<td>-16</td>
<td>internal On-Hook command returns ERROR</td>
</tr>
<tr>
<td>ED_ERR_BUSY</td>
<td>-17</td>
<td>A busy signal is detected on the remote site</td>
</tr>
<tr>
<td>ED_ERR_NO CARRIER</td>
<td>-18</td>
<td>The modem handshaking process with the remote host is interrupted or unsuccessful</td>
</tr>
<tr>
<td>ED_ERR PPP ERROR</td>
<td>-19</td>
<td>The PPP negotiation failed (check ISPUN, ISPPW (see §2.3.2.2) and PPPMODE (see §2.3.3.2))</td>
</tr>
<tr>
<td>ED_ERR PPP TIMEOUT</td>
<td>-20</td>
<td>PPP negotiation timeout expired</td>
</tr>
<tr>
<td>Codes</td>
<td>Numeric Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ED_ERR_INVALID_EVENT</td>
<td>-21</td>
<td>An invalid event occurred during physical activation. The activation is aborted</td>
</tr>
<tr>
<td>ED_ERR_PHY_TIMEOUT</td>
<td>-22</td>
<td>Inactivity timeout expired. Physical connection is closed</td>
</tr>
<tr>
<td>ED_ERR_GPRS_ABORTED</td>
<td>-23</td>
<td>An internal GPRS command error occurred. The activation is aborted</td>
</tr>
<tr>
<td>ED_ERR_WM_UNABLE_SUBS_EVENT</td>
<td>-24</td>
<td>Subscription of command, timer or unsolicited message failed</td>
</tr>
<tr>
<td>ED_ERR_WM_FLOW_OPEN</td>
<td>-25</td>
<td>Unable to open internal data flows</td>
</tr>
<tr>
<td>ED_ERR_WM_FLOW_CLOSE</td>
<td>-26</td>
<td>Unable to close internal data flows</td>
</tr>
<tr>
<td>ED_ERR_WM_PARSER_FLOW</td>
<td>-27</td>
<td>Timeout on reception of flows event</td>
</tr>
<tr>
<td>ED_ERR_WM_GPRS_OPEN</td>
<td>-28</td>
<td>Open GPRS internal session failed</td>
</tr>
<tr>
<td>ED_ERR_WM_GPRS_AUTH</td>
<td>-29</td>
<td>GPRS authentication failed</td>
</tr>
<tr>
<td>ED_ERR_WM_GPRS_IPCP</td>
<td>-30</td>
<td>Error when IP connectivity library get GPRS IPCP data</td>
</tr>
<tr>
<td>ED_ERR_STACK_INTERNAL</td>
<td>-31</td>
<td>IP connectivity library internal error</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_DNS</td>
<td>-32</td>
<td>Remote server name has not been mapped by DNS. TCP/IP stack is not able to reach the primary and secondary DNS servers or a wrong server address has been filled in.</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_NO_RESP</td>
<td>-33</td>
<td>No response from remote server.</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_OPEN</td>
<td>-34</td>
<td>Error during IP connectivity library application session opening</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_CLOSE</td>
<td>-35</td>
<td>Error during IP connectivity library application session closing</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_CCREC_ADDR</td>
<td>-36</td>
<td>CCREC e-mail address(es) is rejected by remote SMTP server</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_REC_ADDR</td>
<td>-37</td>
<td>REC e-mail address(es) is rejected by remote SMTP server</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_SENDER_ADDR</td>
<td>-38</td>
<td>Sender e-mail address(es) is rejected by remote SMTP server</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_BODY</td>
<td>-39</td>
<td>e-mail body to send is rejected by remote SMTP server</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_SEND</td>
<td>-40</td>
<td>Data sending attempt failed</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_TCP_CLOSED</td>
<td>-41</td>
<td>TCP session is closed by IP connectivity library</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_TCP_CLOSED_BY_PEER</td>
<td>-42</td>
<td>TCP session is closed by remote server</td>
</tr>
<tr>
<td>Codes</td>
<td>Numeric Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_USERNAME</td>
<td>-43</td>
<td>Username is rejected by remote SMTP server</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_PASSWORD</td>
<td>-44</td>
<td>Password is rejected by remote SMTP server</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_DATA_RETR</td>
<td>-45</td>
<td>Remote server rejected data retrieval request</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_EMAIL_RETR</td>
<td>-46</td>
<td>E-mail retrieval failed</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_EMAIL_HDR_RETR</td>
<td>-47</td>
<td>E-mail header retrieval failed</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_DEL</td>
<td>-48</td>
<td>Deletion of e-mail rejected by remote server</td>
</tr>
<tr>
<td>ED_ERR_DISTANT_CD</td>
<td>-49</td>
<td>Change directory on remote server failed</td>
</tr>
<tr>
<td>ED_ERR_INVALID_SIZE</td>
<td>-50</td>
<td>The DataLen parameter of the ed_SendData() function is greater than expected.</td>
</tr>
<tr>
<td>ED_ERR_INVALID_CALL</td>
<td>-51</td>
<td>The ed_SendData() function can not be called now (no matching ed_DataRequest_f callback call was issued or because no IP connectivity library service is currently active).</td>
</tr>
<tr>
<td>ED_ERR_NOT_INIT</td>
<td>-52</td>
<td>The IP connectivity library can not be used because its init function ed_Init() has not been called yet.</td>
</tr>
<tr>
<td>ED_ERR_LISTEN_STOP</td>
<td>-53</td>
<td>The listening TCP socket has been canceled.</td>
</tr>
<tr>
<td>ED_ERR_NO_LISTEN</td>
<td>-54</td>
<td>ed_SocketLTCPStop() has been called while no TCP socket was opened in listen mode.</td>
</tr>
<tr>
<td>ED_ERR_NO_GPRS</td>
<td>-55</td>
<td>GPRS feature unavailable.</td>
</tr>
<tr>
<td>ED_ERR_NO_ANSWER</td>
<td>-56</td>
<td>There is no response from the remote site</td>
</tr>
<tr>
<td>ED_ERR_CLI_INVALID</td>
<td>-57</td>
<td>CallerID does not provide caller phone number or it is invalid</td>
</tr>
<tr>
<td>ED_ERR_NO_REPLY</td>
<td>-58</td>
<td>No echo reply received (PING service).</td>
</tr>
<tr>
<td>ED_ERR_PARAM_BAD_MAILIDX</td>
<td>-59</td>
<td>E-mail address index is invalid.</td>
</tr>
<tr>
<td>ED_ERR_GPRS_SESSION_LOST</td>
<td>-60</td>
<td>The GPRS session is lost(context deactivated).</td>
</tr>
<tr>
<td>ED_ERR_SERVICE_ALREADY_RUNNING</td>
<td>-61</td>
<td>It is not possible to set new parameter value because the associated service is already in use.</td>
</tr>
</tbody>
</table>
## Codes

<table>
<thead>
<tr>
<th>Codes</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ED_INFO_CONNECT</td>
<td>100</td>
<td>Modem is connected</td>
</tr>
<tr>
<td>ED_INFO_WAITING_FOR_DATA</td>
<td>107</td>
<td>IP connectivity library is waiting for data to send</td>
</tr>
<tr>
<td>ED_INFO_MAIL</td>
<td>108</td>
<td>Start of the e-mail data retrieval from the mailbox</td>
</tr>
<tr>
<td>ED_INFO_NOMAIL</td>
<td>109</td>
<td>No new e-mail in the mailbox</td>
</tr>
<tr>
<td>ED_INFO_DATA_BEGIN</td>
<td>110</td>
<td>Start of the data retrieval</td>
</tr>
<tr>
<td>ED_INFO_LISTEN_SET</td>
<td>111</td>
<td>The listening TCP socket has been successfully created. It is now waiting for a remote entity to connect.</td>
</tr>
<tr>
<td>ED_INFO_ECHO_REQUEST</td>
<td>112</td>
<td>The PING echo request is being sent</td>
</tr>
</tbody>
</table>

## Codes

<table>
<thead>
<tr>
<th>Codes</th>
<th>Numeric Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ED_OK_PPP</td>
<td>1</td>
<td>PPP negotiation is OK. As soon as the TCP/IP stack software displays this message, it is ready to receive commands.</td>
</tr>
<tr>
<td>ED_OK_GPRS_SESSION_SET</td>
<td>2</td>
<td>GPRS session is activated</td>
</tr>
<tr>
<td>ED_OK_ON_HOOK</td>
<td>3</td>
<td>Connection is correctly released</td>
</tr>
<tr>
<td>ED_OK_MAIL_SENT</td>
<td>4</td>
<td>ed_SendMail or ed_PutMail has successfully completed</td>
</tr>
<tr>
<td>ED_OK_TCP_CLOSED</td>
<td>5</td>
<td>ed_SocketTCPStart has successfully completed. The TCP Socket is closed correctly.</td>
</tr>
<tr>
<td>ED_OK_POP3</td>
<td>6</td>
<td>ed_GetMail has successfully completed. This response is issued when the POP3 session is gracefully closed.</td>
</tr>
<tr>
<td>ED_OK_FILE_TRANSFERED</td>
<td>7</td>
<td>ed_FTPGet or ed_FTPPut have successfully completed. The FTP transfer file has finished successfully.</td>
</tr>
<tr>
<td>ED_OK_UDP_CLOSED</td>
<td>8</td>
<td>The UDP socket was correctly closed</td>
</tr>
<tr>
<td>ED_OK_ECHO_REPLY</td>
<td>9</td>
<td>Echo reply received (PING service).</td>
</tr>
</tbody>
</table>
4 Appendix: Open AT Resources

Constraints about the use of common resources for the whole Open AT software are expressed hereafter.

4.1 EEPROM objects

Open AT IP Connectivity library uses EEPROM objects whose identifiers are between 1 and 1000.

4.2 Timers

5 timers are used by the IP Connectivity library.

4.3 RAM and ROM occupation

<table>
<thead>
<tr>
<th>RAM</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>820 bytes (RW)</td>
<td></td>
</tr>
<tr>
<td>40113 bytes (ZI)</td>
<td></td>
</tr>
<tr>
<td>40933 bytes</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>53276 bytes (code)</td>
<td></td>
</tr>
<tr>
<td>1226 bytes (RO)</td>
<td></td>
</tr>
<tr>
<td>820 bytes (RW)</td>
<td></td>
</tr>
<tr>
<td>55322 bytes</td>
<td></td>
</tr>
</tbody>
</table>

Notes:

- These values do not include Open AT or ADL libraries resource allocation.
- Some received AT commands can lead to additional memory allocation. These commands are +CREG, +CGREG and +CLIP. The allocated memory in this case is less than 10 bytes (size of the received string).