

## IPC@Chip Documentation index - BIOS V1.02 Beta

[@Chip-RTOS overview](#)

[Scaled @Chip-RTOS versions](#)

[IPC@Chip startup initialization](#)

[BIOS: Interrupts for several PC services](#)

[CGI: Software interface to CGI](#)

[COMMAND: Description of the command processor.](#)

[CONFIG: System configuration based on CHIP.INI file.](#)

[DOS: Interrupt 0x21 functions](#)

[External Disk Drive B: Interface Definition](#)

[FOSSIL: Interface to the serial ports.](#)

[Hardware API: Including PFE and HAL](#)

[I2C: Interface definition for the I2C Bus Interface](#)

[Ethernet: Packet Driver Interface](#)

[PPP Interface: How to configure the SC12 PPP server.](#)

[RTOS API: Interface definition for RTOS interface.](#)

[TCP/IP API: Interface definition for the TCP/IP sockets.](#)

[TFTP server: Short description of the SC12 TFTP server.](#)

[Security notes](#)

[Programming notes](#)

---

End of document

---

## @CHIP-RTOS Software overview

IPC@Chip Documentation [Index](#)

### RTOS

- 35 Tasks
- 15 Timers
- 60 Semaphores
- 10 Message exchanges
- 2 Event groups

### RTOS Filesystem for

- Internal ramdisk
- Internal flashdiskdrive
- External drive

### TCP/IP Stack

- TCP
- UDP
- ARP
- ICMP
- Socket interface
- 64 Sockets
- 3 device interfaces
  - Ethernet
  - PPP server
  - PPP client

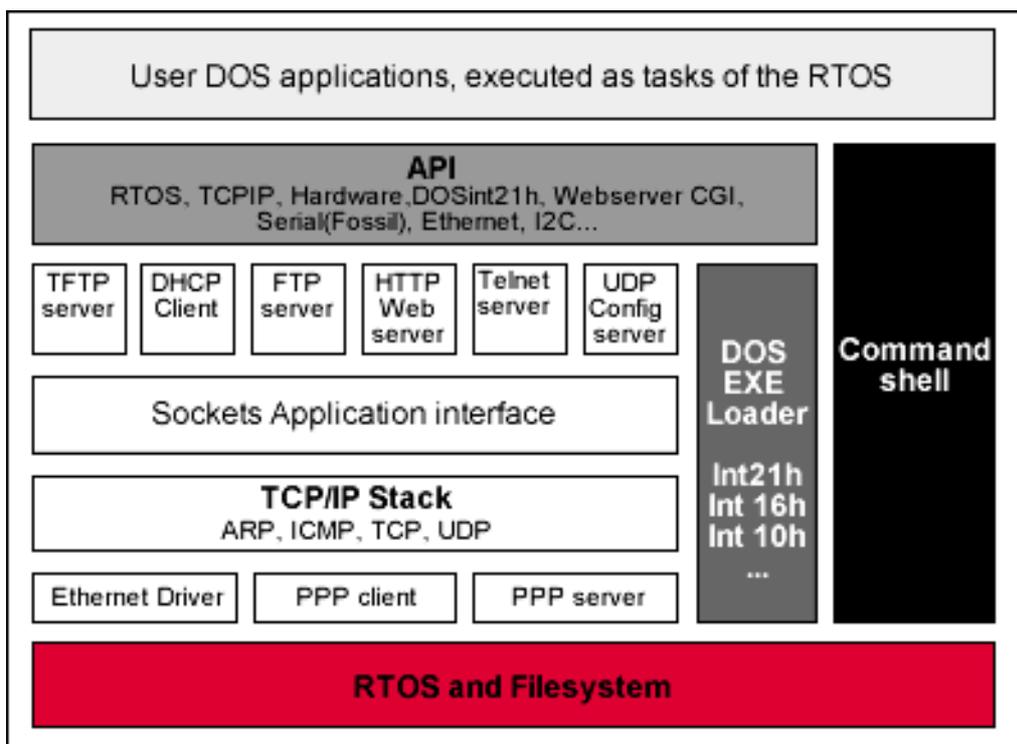
### TCP/IP applications

- HTTP Webserver
- FTP server
- Telnet server
- TFTP server
- DHCP client
- UDP config server for @CHIP-RTOS upgrade

### DOS-EXE Loader

- Up to 12 DOS application programmes can run as tasks of the RTOS

### DOS-like command shell



@Chip-RTOS architecture

### Application Programmer Interface

- RTOS
- TCP/IP socketinterface
- DOS interrupt 21h and others
- Webserver CGI
- Hardware
- I2C
- Serial devices (Fossil interface)
- Ethernet Packet driver
- Special @CHIP-RTOS services

### DOS application examples

- RTOS API examples
- TCP/IP API examples
  - FTP client
  - HTTP client
  - Other TCP/IP examples

- Supports a subset of DOS commands and @CHIP-RTOS specific commands via Telnet or serial devices

#### **Packet driver interface**

- Accessing the ethernet device without TCP/IP

#### **Serial filetransfer via xmodem**

#### **Scalable @CHIP-RTOS**

- Support of 6 different versions, including various @CHIP RTOS features

#### **@CHIP-RTOS Upgrade via Ethernet/UDP or serial Bootstraploader**

- Webserver CGI examples
- External IDE disk driver
- Hardware API examples
- I2C examples
- Fossil examples
- API examples written in Turbo Pascal
- API C-Libraries
- more

---

End of document

---

## Scalable BIOS Versions of the IPC@Chip

IPC@Chip Documentation [Index](#)

1. SC12Vxxxx\_TINY.hex
2. SC12Vxxxx\_SMALL.hex
3. SC12Vxxxx\_MEDIUM.hex
4. SC12Vxxxx\_LARGE.hex
5. SC12Vxxxx\_MEDIUM\_PPP.hex
6. SC12Vxxxx\_LARGE\_PPP.hex

	Tiny	Small	Medium	Large
RTOS-Kernel	x	x	x	x
Serial	x	x	x	x
RTOS-Filesystem	x	x	x	x
Ext Disk			x	x
XMODEM-Protocol	x	x	x	x
TCPIP-Ethernetdriver		x	x	x
Ethernet Packet-Interface	x	x	x	x
TCP-IP		x	x	x
I2C	x	x	x	x
Hardware API	x	x	x	x
CFG server		x	x	x
Webserver				x
FTP Server			x	x
Telnet Server			x	x
PPP only with Medium & Large				
PPP client & server			x	x

TFTP server is not a part of the 6 official BIOS versions, but the customers could order a BIOS version which includes this feature.

Each BIOS version with TCPIP protocol stack includes a DHCP client. It is possible to order a BIOS version without DHCP.

The IPC@Chip offers 512kB RAM and 512kB Flash disk.

Here are the sizes of available RAM and flash disk for the different versions of BIOS 1.02 Beta

---

	Available RAM(kBytes)	Available Flash memory(kBytes)
Tiny	451	388
Small	354	281
Medium	348	250
Medium_PPP	343	198
Large	322	227
Large_PPP	317	175

---

End of document

---

## IPC@Chip Initialization - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### Programmable I/O Pins

At turn-on the IPC@Chip I/O pins are configured as follows:

Pin1: RXD0/PIO7 = RXD0  
Pin2: TXD0/PIO8 = TXD0  
Pin3: CTS0/PIO9 = Input pullup  
Pin4: RTS0/PIO10= Input pullup  
Pin5: TXD1/PIO11= TXD1  
Pin6: RXD1/PIO12= RXD1  
Pin7: TMROUT0/INT0/PIO13 = Input pulldown  
Pin17: RESET/PFAIL/LILED = Input  
Pin24: ALE/PCS0 = Output, value 1  
Pin25: CTS1/PCS2/PIO6/INT2 = Input pullup  
Pin26: RTS1/PCS3/PIO5/INT4 = Input pullup  
Pin27: PCS1/PIO4/TMRIN0/A0 = Input pullup  
Pin28: PCS5/PIO3/TMROUT1/TMRIN1/A1= Input pullup  
Pin29: PCS6/PIO2/A2= Input pullup  
Pin30: I2CDAT/INT5/PIO1 = Input  
Pin31: I2CCLK/INT6/PIO0 = Input

[Back to main index page](#)

---

End of document

---

## BIOS Interface Documentation - BIOS V1.02 Beta

IPC@Chip Documentation [Index](#)

### BIOS Interrupts

Here are the interface definition for the BIOS Interrupts

The system BIOS in a regular PC offers several services. Only a subset is required for embedded systems. This subset is described here.

Other functions not found in a regular PC systems have been added for your convenience.

Functions that are not supported, are handled in a default handler, issuing a message to the console.

New in version 1.02B: [Enable/Disable Filesharing](#)

New in version 1.02B: [Install System Server Connetion Handler functions](#)

New in version 1.02B: [Install user stdio handler functions](#)

New in version 1.02B: [Get stdin/stdout settings](#)

New in version 1.02B: [Set stdin/stdout channel](#)

New in version 1.02B: [Set a memory gap between the loaded DOS programs](#)

New in version 1.02B: [Detect ethernet link state](#)

New in version 1.02B: [Fast FindFirst/Next](#)

New in version 1.02B: [Get the IPC@Chip device names](#)

New in version 1.02B: [Suspend/Resume System Servers](#)

New in version 1.01B: [Modified BIOS feature function](#)

New in version 1.01B: [Set the Stdio Focuskey](#)

New in version 1.00: [Insert an entry at chip.ini](#)

New in version 1.00: [Find an entry at chip.ini](#)

- [Interrupt 0x1A function 0x00: Get clock count since midnight](#)
- [Interrupt 0x11 function x: Get equipment list](#)
- [Interrupt 0x10 function 0x00: Get char from standard input](#)
- [Interrupt 0x10 function 0x01: Check if a character is available from std in](#)
- [Interrupt 0x10 function 0x08: Read character at cursor](#)
- [Interrupt 0x10 function 0x0E: Teletype output](#)
- [Interrupt 0x10 function 0x0f: Get video state](#)
- [Interrupt 0x10 function 0x12: Video subsystem configuration](#)
- [Interrupt 0x16 function 0x00: Get char from standard input](#)
- [Interrupt 0x16 function 0x01: Check if a character is available from std in](#)
- [Interrupt 0xA0 function 0x00: Get serial number](#)
- [Interrupt 0xA0 function 0x01: Get IP address of the Ethernet interface](#)
- [Interrupt 0xA0 function 0x02: Set IP address of the Ethernet interface](#)
- [Interrupt 0xA0 function 0x03: Get IP subnet mask of the Ethernet interface](#)
- [Interrupt 0xA0 function 0x04: Set IP subnet mask](#)
- [Interrupt 0xA0 function 0x05: Get IP gateway](#)
- [Interrupt 0xA0 function 0x06: Set IP default gateway](#)
- [Interrupt 0xA0 function 0x07: Execute a command shell command](#)
- [Interrupt 0xA0 function 0x08: Set timer 0x1C's interval](#)
- [Interrupt 0xA0 function 0x09: Set timer interrupt 0xAF's interval](#)

- [Interrupt 0xA0 function 0x11: Set STDIO focus](#)
  - [Interrupt 0xA0 function 0x12: Get bootstrap version number](#)
  - [Interrupt 0xA0 function 0x13: Get BIOS version number](#)
  - [Interrupt 0xA0 function 0x14: Set batch file execution mode.](#)
  - [Interrupt 0xA0 function 0x15: Allow immediate further batch file execution in BATCHMODE 1.](#)
  - [Interrupt 0xA0 function 0x16: Get information about the BIOS features](#)
  - [Interrupt 0xA0 function 0x17: Get MAC address of the Ethernet interface](#)
  - [Interrupt 0xA0 function 0x18: Power save](#)
  - [Interrupt 0xA0 function 0x19: Change level for configuration server.](#)
  - [Interrupt 0xA0 function 0x20: Install a user fatal error handler](#)
  - [Interrupt 0xA0 function 0x21: Rebooting the SC12](#)
  - [Interrupt 0xA0 function 0x22: Get version string](#)
  - [Interrupt 0xA0 function 0x23: Insert an entry at chip.ini](#)
  - [Interrupt 0xA0 function 0x24: Find an entry at chip.ini](#)
  - [Interrupt 0xA0 function 0x25: Set the Stdio Focuskey](#)
  - [Interrupt 0xA0 function 0x26: Get the IPC@Chip device names](#)
  - [Interrupt 0xA0 function 0x27: Suspend/Resume System Servers](#)
  - [Interrupt 0xA0 function 0x28: Fast Findfirst](#)
  - [Interrupt 0xA0 function 0x29: Fast Findnext](#)
  - [Interrupt 0xA0 function 0x30: Fast Finddone](#)
  - [Interrupt 0xA0 function 0x31: Detect ethernet link state](#)
  - [Interrupt 0xA0 function 0x32: Set a memory gap between the loaded DOS programs](#)
  - [Interrupt 0xA0 function 0x33: Set stdin/stdout channel](#)
  - [Interrupt 0xA0 function 0x34: Get stdin/stdout settings](#)
  - [Interrupt 0xA0 function 0x35: Install specific user stdio handlers](#)
  - [Interrupt 0xA0 function 0x36: Install a System Server Connection Handler function](#)
  - [Interrupt 0xA0 function 0x37: Enable/Disable Filesharing](#)
  - [Interrupt 0xA0 function 0x38: Get Filename by Handle](#)
- 

## Interrupt 0x1A service 0x00: Get clock count since midnight

Returns the number of clock ticks since midnight.  
The frequency of the clock is 18.2 Hz (e.g. 54.945 ms per tick).

### Parameters

*AH*  
Must be 0.

### Return Value

Returns the 32 bit tick count in CX (high word) and DX (low word)  
If an overflow occurred since the last call, AX is set to 1.

### Comments

Please note that the overflow indication returned in register AX can not be relied upon if several tasks are using this service.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x11 service x: Get equipment list

Get the bios equipment list

## Return Value

Returns the equipment list in AX, currently 0x013C. This bit field indicates:

bit 8: 1 : No DMA  
bit 6-7: 00: One floppy  
bit 4-5: 11: 80x25 mono  
bit 2-3: 11: system ram  
bit 1: 0 : no 8087  
bit 0: 0 : no disk drives

## Comments

This function is needed to make sure an application finds no 8087 coprocessor so it can load a floating-point emulator.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x10 service 0x00: Get char from standard input

Get a character from std in, wait if none available

## Parameters

*AH*

Must be 0.

## Return Value

Returns input character in AL

Return at DX the source stdin channel: 1: EXT , 2: COM , 4: Telnet

## Comments

Please note that AH does not contain the scan code, but is always 0.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x10 service 0x01: Check if a character is available from std in

Check if a character is available from standard input

## Parameters

*AH*

Must be 1.

## Return Value

AX=1 if a character is available, AX=0 and zero-flag is cleared if no character is available.

## Comments

Please note that AH does not contain the scan code, but is instead always 0.

[Top of list](#)

[Index page](#)

---

### **Interrupt 0x10 service 0x08: Read character at cursor**

Read character at cursor position, always returns 0

#### **Parameters**

*AH*

Must be 0x08

#### **Return Value**

AX = 0. (This function exists only for PC compatibility.)

[Top of list](#)

[Index page](#)

---

### **Interrupt 0x10 service 0x0E: Teletype output**

Write a character to the standard output.

#### **Parameters**

*AH*

Must be 0x0E

*AL*

Character to write

#### **Return Value**

Returns nothing.

#### **Comments**

This call returns immediately after space becomes available in the transmit ring buffer.  
(The data transfer from the transmit ring buffer to the hardware transmitter is interrupt driven.)

[Top of list](#)

[Index page](#)

---

### **Interrupt 0x10 service 0x0f: Get video state**

Get video state

#### **Parameters**

*AH*

must be 0x0F

## Return Value

number of screen columns, 80 in AH  
mode currently set, 1 in AL  
mode currently display page ,0 in BH

[Top of list](#)

[Index page](#)

---

## Interrupt 0x10 service 0x12: Video subsystem configuration

Video subsystem configuration

### Parameters

*AH*  
must be 0x12

### Return Value

AX=0x0012  
BX=0  
CX=0

[Top of list](#)

[Index page](#)

---

## Interrupt 0x16 service 0x00: Get char from standard input

Get a character from std in, wait if none available

### Parameters

*AH*  
Must be 0.

### Return Value

Returns character in AL  
Returns at DX the source stdin channel of the character: 1: EXT , 2: COM , 4: Telnet

### Comments

Please note that AH does not contain the scan code, but is always 0.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x16 service 0x01: Check if a character is available from std in

Check if a character is available from standard input

## Parameters

*AH*

Must be 1.

## Return Value

AX=1 if a character is available, AX=0 and zero-flag is cleared if no character is available.

## Comments

Please note that AH does not contain the scan code, but is always 0.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x00: Get serial number

Get the serial number of the CPU device

## Parameters

*AH*

Must be 0.

## Return Value

AX=low word, BX=high word.

## Comments

The serial number is a 24 bit value.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x01: Get IP address of the Ethernet interface

Get the IP address as a string.

## Parameters

*AH*

Must be 1.

*ES:DX*

Pointer to a 16 byte memory area where the IP address is to be stored as a null terminated string.

## Related Topics

Ethernet [IP address](#) initial value

[Top of list](#)

## Interrupt 0xA0 service 0x02: Set IP address of the Ethernet interface

Set the Ethernet interface's IP address based on the supplied string.

### Parameters

*AH*  
Must be 2.

*ES:DX*  
Pointer to a 16 byte memory area where the IP address is stored as a null terminated string.

### Comments

A new IP configuration must be activated by calling the `ipeth` [command](#) or by calling the TCP/IP API interrupt 0xAC [service 0x71](#) (`RECONFIG_ETHERNET`).

### Related Topics

TCP/IP API [RECONFIG\\_ETHERNET](#) service

Ethernet [IP address](#) initial value

[IP](#) command line

Important:

This API function writes to `chip.ini` and is not reentrant.

Don't use in different tasks or in combination with BIOS commands, which are writing to `chip.ini`, e.g. DHCP. Avoid race conditions with any other API call, which writes or read also to/from `chip.ini`

e.g. [service 0x23](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x03: Get IP subnet mask of the Ethernet interface

Get the IP subnet mask as a string.

### Parameters

*AH*  
Must be 3.

*ES:DX*  
Pointer to a 16 byte memory area where the IP subnet mask is to be stored as a null terminated string.

### Related Topics

Ethernet [IP subnet mask](#) initial value

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x04: Set IP subnet mask

Set the IP subnet mask to the string supplied

### Parameters

*AH*  
Must be 4.

*ES:DX*  
Pointer to a 16 byte memory area where the IP subnet mask is stored as a null terminated string.

### Comments

A new IP configuration must be activated by calling the `ipeth command` or by calling the TCP/IP API interrupt 0xAC [service 0x71](#) (`RECONFIG_ETHERNET`).

### Related Topics

Ethernet [IP subnet mask](#) initial value

[NETMASK](#) command line

TCP/IP [API](#) documentation

Important:

This API function writes to `chip.ini` and is not reentrant.

Don't use in different tasks or in combination with BIOS commands, which are writing to `chip.ini`, e.g. DHCP. Avoid race conditions with any other API call, which writes or read also to/from `chip.ini`

e.g. [service 0x23](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x05: Get IP gateway

Get the IP gateway as a string.

### Parameters

*AH*  
Must be 5.

*ES:DX*  
Pointer to a 16 byte memory area where the IP gateway is to be stored as a null terminated string.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x06: Set IP default gateway

Set the IP gateway to the string supplied

### Parameters

*AH*

Must be 6.

*ES:DX*

Pointer to a 16 byte memory area where the IP gateway is stored as a null terminated string.

## Comments

A new IP configuration must be activated by calling the `ipeth` [command](#) or by calling the TCP/IP API interrupt 0xAC [service 0x71](#) (`RECONFIG_ETHERNET`).

Important:

This API function writes to `chip.ini` and is not reentrant.

Don't use in different tasks or in combination with BIOS commands, which are writing to `chip.ini`, e.g. DHCP. Avoid race conditions with any other API call, which writes or read also to/from `chip.ini`

e.g. [service 0x23](#) The TCPIP stack of the IPC@Chip supports only one valid default gateway for all device interfaces: Ethernet, pppserver and pppclient.

The `ipcfg` [command](#) shows the current default gateway.

## Related Topics

Ethernet default [gateway](#) initialization

[GATEWAY](#) command line

[ADD\\_DEFAULT\\_GATEWAY](#) API function

PPP server default [gateway](#) initialization

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x07: Execute a command shell command

Passes a command string to the command interpreter.

### Parameters

*AH*

Must be 7.

*ES:DX*

Pointer to a null terminated command line.

### Comments

Internal commands are processed in the current task, external commands (.exe files) are loaded and executed in a new task.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x08: Set timer 0x1C's interval

Define the interval in milliseconds that timer interrupt 0x1C will use.

### Parameters

*AH*

Must be 0x08.

*BX*

Interval in milliseconds

## Comments

Use `setvect(0x1c, your_routine)` to change the interrupt vector.

Define your routine as:

```
void interrupt my_function(void)
```

You must restore the old timer interrupt vector before ending the program.

Your interrupt routine must be as short as possible without any waiting or endless loops.

Avoid the usage of large CLib functions like `printf`.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x09: Set timer interrupt 0xAF's interval

Define the interval in milliseconds that timer 0xAF will use

### Parameters

*AH*

Must be 0x09.

*BX*

Interval in milliseconds

## Comments

Use `setvect(0xAF, your_routine)` to change the interrupt vector.

Define your routine as:

```
void interrupt my_function(void)
```

You must restore the old timer interrupt vector before ending the program.

Your interrupt routine must be as short as possible without any waiting or endless loops.

Avoid the usage of large CLib functions like `printf`.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x11: Set STDIO focus

Set the focus of STDIO to either console, application or both

### Parameters

*AH*

Must be 0x11

*AL*

1: command shell (console), 2: Application, 3: both

## Comments

If your application requires input from the user, you should set the focus to the application. You should take care that only one application requests input from STDIO. The user can change the focus by using the focus hot key (default Ctrl-F). Changing the focus clears the serial input and output queues immediately.

**Important :**

All buffered incoming and outgoing characters in the internal serial send and receive queues are lost after this call.

## Related Topics

[Focus](#) key definition

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x12: Get bootstrap version number

Get the version number of the bootstrap loader.

### Parameters

*AH*

Must be 0x12

### Return Value

AX=version number, AH is major version, AL is minor version.

### Comments

Example:

If the function returns 0x0100 in AX, this means that you have version 1.00.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x13: Get BIOS version number

Get the version number of the BIOS.

### Parameters

*AH*

Must be 0x13

### Return Value

AX=version number, AH is major version, AL is minor version.

### Comments

Example:

If the function returns 0x0043 in AX, this means that you have version 0.67.

## Interrupt 0xA0 service 0x14: Set batch file execution mode.

Sets the batch file execution mode of DOS programs for either concurrent or sequential execution. See BATCHMODE initialization [documentation](#) for details.

### Parameters

*AH*

Must be 0x14

*AL*

AL = 0: (Selects default BATCHMODE=0 , = concurrent)

AL = 1: (Sets BATCHMODE=1 , = sequential)

*BX*

BX = 0: Disable the max. delayed execution timeout of DOS programs

BX = 1: Enable the max. delayed execution timeout of DOS programs at a batchfile, if BATCHMODE=1

### Return Value

returns nothing

### Comments

#### Important:

If BATCHMODE=1 take care that every program in your batch file which has a successor program either exits ([int21h 0x4C](#)) or terminates resident with [int21h 0x31](#).

A program which runs forever should call from the main function BIOS Interrupt [0xA0 Service 0x15](#), which immediately enables the further batch file sequencing.

By default the maximum delay time for execution of the next listed program in the batch file is 15 seconds.

If BX is set to 0, the successor program at a batch file waits forever for execution, if the predecessor program not finishes or calls [0xA0 Service 0x15](#)

### Related Topics

Initial [batch mode](#) configuration  
[BATCHMODE](#) command

## Interrupt 0xA0 service 0x15: Allow immediate further batch file execution in BATCHMODE 1.

This call allows the next program listed in a batch file to start execution. This is implemented by waking up a batch file execution task which dispatches any subsequent program listed in the batch file.

### Parameters

AH

Must be 0x15

## Return Value

returns nothing

## Related Topics

Initial [batch mode](#) configuration

Run-time [batch mode](#) selection

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x16: Get information about the BIOS features

Get information about running servers, interfaces and features of the BIOS

### Parameters

AH

0x16

### Return Value

Bits of AX, BX and DX indicate the services or devices available, coded as:

Bit=0: service or device is not available.

Bit=1: service or device is available.

AX:

Bit 0: Ethernet device for TCPIP

Bit 1: PPP server

Bit 2: PPP client

Bit 3: Web server

Bit 4: Telnet server

Bit 5: FTP server

Bit 6: TFTP server

Bit 7: DHCP client

BX:

Bit 0: SNMP MIB variables support (see [TCPIP API service 0x71](#))

Bit 1: UDP config server

Bit 2: Ping client (see [TCPIP API service 0x75](#))

DX:

Bit 0: I2C-Bus API

Bit 1: Hardware API

Bit 2: RTOS API

Bit 3: Packet driver interface for ethernet

Bit 4: Serial XMODEM filetransfer

Bit 5: External disk interface

---

## Developer Notes

Since BIOS version 1.01 we also use the BX register for returning information about the BIOS features.

## Interrupt 0xA0 service 0x17: Get MAC address of the Ethernet interface

Get the MAC address as a 6 Byte array.

### Parameters

*AH*  
Must be 0x17.

*ES:DX*  
Pointer to a 6 byte memory area where the MAC address is to be stored.

## Interrupt 0xA0 service 0x18: Power save

Slows down the internal timer for the RTOS and puts the CPU in a halt mode until the next interrupt occurs. Please note that the internal time/date will be affected.

### Parameters

*AH*  
Must be 0x18.

### Comments

Call this function when your program is in idle state.  
Power savings are marginal since we use a DRAM. Please note that power consumption may differ slightly when the date code of the IPC@Chip is changed.

## Interrupt 0xA0 service 0x19: Change level for configuration server.

Change the supported level for the configuration server. For a description of the possible levels, please refer to [config.htm](#) document.

### Parameters

*AH*  
Must be 0x19.

*BX*  
The supported level.

### Comments

Please note that if the level defined in the [chip.ini](#) is 0 (zero), the configuration server task is not started and changing the

supported level does not have any effect. To avoid this, use a unlisted support level such as 0x1000 in the `chip.ini`. The entry in the `chip.ini` file is not changed by this call.

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA0 service 0x20: Install a user fatal error handler

Install a user fatal error function. This function will be called at fatal errors at execution time. (for further details see comment)

### Parameters

*AH*  
Must be 0x20.

*ES:DI*  
Address of the user error handler function

### Comments

The user should be able to call an error handler function, if a fatal error occurs at his application or inside of the SC12 BIOS.

This mechanism will (of course) fail if the code of the user error handler is itself overwritten (corrupt).

Function must be from type `huge _pascal` with a parameter `int errorcode` (see the following example). The SC12 BIOS calls the user error with handler an error code, for detecting the cause of the occurred error inside of the user error handler function.

If an user error handler is defined, it will be called from the SC12 BIOS with following errorcodes:

- 1: Invalid processor opcode (usually caused by corrupted memory), the calling task is suspended
- 2: Fatal kernel error (usually caused by corrupted memory or a taskstack overflow)with errorcode 2
- 3: Fatal internal TCPIP error, calling task is suspended
- 4: TCPIP stack reaches memory limit
5. TCPIP memory allocation error

In all cases we recommend a reboot with BIOSint 0xA0 0x21 (see below).

Important: Do not use any message printing inside your errorhandler if errorcode is 3 or 4, because if telnet is part of your stdio, your exit handler hangs inside of the print call.

```
//example for Borland C
void huge _pascal user_error_handler(int errorcode)
{
    union REGS inregs;
        union REGS outregs;

    //e.g. resetting outputs
    outportb(0x600,0x00);

    //rebooting the SC12
    inregs.h.ah = 0x21;
    int86(0xA0,&inregs,&outregs);
}

//install the function at main
inregs.h.ah=0x20;
sregs.es =FP_SEG(user_error_handler);
inregs.x.di=FP_OFF(user_error_handler);
int86x(0xA0,&inregs,&outregs,&sregs);
```

[Top of list](#)

## Interrupt 0xA0 service 0x21: Rebooting the SC12

This function works in the same way than the shell command reboot

### Parameters

*AH*  
0x21

### Return Value

Reboot: No return from this function

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x22: Get version string

Copies the BIOS version information in to a text buffer. The string is null terminated

### Parameters

*AH*  
Must be 0x22

*CX*  
Buffer length, including space for null terminator

*ES*  
Segment of memory buffer for the string

*DI*  
Offset of memory buffer for the string

### Related Topics

[VER](#) VER command

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x23: Insert an entry at chip.ini

The functions 0x23 and 0x24 allows the user to modify/place and find/read own chip.ini entries.

### Parameters

*AH*  
Must be 0x23.

*BX:SI*

Pointer to section string (max. 40 chars)

*ES:DI*

Pointer to item name (max. 40 chars)

*DS:DX*

Pointer to item text (max. 40 chars)

## Return Value

AX=0 success , AX=-1 invalid string length

## Comments

Important: The API functions 0x23 and 0x24 are not reentrant. Don't use in different tasks or in combination with BIOS commands, which are writing to chip.ini e.g. DHCP. Avoid race conditions with any other API call, which writes or read also to/from chip.ini e.g. [service 0x02](#)

Example(tested with Borland C/C++ 5.02:

```
int iniPutString(char *sectionName, char *itemName, char *text)
{
    union REGS  inregs;
    union REGS  outregs;
    struct SREGS sregs;

    inregs.h.ah = 0x23;
    inregs.x.bx = FP_SEG(sectionName);
    inregs.x.si = FP_OFF(sectionName);
    sregs.es   = FP_SEG(itemName);
    inregs.x.di = FP_OFF(itemName);
    sregs.ds   = FP_SEG(text);
    inregs.x.dx = FP_OFF(text);
    int86x(0xA0, &inregs, &outregs, &sregs);
    return outregs.x.ax;
}
//Call of this function:
iniPutString("MY_SECTION", "MY_ITEM", "VALUE_TEXT");
//and produces the following chip.ini entry
[MY_SECTION]
MY_ITEM=VALUE_TEXT
<nl>
```

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x24: Find an entry at chip.ini

Find a entry at chip.ini

### Parameters

*AH*

Must be 0x24.

*CX*

Maxlen for target string.

*BX:SI*  
Pointer to section string

*ES:DI*  
Pointer to item name

*DS:DX*  
Pointer to target

## Return Value

AX=0: Entry not found  
else Success: pointer at DS:DX contains the found string AX contains length of the found string

## Comments

Important: API functions 0x23 and 0x24 are not reentrant.  
Don't use in different tasks or in combination with BIOS commands, which are writing to chip.ini, e.g. DHCP. Avoid race conditions with any other API call, which writes or read also to/from chip.ini  
e.g. 0xA0 0x02 Set IP address

Example (tested with Borland C/C++ 5.02:

```
int iniGetString(char *sectionName, char *itemName, char *target, int maxlen)
{
    union REGS inregs;
    union REGS outregs;
    struct SREGS sregs;

    inregs.h.ah = 0x24;
    inregs.x.bx = FP_SEG(sectionName);
    inregs.x.si = FP_OFF(sectionName);
    sregs.es = FP_SEG(itemName);
    inregs.x.di = FP_OFF(itemName);
    inregs.x.cx = maxlen;
    sregs.ds = FP_SEG(target);
    inregs.x.dx = FP_OFF(target);
    int86x(0xA0, &inregs, &outregs, &sregs);
    return outregs.x.ax;
}
//Declare a target buffer
unsigned char target[100];
//After the of this function
iniGetString("MY_SECTION", "MY_ITEM", target, 50);
//target contains the chip.ini text for this item
```

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x25: Set the Stdio Focuskey

Set the Stdio Focuskey

### Parameters

*AH*  
Must be 0x25.

*AL*

Focuskey character (default CTRL-F, ASCII 6)

## Return Value

Returns nothing

## Comments

By default, the focus key is set to CTRL-F (ASCII 6)  
At runtime, the pressed key Ctrl-F toggles between these three modes and shows the current mode.

Key Range: 0..254

If the key is set to zero, the switching of stdio is disabled.  
The focus key is not usable by the command shell or dos executable.

## Related Topics

[Focus](#) key definition

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x26: Get the IPC@Chip device names

Get the IPC@Chip device names

## Parameters

*AH*

Must be 0x26.

## Return Value

AX=0  
ES:DI contains pointer to the fixed device name stored at the IPC@Chip flash  
BX:SI contains pointer to the device name configured at chip.ini.

## Related Topics

Devicename [Device name](#) definition

---

## Developer Notes

You cannot write at the pointer es:di, because it is located at the flash memory  
Both returned strings are terminated by 0.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x27: Suspend/Resume System Servers

Suspend/Resume FTP, Telnet or Web Server

## Parameters

*AH*  
Must be 0x27.

*AL*  
0: Resume, 1: Suspend

*BX*  
0: FTP Server, 1: Telnet Server, 2: Web Server

## Return Value

AX=0: Success  
AX=1: Server was already in the postulated state  
AX=-1: Invalid Parameter

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA0 service 0x28: Fast Findfirst

Provide a faster access to the filesystems directories

## Parameters

*AH*  
Must be 0x28.

*CX*  
File attribute

*BX:SI*  
Nullterminated File Specification

*ES:DI*  
Pointer to fjxfind struct

## Return Value

AL=1 DX=0: Success  
AL=0 DX=0: no file found  
DX=-1: Findfirst already active

## Comments

The three filefind functions (0x28 - 0x30) provides a faster Findfirst/next access than dos compatible functions at INT21h. They work in a similiar way as the INT21h Findfirst/next functions. You have to call at first Findfirst (0x28). After that call, you could call Findnext (0x29) as much as you need. To get the hole directory call it until it returns an error. The directory being searched will be locked for exclusiv access by the calling task. To unlock the directory you have to call Finddone (0x30).

Note: This function call (or the Fast Findnext function call) must be followed by a call of the **Fast Finddone** function!

These functions (0x28 - 0x30) are not reentrant, do not call findfirst/findnext sequences from different tasks without semaphore protection.

The filefind struct is defined as the following description:

```

typedef struct filefind
{
    char                filename[12]; // Null terminated filename
    char                fileext[4];  // and extension
    unsigned short int fileattr;     // MS-DOS file attributes
    short int           reserved;    // Reserved
    struct tag_filetimestamp
    {
        unsigned short int filedate; // Date = ((year - 80) shl 9) or (month shl 5) or
day
        unsigned short int filetime; // Time = (hour shl 11) or (min shl 5) or
(sec / 2)
    }filetimestamp; // Time & date last modified
    unsigned long      filesize; // File size
}

```

## Related Topics

[Fast Findnext](#)

[Fast Finddone](#) must be called at end of the search

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x29: Fast Findnext

Continues a search which was started by Fast Findfirst (0x28)

### Parameters

*AH*

Must be 0x29.

*ES:DI*

Pointer to filefind struct

### Return Value

AL=1 DX=0: Success

AX=0 DX=0: no file found

DX=-1: Findfirst not called before

### Comments

See also [Fast Findfirst function \(0x28\)](#) for description.

Note: A call of this function must be followed by a call of the [Fast Finddone](#) function!

## Related Topics

[Fast Findfirst](#) must be called to start the search

[Fast Finddone](#) must be called at end of the search

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x30: Fast Finddone

Close a Findaccess started with Fast Findfirst. Do not call without Findfirst call before.

### Parameters

*AH*  
Must be 0x30.

*ES:DI*  
Pointer to filefind struct

### Return Value

AX=0 DX=0: Success  
DX=-1: Findfirst not active

### Comments

See also Fast Findfirst function (0x28) for description.

### Related Topics

[Fast Findfirst](#) must be called before Finddone  
[Fast Findnext](#)

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA0 service 0x31: Detect ethernet link state

Detect ethernet link state

### Parameters

*AH*  
Must be 0x31.

### Return Value

AX=0: Link ok  
AX!=0: No Link, no cable connected??

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA0 service 0x32: Set a memory gap between the loaded DOS programs

Set a memory gap between the loaded DOS programs as a memory reserve.  
Some programs compiled with Borland C 5.02 (other compilers??) tries to increase their program memory block at runtime before they e.g. opening a file with Borland C-Library function fopen, because it requires some more memory. The programs calls int21h 0x4A, this happens inside of the e.g. Borland C-Library fopen function and is not visible for the application programmer. This memory resize call failed, if another program is loaded after the previous one, because now there is no memory space left for increasing the memory size of the previous executed program. The program returns at fopen with an error. The global program variable errno is set to value 8(not enough memory). To prevent this error BIOS 1.02B allows to

define the size of a memory gap between two loaded programs. The value must be defined as a number of paragraphs (1 paragraph==16 Bytes). This strategy could fail in that cases, when programs are terminated and restart again.

## Parameters

*AH*  
Must be 0x32.

*BX*  
Number of paragraphs (range between 0 to 2048 paragraphs).

## Return Value

AX=0 DX=0: Success  
AX=-1: Invalid value at bx

## Comments

Value can also defined at [chip.ini](#)

---

## Developer Notes

It is not necessary to set this entry, if the application doesn't show the described error. Only if a C-Library function call sets errno to 8, this value should be defined. We recommend in that case a value of 128 paragraphs (2048 Bytes). The described problem was noticed, if the BorlandC-Library function fopen (...) was used. Same happens at memory model Large and the usage of CLIB function malloc. Malloc returns a NULL pointer.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x33: Set stdin/stdout channel

Set the stdin/stdout channel

## Parameters

*AH*  
Must be 0x33.

*AL*  
Bit 0 = 1 set stdout, Bit 1 = 1 set stdin

*BX*  
Channel bits, see comment

## Return Value

AX=0 DX=0: Success  
AX=DX=-1: Invalid parameter

## Comments

Channel bits for bx register:

Bit 0: Serial port 0 (PORT EXT)  
Bit 1: Serial port 1 (PORT COM)  
Bit 2: Telnet server  
Bit 3: User channel

Setting a bit to zero deactivates this channel, setting a bit to one activates the specified channel

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA0 service 0x34: Get stdin/stdout settings

Get the stdin/stdout settings

### Parameters

*AH*  
Must be 0x34.

### Return Value

AX=0, BX contains stdout settings, CX: stdin settings

### Comments

Return values  
Bit 0 == 1: Serial port 0 (EXT)  
Bit 1 == 1: Serial port 1 (COM)  
Bit 2 == 1: Telnet server  
Bit 3 == 1: User channel

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA0 service 0x35: Install specific user stdio handlers

Install specific user stdio channel handlers

This API call allows the user to install own stdio handler functions, e.g. for an own developed input/output device connected to the IPC@Chip (e.g. a display and/or keyboard device) or an own TCP application similiar to telnet.

The user must implement inside of his application four functions for reading and writing characters from/to his own stdin/stdout device.

After installing these functions with this API call and setting stdin and/or stdout to the user channel with [Set stdio channels](#) the @Chip-RTOS of the IPC@Chip calls these user functions at every stdin and stdout operation. For further details of programming and installing those handler functions see the comment below.

### Parameters

*AH*  
Must be 0x35.

*ES:DI*  
Pointer to struct variable of User\_Stdio\_Funcs, see comment

### Return Value

AX=0 DX=0: Success

## Comments

Necessary type definitions:

```
typedef int (huge *User_Kbhit)(void);
typedef void (huge *User_PutCh)(char chr);
typedef void (huge *User_PutStr)(char * pch, int n);
typedef int (huge *User_Getch)(void);
```

```
typedef struct tag_user_stdio
{
    User_Kbhit user_kbhit;
    User_Getch user_getch;
    User_PutCh user_putch;
    User_PutStr user_putstr;
}User_Stdio_Funcs;
```

Functions to implement by the user:

```
int huge user_kbhit(void); //returns 1 ,if a character is available, 0 if not
int huge user_getch(void); //read a char from stdin (wait, if none available)
void huge user_putch(char chr); //write a single char to stdout
void huge user_putstr(char * pch, int n); //write a string with n chars to stdout
```

The user must set the pointers at his variable from type User\_Stdio\_Funcs and call the API function with the address of User\_Stdio\_Funcs at es:di. With es = di = 0, the user uninstalls his stdio handler.

Important:

1. If your applications exits, don't forget to uninstall your stdio handler, by calling this API call with es=di=0.
2. Do not call your implemented stdio functions inside your application, these installed functions must be called only inside of the @Chip-RTOS.

Example(Borland C):

```
User_Stdio_Funcs user_stdio_funcs; //global variable
```

```
//implementation of users stdio functions
```

```
int huge my_kbhit(void)
{
    //.....
}
```

```
char huge my_getch(void)
{
    //.....
}
```

```
void huge my_putch(char chr)
{
    //.....
}
```

```
void huge my_putstr(char * pch, int n)
{
    //.....
}
```

```
void install_mystdio_channel(void)
{
    union REGS inregs;
    union REGS outregs;
```

```

struct SREGS sregs;

user_stdio_funcs.user_kbhit = my_kbhit;
user_stdio_funcs.user_getch = my_getch;
user_stdio_funcs.user_putch = my_putch;
user_stdio_funcs.user_putstr = my_putstr;

inregs.h.ah=0x35;
sregs.es =FP_SEG(&user_stdio_funcs);
inregs.x.di=FP_OFF(&user_stdio_funcs);
int86x(0xA0,&inregs,&outregs,&sregs);
}

void remove_mystdio_channel(void)
{
    union REGS inregs;
    union REGS outregs;

    inregs.h.ah=0x35;
    sregs.es = 0;
    inregs.x.di= 0;
    int86x(0xA0,&inregs,&outregs,&sregs);
}

unsigned int set_stdio_channel(unsigned int channels)
{
    union REGS inregs;
    union REGS outregs;

    inregs.h.ah=0x35;
    inregs.h.al=3; //set both: stdin and stdout
    inregs.x.bx = channels;
    int86(0xA0,&inregs,&outregs);
    return outregs.x.ax;
}

int main(void)
{
    //.....
    install_mystdio_channel();
    set_stdio_channel(0x0D); //COM,TELNET,USER
    //....

    //at the end of program

    set_stdio_channel(0x06); //COM,TELNET
    remove_mystdio_channel();
}

```

[Top of list](#)

[Index page](#)

## Interrupt 0xA0 service 0x36: Install a System Server Connection Handler function

Install a user specific System Server Connection Handler function.

The handler function will be called if a client establishes a connection. This functions allows application programmers to implement own callback functions for controlling access to the default @Chip-RTOS servers.

### Parameters

*AH*

Must be 0x36.

*BX*

0: FTP Server, 1: Telnet Server, 2: Web Server

ES:DI

Pointer to handler function

## Return Value

AX=0: Success

AX=-1: Invalid Parameter

## Comments

A connection Handler function must be declared in the following way:

```
int huge UserConnectionHandler( struct sockaddr_in *sockptr );
```

The connection handler will be called if a client establishes a connection to the server (FTP, WEB, Telnet). The handler could read the IP Address and the Port in the [sockaddr\\_in struct](#) and using TCPIP Api function 0x11 API\_INETTOASCII (see [TCPIP\\_API](#) description). If the handler return 0 the connection will be established. If it returns a value unequal to 0, the connection will be abort.

Example for usage:

The implemented handler function could check the source IP address (Clients IP), compare this IP with an application internal list of allowed IP addresses and reject the connection by returning a value unequal to zero, if the source IP is not a member of the list.

To uninstall a connection handler call this function with a Null pointer (es=di=0).

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA0 service 0x37: Enable/Disable Filesharing

Enable/disable Filesharing of Int21h open/create

### Parameters

AH

Must be 0x37.

AL

0: set mode, 1: get mode

BX

0: disable, 1: enable (Sharing mode)

### Return Value

AX=0: Success, contains Sharing mode if al=1

AX=-1: Invalid Parameter

### Comments

As default the filesharing is disabled. This has the effect, that a file which is opened for write access can't be opened a second time for read or write access. Also a file which is opened for read access can be only opened for read access for a second time.

To avoid this security feature you can enable the filesharing. This could be also done with the CHIP.INI entry [FILESHARING](#).

NOTE: Be careful by open a file two times with one or two write accesses!

## Interrupt 0xA0 service 0x38: Get Filename by Handle

Returns the Filename string depending on a specific filehandle.

### Parameters

*AH*  
Must be 0x37.

*CX*  
Filehandle

### Return Value

ES:BX conatins Pointer to Filename string AX=0: Success, contains Sharing mode if al=1  
AX=-1: Invalid Filehandle

## CHIP.INI Documentation - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

Configuration [News](#)

---

### CONFIG

The IPC@Chip system configuration is controlled via the CHIP.INI file.  
At startup, the system reads the file `A:\chip.ini` and uses the settings found here to initialize the system.

### CONFIG [News](#)

- [STDIO\\_STDIN](#)
- [STDIO\\_STDOUT](#)
- [STDIO\\_FOCUS](#)
- [STDIO\\_FOCUSKEY](#)
- [STDIO\\_CTRL\\_C](#)
- [IP\\_ADDRESS](#)
- [IP\\_NETMASK](#)
- [IP\\_GATEWAY](#)
- [IP\\_DHCP](#)
- [IP\\_TCPIPMEM](#)
- [UDPCFG\\_LEVEL](#)
- [PPPSERVER\\_ENABLE](#)
- [PPPSERVER\\_MODEMTRACE](#)
- [PPPSERVER\\_COMPORT](#)
- [PPPSERVER\\_ADDRESS](#)
- [PPPSERVER\\_REMOTEADDRESS](#)
- [PPPSERVER\\_NETMASK](#)
- [PPPSERVER\\_GATEWAY](#)
- [PPPSERVER\\_AUTH](#)
- [PPPSERVER\\_IDLETIME](#)
- [PPPSERVER\\_FLOWCTRL](#)
- [PPPSERVER\\_MODEM](#)
- [PPPSERVER\\_USERx](#)
- [PPPSERVER\\_PASSWORDx](#)
- [PPPSERVER\\_BAUD](#)
- [PPPSERVER\\_INITCMDx](#)

- PPPSERVER\_INITANSWERx
- PPPSERVER\_INITTIMEOUTx
- PPPSERVER\_INITRETRIESx
- PPPSERVER\_MODEMCTRL
- PPPSERVER\_CTRLTIME
- PPPSERVER\_CTRLCMDx
- PPPSERVER\_CTRLANSWERx
- PPPSERVER\_CTRLTIMEOUTx
- PPPSERVER\_CTRLRETRIESx
- PPPSERVER\_CMDMODE
- PPPSERVER\_HANGUPDELAY
- PPPSERVER\_HANGUPCMDx
- PPPSERVER\_HANGUPANSWERx
- PPPSERVER\_HANGUPTIMEOUTx
- PPPSERVER\_HANGUPRETRIESx
- PPPSERVER\_CONNECTMSGx
- PPPSERVER\_CONNECTANSWERx
- PPPSERVER\_CONNECTTIMEOUTx
- RAMDRIVE\_SIZE
- TIMER\_1C
- TIMER\_AF
- FTP\_ENABLE
- FTP\_CMDPORT
- FTP\_LOGINDELAY
- FTP\_TIMEOUT
- FTP\_USERx
- FTP\_PASSWORDx
- FTP\_ACCESSRIGHTx
- FTP\_DRIVEx
- FTP\_ROOTDIRx
- WEB\_ENABLE
- WEB\_MAINPAGE
- WEB\_TEMPPATH
- WEB\_DRIVE
- WEB\_ROOTDIR
- WEB\_MAXCGIENTRIES
- WEB\_WEBSERVERSTACK
- WEB\_HTTPPORT
- WEB\_USER0
- WEB\_PASSWORD0
- TFTP\_TFTPPORT
- TELNET\_TELNETPORT
- TELNET\_TIMEOUT
- TELNET\_LOGINDELAY
- TELNET\_LOGINRETRIES
- TELNET\_USERx
- TELNET\_PASSWORDx
- TELNET\_ENABLE

- [DEVICE FILESHARING](#)
  - [DEVICE\\_NAME](#)
  - [SERIAL\\_EXT\\_DMA](#)
  - [SERIAL\\_COM\\_DMA](#)
  - [SERIAL\\_SEND\\_DMA](#)
  - [SERIAL\\_EXT\\_RECVQUEUE](#)
  - [SERIAL\\_EXT\\_SENDQUEUE](#)
  - [SERIAL\\_COM\\_RECVQUEUE](#)
  - [SERIAL\\_COM\\_SENDQUEUE](#)
  - [SERIAL\\_COM\\_BAUD](#)
  - [SERIAL\\_EXT\\_BAUD](#)
  - [DOSLOADER\\_MEMGAP](#)
  - [BATCH\\_BATCHMODE](#)
  - [BATCH\\_EXECTIMEOUT](#)
- 

## STDIO

**[STDIO]**  
**STDIN=Define standard input device**

Define your device for standard input.  
Valid devices are COM, EXT and TELNET. You can define several devices simultaneously.

## Comments

The following example defines both COM and TELNET for stdin:

```
[STDIO]  
STDIN=COM TELNET
```

By default, both COM and TELNET are used.

[Top of list](#)

[Index page](#)

---

## STDIO

**[STDIO]**  
**STDOUT=Define standard output device**

Define your device for standard output.  
Valid devices are COM, EXT and TELNET. You can define several devices simultaneously.

## Comments

The following example defines both COM and TELNET for stdout:

```
[STDIO]
STDOUT=COM TELNET
```

By default, both COM and TELNET are used.

[Top of list](#)  
[Index page](#)

---

## STDIO

### [STDIO] FOCUS=Command shell and/or user executables

Set the stdio focus to the command shell and/or to the user executables.

Valid entries are USER or SHELL

If only USER is defined, stdio in the command shell is suppressed.

If only SHELL is defined, stdout and stdin in the users DOS executables are disabled.

## Comments

The following example enables stdio for both USER and SHELL:

```
[STDIO]
FOCUS=SHELL USER
```

By default, stdin and stdout for both SHELL and USER are enabled.

**Important :** If stdio is enabled for both, there is a rivalry between USER and SHELL.

At runtime, pressing of the focus key (default is Ctrl-F) toggles between these three modes and shows the current mode.

[Top of list](#)  
[Index page](#)

---

## STDIO

### [STDIO] FOCUSKEY=Key

Set the key to switch the stdio focus

## Comments

The following example sets Ctrl-F (ASCII 6) as the current stdio focus key:

```
[STDIO]
FOCUSKEY=6
```

By default, the focus key is set to CTRL-F (ASCII 6)  
At runtime, the pressed key Ctrl-F toggles between these three modes  
and shows the current mode.

Key Range: 0..254

If the key is set to zero, the switching of stdio is disabled.  
The focus key is not usable by the command shell or dos executable.

[Top of list](#)  
[Index page](#)

---

## STDIO

```
[STDIO]
CTRL_C=0/1
```

Disable/enable terminating the execution of autoexec.bat via ctrl-c key.  
The following example disables the ctrl-c control.

```
[STDIO]
CTRL_C=0
```

By default, CTRL\_C is enabled.

[Top of list](#)  
[Index page](#)

---

## IP

```
[IP]
ADDRESS=IP Address of the Ethernet interface
```

Defines the IP address if no DHCP is used.

### Comments

Only numerical IP addresses are allowed here.

Example: ADDRESS=192.168.200.1

If no address entry was found, the IP address will be set to 1.1.1.1.

### Related Topics

[IP](#) command line

[Top of list](#)  
[Index page](#)

---

## IP

### [IP] **NETMASK=IP Address mask of the Ethernet interface**

Defines the IP address subnet mask if no DHCP is used.

## Comments

Example: `NETMASK=255.255.255.224`

If no subnet mask entry was found, the subnet mask will be set to `255.255.255.0`.

## Related Topics

[NETMASK](#) command line  
Set IP subnet mask [API](#) function

[Top of list](#)  
[Index page](#)

---

## IP

### [IP] **GATEWAY=Gateway IP Address of the Ethernet interface**

Defines the IP address of the gateway.

## Comments

Example: `GATEWAY=195.243.140.65`

If no gateway entry was found, the gateway address will be set to `0.0.0.0`.

The TCP/IP stack of the IPC@Chip supports only one valid default gateway for all device interfaces: Ethernet, PPP Server and PPP Client.

If you define a gateway in the PPPSERVER section of the `chip.ini` for the PPP server interface, it becomes the default gateway for all interfaces when a PPP link to the server is established. During a PPP server connection the command [ipcfg](#) indicates this PPP default gateway. After the PPP session, the old gateway (if any) for the Ethernet interface will be restored.

As of BIOS version 070, the TCP/IP API supports adding and deleting a default gateway:

- Interrupt 0xAC, [Service 0x80](#) : add a default gateway
- Interrupt 0xAC, [Service 0x81](#) : delete default gateway
- Interrupt 0xAC, [Service 0x82](#) : get default gateway

## Related Topics

TCP/IP [API](#) documentation

[GATEWAY](#) command line

Set IP gateway [API](#) function

[ADD\\_DEFAULT\\_GATEWAY](#) API function

[Top of list](#)

[Index page](#)

---

## IP

### [IP] DHCP=0/1 Ethernet interface

Set to 1 if DHCP should be used to get the IP configuration from a DHCP server.  
If defined as 0, a static network configuration is used.

## Comments

Any settings for IP Address, subnet mask and gateway are ignored if DHCP is used.

## Related Topics

[DHCP](#) command line

[Top of list](#)

[Index page](#)

---

## IP

### [IP] TCPIPMEM=Size

Set the size of the TCP/IP memory block in kBytes. This block is allocated at the start of the TCP/IP stack.

Valid Range: Between 30 kBytes and 160 kBytes (An out of range value for TCPIPMEM will be set to closest of these limit values.)

Default value: 90 kBytes when BIOS configured without PPP capability (server or client)

98 kBytes when BIOS configured with PPP capability

Example: TCPIPMEM=60

## Comments

The TCP/IP API function 0x78, [GET\\_MEMORY\\_INFO](#), reports the current used memory of the TCP/IP stack

Since BIOS version 1.02B we allow configuring a max value of 160kBytes, because some application programmers may require more then the old limit of 132 kByte.

[Top of list](#)

[Index page](#)

---

## UDPCFG

**[UDPCFG]  
LEVEL=mask**

Defines the supported functions of the configuration server.

Set LEVEL as a bit mask to define the functions that the configuration server should listen to.

The bit assignments are as follows:

- 0x00 No function
- 0x01 Allow detection on the network.
- 0x02 Allow change of IP configuration.
- 0x10 Allow programming of flash.

If defined as 0, the configuration server task will not start.

Example: LEVEL=0x03

This would allow detection on the network and changing the IP configuration, but no BIOS update.

## Comments

By default, all options are enabled.

## Related Topics

Run-time adjustments to [LEVEL](#)

[Top of list](#)

[Index page](#)

---

## PPPSERVER

**[PPPSERVER]  
ENABLE=0/1**

Disable/enable PPP server

```
[ PPPSERVER ]  
ENABLE=1
```

By default, PPP server is enabled.

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

**[PPPSERVER]  
MODEMTRACE=0/1**

Disable/enable the trace of the control communication between SC12 and a connected modem. The defined modem strings(AT commands and answers) at chip.ini will be printed at STDOUT, if MODEM\_TRACE=1.

Could be useful for testing the modem configuration and debugging the ppp dial procedures.

```
[ PPPSERVER ]  
MODEMTRACE=1
```

By default, tracing is disabled.

## Comments

Received characters with an ASCII value smaller 0x20 are printed as numbers.

[Top of list](#)

[Index page](#)

---

## PPPSERVER

**[PPPSERVER]  
COMPORT=Define serial device for the PPP server**

Define your serial device for the PPP server.  
Valid devices are COM or EXT.

## Comments

The following example defines EXT as device for the PPP server:

```
[ PPPSERVER ]  
COMPORT=EXT
```

By default, no COM port is enabled.

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER] ADDRESS=IP Address of the PPP server interface

Defines the IP address for the PPP server.

## Comments

Only numerical IP addresses are allowed here.

Example: ADDRESS=192.168.205.1

If no address entry was found, the address will be set to 1.1.2.1.

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER] REMOTEADDRESS=IP Address for the remote PPP client

Defines the IP address for the remote PPP client.

## Comments

Only numerical IP addresses are allowed here.

Example: REMOTEADDRESS=192.168.205.2

If no address entry was found, the remote address will be set to 1.1.2.2.

## Related Topics

[Top of list](#)  
[Index page](#)

---

## PPPSERVER

### [PPPSERVER]

#### NETMASK=IP Address subnet mask of the PPP server

Defines the IP address subnet mask of the PPP server.

### Comments

Example: `NETMASK=255.255.255.0`

If no subnet mask entry was found, the subnet mask will be set to `255.255.255.0`.

### Related Topics

PPP server [configuration](#) instructions

[Top of list](#)  
[Index page](#)

---

## PPPSERVER

### [PPPSERVER]

#### GATEWAY=IP Address of gateway

Defines the IP address of the gateway for PPP interface.

### Comments

Example: `GATEWAY=195.243.140.65`

If no gateway entry was found, the gateway address will be set to `0.0.0.0`.

The TCP/IP stack of the IPC@Chip supports only one valid default gateway for all device interfaces: Ethernet, PPP Server and PPP Client.

If you define a gateway in the PPPSERVER section of the `chip.ini` for the PPP server interface, it becomes the default gateway for all interfaces when a PPP link to the server is established. During a PPP server connection the command [ipcfg](#) indicates this default gateway. After the PPP session, the old gateway (if any) for the Ethernet interface will be restored.

As of BIOS version 070, the TCP/IP API supports adding and deleting a default gateway:

- Interrupt 0xAC, [Service 0x80](#) : add a default gateway
- Interrupt 0xAC, [Service 0x81](#) : delete default gateway
- Interrupt 0xAC, [Service 0x82](#) : get default gateway

## Related Topics

PPP server [configuration](#) instructions

TCP/IP [API](#) documentation

Description of the [command processor](#)

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER] AUTH=0/1/2

Set PPP authentication mode for the remote PPP client

0: No authentication

1: PAP authentication

2: CHAP authentication

## Comments

The following example selects PAP authentication mode:

```
[PPPSERVER]  
AUTH=1
```

By default, authentication is disabled.

If AUTH=0 you must define two [username](#) / [password](#) pairs used to authenticate the PPP client. The client must use one of these pairs to get connected to the SC12 PPP server.

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER] IDLETIME=Seconds

Sets the idle time, after which the PPP server closes the connection.

```
[PPPSEVER]
IDLETIME=500
```

By default, PPP server idle time is 120 seconds. A value of 0 means no idle timeout.

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSEVER

```
[PPPSEVER]
FLOWCTRL=0/1/2
```

Set flow control mode of the PPP servers serial device:

0: none  
1: XON/XOFF (See caution below!)  
2: RTS/CTS

Example: Here XON/XOFF flow control is enabled

```
[PPPSEVER]
FLOWCTRL=1
```

By default, FLOWCTRL=2 (RTS/CTS)

## Comments

### Caution:

If you use the default DMA mode for the selected COM port, XON/XOFF mode is not available. XON/XOFF is available only if the serial [DMA mode](#) is disabled in `chip.inifile`.

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSEVER

```
[PPPSEVER]
```

## MODEM=0/1

Disable/enable usage of a modem

```
[ PPPSERVER ]  
MODEM=1
```

By default, the usage of a modem is disabled.

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

```
[PPPSERVER]  
USERx=user
```

Define the user name for PPP server, using PAP authentication

## Comments

You can define a USER0 and a USER1.  
Default user is 'ppps' , password is 'ppps' for both USER0 and USER1.  
You must specify both the user name and his password.  
Both user name and password must be lower case.  
The entries are only valid, if AUTH=1 is specified.

Maximum Name Size: 49 characters

Important notice: To avoid security leaks you must define both user names and passwords.

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

```
[PPPSERVER]  
PASSWORDx=password
```

Define the password for a PPP server user, using PAP authentication.

## Comments

You can define a PASSWORD0 for USER0 and a PASSWORD1 for USER1  
Default user is 'ppps' , password is 'ppps'.  
Both user name and password must be lower case.  
The entries are only valid, if AUTH=1 is specified

Maximum Password Size: 49 characters

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER] BAUD=BAUD Rate

Sets the BAUD rate of the PPP server COM port.

## Comments

The following example sets the PPP server COM port to 19,200 BAUD.

```
[PPPSERVER]  
BAUD=19200
```

By default, PPP server BAUD rate is 38400 (with 8 data bits, no parity, 1 stop bit).

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER] INITCMDx=modem command

Defines modem commands to initialize your modem connected to the SC12

at the start of the SC12 PPP server and after a modem hang-up following a PPP session.

## Comments

You can define a maximum of three modem commands e.g. `INITCMD0=ATZ` .  
The entries are only valid, if `MODEM=1` is specified.

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER] INITANSWERx=modems answer of init command x

Defines the expected modem answer x for the initialize command x.

## Comments

You can define a maximum of three modem answers e.g. `INITANSWER0=OK` .  
The entries are only valid, if `MODEM=1` is specified.

Default for all answer strings: OK

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER] INITTIMEOUTx=timeout in seconds for wait an the modem's answer

Define the timeout value in seconds for waiting on an answer from the modem.  
A value of 0 means wait forever for the modem answer.

## Comments

Example: `INITTIMEOUT0=2`

The entries are only valid, if MODEM=1 is specified.

Default value: 3 seconds

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER]

#### INITRETRIESx=Retries, if the modem init answer failed

Define the number of retries used when the modem initial answer fails.

## Comments

Example: INITRETRIES0=2

This entry is only valid if MODEM=1 is specified.

Default value: 1

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER]

#### MODEMCTRL=0/1

Allow modem online control by PPP server.

[PPPSERVER]

MODEMCTRL=1

By default, the usage of a modem online control is disabled.

## Related Topics

PPP server [configuration](#) instructions

## PPPSERVER

### [PPPSERVER] CTRLTIME=Seconds

Sets the idle interval time, at which the PPP server executes the configured control commands (see below).

```
[PPPSERVER]  
CTRLTIME=120
```

By default, PPP server idle control time is 60 seconds.  
The PPP server executes the control commands only, if within this interval no PPP data comes in and closes the current PPP connection, if one of the commands failed.

The CTRLTIME must be a smaller value than the IDLETIME of the PPP server (see above)

## Related Topics

PPP server [configuration](#) instructions

## PPPSERVER

### [PPPSERVER] CTRLCMDx=modem online control command

Defines modem command to control, if modem is online or not at the start of the SC12 PPP server and after a modem hang-up following a PPP session.

## Comments

You can define a maximum of three modem commands e.g. CTRLCMD0=+++ .  
The entries are only valid, if MODEMCTRL=1 is specified.

## Related Topics

PPP server [configuration](#) instructions

---

## PPPSERVER

**[PPPSERVER]**  
**CTRLANSWERx=modems answer of ctrl command x**

Defines the expected modem answer x for the online control command x.

### Comments

You can define a maximum of three modem answers e.g. `INITANSWER0=OK` .  
The entries are only valid, if `MODEMCTRL=1` is specified.

### Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

**[PPPSERVER]**  
**CTRLTIMEOUTx=timeout in seconds for wait on the modem's answer**

Define the timeout value in seconds for waiting on an answer from the modem.  
A value of 0 means wait forever for the modem answer.

### Comments

Example: `CTRLTIMEOUT0=2`  
The entries are only valid, if `MODEMCTRL=1` is specified.

Default value: 3 seconds

### Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

**[PPPSERVER]**

## CTRLRETRIESx=Retries, if the modem online control answer failed

Define the number of retries used when the modem control answer fails.

### Comments

Example: CTRLRETRIES0=2

This entry is only valid if MODEMCTRL=1 is specified.

Default value: 1

### Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER]

#### CMDMODE=switch to modem command mode

Define the string, which switches the modem into the command mode.

### Comments

The entries are only valid if MODEM=1 is specified.

Default string for CMDMODE:+++

### Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER]

#### HANGUPDELAY=Time in seconds for switching modem into command mode

Defines the time in seconds for switching modem into command mode for hang-up commands.

### Comments

The entries are only valid if MODEM=1 is specified.

Default time: 2 seconds

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER] HANGUPCMDx=modem command

Define modem commands to hang-up the modem connected to the SC12.

## Comments

You can define a maximum of three modem hang-up commands e.g. HANGUPCMD0=ATH , which will be executed if the PPP connection is closed

## Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER] HANGUPANSWERx=modems answer for hang-up command x

Define the expected modem answer x for the hang-up command x

## Comments

You can define a maximum of three modem answers e.g. HANGUPANSWER0=OK

Default for all answer strings: OK

## Related Topics

[Top of list](#)  
[Index page](#)

---

## PPPSERVER

### [PPPSERVER] HANGUPTIMEOUTx=timeout in seconds for wait on answer from modem

Define the timeout value in seconds used when waiting on the modem's answer.  
A value of 0 means wait forever.

### Comments

Example: `HANGUPTIMEOUT0=2`

Default value: 3 seconds

### Related Topics

PPP server [configuration](#) instructions

[Top of list](#)  
[Index page](#)

---

## PPPSERVER

### [PPPSERVER] HANGUPRETRIESx=Retries, if the modem hang-up answer failed

Define the number of retries used if the modem hang-up answer fails.

### Comments

Example: `HANGUPRETRIES0=2`

Default value: 1 try

### Related Topics

PPP server [configuration](#) instructions

[Top of list](#)  
[Index page](#)

---

---

## PPPSERVER

### [PPPSERVER] CONNECTMSGx=modem message

Define the expected modem message to get connected to a peer modem

### Comments

You can define a maximum of three modem messages e.g. CONNECTMSG0=RING

Defaults:

CONNECTMSG0=RING

CONNECTMSG1=CONNECT

### Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER] CONNECTANSWERx=modem command for incoming connect message x

Define the expected modem answer x for the incoming connect message x

### Comments

You can define a maximum of three modem answers e.g. CONNECTANSWER0=ATA

Defaults: CONNECTANSWER0=ATA

### Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## PPPSERVER

### [PPPSERVER]

## CONNECTTIMEOUTx=timeout seconds for wait on the modem's connect message x

Define the timeout value in seconds used when waiting on the modem connect message.  
A value of 0 means wait forever.

### Comments

Example: `CONNECTTIMEOUT0=0`

Default values:

`CONNECTTIMEOUT0=0`

`CONNECTTIMEOUT1=60`

### Related Topics

PPP server [configuration](#) instructions

[Top of list](#)

[Index page](#)

---

## RAMDRIVE

**[RAMDRIVE]**

**SIZE=size**

Set the size in KByte of the RAM drive C:.  
If defined as 0, no RAM drive is configured.  
Maximum size is 256 Kbyte

### Comments

Default size: 0 (no RAM drive C:)

[Top of list](#)

[Index page](#)

---

## TIMER

**[TIMER]**

**1C=ms**

Set the interval in milliseconds for timer interrupt 0x1C.  
Range: 1 to 32767, Default value=55 ms.

[Top of list](#)

[Index page](#)

---

## TIMER

**[TIMER]**  
**AF=ms**

Set the interval in milliseconds for timer interrupt 0xAF.  
Range: 1 to 32767, Default value=4 ms.

[Top of list](#)

[Index page](#)

---

## FTP

**[FTP]**  
**ENABLE=0/1**

Define if the FTP server should be activated.

## Comments

Use 0 to disable, 1 to enable.  
By default, the FTP server is enabled.

## Related Topics

[BATCHMODE](#) command

[Top of list](#)

[Index page](#)

---

## FTP

**[FTP]**  
**CMDPORT=port**

Set the command port number of the ftp server.  
Default FTP command port: 21  
Example:

```
[FTP]  
CMDPORT=5000
```

[Top of list](#)

[Index page](#)

---

## FTP

**[FTP]**  
**LOGINDELAY=0/1**

Define if the delayed login of the FTP server should be (de)activated.

### Comments

Use 0 to deactivate, 1 to activate.

By default, the delayed login is enabled.

The delay time starts with 400 milliseconds.

After each following failed login the delay time will be doubled until it reached 20 seconds.

After successful login the delay time will be set back to 400 milliseconds.

[Top of list](#)

[Index page](#)

---

## FTP

**[FTP]**  
**TIMEOUT=sec**

Define the inactivity timeout for the FTP server in seconds.

The minimum value for the timeout is 20 seconds, maximum is 65535 seconds.

### Comments

Default is 300 seconds.

RFC 1123 states that the minimum idle timeout should be 5 minutes.

[Top of list](#)

[Index page](#)

---

## FTP

**[FTP]**  
**USERx=user**

Define the user name for FTP

### Comments

You can define a USER0 and a USER1

Default users are: 'anonymous' (no password) and 'ftp' (password is 'ftp').

You must specify both the user name and his password.

Both user name and password are not case sensitive.

Maximum Name Size: 19 characters

Important notice: To avoid security leaks you must define both user names and passwords.

[Top of list](#)

[Index page](#)

---

## FTP

[FTP]

### **PASSWORDx=password**

Define the password for a FTP user

## Comments

You can define a PASSWORD0 for USER0 and a PASSWORD1 for USER1

Default users are anonymous (no password) and ftp (password is 'ftp').

Both user name and password are not case sensitive.

Maximum Password Size: 19 characters

[Top of list](#)

[Index page](#)

---

## FTP

[FTP]

### **ACCESSRIGHTx=Access rights for defined Users**

By using this CHIP.INI entry you can deny the write access for USER0 or USER1.

0 - write and read access enabled

1 - write access denied, read access enabled

Example:

```
[FTP]
USER0=otto
PASSWORD0=otto53pass
ACCESSRIGHT0=1
```

## Comments

You can only forbid the write acces if you have defined a User before by using USERx and PASSWORDx entries.

By default the write access is enabled for both Users.

## FTP

### [FTP] DRIVEx=Set users FTP drive

Set users FTP drive

0: Drive A

1: Drive B

2: Drive C

If the drive does not exist, the default drive A will be set.

## Comments

The following example defines the root drive for USER0

```
[FTP]  
DRIVE0=1
```

## FTP

### [FTP] ROOTDIRx=Name of the users FTP server root directory

Define the name of users FTP server root directory.

## Comments

The following example defines both root directory for USER1

```
[FTP]  
ROOTDIR1=USERDIR
```

If the directory doesn't exist, the FTP server closes the connection.

Important notice: To avoid security leaks you should define one "normal" user with a directory below the "\" directory.

The other user rootdir should not defined, this "superuser" or "admin" is able to access all files of the system.

If ROOTDIRx is set you must also specify DRIVEx (see above).

Maximum ROOTDIRx size: 64 characters

---

## WEB

**[WEB]  
ENABLE=0/1**

Define if the Web server should be activated.

### Comments

Use 0 to disable, 1 to enable.  
By default, the Web server is enabled.

[Top of list](#)

[Index page](#)

---

## WEB

**[WEB]  
MAINPAGE=Name of the main page**

Define the name of Web server's main page. The Web server opens this page if a browser request like `http://192.168.200.4/` is received. Typical names are "main.htm" (default) or "index.htm". The console command [webstat](#) shows the current main page.

### Related Topics

Set Web Server [Main](#) Page API Function

[Top of list](#)

[Index page](#)

---

## WEB

**[WEB]  
TEMPPATH=Name of a temporary Web server path**

Defines a temporary path for finding files.  
If the Web server cannot find the file in its default directory, it will try to find it in this temporary path.  
Pathname should include the drive specification.

### Comments

This function allows the Web server to locate HTML files produced on the SC12 RAMDISK by application programs.

The maximum string length is 32.

Example:

```
[WEB]  
TEMPPATH=C:\web
```

The [webstat](#) command shows the current temporary path.

[Top of list](#)

[Index page](#)

---

## WEB

### [WEB] DRIVE=Set Web server's disk drive

Set Web server's disk drive.

- 0: Drive A
- 1: Drive B
- 2: Drive C

If the drive does not exist, the default drive A will be set.

The console command [webstat](#) shows the current Web server drive.

[Top of list](#)

[Index page](#)

---

## WEB

### [WEB] ROOTDIR=Name of the root directory

Define the name of Web server's root directory. If the directory does not exist, the Web server sets "\" as the default root directory.

The console command [webstat](#) shows the current root directory.

## Comments

Important notice: To avoid security leaks you should define a directory below the "\" directory. If you use "\" as web root directory everybody could read all your files.

## Related Topics

Set Web Server [Root](#) Directory API Function

[Top of list](#)

## WEB

### [WEB]

#### **MAXCGIENTRIES=Maximum number of available CGI entries**

Set the maximum number of entries for the Web server (Default: 10)

Range: 2 to 128

The console command [cgistat](#) shows the current number.

[Top of list](#)

[Index page](#)

---

## WEB

### [WEB]

#### **WEBSERVERSTACK=Stack size**

Set stack size (bytes) for the Web server task. The default and minimal stack size is 2048 Bytes.

Programmers of CGI functions, who are using Microsoft C-Compilers with C-Library functions

e.g. `printf`, which require a lot of stack space, should increase this size to 6144 (6KBytes).

The maximum value is 10240.

[Top of list](#)

[Index page](#)

---

## WEB

### [WEB]

#### **HTTPPORT=port**

Set the port number of the web server.

Default http port: 80

Example:

```
[WEB]
```

```
HTTPPORT=81
```

The console command [webstat](#) shows the current http port number.

[Top of list](#)

[Index page](#)

---

## WEB

## **[WEB] USER0=Username for WebServer Put Methode**

Define the Username to transfer files to the Web server's root directory using the HTTP Put method.  
Standard Username is 'WEB'

The console command [webstat](#) shows the Username and Password.

### **Comments**

Important notice: To avoid security leaks you should define a Username and Password.

[Top of list](#)  
[Index page](#)

---

## **WEB**

### **[WEB] PASSWORD0=Password for WebServer Put Methode**

Define the Password to transfer files to the Web server's root directory using the HTTP Put method.  
Standard Password is 'WEB'

The console command [webstat](#) shows the Password and Username.

### **Comments**

Important notice: To avoid security leaks you should define a Username and Password.

[Top of list](#)  
[Index page](#)

---

## **TFTP**

### **[TFTP] TFTPPORT=port**

Set the port number of the TFTP server.  
Default TFTP port: 5000  
Example:

```
[ TFTP ]  
TFTPPORT=69
```

[Top of list](#)  
[Index page](#)

---

## TELNET

**[TELNET]  
TELNETPORT=port**

Set the port number of the telnet server.

Default telnet port: 23

Example:

```
[ TELNET ]  
TELNETPORT=5000
```

[Top of list](#)

[Index page](#)

---

## TELNET

**[TELNET]  
TIMEOUT=Telnet timeout minutes**

Telnet session will automatically close after TIMEOUT minutes without any characters received from the client. A TIMEOUT setting of zero means no timeout.

Default Value: TIMEOUT=0 (no timeout)

[Top of list](#)

[Index page](#)

---

## TELNET

**[TELNET]  
LOGINDELAY=0/1**

Define if the delayed login of the Telnet server should be (de)activated.

### Comments

Use 0 to deactivate, 1 to activate.

By default, the delayed login is enabled.

The delay time starts with 400 milliseconds.

After each following failed login the delay time will be doubled until it reached 20 seconds.

After successful login the delay time will be set back to 400 milliseconds.

[Top of list](#)

[Index page](#)

---

## TELNET

**[TELNET]  
LOGINRETRIES=number of login retries**

Define the number of login retries until telnet session will be closed.  
Example:

```
[ TELNET ]  
LOGINRETRIES=3
```

### Comments

The default value is 5.

[Top of list](#)

[Index page](#)

---

## TELNET

**[TELNET]  
USERx=user**

Define the user name for TELNET

### Comments

You can define a USER0 and a USER1  
Default user is: 'tel' , password is 'tel'.  
You must specify both the user name and his password.  
Both user name and password are not case sensitive.

Maximum Name Size: 19 characters

Important notice: To avoid security leaks you must define both user names and passwords.

[Top of list](#)

[Index page](#)

---

## TELNET

**[TELNET]  
PASSWORDx=password**

Define the password for a telnet user

### Comments

You can define a PASSWORD0 for USER0 and a PASSWORD1 for USER1

Default user is 'tel' , password is 'tel'.

Both user name and password are not case sensitive.

Maximum Password Size: 19 characters

[Top of list](#)

[Index page](#)

---

## TELNET

**[TELNET]  
ENABLE=0/1**

Define if the Telnet server should be activated.

### Comments

Use 0 to disable, 1 to enable.

By default, the Telnet server is enabled.

[Top of list](#)

[Index page](#)

---

## DEVICE

**[DEVICE]  
FILESHARING=0/1**

Disable/Enable the filesharing. See also [BIOS Ints](#) Function 0x37.

### Comments

0=disable, 1=enable

[Top of list](#)

[Index page](#)

---

## DEVICE

**[DEVICE]  
NAME=name**

Define the name of this device.

## Comments

This name will show up with the 'Chiptool' software when the network is scanned.

Maximum Name Size: 20 characters

[Top of list](#)

[Index page](#)

---

## SERIAL

### [SERIAL] EXT\_DMA=0/1

Disable/enable DMA receive mode (DMA0/Int5) at the EXT port. If DMA receive mode is disabled, the EXT port works with the standard serial interrupt. The recommended mode is the DMA receive mode. It is only necessary to disable the DMA receive mode for the EXT port, if DMA0 is needed by an external device (in the future). In the IRQ receive mode, you may lose characters if the system gets lots of interrupts (e.g. network) or if you are writing to the flash disk (file system calls). See documentation of the [Hardware API](#).

Example which disables DMA receive mode on the EXT port.

```
[SERIAL]
EXT_DMA=0
```

By default, DMA receive mode is enabled.

[Top of list](#)

[Index page](#)

---

## SERIAL

### [SERIAL] COM\_DMA=0/1

Disable/enable DMA receive mode (DMA1/Int6) at the COM port. If DMA transfer is disabled, the COM port works with the standard serial interrupt. The recommended mode is the DMA receive mode. It is only necessary to disable the DMA receive mode for the COM port if DMA1 will be needed by an external device (in the future). In the IRQ receive mode, you may lose characters if the system gets lots of interrupts (e.g. network) or if you are writing to the flash disk (file system calls).

See documentation of the [Hardware API](#).

Example which disables DMA receive mode on the COM port.

```
[SERIAL]
COM_DMA=0
```

By default, DMA receive mode is enabled.

[Top of list](#)  
[Index page](#)

---

## SERIAL

**[SERIAL]  
SEND\_DMA=0/1**

Select the DMA send mode for a serial port.

### Comments

Use 0 to enable the DMA send mode for the EXT port.  
Use 1 to enable the DMA send mode for the COM port.

Example which enables the DMA send mode on the COM port.

```
[SERIAL]  
SEND_DMA=1
```

By default, DMA send is disabled.

Important:

The IPC@Chip SC12 has only two DMA channels. By default both are used for receiving characters from the COM and EXT port. If you want to use the DMA send mode for one serial port (e.g. the EXT port), the second port (e.g. the COM port) works automatically with the standard serial interrupt.

---

## Developer Notes

This mode is still under development, please contact us at our internet newsgroup or direct by email, if you have questions, problems or suggestions.

[Top of list](#)  
[Index page](#)

---

## SERIAL

**[SERIAL]  
EXT\_RECVQUEUE=size**

Set the receive queue size of the EXT port. Minimum size is 1024.  
Maximum size is 10240 byte.

### Comments

Example:

```
[ SERIAL ]
```

```
EXT_RECVQUEUE=2048
```

By default, the receive queue size is 1024 byte.

If it is planned to use PPP server or PPP client the size should set to 4096.

[Top of list](#)

[Index page](#)

---

## SERIAL

**[SERIAL]**

**EXT\_SENDQUEUE=size**

Set the send queue size of the EXT port. Minimum size is 1024.

Maximum size is 10240 byte.

### Comments

Example:

```
[ SERIAL ]
```

```
EXT_SENDQUEUE=2048
```

By default, the send queue size is 1024 byte.

If it is planned to use PPP server or PPP client the size should set to 4096.

[Top of list](#)

[Index page](#)

---

## SERIAL

**[SERIAL]**

**COM\_RECVQUEUE=size**

Set the receive queue size of the COM port. Minimum size is 1024.

Maximum size is 10240 byte.

### Comments

Example:

```
[ SERIAL ]
```

```
COM_RECVQUEUE=2048
```

By default, the receive queue size is 1024 byte.

If it is planned to use PPP server or PPP client the size should set to 4096.

## SERIAL

### [SERIAL] COM\_SENDQUEUE=size

Set the send queue size of the COM port. Minimum size is 1024.  
Maximum size is 10240 byte.

### Comments

Example:

```
[SERIAL]  
COM_SENDQUEUE=2048
```

By default, the send queue size is 1024 byte.  
If it is planned to use PPP server or PPP client the size should set to 4096.

## SERIAL

### [SERIAL] COM\_BAUD=BAUD Rate

Set the BAUD rate of the COM port.

### Comments

Example:

```
[SERIAL]  
COM_BAUD=9600
```

By default, the BAUD rate of the COM port is 19200 (with 8 data bits, no parity, 1 stop bit).

## SERIAL

### [SERIAL] EXT\_BAUD=BAUD Rate

Set the BAUD rate of the EXT port.

## Comments

Example:

```
[SERIAL]  
EXT_BAUD=9600
```

By default, the BAUD rate of the EXT port is 19200 (with 8 data bits, no parity, 1 stop bit).

[Top of list](#)

[Index page](#)

---

## DOSLOADER

### [DOSLOADER] MEMGAP=Paragraphs

Set a memory gap between the loaded DOS programs as a memory reserve.

Some programs compiled with Borland C 5.02 (other compilers??) tries to increase their program memory block at runtime before they e.g. opening a file with Borland C-Library function fopen, because it requires some more memory. The programs calls int21h 0x4A, this happens inside of the e.g. BorlandC-Library fopen function and is not visible for the application programmer. This memory resize call failed, if another program is loaded after the previous one, because now there is no memory space left for increasing the memory size of the previous program. The program returns at fopen with an error The global program variable errno is set to value 8(not enough memory). To prevent this error BIOS 1.02B allows to define the size of a memory gap between two loaded programs. The value must be defined as a number of paragraphs (1 paragraph==16 Bytes, see comment). This strategy could fail in that cases, when programs are terminated and restart again.

## Comments

Example:

```
[DOSLOADER]  
MEMGAP=128
```

By default, MEMGAP is set to 0, Maxvalue is 2048 paragraphs, a larger is automatic reduced to 2048

## Related Topics

Set memory gap [API](#) function

---

## Developer Notes

It is not necessary to set this entry, if the application doesn't show the described error, or if only the last executed DOS program does file access with C-Library call fopen . Only if a C-Library function call sets the C-Lib variable errno to 8, this value should to be defined. We recommend in that case a value of 128 paragraphs (2048 Bytes).

The described problem was noticed, if the BorlandC-Library function fopen (...) was used. Same happens at memory model Large and the usage of CLIB function malloc. Malloc returns a NULL pointer.

[Top of list](#)

[Index page](#)

---

## BATCH

### [BATCH] BATCHMODE=0/1

Sets the batch file execution mode of DOS programs.

BATCHMODE=0 : (Default mode)

The programs listed in the batch file will be **executed concurrently**, starting one after another, without waiting for completion (or going resident) of the predecessor program. The only exceptions are the WAIT and REBOOT commands.

BATCHMODE=1 :

The listed programs will be **executed sequentially**, one at time (similar to DOS). The execution of the successor program will be delayed until the current program either finishes, terminates resident by calling DOS Interrupt **21h Service 0x31** or makes the BIOS Interrupt **0xA0 Service 0x15** batch file wakeup call.

The maximum delay time for execution of the next listed program in the batch file is 15 seconds, unless this limit has been deactivated with the EXECTIMEOUT=0 **configuration** control.

#### Important:

If BATCHMODE=1 take care that every program in your batch file which has a successor program either exits (**int21h 0x4C**) or terminates resident with **int21h 0x31**.

A program which runs forever should call from the main function BIOS Interrupt **0xA0 Service 0x15**, which immediately enables the further batch file sequencing.

## Related Topics

**BATCHMODE** command

Run-time **batch mode** selection API

[Top of list](#)

[Index page](#)

---

## BATCH

### [BATCH] EXECTIMEOUT=0/1

Disable/enable the batchfile DOS program execution delay time limit for BATCHMODE=1. By default the

maximum delay time for execution of the next listed program in a batch file is 15 seconds. If EXECTIMEOUT is set to 0, the successor program in a batch file waits forever if the predecessor program neither finishes nor calls [0xA0 Service 0x15](#) .

## Comments

Example:

```
[ BATCH ]  
EXECTIMEOUT=0
```

By default, EXECTIMEOUT is set to 1

## Related Topics

[BATCHMODE](#) Configuration

[Top of list](#)

[Index page](#)

---

End of document

---

## CHIP.INI Updates - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### CONFIG Updates

The following changes to the system [configuration](#) initialization are in the indicated BIOS revisions.

- New in version 1.02B: [Enable/Disable filesharing](#)
- New in version 1.02B: [Select DMA send mode for a serial port](#)
- New in version 1.02B: [Set Accessrights for FTP Users](#)
- New in version 1.02B: [Set Username and Password for Web Put Method](#)
- New in version 1.02B: [Reserve a memory gap between loaded programs](#)
- New in version 1.01B: [Name of the users FTP server root directory](#)
- New in version 1.01B: [Name of the users FTP server root directory](#)
- New in version 1.01B: [Number of login retries](#)
- New in version 1.01B: [Telnet login delay 0/1](#)
- New in version 1.01B: [FTP login delay 0/1](#)

---

End of document

---

## Command Processor - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### COMMAND

Command processor

Interprets the commands in `autoexec.bat` and those issued at the console.

New in version 1.01B: [Enable non-multiplexed address bus](#)

New in version 1.01B: [Enable PIO pins and show PIO state](#)

New in version 1.00: [Enable/Disable TFTP](#)

New in version 1.00: [Enable/disable address latch enable pin](#)

New in version 1.00: [Display TCPIP memory usage](#)

New in version 0.69: [ICMP echo command \(ping\)](#)

New in version 0.69: [Display list of all available commands](#)

New in version 0.68: [Set batch file execution mode](#)

New in version 0.68: [Restart the Ethernet interface](#)

New in version 0.68: [XMODEM file transfer](#)

New in version 0.68: [Enable chip select](#)

New in version 0.67: [Enable/disable memory optimization](#)

- [DEL filename](#)
- [DIR filename](#)
- [TYPE file](#)
- [COPY file1 file2](#)
- [REN file1 file2](#)
- [MD dir](#)
- [XTRANS](#)
- [MEMOPT 0/1](#)
- [CD dir](#)
- [RD dir](#)
- [CON](#)
- [IW](#)
- [IB](#)
- [OW](#)
- [OB](#)
- [PCS](#)

- [ALE](#)
- [ADR](#)
- [PIO](#)
- [IP address](#)
- [NETMASK mask](#)
- [GATEWAY address](#)
- [DHCP 0/1](#)
- [IPETH](#)
- [TCPIPMEM](#)
- [BATCHMODE](#)
- [FTP 0/1](#)
- [TFTP](#)
- [IPCFG](#)
- [REBOOT](#)
- [WAIT secs](#)
- [FORMAT A: \[/C:n\] \[/E\] \[/R:n\]](#)
- [VER](#)
- [MEM](#)
- [CGISTAT](#)
- [CLOSETELNET](#)
- [WEBSTAT](#)
- [PING](#)
- [TASKS](#)
- [HELP](#)

In the `autoexec.bat` or any other batch file you can only list the internal commands and the names of any program files on the flash drive. A batch file must have the file extension `.bat`, e.g. `test.bat`.

The commands are executed sequentially, with one difference to the 'normal' DOS: When the SC12's **BATCHMODE** is configured for concurrent batch file execution, the next command is executed before a previous command has finished. The only exceptions are the `WAIT` and `REBOOT` commands.

Please note that very little syntax checking is done.

---

## **DEL filename** **Delete a file**

Delete a file or all files that match the wildcard.

## **Example**

```
del *.dat
```

[Top of list](#)

[Index page](#)

---

## **DIR filename** **List a directory.**

List the directory entry or all entries that match the wildcard.

## **Comments**

If no argument is given, \*.\* is assumed.

## **Example**

```
dir *.exe
```

[Top of list](#)  
[Index page](#)

---

## **TYPE file** **Type a file.**

Show contents of a file on the console.

[Top of list](#)  
[Index page](#)

---

## **COPY file1 file2** **Copy a file.**

Copy a file. The two file specifiers must be complete file names.  
Wildcards such as \* or ? are not allowed.

[Top of list](#)  
[Index page](#)

---

## **REN file1 file2** **Rename a file.**

Rename a file. The two file specifiers must be complete file names.  
Wildcards such as \* or ? are not allowed.  
Both files must reside in the same directory.

## **MD dir** **Creates a directory.**

Creates a directory.

### **Example**

```
md temp
```

## **XTRANS** **File transfer: Send/Receive file with Xmodem.**

Send/Receive file with XMODEM/CRC protocol. Possible devices are COM or EXT.

### **Example**

```
XTRANS COM R chip.ini ;Receive chip.ini file over COM  
XTRANS EXT S test.txt ;Send test.txt file over EXT
```

## **MEMOPT 0/1** **Disable or enable memory optimize when loading exe file.**

Use MEMOPT 1 to optimize memory usage when loading an exe file.

By default, the memory optimization is disabled. An exe file will obtain almost all memory available at startup. In the startup code, the program will then resize this memory.

When enabled, the program will obtain only the memory it defines as required in the header of the exe file. This can leave more memory to other programs, but it can result in errors when allocating memory from the heap.

Users of Borland C/C++ will probably not need this command. Only users of Borland Pascal might need it since programs written in Pascal usually do not resize their memory at startup.

### **Comments**

With SC12 BIOS version 0.67, the default for MEMOPT is disabled. Earlier versions had this feature

enabled but this resulted in errors with `malloc()` .

## Example

```
MEMOPT 0
```

[Top of list](#)

[Index page](#)

---

## CD dir Change the current working directory.

Changes the current working directory.

## Example

```
cd temp
```

[Top of list](#)

[Index page](#)

---

## RD dir Removes a directory.

Removes a directory. This command cannot be executed on directories containing data.

## Example

```
rd temp
```

[Top of list](#)

[Index page](#)

---

## CON Direct console I/O.

Define what device is used as the console. Possible devices are COM, EXT or TELNET. Multiple devices are possible.

## Comments

This setting is only valid until the next reboot.

## Example

```
CON COM
CON EXT
CON COM TELNET
```

## Related Topics

Default input device [STDIN](#) initial value

Default output device [STDOUT](#) initial value

[Top of list](#)

[Index page](#)

---

## IW Input word.

Perform a 16 bit input from a given address.  
The address and the result are hexadecimal.

## Example

```
IW 600
```

[Top of list](#)

[Index page](#)

---

## IB Input byte.

Perform an 8 bit input from a given address.  
The address and the result are hexadecimal.

## Example

```
IB 600
```

[Top of list](#)

[Index page](#)

---

## OW Output word.

Perform a 16 bit output on a given address with given data.

The address and the data are hexadecimal. The address is the first parameter followed by data.

## Example

OW 600 F

[Top of list](#)

[Index page](#)

---

## OB Output byte.

Perform an 8 bit output on a given address with given data.  
The address and the data are hexadecimal. The address is the first parameter followed by data.

## Example

OB 600 F

[Top of list](#)

[Index page](#)

---

## PCS Enable chip select.

Enables a chip select line.  
The command expects only one parameter: the chip select line. Valid arguments are 0, 1, 2, 3, 5 or 6.

## Example

PCS 6

[Top of list](#)

[Index page](#)

---

## ALE Enable/disable ALE pin.

Enables the address latch enable (ALE) pin.  
The command expects only one parameter: 1 enable / 0 disable.

## Example

[Top of list](#)  
[Index page](#)

---

## **ADR** **Enable non-multiplexed address bus pins.**

Enables the non-multiplexed address bus pins (A0/A1/A2).  
The command expects only one parameter: 0=enable A0 / 1=enable A1 / 2=enable A2

### **Example**

ADR 0

[Top of list](#)  
[Index page](#)

---

## **PIO** **Enable and show pio pins.**

Enables the programmable pio pins (PIO0-13).  
The command expects two parameters: PIO MODE

PIO: PIO number (0-13)

MODE: PIO mode

- 1 = Input without pullup/pulldown
- 2 = Input with pullup (not PIO13)
- 3 = Input with pulldown (only for PIO3 and PIO13)
- 4 = Output value = High
- 5 = Output value = Low

When no cmdline is given, the pio state is shown.

### **Example**

```
PIO 3 5 = Pio3 Output low  
PIO      = Shows pio states
```

[Top of list](#)  
[Index page](#)

---

## **IP address** **Sets the IP address.**

Sets the IP address of this device.  
Modifies the information stored in A:\chip.ini  
The DHCP option is also switched off.  
The new address is only valid after a restart of the system.

## Comments

Use the [IPCFG](#) command to verify your entry before restarting the system.

## Example

```
IP 195.243.140.85
```

## Related Topics

IP [address](#) initial value  
Set IP Address [API](#) function  
[PPP](#) server initial IP address  
Initial [DHCP](#) setting

[Top of list](#)

[Index page](#)

---

## NETMASK mask Set the network mask for IP addressing.

Sets the subnet mask for IP addressing.  
Modifies the information stored in A:\chip.ini  
The DHCP option is also switched off.  
The new subnet mask is only valid after a restart of the system.

## Comments

Use the [IPCFG](#) command to verify your entry before restarting the system.

## Example

```
NETMASK 255.255.255.192
```

## Related Topics

IP [subnet mask](#) initial value  
Initial [DHCP](#) setting  
Set IP subnet mask [API](#) function

## **GATEWAY address** **Define the IP address of the gateway**

Sets the IP address of the gateway to use.  
Modifies the information stored in A:\chip.ini  
The DHCP option is also switched off.  
The new address is only valid after a restart of the system.

### **Comments**

Use the [IPCFG](#) command to verify your entry before restarting the system.

### **Example**

```
GATEWAY 195.243.140.1
```

### **Related Topics**

IP [GATEWAY](#) initial value  
Set gateway IP address [API](#) function  
[ADD\\_DEFAULT\\_GATEWAY](#) API function  
Initial [DHCP](#) setting

## **DHCP 0/1** **Enable/Disable DHCP.**

Enables or disables the use of DHCP to obtain an IP configuration.

### **Comments**

DHCP Means Dynamic Host Configuration Protocol.  
Using a DHCP Server, the network administrator can define the IP configuration of the network, without manually configuring each device on the network.  
Network servers and some ISDN routers offer a DHCP server.

### **Example**

```
dhcp 1
```

### **Related Topics**

[Top of list](#)  
[Index page](#)

---

## IPETH Restart the Ethernet interface

Restart the Ethernet interface, e.g. after changing the IP configuration, without rebooting the system.

### Comments

If the restart command prints an error message, check your IP parameters. In most cases an invalid gateway IP address is the reason why the restart failed. The errorcode 237 signals that a ethernet configuration was already in progress

[Top of list](#)  
[Index page](#)

---

## TCPIPMEM Display TCPIP memory usage

Display TCPIP memory usage, the command shows the max. reserved memory for the TCPIP stack, and the current used memory.

[Top of list](#)  
[Index page](#)

---

## BATCHMODE Set batch file execution mode

Sets the batch file execution mode of DOS programs for either concurrent or sequential execution. See BATCHMODE initialization [documentation](#) for details.

### Example

```
BATCHMODE 1 ; Selects sequential batch file processing mode
BATCHMODE 0 ; Selects concurrent batch file processing mode
```

### Related Topics

Initial **BATCHMODE** setting  
Run-time **batch mode** selection API

[Top of list](#)  
[Index page](#)

---

## FTP 0/1 Enable/Disable FTP.

Enables or disables the start of the FTP server after a reboot.  
The file `chip.ini` contains the new value for FTP enable to be applied after rebooting the system.

### Example

FTP 1

### Related Topics

Initial **FTP** ENABLE setting

[Top of list](#)  
[Index page](#)

---

## TFTP Enable/Disable TFTP

Enables or disables filetransfers via TFTP server.  
0 disables the server, 1 enables TFTP filetransfer.

### Comments

By default the TFTP server is disabled to avoid security leaks.

### Example

TFTP 1

[Top of list](#)  
[Index page](#)

---

## IPCFG Display IP configuration of the Ethernet interface.

Display IP configuration: DHCP, IP address, network mask, default gateway, serial number and Ethernet address.

The information shown may not be valid until after the system has restarted.

## Comments

During a PPP session (client or server) the command ipcfg shows as the default gateway the defined gateway (if any) from the PPP server section of the chip.ini file. After the PPP session, the old gateway (if any) of the Ethernet interface will be restored.

## Example

```
IPCFG
```

[Top of list](#)

[Index page](#)

---

## REBOOT Restart the system

Restarts the system.

First, the file system is closed, then the watchdog is configured to issue a reset.

Please note that the tasks are not informed of this restart !

## Example

```
reboot
```

[Top of list](#)

[Index page](#)

---

## WAIT secs Suspends the command interpreter.

Suspends execution of the command interpreter for the specified interval.

The time interval is defined in seconds.

## Example

```
wait 1
```

[Top of list](#)

[Index page](#)

---

## FORMAT A: [/C:n] [/E] [/R:n] Format Flash disk A:.

Format flash disk A:.

**All information on drive A: will be lost !**

The cluster size parameter /C: is optional, a value of 2 is default on A:, value 4 on B:.

If the /E parameter is specified, the data area will be filled with null-data.

With parameter /R you can select the number of root directory entries. Note: This must be a multiple of 16.

## Comments

Make sure that other tasks do not access drive A: when formatting.

**Important** : If you use retentive operators, only format flash disk with default cluster size!!

## Example

```
FORMAT A: /C:2 /E
```

```
FORMAT B: /C:4
```

```
FORMAT B: /R:256
```

[Top of list](#)

[Index page](#)

---

## VER BIOS Version.

Output the IPC@CHIP serial number, BIOS version and build date.

[Top of list](#)

[Index page](#)

---

## MEM Display memory map.

Displays a memory map, including the name of the task owning the memory.

## Comments

The size indicated is the actual usable size.

One sector (16 bytes) is added for memory management.

[Top of list](#)

[Index page](#)

---

## **CGISTAT**

### **List Installed CGI handlers.**

This function will list all installed CGI handlers.

## **Related Topics**

[CGI\\_INSTALL](#) API function

[Top of list](#)

[Index page](#)

---

## **CLOSETELNET**

### **Closing current telnet session.**

This function will finish the current telnet session.

[Top of list](#)

[Index page](#)

---

## **WEBSTAT**

### **Show the current settings of the webserver**

This function will show the current settings of the webserver  
e.g. root directory, root drive,...., default start page.

See at config.htm [Webserver config](#) the available chip.ini entries for the Webserver.

[Top of list](#)

[Index page](#)

---

## **PING**

### **The ICMP echo request (ping)**

Test the network connection with the ICMP command ping.

This command sends 4 ICMP echo requests (64 Bytes) to the remote host,  
with an interval of 1 second and shows the results.

## **Example**

```
PING 192.168.200.10<n1>
```

## **Related Topics**

## TASKS

### Display list of tasks.

Displays a lists of all tasks, including the CPU load caused by the task, the task status and the stack space usage.

Sample output:

```
task 1094 count 3515 MTSK prio= 12 stack=3000 used=35% state=0
task 1606 count 81 ETH0 prio= 5 stack=2048 used=41% state=4
task 256 count 4568 AMXK prio= 0
task 2374 count 1072 WEBS prio= 41 stack=2048 used=24% state=81
task 2886 count 100 DOS1 prio= 25 stack=128 used=78% state=81
task 3142 count 157 DOS2 prio= 25 stack=128 used=78% state=81
task 1862 count 24 CFGS prio= 7 stack=1400 used=28% state=4
```

At every one millisecond clock tick, the count for the active task is increased by one. After 10 seconds, the counters are copied and reset to zero.

## Comments

At the first call of TASKS, the timer interrupt routine of the RTOS is exchanged by a version for the task monitor. Only after 10 seconds will the TASKS command return usable results. The command shows for DOS applications only a task stack size of 128 Byte, since the DOS program at run time switches to its own internal stack which is not visible to the Kernel.

A maximum of 35 tasks can be monitored.

Please be aware that using TASKS has a performance penalty. Use UTASKS to shut off the task monitoring.

The printed task state is only a one moment snapshot.

Task states (16Bit hex value):

Bit0	timer wait (used with other bits)
Bit1	trigger wait (i.e. idle)
Bit2	semaphore wait
Bit3	event group wait
Bit4	message exchange wait
Bit5	message send wait
Bit6	suspended (waiting for resume)
Bit7	waiting for wake
Bit7-15	internal use only

Current running system tasks (if not disabled in `chip.ini`)

Very high priority:

AMXK	prio= 00	Kernel task
ETH0	prio= 05	Ethernet receiver task

Normal:

PPPS	prio= 06	PPP server
TCPT	prio= 06	TCPIP timer task
CFGS	prio= 07	UDP config server
TELN	prio= 11	Telnet server

MTSK	prio= 12	Console task (command shell)
Low priority:		
WEBS	prio= 41	Web server
FTPS	prio= 41	FTP server

[Top of list](#)

[Index page](#)

---

## **HELP**

### **Display list of all console commands.**

Displays a list of all available console commands.

[Top of list](#)

[Index page](#)

---

End of document

---

## TCP/IP Application Programmer's Interface - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

TCP/IP API [News](#)

---

### TCP/IP API

Here is the TCP/IP Socket-Interface definition for the IPC@Chip. The TCP/IP API are all reached through interrupt 0xAC. The desired service is selected with the high order byte of the AX register (AH). This interface provides access to the TCP/IP stack of the IPC@Chip for programming TCP/IP applications.

Please note, that we cannot explain the whole functionality of the TCP/IP protocol and the working of the socket interface at this document. Good books for understanding TCP/IP and the socket interface are e.g.:

1. Internetworking with TCP/IP, Volume 1-3 from Douglas E. Comer
2. TCP/IP Illustrated, Volume 1 from W. Richard Stevens

All needed constants and data structures are defined in the header file `tcpipapi.h`. The service selection indexes passed to the TCP/IP API interrupt 0xAC in register AH (e.g. `API_OPENSOCKET`) are defined in this header file.

TCP/IP API [News](#)

TCP/IP API [Error Codes](#) Listing

TCP/IP API [Developer](#) Notes

TCP/IP API [Data Structures](#)

TCP/IP API [Client/Server applications](#)

#### Notes :

- "Network byte order" is big endian (like Motorola machines, unlike Intel).
- At return of most API the DX-Register is used for error checking:  
DX: = 0 = `API_ENOERROR` ==> success  
DX: = -1 = `API_ERROR` ==> error, AX contains error code

#### API Functions :

- [Interrupt 0xAC function 0x01: API\\_OPENSOCKET, open a socket](#)
- [Interrupt 0xAC function 0x02: API\\_CLOSESOCKET, close a socket](#)
- [Interrupt 0xAC function 0x03: API\\_BIND, bind TCP or UDP server socket](#)
- [Interrupt 0xAC function 0x04: API\\_CONNECT, connect to another socket](#)

- Interrupt 0xAC function 0x05: API\_RECVFROM, receive message
- Interrupt 0xAC function 0x06: API\_SENDTO, transmit a datagram
- Interrupt 0xAC function 0x07: API\_HTONS, convert byte order
- Interrupt 0xAC function 0x08: API\_INETADDR, convert an IP-String
- Interrupt 0xAC function 0x09: API\_SLEEP, sleep
- Interrupt 0xAC function 0x0A: API\_MALLOC, alloc a buffer
- Interrupt 0xAC function 0x0B: API\_FREE, free an allocated buffer
- Interrupt 0xAC function 0x0C: API\_GETRCV\_BYTES, get waiting bytes count
- Interrupt 0xAC function 0x0D: API\_ACCEPT, accept the next incoming connection
- Interrupt 0xAC function 0x0E: API\_LISTEN, listening for incoming connections
- Interrupt 0xAC function 0x0F: API\_SEND, transmit a message
- Interrupt 0xAC function 0x10: API\_RECV, receive message
- Interrupt 0xAC function 0x11: API\_INETTOASCII, convert an IP address to an IP string
- Interrupt 0xAC function 0x12: API\_RESETCONNECTION, abort a connection on a socket
- Interrupt 0xAC function 0x13: API\_SETLINGER, set linger time on close
- Interrupt 0xAC function 0x14: API\_SETREUSE, set reuse option on a listening socket
- Interrupt 0xAC function 0x15: API\_SETIPTOS, set IP Type-OF-Service
- Interrupt 0xAC function 0x16: API\_SETSOCKOPT, set options on socket
- Interrupt 0xAC function 0x17: API\_GETSOCKOPT, get options on socket
- Interrupt 0xAC function 0x18: API\_SETBLOCKINGMODE, set socket mode
- Interrupt 0xAC function 0x19: API\_REGISTERCALLBACK, register an user callback function
- Interrupt 0xAC function 0x20: API\_REGISTERCALLBACK\_PASCAL, register a pascal user callback function.
- Interrupt 0xAC function 0x21: API\_GET\_SOCKET\_ERROR, get last socket error.
- Interrupt 0xAC function 0x40: PPPCLIENT\_INSTALLED, check if PPP client installed.
- Interrupt 0xAC function 0x41: PPPCLIENT\_OPEN, open a PPP connection
- Interrupt 0xAC function 0x42: PPPCLIENT\_CLOSE, closing a ppp client connection
- Interrupt 0xAC function 0x43: PPPCLIENT\_GET\_STATUS, get PPP client status
- Interrupt 0xAC function 0x50: PPPSERVER\_INSTALLED, check if PPP server installed
- Interrupt 0xAC function 0x51: PPPSERVER\_SUSPEND, suspend PPP server task
- Interrupt 0xAC function 0x52: PPPSERVER\_ACTIVATE, activate PPP server
- Interrupt 0xAC function 0x53: PPPSERVER\_GET\_STATUS
- Interrupt 0xAC function 0x54: PPPSERVER\_GET\_CFG
- Interrupt 0xAC function 0x60: Get internal TCPIP SNMP variables
- Interrupt 0xAC function 0x65: Get login counters of the FTP server
- Interrupt 0xAC function 0x66: Get login counters of the Telnet server
- Interrupt 0xAC function 0x67: Get the state of the telnet server
- Interrupt 0xAC function 0x70: GET\_INSTALLED\_SERVERS and interfaces
- Interrupt 0xAC function 0x71: RECONFIG\_ETHERNET, Reconfigure Ethernet interface
- Interrupt 0xAC function 0x72: DHCP\_USE, Set/Reset DHCP usage of the Ethernet interface
- Interrupt 0xAC function 0x73: DHCP\_STAT, Get DHCP status of the Ethernet interface
- Interrupt 0xAC function 0x74: TCPIP\_STATISTICS, Get access to the internal network packet counter
- Interrupt 0xAC function 0x75: PING\_OPEN, open and start ICMP echo requests
- Interrupt 0xAC function 0x76: PING\_CLOSE, finish ICMP echo requests
- Interrupt 0xAC function 0x77: PING\_STATISTICS, retrieve ping information
- Interrupt 0xAC function 0x78: GETMEMORY\_INFO, get information about TCP/IP stack memory usage
- Interrupt 0xAC function 0x79: SET\_SERVER\_IDLE\_TIMEOUT

- [Interrupt 0xAC function 0x80: ADD\\_DEFAULT\\_GATEWAY, add the default gateway](#)
  - [Interrupt 0xAC function 0x81: DEL\\_DEFAULT\\_GATEWAY, delete the default gateway](#)
  - [Interrupt 0xAC function 0x82: GET\\_DEFAULT\\_GATEWAY, get the current default gateway](#)
  - [Interrupt 0xAC function 0x83: ADD\\_STATIC\\_ROUTE, add a route for an interface](#)
  - [Interrupt 0xAC function 0x84: DEL\\_STATIC\\_ROUTE, delete a route for an interface](#)
  - [Interrupt 0xAC function 0x90: ADD\\_IGMP\\_MEMBERSHIP, Install an IP multicast address entry](#)
  - [Interrupt 0xAC function 0x91: DROP\\_IGMP\\_MEMBERSHIP, Delete an IP multicast address entry](#)
  - [Interrupt 0xAC function 0x92: MCASTIP\\_TO\\_MACADDR, Map IP multicast address to ethernet address](#)
- 

## Interrupt 0xAC service 0x01: API\_OPEN\_SOCKET, open a socket

Creates an endpoint for communication and returns a socket descriptor (i.e. a handle). This function provides the BSD socket() functionality.

### Parameters

AH

0x01 (= API\_OPEN\_SOCKET)

AL

type of socket :

AL = 1 (= SOCK\_STREAM) ==> TCP

AL = 2 (= SOCK\_DGRAM) ==> UDP

### Return Value

DX=0 success AX: socket descriptor

DX!=0 AX: contains error code

### Related Topics

Close socket [API\\_CLOSE\\_SOCKET](#)

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x02: API\_CLOSE\_SOCKET, close a socket

Closes the socket indicated by BX and releases all of its associated resources.

### Parameters

AH

0x02 (= API\_CLOSESOCKET)

*BX*

Socket descriptor

## Return Value

DX=0 success AX: 0

DX= -1 = API\_ERROR AX: contains error code

## Related Topics

Open socket [API\\_OPENSOCKET](#)

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x03: API\_BIND, bind TCP or UDP server socket

Bind a unnamed socket with an address and port number.

## Parameters

*AH*

0x03 (= API\_BIND)

*BX*

Socket descriptor

*DX:SI*

Pointer to a `sockaddr_in` structure (see `tcpipapi.h`)

## Return Value

DX=0 success AX: 0

DX!=0 AX: contains error code

## Comments

The `sockaddr_in` [structure](#) at [DX:SI] must be filled in by the caller prior to making this API call.

It is only necessary to use the bind call in server applications.

The older TCP and UDP echo client examples also uses a bind call, but this was not necessary.

If you use the bind call in a client application, the client uses the given port number as its own source port address. Otherwise a random 16-bit source port number will be used when no bind call is made.

## Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

### Interrupt 0xAC service 0x04: API\_CONNECT, connect to another socket

TCP only, The connect call attempts to make a connection to another socket (either local or remote). This call is used by a TCP client

#### Parameters

*AH*

0x04 (= API\_CONNECT)

*BX*

Socket descriptor

*DX:SI*

Pointer to a `sockaddr_in` structure containing host's IP address and port number.

#### Return Value

DX=0 success AX: 0

DX!=0 AX: contains error code

#### Comments

The caller must fill in the `sockaddr_in` data [structure](#) at [DX:SI] prior to calling here.

## Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

### Interrupt 0xAC service 0x05: API\_RECVFROM, receive message

UDP only, receive message from another socket

#### Parameters

*AH*  
0x05 (= API\_RECVFROM)

*BX*  
Socket descriptor

*DX:SI*  
Pointer to a `recv_params` data structure (`tcpipapi.h`)

## Return Value

DX=0 success AX: number of received bytes 0 bytes -> timeout  
DX!=0 AX: contains error code

## Comments

The caller must fill in the `recv_params` data **structure** at [DX:SI] prior to calling here. This function will output up to `recv_params.bufferLength` bytes to the buffer referenced by `recv_params.bufferPtr` pointer. The return value indicates the number of byte put into the buffer.

## Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x06: API\_SENDTO, transmit a datagram

UDP only, transmit message to another transport end-point

## Parameters

*AH*  
0x06 (= API\_SENDTO)

*BX*  
Socket descriptor

*DX:SI*  
Pointer to a `send_params` structure (see `tcpipapi.h`)

## Return Value

DX=0 success AX: number of bytes sent  
DX!=0 AX: contains error code

## Comments

The caller must fill in the `send_params` data [structure](#) at [DX:SI] prior to calling here. This function will output up to `send_params.bufferLength` bytes from the buffer at `send_params.bufferPtr` to the IP address specified by the `sockaddr_in` [structure](#) referenced by the `send_params.toPtr` pointer. The return value indicates the actual number of bytes sent.

## Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x07: API\_HTONS, convert byte order

Converts a short (16 bit) value from host byte order to network byte order. This is used to convert port numbers; e.g. `htons(7)`

## Parameters

*AH*  
0x07 (= API\_HTONS)

*BX*  
short value

## Return Value

DX=0, AX contains converted value

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x08: API\_INETADDR, convert an IP-String

Converts a dotted decimal IP address string to an unsigned long.

## Parameters

*AH*  
0x08 (= API\_INETADDR)

*BX:SI*

Pointer to the ip dotted decimal string set by caller.

*ES:DI*

Pointer to a 32 bit unsigned long variable, where this function outputs the converted IP address value.

## Return Value

DX =0 [ES:DI] contains converted value  
DX!=0 syntax error

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x09: API\_SLEEP, sleep

The application sleeps for given number of milliseconds

## Parameters

*AH*

0x09 (= API\_SLEEP)

*BX*

milliseconds

## Return Value

DX=0

## Comments

This call is identical with the RTOS API call [RTX\\_SLEEP\\_TIME](#) . At our earliest BIOS version we had no RTS API, so we provide this call at the TCPIP API. We do not remove this call from the TCPIP API because of compatibility with older BIOS versions.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x0A: API\_MALLOC, alloc a buffer

Memory allocator

## Parameters

*AH*  
0x0A (= API\_MALLOC)

*BX*  
size in bytes

## Return Value

DX =0 ES:DI points to the allocated buffer  
DX!=0 allocation error, ES:DI is a NULL-Pointer

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAC service 0x0B: API\_FREE, free an allocated buffer

Releases a block of allocated memory.

## Parameters

*AH*  
0x0B (= API\_FREE)

*DX:SI*  
Pointer to the buffer

## Return Value

DX =0 success  
DX!=0 free failed

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAC service 0x0C: API\_GETRCV\_BYTES, get waiting bytes count

Get the number of bytes on a socket, waiting for read.

## Parameters

*AH*  
0x0C (= API\_GETRCV\_BYTES)

*BX*  
socket descriptor

## Return Value

DX =0 success, AX contains the number of bytes ready  
DX!=0 failed, AX contains error code

## Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x0D: API\_ACCEPT, accept the next incoming connection

Accept extracts the first connection on the queue of pending connections (from API\_LISTEN) and creates a new socket for this connection. This call is used by a TCP server.

## Parameters

*AH*

0x0D (= API\_ACCEPT)

*BX*

Socket descriptor from API\_LISTEN call

*DX:SI*

Pointer to a `sockaddr_in` [structure](#) (see `tcpipapi.h`)

## Return Value

DX =0 success, AX: contains new socket descriptor for the connection  
DX!=0 failure, AX: contains error code

## Comments

On success, this function fills in the `sockaddr_in` [structure](#) at [DX:SI] with the IP address and port number of the accepted connection.

## Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x0E: API\_LISTEN, listening for incoming connections

Place the socket in passive mode and set the number of incoming TCP connections the system will queue. This call is used by a TCP server.

### Parameters

- AH*  
0x0E (= API\_LISTEN)
- BX*  
Socket descriptor
- CX*  
The maximum number (limited to 5) of allowed outstanding connections

### Return Value

DX =0 success, AX: 0  
DX !=0 failure, AX: contains error code

### Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x0F: API\_SEND, transmit a message

TCP only, transmit message to another transport end-point.  
API\_SEND may be used only if the socket is in a connected state.

### Parameters

- AH*  
0x0F (= API\_SEND)
- BX*  
Socket descriptor
- DX:SI*  
Pointer to a send\_params [structure](#) (see tcpipapi.h)

### Return Value

DX =0 success, AX: number of bytes sent  
DX!=0 failure, AX: contains error code

## Comments

The caller must fill in the `send_params` [structure](#) prior to calling here.  
On success, the return value in AX contains the number of bytes which are successfully inserted into the socket send queue.

## Related Topics

TCP/IP API [Error Codes](#)

---

## Developer Notes

Since BIOS 1.00 the `MSG_DONTWAIT` flag option at the struct `send_params` is enabled for the `API_SEND` and `API_RECV` function. If flag is set to `MSG_DONTWAIT` the send call returns immediatley. As much data as can fit into internal TCP buffer is being sent and the length of the sent data is returned at AX. If none of the data fits then -1 is returned at DX and errorcode 235 at AX.

If flag is set to `MSG_BLOCKING`, the send call blocks until enough internal buffer space is available or socket error occurs. By default the blocking mode is set to all sockets at the open call. If a socket will be set to non blocking with the `API_SETBLOCKINGMODE` function, the `MSG_BLOCKING` flag will not work, the call is still non blocking.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x10: API\_RECV, receive message

TCP only, receive message from another socket.  
API\_RECV may only be used when the socket is in a connected state.

## Parameters

*AH*

0x10 (= API\_RECV)

*BX*

Socket Descriptor

*DX:SI*

Pointer to `recv_params` [structure](#) (see `tcpipapi.h`)

## Return Value

DX =0 success, AX: number of received bytes; 0 bytes -> timeout  
DX!=0 failure, AX: contains error code

## Comments

The caller must fill in the `recv_params` **structure** prior to calling here.

On success, the return value in AX contains the number of bytes which were successfully inserted into the caller's receive buffer at `recv_params.bufferPtr`.

If member `flag` of struct `recv_params` is set to `MSG_DONTWAIT` the `recv` call returns immediately. If no data is available -1 is returned at DX and errorcode 235 at AX. If flag is set to `MSG_BLOCKING`, the `recv` call waits for a message to arrive.

By default the blocking mode is set to all sockets at the open call. If a socket was set to non blocking with the `API_SETBLOCKINGMODE` function, the `MSG_BLOCKING` flag will not work, the call is still non blocking.

## Related Topics

TCP/IP API [Error Codes](#)

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x11: `API_INETTOASCII`, convert an IP address to an IP string

Converts an unsigned long IP address to a dotted decimal IP string

### Parameters

*AH*

0x11 (= `API_INETTOASCII`)

*BX:SI*

Pointer to the 32 bit IP address in network byte order

*ES:DI*

Pointer to the string buffer, where this function can fill in the converted value.  
This buffer must have space for 17 Bytes!

### Return Value

DX=0 [ES:DI] buffer contains converted string value

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x12: API\_RESETCONNECTION, abort a connection on a socket

Works only with TCP sockets.

### Parameters

*AH*  
0x12 (= API\_RESETCONNECTION)

*BX*  
Socket descriptor

### Return Value

DX =0 success  
DX!=0 failure, AX: contains error code

### Comments

Note: API\_RESETCONNECTION doesn't close the socket, this must be done with API\_CLOSESOCKET  
The Reseted Socket can not be used for a new connection. You have to close the socket and open a new one for a different connection.

### Related Topics

TCP/IP API [Error Codes](#)  
Close socket [API\\_CLOSESOCKET](#)

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAC service 0x13: API\_SETLINGER, set linger time on close

Works only with TCP sockets

### Parameters

*AH*  
0x13 (= API\_SETLINGER)

*BX*  
Socket descriptor

*CX*  
Linger time in seconds, default: 60 sec, 0 means linger turned off.

## Return Value

DX =0 success  
DX!=0 failure, AX: contains error code

## Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x14: API\_SETREUSE, set reuse option on a listening socket

Works only with TCP sockets, this is necessary, if a listening socket was closed and will be open and bind on the same port as it was bound to before (see example TCPservm.c)

## Parameters

*AH*

0x14 (= API\_SETREUSE)

*BX*

Socket descriptor

## Return Value

DX =0 success  
DX!=0 failure, AX: contains error code

## Related Topics

TCP/IP API [Error Codes](#)

---

## Developer Notes

Since BIOS 071, we set every TCP socket at the open call as a default reusable, so now it is not for further use of this function.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x15: API\_SETIPTOS, set IP Type-Of-Service

Set socket's default Type-Of-Service put into the IP datagram header's TOS field.

### Parameters

*AH*  
0x15 (= API\_SETIPTOS)

*AL*  
Type-Of-Service for IP datagrams

*BX*  
Socket descriptor

### Return Value

DX =0 success  
DX !=0 failure, AX: contains error code

### Comments

Bits in the IP Type-Of-Service field of a IP datagram:  
Bits 0-2: Datagram precedence, 7 is the highest.  
Bits 3-5: type of transport, see TCP/IP documentation e.g.  
Internet networking with TCP/IP by Douglas E.Comer  
Bits 6-7: unused

Note: Many routers ignore this IP datagram header field.

### Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x16: API\_SETSOCKOPT, set options on socket

Set options on a socket

### Parameters

*AH*  
0x16 (= API\_SETSOCKOPT)

*BX*  
Socket descriptor

*ES:DI*

Pointer to `SetSocketOption` [type](#) that specifies the socket options (see `tcpipapi.h`)

## Return Value

DX=0 success

DX!=0 AX: contains error code

## Comments

This API function makes it possible to manipulate options associated with a socket. Prior to calling this function the caller must fill in a `SetSocketOption` type data structure.

See `SetSocketOption` [type](#) definition for example usage of this API function.

## Related Topics

`GetSocketOption` [typedef](#) with option names

[API\\_GETSOCKOPT](#) - Get socket options

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x17: API\_GETSOCKOPT, get options on socket

Get options on a socket

## Parameters

*AH*

0x17 (= `API_GETSOCKOPT`)

*BX*

Socket descriptor

*ES:DI*

Pointer to `GetSocketOption` [type](#) (see `tcpipapi.h`) data structure.

## Return Value

DX =0 success, Buffer pointed to by the `optionValue` member of `GetSocketOption` type at `[ES:DI]` contains the requested socket option value.

DX!=0 failure, AX: contains error code

## Comments

This API function makes it possible to read options associated with a socket.

Prior to calling this function the caller must fill in a `GetSocketOption` type data structure. The user must set the `protocol_level` and `optionName` members. Also the pointer `optionValue` must point to a valid buffer in the user application's memory. The correct length of the buffer must be specified in `optionLength`.

## Related Topics

`GetSocketOption` [typedef](#) with option names

[API\\_SETSOCKOPT](#) - Set socket options

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x18: API\_SETBLOCKINGMODE, set socket mode

Set a socket in blocking or non-blocking mode

### Parameters

*AH*

0x18 (= API\_SETBLOCKINGMODE)

*AL*

0: switch blocking off, 1:switch blocking on

*BX*

Socket descriptor

### Return Value

DX =0 success

DX!=0 failure, AX: contains error code

## Comments

By default all sockets are in blocking mode. If a socket is set to non blocking mode, socket calls like `CONNECT`, `ACCEPT`, ... do not wait until full completion, they return immediately. Example for usage: The connect call returns at a non blocking socket with -1 at DX and the errorcode 236, if the connection was not completed. The user could call connect at a loop and wait active for completion or exit.

## Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x19: API\_REGISTERCALLBACK, register an user callback function

Register a user callback function for occurring events on a TCP socket  
The TCP/IP stack of the IPC@Chip is able to execute a registered user callback function, if one or more some specified events happens on a TCP socket connection.

### Parameters

<i>AH</i>	0x19 (= API_REGISTERCALLBACK)
<i>BX</i>	Socket descriptor
<i>CX</i>	Event flag mask (see below)
<i>ES:DI</i>	Pointer to callback function (see below)

### Return Value

DX=0 success  
DX!=0 AX: contains error code

### Comments

The callback function must be from the following type (Borlandc):  
`void huge socketCallBackFunc(int socketdescriptor, int eventFlags)`  
Before closing a socket, you should remove the callback function. You must call again this API function with a null pointer at es:di and value 0 at register cx

The parameter eventFlags contains the event(s) that have occurred.

Possible event flags (also defined at TCPIPAPI.H):

```
#define CB_CONNECT_COMPLT 0x0001 //connection complete
```

```
#define CB_ACCEPT 0x0002 //remote has establ. a connection to our listening server
#define CB_RECV 0x0004 //incoming data arrived
#define CB_SEND_COMPLT 0x0010 //sending of data has been acked by the peer
#define CB_REMOTE_CLOSE 0x0020 //peer has shutdown the connection
#define CB_SOCKET_ERROR 0x0040 //an error occured on the connection
#define CB_RESET 0x0080 //peer has sent a rest on the connection
#define CB_CLOSE_COMPLT 0x0100 //close has been completed
```

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAC service 0x20: API\_REGISTERCALLBACK\_PASCAL, register a pascal user callback function.

Register a user callback function written in Pascal for occuring events on a TCP socket  
The count of Pascal TCP callback functions are limited to 10.  
See also API\_REGISTERCALLBACK (0x19)

### Parameters

*AH*  
0x20 (= API\_REGISTERCALLBACK\_PASCAL)

*BX*  
Socket descriptor

*CX*  
Event flag mask (see below)

*ES:DI*  
Pointer to callback function (see below)

### Return Value

DX=0 success  
DX!=0 AX: contains error code

### Comments

The first what you have to do in the callback function is to read the pointer to the [PacCallBack](#) record from ES:DI.

The callback function must be implemeneted as the followong description (Borland Pascal):

```
procedure socketCallBackFunc; interrupt;
var
    ESReg      : Integer;
    DIReg      : Integer;
```

```

    CBParamPtr      : CallbackParamPtr;
begin
    (*****)
    (* Required to get the Parameter *)
    asm
        mov ax, es
        mov ESReg, ax
        mov ax, di
        mov DIReg, ax
    end;
    (*****)

    [... your code ...]

end;

```

Before closing a socket, you should remove the callback function. You must call again this API function with a null pointer at es:di and value 0 at register cx

The parameter eventFlags contains the event(s) that have occurred.

Possible event flags:

```

const
    CB_CONNECT_COMPLT = $0001;
    CB_ACCEPT = $0002;
    CB_RECV = $0004;
    CB_SEND_COMPLT = $0010;
    CB_REMOTE_CLOSE = $0020;
    CB_SOCKET_ERROR = $0040;
    CB_RESET = $0080;
    CB_CLOSE_COMPLT = $0100;

```

[Top of list](#)

[Index page](#)

## Interrupt 0xAC service 0x21: API\_GET\_SOCKET\_ERROR, get last socket error.

Returns the last error which occurred in the socket in register bx.

### Parameters

*AH*  
0x21 (= API\_GET\_SOCKET\_ERROR)

*BX*  
Socket descriptor

## Return Value

DX =0: success AX: contains last socket error code

## Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x40: PPPCLIENT\_INSTALLED, check if PPP client installed.

Tests if PPP client services are available in BIOS version.

## Parameters

*AH*

0x40 (= PPPCLIENT\_INSTALLED)

## Return Value

AX =0: PPP client is not installed

AX !=0: PPP client is installed

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x41: PPPCLIENT\_OPEN, open a PPP connection

Open a PPP connection

## Parameters

*AH*

0x41 (= PPPCLIENT\_OPEN)

*ES:DI*

Pointer to a `PPPClient_Init` [type](#) data structure (declared at `tcpipapi.h`)

## Return Value

DX:0 AX:0, success [ES:DI] contains the needed IP data for further TCP/IP socket communication

DX:-1 AX: contains error code, open failed

## Comments

Refer to the PPPCLIE.C example for how to use this API. Also refer to the PPPClient\_Init [type](#) data structure documentation.

Note: **Only one ppp client connection can be open at a time!!**

## Related Topics

PPP [Client](#) Error Codes

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x42: PPPCLIENT\_CLOSE, closing a ppp client connection

### Parameters

AH

0x42 (= PPPCLIENT\_CLOSE)

### Return Value

DX:0, AX =0: PPP client connection is closed

DX:-1, AX contains error code, Connection closed timed out

## Comments

At the close call the PPP client also executes after closing the PPP session the modem hangup commands from the struct PPPClient\_Init [type](#) defined at the PPPCLIENT\_OPEN call (see above).

## Related Topics

PPP [Client](#) Error Codes

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x43: PPPCLIENT\_GET\_STATUS, get PPP client status

### Parameters

AH

0x43 (= PPPCLIENT\_GET\_STATUS)

## Return Value

AX =-2 (=API\_NOT\_SUPPORTED), DX=-2: PPP client is not installed

AX =-1 (=API\_ERROR), DX= PPP client status

AX >= 0, then AX = PPP client status, DX unchanged

## Related Topics

PPP Client [Status](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x50: PPPSERVER\_INSTALLED, check if PPP server installed

Test if PPP server is installed.

## Parameters

*AH*

0x50 (= PPPSERVER\_INSTALLED)

## Return Value

AX =0: PPP server is not installed

AX !=0: PPP server is installed

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x51: PPPSERVER\_SUSPEND, suspend PPP server task

Suspend the PPP server task

## Parameters

*AH*

0x51 (= PPPSERVER\_SUSPEND)

*BX*

Timeout seconds

## Return Value

AX =0, DX=0 PPP server is suspended

AX!=0, DX!=0 Suspending PPP server failed, PPP server is not installed or timeout

## Comments

Note, that the timeout value at bx depends on your timeout entries for the modemcommands at chip.ini. If this call returns with -1 at AX and DX, the most reason is that the modem commands are not finished after the timeout of bx.

## Related Topics

PPPSERVER [HANGUPTIMEOUTx](#) timeout in seconds for wait on answer from modem

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x52: PPPSERVER\_ACTIVATE, activate PPP server

The PPP server is now able to serve a connection

## Parameters

*AH*

0x52 (= PPPSERVER\_ACTIVATE)

*BX*

Timeout seconds

## Return Value

AX =0, DX=0 PPP server is activated

AX!=0, DX!=0 Activating PPP server failed, PPP server is not installed or timeout

## Comments

Note, that the timeout value at bx depends on your timeout entries for the modemcommands at chip.ini. If this call returns with -1 at AX and DX, the most reason is that the modem commands are not finished after the timeout of bx.

## Related Topics

PPPSERVER [HANGUPTIMEOUTx](#) timeout in seconds for wait on answer from modem

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x53: PPPSERVER\_GET\_STATUS

Get the current state of the PPP server

### Parameters

*AH*  
0x53 (= PPPSERVER\_GET\_STATUS)

### Return Value

DX!=0 PPP server is not installed  
DX=0 AX contains the current PPP server state

### Comments

PPP server states:

- 1 Error state, should not happen
- 01 Server disabled
- 02 Server enabled, waiting for connection
- 03 PPP connection is established
- 04 Server tries to hang up modem
- 05 Server tries to initialize modem

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x54: PPPSERVER\_GET\_CFG

Get the current main configuration data of the PPP server

### Parameters

*AH*  
0x54 (= PPPSERVER\_GET\_CFG)

*ES:DI*  
Pointer to PPP\_IPCfG\_Data **type** data structure which will be output to by this function.

### Return Value

DX!=0 AX!=0 :PPP server is not installed  
DX =0 The user structure PPP\_IPCfG\_Data (at ES:DI) is filled with the PPP server configuration data

### Related Topics

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAC service 0x60: Get internal TCPIP SNMP variables

Accessing the defined MIB structures inside of the BIOS TCPIP stack

### Parameters

AH

0x60 (= API\_SNMP\_GET)

AL

- 1 Get pointer to [IfMib](#) data structure
- 2 Get pointer to [IpMib](#) data structure
- 3 Get pointer to [IcmpMib](#) data structure
- 4 Get pointer to [TcpMib](#) data structure
- 5 Get pointer to [UdpMib](#) data structure
- 6 Get pointer to [atEntry](#) data structure

### Return Value

DX!=0 AX!=0 : BIOS with out internal SNMP mib variables  
DX = AX = 0 ES:DI contains a pointer to the structure

### Comments

Note: These structures are only available at BIOS versions, which contain the SNMP option. A SNMP agent is not part of the BIOS. But if an user is able to implement an agent based on the TCPIP API, he needs access to the internal TCPIP SNMP variables. The SNMP variables are not a part of our current official 6 BIOS versions. You must order direct a BIOS version, which includes this feature.

---

### Developer Notes

The access to struct tag\_atEntry is currently not supported.

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAC service 0x65: Get login counters of the FTP server

Accessing the FTP server login counters

## Parameters

*AH*

0x65 (= API\_FTP\_GET\_LOGIN)

## Return Value

DX!=0 AX!=0 : BIOS doesn't support FTP server

DX = AX = 0

ES:DI contains the address of the 32 Bit (unsigned long) login counter

DS:SI contains the address of the 32 Bit (unsigned long) login fail counter

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x66: Get login counters of the Telnet server

Accessing the Telnet server login counters

## Parameters

*AH*

0x66 (= API\_TELNET\_GET\_LOGIN)

## Return Value

DX!=0 AX!=0 : BIOS doesn't support Telnet server

DX = AX = 0

ES:DI contains the address of the 32 Bit (unsigned long) login counter

DS:SI contains the address of the 32 Bit (unsigned long) login fail counter

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x67: Get the state of the telnet server

Check, if the telnet server handles a active telnet session

## Parameters

*AH*

0x67 (= API\_GET\_TELNET\_STATE)

## Return Value

DX==1 AX==1 : BIOS doesn't support Telnet server  
DX = 0 AX = 1 Telnet session is active  
DX = 0 AX = 0 no telnet session

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x70: GET\_INSTALLED\_SERVERS and interfaces

Get information about running servers and interfaces of the IPC@Chip TCP/IP Stack

### Parameters

AH  
0x70 (= GET\_INSTALLED\_SERVERS)

### Return Value

Bits of AX and DX contains the asked information  
Bit=0 service or device is not available.  
Bit=1 service or device is available.

AX:  
Bit 0: Ethernet device  
Bit 1: PPP server  
Bit 2: PPP client  
Bit 3: Web server  
Bit 4: Telnet server  
Bit 5: FTP server  
Bit 6: TFTP server  
Bit 7: DHCP client

DX:  
Bit 0: SNMP MIB variables support  
Bit 1: UDP Config server  
Bit 2: Ping client

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x71: RECONFIG\_ETHERNET, Reconfigure Ethernet interface

Reconfigure Ethernet interface e.g. after changing the IP configuration

### Parameters

AH

0x71 (= RECONFIG\_ETHERNET)

## Return Value

AX:0 success

else restart failed, should not happen, the errorcode 237 signals that a ethernet interface configuration was already in progress.

## Comments

A new IP configuration set with the prompt **commands** `ip`, `netmask` and `gateway` (or the corresponding BIOS **API** calls) becomes valid after a successful call to this function.

If DHCP is changed from 1 to 0 then a new IP address, subnet mask and gateway should be set with the prompt **commands** `ip`, `netmask` and `gateway` or with the BIOS API interrupt 0xA0 services **0x02** , **0x04** , **0x06** before using this function.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x72: DHCP\_USE, Set/Reset DHCP usage of the Ethernet interface

Set/Reset DHCP usage of the Ethernet interface

## Parameters

AH

0x72 (= DHCP\_USE)

AL

0: DHCP not used, 1:DHCP\_USE

## Return Value

AX:0

## Comments

This entry becomes valid only after rebooting the system or after calling function **0x71** .

If DHCP is changed from 1 to 0 then a new IP address, subnet mask and gateway should be set with the prompt **commands** `ip`, `netmask` and `gateway` or with the BIOS API interrupt 0xA0 services **0x02** , **0x04** , **0x06** .

## Interrupt 0xAC service 0x73: DHCP\_STAT, Get DHCP status of the Ethernet interface

Get DHCP status of the Ethernet interface

### Parameters

AH  
0x73 (= DHCP\_STAT)

### Return Value

AX: 1 System uses DHCP, AX:0 DHCP is not used  
DX: 1 System is configured by a DHCP Server, DX:0 System is not configured

## Interrupt 0xAC service 0x74: TCPIP\_STATISTICS, Get access to the internal network packet counter

This function returns the address of a structure which contains pointers to network packet counters.

### Parameters

AH  
0x74 (= TCPIP\_STATISTICS)

### Return Value

AX: 0, DX:0  
ES:DI contains a pointer to the Packet\_Count [structure](#)

### Comments

Note: The counters `count_all_packets` and `count_all_sended_packets` count only Ethernet packets.  
Other counters also count the packets from and to other devices e.g. local loopback packets and PPP packets.

The user is free to read and/or reset these counters.

### Related Topics

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAC service 0x75: PING\_OPEN, open and start ICMP echo requests

This function opens and starts periodic ICMP echo request (ping) to a given remote host

### Parameters

*AH*  
0x75 (= PING\_OPEN)

*ES:DI*  
Pointer to user's Ping [structure](#)

### Return Value

DX:-1, AX contains error code, Ping open failed  
else  
DX:socket descriptor, AX:0

### Comments

Note: This API will send ICMP packets forever, unless you call PING\_CLOSE.

### Related Topics

Ping structure [type](#) definition  
[PING\\_CLOSE](#) API  
[PING\\_STATISTICS](#) API  
[PING](#) command

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAC service 0x76: PING\_CLOSE, finish ICMP echo requests

This function stops cyclic ICMP echo request (ping)

### Parameters

*AH*  
0x76 (= PING\_CLOSE)

*BX*  
Socket descriptor from call PING\_OPEN

## Return Value

DX:-1, AX -1, Ping close failed, should not happen, only if socket descriptor is invalid  
else  
DX: 0, AX:0: success

## Related Topics

[PING\\_OPEN](#) API

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x77: PING\_STATISTICS, retrieve ping information

The user can retrieve ping information by calling PING\_STATISTICS

## Parameters

*AH*  
0x77 (= PING\_STATISTICS)

*ES:DI*  
Pointer to user's Ping [structure](#) from PING\_OPEN call

## Return Value

DX:-1, AX contains error code, Ping open failed  
else DX:socket descriptor AX:0

## Comments

Structure at [ES:DI] is filled with the ping statistics.

## Related Topics

Ping structure [type](#) definition

[PING\\_OPEN](#) API

[PING\\_CLOSE](#) API

[Top of list](#)

## Interrupt 0xAC service 0x78: GETMEMORY\_INFO, get information about TCP/IP stack memory usage

This function returns the maximum available and the current used memory of the TCP/IP stack

### Parameters

*AH*

0x78 (= GETMEMORY\_INFO)

*ES:DI*

Address of user variable (unsigned long) to storage the max available memory

*DS:SI*

Address of user variable (unsigned long) to storage the current used memory

### Return Value

DX: 0, AX 0 (The described memory sizes are stored at [ES:DI] and [DS:SI])

### Comments

The max. available memory for the TCP/IP stack could be configured at `chip.ini` (see [TCPIPMEM](#) ). The memory required by the TCPIP Stack depends on the number of open sockets, the size and number of transported datapackets. For memory blocks equal or smaller than 4096 bytes the TCP/IP stack inside the SC12 BIOS allocates memory from this block. The TCP/IP stack doesn't release this memory back to the system. It will be internally recycled for further usage. The default size of this memory block is 90 kBytes in the SC12 Large version and 98 kBytes in the BIOS version including PPP. Memory blocks bigger than 4096 bytes are allocated from the SC12 BIOS memory. The TCP/IP stack release these blocks back to the SC12 memory management. If your application requires a lot of memory you should avoid sending and receiving packets larger than 2048 bytes. Larger packets should be split into some smaller ones prior to sending. With BIOSINT 0xA0 0x20 it is possible to install a user errorhandler function, which will be called, if the memory limit is reached.

### Related Topics

0xA00x20 [Install a user fatal error handler](#)

IPC@Chip [TCPIP memory](#) `chip.ini` Configuration

## Interrupt 0xAC service 0x79: SET\_SERVER\_IDLE\_TIMEOUT

Set/Get the Idle timeout of the FTP- and Telnet Server.

### Parameters

*AH*  
0x79 (= SET\_SERVER\_IDLE\_TIMEOUT)

*AL*  
0: Set Timeout, 1: Get Timeout

*BX*  
0: Ftp Server, 1: Telnet Server

*DX*  
Timeout Value

### Return Value

AL=0(Set idle timeout): DX=0, AX=0 success  
AX=DX=-1 Server not provided  
AL=1(Get idle timeout): AX=0, DX contains Timeout Value  
AX=DX=-1 Server not provided

### Comments

The call of this function inserts the timeout in the Chip.Ini, if AL is equal to 0. The timeout will be valid without reboot. If AL is 1, the call returns the idle timeout of the server committed in BX. See also CHIP.INI documentation.

### Related Topics

[Ftp Timeout](#) in `chip.ini` Configuration

[Telnet Timeout](#) in `chip.ini` Configuration

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x80: ADD\_DEFAULT\_GATEWAY, add the default gateway

This function is used to add the default gateway for all interfaces

### Parameters

*AH*

0x80 (= ADD\_DEFAULT\_GATEWAY)

*BX*

the device entry of the gateway, 0: Ethernet, 1: PPP server, 2: PPP client

*ES:DI*

Address of user variable (unsigned long) of the gateway IP

## Return Value

DX: 0, AX 0 success

DX:-1, AX contains error code

## Comments

If this function is used, the gateway entry in the `chip.ini` becomes invalidated, but unchanged.

## Related Topics

[DEL\\_DEFAULT\\_GATEWAY](#) API function

TCP/IP API [Error Codes](#)

IP [Gateway](#) `chip.ini` Configuration

PPP server [Gateway](#) `chip.ini` Configuration

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x81: DEL\_DEFAULT\_GATEWAY, delete the default gateway

This function is used to add the default gateway for all interfaces

## Parameters

*AH*

0x81 (= DEL\_DEFAULT\_GATEWAY)

*ES:DI*

Address of user variable (unsigned long) of the gateway IP

## Return Value

DX: 0, AX 0 success

DX:-1, AX contains error code

## Comments

If this function is used, the gateway entry in the `chip.ini` becomes invalidated, but unchanged.

## Related Topics

[ADD\\_DEFAULT\\_GATEWAY](#) API function

TCP/IP API [Error Codes](#)

IP [Gateway](#) `chip.ini` Configuration

PPP server [Gateway](#) `chip.ini` Configuration

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x82: GET\_DEFAULT\_GATEWAY, get the current default gateway

This function is used to get the default gateway for all interfaces.

### Parameters

*AH*  
0x82 (= GET\_DEFAULT\_GATEWAY)

*ES:DI*  
Address of user variable (unsigned long) for storage the gateway IP

### Return Value

DX: 0, AX 0 success, Location at [ES:DI] contains the gateway IP in network byte order.  
DX:-1, AX contains error code

## Related Topics

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x83: ADD\_STATIC\_ROUTE, add a route for a interface

This function is used to add a route for a interface. It allows packets for a different network to be routed by the interface.

### Parameters

*AH*

0x83 (= ADD\_STATIC\_ROUTE)

*BX*

the device entry of the gateway, 0: Ethernet, 1: PPP server, 2: PPP client

*ES:DI*

Pointer to user `Route_Entry` [structure](#)

## Return Value

DX: 0, AX 0 success

DX:-1, AX contains error code

## Comments

The `Route_Entry` structure is defined in `tcpipapi.h`:

## Related Topics

`Route_Entry` structure [type](#) definition

TCP/IP API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x84: DEL\_STATIC\_ROUTE, delete a route for an interface

This function is used to delete a route for an interface.

## Parameters

*AH*

0x84 (= DEL\_STATIC\_ROUTE)

*ES:DI*

Pointer to user `Route_Entry` [structure](#)

## Return Value

DX: 0, AX 0 success

DX:-1, AX contains error code

## Comments

Only structure members `destIPAddress` and `destNetmask` must be valid at the functions call.

## Related Topics

[Route\\_Entry structure \*\*type\*\* definition](#)

[TCP/IP API \*\*Error Codes\*\*](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x90: ADD\_IGMP\_MEMBERSHIP, Install an IP multicast address entry

Install an IP multicast address entry, becoming a member of an IP multicast group.

IP Multicasting is the internet abstraction of hardware multicasting. It allow transmission of IP datagrams to a group of hosts that build a single multicast group. Membership in an multicast group is dynamic. Hosts may join or leave group at any time. Each multicast group has unique IP multicast address (Class D address). The first four bits of an IP multicast address must match to 1110. IP multicast addresses range from 224.0.0.0 through 239.255.255.255.

For the usage of IP multicasting at an ethernet interface IP multicast addresses must be mapped to ethernet hardware addresses. The ethernet device of the IPC@Chip will be switched into the ethernet multicast mode. At this mode it receives any incoming IP packet with the mapped ethernet multicast address and forwards it to the TCPIP layer. Each IP multicast packet will be send with the mapped ethernet multicast address.

Because of the feature, that a multicast IP packet will be received by any member of a multicast host, sending and receiving of IP Multicasts packets is only usable at UDP sockets (datagram sockets).

After installing a IP multicast address with that API call, the application programmer is able use this address as destination address for sending datagrams. A UDP socket is able to receive datagrams with the specified multicast address.

## Parameters

*AH*  
0x90 (= ADD\_IGMP\_MEMBERSHIP)

*ES:DI*  
Pointer to multicast IP address from type unsigned long

*DS:SI*  
Pointer to a 6 byte array that contains the corresponded ethernet address

## Return Value

DX: 0, AX 0 success

DX:-1, AX contains errorcode, invalid class D address, or no free entry at IGMP table available

## Comments

The max. number of supported IP multicast addresses at the IPC@Chip is limited by 15. Before installing an IP multicast address, you can find out the corresponding ethernet multicast address with [Map IP](#)

## [multicast address to ethernet address](#)

Using multicast addresses is only possible with the ethernet interface. Because of the feature, that a multicast IP packet will be received by any member of a multicast host group, sending and receiving of IP Multicasts packets is only usable at UDP sockets (datagram sockets).

## Related Topics

API function [DROP\\_IGMP\\_MEMBERSHIP](#) - Delete an IP multicast address entry

API function [MCASTIP\\_TO\\_MACADDR](#) - Map IP multicast address to ethernet address

---

## Developer Notes

This implementation is still under development. It does not support multicast routing. Sending and receiving multicast datagrams works only at local network. Please contact us at our internet newsgroup or direct by email, if you have questions, problems or suggestions.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAC service 0x91: DROP\_IGMP\_MEMBERSHIP, Delete an IP multicast address entry

Delete an IP multicast address entry, leaving a multicast host group.

## Parameters

*AH*

0x91 (= ADD\_IGMP\_MEMBERSHIP)

*ES:DI*

Pointer to multicast IP address from type unsigned long.

## Return Value

DX: 0, AX 0 success

DX:-1, AX -1, ip address entry not found

## Comments

## Related Topics

API function [ADD\\_IGMP\\_MEMBERSHIP](#) - Install an IP multicast address entry

API function [MCASTIP\\_TO\\_MACADDR](#) - Map IP multicast address to ethernet address

## Interrupt 0xAC service 0x92: MCASTIP\_TO\_MACADDR, Map IP multicast address to ethernet address

Map an IP multicast address to the corresponding ethernet address

### Parameters

*AH*

0x92 (= MCASTIP\_TO\_MACADDR)

*ES:DI*

Pointer to multicast IP address from type unsigned long

*DS:SI*

Pointer to a 6 byte array to storage the generated ethernet multicast address

### Return Value

DX: 0, AX 0 success, storage at DS:SI contains the generated ethernet address  
DX:-1, AX -1, invalid ip address

### Comments

This API function computes the mac address in the following way: To map an IP multicast address to a corresponding ethernet multicast address place the low-order 23 bits of the IP mulitcast address into the low order 23 bits of the special Ethernet multicast address 01 00 5E 00 00 00 e.g. IP multicast address 224.0.0.1 becomes ethernet address 01 00 5E 00 00 01

### Related Topics

API function [ADD\\_IGMP\\_MEMBERSHIP](#) - Install an IP multicast address entry  
API function [DROP\\_IGMP\\_MEMBERSHIP](#) - Delete an IP multicast address entry

## TCP/IP API Updates - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### TCP/IP API News

The following extension to the TCP/IP [API](#) are available in the indicated BIOS revisions.

- New in version 1.02B: [ADD\\_IGMP\\_MEMBERSHIP, Install an IP multicast address entry](#)
- New in version 1.02B: [DROP\\_IGMP\\_MEMBERSHIP, Delete an IP multicast address entry](#)
- New in version 1.02B: [MCASTIP\\_TO\\_MACADDR, Map IP multicast address to ethernet address](#)
- New in version 1.02B: [SET\\_SERVER\\_IDLE\\_TIMEOUT](#)
- New in version 1.02B: [API\\_GET\\_SOCKET\\_ERROR, returns the last socket error](#)
- New in version 1.02B: [API\\_GET\\_TELNET\\_STATE, Get the state of the telnet server](#)
- New in version 1.02B: [API\\_REGISTER\\_CALLBACK\\_PASCAL](#)
- New in version 1.01B: [Modified: GET\\_INSTALLED\\_SERVERS and interfaces](#)
- New in version 1.01B: [GetSocketOption: More socket options](#)
- New in version 1.01B: [API\\_REGISTER\\_CALLBACK](#)
- New in version 1.01B: [API\\_FTP\\_GET\\_LOGIN](#)
- New in version 1.01B: [API\\_TELNET\\_GET\\_LOGIN](#)
- New in version 1.01B: [API\\_SNMP\\_GET](#)
- New in version 1.00: [API\\_SETBLOCKINGMODE](#)

---

End of document

---

## TCP/IP Error Codes - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### TCP/IP API Error Codes

Network API error codes returned by [API](#) calls (stated here in decimal):

- [PPPclient](#)
- [PPPstatus](#)

201 Operation not permitted  
202 No such file or directory  
203 No such process  
204 Interrupted system call  
205 Input/output error  
206 Device not configured  
209 Bad file descriptor  
210 No child processes  
211 Cannot allocate memory  
213 Permission denied  
214 Bad address  
217 File exists  
219 Operation not supported by device  
220 Not a directory  
221 Is a directory  
222 Invalid argument  
235 Operation would block  
236 Operation now in progress  
237 Operation already in progress  
238 Socket operation on non-socket  
239 Destination address required  
240 Message too long  
241 Protocol wrong type for socket  
242 Protocol not available  
243 Protocol not supported  
244 Socket type not supported  
245 Operation not supported  
246 Protocol family not supported  
247 Address family not supported by protocol family  
248 Address already in use  
249 Can't assign requested address  
250 Network is down  
251 Network is unreachable  
252 Network dropped connection on reset  
253 Software caused connection abort  
254 Connection reset by peer  
255 No buffer space available  
256 Socket is already connected  
257 Socket is not connected  
258 Can't send after socket shutdown  
259 Too many references: can't splice  
260 Operation timed out

261 Connection refused  
264 Host is down  
265 No route to host

-1 socket call failed  
0 no error

---

## PPP client error codes

```
// Possible client error codes
#define PPP_INV_COMPORT -1 // invalid port number or PPP server is active at this port
                                // this errorCode also occurs, if
                                // the pppclient is interrupted while dialing
                                // (e.g. user break by setting the flag modem_break
                                // at the struct pppclient_init
                                // or another modem error

#define PPP_INUSE -2 // client is already active
#define PPP_INV_USER -3 // invalid user or password
#define PPP_OPEN_FAIL -4 // open of the interface failed
#define PPP_INV_DEV -5 // interface was not found
#define PPP_IPCFG_FAIL -6 // got an invalid IP from the peer
#define PPP_CONNECT_FAIL -7 // connection to the peer failed
#define PPP_CLOSETIMEOUT -8 // Closing connection timed out
```

[Top of list](#)

[Index page](#)

---

## PPP client status codes

```
// Possible states of a PPP client connection
#define PPP_NOTAVAIL -1 // Client is not running
#define PPP_LNKDOWN 0 // Link is down
#define PPP_LNKWILLOPEN 1 // open link in progress
#define PPP_LNKUP 2 // Link is established
```

[Top of list](#)

[Index page](#)

TCP/IP [API](#) Listing

---

End of document

---

## TCP/IP Application Developers Note - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

TCP/IP API [News](#)

---

### TCP/IP Applications

---

#### Developer Notes

The given examples should be used and modified by the application programmer.

The application programmer should know how the socket-interface works!!

Some of the programs are using `int86x` calls with the CPU registers loaded as described. We also provide a C-Library (`tcpip.c`), which places C wrapper functions around the software interrupt calls.

All program examples built with C-API-functions use the files `tcpip.c`, `tcpip.h` and `tcpipapi.h`. The SC12 Beta version contains revised TCP/IP API calls, so you should always use the current TCP/IP API C and H files (`tcpipapi.h`, `tcpip.c`, and `tcpip.h`).

These files contain all available API calls.

Available examples:

1. UDPEchoClient, `udpclie.c`, built with `int86x` calls
2. UDPEchoServer, `udpserv.c`, built with `int86x` calls
3. TCPEchoClient, `tcpclie.c`, built with `int86x` calls
4. TCPEchoServer, `tcpserc.c`, built with `int86x` calls
5. TCPEchoClient, `tcpclie.c`, built with C-API-functions, using `tcpip.c`
6. TCPEchoServer, `tcpserc.c`, built with C-API-functions, using `tcpip.c`
7. Reconfigure Ethernet interface, `cfgip.c`
8. TCP/IP recv packet counting, `pkt_cnt.c`
9. PPP server API test, `ppps.c`
10. PPP client example, `pppclie.c`

TCP/IP [API](#)

---

End of document

---

## Data Structures used in TCP/IP API - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### Data Structures

Here are the BSD structures and own datatypes used by the [TCP/IP API](#) .  
All constants and data structures are defined in the header file tcpipapi.h

Notes:

1. **Byte alignment is required** within all data structures used within the API.
2. The phrase "network byte order" means big endian (like Motorola machines, unlike Intel).

Content :

- [typedefGetSocketOption](#)
- [structin\\_addr](#)
- [typedefPacket\\_Count](#)
- [typedefPing](#)
- [typedefPPPClient\\_Init](#)
- [typedefPPPDial](#)
- [typedefPPP\\_ModemHangup](#)
- [typedefPPP\\_IPCfg\\_Data](#)
- [structrecv\\_params](#)
- [typedefRoute\\_Entry](#)
- [typedefSetSocketOption](#)
- [typedefPasCallBack](#)
- [structsend\\_params](#)
- [structsockaddr](#)
- [structsockaddr\\_in](#)
- [structtagIfMib](#)
- [structtagIpMib](#)
- [structtagIcmpMib](#)
- [structtagTcpMib](#)
- [structtagUdpMib](#)
- [structtag\\_atEntry](#)

---

### GetSocketOption

```
typedef struct tag_getsockopt
{
    int         protocol_level; // protocol level, ip level, tcp level or socket level
    int         optionName;     // option's name
    char        *optionValue;   // pointer to the option value buffer (type varies)
    int         *optionLength;  // length of option value buffer
}
```

```
}GetSocketOption;
```

## Comments

The type of data found at the location referenced by the `optionValue` member varies depending on the particular socket option that is being dealt with.

When manipulating socket options, the `protocol_level` at which the option resides and the name of the option (`optionName` member) must be specified. The size of the buffer required pointed to by the `optionValue` member depends on the option. These sizes are stated in the list below. The parameters `optionValue` and `optionLength` are used to access option values.

The following list shows the three different protocol levels and their valid socket options, including a short description and the length of the option value. Protocol level and `optionName` constants referred to here are defined in `tcpipapi.h`

1. `protocol_level=IP_PROTOIP_LEVEL` (IP level options)
  - `optionName=IPO_TOS` : Size: 8 bit - Set IP type of service, default 0
  - `optionName=IPO_TTL` : Size: 8 bit - Set IP time-to-live seconds, default 64
2. `protocol_level=IP_PROTOTCP_LEVEL` (TCP level options)
  - `optionName=TCP_NODELAY` : Size: 16 bit - 1:Disable nagle algorithm, default is 0: nagle algorithm is enabled
  - `optionName=TCP_NOPUSH` : Size: 16 bit -1: Force TCP to delay sending any TCP data until a full sized segment is buffered in the TCP buffers, useful, if continuous large amount of data is sent like FTP, default is 0
  - `optionName=TCP_SLOW_START` : Size: 16 bit -1: Enable the TCP slow start algorithm(default), 0:disable
  - `optionName=TCP_KEEPALIVE` : Size: 16 bit - Set idle time seconds before sending keep alive probes, default 7200 seconds.  
Notes:
    - The socket level option `SO_KEEPALIVE`(see below) must be enabled, default 7200 seconds.
    - The minimum value was changed in BIOS070 to 10 seconds (was 7200 seconds in BIOS069).
  - `optionName=TCP_DELAY_ACK` : Size: 16 bit - Set the TCP delay ACK time in milliseconds, default:200 milliseconds
  - `optionName=TCP_FINWT2TIME` : Size: 16 bit - Set the max. amount of time TCP will wait for the remote side to close, default 600 seconds
  - `optionName=TCP_2MSLTIME` : Size: 16 bit - Set the max. amount of time TCP will wait in the TIME WAIT state, once it has initiated a close,default 2 seconds
  - `optionName=TCP_KEEPALIVE_INTV` : Size: 16 bit - Set keep alive interval probes, default 75 seconds  
Note:
    - This value can only be modified after a listen call, before a connection is established.
  - `optionName=TCP_KEEPALIVE_CNT` : Size: 16 bit - Set maximum number of keep alive probes before TCP gives up and closes the connection, default: 12
3. `protocol_level=SOCKET_LEVEL`
  - `optionName=SO_REUSEADDR` : Size: 16 bit - Enable/disable local address reuse, coding: 1=enable, 0=disable default: enable
  - `optionName=SO_KEEPALIVE` : Size: 16 bit - Keep connections alive, coding: 1=enable, 0=disable default: disable
  - `optionName=SO_SNDBUF` : Size: 32 bits - Socket send buffer size, default TCP 4096, UDP 2048 bytes. (We recommend a maximum size of 8192 Bytes.)
  - `optionName=SO_RCVBUF` : Size: 32 bits - Socket input buffer size, default TCP 4096, UDP 2048 bytes. (We recommend at maximum size of 8192 Bytes.)

## Related Topics

API function [API\\_GETSOCKOPT](#) - Get socket options

[SetSocketOption](#) typedef

[Top of list](#)

[Index page](#)

```
struct in_addr
{
    u_long s_addr; // 32 bit netid/hostid address in network byte order
};
```

[Top of list](#)  
[Index page](#)

---

## Packet\_Count

```
typedef struct tag_cnt_packet
{
    unsigned int * cnt_all_packets;           // count all incoming Ethernet packets
    unsigned int * cnt_ip_packets;          // count incoming IP packets
    unsigned int * cnt_arp_packets;         // count incoming ARP packets
    unsigned int * cnt_tcp_packets;        // count incoming TCP packets
    unsigned int * cnt_udp_packets;        // count incoming UDP packets
    unsigned int * cnt_icmp_packets;       // count incoming ICMP packets

    unsigned int * cnt_all_sended_packets;  // count all sent Ethernet packets
    unsigned int * cnt_ip_sended_packets;   // count all sent IP packets
    unsigned int * cnt_arp_sended_packets;  // count all sent ARP packets
    unsigned int * cnt_tcp_sended_packets;  // count all sent TCP packets
    unsigned int * cnt_udp_sended_packets;  // count all sent UDP packets
    unsigned int * cnt_icmp_sended_packets; // count all sent ICMP packets

    unsigned int * cnt_ip_chksum_errs;     // count all checksum errors on incoming IP packets
    unsigned int * cnt_udp_chksum_errs;    // count all checksum errors on incoming UDP packets
    unsigned int * cnt_tcp_chksum_errs;    // count all checksum errors on incoming TCP packets
    unsigned int * cnt_eth_errs;          // count all errors on incoming Ethernet packets
} Packet_Count;
```

## Comments

Note: The counters `count_all_packets` and `count_all_sended_packets` count only Ethernet packets. Other counters also count the packets from and to other devices e.g. local loopback packets and PPP packets.

## Related Topics

API function [TCPIP\\_STATISTICS](#) - Open PPP client session.

[Top of list](#)  
[Index page](#)

---

## Ping

```
typedef struct tag_ping_command
{
    int sd; // socket descriptor, set by PING_OPEN
    // User must set following four values prior to PING_OPEN API call
    char * remoteHostNamePtr; // Remote IP
    int pingInterval; // seconds
    int pingDataLength; // maximum 1024 bytes
    unsigned long count; // number of pings to send

    unsigned char pingstate; // pin socket state, 1: open 0: closed
    //statistics, filled by ping command
    unsigned long transmitted; // sent ping requests
```

```

unsigned long received; // received replies
unsigned int lastsenderr; // last send error
unsigned int lastrcvrr; // last receive error
unsigned long maxRtt; // Maximum round trip time, rounded off to 100 ms step, in
milliseconds
unsigned long minRtt; // Minimum round trip time in milliseconds, (100 ms steps)
unsigned long lastRtt; // round trip time (100 ms steps) of the last ping request/reply
} Ping;

```

## Comments

Caller to PING\_OPEN API must initialize structure members:

```

remoteHostNamePtr... who to "ping"
pingInterval... block repetition rate
pingDataLength... size of ping data blocks
dummy... simply zero this value

```

The remainder of the data structure is managed within the API functions.

## Related Topics

API function [PING\\_OPEN](#) - Start ICMP echo requests

[Top of list](#)

[Index page](#)

---

## PPPClient\_Init

```

typedef struct tag_ppp_client
{
    int port; // serial port (0:EXT 1:COM)
    int auth; // 0: no authentication
              // 1:PAP Client must send username and password for PAP
authentication to the peer
              // 2:CHAP Client must send username and password for CHAP
authentication to the peer
              // 3:PAP Client expects PAP username and password from the peer
              // 4:CHAP Client expects CHAP username and password from the peer

    int modem; // modem usage (0:nullmodem 1:modem)
    int flow; // serial flow control (0: none, 1:XON/XOFF, 2:RTS/CTS)
    long baud; // Serial port BAUD rate
    unsigned long idletimeout; // closing PPP after idle time seconds (0: no closing after idle
time)
    char username[50]; // Used if .auth != 0
    char password[50]; // Used if .auth != 0
    void * dptr; // dummy ptr

    char PPPClieipAddrStr[16]; //If IP is set to "0.0.0.0" client expect IP form the peer, IP is
filled in after sucessful connection
                          //If IP is set to a string != "0.0.0.0" client wants to use this
IP during the ppp session

    char PPPClieRemipAddrStr[16]; //If RemoteIP is set to "0.0.0.0" client allow the peer to use its
own IP during the PPP session,
                          the RemoteIP is filled in after sucessful connection

                          //If RemoteIP is set to a string != "0.0.0.0" client wants to
configure the remote peer with this IP

    char PPPClieNetMaskStr[16]; // subnet mask

```

```

char PPPClieipGatewayStr[16]; // gateway

PPPDial                pppdial[PPPCLIE_MAX_DIAL]; // modem/dial entries
PPP_ModemHangup        modem_hangup;             // modem hang-up commands
int                     break_modem;             // Flag for breaking SC12 - modem
                    // control communication (dialing, waiting for connect)
                    // Setting break_modem to 1, breaks current modem control communication
                    // between SC12 and the modem at a PPP client open or close call,
                    // The PPP client reads this flag and breaks the dialing, if flag is set.
                    // This flag could be set from another task. It will not break an established
                    // PPP link! Don't forget to clear this flag to zero after breaking.
} PPPClient_Init;

```

## Comments

The PPPClient\_Init structure is used to open a PPP client session.

The flow control mode XON/XOFF is not tested. It is not advisable to use it.

Since BIOS 1.02B XON/XOFF mode is also available if the **DMA** mode for the selected serial port is enabled but because of the internal functionality of DMA it is not possible to detect an XON or XOFF of the peer immediately. It is possible that an overrun situation at the connected peer (e.g. GSM modem) could occur. We enable this mode now because GSM modems (any??) supports only XON/XOFF flow ctrl.

The PPPClient\_Init structure contains an array of the PPPdial **structures** used for initializing and dialing a modem. These modem commands will be executed at the start of establishing a connection to a PPP server.

The PPPClient\_Init structure also contains a PPP\_ModemHangup **structures** for closing the modem connection

For how to initialize and use these structures see the PPPCLIE.C example.

## Related Topics

API function **PPPCLIENT\_OPEN** - Open PPP client session.

**PPPDial** dial-up command data

[Top of list](#)

[Index page](#)

## PPPDial

```

typedef struct tag_pppdial_init
{
    char * modemcmd;           // modem command string
    char * modemans;          // modem answer string
    int timeout;              // seconds, 0 = no time out
    int retries;              // Maximum number of dial attempts.
    char expect_send;         // = 0: PPP client sends modem AT command
                                // and expects modem answer (e.g. OK).
                                // = 1: PPP client expects modem answer
                                // (e.g. CONNECT) and sends modem command.
} PPPDial;

```

## Comments

The PPPDial structure is used during PPP client dial-up.

## Related Topics

API function **PPPCLIENT\_OPEN** - Open PPP client session

**PPPClient\_Init** PPP client open data structure

## PPP\_ModemHangup

```
typedef struct tag_pppcli_hangup
{
    char    *modemcmdmode;    // string for switching modem into command mode e.g. +++
    int     delay;           // delay time after switching in seconds
    PPPDial pppdial[PPPCLIE_MAX_DIAL]; // modem commands and answer for hang-up procedure
} PPP_ModemHangup;
```

## Comments

The PPP\_ModemHangup structure is used when closing a PPP modem connection.

## Related Topics

API function [PPPCLIENT\\_OPEN](#) - Open PPP client session  
[PPPClient\\_Init](#) PPP client open data structure

## PPP\_IPCfgr\_Data

```
typedef struct tag_pppipcfg_data
{
    char        IP[16];        // PPP server IP
    char        RemIP[16];    // Remote IP (given to the client, if connected)
    char        Netmask[16];   // Subnet mask
    char        Gateway[16];   // Gateway
    unsigned int comport;      // COM port: EXT=0, COM=1
    unsigned int papauth;     // 0: no authentication 1:PAP 2:CHAP
    unsigned int modem;       // Analog Modem=1, Null Modem cable=0
    unsigned int flow;        // Flow control
    long        baud;         // BAUD rate
} PPP_IPCfgr_Data;
```

## Comments

The PPP\_IPCfgr\_Data structure is used to read out the configuration of the PPP server.

## Related Topics

API function [PPPSERVER\\_GET\\_CFG](#) - Get PPP server configuration

```

struct recv_params
{
    char            *bufferPtr;           // Store incoming data here
    int             bufferLength;        // Maximum bytes to store
    int             flags;                // Blocking, timeout or no wait
    struct sockaddr *fromPtr;            // only needed for UDP
    int             *fromlengthPtr;      // only needed for UDP
    unsigned long   timeout;             // timeout milliseconds
};

```

## Comments

The [sockaddr](#) structure referenced by the `fromPtr` member is cast to a `sockaddr_in` [structure](#) prior to usage.

The integer referenced by the `fromlengthPtr` pointer should be set to `sizeof(struct sockaddr_in)`.

Flags:

MSG\_BLOCKING: Sleeping until data comes in

MSG\_TIMEOUT : The caller wakes up from `recv` after timeout or if data comes in, struct member `timeout` must be filled

MSG\_DONTWAIT: Return immediately, if nothing received

[Top of list](#)

[Index page](#)

---

## Route\_Entry

```

typedef struct tag_route_entry{
    unsigned long destIPAddress; // The IP address to add the route for
    unsigned long destNetmask;   // The netmask for the route
    unsigned long gateway;      // IP address of the gateway of the route
    int hops;                    // Number of routers between this host and route
} Route_Entry;

```

## Comments

[Top of list](#)

[Index page](#)

---

## SetSocketOption

```

typedef struct tag_setsockopt
{
    int     protocol_level; // protocol level, ip level,tcp level or socket level
    int     optionName;     // option's name
    const char *optionValue; // pointer to the option value (type varies)
    int     optionLength;  // length of option value data
}SetSocketOption;

```

```

// Example usage of API_SETSOCKOPT and SetSocketOption type to set
// IP Time-to-Live to 69 seconds:

```

```

union REGS inregs, outregs;
struct SREGS segregs;
unsigned char time_to_live = 69;
int socketdescriptor;
SetSocketOption sockopt = {IP_PROTOIP_LEVEL,           // .protocol_level
                           IPO_TTL,                   // .optionName
                           (const char*)&time_to_live, // .optionValue
                           sizeof(unsigned char)};     // .optionLength

```

```
inregs.h.ah = API_SETSOCKOPT;
inregs.x.bx = socketdescriptor;
segregs.h.es = FP_SEG(&sockopt);           // Fill in ES:DI with a pointer to sockopt
inregs.x.di = FP_OFF(&sockopt);
int86x(TCPIPVECT, &inregs, &outregs, &segregs); // Call API interrupt 0xAC function
API_SETSOCKOPT
```

## Comments

The type of data found at the location referenced by the `optionValue` member varies depending on the particular socket option that is being dealt with.

## Related Topics

For list of options and sizes see [GetSocketOption](#)  
API function [API\\_SETSOCKOPT](#) - Set socket options

[Top of list](#)

[Index page](#)

---

## PasCallback

```
typedef struct tag_PasCallback
{
    int         sd;           // socket descriptor
    int         event;       // occurred event
}PasCallback;
```

## Comments

The Pointer to this struct will be committed to a Pascal callback function in the registers ES:DI. You can read out the information about the socket and the event which has invoked the callback function.

## Related Topics

To register a pascal callback function see [API\\_REGISTER\\_CALLBACK\\_PASCAL](#)

[Top of list](#)

[Index page](#)

---

## send\_params

```
struct send_params
{
    char          *bufferPtr;           // Pointer to send data
    int           bufferLength;         // Number of bytes to send
    int           flags;                // Blocking or no wait
    struct sockaddr *toPtr;             // only needed for UDP
    int           *tolengthPtr;         // only needed for UDP
};
```

## Comments

The [sockaddr](#) structure referenced by the `toPtr` member is cast to a `sockaddr_in` [structure](#) prior to usage.  
The integer referenced by the `tolengthPtr` pointer should be set to `sizeof(struct sockaddr_in)`.

Flags:  
MSG\_BLOCKING: Sleeping until data comes in  
MSG\_DONTWAIT: Return immediately, if nothing received

[Top of list](#)  
[Index page](#)

---

## sockaddr

```
struct sockaddr
{
    u_char  sa_len;        // Total Length
    u_char  sa_family;    // Address Family AF_XXX
    char    sa_data[14]; // up to 14 bytes of protocol specific address
};
```

## Comments

This generic "one size fits all" BSD structure is treated as a `sockaddr_in` [structure](#) within the TCP/IP API functions.

[Top of list](#)  
[Index page](#)

---

## sockaddr\_in

```
struct sockaddr_in
{
    short      sin_family; // AF_INET
    u_short    sin_port;   // 16bit Port Number in network byte order
    struct in_addr sin_addr; // 32bit netid/hostid in network byte order
    char       sin_zero[8]; // unused
};
```

## Comments

The `sin_family` member should be set to `AF_INET` (=2).  
The `sin_addr` member's `in_addr` [structure](#) is simply a long IP address in big endian byte order.

[Top of list](#)  
[Index page](#)

---

## tagIfMib

```
typedef struct tagIfMib //interface, only ethernet is supported
{
    long      ifIndex; // index of this interface 1 for ethernet, currently only
ethernet
    char      ifDescr[32]; // description of interface
    long      ifType; // network device type
    long      ifMtu; // maximum transfer unit
    unsigned long ifSpeed; // bandwidth in bits/sec
    unsigned char ifPhysAddress[11]; // interface's address
    unsigned char PhysAddrLen; // length of physAddr: 6
    long      ifAdminStatus; // desired state of interface, not supported
    long      ifOperStatus; // current operational status, not supported
};
```

```

//counters
unsigned long devLastChange; // value of sysUpTime when current state entered
unsigned long devInOctets; // number of octets received on interface
unsigned long devInUcastPkts; // number of unicast packets delivered
unsigned long devInMulticastPkts; // number of multicast packets delivered, not supported
unsigned long devInBroadcastPkts; //broadcasts delivered
unsigned long devInDiscards; // number of broadcasts
unsigned long devInErrors; // number of packets containing errors
unsigned long devInUnknownProtos; // number of packets with unknown protocol
unsigned long devOutOctets; // number of octets transmitted
unsigned long devOutUcastPkts; // number of unicast packets sent
unsigned long devOutMulticastPkts; // number of multicast packets sent
unsigned long devOutBroadcastPkts; //broadcasts sent
unsigned long devOutDiscards; // number of packets discarded with no error
unsigned long devOutErrors; // number of pkts discarded with an error
unsigned long devOutQLen; // number of packets in output queue
}IfMib;

```

## Comments

Note: These structures are only available at BIOS versions, which contain the SNMP option. A SNMP agent is not part of the BIOS. But if an user is able to implement an agent based on the TCPIP API, he needs access to the internal TCPIP SNMP variables. The SNMP variables are not a part of our current official 6 BIOS versions. You must order direct a BIOS version, which includes this feature.

## Related Topics

API function [API SNMP\\_GET](#) - Get internal TCPIP SNMP variables

[Top of list](#)

[Index page](#)

## tagIpMib

```

typedef struct tagIpMib
{
    long ipForwarding; // 1
    long ipDefaultTTL; // default TTL for pkts originating here
    unsigned long ipInReceives; // no. of IP packets received from interfaces
    unsigned long ipInHdrErrors; // number of pkts discarded due to header errors
    unsigned long ipInAddrErrors; // no. of pkts discarded due to bad address
    unsigned long ipForwDatagrams; // number pf pkts forwarded through this entity
    unsigned long ipInUnknownProtos; // no. of local-addressed pkts w/unknown proto
    unsigned long ipInDiscards; // number of error-free packets discarded
    unsigned long ipInDelivers; // number of datagrams delivered to upper level
    unsigned long ipOutRequests; // number of IP datagrams originating locally
    unsigned long ipOutDiscards; // number of error-free output IP pkts discarded
    unsigned long ipOutNoRoutes; // number of IP pkts discarded due to no route
    long ipReasmTimeout; // seconds fragment is held awaiting reassembly
    unsigned long ipReasmReqds; // no. of fragments needing reassembly (here)
    unsigned long ipReasmOKs; // number of fragments reassembled
    unsigned long ipReasmFails; // number of failures in IP reassembly
    unsigned long ipFragOKs; // number of datagrams fragmented here
    unsigned long ipFragFails; // no. pkts unable to be fragmented here
    unsigned long ipFragCreates; // number of IP fragments created here
    unsigned long ipRoutingDiscards;
} IpMib;

```

## Comments

Note: These structures are only available at BIOS versions, which contain the SNMP option. A SNMP agent is not part of the BIOS. But if an user is able to implement an agent based on the TCPIP API, he needs access to these internal TCPIP variables

## Related Topics

API function [API SNMP\\_GET](#) - Get internal TCPIP SNMP variables

[Top of list](#)

[Index page](#)

---

## tagIcmpMib

```
typedef struct tagIcmpMib
{
    unsigned long    icmpInMsgs;           // Total of ICMP msgs received
    unsigned long    icmpInErrors;        // Total of ICMP msgs received with errors
    unsigned long    icmpInDestUnreachs;
    unsigned long    icmpInTimeExcds;
    unsigned long    icmpInParmProbs;
    unsigned long    icmpInSrcQuenchs;
    unsigned long    icmpInRedirects;
    unsigned long    icmpInEchos;
    unsigned long    icmpInEchoReps;
    unsigned long    icmpInTimestamps;
    unsigned long    icmpInTimestampReps;
    unsigned long    icmpInAddrMasks;
    unsigned long    icmpInAddrMaskReps;
    unsigned long    icmpOutMsgs;
    unsigned long    icmpOutErrors;
    unsigned long    icmpOutDestUnreachs;
    unsigned long    icmpOutTimeExcds;
    unsigned long    icmpOutParmProbs;
    unsigned long    icmpOutSrcQuenchs;
    unsigned long    icmpOutRedirects;
    unsigned long    icmpOutEchos;
    unsigned long    icmpOutEchoReps;
    unsigned long    icmpOutTimestamps;
    unsigned long    icmpOutTimestampReps;
    unsigned long    icmpOutAddrMasks;
    unsigned long    icmpOutAddrMaskReps;
} IcmpMib;
```

## Comments

Note: These structures are only available at BIOS versions, which contain the SNMP option. A SNMP agent is not part of the BIOS. But if an user is able to implement an agent based on the TCPIP API, he needs access to these internal TCPIP variables

## Related Topics

API function [API SNMP\\_GET](#) - Get internal TCPIP SNMP variables

[Top of list](#)

[Index page](#)

---

## tagTcpMib

```

typedef struct tagTcpMib
{
    long            tcpRtoAlgorithm; // retransmission timeout algorithm
    long            tcpRtoMin;      // minimum retransmission timeout (mS)
    long            tcpRtoMax;      // maximum retransmission timeout (mS)
    long            tcpMaxConn;      // maximum tcp connections possible
    unsigned long   tcpActiveOpens; // number of SYN-SENT -> CLOSED transitions
    unsigned long   tcpPassiveOpens; // number of SYN-RCVD -> LISTEN transitions
    unsigned long   tcpAttemptFails; // (SYN-SENT,SYN-RCVD)->CLOSED or SYN-RCVD->LISTEN
    unsigned long   tcpEstabResets;  // (ESTABLISHED,CLOSE-WAIT) -> CLOSED
    unsigned long   tcpCurrEstab;    // number in ESTABLISHED or CLOSE-WAIT state
    unsigned long   tcpInSegs;       // number of segments received
    unsigned long   tcpOutSegs;      // number of segments sent
    unsigned long   tcpRetransSegs;  // number of retransmitted segments
    unsigned long   tcpInErrs;       // number of received errors
    unsigned long   tcpOutRsts;      // number of transmitted resets
} TcpMib;

```

[Top of list](#)

[Index page](#)

---

## tagUdpMib

```

typedef struct tagUdpMib
{
    unsigned long   udpInDatagrams; // UDP datagrams delivered to users
    unsigned long   udpNoPorts;     // UDP datagrams to port with no listener
    unsigned long   udpInErrors;    // UDP datagrams unable to be delivered
    unsigned long   udpOutDatagrams; // UDP datagrams sent from this entity
}UdpMib;

```

## Comments

Note: These structures are only available at BIOS versions, which contain the SNMP option. A SNMP agent is not part of the BIOS. But if an user is able to implement an agent based on the TCPIP API, he needs access to the internal TCPIP SNMP variables

## Related Topics

API function [API SNMP\\_GET](#) - Get internal TCPIP SNMP variables

[Top of list](#)

[Index page](#)

---

## tag\_atEntry

```

typedef struct tag_atEntry {
    long            IfIndex;        // interface on which this entry maps
    unsigned char   PhysAddress[11]; // physical address of destination
    unsigned char   PhysAddressLen; // length of atPhysAddress
    unsigned long   NetAddress;     // IP address of physical address
}atEntry;

```

## Comments

Note: These structures are only available at BIOS versions, which contain the SNMP option. A SNMP agent is not part of the BIOS. But if an user is able to implement

an agent based on the TCPIP API, he needs access to the internal TCPIP SNMP variables

## Related Topics

API function [API SNMP\\_GET](#) - Get internal TCPIP SNMP variables

[Top of list](#)

[Index page](#)

---

End of document

---

## Programming client server applications - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### Client-Server

Programming client/server applications

Here is a short common description for programming client/server applications with the TCPIP API. The most used methods for programming TCP/IP applications are client or server applications.

The term server applies to any process or program that offers a service that can be reached over the network. Servers accept request that arrive over the network, perform their service, and return the result to the requester. An example for the simplest service is the standard echoserver application. The server echoes the received data over the network back to the requester. A process becomes a client when its sends a request to a server and waits for an answer. the client-server model is the standard model for interprocess communication. At TCP/IP communications there exist two different methods for client-server connections:

#### 1.UDP protocol:

This protocol realizes connectionless communication between a client and server, based on sending and receiving of single datagrams.

TCPIP API calls for an UDP client:

- [Open a socket](#)
- [Send an outgoing datagram usually a prerecorded endpoint address.](#)
- [Receive the next incoming datagram and record its source endpoint address.](#)
- [Close a socket](#)

TCPIP API calls for an UDP server:

- [Open a socket](#)
- [Bind a socket, assign an address to an unnamed socket](#)
- [Receive the next incoming datagram and record its source endpoint address.](#)
- [Send an outgoing datagram usually a prerecorded endpoint address.](#)

#### 2.TCP protocol:

The TCP protocol is a connection- and bytestream-oriented protocol

TCPIP API calls for a TCP client:

- [Open a socket](#)
- [Connect to a remote peer](#)
- [Send an outgoing stream of characters](#)
- [Receive an outgoing stream of characters](#)
- [Close a socket](#)

TCPIP API calls for a TCP server:

- [Open a socket](#)
- [Bind a socket, assign an address to an unnamed socket](#)
- [Place the socket in a passive mode](#)
- [Accept the next incoming connection](#)
- [Receive an outgoing stream of characters](#)
- [Send an outgoing stream of characters](#)
- [Close a socket](#)

We provide several examples for programming client/server applications:

- UDPEchoClient, udpcle.c, built with int86x calls
- UDPEchoServer, udpser.c, built with int86x calls
- TCPEchoClient, tcpclie.c, built with int86x calls
- TCPEchoServer, tcpserv.c, built with int86x calls
- TCPEchoClient, tcpclie.c, built with C-API-functions, using tcpip.c
- TCPEchoServer, tcpserv.c, built with C-API-functions, using tcpip.c

Note:

All program examples built with C-API-functions use the files tcpip.c, tcpip.h and tcpipapi.h.

---

End of document

---

## RTOS API - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

RTOS API [News](#)

---

### RTOS API

Here is the documentation for the RTOS API. This interface provides access to the RTOS of the IPC@Chip. Please note, that we cannot explain detailed all principles of a multitasking system at this document. A good book for understanding the architecture of Real-Time-Kernels is e.g.: MicroC/OS from Jean.J.Labrosse.

### Topics

RTOS API [Overview](#)

RTOS API [News](#)

RTOS API [Error Codes](#)

RTOS API [Developer](#)Notes

RTOS API [Examples](#)Available

RTOS API [Data Structures](#)

IPC@Chip System [Tasks](#)

### API Functions

The RTOS API uses interrupt 0xAD with a service number in the high order byte of the AX register (AH). The implemented RTOS services are listed below.

- [Interrupt 0xAD function 0x00: RTX\\_SLEEP\\_TIME, Go to sleep for a specified time](#)
- [Interrupt 0xAD function 0x01: RTX\\_TASK\\_CREATE, Create and start a task](#)
- [Interrupt 0xAD function 0x11: RTX\\_TASK\\_CREATE\\_WITOUT\\_RUN, Create a task](#)
- [Interrupt 0xAD function 0x02: RTX\\_TASK\\_KILL, Stop and kill specified task](#)
- [Interrupt 0xAD function 0x03: RTX\\_TASK\\_DELETE, Remove a task from the system](#)
- [Interrupt 0xAD function 0x04: RTX\\_GET\\_TASKID, Get ID of the currently executing task](#)
- [Interrupt 0xAD function 0x05: RTX\\_SLEEP\\_REQ. Sleep unconditionally for a wake request](#)
- [Interrupt 0xAD function 0x06: RTX\\_WAKEUP\\_TASK, To wake up a task](#)
- [Interrupt 0xAD function 0x07: RTX\\_END\\_EXEC, End execution of a task by itself](#)
- [Interrupt 0xAD function 0x08: RTX\\_CHANGE\\_PRIO, Change priority of a task](#)
- [Interrupt 0xAD function 0x14: RTX\\_CREATE\\_SEM, Create a semaphore](#)
- [Interrupt 0xAD function 0x15: RTX\\_DELETE\\_SEM, Delete a semaphore](#)
- [Interrupt 0xAD function 0x16: RTX\\_FREE\\_RES, Unconditionally free a resource semaphore](#)

- Interrupt 0xAD function 0x17: RTX\_GET\_SEM, Get use of a counting semaphore (no wait)
- Interrupt 0xAD function 0x18: RTX\_RELEASE\_SEM, Release a resource semaphore
- Interrupt 0xAD function 0x19: RTX\_RESERVE\_RES, Get use of a resource semaphore
- Interrupt 0xAD function 0x1A: RTX\_SIGNAL\_SEM, Signal a counting semaphore
- Interrupt 0xAD function 0x1B: RTX\_WAIT\_SEM, Wait on a counting semaphore
- Interrupt 0xAD function 0x28: RTX\_GET\_TIMEDATE, Get system time and date
- Interrupt 0xAD function 0x29: RTX\_SET\_TIMEDATE, Set system time and date
- Interrupt 0xAD function 0x2A: RTX\_GET\_TICKS, Get tick count of system clock
- Interrupt 0xAD function 0x09: RTX\_ACCESS\_FILESYSTEM, Enable file access for the calling task
- Interrupt 0xAD function 0x0A: RTX\_GET\_TASK\_STATE, Get state of a task
- Interrupt 0xAD function 0x0B: RTX\_GET\_TASK\_LIST, Get list of current tasks in the system
- Interrupt 0xAD function 0x0C: RTX\_START\_TASK\_MONITOR, Enable task monitoring
- Interrupt 0xAD function 0x0D: RTX\_STOP\_TASK\_MONITOR, Disable task monitoring
- Interrupt 0xAD function 0x0E: RTX\_SUSPEND\_TASK, Suspend a task
- Interrupt 0xAD function 0x0F: RTX\_RESUME\_TASK, Resume a task
- Interrupt 0xAD function 0x10: RTX\_RESTART\_TASK, Restart a task
- Interrupt 0xAD function 0x12: RTX\_GET\_TASK\_STATE\_EXT, Get state without task monitoring mode
- Interrupt 0xAD function 0x20: RTX\_DISABLE\_TASK\_SCHEDULING, Disable the task scheduler
- Interrupt 0xAD function 0x21: RTX\_ENABLE\_TASK\_SCHEDULING, enable the task scheduler
- Interrupt 0xAD function 0x30: RTX\_INSTALL\_TIMER, Install a timer procedure
- Interrupt 0xAD function 0x31: RTX\_REMOVE\_TIMER, Remove a timer procedure
- Interrupt 0xAD function 0x32: RTX\_START\_TIMER, Start periodic execution of a timer procedure
- Interrupt 0xAD function 0x33: RTX\_STOP\_TIMER, Stop execution of a timer procedure
- Interrupt 0xAD function 0x40: RTX\_CREATE\_EVENTGROUP, Create an event group
- Interrupt 0xAD function 0x41: RTX\_DELETE\_EVENTGROUP, Delete an event group
- Interrupt 0xAD function 0x42: RTX\_SIGNAL\_EVENTS, Signal one or more events in a group
- Interrupt 0xAD function 0x43: RTX\_WAIT\_EVENTS, Wait for events in a group
- Interrupt 0xAD function 0x44: RTX\_GET\_EVENTGROUP\_STATE, Read the current event states in a group
- Interrupt 0xAD function 0x45: RTX\_GET\_EVENT\_FLAGS, Get the saved event flags
- Interrupt 0xAD function 0x46: RTX\_FIND\_EVENTGROUP, Find an event group
- Interrupt 0xAD function 0x50: RTX\_CREATE\_MSG, Create a message exchange
- Interrupt 0xAD function 0x51: RTX\_DELETE\_MSG, Delete a message exchange
- Interrupt 0xAD function 0x52: RTX\_SEND\_MSG, Send message
- Interrupt 0xAD function 0x53: RTX\_GET\_MSG, Get message
- Interrupt 0xAD function 0x54: RTX\_WAIT\_MSG, Wait for a message
- Interrupt 0xAD function 0x55: RTX\_FIND\_MSG, Find a message exchange

At return from most of the API calls, the DX-Register is used for error checking as follows:

```
DX: 0 RTX_ENOERROR -> success
DX: -1 RTX_ERROR -> error, ax contains error code
DX: -2 RTX_NOT_SUPPORTED -> service is not supported by the API
```

All needed constants and data [structures](#) for the usage of the RTOS API are defined in header file rtxapi.h. For a better understanding of the RTOS API, some [example](#) programs written in C are provided. The user should read these example and modify them for your own applications.

---

## Interrupt 0xAD service 0x00: RTX\_SLEEP\_TIME, Go to sleep for a specified time

### Parameters

*AH*  
0 (=RTX\_SLEEP\_TIME)

*BX*  
Sleep time in milliseconds

### Return Value

DX =0 success AX: 0  
DX!=0 failure AX: contains error code

### Comments

Note:  
The [RTX\\_WAKEUP\\_TASK](#) API (service 0x06) can wake up a sleeping task before its sleep timer has expired.

### Related Topics

RTOS API [Error Codes](#)  
RTOS [Task](#) Control Services

---

## Developer Notes

A sleep call with parameter 1 millisecond takes equal or less than one millisecond. If an user needs a minimal sleeptime of 1 millisecond he must call RTX\_SLEEP\_TIME with value 2.

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAD service 0x01: RTX\_TASK\_CREATE, Create and start a task

### Parameters

*AH*

0x01 (= RTX\_TASK\_CREATE)

*BX:SI*

Pointer to 16 bit storage for the taskID, allocated by the caller

*ES:DI*

Pointer to a [TaskDefBlock](#) type data structure

## Return Value

DX =0 success AX: 0, task is running, location [BX:SI] contains the 16 bit taskID

DX!=0 failure AX: contains error code

## Comments

The caller must [fill](#) in portions of the [TaskDefBlock](#) structure prior to making this call.

The new task is immediately placed in the system's task ready queue. Execution begins if the task is higher priority than any other task currently ready (including task which called RTX\_TASK\_CREATE ).

Difference to [RTX\\_TASK\\_CREATE\\_WITHOUT\\_RUN](#) call: The new task runs after this call.

## Related Topics

RTOS API [Error Codes](#)

IPC@Chip System [Tasks](#)

RTOS [Task](#) Control Services

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x11: RTX\_TASK\_CREATE\_WITOUT\_RUN, Create a task

### Parameters

*AH*

0x11 (= RTX\_TASK\_CREATE\_NORUN)

*BX:SI*

Pointer to 16 bit storage for the taskID, allocated by the caller

*ES:DI*

Pointer to a [TaskDefBlock](#) type data structure

### Return Value

DX =0 success AX: 0, task is running, location [BX:SI] contains the 16 bit taskID

DX!=0 failure AX: contains error code

## Comments

The caller must **fill** in portions of the **TaskDefBlock** structure prior to making this call.

Difference to **RTX\_TASK\_CREATE** call: The new task must be started with **RTX\_RESTART\_TASK** call)

## Related Topics

RTOS API **Error Codes**

IPC@Chip System **Tasks**

RTOS **Task** Control Services

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x02: RTX\_TASK\_KILL, Stop and kill specified task

You should not delete or kill a task, which is waiting, or is about to wait for a resource or counting semaphore.

## Parameters

*AH*

0x02 (= RTX\_TASK\_KILL)

*BX*

taskID

## Return Value

DX =0 success AX: 0, task is terminated

DX!=0 failure AX: contains error code

## Related Topics

RTOS API **Error Codes**

RTOS **Task** Control Services

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x03: RTX\_TASK\_DELETE, Remove a task from the system

Remove specified task from system.

## Parameters

*AH*

0x03 (= RTX\_TASK\_DELETE)

*BX*

taskID

## Return Value

DX =0 success AX: 0, task is removed

DX!=0 failure AX: contains error code

## Comments

You should not delete or kill a task which is waiting, or is about to wait for a resource or counting semaphore.

After making this call, the taskID is no longer valid. A task can remove itself.

## Related Topics

RTOS API [Error Codes](#)

RTOS [Task](#) Control Services

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x04: RTX\_GET\_TASKID, Get ID of the currently executing task

Get ID of the currently executing task

## Parameters

*AH*

0x04 (= RTX\_GET\_TASKID)

## Return Value

DX=0 (success always) AX: contains the TaskID

## Related Topics

RTOS [Task](#) Control Services

## Interrupt 0xAD service 0x05: RTX\_SLEEP\_REQ. Sleep unconditionally for a wake request

The calling task will be suspended until some other task issues a RTX\_WAKEUP\_TASK call to wake this calling task.

### Parameters

*AH*  
0x05 (= RTX\_SLEEP\_REQ)

### Return Value

DX=0 (always success) AX: 0

### Comments

No return from this call occurs until task wakes up again.

### Related Topics

RTOS [Task](#) Control Services

## Interrupt 0xAD service 0x06: RTX\_WAKEUP\_TASK, To wake up a task

To wake up a task known to be waiting because of an RTX\_SLEEP\_REQ or RTX\_SLEEP\_TIME call.

### Parameters

*AH*  
0x06 (= RTX\_WAKEUP\_TASK)

*BX*  
taskID

### Return Value

DX =0 success AX: contains the TaskID  
DX!=0 failure AX: contains error code

## Related Topics

RTOS API [Error Codes](#)

RTOS [Task](#) Control Services

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x07: RTX\_END\_EXEC, End execution of a task by itself

End execution of a task by itself, if any task is waiting for finish processing its message (see [RTX\\_SEND\\_DIRECT\\_MSG](#) ) the Chip-RTOS wakes automatically this task. This call could only be used to terminate the task which is making the call.

### Parameters

*AH*

0x07 (= RTX\_END\_EXEC)

### Return Value

There is no return from this function

### Comments

This call should be used at the end of a task's function.

## Related Topics

RTOS [Task](#) Control Services

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x08: RTX\_CHANGE\_PRIO, Change priority of a task

### Parameters

*AH*

0x08 (= RTX\_CHANGE\_PRIO)

*BX*

taskID

*CX*  
priority, range 3 to 127 inclusive (3 is highest priority)

## Return Value

DX =0 success AX: 0 DX!=0 failure AX: contains error code

## Comments

An out of range priority value (*CX*) will be limited to range 3..127 inside this function.

## Related Topics

RTOS [Task](#) Control Services

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAD service 0x14: RTX\_CREATE\_SEM, Create a semaphore

### Parameters

*AH*  
0x14 (= RTX\_CREATE\_SEM)

*BX:SI*  
Pointer to 16 bit storage allocated by caller for a semaphoreID

*CX*  
Initial value

*ES:DI*  
Pointer to 4 character unique name tag for the new semaphore

### Return Value

DX =0 success AX: 0, Location referenced by [BX:SI] contains the unique semaphoreID  
DX!=0 failure AX: contains error code

## Related Topics

RTOS [Semaphore](#) Services  
RTOS API [Error Codes](#)

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAD service 0x15: RTX\_DELETE\_SEM, Delete a semaphore

### Parameters

*AH*

0x15 (= RTX\_DELETE\_SEM)

*BX*

ID of the semaphore acquired by RTX\_CREATE\_SEM

### Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code

### Related Topics

RTOS [Semaphore](#) Services

RTOS API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x16: RTX\_FREE\_RES, Unconditionally free a resource semaphore

The resource's use count is set to zero and the resource freed

### Parameters

*AH*

0x16 (= RTX\_FREE\_RES)

*BX*

ID of the resource semaphore acquired by RTX\_CREATE\_SEM

### Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code

### Related Topics

RTOS [Semaphore](#) Services

RTOS API [Error Codes](#)

## Interrupt 0xAD service 0x17: RTX\_GET\_SEM, Get use of a counting semaphore (no wait)

This call returns with an error, if the semaphore is in use by another caller

### Parameters

*AH*  
0x17 (= RTX\_GET\_SEM)

*BX*  
ID of the semaphore acquired by RTX\_CREATE\_SEM

### Return Value

DX =0 success AX: 0  
DX!=0 failure AX: contains error code, semaphore is in use

### Related Topics

RTOS [Semaphore](#) Services  
RTOS API [Error Codes](#)

## Interrupt 0xAD service 0x18: RTX\_RELEASE\_SEM, Release a resource semaphore

This call returns with an error, if the semaphore is in use by another caller

### Parameters

*AH*  
0x18 (= RTX\_RELEASE\_SEM)

*BX*  
ID of the resource semaphore acquired by RTX\_CREATE\_SEM

### Return Value

DX =0 success AX: 0  
DX!=0 failure AX: contains error code

## Related Topics

RTOS [Semaphore](#) Services

RTOS API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x19: RTX\_RESERVE\_RES, Get use of a resource semaphore

This call waits for a defined time to reserve a semaphore and returns with an error, if the semaphore is in use by another caller. The callers wait in FIFO order for the semaphore.

### Parameters

*AH*

0x19 (= RTX\_RESERVE\_RES)

*BX*

ID of the semaphore acquired by RTX\_CREATE\_SEM

*ES:DI*

Pointer to buffer, stored the timeout in milliseconds  
if \*timeout == 0, the caller waits forever for the resource

### Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code, semaphore is in use

## Related Topics

RTOS [Semaphore](#) Services

RTOS API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x1A: RTX\_SIGNAL\_SEM, Signal a counting semaphore

The semaphore will be given to the task which is waiting at the head of the wait queue of this semaphore.

### Parameters

*AH*

0x1A (= RTX\_SIGNAL\_SEM)

*BX*

ID of the semaphore acquired by RTX\_CREATE\_SEM

## Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code

## Related Topics

RTOS [Semaphore](#) Services

RTOS API [Error Codes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x1B: RTX\_WAIT\_SEM, Wait on a counting semaphore

This call waits for a defined time of a semaphore and returns with an error, if the semaphore is in use by another caller. The callers wait in FIFO order of the semaphore.

## Parameters

*AH*

0x1B (= RTX\_WAIT\_SEM)

*BX*

ID of the semaphore acquired by RTX\_CREATE\_SEM

*ES:DI*

Pointer to long buffer, stored the timeout in milliseconds  
if \*timeout == 0, the caller waits forever for the resource

## Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code, semaphore is in use

## Related Topics

RTOS [Semaphore](#) Services

RTOS API [Error Codes](#)

## Interrupt 0xAD service 0x28: RTX\_GET\_TIMEDATE, Get system time and date

### Parameters

*AH*  
0x28 (= RTX\_GET\_TIMEDATE)

*BX:SI*  
Pointer to `TimeDate_Structure` [type](#) allocated by user.

### Return Value

DX=0 success AX: 0, Location at [BX:SI] contains system date and time

### Related Topics

RTOS [Time/Date](#) Services  
`TimeDate_Structure` [type](#) definition

## Interrupt 0xAD service 0x29: RTX\_SET\_TIMEDATE, Set system time and date

### Parameters

*AH*  
0x29 (= RTX\_SET\_TIMEDATE)

*BX:SI*  
Pointer to `TimeDate_Structure` [type](#) filled in by user.

### Return Value

DX=0 success AX: 0

### Comments

The *Day Of Week* field (`.dow`) in `TimeDate_Structure` need not be set by caller. This API function computes this field based on the other member data.

Values for time/date supplied by the caller are not checked for validity.

## Related Topics

RTOS [Time/Date](#) Services  
TimeDate\_Structure [type](#) definition

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x2A: RTX\_GET\_TICKS, Get tick count of system clock

Accesses the system clock tick count.

### Parameters

*AH*

0x2A (= RTX\_GET\_TICKS)

*BX:SI*

Pointer to an unsigned long where the tick count will be stored.

### Comments

The system clock runs at 1000 Hz. So each tick represents 1 millisecond.

## Related Topics

RTOS [Time/Date](#) Services

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x09: RTX\_ACCESS\_FILESYSTEM, Enable file access for the calling task

### Parameters

*AH*

0x09 (= RTX\_ACCESS\_FILESYSTEM)

### Return Value

DX =0 success AX: 0  
DX!=0 failure (Mostly to many processes with fileaccess)

## Comments

If DX==3, file access is enabled, but reserving a data entry for findfirst/findnext failed.

## Related Topics

RTOS [Task](#) Control Services

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x0A: RTX\_GET\_TASK\_STATE, Get state of a task

### Parameters

*AH*

0x0A (= RTX\_GET\_TASK\_STATE)

*ES:DI*

Pointer to 4 character unique name tag of the task whose state information is desired

*DS:SI*

Pointer to Task\_StateData [type](#) structure, allocated by the user

### Return Value

DX =0 AX: contains taskID

Task\_StateData structure at [DS:SI] contains the current task state data

DX!=0 AX:0, task monitoring is not enabled

DX!=0 AX!=0, task not found

## Comments

For using this function the task monitoring mode must be enabled.

## Related Topics

Task\_StateData [structure](#) definition

RTOS [Task](#) Control Services

[Start](#) Task Monitor API call

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x0B: RTX\_GET\_TASK\_LIST, Get list of current tasks in the system

### Parameters

*AH*  
0x0B (= RTX\_GET\_TASK\_LIST)

*ES:DI*  
Pointer to array of `TaskList` [type](#) structures allocated by user.

*CX*  
length of the list

### Return Value

DX=0, BX = number of tasks listed in [ES:DI] array.

### Comments

The caller must allocate sufficient buffer space at [ES:DI] to allow all tasks to be reported, including those [created](#) by the system.

### Related Topics

`TaskList` [structure](#) definition  
RTOS [Task](#) Control Services

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAD service 0x0C: RTX\_START\_TASK\_MONITOR, Enable task monitoring

### Parameters

*AH*  
0x0C (= RTX\_START\_TASK\_MONITOR)

### Return Value

DX=0, AX=0

### Comments

This function installs a task timing function in the 0x13 timer interrupt, which will poll at 1000 Hz to check which task is currently executing. (Data collected by this timing function will provide a coarse indication of which tasks are occupying the CPU.)

## Related Topics

RTOS [Task](#) Control Services

[Stop](#) Task Monitor API call

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x0D: RTX\_STOP\_TASK\_MONITOR, Disable task monitoring

### Parameters

*AH*  
0x0D (= RTX\_STOP\_TASK\_MONITOR)

### Return Value

DX=0, AX=0

### Comments

This function restores the system's original 0x13 timer handler, removing any task timing function installed by RTX\_START\_TASK\_MONITOR call.

## Related Topics

RTOS [Task](#) Control Services

[Start](#) Task Monitor API call

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x0E: RTX\_SUSPEND\_TASK, Suspend a task

Suspend the execution of a task until some other task calls [RTX\\_RESUME\\_TASK](#) to resume the task.

### Parameters

*AH*  
0x0E (= RTX\_SUSPEND\_TASK)

*BX*  
taskID (value from [RTX\\_TASK\\_CREATE](#) call)

## Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code, invalid taskID

## Related Topics

RTOS [Task](#) Control Services

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x0F: RTX\_RESUME\_TASK, Resume a task

Enable the execution of a suspended task

## Parameters

*AH*

0x0F (= RTX\_RESUME\_TASK)

*BX*

taskID (value from [RTX\\_TASK\\_CREATE](#) call)

## Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code, invalid taskID

## Related Topics

RTOS [Task](#) Control Services

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x10: RTX\_RESTART\_TASK, Restart a task

Restart a killed task (killed with [RTX\\_TASK\\_KILL](#) call)

## Parameters

*AH*

0x0F (= RTX\_RESTART\_TASK)

*BX*

taskID

## Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code, invalid taskID

## Related Topics

RTOS [Task](#) Control Services

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x12: RTX\_GET\_TASK\_STATE\_EXT, Get state without task monitoring mode

### Parameters

*AH*

0x12 (= RTX\_GET\_TASK\_STATE\_EXT)

*ES:DI*

Pointer to 4 character unique name tag of the task whose state information is desired

### Return Value

DX =0 AX: contains taskID BX: contains the state of task (see below)

DX!=0 AX:!=0, task not found

### Comments

The `taskState` bit field is coded as follows:

- B0: timer wait (used with other bits)
- B1: trigger wait (i.e. idle)
- B2: semaphore wait
- B3: event group wait
- B4: message exchange wait
- B5: message send wait
- B6: suspended (waiting for resume)
- B7: waiting for wakeup

B8 - B15 internal use only

### Related Topics

API function [RTX\\_GET\\_TASK\\_STATE](#) - Get state of a task

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x20: RTX\_DISABLE\_TASK\_SCHEDULING, Disable the task scheduler

The task which call this function wont be interrupted by other tasks until RTX\_ENABLE\_TASK\_SCHEDULER will be called. Interrupts are enabled while this process. Installed Timer procedures wont be executed. The watchdog must be triggered (using the Hardware API Call [Refresh Watchdog](#) ) by the user until the task scheduler is disabled. API fuctions which calls another task (like TCPIP API functions) and also the API Sleep function must not used.

### Parameters

AH  
0x20 (= RTX\_DISABLE\_TASK\_SCHEDULING)

### Comments

NOTE: Must be followed by a call to [RTX\\_ENABLE\\_TASK\\_SCHEDULER](#) as soon as possible to enable the task scheduler.

### Related Topics

API function [RTX\\_ENABLE\\_TASK\\_SCHEDULER](#) - Enables the task scheduler

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x21: RTX\_ENABLE\_TASK\_SCHEDULING, enable the task scheduler

Must be follow to a RTX\_DISABLE\_TASK\_SCHEDULER to enable the task switching again.

### Parameters

AH  
0x21 (= RTX\_ENABLE\_TASK\_SCHEDULING)

### Related Topics

API function [RTX\\_DISABLE\\_TASK\\_SCHEDULER](#) - Disables the task scheduler

## Interrupt 0xAD service 0x30: RTX\_INSTALL\_TIMER, Install a timer procedure

Install a timer procedure that will be periodically executed by the kernel.

### Parameters

*AH*  
0x30 (= RTX\_INSTALL\_TIMER)

*ES:DI*  
Pointer to a `TimerProc_Structure` [type](#)

### Return Value

DX = 0 success AX: 0  
DX != 0 failure AX: contains error code, no free timer available

### Comments

See the `TimerProc_Structure` [type](#) description for instructions on how to call this function.

A timer ID is output to the 16 bit location referenced by `timerID` [member](#) of your `TimerProc_Structure`.

You must call the [RTX\\_START\\_TIMER](#) API function to get the kernel to start calling your new timer procedure.

#### Important:

Timer procedures are executed on the stack of the kernel task at a high priority, so they should be as short as possible. Avoid calling large functions like `printf()`.

### Related Topics

`TimerProc_Structure` [definition](#)  
RTOS [Timer](#) Procedures

## Interrupt 0xAD service 0x31: RTX\_REMOVE\_TIMER, Remove a timer procedure

Stop execution and remove a timer procedure.

## Parameters

*AH*

0x31 (= RTX\_REMOVE\_TIMER)

*BX*

timerID produced by the [RTX\\_INSTALL\\_TIMER](#) call

## Return Value

DX =0 success AX: 0

DX!=0, failure AX contains error code, invalid timerID.

## Comments

It is possible to restart a timer procedure after removing it from the system.

## Related Topics

RTOS [Timer](#) Procedures

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x32: RTX\_START\_TIMER, Start periodic execution of a timer procedure

Start periodic execution of a timer procedure.

## Parameters

*AH*

0x32 (= RTX\_START\_TIMER)

*BX*

timerID produced by the [RTX\\_INSTALL\\_TIMER](#) call

## Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code, invalid timerID.

## Related Topics

RTOS [Timer](#) Procedures

## Interrupt 0xAD service 0x33: RTX\_STOP\_TIMER, Stop execution of a timer procedure

Stop execution of a timer procedure

### Parameters

*AH*  
0x33 (= RTX\_STOP\_TIMER)

*BX*  
timerID produced by the [RTX\\_INSTALL\\_TIMER](#) call

### Return Value

DX =0 success AX: 0  
DX!=0 failure AX: contains error code, invalid timerID.

### Related Topics

RTOS [Timer](#) Procedures

## Interrupt 0xAD service 0x40: RTX\_CREATE\_EVENTGROUP, Create an event group

### Parameters

*AH*  
0x40 (= RTX\_CREATE\_EVENTGROUP)

*BX*  
Initial value of the 16 event flags of the group.

*ES:DI*  
Pointer to 16 bit storage for the unique group ID integer

*DS:SI*  
Pointer to unique four character tag

### Return Value

DX =0 success AX: 0 , location at [ES:DI] contains the unique group ID.  
DX!=0 failure AX: contains error code, no free event group entry available

## Comments

Each event group includes 16 event flags, the maximum number of event groups is 2. Each event group has a unique ID. You can provide a unique 4 character tag to identify the event group. Each event in a group is represented by a Boolean flag representing the state of the event. The 16 Boolean flags are represented in one 16-Bit word. If a event flag (bit) has the value one, the event has occurred. A zero value means, that the event has not occurred.

## Related Topics

RTOS [Event](#) Manager

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x41: RTX\_DELETE\_EVENTGROUP, Delete an event group

### Parameters

*AH*  
0x41 (= RTX\_DELETE\_EVENTGROUP)

*BX*  
Event group ID acquired by [RTX\\_CREATE\\_EVENTGROUP](#) call.

### Return Value

DX =0 success AX: 0  
DX!=0 failure AX: contains error code, event group still in use or invalid group ID

## Comments

You should not delete an event group which is in use by another task or timer procedure.

## Related Topics

RTOS [Event](#) Manager

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x42: RTX\_SIGNAL\_EVENTS, Signal one or more events in a group

Signal that one or more events in a group has occurred by setting the event flags.

If any tasks are waiting on these events, the Event Manager will now wake them up.

## Parameters

*AH*

0x42 (= RTX\_SIGNAL\_EVENTS)

*BX*

Event group ID acquired by [RTX\\_CREATE\\_EVENTGROUP](#) call.

*CX*

16-Bit mask identifying the flags of interest in the group.

*DX*

Event value for the 16 event flag/bits. Only the bits marked '1' in the CX mask are relevant in DX.

## Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code

## Comments

Tasks wake up only on the 0 to 1 (=occurred) event bit transitions. The resetting of an event bit will not cause any task waiting on that event to wake up.

## Related Topics

RTOS API [Error Codes](#)

RTOS [Event](#) Manager

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x43: RTX\_WAIT\_EVENTS, Wait for events in a group

The calling tasks are waiting until the Event Manager wakes them up (the events has occurred) or the specified timeout was reached

## Parameters

*AH*

0x43 (= RTX\_WAIT\_EVENTS)

*BX*

Event group ID acquired by [RTX\\_CREATE\\_EVENTGROUP](#) call.

*ES:DI*

Pointer to user `RTX_Wait_Event` [type](#) structure

## Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code

## Comments

The user has to fill in the `RTX_Wait_Event` [structure](#) before making this call.

## Related Topics

`RTX_Wait_Event` [type](#) definition

RTOS API [Error Codes](#)

RTOS [Event](#) Manager

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x44: RTX\_GET\_EVENTGROUP\_STATE, Read the current event states in a group

Return the current state of the 16 event flags (bits) of a specified event group

## Parameters

*AH*

0x44 (= `RTX_GET_EVENTGROUP_STATE`)

*BX*

Event group ID acquired by [RTX\\_CREATE\\_EVENTGROUP](#) call.

*ES:DI*

Pointer to 16 bit location to receive the current state of the event flags

## Return Value

DX =0 success AX: 0 , Location at [ES:DI] contains the event states of the specified group

DX!=0 failure AX: contains error code, invalid group ID

## Related Topics

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x45: RTX\_GET\_EVENT\_FLAGS, Get the saved event flags

Return the state of the 16 event flags as they were at the time the calling task most recently completed a [RTX\\_WAIT\\_EVENTS](#) call.

### Parameters

*AH*

0x45 (= RTX\_GET\_EVENT\_FLAGS)

### Return Value

DX =0 success AX: contains the saved event states

### Comments

The returned event flags apply to this task's most recent event wait wake up or timeout.

### Related Topics

RTOS API [Error Codes](#)

RTOS [Event](#) Manager

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x46: RTX\_FIND\_EVENTGROUP, Find an event group

Find an event group by specified name tag and return the unique event group ID.

### Parameters

*AH*

0x44 (= RTX\_FIND\_EVENTGROUP)

*ES:DI*

Pointer to 4 character name tag

## Return Value

DX =0 success AX: contains the event group ID  
DX!=0 failure AX: contains error code, not found

## Comments

The name string at [ES:DI] does not need to be zero terminated.

## Related Topics

RTOS API [Error Codes](#)  
RTOS [Event](#) Manager

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x50: RTX\_CREATE\_MSG, Create a message exchange

A message exchange must be created by an application before it can be used. The Message Exchange Manager returns a 16-Bit unique ID to the caller. You can provide a unique 4 character tag to identify the message exchange.

## Parameters

*AH*

0x50 (= RTX\_CREATE\_MSG)

*ES:DI*

Pointer to user RTX\_Msg [type](#) structure

## Return Value

DX =0 success AX: 0, RTX\_Msg structure contains the new msgID  
DX!=0 failure AX: contains error code

## Comments

The user has to fill in portions of the RTX\_Msg [structure](#) prior to calling here.

The maximum number of message exchanges supported by the system is ten.

## Related Topics

RTX\_Msg [type](#) definition  
RTOS API [Error Codes](#)  
RTOS [Message](#) Exchange Manager

## Interrupt 0xAD service 0x51: RTX\_DELETE\_MSG, Delete a message exchange

### Parameters

*AH*  
0x51 (= RTX\_DELETE\_MSG)

*BX*  
Message exchange ID acquired by [RTX\\_CREATE\\_MSG](#) call.

### Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code, message exchange still in use or invalid ID

### Comments

You should not delete a message exchange which is in use by another task or timer procedure.

### Related Topics

RTOS API [Error Codes](#)

RTOS [Message](#) Exchange Manager

## Interrupt 0xAD service 0x52: RTX\_SEND\_MSG, Send message

Send provided message to a specified message exchange.

### Parameters

*AH*  
0x52 (= RTX\_SEND\_MSG)

*BX*  
Message exchange ID acquired by [RTX\\_CREATE\\_MSG](#) call.

*CX*

Message priority (mailbox) 0 - 3 where 0 is highest priority

*ES:DI*

Pointer to a 12 byte message to be sent

## Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code

## Comments

If one or more tasks are waiting at the exchange for a message, the message will be immediately given to the task waiting at the head of the exchange's wait queue.

The format of the 12 byte message being sent is defined by the application program.

## Related Topics

RTOS API [Error Codes](#)

RTOS [Message](#) Exchange Manager

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x53: RTX\_GET\_MSG, Get message

Get a message from a specified message exchange.

## Parameters

*AH*

0x53 (= RTX\_GET\_MSG)

*BX*

Message exchange ID acquired by [RTX\\_CREATE\\_MSG](#) call.

*ES:DI*

Pointer to a 12 byte user buffer for storing the message (if any)

## Return Value

DX =0 success AX: 0, Location at [ES:DI] holds the message

DX!=0 failure AX: contains error code, invalid ID or -28: no message available

## Comments

This function returns immediately with ax= -28 if no message is available.

## Related Topics

RTOS API [Error Codes](#)

RTOS [Message](#) Exchange Manager

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x54: RTX\_WAIT\_MSG, Wait for a message

This function parks the calling task on the specified message exchange message queue. The Message Exchange Manager will then wake up this task when either a message is available or a specified wait period timeout expires.

### Parameters

*AH*  
0x54

*ES:DI*  
Pointer to user RTX\_Wait\_Msg [type](#) structure

### Return Value

DX =0 success AX: 0  
DX!=0 failure AX: contains error code

### Comments

The user must fill in the RTX\_Wait\_Msg [structure](#) prior to calling here.

To wait in FIFO order, all callers have to wait with the same priority

## Related Topics

RTOS\_Wait\_Msg [type](#) definition

RTOS API [Error Codes](#)

RTOS [Message](#) Exchange Manager

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAD service 0x55: RTX\_FIND\_MSG, Find a message exchange

Find a message exchange by specified name **tag** and return the unique exchange ID.

## Parameters

*AH*

0x55 (= RTX\_FIND\_MSG)

*ES:DI*

Pointer to 4 character name **tag** (no zero terminator needed)

## Return Value

DX =0 success AX: contains the message exchange ID

DX!=0 failure AX: contains error code, not found

## Comments

If more than one message exchange was created with the same tag, you will get back the message exchange ID of one with this tag, but which one is not certain.

## Related Topics

RTOS API [Error Codes](#)

RTOS [Message](#) Exchange Manager

[Top of list](#)

[Index page](#)

---

End of document

---

## Web server CGI interface - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

CGI API [News](#)

---

### CGI API

The CGI ("Common Gateway Interface") API uses interrupt 0xAB with a service number in the high order byte of the AX register (AH). This interface provides access to the CGI implementation of the IPC@Chip Web server.

### Topics

Web Server [Overview](#)

Web Server [FileTypes](#)

CGI API [News](#)

CGI API [Error Codes](#)

CGI API [DeveloperNotes](#)

CGI API [Examples](#) Available

CGI API [Data Structures](#)

### API Functions Available

- [Interrupt 0xAB function 0x01: CGI\\_INSTALL, Install a CGI function](#)
- [Interrupt 0xAB function 0x02: CGI\\_REMOVE, Remove a CGI function](#)
- [Interrupt 0xAB function 0x03: CGI\\_SETMAIN, Set a new main page](#)
- [Interrupt 0xAB function 0x04: CGI\\_SETROOTDIR, Set Web server's root directory](#)
- [Interrupt 0xAB function 0x05: CGI\\_GETROOTDIR, Get Web server's root directory](#)
- [Interrupt 0xAB function 0x06: CGI\\_GETMAIN, Get main page name](#)
- [Interrupt 0xAB function 0x07: CGI\\_GETFORMITEM, Split a formular into name and value](#)
- [Interrupt 0xAB function 0x08: CGI\\_FINDNEXTITEM, Return the address of the next formular tag](#)
- [Interrupt 0xAB function 0x09: CGI\\_INSTALL\\_PAS, Install a Turbo Pascal CGI procedure](#)

---

### Interrupt 0xAB service 0x01: CGI\_INSTALL, Install a CGI function

### Parameters

AH

0x01 (= CGI\_INSTALL)

*DX:SI*

Pointer to a temporary CGI\_Entry **type** structure.

## Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code CGI\_NO\_FREE\_ENTRY or CGI\_INVALID\_METHOD

## Comments

This API function makes a copy of the information in the provided CGI\_Entry structure, so your structure at [DX:SI] need not be persistent.

## Related Topics

CGI API **Error** Codes

cgistat **command** line

**CGI\_INSTALL\_PAS** API Function, for Pascal CGI Procedures

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAB service 0x02: CGI\_REMOVE, Remove a CGI function

### Parameters

*AH*

0x02 (= CGI\_REMOVE)

*DX:SI*

Pointer to the null terminated URL path name

### Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code CGI\_NOT\_FOUND

### Comments

The CGI function to be deleted is identified by the provided URL string. It is also possible to remove the two predefined cgi functions main.htm and ChipCfg from the table.

## Related Topics

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAB service 0x03: CGI\_SETMAIN, Set a new main page

### Parameters

*AH*

0x03 (= CGI\_SETMAIN)

*DX:SI*

Pointer to name of new main page

### Return Value

DX =0 success AX: 0

DX!=0 failure AX: error code CGI\_INVALID\_NAME

### Comments

The string at [DX:SI] is null terminated with a maximum length of 64 characters (not counting the terminating zero).

### Related Topics

[CGI\\_GETMAIN](#) API Function

CGI API **Error** Codes

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAB service 0x04: CGI\_SETROOTDIR, Set Web server's root directory

### Parameters

*AH*

0x04 (= CGI\_SETROOTDIR)

*DX:SI*

Pointer to the name of new root directory

### Return Value

DX =0 success AX: 0  
DX!=0 failure AX: error code CGI\_INVALID\_DIR

## Comments

The string at [DX:SI] is null terminated with a maximum length of 64 characters (not counting the terminating zero).

## Related Topics

[CGI\\_GETROOTDIR](#) API Function  
CGI API [Error](#) Codes

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAB service 0x05: CGI\_GETROOTDIR, Get Web server's root directory

### Parameters

AH

0x05 (= CGI\_GETROOTDIR)

### Return Value

DX=0 AX=0 , ES:DI contains pointer to root directory name

## Comments

The string referenced by [ES:DI] is null terminated and is in the RTOS's data space.

## Related Topics

[CGI\\_SETROOTDIR](#) API Function

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAB service 0x06: CGI\_GETMAIN, Get main page name

### Parameters

AH

0x06 (= CGI\_GETMAIN)

## Return Value

DX=0 AX=0 , ES:DI contains pointer to current main page name

## Comments

The string referenced by [ES:DI] is null terminated and is in the RTOS's data space.

## Related Topics

[CGI\\_SETMAIN](#) API Function

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAB service 0x07: CGI\_GETFORMITEM, Split a formular into name and value

Parse the argument buffer to obtain name and value.

## Parameters

*AH*

0x07 (= CGI\_GETFORMITEM)

*BX:SI*

Pointer to argument buffer to be parsed.

*ES:DI*

Pointer to a `FormItem` [type](#) structure

## Return Value

DX=0 AX=0, User buffers referenced by pointers in `FormItem` structure at [ES:DI] are filled in with name and value

## Comments

On initial call, the argument buffer pointer provided by the caller in BX:SI is a copy of the `fArgumentBufferPtr` member of the `rbCgi` [structure](#) passed by the Web server to the CGI callback function. On subsequent calls here to pick up additional formular, the pointer returned from the `CGI_FINDNEXTITEM` API call can be used here.

The caller must set the two members of the `FormItem` [structure](#) prior to calling here. Both pointers reference buffers allocated by the user, which will receive strings produced by this API call.

See [example](#) submit.c.

## Related Topics

[CGI\\_FINDNEXTITEM](#) API Function

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAB service 0x08: CGI\_FINDNEXTITEM, Return the address of the next formular tag

Most formulars have more than one item, this function searches for the next form item in a CGI request argument string. This function can only be used after a [CGIFORMITEM](#) API call. (See [example](#) submit2.c)

## Parameters

*AH*

0x08 (= CGI\_FINDNEXTITEM)

*BX:SI*

CGI request argument pointer

## Return Value

DX=0 AX=0, ES:DI: pointer to the found item

DX=-1 AX=0, no next item was found

## Comments

The CGI request argument buffer pointer provided by the caller in BX:SI is initially taken from the `rbCgi` structure passed by the Web server to the CGI callback function.

This function scans the buffer at [BX:SI] for an ampersand character, '&', and if found returns a pointer to the character in the string following the ampersand.

The strings must be null terminated.

## Related Topics

[CGI\\_GETFORMITEM](#) API Function

[Top of list](#)

[Index page](#)

---

# Interrupt 0xAB service 0x09: CGI\_INSTALL\_PAS, Install a Turbo Pascal CGI procedure

Special install function for Turbo Pascal CGI procedures

## Parameters

*AH*

0x09 (= CGI\_INSTALL\_PAS)

*DX:SI*

Pointer to a temporary CGI\_Entry **type** structure.

## Return Value

DX =0 success AX: 0

DX!=0 failure AX: contains error code CGI\_NO\_FREE\_ENTRY or CGI\_INVALID\_METHOD

## Comments

This API function makes a copy of the information in the provided CGI\_Entry structure, so your structure at [DX:SI] need not be persistent.

## Related Topics

[CGI\\_INSTALL](#) API Function, for C CGI Procedures

[Top of list](#)

[Index page](#)

---

End of document

---

## CGI API Updates - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### CGI API News

The following extensions to the CGI [API](#) are available in the indicated BIOS revisions.

New in version 0.65: [Added install function for Turbo Pascal CGI procedures](#)

---

End of document

---

## Web Server Overview - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

CGI API [News](#)

---

### Web Server Overview

The explanations here assume the reader is somewhat familiar with the HTTP protocol.

#### Web Server / CGI topics:

- [CGI API Functions](#)
  - [CGI in the IPC@Chip](#)
  - [Built-In CGI Functions](#)
  - [Building a CGI function using a C compiler](#)
- 

### CGI API Functions

The CGI [API](#) uses interrupt 0xAB, enabling the application programmer to install their own DOS program CGI functions in the IPC@Chip Web server's CGI table. This makes it possible to visualize and control an application running in the IPC@Chip via an Internet browser, using all features of modern Internet technology.

[Top of list](#)

[Index page](#)

---

### CGI in the IPC@Chip

The Web server of the IPC@Chip uses an internal CGI table. The CGI table is an array of CGI\_Entry [type](#) structures. Defined for each entry is the URL name, the expected HTTP [method](#) (Get, Head or Post) and a pointer to a [function](#) which will be executed if a matching browser request arrives at the IPC@Chip.

[Top of list](#)

[Index page](#)

---

## Built-In CGI Functions

There are two pre-installed CGI functions in the IPC@Chip, one of them named "ChipCfg". The preset entry in the internal CGI table has the name "ChipCfg" (Note: URL names of CGI are case sensitive!). A CGI callback function for this entry supports the method "Get" by producing a Web "file" in RAM. This function is executed if a browser makes the following request to an IPC@Chip device with an IP address of 192.168.205.4:

```
http://192.168.205.4/ChipCfg
```

The CGI function then produces a HTML page in memory which contains specific configuration data, like IP address, subnet mask, serial number, etc..

[Top of list](#)

[Index page](#)

---

## Building a CGI function using a C compiler

A CGI function built with the Borland C compilers must be declared as:

```
void huge _pascal CGI_Func(rpCgi far *CgiRequest);
```

Using the Microsoft Visual C compilers the CGI functions are declared as:

```
void far _saveregs _loadds _pascal CGI_Func(rpCgi far *CgiRequest);
```

The Web server calls this function with the address of a `rpCgi` **typedata** structure, which contains all needed information about the browser request. Another part of this structure is the response fields. These fields must be set by the CGI function. They hold information needed by the Web server.

[Top of list](#)

[Index page](#)

---

End of document

---

## CGI File Types - BIOS V1.02 Beta

IPC@Chip Documentation [Index](#)

CGI API [News](#)

### CGI File Extensions

Here is a mapping between values for the fDatatype member of the rpCGI [type](#) and file extensions supported by the IPC@Chip Web server.

fDatatype Value	File Extension	Web Server File Type
0	*.htm	text/HTML
1	*.gif	image/gif
3	*.txt	text/plain
4	*.jpg	image/jpeg
5	*.pct	image/pict
6	*.tif	image/tiff
10	*.css	text/css
11	*.xml	text/xml
12	*.wav	audio/wav
13	*.pdf	application/pdf
14	*.jar	application/java-archive
16	*.wml	text/vnd.wap.wml
17	*.vmp	image/vnd.wap.wbmp
18	*.vmc	application/vnd.wap.wmlc
19	*.vms	text/vnd.wap.wmlscript
20	*.wmx	text/vnd.wap.wmlscriptc
7		image/png
8		application/x-www-form-urlencoded
9		application/ipp
15		application/octet-stream

The default type is application/octet-stream (fDatatype = 15)

[Top of list](#)

[Index page](#)



## CGI Error Codes - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### CGI API Error Codes

All error codes listed here are defined in the header file, cgi.h.

CGI error codes returned by CGI **API** calls in the **DX-Register**

0 = CGI_ENOERROR	-> success
-1 = CGI_ERROR	-> error, AX contains error code
-2 = CGI_NOT_SUPPORTED	-> invalid function number was in the AH-reg.

CGI specific error codes returned by CGI API calls in the **AX register** when DX register was non-zero (error indication):

-1 = CGI_INVALID_METHOD	-> invalid method
-2 = CGI_INVALID_NAME	-> invalid URL name
-3 = CGI_INVALID_DIR	-> invalid directory
-4 = CGI_NO_FREE_ENTRY	-> no space in CGI table
-5 = CGI_NOT_FOUND	-> entry not found

---

End of document

---

## CGI Application Developers Note - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

CGI [News](#)

---

### Developer Notes

Since Bios version 0.65, we added five new content [types](#) for CGI.

Since BIOS version 0.65, it is possible to write CGI procedures with Turbo Pascal. Consequently the parameter passing mechanism for CGI functions has been changed to that used by Pascal. C programmers will now need to use the following forms of CGI functions.

A CGI function using the Borland C compilers must be declared as follows:

```
void huge _pascal CGI_Func(rpCgi far *CgiRequest);
```

... and using the Microsoft Visual C compilers:

```
void far _saveregs _loadds _pascal CGI_Func(rpCgi far *CgiRequest);
```

---

CGI [API](#) Listing

---

End of document

---

## CGI Examples Available - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

CGI [News](#)

---

### CGI Examples

Available CGI [API examples](#) in C:

1. `example1.c` - **Builds** an HTML page
2. `example2.c` - Builds a **dynamic** HTML page
3. `dk40cgi.c` - Reads DK40 I/O **pins**
4. `dk40_set.c` - Reads and **writes** DK40 I/O pins
5. `secure.c` - Example of **password** protected page
6. `submit.c` - Building of **formular**
7. `submit2.c` - Formulars with **multiple** items

Also there are source files (`cgiapi.c`, `cgiapi.h`) containing wrapper functions which provide a C-Library interface to the assembly language/software interrupt based CGI [API](#).

Available Turbo [Pascal](#) examples:

1. `example1.pas` - **Builds** an HTML page
2. `example2.pas` - Builds a **dynamic** HTML page

### Examples in C

For understanding CGI in the IPC@Chip we provide some example programs written in C. The examples are compiled with various versions of Borland and Microsoft compilers. Which compiler version was used for a particular example is stated within the example's source files. The include file `cgi.h` contains all the required type definitions and constants.

#### 1. **example1.exe**

The CGI function installed by this program produces a HTML page which contains some of the browser's request parameters. This program was tested and compiled with Borland C 3.0 and Microsoft Visual C 1.52. The compiler differences are described in the source files.

Examples browser inputs:

```
http://192.168.205.4/example1
```

```
http://192.168.205.4/example1?Argument
```

#### 2. **example2.exe**

We build a dynamic HTML page which contains the current value of a counter incremented in the main loop of the program. This program is tested and compiled with Borland C 3.0 and Microsoft Visual C 1.52. The compilers differences are described in the source files.

Example browser input:

```
http://192.168.205.4/example2
```

### 3. **dk40cgi.exe**

The CGI function of this program produces a HTML page which contains the current values of the DK40 I/O pins.

Example browser inputs:

```
http://192.168.205.4/dk40
```

### 4. **dk40\_set.exe**

Demonstrates set and reset control over the DK40 output pins via browser.

Example browser input:

```
http://192.168.205.4/dk40
```

### 5. **secure.exe**

This is an example of a protected page.

The first browser input ...

```
http://192.168.205.4/dk40_secure
```

requires the input of a valid username and password, e.g.:

```
Username: user
```

```
Password: password
```

### 6. **submit.exe**

The building of formulars is demonstrated here.

### 7. **submit2.exe**

The building of formulars with more than one item is demonstrated here.

## Important:

1. The recommended memory model for DOS programs is "Large".
2. CGI functions should be programmed as short as possible, without long or endless waits.
3. CGI functions compiled with Borland C must be declared as "huge".
4. CGI functions compiled with Microsoft C must be declared as "far \_saveregs \_loadds".
5. Users of Microsoft Visual C must set the compiler option "struct member byte alignment" to "1 byte" or must use "#pragma pack(1)" in their source.
6. The **command** `cgistat` at the IPC@Chip command prompt lists all CGI functions installed.
7. URL names for CGI functions are case-sensitive.
8. If you use Microsoft C-Compilers then increase the Web server's **WEBSERVERSTACK** stack size value in the `chip.ini` file. The default stack size of the Web server task is 2048 Bytes. Programmers of CGI functions who are using Microsoft C-Compilers with C-Library functions, e.g. `sprintf`, which requires a lot of stack space should increase this allocation to 6144 (6 Kbytes). More stack space for the Web server task is also required if your CGI function uses a large amount of stack for automatic data (local variables) declared inside the CGI function call.

---

## Building Turbo Pascal CGI procedures

Since BIOS version 0.65, it is possible to write CGI procedures with Turbo Pascal  
This is a little bit different from writing a CGI function with the C compilers.

A Turbo Pascal program which contain a CGI procedure uses the same data structures (records) as a C language CGI function.

## Declaration of Turbo Pascal CGI procedures

A CGI procedure written with Turbo Pascal must be declared without any parameters and with the interrupt declaration, e.g.:

```
procedure Example1_Proc;interrupt;
```

The interrupt declaration will motivate the compiler to emit code that sets the CPU's DS data segment register on entry to the procedure, thus allowing access to the Pascal program's data.

The IPC@Chip Web server handles the call to a Pascal CGI function differently than it does the call to a C language CGI function. One input parameter is needed by all CGI functions, C or Pascal. This one parameter is a far pointer to a `rpCgi` **type** structure (or in Pascal, a record). When calling a C language CGI function this pointer is pushed onto the stack using a normal parameter passing mechanism. For Pascal CGI functions this pointer is instead passed in the CPU's ES:DI registers. Consequently this pointer must be recovered by the Pascal program. This can be done as follows:

```
procedure Example1_Proc;interrupt;
var
  ESreg      :   Integer;
  DIreg      :   Integer;
  CGIRequest :   rpCGIptr;

begin
  asm
    mov ax,es
    mov Esreg,ax
    mov ax,di
    mov DIreg,ax
  end;
  CGIRequest := ptr(ESreg,DIreg);
  .....
end;
```

## Installing a Pascal CGI procedure:

Pascal CGI procedure must installed at the start of the DOS program with the **CGI\_INSTALL\_PAS** API call. (CGI functions written in C must still be installed with the standard **CGI\_INSTALL** API.)

For better understanding of programming CGI with Turbo Pascal, we provide some example programs compiled with Borland Pascal 7.0.

### 1. **example1.exe**

The CGI function installed by this program produces a HTML page which contains some of the request parameters.

Examples for browser inputs:

```
http://192.168.205.4/example1
```

```
http://192.168.205.4/example1?Argument
```

### 2. **example2.exe**

We build a dynamic HTML page which contains the current value of a counter incremented in the main loop of the program.

Browser input e.g.: `http://192.168.205.4/example2`

---

End of document

---

## Data Structures used in CGI API - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### Data Structures

Here are the data structures used by the CGI [API](#) .

All constants and data structures are defined in the header file `cgi.h`

Notes:

- **Byte alignment is required** for all data structures used within the API.

Content :

- [typedefCGI\\_Entry](#)
- [defineCGI HTTP Request](#)
- [typedefFormItem](#)
- [typedefrpCgi](#)

---

### CGI\_Entry

```
typedef struct tag_cgi_table
{
    char          *PathPtr;      // Name of the page, URL
    int           method;       // http method: Get, Head or Post
    RpCgiFuncPtr CgiFuncPtr;   // ptr to callback function for this page
} CGI_Entry;
```

### Comments

The members of the `CGI_Entry` structure should be filled as follows.

#### **PathPtr**

This is a far pointer to a zero terminated URL address of the page. (URL's are case sensitive.)

#### **method**

This `int` is an [enumerator](#) used to specify what HTTP method this CGI function supports.

#### **CgiFuncPtr**

This is a far vector to the CGI function for this Web page. For Borland C compilers the `RpCgiFuncPtr` type is:

```
typedef void (huge _pascal *RpCgiFuncPtr)(rpCgi far *CgiRequestPtr);
... and therefore the CGI function itself is declared as ...
void huge _pascal CGI_Func(rpCgi far *CgiRequest);
```

For Microsoft Visual C compilers the RpCgiFuncPtr type is:

```
typedef void far _saveregs _loadds _pascal (*RpCgiFuncPtr)(rpCgi far
*CgiRequestPtr);
... and the CGI function declared as ...
void far _saveregs _loadds _pascal CGI_Func(rpCgi far *CgiRequest);
```

## Related Topics

API function [CGI\\_INSTALL](#) - Install a CGI function  
rpCgi [structure](#) type

[Top of list](#)  
[Index page](#)

---

## CGI HTTP Request

```
#define CgiHttpGet 1 // Cgi request is HTTP GET
#define CgiHttpHead 2 // Cgi request is HTTP HEAD
#define CgiHttpPost 3 // Cgi request is HTTP POST
```

## Comments

These defines are used as enumeration names for request methods.

## Related Topics

[method](#) member of CGI\_Entry type

[Top of list](#)  
[Index page](#)

---

## FormItem

```
typedef struct tag_form_item
{
    char *NamePtr;
    char *ValuePtr;
} FormItem;
```

## Comments

Both strings referenced here are null terminated.

## Related Topics

[Top of list](#)  
[Index page](#)

---

## rpCgi

```
typedef struct {
    /*******
    //      Request fields  (Read Only!!!!)
    /*******
    unsigned char  fConnectionId;           // -- internal use only --
    int            fHttpRequest;           // get, post, head
    char          *fPathPtr;               // URL
    char          *fHostPtr;               // Host:
    char          *fRefererPtr;            // (at time not supported)
    char          *fAgentPtr;              // (at time not supported)
    char          *fLanguagePtr;           // (at time not supported)
    unsigned long  fBrowserDate;           // Date:  (internal)
    char          *fArgumentBufferPtr;     // Pointer to argument buf
    long          fArgumentBufferLength;  // Length of argument buf
    char          *fUserNamePtr;           // Username from Authorization
    char          *fPasswordPtr;           // Password from Authorization
    long          *fRemoteIPPtr;           // new at V1.00 Beta, points to the remoteIP,
                                           // you must split the octets

    /*******
    //      Response fields  (Set by CGI function)
    /*******
    int            fResponseState;         // -- internal, do not modify --
    int            fHttpResponse;          // Response httpmsg e.g. CgiHttpOk
    int            fDataType;              // Content type, e.g. text/HTML, text/plain
    char          *fResponseBufferPtr;     // Pointer to the created page
    long          fResponseBufferLength;   // Length of the page
    unsigned long  fObjectDate;            // -- internal, do not modify --
    unsigned int   fHostIndex;             // -- internal, do not modify --
} rpCgi, *rpCgiPtr;
```

## Comments

### fDataType

This is an enumeration type (disguised as a 16 bit integer) that specifies the [filetype](#).

## Related Topics

CGI Callback [Function](#)

[Top of list](#)  
[Index page](#)

---



## RTOS API Updates - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### RTOS API News

The following extensions to the RTOS [API](#) are available in the indicated BIOS revisions.

New in version 1.02B: [Disable the task scheduler](#)

New in version 1.02B: [Enable the task scheduler](#)

New in version 1.01B: [Create a task without start](#)

New in version 1.01B: [Get state without task monitoring mode](#)

---

End of document

---

## RTOS Overview - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

RTOS API [News](#)

---

### RTOS Overview

The RTOS [API](#) services split into the following groups:

- [TaskControl Services](#)
- [Semaphoreservices](#)
- [Time/ Date Services](#)
- [TimerProcedures](#)
- [EventManager](#)
- [MessageExchange Manager](#)

---

### Task Control Services

<a href="#">RTX_SLEEP_TIME</a>	Sleep for a specified time
<a href="#">RTX_TASK_CREATE</a>	Create and start a task
<a href="#">RTX_TASK_KILL</a>	Kill a task
<a href="#">RTX_TASK_DELETE</a>	Remove a task from the system
<a href="#">RTX_GET_TASKID</a>	Get ID of the current running task
<a href="#">RTX_SLEEP_REQ</a>	Sleep, until a wake request
<a href="#">RTX_WAKEUP_TASK</a>	Wake a task, which is waiting
<a href="#">RTX_END_EXEC</a>	End execution of a task by itself
<a href="#">RTX_CHANGE_PRIO</a>	Change the priority of a task
<a href="#">RTX_ACCESS_FILESYSTEM</a>	Enable file access for the calling task
<a href="#">RTX_GET_TASK_STATE</a>	Get state of a task (task monitoring)
<a href="#">RTX_GET_TASK_LIST</a>	Get state of a task
<a href="#">RTX_START_TASK_MONITOR</a>	Start task monitoring
<a href="#">RTX_STOP_TASK_MONITOR</a>	Stop task monitoring
<a href="#">RTX_SUSPEND_TASK</a>	Suspend a task
<a href="#">RTX_RESUME_TASK</a>	Resume a task
<a href="#">RTX_RESTART_TASK</a>	Restart a killed task
<a href="#">RTX_TASK_CREATE_WITHOUT_RUN</a>	Create a task
<a href="#">RTX_GET_TASK_STATE_EXT</a>	Get state of a task (without task monitoring)

## Semaphore services

Semaphores are used to guarantee a task mutually exclusive access to a critical resources. Semaphores synchronize asynchronous occurring activities. They are an essential part of a multitasking system. A good description of multitasking systems and semaphores is available in the book "Operating systems" from Andrew Tanenbaum.

The RTOS API provides two types of semaphores:

- A **counting semaphore** is a semaphore with an associated counter, which can be incremented (signal) and decremented (wait). The resource controlled by the semaphore is free (available), when the counter is greater than 0.
- A **resource semaphore** is a counting semaphore with a maximum of count of one. It can be used to provide mutually exclusive access to a single resource. A resource semaphore is also called a binary semaphore. It differs from a counting semaphore in one significant feature: The resource ownership is tied to a specific task. No other task except the task owning the resource is allowed to signal the associated semaphore to release the resource.

The counting and resource semaphores provide automatic timeout. Tasks can specify the maximum time for waiting on a semaphore. The tasks wait in FIFO order for a resource. A semaphore is created with `RTX_CREATE_SEM` service call. The RTOS needs a unique four byte semaphore name and on success returns a unique ID semaphore ID (or handle) to the caller. This ID identifies the semaphore and is needed for other semaphore services.

### Using a counting semaphore:

A counting semaphore is created by specifying an initial count greater or equal to zero in the call of `serviceRTX_CREATE_SEM`. If a semaphore is initialized with a value n, it can be used to control access to n resources, e.g. a counting semaphore with the initial value three assures, that no more than three tasks can own a resource at any one time. Access to a resource controlled by a counting semaphore is acquired with a call to `serviceRTX_WAIT_SEM` or `serviceRTX_GET_SEM`. If the resource is available the RTOS gives it to the task immediately. When the task finished using the resource it signals its release by calling `serviceRTX_SIGNAL_SEM`.

### Using a resource semaphore:

A resource semaphore is created by specifying an initial count of -1 in the call of `serviceRTX_CREATE_SEM`. The RTOS creates a resource semaphore and automatically gives it an initial value of one indicating that the resource is free. A resource is reserved by calling `serviceRTX_RESERVE_RES` using the semaphore ID which was returned by `RTX_CREATE_SEM`. The resource is released with a call of `serviceRTX_RELEASE_SEM`.

### Semaphore Services:

<code>RTX_CREATE_SEM</code>	Create a semaphore
<code>RTX_DELETE_SEM</code>	Delete a semaphore
<code>RTX_FREE_RES</code>	Free a resource semaphore
<code>RTX_GET_SEM</code>	Get use of a counting semaphore(no wait)
<code>RTX_RELEASE_SEM</code>	Release a resource semaphore
<code>RTX_RESERVE_RES</code>	Reserve a resource semaphore
<code>RTX_SIGNAL_SEM</code>	Signal a counting semaphore

[Top of list](#)  
[Index page](#)

---

## Time / Date Services

The following Time/Data services are available.

<a href="#">RTX_GET_TIMEDATE</a>	Get system time and date
<a href="#">RTX_SET_TIMEDATE</a>	Set system time and date
<a href="#">RTX_GET_TICKS</a>	Get tick count of system clock

## Related Topics

TimeDate\_Structure [type](#) definition

[Top of list](#)  
[Index page](#)

---

## Timer Procedures

The RTOS API provides four calls for the usage of timer procedures. The kernel is able to execute periodic user timer procedures at a given time interval. Your timer procedure must be as short as possible without any waiting or endless loops. Avoid the usage of large CLib functions like `printf()`. The maximum number of possible timer procedures is 15.

<a href="#">RTX_INSTALL_TIMER</a>	Install a timer procedure
<a href="#">RTX_REMOVE_TIMER</a>	Remove a timer procedure from the system
<a href="#">RTX_START_TIMER</a>	Start periodic execution of a installed timer procedure
<a href="#">RTX_STOP_TIMER</a>	Stop periodic execution of a timer procedure

[Top of list](#)  
[Index page](#)

---

## Event Manager

The internal RTOS Event Manager provides a convenient mechanism for coordinating tasks waiting for events with tasks and/or timer procedures which can signal the events. The RTOS Event Manager allows more than one task to simultaneously wait for a particular event. Tasks can also wait for a particular combination of events or for any one in a set of events to occur. The Event Manager provides a set of event flags, which can be associated with specific events in your system. These event flags are provided in groups with 16 event flags per group. The system supports a maximum of two event groups.

It could be useful to use the Event Manager when two or more tasks will wait for the same event, e.g. waiting for the start of a motor. An event flag is defined to represent the state of the motor (off or on). When tasks must wait for the motor, they do so by calling the Event Manager requesting a wait until the motor control event flag indicates that the motor is on. When the motor control task or timer procedure detects that the motor is on, it signals the event with a call to the Event Manager. The Event Manager wakes up all tasks which are waiting for the motor to be on. For further explanations read the function description in the API call specifications.

### Event Services:

<a href="#"><u>RTX_CREATE_EVENTGROUP</u></a>	Create an event group
<a href="#"><u>RTX_DELETE_EVENTGROUP</u></a>	Delete an event group
<a href="#"><u>RTX_SIGNAL_EVENTS</u></a>	Signal one or more events in a group
<a href="#"><u>RTX_WAIT_EVENTS</u></a>	Wait for all/any of a set of events in a group
<a href="#"><u>RTX_GET_EVENTGROUP_STATE</u></a>	Read current state of events in a group
<a href="#"><u>RTX_GET_EVENT_FLAGS</u></a>	Get saved event flags
<a href="#"><u>RTX_FIND_EVENTGROUP</u></a>	Find the group ID of an group with a specific name

[Top of list](#)  
[Index page](#)

---

## Message Exchange Manager

The internal RTOS Message Exchange Manager provides a mechanism for interprocess communication and synchronization. In particular, it offers an instant solution to a common producer/consumer problem:

One or more processes (producers) having to asynchronously deliver requests for service to one or more servers (consumers) whose identity is unknown to the producer.

An often cited example of using message exchange is a print request queue. Assume that there are two different server tasks (consumers), each of which is connected to a different printer. There are some other tasks (producers) which want to asynchronously use one of the two servers for printing and they don't care which of the two printers is used. The solution is to synchronize those requests: The producer tasks send their requests (messages) to the Message Exchange Manager. The two server tasks waiting for messages take a message (if any) from the message queue and execute the requested print job.

The internal Message Exchange Manager uses a message exchange to deliver messages. A message exchange consists of four mailboxes into which messages can be deposited. The mailboxes are ordered according to priority (0-3), where mailbox 0 has the highest priority.

Messages are delivered to the mailboxes of the message exchange in message envelopes. The system's maximum number of available messages envelopes is 64. The maximum number of message exchanges is ten. The maximum message length is 12 bytes. (Note: Larger messages can be implemented with a pointer and a length parameter in the message.) Maximum depth of a mailbox is four. Any task or timer procedure can send a message to a Message Exchange. The sender indicates the priority of its message (0-3), thereby identifying the mailbox into which it will be delivered.

Any task or timer procedure can request a message from a Message Exchange, but only tasks are allowed to wait for the arrival of a message, if none is available. A task can specify the priority at which it is willing to wait and the maximum time. A task which uses a Message Exchange must specify the depth of its mailboxes in its `TaskDefBlock` [structure](#). A maximum of four messages can reside in each task

mailbox. Unused mailboxes must have a zero specified in the respective mailbox's `mailboxlevelN` field in the `TaskDefBlock` structure. For further explanations read the function description in the API call specifications.

### Message Exchange Services:

<a href="#"><u>RTX_CREATE_MSG</u></a>	Create a message exchange
<a href="#"><u>RTX_DELETE_MSG</u></a>	Delete a message exchange
<a href="#"><u>RTX_SEND_MSG</u></a>	Send a message to a message exchange
<a href="#"><u>RTX_GET_MSG</u></a>	Get a message, if any (no wait)
<a href="#"><u>RTX_WAIT_MSG</u></a>	Wait for message to arrive (optional timeout)
<a href="#"><u>RTX_FIND_MSG</u></a>	Find a message exchange, specified by name

[Top of list](#)

[Index page](#)

---

End of document

---

## RTOS Error Codes - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### RTOS API Error Codes

All error codes here are stated in decimal.

RTOS error codes returned by RTOS [API](#) calls in the **DX-Register**

0	call successful
-1	RTOS call failed
-2	RTOS API function number (AH input) is not supported

RTOS specific error codes returned by RTOS API calls in the **AX register**

3	task is still waiting
2	task not waiting; wake is pending
1	no buffer available
0	no error
-1	invalid taskid
-2	If DX=-1, no free message available; If DX=-2, Bad API function number
-3	no mailbox defined
-4	mailbox full
-5	wakened before timeout
-6	task not waiting (after 2nd wake)
-7	calling task not waiting
-8	invalid message call
-12	resource not owned by you (caller)
-13	no such buffer pool (invalid id)
-14	not enough memory
-15	memory error
-16	memory error
-17	invalid task priority
-18	no free Task Control Block
-19	no free interval timer
-20	task abort (stop, kill, delete) not allowed
-21	access error
-22	invalid semaphore id

-23 semaphore already in use  
-24 invalid semaphore value

-27 timed out  
-28 no message available  
-29 calling task still waiting

-30 no buffers defined  
-31 memory error

-32 no free event group  
-33 event group in use

-37 memory not available  
-38 invalid memory block  
-39 memory block not in use  
-40 memory block use count overflow

-41 no such message exchange (invalid id)  
-42 no free message exchange  
-43 message exchange in use  
-44 invalid message mailbox size

-45 no free semaphore  
-46 no such event group (invalid id)  
-47 no such timer (invalid id)  
-48 invalid timing interval  
-49 invalid result status  
-50 memory fill exceeds segment limit  
-51 semaphore is busy  
-52 invalid task trap type

---

End of document

---

## RTOS Application Developers Note - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

RTOS [News](#)

---

### RTOS Application Notes

RTOS [API](#) Listing

---

### Developer Notes

The provided services are a subset of the RTOS calls. If it should be necessary, we will add needed functions in the future. The given examples should be used and modified by the API programmer. The API programmer should know something about the basics of programming multitasking applications!!

The number of tasks and semaphores in the system is limited. Since BIOS version 1.00 we set fixed limits for creating tasks, semaphores, eventgroups, timers and message exchanges with the RTOS API:

20 Tasks

30 Semaphores

5 Timers

2 Eventgroups

10 Message exchanges

It's very important to declare large enough task **stack**. The example `taskexp1` shows this problem. When the program is compiled with Microsoft C V1.52, we must set the stack size of the tasks to 3072 words. The same program compiled with Borland 4.52 requires only a stack size of 512 words. It's not advised to use the `printf` functions in a task procedure because it requires a lot of stack space. In the example program `taskexp1.exe`, we use `printf` calls and it works, but there is no guarantee that it will work in other applications. Timer procedures are executed on the stack of the kernel task, so they should be as short as possible. Avoid the calling of large C-Library functions like `printf`.

### Task priorities:

We recommend the usage of a task priority between 20 and 30, because a task with a higher priority is able to block other important tasks of our system. e.g. the serial and Ethernet receiver tasks.

---

End of document

---

## RTOS Examples Available - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

RTOS [News](#)

---

### RTOS Examples

Available RTOS [API](#) examples:

1. `taskexpl.c`      Creating and starting tasks and usage of a resource [semaphore](#)
2. `tcpservm.c`      `tcpechoserver`, which is able to serve a maximum of three clients at the same time
3. `timer.c`          [Timer](#) procedure example for DK40
4. `Pastimer.c`        [Timer](#) procedure example written in Turbo Pascal
5. `event.c`          Example of using [event](#) groups
6. `msg.c`             Example of using [message](#) exchange

The examples are compiled with Borland C 4.5 or 5.02.

Also we build a C-Library (`rtos.c`), which contains the described software interrupt calls.  
All example programs built with C API-functions use the files `rtos.c`, `rtos.h` and `rtxapi.h`.

---

End of document

---

## Data Structures used in RTOS API - BIOS V1.02 Beta

IPC@Chip Documentation [Index](#)

### Data Structures

Here are the data structures used by the RTOS [API](#) .

All constants and data structures are defined in the header file rtxapi.h

Notes:

- **Byte alignment is required** for all data structures used within the API.

Content :

- [typedefRTX\\_Msg](#)
- [typedefRTX\\_Wait\\_Event](#)
- [typedefRTX\\_Wait\\_Msg](#)
- [typedefTaskDefBlock](#)
- [typedefTaskList](#)
- [typedefTask\\_StateData](#)
- [typedefTimeDate\\_Structure](#)
- [typedefTimerProc\\_Structure](#)

### RTX\_Msg

```
typedef struct tag_rtx_msg{
    int msgID; // Unique Message Exchange ID
    char name[4]; // 4 characters, not null terminated
    int mb0; // Numbers of message envelopes which can reside
    int mb1; // in each of the four exchange mailboxes,
    int mb2; // maximum 4 envelopes.
    int mb3;
} RTX_Msg;
```

### Comments

This structure is defined in the header file rtxapi.h.

Prior to making the `RTOS_CREATE_MSG` [API](#) call, the caller must set the following members of the `RTOS_Msg` structure.

#### **name**

Here you can give the Message Exchange a unique four character name. This field is optional. If you plan to use the `RTOS_FIND_MSG` [API](#) then you should provide a name here.

#### **mb0 through mb3**

State here the number of message envelopes for each of this Message Exchange's four mailboxes. The system's maximum number of available messages envelopes is 64 (since BIOS version 1.02B, else 32).

The `msgID` member is an output parameter from the `RTX_CREATE_MSG` API call.

## Related Topics

API function [RTX\\_CREATE\\_MSG](#) - Create a Message Exchange  
RTOS [Message](#) Exchange Manager

[Top of list](#)

[Index page](#)

---

## RTX\_Wait\_Event

```
typedef struct tag_rtx_event_wait
{
    unsigned int mask;    // 16-Bit mask identifying the flags of interest of the group.
    unsigned int value;  // 16 Bit value, which specifies the states of interest
                        // for each flag selected by the mask.
    int match;           // event match requirements, 0:only one flag must match
                        // with value, !=0: all by mask specified flags must match
    long timeout;        // Maximum time (milliseconds) for waiting for an event match
} RTX_Wait_Event;
```

## Comments

This structure is defined in the header file `rtxapi.h`.

## Related Topics

API function [RTX\\_WAIT\\_EVENTS](#) - Wait for events in a group  
RTOS [Event](#) Manager

[Top of list](#)

[Index page](#)

---

## RTX\_Wait\_Msg

```
typedef struct tag_rtx_wait_msg{
    int  msgID;    // ID of the message exchange
    int  prio;     // priority for wait (0-3), 0=highest
    char *msg;     // Pointer to user buffer to store the arrived message
    long timeout; // Maximum time (milliseconds) for waiting for a message
} RTX_Wait_Msg;
```

## Comments

This structure is defined in the header file `rtxapi.h`.

Prior to making the `RTX_WAIT_MSG` [API](#) call, the caller must set the following members of the `RTX_Wait_Msg` structure.

**msgID**

Here you put the message exchange ID acquired by the `RTX_CREATE_MSG` [call](#).

**prio**

Specify here the priority of the calling task's access to the messages. To wait in FIFO order, have all `RTX_WAIT_MSG` API callers use the same value here. A task can cut in line ahead of other waiting tasks by setting

this field to a higher priority (lower number) than used by the other tasks.

#### **msg**

Put a far pointer to a 12 byte buffer allocated in your application program memory. The Message Exchange Manager will output the message to this buffer if a message is available.

#### **timeout**

Here you can specify the maximum number of milliseconds you are willing to wait for a message. A value of zero indicates you will wait forever.

## Related Topics

API function [RTX\\_WAIT\\_MSG](#) - Wait for a message from a message exchange

RTOS [Message](#) Exchange Manager

[Top of list](#)

[Index page](#)

---

## TaskDefBlock

```
typedef struct tag_taskdefblock
{
    void          (*proc)();           // Task entry vector (far)
    char          name[4];             // Task name, 4 characters not null terminated
    unsigned int  *stackptr;           // Task stack pointer (far)
    unsigned int  stacksize;           // size of stack (bytes)
    unsigned short attrib;             // task attributes (not supported by the RTOS API)
    short int     priority;             // task priority, range: 3..127 inclusive
    unsigned short time_slice;         // 0: none, !=0: number of milliseconds before task
                                        // is forced to relinquish processor
    short         mailboxlevel1;       // depth of mailbox i: 0 not used, else 1 - 4 mail box depth
    short         mailboxlevel2;       // maximum number of messages reside inside a task mail box
    short         mailboxlevel3;
    short         mailboxlevel4;
} TaskDefBlock;
```

## Comments

This structure is defined in the header file rtxapi.h.

Prior to making the `RTX_TASK_CREATE` API call, the caller must set the following members of the `TaskDefBlock` structure.

#### **proc**

Here you must set the far vector to your task's entry point. The task's main procedure should be declared (depending on compiler used) as:

Borland C: `void huge taskfunc(void)`

Microsoft C: `void far _saveregs _loadds taskfunc(void)`

to assure that the data segment register (DS) is loaded on entry into your task.

#### **name**

Make up a four letter name for your task by which it can be uniquely identified. Avoid names already occupied by the [system](#) tasks.

#### **stackptr**

This far pointer should point to the top of your task's stack space (highest address). Your task's stack pointer will be initialized to this value, which points to the first byte of memory following your actual stack space. (Note that x86 CPU decrements the stack pointer prior to writing to the stack.) The stack memory space resides within your application program.

#### **stacksize**

Here you specify the size of your task's stack space in bytes. The amount of stack space required for your task depends on the nature of your task. If large automatic objects are declared in your task's procedures, a large amount of stack space will be needed. **Caution:**

Some of the system's interrupts use your task's stack. Consequently we recommend a minimum stack space of 1024 bytes per task.

Since the problems resulting from stack overflow are often difficult to diagnose and analyze, the following design steps are recommended:

1. Initially allocate way more stack space than you believe you will need.
2. When you have your task performing what it was designed to do, measure the amount of stack space being used by your task. The [RTX\\_GET\\_TASK\\_STATE](#) API can be used to obtain this measurement.
3. Refine your stack allocation based on this measurement, arriving at a compromise between the conflicting requirements: efficiency on the one hand (small stack desired) and program maintainability and reliability on the other hand (big stack desired).

Software maintainability becomes an issue here if you have the stack space wired so tight that the slightest code change will lead to stack overflow. Reliability is an issue when paths in your task (or interrupts) are executed that did not execute during your stack space measurement trials.

#### **priority**

Application program tasks can range in priority from 3 to 127 (inclusive), where 3 is higher priority. Task priorities between 20 and 30 are recommended. This recommendation is based on the priority assignments of the [system](#) tasks. In particular, too high a priority for an application task may block important system tasks: e.g. the serial and Ethernet receiver tasks. A user application will be started with priority 25.

#### **time\_slice**

Set this value to zero if you do not want round-robin time slicing between this task and others at the same priority level. If you do want round-robin switching, then specify here the number of milliseconds of CPU time that this task should receive before the system switches to the next task at this same priority. System timing granularity is one millisecond.

#### **mailboxlevelN**

These four values specify whether or not this task is using mailboxes. See the [MessageExchange Manager](#) topic for more details.

## **Related Topics**

API function [RTX\\_TASK\\_CREATE](#) - Create and start a task

[Top of list](#)

[Index page](#)

---

## **TaskList**

```
typedef struct tagtasklist
{
    unsigned int taskID;           // Task handle
    char         taskname[5];      // Four character string terminated with zero.
} TaskList;
```

## **Comments**

This structure is defined in the header file rtxapi.h

## **Related Topics**

API function [RTX\\_GET\\_TASK\\_LIST](#) - Get list of current tasks in the system

[Top of list](#)

[Index page](#)

---

## **Task\_StateData**

```
typedef struct tag_task_statedata
{
```

```
unsigned int taskID;        // Task handle
unsigned int taskPrio;     // Task priority
unsigned int taskState;    // Bit field, see below
unsigned int taskCount;    // count of the task monitor, if task monitor is active
unsigned int stackused;    // Percentage of stack space used
unsigned int stacksize;    // Task's total stack size, in bytes
```

```
} Task_StateData;
```

## Comments

This structure is defined in the header file rtxapi.h.

The `taskState` bit field is coded as follows:

B0: timer wait (used with other bits)

B1: trigger wait (i.e. idle)

B2: semaphore wait

B3: event group wait

B4: message exchange wait

B5: message send wait

B6: suspended (waiting for resume)

B7: waiting for wakeup

B8 - B15 internal use only

## Related Topics

API function [RTX\\_GET\\_TASK\\_STATE](#) - Get state of a task

[Top of list](#)

[Index page](#)

---

## TimeDate\_Structure

```
typedef struct tag_time
{
    unsigned char sec;        // Seconds   (0-59)
    unsigned char min;       // Minutes  (0-59)
    unsigned char hr;        // Hours    (0-23)
    unsigned char dy;        // Day      (1-31)
    unsigned char mn;        // Month    (1-12)
    unsigned char yr;        // Year     (0-99)
    unsigned char dow;       // Day of week (Mon=1 to Sun=7)
    unsigned char dcen;      // Century if time/date is correct
} TimeDate_Structure;
```

## Comments

This structure is defined in the header file rtxapi.h.

## Related Topics

API function [RTX\\_GET\\_TIMEDATE](#) - Get system time and date

API function [RTX\\_SET\\_TIMEDATE](#) - Set system time and date

[Top of list](#)

[Index page](#)

---

## TimerProc\_Structure

```
typedef struct tag_timer_proc
{
    int *timerID;           // pointer to storage the unique timerID
    void (*proc)();        // pointer to the procedure to be executed
    void *dummyptr;       // -- currently not used --
    char name[4];         // unique 4 character task name
    long interval;        // timer execution interval in milliseconds
} TimerProc_Structure;
```

## Comments

This structure is defined in the header file rtxapi.h.

Before calling the [RTX\\_INSTALL\\_TIMER](#) API function the caller must allocate a TimerProc\_Structure with the following members preset as specified here:

### timerID

Put here a far pointer to a 16 bit location in your program's memory space. The RTX\_INSTALL\_TIMERAPI function will output a Timer ID to this referenced location. This Timer ID value is used as handle for this new timer procedure within the other Timer Procedure API [functions](#).

### proc

This is a far vector to your timer procedure. This routine will be called periodically from the kernel. Your timer procedure should be declared (depending on compiler used) as:

Borland C: void huge MyTimerProc(void)

Microsoft C: void far \_saveregs \_loadds MyTimerProc(void)

Turbo Pascal:

```
procedure Timer1_Proc;interrupt;
begin
```

```
[... your code ...]
```

```
(*****
(* This is needed at the end of the Timer Proc. *)
```

```
asm
```

```
POP BP
```

```
POP ES
```

```
POP DS
```

```
POP DI
```

```
POP SI
```

```
POP DX
```

```
POP CX
```

```
POP BX
```

```
POP AX
```

```
RETF
```

```
end;
```

```
(*****
end;
```

so that the compiler will generate code to set the CPU's DS register, enabling access to your program's data.

### name

Place here a four letter name for your timer procedure to uniquely identified it. (This string is **not** zero terminated.)

### interval

Specify here the interval, in milliseconds, at which you want your timer procedure to be called.

## Related Topics

API function [RTX\\_INSTALL\\_TIMER](#) - Install a timer procedure

RTOS [Timer](#) Procedures

[Top of list](#)

[Index page](#)

---

End of document

---

## RTOS Tasks - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

RTOS API [News](#)

---

### IPC@Chip System Tasks

The RTOS itself creates a set of tasks used to support the IPC@Chip services. Your application programs will be executing concurrently with these *built-in* tasks. The fact that these RTOS tasks exists is particularly relevant when you choose priorities for your own application program's tasks.

Each task in the IPC@Chip system must have a unique four letter task name. Consequently the user must take care to avoid the names used by these system tasks when [naming](#) tasks.

### System Task List

The priorities stated here are in decimal, where 0 is highest priority. Some tasks may not be present in your system due to your system configuration specified in the `chip.ini` configuration file. For these tasks the relevant configuration file parameter is stated here with a hyper-link.

AMXK	priority= 0	Kernel task
TCPT	priority= 4	TCP/IP timer task
ETH0	priority= 5	Ethernet receiver task
PPPS	priority= 6	PPP server ( <a href="#">PPPSERVER</a> ENABLE)
PPPC	priority= 6	PPP client task
CFGS	priority= 7	UDP config server
TELN	priority= 11	Telnet server ( <a href="#">TELNET</a> ENABLE)
MTSK	priority= 12	Console task (command shell)
WEBS	priority= 41	Web server ( <a href="#">WEB</a> ENABLE)
FTPS	priority= 41	FTP server ( <a href="#">FTP</a> ENABLE)

### Related Topics

API function [RTX\\_TASK\\_CREATE](#) - Create and start a task

## Hardware API - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

Hardware API [News](#)

---

### Hardware API

Here are the interface definitions for access to the IPC@Chip's hardware.

### Topics

Hardware API [Layer](#) Model

Hardware API [News](#)

### API Functions

The hardware API uses interrupts 0xA2 ([PFE](#) functions) and 0xA1 ([HAL](#) functions) with a service number in the high order byte of the AX register (AH). The implemented hardware services are listed below.

- [Interrupt 0xA2 function 0x80: PFE: Enable Data Bus](#)
- [Interrupt 0xA2 function 0x81: PFE: Enable Non-Multiplexed Address Bus](#)
- [Interrupt 0xA2 function 0x82: PFE: Enable Programmable I/O Pins](#)
- [Interrupt 0xA2 function 0x83: PFE: Enable Programmable Chip Selects](#)
- [Interrupt 0xA2 function 0x84: PFE: Enable External Interrupt Requests](#)
- [Interrupt 0xA2 function 0x85: PFE: Enable External Timer Inputs/Outputs](#)
- [Interrupt 0xA2 function 0x86: PFE: Set Edge/Level Interrupt Mode](#)
- [Interrupt 0xA2 function 0x87: PFE: Enable PWD Mode](#)
- [Interrupt 0xA2 function 0x88: PFE: Enable External DMA](#)
- [Interrupt 0xA2 function 0x89: PFE: Enable INT0/INTA cascade mode](#)
- [Interrupt 0xA1 function 0x10: HAL: Set int0 Vector](#)
- [Interrupt 0xA1 function 0x80: HAL: Read Data Bus](#)
- [Interrupt 0xA1 function 0x81: HAL: Write Data Bus](#)
- [Interrupt 0xA1 function 0x82: HAL: Read Programmable I/O Pins](#)
- [Interrupt 0xA1 function 0x83: HAL: Write Programmable I/O Pins](#)
- [Interrupt 0xA1 function 0x84: HAL: Install Interrupt Service Routine](#)
- [Interrupt 0xA1 function 0x85: HAL: Initialize Timer Settings](#)
- [Interrupt 0xA1 function 0x86: HAL: Start Timer](#)

- [Interrupt 0xA1 function 0x87: HAL: Stop timer](#)
  - [Interrupt 0xA1 function 0x88: HAL: Read Timer Count](#)
  - [Interrupt 0xA1 function 0x89: HAL: Write Timer Count](#)
  - [Interrupt 0xA1 function 0x8A: HAL: Get Frequencies](#)
  - [Interrupt 0xA1 function 0x8B: HAL: Set Timer Duty Cycle Waveform](#)
  - [Interrupt 0xA1 function 0x8C: HAL: Read Specific I/O Pin](#)
  - [Interrupt 0xA1 function 0x8D: HAL: Write to Specific I/O Pin](#)
  - [Interrupt 0xA1 function 0x8E: HAL: Give EOI](#)
  - [Interrupt 0xA1 function 0x90: HAL: Get/Set Watchdog Mode](#)
  - [Interrupt 0xA1 function 0x91: HAL: Refresh Watchdog](#)
  - [Interrupt 0xA1 function 0xA0: HAL: Block Read Data Bus](#)
  - [Interrupt 0xA1 function 0xA1: HAL: Block Write Data Bus](#)
  - [Interrupt 0xA1 function 0xB0: HAL: Start DMA Mode](#)
  - [Interrupt 0xA1 function 0xB1: HAL: Stop DMA Transfer](#)
  - [Interrupt 0xA1 function 0xB2: HAL: Get DMA Info](#)
  - [Interrupt 0xA1 function 0xC0: HAL: Initialize/Restore Non-Volatile Data](#)
  - [Interrupt 0xA1 function 0xC1: HAL: Save Non-Volatile Data](#)
  - [Interrupt 0xA1 function 0xC2: HAL: Get Reboot Reason](#)
- 

## Interrupt 0xA2 service 0x80: PFE: Enable Data Bus

Initialize data bus I/O mask and ALE usage. The I/O mask defines which data bits on the bus are inputs and which are outputs. The DX mask bit for bi-directional data bus lines (read/write) should be set to '1'.

### Parameters

*AH*

Must be 0x80.

*AL*

0: Disable ALE, 1: Enable ALE

*DX*

Mask

Bit 0 = 0: Data bus bit 0 is input, 1: is output

Bit 1 = 0: Data bus bit 1 is input, 1: is output

:

:

Bit 7 = 0: Data bus bit 7 is input, 1: is output

Bit 8..15 not used (for future extensions)

### Return Value

none

### Comments

used pins:

ALE, AD[0..7], RD#, WR#

excluded pins:

if ALE is used, then PCS0# is not available.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA2 service 0x81: PFE: Enable Non-Multiplexed Address Bus

The IPC@Chip has three non-multiplexed address bit outputs, A0 through A2. The enabling of these pins is done here.

### Parameters

*AH*

Must be 0x81.

*DX*

Mask

Bit 0 = 1 Enable A0

Bit 1 = 1 Enable A1

Bit 2 = 1 Enable A2

Bit 3..15 not used

### Return Value

none

### Comments

used pins:

A[0..2], AD[0..7], RD#, WR#

excluded pins:

If A0 is enabled then PCS1#, TMRIN0, PIO4 are not available

If A1 is enabled then PCS[5..6]#, TMRIN1, TMROUT1, PIO3 are not available

If A2 is enabled then PCS[5..6]#, PIO2 are not available.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA2 service 0x82: PFE: Enable Programmable I/O Pins

Enable used programmable I/O pins. Define which pins are inputs and which are outputs. This function can be called several times for definition of different PIO pins. With repeated selection of the same pin, the definition made last is valid. The selection of a PIO pin can be cancelled by calling the appropriate PFE function that causes the respective PIO pin to be used for another purpose (e.g. function [0x83](#) for

PIO[2..6] pins).

## Parameters

*AH*

Must be 0x82.

*AL*

Mode

- 0 = Only read PIO state
- 1 = Input without pullup/pulldown
- 2 = Input with pullup (not PIO13)
- 3 = Input with pulldown (only for PIO3 and PIO13)
- 4 = Output init value = High
- 5 = Output init value = Low

*DX*

PIO pin

- Bit 0 = 1 Enable PIO0
- Bit 1 = 1 Enable PIO1
- :
- :
- Bit 13 = 1 Enable PIO13
- Bit 14..15 not used (for future extensions)

## Return Value

*AX* = wPIO

- Bit 0 = 1: PIO0 is PIO
- Bit 1 = 1: PIO1 is PIO
- :
- :
- Bit 13 = 1: PIO13 is PIO
- Bit 14..15 not used (for future extensions)

*DX* = wINPUTS (all pins, including non-PIO pins)

- Bit 0 = 1: PIO0 is input
- Bit 1 = 1: PIO1 is input

:

- Bit 13 = 1: PIO13 is input
- Bit 14..15 not used (for future extensions)

*CX* = wOUTPUTS (all pins, including non-PIO pins)

- Bit 0 = 1: PIO0 is output
- Bit 1 = 1: PIO1 is output

:

- Bit 13 = 0: PIO13 is output
- Bit 14..15 not used (for future extensions)

*AX* = *DX* = *CX* = 0, Error: Wrong arguments

## Comments

used pins:

PIO[0..13]  
excluded pins:  
All other functionality on the selected PIO pin.

## Related Topics

[Read](#) Specific I/O Pin  
[Write](#) to Specific I/O Pin  
[Read](#) Programmable I/O Pins  
[Write](#) Programmable I/O Pins  
Initialize the [I2C](#) Bus

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA2 service 0x83: PFE: Enable Programmable Chip Selects

Enable chip selects PCS[0..3]#, PCS[5..6]#.

### Parameters

*AH*

Must be 0x83.

*DX*

Mask

Bit 0 = 1 Enable PCS0#, active when I/O address between 000h..0FFh  
Bit 1 = 1 Enable PCS1#, active when I/O address between 100h..1FFh  
Bit 2 = 1 Enable PCS2#, active when I/O address between 200h..2FFh  
Bit 3 = 1 Enable PCS3#, active when I/O address between 300h..3FFh  
Bit 4 = don't care  
Bit 5 = 1 Enable PCS5#, active when I/O address between 500h..5FFh  
Bit 6 = 1 Enable PCS6#, active when I/O address between 600h..6FFh  
Bit 7..15 = don't care

### Return Value

none

### Comments

used pins:

PCS[0..3]#, PCS[5..6]#

excluded pins:

if PCS0#: ALE (multiplexed address / data bus)

if PCS1#: A0, PIO4, TMRIN0, SPI

if PCS2#: PIO6, INT2, INTA#, PWD, SPI, hw flow control serial port 1,  
cascaded interrupt controller

if PCS3#: PIO5, INT4, SPI, hw flow control serial port 1

if PCS5#: A[1..2], PIO3, TMROUT1, TMRIN1  
if PCS6#: A[1..2], PIO2

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA2 service 0x84: PFE: Enable External Interrupt Requests

Enable external interrupt requests INT[0], INT[2..6].

### Parameters

*AH*

Must be 0x84.

*DX*

Mask

Bit 0 = 1 Enable INT0  
Bit 1 = don't care  
Bit 2 = 1 Enable INT2  
Bit 3 = 1 Enable INT3  
Bit 4 = 1 Enable INT4  
Bit 5 = 1 Enable INT5  
Bit 6 = 1 Enable INT6  
Bit 7..15 = don't care

### Return Value

none

### Comments

used pins:

INT0, INT[2..6]

excluded pins:

if INT0: PIO13, TMROUT0, cascaded interrupt controller  
if INT2: PIO6, PCS2#, INTA#, PWD, SPI, hw flow control serial port 1  
if INT3: PIO12, serial port 1  
if INT4: PIO5, PCS3#, SPI, hw flow control serial port 1  
if INT5: PIO1, DRQ0, default I<sup>2</sup>C-Bus pins, SPI  
if INT6: PIO0, DRQ1, default I<sup>2</sup>C-Bus pins, SPI

### Related Topics

Set [Edge/Level](#) Interrupt Mode

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA2 service 0x85: PFE: Enable External Timer Inputs/Outputs

Enable external timer inputs (TMRIN0 , TMRIN1 ) or timer outputs (TMROUT0 , TMROUT1 ). If on a given timer the external input is selected, then that timer's external output is not available and vice-versa.

### Parameters

*AH*  
Must be 0x85.

*DX*  
Mode  
Bit 0..1 = 10 Enable TMRIN0  
          = 11 Enable TMROUT0  
Bit 2..3 = 10 Enable TMRIN1  
          = 11 Enable TMROUT1  
Bit 4..15 = don't care

### Return Value

none

### Comments

used pins:  
  TMRIN[0..1], TMROUT[0..1]  
excluded pins:  
  if TMRIN0: A0, PCS1#, PIO4, TMROUT0  
  if TMRIN1: A[1..2], PCS5#, TMROUT1  
  if TMROUT0: PIO13, INT0, cascaded interrupt controller, TMRIN0  
  if TMROUT1: A[1..2], PCS5#, TMRIN1, PIO3

### Related Topics

HAL [Initialize](#) Timer Settings

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA2 service 0x86: PFE: Set Edge/Level Interrupt Mode

Set edge/level interrupt mode for INT0, INT2, INT3, INT4.

### Parameters

*AH*

Must be 0x86.

*AL*

1 = active high, level-sensitive interrupt  
0 = low-to-high, edge-triggered interrupt

*DX*

Mask, bits set to designate interrupts affected:

Bit 0 = INT0

Bit 1 = don't care

Bit 2 = INT2

Bit 3 = INT3

Bit 4 = INT4

Bit 5..15 = don't care

## Return Value

none

## Comments

Default for all interrupts is edge-triggered mode. In each case (edge or level) the interrupt pins must remain high until they are acknowledged.

Mode for INT5/INT6 is not supported.

## Related Topics

[Enable External](#) Interrupt Requests

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA2 service 0x87: PFE: Enable PWD Mode

Enable Pulse Width Demodulation (PWD)

## Parameters

*AH*

Must be 0x87.

## Return Value

none

## Comments

In PWD mode, TMRIN0, TMRIN1, INT2 and INT4 are configured internal to the chip to support the detection of rising (INT2) and falling (INT4) edges on the PWD input pin and to enable either timer0 when the signal is high or timer1 when the signal is low. The INT4, TMRIN0 and TMRIN1 pins are not used in PWD mode and so are available for use as PIO's.

The ISR for the INT2 and the INT4 interrupts should examine the current count of the associated timer, timer1 for INT2 and timer0 for INT4, in order to determinate the pulse width. The ISR should then reset the timer count in preparation for the next pulse.

Overflow conditions, where the pulse width is greater than the maximum count of the timer, can be detected by monitoring the MaxCount bit in the associated timer or by setting the timer to generated interrupt requests.

used pins:

PWD

excluded pins:

TMRIN0, TMRIN1, TMROUT0, TMROUT1, INT4, INT2

PCS2#, INTA#, PIO6, hw flow control serial port 1

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA2 service 0x88: PFE: Enable External DMA

Enables DRQ pin to start DMA transfer

### Parameters

*AH*

Must be 0x88.

*AL*

DRQ channel:

0 = DRQ0

1 = DRQ1

### Return Value

AX = 0 no error

AX = -1 invalid DRQ channel

AX = -2 DMA channel is used for serial interface

### Comments

For using external DMA you have to disable the serial DMA mode. Do this with an [CHIP.INI entry](#) . COM uses DRQ1, EXT uses DRQ0.

used pins:

DRQ[0..1]

excluded pins:

if DRQ0: PIO1, INT5, default I<sup>2</sup>C-Bus pins, SPI

if DRQ1: PIO0, INT6, default I<sup>2</sup>C-Bus pins, SPI

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA2 service 0x89: PFE: Enable INT0/INTA cascade mode

Enable INT0/INTA cascade mode

### Parameters

*AH*  
Must be 0x89.

### Return Value

none

### Comments

To install a service interrupt routine in cascade mode, the HAL func. "Install Interrupt Service Routine" can not be used, because the cascaded interrupt controller supply the interrupt type over the bus. Install a normal interrupt routine with setvect for the vector you use and at the end of this function give an EOI to the cascaded controller and to the [internal interrupt controller](#) (INT0).

used pins:

INT0, INTA#

excluded pins:

PIO13, TMROUT0, INT0 in normal mode, PIO6, PCS2#, PWD, SPI, hw flow control serial port 1

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA1 service 0x10: HAL: Set int0 Vector

Define the interrupt handler for the hardware interrupt 0

### Parameters

*AH*  
Must be 0x10

*ES:DX*  
Pointer to your interrupt handler

## Return Value

none

## Comments

The interrupt handler should be defined as

```
void interrupt my_handler(void)
```

Interrupts are enabled upon entry into your routine.

There is no need to signal any end of interrupt. This is handled by the system when your handler performs its return.

The stack size must be at least 400 bytes.

Do not use any floating point arithmetic in your interrupt service routine.

---

## Developer Notes

This function is only for compatibility to older version of the hardware API.

Please use interrupt [0xA2](#) function 0x84 and interrupt [0xA1](#) function 0x84 instead.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x80: HAL: Read Data Bus

Read from specified address. The result is combined with wAND and wXOR parameters. To read the data bus without change, set wAND=0xFFFF and wXOR=0x0000.

## Parameters

*AH*

Must be 0x80.

*DI*

Address

*BX*

wAND

*CX*

wXOR

## Return Value

8 Bit data in ax,  $ax = (databus \& wAND) \wedge wXOR$

## Comments

& = bit wise AND

^ = bit wise XOR

## Related Topics

Block [Read](#) Data Bus

[Write](#) Data Bus

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x81: HAL: Write Data Bus

Write to specified address. The provided parameters are combined as follows to form the output byte value:

$$\text{output value} = (\text{data} \ \& \ \text{wAND}) \ \wedge \ \text{wXOR}$$

To write the value in DL to the address without modification, set wAND=0xFFFF and wXOR=0x0000.

## Parameters

*AH*

Must be 0x81.

*DI*

Address

*DL*

8 bit data

*BX*

wAND

*CX*

wXOR

## Return Value

none

## Comments

& = bit wise AND

^ = bit wise XOR

## Related Topics

Block [Write](#) Data Bus

[Read](#) Data Bus

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x82: HAL: Read Programmable I/O Pins

Read the programmable I/O pins. The result is combined with the wAND and wXOR parameters. To read the PIO pins without modification, set wAND=0xFFFF and wXOR=0x0000. To read only the input pins, set wAND = wPIO & wINPUTS. The [wPIO](#) and [wINPUTS](#) values are return values from PFE [function](#) 0x82.

### Parameters

*AH*  
Must be 0x82.

*BX*  
wAND

*CX*  
wXOR

### Return Value

$ax = (PIO[0..13] \& wAND) \wedge wXOR.$

### Comments

& = bit wise AND

^ = bit wise XOR

### Related Topics

Read [Specific](#) I/O Pin

[Write](#) Programmable I/O Pins

PFE: [Enable](#) Programmable I/O Pins

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x83: HAL: Write Programmable I/O Pins

Write to the programmable I/O pins. Before the value is written, the value is combined with the wAND and wXOR parameters as:

$$\text{PIO}[0..13] = (\text{data} \& \text{wAND}) \wedge \text{wXOR}$$

To write value in DX to the programmable I/O pins without change, set wAND=0xFFFF and wXOR=0x0000.

Only PIO pins that are defined as outputs can be written.

## Parameters

*AH*

Must be 0x83.

*DX*

data applied to PIO[0..13]  
where DX bit 0 maps to PIO[0]  
and DX bit 13 maps to PIO[13]

*BX*

wAND

*CX*

wXOR

## Return Value

none

## Comments

& = bit wise AND

^ = bit wise XOR

## Related Topics

Write [Specific](#) I/O Pin

[Read](#) Programmable I/O Pins

PFE: [Enable](#) Programmable I/O Pins

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x84: HAL: Install Interrupt Service Routine

Install user interrupt service routine to be invoked by system interrupt handler.

## Parameters

*AH*

Must be 0x84.

*DX*

HAL interrupt number from following list:

- 0 = INT0 (external)
  - 1 = Network controller (internal) (\*)
  - 2 = INT2 (external)
  - 3 = INT3 (external)
  - 4 = INT4 (external)
  - 5 = INT5 (external) / Terminal Count DMA Channel 0 (if DMA is used)
  - 6 = INT6 (external) / Terminal Count DMA Channel 1 (if DMA is used)
  - 7 = reserved
  - 8 = Timer0 (internal)
  - 9 = Timer1 (internal)
  - 10 = Timer 1ms (internal) (\*)
  - 11 = Serial port 0 (internal) (\*)
  - 12 = Serial port 1 (internal) (\*)
  - 13 = reserved
  - 14 = reserved
  - 15 = NMI (internal/external)
- (\* = currently not supported)

*CX*

Number of interrupts generated before new user interrupt service routine is called.  
CX = 0 disables the user ISR (same as a NULL in ES:BX).

*ES:BX*

far pointer to user interrupt service routine  
if pointer is NULL user ISR is disabled

## Return Value

Far pointer to old handler in ES:BX

## Comments

The user-defined ISR is called from a system ISR with the interrupt identifier number in the BX CPU register, thus allowing for a single user ISR to handle multiple interrupt sources. The user ISR must be terminated with an IRET instruction and may not change any CPU registers except for AX and BX.

Any required EOI signal is issued by the system ISR that called your user ISR.

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x85: HAL: Initialize Timer Settings

Initialize the timer settings of timer0 or timer1.

## Parameters

*AH*

Must be 0x85.

*AL*

Timer

0=Timer0 / 1=Timer1

*DX*

Mode

Bit 0: 0=run single time / 1=run continuous

Bit 1: 0=disable timer interrupt / 1=enable timer interrupt

Bit 2: 0=use internal clock / 1=use TMRIN pin as external clock

Bit 3..15: not used

*CX*

Clock divider (maximum count value)

## Return Value

none

## Comments

The clock divider value serves as a comparator for the associated timer count. The timer count is a 16 bit value that is incremented by the processor internal clock (see HAL [function](#) 0x8A) or can also be configured to increment based on the TMRIN0 or TMRIN1 external signals (see PFE [function](#) 0x85). The TMROUT0 und TMROUT1 signals can be used to generate waveforms of various duty cycles. The default is a 50% duty cycle waveform (Change waveform with HAL [function](#) 0x8B).

Note that TMRIN pin and TMROUT pin can not be used at the same time.

If the clock divider value is set to 0000h, the timer will count from 0000h to FFFFh (maximum divider). When the timer reaches the clock divider value, it resets to 0 during the same clock cycle. The timer count never dwells equals to the clock divider value (except for special case when divider value is set to 0000h).

When the timer is configured to run in single time mode, the timer clears the count and then halts on reaching the maximum count (clock divider value).

If the timer interrupt is enabled, the interrupt request is generated when the count equals the clock divider value. Use HAL [function](#) 0x84 to install your interrupt service routine.

## Related Topics

HAL [Start](#) Timer function

[Read](#) Timer Count

[Write](#) Timer Count

---

## Developer Notes

The timer output frequency is dependent on the internal processor clock.  
For compatibility with future versions of @Chip, please use the HAL [function](#) 0x8A, "Get frequencies", to compute the correct clock divider value.

Available examples

1. TimerIn example, timerin.c
2. TimerOut example, timerout.c

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x86: HAL: Start Timer

Enable the specified timer to count.

### Parameters

*AH*

Must be 0x86.

*AL*

Timer  
0=Timer0 / 1=Timer1

### Return Value

none

### Related Topics

HAL [Initialize](#) Timer Settings  
[Stop](#) Timer function

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x87: HAL: Stop timer

Stop the specified timer's counting

### Parameters

*AH*

Must be 0x87.

*AL*

Timer

0=Timer0 / 1=Timer1

## Return Value

none

## Comments

The specified timer is disabled.

## Related Topics

[Start](#) Timer function

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x88: HAL: Read Timer Count

Read the timer count.

## Parameters

*AH*

Must be 0x88.

*AL*

Timer

0=Timer0 / 1=Timer1

## Return Value

AX = Counter reading

DX = 1=MaxCount reached / 0=MaxCount not reached

## Comments

AX contains the current count of the associated timer. The count is incremented by the processor internal clock (see HAL [function](#) 0x8A), unless the timer is configured for external clocking (then it is clocked by the TMRIN0 and TMRIN1 [signals](#) ).

## Related Topics

HAL [Initialize](#) Timer Settings

[Stop](#) Timer function

[Write](#) Timer Count

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x89: HAL: Write Timer Count

Preset the specified timer's count register to provided value.

### Parameters

*AH*

Must be 0x89.

*AL*

Timer

0=Timer0 / 1=Timer1

*DX*

Value to write to 16 bit counter

### Return Value

none

### Comments

The timer count can be written at any time, regardless of whether the corresponding timer is running.

### Related Topics

HAL [Initialize](#) Timer Settings

[Stop](#) Timer function

[Read](#) Timer Count

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x8A: HAL: Get Frequencies

Get the system frequencies.

## Parameters

*AH*

Must be 0x8A.

*AL*

Which frequency to get:

- 0 = Return processor frequency
- 1 = Return timer base frequency
- 2 = Return maximum baud rate
- 3 = Return PWD timer frequency

## Return Value

DX:AX frequency [Hz]

## Comments

Use the timer base frequency to compute the correct timer clock divider value, where:

$$\text{Output frequency} = \text{timer base frequency} / \text{clock divider}$$

Use the maximum baud rate compute the correct baud rate for the processor specific baud rate initialize function (See Fossil *Extended line control initialization* [function](#) ).

$$\text{Baud rate} = \text{maximum baud rate} / \text{baud rate divider}$$

## Related Topics

HAL [Initialize](#) Timer Settings

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x8B: HAL: Set Timer Duty Cycle Waveform

Set the duty cycle waveform of specified timer.

## Parameters

*AH*

Must be 0x8B.

*AL*

Which Timer

0=Timer0 / 1=Timer 1

*DX*

Mode

0=disable duty cycle / 1=enable duty cycle

CX

Alternate clock divider (if DX = 1)

## Return Value

none

## Comments

Use this function to modify the timer waveform behavior. For example a 50% duty cycle waveform can be generated by specifying here an *alternate clock divider* value in CX that is the same value as was used for the main *clock divider* value set in the Timer Initialization [function](#) call.

Please note that the timer frequency will change if you use this function. If you disable the duty cycle, the timer output will no longer generate a rectangle signal. When duty cycle mode is disabled, the TMROUT pin switches low for only one clock cycle after the maximum count is reached.

## Related Topics

HAL [Initialize](#) Timer Settings

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x8C: HAL: Read Specific I/O Pin

Read specified programmable I/O pin.

## Parameters

AH

Must be 0x8C.

AL

SC12 PIO No. [0..13]

## Return Value

AX=0 PIO pin is low, AX!=0 PIO pin is high

## Related Topics

[Write](#) to Specific I/O Pin

[Read](#) Programmable I/O Pins

PFE: [Enable](#) Programmable I/O Pins

## Interrupt 0xA1 service 0x8D: HAL: Write to Specific I/O Pin

Write to a specified programmable I/O pin. Only PIO pins that are defined as outputs can be written.

### Parameters

*AH*  
Must be 0x8D.

*AL*  
SC12 PIO No. [0..13]

*DX*  
DX = 0 ==> set PIO to low  
DX non-zero ==> set PIO to high

### Return Value

none

### Related Topics

[Read](#) Specific I/O Pin

[Write](#) Programmable I/O Pins

PFE: [Enable](#) Programmable I/O Pins

## Interrupt 0xA1 service 0x8E: HAL: Give EOI

Give End-Of-Interrupt for INT0-INT4

### Parameters

*AH*  
Must be 0x8E.

*AL*  
Type (0=INT0 ... 4=INT4)

### Return Value

none

## Comments

When installing a interrupt service routine through HW API, it's not necessary to call this function, because the HW API does it for you. This function is provided for writing own interrupt service routines without the HAL func. [Install Interrupt Service Routine](#) .

Especially when using cascaded mode of the interrupt controller with INT0/INTA, this function is needed for generating an EOI for INT0.

## Related Topics

[Enable INT0/INTA](#) cascade mode

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0x90: HAL: Get/Set Watchdog Mode

Get or set the watchdog mode.

## Parameters

*AH*

Must be 0x90.

*AL*

Mode

0 = only get mode

2 = Watchdog will be triggered by user program

3 = Watchdog will be triggered by BIOS (default)

## Return Value

AX=mode

## Comments

The watchdog timeout period is 800 ms.

If you select the *User Program* mode, you must call the HAL [Refresh](#) Watchdog function 0x91 at least every 800 ms to prevent the watchdog from resetting the system. In BIOS mode, the BIOS performs the watchdog strobing provided that the system's timer interrupt is allowed to execute. Beware that excessive interrupt masking periods can lead to system resets.

## Related Topics

## Interrupt 0xA1 service 0x91: HAL: Refresh Watchdog

Strobe the hardware watchdog to reset its 800 ms timeout period.

### Parameters

*AH*  
Must be 0x91.

### Return Value

none

### Comments

If the watchdog is in *User Program* mode, this function must be called at least every 800 ms to prevent a CPU reset due to watchdog timeout.

### Related Topics

Get/Set [Watchdog](#) Mode

## Interrupt 0xA1 service 0xA0: HAL: Block Read Data Bus

Read block of bytes from data bus into provided buffer.

### Parameters

*AH*  
Must be 0xA0.

*DI*  
First address

*SI*  
Second address

*ES:BX*  
Pointer to buffer

*CX*  
Number of bytes to read into buffer

## Return Value

none

## Comments

IF  $SI \neq DI$ , this function will alternate reads between the two addresses until  $CX$  bytes are read, starting at first address. Set  $SI == DI$  if you want to read from only a single address.

## Related Topics

Block [Write](#) Data Bus

[Read](#) Data Bus

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0xA1: HAL: Block Write Data Bus

Output byte stream from buffer to specified address or addresses.

## Parameters

*AH*  
Must be 0xA1.

*DI*  
First address

*SI*  
Second address

*ES:BX*  
Pointer to buffer

*CX*  
Number of bytes in buffer to output

## Return Value

none

## Comments

IF SI != DI, this function will alternate between writes to first and second address. Set SI == DI if you want all data written to a single address.

## Related Topics

Block [Read](#) Data Bus

[Write](#) Data Bus

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0xB0: HAL: Start DMA Mode

Starts the DMA mode. After calling this function, the DMA transfer will be started once the external DRQ pins will be activated.

## Parameters

*AH*

Must be 0xB0

*AL*

DMA channel:

0 = DRQ0

1 = DRQ1

*CX*

Number of bytes which has to be tranfered

*DX*

Control Register:

Bit 0: 1=Priority for the channel / 0=Priority for the other channel

Bit 1: 1=Source synchronized / 0=No source synchronization

Bit 2: 1=Destination synchronized / 0=No destination synchronization

Bit 3: 1=Use interrupt at end of transfer / 0=do not use an interrupt

Bit 4: must be set to '1'

Bit 5: 1=Source address increment / 0=No increment of source address

Bit 6: 1=Source address decrement / 0=No decrement of source Address

Bit 7: 1=Source is in memory space / 0=Source is in IO space

Bit 8: 1=Destination address increment / 0=No increment of destination address

Bit 9: 1=Destination address decrement / 0=No decrement of destination address

Bit 10: 1=Destination is in memory space / 0=Destination is in IO space

Bit 11: must be set to '0'

Bit1 and Bit2 can't be set at the same time.

*BX:SI*

Pointer to unsigned long (20-Bit physical) source address

*ES:DI*

Pointer to unsigned long (20-Bit physical) destination address

## Return Value

Success: AX = 0

AX = -1 Invalid DMA channel

AX = -2 DMA channel used for serial interface

## Comments

This function starts a DMA transfer. After calling this function the DMA controller is ready for transfer. Once the DRQ pin will be activated the DMA transfer will be started. The number of bytes which will be transferred has to be put in the CX register. Before calling the function you have to enable the DRQ pin for this channel (and with that the serial DMA mode must be disabled in the CHIP.INI).

## Related Topics

[Enable](#) external DRQ Pin

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0xB1: HAL: Stop DMA Transfer

Disables the DMA controller. A running DMA transfer would be halted.

## Parameters

*AH*

Must be 0xB1

*AL*

DMA channel:

0 = DRQ0

1 = DRQ1

## Return Value

Success: AX = 0

AX = -1 Invalid DMA channel

AX = -2 DMA channel used for serial interface

## Comments

Stops a DMA transfer (disables the DMA controller). The transfer could be continued by calling [Continue DMA transfer](#)

## Related Topics

[Start DMA Transfer](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0xB2: HAL: Get DMA Info

Get the state of the DMA channel

### Parameters

*AH*

Must be 0xB2

*AL*

DMA channel:

0 = DRQ0

1 = DRQ1

### Return Value

AX = 0 DMA channel disabled

AX = 1 DMA channel (user mode) enabled for transfer

AX = -1 Invalid DMA channel

AX = -2 DMA channel used for serial interface

CX = DMA counter (bytes which have to be transferred)

DX = Control Register: Bit 0: 1=Priority for the channel / 0=Priority for the other channel

Bit 1: 1=Source synchronized / 0=No source synchronization

Bit 2: 1=Destination synchronized / 0=No destination synchronization

Bit 3: 1=Use interrupt at end of transfer / 0=do not use an interrupt

Bit 4: must be set to '1'

Bit 5: 1=Source address increment / 0=No increment of source address

Bit 6: 1=Source address decrement / 0=No decrement of source Address

Bit 7: 1=Source is in memory space / 0=Source is in IO space

Bit 8: 1=Destination address increment / 0=No increment of destination address

Bit 9: 1=Destination address decrement / 0=No decrement of destination address

Bit 10: 1=Destination is in memory space / 0=Destination is in IO space

BX:SI = Pointer to unsigned long (20-Bit physical) source address

ES:DI = Pointer to unsigned long (20-Bit physical) destination address

### Comments

This function returns the status of the DMA channel in AX.

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xA1 service 0xC0: HAL: Initialize/Restore Non-Volatile Data

Initialize/Restore non-volatile data. Tell the BIOS where your variables are located, which should be saved and reload their saved values, if available.

The non-volatile (remanent) data is stored in A:\rema.bin file.

### Parameters

*AH*

Must be 0xC0

*ES:BX*

Pointer to a `_REMOP` structure:

```
struct _REMOP
{
    unsigned entries; // number of entries in struct REMOP_ENTRY x[]
    unsigned segment; // common segment address

    struct REMOP_ENTRY
    {
        unsigned offs; // offset address
        unsigned size; // number of bytes (must be <= maxsize)
        unsigned maxsize; // maximal number of bytes
        unsigned elemsize; // number of bytes per data element
        unsigned distance; // distance to next data element (must be >=
elemsize)
    }x[MAX_RETENTIVE_AREAS];
};
```

### Return Value

Success: AX = 0

Failure: AX < 0, Could not create file

### Comments

Call this function at the beginning of your program.

The `_REMOP` structure reference by `ES:BX` must be static. (This function does not make a copy of the structure's content.)

### Related Topics

[Save](#) Non-Volatile Data

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0xC1: HAL: Save Non-Volatile Data

This function save your non-volatile data listed in the `_REMOP` structure registered with HAL [function](#) 0xC0. The data is stored in `A:\rema.bin` file.

### Parameters

*AH*

Must be 0xC1

### Return Value

none

### Comments

Call this function on exit from your program and in your NMI (Non-Maskable Interrupt) handler. Your hardware around the IPC@Chip must support the Pfail signal, so that the IPC@Chip can generate an NMI sufficiently in advance of power loss to the CPU.

**Reminder** : The DK40 does not support the Pfail signal.

### Related Topics

[Initialize/Restore](#) Non-Volatile Data

[Top of list](#)

[Index page](#)

---

## Interrupt 0xA1 service 0xC2: HAL: Get Reboot Reason

Check cause of most recent reboot.

### Parameters

*AH*

Must be 0xC2

### Return Value

AX = reason:

0 = UNKNOWN  
3 = WATCHDOG  
4 = POWER FAIL

## Comments

This function only returns valid results if the init/restore [function](#) (0xC0) was called following the reboot.

## Related Topics

[Initialize/Restore](#) Non-Volatile Data

[Top of list](#)

[Index page](#)

---

End of document

---

## Hardware API Updates - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### Hardware API Updates

The following extensions to the hardware [API](#) are available in the indicated BIOS revisions.

- New in version 1.02B: [New PFE function to enable INT0/INTA cascade mode](#)
- New in version 1.02B: [New HAL function to give EOI \(for cascade mode\)](#)
- New in version 1.02B: [New PFE function to enable DRQ pins](#)
- New in version 1.02B: [New HAL functions for external DMA transfers](#)
- New in version 1.00: [Changed data bus read/write functions, A1/A2 is not switched anymore](#)
- New in version 0.70: [New PFE function for PWD mode](#)
- New in version 0.70: [New HAL read/write functions for specific programmable I/O pins](#)
- New in version 0.70: [New HAL read/write functions for data bus with enabling/disabling A1/A2](#)
- New in version 0.70: [Changed watchdog mode function / disable no longer available](#)
- New in version 0.68: [New PFE function for edge/level interrupt mode](#)
- New in version 0.68: [New HAL functions for retentive operators](#)
- New in version 0.67: [New HAL functions for watchdog](#)
- New in version 0.65: [New PFE function for timers](#)
- New in version 0.65: [New HAL functions for timers](#)

---

End of document

---

## Hardware API Layers - BIOS V1.02 Beta

---

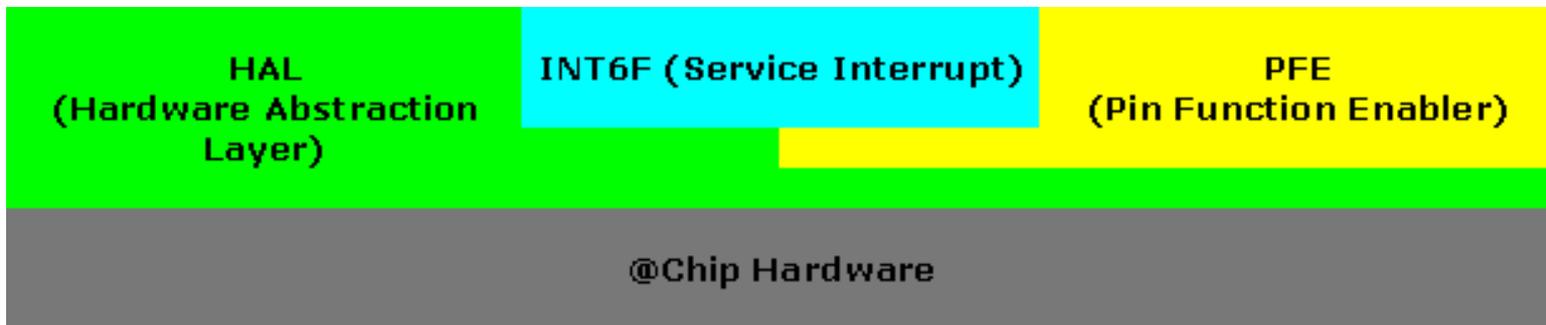
IPC@Chip Documentation [Index](#)

Hardware API [News](#)

---

### Hardware API Layers

Layer model:



#### 1. INT 6Fh - Service Interrupt Handler

The service interrupt provides software compatibility with the Beck FEC product line. This interrupt is used only in Beck products based on the IPC@Chip and is not part of the standard BIOS. These services use the PFE and HAL interfaces to contact the actual hardware.

#### 2. PFE - Pin Function Enabler

This part of the hardware [API](#) provides functions to control the IPC@Chip's multi-function I/O pins.

#### 3. HAL - Hardware Abstraction Layer

This part of the hardware API provide an isolation layer between your application software and the IPC@Chip hardware (PIO pins, timers, etc.) which minizes hardware dependancies.

---

End of document

---

## I2C Bus Interface - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

I2C Bus API [News](#)

---

### I2C Bus API

Here are the interface definitions for access to the IPC@Chip's I2C bus.

- [I2C Bus API News](#)

#### API Functions

The I2C Bus API provides interrupt 0xAA with a service number in the high order byte of the AX register (AH). This interface provides access to the I2C Bus of the IPC@Chip for application programs.

- [Interrupt 0xAA function 0x80: Initialize the I2C Bus](#)
- [Interrupt 0xAA function 0x81: Scan I2C Devices](#)
- [Interrupt 0xAA function 0x82: Transmit / Receive Character](#)
- [Interrupt 0xAA function 0x83: Transmit/ Receive Block](#)
- [Interrupt 0xAA function 0x84: Release I2C Bus](#)
- [Interrupt 0xAA function 0x8b: Restart the I2C Bus](#)
- [Interrupt 0xAA function 0x8E: Select I2C Clock Pin](#)
- [Interrupt 0xAA function 0x8F: Select I2C Data Pin](#)

---

### Interrupt 0xAA service 0x80: Initialize the I2C Bus

This function sets the I2C bus clock speed to that specified by the caller. It also configures two of the programmable I/O (PIO) pins for usage as I2C bus data and clock signals.

#### Parameters

AH

0x80

AL

no longer used

CX

0

## Comments

The user can specify which two PIO are used for I2C **clock** and **data** . After calling this initialization function, these two pins will no longer be available as PIO pins unless the **PFE Enable** function is called for these pins following this function call.

## Related Topics

Select I2C **Clock** Pin

Select I2C **Data** Pin

PFE: **Enable** Programmable I/O Pins

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAA service 0x81: Scan I2C Devices

Report addresses of slave devices, one at a time.

## Parameters

*AH*

0x81

*AL*

First slave address (even address, LSB=0)

*CL*

Last slave address (even address, LSB=0)

## Return Value

AL: 0 no slave found

AL: -1 Timeout

AL: address of the first found slave

## Comments

This is an iterator function which is called repetitively to determine all connected slaves. Specify on each successive call a new restricted slave address range until no further address is returned by this function.

[Top of list](#)

## Interrupt 0xAA service 0x82: Transmit / Receive Character

Send or receive a single character.

### Parameters

*AH*

0x82

*AL*

Slave address, LSB:0 => Transmit, LSB:1 => Receive

*CL*

If Transmit: CL = Byte to transmit

If Receive: CL = 0 for last char to be received

### Return Value

Success: Carry flag cleared and CH contains received byte (if receiving)

Failure: Carry flag set and AL contains I2C error [code](#)

### Comments

The IPC@Chip is the I2C bus master.

The least significant bit of the slave address determines the direction of the communication.

Even address: Master sending to slave

Odd address: Master receiving from slave

#### I2C error codes:

5: Bus allocated

6: Bus arbitration failed

7: Bus error

8: Timeout

9: Slave faulty or not available

### Related Topics

I2C Transmit/Receive [Block](#)

[Top of list](#)

[Index page](#)

---

## Parameters

<i>AH</i>	0x83
<i>AL</i>	Slave address, LSB:0 => Transmit, LSB:1 => Receive
<i>CX</i>	Number of bytes to transmit or receive
<i>ES:BX</i>	Buffer address

## Return Value

Success: Carry flag cleared  
Failure: Carry flag set and AL contains I2C error [code](#)

## Comments

If an odd slave address is specified in AL then this function will dwell here until either CX bytes are received and stored in user buffer at [ES:BX], or until an error occurs. For an even slave address this function will dwell here until CX bytes from user buffer at [ES:BX] are transmit or until an error occurs.

## Related Topics

I2C Transmit/Receive [Character](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAA service 0x84: Release I2C Bus

### Parameters

<i>AH</i>	0x84
-----------	------

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAA service 0x8b: Restart the I2C Bus

## Parameters

*AH*  
0x8b

## Return Value

CF: 0

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAA service 0x8E: Select I2C Clock Pin

Select IPC@Chip I/O pin to be used for I2CCLK signal.

## Parameters

*AH*  
0x8E

*AL*  
PIO pin number, [0..13]

## Return Value

none

## Comments

The default I2C clock pin is PIO 0

To change the I2CCLK pin this function must be called before the I2C initialize [function](#) (0x80) is called.

## Related Topics

[Initialize](#) I2C Bus Function

Select I2C [Data](#) Pin

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAA service 0x8F: Select I2C Data Pin

Select IPC@Chip I/O pin to be used for I2CDAT signal.

## Parameters

*AH*  
0x8F

*AL*  
PIO pin number, [0..13]

## Return Value

none

## Comments

The default I2C data pin is PIO 1

To change the I2CDAT pin this function must be called before the I2C initialize [function](#) (0x80) is called.

## Related Topics

[Initialize](#) I2C Bus Function  
Select I2C [Clock](#) Pin

[Top of list](#)

[Index page](#)

---

End of document

---

## Fossil API - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### FOSSIL

API definition for access to the serial ports COM and EXT.  
The fossil standard uses software interrupt 0x14.

New in version 0.68: [Added Get driver info function](#)

New in version 0.66: [Added RS485 pin select function](#)

New in version 0.66: [Changed function RS485 Enable/Disable](#)

- [Interrupt 0x14 function 0x00: Set baud rate](#)
- [Interrupt 0x14 function 0x01: Put byte in output buffer, wait if needed.](#)
- [Interrupt 0x14 function 0x02: Get a byte from the serial port, wait if none available.](#)
- [Interrupt 0x14 function 0x03: Status request](#)
- [Interrupt 0x14 function 0x04: Initialize fossil driver](#)
- [Interrupt 0x14 function 0x05: Deinitialize fossil driver](#)
- [Interrupt 0x14 function 0x08: Flush output buffer waiting until done.](#)
- [Interrupt 0x14 function 0x09: Purge output buffer.](#)
- [Interrupt 0x14 function 0x0A: Purge receive buffer.](#)
- [Interrupt 0x14 function 0x0B: Transmit byte, do not wait.](#)
- [Interrupt 0x14 function 0x0C: Peek if next byte is available.](#)
- [Interrupt 0x14 function 0x0F: Enable/disable flow control.](#)
- [Interrupt 0x14 function 0x18: Read block of data](#)
- [Interrupt 0x14 function 0x19: Write a block of data](#)
- [Interrupt 0x14 function 0x1B: Get driver info](#)
- [Interrupt 0x14 function 0x80: Enable/Disable RS485 mode](#)
- [Interrupt 0x14 function 0x81: Extended line control initialization](#)
- [Interrupt 0x14 function 0x82: Select RS485 pin](#)

---

### Interrupt 0x14 service 0x00: Set baud rate

Set the baud rate of the serial port

## Parameters

*AH*

0x00

*AL*

Configuration parameter.

Bits 7--5: Baud rate, 4--3: Parity, 2: Stop bits, 1--0 Word length.

Bits 7--5: Baud rate

000	19200
001	38400
010	300
011	600
100	1200
101	2400
110	4800
111	9600

Bits 4--3: Parity

00	None
01	Odd
11	Even

Bit 2: Stop bits

0:	1 Stop bit
1:	2 Stop bits (available only when no parity is set)

Bits 1--0: Word length

1 0:	7 bits
1 1:	8 bits

*DX*

Port number: 0 for EXT, 1 for COM

## Return Value

AH: Status code (see service [0x03](#) )

## Comments

For higher baud rates use service [0x81](#) "Extended line control initialization"

Two stop bits are only available if no parity is set.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x01: Put byte in output buffer, wait if needed.

Character is queued for transmission. If there is space in the transmitter buffer when this call is made, the character will be stored and control returned to caller. If the buffer is full, the driver will wait for

space. (This can be dangerous when used in combination with flow control.)

## Parameters

*AH*  
0x01

*AL*  
Byte to be written

*DX*  
Port number: 0 for EXT, 1 for COM

## Return Value

AH: Status code (see service [0x03](#) )

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x02: Get a byte from the serial port, wait if none available.

Reads a byte from the receiver buffer. Wait for a byte to arrive if none is available.

## Parameters

*AH*  
0x02

*DX*  
Port number: 0 for EXT, 1 for COM

## Return Value

AL: The byte received

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x03: Status request

Return the status of the serial port.

## Parameters

*AH*  
0x03

*DX*  
Port number: 0 for EXT, 1 for COM

## Return Value

AH: Status code (bit field):

- bit 6: Set if output buffer is empty.
- bit 5: Set if output buffer is not full.
- bit 4: Line break detected
- bit 3: Framing error detected
- bit 2: Parity error detected
- bit 1: Set if overrun occurred on receiver.
- bit 0: Set if data is available in receiver buffer.

## Comments

Any reported UART error flags are cleared by hardware after the read is made for this call.

[Top of list](#)  
[Index page](#)

---

## Interrupt 0x14 service 0x04: Initialize fossil driver

Initialize the fossil driver for specified port.

## Parameters

*AH*  
0x04

*DX*  
Port specifier: 0 for EXT, 1 for COM

## Return Value

AX: 0x1954 if success

## Comments

Use this function to detect if the fossil driver is available for this port. The user must make sure that only one process opens a port. If this port is used for standard [input](#) or [output](#) (console), then stdin/stdout will be disabled for this port.

[Top of list](#)

## Interrupt 0x14 service 0x05: Deinitialize fossil driver

Deinitialize the fossil driver for specified port.

### Parameters

*AH*  
0x05

*DX*  
Port specifier: 0 for EXT, 1 for COM

### Return Value

none

## Interrupt 0x14 service 0x08: Flush output buffer waiting until done.

Wait for all output in the output buffer to be transmitted.

### Parameters

*AH*  
0x08

*DX*  
Port number: 0 for EXT, 1 for COM

### Return Value

none

## Interrupt 0x14 service 0x09: Purge output buffer.

Remove all data from the output buffer.

## Parameters

*AH*  
0x09

*DX*  
Port number: 0 for EXT, 1 for COM

## Return Value

none

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x0A: Purge receive buffer.

Remove all data from the receive buffer.

## Parameters

*AH*  
0x0A

*DX*  
Port number: 0 for EXT, 1 for COM

## Return Value

none

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x0B: Transmit byte, do not wait.

Place a byte in the transmit buffer if there is space available. Otherwise simply return with AX=0, without handling the transmit byte.

## Parameters

*AH*  
0x0B

*AL*

Byte to transmit

*DX*

Port number: 0 for EXT, 1 for COM

## Return Value

AX=0 if byte was not accepted (no space in buffer)

AX=1 if byte was placed in buffer

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x0C: Peek if next byte is available.

Returns the next byte available in the receive buffer, without removing it from the buffer.

## Parameters

*AH*

0x0C

*DX*

Port number: 0 for EXT, 1 for COM

## Return Value

AX=0xFFFF if no byte was available

AH=0x00 and AL=next byte, if a byte was available.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x0F: Enable/disable flow control.

Configure the flow control for a port.

## Parameters

*AH*

0x0F

*AL*

Bit mask describing requested flow control.

*DX*

Port number: 0 for EXT, 1 for COM

## Return Value

none

## Comments

Bit fields for FOSSIL data flow control:

- B0: XON/XOFF on transmit (watch for XOFF while sending)
- B1: CTS/RTS (CTS on transmit/RTS on receive)
- B2: reserved
- B3: XON/XOFF on receive (send XOFF when buffer near full)
- B4-B7: Ignored

Notes:

- XON/XOFF and CTS/RTS are not allowed at the same time.

---

## Developer Notes

Since BIOS 1.02B XON/XOFF mode is also available if the **DMA** mode for the COM port is enabled but because of the internal functionality of DMA it is not possible to detect an XON or XOFF of the peer immediately. It is possible that an overrun situation at the connected peer (e.g. GSM modem) could occur. We enable this mode now because GSM modems (any??) supports only XON/XOFF flow ctrl.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x18: Read block of data

Read up to a specified number of bytes from a serial port.

## Parameters

*AH*

0x18

*CX*

Maximum number of bytes to transfer.

*DX*

Port number: 0 for EXT, 1 for COM

*ES:DI*

Pointer to user buffer.

## Return Value

AX= Number of bytes transferred.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x19: Write a block of data

Write a block of data to the serial port buffer.

### Parameters

*AH*

0x19

*CX*

Maximum number of bytes to transfer.

*DX*

Port number: 0 for EXT, 1 for COM

*ES:DI*

Pointer to user buffer.

## Return Value

AX= Number of bytes actually transferred.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x1B: Get driver info

Get information about a serial port and driver

### Parameters

*AH*

0x1B

*CX*

Size of buffer

*DX*

Port number: 0 for EXT, 1 for COM

*ES:DI*

Pointer to user buffer.

## Return Value

AX=Number of bytes actually transferred.

## Comments

Offset 0 (word) = Structure size  
Offset 2 (byte) = FOSSIL spec version (not used)  
Offset 3 (byte) = Driver rev level (not used)  
Offset 4 (dword) = Pointer to ASCII ID (not used)  
Offset 8 (word) = Input buffer size  
Offset 0A (word) = Bytes available (input)  
Offset 0C (word) = Output buffer size  
Offset 0E (word) = Bytes available (output)  
Offset 10 (byte) = Screen width, chars (not used)  
Offset 11 (byte) = Screen height, chars (not used)  
Offset 12 (byte) = Baud rate mask (not used)

This function was only added for compatibility to older fossil applications

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x80: Enable/Disable RS485 mode

Enable the RS485 mode.

## Parameters

*AH*

0x80

*AL*

0=TxEnable low active  
1=TxEnable high active  
2=Disable RS485 mode

*DX*

Port number: 0 for EXT, 1 for COM

## Return Value

none

## Comments

By default the RTS0 and RTS1 signals (pins) are used to enable/disable the respective (EXT or COM) transmitter. (TxEnable)

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x81: Extended line control initialization

Extended line control initialization.

### Parameters

*AH*

0x81

*AL*

UART character data bits

2: 7 bits

3: 8 bits

*BH*

Parity

0: no parity

1: odd parity

2: even parity

3: mark parity (always 1)

4: space parity (always 0)

*BL*

Stop bits

0: 1 Stop bit

1: 2 Stop bits (only available when no parity is selected)

*CX*

Baud rate divider

(for maximum baud rate see HAL function [0x8A](#) )

*DX*

Port number: 0 for EXT, 1 for COM

### Return Value

none

## Comments

Two stop bits are only available if no parity is set.

---

## Developer Notes

Parity settings "Mark" and "Space", and two stop bits are not checked on received data by the @Chip (UART) or the API. This is due to these modes are not available in hardware. These modes are provided to communicate with hardware that can operate only in these modes.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x14 service 0x82: Select RS485 pin

Select the RS485 TxEnable pin

### Parameters

*AH*

0x82

*AL*

No of PIO pin [0..13]

*DX*

Port number: 0 for EXT, 1 for COM

### Return Value

none

### Comments

By default the RTS0 and RTS1 signals (pins) are used to enable/disable the respective transmitter. (TxEnable) This function lets you select any PIO from 0-13 as TxEnable, but not any makes sense. To change the default, call this function before you call the RS485 **Enable** function.

[Top of list](#)

[Index page](#)

---

End of document

---

## PPP Interface - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### Introduction

Here is a short description of how to configure the SC12 PPP server.

The PPP server is available starting with BIOS version SC12V0067PPP. (There also exists a BIOS version SC12V0067 without the PPP server.) The PPP client API is available starting with BIOS 070.

The PPP client and server API calls are part of the TCP/IP API. These API calls are described in the [TCP/IP API](#) documentation. Only configuration of the PPP server is described here.

#### Topics:

- [About PPP](#)
- [Configuring the PPP Server](#)
- [PPP Server API](#)
- [PPP Client API](#)
- [Available Examples](#)

---

### About PPP

Since SC12 BIOS version 067 Beta a PPP server is available in the SC12. PPP (the Point to Point Protocol) is a mechanism for creating and running TCP/IP over a serial link - be that a direct serial connection (using a null-modem cable), or a link made using an analogue modem.

Other computers can dial into the SC12 PPP server and communicate via the TCP/IP link using FTP, Telnet, Web, etc. in the same manner as with an Ethernet TCP/IP link. One major difference between a PPP and an Ethernet connection is of course the speed. A standard Ethernet connection operates at 10 Mbs maximum theoretical throughput, whereas an analogue modem operates at speeds up to 56 kbps.

PPP is strictly a peer to peer protocol; there is no difference between the machine that dials in and the machine that is dialed into. However, it is still useful to think in terms of servers and clients. When you dial into a site to establish a PPP connection, you are considered the client. The machine to which you connect (e.g. the SC12) is considered the server.

PPP at the IPC@Chip includes the subprotocols LCP and NCP(IPCP).

Supported authentication protocols are PAP and CHAP.

[Top of list](#)  
[Index page](#)

---

## Configuring the PPP Server

The PPP server must be configured using the `chip.ini` file sections [PPPSERVER] and [SERIAL]. All entries for configuring the PPP server are listed in the [config.htm](#) file.

Here is an outline of the steps required to configure the PPP server.

1. Disable/[enable PPP server](#)  
ENABLE=0 or ENABLE=1  
By default, PPP server is enabled.
2. Select a COM port for the PPP server to use with the [COMPORT](#) directive.

### Important :

The COM and EXT port of the SC12 has only CTS,RTS,RxD and TxD lines, so you have to configure your modem with DTR always on. (e.g. AT cmd for a ZyxEL modem: AT&D0)

3. Increase the [send](#) and [receive](#) queue sizes of the chosen serial port (EXT or [COM](#) ). Recommended size is 4096 Bytes.

Example `chip.ini` settings for the EXT port:

```
[SERIAL]  
EXT_RECVQUEUE=4096  
EXT_SENDQUEUE=4096
```

4. Select the [flow control mode](#) for the PPP server COM port.
5. Set the baud rate of PPP server's COM port e.g. [BAUD](#) =19200 (default is 38400)
6. Select usage of an analogue modem: [MODEM](#) =1.  
The default is 0 (null modem cable).
7. Set the [IP Address](#) of the PPP server interface e.g. ADDRESS=192.168.206.4
8. Set the [REMOTE IP Address](#) for the connected host.

Three different possibilities for configuring PPP server IP addresses at the IPC@Chip:

1. If valid addresses for [IP Address](#) and [REMOTE IP Address](#) are declared at `chip.ini`, the PPP server wants to use this configured IP for its own and want to configure the remote peer with the defined remote address.
2. If only [IP Address](#) is declared at `chip.ini` and [REMOTE IP Address](#) is set to 0.0.0.0, the

PPP server wants to use his defined address and expects that the client uses his own address.

3. If both entries **IP Address** and **REMOTE IP Address** are set to 0.0.0.0, the PPP server expects an IP address from the peer.

9. Define **net mask** and router **default gateway**, e.g.

```
NETMASK=255.255.255.0  
GATEWAY=192.168.206.4
```

Note about IP FORWARDING:

Since BIOS version 067 the SC12 has two network interfaces, Ethernet and PPP, so the SC12 can forward IP packets between these interfaces.

If you define a gateway in the PPPSERVER section of the `chip.ini` for the PPP server interface, it becomes the default gateway for all interfaces when a PPP link to the server is established. During a PPP server connection the command **ipcfg** indicates this

default gateway. After the PPP session, the old gateway (if any) for the Ethernet interface will be restored. As of BIOS version 070, the TCP/IP API supports adding and deleting a default gateway:

10. Choose **authentication mode** with the `AUTH` directive.

11. Initialize the analogue modem.

You can define three sets of modem initialization parameters. These parameters are used to initialize the modem at the start of the SC12 BIOS and after a modem hang-up following a PPP session.

Each of the three parameter sets consists of the following four parameters:

- **INITCMD** string - Command sent to the modem to initialize it.
- **INITANSWER** string - Expected modem response to initialization command.
- **INITTIMEOUT** integer - Number of seconds to wait on answer from modem.
- **INITRETRIES** integer - Number of times to repeat modem initialization sequence if a previous attempt fails.

Example:

```
INITCMD0=ATZ  
INITANSWER0=OK  
INITTIMEOUT0=2  
INITRETRIES0=3
```

```
INITCMD1=AT&D0  
INITANSWER1=OK  
INITTIMEOUT1=2  
INITRETRIES1=3
```

```
INITCMD2=AT  
INITANSWER2=OK  
INITTIMEOUT2=2  
INITRETRIES2=2
```

A timeout value 0 means wait forever for the modem's answer.  
If you enter the string NULL at an INITANSWER (e.g. INITANSWER0=NULL), the SC12 PPP server will not wait for an answer from the modem.

12. Define a maximum of three modem commands for getting **connected** to the remote peer.

Example:

```
CONNECTMSG0=RING
CONNECTANSWER0=ATA
CONNECTTIMEOUT0=0
CONNECTMSG1=CONNECT
CONNECTTIMEOUT1=60
```

These are the default values for modem connect commands.

In this example the PPP server waits forever for the RING message and sends an ATA to the modem if it responds to the RING. After that the server waits a maximum of 60 seconds for a response to the CONNECT message. The modem link is established. The server now establishes the PPP connection to the remote client.

Note:

Do not use the AT command ATS0=1, this will cause the modem to automatically answer the call without waiting for the PPP server. This is too fast for the PPP server.

13. Hang-up the connection.

The PPP server will attempt to hang-up the modem when either a connection is closed by a remote peer, or if the modem initialization failed during the SC12 boot process.

- Switch the modem into the command mode (**CMDMODE** and **HANGUPDELAY** )

Example:

```
CMDMODE=+++
HANGUPDELAY=2
```

These are the default values.

- Define modem commands and expected answers for **hang-up** .  
Again, up to three sets of these parameters can be given here.

Example:

```
HANGUPCMD0=ATH0
HANGUPANSWER0=OK
HANGUPTIMEOUT0=2
HANGUPRETRIES0=1
```

If you enter the string NULL at an HANGUPANSWERx (e.g. HANGUPANSWER0=NULL), the SC12 PPP server will not wait for an answer from the modem.

14. Control the online state while PPP session is open

You can define three sets of modem control parameters. These parameters are used to check the online state of the modem at an open PPP connection.

- \. Enable online control sequence **MODEMCTRL** =1.  
The default is 0 (disabled).

Example:

```
MODEMCTRL=1
```

Also you have to configure a control time interval (in seconds). After each time interval, within the PPP server receive no data, the server executes the configured modem commands.

The server closes the connection, if one of the expected answers timed out.

- a. Define a **CTRLTIME** I

E.g. **CTRLTIME**=120      default is 60 seconds

Each of the three parameter sets consists of the following four parameters:

- **CTRLCMD** string - Command sent to the modem to initialize it.
- **CTRLANSWER** string - Expected modem response to control command.
- **CTRLTIMEOUT** integer - Number of seconds to wait on answer from modem.
- **CTRLRETRIES** integer - Number of times to repeat modem control sequence if a previous attempt fails.

Example and default settings:

```
CTRLCMD0=+++  
CTRLANSWER0=OK  
CTRLTIMEOUT0=3  
CTRLRETRIES0=1
```

```
CTRLCMD1=AT0  
CTRLANSWER1=NULL  
CTRLTIMEOUT1=1  
CTRLRETRIES1=0
```

If you enter the string NULL at an CTRLANSWERx (e.g. CTRLANSWER0=NULL), the SC12 PPP server will not wait for an answer from the modem.

15. Define a **time out** value in seconds after which the PPP server hangs up the connection if no data comes in from client during this timeout period.

E.g. **IDLETIME**=160      default is 120

[Top of list](#)

[Index page](#)

---

The TCP/IP API provides five calls that apply to the PPP server.

1. Interrupt 0xAC, [Service 0x50](#) : Check if the PPP server is installed.
2. Interrupt 0xAC, [Service 0x51](#) : Suspend PPP server task
3. Interrupt 0xAC, [Service 0x52](#) : Activate PPP server
4. Interrupt 0xAC, [Service 0x53](#) : Get current state of the PPP server
5. Interrupt 0xAC, [Service 0x54](#) : Get the PPP server configuration

[Top of list](#)

[Index page](#)

---

## PPP Client API

The TCP/IP API provides four calls that apply to the PPP client.

1. Interrupt 0xAC, [Service 0x40](#) : Check if the PPP client is installed.
2. Interrupt 0xAC, [Service 0x41](#) : Open a connection to PPP server
3. Interrupt 0xAC, [Service 0x42](#) : Close connection
4. Interrupt 0xAC, [Service 0x43](#) : Get current state of the PPP client

[Top of list](#)

[Index page](#)

---

## Available Examples

The following example code is available:

- o PPP server API test, PPS.C
- o PPP client example, PPPCLIE.C

[Top of list](#)

[Index page](#)

---

End of document

---

## I2C Bus API Updates - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### I2C Bus API News

The following extensions to the I2C [API](#) are available in the indicated BIOS revisions.

New in version 1.00: [Changed I2C Bus init function](#)

New in version 0.70: [I2C functions for definition of I2C bus pins](#)

---

End of document

---

## DOS Interface Documentation - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

DOS API [News](#)

---

### DOS

Here are the DOS interface definitions. DOS uses interrupt 0x21 with a service number in the high order byte of the AX register (AH).

#### DOS API [News](#)

All implemented DOS services are listed here:

- [Interrupt 0x21 function 0x00: Terminate Program](#)
- [Interrupt 0x21 function 0x02: Output Character to standard output](#)
- [Interrupt 0x21 function 0x06: Direct Console Output](#)
- [Interrupt 0x21 function 0x07: Direct Console Input](#)
- [Interrupt 0x21 function 0x08: Read Keyboard](#)
- [Interrupt 0x21 function 0x09: Send string to standard output](#)
- [Interrupt 0x21 function 0x0B: Character Available Test](#)
- [Interrupt 0x21 function 0x0E: Set Default Drive](#)
- [Interrupt 0x21 function 0x19: Get Current Drive](#)
- [Interrupt 0x21 function 0x1A: Set Disk Transfer Area Address](#)
- [Interrupt 0x21 function 0x25: Set IRQ Vector](#)
- [Interrupt 0x21 function 0x2A: Get System Date](#)
- [Interrupt 0x21 function 0x2B: Set System Date](#)
- [Interrupt 0x21 function 0x2C: Get System Time](#)
- [Interrupt 0x21 function 0x2D: Set System Time](#)
- [Interrupt 0x21 function 0x2F: Get Disk Transfer Area Address](#)
- [Interrupt 0x21 function 0x30: Get DOS Version](#)
- [Interrupt 0x21 function 0x31: Keep Process](#)
- [Interrupt 0x21 function 0x35: Get IRQ Vector](#)
- [Interrupt 0x21 function 0x36: Get Disk Free Space](#)
- [Interrupt 0x21 function 0x39: Create Directory](#)
- [Interrupt 0x21 function 0x3A: Remove Directory](#)
- [Interrupt 0x21 function 0x3B: Set Current Working Directory](#)
- [Interrupt 0x21 function 0x3C: Create New File Handle](#)
- [Interrupt 0x21 function 0x3D: Open an Existing File](#)

- [Interrupt 0x21 function 0x3E: Close File Handle](#)
- [Interrupt 0x21 function 0x3F: Read from File](#)
- [Interrupt 0x21 function 0x40: Write to File](#)
- [Interrupt 0x21 function 0x41: Delete File](#)
- [Interrupt 0x21 function 0x42: Set Current File Position](#)
- [Interrupt 0x21 function 0x43: Get/Set File Attributes](#)
- [Interrupt 0x21 function 0x44: IOCTL, Set/Get Device Information](#)
- [Interrupt 0x21 function 0x47: Get Current Working Directory](#)
- [Interrupt 0x21 function 0x48: Allocate Memory](#)
- [Interrupt 0x21 function 0x49: Free Allocated Memory](#)
- [Interrupt 0x21 function 0x4A: Resize Memory](#)
- [Interrupt 0x21 function 0x4B: EXEC](#)
- [Interrupt 0x21 function 0x4C: End Process](#)
- [Interrupt 0x21 function 0x4E: Find First File](#)
- [Interrupt 0x21 function 0x4F: Find Next File](#)
- [Interrupt 0x21 function 0x50: Debugger Support](#)
- [Interrupt 0x21 function 0x51: Get PSP Segment Address](#)
- [Interrupt 0x21 function 0x56: Change Directory Entry, Rename File](#)
- [Interrupt 0x21 function 0x57: Get/Set File Date and Time](#)
- [Interrupt 0x21 function 0x58: Get/Set memory strategy \(dummy function\)](#)
- [Interrupt 0x21 function 0x62: Get PSP Segment Address](#)
- [Interrupt 0x21 function 0x63: Get Leading Byte \(stub\)](#)
- [Interrupt 0x21 function 0x68: Flush DOS Buffers to Disk](#)

Any service not listed is not supported. A warning will be issued on the console when an unimplemented DOS interrupt 0x21 service is requested. If you need a function that is not supported, please let us know at <mailto:atchip@beck-ipc.com>

A maximum of 12 DOS programs can run.

All DOS tasks together can open a maximum of 10 files.

## Interrupt 0x21 service 0x00: Terminate Program

Refer to interrupt 0x21, [service 0x4C](#) .

### Parameters

*AH*  
0x00

### Comments

This service has been replaced by service 0x4C. The system will treat both as the same function.

[Top of list](#)

## Interrupt 0x21 service 0x02: Output Character to standard output

Send the character in DL to the standard output.

### Parameters

*AH*  
0x02

*DL*  
Character to be output to [stdout](#)

### Return Value

Returns nothing

### Comments

Each potential output device has its own output buffer. This function queues the provided output character into each device's output buffer for which [stdout](#) is configured.

The transmitters are interrupt driven buffered I/O. If space is available in the transmit buffer(s) when this call is made, the character is stored and control returned immediately to the caller. Otherwise a wait loop is entered, awaiting space in each configured transmit buffer.

This function does not check for Ctrl-C.

### Related Topics

[stdout](#) configuration

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x06: Direct Console Output

If DL!=0xFF: Send the character in DL to the standard output.  
If DL==0xFF: read character from `stdin` if one is available.

### Parameters

*AH*  
0x06

*DL*

Character to be output to `stdout`

## Return Value

If call with `DL!=0xFF` then no return value (only output to `stdout` )

If call with `DL==0xFF` then

    If input character is available at `stdin` then

        Return input character in AL and reset CPU's zero flag

    Else

        Set CPU's zero flag to indicate no character available

    Endif

Endif

Returns at BX the source `stdin` channel of the character: 1: EXT , 2: COM , 4: Telnet

## Comments

Output is buffered and interrupt driven. This function will return after placing output character into the transmit buffer.

This function does not check for Ctrl-C.

## Related Topics

[stdin](#) configuration

[stdout](#) configuration

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x07: Direct Console Input

Wait for a character to be read from standard input

## Parameters

*AH*

0x07

## Return Value

Returns the character read in AL.

## Comments

This function is identical to interrupt 0x21, [service 0x08](#) .

This function does not echo the received character and it does not check for Ctrl-C.

## Related Topics

[stdin](#) configuration

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x08: Read Keyboard

Wait for a character to be read from standard input

### Parameters

*AH*  
0x08

### Return Value

Returns the character read in AL.

Returns at BX the source stdin channel of the character: 1: EXT , 2: COM , 4: Telnet

### Comments

This function is identical to interrupt 0x21, [service 0x07](#) .

This function does not echo the character and it does not check for Ctrl-C.

## Related Topics

[stdin](#) configuration

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x09: Send string to standard output

Send a string to `stdout` ending with '\$' or null terminated.

### Parameters

*AH*  
0x09

*DS:DX*

Specifies a pointer to the first character of the string.

## Return Value

Returns nothing.

## Comments

This function does not check for Ctrl-C.

## Related Topics

[stdout](#) configuration

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x0B: Character Available Test

Check if a character from standard input is available.

## Parameters

*AH*

0x0B

## Return Value

AL=0x00: No character is available.

AL=0xFF: Character is available.

## Comments

This function does not check for Ctrl-C.

## Related Topics

[stdin](#) configuration

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x0E: Set Default Drive

Change the default drive for the current task.

## Parameters

*AH*  
0x0E

*DL*  
new default drive (00h = A:, 01h = B:, etc)

## Related Topics

[Get](#) current drive

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x19: Get Current Drive

Returns the current drive for this process.

## Parameters

*AH*  
0x19

## Return Value

AL=drive where 0 is A:, 1 is B:, ..., 4 is E:

## Related Topics

[Set](#) default drive

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x1A: Set Disk Transfer Area Address

Set address of the Disk Transfer Area (DTA) needed for `findfirst`/`findnext` functions.

## Parameters

*AH*

0x2A

*DS:DX*

Pointer to DTA

## Comments

The main task of your application has a standard pointer to a Disk Transfer Area of the program. Tasks created with the RTOS API inside of your application use a DTA from an internal list. A task, which gets access to the file system (via [RTX\\_ACCESS\\_FILESYSTEM](#) service) reserves an entry in this list.

Only ten DTA entries for user tasks are available.

## Related Topics

DOS [Get](#) Disk Transfer Area address service  
RTOS's [RTX\\_ACCESS\\_FILESYSTEM](#) Service

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x25: Set IRQ Vector

This function allows you to set an interrupt vector to your interrupt function.

You can use the following IRQ's

0x0ADMA0 / INT5

0x0BDMA1 / INT6

0x0CINT0

0x0EINT2

0x0FINT3

0x10 INT4

## Parameters

*AH*

0x25

*AL*

Specifies vector number.

*DS:DX*

Vector to your interrupt procedure.

## Return Value

No return value.

## Comments

IRQ 0x0D (Ethernet) and IRQ 0x13 (Timer) cannot be changed !

Also this DOS service interrupt 0x21 vector cannot be changed using this service.

## Related Topics

[Get](#) IRQ vector

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x2A: Get System Date

Returns the system date

## Parameters

*AH*

0x2A

## Return Value

CX=Year (full 4 digits), DH=Month, DL=Day, AL=day of week (0=Sunday)

## Related Topics

[Set](#) system date

[Get](#) system time

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x2B: Set System Date

Sets the system date.

## Parameters

*AH*

0x2B

*CX*

Year (including century, e.g. 2001)

*DH*  
Month (1..12)

*DL*  
Day (1..31)

## Comments

This function performs no error checking on entered date.

## Related Topics

[Get](#) system date

[Set](#) system time

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x2C: Get System Time

Returns the system time

## Parameters

*AH*  
0x2C

## Return Value

CH=Hour, CL=Minute, DH=Second, DL=0

## Related Topics

[Get](#) system date

[Set](#) system time

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x2D: Set System Time

Sets the system time

## Parameters

<i>AH</i>	0x2D
<i>CH</i>	Hour
<i>CL</i>	Minute
<i>DH</i>	Second

## Related Topics

[Set](#) system date

[Get](#) system time

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x2F: Get Disk Transfer Area Address

Get address of the Disk Transfer Area (DTA) needed for findfirst/findnext.

## Parameters

<i>AH</i>	0x2F
-----------	------

## Return Value

Returns the address of the DTA in ES:BX

## Comments

The main task of your application has a standard pointer to a Disk Transfer Area of the program. Tasks created with the RTOS API inside of your application use a DTA from an internal list. A task, which gets access to the file system (via [RTX\\_ACCESS\\_FILESYSTEM](#) service) reserves an entry in this list.

Only ten DTA entries for user tasks are available.

## Related Topics

DOS [Set](#) Disk Transfer Area address service  
RTOS's [RTX\\_ACCESS\\_FILESYSTEM](#) Service

## Interrupt 0x21 service 0x30: Get DOS Version

Get the version number of DOS.

### Parameters

*AH*  
0x30

### Return Value

Returns the DOS version in AX.

### Comments

This function always returns 0x0003 (meaning version 3.00).  
However, this does not mean that we have a full implementation of DOS 3.0 !

## Interrupt 0x21 service 0x31: Keep Process

Makes a program remain resident after it terminates.

### Parameters

*AH*  
0x31

*DX*  
Memory size, in paragraphs, required by the program

### Return Value

None

## Interrupt 0x21 service 0x35: Get IRQ Vector

Get the address of an interrupt service routine.

### Parameters

*AH*  
0x35

*AL*  
Specifies vector number.

### Return Value

Returns the vector in ES:BX

### Related Topics

[Set](#) IRQ vector

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x36: Get Disk Free Space

Simplified DOS function for detecting disk free space.

### Parameters

*AH*  
0x36

*DL*  
Drive (0 current drive, 1=A, ...)

### Return Value

AX: -1 Invalid drive number  
else

AX: always 1

BX: number of free sectors

CX: bytes per sector

DX: always 1

### Comments

When call is successful (AX=1), free disk space can be computed from the return values as:

free disk space (bytes) = BX \* CX

At BIOS version 1.00 this function had a bug: Parameter 0 at DL was drive A:, but value of 0 at DL should mean the current drive.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x39: Create Directory

Create a new subdirectory

### Parameters

*AH*

0x39

*DS:DX*

Pointer to null terminated path name.

### Return Value

Carry flag is cleared on success, set on error.

On error AX contains error code:

2: File not found

3: Path not found

5: Path exists or access denied

### Related Topics

[Remove](#) Directory

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x3A: Remove Directory

Delete a subdirectory.

### Parameters

*AH*

0x3A

*DS:DX*

Pointer to null terminated path name.

## Return Value

Carry flag is cleared on success, set on error.

On error AX contains error code:

2: File not found

3: Path not found

5: access denied, not a directory, not empty, or in use

## Comments

The subdirectory must not contain any files.

## Related Topics

[Create](#) Directory

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x3B: Set Current Working Directory

Set current working directory.

## Parameters

*AH*

0x3B

*DS:DX*

Null terminated path of new current working directory

## Return Value

Carry flag is cleared on success, set on error.

## Comments

If the path contains a drive name, the current working directory of that drive is changed without changing the default drive. Otherwise, the current working directory is changed for the default drive.

Each task has it's own current working directory.

## Related Topics

[Get](#) current working directory

## Interrupt 0x21 service 0x3C: Create New File Handle

Create a file of specified name. If a file by this name already exists, it is deleted. The returned file handle is for a new empty file.

### Parameters

*AH*  
0x3C

*CX*  
File attributes (bit field):  
B0 - Read Only  
B1 - Hidden File  
B2 - System File  
B5 - Archive Flag

*DS:DX*  
Pointer to a null terminated file name and path

### Return Value

Success: Carry flag cleared, AX = file handle  
Failure: Carry flag set, AX = error code:  
AX=2: Path not found  
AX=3: File name length exceeded 147 character limit  
AX=4: Too many files open  
AX=5: Invalid file name or access denied

### Comments

Files are always opened in a non-sharing mode.

### Related Topics

[Open](#) Existing File  
[Close](#) File Handle

## Interrupt 0x21 service 0x3D: Open an Existing File

Opens an existing file.

## Parameters

*AH*  
0x3D

*AL*  
Open mode:  
AL=0: Open for read  
AL=1: Open for write  
AL=2: Open for read and write

*DS:DX*  
Pointer to a null terminated file path.

## Return Value

Success: Carry flag cleared, AX = file handle  
Failure: Carry flag set, AX = error code:  
AX=2: Path or file not found  
AX=4: Too many files open  
AX=5: Access denied

## Comments

In writing mode files are always opened in a non-sharing mode.

## Related Topics

[Create](#) New File Handle

[Close](#) File Handle

Get/Set File [Attributes](#)

The filesystem doesn't make a difference if the file was not found or the path.  
The return value in both error cases is 2.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x3E: Close File Handle

Closes an open file.

## Parameters

*AH*  
0x3E

*BX*  
File handle

## Return Value

Success: Carry flag cleared  
Failure: Carry flag set, AX = error code:  
AX=6: file not open

## Related Topics

[Create](#) New File Handle

[Open](#) Existing File

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x3F: Read from File

Reads a number of bytes from a file, the handle of which is specified in BX.

## Parameters

*AH*  
0x3F

*BX*  
File handle

*CX*  
Number of bytes to read

*DS:DX*  
Pointer to the destination data buffer.

## Return Value

Success: Carry flag cleared, AX = number of bytes read into buffer from file  
Failure: Carry flag set, AX = error code:  
AX=5: Access denied  
AX=6: Invalid file handle

## Related Topics

[Create](#) New File Handle

[Open](#) Existing File

[Write](#) to File

## Interrupt 0x21 service 0x40: Write to File

Writes a number of bytes to a file, the handle of which is specified in BX.

### Parameters

*AH*

0x40

*BX*

File handle

*CX*

Number of bytes to write

*DS:DX*

Pointer to the source data buffer.

### Return Value

Success: Carry flag cleared, AX = number of bytes written into the file

Failure: Carry flag set, AX = error code:

AX=5: Access denied

AX=6: Invalid file handle

### Comments

Requesting zero bytes written to file (CX=0) truncates the file at the current position.

### Related Topics

[Create](#) New File Handle

[Open](#) Existing File

[Read](#) from File

[Set](#) Current File Position

## Interrupt 0x21 service 0x41: Delete File

Deletes a file. Wildcards are not allowed.

## Parameters

*AH*  
0x41

*DS:DX*  
Pointer to null terminated file name and path.

## Return Value

Success: Carry flag is cleared

Failure: Carry flag is set, AX holds error code:

AX=2: File not found

AX=5: Access denied

## Related Topics

[Create](#) New File Handle

Get/Set File [Attributes](#)

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x42: Set Current File Position

The operating system maintains a 32 bit file pointer that it uses for read or write requests to the respective file. This service can be used to either read or set this file pointer. The file pointer associated with handle is set to a new byte position offset relative to the origin of the move.

## Parameters

*AH*  
0x42

*AL*  
Origin of move  
0x00: Relative to start of file  
0x01: Relative to current position  
0x02: Relative to end of file

*BX*  
File handle

*CX,DX*  
Offset of the displacement with higher order word in CX.

## Return Value

Success: Carry flag is cleared, DX,AX holds the new position relative to the start of the file with high word in DX.

Failure: Carry flag is set, AX holds error code:

AX = 0x06: Invalid handle

AX = 0x19: Invalid displacement

## Comments

If you attempt to seek beyond the end of file, the file pointer will be positioned at the end of the file.

To read current file position without changing it, call with AL=1, CX:DX = 0:0.

## Related Topics

[Read](#) from File

[Write](#) to File

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x43: Get/Set File Attributes

Use this function to inspect or change the attributes of a file.

### Parameters

*AH*

0x43

*AL*

0: get, 1: set

*CX*

File attributes (bit field):

B0 - Read Only

B1 - Hidden File

B2 - System File

B5 - Archive Flag

*DS:DX*

Pointer to a null terminated string holding the file path.

## Return Value

Success: Carry flag cleared, file attributes in CX (bit field):

... same flag bits as CX input parameter with additional bits ...

B3 - Volume

B4 - Directory Entry

Failure: Carry flag set, AX holds error code:

AX=1: Invalid function (wrong value in AL)

AX=2: File not found

AX=5: Access denied

## Comments

Input parameter CX is not used when input parameter AL = 0.

## Related Topics

[Delete](#) File

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x44: IOCTL, Set/Get Device Information

Changes the data that DOS uses to control a device.

## Parameters

*AH*

0x44

*AL*

0: Get device data, 1: Set device data

*BX*

Handle

*DX*

Device data

## Return Value

Success: Carry flag cleared, Device data in DX.

Failure: Carry flag set, AX holds error code

AX=1: Invalid function (wrong value in AL)

AX=6: Invalid handle

## Comments

If bit 7 of the data is 1, the handle refers to a device and data bit assignments are as follows:

Bit	Meaning when bit is set to '1'
B15	Reserved

B14	Device can handle function 0x44, codes 2 and 3
B13	Device supports output until busy
B12	Reserved
B11	Device understands open/close
B10-8	Reserved
B7	Set to '1' to indicate handle refers to a device.
B6	Not "end of file" on input
B5	Don't check for control characters
B4	Reserved
B3	Clock device
B2	Null device
B1	Console Output device
B0	Console Input device

If bit 7 of the data is 0, the handle refers to a file and data bit assignments are as follows:

Bit	Meaning
B15-8	Reserved
B7	Set to '0' to indicate handle refers to a file.
B6	Set to '0' when the file has been written
B0-5	Drive number (0=A, 1=B, etc)

The first 3 file handles are used for the stdio devices:

- 0: Input
- 1: Output
- 2: Error

This service was implemented to be compatible with the older DOS compilers. The data is saved when you issue a write, but the data is not used by DOS. Control characters are not recognized as such. Function 0x44 codes 2 and 3 are not supported.

[Top of list](#)  
[Index page](#)

---

## Interrupt 0x21 service 0x47: Get Current Working Directory

Get current working directory for a drive.

### Parameters

*AH*

0x47

*DS:SI*

Pointer to a 64 byte memory area to receive null terminated path of current working directory (cwd).

*DL*

Drive number, 0 for current, 1 for A, ..

### Return Value

Success: Carry flag cleared, Buffer at [DS:SI] contains cwd path.

Failure: Carry flag set, error code in AX=15

## Comments

Each task has its own current working directory. When a program starts, its current drive and working directory will be set to the drive and directory that was current before the program started.

## Related Topics

[Set](#) current working directory

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x48: Allocate Memory

Allocate memory for the process

### Parameters

*AH*

0x48

*BX*

Size counted in paragraphs

### Return Value

Success: Carry flag cleared, AX holds the segment of the memory area

Failure: Carry flag set due to not enough memory available, AX = 0,

BX holds the size of the largest free block available expressed by paragraph count.

## Comments

A paragraph is 16 bytes in length.

## Related Topics

[Free](#) allocated memory

[Resize](#) memory

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x49: Free Allocated Memory

Release allocated memory

## Parameters

*AH*  
0x49

*ES*  
Segment of the memory area as returned by function [0x48](#)

## Return Value

Success: Carry flag cleared  
Failure: Carry flag set, AX holds the error code =9.

## Related Topics

[Allocate](#) memory

[Resize](#) memory

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x4A: Resize Memory

Increase or decrease the size of a memory allocation block

## Parameters

*AH*  
0x4A

*BX*  
Desired new size expressed in paragraphs

*ES*  
Segment address of memory block

## Return Value

Success: Carry flag cleared  
Failure: Carry flag set due to not enough memory available and the size of the largest free block is returned in BX (paragraph count).

## Comments

A paragraph is 16 bytes in length The call failed, if a user tries at his application to increase a memory

block, allocated with int21h 0x48. In that case it only possible to decrease the size of the allocated memory block.

## Related Topics

[Allocate](#) memory

[Free](#) memory

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x4B: EXEC

Load and/or execute a program. This function loads the program and builds its PSP (Program Segment Prefix) based on a parameter block that you provide.

### Parameters

*AH*

0x4B

*AL*

Type of load:

00h - Load and execute

01h - Load but do not execute

*DS:DX*

Null terminated program name (must include extension)

*ES:BX*

parameter block

Offset	Size	Description
00h	WORD	Segment of environment to copy for child process (copy caller's environment if 0000h)
02h	DWORD	Pointer to command tail to be copied into child's PSP
06h	DWORD	Pointer to first FCB to be copied into child's PSP
0Ah	DWORD	Pointer to second FCB to be copied into child's PSP
0Eh	DWORD	(AL=01h) will hold subprogram's initial SS:SP on return
12h	DWORD	(AL=01h) will hold entry point (CS:IP) on return

### Return Value

Success: Carry flag cleared, AX= task ID,

BX= segment of PSP, add sizeof PSP/16 for relocation offset.

(If sub function 01h used, the task is waiting for it's trigger.)

Failure: Carry flag set, AX = 1

### Comments

Use sub function 0x00 to load and execute another program.  
Subfunction 0x01 is used to load a program without executing it. This option is used by debuggers.  
For both functions, the calling process must ensure that there is enough unallocated memory available; if necessary, by releasing memory with services [AH=0x49](#) or [AH=0x4A](#) .

---

## Developer Notes

This function (int 0x21, function 0x4B) is still under development.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x4C: End Process

Terminates a process

### Parameters

AH  
0x4C

### Comments

The memory used by the process is released, with following exception. This function will not free system memory **allocated** by a task that was created within a program. Only memory allocated by the program's main task will be freed here.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x4E: Find First File

Find first file matching file name specification and attribute. The results are stored in the Disk Transfer Area (DTA).

### Parameters

AH  
0x4E

CX  
File attribute

*DS:DX*

Null terminated file specification (may include path and wildcards)

## Return Value

Success: Carry flag cleared, search results are in DTA

Failure: Carry flag set

## Comments

The main task of your application has a standard pointer to a Disk Transfer Area of the program. This DTA address should be set with interrupt 0x21 function **0x1A** before calling this `findfirst` function.

The `findfirst`/`findnext` sequence is handled by your compiler library so the Disk Transfer Area is therefore not described here.

Tasks created with the RTOS API inside of your application use a DTA from an internal list. A task which gets **access** to the file system reserves an entry in this list.

Only ten DTA entries for user tasks are available.

## Related Topics

Find **next** file

RTOS **RTX\_ACCESS\_FILESYSTEM** Service

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x4F: Find Next File

Finds the next file in the `findfirst`/`findnext` sequence. The results are stored in the Disk Transfer Area (DTA).

The task that calls the `findnext` must be the same task that called the `findfirst`.

## Parameters

*AH*

0x4F

## Return Value

Success: Carry flag cleared, search results are in DTA

Failure: Carry flag set

## Comments

Since the entire state of a `findfirst`/`findnext` sequence is held in the DTA data block, other disk operations such as renaming, moving, deleting, or creating files can cause inaccurate directory searches

such as finding the same file twice. Please look at the findfirst function (Interrupt 0x21, function [0x4E](#) ) for other restrictions.

Important restriction:

Findnext must be called in a loop until it failed.(until the last file is found)

## Related Topics

Find [first](#) file

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x50: Debugger Support

Get address of internal program task list and set callback vector used for program startup notification.

### Parameters

*AH*

0x50

*DX:BX*

Callback vector

*CX*

Sanity Check = 0x8765

### Return Value

DX:BX contains pointer to internal task list.

AX = Task list length = 12

CX = size of task list elements

SI = RTOS Private Data Segment

### Comments

This function is intended only for debugger usage.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x51: Get PSP Segment Address

Get PSP (Program Segment Prefix) segment address

## Parameters

*AH*  
0x51

## Return Value

BX contains the PSP segment address, if BX=0, PSP was not found

## Comments

This function is identical to interrupt 0x21 service [0x62](#) .

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x56: Change Directory Entry, Rename File

Rename a file by changing its directory entry

## Parameters

*AH*  
0x56

*DS:DX*  
Pointer to null terminated old file name

*ES:DI*  
Pointer to null terminated new file name

## Return Value

Success: Carry flag cleared  
Failure: Carry flag set, AX holds error code  
AX: 1 - File not found  
AX: 4 - File new file name already exists  
AX: 5 - Internal error  
AX: 7 - Directory update failed

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x57: Get/Set File Date and Time

Get or set file date and time. The file is specified by file handle.

## Parameters

*AH*

0x57

*AL*

AL=0 to get the date and time of the last modification.  
AL=1 to set the file date and time.

*BX*

File handle

*CX*

if AL==1: Time in the format described below

*DX*

if AL==1: Date in the format described below

## Return Value

Success: Carry flag cleared, if input parameter AL=0 then:

CX = time of last modification

DX = date of last modification

Failure: Carry flag set

## Comments

The time/date registers are coded as follows:

CX time of last modification

bits 15-11: hours (0..23)

bits 10-5: minutes

bits 4-0: seconds/2

DX date of last modification

bits 15-9: year-1980

bits 8-5: month

bits 4-0: day

## Related Topics

[Create](#) New File Handle

[Open](#) Existing File

[Top of list](#)

[Index page](#)

Get/Set memory strategy, only a dummy function for compatibility

## Parameters

<i>AH</i>	0x58
<i>AL</i>	AL=0 Get strategy AL=1 Set strategy.
<i>BX</i>	(AL=1) Strategy 0: First fit, 1: Best fit, 2: Last fit

## Return Value

If parameter AL==0 (get strategy) AX contains as a dummy value the memory strategy, Carry flag cleared  
If parameter AL==1 (set strategy) AX contains always the parameter given at BX. If BX contains a value bigger 2, carry flag is set and AX == 1.

## Comments

This is only a dummy function, added for compatibility. The @Chip-RTOS has its own memory strategy. The @Chip-RTOS memory always allocates memory in the following way:  
DOS programs are always loaded at the first lowest free memory block of the @Chip-RTOS memory area. For memory blocks allocated inside of the @Chip-RTOS or e.g. with [Int21h 0x48](#) the @Chip-RTOS always start searching for a free memory block from the highest possible RAM address. So the largest free memory block of the system is always located in the middle of the @Chip-RTOS memory area. The shell command [mem](#) shows the state of the internal memory map.

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x62: Get PSP Segment Address

Get PSP (Program Segment Prefix) segment address

## Parameters

<i>AH</i>	0x62
-----------	------

## Return Value

BX contains the PSP segment address, if BX=0, PSP was not found

## Comments

This function is identical to interrupt 0x21 service [0x51](#) .

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x63: Get Leading Byte (stub)

Dummy function, not supported by the SC12 BIOS

### Parameters

*AH*  
0x63

### Return Value

Always returns with Carry flag set  
AL = 0xFF  
DS = 0  
SI = 0

[Top of list](#)

[Index page](#)

---

## Interrupt 0x21 service 0x68: Flush DOS Buffers to Disk

Flush DOS file buffers to disk for specified file.

### Parameters

*AH*  
0x68

*BX*  
file handle

### Return Value

Success: Carry flag cleared  
Failure: Carry flag set, AX holds error code  
AX : 2 Invalid handle  
AX : 7 I/O error occurred

[Top of list](#)

[Index page](#)

---

End of document

---

## DOS API Updates - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### DOS API News

The following extensions to the DOS [API](#) are available in the indicated BIOS revisions.

New in version 1.02b: [Get/Set memory strategy \(dummy function\)](#)

New in version 1.01b: [Bugfix at Get Disk Free Space](#)

New in version 1.00: [Add stdin channel return value \(see documentation\)](#)

New in version 1.00: [Modified: Read from stdin, add stdin channel return value](#)

---

End of document

---

## External Disk Interface - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### External Disk Drive

Here is the interface definition for an external disk B: drive . This interface allows you to add an external B: drive. This drive must be block (sector) oriented. Each sector should be 512 bytes long. The application must provide a software interrupt 0xB1 function to read and write these sectors on this drive.

Maximum disk size is about 2 Gigabytes.

- [Interrupt 0xB0 function 0x01: Install External Disk](#)
- [Interrupt 0xB0 function 0x02: Deinstall External Disk](#)

---

### Interrupt 0xB0 service 0x01: Install External Disk

This function will logically install a B: drive.

Prior to this call the application must provide a disk read/write function **installed** at interrupt 0xB1. This installed handler function should expect:

AX	1 for write, 0 for read.
BX,DX	Unsigned long (BX holds high word) of sector number
CX	Number of sectors to read/write
ES:DI	Segment:Offset of memory area to read/write

The handler should return 0 in AX if OK.

### Parameters

*AH*  
0x01

*BX,DX*  
An unsigned long with the total number of sectors. BX holds high word.

### Return Value

AX is 0 if OK.

## Interrupt 0xB0 service 0x02: Deinstall External Disk

Used to deinstall drive B: which was installed before with function 0x01 (above).

### Parameters

*AH*  
0x02

### Return Value

AX =0 success  
AX !=0 failed (may be drive was not installed?)

## Ethernet Packet Driver Interface - BIOS V1.02 Beta

IPC@Chip Documentation [Index](#)

### Ethernet: Packet Driver Interface

Packet driver interface definition for direct access to the SC12 Ethernet device

The SC12 BIOS uses interrupt 0xAE with a service number in the high order byte of the AX register (AH) to access the IPC@Chip Ethernet packet driver. All supported services are listed here.

New in version 1.02B: [GET\\_RCV\\_MODE, Get recv mode of the ethernet device](#)

New in version 1.02B: [SET\\_RCV\\_MODE, Set receive mode of IPC@Chip ethernet device](#)

New in version 1.02B: [SET\\_MULTICAST, Set/add ethernet multicast address](#)

New in version 1.02B: [DET\\_MULTICAST, remove ethernet multicast address](#)

New in version 1.02B: [INSTALL\\_WILDCARD, Install Access wildcard handler](#)

- [Interrupt 0xAE function 0x01: DRIVER\\_INFO, Get Driver Information](#)
- [Interrupt 0xAE function 0x02: ACCESS\\_TYPE, Install Access Handler](#)
- [Interrupt 0xAE function 0x03: RELEASE\\_TYPE, Unload an Access Handler](#)
- [Interrupt 0xAE function 0x04: SEND\\_PKT, Send an Ethernet Packet](#)
- [Interrupt 0xAE function 0x06: GET\\_ADDRESS, Get the SC12 Ethernet Address](#)
- [Interrupt 0xAE function 0x14: SET\\_RCV\\_MODE, Set receive mode of IPC@Chip ethernet device](#)
- [Interrupt 0xAE function 0x15: GET\\_RCV\\_MODE, Get recv mode of the ethernet device](#)
- [Interrupt 0xAE function 0x16: SET\\_MULTICAST, Set Ethernet multicast address](#)
- [Interrupt 0xAE function 0x27: DEL\\_MULTICAST, Remove ethernet multicast address](#)
- [Interrupt 0xAE function 0x28: INSTALL\\_WILDCARD, Install access wildcard access handler](#)

On return the CPU carry flag is set if an error has occurred. The DH register holds an error code:

```

0x00    NO_ERROR
0x01    BAD_HANDLE        // invalid handle number
0x05    BAD_TYPE          // Bad packet type specified
0x09    NO_SPACE          // operation failed because of insufficient space
0x0A    TYPE_INUSE        // the type had previously been accessed and not
released
0x0B    BAD_COMMAND       // the command was out of range, or not implemented
0x0C    CANT_SEND         // the packet couldn't be sent (usually a hardware

```

error)

## Important:

The implemented services are only a small subset of the packet driver interface definition. One important difference is that the software interrupt 0xAE does not have the typical PKTDRVR signature at offset 3 in the interrupt source code. So instead of looking for this signature, use BIOS interrupt 0xA0 [function](#) 0x16 to check if the packet driver interface is available in the IPC@Chip BIOS. The maximum number of installed packet handlers is five.

Examples for the usage of the packet interface:

1. PKTDRV.C: C-Source for the API calls
2. PKTDRV.H: Defines, typedefs, prototypes, ... for the API calls
3. REC.ASM : Assembler example for a receiver handler function, installed with function ACCESS\_TYPE
4. SEND.EXE and RCV.EXE (with source): Simple program pair for sending and receiving of Ethernet packets

---

## Interrupt 0xAE service 0x01: DRIVER\_INFO, Get Driver Information

Added only for compatibility

### Parameters

AH  
0x01

### Return Value

Carry flag: 0, Success:

AL: 1, Basic functions present

BX: 0x0B, Version

CH: 0x01, Class

CL: 0x00, Number

DL: 0x36, Type

DH: 0x00, Error code = NO\_ERROR

DS:SI : Pointer to the null terminated driver name string, "SC12PKT"

[Top of list](#)

[Index page](#)

---

## Interrupt 0xAE service 0x02: ACCESS\_TYPE, Install Access Handler

### Parameters

AH

0x02

*AL*

class, from DRIVER\_INFO [call](#)

*CX*

type length, must be 2!!

*DL*

number, from DRIVER\_INFO [call](#)

*DS:SI*

Pointer to the desired packet type, e.g. 0x800 for IP  
(The object pointed to here must be persistent, not a momentary value.)

*ES:DI*

Vector to user's receiver handler function for this packet type

## Return Value

Carry flag: 0 Success, AX contains the handle number (needed for RELEASE\_TYPE [call](#) )  
Carry flag :1 Failure, DH contains error code

## Comments

We support the following Ethernet V2.0 frame format:

48 Bits (6 Bytes) Destination address

48 Bits (6 Bytes) Source address

16 Bits (2 Bytes) Type field e.g 0x0608 for ARP, 0x0008 for IP or user defined types

46 to 1500 Bytes of user data.

The maximum number of installed handlers is limited to five. In all BIOS versions except the TINY version, setting of ARP or IP handlers is not allowed here.

When an Ethernet packet of the type specified here at [DS:SI] is received by the the SC12's network device driver, this driver will perform callbacks to the receiver handler function specified here in ES:DI. This callback will be done twice:

- First call:

Input parameters to your handler: AX = 0, CX = received packet length

Return Value from your handler: ES:DI - Pointer to buffer where driver can load the CX received bytes.

- Second call:

Input parameters to your handler: AX = 1, data ready in your buffer now

Return Value from your handler: -- none --

On the first call, your handler produces a buffer into which the driver can transfer the received packet. This byte transfer occurs between the two calls to your handler function.

Important: Because of our Little Endian processor you must exchange the Bytes for the packet type e.g. use 0x0008 for the IP type instead of 0x0800.

## Interrupt 0xAE service 0x03: RELEASE\_TYPE, Unload an Access Handler

The user installed Ethernet packet type access handler is removed.

### Parameters

*AH*  
0x03

*BX*  
Handle from ACCESS\_TYPE [call](#) 0x02

### Return Value

Carry: 0, AX:0, DX:0: success  
Carry: 1, DH contains error code

## Interrupt 0xAE service 0x04: SEND\_PKT, Send an Ethernet Packet

Send bytes in provided packet buffer over Ethernet.

### Parameters

*AH*  
0x04

*CX*  
Length of packet

*DS:SI*  
Pointer to packet buffer

### Return Value

Carry flag:0, AX:0, DX:0: success  
Carry flag:1, DH contains error code

## Interrupt 0xAE service 0x06: GET\_ADDRESS, Get the SC12 Ethernet Address

### Parameters

*AH*  
0x06

*ES:DI*  
Pointer to user buffer (6 bytes), for storing the Ethernet address

### Return Value

Carry flag:0, AX:0, DX:0: success  
Location at [ES:DI] contains the six bytes of the Ethernet address.

[Top of list](#)  
[Index page](#)

---

## Interrupt 0xAE service 0x14: SET\_RCV\_MODE, Set receive mode of IPC@Chip ethernet device

### Parameters

*AH*  
0x14

*CX*  
Receive mode  
3: Receiving packets mit own ethernet address and broadcasts

### Return Value

Carry flag:0, AX:0, DX:0: success  
Carry flag:1, AX:-1;DX:-1 Bad parameter at CX

### Comments

Default receive mode is 3.  
Receive mode 6 should only be used at the SC12 Tiny version (@Chip-RTOS without the TCPIP stack).  
Calling [SET\\_MULTICAST](#) overrides the current mode with mode 5.  
API call [Get receive mode](#) returns the current mode.

---

### Developer Notes

This mode is still under development, please contact us at our internet newsgroup or direct by email, if you have questions, problems or suggestions.

## Interrupt 0xAE service 0x15: GET\_RCV\_MODE, Get recv mode of the ethernet device

### Parameters

*AH*  
0x15

### Return Value

Carry flag:0, AX:3 accept any packet with own address or broadcast address  
AX:5 accept any packet with own address, broadcast address or multicast addresses  
Carry flag:1, DH contains error code

## Interrupt 0xAE service 0x16: SET\_MULTICAST, Set Ethernet multicast address

### Parameters

*AH*  
0x16

*ES:DI*  
Pointer to multicast mac address buffer 6 bytes

*CX*  
length of ES:DI buffer, must be 6

### Return Value

Carry flag:0, AX:0, DX:0: success  
Carry flag:1, DH contains error code

---

## Developer Notes

The address will be installed inside the ethernet device of the IPC@Chip. Max. 64 addresses could be installed.

## Interrupt 0xAE service 0x27: DEL\_MULTICAST, Remove ethernet multicast address

### Parameters

- AH*  
0x27
- ES:DI*  
Pointer to multicast mac address buffer 6 bytes
- CX*  
length of ES:DI buffer, must be 6

### Return Value

Carry flag:0, AX:0, DX:0: success  
Carry flag:1, DH contains error code

## Interrupt 0xAE service 0x28: INSTALL\_WILDCARD, Install access wildcard access handler

Installing a wildcard packettype access handler.  
Any incoming ethernet packet will be accepted, if one or more bits of the packettype matches to the bits of the installed wildcard packettype.

### Parameters

- AH*  
0x28
- AL*  
class, from DRIVER\_INFO [call](#)
- CX*  
type length, must be 2!!
- DL*  
number, from DRIVER\_INFO [call](#)
- DS:SI*

Pointer to the desired packet type, e.g. 0xFFFF for accepting any incoming ethernet packet  
(The object pointed to here must be persistent, not a momentary value.)

*ES:DI*

Vector to user's receiver handler function for this packet type

## Return Value

Carry flag: 0 Success, AX contains the handle number [call](#) )

Carry flag :1 Failure, DH contains error code

## Comments

Only one wildcard could be installed. Installing a new type overwrites the previous.

If a packettype of 0xFFFF is installed, any incoming packet will be accepted.

Delete a wild card handle by installing a wildcard with the packet type 0x0000.

"Normal" handlers installed with [call](#) are not overwritten by installing a wildcard

[Top of list](#)

[Index page](#)

---

End of document

---

## TFTP server - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### Introduction

Here is a short description of the SC12 TFTP server.

The TFTP server is available starting with BIOS version SC12V0100 and allows only sending and receiving of files to a remote TFTP client.

A SC12 BIOS version with TFTP could be used as a simple alternative to FTP. TFTP doesn't provide extended filesystem features like listing directories or deleting files. If the user don't need the extended features of the FTP server, it could be useful to use a SC12 BIOS with TFTP instead of FTP. This saves 13 KByte of flash memory.

The TFTP server is not a part of our current official 6 BIOS versions. You must order direct a BIOS version, which includes this feature.

By default the server listens at the standard TFTP port 69 for incoming TFTP client requests. The TFTP port could configure an own port number at chip.ini.

- See TFTP configuration [Set TFTP port number](#)

The server is able to serve one client at one time. Only the binary (octet) transfer mode is supported. The size of TFTP data packets must be 512 Bytes. By default TFTP file transfers are not allowed, for avoiding a security leak. You can enable/disable TFTP with the command TFTP 0/1.

- See TFTP command [Enable/Disable TFTP](#) : Allow/Disallow the usage of TFTP

### Comments

It is also possible to execute this command inside of an user application with

- See [Execute a shell command](#)



## Security notes - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### Introduction

Here some notes how to protect the IPC@Chip against unauthorized access. By default most of the further described security features are disabled. This to enable every starting user to have access to them from the start for his IPC@Chip development.

- [WEB server](#)
  - [TELNET server](#)
  - [FTP server](#)
  - [PPP server](#)
  - [Chiptool UDP config server](#)
  - [TFTP server](#)
  - [General Security](#)
- 

### WEB server

Steps to protect the IPC@Chip against unauthorized access via HTTP:

1. [Webserver default drive](#) : Set a webserver root drive at chip.ini
2. [Webserver root directory](#) : Set a webserver root directory at chip.ini
3. [Remove CGI page ChipCfg](#) : Delete this preconfigured page from the CGI table with the CGI API
4. [PUT Method User and Password](#) : Define User and Password if you have a BIOS Variant which provides the HTTP PUT method. Otherwise everyone can transfer files to your server with the standard password and user 'WEB'. The HTTP PUT method wont be provided by the standard BIOS Variants LARGE, MEDIUM, SMALL, TINY.

[Top of list](#)  
[Index page](#)

---

### TELNET server

Steps to protect the IPC@Chip against unauthorized access via Telnet:

1. [Telnet timeout minutes](#) : Define telnet idle timeout minutes at chip.ini
2. [Telnet login delay](#) : Enable telnet login delay at chip.ini
3. [Telnet login retries](#) : Set telnet login retries at chip.ini
4. [Telnet user and passwords](#) : Define both user and password names at chip.ini
5. [Set the Stdio Focuskey to zero](#) : at chip.ini or inside of the application with [Set Stdio focus key](#)  
This disables the switching of stdio.

## Comments

Since BIOS version 1.01B telnet doesn't tell if the input of the username or the password input was wrong.

[Top of list](#)

[Index page](#)

---

## FTP server

Steps to protect the IPC@Chip against unauthorized access via FTP:

1. [FTP timeout](#) : Define FTP idle timeout seconds at chip.ini
2. [FTP login delay](#) : Enable FTP login delay at chip.ini
3. [FTP user and passwords](#) : You should define both user and password names at chip.ini
4. [FTP user root directory](#) : For a "normal" user you should define a root directory above "\".
5. [FTP user drive](#) : If you specify a rootdirectory you also must set a drive.
6. [Hide files with int21h 0x43](#) : Hidden files are not visible at FTP sessions or by the DIR command

## Comments

Since BIOS version 1.01B FTP doesn't tell if the username input or the password input was wrong.

[Top of list](#)

[Index page](#)

---

## PPP server

Steps to protect the IPC@Chip against unauthorized access via PPP:

1. [PPP server idle timeout](#) : Define PPP server idle timeout seconds at chip.ini
2. [PPP users and passwords](#) : Define both user and password names for the the PPP server at chip.ini

[Top of list](#)

[Index page](#)

---

## Chiptool UDP config server

Protect the IPC@Chip against unauthorized access by using the Chiptool program:

1. [UDP config server](#) : Set the UDP config server security level at chip.ini

[Top of list](#)

[Index page](#)

---

## TFTP server

Protect the IPC@Chip against unauthorized access via TFTP:

1. [Disable/enable TFTP](#) : Disable/enable TFTP with shell command

[Top of list](#)

[Index page](#)

---

## General Security

1. [Install System Server Connection Handlers](#) gives the possibility to generate IP- and/or Port - filters and forbid connections to FTP, WEB or Telnet
2. With the BIOS Ints Api call [Suspend System Servers](#) you can Suspend/Resume the FTP, Web and Telnet Server while runtime.

[Top of list](#)

[Index page](#)

---

End of document

---

## Programming notes - BIOS V1.02 Beta

---

IPC@Chip Documentation [Index](#)

---

### Introduction

Here are some useful notes for programming applications for the IPC@Chip. These notes contains some general rules for programming DOS applications for the IPC@CHIP and should prevent the programmers from fatal errors.

- [Common notes for building IPC@Chip user applications](#)
  - [Using RTOS API](#)
  - [Programming CGI functions](#)
  - [Configure the FTP server](#)
  - [Usage of the TCPIP API](#)
  - [General notes for the usage of the DOS and BIOS int API calls](#)
  - [Using Hardware API](#)
  - [Using Fossil API](#)
  - [Working with Float Data Types](#)
  - [Configure PPP client or server](#)
- 

### Common notes for building IPC@Chip user applications

1. Compiler option settings:  
We recommend the usage of large memory model for user applications  
The compiler must produce 186 processor instruction code  
The data alignment must be set to 8 Bit (Byte alignment), not to 16-Bit (Word alignment)!
2. No "exception handling" at Borland C 4.x or 5.x projects:  
Users of Borland C-Compilers (which provides "exception handling libraries" e.g. it's the default setting at 4.5 or 5.02) should choose for their program project "No exceptions" at the "Target Expert" window. This saves Flash and RAM memory space.
3. Turbo Pascal programmers should include the example file `SC12.INC` (part of each IPC@Chip Pascal example program). At the start of their program, they should install the procedure `Terminate_Program` as the default exit procedure of the program. Usage of the standard `unit crt` is not allowed.

4. Before starting the first Turbo Pascal program at the IPC@Chip the [memopt 1](#) command must be executed (e.g. in the autoexec.bat)
5. If programs that depend on one another are started from a batch file (e.g. if a user program needs a previous driver program) the [Batchmode 1](#) in chip.ini should be set.
6. The CLIB library API calls are using the standard int86 and int86x calls e.g. from the Borland C standard libs. We used these functions for more readable code. If developers need higher performance in their applications, it will be better to implement often used API calls with inline assembler code. With inline assembler instructions it is possible to load the processor registers directly with the needed parameters and directly invoke the appropriate software interrupt `int xxh`.
7. For some reasons it is not advisable to design an IPC@Chip application as a set of several DOS programs:
  1. It is more meaningful to run several tasks (builded with the RTOS API) inside of one DOS program because this requires much less RAM and Flash memory space, than running several DOS programs as tasks of the @chip-RTOS.
  2. It could lead to high memory fragmentation and insufficient memory, if more than one or two DOS programs are running at the IPC@Chip, which will be often terminated and restarted again.
8. Most of the used pointer variables and function parameters at the CLIB library C files are not declared as FAR pointer, because all of our testprograms are using the memory model Large (at model Large all pointers become automatically FAR pointers). If you want to use e.g. memory model Small for your application you must rewrite the CLIB source files by declaring all pointer variables and function parameters explicit as FAR pointers.
9. It is not advisable to use the Borlandc-CLIB call malloc at memory model Large inside of your application. In that case the DOS program tries to increase its own memory block with `int21h 0x4A`. If another program is loaded after that program the malloc call failed, because there is no memory space left between the two programs. It is possible to define a memory gap between two loaded programs at [chip.ini](#) but then you must know how max. memory is required at your malloc calls. It is better to reserve inside of your application the memory by declaring an unsigned char array (or the type you require) with the needed memory size.
10. The @Chip-RTOS has its own memory strategy.

The @Chip-RTOS memory always allocates memory in the following way:  
DOS programs are always loaded at the first lowest free memory block. For memory blocks allocated inside of the @Chip-RTOS or e.g. with [DOS service 0x48](#) the @Chip-RTOS always start searching for a free memory block from the highest possible RAM address. So the largest free memory block of the system is always located in the middle of the @Chip-RTOS memory area. The shell command [mem](#) shows the state of the internal memory map.

[Top of list](#)  
[Index page](#)

---

## Using RTOS API

1. Timer procedures must be implemented as short as possible, because they are executed in the Kernel task. Large timer procedures are blocking the kernel and other tasks of the @Chip-RTOS. Do not use large Clib functions like printf inside of a timer procedure. This could cause a fatal stack overflow in the kernel task (stack size 1024 Bytes)

2. Declaring of timer procedures with Borland C:

```
void huge my_timer(void)
```

3. Microsoft C:

```
void far _saveregs _loadds my_timer(void)
```

4. Turbo Pascal:

```
procedure Timer1_Proc;interrupt;
begin

  [... your code ...]

  (*****
  (* This is needed at the end of the Timer Proc. *)
  asm
    POP BP
    POP ES
    POP DS
    POP DI
    POP SI
    POP DX
    POP CX
    POP BX
    POP AX
    RETF
  end;
  (*****
end;
```

5. DOS applications are running as a task of the @CHIP-RTOS with a default priority of 25. If the users application doesn't go to **sleep** , lower priority tasks like the FTP server or the webserver will not work. In major loops within user applications, the programmer should insert **sleep calls** if the FTP server or webserver should work during the runtime of the user application.
6. The stack of a task **created** inside of a DOS application should have a minimum stack size of 1024 Bytes. Programmers of task functions who are using Microsoft C-Compilers with C-Library functions, e.g. `sprintf` , which requires a lot of stack space should increase this allocation to 6144 (6 Kbytes). More stack space for the task is also required if your task function uses a large amount of stack for automatic data (local variables) declared inside the task function call.
7. Declaring of task functions with Borland C:  

```
void huge my_task(void)
```
8. Declaring of task functions with Microsoft C:  

```
void far _saveregs _loadds my_task(void)
```
9. Before exiting an application, every task or timer procedure created inside of this application program must be removed.
10. A **sleep call** with parameter 1 millisecond takes equal or less than one millisecond. If a user needs a minimal sleep time of 1 millisecond then `RTX_SLEEP_TIME` must be called with value 2 milliseconds.

## Programming CGI functions

1. Avoid large loops inside of CGI functions. CGI functions must be executed as fast as possible. Large execution times in CGI functions block the webserver task preventing response to other http requests.
2. Declaring of CGI functions with Borland C:  

```
void huge my_cgi_func(rpCgiPtr CgiRequest)
```
3. Declaring of CGI functions with Microsoft C:  

```
void far _saveregs _loadds my_cgi_func(rpCgiPtr CgiRequest)
```
4. Declaring of CGI functions with Borland Turbo Pascal:  

```
procedure my_cgi_func;interrupt;
```
5. Turbo Pascal users must install their CGI functions with API call [Install CGI Pascal function](#)
6. C-Programmers must use [Install CGI function](#)
7. Avoid the declaration of large arrays as local variables inside of the CGI function to prevent stack overflows
8. Users of Microsoft C should set [Webserver stacksize](#) in chip.ini up to 6144 KBytes, to prevent stack overflows, if Microsoft CLib functions like sprintf are used.
9. Dynamic html or text pages, which are created inside of a user CGI function should be as small as possible. The webserver inside must allocate a tempbuffer for storing this page before sending it to the browser. If your application builds large dynamic html page in RAM, your application should not use all available memory in the IPC@Chip because the webserver will need some memory for allocating tempbuffers for this page. How much memory should be left: This depends on the application and the sizes of the created dynamic html or CGI pages.
10. Before exiting an application, every CGI function installed by the application must be removed with [Remove CGI page](#)

## Configure the FTP server

1. The default FTP idletimeout is set to 300 seconds. This is a very long time for waiting, if FTP commands fail. The [idletimeout](#) can be reduced in chip.ini.

---

## Usage of the TCPIP API

1. Processing of a socket callback functions (see [Register callback](#) ) should be kept at a minimum to prevent stack overflows.
2. Declaring of socket callback functions with Borland C:  

```
void huge my_callback(int socketdescriptor, int eventflagmask)
```
3. Declaring of socket callback functions with Microsoft C:  

```
void far _saveregs _loadds my_callback(int sd, int eventmask)
```
4. The internal TCPIP stack of the IPC@Chip allocates memory for buffers smaller than 4096 bytes from a preallocated memory block. Larger buffers are allocated direct from the Chip-RTOS. If user TCPIP network communication sends/receives packets with a size larger than 4096 bytes, the user application should not use all available memory in the IPC@Chip, because of these additional allocations. There should be in that case always a minimum of 30-40 KBytes of free available memory at the IPC@Chip. The [mem command](#) shows the whole memory list of the IPC@Chip at runtime. The maximum amount of TCPIP memory could be configured in chip.ini (see [Set TCPIP memory size](#) ). The application programmer could reduce or increase this size in chip.ini. With the API call [Get TCPIP memory info](#) it is possible to control the TCPIP memory usage at the application runtime.

[Top of list](#)  
[Index page](#)

---

## General notes for the usage of the DOS and BIOS int API calls

1. At the start of a user program the Stdio focus should be set to USER. Before ending the application switch the focus back to SHELL or BOTH (see [Set Stdio focus](#) ).
2. If more than one user program runs in the IPC@Chip, only one of them should read characters from [Stdin](#)
3. The functionality of most of the shell commands is also available through calls into the [BIOS Int API](#) . If not, use the BIOS int call [Execute a shell command](#) . This call executes a shell command from inside the user application.
4. Install a fatal user error handler, which does a reboot of the IPC@Chip with [Install user fatal error handler](#)
5. Used software interrupts (all others are free for use):
  - 0x00 - Reserved (BIOS Divide Overflow Handler)
  - 0x01 - Reserved (Debugger Trace Interrupt)
  - 0x02 - Reserved (Hardware Non-Maskable Interrupt (NMI))
  - 0x03 - Reserved (Debugger Breakpoint Interrupt)
  - 0x04 - Reserved (BIOS INTO Overflow Handler)

- 0x05 - Reserved (BIOS Array Bounds Exception Handler)
- 0x06 - Reserved (BIOS Invalid Opcode Exception Handler)
- 0x07 - Reserved (BIOS ESC Opcode Exception Handler)
- 0x08 - Reserved (Hardware, Timer #0 Handler)
- 0x0A - Reserved (Hardware, DMA #0 / INT5 Handler)
- 0x0B - Reserved (Hardware, DMA #1 / INT6 Handler)
- 0x0C - Reserved (Hardware, INT0 Handler)
- 0x0D - Reserved (Hardware, INT1 Ethernet Handler)
- 0x0E - Reserved (Hardware, INT2 Handler)
- 0x0F - Reserved (Hardware, INT3 Handler)
- 0x10 - [Biosint](#)
- 0x11 - [Biosint](#)
- 0x12 - Reserved (Hardware, Timer #1 Handler)
- 0x13 - Reserved (Hardware, Timer #2 Handler)
- 0x14 - [Fossil Interface](#)
- 0x16 - [Biosint](#)
- 0x1A - [Biosint](#)
- 0x1C - Timer Interrupt, see [Set timer 1C interval](#)
- 0x20 - Terminate Program (Only for compatibility, instead use [DOS service 0x4C](#) )
- 0x21 - [DOSEmu](#) Interrupt Interface
- 0xA0 - [Several 'chip' related services](#)
- 0xA1 - [Hardware API](#) (HAL)
- 0xA2 - [Hardware API](#) (PFE)
- 0xAA - [I2C Interface](#)
- 0xAB - [CGI Interface](#)
- 0xAC - [TCP/IP API](#)
- 0xAD - [RTOS API](#)
- 0xAE - [Ethernet Packet Driver](#)
- 0xAF - Timer Interrupt, see [Set timer AF interval](#)
- 0xB0 - [External Disk API](#)
- 0xB1 - [External Disk Driver](#)
- 0xBF - This vector is reserved to start a DOS executable

If you are using a BIOS variant in which some modules are not included, then the interrupts corresponding to these modules are free for use.

6. External hardware interrupts are enabled (STI opcode) during execution of the following API software interrupts:  
DOS ints 0x10, 0x14(Fossil), 0x16, 0x20, 0x21, BIOS int 0xA0, CGI int 0xAB, I2C int 0xAA, CGI int 0xAB, TCP/IP int 0xAC, RTOS int 0xAD Pkt int 0xAE, Extdisk int 0xB0, Extdisk user int 0xB1

1. The HAL functions keep interrupts disabled, so you can call them inside an interrupt routine. The PFE functions are only for choosing and initializing a specific function on the selected pin. They should be called once in your application for initializing your hardware environment and not at runtime or inside interrupt routines.
2. Do not use functions of the RTOS API inside of a user isr installed with [Install ISR](#)
3. The latency time of the user ISR (from generation of an interrupt until first line of code inside the user ISR) is about 65 µs.
4. Instead of HAL functions [Read data bus](#) and [Write data bus](#) you can call C-functions `inportb` and `outportb` from DOS.H for faster data bus accesses.

[Top of list](#)  
[Index page](#)

---

## Using Fossil API

1. The [receive and send queue size](#) can be configured over the CHIP.INI.
2. If you want external DMA, you have to disable the [serial DMA mode](#). Otherwise the DMA mode is recommended.
3. Since BIOS 1.02B XON/XOFF mode is available, if the serial DMA mode is set.  
Please note: Because of the internal functionality of DMA it is not possible to detect an XON or XOFF character from the connected peer immediately. It is possible that an overrun situation at the peer (e.g. GSM modem) could occur. Nevertheless we enable this mode because some GSM modems (any??) supports only XON/XOFF as serial flow control mechanism.
4. The default serial recv queue size is 1024 Byte. If the default DMA receive mode is used, it is advised to increase the recv queue size at [Chip.ini](#) up to a minimum value of 2048 Bytes to prevent a possible buffer overrun (even if hard handshake is used). This could only happen with the default queue size of 1024 Bytes, if the user doesn't call the [Fossil API readblock function](#) fast enough. If the application programmer will not increase this buffer size up to the recommended value, he should call the [Fossil API readblock function](#) with the highest possible size at the CX-Register for flushing the internal buffers and preventing an receive buffer overrun
5. For a given serial port the fossil functions are not reentrant. Do not call fossil functions for the same serial port from different tasks. However for different serial ports, the fossil functions are reentrant. E.g. task A can operate the COM port using fossil functions concurrently with task B operating the EXT port using the same fossil functions.

[Top of list](#)  
[Index page](#)

---

## Working with Float Data Types

1. The IPC@CHIP does not provide a floating point co-processor. So if you want to use floating point data types you need to enable the math-emulation in your compiler. In Borland C++ 5.02 see the option "Emulation" under the Target Expert's (right mouse click on your Exe-file in your project) "Math Support".
2. If you want to use float data types in tasks other than your main (DOS) task, you have to reinite the floating point emulation. Using the Borland C++ 5.02 compiler you do that with the command "\_fpreset(void)" (available in float.h). Do that in your task procedure before using floating point data types.

[Top of list](#)  
[Index page](#)

---

## Configure PPP client or server

1. Connected modems should be configured in chip.ini with the modem command ATE0 **INITCMD** . This prevents the modem from echoing characters to the peer in command mode. The COM and the serial ports of the IPC@Chip SC12 have only 4 lines (TxD/RxD/CTS/RTS) and no line for detecting a hang-up of the modem. Because of this fact we provided the **idletimeout** and the **MODEMCTRL** configuration features in chip.ini or in the PPP client init struct **type** . But the idletimeout and modemctrl detection could fail if a modem has switched its echo mode on. If the peer modem hangs up without correctly closing a PPP session, the IPC@chip modem also hangs up and goes into the command mode. Because of the missing line for detecting a modem hang-up, the PPP server doesn't know anything about the broken connection and still sends PPP frames to the modem. It could happen that the modem echoes these characters back to the IPC@Chip due to being in "Echo on" mode. This will cause the idletimeout to not work.
2. We recommend that the PPP server or client and the connected modems should run (if possible) with RTS/CTS flow control (see chip.ini **flow control mode** or in the PPP client init struct **type** ). Most modems use RTS/CTS flow control, if they get the AT command AT\Q3.
3. The COM and EXT port of the SC12 has only CTS, RTS, RxD and TxD lines, so you have to configure your modem with DTR always on. (e.g. AT cmd for a most modem types: AT&D0)

[Top of list](#)  
[Index page](#)

---

End of document

---