

IPC@CHIP®
UDP Configuration Server Protocol Specification
Version 2.0

TABLE OF CONTENTS

| | | |
|-------|---|----|
| 1 | INTRODUCTION | 3 |
| 2 | GENERAL INFORMATION | 3 |
| 2.1 | PACKET FORMAT | 3 |
| 3 | COMMANDS | 4 |
| 3.1 | HELLO | 4 |
| 3.1.1 | Request..... | 4 |
| 3.1.2 | Reply..... | 5 |
| 3.2 | IP CONFIGURATION..... | 7 |
| 3.2.1 | Request..... | 7 |
| 3.2.2 | Reply..... | 10 |
| 3.3 | USER CALLBACK..... | 10 |
| 3.3.1 | Request..... | 10 |
| 3.3.2 | Reply..... | 11 |
| 3.3.3 | Callback function | 11 |
| A | COMPLETE EBNF DEFINITION OF THE PROTOCOL..... | 13 |
| B | HISTORY..... | 14 |

1 Introduction

The UDP configuration server provides means for finding and configuring IPC@CHIP®s via the network. It runs by default on each IPC@CHIP® and waits for configuration requests transmitted by a client. Such a client is for example the @CHIPTOOL, which you can download from our website. It implements the configuration server protocol to find IPC@CHIP®s on the network and to configure them.

This document describes the protocol used by the configuration server. It enables you to include IPC@CHIP® configuration functionality into your own applications. For your convenience we also provide a DLL (Dynamic Link Library), the ChipControl DLL, which implements the protocol. You can download it from our website.

2 General information

The UDP configuration server listens on port 8001 UDP. All configuration requests must be sent to this port. Depending on the requested action and the IPC@CHIP®s configuration status the requests must be sent as a broadcast – especially if the IPC@CHIP® doesn't have a valid IP address, yet. In this case addressing is done via the chip's ID (see below).

2.1 Packet format

The configuration protocol is a text-based protocol, like for example the FTP protocol.

Note: The extended Backus-Naur form (EBNF) is used to describe the packet format. However, to keep the description concise, simplified and pseudo definitions are used throughout the text. The complete formal definition can be found in appendix A.

The general request packet format is as follows:

```
packet = id, " ", command, " ", data, zero character;
```

Each request packet starts with the `id` field. The ID can be either the serial number or the MAC address (also referred to as the MAC ID) of the IPC@CHIP®s (first) Ethernet interface in hex string representation. When sending a request packet to an IPC@CHIP®, you should always insert the MAC address here, since the serial number might be ambiguous. There might be two different IPC@CHIP®s (e.g. an SC12 and an SC123) with the same serial number. But they'll have different MAC addresses. However IPC@CHIP®s with an older @CHIP-RTOS will not understand packets starting with the MAC address. These are all SC12 RTOS versions below 1.10 and SC13 RTOS versions below 0.9. If a packet is not sent to a special chip but to all IPC@CHIP®s on the network, set the ID to 0. Reply packets will always start with the serial number.

```
id = serial number | mac id;
```

The `command` field contains the decimal number of the command that is to be executed (see below).

```
command = decimal number;
```

The contents of the `data` section depend on the command.

All packets are terminated with a zero character, so that they can be handled as strings programmatically.

Note: The hello reply packet and the advanced hello reply packet are exceptions to the above rules. They lack the `id` and `command` field (see section 3.1.2).

3 Commands

The following commands are supported by the UDP configuration server. Note that not all commands are available on all @CHIP-RTOS versions, since the protocol has been advanced over time.

| No. | Brief description | Availability |
|-----|---|---|
| 1 | <i>Hello</i> request | All RTOS versions |
| | <i>Hello</i> request with <i>advanced</i> option | SC12 RTOS ≥1.10, SC13 RTOS, SC1x3 RTOS SC23 RTOS |
| 2 | <i>Hello</i> reply | All RTOS versions ¹ |
| | Advanced <i>Hello</i> reply | SC12 RTOS ≥1.10, SC13 RTOS, SC1x3 RTOS SC23 RTOS |
| | Advanced <i>Hello</i> reply with multi home index | SC1x RTOS ≥1.20, SC1x3 RTOS ≥1.05 SC23 RTOS |
| 5 | IP configuration request and reply (basic) | All RTOS versions ² |
| 4 | IP configuration request and reply (advanced 1, with device index) | SC12 RTOS ≥1.10, SC13 RTOS, SC1x3 RTOS SC23 RTOS |
| 3 | IP configuration request and reply (advanced 2, with multi home index) | SC1x RTOS ≥1.20, SC1x3 RTOS ≥1.05 SC23 RTOS |
| 8 | IP configuration request and reply (advanced 3, with password protection) | SC1x3 RTOS ≥1.15 SC23 RTOS |
| 6 | User callback request | SC12 RTOS ≥1.04, SC13 RTOS, SC1x3 RTOS SC23 RTOS |
| 7 | User callback acknowledge | SC12 RTOS ≥1.04, SC13 RTOS, SC1x3 RTOS SC23 RTOS |

Table 1: Overview of the UDP configuration server commands

3.1 Hello

3.1.1 Request

A client can send a *Hello* request in order to find IPC@CHIP®s on the network. If a special chip is sought, the request can be sent to this chip's IP address and the chip's ID can be inserted

¹ SC12 RTOS versions starting from 1.10 and all SC13, SC1x3 and SC23 RTOS versions don't support the simple *Hello* reply anymore. These versions will always reply with an advanced reply. However this will not affect older clients due to the zero character delimiters.

² SC12 RTOS versions older than 1.10 don't send a reply to the basic configuration request. All RTOS versions for other IPC@CHIP® variants (e.g. SC1x3) do.

into the request packet. If a general scan for any IPC@CHIP®s on the local network is to be performed, the packet is to be sent as a broadcast (IP address 255.255.255.255) and '0' is to be inserted into the ID field of the request packet.

The structure of the *Hello* request packet is as follows:

```
packet hello request3 = id, " 1 ", [ "A" ], zero character;
```

Besides the ID and the command number, there is an optional field containing only the character 'A'. If this field is present, the configuration server of IPC@CHIP®s receiving this request is instructed to send an advanced *Hello* reply packet (see below).

Here are some examples of *Hello* request packets⁴:

```
0_1  
0_1_A  
3838_1_A
```

- The first example scans the local network for IPC@CHIP®s.
- The second example also scans the local network for IPC@CHIP®s, but expects advanced reply packets.
- The third example requests an advanced reply packet from the IPC@CHIP® with the serial number '3838'. This example could be sent to the IP address of the corresponding chip, if it is known.

3.1.2 Reply

The simple *Hello* reply packet has the following format:

```
packet hello reply = serial number, " ", dhcp status, " ", ip address, " ",  
netmask, " ", gateway, " ", device name, zero character;
```

The fields have the following meaning:

serial number: Serial number of the IPC@CHIP®
dhcp status: "1", if the IPC@CHIP® has been configured via DHCP; "0" if not
ip address: IPC@CHIP®s IP address in dotted notation
netmask: IPC@CHIP®s network mask in dotted notation
gateway: IPC@CHIP®s default gateway's IP address in dotted notation
device name: IPC@CHIP®s device name (Can be assigned via an entry in the CHIP.INI configuration file)

The client can request an advanced *Hello* reply packet. This extension of the *Hello* mechanism is supported by newer @CHIP-RTOS versions (see Table 1). IPC@CHIP®s with an older RTOS version will reply with the simple *Hello* reply packet anyhow. The advanced *Hello* reply has been advanced another time. Even newer @CHIP-RTOS versions (see Table 1) include another field called `multi home index` in the reply (see below). The advanced reply has the following format:

³ The non-terminal symbol *packet hello request* cannot be found in the complete EBNF definition in appendix A. It is a pseudo non-terminal symbol. In fact *packet hello request* is a *packet* where the non-terminal *data* is a *data hello request*. This kind of simplification will be used in several places throughout the text.

⁴ In example packets the trailing zero characters are not displayed and space characters that belong to the protocol are represented by the symbol '␣'. All other white space characters and line breaks within the examples are there just to increase readability. They don't belong to the protocol.

```
packet hello reply advanced = serial number, " ", dhcp status, " ", ip address,  
" ", netmask, " ", gateway, " ", device name,  
zero character, signature, " ", rtos version, " ",  
rtos variant, " ", boot loader version, " ",  
hardware revision, " ", mac id, " ", device index,  
" ", device type, " ", physical address, " ",  
target type, [ " ", multi home index ],  
zero character, valid marker,  
[ beck serial number, " ", beck hardware revision,  
" ", beck device name ], zero character;
```

The additional fields have the following meaning:

| | |
|-------------------------|---|
| zero character: | A zero character is used to separate the simple <i>Hello</i> reply part from the advanced. Clients that don't know of the advanced part interpret the zero character as the end of the reply and thus ignore the advanced part, which they cannot interpret. |
| signature: | RTOS signature (Used by @CHIPTOOL only) |
| rtos version: | RTOS version (e.g. V0.90B or V1.01) |
| rtos variant: | Variant of the RTOS version (e.g. TINY, LARGE or FULL) |
| boot loader version: | Boot loader version |
| hardware revision: | Hardware revision |
| mac id: | MAC address of the (first) Ethernet controller of the IPC@CHIP® (used to identify the chip) |
| device index: | Index of the network device via which this reply has been sent. This can be for example "2" for the first Ethernet controller of the IPC@CHIP®. The device index is also displayed along with the output of the IPCCFG shell command. Note that if the IPC@CHIP® is reachable via several interfaces, one reply will be received for each interface. The device index will be different in these packets. |
| device type: | Decimal number identifying the type of the network device via which the reply has been sent. ("0": Unknown, "1": Ethernet, "2": PPP) |
| physical address: | Physical device address of the network interface. For Ethernet interfaces this is the MAC address, for PPP interfaces it is "000000000000". The length of this field is fixed at 12. |
| target type: | Target type of the IPC@CHIP® (e.g. SC12 or SC123) |
| multi home index: | On newer RTOS versions several IP configurations may be assigned to a single interface. Each IP configuration has an index (starting from 0 for the first configuration), the multi home index. Note that one <i>Hello</i> reply will be received for each IP configuration assigned to an interface. The multi home index will be different in these packets. |
| zero character: | Another zero character separating the next part of the reply |
| valid marker: | This field consists of a single character indicating whether the following fields are valid. A space character indicates they're valid, a zero character indicates they're invalid. |
| beck serial number: | Serial number of the IPC@CHIP® based product (only used for products offered by Beck IPC) |
| beck hardware revision: | Product's hardware revision (only used for products offered by Beck IPC) |
| beck device name: | Product's device name (only used for products offered by Beck IPC) |

Here are examples of a simple and an advanced *Hello* reply packet:

```
00005A_1_172.30.10.62_255.255.240.0_172.30.0.125_DK60
```

```
00005A_1_172.30.10.62_255.255.240.0_172.30.0.125_DK60\08_V1.15B_FULLL_V3.00_V0.01_  
003056A0005A_2_1_003056A0005A_SC143\0_000001_V0.01_DK605
```

3.2 IP configuration

3.2.1 Request

The IP configuration request can be used to configure the network interfaces of the IPC@CHIP®. Along with new features included into the @CHIP-RTOS, the IP configuration functionality of the configuration server has been extended to provide means to configure and/or use these features. Because of this there are by now four different IP configuration request formats.

Note: For the IP configuration requests the ID field cannot be 0, but must contain a valid serial number or MAC ID.

3.2.1.1 The basic request

The basic IP configuration request has the following format:

```
packet ip config request 5 = id, " 5 ", dhcp status,  
    [ " ", ip address, " ", netmask,  
    [ " ", gateway ] ],  
    zero character;
```

The fields have the following meaning:

dhcp status: "1", if the IPC@CHIP®'s network interface is to be configured via DHCP;
"0" if not
ip address: IP address that is to be configured (omitted, if DHCP is to be used)
netmask: Network mask that is to be configured (omitted, if DHCP is to be used)
gateway: Default gateway's IP address that is to be configured (can be omitted)

Here are two examples of valid IP configuration requests:

```
003056F001D9_5_1  
003056F001D9_5_0_192.168.0.11_255.255.255.0_192.168.0.1
```

The first request configures the IPC@CHIP® with the MAC ID 00:30:56:F0:01:D9 with DHCP enabled. The second configures the same chip with a fixed address (IP address: 192.168.0.11, netmask: 255.255.255.0, gateway: 192.168.0.1).

3.2.1.2 First advancement: Device index

The first advanced version of the IP configuration request features an additional field which defines the index of the IPC@CHIP®'s network interface that is to be configured. The format of the IP configuration request in this case is as follows:

```
packet ip config request 4 = id, " 4 ", device index, " ", dhcp status,  
    [ " ", ip address, " ", netmask,  
    [ " ", gateway ] ],
```

⁵ '\0' represents a zero character.

```
zero character;
```

All fields that are also present in the basic IP configuration request, have the same meaning here. As already mentioned the field `device index` specifies the index of the network device to be configured. Valid values for this field can for example be determined with the `IPCFCG` command on the IPC@CHIP®'s command shell. This command also displays the device index. It's labelling is `Idx` here.

Two example requests:

```
003056F001D9_4_2_1  
003056F001D9_4_3_0_192.168.0.11_255.255.255.0_192.168.0.1
```

The first request configures the interface with the index 2 with DHCP. The second configures the interface with the index 3 with a fixed address.

3.2.1.3 Second advancement: Multi home index

The second advancement of the IP configuration request is the support for multi homing. Multi homing in this case means that one network interface can have several IP addresses. By passing a multi home index along with the IP configuration request one can determine which IP configuration is to be changed. This version of the request has the following format:

```
packet ip config request 3 = id, " 3 ", device index, " ", dhcp status,  
    [ " ", ip address, " ", netmask,  
      [ " ", gateway ] ],  
    " ", multi home index, zero character;
```

We already got to know most of the fields of this request in the previous sections. The only additional field is the `multi home index`. This is simply an index starting from 0. Let's have an example to clarify this:

Assuming the IPC@CHIP® with the MAC address 00:30:56:F0:01:D9 has not yet been assigned a valid IP configuration, we send the following two requests:

```
003056F001D9_3_2_0_192.168.0.11_255.255.255.0_192.168.0.1_0  
003056F001D9_3_2_0_192.168.1.23_255.255.255.0_192.168.1.1_1
```

This will assign the IP addresses 192.168.0.11 (multi home index 0) and 192.168.1.23 (multi home index 1) to the network interface with the index 2. The following request will change the first IP address from 192.168.0.11 to 192.168.0.13, leaving the second one (multi home index 1) untouched.

```
003056F001D9_3_2_0_192.168.0.13_255.255.255.0_192.168.0.1_0
```

3.2.1.4 Third advancement: Password protection

This advancement supports configuring IPC@CHIP®s which have password protection enabled for the IP configuration. The packet format looks as follows:

```
packet ip config request 8 = id, " 8 ", password hash, " ", device index, " ",  
    dhcp status, [ " ", ip address, " ", netmask,  
                  [ " ", gateway ] ],  
    " ", multi home index, zero character;
```

The password consists of up to 16 printable non-whitespace characters and can be configured via the IPC@CHIP®'s `CHIP.INI` configuration file. To protect the password from being tapped by someone else on the network, it is not sent as plain text but encapsulated in an MD5⁶ hash. This hash is build from the password itself, some random data (called the *random token*) and the IP configuration packet (with the `password hash` field set to all zeros). The *random token* is sent along with an IP configuration reply packet when password protection is enabled. To configure an IPC@CHIP®, which has password protection enabled, you have to perform the following steps:

1. Send an IP configuration request (either one with command 8 and a random `password hash` field or one of the older versions of the request).
2. The IPC@CHIP® will send a reply packet with the `command result` field (see section 3.2.2) indicating that a password is needed.
3. Get the random token from the reply packet (Convert it from string representation to a byte array).
4. Prepare the next IP configuration request packet, set the `password hash` field to all zeros.
5. Build an MD5 hash over (in this order) the password, the random token and the prepared request packet.
6. Insert the generated MD5 hash (in string representation) into the prepared packet.
7. Send the new request packet and check the reply.

Pseudo code:

```
// Command 3, device index 2, DHCP status 1, multi home index 0
sprintf(txBuffer, "%02X%02X%02X%02X%02X%02X 3 2 1 0",
        macId[0], macId[1], macId[2], macId[3], macId[4], macId[5]);

sendPacket(txBuffer, strlen(txBuffer));

rxLength = receivePacket(rxBuffer);

if(getReplyResult(rxBuffer, rxLength) == -1)
{
    parseRandomToken(&randomToken[0], rxBuffer, rxLength);

    sprintf(txBuffer, "%02X%02X%02X%02X%02X%02X 8 "
                "00000000000000000000000000000000 2 1 0",
            macId[0], macId[1], macId[2], macId[3], macId[4], macId[5]);

    md5Prepare(&md5Context);
    md5Update(&md5Context, password, strlen(password));
    md5Update(&md5Context, randomToken, 16);
    md5Update(&md5Context, txBuffer, strlen(txBuffer));
    md5Finalise(&hash[0], &md5Context);

    sprintf(txBuffer, "%02X%02X%02X%02X%02X%02X 8 "
                "%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X "
                "2 1 0",
            macId[0], macId[1], macId[2], macId[3], macId[4], macId[5],
            hash[0], hash[1], hash[2], hash[3], hash[4], hash[5], hash[6],
            hash[7], hash[8], hash[9], hash[10], hash[11], hash[12], hash[13],
            hash[14], hash[15]);

    sendPacket(txBuffer, strlen(txBuffer));

    rxLength = receivePacket(rxBuffer);

    // Check result...
}
```

⁶ For further information on MD5 see <http://en.wikipedia.org/wiki/md5>. The corresponding specification can be found on <http://tools.ietf.org/html/rfc1321>.

```
else // ...
```

Here is an example of an IP configuration request with password hash:

```
003056A03838_8_305C002022AC20223B5C2D232C23235C_2_1_0
```

3.2.2 Reply

The IP configuration reply packet has the following format:

```
packet ip config reply = serial number, " ", command, " ", command result,  
                        [ " ", random token ], zero character;
```

The fields have the following meaning:

serial number: IPC@CHIP®'s serial number
command: Command number copied from the request packet
command result: Result of the previous request:
 "-1" Password needed/wrong password
 "0" IP configuration failed
 "1" IP configuration succeeded
random token: Only supplied from RTOS versions supporting password protection for the IP configuration. 16 bytes of random data (in hex string representation e.g. "A5677BF4200CA891FF80") used by the client to "encrypt" the password (see section 3.2.1.4).

Some examples:

```
3838_4_1_1_8C5711CB1611C3166FC3E36F56E3B156  
3838_8_1
```

The first reply tells the client, that a password is needed to configure the IPC@CHIP® (command was "4", result is "-1", random token is supplied).

The second reply states that the configuration was successful (command was "8", result is "1").

3.3 User callback

A callback function can be installed on the IPC@CHIP® that can be triggered via the UDP configuration server. A string can be send along with the respective request packet which will then be passed to the callback function. The Function in turn may also return a string that will then be sent back to the client with a reply packet. This feature enables you to add custom commands to the configuration server.

3.3.1 Request

The user callback request packet has the following format:

```
packet callback request = id, " 6", [ " ", data callback request ],  
                        zero character;  
  
data callback request = printable character,  
                        299 * ( printable character | (* nothing *) );
```

Besides the id and command number fields, the packet may contain a text with up to 300 printable characters. But the packet may also only consist of the ID and the command number. The callback function will then be called without an argument.

Here's an example:

```
003056F001D9_6_Message_for_UDP_configuration_server
```

3.3.2 Reply

The user callback reply packet will only be sent back to the client, if the callback function returns a text which is to be sent. The packet has the following format:

```
packet callback reply = serial number, " 7 ", data callback reply,  
                        zero character;  
data callback reply = printable character,  
                    299 * ( printable character | (* nothing *) );
```

The text may also consist of up to 300 printable characters.

An example:

```
1D9_7_Reply_for_UDP_configuration_client
```

3.3.3 Callback function

The user callback function must have the following format:

```
void huge callback(struct UdpCfgSrv_UserCBInfo far *info);
```

The structure `UdpCfgSrv_UserCBInfo` is defined as follows:

```
struct UdpCfgSrv_UserCBInfo  
{  
    int length;  
    struct sockaddr_in far *fromAddrPtr;  
    int udpCfgSD;  
    char far *dataPtr;  
    unsigned dataLength;  
}
```

The most important fields are `dataPtr` and `dataLength`, which hold the message from the client and which are also used to store the message that is to be sent back to the client. A callback function could for example look like this:

```
char message[301];  
  
void huge callback(struct UdpCfgSrv_UserCBInfo far *info)  
{  
    // Check for valid structure  
    if(info->length)  
    {  
        // Extract message  
        memcpy(message, info->dataPtr, info->dataLength);  
        message[min(info->dataLength, 300)] = '\0';  
  
        // Check message  
        if(strcmpi(message, "Message for UDP configuration server") == 0)  
        {  
            // Correct message. Send reply  
            sprintf(message, "Reply for UDP configuration client");  
        }  
    }  
}
```

```
    info->dataPtr = message;  
    info->dataLength = strlen(message);  
  }  
}  
  
// Not the correct message. Don't send a reply.  
info->dataPtr = NULL;  
info->dataLength = 0;  
}
```

To install the callback function use the API function `BIOS_Install_UDP_Cfg_Callback()`. Its only argument is a pointer to the callback function. So the installation of the above callback function could look like this:

```
void main()  
{  
  ...  
  BIOS_Install_UDP_Cfg_Callback(callback);  
  ...  
}
```

For a complete reference of the API function `BIOS_Install_UDP_Cfg_Callback()` and the structure `UdpCfgSrv_UserCBInfo` refer to the [@CHIP-RTOS C Library documentation](#).

A Complete EBNF definition of the protocol

```
(* ----- *)
(* General packet definition *)
packet = ( packet request | packet reply ), zero character;
packet request = id, " ", command, [ " ", data ];
packet reply = ( serial number, " ", command, " ", data ) | packet reply hello;

id = serial number | mac id;
serial number = hex character, 5 * ( hex character | (* nothing *) );
mac id = mac address;

command = decimal number;

data = data hello request |
      | data ip config request 5 | data ip config request 4
      | data ip config request 3 | data ip config request 8
      | data ip config reply
      | data callback request | data callback reply;

(* ----- *)
(* Definitions concerning the hello request and reply *)
data hello request = (* nothing *) | "A";

packet hello reply = serial number, " ", dhcp status, " ", ip address, " ",
                    netmask, " ", gateway, " ", device name;

packet hello reply advanced = serial number, " ", dhcp status, " ", ip address,
                             " ", netmask, " ", gateway, " ", device name,
                             zero character, signature, " ", rtos version, " ",
                             rtos variant, " ", boot loader version, " ",
                             hardware revision, " ", mac id, " ", device index,
                             " ", device type, " ", physical address, " ",
                             target type, [ " ", multi home index ],
                             zero character, valid marker,
                             [ beck serial number, " ", beck hardware revision,
                               " ", beck device name ];

dhcp status = "0" | "1";
ip address = ip address format;
netmask = ip address format;
gateway = ip address format;
device name = printable character, 19 * ( printable character | (* nothing *) );
signature = decimal number; (* 1 .. 256 *)
rtos version = version string, [ "B" ];
rtos variant = "TINY" | "SMALL" | "MEDIUM" | "MEDIUM_PPP" | "LARGE"
              | "LARGE_PPP" | "FULL";
boot loader version = version string;
hardware revision = version string;
device index = decimal number;
device type = decimal number;
physical address = mac address;
target type = printable character, 5 * ( printable character | (* nothing *) );
multi home index = decimal number;
valid marker = " " | zero character;
beck serial number = hex character, 5 * ( hex character | (* nothing *) );
beck hardware revision = version string;
beck device name = printable character,
                  5 * ( printable character | (* nothing *) );

(* ----- *)
(* Definitions concerning the ip configuration request and reply *)
```

```

data ip config request 5 = dhcp status, [ " ", ip address, " ", netmask,
                                     [ " ", gateway ] ];
data ip config request 4 = device index, " ", dhcp status,
                          [ " ", ip address, " ", netmask, [ " ", gateway ] ];
data ip config request 3 = device index, " ", dhcp status,
                          [ " ", ip address, " ", netmask, [ " ", gateway ] ],
                          " ", multi home index;
data ip config request 8 = password hash, " ", device index, " ", dhcp status,
                          [ " ", ip address, " ", netmask, [ " ", gateway ] ],
                          " ", multi home index;
data ip config reply = command result, [ " ", random token ];

command result = decimal number
password hash = hex character, 31 * ( hex character | (* nothing *) );
random token = hex character, 31 * ( hex character | (* nothing *) );

(* ----- *)
(* Definitions concerning the user callback request and reply *)
data callback request = printable character,
                      299 * ( printable character | (* nothing *) );
data callback reply = printable character,
                     299 * ( printable character | (* nothing *) );

(* ----- *)
(* Some general definitions *)
ip address format = ip address octet, ".", ip address octet, ".",
                  ip address octet, ".", ip address octet;
ip address octet = decimal number; (* 0..255 *)

mac address = 12 * hex character;

version string = "V", hex number, ".", 2 * hex character;

zero character = ? ASCII character 0 ?;
decimal character = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
decimal number = [ "-" ] , decimal character, { decimal character };
hex character = decimal character | "A" | "B" | "C" | "D" | "E" | "F";
hex number = hex character, { hex character };
printable character = ? ASCII characters starting from 32 ?;

```

B History

| Version | Date | Author | Comments |
|---------|------------|------------|-------------------|
| 2.0 | 29.08.2007 | Jan Schatz | Complete revision |

Copyright © 2007 BECK IPC GmbH All rights reserved

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of BECK IPC GmbH. The information in this document is subject to change without notice. Devices sold by BECK IPC GmbH are covered by warranty and patent indemnification provisions appearing in BECK IPC GmbH Terms and Conditions of Sale only.

BECK IPC GmbH MAKES NO WARRANTY, EXPRESS, STATUTORY, IMPLIED OR BY DESCRIPTION, REGARDING THE INFORMATION SET FORTH HEREIN OR REGARDING THE FREEDOM OF THE DESCRIBED DEVICES FROM INTELLECTUAL PROPERTY INFRINGEMENT. BECK IPC GmbH MAKES NO WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PURPOSE.

BECK IPC GmbH shall not be responsible for any errors that may appear in this document. BECK IPC GmbH makes no commitment to update or keep current the information contained in this document.

Life critical applications - BECK products are not authorized for use as critical components in life support devices or systems unless a specific written agreement pertaining to such intended use is executed between the customer and BECK prior to use. Life support devices or systems are those which are intended for surgical implantation into the body, or which sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in significant injury to the user. BECK IPC GmbH customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify BECK IPC GmbH for any damages resulting from such application.

Right to make changes - Beck IPC GmbH reserves the right to make changes without notice in the products, including software, described or contained herein in order to improve design and/or performance. Beck IPC GmbH assumes no responsibility or liability for the use of any of these products.